

Adafruit will not be shipping orders [Labor Day, Monday September 3rd](#). Expedited orders placed after 11am ET Friday August 31st will go out Tuesday September 4th.

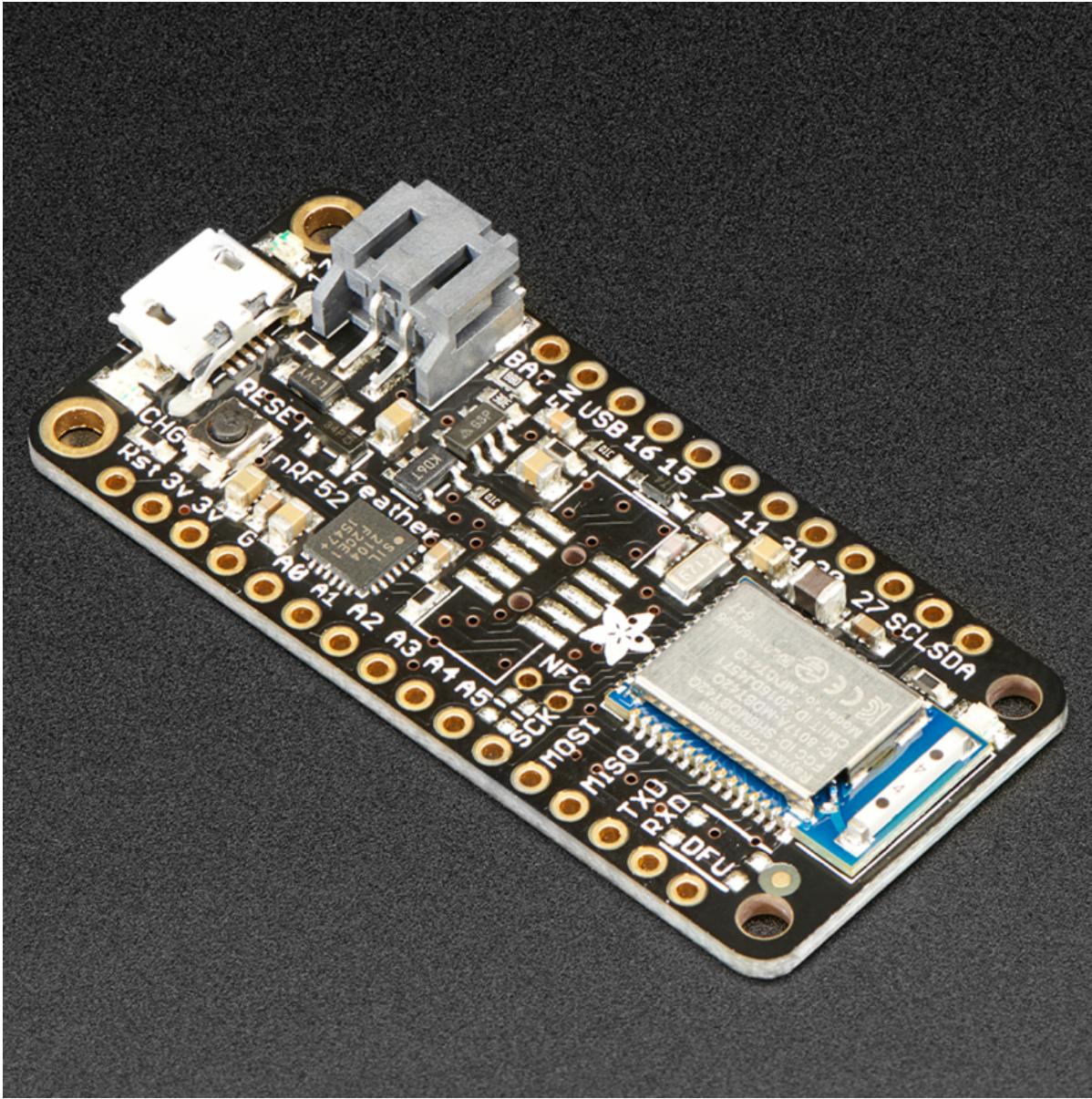
0

- 
- [SHOP](#)
- [BLOG](#)
- [LEARN](#)
- [FORUMS](#)
- [VIDEOS](#)
- [MY ACCOUNT](#)
- [SIGN OUT](#)
- [CLOSE MENU](#)

0 Items

Hello, Clayton Auzenne | [Sign Out](#) | [My Account](#)

- [SHOP](#)
- [BLOG](#)
- [LEARN](#)
- [FORUMS](#)
- [VIDEOS](#)
- [ADABOX](#)



## [Bluefruit nRF52 Feather Learning Guide](#)

Get started now with our most powerful Bluefruit board yet!

