

**IMAGINATION大學計劃**

**RVfpga實驗3**

**RISC-V組合語言**

# 簡介

對於程式設計師而言，C、Java和Python等高階語言能夠提高程式設計的效率。這些高階語言被轉換成組合語言，即一組簡單的指令。有時需要用組合語言編寫對性能或時序要求嚴格的程式碼段，以保證特定的時序或縮短計算時間。本實驗將展示如何使用PlatformIO建立一個可在RVfpga系統上執行的RISC-V組合語言程式。我們首先簡要概述RISC-V組合語言，然後展示如何在RVfpgaNexys上建立和執行組合語言程式（請記住，也可以使用Verilator或Whisper模擬程式）。然後，我們會提供一些練習，讓您編寫自己的RISC-V組合語言程式。

# RISC-V組合語言概述

RISC-V組合語言包含用於實作高階程式碼的簡單指令。例如add、sub和mul等常用的RISC-V指令（分別用於對兩個運算元進行加法、減法和乘法運算）。

RISC-V指令的基本類型包括：計算（算術、邏輯和移位）指令、記憶體存取操作和分支/跳轉。表1中列出了最常用的RISC-V指令。指令使用的運算元位於暫存器或記憶體中，或者被編碼為常數（即*立即數*）。RISC-V包含32個32位元暫存器。表2列出了這32個RISC-V暫存器的名稱。這些暫存器可以使用暫存器名稱（例如zero、s0、t5等）或暫存器編號（即x0、x8、x30）來指定。程式設計師通常使用暫存器名稱，其中保留了部分暫存器典型用途的相關資訊。例如，儲存暫存器s0-s11通常用於程式變量，而臨時暫存器t0-t6用於臨時計算。zero暫存器（x0）中始終包含值0，因為程式中通常需要該值。其他暫存器也均有特定用途，如表2所示，但在本實驗中，只需使用zero暫存器、臨時暫存器和儲存暫存器。

表1. 常用的RISC-V組合語言程式碼指令

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | **RISC-V組合語言** | | **說明** | | **操作** |
| **計算** | add s0, s1, s2 | | 加法 | | s0 = s1 + s2 | |
| sub s0, s1, s2 | | 減法 | | s0 = s1 - s2 | |
| addi t3, t1, -10 | | 加立即數 | | t3 = t1 - 10 | |
| mul t0, t2, t3 | | 32位元乘法 | | t0 = t2 \* t3 | |
| div s9, t5, t6 | | 除法 | | t9 = t5 / t6 | |
| rem s4, s1, s2 | | 求餘數 | | s4 = s1 % s2 | |
| and t0, t1, t2 | | 按位元AND | | t0 = t1 & t2 | |
| or t0, t1, t5 | | 按位元OR | | t0 = t1 | t5 | |
| xor s3, s4, s5 | | 按位元XOR | | s3 = s4 ^ s5 | |
| andi t1, t2, 0xFFB | | 按位元AND（立即數） | | t1 = t2 & 0xFFFFFFFB | |
| ori t0, t1, 0x2C | | 按位元OR（立即數） | | t0 = t1 | 0x2C | |
| xori s3, s4, 0xABC | | 按位元XOR（立即數） | | s3 = s4 ^ 0xFFFFFABC | |
| sll t0, t1, t2 | | 邏輯左移 | | t0 = t1 << t2 | |
| srl t0, t1, t5 | | 邏輯右移 | | t0 = t1 >> t5 | |
| sra s3, s4, s5 | | 算術右移 | | s3 = s4 >>> s5 | |
| slli t1, t2, 30 | | 邏輯左移（立即數） | | t1 = t2 << 30 | |
| srli t0, t1, 5 | | 邏輯右移（立即數） | | t0 = t1 >> 5 | |
| srai s3, s4, 31 | | 算術右移（立即數） | | s3 = s4 >>> 31 | |
| **記憶體存取** | lw s7, 0x2C(t1) | | 載入字 | | s7 = memory[t1+0x2C] | |
| lh s5, 0x5A(s3) | | 載入半字 | | s5 = SignExt(memory[s3+0x5A]15:0) | |
| lb s1, -3(t4) | | 載入位元組 | | s1 = SignExt(memory[t4-3]7:0) | |
| sw t2, 0x7C(t1) | | 儲存字 | | memory[t1+0x7C] = t2 | |
| sh t3, 22(s3) | | 儲存半字 | | memory[s3+22]15:0 = t315:0 | |
| sb t4, 5(s4) | | 儲存位元組 | | memory[s4+5]7:0 = t47:0 | |
| **分支** | beq s1, s2, L1 | | 如相等則分支跳轉 | | if (s1==s2), PC = L1 | |
| bne t3, t4, Loop | | 如不相等則分支跳轉 | | if (s1!=s2), PC = Loop | |
| blt t4, t5, L3 | | 如小於則分支跳轉 | | if (t4 < t5), PC = L3 | |
| bge s8, s9, Done | | 如大於或等於則分支跳轉 | | if (s8>=s9), PC = Done | |
| **虛擬指令** | li s1, 0xABCDEF12 | | 立即載入 | | s1 = 0xABCDEF12 | |
| la s1, A | | 載入位址 | | s1 = 儲存變數A的記憶體位址 | |
| nop | | 無操作 | | 無操作 | |
| mv s3, s7 | | 移動 | | s3 = s7 | |
| not t1, t2 | | 非（求反） | | t1 = ~t2 | |
| neg s1, s3 | | 求補 | | s1 = -s3 | |
| j Label | | 跳轉 | | PC = 標籤 | |
| jal L7 | | 跳轉並連結 | | PC = L7；ra = PC + 4 | |
| jr s1 | | 跳轉暫存器 | | PC = s1 | |

除了實際的RISC-V指令，RISC-V還包括虛擬指令（如表1底部所示）,虛擬指令並非真正的RISC-V指令，但程式設計師經常使用。虛擬指令是使用一個或多個真正的RISC-V指令實作的。例如，移動偽指令（mv s1, s2）會複製s2的內容並將其放入s1。該指令是使用以下真正的RISC-V指令實作的：addi s1, s2, 0。

**表2. RISC-V暫存器**

|  |  |  |
| --- | --- | --- |
| **名稱** | **暫存器編號** | **用途** |
| **zero** | **x0** | 常數值0 |
| **ra** | **x1** | 返回位址 |
| **sp** | **x2** | 堆疊指標 |
| **gp** | **x3** | 全域指標 |
| **tp** | **x4** | 執行緒指標 |
| **t0-2** | **x5-7** | 臨時變數 |
| **s0/fp** | **x8** | 儲存暫存器/架構指標 |
| **s1** | **x9** | 儲存暫存器 |
| **a0-1** | **x10-11** | 函數引數/傳回值 |
| **a2-7** | **x12-17** | 函數引數 |
| **s2-11** | **x18-27** | 儲存暫存器 |
| **t3-6** | **x28-31** | 臨時變數 |

