嵌入式系統開機程式：RedBoot

陳思豪　曹世強　林盈達

國立交通大學資訊科學系
新竹市大學路1001 號
TEL：（03）5712121 EXT. 56667
E-MAIL：{cschen, weafon, ydlin}@cis.nctu.edu.tw

摘要

開機程式是系統啓動後的第一個執行的程式，其背負著將硬體初始與讀進作業系統的重大使命。開機程式的工作，在一般的 PC 上，大都分爲 BIOS 與 BootManager(開機管理員)兩段程式來完成，但是在嵌入式系統上因一般來說無 BIOS 的設計，因此通常以一段程式完成。我們通稱此程式爲 Bootloader（開機程式）。Redboot 是一個常用在嵌入式系統上的開機程式，其運作在 Embedded Configurable Operating System (eCos)這套作業系統的硬體描述層(Hardware Abstraction Layer, HAL) 上。eCos 是套以高度「可設定性」及「模組化」爲目標發展的嵌入式作業系統，Redboot 可視爲在其上開發的一套應用程式。eCos 與 Redboot 是個開放源碼的計畫，使用者不但可輕易的得到其原始碼，且具備 Buyout-Free 與 loyalty-Free 等優點，且可以從眾多的貢獻者處得到移植到新平台的程式碼，所以其支援廣泛的硬體，相當具備有產品價值。

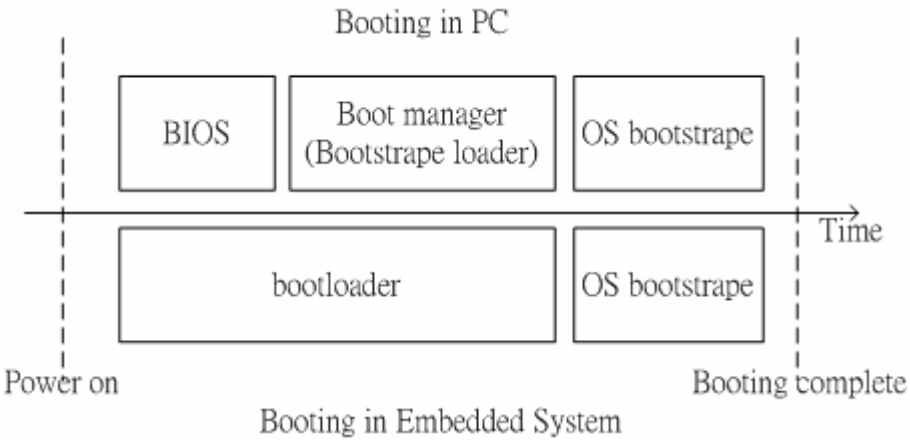
關鍵字： 開機程式、bootloader、RedBoot、eCos

1 簡介

我們時常聽到 BIOS、BootManager、Bootstrap、Bootloader 這幾個跟開機程式有關的名詞，但卻分不出到底有哪些差異。以一般常見的 x86 PC 系統爲例，所謂的 BIOS(Basic input output system 基本輸出入系統) ，是指存在主機板上的 flash ROM 上，負責許多與硬體溝通和初始化硬體的動作的程式。幾乎每張主機板都要有針對其特別寫的 BIOS。我們一般用其設定一些系統參數如 CPU 頻率、RAM 的速度及抓取硬碟，光碟等周邊。而所謂 BootManager(開機管理員)，其不像 BIOS 般，與硬體結合得如此緊密。在一般的 x86 機器上，一套 BootManager 寫好後可套用在所有的 x86 機器上，大部分存在硬碟主要開機磁區(Master Boot Record, MBR)。BootManager 的工作是將 OS 的核心(Kernel)載入，有時還可從多個硬碟分割區核心載入。例如在同一台電腦上若同時裝上 linux 與 Windows，我