- [Introduction](#)
- [Device Pinout](#)
- [Assembly](#)
- [Arduino BSP Setup](#)

- [Arduino Board Setup](#)
- [Using the Bootloader](#)
- [Flashing the Bootloader](#)
- [Examples](#)
  - [Advertising: Beacon](#)
  - [BLE UART: Controller](#)
  - [Custom: HRM](#)
  - [BLE Pin I/O](#)
  - [Central BLEUART](#)
  - [Dual Roles BLEUART](#)
  - [Custom: Central HRM](#)
- [Bluefruit nRF52 API](#)
  - [AdafruitBluefruit](#)
  - [BLEGap](#)
  - [BLEAdvertising](#)
  - [BLEScanner](#)
  - [BLEService](#)
  - [BLECharacteristic](#)
  - [BLEClientService](#)
  - [BLEClientCharacteristic](#)
  - [BLEDiscovery](#)
  - [BLEDis](#)
  - [BLEUart](#)
  - [BLEClientUart](#)
  - [BLEBeacon](#)
  - [BLEMidi](#)
  - [BLEHidAdafruit](#)
  - [BLEAacs](#)
  - [BLEClientCts](#)
  - [BLECentral](#)
- [nRF52 ADC](#)
- [Memory Map](#)
- [Software Resources](#)
- [Downloads](#)
- [FAQs](#)
- 
- [Featured Products](#)
- [Single Page](#)
- [Download PDF](#)

## Contributors

[Kevin Townsend](#)

[Thach Ha](#)

[Feedback? Corrections?](#)

[MICROCONTROLLERS](#) [FEATHER](#) / [FEATHER BOARDS](#) [INTERNET OF THINGS - IOT](#) / [BLUEFRUIT](#) / [BLE](#)

## FAQs

by [Kevin Townsend](#)

**NOTE:** For FAQs relating to the **BSP**, see the dedicated [BSP FAQ list](#).

## What are the differences between the nRF51 and nRF52 Bluefruit boards? Which one should I be using?

The two board families take very different design approaches.

All of the nRF51 based modules are based on an AT command set (over UART or SPI), and require two MCUs to run: the nRF51 hosting the AT command parser, and an external MCU sending AT style commands.

The nRF52 boards run code directly on the nRF52, executing natively and calling the Nordic S132 SoftDevice (their proprietary Bluetooth Low Energy stack) directly. This allows for more efficient code since there is no intermediate AT layer or transport, and also allows for lower overall power consumption since only a single device is involved.

The nRF52 will generally give you better performance, but for situation where you need to use an MCU with a feature the nRF52 doesn't have (such as USB), the nRF51 based boards will still be the preferable solution.

## Can I run nRF51 Bluefruit sketches on the nRF52?

No. The two board families are fundamentally different, and have entirely separate APIs and programming models. If you are migrating from the nRF51 to the nRF52, you will need to redesign your sketches to use the newer API, enabling you to build code that runs natively on the nRF52832 MCU.

## Can I use the nRF52 as a Central to connect to other BLE peripherals?

The S132 Soft Device and the nRF52832 HW support Central mode, so yes this is *possible*. At this early development stage, though, there is only bare bones support for Central mode in the Adafruit nRF52 codebase, simply to test the HW and S132 and make sure that everything is configured properly. An example is provided of listening for incoming advertising packets, printing the packet contents and meta-data out to the Serial Monitor. We hope to add further Central mode examples in the future, but priority has been given to the Peripheral API and examples for the initial release.

