



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga Lab 2

C 프로그래밍

1. 소개

대부분의 컴퓨터 프로그램은 C 와 같은 고급 언어로 작성됩니다. 이 실습에서는 RVfpga 시스템에서 실행할 수 있는 플랫폼 IO 를 이용하여 C 프로젝트를 생성하는 방법을 보여 줍니다. 먼저 C 프로그램을 만들고 실행하는 방법에 대한 예제를 제공합니다. 그런 다음 C 프로그램을 직접 작성하는 예제를 설명합니다.

2. C Program for RVfpga

다음 단계에서 플랫폼 IO 을 사용하여 RVfpgaNexys 에서 C 프로그램을 만들고 실행합니다.
(Verilator 와 Whisper 를 사용하여 시뮬레이션 환경에서 이러한 프로그램을 수행할 수 있습니다):

1. RVfpga 프로젝트 생성
2. C 프로그램 작성
3. Nexys A7 FPGA 보드에 RVfpgaNexys 다운로드
4. C 프로그램 컴파일, 다운로드 및 실행

Step 1. RVfpga 프로젝트 생성

VScode 를 엽니다(RVfpga 시작하기 가이드 참조). 만약 플랫폼 IO 가 VScode 를 시작할 때 자동으로 열리지 않을 경우, 왼쪽 메뉴 리본의 Platform IO 아이콘을 클릭하십시오. 클릭한 다음 PIO Home → Open 을 클릭합니다(그림 1 참조).

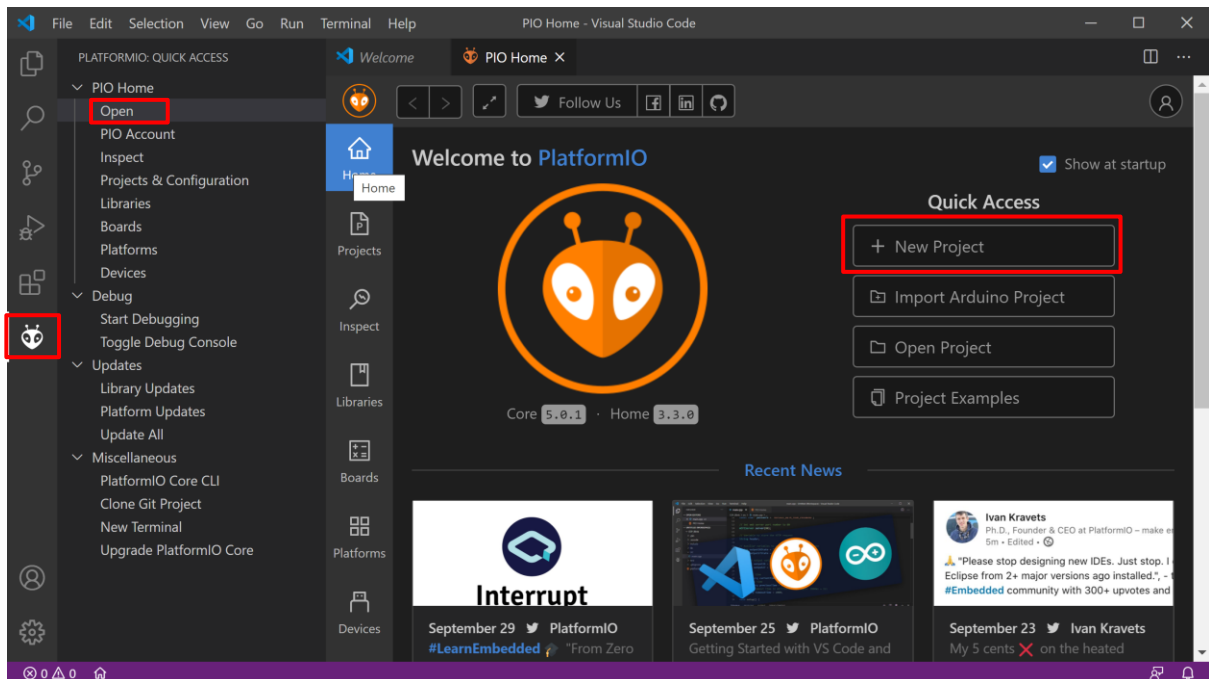


그림 1 오픈 플랫폼 IO 및 새 프로젝트 생성

이제 PIO Home 시작 창에서 새 프로젝트를 클릭합니다(그림 1 참조).

그림 2 와 같이 프로젝트 1 의 이름을 지정하고 보드를 RVfpga: Digilent Nexys A7 로 선택합니다.
(RVfpga 입력을 시작하면 보드가 옵션으로 나타납니다).

기본 프레임워크를 WD-firmware 로 유지합니다. WD-firmware 는 Western Digital 의 펌웨어로, Freedom-E SDK gcc 와 gdb 는 물론, 이러한 labs 에서 사용하는 PSP 와 BSP(프로세서 지원 패키지 및 보드 지원 패키지)도 포함하고 있습니다. 기본 위치 사용을 해제하고 프로젝트를 배치합니다:

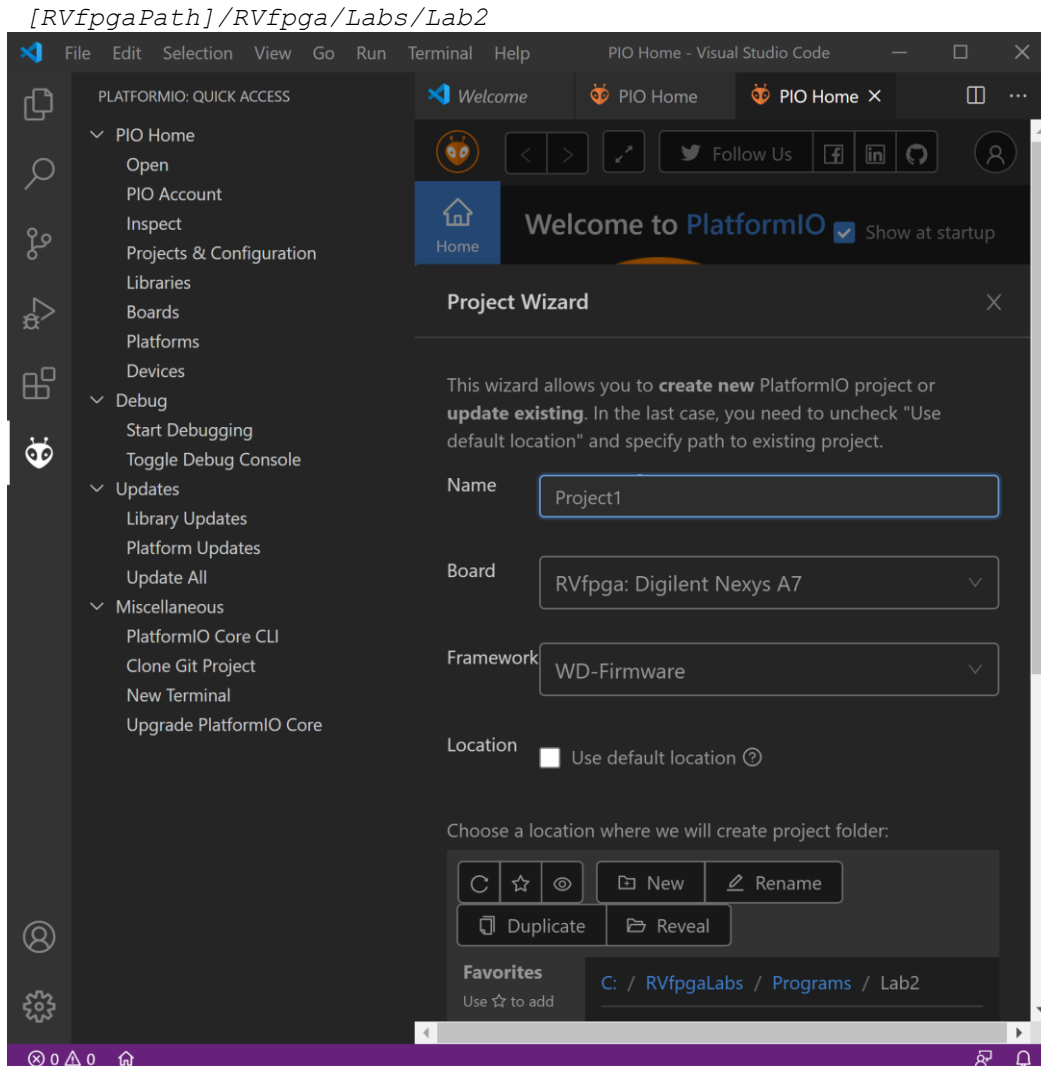


그림 2 프로젝트 이름 지정 및 보드 / 프로젝트 폴더 선택

그런 다음 창 하단에 있는 마침을 클릭합니다(그림 3 참조).

