CEC Continuing Education Center

DesignNews

Mastering Zephyr RTOS

DAY 2 : Build Systems, Kconfig, and Device Tree

Sponsored by

DigiKey

BENINGO EMBEDDED GROUP

informa markets

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.

CEC Continuing Education Center

BENINGO EMBEDDED GROUP

DigiKey

THE SPEAKER

Jacob Beningo

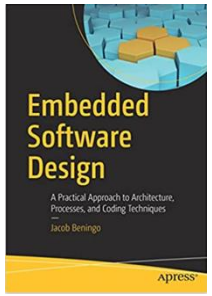
Jacob@beningo.com

jacobbeningo

Beningo Embedded Group – CEO / Founder

Focus: Software Architecture, Processes, and Dev Skills

At Beningo Embedded Group, we believe everyone deserves the skills to confidently advance their careers, meet deadlines, and deliver quality embedded systems. We provide modern strategies, insights, and hands-on training to equip developers and teams with the tools they need to succeed.



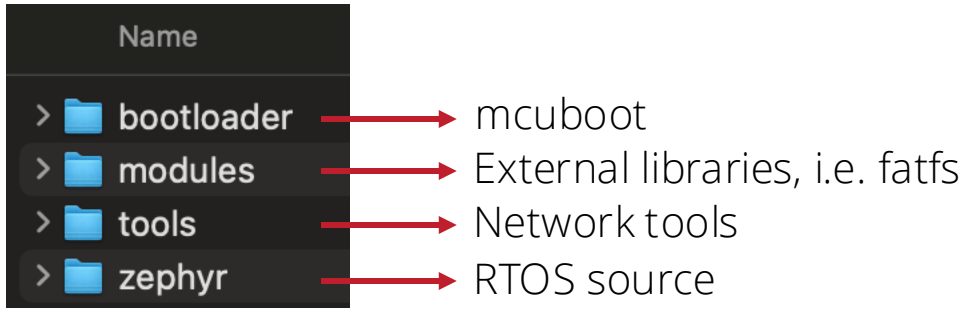
EMBEDDED SOFTWARE ACADEMY BY BENINGO

Visit **www.beningo.com** to learn more

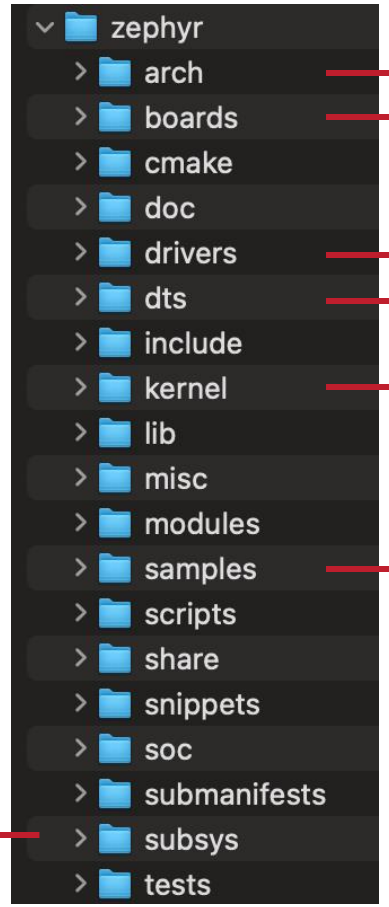
Build Systems

01

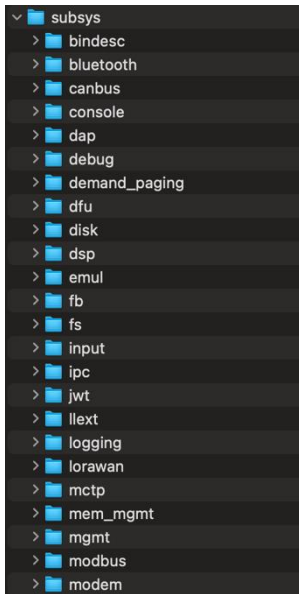
Zephyr Project Organization

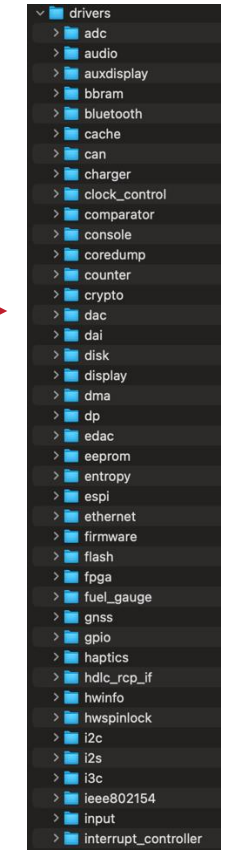


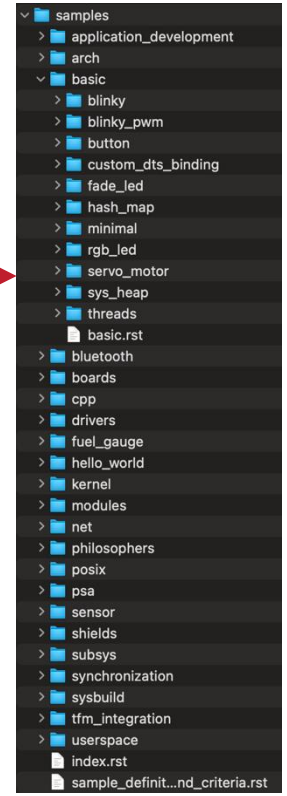
mcuboot
External libraries, i.e. fatfs
Network tools
RTOS source



CPUs
Dev Boards
Device Drivers
Device Tree
Kernel source
Examples
Software components







