



IMAGINATION大學計劃

RVfpga實驗2

C語言程式設計

1. 簡介

大多數電腦程式是以高階語言（如C語言）編寫的。本實驗將展示如何在PlatformIO中建立一個可在RVfpga系統上執行的C語言專案。我們首先提供關於建立和執行C程式的教程。接著我們會提供一些練習，讓您編寫自己的C程式。

2. RVfpga的C程式

要使用PlatformIO在RVfpgaNexys上建立和執行C程式，需要完成以下步驟（請記住，也可以使用Verilator和Whisper在模擬中執行這些程式）：

1. 建立RVfpga專案
2. 編寫C程式
3. 將RVfpgaNexys下載到Nexys A7 FPGA開發板
4. 編譯、下載和執行C程式

第1步. 建立RVfpga專案

開啟VSCode（如「RVfpga入門指南」中所述）。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1）。

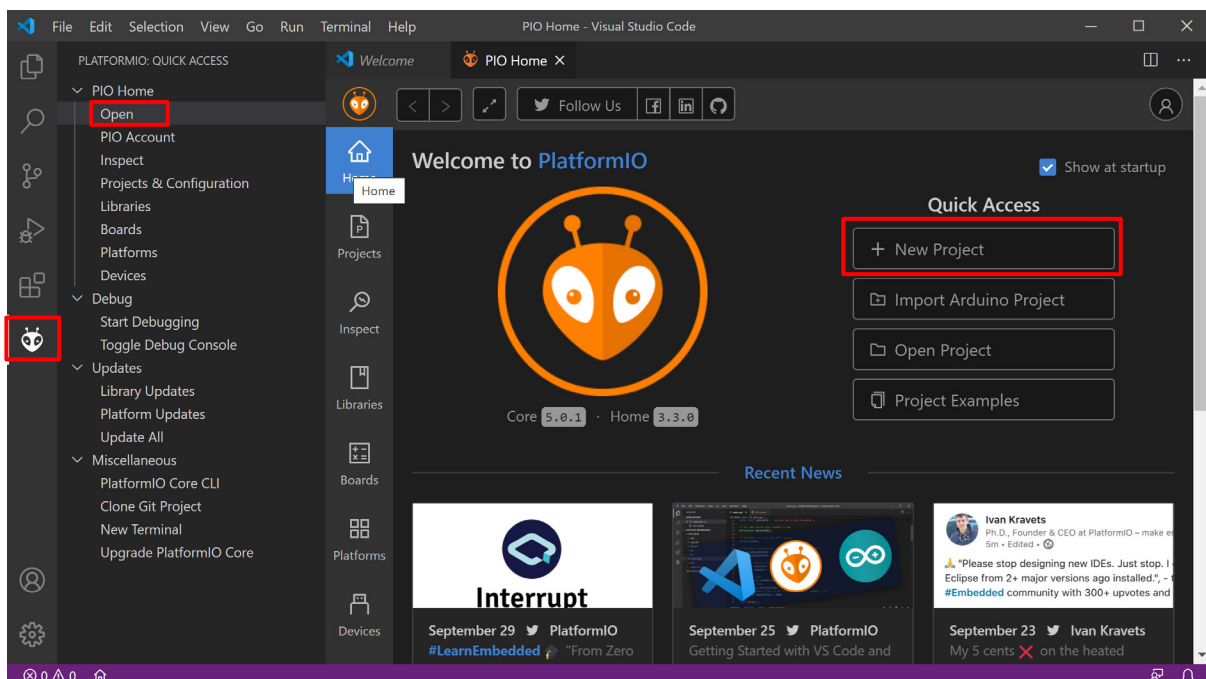


圖1. 開啟PlatformIO並建立新專案

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1）。

如圖2所示，將專案命名為「Project1」，在「Board」（開發板）部分選擇「RVfpga: Diligent Nexys A7」（輸入RVfpga，即可出現該選項）。保留架構的預設選項WD-Firmware。WD-Firmware是Western Digital的一款韌體，其中包含Freedom-E SDK gcc和gdb以及我們在這些實驗中使用的PSP和BSP（處理器支援套件和開發板支援套件）。取消點按「Use default location」（使用預設位置），然後將專案放到以下路徑：

