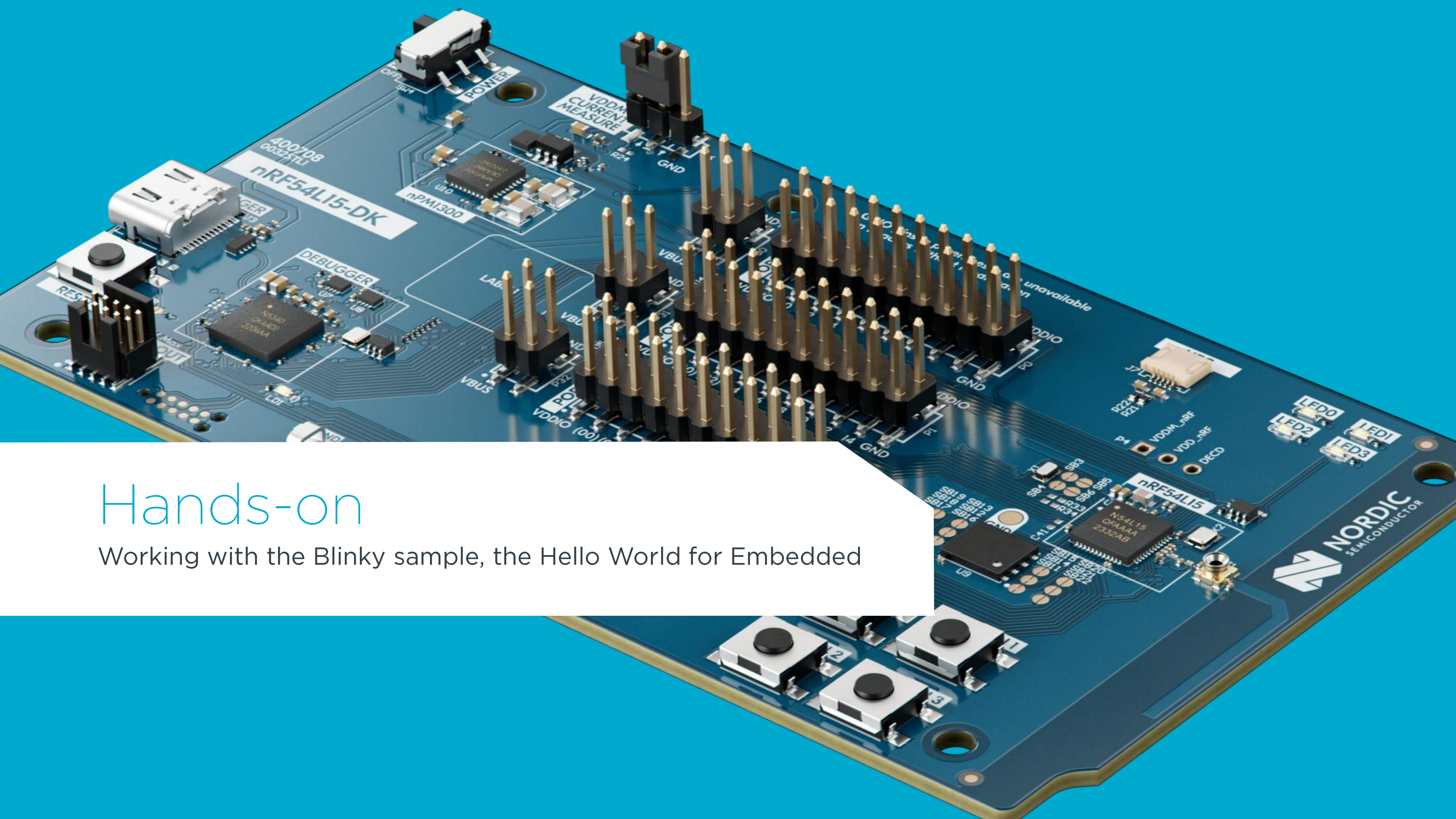
Wireless IoT Entwicklung mit Nordic Semiconductor

Kapitel 1: IDE Einführung & Blinky

February 2026

nRF54L15DK / nRF Connect SDK 3.2.1

NORDIC®
SEMICONDUCTOR



Hands-on

Working with the Blinky sample, the Hello World for Embedded

Simple Blinky

The Hello World for embedded MCU

- Build the Zephyr “blinky” sample
- Understand the nRF Connect SDK VS Code Extension
- Understand the source code
- Modify the devicetree showing different approaches
- (opt.) Change while loop into a Zephyr WorkQueue thread
- (opt.) Create custom kernel configuration (Kconfig) settings

Preparations

- Exercise 1

  Installing nRF Connect SDK and VS Code

  https://academy.nordicsemi.com/topic/exercise-1-1/

- (opt.) Exercise 2

  Build and flash your first nRF Connect SDK application

  https://academy.nordicsemi.com/topic/exercise-2-1/

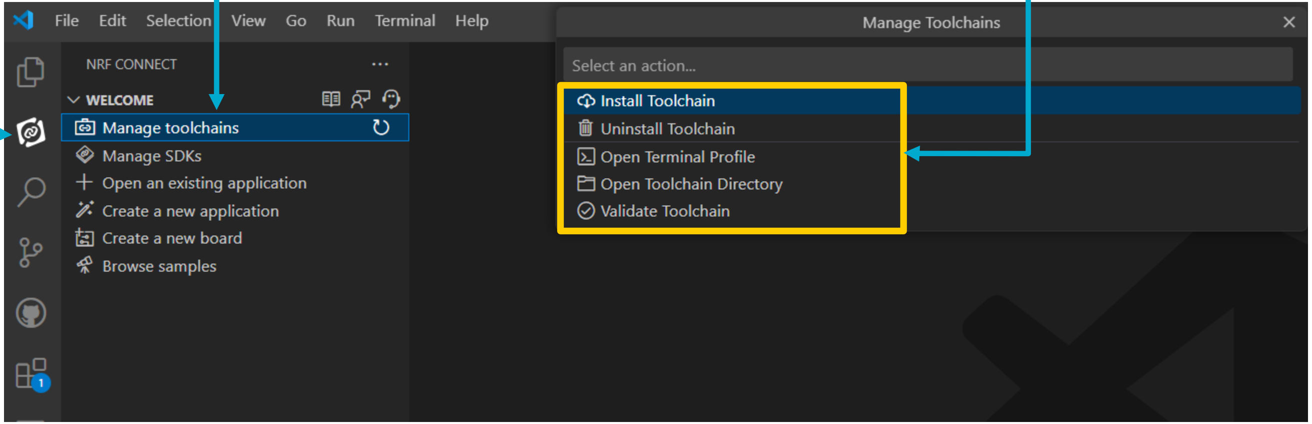
Navigating the IDE

Manage Toolchain

Select the nRF Connect Extension

Manage toolchains

Install Toolchain, default installation path
<drive>:\ncs\toolchain\<build version>

