THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga Lab 1
C Programming

1. INTRODUCTION

Most computer programs are written in a high-level language such as C. This lab shows you how to create a C project in PlatformIO that you can run on the RVfpga System. We first provide a tutorial on how to create and run a C program. Then we describe exercises for you to practice writing your own C programs.

> **IMPORTANT:** Before starting RVfpga Labs, you must have already completed the RVfpga Getting Started Guide provided by the Imagination University Programme (https://university.imgtec.com/).
>
> For example, if you have not already, install the software tools (at least VSCode and PlatformIO) following the instructions in the RVfpga Getting Started Guide. Also, make sure that you have copied the *RVfpga* folder that you downloaded from Imagination's University Programme to your machine. We will refer to the absolute path of the directory where you place folder RVfpga as [*RVfpgaPath*]. The [*RVfpgaPath*]/RVfpga/src folder contains the Verilog and SystemVerilog sources for the RVfpga System, the RISC-V SoC that we will use and modify throughout the labs. The [*RVfpgaPath*]/RVfpga/Labs folder contains some programs that you will use during Labs 1 to 20.

2. C Program for RVfpga

You will complete the following steps to create and run a C program on RVfpgaNexys using PlatformIO (remember that you can also run these programs on simulation, using Verilator and Whisper):
1. Create an RVfpga project
2. Write a C program
3. Download RVfpgaNexys onto Nexys A7 FPGA board
4. Compile, download, and run a C program

Step 1. Create an RVfpga project
Open VSCode (as described in the RVfpga Getting Started Guide). If PlatformIO does not automatically open when you start VSCode, click on the PlatformIO icon in the left menu ribbon and then click on PIO Home → Open (see Figure 1).
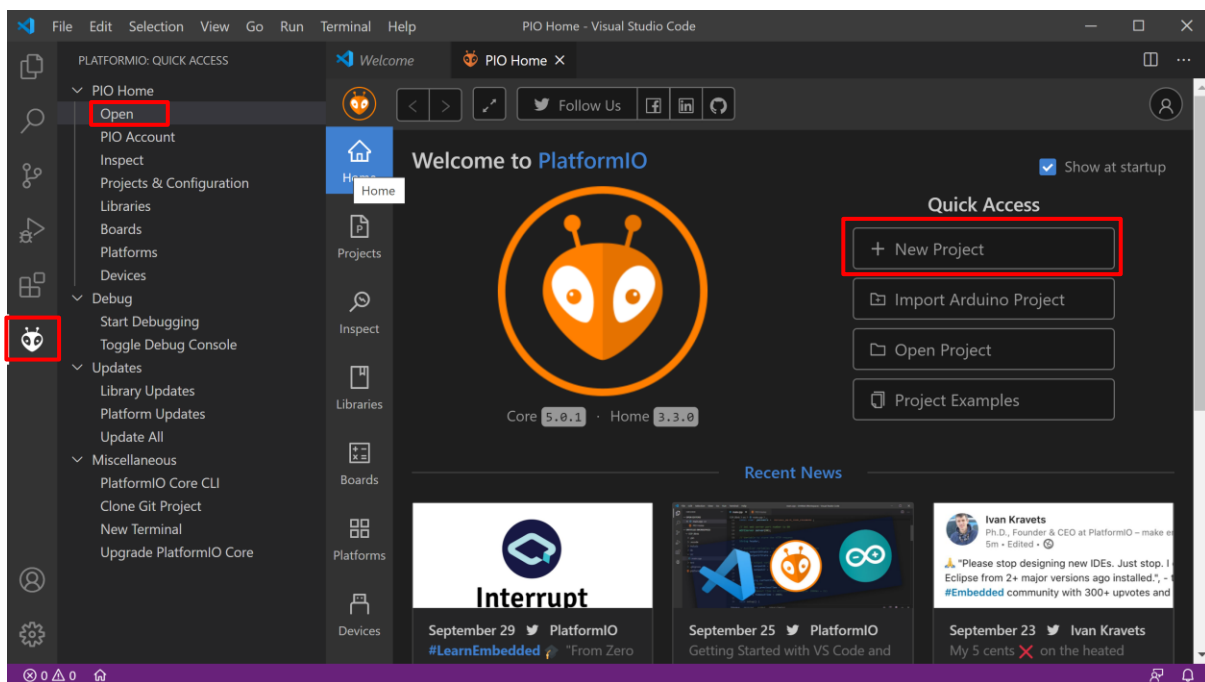
Figure 1. Open PlatformIO and create new project

Now in the PIO Home welcome window, click on New Project (see Figure 1).

As shown in Figure 2, name the project Project1 and choose the Board as RVfpga: Digilent Nexys A7 (start typing in RVfpga and the board will appear as an option). Leave the default framework as WD-firmware. WD-firmware is Western Digital's firmware, which includes the Freedom-E SDK gcc and gdb as well as the PSP and BSP (processor support package and board support package) that we use in these labs. Unclick the Use default location and place your project in:
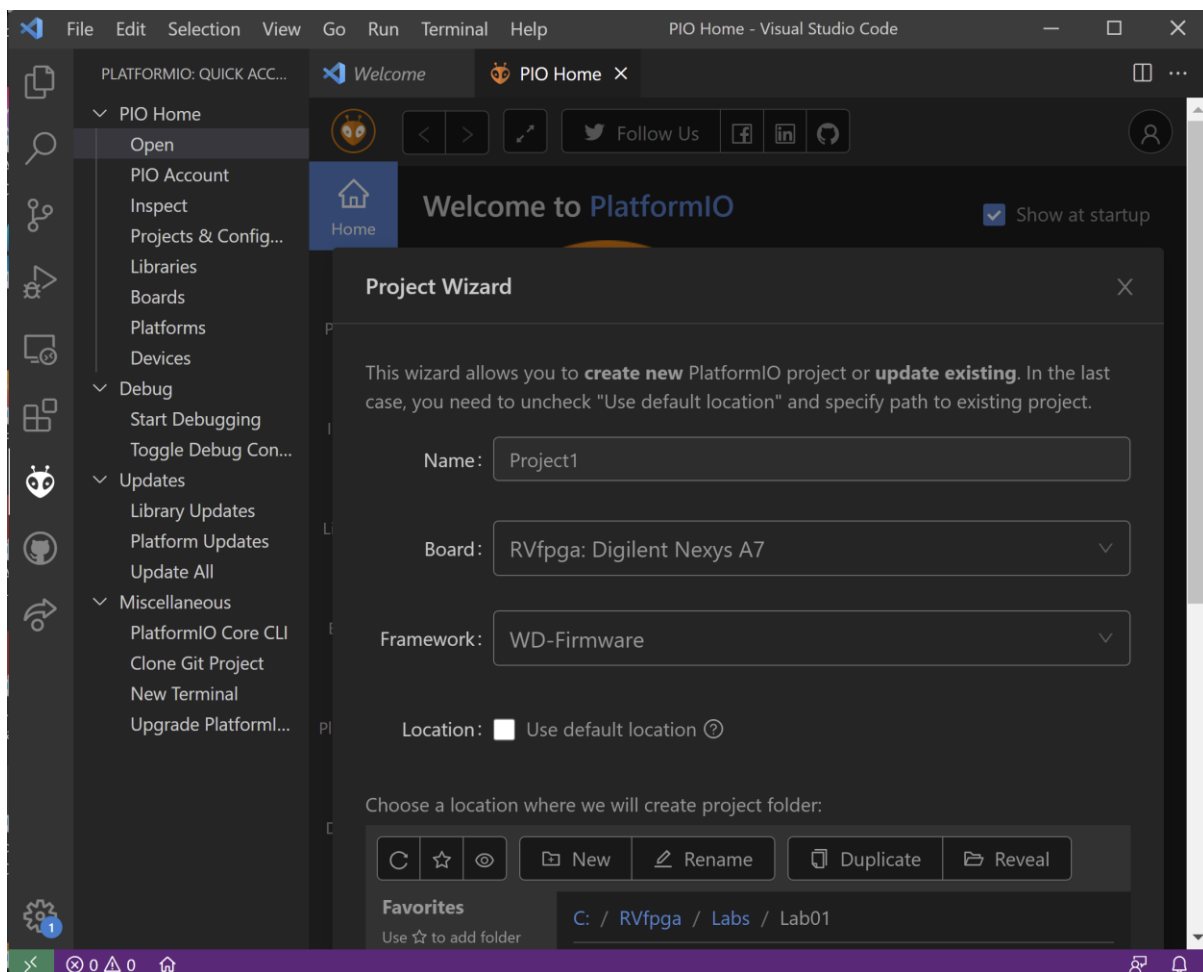
```
[RVfpgaPath]/RVfpga/Labs/Lab01
```

