晶片系統之軟硬體共同設計

黃駿賢、熊博安

國立中正大學資訊工程學系 嵌入式系統實驗室

一、簡介

在這個競爭激烈的時代，一個電子產品的設計，受到市場時間極嚴格的挑戰。而產品處於領先的時機相當短暫，通常是以月來計算。因此產品導入市場的時間早晚，對於獲利的多少有著極大的影響。以目前的狀況，大多數的專業工程師不是侷限在硬體設計，就是在軟體設計，通常無法兩者兼俱。

而系統晶片(Soc, System-on-a-chip)一般又以現場可規劃閘陣列(Field Programmable Gate Array, FPGA)和積體電路(ASIC)來實現。傳統設計的方法(如圖 1)，在設計的早期就將軟硬體明顯分離，由於軟體設計者不熟悉硬體的架構，硬體設計者不了解軟體設計的方法，軟硬體個別獨立設計，最後階段才將軟硬體整合。若合成發生問題，常常都無法確定是軟硬設計上何種錯誤，導致修改上的困難度提高，延遲了產品上市的時間。再者，隨著晶片容量增大，功能需求增加，嵌入式系統的複雜度也相對地提高，這種傳統設計方法，在最後合成時除了時間的延遲外，也會浪費人力和成本在系統整合上。

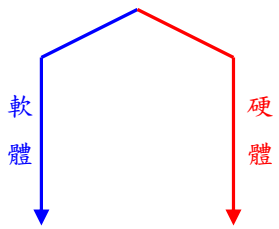

圖 1.傳統設計流程

然而，晶片設計的生產力在這幾十年來是有目共睹，但是與晶片容量的成長，卻是有著越來越大的落差。因此，一個新的設計方法論就因應的誕生－軟硬體共同設計(Software/Hardware Co-design)。這是一個以系統觀點出發的方法論，強調軟硬體設計時的一致性與整合性(如圖 2)，對於軟體發展的驗證(Verification)更加容易，且硬體設計的錯誤也能提早發現，降低設計所需的時間與成本，大大縮短系統設計的時間，因應市場的即時性。

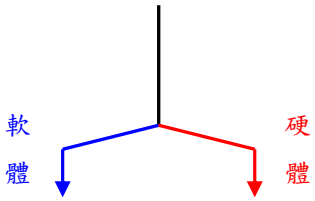

圖 2.軟硬體共同設計流程

以下分成五個部分，我們將介紹一些用於軟硬體共同設計的工具，典型的設計流程，並提出一個例子加以說明，最後做結論並列出我們的參考資料。

二、工具的使用

以下就一些常用於軟硬體共同設計流程中的工具做簡單介紹。

Mentor Graphics 公司所出的 Seamless Co-verification Environment，適用於軟硬體共同模擬(Co-simulation)與共同驗證(Co-verification)，所支援的嵌入式系統的控制器和數位訊號處理器(DSP)就超過一百個以上。藉由 Seamless，硬體設計師可以直接用嵌入式程式碼來當成測試平台(Testbench)，而硬體直接傳資訊給軟體去測試，就不需要撰寫一些特定功能來模擬硬體的狀態，大大提高驗證時的真實性。此外，它加入記憶體最佳化的技術，將記憶體的資料複製一份到一致性記憶體伺服器(Coherent Memory Server)，軟體若要存取記憶裡的資料，不需再到真實記憶體存取，加快模擬

的速度。

Synopsys 公司所出的 CoCentric System Studio(簡稱 CCSS)，是基於 SystemC 的系統發展與模擬工具。而 SystemC 是以 C++為基礎的系統程式語言，包含描述軟硬體及系統各部分功能的類別函式庫(class library)，能在軟硬體之間交互運作溝通。CCSS 提供一個設計中心(Design Center)供使用者自行設計或管理系統的 IP 和測試平台，且包含一個模組庫可供使用者選用，最後用 DAVIS 呈現出模擬結果，做為系統前端設計工具。

