多媒體應用的系統單晶片平台設計-以JPEG2000解碼器為例

陳冠宏　王勝德
國立台灣大學電機系
E-mail: sdwang@ntu.edu.tw

摘要

本論文利用 CoWare 公司的 Platform Architect 電子層級設計軟體，來建立處理程序層級的系統單晶片平台，並以多媒體應用 jpeg2000 為例，做軟硬體執行時間的改善。在我們所提出的架構中，軟體負責輸入資料及控制硬體動作，硬體負責執行解碼運算。在硬體改善方面，以改善平台溝通方式為主，方法有兩種，一是減少溝通所需路徑；二是將硬體 IP 改成具有 bus master 功能後，減少由軟體控制每一筆資料傳輸的時間。而後在軟體改善方面，使用快取記憶體及編譯器參數，並針對不同的溝通方式，對應到軟體作原始碼的改寫，而後探討在不同的平台溝通架構下的執行時間改善比率。

關鍵詞：電子層級設計軟體、處理程序層級、系統單晶片。

Abstract

Platform Architect, one of the CoWare corporation's electronic system level design tool, can be used to build and simulate various virtual System-on-chip(SoC) platforms. This thesis explores a jpeg2000 decoder on SoC platforms that are based on Transaction-Level Modeling (TLM). We improve the system execution time by using both methods of hardware and software optimizations. In the hardware part, we refine the communication architecture and make use of master ports on the SoC platform to reduce communication time. In the software part, we enable caches, compiler options and rewriting parts of the source code for the different architectures to improve the system execution time. Finally we can get the best solution from exploring the hardware and software optimizations.

1. 前言

在消費者市場中，趨向將多種功能集中在單一產品上，舉例來說，在多媒體的環境中，對於聲音、影像對於效能有不同的需求，若是要將其集中到單一晶片上，以達到低功率消耗，如此一來，這樣的設計導向變成在一個晶片上，將依各個不同的功能，有所其專職的處理單元，其資料的處理量日益增大，相對而言，溝通傳輸方式扮演著重要的角色，且因處理單元特性的不同，所造成的傳輸方式隨之改變，故其間的溝通傳輸變成效能上的所需要打破的瓶頸。

在軟硬體執行的環境下，一般而言，軟體負責控制硬體，大部份工作分配到硬體，在這個實驗環境下以增進效能為前提做改善，達到每個多媒體應用皆合適的原則。

本論文研究目標是利用 CoWare Platform Architect[3][17][19] (前稱 ConvergenSC)，是一套以 SystemC 為基礎的軟硬體整合開發環境，來模擬一個軟硬體共同執行的平台架構，並就軟硬體最佳化作效能的改善。

平台架構以多媒體應用[13]為主，本實驗採用 jpeg2000 解壓縮程式，並將程式分為軟硬體兩部分：軟體部分負責輸入資料與控制信號，硬體部分負責解壓縮運算。在軟硬體改善中，先對平台溝通方式做改善，會以三個架構做討論，而後在三個架構下做軟體改善，改善比率因架構而異。若要做 jpeg2000 壓縮部分，是做在軟體部分，實驗的重點主要是硬體(資料的溝通所影響的效能)，且因軟體執行較耗時，故不考慮軟體的影響，實驗結果也較能準確呈現，所以就不包括 jpeg2000 壓縮部分。

本論文第二部分針對本論文的背景知識做說明及介紹。第三部分說明實驗流程、平台建立及分析方法。第四部分呈現實驗過程、實驗結果。第五部分對本研究做出結論。

2. 背景知識

SystemC[18]可以用來描述整個系統模型設計，如所圖 1 所示，一個系統能包含許多模組，模組代表軟硬體或小型系統設計區塊，模組內含有並行程序(Process)來描述功能，程序間是利用通道或事件(event)來互相溝通，事件提供了一個較低階的處理程序之間同步及重新啟動的方式，它也能用來實現通道的功能，而事件沒有資料型別，只有控制能力；而模組和模組間的溝通則是利用通道，這是最基本的系統建構模組概念。

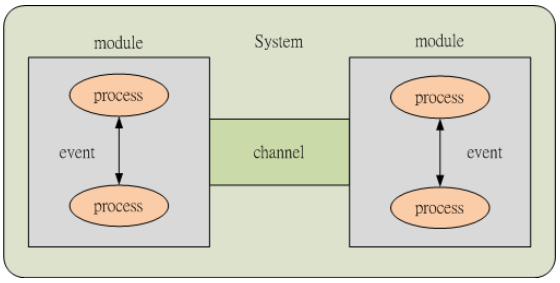

圖 1 SystemC 系統建構模組概念

2.1 處理程序層級系統平台介紹

使用 SystemC 語言的處理程序層級(Transaction Level Modeling, TLM)已經成為越來越受歡迎的抽象層級建構系統方法，可用於單晶片系統架構的前端開發。處理程序層級的執行速度通常比暫存器傳輸級(RTL)高幾個數量級，SystemC TLM 能夠與基於 C++的指令集模擬器(ISS)或主機系統在一起來實現系統層級的直接測試和實際應用。

