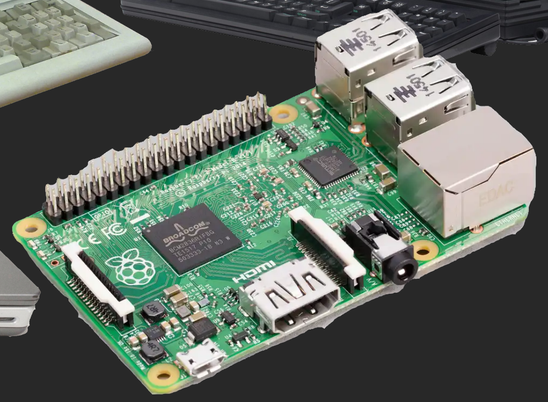
Blink them to death using Embedded Swift

Hostname: Eric Bariaux

Occupation: Software Engineer

Really: Geek at heart

Up time: 53y 207d 7h 11m



Embedded Swift

Swift

- Introduced at WWDC 2014 for Apple platforms
- Dec 2015, OSS Swift project, swift.org site and Linux port
- macOS, Linux and Windows
- Scripting, Server side, Serverless, …

- No support for Swift on embedded systems

Embedded Swift

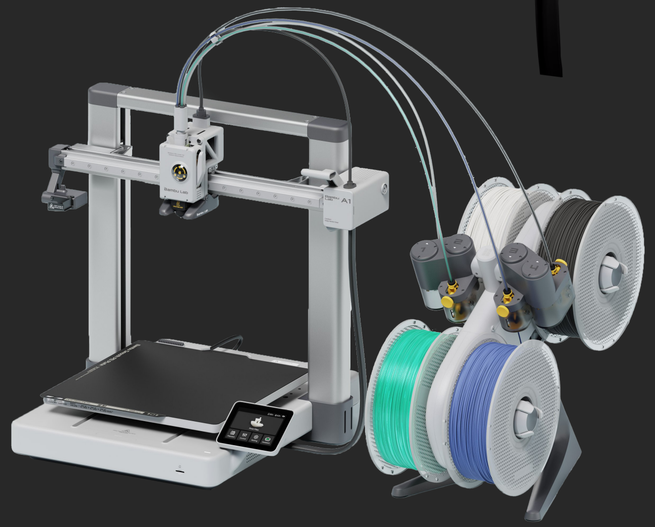
Embedded


POLAR


WITHINGS


TESLA

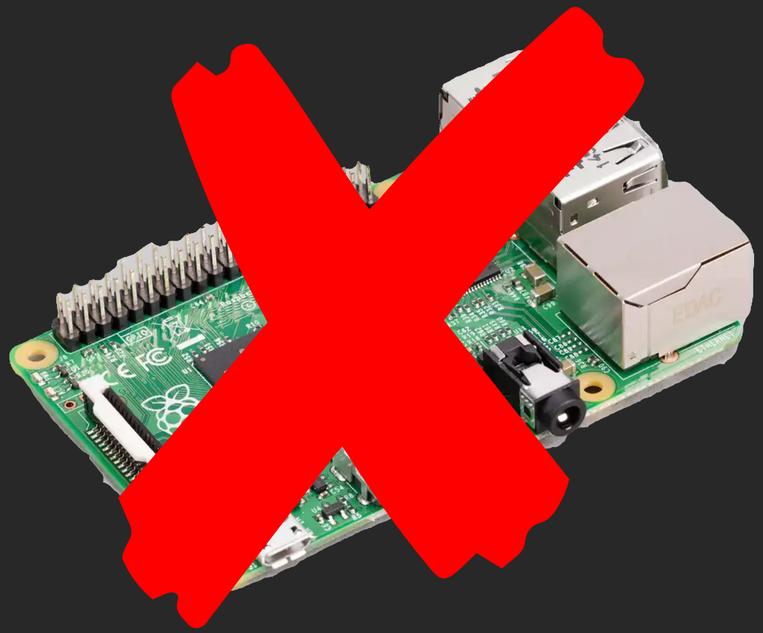

kg
54.3
Withings




marantz
AV Surround Receiver
Blu-ray
INPUT
VOLUME


Pencil Pro

Embedded



Embedded systems limitations

Processor	Single core 32-bit	Single core 64-bit		Reduce
Frequency	64 MHz	520 MHz	8x	Runtime overhead
RAM	256 KB	512 MB	2000x	Memory footprint
Flash	1 MB	8 GB	8000x	Executable code size

Embedded Swift

- Official initiative from Apple
- Subset of the language, not dialect
- Compilation mode enforces constraints to achieve goals of reducing:
  - Runtime overhead
  - Memory footprint
  - Executable code size

How ?

- Remove everything that is dynamic
  - Dynamic reflection facilities (such as mirrors, as? downcasts, and printing arbitrary values)
  - Existential types (any)
  - Generics instantiation
  - Obj-C interop
  - Dynamic code loading (plug-ins)
- Minimal runtime library / no need for metadata
- Reduced Swift Standard Library (e.g. no Codable)
- Agressive dead code stripping

In practice

- `swiftc -target <target triple> -enable-experimental-feature Embedded -wmo file.swift -c -o output.o`
- Target triple e.g. for nRF: armv7em-none-none-eabi
- From Embedded Swift user manual
  - Embedded Swift is a compilation model that's analogous to a traditional C compiler in the sense that the compiler produces an object file (.o) that can be simply linked with your existing code, and it's not going to require you to port any libraries or runtimes.
- Linking is done as usual using the embedded platform toolchain

