2025年臺灣國際科學展覽會
優勝作品專輯

作品編號　100035

參展科別　工程學

作品名稱　基於LS1043A多核處理器之嵌入式系統開機速度之精準計時方案

得獎獎項　四等獎

就讀學校　臺北市私立靜心高級中學
指導教師　呂正基
作者姓名　張正楷

關鍵詞　嵌入式系統、韌體 (Firmware)、Bootloader

作者簡介

我是靜心高中的張正楷。我對電動車及其智能電子系統充滿興趣，曾因試乘無人車而開始研究其核心技術，如LS1043A晶片等。在參與黑客松培訓中，我學習了智能傳感器、微控制器等處理器平台的應用，了解它們在車輛智能化中的關鍵作用。這段經歷不僅提升了我對電子系統的理解，我希望未來能以專業知識推動智慧交通技術發展。

研究報告封面

2025 年臺灣國際科學展覽會

研究報告

區別：

科別：工程學科

作品名稱：基於 LS1043A 多核處理器之嵌入式系統開機速度之精準計時方案

關鍵詞：嵌入式系統、韌體 (Firmware)、Bootloader

編號：

摘要

電子智能系統被廣泛應用在現代人的日常生活中，但是使用任何電子系統都必須面對系統失效的風險，就像我使用電腦可能會當機一樣。在智能車輛的應用中，電子系統的可靠性與我們的人身安全直接相關。一旦發生系統失效的狀況，恢復系統正常運行的最後一個手段是重新啟動系統，藉由系統重啟來恢復系統的正常功能。因此系統重啟的耗時，對系統的可靠性至為重要。

LS1043A 晶片是多核心高階處理器，能夠運行完整的 Linux 操作系統。由於 LS1043A 架構及功能的複雜性，它的開機程序需要遵守嚴格的步驟，耗時也比一般的微控制器更長。在這個實驗中，我將了解 LS1043A 處理器的架構著手，由此設計一個計時器，能做到精準量測開機時間達千分之一秒，使開機時間成為能夠量化的指標。本實驗的計時方案是一個通用的設計概念，可用來量測不同類型高階處理器的開機耗時及系統可靠性評估之參考指標。

Abstract

Smart electronic systems have been widely applied in our daily lives. While making our lives easier, it also brings risk to us as all electronic systems can potentially fail during operations. In the use case of a smart vehicle, the reliability of the electronic systems plays a major factor to safety. When a system fails, rebooting or system reset is usually the ultimate way to recover the system. Thus, the reliability of a system is often determined by how quickly it can be rebooted.

This experiment is made to realize an accurate timer that is capable of measuring the complete boot time of LA1043A processor platform to a precision in millisecond. LS1043A is a high-end multicore processor that is capable of running a full Linux operation system. LS1043A follows a strict multi-phase boot process due to its architectural complexity as all other high-end processors. The implementation of boot time measurement from this experiment can be commonly applied to all high-end processor platforms, and also can serve as an effective tool in benchmarking the reliability in all high-end electronic systems.

壹、 前言

一、研究動機

我參與了一個半導體企業所舉辦的黑客松培訓計畫，在這個培訓課程中，講師和我們介紹了各種不同類型的處理器平台。以電動車的智能電子系統為例，根據負責功能的不同，所使用到的處理器，由簡單到複雜就可以約略分為智能傳感器（Sensors)、微控制器（Micro Controllers)、多媒體處理器（Media Processor）以及車內通訊處理器(Communication Processor)。

在考慮到車輛的應用場景時，講師提到系統開機速度以及系統啟動時間，對車用系統可靠性有直接的影響，因為系統開機時間越短，就可以越快向使用者提供服務。微控制器因為架構簡單、功能單一，開機後就能夠立刻發揮功能，所以十分符合車用場景的需求。但是車輛上更加複雜的多媒體以及通訊系統，就需要工程師透過大量的系統優化，來達到快速開機的效果。

這句話引起了我的興趣，低階與高階處理器的區別在哪裡？為何高階處理器會需要更長的開機時間？所需的開機時間為何可以被縮短？為此我進一步請教了培訓的專家，所以有了這篇科展的構想。培訓講師建議我可以從認識LS1043A這款晶片著手。本晶片是一款多核心高階通訊處理器，能夠運行完整的Linux操作系統。由於LS1043A架構及功能的複雜性，它的開機程序需要遵守嚴格的步驟，耗時也較長。

從工程師的角度來看，開機速度不能只是使用者的感覺及經驗，而需要是一個能夠被量化的指標。因此我想要設計一個計時器，能做到精準量測開機時間，並且希望該計時器的精度能達千分之一秒。

二、研究目的

我要設計一個計時器，能做到精準量測開機時間，並且希望該計時器的精度能達千分之一秒，使開機時間成為一個能夠被量化的指標，用來衡量基於LS1043A所開發的各種平台開機速度之快慢。本實驗的成果也可以用於促進電子系統的可靠性分析，以及進一步縮短高階嵌入式系統開機時間之基礎。本實驗將研究以下問題：

(一)高階處理器是如何開機啟動的？

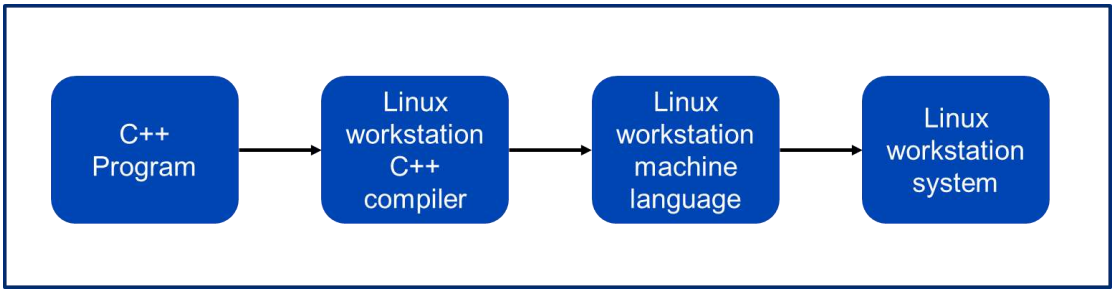
(二)如何使開機時間成為一個可量化的指標？

(三)要求本計時器方案的時間精度達到千分之一秒。

貳、研究方法

衡量開機速度的快慢不能只是憑藉使用者的感覺或經驗，而是需要建立一個能夠量化的指標。因此我想要設計一個計時器，能做到精準量測開機時間，使開機時間成為一個能夠量化的指標。LS1043A 是一款被廣泛運用在網路通訊及車用領域的晶片，該款晶片是我這次想要量測的目標晶片。LS1043ARDB 是基於這款晶片所設計的參考開發平台，我可以藉由這個平台來操作 LS1043A 晶片的各種功能。如果想要在該平台上自行開發需要的新功能，就必須開發新軟體來實現這個功能。而我這次想在 LS1043ARDB 平台上開發的功能，是一個用來測量及記錄開機時間的計時器。

為了開發新軟體，我學習了一些撰寫 C++程式語言的基礎知識，在這個階段的準備工作中，我需要對開發撰寫軟體及編譯軟體具備基本的概念，這部分的學習我參照了參考資料[9] Programming and Problem Solving with C++，對軟體撰寫及編譯的基本操作概念如下圖二之一所示。



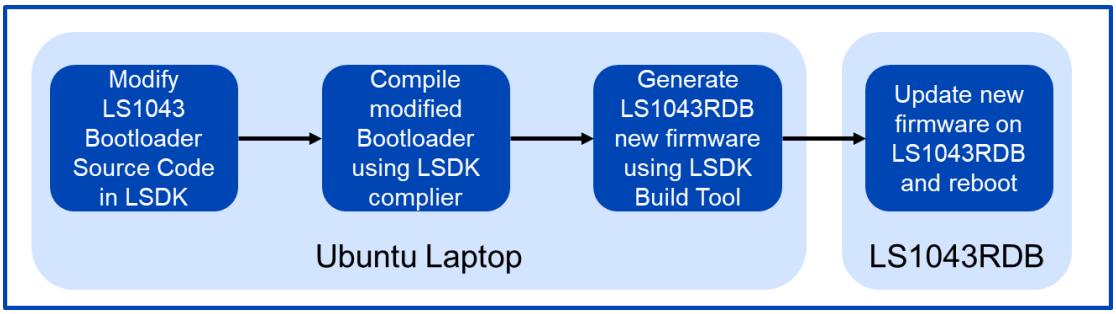
圖二之一、軟體撰寫及編譯的操作概念圖(自行繪製)

根據圖二之一所示，為了開發一項新的軟體功能，第一個步驟必須將程式代碼完整的寫出來。第二步驟需要將寫出的完整程式碼（source code）輸入 C++編譯器中重新編譯。第三步驟要將重新編譯好的機器碼（machine code）更新到目標平台上。最後將目標平台重新啟動，以確認我們所要開發的新功能可以在系統中正確運行。接下來我要思考，我要如何將以上概念落實在具體的實驗環境當中。

為了設計我的實驗環境，我需要閱讀並研究 LS1043A 這款晶片的相關技術文件，包含技

術文件［1］到［8］，閱讀這些文件的目的是為了瞭解 LS1043A 的晶片廠商提供了那些設計資源，我將運用既有的設計資源來實現圖二之一的操作環境，以下是具體的研究方法。

若是將本實驗的實驗內容對應到上述四個步驟的話，第一步驟需要將測量 LS1043ARDB 開機時間的程式碼成功撰寫出來。第二步驟需要將撰寫好的程式碼輸入進 LS1043A Software Development Kit（LSDK）開發環境中進行編譯。第三步需要將 LSDK 編譯好的機器碼（firmware）更新到 LS1043ARDB 上。最後將 LS1043ARDB 重新開機，並確認新增的計時器功能能夠正確測量開機時間。以上四個步驟必須落實在基於 LS1043ARDB 平台的完整實驗環境中進行。我將本實驗的實驗環境及實驗方法表示為下圖，圖二之二。



圖二之二、實驗環境及實驗方法表示圖(自行繪製)

參、研究設備及器材

一、硬體設備

實驗設備分述如下，見圖三之一。

（一） QorlQ LS1043A Reference Design Board

（二） ASUS Vivobook X1405ZA（負責記錄的電腦）

（三） ASUS UX305L 筆記型電腦/ 充電線(安裝 Ubuntu 系統的電腦)

（四） AC ADAPTER Model: NBS12C120150VU

（五） RJ45 轉 RS232 轉接線/Console 控制線(BRS0150FC)

（六） USB to RS232 單埠轉換器 FTDI 晶片 WIN11

（七） USB Drive（製作 Ubuntu 開機隨身碟）

（八） RJ45 網路線(綠色的線)

（九） WiFi 網路機

		
（一）QorlQ LS1043A Reference Design Board(自行拍攝)	（二）ASUS Vivobook X1405ZA（負責記錄的電腦）(自行拍攝)	（三）ASUS UX305L 筆記型電腦/充電線(安裝 Ubuntu 的電腦)（自行拍攝）
		
（四）AC ADAPTER Model: INPUT: 100-240V 50/60HZ 0.6A OUTPUT: 12.0V 0.6A(自行拍攝)	（五）RJ45 轉 RS232 轉接線/Console 控制線(BRS0150FC)（自行拍攝）	（六）USB to RS232 單埠轉換器 FTDI 晶片 WIN11(自行拍攝)
		
（七）USB Drive（製作 Ubuntu 開機隨身碟）(自行拍攝)	（八）RJ45 網路線(綠色的線)（自行拍攝）	（九）WIFI 網路機(自行拍攝)

圖三之一、實驗設備圖

二、軟體設備

(一) VenToy downloading Ubuntu 20.04

(二) UBUNTU 20.04 LTS Focal Fossa

(三) PuTTy on Ubuntu 20.04

(四) Terminal on Ubuntu 20.04

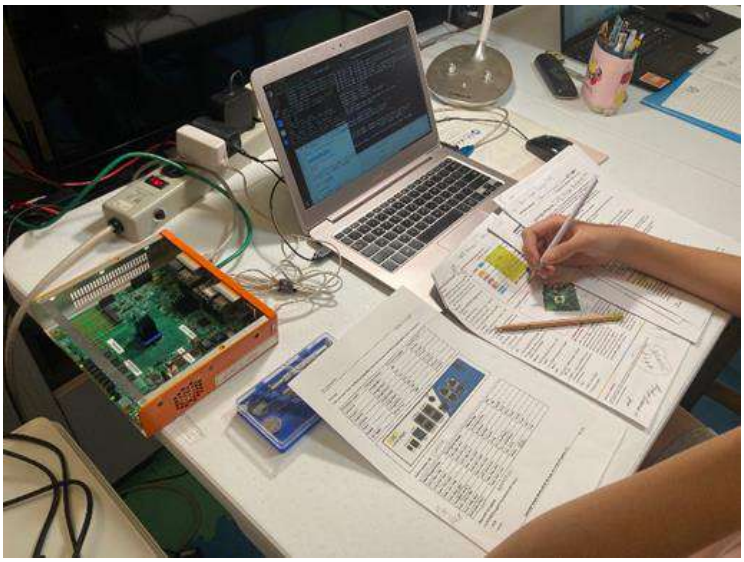
(五) LS1043A 處理器軟體開發工具（LSDK）

(六) TFTP 服務器

(七) Kate（text editor）

三、組裝實驗設備

下圖三之二為實際的實驗開發環境圖片，左上角橘色的盒子就是 LS1043ARDB，電腦就是 Ubuntu 系統的電腦，綠線為網路線。



圖三之二、組裝完成的實驗環境圖(自行拍攝)

肆、 研究過程

整個研究過程共分為四個階段，分別敘述如下。

一、架設並驗證 LS1043ARDB 平台及實驗環境的可靠性。

二、架設並驗證 LSDK 開發環境的可靠性。

三、設計 LS1043ARDB 的開機速度之精準計時方案。

四、開發並驗證 LS1043ARDB 的開機速度之精準計時方案。

以下分述實驗詳細過程。

一、架設並驗證 LS1043ARDB 平台及實驗環境的可靠性

當我拿到 LS1043ARDB 的當下，我感到錯愕，因為它不具備鍵盤及螢幕，跟我所熟悉的電腦完全不同，我完全不知道如何與之互動。為了學習使用 LS1043ARDB，我開始閱讀它的使用手冊 [1]，才了解到我需要近一步架設周邊的實驗環境，如此才能經由該實驗環境去操控 LS1043ARDB 平台，以下是實驗環境的架設流程。

(一)安裝實驗用電腦主機

依照 Ubuntu 20.04.6 LTS (Focal Fossa) 提供的安裝手冊安裝電腦。我需要安裝 Ubuntu 20.04 電腦的主要原因是因為，實驗所需的軟體開發環境 (LSDK)，要求安裝在 Ubuntu 20.04 電腦上，詳細安裝過程參照附錄一。