處理程序層級的主要目標是使模擬加快[16]，它是將溝通抽象化(使用 Application Programming Interface, API 溝通)，不必對底層的細節描述，所以模擬器需要處理的資訊相對減少，模擬因而加快，作為一種更高的抽象層級，降低系統單晶片開發的複雜度。更進一步說明，將實際的通訊協定使其抽象化，就是指處理程序層級的通訊協定。

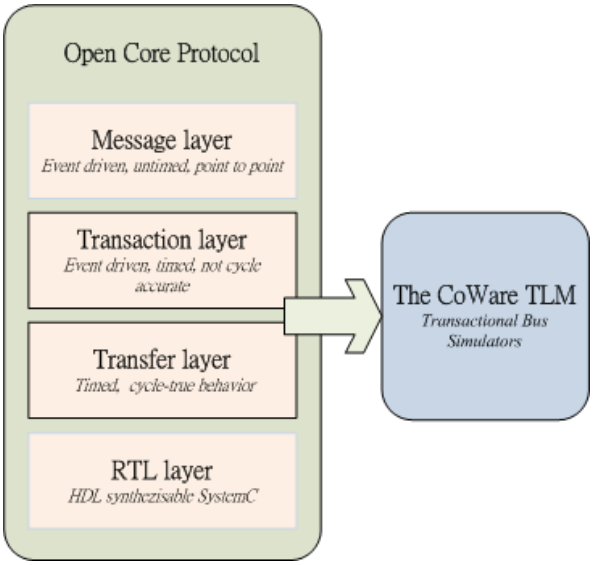

圖 2 系統建構層級

如圖 2 所示，為內核協議國際同盟(Open Core Protocol International Partnership, OCP-IP)對於系統建構層級的訂定，其中 CoWare 公司所建立的處理程序層級的匯流排模擬器(Transaction Bus Simulators, TBS)是建立 Transaction layer 和 Transfer layer 之上[12]。CoWare 公司對於 Transfer，將其定義為具有時脈的 event，而 Transaction 是由 Transfer 所組成，此為 CoWare TLM 的建立基礎。

2.2 CoWare 之匯流排模擬器

CoWare 的匯流排模擬器為週期精確(cycle-accurate)，匯流排模擬器主要處理 Transfer 的資訊，這表示每一個 Transfer 在接收或是被傳送都是在正確的時脈週期並依據所實現的匯流排協定，initiator port 可以傳送 transfer 或 transactions 給匯流排模擬器，當 transaction 被傳送後，匯流排模擬器便會將 transaction 分解成為 transfer，以便於其間的溝通。

以圖 3 表示 CoWare 匯流排模擬器之匯流排模型[4]，是由 systemc 的類別(class)所組成，藉由 method call 作為 initiator port 與 target port 的溝通橋樑。而其組成就是指處理程序層級的通訊協定，為其提供一種抽象的方法(Transactions or Transfers)，使匯流排跟周邊交換資訊。

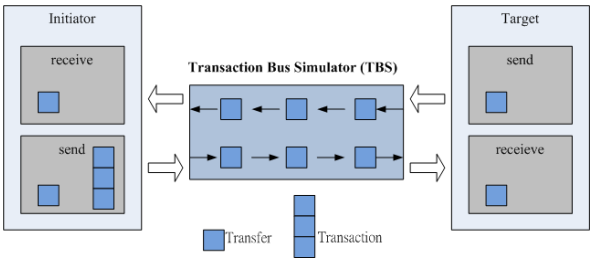

圖 3 CoWare 之匯流排模型圖

以 AMBA 2.0[1]為例，傳送資料時並沒有包含“request” phase，但是在此的通訊協定中，initiator 會對匯流排送出“request” transfer，如果接收到 granted 表示 initiator 得到匯流排的使用權並開始溝通。initiator 只可以接收 transfers(例如 GrantTrf, ReadDataTrf, or EotTrf)。而 Target 可以接收或傳送 transfers。

在一般的通道(channel)中，匯流排模擬器包含所有硬體模組的所使用的協定已達到週期精確，這個觀念是將硬體功能及溝通分開因為使其匯流排模擬器可以完全驅動 initiator ports 與 target ports 之間的溝通。