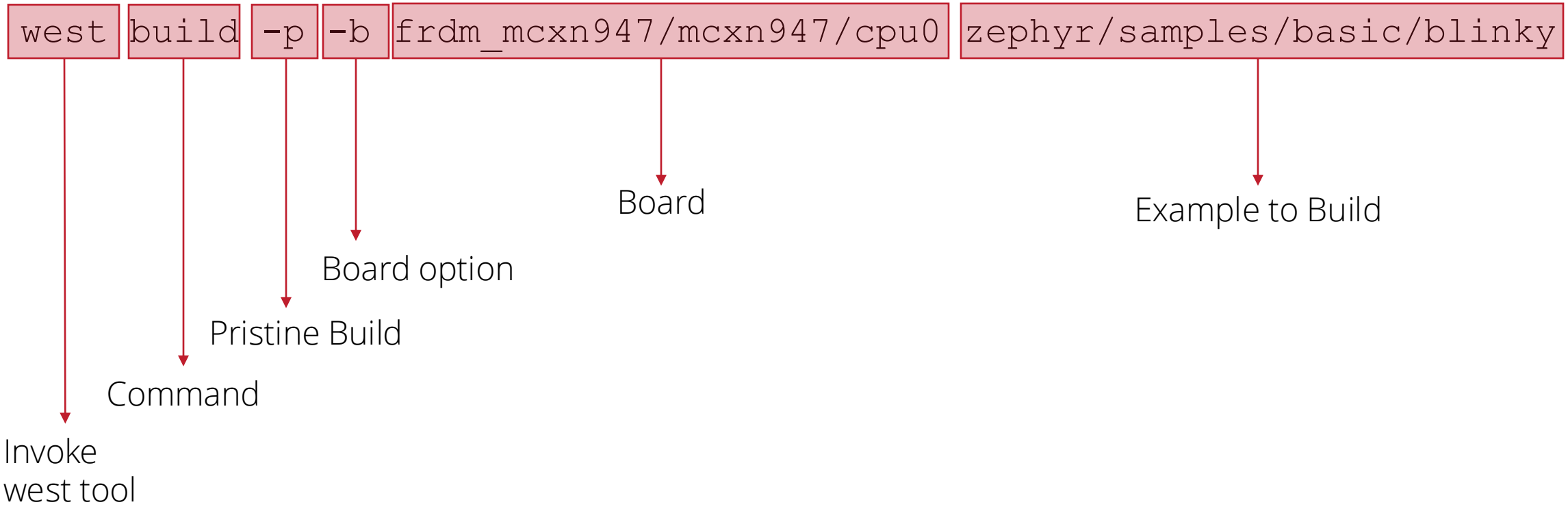
Zephyr Build System

The Zephyr Build System is based on CMake and Python, managed by the west meta-tool. It orchestrates configuration, dependency management, and build generation across a modular, multi-repo project.

Key Components:

- CMake: Core build system generator (creates Makefiles or Ninja files)
- Kconfig: Handles project configuration through menuconfig, guiconfig, or prj.conf
- West: A Python-based meta-tool that wraps CMake and manages the workspace

Building Blinky using West



Audience POLL Question

What is the primary role of the west tool in the Zephyr build system?

a) It compiles source files directly using GCC
b) It manages Zephyr's device tree overlays
c) It acts as a meta-tool to manage repositories, build configuration, and project structure
d) It replaces CMake as the build system backend

KConfig

02

What is Kconfig?

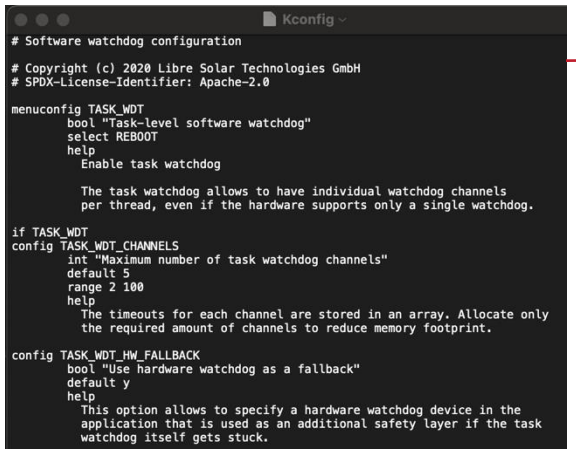
Kconfig is the configuration system used to manage compile-time options for the kernel, subsystems, device drivers, and application settings.

It's based on the same system originally used in the Linux kernel.