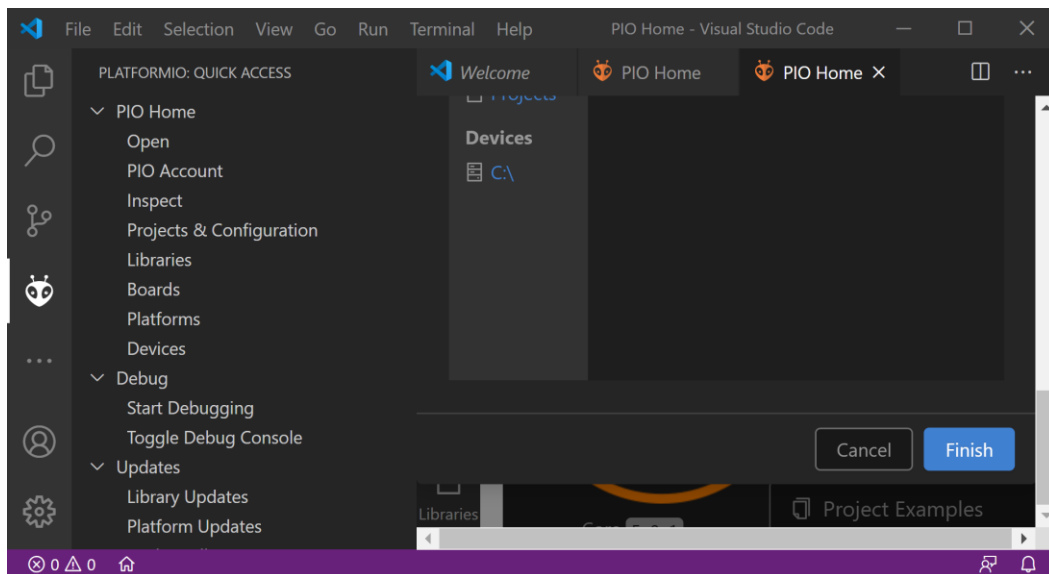


그림 3 프로젝트 생성 완료

왼쪽 탐색기 창에서 (화면 창 확대가 필요) Platformio.ini 를 두 번 클릭하여 열어야 합니다(그림 4 참조). 이것이 플랫폼 IO 초기화 파일입니다.

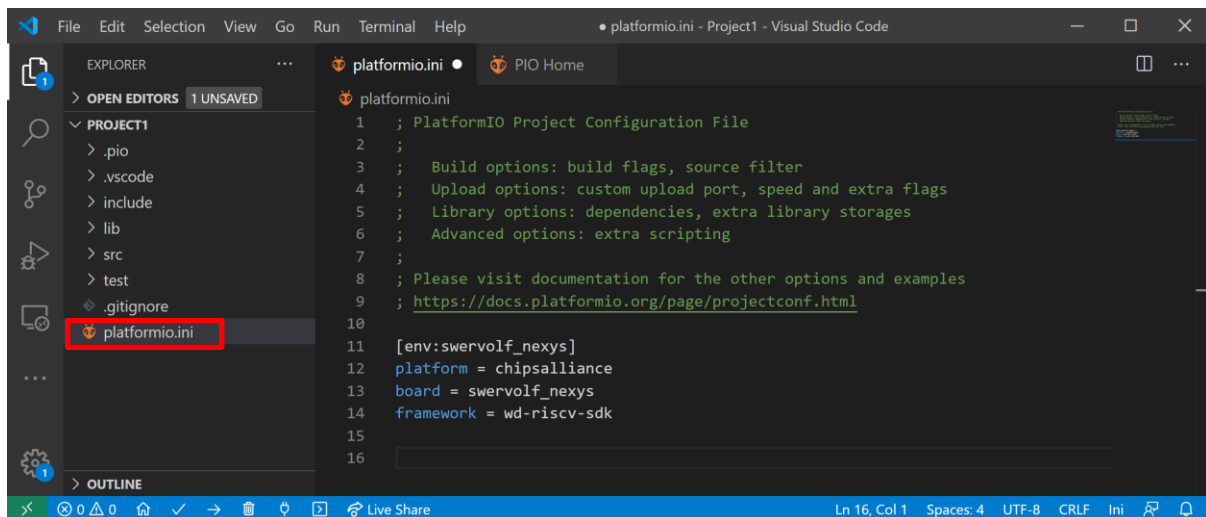


그림 4 플랫폼 IO 초기화 파일: Platformio.ini

그림 5 와 같이 Platformio.ini 파일에 다음 라인을 추가합니다:

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

(*[RVfpgaPath]*) 는 사용자 컴퓨터의 폴더 위치로 바뀌어야 합니다.) 이 라인은 PlatformIO 가 FPGA 에 로드할 비트스트림 파일의 위치를 나타냅니다. 위 경로는 Lab 1 에서 사용자가 생성한 비트스트림의 위치입니다. (Lab 1 을 완료하지 않은 경우 배포한 시작 가이드, *[RVfpgaPath]/RVfpga/src/rvfpganexys.bit* 에 위치, RVfpganexys 비트스트림을 사용할 수 있습니다) Ctrl-s 를 눌러 Platformio.ini 파일을 저장합니다.

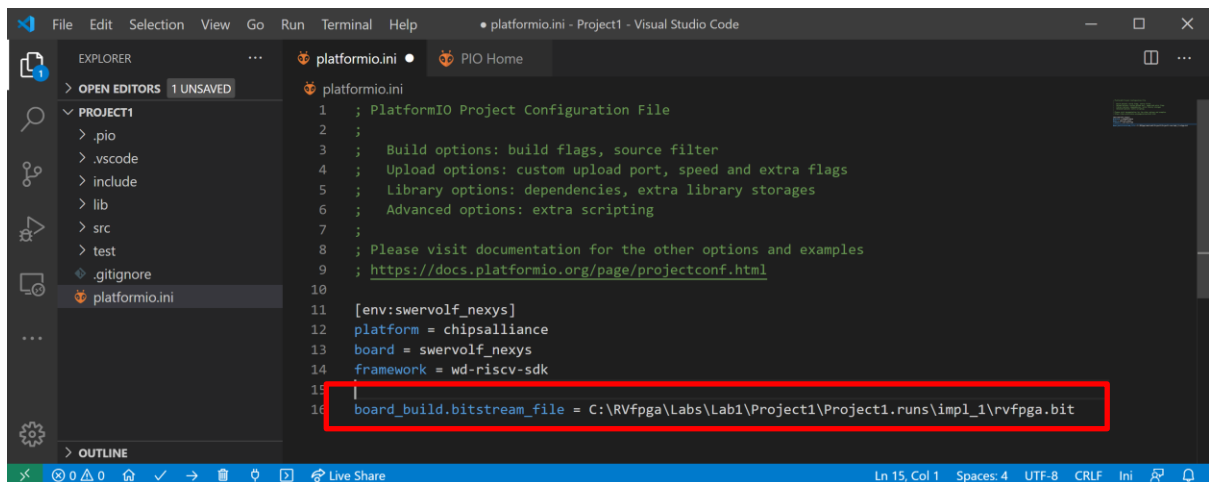


그림 5. RVfpgaNexys 비트스트림 파일 위치 추가(rvfpganexys.bit)

시작 안내서에 사용된 예제에는 좀더 구체적인 *platformio.ini* 파일이 사용되었습니다. 사용자가 명령어나 기능을 추가하는 경우에는 (예: Verilator 시뮬레이터 경로, 직렬 콘솔 구성, 콧속말 디버그 도구 등) 해당 예제의 *platformio.ini* 를 사용할 수 있습니다.

Step 2. C 프로그램 작성

이제 C 프로그램을 작성합니다. 파일 → 새 파일을 클릭합니다 (그림 6 참조).

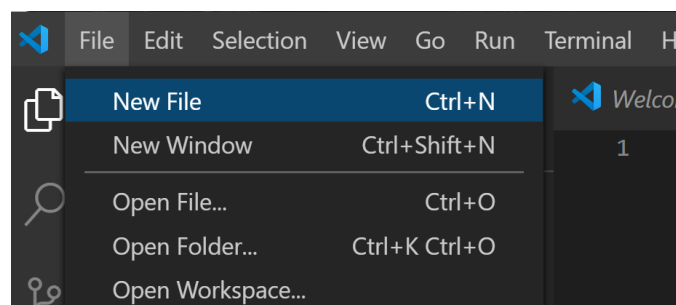


그림 6. 프로젝트에 파일 추가

빈 창이 열립니다. 다음 C 프로그램을 해당 창에 입력(또는 복사/붙여넣기) 합니다(그림 7 참조). 이 프로그램은 LED의 스위치 값을 표시합니다.

```
// 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);
}

```

이 프로그램은 사용자의 편의를 위해 다음 파일에서도 사용할 수 있습니다.

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

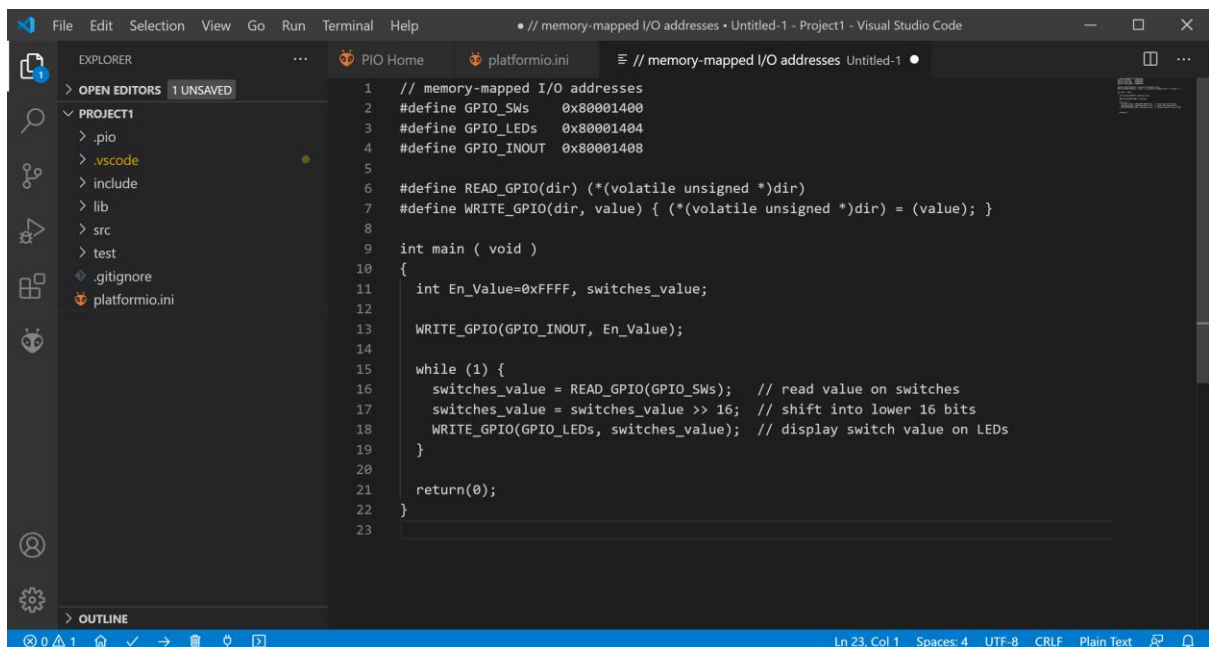


그림 7. C 프로그램 입력

창에 프로그램을 입력한 후 Ctrl-s 를 눌러 파일을 저장합니다. 이름을 DisplaySwitchs.c 로 지정하고 Project1 디렉토리의 src 폴더에 저장합니다(그림 8 참조).