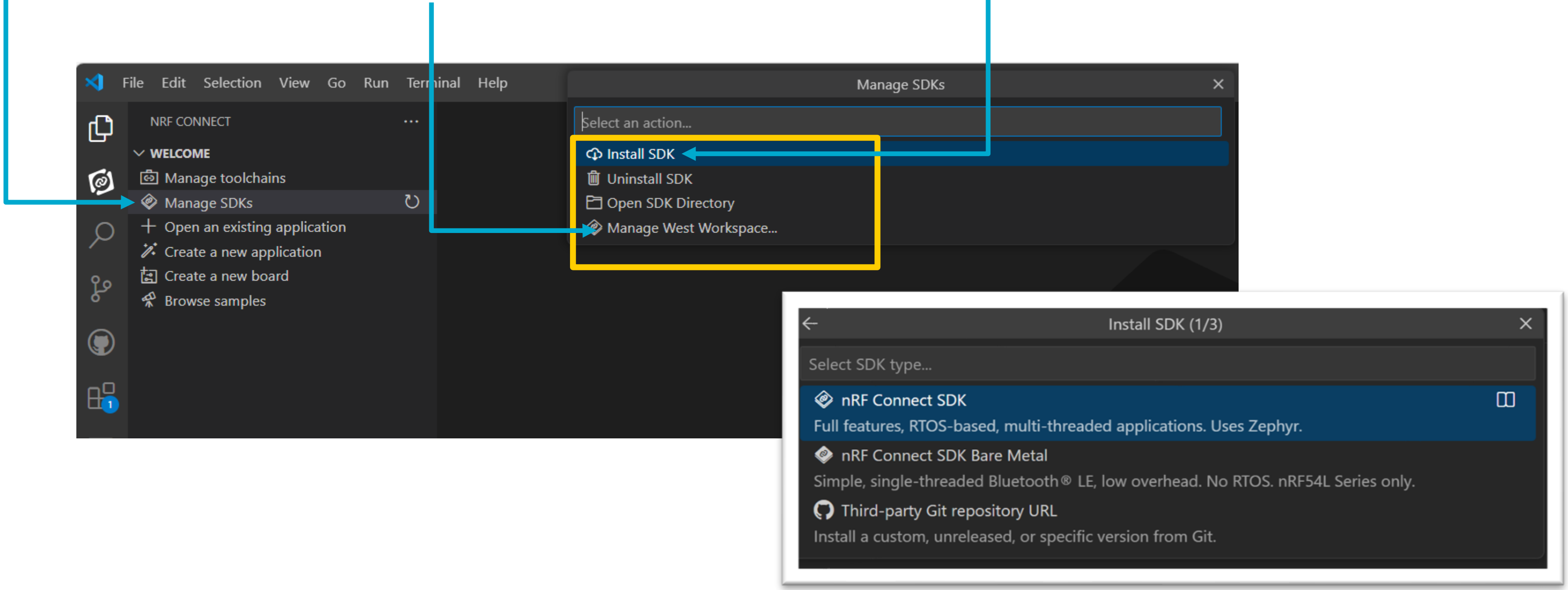


Manage SDK’s installations

Manage SDKs

Create/open
Workspace projects

Install SDK, default installation path
<drive>:\ncs\v3.2.1



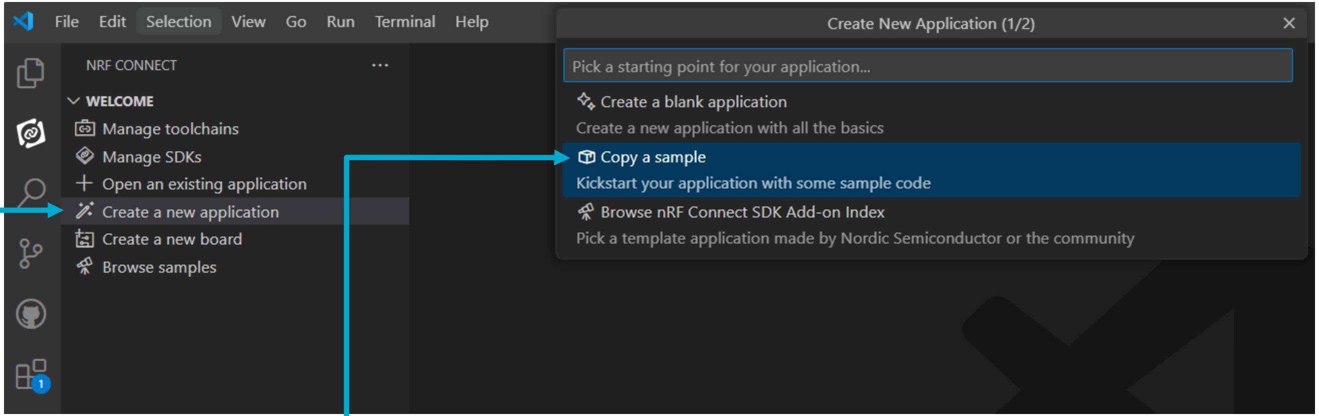
nRF Connect SDK
Full features, RTOS-based, multi-threaded applications. Uses Zephyr.

nRF Connect SDK Bare Metal
Simple, single-threaded Bluetooth® LE, low overhead. No RTOS. nRF54L Series only.

Third-party Git repository URL
Install a custom, unreleased, or specific version from Git.

Building a application

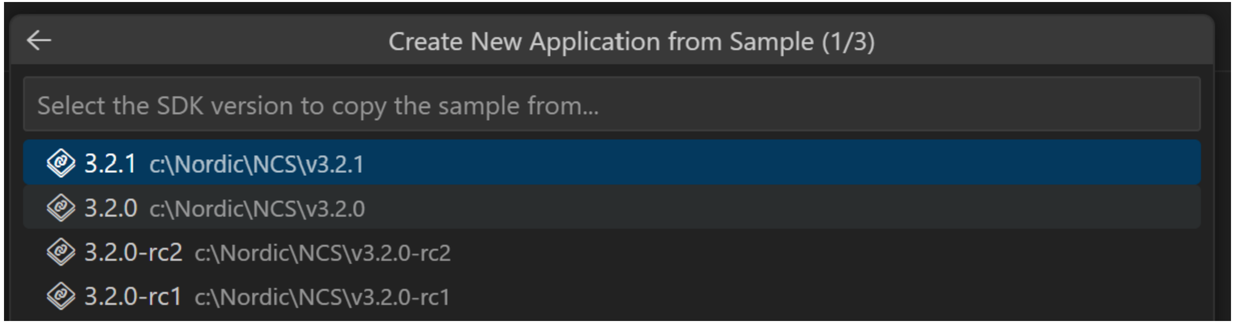
Create a new application


File
Edit
Selection
View
Go
Run
Terminal
Help
NRF CONNECT
WELCOME
Manage toolchains
Manage SDKs
Open an existing application
Create a new application
Create a new board
Browse samples
Create New Application (1/2)
Pick a starting point for your application...
Create a blank application
Create a new application with all the basics
Copy a sample
Kickstart your application with some sample code
Browse nRF Connect SDK Add-on Index
Pick a template application made by Nordic Semiconductor or the community

Copy a sample

This will create a copy from a sample or application out of the SDK repository. This will create a stand-alone application.

Select the version of the SDK



Create New Application from Sample (1/3)

Select the SDK version to copy the sample from...

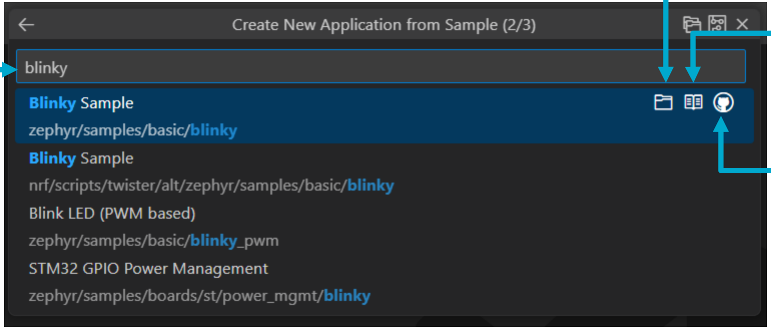
- 3.2.1 c:\Nordic\NCS\v3.2.1
- 3.2.0 c:\Nordic\NCS\v3.2.0
- 3.2.0-rc2 c:\Nordic\NCS\v3.2.0-rc2
- 3.2.0-rc1 c:\Nordic\NCS\v3.2.0-rc1

Copy a sample

This will ask you to select a SDK version to browse and copy the samples or application.

Selecting a sample: blinky

Search for the **“blinky”** sample that is part of the **zephyr** repository



Create New Application from Sample (2/3)

blinky

Blinky Sample
zephyr/samples/basic/**blinky**

Blinky Sample
nrf/scripts/twister/alt/zephyr/samples/basic/**blinky**

Blink LED (PWM based)
zephyr/samples/basic/**blinky**_pwm

STM32 GPIO Power Management
zephyr/samples/boards/st/power_mgmt/**blinky**

Open file location

Open documentation

Open github repository

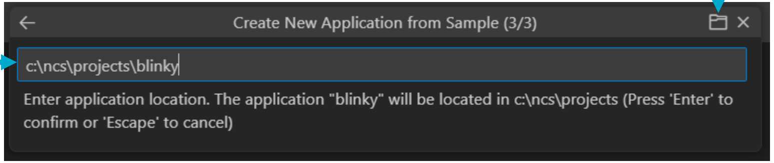
If the location of the sample/application starts with **“zephyr”** this sample is provided by Zephyr.

If the location starts with **“nrf”** the sample is provided by Nordic Semiconductor and part nRF Connect SDK

Location of new application

Full path to location of project.
By default the last used location followed by the application name.

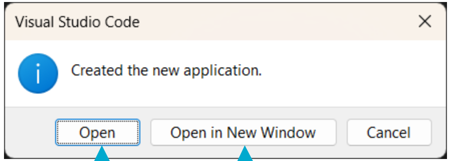
Use icon to browse for target file location of new application



c:\ncs\projects\blinky

Enter application location. The application "blinky" will be located in c:\ncs\projects (Press 'Enter' to confirm or 'Escape' to cancel)

Where to open the new application

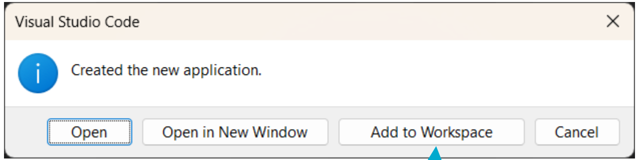


Visual Studio Code

Created the new application.

Open project in current Visual Code instance

Open a new Visual Code instance and open the project



Visual Studio Code

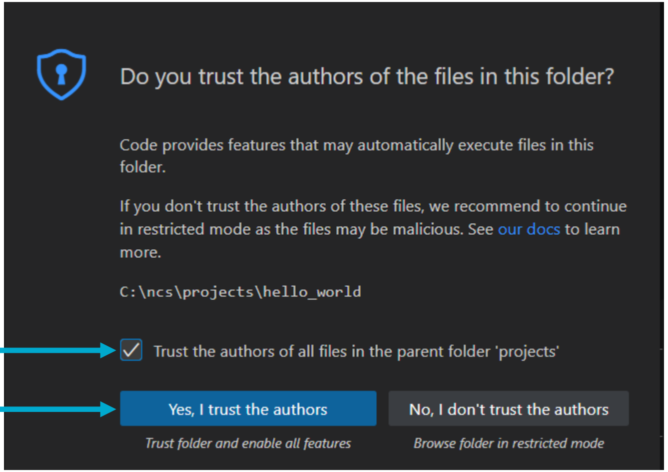
Created the new application.

