電子系統層級設計與驗證

Homework 4

DNN Accelerator

組別:第 7 組

組員:蔡明翰 B083012028

組員:王證皓 B083012043

日期：112/05/30

(一)實驗主題

使用C++完成DNN(Deep Neural Network) Accelerator，DNN的定義為三層以上的神經網路，所以我們結合LeNet與DNN的架構去實作，實驗步驟為先使用Python訓練產生各層的Weights，訓練集採用MNIST手寫數字辨識，接著用C++完成並用HLS合成DNN加速電路，並使用Pynq-z2去做驗證。

(二)實驗內容

1. Python整體架構

```python
model = Sequential(
    [
        Conv2D(filters=1,kernel_size=3,activation="relu",input_shape=(10, 10, 1),padding="valid",use_bias=False),
        MaxPooling2D(pool_size=2),
        Conv2D(filters=16,kernel_size=3,activation="relu",padding="valid",use_bias=False),
        MaxPooling2D(pool_size=2),
        Flatten(),
        Dense(16, activation="relu", use_bias=False),
        Dense(12, activation="relu", use_bias=False),
        Dense(10, activation="softmax", use_bias=False),
    ]
)
```

2. C++程式碼

(1)Convolutional Layer 1

```cpp
void convolution1(float input[100], const float kernel[3][3], float output[8][8]) {
    int i, j, m, n;
    float sum;
    float temp_input[10][10]={1};

    for(int p=0;p<10;p++){
        for(int q=0;q<10;q++){
            temp_input[p][q]=input[p*10+q];
        }
    }

    for (i = 0; i < 8; i++) {
        for (j = 0; j < 8; j++) {
            sum = 0.0;
            for (m = 0; m < 3; m++) {
                for (n = 0; n < 3; n++) {
                    sum += temp_input[i + m][j + n] * kernel[m][n];
                }
            }

            output[i][j] = ReLU(sum);
        }
    }
}
```

(2)MaxPooling Layer 1

```
void maxPooling1(float input[8][8], float output[4][4]) {
    int i, j, m, n;
    float max_value;
    for (i = 0; i < 8; i += 2) {
        for (j = 0; j < 8; j += 2) {
            max_value = input[i][j];
            for (m = 0; m < 2; m++) {
                for (n = 0; n < 2; n++) {
                    if (input[i + m][j + n] > max_value) {
                        max_value = input[i + m][j + n];
                    }
                }
            }
            output[i / 2][j / 2] = max_value;
        }
    }
}
```

(3)Convolutional Layer 2

```
void convolution2(float input[4][4], const float kernel[16][3][3], float output[16][2][2]) {
    int i, j, m, n,k;
    float sum;
for(k=0;k<16;k++){
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++) {
            sum = 0.0;
            for (m = 0; m < 3; m++) {
                for (n = 0; n < 3; n++) {
                    sum += input[i + m][j + n] * kernel[k][m][n];
                }
            }
            output[k][i][j] =ReLU(sum);
        }
    }
}
}
```

(4)MaxPooling Layer 2

```
void maxPooling2(float input[16][2][2], float output[16][1][1]) {
    int i, j, m, n,p;
    float max_value;
for(p=0;p<16;p++){
    for (i = 0; i < 2; i += 2) {
        for (j = 0; j < 2; j += 2) {
            max_value = input[p][i][j];
            for (m = 0; m < 2; m++) {
                for (n = 0; n < 2; n++) {
                    if (input[p][i + m][j + n] > max_value) {
                        max_value = input[p][i + m][j + n];
                    }
                }
            }

            output[p][i / 2][j / 2] = max_value;
        }
    }
}
}
```

(5)Fully-Connected

```
for(int p=0;p<16;p++){
for (int i = 0; i < 1; i++) {
    for (int j = 0; j < 1; j++) {
        flatten_output[p] = pool2_output[p][i][j];
    }
}
}
```

(6)Hidden Layer 1

```
for (int i = 0; i < 16; i++) {
    float sum = 0.0;
    for (int j = 0; j < 16; j++) {
        sum += flatten_output[j] * weights::fc1_weight[j][i];
    }
    fc1_output[i] = ReLU(sum);
}
```

