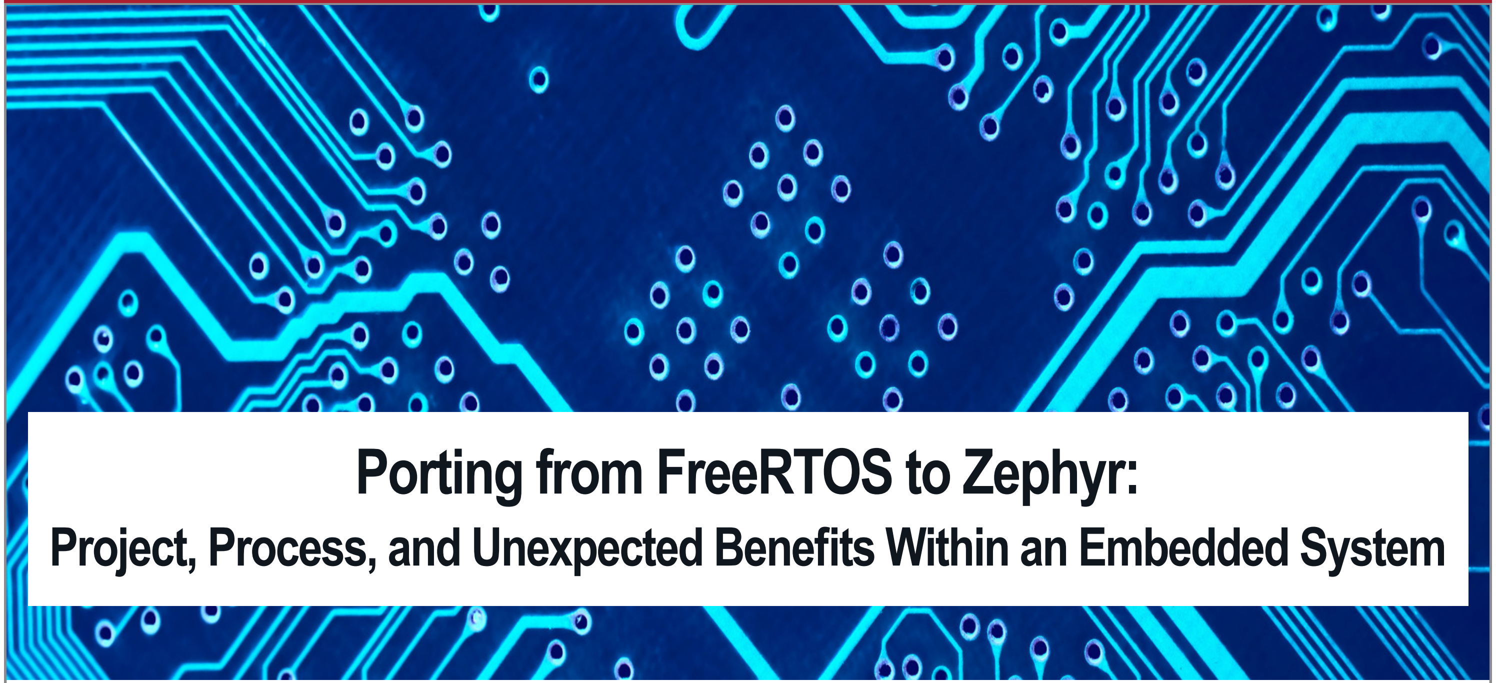
BOSTON ENGINEERING™

Imagine the Impact™



Porting from FreeRTOS to Zephyr:

Project, Process, and Unexpected Benefits Within an Embedded System

The Boston Engineering software team recently completed a project that involved porting the software for an existing system from FreeRTOS to Zephyr. Below is a review of why they performed this port, some thoughts about the two different Real Time Operating Systems (RTOSes), and some interesting insights that were gained along the way.

Challenge: Integrate embedded system with CANopen devices

Solution: Port FreeRTOS to Zephyr

Result:

- Requirements met
- Savings gained through Zephyr Functionality
- Secured connectivity features for future use

Presented by:
Chris Morgan, Director Software Engineering
Boston Engineering

Brief History of FreeRTOS and Zephyr

Released in 2003, FreeRTOS has been used in embedded devices for nearly twenty years. Amazon's support of FreeRTOS has been welcome news, in addition, it establishes that there is strong continued corporate support around the OS along side its wide user base.