## How are Arduino sketches executed on the nRF52832? Can I do hard real time processing (bit-banging NeoPixels, etc.)?

In order to run Arduino code on the nRF52 at the same time as the low level Bluetooth Low Energy stack, the Bluefruit nRF52 Feather uses FreeRTOS as a task scheduler. The scheduler will automatically switch between tasks, assigning clock cycles to the highest priority task at a given moment. This process is generally transparent to you, although it can have implications if you have hard real time requirements. There is no guarantee on the nRF52832 to meet hard timing requirements when the radio is enabled and being actively used for Bluetooth Low Energy. This isn't possible on the nRF52832 even without FreeRTOS, though, since the SoftDevice (Nordic's proprietary binary blob stack) has higher priority than any user code, including control over interrupt handlers.

## Can I use GDB to debug my nRF52832?

You can, yes, but it will require a Segger J-Link (that's what we've tested against anyway, other options exist), and it's an advanced operation. But if you're asking about it, you probably know that.

Assuming you have the Segger J-Link drivers installed, you can start Segger's GDB Server from the command line as follows (OSX/Linux used here):

```
$ JLinkGDBServer -device nrf52832_xxaa -if swd -speed auto
```

Then open a new terminal window, making sure that you have access to `gcc-arm-none-eabi-gdb` from the command line, and enter the following command:

```
$ ./arm-none-eabi-gdb something.ino.elf
```

'`something.ino.elf`' is the name of the .elf file generated when you built your sketch. You can find this by enabling 'Show verbose output during: [x] compilation' in the Arduino IDE preferences. You CAN run GDB without the .elf file, but pointing to the .elf file will give you all of the meta data like displaying the actual source code at a specific address, etc.

Once you have the (gdb) prompt, enter the following command to connect to the Segger GDB server (updating your IP address accordingly, since the HW isn't necessarily local!):

```
(gdb) target remote 127.0.0.1:2331
```

If everything went well, you should see the current line of code where the device is halted (normally execution on the nRF52 will halt as soon as you start the Segger GDB Server).

At this point, you can send GDB debug commands, which is a tutorial in itself! As a crash course, though:

- To continue execution, type 'monitor go' then 'continue'
- To stop execution (to read register values, for example.), type 'monitor halt'
- To display the current stack trace (when halted) enter 'bt'
- To get information on the current stack frame (normally the currently executing function), try these:
  - info frame: Display info on the current stack frame
  - info args: Display info on the arguments passed into the stack frame
  - info locals: Display local variables in the stack frame
  - info registers: Dump the core ARM register values, which can be useful for debugging specific fault conditions

## Are there any other cross platform or free debugging options other than GDB?

If you have a [Segger J-Link](#), you can also use [Segger's OZone debugger GUI](#) to interact with the device, though check the license terms since there are usage restrictions depending on the J-Link module you have.

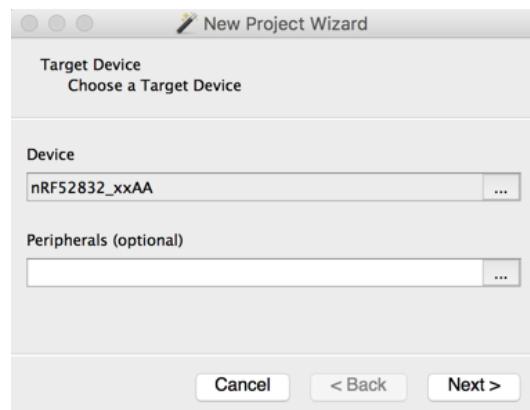
You will need to connect your nRF52 to the J-Link via the SWD and SWCLK pins on the bottom of the PCB, or if you are OK with fine pitch soldering via the SWD header.

