ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-MO2BC006

EMBEDDED CONTROLLER SOFTWARE DEVELOPMENT BEST PRACTICES AT THE NATIONAL IGNITION FACILITY

V. Gopalan, P. K. Singh, V. J. Hernandez, J. Fisher, C. M. Estes, P. Kale, A. Pao, A. Barnes
Lawrence Livermore National Laboratory, Livermore, CA, USA

Abstract

Software development practices such as continuous integration and continuous delivery (CI/CD) are widely adopted by the National Ignition Facility (NIF) which helps to automate the software development, build, test, and deployment processes. However, using CI/CD in an embedded controller project poses several challenges due to the limited computing resources such as processing power, memory capacity and storage availability in such systems. This paper will present how CI/CD best practices were tailored and used to develop and deploy software for one of the NIF Master Oscillator Room (MOR) embedded controllers, which is based on custom designed hardware consisting of a microcontroller and a variety of laser sensors and drivers. The approach included the use of automated testing frameworks, customized build scripts, simulation environments, and an optimized build and deployment pipeline, leading to quicker release cycles, improved quality assurance and quicker defect correction. The paper will also detail the challenges faced during the development and deployment phases and the strategies used to overcome them. The experience gained with this methodology on a pilot project demonstrated that using CI/CD in embedded controller projects can be challenging, yet feasible with the right tools and strategies, and has the potential to be scaled and applied to the vast number of embedded controllers in the NIF control system.

INTRODUCTION

Software engineering best practices such as continuous integration and continuous delivery (CI/CD) [1] have transformed the development, testing and deployment of software applications in many domains. Once primarily used by web and cloud-based applications, these practices are now being increasingly extended to other areas, such as in the context of resource constrained embedded systems. Such systems typically have limited computing capabilities, real-time requirements, and may even lack an operating system. These practices will undoubtably benefit such applications as well, given the increased complexity and faster development cycle demands of such systems, particularly in applications that involve control and monitoring systems that demand high reliability, and real-time responsiveness.

Embedded controllers used in the National Ignition Facility (NIF) [2] control systems are a case in point. An "embedded controller" in this context refers to a custom-built computing device that is designed to perform dedicated functions within the NIF control system. The controller is typically enclosed within the machine or equipment and directly controls all aspects of its operation autonomously without needing to continuously communicate with other parts of the control system. However, they can have control or monitoring interfaces to integrate with the larger control system when required, while still maintaining autonomy.

These controllers are critical to the functioning of the overall NIF control system and typically require high reliability, resilience, and performance. This demands that their development, testing, and deployment procedures be rigorous and extensive. Incorporating software engineering best practices can address many of the challenges associated with these procedures. This will help in automating these procedures to achieve faster development cycles, enhanced quality, and improved system reliability.

However, applying best practices such as CI/CD to embedded controllers presents a unique set of challenges, due to the hardware-software interactions, real-time requirements, safety standards, and the limited computing capabilities. A careful approach is necessary when extending these practices to embedded controllers.

This paper explores the application of software engineering best practices to the development and deployment of embedded controllers used in the NIF control systems, specifically the controllers used in the Master Oscillator Room (MOR). It outlines the potential benefits of this approach, discusses the challenges that need to be addressed, proposes strategies for effectively implementing CI/CD best practices and presents a case study through a pilot project designed to validate this strategy.

EMBEDDED CONTROLLERS IN NIF MASTER OSCILLATOR ROOM (MOR)

The Master Oscillator Room (MOR) [3] at NIF produces laser pulses from an optical fiber laser, originating with a few nanojoules and a beam diameter of a few micrometers. With four distinct oscillators (Fig. 1) corresponding to different beam cones on the target, each oscillator can be set up independently to generate the low-energy laser pulses.