Coware 公司所推出的 ConvergenSC，是以 SystemC 為基礎，主要針對系統架構和設計，切分成三個部分。System Verifier 是針對系統做模擬和除錯，System Designer 是對系統階層的硬體、軟體、匯流排和記憶體做效能的分析，而 Advanced System Design 用於平台的組合和建構，最佳化軟硬體的 SystemC 系統。其它如 IP 的發展，軟體框架設計等，都可以使用這套軟體來實作。

其它如 Cadence 的 NCLaunch，Xilinx 的 ISE，Agilent 的 ADS 等，也是常用於晶片系統軟硬體共同設計的工具，若讀者有興趣可以至這些公司得到更詳細的認識。

三、軟硬體共同設計流程

一般而言，典型的軟硬體共同設計流程，可以用圖 3 來表示。

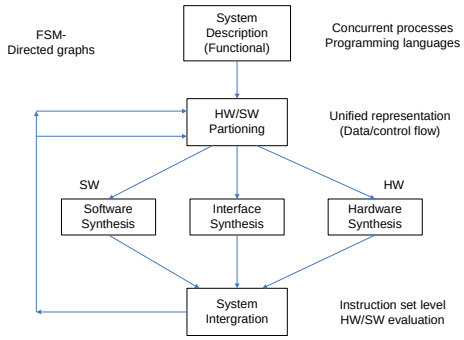

圖 3.典型軟硬體設計流程

軟硬體共同設計的目的，是為了軟體與硬體的協同描述、驗證與合成提供一個整合的環境，以圖 3 可知，首先需對你所設計的系統的規格詳細描述與分析，有哪些功能，需要哪些硬體的元件，控制流程為何等。其次利用一套合適的演算法，將軟硬體部分切割，並設計一個介面供軟硬體溝通之用。最後便是將軟硬體部分與溝通的介面實作出來，並加以整合，做軟硬體共同的驗證(Verification)與測試(Testing)，再依設計成本、執行時間或低功率(Low power)等不同方向考量，若需要改進再重做軟硬體切割，直到最佳的效果。

以下我們再深入介紹晶片系統之軟硬體共同設計方法，如圖 4 所示，大至可以分成三個部分：甲.統一表述；乙.軟硬體切割；丙.軟硬體合成。

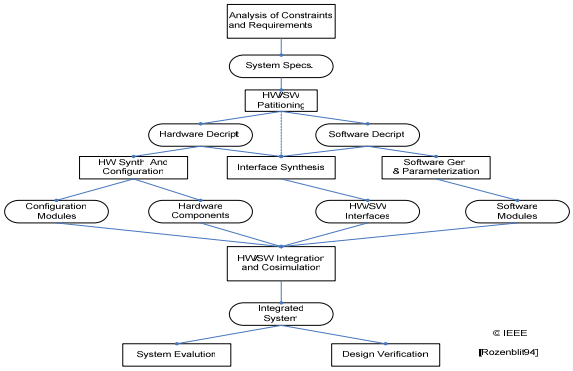

圖 4.公定軟硬體共同設計方法論

甲.統一表述

模型(Models)來說，是以系統功能性為觀點，而結構(Architectures)為系統實作的抽象化，兩者之間的關係可以用圖 5 來表示。

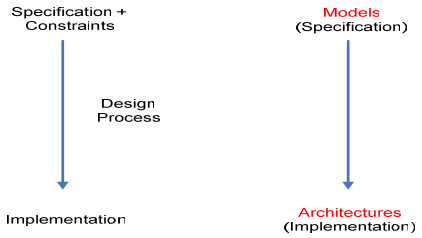

圖 5.模組與架構關係

將抽象化軟硬體模型統一表述的目的，在於系統可以較早做效能的分析，一般常用的有狀態導向模型(State-Oriented Models)，如有限狀態機(FSM)、Petri-Nets、階層並行有限狀態機

(Hierarchical Concurrent FSM)；行為導向模型(Activity-Oriented Models)，如資料流程圖(Data Flow Graph)、Flow-Chart；結構導向模型(Structure-Oriented Models)，如區塊圖(Block Diagram)、RT netlist、Gate netlist;資料導向模組(Data-Oriented Models)，如實質關係圖(Entity-Relationship Diagram)；不同成分組合模組(Heterogeneous Models)，如 UML(Unified Modeling Language)、CDFG(Control-Data Flow Graph)、PSM(Platform-Specific Models)、Queuing model、Programming Language Paradigm、Structure Chart 等。