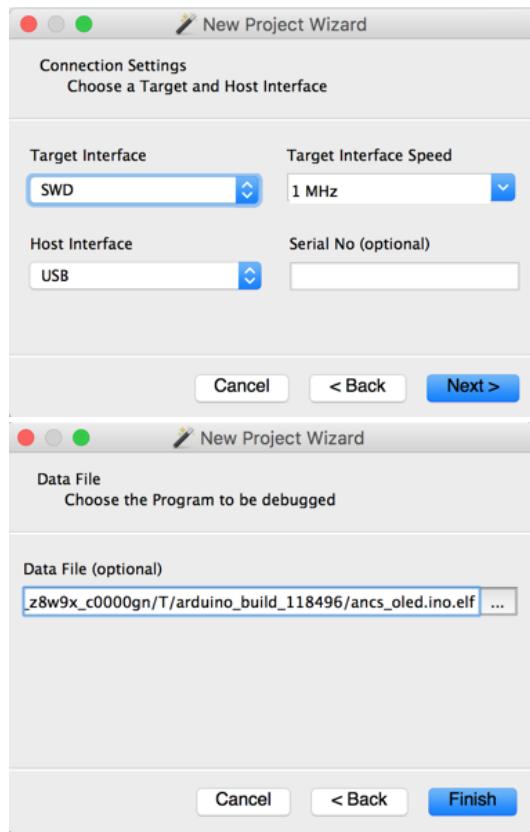
You can either solder on a standard [2x5 SWD header](#) on the pad available in the board, or you can solder wires to the SWD and SWCLK pads on the bottom of the PCB and use an [SWD Cable Breakout Board](#), or just connect cables directly to your J-Link via some other means.

You will also need to connect the **VTRef** pin on the JLink to **3.3V** on the Feather to let the J-Link know what voltage level the target has, and share a common GND by connecting the GND pins on each device.

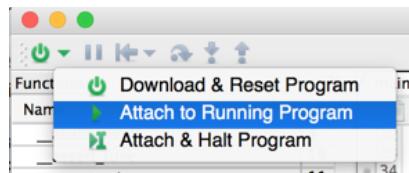
Before you can start to debug, you will need to get the .elf file that contains all the debug info for your sketch. You can find this file by enabling **Show Verbose Output During: compilation** in the **Arduino Preferences** dialogue box. When you build your sketch, you need to look at the log output, and find the .elf file, which will resemble something like this (it will vary depending on the OS used): /var/folders/86/hb2vp14n5\_5\_yvdz\_z8w9x\_c0000gn/T/arduino\_build\_118496/ancs\_oled.elf

In the OZone New Project Wizard, when prompted to select a target device in OZone select **nRF52832\_xxAA**, then make sure that you have set the Target Interface for the debugger to **SWD**, and finally point to the .elf file above:





Next select the **Attach to running program** option in the top-left hand corner, or via the menu system, which will cause the debugger to connect to the nRF52 over SWD:



Ozone - The J-Link Debugger V2.32 - /var/folders/86/hb2vp14n5\_5\_yvdz\_z8w9x\_c0000gn/T/arduino\_build\_118496/ancs\_oled.ino.elf

**Functions**

Name	#In
_aeabi_atexit	4
_assert_func	19
_cxa_atexit	11
_cxa_pure_virtual	1
_do_global_dtors_aux	12
_DSB	1
_get_FPSCR	1
_ISB	2
_libc_init_array	27
_NOP	3
_sclose	2
_set_BASEPRI	2
_set_CONTROL	1
_set_FPSCR	2
_SEV	1
_sflush_r	126
_smoreglue	17
_sfp	52
_sfputc_r	21
_sfputs_r	16
_sinit	38
_smakebuf_r	61
_sread	14
_seek	14
_ssputs_r	73
_swbuf_r	73

**Registers**

Name	Value
R0	-----
R1	-----
R2	-----
R3	-----
R4	-----
R5	-----
R6	-----
R7	-----
R8	-----
R9	-----
R10	-----
R11	-----
R12	-----
R13	-----
R14	-----
R15	-----
APSR	-----
EPSR	-----
IPSR	-----
PriMask	-----
BasePri	-----
FaultMask	-----
Control	-----
CycleCount	-----

**Memory1 @ 0**

00000000	00 04 00 20 E5 08 ...
00000010	...
00000020	...
00000030	...
00000040	...
00000050	...
00000060	...
00000070	...
00000080	...
00000090	...
000000A0	...
000000B0	...
000000C0	...
000000D0	...
000000E0	...