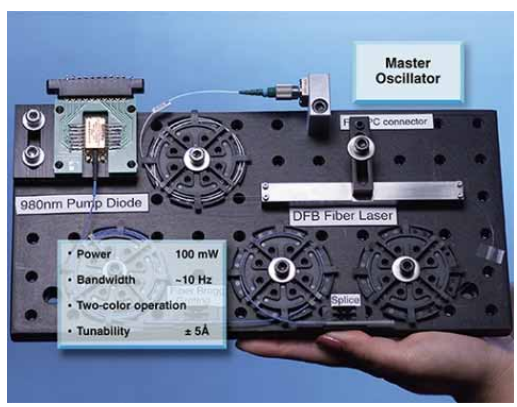

Figure 1: The NIF master oscillator.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-MO2BC006

The equipment in the MOR sets the stage for these pulses to be subsequently amplified several billion times before they are channeled into NIF's main laser amplifiers. These pulses undergo multiple stages of splitting and amplification using industry-standard fiber tools, resulting in 48 distinct pulses each feeding a 'quad' of laser beams, eventually leading to the carefully aligned and timed 192 beamlines feeding focused energy to the target.

The MOR employs over 100 embedded controllers (out of approximately 1000 overall in NIF controls) to achieve its critical operating functions. These controllers consist of optical amplifiers that control output energy and polarization, failsafe switches, phase modulators, RF oscillators and RF switches. These controllers also make use of a variety of hardware technologies for the control computer, peripheral interfaces, and local user interfaces.

Embedded Controller – Generic Architecture

When discussing the challenges faced in embedded controller development, it is helpful to define a "generic" structure of the embedded controller. This serves as a common base architecture that represents the key aspects of all embedded controllers. While embedded controllers can utilize various technologies depending on their function, such as Digital Signal Processor (DSP), Field-programmable gate array (FPGA) etc., this paper focuses on microcontroller-based systems, which are extensively used in the NIF MOR.

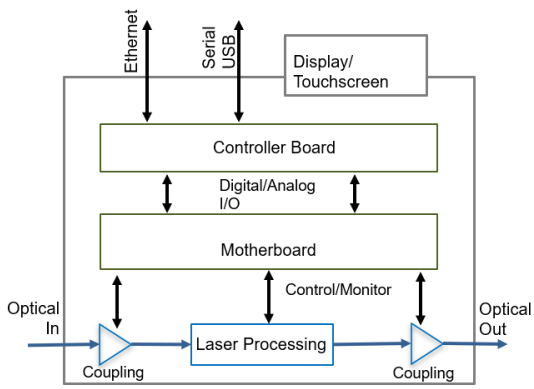

Figure 2: Generic embedded controller.

As shown in Fig. 2, at the core of the embedded controller is a microcontroller-based controller board responsible for managing the primary logic for the control functions. This central unit interfaces with and controls a variety of components: optical elements that channel the laser, a network interface for connectivity to the broader systems, and a serial/USB interface for direct, point-to-point communications to locally connected peripherals.

Complementing the controller board, a dedicated motherboard acts as an intermediary, bridging the controller board to hardware components such as laser diodes. The motherboard incorporates data translation and hardware interfacing and includes features such as signal conditioning, analog-to-digital (ADC) and digital-to-analog (DAC) converters. A display/touchscreen offers a local interface, allowing for local monitoring, adjustments, and diagnostics. Enclosing all these components is a robust enclosure, ensuring protection of the internal components from environmental factors while also protecting the users from internal hazards such as high voltages and laser emissions.

Embedded Controller – Operational Ecosystem

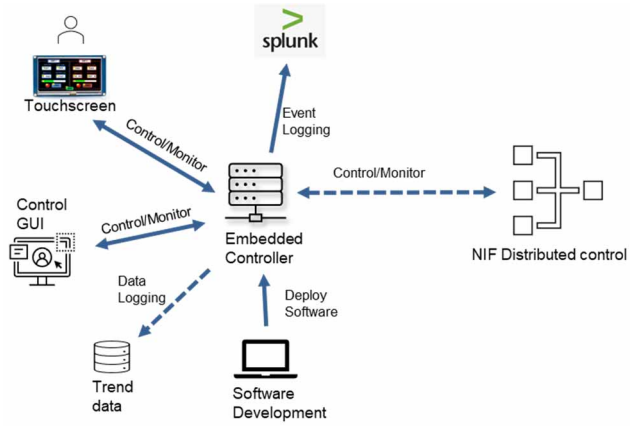

Figure 3: Operational ecosystem.