# 以句點開頭的命令為組合語言程式碼器指令。這些指令是組合語言程式碼器需執行的命令，而不是組合語言程式碼器要轉換的程式碼。這些指令會告知組合語言程式碼器放置程式碼和資料的位置，指定在程式中使用的文字和資料常數等。表3列出了RISC-V的主要組合語言程式碼器指令（*《RISC-V讀者：開放架構Atlas》，作者：Patterson & Waterman*，© 2017）。

**表3. RISC-V主要指令**

|  |  |
| --- | --- |
| **指令** | **說明** |
| **.text** | 後續項目儲存在text部分（機器程式碼）。 |
| **.data** | 後續項目儲存在data部分（全域變數）。 |
| **.bss** | 後續項目儲存在bss部分（全域變數初始化為0）。 |
| **.section .foo** | 後續項目儲存在名為.foo的部分。 |
| **.align n** | 以2n位元組邊界對齊下一資料。例如，.align 2會以字邊界對齊下一個值。 |
| **.balign n** | 以n位元組邊界對齊下一資料。例如，.balign 4會以字邊界對齊下一個值。 |
| **.globl sym** | 宣告標籤sym為全域變數，可以從其他檔案參考 |
| **.string “str”** | 將字串str儲存在記憶體中並以 null 結尾。 |
| **.word w1,…,wn** | 將n個32位元數值儲存在連續的記憶體字中。 |
| **.byte b1,…,bn** | 將n個8位元數值儲存在連續的記憶體位元組中。 |
| **.space** | 保留記憶體空間以儲存沒有初始值的變數。該指令通常用於宣告輸出變數（這些輸出變數不得同時作為輸入變數）。要保留的空間必須始終以位元組數形式表示。例如，指令RES: .space 4會保留四個未初始化的位元組（即一個字）。 |
| **.equ name,constant** | 定義符號名稱及其常數值。例如，.equ N,12會定義符號N，其值為12。 |
| **.end** | 組合語言程式碼器在到達.end指令時會停止工作。該指令之後的所有文字都將被忽略。 |

# 以下範例（請參閱表4 - 表5）展示了如何以RISC-V組合語言編碼一些常見的高階結構。請注意，分支指令（beq、bne、blt和bge）會有條件地跳轉到特定標籤；而跳轉指令（j）會無條件地跳轉到特定標籤。單行註釋在C語言中以//表示，在RISC-V組合語言中以#表示。

請注意，在第一個範例中（有關if/else陳述式的實作，請參閱表4），C程式碼和RISC-V組合語言程式碼會檢查相反的情況：C程式碼檢查小於（<）某值的情況，而相應的組合語言程式碼檢查大於或等於（>=）某值的情況。

**表4. RISC-V組合語言程式碼範例1：if/else陳述式**

|  |  |
| --- | --- |
| **// C Code**  int a, b, c;  if (a < b)  c = 5;  else  c = a + b; | **# RISC-V組合語言**  # s0 = a, s1 = b, s2 = c  bge s0, s1, L1 # if (a >= b) goto L1  addi s2, zero, 5 # c = 5  j L2 # jump over else block  L1: add s2, s0, s1 # c = a + b  L2: |

在第二個範例中（有關整數陣列的操作，請參閱表5），RISC-V組合語言程式碼使用臨時暫存器（t0-t3）來儲存臨時值，例如常數100和資料陣列的基本位址。對前三個指令中的暫存器進行初始化後，RISC-V組合語言程式碼會使用bge（如大於或等於則分支）指令檢查i >= 100的情況；這同樣與C程式碼相反。如果滿足該條件，則完成for迴圈。如果**未**進行分支，則i小於100，將執行其餘程式碼。請注意，索引i要先乘以4（使用slli t2, s0, 2指令）再新增到基本位址中，因為整數（32位元二進位補數）會占用4個位元組的儲存空間。在RISC-V中，儲存空間可按位元組尋址（即每個位元組都有自己的位址）。如果陣列為字元陣列（即char data[100];），每個陣列元素將只占用一個位元組，並且可以直接將i新增到基本位址中，形成陣列索引i的位址，即array[i]。讀取陣列元素後，會將其減10，然後將其寫入（分別通過lw、addi和sw指令），陣列索引i（即s0）將遞增，程式跳回到for迴圈的開頭（使用j L5指令）。

**表5. RISC-V組合語言程式碼範例2：操作整數陣列**

|  |  |
| --- | --- |
| **// C Code**  int i;  int data[100];  for (i=0; i<100; i++)  array[i] = array[i]-10; | **# RISC-V組合語言**  # s0 = i, t1 = base address of data (assumed  # to be at 0x300)  addi s0, zero, 0 # i = 0  addi t0, zero, 100 # t0 = 100  li t1, 0x300 # base address of array  L5: bge s0, t0, L7 # if (i>=100) exit loop  slli t2, s0, 2 # t2 = i\*4  add t2, t1, t2 # address of data[i]  lw t3, 0(t2) # t3 = array[i]  addi t3, t3, -10 # t3 = array[i]-10  sw t3, 0(t2) # array[i] = array[i]-10  addi s0, s0, 1 # i++  j L5 # loop  L7: |

有關RISC-V組合語言的更多詳細資訊，請參見《RISC-V指令集手冊》（下載位址：<https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf>）或*《數位設計和電腦體系結構》*（作者：Harris & Harris，Elsevier，© 2021，出版時間：2021年夏季）或*《RISC-V讀者：開放架構Atlas》*（作者：Patterson & Waterman，© 2017）。

# 為RVfpga編寫RISC-V組合語言程式

現在您可以著手編寫自己的RISC-V組合語言程式。在自行編寫程式之前，請按照以下步驟設定PlatformIO專案，然後在RVfpgaNexys上建立並執行組合語言程式（請記住，也可以使用Verilator或Whisper模擬這些程式）：

1. 建立RVfpga專案
2. 編寫RISC-V組合語言程式
3. 將RVfpgaNexys下載到Nexys A7 FPGA開發板
4. 編譯、下載和執行組合語言程式

第1步. 建立RVfpga專案

按RVfpga實驗2中的第1步進行操作 - 為方便起見，下文將重述相關操作。按一下「Start」（開始）按鈕，輸入「VSCode」，然後按一下Visual Studio Code以開啟VSCode（請參閱圖1）。

