國立清華大學

系統晶片設計

SOC Design

國立清華大學

NATIONAL TSING HUA UNIVERSITY

Lab 2

系所級:電子所二年級

學號:111063548

姓名:蕭方凱

指導老師:賴瑾教授

目錄

I. FIR with Interface AXI-Master

1. HLS/IP Design

(1) FIR.h

```
#ifndef __FIR_H__
#define __FIR_H__

#include <ap_axi_sdata.h>

#define N                       11

#define MAP_ALIGN_4INT          (((N + 3) >> 2) << 2)

typedef ap_uint<32> reg32_t;
typedef signed int int32_t;
typedef unsigned int uint32_t;

void fir_n11_maxi(volatile int32_t* pn32HPInput, volatile int32_t* pn32HPOutput, int32_t an32Coef[MAP_ALIGN_4INT], reg32_t regXferLeng);

#endif
```

ifndef&define 上一個 lab 已經使用過，故不再贅述。這個標頭檔首先定義了常數 N 為 11，接著〝typedef ap_uint<32> reg32_t;〞代表定義一個無符號數 reg32_t，其位元長度為 32，這種定義方法是 HLS 提供給用戶進行任意長度的數據運算，範圍可從 1 至 1024。至於 MAP_ALIGN_4INT 的部分，是控制 an32coef 為大於 11 同時又是 4 的倍數，方便 FIR 運算。

(2) FIR.cpp

```
#include "FIR.h"

void fir_n11_maxi(volatile int32_t* pn32HPInput, volatile int32_t* pn32HPOutput, int32_t an32Coef[MAP_ALIGN_4INT], reg32_t regXferLeng)
{
    static int32_t an32ShiftReg[N];
    int32_t n32Acc;
    int32_t n32Data;
    int32_t n32Temp;
    int32_t n32Loop;
    int32_t n32NumXfer4B;
    int32_t n32XferCnt;

    n32NumXfer4B = (regXferLeng + (sizeof(int32_t) - 1)) / sizeof(int32_t);
XFER_LOOP:
    for (n32XferCnt = 0; n32XferCnt < n32NumXfer4B; n32XferCnt++) {
        n32Acc = 0;
        n32Temp = pn32HPInput[n32XferCnt];
SHIFT_ACC_LOOP:
        for (n32Loop = N - 1; n32Loop >= 0; n32Loop--) {
            if (n32Loop == 0) {
                an32ShiftReg[0] = n32Temp;
                n32Data = n32Temp;
            } else {
                an32ShiftReg[n32Loop] = an32ShiftReg[n32Loop - 1];
                n32Data = an32ShiftReg[n32Loop];
            }
            n32Acc += n32Data * an32Coef[n32Loop];
        }
        pn32HPOutput[n32XferCnt] = n32Acc;
    }

    return;
}
```

這是一個實現FIR 濾波器運算的程式碼，首先定義多個參數，型別為 int32_t，用於存儲變數或累積變數(for 迴圈)。XFER_LOOP 為進行數據傳輸的迴圈，

並在每次執行 for 迴圈時將 n32Acc 初始化為零。SHIFT_ACC_LOOP 迴圈則是負責移位和累積的運算，將移位暫存器 an32ShiftReg 中的值移位，並將新的值存儲在 n32Data 中，用於與濾波器的係數相乘(n32Acc += n32Data * an32Coef[n32Loop])，最終將 n32Acc 存儲在輸出 pn32HPOutput[n32XferCnt]。