The operational ecosystem of the embedded controller (Fig. 3) consists of an interconnected framework, designed to ensure full operational control and autonomy. At its core lies the embedded controller device, such as an MOR Dual Fiber Amplifier, functioning as the central hub of the ecosystem. While this embedded controller functions largely independently, it has a peripheral relationship with the NIF distributed control system through an optional network interface for control/monitor interactions between the controller and the larger control system.

The controller's operational environment also consists of a display/touchscreen interface for direct user interactions. A dedicated serial/USB interface supports its programming and software deployment. The controller interfaces with advanced logging systems like Splunk for event logging and deriving performance insights. Since the controller does not typically have file system capabilities, data logging is performed by external applications which read trend data from the controller and write to the network drives. The environment may also contain external control and monitoring GUI applications for enhanced operational interface. This design ensures that the embedded controller can operate without disruption, even when the larger control system or interface to it experiences downtime, ensuring the ecosystem's resilience and autonomy.

EMBEDDED CONTROLLER SOFTWARE ENGINEERING CHALLENGES

Complex Hardware and Software Ecosystem

While various embedded controller solutions were used in the past, designs based on AVR [4] and PIC [5] microcontrollers have emerged as the platform of choice, offering a range of solutions that address the varying complexities and functionalities of MOR embedded controllers.

AVR and PIC microcontrollers, developed by Microchip Technology, are widely used in MOR embedded controller

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-MO2BC006

applications. The AVR microcontrollers were found to be suitable for many embedded controller applications owing to their high performance and low power consumption. On the other hand, PIC microcontrollers offer a good balance of performance and a range of peripheral interface options, proving to be a good fit for various embedded controller applications where robust peripheral support is a priority.

Figure 4: Development tools ecosystem complexity.

These microcontroller platforms, however, necessitate usage of a broad range of programming tools (Fig. 4) [6]. While each tool is associated with its target platform, this tool ecosystem demands that developers navigate multiple learning curves and setup processes.

Developers are also faced with an extensive ecosystem of test and diagnostic tools. This includes hardware diagnostic tools like oscilloscopes to software utilities for simulation and debugging. A fundamental task such as deploying software into the microcontrollers comes with its unique challenges. Different microcontrollers often come with specific flash programmers, which connect to the microcontroller using a variety of interfaces including JTAG, serial, among others. This vast set of tools and interfaces, although essential, can present challenges in compatibility, version control, and developer expertise.

Low Memory and Computational Constraints

Embedded controllers, which are fundamentally based on microcontrollers, inherently contain a low amount of memory, sometimes limited to just a few kilobytes. This constraint creates a significant challenge when designing and developing code in high level programming languages such as C++. Developers must carefully architect their code to be lean and efficient, minimizing computational overhead wherever possible. The generally accepted best practice of prioritizing clarity over conciseness to ensure that the code is maintainable may not apply directly in the context of embedded controllers, where the priority shifts to writing concise code that makes optimal use of the available resources, even if it means sacrificing some level of readability.

Hardware-Software Interactions

Embedded controllers are characterized by their direct hardware-software interactions. This introduces a significant challenge in setting up test environments in general, and automated continuous test systems in particular. The hardware and software in these systems are closely coupled and hence changes in software or hardware can have a direct impact on each other. Testing embedded controller software in isolation from associated hardware can lead to incomplete or incorrect conclusions. As a result, traditional automated test processes (e.g., using simulation techniques), which were designed primarily for software, may not be sufficient for embedded controllers.