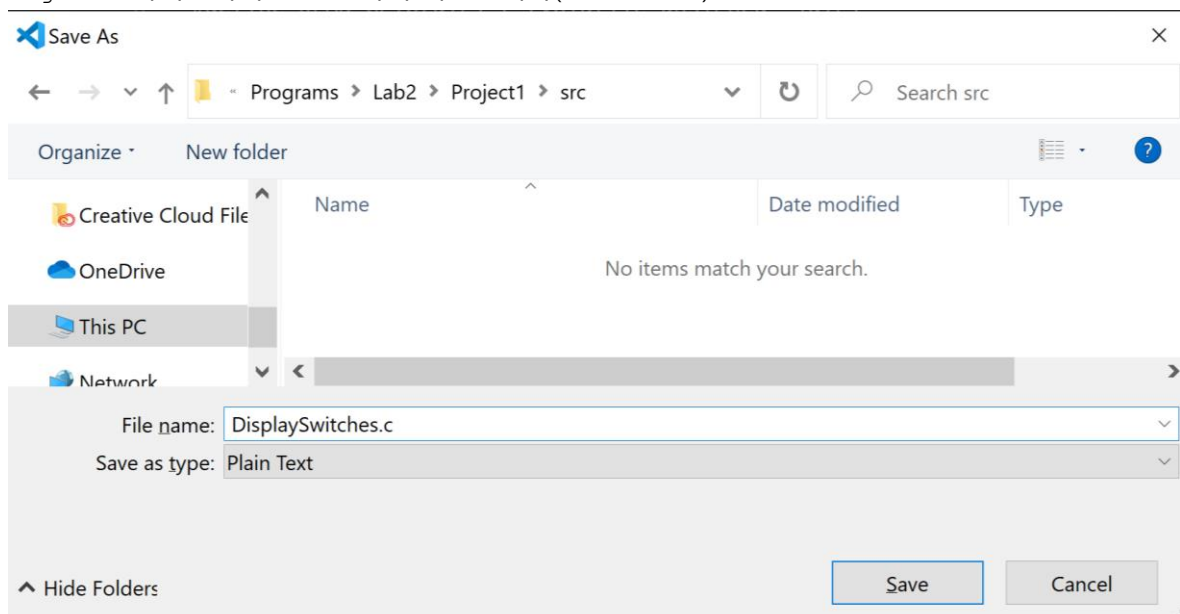


그림 8. DisplaySwitchs.c 로 파일 저장

이 프로그램은 먼저 다음 라인을 사용하여 Nexys A7 FPGA 보드의 LED 및 스위치에 연결된 메모리 맵 I/O 레지스터의 주소를 정의합니다.

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

스위치의 값은 주소 0x80001400 에 매핑된 레지스터를 읽으면 확인되며, 주소 0x80001404 에 매핑된 레지스터를 기록하면 LED 에 값이 표시됩니다. 스위치 값은 레지스터의 상반부에 있고 LED 는 하반부에 있습니다.

GPIO_INOUT 레지스터는 GPIO (범용 I / O)의 비트가 입력인지 출력인지를 정의합니다. 최하위 16 GPIO 핀 (15 : 0)은 Nexys A7 보드의 16 개 LED 에 연결됩니다. 최상위 16 개의 GPIO 핀 (31:16)은 16 개의 온 보드 스위치용 입니다. 0 은 입력을 나타내고 1 은 출력을 나타냅니다. 따라서 GPIO_INOUT 레지스터는 0xFFFF 로 기록되므로 스위치는 RVfpgaNexys 에 대한 입력이고 LED 는 RVfpgaNexys 에 의해 구동되는 출력입니다.

그림 9 는 Nexys A7 FPGA 보드에 있는 LED 와 스위치의 물리적 위치와 USB 커넥터, ON 스위치, 푸시 버튼 및 7-세그먼트 디스플레이를 보여줍니다.

Lab 6 에서는 GPIO 기능과 RVfpgaNexys GPIO 하드웨어에 대해 자세히 설명합니다. 또한 이후 실습 (실습 6-10)에서 푸시 버튼 및 7 세그먼트 디스플레이와 같은 다른 보드 주변 장치를 사용하는 방법에 대해서도 논의합니다.

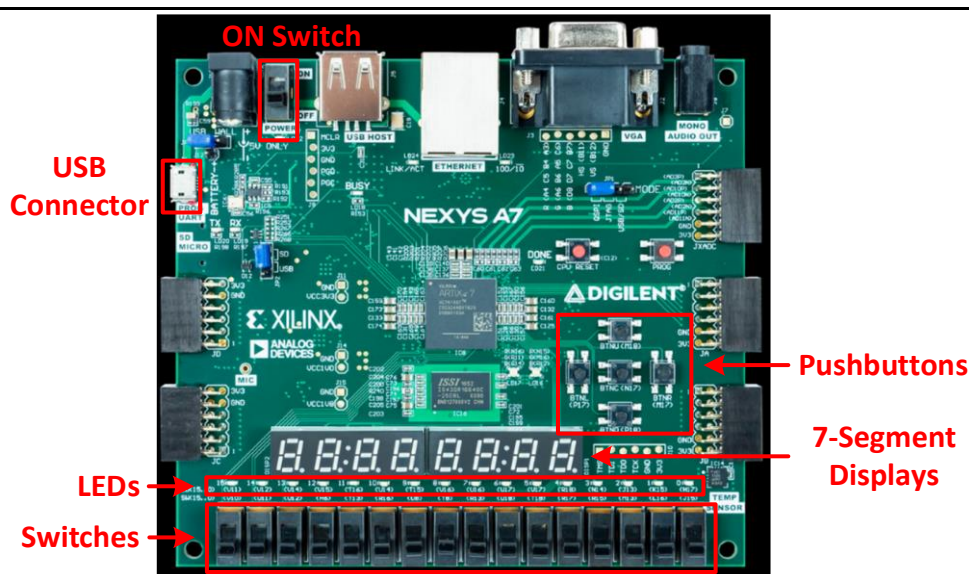


그림 9. Digilent 의 Nexys A7 FPGA 보드의 I/O 인터페이스

(그림 출처: <https://reference.digilentinc.com/>)

LED 및 스위치의 메모리 매핑 I/O 주소를 정의한 후 프로그램은 다음을 수행합니다.

1. GPIO_INOUT 레지스터의 상반부를 0 으로 설정하여 최상위 16 개의 GPIO 핀(스위치에 연결된)을 입력으로 정의하고, 다음 코드를 실행하여 GPIO_INOUT 레지스터의 하반부를 1 로 설정하여 최하위 16 개의 GPIO 핀(LED 에 연결된)을 출력으로 정의합니다.

```
int En_Value=0xFFFF;

WRITE_GPIO(GPIO_INOUT, En_Value);
```