Add project to workspace in current Visual Code instance

Permission to access project location

Optional to trust the complete parent folder of the project

Mandatory for Visual Code to trust the location of the project



Do you trust the authors of the files in this folder?

Code provides features that may automatically execute files in this folder.

If you don't trust the authors of these files, we recommend to continue in restricted mode as the files may be malicious. See our docs to learn more.

```
C:\ncs\projects\hello_world
```

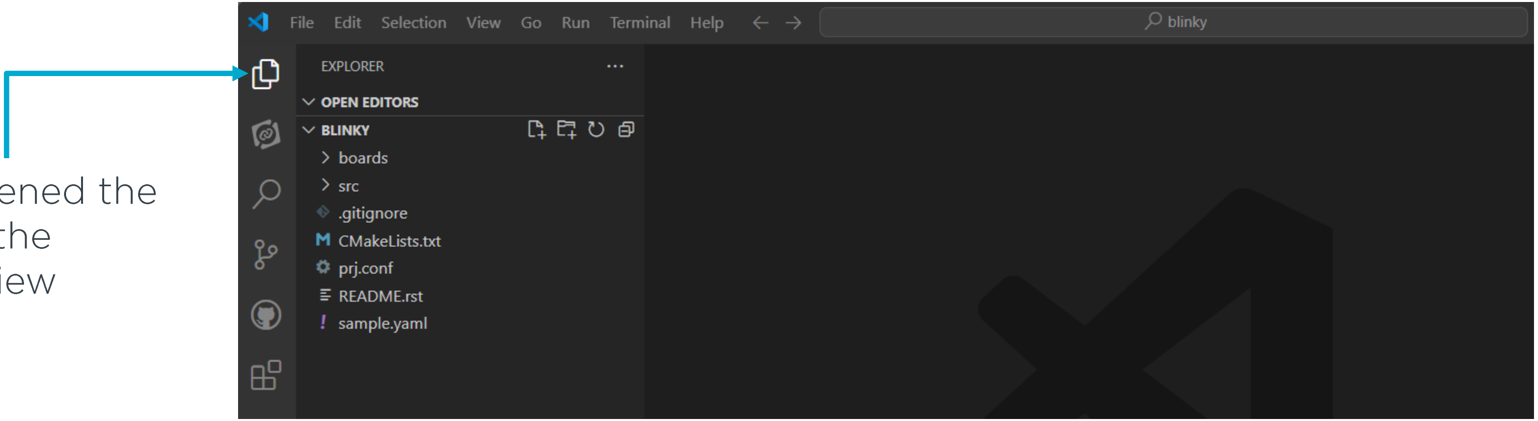
☑ Trust the authors of all files in the parent folder 'projects'

Yes, I trust the authors
Trust folder and enable all features
No, I don't trust the authors
Browse folder in restricted mode

Application created

The new application based on the sample has been created in a Visual Code Workspace (not to confuse with West Workspace)

VC has opened the project in the Explorer view



Setting up build configurations 1/3

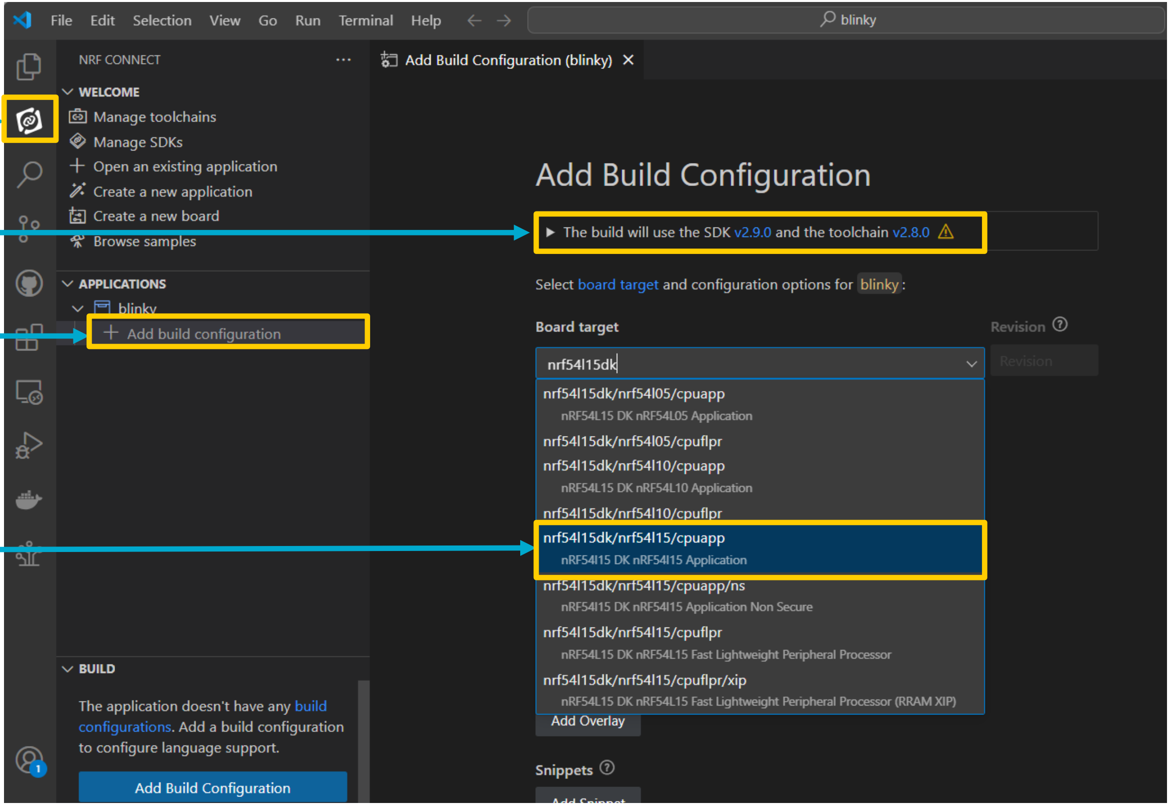
❶ Select the nRF Connect Extension

❸ Select the version of the SDK and toolchain

❷ Add build configuration

❹ Select the target board nRF54l15dk/nrf54l15/cpuapp

- Board nRF54L15DK
- SoC nRF54L15
- Core cpuapp



Add Build Configuration

The build will use the SDK v2.9.0 and the toolchain v2.8.0 ⚠

Select board target and configuration options for blinky:

Board target: nrf54l15dk

- nrf54l15dk/nrf54l05/cpuapp
  nRF54L15 DK nRF54L05 Application
- nrf54l15dk/nrf54l05/cpuflpr
- nrf54l15dk/nrf54l10/cpuapp
  nRF54L15 DK nRF54L10 Application
- nrf54l15dk/nrf54l10/cpuflpr
- nrf54l15dk/nrf54l15/cpuapp
  nRF54l15 DK nRF54l15 Application
- nrf54l15dk/nrf54l15/cpuapp/ns
  nRF54l15 DK nRF54l15 Application Non Secure
- nrf54l15dk/nrf54l15/cpuflpr
  nRF54L15 DK nRF54L15 Fast Lightweight Peripheral Processor
- nrf54l15dk/nrf54l15/cpuflpr/xip
  nRF54L15 DK nRF54L15 Fast Lightweight Peripheral Processor (RRAM XIP)

The application doesn't have any build configurations. Add a build configuration to configure language support.

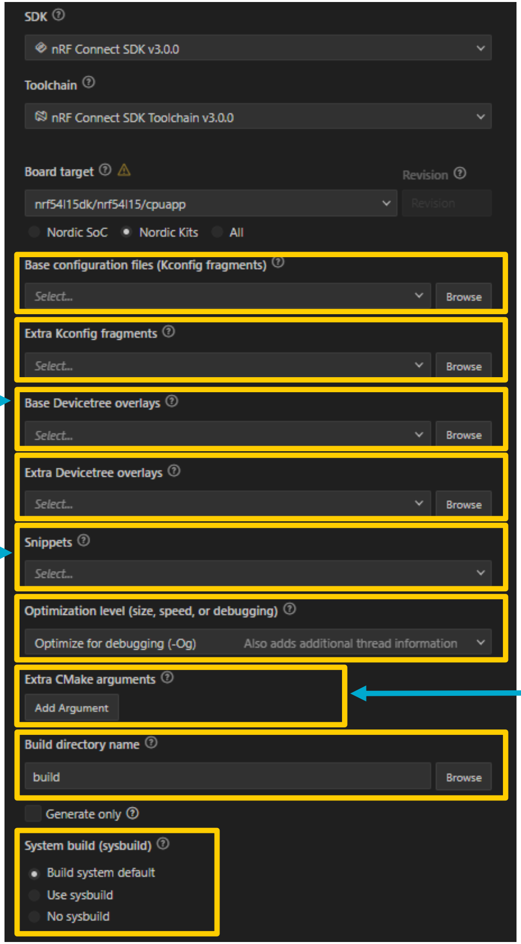
Select Kconfig configuration if empty the **default** is used

Select device tree overlay if empty the **default** is used

Add code snippet common build code

Give a name for the current build directory

Use **Build system default** or **Use sysbuild**



SDK ⓘ

nRF Connect SDK v3.0.0

Toolchain ⓘ

nRF Connect SDK Toolchain v3.0.0

Board target ⓘ ⚠ Revision ⓘ

nrf54l15dk/nrf54l15/cpuapp Revision

○ Nordic SoC ● Nordic Kits ○ All

Base configuration files (Kconfig fragments) ⓘ

Select... Browse

Extra Kconfig fragments ⓘ

Select... Browse

Base Devicetree overlays ⓘ

Select... Browse

Extra Devicetree overlays ⓘ

Select... Browse

Snippets ⓘ

Select...

