16-17 June 2026 | Mumbai, India
#OSSummit

Does Zephyr Scare the Bare Metal Embedded Developer World ... ?

Khasim Syed Mohammed & Soumya Tripathy

Texas Instruments

Texas Instruments

Topic Abstract

- Bare-metal developers pride themselves on simplicity, control, and understanding every line of code. Then along comes Zephyr—with device trees, Kconfig, west, and layers of abstraction—and suddenly, even blinking an LED feels complicated. So… is Zephyr actually scary?

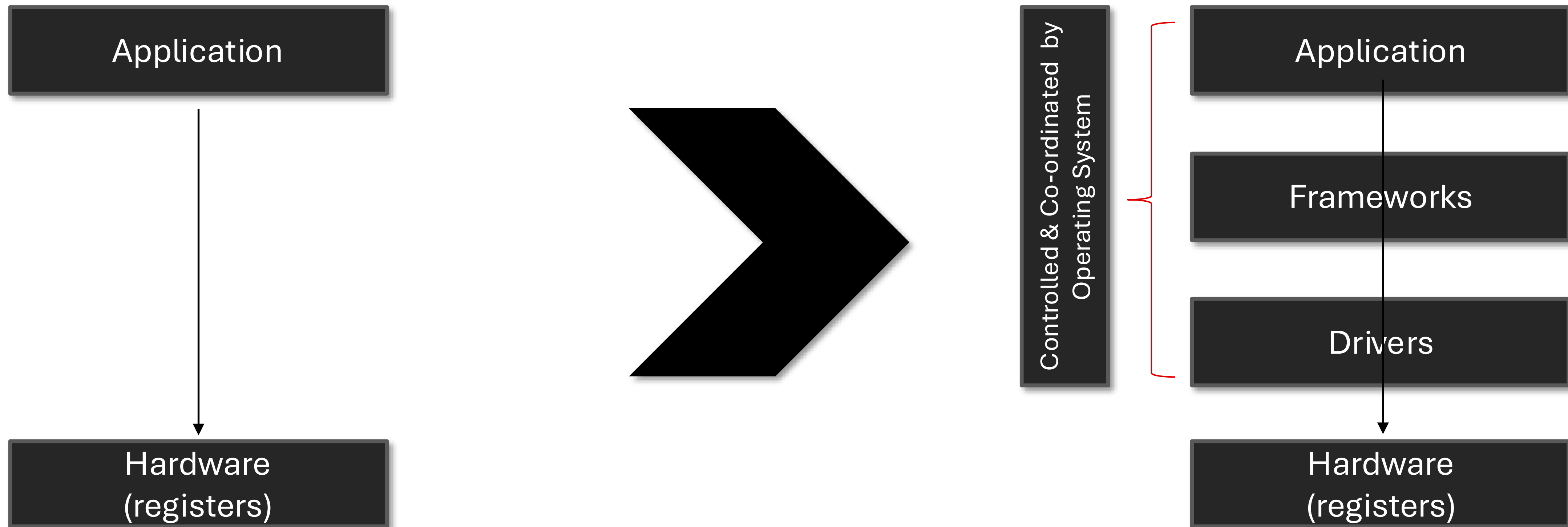
- In this talk, we take a practical and honest look at why Zephyr often feels overwhelming to bare-metal developers, what's really going on under the hood, and whether that complexity is justified. Through side-by-side comparisons and live examples, we map familiar bare-metal concepts to their Zephyr equivalents and uncover where the fear comes from—and where it disappears.

- This isn't a "Zephyr is better" talk. It's about understanding trade-offs, choosing the right tool, and making the transition without losing your mental model.

