FreeRTOS v/s SafeRTOS
&
FreeRTOS v/s Zephyr OS

❑ Agenda

- ➔ FreeRTOS v/s SafeRTOS
  - ◆ Introduction
  - ◆ Features
  - ◆ Key Differences
- ➔ FreeRTOS v/s Zephyr
  - ◆ Introduction
  - ◆ Architecture
  - ◆ Zephyr Directory Structure
  - ◆ Why We Moved from FreeRTOS to Zephyr RTOS

SafeRTOS

❑ Introduction

- SAFERTOS is a safety critical, Real Time Operating System, delivering superior performance and pre-certified dependability, while using minimal resources.
- SafeRTOS is a safety focused upgrade of FreeRTOS.
- SafeRTOS was developed by Wittenstein High Integrity Systems, a safety systems company.
- SafeRTOS is designed and tested to comply with IEC61508 Safety Integrity Level 3, and with ISO26262, ASIL D.
- SafeRTOS follows the FreeRTOS functional model, upgrading is easy.
- SafeRTOS is not a open-source and requires commercial license for use.

SafeRTOS

❑ Features

- SafeRTOS offers an extremely fast boot time.
- Ultra-low power mode.
- No dynamic memory operations.
- Spatial separation of tasks
- Compact Footprint

  Typical ROM Requirements 6-15 kB.

  Typical RAM Requirements 500 bytes.

  Typical Stack Requirements 400 bytes/task.

SafeRTOS

❑ Key Differences

FreeRTOS and SafeRTOS share a similar usage model but are not direct drop-in replacements for each other, These differences primarily arise from:

- Memory allocation
- Restricted Functionality
- Function parameter checking
- Internal data checking

SafeRTOS

❑ Memory Allocation

- Each task and queue created consumes a small amount of RAM. Under FreeRTOS the required RAM is automatically dynamically allocated at run time.
- SafeRTOS does not permit dynamic memory allocation so the required RAM must instead be statically allocated at compile time, then manually passed into the create functions for the various kernel objects.
- In FreeRTOS MPU support is optional, whereas in SafeRTOS it is a mandatory to use MPU, as it is a key and central component.
- FreeRTOS projects include pvPortMalloc() and vPortFree() implementations in files like heap_1.c, heap_2.c, etc. These files are not required when using SafeRTOS.