結構為系統實作抽象化，明確地訂出軟硬體的組成元件，再與系統行為作對映。而語言的使用，常見有硬體描述語言，如 VHDL、Verilog;軟體程式語言，如 C、C++、Java;結構描述語言，如 EXPRESSION、LISA;系統規格語言，如 SystemC、SDL;驗證用語言，如 PSL、OpenVERA。

對於模組、結構和語言等，在設計的初期就需要明確訂定，以確保系統設計的一致性，利於之後的軟硬體切割與合成的部分。

乙.軟硬體切割

因為要符合不同的效能需求，如成本、低功率，執行時間等，必須就系統不同特性，做軟硬體切割來達到最近乎我們期望的結果。一般而言，若用硬體來實作，執行時間和平行處理的效能最高，但相對地所需的成本也提高。而採用軟體來實作，雖然成本會降低，但是相對地執行時間也需要較多。基本來說，切割可以用圖 6 來表示。

依圖 6 可以分成四個步驟。首先將高階部分抽象化表示，以系統規格做為切割的依據，利用如資料流程圖(Dataflow graph)或有限狀態機(FSM)等描述控制的流程。第二，就執行時間、成本、電源消耗、資料大小、程式大小等的單元(metrics)做為軟硬體切割評估的依據，且以結構而言，這些單元是軟體實作其功能為目標，若在切割時沒有這些單元存在便難以計算所需代價。然而，這些評估依據常常是兩兩互相影響，每單元都有其重要性存在，此時就需要將這些不同的單元統一用一個單位來當成評估的標準。如:

成本

=K1*(程式大小)+K2*(延遲時間)+K3*(耗電量)

(K1,K2,K3 為常數)

接著利用一個合適的切割演算法來切割出最佳的情況，一般常見的演算法如隨機對映法(Random mapping)、比例切割法(Radio cut)、基因演化法(Genetic evolution)、模擬退火法(Simulated Annealing)等。第三步是手動或是做切割演算法同時，將構成系統的各個要素做分配(Allocation)的動作，確定系統最後規格。第四步即將這些結果輸出，以做為實作時的依據。

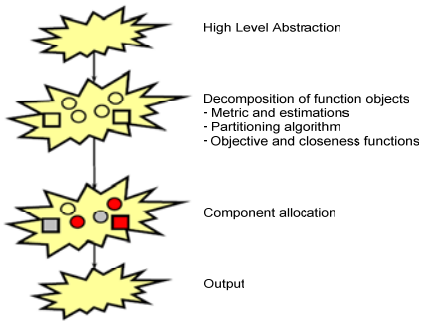

圖 6.切割的流程

丙.軟硬體合成

這邊主要就是實作的部分。以硬體而言，先採用硬體高階合成工具，加上行為描述與系統設計限制的條件，再以硬體描述語言來模擬，並採用低階合成工具將暫存器轉換成即時系統階層和低階設計電路。而軟體除了組件規格(Module Specification)的程式撰寫外，必須對更詳細的程序也要加以撰寫，而在高階時使用如 C 和 C++來編碼，再做單元、整合、系統、復原和接受等的測試(Testing)。再來使用共同模擬(Co-simulation)的方法，建立混合硬體、軟體、即時作業系統和介面的模型，模仿即時作業系

統對 I/O 做監控(Monitoring)和排程(Scheduling)的動作，最後使用共同驗證(Co-verification)的工具對系統做分析，改善其效能至最佳狀態，最後實作到晶片。

