Crabs Flying Kites

Writing A Zephyr Application In Rust

Mohammed Billoo

MAB Labs Embedded Solutions

Open Source Summit EU 25, Amsterdam

Agenda

- Motivation
- Getting started with Rust in Zephyr
- Hello World Demo
- Code Review
- **Summary + Next Steps**

Mohammed Billoo

mab@mab-labs.com

/mab-embedded

@mabembedded

THE SPEAKER

- Embedded Software Consultant (NYC, USA)
- Design Work
  - Medical Devices
  - LIDAR
  - Custom ASICs
- Expertise
  - Zephyr
  - Embedded Linux/Yocto

yocto PROJECT

www.mab-labs.com

TRAINING/WORKSHOPS

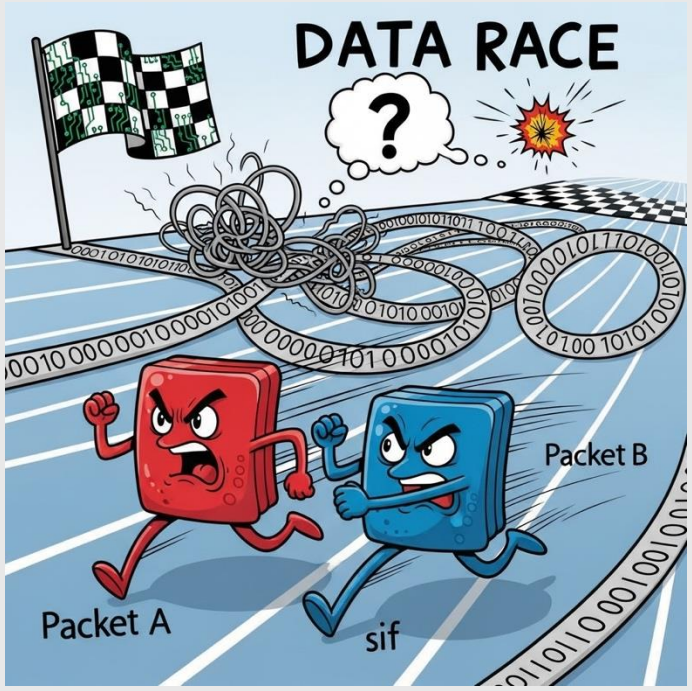
MOTIVATION

Zephyr Memory Related Issues

- Personally encountered
  - Data race
  - Stack overflow
- Data race
  - Multiple threads
  - Both accessing same resource (variable)
  - No synchronization

Zephyr Memory Related Issues

- Data Race
- Have (incorrect) synchronization
- K_NO_WAIT vs K_FOREVER
- Choosing the wrong one can affect entire application
- Problem gets worse if have multiple threads



Zephyr Memory Related Issues

- Stack Overflow
- Misconception encountered
  - West shows Flash and RAM use after compilation
  - Complains if either is overused
  - Doesn’t that eliminate the possibility of encountering a stack overflow?

```
[6/6] Linking C executable zephyr/zephyr.elf
Memory region         Used Size  Region Size  %age Used
           FLASH:       44688 B         1 MB      4.26%
             RAM:       11136 B        96 KB     11.33%
        IDT_LIST:          0 GB        32 KB      0.00%
```

Zephyr Memory Related Issues

/home/mab/zephyr-sdk-0.16.4/arm-zephyr-eabi/bin/../lib/gcc/arm-zephyr-eabi/12.2.0/../../../../arm-zephyr-eabi/bin/ld.bfd: zephyr/zephyr_pre0.elf section `noinit' will not fit in region `RAM'

/home/mab/zephyr-sdk-0.16.4/arm-zephyr-eabi/bin/../lib/gcc/arm-zephyr-eabi/12.2.0/../../../../arm-zephyr-eabi/bin/ld.bfd: region `RAM' overflowed by 4544 bytes

collect2: error: ld returned 1 exit status

ninja: build stopped: subcommand failed.