`[RVfpgaPath]/RVfpga/Labs/Lab2`

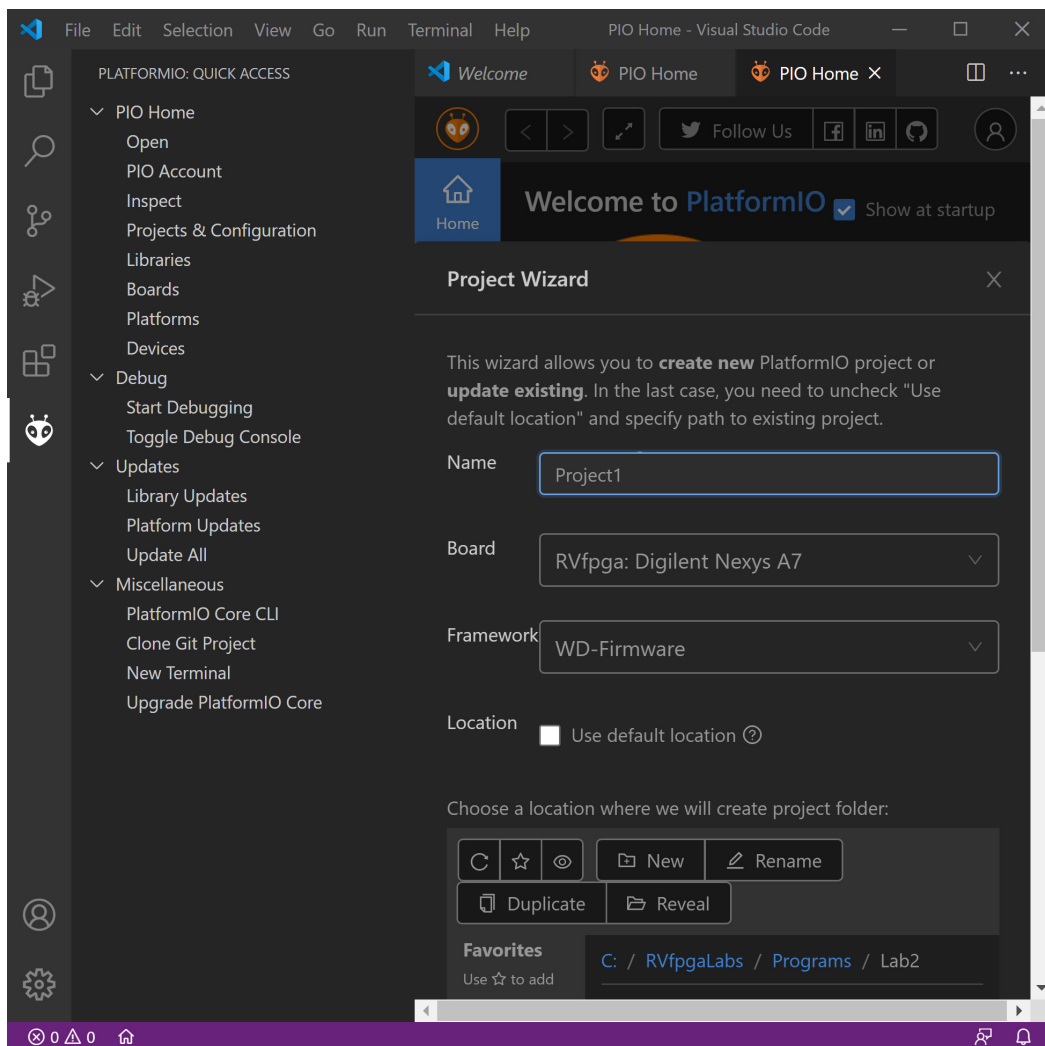


圖2. 命名專案並選擇開發板和專案資料夾

然後點擊視窗底部的「Finish」（完成）（請參閱圖3）。

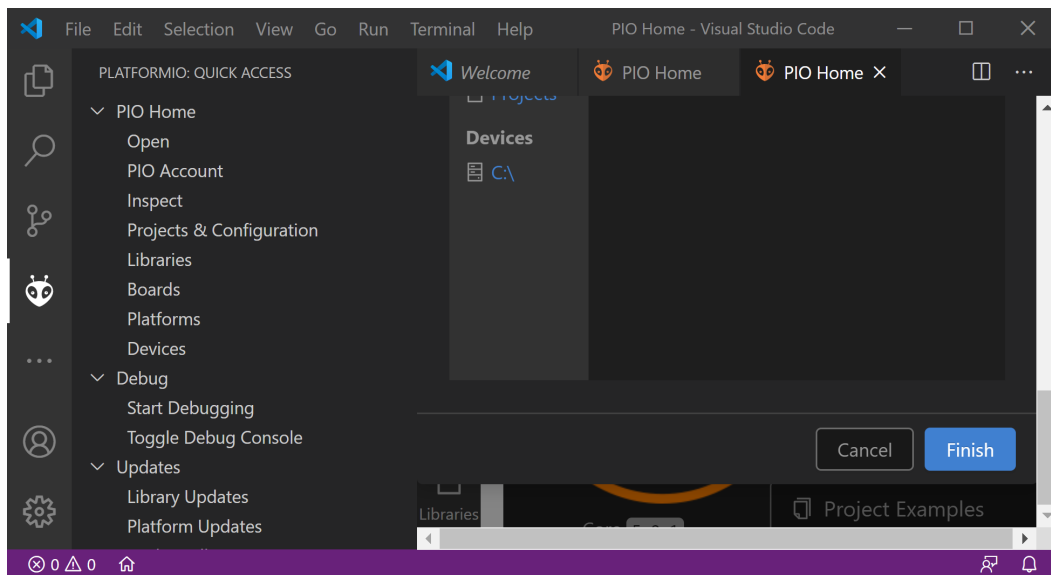


圖3. 完成專案建立

在左側的「Explorer」（檔案總管）窗格中（可能需要展開），按兩下platformio.ini開啟該檔案（請參閱圖4）。該檔案為PlatformIO初始化檔。

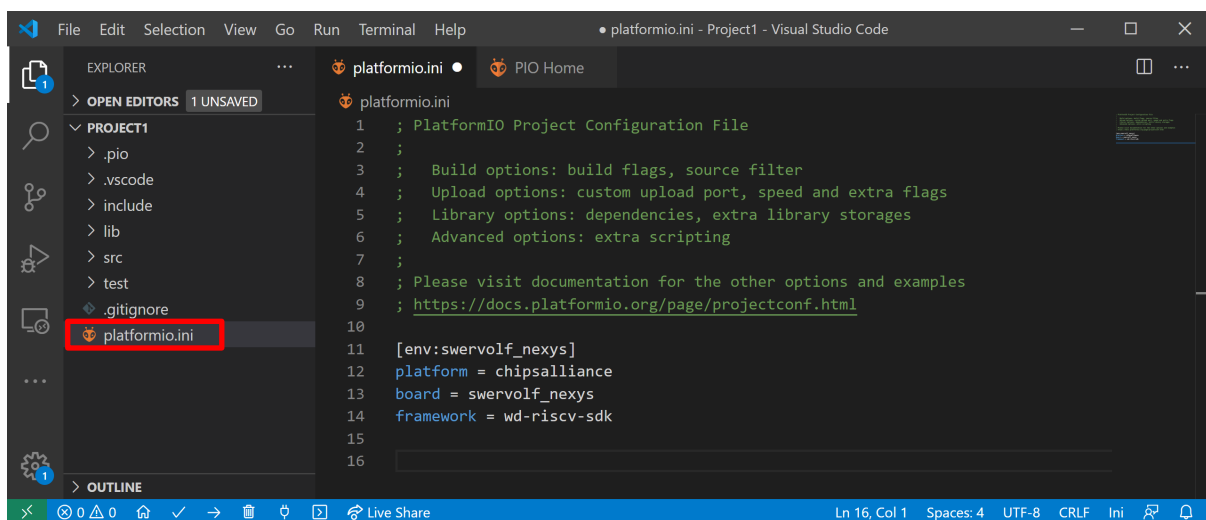


圖4. PlatformIO初始化檔：platformio.ini

將以下程式碼行新增到platformio.ini檔，如圖5所示：

```
board_build.bitstream_file =
[RVfpgaPath]/RVfpga/Labs/Lab1/Project1/Project1.runs/impl_1/rvfpganexys.bit
```

（請記得將[RVfpgaPath]替換為此資料夾在您電腦上的位置。）此程式碼行指示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。

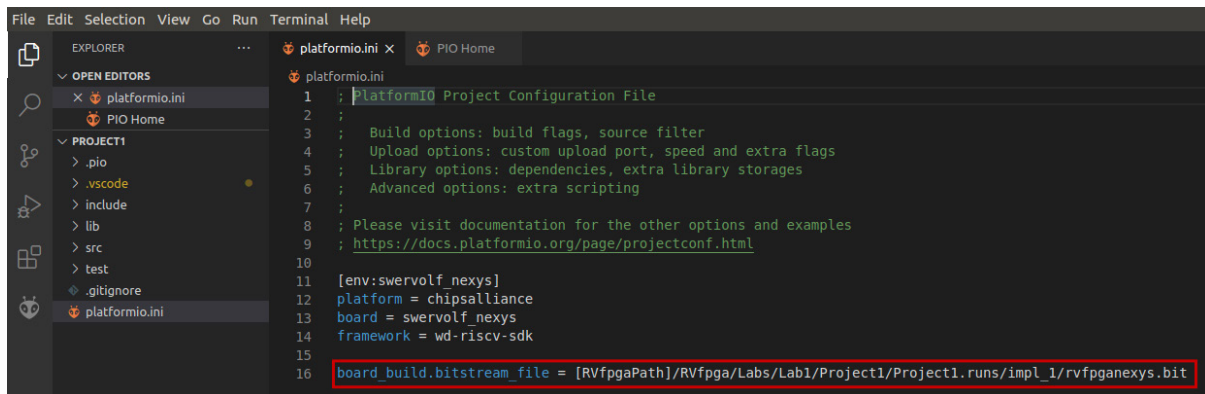


圖5. RVfpgaNexys位元串流檔案 (rvfpganexys.bit) 的新增位置

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C程式

現在需要編寫C程式。按一下「File」（檔案）→「New File」（新建檔案）（請參閱圖6）

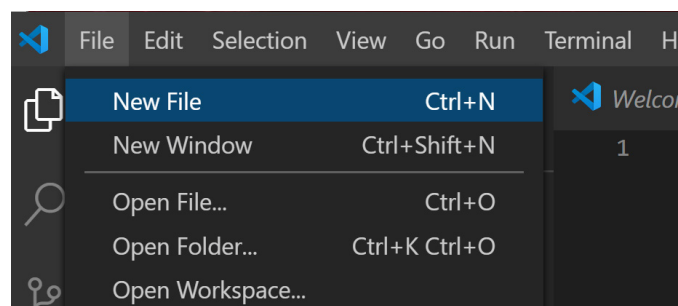


圖6. 在專案中新增檔案

將開啟一個空白視窗。在該視窗中輸入（或複製/貼上）以下C程式（請參閱圖7）。該程式會在LED上顯示開關的值。

```
// memory-mapped I/O addresses
#define GPIO_SWs      0x80001400
#define GPIO_LEDs     0x80001404
#define GPIO_INOUT    0x80001408

#define READ_GPIO(dir) (*(volatile unsigned *)dir)
#define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }

int main ( void )
{
    int En_Value=0xFFFF, switches_value;

    WRITE_GPIO(GPIO_INOUT, En_Value);

    while (1) {
        switches_value = READ_GPIO(GPIO_SWs);    // read value on switches
        switches_value = switches_value >> 16;  // shift into lower 16 bits
        WRITE_GPIO(GPIO_LEDs, switches_value);  // display switch value on LEDs
    }
}
```

```
return(0);
}
```

