東海大學資訊工程學系研究所

碩士論文

Department of Computer Science Engineering

Tunghai University

多核心嵌入式軟體設計工具系統之架構支援實作

The architecture mapping support of a real time multi-core embedded software design tool development framework

研究生：吳建庭

Graduate Student: Chien-Ting Wu

指導教授：石志雄博士

Advisor: Chihhsiong Shih, Ph. D.

中華民國九十九年六月

June, 2010

摘要

現今的嵌入式系統在資訊家電的帶領下，已經逐漸擺脫過去比較小巧簡單的裝置，進入現在複雜多功能而且相當強悍的裝置，因而造成嵌入式系統的盛行，嵌入式作業系統也隨之變的非常多元化，因而造就了嵌入式多樣且架構多元的作業系統平台，然而其所延伸發展出來的相關應用軟體更是不勝枚舉。看準了此多元化的特性，因而發展出透過此研究的目的來達到跨平台的為設計目標。在上述的觀念下，本研究規劃了一個支援多核心嵌入式軟體設計開發環境，來支援多核心嵌入式軟體開發。主要著重於架構對應及整合後工具組之實際應用，首要之目標將嵌入式軟體規格，實作出自動淬取及合成核心及對應硬體及驅動軟體之類別程式庫可重用物件檔，其中包含了 xml 檔剖析功能，使用 UML 的技術來架構出框架與可重用元件模組化方式開發，增加未來的可擴充性。搭配上完整的自動化流程，簡化了整體工作的複雜度，大幅度減少人為因素的錯誤發生。最終產生 Makefile 目標執行檔，可順利編譯一多核心

程式並產出多重實驗數據。這樣的架構支援實作將與 VERTAF 中的多核心嵌入式軟體之合成器在程式碼生成時做重要的結合並配合與系統資源調配時使用，促使整體完善對應於硬體平台，達到自動化淬取及合成核心。研究成果可以直接減少人力參與繁複多核心程式設計的程度，不只具有提昇數位產業產值之潛力，同時在學術上亦可驗證設計樣式及軟工理論對多核心應用之影響。

Abstract

With the proliferation of multicore architectures for embedded processors such as ARM's Cortex-A9 MPCore [34], Intel's Core™ 2 Duo E4300, T7500, T7400, L7500, L7400, U7500 [35], Quad-Core Intel Xeon Processor E5300 series [36], multicore programming for embedded systems is no longer a luxury, but has become a necessity. We need embedded software engineers to be adept in programming such processors; however, the reality is that very few engineers know how to program them. The current state-of-the-art technology in multicore programming is based on the use of language extensions such as OpenMP [37] or libraries such as Intel *Threading Building Block* (TBB)[20]. Both OpenMP and TBB are very useful when programmers are already experts in multithreading and multicore programming[23]; however, for the vast majority of programmers and embedded software designers, there still exists a tremendous challenge in this urgent transition from unicore systems to multicore systems. To aid embedded software designers in a smoother transition, our primary goal will be *model-driven architecture* (MDA)[31] development for such software. In this thesis, we focus on architecture mapping where functional

requirements from embedded system designer is converted to a meta model for input to the quantum platform in the subproject 4 [1] for embedded parallel code generation. A SysML tool[31][32], Papyrus, is used to model the architecture input parameters. An XML parser[10] is then implemented to parse the architecture information embedded in the XML file generated from the Papyrus interface. The goal is to reduce human efforts and prevent user mistakes in generating a software compilation tool: makefile[17]. The makefile contains both the compilation info. and system performance tuning info. By coping with such an automatic flow, user can not only benefit for a reduced time, this process can also help optimize the multicore system performance.

This thesis provides the opportunity to explore the in depth knowledge of various theories used, e.g. parallel programming, multicore processor, optical flow, etc. It also has direct impact to the emerging industries such as multicore embedded system and vehicle communication industry.

目錄

圖次

表次

第一章　前言

近年來多核心架構的資訊產品不斷成長，嵌入式軟體及相關服務軟體產品是決定硬體是否實用的關鍵因素。其市面上的嵌入式作業系統非常的多元化，像是在嵌入式領域耕耘已久的 VxWORK、QNX、Nucleu…等等之外，還有新興的主要競爭產品包括 Palm OS、Windows CE、Linux 等多款嵌入式作業系統，其中以 Embedded Linux 作業系統免費授權的特性，已為業界國際大廠或學術界授課研究所採用。

嵌入式作業系統的種類相當繁多其所發展出來的應用軟體更是不勝枚舉。程式開發人員在開發軟體過程，難免會遇到系統平台與嵌入式硬體裝置的限制。嵌入式硬體裝置推陳出新的速度相當快，然後之前的所開發出的應用軟體就被迫跟著淘汰，必須從頭開始做新的開發。而要軟體系統在多核心架構的嵌入式資訊產品上發揮效能，多核心程式開發設計是未來軟體資訊產業發展的趨勢。但目前多核心程式開發設計的工程師是較不普遍，嵌入式產品相關軟體需求也越來越多，以軟體工程的技術及軟體開發工具的導入是迫切需要的。

在上述的觀念下，本研究規畫了一個支援多核心嵌入式軟體設計開發環境，來支援多核心嵌入式軟體開發。主要著重於架構對應及整合後工具組之實際應用[1]，首要目標為將嵌入式多核心軟體規格，經由

thread-mapping、thread-scheduling [2][4]正式的驗證以後，實作一自動淬取及合成核心。其中包含了.xml 檔剖析功能，實作各對應硬體及驅動軟體之類別程式庫物件檔。同時將設計一對應之轉換模組以將嵌入式多核心軟體規格中每一類別自動轉換成 active object。此為中繼模型(meta model)。此模型為產生程式碼之 Quantum Framework [8][9] 所需之前置作業。研究過程將軟體工程上層產生之嵌入式多核心軟體規格，實作各對應硬體及驅動軟體之類別程式庫物件檔。並轉製成一 Automake 檔[7]，產出之 Makefile 可順利編譯一多核心程式並產出多重實驗數據。

第二章　文獻探討

1. Unified Modeling Language and System Modeling Language

UML [5]是目前工業界最廣泛使用的塑模語言之一，讓設計師以 UML 為輸入的模形，可以減少設計師建立模型或學習新語言的時間，並且利用UML 的特性完整的描述系統的行為。物件導向技術使得軟體元件能夠擁有重複使用性，設計師不需要花很多的時間去適應新的平台，他可以直接使用現存的軟體元件加以整合，大大地減低了開發的時間。軟體排程技術確保產生的程式能夠符合使用者指定的時間限制，對於即時嵌入式系統的設計有很大的幫助。軟體驗證技術可以讓設計者經由正規驗證的方式，完整地驗證系統行為的正確性，避免系統執行可能發生的錯誤。自動程式碼生成技術讓使用者可以減少自行撰寫程式碼的需要，可以減少人為的程式錯誤及撰寫程式碼的時間。

本來用來作為一種軟體設計和模型語言，UML 現在很密集而且廣泛地使用在系統設計中。這可從最近 UML 2.0 版和廣被宣揚的OMG系統模型語言(SysML)[6][32]使用在很多的系統模型上的實例中看出 。現在，UML 已經用於 IBM Rational Rose的模型驅動的之平行運算發展工具中，延伸 UML 產生平行程式碼和支持 OpenMP。在學術界或工業中少有研究UML如何能為多核心軟體之設計發展。 然而，我們相信這是最有希

望的未來趨勢。因為多執行緒對 unicore 處理器已是非常複雜的，如果我們繼續直接地規劃使用多行緒的範例，例如 TBB ，它將一定對多核心處理器的運用更增加其挑戰。

2. GNU Automake

無論是在 Linux 還是在 Unix 環境中，make [17]都是一個非常重要的編譯命令。不管是自己進行項目開發還是安裝應用軟件，我們都經常要用到 make或 make install。利用make工具，我們可以將大型的開發項目分解成為多個更易於管理的模塊，對於一個包括幾百個源文件的應用程序，使用 make 和 Makefile 工具就可以輕而易舉的理順各個源文件之間紛繁複雜的相互關係。

GNU Automake[7]是一種編程工具，可以產生供make程式使用的Makefile，用來編譯程式。它是自由軟體基金會所製作的GNU程式的其中一項，作為GNU建構系統的一部分。Automake 所產生的 Makefile符合GNU編程標準。典型的 Automake 輸入文件是一系列簡單的宏定義。處理所有這樣的文件以創建 Makefile.in 檔案。在一個專案的每個目錄中通常會包含一個 Makefile.am 檔案。

Automake在幾個方面對一個項目做了限制，並且對configure.in的內容施加了某些限制。為生成 Makefile.in ， Automake 需要perl。

但是由 Automake 創建的發佈完全服從 GNU 標準，並且在創建中不需要 perl 。 Automake 是一個從文件 Makefile.am 自動生成 Makefile.in 的工具。每個 Makefile.am 基本上是一系列make的宏定義。生成的 Makefile.in's 服從GNU Makefile 標準。以下一些基本通用性概念將有助於理解Automake。

通用操作

Automake 讀入 Makefile.am 檔案並且生成 Makefile.in 。在 Makefile.am 中定義的一些宏和目標（targets）指揮 Automake 生成更多特定的代碼；例如一個 bin_PROGRAMS 宏定義將生成一個需要被編譯、連接的目標。Makefile.am 中的宏定義和目標被複製到生成的文件中。這使得你可以把任何代碼添加到生成的‘Makefile.in 文件中。在 Makefile.am 中定義的目標通常覆蓋了所有由 Automake 自動生成的擁有相似名字的目標。

深度

目前Automake支持三種目錄層次：flat（平包）、shallow（深包）和deep（淺包）。

1）flat（平包）指的是所有文件都位於同一個目錄中。就是所有源文件、頭文件以及其他庫文件都位於當前目錄中，且沒有子目錄。

2）shallow（深包）指的是主要的源代碼都儲存在頂層目錄，其他各個部分則儲存在子目錄中。就是主要源文件在當前目錄中，而其它一些實現各部分功能的源文件位於各自不同的目錄。

3）deep（淺包）指的是所有源代碼都被儲存在子目錄中；頂層目錄主要包含配置信息。就是所有源文件及自己寫的頭文件位於當前目錄的一個子目錄中，而當前目錄裡沒有任何源文件。

統一命名機制

Automake 通常服從統一的命名機制，以易於確定如何創建和安裝程序。這個機制還支持在運行configure的時候確定應該創建那些對象。在運行make時，某些變量被用於確定應該創建那些對象。這些變量被稱為主變量。

依存性資訊產生

Automake 能夠自動生成依存性的資訊，因此，當一個 source 文件被修改，下次呼叫make命令的時候就會知道哪些source文件需要重新編譯。如果編譯器允許， Automake會試著讓依存性系統保持動態：無論何時source文件被編譯，都會要求編譯器重新產生依存性列表更新該文件的依存性。換句話說，依存性追蹤是編譯過程的一種邊際效應。

這企圖避免一些靜態依存性系統的問題，比如依存性只會在程式員

開始專案時才會被偵測到。在這種情況下，如果源文件獲得一個新的依存性(例如，如果程式員增加了一個新的＃include指令在C語言的source文件，這樣在真實的依存性和編譯系統所使用的依存性之間就會產生差異。）程式員應該重新產生依存性，但很有可能忘了那樣做。在一般情況下，Automake 透過隨附的 depcomp 腳本生成依存性，這會適當的呼叫編譯器或是回到 makedepend。如果gcc編譯器的版本夠新的話，Automake 將會inline依存性生成碼，直接呼叫gcc。

3. Make

Make[17]是一個自動轉化文件形式的工具，轉換的目標稱為target；與此同時，它也檢查文件的依賴關係，如果需要的話，它會調用一些外部軟體來完成任務。它的依賴關係檢查系統非常簡單，主要根據依賴文件的修改時間進行判斷。大多數情況下，它被用來編譯原始碼，生成結果代碼，然後把結果代碼連接起來生成可執行文件或者庫文件。它使用叫做 makefile 的文件來確定一個target文件的依賴關係，然後把生成這個target的相關命令傳給shell去執行所提供的任何命令來完成想要的工作。

4. Makefile

Makefile 關係到了整個工程的編譯規則。一個工程中的源文件不計

數，其按類型、功能、模塊分別放在若干個目錄中， Makefile 定義了一系列的規則來指定，哪些文件需要先編譯，哪些文件需要後編譯，哪些文件需要重新編譯，甚至於進行更複雜的功能操作，因為 Makefile 就像一個Shell腳本一樣，其中也可以執行操作系統的命令。

Makefile 基本上就是目標，關連，和動作三者所組成的一連串規則。而 make 就會根據 Makefile 的規則來決定如何編譯和連結（link）程式。Makefile 基本構造雖然簡單，但是妥善運用這些規則就也可以變出許多不同的花招。程式設計師只需寫一些預先定義好的巨集，交給 Automake 處理後會產生一個可供 Autoconf 使用的Makefile.in 檔。再配合利用 Autoconf 產生的自動設定檔 configure 即可產生一份符合 GNU Makefile 慣例的 Makeifle 了。

Makefile 帶來的好處就是自動化編譯，一旦寫好，只需要一個make命令，整個工程完全自動編譯，極大的提高了軟件開發的效率。

5. 可擴展標示語言

XML[14]全名是Extensible Markup Language，也就是延伸標記語言, 是用來描述資料的一種標記語言. 雖然現在整體的標準還有一點點不穩定的變數, 但是他對於未來的文件上技術層面的影響是十分令人期待的。XML和HTML一樣都是一種標記語言, 利用標籤來定義

各種屬性。HTML的使用目的是整理資料並排版, 修改資料的顯示; XML的使用目的則是描述資料。一個符合 XML 規格的軟體工具一定都支援 UTF-8 及 UTF-16 碼 (Unicode).但是任何的字碼集都有可能用來編寫 XML 資料。撰寫剖析器(parser) 的設計決定其 XML 軟體工具支援那些字碼集。