Figure 2. Name project and select board and project folder

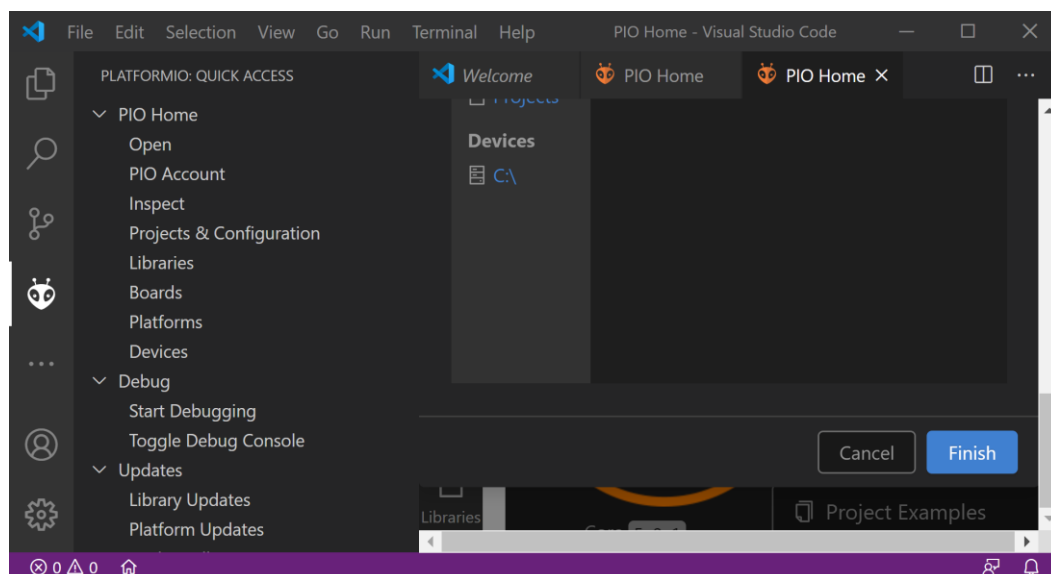Then click Finish at the bottom of the window (see Figure 3).

Figure 3. Finish creating project

In the Explorer pane on the left (which you may need to expand), double-click on platformio.ini to open it (see Figure 4). This is the PlatformIO initialization file.
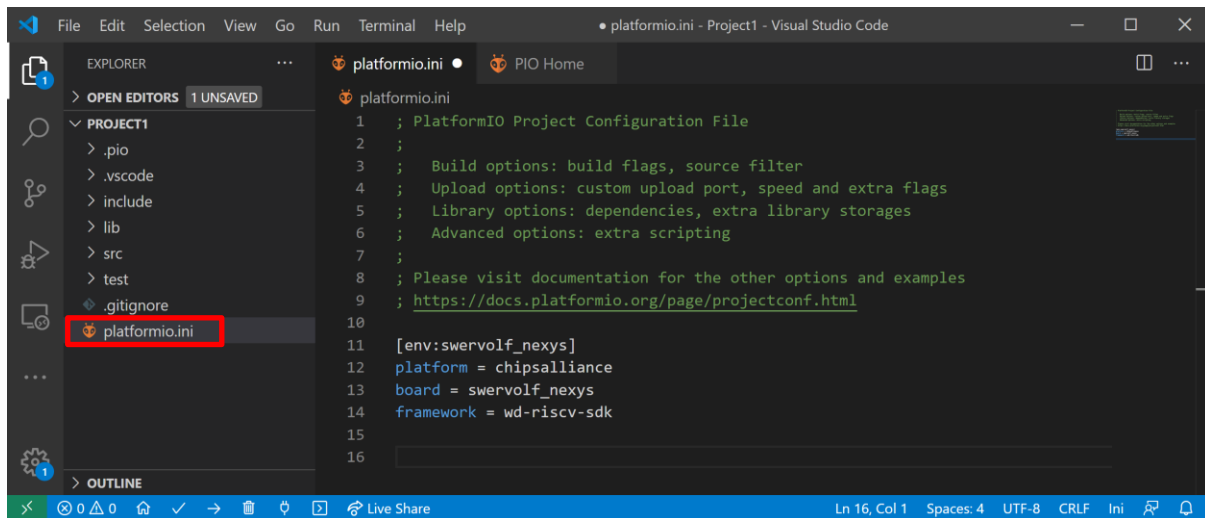
Figure 4. PlatformIO initialization file: platformio.ini

Add the following line to the platformio.ini file, as shown in Figure 5:

```
board_build.bitstream_file = [RVfpgaPath]/RVfpga/src/rvfpganexys.bit
```

(Remember to replace *[RVfpgaPath]* with the location of this folder on your machine.) This line indicates where PlatformIO should find the bitstream file to load onto the FPGA. You will use the RVfpgaNexys bitstream distributed with the Getting Started Guide at: *[RVfpgaPath]/RVfpga/src/rvfpganexys.bit.* Press Ctrl-s to save the platformio.ini file.
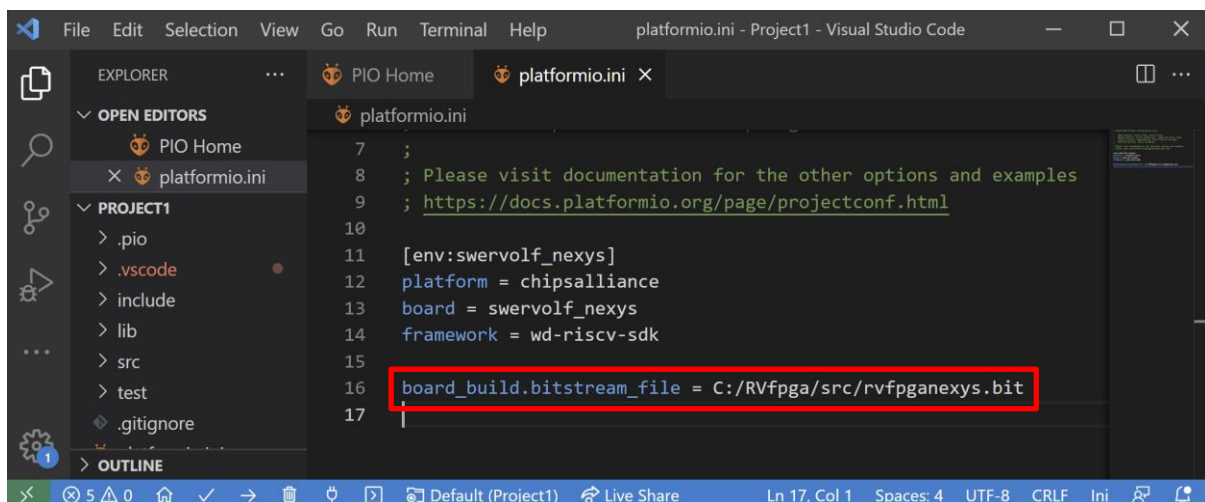

Figure 5. Add location of RVfpgaNexys bitstream file (rvfpganexys.bit)

Remember that a more complete *platformio.ini* file was used in the examples used in the Getting Started Guide. If you want to use any functionality that requires extra commands (such as the path to the Verilator simulator, the configuration of the serial console, the whisper debug tool, etc.), you can use the *platformio.ini* from those examples.