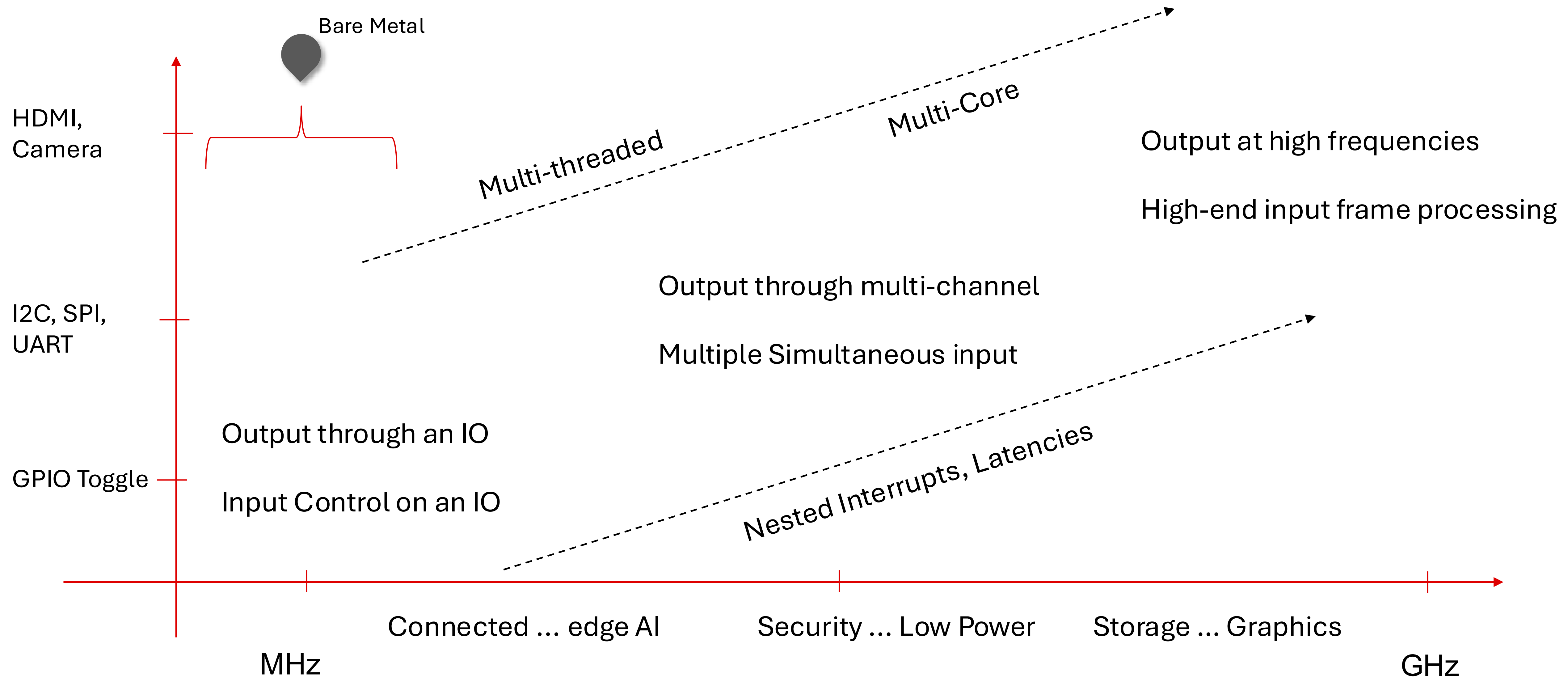
- By the end, you'll see that Zephyr isn't replacing bare metal—it's structuring the complexity you were already managing.

What is bare metal software programming ?