SafeRTOS

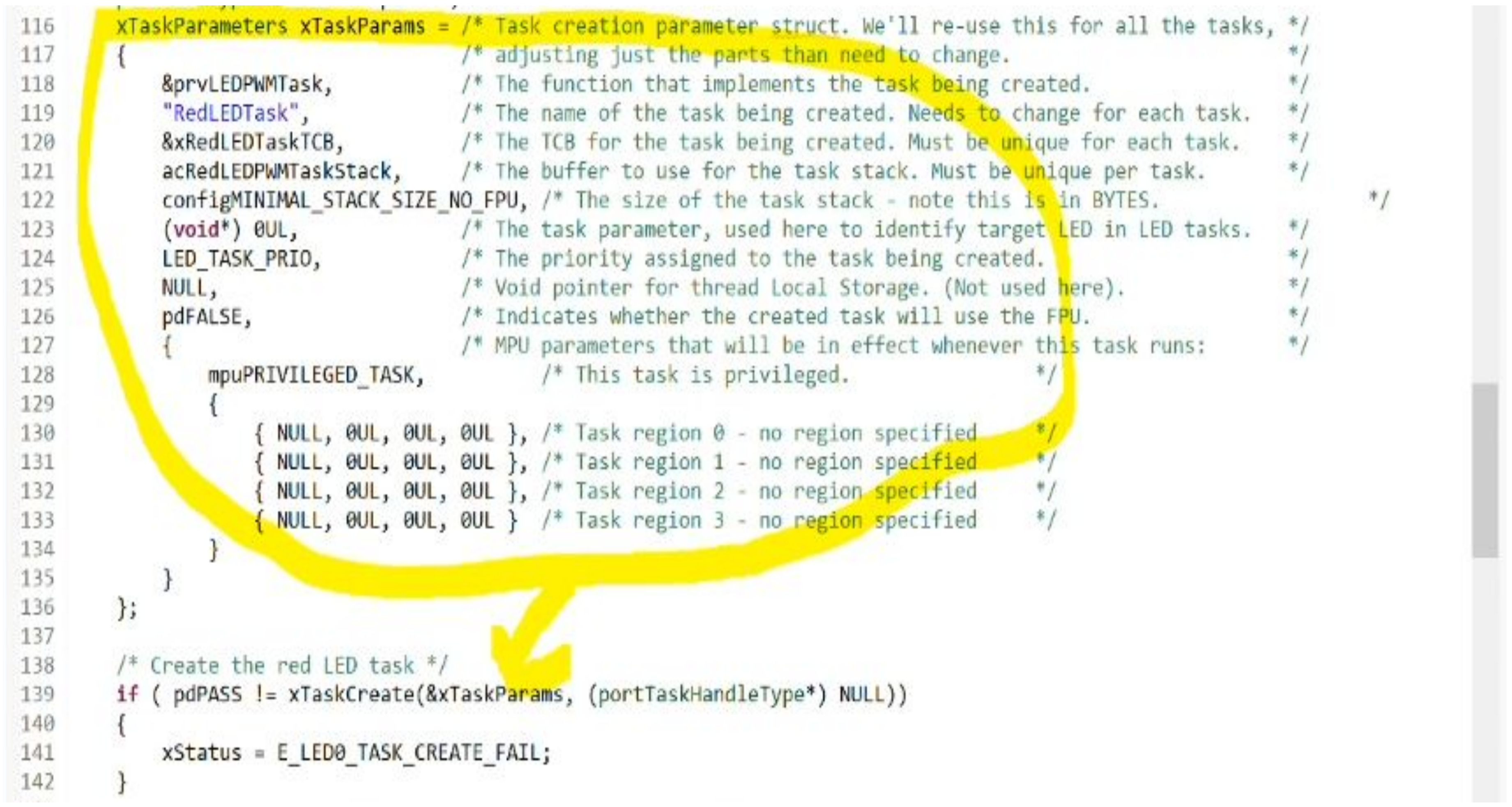
❑ Contd…

- **Changes to XTaskCreate()**
  - SafeRTOS version of xTaskCreate() requires more parameters than its FreeRTOS counterpart.
  - The SAFERTOS version of xTaskCreate() accepts 2 parameters
    - pxTaskParameters - which is a pointer to a xTaskParameters structure
    - pxCreatedTask - which is used to pass back the handle of the created task.
- portBaseType **xTaskCreate**(const xTaskParameters * const **pxTaskParameters**,

  portTaskHandleType * **pxCreatedTask**);

SafeRTOS

❑ Contd…



```c
xTaskParameters xTaskParams = /* Task creation parameter struct. We'll re-use this for all the tasks, */
{                             /* adjusting just the parts than need to change.                        */
    &prvLEDPWMTask,           /* The function that implements the task being created.                 */
    "RedLEDTask",             /* The name of the task being created. Needs to change for each task.   */
    &xRedLEDTaskTCB,          /* The TCB for the task being created. Must be unique for each task.    */
    acRedLEDPWMTaskStack,     /* The buffer to use for the task stack. Must be unique per task.       */
    configMINIMAL_STACK_SIZE_NO_FPU, /* The size of the task stack - note this is in BYTES.                  */
    (void*) 0UL,              /* The task parameter, used here to identify target LED in LED tasks.   */
    LED_TASK_PRIO,            /* The priority assigned to the task being created.                     */
    NULL,                     /* Void pointer for thread Local Storage. (Not used here).              */
    pdFALSE,                  /* Indicates whether the created task will use the FPU.                 */
    {                         /* MPU parameters that will be in effect whenever this task runs:       */
        mpuPRIVILEGED_TASK,          /* This task is privileged.                  */
        {
            { NULL, 0UL, 0UL, 0UL }, /* Task region 0 - no region specified       */
            { NULL, 0UL, 0UL, 0UL }, /* Task region 1 - no region specified       */
            { NULL, 0UL, 0UL, 0UL }, /* Task region 2 - no region specified       */
            { NULL, 0UL, 0UL, 0UL }  /* Task region 3 - no region specified       */
        }
    }
};

/* Create the red LED task */
if ( pdPASS != xTaskCreate(&xTaskParams, (portTaskHandleType*) NULL))
{
    xStatus = E_LED0_TASK_CREATE_FAIL;
}
```

SafeRTOS

❑ Contd…

- **Changes to XQueueCreate()**
  - SAFERTOS version of xQueueCreate() requires three more parameters than its FreeRTOS counterpart.
    - pcQueueMemory - Points to the statically declared buffer to be used by the kernel to hold the queue data structures and storage area.
    - uxBufferLength - The size of the buffer pointed to by the pcQueueMemory parameter.
    - pxQueue - Used to return a handle to the queue being created.

SafeRTOS

❑ Contd…

- Prototype of xQueueCreate in SafeRTOS

```
portBaseType xQueueCreate( portInt8Type *pcQueueMemory,
                           portUnsignedBaseType uxBufferLength,
                           portUnsignedBaseType uxQueueLength,
                           portUnsignedBaseType uxItemSize,
                           xQueueHandle *pxQueue );
```

- Other FreeRTOS APIs like xSemaphoreCreateBinary(), xSemaphoreCreateCounting(), xTimerCreate(), xEventGroupCreate(), xMutexCreate(), and xStreamBufferCreate(), etc also require modifications for static memory allocation in SafeRTOS.

SafeRTOS

❑ Restricted Functionality

The SafeRTOS functionality has been restricted to include only the core components necessary. This is standard practice for source code that is intended to undergo formal audit or certification as it greatly reduces the test burden.

- **Queue Sets :**
  - SafeRTOS does not provide an implementation of Queue Sets.
  - It implements the Event Multiplex API, which supersedes the FreeRTOS Queue Sets functionality.
  - An event multiplex object can be used to wait for the same event on multiple different target objects.
  - For example if there are multiple queues which a task wants to receive from, the task can use an event multiplex to wait for an item to be on any of the queues.