Step 2. Write a C program
Now you will write a C program. Click on File → New File (see Figure 6)
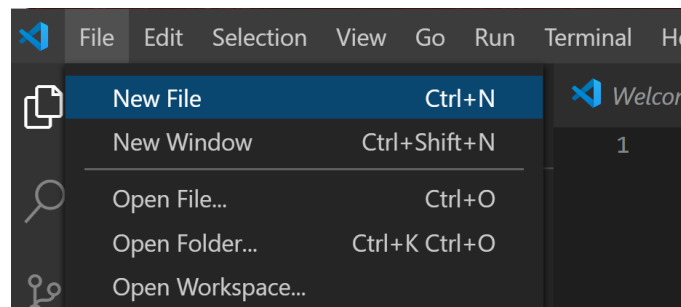
Figure 6. Add file to project

A blank window will open. Type (or copy/paste) the following C program into that window (see Figure 7). This program displays the value of the switches on the LEDs.

```c
// 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);
}
```

This program is also available in the following file for your convenience:

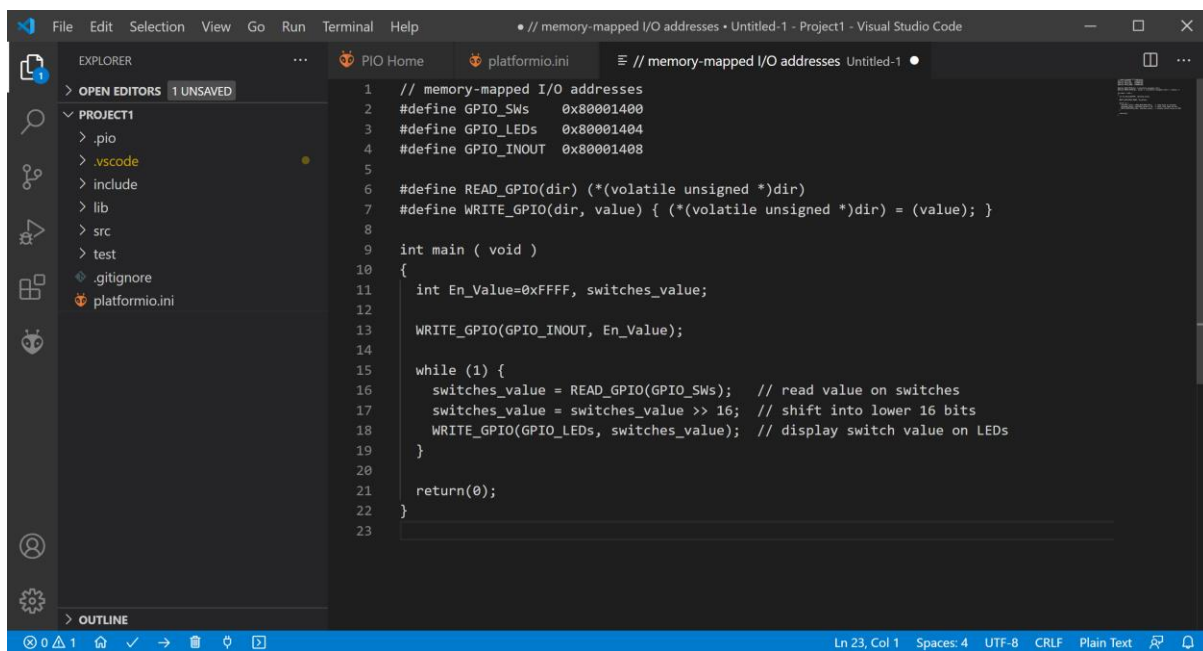*[RVfpgaPath]/RVfpga/Labs/Lab01/DisplaySwitches.c*

Figure 7. Enter C program

After entering the program into the pane, press Ctrl-s to save the file. Name it DisplaySwitches.c and save it in the `src` folder of the `Project1` directory (see Figure 8).
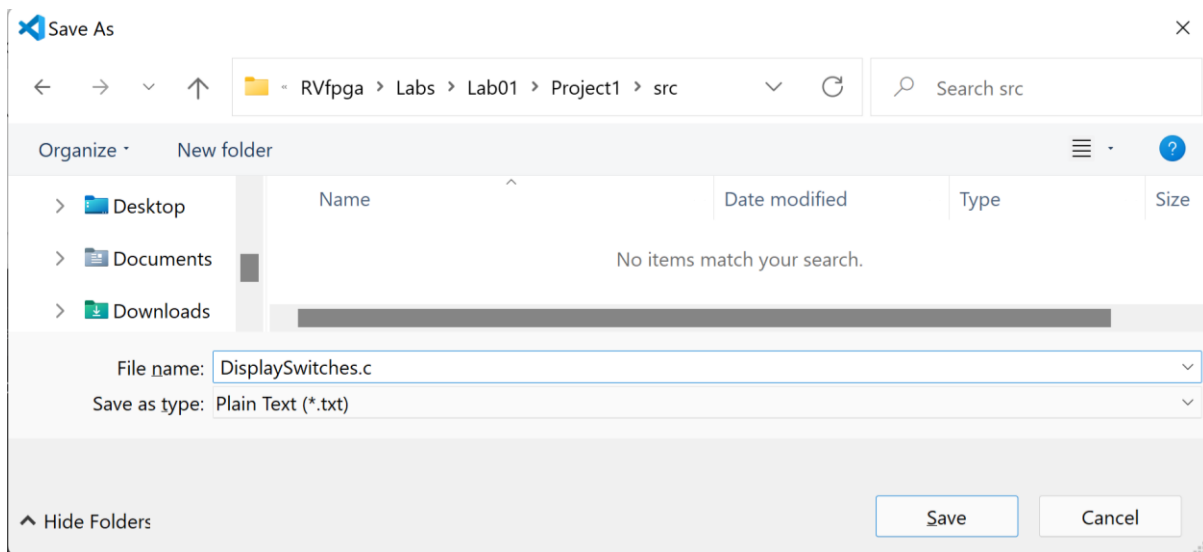

Figure 8. Save file as DisplaySwitches.c

This program first defines the addresses of the memory-mapped I/O registers connected to the LEDs and switches on the Nexys A7 FPGA board using the following lines:

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

The value of the switches is found by reading the register mapped to address 0x80001400 and values are displayed on the LEDs by writing the register mapped to address

0x80001404. The switch values are in the upper half of the register, and the LEDs in the lower half.

The GPIO_INOUT register defines whether a bit of the general-purpose I/O (GPIO) is an input or an output. The least significant 16 GPIO pins, 15:0, are connected to the 16 LEDs on the Nexys A7 board. The most significant 16 GPIO pins, 31:16, are for the 16 on-board switches. A 0 indicates an input and a 1 indicates an output. So, the GPIO_INOUT register is written with 0xFFFF so that the switches are inputs to RVfpgaNexys and the LEDs are outputs driven by RVfpgaNexys.

Figure 9 shows the physical locations of the LEDs and switches on the Nexys A7 FPGA board as well as the USB connector, ON switch, pushbuttons, and 7-segment displays.

Note that in Lab 6 we describe the GPIO features and RVfpgaNexys GPIO hardware in detail. We also discuss how to use the other board peripherals, such as pushbuttons and 7-segment displays, in later labs (Labs 6-10).
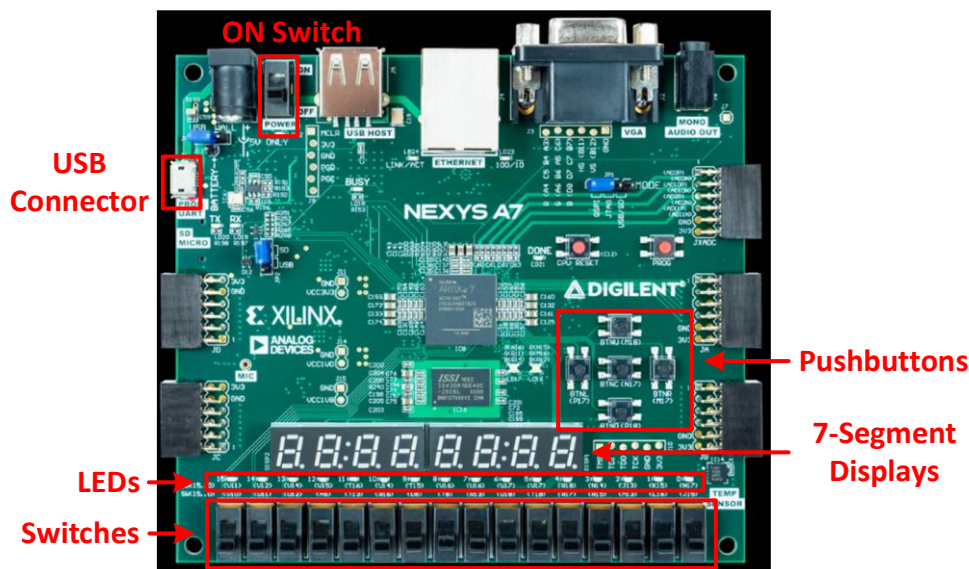

Figure 9. Digilent's Nexys A7 FPGA board's I/O interfaces
(figure of board from https://reference.digilentinc.com/)

After defining the memory-mapped I/O addresses of the LEDs and switches, the program does the following:

1. Defines the most significant 16 GPIO pins (which are connected to the switches) as inputs by setting the upper half of the GPIO_INOUT register to 0's and defines the least significant 16 GPIO pins (which are connected to the LEDs) as outputs by setting the lower half of the GPIO_INOUT register to 1's by executing the following code:

```
int En_Value=0xFFFF;

WRITE_GPIO(GPIO_INOUT, En_Value);
```

2. Repeatedly reads the value of the switches and writes that value to the LEDs by executing the code below. Recall that the value of the switches is read into the upper half of the

memory-mapped I/O register, so the value must be shifted to the right by 16 bits before writing it to the memory-mapped I/O register physically connected to the LEDs.

```
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
}
```

The READ_GPIO and WRITE_GPIO macros respectively read or write a value at the specified memory-mapped I/O address.

