

**Imagination大學計劃**

**RVfpga-SoC實驗1**

**RVfpga-SoC簡介**

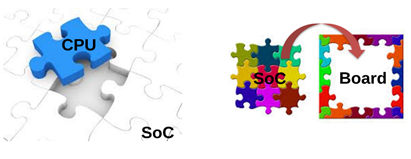
**表1. RVfpga術語**

|  |  |
| --- | --- |
| **名稱** | **說明** |
| **課程** | |
| **RVfpga** | 一門介紹如何使用RVfpgaNexys、RVfpgaSim和RISC-V晶片上系統（System-on-Chip，SoC）的課程，內容涵蓋通過新增週邊設備來執行程式和擴展系統（RVfpga實驗1-10），以及通過執行模擬、測量效能、新增指令和修改記憶體系統來探究核心和記憶體統（RVfpga實驗11-20）。在本課程中，使用者還將學習如何使用RISC-V工具鏈（編譯器和偵測工具）和模擬器、Verilator HDL模擬器，以及Western Digital的Whisper指令集模擬器（Instruction Set Simulator，ISS）。 |
| **RVfpga-SoC** | 一門介紹如何使用SweRV核心、記憶體和週邊設備等構件，從頭開始構建SweRVolfX SoC子集的課程。本課程還介紹如何將Zephyr即時作業系統（Real-Time Operating System，RTOS）載入到SweRVolf上，以及如何基於該作業系統執行包括Tensorflow Lite的hello world範例在內的程式。 |
| **核心和SoC** | |
| **SweRV EH1 核心** | 由Western Digital開發的開放原始碼商用RISC-V核心  （<https://github.com/chipsalliance/Cores-SweRV>）。 |
| **SweRV EH1 核心組合** | 一種增加了記憶體（ICCM、DCCM和指令快取）、可編程中斷控制器（Programmable Interrupt Controller，PIC）、匯流排介面和偵錯單元的SweRV EH1核心（<https://github.com/chipsalliance/Cores-SweRV>）。 |
| **SweRVolfX** | 我們在RVfpga課程中使用的晶片上系統。它是SweRVolf的延伸模組。  **SweRVolf**（<https://github.com/chipsalliance/Cores-SweRVolf>）：一種圍繞SweRV EH1核心組合構建的開放原始碼SoC。它增加了開機ROM、UART介面、系統控制器、互連（AXI互連、Wishbone互連和AXI轉Wishbone橋接器）以及SPI控制器。  **SweRVolfX**：與SweRVolf相比增加了四個新的週邊設備：GPIO、PTC、另一個SPI，以及用於8位7段顯示器的控制器。 |
| **RVfpgaNexys** | 以Nexys A7電路板及其週邊設備為目標的SweRVolfX SoC。它增加了DDR2介面、CDC（時鐘域交叉）單元、BSCAN邏輯（用於JTAG介面）和時鐘產生器。  RVfpgaNexys與SweRVolf Nexys基本相同（<https://github.com/chipsalliance/Cores-SweRVolf>），只是後者基於SweRVolf。 |
| **RVfpgaSim** | 一種具有測試平台包裝程式和AXI記憶體的SweRVolfX SoC，用於模擬用途。  RVfpgaSim與SweRVolf Sim基本相同（<https://github.com/chipsalliance/Cores-SweRVolf>），只是後者基於SweRVolf。 |

# 簡介

1. **片上系統簡介**

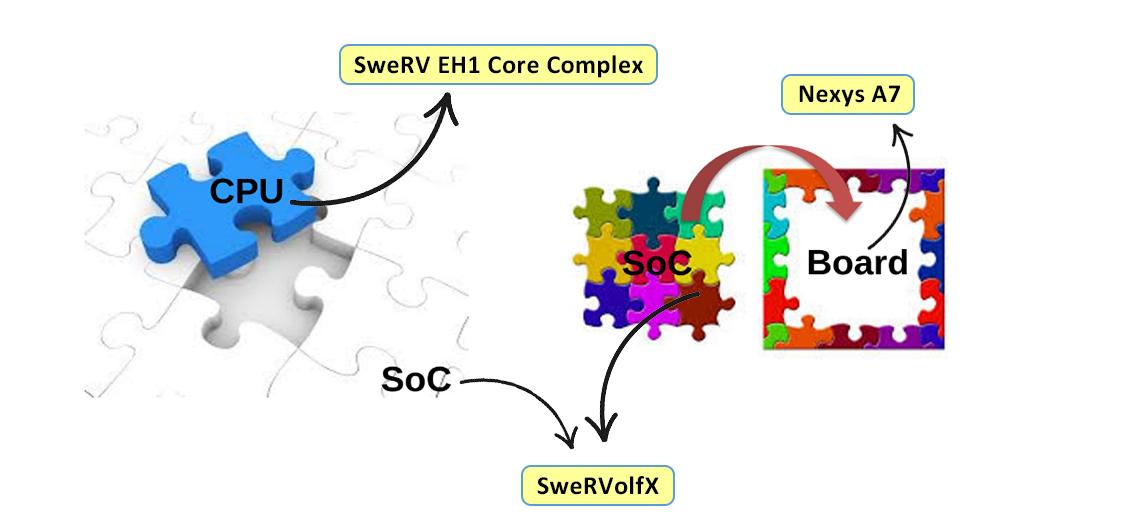
在本實驗中，我們將介紹如何通過構件構建RISC-V晶片上系統（SoC）。**SoC**就是積體電路（即IC），可將整個電子或電腦系統整合到其中。SoC包括一個核心以及載入作業系統和執行程式所需的所有週邊設備和介面。圖1說明了嵌入式系統的一般階層結構，從處理器核心開始，然後是圍繞核心編譯的SoC，最後是系統和電路板介面。



**圖1. 典型嵌入式系統**

SoC的設計過程從FPGA的原型設計開始。我們的重點是使SoC以FPGA為目標。

我們將使用的RISC-V CPU是Western Digital的**SweRV EH1核心組合**，而將在本實驗中設計的SoC是**SweRVolfX**的子集，我們將使其以**Nexys A7-100T**電路板為目標。圖2所示為各種元件及其裝配方式。



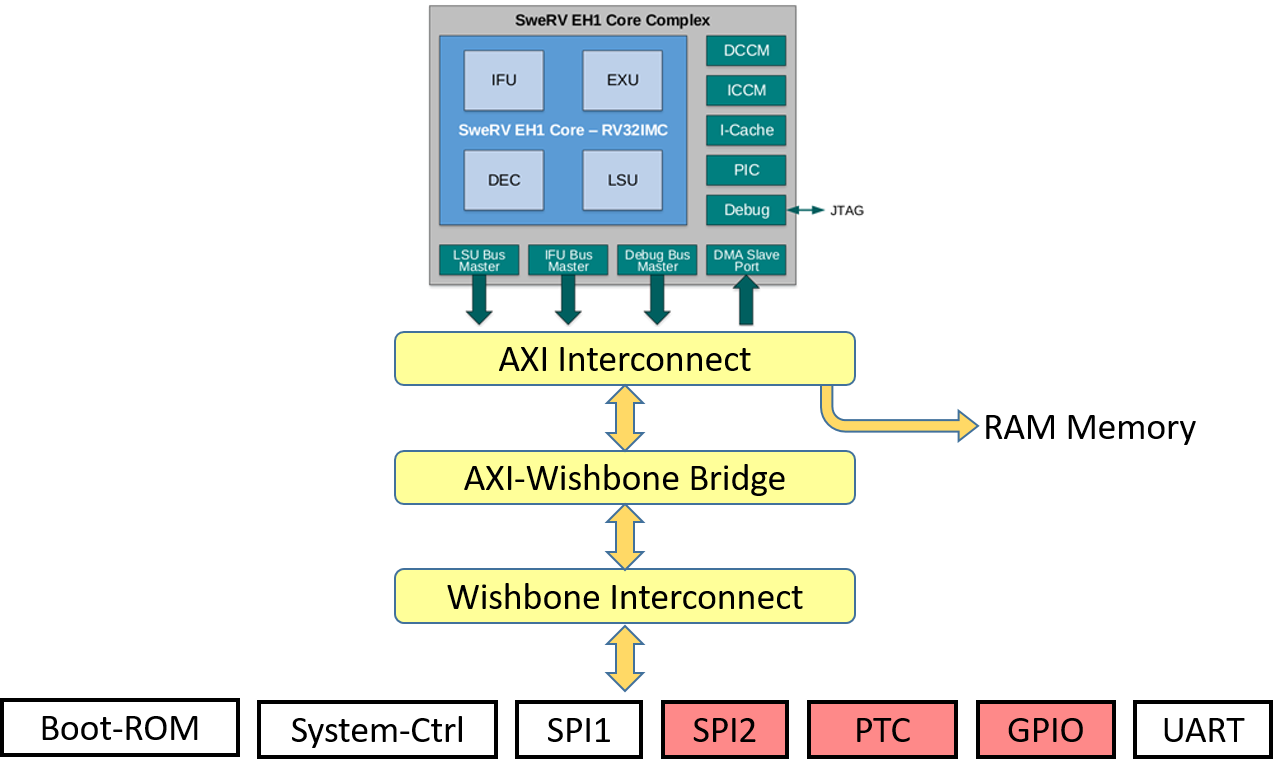
**圖2. 基於RVfpga系統的嵌入式系統**

1. **SweRVolfX和RVfpga系統簡介**

在開始本實驗之前，我們強烈建議學員閱讀RVfpga課程入門指南以瞭解整個RVfpga系統。以下是RVfpga課程中引入的RVfpga系統的簡要說明。

表1列出了RVfpga系統的層級結構（從SweRV EH1核心開始，一直到RVfpgaNexys和RVfpgaSim）。RVfpga系統中使用的晶片上系統（SoC）稱為**SweRVolfX**（如圖3所示），它基於**SweRVolf**版本0.7.3（[https://github.com/chipsalliance/Cores-SweRVolf/releases/tag/v0.7.3](https://github.com/chipsalliance/Cores-SweRVolf/releases/tag/v0.7)），而後者基於**SweRV EH1核心組合**構建。除SweRV EH1核心組合外，SweRVolf SoC還包括開機ROM、UART、系統控制器和SPI控制器。SweRV EH1核心使用AXI匯流排，週邊設備使用Wishbone匯流排；SoC還具有AXI-Wishbone橋接器。

在RVfpga系統中，SweRVolf SoC擴展了一些功能，例如另一個SPI控制器（SPI2）、GPIO（通用輸入/輸出）控制器和PTC（PWM/定時器/計數器）模組。（在圖3中，這些新週邊設備以紅色顯示）。該晶片上系統稱為**SweRVolfX**（X代表擴展）。



**圖3. SweRVolfX**

表4列出了通過Wishbone互連連接到SweRV EH1核心的週邊設備的記憶體映射位址。

**表4. SweRVolf的記憶體映射位址**

|  |  |
| --- | --- |
| **系統** | **位址** |
| 開機ROM | 0x80000000 - 0x80000FFF |
| 系統控制器 | 0x80001000 - 0x8000103F |
| SPI1\* | 0x80001040 - 0x8000107F |
| SPI2\* | 0x80001100 - 0x8000113F |
| 定時器\* | 0x80001200 - 0x8000123F |
| GPIO\* | 0x80001400 - 0x8000143F |
| UART | 0x80002000 - 0x80002FFF |

\* SweRVolfX中新增的週邊設備

1. **RVfpga-SoC簡介**

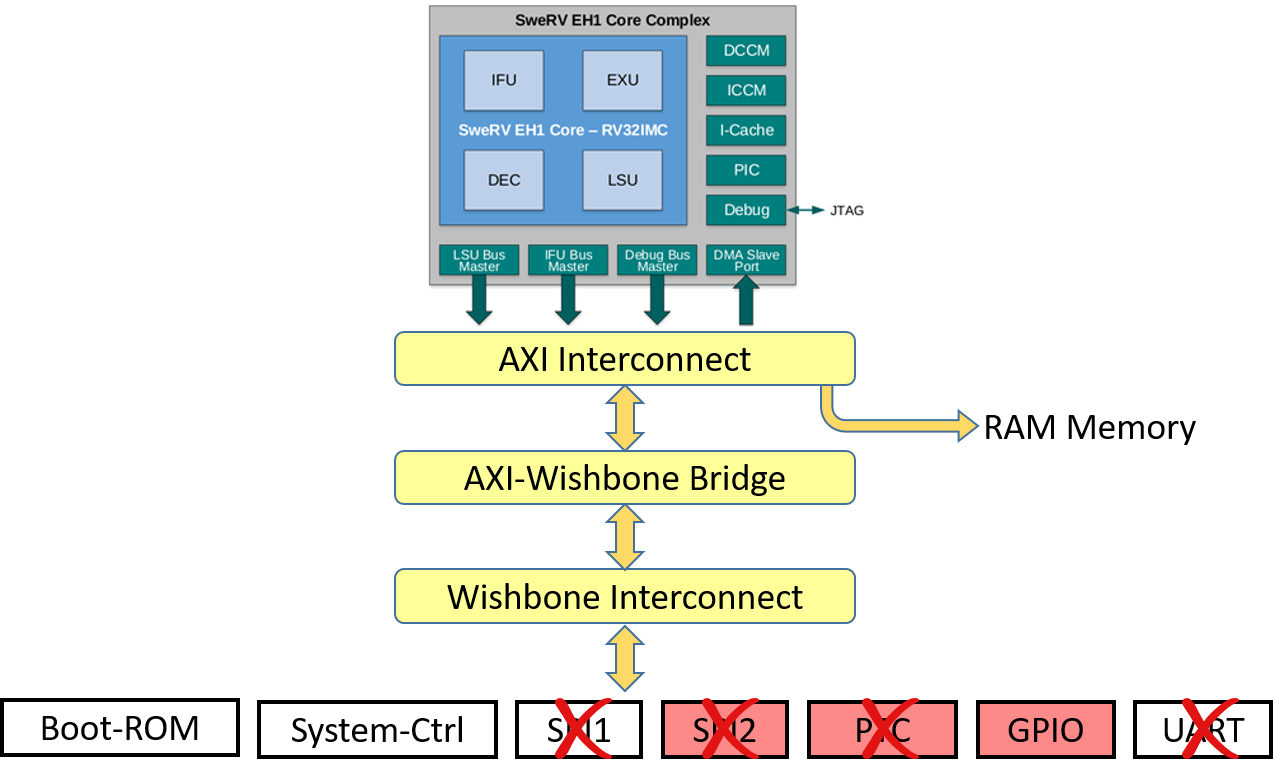
在RVfpga中，介紹SweRVolfX時沒有提供關於如何建立SweRVolfX的任何細節。RVfpga-SoC課程介紹如何使用SweRV核心、記憶體和週邊設備等構件，以從頭開始構建SweRVolfX SoC的子集。

本實驗將逐步介紹如何從CPU（SweRV EH1核心組合）開始，然後如何將其構建到SoC中。我們將使用Vivado區塊設計工具。Vivado的區塊設計工具以圖形方式簡化了組件佈線，使流程更易於理解和可視化。這種可視化方法還會說明每個模組如何與其他模組連接以形成SoC。

這些模組可以分為三個主要區塊或類別：

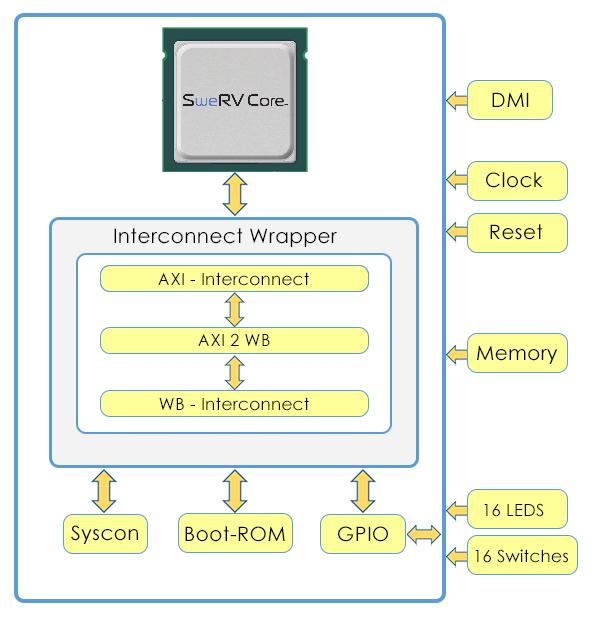
1. CPU（SweRV EH1核心組合）
2. 互連（AXI互連、AXI2WB和WB互連）
3. 週邊設備（開機ROM、GPIO控制器和系統控制器）

SweRVolfX有許多不同的模組，但有些並不是準系統RISC-V SoC所必需的。因此，我們精簡掉了這些額外的模組，以此來簡化實驗並專注於使CPU核心工作所需的準系統功能。圖4所示為我們將排除的模組（UART、PTC、SPI1和SPI2）。



**圖4. SweRVolfX的子集**

圖5所示為我們將實現的SoC的高階方塊圖。



**圖5. 實驗1 SoC的高階方塊圖**

為了便於學習和理解，構成互連（AXI互連、Wishbone互連和AXI轉Wishbone橋接器）的一些元件已封裝到一個互連包裝程式模組中。

|  |
| --- |
| 對於專注於CPU和CPU內部的實驗，請參閱RVfpga課程。RVfpga（也稱為RISC-V FPGA）課程包含相關說明、工具和實驗，用於使商用RISC-V處理器和SoC以現場可編程閘道陣列（Field-Programmable Gate Array，FPGA）為目標，然後使用它進行擴展來學習電腦架構、數化設計、嵌入式系統和編程的相關知識。  有關RVfpga的更多資訊，請造訪[https://university.imgtec.com/rvfpga/。](https://university.imgtec.com/rvfpga/) |

# 要求

要完成本實驗，您需要安裝以下軟體：

* Vivado 2019.2 Web Pack （請參閱安裝指南（第4頁））
* Digilent Board Files （請參閱安裝指南（第5頁））

**重要資訊：**在開始RVfpga-SoC實驗之前，我們強烈建議按照RVfpga-SoC安裝指南完成相關軟體的安裝。

例如，如果您尚未完成，請按照RVfpga-SoC安裝指南中的說明安裝Xilinx的Vivado。確保您已將從Imagination大學計劃下載的*RVfpgaSoC*資料夾複製到電腦。

