

# Zephyr @ SMIGHT

Developing firmware for smart grid solutions



# About Me

## Anouar Raddaoui

- Embedded Software Engineer at SMIGHT (since 2020)
- Developing firmware for smart grid devices
- Using Zephyr RTOS since 2020



Embedded Systems



IoT Protocols



Smart Grid

# Agenda

1. Why We Chose Zephyr
2. Our Product Journey
3. Upstream Contribution Approach
4. The Flash "Crisis"
5. Challenges We Faced
6. Key Takeaways

# Why We Chose Zephyr

1

## Thriving Ecosystem

- Active community
- Frequent releases
- Responsive maintainers
- Backed by the Linux Foundation

2

## Built-in BLE Stack

- Production-ready Bluetooth Low Energy

3

## RTOS Advantages

- Multi-threadings
- Modularity
- Synchronization primitives simplified our application architecture

# Our Product Journey

Enabled by Zephyr

1

## BLE Communication

2020 | Wireless connectivity efficiently implemented

Zephyr v2.3.0, v2.7.0

2

## Modbus Protocol

2024 | Driver and samples working out of the box for slow baudrates \*

Zephyr v3.2.0

3

## SPI Integration

2024 | Straightforward with existing drivers

Zephyr v3.2.0

4

## Flash Code Relocation

2025 | Custom relocation replaced with official implementation \*

Zephyr v3.7.1 (LTS)

5

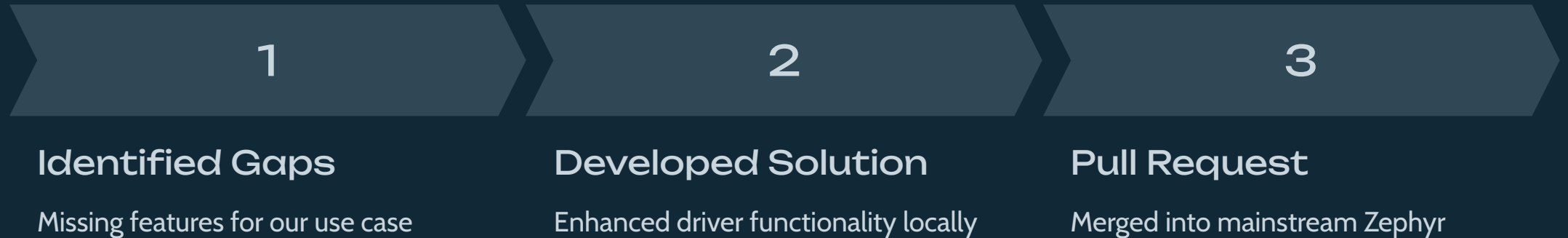
## The journey goes on...

New features, new possibilities...

Zephyr v4.x

# Upstream Contribution Approach

- Interrupt driven API chosen by default for Modbus Subsys was not robust enough for our use case: high baudrates
- Extended Zephyr's Modbus driver by adding Async serial implementation
- Contributed the changes upstream



This reinforced the open-source model's value: solving our problems, sharing solutions for all.

# The Flash "Crisis"

1

## Problem: Running Out of Space

Product code size exceeded available partition flash capacity.

2

## Interim Fix: Custom Relocation

- Initial custom relocation scheme as a temporary solution.
- Implementation difficult to maintain over time.

3

## Zephyr Update: Code Relocation Feature

- Zephyr's built-in Code Relocation Feature with growing functionalities added upstream meanwhile.
- Sustainable way to handle the challenge.

# Challenges We Faced

Not everything was smooth sailing — lessons learned along the way



## Steep Learning Curve

- Device tree syntax and Kconfig options required some time
- However, the active community makes up for it



## Zephyr Updates

- APIs/Kconfigs keep evolving between releases, requiring migration effort with every Zephyr Update
- Follow LTS: That's what they're made for ;)



## Long PR Approval Process Cycles

- PRs can take months, BUT understandable for a major project.
- Tip: keep your PR warm, stay engaged!



## Vendor-Specific Features vs. Portability

- Some advanced chip features require bypassing Zephyr drivers to get the most out of your hardware
- e.g. ADC driver with DMA, PPI support, etc.



# Key Takeaways

1

## Product Development Acceleration

Zephyr enabled faster feature delivery as one-stop shop.

2

## Effective Open-Source Collaboration

Contributing improved our code and the ecosystem.

3

## Ecosystem Maturity matters

Choosing a platform with a strong community pays out.

4

## Balance Innovation with Stability

Use LTS for production, but stay tuned on new features!

# Conclusion

Our choice of Zephyr has been validated across multiple products over the years - a decision we've never doubted.

# Questions?

Let's discuss your Zephyr experiences.