Step 3. Download RVfpgaNexys onto Nexys A7 FPGA board
You will now download RVfpgaNexys onto the Nexys A7 FPGA board. Click on the PlatformIO icon in the left menu ribbon, then expand Project Tasks → env:swervolf_nexys → Platform and click on Upload Bitstream, as shown in Figure 10.

Note: if you are using a **Windows** system and did not already replace the Nexys A7 FPGA board driver, do so by following the instructions in Appendix B of the Getting Started Guide.
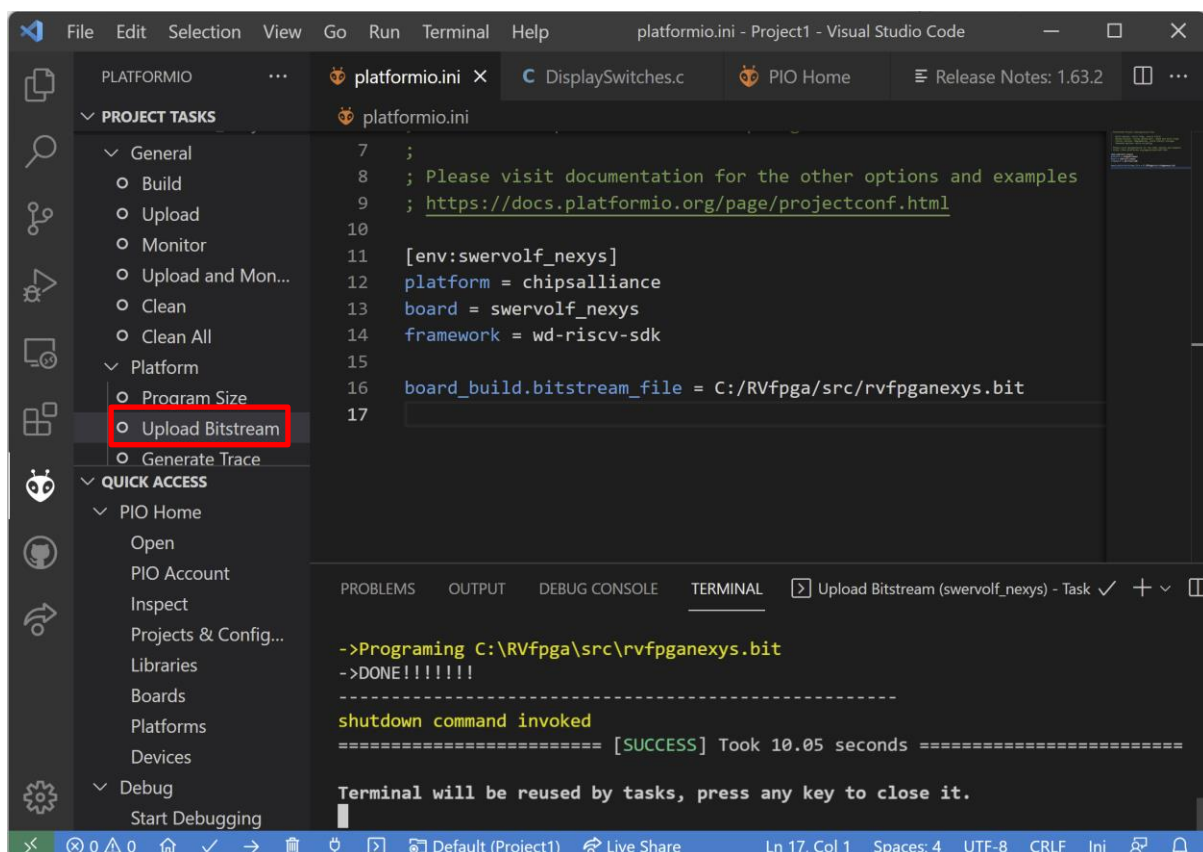

Figure 10. Upload RVfpgaNexys onto Nexys A7 FPGA Board using PlatformIO

As an alternative you can download RVfpgaNexys from a PlatformIO terminal window as shown in Figure 11. Click on the PlatformIO: New Terminal button (⬛) at the bottom of the PlatformIO window, and then type (or copy) the following into the PlatformIO terminal:
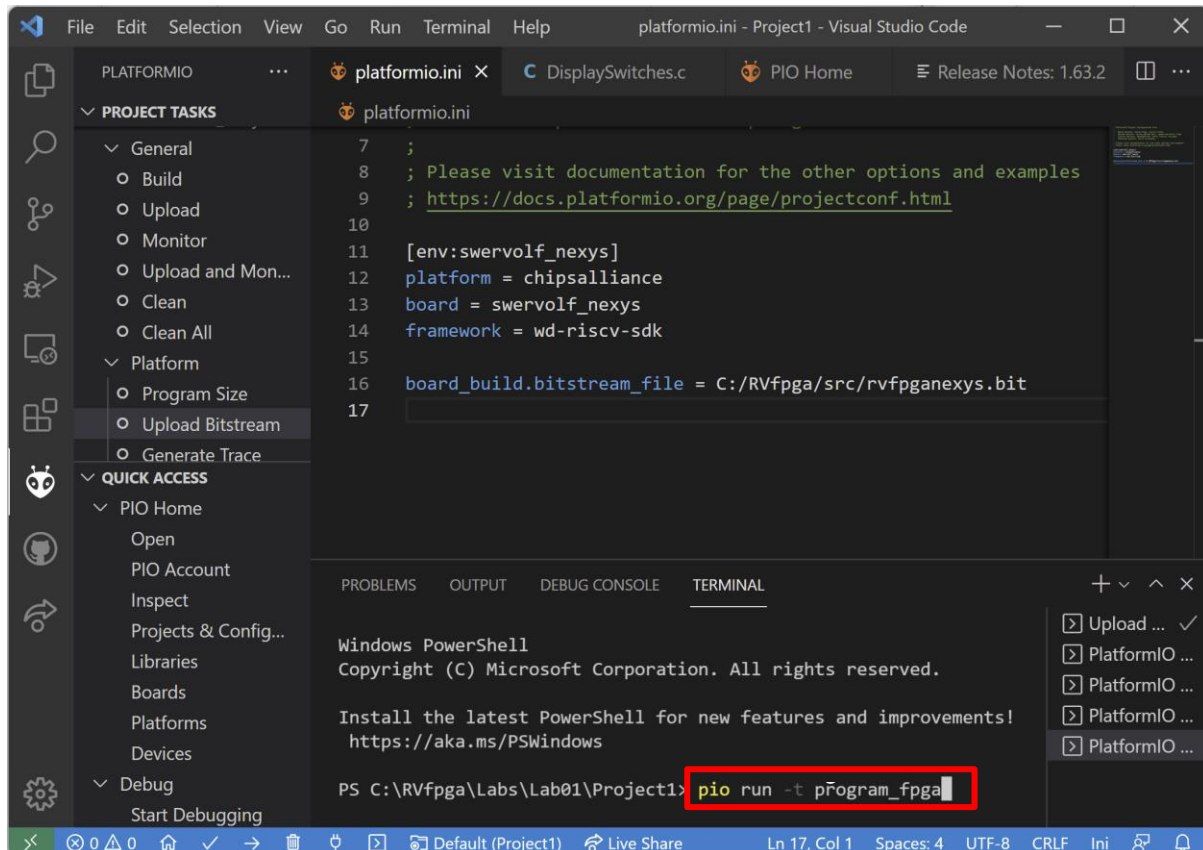
```
pio run -t program_fpga
```

Figure 11. Upload RVfpgaNexys onto Nexys A7 FPGA Board using PlatformIO terminal

Step 4. Compile, download, and run C program
Now that RVfpgaNexys is running on the board, you will compile your program, download it onto RVfpgaNexys, and run/debug it. If VSCode is not already open, open it. Your last project, Project1, should automatically open. If not, make sure the PlatformIO extension is open and click on File → Open Folder and select (but don't open) Project1, that you created earlier in this lab.

