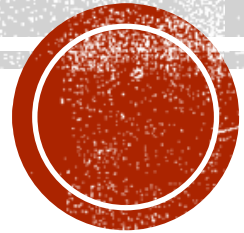
學期實驗大綱

1. MIPS assembly programming
2. FC-DNN design using PyTorch
3. FC-DNN implementation on Zynq
4. Verilog modeling & simulation
5. Master/slave coprocessing on Zynq
6. Mips-Core accelerator
7. Trace-driven cache simulation

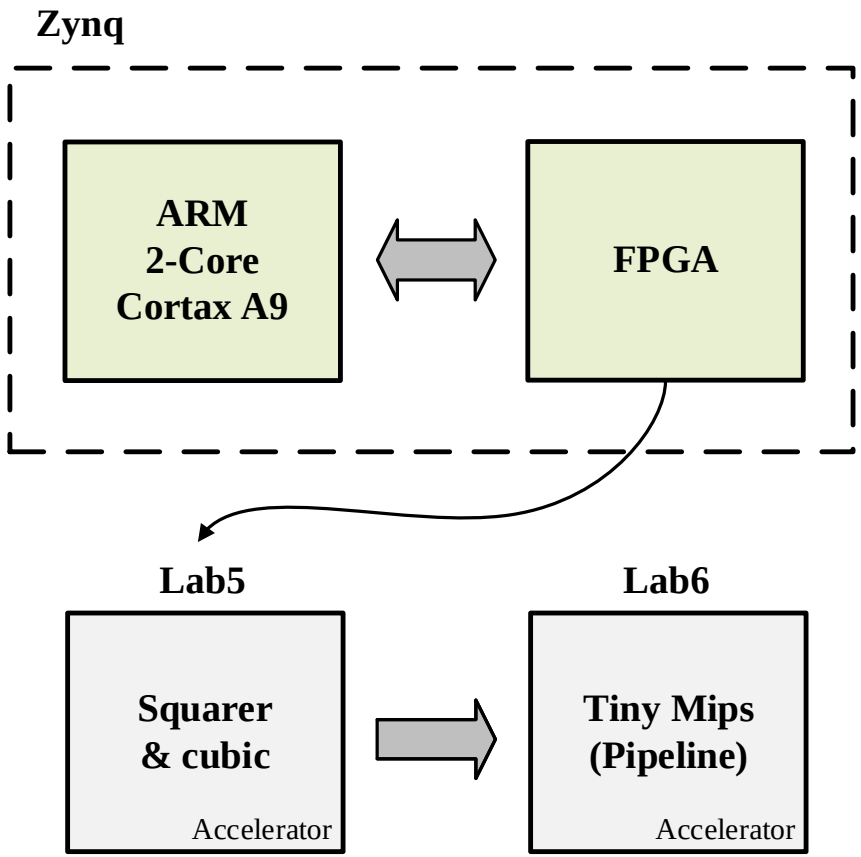
Mips-Core accelerator (Software based)

助教: 郭帛霖、廖柏宸



前情提要

- 在上一次的實驗中，學會了有關 Zynq 處理系統（PS）和可程式邏輯（PL）傳輸的基本概念。
- 這次的實驗目標是實現一個簡單的 MIPS-Core 加速器，並能執行手寫數字辨識的FC-DNN專案計算。



授課大綱

- **Lab6:**

A. Software (11/29)

1. Zynq與Mips-Core的交互方式
2. 如何在上面運行FC-DNN

B. Hardware (12/4)

1. Tiny mips verilog解析
2. 硬體優化方式

Outline

- **Mips-Core**
  1. 系統規格
  2. Zynq交互方式
  3. Memory map
- **FC-DNN**
  1. 計算資源分配
  2. Mips-Core 計算方式
  3. Mips-Core 單顆神經元計算
     a) C code
     b) Mips code
- **Exercise**