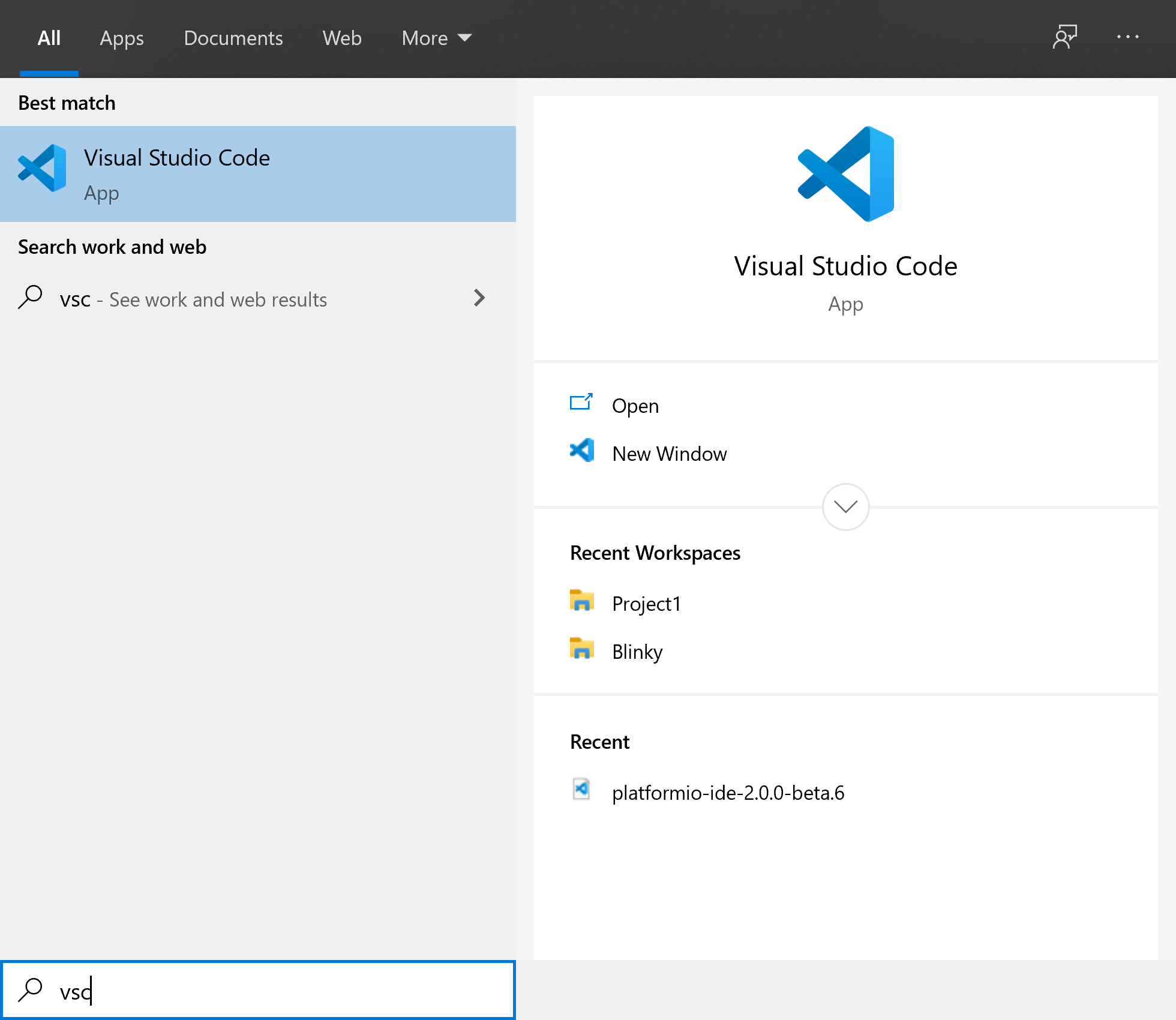


圖1. 開啟VSCode

如果啟動VScode時PlatformIO沒有自動開啟，請按一下左側功能表功能區中的PlatformIO圖示，然後按一下「PIO Home」（PIO首頁）→「Open」（開啟）（請參閱圖2）。