Real-Time Requirements

Embedded controllers often operate under real-time constraints, where tasks must be completed within a specified time limit. Correspondingly, the test and validation must be designed to not only ensure functional correctness but also guarantee that the timing specifications are met. This requires specialized testing tools and techniques, which adds complexity to the testing process.

Siloed Development Environments

Different microcontrollers, like PIC and AVR, require their own unique ecosystems, making the creation of a unified development process difficult. This restricts code portability between these platforms and necessitates developer expertise in multiple platforms which will potentially slow down the development cycle.

Configuration Management

Embedded controllers typically store data on the microcontroller's EEPROM (a type of persistent storage), which is quite different from how configuration is stored for other NIF control subsystems (generally in relational databases). This presents a challenge in terms of establishing configuration management practices for such data. The EEPROM lacks the management functionalities found in database systems, thereby making processes related to version control, integrity checks, schema upgrades, and data reverting very challenging.

Rollback Requirements

The necessity for a smooth and swift rollback mechanism is important for uninterrupted operations in a 24x7 facility like NIF. While other NIF subsystems typically implement such processes already, incorporating it in embedded controllers presents a set of unique challenges. This is because the software and configuration data reside in the microcontroller's internal flash memory and is accessed through interfaces like serial ports or other specialized programming interfaces. The use of such interfaces and the associated procedures cost operational time, and the complexity of hardware/software configuration steps to achieve rollbacks may involve manual steps and therefore can increase the potential for errors.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-MO2BC006

BEST PRACTICES STRATEGY FOR EMBEDDED CONTROLLERS

Utilizing Containers to Encapsulate Complex Development Tools Ecosystem

Containers are software packages that encapsulate both software modules and their necessary dependencies, allowing them to execute as isolated entities in any host operating system environment. These containers, hence, can be used to encapsulate all tools, software, compilers, static checkers, and test execution environments specific to each platform (like the PIC or AVR) into self-contained environments. This encapsulation also facilitates the versioning of specific environment configuration, thereby enhancing the reproducibility and consistency of development workflows. Most importantly, once the containers are created, these containers can be made part of a development process flow for automating build, test, and deploy tasks.

A Streamlined Software Development Process based on CI/CD Best Practices

A CI/CD pipeline is an automated software development workflow that facilitates the continuous integration of code changes and the rapid deployment of these changes to the target environments. The containerized toolchains, such as those created for AVR/PIC build environments, can be used to construct the components for a CI/CD pipeline for the embedded controller development. Utilizing DevOps platforms, these containers can then be interconnected to create an automated flow from code integration to deployment.

Despite the potentially differing contents within each container per hardware architecture, they still facilitate standardization of the high-level processes in the CI/CD pipeline. This means that regardless of the underlying platform, the pipeline can maintain a uniform process structure, enabling a common development and deployment cycle for different types of embedded controllers.

It is important to note that in the domain of NIF controls software, the Continuous Delivery (CD) stage does not imply a direct delivery to the production environment. Given the high stakes involved, the CD process is structured to deploy to offline test environments, where comprehensive tests can be executed to validate the system's readiness for production. Once thoroughly validated, the build can then be manually scheduled for production deployment. This helps in maintaining a balanced mix of automation and manual control, leading to a reliable and efficient deployment process for embedded controllers.

Adopting Open-Source Components

Adopting open-source components and tools within the containers instead of using manufacturer-provided libraries (which are typically GUI-based and compatible only with Windows platforms), allows easy integration in Linux based containers and allows the CI/CD tools to benefit from future community-driven improvements.

Coding Practices Tailored for Low Memory, Computationally Constrained Systems

Adopting specialized coding practices becomes essential to overcome the challenges posed by low memory and computational constraints in microcontroller environments. This involves methods such as avoidance of dynamic memory allocation and strategies to minimize function call depth, thus conserving memory and improving code execution speed.