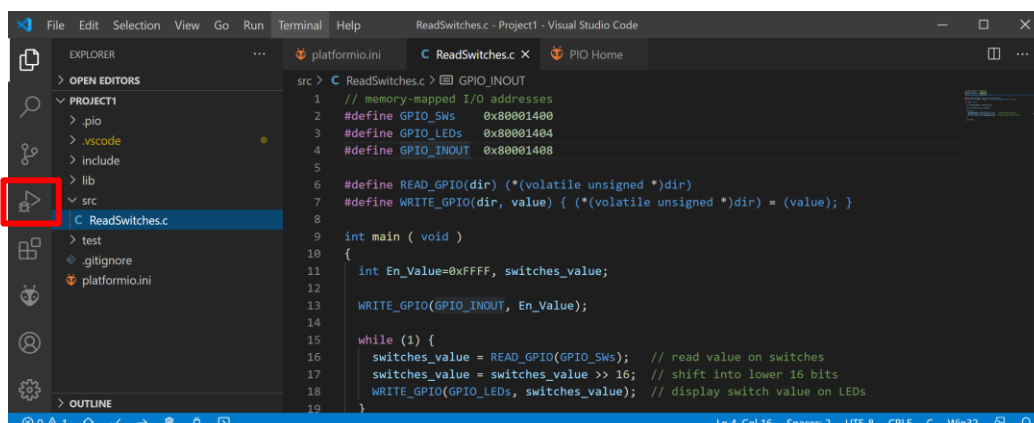Click on the Run button in the left menu ribbon (see Figure 12).

Figure 12. Run program on RVfpgaNexys

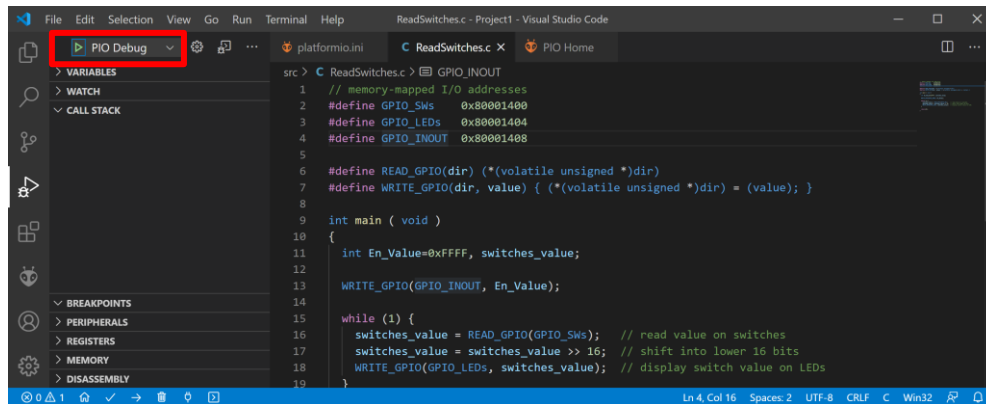Now click on the Start Debugging button (see Figure 13).

Figure 13. Start running and debugging program

The program will compile and then download onto RVfpgaNexys, which is running on the FPGA on the Nexys A7 board. (Note that a statement about a missing sys/cdefs.h file may show up in the terminal – but the program still functions correctly.) Now you can begin running and debugging the program (see Figure 14).
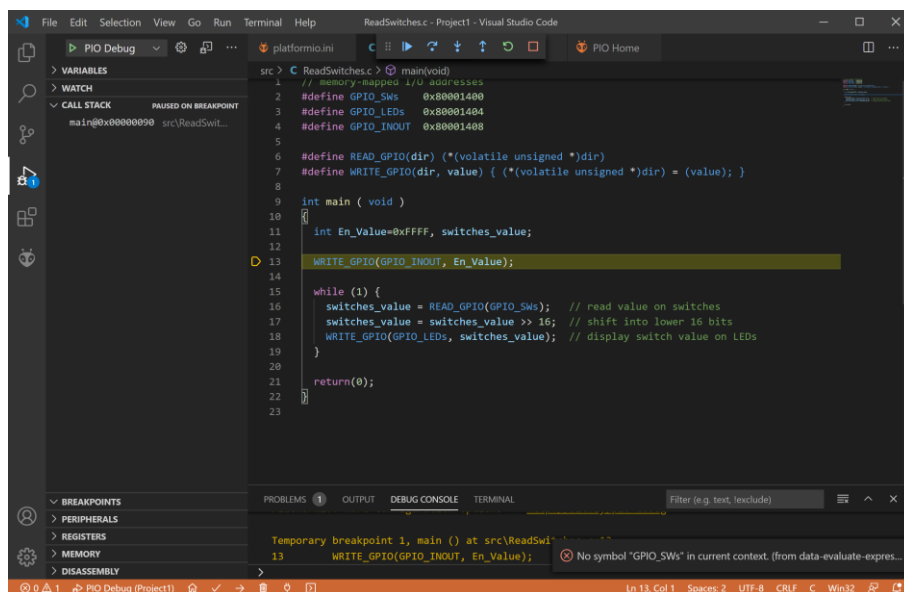

Figure 14. Program running on RVfpgaNexys

As described in the RVfpga Getting Started Guide, to control your debugging session, use the debugging toolbar that appears near the top of the editor (see Figure 15). Below are the options:

1. **Continue** executes the program until the next breakpoint.
2. **Breakpoints** can be added by clicking to the left of the line number in the editor.
3. **Step Over** executes the current line and then stops.
4. **Step Into** executes the current line and if the current line includes a function call, it will jump into that function and stop.
5. **Step Out** executes all of the code in the function you are in and then stops once that function returns.
6. **Restart** restarts the debugging session from the beginning of the program.

7. **Stop** stops the debugging session and returns to normal editing mode. Note that when you press the Stop button, the **program continues running** on RVfpgaNexys, but the debugging session ends.
8. **Pause** pauses execution. When the program is running, the Continue button is replaced by the Pause button.

| CONTINUE | STEP-OVER | STEP-INTO | STEP-OUT | RESTART | STOP |

Figure 15. Debugging tools

On the left sidebar, you can view the Debugger options. The following options are available:
- **Variables:** lists local, global, and static variables present in your program along with their values.
- **Call Stack:** shows you the current function being run, the calling function (if any), and the location of the current instruction in memory.
- **Breakpoints:** show any set breakpoints and highlight their line number. Breakpoints can be managed in this section. Breakpoints can also be temporarily deactivated without removing them by toggling the checkbox.
- **Peripherals**: shows the status of the registers of the memory-mapped peripherals of the device.
- **Registers:** lists the current values present in each of the registers of the processor.
- **Memory:** displays the contents of a specific address of memory.
- **Disassembly:** shows the assembly code for a specific function – for higher-level code such as C, this allows you to view the assembly for debugging the instructions one-by-one.

For example, click to the left of line 18 to set a breakpoint just before the value of the switches is written to the LEDs, as shown in Figure 16. Now run the program by clicking the

