



**590005 - INGENIERÍA Y ARQUITECTURA DE
COMPUTADORES**

Práctica 2 RVfpga

Programación en C

1. INTRODUCCIÓN

La mayoría de los programas informáticos se escriben en un lenguaje de alto nivel como C. En esta práctica se muestra cómo crear un proyecto C utilizando PlatformIO que se pueda ejecutar en el Sistema RVfpga. En primer lugar, se proporciona un tutorial sobre cómo crear y ejecutar un programa en C, y a continuación se describen ejercicios para que el usuario pueda practicar la escritura de sus propios programas en C.

2. PROGRAMAS EN C PARA RVFPGA

El usuario completará los siguientes pasos para crear y ejecutar un programa en C sobre RVfpgaNexys usando PlatformIO (recuerda que puede ejecutar estos programas en simulación, en Verilator o en Whisper):

1. Crear un proyecto RVfpga
2. Escribir un programa en C
3. Descargar RVfpgaNexys en la placa FPGA Nexys A7
4. Compilar, descargar y ejecutar el programa en C

Paso 1. Crear un proyecto RVfpga

Abra VSCode (como se describe en la Guía de inicio de RVfpga). Si PlatformIO no se abre automáticamente al iniciar VSCode, haga clic en el icono de PlatformIO en la barra de menú de la izquierda y a continuación haga clic en PIO Home → Open (ver Figura 1).

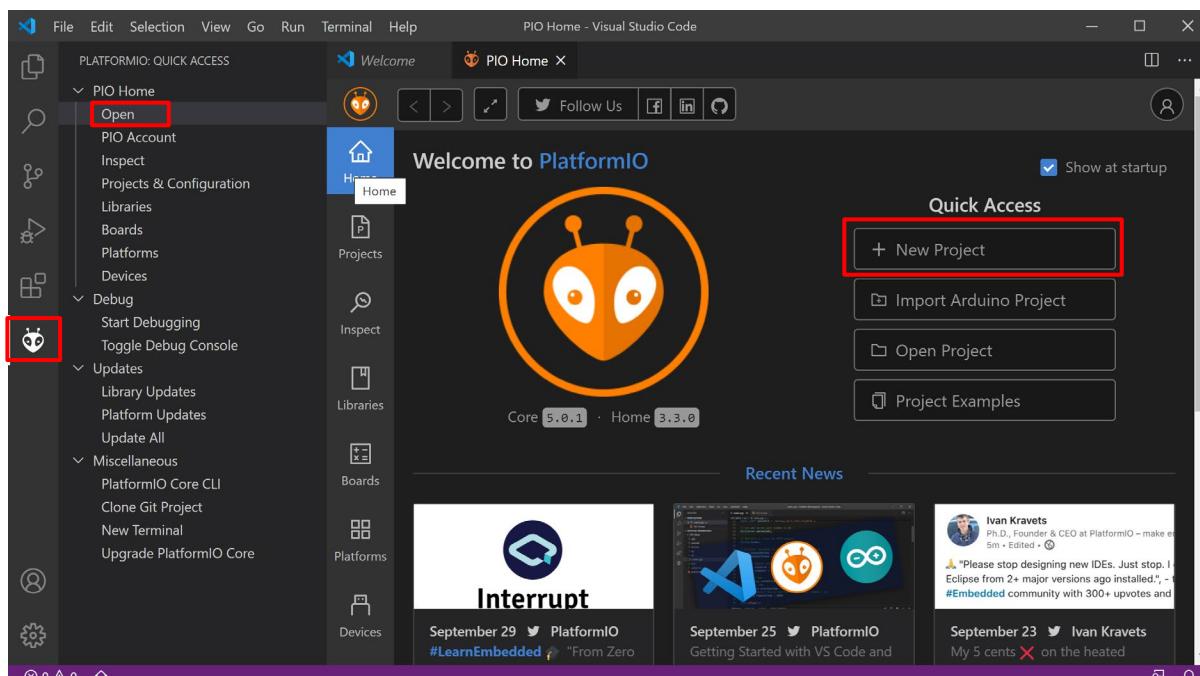


Figura 1. Abrir PlatformIO y crear un nuevo proyecto

Ahora en la ventana de bienvenida de PIO Home, haga clic en New Project (ver Figura 1).

Como se muestra en la figura 2, como nombre del proyecto utilice Project1 y elija como placa “RVfpga: Digilent Nexys A7” (si comienza a escribir RVfpga la placa aparecerá como opción). Deje WD-firmware como Framework por defecto. WD-firmware es el firmware de Western Digital, e incluye el gcc y gdb de Freedom-E SDK así como el PSP y BSP

(paquetes de soporte del procesador y de la placa, respectivamente) que se usan en estas prácticas. Desactive la opción de usar la ubicación predeterminada y guarde su proyecto en:

[RVfpgaPath]/RVfpga/Labs/Lab2

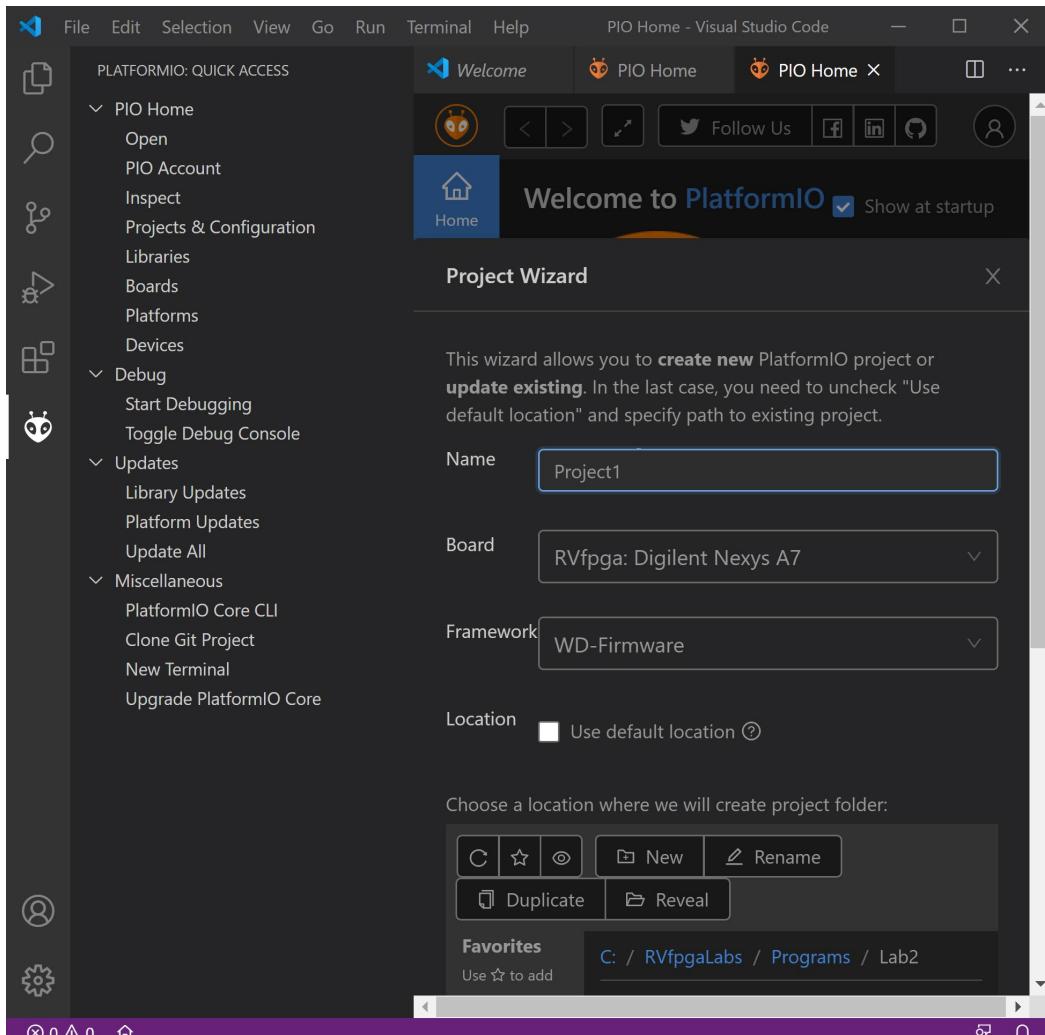


Figura 2. Nombre del proyecto y selección de placa y ubicación del proyecto

A continuación, haga clic en Finalizar en la parte inferior de la ventana (ver Figura 3).

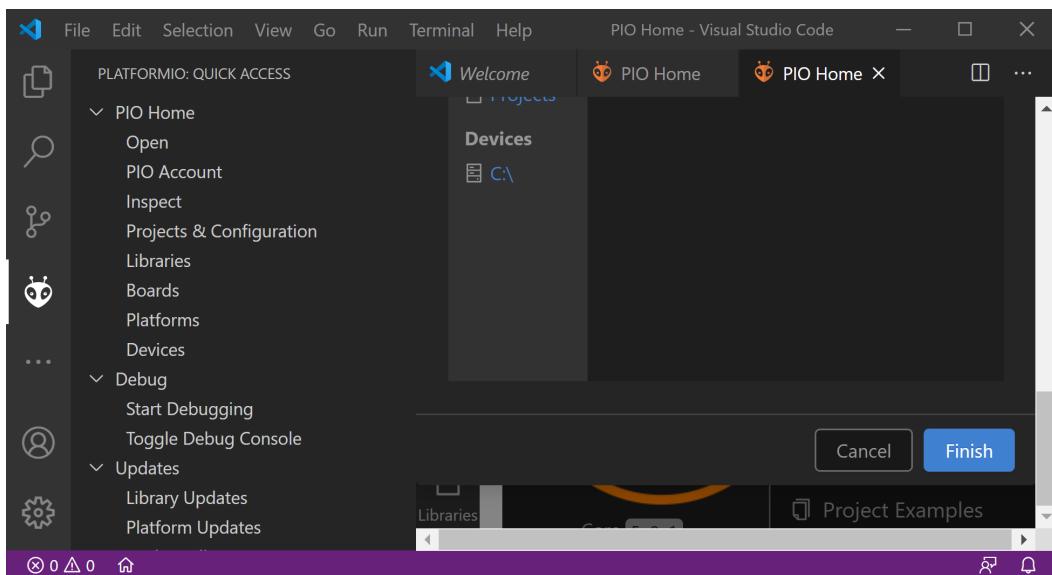


Figura 3. Finalizar la creación del proyecto

En el explorador del proyecto en la zona izquierda (que tal vez necesite ampliar), haga doble clic en platformio.ini para abrirlo (véase la figura 4). Este es el archivo de inicialización de PlatformIO.