Simulation

Simulation involves creating software models to replicate real-world hardware. In the context of embedded controllers, the functionalities and behaviors of the hardware layer can be replaced with a software simulation to evaluate the system in various scenarios without the necessity for actual hardware components. This allows for continuous testing and validation at different stages of the development cycle while significantly reducing the time and resources required for hardware and environment setup. In addition, the integration of simulators into CI/CD pipelines allows quicker iterations, enabling developers to promptly identify and address defects, ultimately leading to a more reliable and high-quality product.

Hardware-in-the-Loop (HIL) Testing

In addition to simulation, the CI/CD pipeline must include tests on real hardware, to effectively address the challenges associated with automated hardware-software integration and real-time testing in embedded controller environments. This approach, commonly referred to as Hardware-in-the-Loop (HIL) testing, enables developers to validate the system's performance and behavior in real, production-like environments, allowing adjustments and corrections before production deployments.

PILOT PROJECT: MOR DUAL FIBER AMPLIFIER EMBEDDED CONTROLLER

The MOR Dual Fiber Amplifier (DUAL-AMP) is a sophisticated optical device that performs precise control of amplification and output optical energy through closed-loop control of the pump diode current and polarization. Additionally, DUAL-AMP offers a touchscreen interface where operators can control and monitor various parameters including input and output energy for amps, pump diode currents, and the output power for pump diodes. The amplifier supports remote configuration and provides an interface for setting calibration values and limits through an external program via an Ethernet connection.

The DUAL-AMP amplifier's ecosystem (Fig. 5) consists of three distinct software environments. The first is for the AVR microcontroller which forms the core of the embedded controller, the second environment for the touchscreen interface, and the third for remote GUI applications. The deployment process varies across these environments, involving USB/serial updates for controller software, flash memory (SD) card updates for touchscreen software, and network disk hosting for the remote apps.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-MO2BC006

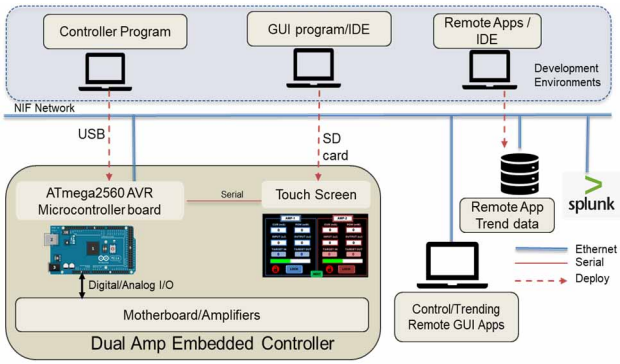

Figure 5: DUAL-AMP ecosystem.

This setup represents the conventional complexity and challenges encountered in embedded controller development. Without the implementation of software engineering principles and CI/CD processes, the development and deployment make use of these multiple siloed environments without a centralized source control, versioning mechanism and automation. This leads to inefficiencies and integration challenges.

The pilot project was structured to incorporate and validate the best practices strategies discussed in the preceding sections, focusing on the development and deployment of DUAL-AMP's software components.

CI/CD Pipeline Construction

Building a robust CI/CD pipeline in the context of microcontroller-based systems necessitates a careful design and integration of various steps, including version control, static analysis, unit tests, integration tests, simulation, HIL tests and deployment.

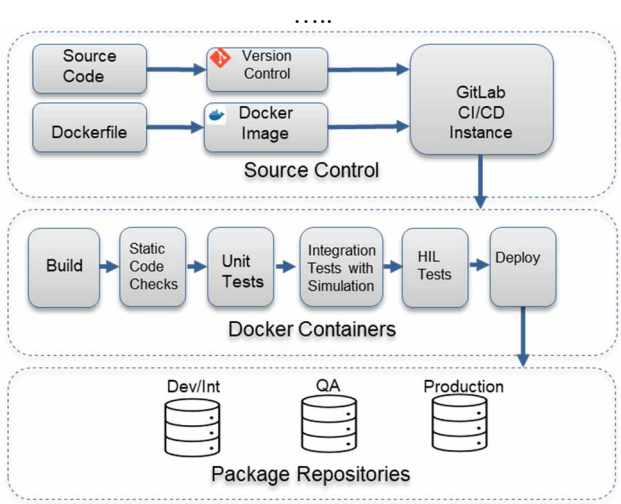

Figure 6: DUAL-AMP CI/CD pipeline.