們就可以靠著一套存在於 MBR 的 BootManager 來選擇要載入 linux 或是 Windows。Bootstrap 就是我們利用了 BootManager 載入 kernel 後，OS 開始做一堆的初始動作直到我們可以開始使用，在這期間的一連串動作。所以 BootManager 也常稱爲 Bootstrap Loader。至於 Bootloader（開機程式）其實跟前面 所提的 BootManager 幾乎一樣（也有許多人將這兩詞視爲同意），主要任務就是要讀進 kernel 使系統進入 bootstrap 的動作。只是在嵌入式系統中常把 BIOS 與 BootManager 做在一起，且也不需要選擇 OS 這種功能，而把從電源一開啓到讀進 kernel 的所有動作統歸爲 bootloader 所負責。

所以在一般的 PC 架構中，我們常把開機分三個階段 BIOS, BootManager, OS。像是 LILO (LInux boot LOader) and Grub (GRand Unified Bootloader) 這兩個都是在 linux 上廣泛的 bootmanager。他們負責透過 BIOS 所提供的 IO 功能，OS 從軟碟或是硬碟讀進來執行，並具有多重 OS 選擇功能，可開啓所有在 x86 上的 OS。LILO 是歷史悠久、簡單易用，而 Grub 提供了許多比 LILO 更強大的功能，逐漸取代 LILO 的地位。而在嵌入式系統中則只分兩階段 Bootloader 及 OS。常見的 *RedBoot* (RedHat Embedded Debug and Bootstrap)便是一個所謂的 Bootloader，它也是本篇要介紹的主角。他能夠提供嵌入式系統上的開機與除錯的工作。由於沒有 BIOS 的幫忙，原本 Bootloader 勢必有許多和硬體平台相依的動作。RedBoot 在這邊藉由利用 eCos[2-5]這套嵌入式 OS 所提供的 HAL 而能夠達到可一般 BIOS 所做到的功能。某種層面來看，Redboot 也可以說是一套 eCos 上的應用程式。不過其可載入各種在嵌入式系統上的應用程式，不限於是否是使用 eCos 開發的。圖 1 表示一般 PC 架構與嵌入式系統差異



: 圖 1. 一般 PC 架構與嵌入式系統的開機步驟差異

第二章介紹 eCos 這套嵌入式系統用的作業系統，包含其幾個重要開發觀念與功能，在其上開發的應用程式架構及其 HAL 在開機時對硬體所做的初始動作。第三章介紹 Redboot 的功能、介紹其架構及其與 eCos 的關係、其運作的流程、當要移植到一個新的平台時所要注意的步驟等。最後是我們的結論。

1. eCos 介紹

2.1 eCos 簡介

介紹 redboot 前，先介紹 eCos 這套針對嵌入式系統所設計的即時作業系統。eCos（Embedded Configurable Operating System），是個開放原碼的軟體，開發者可用 GNU 開放源碼發展工具（GCC、GDB 等）在其上的開發自己的應用程式，其專利受 eCos license 所保護，這是一個 GPL license 的修改版，其准許開發者在其上開發的應用程式（eCos 以外自行撰寫的部分）可以不用跟著 GPL 一起散佈。應用程式開發者可免費的取得其完整的源碼（buyout-free），並針對其作任意的修改與在其上開發自己的應用程式並散佈，唯一的限制只是若有修改到 eCos 本身，其需將修改的源碼回報給 eCos 開發小組。當開發者將其當爲產品時，也不需支付版稅（royalty-free）。其已直接支援了時下絕大部分的硬體，可在 eCos 官方網站[2]上找到支援列表。

2.2 eCos 功能概觀

高可設定性

eCos 最大的特色就是字面上 C 表示的「高可設定性」。eCos 可以讓開發者像在玩積木般地自由選擇其執行期的元件，應用程式開發者可以針對自己的應用程式來設定一個對其最小的作業系統環境，這跟以往應用程式就是跑在一個完整的作業系統上本質上不同，在嵌入式系統資源與記憶體寸土寸金的環境上，這樣的開發方式是很重要的。在以往的嵌入式開發方式都是自己手工的將作業系統作縮減[1]，對經驗不足或對該作業系統不夠熟悉的人將會花去許多時間，或是根本很難將作業系統拆開，但在 eCos 上，由於設計之初就是朝向可設定的原則，各種元件都遵守著模組化的開發方式，而應用程式開發者只要使用 eCos 所附的設定化工具（圖 2），即可輕鬆簡單的對 eCos 元件作量身打造，也不需對其內部實作有所瞭解。