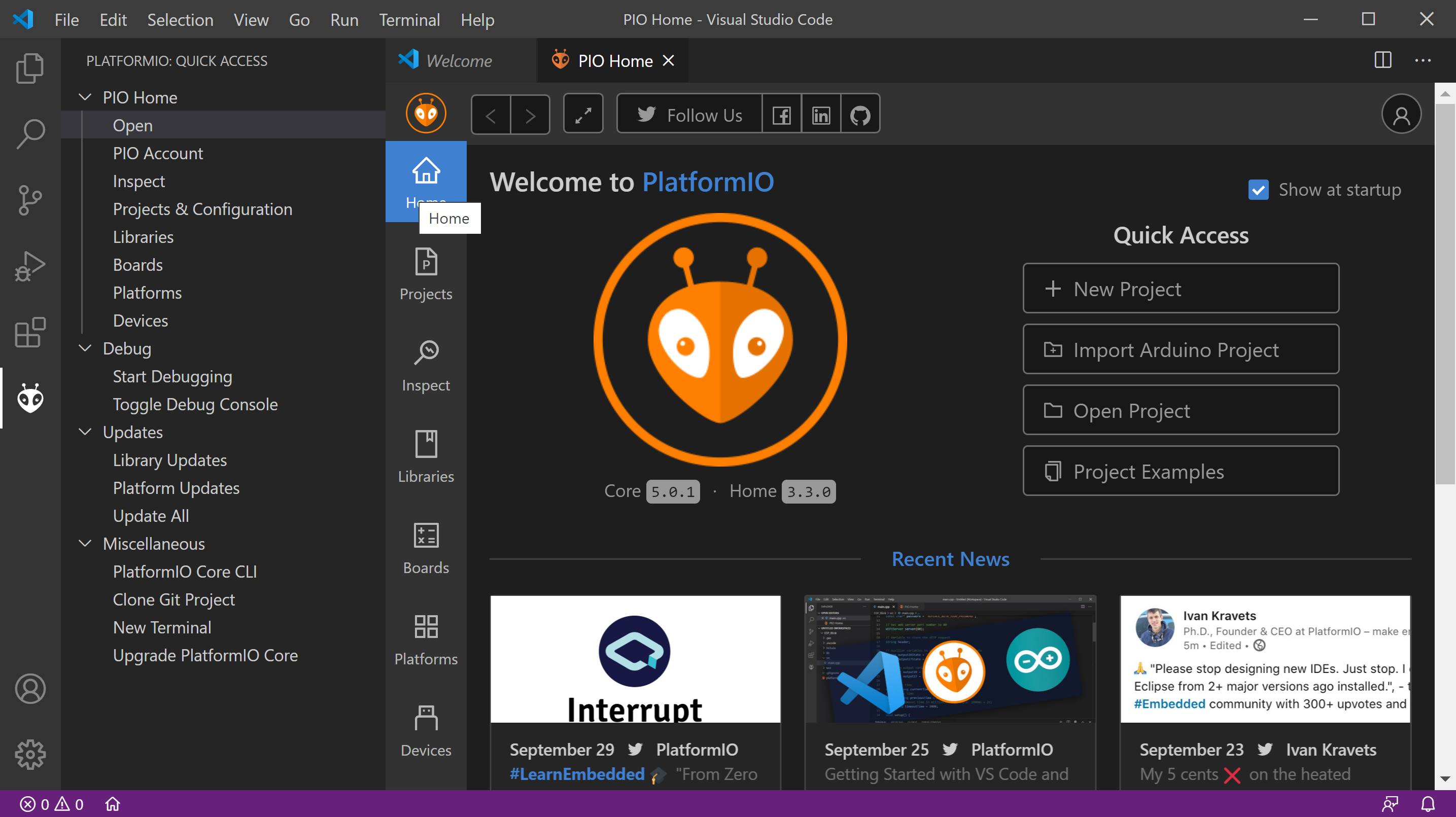


圖2. 開啟PlatformIO並建立新專案

現在，在「PIO Home」（PIO首頁）歡迎視窗中，按一下「New Project」（新建專案）（請參閱圖2）。

如圖3所示，將專案命名為「Project1」，在「Board」（開發板）部分選擇「RVfpga: Digilent Nexys A7」（輸入RVfpga，即可出現該選項）。保留框架的預設選項WD-framework（Western Digital框架，其中包含Freedom-E SDK gcc和gdb）。取消點按「Use default location」（使用預設位置），然後將程式放到以下路徑：