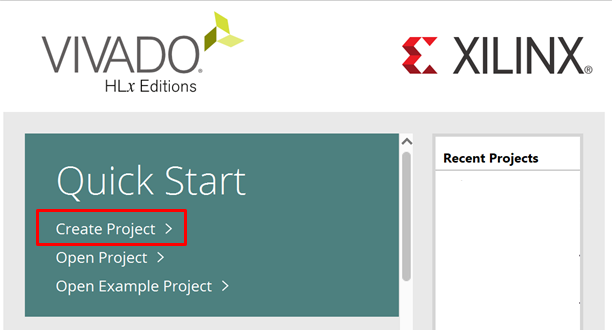
# 建立Vivado專案

您將使用Xilinx的Vivado設計套件通過RTL（定義系統的Verilog檔案）構建SweRVolfX子集。按照下面詳述的步驟建立Vivado專案。

**步驟1. 開啟Vivado**

如果您沒有按照RVfpga-SoC安裝指南中的說明在電腦上安裝Vivado，請立即安裝。請確保同時安裝電路板檔案。

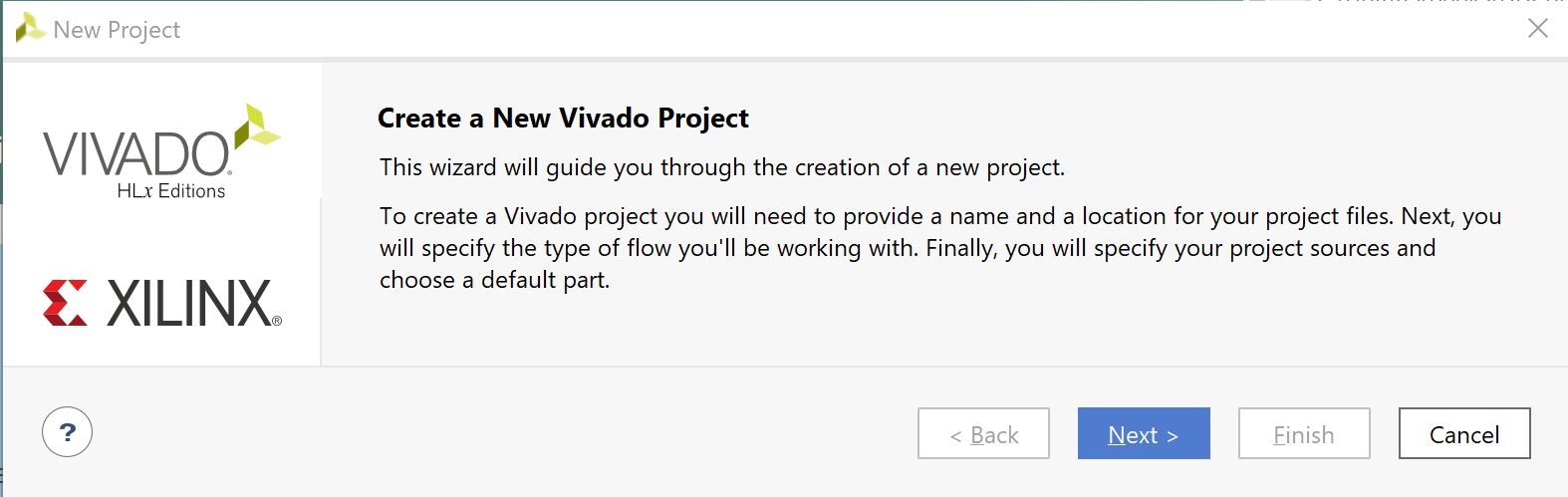
現在，執行Vivado（在**Linux**中，開啟終端機並輸入：**vivado**；在**Windows**中，從「Start」（開始）功能表開啟Vivado）。Vivado歡迎畫面隨即開啟。按一下「Create Project」（建立專案）（請參閱圖6）。



**圖6. Vivado歡迎畫面：「Create Project」（建立專案）**

**步驟2. 建立一個新的RTL專案**

「Create a New Vivado Project」（建立一個新的Vivado專案）精靈隨即開啟（請參閱圖7）。按「Next」（下一步）。

****

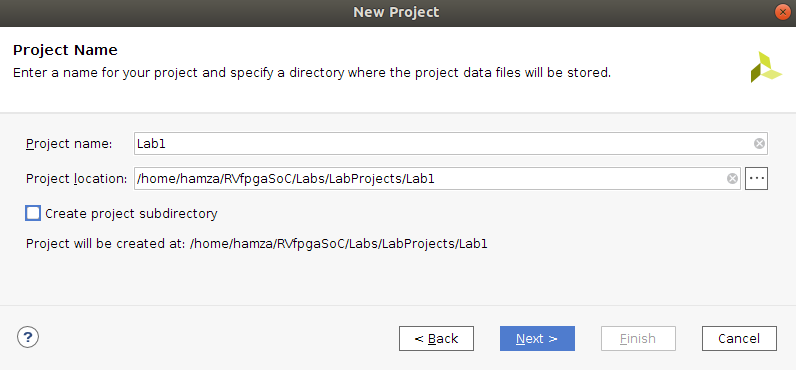
**圖7.「Create a New Vivado Project」（建立一個新的Vivado專案）精靈**

輸入專案名稱“**Lab1**”，沒有空格。然後按「Next」（下一步）（請參閱圖8）。

選擇以下專案位置路徑：

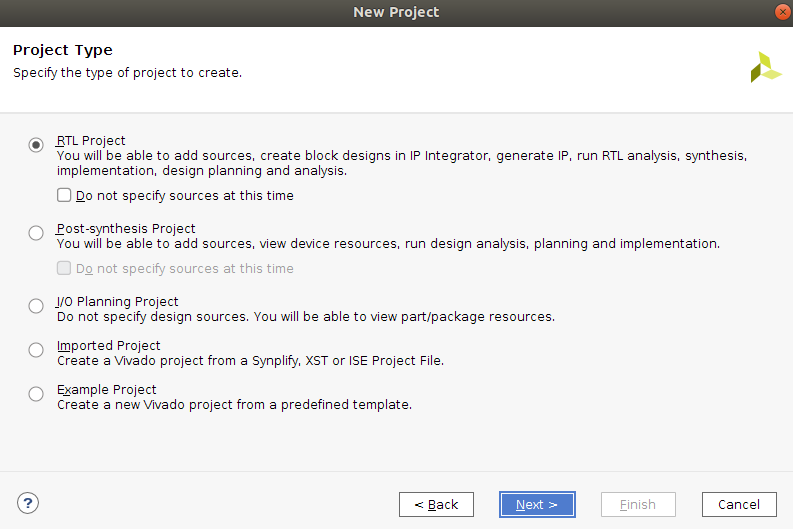
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabProjects/Lab1

取消勾選「Create project subdirectory」（建立專案子目錄）核取方塊，因為“**LabProjects**”資料夾中已經有一個名為“**Lab1**”的資料夾。



**圖8. 專案名稱**

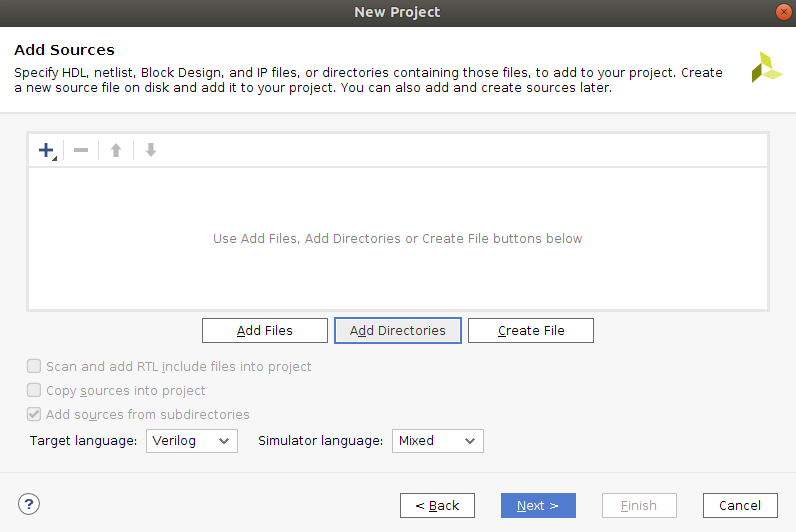
選擇專案類型「RTL Project」（RTL專案），然後按「Next」（下一步）（請參閱圖 9）。

****

**圖9. RTL專案**

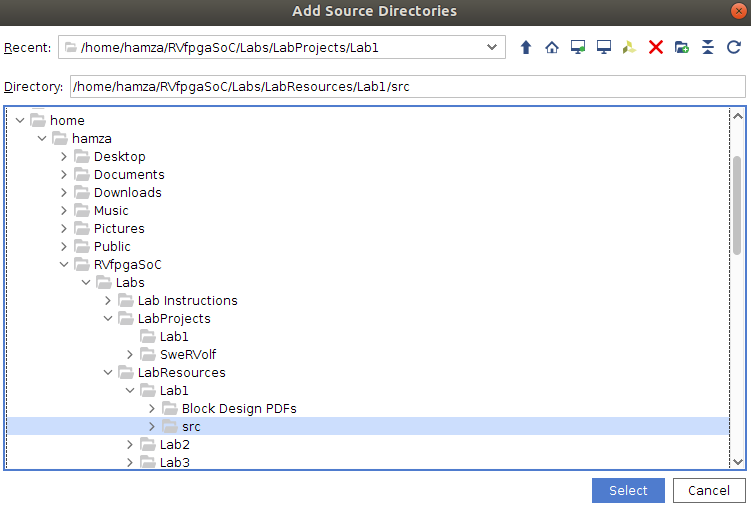
**步驟3. 新增RTL原始程式檔和限制檔**

在「Add Sources」（新增原始程式碼）視窗中，按一下「**Add Directories」**（新增目錄）（請參閱圖10）。

****

**圖10. 新增原始程式碼目錄**

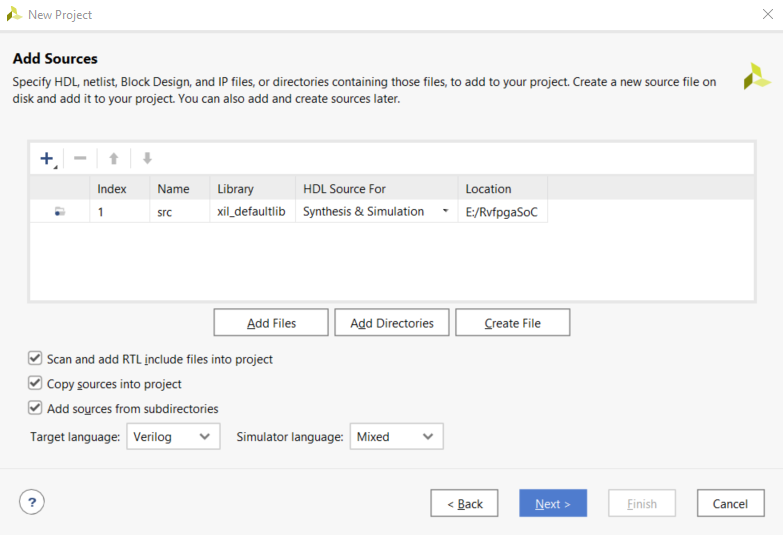
現在選擇“src”目錄，路徑如下：[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src（請參閱圖11）。



**圖11. 選擇“src”目錄**

按一下「Select」（選擇）。

然後按一下「**Add Files」**（新增檔案）按鈕。

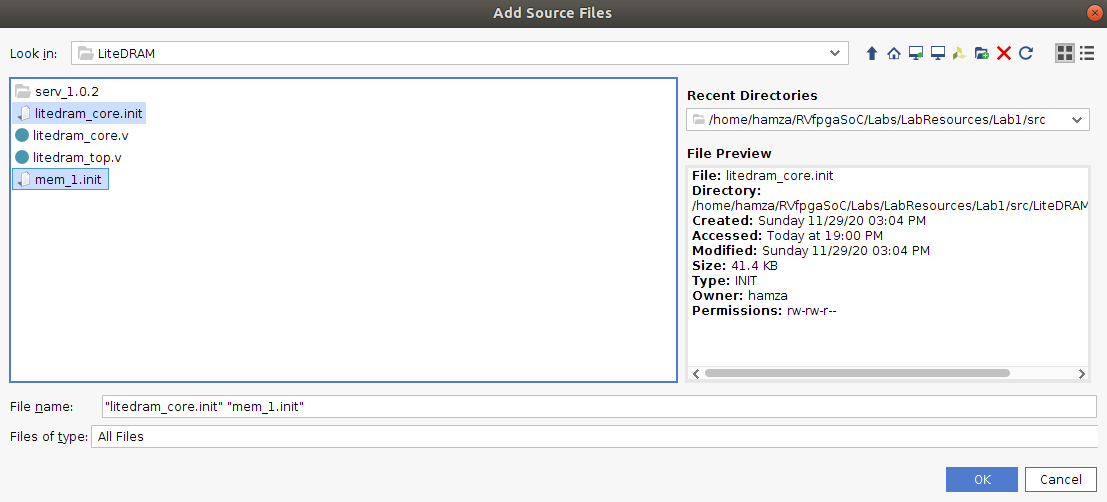
****

**圖12. 新增檔案**

選擇「**All Files」**（所有檔案）作為檔案類型。現在導覽到我們剛剛新增的**src**目錄下的**LiteDRAM**目錄。

* [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/LiteDRAM/mem\_1.init
* [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/LiteDRAM/litdram\_core.init

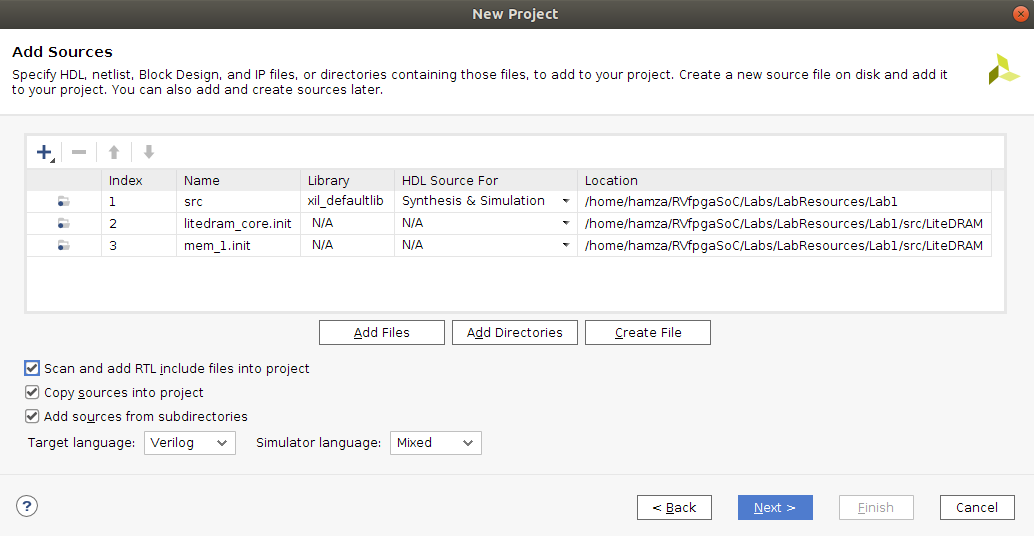
選擇兩個“**.init**”檔案，然後按一下「OK」（確定）以新增它們（請參閱圖13）。



**圖13. 新增LiteDram原始程式檔**

確保勾選全部三個核取方塊（請參閱圖14）。

按「**Next」**（下一步）進入下一步。



**圖14. 新增原始程式碼**

接下來將新增系統的限制檔。這些檔案會將訊號名稱映射到電路板上的引腳。例如，Nexys A7 FPGA電路板的LED通過PCB走線連接到電路板上的FPGA引腳。Vivado必須知道這一點才能正確地將RTL中的訊號名稱映射到對應的FPGA引腳。例如，   
*[RVfpgaSoCPath]/RVfpgaSoC/src/rvfpga.xdc*檔（Xilinx設計約束檔案）中的以下行表示FPGA引腳H17映射到最低有效LED（o\_led[0]），且其使用LVCMOS 3.3V訊號：

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD LVCMOS33 } [get\_ports { o\_led[0] }]

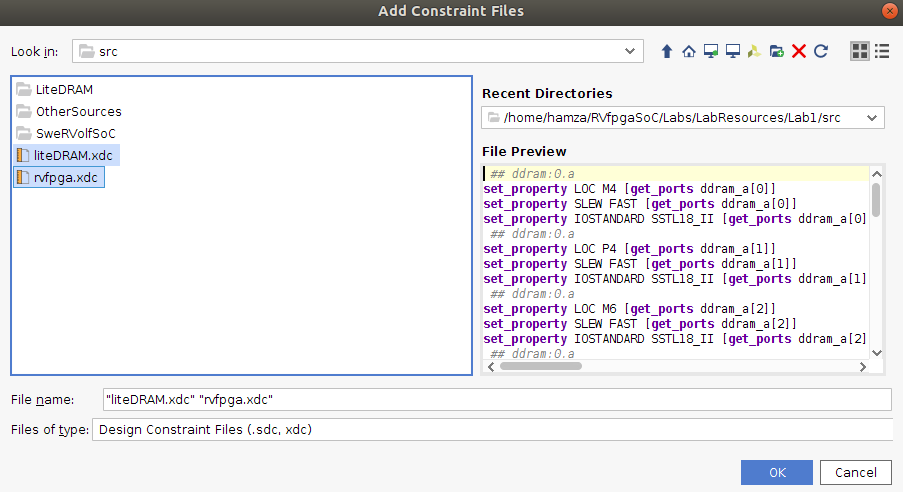
請注意，訊號名稱o\_led是RVfpga的Verilog程式碼中用於驅動Nexys A7電路板LED的名稱。