Optimization level (size, speed, or debugging) ⓘ

Optimize for debugging (-Og) Also adds additional thread information

Extra CMake arguments ⓘ

Add Argument

Build directory name ⓘ

build Browse

□ Generate only ⓘ

System build (sysbuild) ⓘ

● Build system default

○ Use sysbuild

○ No sysbuild

Kconfig overlay added to base Kconfig

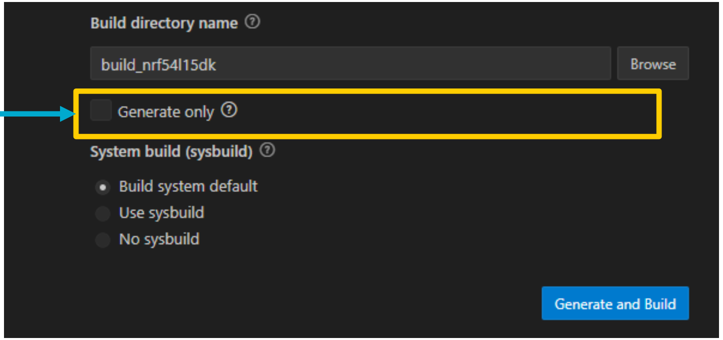
Additional devicetree overlay added to base devicetree

Select **Optimize for debugging**

Add Kconfig fragments and extra Cmake arguments if needed

Setting up build configurations 3/3

Keep box unchecked to generate the configuration and directly build the project.



Build directory name ⓘ

build_nrf54l15dk

□ Generate only ⓘ

System build (sysbuild) ⓘ

● Build system default
○ Use sysbuild
○ No sysbuild

Finally, hit Generate and Build

Tip: Press CTRL+J and select the “Terminal” tab page to view to progress of the building of the configuration

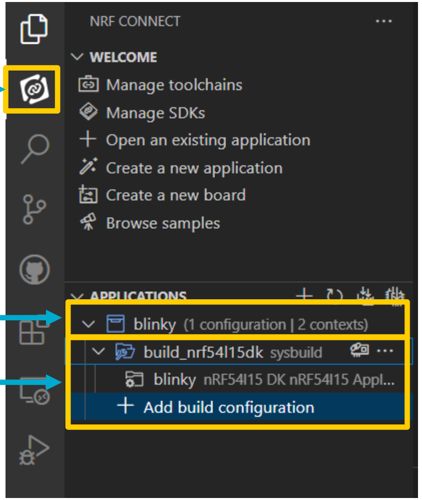
Build completed 1/2

Select the nRF Connect extension

Refers to the “blinky” project in the VC Workspace

Show the available build configurations for this application.

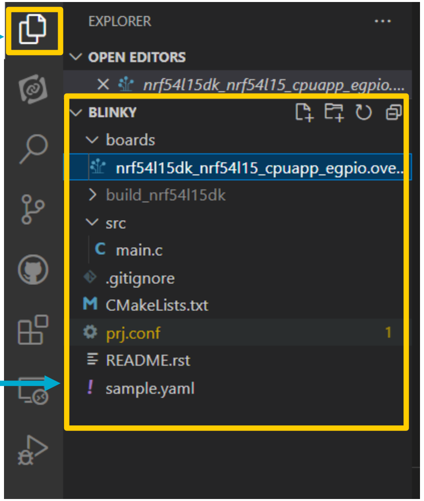
Multiple different builds are possible for a application. For example a debug and a production build.



Build completed 2/2

Select the File explorer extension

Show the available files in with in the application project.



EXPLORER

OPEN EDITORS

nrf54l15dk_nrf54l15_cpuapp_egpio....

BLINKY

boards

nrf54l15dk_nrf54l15_cpuapp_egpio.ove...

build_nrf54l15dk

src

main.c

.gitignore

CMakeLists.txt

prj.conf 1

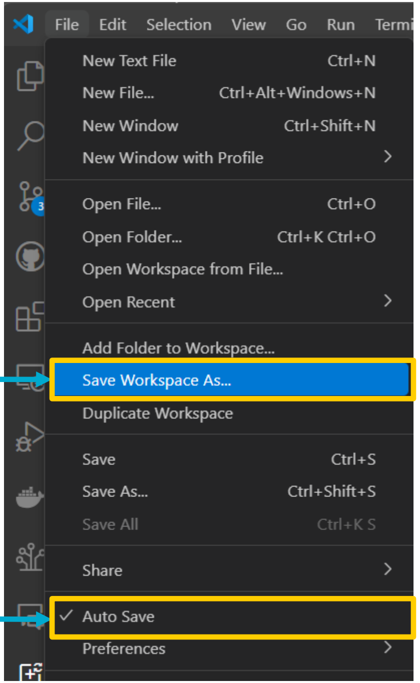
README.rst

sample.yaml

Visual Code – Workspace project

It is always advised to save your application into a Visual Code Workspace. The workspace contains configuration and setting specific of the project.

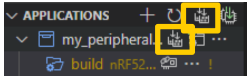
Don't confuse the VC Workspace with a Zephyr Workspace.



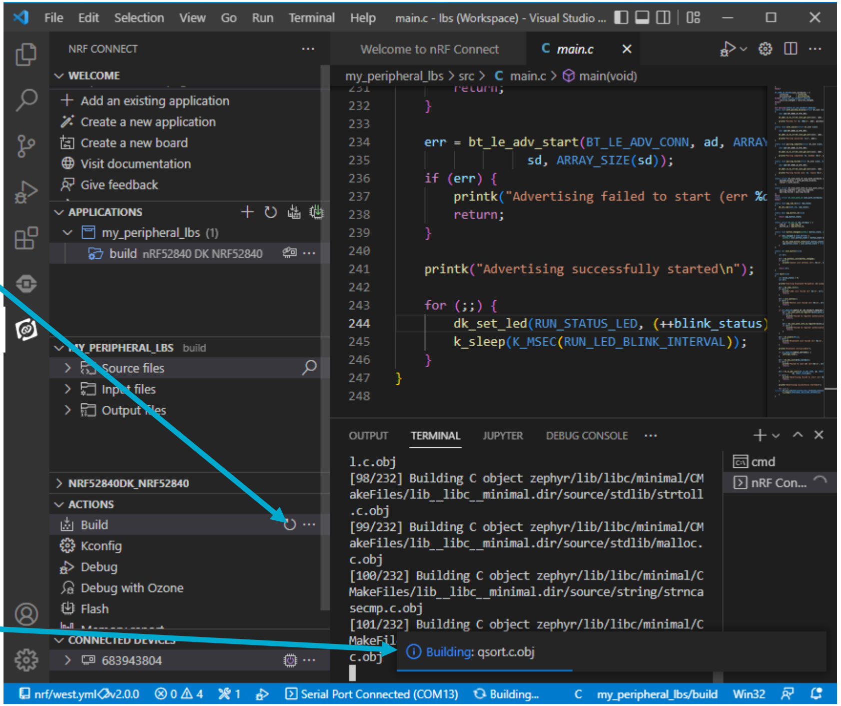
Save your Workspace

Enable Auto-Save

Building your first app

- To (re)build you app click “build” in the Actions view
  - Click here to force a pristine build (a forced clean build)
- If you have several applications or build configurations, you can build all with one click: 

- Building progress can be viewed by clicking



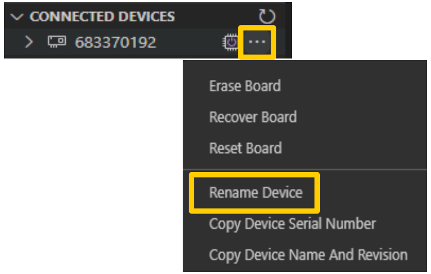
```
        return;
    }

    err = bt_le_adv_start(BT_LE_ADV_CONN, ad, ARRAY
                  sd, ARRAY_SIZE(sd));
    if (err) {
        printk("Advertising failed to start (err %c
        return;
    }

    printk("Advertising successfully started\n");

    for (;;) {
        dk_set_led(RUN_STATUS_LED, (++blink_status)
        k_sleep(K_MSEC(RUN_LED_BLINK_INTERVAL));
    }
}
```