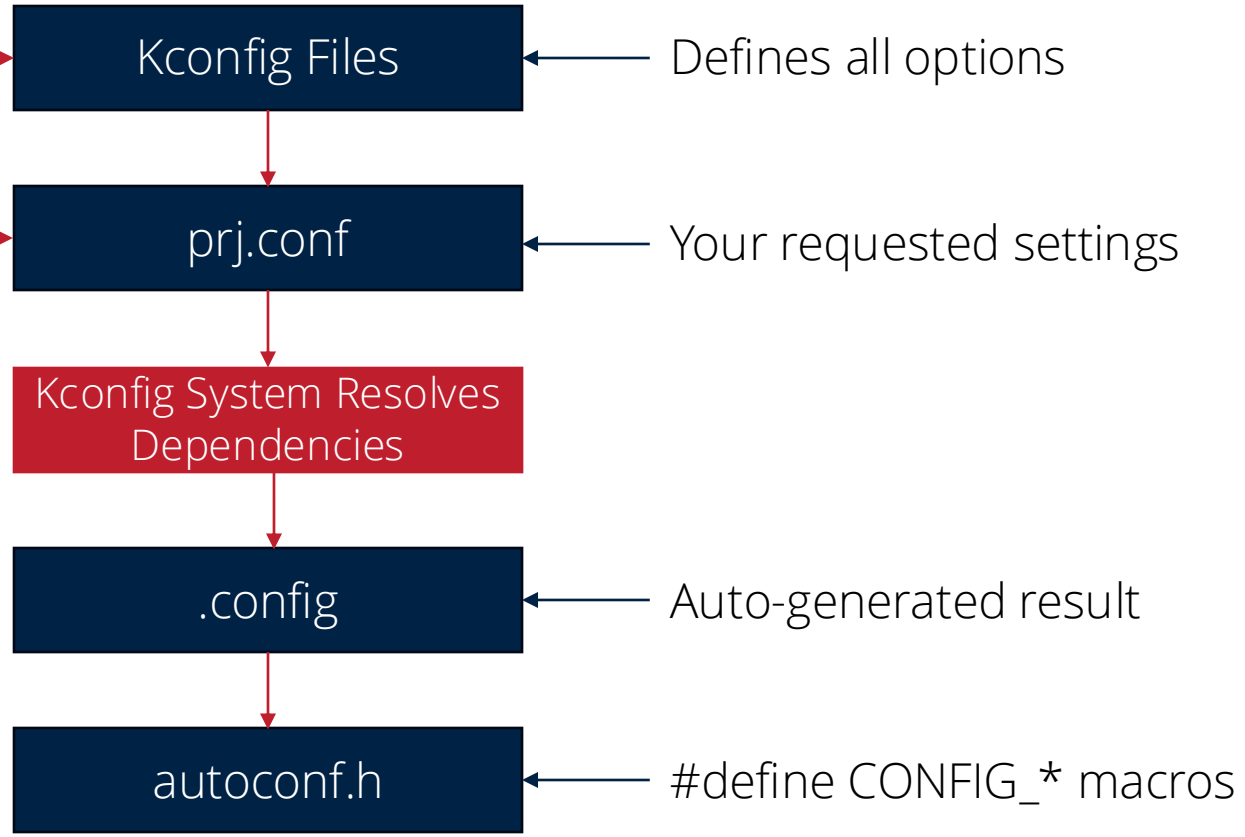
Purpose of Kconfig in Zephyr:

- Enables developers to select features, tune options, and control dependencies at build time
- Helps create configurable, modular, and scalable firmware projects
- Works hand-in-hand with menuconfig, guiconfig, and prj.conf

How it works

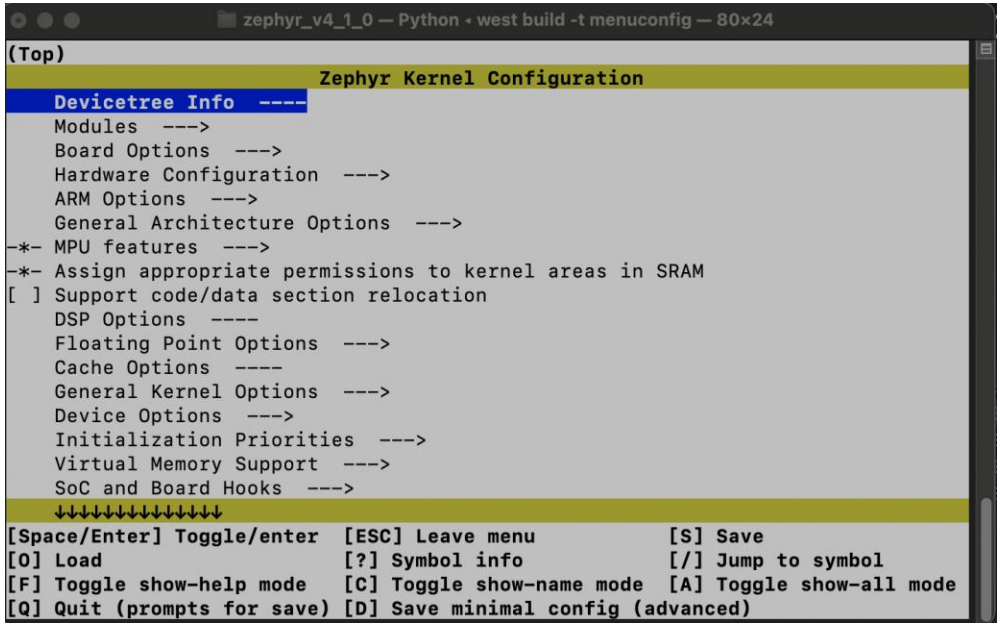



```
# Enable the Task Watchdog subsystem
CONFIG_TASK_WDT=y

# Automatically start the watchdog at boot (optional)
CONFIG_TASK_WDT_AUTO_START=y

# Number of tasks that can register with the watchdog
CONFIG_TASK_WDT_MAX_SUBSCRIBERS=3

# Watchdog timeout (in milliseconds)
CONFIG_TASK_WDT_TIMEOUT_MS=1000
```