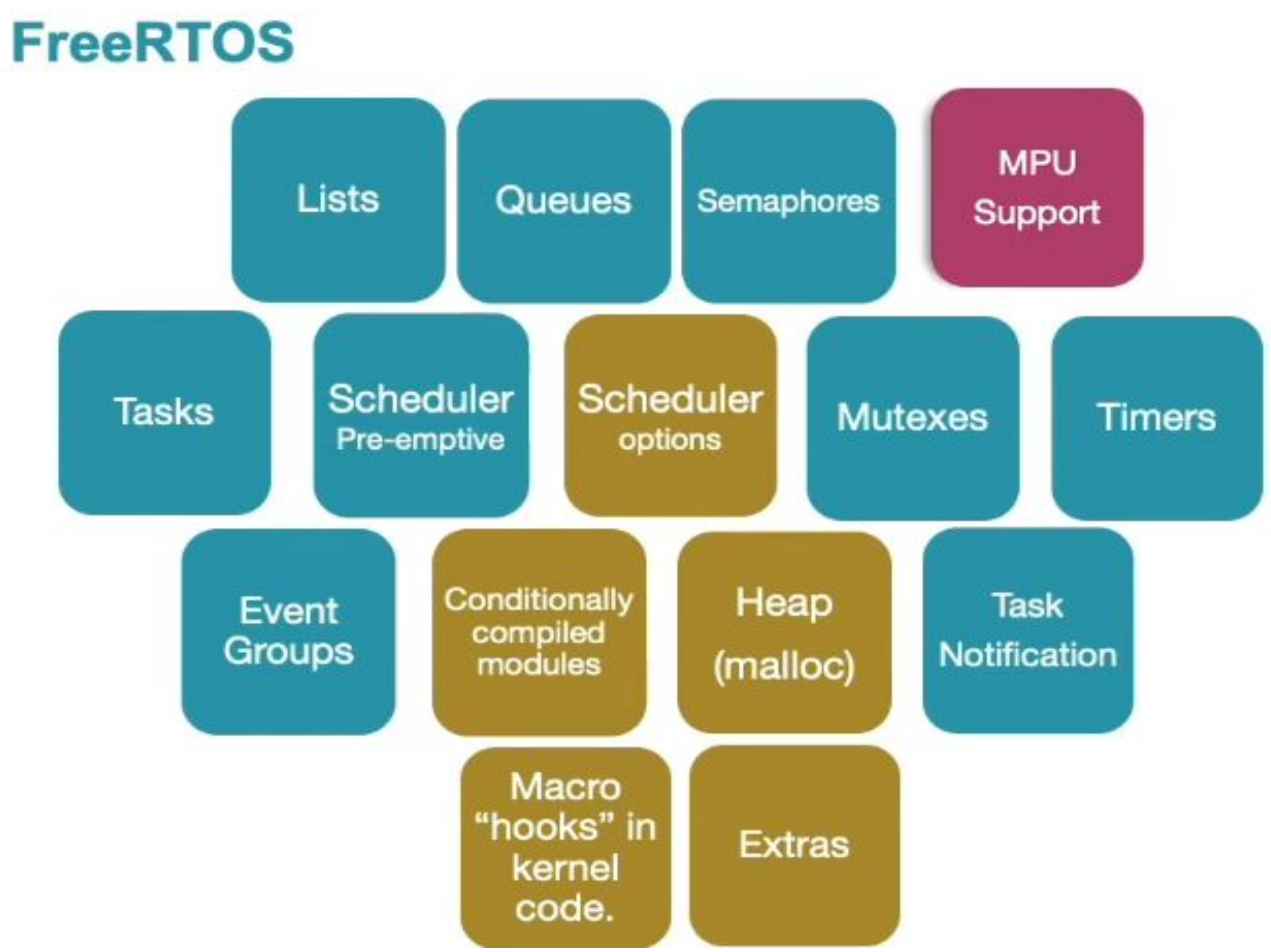
SafeRTOS

❑ Contd…

- **Co-routines** : SafeRTOS does not provide a co-routine implementation.
- **Thread Local Storage :**
  - FreeRTOS allows associating each task with a configurable number of pointers to Thread Local Storage (TLS) objects.
  - SafeRTOS always implements one TLS pointer, therefore the number is not configurable and cannot be disabled.
- **Task Tags :**
  - FreeRTOS supports the possibility of assigning an application-defined tag to each task.
  - SafeRTOS does not support task tags. However, it should be possible to use the TLS pointer instead.