**Console**

```

Project.SetDevice ("nRF52832_xxAA");
Project.SetHostIF ("USB");
Project.SetTargetIF ("SWD");
Project.SetTIFSpeed (1 MHz);
Project.AddSvdFile ("Volumes/Ozone_MacO
File.Open ("~/var/folders/86/hb2vp14n5_5_y
File.Open: completed in 308 ms
Debug.SetConnectMode (CM_ATTACH);
Debug.Start();
J-Link: connected to target device
executed J-Link command "ReadintoTraceCa
executed J-Link command "ReadintoTraceCa

```

CPU running... | Ln 64 Ch 1 | Connected @ 1 MHz

At this point, you can click the **PAUSE** icon to stop program execution, and then analyze variables, or set breakpoints at appropriate locations in your program execution, and debug as you would with most other embedded IDEs!

The screenshot shows the Ozone - The J-Link Debugger V2.32 interface with the following windows:

- Functions**: Shows a list of global functions and their addresses.
- main.cpp**: The source code file being debugged. A breakpoint is set at line 69, which is highlighted in green.
- Disassembly**: Shows the assembly code corresponding to the C code. The instruction at address 00011158 is highlighted in green.
- Registers @ 00011158**: Displays the current values of R0, R1, and R2.
- Console**: Shows the command-line interface for interacting with the debugger.
- Memory1 @ 0**: Displays memory dump starting at address 0.

**Registers @ 00011158**

Name	Value
R0	0x00000001
R1	0x00000000
R2	0x80000000

**Console**

```

Project.SetHostIF("USB","");
Project.SetTargetIF("SWD");
Project.SetTIFSpeed("1 MHz");
Project.AddSDVIFile("/Volumes/Ozone_MacO
File.Open("/var/folders/86/hb2vp14n5_5_yvdz_z8w9x_c0000gn/T/arduino_build_118496/ancs_oled.ino.elf
File.Open completed in 308 ms
Debug.SetConnectMode(CM_ATTACH);
Debug.Start();
J-Link connected to target device
executed J-Link command "ReadintoTraceCa
executed J-Link command "ReadintoTraceCa
Debug.Halt();

```

**Memory1 @ 0**

Address	Value
00000000	00 04 00 20 E5 08 00 00 79 05 00 00 C5 08 00 00
00000010	B3 05 00 00 8D 05 00 00 97 05 00 00 00 00 00 00
00000020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030	A1 05 00 00 00 00 00 AB 05 00 00 B5 05 00 00
00000040	BF 05 00 00 C9 05 00 00 D3 05 00 00 DD 05 00 00
00000050	E7 05 00 00 F1 05 00 00 FB 05 00 00 05 06 00 00
00000060	0F 05 00 00 19 06 00 00 23 06 00 00 20 06 00 00
00000070	37 05 00 00 41 06 00 00 48 06 00 00 55 06 00 00
00000080	5F 05 00 00 69 06 00 00 73 06 00 00 70 06 00 00
00000090	87 05 00 00 91 06 00 00 9B 06 00 00 A5 06 00 00
000000A0	AF 06 00 00 B9 06 00 00 C3 06 00 00 CD 06 00 00
000000B0	D7 06 00 00 E1 06 00 00 EB 06 00 00 F5 06 00 00
000000C0	FF 06 00 00 09 07 00 00 13 07 00 00 10 07 00 00
000000D0	27 07 00 00 31 07 00 00 3B 07 00 00 45 07 00 00
000000E0	4F 07 00 00 59 07 00 00 63 07 00 00 6D 07 00 00

CPU halted

Clicking on the left-hand side of the text editor will set a breakpoint on line 69 in the image below, for example, and the selecting **Debug > Reset > Reset & Run** from the menu or icon will cause the board to reset, and you should stop at the breakpoint you set:

Ozone - The J-Link Debugger V2.32 - /var/folders/86/hb2vp14n5\_5\_yvdz\_z8w9x\_c0000gn/T/arduino\_build\_118496/ancs\_oled.ino.elf

The screenshot shows the Ozone J-Link Debugger interface with several windows open:

- Functions**: A list of global functions and variables, including `InitVariant`, `is\_within`, `ledOff`, `ledOn`, `ledToggle`, `loop`, `loop\_task`, `main`, `malloc`, `memchr`, `memcmp`, `memcpy`, `memmove`, `MemoryManagement\_Handler`, `memset`, `millis`, `min16`, `nffs\_area\_find\_corrupt\_scratch`, `nffs\_area\_free\_space`, `nffs\_area\_is\_current\_version`, `nffs\_area\_is\_scratch`, `nffs\_area\_magic\_is\_set`, `nffs\_area\_set\_magic`, `nffs\_area\_to\_disk`, `nffs\_block\_delete\_from\_ram`, and `nffs\_block\_entry\_alloc`.
- Disassembly**: Shows assembly code for the `setup` function. A specific instruction at address 0001C9E0 (opcode B570) is highlighted in yellow. The assembly code includes:
 

```

0001C9D6 2000    MOV
0001C9D8 3880    SUB
0001C9DA 2000    MOV
0001C9DC CAB5    LDM
0001C9DE 0001    MOV
0001C9E0 B570    PUSH
pinMode(BUTTON_A, INPUT_PULLUP)
0001C9E2 2102    MOV
0001C9E4 201F    MOV
oled.begin(SSD1306_SWITCHCAPVCC)
0001C9E6 4C2A    LDR
Bluefruit.begin();
0001C9E8 402A    LDR
bleancs.begin();
0001C9EA 4E2B    LDR
pinMode(BUTTON_A, INPUT_PULLUP)
0001C9EC F007 F004    BL
pinMode(BUTTON_B, INPUT_PULLUP)
0001C9F0 2102    MOV
0001C9F2 201E    MOV
0001C9F4 F007 F000    BL
pinMode(BUTTON_C, INPUT_PULLUP)
0001C9F8 2102    MOV
0001C9FA 201B    MOV
0001C9FB 0000    EOR
0001C9FC 0000    EOR
0001C9FD 0000    EOR
0001C9FE 0000    EOR
0001C9FF 0000    EOR
      
```
- Registers**: Shows the current values of registers R0, R1, and R2.
- Console**: Displays a series of debugger commands:
 

```

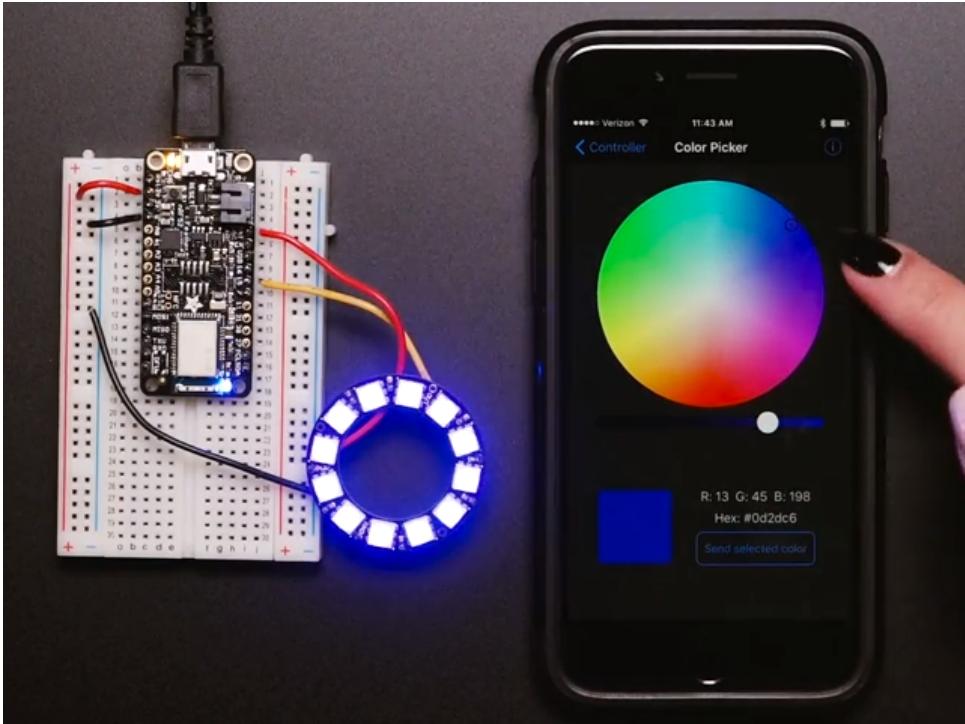
Break.SetOnSrc ("ancs_oled.ino:122");
Debug.SetResetMode (RM_RESET_AND_RUN)
Debug.Reset();
Debug.Continue();
Debug.SetResetMode (RM_BREAK_AT_SYMBC)
Debug.Reset();
Debug.Continue();
Break.ClearOnSrc ("ancs_oled.ino:69");
Break.SetOnSrc ("ancs_oled.ino:69");
Debug.SetResetMode (RM_BREAK_AT_SYMBC)
Debug.Reset();
Debug.Continue();
      
```