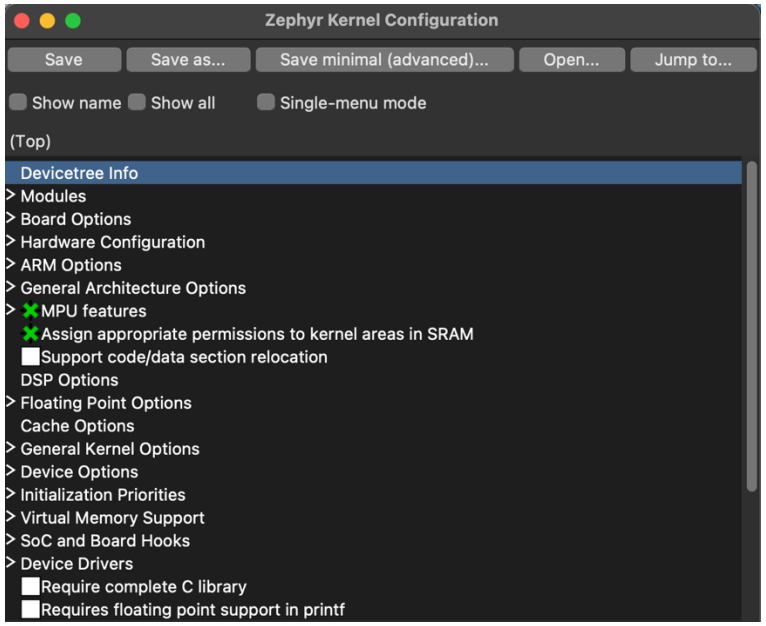
Configuration Tools

menuconfig



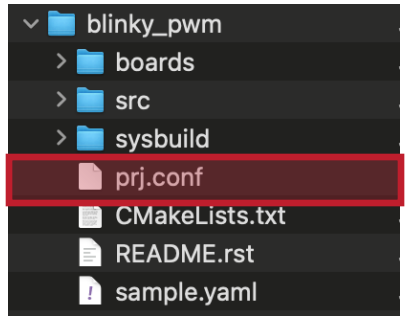
west build -b <board> zephyr/samples/basic/blinky
west build -t menuconfig

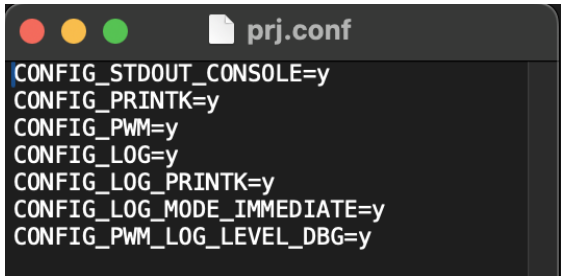
guiconfig



west build -b <board> zephyr/samples/basic/blinky
west build -t guiconfig

prj.conf





Audience POLL Question

What is the primary role of Kconfig files?

a) To store runtime configuration values that the application can change during execution
b) To define and manage compile-time configuration options, including defaults, dependencies, and menus
c) To manage device tree overlays and hardware abstraction for board support
d) To specify linker sections for placing kernel and driver symbols

Device Tree

03

What is the Devicetree

A devicetree is a hierarchical data structure primarily used to describe hardware. Zephyr uses devicetree in two main ways:

- to describe hardware to the Device Driver Model
- to provide the hardware's initial configuration