SafeRTOS

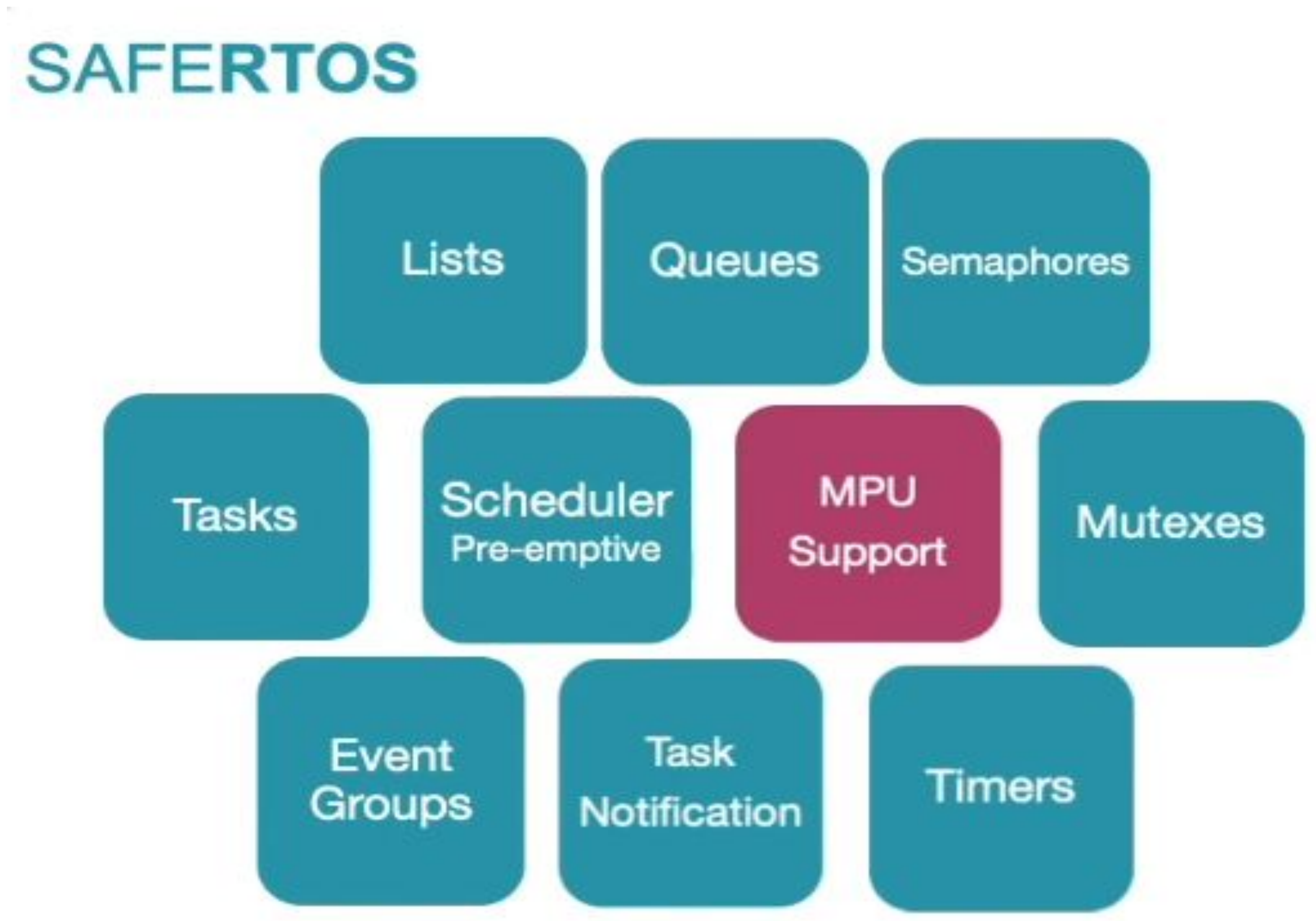
❑ Contd…



SoCtronics

SafeRTOS

- **Contd…**



SafeRTOS

❑ Function Parameter Checking

- FreeRTOS contains very little in the way of API function input parameter checking.
- As a result, many FreeRTOS API functions either just return a simple pass or fail result, or do not return any status information at all.
- SAFERTOS checks the validity of every appropriate input parameter.
- This means not only do more SAFERTOS API functions return status information, but the status information returned is also more detailed.

SafeRTOS

❑ Contd…

- **Example** :
  - In FreeRTOS, if the queue is created successfully then a **handle** to the created queue is returned. If the memory required to create the queue could not be allocated then **NULL** is returned.
  - In SafeRTOS;

    pdPASS - The queue was created successfully.

    errINVALID_ALIGNMENT - The alignment of the pcQueueMemory value was not correct for the target hardware.

    errINVALID_QUEUE_LENGTH - uxQueueLength was found to equal zero.

SafeRTOS

❑ **Contd…**

errINVALID_BUFFER_SIZE - uxBufferLengthBytes was found to not equal(uxQueueLength*uxItemSize)+portQUEUE_OVERHEAD_BYTES.

errNULL_PARAMETER_SUPPLIED - Either pcQueueMemory or pxQueue was NULL.

errQUEUE_ALREADY_IN_USE - The pcQueueMemory pointer refers to a buffer that is already holding another queue.