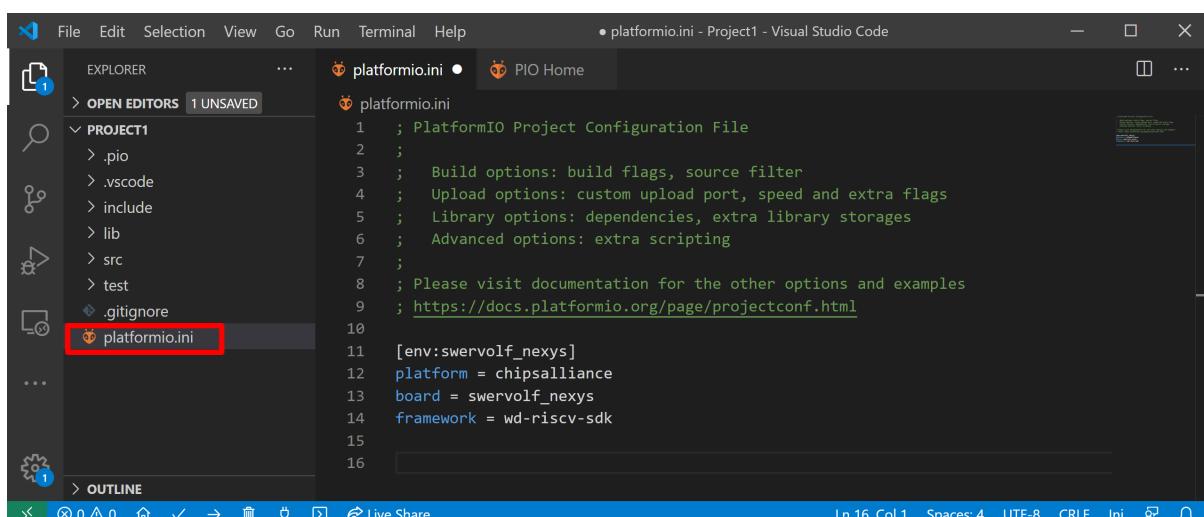
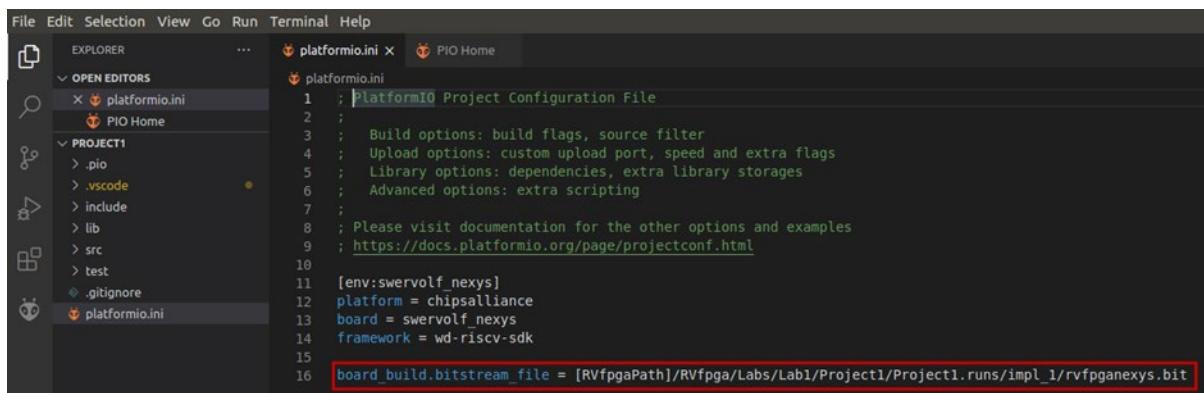


Figura 4. Archivo de inicialización de PlatformIO: platformio.ini

Añada la siguiente línea al archivo platformio.ini, como se muestra en la figura 5:

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

(Recuerde reemplazar *[RVfpgaPath]* con la ubicación de esta carpeta en su computadora). La línea anterior le indica a PlatformIO en dónde debe buscar el archivo bitstream para cargar en la FPGA. La ruta incluida en la mencionada línea es la ubicación del archivo bitstream que se creó en la Práctica 1. (Si no ha completado la Práctica 1, puede usar el archivo bitstream de RVfpgaNexys distribuido con la Guía de inicio en: *[RVfpgaPath]/RVfpga/src/rvfpganexys.bit*). Presione Ctrl-s para guardar el archivo platformio.ini.



```

File Edit Selection View Go Run Terminal Help
EXPLORER ... platformio.ini PIO Home
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 = chipsaliance
13 board = swervolf_nexys
14 framework = wd-riscv-sdk
15
16 board_build.bitstream_file = [RVfpgaPath]/RVfpga/Labs/Lab1/Project1/Project1.runs/impl_1/rvfpganexys.bit

```

Figura 5. Agregar la ubicación del archivo bitstream de RVfpgaNexys (rvfpganexys.bit)

Recuerde que se utilizó un archivo *platformio.ini* más completo en los ejemplos utilizados en la Guía de inicio. Si desea utilizar alguna funcionalidad que requiera comandos adicionales (como la ruta al simulador Verilator, la configuración de la consola serie, la herramienta de depuración *whisper*, etc.), puede utilizar el archivo *platformio.ini* de esos ejemplos.

Paso 2. Escribir un programa en C

Ahora el usuario escribirá un programa en C. Haga clic en File → New File (ver Figura 6)

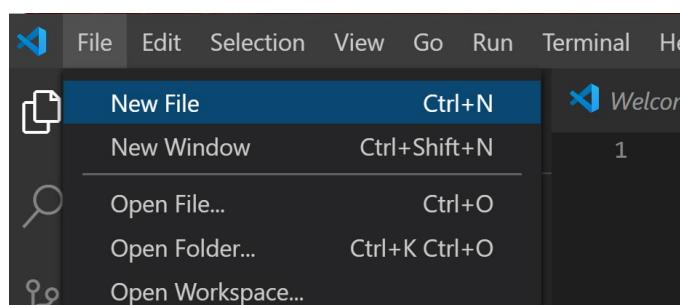


Figura 6. Añadir un archivo al proyecto

Se abrirá una ventana en blanco. Escriba (o copie/pegue) el siguiente programa en C en esa ventana (ver Figura 7). Este programa muestra el valor de los interruptores en los LED.

```

// 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);

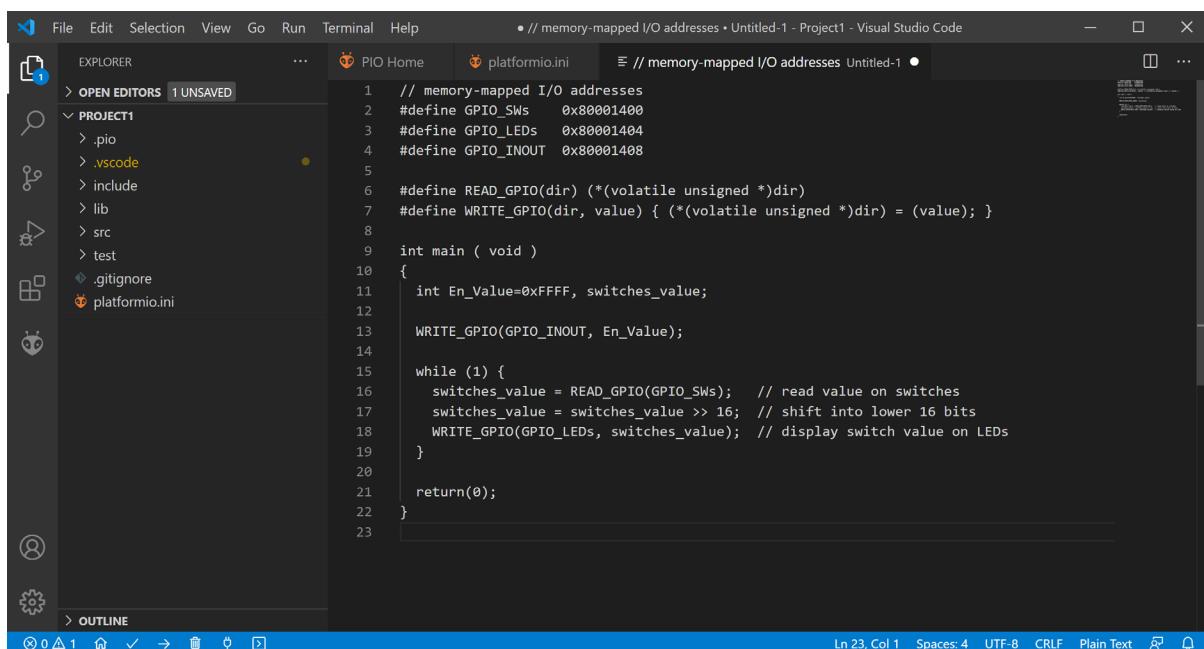
    while (1) {
        switches_value = READ_GPIO(GPIO_SWs); // read value on switches
        switches_value = switches_value >> 16; // shift into lower 16 bits
        WRITE_GPIO(GPIO_LEDs, switches_value); // display switch value on LEDs
    }
}

```

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

Para su comodidad, este programa también está disponible en el siguiente archivo:

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



```
// memory-mapped I/O addresses • Untitled-1 - Project1 - Visual Studio Code
PIO Home platformio.ini // memory-mapped I/O addresses Untitled-1

1 // memory-mapped I/O addresses
2 #define GPIO_SWS 0x80001400
3 #define GPIO_LEDs 0x80001404
4 #define GPIO_INOUT 0x80001408
5
6 #define READ_GPIO(dir) (*(volatile unsigned *)dir)
7 #define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }
8
9 int main ( void )
10 {
11     int En_Value=0xFFFF, switches_value;
12
13     WRITE_GPIO(GPIO_INOUT, En_Value);
14
15     while (1) {
16         switches_value = READ_GPIO(GPIO_SWS); // read value on switches
17         switches_value = switches_value >> 16; // shift into lower 16 bits
18         WRITE_GPIO(GPIO_LEDs, switches_value); // display switch value on LEDs
19     }
20
21     return(0);
22 }
```

Figura 7. Introducción del programa en C

Después de introducir el programa, presione Ctrl-s para guardar el archivo. Llámelo **DisplaySwitches.c** y guárdelo en la carpeta **src** del directorio **Project1** (ver Figura 8).

