計畫名稱：**硬體-軟體同步設計方法論** (I)

Hardware-Software Co-Design (I)

計畫編號：NSC 88-2215-E002-037

執行期限：87 **年** 8 **月** 1 **日 至** 88 **年** 7 **月** 31 **日**

主持人：**陳少傑　　臺灣大學電機工程研究所　教授**

一、 摘要

幾年來，由於積體電路製程技術以及 VLSI **設計技術之進步，使得電子零組件之功能愈形豐富，性能也不斷提升，複雜度也愈來愈高。一個系統晶片可包括處理器或** ASIP、ASIC、**記憶體、介面及聯結系統等。一個最佳化的應用系統必須把運算工作適當地分配到處理器或**ASIP **上的軟體及**ASIC **內的硬體去執行。**

目前的設計環境缺少專為硬體－軟體同步設計的需要而發展之輔助工具，因此設計生產力與效率偏低。所以硬體－軟體同步設計的主要研究就是如何定義出描述系統規格 (System specification**）之工具及如何完成硬體－軟體間的分工合作（**Hardware-Software partitioning**）。這通常需要一個能夠處理硬體－軟體同步模擬（**Co-simulation**）的環境來支援硬體－軟體分工合作的驗證及分析其成效。**

Abstract

Recently, due to the progress of VLSI design and fabrication technologies, the functions of electronic components become more and more complicated. Thus, the complexity of design increases much rapidly. A system-on-chip usually comprises of a processor core, application-specific instruction set processor (ASIP), some ASICs, memory, interface, and their inter-connections. A system designer is required to optimally divide the application tasks into software (S/W) and hardware (H/W) components and to distribute them to the processor core (or ASIP) and ASIC, respectively.

Nowadays, little CAD tools are designed for the H/W-S/W co-design, leading to low productivity and long development time. The main research objectives of H/W-S/W co-design are as follows: (1) define a language suitable for H/W-S/W specifications, (2) accomplish the H/W-S/W partitioning, and (3) need a H/W-S/W co-simulation environment to evaluate the correctness and efficiency of the H/W-S/W partitioning and synthesis results.

二、 計畫緣由與目的

近幾年來，由於積體電路製程技術以及 VLSI **設計技術之進步，使得電子零組件之功能愈形豐富，性能也不斷提升，複雜度也愈來愈高。一個系統晶片上可包括處理器或** ASIP、ASIC、**記憶體、介面及聯結系統等。一個最佳化的應用系統必須把運算工作適當地分配到處理器或**ASIP **上的軟體及**ASIC **內的硬體去執行。硬體－軟體的同步設計對系統設計師是一大挑戰。硬體與軟體元件的使用與再使用能比傳統的積體電路設計方法花更少的時間而產生更高品質（即成本／效能、適應性等）的產品。**

目前的設計環境缺少專為硬體－軟體同步設計的需要而發展之輔助工具，因此設計生產力與效率偏低。所以硬體－軟體同步設計的主要研究就是如何定義出描述系統規格（System specification**）之工具及如何完成硬體－軟體間的分工合作（**Hardware-Software

partitioning）。這通常需要一個能夠處理硬體－軟體同步模擬（Co-simulation）的環境來支援硬體－軟體分工合作的驗證及分析其成效。

完成系統的硬體-軟體間的分工合作可說是最重要的工作，因為它直接影響到系統最後的成本／效能(cost/performance)。因此任何的分工合作的決定都必須將其中的硬體與軟體的特性考慮進去。

硬體－軟體間的分工合作的問題會根據同步設計所面對問題的不同而有所不同。以內嵌式系統（embedded system）為例，硬體－軟體間的分工合作的主要工作是將系統的實體功能分離為特定應用（application-specific IC, ASIC）硬體與在處理器（Processor core）上執行的軟體。各種硬體－軟體間的分工合作的問題都需要以（一）目標架構（target architecture）的設定、（二）分工合作的目標（partitioning goal）與（三）分工合作的策略（strategy）等三項基礎來作比較分析。茲分述如下：

