SoC 軟硬體協同設計方法與實例

劉岱穎 林義能 林盈達

摘要

SoC (System on Chip) 有著面積小、速度快、耗電低等優點，使其越來越受到矚目。為了加速SoC的設計流程，相關技術包括軟硬體協同設計、IP的可重用性、SoC驗證技術、低功率消耗、嵌入式軟體移植與開發等也隨之興起。其中軟硬體協同設計讓軟硬體從設計到驗證的部分都能緊密結合，減少了軟硬體整合所花費的時間。

軟硬體協同設計首先需訂定系統規格，對此作系統分析並把某些功能分別以軟硬體實現。至於何種功能需用軟體或硬體實現則必須考慮系統的效率與成本；因為硬體處理速度較快但是成本較高，通常將常用且便宜的部份做成硬體。確定軟硬體分割後即可對軟硬體部分開始實作。就硬體來說可以使用VHDL、Verilog等硬體描述語言撰寫其規格，軟體部分則使用C語言等來驅動這個硬體。之後進入軟硬體協同模擬階段，亦即當要從硬體抓取資料時以軟體模擬並確認結果是否正確。如果此一階段沒問題，便可進入軟硬體協同驗證的階段，將這些資料放到模擬板上執行，驗證其正確性。我們使用Xilinx Vertex-II Pro ML310這塊模擬版來實作一個加法器連接至OPB上，利用軟體使用此加法器使其產生從1加到10後之結果，並以軟硬體協同設計的流程配合解釋實作的過程。

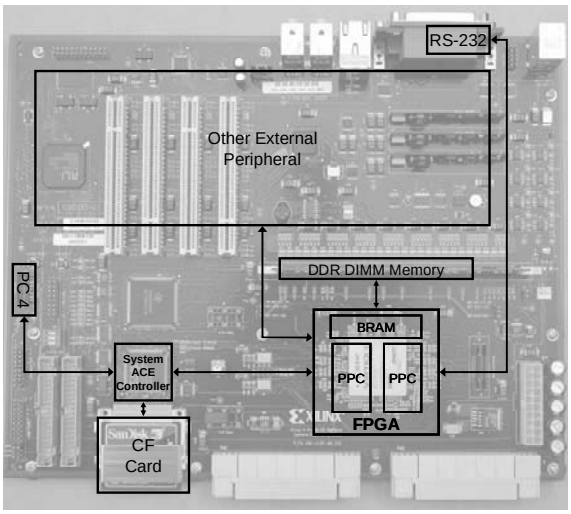
在軟硬體協同設計上可能會遭遇到軟硬體無法互相溝通的情況，常見的原因在於實作硬體時的規格設計有問題，或是沒有依照標準的匯流排協定的格式來設定，導致資料無法正確讀取與寫入等。

1. 軟硬體協同設計之重要性

隨著半導體工業技術的快速演進，現今市場對電子產品的需求傾向於輕薄短小，SoC的開發也日顯重要。因而帶動軟硬體協同設計。軟硬體協同設計包含了軟硬體協同模擬、與協同驗證等，不僅減少硬體設計的風險、加速硬體的開發時間，更可在協同驗證環境中發現軟硬體的各項優缺點，避免過去在最後整合階段時才重新調整，造成開發時間的浪費。

硬體平台簡介

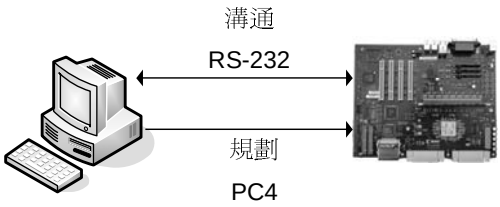
Xilinx Virtex-II Pro ML310[1](見圖一)擁有兩個PowerPC的處理器，30816個Logic Cells、2448kb的BRAM，板子中央銀白色的FPGA晶片，我們可以規劃此晶片以建立所需的硬體。晶片外部包含一些如DDR記憶體，網路卡等I/O裝置及其介面以便連接，可使用提供之IP驅動這些週邊並可利用PCI匯流排外接其他裝置。例如ML310本身內建許多高速I/O裝置如網路卡，IDE連接器等，便可利用Xilinx所發展的RocketIO技術使用這些高速裝置。



