COMARCH

Bluetooth®

Zephyr®

Case Study

Bluetooth stack integration with Zephyr OS

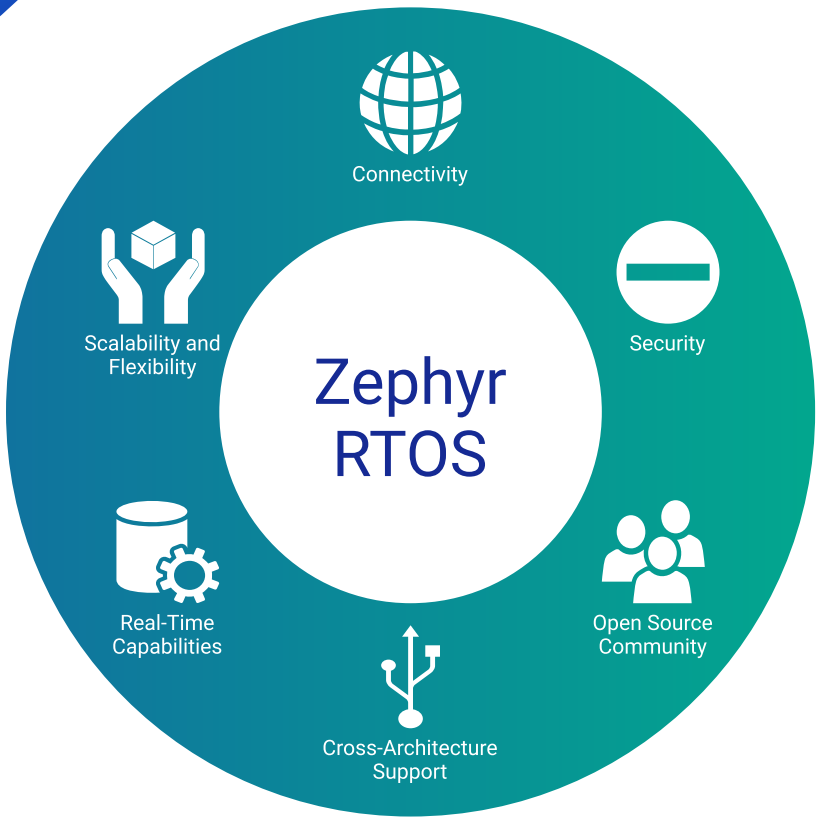
Project overview

Engineers from Comarch Software and Hardware Services unit are masters of embedded systems, especially when it comes to wireless connectivity. This is thanks to our long term **strategy of technical trainings and mentoring, knowledge sharing and research and development work**. We follow the newest trends on the market and we play an active part in bringing the technology to the people. As a result, we are not only able to build our own successful products, but also **offer services and consultancy** to the customers all around the world – no matter how challenging and innovative the task is.

Our recent R&D work focused on **integrating Bluetooth Low Energy (BLE)** stack to Zephyr OS. We believe that results of such work (also for different wireless protocols like **Ultra-Wideband**) would bring any embedded system to **a higher level of performance and quality**.

Why Zephyr?

The Zephyr Project is an open-source initiative hosted by the Linux Foundation, focused on developing a lightweight, modular, and secure real-time operating system (RTOS) for resource-constrained devices. It is designed to meet the growing needs of the Internet of Things (IoT), embedded systems, and edge computing applications. Zephyr is suitable for a wide range of devices, from simple microcontrollers to more complex systems, making it a versatile and scalable solution.



Connectivity: It includes support for a wide array of communication protocols, such as Bluetooth, Wi-Fi, Ethernet, Thread, Zigbee, and LoRa, enabling seamless integration into connected ecosystems.

Security: Zephyr places a strong emphasis on security, incorporating features like secure boot, trusted execution environments, and regular security audits. This focus helps address the critical need for safe and reliable IoT solutions in industries such as healthcare or automotive.

Open Source Community: Being an open-source project, Zephyr benefits from contributions by a global community of developers, technology companies, and academic institutions. This collaboration fosters innovation, improves quality, and provides extensive documentation and support.

Cross-Architecture Support: The project supports multiple hardware architectures, including ARM, x86, RISC-V, ARC, and others, ensuring portability across diverse platforms. This broad compatibility enables widespread adoption in embedded and IoT environments.

Real-Time Capabilities: With its RTOS features, Zephyr ensures predictable and low-latency responses, critical for time-sensitive applications such as robotics, industrial automation, and sensor networks.

Scalability and Flexibility: Zephyr is highly scalable, allowing developers to customize the OS by selecting only the necessary components. This modularity makes it ideal for devices with varying hardware capabilities, from low-power sensors to advanced industrial controllers.

BLE Host integration work

The objective of this project was to develop a versatile BLE Host environment capable of facilitating seamless transitions between various versions and implementations of BLE Host. This initiative aimed to enhance flexibility and adaptability in BLE development. Our team focused on integrating an external Open Source BLE Host (shared publicly by Packetcraft) into the Zephyr project, enabling its usage within BLE sample applications.

Scope of work and key activities

Porting the External Host: The external BLE Host was integrated into the Zephyr project as a component of the Zephyr Bluetooth subsystem. This required modifications to build system files, including CMakeLists and configuration files, to ensure smooth transition.

Host Selection Flexibility: We introduced a mechanism that allows users to choose which BLE Host to use when building applications. Developer can now select during compilation process either the default BLE Host provided by Zephyr or the newly integrated external stack, offering greater customization.

Translation Layer Development: A dedicated translation layer was implemented to facilitate sample application compatibility with multiple BLE Hosts. This layer incorporates conditional compilation to dynamically adjust based on the selected Host.

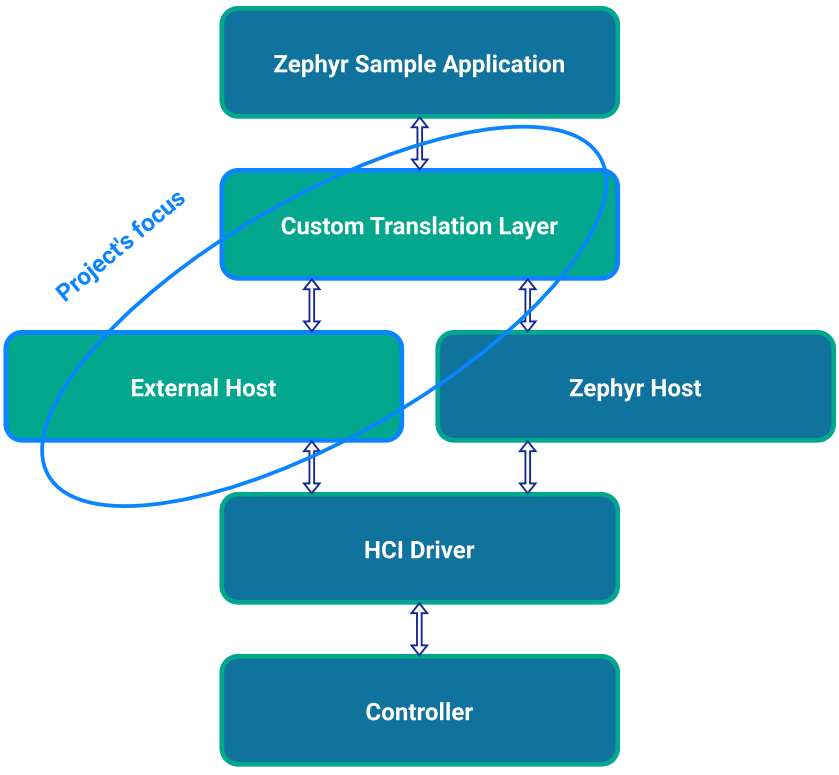
HCI Driver Integration: The imported BLE Host was successfully integrated with Zephyr's Host Controller Interface (HCI) drivers, ensuring seamless communication and functionality.

CI/CD Pipeline Implementation: A robust Continuous Integration/Continuous Deployment (CI/CD) pipeline was created to test builds across different configurations, ensuring reliability and consistency. One of the pipelines run PTS qualification tests. The other measured current consumption during streaming and usual stack states.

Demo Application Development: To demonstrate the capabilities of the dual-host environment, a couple of sample applications were prepared, showcasing the functionality such as connection, advertising and broadcasting of both the original Zephyr BLE Host and the integrated external one. We run the BLE audio example with the new Host and nRF5340 Audio DK and checked unidirectional and bidirectional streaming with Google Pixel 7.

Translation layer

The idea of creating a translation layer stems from the differences between various Host implementations. While their purpose is the same, they achieve it in different ways. This has created a strong need for an additional layer that acts as a bridge between the Application Layer and the chosen Host Layer. To address this, we decided to design a translation layer that ensures consistent results regardless of the Host being used. Simple Zephyr Bluetooth API calls are translated into specific Host API commands, creating the necessary environment for Host Layer tools such as ATT, SMP, and others.



The integration process was challenging due to significant differences in how Hosts were designed and structured. Each stack featured unique functionalities and architectural approaches, making the planning and development of a translation layer both challenging and time-consuming. This translation layer was critical to enable communication and compatibility between the sample applications and the varying Hosts. Developing it involved a deep understanding of both Hosts architectures and the Zephyr ecosystem, as well as significant effort to address edge cases and ensure robust operation.

Further development process

The imported Host relies on its own set of resources to perform various operations essential for embedded systems, such as memory allocation, queue management, and other related functionalities. These built-in mechanisms, while functional, could be replaced or adapted to utilize the existing tools and utilities provided by Zephyr. This approach would not only enhance integration but also align the external Host with Zephyr's ecosystem, potentially improving consistency and maintainability. This was identified as the next step and addressed by systematically replacing Packetcraft resources, starting with the Logging System and concluding with Memory Pool allocation.

Challenges during development

Certain components of the integrated Packetcraft's Host initially relied on external drivers for their operation. To ensure seamless integration within the Zephyr environment, these external drivers were replaced with equivalent Zephyr drivers. This replacement process required careful adaptation to ensure compatibility and maintain the functionality of this Host within the Zephyr framework. Another key consideration during the integration was memory usage. Memory is a critical resource in embedded systems, and the imported Host had a different construction and resource management approach compared to the Zephyr Host. Packetcraft's implementation is based on Memory Pools being allocated on Heap during initialization stage, while Zephyr Host implementation is more focused on Heap available for each thread and dynamic allocation. These differences necessitated modifications to the file integration process to optimize memory utilization and align with the constraints and expectations of the Zephyr environment, such as adjusting Memory Pools usage to available thread heap size. These adjustments were vital to achieving efficient and reliable performance of the integrated Host.

Outcome

Our team of engineers was able to successfully integrate the external Host with Zephyr OS just as planned. We are also planning future features and improvements to put them into our R&D backlog. The knowledge gained during development will not only improve our products, but also enhance our offerings and expertise in delivering services. Do you want to use RTOS as a part of your implementation and environment? Zephyr is a great choice, and with our expertise in connectivity protocols, we can help you create your new product, SDK, Proof of concept, or anything else you need. Contact us for more info and other references now!

For more information:

technologies.comarch.com
technologies@comarch.com