TeraSoft 鈦思科技
www.terasoft.com.tw

嵌入式系統工程師如何藉由建模來幫助晶片系統應用程式的開發

By Mark Corless and Eric Cigan, MathWorks

您將從本文了解到建模(modeling)如何幫助演算法及嵌入式軟體工程師的團隊來設計馬達控制演算法，並將其實現到可程式晶片系統(SOC, system-on-chip)上。假設我們就是這個團隊的嵌入式工程師，我們將為您展示建立模型如何幫助我們劃分設計、透過實現資源來平衡晶片的功能行為，及在實驗室進行測試。

可程式 SoCs 像是 Xilinx® Zynq® SoCs 以及 Altera® SoC FPGAs，是在同一片晶片上結合了可程式邏輯及微處理器核心，提供測試團隊把演算法轉檔佈署到範圍廣泛的應用的新平台，包含嵌入式視覺、通訊、馬達與電力電子的控制。這些設計團隊通常包含兩大類的工程師：演算法工程師，負責概念上的開發及以數學或法則為基礎的演算法的精進；而嵌入式工程師則負責將演算法再精煉，並在嵌入式裝置的軟體或硬體上實現。

演算法工程師通常在開發過程的早期會使用建模的方式，來確定對該演算法具備應用程式功能性正確(functionally correct)的信心。另一方面，嵌入式工程師不會一直感受到建模帶來的好處。不過，當這些團隊合作不密切時，就有可能導致遲來的錯誤偵測、計劃延遲、資源使用過度、或者因為設計及測試次數的不足而必須進行功能設計的妥協。

現在我們來看看模型建立是否真能幫助演算法工程師及嵌入式工程師產生一個更有效率且更具協調性的設計流程。我們想要把重點放在可藉由模擬來探索的建模演算法之元件。我們利用模擬的方式來幫助設計劃分決策，使用能將產生的程式碼及手動編碼的自動整合及轉檔佈署的方式，讓實驗室使用時間更有效率。

建議的工作流程

我們提出一個混合了從模型產生程式碼以及手寫程式碼的建議工作流程。(在這一整篇文章，我們將手寫程式碼的部分的設計稱為*參考設計*。)我們先從演算法開發人員提供的模型開始，並透過增加實現的細節來反覆地精進這個模型。在每一次的疊代當中，我們會模擬系統行為來確保演算法模型的功能性正確，並透過自動產生程式碼去實現該演算法，讓程式碼表現如同模型一樣，接著自動地與我們的參考設計整合，以確定建立一個能在硬體上實現的可重覆過程(圖 1)。

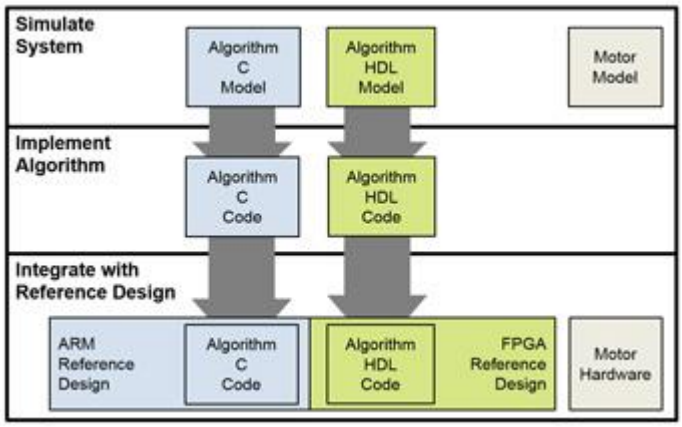

圖 1：開發及轉檔佈署馬達控制演算法至晶片系統工作流程

選擇一個硬體平台

在本案例分析，我們決定使用一個磁場定向控制(FOC, field-oriented control)演算法來設計一個永磁同步馬達的速度控制器，接著轉檔佈署到 Zynq-7000 All Programmable SoC Intelligent Drives Kit II (圖 2)。我們選擇馬達控制，因為它是一個常常需要演算法工程師與嵌入式工程師共同合作的應用。我們選擇 Zynq Intelligent Drives Kit II，因為它現成可以利用、並且提供我們需要的 I/O 支援。