圖一：Xilinx Virtex-II Pro ML310

規劃FPGA的來源有二，一個是CF Card另一個是PC4[2]，兩者皆須經由System ACE Controller晶片處理規劃的資訊並規劃FPGA ML310預設會從CF Card中讀取規劃的資訊，並將我們所撰寫的軟體載入BRAM中執行，亦可將板子連接至一台主機將規劃的資訊與軟體下載到板子上執行。

電腦主機透過RS-232 Port連接板子(見圖二)，並使用終端機軟體相互溝通。利用板子上的Parallel Cable IV(PC4)連結到電腦主機的Parallel Port，由電腦主機透過PC4將FPGA規劃之資訊與軟體載入板子中。



圖二：主機與板子。

2. 軟硬體協同設計流程

軟硬體協同設計主要分為5個階段(見圖三)[3]：(1)訂定系統規格、(2)軟硬體分割、(3)軟硬體實作、(4)軟硬體協同模擬，以及(5)軟硬體協同驗證。

(1)訂定系統規格(System spec.)

確定系統的目的與可行性，接著訂出此系統需有哪些功能。

(2)軟硬體分割(H/W & S/W partition)

依照系統的功能，為了求取最大之效率，可將某些元件作成硬體以產生最快的速度與最多的產出。此時必須評估哪些元件須作成硬體，哪些元件須以軟體方式呈現，為降低整體成本與達到最高效率取得平衡。

(3)軟硬體實作

- 硬體實作：當我們把硬體規格訂出來後，利用HDL語言描述此硬體並根據板子的規格實作此硬體。

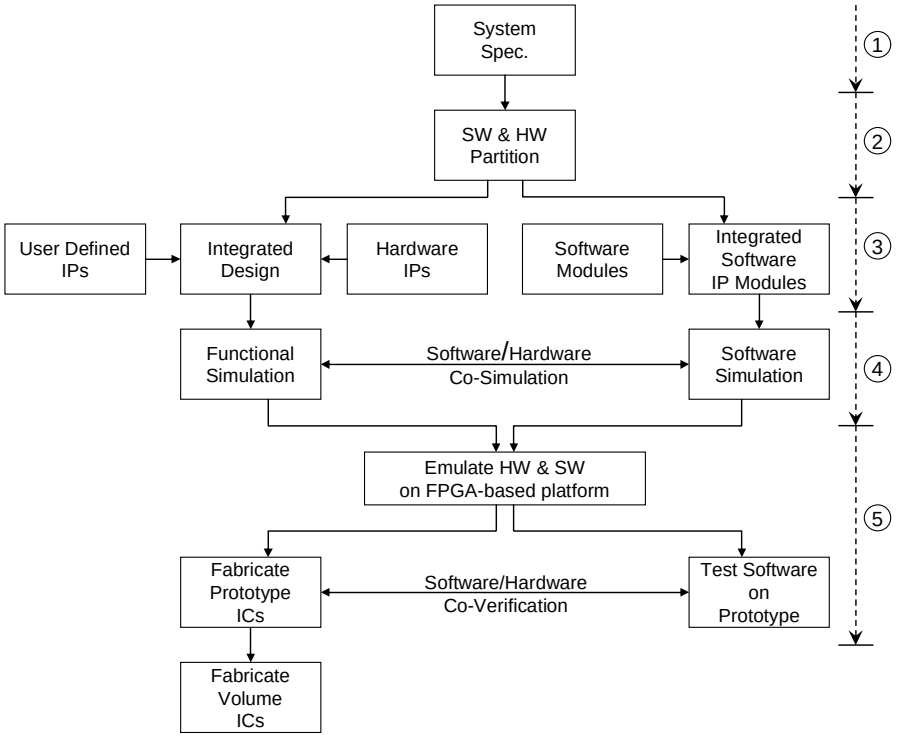
- 軟體實作：分別撰寫驅動程式與使用此硬體之軟體，整合系統產生之標頭檔與使用者之檔案。

(4)軟硬體協同模擬(Software/Hardware Co-Simulation)

在使用硬體的地方以軟體模擬硬體工作後得到的資訊，並以之作為輸入以驗證軟體的正確性。

(5)軟硬體協同驗證(Software/Hardware Co-Verification)

實際將軟硬體放到模擬板上驗證是否可以正確工作。



圖三：軟硬體協同設計流程。

3. 開發環境與實例分析[4]