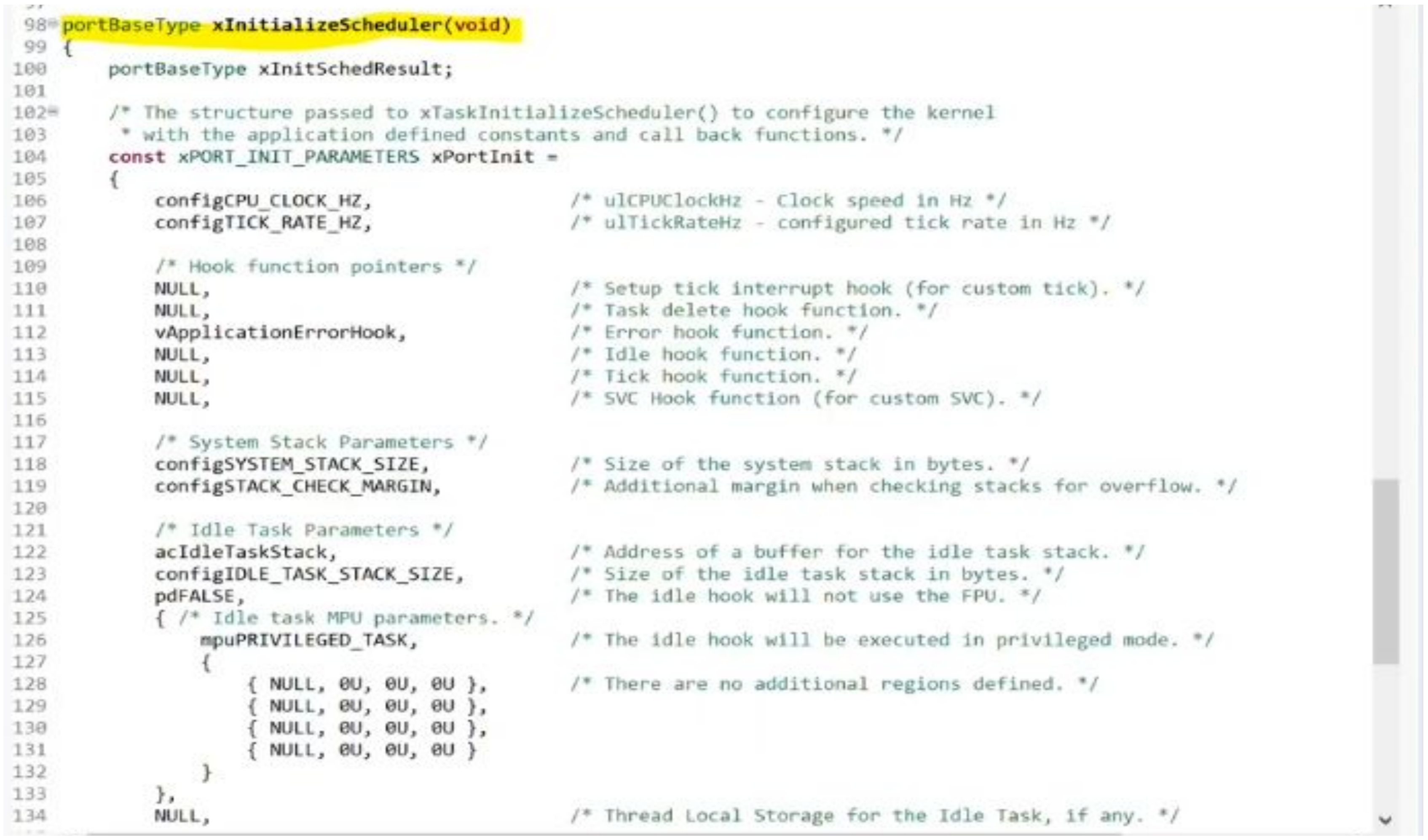
SafeRTOS

❑ Internal data checking

- FreeRTOS provides a configuration file, “FreeRTOSConfig.h”, which contains configuration parameters (macros) that modify the kernel's behavior according to the application needs.
- In SafeRTOS, application code cannot be built into the kernel as it may cause unpredictable behavior. Therefore, no configuration parameters (macros) are used.
- SafeRTOS includes a small set of standard hook functions that the application can supply during kernel configuration.
- The host application must provide an "Error Hook" function. Optionally, it can supply additional hooks: "Idle Hook", "Task Delete Hook", "System Call Hook", "Tick Hook", and other architecture-specific hooks.

SafeRTOS

❑ Contd…

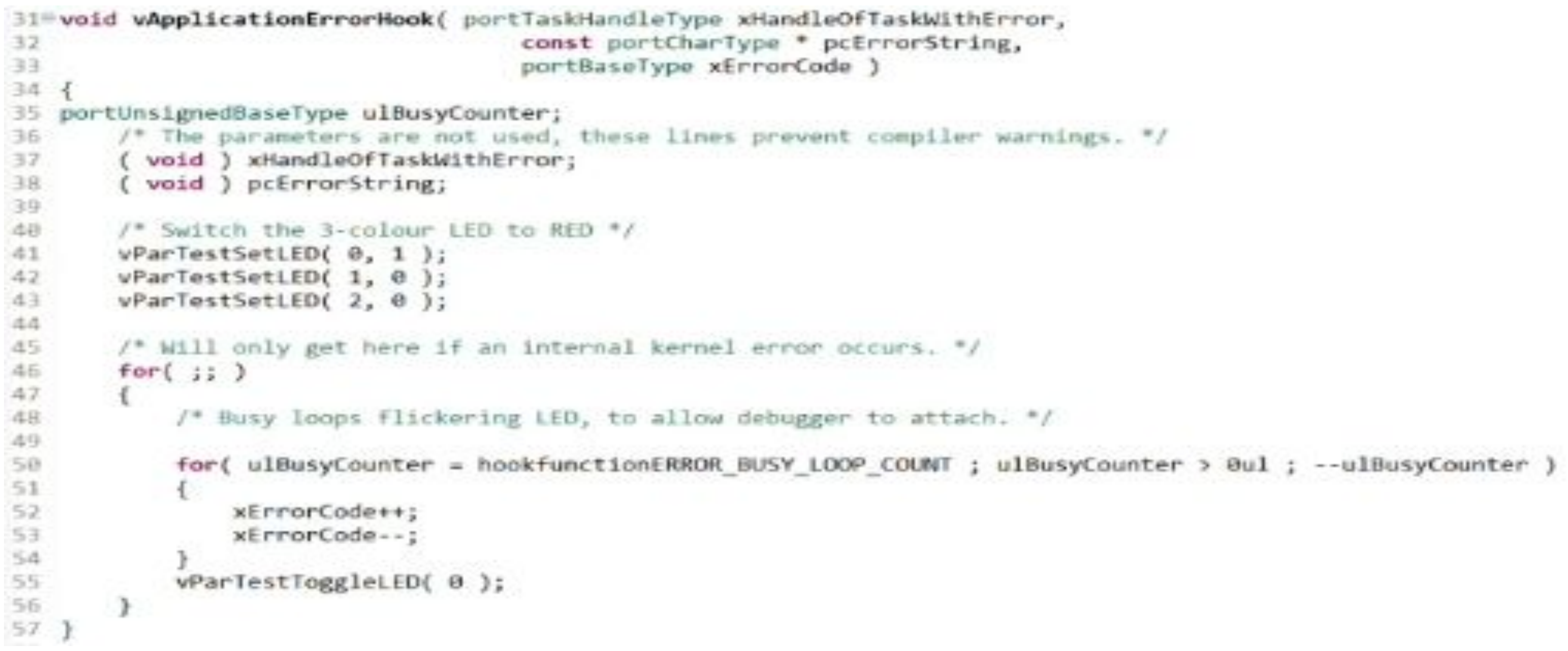


```
portBaseType xInitializeScheduler(void)
{
    portBaseType xInitSchedResult;

    /* The structure passed to xTaskInitializeScheduler() to configure the kernel
     * with the application defined constants and call back functions. */
    const xPORT_INIT_PARAMETERS xPortInit =
    {
        configCPU_CLOCK_HZ,                 /* ulCPUClockHz - Clock speed in Hz */
        configTICK_RATE_HZ,                 /* ulTickRateHz - configured tick rate in Hz */

        /* Hook function pointers */
        NULL,                               /* Setup tick interrupt hook (for custom tick). */
        NULL,                               /* Task delete hook function. */
        vApplicationErrorHook,              /* Error hook function. */
        NULL,                               /* Idle hook function. */
        NULL,                               /* Tick hook function. */
        NULL,                               /* SVC Hook function (for custom SVC). */

        /* System Stack Parameters */
        configSYSTEM_STACK_SIZE,            /* Size of the system stack in bytes. */
        configSTACK_CHECK_MARGIN,           /* Additional margin when checking stacks for overflow. */

        /* Idle Task Parameters */
        acIdleTaskStack,                    /* Address of a buffer for the idle task stack. */
        configIDLE_TASK_STACK_SIZE,         /* Size of the idle task stack in bytes. */
        pdFALSE,                            /* The idle hook will not use the FPU. */
        { /* Idle task MPU parameters. */
            mpuPRIVILEGED_TASK,             /* The idle hook will be executed in privileged mode. */
            {
                { NULL, 0U, 0U, 0U },       /* There are no additional regions defined. */
                { NULL, 0U, 0U, 0U },
                { NULL, 0U, 0U, 0U },
                { NULL, 0U, 0U, 0U }
            }
        },
        NULL,                               /* Thread Local Storage for the Idle Task, if any. */
```

