



**DesignNews**

Mastering Zephyr RTOS

# DAY 5: Debugging, Logging, and Best Practices

Sponsored by

**DigiKey**



## 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.

## THE SPEAKER



# Jacob Beningo

Jacob@beningo.com

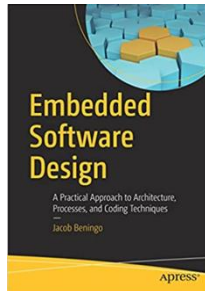


[jacobbeningo](#)

## 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.

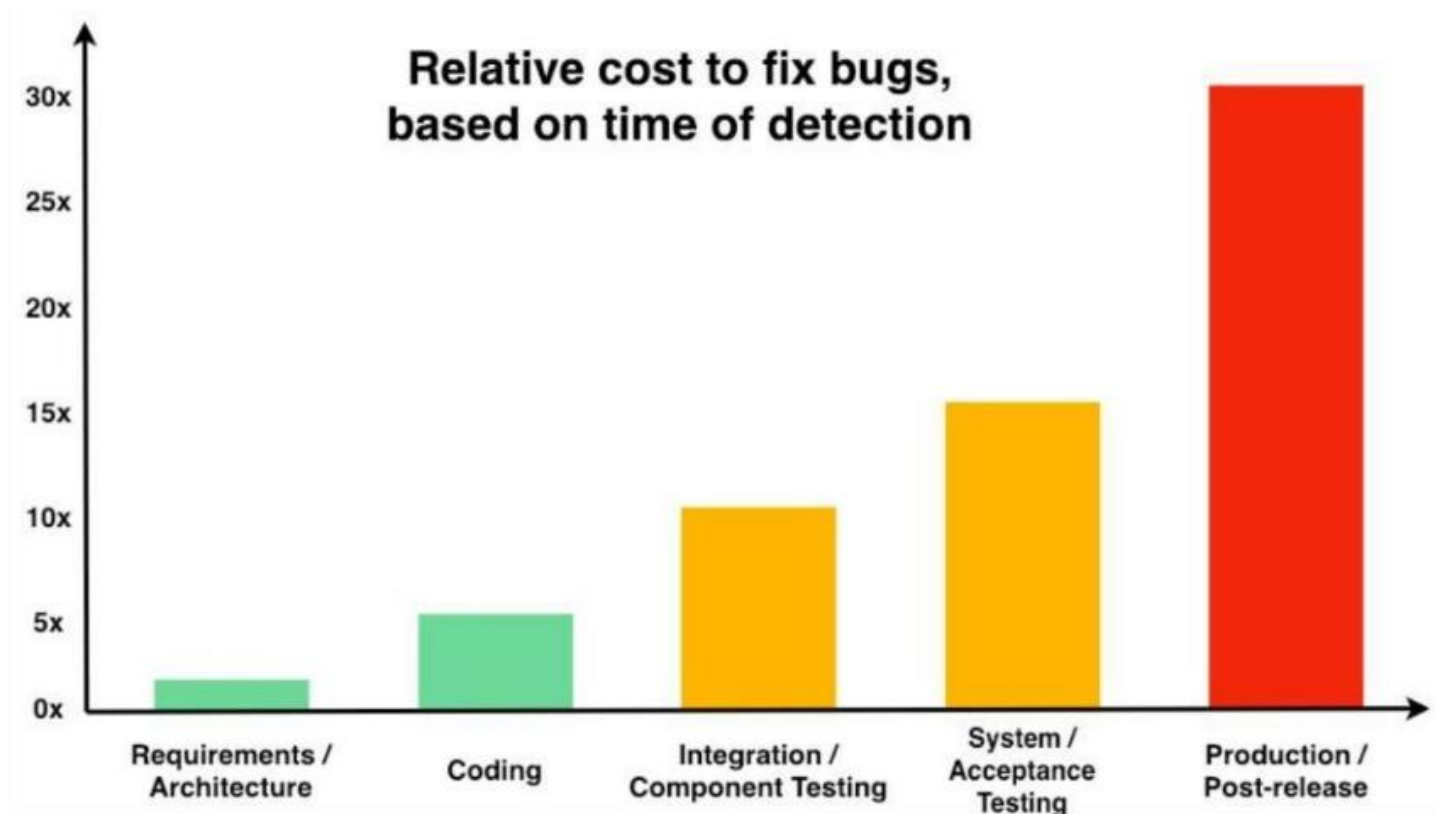


Visit [www.beningo.com](http://www.beningo.com) to learn more

•• Debugging

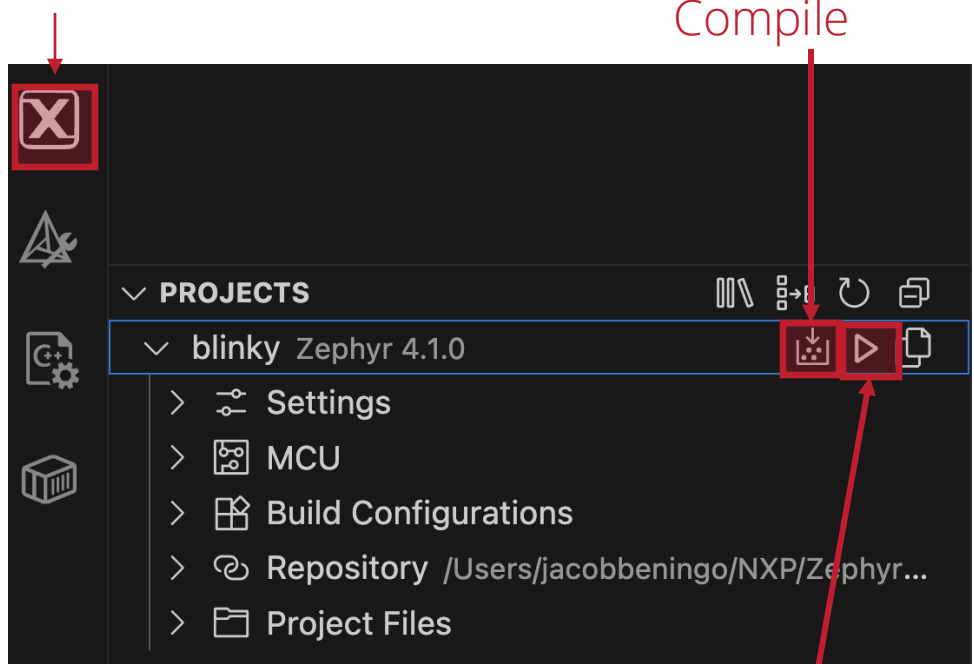
01

# Bugging is Expensive!



# Debug Using VS Code

MCUXpresso for VS Code Extension



```
synchronization > .vscode > {} launch.json > Launch Targets > {} Debug with SEGGER J-Link
1
2 {
3   "configurations": [
4     {
5       "type": "mcuxpresso-debug",
6       "name": "Debug with SEGGER J-Link",
7       "request": "launch",
8       "cwd": "${workspaceFolder}",
9
10      "executable": "${workspaceFolder}/build/zephyr/zephyr.elf",
11      "stopAtSymbol": "main",
12      "probeSerialNumber": "505100646",
13      "isAttach": false,
14      "probeType": "SEGGER",
15      "skipBuildBeforeDebug": false,
16      "skipPreFlashActions": false,
17
18      "gdbInitCommands": [
19        "set remotetimeout 600",
20        "set debug-file-directory",
21        "set non-stop off"
22      ],
23    }
24  ]
25 }
```

```
blinky > src > C main.c > main(void)
61
49   while (1) {
50     gpio_pin_set_dt(&led0, led_state);
51     gpio_pin_set_dt(&my_led, led_state);
52
53     printk("Blink! LED0 and myLED state: %s\n", led_state ?
54
55     led_state = !led_state;
56     k_msleep(SLEEP_TIME_MS);
57   }
58
59   return 0;
60 }
```