在此我們循用軟硬體協同設計流程，分階段敘述如何設計一個系統使其計算從1加到10的數值並將結果顯示在終端機上。使用的工具及開發環境的設定也將一併介紹。

開發環境

我們在主機上(Windows XP)安裝Xilinx Platform Studio(XPS)[5]、Xilinx ISE[6]以及ModelSim來模擬HDL，前兩套軟體主要的功能是(1)建立嵌入式平台、(2)撰寫HDL碼，和(3)提供下載到FPGA執行。

Xilinx ISE提供一套整合式介面來撰寫HDL，可以對HDL做模擬，驗證其正確性，合成邏輯閘與繞線。

Xilinx Platform Studio(XPS)產生嵌入式平台，可以加入我們需要的IP並設定它，建立平台上的軟體與驅動程式，是一套整合系統的工具。

實例分析

1. **訂定系統規格**

我們設計一個系統，計算從1加到10的數值並顯示在終端機上。

2. **系統劃分，分成軟體跟硬體兩大部分**

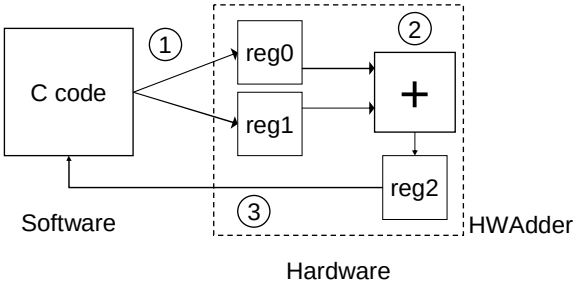
如圖四所示，我們將此系統中的加法器以硬體的方式實作，從1連加到10的部份以軟體實作。

我們建立一個新的裝置HWAdder，內含3個暫存器分別存放加數(reg0)、被加數(reg1)和結果(reg2)。計算過程分為以下三個步驟：

(1) 軟體把加數與被加數分別寫到reg0及reg1暫存器中。

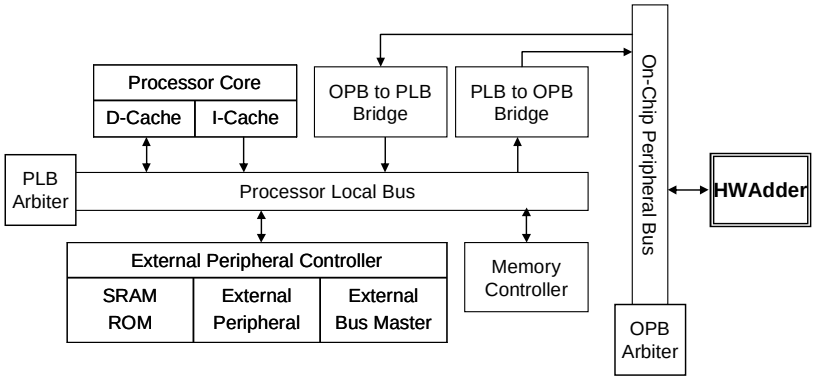
(2) 當軟體要去讀取相加的結果(reg2)時，HWAdder的加法器會將reg0與reg1內之數值相加後寫到reg2中。

(3) 回傳相加的結果給軟體並顯示在終端機軟體上。



圖四：HWAdder架構。

我們將 HWAdder 連接至 OPB 上(見圖五)，OPB(On-chip Peripheral Bus)與 PLB(Processor Local Bus)是 IBM 提出的 CoreConnect™ 匯流排架構[7]。PLB 連接至處理器且資料寬度大因此適用速度快的硬體如 DMA 或 SRAM 等裝置；相反的，OPB 屬於低速匯流排，連接如 HD 等外部硬體。兩個匯流排之間經由 Bridge 互相溝通，如資料要從 OPB 到 PLB 上則使用 OPB to PLB Bridge 來傳輸，如資料要從 PLB 到 OPB 上則使用 PLB to OPB Bridge 來傳輸。因 HWAdder 並不需要高速的存取，故我們將其連接到 OPB 上。



圖五：HWAdder 連接至 OPB。

軟硬體實作

環境設定

(1) 開啟XPS，利用Base System Builder Wizard來選擇我們的這塊裝置，首先選擇此Project存放之位置，我們放在C:\AdderDemo內。

(2) 接下來會詢問是要產生一個新的設計或使用之前產生的設計，在此我們使用"產生新的設計"，然後選擇Xilinx Virtex-II Pro ML310 Evaluation Platform，Board Revision選D。