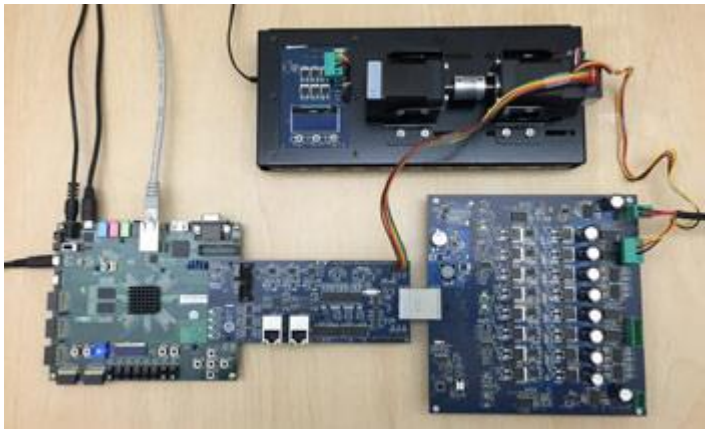

圖 2：帶有選擇性測力計系統的 Zynq Intelligent Drives Kit II (圖來自 Avnet Electronics 行銷部)。

Zynq Intelligent Drives Kit II 是一個開發平台，可提供工程師在 Zynq Z-7020 Soc 裝置上去測試馬達控制演算法。以 ZeaBoard 開發板為基礎，這個套件包含了一個 Analog Devices 公司的 FMC 馬達控制模組以及一個配有一個 1,250 cycles/revolution 編碼器的 24V 無刷直流馬達。由於我們想在一個操作條件的範圍下測試馬達控制演算法，我們使用這個帶有選擇性測力計系統的 Zynq Intelligent Drives Kit II。

演算法元件的設計劃分(Partitioning)

在選擇了硬體平台之後，我們重新檢視演算法工程師提供的初始系統模擬模型，並定義其他一些在轉檔佈署到 SoC 時所需要的演算法單元。這個模型包含了一個以資料表參數為基礎的馬達控制器演算法，而這個演算法由一個外部速度控制迴圈使用 FOC 來調節一個內部電流控制迴圈。

這個模型雖然涵蓋了控制器的核心數學運算，卻未考慮周邊設備(例如 ADC、編碼器、PWM)或其他操作模式需要的演算法單元(停用、開放迴圈、編碼器校正)帶來的影響。我們與演算法工程師一起合作找出哪些演算法單元以建立模型，並決定是否在 ARM 執行這些單元或者是在 SoC 執行可程式邏輯 (圖 3)。

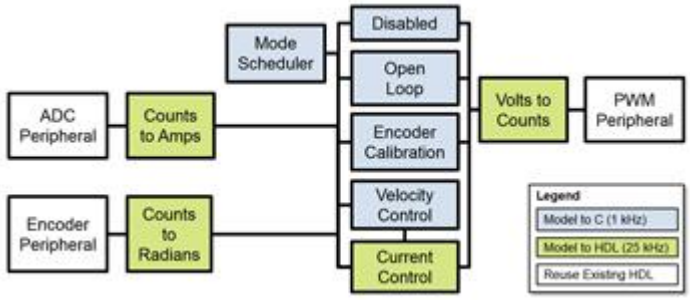

圖 3：演算法元件的劃分。

我們放入新的演算法元件讓初始系統模型更精進(圖 4)。為了確保系統的模擬，我們建立現有與馬達模型相互互動的外部設備的塊狀參數模型。舉例來說，我們已現有一些外部編碼器的 HDL 程式碼，並計劃重複使用進行轉檔佈署設計。這個外部編碼器讀取 50 MHz 的數位脈衝流，並將其譯為控制器演算法讀取的 25kHz 計數訊號。如果我們直接建立這個脈衝流模型，我們將在這個系統模型引入 50MHz 的力度，這會大量增加模擬時間，因此另外的作法是，我們建立了一個編碼器的塊狀參數模型，能將轉子的理想位置從馬達模型轉換成計數訊號並以演算法元件形式呈現。這種精度的建模能幫助我們在保持合理的模擬時間時，既能夠模擬測試編碼器校正(Encoder Calibration)元件時需要的啟動條件，也能納入位置量化效果來測試速度控制(Velocity Control)單元(圖 5)。

