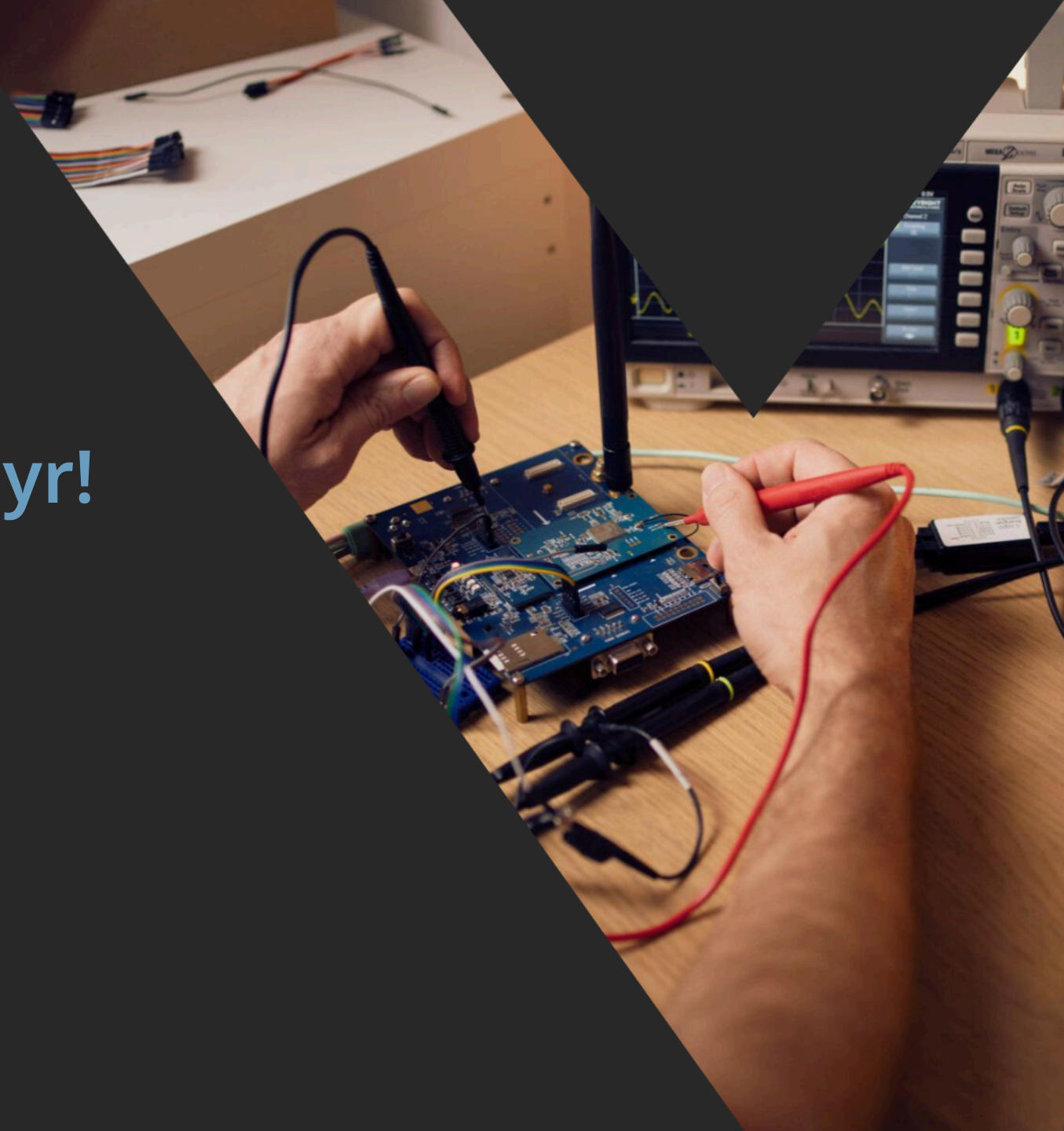


Hello world! Hello Zephyr!

Dawid Marszałkiewicz

February 4, 2025

{ GoodByte
embedded software }

Czym jest Zephyr?

- **RTOS dla systemów wbudowanych**, rozwijany przez Linux Foundation.
- Obsługuje **wiele architektur** (ARM, RISC-V, x86).
- **Modularna budowa** oparta na Kconfig i Devicetree.
- **System sterowników** dla różnych urządzeń.
- **Społeczność i aktywny rozwój** w ramach Linux Foundation.

Zephyr®

Jak to się zaczęło?

- **Początek w 2014 roku** jako projekt Intela.
- **Przekazany do Linux Foundation** w 2016 roku.
  - Stan repo około 7k commitów
- Początkowo wspierał **tylko 4 platformy**.
  - Arduino 101
  - Generic ARM (np. NXP FRDM-K64f)
  - Arduino Due
  - Intel Galileo Board

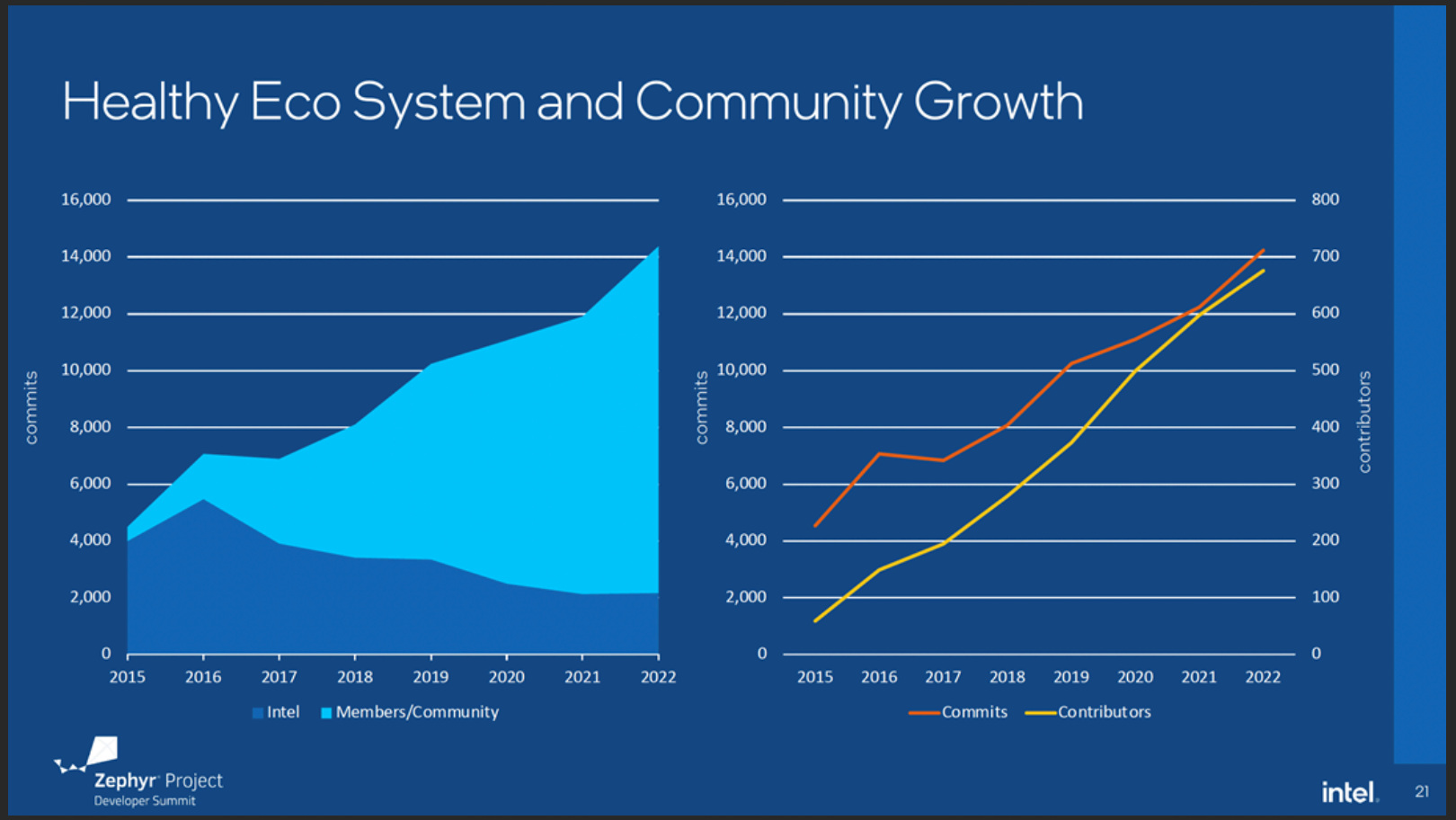
Żródło: The Zephyr Story - Intel

Jak to się kręci?