SafeRTOS

❑ Contd…

- If SafeRTOS detects any errors like memory corruption, stack overflow or other error from which it is unsafe to attempt any sort of recovery then Error hook function is called.
- Error hook function is responsible for ensuring that system enters a safe state.

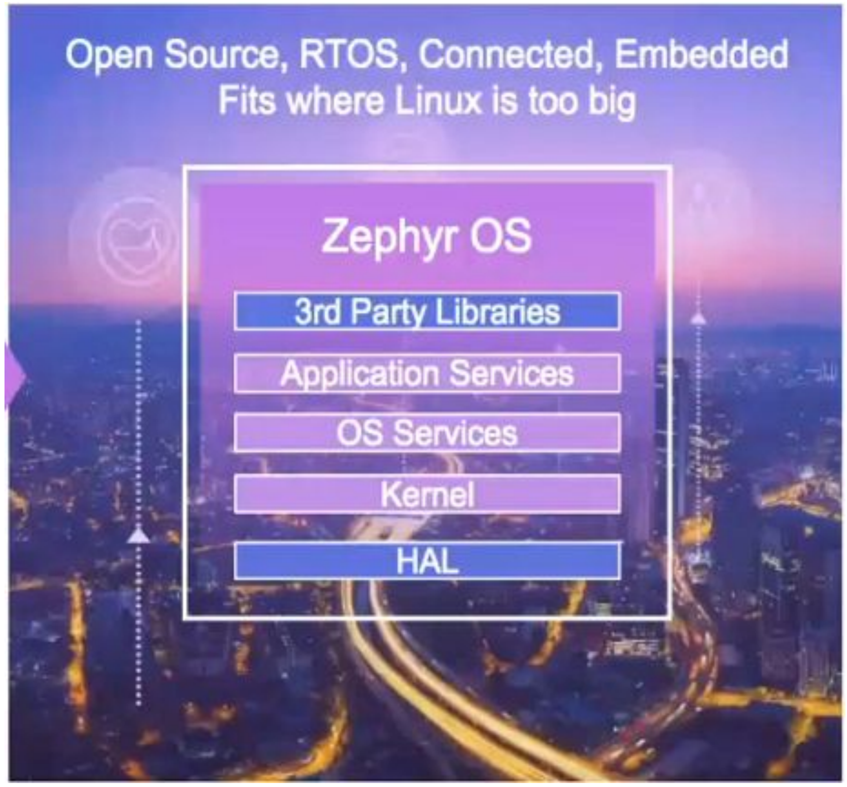


```c
void vApplicationErrorHook( portTaskHandleType xHandleOfTaskWithError,
                                    const portCharType * pcErrorString,
                                    portBaseType xErrorCode )
{
portUnsignedBaseType ulBusyCounter;
    /* The parameters are not used, these lines prevent compiler warnings. */
    ( void ) xHandleOfTaskWithError;
    ( void ) pcErrorString;

    /* Switch the 3-colour LED to RED */
    vParTestSetLED( 0, 1 );
    vParTestSetLED( 1, 0 );
    vParTestSetLED( 2, 0 );

    /* Will only get here if an internal kernel error occurs. */
    for( ;; )
    {
        /* Busy loops flickering LED, to allow debugger to attach. */

        for( ulBusyCounter = hookfunctionERROR_BUSY_LOOP_COUNT ; ulBusyCounter > 0ul ; --ulBusyCounter )
        {
            xErrorCode++;
            xErrorCode--;
        }
        vParTestToggleLED( 0 );
    }
}
```

Zephyr

❑ Introduction

- The Zephyr OS is based on a small-footprint kernel designed for use on resource - constrained and embedded systems.
  - Fits where linux is too big
- Zephyr OS is built to be secure and safe.
- Zephyr OS is fully integrated, highly configurable, modular and flexibe.
- Highly connected
  - Bluetooth 5.0 & BLE
  - Wi-Fi, Ethernet, CANbus, open thread..
  - USB & USB-C