# SEGGER Ozone

The screenshot displays the SEGGER Ozone IDE interface. The main window shows a C source file named 'device.h' with the following code:

```

1 /* auto-generated by gen_syscalls.py, don't edit */
2
3 #ifndef Z_INCLUDE_SYSCALLS_DEVICE_H
4 #define Z_INCLUDE_SYSCALLS_DEVICE_H
5
6 #include <zephyr/tracing/tracing_syscall.h>
7
8 #ifndef _ASMLANGUAGE
9 #include <stdarg.h>
10 #include <zephyr/syscall_list.h>
11 #include <zephyr/syscall.h>
12 #include <zephyr/linker/sections.h>
13
14 #ifdef __cplusplus
15 extern "C" {
16 #endif
17
18 extern const struct device * z_impl_device_get_binding(const char * name);
19
20 #ifdef CONFIG_USERSPACE
21 static inline const struct device * device_get_binding(const char * name)
22 {
23     if (z_syscall_trap()) {
24         union { uintptr_t x; const char * val; } parm0 = { .val = name };
25         return (const struct device *) arch_syscall_invoke1(parm0.x, K_SYSCALL_DEVICE_GET_BINDING);
26     }
27 }
28 #endif
29
30 #ifdef CONFIG_USERSPACE
31 #define device_get_binding(name) ({ const struct device * syscall__retval; syscall__retval = device_get_binding(name); syscall__retval })
32 #endif
33
34 extern bool z_impl_device_is_ready(const struct device * dev);
35
36 #ifdef CONFIG_USERSPACE
37 static inline bool device_is_ready(const struct device * dev)
38 {
39     if (z_syscall_trap()) {
40         union { uintptr_t x; const struct device * val; } parm0 = { .val = dev };
41         return (bool) arch_syscall_invoke1(parm0.x, K_SYSCALL_DEVICE_IS_READY);
42     }
43 }
44 #endif
45
46 #ifdef CONFIG_USERSPACE
47 #define device_is_ready(dev) ({ bool syscall__retval; syscall__retval = device_is_ready(dev); syscall__retval })
48 #endif
49
50 extern int z_impl_device_init(const struct device * dev);
51
52 #ifdef CONFIG_USERSPACE
53 static inline int device_init(const struct device * dev)
54 {
55     if (z_syscall_trap()) {
56         union { uintptr_t x; const struct device * val; } parm0 = { .val = dev };
57         return (int) arch_syscall_invoke1(parm0.x, K_SYSCALL_DEVICE_INIT);
58     }
59 }
60 #endif
61
62 #ifdef CONFIG_USERSPACE
63 #define device_init(dev) ({ int syscall__retval; syscall__retval = device_init(dev); syscall__retval })
64 #endif
65
66 #endif
67
68 #endif
69
70 #endif
71
72 #endif
73
74 #endif
75
76 #endif
77
78

```

The right-hand pane shows the 'Registers 1 (CPU)' window, listing various CPU registers with their names, values, and descriptions. The list includes:

- CPU: 694 Registers
- Core (N): 26 Registers
- R0: General purpose
- R1: General purpose
- R2: General purpose
- R3: General purpose
- R4: General purpose
- R5: General purpose
- R6: General purpose
- R7: General purpose
- R8: General purpose
- R9: General purpose
- R10: General purpose
- R11: General purpose
- R12: General purpose
- R13 (SP): Stack pointer (SP)
- R14 (LR): Link register (LR)
- R15 (PC): Program counter
- MSP: Main stack pointer
- PSP: Process stack pointer
- APSR: Application program status register
- EPSR: Exception program status register
- IPSR: Interrupt program status register
- PrIMask: Priority mask register
- BasePRI: Base priority mask register
- FaultMask: Fault mask register
- Control: Control register
- CycleCount: Cycle count
- Core (A): 39 Registers
- All CPU registers
- FPU: 33 Registers
- FPU Registers (d0-d15): 17 Registers
- FPU Registers (d16-d31): 16 Registers
- Peripherals: 379 Registers
- Cache Maintenance: 6 Registers
- DCB: 6 Registers
- DIB: 17 Registers
- DPB: 159 Registers
- DWT: 52 Registers
- FIP: 6 Registers
- ICB: 3 Registers
- ITM: 26 Registers
- MPU: 13 Registers
- NVIC: 213 Registers
- SAU: 7 Registers
- SCB: 39 Registers
- SIG: 1 Register
- SYST: 4 Registers
- TPU: 23 Registers

## Audience POLL Question

What is your preference for debugging your Zephyr RTOS applications?