- **Liczba commitów** wzrosła do ponad 80 tysięcy.
- **Obsługa rozszerzona** z 4 do ponad **500 platform sprzętowych**.
- **Wiele nowych architektur**, m.in. **RISC-V, Xtensa, ARC**.
- **Silne wsparcie społeczności.**

Żródło: The Zephyr Story - Intel

Jak silne?


Healthy Eco System and Community Growth
commits
16,000
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
2015
2016
2017
2018
2019
2020
2021
2022
Intel
Members/Community
commits
contributors
800
700
600
500
400
300
200
100
Commits
Contributors
Zephyr Project
Developer Summit
intel
21

Co jeszcze jest w Zephyrze?

Komunikacja

- TCP/IP, MQTT, HTTP, CoAP
- USB, Bluetooth, CAN, LoRa
- Moduły sieciowe dla IoT i przemysłu

Obsługa urządzeń

- Sterowniki dla SPI, I2C, UART, PWM
- Obsługa wyświetlaczy i czujników
- USB device & host stack

Co jeszcze jest w Zephyrze?

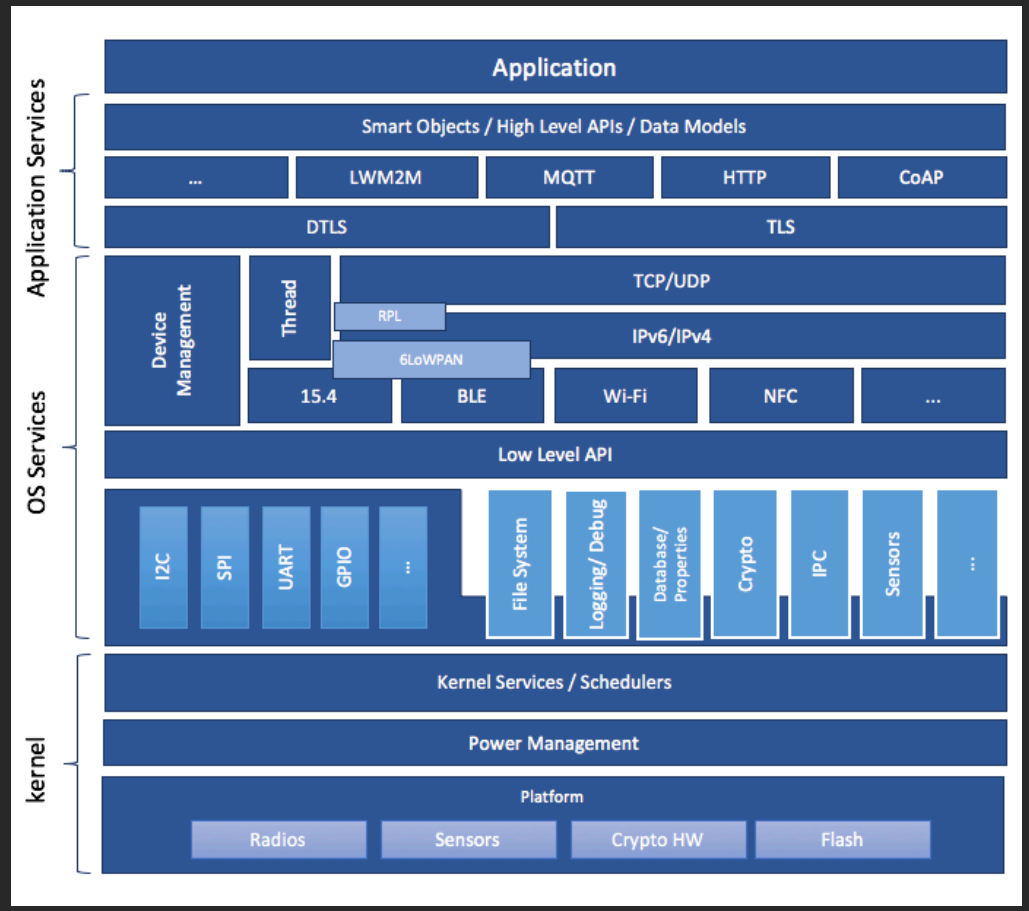
Warstwa aplikacji

- LVGL – grafika dla urządzeń wbudowanych
- Protobuf – serializacja danych
- Filesystems (LittleFS, FAT, NVS)

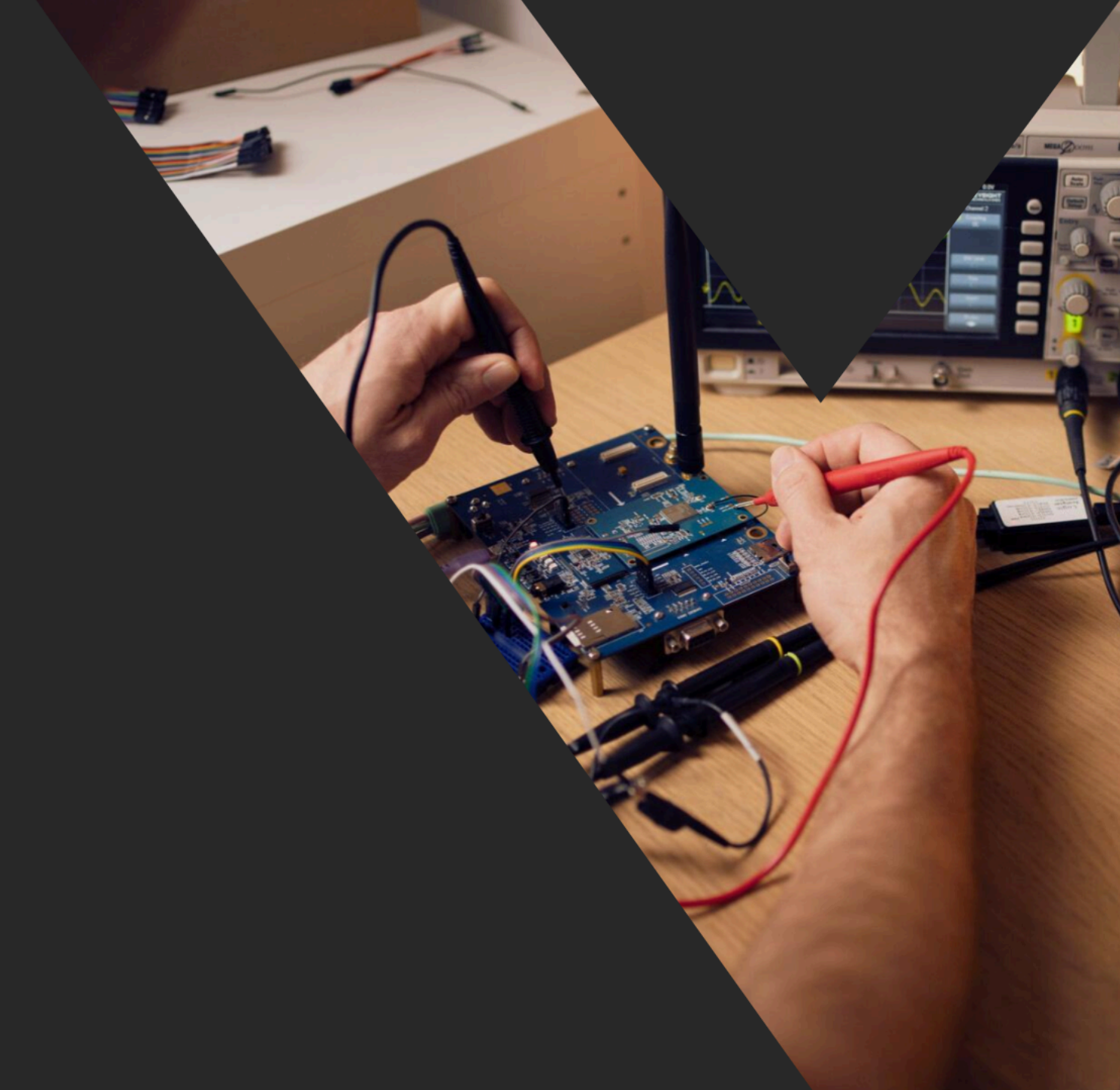
Narzędzia i rozwój

- System logowania i śledzenia błędów
- Obsługa aktualizacji OTA
- Testowanie, symulowanie, debugowanie

Co jeszcze jest w Zephyrze?



Hello world!



