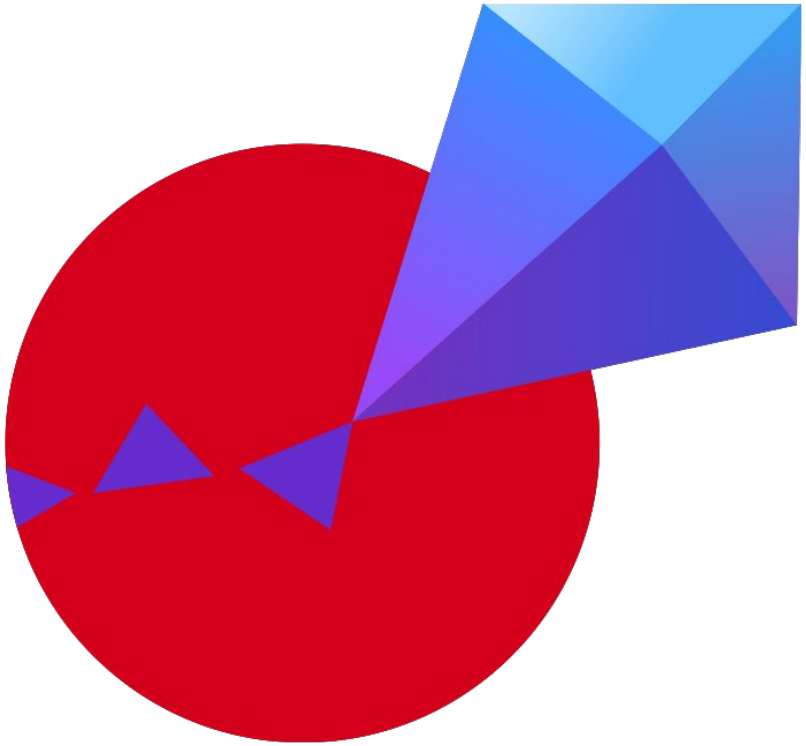
Zephyr

Real time operating system for IoT

Everything you always wanted to know
but were afraid to ask…



rndity;

Zephyr™ Project

Agenda

- What is Zephyr OS and why do we need another one?
- Main features supported by Zephyr kernel.
- Zephyr Kernel Primer
  - Threads
  - Timing
  - Memory Allocation
  - Synchronization
  - Data Passing
- Supported development boards
- Device Drivers/Model/Trees
- Nucleo-64 for Zephyr.

rndity;

What is Zephyr RTOS

and why do we need another one

rndity;

What is Zephyr OS

and why do we need another one

The base for Zephyr kernel is VxWorks Microkernel Profile created by Wind River (evolved over 20 years).

Support multiple architectures:

- ARM Cortex-M, Intel x86, ARC, NIOS II, Tensilica Xtensa and RISC-V

rndity;

What is Zephyr RTOS

and why do we need another one

Perfect for:

- simple connected sensors
- environmental sensors
- LED wearables
- fitness wearables
- smart watches
- modems and small IoT wireless gateways

Available through the Apache 2.0 open source license.

rndity;

What is Zephyr RTOS

and why do we need another one

Modular:

- resource constrained devices: available Flash memory from 8kB all the way up to 512kB
- Kconfig: system of activation and deactivation of selected functionalities of RTOS
- Zephyr SDK: tools for debugging and downloading, 5 different cross-compilers, host tools such as a custom QEMU, baremetal c library (based on newlib) but developers can use the tool suite of their choice

Connectivity:

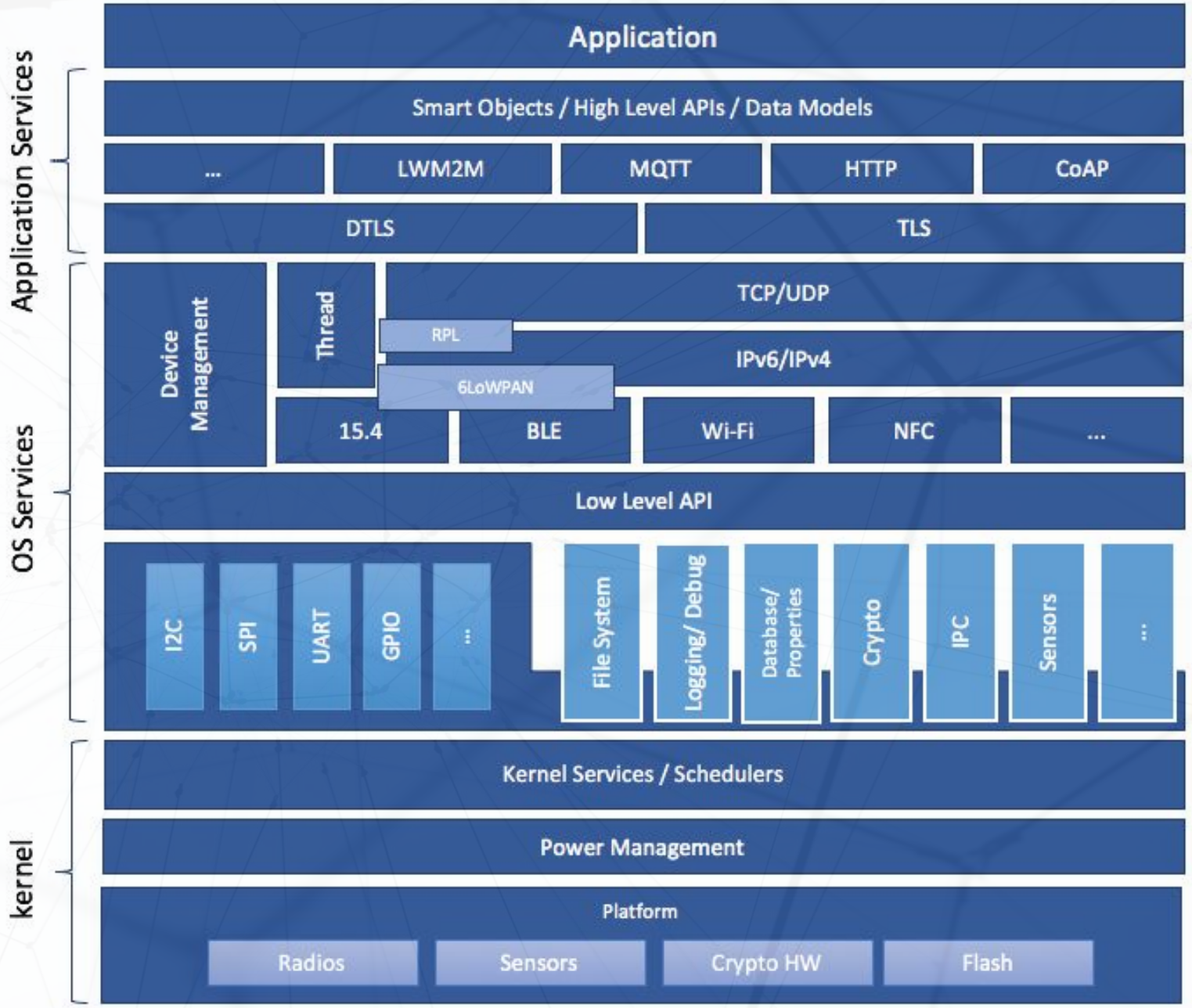
- Bluetooth®, Bluetooth® Low Energy, Wi-Fi*, 802.15.4 as well as other standards like 6LoWPAN, CoAP, IPv4, IPv6, and NFC

rndity;

Main features

supported by Zephyr Kernel

System architecture



rndity;

Main features

supported by Zephyr Kernel

What does make Zephyr different from other OSes?

- **Single address-space**
- **Highly configurable**
- **Compile-time resource definition**
- **Minimal error checking**
- **Extensive suite of services:**
  - *Multi-threading Services*
  - *Interrupt Services*
  - *Memory Allocation Services*
  - *Inter-thread Synchronization Services*
  - *Inter-thread Data Passing Services*
  - *Power Management Services*

Zephyr Kernel Primer

Threads

```
k_tid_t k_thread_create(struct k_thread *new_thread,
                        k_thread_stack_t *stack,
                                size_t stack_size,
                        k_thread_entry_t entry,
                        void *p1, void *p2, void *p3,
                        int prio,
                        u32_t options,
                        s32_t delay)
```

Lifecycle:

- creation
- termination
- aborting
- suspension

Key properties:

- stack area
- thread control block
- entry point function (up to 3 argument values)
- scheduling priority
- thread options: *K_ESSENTIAL* | *K_FP_REGS* | *K_SSE_REGS*
- start delay

Zephyr Kernel Primer

Threads

Scheduling:

- thread states: ready / unready
- thread priorities:
  - *cooperative thread*
  - *preemptible thread*
- scheduling algorithm
  - cooperative time slicing
  - preemptive time slicing
- scheduler locking
- thread sleeping vs. busy waiting

Zephyr Kernel Primer

Threads

Custom Data:

- a thread's *custom data* is a 32-bit, thread-specific value that may be used by an application for any purpose
- CONFIG_THREAD_CUSTOM_DATA

System Threads:

- a *system thread* is a thread that the kernel spawns automatically during system initialization
- Main thread:
  - kernel initialization
  - invokes the application's main() function
- Idle thread
  - performs a "do nothing" loop
  - activates the board's power management support to save power

rndity;

Zephyr Kernel Primer

Threads

Workqueue Threads:

- kernel object that uses a dedicated thread to process work items in a first in, first out manner
- to defer complex interrupt-related processing from an ISR to a cooperative thread
- to offload non-urgent processing to a lower-priority thread
- *system workqueue* and user's defined workqueues

Work Item key properties:

- handler function
- pending flag
- queue link

rndity;

Zephyr Kernel Primer

Timing

Kernel Clocks

- the foundation for all of its time-based services
- 32-bit hardware clock
- 64-bit system clock

Timers

- *timer* is a kernel object that measures the passage of time using the kernel's system clock
- key properties:
  - *duration / period / expiry function / stop function / status*

rndity;

Zephyr Kernel Primer

Memory Allocation

Memory Slabs

- kernel object that allows memory blocks to be dynamically allocated from a designated memory region; all memory blocks in a memory slab have a single fixed size

Memory Pools

- kernel object that allows memory blocks to be dynamically allocated from a designated memory region; memory blocks in a memory pool can be of any size

Heap Memory Pool

- predefined memory pool object that allows threads to dynamically allocate memory from a common memory region in a malloc() -like manner
- CONFIG_HEAP_MEM_POOL_SIZE

rndity;

Zephyr Kernel Primer

Synchronization

Semaphores

- kernel object that implements a traditional counting semaphore
- use a semaphore to control access to a set of resources by multiple threads
- use a semaphore to synchronize processing between a producing and consuming threads or ISRs

Mutexes

- kernel object that implements a traditional reentrant mutex
- allows multiple threads to safely share an associated hardware or software resource by ensuring mutually exclusive access

Zephyr Kernel Primer

Synchronization

Alerts

- kernel object that allows an application to perform asynchronous signaling when a condition of interest occurs
- use to minimize ISR processing
- alert handler function is invoked by the system workqueue thread

rndity;

Zephyr Kernel Primer

Data passing

Fifos

- kernel object that implements a traditional first in, first out (FIFO) queue, allowing threads and ISRs to add and remove data items of any size
- use to asynchronously transfer data items of arbitrary size in a “first in, first out” manner

Lifos

- kernel object that implements a traditional last in, first out (LIFO) queue, allowing threads and ISRs to add and remove data items of any size
- use to asynchronously transfer data items of arbitrary size in a “last in, first out” manner

Zephyr Kernel Primer

Data passing

Stacks

- kernel object that implements a traditional last in, first out (LIFO) queue, allowing threads and ISRs to add and remove a limited number of 32-bit data values
- use to store and retrieve 32-bit data values in a “last in, first out” manner, when the maximum number of stored items is known

Zephyr Kernel Primer

Data passing

Message Queues

- kernel object that implements a simple message queue, allowing threads and ISRs to asynchronously send and receive fixed-size data items
- use to transfer small data items between threads in an asynchronous manner

Mailboxes

- kernel object that provides enhanced message queue capabilities that go beyond the capabilities of a message queue object
- messages exchanged using a mailbox are handled non-anonymously, allowing both threads participating in an exchange to know the identity of the other thread (only)
- point-to-multipoint and broadcast messaging is not supported

Zephyr Kernel Primer

Data passing

Message Descriptor

- info
- size
- tx_data (**pointer to the sending thread's message buffer**)
- tx_block (**descriptor for the sending thread's memory block**)
- tx_target_thread
- rx_source_thread

rndity;

Zephyr Kernel Primer

Data passing

Pipes

- kernel object that allows a thread to send a byte stream to another thread
- use to transfer chunks of data in whole or in part, and either synchronously or asynchronously

Zephyr Kernel Primer

Interrupts

***Interrupt service routine* (ISR)** is a function that executes asynchronously in response to a hardware or software interrupt. ISR normally preempts the execution of the current thread.