[RVfpgaPath]/RVfpga/Labs/Lab3

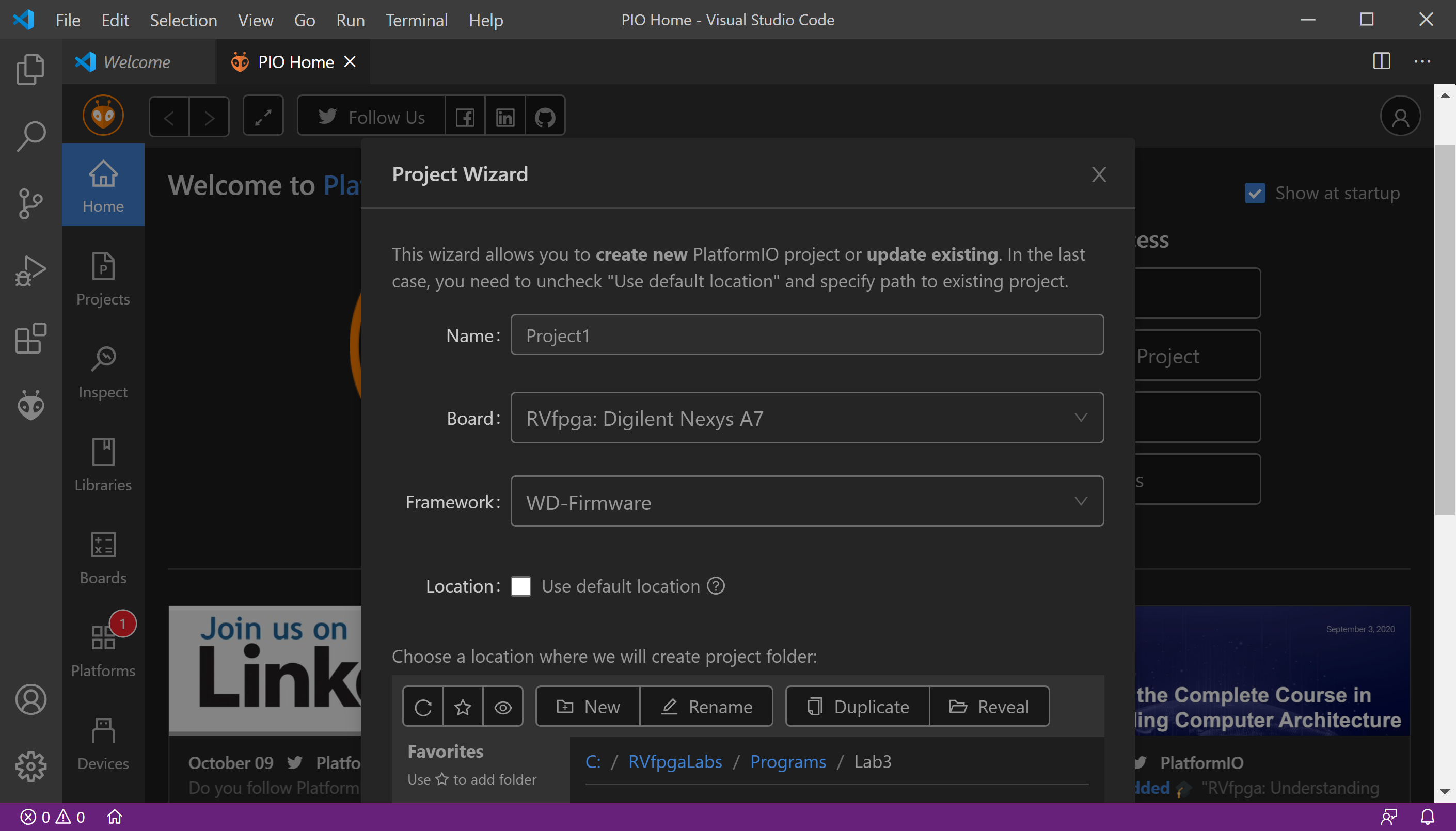


圖. 命名專案並選擇開發板和專案資料夾

然後點擊視窗底部的「Finish」（完成）（請參閱圖4）。

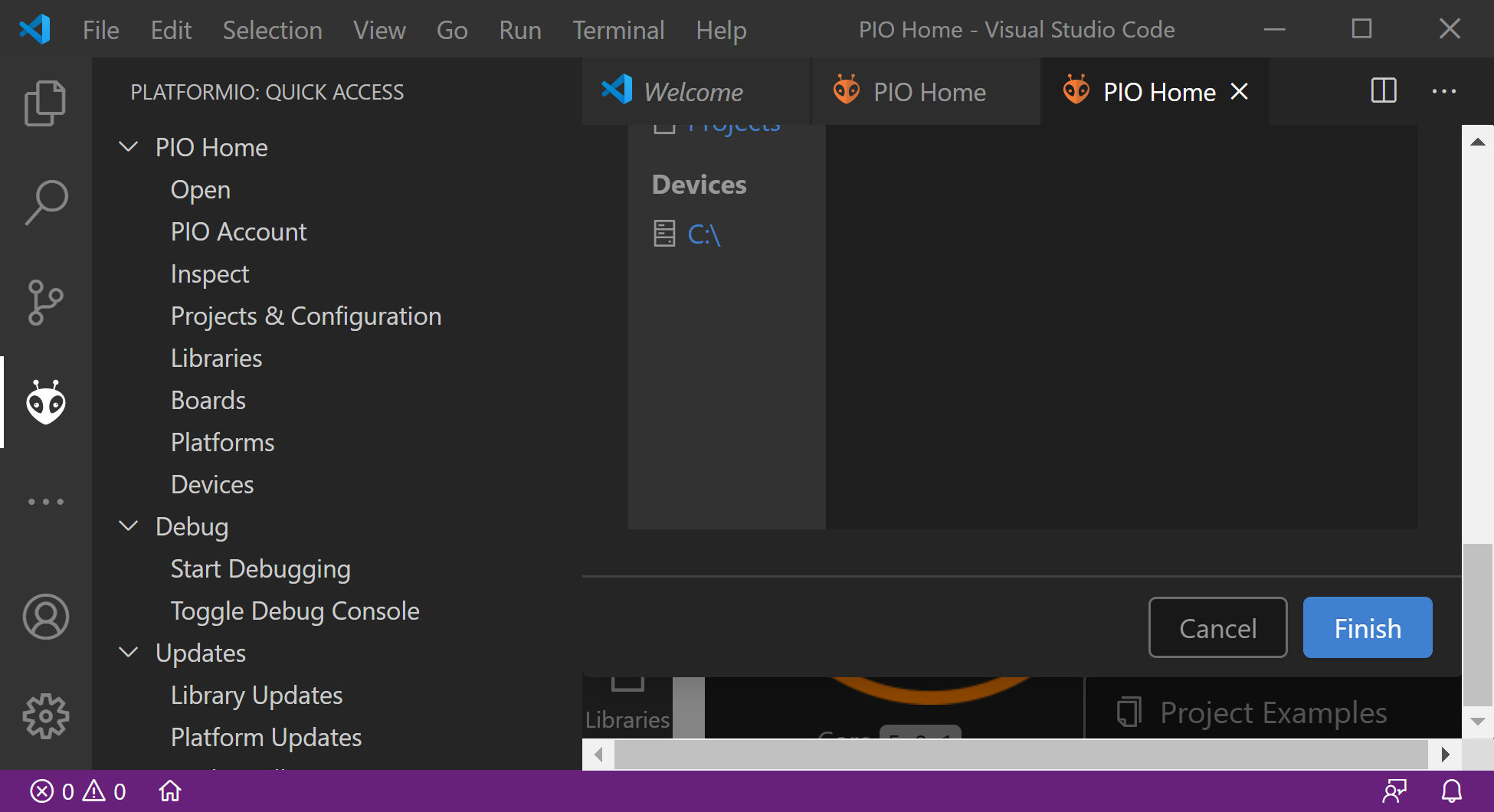


圖4. 完成專案建立

在左側「Explorer」（檔案總管）窗格的PROJECT1下（可能需要展開），按兩下platformio.ini開啟該檔案（請參閱圖5）。該檔案為PlatformIO初始化檔。

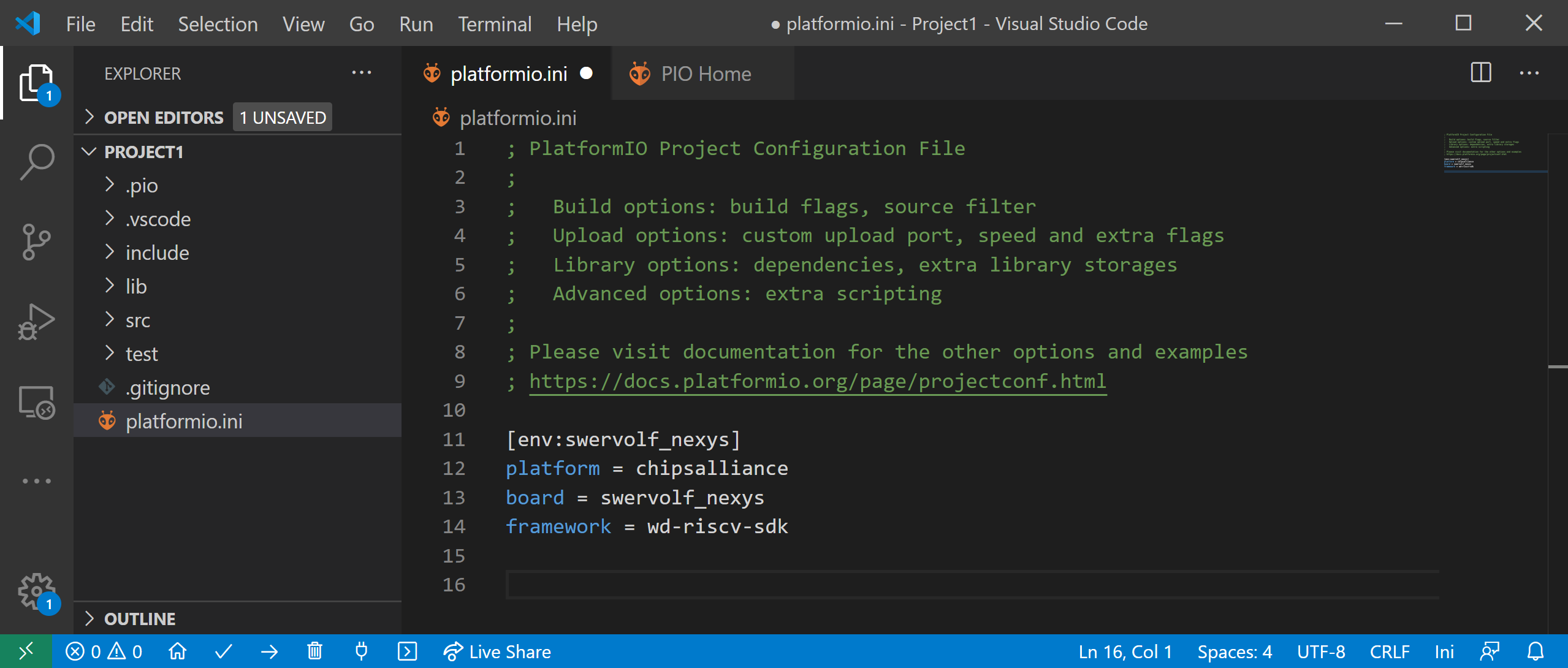


圖5. PlatformIO初始化檔：platformio.ini

將以下程式碼行新增到platformio.ini檔，如圖6所示：

board\_build.bitstream\_file = [RVfpgaPath]/RVfpga/Labs/Lab1/Project1/Project1.runs/impl\_1/rvfpganexys.bit

此行程式碼指示PlatformIO可以在哪裡找到要載入到FPGA上的位元串流檔。上述路徑是在實驗1中建立的位元串流位置。（如果尚未完成實驗1，可以使用「入門指南」中隨附的RVfpgaNexys位元串流，路徑為：*[RVfpgaPath]/RVfpga/src/rvfpganexys.bit*。）按Ctrl-s儲存platformio.ini檔。