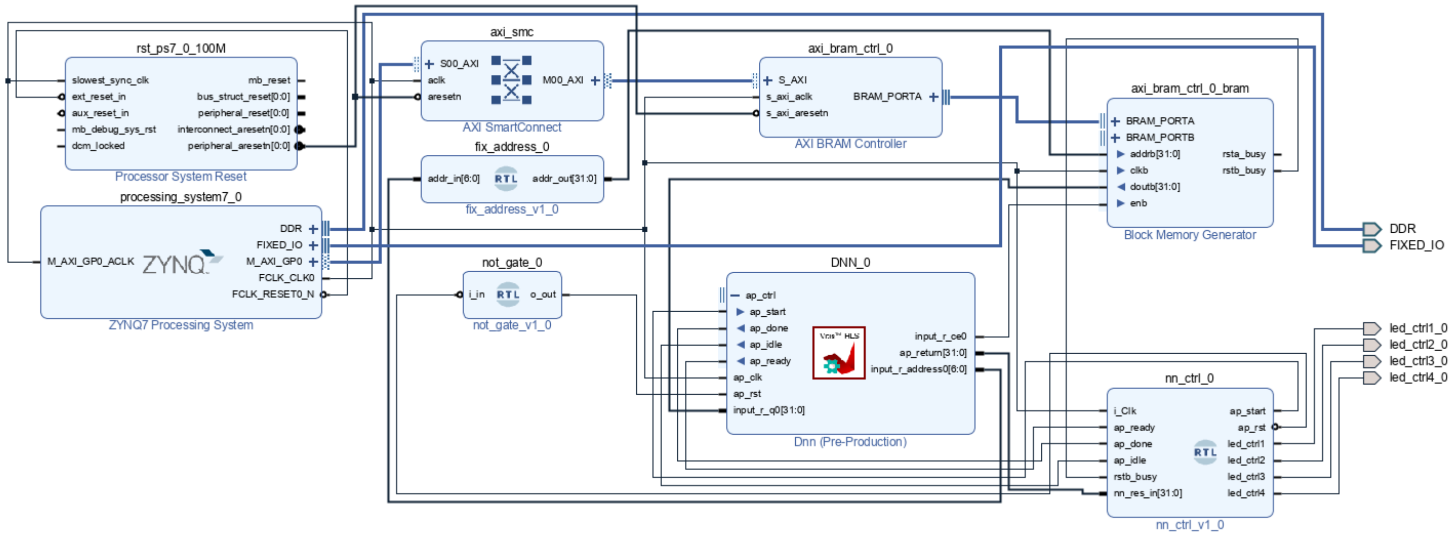
(7)Hidden Layer 2

```
for (int i = 0; i < 12; i++) {
    float sum = 0.0;
    for (int j = 0; j < 16; j++) {
        sum += fc1_output[j] * weights::fc2_weight[j][i];
    }
    fc2_output[i] = ReLU(sum);
}
```

(8)Output Layer & Prediction

```
for (int i = 0; i < 10; i++) {
    float sum = 0.0;
    for (int j = 0; j < 12; j++) {
        sum += fc2_output[j] * weights::output_weight[j][i];
    }
    temp_output[i] = exp(sum);
}
for(int i=0;i<10;i++){
    s+=temp_output[i];
}
for (int i = 0; i < 10; i++){
    output[i]=temp_output[i]/s;
}
int max_idx = -1;
float max_val = -999.9;
for (int i = 0; i < 10; i++){
    if (output[i] > max_val){
        max_idx = i;
        max_val = output[i];
    }
}
```

3.Vivado Block Diagram



(三)實驗結果

1.Python訓練結果

```
Epoch 1000/1000
27/27 [==============================] - 1s 20ms/step - loss: 0.3545 - accuracy: 0.8868 - val_loss: 0.3700 - val_accuracy: 0.8840
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 8, 8, 1)           9

 max_pooling2d (MaxPooling2D  (None, 4, 4, 1)          0
 )

 conv2d_1 (Conv2D)           (None, 2, 2, 16)          144

 max_pooling2d_1 (MaxPooling  (None, 1, 1, 16)         0
 2D)

 flatten (Flatten)           (None, 16)                0

 dense (Dense)               (None, 16)                256

 dense_1 (Dense)             (None, 12)                192

 dense_2 (Dense)             (None, 10)                120

=================================================================
Total params: 721
Trainable params: 721
Non-trainable params: 0
_________________________________________________________________
Inference time for  10000  test image:  0.5619938373565674  seconds
test loss, test acc:  [0.35278642177581787, 0.8876000046730042]
test_image[0] label:  7
1/1 [==============================] - 0s 104ms/step
NN Prediction:  7
Finished
```