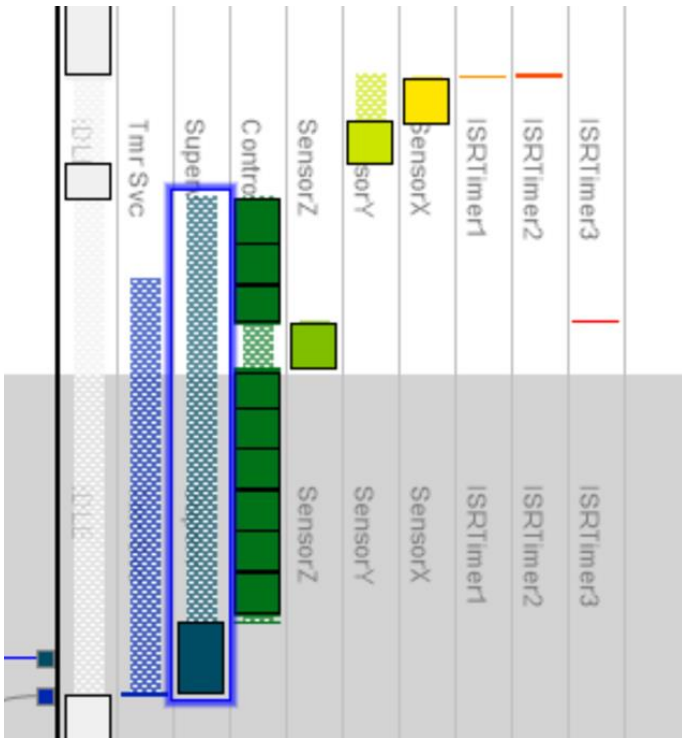
- a) Using raw gdb with west debug
- b) Using Visual Studio Code
- c) Using SEGGER Ozone
- d) Using run-time capabilities like logs, the shell, etc