(3) CPU的部分選擇PowerPC，Processor Clock及Bus Clock都使用100MHz。JTAG Debug Interface選擇"FPGA JTAG"，OCM選擇NONE，不使用Cache。

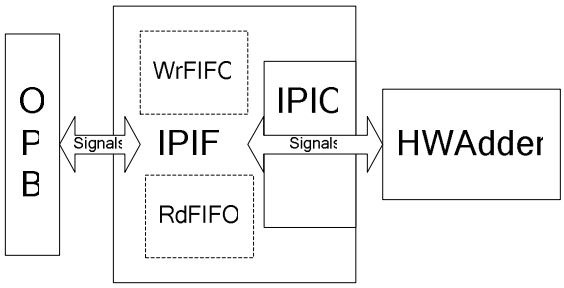
(4) I/O Devices除了RS232外其他全部取消。

(5) Peripherals把PLB BRAM IF CNTLR之Memory Size設為32KB。其他皆照預設值即可，如此環境裝置設定完成。

(6) 利用Windows XP提供之超級終端機程式和ML310溝通，首先將ML310連接到主機的com1 port上，執行超級終端機，使用連線選取com1、每秒傳輸位元：9600、資料位元：8、同位檢查：無、停止位元：1、流量控制：無。

硬體實作

(1)Xilinx提供一個介面－IPIF(見圖七)，IPIF(IP interface)將使用者建立的硬體包裝起來以便連接至OPB上，內含一些事先定義的訊號如WrFIFO與RdFIFO。IPIC(IP interconnect)包含在IPIF內提供硬體與IPIF之間的聯接，如此HWAdder可以經由IPIC連接至IPIF，再經由IPIF連接上OPB。



圖七：IPIF架構圖。

(2)開啟Tools->Create/Import Peripheral，Create templates for new peripheral...，選擇儲存路徑為C:\edk_user_repository。

(3)指定新增的裝置名稱為HWAdder，並把它加到OPB中。

(4)在IPIF中我們勾選User Logic S/W Register Support，並產生3個32bits之暫存器，接下來會產生數條訊號線，使用XPS之預設值即可，勾選產生ISE之Project檔及產生Driver檔，如此一個peripheral之模板即完成。

(5)我們將此硬體加入OPB上，Project->Add/Edit Cores...，點選Peripherals標籤在右下角找到HWAdder並按<<Add加入它。點選Bus

Connections標籤，找到HWAdder_0_sopb在其右邊點選，使其出現s。

(6)為此硬體設定Memory Map位址。點選Addresses標籤，Generate Addresses，以產生Memory Map之位址。

(7)實際撰寫HWAdder，我們修改 C:\edk_user_repository\MyProcessorIPLib\pcores\HWadder_v1_00_a\hdl\vhdl\user_logic.vhd。

```
SLAVE_REG_READ_PROC : process( slv_reg_read_select, slv_reg0, slv_reg1,
slv_reg2 ) is
    begin
         case slv_reg_read_select is
              when "100" => slv_ip2bus_data <= slv_reg0;
              when "010" => slv_ip2bus_data <= slv_reg1;
              //預設
              //when "001" => slv_ip2bus_data <= slv_reg2;
              //修改成下行
              when "001" => slv_ip2bus_data <= slv_reg0+slv_reg1;
              slv_reg2<= slv_reg0+slv_reg1;
              when others => slv_ip2bus_data <= (others => '0');
    end case;
end process SLAVE_REG_READ_PROC;
```

(8)點選 Tools->Generate Libraries and BSPs，將必要的Library及Driver複製到此專案中。

(9)撰寫Driver，從C:\AdderDemo\ppc405_0\include\xparameters.h 知道HWAdder之記憶體起始位置為XPAR_HWADDER_0_BASEADDR。

```
#define XPAR_HWADDER_NUM_INSTANCES 1
#define XPAR_HWADDER_0_DEVICE_ID 0
#define XPAR_HWADDER_0_BASEADDR 0x80000000
#define XPAR_HWADDER_0_HIGHADDR 0x8000FFFF
```

編輯C:\AdderDemo\ppc405_0\include\HWadder.h加入。

```
Xuint32 Add(Xuint32 a, Xuint32 b){
    HWADDER_mWriteReg(XPAR_HWADDER_0_BASEADDR,
                          HWADDER_SLAVE_REG0_OFFSET, a);
    HWADDER_mWriteReg(XPAR_HWADDER_0_BASEADDR,
                          HWADDER_SLAVE_REG1_OFFSET, b);
    return HWADDER_mReadReg(XPAR_HWADDER_0_BASEADDR,
                          HWADDER_SLAVE_REG2_OFFSET);
}
```