Let's get started

Step 1

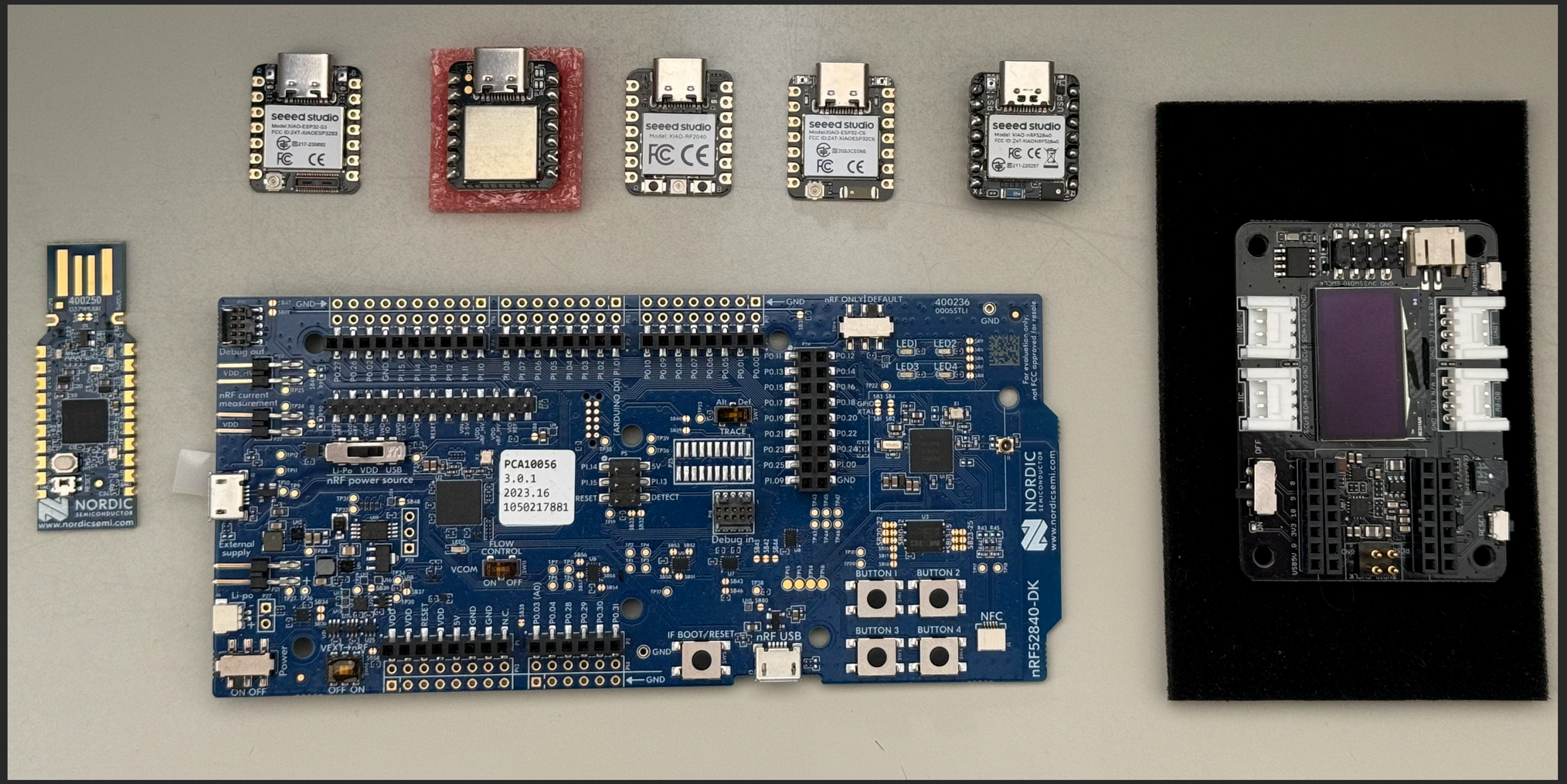
Pick a board

Pick a board

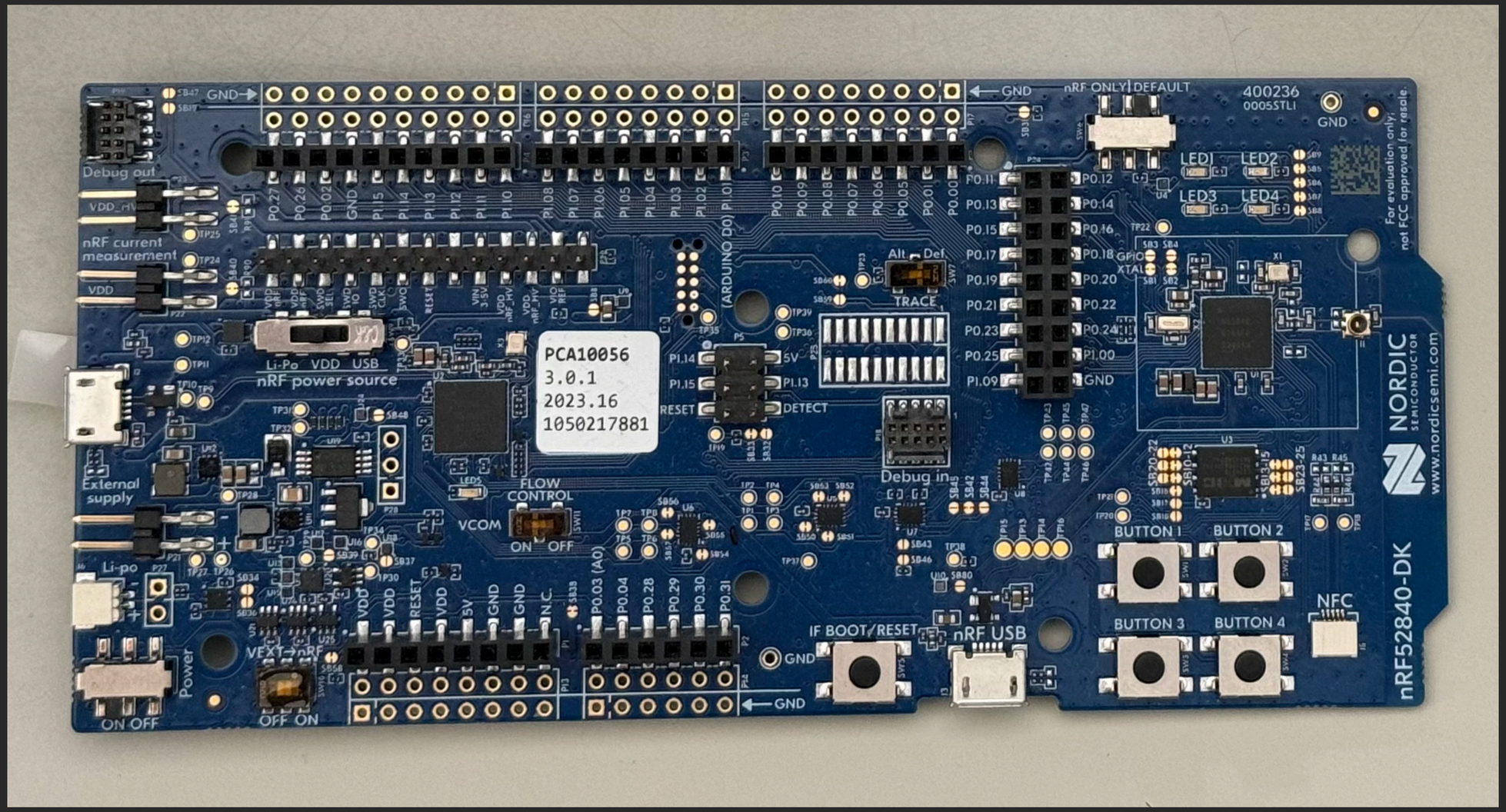
- STM32
- RP2040 / RP2035
- nRF52x
- ESP32C6

