CEC Continuing Education Center

DesignNews

Mastering Zephyr RTOS

DAY 4 : Drivers, Peripherals, and Customization

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

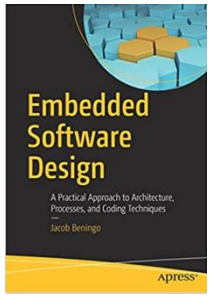
Jacob@beningo.com

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

Zephyr's Driver Model

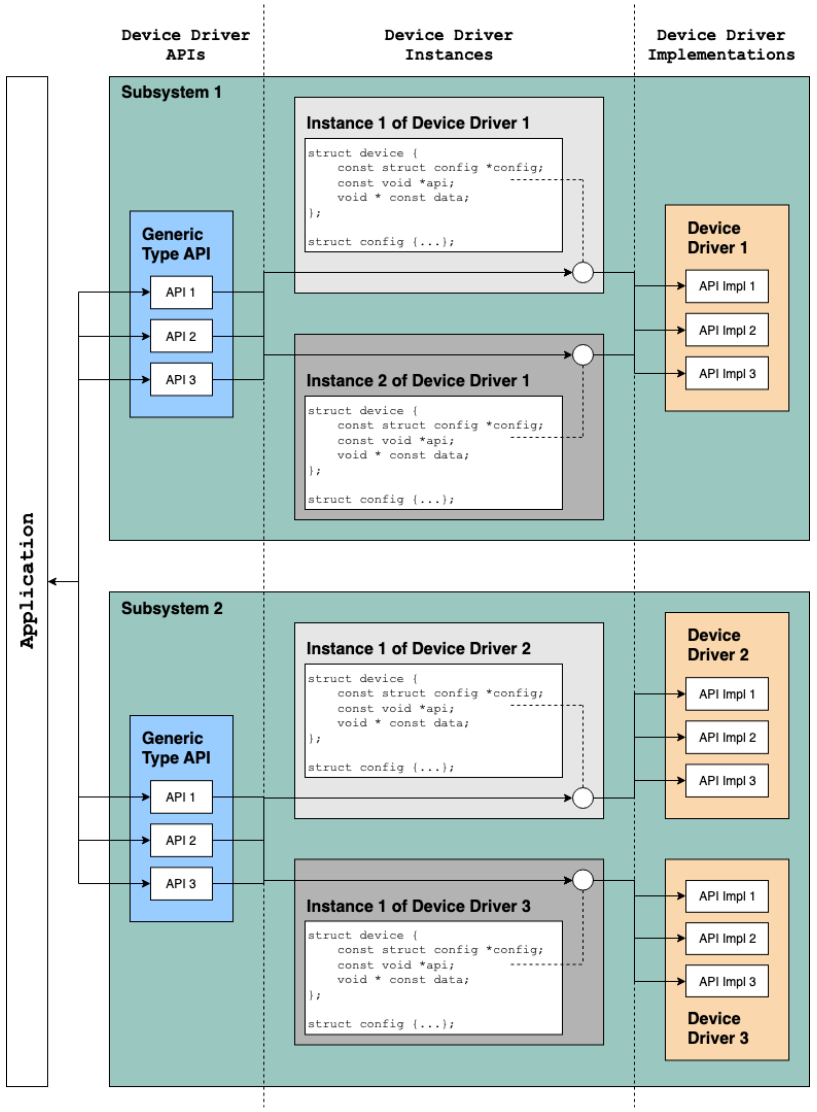
03

The Device Driver Model

The Zephyr device model provides a consistent device model for configuring system drivers.

The device model is responsible for initializing all the drivers configured into the system.

Most drivers implement a device-independent subsystem API. Applications can simply program to that generic API, and application code is not specific to any particular driver implementation.



Standard Drivers

Device drivers that are present on all supported board configurations are listed below.

- Interrupt controller: This device driver is used by the kernel's interrupt management subsystem.
- Timer: This device driver is used by the kernel's system clock and hardware clock subsystem.
- Serial communication: This device driver is used by the kernel's system console subsystem.
- Entropy: This device driver provides a source of entropy numbers for the random number generator subsystem.

What makes up a driver?

- Driver API: Public interface exposed to applications and subsystems (e.g., gpio_pin_set_dt())
- Driver Implementation: SoC-specific logic tied to peripheral registers and HAL
- Devicetree Bindings: Describe hardware config and link it to the driver at build time
- Kconfig Entries: Enable or configure the driver in the build system
- Init Function (DEVICE_DT_DEFINE): Registers the driver during boot with priority, init level, and state

Audience POLL Question

In Zephyr's driver model, what role does the subsystem API layer (e.g., gpio.h, uart.h) play?

a) It initializes hardware peripherals during system boot.
b) It maps Devicetree nodes to hardware register addresses.
c) It defines standard interfaces that application code uses to interact with drivers.
d) It configures pinmux and clock trees for the underlying SoC.