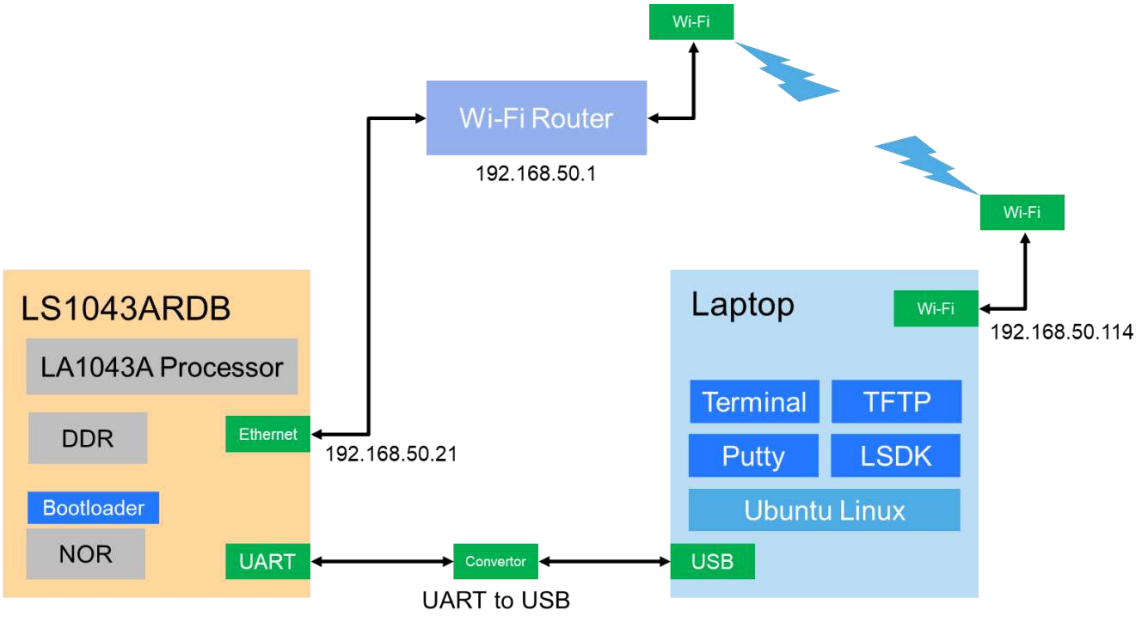
(二)架設並驗證實驗環境中的管理通道

為了達到透過 Ubuntu 20.04 主機去操控 LS1043ARDB 平台的目的，我需要建立起兩者之間的溝通機制。透過運用 RS232 Console 控制線再轉接成 USB to RS232 單埠轉換器，將 LS1043ARDB 平台連接到 Ubuntu 20.04 電腦，以此來建立兩者之間的管理通道。透過管理通道，可以命令 LS1043A 去下載新的韌體並更新韌體 (Firmware)。詳細過程參見附錄二。管理通道的驗證已經在附錄二中完成，詳見附錄二中如何利用 Putty 介面，對 LS1043ARDB 發送各類指令並進行操作，這裡不再贅述。

(三)架設並驗證實驗環境中的數據通道

為了達到 Ubuntu 20.04 主機及 LS1043ARDB 平台間數據可以互通的目的，我需要建立起兩者之間的數據通道。透過數據通道以太網 Ethernet 連接到 WiFi Router，數據通道讓 LS1043ARDB 透過網路去訪問 Ubuntu TFTP server，下載 LS1043ARDB 上新的 Bootloader 版本。詳細過程參見附錄二。本實驗的數據通道包含有線的以太網 Ethernet 以及無線的 WiFi 協議，數據的溝通須透過相同的 IP 網域，使 LS1043ARDB、WiFi Router 及 Ubuntu 主機之間能夠互相傳輸數據。數據通道的架設方法見附錄三。

總結本實驗的完整開發環境包括 LS1043ARDB、Ubuntu 主機，以及兩者之間的管理通道與數據通道，我需要架設的實驗環境簡圖如下圖四之一所示。在後續的實驗中，我將透過操作基於 Putty Serial Condole 的管理通道，要求 LS1043RDB 對 TFTP server 發出存取檔案的請求，並通過以太網連結的數據通道來下載檔案。通過這樣的操作，我可以同時驗證管理通道，以及數據通道的正確性，並同時驗證整個實驗環境的可靠性。數據通道的驗證詳見附錄四。



圖四之一、本實驗完整開發環境簡圖(自行繪製)

以上圖四之一是我所設計的 LS1043A 完整實驗開發環境，若將圖四之一與上文圖二之二實驗方法表示圖進行對比，我將從圖四之一的 Ubuntu Laptop 中的 LS1043A Software Development Kit (LSDK) 取出現成的程式碼並對其進行小部分的修改。將開發好的程式碼輸入圖四之一中的 LSDK 開發環境中進行編譯。接著利用管理通道及數據通道對 LS1043ARDB 進行 Firmware 的更新。最後將 LS1043ARDB 重新開機確認新開發的計時功能可以正確測量開機時間。

二、架設並驗證 LSDK 開發環境的可靠性

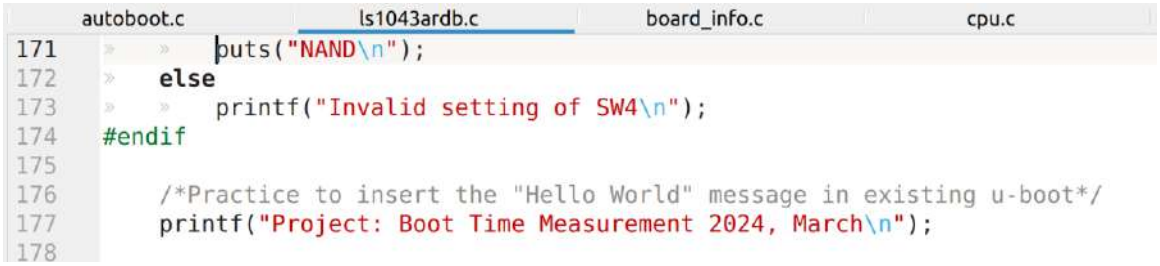
LS1043A Software Development Kit（LSDK）是晶片廠商提供的完整開發環境，其中包含我需要修改的 Bootloader 開機程序的原始代碼、軟體的編譯環境（compiler）和相對應的使用手冊及文件。完整的 LSDK 環境，可以從晶片廠商的網站上註冊後免費下載，見參考資料 [3]。

下載 LSDK 後，接著要在 Ubuntu 主機上架設 LSDK 開發環境。LSDK 開發環境可以用來測試修改 LS1043A Bootloader 原代碼，並重新編譯 Bootloader，以產生新的 Bootloader Firmware。要測試如何修改 LS1043A Bootloader 原代碼的原因，是因為需要確認 LSDK 是否可以成功編譯原代碼，以便在後續實驗中，運用相同的軟體開發和編譯程序來新增計時器功能，以及顯示計時器的數值。

重新編譯後的韌體，可以在上節所介紹的完整實驗開發環境中（如上圖四之一所示），由 Ubuntu 主機更新至 LS1043ARDB 平台上，達到更新韌體的目的。本部分實驗的挑戰，在於如何確保更新後的韌體，能夠成功並正確的啟動開機程序。本部分實驗內容參照參考資料 [1] 的內容。詳細實驗過程紀錄於下。

(一)測試修改 LS1043A Bootloader 原代碼

在這一部份的實驗當中，運用 Kate (text editor)來修改 LS1043A 處理器的 Bootloader source code 原始代碼。並使用 LSDK 重新編譯，產生新的 Bootloader image。LS1043A 處理器將透過 TFTP 服務器下載新的 Bootloader image，並更新到 NOR Flash 地址“0x60000000”。首先運用 Kate（text editor）來修改 LS1043A 處理器的 Bootloader source code 原始代碼。我在原始檔案 ls1043ardb.c 原代碼的第 177 行增加了 “Project: Boot Time Measurement 2024, March” 這一行字。如下圖四之二所示。



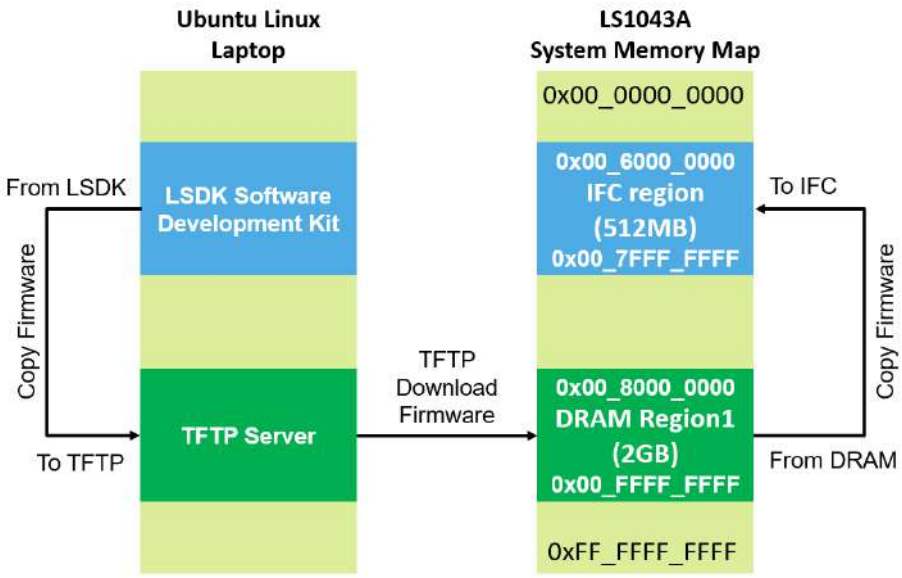
圖四之二、修改 LS1043A Bootloader 原代碼(自行拍攝)

接著在 LSDK 開發工具中，對更新過的 Bootloader 代碼進行編譯。LSDK 開發工具使用的是 FlexBuild 的編譯指令。我參考了 LSDK 使用手冊進行編譯，如下圖四之三所示。

圖四之三、對修改過後的 Bootloader 原代碼進行重新編譯(自行拍攝)

(二)更新 LS1043ARDB 上的 Firmware

接續前章節，將編譯完成後的新的 Bootloader image，放在 Ubuntu TFTP 服務器中，讓 LS1043A 處理器來下載至 DRAM。完成下載後，LS1043A 處理器再將新的 Bootloader image 從 DRAM 更新至 IFC NOR Flash 中，如下圖四之四所示。在驗證這一部份的功能前，需要對於 LS1043A 處理器的 System Memory Map[5]有初步的概念，了解如何將 IFC NOR Flash 在 LS1043 System Memory 中定址，之後 LS1043 晶片才能利用這個特定地址去存取 Flash 中的內容。我將這部分的學習材料，放在附錄五。在學習 LS1043ARDB 使用手冊 [1] 後，我總結 LS1043ARDB 的韌體更新程序如下圖四之四所示。



圖四之四、LS1043ARDB：更新 Firmware 流程圖(自行繪製)

三、設計 LS1043ARDB 的開機速度之精準計時方案

我們的目的是量測開機所需要的時間。從按下開機鍵的瞬間到 Linux 作業系統完成啟動為止，是我們所定義的開機時間。計時的方法有非常多種，例如手錶或是碼表。但是我們不選用手錶、碼表來計時，因為上述方法皆無法精準計時。開機計時的時間精度要求到千分之一秒，而手動計時的方式誤差太大，已失去計時的意義。

因此我們需要研究 LS1043A 內部的功能，運用其內部的計時功能，在開機時同步啟動計時功能，使用內部計時器的好處是他的精度很高，可以達到百萬分之一秒以上的精度，足夠滿足千分之一秒的精度要求，方便我們精準計算開機所需的時間。我通過了以下五個步驟來實現並控制開機計器：

(一)認識 LS1043A 重啟，時鐘及初始化

(二)認識 Pre-Boot Loader (PBL) 的數據結構

(三)認識 LS1043A System Memory Map

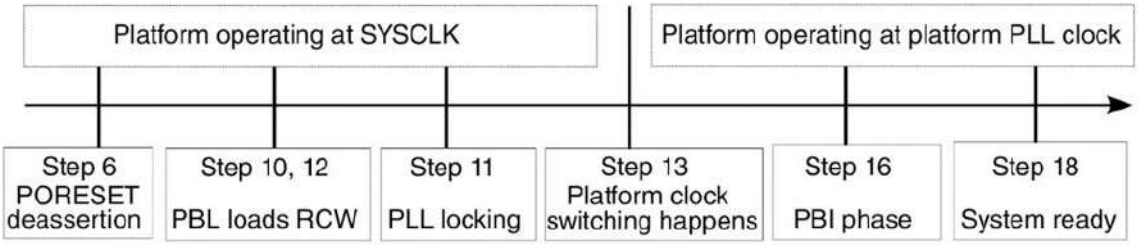
(四)設計啟動 EL1 Secure physical Timer 計時器

(五)設計 Pre-Boot Initialization (PBI) 指令以啟動計時器

以上第（四）及第（五）步驟是本實驗的研發重點。我將以上五個步驟的內容分述於下。

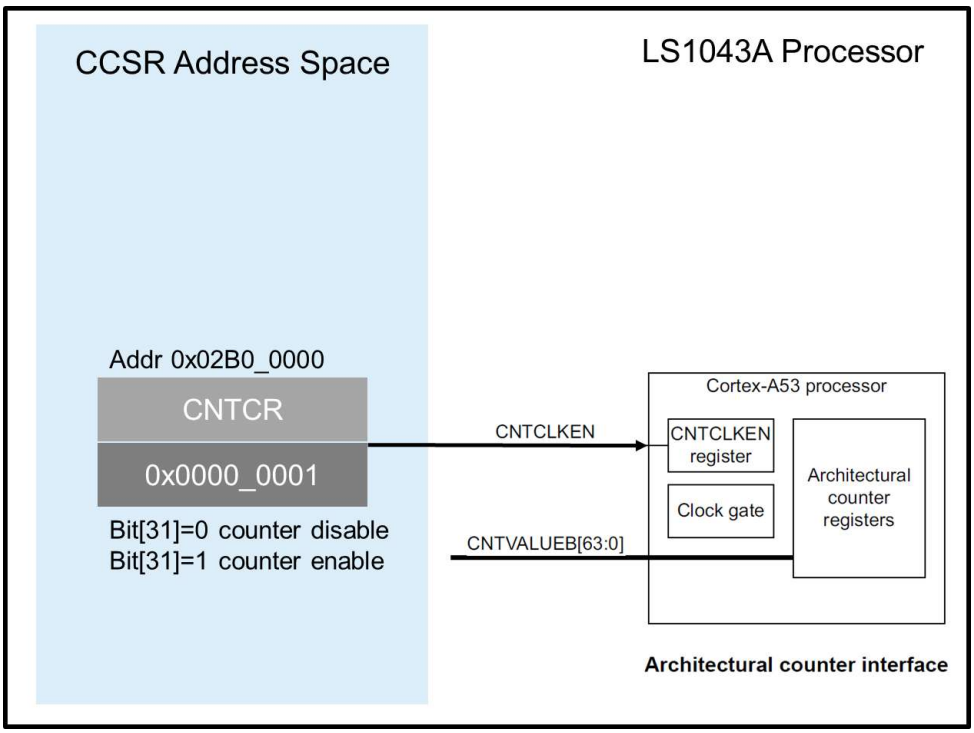
(一)認識 LS1043A 重啟，時鐘及初始化

首先我們需要了解 LS1043A 的開機步驟[5]，最理想的設計是在按下開機鍵的那一瞬間開始計時，如下圖四之五 Step 6，但是 LS1043A 晶片在開機早期是無法修改或是增加內容的，因此最早的計時時機在於 PBI 啟動的階段，如下圖四之五 Step 16，因為那時我們才可以藉由 PBI 指令開啟 LS1043A 內部計時器的計時功能，Step18 System Ready 完成後，晶片才開始執行 Bootloader（u-boot）以及啟動 operation system (Linus)，所以在 step 16 開啟計時器，可以完整的紀錄系統啟動時間。如何啟動計時器與啟動哪個計時器是我接下來的研究內容。



圖四之五、LS1043A 開機啟動步驟時序圖(自行拍攝)

