DE10-Nano 全方位教學

從硬體到軟體：板子 · 系統 · GHRD · 程式 · 文件

一份把「看得到的硬體」到「跑得動的軟體」串起來的完整教材

淡江大學電機工程學系　AI 機器學習實驗室

2026-07

淡江大學
Tamkang University

AI 機器學習實驗室
AI & Machine Learning Lab
淡江大學 電機工程學系

這份教材怎麼讀（學習地圖）

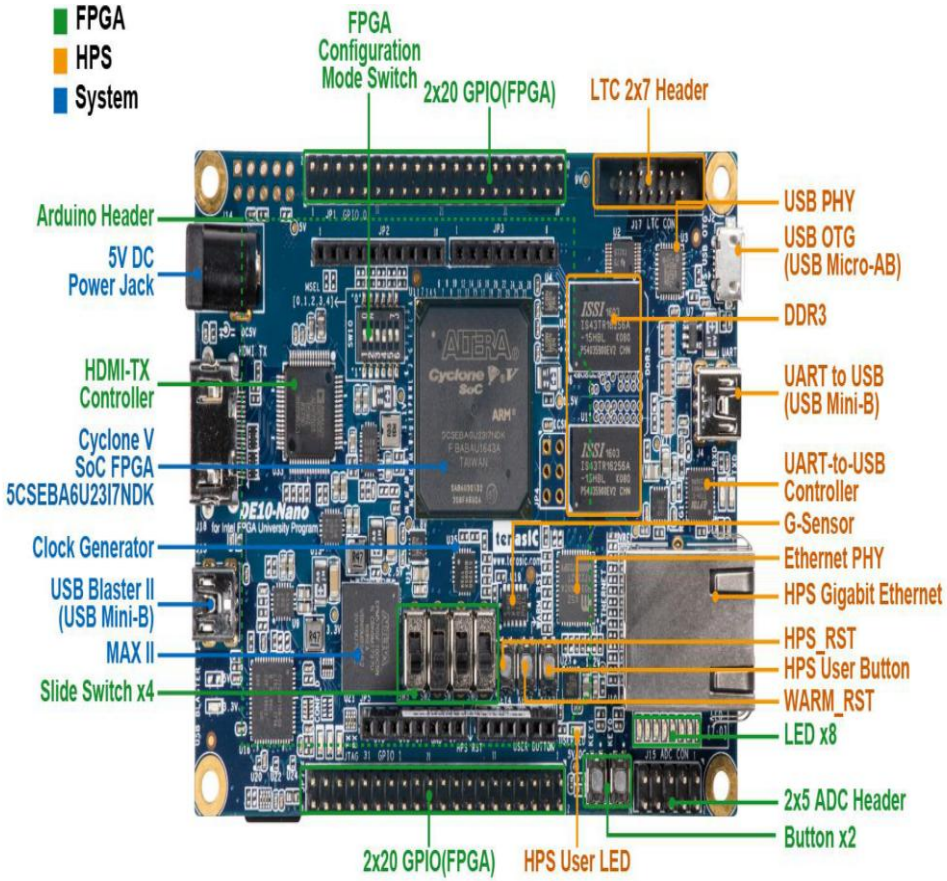
五個段落，從硬體一路到文件

- **① 硬體：認識板子、ARM 與 FPGA 兩個腦**
- ② 系統：SD 卡上的 Linux、Python、gcc、mmap、numpy
- ③ GHRD：FPGA 裡現成的設計，給 ARM 用的位址
- ④ 軟體：用 C + mmap 戳硬體、用 Python 做運算
- ⑤ 文件：每個資料在哪個檔、哪一頁查得到

認識板子：DE10-Nano 與製造商 Terasic

這塊板子是誰做的、規格是什麼

- **製造商 Terasic（友晶科技）**
  - 台灣新竹的公司，Intel(原 Altera) 官方開發板夥伴
  - 做 DE 系列教學/開發板；DE10-Nano 就是他們的產品
- **核心晶片：Cyclone V SE SoC（5CSEBA6）**
  - 一顆晶片裡同時有 ARM(HPS) + FPGA(可程式邏輯)
  - ARM：雙核 Cortex-A9 @925MHz + 1GB DDR3
  - FPGA：約 41,910 ALM / 112 DSP
- **出貨附：板子 + 燒好系統的 microSD 卡 + 手冊/範例(System CD)**



一塊板子，兩個腦：ARM 與 FPGA

最重要、最容易混淆的觀念

比較項目	ARM (HPS)	FPGA (Fabric)
是什麼	處理器，像一台小電腦	可重設線路的邏輯
跑什麼	Linux ＋ 軟體 (C/Python)	硬體設計 (Verilog→位元流)
這次誰主角	我們的程式跑這裡	用現成的 GHRD
怎麼溝通	透過「橋接」：ARM 用 mmap 讀寫 FPGA 的暫存器（位址 0xFF20_xxxx）	

✓ 本教材大部分在教「ARM 這顆」怎麼用；FPGA 那邊用現成的 GHRD，我們只去「戳」它。

官方板子方塊圖（看硬體全貌）

DE10-Nano User Manual 第 8 頁 · Figure 2-3

- 1 Gigabit Ethernet PHY with RJ45 connector
- port USB OTG, USB Micro-AB connector
- Micro SD card socket
- Accelerometer (I2C interface + interrupt)
- UART to USB, USB Mini-B connector
- Warm reset button and cold reset button
- One user button and one user LED
- LTC 2x7 expansion header

2.2 Block Diagram of the DE10-Nano Board

Figure 2-3 is the block diagram of the board. All the connections are established through the Cyclone V SoC FPGA device to provide maximum flexibility for users. Users can configure the FPGA to implement any system design.

Detailed information about Figure 2-3 are listed below.