6. JDOM

JDOM[15]是一個開源項目，它基於樹型結構，利用純JAVA的技術對XML文檔實現解析、生成、序列化以及多種操作。JDOM 直接為JAVA編程服務。它利用更為強有力的JAVA語言的諸多特性（方法重載、集合概念以及映射），把SAX[13]和DOM[15]的功能有效地結合起來。在使用設計上盡可能地隱藏原來使用XML過程中的複雜性。利用JDOM處理XML文檔將是一件輕鬆、簡單的事。JDom 是不錯的API，算得上簡單高效，最重要是已經成為jcp的一部分，涵蓋了大部分的操作：元素、屬性、命名空間、PI、DTD、Schema。

7. 語法分析器

在計算機科學和語言學中，語法分析(parsing)是根據某種給定的形式文法(formal grammar)對輸入的單詞(token)序列進行分析並確定其語法結構的一種過程。而語法分析器通常是以編譯器或解釋器的組

件出現的，它的作用是從輸入中分析出其結構並將其轉換為在後續處理過程中更易於訪問的資料結構(一般是樹類的資料結構)，並檢測可能存在的語法錯誤。語法分析器通常使用一個詞法分析器(lexer)從輸入的字元流中分離出一個個的『單詞』，並將單詞流作為其輸入。在實際開發中，語法分析器可以手工編寫，也可以使用自動生成程序。詞法分析器階段的任務：從左至右逐個讀入源程序，對源程序的字元流進行掃描和分析，識別出是否為該類別程序語言的保留字，其他的單詞則標為用戶定義的標識符。另外，在詞法分析階段，可以分析程序的用戶自定義的標識符是否符合構詞規則。並表標識出行號位置。

8. Shell 腳本

shell腳本[33]是一個常見任務就是把各種不同的已有組件連接起來以完成相關任務。大多腳本語言共性是：良好的快速開發，高效率的執行，解釋而非編譯執行，和其它語言編寫的程序組件之間通信功能很強大。許多腳本語言用來執行一次性任務，尤其是系統管理方面。它可以把服務組件粘合起來，因此被廣泛用於GUI創建或者命令行，操作系統通常提供一些默認的腳本語言，即通常所謂shell腳本語言。腳本通常以文本(如ASCII)保存，只在被調用時進行解釋或編譯。

有些腳本是為了特定領域設計的，但通常腳本都可以寫更通用的腳本。在大型項目中經常把腳本和其它低級編程語言一起使用，各自發揮優勢解決特定問題。腳本經常用於設計互動通信，它有許多可以單獨執行的命令，可以做很高級的操作， 這些高級命令簡化了代碼編寫過程。諸如內存自動管理和溢出檢查等性能問題可以不用考慮。在更低級或非腳本語言中，內存及變量管理和數據結構等耗費人工，為解決一個給定問題需要大量代碼，當然這樣能夠獲得更為細緻的控制和優化。腳本缺少優化程序以提速或者降低內存的伸縮性。腳本通常是解釋執行的，速度可能很慢，且運行時更耗內存。在很多案例中，如編寫一些數十行的小腳本，它所帶來的編寫優勢就遠遠超過了運行時的劣勢，尤其是在當前程序員工資趨高和硬件成本趨低時。腳本語言中，有經驗的程序員可以進行大量優化工作。

第三章　研究方法

本研究方法，預定於完成相關文獻與理論之彙閱與研究，綜合先前研究成果，首先針對系統各相關子功能進行模組化之關連性探討、模組分析與設計、整合分析、原型發展後，設計一套具實用性質的工具與系統環境。

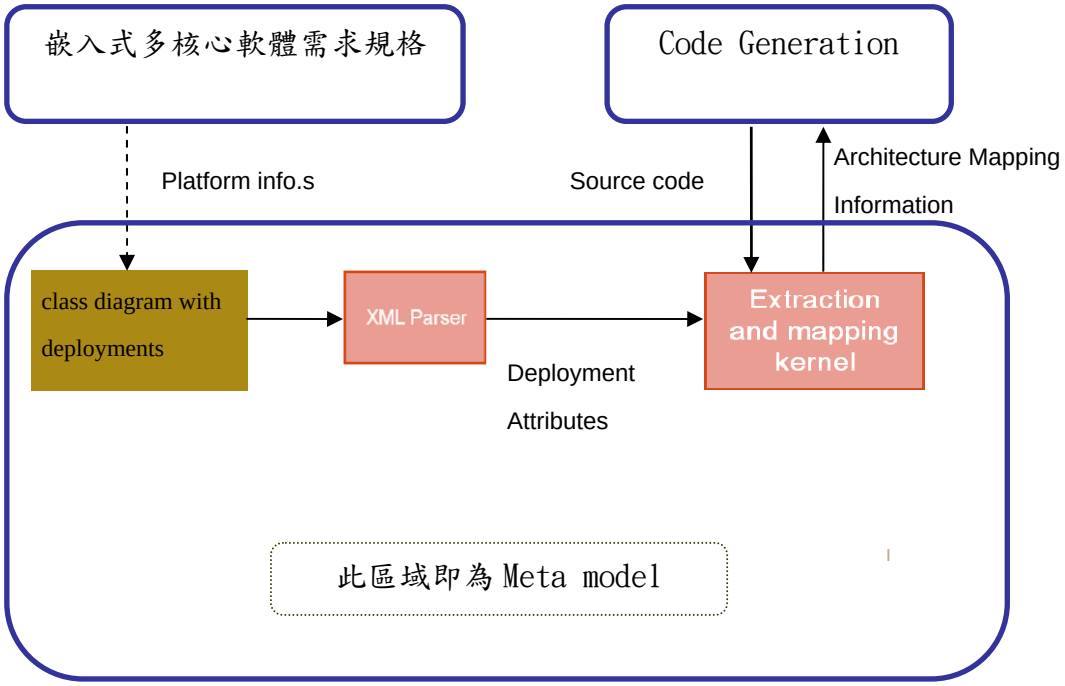


圖 1 為研究架構圖

3.1 中繼模型(meta model)建立方法流程

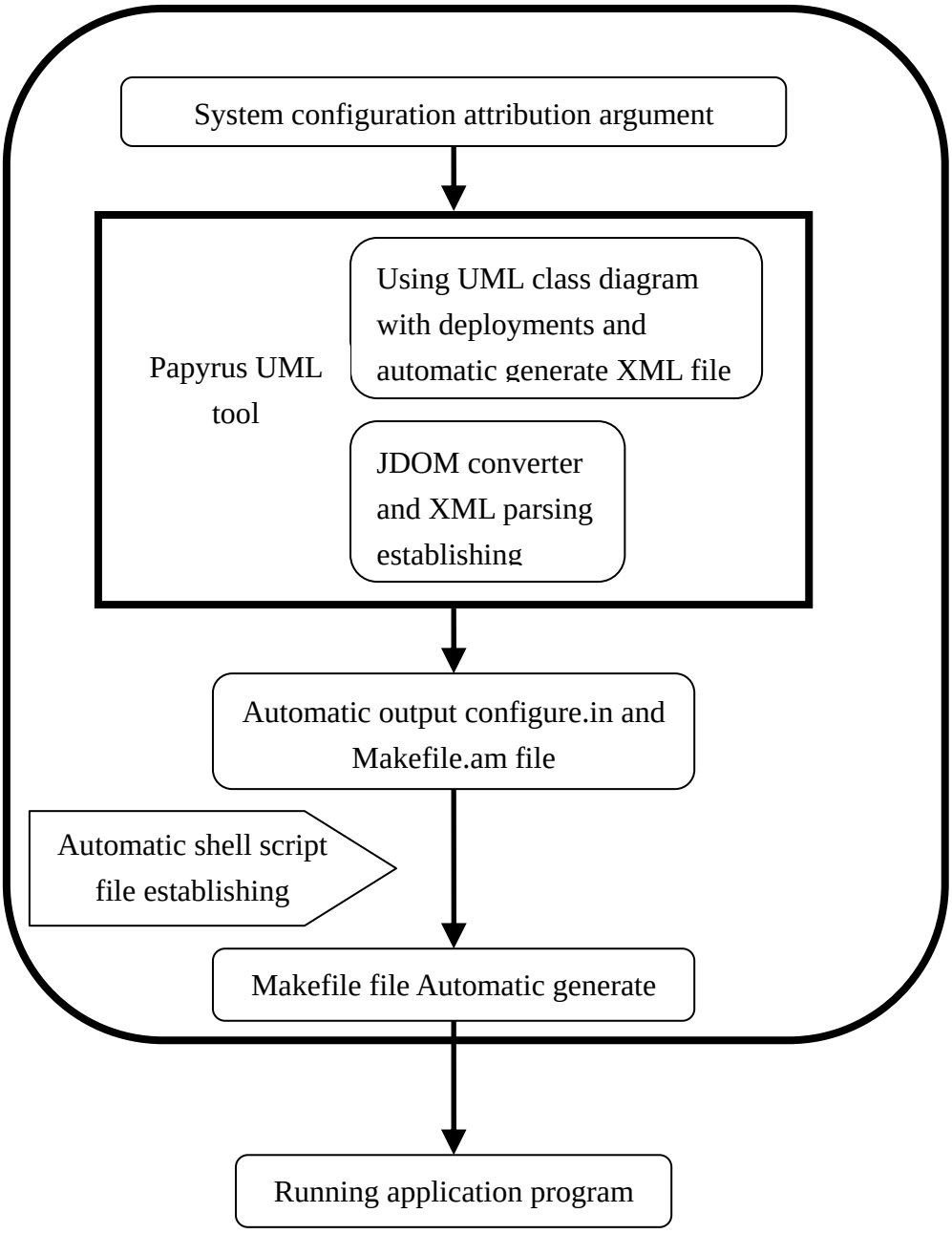


圖 2 為研究方法細部流程圖

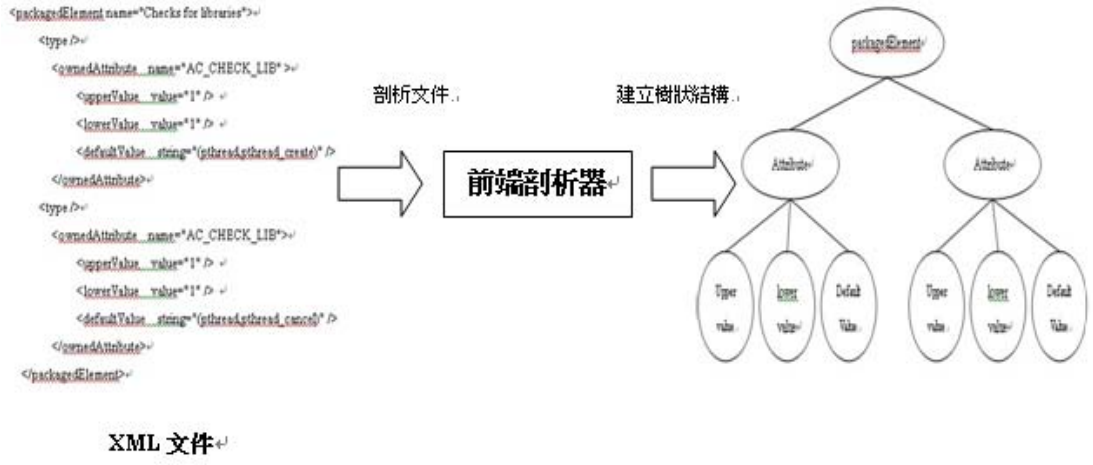
3.2 可重用元件與對應硬體及驅動之類別程式庫物件檔建立

由於嵌入式系統的應用程式可能對應許多不同的硬體，當嵌入式系統程式設計師必須針對新的平台開發應用程式時，他需要先建立相關於這個硬體平台的軟體元件。在建立這些軟體元件之前，他還要花費大量的時間熟悉軟硬體的相關資訊，這對於開發時間緊迫的嵌入式系統開發流程來說是不被允許的。如果可以將這些相關於安裝硬體平台的軟體元件蒐集起來，整理於資料庫中，就可以在需要的時候立刻拿來使用。在這邊透過 Automake 的套件將已經定義好的巨集元件做初步的建立，初步建立的巨集元件內容則根據所對應規格硬體平台及相關驅動之類別程式庫物件檔依序建立在 papyrus UML 的 class diagram with deployment 屬性欄位內容。並配合 SysML 物件導向和 class diagram 的方式開發，以增加未來的可擴充性。

3.3 前端剖析器設計

此階段是由所產生之嵌入式多核心軟體規格，是經由 thread-mapping、thread-scheduling 和正式的驗證以後，預計將根據嵌

入式多核心軟體需求規格之相關平台資訊來做為建立檔案的方針，實作自動淬取及合成核心(extraction and mapping kernel)。前端剖析器的設計是搭配 JDOM 套件來設計的，前端剖析器將複雜的 xml 文件作解析後，同時並整理出我們所要的資料，以達到解析 xml 文件的目的。而前端剖析器結合物件模型「DOM」(Document Object Model）的特性可以將一份結構化 xml 文件轉換成一棵由節點（Nodes）組成的樹狀結構，提供節點的相關屬性和方法來存取元素內容，或新增、刪除和修改節點內容。透過這樣的特性來做為初步建立語法分析(parsing)的準則，並在語法分析(parsing)後能夠產出系統設定的參數目標檔。將複雜的 xml 文件作解析後並整理出我們所要的資料。