Serial Communication printk()

03

Blinky Serial Interface Configuration

- The blinky application has no obvious serial configuration, yet, we get the following output:

```
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
```

prj.conf

```
CONFIG_GPIO=y
```

```
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;
```

How is the serial console enabled?

boards/nxp/frdm_mcxn947/frdm_mcxn947_mcxn947_cpu0_defconfig

frdm_mcxn947_mcxn947_cpu0_defconfig

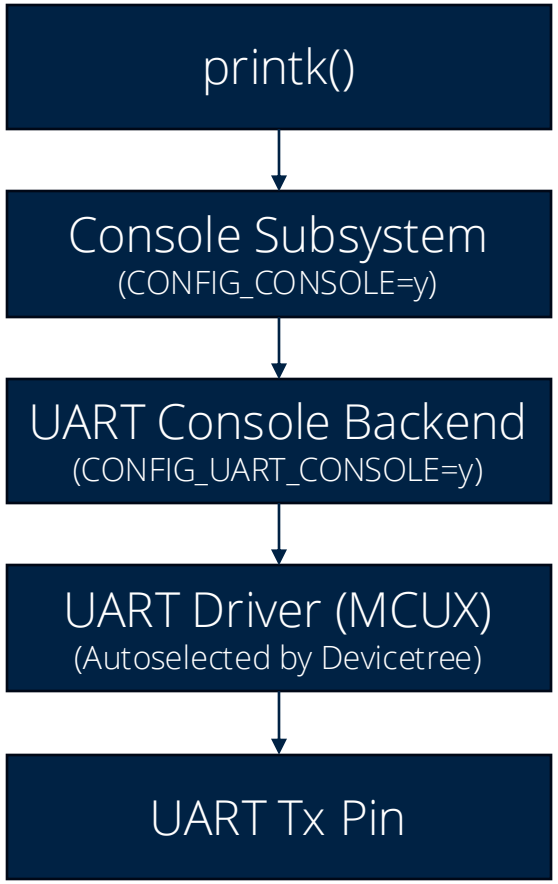
```
#
# Copyright 2024 NXP
#
# SPDX-License-Identifier: Apache-2.0
#

CONFIG_CONSOLE=y
CONFIG_UART_CONSOLE=y
CONFIG_SERIAL=y
CONFIG_UART_INTERRUPT_DRIVEN=y
CONFIG_GPIO=y

CONFIG_ARM_MPU=y
CONFIG_HW_STACK_PROTECTION=y

# Enable TrustZone-M
CONFIG_TRUSTED_EXECUTION_SECURE=y
```

frdm_mcxn947_mcxn947_cpu0.dtsi

console — Done — Replace

```
/ {
        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;
        };
};
```

How it works



Map to a different UART

app.overlay

```
/ {
    chosen {
        zephyr,console = &flexcomm2_lpuart2;
        zephyr,shell-uart = &flexcomm2_lpuart2;
        zephyr,uart-mcumgr = &flexcomm2_lpuart2;
    };

    leds {
        compatible = "gpio-leds";

        my_led: my_led {
            gpios = <&gpio1 22 GPIO_ACTIVE_LOW>;
            label = "My Custom LED";
        };
    };

    aliases {
        myled = &my_led;
    };
};
```

Audience POLL Question

In Zephyr, how can you redirect printk() output to a different UART peripheral (e.g., flexcomm2_lpuart2)?

a) Set CONFIG_UART_REDIRECT=y in prj.conf
b) Update the /aliases section in the board's .dts file
c) Modify the chosen node in app.overlay to point zephyr,console to the desired UART
d) Call uart_redirect(flexcomm2_lpuart2) in main()

The Console Shell

03

The Shell

The Zephyr Shell is a modular, configurable command-line interface that enables developers to control, inspect, and interact with their embedded application through a terminal connection.

- Interactive prompt (shell>) over UART or USB.
- Built-in commands for inspecting devices, threads, uptime, logs, etc.
- Custom commands via SHELL_CMD_REGISTER().
- Command history and auto-completion (via Tab).
- Modular backends: UART, USB CDC ACM, RTT, etc.

```
shell>device list
devices:
- syscon@0 (READY)
  DT node labels: syscon
- flexcomm@b7000 (READY)
  DT node labels: flexcomm7
- flexcomm@b4000 (READY)
  DT node labels: flexcomm4
- flexcomm@94000 (READY)
  DT node labels: flexcomm2
- flexcomm@93000 (READY)
  DT node labels: flexcomm1
- pinmux@42000 (READY)
  DT node labels: portf
- pinmux@11a000 (READY)
  DT node labels: porte
- pinmux@119000 (READY)
  DT node labels: portd
- pinmux@118000 (READY)
  DT node labels: portc
- pinmux@117000 (READY)
  DT node labels: portb
- pinmux@116000 (READY)
  DT node labels: porta
- lpuart@b4000 (READY)
  DT node labels: flexcomm4_lpuart4
- lpuart@94000 (READY)
  DT node labels: flexcomm2_lpuart2
- gpio@9e000 (READY)
  DT node labels: gpio4
- gpio@9a000 (READY)
  DT node labels: gpio2
- gpio@98000 (READY)
  DT node labels: gpio1
- gpio@96000 (READY)
  DT node labels: gpio0
shell>
```