•• Debug Trace

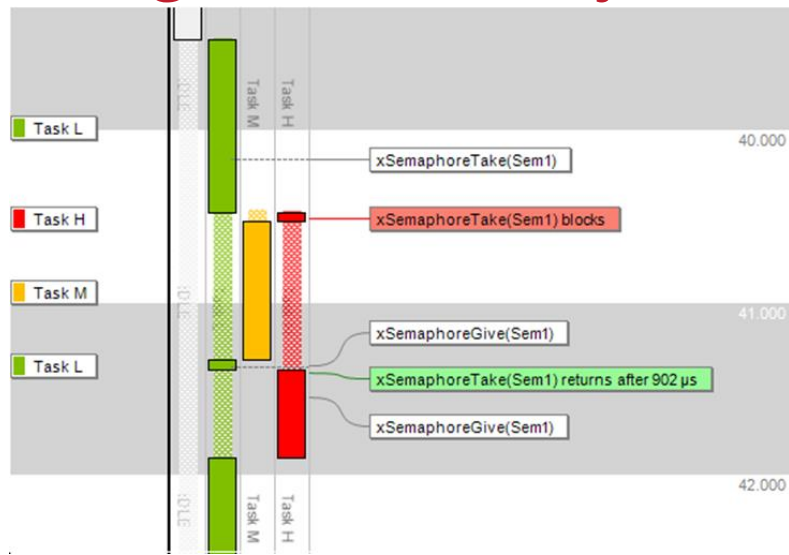
02

# The Power of Tracing

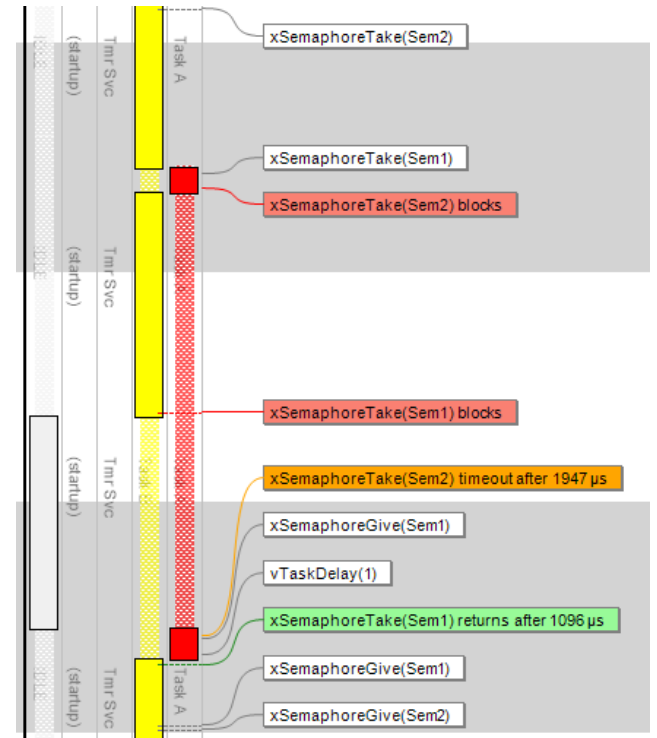
Danger #1 – Task



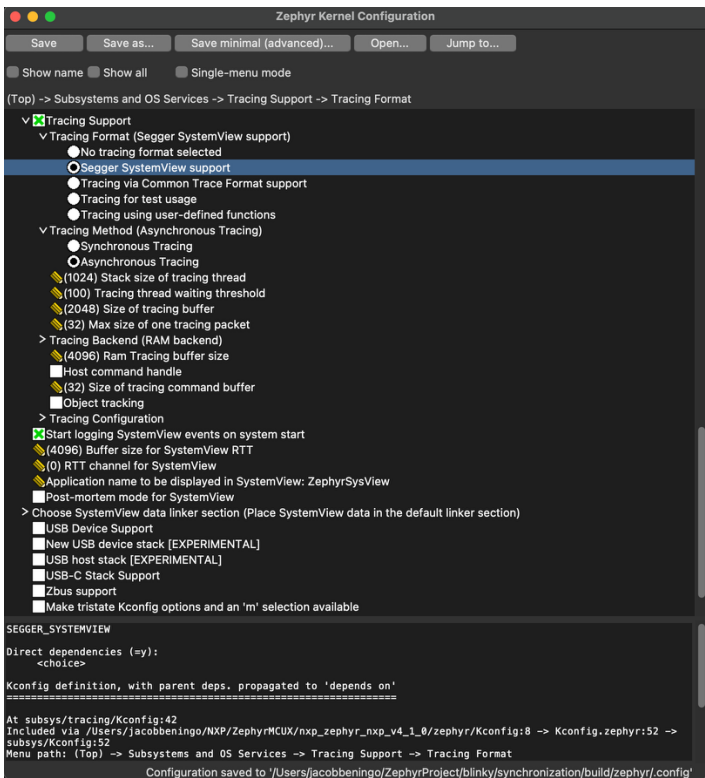
Danger #2 – Priority



