Migrating an embedded systems lab to Zephyr – an experience report

Olaf Hagendorf[1, 2, †], Niklas Huhs[2, †]

[2] *Hochschule Wismar, Philipp-Müller-Str. 14 23966 Wismar, Germany*

Abstract

With the deprecation of Mbed OS and increasing demands from both industry and students for modern, scalable embedded education, an embedded development lab, formerly based on Mbed OS had to be migrated to a new framework. Existing hardware based on STM32 Nucleo-144 boards and a custom-developed extension shield was retained, while the software stack was transitioned to Zephyr, supplemented by Arduino and MATLAB/Simulink for introductory and model-based development. The hardware features enable a comprehensive set of hands-on lab exercises, from basic GPIO control to real-time data acquisition and networked applications. The paper outlines the structure and progression of the labs, the technical and pedagogical rationale behind tool choices, and the process of customizing Zephyr device tree overlays to support diverse peripherals. PlatformIO was adopted to facilitate seamless switching between frameworks, enhancing the learning experience. The migration demonstrates how Zephyr can serve as a sustainable and industry-relevant foundation for embedded systems education, effectively bridging academic and practical skill development.

Keywords

Zephyr, education, Mbed OS, embedded systems

1. Introduction

The information and electrical engineering course at Wismar University of Applied Sciences has included a basic microcontroller module for several decades. In addition to the internal structures and function blocks, the microcontroller module includes programming in assembler and C at register and bit level. The basic knowledge required to work with this technology is taught.

Another module in the field of embedded systems, long time existing, is real-time and network programming. This module is based on universal and real time operating system, mainly Linux.

During a redesign of the curriculum of the Information and Electrical Engineering degree course and the introduction of Mechatronics, 11 years ago, the need from industry to strengthen training in the field of embedded systems was addressed. The two modules Embedded Control Systems I (ECS) Bachelor and ECS II Master were introduced in these degree courses mentioned above.

At that time, the Arduino Environment [1] was known to both students and teachers and was used in projects by both. One of the authors successfully used it in research projects to set up a measuring and test stand for an air heat pump and to develop a houseboat with electric motors, solar panels and joystick control. The hardware basis at that time was the microcontroller Atmega328, with relatively low memory resources and processor performance (32kB Flash, 2kB RAM, 8Bit CPU with max. 20MHz). The low available resources often became a problem in these projects. A major advantage, however, was the simplicity of use, the commissioning of the Arduino Integrated Development Environment (IDE) and the numerous tutorials, examples, field reports, etc.

[1*] Corresponding author.
[†] These authors contributed equally.
olaf.hagendorf@hs-wismar.de (O. Hagendorf); niklas.huhs@hs-wismar.de (N. Huhs)
+49 3841 753–7176 (O. Hagendorf); +49 3841 753–7220 (N. Huhs)

Learning with Matlab/Simulink [2] plays an essential role in the education of the above-mentioned degree courses from the first semester onwards, from the introduction to numerical mathematics in the basic mathematics course to the C code generation of Simulink models and execution as a real-time system in the control and automation technology modules.

The Embedded Coder Target for Arduino was published as an open source project on the Matlab File Exchange platform ([3] project deleted 2024.07.11, copy [4]). This made it very easy to start generating C code with the Matlab Embedded Encoder Toolboxes and low-cost hardware.