如此我們便可以透過Add函式利用HWAdder來計算兩數相加。

軟體實作

(1) 修改TestApp.c。xil_printf及print可以將結果輸出至終端機軟體中。

```
#include “xparameters.h”
#include "HWAdder.h"
#include "xutil.h"
int main (void) {
   Xuint32 i=0; Xuint32 sum=0;
   print("-- Entering main() --\r\n");
   for(i=1;i<=10;i++){
        sum = Adder(i，sum);
        xil_printf("%d. %d\r\n"， i， sum);
   }
   xil_printf("%d\r\n"，sum);
   print("-- Exiting main() --\r\n");
   return 0;
}
```

(2) Compile TestApp.c，Tools->Build All User Applications。

軟硬體協同模擬

(1) 在TestApp.c中加入以下程式碼

```
Xuint32 Adder(Xuint32 a， Xuint32 b){
    return a+b;
}
```

下載至板子中觀察結果後發現與期待結果相同，證明設計無誤。

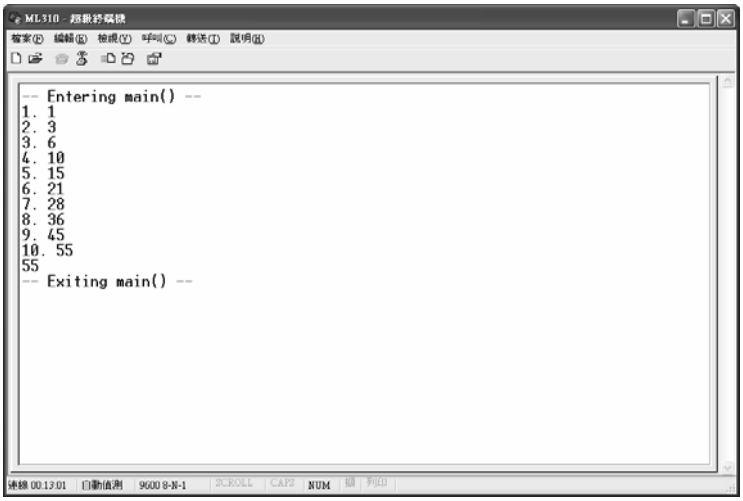
軟硬體協同驗證

(1) 將硬體資訊包進Bitstream中，Tools->Generate Bitstream。

(2) 將軟體包進Bitstream中，Tools->Update Bitstream。

(3) 將Bitstream下載到ML310中驗證其正確性，Tools->Download。

(4) 從畫面中看到我們的系統已可正常運作(見圖十一)



圖十一：計算結果之呈現。

4. 總結

過去軟體人員通常不懂硬體的設計而硬體人員也不懂軟體的設計，因此常導致軟硬體互相溝通時的問題。在軟硬體協同設計的流程下，設計人員需要兼備軟硬體的設計方法，因此進入門檻提高不少。然而在SoC的趨勢下，軟硬體協同設計的實務經驗值得相關人員深入探討。本文敘述了軟硬體協同設計的流程，並依照此流程實作一個加法器，在實作的過程中了解匯流排協定，因為要連接上匯流排所以必須實作出其訊號線，驗證訊號是否有錯誤，在此花費的時間很大。

[1]ML310 Documentation and Tutorials, http://www.xilinx.com/products/boards/ml310/current/.
[2]Vertex-II Pro System Wake-up Solution, http://www.xilinx.com/bvdocs/userguides/ug028.pdf.
[3]蔡文聖、黃正才, "應用於系統晶片之矽智財共同驗證與快速雛型技術," 系統晶片 001 期.
[4]ML310 base design tutorial, http://www.xilinx.com/products/boards/ml310/current/.
[5]Xilinx Platform Studio User Guide, http://www.xilinx.com/ise/embedded/edk6_2docs/platform_studio_ug.pdf.
[6]ISE 6 In-Depth Tutorial, http://direct.xilinx.com/direct/ise6_tutorials/ise6tut.pdf.
[7]On-Chip Peripheral Bus, http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/9A7AFA74DAD200D087256AB30005F0C8/$file/OpbBus.pdf.