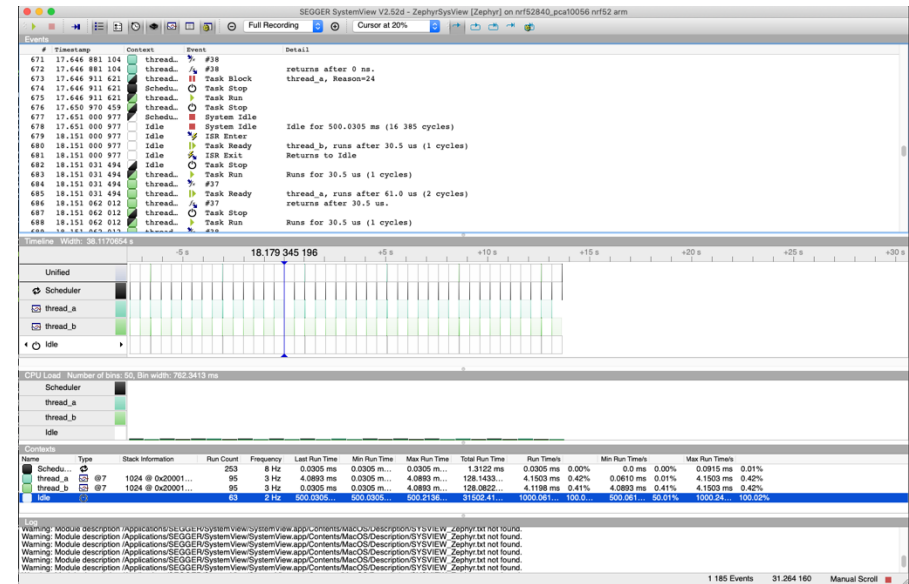
Danger #3 - Deadlock



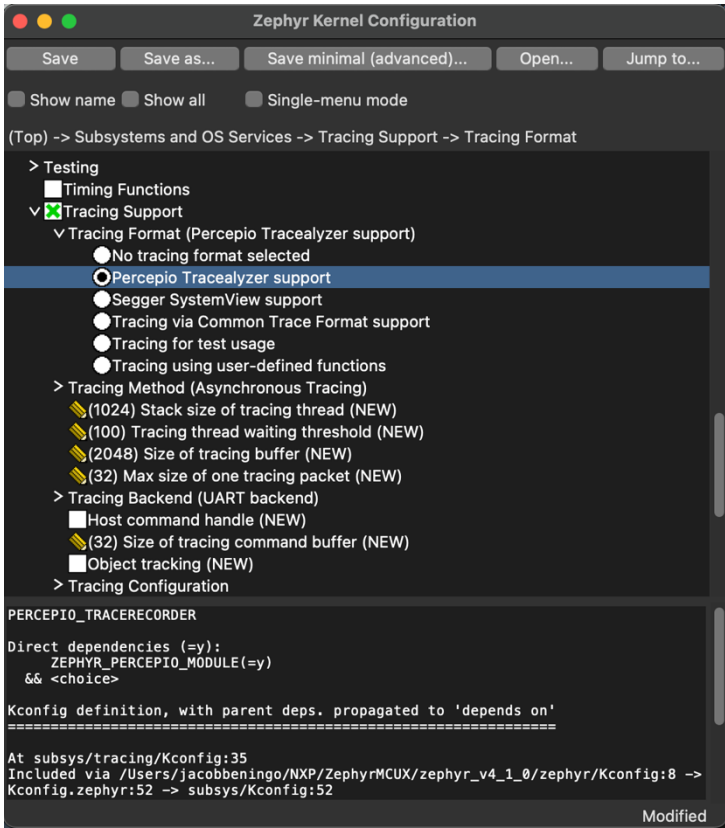
# SEGGER SystemView



```
CONFIG_STDOUT_CONSOLE=y
CONFIG_THREAD_NAME=y
CONFIG_SEGGER_SYSTEMVIEW=y
CONFIG_USE_SEGGER_RTT=y
CONFIG_TRACING=y
```