Zephyr Memory Related Issues

- Stack Overflow
- Almost **everything** is configured with a stack size
  - Main thread
  - Other threads
  - Workqueue
  - Interrupt service routine
- Don’t find out until runtime if blow through stack
  - West debug ➔ hard fault handler
  - Can use stack canary

Zephyr Memory Related Issues

- Other?
- Try to avoid dynamically allocating memory
- But if needed, use k_malloc in Zephyr
  - CONFIG_HEAP_MEM_POOL_SIZE
- You are responsible for freeing dynamically allocated memory
- You are responsible for not using memory after freeing it
  - **Use after free**

Rust + Embedded

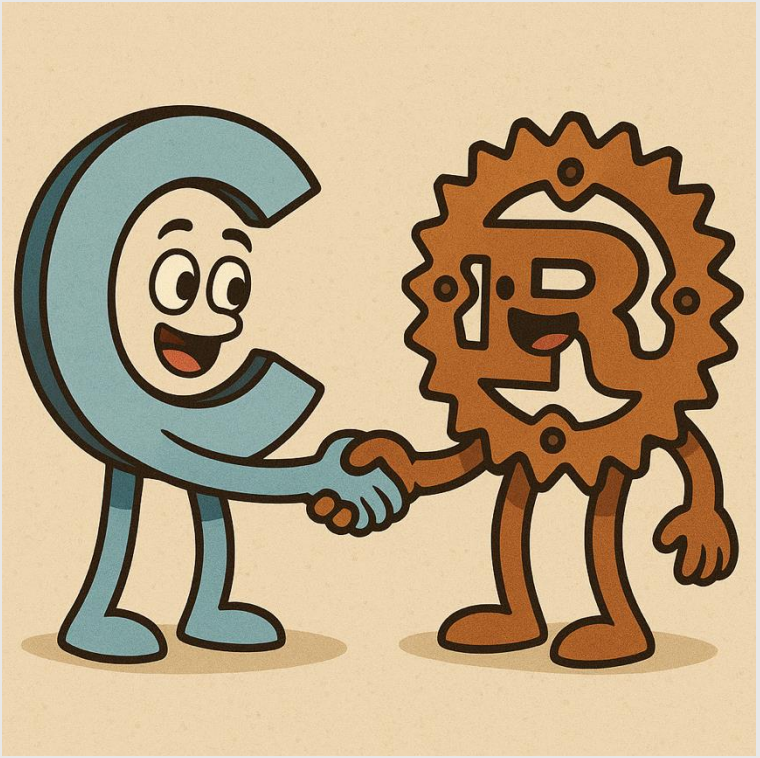
- Rust helps address issues outlined above
  - Data races
  - Stack overflow
  - Memory management
- Memory ownership model
  - Tracks memory ownership
  - Frees memory once determined it is no longer used
  - Ownership “transferred” according to a set of rules

Rust + Embedded

- Memory ownership model
  - If access memory after ownership has been transferred, compilation error!
  - If modifying data across threads in an **unsafe** manner, compilation error!
  - Forces us to think about problems sooner rather than later
    - During compile time vs during run time
- "**Oh dear, I have to redesign how these threads interact**"
  - Nicer to know during compilation vs when devices out in the field

Rust + Embedded

- “Wait, is there another version of Zephyr written entirely in Rust”?
- **NO!**
- Rust and C can coexist
  - Foreign Function Interface (FFI)
  - Common paradigm to go Rust 🡐 🡒 C
- Identify how to best expose C side to Rust
  - This is the job of Zephyr maintainers/community
  - Not easy



Rust + Embedded

- **Beware**
  - **Rust is not a magical bullet that can solve ALL problems**
  - **”If we switch to Rust, it will solve our performance issues”**
  - **“When you have a hammer, not everything is a nail”**

GETTING STARTED WITH RUST IN ZEPHYR

Environment Setup

- Install rust (obviously)

```
$> curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

- Install clang
  - Needed to establish interface between C and Rust

```
$> sudo apt install clang
```

- Setup Python virtual environment and install dependencies

```
$> python3 –m venv .env
$> source .env/bin/activate
$> pip3 install west pyelftools intelhex
```

Zephyr + Rust

- Retrieve Zephyr v.4.2.0

```
$> west init -m https://github.com/zephyrproject-rtos/zephyr.git --mr v4.2.0
```

- Add Rust support

```
$> west config manifest.project-filter +zephyr-lang-rust
$> west update
```

- Build hello world!

```
$> cd modules/lang/rust
$> west build -b disco_l475_iot1 samples/hello_world/
```

Zephyr + Rust

- Success?
- **NO!**

```
     |
2562 -                          let device = super::super::super::get_instance_raw();
2562 +                          let device = get_instance_raw();
     |