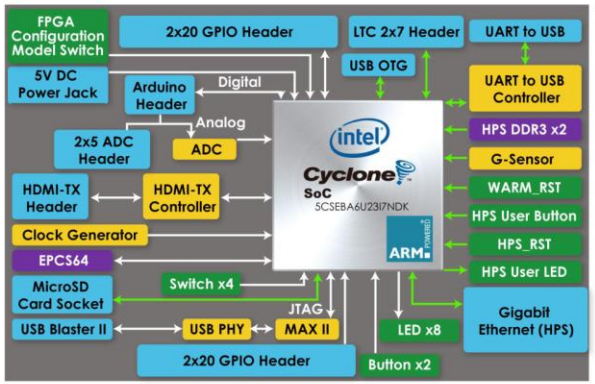
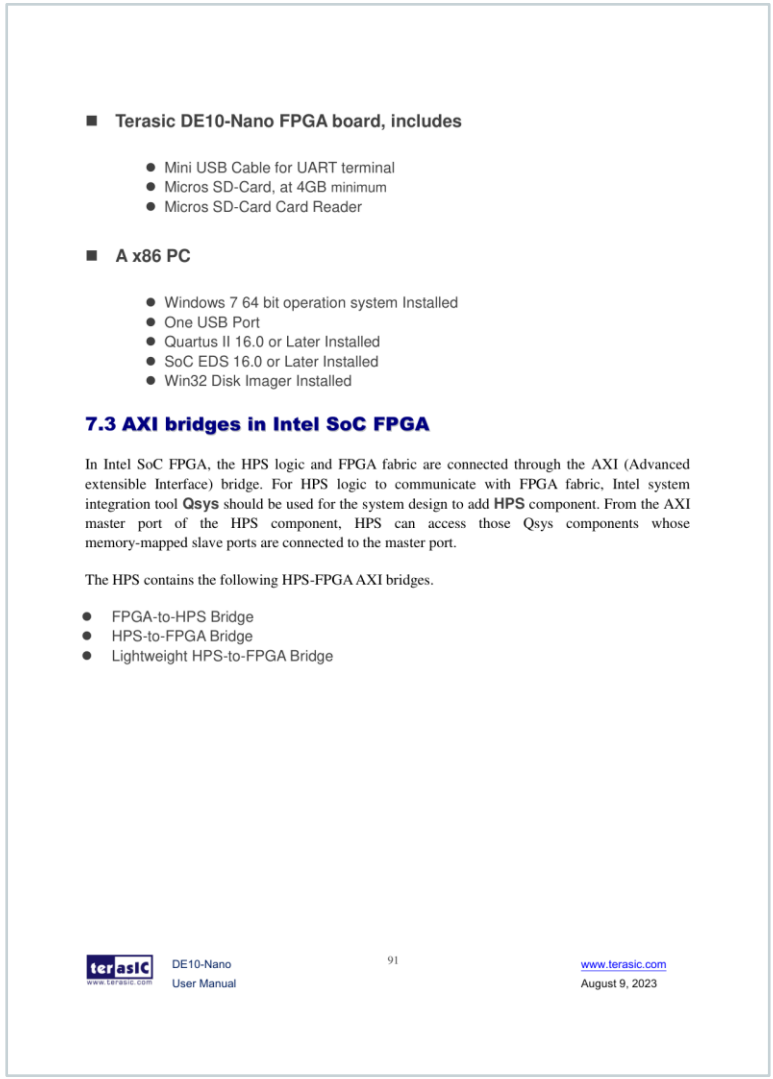


Figure 2-3 Block diagram of DE10-Nano

- **一張圖看懂板子上有什麼**
- 中央：Cyclone V SoC（ARM + FPGA）
- FPGA 側：LED×8、Switch×4、KEY、2×20 GPIO、ADC、HDMI…
- ARM(HPS) 側：Gigabit 乙太網、USB OTG、DDR3、micro SD、UART…
- 出處：DE10-Nano_User_manual.pdf 第 8 頁

ARM 與 FPGA 怎麼連：AXI 橋接

兩個腦中間的那座橋



■ **Terasic DE10-Nano FPGA board, includes**

- Mini USB Cable for UART terminal
- Micros SD-Card, at 4GB minimum
- Micros SD-Card Card Reader

■ **A x86 PC**

- Windows 7 64 bit operation system Installed
- One USB Port
- Quartus II 16.0 or Later Installed
- SoC EDS 16.0 or Later Installed
- Win32 Disk Imager Installed

7.3 AXI bridges in Intel SoC FPGA

In Intel SoC FPGA, the HPS logic and FPGA fabric are connected through the AXI (Advanced extensible Interface) bridge. For HPS logic to communicate with FPGA fabric, Intel system integration tool **Qsys** should be used for the system design to add **HPS** component. From the AXI master port of the HPS component, HPS can access those Qsys components whose memory-mapped slave ports are connected to the master port.

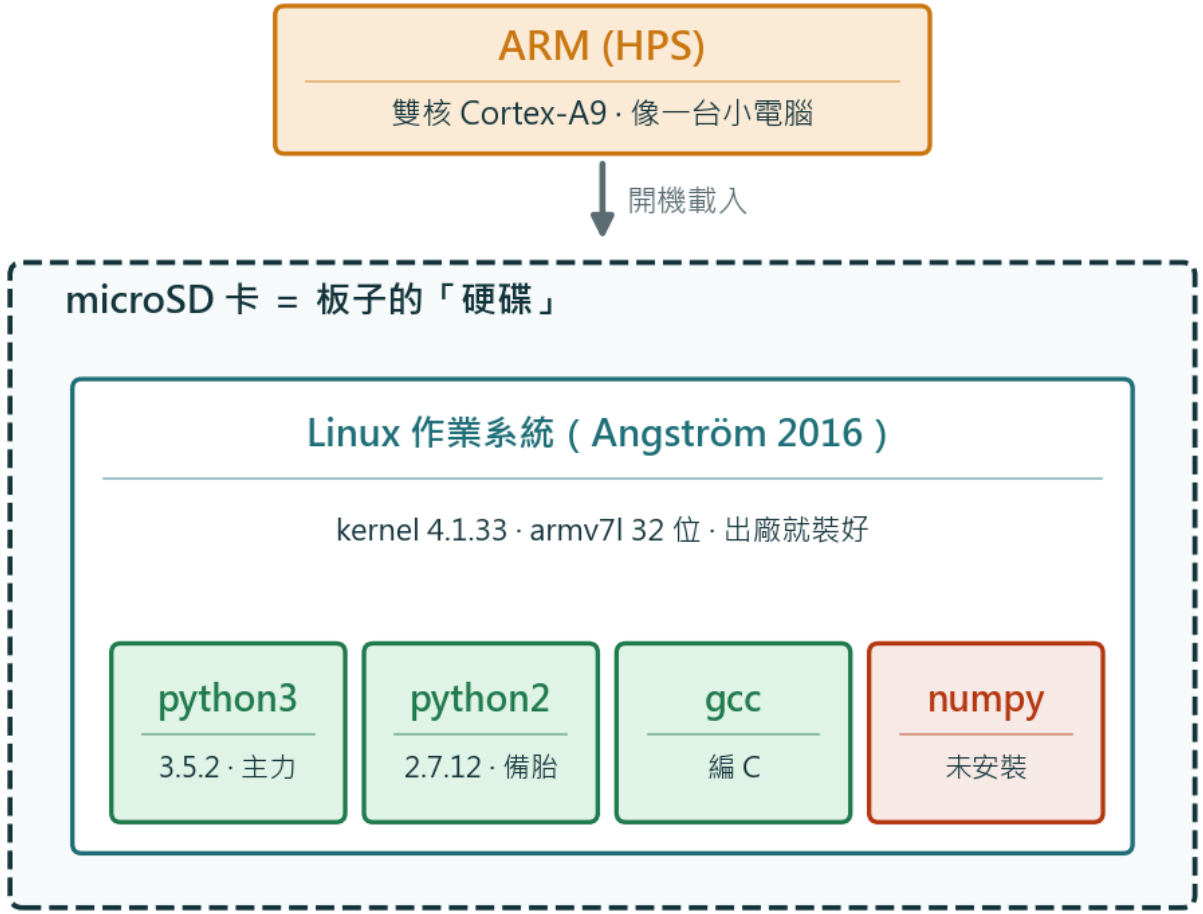
The HPS contains the following HPS-FPGA AXI bridges.

