以 SystemC 為基礎應用於動態可重組式晶片系統的效能評估框架

劉智文、廖志峰、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

可重組技術（Reconfigurable Technologies，以下簡稱 RCT）可在系統運作時動態變更硬體電路，使得單一硬體晶片所提供的功能更具有彈性，將原本需要花費大量微處理器運算時間的工作轉移至硬體(例如：FPGA)來加速執行。

然而 RCT 在業界還不是很普及，大致有以下三個原因：首先，使用 FPGA 的系統效能及功耗仍不敵專門針對某種應用而特別設計的 IC，但這可隨著「摩爾定律」而被解決；第二，工具鏈(toolchain)及設計流程不完整；第三，很少有同時具備軟體及硬體背景知識的工程師。第二及第三個問題可隨著有效的發展工具而一併被解決，工程師可以利用完整的工具鏈，並依照設計流程而做出一個效能不錯的系統。

目前已經有許多研究團體投注很多心力在 RCT，但是他們大部分都專注在軟硬體切割及排程，卻很少有工具可以整合這些技術，而且用傳統的工具很難評估含有這種技術的系統效能，因此我們提出了一個很容易使用的系統層次效能評估框架，用來偵測系統效能瓶頸發生的地方，並可以藉由產生的效能評估報告來選擇適合的系統設計方案。

二、系統模型

本文所提出的架構是基於 SystemC，這是一個系統層次的塑模語言，不但可以支援軟體及硬體的模型，也可以做模擬以便觀察系統的運作是否正確。讀者若想要進一步了解 SystemC，可以參考 SystemC 的官方網站 http://www.systemc.org。

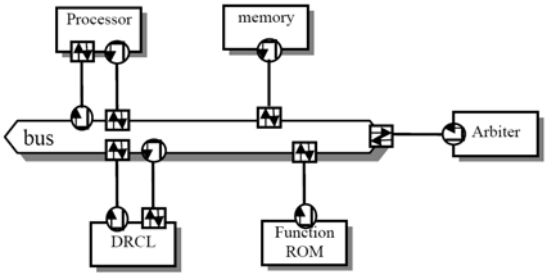

圖 1. DRSoC 架構模式

如圖 1 所示，dynamically reconfigurable system-on-chip（DRSoC）主要由處理器（processor）、記憶體（memory）、唯讀記憶體（ROM）、匯流排（bus）、匯流排仲裁器（arbiter）、及動態可重組邏輯(dynamic reconfigurable logic, 以下簡稱 DRCL)所組成。我們在此採用了一個“簡單”的匯流排模式，所謂“簡單”指的是此匯流排模式沒有管線（pipeline），沒有分割交易（split transaction），沒有要求/回應機制（request/response mechanism）。其中唯讀記憶體用來放置將要下載到 DRCL 上執行的位元流（bitstream）。以下將逐一介紹其它元件的模型。

甲、處理器模型

處理器必須執行 DRSoC 上的軟體，除了控制周邊設備之外，它主要有兩個功能，分別以圖 2(a)與 2(b)表示。一個功能是使用匯流排來存取資料。另一個功能是下命令給 DRSoC；例如下重新組態或執行的命令去執行硬體指令。

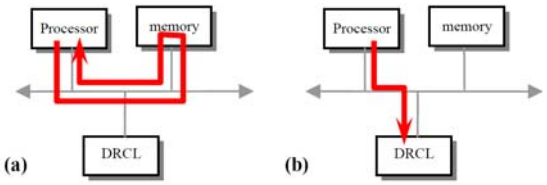

圖 2. 處理器行為:(a)存取記憶體，(b)發出命令

乙、匯流排仲裁者模型

由於系統中有超過一個以上的主要元件，例如處理器及 DRCL，當很多元件同時提出使用匯流排的請求（request）時，匯流排仲裁者會裁定哪個元件可以優先使用匯流排，其他已經提出的請求則被放置在等待序列（waiting queue）中。仲裁者是根據以下的規則來選擇使用匯流排的請求：

1. 優先選擇 locked burst 的請求。
2. 如果有元件已經提出兩次請求，若它上一次請求的 lock 旗標值被設定，則仲裁者會允許提出此請求的元件使用匯流排。
3. 選擇等待序列中具有最高優先權的請求。