Outline

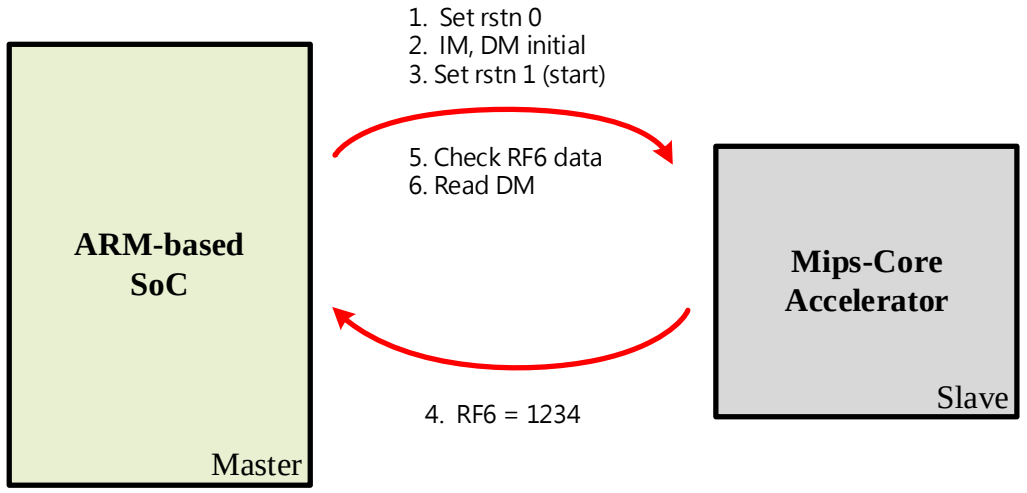
- **Mips-Core**
  1. 系統規格
  2. Zynq交互方式
  3. Memory map
- **FC-DNN**
  1. 計算資源分配
  2. Mips-Core 計算方式
  3. Mips-Core 單顆神經元計算
     a) C code
     b) Mips code
- **Exercise**

Mips-Core – 系統規格

➢ Mips-Core：

1. **Pipeline:** 5 Stage (IF, ID, EXE, MEM, WB), without forwarding unit & branch prediction
2. **IM/DM size:** 8KB (2048 * 32 bits)
3. **Register:** 32 integer registers & 32 floating point register (IEEE754)
4. **Instruction support:**
    1. **Integer:**
        1. R type: add, sub, and, or, slt
        2. I type: addi, lw, sw, beq, bne
        3. J type: j
    2. **Floating point:**
        1. R type: add.s, mul.s
        2. I type: lwc1, swc1

Mips-Core – Zynq交互方式(流程)



- **初始化IM、DM**

  1. Zynq將Mips rstn設為0 (暫停Mips-Core)
  2. Zynq將Instruction寫入Mips-Core IM，Data寫入DM

- **開始計算**

  3. Zynq將Mips-Core rstn設為1 (Mips-Core開始計算)
  4. Mips-Core計算完成後，會將Register 6的值設為1234 (完成值1234，可透過軟體修改)

- **確認計算是否完成**

  5. Zynq polling Register 6的值，當值等於1234，則將Mips-Core rstn設為0 (將Mips-Core做Reset)

- **讀取計算結果**

  6. Zynq從DM讀取計算結果

Mips-Core – Zynq交互方式 (Pseudocode)

```
1.  //mips_  prefix代表mips中的資料
2.  mips_cpu_rstn = 0;                  //停止Mips動作
3.  for i = 1, 2, …, N+1
4.      mips_im[i] = im[i];             //將指令寫入Mips 的instruction memory
5.  for i = 1, 2, …, N+1
6.      mips_dm[i] = dm[i];             //將資料寫入Mips 的data memory
7.
8.  mips_cpu_rstn = 1;                  //讓Mips開始計算
9.  while(mips_rf6_data != 1234);       //確認Mips是否計算完成
10. mips_cpu_rstn = 0;                  //計算完成後，停止Mips動作
11.
12. for i = 1, 2, …, N+1
13.     dm[i] = mips_dm[i] ;            //讀取Mips data memory中的計算結果
```

Mips-Core – Memory map

➢ Memory map：

	Address (multiples of 4)
IM (8KB)	0x40000000 (IM[0]) - 0x40001FFC (IM[2047])
DM (8KB)	0x40002000 (DM[0]) - 0x40003FFC (DM[2047])
RF	0x40004000 (RF[0]) - 0x4000407C (RF[31])
MIPS_RSTN	0x40008004

➢ Zynq C code撰寫方式：

1. 存取IM[0]： *(io + (IM_BASE_OFFSET / 4) + 0)
2. 存取IM[2047]： *(io + (IM_BASE_OFFSET / 4) + 2047)
3. 存取DM[0]： *(io + (DM_BASE_OFFSET / 4) + 0)
4. 存取DM[2047]： *(io + (DM_BASE_OFFSET / 4) + 2047)

Outline

- **Mips-Core**
  1. 系統規格
  2. Zynq交互方式
  3. Memory map
- **FC-DNN**
  1. 計算資源分配
  2. Mips-Core 計算方式
  3. Mips-Core 單顆神經元計算
     a) C code
     b) Mips code
- **Exercise**

FC-DNN – 計算資源分配

- 原先我們將所有的FC-DNN計算全程交給Zynq做處理
- 現在我們讓Mips-Core負責FC-DNN的加權計算，待Mips-Core計算完成後再傳回Zynq做Relu，完成單顆節點的神經元計算。

1. for i=1, 2, …, L+1
2. for j=0, 1, 2, …, N_i-1
3. $X_{i,j} = b_{i,j}$; init. w/ bias (Zynq)
4. for k=0, 1, 2, …, N_{i-1} -1
5. $X_{i,j} += W_{i,j,k} * X_{i-1,k}$; 加權計算(Mips)
6. $X_{i,j} = \max(0, X_{i,j})$;

