Iron Coder – An Integrated Development Environment for Embedded Development in Rust

Abstract

Iron Coder is an integrated development environment for the development of embedded firmware in the Rust programming language, designed with a focus on ease of use. Embedded devices can serve as hands-on platforms that engage those who use them, and thus play an important role in computing education; however, these same devices also pose a significant barrier to entry due to the wide variety of hardware options, complexity of the toolchains and libraries needed to program them, and enigmatic nature of documentation. Platforms such as Arduino, CircuitPython, and Raspberry Pi, among others, have redefined who is able to work with embedded systems by providing developers with an all-in-one hardware-software system that addresses these challenges. Iron Coder provides this approachability for the Rust language by allowing programmers to graphically define the hardware architecture of their system, assist in generating and validating the associated firmware, linking to related library crates, and providing example code and platform-specific tools for development. The tool's design is oriented towards students and hobbyists, whose primary concerns are ease of use, community support, and a rewarding experience that builds intuition. By decreasing the barrier to entry for embedded Rust, our hope is to increase the language's adoption in academia, industry, and hobby use, resulting in the fulfillment of the language's promise as a safe, robust, and performant platform for embedded systems.

With its integration of hardware and software elements, Iron Coder enables the creation of a cohesive, streamlined experience for educators and students in computer engineering fields such as robotics, IoT, and autonomous systems. Beyond this, the platform has potential as a research tool to study the ways in which students learn computer engineering material, especially at the hardware-software interface.

Iron Coder is designed to target all major operating systems as well as WebAssembly, with smooth operation even on low-power devices such as single-board computers. Additionally, the tool is designed to continue to grow via community-driven support – the project is open-source and hosted on GitHub, open to public contributions, and will grow as community members add support for additional hardware platforms. Features and documentation that will allow for further community engagement are underway, and the long-term goal of the project is to become a popular and useful tool among open-source development environments, especially in an educational setting.

Introduction

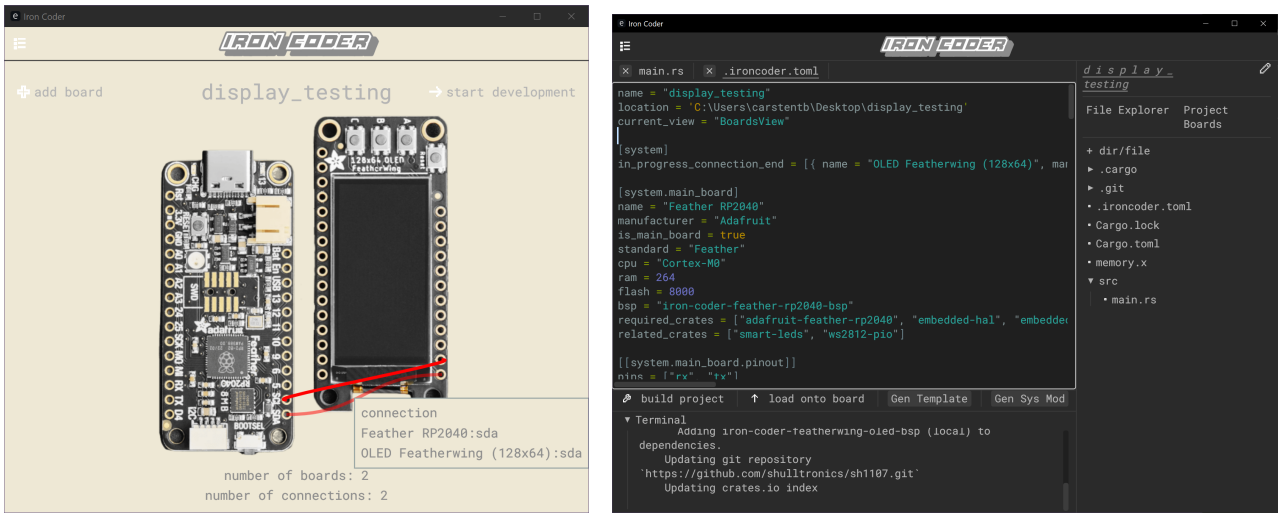
Modern open-source hardware ecosystems such as Adafruit's Feather boards [1], Sparkfun's MicroMod boards [2], and Raspberry Pi's single-board computers [3], among others, are built around modular components, allowing engineers and makers to mix-and-match main boards with a variety of peripheral boards to make systems that can perform many different functions. Programming platforms such as Arduino and CircuitPython have aimed to reduce the barrier to entry for writing firmware for these types of systems by formulating consistent ways of writing code for a multitude of hardware boards, integrating hardware and library management tools into their ecosystems, and allowing for the community-led expansion of supported boards and libraries. The combined evolution of this hardware-software ecosystem over the last two decades has radically shifted the embedded development space from an expensive and intellectually challenging professional endeavor, to one of today's most accessible realms of computing [4]. The rapid success and growth of hobby development boards has had a significant impact on computer science and engineering education, with physical computing used to inspire and excite new developers [5]. The availability of low-cost, high-performance development boards is accelerating the development of novel technologies and finding use in a variety of industries such as the IoT, industrial automation, connected infrastructure and agriculture, and edge-AI [6]. This so-called "fourth industrial revolution" has the potential to significantly impact the future of our world [7].