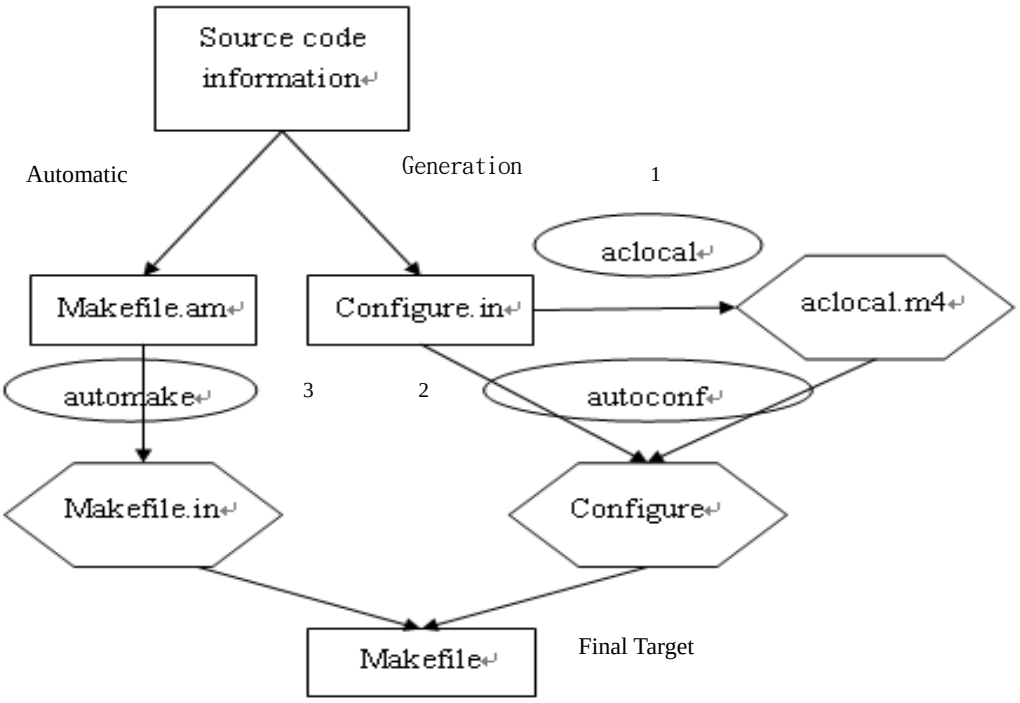


XML 文件

圖 3 為語法分析的設計方法

3.4 自動化腳本程式建立

自動化程式是透過 shell script 的方式建立，把一連串所有需要使用者輸入的指令寫成一可執行的檔案。shell script 是利用 shell 的功能所寫的一個『程式 (program)』，這個程式是使用純文字檔，將一些 shell 的語法與指令(含外部指令)寫在裡面， 搭配正規表示法、管線命令與資料流重導向等功能，以達到我們所想要的處理目的。讓使用者不用在額外手動輸入大量繁瑣的指令，只需執行自動化程式便可以完成一切動作，直到 Makefile 順利產生。



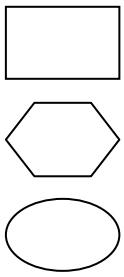


圖 4 為自動化建立程式整體流程圖

3.5 規劃系統架構對應模組自動生成系統編譯調控檔

在這之前可根據從 meta model 中獲取相依性關係，讓爾後能夠成功建構出 Makefile，如此可確保環境調控檔的正確性與 make 時的效率。上述將會分別建構出程式庫資源檔的 Configure.in File 模板及 Makefile.am File 模板。Configure.in File 模板可以設定原始程序以符合各種不同平台上Unix 系統的特性，並且根據系統參數及環境產生合適的 Makefile 目標執行檔，讓原始程式可以很方便地在不同的平台上進行編譯。而 Makefile.am File 模板將被 Automake 讀入，而後會把這一連串已經定義好的巨集元件展開並產生相對應 Makefile.am File 及 configure.in File。然後再由 configure 這個 shell script 檔案根據 Makefile.in File 產生合適的

Makefile。最後即是實作一自動合成系統核心之 Meta model。Meta model 的建立與對應的 System configuration 及最後的核心系統硬體平台的結合跟周邊I/O裝置的對應，即為 Architecture mapping information 。Architecture mapping information 為 VMC Code Generation 所需要之資訊，因此將 Architecture mapping information 回傳到 Code Generation [11][12]，以便未來在 Code Generation 過程後可與硬體平台方面順利結合起來。

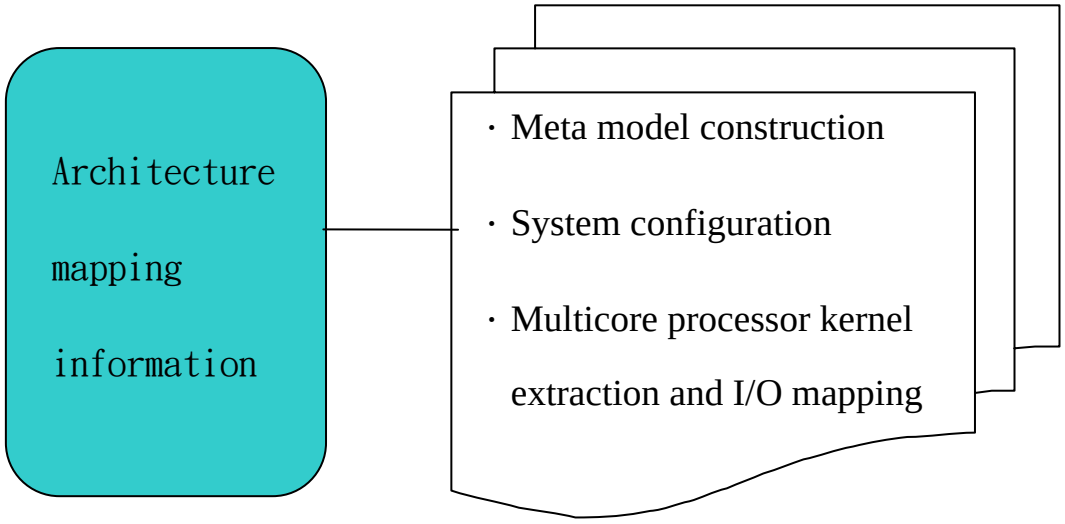


圖 5 為 Architecture mapping information

3.6 效能預測模型

方法是利用高斯機率分佈來估算數位監視系統DVR中的Parallel Video Encoder平行化影像編碼程式最大可能出現執行時間之機率分佈，故執行時間之變化將會造成整體效能之影響結果，首先假設執行時間之機率分佈呈現為高斯機率分佈，且利用高斯機率分佈來分析執行時間之機率分佈。我們實際量測，以 Parallel Video Encoder[1]平行化影像編碼每次處理一個raw data量測Parallel Video Encoder平行化影像編碼從數位攝影機接收影像後，取入100個frame後紀錄並分析，共完成100次之分析。但量測過程之中，我們也紀錄了其原始值(Normal)、利用公式(1)計算平均值(Average Mean)、中間值(Median)與利用公式(2)計算標準差(Standard Deviation)[15]以利用公式(3)將其Modeling成為高斯分佈(Gaussian Distribution)，並比較實際執行時間與預測執行時間的高斯分佈之關聯性與差異。

另外，公式(1)中其目的為求其平均值(Average Mean)m，其中X_i為處理第i個frame執行時間，而n為總接收frame之數量。

$$m = \frac{\sum_{i=1}^{n} X_i}{n} = \frac{X_1 + X_2 + \cdots + X_n}{n} \quad (1)$$

公式(2)中其目的為求其標準差(Standard Deviation) σ ，其中 X_i 為處理第 i 個 frame 執行時間，m 為其處理 i 個 frame 執行時間之平均值，而 n 為總處理 frame 之數量。

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(X_i - m)^2}{n}} \quad (2)$$

公式(3)中其目的為 Modeling 成為高斯分佈(Gaussian Distribution) $G(x)$ ，其中 m 為其平均值(Average Mean)，而 σ 為其標準差(Standard Deviation)。

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-m)^2}{2\sigma^2}} \quad (3)$$

第四章　研究過程與結果

4.1 應用實例

我們使用一個實例來實驗這個設計工具之架構支援的實作。我們採用的例子是一個Digital Video Recording System (DVR System)數位監視系統，實驗軟體設計工具系統之支援架構實作。目前這軟體實際是開發在 Linux 平台上的應用程式。 Digital Video Recording System 這是一個可以控制數位攝影機調控監視畫面即時壓縮並儲存影像資料的系統，圖六為數位監視系統概念圖。在經過自動化調配與硬體及驅動之類別資源程式庫對應後，執行於Digital Video Recording System (DVR System)之自動化程式碼合成，以展示延展到多核心平台的應用。

DVR 的系統主要可分為 Video Streaming Server (VSS)、 Remote Monitor System (RMC)、 Parallel Video Encoding (PVE)。以下圖 6 為數位監視系統的概念圖， 圖 7 為 Digital Video Recording System 整體架構圖。

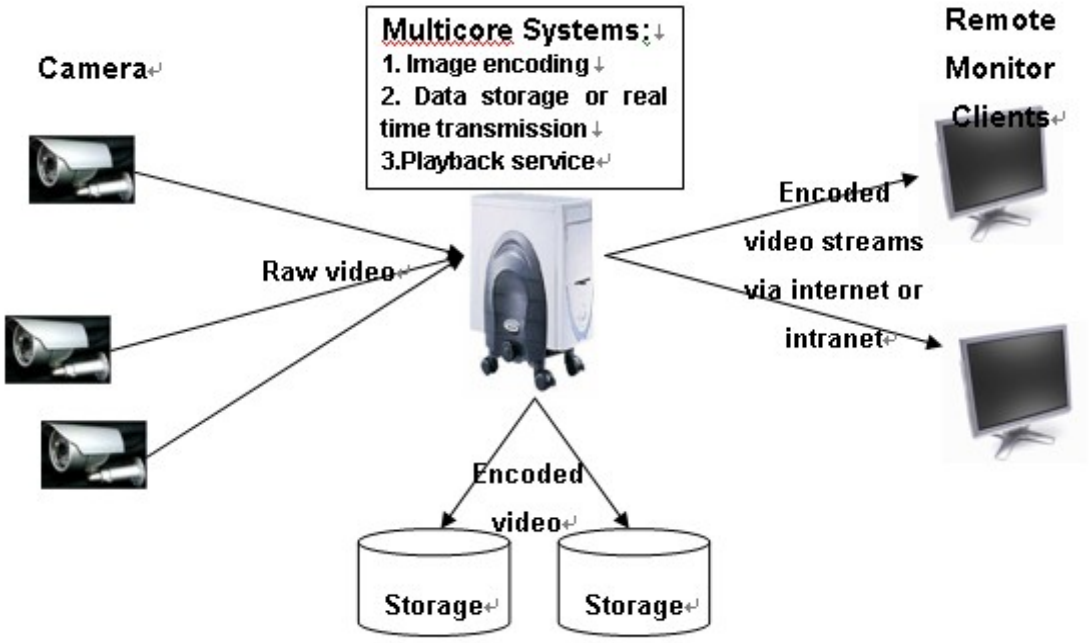


圖 6 為數位監視系統的概念圖

	Description
Remote Monitor System	對 Remote Client User 而言，即時地監控或調閱過往的紀錄是整個重要核心功能。RMC 就是根據此核心功能在用戶端開發的遠端監控與影像播放的軟體應用程式。
Video Streaming Server	這個 server 主要的功能是允許使用者透過 client server 提供的連線功能，與這個 server 建立連線之後，從 Server 取得 DC 所錄製的影像畫面以及相關資訊。
Parallel Video Encoding	這邊是將影像編碼的程式，以平行化的方式執行，以期望達到更高的效率的一種技術。

圖表 1 為數位監視系統的模組簡述

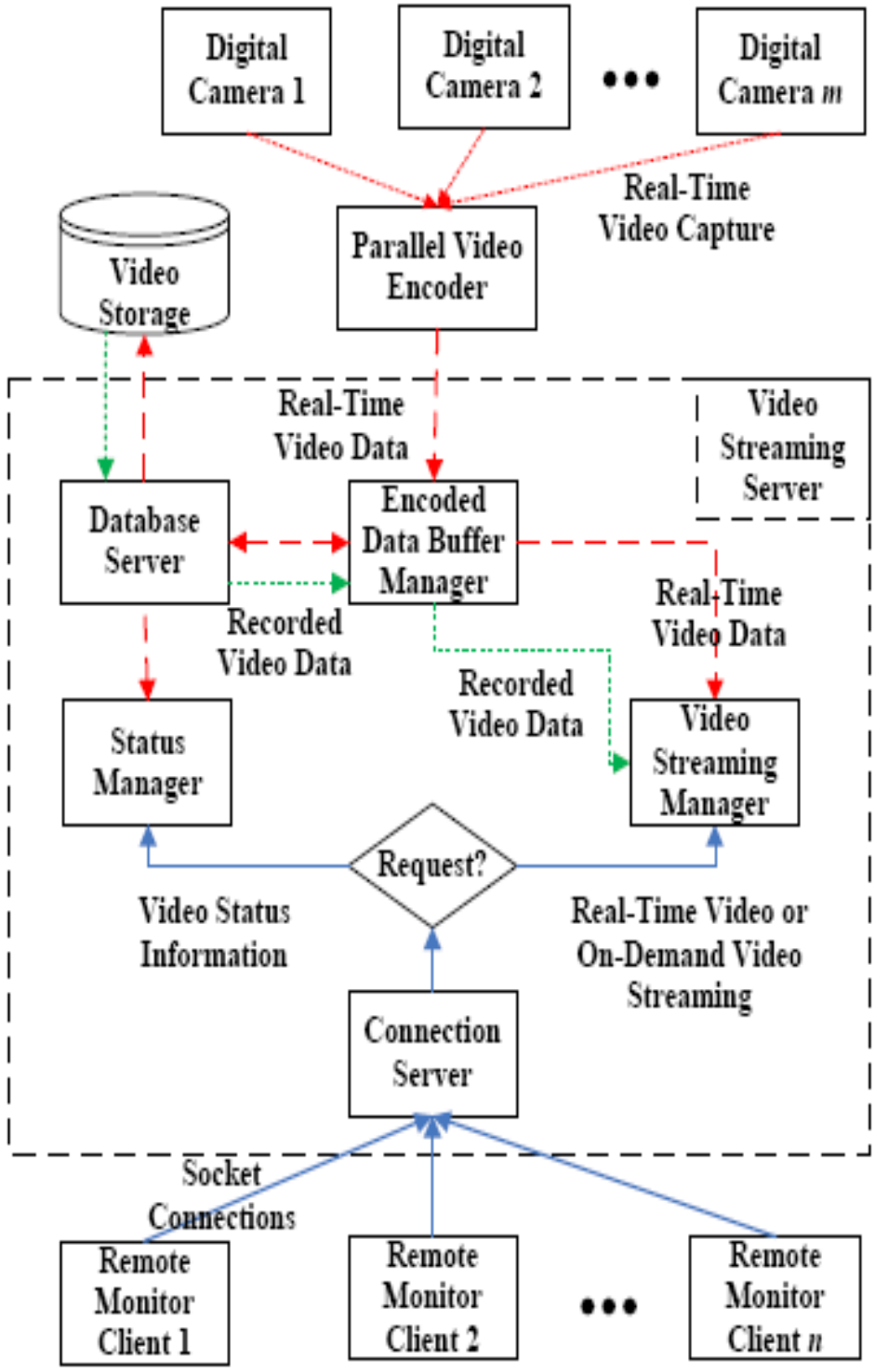


圖 7 Digital Video Recording System 主體架構圖

Parallel Video Encoder，簡稱PVE，是將影像編碼的程式，以平行化的方式執行，以期達到更高的效率的一種技術。這是我們所採用的數位攝影的例子中，主要的核心技術之一。它主要包括了四個部份：Image-Capture、DCT (Discrete cosine transform)、Quantization以及Entropy Encoding。在傳統的單核心系統中，執行一個編碼的程式，這四個主要部份是依序執行；而在多核心系統中，我們希望能夠以平行執行程式的方式，更快速能得到編碼後的結果。