2. 아래의 코드를 실행하여 스위치의 값을 반복적으로 읽고 해당 값을 LED 에 기록합니다. 스위치 값이 메모리 매핑 I/O 레지스터의 상반부에서 읽혀지기 때문에 LED 에 물리적으로 연결된 메모리 매핑 I/O 레지스터에 쓰기 전에 값이 16 비트 만큼 오른쪽으로 이동해야 합니다.

```
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 매크로와 WRITE_GPIO 매크로가 각각 지정된 메모리 매핑 I/O 주소에서 값을 읽거나 씁니다.

Step 3. Nexys A7 FPGA 보드에 RVfpgaNexys 다운로드

이제 Nexys A7 FPGA 보드에 RVfpgaNexys 를 다운로드합니다. 왼쪽 메뉴 리본을 클릭 후 PlatformIO 아이콘을 선택한 다음, 프로젝트 작업 → env:swerbolf_develys → 플랫폼을 확장하고 그림 10 과 같이 Upload Bitstream 을 클릭합니다.

주의사항: Windows 시스템을 사용하고 있지만 Nexys A7 FPGA 보드 드라이버를 아직 교체하지 않은 경우 시작 가이드의 부록 B 에 있는 지침을 따르십시오.

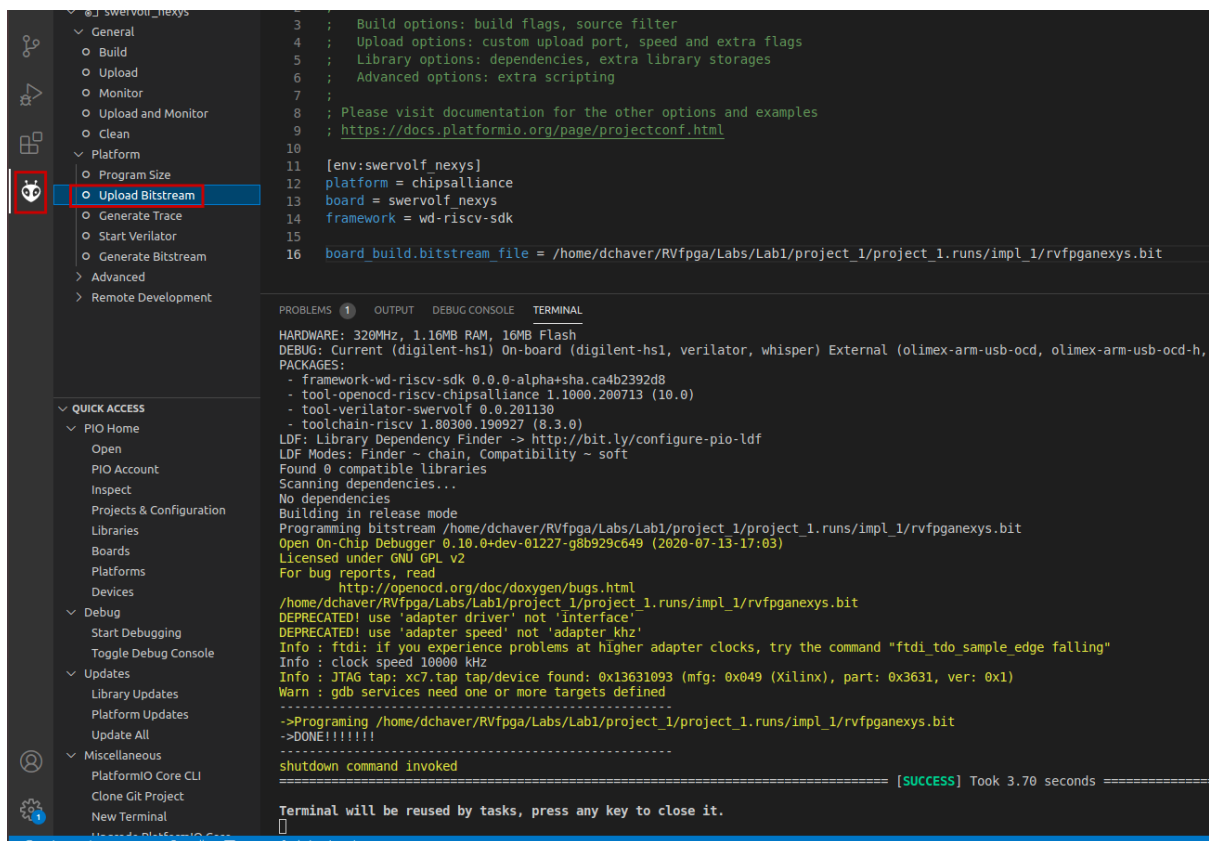



그림 10. PlatformIO 을 사용하여 RVfpgaNexys 를 Nexys A7 FPGA 보드로 업로드

그림 11 과 같이 PlatformIO 터미널에서 RVfpgaNexys 를 다운로드할 수도 있습니다.

Platform IO 클릭: PlatformIO 창 하단의 새 터미널 버튼() , PlatformIO 터미널로 아래 내용을 입력 또는 복사합니다.