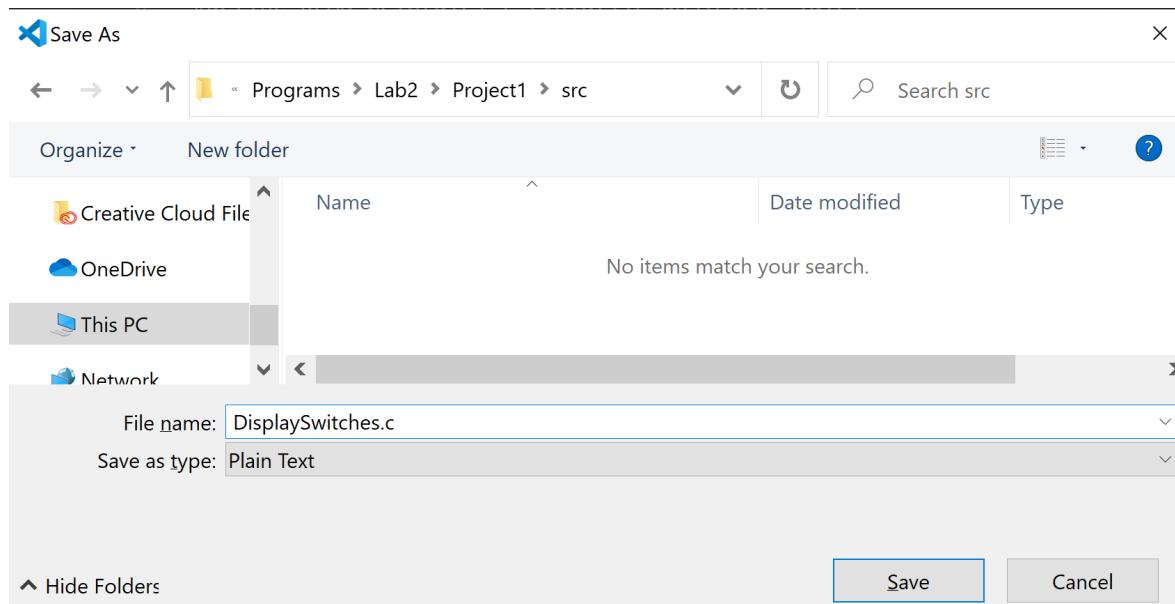


Figura 8. Guardado de archivo como DisplaySwitches.c

Este programa define en primer lugar las direcciones de los registros de Entrada/Salida mapeados en memoria y conectados a los LEDs e interruptores de la placa FPGA Nexys A7, utilizando para ello las siguientes líneas:

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

El valor de los interruptores se obtiene leyendo el registro mapeado a la dirección 0x80001400, mientras que los valores se muestran en los LEDs escribiendo el registro mapeado a la dirección 0x80001404. Los valores de los interruptores están en la mitad del registro correspondiente a los bits más significativos y los LEDs en los bits menos significativos.

El registro GPIO_INOUT define si un bit de la Entrada/Salida de propósito general (GPIO) es una entrada o una salida. Los 16 pines GPIO menos significativos, 15:0, están conectados a los 16 LEDs de la placa Nexys A7. Los 16 pines GPIO más significativos, 31:16, son para los 16 interruptores de la placa. Un 0 indica una entrada y un 1 indica una salida. Así, el registro GPIO_INOUT se escribe con 0xFFFF para que los interruptores sean entradas en RVfpgaNexys y los LEDs sean salidas controladas por RVfpgaNexys.

La figura 9 muestra las ubicaciones físicas de los LED e interruptores de la placa FPGA Nexys A7, así como el conector USB, el interruptor de encendido, los pulsadores y los displays de 7 segmentos.

Tenga en cuenta que en la Práctica 6 se describen en detalle las características de la GPIO y el hardware GPIO de RVfpgaNexys. En prácticas posteriores (Prácticas 6-10), también se analiza cómo usar los otros periféricos de la placa, como los pulsadores y los displays de 7 segmentos.

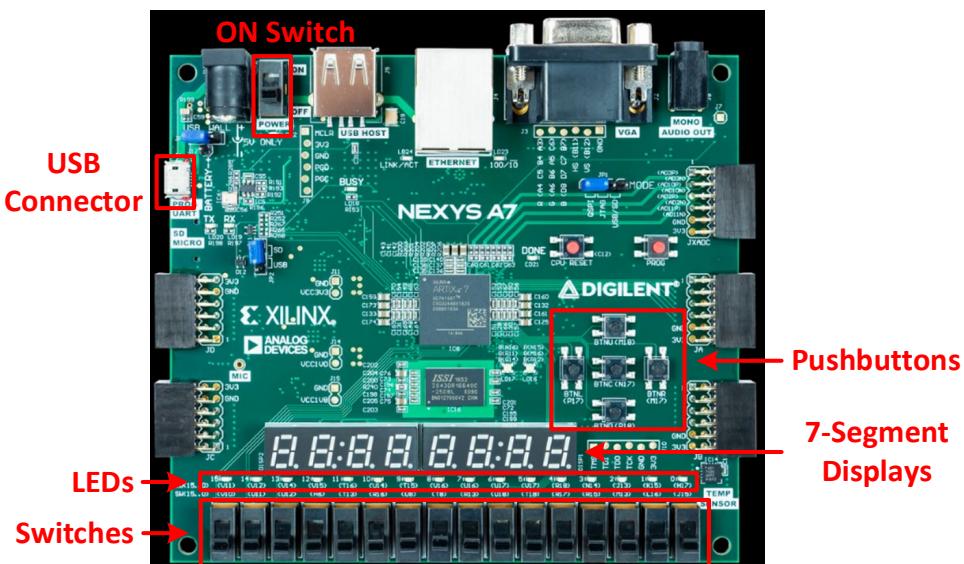


Figura 9. Interfaces de E/S de la placa FPGA Nexys A7 de Digilent
(figura de la placa extraída de <https://reference.digilentinc.com/>)

Después de definir las direcciones de E/S mapeadas en memoria correspondientes a los LED e interruptores, el programa hace lo siguiente:

1. Define los 16 pines GPIO más significativos (que están conectados a los interruptores) como entradas poniendo la mitad superior del registro GPIO_INOUT a 0 y define los 16 pines GPIO menos significativos (que están conectados a los LEDs) como salidas poniendo la mitad inferior del registro GPIO_INOUT a 1, mediante la ejecución del siguiente código:

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

2. Ejecutando el siguiente código, lee repetidamente el valor de los interruptores y escribe ese valor en los LEDs. Recuerde que el valor de los interruptores se lee en la mitad superior del registro de Entrada/Salida mapeado en memoria, por lo que el valor debe desplazarse 16 bits a la derecha antes de escribirlo en el registro de Entrada/Salida mapeado en memoria conectado físicamente a los LEDs.

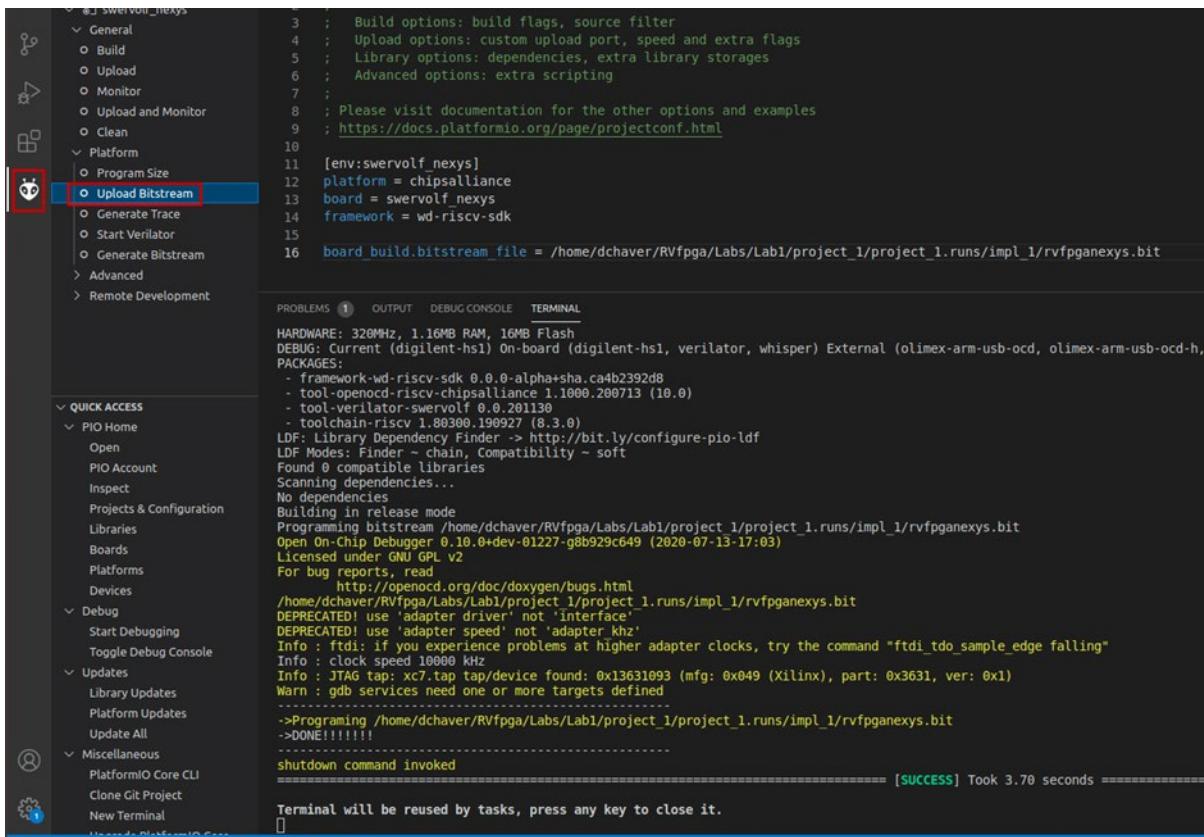
```
while (1) {  
    switches_value = READ_GPIO(GPIO_SWS);      // read value on switches  
    switches_value = switches_value >> 16;      // shift into lower 16 bits  
    WRITE_GPIO(GPIO_LEDs, switches_value);      // display switch value on LEDs  
}
```

Las macros READ_GPIO y WRITE_GPIO leen o escriben, respectivamente, un valor en una dirección específica de Entrada/Salida mapeada en memoria.