(3) FIRTester.cpp

```cpp
#include <iostream>
#include "FIR.h"

using namespace std;

#define NUM_SAMPLES        600

int main(int argc, char *argv[])
{
    FILE        *fp;

    int32_t signal;
    int32_t input[NUM_SAMPLES], output[NUM_SAMPLES];
    int32_t taps[N] = {0,-10,-9,23,56,63,56,23,-9,-10,0};

    int i, ramp_up;

    cout << ">> Start test!" << endl;

    signal = 0;
    ramp_up = 1;

    fp = fopen("out.dat", "w");
    for (i = 0; i < NUM_SAMPLES; i++) {
        if (ramp_up == 1)
            signal = signal + 1;
        else
            signal = signal - 1;

        input[i] = signal;

        if ((ramp_up == 1) && (signal >= 75))
            ramp_up = 0;
        else if ((ramp_up == 0) && (signal <= -75))
            ramp_up = 1;
    }

    // Execute the function with latest input
    fir_n11_maxi(input, output, taps, NUM_SAMPLES * sizeof(int32_t));

    // Save the results.
    for (i = 0; i < NUM_SAMPLES; i++) {
        fprintf(fp,"%i %d %d\n", i, input[i], output[i]);
    }

    fclose(fp);

    cout << ">> Comparing against output data..." << endl;
    if (system("diff -//out.dat /home/ubuntu/LAB/soc_lab2/FIRN11MAXI/vitis_hls/out_gold.dat")) {
        cout << ">> Test failed!" << endl;
        return 1;
    }

    cout << ">> Test passed!" << endl;
    cout << "------------------------" << endl;
    return 0;
}
```

這段程式碼是為了生成一鋸齒波信號，並與 outgold.dat 進行比對，若相符則顯示 Test Passed，否則顯示 Test failed。

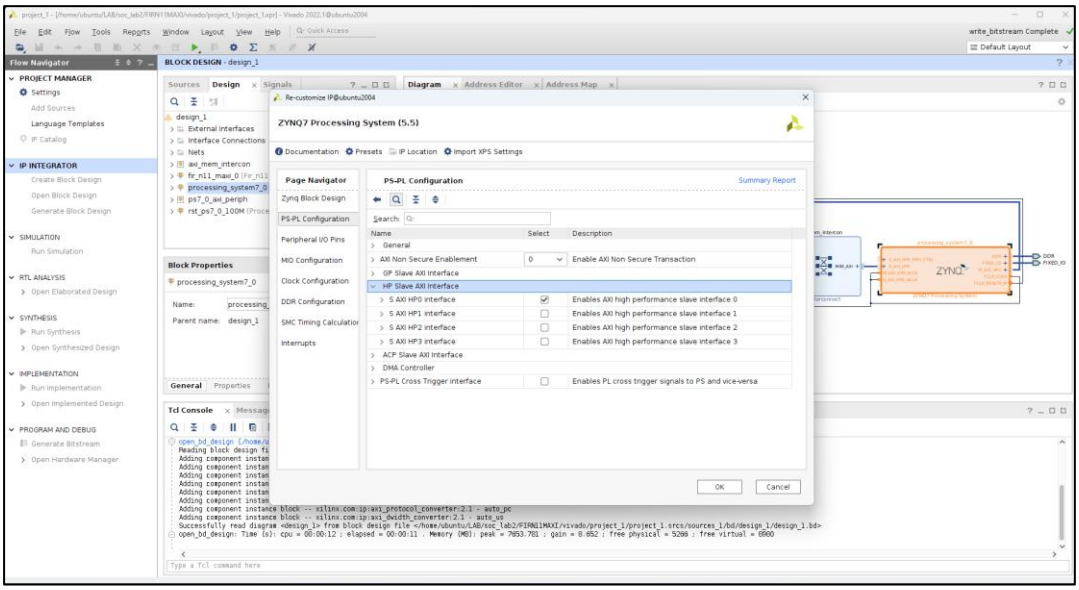
(4) Directives.tcl

```
############################################################
## This file is generated automatically by Vitis HLS.
## Please DO NOT edit it.
## Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
############################################################
set_directive_interface -mode s_axilite "fir_n11_maxi"
set_directive_interface -mode m_axi -depth 600 -offset slave "fir_n11_maxi" pn32HPInput
set_directive_interface -mode m_axi -depth 600 -offset slave "fir_n11_maxi" pn32HPOutput
set_directive_interface -mode s_axilite "fir_n11_maxi" an32Coef
set_directive_interface -mode s_axilite "fir_n11_maxi" regXferLeng
```