- Panic Playdate
- Flipper Zero

Really. Just pick one

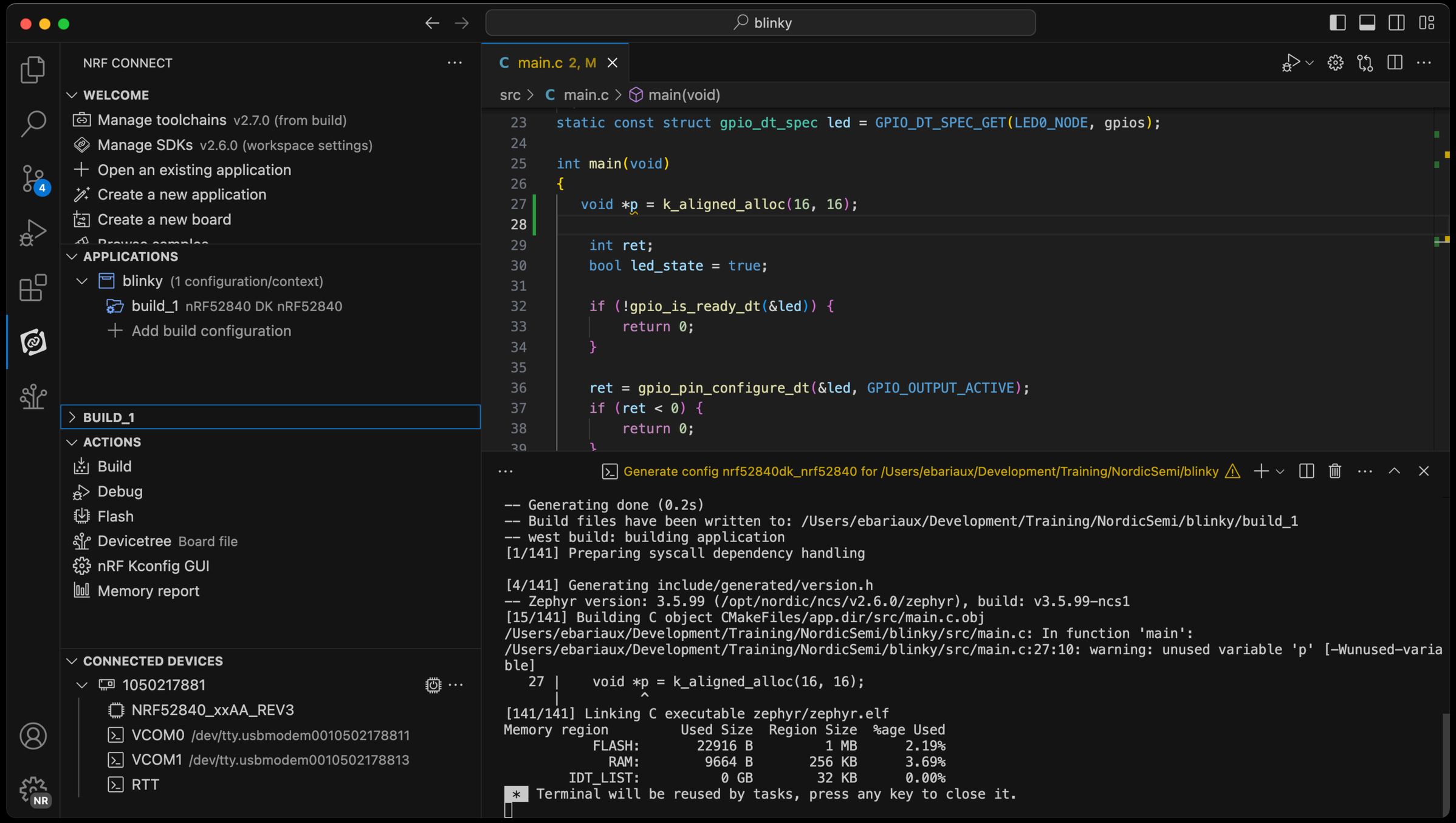


Target : nRF52840 DK



Step 2

Install and test the dev environment for your target (No Swift)