Paso 3. Descargar RVfpgaNexys en la placa FPGA Nexys A7

Ahora se procederá a la descarga de RVfpgaNexys en la placa FPGA Nexys A7. Haga clic en el icono de PlatformIO en la barra del menú de la izquierda, luego expanda Project Tasks → env:swervolf_nexys → Platform y haga clic en Upload Bitstream, como se muestra en la figura 10.

Nota: si utiliza un sistema **Windows** y no ha sustituido aún el controlador de la placa FPGA Nexys A7, hágalo siguiendo las instrucciones del Apéndice B de la Guía de inicio.



```

@J swervolf_nexys
  General
    Build
    Upload
    Monitor
    Upload and Monitor
    Clean
  Platform
    Program Size
    Upload Bitstream
      Generate Trace
      Start Verilator
      Generate Bitstream
    Advanced
    Remote Development

[env:swervolf_nexys]
platform = chipsalliance
board = swervolf_nexys
framework = wd-riscv-sdk

board_build.bitstream_file = /home/dchaver/RVfpga/Labs/Lab1/project_1/project_1.runs/impl_1/rvfganexys.bit

PROBLEMS ① OUTPUT DEBUG CONSOLE TERMINAL
HARDWARE: 320MHz, 1.16MB RAM, 16MB Flash
DEBUG: Current (digilent-hs1) On-board (digilent-hs1, verilator, whisper) External (olimex-arm-usb-ocd, olimex-arm-usb-ocd-h,
PACKAGES:
- framework-wd-riscv-sdk 0.0.0-alpha+sha.ca4b2392d8
- tool-openocd-riscv-chipsalliance 1.1000.200713 (10.0)
- tool-verilator-swervolf 0.0.201130
- toolchain-riscv 1.80300.190927 (8.3.0)
LDF: Library Dependency Finder -> http://bit.ly/configure-pio-ldf
LDF Modes: Finder ~ chain, Compatibility ~ soft
Found 0 compatible libraries
Scanning dependencies...
No dependencies
Building in release mode
Programming bitstream /home/dchaver/RVfpga/Labs/Lab1/project_1/project_1.runs/impl_1/rvfganexys.bit
Open On-Chip Debugger 0.10.0+dev-01227-g8b929c649 (2020-07-13-17:03)
Licensed under GNU GPL v2
For bug reports, read
http://openocd.org/doc/doxygen/bugs.html
/home/dchaver/RVfpga/Labs/Lab1/project_1/project_1.runs/impl_1/rvfganexys.bit
DEPRECATED! use 'adapter driver' not 'interface'
DEPRECATED! use 'adapter speed' not 'adapter khz'
Info : ftdi: if you experience problems at higher adapter clocks, try the command "ftdi_tdo_sample_edge falling"
Info : clock speed 10000 kHz
Info : JTAG tap: xc7.tap (tap/device found: 0x13631093 (mfg: 0x049 (Xilinx), part: 0x3631, ver: 0x1)
Warn : gdb services need one or more targets defined
----->-Programming /home/dchaver/RVfpga/Labs/Lab1/project_1/project_1.runs/impl_1/rvfganexys.bit
->DONE!!!!!!
shutdown command invoked
===== [SUCCESS] Took 3.70 seconds =====
Terminal will be reused by tasks, press any key to close it.

```

Figura 10. Carga de RVfpgaNexys en la placa FPGA Nexys A7 usando PlatformIO

Como alternativa, puede descargar RVfpgaNexys desde una ventana de terminal de PlatformIO como se muestra en la figura 11. Haga clic en el botón PlatformIO: New Terminal () en la parte inferior de la ventana de PlatformIO, y luego escriba (o copie) lo siguiente en el terminal de PlatformIO:

```
pio run -t program_fpga
```

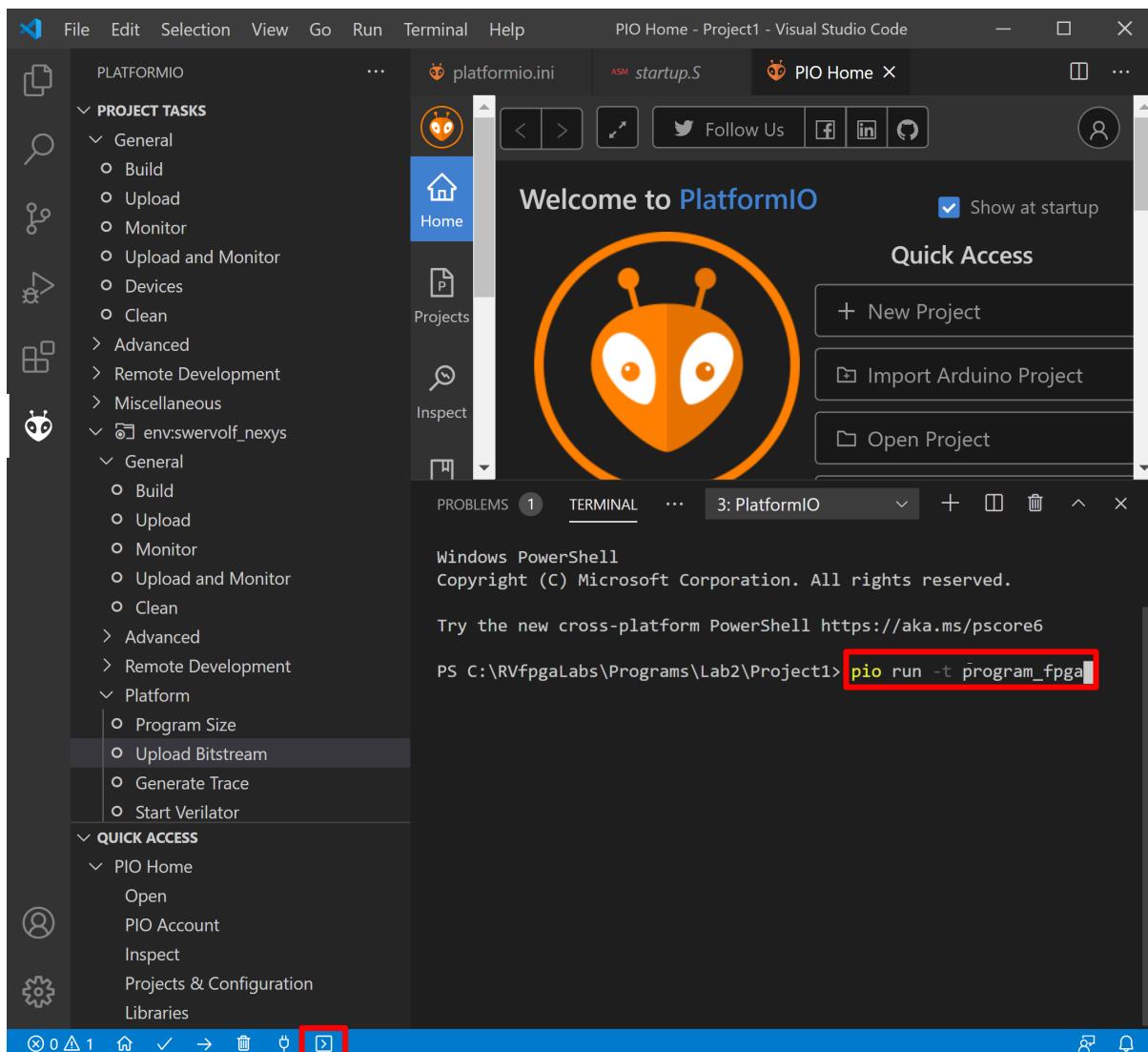
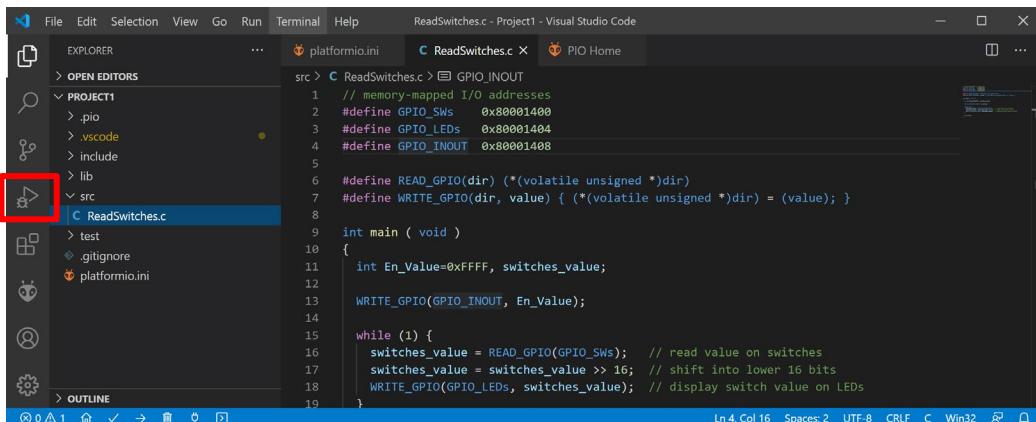


Figura 11. Carga de RVfpgaNexys en la placa FPGA Nexys A7 usando la terminal de PlatformIO

Paso 4. Compilar, descargar y ejecutar el programa en C

Ahora que RVfpgaNexys está cargado en la placa, debe compilar el programa, descargarlo en RVfpgaNexys y ejecutar/depurar el programa. Si VSCode no está ya abierto, ábralo. Su último proyecto, Project1, debería abrirse automáticamente. Si no, asegúrese de que la extensión de PlatformIO esté abierta y haga clic en File → Open Folder y seleccione (pero no abra) Project1, el proyecto que creó anteriormente en esta práctica.