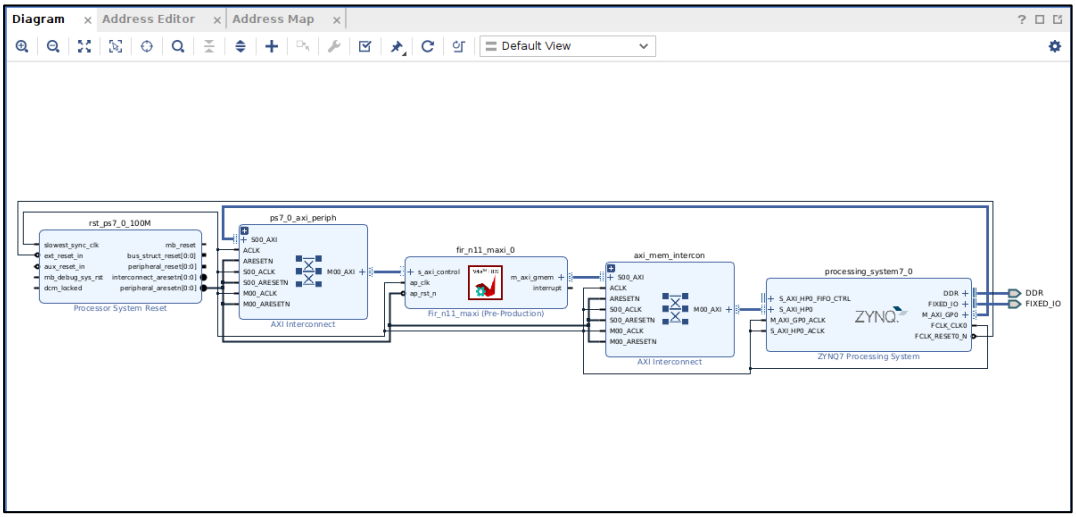
本次 lab 與上次使用#pragma 方式不同，是直接在 FIR.cpp 的 DIRECTIVE 部分 insert，insert 完成後可於 solution/constraints/directives.tcl 部分檢查。

2. Vivado Implementation

在 Vitis_HLS Export RTL 後，接著在 Vivado 開啟 IP。步驟大致與 LAB1 相同，較不同的地方是針對 ZYNQ7 Processing System，需調整 HP Port 設定，如下圖:



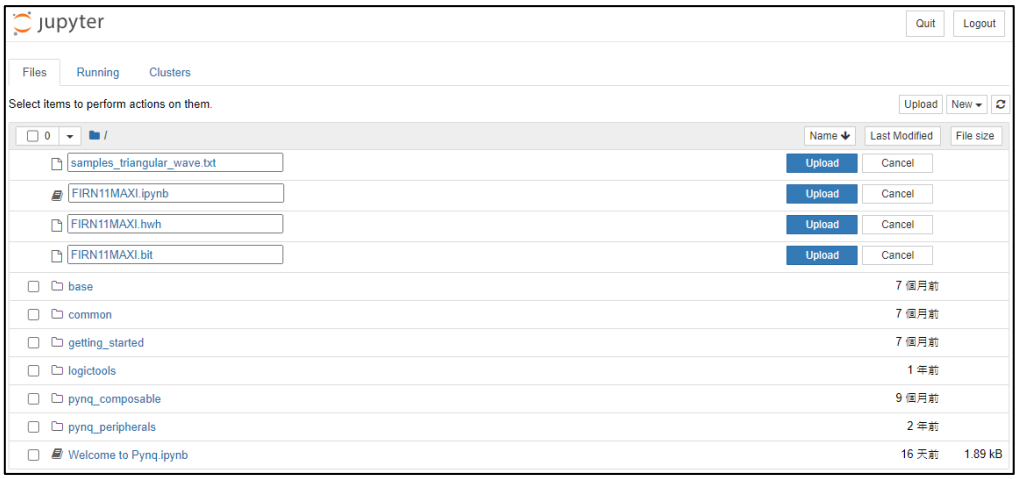
調整完成後即可將 ZYNQ7 與自行設計的 IP 連線，進行 auto connection，完成後如下圖:



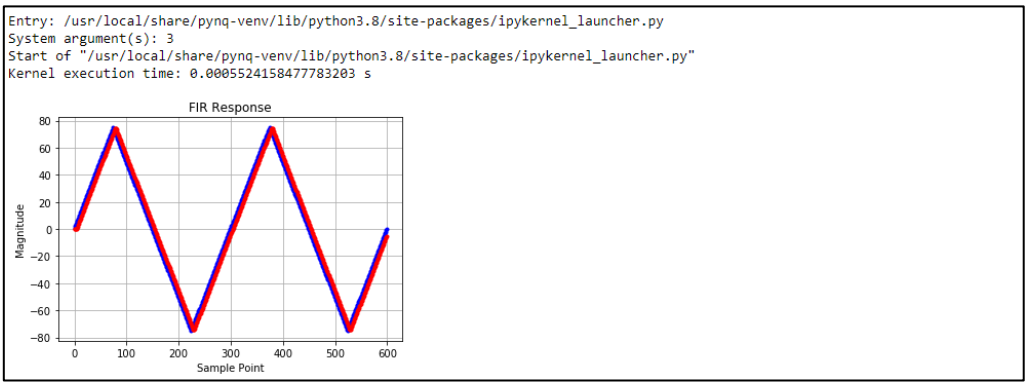
接著與 LAB1 相同，Create HDL Wrapper → Generate bitstream。

3. Python Code Validation via Jupyter Notebook

Upload .bit, .hwh, .ipynb, samples_triangular_wave.txt 檔案至 Jupyter Notebook:



執行.ipynb 檔案，觀察下方輸出之圖形:



II. FIR with Interface Streaming

1. HLS/IP Design

(1) FIR.h

```
#ifndef __FIR_H__
#define __FIR_H__

#include <ap_axi_sdata.h>
#include <hls_stream.h>

#define N                       11

#define MAP_ALIGN_4INT          (((N + 3) >> 2) << 2)

typedef ap_axiu<32,1,1,1> value_t;
typedef hls::stream<value_t> stream_t;
typedef ap_uint<32> reg32_t;
typedef signed int int32_t;
typedef unsigned int uint32_t;

void fir_n11_strm(stream_t* pstrmInput, stream_t* pstrmOutput, int32_t an32Coef[MAP_ALIGN_4INT], reg32_t regXferLeng);

#endif
```

與 FIRMAXI 大致相同，唯一不同處在於〝typedef hls::stream<value_t> stream_t;〞，hls::stream 是用來對流數據結構進行建模，流數據(stream data)是指數據的採樣是按順序進行的，所以不需要地址層面的寫入或讀出等操作。

(2) FIR.cpp

```
#include "FIR.h"

void fir_n11_strm(stream_t* pstrmInput, stream_t* pstrmOutput, int32_t an32Coef[MAP_ALIGN_4INT], reg32_t regXferLeng)
{
#pragma HLS INTERFACE s_axilite port=regXferLeng
#pragma HLS INTERFACE s_axilite port=an32Coef
#pragma HLS INTERFACE axis register both port=pstrmOutput
#pragma HLS INTERFACE axis register both port=pstrmInput
#pragma HLS INTERFACE s_axilite port=return
    static int32_t an32ShiftReg[N];
    int32_t n32Acc;
    int32_t n32Data;
    int32_t n32Temp;
    int32_t n32Loop;
    int32_t n32NumXfer4B;
    int32_t n32XferCnt;
    value_t valTemp;

    n32NumXfer4B = (regXferLeng + (sizeof(int32_t) - 1)) / sizeof(int32_t);
XFER_LOOP:
    for (n32XferCnt = 0; n32XferCnt < n32NumXfer4B; n32XferCnt++) {
        n32Acc = 0;
        value_t valTemp = pstrmInput->read();
        n32Temp = valTemp.data;
SHIFT_ACC_LOOP:
        for (n32Loop = N - 1; n32Loop >= 0; n32Loop--) {
            if (n32Loop == 0) {
                an32ShiftReg[0] = n32Temp;
                n32Data = n32Temp;
            } else {
                an32ShiftReg[n32Loop] = an32ShiftReg[n32Loop - 1];
                n32Data = an32ShiftReg[n32Loop];
            }
            n32Acc += n32Data * an32Coef[n32Loop];
        }
        valTemp.data = n32Acc;
        pstrmOutput->write(valTemp);
        if (valTemp.last) break;
    }

    return;
}
```