blinky
NRF CONNECT
WELCOME
Manage toolchains v2.7.0 (from build)
Manage SDKs v2.6.0 (workspace settings)
Open an existing application
Create a new application
Create a new board
APPLICATIONS
blinky (1 configuration/context)
build_1 nRF52840 DK nRF52840
Add build configuration
BUILD_1
ACTIONS
Build
Debug
Flash
Devicetree Board file
nRF Kconfig GUI
Memory report
CONNECTED DEVICES
1050217881
NRF52840_xxAA_REV3
VCOM0 /dev/tty.usbmodem0010502178811
VCOM1 /dev/tty.usbmodem0010502178813
RTT
main.c 2, M
src > main.c > main(void)
23 static const struct gpio_dt_spec led = GPIO_DT_SPEC_GET(LED0_NODE, gpios);
24
25 int main(void)
26 {
27 void *p = k_aligned_alloc(16, 16);
28
29 int ret;
30 bool led_state = true;
31
32 if (!gpio_is_ready_dt(&led)) {
33 return 0;
34 }
35
36 ret = gpio_pin_configure_dt(&led, GPIO_OUTPUT_ACTIVE);
37 if (ret < 0) {
38 return 0;
Generate config nrf52840dk_nrf52840 for /Users/ebariaux/Development/Training/NordicSemi/blinky
-- Generating done (0.2s)
-- Build files have been written to: /Users/ebariaux/Development/Training/NordicSemi/blinky/build_1
-- west build: building application
[1/141] Preparing syscall dependency handling
[4/141] Generating include/generated/version.h
-- Zephyr version: 3.5.99 (/opt/nordic/ncs/v2.6.0/zephyr), build: v3.5.99-ncs1
[15/141] Building C object CMakeFiles/app.dir/src/main.c.obj
/Users/ebariaux/Development/Training/NordicSemi/blinky/src/main.c: In function 'main':
/Users/ebariaux/Development/Training/NordicSemi/blinky/src/main.c:27:10: warning: unused variable 'p' [-Wunused-variable]
27 | void *p = k_aligned_alloc(16, 16);
[141/141] Linking C executable zephyr/zephyr.elf
Memory region Used Size Region Size %age Used
FLASH: 22916 B 1 MB 2.19%
RAM: 9664 B 256 KB 3.69%
IDT_LIST: 0 GB 32 KB 0.00%
* Terminal will be reused by tasks, press any key to close it.

Step 3

Test the Embedded Swift examples

https://github.com/apple/swift-embedded-examples/tree/main/nrfx-blink-sdk

- Before trying to use Swift with the Zephyr SDK, make sure your environment works and can build the provided C/C++ sample projects, in particular:
  - Try building and running the "simple/blink" example from Zephyr written in C.

Building

- Make sure you have a recent nightly Swift toolchain that has Embedded Swift support.
- Build the program in the Zephyr virtualenv, specify the nightly toolchain to be used via the `TOOLCHAINS` environment variable and the target board type via the `-DBOARD=...` CMake setting:

```
$ cd nrfx-blink-sdk
$ source ~/zephyrproject/.venv/bin/activate
(.venv) export TOOLCHAINS='<toolchain-name>'
(.venv) cmake -B build -G Ninja -DBOARD=nrf52840dk_nrf52840 -DUSE_CCACHE=0 .
(.venv) cmake --build build
```

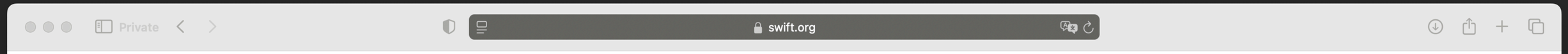
Running

- Connect the nRF52840-DK board over a USB cable to your Mac using the J-Link connector on the board.
- Use `nrfjprog` to upload the firmware and to run it:

```
(.venv) nrfjprog --recover --program build/zephyr/zephyr.hex --verify
(.venv) nrfjprog --run
```

- The green LED should now be blinking in a pattern.

https://www.swift.org/download/#snapshots



Development Snapshots

Swift snapshots are prebuilt binaries that are automatically created from the branch. These snapshots are not official releases. They have gone through automated unit testing, but they have not gone through the full testing that is performed for official releases.

Toolchain

main

January 10, 2025

Toolchain package installer (.pkg)

- Debugging Symbols

Download Toolchain

release/6.1

January 23, 2025

Toolchain package installer (.pkg)

- Debugging Symbols

Download Toolchain

Instructions (Toolchain)