在「Add Constraints」（新增約束）視窗中，按一下「**Add Files**」（新增檔案）並選擇以下兩個檔案（請參閱圖15）：

*[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/rvfpga.xdc*

*[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/litedram.xdc*

然後按「**Next**」（下一步）。

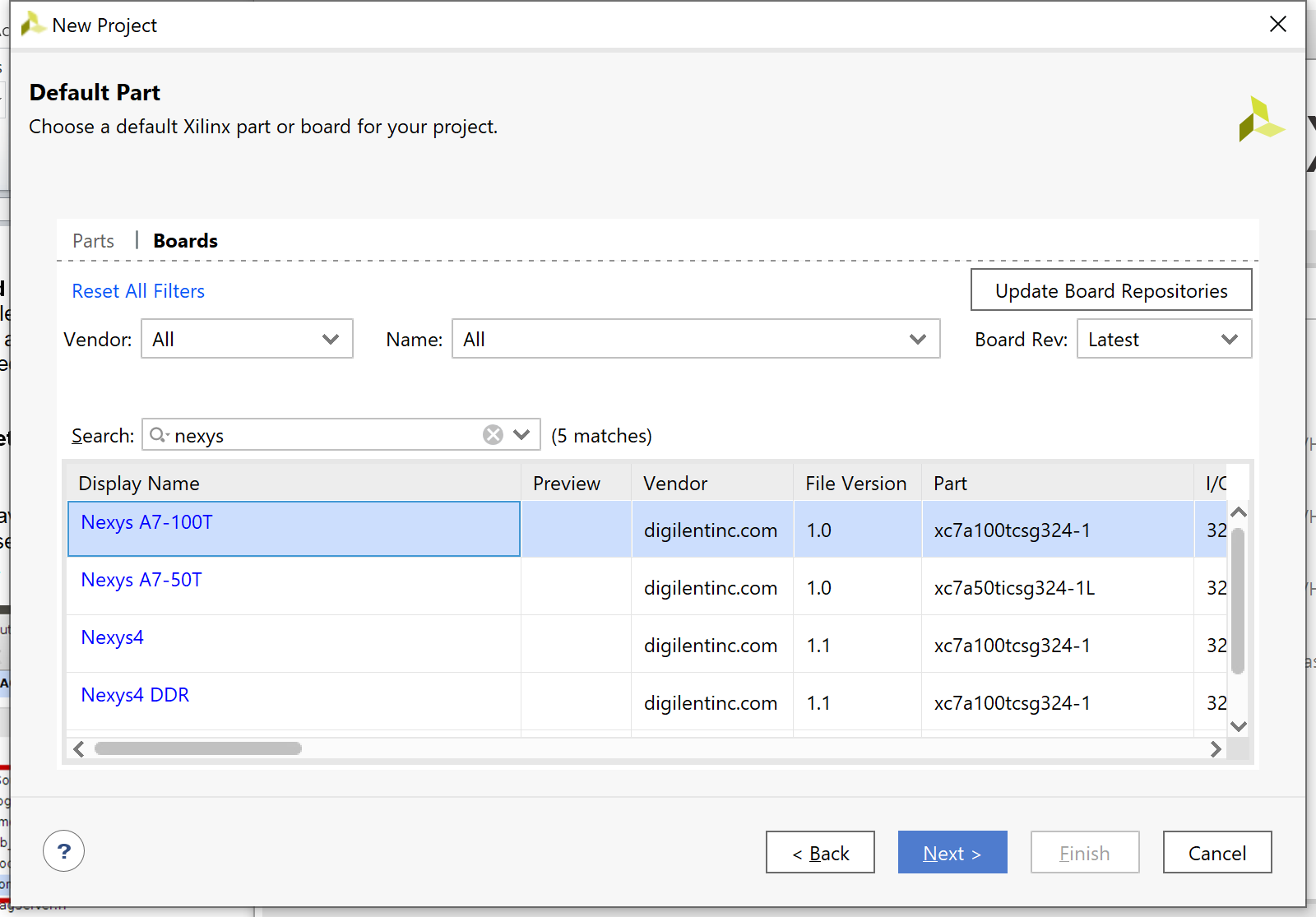
****

**圖15. 新增約束檔**

**步驟4. 選擇Nexys A7作為目標電路板**

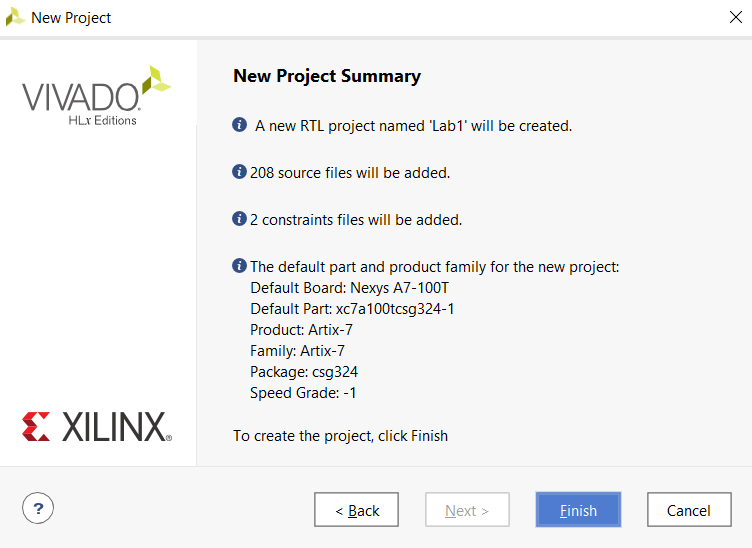
在「Default Part」（預設零件）視窗中，按一下「Boards」（電路板），然後選擇Nexys A7-100T（請參閱圖16）。可以使用「Search」（搜尋）方塊來縮小結果範圍。您還將注意到，「Part」（零件）欄中會列出實際目標FPGA的名稱：xc7a100tcsg324-1。這表明目標FPGA為Xilinx Artix-7 FPGA，具有100000個採用CSG（晶片級網格）套件的等效閘門和324個引腳。

按「Next」（下一步）。



**圖16. 選擇目標板：Nexys A7-100T**

在「New Project Summary」（新增專案摘要）視窗中，按一下「**Finish**」（完成）（請參閱圖17**）。**

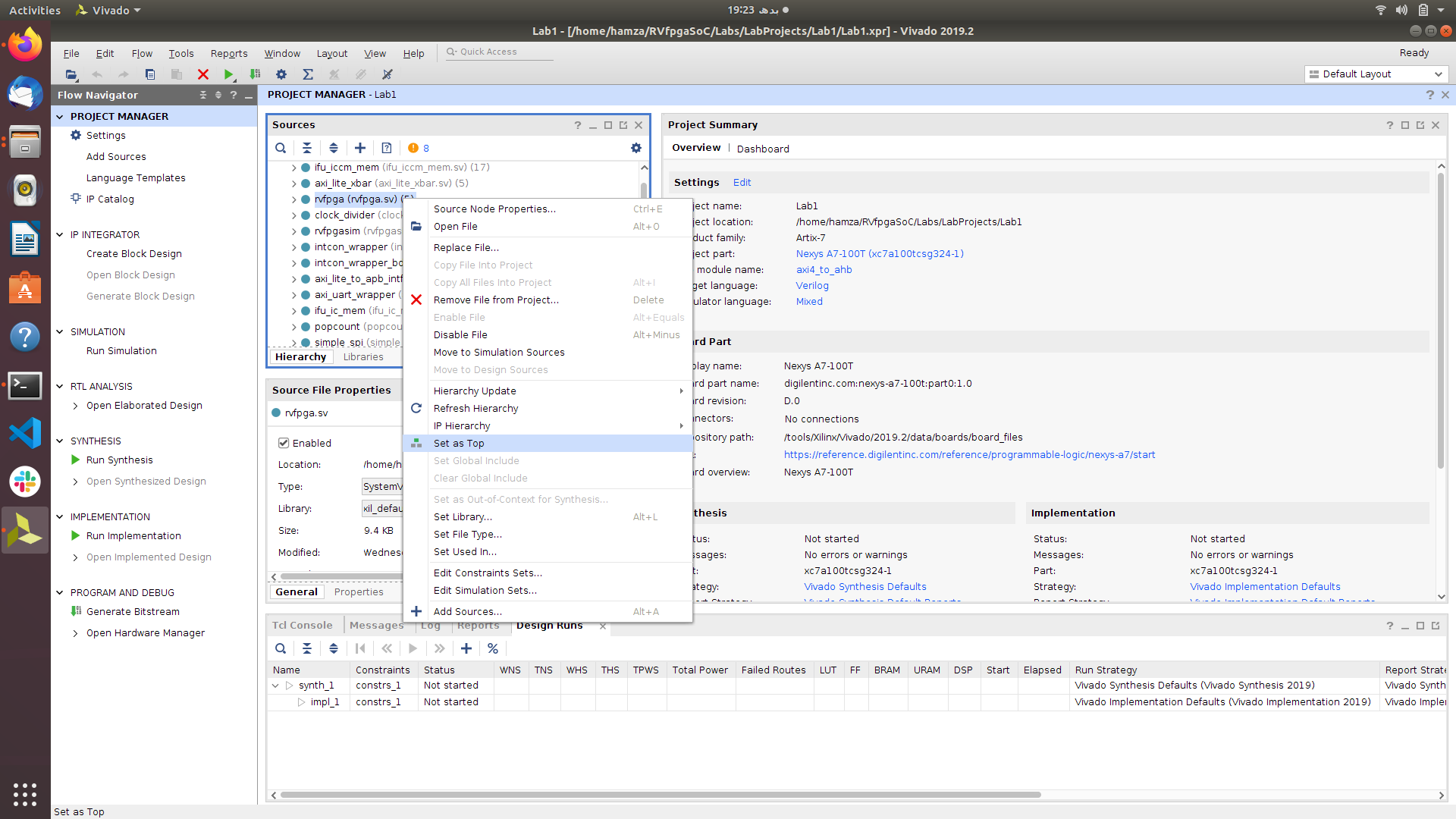


**圖17.「New Project Summary」（新增專案摘要）視窗**

請注意，專案設定完成後，系統將指出檔案存在語法錯誤，下一步將修復此錯誤。

**步驟5. 將rvfpga設為頂層模組**

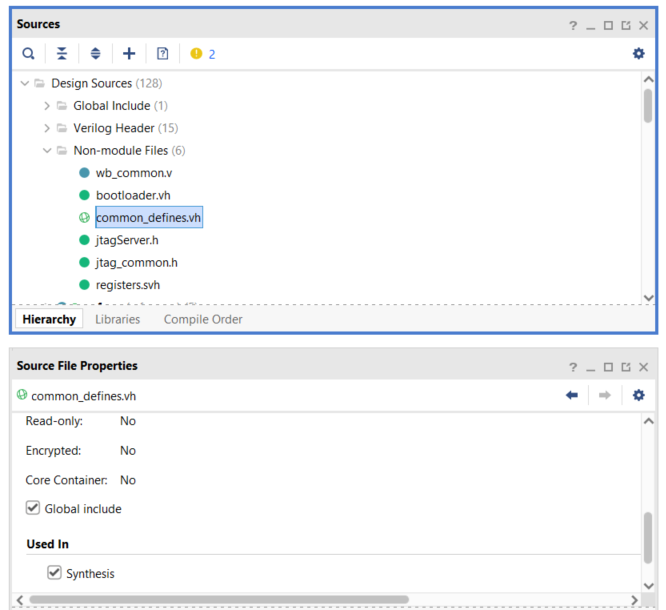
專案將初始化。接下來將rvfpga模組設為頂層模組。在「Design Sources」（設計原始程式碼）下的「Sources」（原始程式碼）窗格中向下捲動，用滑鼠右鍵按一下rvfpga模組，然後選擇「Set as Top」（設為頂層模組）（請參閱圖18）。也可以通過在搜尋方塊中輸入相應名稱來尋找rvfpga模組，如下圖所示。這會將rvfpga設為階層中級別最高的模組，並將其作為要在FPGA上合成並實作的目標。將rvfpga設為頂層模組後，階層將更新。



**圖18. 將rvfpga設為頂層模組**

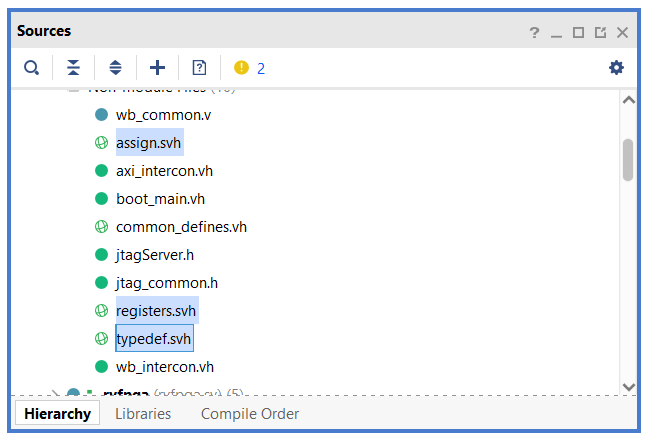
**步驟6. 將Verilog標頭檔設定為全域包含檔**

然後，還是在「Design Sources」（設計原始程式碼）下的「Sources」（原始程式碼）窗格中，展開「Non-modules」（非模組）檔案群組，然後按一下common\_defines.vh。該檔案的屬性將顯示在「Sources」（原始程式碼）窗格下面的「Source File Properties」（原始程式檔屬性）窗格中。按一下「Global Include」（全域包含檔）以勾選方塊（請參閱圖19）。階層隨即更新，並將該檔案包含在「Design Sources」（設計原始程式碼）/「Global Include」（全域包含檔）中。



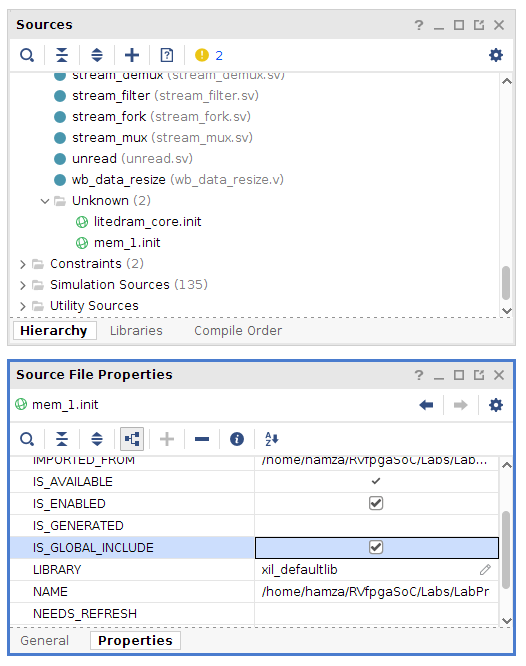
**圖19. 將common\_defines.vh設為全域包含檔**

同理，將**“assign.svh”**、**“registers.svh”**和**“typedef.svh”**SystemVerilogHeader檔案設定為全域包含檔（請參閱圖20）。



**圖20. 將“.svh”檔案設定為全域包含檔**

現在展開「**unknown」**（未知）檔案組並按一下「**litedram\_core.init**」。然後按一下「Source File Properties」（原始程式檔屬性）面板中「General」（一般）按鈕旁邊的「Properties」（屬性）按鈕。按一下「**IS\_Global\_INCLUDE**」，選中該核取方塊（請參閱圖21）。

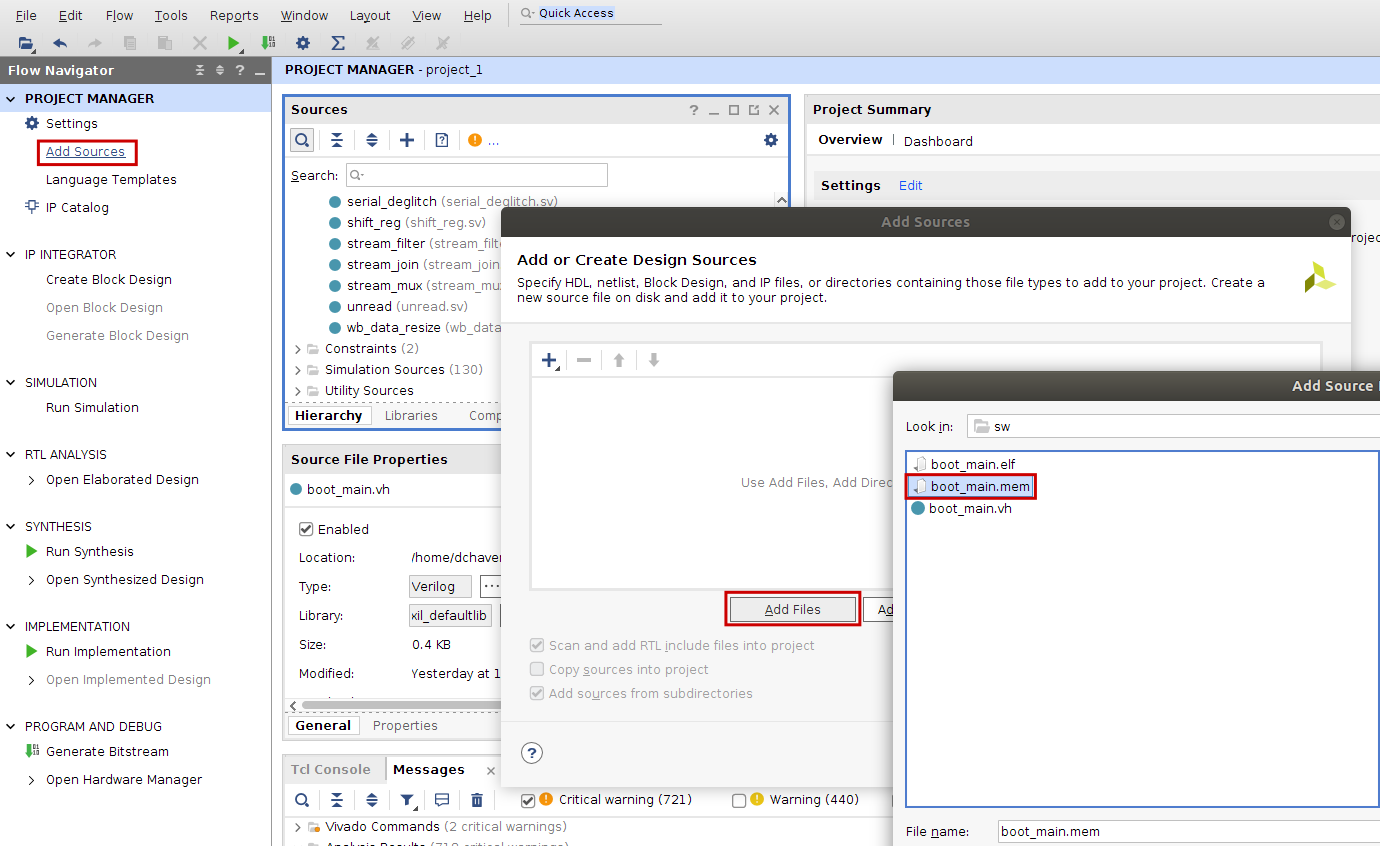


**圖21. 將litdram\_core.init設定為全域包含檔**

現在對“**mem\_1.init**”檔執行與“**litedram\_core.init**”檔相同的操作，將該檔案也設定為全域包含檔。

**步驟7. 將boot\_main.mem新增到專案中**

在「Flow Navigator」（流程導覽器）窗格中，按一下「Add Sources」（新增原始程式碼），保持預設選項（「Add or create design sources」（新增或建立設計原始程式碼）），然後按一下「Add Files」（新增檔案）（請參閱圖22）。導覽至   
*[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/SweRVolfSoC/BootROM/sw*並選擇*boot\_main.mem*（如圖22所示）。階層將更新並會將該檔案包含在「Design Sources」（設計原始程式碼）/「Memory File」（記憶體檔案）中。



**圖22. 新增記憶體檔案boot\_main.mem**

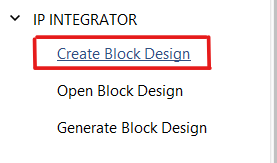
現在已經新增了設計原始程式檔，我們可以繼續開始建立區塊設計。

# 建立區塊設計

我們將使用Vivado的區塊設計功能來新增建立SweRVolfX子集所需的模組，然後將模組彼此連接。

**步驟1. 按一下「Create Block Design」（建立區塊設計）**

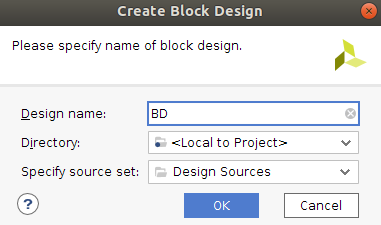
按一下「IP Integrator」（IP整合器）標題下的「Create block design」（建立區塊設計），在「Flow Navigator」（流程導覽器）中建立一個新的區塊設計（請參閱圖23）。