ReLu (Zynq)

FC-DNN – Mips-Core計算方式

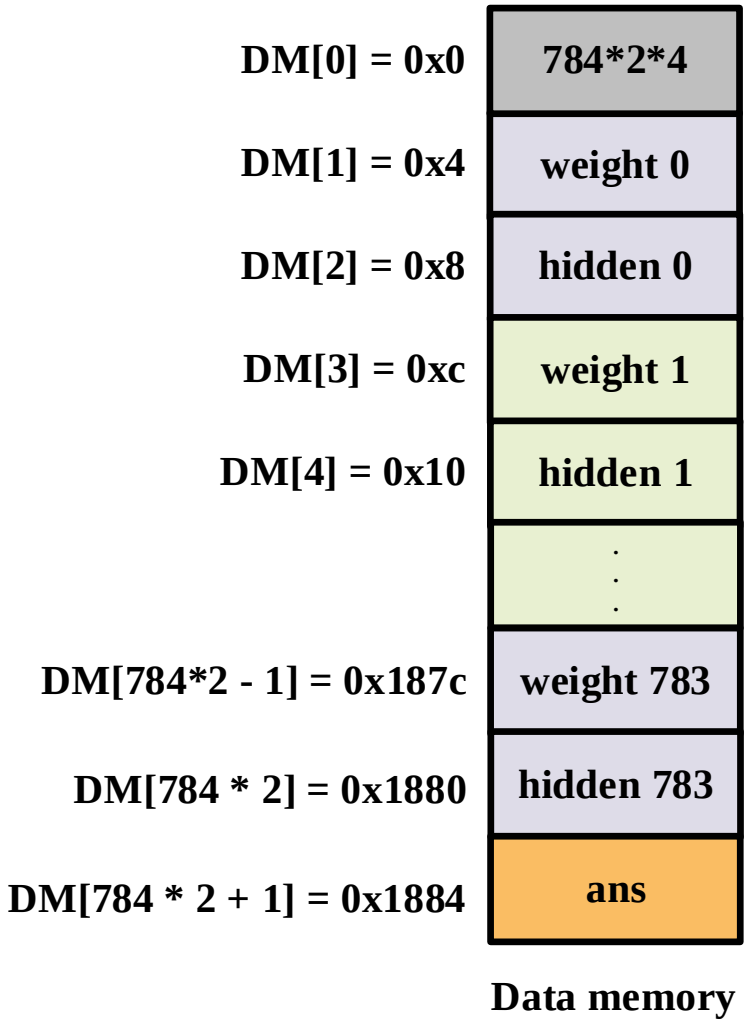
➢ FC-DNN規格：

1. 輸入層：784
2. 隱藏層：64
3. 輸出層：10

➢ Mips-Core FC-DNN計算方式：

1. 依序將隱藏層64顆神經元、輸出層10顆神經元的weight、hidden存入DM，並以單顆神經元做計算，故Mips需重複計算74次。

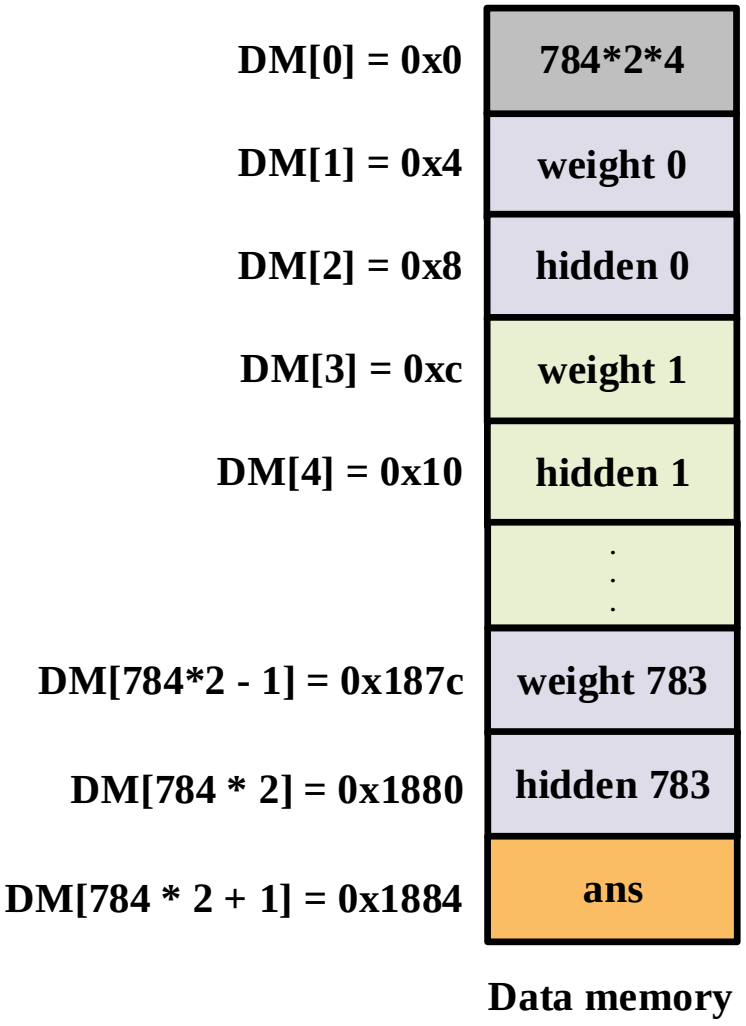
FC-DNN – Mips-Core 單顆神經元計算 (C code)