Key properties:

- An **interrupt request (IRQ) signal** that triggers the ISR.
- A **priority level** associated with the IRQ.
- An i**nterrupt handler function** that is invoked to handle the interrupt.
- An **argument value** that is passed to that function.

Interrupt Descriptor Table or a vector table is used to associate a given interrupt source with a given ISR

The kernel supports **interrupt nesting**. This allows an ISR to be preempted in mid-execution.

Offloading ISR work: helper thread / signal an alarm.

rndity;

Supported boards

x86 Boards	ARM Boards	ARC Boards	NIOS II Boards	XTENSA Boards
Arduino / Genuino 101	96Boards Carbon / nRF51 / Neonkey / Nitrogen	Arduino / Genuino 101	Altera MAX10	ESP32
Galileo Gen1/Gen2	Arduino / Genuino 101 (BLE) / Arduino Due	DesignWare(R) ARC(R) EM Starter Kit		Xtensa Emulation (QEMU)
MinnowBoard Max	CC2650 SensorTag / CC3220SF LaunchXL			Xtensa simulator
X86 Emulation (QEMU)	NXP FRDM-K64F / FRDM-KL25Z / FRDM-KW41Z			
Quark D2000 Development Board	ST Disco L475 IOT01			
tinyTILE	Hexiwear / Hexiwear KW40Z			
	ST Nucleo F0/F3/F4/L4			
	OLIMEX STM32-E407 / STM32-P405			
	STM3210C-EVAL / STM32373C-EVAL			
	ARM V2M Beetle			

rndity;

Add your own board

Source Tree Structure

- arch
- boards
- doc
- drivers
- dts
- ext
- include
- kernel
- lib
- misc
- sample
- scripts
- subsys
- tests

Configuration in Zephyr comes from a number of different places:

- Kconfig files
- CMSIS header files
- vendor header files
- prj.conf files
- Device Tree

Add your own board

1. Architecture support
   - .dtsi files form base SoC
2. SoC family suport
   - one or more SoC family .dtsi files
3. Board specific suport
   - board level .dts file that includes the SoC family .dtsi files
4. Fixup files
   - mappings from existing Kconfig options to the actual underlying DTS derived configuration #defines

STM32 Nucleo F334R8:

- dts/arm/st/stm32f3.dtsi
- dts/arm/st/stm32f334.dtsi
- dts/arm/st/stm32f3-pinctrl.dtsi
- dts/arm/nucleo_f334r8.dts
- dts/arm/nucleo_f334r8.fixup

Add your own board

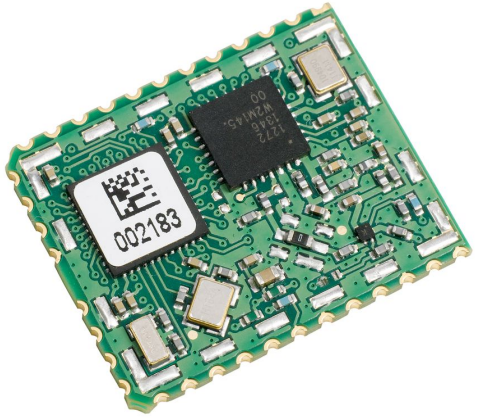
Device drivers

Interface	Controller	Driver/Component
NVIC	on-chip	nested vector interrupt controller
UART	on-chip	serial port-polling; serial port-interrupt
PINMUX	on-chip	pinmux
GPIO	on-chip	gpio
CLOCK	on-chip	reset and clock control
FLASH	on-chip	flash memory
IWDG	on-chip	independent watchdog

How we do it...

MANTA INSTRUMENTS

LoRa Alliance™



rndity;

Q/A

Wanna ask a question?

rndity;

Thank You