**圖23. 建立區塊設計**

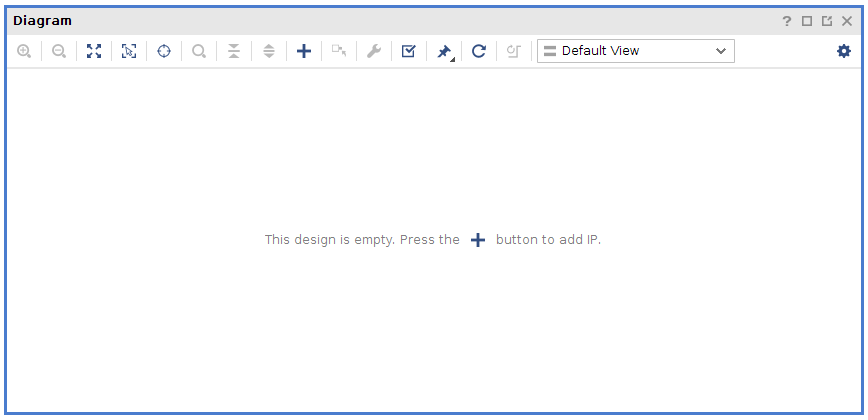
**步驟2. 選擇區塊設計的名稱**

選擇“**BD**”作為「Design name」（設計名稱），以避免實驗後期出現命名衝突（請參閱圖24）。

****

**圖24. 選擇區塊設計的名稱和目錄**

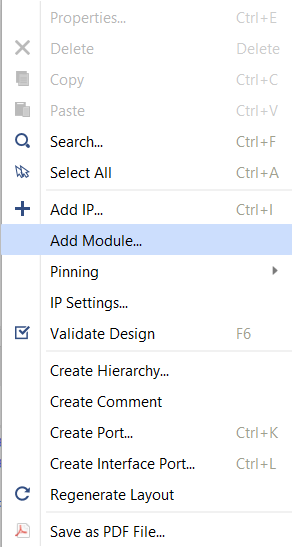
現在您將看到一個空白的區塊設計圖面板。（請參閱圖25）



**圖25. 空白區塊設計**

**步驟3. 將模組新增到區塊設計中**

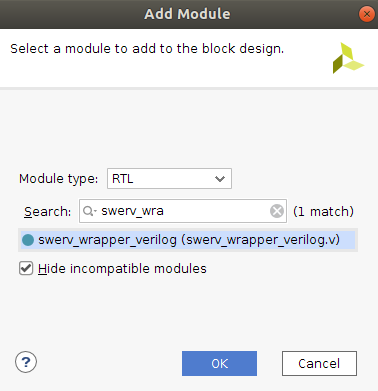
現在，我們可以開始將模組新增到區塊設計中。為此，右鍵按一下空白區塊設計並選擇「**Add Module**」（新增模組）選項（請參閱圖26）。



**圖26. 新增模組**

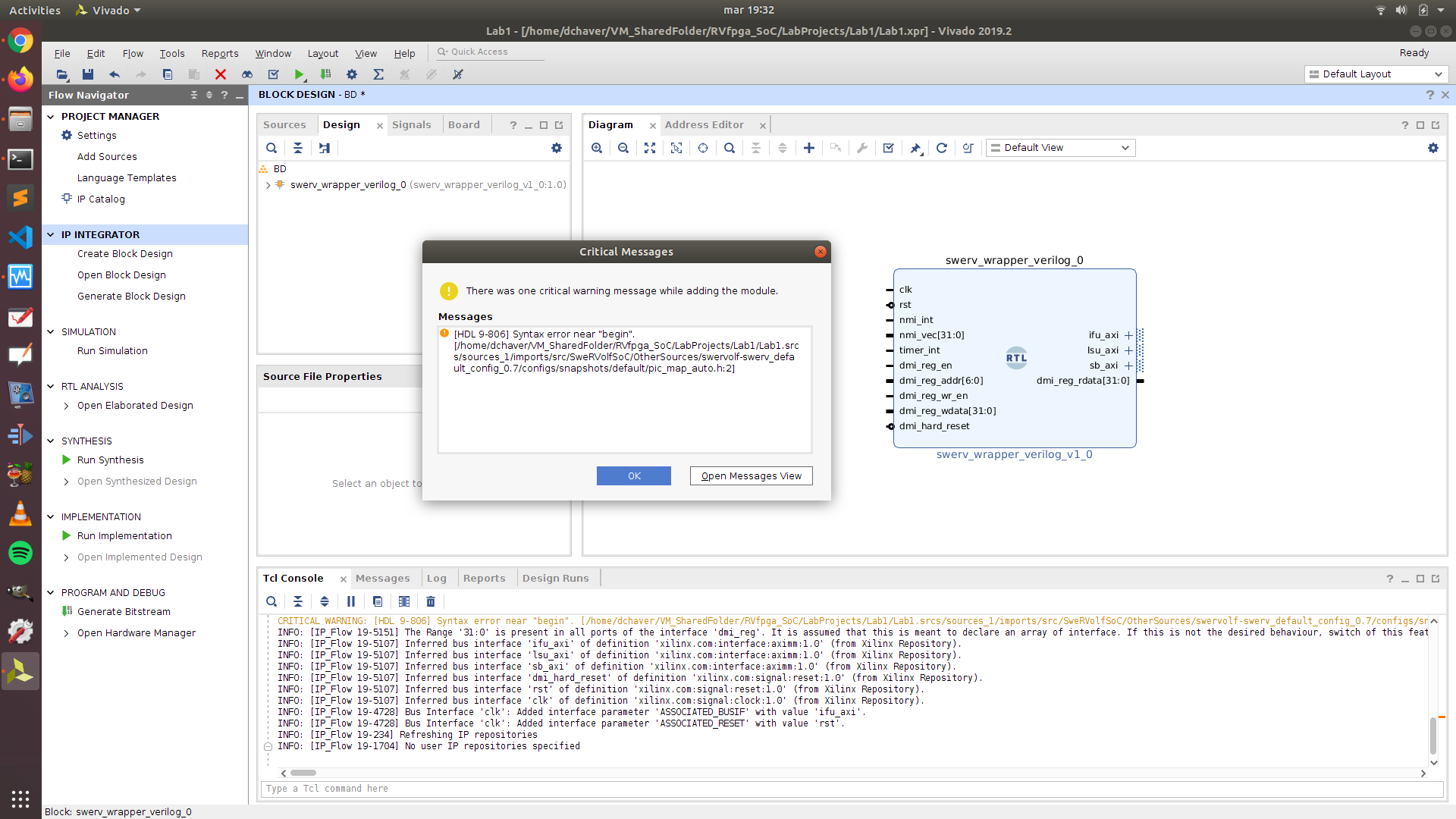
將出現一個對話方塊；您可以向下捲動找到需要新增的模組或在搜尋方塊中輸入該模組的名稱。我們首先新增SweRV EH1核心組合。

選擇**「swerv\_wrapper\_verilog」**並按一下「OK」（確定）。



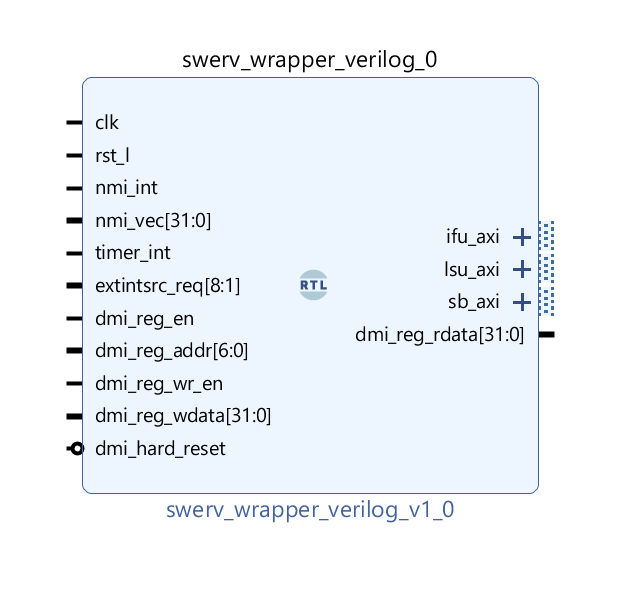
**圖27. 新增swerv\_wrapper\_verilog**

將彈出一條嚴重警告消息（請參閱圖28）。按一下「OK」（確定）忽略此警告消息。



**圖28. 嚴重警告消息**

新增模組後，我們可通過按一下模組上的「**+**」圖示查看並造訪“**ifu\_axi**”、“**lsu\_axi**”或“**sb\_axi**”的所有引腳。



**圖29.“swerv\_wrapper\_verilog”模組**

同理，我們現在將新增以下模組：

* **“intcon\_wrapper\_bd”**（互連包裝程式模組）：它是一個包裝程式模組，其中封裝了全部三個互連模組。



現在我們將新增SoC所需的週邊設備：

* **“bootrom\_wrapper”**（開機ROM模組）



* **“gpio\_wrapper”**（GPIO頂層模組）



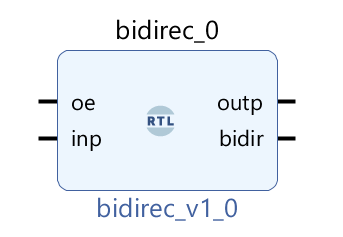
* **“syscon\_wrapper”**（系統控制器模組）



我們將新增32個“**bidirec”**模組，以便與GPIO模組連接。其中16個用於LED，另外16個用於開關。

* **“bidirec”**（雙向GPIO模組）

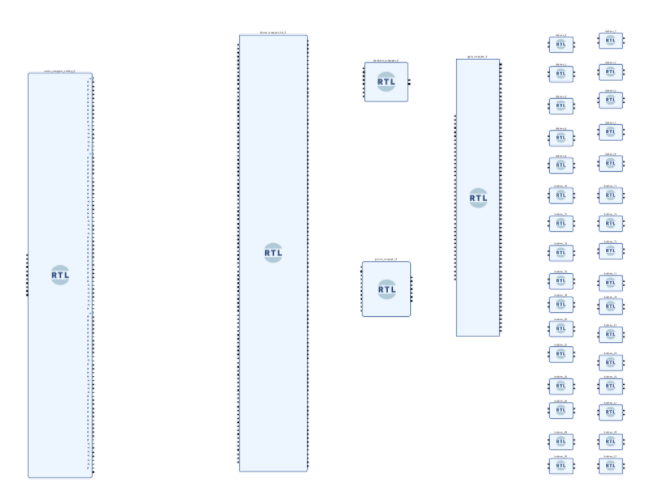




**圖30. bidir GPIO模組**

同理，我們將在區塊設計中新增這32個模組。

複製-貼上圖中的區塊可以快速新增這32個模組。首先複製1個“**bidirec**”區塊並貼上，然後複製2個區塊並貼上，接著重複複製和貼上過程，直到將32個“bidirec”模組新增到區塊設計中為止。



**圖31. 所需的模組已新增到區塊設計中**

（請參閱圖31）從左側開始，首先是**SweRV核心**模組（swerv\_wrapper\_verilog\_0）；緊接著右側是**互連包裝程式**模組（intcon\_wrapper\_bd\_0）和三個週邊設備模組，分別是**開機ROM**（bootrom\_wrapper\_0）模組、**系統控制器**（syscon\_wrapper\_0）模組和**GPIO**（gpio\_wrapper\_0）模組。最右側是32個**Bidirec**（bidirec\_x）模組。

**步驟4. 模組佈線**

現在將這些模組逐引腳或（在某些情況下）逐匯流排相互連接。我們將開始連接“**swerv\_wrapper\_verilog**”與“**intcon\_wrapper\_bd**”。在這些模組之間需要連接三組不同的引腳，分別與核心的以下子模組相關：

* **IFU**（取指單元）
* **LSU**（載入儲存單元）
* **SB**（儲存位元組）

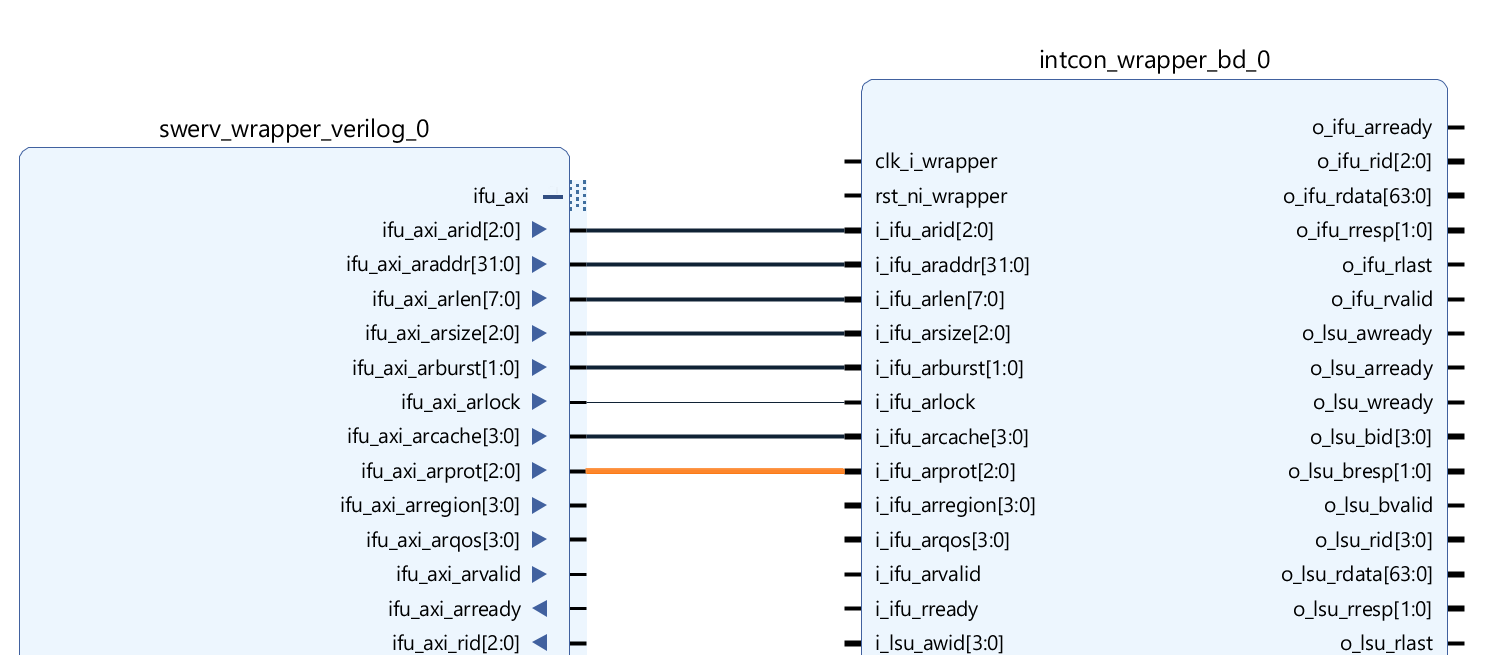
我們將首先連接與IFU相關的引腳。將“**swerv\_wrapper\_verilog**”模組的“*ifu\_axi\_arid[2:0]*”引腳連接到“**intcon\_wrapper\_bd**”的“*i\_ifu\_arid[2:0]*引腳。

同理，

*ifu\_axi\_araddr[31:0]*將連接到*i\_ifu\_araddr[31:0]，*

*ifu\_axi\_arlen[7:0]*將連接到*i\_ifu\_arlen[7:0]，*

*ifu\_axi\_arsize[2:0]*將連接到*i\_ifu\_arsize[2:0]*，依此類推（請參閱圖32）。



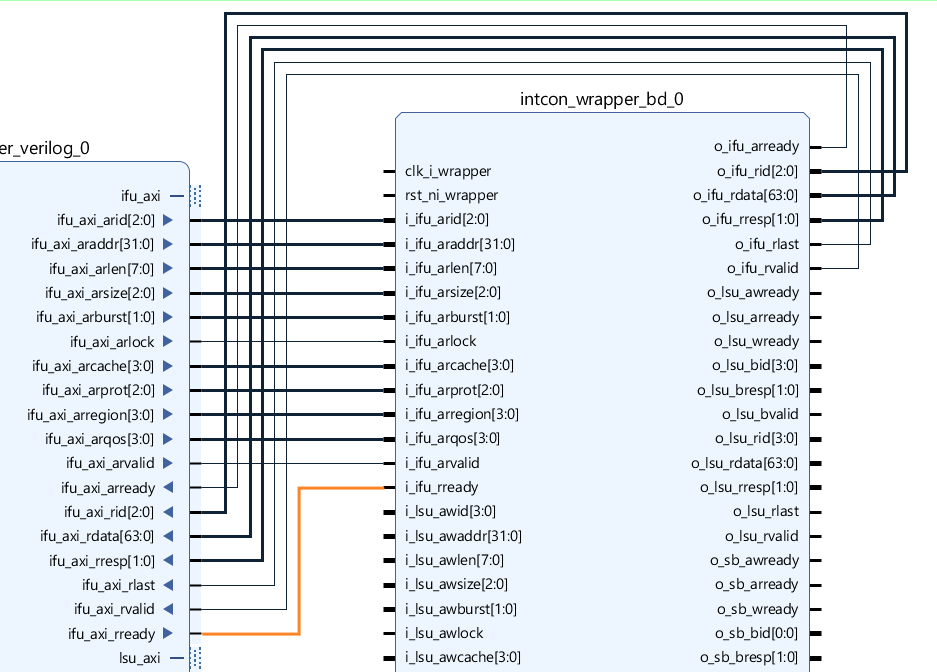
**圖32. 連接相關引腳**

同理，我們會將“**swerv\_wrapper\_verilog**”的所有**IFU（取指單元）**引腳與“**intcon\_wrapper\_bd**”的**IFU**引腳連接（請參閱圖33）。

**PDF：**如需查看區塊設計的接線細節特寫，可從此處獲取高品質PDF檔案：

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

/InternalConnections/1\_SwervW\_IntconW\_IFU.pdf



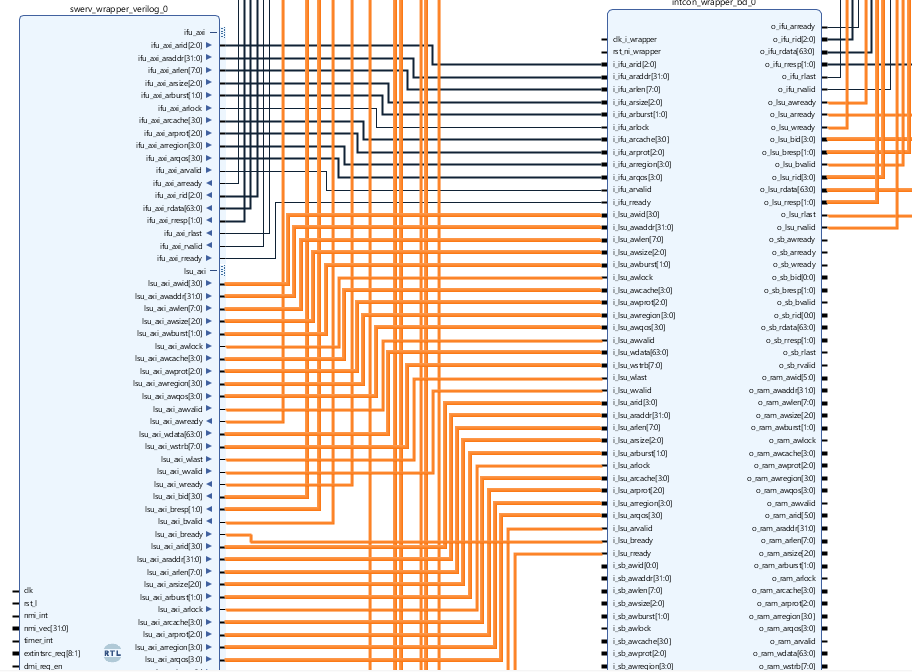
**圖33. 連接所有IFU引腳**

現在我們繼續將“**swerv\_wrapper\_verilog**”的所有**LSU（載入儲存單元）**引腳連接到“**intcon\_wrapper\_bd**”的**LSU**引腳。執行過程與**IFU**引腳相同，將“**swerv\_wrapper\_verilog**”模組的每個**LSU**引腳與其在“**intcon\_wrapper\_bd**”模組上對應的引腳連接（請參閱圖34）。