2.3 Transfer 層級介紹

Transfer 是基於不同的控制信號、位址及資料上的通訊協定，Transfer 層級相當於處理程序層級的信號層級，也就是說 Transfer 代表實際存取的協

定(request, grant, address phase, data phase 或 response phase)，而一個完整的協定是由不同的 Transfer 所組成的。

Transfer level 的匯流排傳輸方式是以 pipeline 為主，當通訊協定有不同的 phase 的時候，其行為(behavior)必須在正確的時間提供所需的資訊(address or data)，當在處理程序層級工作的時候，模組的 behavior 在傳送到匯流排模型前，必須產生 transaction 的單元資訊，此時精確度便依賴匯流排的模型為主，這樣的精確定並不一定在所有狀況下都是準確的，把 transfers 和 transactions 合併起來並使用管線(pipeline)傳輸匯流排。在這樣的傳輸方式的匯流排，行為及溝通兩者特性會交錯。

以下介紹處理程序層級通訊協定與匯流排及周邊通訊關係，如圖 4 表示 transfer 在匯流排跟周邊間之傳輸關係[5]，分別與 Target 或 Initiator 的關係，除了 transfer 傳輸方向不同之外，Initiator 多了 arbitration transfer(ReqTrf、GrantTrf)。

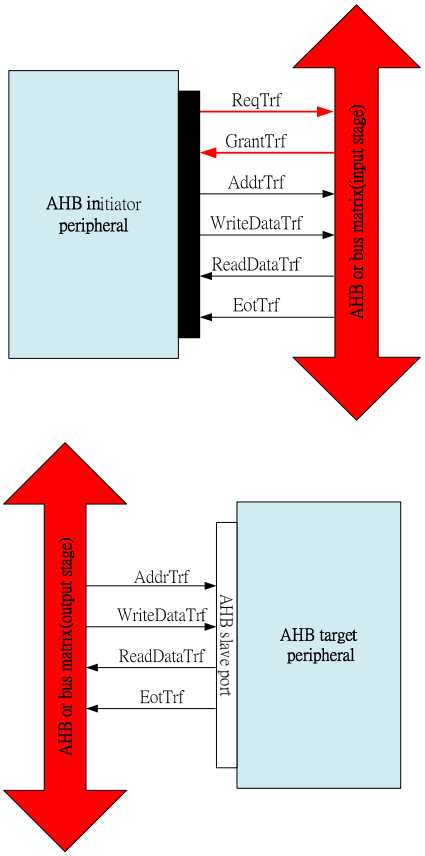

圖 4 AHB與Target與Initiator之間所需要溝通的 Transfer關係

2.4 Transaction、Transfer 和 Attributes 之間的組成關係

在 CoWare TLM 中，把 Transfer 當作是一個最基本的動作單元，Attribute 則是表示 Transfer 的性質，舉例而言，若目前 Initiator 送出位址到匯流排，使用的 Transfer 是 AddrTrf，而其位址、存取的資料大小便是 Attribute。而其 Transaction 是由 Transfer 所組成，所以當在 transaction 層級工作，所有需要設定的 attribute 都是在同一時間設定，若是所使用的 attribute 並不是在同一時間設定，這時候就要使用 transfer 層級工作。

如圖 5 所示，表示 Transaction、Transfer 及 Attributes 之間的組成關係，Attribute 根據匯流排規格而定，若要存取 Attribute 需要透過 method call 來完成。我們的架構都是以 AMBA 2.0 為標準，根據 AMBA 2.0 Bus Library，Transfer 可分為兩類；Arbitration 機制由 ReqTrf、GrantTrf 負責，而交換資料與 AddrTrf、WriteDataTrf、ReadDataTrf、EotTrf 有關，其中 AddrTrf 表示交換資料的種類。

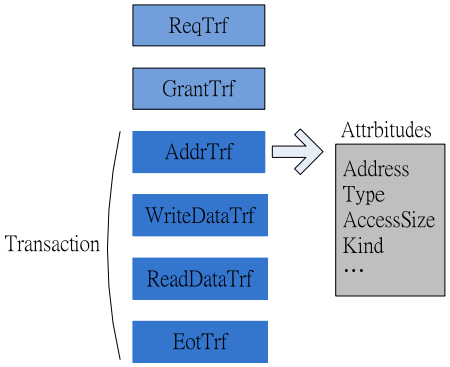

圖 5 Transaction、Transfer 和 Attributes 之間的組成關係圖

3. 方法

在這個章節中，首先介紹所使用的軟體工具，並對實驗流程做一個介紹，並介紹如何使用 Platform Creator 去建立實驗架構，其中包含 Platform Creator 圖形介面介紹，而後提到分析流程。