Pierwsza aplikacja - main.cpp

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

#define LED_NODE DT_ALIAS(led0)
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(1000);
    }
}
```

Pierwsza aplikacja - main.cpp

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

#define LED_NODE DT_ALIAS(led0)
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(1000);
    }
}
```

Pierwsza aplikacja - main.cpp

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

#define LED_NODE DT_ALIAS(led0)
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(1000);
    }
}
```

Pierwsza aplikacja - main.cpp

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

#define LED_NODE DT_ALIAS(led0)
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(1000);
    }
}
```

Pierwsza aplikacja - CMakeLists

```
cmake_minimum_required(VERSION 3.20.0)

find_package(Zephyr REQUIRED)
project(my_zephyr_app)

target_sources(app PRIVATE src/main.cpp)
```

Pierwsza aplikacja - prj.conf

```
CONFIG_GPIO=y
```

Pierwsza aplikacja - west.yml

Czym jest West?

Czym jest West?

- **West** to narzędzie do zarządzania projektami Zephyr.
- Pozwala na:
  - zarządzanie repozytoriami,
  - budowanie kodu,
  - flashowanie, debugowanie.
- Pozwala na moduły rozszerzające jego funkcjonalność.

Pierwsza aplikacja - west.yml

```
manifest:
  version: 0.13
  projects:
    - name: zephyr
      url: https://github.com/zephyrproject-rtos/zephyr
      revision: v3.7.0
      import:
        name-allowlist:
          - cmsis
          - hal_nordic
          - hal_stm32
        path-prefix: deps
```

Pierwsza aplikacja - west.yml

```
manifest:
  version: 0.13
  projects:
    - name: zephyr
      url: https://github.com/zephyrproject-rtos/zephyr
      revision: v3.7.0
      import:
        name-allowlist:
          - cmsis
          - hal_nordic
          - hal_stm32
        path-prefix: deps
```

Pierwsza aplikacja - west.yml

```
manifest:
  version: 0.13
  projects:
    - name: zephyr
      url: https://github.com/zephyrproject-rtos/zephyr
      revision: v3.7.0
      import:
        name-allowlist:
          - cmsis
          - hal_nordic
          - hal_stm32
        path-prefix: deps
```

Pierwsza aplikacja - west.yml

```
manifest:
  version: 0.13
  projects:
    - name: zephyr
      url: https://github.com/zephyrproject-rtos/zephyr
      revision: v3.7.0
      import:
        name-allowlist:
          - cmsis
          - hal_nordic
          - hal_stm32
        path-prefix: deps
```

Pierwsza aplikacja - west.yml

```
manifest:
  version: 0.13
  projects:
    - name: zephyr
      url: https://github.com/zephyrproject-rtos/zephyr
      revision: v3.7.0
      import:
        name-allowlist:
          - cmsis
          - hal_nordic
          - hal_stm32
        path-prefix: deps
```

Skonfigurujmy workspace

```
west init -l app
west update
west zephyr-export
pip3 install -r deps/zephyr/scripts/requirements.txt
source deps/zephyr/zephyr-env.sh
```

Skonfigurujmy workspace

```
west init -l app
west update
west zephyr-export
pip3 install -r deps/zephyr/scripts/requirements.txt
source deps/zephyr/zephyr-env.sh
```

Skonfigurujmy workspace

```
west init -l app
west update
west zephyr-export
pip3 install -r deps/zephyr/scripts/requirements.txt
source deps/zephyr/zephyr-env.sh
```

Skonfigurujmy workspace

```
west init -l app
west update
west zephyr-export
pip3 install -r deps/zephyr/scripts/requirements.txt
source deps/zephyr/zephyr-env.sh
```

Zbudujmy i wgrajmy pierwszą aplikację

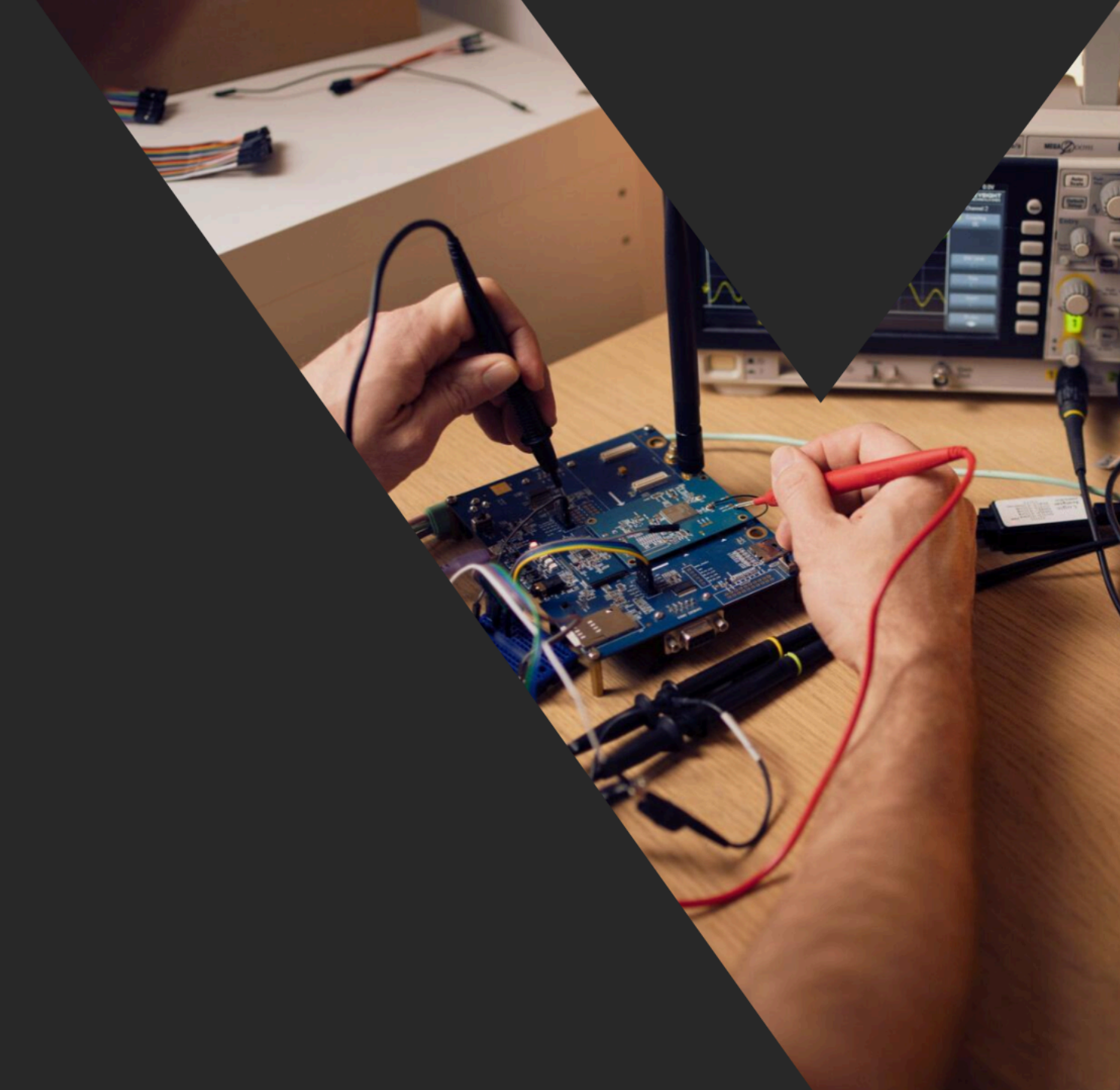
- Budowanie na nRF52DK

```
west build --board nrf52dk/nrf52832 app --pristine
```

- Flashowanie

```
west flash --runner pyocd
```

Demo



Czy da się bez westa?