**PDF：**如需查看區塊設計的接線細節特寫，可從此處獲取高品質PDF檔案：

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

/InternalConnections/2\_SwervW\_IntconW\_LSU.pdf



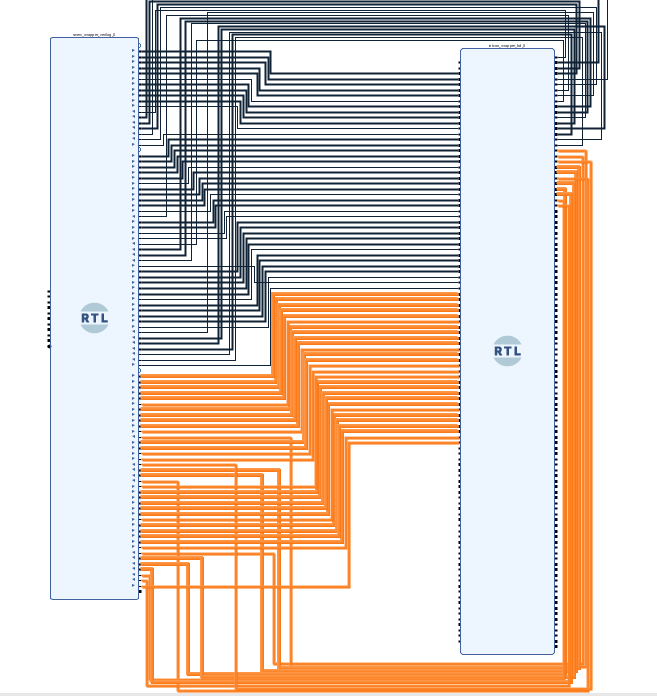
**圖34. 連接所有LSU引腳**

現在我們繼續連接**SB**引腳。同理，我們將“**swerv\_wrapper\_verilog**”的所有**SB**引腳與其在“**intcon\_wrapper\_bd**”上對應的**SB**引腳連接（請參閱圖35）。

**PDF：**如需查看區塊設計的接線細節特寫，可從此處獲取高品質PDF檔案：

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

/InternalConnections/3\_SwervW\_IntconW\_SB.pdf



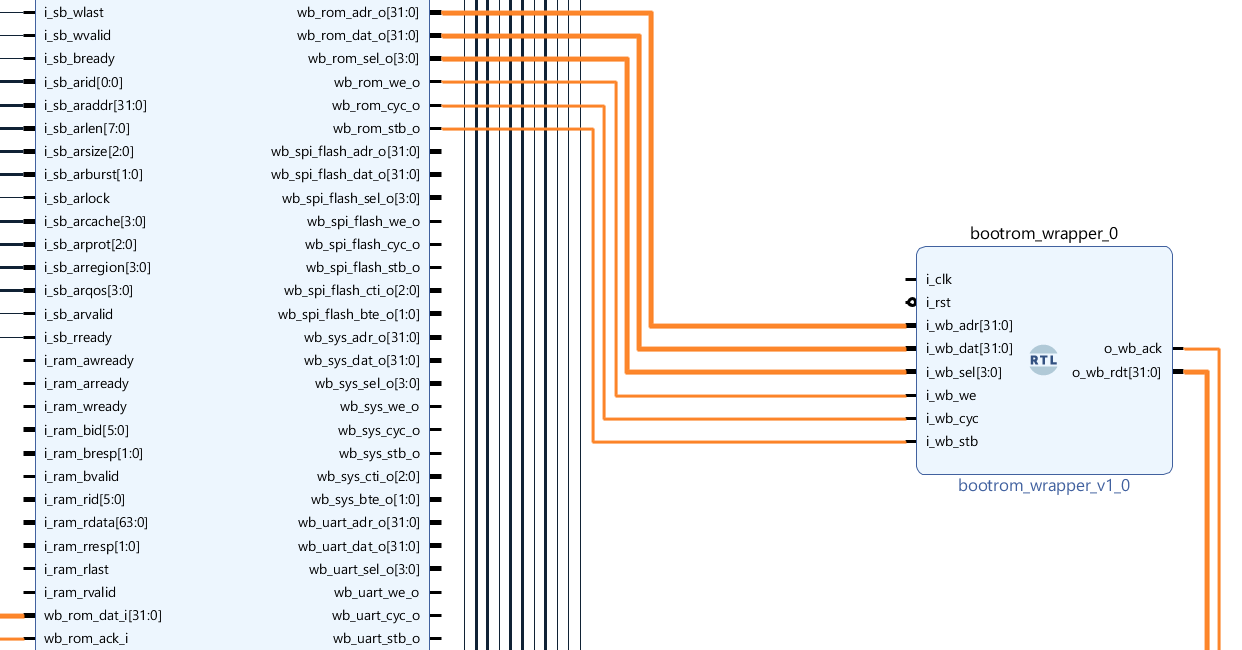
**圖35. 連接所有SB引腳**

接下來，我們會將週邊設備與“**Intcon\_wrapper\_bd**”連接。我們從“**bootrom\_wrapper**”模組開始，方法是連接“**Intcon\_wrapper\_bd**”的“wb\_rom\_xxx\_x”線（請參閱圖36）。

**PDF：**如需查看區塊設計的接線細節特寫，可從此處獲取高品質PDF檔案：

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

/InternalConnections/4\_BootRomW\_IntconW.pdf



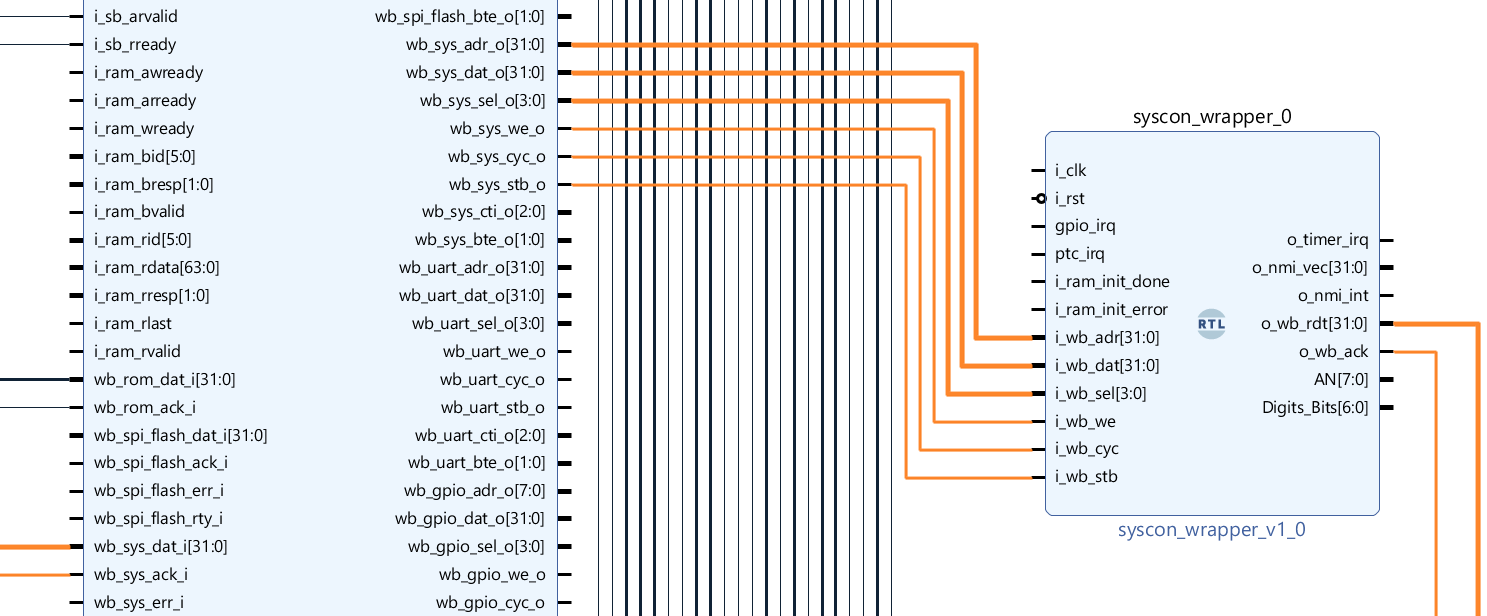
**圖36. 將BootROM模組與互連包裝程式模組連接**

現在我們會將“**syscon\_wrapper**”模組與“**Intcon\_wrapper\_bd**”模組連接（請參閱圖37）。

**PDF：**如需查看區塊設計的接線細節特寫，可從此處獲取高品質PDF檔案：

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

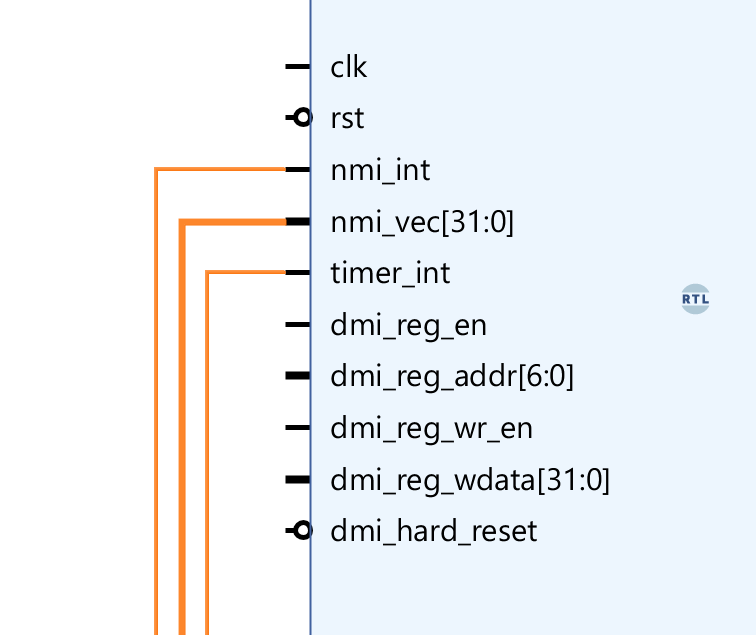
/InternalConnections/5\_SysconW\_IntconW.pdf

****

**圖37. 將Syscon與WB互連引腳連接**

“**syscon\_wrapper**”的以下引腳將連接到“**swerv\_wrapper\_verilog**”（請參閱圖38）。

* o\_timer\_irq
* o\_nmi\_vec[31:0]
* o\_nmi\_int

****

**圖38. 將syscon\_wrapper與swerv\_wrapper\_verilog引腳連接**

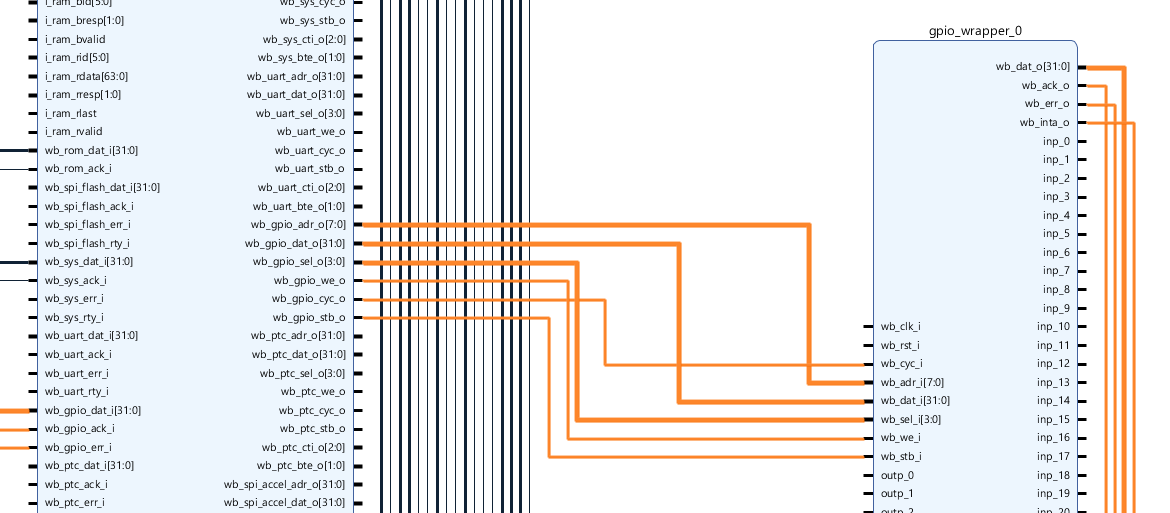
現在我們會將“**gpio\_wrapper**”模組與“**intcon\_wrapper\_bd**”連接。將“**intcon\_wrapper\_bd**”模組的“wb\_gpio\_xxx\_x”引腳與“**gpio\_wrapper**”模組的引腳連接（請參閱圖39）。

將“**gpio\_wrapper**”模組的“**wb\_inta\_o**”引腳與“**syscon\_wrapper**”模組的“**gpio\_irq**”引腳連接。

**PDF：**如需查看區塊設計的接線細節特寫，可從此處獲取高品質PDF檔案：

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

/InternalConnections/6\_GpioW\_IntconW.pdf

****

**圖39. 將gpio\_wrapper與intcon\_wrapper引腳連接**

我們會將32個GPIO“**bidirec**”模組與我們已連接的“**gpio\_wrapper**”模組連接。具體來說，我們會將“**gpio\_wrapper**”模組與“**bidirec\_x**”模組連接，其中x是從1到32的數字。連接方式如下：

“**gpio\_wrapper\_0”**的*“****inp\_0****”*引腳將連接到**“bidirec\_0”的“*inp”***，

*“****gpio\_wrapper\_0****”的“****inp\_1****”*引腳將連接到“**bidirec\_1**”的**“*inp”***，依次類推一直到最後一個“***inp***”連接，即“**gpio\_wrapper**”的“**inp\_31**”將連接到“**bidirec\_31**”的“***inp***”。

同理，

*“****gpio\_wrapper\_0”****的“****oe\_0****”*引腳將連接到“**bidirec\_0**”的**“*oe”***，

*“****gpio\_wrapper\_0****”的“****oe\_1****”*引腳將連接到“**bidirec\_1**”的**“*oe”***，依次類推一直到最後一個“***oe***”連接，即“**gpio\_wrapper**”的“**oe\_31**”將連接到“**bidirec\_31**”的“***oe***”。

同理，

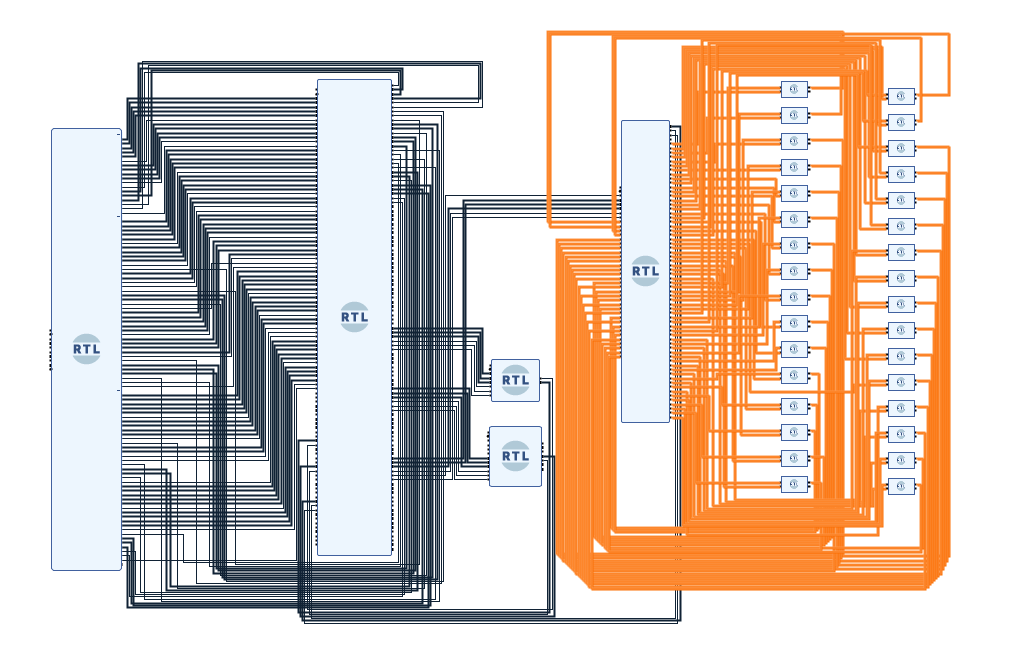
“**gpio\_wrapper\_0”**的*“****outp\_0****”*引腳將連接到**“bidirec\_0”的“*outp”***，

*“****gpio\_wrapper\_0****”的“****outp\_1****”*引腳將連接到“**bidirec\_1**”的**“*outp”***，依次類推一直到最後一個“***outp***”連接，即“**gpio\_wrapper**”的“**outp\_31**”將連接到“**bidirec\_31**”的“***outp***”。

**PDF：**如需查看區塊設計的接線細節特寫，可從此處獲取高品質PDF檔案：

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

/InternalConnections/7\_GpioW\_32xBidirec.pdf

****

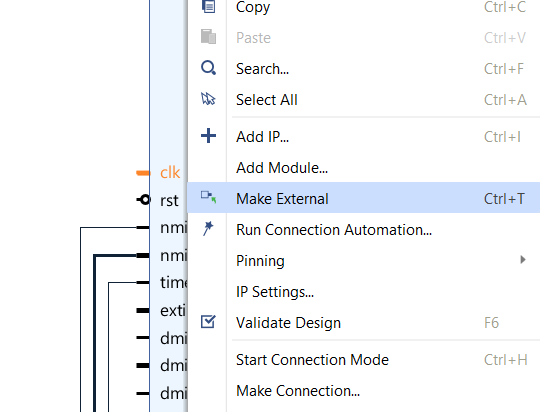
**圖40. 所有GPIO Bidirec模組均連接到gpio\_wrapper模組**