For the design of the ECS modules, a modern and future-proof laboratory environment was to be created that would enable something from simple microcontroller programming through to C code generation. Due to resource constraints, the Arduino environment was not chosen. In control and automation technology, floating point calculations are important and software implementation on an 8-bit platform is memory and computing time intensive. The main argument against the Arduino hardware was the resource constraints. Around the same time of decision making, a new platform was introduced by STM, the Nucleo-64 family, starting with the two microcontrollers STM32F401 and STM32F411 [5]. The form factor and the connections for extensions corresponded to the Arduino UNO, but with a compatible extension of significantly more pins. Both microcontrollers have a 32-bit Cortex-M4 with FPU. Programming with the Arduino IDE was not possible at that time, but STM advertised the two Nucleo boards with mbed compatibility. The mbed library, or with the current naming Mbed OS, turned out to be the ideal environment. Simple C/C++ programming comparable to Arduino but with an extended functionality was possible. Both an online and various conventional offline IDEs, commercial and free, were available. Based on the Embedded Coder Target (for Arduino) [4], the MbedTarget was created [6] [7] [8], a Matlab/Simulink Embedded Coder target for Mbed OS. In order to make student labs more diverse and interesting than would have been possible with the pure Nucleo board, the Mbed Application Shield [9] was implemented in a separate, largely functionally compatible version in a cost-optimized manner. The current version is presented in the next chapter.

Since Mbed OS was first used in teaching, the project has been continuously developed. In 2016, there was a significant increase in commits, maybe as a result of the IOT hype [10], as can be seen in fig. 1. In 2021, the number of commits then decreased significantly to reach 0 commits in 2025 up to the end of May.

The end of the project was announced by ARM on August 30, 2024, with archiving taking place in July 2026 [11].

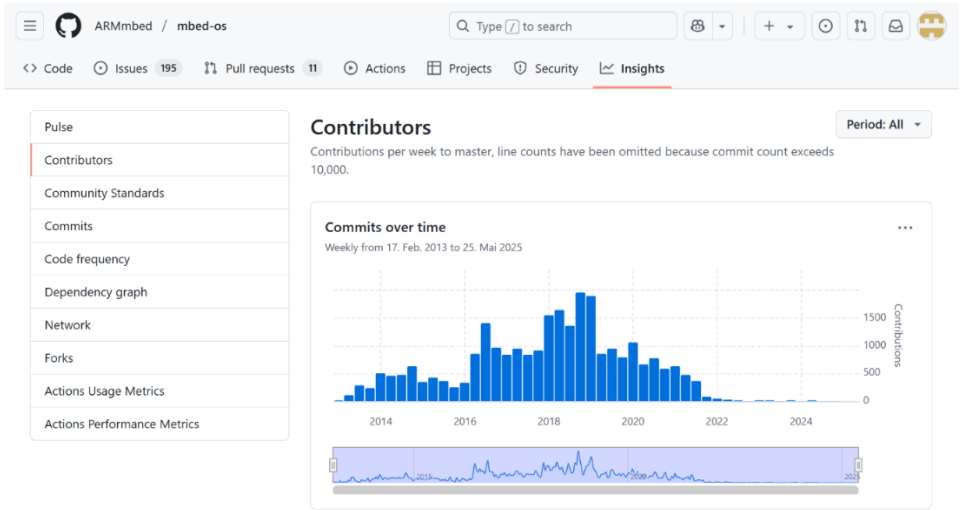

Figure 1: Contributions per week to ARMmbed/mbed-os github repository

Since the winter semester 2023, the curriculum in the Master's in Applied Computer Science and since the winter semester 2024 in the Bachelor's in Applied Computer Science has also been expanded to include the compulsory elective modules Embedded Systems I and II. In this case, the request to introduce these modules came from the student body. The module content was based on the experience of the Information and Electrical Engineering and Mechatronics degree programs.

Due to the discontinuation of the further development of Mbed OS, another redesign is required. Continued use of Mbed OS is only acceptable for a brief moment of redesign; in the medium term, a reorientation is unavoidable and is the current work in progress, described in this paper.

No hardware changes are planned. The use of the Arduino IDE is being reconsidered for programming. A test run in the winter semester 24/25 with students of Applied Computer Science was very promising. However, the library seems too simplified, well suited for introducing students without microcontroller experience, but there is neither real-time, general device driver nor general network functionality. STM32duino open source community is responsible not only for the STM32 Arduino core library but also for extensions for FreeRTOS and Ethernet [12]. But it is limited to the microcontrollers of the STM32 families. Broader hardware support would be desirable.