3.1 實驗使用到的軟體版本及工具介紹

1. Platform Creator V2005.2.2.：以圖形或指令操作介面，建立系統平台與自訂 IP，以便於系統設計者，並提供效能模擬並分析在以 SystemC 為基礎的處理程序層級。由於在處理程序層級，所以軟硬體共同模擬時，可以快速得到驗證，其軟體分析可以依據架構的不同而自動產生，因而加

快整個設計流程。

2. System Verifier (scsh) V2005.1.0：以指令方式操作，可以建立並執行模擬。從載入到除錯是線性流程的關係，並收集相關實驗的數據，也可以運用設定中斷點的方式，追蹤信號或事件(signals or events)，以得到所設計的系統更多的資訊。
3. ARM926EJS_AHB PSP V2005.2.2：CoWare 公司所設計的 IP 套件模型函式庫，函式庫以 SystemC 為組成基礎，模型函式庫提供系統設計師，高品質、高效能的模型以便於平台設計和除錯。ARM926EJS_AHB 的設計依據是 ARM9 系列的中央處理器，由 ARM 公司訂定標準，用於 AMBA 匯流排架構。實驗中，使用的內部記憶體也是源於此模型函式庫，另外實驗中沒有使用 ITCM 與 DTCM 記憶體模型。
4. AMBA Bus Library V2005.2.2：CoWare 公司所設計的 IP 套件模型函式庫，其依據是由 ARM 公司所訂定的匯流排標準。
5. ARM ADS V1.2：ARM ADS (ARM Developer Suite)是 ARM 公司推出的新一代 ARM 整合開發工具，其中包括組譯器、編譯器、連結器、除錯器、模擬器、C/C++函數庫。實驗中用來編譯軟體(ARM core)所需要的執行檔。

3.2 實驗流程

如圖 6 所示，先驗證程式的正確性及做改寫以便於移植到 ARM 平台，選擇適當的 IP 套件模型函式庫，本實驗大部分是選擇 CPU 及匯流排，我們的平台中使用的是 ARM 926EJS[6]及 AMBA 2.0 匯流排，另外也建立 GU IP 為了做 jpeg 2000 解碼器運算；Display IP 顯示出運算結果，藉由以上周邊元件，建立好整個平台後，以效能最佳化為前提做軟硬體改善；其中硬體平台最佳化方面，討論其溝通方式，而衍生出三個架構，使其效能提升，並對 GU、Display IP 其傳輸特性改寫；而軟體最佳化方面，針對三個平台架構，使用快取記憶體[2]，及編譯時加入合適的編譯器參數[10]，並針對架構的不同對軟體部分改寫，有助於效能的提升，並探討其差異處。

3.3 平台建立流程

藉由圖 7 來加以說明建立流程[8]，以下所有步驟 1~6 可使用 Platform Creator 及 System Verifier，不過建議步驟 1~5 使用圖形化介面，Platform Creator 比較快速且易上手；建立模擬及執行模擬，使用指令介面 System Verifier。

1. 開啟所要使用的模型函式庫或以直接開啟所要使用的模組區塊。
2. 放置周邊元件(IP)到 System Diagram 中，並設定周邊元件參數(例如:address width)。
3. 放置所需要匯流排到 System Diagram 中，在加以連線，並設定匯流排參數(例如:arbitration scheme)。
4. 設定周邊元件與軟體的記憶體映照位址，並需與軟體的 boot code (start.s)相符。
5. 軟硬體平台架構好之後，則可以將整個系統輸出到目錄(Exporting)，
6. 藉由輸出到目錄的檔案，建立模擬並執行模擬。