與 AXI 大致相同，〝value_t valTemp = pstrmInput->read();〞value_t 是存儲數據流內讀取的數據，並從 valTemp 中提取整數數據，將其存儲在 n32Temp 變數中。在 for 迴圈內做一些位移判斷或是乘積運算(FIR 運算)，最後將得到的輸出值 n32Acc 存回 valTemp，寫入輸出數據流 pstrmOutput 中。

(3) FIRTester.cpp

```cpp
#include <iostream>
#include "FIR.h"

using namespace std;

#define NUM_SAMPLES         600

int main(int argc, char *argv[])
{
    FILE         *fp;

    int32_t signal;
    int32_t input[NUM_SAMPLES], output[NUM_SAMPLES];
    stream_t strmInput, strmOutput;
    int32_t taps[N] = {0,-10,-9,23,56,63,56,23,-9,-10,0};
    value_t valDataCtrl;

    int i, ramp_up;

    cout << ">> Start test!" << endl;

    signal = 0;
    ramp_up = 1;
    valDataCtrl.data = 0;
    valDataCtrl.keep = 0xF;
    valDataCtrl.strb = 0;
    valDataCtrl.user = 0;
    valDataCtrl.last = 0;
    valDataCtrl.id = 0;
    valDataCtrl.dest = 0;
    fp = fopen("out.dat", "w");
    for (i = 0; i < NUM_SAMPLES; i++) {
        if (ramp_up == 1)
            signal = signal + 1;
        else
            signal = signal - 1;

        input[i] = signal;
        valDataCtrl.data = signal;
        if (i >= (NUM_SAMPLES - 1)) valDataCtrl.last = 1;
        strmInput.write(valDataCtrl);

        if ((ramp_up == 1) && (signal >= 75))
            ramp_up = 0;
        else if ((ramp_up == 0) && (signal <= -75))
            ramp_up = 1;
    }

    // Execute the function with latest input
    fir_n11_strm(&strmInput, &strmOutput, taps, NUM_SAMPLES * sizeof(int32_t));

    // Save the results.
    for (i = 0; i < NUM_SAMPLES; i++) {
        output[i] = strmOutput.read().data;
        fprintf(fp,"%i %d %d\n", i, input[i], output[i]);
    }

    fclose(fp);
```

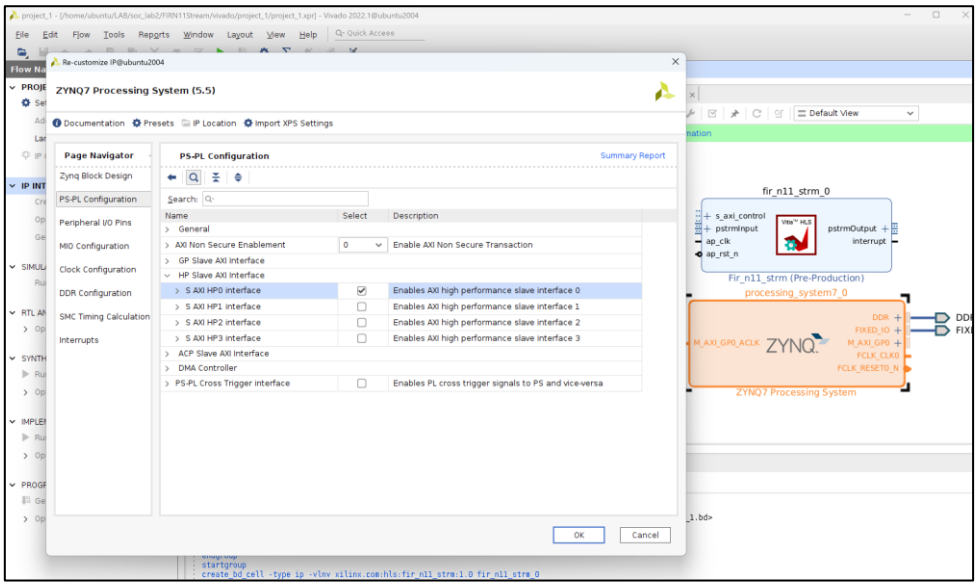
與 FIRN11MAXI 的 FIRTester 大致相同，除了一些部份改成用 streaming data 方式傳輸，主要還是產生一鋸齒波功能。