- FPGA-to-HPS Bridge
- HPS-to-FPGA Bridge
- Lightweight HPS-to-FPGA Bridge

- **ARM(HPS) 與 FPGA 之間有三種 AXI 橋**
- 輕量 HPS→FPGA 橋（lwh2f）：設定週邊，起點 0xFF200000
- HPS→FPGA 橋（h2f）：大量資料
- FPGA→HPS 橋（f2h）：FPGA 主動寫進 ARM 記憶體
- **我們戳 LED 走的就是「輕量橋」→ 位址都長 0xFF20_xxxx**
- 出處：User Manual 第 92 頁（7.3 AXI bridges）

軟體住在哪裡？SD 卡上的一整套系統

ARM 一開機就從 SD 卡載入 Linux



關鍵：整套系統（Linux + Python + gcc）都在 microSD 卡上 → 換卡 = 換整台系統、燒壞重燒卡就救回。

Linux 是什麼？誰擁有它？

板子的「作業系統」＋一個常見誤會

- **作業系統 = 管理硬體、讓程式能跑的底層軟體（像 Windows）**
- Linux 是其中一種：1991 年 Linus Torvalds 建立的「作業系統核心」
- **★ 沒有任何一家公司「擁有」Linux**
  - 開源(open source)：程式碼公開，任何人可自由使用 / 修改
  - 由全世界社群 ＋ Linux Foundation（非營利）共同維護
  - 「發行版」才可能有公司（Red Hat / Canonical…），但核心仍屬社群
  - 板子上的 Angström 也是社群做的、免費的發行版
  - 比喻：Linux 是引擎，發行版是不同廠牌的車（Ubuntu / Angström…）
- **它幫我們做：開機、管檔案記憶體、跑程式、連網路、SSH、提供 /dev/mem 讓 mmap 碰硬體**

Linux 為什麼「老舊」？夠不夠用？

老 ≠ 不能用

- **「老舊」指的是這個「映像檔的版本」，不是 Linux 本身老**
- Terasic 2016 打包好、燒進 SD 卡後就沒再更新 → 板子照著用就凍在那版
  - （現在的 Linux 早更新到很新版；只是這張卡沒跟上）

夠不夠用？→ 對我們的目標，完全夠

✓ 夠用的（正是我們要的）	▵ 老版的小代價
SSH / 跑 C(gcc) / 跑 Python / mmap 戳 FPGA / 架網頁 / SD 讀寫檔	python3 缺 mmap 模組（戳硬體改用 C）
辨識後段軟體、影像運算都跑得動	opkg 裝不到新套件、numpy 沒附

結論：能用、夠用；只有「想裝很多新套件」時才會卡——那時再處理（見下一頁）。

Linux 能不能換？建議換什麼？

能換，但比換 PC 系統複雜、有風險

- **能換嗎？→ 能。但要動到 bootloader + kernel + 根檔系統(rootfs) + FPGA 設定**
- 比在 PC 換 Windows/Ubuntu 難：弄錯會開不了機，要重燒卡救回

換法	說明	難度
官方較新映像	用 Terasic / RocketBoards 較新的 SD image（GSRD）	★ 低
換 Ubuntu rootfs	Cyclone V 可換 Ubuntu 根檔系統，套件新、好裝	★★ 中
自建 Yocto / Buildroot	自己編一版最新、最精簡、可完全掌控	★★★ 高

- **我的建議：非必要不換（現版對辨識目標夠用）**
- 真要換 → 先用「官方 RocketBoards 較新 GSRD / Ubuntu 映像」，別自己從零建
- 只是缺某個套件 → 優先「交叉編譯那個東西丟上板」，不必換整個系統

板子的系統是哪一版？能做什麼？

實機 SSH 連線實際查到的資料

```
Linux de10-nano 4.1.33-ltsi-altera #1 SMP armv7l GNU/Linux
```

- **發行版：Angström（精簡型嵌入式 Linux，2016 年版）**
- 核心：Linux kernel 4.1.33（2017）｜ CPU 架構：armv7l = 32 位 ARM
- 特性：小而省，專為「有限硬體的小裝置」設計；舊版、已凍結不更新

這台「小電腦」能做的事

✓ 被 SSH 遠端登入、下指令	✓ 跑 C 程式（gcc 編譯）	✓ 跑 Python 程式
✓ 連網路、架網頁伺服器	✓ 用 mmap 透過橋接戳 FPGA	✓ 在 SD 卡讀寫檔案

一句話：它就是「一台會跑 Linux、又能直接碰 FPGA 硬體的小電腦」。

Python（板子上有兩個版本）

好寫、直譯式，適合做運算/辨識

- **Python = 好讀好寫的程式語言；直譯式（直接跑，不用先編譯）**
- **板子實機查到：python3 = 3.5.2、python2 = 2.7.12（兩個都有）**

差別	Python 2（舊·已退役）	Python 3（主力）
中文編碼	易亂碼	Unicode，中文直接通
除法 5/2	= 2（砍小數，陷阱）	= 2.5（正常）
新套件	裝不到	都支援

- **結論：辨識、跑新東西一律用 python3**
- 它能做什麼：算數值、處理影像、跑辨識邏輯（讀圖片檔，不必碰硬體）

Python 2 vs 3：用小程式看差異

同一段程式，兩版結果可能不一樣（陷阱）

① print 寫法

```
# Python 2
print "hello"

# Python 3
print("hello")
```

② 除法 5/2（最陰險）

```
# Python 2
5 / 2   ->  2    (小數被砍!)

# Python 3
5 / 2   ->  2.5  (正常)
```

③ 中文字串

```
# Python 2：預設 bytes，處理中文容易亂碼、要自己 encode/decode
# Python 3：預設 Unicode，   s = "影像"   直接就對，中文不再踩雷
```

為什麼重要：辨識要算一堆數值（除法）、又常處理中文檔名 → 一律用 python3 才不踩雷。

小提醒：板子兩版都在，明確打「python3」指定版本（別只打 python）。

gcc：把 C 語言「翻譯」成 CPU 能跑的東西

編譯器（GNU C Compiler）

- **CPU 看不懂人寫的 C 文字 → gcc 把 .c 翻成「機器碼執行檔」**

流程：fpga_led.c →(gcc 編譯)→ fpga_led 執行檔 →(./fpga_led)→ LED 亮

- 板子上有 gcc（Linaro），所以能「在板子上編、在板子上跑」
- 注意：機器碼認 CPU → 在板子上編出的是 ARM 碼；PC 的 x86 碼不能直接搬上板