- **Git** – do pobrania kodu źródłowego Zephyr i modułów.
- Konfiguracja środowiska:

```
export ZEPHYR_BASE=$(pwd)
source zephyr-env.sh
```

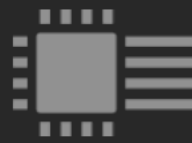
- **CMake** + **Ninja/Make** – do budowania projektu.

```
cmake -DBOARD=nrf52dk_nrf52832 -GNinja .. && ninja
```

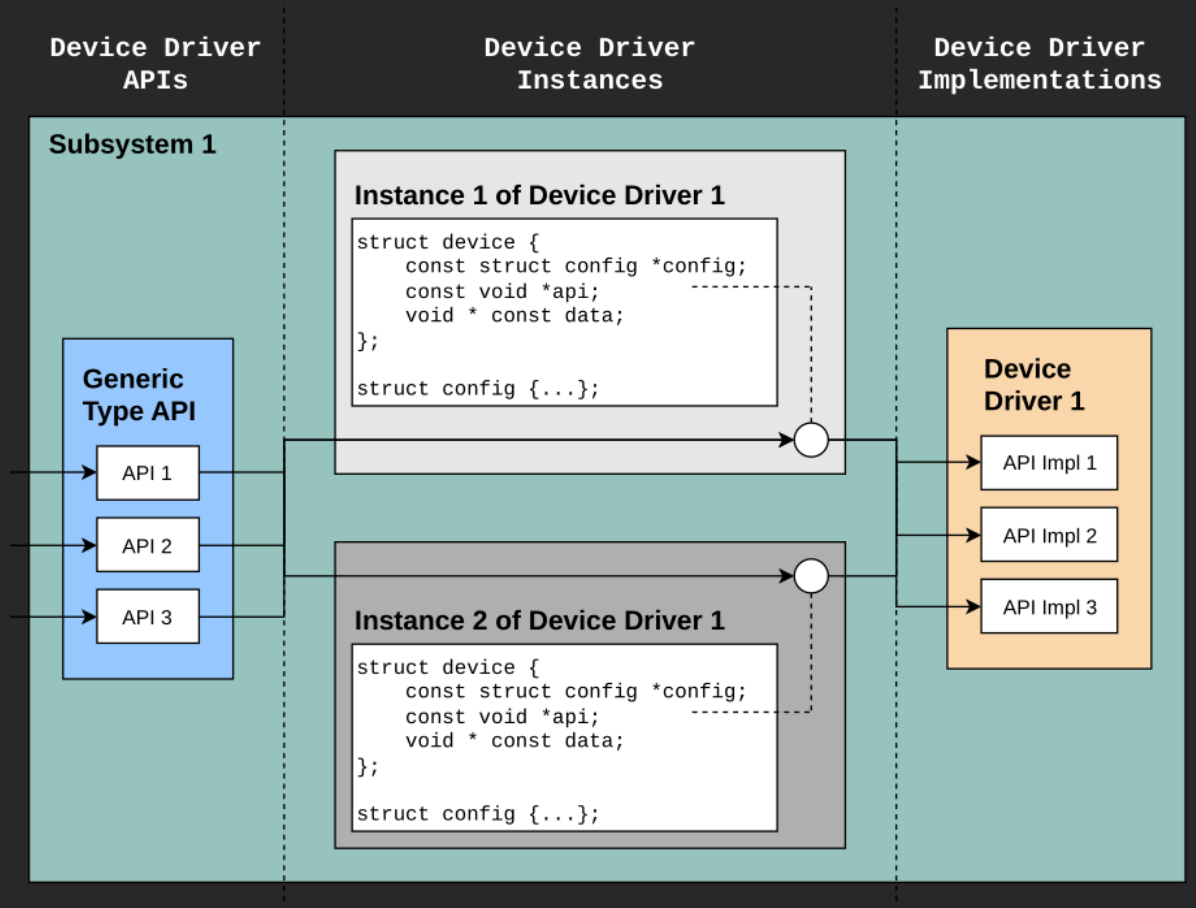
Dlaczego zadziałało na Nucleo?

- Kod korzysta z **abstrakcyjnych API Zephyra** (GPIO).
- Zephyr wybiera odpowiednie drivery na podstawie `BOARD`.

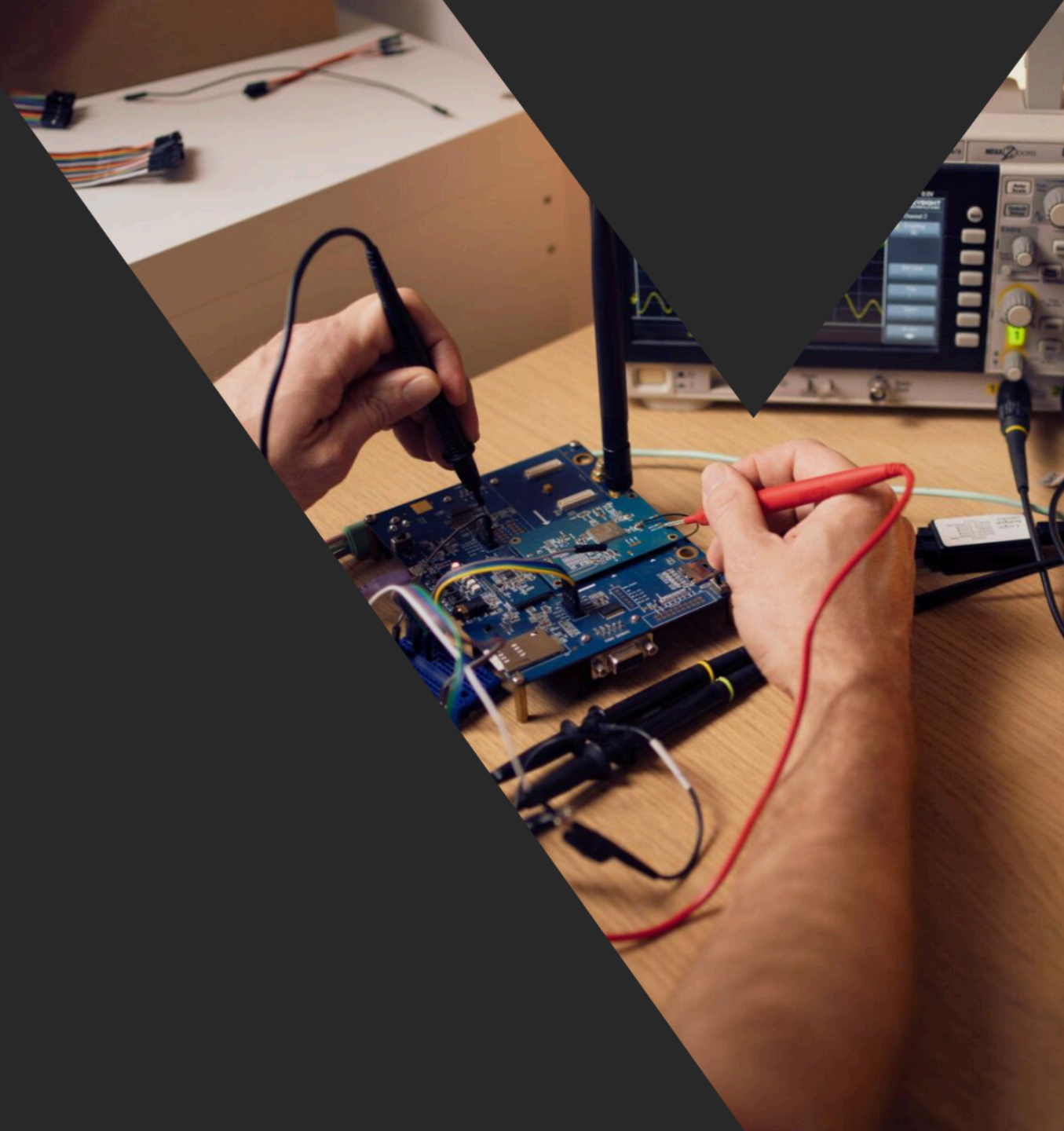
```
#define LED_NODE DT_ALIAS(led0)
struct gpio_dt_spec led = GPIO_DT_SPEC_GET(LED_NODE, gpios);
```

Dlaczego zadziałało na Nucleo?