我請教了培訓的專家，他們建議我使用的計時器的名稱是 EL1 Secure physical Timer，這個計時器在 Cortex-A53 processor 內部[6]。要開啟這個計時器，可以透過特別設計的 PBI 指令去更改 CNTCR Register，對 CNTCR 寫入特定的值 0x0000_0001[5]，以開啟 EL1 Secure physical Timer 計時器的計時功能，系統計時器的開啟及關閉之控制方法如下圖四之六所示。



圖四之六、系統計時器開啟及關閉之控制方法(自行拍攝)

本實驗目的是計算開機所需時間，透過改寫開機程序，在開機過程中盡可能及早啟動內部計時機制，即可盡量完整紀錄啟動時間長度。如圖四之五所示，因為 Step 16 是開機後最早能夠啟動計時器的時機，所以我們需要學習 PBI 指令的使用方式，如此才能夠使用 PBI 指令去啟動計時器。

(二)認識 Pre-Boot Loader (PBL) 的數據結構

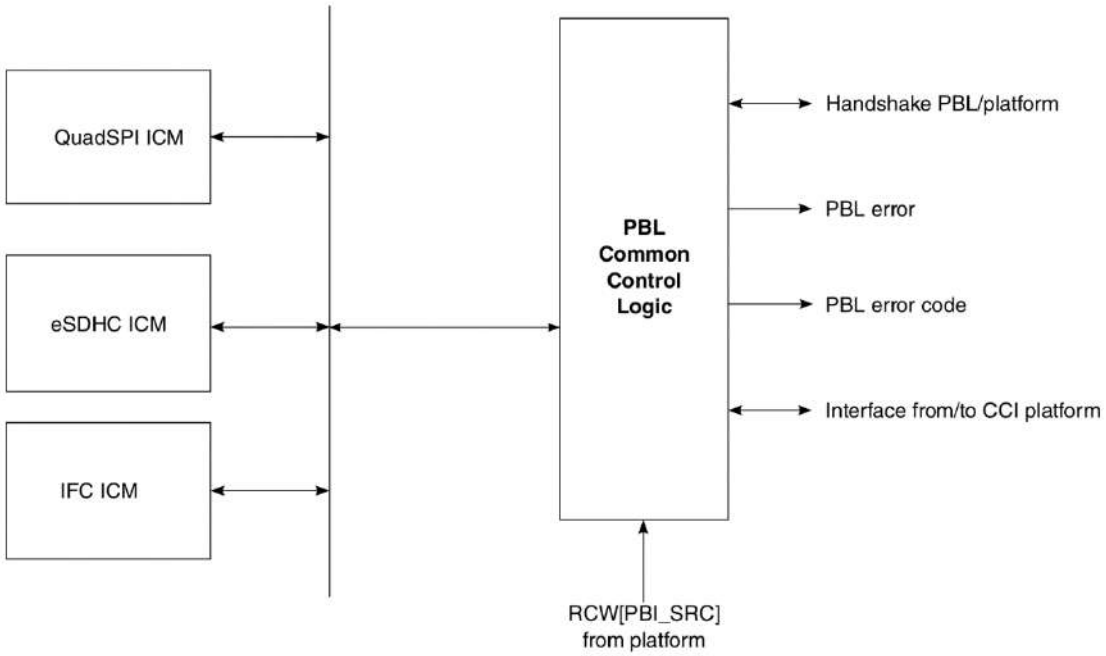
本硬體設備 LS1043ARDB 運用 IFC NOR Flash 開機，所以 NOR Flash 內部要有 PBL 的 Data Structure，而 PBL 的 Data Structure 又包含 Reset Control Word (RCW)，所以開機時 LS1043ARDB 要透過 IFC 去讀取 NOR Flash 內部的 PBL，其中就包含 RCW。

LS1043A 需要 PBL 是因為其結構較複雜，開機處理器配置選項的設定選擇多達 20、30 個，無法透過硬體設定完整回答，因為開機時沒有工作的腳位不夠去回答所有的配置選項，故需要 PBL。

硬體設定為在開機的瞬間去讀取 LS1043A 特定腳位的電壓數值高低並轉化為 0 跟 1，但

後續要變回正常狀態使用。開機時可使用是因為那時腳位還沒有工作，所以可以先把腳位變換工作，但後續還是要變回來否則處理器無法運作。

PBL 負責回答 Boot Rom 發出的選擇問題，選項配置是根據運用時的實際情況來回答，也就是存在 PBL 內部的答案。LS1043A 從 IFC NOR Flash 讀取 PBL 後，將讀取到的 RCW 內容存入 LS1043A 內部寄存器，Reset Control Word Status Register (DCFG_CCSR_RCWSRn)，參照 QorIQ LS1043A Reference Manual, Rev 6 [5]，參照下圖四之七。



圖四之七、PBL 讀取 RCW 流程控制圖(自行拍攝)

在此，為了確認以上的機制，我們可以檢查 1043 晶片內部的 DCFG_CCSR_RCWSRn 所儲存的 RCW 的值，與 NOR Flash 內的 RCW 值是否相同，若結果相同的話，就代表開機後 1043 晶片有正確的讀取到 NOR Flash 內的 RCW 值。下圖四之八所示為 PBL 及 RCW 數據結構圖[5]，PBL 的起始前四個 Bytes 為規定數值 (preamble) 0xAA55AA55。接著後四個 Bytes 也是規定的數值 0x01EE0100，其中 0xEE0100 指向 DCFG_CCSR_RCWSRn。

	0	1	2	3	4	5	6	7
Preamble (required)	1	0	1	0	1	0	1	0
	0	1	0	1	0	1	0	1
	1	0	1	0	1	0	1	0
	0	1	0	1	0	1	0	1
RCW data	ACS=0	BYTE_CNT = 000000 (64 bytes)						CONT=1
	SYS_ADDR[23-16][1]							
	SYS_ADDR[15-8] [1]							
	SYS_ADDR[7-0] [1]							
	BYTE0							
	BYTE1							
	BYTE2							
	.....							
	BYTE63							

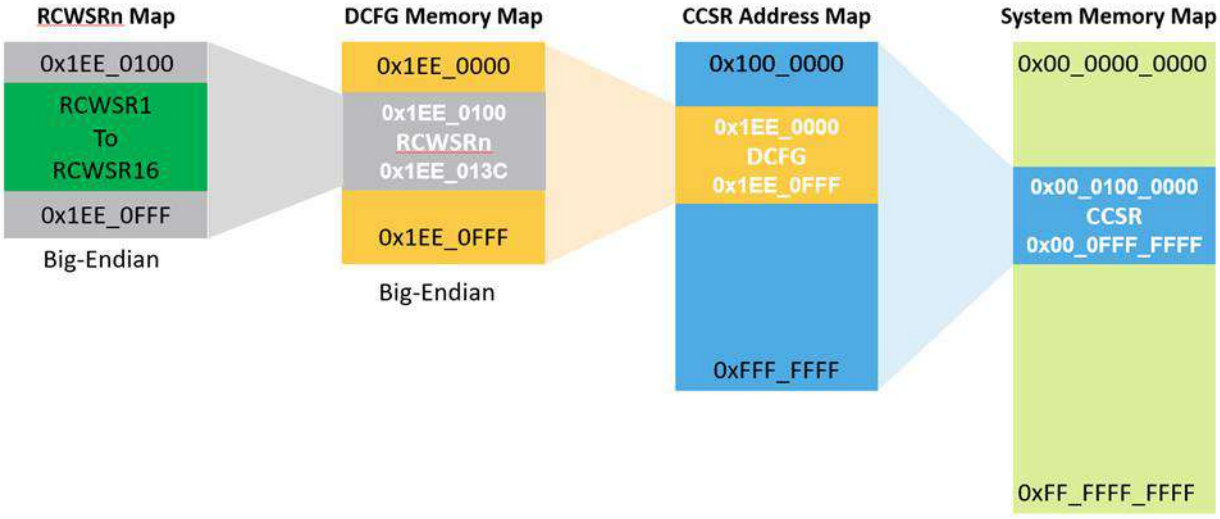
圖四之八、PBL 及 RCW 數據結構圖(自行拍攝)

接下來要試著去查詢 DCFG_CCSR_RCWSR 的內容。要查詢該 Register，需要先對 LS1043A System Memory Map 有基本的概念。我將這部分的學習材料，放在附錄五。

(三)認識 LS1043A System Memory Map

LS1043A 的 System Memory Map 涵蓋了晶片內部所有可用的資源，System Memory 被切割成不同的區塊，有些區塊可以針對特定的存儲介面做資料的讀寫，例如說 IFC 及 DDR。System Memory 中有一個特定的區塊，可以用來配置及控制及檢察晶片內各種不同功能目前的狀態，這個區塊就稱為 Configuration, Control and Status Register (CCSR)。

特別需要查找 CCSR 的原因是因為要確認 1043 晶片有正確的讀取到 NOR Flash 內的 RCW 值，而 DCFG 的地址是 0x1EE0000，參照 QorIQ LS1043A Reference Manual, Rev 6 [5]。將 RCWSR 在 System Memory Map 中的位置總結如下圖四之九所示。



圖四之九、RCWSR 在 System Memory Map 中的位置圖(自行繪製)

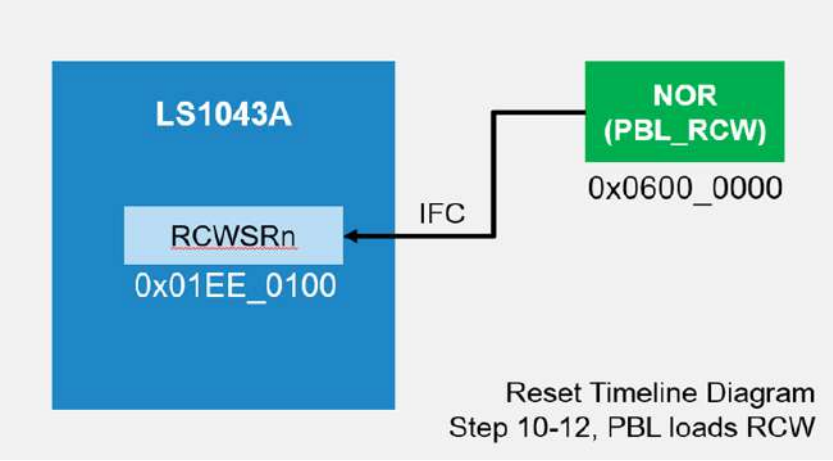
辨別開機後 1043 晶片是否有正確的讀取到 NOR Flash 內的 RCW 值的方法是去讀取 DCFG_CCSR_RCWSR 以及 IFC Region 的內容。兩者分別為 1043A 晶片的 RCW 及 NOR Flash 內的 RCW 值。“0x60000000”在 memory map 上指向 NOR Flash，訪問這個地址就代表要訪問 NOR Flash。而“0x1EE0100”在 memory map 上是指向 DCFG_CCSR_RCWSRn，所以訪問這個地址就代表要訪問 DCFG_CCSR_RCWSRn。兩者顯示的內容相同，代表開機後 1043 晶片有正確的讀取到 NOR Flash 內的 RCW 值。0x60000000 這個地址指向 NOR Flash 的開頭，頭四個 Bytes 的值為 0xAA55AA55 (Big Endian Swap)，在 PBL 的 Data Structure 中，RCW 的值是從第 9 個 Byte 開始，如下圖四之十所示。

```
=> md 0x1EE0100 14
01ee0100: 10001008 0000000a 00000000 00000000
01ee0110: 02005514 12400080 005002e0 002000c1
01ee0120: 00000000 00000000 00000000 00880300
01ee0130: 00000000 01110000 96000000 01000000
01ee0140: 00000000 00000000 00000000 00000000
=> md 0x60000000
60000000: 55aa55aa 0001ee01 10001008 0000000a
60000010: 00000000 00000000 02005514 12400080
60000020: 005002e0 002000c1 00000000 00000000
60000030: 00000000 00880300 00000000 01110000
60000040: 96000000 01000000 78015709 10e00000
```

圖四之十、NOR Flash 及 RCWSRn 內 RCW 值的比較圖(自行拍攝)

由圖四之五所示，LS1043A 在開機上電後，透過腳位的電壓高低來決定開機所需的第一條指令在哪裡讀取。在圖四之五 Step10, 12 中，依照目前的設定，LS1043A 會透過 IFC 通道讀取 NOR FLASH 上 PBL 的 RCW。而後會再寫到 RCWSR 上，在實際執行時，LS1043A 透過讀取地址 0x06000000，將讀取到 PBL 的值，寫入地址 0x01EE0100。

“0x01EE0100” 在 memory map 上是指向 RCWSR，所以訪問這個地址就代表要訪問 DCFG_CCSR_RCWSR。LS1043A 於開機重啟階段讀取外部 PBL 的方法如下圖四之十一所示。



圖四之十一、LS1043A 於開機重啟階段讀取外部 PBL(自行繪製)

RCW 共有 64Bytes，512bits。我們需要特別查找 Bit(192-195)。Bit(192-195)是代表

PBI_SRC，它的意義是要辨識 PBI_SRC，因為可能的 PBI_SRC 包括 QSPI 或 SD/MMC 或 IFC，根據 Bit(192-195)就可以決定是這三種可能中的哪一種。舉例來說，QSPI 為 010x，SD/MMC 為 0110，IFC 為 1110。而讀出來的結果為 1110，代表說，目前這個 PBI_SRC 是設定為 IFC，參照下圖四之十二。這符合我的預期，所以我可以確定開機後 1043 晶片有正確的讀取到 NOR Flash 內的 RCW 值，也可以確定該 RCW 的值是合理的。

Boot configuration (fields 192-223)			
192-195	PBI_SRC	Pre-boot initialization source. The pre-Boot loader fetches address/data pairs from the selected interface for the purpose of pre-boot Initialization of CCSR and/or local memory space.	The following restriction apply: • RCW and pre-boot initialization data must be loaded from the same non-volatile memory device The hard coded RCW source options are not considered their own memory interface for this purpose. Options: 010x QSPI 0110 SD/MMC 1110 IFC (The RCW field IFC_MODE configures the IFC, provided the IFC has not already been configured by the cfg_rcw_src configuration input signals, which have precedence.) All other encodings are reserved.