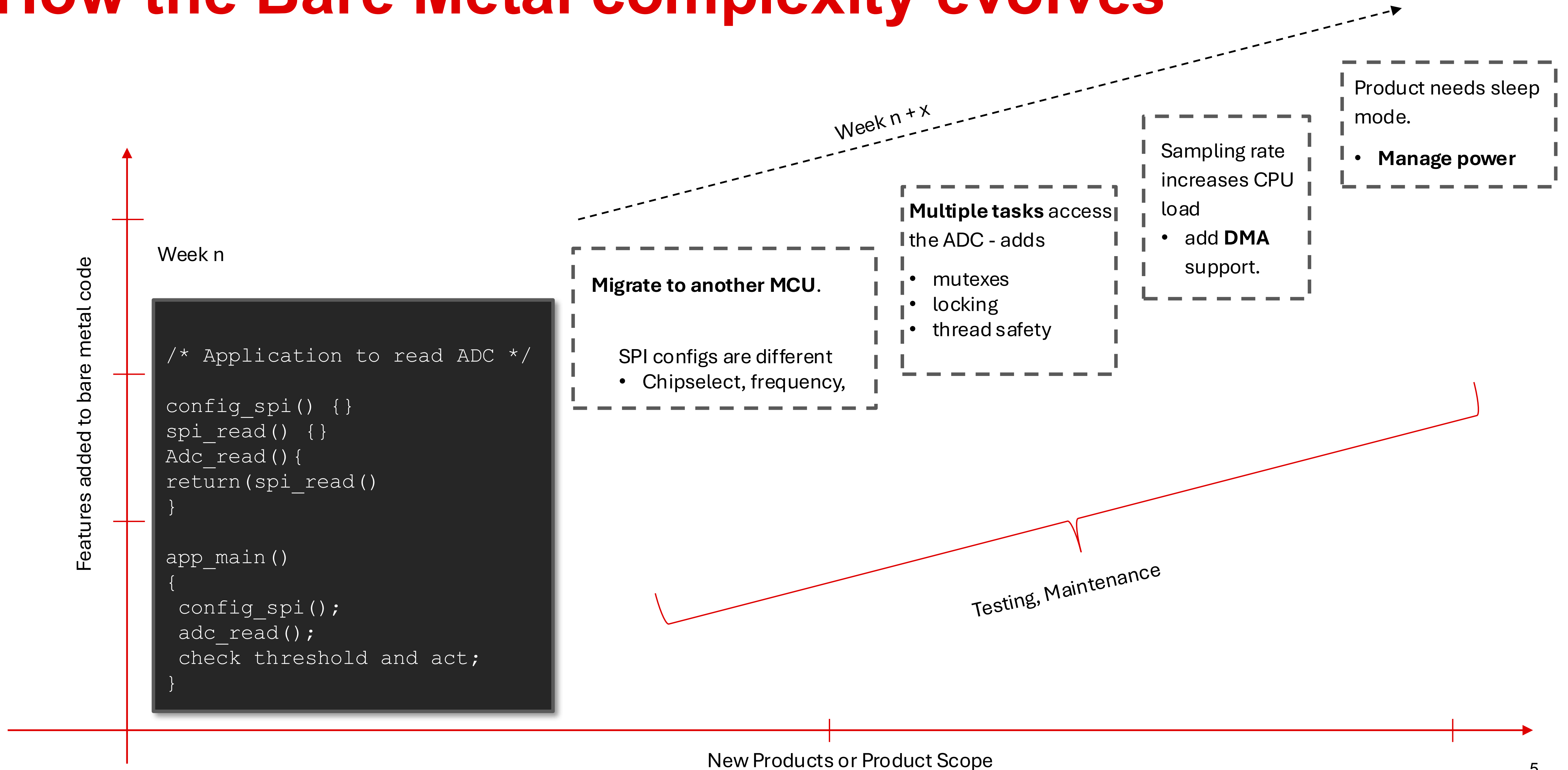
Access hardware directly by accessing the registers without any frameworks or standard definitions as constraints.



Products and their Software Complexities



How the Bare Metal complexity evolves



Texas Instruments

What happened in the process of evolution

- Configuration system
- Device model
- Driver framework
- Power management framework
- Synchronization framework
- Testing framework

- In other words… **started reinventing Zephyr.**

Texas Instruments

Let’s Look at Examples : Bare metal vs Zephyr

- Blink LED
- ADC
- Interrupt

Texas Instruments

Simple GPIO Toggle

Bare Metal View

```
#include "gpio.h"

int main(void)
{
   gpio_init(GPIO0, 12, GPIO_OUTPUT);

   while (1) {
      gpio_set(GPIO0, 12);
      delay_ms(500);

      gpio_clear(GPIO0, 12);
      delay_ms(500);
   }
}
```

Zephyr View

```
#include <zephyr/kernel.h>
#include <zephyr/drivers/gpio.h>
#define LED_NODE DT_ALIAS(led0)

static const struct gpio_dt_spec led =
   GPIO_DT_SPEC_GET(LED_NODE, gpios);
int main(void)
{
   gpio_pin_configure_dt(&led,
                  GPIO_OUTPUT_ACTIVE);
   while (1) {
      gpio_pin_toggle_dt(&led);
      k_msleep(500);
   }
}
```

Application

GPIO Regs

LED

Application

GPIO API

GPIO Driver

LED

Most real products eventually contain:

```
#ifdef BOARD_A
#define LED_PIN 12
#endif

#ifdef BOARD_B
#define LED_PIN 18
#endif

#ifdef BOARD_C
#define LED_PIN 23
#endif
```

which grows into:

- board.h
- board_v2.h
- customer_a.h
- customer_b.h

Zephyr moves this into DTS:

```
led0 {
   gpios = <&main_gpio0 12 GPIO_ACTIVE_HIGH>;
};
```

Application never changes.