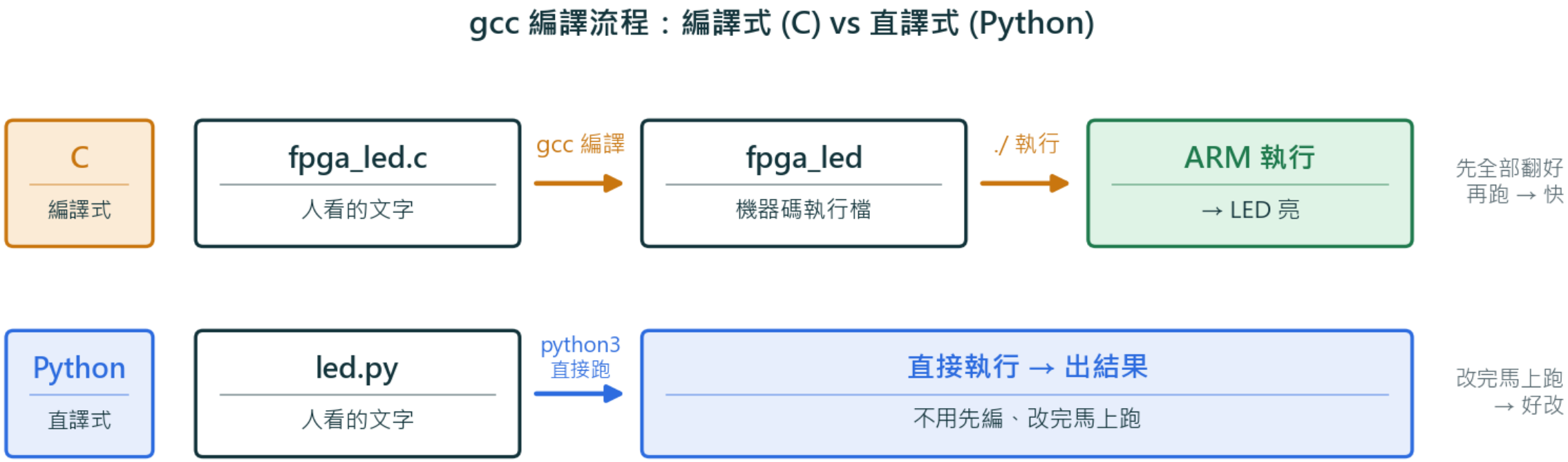
一秒看懂的類比（FPGA 人專屬）

你寫的設計	用什麼轉換	變成晶片能跑的
Verilog（硬體）	Quartus 合成	FPGA 位元流 (.sof)
C（軟體）	gcc 編譯	ARM 機器碼執行檔