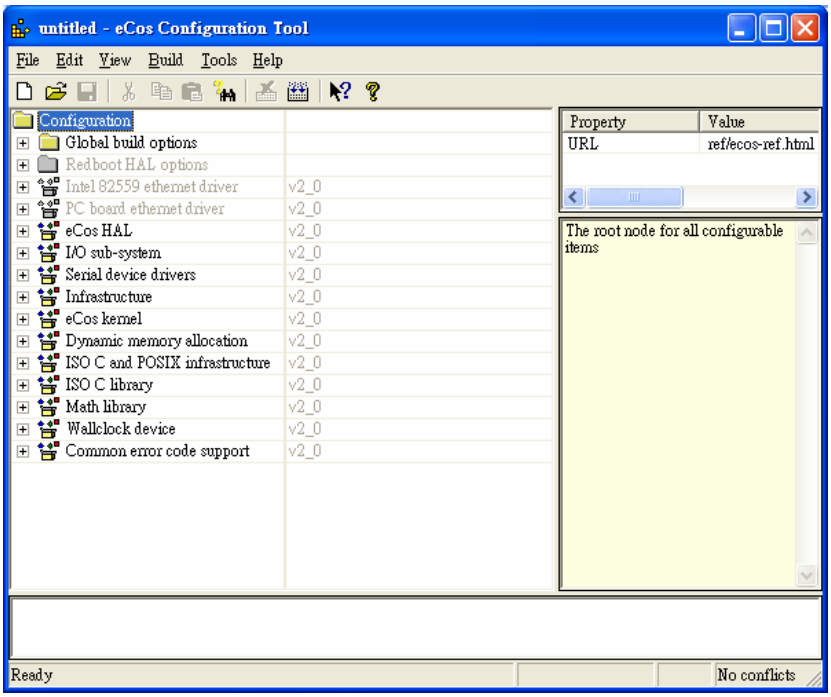


圖 2　eCos 設定工具

即時作業系統的核心並提供標準系統 API

eCos 的核心支援一般 OS 常見的項目如裝置驅動程式（Device Driver）、記憶體管理（Memory managemant）、例外處理（exception handling）、中斷處理（exception handling）、執行緒的支援（thread support）、計時器（Timer）、計數器（Counter），對於即時作業系統的支援如完全優先（full preemptability）、最小中斷延遲（minimal interrupt latencies）、多樣的同步方法（synchronization primitive）、可自訂的排程原則（schedule policies）。此外也支援 POSIX 等作業系統的標準 API 及 ANSI C 與常用的數學函式。

支援常用的周邊及通訊協定組(networking stacks)

支援以太網路卡，串列埠，USB slave（被動式 USB）等常用周邊。並支援一般常用的通訊協定組如 IP、IPV6、ICMP、UDP、TCP、SNMP、HTTP、TFTP、FTP 等。網路設定部分，可將靜態 IP 寫死在程式中，或是透過 DHCP 自己抓取

GDB 除錯支援

可支援主控端使用 GDB 遠端透過序列埠或是以太網路對應用程式除錯。

2.3　eCos 上開發的應用程式架構

圖 3 中 File system 指的是對檔案系統如 ext2 等的支援，library 是上節所提包括 POSIX，ANSI C 等的函式庫。這張圖，由上到下，表示從高層到底層的 eCos 架構。最底層的是我們的硬體，在硬體上面有 HAL 與裝置驅動程式，而我們大部分會利用 eCos 設定工具去設定中間 kernel、networking stack、library 那層（OS 層），只留下我們需要的部分。最上層的應用程式就是我們自行撰寫的部分，透過中間 OS 層的輔助來達成我們的目的。由這張圖，我們可以看出，Redboot 是一個架構在 eCos HAL 與 Device Dr

其中與硬體最關係密切的就是 HAL，可以用「最接近硬體的軟體」來形容，HAL 將所有與硬體相關的地方對外隱藏在裡面。針對不同的硬體時，只需抽換掉 HAL，換上針對新硬體而撰寫的 HAL 即可。下節將以 x86 爲例，說明 eCos HAL 在開機後作了哪些動作。