此部分主要是學習 AXI 與 Stream 的差別，其實現的功能都是 FIR 運算，還有學會如何不靠指令完成 directives 的 insertion。針對 HLS C++語言還是有非常多特別的用法，需要花時間慢慢理解每一行的意義及其目的。

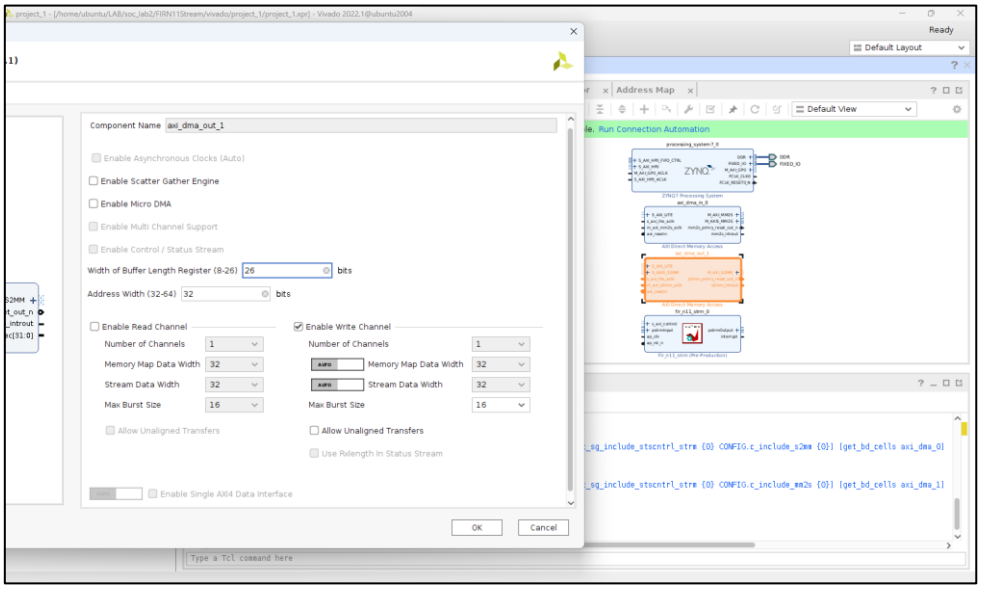
2. Vivado Implementation

與之前相同，先至 settings 將 vitis_hls export 的 IP 匯入。

匯入後，因 AXI-Lite 與 AXI-Stream 在 processing system block 使用的 port 並不相同，所以須調整 HP port 設定，如下圖:



另外，AXI-Master to stream 是用 Xilinx DMA IP 來實現，除了自行設計的 IP 及 ZYNQ7，還需加入 2 個 AXI Direct Memory Access，下圖為其中一個 IP(axi_dma_out_0):