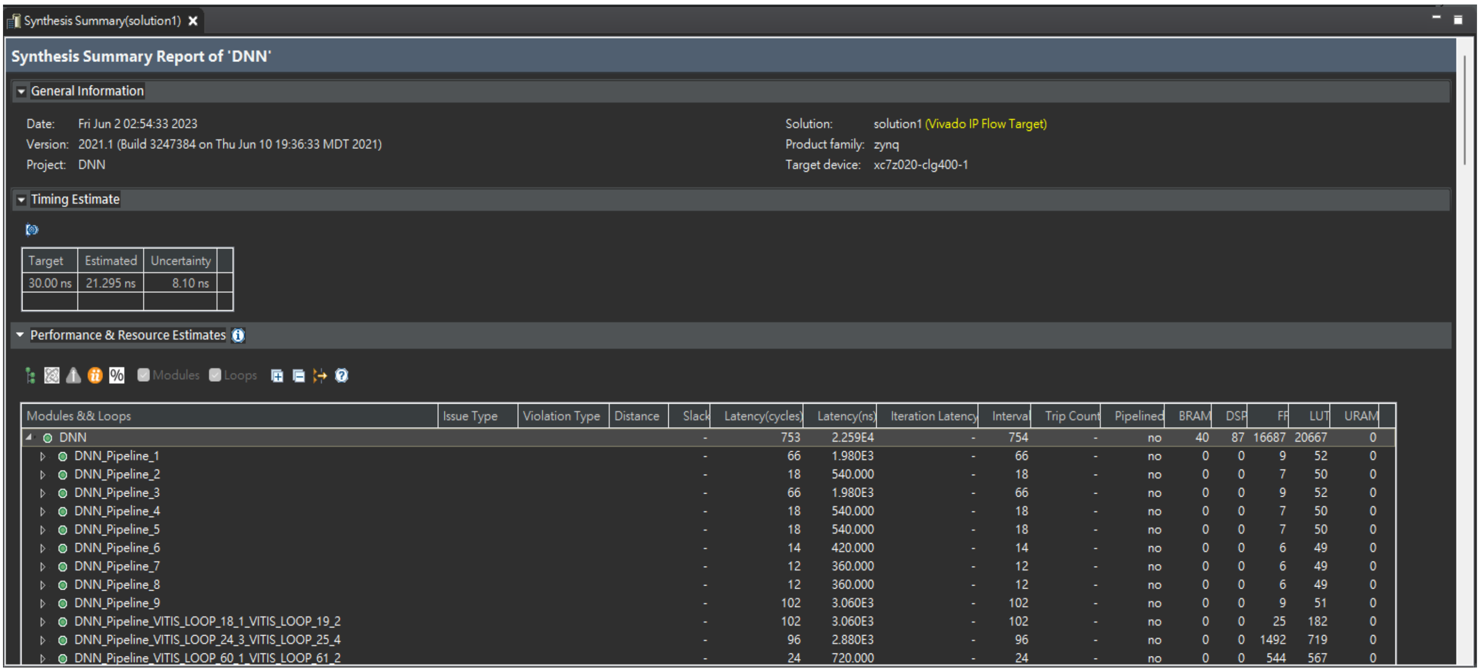
準確率:88.68%

2.C Simulation

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../DNN_tb.cpp in debug mode
   Compiling ../../../../DNN.cpp in debug mode
   Generating csim.exe

NN Prediction: 1


NN Prediction: 3

INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

輸入第一張圖為數字1、第二張圖為數字3，預測結果皆與預期相符。

3.C Synthesis



Synthesis Summary(solution1)

Synthesis Summary Report of 'DNN'