Haga clic en el botón Run en la barra del menú de la izquierda (ver Figura 12).



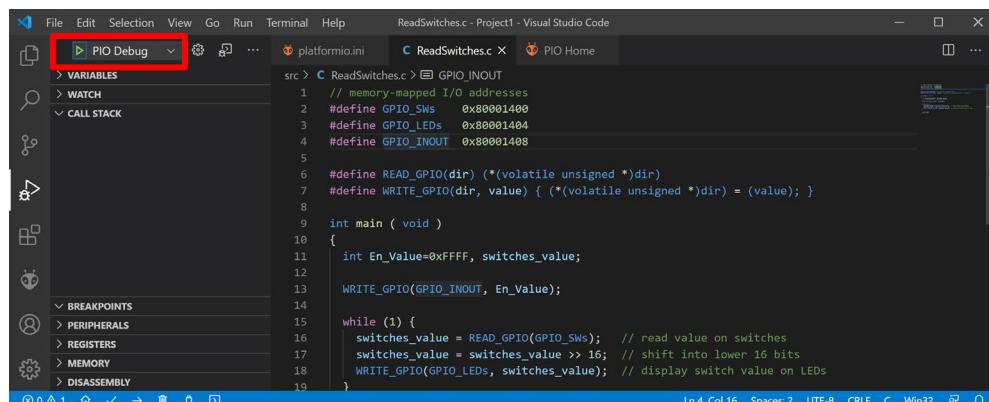
```

File Edit Selection View Go Run Terminal Help ReadSwitches.c - Project1 - Visual Studio Code
src > C ReadSwitches.c > GPIO_INOUT
1 // memory-mapped I/O addresses
2 #define GPIO_SWs 0x80001400
3 #define GPIO_LEDs 0x80001404
4 #define GPIO_INOUT 0x80001408
5
6 #define READ_GPIO(dir) (*(volatile unsigned *)dir)
7 #define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }
8
9 int main ( void )
10 {
11     int En_Value=0xFFFF, switches_value;
12
13     WRITE_GPIO(GPIO_INOUT, En_Value);
14
15     while (1) {
16         switches_value = READ_GPIO(GPIO_SWs); // read value on switches
17         switches_value = switches_value >> 16; // shift into lower 16 bits
18         WRITE_GPIO(GPIO_LEDs, switches_value); // display switch value on LEDs
19     }

```

Figura 12. Ejecución del programa en RVfpgaNexys

Ahora haga clic en el botón Iniciar depuración (ver figura 13).



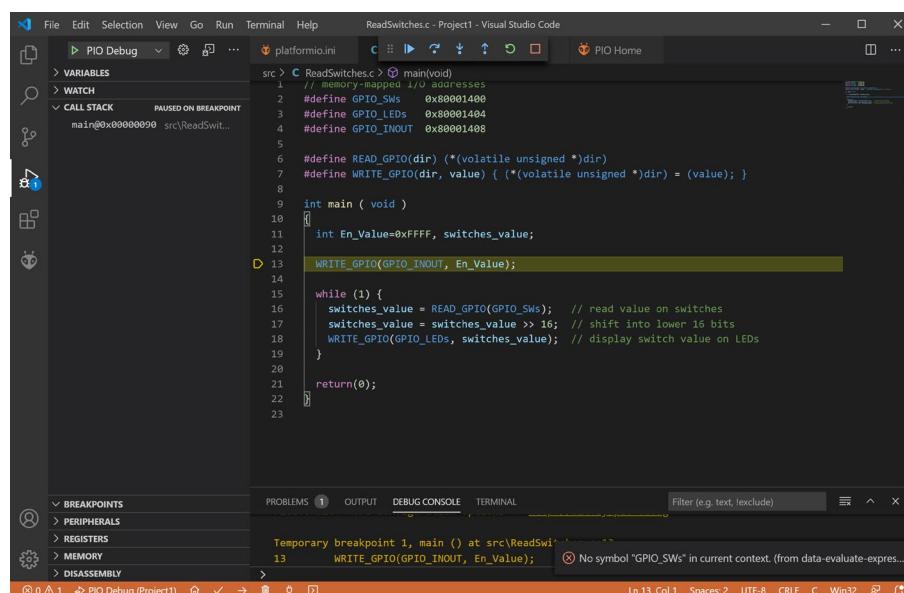
```

File Edit Selection View Go Run Terminal Help ReadSwitches.c - Project1 - Visual Studio Code
src > C ReadSwitches.c > GPIO_INOUT
1 // memory-mapped I/O addresses
2 #define GPIO_SWs 0x80001400
3 #define GPIO_LEDs 0x80001404
4 #define GPIO_INOUT 0x80001408
5
6 #define READ_GPIO(dir) (*(volatile unsigned *)dir)
7 #define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }
8
9 int main ( void )
10 {
11     int En_Value=0xFFFF, switches_value;
12
13     WRITE_GPIO(GPIO_INOUT, En_Value);
14
15     while (1) {
16         switches_value = READ_GPIO(GPIO_SWs); // read value on switches
17         switches_value = switches_value >> 16; // shift into lower 16 bits
18         WRITE_GPIO(GPIO_LEDs, switches_value); // display switch value on LEDs
19     }

```

Figura 13. Comienzo de la ejecución y depuración del programa

El programa compilará y luego se descargará en RVfpgaNexys, que se ejecuta en la FPGA de la placa Nexys A7. (Tenga en cuenta que puede aparecer en la terminal un aviso sobre la falta del archivo sys/cdefs.h, pero el programa sigue funcionando correctamente). Ahora puede comenzar a ejecutar y depurar el programa (ver Figura 14).



```

File Edit Selection View Go Run Terminal Help ReadSwitches.c - Project1 - Visual Studio Code
src > C ReadSwitches.c > GPIO_INOUT
1 // memory-mapped I/O addresses
2 #define GPIO_SWs 0x80001400
3 #define GPIO_LEDs 0x80001404
4 #define GPIO_INOUT 0x80001408
5
6 #define READ_GPIO(dir) (*(volatile unsigned *)dir)
7 #define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }
8
9 int main ( void )
10 {
11     int En_Value=0xFFFF, switches_value;
12
13     WRITE_GPIO(GPIO_INOUT, En_Value);
14
15     while (1) {
16         switches_value = READ_GPIO(GPIO_SWs); // read value on switches
17         switches_value = switches_value >> 16; // shift into lower 16 bits
18         WRITE_GPIO(GPIO_LEDs, switches_value); // display switch value on LEDs
19     }
20
21     return(0);
22 }


```

Figura 14. Programa ejecutándose en RVfpgaNexys

Como se describe en la Guía de inicio de RVfpga, para controlar su sesión de depuración, utilice la barra de herramientas de depuración que aparece en la parte superior del editor (véase la figura 15). A continuación se muestran las opciones:

1. **Continue** ejecuta el programa hasta el siguiente punto de interrupción (*breakpoint*).
2. **Breakpoints** Los puntos de interrupción se pueden añadir haciendo clic a la izquierda del número de línea en el editor.
3. **Step Over** ejecuta la línea actual y luego se detiene.
4. **Step Into** ejecuta la línea actual y si la línea actual incluye una llamada a función, saltará a esa función y se detendrá.
5. **Step Out** ejecuta todo el código de la función en la que está y luego se detiene una vez que la función regresa.
6. **Restart** reinicia la sesión de depuración desde el principio del programa.
7. **Stop** detiene la sesión de depuración y vuelve al modo de edición normal. Tenga en cuenta que cuando pulsa el botón Stop, el **programa sigue ejecutándose** en RVfpgaNexys, pero la sesión de depuración termina.
8. **Pause** detiene la ejecución. Cuando el programa se está ejecutando, el botón de Continue es reemplazado por el botón de Pause.



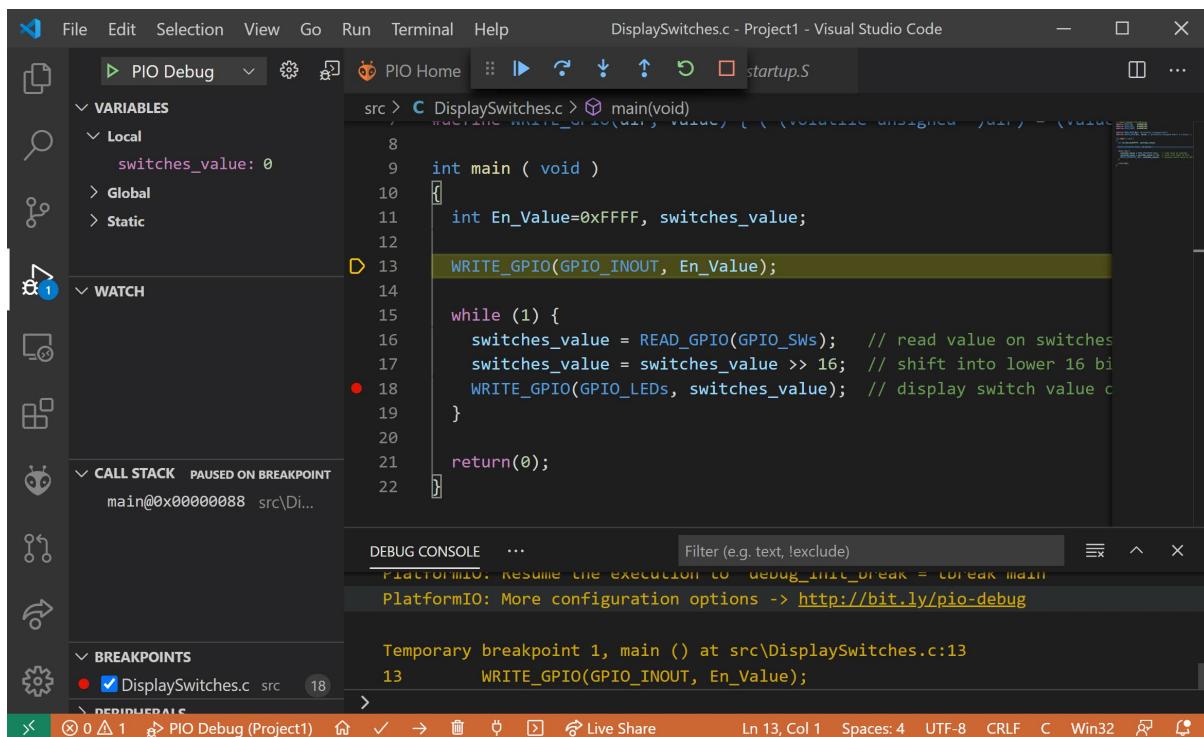
Figura 15. Herramientas de depuración

En la barra lateral izquierda, puede ver las opciones del Depurador. Están disponibles las siguientes opciones:

- **Variables:** lista las variables locales, globales y estáticas presentes en su programa junto con sus valores.
- **Call Stack:** muestra la función que se está ejecutando actualmente, la función que la llama (si la hay) y la ubicación de la instrucción actual en la memoria.
- **Breakpoints:** muestra cualquier punto de interrupción establecido y resalta su número de línea. Los puntos de interrupción se pueden gestionar en esta sección, y también se pueden desactivar temporalmente sin eliminarlos cambiando la casilla de verificación.
- **Peripherals:** muestra el estado de los registros de los periféricos del dispositivo mapeados en memoria.
- **Registers:** lista los valores actuales de cada uno de los registros del procesador.
- **Memory:** muestra el contenido de una dirección específica de la memoria.
- **Disassembly:** muestra el código ensamblado para una función específica (para código de alto nivel como C, esto permite ver el código ensamblado para depurar las instrucciones una por una).