Prawdziwe Hello World



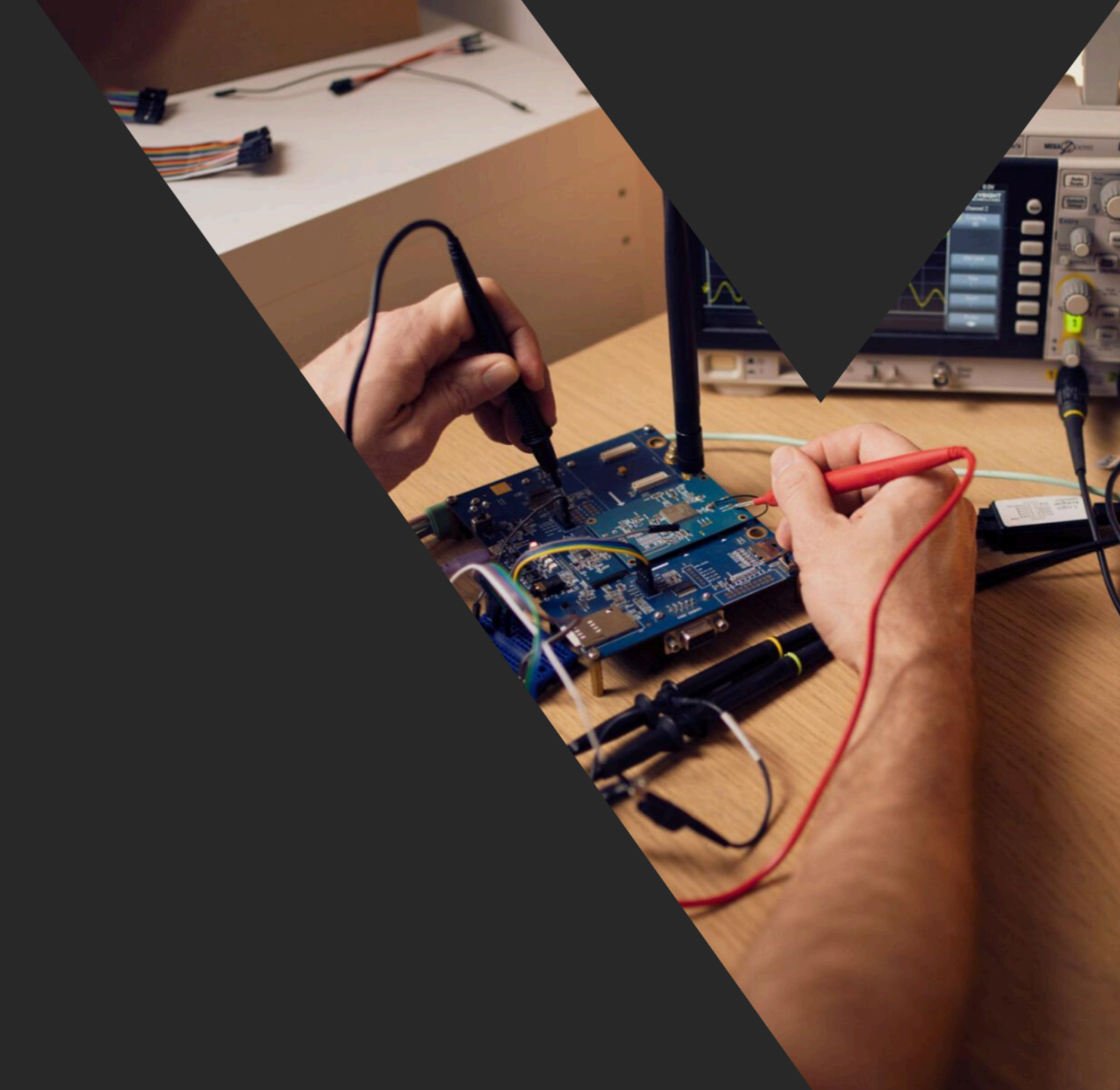
Konfiguracja w prj.conf

```
CONFIG_LOG=y
```

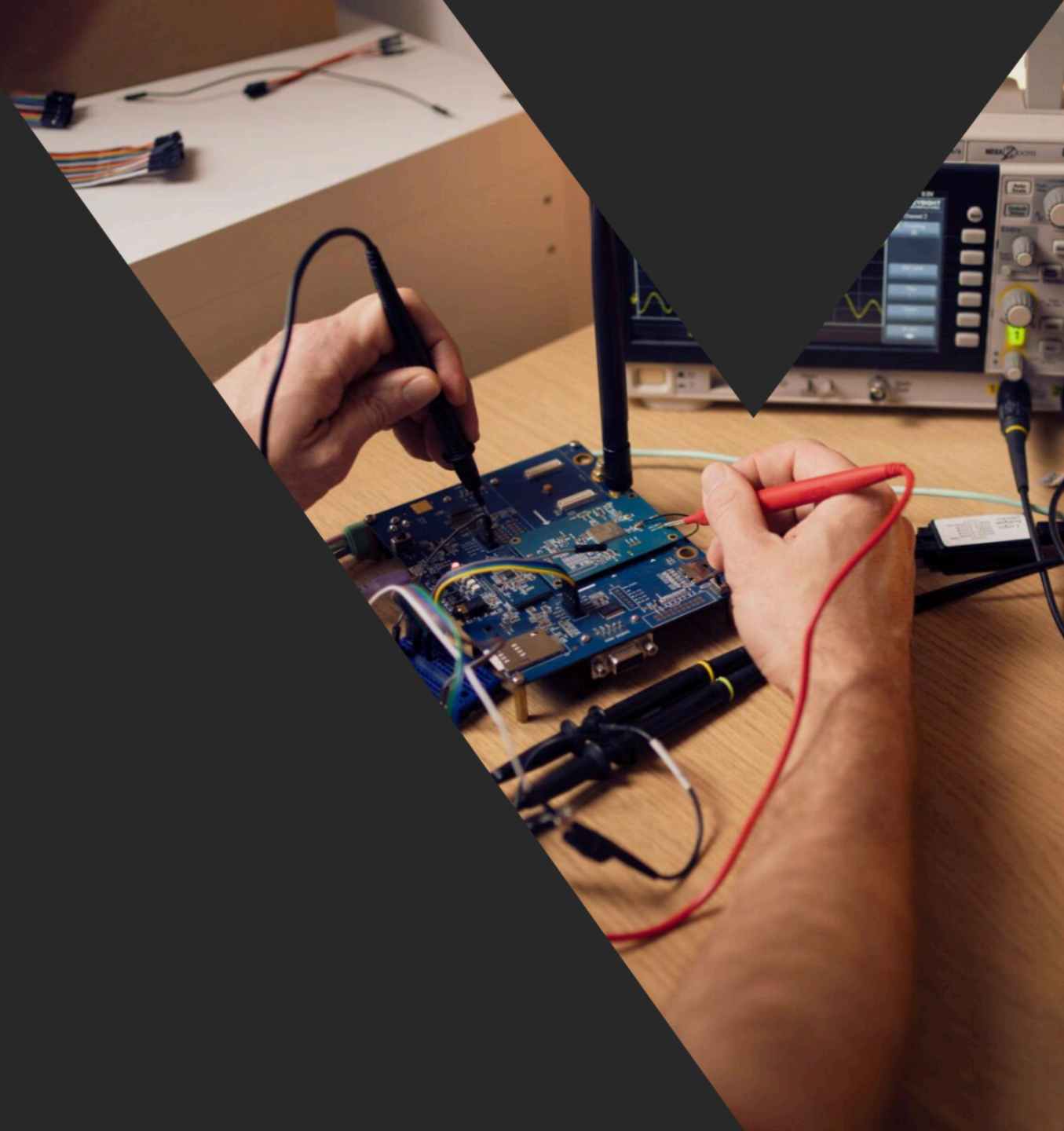
Modyfikacja main.cpp

```
// ...
#include <zephyr/logging/log.h>
// ...
LOG_MODULE_REGISTER(main, LOG_LEVEL_INF);
int main(void) {
    // ...
    while (1) {
        //...
        LOG_INF("LED state: %s", led_state ? "ON" : "OFF");
        k_msleep(1000);
    }
}
```

Demo



Obsługa poprzez Shella

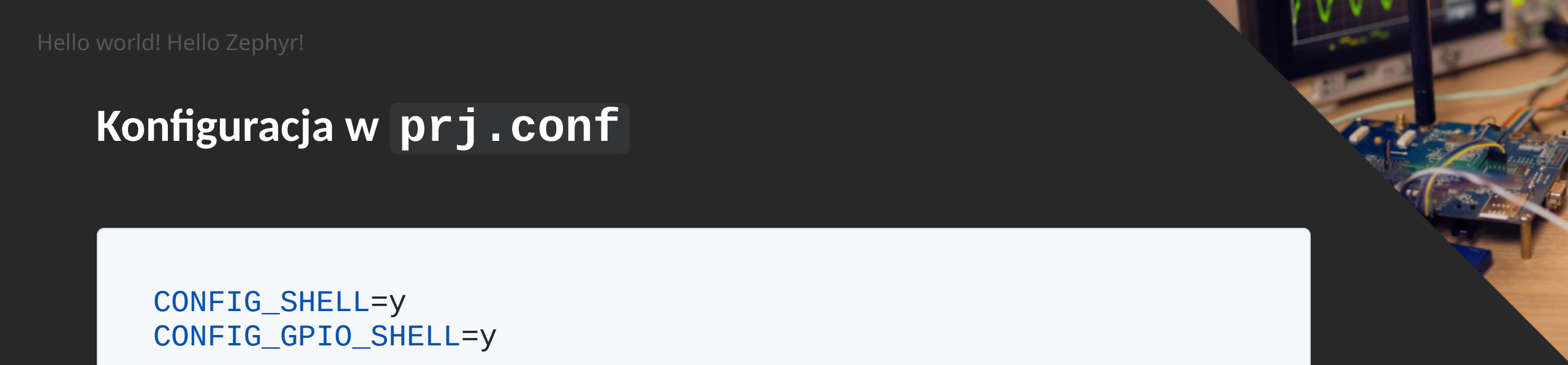


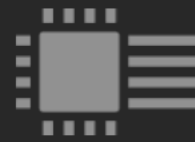
W czym shell może nam pomóc?