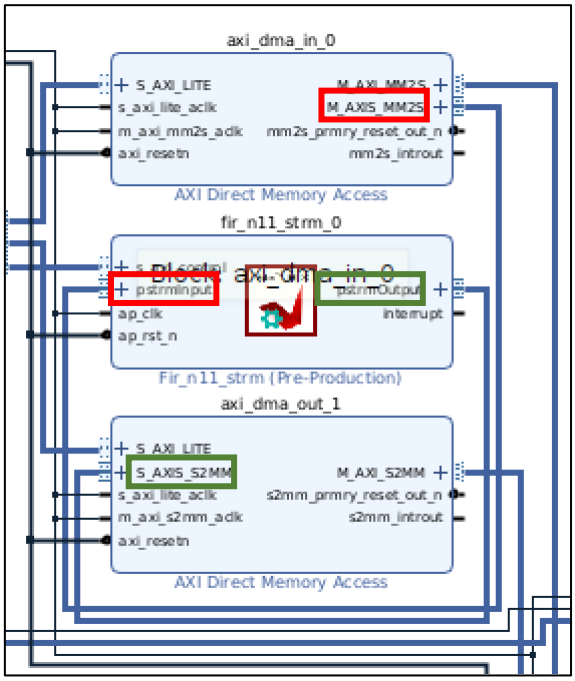


加入完成後即可進行連線(auto connection)，自動連線完成還需手動連線，

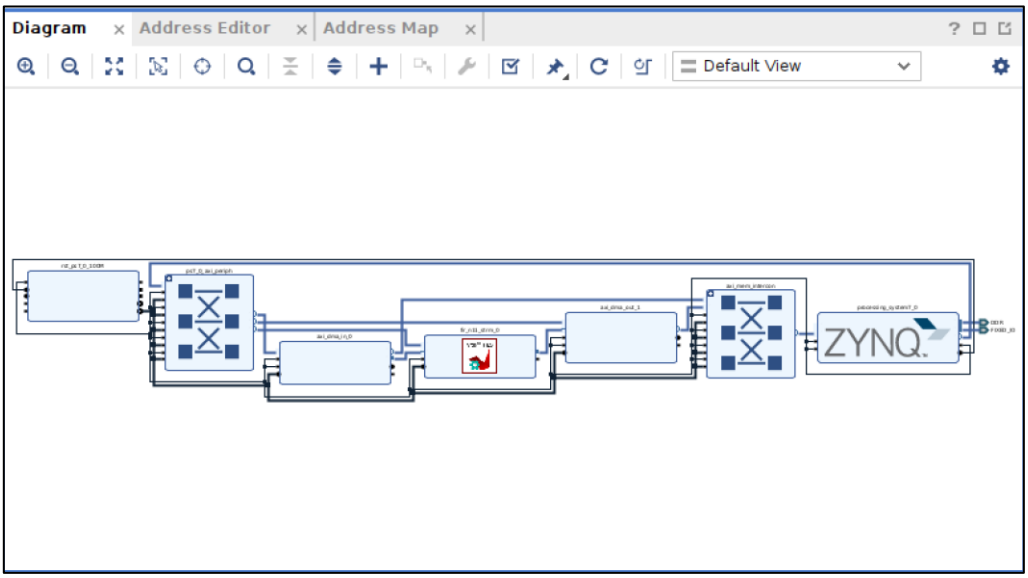
1. FIR_N11_STRM 的 pstrminput 接到 axi_dma_in_0 的 M_AXIS_MM2S

2. FIR_N11_STRM 的 pstrmoutput 接到 axi_dma_out_0 的 S_AXIS_S2MM

如下圖:

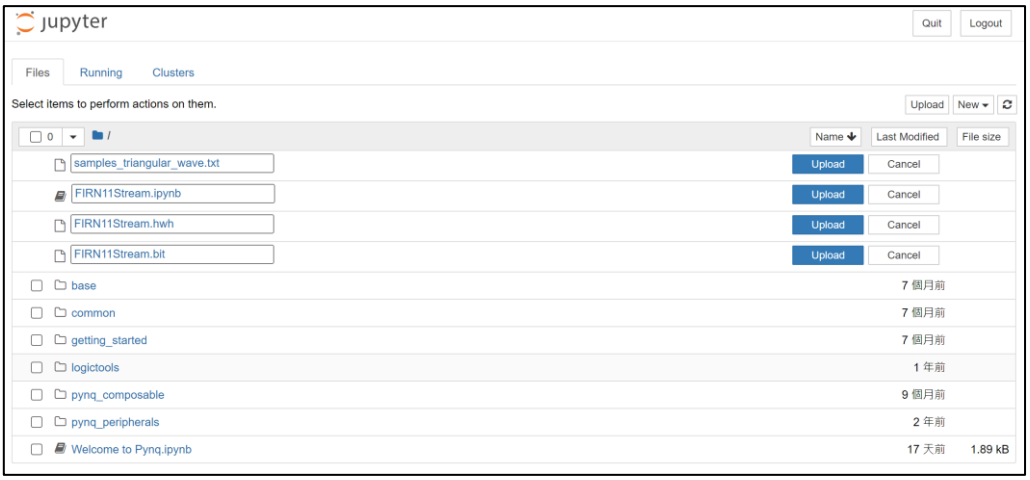


整體 block diagram 如下圖:

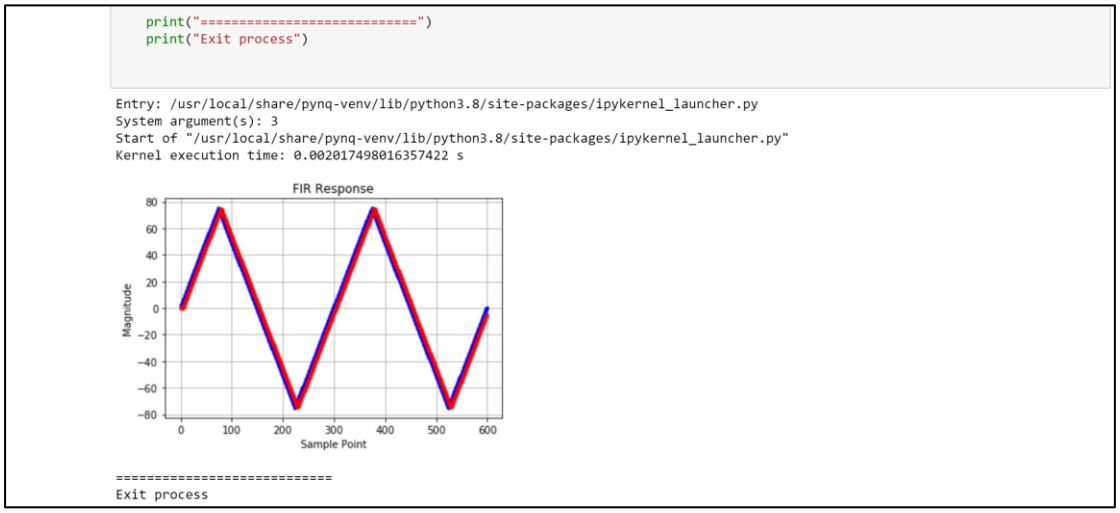


3. Python Code Validation via Jupyter Notebook

Upload .bit, .hwh, .ipynb, samples_triangular_wave.txt 檔案至 Jupyter Notebook:



執行.ipynb 檔案，觀察下方輸出之圖形:



III. Discussion

這次 lab 主要是學習兩種數據傳輸、儲存的方式，分別為 AXI Master Interface 與 Stream Interface，接下來針對這兩者整理出其差別之處:

數據交互方式:

AXI Master Interface 通常可以指定特定位址進行讀取及寫入，且可以在暫存器上進行讀寫，而 Stream Interface 的數據傳輸因為是連續的，沒有位址的概念，可應用在信號處理或是圖像處理等。

接口配置:

AXI Master Interface 需要配置地址、數據深度(depth)、傳輸協議等參數，而 Stream Interface 只需配置數據寬度，因為數據是以流(stream)的方式傳輸，故不涉及地址層面的配置參數。

數據傳輸方式:

AXI Master Interface 支持隨機傳輸，可以對任意點對點之間進行傳輸讀寫，而 Stream Interface 則以數據流的方式進行傳輸，沒有地址或暫存器的概念，客製程度不如 AXI Master Interface，但其運算處理的速度會更為快速。