CPU halted

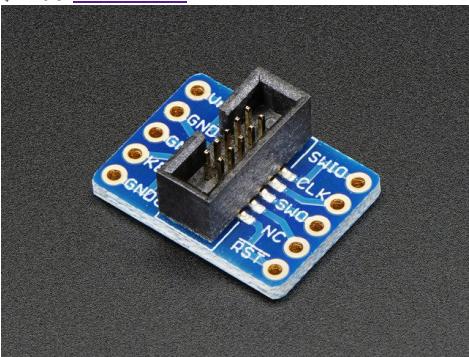
You can experiment with adding some of the other debug windows and options via the **View** menu item, such as the **Call Stack** which will show you all of the functions that were called before arriving at the current breakpoint:

The screenshot shows the J-Link Debugger interface with several open windows:

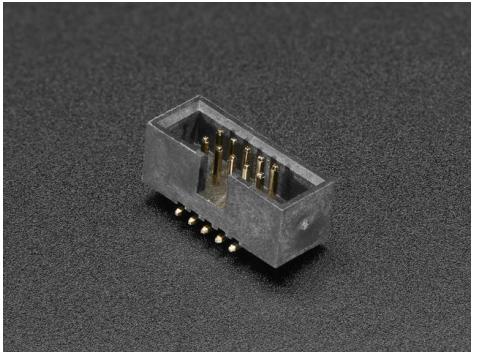
- Functions**: Shows function names like `_eagle_start`, `_cxt_start`, `cxt_axiom`, `cxt_axiom_virtual`, `_global_dblots_aux`, `DSR`, `_get_FPSCR`, `l2c_init_array`, `_NOP`, `_sclose`, `_set_BASEPR`, `_set_CTRNOL`, `_set_EPSR`, `_SEV`, `_fflush_r`, `_smoreglue`.
- Source Files**: Shows file details for `annc_oled.ino` and `main.cpp`. The `main.cpp` pane highlights the `oled.begin()` call.
- Disassembly**: Shows assembly code for the `annc_oled.ino` file, with the instruction `push r4, r5, r6, r7` highlighted.
- Registers**: Shows register values for R0 through R11.
- Call Stack**: Shows the stack trace for the current task.
- Console**: Displays J-Link connection logs and command history.



Adafruit Feather nRF52 Bluefruit LE  
\$24.95 [Add to Cart](#)



SWD (2x5 1.27mm) Cable Breakout Board  
\$1.95 [Add to Cart](#)



SWD 0.05" Pitch Connector - 10 Pin SMT Box Header

\$1.50 [Add to Cart](#)



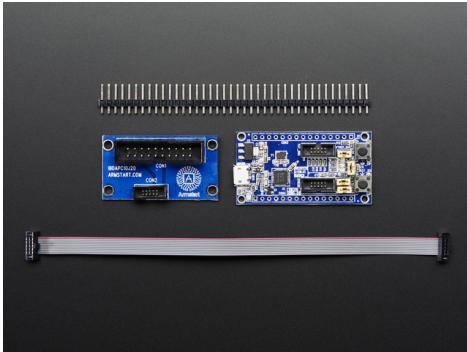
SEGGER J-Link BASE - JTAG/SWD Debugger

\$399.95 [Add to Cart](#)