C code:

```
1.  DM[0] = 784 * 2 *4;                    //DM[0]存放有幾筆數值要做加權
2.  int idx = 0;
3.  float weight, hidden, ans = 0.0f;
4.  while (idx * 4 < DM[0]){               //判斷是否所有的值皆計算完成
5.      idx = idx +  1;
6.      weight = DM[idx];                  //將對應的weigh讀出
7.      idx = idx +  1;
8.      hidden = DM[idx];                  //將對應的hidden讀出
9.      ans = ans + weight * hidden;       //將相乘後的結果，做累加
10. }
11. DM[DM[0] + 1]= ans;                    //將結果存入Data memory
12.
```

FC-DNN – Mips-Core 單顆神經元計算 (Mips code)



Mips code (需插NOP，解決hazard，並且變數需初始化):

```
1.    lw $4, 0($0)              //Load DM[0]
2.    acc:
3.        beq $5, $4, exit      //比對idx與DM[0]是否相同
4.        addi $5, $5, 4        //遞增idx
5.        lwc1 $f1, 0($5)       //載入weight
6.        addi $5, $5, 4        //遞增idx
7.        lwc1 $f2, 0($5)       //載入hidden
8.        mul.s $f3, $f2, $f1   //將weight與hidden相乘
9.        add.s $f4, $f4, $f3   //累加權值
10.       j acc
11.   exit:
12.       swc1 $f4, 4($4)       //輸出結果
```

Outline

- **Mips-Core**
  1. 系統規格
  2. Zynq交互方式
  3. Memory map
- **FC-DNN**
  1. 計算資源分配
  2. Mips-Core 計算方式
  3. Mips-Core 單顆神經元計算
     a) C code
     b) Mips code
- **Exercise**

Exercise

➢ 說明：

目前的範例程式，在計算同一層的神經元時，會重複寫入相同的input值。為了重複寫入相同的input值，導致資源浪費，須修改以下內容：

1. 同一層 layer 中，只進行一次 input 值寫入。

```
116   for (k = 0; k < NUM_OF_TEST; k++)
117   {
118       read_ppm_file(k);
119       for (idxz = 1; idxz < 3; idxz++)
120       {
121           for (idxi = 0; idxi < DIM[idxz]; idxi++)
122           {
123               tempVal = b_hidden[idxz - 1][idxi];
124
125               // 計算加權總和
126               *(io + (DM_BASE_OFFSET / 4)) = DIM[idxz - 1] * 2 * 4;
127               for (idxj = 1; idxj ≤ DIM[idxz - 1] * 2; idxj = idxj + 2)
128               {
129                   *(io + (DM_BASE_OFFSET / 4) + idxj) = *(unsigned int *)&w_hidden[idxz - 1][idxi][(idxj - 1) / 2];
130                   *(io + (DM_BASE_OFFSET / 4) + idxj + 1) = *(unsigned int *)&v_hidden[idxz - 1][(idxj - 1) / 2];
131               }
```

程式修改處

Exercise

➢ 實作步驟：

1. 將提供的zynq_wrapper.bit燒入至FPGA (燒錄流程詳見附錄)
2. 修改practice/software/mips_dnn.c
3. 將software資料夾壓縮成software.zip，並放入usb中
4. 將usb插上zedboard，並將software.zip解壓縮

   root@localhost:~# mount /dev/sda1 /mnt

   root@localhost:~# cp /mnt/software.zip ./

   root@localhost:~# unzip software.zip
5. 編譯並使用perf量測效能

   root@localhost:~# cd software

   root@localhost:~/software# gcc mips_dnn.c -O3 -o mips_dnn

   root@localhost:~/software# perf stat --repeat 1 -e cycles,instructions,cache-misses,cache-references ./mips_dnn