General Information

Date: Fri Jun 2 02:54:33 2023
Version: 2021.1 (Build 3247384 on Thu Jun 10 19:36:33 MDT 2021)
Project: DNN
Solution: solution1 (Vivado IP Flow Target)
Product family: zynq
Target device: xc7z020-clg400-1

Timing Estimate

Target	Estimated	Uncertainty
30.00 ns	21.295 ns	8.10 ns

Performance & Resource Estimates

Modules Loops

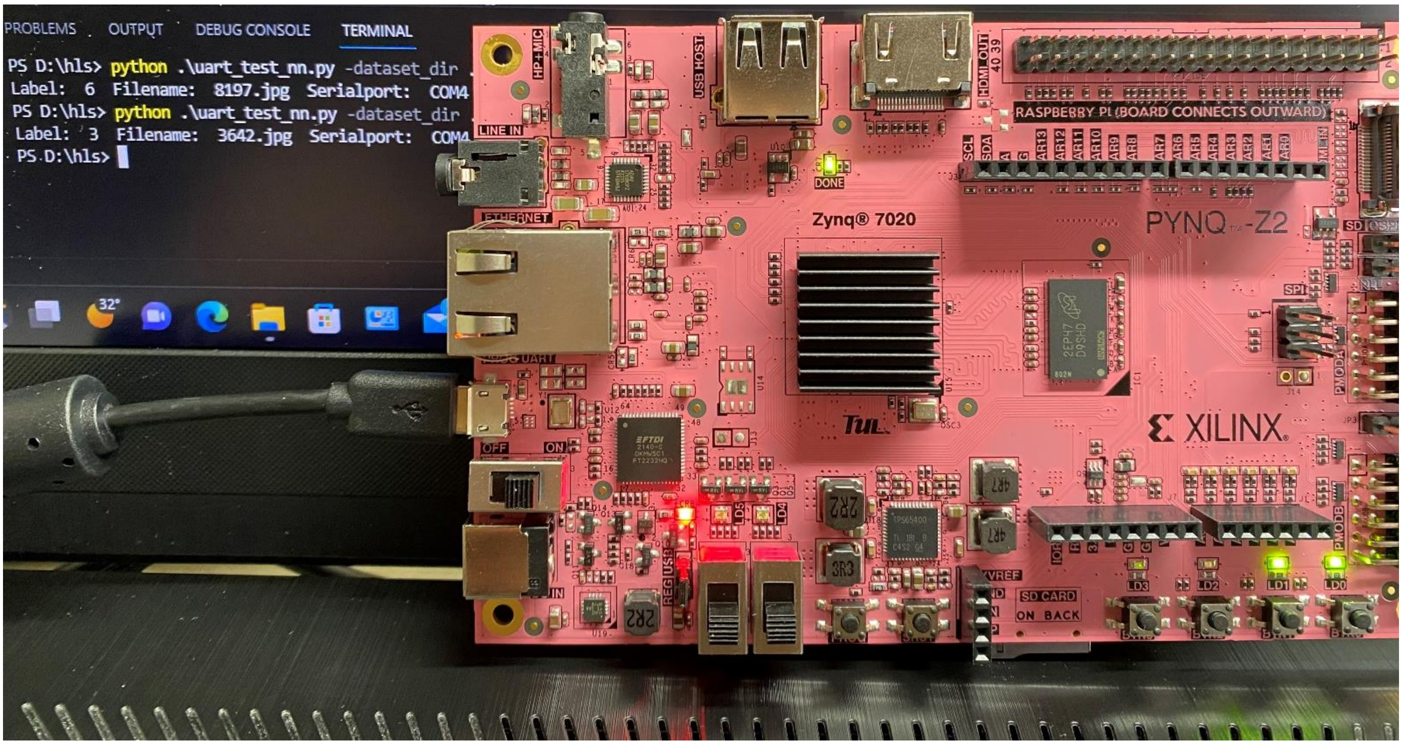
Modules && Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
DNN				-	753	2.259E4	-	754	-	no	40	87	16687	20667	0
DNN_Pipeline_1				-	66	1.980E3	-	66	-	no	0	0	9	52	0
DNN_Pipeline_2				-	18	540.000	-	18	-	no	0	0	7	50	0
DNN_Pipeline_3				-	66	1.980E3	-	66	-	no	0	0	9	52	0
DNN_Pipeline_4				-	18	540.000	-	18	-	no	0	0	7	50	0
DNN_Pipeline_5				-	18	540.000	-	18	-	no	0	0	7	50	0
DNN_Pipeline_6				-	14	420.000	-	14	-	no	0	0	6	49	0
DNN_Pipeline_7				-	12	360.000	-	12	-	no	0	0	6	49	0
DNN_Pipeline_8				-	12	360.000	-	12	-	no	0	0	6	49	0
DNN_Pipeline_9				-	102	3.060E3	-	102	-	no	0	0	9	51	0
DNN_Pipeline_VITIS_LOOP_18_1_VITIS_LOOP_19_2				-	102	3.060E3	-	102	-	no	0	0	25	182	0
DNN_Pipeline_VITIS_LOOP_24_3_VITIS_LOOP_25_4				-	96	2.880E3	-	96	-	no	0	0	1492	719	0
DNN_Pipeline_VITIS_LOOP_60_1_VITIS_LOOP_61_2				-	24	720.000	-	24	-	no	0	0	544	567	0