Simple ADC Driver

Bare Metal View

```
int main(void)
{
  ads7066_init();
  while (1) {
    uint16_t ch0 =
      ads7066_read_channel(0);
    uint16_t ch1 =
      ads7066_read_channel(1);
    printf("CH0=%u CH1=%u\n", ch0, ch1);
  }
}
```

Zephyr View

- prj.conf
- west.yml
- CMake
- Kconfig
- Device Tree
- Drivers
- Subsystems

Application

ADC Driver

SPI HAL

LED

- adc_ads7066.c
- ti,ads7066.yaml
- Kconfig
- CMakeLists
- board overlay
- sample app

Most real products eventually contain:

Device Tree

```
&main_spi0 {
  status = "okay";
  ads7066: adc@0 {
    compatible = "ti,ads7066";
    reg = <0>;
    spi-max-frequency = <10000000>;
    status = "okay";
  };
};
```

Binding

```
compatible: "ti,ads7066"
include:
 - spi-device.yaml
properties:
 spi-max-frequency:
  required: true
```

Application Example

```
const struct device *adc =
    DEVICE_DT_GET(DT_NODELABEL(ads7066));
struct adc_sequence seq = {
  .channels = BIT(0),
  .buffer = &sample,
  .buffer_size = sizeof(sample),
};
adc_read(adc, &seq);
```


Texas Instruments

Simple Interrupt Logic

Bare Metal View

```
void GPIO_IRQHandler(void)
{
  if(GPIO->STATUS & BUTTON_INT)
  {
   // Clear interrupt
    GPIO->STATUS = BUTTON_INT;
    led_toggle();
    counter++;
  }
}
```

Zephyr View

```
static void button_isr(const struct device *dev,
                       struct gpio_callback *cb, uint32_t pins). {
  k_work_submit(&button_work);
}
```

Deferred work:

```
static void button_work_handler(struct k_work *work)
{
  led_toggle ();
  counter++;
}
```

Most real products eventually contain:

ISR:

```
void adc_isr(...)
{
  latest_sample = read_sample();
  k_sem_give(&adc_sem);
}
```

Worker thread:

```
while (1)
{
  k_sem_take(&adc_sem, K_FOREVER);
  process_sample(latest_sample);
  update_filter(latest_sample);
  update_display(latest_sample);
}
```

ISR stays small.

Latency is High in Zephyr – if you differ to do a lot.

Bare Metal gives you the freedom to do everything in the ISR. Zephyr encourages you not to."

```
Hardware Interrupt
        ↓
 Interrupt Vector
        ↓
       ISR
        ↓
    App Logic
```

```
Hardware Interrupt
        ↓
   Zephyr ISR
        |→ Quick ISR Work
        |→ Wake Thread
        |→ App Logic
        ↓
```

Option : A Zero Latency ISR behaves like a bare-metal ISR because Zephyr treats it like one.

How Zephyr makes migration from bare-metal easy

Zephyr Doesn't Ask You to Stop Being a Bare-Metal Developer

Texas Instruments

Bare-metal concepts – just extended

Approach

Stage 1
Focus on
Startup
Build System
Drivers

No RTOS usage.

Stage 2
Add:
Logging
Shell
Device Tree

Stage 3
Add:
Threads
Queues
Synchronization

Stage 4
Add:
Networking
Bluetooth
USBa

1. Direct Hardware Access Still Exists.

A common fear is:
"Zephyr will stop me from touching registers." Not true, following still works :

```
GPIO0->DATA = 1;
or
sys_write32(val, reg_addr);
```

2. Main() Still Exists

```
int main(void)
{
   while (1) {
   }
}
```

The familiar entry point remains. Start with a single-threaded app and add RTOS features later.

3. RTOS Features Are Optional
Developers think:
Zephyr = Threads - Not True,

Start with:
No threads
No networking
No filesystem
No shell

Enable features only when needed.

4. Excellent Debugging Support

- **GDB works**
- **JTAG works**
- **OpenOCD works**
- **Trace tools work**

Can still step into:

gpio_pin_set();

and eventually reach the register write.

5. Drivers Look Familiar
A bare-metal driver:

spi_transfer();
adc_read();

A Zephyr driver:

spi_transceive_dt();
adc_read();

Different API. Same concepts.

Application
↓
Driver
↓
Peripheral

6. Device Tree Removes Board-Specific Code

Bare metal often becomes:

```
#ifdef BOARD_A
#define ADC_CS 5
#endif

#ifdef BOARD_B
#define ADC_CS 10
#endif
```

Over time:
board.h
board_v2.h
customer_a.h
customer_b.h

Zephyr moves that information into DTS.

Result:
Driver unchanged
Board description changes
This is one of the biggest migration benefits.

With AI giving code Bare-metal or Zephyr how does it matter ?

Texas Instruments

If AI writes code – does Zephyr infra matter ?

Bare Metal + AI

Ask Claude to generate:
ads7066_read();

Later another engineer asks:

adc_read_ads7066();

Another asks:
get_adc_sample();

All are technically correct.
Now we have:

- Three APIs
- Three coding styles
- Three driver architectures

AI helped write code.
It didn't create a platform.

What Zephyr Gives AI

Zephyr provides constraints.

