1. Introduction

The embedded world of today ranges from simple controllers to blazingly fast processors, with software being the "brain" of any device. An embedded system is a unique computer system with its components assembled to perform specific tasks. An Operating System (OS) manages both hardware and software components of an electronic device. The OS is crucial to prioritize and organize the software and hardware interaction routines. Smartphones, Cameras, Washing Machines, hair straighteners, and ATMs are typical examples of Embedded Systems.

2. Objectives

Upon completion of this module, you will be able to:

- Understand the basics of Embedded Operating Systems
- Select the right OS for your needs
- Understand the basics of Android OS
- Understand Windows for Embedded applications
- Understand RTOS

3. Scope

An embedded OS is a distinct operating system running on purpose-built hardware configured for specific applications. It is written to suit its particular hardware and consequently incompatible with other hardware systems having different configurations. An OS is comprised of a kernel and a shell:

- The kernel is a perpetually running program (for example, a device driver) that interacts with the hardware to play an active role in the OS.

- The shell is a program set used to communicate with application programs. It is also known as a command interpreter, since it executes instructions sent to the OS via
commands. The OS shoulders the responsibility of managing system peripherals and scheduling software tasks to ensure that each program gets processor time.

An embedded OS is available in both open source and commercial variants. OS functionality ranges from basic task schedulers to feature-rich OSs enriched with value additions like messaging, protocol stacks, device driver frameworks, and memory management.

The scope of this learning module is to cover the essentials of embedded operating systems, including their purpose, function, and some basic concepts, as well as a discussion on Embedded Linux, Android, Windows CE, and RTOS.

4. Basic Concepts

Embedded systems OSs are written to execute a specific task set, and may run on battery power. They are comparatively more power-efficient. Embedded OSs (like Embedded Linux) are highly configurable, allowing OS customization for precise application requirements.

The process of choosing an Embedded OS must weigh factors like processor architecture. The latter may be symmetrical or asymmetrical multi-core. A good example is an Arm-based architecture with Cortex-A and Cortex-M microcontroller core-based cores like single-, dual-, quad- or octa- cores.

They must deliver the best possible run-time performance, depending on the selected processor and connected peripherals. Embedded OSs are suitable for custom builds as dictated by application requirements.

Tiny microcontrollers can run several Embedded OSs, ranging from limited, sparse memory 8-bit architectures to all-frills OSs suitable for demanding high-end applications running higher memory 32-bit processors architecture.

Embedded systems can be broadly classified as either OS-based or without them. OS-less systems—as the name implies—lack any Operating System. The software, utilizing CPU resources, runs directly on the hardware. Such systems are usually observed in low end embedded systems executing a simple task. Conversely, OS-centric embedded systems are designed for real-time applications and deliver time-critical performance.
4.1 Kernel

A kernel is a memory resident software constituting the central core of an OS. It has several key functions such as resource management, task management, task scheduling, I/O management, memory management, intertask communication, interrupt management, and time management.

The kernel's Application Program Interface (API) gets called to request a service. A kernel resides in memory, and it is active during real-time operations. One important kernel function is to process those interrupts that take place as a consequence of internal or external events. If an interrupt takes place, the processor transfers control to a specific interrupt service routine, which logs that interrupt. A message gets sent to the scheduler and reverts to active code. The scheduler subsequently sets up the individual tasks' execution order to confirm that a higher-priority task preempts a lower-priority task. A deterministic response is maintained.

Figure 2: OS Architecture

The main kernel can be used to create multiple Linux distributions like Ubuntu, Arch Linux, Linux Mint Cinnamon, MX Linux, Elementary OS, and Gentoo.

5. Analysis

In this section, we will discuss different embedded operating systems used today.

5.1 Embedded Linux

Embedded Linux is best described as a compact version of Linux. It is designed to be installed and used within embedded devices. The OS provides services and features as per the operating needs and application of the embedded system.
An Embedded Linux for a particular system is bespoke for that embedded system, with a characteristic smaller size, reduced processing power, and minimal features. The demands of the underlying embedded system influence Linux kernel modifications and optimization of the embedded Linux version. Embedded Linux can exclusively manage device-specific purpose-built applications.