圖四之十二、RCW 中的 PBI_SRC Options 選項圖(自行拍攝)

(四)設計啟動 EL1 Secure physical Timer 計時器

計時器的名稱是 EL1 Secure physical Timer，計時器在 Cortex-A53 processor 內部，在圖四之五 Step 16、PBI phase 啟動始計器是比較複雜的做法，因為此階段對晶片的操作必須要遵守 PBI 指令的特定格式，若是在 U-Boot 階段才開始計時，可以使用一般 C 語言，雖然比較簡單，但是也失去了計時開機時間這個目的的意義了，因為到那時才開始計時，其實電腦已經開機到半途了，這樣紀錄下的時間就不是完整的開機時間。

而想要啟動 EL1 Secure physical Timer 計時器，就需要修改 Secure_system_counter_CNTCR 的內容，CNTCR 寄存器內容如下圖四之十三所示。這個寄存器是用來規定 EL1 Secure physical Timer 計時器使否開啟的。如圖四之六所示，Bit0 如果顯示 0 的話就代表計時器是關閉狀態的，Bit0 如果顯示 1 的話就代表計時器

此刻正在運作。所以這個 register 需要填入 0x00000001，代表計時器開始運作。

Address: 2B0_0000h base + 0h offset = 2B0_0000h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

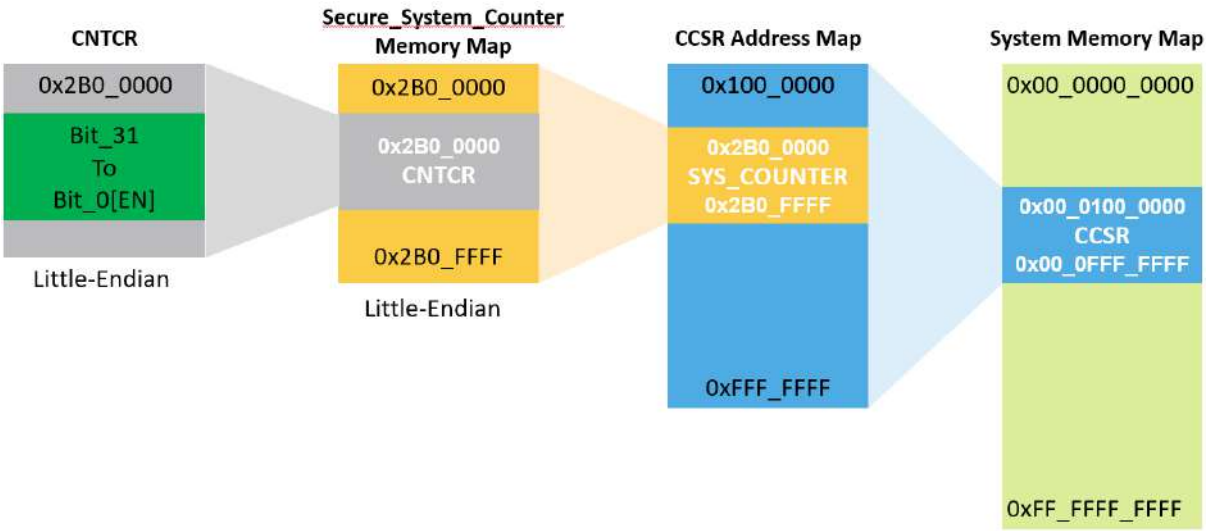
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved															EN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Secure_system_counter_CNTCR field descriptions

Field	Description
31–1 -	This field is reserved.
0 EN	Enables the counter 0 System counter disabled. 1 System counter enabled.

圖四之十三、CNTCR 寄存器內容圖(自行拍攝)

根據 Secure_system_counter_CNTCR 的說明，該 Register 的地址位於 0x02B0_0000，0x02B0_000 屬於 QorlQ LS1043A Reference Manual [5] Chapter 8 、Secure System Counter，0x02B0_000 到 0x2B0_FFFF 的區塊中。而 Secure System Counter 又屬於 CCSR Block Base Address Map 0x00_0100_0000 到 0x00_0FFF_FFFF 的區塊內。而 CCSR Block Base Address Map 又屬於 System Memory Map 內。下圖四之十四顯示了 Secure_system_counter_CNTCR 在 System Memory Map 內的位置。“0x02B0_0000”在 memory map 上是指向 Secure_system_counter_CNTCR，所以訪問這個地址就代表要訪問 Secure_system_counter_CNTCR。



圖四之十四、CNTCR 在 System Memory Map 中的位置圖(自行繪製)

為了讓計時器成功開始運作。因此，我們需要在 0x02B0_0000 這個地址填入 0x00000001。以上操作代表我在 Secure_system_counter_CNTCR 這個 Register 內將 CNTCR[31]配置成了 1，1 代表我啟動(enable)了該計時器。

(五)設計 Pre-Boot Initialization (PBI) 指令以啟動計時器

要啟動 EL1 Secure physical Timer 計時器，就必須把 CNTCR register 的 Bit 0 填入 1。Bit0 如果顯示 0 的話就代表計時器是關閉狀態的，Bit0 如果顯示 1 的話就代表計時器此刻正在運作計時。若是需要修改 bit 0 的值的話，就需要去查 CNTCR 的地址，而 CNTCR 的地址就是 0x02B00000。而想要讓計時器開始運作的話，這個 CNTCR register 就需要填入以下的值 0x00000001，代表計時器開始運作。

但是由於 CNTCR 的地址 0x02B00000 超出了標準 Base 0x01000000 的範圍，所以需要用到 ALTCBAR 寄存器，以設定替代 Base 為 0x02000000。ALTCBAR 寄存器內容如下圖四之十五所示，ALTCBAR[0-23] 表示 Alt configuration base address，這個解釋讓我難以理解，在請教了專家後，我得知 ALTCBAR[0-23] 代表了 48-bit address mode 中的高位 24-bit，也就是說，若是我要設定替代 Base 為 0x02000000，以 48-bit address mode 來表示就是 0x0000_0200_0000，所以我需要在 ALTCBAR 填入 0x0000_0200。

Address: 157_0000h base + 158h offset = 157_0158h

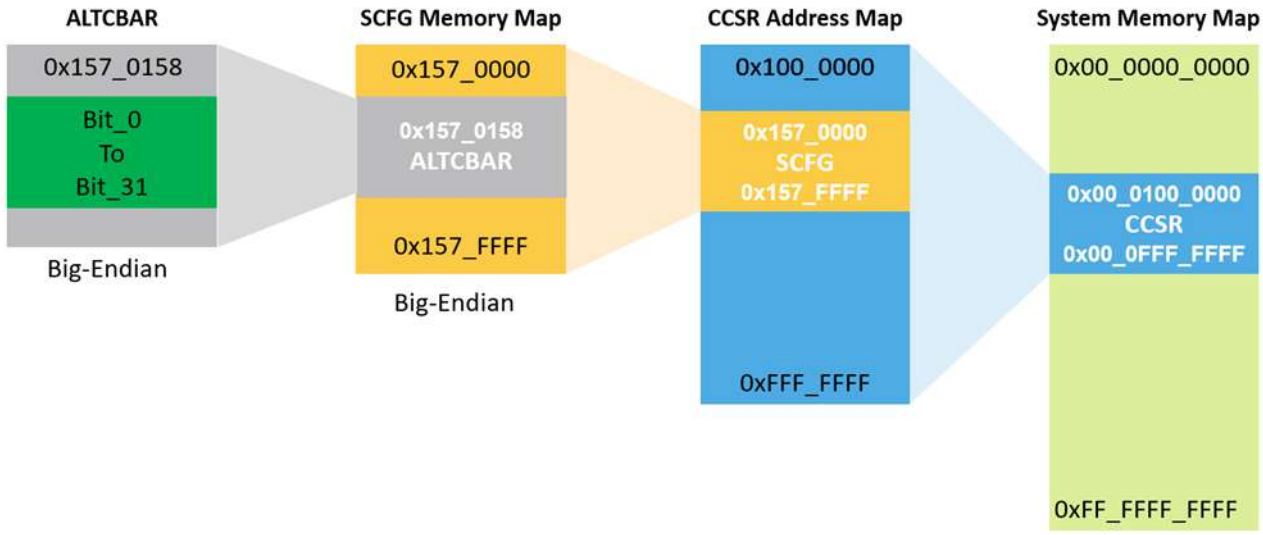
Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ALTCBAR																								—							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SCFG_ALTCBAR field descriptions

Field	Description
0–23 ALTCBAR	Alt configuration base address register for PBL.
24–31 —	Reserved

圖四之十五、ALTCBAR 寄存器內容圖(自行拍攝)

要設定 Alternate Configuration Space 就需要先查詢 ALTCBAR 的地址。如下圖四之十六顯示，查詢 System Memory Map 後，ALTCBAR 位於 CCSR 中的 SCFG，由此可知 ALTCBAR 的地址就是 0x01570158。



圖四之十六、ALTCBAR 在 System Memory Map 中的位置圖(自行繪製)

總結以上開啟計時器所需的步驟，需使用 PBI 指令修改 ALTCBAR 及 CNTCR 兩個寄存器，對 ALTCBAR 寫入 0x0000_0200，對 CNTCR 寫入 0x0000_0001。由於 CNTCR 的地址 0x02B0_0000 超出了標準 Base，0x0100_0000 到 0x01FF_FFFF，的範圍，所以要先利用 ALTCBAR 定義 Alternate Base，0x0200_0000 到 0x02FF_FFFF，如此一來 CNTCR 的地址就可以被覆蓋在 Alternate Base 的範圍當中。接下來我將研究如何正確使用 PBI 指令來對寄存器做寫入的操作。

要使用 PBI 指令，就必須依照其所規定的數據結構。參考 LS1043A 處理器使用手冊[5]，PBI 數據結構如圖四之十七所示。

First pre-boot initialization command (optional)	ACS	BYTE_CNT	CONT=1
	SYS_ADDR[23-16]		
	SYS_ADDR[15-8]		
	SYS_ADDR[7-0]		
	BYTE0		
	BYTE1		
	BYTE2		
	.....		
	BYTE N-1 (up to 63)		

圖四之十七、PBI 指令數據結構圖(自行拍攝)

如上圖四之十七所示，一個完整的 PBI 指令包含了 ACS、BYTE_CNT、CONT、SYS_ADDRn、BYTEn 等五個欄位。要填入有效的 PBI 指令，就必須對五個欄位有所了解，其五個欄位的意義分述如下。

如下圖四之十八，設定 ACS 的值為 0 或 1。當 ACS = 0 時，代表使用標準的 Address Base = 0x0100_0000，這個 Address Base 指向 CCSR address map 的開頭起始位置。當當 ACS = 1 時，代表使用替代的 Address Base，這個替代的 Address Base 是定義在 ALTCBAR Register 當中。

Field name	Description
ACS	Alternate Configuration Space. Logic 1 for alternate configuration space. Logic 0 for CCSR Space. Note that this field must be logic 0 for the RCW data.

圖四之十八、ACS 定義(自行拍攝)

如下圖四之十九，BYTE_CNT 欄位是在定義 BYTEn 總共有幾個 Byte。BYTE_CNT=0b000000 就代表有 64bytes，BYTE_CNT=0b000100 就代表有 4bytes。這個欄位主要的作用，是告訴 PBL 這個 PBI 指令的數據共包含幾個 Byte。而 PBL 會根據 BYTE_CNT 欄位的定義，去讀取相對應長度的數據。

Field name	Description
BYTE_CNT	Byte Count. Represents the number of bytes associated with this address/data pair. Note that an encoding of all zeros represents 64 bytes. Only power of two multiples are legal (1, 2, 4, 8, 16, 32, and 64 bytes). The associated SYS_ADDR must be aligned to the byte count of the command. 000000 64 bytes 000001 1 byte 000010 2 bytes 000100 4 bytes 001000 8 bytes 010000 16 bytes 100000 32 bytes All other encoding are reserved

圖四之十九、BYTE_CNT 定義(自行拍攝)

如下圖四之二十，CONT 欄位是在表示這個 PBI 指令是否為最後一個 PBI 指令。若 CONT = 1，則代表說這個 PBI 指令還不是最後一個，PBL 還需要再往下讀取下一個指令。若 CONT = 0，則代表說這個 PBI 指令是最後一個，PBL 已經讀取完所有指令。

CONT	Continue. This bit should be logic 1 for all commands except for the End command.

圖四之二十、CONT 定義(自行拍攝)

如下圖四之二一，SYS_ADDRn 欄位表示 System Address，共有三個 byte，代表存取地址的 offset 值。這個欄位要和第一個欄位 ACS 合併使用。例如說，ACS = 0，搭配 SYS_ADDRn = 0x57，0x01，0x58，則代表使用標準 Address Base = 0x0100_0000 加上 Address offset = 0x570158，最後用來存取目標地址 Address = 0x01570158。

SYS_ADDR	System Address. The lowest order system address bits. Note that this permits addressability down to a byte for single byte transactions. SYS_ADDR must be aligned to the byte count of the command. The upper bits are either selected from alternate configuration BAR (depending on value of ACS) and then concatenated with SYS_ADDR to form the full address associated with the command.

圖四之二一、SYS_ADDRn 定義(自行拍攝)

如下圖四之二二，BYTEn 欄位要和欄位 BYTE_CNT 搭配使用。舉例來說，若是 BYTE_CNT=0b000100，就代表說 BYTEn 共有四個 Bytes，包括 Byte0、Byte1、Byte2、Byte3。所以 PBL 需要去讀取四個 Byte 長度的值。

BYTEn	Byte number *n* where *n* may range from 0 to 63. Byte 0 corresponds to address 0 relative to the SYS_ADDR (starting address), byte 1 for address 1,..., and byte 63 for address 63. Note the first command must be the RCW address/data pair and must be 64 bytes of data . Note that BYTE*n* is only present/valid in the data structure if *n* is less than the BYTE_CNT.

圖四之二二、BYTEn 定義(自行拍攝)

在正確理解了 PBL 的數據結構後，我開始設計第一個 PBI 指令，也就是使用 PBI 指令去修改 ALTCBAR 寄存器，對 ALTCBAR 寫入 0x0000_0200，如下圖四之二三所示。

ACS=0b0	**BYTE_CNT=0b000100**	**CONT=0b1**	**0x09**
SYS_ADDR[23-16]			**0x57**
SYS_ADDR[15-8]			**0x01**
SYS_ADDR[7-0]			**0x58**
BYTE0			**0x00**
BYTE1			**0x00**
BYTE2			**0x20**
BYTE3			**0x00**

圖四之二三、PBI 指令寫入 ALTCBAR 設計摘要圖(自行繪製)

1. 目標: ADDR: 0x01570158， DATA: 0x00000200
2. ACS=0，使用 Default Configuration Space，因為目標 ADDR Base 是 0x01000000
3. BYTE_CNT=000100，因為 ALTCBAR Register 共 32bit，4 個 Bytes，故按圖四之十九選擇 000100 4bytes
4. CONT=1，因為本指令不是 PBI 最後一個指令，後面還有其他指令，故按照規定 CONT=1
5. 綜合以上欄位，本 PBI 指令第一個 BYTE = 0b00001001 = 0x09

6. SYS_ADDR 的目標地址為 0x01570158，其中 0x01000000 即為 Default Configuration Space，再加上 Address offset = 0x570158，所以填入以下三個欄位。

   1) SYS_ADDR[23-16]= 0x57

   2) SYS_ADDR[15-8]= 0x01

   3) SYS_ADDR[7-0]= 0x58

7. BYTE[0-3] = 0x00000200，則表示對 ALTCBAR Register 寫入值 0x00000200。

   1) BYTE0 = 0x00

   2) BYTE1 = 0x00

   3) BYTE2 = 0x02

   4) BYTE3 = 0x00

8. 綜合以上步驟，寫入 ALTCBAR Register 的 PBI 指令為

   1) 0x0957_0158

   2) 0x0000_0200

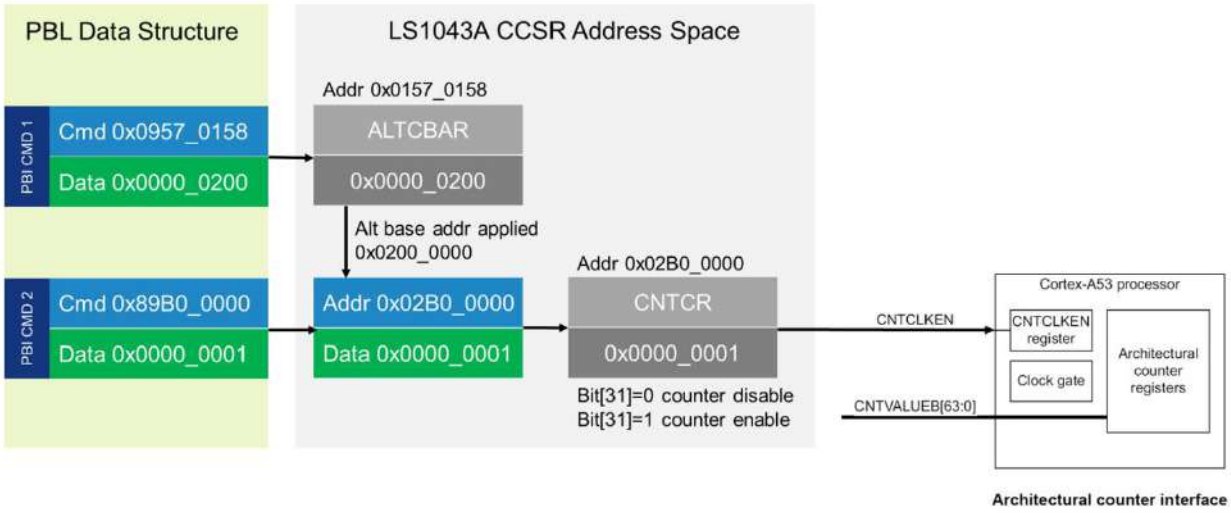
我接著設計第二個 PBI 指令，也就是使用 PBI 指令去修改 CNTCR 寄存器，對 CNTCR 寫入 0x0000_0001，如下圖四之二四所示。