GitLab [7] was used as the DevOps platform, serving as the orchestrator for the entire CI/CD pipeline (Fig. 6). A key aspect of the project was the creation of Docker [8] containers as part of the CI/CD pipeline, specified using the container configuration (Dockerfile) to encapsulate all the essential tools and environments required for the AVR platform. This container is utilized throughout all phases of the process flow, offering a consistent environment where each pipeline step is executed.

The development process implemented in the CI/CD pipeline starts with source control using Git [9] version control system that ensures traceability and management of the software components and container configuration. Build tools using avr-gcc [10] perform the compile step, followed by static analysis using cppcheck [11] and unit tests using Google Test [12] further enhance the quality by identifying potential issues at the early stages. Integration tests, based on simulations, validate the integrated functioning of the different modules. The integration tests made use of a Python-based tester which interfaced with the embedded controller through a UDP based command/status interface. This interface is used both for actual command and control as well as integration testing. Subsequently, the HIL step rigorously tests the real-time interactions between hardware and software components on an offline platform, and finally the validated software image is deployed to the package repositories.

A modification in the source code or a manual trigger initiates the activation of the CI/CD pipeline. This automated process runs the full build and test sequence without requiring additional effort from the developer, thereby providing near-instant feedback on the changes made.

Enhancing CI/CD Pipeline Efficiency through Code Modularity

Modularity - the process of partitioning the software into distinct, independent *units* – enables parallel development and testing, and results in a clean, organized codebase. Modularity is also essential for robust Continuous Integration/Continuous Delivery (CI/CD) pipelines. The modularized implementation of the DUAL-AMP software (Fig. 7) allows for comprehensive unit testing, where each functional unit can be tested in isolation, thereby promoting early bug detection and targeted debugging.

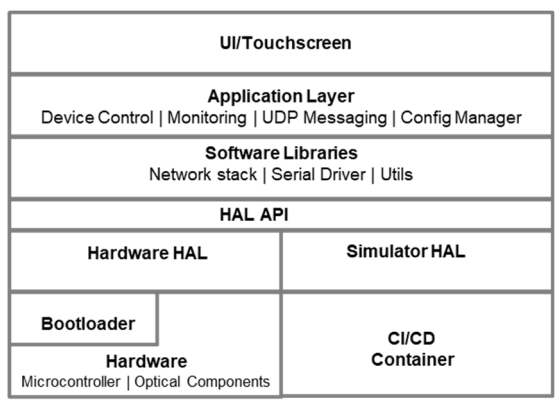

Figure 7: DUAL-AMP software stack.

Another strategy in modular development involves the specification of a Hardware Abstraction Layer (HAL). The HAL serves as a buffer layer between the hardware components of the system and the software, bringing ease of portability and promoting separation of concerns. This means that software components can be developed

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-MO2BC006

independently of the specific hardware details, thus enhancing code reusability across different microcontroller platforms.

Optimized Coding Practices for Resource Constraints

The project adopted coding practices designed for optimized resource usage to address the challenges presented by the AVR microcontroller's limited memory and computational resources. This involved favoring static or stack memory allocation that ensures predictable memory usage and avoiding or reducing dynamic memory allocations that can potentially lead to memory fragmentation. Reducing function call depth, by avoiding deeply nested function calls, prevented stack overflows, and facilitated faster code execution, optimizing the limited computational resources available. In addition, utilizing data types defined in *<stdint.h>* used by C/C++ environments, enabled code portability across different platforms, helping in maintaining consistency in data type sizes.