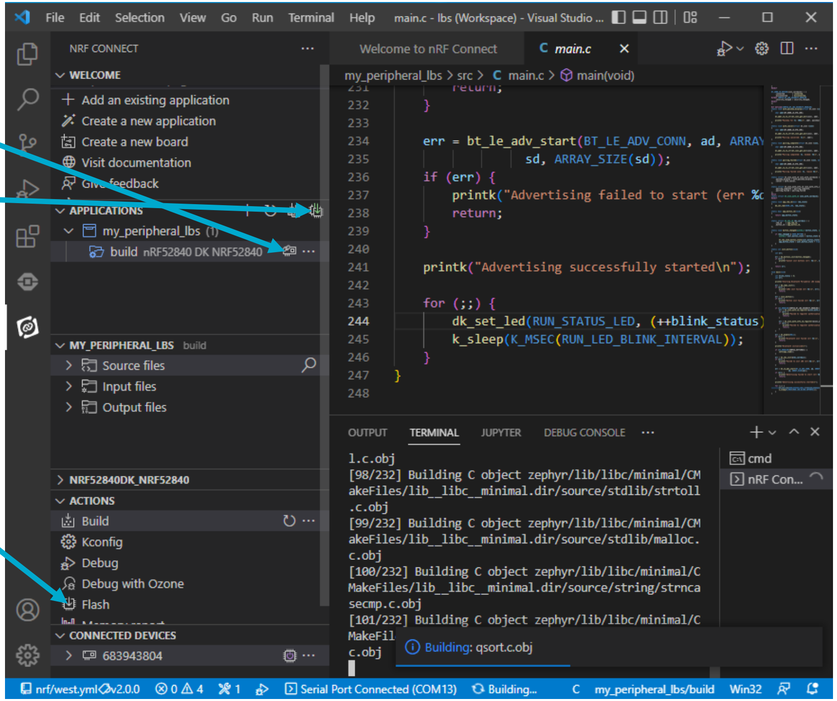
```
l.c.obj
[98/232] Building C object zephyr/lib/libc/minimal/CM
akeFiles/lib__libc__minimal.dir/source/stdlib/strtoll
.c.obj
[99/232] Building C object zephyr/lib/libc/minimal/CM
akeFiles/lib__libc__minimal.dir/source/stdlib/malloc.
c.obj
[100/232] Building C object zephyr/lib/libc/minimal/C
MakeFiles/lib__libc__minimal.dir/source/string/strnca
secmp.c.obj
[101/232] Building C object zephyr/lib/libc/minimal/C
MakeFil
c.obj
```

Building: qsort.c.obj

Flashing connected boards

- Link your build config to a connected board
- And flash all linked boards with one click
- Flash your active build
- Give your board a name:



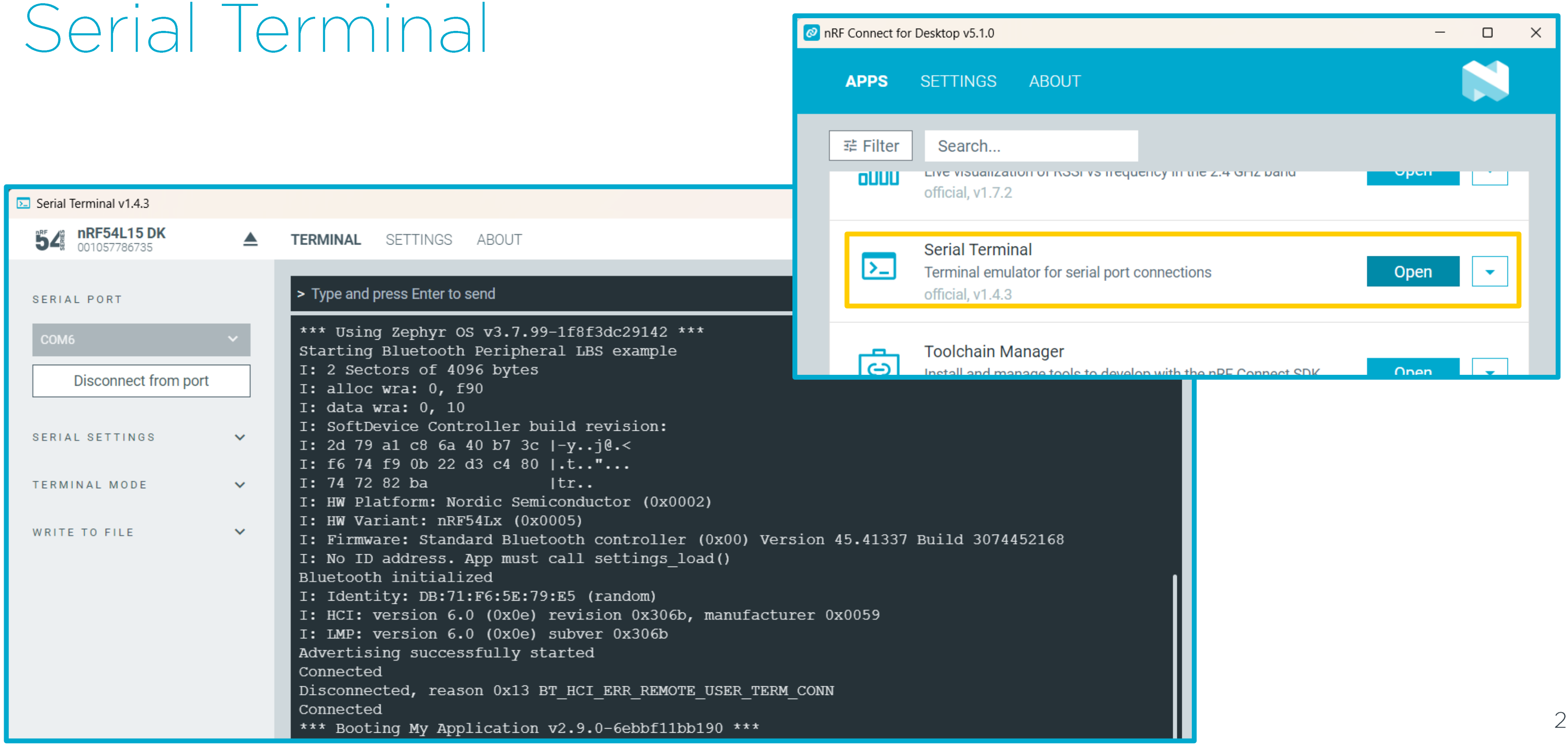


```c
        return;
    }

    err = bt_le_adv_start(BT_LE_ADV_CONN, ad, ARRAY
                  sd, ARRAY_SIZE(sd));
    if (err) {
        printk("Advertising failed to start (err %c
        return;
    }

    printk("Advertising successfully started\n");

    for (;;) {
        dk_set_led(RUN_STATUS_LED, (++blink_status)
        k_sleep(K_MSEC(RUN_LED_BLINK_INTERVAL));
    }
}
```

```
l.c.obj
[98/232] Building C object zephyr/lib/libc/minimal/CM
akeFiles/lib__libc__minimal.dir/source/stdlib/strtoll
.c.obj
[99/232] Building C object zephyr/lib/libc/minimal/CM
akeFiles/lib__libc__minimal.dir/source/stdlib/malloc.
c.obj
[100/232] Building C object zephyr/lib/libc/minimal/C
MakeFiles/lib__libc__minimal.dir/source/string/strnca
secmp.c.obj
[101/232] Building C object zephyr/lib/libc/minimal/C
MakeFil
c.obj
```

Building: qsort.c.obj

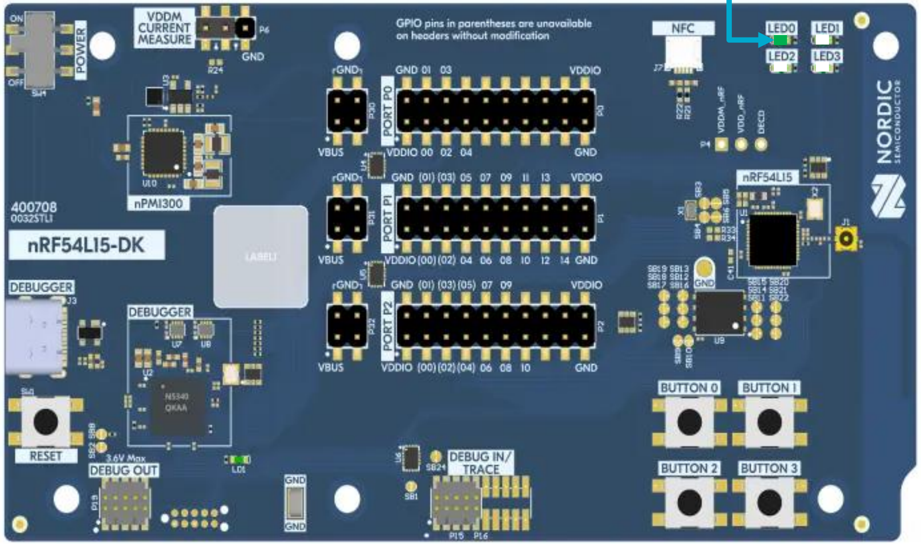
nRF Connect For Desktop Serial Terminal



```
*** Using Zephyr OS v3.7.99-1f8f3dc29142 ***
Starting Bluetooth Peripheral LBS example
I: 2 Sectors of 4096 bytes
I: alloc wra: 0, f90
I: data wra: 0, 10
I: SoftDevice Controller build revision:
I: 2d 79 a1 c8 6a 40 b7 3c |-y..j@.<
I: f6 74 f9 0b 22 d3 c4 80 |.t.."...
I: 74 72 82 ba             |tr..
I: HW Platform: Nordic Semiconductor (0x0002)
I: HW Variant: nRF54Lx (0x0005)
I: Firmware: Standard Bluetooth controller (0x00) Version 45.41337 Build 3074452168
I: No ID address. App must call settings_load()
Bluetooth initialized
I: Identity: DB:71:F6:5E:79:E5 (random)
I: HCI: version 6.0 (0x0e) revision 0x306b, manufacturer 0x0059
I: LMP: version 6.0 (0x0e) subver 0x306b
Advertising successfully started
Connected
Disconnected, reason 0x13 BT_HCI_ERR_REMOTE_USER_TERM_CONN
Connected
*** Booting My Application v2.9.0-6ebbf11bb190 ***
```