For example:

- ADC API
- SPI API
- GPIO API
- UART API
- Driver Model
- Device Tree

Now AI is told:
Write a Zephyr ADC driver. The result must fit a known structure.

Instead of: Infinite possible implementations we get:

- **One ecosystem**
- **One architecture**

Historically value is in :

- SDK
- Examples
- Drivers

AI can generate much of that.

What becomes harder to replicate is:

- Upstream support
- Standards
- Ecosystem
- Community
- Compatibility

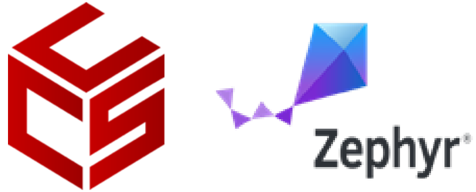
These are exactly the areas where **Zephyr takes lead**.

What TI offers to make migration easier

Texas Instruments

Scalable, future-ready portfolio from TI

The only portfolio that brings true edge AI, Low Power, Security, Connectivity and Performance across every embedded platform.



Unified, Open Source Software and tools

MHz → GHz

General-purpose microcontrollers

Wireless connectivity

Real-time microcontrollers

Arm A Core Processors

Multicore Processors

Technology	Zephyr™ For MCU*		Zephyr™ For Multi-core MPU*	
Core	Arm Cortex-M0 & -M33	Arm Cortex-M33, -R	Arm Cortex M4F & R5F	Arm Cortex-A53 + M4 + R5 or M85
CPU frequency	24 – 200 MHz	200 MHz – 1 GHz	160-400 MHz	1GHz – 2GHz
Example applications	Time series classification Keyword wakeup Glass break or motion detect Sensors & controls	Motor Control, PLCs, Preventive diagnostics, maintenance & fault detection	Industrial : HMI, Industrial PLCs, Networking Edge AI : Vision, radar, lidar, speed, audio Automotive : Audio, Radar, Controls	

* Micro Controllers and Multi-core Microprocessors

Summary

Does Zephyr Scare the Bare Metal Embedded Developer World ... ?

- Upstream support
- Standards
- Ecosystem
- Community
- Compatibility

- In the AI era, the value of Zephyr is not that it writes less code. It's that everyone writes code the same way.

Benefit	Why It Matters
Familiar hardware model	GPIO, SPI, I2C still look familiar
Incremental adoption	No need to learn everything at once
Portability	Change boards without rewriting applications

The easiest way to learn Zephyr is not to stop writing bare-metal software. It's to start writing bare-metal software inside Zephyr.

Requesting you to participate in the Zephyr Survey 2026

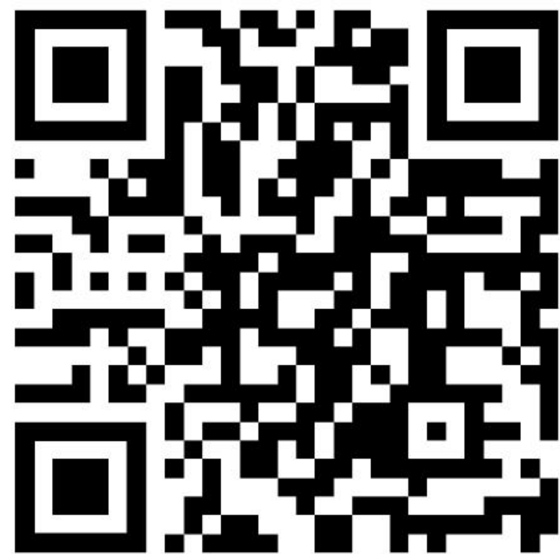
Texas Instruments

// DEVELOPER SURVEY 2026

Take the survey.

10 minutes to help shape the next
10 years of Zephyr RTOS.

• CLOSES JUNE 30 • ANONYMOUS • ~10 MIN



zephyrproject.org/devsurvey2026

Q&A

- Contact Information:
  - https://www.ti.com/opensource
  - https://www.ti.com/zephyr
  - https://discord.com

- Thanks to our ecosystem partners
  - https://www.ti.com/ww/in/third_party.html

Why choose TI MCUs and processors?

Scalability

Our products offer scalable performance that can adapt and grow as the needs of your customers evolve.

Efficiency

We design products that extend battery life, maximize performance for every watt expended, and unlock the highest levels of system efficiency.

Affordability

We strive to make innovation accessible to all by creating cost-effective products that feature state-of-the-art technology and package designs.

Availability

Our investment in internal manufacturing capacity provides greater assurance of supply, supporting your growth for decades to come.