SEGGER J-Link EDU - JTAG/SWD Debugger

\$69.95 [Add to Cart](#)



IBDAP - CMSIS-DAP JTAG/SWD Debug Adapter Kit

\$29.95 [Add to Cart](#)



10-pin 2x5 Socket-Socket 1.27mm IDC (SWD) Cable - 150mm long

\$2.95 [Add to Cart](#)

## RELATED GUIDES

[Adafruit seesaw](#)

[An I2C to ... whatever! interface](#)

by [Dean Miller](#)



[PWM, ADC, NeoPixels...You want to use em but if you don't have hardware support it's impossible! Glue an Arduino to the side? No longer! Seesaw gives you multiple hardware interfaces over I2C, for robotics, sensing, lighting and more!](#)

## [Debugging CircuitPython On SAMD w/Atmel Studio 7](#)

### [Un-Constricted CircuitPython Debugging in a Windows Environment!](#)

by M. Schroeder



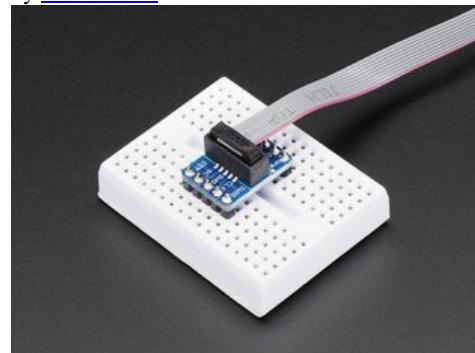
```
Output
Show output from: Debug
pulseIn_intr_hdlr:: total_diff: 4482; last_ms: 28500; last_us: 69
pulseIn_intr_hdlr:: total_diff: 63; last_ms: 33327; last_us: 152
pulseIn_intr_hdlr:: total_diff: 69; last_ms: 33327; last_us: 215
pulseIn_intr_hdlr:: total_diff: 62; last_ms: 33330; last_us: 28
pulseIn_intr_hdlr:: total_diff: 41; last_ms: 33330; last_us: 90
pulseIn_intr_hdlr:: total_diff: 63; last_ms: 33346; last_us: 124
pulseIn_intr_hdlr:: total_diff: 40; last_ms: 33346; last_us: 187
pulseIn_intr_hdlr:: total_diff: 62; last_ms: 33350; last_us: 863
pulseIn_intr_hdlr:: total_diff: 17; last_ms: 33350; last_us: 925
pulseIn_intr_hdlr:: total_diff: 63; last_ms: 33357; last_us: 637
pulseIn_intr_hdlr:: total_diff: 63; last_ms: 33368; last_us: 119
```

This guide will help you to start debugging CircuitPython on the SAMD21 or SAMD51, using Windows and Atmel Studio 7. Take the squeeze off virtual environments and do some work in your native operating system!

## [Programming an M0 using an Arduino](#)

### [computers are overrated anyhow](#)

by Dean Miller



Programming ARM core microcontrollers can be bulky and time consuming. This guide shows how to do it simply using an ATSAMD21 based arduino-compatible board.

## [Adafruit seesaw](#)

### [An I2C to ... whatever! interface](#)

by Dean Miller



PWM, ADC, NeoPixels... You want to use em but if you don't have hardware support it's impossible! Glue an Arduino to the side? No longer! SeeSaw gives you multiple hardware interfaces over I2C for robotics, sensing, lighting and more!

X

## OUT OF STOCK NOTIFICATION

YOUR NAME

YOUR EMAIL

[NOTIFY ME](#)

X

## Create Wishlist

Title

Description

[Create Wishlist](#) [CLOSE](#)

- [CONTACT](#)
- [SUPPORT](#)
- [DISTRIBUTORS](#)
- [EDUCATORS](#)
- [JOBS](#)
- [FAQ](#)
- [SHIPPING & RETURNS](#)
- [TERMS OF SERVICE](#)
- [PRIVACY & LEGAL](#)
- [ABOUT US](#)

[ENGINEERED IN NYC](#) Adafruit ®

"You're only given one little spark of madness. You mustn't lose it" - [Robin Williams](#)