Android is a prominent example of an embedded Linux. The Google product is based on a modified Linux kernel and was released as an open-source license. Such an arrangement permits manufacturers to alter the written code to suit their specific hardware.

A Linux kernel is a fully preemptive multitasking OS. Other than real-time property support, Linux also supports threads and processes. The Linux kernel scheduler can schedule within a fixed time. A Linux kernel differentiates between real-time processes. The latter exist outside scheduler logic and enjoy top priority. If processes have the same priority, round-robin scheduling is used. The Linux kernel supports dynamic priority-based scheduling for the real-time property.

5.2 Building a Custom Image for Linux OS

The embedded systems programming development environment is distinct from other OSs since it uses different software stacks, operating systems, and chip architectures. A typical build will be a full software image, complete with the kernel, application software, device drivers, and libraries for that particular target device. The project’s hardware or software demands dictate Linux image optimization.

Several methods can be used to put together Embedded Linux systems.

Yocto, Desktop distros, OpenWRT/LEDE, and Buildroot are examples of a few such techniques.

The Yocto project is an open-source collaborative effort offering templates, methods, and tools to aid the creation of custom Linux-based systems suitable for embedded products independent of hardware architecture. The Buildroot project generates the embedded Linux systems via cross-compilation. Both Buildroot and Yocto share common minimalism and simplicity objectives.

A configured build environment is compulsory for building any Embedded Linux image. Toolchains like GCC, Binutils, and relevant software packages must be installed with the vital "development tools" package to compile and build the custom kernel. The generated file post-final build is labeled 'zImage.' It is board-specific and must be copied to eMMC memory or the board SD Card.

There are few community-supported and open source development boards like BeagleBone, Arduino, and Raspberry Pi. The Arm-based Linux kernel can run on Raspberry Pi boards, along with several OSs. The different types of Embedded OS that
run on this board include OpenELEC (Open Embedded Linux Entertainment Center), Retropie, RISC OS, Free BSD, LinuTop OS, and Windows 10 IoT.

OpenELEC (Open Embedded Linux Entertainment Center) is a Linux-derived Just Enough Operating System (JeOS) coded from scratch to convert PCs into Kodi media centers. The Windows IoT Core is a bespoke Windows OS crafted exclusively for Raspberry Pi. Coders and programmers can use it to craft IoT prototypes using Windows 10 and Raspberry Pi. For the BeagleBone Black, the default OS is Angstrom. Other Linux distro OSs in common use include Android, Ubuntu, Fedora, Sabayon, Debian, and Gentoo.

Debian engineered Raspbian is crafted specially for Raspberry Pi. The Open Source Media Center (OSMC) is a Kodi OS that can play almost any media format. The OSMC is simple, open-source, and free. It is also easy-to-use.

Retro games can be emulated on the open-source Debian based RetroPie. It is a software library where older games can be emulated on ODroid C1/C2, Raspberry Pi, and a PC. Ubuntu Core is the Ubuntu version specially written for IoT applications.

- **5.3 Android**

The mobile OS 'Android' is a Linux distro enclosing a Linux kernel and additional software like utilities, libraries, GUI, and pre-installed applications. It debuted as open-source code. You can work in Linux, Windows or Mac to further develop the OS. Google wrote Dalvik, a virtual machine created particularly for Android. It runs recompiled Java code and subsequently reads it as Dalvik bytecode. Google designed it to sustain functionality and optimize battery power in an ecosystem constrained by limited CPU power and memory.

One of Android's many differentiating characteristics is its ability to break application boundaries. It helps that the OS can be easily developed. Android Studio is the official Integrated Development (IDE) for the Android platform by Google.

Android Studio helps you access the Android Software Development Kit or Android SDK. It is a Java code extension that runs smoothly on Android devices and takes advantage of native hardware. Programs are written in Java, and the Android SDK runs such programs on the Android platform. The Android Studio consolidates the job and runs the code, either through an emulator or via a hardware piece connected to the machine.