ACS=0b1，註 2.	**BYTE_CNT=0b000100，註 3.**	**CONT=0b1，註 4.**	**0x89，註 5.**
SYS_ADDR[23-16]，註 6.			**0xB0，註 6.**
SYS_ADDR[15-8] ，註 6.			**0x00，註 6.**
SYS_ADDR[7-0] ，註 6.			**0x00，註 6.**
BYTE0，註 7.			**0x00，註 7.**
BYTE1，註 7.			**0x00，註 7.**
BYTE2，註 7.			**0x00，註 7.**
BYTE3，註 7.			**0x01，註 7.**

圖四之二四、PBI 指令寫入 CNTCR 設計摘要圖(自行繪製)

1. 目標: ADDR: 0x02B00000， DATA: 0x00000001
2. ACS=1，使用 Alternate Configuration Space，因為目標 ADDR Base 是 0x02000000
3. BYTE_CNT=000100，因為 CNTCR Register 共 32bit，4 個 Bytes，故選擇 000100 4bytes
4. CONT=1，因為本指令不是 PBI 最後一個指令，後面還有其他指令，故按照規定 CONT=1
5. 綜合以上欄位，本 PBI 指令第一個 BYTE = 0b10001001 = 0x89
6. SYS_ADDR 的目標地址為 0x02B0_0000，其中 0x0200_0000 即為 Alternate Configuration Space，再加上 Address offset = 0xB00000，所以填入以下三個欄位。
   1) SYS_ADDR[23-16]= 0xB0
   2) SYS_ADDR[15-8]= 0x00
   3) SYS_ADDR[7-0]= 0x00
7. BYTE[0-3] = 0x00000001，則表示對 CNTCR Register 寫入值 0x00000001。
   1) BYTE0 = 0x00
   2) BYTE1 = 0x00
   3) BYTE2 = 0x00
   4) BYTE3 = 0x01
8. 綜合以上步驟，編寫 CNTCR Register 的 PBI 指令為
   1) 0x89B0_0000
   2) 0x0000_0001

要開啟 EL1 Secure physical Timer 計時器，就必須把 CNTCR register 的 Bit 0 填入 1，因為這樣代表啟動計時器。而開啟計時器所需的步驟，需使用 PBI 指令修改 ALTCBAR 及 CNTCR 兩個寄存器，所以開啟 EL1 Secure physical Timer 計時器，共需要以下兩條指令，透過 PBI 指令去啟動 LS1043A 內部的計時器流程，總結如下圖四之二五。在後續的實驗中，我發現了我在這個階段的一個設計疏漏，這個錯誤導致我的計時器無法如預期中開啟，我在以下小節將討論這個設計疏漏，以及修正的方法。



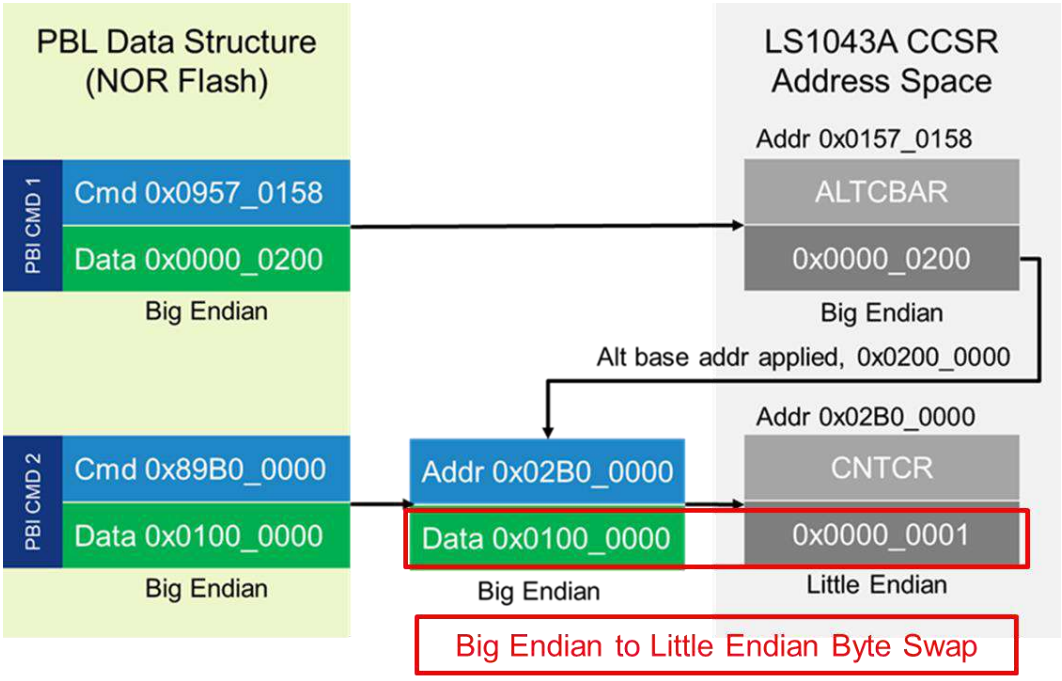
圖四之二五、以 PBI 指令啟動 LS1043A 內部的計時器流程圖(自行繪製)

我使用以上兩個 PBI 指令啟動 EL1 Secure physical Timer 計時器，然而在後續實驗中，當我嘗試開啟計時器的時候，發現計時器無法如我預期般開啟。所以我懷疑我的 PBI 指令有寫錯的地方，而在我重新仔細查看操作手冊後，發現跟數據排列的 Endian Mode 有關。

其中所有儲存在 NOR Flash 的數據排列皆是由 Big Endian 組成，但是儲存在 CCSR Address Space 中的數據排列可能由 Big Endian 或是 Little Endian 組成，所以有需要改變排列(Byte Ordering)的問題。我在上面的實驗中不了解 CNTCR 的數據由 Big Endian 轉變為 Little Endian，因此造成了計時器無法順利的啟動。

我在將 CNTCR 的數據成功換成相對應的 Endian Mode 後，發現正確的 CNTCR 數據應該要排列為 0x0100_0000，在經過 Byte Swap 的轉換之後，CNTCR 所讀取到的數據，才是正確意義上的 0x0000_0001。如此一來就可以順利啟動 EL1 Secure physical Timer 計

時器。CNTCR 數據在不同 Endian Mode 之間的轉換，如下圖四之二六所示。



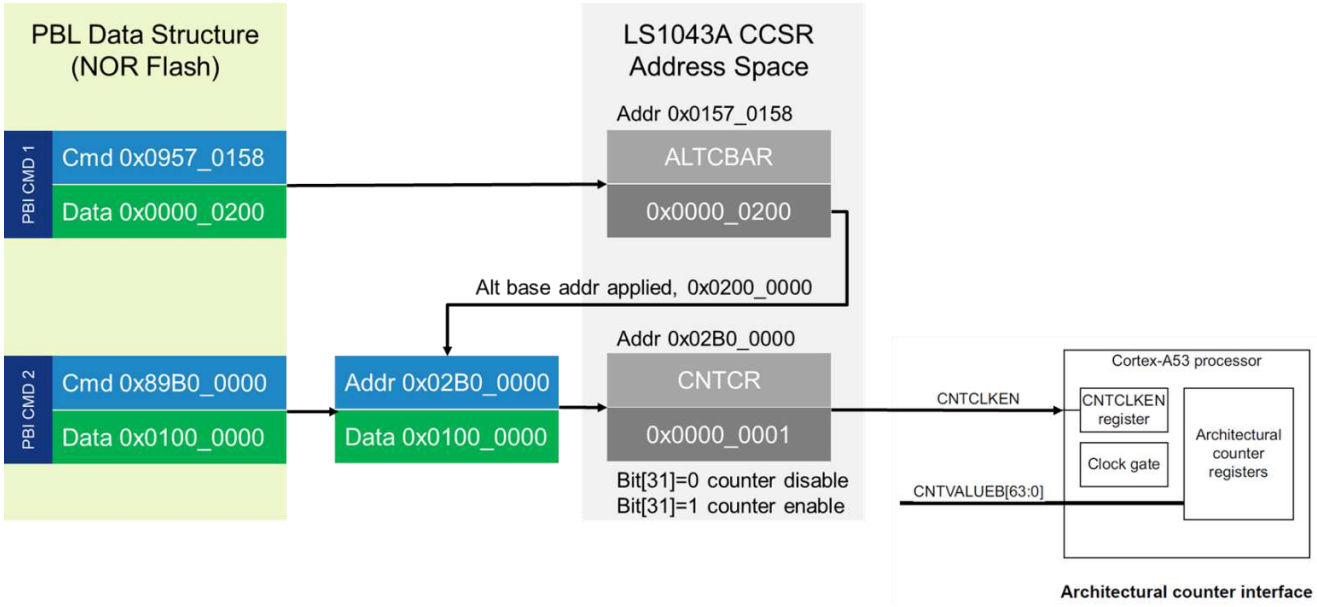
圖四之二六、CNTCR 數據在不同 Endian Mode 之間的轉換圖(自行繪製)

總結以上所有步驟，我的目標是要在開機最短的時間內啟動 EL1 Secure physical Timer 計時器，以達到精準計時的目標。而最快可以啟動計時器的時機在於 PBL loads RCW 的下一個步驟，即可透過 PBI 指令去修改兩個寄存器 CNTCR 及 ALTCBAR 的內容，以此達到啟動計時器並精確計時的目的。而在操作 PBI 指令時也要特別注意 Endian Mode 的轉換。在修正 Endian Mode Byte Swap 需求後，我總結兩條 PBI 指令如下，接下來我會在下一個實驗章節，在 Bootloader 中去開發新增這兩條指令。

PBI 指令一，0x0957_0158、0x0000_0200

PBI 指令二，0x89B0_0000、0x0100_0000

啟動計時器的操作流程總結如下圖四之二七所示。



四之二七、開啟 EL1 Secure physical Timer 計時器操作流程摘要圖(自行繪製)

四、驗證 LS1043ARDB 的開機速度之精準計時方案

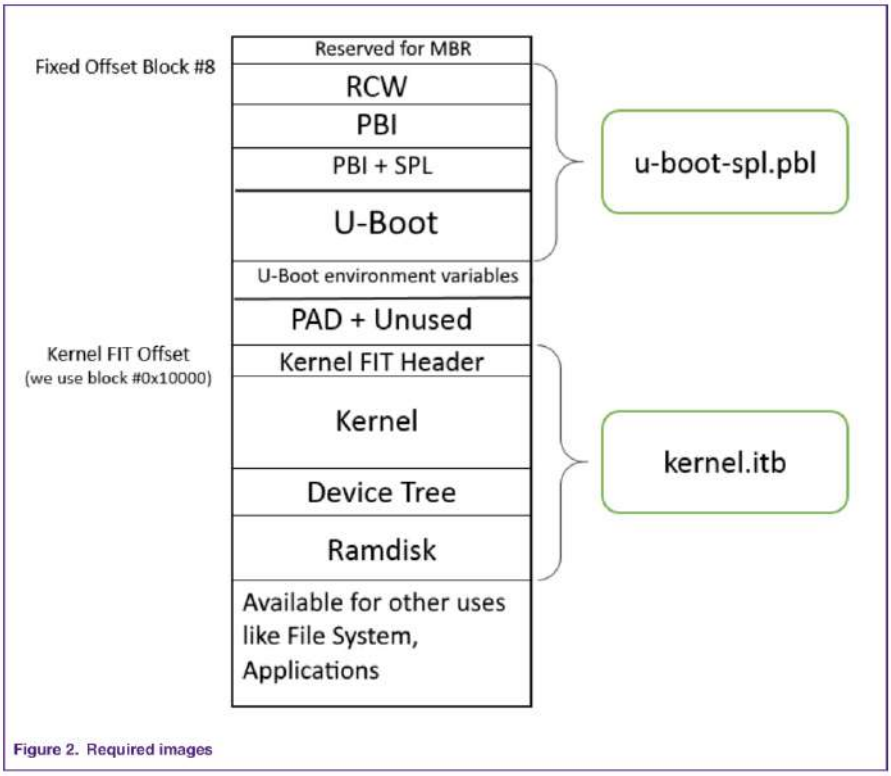
在上一部份實驗三的階段，我設計出可以啟動紀錄 LS1043A 開機時間的軟體代碼(source code)，我需要進一步將這個啟動計時器軟體代碼整合進 LS1043A 的韌體(Firmware)當中，因為 LS1043A 所使用的開機韌體(Bootloader)是 U-Boot，因此我需要將我開發的軟體計時器代碼整合進 LS1043A U-Boot Bootloader，使它成為 Bootloader 的一個新增功能。

LS1043A 晶片開機時需要運用到開機程序(PBL)的結構，並搭配 PBL 所規範的 PBI 指令。而我學習 U-Boot 相關知識是為了了解 U-Boot 的基本結構，以便找出相對應原始代碼檔案的位置，以此修改 PBL 結構以增加 PBI 指令，在我需要的地方啟動計時器，並且在後續 Bootloader 啟動的過程中，插入顯示計時數值的軟體程序，如此便可以在開機過程的各個重要環節，顯示累計的開機時間之讀數，如此便可以精確測量開機過程所花費的時間。

本部分的實驗包括以下步驟，開發新增計時器的原代碼、修改 Bootloader 原代碼並重新編譯、更新 LS1043ARDB 開機韌體以及觀察開機過程並紀錄實驗結果。本部分實驗將完整呈現圖二之二，實驗環境及實驗方法表示圖，所表示實驗過程。

(一)修改 Bootloader 原代碼並重新編譯

使用 LSDK 可以編譯 LS1043ARDB 所使用的韌體，我參考了 LS1043ARDB 的使用手冊，找到韌體的格式如下圖四之二八。接下來的步驟需要在開機韌體（Bootloader）U-Boot 中新增 PBI 的指令，必須要找到相對應的原代碼來進行修改。如下圖四之二八所示，PBI 指令的位置位於開機 RCW 以及 U-Boot 的中間，但是更靠近 RCW，所以判斷 PBI 的原代碼與 RCW 的原代碼可能處於相同位置。所以接下來需要去查找開機韌體(Bootloader)中 RCW 的原代碼。



圖四之二八、PBL、PBI、RCW 三者關係結構圖(自行拍攝)