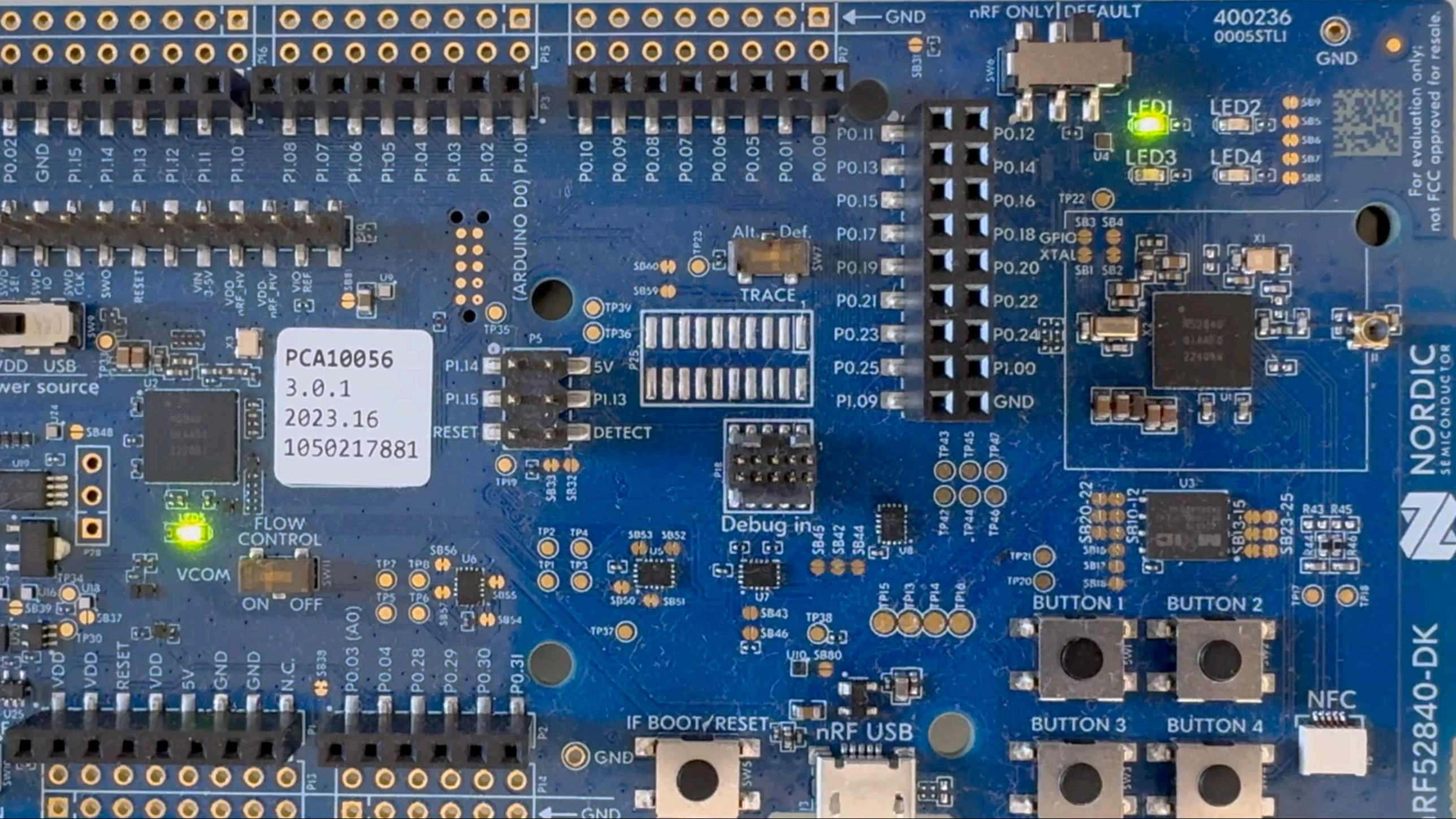
Static Linux SDK

main

release/6.1

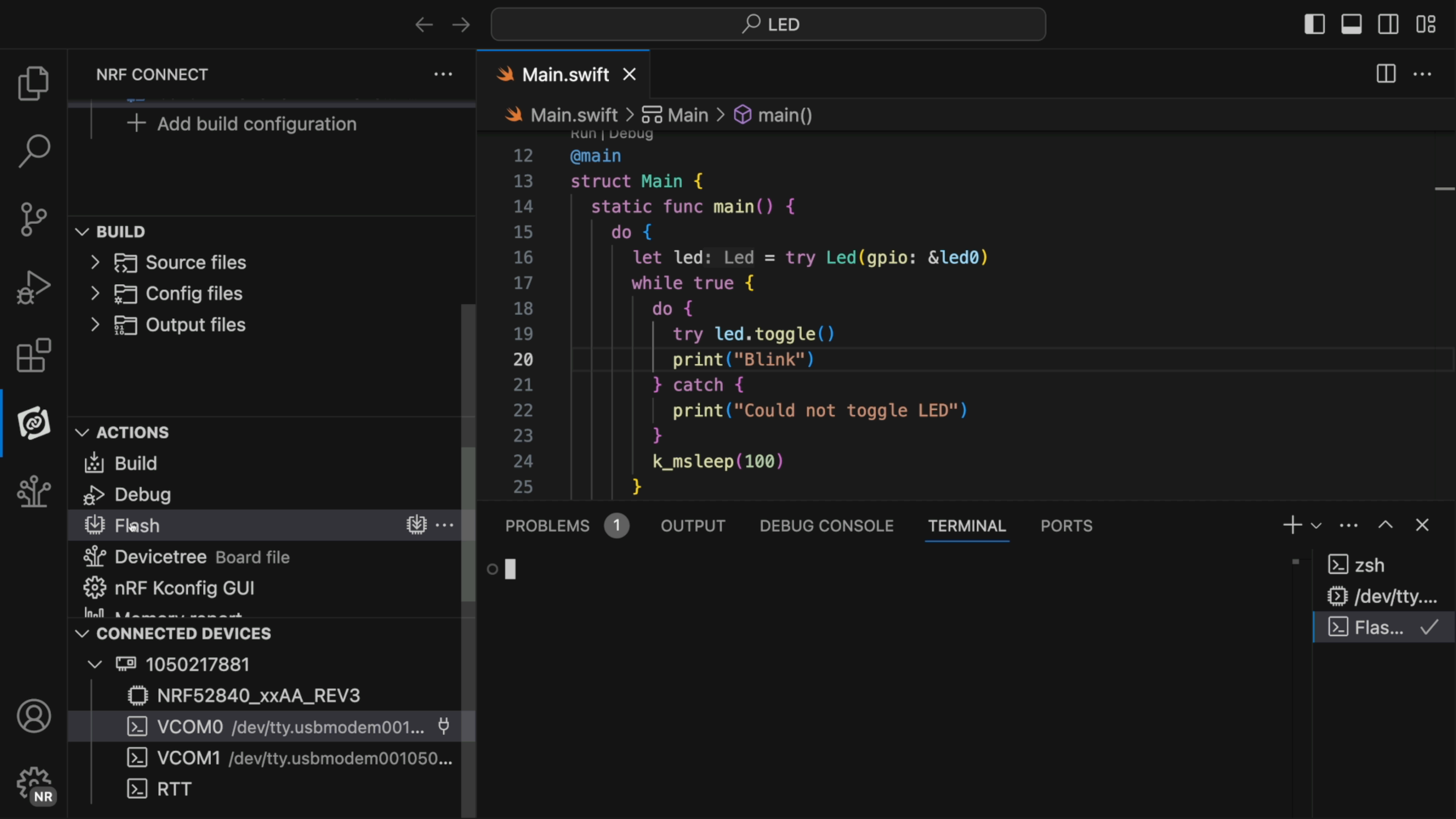
```
ebariaux@eadu nrfx-blink-sdk % source ~/bin/nrf-2.7.0-env.sh
ebariaux@eadu nrfx-blink-sdk % plutil -extract CFBundleIdentifier raw /Library/Developer/Toolchains/swift-DEVELOPMENT-SNAPSHOT-2025-01-10-a.xctoolchain/Info.plist
org.swift.62202501101a
ebariaux@eadu nrfx-blink-sdk % export TOOLCHAINS='org.swift.62202501101a'
ebariaux@eadu nrfx-blink-sdk % cmake -B build -G Ninja -DBOARD=nrf52840dk_nrf52840 -DUSE_CACHE=0 .
Loading Zephyr default modules (Zephyr base).
-- Application: /Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk
-- CMake version: 3.31.3
-- Using NCS Toolchain 2.7.20240620.1065210518403 for building. (/opt/nordic/ncs/toolchains/f8037e9b83/cmake)
-- Found Python3: /opt/nordic/ncs/toolchains/f8037e9b83/bin/python3 (found suitable version "3.9.6", minimum required is "3.8") found components: Interpreter
-- Cache files will be written to: /Users/ebariaux/Library/Caches/zephyr
-- Zephyr version: 3.6.99 (/opt/nordic/ncs/v2.7.0/zephyr)
-- Found west (found suitable version "1.2.0", minimum required is "0.14.0")
CMake Warning at /opt/nordic/ncs/v2.7.0/zephyr/cmake/modules/boards.cmake:110 (message):
  Deprecated BOARD=nrf52840dk_nrf52840 specified, board automatically changed
  to: nrf52840dk/nrf52840.
Call Stack (most recent call first):
  /opt/nordic/ncs/v2.7.0/zephyr/cmake/modules/zephyr_default.cmake:132 (include)
  /opt/nordic/ncs/v2.7.0/zephyr/share/zephyr-package/cmake/ZephyrConfig.cmake:66 (include)
  /opt/nordic/ncs/v2.7.0/zephyr/share/zephyr-package/cmake/ZephyrConfig.cmake:92 (include_boilerplate)
  CMakeLists.txt:2 (find_package)


-- Board: nrf52840dk, qualifiers: nrf52840
-- Found host-tools: zephyr 0.16.5 (/opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk)
-- Found toolchain: zephyr 0.16.5 (/opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk)
-- Found Dtc: /opt/nordic/ncs/toolchains/f8037e9b83/bin/dtc (found suitable version "1.6.1", minimum required is "1.4.6")
-- Found BOARD.dts: /opt/nordic/ncs/v2.7.0/zephyr/boards/nordic/nrf52840dk/nrf52840dk_nrf52840.dts
-- Generated zephyr.dts: /Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/build/zephyr/zephyr.dts
-- Generated devicetree_generated.h: /Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/build/zephyr/include/generated/devicetree_generated.h
-- Including generated dts.cmake file: /Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/build/zephyr/dts.cmake
Parsing /opt/nordic/ncs/v2.7.0/zephyr/Kconfig
Loaded configuration '/opt/nordic/ncs/v2.7.0/zephyr/boards/nordic/nrf52840dk/nrf52840dk_nrf52840_defconfig'
Merged configuration '/Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/prj.conf'
Configuration saved to '/Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/build/zephyr/.config'
Kconfig header saved to '/Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/build/zephyr/include/generated/autoconf.h'
-- Found GnuLd: /opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk/arm-zephyr-eabi/arm-zephyr-eabi/bin/ld.bfd (found version "2.38")
-- The C compiler identification is GNU 12.2.0
-- The CXX compiler identification is GNU 12.2.0
-- The ASM compiler identification is GNU
-- Found assembler: /opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk/arm-zephyr-eabi/bin/arm-zephyr-eabi-gcc
-- Using ccache: /opt/nordic/ncs/toolchains/f8037e9b83/bin/ccache
-- The Swift compiler identification is Apple 6.2
-- Configuring done (5.9s)
-- Generating done (0.2s)
CMake Warning:
  Manually-specified variables were not used by the project:

    USE_CACHE


-- Build files have been written to: /Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/build