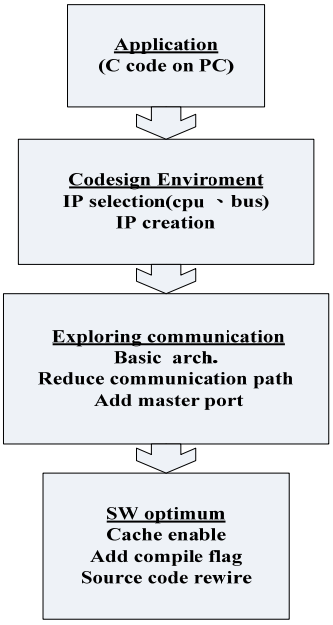

圖 6 實驗流程圖

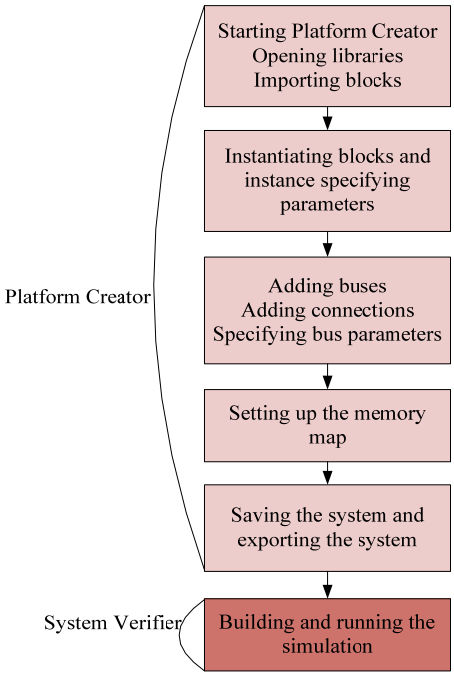

圖 7 平台建立流程

3.4 分析流程

在 Platform Architect 下，所有分析的資訊，都是在執行一個或多個模擬的期間收集， 並會將此資訊存放在同一個資料庫，也可以設定在編譯模擬時收集資訊，藉著在模擬前加入特定的分析函式。分析函式分為四類(Generic Analysis、Software Analysis、Port/Memory Analysis、Bus Analysis)[7]，實驗中利用軟體分析(Software Analysis)函式[9]，量測執行系統模擬時間。

在執行模擬時，主要工作是設定其所要分析資料的項目設定，例如收集分析資料時，設定其時間間隔大小，這個階段稱之組態階段(configuration phase)。

所有分析項目的資訊收集，都是執行以下步驟：

1. 若需在模擬時收集資料，則需加入 CoWare 設計的分析函數，分析函式在 SystemC 中如同 C++ 的類別，定義資料如何處理及資料表達方式(例：時間單位)，用於執行模擬時收集所需資訊。
2. 在開始執行模擬器前，設定是否要執行分析資料收集並設定相關檔案(file setup)，檔案設定包含記憶體映照檔案或符號檔案(symbol file)。若要執行記憶體分析，則需設定軟體所對應的記憶體映照檔案；若要執行軟體分析，則需設定其欲輸入的執行檔之符號(symbol)檔案。CoWare 提拱的 SystemC IP 已附有其分析函式，所以實驗中所使用的 ARM926EJS_AHB PSP、AMBA 2.0 transactional bus simulator，不需另外加入分析函式。
3. 在建置階段(elaboration phase)後及開始真正的模擬前，當模擬器已經將模擬建置完畢後，並知道何時載入相關物件，此時設定欲分析項目之資料收集，並設定所呈現形式，包含模擬開始到結束的資訊收集時間、資訊收集的時間間隔…等等，這邊稱為組態階段(configuration phase)，組態階段中可以使用圖形或指令操作介面。
4. 繼續執行模擬。

模擬結束後，執行 sdviewer(postprocessing)以顯示在執行模擬期間所收集的分析資訊，可以個別顯示或以群組方式顯示所收集資訊。在實驗中，軟硬體執行時間量測，因為不包含系統初始化的時間(執行 boot code 的時間)，所以不是把所有函數的執行時間加總，而是以 main( )的 totalTime 為軟硬體執行時間，為實驗中的量測標準。

4. 實驗過程和實驗結果

我們會先建立一個基本平台[15]，平台中在軟體所執行的程式與硬體GU 、Display IP 為自訂的，其他部分是使用 CoWare 所設計的模型函式庫(ARM core，匯流排，外部記憶體)，所實驗的架構中，都是以 jpeg2000 解碼運算為應用，軟體部分為輸入資料及控制硬體動作，GU IP 則是進行解碼器運算，Display IP 為顯示運算結果。

在實驗前，先將大小 400x300 的圖以 jpeg2000 編碼運算後存成一個數值串流(streaming)格式的檔案，此為系統中在軟體部分所輸入，欲做解碼運算的資料。

實驗過程如下，首先建立硬體平台架構，並以三個架構來討論三種不同的資料溝通傳輸方式，相對而言，軟體部分也會因硬體不同而也有所不同；而後軟體最佳化方面，方法相同但由於各個架構不同所呈現出來的結果也會有所不同，這部分也會加以討論彼此的差異。

4.1 基本架構(架構一)

以下這個基本架構(圖 8)執行 jpeg2000 解碼器，在軟體輸入欲解碼的資料，再傳送到 GU，進行解碼，將解碼後的資料送到 Display，資料量為 15238 筆。由於 GU 跟 Display IP 都是以 slave port 進行傳輸，所以每一筆的傳輸資料都須經由軟體控制。