Continue button (or pressing F5). The program will continue until it reaches the breakpoint set. (To remove a breakpoint, click on the existing breakpoint, just to the left of the line number.)
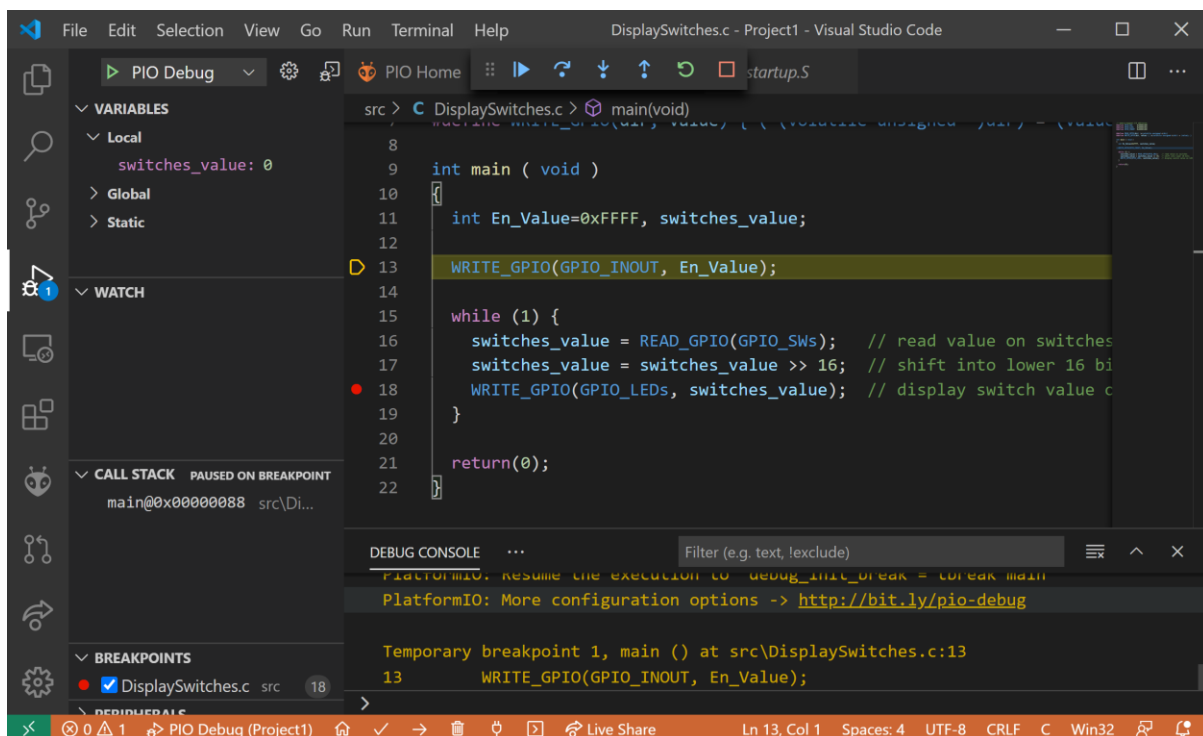
Figure 16. Setting a breakpoint before displaying value on LEDs

After reaching the breakpoint, expand the Variables section in the left pane and view the value of the variable switches_value, as shown in Figure 17. In this case, the value of the switches is 1029 = 0x405 (in binary, 0000_0100_0000_0101), which corresponds to the following pattern:

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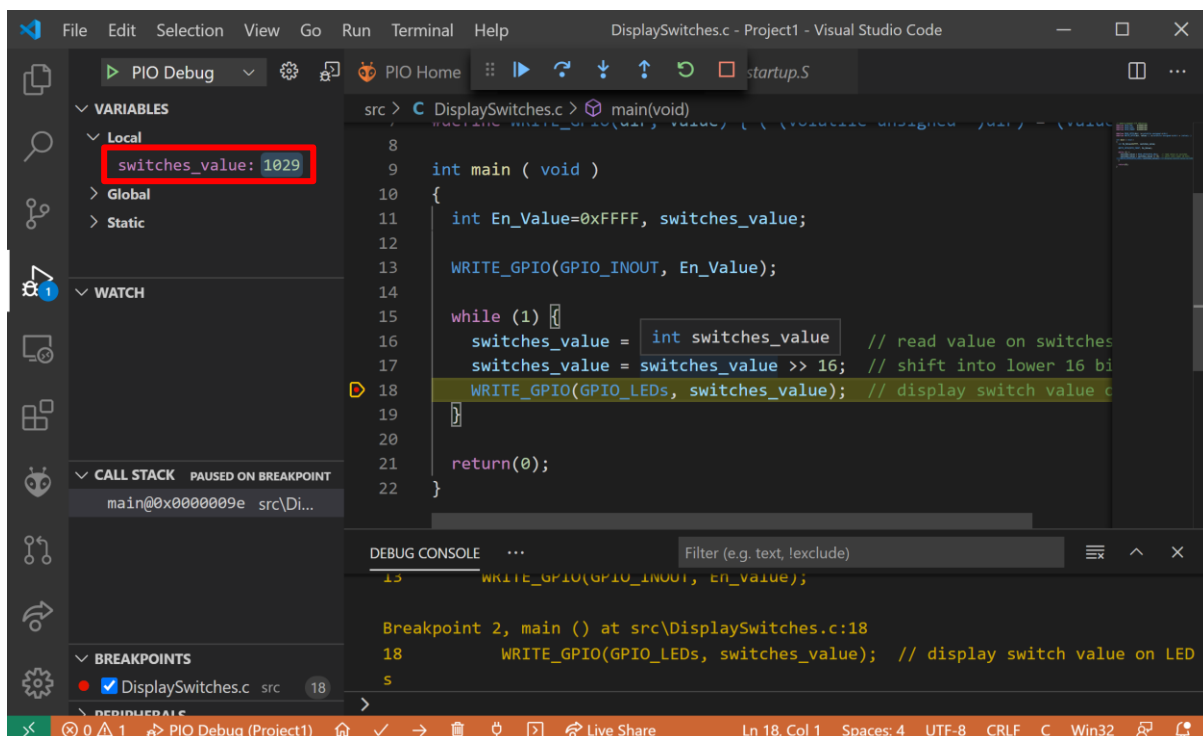

Figure 17. Viewing values of variables

You may also view the RISC-V assembly code generated from the C program. Do so by clicking on DISASSEMBLY → Switch to assembly, as shown in Figure 18.
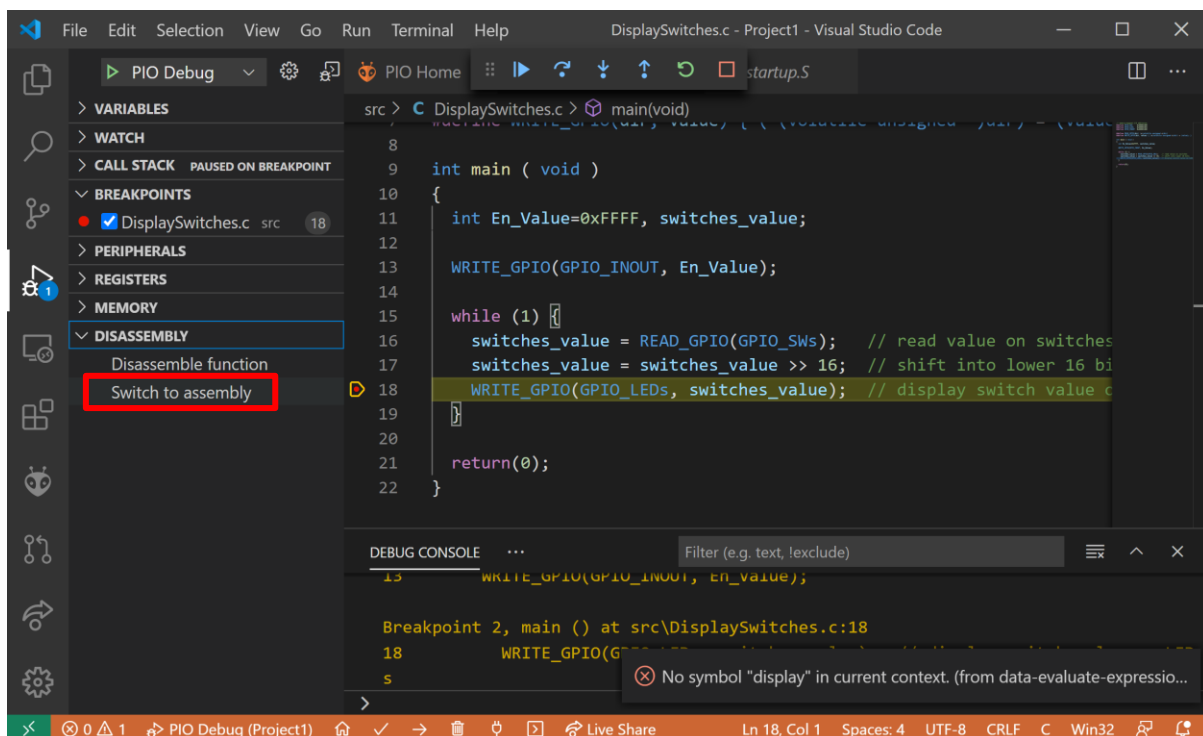