我查找了 LSDK 內部 RCW 的原代碼，原代碼及其路徑如下圖四之二九所示，原代碼中包含了多個附加檔案，例如 #include <../ls1043aqds/cci_barrier_disable.rcw> 。查看其內容後發現這個檔案就是既有的 PBI 指令。

flexbuild_lsdk2108_github/components/firmware/rcw/ls1043ardb/RR_FQPP_1455/rcw_1600.rcw

```
/*
 * LS1043ARDB RCW for SerDes Protocol 0x1455
 *
 * 15G configuration -- 2 RGMII + 1 QSGMII + 1 XFI
 *
 * Frequencies:
 *
 * Sys Clock: 100 MHz
 * DDR_Refclock: 100 MHz
 *
 * Core          -- 1600 MHz (Mul 16)
 * Platform      -- 400 MHz (Mul 4)
 * DDR           -- 1600 MT/s (Mul 16)
 */

#include <../ls1043aqds/cci_barrier_disable.rcw>
#include <../ls1043aqds/a009929.rcw>
#include <../ls1043aqds/usb_phy_freq.rcw>
#include <../ls1043aqds/uboot_address.rcw>
#include <../ls1043aqds/a009859.rcw>
```

圖四之二九、RCW 的原代碼及其路徑程式圖(自行拍攝)

我進一步研究上述的 PBI 指令後，我推測可以在這個檔案中增加我需要的 PBI 指令，我依據之前的設計，在這個檔案中新增 PBI 指令如下圖四之三十所示，見下圖 8/9/10 三行指令。

```
/*
 * PBI Commands for LS1043AQDS and LS1043ARDB
 */

.pbi
write 0x570178, 0x0000e010
write 0x180000, 0x00000008
write 0x570158, 0x00000200
flush
awrite 0xb00000, 0x01000000
.end

/*
 * Add line 8/9/10 to enable system counter, March 9, 2024
 *
 * About Flush command: Follows the previous write with a read from the
 * same address. The purpose of this command is
 * to ensure the previous write has taken effect.
 */
```

圖四之三十、修改 u-boot 原代碼已新增 PBI 指令(自行拍攝)

如上圖四之三十所示，我模仿了其他附加檔案中的 PBI 指令格式，根據我的需要，在以上檔案中增加了 8/9/10 三行指令，根據之前的設計，這三行指令可以啟動計時器，紀錄開機所需要花費的時間。使用 flush 以及 awrite 的原因是要確定前一個 write 指令已經發生作用，才接著執行下一個指令，我是在這個檔案的說明中學到了這個用法。

修改完原代碼後，我需要用 LSDK 編譯新的韌體。如下圖四之三一所示。以下為四條常用的指令，第一條指令設定編譯環境參數，第二條指令清除之前的舊版本編譯，第三條指令是用來編譯 U-Boot，第四條指令用來編譯完整的開機韌體。

```
#root@john-UX305LA:/home/john/Downloads/flexbuild_lsdk2108_github#
source setup.env
flex-builder clean-firmware
flex-builder -m ls1043ardb -c uboot -a arm64 -b nor
flex-builder -i mkfw -m ls1043ardb -b nor
```

圖四之四之三一、LSDK 常用編譯指令(自行拍攝)

我使用第四條指令編譯了完整的開機韌體，如下圖四之三二所示，編譯成功，代表我新增的指令可以被成功編譯，但是它的功能是否正確還需要後續的實驗加以驗證。

```
make[1]: Leaving directory '/home/john/flexbuild_lsdk2108_github/packages/firmware'
make: Leaving directory '/home/john/flexbuild_lsdk2108_github'
 Writing 0x00000000 <---> firmware/atf/ls1043ardb/bl2_nor.pbl
 Writing 0x00100000 <---> firmware/atf/ls1043ardb/fip_uboot.bin
 Writing 0x00900000 <---> firmware/fm_ucode/fsl_fman_ucode_ls1043_r1.1_106_4_18.bin
 Writing 0x00940000 <---> firmware/qe_ucode/iram_Type_A_LS1021a_r1.0.bin
 Writing 0x00980000 <---> firmware/phy_cortina/cs4315-cs4340-PHY-ucode.txt
 Writing 0x00F00000 <---> linux/kernel/arm64/LS/fsl-ls1043a-rdb-sdk.dtb
 Writing 0x01000000 <---> images/lsdk2108_yocto_tiny_LS_arm64.itb
 /home/john/flexbuild_lsdk2108_github/build/images/firmware_ls1043ardb_norboot.img      [Done]
```

圖四之三二、LSDK 開發環境確認編譯成功(自行拍攝)

假設我已經成功開啟了計時器，接下來就需要把開機各個階段所花費的時間讀取出來。所以我需要 U-Boot 的原代碼中，新增讀取時間以及顯示時間的程式。我請教了專家如何讀取及顯示 EL1 Secure physical Timer 計時器，我了解到已經有現成的軟體 Function Call 可以使用。現有的 Function Call 分列如下圖四之三三所示。下圖所示為讀取時間的 get_timer() Function Call。因為這是既有的 Function Call，所以我可以直接調用不需要在額外新增原代碼。

```
flexbuild_lsdk2108_github/components/firmware/uboot/lib/time.c

/* Returns time in milliseconds */
ulong __weak get_timer(ulong base)
{
        return tick_to_time(get_ticks()) - base;
}
```

圖四之三三、現有 u-boot 中讀取時間的 get_timer() Function Call(自行拍攝)

我也了解到也有現成顯示時間的 printf() Function Call 可以使用。下圖四之三四所示為顯示時間的 Function Call。

```
================================================================================
   printf("Project: PBI start to u-boot console init: %ld ms\n", get_timer(0));
================================================================================
```

圖四之三四、現有 u-boot 中顯示時間的 printf() Function Call

為了記錄開機速度，我決定在開機過程中設置三個量測點，第一個量測點是在 U-Boot 啟動 UART Console 後，第二個量測點是在 U-Boot 完成晶片外部周邊硬體檢查後，第三個量測點是在 U-Boot 執行完畢的時候。為了實現上述三個量測點，我必須要在 Bootloader 中找到相對應的原代碼，並插入顯示時間的 Function Call。修改第一量測點，第一個量測點是在 U-Boot 啟動 UART Console 後，我在以下路徑找到了原始檔案，檔案名稱為 cpu.c。如下圖四之三五所示。

```
root@john-UX305LA:/home/john# ls flexbuild_lsdk2108_github/components/firmware/uboot/arch/arm/cpu
/armv8/fsl-layerscape/cpu.c
flexbuild_lsdk2108_github/components/firmware/uboot/arch/arm/cpu/armv8/fsl-layerscape/cpu.c
```

圖四之三五、原始檔案的存放位置及其名稱(自行拍攝)

接下來就需要修改 cpu.c 中的原代碼，如下圖四之三六所示，在 print_cpuinfo 這個 Function Call 當中，在 1005 及 1006 行中增加顯示時間的程式。

```
autoboot.c   ls1043ardb.c   board_info.c   cpu.c   putty.log
992  printf("after u-boot console init: ms\n");
993  wrong location to insert the code here.
994  */
995
996  #ifdef CONFIG_DISPLAY_CPUINFO
997  int print_cpuinfo(void)
998  {
999      struct ccsr_gur __iomem *gur = (void *)(CONFIG_SYS_FSL_GUTS_ADDR);
1000     struct sys_info sysinfo;
1001     char buf[32];
1002     unsigned int i, core;
1003     u32 type, rcw, svr = gur_in32(&gur->svr);
1004
1005     /*Adding system counter. Project: Boot Time Measurement, March 2024*/
1006     printf("Project: PBI start to u-boot console init: %ld ms\n", get_timer(0));
1007
1008     puts("SoC: ");
```

圖四之三六、在 U-Boot 啟動 UART Console 後增加開機計時顯示(自行拍攝)

修改第二量測點，第二個量測點是在 U-Boot 完成晶片外部周邊硬體檢查後，我在以下路徑找到了原始檔案，檔案名稱為 ls1043ardb.c。接下來就需要修改 ls1043ardb.c 中的原代碼，如下圖四之三七所示，在 189 及 190 行中增加顯示時間的程式。

```
File  Edit  View  Projects  Bookmarks  Sessions  Tools  Settings  Help
Documents  Projects  Filesystem Browser
ls1043ardb.c   john_notes_March
177      printf("Project: Boot Time Measurement 2024, March\n");
178
179  #ifdef CONFIG_TFABOOT
180      }
181  #endif
182      printf("CPLD:  V%x.%x\nPCBA:  V%x.0\n", CPLD_READ(cpld_ver),
183             CPLD_READ(cpld_ver_sub), CPLD_READ(pcba_ver));
184
185      puts("SERDES Reference Clocks:\n");
186      sd1refclk_sel = CPLD_READ(sd1refclk_sel);
187      printf("SD1_CLK1 = %s, SD1_CLK2 = %s\n", freq[sd1refclk_sel],
         freq[0]);
188
189      /*Adding system counter. Project: Boot Time Measurement, March 2024*/
190      printf("Project: u-boot check board complete: %ld ms\n",
         get_timer(0));
191
192      return 0;
193  }
194
```

圖四之三七、在 U-Boot 完成晶片外部周邊硬體檢查後增加開機計時顯示(自行拍攝)

修改第三量測點，第三個量測點是在 U-Boot 執行完畢的時候，我在以下路徑找到了原始檔案，檔案名稱為 auotboot.c。如下圖四之三八所示。

```
root@john-UX305LA:/home/john# ls flexbuild_lsdk2108_github/components/firmware/uboot/common/a
utoboot.c
flexbuild_lsdk2108_github/components/firmware/uboot/common/autoboot.c
```

圖四之三八、原始檔案的存放位置及其名稱(自行拍攝)

接下來就需要修改 auotboot.c 中的原代碼，在 abortboot_single_key 這個 Function Call 當中，在 266 及 267 行中增加顯示時間的程式，如下圖四之三九所示。

autoboot.c | ls1043ardb.c | board_info.c | cpu.c

```
258     return abort;
259 }
260
261 static int abortboot_single_key(int bootdelay)
262 {
263     int abort = 0;
264     unsigned long ts;
265
266     /*Adding system counter. Project: Boot Time Measurement, March 2024*/
267     printf("Project: u-boot boot process complete: %ld ms\n", get_timer(0));
268     printf("Hit any key to stop autoboot: %2d ", bootdelay);
269
```

圖四之三九、在 U-Boot 執行完畢的時候增加開機計時顯示(自行拍攝)

(二)更新 LS1043ARDB 開機韌體

在更新完韌體後，我就可以對 LS1043ARDB 重新開機，並透過實驗環境中的管理通道 (UART to USB Serial Console) 在 Ubuntu 20.04 的主機上，透過 Putty Console 畫面讀取 LS1043ARDB 開機系統訊息，並觀察實驗結果。LS1043ARDB 重新開機後完整畫面如下圖四之四十所示，三個量測點已經成功量測到開機時間。開機的時間精度達到了千分之一秒。

U-Boot 2021.04-dirty (Mar 12 2024 - 11:51:50 +0800)

Project: PBI start to u-boot console init: 757 ms，Line-1

SoC: LS1043AE Rev1.0 (0x87920010)

Clock Configuration:

CPU0(A53):1600 MHz CPU1(A53):1600 MHz CPU2(A53):1600 MHz

CPU3(A53):1600 MHz

Bus: 400 MHz DDR: 1600 MT/s FMAN: 500 MHz

Reset Configuration Word (RCW):

00000000: 08100010 0a000000 00000000 00000000

00000010: 14550002 80004012 e0025000 c1002000

00000020: 00000000 00000000 00000000 00038800

00000030: 00000000 00001101 00000096 00000001

Model: LS1043A RDB Board

Board: LS1043ARDB, boot from vBank 4
Project: Boot Time Measurement 2024, March，Line-2
CPLD: V1.4
PCBA: V3.0
SERDES Reference Clocks:
SD1_CLK1 = 156.25MHZ, SD1_CLK2 = 100.00MHZ
Project: u-boot check board complete: 820 ms，Line-3
DRAM: 1.9 GiB (DDR4, 32-bit, CL=11, ECC off)
Using SERDES1 Protocol: 5205 (0x1455)
Firmware 'Microcode version 0.0.1 for LS1021a r1.0' for 1021 V1.0
QE: uploading microcode 'Microcode for LS1021a r1.0' version 0.0.1
Flash: 128 MiB
NAND: 512 MiB
MMC: FSL_SDHC: 0
Loading Environment from Flash... OK
EEPROM: NXID v1
In: serial
Out: serial
Err: serial
SEC0: RNG instantiated
Net: Fman1: Uploading microcode version 106.4.18
eth0: fm1-mac1, eth1: fm1-mac2, eth2: fm1-mac3, eth3: fm1-mac4, eth4: fm1-mac5, eth5: fm1-mac6, eth6: fm1-mac9
Project: u-boot boot process complete: 1270 ms，Line-4
Hit any key to stop autoboot: 10 9 8 7 6 5 4 3 2 1 0

圖四之四十、LS1043RDB 開機 U-boot 啟動系統完整紀錄

以上圖四之四十是 LS1043ARDB 所顯示的標準開機系統訊息，其中包含了所使用的 U-Boot 軟體版本、編譯時間、LS1043A 晶片的詳細型號、處理器核心運行的速度、平台名稱以及其餘系統周邊重要裝置。而其中以藍色標記的系統訊息（共四行），是本實驗實現的計時功能。其中 Line-2 是我設計來驗證韌體能夠重新編譯並且正確更新至 LS1043ARDB 的方法，如圖四之四所示。而 Line-1、Line-3、Line-4，分別代表啟動過程中各階段時間的量測點。第一個量測點是在 U-Boot 啟動 UART Console。第二個量測點是在 U-Boot 完成晶片外部周邊硬體檢查後。第三個量測點是在 U-Boot 執行完畢的時

候。實驗順利完成，我可以正確讀取各個量測點的時間數值至精度千分之一秒(millisecond, ms)。

上圖四之四十紀錄了系統開機 (power on)到 U-Boot 啟動完成的時間，下圖四之四一紀錄了從 U-boot 完成啟動後，Linux 作業系統開始啟動，到 Linux 作業系統啟動完成的時間，兩段時間相加，就是 LS1034RDB 開機到作業系統啟動完成的完整開機耗時。因為 Linux 系統開機訊息過長，我選擇簡單節錄重點訊息。

```
## Loading kernel from FIT Image at a0000000 ...
   Using 'ls1043ardb' configuration
   Trying 'kernel' kernel subimage
      Description:   ARM64 Kernel
      Type:          Kernel Image
      Compression:   gzip compressed
      Data Start:    0xa00000d0
      Data Size:     15188322 Bytes = 14.5 MiB
      Architecture: AArch64
      OS:            Linux
      Load Address: 0x84080000
      Entry Point:   0x84080000
   Verifying Hash Integrity ... OK
## Loading ramdisk from FIT Image at a0000000 ...
   Using 'ls1043ardb' configuration
   Trying 'initrd' ramdisk subimage
      Description:   initrd for arm64
      Type:          RAMDisk Image
      Compression:   uncompressed
      Data Start:    0xa0e7c2e0
      Data Size:     21968191 Bytes = 21 MiB
      Architecture: AArch64
      OS:            Linux
      Load Address: 0x00000000
      Entry Point:   0x00000000
   Verifying Hash Integrity ... OK
## Loading fdt from FIT Image at a0000000 ...
   Using 'ls1043ardb' configuration
   Trying 'ls1043ardb-dtb' fdt subimage
      Description:   ls1043ardb-dtb
```