The blinky in action

nRF54L15DK with LED0 blinking



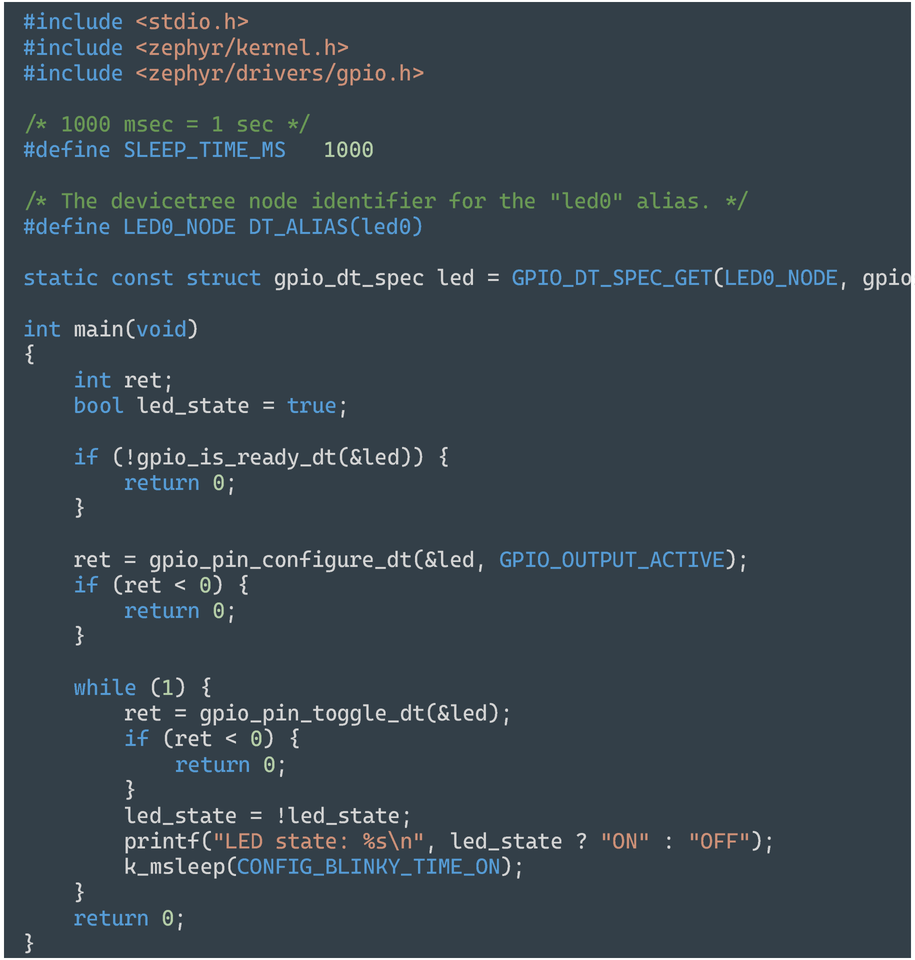
Playing with Blinky

https://academy.nordicsemi.com/topic/exercise-2-1/

Zephyr Sample: Blinky

In the following slides, we will examine the blinky sample line by line to understand how this program works.

The blinky sample comes as part of the nRF Connect SDK and is found at this location:
<ncs>\zephyr\samples\basic\blinky



```c
#include <stdio.h>
#include <zephyr/kernel.h>
#include <zephyr/drivers/gpio.h>

/* 1000 msec = 1 sec */
#define SLEEP_TIME_MS   1000

/* The devicetree node identifier for the "led0" alias. */
#define LED0_NODE DT_ALIAS(led0)

static const struct gpio_dt_spec led = GPIO_DT_SPEC_GET(LED0_NODE, gpios

int main(void)
{
    int ret;
    bool led_state = true;

    if (!gpio_is_ready_dt(&led)) {
        return 0;
    }

    ret = gpio_pin_configure_dt(&led, GPIO_OUTPUT_ACTIVE);
    if (ret < 0) {
        return 0;
    }

    while (1) {
        ret = gpio_pin_toggle_dt(&led);
        if (ret < 0) {
            return 0;
        }
        led_state = !led_state;
        printf("LED state: %s\n", led_state ? "ON" : "OFF");
        k_msleep(CONFIG_BLINKY_TIME_ON);
    }
    return 0;
}
```

Include Modules

The blinky sample uses the following modules of the nRF Connect SDK/Zephyr:

- C Standard library **<stdio.h>** for IO, printf
- Kernel services **<zephyr/kernel.h>** for the sleep function **k_msleep()** or kernel **printk()**
- The generic GPIO interface **<drivers/gpio.h>** for the structure **gpio_dt_spec**, the macros **GPIO_DT_SPEC_GET()**, and the functions **gpio_pin_configure_dt()** and **gpio_pin_toggle_dt()**.

The modules are included through the following include lines:

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

Define the Node Identifier

- The line below uses the devicetree macro **DT_ALIAS()** to get the node identifier symbol **LED0_NODE**, which will represent **node led_0**.

```
/* The devicetree node identifier for the
   "led0" alias. Defined in the .dts file */
#define LED0_NODE DT_ALIAS(led0)
```

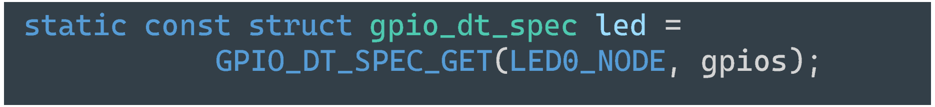
- The **led_0** node is defined in the devicetree of the nRF54L15-DK. **LED0_NODE** is now a source code symbol that represents the hardware for **LED0**.
- The **DT_ALIAS()** macro gets the node identifier from the node's alias name, which as we saw in the Devicetree section, is **led0**.

NOTE:

There are many ways to retrieve the node identifier. The macros **DT_PATH(), DT_NODELABEL(), DT_ALIAS(),** and **DT_INST().** All return the node identifier, based on different input parameters.

Retrieve Device pointer, Pin nb and config flags

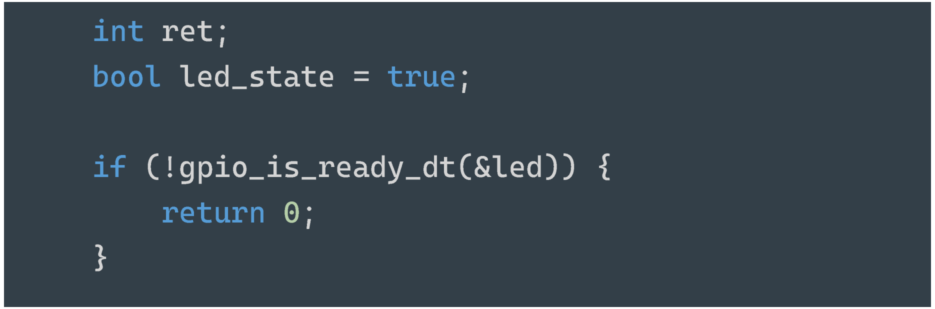
- The macro call **GPIO_DT_SPEC_GET()** returns the structure **gpio_dt_spec led**, which contains the device pointer for node **led_0** as well as the pin number and associated configuration flags.
- The node identifier **LED0_NODE** has this information embedded inside its **gpios** property.
- The second parameter **gpios**, the name of the property containing all this information.



```
static const struct gpio_dt_spec led =
            GPIO_DT_SPEC_GET(LED0_NODE, gpios);
```

Verify that device is ready to use

Now we must pass the device pointer of the device, in this case the led, to gpio_is_ready_dt(), to verify that the device is ready for use.



```
int ret;
bool led_state = true;

if (!gpio_is_ready_dt(&led)) {
    return 0;
}
```

Configure GPIO Pin

The generic GPIO API function **gpio_pin_configure_dt()** is used to configure the GPIO pin

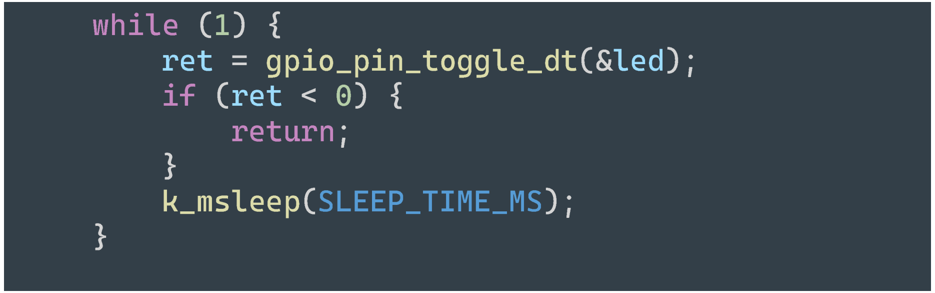
The led pin has an output (**active high**) and initializes it to a logic 1.

```
ret = gpio_pin_configure_dt(&led,
                             GPIO_OUTPUT_ACTIVE);
if (ret < 0) {
    return;
}
```

Continuously toggle the GPIO Pin

Finally, the blinky main function will enter an infinite loop where we continuously toggle the GPIO pin using **gpio_pin_toggle_dt()**.