Example Commands

```
shell>help
Please press the <Tab> button to see all available commands.
You can also use the <Tab> button to prompt or auto-complete all commands or its
subcommands.
You can try to call commands with <-h> or <--help> parameter for more information.

Shell supports following meta-keys:
  Ctrl + (a key from: abcdefklnpuw)
  Alt  + (a key from: bf)
Please refer to shell documentation for more details.

Available commands:
  device    : Device commands
  devmem    : Read/write physical memory
Usage:
Read memory at address with optional width:
devmem <address> [<width>]
Write memory at address with mandatory width and value:
devmem <address> <width> <value>
  hello     : Hello
  help      : Prints the help message.
  history   : Command history.
  kernel    : Kernel commands
  retval    : Print return value of most recent command
  shell     : Useful, not Unix-like shell commands.
shell>
```

```
kernel - Kernel commands
Subcommands:
  cycles    : Kernel cycles.
  sleep     : ms
  thread    : Kernel threads.
  uptime    : Kernel uptime. Can be called with the -p or --pretty options
  version   : Kernel version.
shell>
```

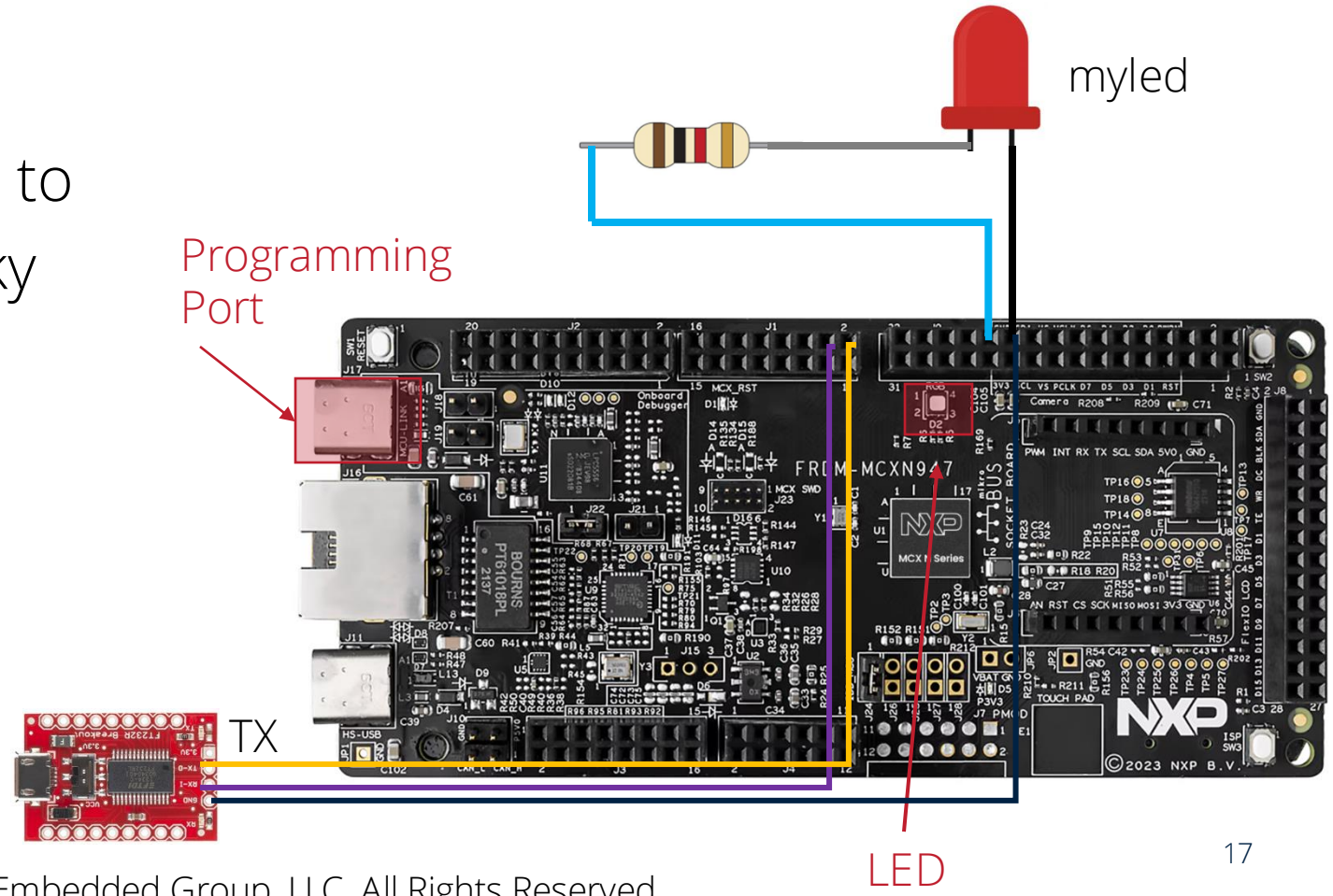
```
shell>kernel uptime
Uptime: 319352 ms
```

```
shell>kernel thread list
Scheduler: 800 since last call
Threads:
*0x30000198 shell_uart
.options: 0x0, priority: 14 timeout: 0
.state: queued, entry: 0x10002905
.stack size 2048, unused 1040, usage 1008 / 2048 (49 %)

 0x30000580 sysworkq
.options: 0x1, priority: -1 timeout: 0
.state: pending, entry: 0x10006561
.stack size 1024, unused 848, usage 176 / 1024 (17 %)

 0x30000338 idle
.options: 0x1, priority: 15 timeout: 0
.state: , entry: 0x10009037
.stack size 320, unused 256, usage 64 / 320 (20 %)

 0x300003f0 main
.options: 0x1, priority: 0 timeout: 7767
.state: sleeping, entry: 0x10005201
.stack size 1024, unused 720, usage 304 / 1024 (29 %)
```