丙、DRCL 模型

Dynamically reconfigurable logic (DRCL) 模型主要有三種行為。第一種是存取記憶體。如圖 3（a）所示，當 DRCL 從處理器收到一個執行的命令時，DRCL 會根據收到的位址參數去存取記憶體，之後 DRCL 會從唯讀記憶體中取出位元流，如圖 3（b）所示。當 DRCL 完成工作時，會發出一個回應的命令通知處理器，以圖 3（c）表示。

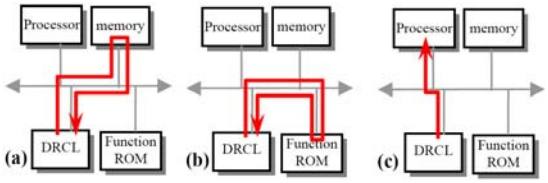

圖 3. DRCL 行為:(a)存取記憶體，(b)取得位元流，(c) 發出回應通知處理器

Reconfigurable logic（RCL）可以被分為兩種組態形式，分別為完全組態(靜態)及部分組態（動態）。傳統的可重組式系統為完全組態形式，這意味著系統必須在開始執行前就完成組態並且保持不變，直到應用程式執行完。在這種系統中，組態時整個系統必須停止執行工作直到組態完畢。部分組態形式的系統則允許在系統執行期間做部分重新組態，因此動態可重組式系統可以重新組態一個電路到晶片上的某一部分，而不影響其它正在執行的電路；我們則是採用動態可重組式系統的架構。

在這個效能評估框架中，組態的基本單位稱為 slice。DRCL 會根據以下規則挑選一個區塊作組態設定。

1. 如果 DRCL 中有一個區塊已經有組態好的相同電路，而且目前已經處於完成狀態（done status），則我們就在這個區塊執行。
2. 如果 DRCL 中存在一個區塊有相同數目的 slice 並處於完成狀態，但此區塊執行的功能與我們要的功能不相同，則我們選擇在此區塊作組態。
3. 如果 DRCL 存在未組態區塊（即此區塊處於閒置狀態）且此區塊有足夠的 slice 數目，則我們組態此區塊。
4. 如果 DRCL 中剩餘的 slice 數不夠，則我們會釋放處於完成狀態之區塊轉換成閒置狀態，並再次檢查規則 1 與 3。

以下舉一個例子來說明。如圖 4(a)所示，DRCL被分成 6 個slices S_0到S_5，假設目前已經有 3 個電路C_1、C_2、C_3在DRCL中，且電路C_1及電路C_3處於完成狀態，電路C_2處於執行狀態，如果下一個要執行的電路也是C_2，則根據規則 2 會選擇S_0做組態並在此執行。若接著又有一個要執行的電路C_4且此電路需要 3 個slices，則根據規則 4 及規則 2，slice S_2、S_3會先被釋放成閒置狀態，再將C_4組態到S_2至S_4，如圖 4（b）所示。