The development of Zephyr has been observed by the authors for several years. Many features of the project seem to meet the requirements very well. A master thesis examined various open source implementations of (D)TLS libraries in different environments: MbedOS, Zephyr and Micropython [13] [14]. Regardless of the results concerning the security libraries, the support of (D)TLS environments with the investigated protocols (http, mqtt, coap) was most extensive in Zephyr.

For these reasons, the decision was made to use Zephyr in the various embedded modules.

2. Hardware

The hardware used in our laboratories is divided into two parts: the commercial available microcontroller platform Nucleo-144 STM32F767ZI from STM [15] with an expansion shield developed in-house [16]. The Nucleo-144 board uses a 32-bit Cortex®-M7 CPU with DP-FPU, a maximum clock frequency of 216 MHz, an Ethernet interface, 2 MBytes of flash memory and 512 KB of SRAM. Further features can be found in the data sheet [15].

The extension shield is based on the Mbed Application Shield from ARM [9], which was originally slightly modified to optimize costs and later developed further with functional enhancements. The current version has been adapted from Nucleo-64 for the Nucleo-144 format. This means that all labs can be implemented with the same hardware. Before Nucleo-64 together with the Application shield and Nucleo-144 with its ethernet port were used.

The current Application shield version offers the following hardware components, as shown in the schematic in fig. 2:

- Buses: UART, I2C and SPI
- 5 digital inputs for 5-way navigation switch
- 2 analog inputs for potentiometers
- 3 digital outputs for an RGB LED
- 1 digital output for a serial NeoPixel LED [17]

- PWM output for a buzzer
- TFT display 128x32 pixels b/w on SPI bus
- SD card on the SPI bus
- LM75 temperature sensor on the I2C bus
- QWIIC and Grove (I2C) connectors
- UART connector for an SDS011 particle sensor

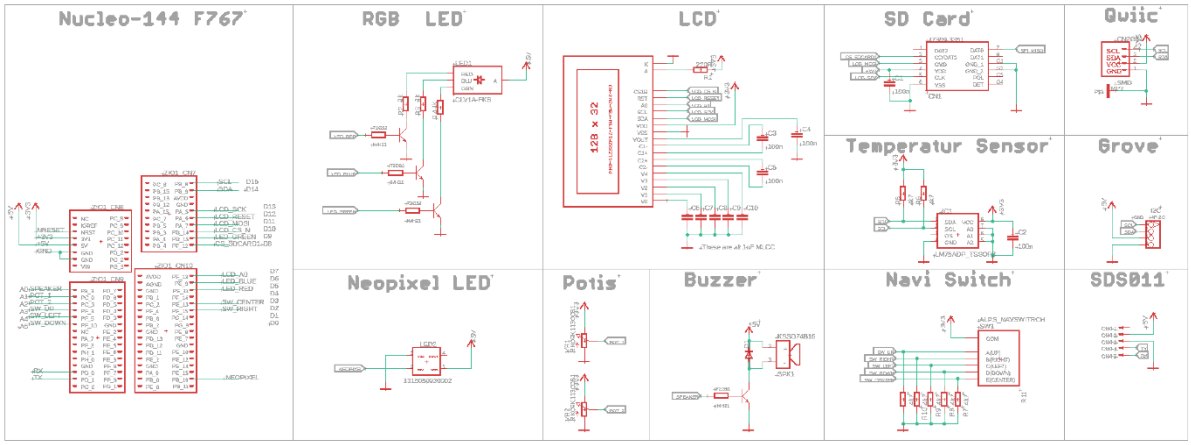

Figure 2: Application Shield schematic v9 2025.01.11

The hardware offers the possibility to implement a comprehensive set of laboratory tasks, starting with a flashing LED, a button input, the implementation of various sensors, display and sensor control via SPI and I2C buses through to the implementation of Ethernet communication for sending data to an IOT cloud application [18] using various unsecured and secured protocols.

The Nucleo-144 board is officially supported by STM for Zephyr. But the device tree only defines rudimentary hardware blocks, the 16 standard digital Arduino-Uno pins, one SPI and I2C interface, one ADC and DAC pin. The pin definitions fall far short of the definitions in the Mbed OS and Arduino environments.