Figure 18. View RISC-V assembly code

Now the RISC-V assembly shows up in the viewing pane, as shown in Figure 19. The assembly shows the memory address of the instruction, the machine code, and the assembly code. As you can see, the C code compiles to a mix of compressed (16-bit instructions) and 32-bit instructions. Below is the commented assembly code.

```
# 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
```
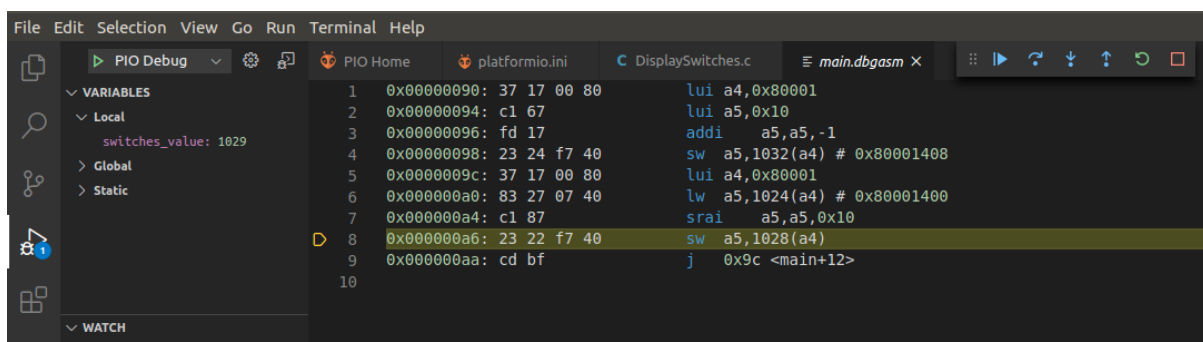

Figure 19. View RISC-V assembly code

Click on DISASSEMBLY → Switch to code to view the C code again.

After you are finished running/debugging the program, stop the debugging session by

pressing the Stop button (or Shift - F5) and go back to the

Explorer window by clicking on at the top of the left-most side bar. Notice that the
program continues running on RVfpgaNexys – only the debugging session ends. Close
the project by clicking on *File → Close Folder* in the top menu bar.

3. Using printf and the serial monitor

Using print statements in a program is a useful way to track program progress or provide
users with feedback, for example, computation results. Recall from the RVfpga Getting
Started Guide (example *HelloWorld_C-Lang* at Section 6.F) that you can use the
printfNexys function, that is similar to the printf function in typical C programs. To do
so, you must use Western Digital's PSP and BSP (processor support package and board
support package) that provide common functions for a given processor and board, in this
case the SweRV EH1 core and the Nexys A7 FPGA board.

Create a PlatformIO project called PrintfExample in the *[RVfpgaPath]/RVfpga/Labs/Lab01*
folder. Add the following program to that project (see Figure 20). Name the program file
PrintfExample.c.

The program is also available here for your convenience:
   [RVfpgaPath]/RVfpga/Labs/Lab01/PrintfExample.c

```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++;
   }
}
```
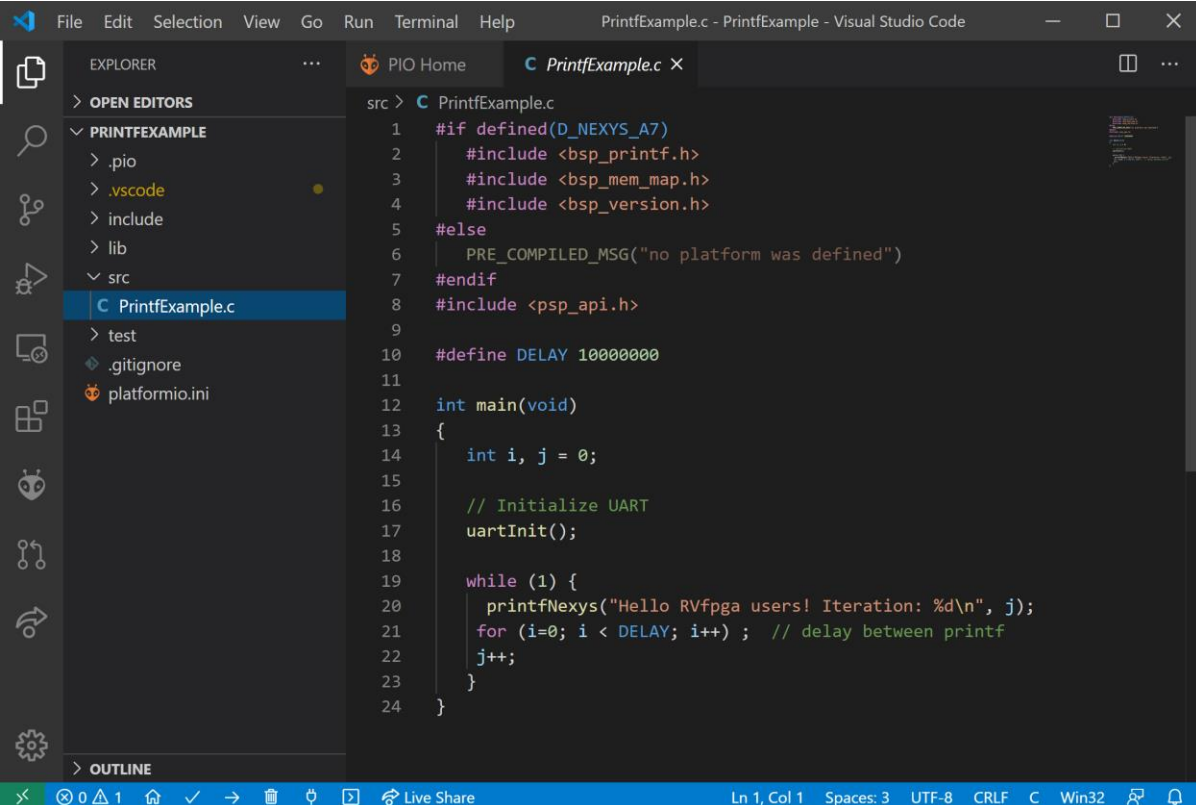
Figure 20. PrintfExample.c

Lines 1-8 (see Figure 20) are the include files needed to use the `printfNexys` function. They are provided in Western Digital's BSP/PSP. Line 17 in Figure 20 calls the `uartInit` function; this line is required to initialize the UART connection used to communicate between RVfpgaNexys (running on the Nexys A7 board) and the serial monitor. Finally, the while loop repeatedly writes to the serial monitor using the `printfNexys` function on line 20 followed by a delay (line 21).

In order to use the `printfNexys` function and the UART, the platfomio.ini file must be modified to include the speed of the UART. Add the following line to the platformio.ini file, as shown in Figure 21:

```
monitor_speed = 115200
```

RVfpgaNexys expects the UART to communicate at 115200 baud (symbols/second), so this rate must be set in platformio.ini, as shown in Figure 21, line 16. Also remember to add the location of your RVfpgaNexys bitfile using `board_build.bitstream_file = …` (as shown in Figure 21, line 18).

Figure 21. Setting UART speed

LINUX: Remember that, if you are using Linux, before being able to use the serial monitor you need to prepare the system by adding your user to the `dialout, tty` and `uucp` groups, as explained in Section 6.F of the Getting Started Guide. If you did this process in the GSG, everything should work; otherwise, do it now.

Now upload the bitfile (as described in Section 2) and run/debug the program.

After the program begins running (and **only after** program begins running), click on the serial monitor button ⚡ at the bottom of the PlatformIO window (see Figure 22).

Warning: If you open the serial monitor before the program begins running (and hits the first – temporary – break point), the UART will be scrambled and not operate correctly.
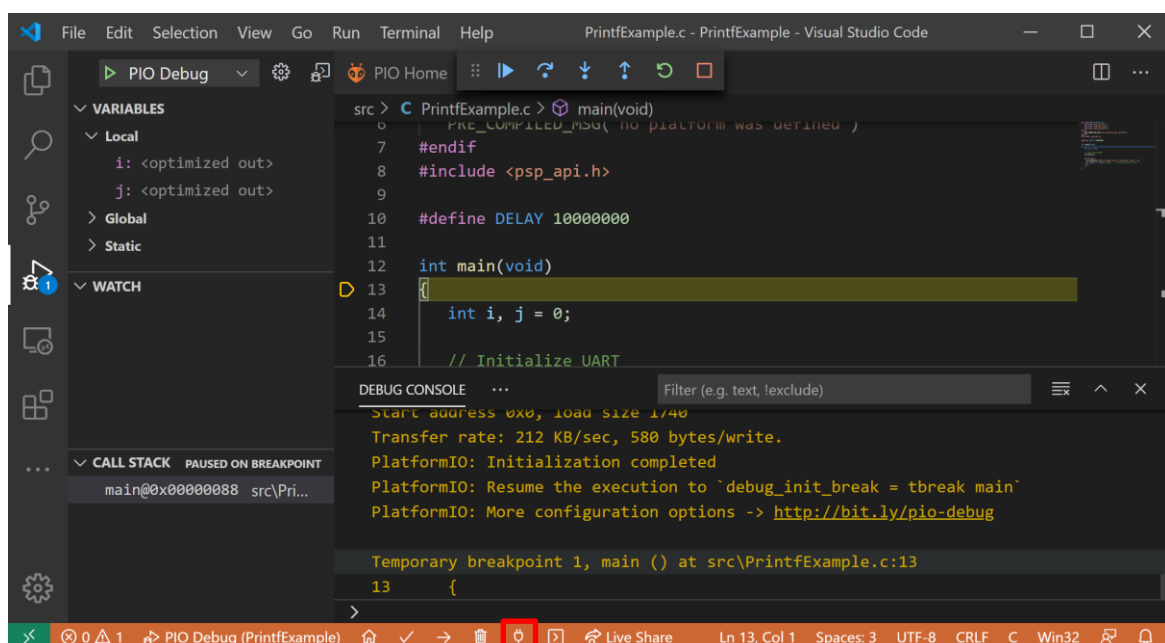