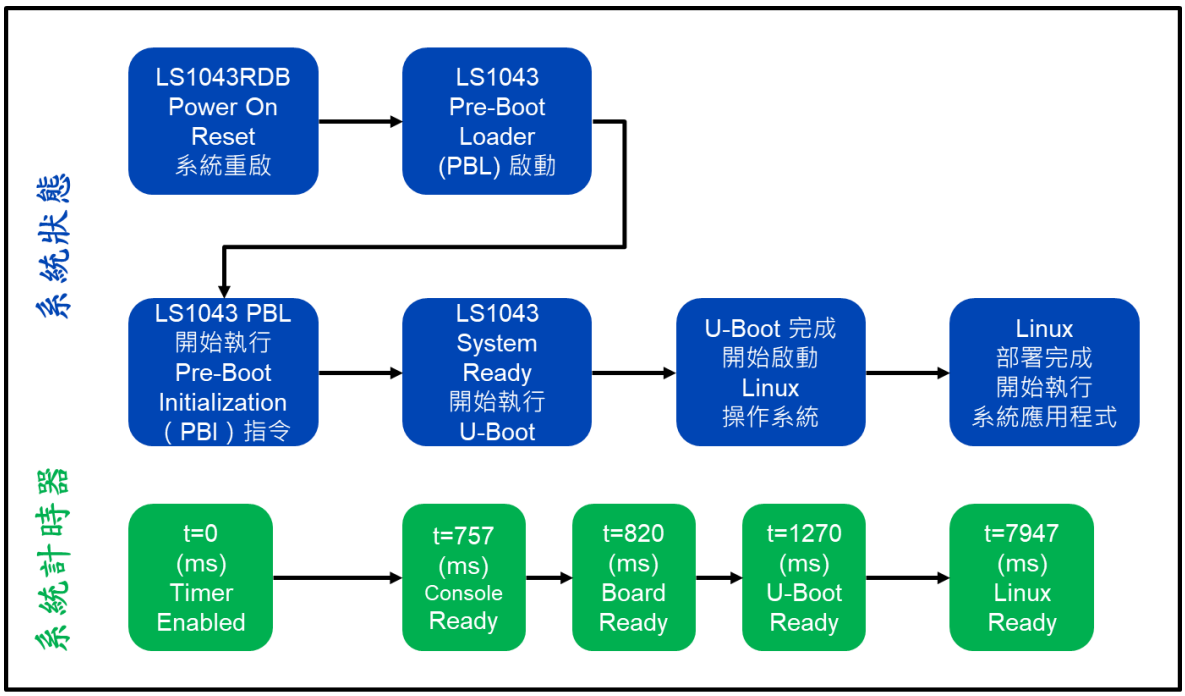
```
   Type:                Flat Device Tree
   Compression:    uncompressed
   Data Start:      0xa238bc48
   Data Size:       39769 Bytes = 38.8 KiB
   Architecture: AArch64
   Load Address: 0x90000000
  Verifying Hash Integrity ... OK
  Loading fdt from 0xa238bc48 to 0x90000000
  Booting using the fdt blob at 0x90000000
  Uncompressing Kernel Image
  Using Device Tree in place at 0000000090000000, end 000000009001cb58

Starting kernel ...

[      0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd034]
[      0.000000] Linux version 5.10.35-dirty (root@john-UX305LA)
[      0.000000] Machine model: LS1043A RDB Board
[      0.000000] earlycon: uart8250 at MMIO 0x00000000021c0500 (options '')
[      0.000000] printk: bootconsole [uart8250] enabled
[      4.344205] udevd[192]: starting version 3.2.10
[      4.372135] udevd[193]: starting eudev-3.2.10
[      4.451532] fsl_dpa soc:fsl,dpaa:ethernet@0 fm1-mac1: renamed from eth0
[      4.524032] fsl_dpa soc:fsl,dpaa:ethernet@1 fm1-mac2: renamed from eth1
[      4.648186] fsl_dpa soc:fsl,dpaa:ethernet@2 fm1-mac3: renamed from eth2
[      6.673863] random: crng init done
[      6.677291] random: 5 urandom warning(s) missed due to ratelimiting

Starting syslogd/klogd: done
NXP LSDK tiny 2108 (based on Yocto)
TinyLinux login: root
root@TinyLinux:~#
```

圖四之四一、LS1043RDB 開機 Linux 啟動系統節錄

本試驗到此完成，綜合以上實驗數據，LS1043ARDB 系統開機的總耗時為，晶片運作初始化到 U-Boot 啟動時間為 1270ms，加上 Linux 啟動時間為 6.677s，共耗時 7947ms，也就是 7.947 秒。

伍、 研究結果

本實驗的研究目的為設計一個計時器，能做到精準量測開機時間，並且希望該計時器的精度能達千分之一秒，使開機時間成為一個能夠量化的指標。根據以上的實驗結果，以及我對 LS1043ARDB 嵌入式系統的理解，我將開機的過程以及開機的耗時，總結為下圖五之一。根據上一章節肆之四的實驗數據，LS1043ARDB 系統開機的耗時為，晶片運作初始化到 U-Boot 啟動時間為 1270ms，加上 Linux 啟動時間為 6.677s，共耗時 7947ms，也就是 7.947 秒。

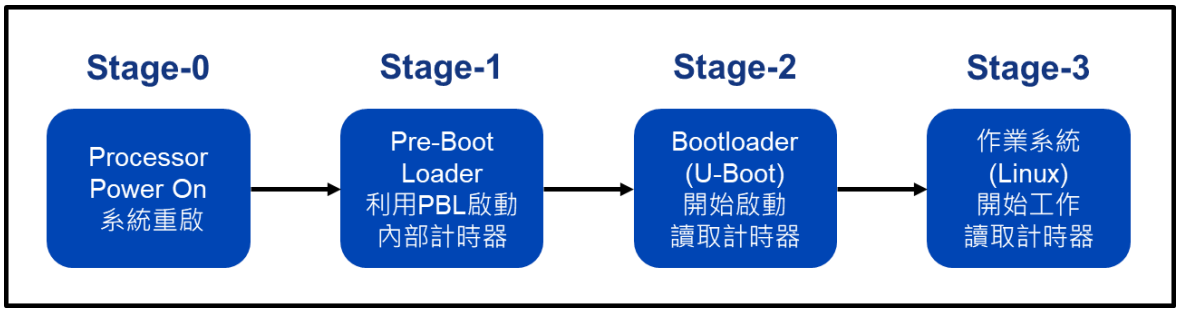


圖五之一、LS1043ARDB 系統啟動過程及耗時圖(自行繪製)

總結本實驗的過程，我了解到我所設計的計時方案，其實是一個通用的設計概念，可以用來量測各種不同類型高階處理器的開機時間。這個計時器的設計思路，可以套用於各類型高階處理器平台，使開機時間成為一個可量化的指標。我總結這個計時器的設計思路於下圖五之二。

高階處理器的功能通常都很複雜，在處理器重啟初始化階段，為了配合實際應用場景的需求，需要做很多晶片內部的功能設定，所以通常晶片在啟動 Bootloader 之前，會先啟動

晶片內部的初始化配置工具（Pre-Boot Loader），來設定晶片內部的功能，就如下圖五之二 Stage-1 所示。我們可以利用 Stage-1 中晶片配置工具的彈性，來配置開啟一個晶片內部的高精度計時器（Timer），這樣就可以在晶片最早期初始化階段，就開始對系統計時。



圖五之二、開機計時方案的通用設計概念(自行繪製)

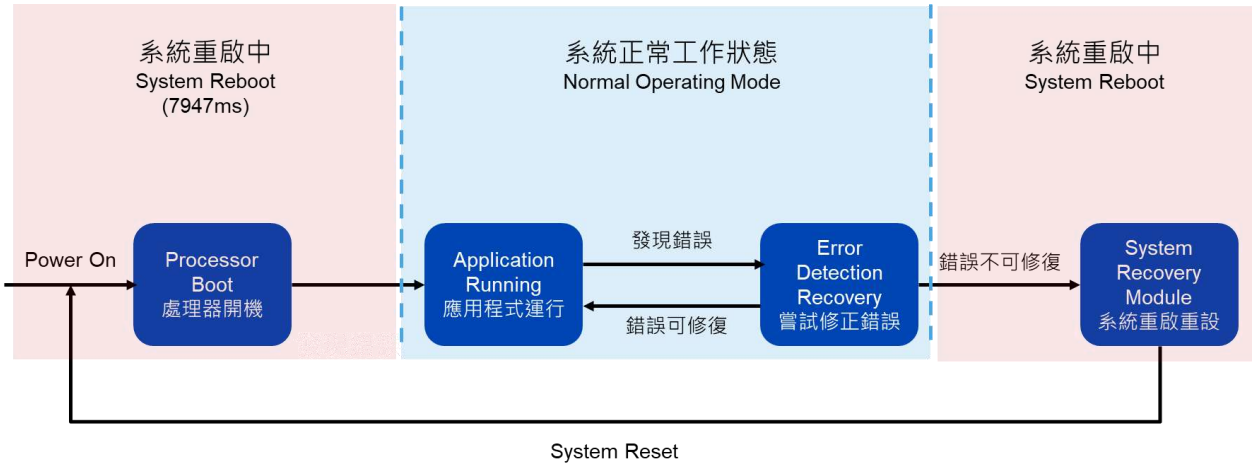
這個計時方案的重點，在於從高階處理器晶片內部的初始化工具(PBL)著手，在晶片開始運行 Bootloader 軟體之前，就啟動計時器，如此可以最大限度完整紀錄系統開機時間，如上圖五之二所示，系統開機時間發生於 Stage-2 以及 Stage-3。或是更進一步，在特定程式執行起點和終點，分別讀取計時器時間，利用時間差概念，即可推算該特定程式運行所需時間。

陸、 討論與應用

高階處理器通常是指能夠運行完整操作系統的晶片平台，目前常見的操作系統包含 Linux、Windows、Android、或是 iOS 等等。這些高階處理器，被廣泛使用於各類型的智能平台，像我這次實驗使用的 LS1043A 處理器，就常常被使用在智能汽車平台上，做為智能汽車裡的各種電子系統，提供各種不同功能，像是行車輔助自動跟車系統，自動泊車系統，車內影音娛樂系統，或是車內門窗座椅控制系統等等。

使用電子系統都必須面對系統可能失效的風險，就像我使用的電腦可能會當機一樣。但是車用電子系統必須特別注意系統的可靠性，因為這直接關係到駕駛及乘客的人身安全。一般的電子系統都有一個失效管理機制，用來維持系統的可靠性，一旦系統失效，可以用來嘗試恢復系統正常功能，我根據我的理解，結合了這次實驗內容，將這個可靠性管理機制總結

為下圖六之一。



圖六之一、系統運行狀態及可靠性之分析圖(自行繪製)

由上圖六之一所示，當我們使用電子系統的功能時，我們都預期系統處於正常工作的狀態。任何電子系統都有發生錯誤的可能，當錯誤發生時，系統本身會嘗試修復這個錯誤，以維持系統正常的運行狀態。但是在最糟糕的情況，當系統本身無法修復既有錯誤時，這個系統就失效了，也無法繼續提供正常服務。當這個狀況發生時，做為恢復系統正常運行的最後一個手段，就是重新啟動系統（system reset），也就是我們平常說的重新開機。

在系統重啟的過程中，電子系統是不能提供正常服務的。這對正在行駛中的車輛來說，就代表了很大的風險。我試著就我的實驗結果去分析這個風險的嚴重程度。假設車輛以時速 120 公里行駛在高速公路上，這個速度換算成秒速就是每秒 33.33 公尺。根據我的實驗結果，LS1043ARDB 系統重新啟動需要消耗 7.947 秒，也就是說，在這段重啟時間內，車輛已經在高速公路上前進了 264.899 公尺，也就是超過了四分之一公里的距離。

從以上的分析可知，能夠精準量測電子系統的開機時間，使開機耗時成為一個可以量化的指標，有助於促進電子系統的可靠性分析。在上述的行車案例裡，我分析了 LS1043A 系統重啟時間，和汽車行進速度與距離的關聯性，由於我設計的計時功能達到了千分之一秒的時間精度，我在推算汽車行進距離的精度就可以達到公分的程度，這說明了計時器精度的重要性。

柒、 參考文獻資料

以下所有參考文件都是可以由相關科技公司官網取得的公開文件。

[1] QorIQ LS1043A Reference Design Board Getting Started Guide. NXP Semiconductor Documentation.

[2] QorIQ LS1043A Reference Design Board Fact Sheet. NXP Semiconductor Documentation.

[3] Layerscape Software Development Kit User Guide, Rev. 21.08, 05 September 2022. NXP Semiconductor Documentation.

[4] QorIQ LS1043A and LS1023A Communication Processors - Fact Sheet. NXP Semiconductor Documentation.

[5] QorlQ LS1043A Reference Manual, Rev 6, 07/2020. NXP Semiconductor Documentation.

[6] ARM Cortex-A53 MPCore Processor Technical Reference Manual, Revision: r0p3. Arm Documentation.

[7] Ubuntu 20.04.6 LTS (Focal Fossa). Ubuntu Documentation.

[8] u-boot/README at master. GitHub Documentation.

[9] Dale, Nell B. Programming and Problem Solving with C++. Jones & Bartlett Learning, 2018.

[10] ARM Cortex-R Series Programmer's Guide，Endianness.

捌、 附錄

一、安裝實驗用電腦主機

以下是我的安裝筆記。首先製作出一個 Bootable 的 USB 硬碟。將電腦洗空才可以安裝成 20.04。到網路上下載 VENTOY 軟體到 WINDOWS 內把 VENTOY 解壓縮，之後再開啟並選擇 GPT 檔案。在 WINDOW 內變成一個 VENTOY 的檔案夾。VENTOY 啟動以後，選擇空白的 USB DRIVE，並且執行安裝 VENTOY 到 USB DRIVE 內。VENTOY 會把空白的 USB DRIVE 內切成兩個部分，一個是 VENTOY 系統 PARTITION，另一個是一般的 PARTITION。由於 UBUNTU 的安裝 ISO 檔案超過 4GB 大小，所以第二 PARTITION 我們選用 EXFAT 的 FORMAT。把下載下來的 UBUNTU20.04 的 ISO 檔案儲存到 VENTOY 的第二個 PARTITION。

安裝之前，先檢查下載下來的 UBUNTU20.04 是否完整且正確。UBUNTU 的網站會提供 SHA256 數值。UBUNTU 電腦開機進入 NOTEBOOK 的 BIOS 系統，把 SECUREBOOT 關閉。選擇正確的 USB DRIVE 來 BOOT 這台電腦。電腦 BOOT 成功後會進入 VENTOY，VENTOY 有提供 SHA256 數值的功能。選擇 VENTOY 計算在下載下來的 UBUNTU 的 CHECK SUM。比較電腦所計算出的結果與網站上提供的 UBUNTU 20.04 數值的比較，如果不同表示在失敗，如果相同則可以進行安裝工作。這樣 USB DRIVE 就製作成功。

第二步要將 USB 插入電腦內並開始安裝 Ubuntu20.04。在電腦按下開機鍵的同時按下 F2 鍵進入 BIOS。選取 USB 為開機裝置，點選 USB 內的 Ubuntu20.04 並開機。開機之後即會進入 Ubuntu 的裝機介面。按照指示完成 Ubuntu 安裝。

二、架設實驗環境中的管理通道

以下是我的實驗筆記。根據 LS1043ARDB 使用手冊，LS1043A 處理器的標準開機輸出介面是 UART1，運用的線路是 RS232 Console 控制線。而因為筆記型電腦並不支持 RS232 的接口，所以我需要運用 USB to RS232 單埠轉換器來連接上筆記型電腦的 USB 孔。