ebariaux@eadu nrfx-blink-sdk % cmake --build build
[1/141] Preparing syscall dependency handling

[6/141] Generating include/generated/version.h
-- Zephyr version: 3.6.99 (/opt/nordic/ncs/v2.7.0/zephyr), build: v3.6.99-ncs2
[69/141] Linking Swift static library app/libapp.a
warning: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/ranlib: archive library: app/libapp.a the table of contents is empty (no object file members in the library define global symbols)
[136/141] Linking C executable zephyr/zephyr_pre0.elf
/opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk/arm-zephyr-eabi/bin/../lib/gcc/arm-zephyr-eabi/12.2.0/../../../../arm-zephyr-eabi/bin/ld.bfd: warning: orphan section `.swift_modhash' from `app/libapp.a(Main.swift.obj)' being placed
in section `.swift_modhash'
[141/141] Linking C executable zephyr/zephyr.elf
/opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk/arm-zephyr-eabi/bin/../lib/gcc/arm-zephyr-eabi/12.2.0/../../../../arm-zephyr-eabi/bin/ld.bfd: warning: orphan section `.swift_modhash' from `app/libapp.a(Main.swift.obj)' being placed
in section `.swift_modhash'
```

```
[ebariaux@eadu nrfx-blink-sdk % source ~/bin/nrf-2.7.0-env.sh
[ebariaux@eadu nrfx-blink-sdk % plutil -extract CFBundleIdentifier raw /Library/Developer/Toolchains/swift-DEVELOPMENT-SN
org.swift.62202501101a
[ebariaux@eadu nrfx-blink-sdk % export TOOLCHAINS='org.swift.62202501101a'
[ebariaux@eadu nrfx-blink-sdk % cmake -B build -G Ninja -DBOARD=nrf52840dk_nrf52840 -DUSE_CACHE=0 .
Loading Zephyr default modules (Zephyr base).
-- Application: /Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk
-- CMake version: 3.31.3
-- Using NCS Toolchain 2.7.20240620.1065210518403 for building. (/opt/nordic/ncs/toolchains/f8037e9b83/cmake)
-- Found Python3: /opt/nordic/ncs/toolchains/f8037e9b83/bin/python3 (found suitable version "3.9.6", minimum required is
-- Cache files will be written to: /Users/ebariaux/Library/Caches/zephyr
-- Zephyr version: 3.6.99 (/opt/nordic/ncs/v2.7.0/zephyr)
-- Found west (found suitable version "1.2.0", minimum required is "0.14.0")
CMake Warning at /opt/nordic/ncs/v2.7.0/zephyr/cmake/modules/boards.cmake:110 (message):
  Deprecated BOARD=nrf52840dk_nrf52840 specified, board automatically changed
  to: nrf52840dk/nrf52840.
Call Stack (most recent call first):
  /opt/nordic/ncs/v2.7.0/zephyr/cmake/modules/zephyr_default.cmake:132 (include)
  /opt/nordic/ncs/v2.7.0/zephyr/share/zephyr-package/cmake/ZephyrConfig.cmake:66 (include)
  /opt/nordic/ncs/v2.7.0/zephyr/share/zephyr-package/cmake/ZephyrConfig.cmake:92 (include_boilerplate)
  CMakeLists.txt:2 (find_package)


-- Board: nrf52840dk, qualifiers: nrf52840
-- Found host-tools: zephyr 0.16.5 (/opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk)
-- Found toolchain: zephyr 0.16.5 (/opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk)
-- Found Dtc: /opt/nordic/ncs/toolchains/f8037e9b83/bin/dtc (found suitable version "1.6.1", minimum required is "1.4.6"
-- Found BOARD.dts: /opt/nordic/ncs/v2.7.0/zephyr/boards/nordic/nrf52840dk/nrf52840dk_nrf52840.dts
-- Generated zephyr.dts: /Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sd
-- Generated devicetree_generated.h: /Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/n
-- Including generated dts.cmake file: /Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples
Parsing /opt/nordic/ncs/v2.7.0/zephyr/Kconfig
Loaded configuration '/opt/nordic/ncs/v2.7.0/zephyr/boards/nordic/nrf52840dk/nrf52840dk_nrf52840_defconfig'
Merged configuration '/Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/p
Configuration saved to '/Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk
```