In every iteration, we are calling the kernel service function **k_msleep()**, which puts the main function to sleep for 1 second



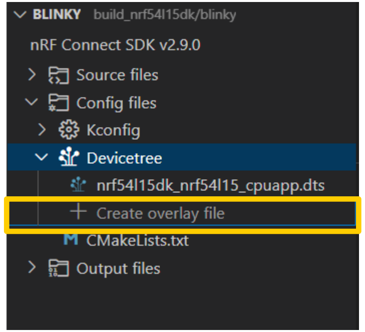
```
while (1) {
    ret = gpio_pin_toggle_dt(&led);
    if (ret < 0) {
        return;
    }
    k_msleep(SLEEP_TIME_MS);
}
```

Update our Blinky

Change the LED Pin

Change the LED using devicetree overlays

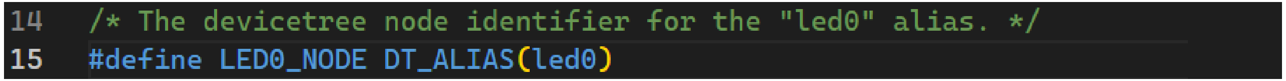
- First step: Create an overlay file
- When the file name is identical to the specified board in the “build configuration” it will be used automatically
- When using the nRF54L15DK the file name is “nrf54l15dk_nrf54l15_cpuapp.overlay”
- When another name is used, it must be added tothe build configuration as a “Extra devicetree overlay” file
- Use VC to create an overlay file
- TIP: Skip the pristine build & first perform the changes in the overlay file, then do a pristine build!



BLINKY build_nrf54l15dk/blinky
nRF Connect SDK v2.9.0
Source files
Config files
Kconfig
Devicetree
nrf54l15dk_nrf54l15_cpuapp.dts
+ Create overlay file
CMakeLists.txt
Output files

Option A: overwrite led0 node in overlay file

- Blinky uses the "led0" node referenced in main.c



```
/* The devicetree node identifier for the "led0" alias. */
#define LED0_NODE DT_ALIAS(led0)
```

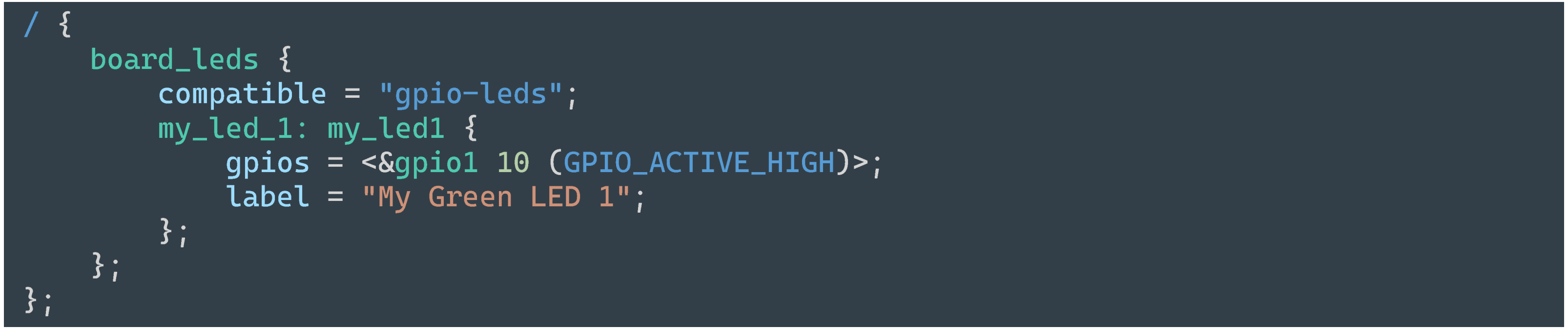
- Overwrite led0 to a different pin (here onboard LED2, add to overlay:

```
&led0 {
    gpios = <&gpio2 7 GPIO_ACTIVE_HIGH>;
};
```

```
nRF54L15-DK - HW Setup:
        LED0 = Port 2, Pin 9
        LED1 = Port 1, Pin 10
        LED2 = Port 2, Pin 7
        LED3 = Port 1, Pin 14
```

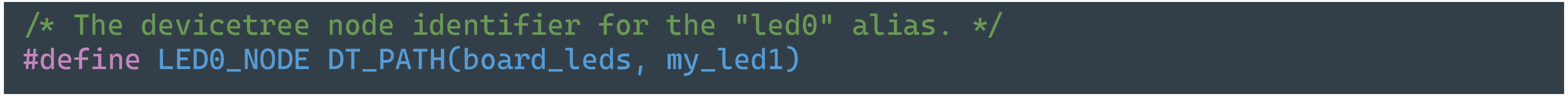
Option B: Create a new DTS node group

- Create the DTS node group "**board_leds**" and add the node **my_led1** with its node label **my_led_1**



```
/ {
    board_leds {
        compatible = "gpio-leds";
        my_led_1: my_led1 {
            gpios = <&gpio1 10 (GPIO_ACTIVE_HIGH)>;
            label = "My Green LED 1";
        };
    };
};
```

- Change devicetree node identifier in **main.c** to new node definition (**my_led1**).



```
/* The devicetree node identifier for the "led0" alias. */
#define LED0_NODE DT_PATH(board_leds, my_led1)
```

Option C: Refer with alias 1/2

- Just as example we could also used DT_ALIAS instead of DT_PATH.
- Add an alias to the board_leds node

```
    aliases {
        led0 = &my_led_1;
    };
```

- This defines an alias to refer led0 now to new defined my_led1
- Change in main.c the node identifier back to the alias led0.

```
/* The devicetree node identifier for the "led0" alias. */
#define LED0_NODE DT_ALIAS(led0)
```

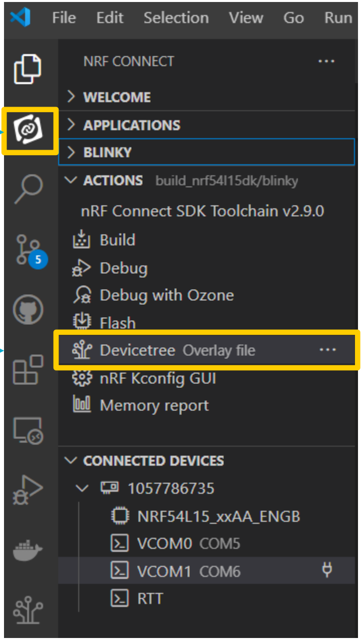
Option C: Refer with alias 2/2

- Node definition with the alias added:

```
/ {
    board_leds {
        compatible = "gpio-leds";
        my_led_1: my_led1 {
            gpios = <&gpio1 10 (GPIO_ACTIVE_LOW)>;
            label = "My Green LED 1";
        };
    };
    aliases {
        led0 = &my_led_1;
    };
};
```

Open visual devicetree editor

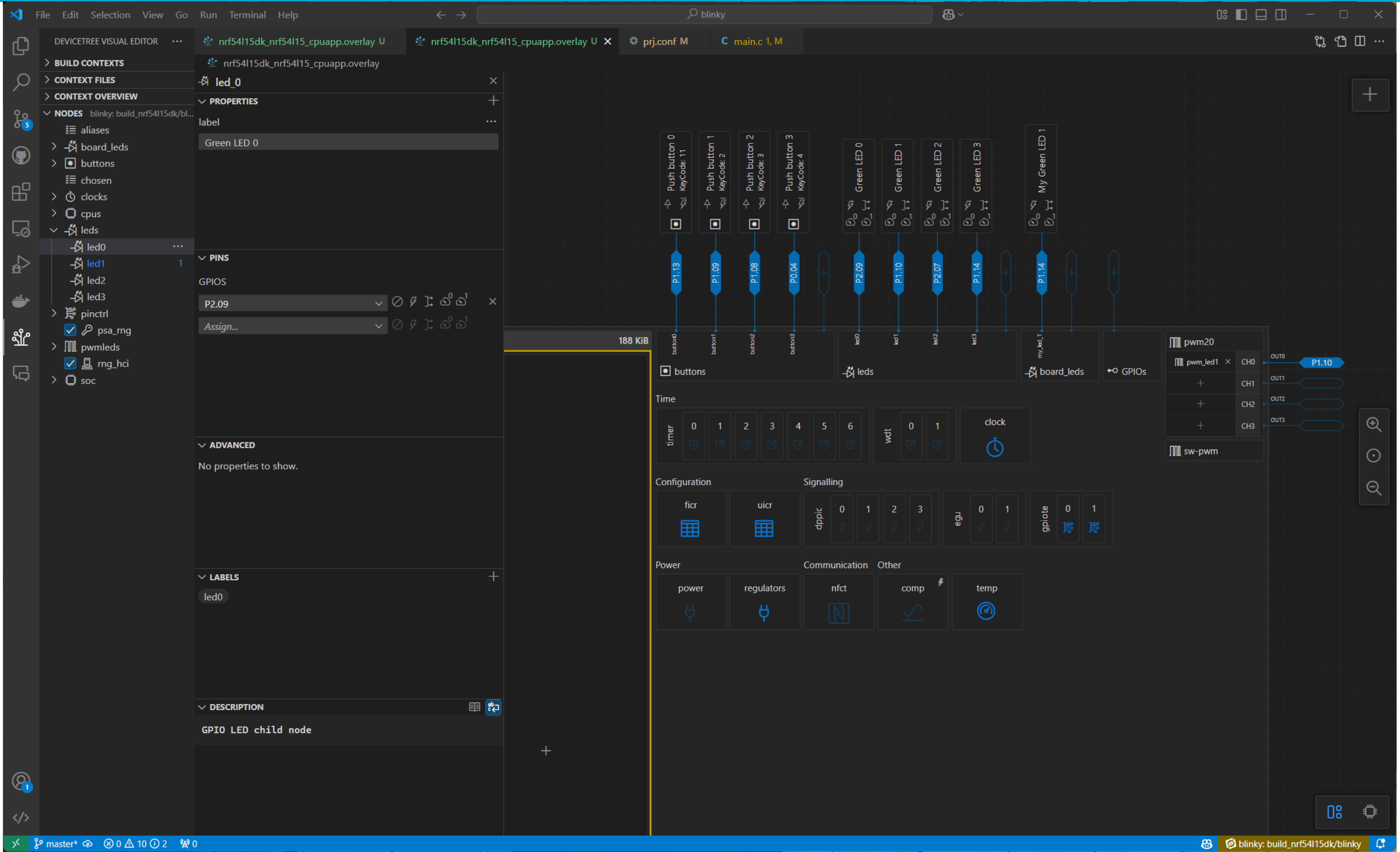
Select the nRF Connect Extension



In the actions section Select “Devicetree” to open the visual editor

Modifies directly the devicetree files.