我們將透過數位攝影的方式，擷取影像後，經過平行化的PVE程式，得到的編碼結果，再做進一步的影像傳輸和處理。實作將以Threading Building Blocks (TBB)的方式，做到Data parallelism以及Data flow parallelism。以下我們將仔細介紹PVE裡各主要部份的功能。

Pipeline Stages

如同硬體指令的pipeline execution的觀念，我們將軟體也做pipeline的執行。在數位攝影機擷取影像後，對於PVE的編碼處理過程區域設計成pipeline的形式。pipeline的形式是利用 Threading Building Blocks (TBB) 的方式實作，透過 TBB 平行庫套件中 parallel_pipeline 演算法對 task scheduler 中的任務調度。TBB

平行庫可以看作是一種對多線程更高層面的封裝，它不再使用 thread ，而是以 task 作為基本的抽象任務，從而能夠更好的整合計算資源並優化的調度任務。PVE的主要四個功能的部份，分成4個工作(task)設計成pipeline的形式，讓不同部份的影像能平行做編碼的處理。以下，我們以圖例的方式說明這個pipeline stage以及如何做到平行化編碼。IC為Image-Capture的縮寫，DCT為Discrete cosine transform的縮寫，Quan為Quantization的縮寫，Enc為Entropy Encoding的縮寫。每一個raw data都有四個stages要處理，分別由四個tasks來做。當第一個raw data做完IC就接著做DCT；當第一個raw data在做DCT時，第二個raw data就可以做IC了。以此類推，當第一個raw data做到Enc時，第四個raw data就開始做IC。

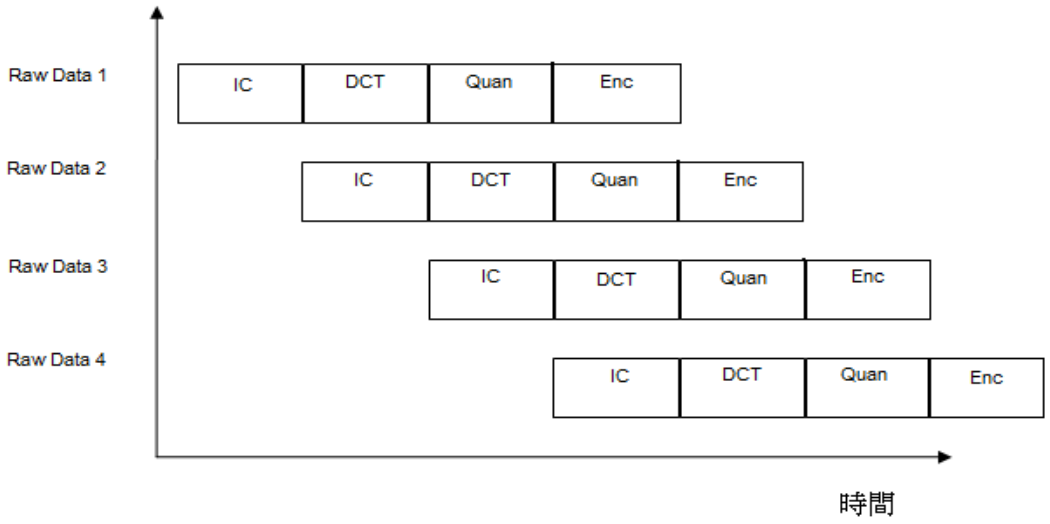


圖 8 Parallel Video Encoder 的 pipeline task stage

以下我們將分項介紹這些功能。

(1) Image-Capture：

從數位攝影機中取得 raw data，將取得的 raw data 要存在暫存的元件中。

(2) Discrete cosine transform (DCT)：

DCT 是與 Fourier Transform 相關的一種轉換，經常被使用在信號處理和圖像處理(包括靜止的圖像和動態的圖像)的使用，對於信號或圖像進行有損資料壓縮(lossy data compression)。

(3) Quantization：

量化的目的是將經 DCT 轉換後出來之區塊內各係數值變小，以利後續之編碼能取得更好的壓縮效果。

(4) Entropy Encoding：

這個 task 包含二種編碼(encode)的方式：脈差調變編碼與長度變動編碼。DC 的係數部份使用 DPCM 編碼，AC 的係數則以長度變動編碼。接著再分別以 Huffman 編碼(Huffman encoding)。

4.2 研究過程

UML 模型輸入

首先我們選用 UML 中的類別圖作為系統的輸入模型，在根據所對應規格硬體平台資訊做為開發驅動之類別程式庫物件檔依據，另外配合物件導向的規則與UML結合依序建立class diagram with deployments 的屬性。這些屬性的內容是透過 Automake的套件將已經定義好的巨集元件收集到class diagram with deployments 做為可重用元件與對應硬體及驅動之類別程式庫物件檔。圖9 為對應硬體及驅動之類別測試程序初始化物件檔。圖10 為對應硬體及驅動之類別測試程序編譯器物件檔。圖11 對應硬體及驅動之類別測試程式庫物件檔。圖12 為對應硬體及驅動之類別測試程式庫物件檔。圖13 為對應硬體及驅動之類別測試結構類型與編譯器特性物件檔。圖14 為對應硬體及驅動之類別測試函數物件檔。圖15 為對應硬體及驅動之類別測試系統調用物件檔，圖16 為對原始程式檔之類別程式庫物件檔。

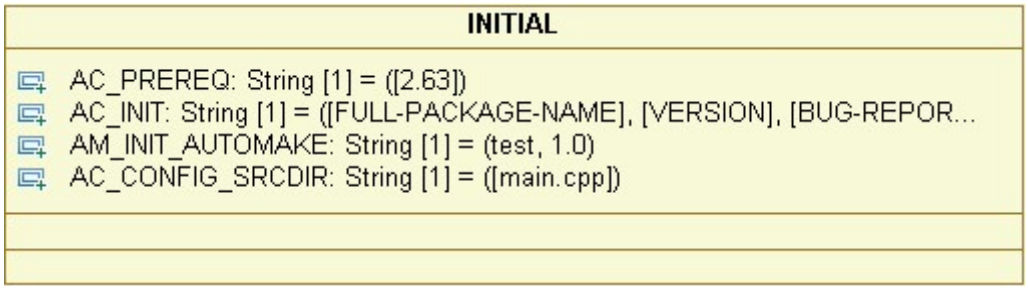


圖 9 對應硬體及驅動之類別測試程序初始化物件檔

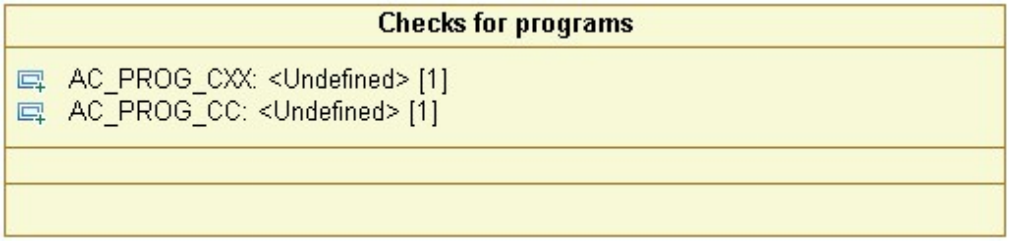


圖 10 對應硬體及驅動之類別測試程序編譯器物件檔

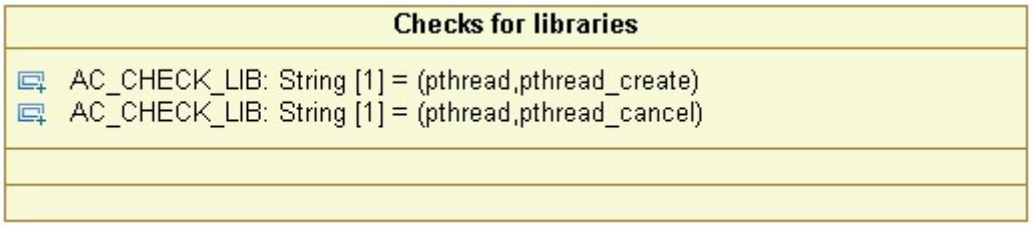


圖 11 對應硬體及驅動之類別測試程式庫物件檔

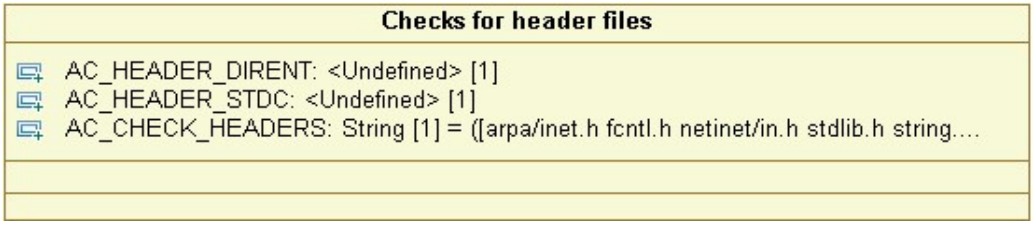


圖 12 對應硬體及驅動之類別測試程式庫標頭檔物件檔

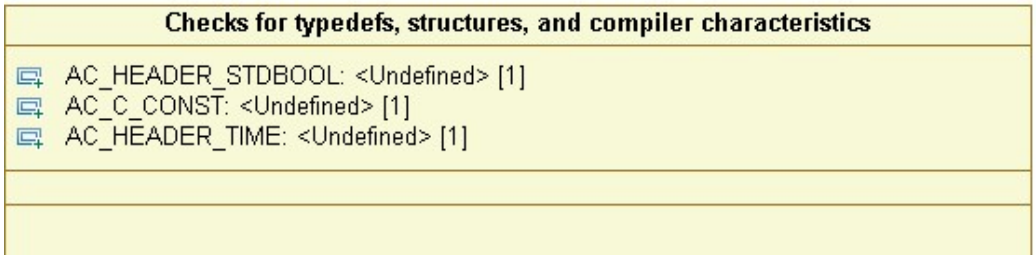


圖 13 對應硬體及驅動之類別測試結構類型與編譯器特性物件檔

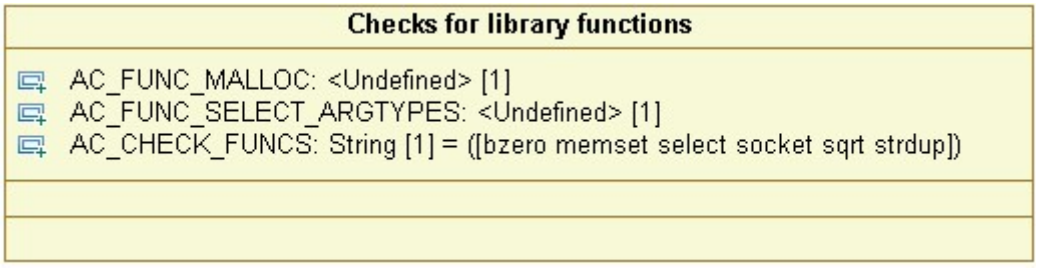


圖 14 對應硬體及驅動之類別測試函數物件檔

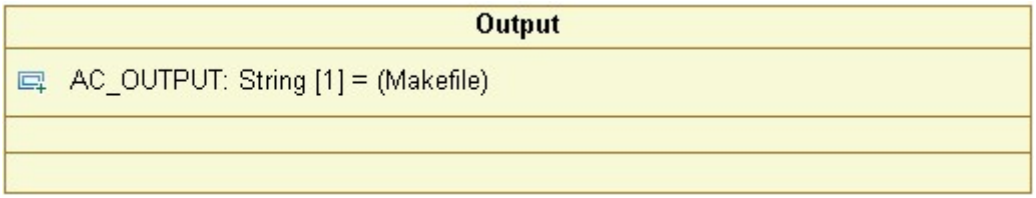


圖 15 對應硬體及驅動之類別測試系統調用物件檔

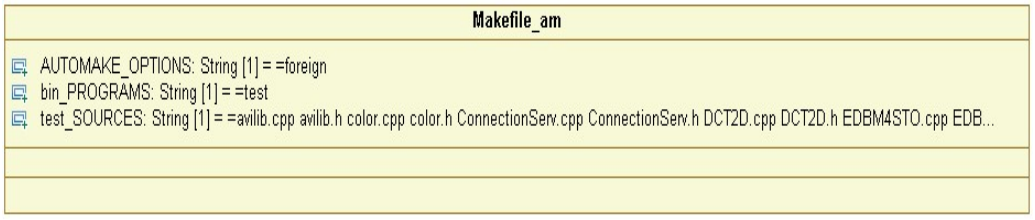


圖 16 對原始程式檔之類別程式庫物件檔

下列圖表為程式庫物件檔屬性參數說明：

AUTOMAKE parameters	parameters definition	Needed DVR input	property
AC_PREREQ	Automake version		float
AC_INIT ([FULL-PACKAGE-NAME], [VERSION], [BUG-REPORT-ADDRESS])	Package & Version information		String
AM_INIT_AUTOMAKE(package, version, [bug-report], [tarname])	Runs many macros required for proper operation of the generated Makefiles	test, 1.0	String
AC_CONFIG_SRCDIR	User input source code	main.cpp	String
AC_PROG_CC ([compiler-search-list])	Determine a C compiler to use	Used when source is C code	Undefined
AC_PROG_CXX ([compiler-search-list])	Determine a C++ compiler to use	Used when source is C++ code	Undefined
AC_CHECK_LIB (library, function, [action-if-found], [action-if-not-found], [other-libraries])	Test whether the library library is available by trying to link a test program that calls function function with the library. function should be a function provided by the library.	(pthread,pthread_create) (pthread,pthread_cancel)	String
AC_HEADER_DIRENT	Check for the following header files. For the first one that is found and defines “DIR”, define the listed C preprocessor macro		Undefined
AC_HEADER_STDC	if the system has C header files conforming. this macro checks for ‘stdlib.h’, ‘stdarg.h’, ‘string.h’, and ‘float.h’.		Undefined
AC_CHECK_HEADERS	check for the existence of the	([arpa/inet.h	String