Exercise

➢ 預期結果

```
Accuracy:       0.980200  (9802 / 10000)

 Performance counter stats for './mips_dnn':

     204365990025      cycles
      16063238966      instructions
         89128277      cache-misses
       5062397317      cache-references

     306.765632130 seconds time elapsed
```

▲ 優化前，執行時間約300秒

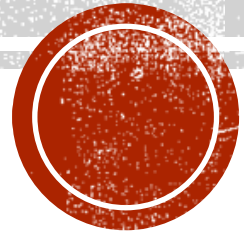
```
Accuracy:       0.980200  (9802 / 10000)

 Performance counter stats for './mips_dnn_new':

     151231904582      cycles
      13513509841      instructions
         79248613      cache-misses
       4584610584      cache-references

     227.141534841 seconds time elapsed
```

▲ 優化後，執行時間約225秒

➢ 驗收方式：

1. 給助教看優化後的Perf輸出結果，並且在Zedboard上維持98%準確率

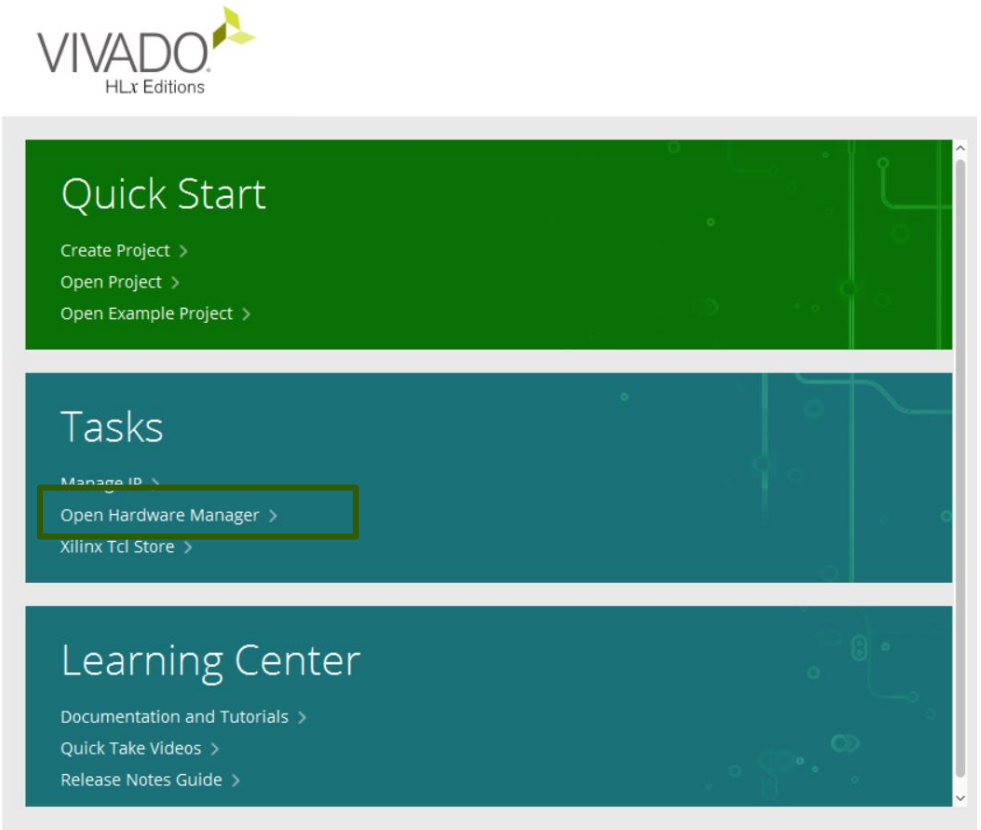
APPENDIX



Vivado 燒錄bitsream(1/6)