```
pio run -t program_fpga
```

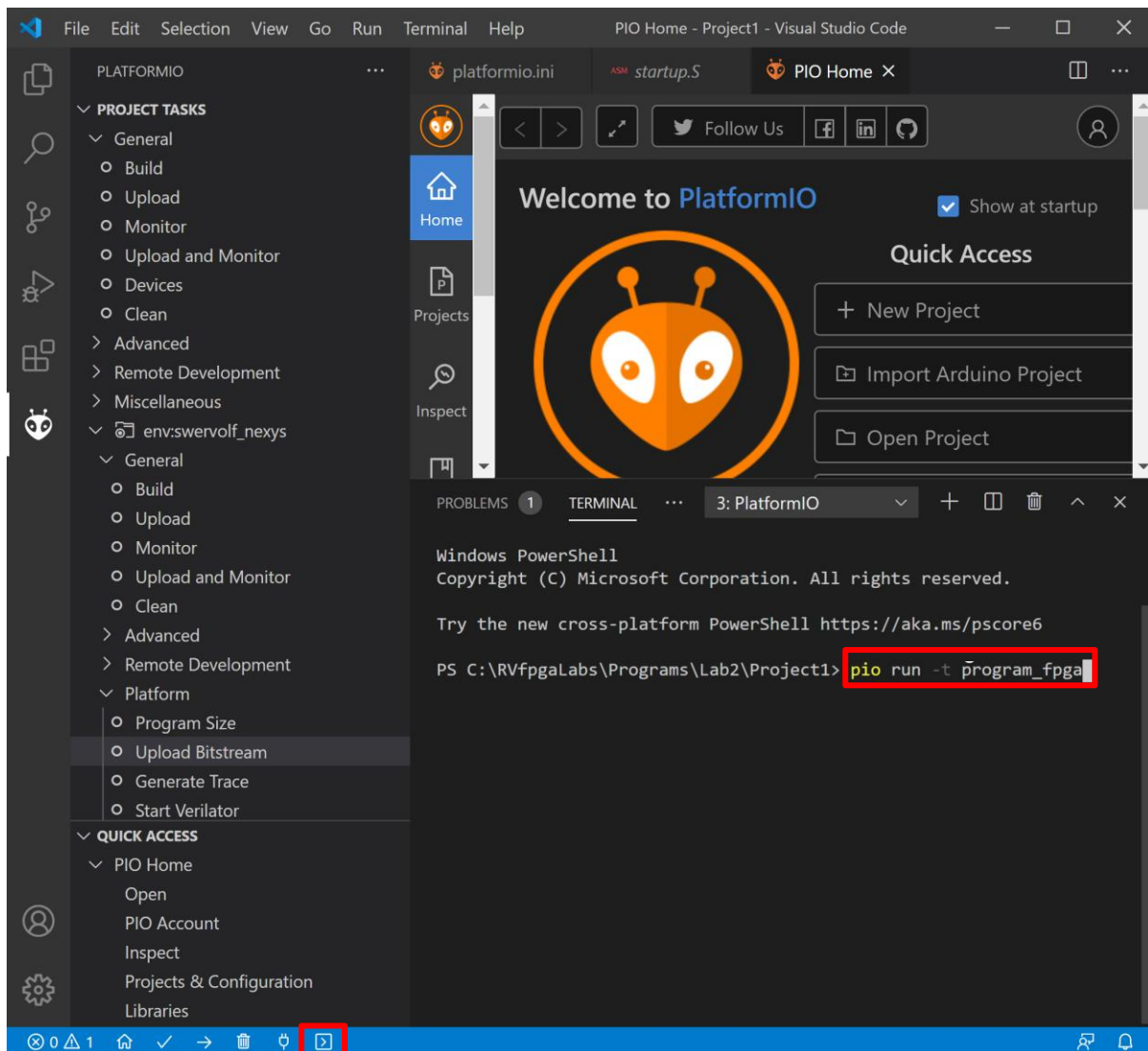


그림 11. PlatformIO 단자를 사용하여 RVfpgaNexys 를 Nexys A7 FPGA 보드에 업로드

Step 4. C 프로그램 컴파일, 다운로드 및 실행

이제 RVfpgaNexys 가 보드에서 실행되므로 프로그램을 컴파일하여 RVfpgaNexys 에 다운로드한 후 실행/디버깅할 수 있습니다. VSCode 가 아직 열려 있지 않으면 열어 주세요. 마지막 프로젝트인 프로젝트 1 이 자동으로 열립니다. 그렇지 않은 경우 PlatformIO 확장명이 열려 있는지 확인한 후 파일 → 폴더 열기를 클릭하고 이 실습에서 이전에 생성한 프로젝트 1 을 선택(열지는 않음)합니다.

왼쪽 메뉴 리본에서 실행 버튼을 클릭합니다(그림 12 참조).

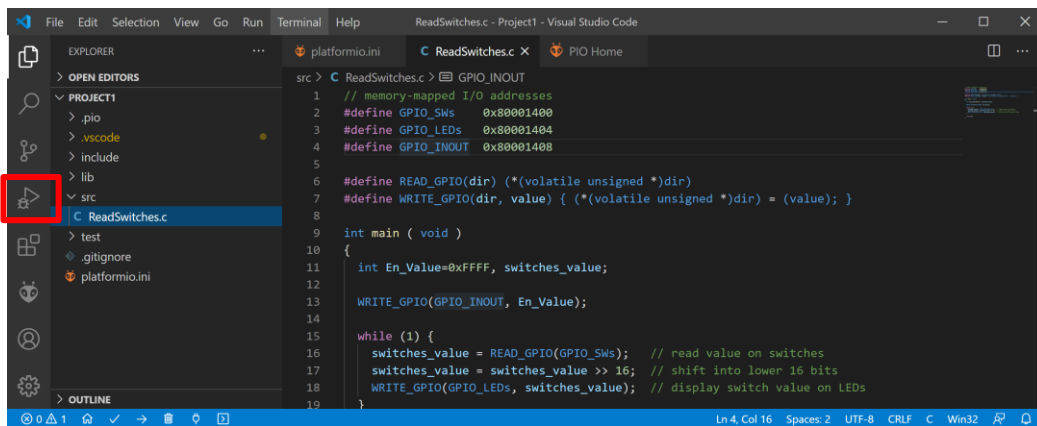


그림 12. RVfpgaNexys 에서 프로그램 실행

이제 Start Debugging 버튼을 클릭합니다(그림 13 참조).

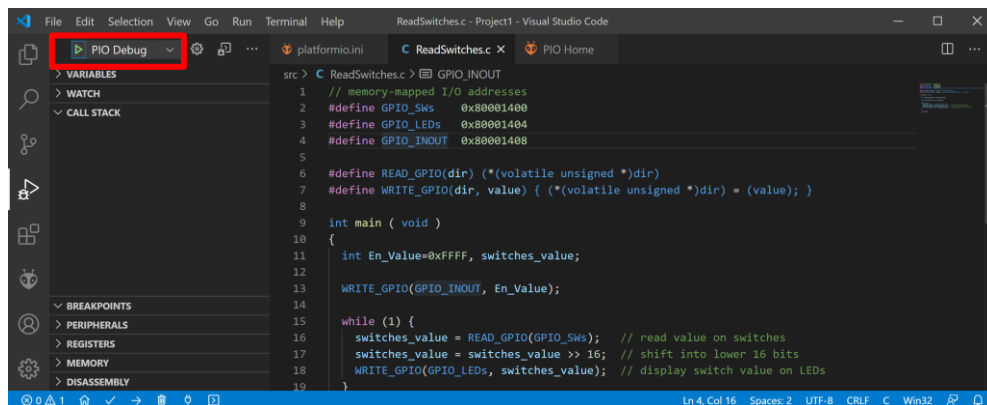


그림 13. 프로그램 실행 및 디버깅 시작

이 프로그램은 컴파일된 다음 Nexys A7 보드의 FPGA 에서 실행 중인 RVfpgaNexys 에 다운로드 됩니다(단말기에는 누락된 sys/cdefs.h 파일에 대한 문구가 표시될 수 있지만 프로그램은 여전히 올바르게 작동합니다). 이제 프로그램 실행 및 디버깅을 시작할 수 있습니다(그림 14 참조).

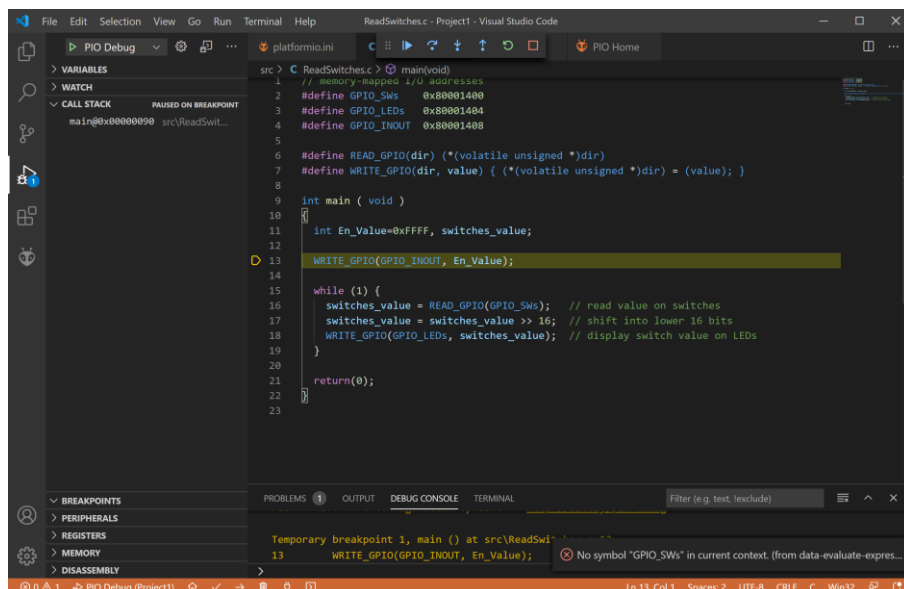
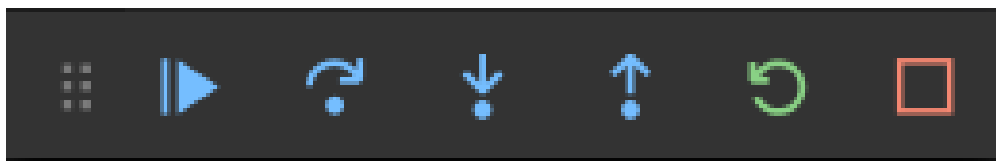


그림 14. 프로그램 RVfpgaNexys 에서 실행

RVfpga Getting Started Guide 에 설명된 대로 디버깅 세션을 제어하려면 편집기 상단 근처에 나타나는 디버깅 도구 모음을 사용하십시오(그림 15 참조). 다음은 옵션 내용입니다.

1. **Continue** 다음 중단점까지 프로그램을 계속 실행합니다.
2. **Breakpoints** 편집기에서 줄 번호의 왼쪽을 클릭하여 추가할 수 있습니다.
3. **Step Over** 현재 라인을 실행한 다음 중지합니다.
4. **Step Into** 현재 회선을 실행하며, 현재 회선에 함수 호출이 포함된 경우 해당 기능으로 진입 후 중지합니다.
5. **Step Out** 현재 사용 중인 기능의 모든 코드를 실행한 다음 해당 기능이 돌아오면 중지합니다.
6. **Restart** 프로그램 시작부터 디버깅 세션을 다시 시작합니다..
7. **Stop** 디버깅 세션을 중지하고 일반 편집 모드로 돌아갑니다. 중지 버튼을 누르면 RVfpgaNexys 에서 **프로그램이 계속 실행되지만** 디버깅 세션은 종료됩니다.
8. **Pause** 실행을 일시 중지합니다. 프로그램이 실행 중이면 Continue 단추가 Pause 단추로 바뀝니다.




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

그림 15. 디버깅 도구

왼쪽 사이드바에서 디버거 옵션을 볼 수 있습니다. 다음 옵션을 사용할 수 있습니다.

- **Variables(변수):** 프로그램에 있는 로컬, global 및 static 변수 값과 함께 해당 변수가 나열됩니다.
- **Call Stack(스택 호출):** 실행 중인 현재 기능, 호출 기능(있을 경우) 및 메모리에 있는 현재 명령의 위치를 표시합니다.
- **Breakpoints(중단점):** 설정된 중단점을 표시하고 해당 라인 번호를 강조 표시합니다. 중단점은 이 섹션에서 관리할 수 있습니다. 확인란을 전환하여 중단점을 제거하지 않고 일시적으로 비활성화할 수도 있습니다.
- **Peripherals (주변 장치):** 장치의 메모리 매핑된 주변 장치의 레지스터 상태를 표시합니다.
- **Registers (레지스터):** 프로세서의 각 레지스터에 존재하는 현재 값을 나열합니다.
- **Memory(메모리):** 메모리의 특정 주소의 내용을 표시합니다.
- **Disassembly (디스어셈블리):** 특정 기능에 대한 어셈블리 코드를 표시합니다. C 와 같은 상위 레벨의 코드에 대해서는 지침을 하나씩 디버깅하기 위해 어셈블리를 볼 수 있습니다.

예를 들어, 스위치 값이 LED 에 기록되기 직전에 중단점을 설정하려면 18 행의 왼쪽을 클릭하십시오, 그림 16 과 같이. 이제 계속 버튼을 클릭하거나  F5 키를 눌러 프로그램을 실행합니다. 중단점까지 프로그램이 계속됩니다. (중단점을 제거하려면 줄 번호의 왼쪽에서 기존 중단점을 클릭하십시오.)

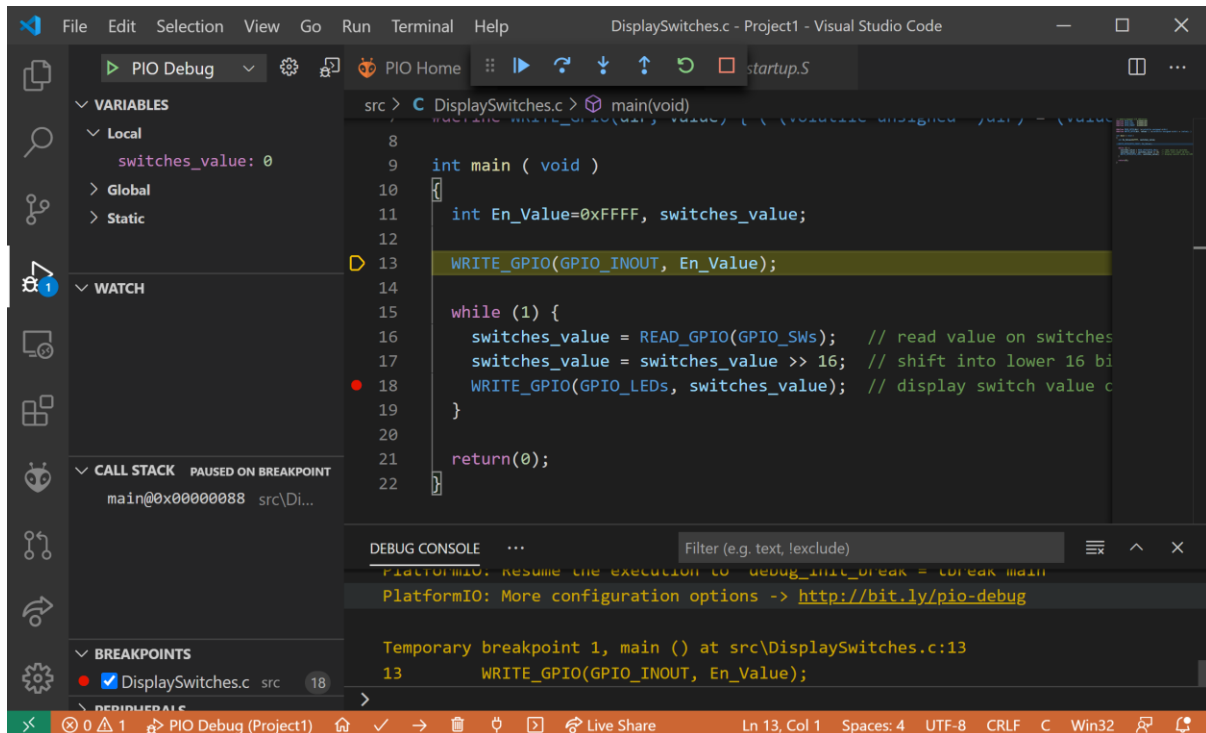


그림 16. LED 에 값을 표시하기 전에 중단점 설정

중단점에 도달하면 그림 17 과 같이 왼쪽 창에서 Variables 섹션을 확장하고 variable switches_value 의 값을 확인합니다. 이 경우 스위치의 값은 1029 = 0x405(이진수, 0000_0100_0000_0101)이며, 이는 다음 패턴에 해당합니다.

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

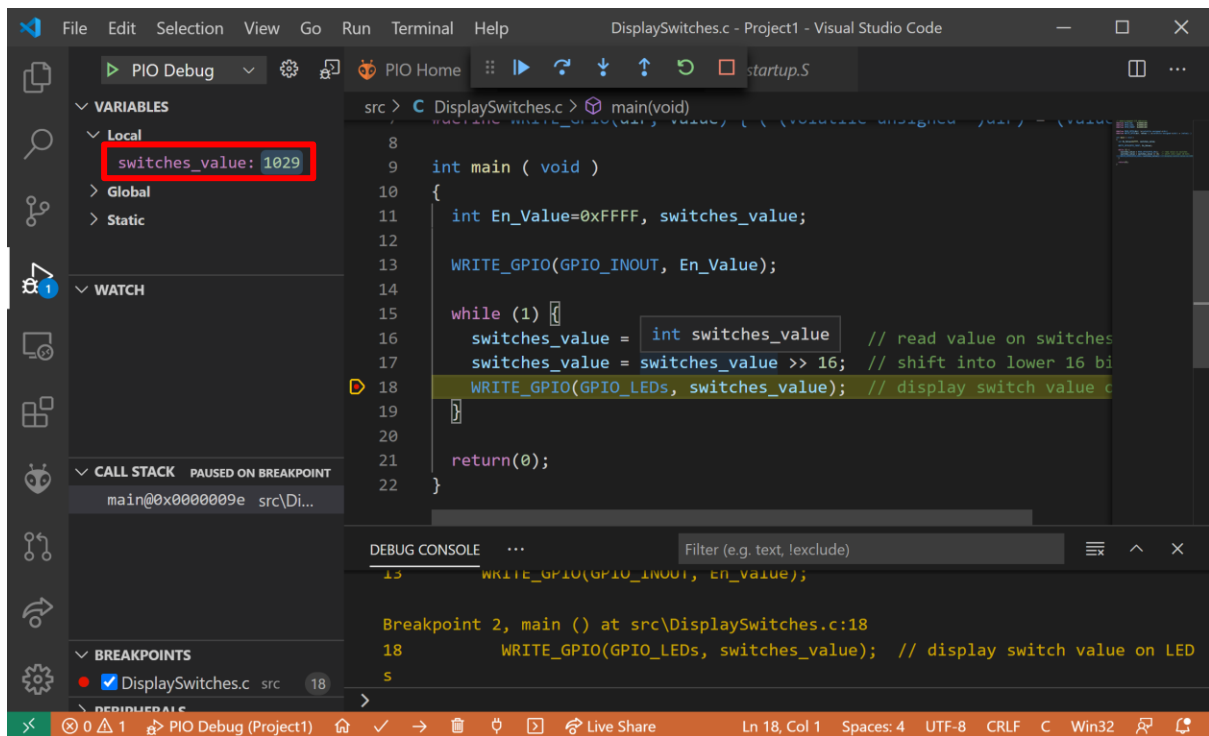


그림 17. 변수 값 보기

C 프로그램에서 생성된 RISC-V 어셈블리 코드를 볼 수도 있습니다. 그림 18 과 같이 DISAssembly → Switch to Assembly(어셈블리로 전환)를 클릭하여 이 작업을 수행합니다.

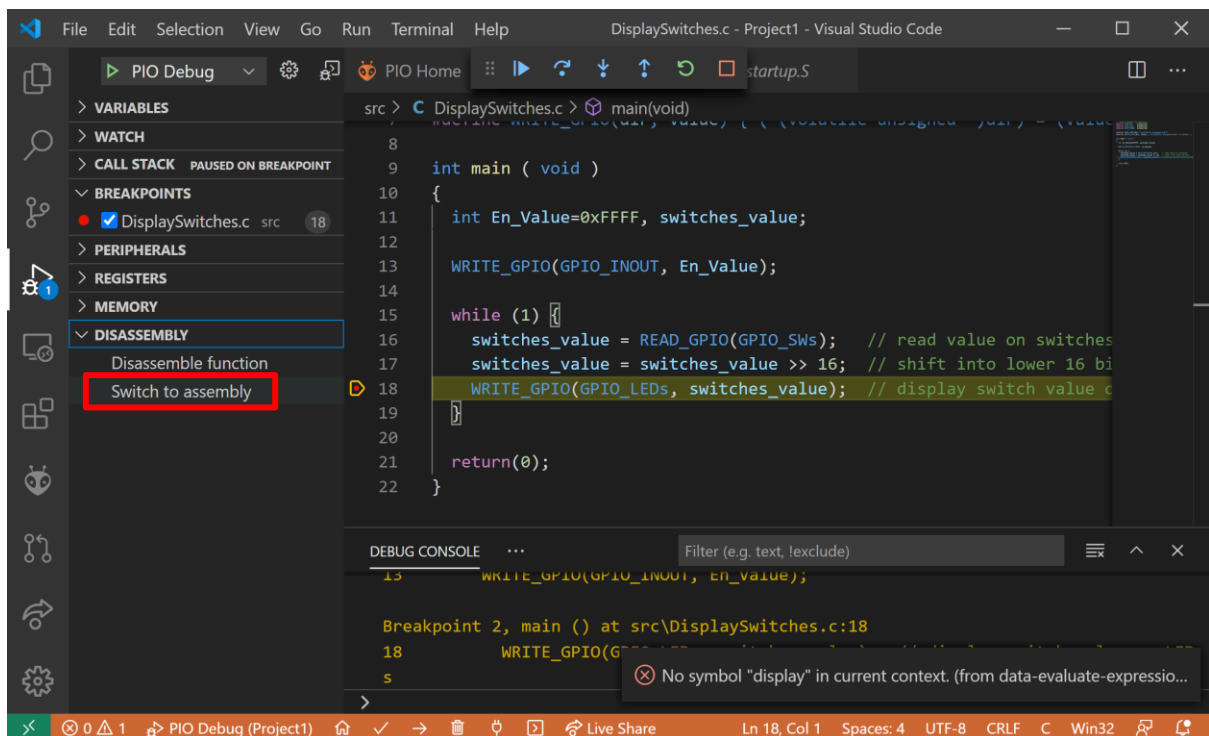


그림 18. RISC-V 어셈블리 코드 보기

이제 그림 19 와 같이 RISC-V 어셈블리가 보기 창에 표시됩니다. 어셈블리는 명령의 메모리 주소, 기계 코드 및 어셈블리 코드를 표시합니다. 보시는 것처럼 C 코드는 혼합된 압축(16 비트 명령어)과 32 비트 명령어로 컴파일 됩니다. 아래는 주석 처리된 어셈블리 코드입니다.

