



Imagination大学计划

RVfpga实验2

C语言编程

1. 简介

大多数计算机程序是以高级语言（如C语言）编写的。本实验将展示如何在PlatformIO中创建一个可在RVfpga系统上运行的C语言项目。我们首先提供关于创建和运行C程序的教程。然后，我们会提供一些练习，让您编写自己的C程序。

2. RVfpga的C程序

要使用PlatformIO在RVfpgaNexys上创建和运行C程序，需要完成以下步骤（请记住，也可以使用Verilator和Whisper在仿真中运行这些程序）：

1. 创建RVfpga项目
2. 编写C程序
3. 将RVfpgaNexys下载到Nexys A7 FPGA开发板
4. 编译、下载和运行C程序

步骤1. 创建RVfpga项目

打开VSCode（如“RVfpga入门指南”中所述）。如果启动VSCode时PlatformIO没有自动打开，请单击左侧功能区菜单中的PlatformIO图标，然后单击“PIO Home”（PIO主页）→“Open”（打开）（参见图1）。

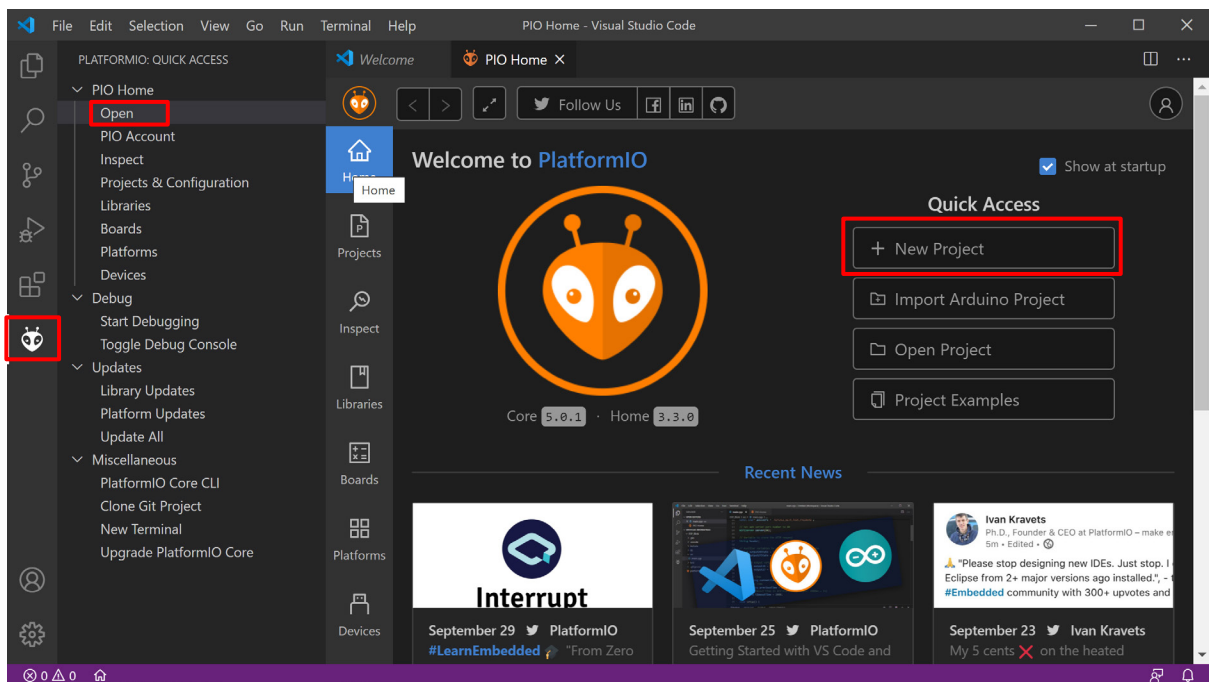


图1. 打开PlatformIO并创建新项目

现在，在“PIO Home”（PIO主页）欢迎窗口中，单击“New Project”（新建项目）（参见图1）。

如图2所示，将项目命名为“Project1”，在“Board”（开发板）部分选择“RVfpga: Digilent Nexys A7”（输入RVfpga，即可出现该选项）。保留框架的默认选项WD-Firmware。WD-Firmware是Western Digital的一款固件，其中包含Freedom-E SDK gcc和gdb以及我们在这些实验中使用的PSP和BSP（处理器支持包和开发板支持包）。取消选中“Use default location”（使用默认位置），然后将项目放到以下路径：