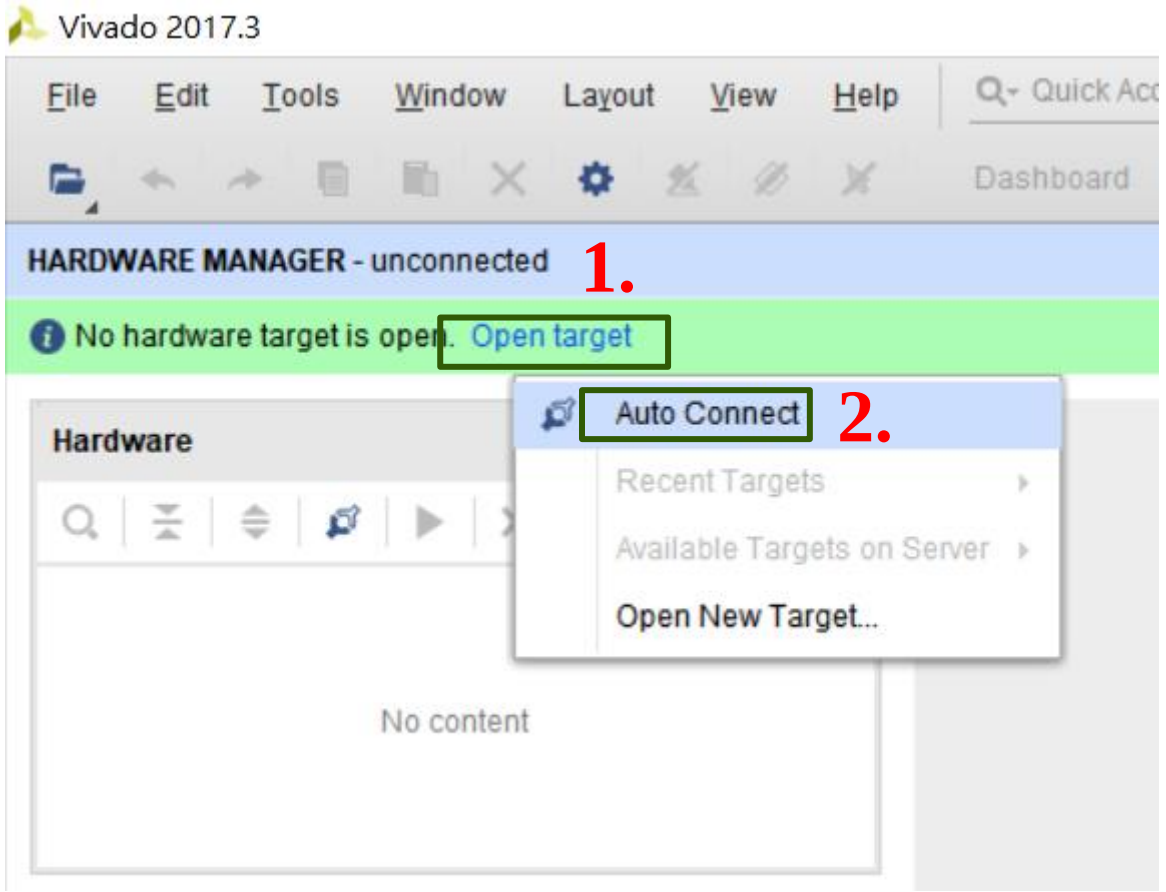
- 將Zedboard Jtag接口與PC用USB線連接：

- 開啟Vivado，選擇Open Hardware Mananger



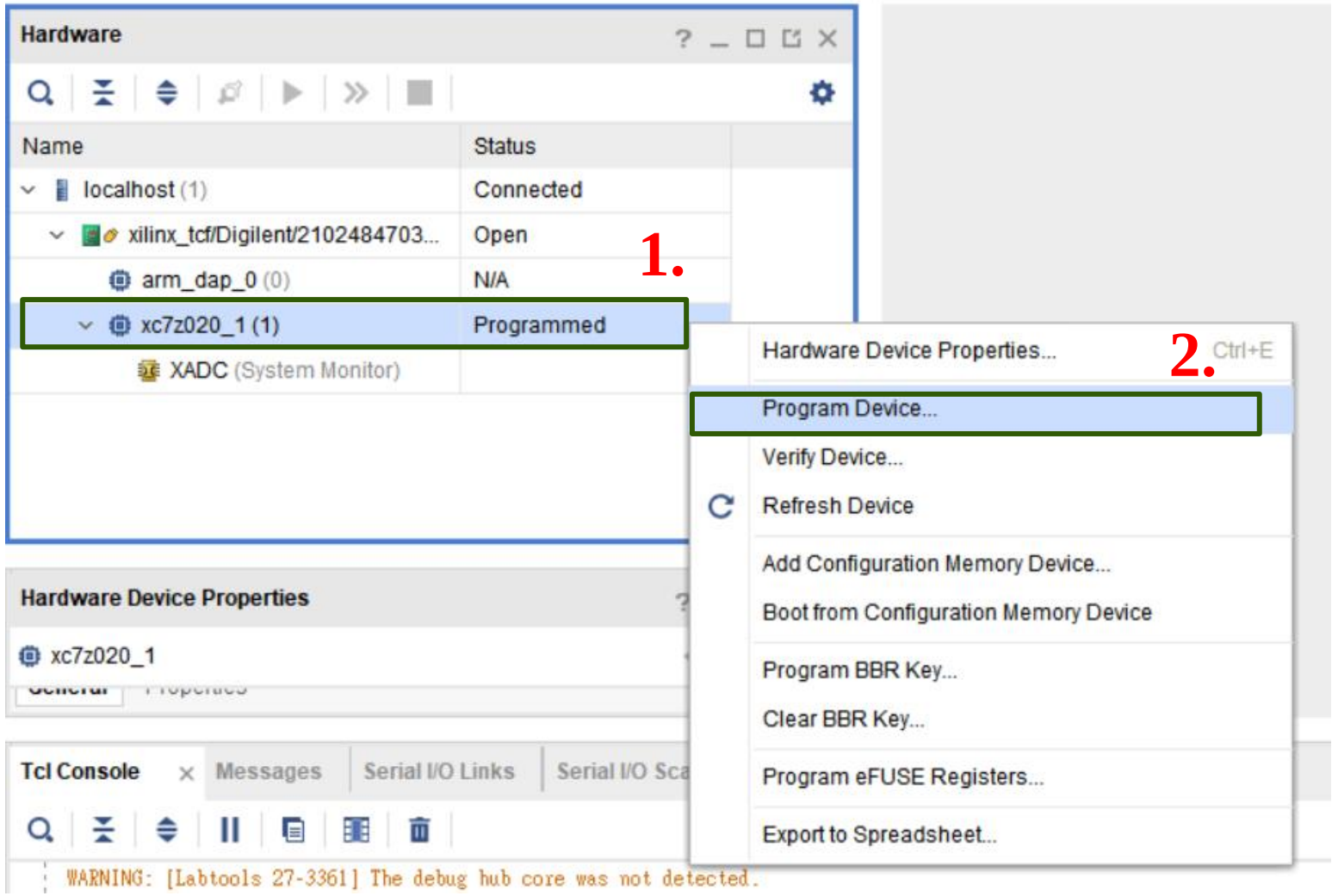
Vivado 燒錄bitsream(2/6)