```
# 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
```

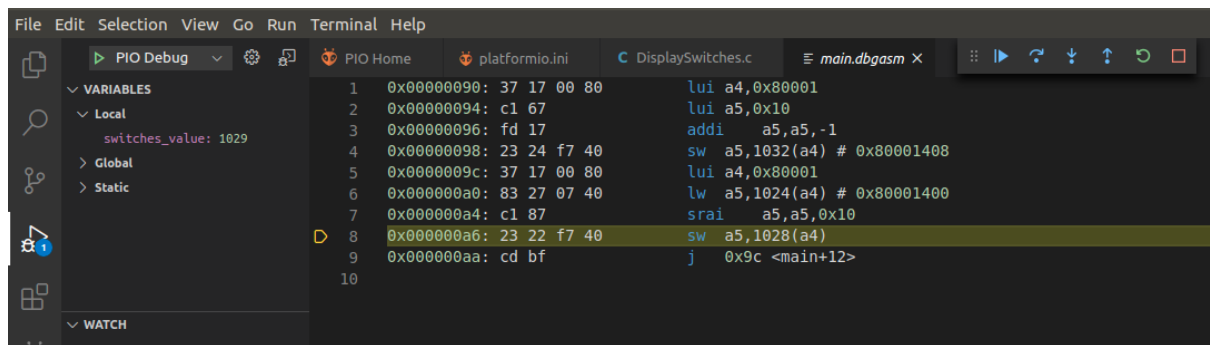



그림 19. RISC-V 어셈블리 코드 보기

클릭 DISASSEMBLY → C 코드를 다시 확인합니다.

프로그램 실행/디버거가 완료되면  Stop(또는 Shift – F5) 버튼을 눌러

디버깅 세션을 중지합니다. 맨 왼쪽 사이드바 맨 위에서  를 클릭하여 탐색기 창으로 돌아갑니다. 프로그램은 RVfpgaNexys 에서 계속 실행되며 디버깅 세션만 종료됩니다. 상단 메뉴 모음에서 File(파일) → Close Folder(폴더 닫기)를 클릭하여 프로젝트를 닫습니다.

3. printf 및 serial monitor 사용

프로그램에서 인쇄 문장을 사용하는 것은 프로그램 진행 상황을 추적하거나 사용자에게 계산 결과 등의 피드백을 제공하는 유용한 방법이다. RVfpga 시작하기 가이드의 섹션 6.F 의 (*HelloWorld_C-Lang*)에서 호출합니다. 일반적인 C 프로그램의 printf 기능과 유사한 printfNexys 기능을 사용할 수 있습니다. 이렇게 하려면 지정된 프로세서 및 보드에 공통 기능을 제공하는 Western Digital 의 PSP 및 BSP(프로세서 지원 패키지 및 보드 지원 패키지)를 사용해야 합니다. (이 경우 SwRV EH1 코어 및 Nexys A7 FPGA 보드).

[RVfpgaPath]/RVfpga/Labs/Lab2 폴더에 PrintfExample 이라는 PlatformIO 프로젝트를 만듭니다. 다음 프로그램을 해당 프로젝트에 추가합니다(그림 20 참조). 프로그램 파일의 이름을 PrintfExample.c 로 지정합니다.

이 프로그램은 또한 당신의 편의를 위해 여기서 이용할 수 있습니다:

[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++;
    }
}
```