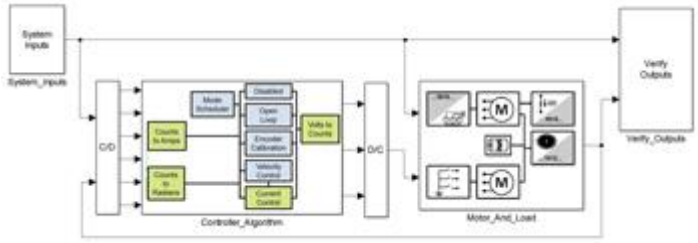

圖 4：系統模擬模型

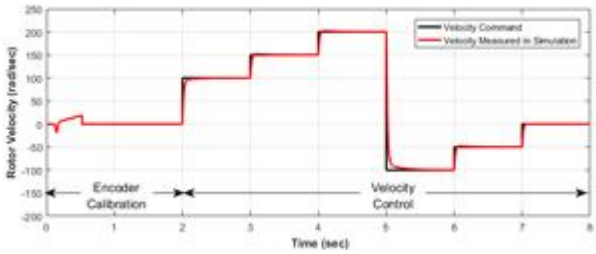

圖 5：編碼器校正及步進速度指令的系統模擬結果

如果只需要幾 kHz 或更低速的訊號，我們選擇在 ARM 上執行演算法元件。以幾 kHz 的速度做為限制是因為我們計畫在 ARM 執行一個 Linux® 操作系統，而需要更高速率的演算法單元會在 FPGA 上執行。我們希望盡可能在 ARM 執行演算法單元，因為我們發現在 ARM 上設計疊代比在 FPGA 更快，以 ARM 核心作為演算法目標更簡單因為它支援原生的浮點數運算。大部分 FPGA 執行浮點數的效率不佳，所以若將可程式邏輯晶片作為目標實現硬體的話，則需要額外的步驟將演算法轉換為定點數。除此之外，我們也發現編譯 C 程式碼到 ARM 上，基本上會比 FPGA HDL 程式碼編譯流程快些。

我們使用模擬來決定演算法元件是否能以夠低的速度在 ARM 執行，或者需要使用 FPGA。舉例來說，演算法工程師提出一個以 25 kHz 執行的編碼器校正慣例程序，則該程序必須要在 FPGA 上執行。我們也使用模擬來測試我們是否可以 1 kHz 執行編碼器校正元件，結果是可行的，所以我們決定在 ARM 執行該元件。

功能行為與執行資源的平衡

當我們已具備有期望的元件速率、功能性正確的模型，我們把所有元件組合，準備進行 C 演算法模型的 C 程式碼產生，以及為 HDL 演算法模型的 HDL 程式碼產生(圖 6)。我們接著在模型中透過互動的環境加入實現所需的細節，當我們覺得模型可以達到令人滿意的記憶量並可以元件的速率執行時，就開始進行程式碼產生的工作。

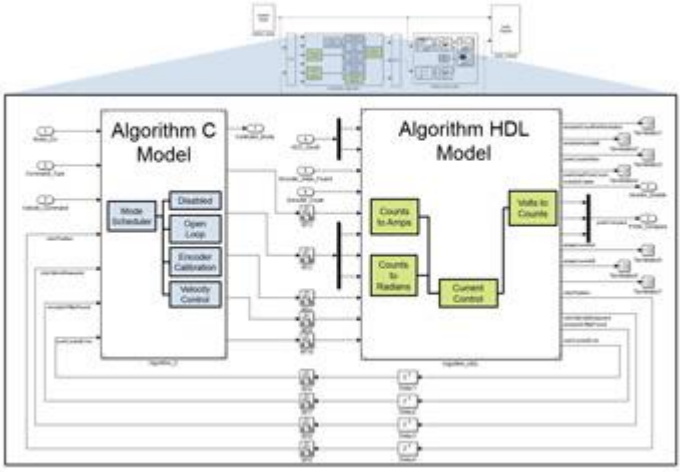

圖 6：控制器演算法模型產生 C 及 HDL 程式碼

我們使用 Embedded Coder® 從 C 演算法模型產生 C 程式碼，並產生一份總結調用介面及預估資料記憶體用量的報告。檢閱這份報告時，我們發現所有的資料類型均為雙精度的浮點數(double-precision floating point)。我們想要讓這些資料能以整數、定點、或其他單精度浮點的數學形式串接到 FPGA。我們在模型應用這些資料形式，並利用模擬來驗證此行為是可以被接受的，接著再次產生改善的程式碼，在這個階段我們確信程式碼是適合在 ARM 上執行的。

我們以定點數方式來實現 HDL 演算法模型，因為定點運算在 FPGA 上耗費較少的資源。為了完成這個步驟，我們與演算法工程師合作，一起鑑定及限定設計的主要訊號範圍(電流、電壓、及速度)，接下來使用 Fixed-Point Designer™來定義可確保計算不會滿溢的定點資料類型。我們使用 HDL Coder™來產生程式碼及一份簡要的報告。