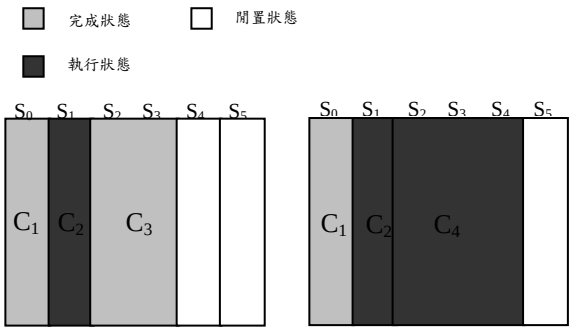

圖 4. 放置圖:(a)放置電路C4之前，(b)放置C4之後

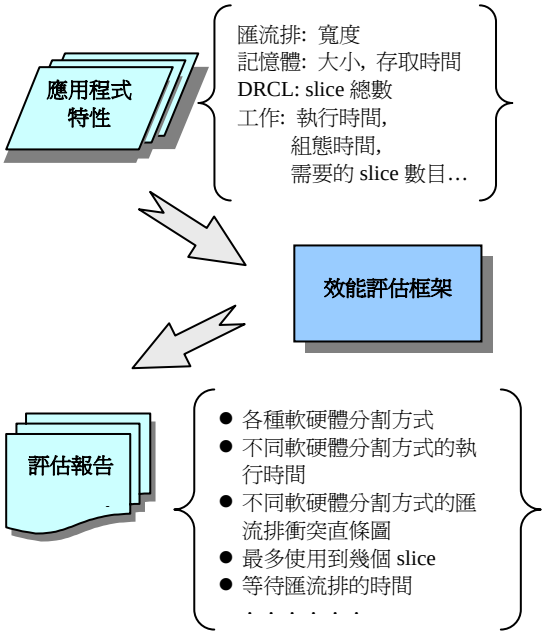

圖 5. 效能評估框架流程圖

三、效能評估框架

藉由以下設定，我們可以在效能評估框架中對一個應用程式做分析：

1. 匯流排寬度：這會影響記憶體存取的次數，寬度的基本單位為 word，一個 word 為 4 個位元組。
2. 記憶體大小：記憶體大小為 4 的倍數，因為最小的記憶體單位為 4 個位元組。
3. 記憶體存取時間：存取記憶體所需花費的時間。
4. 工作執行時間、組態花費的時間、在 FPGA 上所需的面積大小（以 slice 為單位）及功能行為程式碼（function behavior code）：在這裡，我們假設每個工作都是執行一個功能。工作執行時間指的只是計算時間，不包含記憶體存取時間。組態時間則是工作被下載到 DRCL 所花費的時間。slice 數目則是一個工作所需的 RCL 面積。功能行為程式碼則是一段抽象的 SystemC 程式碼，用來描述功能的行為。
5. 排程器：框架中預設的排程器演算法為先進先出演算法，使用者可以使用自訂的排程器來最佳化他們的應用程式。

做完模擬之後，效能評估框架會產生評估報告，報告的內容留至第四部份詳述。

圖 5 展示了整個效能評估框架的流程。

四、實例

應用程式是由一組協力完成的工作所組成，而這些工作之間有一定的順序關係，意即某些工作必須等待其他工作完成才可以開始執行。以下這個例子中，共有六個工作，分別執行四種不同功能，以表 1 列舉之。表 1 呈現了工作與功能間之對應關係，並描述工作 3 必須等待工作 5 完成後方可開始執行。

表 2 列出使用者所給定的各個功能之相對應屬性，例如以軟體執行功能 3 時，需要 1300 ns 的執行時間；若以硬體執行，則需要 150 ns 的組態時間，600 ns 的執行時間，以及使用 2 個 slices。

表 1. 各項工作的資訊

\ Task#	1	2	3	4	5	6
Func#	1	3	2	2	4	3
Wait#	0	0	5	0	0	0

*Wait#: 0 - 不須等待其它工作

- 須等待工作#完成後才可以開始執行

表 2. 各個功能的資訊

Func#	CT (ns)	S_ET (ns)	H_ET (ns)	Slice
1	0	200	0	0
2	100	1000	500	1
3	150	1300	600	2
4	200	2000	1000	1

* CT: 組態時間　　* Slice: 需要用到的 slice 數目

* S_ET: 以軟體執行花費的時間

表 3. 軟硬體分割方式

Run# \ Func#	1	2	3	4
0	0	1	1	1
1	0	1	1	0
2	0	1	0	1
3	0	1	0	0
4	0	0	1	1
5	0	0	1	0
6	0	0	0	1
7	0	0	0	0

這個框架會產生所有可能的軟硬體分割方式， 如表 3 所示。S 此系統之規格如下：記憶體存取時間為 10 ns、匯流排寬度為 2 words、最大的 DRCL slices 數為 5。根據以上的設定，效能評估框架會產出一些效能評估報告，第一份報告列於表 4，其中列出了每個工作的執行時間及所需的 slice 數，以軟硬體分割方式 P0 來看，此種分割方式所需花費的執行時間為 2033 ns，DRCL 的平均使用率為 54.63%，DRCL 的 slice 最大使用數為 3。工作 T4 所花費的執行時間為 752 ns，其中包含了執行時間（500 ns）、組態時間（100 ns）、記憶體存取時間（90 ns）、以及匯流排等待時間（62 ns），由此我們可以得到以下結論：軟硬體分割方式 P0 需要最少的執行時間，而分割方式 P7 需要最多的執行時間。分割方式 P1 及 P2 所花費的執行時間很相近，