（一）目標架構：因為一般的內嵌式系統都是由核心處理器（Core）與特定應用硬體（ASIC)製作而成，所以我們可以將這些系統視為是一種由處理器與特定硬體間的相輔處理（coprocessing）。這種相輔處理的架構因硬體元件與軟體元件間的平行程度而有所不同。例如：相輔處理的硬體可能由處理器直接控制[1]，或是相輔處理是由軟體平行執行來完成[2]。同樣地，最終目標架構的處理器的選擇也直接影響到分工合作的策略。

硬體／軟體介面定義出另一個直接影響合工合作問題描述的架構性變數。此介面通常指的就是由處理器的記憶體對映的（memory-mapped）I/O 所管理的通訊介面[3]。然而，記憶體對映的 I/O 對資料傳遞是一種無效率的方法[4]。對相輔處理架構有更多有效的通訊方法被提出[5]，這些通訊方法與分工合作的關係必須清楚地描述出來。

（二）分工合作的目標：相輔處理的架構通常是為了改善在執行特定演算法時的系統效能而決定。因此，在某些分工合作的方法是為了在執行特定的應用程式時有最大的整體加速（speedup）而設計[1,4,6]。加速的估算都是分析應用程式行為的基本資料集而來。由於這些資料集的相關性，在某些應用程式領域整體中。例如：藉由管線化多功能的分工合作來改進硬體的使用率[7]，加速並不是較好的評估標準。

有限制的分工合作通常使用各個硬體或軟體部份的大小容量的限制，來產生功能性的實體部份，使得每一個部份都能在單一個元件中製作。容量限制的分工合作通常都使用在系統雛型應用程式上，如將應用程式對映到多個 FPGA 元件的分工合作方法[8]。

效能限制的分工合作方法不是集中在整體時間限制、如整體延遲時間（latency），就是在滿足運算間的時間限制。在這些清形下，分工合作的目標是藉由將系統功能以軟體來完成而縮減系統費用，因此在減少的特定硬體的製作的同時，必須滿足系統效能的需求。

（三）分工合作的策略：在一個特定的目標架構下，系統層功能描述旳分工合作的結果是標記著硬體或軟體運算的工作（task）。最近提出以數學模式解決分工合作問題的有整數規劃（integer programming, IP）與整數線性規劃（integer linear programming, ILP）兩種方法[2,9]。在分工合作方法中有一個共同的誤解是認為使用 CAD 自動化工具就足以解決分工合作的問題。通常決定某特定功能需使用硬體或軟體來製作是取決於一些沒有出現在系統規格描述中的抽象層次之資料。在缺乏模塑抽象層次資料之能力的情形下，並不能使用自動化工具產生出系統的分工合作。因此在捕捉系統功能的模塑方法與產生分工合作的抽象層次間有著不可分的關係。因此在某種假定的目標架

構下，我們需考慮使用經驗法則（heuristic）方法與問題分離(decomposition)策略來處理分工合作問題的複雜度。

就整體來說，電子系統的硬體－軟體的同步設計是一個廣泛的研究範圍，因為其具有各種不同的應用範疇、設計風格或製作技術等關係。欲改進電子系統的設計能力及時效，需要各種不同硬體－軟體的同步設計的方法論及工具。

三、 研究方法

在我們的分工合作的演算法中[10]，首先，我們將每個物件的成本／效能比值與系統的限制輸入。將所有物件依照成本／效能比值排序。接著，將前半部物件以軟體方式製作，後半部物件以硬體方式製作，評估此分工合作的結果是否有符合系統限制。如果成本符合而效能沒有符合，則增加硬體物件；如果效能符合而成本沒有符合，則增加軟體物件；如果都沒有符合，則增加硬體物件，因為我們認為效能比較重要；而如果兩者都符合，則我們檢查此結構是否是最佳，如果是就是我們的答案，如果不是，則視成本與效能的關係，再增加軟體物件或硬體物件。如此重複一直到找到最佳結構或是找不到符合成本／效能的結構。詳細演算法如圖１。

在我們的共同評估的環境中，亦有一個演算法可以來驗證此分工合作的最後結果是否真正符合系統在時序方面的要求[11]。此方法如圖２。

四、 結論

我們提出的硬體－軟體分工合作的架構，此架構包括硬體－軟體間分工合作的演算法及硬體／軟體的共同評估。我們可以先用物件導向方法來描述系統，再以我們的硬體－軟體分工合作的架構，來設計出此系統的結構。本年度之計畫主要成果已有兩篇論文發表在今年之 International Conference On Parallel and Distributed Processing Techniques and Applications (PDPTA'99) 上。

五、 參考文獻

[1] R. Ernst, J. Henkel, and T. Benner, "Hardware-software co-synthesis for micro-controllers," IEEE Design and Test of Computers, pp. 64-75, Dec. 1993.