我們已完成模組之間的所有內部連接，現在將設定外部連接。

**步驟5. 為I/O引腳設定外部連接**

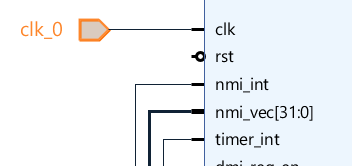
現在需要連接區塊設計的輸入/輸出引腳。我們會將這些引腳作為外部引腳/連接埠進行連接。這些外部引腳包括**RAM**（DDR）、**CLK**（時鐘）、**RST**（重設）和**DMI**（偵錯模組介面）的引腳。

我們首先連接“**clk**”引腳。移至「**swerv\_wrapper\_verilog**」模組，右鍵按一下「**clk」**引腳，隨後將顯示一個下拉功能表（請參閱圖41）。從所有下拉選項中選擇「Make External」（設定為外部）選項。也可以按一下引腳，使用快捷鍵「Ctrl + T」將引腳設定為外部。

****

**圖41. 將“clk”設定為外部連接**

您現在將看到“**swerv\_wrapper\_verilog**”的“**clk**”引腳已連接到外部引腳“**clk\_0**”（請參閱圖42）。

****

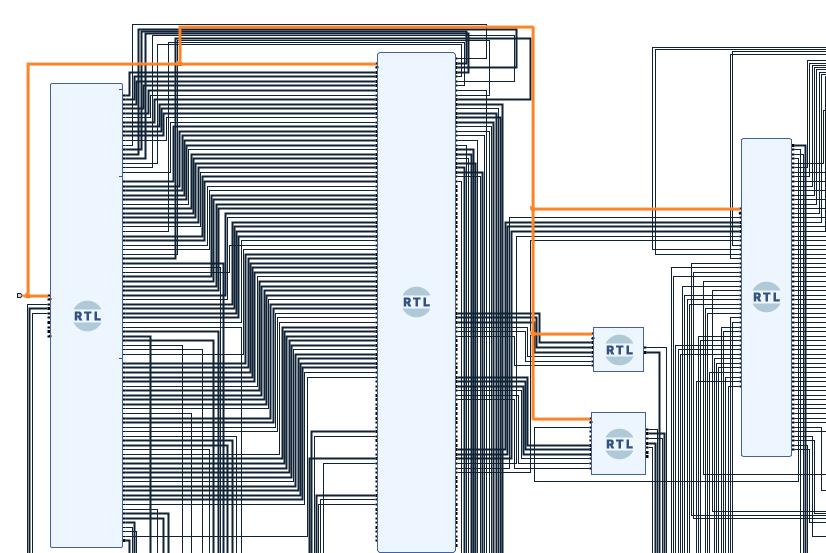
**圖42.“clk”成為外部連接**

現在，我們可以將**“clk”**外部引腳連接到其餘模組，包括intcon\_wrapper\_bd、syscon\_wrapper、bootrom\_wrapper和gpio\_wrapper。

**PDF：**如需查看區塊設計的接線細節特寫，可從此處獲取高品質PDF檔案：

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

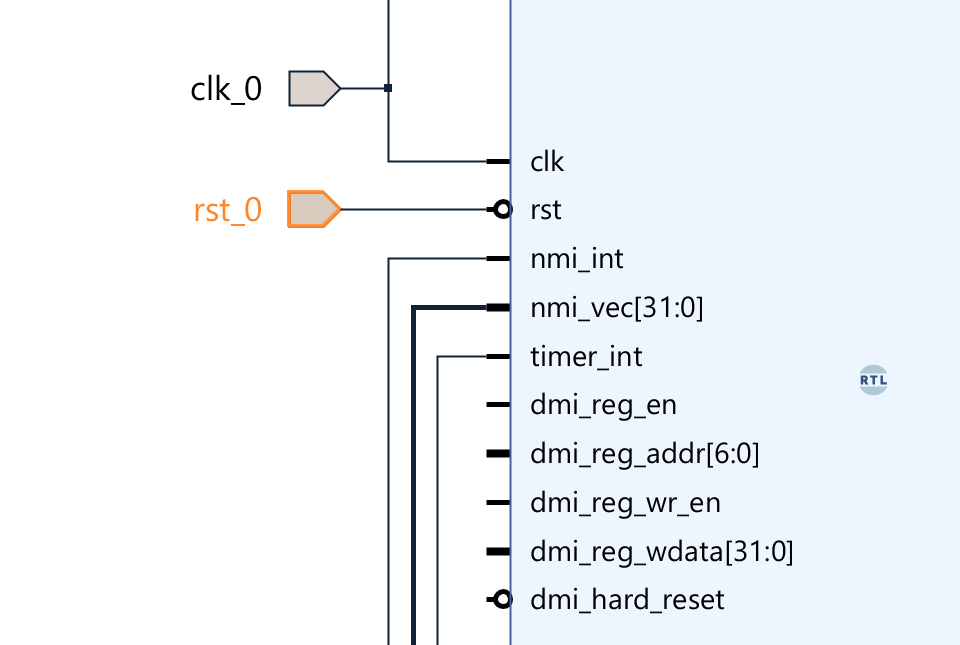
/ExternalConnections/1\_Clock.pdf



**圖43. 訊號clk連接到所有模組**

同理，我們可以將**“rst”**引腳連接到所有模組。

與我們為“**clk**”建立的外部引腳一樣，我們將為“**rst**”建立一個外部引腳。再次移至「**swerv\_wrapper\_verilog**」模組，右鍵按一下「**rst**」引腳，將其設定為外部。

****

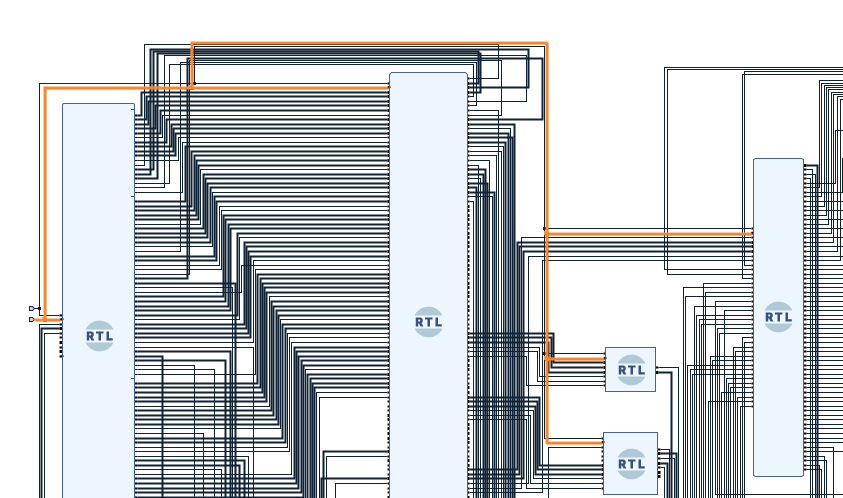
**圖44. 將rst\_l設定為外部引腳**

現在我們會將**“rst\_0”**外部引腳連接到其餘模組，包括intcon\_wrapper、syscon\_wrapper、bootrom\_wrapper和gpio\_wrapper。

**PDF：**如需查看區塊設計的接線細節特寫，可從此處獲取高品質PDF檔案：

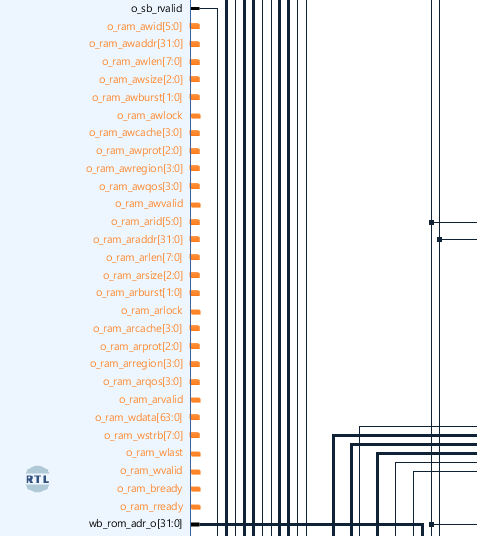
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

/ExternalConnections/2\_Reset.pdf

****

**圖45. 將反相“rst\_0”引腳與其餘模組連接**

現在，我們將完成以下步驟，將“**Intcon\_wrapper\_bd**”模組的所有RAM（DDR）引腳連接到外部RAM引腳。



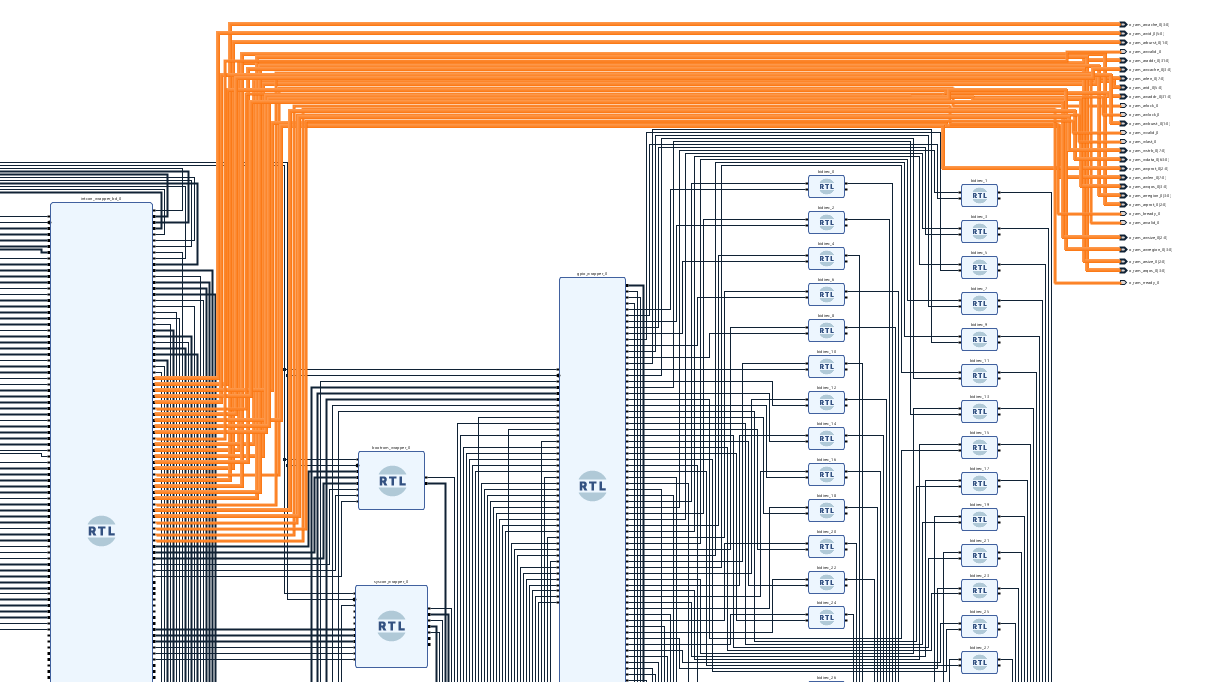
**圖46. Intcon包裝程式右側RAM引腳**

現在我們會將“**Intcon\_wrapper\_bd**”模組中的所有右側RAM引腳設定為外部引腳（請參閱圖47）。

**PDF：**如需查看區塊設計的接線細節特寫，可從此處獲取高品質PDF檔案：

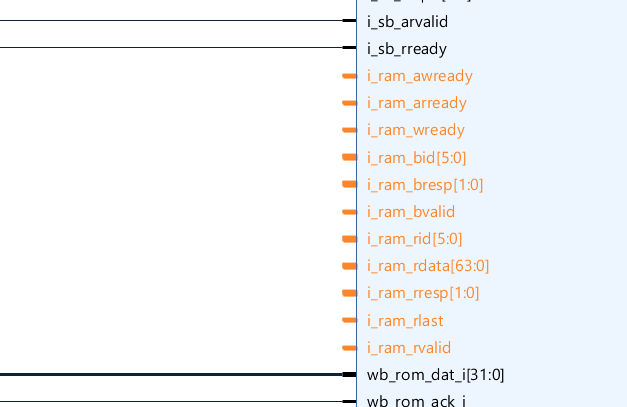
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

/ExternalConnections/3\_RAM\_R.pdf



**圖47. 將所有右側RAM引腳設定為外部**

現在我們會將“**Intcon\_wrapper\_bd**”的左側RAM引腳設定為外部引腳。



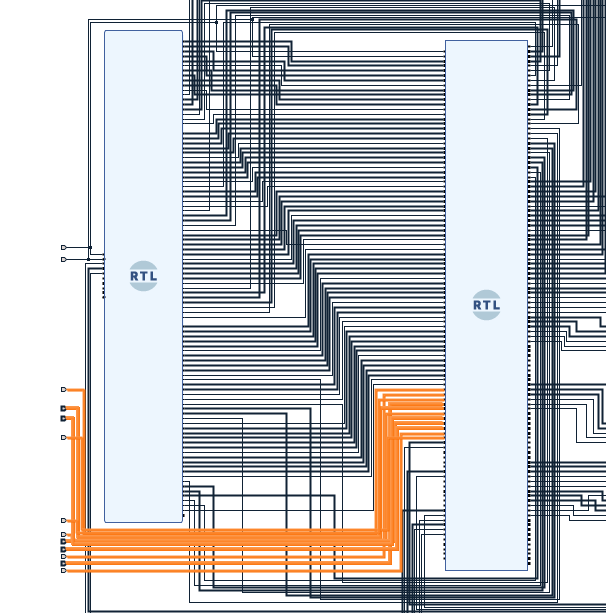
**圖48. 互連包裝程式的左側RAM引腳**

我們會將所有這些RAM引腳設定為外部引腳（請參閱圖49）。

**PDF：**如需查看區塊設計的接線細節特寫，可從此處獲取高品質PDF檔案：

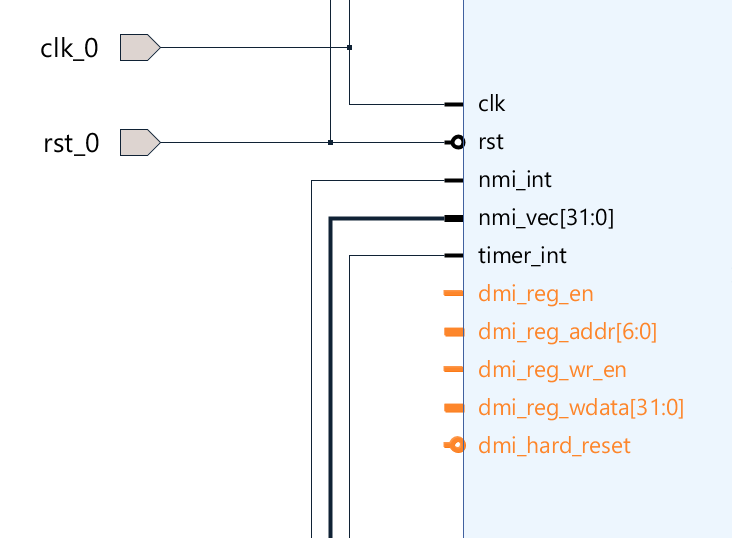
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

/ExternalConnections/4\_RAM\_L.pdf



**圖49. 將所有左側RAM引腳設定為外部**

現在我們會將**“swerv\_wrapper\_verilog”**模組的**DMI**引腳與外部引腳連接。

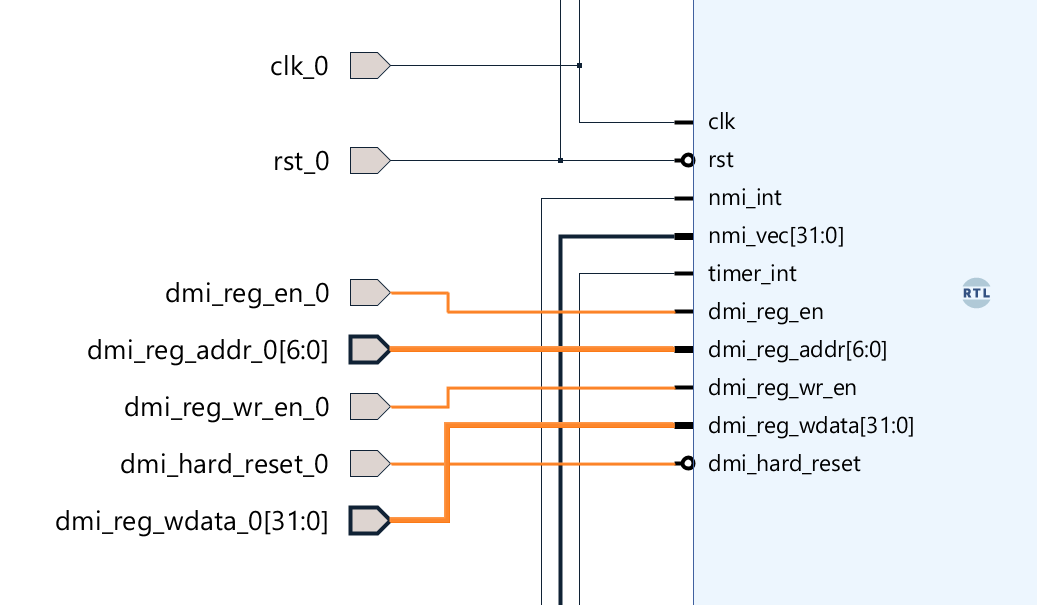
****

**圖50. swerv\_wrapper\_verilog的dmi引腳（左側）**

**PDF：**如需查看區塊設計的接線細節特寫，可從此處獲取高品質PDF檔案：

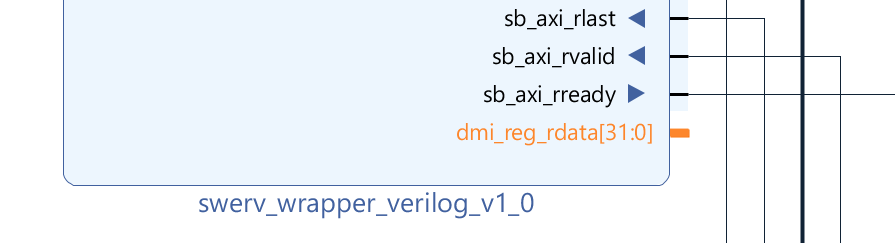
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

