Zephyr Project:
An RTOS to change the face of IoT

Maureen Helm
Microcontroller Software Architect
NXP
## What is Zephyr?

### Small Footprint
**RTOS for IoT**
- As small as 8KB
- Enables applications code to scale

### Truly Open Source
- Apache 2.0 License
- Hosted by Linux Foundation
- Transparent development

### Cross Architecture
- ARM
- x86
- ARC
- Others
Zephyr’s core values

- Modularity
- Security
- Cross Architecture
- Connectivity
- Community Developed
Small OS & RTOS market analysis

- Saturated RTOS/Fragmentation
- Roll Your Own/No OS
- Adoption growth in IoT development
- Compromised Devices

Opportunity to build a leading IoT OS
Why Zephyr?

- Strategic Investment
- Best-of-Breed RTOS for IoT
- True Open Source Development and Governance
- Permissively Licensed
- Modular
- Established Code Base
Zephyr project governance

- Governance
- Security
- TSC
- Marketing
- Contributors
Current platinum members

NXP  Linaro

intel  synopsys
## Zephyr™ Project Architecture

- Provide an OS that runs best on MCUs for wearable and IoT devices, where the cost of the silicon is minimal
- Highly Configurable, Highly Modular
- Kernel mode only
  - Nanokernel: Limited functionality targeting small footprint (below 10k)
  - Microkernel (superset of nanokernel): with additional functionality and features
- No user-space and no dynamic runtimes
- Memory and Resources are typically statically allocated
- Cross architecture (IA32, ARM®, ARC, NIOS-II, others under discussion)

### Platform

<table>
<thead>
<tr>
<th>Device Drivers</th>
<th>Power Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4, IPv6</td>
<td>6LowPAN</td>
</tr>
<tr>
<td>CoAP, MQTT, HTTP, LWM2M</td>
<td>DTLS, TLS, ..</td>
</tr>
<tr>
<td>C APIs</td>
<td>Applications</td>
</tr>
<tr>
<td>3rd Party Libraries</td>
<td>3rd Party Libraries</td>
</tr>
<tr>
<td>Applications</td>
<td>3rd Party Libraries</td>
</tr>
<tr>
<td>C APIs</td>
<td>3rd Party Libraries</td>
</tr>
<tr>
<td>CoAP, MQTT, HTTP, LWM2M</td>
<td>DTLS, TLS, ..</td>
</tr>
<tr>
<td>Device Management</td>
<td>Power Management</td>
</tr>
<tr>
<td>Microkernel</td>
<td>Nanokernel</td>
</tr>
<tr>
<td>Nanokernel</td>
<td>Platform</td>
</tr>
<tr>
<td>UART</td>
<td>SPI</td>
</tr>
<tr>
<td>GPIO</td>
<td>IPC</td>
</tr>
<tr>
<td>Radios</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of Zephyr™ Project Architecture](image_url)
Status Update

- **Zephyr 1.5 Released August 2016**
  - mbedTLS Integration
  - IOT Protocols (MQTT, NATS, ..)
  - TCP support, Flash filesystem support, improved BR/EDR support (for L2CAP, in particular),
  - Support for the Altera Nios II/f soft CPU architecture

- **Zephyr 1.6 planned for November 2016**
  - Unified Kernel
  - Native IP Stack
  - Cortex-M0+
  - Bluetooth Link Layer
  - Expand Board Support
The Kernel Today
Zephyr Kernel – Key Features

Multi-threading services, including both priority-based, non-preemptive fibers and priority-based, preemptive tasks (with optional round robin time-slicing).

Interrupt services, including both compile-time and run-time registration of interrupt handlers, which can be written in C or assembly language.

Inter-thread synchronization services, including binary semaphores, counting semaphores, and mutex semaphores.

Inter-thread data passing services, including basic message queues, enhanced message queues, and byte streams.