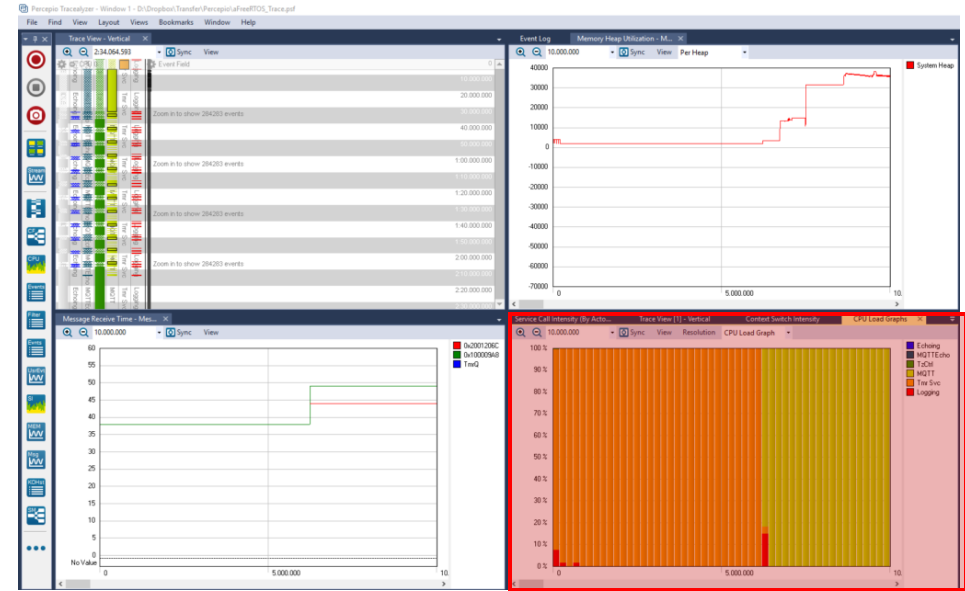


# Percepio TraceView



```

CONFIG_TRACING=y
CONFIG_PERCEPIO_TRACERECORDER=y
    
```



## Audience POLL Question

Which of the following best describes the purpose of tracing in an embedded system running Zephyr RTOS?

- a) To encrypt communication between tasks and peripherals
- b) To monitor and record system events for debugging and performance analysis
- c) To configure GPIO pins dynamically during runtime
- d) To allocate stack memory for kernel threads

•• Logging

03

# What is logging?

A structured way to report system messages like errors, warnings, info, and debug output.