Por ejemplo, haga clic a la izquierda de la línea 18 para establecer un punto de interrupción justo antes de que el valor de los interruptores se escriba en los LED, como se muestra en

la figura 16. Ahora ejecute el programa haciendo clic en el botón Continue (o pulsando F5). El programa continuará hasta que alcance el punto de interrupción establecido. (Para eliminar un punto de interrupción, haga clic en el punto de interrupción existente, justo a la izquierda del número de línea).



```

src > C DisplaySwitches.c > main(void)
8
9 int main ( void )
10 {
11     int En_Value=0xFFFF, switches_value;
12
13     WRITE_GPIO(GPIO_INOUT, En_Value);
14
15     while (1) {
16         switches_value = READ_GPIO(GPIO_SWs); // read value on switches
17         switches_value = switches_value >> 16; // shift into lower 16 bi
18         WRITE_GPIO(GPIO_LEDs, switches_value); // display switch value c
19
20     }
21
22     return(0);
}

```

DEBUG CONSOLE ... Filter (e.g. text, exclude)

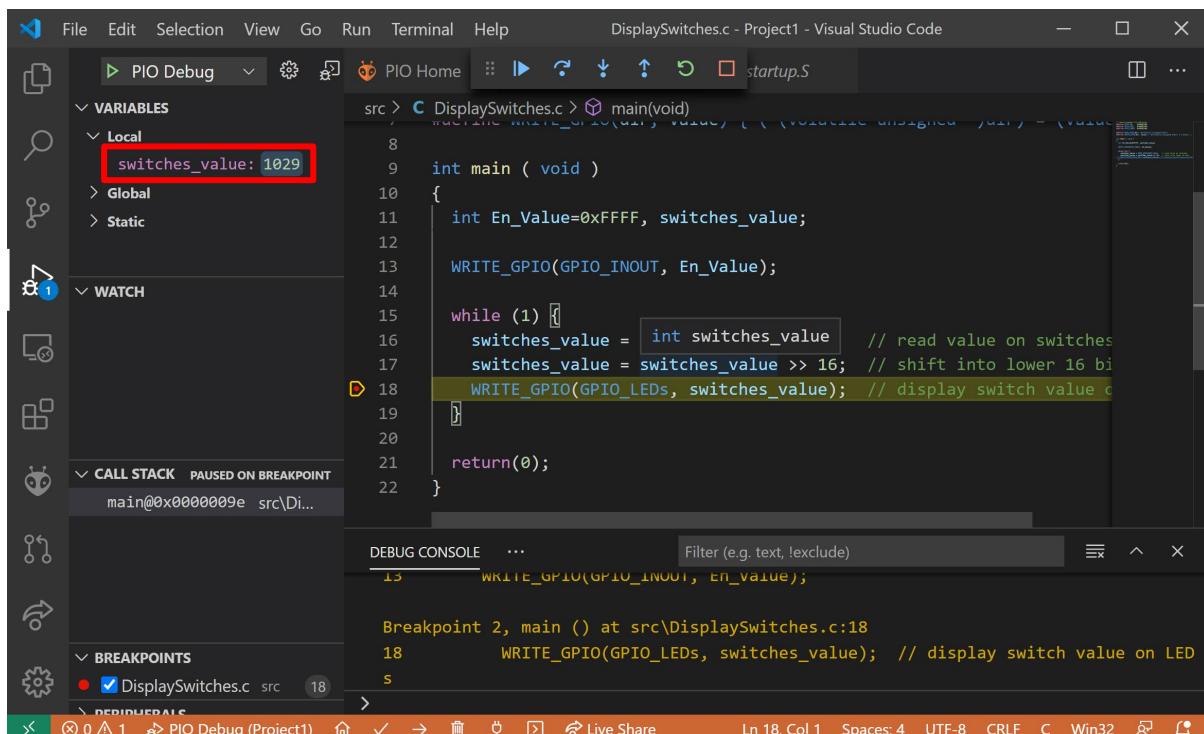
Temporary breakpoint 1, main () at src\DisplaySwitches.c:13

13 WRITE_GPIO(GPIO_INOUT, En_Value);

Figura 16. Establecer un punto de interrupción antes de mostrar el valor en los LEDs

Después de alcanzar el punto de interrupción, amplíe la sección Variables del panel a la izquierda y visualice el valor de la variable `switches_value`, como se muestra en la figura 17. En este caso, el valor de los interruptores es 1029 = 0x405 (en binario, 0000_0100_0000_0101), que corresponde al siguiente patrón:

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



```

src > C DisplaySwitches.c > main(void)
8
9 int main ( void )
10 {
11     int En_Value=0xFFFF, switches_value;
12
13     WRITE_GPIO(GPIO_INOUT, En_Value);
14
15     while (1) {
16         switches_value = int switches_value // read value on switches
17         switches_value = switches_value >> 16; // shift into lower 16 bi
18         WRITE_GPIO(GPIO_LEDs, switches_value); // display switch value c
19
20     }
21
22     return(0);
}

```

DEBUG CONSOLE ... Filter (e.g. text, exclude)

Breakpoint 2, main () at src\DisplaySwitches.c:18

18 WRITE_GPIO(GPIO_INOUT, En_Value);

Figura 17. Visualización de los valores de las variables

También puede ver el código ensamblador de RISC-V generado por el programa en C. Para ello haga clic en DISASSEMBLY → Switch to assembly, como se muestra en la Figura 18.

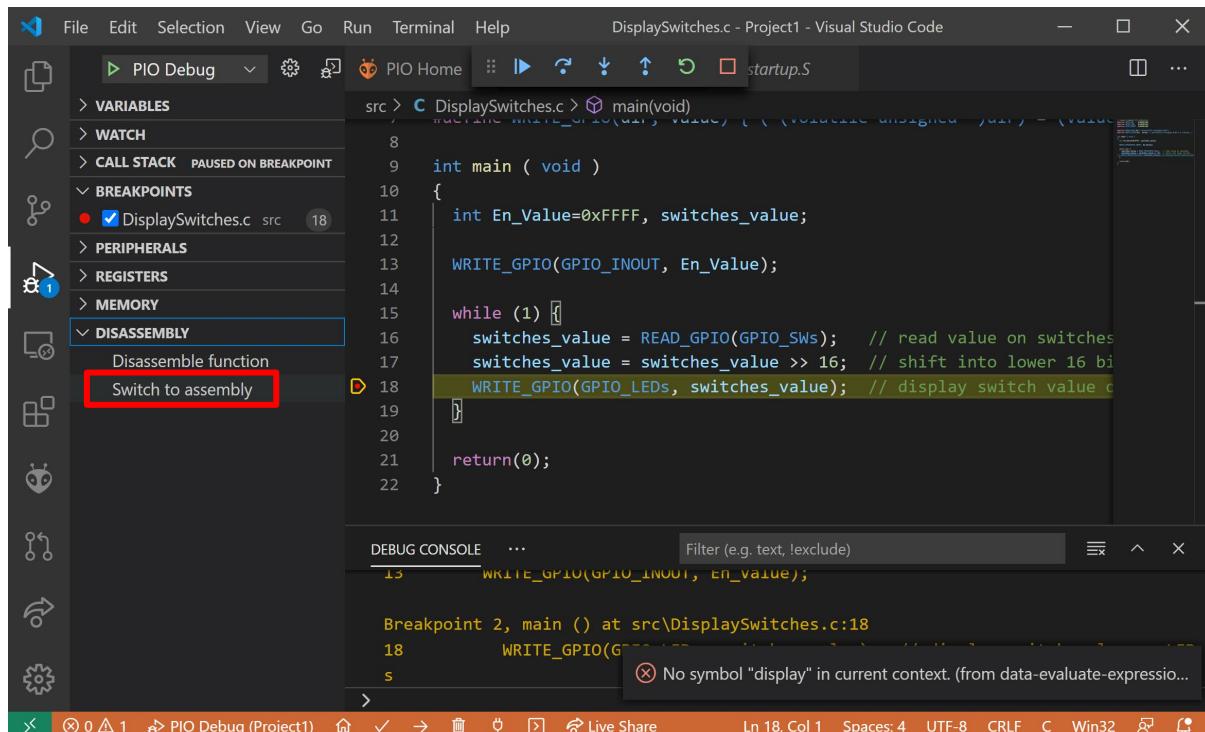


Figura 18. Cambio a visualización del código en ensamblador de RISC-V

Ahora el código en lenguaje ensamblador de RISC-V aparece en el panel de visualización como se muestra en la figura 19. El ensamblador muestra la dirección de memoria de la instrucción, el código máquina y el código en lenguaje ensamblador. Como se muestra, el código C compila a una mezcla de instrucciones comprimidas (instrucciones de 16 bits) y de 32 bits. A continuación se muestra el citado código en ensamblador.

```
# 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
```

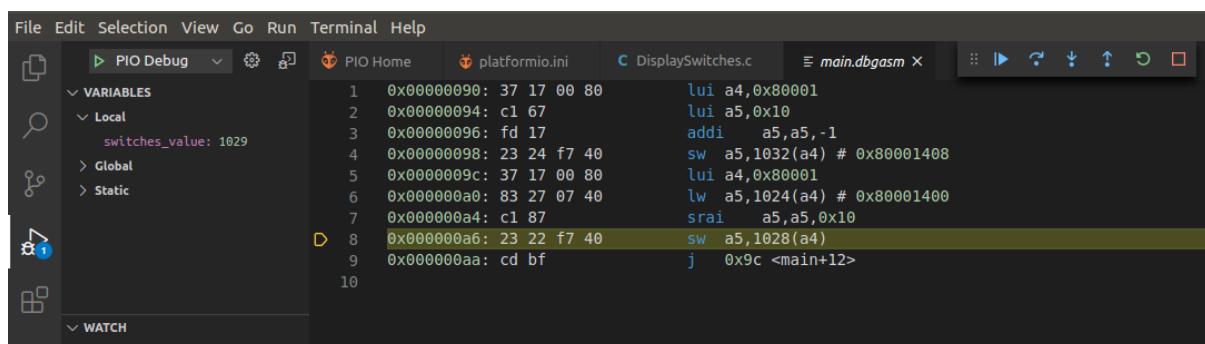


Figura 19. Visualización del código en ensamblador de RISC-V

Haga clic en DISASSEMBLY → Switch to code, para ver el código C de nuevo.