[2] G. De Micheli and R. Gupta, "Haredware/Software Cosynthesis for Digital Systems," IEEE Design and Test of Computers, pp. 29-41, Sept. 1993.

[3] R. Gupta, C. Coelho, and G. De Micheli, "Program implementation schemes for hardware-software systems," IEEE Computer, pp. 48-55, Jan. 1994.

[4] A. Jantasch, P. Ellervee, J. Oberg, A. Henani, and H. Tenhunen, "Hardware/software partitioning and minimizing memory interface traffic," in Proc. EURODAC, 1994, pp.226-231.

[5] P. Chou, E. Walkup, and G. Borriello, "Scheduling strategies in the co-synthesis of reactive real-time systems," IEEE Micro, vol. 14, no. 4, pp. 37-47, Aug. 1994.

[6] D. Thomas, J, Adams, and H. Schmitt,"A model and methodology for hardware-software co-design," IEEE Design and Test, vol. 10, no. 3, pp. 6-15, Sept. 1993.

[7] N. Binh, M. Imai, A. Shiomi, and N. Hickichi, "A HW/SW partioning algorithm for designing pipelined ASIP's with least gate counts," in Proc. DAC, 1996, pp. 527-532.

[8] J. Cong and Y. Ding, "Combinational logic synthesis for LUT based field programmable gate arrays," TODAES, ACM trans. Design Automat. Electron. Syst., vol. 1, no. 2, pp. 145-204, Apr. 1996.

[9] R. Niemann and P. Marwedel, "Hardware/software partitioning using integer programming," in Proc. EDTC, 1996, pp. 473-479.

[10] T.-Y. Lee, P.-A. Hsuing, and S.-J. Chen, "A Case Study in Hardware-Software Codesign of Distributed Systems–Vehicle Parking Management System," in Proc. PDPTA, vol. 6, June, 1999, pp. 2982-2987.

[11] J.-M. Fu and S.-J. Chen, "Hardware-Software Coverification of Distributed Embedded Systems," in Proc. PDPTA, vol.

6, June, 1999, pp. 2995-3001.

Partitioning Algorithm

1 Hw/sw_partition(N_1, N_2, N_3, ..., N_i, ..., N_m) /* N_1, N_2, N_3, ..., N_i, ..., N_m where N_i is an object */
2 Input cost-performance ratio of each object and system constaints
3 Sort all objects according to their cost-performance ratios
4 Let $k = 1$, $j = m$ /* where k is the lower bound object index, j is the upper bound object index */
5 Let $i = \lceil \frac{m}{2} \rceil$
6 Use software to implement objects from N_i to N_j and use hardware to implement objects from N_k to N_{i-1}
7 Check if partitioning result satisfies system constraints
8 Case 1: Both cost and performance satisfies
9 If the partition result is an optimal solution
10 Return partition result
11 else If performance is more important
12 $k = i$
13 $i = i + \lceil \frac{j-i}{2} \rceil$
14 else $j = i$ /* cost is more important */
15 $i = i - \lceil \frac{i-k}{2} \rceil$
16 end
17 end
18 Go to step 6
19 Case 2: Cost satisfies but performance don't satisfies
20 $k = i$
21 $i = i + \lceil \frac{j-i}{2} \rceil$
22 Go to step 6
23 Case 3: Cost doesn't satisfies but performance satisfies
24 $j = i$
25 $i = i - \lceil \frac{i-k}{2} \rceil$
26 Go to step 6
27 Case 4: Both cost and performance don't satisfies
28 Print " No solution"

圖１：分工合作的演算法

```
CONVERT(task_graphs, processor_graph, table_computation_time)
    CONVERT_PROCESSOR_ELEMENT(processor_graph, table_computation_time);
    CONVERT_TASK_GRAPH(task_graphs);
    CONVERT_INTER_TASK_COMMUNICATION(task_graphs);
```

圖２：驗證的演算法