4.C/RTL Co-Simulation

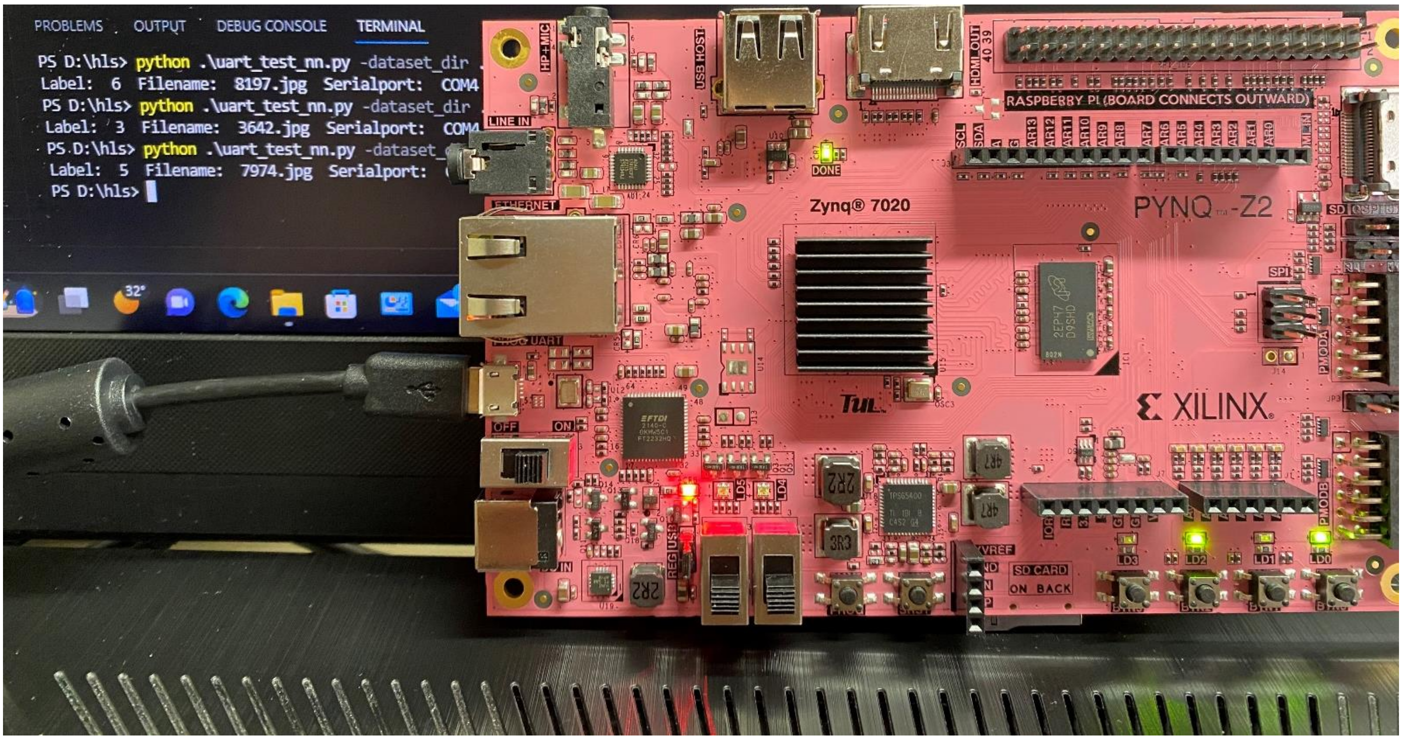
```
INFO: [Common 17-206] Exiting xsim at Fri Jun  2 16:41:13 2023...
INFO: [COSIM 212-316] Starting C post checking ...

NN Prediction: 1

NN Prediction: 3

INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
```