`[RVfpgaPath]/RVfpga/Labs/Lab2`

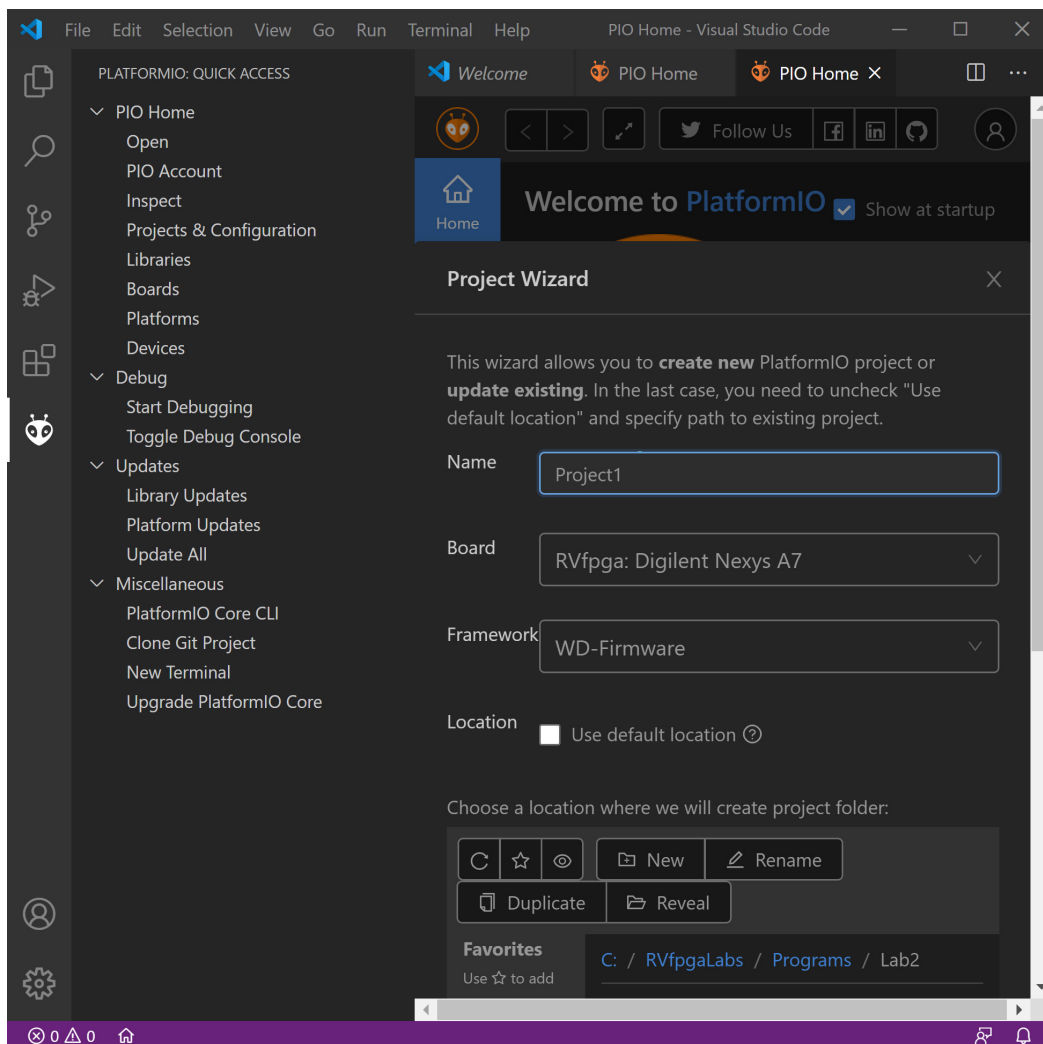


图2. 为项目命名并选择开发板和项目文件夹

然后点击窗口底部的“Finish”（完成）（参见图3）。

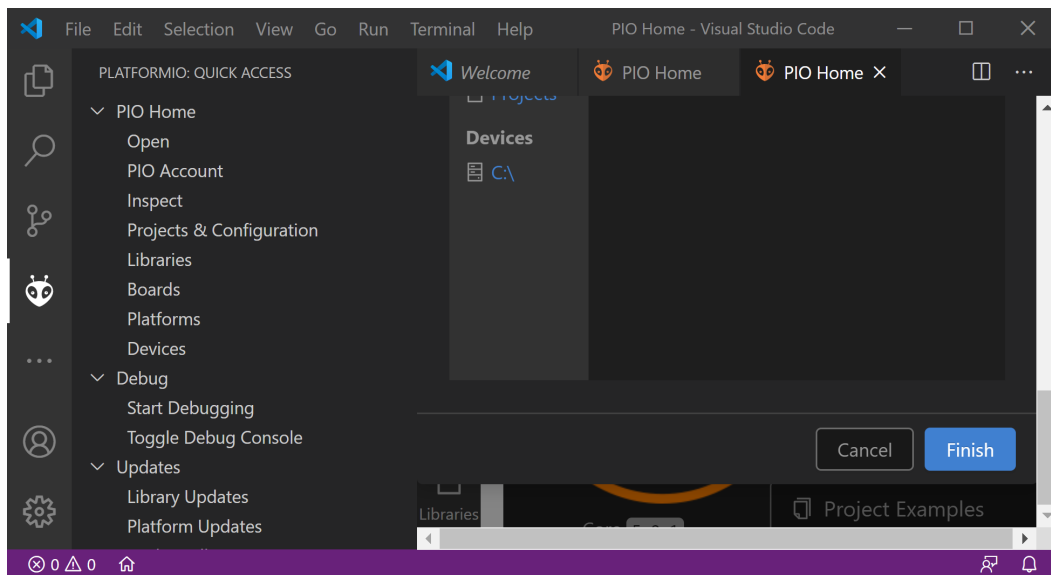


图3. 完成项目创建

在左侧的“Explorer”（资源管理器）窗格中（可能需要展开），双击platformio.ini打开该文件（参见图4）。该文件为PlatformIO初始化文件。

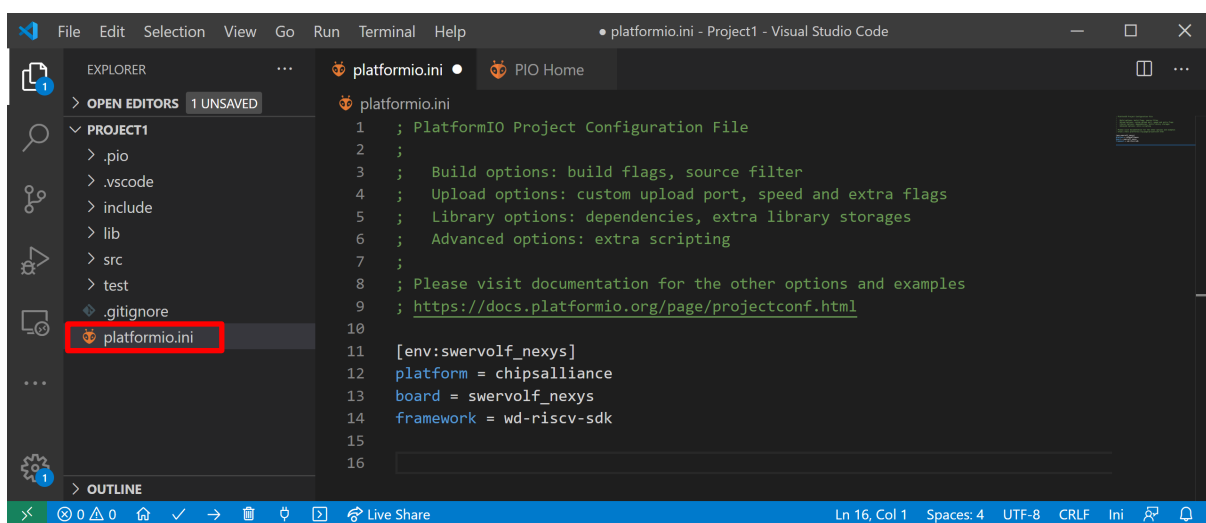


图4. PlatformIO初始化文件：platformio.ini

将以下代码行添加到platformio.ini文件，如图5所示：

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

（请记得将[RVfpgaPath]替换为此文件夹在您计算机上的位置。）此行代码指示PlatformIO可以在哪里找到要加载到FPGA上的比特流文件。上述路径是在实验1中创建的比特流的位置。

（如果尚未完成实验1，可以使用“入门指南”中随附的RVfpgaNexys比特流，路径为：[RVfpgaPath]/RVfpga/src/rvfpganexys.bit。）按Ctrl-s保存platformio.ini文件。

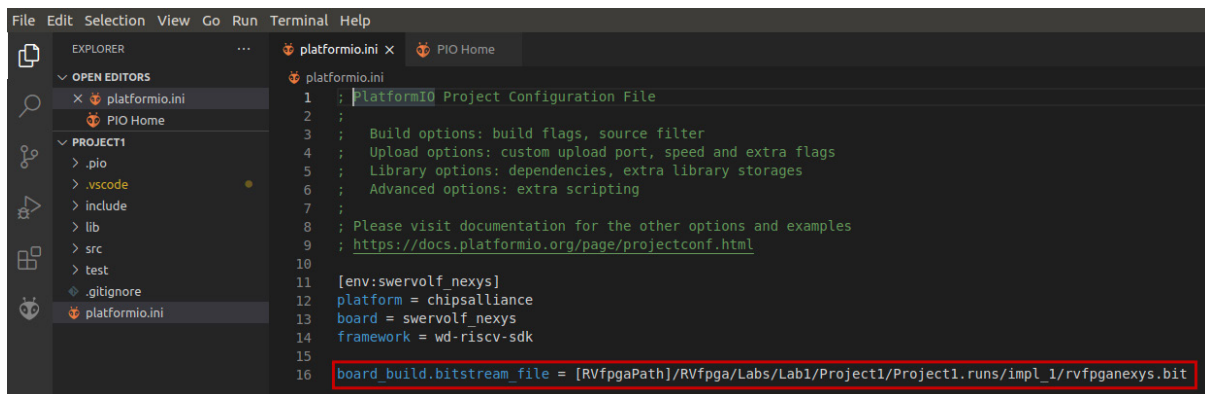


图5. RVfpgaNexys比特流文件 (rvfpganexys.bit) 的添加位置

请记住，“入门指南”的示例中使用了更完整的`platformio.ini`文件。如果要使用任何需要额外命令的功能（例如Verilator仿真器的路径、串行控制台的配置、Whisper调试工具等），则可以使用上述示例中的`platformio.ini`。

步骤2. 编写C程序

现在，需要编写C程序。单击“File”（文件）→“New File”（新建文件）（参见图6）

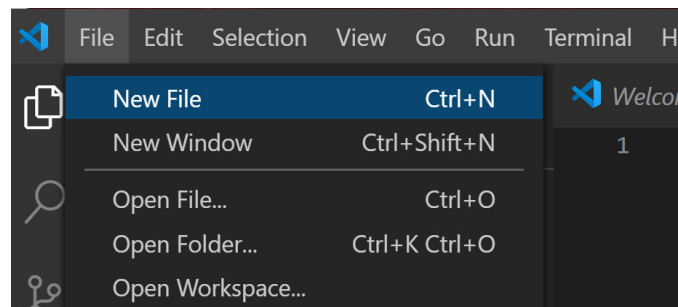


图6. 向项目中添加文件

将打开一个空白窗口。在该窗口中输入（或复制/粘贴）以下C程序（参见图7）。该程序会在LED上显示开关的值。

```
// 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);
}
```