為了方便起見，以下檔案也提供該程式：

`[RVfpgaPath]/RVfpga/Labs/Lab2/DisplaySwitches.c`

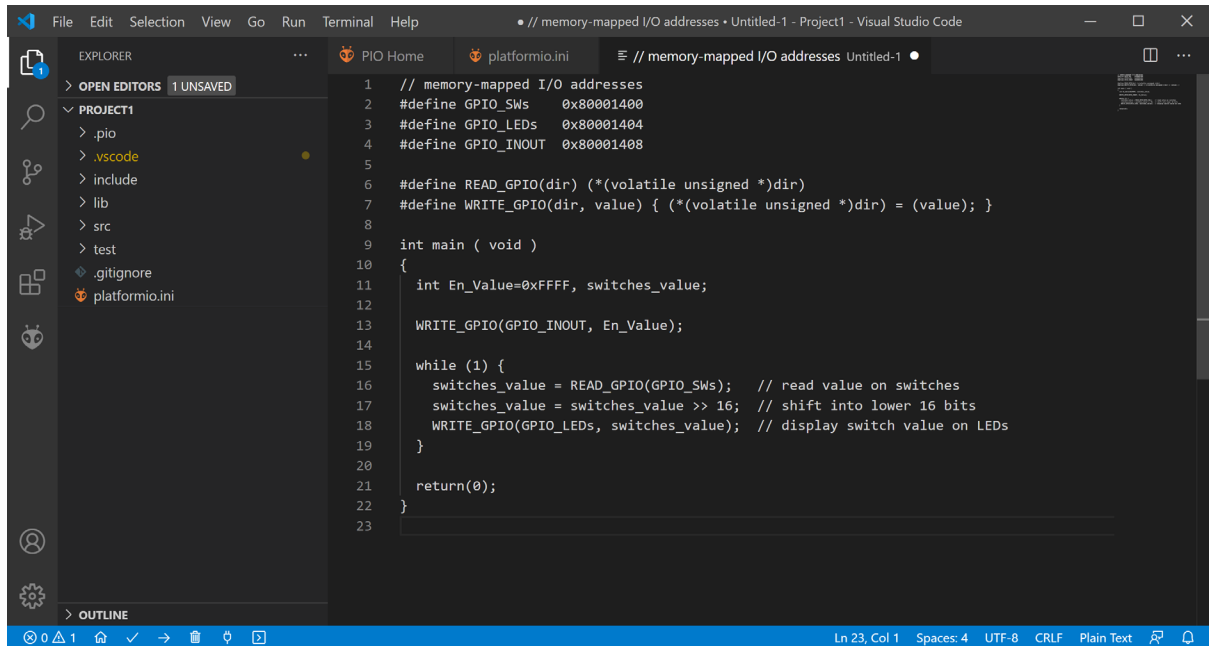


圖7. 輸入C程式

將程式輸入窗格後，按**Ctrl-s**儲存檔案。將其命名為**DisplaySwitches.c**，並儲存到Project1目錄下的src資料夾中（參見圖8）。

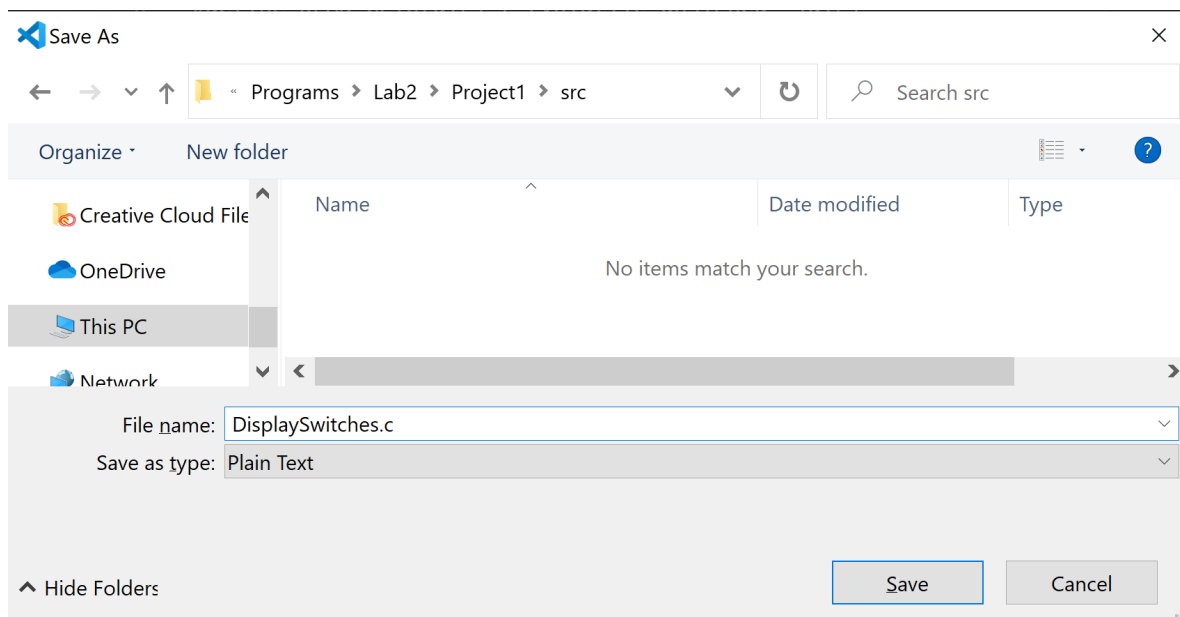


圖8. 將檔案另存為**DisplaySwitches.c**

此程式首先使用以下程式碼行，定義與Nexys A7 FPGA開發板上的LED和開關相連接的記憶體映射I/O暫存器的位址：

```
#define GPIO_SWs      0x80001400
#define GPIO_LEDs     0x80001404
#define GPIO_INOUT    0x80001408
```

通過讀取映射到位址0x80001400的暫存器可以獲取開關的值，通過寫入映射到位址0x80001404的暫存器可以將值顯示在LED上。開關值位於暫存器的上半部，LED值位於暫存器的下半部。

GPIO_INOUT暫存器用於指定通用I/O（General-Purpose I/O，GPIO）的某個位元是輸入還是輸出。低位元16個GPIO引腳（15:0）連接到Nexys A7開發板上的16個LED。高位元16個GPIO引腳（31:16）用於16個開發板上開關。0表示輸入，而1表示輸出。因此，0xFFFF將寫入GPIO_INOUT暫存器，以便將開關作為RVfpgaNexys的輸入，將LED作為由RVfpgaNexys驅動的輸出。

圖9顯示了Nexys A7 FPGA開發板上的LED和開關，以及USB連接器、電源開關、按鈕和7段顯示器的物理位置。

請注意，在實驗6中，我們對GPIO功能和RVfpgaNexys GPIO硬體有詳細的介紹。在後續的實驗中（實驗6-10），我們還會探討如何使用其他開發板週邊設備，例如按鈕和7段顯示器。

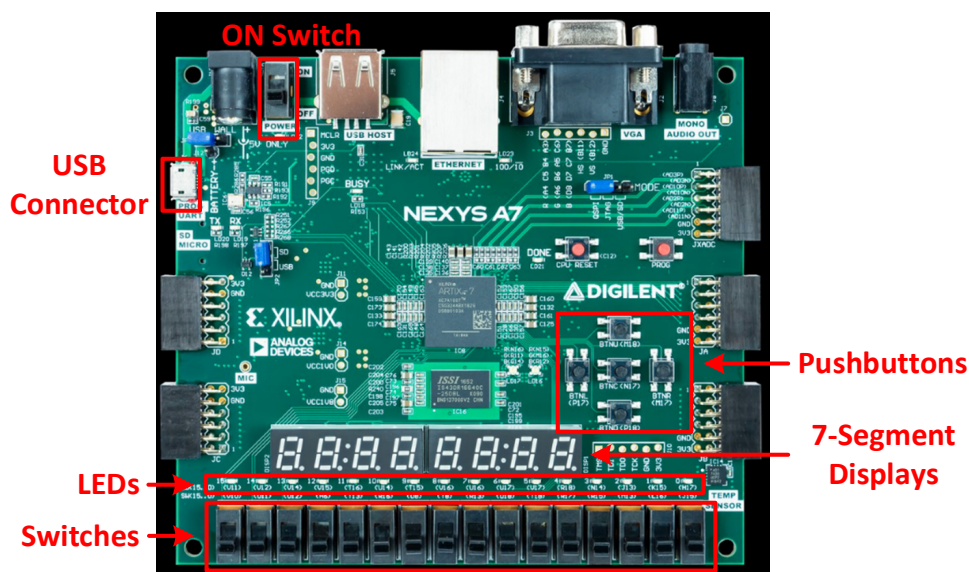


圖9. Digilent的Nexys A7 FPGA開發板的I/O介面
(開發板圖片來源：<https://reference.digilentinc.com/>)

在定義LED和開關的記憶體映射I/O位址後，程式將執行以下操作：

1. 通過執行以下程式碼，將高位元16個GPIO引腳（與開關相連接）定義為輸入（方法是將GPIO_INOUT暫存器的上半部設定為0），將低位元16個GPIO引腳（與LED相連接）定義為輸出（方法是將GPIO_INOUT暫存器的下半部設定為1）：

```
int En_Value=0xFFFF;

WRITE_GPIO(GPIO_INOUT, En_Value);
```

2. 通過執行以下程式碼，重複讀取開關的值並將該值寫入LED。回想一下，開關的值會讀入記憶體映射I/O暫存器的上半部，因此該值必須先右移16位元，再寫入與LED物理連接的記憶體映射I/O暫存器。

```
while (1) {
    switches_value = READ_GPIO(GPIO_SWs);    // read value on switches
    switches_value = switches_value >> 16;    // shift into lower 16 bits
    WRITE_GPIO(GPIO_LEDs, switches_value);    // display switch value on LEDs
}
```