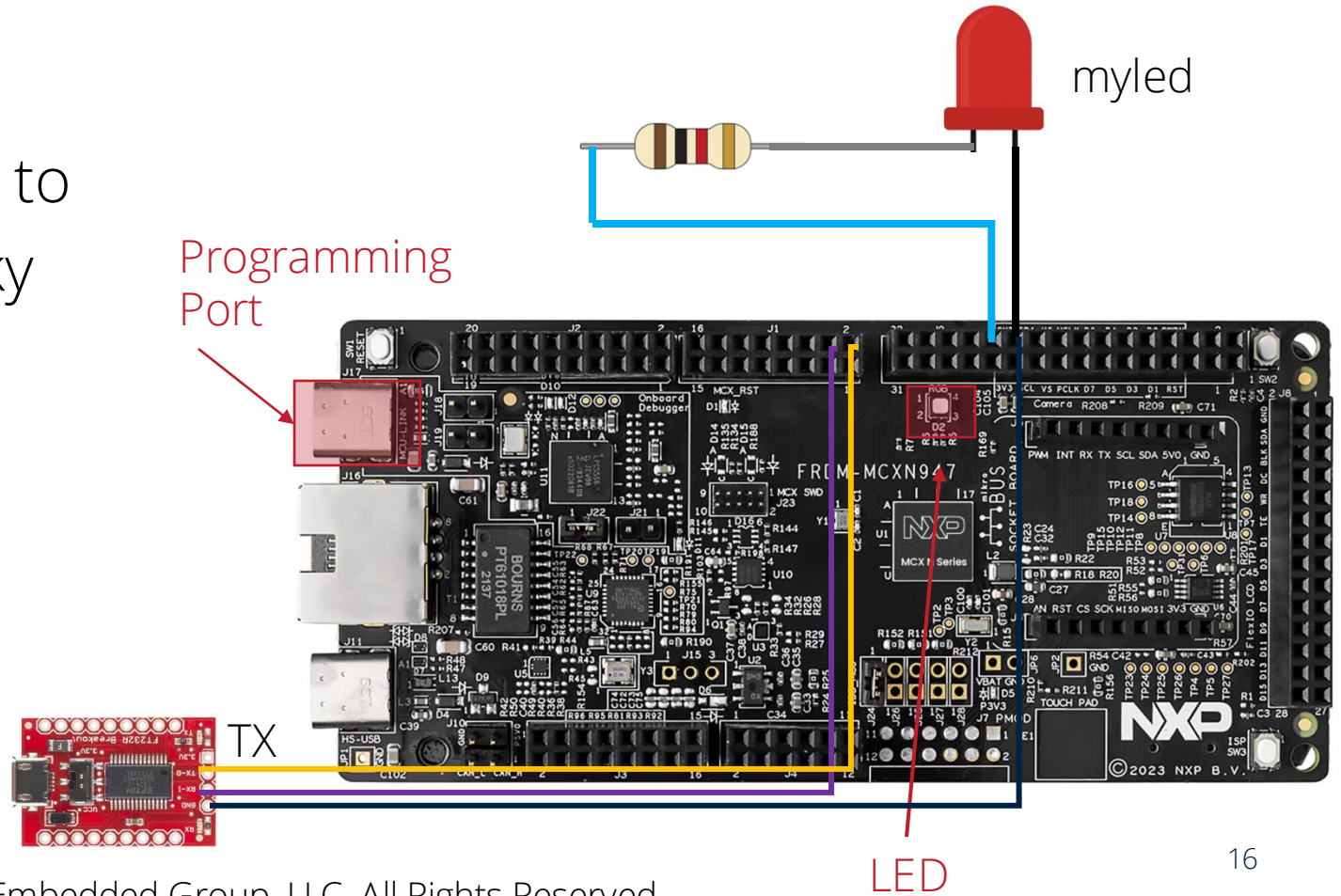
## Key Features in Zephyr Logging:

- LOG\_ERR(), LOG\_WRN(), LOG\_INF(), LOG\_DBG() macros
- Module-based: LOG\_MODULE\_REGISTER(main)
- Adjustable log levels per module
- Deferred or immediate logging modes
- Output backends: UART, RTT, memory

# Goals and Hardware Setup

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

- Display log info on USART2
- Keep the Shell on USART4
- Explore Log messages



# Enabling Logging

```
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;
  };
};
```

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

# Logging configuration - use UART2 for logs
CONFIG_LOG=y
CONFIG_LOG_BACKEND_UART=y
CONFIG_LOG_MODE_DEFERRED=y
CONFIG_LOG_MODE_IMMEDIATE=n
CONFIG_LOG_DEFAULT_LEVEL=3
CONFIG_LOG_BACKEND_UART_AUTOSTART=y

# Disable RTT backend if you don't need it
CONFIG_LOG_BACKEND_RTT=n

# Shell configuration - use UART4 for shell
CONFIG_SHELL=y
CONFIG_SHELL_BACKEND_SERIAL=y
CONFIG_DEVICE_SHELL=y
CONFIG_KERNEL_SHELL=y

# Shell configuration
CONFIG_SHELL_PROMPT_UART="shell>"
CONFIG_SHELL_CMD_BUFF_SIZE=128
CONFIG_SHELL_HISTORY_BUFFER=512
CONFIG_SHELL_VT100_COLORS=n
CONFIG_SHELL_VT100_COMMANDS=n
CONFIG_SHELL_CMDS=y
CONFIG_THREAD_MONITOR=y