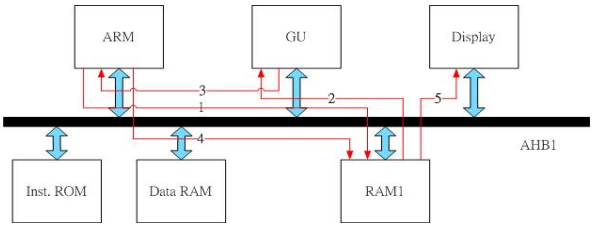

圖 8 架構一

架構一傳輸順序及路徑如下：
動作 1：將待解碼的資料由軟體送到外部記憶體(RAM1)
動作 2：將外部記憶體的資料送到 GU IP 進行解碼器運算
動作 3：將解碼器運算後的結果送到軟體，以陣列方式儲存
動作 4：把以陣列儲存的資料送到外部記憶體(RAM1)
動作 5：把外部記憶體的資料送到 Display IP 顯示運算結果

4.2 減少傳輸路徑(架構二)

由架構一的資料傳輸方式可看出，若是不使用外部記憶體，傳輸方式改以直接在匯流排上交換資料如圖 9 架構二所示，減少兩次傳輸動作，則必能對效能有所提升。

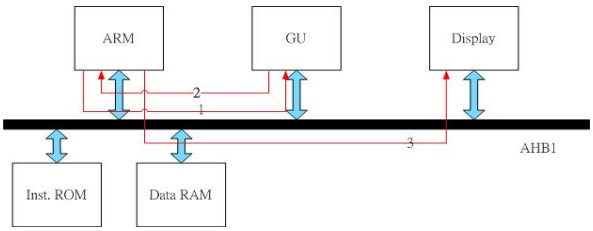

圖 9 架構二

架構二的傳輸順序及路徑如下：
動作 1：將待解碼的資料由軟體(ARM core)送到 GU IP
動作 2：將解碼器運算後的結果送到軟體，以陣列方式儲存
動作 3：把外部記憶體的資料送到 Display IP 顯示運算結果

此種減少傳輸路徑的做法，是由於硬體以 slave port 傳輸資料所致，架構一的外部記憶體，成為不必要的傳輸路徑，將架構一改成 master port 傳輸後，並使用中斷機制，使硬體做完工作後，可以回傳訊息到軟體，外部記憶體則能發揮作用，做為軟硬體間傳輸資料的橋梁，故外部記憶體有助於增加效能。

4.3 硬體 IP 的改善(架構三)

在架構一，由於 GU 跟 Display 都是以 slave port 進行傳輸，所以傳輸資料都須經由 ARM core，若 GU 與 Display IP 可以獨立接收與傳送資料，則可以增進效能。所以分別在 Display 與 GU IP 改成 master port 負責傳送與接收資料且增加中斷機制(ICU)。將 IP 改成具有 master 功能存取資料時不需要經過軟體控制，所以當 IP 動作完畢後，必須通知軟體，所以增加中斷機制，如圖 10 所示。

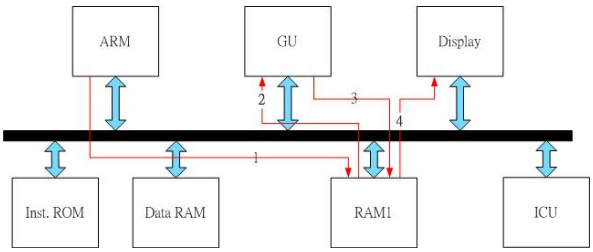

圖 10 架構三

架構三的傳輸順序及路徑如下：
動作 1：將待解碼的資料由軟體送到外部記憶體(RAM1)
動作 2：GU IP 讀取外部記憶體的資料並進行解碼器運算
動作 3：GU IP 將運算結果寫入到外部記憶體
動作 4：Display IP 讀取外部記憶體資料並顯示運算結果

4.4 軟體最佳化

因為各個多媒體演算法皆不同，這邊並不探討 jpeg2000 解碼運算的演算法，表示所有多媒體運算皆適用此次實驗平台，在這個原則下做軟硬體改善。ARM926EJ-S 具有快取記憶體的功能，使用快取記憶體(cache)；並在編譯執行檔時使用參數優化級別(-O1、-O2)與優化方向(-Otime)[10]；並針對原始碼改寫，可藉此增進效能。

4.4.1 開啟快取記憶體後的改善

因為使用快取記憶體會造成 cache miss，但由於在編譯執行檔時使用參數(-O2)，會讓編譯後的原始碼較緊密，因而減少 cache miss，對於 cache miss rate 並沒有做測量，因為在這邊主要討論的是效能分析。在 CoWare 設計的模型函式庫中，ARM926EJ-S 處理器具有 Instruction Cache 及 Data Cache，大小有 4, 8, 16, 32, 64 和 128 KB 可選擇使用，我們的環境使用 16KB，軟體在編譯時應將其開啟。

4.4.2 加入編譯器參數優化級別(-O1、-O2)與優化方向(-Otime)