Figure 22. Starting serial monitor

Then run the program: ▶.

You will see the print string (Hello RVfpga users!) followed by the iteration number print repeatedly on the serial monitor, as shown in Figure 23.
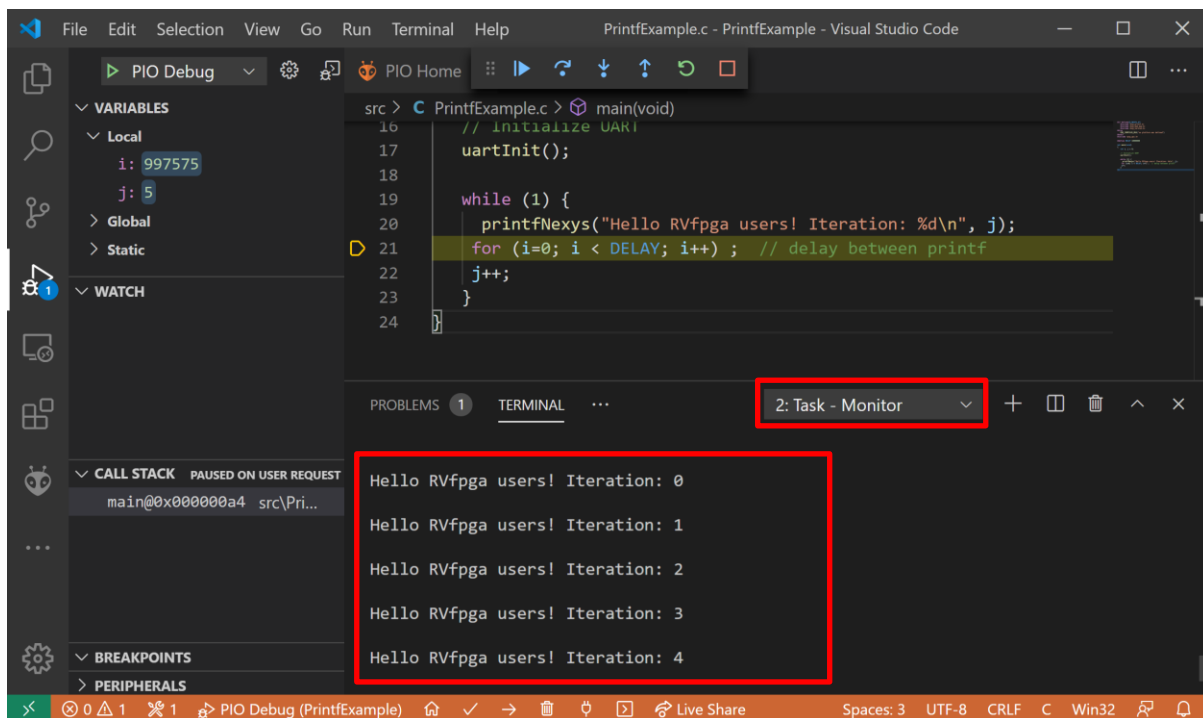

Figure 23. Output of `printfNexys` function on serial monitor in PlatformIO

4. Exercises

Create your own C programs by completing the following exercises. Note that if you leave the Nexys A7 board connected to your computer and powered on, you do not need to reload RVfpgaNexys onto the board between running different programs. However, if you turn off the Nexys A7 board, you will need to reload RVfpgaNexys onto the board using PlatformIO, as described in step 3 of Section 2.

Remember that you can print any variable using Western Digital's BSP function `printfNexys` (see Section 3).

Remember as well that you can run these programs on simulation, using Verilator and Whisper.

Exercise 1. Write a C program that flashes the value of the switches onto the LEDs. The value should pulse on and off slow enough that a person can view the flashing. Name the program **FlashSwitchesToLEDs.c**.

Exercise 2. Write a C program that displays the inverse value of the switches on the LEDs. For example, if the switches are (in binary): 0101010101010101, then the LEDs should display: 1010101010101010; if the switches are: 1111000011110000, then the LEDs should display: 0000111100001111; and so on. Name the program **DisplayInverse.c**.

Exercise 3. Write a C program that scrolls increasing numbers of lit LEDs back and forth until all of the LEDs are lit. Then the pattern should repeat. Name the program **ScrollLEDs.c**.

The program should cause the following to occur:
1. First, one lit LED should scroll from right to left and then left to right.
2. Then two lit LEDs should scroll from right to left and then left to right.
3. Then three lit LEDs should scroll from right to left and then left to right.
4. And so on, until all the LEDs are lit.
5. Then the pattern should repeat.

Exercise 4. Write a C program that displays the unsigned 4-bit addition of the 4 least significant bits of the switches and the 4 most significant bits of the switches. Display the result on the 4 least significant (right-most) bits of the LEDs. Name the program **4bitAdd.c**. The fifth bit of the LEDs should light up when unsigned overflow occurs (that is when the carry out is 1).

Exercise 5. Write a C program that finds the *greatest common divisor* of two numbers, *a* and *b*, according to the Euclidean algorithm. The values *a* and *b* should be statically defined variables in the program. Name the program **GCD.c**. Here is some additional information about the Euclidean algorithm: https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm. You can also simply google "Euclidean algorithm".

Exercise 6. Write a C program that computes the first 12 numbers in the Fibonacci sequence and stores the result in a finite vector (i.e. array), *V*, of length 12. This infinite sequence of Fibonacci numbers is defined as:

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

In words, the Fibonacci number corresponding to element i is the sum of the two previous Fibonacci numbers in the series. Table 1 shows the Fibonacci numbers for i = 0 to 8.

Table 1. Fibonacci series

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

The dimension of the vector, *N*, must be defined in the program as a constant. Name the program **Fibonacci.c**.

Exercise 7. Given an *N*-element vector (i.e., array), *A*, generate another vector, *B*, such that *B* only contains those elements of *A* that are even numbers greater than 0. The C program must also count the number of elements in *B* and print that value at the end of the program. For example: suppose *N*=12 and *A* = [0,1,2,7,-8,4,5,12,11,-2,6,3], then B would be: *B* = [2,4,12,6]. Because B has four elements, the following should print at the end of the program:

```
Number of elements in B = 4.
```

Use the `printfNexys` function to do so. Name the program **EvenPositiveNumbers.c**. Test your program when *A* has 12 elements.

Exercise 8. Given two *N*-element vectors (i.e., arrays), *A* and *B*, create another vector, *C*, defined as:

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

Write a program in C that computes the new vector. Use 12-element arrays in your program. Name the program **AddVectors.c**.

Exercise 9. Implement the bubble sort algorithm in C. This algorithm sorts the components of a vector in ascending order by means of the following procedure:
1. Traverse the vector repeatedly until done.
2. Interchanging any pair of adjacent components if $V(i) > V(i+1)$.
3. The algorithm stops when every pair of consecutive components is in order.

Use 12-element arrays to test your program. Name the program **BubbleSort.c**.

Exercise 10. Write a program in C that computes the factorial of a given non-negative number, *n*, by means of iterative multiplications. While you should test your program for multiple values of n, your final submission should be for *n* = 7. The program should print out the value of factorial(*n*) at the end of the program. *n* should be a variable that is statically defined within the program. Name the program **Factorial.c**.