/ExternalConnections/5\_DMI.pdf



**圖51. 將dmi引腳設定為外部引腳**

我們會將另一個引腳與“**swerv\_wrapper\_verilog**”模組右下方的外部引腳連接。該引腳為“dmi\_reg\_rdata[31:0]”。



**圖52.“dmi\_reg\_rdata[31:0]”引腳（swerv\_wrapper\_verilog右側）**

我們會將“**dmi\_reg\_rdata[31:0]**”也設定為外部引腳。



**圖53. 將“dmi\_reg\_rdata[31:0]”引腳設定為外部引腳**

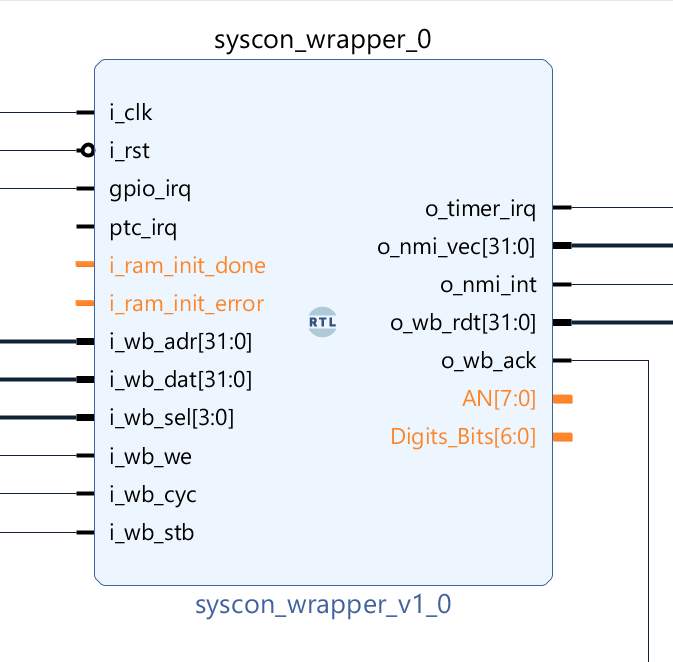
我們現在會將“**syscon\_wrapper**”模組的以下引腳設定為外部引腳。

* i\_ram\_init\_done
* i\_ram\_init\_error
* AN[7:0]
* Digital\_Bits[6:0]

**PDF：**如需查看區塊設計的接線細節特寫，可從此處獲取高品質PDF檔案：

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

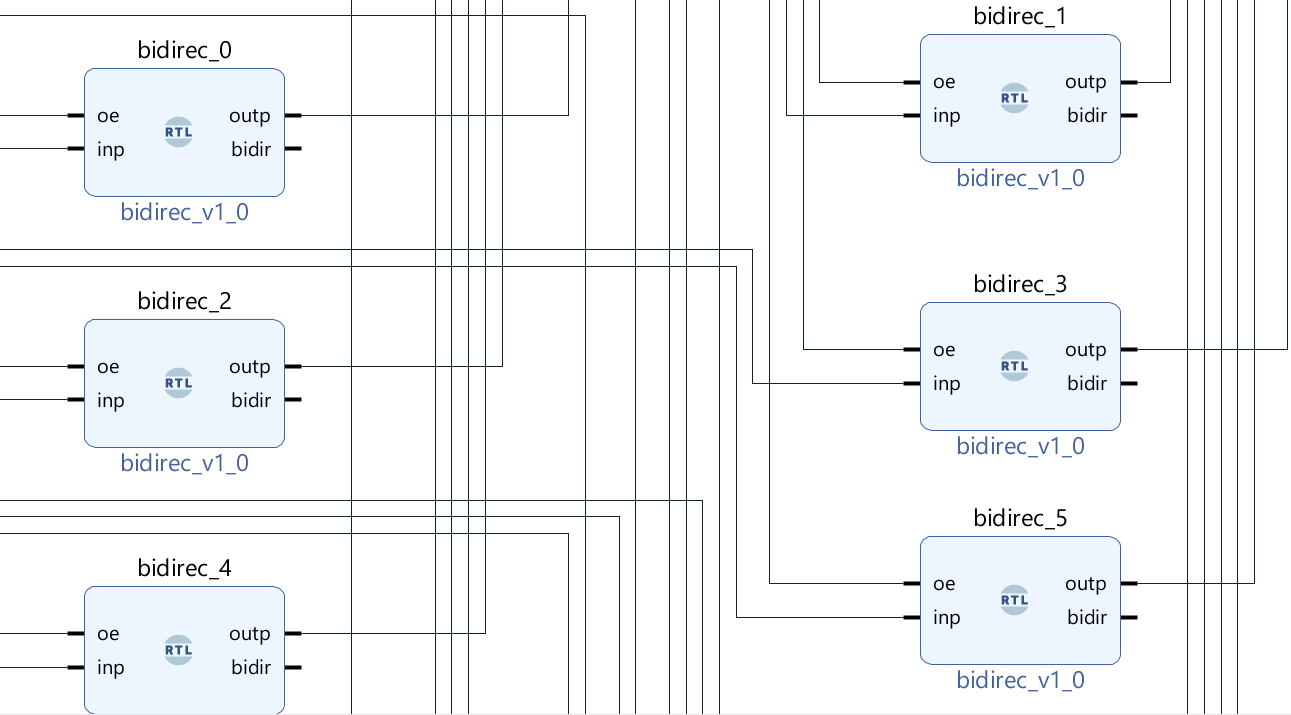
/ExternalConnections/6\_SysconW\_External.pdf



**圖54. syscon\_wrapper的外部引腳**

剩下的最後一個連接會將所有“**bidirec**”模組的所有“bidir”引腳設定為外部引腳。

|  |
| --- |
| **附註：**從“**bidirec\_0**”模組開始，將這些連接逐個設定為外部連接  因此，“**bidirec\_0**”模組的“**bidir**”引腳將連接到外部引腳  “**bidir\_0**”。然後連接“**bidirec\_1**”的“**bidir**”引腳，依此類推。 |

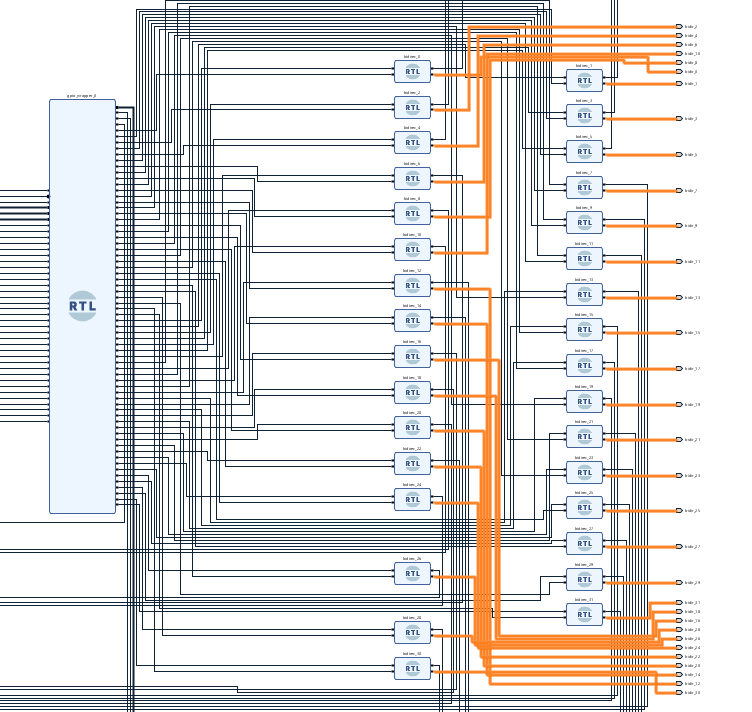


**圖55. 將GPIO Bidirec模組的“bidir”引腳設定為外部引腳**

**PDF：**如需查看區塊設計的接線細節特寫，可從此處獲取高品質PDF檔案：

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

/ExternalConnections/7\_Bidir.pdf

****

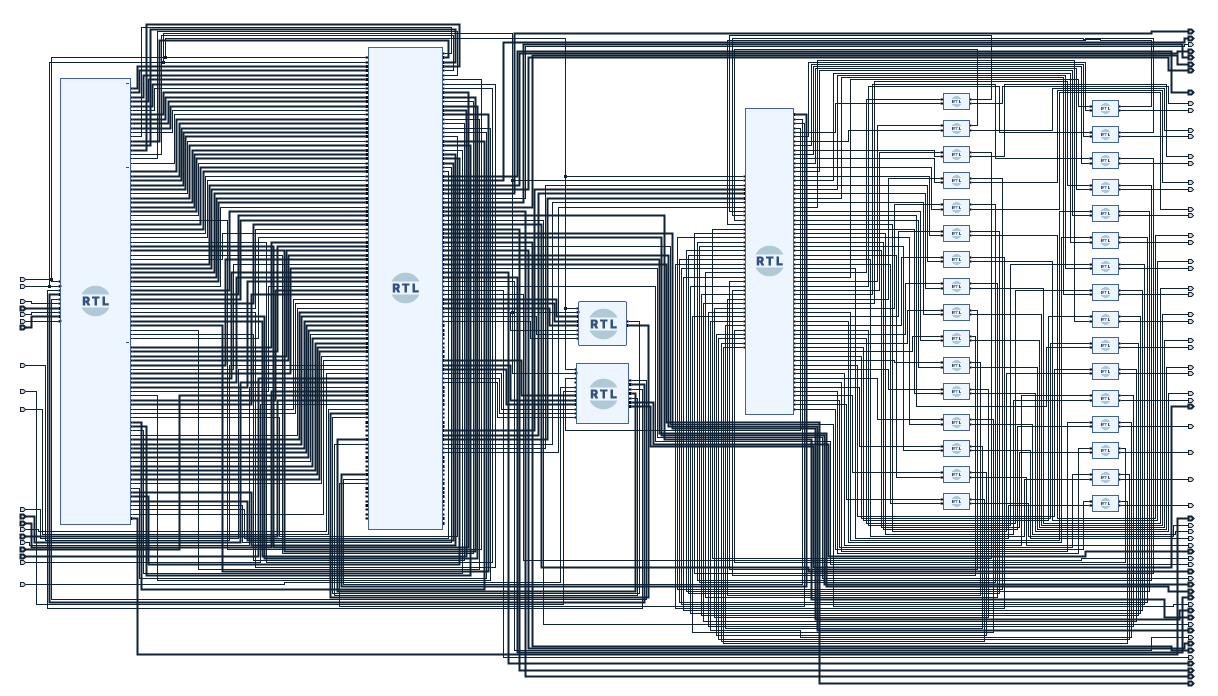
**圖56. 設定“bidir”外部連接**

現在我們已完成區塊設計SoC的所有內部和外部連接。按下「**Ctrl + S**」儲存區塊設計。

我們的區塊設計是在SweRVolfX SoC完成後建模的，現在包含以下連接模組：

* 1個SweRV核心 （swerv\_wrapper\_verilog）
* 1個互連包裝程式 （intcon\_wrapper\_bd）
* 1個開機ROM （bootrom\_wrapper）
* 1個GPIO頂層模組 （gpio\_wrapper）
* 1個系統控制器 （syscon\_wrapper）
* 32個GPIO模組 （bidirec）

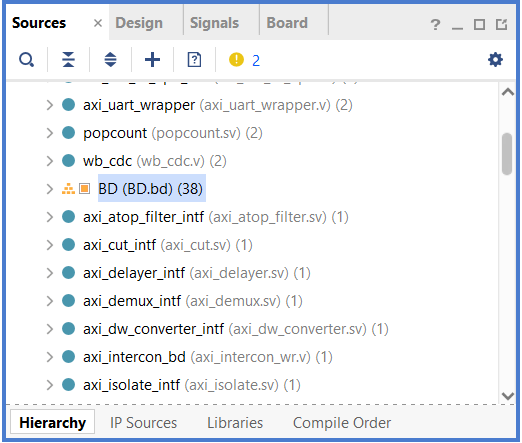
（請參閱圖57）。

**圖57. 區塊設計SoC完成**

# 產生區塊設計模組Verilog檔案

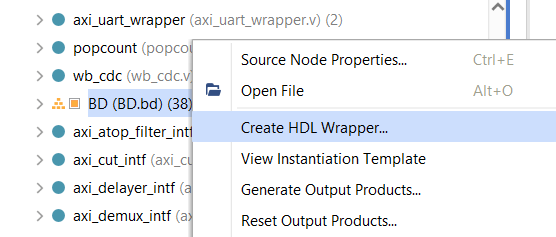
我們現在將產生已建立區塊設計的Verilog模組檔。

**步驟1.** 導覽到原始程式碼面板，找到剛剛建立的區塊設計模組“**BD**”。

****

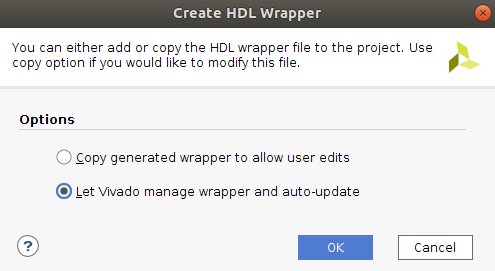
**圖58. 在「Sources」（原始程式碼）中找到「BD」**

**步驟2.**現在右鍵按一下該區塊設計（BD），然後選擇「**Create HDL Wrapper**」（建立HDL包裝程式）（請參閱圖59）。

****

**圖59. 建立HDL包裝程式**

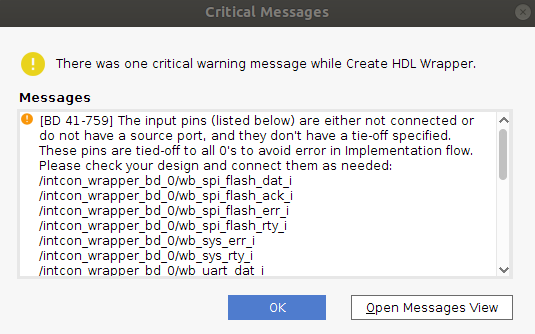
**步驟3.**選擇「**Let Vivado manage wrapper and auto-update**」（由Vivado管理包裝程式並自動更新）選項，然後按一下「OK」（確定）繼續操作。

****

**圖60. 選擇第二個選項**

您將看到嚴重警告快顯視窗，因為區塊設計中有幾個引腳未連接，這些引腳將自動連接到“**0**”（地）。

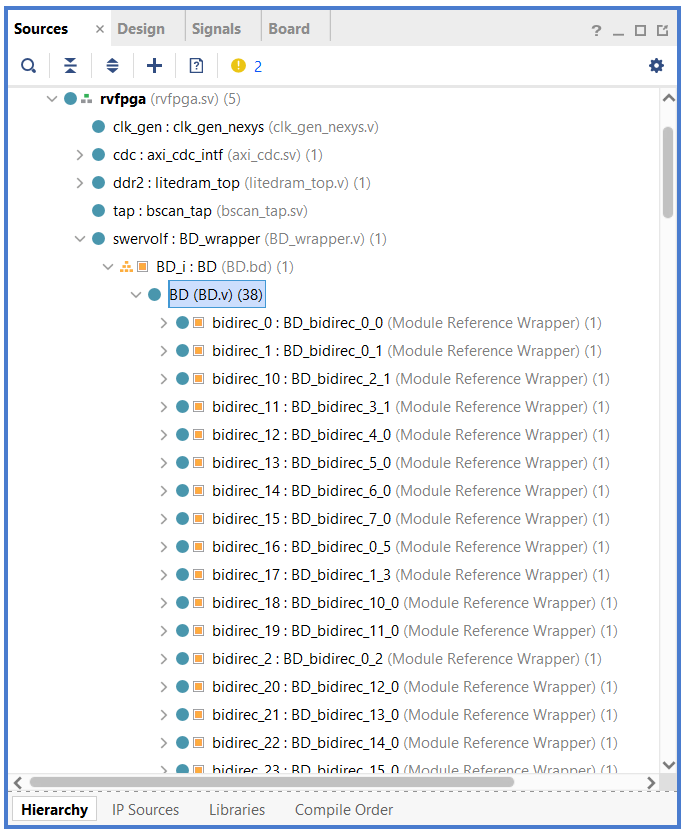
按一下「OK」（確定）。



**圖61. 警告快顯視窗**

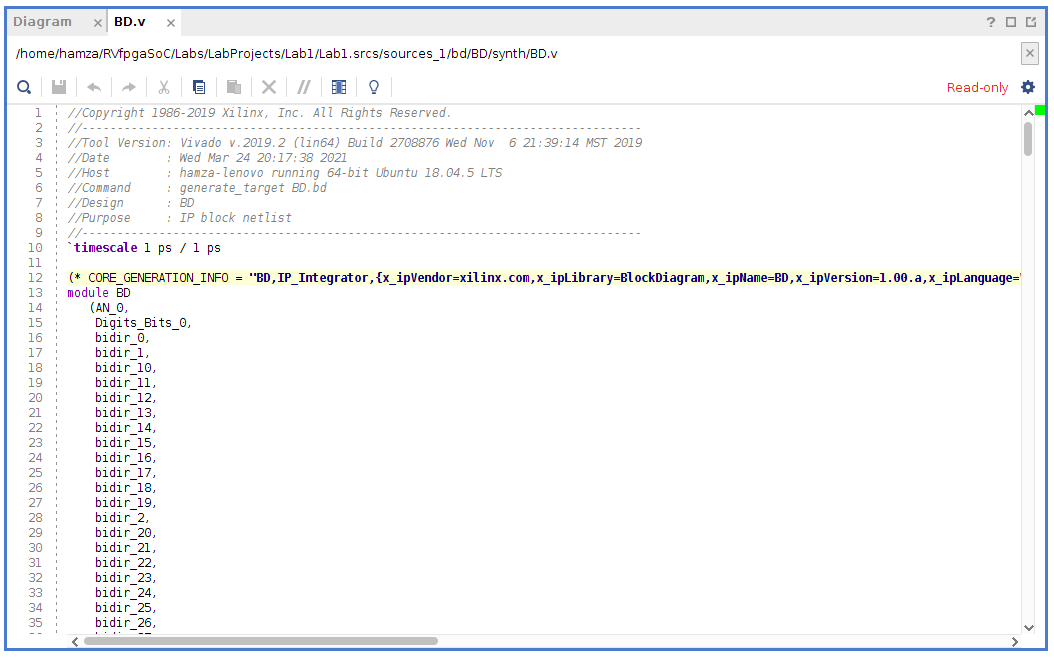
現在，區塊設計的HDL包裝程式已建立完畢。可以導覽到「Sources」（原始程式碼）面板並向下捲動找到「**BD\_wrapper**」。按一下它旁邊的下拉圖示，然後再按一下「**BD\_i**」旁邊的下拉圖示。

現在按兩下開啟「**BD (BD.v)**」檔（請參閱圖62）。

****

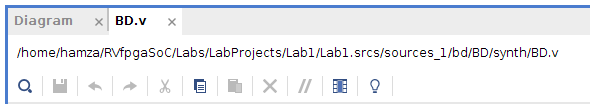
**圖62. 在原始程式碼面板中找到「BD.v」**

此處可以看到使用Vivado的區塊設計工具建立的“**BD.v”**Verilog檔。

****

**圖63.“BD.v”**

檔案頂部將顯示該新建檔案的路徑。在下一個實驗中，我們將使用該路徑造訪該“**BD.v**”檔。

****

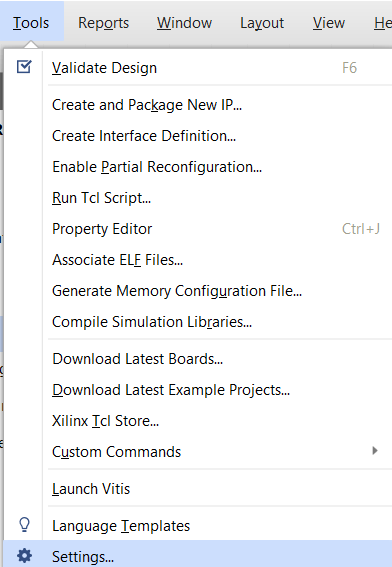
**圖64.“BD.v”檔的路徑**

# 產生位元串流

現在我們已經使用Vivado的區塊設計工具建立了SweRVolfX子集並產生了Verilog包裝程式，接著可以產生將用於配置FPGA的位元串流。要產生位元串流，我們首先需要完成以下步驟來調整Vivado中的一些設定。