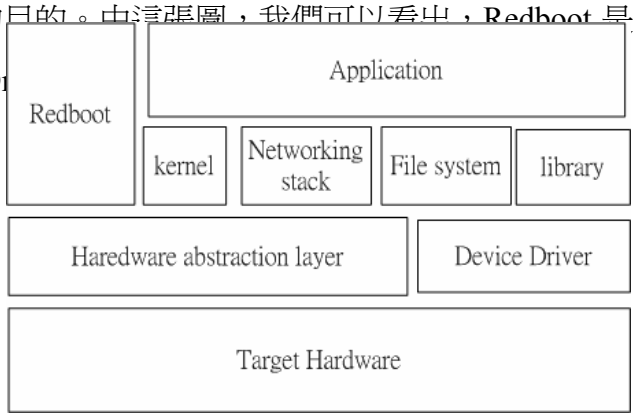


圖 3　eCos 應用程式架構圖

2.4 eCos HAL

表 1，以 x86 的平台爲例，說明 eCos HAL 在開機時，會進行哪些細部動作。這個起始化的動作的主要目的是要初始 CPU、RAM、週邊，使在 HAL 之後的程式可正確無誤的執行。。

	步驟名稱	解釋
1.	CPU INIT	初始 CPU，這裡主要是對一些與 CPU 有關的暫存器作一些處理。
2.	SMP INIT	對 SMP（symmetric multiprocessor 對稱式多處理器）的支援，在某些平台這部分會拿掉。
3.	DIAG INIT	作定義一些診斷系統目前狀態用的巨集。
4.	MMU INIT	對記憶體管理所做的初始化，主要是爲了處理虛擬記憶體位置與真實記憶體位置間的轉換。
5.	MEMC INIT	在 x86 系統中會開啓 A20 gate，這個與 x86 記憶體定址有悠久歷史淵源的東西。
6.	INTC INIT	將所有的中斷預設爲關閉。
7.	CACHE INIT	將 CPU 的快取系統作初始。
8.	TIMER INIT	初始系統的計時器。
9.	IDT INIT	設定 IDT（Interrupt Descriptor Table 中斷描述表格）以初

		始中斷堆疊區段，以後有中斷發生時就會使用到這些區段。
10.	MON INIT	當程式是放在 ROM 中被執行時，這個函式會去確定所有例外情形的例外處理器是否皆已裝設。
11.	FPU INIT	初始許多與浮點運算單元相關的暫存器。
12.	BSS INIT	會去清除記憶體中的 BSS 區段，BSS 會放所有被宣告成靜態（static）的變數。
13.	ROM INIT	在這邊會檢查程式本身是否被放在 ROM 上被執行，如果是的話需要將程式中的資料區段拷貝到 RAM 上。
14.	STACK INIT	設定堆疊使我們可以 call c 函式。在這個步驟之前執行的全是組合語言，這步過後就可以開始呼叫 C 的函式了。
15.	PLATFROM INIT	初始 virtual vector table，virtual vector 是使用在 eCos 上用來指向許多資料與函式的指標。
16.	GDB_STUB INIT	如果需要有與主控端的 GDB 作除錯的動作的話，就會安裝與除錯相關的 trap 與初始相關硬體。
17.	GOTO PROGRAM	總算 HAL 的初始硬體的部分大功告成，這時已經具備足夠的環境給 Redboot 此類應用程式了。

表 1　　eCos HAL 初始步驟

2. RedBoot 介紹

3.1 功能

- **初始 CPU ,RAM 及週邊介面:**

這裡就相當於 x86 上 BIOS 的功能，初始 CPU 包括對於 SMP(Symmetric Multiprocessing 對稱式多處理器)、FPU(Floating point unit 浮點運算單元)、Cache（快取）等初始動作。 RAM 大部分是決定其大小及決定要使用哪些區段。週邊是如網路卡個讓 OS 或應用程式(Application, AP)執行的環境:

- **初始一個讓 OS 或應用程式(Application, AP)執行的環境:**

許多 OS 或是 Application 是用 C 寫成的，我們熟悉的開發環境也是 C 或是 JAVA 這類高階語言，但是實際上要執行這些語言寫成的程式都必須有環境，如 JAVA 需要 JVM、C 需要初始 stack，而 bootloader 又是第一個執行的程式，因此 bootloader 前面初始化的部分都是用組語寫成。直到它初始化了一個可以如 C 這種高階語言可以執行的環境後，之後才可呼叫用 C 寫成的函式。不過許多的 OS 會自己做這些動作（因爲這就是 OS 的所負責的），bootloader 大部分是爲了讓自己後來執行的功能造出最低限度的環境即可。除了上面以外，如果 bootloader 自己需要或是任何讀進 OS 前有需要用到 IO 的動作，bootloader 就必須提供 IO 的介面與環境。

■ **將 OS 或 AP 載入記憶體執行**

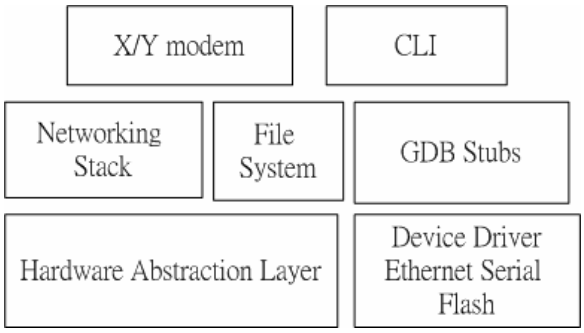
這是 bootloader 存在的意義，它初始化系統就是爲了做這件事，把我們要執行的東西從某處，可能是 ROM 或是硬碟或是軟體或光碟，將其放到 RAM 中，並執行它。做了執行的動作後，如果被執行的是個 OS，一般來說就不會回來了，主控權就交給了 OS，自己就功成身退了。以 Redboot 來說，可將我們開發好的 AP 從以太網路、序列埠（透過 X/Y modem）、USB、軟碟、硬碟等，各種經過正常初始化的裝置，下載到目標硬體的記憶體中執行。至於以太網路的設定，除能事先設定外，更可支援在開機時，以 BOOTP/DHCP 取得。

Redboot 當然也能支援多重 OS 開機及開機執行批次檔（booting script file）的功能。當我們確定開機後會有一連串將被執行時，可以在設定 Redboot 時，便直接寫死在裡面，這樣 Redboot 就會自動在開機時，執行那一連串的指令。

■ **除錯器支援與應用程式監控**

而許多用在嵌入式系統上的 bootloader 如 Redboot 就加上了 GDB 的功能，可以透過序列埠或是透過區域網路來與主控端溝通，主控端通常爲一台 PC，可以透過螢幕對載入執行的 AP 或是 OS 做監控與除錯的動作。Redboot 實作了 eCos 中關於 GDB 連接器的部分，當起始了週邊，如以太網路卡或是序列埠後，主控端可以遠端透過這些介面，用 GDB 對我們載入的程式作除錯的動作。

此外 Redboot 內建了一個類似 UNIX 系統上 shell 的指令式介面(command line interface)，使用者可以透過本地端鍵盤、以太網路、串列埠對 Redboot 下命令，常用的例如上述從指定的周邊載入程式到記憶體中、執行記憶體某個區段的程式、秀出某段記憶體區段等。這個介面還支援了別名（alias）的功能，可替較長或較難記的指令設定別的指令名。



3.2 架構

圖 4 大部分的元件都是在上面的功能概觀上提過的。這個圖就是圖一中把 RedBoot 區塊展開，再加上底層如 HAL 的部分。底層最接近硬體的部分爲 HAL

與裝置驅動程式如以太網卡、序列埠、Flash ROM 的驅動等，中間這層即前面提到的 OS 層，是由 eCos 精簡而來。上面的 CLI 與 X/Y modem 可看成 RedBoot 特別另外撰寫，也就是應用程式層的部分。