PCA10056
3.0.1
2023.16
1050217881
400236
0005STLI
nRF ONLY DEFAULT
LED1
LED2
LED3
LED4
BUTTON 1
BUTTON 2
BUTTON 3
BUTTON 4
NFC
nRF USB
IF BOOT/RESET
TRACE
Debug in
FLOW CONTROL
VCOM
ON OFF
(ARDUINO D0) P1.01
P0.03 (A0)
P0.04
P0.28
P0.29
P0.30
P0.31
P0.11
P0.13
P0.15
P0.17
P0.19
P0.21
P0.23
P0.25
P1.09
P0.12
P0.14
P0.16
P0.18
P0.20
P0.22
P0.24
P1.00
GND
5V
P1.14
P1.15
P1.13
RESET
DETECT
XTAL
GPIO
For evaluation only;
not FCC approved for resale.
NORDIC
SEMICONDUCTOR
nRF52840-DK

Step 4

Set-up a proper dev environment


LED
NRF CONNECT
Add build configuration
BUILD
Source files
Config files
Output files
ACTIONS
Build
Debug
Flash
Devicetree Board file
nRF Kconfig GUI
CONNECTED DEVICES
1050217881
NRF52840_xxAA_REV3
VCOM0 /dev/tty.usbmodem001...
VCOM1 /dev/tty.usbmodem001050...
RTT
Main.swift
Main.swift > Main > main()
Run | Debug
@main
struct Main {
static func main() {
do {
let led: Led = try Led(gpio: &led0)
while true {
do {
try led.toggle()
print("Blink")
} catch {
print("Could not toggle LED")
}
k_msleep(100)
}
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
zsh
/dev/tty....
Flas...