**步驟1**. **導覽到「Settings」（設定）。**

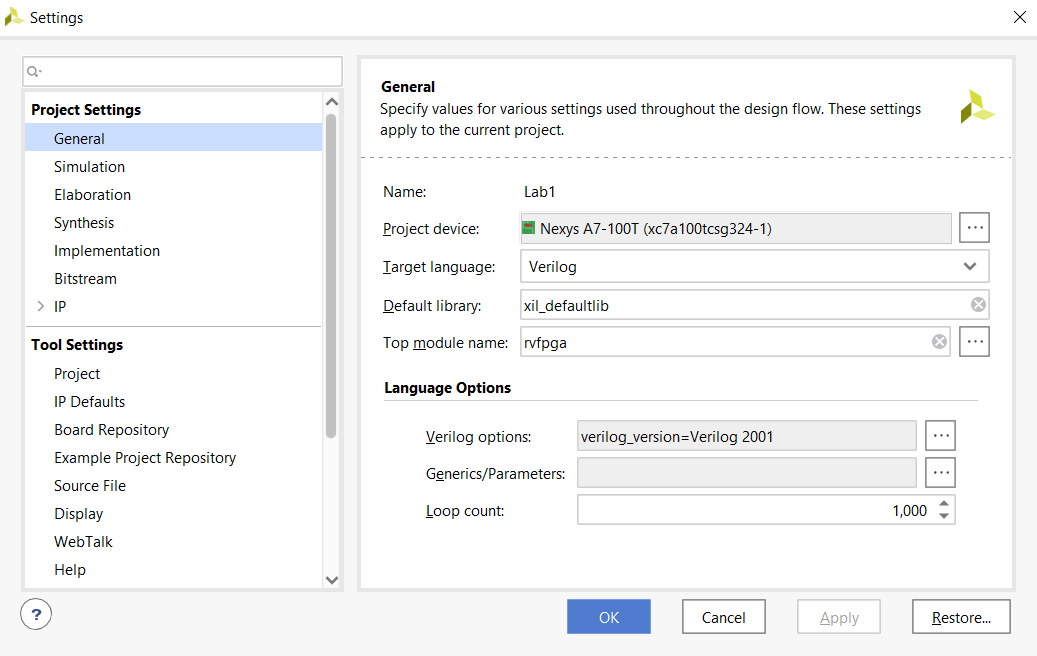
移至Vivado導覽欄左上方的「**Tools**」（工具），然後從選項中選擇「**Settings**」（設定）。



**圖65. 移至「Settings」（設定）**

**步驟2**. **導覽到「General」（一般）索引標籤**

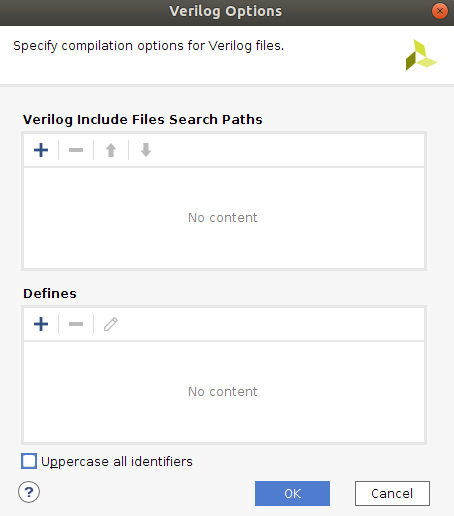
移至「**General**」（一般）索引標籤，然後從語言選項部分選擇「**Verilog options**」（Verilog選項）。

****

**圖66. 一般設定**

**步驟3**. **新增包含檔案的路徑。**

按一下「**+**」按鈕，新增Verilog**包含檔**搜尋路徑。

****

**圖67. Verilog選項**

現在新增以下三個路徑：

* [RVfpgaSoCPath]/RvfpgaSoC/Labs/LabProjects/Lab1/Lab1.srcs /sources\_1/imports/src/SweRVolfSoC/Interconnect

/AxiInterconnect/pulp-platform.org\_\_axi\_0.25.0/include

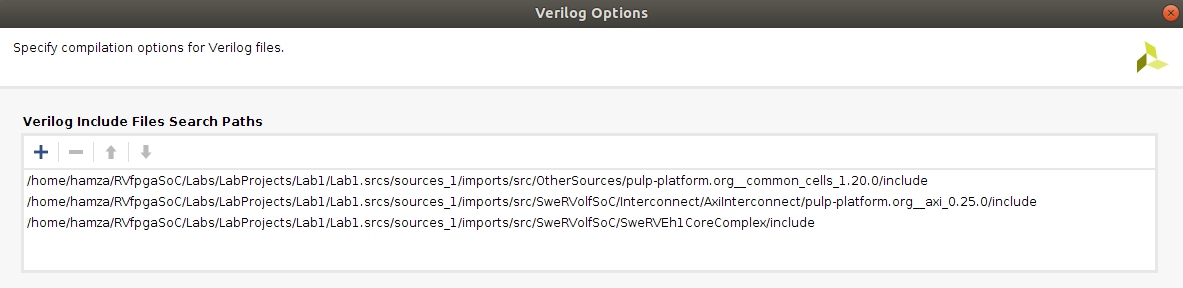
* [RVfpgaSoCPath]/RvfpgaSoC/Labs/LabProjects/Lab1/Lab1.srcs

/sources\_1/imports/src/OtherSources

/pulp-platform.org\_\_common\_cells\_1.20.0/include

* [RVfpgaSoCPath]/RvfpgaSoC/Labs/LabProjects/Lab1/Lab1.srcs

/sources\_1/imports/src/SweRVolfSoC/SweRVEh1CoreComplex/include

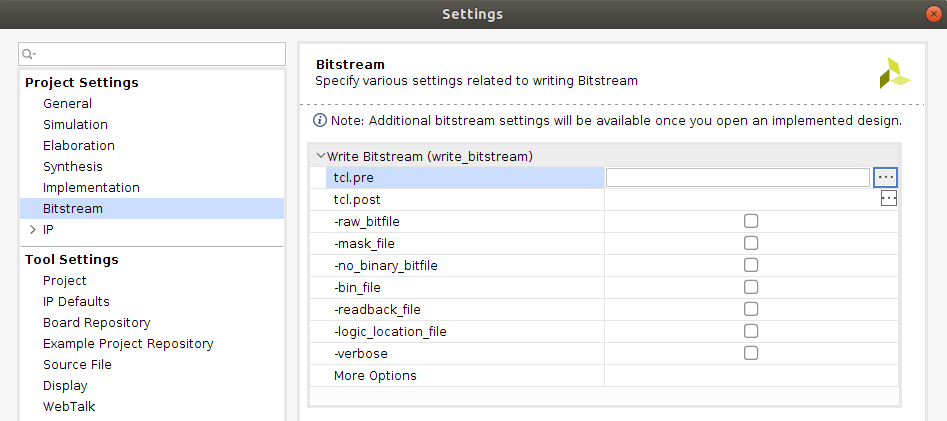
****

**圖68. Verilog包含檔路徑**

按一下「OK」（確定）。

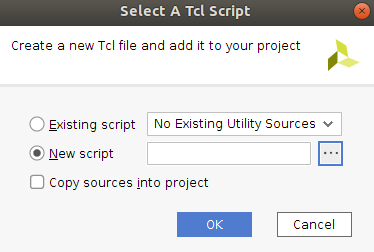
**步驟4**. **導覽到「Bitstream」（位元串流）索引標籤**

移至「**Bitstream**」（位元串流）索引標籤，然後按一下「**tcl.pre**」按鈕。



**圖69. 位元串流設定**

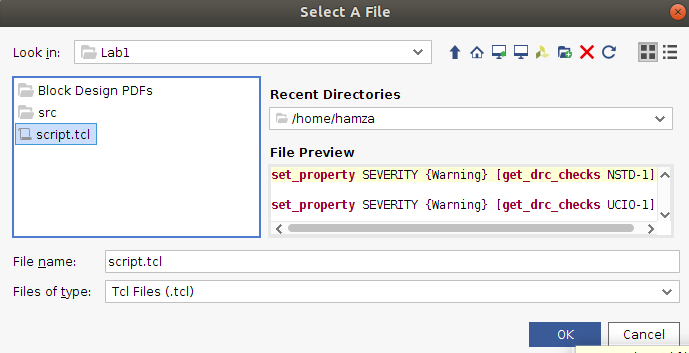
選擇「New script」（新建指令碼）選項。



**圖70. 新建Tcl指令碼**

導覽到以下路徑並選擇「script.tcl」檔。（請參閱圖71）

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/script.tcl

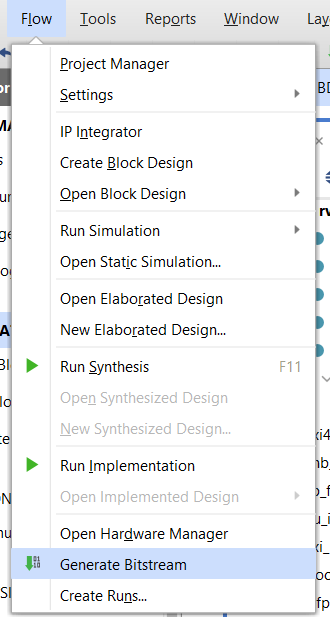


**圖72. 匯入「script.tcl」檔**

按一下「OK」（確定）並套用變更。

**步驟4**. **產生位元串流。**

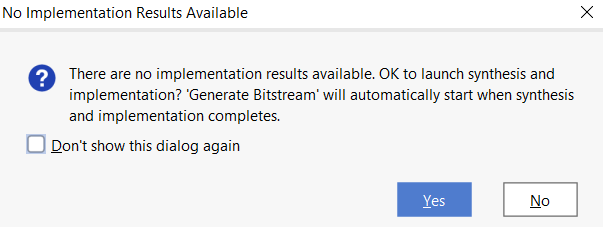
現在按一下「Flow → Generate Bitstream」（流動 → 產生位元串流），如圖73所示。



**圖73. 產生位元串流**

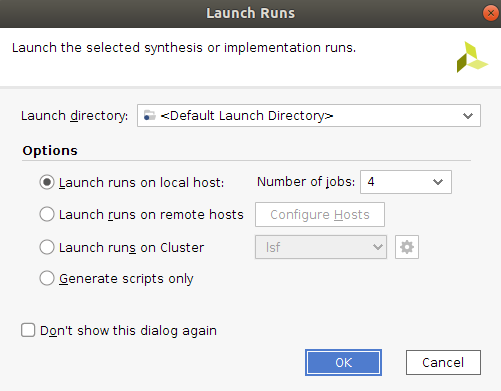
此時可能會彈出一個視窗，提示沒有可用的實作結果，並要求啟動合成和實作。

按一下「Yes」（是）（請參閱圖74）。



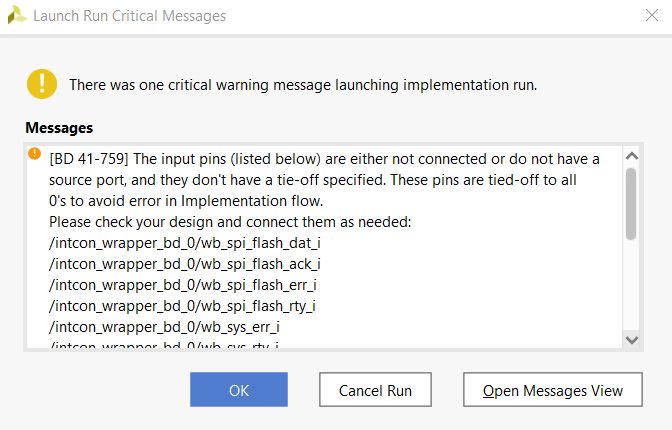
**圖74. 啟動合成和實作視窗**

螢幕上將彈出「**Launch Runs**」（啟動執行）視窗（請參閱圖75）。按一下「OK」（確定）。



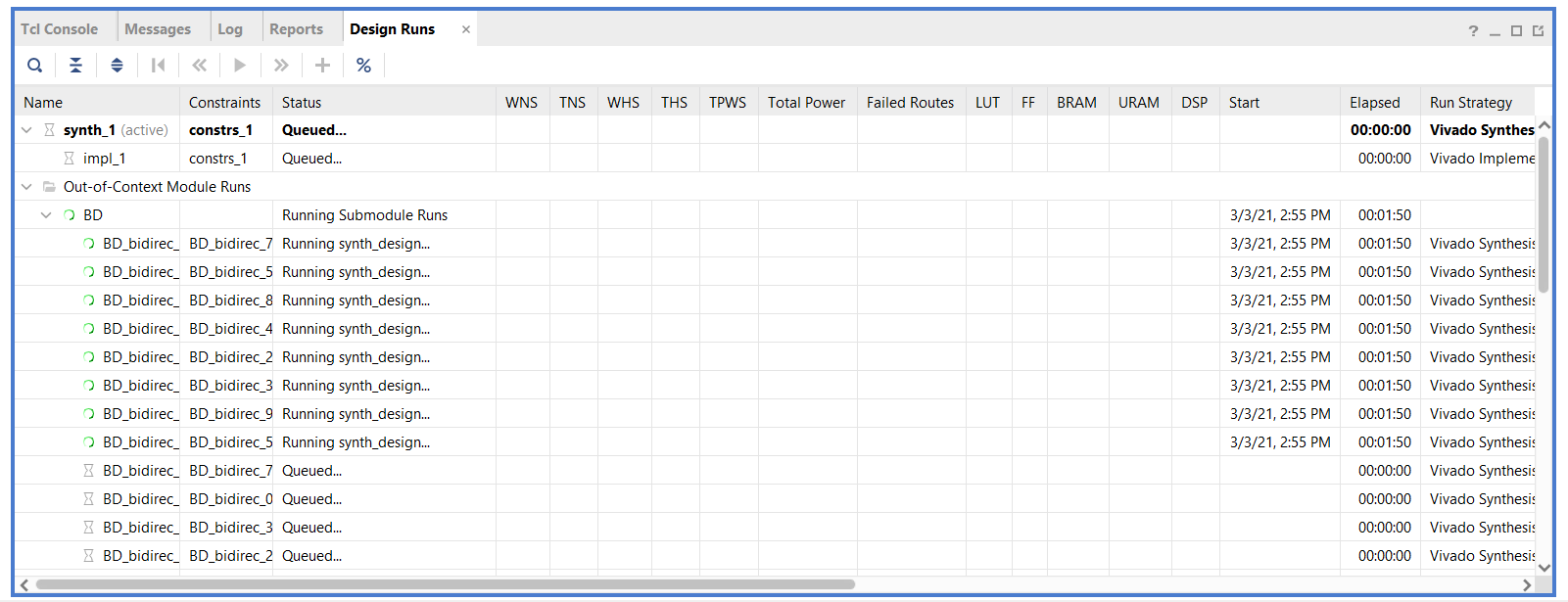
**圖75. 啟動執行**

現在我們將看到一個警告清單，提示未連接的引腳將自動連接到“**0**”。我們將按一下「OK」（確定）。（請參閱圖76）。



**圖76. 啟動執行警告消息**

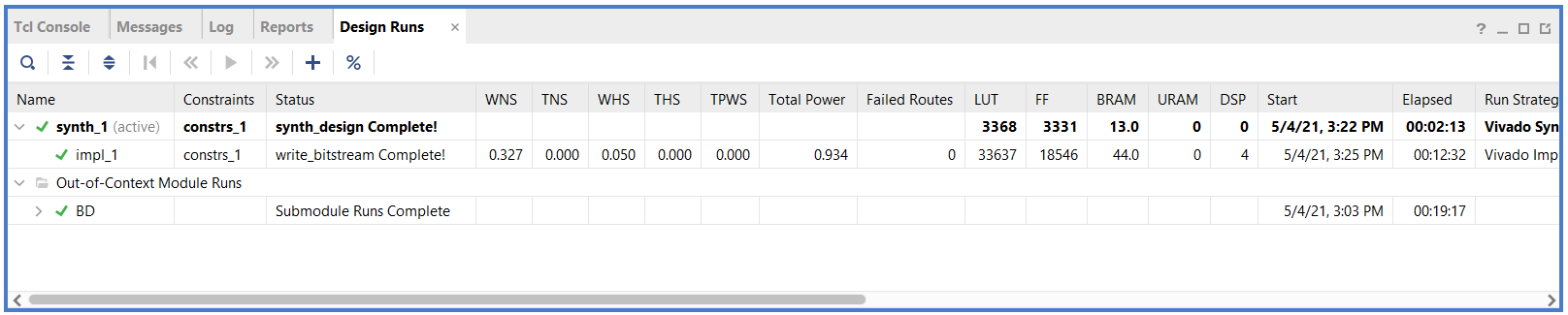
此步驟會對**RVfpgaNexys**進行合成（根據專案中Verilog和SystemVerilog檔案的定義）、將其映射到FPGA並建立位元串流。



**圖77. 設計執行**

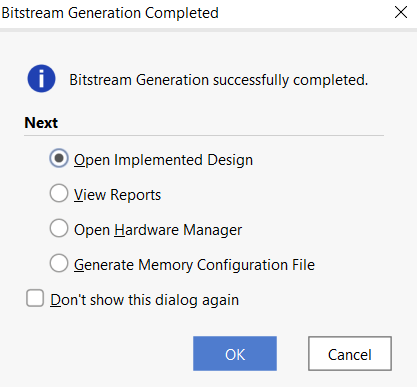
|  |
| --- |
| **附註：** 如果您收到如下錯誤：Gtk-Message: Failed to load module "canberra-gtk-module"  通過以下命令安裝套件即可解決問題。    sudo apt install libcanberra-gtk-module libcanberra-gtk3-module  如果您使用的是VM，則Vivado在合成時可能會由於RAM分配不足而發生崩潰。如果Vivado發生崩潰，建議為VM分配更多的RAM。 |

這個過程可能需要幾分鐘，具體取決於電腦的速度。



**圖78. Verilog包含檔案路徑**

產生位元串流之後，將彈出一個視窗，如圖79所示。按一下右上角的X按鈕，關閉視窗。



**圖79. 已產生位元串流**

現在位元串流已經建立完畢，在下一個實驗中，我們將介紹如何通過PlatformIO將該位元串流上傳到Nexys A7電路板上，然後還將介紹如何在本實驗中剛剛構建的SweRVolfX子集上執行範例程式。