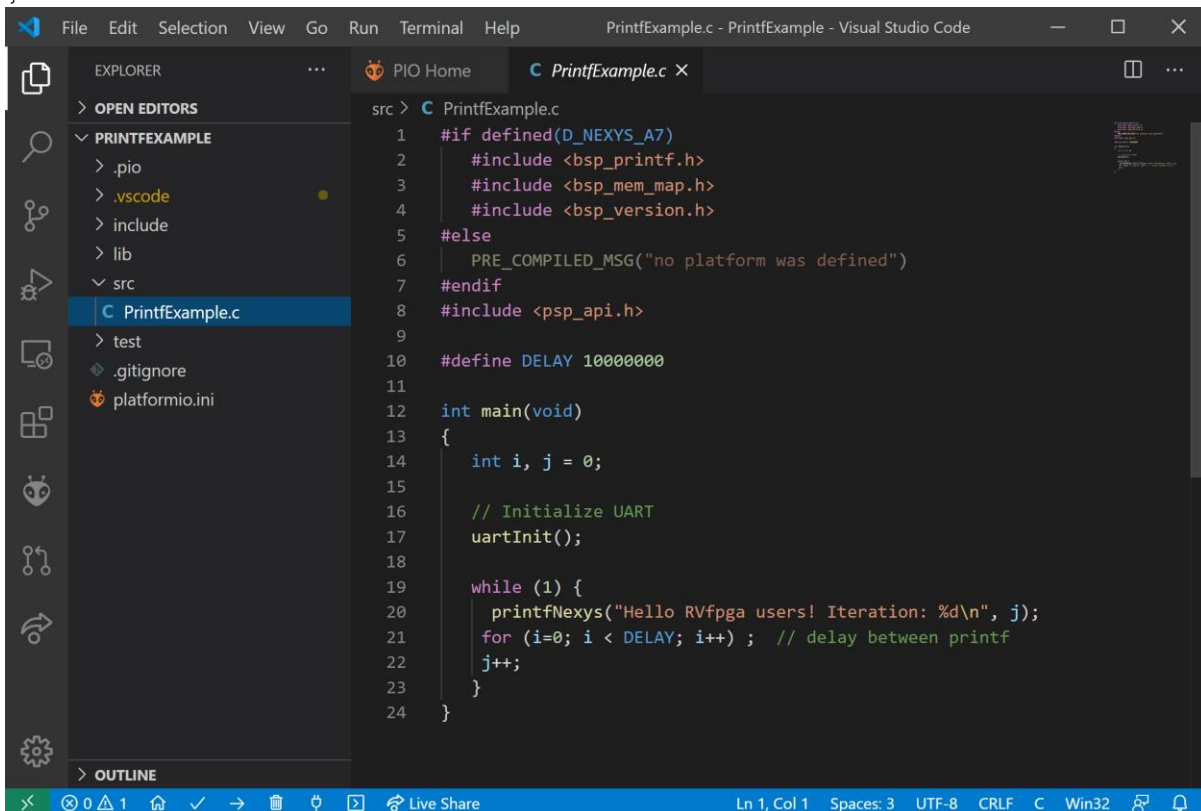


그림 1. PrintfExample.c

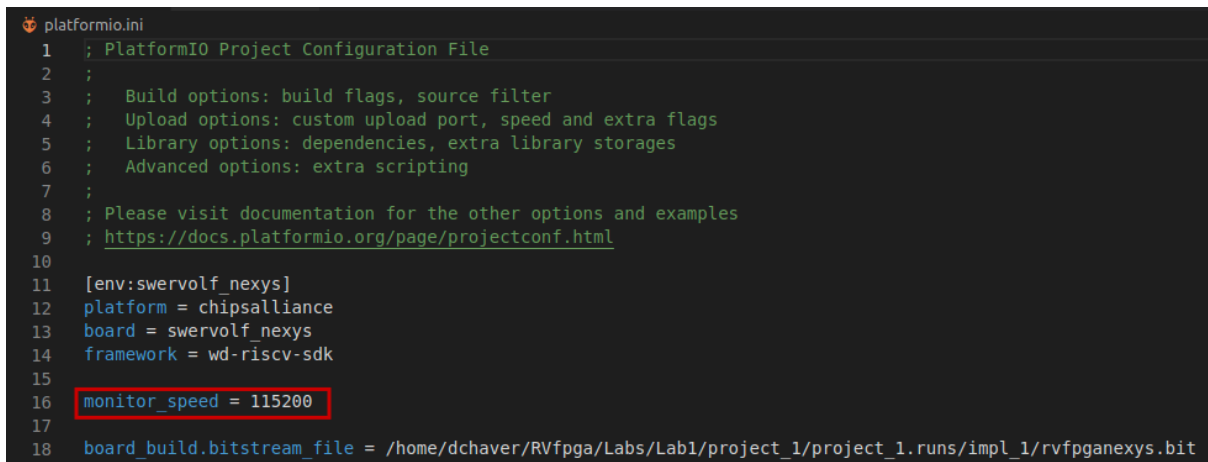
라인 1-8(그림 20 참조)은 `printfNexys` 기능을 사용하는 데 필요한 include 파일들입니다. 그림 20에 나오는 웨스턴 디지털의 BSP/PSP 라인 17에 `uartInit` 기능을 제공합니다. 이 라인은 RVfpgaNexys(Nexys A7 보드에서 실행 중)와 시리얼 모니터 간 통신을 사용하여 UART 연결을 초기화하는데 필요합니다. 마지막으로 while loop 문은 라인 20에서 `printfNexys` 기능을 사용하여 직렬 모니터에 반복적으로 기록한 다음 지연(delay)시킵니다(라인 21).

`printfNexys` 기능과 UART를 사용하려면 UART의 속도를 포함하도록 `platformio.ini` 파일을 수정해야 합니다. 그림 21과 같이 `Platformio.ini` 파일에 다음 줄을 추가합니다.

```
monitor_speed = 115200
```

RVfpgaNexys는 UART가 115200 baud (symbols/sec)로 통신할 것으로 예상하므로 이 속도는 그림 21, 라인 16에 표시된 것과 같이 `Platformio.ini`에 설정되어야 한다.

`board_build.bitstream_file` =...(그림 21, 라인 18 참조)를 사용하여 RVfpgaNexys 비트 파일의 위치를 추가해야 합니다.




```
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
```

그림 21. UART 속도 설정

LINUX: Linux를 사용하는 경우 직렬 모니터를 사용하기 전에 시작 안내서의 섹션 6.F에 설명된 대로 dialout, tty 및 uucp 그룹에 사용자를 추가하여 시스템을 준비해야 합니다. GSG에서 이 프로세스를 수행했다면 모든 것이 작동합니다. 그렇지 않으면 지금 실행하십시오.

이제 비트 파일을 업로드하고(섹션 2에 설명된 대로) 프로그램을 실행/디버그 합니다.

프로그램 실행이 시작된 후(프로그램 실행이 시작된 후에만) PlatformIO 창 하단에 있는 직렬 모니터 버튼  을 클릭합니다(그림 22 참조).

경고: 프로그램이 실행되기 전에 시리얼 모니터를 열면(첫 번째 – 임시 – 중단점에 도달하면) UART가 스캔블되고 올바르게 작동하지 않습니다.

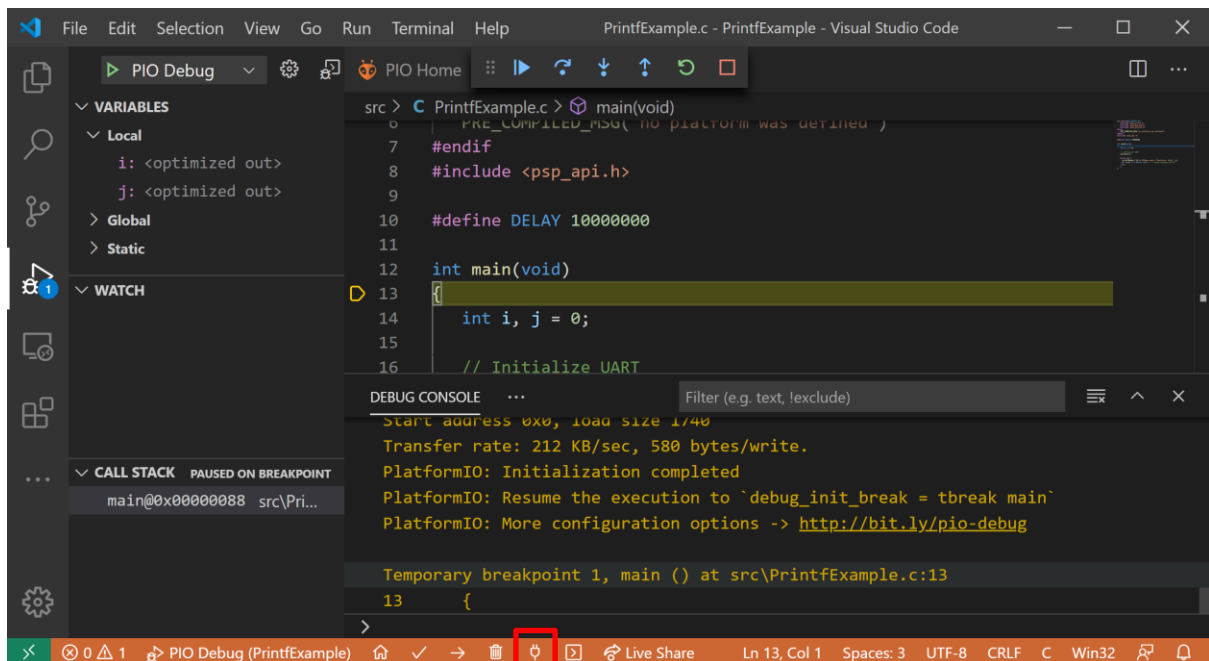


그림 2. Serial 모니터 시작중


그런 다음 프로그램을 실행합니다: 

그림 23 과 같이 인쇄 문자열(Hello RVfpga users!)과 직렬 모니터에 반복 인쇄된 반복 번호를 볼 수 있습니다.