- Tworzenie własnych komend do własnych modułów.
- Diagnostyka i testowanie peryferiów bez zmiany kodu.
- Dynamiczna konfiguracja aplikacji w czasie działania.
- Monitorowanie logów i statusu systemu.
- Zdalne sterowanie urządzeniem przez UART.

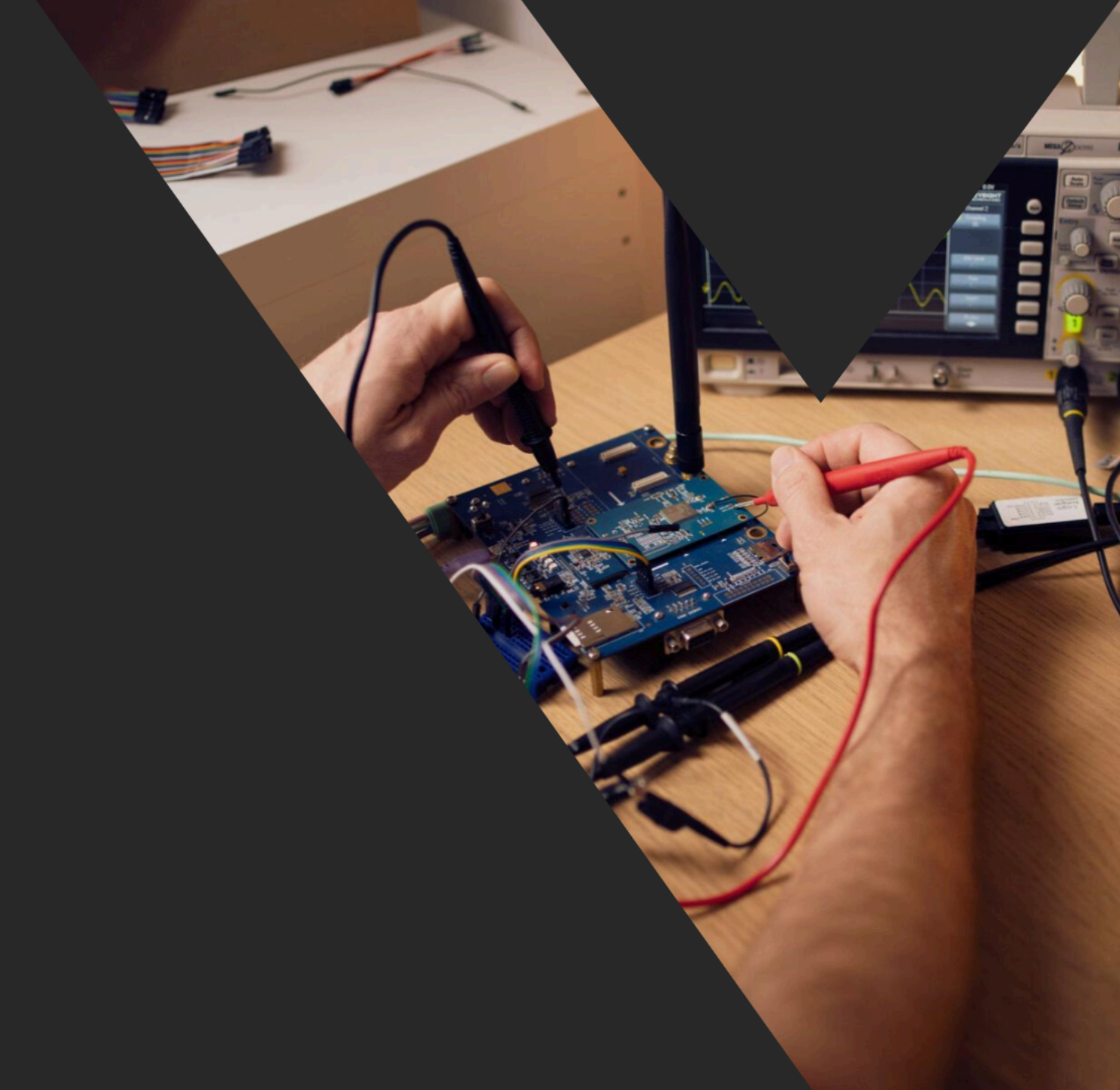
Konfiguracja w `prj.conf`

```
CONFIG_SHELL=y
CONFIG_GPIO_SHELL=y
```

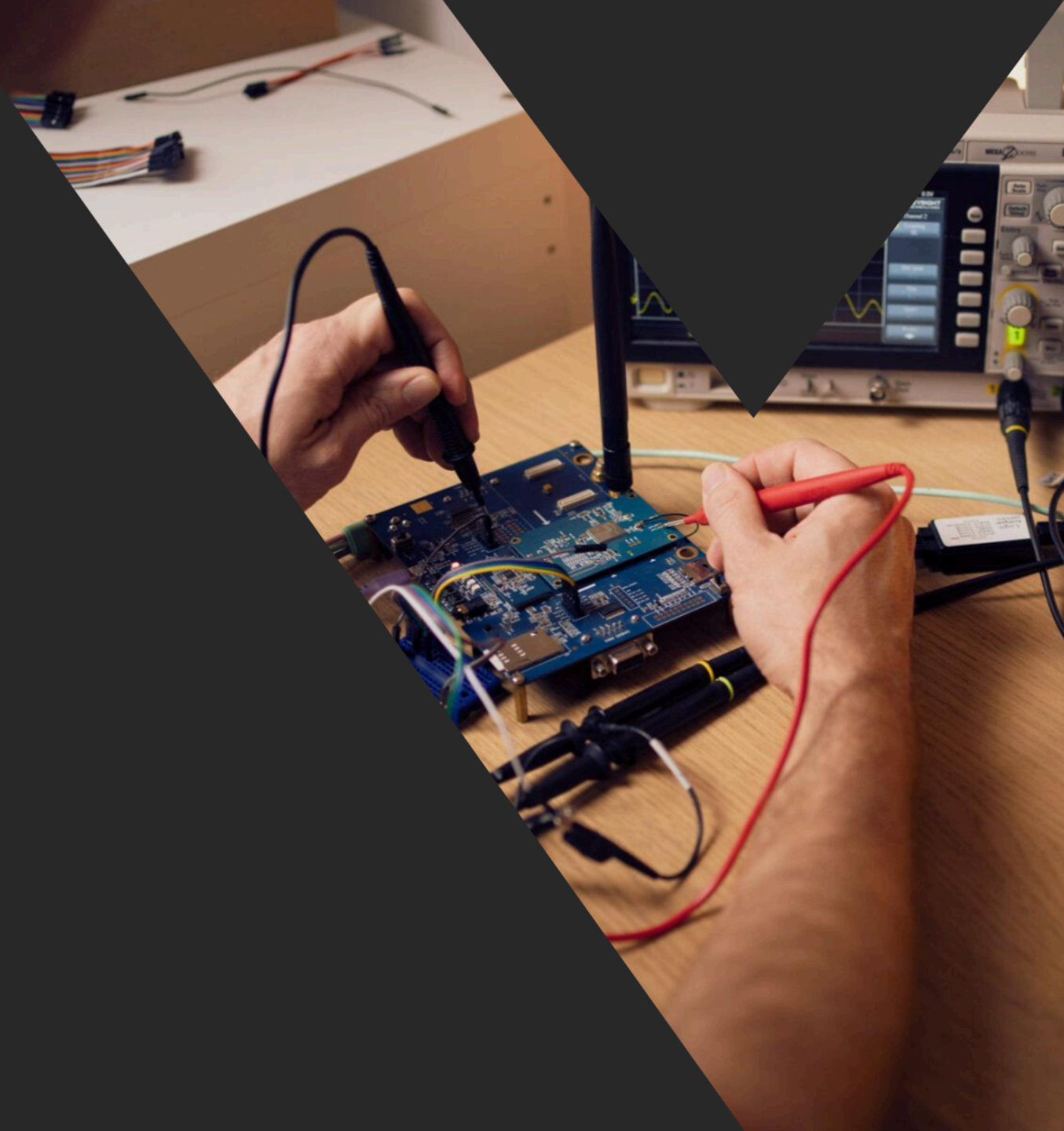




Demo



Tworzenie nowej płytki



Co zrobić gdy chcemy własną płytkę?