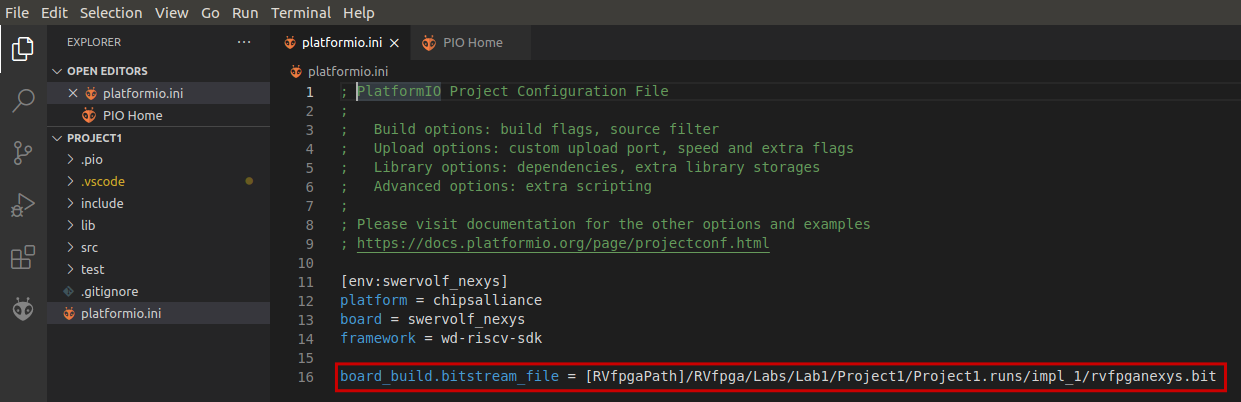


圖6. RVfpgaNexys位元串流檔案（rvfpganexys.bit）的新增位置

請記住，「入門指南」的範例中使用了更完整的*platformio.ini*檔。如果要使用任何需要額外命令的功能（例如Verilator模擬器的路徑、序列控制台的配置、Whisper偵錯工具等），則可以使用上述範例中的*platformio.ini*。

第2步. 編寫RISC-V組合語言程式

現在，需要編寫RISC-V組合語言程式。按一下「File」（檔案）→「New File」（新建檔案）（請參閱圖7）。

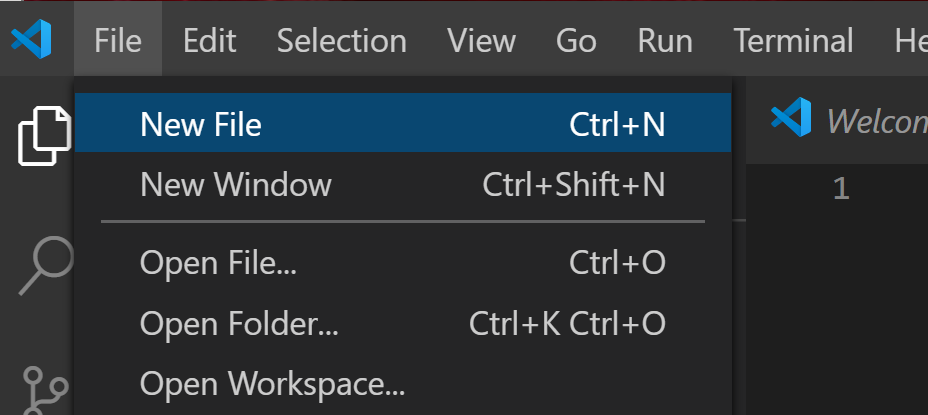


圖7. 在專案中新增檔案

將開啟一個空白視窗。在該視窗中輸入（或複製/貼上）以下RISC-V組合語言程式（請參閱圖8）。也可通過以下路徑獲取該程式：

[RVfpgaPath]/RVfpga/Labs/Lab3/ReadSwitches.S

// memory-mapped I/O addresses

# GPIO\_SWs = 0x80001400

# GPIO\_LEDs = 0x80001404

# GPIO\_INOUT = 0x80001408

.globl main

main:

main:

li t0, 0x80001400 # base address of GPIO memory-mapped registers

li t1, 0xFFFF # set direction of GPIOs

# upper half = switches (inputs) (=0)

# lower half = outputs (LEDs) (=1)

sw t1, 8(t0) # GPIO\_INOUT = 0xFFFF

repeat:

lw t1, 0(t0) # read switches: t1 = GPIO\_SWs

srli t1, t1, 16 # shift val to the right by 16 bits

sw t1, 4(t0) # write value to LEDs: GPIO\_LEDs = t1

j repeat # repeat loop

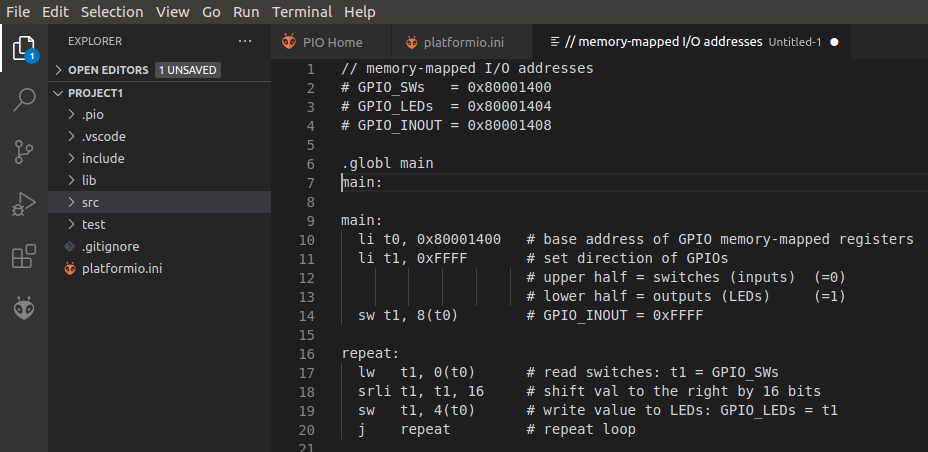


圖8. 輸入RISC-V組合語言程式

組合語言程式碼的開頭必須包含以下程式碼行：

.globl main

main:

組合語言程式碼器指令.globl會在所有連結檔中顯示標籤。開機程式碼（*~/.platformio/packages/framework-wd-riscv-sdk/board/nexys\_a7\_eh1/startup.S*）將配置系統並跳轉到該標籤（*main*）。偵錯工具將在開始工作時在該處設定一個臨時中斷點。

此RISC-V組合語言程式與實驗2中的程式範例相同，但這次是用RISC-V組合語言編寫的。此程式會設定通用I/O（GPIO）的輸入和輸出方向，然後重複讀取開關的值並將該值寫入LED。