(header-file…, [action-if-found], [action-if-not-found], [includes = ‘default-includes’])	header files specified as the first argument.	fcntl.h netinet/in.h stdlib.h string.h sys/socket.h sys/time.h unistd.h])	
AC_HEADER_STDBOOL	determine whether the system has conforming header files (and probably C library functions).		Undefined
AC_C_CONST	If the C compiler does not fully support the const keyword, define const to be empty.		Undefined
AC_C_INLINE	It will then be defined in the compilation flags or by including the file config.h before any library headers.		Undefined
AC_TYPE_OFF_T	Define off_t to a suitable type, if standard headers do not define it. This macro caches its result in the ac_cv_type_off_t variable.		Undefined
AC_TYPE_SIZE_T	Define size_t to a suitable type, if standard headers do not define it. This macro caches its result in the ac_cv_type_size_t variable.		Undefined
AC_TYPE_SSIZE_T	Define ssize_t to a suitable type, if standard headers do not define it. This macro caches its result in the ac_cv_type_ssize_t variable.		Undefined
AC_TYPE_UINT16_T	If stdint.h or inttypes.h does not define the type int8_t, define int8_t to a signed integer type that is exactly 8 bits wide and that uses two's complement		Undefined

	representation, if such a type exists.		
AC_TYPE_UINT32_T	If stdint.h or inttypes.h does not define the type int8_t, define int8_t to a signed integer type that is exactly 8 bits wide and that uses two's complement representation, if such a type exists.		Undefined
AC_TYPE_UINT64_T	If stdint.h or inttypes.h does not define the type int8_t, define int8_t to a signed integer type that is exactly 8 bits wide and that uses two's complement representation, if such a type exists.		Undefined
AC_HEADER_TIME	If a program may include both ‘time.h’ and ‘sys/time.h’		Undefined
AC_FUNC_MALLOC	If the malloc function is compatible with the GNU C library malloc (i.e., ‘malloc (0)’ returns a valid pointer), define HAVE_MALLOC to 1. Otherwise define HAVE_MALLOC to 0,		Undefined
AC_FUNC_SELECT_ARGTYPES	Determines the correct type to be passed for each of the select function's arguments, and defines those types in SELECT_TYPE_ARG1, SELECT_TYPE_ARG234, and SELECT_TYPE_ARG5 respectively. SELECT_TYPE_ARG1 defaults to ‘int’, SELECT_TYPE_ARG234 defaults to ‘int *’, and		Undefined

	SELECT_TYPE_ARG5 defaults to ‘struct timeval*’		
AC_FUNC_CHECK	CHECK functions	([bzero ftruncate memset select socket sqrt strdup strerror strncasecmp])	String
AC_FUNC_MMAP			Undefined
AC_FUNC_REALLOC			Undefined
AC_OUTPUT()	OUTPUT makefile	Makefile	String

圖表 2 為程式庫物件檔屬性參數說明

前端剖析器設計

前端剖析器的設計目的，是為了能夠擷取出class diagram with deployments 屬性資訊。但透過 UML 針對class diagram with deployments屬性資訊所產出來的結構化 xml 檔案資訊，所以必須對這複雜的xml文件進一步做擷取解析的動作。透過利用JDOM套件這樣的特性來做為初步建立語法分析(parsing)的準則，並在語法分析(parsing)後能夠呈現出類別程式庫物件設定的參數目標。將複雜的xml文件作解析後並整理出我們所要的資料。圖16 為語法分析的功能說明與輸出目標。

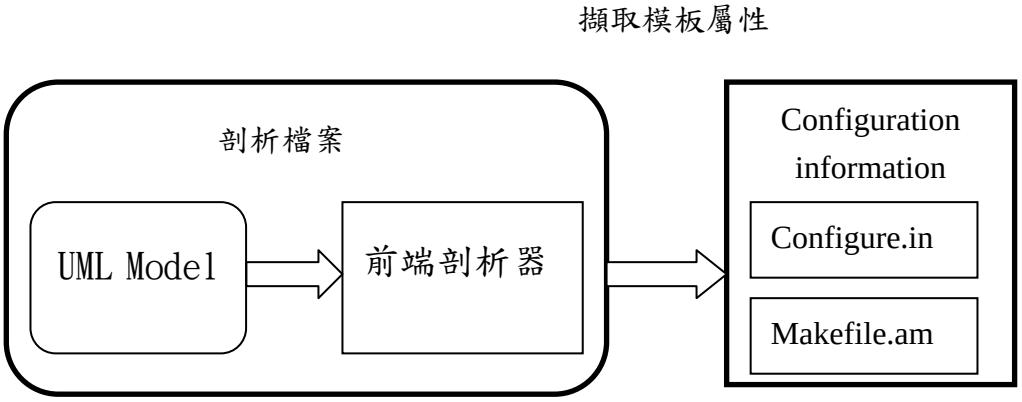


圖 17 語法分析的功能說明與輸出目標

自動化建立 Configure.in File 模板及 Makefile.am File 模板

前端剖析器設計完成後，並將前端剖析器設計的原始程式碼包裝成 HDM.jar 執行檔案。同時將前端剖析器 HDM.jar 執行檔案 plug in 到 UML 軟體工具下，配合已蒐集既定的可重用元件與對應硬體及驅動之類別程式庫物件檔，建立 Configure.in File 模板及 Makefile.am File 模板。

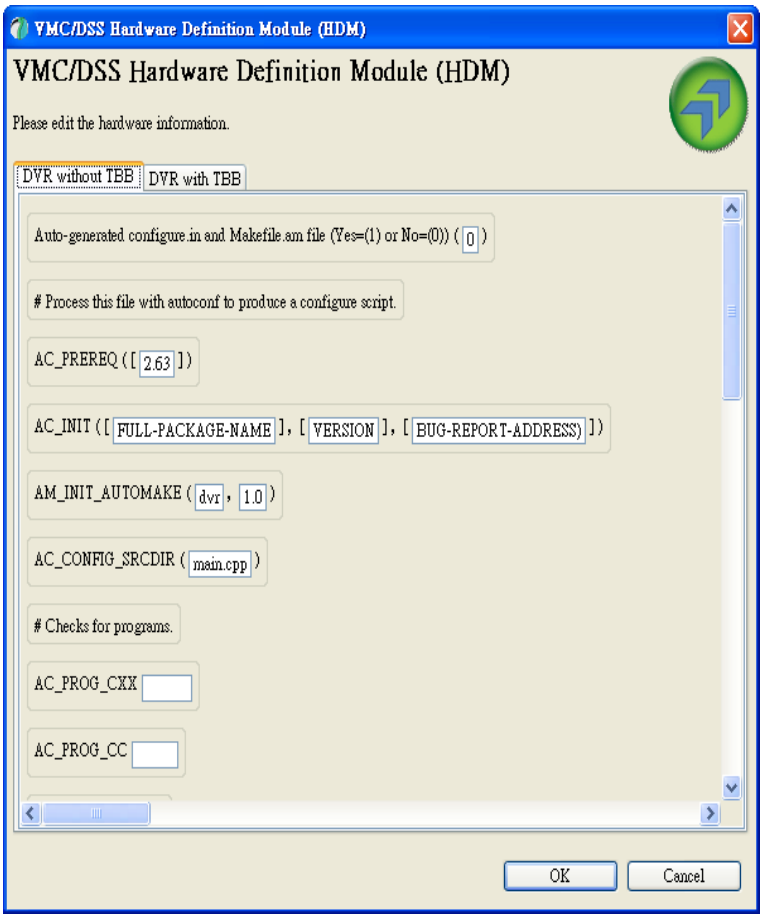


圖 18 前端剖析器設計圖形化介面

圖19 顯示自動化產生屬性資訊過程，圖20 為自動化建立之 Configure.in File 模板資訊。圖21 為自動化建立之 Makefile.am File 模板資訊。

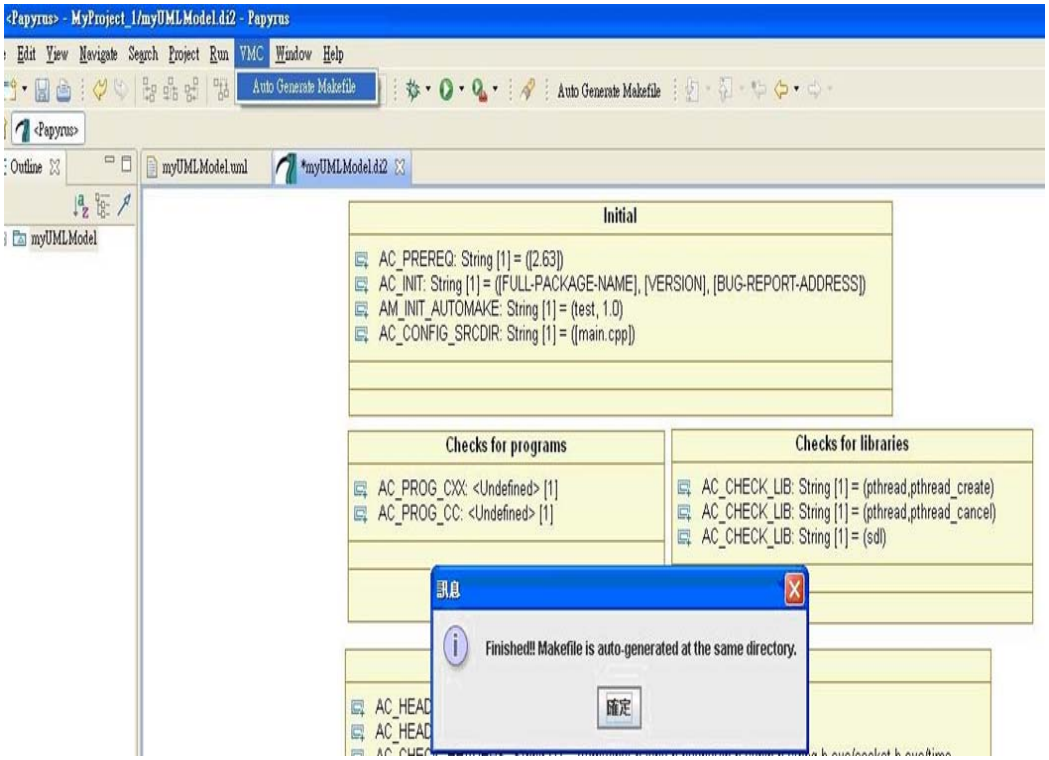


圖 19 自動化產生屬性資訊過程

```
# INITIAL
AC_PREREQ([2.63])
AC_INIT([FULL-PACKAGE-NAME], [VERSION], [BUG-REPORT-ADDRESS])
AM_INIT_AUTOMAKE(test, 1.0)
AC_CONFIG_SRCDIR([main.cpp])
#AC_CONFIG_HEADER([config.h])

# Checks for programs.
AC_PROG_CXX
AC_PROG_CC

# Checks for libraries.
AC_CHECK_LIB(pthread,pthread_create)
AC_CHECK_LIB(pthread,pthread_cancel)
# Checks for header files.
AC_HEADER_DIRENT
AC_HEADER_STDC
AC_CHECK_HEADERS([arpa/inet.h fcntl.h netinet/in.h stdlib.h string.h sys/socket.h sys/time.h unistd.h])

# Checks for typedefs, structures, and compiler characteristics.
AC_HEADER_STDBOOL
AC_C_CONST
AC_HEADER_TIME

# Checks for library functions.
AC_FUNC_MALLOC
AC_FUNC_SELECT_ARGTYPES
AC_CHECK_FUNCS([bzero memset select socket sqrt strdup])

AC_OUTPUT(Makefile)
```

圖 20 自動化建立之 Configure.in File 模板資訊

```
#Makefile_am
AUTOMAKE_OPTIONS=foreign
bin_PROGRAMS=test
test_SOURCES=avilib.cpp avilib.h color.cpp color.h ConnectionServ.cpp ConnectionServ.h DCT2D.cpp
DCT2D.h EDBM4STO.cpp EDBM4STO.h FileManager.cpp FileManager.h global.h huffman_.h Huffman.cpp Huffman.h
Information.h linux.cpp main.cpp Monitor.cpp Monitor.h port.h PVECapture.cpp PVECapture.h PVEEncoding.cpp
PVEEncoding.h qactive.cpp qactive.h qassert.h qepool.cpp qepool.h qequeue.cpp qequeue.h qevent.h qf.cpp
qf.h qf_linux.h qfpkg.h qfsm.cpp qfsm.h qhsm.cpp qhsm.h qhsmTran.cpp qtimer.cpp qtimer.h Quan.cpp Quan.h
RMC.cpp RMC.h SigEvt.h StreamingServ.cpp StreamingServ.h Utility.cpp Utility.h utils.cpp utils.h
uvc_compat.h uvcvideo.h v4l2uvc.cpp v4l2uvc.h
```