The device tree for these missing functions and everything for the expansion shield will be implemented by the authors and students as part of the preparations for all the available modules.

Some of this work is presented in the following chapters.

3. Software

This goal of the software written for these labs was to optimize the learning curve for the students, who might bring some experience in embedded systems as hobbyists and those might have not had any contact with embedded system development. This meant, that the introduction into the frameworks, in this case Arduino, Zephyr and model based development in Matlab Simulink should quickly introduce all the necessary basics and then move on to more advanced topics. The spot in the labs, that was before taken my Mbed OS, was now occupied by Zephyr.

3.1. PlatformIO

PlatformIO is a cross-platform and cross-architecture tool for multiple frameworks [19]. It is distributed as an extension for the VSCode editor which was installed by over 5 million users around the world. As the labs progress through different embedded frameworks, the multi-framework functionality of PlatformIO offers significant advantages by being able to quickly switch between the frameworks and by handling the installation of those frameworks in the background. The version of PlatformIO used in this work is 3.3.4 which uses the Zephyr version 4.1 for the STM32 platform.

3.2. Hardware test program

To validate the capabilities of the updated application shield and demonstrate the integration of the different peripherals within a Zephyr environment, a comprehensive hardware test program was developed. This program systematically exercises all major components of the shield, including analog-to-digital converters (ADC), PWM-controlled RGB LEDs, a Neopixel LED, a navigation switch and an LCD display. The core architecture of the program is built around a user-navigated menu, implemented through a global menuIndex variable and controlled using GPIO interrupts triggered by the directional navigation switch. Each menu item corresponds to a specific hardware feature and is presented on the LCD display using the Zephyr character frame buffer API. To enable the use of the C12832 Display, a driver was written for the ST7565 controller.

In the ADC test mode (case 0 in the main switch statement), the application iterates through the ADC channels and reads their values using the adc_read_dt() function. These values are converted to millivolts and displayed on the LCD. The RGB-LED is used with three PWM-channels and shows fading effects for all the colors. The Neopixel is driven by a manually implemented bit-banging function. Other menu items include the buzzer, the LM75 temperature sensor and an external connected SGP40 air quality sensor.

This program not only serves a hardware testing purpose but also served as a valuable educational resource. It provided students with a cohesive and practical example of how to structure embedded applications using Zephyr's APIs, device tree bindings, interrupt management, and peripheral interaction. The complete source code for this test and all lab exercises is available on GitHub for reference and reuse.

3.3. Structure of the labs

The labs for the students are structured as shown in table 1. As a general introduction to embedded software development, the Arduino framework is chosen because of its relatively easy learning curve and compact code. The first lab in each framework was meant to teach the basic concepts like GPIOs, Interrupts or, in Zephyr's case the device tree. After teaching the students about the library management and how to use external devices over I²C and SPI, they were able to complete a small project, consisting of a simulated bang-bang temperature controller using the LM75 sensor, the RGB-LED and the display with the navigation switch to adjust switching values of the controller.

After being introduced to the basic concepts of Zephyr, the students added nodes to a basic implementation of the device tree for this hardware, while learning about pinmaps, drivers and how to use them in a device tree overlay. This way the use of the potentiometers was implemented the complete way from creating the device tree, based on the schematic, to writing the c-code to read the ADC values and displaying them on the LCD-display. In the final Zephyr-lab, a multi-thread example was introduced.

The overall last lab introduces model-based development in Matlab Simulink.

Table 1
List of embedded lab exercises

Lab index	Framework	Goal	Contents
1	Arduino	Introduction to Arduino	GPIOs, Interrupts
2	Arduino	Advanced Arduino	Library management and external sensors
3	Arduino	Arduino project	Simulated bang-bang controller
4	Zephyr	Introduction to Zephyr	Device tree, Buttons, LEDs
5	Zephyr	Zephyr	Analog readings, Display
6	Zephyr	Zephyr RTOS	Multiple threads
7	Simulink	Model based development	SDS011 reading

4. Learnings and difficulties

The initial implementation of Zephyr into this embedded system lab was heavily influenced by earlier courses that included Mbed OS in its structure and general approach to teaching. Adaptions were made to highlight the differences of the operating systems as some students were familiar with Arduino and Mbed OS.