兩個都是「把人寫的設計，變成矽晶片真的能執行的形式」。

編譯 vs 直譯：用流程圖看 gcc

為什麼 C 要先 gcc、Python 不用

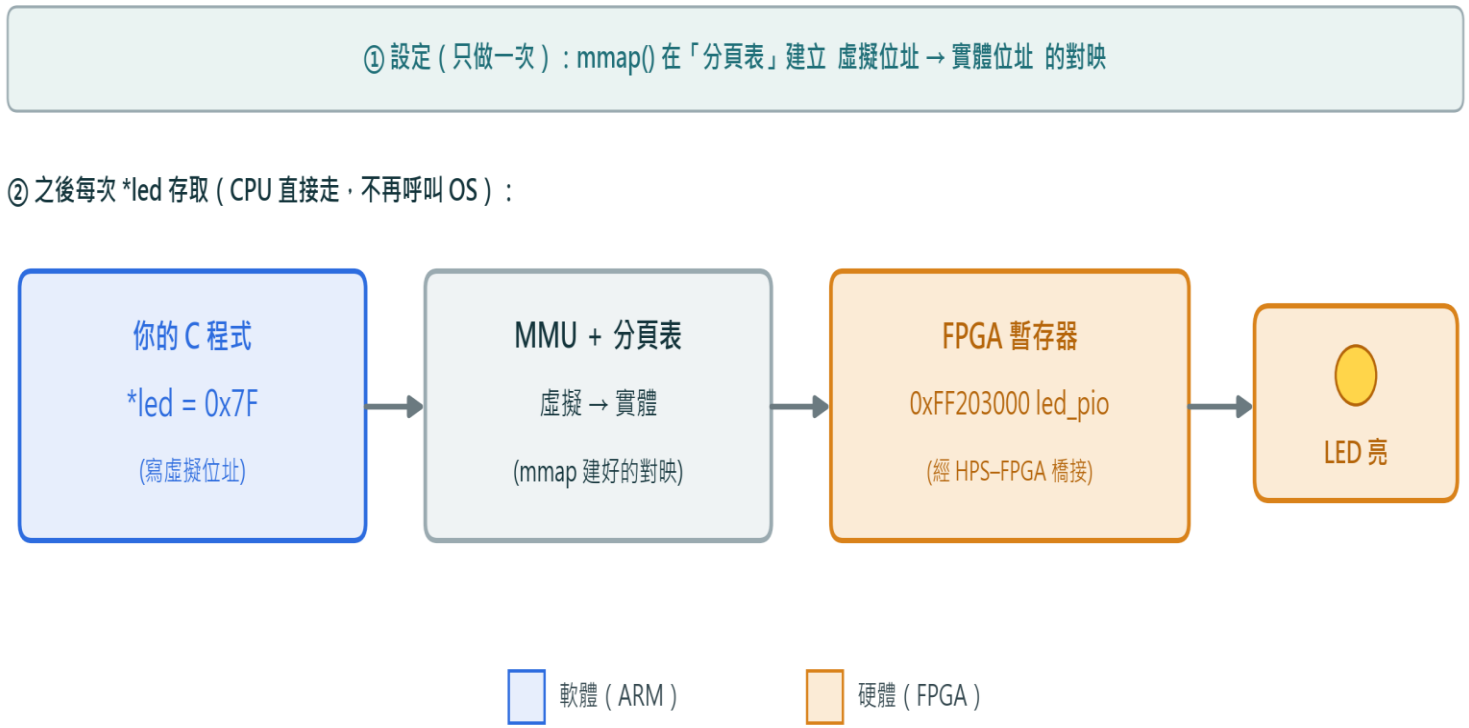


重點：C 改完「一定要再跑一次 gcc」重編才生效；Python 改完直接 python3 跑、免編譯。

mmap（全名 memory map / 記憶體映射）

用途：讓程式直接碰硬體位址的「窗口」——為什麼 *led = 0x7F 就能控硬體

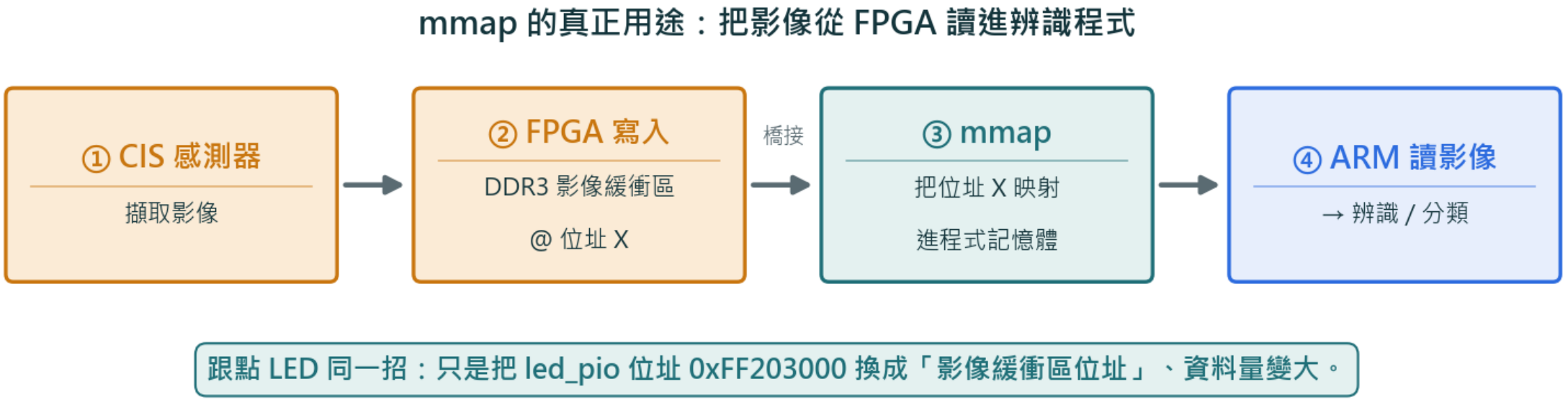
圖三　mmap：把「虛擬位址」對映到 FPGA 的「實體暫存器」



- **硬體暫存器有「實體位址」**
- 程式不能亂碰實體記憶體
- mmap 先建一條「虛擬位址 → 實體暫存器」的對映
- **之後寫那個虛擬位址，CPU 就直接寫進 FPGA 暫存器**
- 板子 python3 沒這模組 → 戳硬體用 C + mmap 最穩

mmap 的真正用途：把影像讀進辨識程式

從點 LED 到讀影像，同一套方法



- **今天練的 led_pio（0xFF203000）= 這條路的「練習版」**
- 真辨識時：FPGA 把 CIS 影像寫進 DDR3 → ARM 用 mmap 讀那塊記憶體 → 辨識
- 程式幾乎一樣，只是把「LED 位址」換成「影像緩衝區位址」、資料量變大

numpy（全名 Numerical Python / 數值 Python）

用途：Python 的「矩陣運算引擎」，做影像/辨識的常用工具

- **numpy = Numerical Python：讓 Python 會算「大矩陣/陣列」的套件**
  - 類比：numpy 之於 Python ≈ 矩陣運算之於 MATLAB
- 為什麼有用：一張影像就是一個大數字矩陣（每像素 R/G/B）
  - 算平均色、直方圖、比對…用 numpy「整張一次算完」，又快又短
  - 純 Python 要一格一格跑迴圈（慢）；numpy 一行搞定（底層 C，快幾十倍）
- **本機實機查到：numpy 沒裝（import 失敗）**
- 不卡關：辨識 demo 改寫「純 Python 版」（零依賴、慢一點但照跑）

```
# 純 Python（慢）            |   # numpy（快，但本機沒裝）
tot=0                        |    avg = img.mean()
for row in img:
    for px in row: tot+=px
```

整理：C + mmap 與 Python 各做什麼

戳硬體 vs 算辨識

板子上兩條路：各做各的

C + mmap → 戳硬體

點 LED / 讀開關 / 之後讀 CIS 影像暫存器

編譯式（gcc 先編成執行檔）→ 跑得快

碰實體位址，需要 mmap

+

Python → 算 / 辨識

影像分類 / 比對（讀圖片檔，不碰硬體）

直譯式（python3 直接跑）→ 好寫

本機沒 numpy → 用純 Python 版

實務上會合作：C + mmap 把影像從 FPGA 搬出來 → 交給 Python 做辨識。今天先各別練。

GHRD：FPGA 裡現成的「標準設計」

Golden Hardware Reference Design

7.4 GHRD Project

The term GHRD is short for Golden Hardware Reference Design. The GRD project provide by Terasic for the DE10-Nano development board is located in the CD folder: CD-ROM\Demonstration\SOC_FPGA\ DE10_NANO_SoC_GHRD.

The project consists of the following components:

- ARM Cortex™-A9 MPCore HPS
- Two user push-button inputs
- Four user DIP switch inputs
- Seven user I/O for LED outputs
- JTAG to Avalon master bridges
- Interrupt capturer for use with System Console
- System ID

The memory map of system peripherals in the FPGA portion of the SoC as viewed by the MPU starts at the lightweight HPS-to-FPGA base address 0xFF20_0000. The MPU can access these peripherals through the Address offset setting in the Qsys. User can open the GHRD project with Quartus II Software. Then open the soc_system.qsys file with the Qsys tool. Figure 7-2 lists the address map of the peripherals which are connected to the lightweight HPS-to-FPGA.

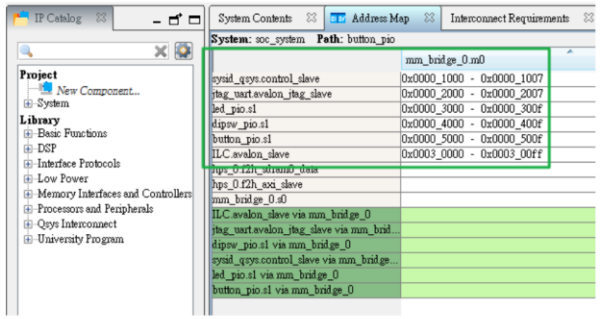


Figure 7-2 FPGA peripherals address map

- GHRD = Golden Hardware Reference Design
- Terasic 給的官方「標準範本」FPGA 設計；FPGA 開機預設就跑它
- **手冊列出它的組成：**
  - 2 個按鈕、4 個 DIP 開關、7 個 LED 輸出
  - JTAG 主橋、ILC 中斷計數、System ID
- **我們戳的 LED/開關，全是 GHRD 接好的（沒自己燒過 FPGA）**
- 出處：User Manual 第 94 頁（7.4 GHRD Project）

GHRD 給 ARM 的 5 個窗口（精確位址）

從 GHRD 的 soc_system.sopcinfo 抽出，並經實機驗證

模組名稱	位址	位元寬	用途
sysid_qsys	0xFF201000	—	FPGA 身分證（驗證載對設計）
jtag_uart	0xFF202000	—	JTAG 序列通訊
led_pio	0xFF203000	7	8 顆綠 LED（LED[7:1]）
dipsw_pio	0xFF204000	4	指撥開關 SW（輸入）
button_pio	0xFF205000	2	按鈕 KEY（輸入）