Cuando termine de ejecutar/depurar el programa, detenga la sesión de depuración pulsando el botón Stop  (o Shift + F5) y vuelva a la ventana del explorador pulsando  en la parte superior de la barra lateral izquierda. Tenga en cuenta que el **programa continúa ejecutándose** en RVfpgaNexys, sólo finaliza la sesión de depuración. Cierre el proyecto haciendo clic en *File* → *Close Folder* en la barra del menú superior.

3. Uso de la función printf y el monitor serie

El uso de instrucciones para imprimir mensajes en un programa es una forma útil de seguir el progreso del programa o proporcionar diversa información, como por ejemplo, los resultados de los cálculos. Recuerde que en la Guía de inicio de RVfpga (ejemplo *HelloWorld_C-Lang* en la sección 6.F) se indica que se puede utilizar la función `printfNexys`, que es similar a la función `printf` característica de los programas en C. Para ello, debe utilizar el PSP y el BSP de Western Digital (paquete de soporte de procesador y paquete de soporte de placa) que proporcionan funciones comunes para un procesador y una placa determinados, en este caso el core SweRV EH1 y la placa FPGA Nexys A7.

A continuación, debe crear un proyecto PlatformIO en la carpeta `[RVfpgaPath]/RVfpga/Labs/Lab2` y denominarlo `PrintfExample`. Añada el siguiente programa a ese proyecto (ver Figura 20), y llame al archivo de programa `PrintfExample.c`.

El programa también está disponible aquí para su comodidad:
`[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>

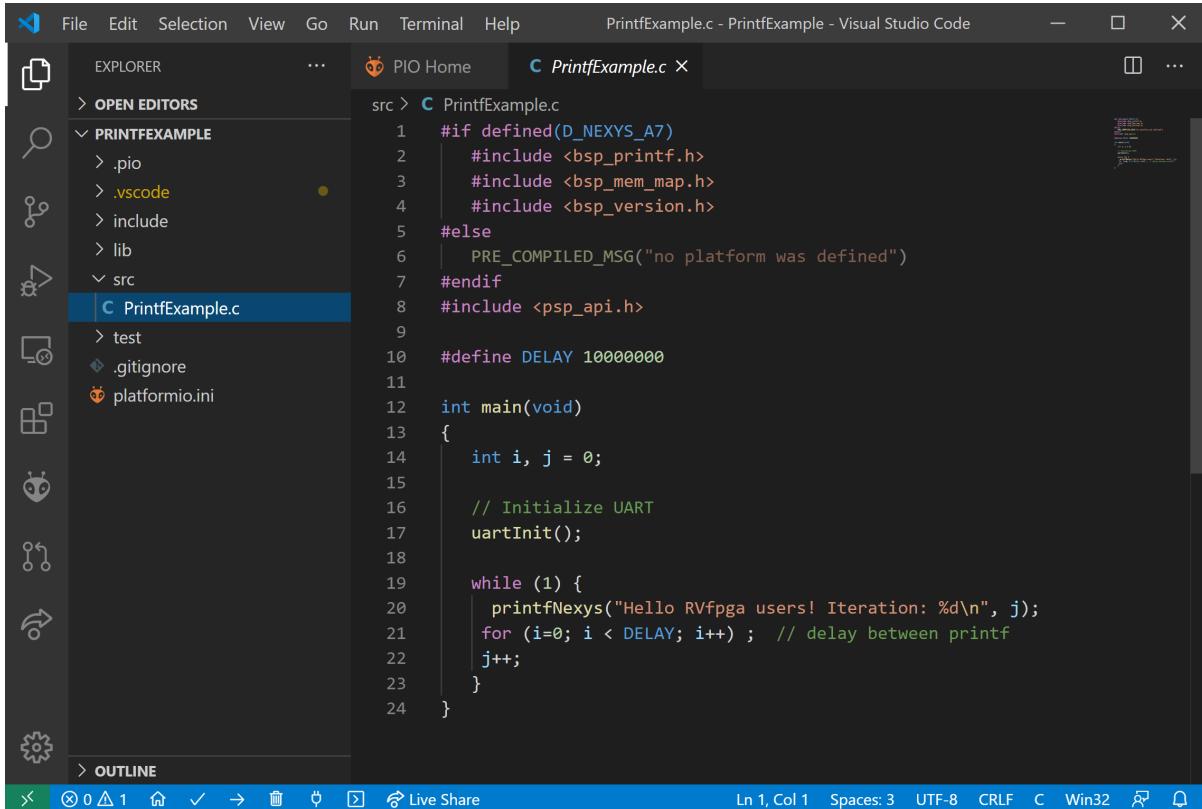
#define DELAY 10000000

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++;
    }
}
```

```
}
```



```

File Edit Selection View Go Run Terminal Help PrintfExample.c - PrintfExample - Visual Studio Code
EXPLORER PIO Home C PrintfExample.c ...
OPEN EDITORS
> PRINTEXAMPLE
> .pio
> .vscode
> include
> lib
> src
C PrintfExample.c
> test
.ignores
platformio.ini
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 }

```

Figura 20. PrintfExample.c

Las líneas 1-8 (véase la figura 20) son los archivos de inclusión necesarios para utilizar la función `printfNexys`. Se proporcionan en el BSP/PSP de Western Digital. La línea 17 de la Figura 20 llama a la función `uartInit`; esta línea es necesaria para inicializar la conexión UART usada para la comunicación entre RVfpgaNexys (que se ejecuta en la placa Nexys A7) y el monitor serie. Finalmente, el bucle while escribe repetidamente en el monitor serie usando la función `printfNexys` en la línea 20 seguida de un retardo (línea 21).

Para poder utilizar la función `printfNexys` y la UART, el archivo `platformio.ini` debe ser modificado para incluir la velocidad de la UART. Añada la siguiente línea al archivo `platformio.ini`, como se muestra en la figura 21:

```
monitor_speed = 115200
```

RVfpgaNexys espera que la UART se comunique a 115200 baudios (símbolos/segundo), por lo que esta velocidad debe fijarse en `platformio.ini`, como se muestra en la figura 21, línea 16. Recuerde también añadir la ubicación de su archivo bitfile de RVfpgaNexys usando `board_build.bitstream_file = ...` (como se muestra en la Figura 21, línea 18).