Zephyr

❑ Supported Hardware Architectures

ARC

arm
Cortex-M, Cortex-R & Cortex-A

intel
x86 & x86_64

MIPS

Nios II Processor

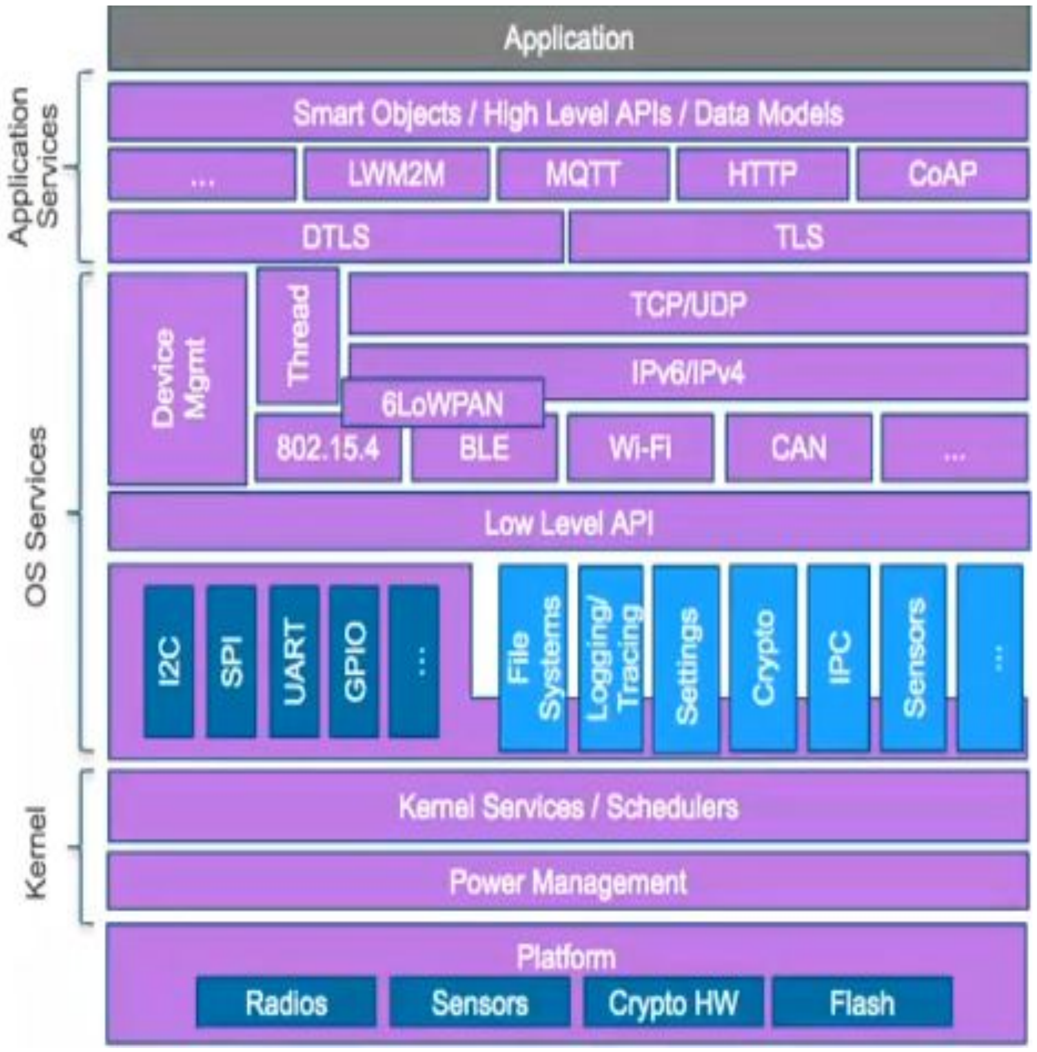
RISC-V
32 & 64 bit

SPARC

tensilica
Xtensa

Zephyr

❑ Architecture



- Highly configurable and modular
- cooperative and preemptive threading
- An integrated device driver interface.
- Memory protection, stack overflow protection.
- Supports Bluetooth® Low Energy (BLE 5.1).
- 802.15.4 OpenThread.
- A native, fully featured and optimized networking stack.
- Supports a range of subsystems, including USB, filesystem, logging.

Zephyr

❑ Zephyr Directory Structure

- **Main Folders:**

```
├── arch
├── boards
├── doc
├── drivers
├── include
├── kernel
├── lib
├── scripts
├── soc
├── subsys
├── tests
└── samples
```

Zephyr

❑ Contd…

```
Zephyr Root
├── arch
├── boards
├── drivers
├── dts
├── include
├── kernel
├── lib
├── modules
├── soc
├── subsys
├── tests
└── samples
```

Contains architecture-specific code, such as CPU and board definitions. Each supported architecture (e.g., ARM, x86) has its own subdirectory.

- **Purpose**: Handles low-level details of each CPU architecture.
- **FreeRTOS**: The equivalent would be the *portable* directory which contains architecture-specific code for different compilers and processors.

Zephyr

❑ Contd…

```
Zephyr Root
├── arch
├── boards
├── drivers
├── dts
├── include
├── kernel
├── lib
├── modules
├── soc
├── subsys
├── tests
└── samples
```

Contains board-specific configurations. Each supported board has a subdirectory with its specific settings.

- **Purpose**: Provides device tree files and configuration for different hardware boards.
- **FreeRTOS**: FreeRTOS does not have a direct equivalent; board support is typically found in *demo* applications or third-party repositories.

Zephyr

❑ Contd…

```
Zephyr Root
├── arch
├── boards
├── drivers
├── dts
├── include
├── kernel
├── lib
├── modules
├── soc
├── subsys
├── tests
└── samples
```

Contains hardware drivers for peripherals like UART, I2C, SPI, etc.

- **Purpose**: Implements hardware abstraction layers to interact with various peripherals.
- **FreeRTOS**: Provides basic drivers like gpio, rcc, .. in *demo* folder, needs manual integration of additional drivers.

Zephyr

❑ Contd…

```
Zephyr Root
├── arch
├── boards
├── drivers
├── dts
├── include
├── kernel
├── lib
├── modules
├── soc
├── subsys
├── tests
└── samples
```

Contains Device Tree Source include(.dtsi) files which describe hardware components.

- **Purpose**: Provides a standardized way to describe the hardware layout.
- **FreeRTOS**: FreeRTOS does not use Device Trees; hardware descriptions are typically hardcoded or configured through header files.

Zephyr

❑ Contd…

```
Zephyr Root
├── arch
├── boards
├── drivers
├── dts
├── include
├── kernel
├── lib
├── modules
├── soc
├── subsys
├── tests
└── samples
```

Contains global header files used throughout the Zephyr project.

- **Purpose**: Provides common definitions and macros accessible by other parts of the OS.
- **FreeRTOS**: The *include* directory serves a similar purpose, providing global header files and APIs.