5.PYNQ-Z2驗證

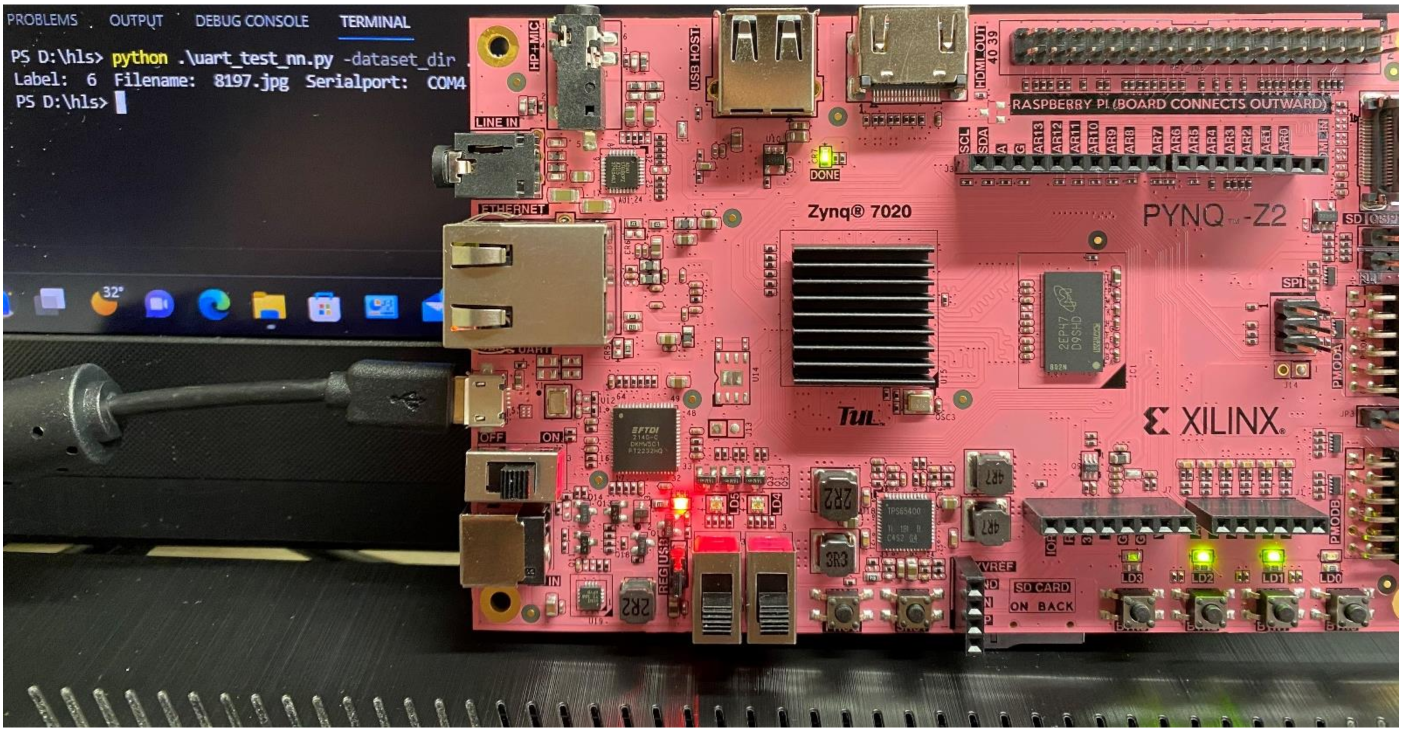
(1)輸入數字3



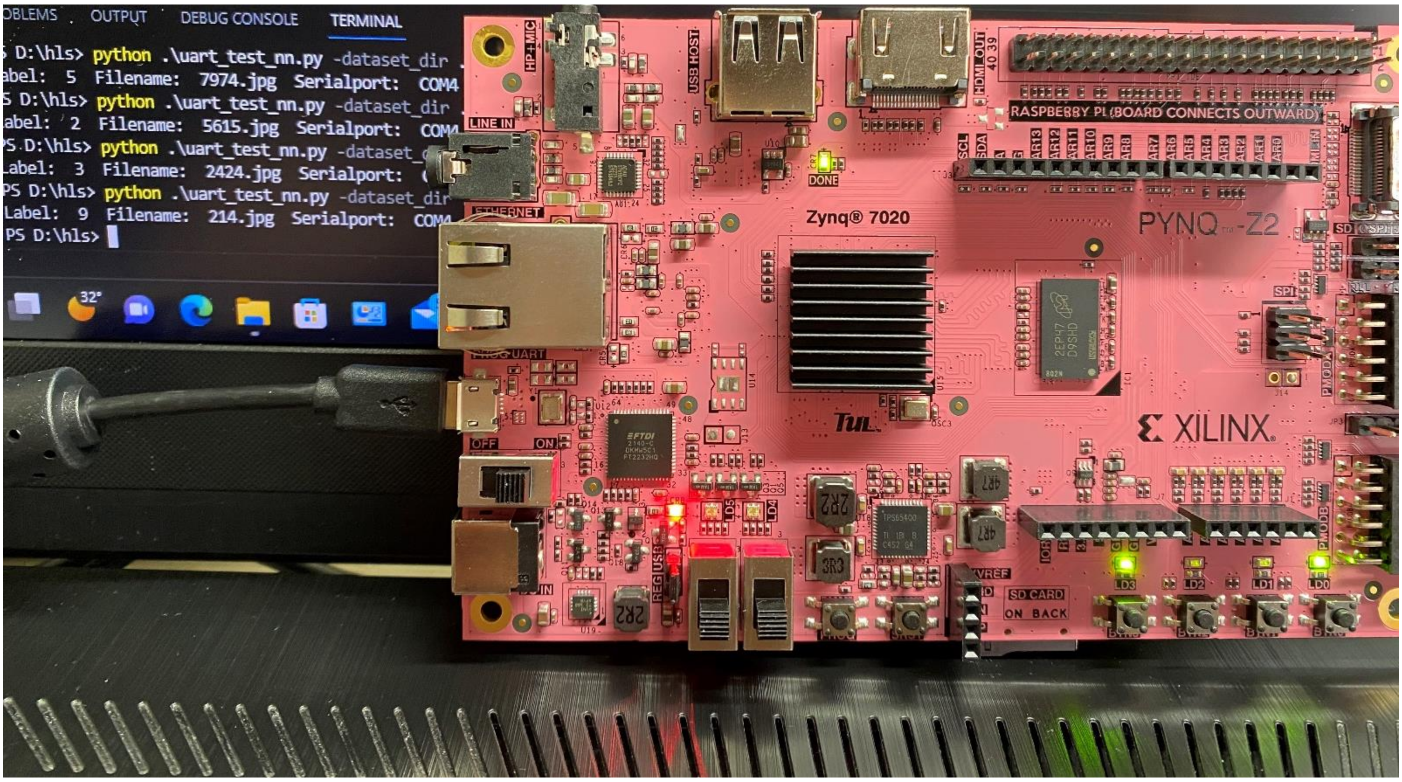
(2)輸入數字5



(3)輸入數字6



(4)輸入數字9



（四）實驗心得

這次實驗透過 HLS 去實作 DNN 加速器，在過程中雖然有遇到許多困難，像是我們以前從未接觸過 Python，因此要透過 Python 來訓練權重時，還自行上網學習相關語法，而且在 C Synthesis 完後因沒做 C/RTL Co-Simulation，沒發現合成完的 Verilog 有錯誤，到上板子時才發現 LED 顯示錯誤，debug 很久才想出來，所以這也讓我們體會到驗證的重要性，一個完整的電路都會經過層層驗證的，不過幸好最後都能成功排除困難，完成最後的實驗。