四、實例說明

以下就簡單的數位相機設計當成例子說明，所需要的如擷取影像的 CCD，儲存裝置 ROM 或 RAM，JPEG 壓縮的 IP 或程式碼，控制流程的 Controller，硬體加速器，影像輸出裝置等。我們希望影像處理速度要快，晶片設計越小越好，電源的消耗低等要求，再依據這些條件建立設計流程圖(如圖 7)。而這邊我們使用統一塑模語言（Unified Modeling Language, UML）進行系統的分析與設計，也就是繪製出整個系統的軟體藍圖，採用類別圖(Class Diagram)描述軟體架構，順序圖(Sequence Diagram)和狀態圖(State chart)來描述各類別執行順序與行為之間的情況(如圖 8)，再使用 Rhapsody 5.0 依這些圖產生 C++程式碼驗證是否錯誤。在硬體架構我們使用 SystemC(如圖 9)來實作，依照不同組件(module)建立，並利用匯流排(Bus)做為各個組件溝通的介面，再將所撰寫的 SystemC 程式碼放到 CoCentric 來分析效能，確定是否重做軟硬體切割。若軟硬體切割已符合我們要求，軟硬體各自將其部分實作出來，同時系統也必須做排程，並將軟硬體部分共同合成，丟到 ModelSim 5.8 做系統驗證和效能分析。假設已符合最佳化需求，便可以燒錄到系統晶片。

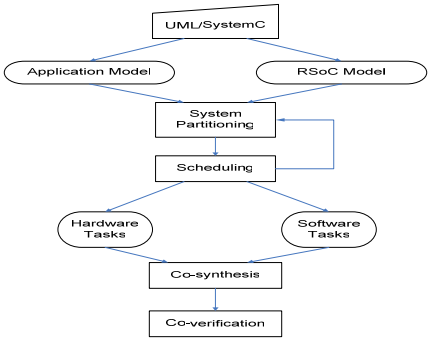

圖 7.簡單數位相機設計流程

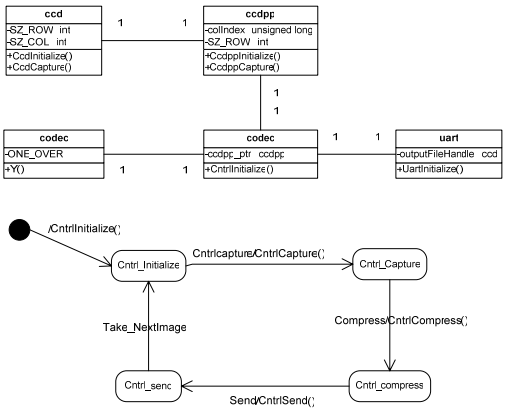

圖 8.數位相機 UML 之類別圖和狀態圖

```
SC_MODULE(ccd)
{
        sc_in<bool>             rst,clk, rd ;
        sc_out<sc_lv<8> >       data;

        sc_signal<int> row;
        sc_signal<int> col;

        void      main_proc();

        SC_CTOR(ccd)
        {
                SC_METHOD(main_proc);
                sensitive << rst;
                sensitive_pos << clk;

        }
};
```

圖 9.CCD 部分之 SystemC 程式碼

五、結論

臺灣科技業發達在國際有目共睹，而晶圓的產量與技術更是處於領先的地位。然而，臺灣在晶片系統的設計上卻沒因此並駕齊驅，乃歸因於仍採用傳統的軟硬體分開設計的方式，耗在系統整合與驗證時間太長，導致在晶片系統的開發上沒有如此出色。因此，本文希望將「晶片系統之軟硬體共同設計」的方法論介紹給大家，推廣此一方法論來降低產品設計時間，提高我們在市場的競爭力。

六、參考資料

1. F. Vahid and T. Givargis, Embedded System Design: A Unified Hardware/Software Introduction, John Wiley & Sons, 2002.
2. OMG workshop ,UML

homepage.http://www.uml.org

3. The Open SystemC Initiative(OSCI). SystemC homepage.http://www.systemc.org
4. Getting Started with CoCentric System Studio Architectural Modeling Student Guide,Synopsys Inc.,USA.
5. CIC,Seamless Training Manual
6. CoWare Inc. http://www.coware.de/coware/index_pc_d.html
7. I-Logix Inc. Rhapsody homepage. http://www.ilogix.com/rhapsody/rhapsody.cfm