为了方便起见，以下文件中也提供该程序：

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

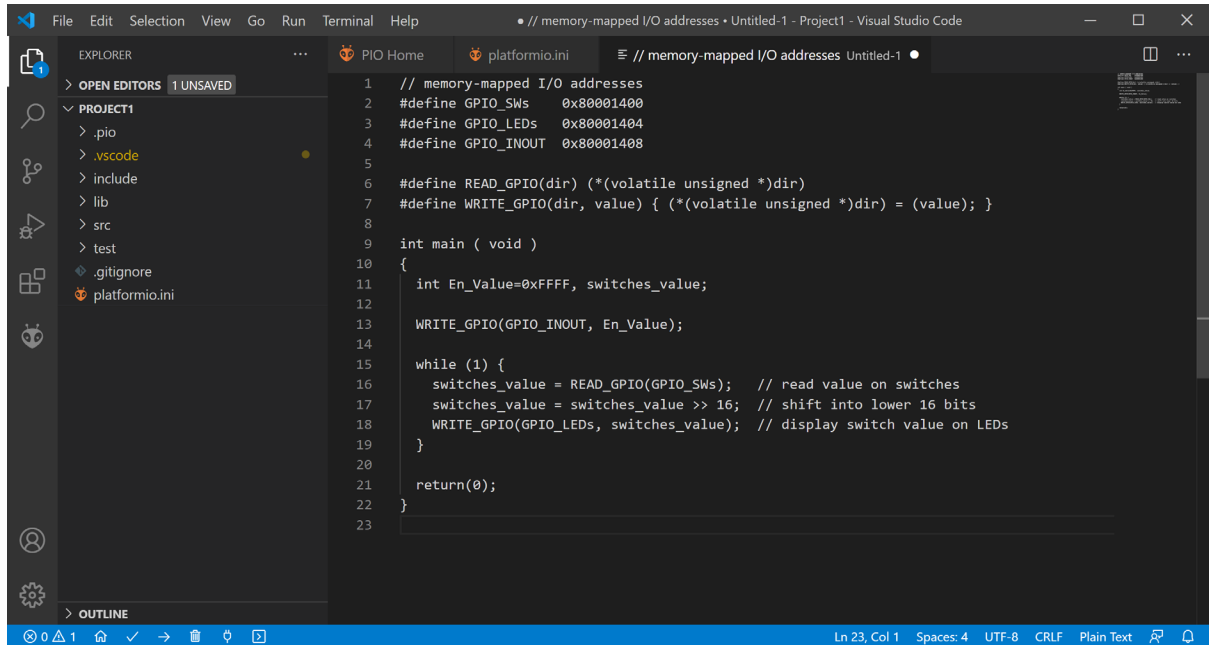


图7. 输入C程序

将程序输入窗格后，按**Ctrl-s**保存文件。将其命名为**DisplaySwitches.c**，并保存到**Project1**目录下的**src**文件夹中（参见图8）。

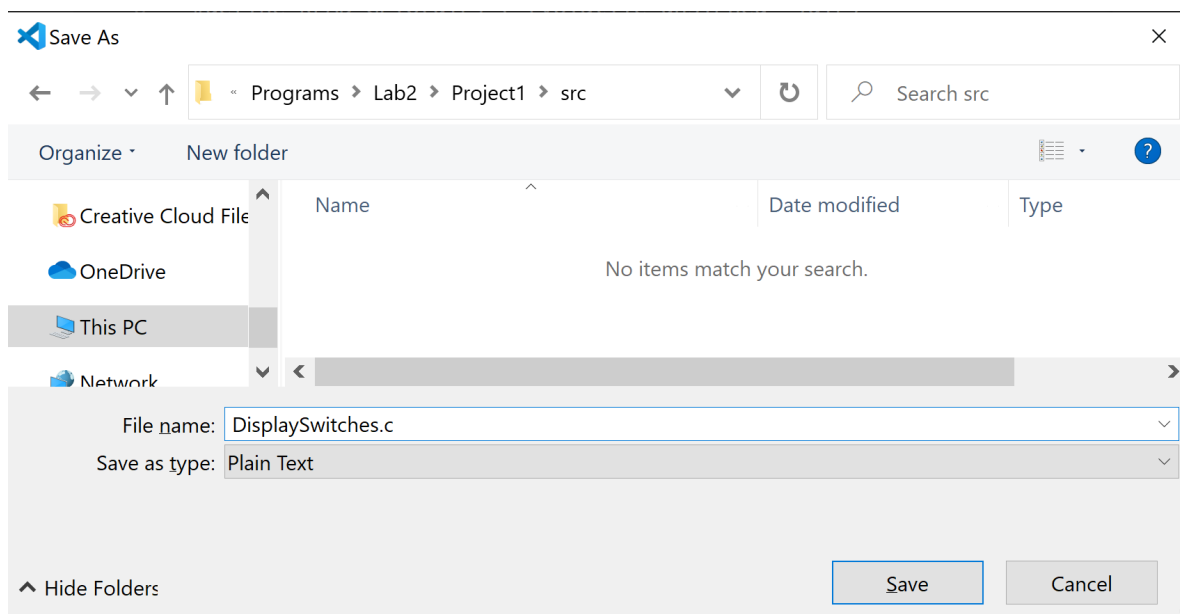


图8. 将文件另存为**DisplaySwitches.c**

此程序首先使用以下代码行定义与Nexys A7 FPGA开发板上的LED和开关相连接的存储器映射I/O寄存器的地址：

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

通过读取映射到地址0x80001400的寄存器可以获取开关的值，通过写入映射到地址0x80001404的寄存器可以将值显示在LED上。开关值位于寄存器的上半部分，LED值位于寄存器的下半部分。

GPIO_INOUT寄存器用于指定通用I/O（General-Purpose I/O，GPIO）的某个位是输入还是输出。低位16个GPIO引脚（15:0）连接到Nexys A7开发板上的16个LED。高位16个GPIO引脚（31:16）用于16个板上开关。0表示输入，而1表示输出。因此，0xFFFF将写入GPIO_INOUT寄存器，以便将开关作为RVfpgaNexys的输入，将LED作为由RVfpgaNexys驱动的输出。

图9显示了Nexys A7 FPGA开发板上的LED和开关，以及USB连接器、接通开关、按钮和7段显示屏的物理位置。

请注意，在实验6中，我们对GPIO功能和RVfpgaNexys GPIO硬件有详细的介绍。在后续的实验中（实验6-10），我们还会探讨如何使用其他开发板外设，例如按钮和7段显示屏。

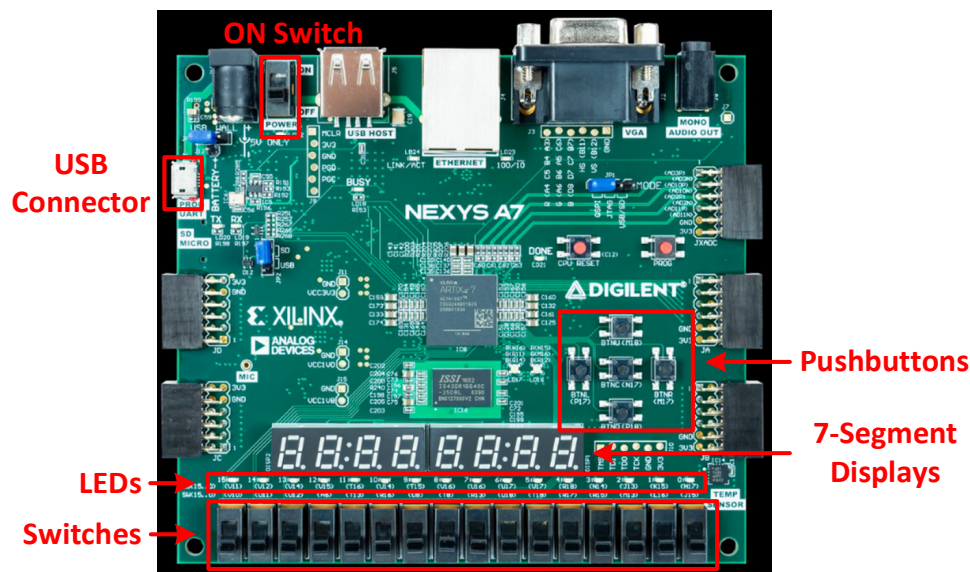


图9. Digilent的Nexys A7 FPGA开发板的I/O接口
(开发板图片来源: <https://reference.digilentinc.com/>)

在定义LED和开关的存储器映射I/O地址后，程序将执行以下操作：

1. 通过执行以下代码，将高位16个GPIO引脚（与开关相连接）定义为输入（方法是将GPIO_INOUT寄存器的上半部分设置为0），将低位16个GPIO引脚（与LED相连接）定义为输出（方法是将GPIO_INOUT寄存器的下半部分设置为1）：


```
int En_Value=0xFFFF;  
  
WRITE_GPIO(GPIO_INOUT, En_Value);
```