Zephyr

❑ Contd…

```
Zephyr Root
├── arch
├── boards
├── drivers
├── dts
├── include
├── kernel
├── lib
├── modules
├── soc
├── subsys
├── tests
└── samples
```

Contains the core RTOS kernel code.

- **Purpose**: Implements fundamental RTOS functionalities such as thread scheduling, inter-task communication, and synchronization.
- **FreeRTOS**: The *FreeRTOS/Source* directory contains the core kernel code.

Zephyr

❑ Contd…

```
Zephyr Root
├── arch
├── boards
├── drivers
├── dts
├── include
├── kernel
├── lib
├── modules
├── soc
├── subsys
├── tests
└── samples
```

Contains libraries that provide various utilities and additional functionality.

- **Purpose**: Offers extended capabilities beyond the core kernel.
- **FreeRTOS**: Does not include extensive additional libraries like Zephyr.Additional libraries need to be integrated separately.

Zephyr

❑ Contd…

```
Zephyr Root
├── arch
├── boards
├── drivers
├── dts
├── include
├── kernel
├── lib
├── modules
├── soc
├── subsys
├── tests
└── samples
```

The modules directory in Zephyr serves as a place to integrate external modules and libraries into the Zephyr project.

- **Purpose**: Offers extended capabilities beyond the core kernel.
- **FreeRTOS**: FreeRTOS does not have a direct equivalent of the modules directory found in Zephyr. Requires more manual integration of external modules.

Zephyr

❑ Contd…

```
Zephyr Root
├── arch
├── boards
├── drivers
├── dts
├── include
├── kernel
├── lib
├── modules
├── soc
├── subsys
├── tests
└── samples
```

Contains System-on-Chip specific code.

- **Purpose**: Handles initialization and configuration for specific SoCs.
- **FreeRTOS**: Similar configurations might be found in board support packages or vendor-provided code.

Zephyr

❑ Contd…

```
Zephyr Root
├── arch
├── boards
├── drivers
├── dts
├── include
├── kernel
├── lib
├── modules
├── soc
├── subsys
├── tests
└── samples
```

Contains various subsystems like USB, dsp, and Bluetooth.

- **Purpose**: Implements modular subsystems that provide additional services and capabilities.
- **FreeRTOS**: Similar functionalities might be implemented as separate modules or libraries in third-party extensions.

Zephyr

❑ Contd…

```
Zephyr Root
├── arch
├── boards
├── drivers
├── dts
├── include
├── kernel
├── lib
├── modules
├── soc
├── subsys
├── tests
└── samples
```

Contains test cases for validating the functionality of Zephyr components.

- **Purpose**: Ensures reliability and correctness through automated tests.
- **FreeRTOS**: The *FreeRTOS/Test* directory contains test suites for validating the FreeRTOS kernel and components.

Zephyr

❑ Contd…

```
Zephyr Root
├── arch
├── boards
├── drivers
├── dts
├── include
├── kernel
├── lib
├── modules
├── soc
├── subsys
├── tests
└── samples
```

Contains example applications demonstrating various features of Zephyr.

- **Purpose**: Provides reference implementations and starting points for new projects.
- **FreeRTOS**: The *FreeRTOS/Demo* directory contains example applications for various hardware platforms.

Zephyr

❑ Why We Moved from FreeRTOS to Zephyr RTOS

- To understand the difference between FreeRTOS and Zephyr, let's refer to a blog written by **Stephen Berard**, Chief Technology Officer at **Nubix**, on March 28, 2023.
- Overview of Nubix's Mission:
  - Enable cloud-native development on resource-constrained edge devices.
  - Simplify building, deploying, managing, and securing applications at the edge.
- Need for a Flexible RTOS:
  - Development for microcontrollers (MCUs) is time-consuming and rigid.
  - A more adaptable RTOS is necessary to support diverse hardware, integrate new technologies, and meet the dynamic needs of modern applications.

Zephyr

❑ Initial Use of FreeRTOS

- Popularity and Limitations:
  - Widely used in embedded applications due to its simplicity and lightweight nature.
  - Limited library support and flexibility hindered development.
- Challenges Encountered:
  - Managing multiple libraries and middleware was difficult.
  - Lack of a comprehensive set of tools and toolchains for development.

Zephyr

Key Differences

- Modularity and Flexibility
  - **FreeRTOS**:
    - Simplicity in design but less modular.
    - Requires manual integration of additional features and libraries.
  - **Zephyr:**
    - Highly modular and configurable.
    - Uses CMake and Kconfig for easy building and configuration.
    - Supports a wide range of devices, from simple microcontrollers to complex systems.

Zephyr

❑ Contd…

- Security and Reliability
  - **FreeRTOS:**
    - Basic security features suitable for simple applications.
  - **Zephyr:**
    - Robust and secure kernel with multiple scheduling algorithms.
    - Features like cryptography, secure boot, and firmware updates ensure high security.

Zephyr

- **Contd…**
  - Driver Libraries
    - **FreeRTOS:**
      - Basic driver support requiring manual integration.
    - **Zephyr:**
      - Built-in driver interfaces for a wide range of peripherals.
      - Supports IP networking, wireless communications (Wi-Fi, Bluetooth, LoRaWAN), and power management.
      - Provides consistent hardware abstraction layer through device class APIs.

Zephyr

❑ Contd…

- Board Support
    - **FreeRTOS:**
        - Limited and less standardized board support.
    - **Zephyr:**
        - Over 450 supported boards from various architectures and vendors.
        - Uses Devicetree for easy configuration and support of new boards.
        - Quick onboarding of new edge devices.

Zephyr

❑ Contd…

- Power Management
  - **FreeRTOS:**
    - Limited capabilities for managing device power.
  - **Zephyr:**
    - Advanced device power management features.
    - Automatically enters low-power states when not in use.
    - Ensures efficient power usage and long battery life.

THANK YOU