圖 4.　Redboot 架構圖

3.3 程式流程

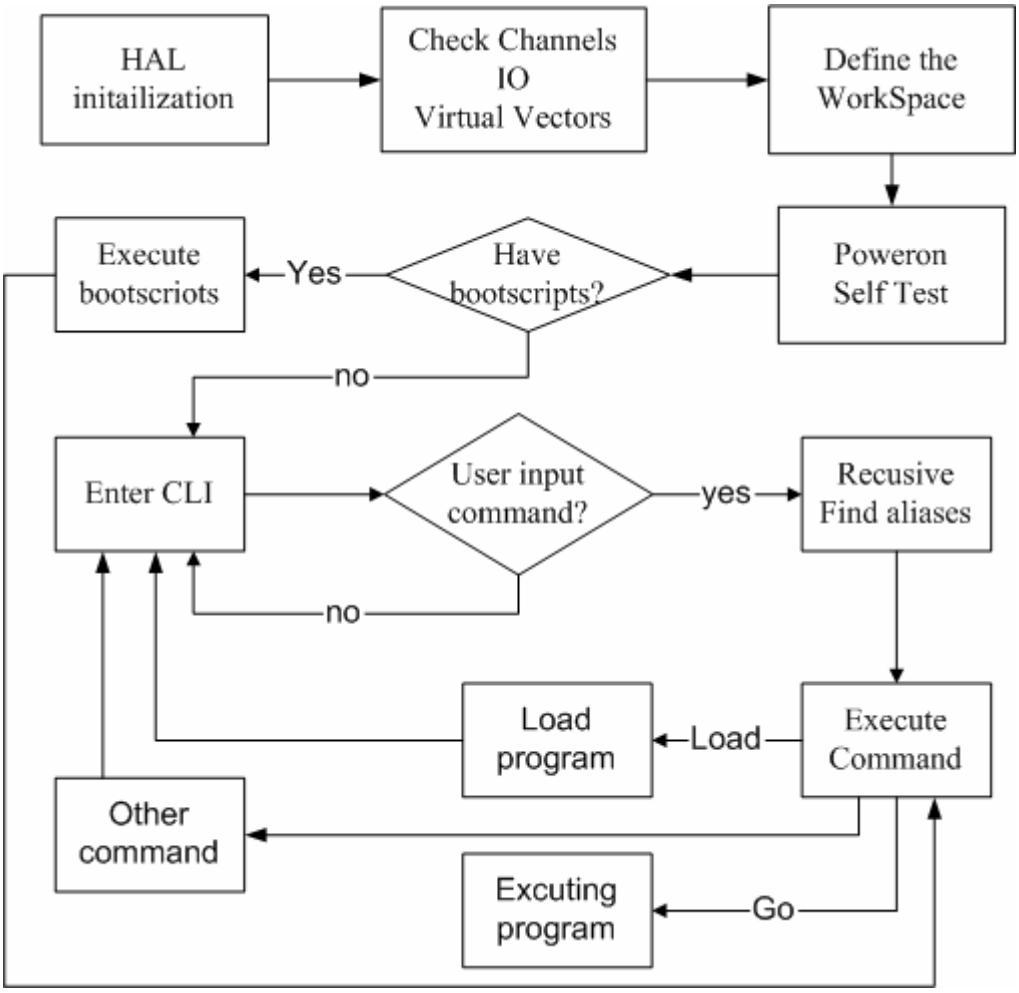


圖 5 Redboot 程式流程圖

圖五顯示 Redboot 的程式運作流程圖。首先，是我們之前 3.1 節介紹過的 HAL 起始化，再來會去檢查系統的 IO、virtual vector 等是否正確初始化了，接著定義 RAM 中的工作區域，然後作開機後測試(Power On Self Test, POST)，就像一般 PC 上看到 BIOS 在作測試 RAM 那類的動作。接下來會執行開機執行序列批次檔，與進入 CLI。在使用者輸入指令後，Redboot 遞迴的找尋別名，接著執行該命令。這張圖列出 Load、Go 與 Other 三種指令的流程，Load 就是將程式透過前面所述的各種介面讀入記憶體中，Go 是執行程式，一般來說開始執行程式後就不會跳回開機程式了，有時會看到可以按 ctrl+c 從程式跳出來的情形，這大部分是靠著攔截鍵盤中斷，實作出來的。