For more information about this error, try `rustc --explain E0425`.
error: could not compile `zephyr` (lib) due to 3 previous errors
FAILED: always-run-cargo.dummy rust/target/thumbv7em-none-eabi/release/librustapp.a rust/wrapper.c /home/mab/zephyr-rust/modules/lang/rust/build/alway
s-run-cargo.dummy /home/mab/zephyr-rust/modules/lang/rust/build/rust/target/thumbv7em-none-eabi/release/librustapp.a /home/mab/zephyr-rust/modules/lan
g/rust/build/rust/wrapper.c
cd /home/mab/zephyr-rust/modules/lang/rust/samples/hello_world && /usr/bin/cmake -E env BUILD_DIR=/home/mab/zephyr-rust/modules/lang/rust/build ZEPHYR
_BASE=/home/mab/zephyr-rust/zephyr DOTCONFIG=/home/mab/zephyr-rust/modules/lang/rust/build/zephyr/.config ZEPHYR_DTS=/home/mab/zephyr-rust/modules/lan
g/rust/build/zephyr/zephyr.dts INCLUDE_DIRS="/home/mab/zephyr-rust/modules/lang/rust/build/zephyr/include/generated/zephyr /home/mab/zephyr-rust/zephy
r/include /home/mab/zephyr-rust/modules/lang/rust/build/zephyr/include/generated /home/mab/zephyr-rust/zephyr/soc/st/stm32 /home/mab/zephyr-rust/zephy
r/lib/libc/common/include /home/mab/zephyr-rust/zephyr/soc/st/stm32/common/. /home/mab/zephyr-rust/zephyr/drivers /home/mab/zephyr-rust/zephyr/soc/st/
stm32/stm32l4x/. /home/mab/zephyr-rust/modules/hal/cmsis/CMSIS/Core/Include /home/mab/zephyr-rust/modules/hal/cmsis_6/CMSIS/Core/Include /home/mab/zep
hyr-rust/zephyr/modules/cmsis_6/. /home/mab/zephyr-rust/modules/hal/stm32/stm32cube/stm32l4xx/soc /home/mab/zephyr-rust/modules/hal/stm32/stm32cube/st
m32l4xx/drivers/include /home/mab/zephyr-rust/modules/hal/stm32/stm32cube/common_ll/include /home/mab/zephyr-rust/modules/hal/ti/mspm0/source/ti/devic
es/msp/. /home/mab/zephyr-rust/modules/hal/ti/mspm0/source/ti/devices/msp/m0p /home/mab/zephyr-rust/modules/hal/ti/mspm0/source/ti/devices/msp/periphe
rals /home/mab/zephyr-rust/modules/hal/ti/mspm0/source/ti/devices/msp/peripherals/m0p /home/mab/zephyr-rust/modules/hal/ti/mspm0/source/ti/devices/msp
/peripherals/m0p/sysctl " INCLUDE_DEFINES="KERNEL __ZEPHYR__=1  __LINUX_ERRNO_EXTENSIONS__ PICOLIBC_LONG_LONG_PRINTF_SCANF __PROGRAM_START __PROGRAM_S
TART STM32L475xx USE_HAL_DRIVER USE_FULL_LL_DRIVER CORE_CM4 HSE_VALUE=8000000 K_HEAP_MEM_POOL_SIZE=0" WRAPPER_FILE="/home/mab/zephyr-rust/modules/lang
/rust/build/rust/wrapper.c" DT_AUGMENTS="/home/mab/zephyr-rust/modules/lang/rust/dt-rust.yaml" BINARY_DIR_INCLUDE_GENERATED="/home/mab/zephyr-rust/mod
ules/lang/rust/build/zephyr/include/generated/zephyr" cargo build --release --config patch.crates-io.zephyr.path=\"/home/mab/zephyr-rust/modules/lang/
rust/zephyr\" --config patch.crates-io.zephyr-build.path=\"/home/mab/zephyr-rust/modules/lang/rust/zephyr-build\" --config patch.crates-io.zephyr-sys.
path=\"/home/mab/zephyr-rust/modules/lang/rust/zephyr-sys\" --target thumbv7em-none-eabi --target-dir /home/mab/zephyr-rust/modules/lang/rust/build/ru
st/target
ninja: build stopped: subcommand failed.
```

Zephyr + Rust

- Check out zephyr-lang-rust at latest main branch

```
$> git fetch
$> git checkout main
```

- Now builds, yay!
  - Remember, still WIP
  - Some gymnastics needed