Why Zephyr Uses a Device Tree

- To decouple hardware configuration from application logic
- To support many boards and SoCs with a common driver infrastructure
- To automatically generate configuration macros your firmware can use at compile time

How it works

```
/dts-v1/;

#include "frdm_mcxn947_mcxn947_cpu0.dtsi"

/ {
        model = "NXP FRDM_N94 board";
        compatible = "nxp,mcxn947", "nxp,mcx";
};
```

```
#include <nxp/nxp_mcxn94x.dtsi>
#include "frdm_mcxn947.dtsi"

/ {
        cpus {
                /delete-node/ cpu@1;
        };

        chosen {
                zephyr,sram = &sram0;
                zephyr,flash = &flash;
                zephyr,flash-controller = &fmu;
                zephyr,code-partition = &slot0_partition;
                zephyr,uart-mcumgr = &flexcomm4_lpuart4;
                zephyr,console = &flexcomm4_lpuart4;
                zephyr,shell-uart = &flexcomm4_lpuart4;
                zephyr,canbus = &flexcan0;
                zephyr,code-cpu1-partition = &slot1_partition;
        };

        aliases{
                watchdog0 = &wwdt0;
                pwm-0 = &flexpwm1_pwm0;
                pwm-1 = &sc_timer;
                rtc = &rtc;
        };
};
```

```
&arduino_serial {
        status = "okay";

        modbus0 {
                compatible = "zephyr,modbus-serial";
                status = "okay";
                de-gpios = <&arduino_header 15 GPIO_ACTIVE_LOW>;        /* D9 */
        };
};
```

Board DTS File
(frdm_mcxn947_mcxn947_cpuo.dts)

SoC Includes
(frdm_mcxn947_mcxn947_cpu0.dtsi)

Application Overlay
(app.overlay)

Devicetree Compiler
(Build step using dtc + CMake)

Flattened DT Blob (.dtb)
+ Auto-generated C headers

Auto-generated Macros
(devicetree_generated.h)