READ_GPIO和WRITE_GPIO巨集分別在指定的記憶體映射I/O位址處讀取或寫入值。

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。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），如圖10所示。

附註：如果使用的是Windows系統並且尚未更換Nexys A7 FPGA開發板驅動程式，請按照「入門指南」附錄B中的說明進行操作。

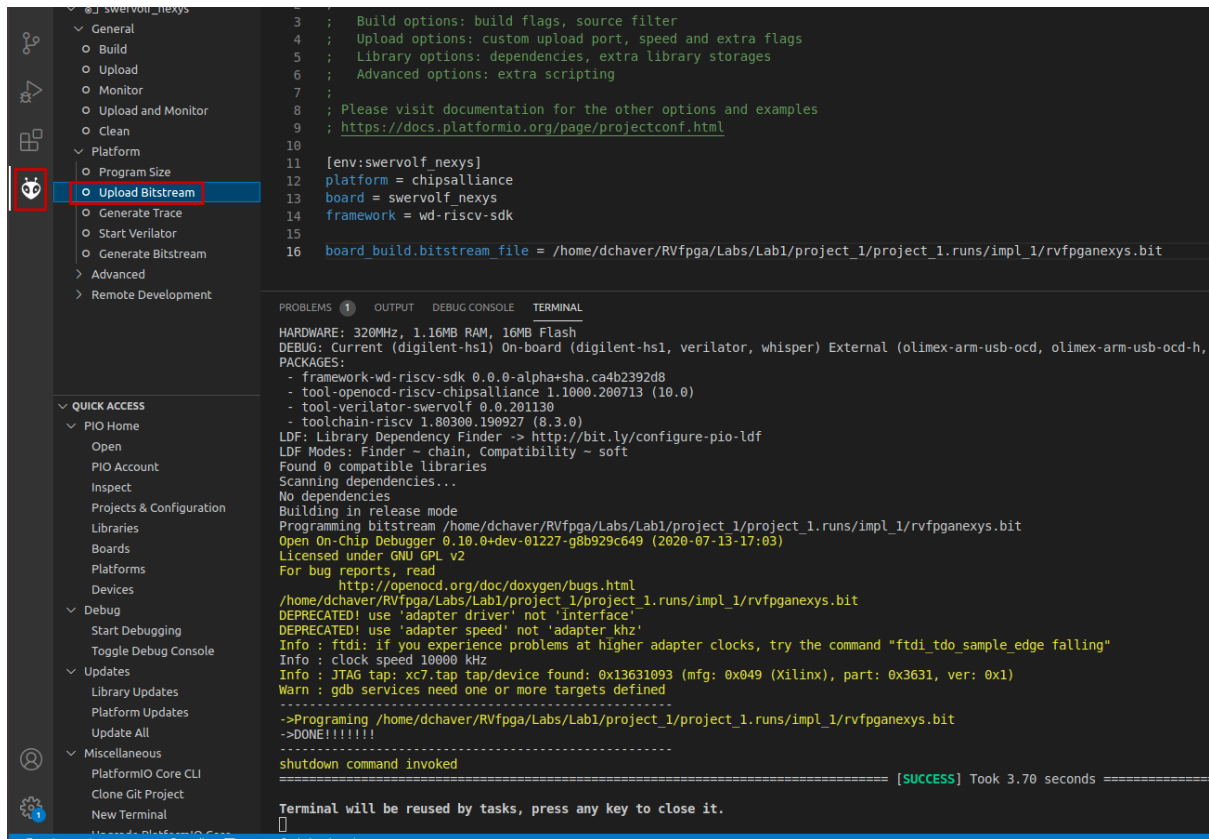



圖10. 使用PlatformIO將RVfpgaNexys上傳到Nexys A7 FPGA開發板上

也可以通過PlatformIO終端機視窗下載RVfpgaNexys，如圖11所示。按一下PlatformIO視窗底部的「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
```

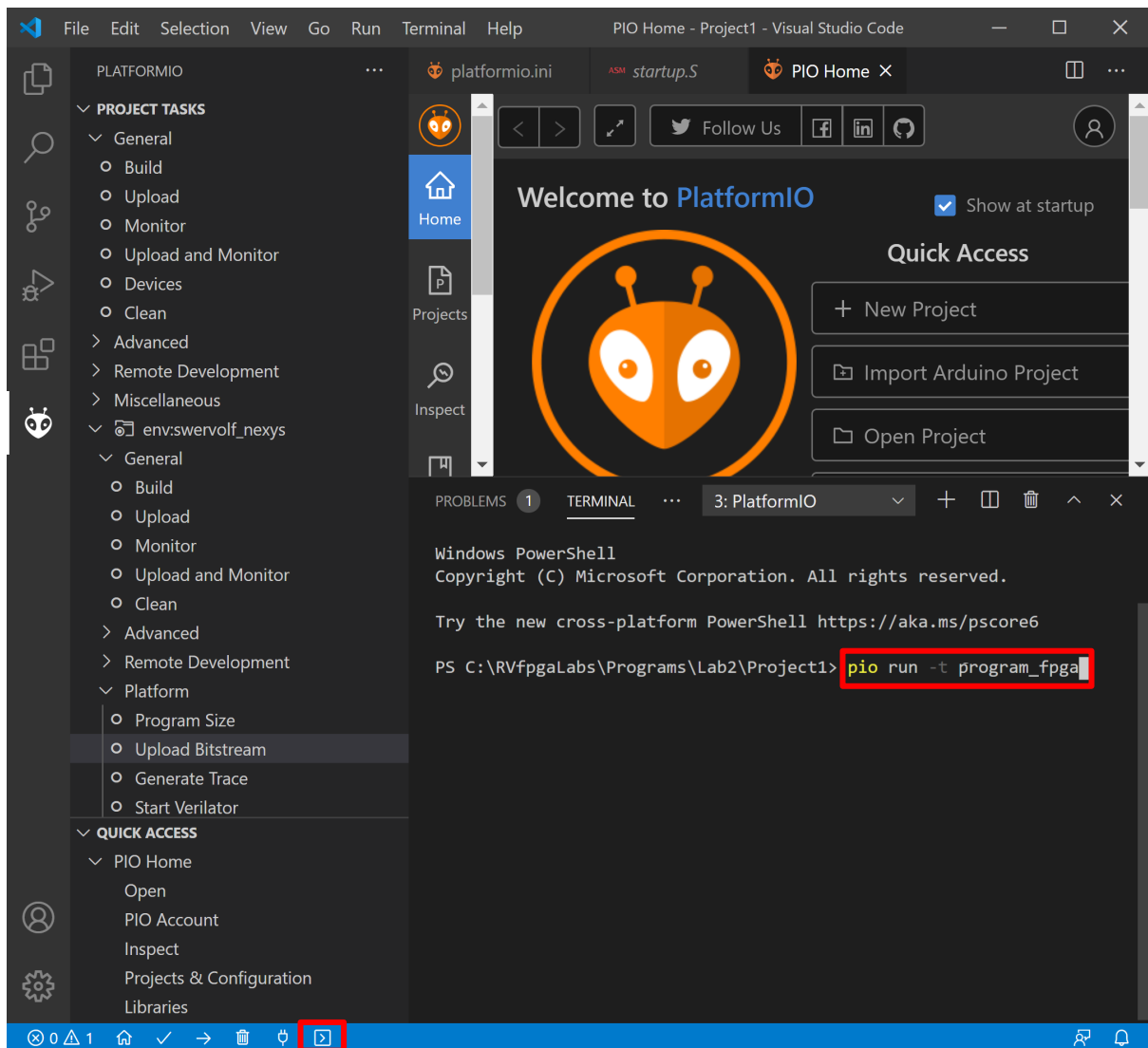


圖11. 使用PlatformIO終端機將RVfpgaNexys上傳到Nexys A7 FPGA開發板上

第4步. 編譯、下載和執行C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（請參閱圖12）。

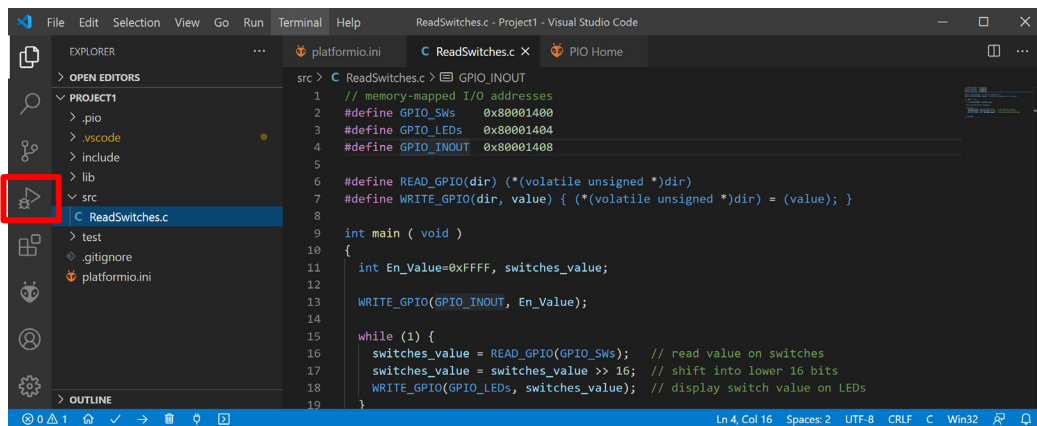


圖12. 在RVfpgaNexys上執行程式

現在，按一下「Start Debugging」（開始偵錯）按鈕（請參閱圖13）。

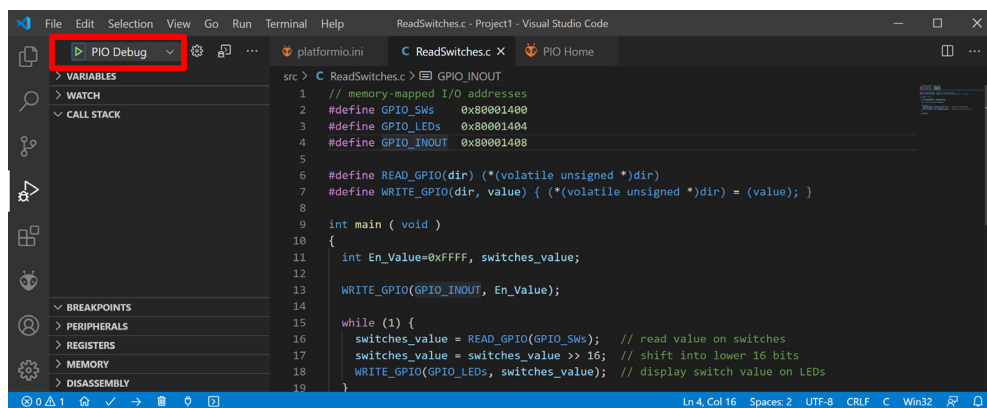


圖13. 開始執行和偵錯程式

程式將先編譯，然後下載至Nexys A7開發板的FPGA上所執行的RVfpgaNexys。（請注意，終端機可能會出現sys/cdefs.h檔案缺失的陳述式，但程式仍可正常執行。）此時即可開始執行和偵錯程式（請參閱圖14）。

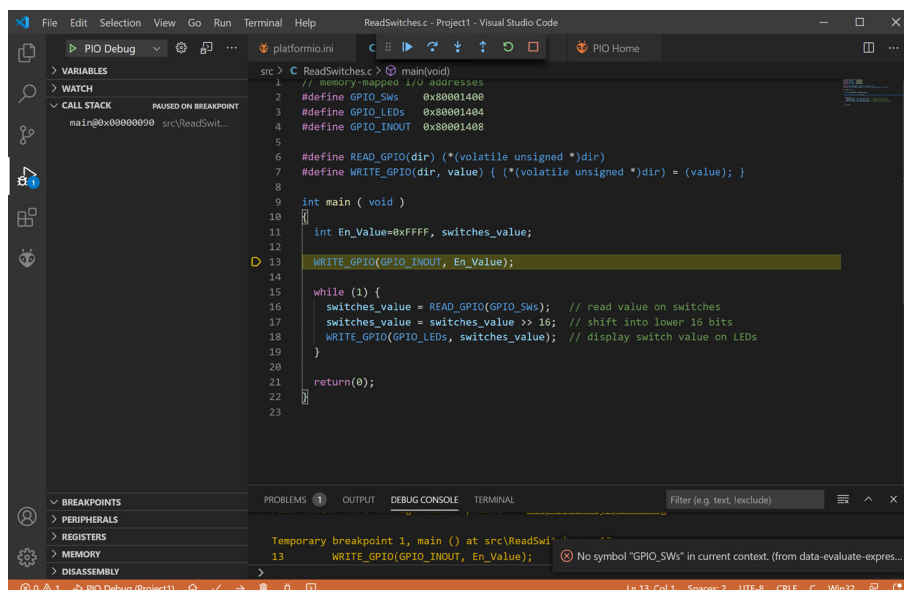


圖14. 在RVfpgaNexys上執行的程式

如「RVfpga入門指南」所述，要控制偵錯工作階段，請使用編輯器頂端位置附近的偵錯工具列（請參閱圖15）。選項如下：


1. **Continue**（繼續）：執行程式，直至達到下一個中斷點。
2. **Breakpoints**（中斷點）：可在編輯器中按一下行號左側予以新增。
3. **Step Over**（單步跳過）：執行目前行，然後停止。
4. **Step Into**（單步進入）：執行目前行，如果目前行包含函數呼叫，將跳轉到該函數並停止。
5. **Step Out**（單步跳出）：執行所在函數中的所有程式碼，然後在該函數返回結果時停止。
6. **Restart**（重新啟動）：從程式的開頭重新啟動偵錯工作階段。
7. **Stop**（停止）：停止偵錯工作階段並返回正常編輯模式。請注意，按下「Stop」（停止）按鈕時，程式將繼續在RVfpgaNexys上執行，但偵錯工作階段將終止。
8. **Pause**（暫停）：暫停執行。程式執行時，「Continue」（繼續）按鈕將替換為「Pause」（暫停）按鈕。