為了確認 Ubuntu 的電腦能夠辨識到這一個 USB 裝置。辨識的方法是運用 Terminal 並輸入”ls usb”這條指令就可以確認電腦是否已經辨識到該 USB 裝置。下圖 2-1 顯示出了所有能被 Ubuntu 系統辨識出的 USB 裝置，其中就包含了這一條轉接線，就是下圖中的 Bus 001

Device 005: ID 067b:2303 Prolific Technology, Inc. PL2303 Serial Port. 這樣就代表 Ubuntu 系統可以使用該裝置。

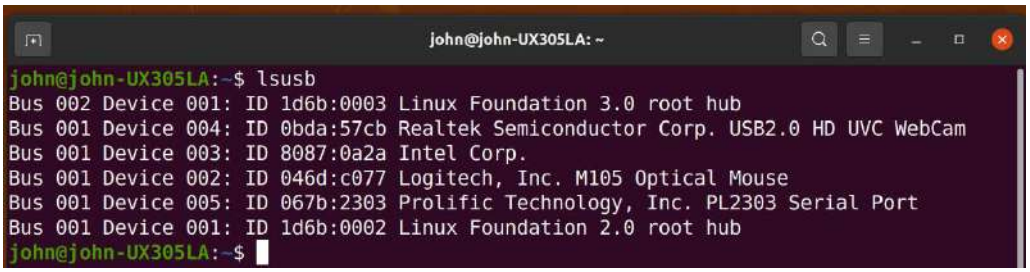


圖 2-1、Ubuntu 系統辨識出的 USB 裝置(自行拍攝)

接著需要確定這個裝置的正確名稱，所以在 Terminal 中搜尋 Ubuntu 系統中的裝置名稱，我使用 ls 指令去查找/dev 目錄下的裝置。結果顯示他被命名為 ttyUSB0。如下圖 2-2 所示。

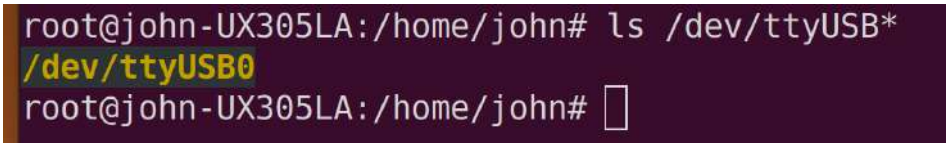


圖 2-2、Ubuntu 系統中的裝置名稱(自行拍攝)

在 Ubuntu 系統上開啟 PuTTy 應用程式，讀取 LS1043ARDB Console 資訊。開機後叫出 Terminal 輸入 Putty, 把 Putty terminal 打開。在 PuTTy Configuration 中輸入正確的裝置名稱。根據 LS1043ARDB 使用手冊，RS232 Console 的標準運行速度 baud rate 為 115200。如下圖 2-3 所示。

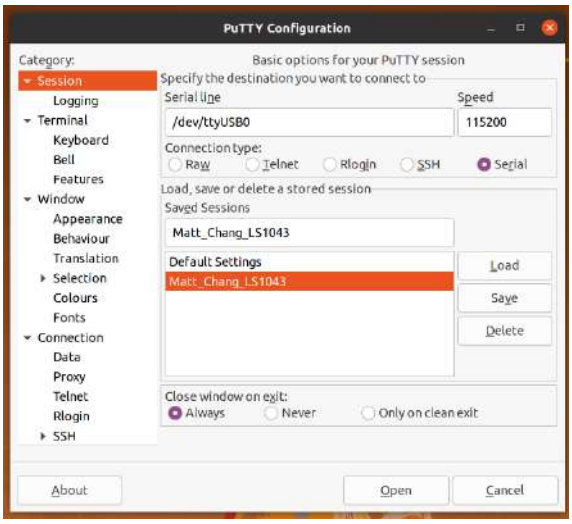


圖 2-3、RS232 Console 的標準運行速度 baud rate 為 115200(自行拍攝)

運用 serial terminal 去控制 LS1043。將 LS1043 開機後，電腦及可讀取到他要傳過來的訊息。這樣就完成了 LS1043ARDB 到 Ubuntu LapTop 的管理通道。如下圖 2-4 所示。

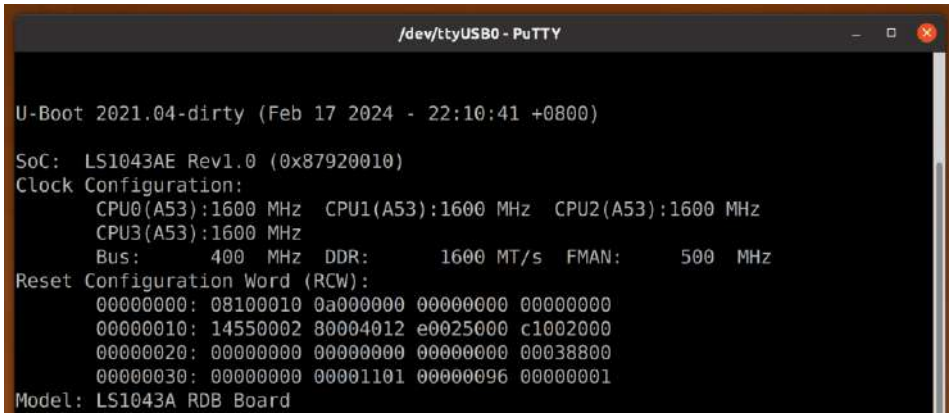


圖 2-4、LS1043ARDB 到 Ubuntu LapTop 的管理通道(自行拍攝)

安裝 LS1043A 處理器軟體開發工具 (LSDK)。LSDK 的全名是 Layerscape Software Development Kit，也就是 LS1043A 處理器軟體開發工具。我們可以運用該開發工具來修改 Bootloader 的程式碼。根據 LSDK 的使用手冊，LSDK 是使用 Flex Build 的編譯環境。我們根據 LSDK 使用手冊的安裝說明，安裝 LS1043A 處理器軟體開發工具，成功後如下圖 2-5 所示。在接下來的實驗中，我們會在該開發環境內優化及開發 Bootloader 的程式。達到修改所短開機所需時間的目的。

圖 2-5、安裝 LS1043A 處理器軟體開發工具(自行拍攝)

在 Ubuntu 系統的電腦上安裝 TFTP 服務器。TFTP 服務器是一個檔案分享及提供下載服務的服務器。我們運用 TFTP 服務器來讓 LS1043ARDB 可以到 Ubuntu 系統的電腦上下載 LSDK 製作好的新的 Bootloader 檔案。在 Ubuntu 系統的電腦上安裝 Kate(text editor)。安裝 Kate(text editor)的用意是為了讓我們可以編寫 Bootloader 的原代碼(source code)。

三、架設實驗環境中的數據通道

以下是我的實驗筆記。建立網路間的連線的原因是為了讓三個裝置間可以互通。LS1043ARDB 和網路機之間是用網路線相連，Ubuntu LapTop 和網路基之間則是用無線網路相連。所有的實驗設備，包含 LS1043ARDB 和 Ubuntu LapTop，都運行在相同的網路 SubNet 之下，使用的 SubNet 為 192.168.50.xxx。而 LS1043ARDB、WiFi Router 和 Ubuntu LapTop 的定址分別為(192.168.50.21)、 (192.168.50.1)和(192.168.50.114)。LS1043ARDB 和 Ubuntu LapTop 建立連線後，可以讓 LS1043ARDB 透過網路去訪問 Ubuntu TFTP server。訪問 Ubuntu TFTP server 的目的是為了讓 LS1043ARDB 可以下載 BootLoader 的新程式碼，用來更新 LS1043ARDB 上的 BootLoader 的版本。下圖 3-1 是為了驗證 LS1043ARDB 跟其他裝置間的連線是互通的，下圖 3-1 顯示表示互通是因為他顯示其他裝置”is alive”，這就表示其他裝置有回應。

圖 3-1、驗證 LS1043ARDB 跟其他裝置間的連線互通(自行拍攝)

我們將以上的六步驟畫成正文圖四之一的開發環境簡圖。LS1043ARDB 透過管理通道 UART1 接到 UART to USB 在轉接到電腦的 USB 孔內。開機後叫出 Terminal 輸入 Putty, 把 Putty terminal 打開。在 PuTTy Configuration 中輸入正確的裝置名稱。運用 serial terminal 去控制 LS1043。將 LS1043 開機後，電腦及可讀取到他要傳過來的訊息。

四、驗證數據通道的功能是否正常運作

以下是我的實驗筆記。在下圖 4-1 中，我們共測試了三條指令。”ping 192.168.50.1”這條指令要求 LS1043ARDB 與 WiFi 網路機之間的連線是互通的，因為下圖 4-1 的顯示表示互

通，是因為他顯示其他裝置”is alive”，這就表示 WiFi 網路機有回應。而”ping 192.168.50.114” 這條指令要求 LS1043ARDB 與 Ubuntu 系統的電腦之間的連線是互通的，因為下圖 4-1 的顯示表示互通是因為他顯示其他裝置”is alive”，這就表示 Ubuntu 系統的電腦有回應。最後”tftp a0000000 test.bin” 這條指令要求 LS1043A 處理器對 Ubuntu 系統的電腦中的 tftp 服務器發出請求，請求下載測試檔案 (test.bin)到定址 a0000000。由下圖 4-1 可以發現，指令執行成功，檔案被寫入 a0000000，由此證明數據通道已經可以正常工作。

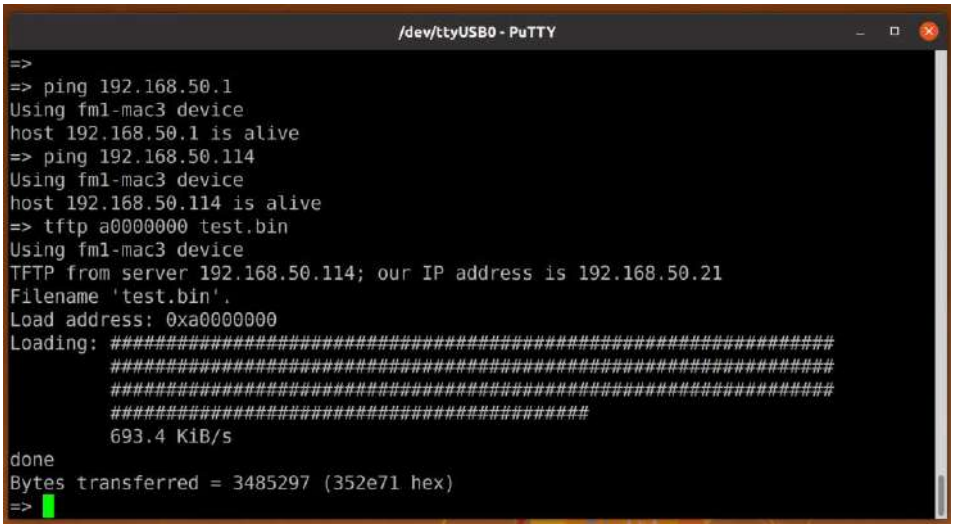


圖 4-1、透過三條指令證明數據通道可以使用(自行拍攝)

五、關於 LS1043A 處理器 System Memory Map 的初步概念

以下是我的實驗筆記。Memory Map 的作用在於給每個 CPU 周圍的裝置一個特定的地址。對於 LS1043A 來說，LS1043A 要訪問 CPU 外圍的各個裝置需要一個定址，並透過 Memory Map 上不同區塊的地址去訪問電路板上不同的裝置，這些裝置包含:DDR 以及 NOR Flash 等。LS1043A System Memory Map 如下圖 5-1 所示，可以根據下表來確認各個裝置對應的地址。

Start Physical Address	End Physical Address	Memory Type	Size
0x00_0000_0000	0x00_000F_FFFF	Secure Boot ROM	1 MB
0x00_0100_0000	0x00_0FFF_FFFF	CCSRBAR	240 MB
0x00_1000_0000	0x00_1000_FFFF	OCRAM0	64 kB
0x00_1001_0000	0x00_1001_FFFF	OCRAM1	64 kB
0x00_2000_0000	0x00_20FF_FFFF	DCSR	16 MB
0x00_6000_0000	0x00_67FF_FFFF	IFC - NOR Flash	128 MB
0x00_7E80_0000	0x00_7E80_FFFF	IFC - NAND Flash	64 kB
0x00_7FB0_0000	0x00_7FB0_0FFF	IFC - FPGA	4 kB
0x00_8000_0000	0x00_FFFF_FFFF	DRAM1	2 GB

圖 5-1、LS1043A System Memory Map(自行拍攝)

例如: 當 LS1043A 要訪問 NOR Flash 的時候，需要在 Bootloader 的指令下使用以下指令"md 60000000"，md 指令的意思是 memory display，"60000000"是要訪問的地址。因為"0x60000000"在 memory map 上是指向 NOR Flash，如圖 5-1 所示，所以訪問這個地址就代表要訪問 NOR Flash。如下圖 5-2 所示，若看到”55aa55aa”就代表是正確的因為”55aa55aa”為開機的第一條指令。而”0x60000000”為開機讀取第一條指令的地址。

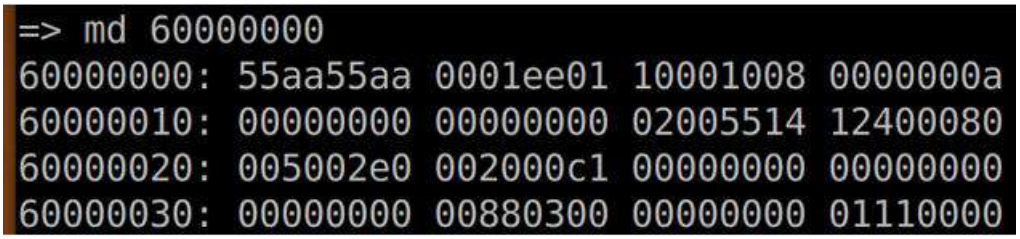


圖 5-2、NOR Flash 的地址以及所儲存的數據內容(自行拍攝)

例如: 當 LS1043A 要訪問 DDR 的時候，需要在 Bootloader 的指令下使用以下指令 “md 80000000”，md 指令的意思是 memory display， “80000000”是要訪問的地址。因為“0x80000000”在 memory map 上是指向 DDR，所以訪問這個地址就代表要訪問 DDR。在更進一步的應用中，後續將會利用 Bootloader 底下 Copy 的指令來更新 Bootloader 的版本，我們使用的指令是 “cp.b a0000000 60000000 $filesize”，這行指令要求 CPU 從地址 a0000000 將數據複製到地址 60000000，其中 a0000000 指向 DDR，而 60000000 指向 NOR Flash。所以這行指令的意思是，複製一次 Bootloader 的版本並從 DDR 將新的數據傳進 NOR Flash 內。

【評語】100035

1. 此一作品設計一計時裝置以量測處理器之開機時間。作品針對所研議之裝置有詳盡描述但若以量測開機時間之目的則應有效且精確地紀錄開始與結束之觸發時間點且評估量測過程之誤差。

2. 相較於此複雜的方式量測系統開機時間，是否可能有更簡單的方式可以得到更準確的結果？例如以另一台電腦透過網路觸發開機程序或是系統供電，扣除可量測的網路延遲時間甚或開關反應時間後，可以得到起始時間；在開機程序結束後，馬上用一個簡單的程式透過網路等傳遞完成信號回控制電腦，扣除網路延遲與程式執行所需時間後，可得到完成時間，並計算開機所耗時間。

3. 建議可針對本研究之應用標的與效益加強說明。