我們檢視報告中資源估計的部分以指出看起來比期待中更大的數學運算。比方說，我們當初在字長的選擇設定中，導致好幾個由兩個 34 位數字的相乘數，這些我們認為是沒有必要耗用到 FPGA 資源等的問題，我們可以從資源利用報告中辨識出這樣的問題，我們可以回去降低模型的精確性後，再使用模擬來驗證功能仍是正確的，然後再產生改善後的程式碼。接著我們使用 Xilinx Vivado® Design Suite 來合成程式碼並驗證其能符合時域的要求。

進行實驗室測試

當我們擁有演算法實現的設計後，我們已準備好將其與我們的參考設計結合。我們開始由手動的方式來整合所產生的 C 函式以及手動編碼的 ARM 嵌入式專案，並將產生的 HDL 程式碼實體與我們手寫程式碼的 Vivado 專案整合。然而，我們發現如果我們一直手動地執行這些整合，我們將需要在實驗室中涉入每一個設計疊代，而我們介紹此新工作流程的其中一個目標，就是為了讓演算法工程師可以在實驗室自動地整合並且轉檔佈署這些過程。

因此我們使用 HDL Coder Support Package，在 Xilinx Zynq-7000 平台上將我們手動編碼的 Vivado 專案登錄為參考設計，接下來我們就可以將已產生的演算法 HDL 程式碼自動地與我們手寫程式碼整合，建立一個位元流並下載到 FPGA。我們在 Embedded Coder Support Package 中選擇針對 Xilinx Zynq-7000 平台，將產生的演算法 C 程式碼與 Linux 操作系統整合，建立一個可執行的程式碼，並將其下載到 ARM，隨後透過 Simulink®與其交流。上述所提及的 Embedded Coder Support Package 的支援套件提供了 AXI 交互連結，有助於演算法元件在 ARM 核心與可程式邏輯之間進行溝通。在初始的系統設定階段，很需要演算法工程師及嵌入式工程師在實驗室合作。

作為嵌入式工程師，我們需要設置轉檔佈署結構，並與演算法工程師合力驗證基礎功能，當系統設定完成，演算法工程師可以獨立地使用 Simulink 進行設計疊代作為 SoC 的初始介面。

而演算法工程師則測試轉檔佈署的控制器，並確定該控制器沒有送出預期的回應。模擬與硬體結果的比較顯示我們並未正確地計算 ADC 到電流的匹配，因此演算法工程師再進行另外的測試以找出更佳的馬達扭力常數，並改善模擬及硬體結果之間的相關性(圖 7)。

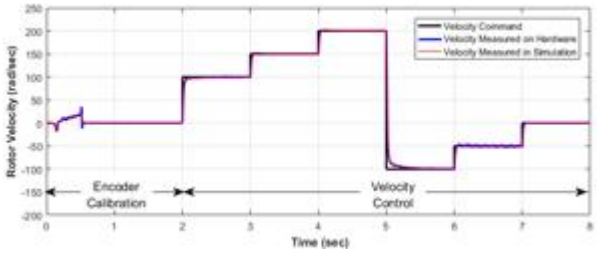

圖 7：模擬及硬體結果之間的比較。

模擬及硬體測試結果所得到的高度相關，讓我們對於做出模型等級設計的決策更具信心，且可進一步節省實驗室時間。舉例來說，馬達於實驗試運轉，但在特定情況下變得不受控制。我們推斷這個問題跟一個執行在 FPGA 的定點速度計算的滿溢有關。我們在模擬實複製這個問題，並辨識到一開始關於馬達速度最大執假設的缺陷，我們可以在模擬中除錯並解決這個問題，而且只在驗證所做的改變時使用到實驗室的時間。

此方法的好處

這裡描述的工作流程可以增進我們與演算法工程師合作的效率。我們透過模擬評估演算法劃分在系統表現上的效果，並驗證編碼器校正元件可以從較高速率的可程式邏輯區域移至較低速率的 ARM 區域。

模擬也讓我們可以既能維持功能性行為所需又能節省實現時的資源，像是縮短可程式邏輯數學運算的字長，或者將要通過 AXI 互相連接的浮點資料轉換為定點資料。最後，我們在實驗室的原型測試，幫助我們辨識 ADC 數到電流的標定錯誤，也讓我們的演算法工程師可以執行更進一步的測試來描繪馬達的扭力常數。

整體來說，這個工作流程助於我們與演算法工程師之間的緊密合作，在更精簡地使用實驗室的情況下創造一個更有效率的實現。