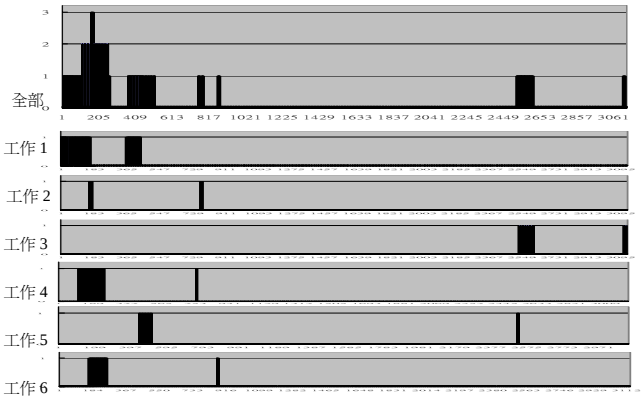

圖 6. 軟硬體分割方式 P1 的匯流排衝突.

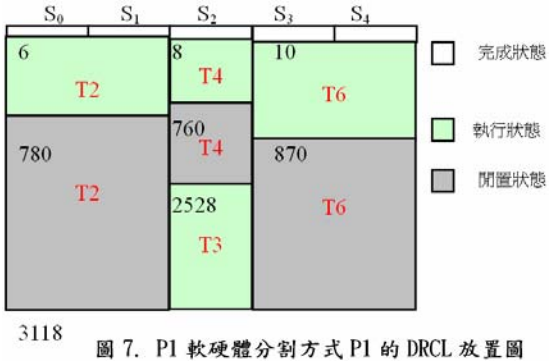

圖 7. P1 軟硬體分割方式 P1 的 DRCL 放置圖

然而 P2 不需要如 P1 一般使用如此多的 DRCL slice 數。如果我們選擇 P2 這種分割方式，對於此應用程式我們便可以使用較少的 DRCL slice 數。

圖 6 展示當分割方式為 P1 時，匯流排的使用情形及各個工作使用匯流排的情況，參考這份報告，我們可以清楚地看出系統瓶頸出現在什麼地方。在匯流排使用圖中（圖 6 上方），我們觀察到同時間最大的匯流排請求數為 3 次，透過個別的工作匯流排使用情況（圖 6 下方），我們可以知道是那些工作在競爭匯流排的使用權，而導致瓶頸發生。

另一份報告顯示於圖 7，此為分割方式 P1 在 DRCL 的工作放置圖，圖上的數字表示區塊被放置的時間。

五、結論

我們發展了一個針對動態可重組的 SoC 系

統並且以 SystemC 為基礎的效能評估框架，如同我們在文章中所舉的例子，我們可以利用此框架做設計空間的探索。

表 4. 結果列表

P#	T_ET (ns)	AUR (%)	Max_S (U/M)	工作總執行時間 = 執行時間 + 組態時間 + 記憶體存取時間 + 等待匯流排的時間 (ns)																	
				T1			T2			T3			T4			T5			T6		
0	2033	54.63	3 / 5	0	240	0	150	20	3	100	90	0	100	90	62	200	60	40	0	20	0
				440			773			690			752			1325			620		
1	3119	29.54	5 / 5	0	240	0	150	20	4	0	90	0	100	90	61	0	60	0	150	20	89
				440			774			590			751			2085			859		
2	3083	17.16	2 / 5	0	240	0	0	20	0	0	90	0	100	90	51	200	60	29	0	20	0
				440			1320			590			741			2085			1320		
3	5169	5.19	1 / 5	0	240	0	0	20	0	0	90	9	100	90	52	0	60	0	0	20	0
				440			1320			599			742			2085			1320		
4	2623	33.45	5 / 5	0	240	0	150	20	2	0	90	0	0	90	0	200	60	0	150	20	9
				440			772			1090			1090			1285			779		
5	4709	13.20	4 / 5	0	240	0	150	20	3	0	90	0	0	90	0	0	60	0	150	20	11
				440			773			1090			1090			2085			781		
6	5265	4.88	1 / 5	0	240	0	0	20	0	0	90	0	0	90	0	200	60	0	0	20	0
				440			1320			1090			1090			1285			1320		
7	7351	0.00	0 / 5	0	240	0	0	20	0	0	90	0	0	90	0	0	60	0	0	20	0
				440			1320			1090			1090			2085			1320		

*T_ET: 總執行時間 *AUR: 平均 DRCL 使用率 *M_S(UM): 最多用到幾個 slice