By comparison Zephyr is a newcomer, having its first release in 2017. Though Zephyr lacks in overall maturity, it's fresh approach compensates with an ability to design without regard for decades of previous development.

With this blank slate Zephyr devs were free to select technologies and approaches without being tied to historical baggage, while also borrowing from the best from systems and technologies, like FreeRTOS, that came before them. Zephyr's development pace is amazing, with an average of some 200 commits per week. FreeRTOS by comparison is a few dozen.

Decision to Port to Zephyr

We have had success working with both FreeRTOS and Zephyr. In this situation the project had requirements to add integration with CANopen devices. While there is a CANopen integration with FreeRTOS, the CANopen integration in Zephyr was better developed and integrated, pointing to a reduction in development time and effort. We also looked at Zephyr's support for a number of connectivity features. Almost all electronic devices developed today have some integrated connectivity, such as Bluetooth Low Energy, WiFi, cellular etc. While the specific project being ported had a near term need for CANopen support, it has longer term future connectivity needs. CANOpen plus IoT integrations led us to selecting Zephyr as our target RTOS.

As a note, we did consider a number of other RTOSes in addition to Zephyr, such as Mbed, embOS, and Nuttx. Again, the integration of connectivity features put Zephyr ahead of the others.

Comparison Against Project Requirements

This project required the following 9 characteristics:

- Integrated CANopen Stack,
- Wide range of connectivity features for future use,
- Ability to run on low cost microprossesors,
- Real-Time operation,
- Minimal RAM/Flash Footprint,
- Maturity (proven usage in the marketplace),
- Active development and support,
- Robust tooling/IDE integration of Debugger, and
- Documentation

After reviewing several options, the decision came down to two familiar options for Boston Engineering: FreeRTOS and Zephyer. A Side by Side comparison showed that while the systems are very similar, Zephyr was the superior choice in this instance due to having an integrated CANopen stack and a wide range of connectivity features. Here is a summary of those details, also shown in exhibit 1 below.

Both operating systems are able to run on low cost microprocessors. Each has documentation available online for easy download. FreeRTOS uses STM32CubeIDE for it's debiggunbg, while Zephyr offer its own Zephyr SDK / VSCode + cortex-debug.

Either systems offer minimal RAM/Flash footprint, with FreeRTOS needing 152k Flash and 169 RAM, Zephyr 178K Flash and 121K Ram to run both the OS and App. WE considered these numbers to be the same given our loose constraints.

As a note, we have not looked to optimize consumption of flash or ram for either FreeRTOS or Zephyr. The microprocessor we are using has a lot of ram and flash available and we've been focusing on features vs. tuning. We expect that a considerable amount of savings could be found in both cases.

We consider both FreeRTOS and Zephyr to be mature, with 20 and 6 years of use, respectively. Each system is actively developed and supported, with over 20 top firms active in their respective development groups. In fact, Many companies support both FreeRTOS and Zephyr, recognizing that there are a number of use cases where each may be better suited. With this corporate support often comes code updates and abstraction layers delivered to the code base for developers to utilize.

Driving our final decision to employ Zepher is the fact that, while a robust solution, FreeRTOS does not offer an integrated CANopen Stack – rather their solutions require porting by the developer – an added time and expense. Zephyr's options are integrated into the

Exhibit 1: Comparing FreeRTOS & Zephyr Offering vs. Project Requirements

Project Requirements	FreeRTOS	Zephyr
Runs on low cost microprocessors	Yes	Yes
Documentation	Yes	Yes
Real-time	Yes	Yes
Robust Tooling / IDE integration of	Yes	Yes
Minimal ram/flash footprint	Yes OS + app: 152k flash, 169k ram	Yes OS + app: 178k flash, 121k ram
Actively developed and supported	Yes: 23 development members	Yes: 9 Platinum Developer members
Mature	Yes – 20 years	Yes – 6 years
Integrated CANopen stack	No	Yes
Wide range of connectivity features for future use	Partially Some - via 3rd party libraries	Yes Extensive - integrated into configuration and build system

configuration and build system.

Finally, Zephyr provides an extensive list of connectivity features, far outpacing those offered by FreeRTOS, and making it the clear choice for maintaining option for future builds.

Actual Porting Activities

For this project, the system is a typical embedded system. We make use of the following:

- Threads and thread synchronization
- Ringbuffer
- Console shell
- External flash
- Other devices: UARTs, RTC, i2c, spi

The porting went relatively smoothly, we were able to learn more about Zephyr, and the results did in fact allow us to save the time and effort we had anticipated. Below is an analysis and notes related to the actual process of porting each of these elements.

Threads and Thread Synchronization

The thread and thread synchronization changes were the most straight forward part of the porting activity. FreeRTOS and Zephyr have very similar functions for creating threads, creating, and waiting on events / semaphores etc.

Ringbuffer

During the Zephyr port it was (re-)discovered that the application had implemented a ringbuffer, rather than making use of the FreeRTOS streambuffer. Code review of the custom ringbuffer revealed a number of conditions that could result in memory overruns.

In addition, there were no unit tests for this ringbuffer implementation.

With these gaps it was decided that we should replace this ringbuffer implementation during the port. We replaced this custom ringbuffer with a Zephyr pipes. The Zephyr pipe provides thread synchronization and is widely used and unit tested. Dropping this custom code and leveraging a Zephyr primitive was a bonus.

Console Shell

The project makes use of a command line interface (cli), provided via a UART, view and configure system settings, and to aid in debugging. With the move from FreeRTOS to Zephyr we transitioned from FreeRTOS+CLI to the Zephyr Shell

External Flash

The FreeRTOS implementation had a few hundred lines of application layer code to interface with an external SPI flash, a W25X20 or similar. With global supply chain issues, this code needed to be altered to add support for an available Flash chip with a slightly different memory layout and command set (AT25SF).

Zephyr has SPI flash drivers which supports both the W25X20 and the AT25SF parts being used in the project. A nice feature, indeed. By making use of the Zephyr driver we take advantage of software that has a lot more usage, testing, and development.

Other Devices

Zephyr has drivers for the processor family that was used, in this case STM32, for uarts, i2c, spi, and rtc. While there was a bit of a learning curve on how to configure these devices in the devicetree, the port of these devices from the STM32 HAL to Zephyr drivers went smoothly.

Unexpected Benefits to Zephyr

During the port we uncovered into multiple aspects of the Zephyr system things that excited us. While these aren't features that would have swayed a decision to port to Zephyr, they are extras that we thought were nice touches or great time savers.

- **Developer support:** We've had quick responses from the Zephyr developers when opening support requests. For example we uncovered a bug with usage of pipes from ISR context that no one else had run into. We reported it (by opening a PR to clarify the documentation) on the Zephyr GitHub issue tracker. After some discussion the root cause was understood and in two months the pipes functionality was rewritten and the issue resolved. We were more than satisfied with the quick identification phase and the time to resolve, especially given the complexity of the pipes and underlying systems.
- **Integrated tooling:** Zephyr ships with an sdk and the west command line tool that simplifies getting up and running
- **Board support:** Zephyr features broad support for most development boards that we commonly use
- **Testing framework:** Zephyr features a built-in framework that allows for units tests to be created and run against functions within your code. While many similar frameworks exist, it can be cumbersome to test code that includes (or is adjacent to) OS primitives such as tasks and semaphores. The built-in Zephyr framework - ZTests - allows these os primitives to be compiled and tested alongside any other line of code.
- **Devicetree:** Like Linux, Zephyr makes use of the devicetree. This lets us leverage our knowledge of Linux devicetrees and the devicetree itself lets us consolidate device configuration (UART pins, spi

data rates) with custom board settings, removing the need for build time #if / #ifdef conditionals.

With FreeRTOS we were using STM32CubeIDE. Built on Eclipse, the STM32CubeIDE also includes the CubeMX GUI tool that generates configuration source code for the processor and its devices. With Zephyr the function of configuring hardware shifts to the device drivers, and the device drivers get their configuration from the devicetree.

The shift from CubeMX to devicetree went relatively smoothly. The devicetree does require that we know more about the hardware configuration and CubeMX provides some helpful checks for invalid configurations. Even still, it wasn't tough to transition to devicetree files. Devicetree files are plain text, which lets us version them in git and review changes. And, even though we aren't using STM32CubeIDE on the project we still make use of the CubeMX tool to aid in clock tree configuration!

- **Linux-feel:** Many of us have been using Linux and POSIX OSes for what seems like forever. Along with support for Linux features like Kconfig, devicetree and SocketCAN, Zephyr also provides a Posix compatibility layer. Sharing these approaches, tools and API with Linux systems lets us leverage our existing knowledge.

- **Kconfig:** FreeRTOS is configured via FreeRTOSConfig.h. This is a perfectly reasonable and simple way to configure OS options. Zephyr has textual config files that you can edit. However, it uses Kconfig to provide a menu driven approach for configuring, reading option help, and searching for OS configuration settings.

Having a menu driven configuration system is more critical for Zephyr than for FreeRTOS, given the Zephyr's range of integrated subsystems and drivers that can be enabled/disabled or configured.

Ultimately, we really like Zephyr Kconfig support!

Trends We See

Connectivity features (IoT) have dominated product design for a number of years. There is a high value brought to usability and maintenance through connectivity. We agree with projections that nearly all devices will become connected in some manner or another.

As the trend towards more powerful and capable processors continues we also expect that many new and updated products will make use of more featureful operating systems such as Linux.

Even still, many products are sensitive to cost and will feature the lowest cost SoCs that meet their feature requirements. These Systems-on-Chips (SoCs) will continue running RTOSes even as RTOSes continue to develop features in the direction of the major OSes like Linux.

While we continue to look forward to expanding our use of Zephyr we hope that FreeRTOS and its developers will embrace its evolution towards improved integration and connectivity features.

About the Author

Chris Morgan
Software Engineering Manager
Boston Engineering

Chris has over 20 years of experience in software and hardware design. He enjoys all things engineering and is an author and contributor of a number of open-source projects.

About Boston Engineering Embedded Systems

Boston Engineering is a leader in the development of custom embedded hardware and software systems. Our integrated, cross functional team of electrical, software, and mechanical engineers will advise, direct, and manage any computing development project. Whether the challenge is to modernize components, increase reliability, improve performance, or synchronize I/O options, Boston Engineering thrives on solving the toughest challenges.

Embedded Systems is one of Boston Engineering's Centers of Excellence. Contact us to learn more about how our Centers of Excellence define and support the commitment we make to our clients and the organizations we serve.

About Boston Engineering Software Development

Boston Engineering's Software Development process leverages the experience and knowledge gained through the development of dozens of software, electromechanical, and robotic systems. Our Subject matter experts have advanced knowledge of:

- Designing embedded systems that combine new features with unmatched performance and reliability
- Delivering precision motion control for robotics, factory automation, and other applications
- Conducting software testing, as well as verification and validation (V&V)
- Reducing risk and accelerating your project development with simulation and software prototyping.

Our skilled software developers are experts across software development tools, including:

- UI development (Web, mobile, embedded)
- Controls software
- Embedded software
- Robotic systems (ROS2)
- Operating systems:
  - Linux application development
  - RTOS (Zephyr, FreeRTOS)
- Internet of Things (IoT)
- Communications systems and protocols (WiFi, Cellular, BLE, Iridium)

About Boston Engineering

Making a meaningful impact drove us to start the business in 1995 and it has driven every project since. From designing advanced products and technologies to accelerating time to market, Boston Engineering thrives on solving tough client challenges. We provide product design and engineering consulting from concept development through commercialization. Clients benefit from our deep product development capabilities, focused industry expertise, and ISO-certified quality management system. Founded in 1995, Boston Engineering is headquartered in Waltham, Mass.

bsi. ISO 9001 Quality Management FM 590151

bsi. ISO 13485 Medical Devices Quality Management FM 590150

BOSTON ENGINEERING™
Imagine the Impact™

Contact us:
boston-engineering.com
info@boston-engineering.com
781-466-8010