Application / Driver C Source Code

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

/*
 * A build error on this line means your board is unsupported.
 * See the sample documentation for information on how to fix this.
 */
static const struct gpio_dt_spec led = GPIO_DT_SPEC_GET(LED0_NODE, gpios);
```

Audience POLL Question

What is the primary role of the Device Tree in a Zephyr application?

a) To configure runtime hardware settings that can be changed during program execution
b) To describe the hardware layout and peripheral connections at build time, separate from application code
c) To manage memory allocation and thread priorities for the RTOS
d) To define file system structures and mount points for storage devices

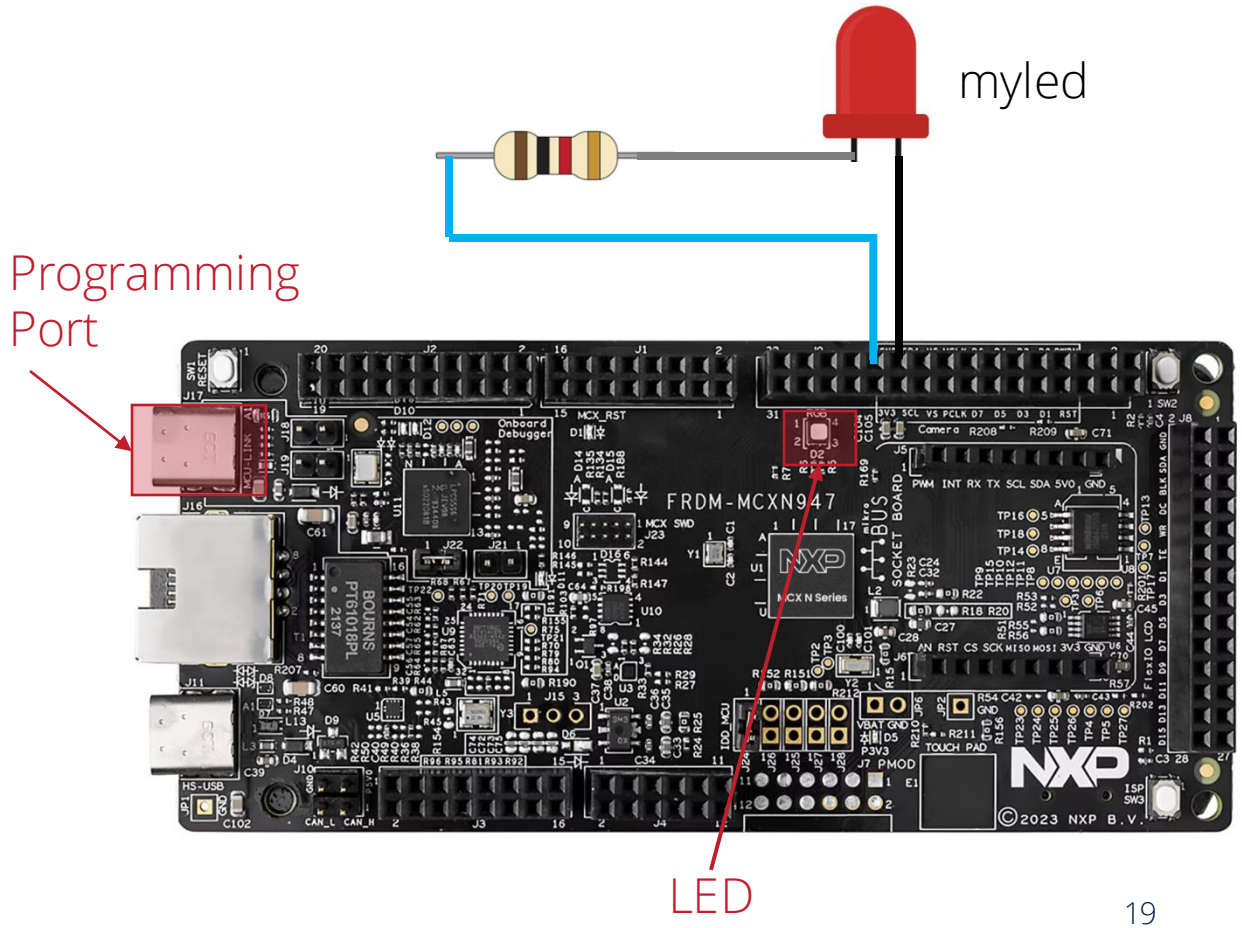
03

Hands-On Example: Blinky

Goals and Hardware Setup

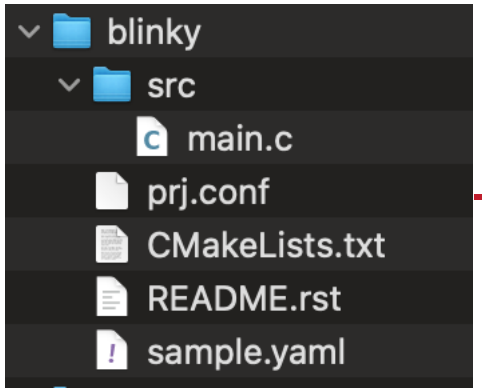
The purpose of this example is to modify the samples/basic/blinky example to:

- Blink the default red LED
- Blink an added LED on GPIO1, pin 22
- Build the project successfully
- Deploy the project with west



Adding an Overlay

sample/basic/blinky



Add app.overlay

New leds node

Use the gpio-leds binding

Node Label (&my_led)

name

Port and Pin

LED On State

Convenient Shortcuts

app.overlay — Edited

```
/ {
    leds {
        compatible = "gpio-leds";
        my_led: my_led {
            gpios = <&gpio1 22 GPIO_ACTIVE_LOW>;
            label = "My Custom LED";
        };
    };
    aliases {
        myled = &my_led;
    };
};
```

I can now write hardware-independent code using:
#define MYLED_NODE DT_ALIAS(myled)

Modify the Blinky Application

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

#define SLEEP_TIME_MS 1000

#define LED0_NODE  DT_ALIAS(led0)
#define MYLED_NODE DT_ALIAS(myled)

static const struct gpio_dt_spec led0   = GPIO_DT_SPEC_GET(LED0_NODE, gpios);
static const struct gpio_dt_spec my_led = GPIO_DT_SPEC_GET(MYLED_NODE, gpios);
```