2. 通过执行以下代码，反复读取开关的值并将该值写入LED。回想一下，开关的值会读入存储器映射I/O寄存器的上半部分，因此该值必须先右移16位再写入与LED物理连接的存储器映射I/O寄存器。

```
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宏分别在指定的存储器映射I/O地址处读取或写入值。

步骤3. 将RVfpgaNexys下载到Nexys A7 FPGA开发板

现在，需要将RVfpgaNexys下载到Nexys A7 FPGA开发板上。单击左侧功能区菜单中的PlatformIO图标，展开“Project Tasks”（项目任务）→“env:swervolf_nexys”→“Platform”（平台），然后单击“Upload Bitstream”（上传比特流），如图10所示。

注：如果使用的是Windows系统并且尚未更换Nexys A7 FPGA开发板驱动程序，请按照“入门指南”附录B中的说明进行操作。

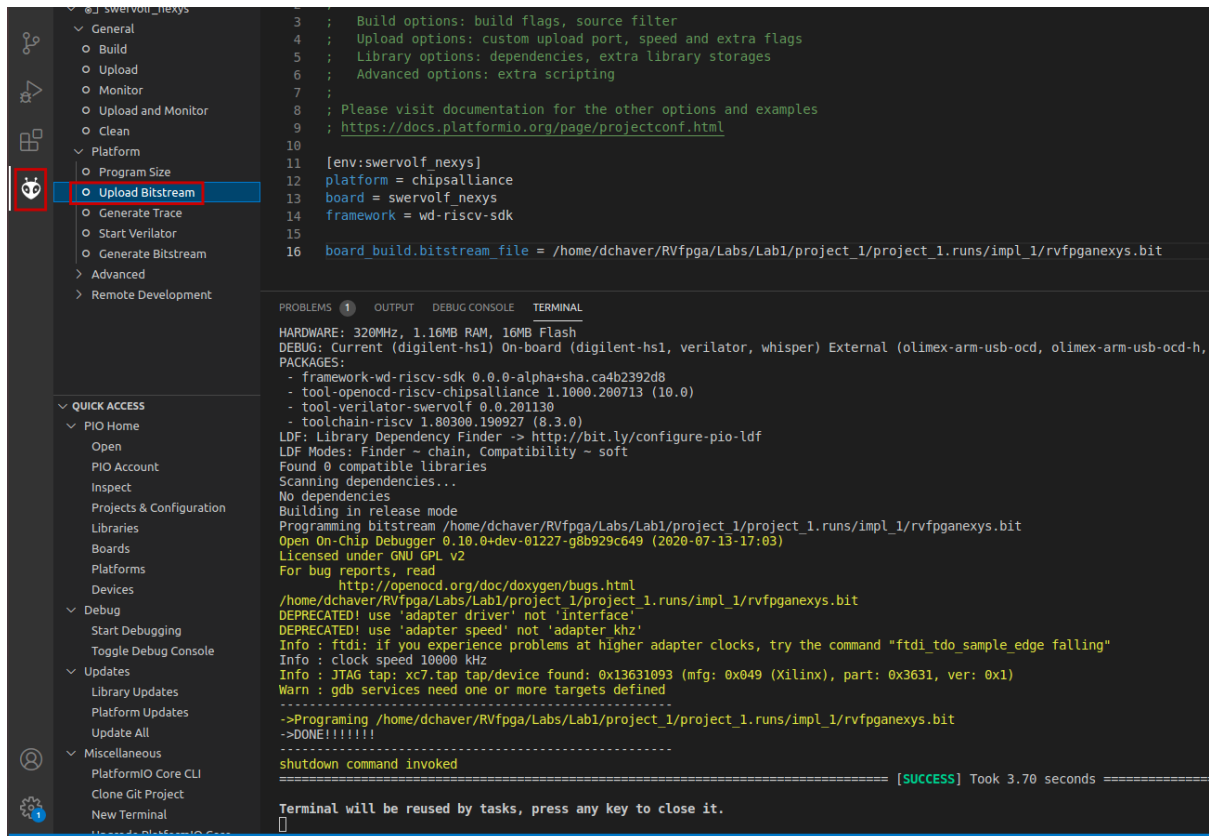



图10. 使用PlatformIO将RVfpgaNexys上传到Nexys A7 FPGA开发板上

也可以通过PlatformIO终端窗口下载RVfpgaNexys，如图11所示。单击PlatformIO窗口底部的“PlatformIO: New Terminal”（PlatformIO：新建终端）按钮（），然后将以下内容输入（或复制粘贴）到PlatformIO终端中：