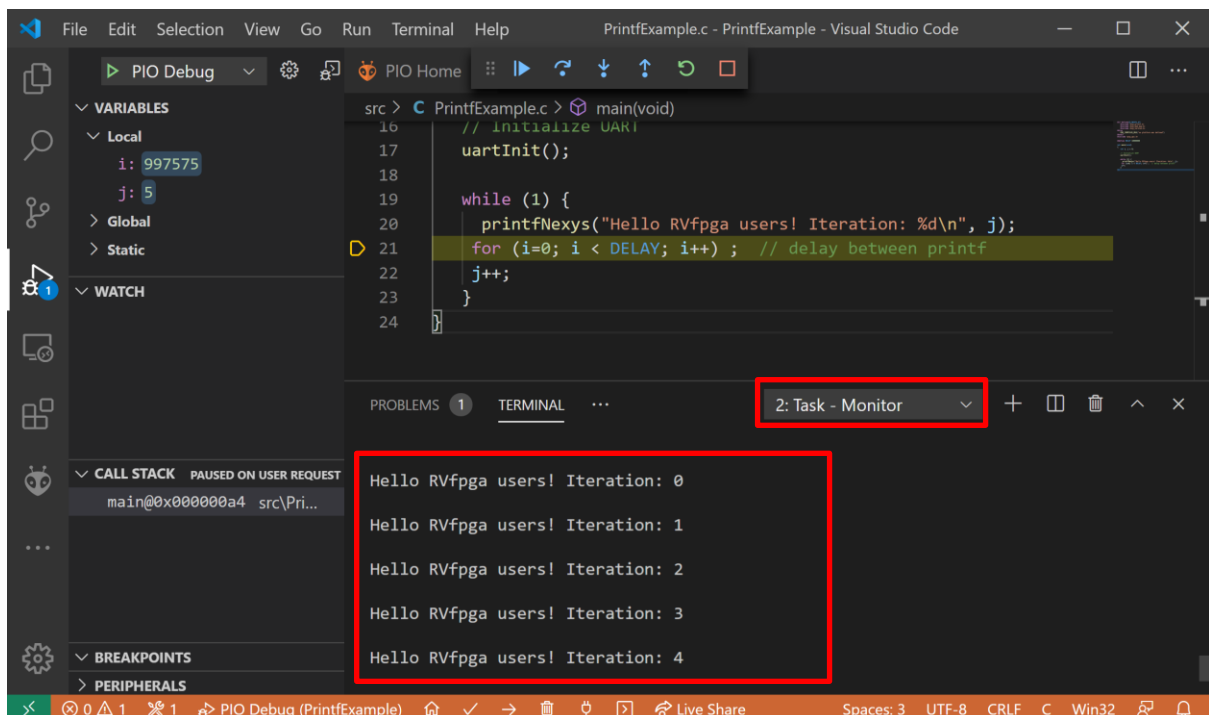


그림 23. PlatformIO 의 직렬 모니터에서 printfNexys 기능 출력

4. 연습

다음 연습을 완료하여 자신만의 C 프로그램을 만드십시오. Nexys A7 보드를 컴퓨터에 연결하고 전원을 켜 상태로 두면 다른 프로그램을 실행하는 사이에 RVfpgaNexys 를 보드에 다시 로드할 필요가 없습니다. 그러나 Nexys A7 보드를 끈 경우 섹션 2 의 3 단계에서 설명한 대로 PlatformIO 를 사용하여 RVfpgaNexys 를 보드에 다시 로드해야 합니다.

Western Digital 의 BSP 기능 `printfNexys` 를 사용하여 변수를 인쇄할 수 있습니다(섹션 3 참조).

Verilator 와 Whisper 를 사용하여 시뮬레이션 환경에서 이러한 프로그램을 수행할 수 있습니다

연습 1. 스위치에 LED 의 깜박이는 값을 C 프로그램으로 작성합니다. 이 값은 사람이 깜박이는 것을 볼 수 있을 정도로 느리게 펄스를 켜고 끌 수 있어야 합니다. 프로그램 이름은 **Flash SwitchesToLEDs.c** 로 지정합니다.

연습 2. LED 에 스위치의 역수 값을 표시하는 C 프로그램을 작성합니다. 예를 들어 스위치가 (이진수로) 0101010101 이면 LED 가 10101010101010101 로 표시되고, 스위치가: 111100001100 이면 LED 가 000011001100111 로 표시되어야 합니다. 프로그램 이름은 **DisplayInverse.c** 로 지정합니다.

연습 3. 모든 LED 가 켜질 때까지 LED 를 앞뒤로 (숫자가 증가하는) 스크롤하는 C 프로그램을 작성합니다. 그러면 패턴이 반복됩니다. 프로그램 이름을 **ScrollLEDs.c** 로 지정합니다.

이 프로그램으로 인해 다음과 같은 상황을 확인할 수 있습니다:

1. 먼저, 하나의 LED 가 오른쪽에서 왼쪽으로, 그리고 왼쪽에서 오른쪽으로 스크롤해야 합니다
2. 그런 다음 두 개의 LED 가 오른쪽에서 왼쪽으로, 그리고 왼쪽에서 오른쪽으로 스크롤해야 합니다.
3. 그런 다음 세 개의 LED 가 오른쪽에서 왼쪽으로, 그리고 왼쪽에서 오른쪽으로 스크롤해야 합니다.
4. 곧, 모든 LED 가 켜질 때까지
5. 패턴이 반복됩니다.

연습 4. 스위치의 4 개의 최하위 비트 (LSB)와 스위치의 4 개의 최상위 비트(MSB) 에 unsigned 4 비트를 추가하여 표시되는 C 프로그램을 작성합니다. LED 중 최하위 비트 (LSB) (가장 오른쪽) 비트 4 개에 결과를 표시합니다. 프로그램의 이름을 **4bitAdd.c** 로 지정합니다. unsigned 오버플로가 발생하면(carry out 이 1 일 때) LED 의 다섯 번째 비트가 켜집니다.

연습 5. Euclidean 알고리즘에 따라 a 와 b 두 숫자의 최대 공통 분수를 찾는 C 프로그램을 작성한다. a 와 b 값은 프로그램에서 정적으로 정의된 변수이어야 합니다. 프로그램 이름을 **GCD.c** 입니다. 여기 유클리드 알고리즘에 대한 몇 가지 추가 정보가 있습니다:

<https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm>

또한 "Euclidean algorithm"(유클리드 알고리즘)을 간단히 검색할 수도 있습니다..

연습 6. 피보나치(Fibonacci) 수열에서 처음 12 개의 숫자를 계산하고 그 결과를 길이 12 의 유한 벡터(즉, 배열) V 에 저장하는 C 프로그램을 작성합니다. 피보나치 수의 무한 순서는 다음과 같이 정의합니다:

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

즉, 요소(Elements) i 에 해당하는 피보나치 수는 영상 시리즈에서 이전 두 피보나치 수의 합입니다. 표 1 은 $i = 0 \sim 8$ 에 대한 피보나치 수를 보여줍니다.

표 1. 피보나치 수열

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

벡터의 dimension, N 은 프로그램에서 상수로 정의되어야 합니다. 프로그램 이름을 **Fibonacci.c** 입니다.

연습 7. N -요소 벡터(즉, 배열)가 주어지면, A 는 다른 벡터 B 를 생성하며, B 는 0 보다 큰 짝수인 A 의 요소만을 포함합니다. 또한 C 프로그램은 B 의 요소 수를 세고 프로그램의 끝에 그 값을 인쇄해야 합니다. 예를 들어, $N=12$ 와 $A = [0,1,2,7,-8,4,5,12,11,-2,6,3]$ 이면 $B = [2,4,12,6]$ 이 됩니다. B 에는 네 가지 요소가 있으므로 프로그램 종료 시 다음 사항이 인쇄 되어야 합니다.

Number of elements in B = 4.

printfNexys 기능을 사용하여 이 작업을 수행합니다. 프로그램 이름을 **EvenPositiveNumbers.c** 로 지정합니다. A 에 12 개의 요소가 있을 때 프로그램을 테스트 하십시오.

연습 8. 두 개의 N -요소 벡터(즉, 배열)가 주어지면, A 와 B 는 다음과 같이 정의되는 또 다른 벡터 C 를 생성한다:

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

새로운 벡터를 계산하는 프로그램을 C 로 작성하세요. 프로그램에 12 요소 배열을 사용합니다. 프로그램 이름을 **AddVectors.c** 로 지정합니다.

연습 9. 버블 정렬(Bubble sort) 알고리즘을 C 에서 구현합니다. 이 알고리즘은 다음 절차를 통해 벡터의 구성 요소를 오름차순으로 정렬합니다:

1. 완료될 때까지 벡터를 반복적으로 트래버스 (Traverse)합니다..
2. $V(i) > V(i+1)$ 인 경우 인접 구성 요소의 쌍을 교환합니다.
3. 모든 연속 구성 요소 쌍이 순서대로 정렬되면 알고리즘이 중지됩니다.

12 요소 배열을 사용하여 프로그램을 테스트합니다. 프로그램의 이름을 **BubbleSort.c** 로 지정합니다..

연습 10. 반복 곱셈을 사용하여 주어진 음수가 아닌 숫자 n 의 요인율(factorial) 계산하는 프로그램을 C로 작성합니다. 프로그램을 n 의 여러 값에 대해 테스트해야 하지만 최종 제출은 $n = 7$ 에 대해 테스트해야 합니다. 프로그램 끝에 요인(n) 값을 인쇄해야 합니다. n 은 프로그램 내에서 정적으로 정의된 변수이어야 합니다. 프로그램의 이름을 **Factor.c**로 지정합니다.