# Important: Disable shell log backend to prevent logs from appearing on shell
CONFIG_SHELL_LOG_BACKEND=n

# Disable colored log output
CONFIG_LOG_BACKEND_FORMAT_TIMESTAMP=y
CONFIG_LOG_BACKEND_SHOW_COLOR=n
```

## The Code

```
#include <zephyr/kernel.h>
#include <zephyr/drivers/gpio.h>
#include <zephyr/logging/log.h>
#include <zephyr/shell/shell.h>

LOG_MODULE_REGISTER(main); // Register the logging module
```

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

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

    if (!gpio_is_ready_dt(&my_led)) {
        LOG_ERR("myLED device not ready");
        return 0;
    }
}
```

```
LOG_INF("Starting LED blink loop...");

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

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

    led_state = !led_state;
    k_msleep(SLEEP_TIME_MS);
}

return 0;
```

# Result

The image shows two terminal windows side-by-side. The left window, titled 'Untitled\_0', shows a shell prompt 'shell>'. The right window, titled 'Untitled\_1', shows the output of a Zephyr OS boot sequence. The boot logs include the following text:

```
*** Booting Zephyr OS build v4.1.0 ***  
[00:00:00.000,000] <inf> main: Starting LED blink loop...  
[00:00:00.000,000] <inf> main: Blink! LED0 and myLED state: ON  
[00:00:01.000,000] <inf> main: Blink! LED0 and myLED state: OFF  
[00:00:02.000,000] <inf> main: Blink! LED0 and myLED state: ON  
[00:00:03.000,000] <inf> main: Blink! LED0 and myLED state: OFF  
[00:00:04.000,000] <inf> main: Blink! LED0 and myLED state: ON  
[00:00:05.000,000] <inf> main: Blink! LED0 and myLED state: OFF  
[00:00:06.000,000] <inf> main: Blink! LED0 and myLED state: ON  
[00:00:07.001,000] <inf> main: Blink! LED0 and myLED state: OFF  
[00:00:08.001,000] <inf> main: Blink! LED0 and myLED state: ON  
[00:00:09.001,000] <inf> main: Blink! LED0 and myLED state: OFF  
[00:00:10.001,000] <inf> main: Blink! LED0 and myLED state: ON  
[00:00:11.001,000] <inf> main: Blink! LED0 and myLED state: OFF  
[00:00:12.001,000] <inf> main: Blink! LED0 and myLED state: ON  
[00:00:13.001,000] <inf> main: Blink! LED0 and myLED state: OFF  
[00:00:14.001,000] <inf> main: Blink! LED0 and myLED state: ON  
[00:00:15.001,000] <inf> main: Blink! LED0 and myLED state: OFF  
[00:00:16.001,000] <inf> main: Blink! LED0 and myLED state: ON  
[00:00:17.002,000] <inf> main: Blink! LED0 and myLED state: OFF  
[00:00:18.002,000] <inf> main: Blink! LED0 and myLED state: ON  
[00:00:19.002,000] <inf> main: Blink! LED0 and myLED state: OFF  
[00:00:20.002,000] <inf> main: Blink! LED0 and myLED state: ON  
[00:00:21.002,000] <inf> main: Blink! LED0 and myLED state: OFF  
[00:00:22.002,000] <inf> main: Blink! LED0 and myLED state: ON  
[00:00:23.002,000] <inf> main: Blink! LED0 and myLED state: OFF
```

At the bottom of each terminal window, there is a status bar. The left window shows 'usbmodem4DOXHPLTUH0BX3 (NXP Semiconductors)' with status indicators for TX, RX, RTS, CTS, DTR, DSR, DCD, and RI. The right window shows 'usbserial-A601LO45 / 115200 8-N-1' with a 'Display Paused' indicator and the same status indicators.

## Audience POLL Question

What is the most important configuration to ensure your log messages don't end up on all your UART devices?

- a) CONFIG\_LOG\_MODE\_DEFERRED=y
- b) CONFIG\_SHELL\_LOG\_BACKEND=n
- c) CONFIG\_SHELL\_LOG\_BACKEND=n
- d) CONFIG\_LOG\_BACKEND\_RTT=n

•• Next Steps

04

## Going Further

Download the extra resources:

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

Zephyr Documentation:

- [Tracing Documentation](#)
- [Logging Documentation](#)

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](http://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



**DesignNews**

Thank You

Sponsored by

**DigiKey**