將程式輸入窗格後，按Ctrl-s儲存檔案。將其命名為ReadSwitches.S，並儲存到Project1目錄下的src資料夾中（請參閱圖9）。

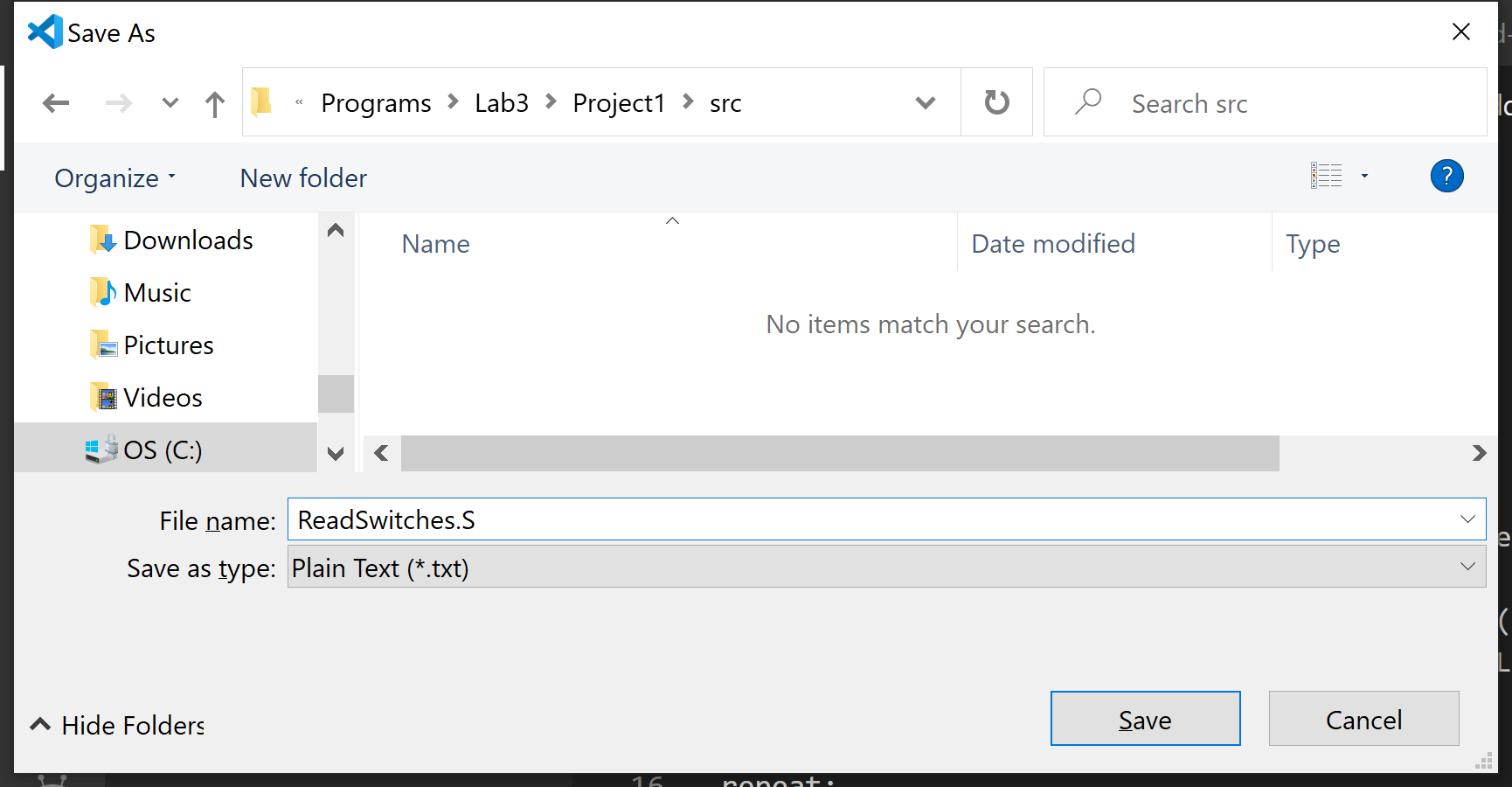


圖9. 將檔案另存為ReadSwitches.S

第3步. 將RVfpgaNexys下載到Nexys A7 FPGA開發板

現在，需要將RVfpgaNexys下載到Nexys A7 FPGA開發板上。根據GSG和實驗2中的指示下載RVfpgaNexys - 為方便起見，下文將重述相關操作。

按一下左側功能表功能區中的PlatformIO圖示，展開「Project Tasks」（專案任務）→ env:swervolf\_nexys →「Platform」（平台），然後按一下「Upload Bitstream」（上傳位元串流），將RVfpgaNexys下載到Nexys A7開發板。

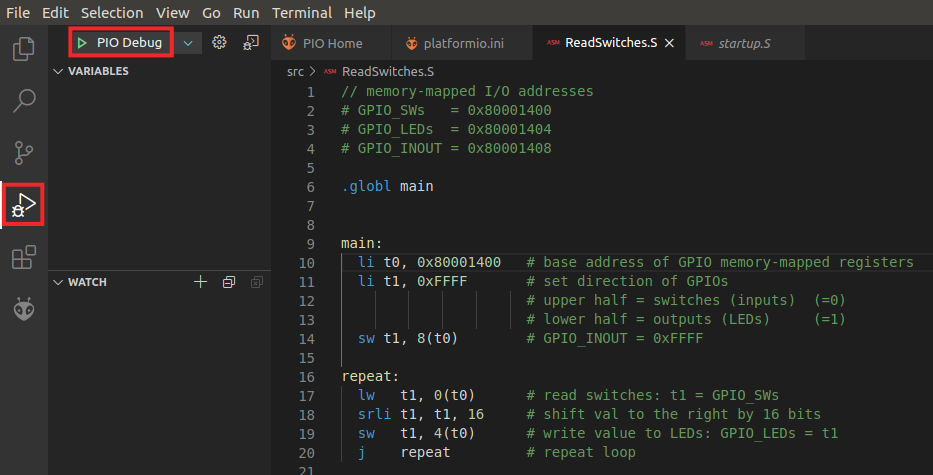
也可以使用PlatformIO終端機視窗下載RVfpgaNexys，方法是按一下PlatformIO視窗底部功能表中的「PlatformIO: New Terminal」（PlatformIO：新建終端機）按鈕（），然後將以下內容輸入（或複製）到PlatformIO終端機中：