4.1. How to approach Zephyr as an educator

The main focus of this course was to provide an understanding of the most basic concepts of Zephyr like the device tree, drivers and configurations. Like in the labs, based on the Arduino framework, the goal was to enable the students to approach a new project, based on simple specifications, and fulfil the set goals, even if the hardware is new to them. The extensive documentation and examples, provided by the Zephyr project were a good basis for the students' independent work. While the contributors of this work were familiar with concepts like the device tree because of a Yocto-based project, the students however had some difficulties in developing an intuitive understanding of the device tree. To better convey the concept, it was useful to not only teach the students to write the code for their program but also let them write a part of the device tree overlay for the hardware used in the labs, starting with a custom led and continuing with configuring two ADC channels as shown in figure 3.

```
&adc1 {
    #address-cells = <1>;
    #size-cells = <0>;
    status = "okay";
    pinctrl-0 = <&adc1_in10_pc0 &adc1_in13_pc3>;
    channel@a {
        reg = <10>;
        zephyr,gain = "ADC_GAIN_1";
        zephyr,reference = "ADC_REF_INTERNAL";
        zephyr,acquisition-time = <ADC_ACQ_TIME_DEFAULT>;
        zephyr,resolution = <12>;
    };
    channel@d {
        reg = <13>;
        zephyr,gain = "ADC_GAIN_1";
        zephyr,reference = "ADC_REF_INTERNAL";
        zephyr,acquisition-time = <ADC_ACQ_TIME_DEFAULT>;
        zephyr,resolution = <12>;
    };
};
```

Figure 3: Initial dtsoverlay for device tree familiarization.

Parallel to writing the necessary nodes for the device tree, the students learned how to activate specific drivers in the the proj.conf, in this case the ADC and STM-specific ADC drivers as shown in figure 4.

```
1   CONFIG_ADC=y
2   CONFIG_ADC_STM32=y
3   CONFIG_ADC_INIT_PRIORITY=90
```

Figure 4: proj.conf for first Zephyr lab.

After setting up these preparatory tasks, the students were able to start programming the c-code. The most important steps, that are repeated in all of the labs are the referencing of the device tree nodes via the aliases, the initialization of the different components and then the main functionality of the program.

The use and programming of drivers for Zephyr was also an essential part of the labs and the advantages of the way Zephyr handles device drivers were easily conveyed. As there was no device driver for the C12832 Display with an ST7565 controller, a driver was written to support this hardware which was an excellent opportunity to learning more about the Zephyr's device driver API.

4.2. What to improve for the next course

One of the most challenging aspects of teaching zephyr in a group was the high potential for mistypes due to the initial amount of steps to be taken to create a functioning zephyr program. To streamline some of the more repetitive code. Functions were written to handle things like GPIO initialization and error handling as seen in figure 5.

```c
int gpio_setup(const struct gpio_dt_spec *spec, uint32_t flags){
    int ret = 0;
    if (!gpio_is_ready_dt(spec)){
        printk("Error: button device %s is not ready\n", spec->port->name);
        return 0;
    }
    ret = gpio_pin_configure_dt(spec, flags);
    if (ret != 0){
        printk("Error %d: failed to configure %s pin %d\n", ret, navswup.port->name, navswup.pin);
        return ret;
    }
    return 0;
}
```

Figure 5: GPIO setup function for code reduction.

These functions could be provided to the students to create another layer of abstraction and simplify some of the code, therefore eliminating some sources of error that might hinder the progress of the labs.

Furthermore, it might be advantageous to dedicate a whole ninety-minute lab to the device tree and debugging errors in the device tree as the error messages are not as easy to debug for new users.