Memory allocation services, including dynamic allocation and freeing of fixed-size or variable-size memory blocks.
Zephyr Kernel – Key Features

- Power management services, including tick-less idle and an advanced idling infrastructure.
- Highly configurable, allowing an application to incorporate only the capabilities it needs, and to specify their quantity and size.
- Zephyr requires all system resources to be defined at compile-time to reduce code size and increase performance.
- Provides minimal run-time error checking to reduce code size and increase performance. An optional error checking infrastructure is provided that can assist in debugging during application development.
- Library based RTOS ("kernel-less")
Library-Based RTOS ("kernel-less")

- One single executable which is executed in one single address space
- No dynamic loading at run-time
  - Minimizing the operating system code
  - System calls are implemented as function calls
  - No context switches are required when calling an operating system call
- Lack of security through hardware memory separation
  - Application and operating system calls are implemented as thread in the same address space
  - Bugs in one part of the system can affect the whole system

Often more efficient and less time consuming as a full context switch with address space changes.

On small microcontrollers on which only one application is executed this disadvantage is acceptable.
Zephyr Nanokernel Overview

- A high-performance, multi-threaded execution environment with a basic set of kernel features
- Ideal for systems with sparse memory (the kernel itself requires as little as 2 KB!) or only simple multi-threading requirements (such as a set of interrupt handlers and a single background task)
- Examples of such systems include:
  - embedded sensor hubs
  - environmental sensors
  - simple LED wearables
  - store inventory tags
Nanokernel Scheduling and Objects
Nanokernel Scheduling and Objects

- Cooperatively scheduled
- Run until they yield or call a blocking API
  - Marked as not runnable
  - Next highest priority fiber is then run
- Typically used for device drivers and performance-critical work
Zephyr Microkernel Overview

- Supplements the capabilities of the nanokernel to provide a richer set of kernel features
- Suitable for systems with heftier memory (50 to 900 KB)
- multiple communication devices (like Wi-Fi and Bluetooth® Low Energy) and multiple data processing tasks
- Examples of such systems include:
  - Fitness wearables
  - Smart watches
  - IoT wireless gateways
Microkernel Scheduling and Objects
Microkernel Scheduling and Objects

- A task is scheduled when no fibers are runnable
- Tasks are preemptible
- Highest priority task runs first
- Round-robin time-slicing between tasks of equal priority
- Used for data processing
Zephyr’s Unified Kernel
Zephyr Kernel Today, the bad…

- Two kernels, Nano and Micro Kernels
- Confusing for some developers
- Middleware needs to be written for the two modes
- Microkernel also has performance issues
- Duplication of object types between the nanokernel and microkernel
- Fibers cannot perform most microkernel operations
- Operation by tasks on nanokernel objects are inefficient
- Microkernel objects and tasks cannot be created at runtime
Unified Kernel

- Unified kernel drops the microkernel server
- The message-passing microkernel model is removed. The only remaining kernel/scheduler is the nanokernel
- The microkernel server stack cannot overflow, since it does not exist anymore.
- Cooperative threads are now allowed to run for any amount of time in any system configuration.
Unified Kernel

- Unified Kernel makes the nanokernel 'preemptible thread'-aware
- Unifies fibers and tasks as one type of threads
- Allows cooperative threads to operate on all types of objects
- Introduces a new, more streamlined API, without any loss of functionality
- Old API is kept and is marked as deprecated to ease transition
- Remove sysgen and MDEF (Microkernel DEfinition File) and support dynamic object creation
Unified Kernel - Summary

- Elimination of separate microkernel and nanokernel build types,
- Elimination of the MDEF in microkernel-based applications,
- Simplifying and streamlining the kernel API,
- Easing restrictions on the use of kernel objects,
- Reducing memory footprint by merging duplicated services, and
- Improving performance by reducing context switching
Device Drivers and HALs
Supporting Driver Frameworks & HALs

Applications

3rd Party Libraries

C APIs

CoAP, MQTT*, HTTP*, LWM2M*

DTLS, TLS, ..

IPv4*, IPv6
6LowPAN

Device Management

Device Drivers

Microkernel

Nanokernel

Hardware Platform

UART
SPI
GPIO
I²C

Radios

Zephyr Native Drivers

NXP Kinetis SDK (KSDK)

ARM® Cortex® Microcontroller Software Interface Standard (CMSIS)

Intel® Quark™ Microcontroller Software Interface (QMSI)