實機驗證： `SSH 讀 0xFF201000 → sysid = 0xACD51402`（與設計檔相符 → 確認板子跑的就是這 GHRD）

全部掛在「輕量橋接起點 0xFF200000」之後，每個間隔 0x1000。

位址怎麼來：0xFF200000 + 偏移

把上一頁畫成一張圖

GHRD 給 ARM 的位址圖（都從輕量橋接 0xFF200000 起算）

0xFF200000	輕量橋接起點	（基準）	
0xFF201000	sysid_qsys	FPGA 身分證	實機=0xACD51402
0xFF202000	jtag_uart	JTAG 序列通訊	
0xFF203000	led_pio	8 顆 LED 輸出	7 位 (LED[7:1])
0xFF204000	dipsw_pio	指撥開關 SW	4 位
0xFF205000	button_pio	按鈕 KEY	2 位

規則：絕對位址 = 輕量橋接起點 0xFF200000 + 該週邊的偏移（led_pio 偏移 0x3000 → 0xFF203000）。

led_pio 細節：hex → 二進位 → 哪幾顆亮

0xFF203000、7 位元

圖四　LED 值怎麼看：7 個 bit，每個 bit 控制一顆 LED（1 = 亮、0 = 暗）

hex 值	二進位（低 7 bit）	LED7	LED6	LED5	LED4	LED3	LED2	LED1	
0x7F	1 1 1 1 1 1 1	●	●	●	●	●	●	●	全部亮
0x55	1 0 1 0 1 0 1	●	○	●	○	●	○	●	交替亮 4 顆
0x2A	0 1 0 1 0 1 0	○	●	○	●	○	●	○	交替亮 3 顆
0x01	0 0 0 0 0 0 1	○	○	○	○	○	○	●	只亮 LED1
0x00	0 0 0 0 0 0 0	○	○	○	○	○	○	○	全部暗

hex 只是二進位的簡寫：led = 0x55 → 1010101 → LED 1、3、5、7 亮。改數字 = 改亮哪幾顆。

在位址 **0xFF203000** 寫一個 7 位元的數字 → 每個 bit 控一顆 LED（1 亮 0 暗）。0x7F = 全亮。

動手：用 C + mmap 點 LED

把前面觀念變成可跑的程式

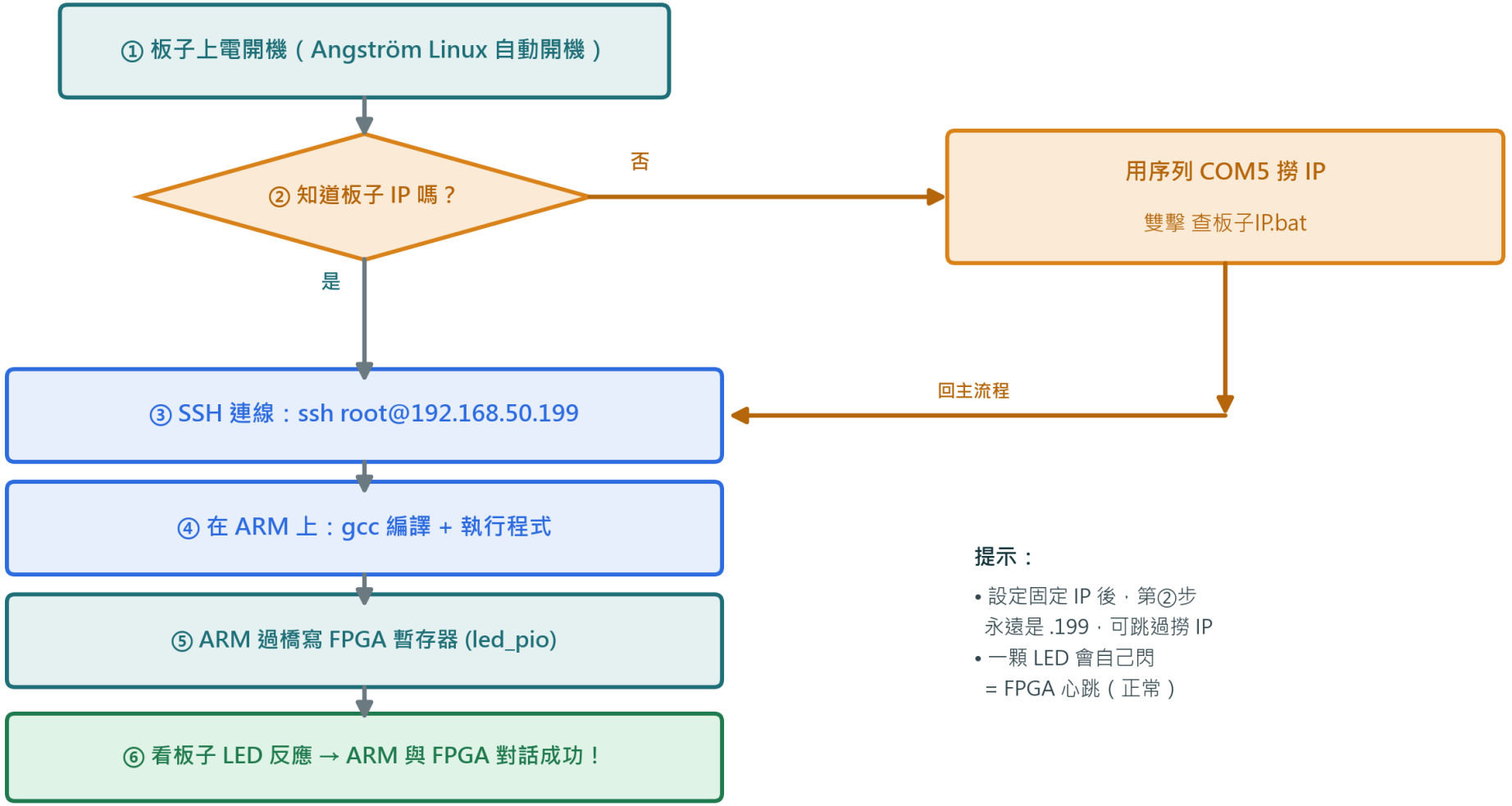
```
int fd = open("/dev/mem", O_RDWR | O_SYNC);        // 開實體記憶體
void *v = mmap(NULL, 0x04000000, PROT_READ|PROT_WRITE,
               MAP_SHARED, fd, 0xFC000000);        // 映射橋接區
volatile uint32_t *led =
        v + ((0xFF200000 + 0x3000) & 0x03FFFFFF);  // led_pio 位址
led[0] = 0x7F;                  // 寫 → 7 顆 LED 全亮 (ARM→FPGA)
printf("LED = 0x%02X\n", led[0]);   // 讀回確認
```

編譯 + 執行：gcc -O2 -o fpga_led fpga_led.c　然後　./fpga_led

- 換成 0xFF204000 就能讀 dipsw、0xFF205000 讀 button——同一套方法
- 把 0x7F 改 0x55、用迴圈移位，就是跑馬燈/呼吸燈