5. Conclusion

Migrating the embedded systems lab to Zephyr turned out to be the right choice considering the virtual end of Mbed OS. Students engaged with key concepts such as the device tree, driver configuration and multi-threaded programming. While the learning curve proved steep, especially in understanding device tree syntax and configuration, the hands-on approach, supported by practical excercises and a custom driver significantly improved their confidence and competence in embedded development. The fact that a good part of the participating students chose a Zephyr-based project for their final project shows their confidence. Future iterations of the course will benefit from improved abstraction layers and more focused sessions on Zephyr's configuration and debugging, ensuring a smoother and more effective learning experience.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] *Arduino – Home*, https://www.arduino.cc/, (access 2025.05.31)

[2] *Mathworks*, https://de.mathworks.com/, (access 2025.05.31)

[3] *Mathworks File Exchange*, https://de.mathworks.com/matlabcentral/, (access 2025.05.31)

[4] *Embedded Coder Target for Arduino*, https://github.com/cea-wismar/EmbeddedCoderTarget4Arduino, (access 2025.05.31)

[5] *STM32 Nucleo Boards*, https://www.st.com/en/evaluation-tools/stm32-nucleo-boards/products.html?querycriteria=productId=LN1847$$1574=Nucleo-64, (access 2025.05.31)

[6] *MbedTarget*, https://github.com/cea-wismar/mbed_target, (access 2025.05.31)

[7] O. Hagendorf: *MbedTarget - A Simulink Target for high level Embedded Programming for Cyber Physical Systems*, 2nd IFAC Conference on Cyber-Physical & Human Systems, December 14-15, 2018, Miami USA

[8] O. Hagendorf: *MbedTarget - A Simulink Target for Cortex-M Microcontrollers.* ASIM Fachtagung STS und GMMS 8./9.03.2018, Hochschule Heilbronn, Germany

[9] *Mbed Application Shield*, https://github.com/ARMmbed/mbed-HDK/tree/master/Production%20Design%20Projects/ARM-mbed/ApplicationShield, (access 2025.05.31)

[10] *Gartner's 2016 Hype Cycle for Emerging Technologies Identifies Three Key Trends That Organizations Must Track to Gain Competitive Advantage*, https://www.gartner.com/en/newsroom/press-releases/2016-08-16-gartners-2016-hype-cycle-for-emerging-technologies-identifies-three-key-trends-that-organizations-must-track-to-gain-competitive-advantage, (access 2025.05.31)

[11] W. Lord, *Important Update on Mbed.* https://os.mbed.com/blog/entry/Important-Update-on-Mbed/, (access 2025.05.31)

[12] *STM32duino open source community*, https://github.com/stm32duino, (access 2025.05.31)

[13] P. Bomball, Master-Thesis: *Analysis and prototypical implementation of cryptographic protocols on embedded systems* (German: Analyse und prototypische Implementierung kryptografischer Protokolle auf eingebetteten Systemen), 03.2025, Hochschule Wismar, Germany

[14] O.Hagendorf, P.Bomball, *Survey of Open Source (D)TLS Libraries for Bare Metal Microcontroller Programming*, The 11th IEEE World Forum on Internet of Things (WF-IoT),2025, Chengdu/China, (submitted 2025.03.31, not yet accepted)

[15] *NUCLEO-F767ZI - STM32 Nucleo-144 development board with STM32F767ZI MCU*, https://www.st.com/en/evaluation-tools/nucleo-f767zi.html, (access 2025.05.31)

[16] *Application Shield*, https://github.com/cea-wismar/EmSysApplicationShield, (access 2025.05.31)

[17] P. Burgess, *The Magic of NeoPixels.* https://learn.adafruit.com/adafruit-neopixel-uberguide, (access 2025.05.31)

[18] *Thingsboard – Open Source IoT Platform*, https://thingsboard.io/ , (access 2025.05.31)

[19] *Your Gateway to Embedded Software Development Excellence – PlatformIO*, https://docs.platformio.org/en/latest/, (access 2025.05.31)