- **Nowa sytuacja:** chcemy dodać obsługę własnego sprzętu.
- **Definicja płytki –** konfiguracja sprzętowa i zasoby.
- **Tworzenie plików DTS, Kconfig, CMake i overlay.**

Definicja płytki - board.yml

```
# app/boards/arm/goodbyte_demo/board.yml
board:
  name: goodbyte_demo
  vendor: arm
  socs:
  - name: nrf52832
```

Tworzenie nowej płytki - DTS

```
// app/boards/arm/goodbyte_demo/goodbyte_demo.dts
/dts-v1/;
#include <nordic/nrf52832_qfaa.dtsi>
/ {
    model = "A goodbyte demo board";
    compatible = "goodbyte,goodbyte-board";
    chosen {
        zephyr,sram = &sram0;
        zephyr,flash = &flash0;
    };
    // ...
};
//...
```

Tworzenie nowej płytki - DTS

```
// app/boards/arm/goodbyte_demo/goodbyte_demo.dts
/dts-v1/;
#include <nordic/nrf52832_qfaa.dtsi>
/ {
    model = "A goodbyte demo board";
    compatible = "goodbyte,goodbyte-board";
    chosen {
        zephyr,sram = &sram0;
        zephyr,flash = &flash0;
    };
    // ...
};
//...
```

Tworzenie nowej płytki - DTS

```
// app/boards/arm/goodbyte_demo/goodbyte_demo.dts
/dts-v1/;
#include <nordic/nrf52832_qfaa.dtsi>
/ {
    model = "A goodbyte demo board";
    compatible = "goodbyte,goodbyte-board";
    chosen {
        zephyr,sram = &sram0;
        zephyr,flash = &flash0;
    };
    // ...
};
//...
```

Tworzenie nowej płytki - DTS

```
// app/boards/arm/goodbyte_demo/goodbyte_demo.dts
/dts-v1/;
#include <nordic/nrf52832_qfaa.dtsi>
/ {
    model = "A goodbyte demo board";
    compatible = "goodbyte,goodbyte-board";
    chosen {
        zephyr,sram = &sram0;
        zephyr,flash = &flash0;
    };
    // ...
};
//...
```

Tworzenie nowej płytki - DTS

```
// app/boards/arm/goodbyte_demo/goodbyte_demo.dts
// ...
/ {
    // ...
    leds {
        compatible = "gpio-leds";
        led1_label: led_1 {
            gpios = <&gpio0 17 GPIO_ACTIVE_LOW>;
            label = "LED 1";
        };
    };
};
//...
```

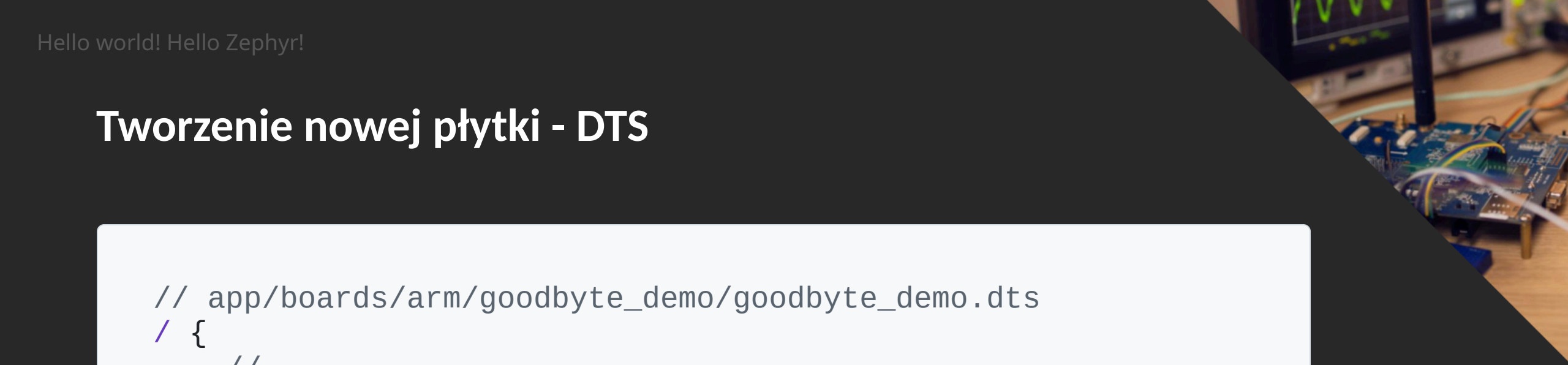
Tworzenie nowej płytki - DTS

```
// app/boards/arm/goodbyte_demo/goodbyte_demo.dts
// ...
/ {
    // ...
    leds {
        compatible = "gpio-leds";
        led1_label: led_1 {
            gpios = <&gpio0 17 GPIO_ACTIVE_LOW>;
            label = "LED 1";
        };
    };
};
//...
```

Tworzenie nowej płytki - DTS

```
// app/boards/arm/goodbyte_demo/goodbyte_demo.dts
// ...
/ {
    // ...
    leds {
        compatible = "gpio-leds";
        led1_label: led_1 {
            gpios = <&gpio0 17 GPIO_ACTIVE_LOW>;
            label = "LED 1";
        };
    };
};
//...
```

Tworzenie nowej płytki - DTS



```
// app/boards/arm/goodbyte_demo/goodbyte_demo.dts
/ {
    //...
};
&gpiote {
    status = "okay";
};
&gpio0 {
    status = "okay";
};
```

Zmiana w DTS poprzez overlay?

- **Zmiana konfiguracji sprzętowej bez ingerencji w kod źródłowy.**
- **Pliki overlay (`.overlay`)** pozwalają na dostosowanie ustawień do konkretnej aplikacji.

```
// app/boards/goodbyte_demo.overlay
/ {
    aliases {
        led0 = &led1_label;
    };
};
```

Tworzenie nowej płytki - Kconfig

- Konfiguracja, jakie opcje są dostępne dla nowej płytki.
- Definiowanie zgodności z konkretnym SoC.

```
# app/boards/arm/goodbyte_demo/Kconfig.goodbyte_demo
config BOARD_GOODBYTE_DEMO
    select SOC_NRF52832_QFAA
```

Tworzenie nowej płytki - CMake

- Integracja nowej płytki z build systemem.

```
board_runner_args(jlink "--device=nRF52832_xxAA")
board_runner_args(pyocd "--target=nrf52832")
set(BOARDS_COMMON ${ZEPHYR_BASE}/boards/common)
include(${BOARDS_COMMON}/nrfjprog.board.cmake)
include(${BOARDS_COMMON}/nrfutil.board.cmake)
include(${BOARDS_COMMON}/jlink.board.cmake)
include(${BOARDS_COMMON}/pyocd.board.cmake)

set(OPENOCD_NRF5_SUBFAMILY "nrf52")
include(${BOARDS_COMMON}/openocd-nrf5.board.cmake)
```

Tworzenie nowej płytki - CMake

- Integracja nowej płytki z build systemem.