Nanokernel

* planned
Hardware Support Hierarchy

- Board 1
  - SoC 1
    - SoC Series 1
      - SoC Family 1
        - CPU 1
    - SoC Family 2
  - SoC Series 2
    - SoC Family 2
  - SoC Series 3
    - SoC Family 3
  - SoC Series 4
    - SoC Family 3
  - SoC Family 3
  - SoC Family 4
- Board 2
  - SoC 2
    - SoC Series 2
  - SoC Series 3
    - SoC Family 3
    - SoC Family 4
  - SoC Series 4
    - SoC Family 4
- Board 3
  - SoC 3
    - SoC Series 3
    - SoC Family 3
  - SoC Series 4
    - SoC Family 4
- Board 4
  - SoC 4
    - SoC Series 4
    - SoC Family 4
  - SoC Series 5
    - SoC Family 5
  - SoC Series 6
    - SoC Family 6
  - SoC Series 7
    - SoC Family 7
## Hardware Support Hierarchy

<table>
<thead>
<tr>
<th>FRDM K64F</th>
<th>nRF52 NITROGEN</th>
<th>nRF51XX</th>
<th>Quark SE C1000 Devboard</th>
<th>Arduino 101</th>
</tr>
</thead>
<tbody>
<tr>
<td>MK64F12</td>
<td>nRF52832</td>
<td>nRF51XX</td>
<td>Quark SE C1000</td>
<td>Curie</td>
</tr>
<tr>
<td>Kinetis K6x Series</td>
<td>Nordic NRF52</td>
<td>Nordic NRF51</td>
<td>Quark SE</td>
<td>Quark SE</td>
</tr>
<tr>
<td>NXP Kinetis</td>
<td>Nordic NRF5</td>
<td>Nordic NRF5</td>
<td>Quark</td>
<td>Quark</td>
</tr>
<tr>
<td>Cortex-M4</td>
<td>Cortex-M4</td>
<td>Cortex-M0+</td>
<td>Lakemont</td>
<td>Lakemont</td>
</tr>
<tr>
<td>ARM</td>
<td>ARM</td>
<td>ARM</td>
<td>x86</td>
<td>x86</td>
</tr>
</tbody>
</table>
Networking and Bluetooth
IOT Technologies: Connectivity

IEEE 802.15.4
Bluetooth® 4.0 Low Energy
Wi-Fi®
NFC
3GPP
IPv6
RPL
6LoWPAN
HTTP
CoAP
MQTT
LWM2M
TCP/UDP
TCP
ZigBee®
Thread
Bluetooth: Status

- Bluetooth 4.2 compliant host stack supporting LE
  - GAP & GATT
  - IPSP (6LoWPAN, using L2CAP Connection oriented Channels)
  - LE Secure Connections
- Controller-Host separation using standard HCI
- Existing HCI drivers for 3- and 5-wire UART transports
- Controller (Link Layer) implementation supporting nRF5x radios
- Controller & Host are modular and can be built in three different Configurations:
  - Host-only (talking over HCI to controller on a separate CPU)
  - Controller-only (talking over HCI to host on a separate CPU)
  - Host & Controller - both running on the same CPU
Bluetooth: Short Term Plans

- Bluetooth BR/EDR (Classic) support:
  - A2DP (Advanced Audio Distribution Profile, i.e. music streaming)
  - AVRCP (Audio/Video Remote Control Profile, i.e. playback control)
  - HFP (Hands-Free Profile), Hands-free role (headset or car kit)

- Creating a proper radio abstraction for the controller support to extend beyond nRF5x radios

- Making Zephyr run on all cores of Intel Curie-based boards (like the Arduino 101), including the nRF51 BLE radio on them.
Last but not least....
Security

- Standardized building block and robust communication stacks
- Cryptographic library based on TinyCrypt 2 and mbedTLS.
- Static and single binary applications, Single address space, No loadable modules
- Planned security features:
  - Device Management and Updates
  - APIs to support vendor specific Crypto implementations (SW/HW)
  - Secure Key Storage

Cloud Services
- CoAP, MQTT, HTTP

Communication Security
- IPv4
- IPv6/6LoWPAN
- DTLS
- Crypto Libraries and Services
- Device Management: LWM2M
- Stack Protection
- HW Crypto Engine
- Secure Boot, TEE**
Summary

- Open-Source, Permissively licensed
- Modular, Configurable and Scalable Design
- Secure and Connected
- Open for your contributions
Participate!

Benefits of early participation:

- Impact architecture
- Direction
- Marketing / Advocacy
- Decision making
NXP, the NXP logo and Kinetis are trademarks of NXP B.V. All other product or service names are the property of their respective owners. ARM and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. © 2016 NXP B.V.