As a rapidly growing field that is likely to highly influence our lives, the safety, robustness, and maintainability of cyber-physical systems is of primary importance. Rust is positioned to be an important programming language for the next generation of computer technologies, indicated not only by it's increasing adoption by developers[8], but also through its evidence-based claims regarding improved safety and performance over current status-quo technologies [9]. In addition to its performant, safe operation, Rust offers conveniences such as `cargo`, a centralized build system and dependency manager, and `rustup`, a centralized toolchain management utility which allows for the cross-compilation of code for a variety of target architectures. Furthermore, Rust integrates styles and ergonomics from a variety of programming paradigms, contributing to its versatility as a low-level language that also has great zero-cost abstractions and high-level constructs [10].

Iron Coder aims to combine the benefits that Rust offers with the engaging and hands-on nature of embedded development. The embedded Rust space is maturing and seeing growing usage, but remains as a space for those who are already familiar with embedded work or Rust programming. Featuring a stylized, hardware-centric interface, Iron Coder is visually intriguing, with the goal of conceptually connecting hardware and software, which is identified as an important aspect for the future of embedded systems education [11]. By providing a system to generate project templates and hardware-initialization code, Iron Coder helps overcome some of the difficulties that entry-level developers might face when first starting. Iron Coder aims to allow more people, especially novices, to develop embedded firmware in Rust, with important impacts and research potential in computer engineering and embedded systems education.

Design

With the goal of easy and enjoyable use by a diverse audience, the development of Iron Coder focuses on cross-platform support, smooth performance even on low-power devices, robust/crash-free operation, and community extensibility. A core tenet of the project is its open-source status; as such, the project is hosted on GitHub [12] under the GNU General Public License, ensuring that educators can adopt and modify the application to suit their own needs. The UI is continuously scalable for comfortable use on a variety of displays, the app supports multiple color schemes for usage in a variety of lighting conditions, and icons are used throughout the application to visually indicate the actions that UI elements will have.



(a) The hardware editor view in Iron Coder

(b) The code editor view in Iron Coder

Figure 1: The two main modes of Iron Coder, showcasing the user interface design.

Current GUI features include a “system editor” view as seen in Figure 1a, in which boards, represented as 2D images with interactable pins, can be added to the system. Similar to a schematic editor, virtual wires can be used to connect boards together. This mode offers an overview of the hardware layout, along with cursor-hover tooltip information about specific pins and connections, and right-clickable menu options for the boards, which allow for quick access to online documentation and product web pages.

After defining the system architecture, the student can switch to the “development view”, where the code editing occurs (Figure 1b). This view includes a text editor that supports multiple tabs and syntax highlighting, a file explorer that the student can use to browse and create directories and files, and a board viewer, where the current system boards are shown along with their properties, related Rust crates, and example projects as starting points for experimentation. Action buttons are used to compile and load code onto the system (invoking `cargo build` and `cargo run`, respectively), and a built-in status terminal provides feedback to the student in real-time. Adding new hardware support to the editor is enabled through plain text files, and when used in conjunction with a public repository of boards (like Arduino’s library manager [13]) will serve as a foundation for an extensive variety of hardware to be targeted by Iron Coder.

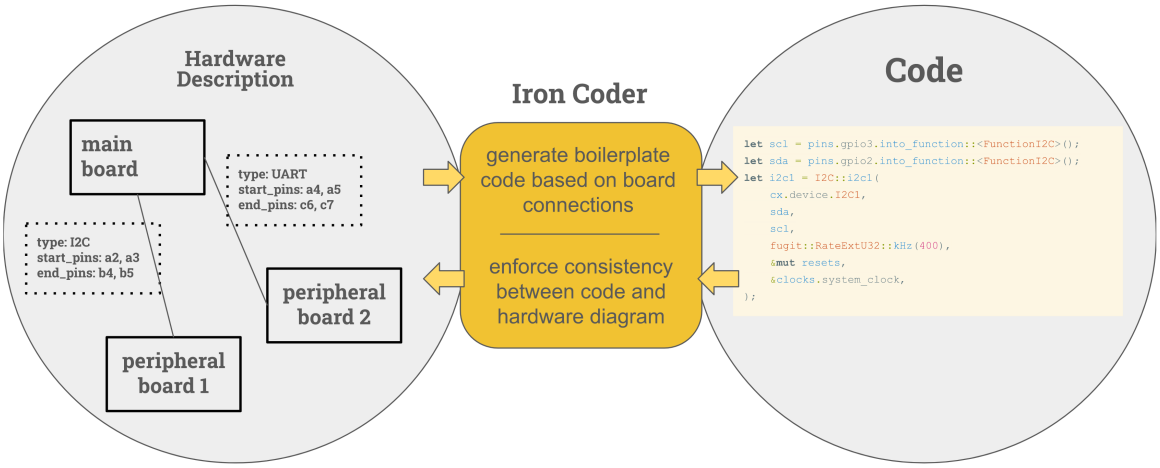


Figure 2: An overview of how Iron Coder synchronizes the hardware diagram with the firmware state.