- **5.4 Windows for Embedded Applications**

Windows 10 IoT Core is a Windows 10 version optimized for smaller devices. A display is optional in this iteration. It runs on both x86/x64 and Arm devices.
Windows CE is a preemptive multitasking OS supporting threads and processes. The preemption conforms solely to thread priority. Higher priority threads preempt lower priority threads at run time. The same priority threads get scheduled as per the round-robin method.

The Hardware Abstraction Layer (HAL) interacts with both the hardware device and the OEM Adaptation Layer (OAL). This code layer is logically sandwiched between the kernel of Windows Embedded CE and the target device hardware, with both layers residing at the bottom and as a component of Windows CE porting. The Graphics Windowing and Events Subsystem (GWES) lies on top of these layers, along with the kernel and communication stacks.

The Remote API (RAPI) capability lies at the top of communication functionality. The file system and database are found on the top of the kernel and can be accessed by RAPI calls. It is available to applications via Win32 interface. An application program runs within its own address space. It interacts with the remaining Windows CE through the Win32 system call interface.

- **5.5 RTOS**

An RTOS is a multitasking operating system written for real-time applications. RTOSs come into play if support is needed for expanding functions, standard communications, and interrupt sources interfaces. An increasing number of MCU applications have an RTOS as the base. Application complexity influences RTOS adoption.

The RTOS enables applications to be more portable. It is achieved by offering a standardized stack and driver API set, thus abstracting the underlying hardware's
specifics. An RTOS generally offers communications stacks like TCP/IP and USB and also their drivers and other peripherals. Such "out of box" software negates a developer's need to spend time integrating them into third party software or from scratch implementation into the concerned run-time environment.

Good RTOSs can be scaled to satisfy the varied requirements set for multiple applications. An RTOS can be a combination of different modules, including the kernel, Portable Operating System Interface (POSIX), ANSI C Library, networking protocol stacks, device drivers, Debugging facilities, and File Systems. The probable components are illustrated in figure 4.

RTOSs are classified into three types: Hard Real-Time, Firm Real-Time, and Soft Real-Time. The degree of tolerance in Hard real-time is minute or zero for missed deadlines and finds extensive use in mission-critical systems. There may be an unacceptable quality reduction in Firm real-time if a deadline is missed. Soft real-time allows frequently missed deadlines. A few popular RTOSs include ChibiOS/RT, eCos, FreeRTOS, Freescale MQX, LibreRTOS, mbed OS, Mongoose OS, Phoenix, RT-Thread, TinyOS, Zephyr and many more.

**VxWorks**

VxWorks is a preemptive multitasking OS. The OS is comprised of two scheduling algorithms: round-robin scheduling and priority scheduling. Each thread or process is accorded priority in priority scheduling. The highest priority thread must be executed first, and so on. The same priority processes get executed according to their arrival time, with the first process being the first to be completed. Decisions related to priority are based on memory, time, or other resource need. The CPU uses Round-robin as the
scheduling algorithm process execution. It is designed specifically for time-sharing systems.

**μC/OS-II**
μC/OS-II is a completely portable, ROMable, scalable, preemptive, real-time, multitasking kernel. It consists of simple OS services like task management, memory management, and time management. It is another example of a preemptive multitasking OS. Every task carries a unique priority, and the task with the highest priority always runs first. The uC/OS-II solely supports the priority-centric scheduling method and does not support round-robin scheduling. There can be a maximum of 64 tasks.

**Keil RTX Real-Time Operating System**
This is a royalty-free and deterministic RTOS specially written for Cortex-M and Arm devices. The created programs perform multiple functions simultaneously, and assist in the creation of better structured and easily maintainable applications. The all-frills commercial level RTX (inclusive of Source Code) is offered as part of Keil MDK-ARM development tools.

The Keil RTOS extracts minimum MCU resources and runs quickly with minuscule memory footprint- as little as 5KB (ROM). It is able to support μVision IDE/Debugger, RTX task aware tools enabling you to not only easily and quickly configure the code but also debug RTX in applications.