```

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

```

Figura 21. Ajuste de la velocidad de la UART

LINUX: Recuerde que, si está usando Linux, antes de poder usar el monitor serie necesita preparar el sistema añadiendo su usuario a los grupos `dialout`, `tty` y `uucp`, como se explica en la Sección 6.F de la Guía de Inicio. Si ya realizó este proceso en la GSG, todo debería funcionar; de lo contrario, hágalo ahora. Recuerde reiniciar la sesión después de agregar el usuario a los grupos requeridos.

Cargue el archivo bitfile (proceso descrito en la sección 2) y ejecute/depure el programa.

Después de que el programa comience a ejecutarse (y **sólo después de que** el programa comience a hacerlo), haga clic en el botón del monitor serie  en la parte inferior de la ventana de PlatformIO (ver Figura 22).

Advertencia: Si abre el monitor serie antes de que el programa comience a ejecutar (y llega al primer punto de interrupción), la UART se desconfigurará y no funcionará correctamente.

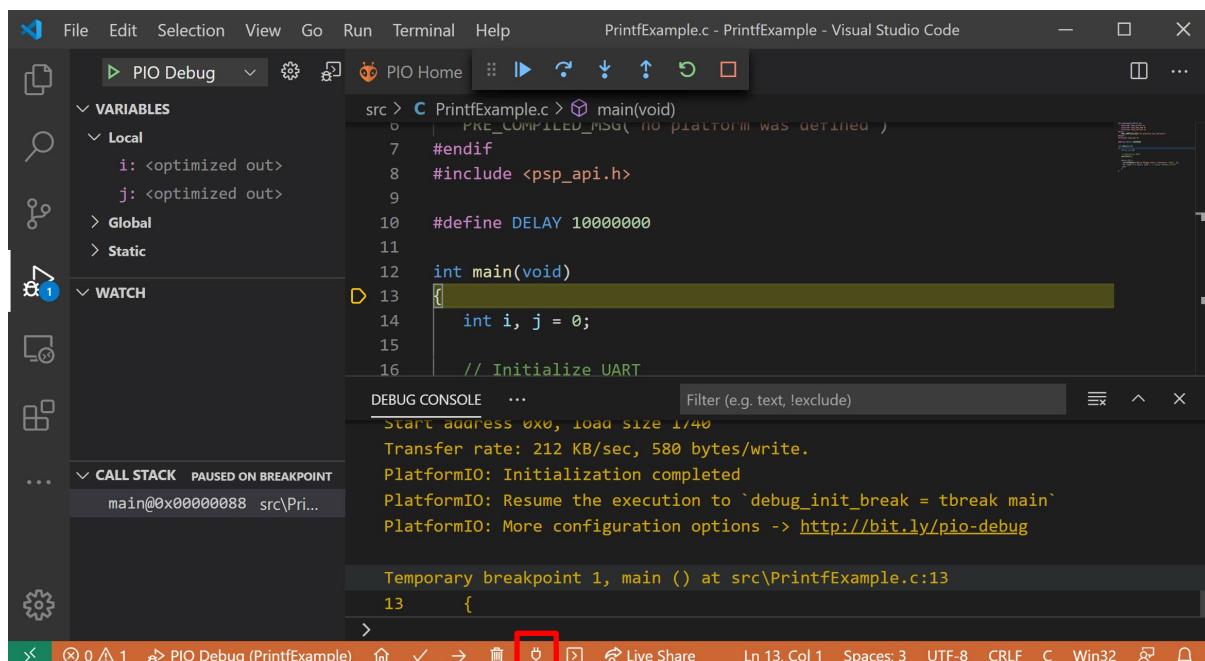
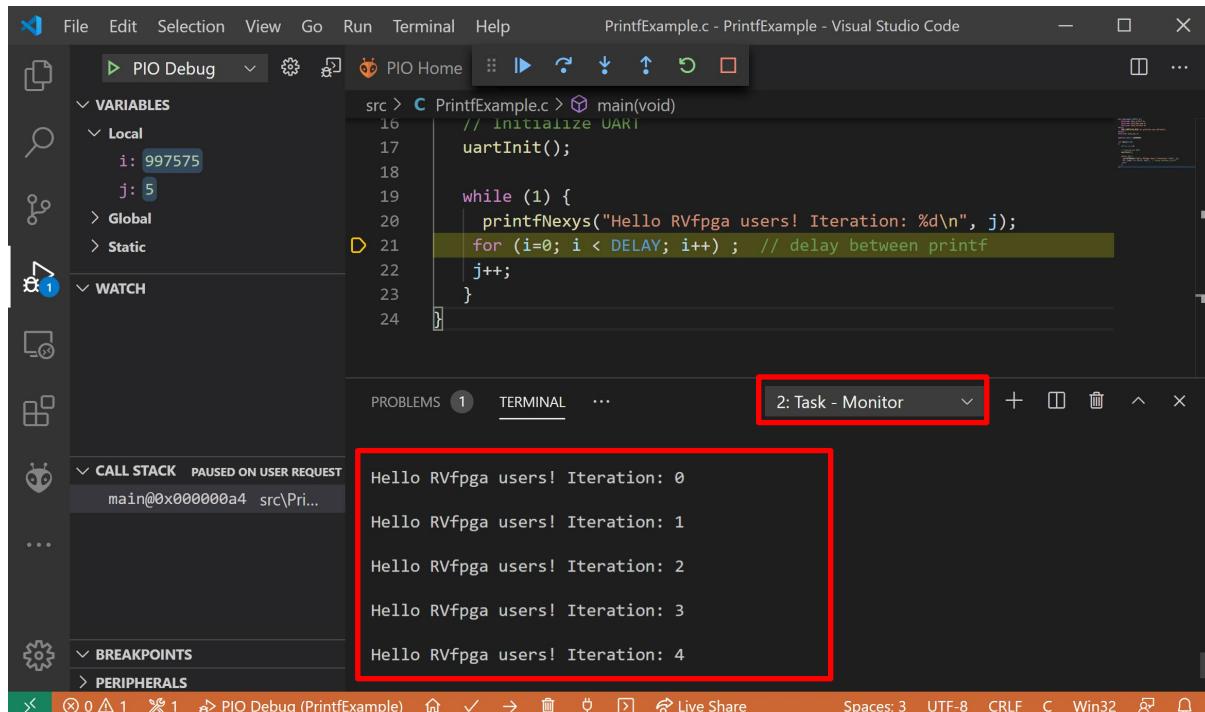


Figura 22. Inicio del monitor serie

Ahora ejecute el programa: .

Verá repetidamente en el monitor serie la cadena de caracteres que se imprime (Hello RVfpga users!) seguida del número de iteración, como se muestra en la figura 23.



```

src > C PrintfExample.c > main(void)
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 }

```

The terminal window shows the following output:

```

Hello RVfpga users! Iteration: 0
Hello RVfpga users! Iteration: 1
Hello RVfpga users! Iteration: 2
Hello RVfpga users! Iteration: 3
Hello RVfpga users! Iteration: 4

```

Figura 23. Salida de la función printfNexys en el monitor serie de PlatformIO

4. Ejercicios

A continuación tendrá la oportunidad de crear sus propios programas en C completando los siguientes ejercicios. Tenga en cuenta que si deja la placa Nexys A7 conectada a su computadora y encendida, no necesita volver a cargar RVfpgaNexys en la placa entre la ejecución de diferentes programas. Sin embargo, si apaga la placa Nexys A7, es necesario recargar RVfpgaNexys en la placa usando PlatformIO, como se describe en el paso 3 de la Sección 2.

Recuerde que puede imprimir cualquier variable usando la función BSP de Western Digital printfNexys (ver sección 3).

Recuerde también que puede ejecutar todos estos programas en simulación, tanto en Verilator como en Whisper.

Ejercicio 1. Escriba un programa en C que proyecte el valor de los interruptores en los LEDs. Ese valor se mantiene durante un tiempo tras el cual los LEDs se apagan, y se repite el proceso. El valor debe encenderse y apagarse lo suficientemente lento como para que una persona pueda ver el parpadeo. Guarde el programa como **FlashSwitchesToLEDs.c**.

Ejercicio 2. Escriba un programa en C que muestre el valor inverso de los interruptores en los LEDs. Por ejemplo, si los interruptores son (en binario): 010101010101, entonces los LEDs deberían mostrar: 1010101010101010; si los interruptores son:

1111000011110000, entonces en los LEDs debería aparecer: 0000111100001111; y así sucesivamente. Use **DisplayInverse.c** como nombre para el programa.

Ejercicio 3 (OPCIONAL). Escriba un programa en C que desplace un número creciente de LEDs encendidos hacia adelante y hacia atrás hasta que todos los LEDs estén encendidos. Entonces el patrón debe repetirse. Guarde el programa como **ScrolILEDs.c**.

El programa debería realizar lo siguiente:

1. Primero, un LED encendido debe desplazarse de derecha a izquierda y luego de izquierda a derecha.
2. A continuación dos LEDs encendidos deben desplazarse de derecha a izquierda y luego de izquierda a derecha.
3. Despues tres LEDs encendidos deben desplazarse de derecha a izquierda y luego de izquierda a derecha.
4. Y así sucesivamente, hasta que todos los LEDs se encienden.
5. A continuación el patrón debe repetirse.

Ejercicio 4. Escriba un programa en C que muestre el resultado de la suma (4 bits sin signo) de los 4 bits menos significativos de los interruptores con los 4 bits más significativos de los mismos. El resultado se debe mostrar en los 4 bits menos significativos (más a la derecha) de los LEDs. Guarde el programa como **4bitAdd.c**. El quinto bit de los LEDs debería encenderse cuando se produzca un desborde sin signo (es decir, cuando el acarreo de salida es 1).

(LOS SIGUIENTES EJERCICIOS DEL 5 AL 10 NO HAY QUE HACERLOS PARA AICC: PUEDE HACER ALGUNO DE FORMA OPCIONAL SI LE INTERESA)

Ejercicio 5. Escriba un programa en C que encuentre el *máximo común divisor* de dos números, a y b , según el algoritmo de Euclides. Los valores a y b deben ser variables definidas estáticamente en el programa. Guarde el programa como **GCD.c**. En el siguiente enlace se proporciona información adicional sobre el algoritmo de Euclides:

<https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm>. También puede simplemente buscar en Google "Algoritmo de Euclides".

Ejercicio 6. Escriba un programa en C que calcule los primeros 12 números de la serie de Fibonacci y almacene el resultado en un vector finito (es decir, un array), V , de longitud 12. La secuencia infinita de números de Fibonacci se define del siguiente modo:

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

Expresado en palabras, el número de Fibonacci correspondiente al elemento i es la suma de los dos números previos de la serie. La tabla 1 muestra los números de Fibonacci para $i = 0$ hasta 8.

Tabla 1. Serie Fibonacci

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

La dimensión del vector, N , debe ser definida en el programa como una constante. Guarde el programa como **Fibonacci.c**.

Ejercicio 7. Dado un vector de N elementos (es decir, un array), A , genere otro vector, B , de manera que B sólo contenga aquellos elementos de A que son números pares mayores que 0. El programa en C también debe contar el número de elementos de B e imprimir ese valor al final del programa. Por ejemplo: supongamos que $N=12$ y $A = [0,1,2,7,-8,4,5,12,11,-2,6,3]$, entonces B sería: $B = [2,4,12,6]$. Dado que B tiene cuatro elementos, al final del programa se debe imprimir lo siguiente:

Número de elementos de $B = 4$.

Utilice la función `printfNexys` para ello. Guarde el programa como **EvenPositiveNumbers.c**. Pruebe su programa para un vector A de 12 elementos.

Ejercicio 8. Dados dos vectores de N elementos (es decir, arrays), A y B , crear otro vector, C , definido como:

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

Escriba un programa en C que calcule el nuevo vector. Utilice arrays de 12 elementos en su programa. Guarde el programa como **AddVectors.c**.

Ejercicio 9. Implementar el algoritmo de ordenación burbuja (*bubble sort*) en C. Este algoritmo ordena los elementos de un vector en orden ascendente mediante el siguiente procedimiento:

1. Recorrer el vector repetidamente hasta terminar.
2. Intercambiar cualquier par de componentes adyacentes si $v(i) > v(i+1)$.
3. El algoritmo se detiene cuando cada par de componentes consecutivos está ordenado.

Use arrays de 12 elementos para probar su programa. Guarde el programa como **BubbleSort.c**.

Ejercicio 10. Escriba un programa en C que calcule el factorial de un número no negativo, n , mediante multiplicaciones iterativas. Aunque debe probar el correcto funcionamiento de su programa para múltiples valores de n , la entrega final del mismo debe ser para $n = 7$. El programa debe imprimir el valor del factorial(n) al final del programa. n debe ser una variable definida estáticamente dentro del programa. Guarde el programa como **Factorial.c**.