3.4 平台移植

Redboot 已經支援了許多硬體了，但若我們要將其移植到一個全新的硬體時，我們也只要更改與硬體相關的部分就可以了。在 eCos 中，與硬體相關的部分只有圖一中第二層的 HAL 與驅動程式裝置，而以上的核心、函式庫、網路部分的實作幾乎都是不需更動的。比起移植整套的 eCos 到新的平台，如果光是移植 Redboot 的話，那麼不需完成全部的 HAL 也能讓 Redboot 跑起來，並可確保基本部分硬體都有正常運作。Redboot 啓動後，我們就可透過 GDB 遠端除錯，所以移植 Redboot 也可當成是移植任何 eCos 應用程式的前置作業。

在移植 Redboot 時，我們會先試著將其以從 RAM 啓動的方式試著執行，這裡的意思就是在 Reboot 前還存在一個別的開機程式或是 BIOS 之類的另一套軟體來將 Redboot 讀入並執行，其中就差在是否 Redboot 自己可將系統作初始化而不用原有的開機程式或 BIOS 來作，如果 Redboot 已經可以完全的取代了原來的開機程式來放到 ROM 中當成唯一的開機程式時，那麼我們就稱其爲 ROM Monitor。

針對新的硬體編寫 HAL，必須只少看懂 eCos 其中一套的 HAL 實作方式，並對目標硬體有相當的瞭解，且需要許多經驗的累積，移植程式到新硬體上的工作都是很苦的，沒什麼速成的指引，以下也只是列出概念上幾個必須完成的事。

1. 先找出一個與目標硬體最接近的成品 HAL 當作我們移植的樣版。
2. 更改 HAL 中的記憶體配置使其與目標硬體相同。
3. 更改 HAL 中 IO 的部分。
4. 關於快取與記憶體管理單元的部分對 Redboot 來說還不太需要，爲了將問題簡單化可先把這麻煩的部分先拿掉。
5. 至少要實作一個可讓我們與其溝通的介面，如以太網路或序列埠。
6. 實作與平台相關的一些初始化動作(如 1.4 節所述)，這項就是前述的 Redboot 是否可以成爲該平台的 ROM monitor 的條件，反之若只是被讀進 RAM 執行的話，這個步驟就還不需作。

如果以 RAM 啓動方式已經可的話，可以試著將原有的 Bootloader 備份下來，換上 Redboot 看看其是否可以當 Rom monitor，這部分除錯如沒有 ICE 之類的硬體除錯器會相當辛苦。成功的話，表示已經有了一份基礎的 HAL 成形了。

3. 結論

嵌入式系統用的開機程式不需要和一般 PC 用開機程式一樣分成兩階段，因爲程式的來源通常是固定的，不能也不需要讓使用者選擇。而能夠存放在 ROM 中，第一個被系統執行的軟體，又能夠讀入其他程式並執行者，我們稱其爲該目標硬體的 ROM monitor，具有初始化硬體的能力，其中定具有可使用在此目標硬體的上 HAL。RedBoot 結合 eCos 的 HAL 後是一套完整的嵌入式系統用開機程

式，其支援多種硬體，又可對讀進來的程式作監控除錯的動作，對於開發階段來說是不可或缺的工具。而以產品化而言，開發者可以輕易取得完整程式碼作修改，再搭配 eCos 設定工具我們可輕鬆打造出精簡短小最適合我們用途的開機程式。要將 RedBoot 移植到全新的平台上時，由於硬體相關與非硬體相關的部分分的非常清楚，所以只要針對新的平台重新撰寫 HAL 即可。

4. 參考資料

[1] 梁元彪、林盈達；「Linux 嵌入式系統減肥實作」； 網路通訊；134 期，2002 年 9 月。

[2] eCos web site , http://sources.redhat.com/ecos/

[3] eCos v2.0 Documentation ,"User Guide", http://sources.redhat.com/ecos/docs-2.0/user-guide/ecos-user-guide.html

[4] eCos v2.0 Documentation , "eCos Reference Manual" , http://sources.redhat.com/ecos/docs-2.0/ref/ecos-ref.html

[5] Anthony Massa, "Embedded software development with eCos" , http://www.phptr.com/browse/product.asp?product_id={0BEE58C4-0812-4B13-9B8C-A0FC31A4C02F}