圖 21 自動化建立之 Makefile.am File 模板資訊

自動化程式建立

將自動化產生的腳本程式與已建立的Configure.in File模板及Makefile.am File模板結合，達到完整自動化過程，加快整體運行的效率。最終目標將得到 Makefile 。此 Makefile 即是合成硬體平台核心與原始程式碼及所需應用到的I/O對應後的目標執行檔。圖22 為自動化腳本程式產生的程式資訊。

```
#!/bin/sh
PATH=/bin:/sbin:/usr/bin:usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
echo "aclocal"
aclocal
echo "autoconf"
autoconf
echo "automake --add-missing"
automake --add-missing
echo "./configure"
./configure
exit 0
```

圖 22 自動化腳本程式產生的程式資訊

4.3 實驗結果

將 Digital Video Recording System 數位監視系統的原始程式檔案與已建立的 Configure.in File 模板及 Makefile.am File 模板加上自動化程式結合，最終目標檔是得到完整的 Makefile。透過 meta model 完整的建立與驗證，實驗結果順利產出我們所預期的 Makefile ，同時並成功編譯一 Digital Video Recording System 多核心 client-server 之程式碼。

Configure.in及 Makefile.am File模板放在相同目錄。自動化產生程式將會與已建立的 Configure.in File 模板及 Makefile.am File模板做結合，只需要一個指令的動作，就可以產生 Makefile 。如圖23、24所示。

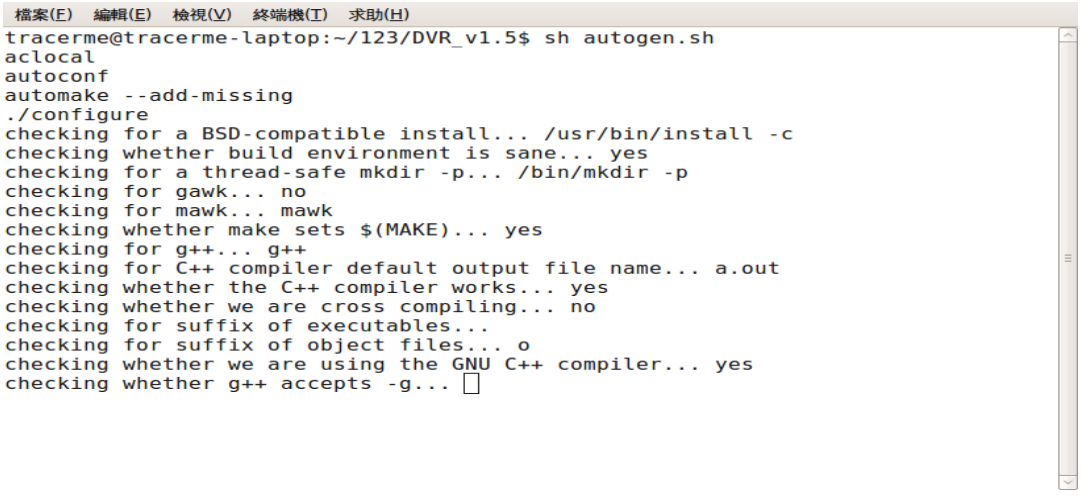


圖 23 自動化執行程式過程

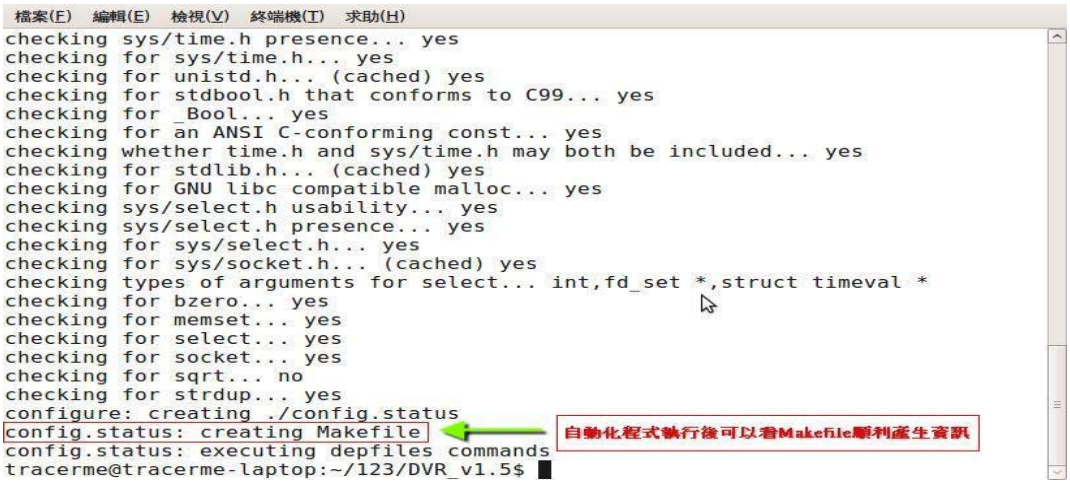


圖 24 自動化執行過程完成後產出 Makefile 資訊

Make最主要功能就是通過Makefile目標執行檔來描述源程序之間的相互關係並自動維護編譯工作。Make 對 Makefile 目標執行檔中需要編譯各個原程式檔案與資源檔並連接產生可執行程式，並要求定義原程式檔案的依賴關係。

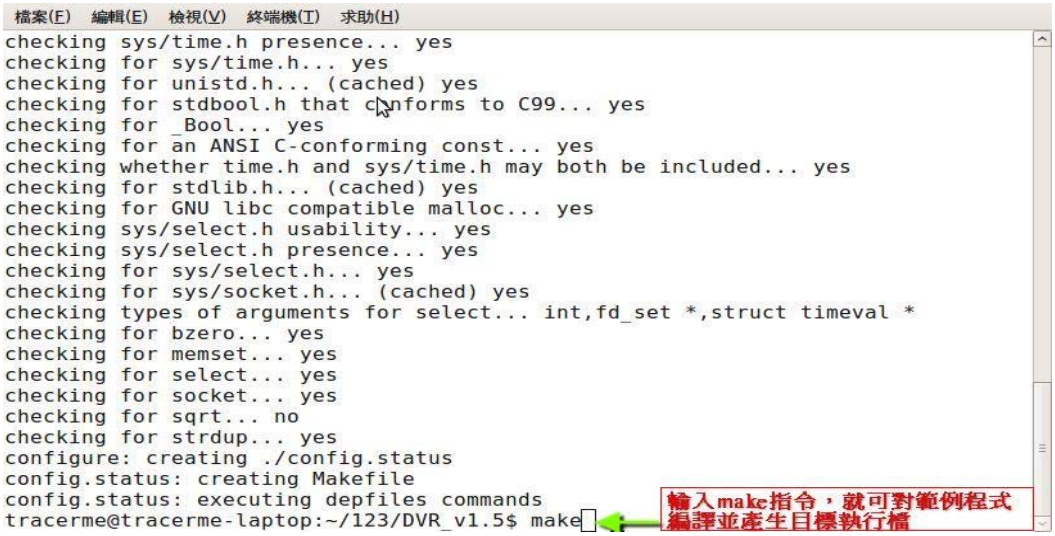


圖 25 為 make 對原始程式檔自動編譯與安裝指令

完成 Make 對各個原程式檔案編譯連結後，即可在 DVR 範例目錄下找到 DVR 範例執行檔。同時對 DVR 範例執行檔執行啟動。左邊的畫面是 real-time 數位攝影機的畫面呈現。右邊的畫面則是 DVR 範例程式的 client 端控制畫面，可以調整數位攝影機及 DVR 監視系統的相關資訊。

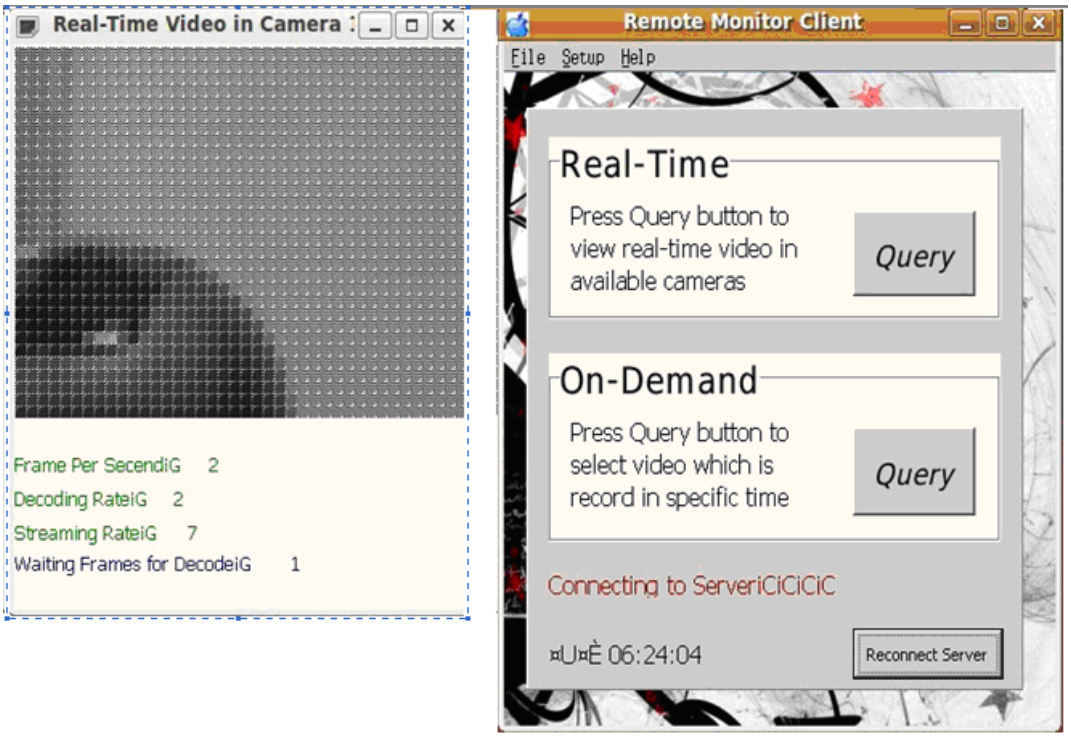


圖 26 全自動化流程編譯後執行數位監視系統成果

在DVR監視系統執行的同時會有DVR監視系統對於硬體平台使用資訊與影像處理的數據產生，然而會把這些數據的值記錄成記錄檔。

這些數據的呈現，是根據DVR監視系統軟體在對應的硬體平台架構上執行時整體效能資訊。裡面的數據記錄著 DVR 監視系統的軟體

平行程序與管線化的影像壓縮處理對於硬體平台架構的核心使用率與周邊裝置數位攝影機的不同所顯示的數據資訊。圖27 為多核心效能結果的部分數據記錄。

	A	B	C	D	E	F	G
1	Record Time (ticks)	CPU Utilization(%)	CPU_1 (%)	CPU_2 (%)	Capture Rate (PVEC1)	Captured Frames	Dropped Frames
2	90	5	0	11	0	0	0
3	120	66	60	69	15	4	0
4	150	83	87	80	12	7	0
5	180	84	92	79	12	10	0
6	210	92	92	92	16	14	0
7	240	92	96	88	16	18	0
8	270	85	88	84	12	21	0
9	300	89	84	92	16	25	0
10	330	93	100	91	16	29	0
11	360	97	95	96	16	33	0
12	390	100	100	100	12	36	0
13	420	94	95	95	16	40	0
14	450	95	96	95	12	43	0
15	480	86	84	88	15	47	0
16	510	88	80	92	16	51	0
17	540	90	91	87	16	55	0
18	570	88	85	92	15	59	0
19	600	82	83	84	16	63	0
20	630	82	88	75	15	67	0
21	660	95	100	92	12	70	0
22	690	90	92	88	15	74	0
23	720	95	96	95	12	77	0