Be carefull that changes could effect not only this application. Some changes can be written into the board files.



label

Green LED 0

GPIOS

P2.09

Assign...

No properties to show.

led0

GPIO LED child node

NORDIC®
SEMICONDUCTOR

Update our Blinky

Make use of Zephyr work queues

Modify blinky using system work queue

- Instead of blinking the LED in the main loop we move it to a system thread.
- First step is to create a delayable work queue

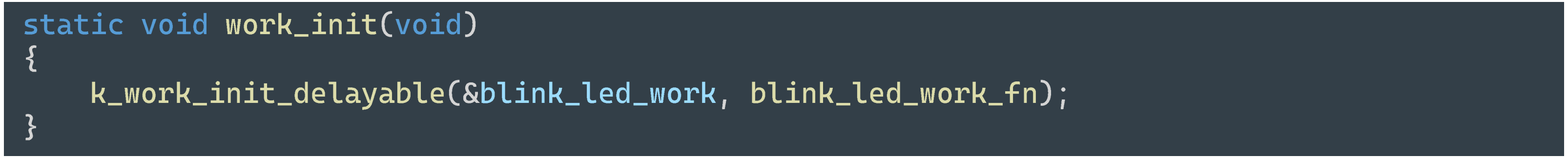
```
static struct k_work_delayable blink_led_work;
```

- Create a work_queue handler function

```
static void blink_led_work_fn(struct k_work *work)
{
    int ret;
    ret = gpio_pin_toggle_dt(&led);
    if (ret < 0) {
        return;
    }
    k_work_schedule(&blink_led_work, K_MSEC(CONFIG_BLINKY_TIME_ON));
}
```

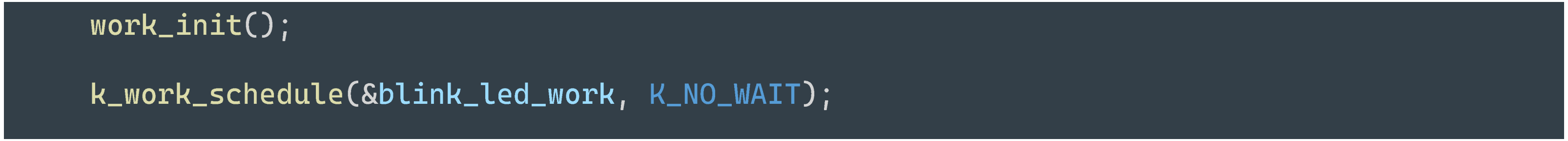
Modify blinky using system work queue

- Create function for initializing the work queues



```
static void work_init(void)
{
    k_work_init_delayable(&blink_led_work, blink_led_work_fn);
}
```

- Call the init routine in **main.c** remove **while(1)** loop and add schedule the delayable work queue immediately with option **K_NO_WAIT**



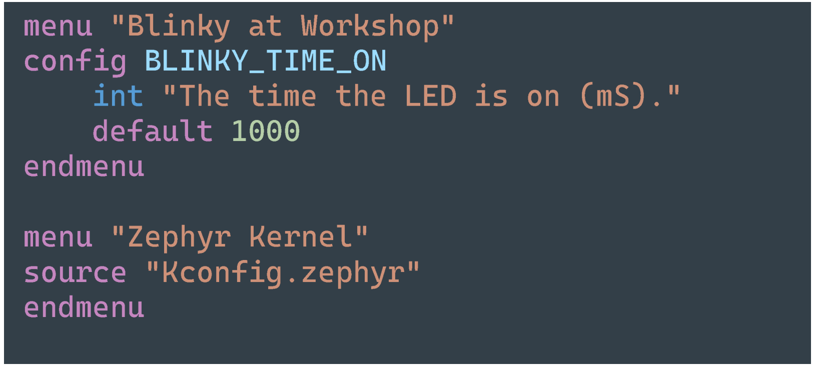
```
    work_init();

    k_work_schedule(&blink_led_work, K_NO_WAIT);
```

Update our Blinky

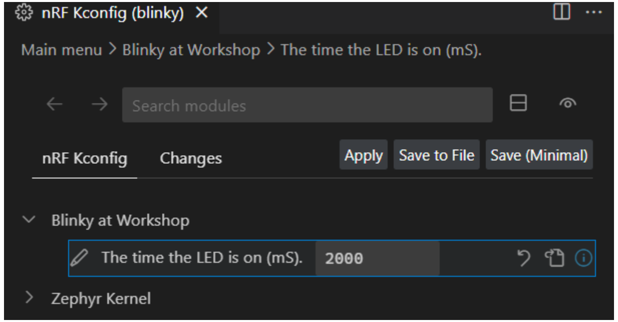
Adding kernel definitions

Build a kernel configuration menu

- Insert a new file in the root of the “**blinky**” project with the name “**Kconfig**”
- This defines **CONFIG_BLINKY_ON_TIME** with a default value of “**1000**”
- Select the menu “nRF Kconfig GUI” in the “ACTIONS” section in the nRF Connect Ext.



```
menu "Blinky at Workshop"
config BLINKY_TIME_ON
    int "The time the LED is on (mS)."
    default 1000
endmenu

menu "Zephyr Kernel"
source "Kconfig.zephyr"
endmenu
```



Main menu > Blinky at Workshop > The time the LED is on (mS).

Blinky at Workshop
The time the LED is on (mS). 2000
Zephyr Kernel

Add to application: edit manually

- Within our “prj.conf” we can modify the value CONFIG_BLINKY_ON_TIME
- Add the following into the “prj.conf” file in your blinky project.

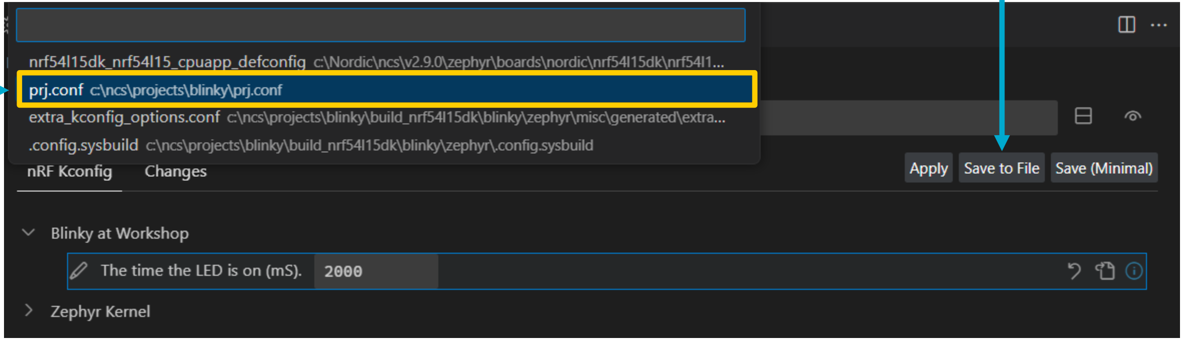
```
CONFIG_BLINKY_TIME_ON=2000
```

- Now replace in our main.c the wait (while(1)) loop with this definition:

```
k_msleep(CONFIG_BLINKY_TIME_ON);
```

Add to application: nRF Kconfig GUI

Select Save to file to make the changes to a Kconfig file



nrf54l15dk_nrf54l15_cpuapp_defconfig c:\Nordic\ncs\v2.9.0\zephyr\boards\nordic\nrf54l15dk\nrf54l1...
prj.conf c:\ncs\projects\blinky\prj.conf
extra_kconfig_options.conf c:\ncs\projects\blinky\build_nrf54l15dk\blinky\zephyr\misc\generated\extra...
.config.sysbuild c:\ncs\projects\blinky\build_nrf54l15dk\blinky\zephyr\.config.sysbuild

Blinky at Workshop
The time the LED is on (mS). 2000
Zephyr Kernel

Select Kconfig file to make modify the changed values.
Use the prj.conf file of the current application.

Selecting the nrf54l15dk_nrf54l15_cpuapp_defconfig will modify the SDK for all applications.
When written in extra_kconfig_options.conf the modification is removed with pristine build