- 將Zedboard Jtag接口與PC用USB線連接：



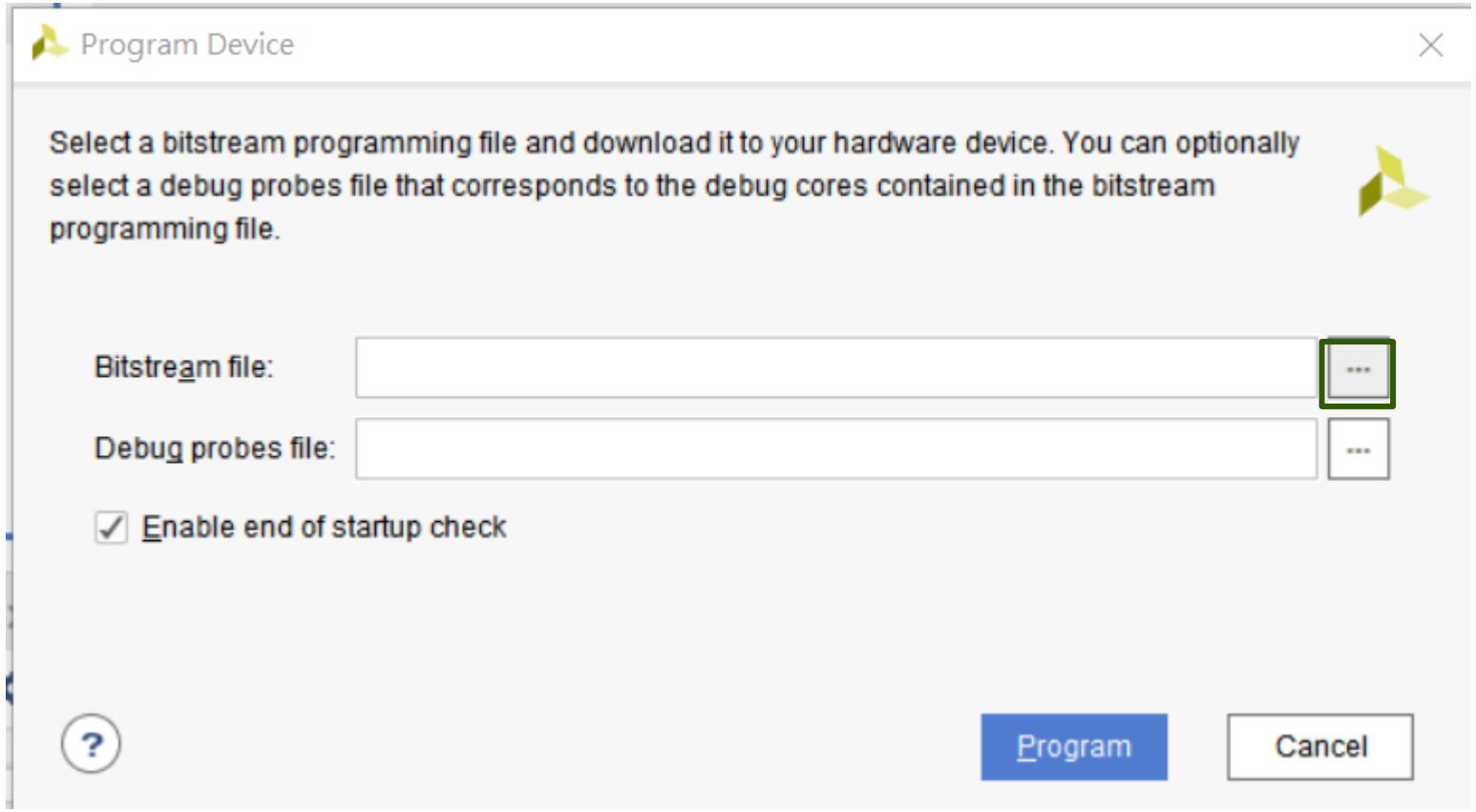
Vivado 燒錄bitsream(3/6)

➢ 對Programmed點右鍵，並選取Program Device...