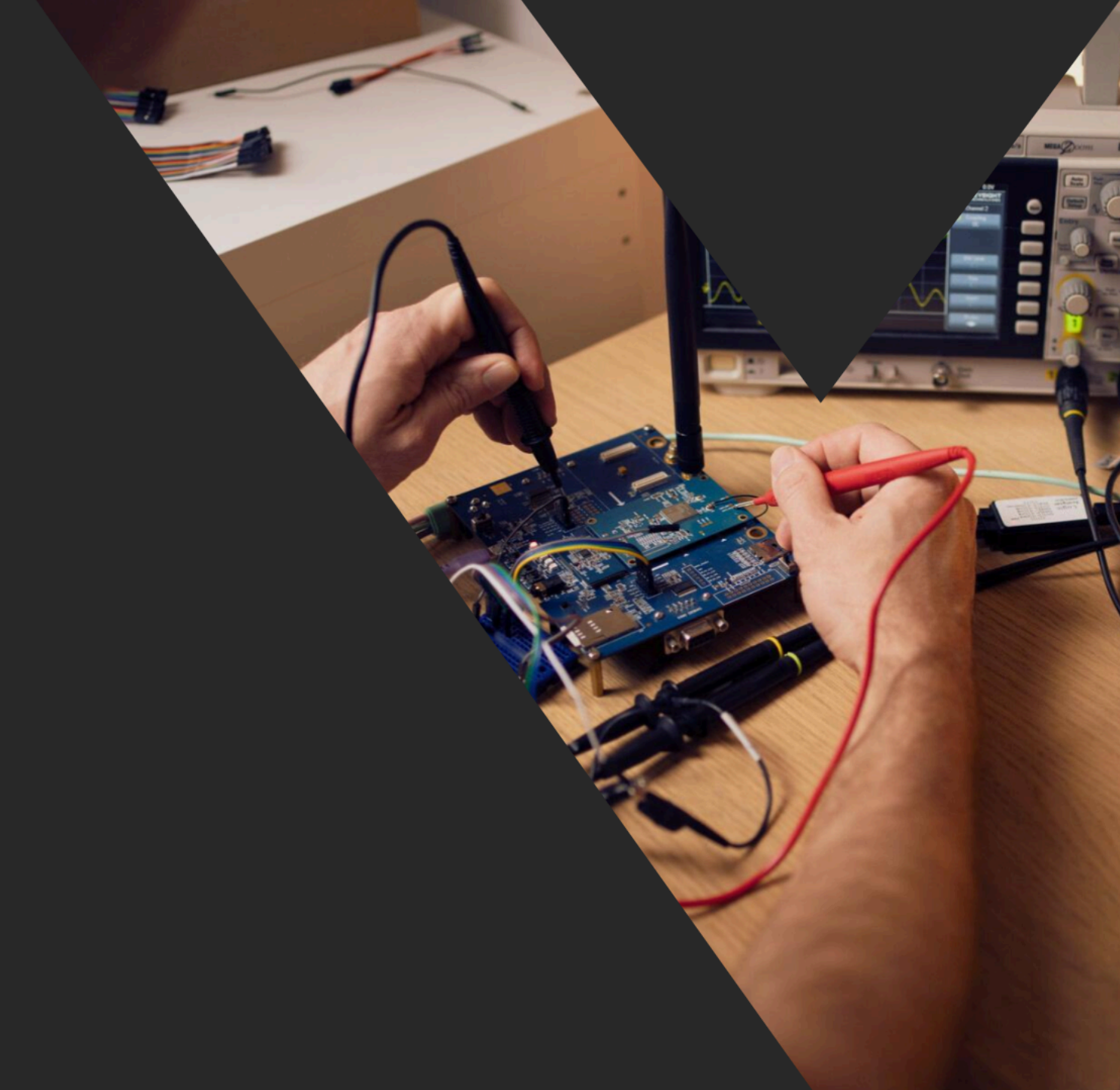
```
board_runner_args(jlink "--device=nRF52832_xxAA")
board_runner_args(pyocd "--target=nrf52832")
set(BOARDS_COMMON ${ZEPHYR_BASE}/boards/common)
include(${BOARDS_COMMON}/nrfjprog.board.cmake)
include(${BOARDS_COMMON}/nrfutil.board.cmake)
include(${BOARDS_COMMON}/jlink.board.cmake)
include(${BOARDS_COMMON}/pyocd.board.cmake)

set(OPENOCD_NRF5_SUBFAMILY "nrf52")
include(${BOARDS_COMMON}/openocd-nrf5.board.cmake)
```

Tworzenie nowej płytki - CMake

- Integracja nowej płytki z build systemem.

```
board_runner_args(jlink "--device=nRF52832_xxAA")
board_runner_args(pyocd "--target=nrf52832")
set(BOARDS_COMMON ${ZEPHYR_BASE}/boards/common)
include(${BOARDS_COMMON}/nrfjprog.board.cmake)
include(${BOARDS_COMMON}/nrfutil.board.cmake)
include(${BOARDS_COMMON}/jlink.board.cmake)
include(${BOARDS_COMMON}/pyocd.board.cmake)
set(OPENOCD_NRF5_SUBFAMILY "nrf52")
include(${BOARDS_COMMON}/openocd-nrf5.board.cmake)
```

Demo



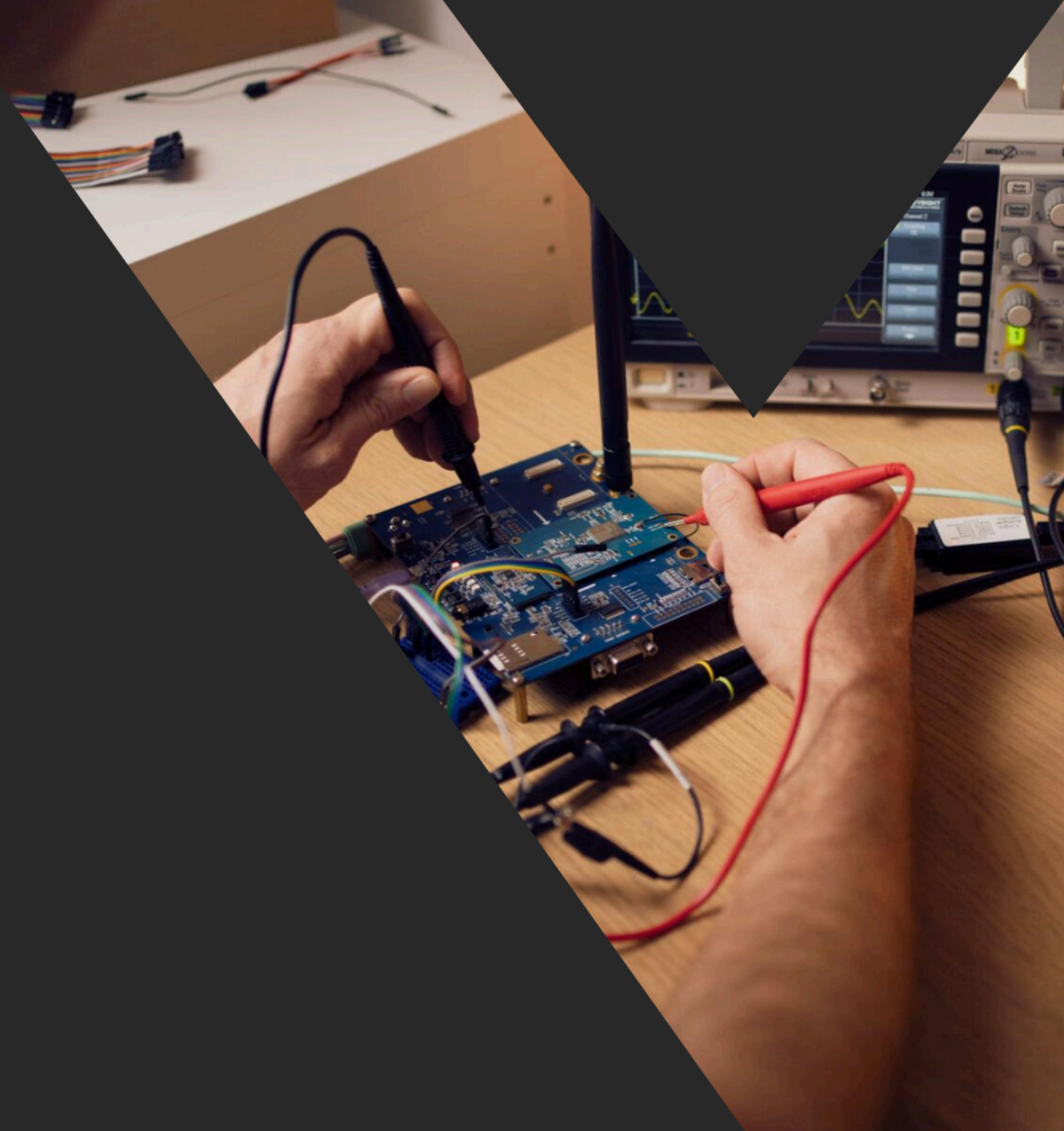
Jakie wartości/korzyści widzimy w Zephyrze?

- **Modularność**.
- **Obsługa wielu architektur –** ARM, RISC-V, x86 i inne.
- **Silne wsparcie społeczności i firm.**
- **Wbudowane wsparcie dla sterowników i protokołów –** Bluetooth, USB, TCP/IP.

Czy jest aż tak różowo? Czyli grillowanko

- **Krzywa uczenia się.**
- **Skomplikowana konfiguracja**
- **West – konieczność czy ułatwienie?**
- **Dostępność dokumentacji** – wciąż rozwijana, ale nie zawsze kompletna.

Czas na pytania



Dziękuję za uwagę! Repozytorium z demo poniżej: