

From Bash to Boards: Integrating Project-Based Learning in a Foundational Embedded Systems Course

Ms. Sunila Akbar, York University

Sunila Akbar is an Assistant Professor (Teaching Stream) in the Department of Electrical Engineering and Computer Science at York University, Toronto, Canada. She received her PhD in Telecommunications from King's College London, UK, and completed her postdoctoral fellowship in the Department of Electrical and Computer Engineering at the University of Toronto, Canada, where she continues as a Visiting Researcher. Her research focuses on wireless communications and adaptive radar resource management using deep reinforcement learning. She is passionate about fostering experiential and inclusive learning in engineering education and advancing evidence-based pedagogical practices. Sunila is a Fellow of the Advance Higher Education, UK, a Senior Member of IEEE, and a Professional Engineer (PEng) in Ontario. Her broader interests include pedagogical innovation, student mentorship, and supporting women in engineering.

Mr. Abdallah Fawzy El-Hamalawy, York University

From Bash to Boards: Integrating Project-Based Learning in an Introductory Embedded Systems Course

Abstract

Embedding experiential learning opportunities early in the curriculum is essential for helping students integrate software and hardware concepts. This paper presents a scaffolded project-based learning (PBL) framework implemented in a second-year introductory embedded systems course that integrates Linux Bash scripting, C programming, and microcontroller-based development. Students progressed from structured laboratories to a multi-stage mini-project emphasizing sensing, timing, data processing, and system-level integration. Instructional impact was evaluated using a voluntary post-course questionnaire administered following ethics approval. Results indicate that approximately 75% of students reported improved understanding of how Bash, C, and microcontroller concepts integrate within a single embedded workflow, 69% reported clearer understanding of the role of Bash scripting in a microcontroller context, and 88% reported improved ability to reason about real-world embedded problems requiring coordinated software–hardware design. Qualitative reflections further suggest that students developed an end-to-end perspective on embedded systems, explicitly linking software decisions with hardware behavior. Collectively, these findings highlight the effectiveness of early, scaffolded PBL in supporting system-level reasoning and cross-domain integration in foundational electrical and computer engineering courses.

Introduction

Introductory embedded systems courses play a critical role in electrical and computer engineering (ECE) curricula by bridging abstract programming concepts with real-world hardware-based system design. However, students in early stages of their programs often struggle to connect software constructs with physical system behavior, particularly when programming, hardware architecture, timing, and system integration are introduced as separate topics. In many curricula, Linux Bash scripting, C programming, and embedded hardware

development are taught in isolation or across different instructional units, which can lead students to perceive these skills as disconnected rather than as complementary components of a complete embedded systems workflow.

Project-based learning (PBL) has been widely adopted in engineering education to address such challenges by situating learning within authentic, applied contexts. When carefully scaffolded, PBL can support deeper conceptual understanding by allowing students to apply multiple skills toward a common system-level goal. However, introducing PBL early in the curriculum requires intentional instructional design to manage cognitive load and to ensure that projects promote integration rather than overwhelming novice learners.

This paper presents a scaffolded PBL framework implemented in a second-year introductory embedded systems course. Course feedback from prior offerings of this course indicated that students often understood Bash scripting, C programming, and microcontroller concepts individually, yet struggled to articulate how these elements interact in practice. As a result, students reported difficulty reasoning about system-level behavior and transitioning from tool-specific tasks to integrated embedded applications. This work explicitly integrates Linux Bash scripting, C programming, and microcontroller-based development within a cohesive mini-project. Rather than treating scripting as a peripheral or introductory skill, the framework positions Bash as a system-level orchestration tool that supports program execution, data handling, and interaction with embedded hardware. Through a structured progression from foundational laboratories to a multi-stage mini-project, students are guided toward end-to-end system reasoning that reflects authentic embedded systems workflows.

Related Work and Research Questions

Previous studies in engineering education have explored project-based and experiential learning approaches, reporting benefits for student engagement, motivation, and conceptual understanding. Early work by Prince [1] established empirical support for active learning strategies, while Mills and Treagust [2] highlighted the suitability of PBL for engineering contexts that emphasize design and implementation. Kolmos et al. [3] further articulated key principles of PBL, emphasizing progressive complexity, alignment with learning objectives, and authentic problem contexts.

Within embedded systems education, prior studies demonstrate that early exposure to hands-on projects can improve students' understanding of software–hardware interaction. Hardin et al. [4] and Sung and Hsu [5] reported positive learning outcomes when embedded projects were introduced early and supported by structured laboratory activities, reinforcing the value of grounding abstract programming and architectural concepts in physical system behavior. However, much of this work has focused primarily on C programming and hardware interfacing.

This leaves open an important pedagogical question: how to help students connect multiple layers of an embedded workflow in foundational courses. Related work in engineering and computing education suggests that linking programming instruction to meaningful application contexts can improve student motivation and perceived relevance [6], while cross-domain learning can support knowledge transfer and system-level reasoning [7]. Recent reviews also confirm the effectiveness of PBL when learning activities reflect authentic professional practice [8].

Yet, despite these broader insights, limited attention has been given in embedded systems education to cohesive instructional approaches that explicitly integrate scripting tools, low-level programming, and embedded hardware within foundational courses. To examine this gap, the present study is guided by the following research questions:

- To what extent does a scaffolded mini-project support students' integrative understanding of Linux Bash scripting, C programming, and microcontroller-based development?
- How does embedding Bash scripting within a hardware-centered project influence students' perceptions of its relevance and application in embedded systems?
- How does early exposure to an integrated, real-world embedded mini-project support students' development of system-level software–hardware reasoning?

Course Context and Pedagogical Model

The proposed work presents a scaffolded PBL framework to promote early system-level reasoning in embedded systems education. Rather than treating Linux Bash scripting, C programming, and microcontroller development as separate topics, the framework integrates these components within a unified instructional design. It is intended to help students connect

scripting, low-level programming, and hardware interaction within a complete embedded systems workflow.

The framework was implemented in EECS 2032 Introduction to Embedded Systems, a required second-year course with a typical enrollment of 100–110 students from Electrical Engineering, Computer Engineering, and Software Engineering programs. Students typically enter the course with basic programming experience but limited prior exposure to embedded platforms or system-level hardware–software integration.

Instruction is organized into three scaffolded phases: foundational programming laboratories, preparatory embedded systems exercises, and a multi-stage applied mini-project. The earlier phases build the prerequisite knowledge and skills needed for the mini-project, which serves as the primary integrative experience. This progression supports system-level learning while maintaining an appropriate level of complexity for early-stage students.

Mini-Project Description and Laboratory Sequence

The mini-project formed the final integrative component of a scaffolded laboratory sequence that developed students' skills in Linux Bash scripting and C programming while introducing foundational embedded systems concepts. Students then applied this preparation in a simplified vehicle speed detection and monitoring system involving peripherals, timing, and sensor interaction. The system measures the time an object takes to travel between two sensors and computes speed using timer-based measurements. A potentiometer sets a configurable speed threshold, while outputs are communicated through LEDs and serial communication with a Linux system. The project reflects core ECE competencies, including sensor interfacing, real-time measurement, peripheral configuration, and hardware–software integration.

Hardware Platform

The mini-project is built around the NXP KL43Z microcontroller board, selected for its rich peripheral set, robust development ecosystem, and suitability for instructional use in early embedded systems education. The platform provides access to key peripherals such as General Purpose Input/Output (GPIO), timers, external interrupts, and analog-to-digital converters (ADCs), enabling students to explore both event-driven and time-driven embedded system behavior.

In addition to the microcontroller board, the project hardware, as shown in Fig. 1, includes a power supply module, two infrared (IR) sensors for vehicle detection, a potentiometer for analog input and speed threshold configuration, and breadboard-based wiring to support rapid prototyping. Real-time measurements are transmitted via serial communication to a Linux system, where Bash scripts handle the data and store the processed measurements in a simple database.

This hardware configuration was intentionally kept consistent across all mini-project laboratories to reduce setup overhead and allow students to focus on system behavior rather than hardware reconfiguration. Maintaining a stable hardware platform throughout the project also reinforced continuity between laboratory exercises and the final integrated system [9].

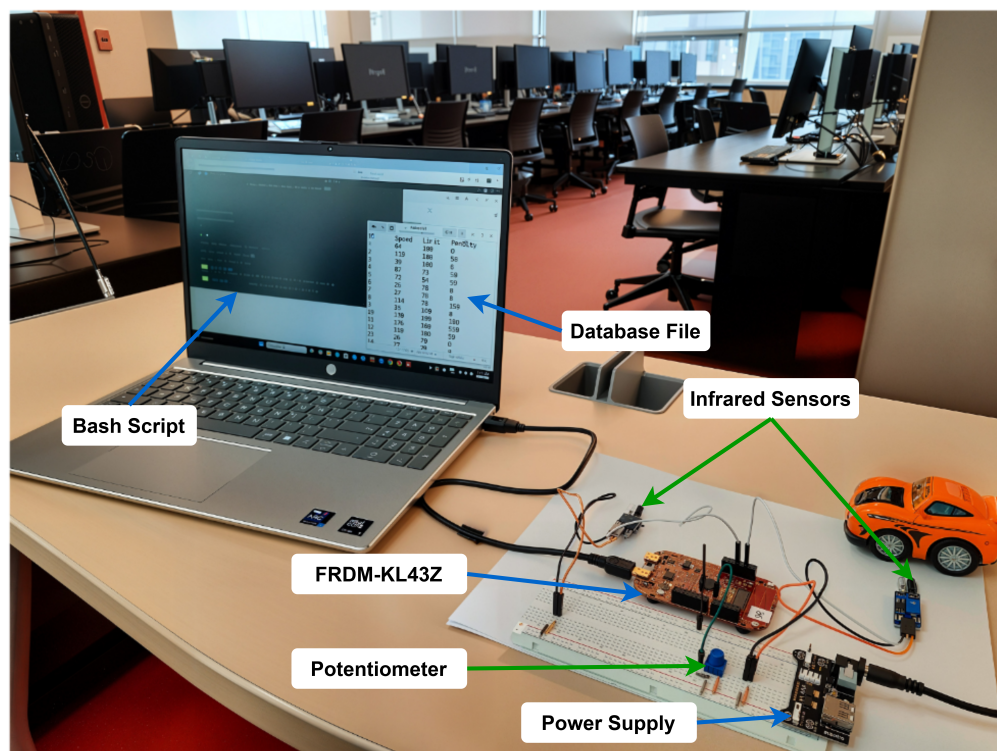


Figure 1. Mini-project hardware platform.

To support student preparedness, a structured hardware tutorial and pre-lab exercises were provided prior to the mini-project sequence. These materials introduced the KL43Z board layout, pin mapping, and basic peripheral configuration, ensuring that students entered the

project laboratories with foundational familiarity rather than encountering the hardware for the first time in an open-ended setting.

Laboratory Sequence

The mini-project was implemented through three tightly coupled laboratory sessions, each preceded by targeted pre-lab exercises. The laboratories were explicitly designed to build incrementally toward the final system, with each stage introducing new embedded concepts while reinforcing skills developed in earlier coursework. Rather than treating the laboratories as isolated activities, each laboratory addressed a specific subsystem that would later be integrated into the complete mini-project.

- **Lab 1: Event Detection Using External Interrupts** — In the first mini-project laboratory, students configured GPIO pins connected to IR sensors and implemented external interrupt service routines (ISRs) on the KL43Z microcontroller. Sensor activation events were detected through interrupt triggering, and visual feedback was provided using onboard LEDs. This laboratory introduced event-driven execution and interrupt configuration within the microcontroller environment.
- **Lab 2: Timing and Analog Thresholds** — The second laboratory introduced hardware timers and ADC. Students measured elapsed time between IR sensor events using timer peripherals and configured the ADC to read potentiometer values that defined adjustable speed thresholds. This lab required coordination among multiple peripherals, including timers, ADC channels, and GPIO inputs.
- **Lab 3: System-Level Integration and Bash Processing** — In the final laboratory, students integrated embedded C code with Linux Bash scripting on a host system. Speed measurements computed on the microcontroller were transmitted via serial communication and processed using Bash scripts to log data and compute penalties based on predefined thresholds. This stage completed the end-to-end system by linking microcontroller execution with host-based scripting and data processing.

By structuring the mini-project laboratories around progressively integrated subsystems and supporting them with targeted pre-lab exercises, the sequence enabled students to develop embedded systems competence incrementally. Each laboratory served both as a standalone

learning activity and as a building block toward the final project, supporting the broader pedagogical goal of early, scaffolded experiential learning.

Results

This section presents the post-course questionnaire and explains how its responses were used to examine the study's research questions. The post-course questionnaire consisted of thirteen questions. Questions Q1–Q4 captured students' prior experience with C programming, Linux Bash scripting, and embedded hardware. Questions Q5–Q10 used five-point Likert-scale items to examine perceived learning gains, integration across tools, and system-level reasoning. Questions Q11–Q13 were open-ended and invited reflective feedback on students' learning experiences, the perceived value of the mini-project, and suggestions for improvement.

The study received approval from York University's Human Participants Review Sub-Committee (Certificate : e2026-017). All data were collected through a voluntary, anonymous post-course questionnaire administered after course completion.

The results are organized around the three research questions introduced earlier, with particular emphasis on Q5, Q6, and Q9, which most directly address integrative understanding, the applied role of Bash scripting, and system-level software–hardware reasoning. Responses to the open-ended questions are incorporated into the relevant subsections to contextualize and help explain the quantitative findings, consistent with common mixed-methods reporting practices. While other Likert-scale items informed the overall analysis, they are not discussed in detail here.

Student Background and Prior Experience

Questions Q1–Q4 captured students' prior exposure to programming, scripting, and embedded hardware. These background data provide essential context for interpreting the perceived learning outcomes associated with the mini-project.

Table I
SUMMARY OF STUDENT BACKGROUND AND PRIOR EXPERIENCE

Aspect	Summary of Responses
C programming	Moderate prior exposure, typically limited to introductory syntax, control structures, and small-scale programming exercises.
Linux Bash scripting	Limited and highly variable exposure; many students reported little experience beyond basic command usage.
Embedded hardware	Minimal prior experience with microcontrollers, sensors, or embedded system development for most students.

As shown in Table I, while many students entered the course with some familiarity with C programming, prior exposure to Bash scripting and embedded hardware was considerably more limited. This heterogeneity is typical of early-stage electrical and computer engineering cohorts and presents a pedagogical challenge when introducing system-level embedded concepts that require concurrent reasoning about software and hardware. These background data underscore the importance of instructional scaffolding when integrating multiple technical domains early in the curriculum.

Integration of Bash, C, and Microcontroller Concepts (Q5)

Question Q5 examined students' perceptions of how the mini-project supported the integration of Linux Bash scripting, C programming, and microcontroller-based development within a single embedded systems workflow. As shown in Fig. 2, responses demonstrate a strong positive trend, with approximately 75% of respondents selecting *Agree* or *Strongly Agree*. No respondents selected *Strongly Disagree*, indicating minimal negative perception of the integrative project design.

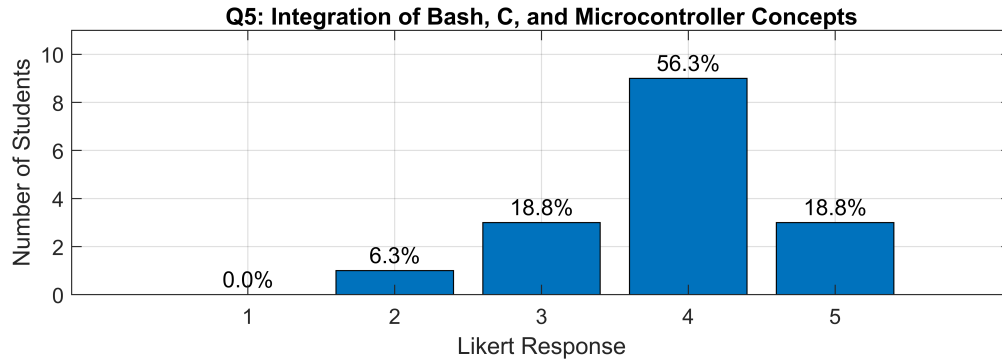


Figure 2. Student responses to Q5: Perceived integration of Bash scripting, C programming, and microcontroller concepts through the mini-project.

These results suggest that the mini-project functioned as a cohesive learning experience rather than a collection of disconnected tasks. Students reported greater clarity in how scripting, low-level programming, and hardware interaction complement one another in embedded systems development. This outcome is particularly notable given the limited prior exposure to Bash scripting and embedded hardware reported by many students in the background questions (Q1–Q4).

Qualitative reflections from the open-ended questions (Q11–Q13) reinforce this integrative effect. One student described the value of applying all course components within a single system context:

“The KL43Z circuit mini-project allowed me to apply all the things I learned in the course in one project and see how Linux, C, and the microcontroller work together seamlessly.”

Other students echoed this sense of conceptual integration:

“Before the project, the topics felt separate, but working on the mini-project helped connect everything.”

“Using Bash alongside C and the microcontroller showed how software and hardware depend on each other.”

A small number of students noted an initial adjustment period when transitioning from

tool-focused laboratories to system-level tasks:

“At first it was challenging to connect everything together, but once the project progressed, the integration started to make sense.”

Taken together, these findings indicate that the scaffolded mini-project effectively supported cross-domain integration, aligning with the core pedagogical intent of the *Bash to Boards* framework.

Application of Bash Scripting in a Microcontroller Context (Q6)

Question Q6 focused on students’ understanding of how Linux Bash scripting can be applied to support microcontroller-based development. As shown in Fig. 3, approximately 69% of students selected *Agree* or *Strongly Agree*, indicating that the mini-project helped many students recognize Bash as a practical component of an embedded systems workflow rather than as an isolated scripting exercise.

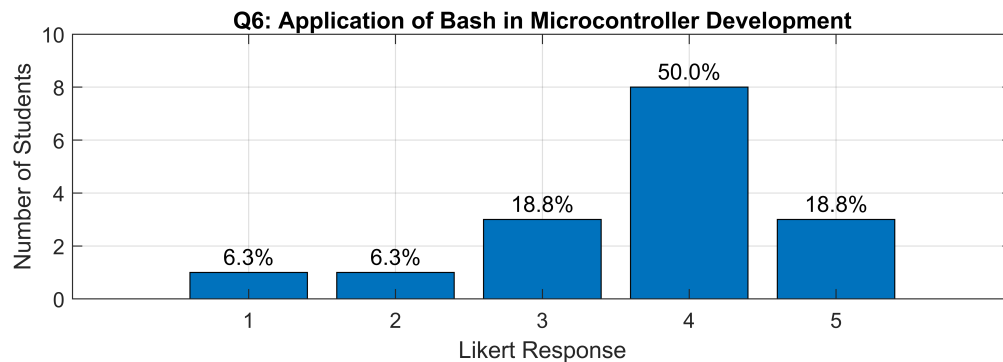


Figure 3. Student responses to Q6: Perceived application of Bash scripting in a microcontroller development context through the mini-project.

Qualitative responses provide insight into how this understanding developed. Several students explicitly described how the project clarified the interaction between real-time embedded execution and host-based scripting:

“By integrating C with Bash, I learned how to hand off data from a real-time environment to a persistent environment. It forced me to treat serial output not just as a debug tool, but as a communication link.”

Others emphasized how Bash enabled system-level reasoning across hardware and software boundaries:

“The connection became clear when I realized that the microcontroller was collecting data while Bash was processing and organizing it. Seeing both work together made the workflow make sense.”

Students also noted that this understanding emerged during the later mini-project laboratories, when Bash was reintroduced as part of system interaction rather than taught in isolation:

“It was during the final lab sessions where we clearly saw the C code and Bash script communicating with the board.”

While most responses were positive, a small number of students reported an initial lack of clarity regarding Bash’s role:

“At first, it was not obvious why we were using Bash again after moving to the board, but once the data started flowing, it made more sense.”

Overall, these findings suggest that embedding Bash scripting within a hardware-centered mini-project supports an end-to-end understanding of embedded workflows, even if initial integration introduces short-term cognitive challenges.

Real-World Embedded Problem Solving and Software–Hardware Reasoning (Q9)

Question Q9 examined students’ perceptions of their ability to reason about real-world embedded problems requiring coordinated software and hardware design. As shown in Fig. 4, approximately 88% of students selected *Agree* or *Strongly Agree*, with no respondents selecting *Strongly Disagree*. This indicates broad agreement that the mini-project supported system-level reasoning within a realistic embedded application context.

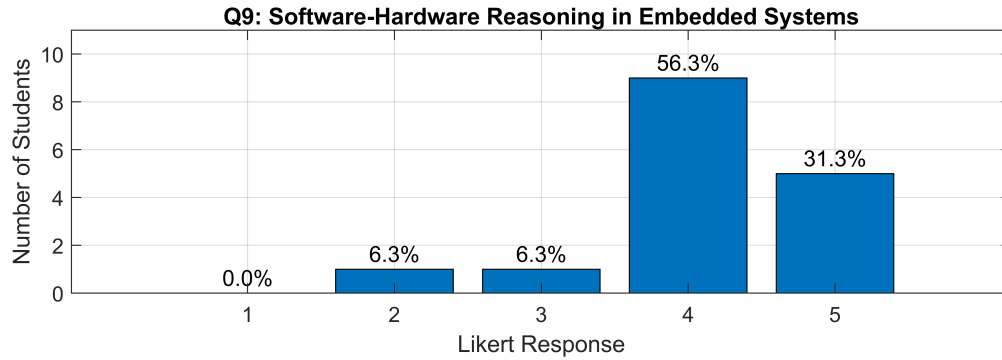


Figure 4. Student responses to Q9: Perceived real-world embedded problem solving and software–hardware reasoning through the mini-project.

Unlike earlier questions that addressed more specific aspects of learning, Q9 captures higher-level reasoning across sensing, timing, decision-making, and communication. The real-world framing of the mini-project, combined with its staged laboratory structure, allowed students to incrementally design, test, and integrate subsystems into a complete embedded solution.

Student reflections highlight this shift toward holistic reasoning:

“The project felt like solving a real embedded system problem rather than just completing a lab.”

Several students explicitly described how the project helped bridge software logic and physical hardware behavior:

“The most significant thing I learned was how to bridge the gap between hardware events and software data processing. It made me realize that what happens on the board doesn’t end there—it directly shapes how the system behaves as a whole.”

Others contrasted the mini-project with earlier component-level exercises:

“With labs that work on one component at a time, we don’t really understand how everything goes together, but here we do.”

Taken together, these findings suggest that the scaffolded mini-project supported early

development of system-level reasoning by helping students understand how software decisions, hardware behavior, and real-world constraints interact within embedded systems—an essential competency in ECE education.

Conclusion

Our study highlights the value of a scaffolded PBL framework for introductory embedded systems education. By connecting Linux Bash scripting, C programming, and microcontroller-based development within a unified instructional sequence, the course enabled students to engage with embedded systems as an end-to-end workflow rather than as a set of isolated technical topics. Student feedback indicates that this approach supported integrative understanding across software and hardware domains, strengthened perceptions of the relevance of Bash scripting within embedded workflows, and fostered the early development of system-level reasoning.

Qualitative reflections further emphasized the value of the mini-project, as students described a stronger appreciation of how programming decisions influence hardware behavior through sensing, timing, and peripheral interaction. While many students found the project meaningful and appropriately challenging, differences in prior experience point to the importance of scaffolding, clear guidance, and structured support in accommodating diverse learning backgrounds.

Future work will extend this study through deeper qualitative analysis of student reflections and examination of longer-term impacts on learning transfer, confidence, and performance in subsequent embedded systems and design-oriented courses. Additional iterations of the framework will explore the integration of advanced sensing modalities, communication interfaces, and real-time constraints, as well as adaptation to other foundational courses in ECE.

References

- [1] M. Prince, “Does active learning work? a review of the research,” *Journal of Engineering Education*, vol. 93, no. 3, pp. 223–231, 2004.
- [2] J. E. Mills and D. F. Treagust, “Engineering education—is problem-based or project-based learning the answer?” *Australasian Journal of Engineering Education*, vol. 3, no. 2, pp. 2–16, 2003.

- [3] A. Kolmos, F. K. Fink, and L. Krogh, *The Aalborg PBL Model: Progress, Diversity and Challenges*. Aalborg, Denmark: Aalborg University Press, 2009.
- [4] J. L. Hardin, J. M. Pearce, and S. E. Watkins, "Project-based learning in an introductory embedded systems course," in *Proceedings of the ASEE Annual Conference & Exposition*, 2018.
- [5] W.-T. Sung and Y.-C. Hsu, "Teaching embedded systems through project-based learning," *IEEE Transactions on Education*, vol. 57, no. 2, pp. 81–88, 2014.
- [6] S. Emara, F. Jazinizadeh, and H. S. Timorabadi, "Connecting the dots: Exploring the benefits of cross-disciplinary learning in first-year programming courses," in *Proceedings of the ASEE Annual Conference*, 2023.
- [7] L. R. Lattuca, D. B. Knight, and I. Bergom, "Developing a measure of interdisciplinary competence," *Journal of Engineering Education*, vol. 106, no. 1, pp. 87–121, 2017.
- [8] S. Lavado-Anguera, P. J. Velasco-Quintana, and M. J. Terrón-López, "Project-based learning as an experiential pedagogical methodology in engineering education: A review," *Education Sciences*, vol. 14, no. 2, 2024.
- [9] NXP Semiconductors, *FRDM-KL43Z Reference Manual*, NXP Semiconductors, 2022. [Online]. Available: <https://www.nxp.com>