Beyond acting as a simple text editor, a core feature of Iron Coder is the ability to parse and generate Rust source code as it relates to the system hardware (Figure 2). By utilizing specific board description files, Iron Coder can identify, extract, and insert firmware components into the source code that match the hardware arrangement. The intended goal of this feature is to aid the student in writing hardware initialization code for the current project, as well as ensuring consistency between the hardware description and the source code. Such hybrid programming environments have been shown to have positive impacts on students' experience [14]. A preliminary inclusion of Rust Analyzer demonstrates the feasibility of including language server support [15] to help the developer navigate their code base and quickly identify which lines of firmware relate to particular hardware components.

In-progress and Future Work

While the core application structure is in place, development of Iron Coder is ongoing, and the use of the IDE as a research tool is just beginning. Following the work done by Alomari, et al, we are designing user-testing scenarios to evaluate the impact that this tool could have on computer engineering education [16]. Initial studies will seek to evaluate the usability of the IDE as defined by ISO 9241-11 [17]; other future studies will seek to use the tool as way to measure the effectiveness of physical computing and embedded systems on students' engagement and learning of computer science and engineering topics [5].

Current development efforts are focusing on improving the ease-of-use of the IDE, finding and fixing bugs, developing a website to host cohesive and accessible documentation, and streamlining the process of adding new supported hardware.

As the project matures, we intend to introduce it to the community at large in a variety of ways, including publishing via social media, forums, maker communities on the internet, conferences on engineering education, as well as through outreach with some of the vendors whose hardware and software inspired Iron Coder's conception.

References

[1] L. Fried, "Introducing adafruit feather." https://learn.adafruit.com/adafruit-feather/overview. Accessed: 2024-01-20.

[2] Sparkfun, "What is micromod?." https://www.sparkfun.com/micromod. Accessed: 2024-01-20.

[3] "Raspberry pi." https://www.raspberrypi.com/. Accessed: 2024-01-20.

[4] "The easiest way to program microcontrollers." https://circuitpython.org/. Accessed: 2024-01-20.

[5] S. Hodges, S. Sentance, J. Finney, and T. Ball, "Physical computing: A key element of modern computer science education," *Computer*, vol. 53, pp. 20–30, APR 2020.

[6] "Tensorflow lite for microcontrollers." https://experiments.withgoogle.com/collection/tfliteformicrocontrollers. Accessed: 2024-01-20.

[7] K. Schwab, *The Fourth Industrial Revolution*. Currency, 2017.

[8] "2023 developer survey." https://survey.stackoverflow.co/2023/. Accessed: 2024-01-20.

[9] W. Bugden and A. Alahmar, "Rust: The programming language for safety and performance," 2022.

[10] R. Jung, J.-H. Jourdan, R. Krebbers, and D. Dreyer, "Rustbelt: securing the foundations of the rust programming language," *Proceedings of the ACM on Programming Languages*, vol. 2, pp. 1–34, DEC 2017.

[11] S. Pasricha, "Embedded systems education in the 2020s: Challenges, reflections, and future directions," *Proceedings of the Great Lakes Symposium on VLSI 2022*, vol. Not available, p. Not available, JUN 2022.

[12] "Iron coder repository." https://github.com/shulltronics/iron-coder. Accessed: 2024-01-20.

[13] Arduino, "Installing libraries." https://docs.arduino.cc/software/ide-v1/tutorials/installing-libraries/. Accessed: 2024-01-20.

[14] J. Blanchard, C. Gardner-McCune, and L. Anthony, "Effects of code representation on student perceptions and attitudes toward programming," *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, vol. Not available, p. Not available, OCT 2019.

[15] Microsoft, "Language server protocol." https://microsoft.github.io/language-server-protocol/. Accessed: 2024-01-20.

[16] H. W. Alomari, V. Ramasamy, J. D. Kiper, and G. Potvin, "A user interface (ui) and user experience (ux) evaluation framework for cyberlearning environments in computer science and software engineering education," *Heliyon*, vol. 6, p. e03917, MAY 2020.

[17] I. Standardization and I. O. for Standardization, *ISO 9241-11: Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs): Part 11: Guidance on Usability*. ISO, 1998.