Creates a structure with the device specs:

```
static const struct gpio_dt_spec my_led = {
    .port = <pointer to GPIO1 device>,
    .pin = 22,
    .dt_flags = GPIO_ACTIVE_LOW
};
```

Searches DT for alias and retrieves the node ID

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

    if (!gpio_is_ready_dt(&led0)) {
        printk("led0 device not ready\n");
        return 0;
    }

    if (!gpio_is_ready_dt(&my_led)) {
        printk("myLED device not ready\n");
        return 0;
    }

    ret = gpio_pin_configure_dt(&led0, GPIO_OUTPUT_ACTIVE);
    if (ret < 0) {
        printk("Failed to configure led0\n");
        return 0;
    }

    ret = gpio_pin_configure_dt(&my_led, GPIO_OUTPUT_ACTIVE);
    if (ret < 0) {
        printk("Failed to configure myLED\n");
        return 0;
    }

    while (1) {
        gpio_pin_set_dt(&led0, led_state);
        gpio_pin_set_dt(&my_led, led_state);

        printk("Blink! LED0 and myLED state: %s\n", led_state ? "ON" : "OFF");

        led_state = !led_state;
        k_msleep(SLEEP_TIME_MS);
    }

    return 0;
}
```

Init / Validate DT

LED App

Build and Deploy

Build the application:

west build -p -b frdm_mcxn947/mcxn947/cpu0 zephyr/samp

```
-- Zephyr version: 4.1.0 (/Users/jacobbeningo/NXP/ZephyrMCUX/zephyr_v4_1_0/zephyr), build: v4.1.0
[143/143] Linking C executable zephyr/zephyr.elf
Memory region         Used Size  Region Size  %age Used
           FLASH:       24294 B         2 MB      1.16%
             RAM:        4312 B       320 KB      1.32%
           SRAM1:           0 GB        96 KB      0.00%
        IDT_LIST:           0 GB        32 KB      0.00%
Generating files from /Users/jacobbeningo/NXP/ZephyrMCUX/zephyr_v4_1_0/build/zephyr/zephyr.elf for board: frdm_mcxn947
```

Deploy the application:

west flash

```
Blink! LED0 and myLED state: ON
Blink! LED0 and myLED state: OFF
Blink! LED0 and myLED state: ON
Blink! LED0 and myLED state: OFF
Blink! LED0 and myLED state: ON
Blink! LED0 and myLED state: OFF
Blink! LED0 and myLED state: ON
Blink! LED0 and myLED state: OFF
Blink! LED0 and myLED state: ON
Blink! LED0 and myLED state: OFF
```

Next Steps

04

Going Further

Download the extra resources:

- https://beningo.short.gy/jVGeiDNZephyrLP

Zephyr Documentation:

- Kconfig Documentation
- Devicetree Documentation

Getting Started with Zephyr RTOS: Hello Blinky! (Webinar)

Downloadable Resources:

- West Cheat Sheet (High-Resolution)
- RTOS Performance Guide
- Application Code Examples
- Zephyr Docker Container
- VS Code Debug Launch Script

Additional Resources

Please consider the resources below:

- Jacob's Blogs
- Jacob's CEC courses
- Embedded Software Academy

- Embedded Bytes Newsletter
  - http://bit.ly/1BAHYXm

Consulting | Coaching | Training

www.beningo.com

Next Steps

- ✓ Welcome to Zephyr RTOS
- ✓ Build Systems, Kconfig, and Device Tree
- Threads, Scheduling, and RTOS Primitives
- Drivers, Peripherals, and Customization
- Debugging, Logging, and Best Practices

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

DigiKey

BENINGO EMBEDDED GROUP

informa markets