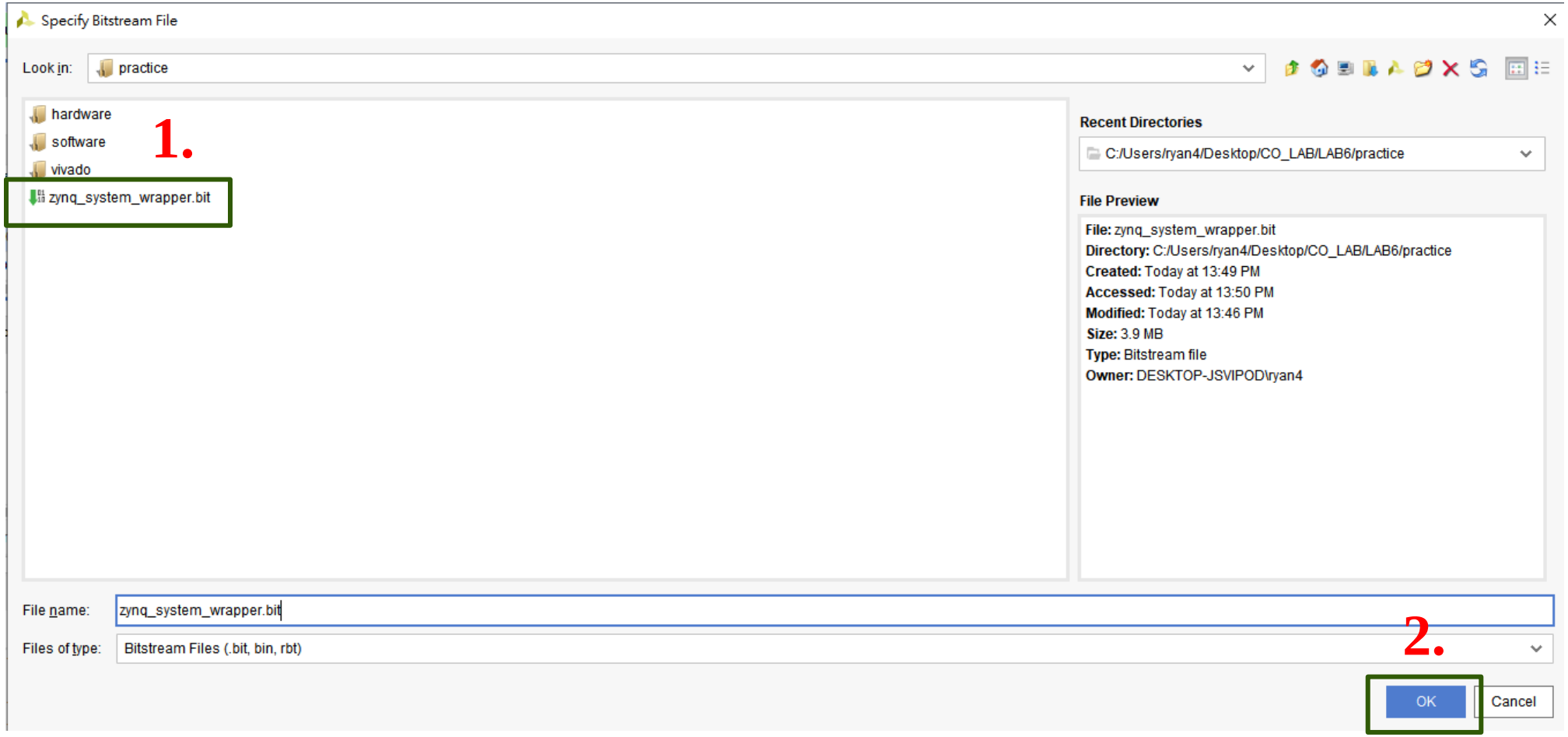
Vivado 燒錄bitsream(4/6)

➢ 點擊 “...”



Vivado 燒錄bitsream(5/6)

➢ 選取pratice/software/zynq_system_wrapper.bit，並按OK



Vivado 燒錄bitsream(6/6)

➢ 點擊Program