**Mbed OS**
Arm Mbed OS is a free, open-source, embedded operating system designed specifically for the "things" in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS, and drivers for sensors and I/O devices. The modularity of Mbed OS enables it to run on Cortex-M0 to Cortex-M4 based devices, with a variety of communications protocols.
### Table 1: Mbed OS

<table>
<thead>
<tr>
<th>Key Mbed OS Components</th>
<th>BLE Beacon</th>
<th>WiFi Appliance</th>
<th>Thread Device</th>
<th>Sub-GHz Mesh</th>
<th>LoRa Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RTOS, Drivers, BLE</td>
<td>RTOS, TLS, Client</td>
<td>RTOS, Thread, TLS, Client</td>
<td>RTOS, 6LoWPAN Mesh, TLS, Client</td>
<td>RTOS Drivers, LoRa Library</td>
</tr>
<tr>
<td>Hardware Components</td>
<td>Cortex-M0 with BLE radio</td>
<td>Cortex-M3 with WiFi Network Co-processor</td>
<td>Cortex-M4 with 2.4 GHz 802.15.4 and Crypto</td>
<td>Cortex-M3 with 802.15.4 Transceiver</td>
<td>Cortex-M0 with LoRa Transceiver</td>
</tr>
</tbody>
</table>

### 5.6 Open source OS

Open-source software, like its proprietary counterparts, is licensed. The difference between the two is that the majority of the open-source licenses need the software source to be freely available to any person who expresses the desire to use it. A user can freely modify the software and also distribute it. The software source must be available at all times. Multiple open-source licenses are currently in use, including the well-known GNU General Public License (GPL). Others include the Netscape public license, the MIT/X license, and the BSD license. Although the licenses are different, all are based on software source availability. The Linux OS is currently the most visible outcome of the open-source-based model.

### 6. Safety Certification

Safety certification programs are found in several industries, including industrial, automotive, aerospace, and medical. The certification process is complex and expensive. The OS source code is essential. A smaller OS is better in this case, as the size of the certified code influences process expense, with bigger ones being more expensive.

An RTOS precertification assures the user that the RTOS satisfies the rigorous design, documentation, test, and quality needs of the certifying agency. All application developers enjoy benefits, whether their particular products require certification or not. Common safety certifications include IEC 61508 SIL 4, IEC 62304 Class C, ISO 26262 ASIL D, UL/IEC 60730, UL/IEC 60335, UL 1998, and EN 50128 SW-SIL 4.

A few RTOS vendors offer a "pre-certification pack." It provides the initial documentation required concerning the OS. It is recommended to select an OS having a reliable track record and certified in similar application areas.
7. Glossary

- **RTOS**: A real-time operating system (RTOS) is an operating system (OS) intended to serve real-time applications responsible for managing hardware resources and hosting applications that run on the target board.

- **Open-source**: The term "open source" refers to something people can modify and share because its design is publicly accessible. Open-source software is software with source code that anyone can inspect, modify, and enhance.

- **SDK (Software Development Kit)**: An SDK is a collection of software used for developing applications for a specific device or operating system, certain software packages, software frameworks, hardware platforms, computer systems, or similar development platforms.

- **Application Program Interface (API)**: This is a programming interface that allows each application to connect, communicate, and share information with the other. This is a set of routines, protocols, and tools for building software and applications.

- **Board Specific Package (BSP)**: This is a standardized interface between hardware and the operating system. It provides an interface to device drivers, which in turn allows the kernel to communicate with the hardware resource.

- **Embedded system**: A combination of hardware and software that is designed to carry out a certain specific task or function.

- **GPL**: General Public License (GPL) is the most commonly used free software license. The GNU General Public License is a free, copyleft license for software and other kinds of works.

- **GCC**: GCC stands for "GNU Compiler Collection". GCC is an integrated distribution of compilers for several major programming languages.

- **Device Driver**: A software component that lets the operating system and a hardware device communicate with each other.

*Trademark. **element14 is a trademark of AVNET**. Other logos, product and/or company names may be trademarks of their respective owners.

[Take Quiz]