由於-O2 參數會使編譯後原始碼比較緊密，所以會相對減少所需執行的指令數及模擬時間。由於-Otime 參數告知編譯器編譯任務的主要目標是執行速度，所以會相對減少所需執行的指令數及模擬時間。

4.4.3 改寫原始碼

由於軟體部分是控制硬體信號及輸入欲解碼資料，控制硬體信號這方面隨著平台溝通方式而變，所以改寫部分以輸入欲解碼資料這部分為主，這部分的改善有兩種方法。

1. 將欲解碼資料先載入記憶體在軟體方面，本來讀取資料的方式是以 streaming 方式讀取檔案，所以讀取占了相當多的時間。為了減少讀取資料的時間，將欲解碼的資料改成以陣列方式儲存(存放在內部記憶體 Data RAM)，所以在編譯所需的執行檔時，會將欲解碼的資料編譯進去，便減少讀取資料的時間。
2. 將欲解碼資料壓縮在軟體方面，壓縮後的資料也是存放在內部記憶體(Data RAM)，由於是內部記憶體，故支援快取記憶體可加快資料存取，在這邊的資料壓縮方式，是以位元位移的方式，將欲解碼資料以兩筆合併成一筆資料，其中所需的運算量，須小於所節省傳輸的時間，才有改善的動機。如此一來，由於內部記憶體傳送到硬體

GU IP(軟體傳送到硬體 GU IP 的資料量便少)，以節省所需執行時間。

5. 實驗結果

經由軟體的改善以循序漸進的方式呈現，由於各個架構的不同，其所軟體改善也其有所不同，如表 1 所示，arch1、2、3 分別表示三個架構，totalTime 表示軟硬體執行時間，improve 表示執行時間改善比率，為三方面的改善比率及整體改善比率表達，selfTime 表示為個別的執行時間改善。

表 1 軟體改善結果

arch1	totalTime	improve	selfTime
original	52989824		
1. data cache	42895968		10093856
2. ins. dache	37105152		15884672
3. data and ins. cache	25444072	51.98%	
4.O1	19113408		
5.O2	18992240		
6.Otime	18748264	26.32%	
7.load data to memory	13323664		5424600
8.data compress	11324184	39.60%	1999480

arch2	totalTime	improve	selfTime
original	42138880		
1. data cache	31070128		11068752
2. ins. Cache	30034344		12104536
3. data and ins. cache	17394328	58.72%	
4.O1	12991512		
5.O2	12748032		
6.Otime	12504272	28.11%	
7.load data to memory	7080912		5423360
8.data compress	6508624	47.95%	572288

arch3	totalTime	improve	selfTime
original	32087088		
1. data cache	21451536		10635552
2. ins. Cache	25565488		6521600
3. data and ins. cache	12366544	61.46%	
4.O1	10047544		
5.O2	9802440		
6.Otime	9803384	20.73%	
7.load data to memory	4378672		5424712
8.data compress	3487784	64.42%	890888

軟體改善分為三方面，1~4 表示為使用快取記憶體，使用 data cache 或 instruction cache，或兩種 cache 一起使用的三種情況；5~7 表示使用編譯參數；8~9 為原始碼的改寫，且以三個架構來綜合討論：

以快取記憶體的改善而言，就使用 data cache 而言，由於輸入的資料量三者都是相同，所以使用 data cache 所減少的時間(selfTime)都差不多；就使用 instruction cache 而言，架構一相對於架構二而言，架構二的溝通次數減少了，架構二相對於架構三而言，架構三不需要每一筆資料傳輸都經由軟體控制。越後面的架構，軟體的工作越來越少，所以使用 instruction cache 所減少的時間為遞減的情形。

若將三個架構使用 data cache 與 instruction cache 比較可看出架構一和架構二的使用 data cache 小於使用 instruction cache 所節省的時間，而架構三則是相反的情形，使用 instruction cache 小於使用 data cache 所節省的時間，這是因為架構一和架構二的軟體對硬體的控制方式都是相同所致，由軟體控制每一筆資料的存取，故需要較多的 instruction，而架構三與前兩者不同。

以使用編譯參數而言，架構一與架構二兩者改善比率差不多，是由於兩者在軟體方面的動作差異在於減少平台溝通次數，架構三改善比率與前兩者架構一和架構二差異較大，是由於架構三軟體部分並不控制每一筆資料的傳輸，其主要工作是輸入資料及控制硬體何時動作。

就資料載入記憶體以減少讀取時間而言，因為輸入資料原本是以 streaming 方式讀取，將改成以陣列方式存取，輸入資料皆相同所致。三個架構所節省的時間都差不多，其間的差距是因為架構不同所致。