圖 27 數位監視系統的整體綜合數據記錄

以下為DVR 監視系統對於硬體平台架構核心數目增加與軟體影像壓縮處理平行度的差異所量測的執行時間的整體數據資訊。

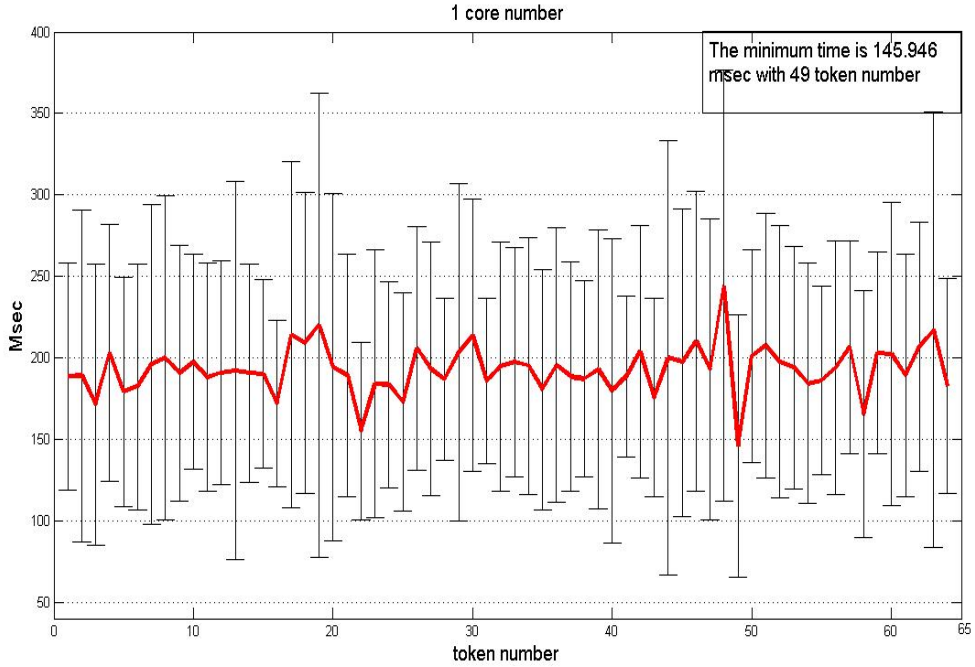


圖 28 量測單核心與 1-64 token number 整體數據變化

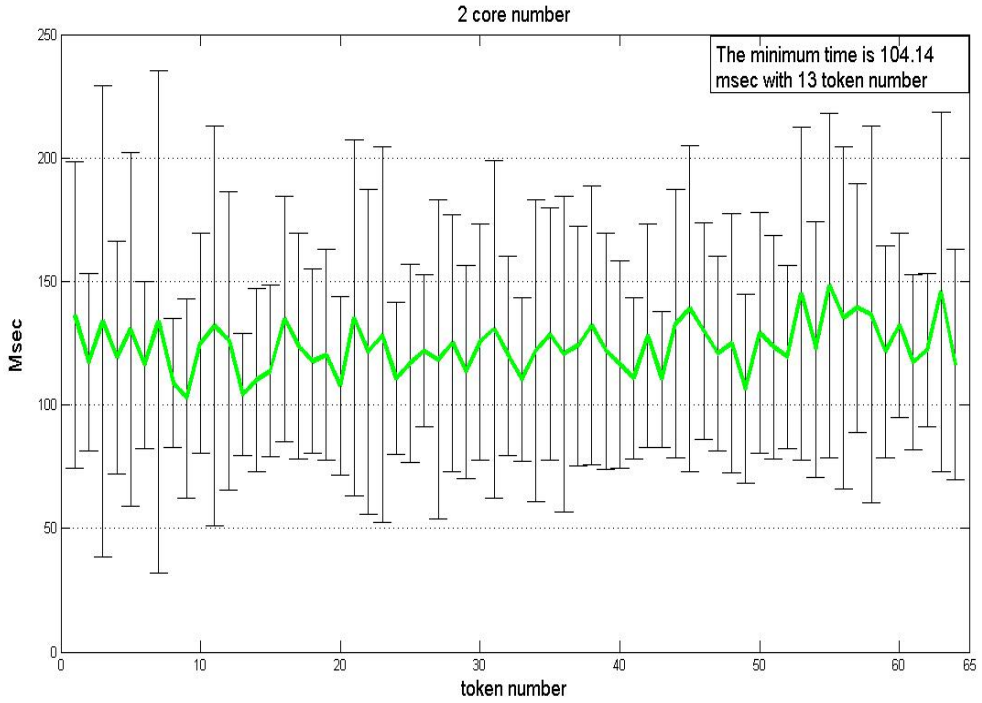


圖 29 量測雙核心與 1-64 token number 整體數據變化

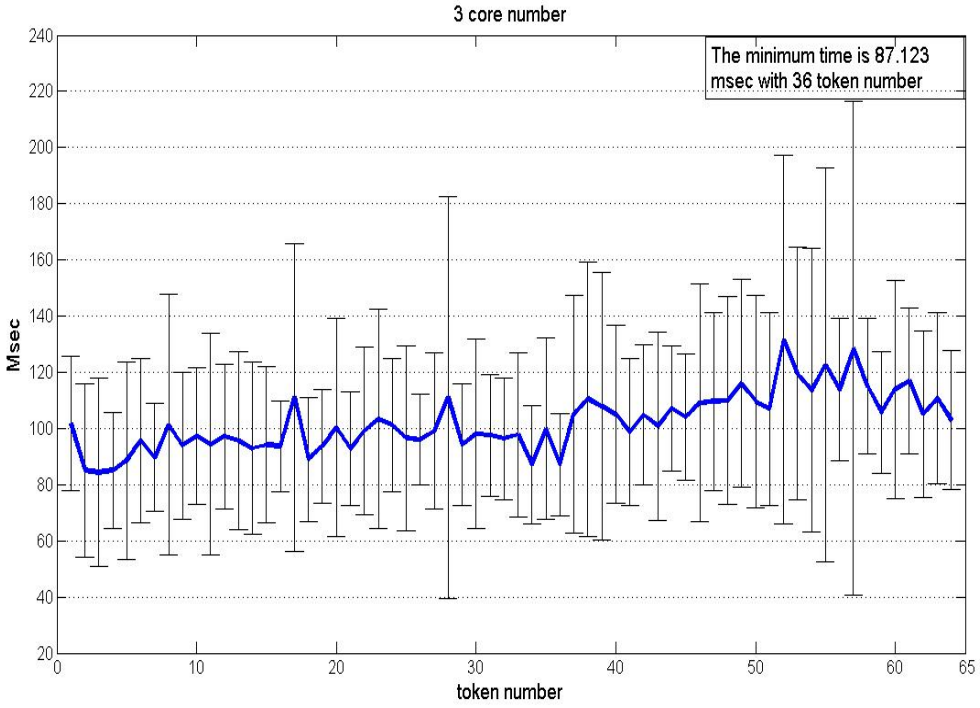


圖 30 量測三核心與 1-64 token number 整體數據變化

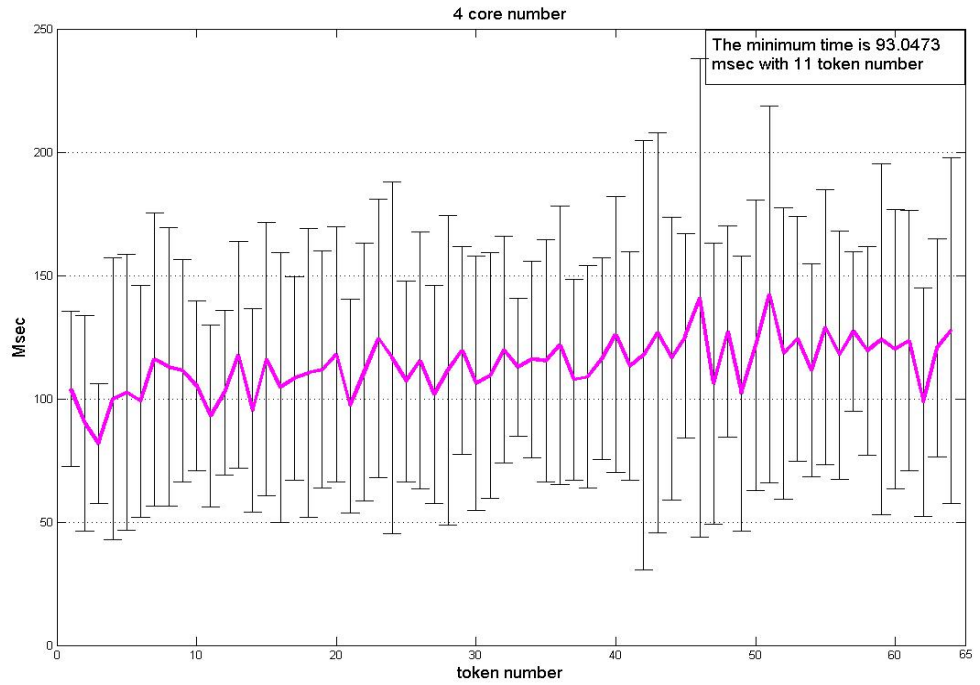


圖 31 量測四核心與 1-64 token number 整體數據變化

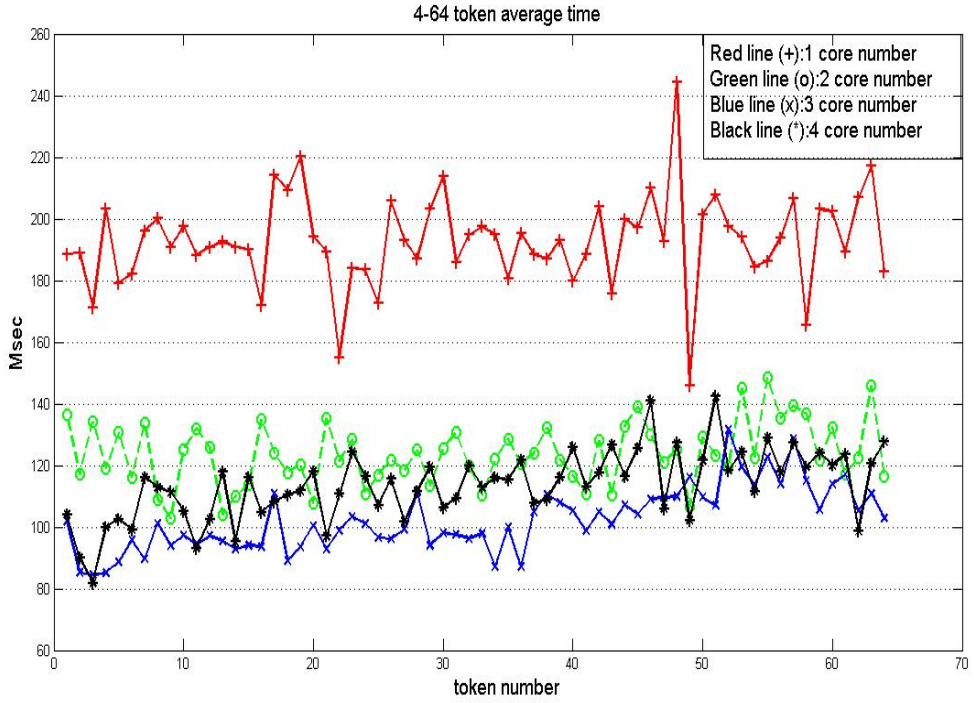


圖 32 一至四核心整體執行時間比較

最佳平均執行時間				
	單核心	雙核心	三核心	四核心
Token number	49	13	36	11
時間單位(毫秒)	145.946	104.14	87.122	93.047

圖表 3 為不同核心與 Token number 的最佳平均執行時間

由上面的表格資訊，我們知道在各個不同核心數在特定的token number上會有最佳的執行時間。而我們在進一步對於這些特定的token number搭配不同核心做分析。

在單核心上搭配49 token number ，可以知道最佳平均執行時間為145.94毫秒，但同樣的token number在雙核、三核、四核心的表現上卻有更好的執行時間。

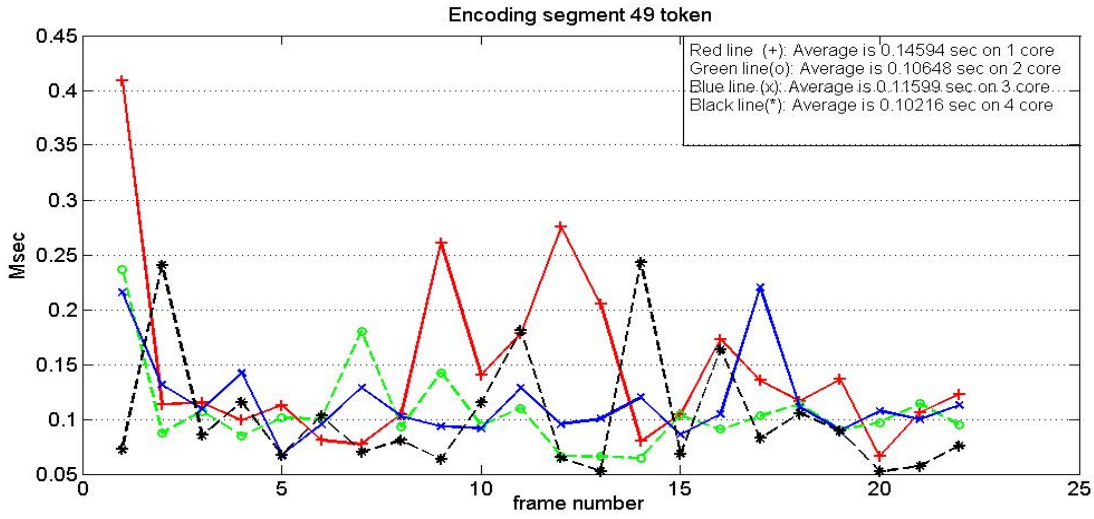


圖 33 一至四核心搭配 49 token number 執行時間比較

在雙核心上搭配13 token number ，可以知道最佳平均執行時間為104.14毫秒，但同樣的token number在三核心的表現上有更好的結果。然而在單核與四核的表現上卻不如雙核心來的理想。