pio run -t program\_fpga

第4步. 編譯、下載和執行RISC-V組合語言程式

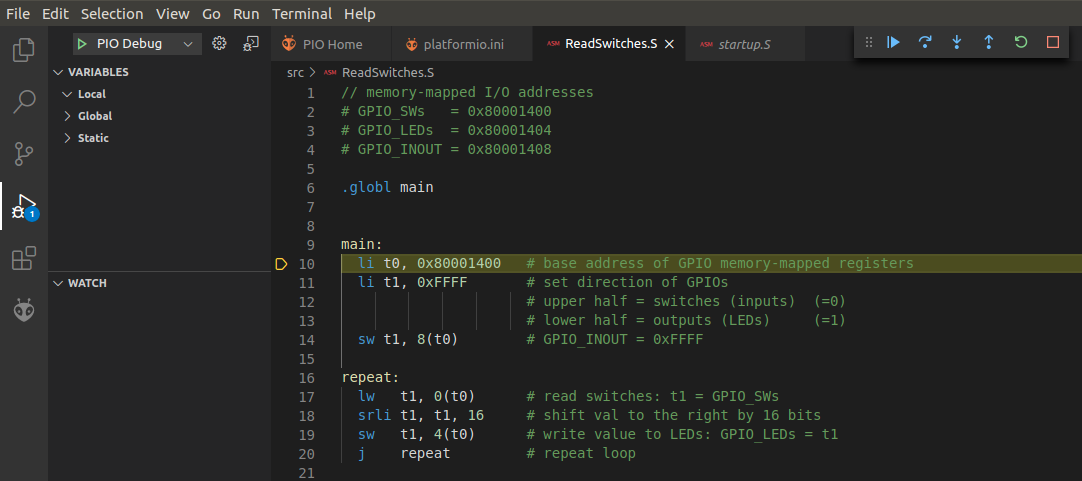
現在RVfpgaNexys已在開發板上執行，接下來需要編譯程式，將程式下載到RVfpgaNexys，然後執行/除錯程式。如果VSCode尚未開啟，請將其開啟。上一個專案Project1應該會自動開啟。如果該專案未自動開啟，請確保已開啟PlatformIO延伸模組，然後按一下「File」（檔案）→「Open Folder」（開啟資料夾）並選擇（但不要開啟）在本實驗前面部分建立的Project1。

按一下左側功能表功能區中的「Run」（執行）按鈕，然後按一下「Start Debugging」（開始偵錯）按鈕（請參閱圖10）。



**圖10. 在RVfpgaNexys上執行程式**

程式將下載至正在Nexys A7的FPGA開發板上執行的RVfpgaNexys中。此時即可開始執行和偵錯程式（請參閱圖11）。



**圖11. 在RVfpgaNexys上執行的程式**

根據「RVfpga入門指南」和實驗2中的指示，使用偵錯工具列和偵錯工具選項來執行和管理程式。例如，可以在第17行設定一個中斷點（只需按一下行號左側），然後檢視暫存器t1，因為開關的值已載入到暫存器中。按下「Stop」（停止）按鈕（或Shift - F5）停止偵錯工作階段後，偵錯工作階段將終止，但程式仍繼續在RVfpgaNexys上執行。

# 練習

現在可再次重複實驗2中的練習，將C語言替換為RISC-V組合語言，建立自己的RISC-V組合語言程式。為方便起見，下文再次提供了練習說明。

請注意，如果使Nexys A7開發板與電腦保持連接並維持通電狀態，則在切換執行不同程式時，無需將RVfpgaNexys重新重新載入到開發板上。但是，如果關閉Nexys A7開發板，則需要使用PlatformIO將RVfpgaNexys重新載入到開發板上。

還要記住，可以使用Verilator或Whisper模擬這些程式。

練習1. 編寫一個RISC-V組合語言程式，使開關的值在LED上閃爍。該值應以足夠慢的速度進行脈衝開關，讓人眼可以觀察到值在閃爍。將程式命名為FlashSwitchesToLEDs.S。

練習2. 編寫一個RISC-V組合語言程式，在LED上顯示開關值反轉後的值。例如，如果開關的值（二進位）為：0101010101010101，則LED應顯示：1010101010101010；如果開關的值為：1111000011110000，則LED應顯示：0000111100001111；依此類推。將程式命名為DisplayInverse.S。

練習3. 編寫一個RISC-V組合語言程式，不斷增加點亮並捲動的LED的數量，直到所有LED都點亮為止。然後應重複執行該模式。將程式命名為ScrollLEDs.S。

該程式應產生以下效果：

1. 首先，有一個點亮的LED從右向左捲動。
2. 到達最左側的LED後，應有兩個點亮的LED從左向右捲動，然後又從右向左捲動。
3. 到達最左側的兩個LED後，應有三個點亮的LED從左向右捲動，然後又從右向左捲動。
4. 接下來應有四個點亮的LED捲動。
5. 依此類推，直到所有LED均點亮。
6. 然後應重複執行該模式。

練習4. 編寫一個RISC-V組合語言程式，顯示開關的4個最低有效位元和開關的4個最高有效位元相加所得的4位元不帶正負號值。在LED的4個最低有效（最右邊）位元上顯示結果。將程式命名為4bitAdd.S。當發生不帶正負號溢位（即進位為1）時，LED的第五位元應亮起。

**練習5.** 編寫一個RISC-V組合語言程式，根據歐幾里得演算法求出*a*和*b*兩個數的*最大公約數*。*a*和*b*兩個值在程式中應該是靜態定義的變數。將程式命名為**GCD.S**。有關歐幾里得演算法的更多資訊，請造訪：<https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm>。也可以直接用google搜尋「歐幾里德演算法」。

**練習6.** 編寫一個RISC-V組合語言程式，計算斐波那契數列中的前12個數，並將結果儲存在長度為12的有限向量（即陣列）*V*中。這個無限的斐波納契數列定義如下：

V(0)=0，V(1)=1，V(i)=V(i-1)+V(i-2)（其中i=0,1,2...）

換言之，與元素i對應的斐波那契數是數列中在其之前的兩個斐波那契數之和。表6顯示了i為0到8時的斐波那契數。

表6. 斐波那契數列

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***i*** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| ***V*** | 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 |

向量的維數*N*在程式中必須定義為常數。將程式命名為**Fibonacci.S**。

**練習7.** 現有一個*N*維向量（即陣列）*A*，請產生另一個向量*B*，使向量*B*只包含*A*中大於0的偶數元素。例如：假設*N* = 12，*A* = [0,1,2,7,-8,4,5,12,11,-2,6,3]，則B為：*B* = [2,4,12,6]。將程式命名為**EvenPositiveNumbers.S**。

**練習8.** 現有兩個*N*維向量（即陣列）*A*和*B*，請產生另一個向量*C*，定義如下：

C(i) = |A[i] + B[N-i-1]|, i = 0,..,N-1。

用RISC-V組合語言編寫一個計算新向量的程式。在程式中使用包含12個元素的陣列。將程式命名為**AddVectors.S**。

**練習9.** 用RISC-V組合語言實作泡泡排序演算法，讓演算法通過以下方式按升序對向量元件進行排序：

1. 重複遍歷向量直至完成。
2. 如果兩個相鄰元件V(i) > V(i+1)，則互換其位置。
3. 當每對相鄰元件的順序均正確時，演算法停止。

使用包含12個元素的陣列測試程式。將程式命名為**BubbleSort.S**。

**練習10.** 用RISC-V組合語言編寫一個程式，通過迭代乘法計算給定非負數*n*的階乘。雖然應使用多個n值測試程式，但最終提交的應為*n* = 7時的結果。*n*應為程式內靜態定義的變數。將程式命名為**Factorial.S**。