完整操作流程（照著做）

圖二　操作流程圖：從開機到讓 ARM 控制 FPGA



提示：

- 設定固定 IP 後，第②步永遠是 .199，可跳過撈 IP
- 一顆 LED 會自己閃 = FPGA 心跳（正常）

文件地圖：哪個資料在哪個檔/哪一頁

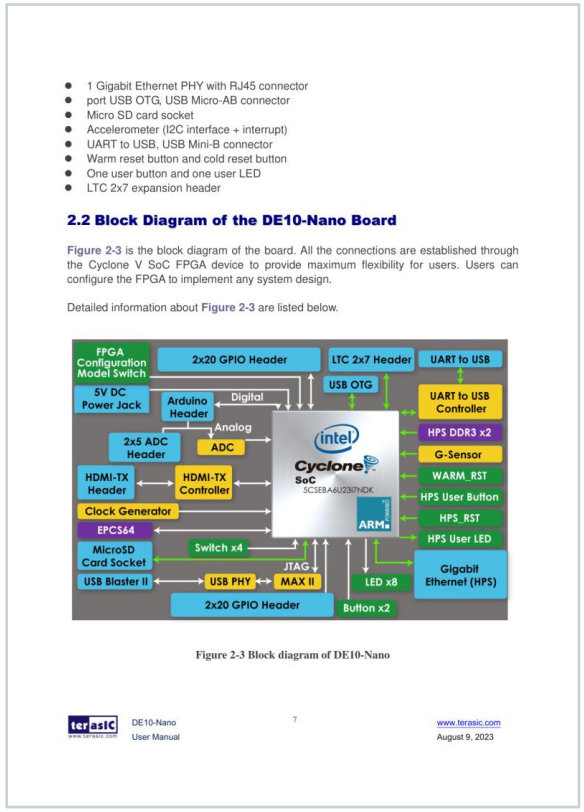
以後要自己查，看這頁

你想查的	看哪個檔案	位置
板子硬體全貌、接腳、按鈕/開關/GPIO	`FPGA\SystemCD\Manual\DE10-Nano_User_manual.pdf`	方塊圖 p.8、元件 p.6、LED/SW/KEY p.23、GPIO p.26
GHRD 是什麼、組成、位址映射圖	（同上 User Manual）	第 7.4 節　p.94
5 個週邊的精確位址（權威）	`DE10_NANO_SoC_GHRD\soc_system.sopcinfo`	純文字；或 .qsys 用 Platform Designer 開
led_pio 接到哪幾顆燈、對到哪支腳	`DE10_NANO_SoC_GHRD.v` + `.qsf`	top Verilog + pin assignment
用 C 存取週邊的範例	（User Manual 7.6 Develop the C Code）	p.96

GHRD 專案資料夾：`FPGA\SystemCD\Demonstrations\SoC_FPGA\DE10_NANO_SoC_GHRD\`

手冊重點頁速覽（截圖 + 頁碼）

要看細節就翻這幾頁



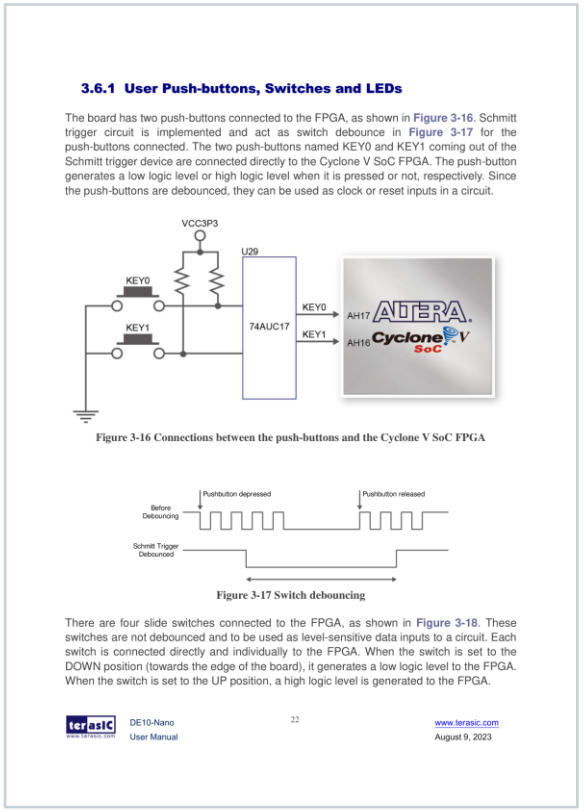
- 1 Gigabit Ethernet PHY with RJ45 connector
- port USB OTG, USB Micro-AB connector
- Micro SD card socket
- Accelerometer (I2C interface + interrupt)
- UART to USB, USB Mini-B connector
- Warm reset button and cold reset button
- One user button and one user LED
- LTC 2x7 expansion header

2.2 Block Diagram of the DE10-Nano Board

Figure 2-3 is the block diagram of the board. All the connections are established through the Cyclone V SoC FPGA device to provide maximum flexibility for users. Users can configure the FPGA to implement any system design.

Detailed information about Figure 2-3 are listed below.

Figure 2-3 Block diagram of DE10-Nano



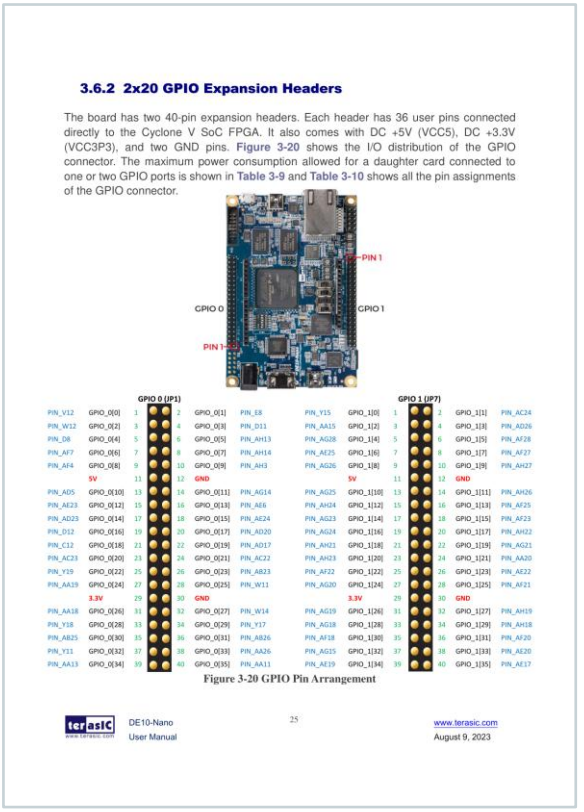
3.6.1 User Push-buttons, Switches and LEDs

The board has two push-buttons connected to the FPGA, as shown in Figure 3-16. Schmitt trigger circuit is implemented and act as switch debounce in Figure 3-17 for the push-buttons connected. The two push-buttons named KEY0 and KEY1 coming out of the Schmitt trigger device are connected directly to the Cyclone V SoC FPGA. The push-button generates a low logic level or high logic level when it is pressed or not, respectively. Since the push-buttons are debounced, they can be used as clock or reset inputs in a circuit.