圖15. 偵錯工具

在左側提要欄位中，可以檢視偵錯工具選項。提供以下選項：

- **Variables**（變數）：列出程式中存在的區域、全域和靜態變數及其變數值。
- **Call Stack**（呼叫堆疊）：顯示目前正在執行的函數、呼叫函數（如果存在），以及目前指令在記憶體中的位置。
- **「Breakpoints」**（斷點）：顯示所有設定的斷點並強調顯示其行號。可在此部分中管理中斷點。也可以暫時停用中斷點，而無需通過切換核取方塊將其刪除。
- **Peripherals**（週邊設備）：顯示裝置的記憶體映射週邊設備的暫存器狀態。
- **Registers**（暫存器）：列出存在於處理器的每個暫存器中的目前值。
- **Memory**（記憶體）：顯示特定記憶體位址的內容。
- **Disassembly**（反組合語言程式碼）：顯示特定函數的組合語言程式碼，對於C語言等較高階的程式碼，此選項支援檢視組合語言程式碼以逐行偵錯指令。

例如，按一下第18行左側區域，可在將開關的值寫入LED之前設定一個中斷點，如圖16所示。然後，按一下「Continue」（繼續）按鈕（或按下F5）執行程式。程式將繼續執行，直至到達設定的中斷點。（要刪除現有中斷點，只需按一下行號左側的該中斷點。）

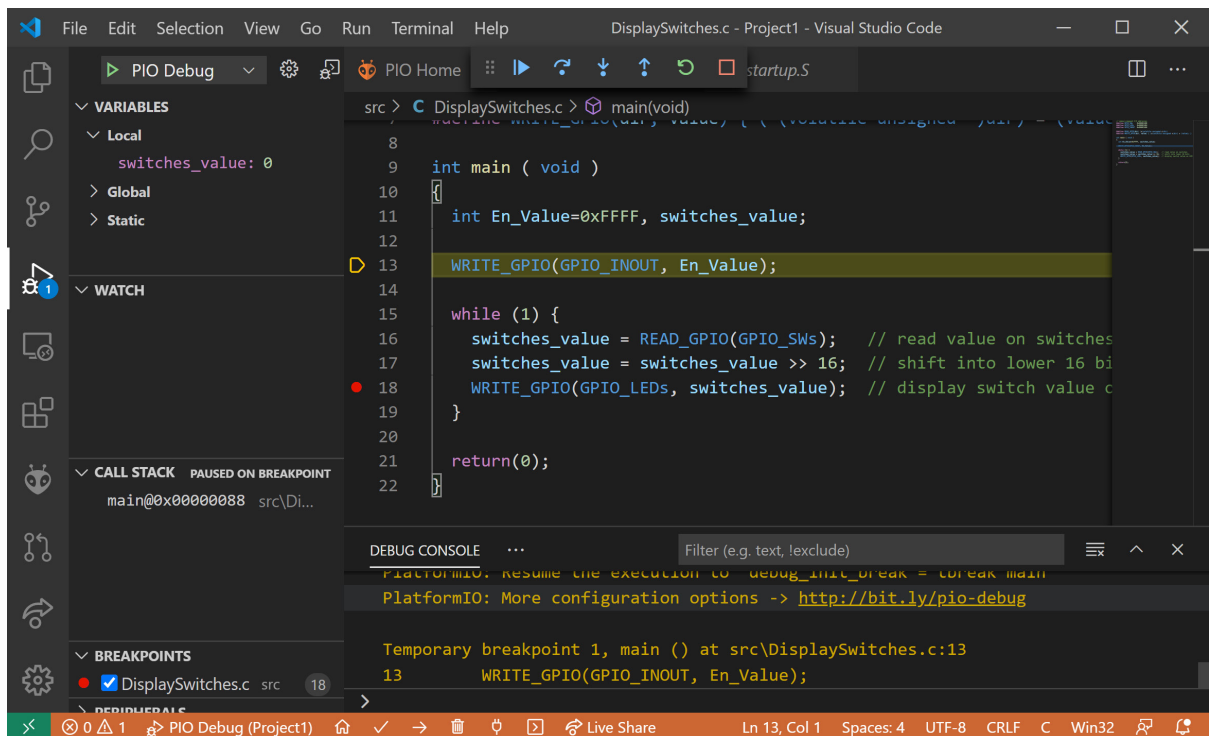


圖16. 在LED上顯示值之前設定中斷點

到達中斷點後，展開左側窗格中的「Variables」（變數）部分，檢視變數switchs_value的值，如圖17所示。在這種情況下，開關的值為1029 = 0x405（二進位形式為0000_0100_0000_0101），對應於以下模式：

Switches[15:0] = OFF-OFF-OFF-OFF-OFF-ON-OFF-OFF-OFF-OFF-OFF-OFF-OFF-ON-OFF-ON

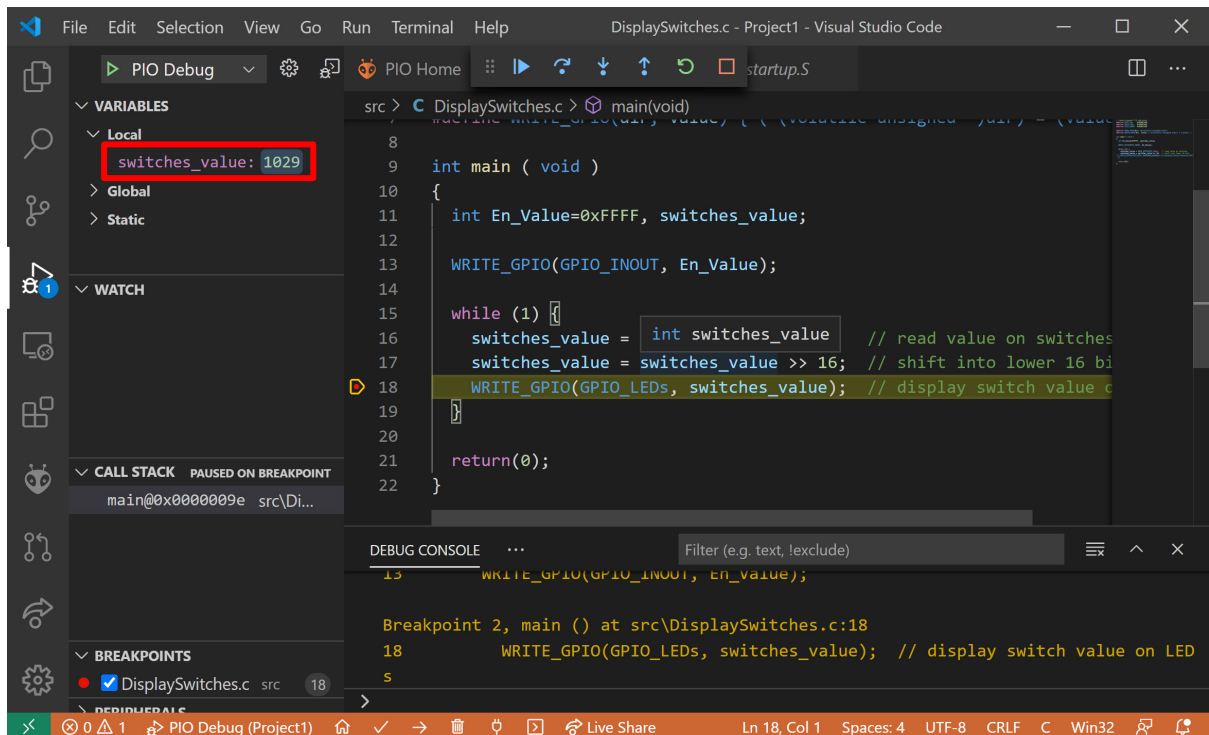


圖17. 檢視變數值

還可以檢視通過C程式產生的RISC-V組合語言程式碼。要執行此操作，請按一下「DISASSEMBLY」（反組合語言程式碼）→「Switch to assembly」（切換至組合語言程式碼），如圖18所示。

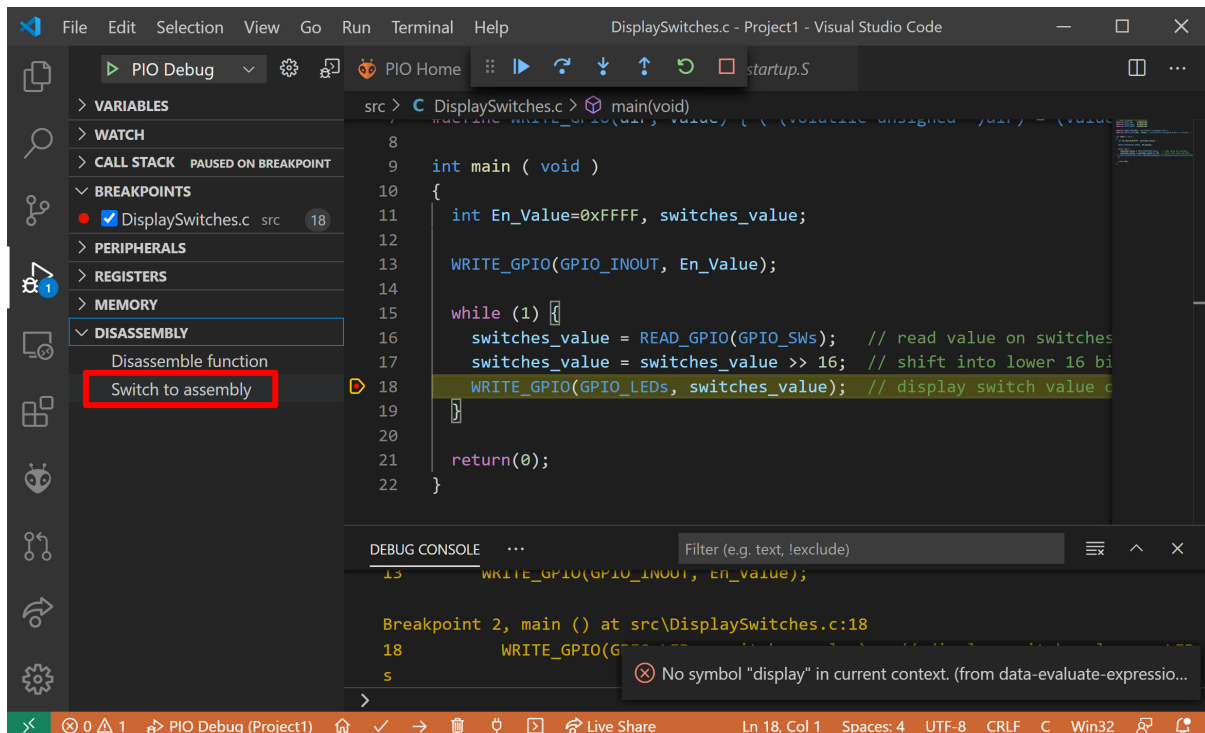


圖18. 檢視RISC-V組合語言程式碼

現在，RISC-V組合語言程式碼會顯示在檢視窗格中，如圖19所示。組合語言程式碼部分顯示指令的記憶體位址、機器程式碼和組合語言程式碼。可以看出，C程式碼被編譯為壓縮指令（16位元指令）和32位元指令的組合。組合語言程式碼及相關註釋如下。

# address	# machine code	# instruction	
0x00000090:	37 17 00 80	lui a4,0x80001	# Base address for I/O
0x00000094:	c1 67	lui a5,0x10	# a5 = 0x10000 - 1 (=0xFFFF)
0x00000096:	fd 17	addi a5,a5,-1	
0x00000098:	23 24 f7 40	sw a5,1032(a4)	# I/O direction: [0x80001408]=0xFFFF
0x0000009c:	37 17 00 80	lui a4,0x80001	# Base address for I/O (redundant)
0x000000a0:	83 27 07 40	lw a5,1024(a4)	# Read switches: a5 = [0x80001400]
0x000000a4:	c1 87	srai a5,a5,0x10	# Move switch value to lower 16 bits
0x000000a6:	23 22 f7 40	sw a5,1028(a4)	# Write LEDs: [0x80001404] = a5
0x000000aa:	cd bf	j 0x9c <main+12>	# repeat

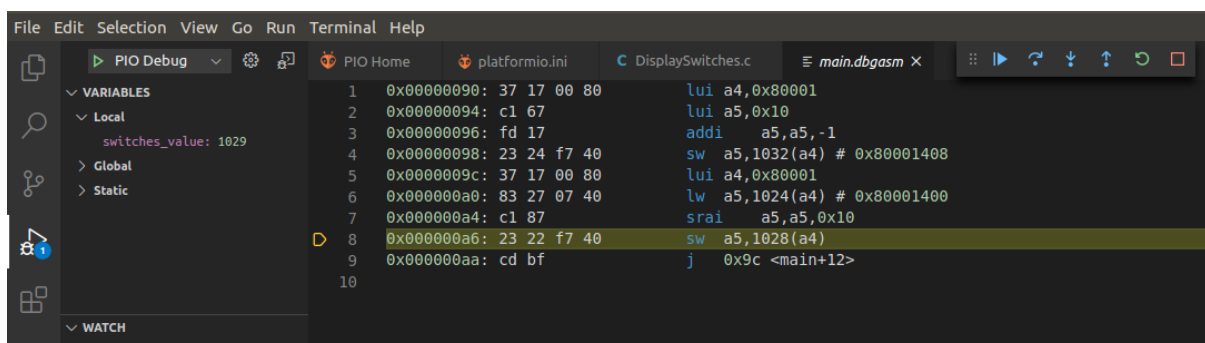



圖19. 檢視RISC-V組合語言程式碼

按一下「**DISASSEMBLY**」（反組合語言程式碼）→「**Switch to code**」（切換至程式碼）再次檢視C程式碼。

執行/偵錯程式完成後，按下「**Stop**」（停止）按鈕  （或Shift-F5）

停止偵錯工作階段，然後按一下最左側提要欄位頂端的  返回「**Explorer**」（檔案總管）視窗。請注意，**程式將繼續在RVfpgaNexys上執行**，只有偵錯工作階段被終止。按一下頂端功能表列中的「**File**」（檔案）→「**Close Folder**」（關閉資料夾）關閉專案。

3. 使用printf和序列監視器

在程式中使用**print**陳述式是追蹤程式進度或向使用者提供反饋（例如計算結果）的一種有效方式。回想一下，在「**RVfpga入門指南**」（第6.F節的範例***HelloWorld_C-Lang***）中，可以使用**printfNexys**函數，該函數類似於典型的C程式中的**printf**函數。為此，必須使用可為給定處理器和開發板提供常用函數的**Western Digital PSP**和**BSP**（處理器支援套件和開發板支援套件）；在這種情況下，應使用**SweRV EH1**核心和**Nexys A7 FPGA**開發板。

在**[RVfpgaPath]/RVfpga/Labs/Lab2**資料夾中建立一個名為**PrintfExample**的PlatformIO專案。將以下程式新增到該專案（請參閱圖20）。將程式檔命名為**PrintfExample.c**。

為方便您使用，也可通過以下路徑獲取程式：

[RVfpgaPath]/RVfpga/Labs/Lab2/PrintfExample.c

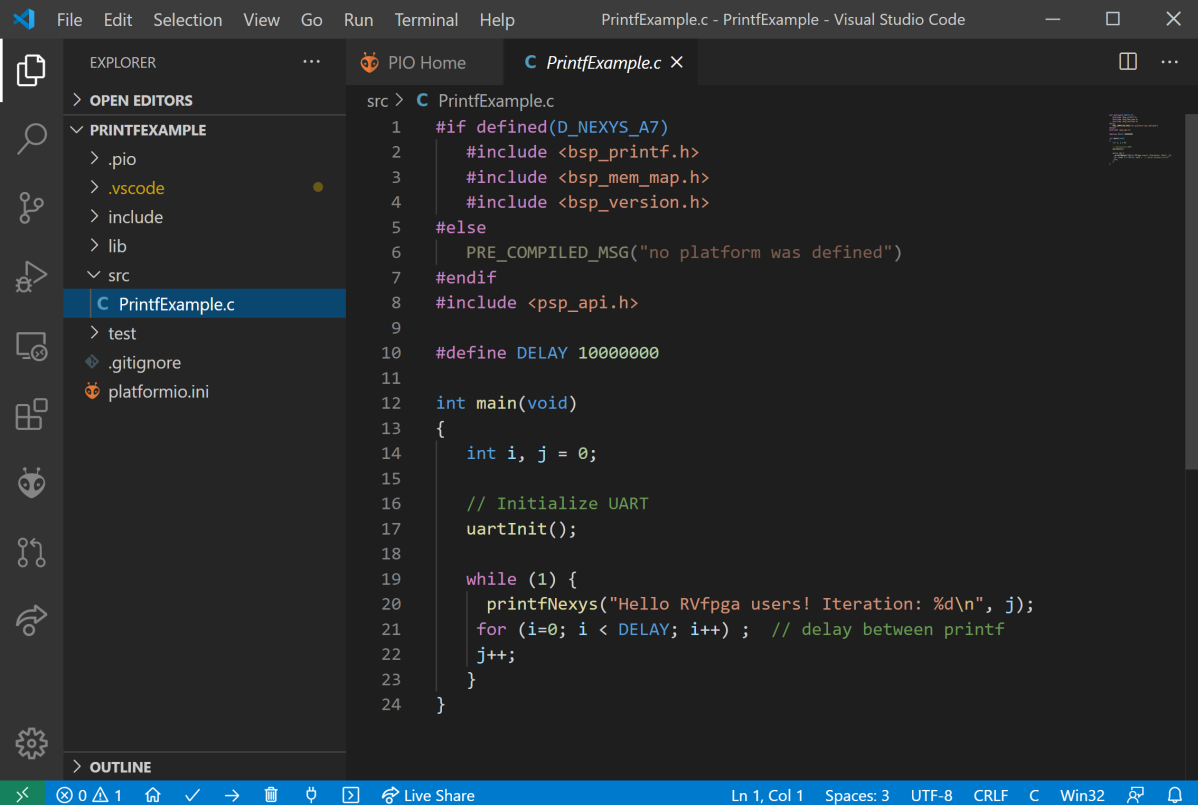
```
#if defined(D_NEXYS_A7)
#include <bsp_printf.h>
#include <bsp_mem_map.h>
#include <bsp_version.h>
#else
    PRE_COMPILED_MSG("no platform was defined")
#endif
#include <psp_api.h>

#define DELAY 10000000

int main(void)
{
    int i, j = 0;

    // Initialize UART
    uartInit();

    while (1) {
        printfNexys("Hello RVfpga users! Iteration: %d\n", j);
        for (i=0; i < DELAY; i++) ; // delay between printf's
        j++;
    }
}
```

```

src > C PrintfExample.c
1  #if defined(D_NEXYS_A7)
2  #include <bsp_printf.h>
3  #include <bsp_mem_map.h>
4  #include <bsp_version.h>
5  #else
6  PRE_COMPILED_MSG("no platform was defined")
7  #endif
8  #include <psp_api.h>
9
10 #define DELAY 10000000
11
12 int main(void)
13 {
14     int i, j = 0;
15
16     // Initialize UART
17     uartInit();
18
19     while (1) {
20         printfNexys("Hello RVfpga users! Iteration: %d\n", j);
21         for (i=0; i < DELAY; i++) ; // delay between printf
22         j++;
23     }
24 }

```

圖20. PrintfExample.c

第1-8行（請參閱圖20）是使用printfNexys函數時所需的包含檔。這些檔案由Western Digital的BSP/PSP提供。圖20中的第17行呼叫了uartInit函數；此程式碼行用於初始化UART連接，從而實作RVfpgaNexys（在Nexys A7開發板上執行）與序列監視器之間的通訊。最後，while迴圈搭配使用第20行的printfNexys函數與第21行的延遲，重複向序列監視器寫入資料。

要使用printfNexys函數和UART，必須修改platformio.ini檔，將UART的速度包含在內。將以下程式碼行新增到platformio.ini檔，如圖21所示：

```
monitor_speed = 115200
```


RVfpgaNexys期望UART以115200波特（每秒符號數）的速率進行通訊，因此必須在platformio.ini中設定此速率，如圖21中第16行所示。此外，不要忘記使用board_build.bitstream_file = ... 新增RVfpgaNexys bit檔的位置（如圖21中第18行所示）。

```
platformio.ini
1 ; PlatformIO Project Configuration File
2 ;
3 ; Build options: build flags, source filter
4 ; Upload options: custom upload port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ; Advanced options: extra scripting
7 ;
8 ; Please visit documentation for the other options and examples
9 ; https://docs.platformio.org/page/projectconf.html
10
11 [env:swervolf_nexys]
12 platform = chipsalliance
13 board = swervolf_nexys
14 framework = wd-riscv-sdk
15
16 monitor_speed = 115200
17
18 board_build.bitstream_file = /home/dchaver/RVfpga/Labs/Lab1/project_1/project_1.runs/impl_1/rvfpganexys.bit
```

圖21. 設定UART速度

LINUX：請記住，如果使用的是Linux，需要先通過將使用者新增到dialout、tty和uucp群組來完成系統準備工作（如「入門指南」第6.F節中所述），之後才能使用序列監視器。如果已在GSG中執行了此操作，則一切功能都會正常運作；否則請立即執行此操作。

然後上傳bit檔（如第2節所述）並執行/偵錯程式。

程式開始執行後（必須等到程式開始執行後），按一下PlatformIO視窗底部的序列監視器按鈕（請參閱圖22）。

警告：如果在程式開始執行（並到達第一個臨時中斷點）之前開啟序列監視器，UART將出現問題，無法正常工作。

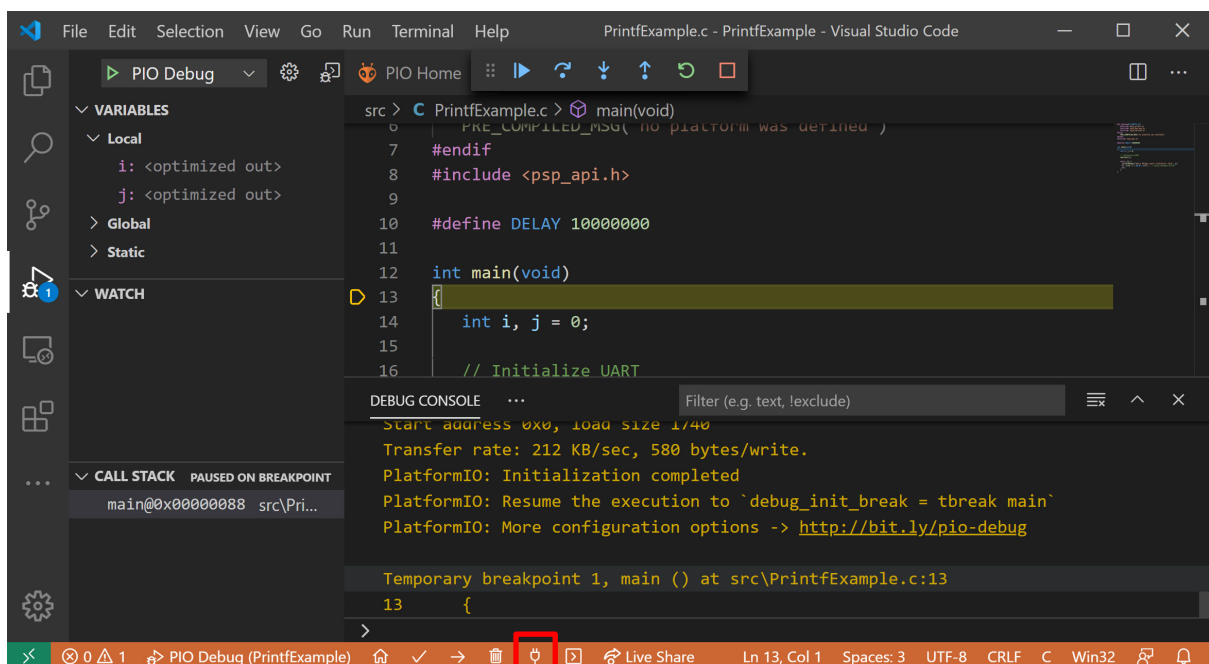



圖22. 啟動序列監視器

然後執行程式：.

您將看到列印字串（Hello RVfpga users!）和迭代數在序列監視器上重複出現，如圖23所示。

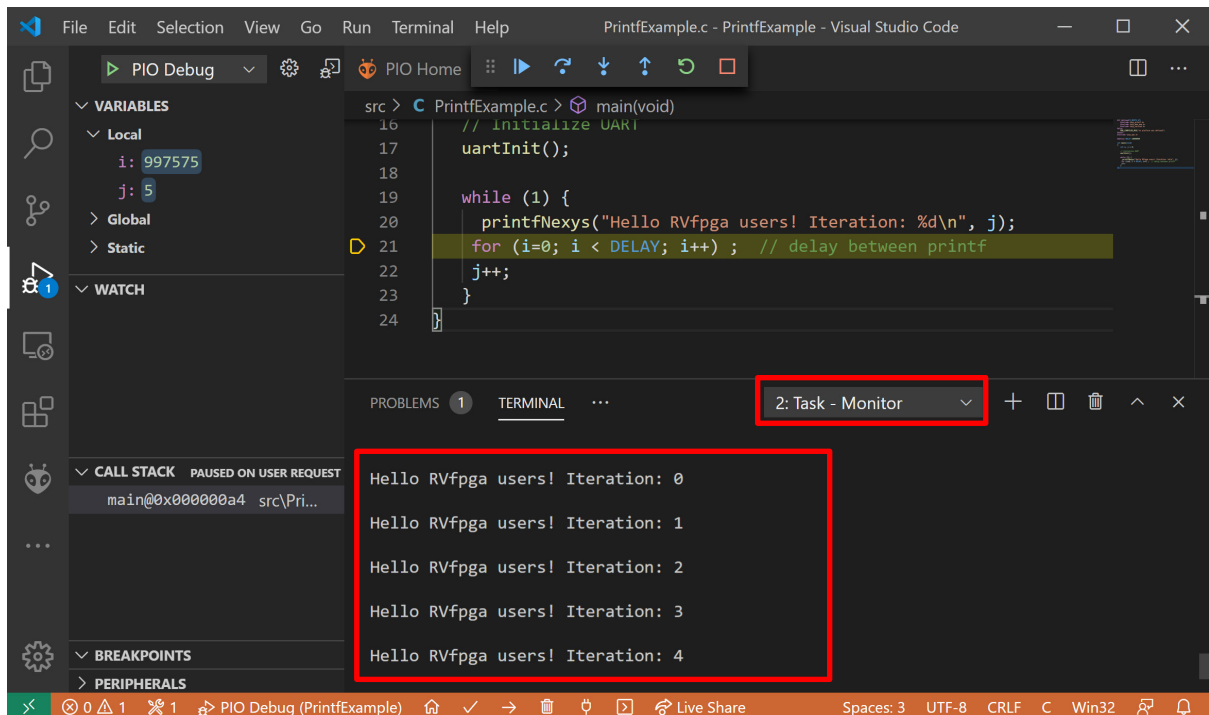


圖23. 在PlatformIO中的序列監視器上輸出printfNexys函數

4. 練習

通過完成以下練習，建立自己的C程式。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，如第2節的第3步所述。

請記住，可以使用Western Digital的BSP函數printfNexys列印任何變數（請參閱第3節）。

還要記住，可以使用Verilator和Whisper在模擬中執行這些程式。

練習1. 編寫一個C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.c。

練習2. 編寫一個C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.c。

練習3. 編寫一個C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.c**。

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，然後又從左向右捲動。
2. 然後，有兩個點亮的LED從右向左捲動，然後又從左向右捲動。
3. 然後，有三個點亮的LED從右向左捲動，然後又從左向右捲動。
4. 依此類推，直到所有LED均點亮。
5. 然後應重複執行該模式。

練習4. 編寫一個C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.c**。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C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.c**。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C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：

$$V(0)=0, V(1)=1, V(i)=V(i-1)+V(i-2) \quad (\text{其中 } i=0, 1, 2, \dots)$$

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1顯示了*i*為0到8時的斐波那契數。

表1. 斐波那契數列

<i>i</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.c**。

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。**C**程式還必須計算**B**中的元素數量，並在程式末尾列印該值。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]**。由於**B**中含有四個元素，因此應在程式末尾列印以下內容：

B中的元素數 = 4。

為此，應使用printfNexys函數。將程式命名為**EvenPositiveNumbers.c**。在**A**中有12個元素時測試程式。

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：

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

用C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.c**。

練習9. 用C語言實作泡泡排序演算法，讓演算法通過以下程序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.c**。

練習10. 用C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。該程式應在程式末尾輸出階乘（ n ）的值。 n 應為程式內靜態定義的變數。將程式命名為**Factorial.c**。