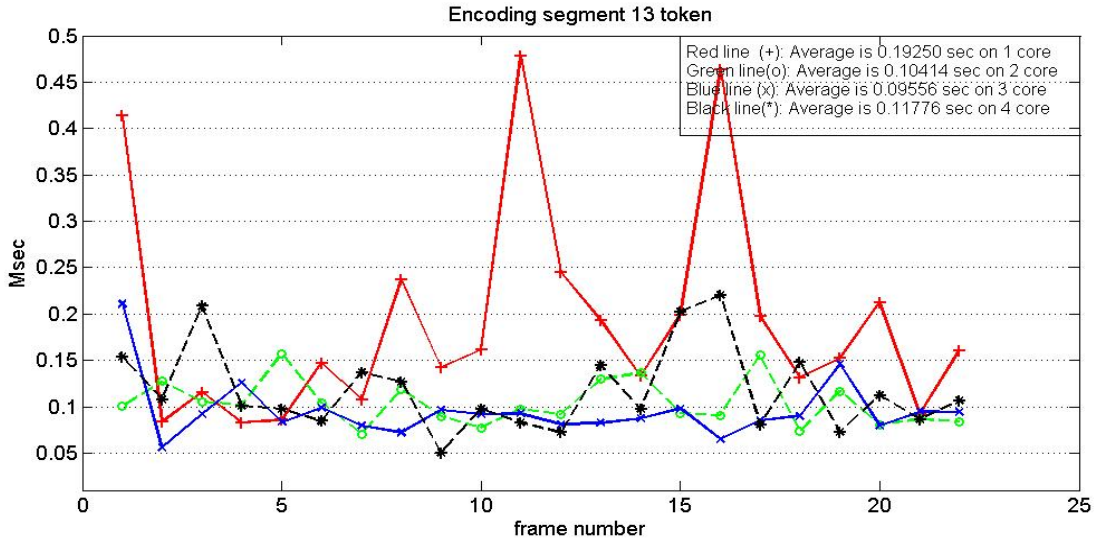


圖 34 一至四核心搭配 13 token number 執行時間比較

在三核心上搭配36 token number ，可以知道最佳平均執行時間為87.12毫秒，這樣的搭配是目前的最佳的結果。

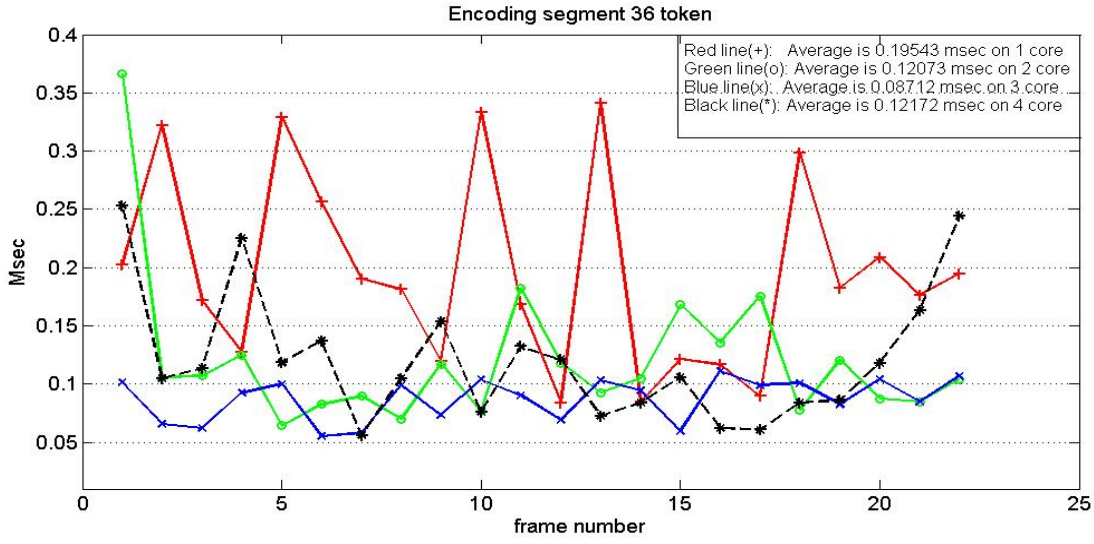


圖 35 一至四核心搭配 36 token number 執行時間比較

在四核心上搭配11 token number ，可以知道最佳平均執行時間為93.04毫秒，雖然四核心的表現優於單核、雙核、三核心，但相對於三核心搭配36 token number的平均執行時間還是略遜一些。

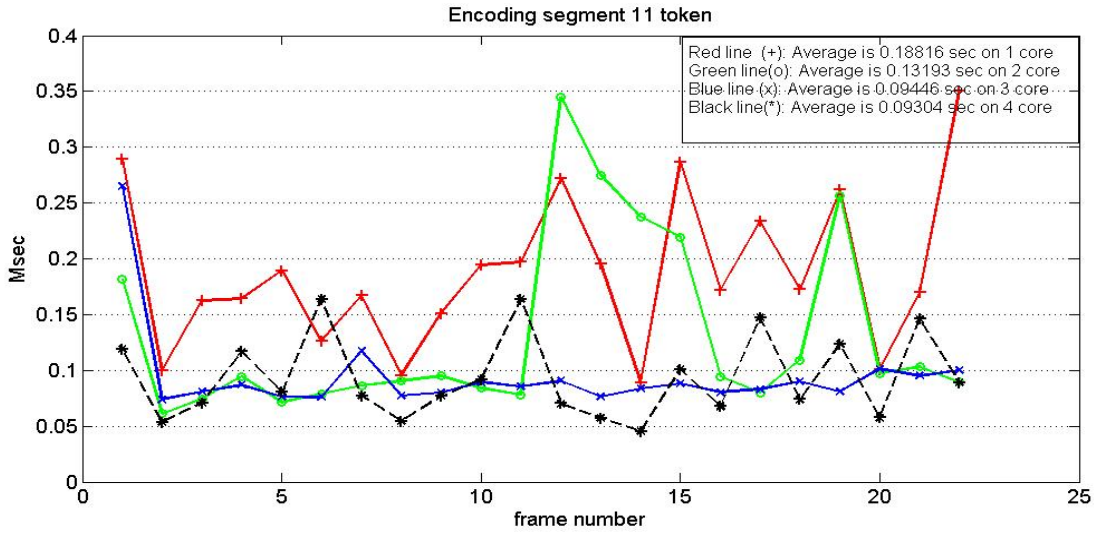


圖 36 一至四核心搭配 11 token number 執行時間比較

這些數據資訊顯示軟體的執行在硬體平台與 system configuration 有著密不可分的關連性，然而 system configuration 及 Makefile 與硬體平台結合必是一門重要且複雜的研究。這些都是 Architecture mapping 所要調查研究之項目，如：thread 數目、核心數目、I/O 周邊裝置…等。這些調查項目將待有明確的結果後，會進一步有系統的呈現。

第五章　結論與建議

嵌入式系統裝置不同於一般桌上型電腦作業系統(windows/linux)相比較之下，嵌入式作業系統並未要求全方位功能的發展，而是必須能夠依據系統設計的規格，有效率的發揮出硬體的運算能力，使得嵌入式產品達到效率與價格比的最佳化，在大多數的系統上會要求全自動完成所需要設定的工作。此研究自動化過程是先將已經蒐集好的軟體元件做初步的建立在 UML 上，同時對於所要結合之硬體平台規格資訊做對應，並搭配 meta model 自動化的流程，最終產生 Makefile 目標執行檔。這樣的自動化流程將與 VERTAF 中的多核心嵌入式軟體之合成器在程式碼生成時做重要的結合並與系統資源調配時使用。達到整體自動化更完整的建立，實作出自動萃取及合成核心(extraction and mapping kernel)。

本論文使用 Digital Video Recording system (DVR system)說明了如何運用相關的架構對應及整合後工具組之實際應用的技術，順利完成預定目標。透過以架構對應及整合後工具組之實際應用並利用轉換模組的方式，將 code gen.時所需之硬體系統資源相關訊息建入一嵌入式多核心軟體規格，系統之需求規格，如板子的核心數(core number)、事件反應速度、OS 種類、硬體成本等。另外再加入 code gen.時所需之多核心

運算程式庫支援，包括程式庫安裝資訊、及程式庫種類等。以 class diagram with deployments 方式呈現。本研究接著將此一 class diagram with deployments diagram 產出 xml 檔，以便在後續階段轉換為 System configuration 所需之模板，自動化產生 Makefile 目標執行檔，同時成功編譯並執行 Digital Video Recording system (DVR System) 多核心程式碼。

我們希望藉由完成這個研究，整合開發架構及開發工具組，在未來可以在軟體工程結合嵌入式系統上能夠進一步幫助節省程式設計者在處理多種系統平台及各種不同硬體裝置相關問題所花費的時間、精力，同時透過自動萃取並產生核心並對應軟硬體驅動程式類別程式庫物件檔，節省程式設計者在開發程式的動作，並且使開發出來的程式可以用最快速、經濟的方式，部署在各種不同的嵌入式平台上。這些研究成果可以直接減少人力參與繁複多核心程式設計的程度，對加速嵌入式資訊家電計算速度的程度有直接的衝擊，不只具有提昇數位產業產值之潛力，有市場之價值，同時在學理上亦可驗證設計樣式及軟工理論對多核心應用之影響。

References

[1] Chao-Sheng Lin, Pao-Ann Hsiung*, Shang-Wei Lin, Yean-Ru Chen,Chun-Hsien Lu,Sheng-Ya Tong, Wan-Ting Su, Chihhsiong Shih, Nien-Lin Hsueh, “VERTAF/MULTI-CORE: A SYSML-BASED APPLICATION FRAMEWORK FOR MULTI-CORE EMBEDDED SOFTWARE DEVELOPMENT” ,Journal of the Chinese Institute of Engineers, Vol. 32, No. 7, pp. 985-991 (2009)

[2] Pao-Ann Hsiung, Shang-Wei Lin, “Automatic synthesis and verification of real-time embedded software for mobile and ubiquitous systems Computer Languages”, Systems & Structures, Volume 34, Issue 4, December 2008, Pages 153-169 (2008)

[3] Pao-Ann Hsiung, “Embedded software verification in hardware–software codesign” Journal of Systems Architecture, Volume 46, Issue 15, 31 December 2000, Pages 1435-1450 (2000)

[4] D. de Niz and R. Rajkumar, “Time Weaver: A Software-Through-Models Framework for Embedded Real-Time Systems”, Proc. Int’l Workshop Languages, Compilers, and Tools for Embedded Systems, pp. 133-143, June 2003.

[5] J. Rumbaugh, G. Booch, and I. Jacobson, “The UML Reference Guide”, Addison Wesley Longman, 1999.

[6] S. Konrad, B.H.C Cheng, and L.A Campbell, “Object analysis patterns for embedded systems,” Software Engineering, IEEE Transactions on Volume 30, Issue 12, Dec. 2004 Page(s):970 – 992.home page

[7] GNU Automake, URL: http://www.gnu.org/software/automake/

[8] M.Samek, “Practical Statecharts in C/C++ Quantum Programming for Embedded Systems.” CMP Books, 2002.

[9] QUANTUM-FRAMWORK (QF), URL: http://www.quantum-leaps.com/

[10] JDOM, URL: http://www.jdom.org/

[11] P.-A. Hsiung and C.-Y. Lin, “Synthesis of Real-Time Embedded Software with Local and Global Deadlines”, Proc. First ACM/IEEE/IFIP Int’l Conf. Hardware-Software Codesign and System Synthesis (CODES+ISSS ’03), pp. 114-119, Oct. 2003.

[12] P.-A. Hsiung, “Embedded Software Verification in Hardware-Software Codesign”, J. Systems Architecture-the Euromicro J., vol. 46, no. 15, pp. 1435-1450, Nov. 2000.

[13] JAXP: JAVA API for XML Processing, Sum Mircosystems,

URL: http://java.sun.com/xml/jaxp/index.jsp.

[14] Clark, J., Derose, S., XML Path Language (XPath) Version 1.0, 1999,

URL:http://www.w3.org/TR/xpath, 1999.

[15] Document Object Model, W3C DOM Working Group

URL: http://www.w3.org/DOM

[16] S. Wang, S. Kodase and K. G. Shin, “Automating embedded software construction and analysis with design models”, in Proc. of International Conference of Euro-uRapid, Frankfurt, Germany, December 2002.

[17] GNU make 中文手冊

URL:http://www.linuxsir.org/main/doc/gnumake/GNUmake_v3.80-zh_CN_html/index.html

[18] P.-A. Hsiung, S.-W. Lin, C.-H. Tseng, T.-Y. Lee, J.-M. Fu, and W.-B. See, J. Reinders, “VERTAF: An Application Framework for the Design and Verification of Embedded Intel Threading Building Blocks”, O’Reilly, 2007.

[19] S. Lange, and U. Kebschull, “Virtual hardware byte code as a design platform for reconfigurable embedded systems Design”, Automation and Test in Europe Conference and Exhibition, 2003 Page(s):302 – 307.

[20] J. Reinders, *Intel Threading Building Blocks*, O’Reilly, 2007. [1] P.-A. Hsiung, S.-W. Lin, C.-H. Tseng, T.-Y. Lee, J.-M. Fu, and W.-B. See, “VERTAF: An Application Framework for the Design and Verification of Embedded Real-Time *Software Engineering*, Vol. 30, No. 10, pp. 656-674, October 2004.

[21] N. Milanovic, J. Richling, and M. Malek, “Lightweight services for embedded systems”, Software Technologies for Future Embedded and Ubiquitous Systems, 2004. Proceedings. Second IEEE Workshop on 11-12 May 2004 Page(s):40 – 44.

[22] The Foundations of Scalable Multi-core Software in Int Building Blocks, Intel Technical Journal, Vol. 11, No. 4, pp. 309-322, November 2007.

[23] S. Akhter and J. Roberts, *Multi-Core Programming*, Intel Press, 2006.

[24] M.Fowler, “Pattern,”IEEE Software, Volume 20, Issue 2, pp.56-57, March/April 2003.

[25] D. Gross and E.Yu., “From non-functional requirements to design through patterns,” In Requirements Engineering, Volume 6, pages 18–36. Springer-Verlag, 2001.

[26] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, “Design Patterns:Elements of

Reusable Software," Addison-Wesley, 1994.

[27] Rui Wang, and Shiyuan Yang, "The design of a rapid prototype platform for ARM based embedded system," Consumer Electronics, IEEE Transactions on Volume 50, Issue 2, May 2004 Page(s):746 –751

[28] J. Gil and D.H. Lorenz, "Design patterns and language design," IEEE Computer, Volume 31, Issue 3, pp. 118-120, March 1998.

[29] Mark Grand., "Java Enterprise Design Patterns: Patterns in Java," Wiley, 2001.

[30] MDA (Model-Driven Architecture), URL: http://www.omg.org/mda/

[31] System Modeling Language (SysML), URL: http://www.omgsysml.org/ , 2008.

[32] Object Management Group (OMG), Unified Modeling Language (UML), URL: http://www.uml.org/, 2008.

[33] Linux Shell Script URL: http://www.gnu.org/software/bash/manual/bashref.html

[34] ARM, ARM Cortex-A9 MPCore, URL: http://www.arm.com/products/CPUs/ARMCortex-A9_MPCore.htm

[35] Intel, Core2TM Duo Processors for Embedded Computing, 2008. URL: http://www.intel.com/design/intarch/core2duo/index.htm?iid=ipp_embed+proc_core

[36] Intel, Quad-Core Intel Xeon Processor 5300 Series for Embedded Computing, URL: http://www.intel.com/design/in

[37] OpenMP, URL: http://www.openmp.org/ , 2008.