Figure 3-16 Connections between the push-buttons and the Cyclone V SoC FPGA

Figure 3-17 Switch debouncing

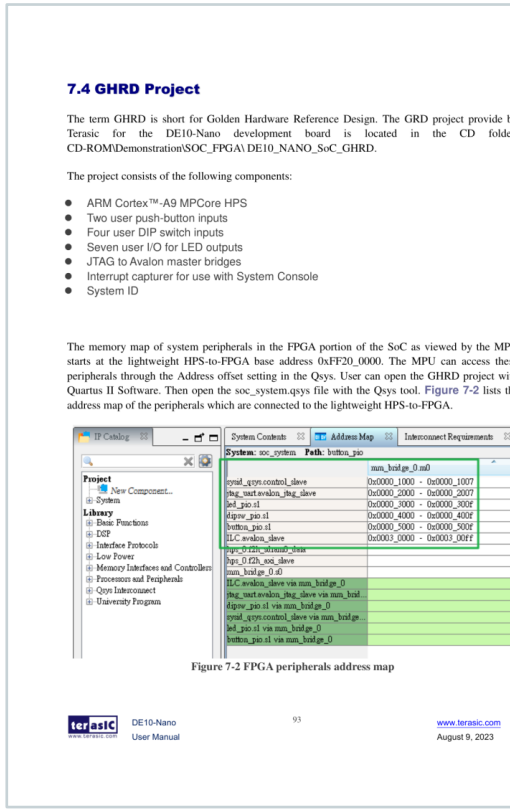
There are four slide switches connected to the FPGA, as shown in Figure 3-18. These switches are not debounced and to be used as level-sensitive data inputs to a circuit. Each switch is connected directly and individually to the FPGA. When the switch is set to the DOWN position (towards the edge of the board), it generates a low logic level to the FPGA. When the switch is set to the UP position, a high logic level is generated to the FPGA.



3.6.2 2x20 GPIO Expansion Headers

The board has two 40-pin expansion headers. Each header has 36 user pins connected directly to the Cyclone V SoC FPGA. It also comes with DC +5V (VCC5), DC +3.3V (VCC3P3), and two GND pins. Figure 3-20 shows the I/O distribution of the GPIO connector. The maximum power consumption allowed for a daughter card connected to one or two GPIO ports is shown in Table 3-9 and Table 3-10 shows all the pin assignments of the GPIO connector.

Figure 3-20 GPIO Pin Arrangement



7.4 GHRD Project

The term GHRD is short for Golden Hardware Reference Design. The GRD project provide b Terasic for the DE10-Nano development board is located in the CD folde CD-ROM\Demonstration\SOC_FPGA\ DE10_NANO_SoC_GHRD.

The project consists of the following components:

- ARM Cortex™-A9 MPCore HPS
- Two user push-button inputs
- Four user DIP switch inputs
- Seven user I/O for LED outputs
- JTAG to Avalon master bridges
- Interrupt capturer for use with System Console
- System ID

The memory map of system peripherals in the FPGA portion of the SoC as viewed by the MP starts at the lightweight HPS-to-FPGA base address 0xFF20_0000. The MPU can access the peripherals through the Address offset setting in the Qsys. User can open the GHRD project wi Quartus II Software. Then open the soc_system.qsys file with the Qsys tool. Figure 7-2 lists th address map of the peripherals which are connected to the lightweight HPS-to-FPGA.

Figure 7-2 FPGA peripherals address map

p.8 板子方塊圖　　p.23 LED/開關/按鈕　　p.26 GPIO 腳位　　p.94 GHRD + 位址圖

全部在 DE10-Nano_User_manual.pdf（共 111 頁）；GHRD 原始碼在 SystemCD 的 DE10_NANO_SoC_GHRD 資料夾。

這一切對「後段辨識」的意義

把今天學的接到真正的目標

- **今天學會：板子兩個腦、SD 卡上的 Linux/Python/gcc、用 C + mmap 戳 FPGA、GHRD 的位址**

前段（FPGA 團隊）	→ 橋接 →	後段（本案）
CIS 影像感測器 掃描圖片、存進記憶體	mmap /dev/mem 0xFF20_xxxx	ARM 軟體 讀影像 → 影像分類 / 辨識

下一步：把今天的「led_pio 位址」換成「影像緩衝區位址」，同一套 mmap 方法就能把影像讀進辨識程式。

所以你今天點的 LED，就是之後「讀影像」的預習版——一模一樣的方法。

常見問題（一）：系統與操作

同學最常問的

Q 板子沒接網路，能用嗎？

A 能。用序列線（Micro-USB → COM5）直接連進去，不靠網路；這也是沒設 IP 時找 IP 的方法。

Q 為什麼用 root？安全嗎？

A 開發方便才用 root。root 無密碼「只限區域網路」，別把板子直接對外網開放即可。

Q 程式 / 檔案重開機會不會不見？

A 看放哪：放 /home、SD 卡上的會留著；放 /tmp 的是記憶體，重開就消失。

Q 我改了 C 程式，要重編嗎？

A 要。C 是編譯式 → 改完一定再跑一次 gcc 重編才生效；Python 改完直接跑、不用編。

常見問題（二）：硬體與辨識

把觀念接到真正的目標

Q 我會不會把 FPGA 寫壞？

A 不會永久壞。別在「正在跑 Linux」時 JTAG 燒「純 FPGA」設計（會害重開）；真弄亂了，重燒卡 / 回 GHRD 就好。

Q Python 為什麼戳不了硬體？

A 這台 python3 沒編進 mmap 模組；戳硬體（碰實體位址）要用 C + mmap（或有 mmap 模組的 python2）。

Q 32 位 armv7 有什麼限制？

A 套件要用對架構（armhf）的版本，例如 numpy / TFLite 要 ARM 版；單一程序記憶體定址也有上限。

Q 兩個腦怎麼合作做辨識？

A FPGA 擷取影像存進記憶體 → ARM 用 mmap 讀出來 → 用 C / Python 做辨識。led_pio 就是這條路的練習版。

總結：一張表記住全部

層	一句話記住
硬體	一塊板子兩個腦：ARM(小電腦) + FPGA(可程式硬體)，中間靠橋接
系統	SD 卡 = 硬碟；上面是 Angström Linux(2016)，內含 python3/2、gcc，無 numpy
GHRD	FPGA 現成設計；給 ARM 5 個窗口，led_pio=0xFF203000(7位)
軟體	C + mmap 戳硬體（快）；Python 做運算/辨識（好寫，本機用純 Python）
文件	硬體看 User Manual(p.8/23/94)；位址看 sopcinfo/qsys；接腳看 .v/.qsf
板子固定網址： `ssh root@192.168.50.199` | `http://192.168.50.199:8080`	

全方位教材完 · 有問題隨時問