```
(.env) mab@mab-HP-Laptop-14-fq1xxx:~/zephyr-rust/modules/lang/rust$ west build -b disco_l475_iot1 samples/hello_world/
[0/11] Building Rust application
   Compiling zephyr-build v0.1.0 (/home/mab/zephyr-rust/modules/lang/rust/zephyr-build)
   Compiling zephyr-sys v0.1.0 (/home/mab/zephyr-rust/modules/lang/rust/zephyr-sys)
   Compiling zephyr v0.1.0 (/home/mab/zephyr-rust/modules/lang/rust/zephyr)
   Compiling rustapp v0.1.0 (/home/mab/zephyr-rust/modules/lang/rust/samples/hello_world)
    Finished `release` profile [optimized] target(s) in 1.56s
[11/11] Linking C executable zephyr/zephyr.elf
Memory region         Used Size  Region Size  %age Used
           FLASH:       45388 B         1 MB      4.33%
             RAM:       11136 B        96 KB     11.33%
        IDT_LIST:          0 GB        32 KB      0.00%
Generating files from /home/mab/zephyr-rust/modules/lang/rust/build/zephyr/zephyr.elf for board: disco_l475_iot1
```

HELLO WORLD DEMO

Hardware

- STM32 BL475 IOT board (~$50)
- https://www.st.com/en/evaluation-tools/b-l475e-iot01a.html
- STM32L4 ultra-low power MCU
- Many peripherals
  - BLE
  - Sub-G RF
  - WiFi
  - NFC
  - Microphone
  - (Accelerometer + Gyro), Magnetometer
  - ToF sensor, barometer, temperature, ….

Demo

CODE REVIEW

Hello World Sample

```
# Copyright (c) 2024 Linaro LTD
# SPDX-License-Identifier: Apache-2.0

# The default 1k stack isn't large enough for rust string formatting with logging.
CONFIG_MAIN_STACK_SIZE=4096

CONFIG_RUST=y

CONFIG_RUST_ALLOC=y
CONFIG_LOG=y

CONFIG_LOG_BACKEND_RTT=n
```

Enables crate for heap-allocated memory (not really required for this sample)

Hello World Sample

- Zephyr divided into three main crates
  - **core**: Standard library without dependencies outside of language and architecture
  - **alloc**: Provides portions of standard library responsible for memory allocation. Requires platform to provide “allocator” implementation
    - Which Zephyr does when CONFIG_RUST_ALLOC=y
  - **std**: Everything else
    - File system, networking
    - Usually not included in Zephyr (or any embedded) application

Hello World Sample

- CMakeLists.txt

```
# SPDX-License-Identifier: Apache-2.0

cmake_minimum_required(VERSION 3.20.0)

find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})

project(hello_rust_world)
rust_cargo_application()
```

Makes the Rust application
“visible” from the C side

Hello World Sample

- src/lib.rs is usual entry point in Rust library crate
- Rust logging is distinct from Zephyr logging
- Some glue logic exists to manage the interface/relationship

Called by C-side

Some glue logic to manage Zephyr/Rust logging

```rust
// Copyright (c) 2024 Linaro LTD
// SPDX-License-Identifier: Apache-2.0

#![no_std]

use log::info;

#[no_mangle]
extern "C" fn rust_main() {
    unsafe {
        zephyr::set_logger().unwrap();
    }

    info!("Hello world from Rust on {}", zephyr::kconfig::CONFIG_BOARD);
}
```

Hello World Sample

- Cargo.toml

Typical Rust

```
# Copyright (c) 2024 Linaro LTD
# SPDX-License-Identifier: Apache-2.0

[package]
# This must be rustapp for now.
name = "rustapp"
version = "0.1.0"
edition = "2021"
description = "A sample hello world application in Rust"
license = "Apache-2.0 or MIT"

[lib]
crate-type = ["staticlib"]

[dependencies]
zephyr = "0.1.0"
log = "0.4.22"
```

Want to compile as library

Include Zephyr crate as dependency (+ Rust log crate)

SUMMARY + NEXT STEPS

Summary

- Understood the value that Rust brings to embedded software applications
  - **NOT GOING TO SOLVE ALL PROBLEMS!**
  - **Does help mitigate nasty problems**
- "Can we replace it all with Rust because of …" -- Your Manager
- Understood how Zephyr intelligently leverages Rust as a force multiplier
  - Use existing crates for task management and synchronization
- Dug into codebase
- Learned how to get started

Next Steps

- Review more complex applications
- Multi-threaded applications and synchronization
  - Leverage Rust “embassy executor” crate
  - Leverage Rust “embassy sync” crate
    - Semaphore, mutex
- Expose Kconfig settings to applications
- Expose Devicetree configurations to applications

Next Steps

- Library and binary crates
  - Fit nicely into Zephyr's testing paradigm
  - Personal best practice
    - Final application source is very minimal (simply initializes a few application libraries)
    - Most implementation contained in application libraries
    - Easily write unit tests
  - Should be able easily leverage that paradigm when using Rust + Zephyr

Next Steps

- Drivers
  - Take a stab at writing device drivers in Rust
  - Have a few PRs open for new device drivers (in C)
  - Might rewrite them in Rust as an experiment

THANK YOU!
QUESTIONS?

Mohammed Billoo

MAB Labs Embedded Solutions

Open Source Summit EU 25, Amsterdam