```
pio run -t program_fpga
```

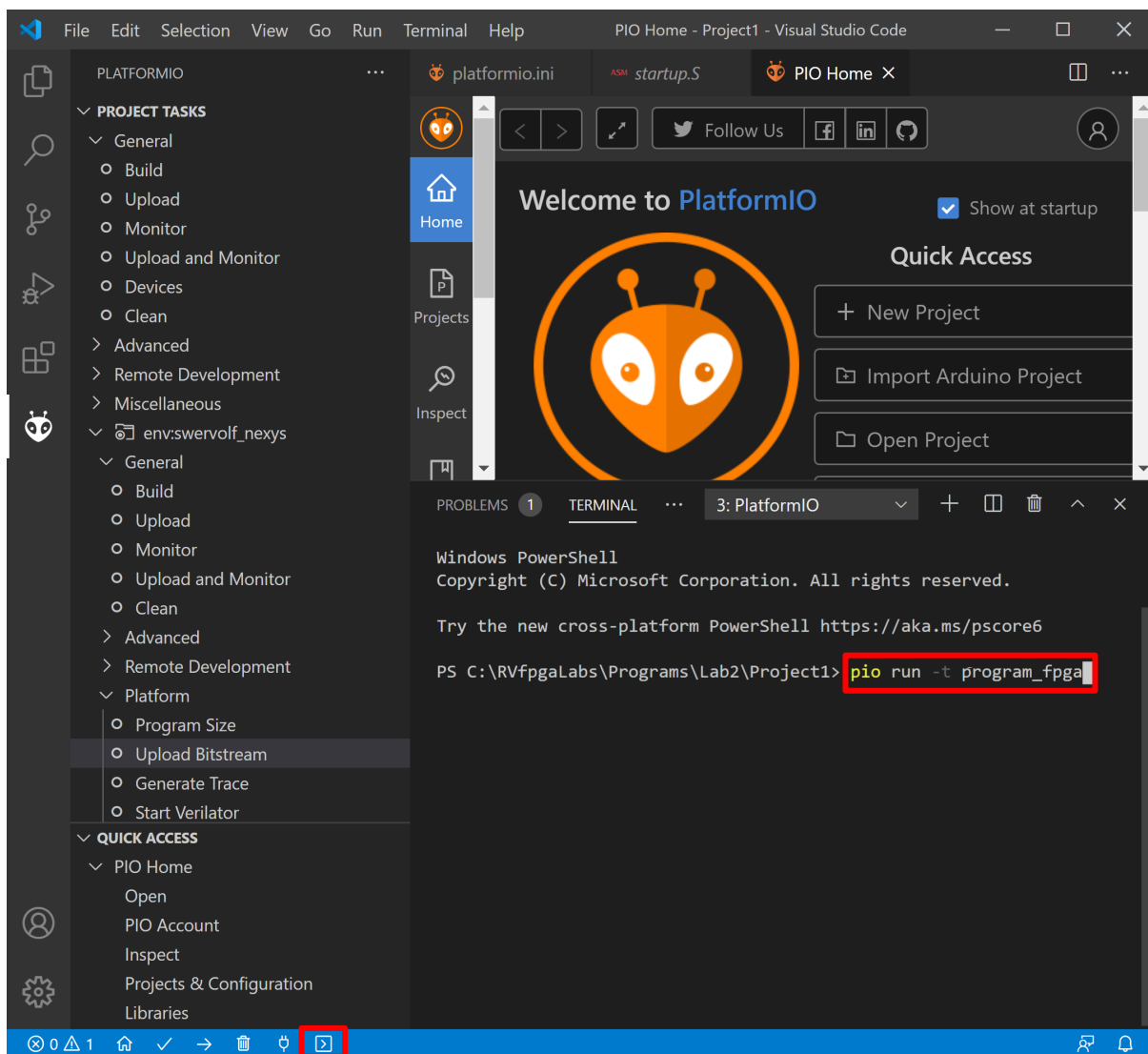


图11. 使用PlatformIO终端将RVfpgaNexys上传到Nexys A7 FPGA开发板上

步骤4. 编译、下载和运行C程序

现在RVfpgaNexys已在开发板上运行，接下来需要编译程序，将程序下载到RVfpgaNexys，然后运行/调试程序。如果VSCode尚未打开，请将其打开。上一个项目Project1应该会自动打开。如果该项目未自动打开，请确保已打开PlatformIO扩展，然后单击“File”（文件）→“Open Folder”（打开文件夹）并选择（但不要打开）在本实验前面部分创建的Project1。

单击左侧功能区菜单中的“Run”（运行）按钮（参见图12）。

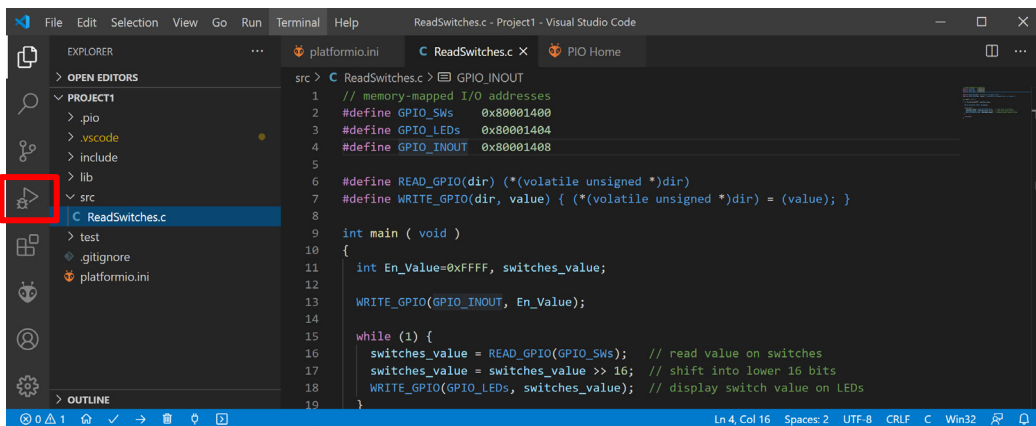


图12. 在RVfpgaNexys上运行程序

现在，单击“Start Debugging”（开始调试）按钮（参见图13）。

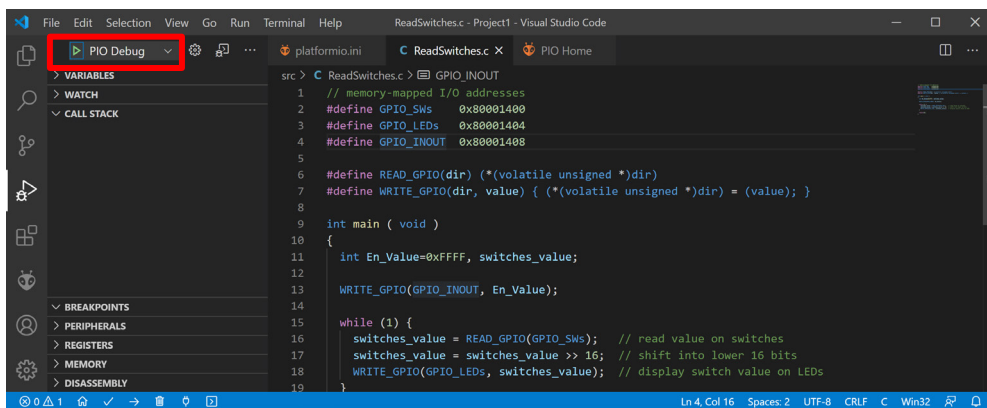


图13. 开始运行和调试程序

程序将先编译，然后下载至Nexys A7开发板的FPGA上所运行的RVfpgaNexys。（请注意，终端中可能会出现sys/cdefs.h文件缺失的声明，但程序仍可正常运行。）此时即可开始运行和调试程序（参见图14）。

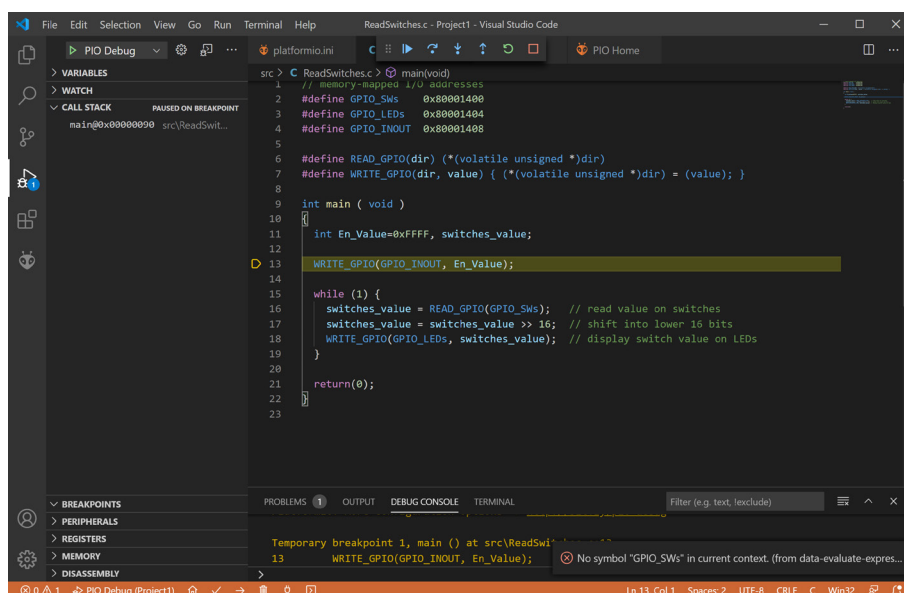


图14. 在RVfpgaNexys上运行的程序

如“RVfpga入门指南”中所述，要控制调试会话，请使用编辑器顶部位置附近的调试工具栏（参见图15）。选项如下：


1. **Continue（继续）**：执行程序，直至达到下一个断点。
2. **Breakpoints（断点）**：可通过在编辑器中单击行号左侧区域进行添加。
3. **Step Over（单步跳过）**：执行当前行，然后停止。
4. **Step Into（单步进入）**：执行当前行，如果当前行包含函数调用，将跳转到该函数并停止。
5. **Step Out（单步跳出）**：执行所在函数中的所有代码，然后在该函数返回结果时停止。
6. **Restart（重启）**：从程序的开头重新启动调试会话。
7. **Stop（停止）**：停止调试会话并返回正常编辑模式。请注意，按下“Stop”（停止）按钮时，程序将继续在RVfpgaNexys上运行，但调试会话将终止。
8. **Pause（暂停）**：暂停执行。程序运行时，“Continue”（继续）按钮将替换为“Pause”（暂停）按钮。



图15. 调试工具

在左侧工具栏中可以查看调试器选项。提供以下选项：

- **Variables（变量）**：列出程序中存在的局部、全局和静态变量及其变量值。
- **Call Stack（调用堆栈）**：显示当前正在运行的函数、调用函数（如果存在）以及当前指令在存储器中的位置。
- **“Breakpoints”（断点）**：显示所有设置的断点并突出显示其行号。可在此部分中管理断点。也可以暂时停用断点，而无需通过切换复选框将其删除。
- **Peripherals（外设）**：显示设备的存储器映射外设的寄存器状态。
- **Registers（寄存器）**：列出存在于处理器的每个寄存器中的当前值。
- **Memory（存储器）**：显示特定存储器地址的内容。
- **Disassembly（反汇编）**：显示特定函数的汇编代码，对于C语言等较高级别的代码，此选项支持查看汇编代码以逐行调试指令。

例如，单击第18行左侧区域，可在将开关的值写入LED之前设置一个断点，如图16所示。然后，单击“Continue”（继续）按钮 （或按下F5）运行程序。程序将继续运行，直至到达设置的断点。（要删除现有断点，只需单击行号左侧的该断点。）

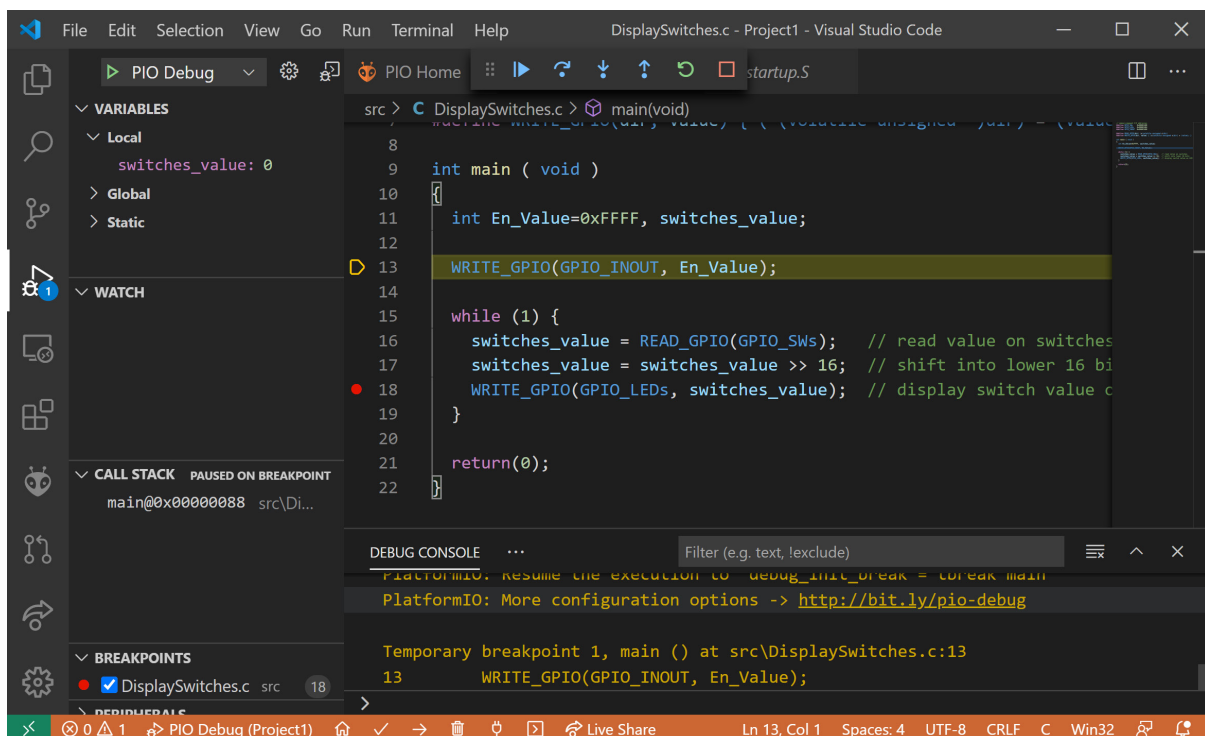


图16. 在LED上显示值之前设置断点

到达断点后，展开左侧窗格中的“Variables”（变量）部分，查看变量switchs_value的值，如图17所示。在这种情况下，开关的值为1029 = 0x405（二进制形式为0000_0100_0000_0101），对应于以下模式：

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

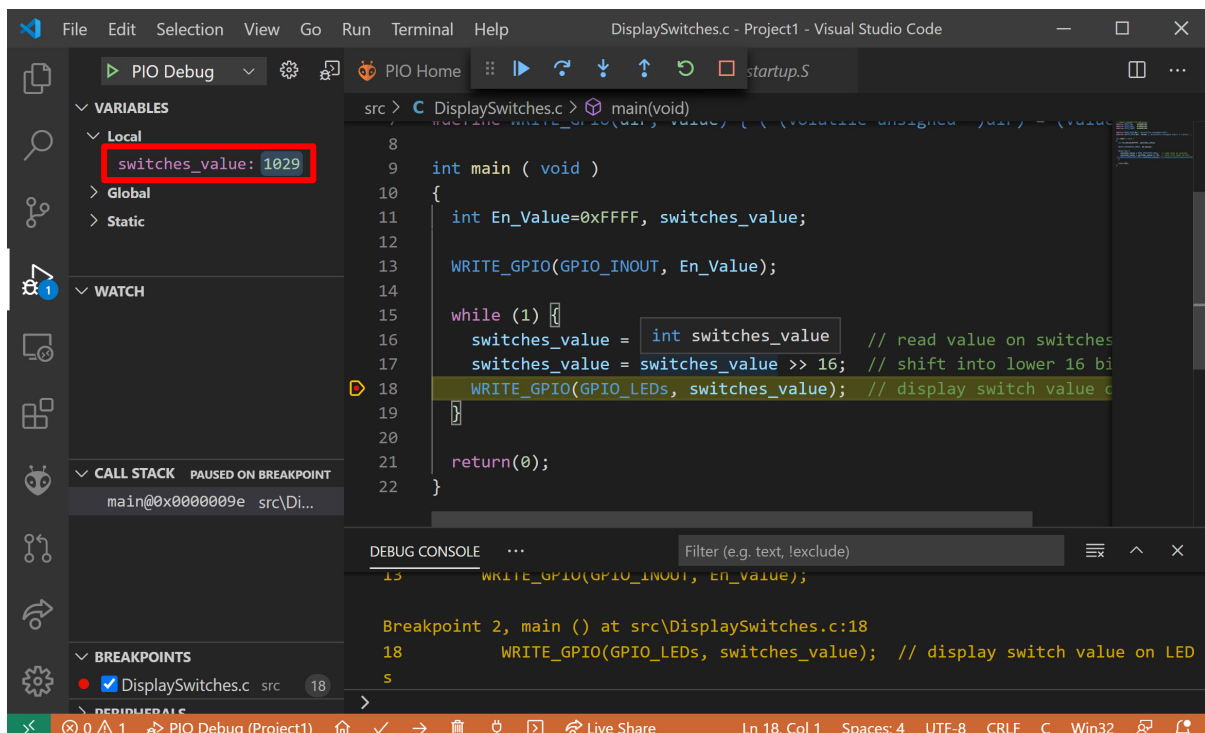


图17. 查看变量值

还可以查看通过C程序生成的RISC-V汇编代码。要执行此操作，请单击“DISASSEMBLY”（反汇编）→“Switch to assembly”（切换至汇编），如图18所示。

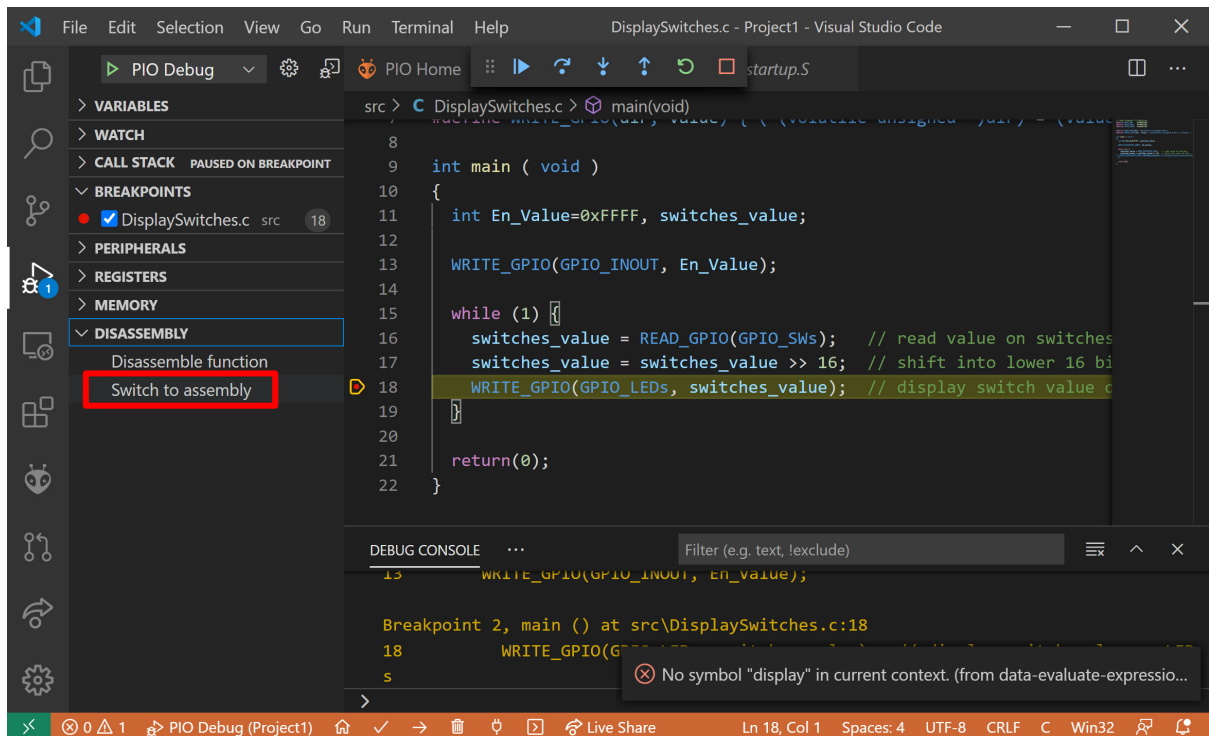


图18. 查看RISC-V汇编代码

现在，RISC-V汇编会显示在查看窗格中，如图19所示。汇编部分显示指令的存储器地址、机器代码和汇编代码。可以看出，C代码被编译为压缩指令（16位指令）和32位指令的组合。汇编代码及相关注释如下。

# 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

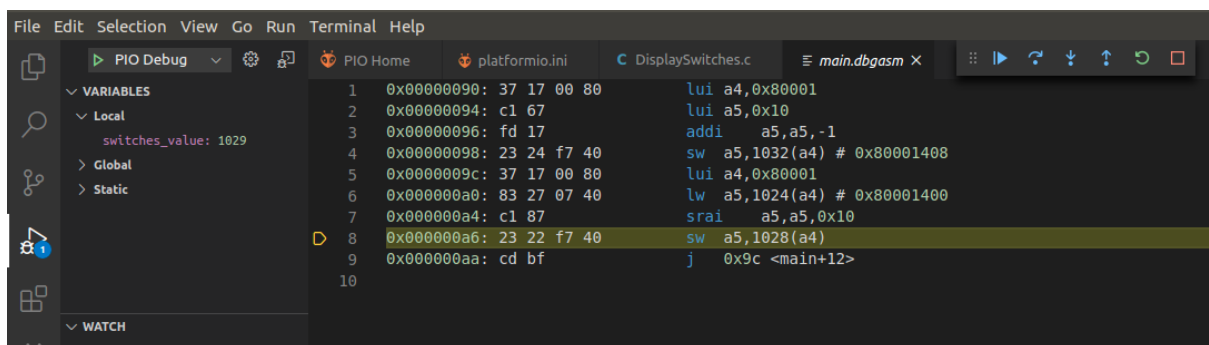




图19. 查看RISC-V汇编代码

单击“DISASSEMBLY”（反汇编）→“Switch to code”（切换至代码）再次查看C代码。

程序运行/调试完成后，按下“Stop”（停止）按钮  （或Shift-F5）

停止调试会话，然后单击最左侧栏顶部的  返回“Explorer”（资源管理器）窗口。请注意，程序将继续在RVfpgaNexys上运行，只有调试会话被终止。单击顶部菜单栏中的“File”（文件）→“Close Folder”（关闭文件夹）关闭项目。

3. 使用printf和串行监视器

在程序中使用print语句是跟踪程序进度或向用户提供反馈（例如计算结果）的一种有效方式。回想一下，在“RVfpga入门指南”（第6.F节的示例*HelloWorld_C-Lang*）中，可以使用printfNexys函数，该函数类似于典型的C程序中的printf函数。为此，必须使用可为给定处理器和开发板提供常用函数的Western Digital PSP和BSP（处理器支持包和开发板支持包），在此种情况下应使用SweRV EH1内核和Nexys A7 FPGA开发板。

在[RVfpgaPath]/RVfpga/Labs/Lab2文件夹中创建一个名为PrintfExample的PlatformIO项目。将以下程序添加到该项目（参见图20）。将程序文件命名为*PrintfExample.c*。

为方便您使用，也可通过以下路径获取程序：

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

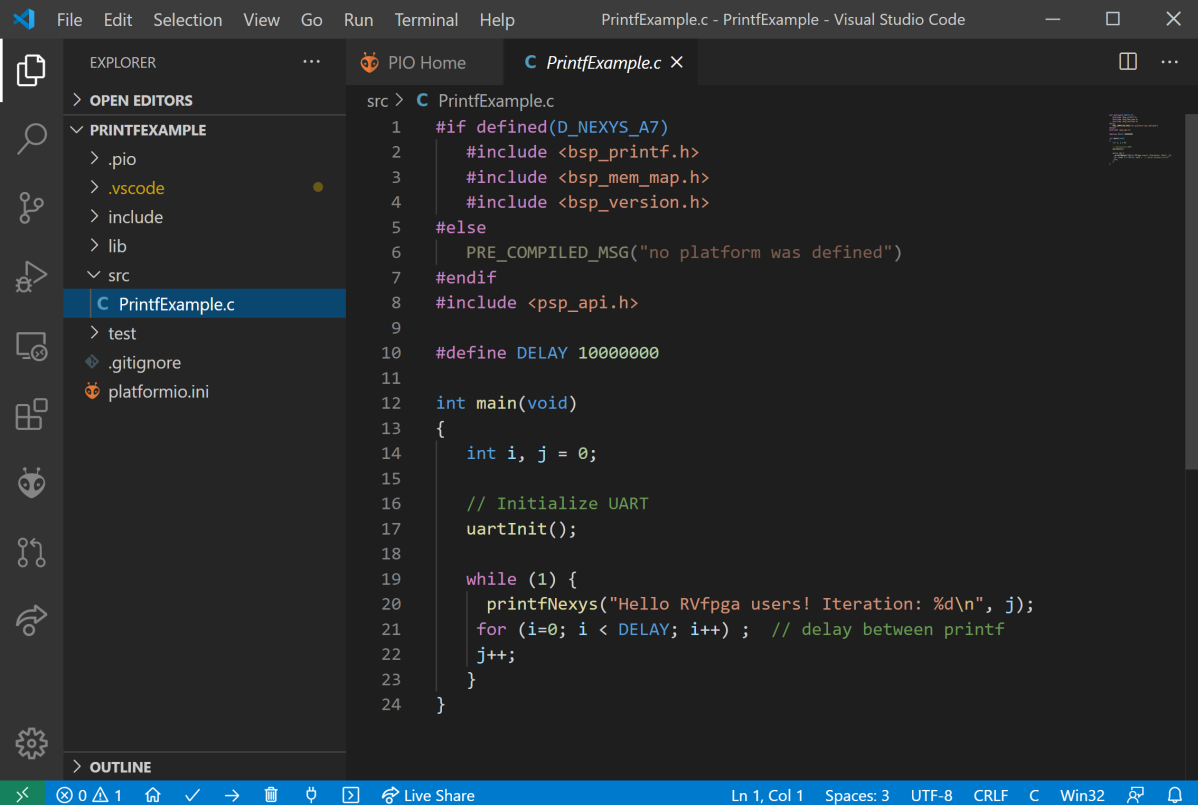
```
#if defined(D_NEXYS_A7)
#include <bsp_printf.h>
#include <bsp_mem_map.h>
#include <bsp_version.h>
#else
    PRE_COMPILED_MSG("no platform was defined")
#endif
#include <psp_api.h>

#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 }

```

图20. PrintfExample.c

第1-8行（参见图20）是使用printfNexys函数时所需的包含文件。这些文件由Western Digital的BSP/PSP提供。图20中的第17行调用了uartInit函数；此代码行用于初始化UART连接，从而实现RVfpgaNexys（在Nexys A7开发板上运行）与串行监视器之间的通信。最后，while循环搭配使用第20行的printfNexys函数与第21行的延时反复向串行监视器写入数据。

要使用printfNexys函数和UART，必须修改platformio.ini文件，将UART的速度包含在内。将以下代码行添加到platformio.ini文件，如图21所示：

```
monitor_speed = 115200
```


RVfpgaNexys期望UART以115200波特（每秒符号数）的速率进行通信，因此必须在platformio.ini中设置此速率，如图21中第16行所示。此外，不要忘记使用board_build.bitstream_file = ... 添加RVfpgaNexys bit文件的位置（如图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
```

图21. 设置UART速度

LINUX: 请记住，如果使用的是Linux，需要先通过将用户添加到dialout、tty和uucp组来完成系统准备工作（如“入门指南”第6.F节中所述），之后才能使用串行监视器。如果已在GSG中执行了此操作，则一切功能都将正常；否则，请立即执行此操作。

然后上传bit文件（如第2节所述）并运行/调试程序。

程序开始运行后（**必须等到**程序开始运行后），单击PlatformIO窗口底部的串行监视器按钮（参见图22）。

警告: 如果在程序开始运行（并到达第一个临时断点）之前打开串行监视器，UART将出现问题，无法正常工作。

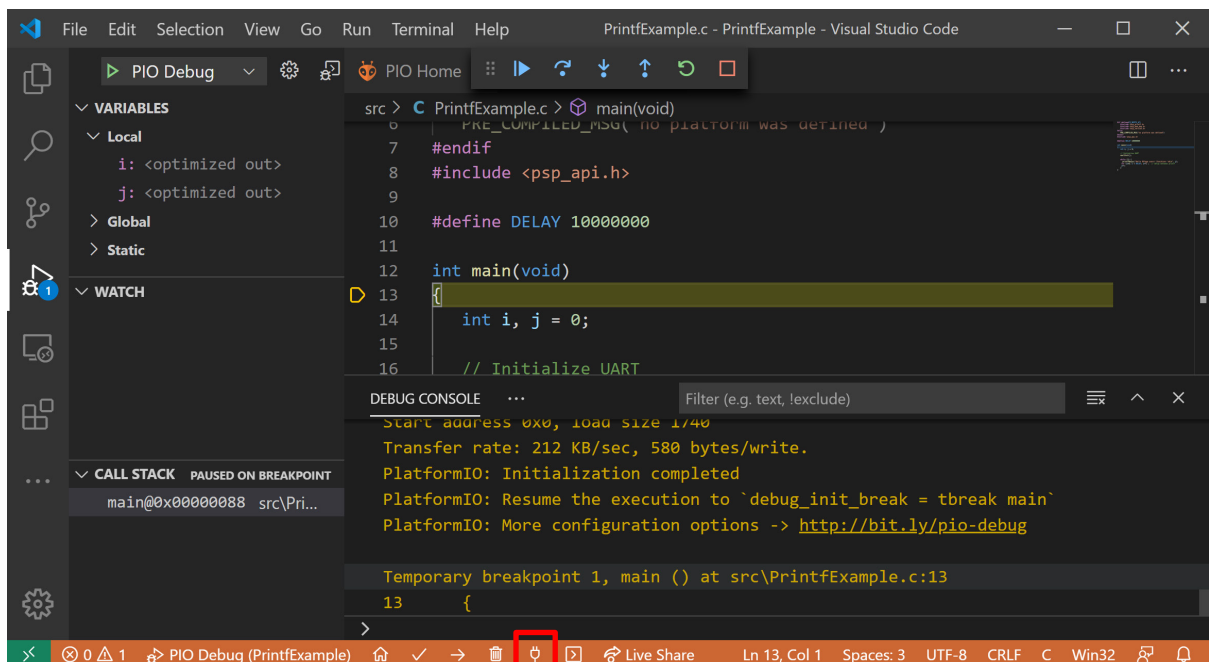


图22. 启动串行监视器

然后运行程序：

您将看到打印字符串（Hello RVfpga users!）和迭代数在串行监视器上反复出现，如图23所示。

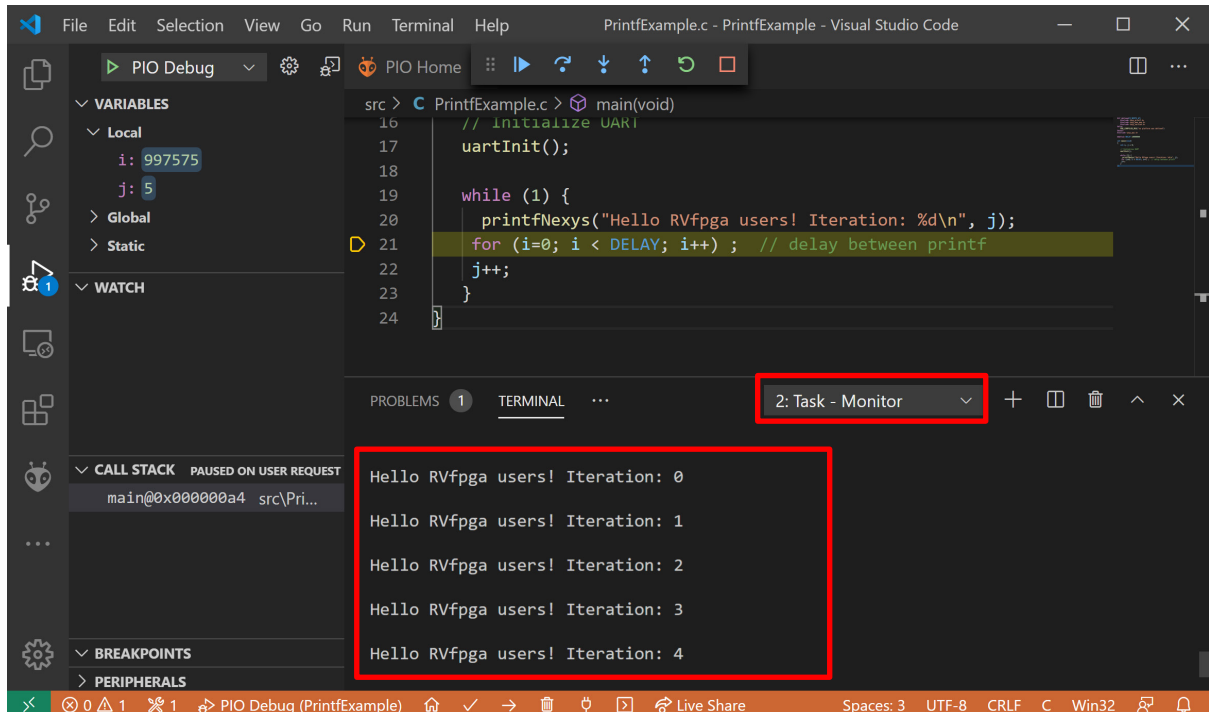


图23. 在PlatformIO中的串行监视器上输出printfNexys函数

4. 练习

通过完成以下练习，创建自己的C程序。请注意，如果使Nexys A7开发板与计算机保持连接并保持上电状态，则在切换运行不同程序时，无需将RVfpgaNexys重新加载到开发板上。但是，如果关闭Nexys A7开发板，则需要使用PlatformIO将RVfpgaNexys重新加载到开发板上，如第2节的第3步所述。

请记住，可以使用Western Digital的BSP函数printfNexys打印任何变量（参见第3节）。

还要记住，可以使用Verilator和Whisper在仿真中运行这些程序。

练习1. 编写一个C程序，使开关的值在LED上闪烁。该值应以足够慢的速度进行脉冲开关，让人眼可以观察到值在闪烁。将程序命名为FlashSwitchesToLEDs.c。

练习2. 编写一个C程序，在LED上显示开关值取反后的值。例如，如果开关的值（二进制）为：01010101010101，则LED应显示：1010101010101010；如果开关的值为：1111000011110000，则LED应显示：0000111100001111；依此类推。将程序命名为DisplayInverse.c。

练习3. 编写一个C程序，不断增加点亮并滚动的LED的数量，直到所有LED都点亮为止。然后该模式应重复进行。将程序命名为**ScrollLEDs.c**。

该程序应产生以下效果：

1. 首先，有一个点亮的LED从右向左滚动，然后又从左向右滚动。
2. 然后，有两个点亮的LED从右向左滚动，然后又从左向右滚动。
3. 然后，有三个点亮的LED从右向左滚动，然后又从左向右滚动。
4. 依此类推，直到所有LED均点亮。
5. 然后该模式应重复进行。

练习4. 编写一个C程序，显示开关的4个最低有效位和开关的4个最高有效位相加所得的4位无符号值。在LED的4个最低有效（最右边）位上显示结果。将程序命名为**4bitAdd.c**。当发生无符号溢出（即进位为1）时，LED的第五位应亮起。

练习5. 编写一个C程序，根据欧几里得算法求出*a*和*b*两个数的最大公约数。*a*和*b*两个值在程序中应该是静态定义的变量。将程序命名为**GCD.c**。有关欧几里得算法的更多信息，请访问：<https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm>。也可以直接用google搜索“欧几里德算法”。

练习6. 编写一个C程序，计算斐波那契数列中的前12个数，并将结果存储在长度为12的有限向量（即数组）**V**中。这个无限的斐波纳契数列定义如下：

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

换言之，与元素*i*对应的斐波那契数是数列中在其之前的两个斐波那契数之和。表1显示了*i*为0到8时的斐波那契数。

表1. 斐波那契数列

<i>i</i>	0	1	2	3	4	5	6	7	8
V	0	1	1	2	3	5	8	13	21

向量的维数**N**在程序中必须定义为常量。将程序命名为**Fibonacci.c**。

练习7. 现有一个**N**维向量（即数组）**A**，请生成另一个向量**B**，使向量**B**只包含**A**中大于0的偶数元素。**C**程序还必须计算**B**中的元素数量，并在程序末尾打印该值。例如：假设**N = 12**，**A = [0,1,2,7,-8,4,5,12,11,-2,6,3]**，则**B**为：**B = [2,4,12,6]**。由于**B**中含有四个元素，因此应在程序末尾打印以下内容：

B中的元素数 = 4。

为此，应使用printfNexys函数。将程序命名为**EvenPositiveNumbers.c**。在**A**中有12个元素时测试程序。

练习8. 现有两个 N 维向量（即数组） A 和 B ，请生成另一个向量 C ，定义如下：

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

用C语言编写一个计算新向量的程序。在程序中使用包含12个元素的数组。将程序命名为**AddVectors.c**。

练习9. 用C语言实现冒泡排序算法，让算法通过以下方式按升序对向量分量进行排序：

1. 反复遍历向量直至完成。
2. 如果两个相邻分量 $v(i) > v(i+1)$ ，则互换其位置。
3. 当每对相邻分量的顺序均正确时，算法停止。

使用包含12个元素的数组测试程序。将程序命名为**BubbleSort.c**。

练习10. 用C语言编写一个程序，通过迭代乘法计算给定非负数 n 的阶乘。虽然应使用多个 n 值测试程序，但最终提交的应为 $n = 7$ 时的结果。该程序应在程序末尾打印阶乘（ n ）的值。 n 应为程序内静态定义的变量。将程序命名为**Factorial.c**。