就資料壓縮以減少傳送到 GU IP 的資料量而言，架構一中傳送到 GU IP 的路徑為 2 次，架構二傳送到 GU IP 的路徑為 1 次，架構三傳送到 GU IP 的路徑為 2 次。以傳送路徑次數可看出，架構二個別的執行時間改善最少；架構一、三中雖然傳送路徑次數相同，由於架構三改以 master port 傳輸，所以執行時間改善架構三少於架構一。

表 2 硬體平台溝通時間

	totalTime	instruction	improve
arch1	11324184	520638	
arch2	6508624	383493	42.52%
arch3	3487784	222913	69.20%

由實驗結果中，三個架構的總體改善比率呈現遞增，表示經由硬體改善溝通方式後，對軟體改善相對有助益。將三個架構的軟體改善最佳化，整理後如表 2 呈現。

6. 結論

在單晶片系統中，將複雜、固定、耗時的運算單元實作在硬體，軟體則是依需求不同，輸入欲處理的資料及控制硬體動作，以提高執行效率。故須先建構硬體平台，而後建立軟體，並針對不同平台架構做軟硬體改善。

在硬體改善中，探討的是減少資料傳輸路徑與減少需要由軟體控制每一筆資料讀寫的時間；在軟體改善中，針對硬體架構的不同，而對其做改善，並分別討論不同架構間其改善的差異，本論文經由軟硬體的改善而得到最佳解。

實驗中，所討論溝通的資料流皆為單向，若要考慮資料流為雙向的情況，須將硬體 IP 進一步改寫使其存取資料時序不需要由軟體控制，以暫存器去溝通硬體間的動作時序，取代軟體控制，但軟硬體間會有時序同步的問題待解決，這是未來需要克服的問題之一。

參考文獻

[1] 吳欣龍：Introduction to AMBA Bus System。2007 年 5 月 1 日，取自 http://tpe-wh3.dwins.net/download/member_file/2002/soc/2002-5-1.pdf.

[2] 錢偉德 (2006)：利用 Platform Architect 完成 ARM-Based Platform 之設計與效能分析。CICeNEWS，74 期，頁 3-14.

[3] CoWare Inc., http://www.coware.com/products/platformarchitect.php

[4] CoWare Inc. (2003, October). Application Note: TLM SystemC Coding Guidelines Using the Bus Simulator, Version 4.1.

[5] CoWare Inc. (2006, July). CoWare Model Library: AMBA Bus Library, Version V2005.2.2.

[6] CoWare Inc. (2006, July). CoWare Model Library: ARM926EJS_AHB PSP, Version V2005.2.2.

[7] CoWare Inc. (2006, July). CoWare Platform Architect/CoWare Virtual Platform Product Family: Analysis Manual, Version V2005.2.2, 17-27.

[8] CoWare Inc. (2005, May). Application Note: Creating Platforms with Platform Creator, Version 2005.1.0 .

[9] CoWare Inc. (2005, May). Application Note: Software Analysis with ConvergenSC, Version 2005.1.0.

[10] Dalal V. & Ravikumar C. P. (2001). Software Power Optimizations In An Embedded System, VLSI Design 14th International Conference, 254.

[11] GWIC, http://www.jole.fi/research/gwic/

[12] Larsen, G. (2005). technical training: Transaction level Modeling, CoWare Inc., 1-5.

[13] Lei, T., Yanhui, Y., & Shaojun, W. (2005, October). Optimizing SoC Platform Architecture For Multimedia Applications. ASICON 6th International Conference, 1, 94-97.

[14] Liu, C. N. & Tsai, T. H. (2005, May). SoC Platform Based Design of MPEG-2/4 AAC Audio Decoder," Circuits and Systems, 3, 2851-2854.

[15] Ogawa, O., Noyer, S.B., Chauvet, P. & Shinohara, K. (2003, March). A Practical Approach for Bus Architecture Optimization at Transaction Level. European Design Automation Conference, 2, 176-181.

[16] Pasrichat, S., Dutt, N. & Ben-Romdhanet, M. (2004). Extending the Transaction Level Modeling Approach for Fast Communication Architecture Exploration. Design Automation Conference 41st, 113-118.

[17] Rompaey, K. V., Verkest, D., Bolsens,I., & Man, H. D. (1996). CoWare – A design environment for heterogeneous hardware/software systems. European Design Automation Conference, 252-257.

[18] SystemC, http://www.systemc.org/

[19] Wieferink, A., Kogel, T., Hoffmann A., Zerres, O. & Nohl, A. (2003, November). Session: Open Forum on SystemC, SOC INTEGRATION OF PROGRAMMABLE CORES, IP Based Design 2003, CoWare Inc.