Step 6

Start from the Embedded Swift examples and create your own project from there

Main.swift

```
@main
struct Main {
  static func main() {
    // Note: & in Swift is not the "address of" operator, but on a global
variable declared in C
    // it will give the correct address of the global.
    gpio_pin_configure_dt(&led0, GPIO_OUTPUT | GPIO_OUTPUT_INIT_HIGH |
GPIO_OUTPUT_INIT_LOGICAL)
    while true {
      gpio_pin_toggle_dt(&led0)
      k_msleep(100)
    }
  }
}
```

BridgingHeader.h

```
#include <autoconf.h>

#include <zephyr/kernel.h>
#include <zephyr/drivers/gpio.h>

#define LED0_NODE DT_ALIAS(led0)
static struct gpio_dt_spec led0 = GPIO_DT_SPEC_GET(LED0_NODE, gpios);
```

Main.swift

```
@main
struct Main {
  static func main() {
    // Note: & in Swift is not the "address of" operator, but on a global
variable declared in C
    // it will give the correct address of the global.
    gpio_pin_configure_dt(&led0, GPIO_OUTPUT | GPIO_OUTPUT_INIT_HIGH |
GPIO_OUTPUT_INIT_LOGICAL)
    while true {
      gpio_pin_toggle_dt(&led0)
      k_msleep(100)
    }
  }
}
```

Main.swift

```
struct Led {
  let gpio: UnsafePointer<gpio_dt_spec>

  init(gpio: UnsafePointer<gpio_dt_spec>) {
    self.gpio = gpio
    // Note: & in Swift is not the "address of" operator, but on a global
variable declared in C
    // it will give the correct address of the global.
    gpio_pin_configure_dt(gpio, GPIO_OUTPUT | GPIO_OUTPUT_INIT_HIGH |
GPIO_OUTPUT_INIT_LOGICAL)
  }

  func toggle() {
     gpio_pin_toggle_dt(gpio)
  }
}
```

Main.swift

```
@main
struct Main {
  static func main() {
    let led = Led(gpio: &led0)
    while true {
      led.toggle()
      k_msleep(100)
    }
  }
}
```

Main.swift

```
enum LedError: Error {
  case notReady
}



struct Led {
  let gpio: UnsafePointer<gpio_dt_spec>

  init(gpio: UnsafePointer<gpio_dt_spec>) throws {
    if (!gpio_is_ready_dt(gpio)) {
      throw LedError.notReady
    }
    …
}
```

Build error

```
…/EmbeddedSwift-nRF52-Examples/LED/Main.swift:30:22: error: cannot use a
value of protocol type 'any Error' in embedded Swift


28 |   init(gpio: UnsafePointer<gpio_dt_spec>) throws {
29 |     if (!gpio_is_ready_dt(gpio)) {
30 |       throw LedError.notReady
   |                      `- error: cannot use a value of protocol type 'any
Error' in embedded Swift
31 |     }
32 |
```

Main.swift

```
struct Led {
  let gpio: UnsafePointer<gpio_dt_spec>

  init(gpio: UnsafePointer<gpio_dt_spec>) throws(LedError) {
    if (!gpio_is_ready_dt(gpio)) {
      throw .notReady
    }
  …
```

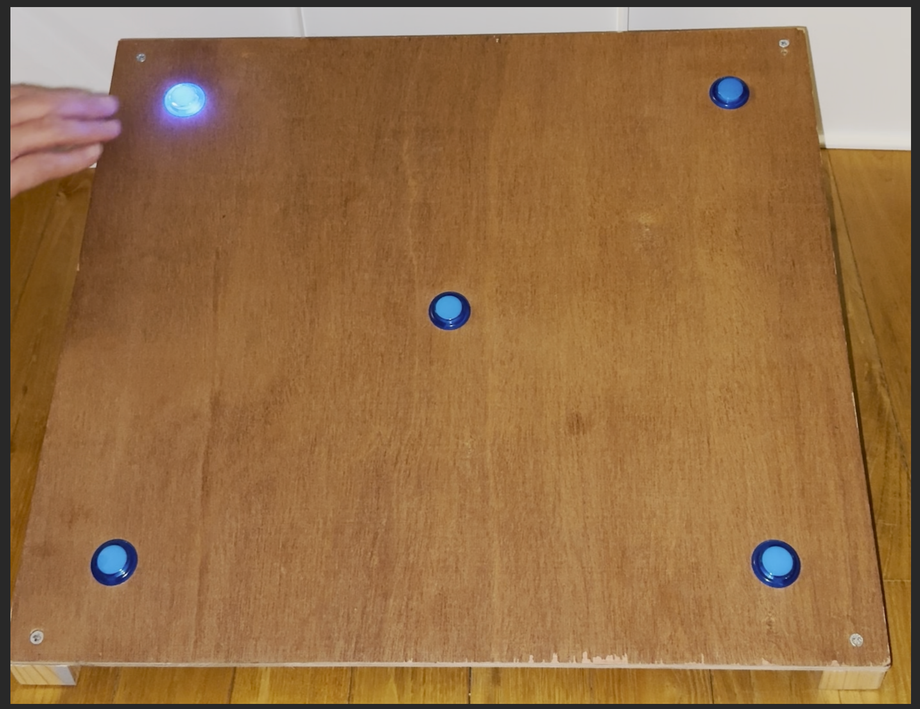
Step 6

Just keep going at it...

Eventually you will get there

```
@main
struct Main {
  static func main() {
    let led = Led(gpio: &led0)

    let _ = Button<Led>(gpio: &button, context: led) { _, callback, _ in
      let led = Button<Led>.getContext(callback)
      print("Button pressed")
      led.toggle()
    }

    let logic = Logic()
    logic.run()
  }
}
```

Personal projects



And many more ideas…

Let's continue the discussion...

⚠️ This is not a CGI, come find me if you want a business card