Goals and Hardware Setup

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

- Map printk to USART2
- Add a Shell on USART4
- Explore shell messages



Creating Custom Commands

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

static int cmd_hello(const struct shell *shell, size_t argc, char **argv)
{
    shell_print(shell, "Hello from my app!");
    return 0;
}

SHELL_CMD_REGISTER(hello, NULL, "Hello", cmd_hello);
```

prj.conf

```
# Required for printk on UART2 (console)
CONFIG_SERIAL=y
CONFIG_PRINTK=y
CONFIG_CONSOLE=y
CONFIG_UART_CONSOLE=y

# Required for Shell on UART4
CONFIG_SHELL=y
CONFIG_SHELL_BACKEND_SERIAL=y
CONFIG_DEVICE_SHELL=y
CONFIG_KERNEL_SHELL=y

# Optional: Adjust shell prompt text
CONFIG_SHELL_PROMPT_UART="shell>"

# Configure the shell:
CONFIG_SHELL_CMD_BUFF_SIZE=128
CONFIG_SHELL_HISTORY_BUFFER=512
CONFIG_SHELL_VT100_COLORS=n
CONFIG_SHELL_VT100_COMMANDS=n

# Configure the shell commands
CONFIG_SHELL_CMDS=y
CONFIG_THREAD_MONITOR=y
```

Compiling and Running

app.overlay

```
/ {
    chosen {
        zephyr,console = &flexcomm2_lpuart2;
        zephyr,shell-uart = &flexcomm4_lpuart4;
        zephyr,uart-mcumgr = &flexcomm2_lpuart2;
    };

    leds {
        compatible = "gpio-leds";

        my_led: my_led {
            gpios = <&gpio1 22 GPIO_ACTIVE_LOW>;
            label = "My Custom LED";
        };
    };

    aliases {
        myled = &my_led;
    };
};
```

west build -p -b frdm_mcxn947/mcxn947/cpu0 zephyr/samples/basic/blinky
west flash

```
shell>hello
Hello from my app!
shell>
```

Audience POLL Question

Which of the following is true about the Zephyr Shell over a UART interface?

a) The shell only works over USB and cannot be routed to a standard UART peripheral.
b) The Zephyr shell allows users to interact with the system at runtime using built-in and custom commands.
c) The shell requires a GUI to function properly and cannot be used in headless environments.
d) The shell cannot display printk() output alongside command responses.

Next Steps

04

Homework: Configure a Timer!

Create a Zephyr application that uses a hardware timer or software timer (k_timer) to toggle an LED every 250ms.

Requirements:

- Use k_timer or k_work_delayable from the Zephyr kernel API.
- Initialize and start the timer during application startup.
- Toggle an LED in the timer's callback function.
- Use printk() to log each toggle event.
- Bonus: Use a shell command (e.g., start_timer / stop_timer) to control the timer from the serial shell.
- Extra Credit: Add a command that manually toggles the LED through the shell

Going Further

Download the extra resources:

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

Zephyr Documentation:

- Shell Services Documentation
- Uart Documentation
- Creating custom drivers
- List of Sample drivers

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