HIL Testing Setup

The HIL test setup (Fig. 8) consists of a dedicated Deployment and Test Server that interfaces directly with the embedded controller through a serial/USB connection. The GitLab pipeline job for HIL is associated with a GitLab Runner [13] hosted on the deployment server, responsible for executing the HIL job using the Executor.

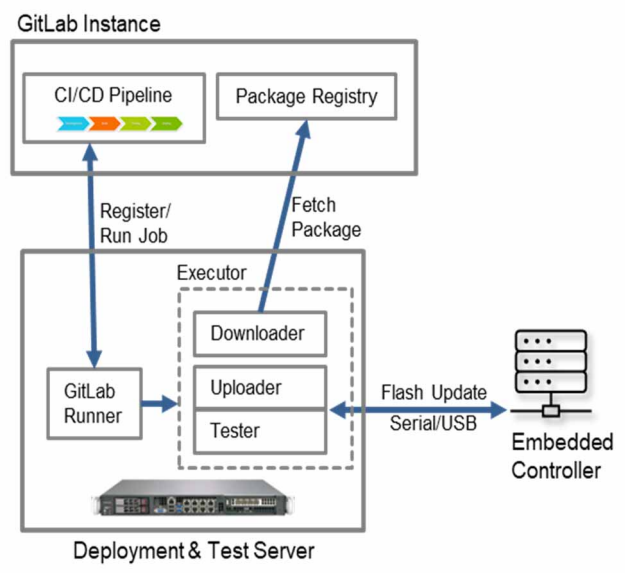

Figure 8: HIL Testing.

As the pipeline runs its sequence of jobs, the HIL job triggers deployment of the necessary software image into the controller via the serial/USB interface, carried out through the downloader/uploader executor scripts associated with the runner. Following this deployment, a series of automated test scripts are run, evaluating the controller's functionalities under various test configurations. The results of these evaluations are then reported back into the GitLab pipeline, allowing the pipeline to report successful completion or failure of the HIL phase within the pipeline.

Complete Build and Release Cycle

After the CI/CD phase, the release undergoes a rigorous quality assurance (QA) evaluation (Fig. 9), where it is subjected to manual testing in an environment that closely resembles the production setup. Initially, configuration data is applied using a config tool, which is subsequently followed by the deployment of the actual software release. The production deployment employs a similar 2-step method, starting with the application of configuration data and then the software release deployment. This configuration and software update sequence also ensures a straightforward revert strategy, where reverting to a previous version involves following the same 2 steps but using a previous good release package.

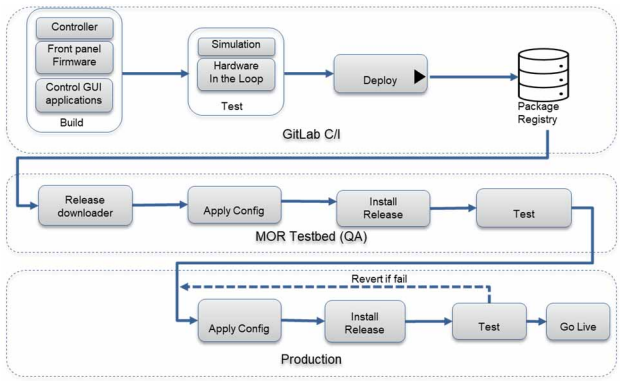

Figure 9: Build and release process.

Config Tool This tool manages updates to the EEPROM map and its contents, facilitating upgrades and providing options to revert changes when necessary. This procedure implements a controlled approach to EEPROM configuration management, ensuring the integrity and consistency of data during software updates and configuration data modifications.

RESULTS AND LESSONS LEARNED

The pilot project successfully implemented a complete CI/CD pipeline for an MOR DUAL-AMP embedded controller using software best practices spanning across the entire development and deployment cycle, resulting in the creation of a well-defined process of designing, testing, and deploying embedded controllers. As the project navigated through this development process, it collected valuable learnings, and this section describes some of the lessons learned during the pilot project.

Modularization-Performance Trade-off

Modularization, although important for maintaining organized and manageable codebases, can potentially incur a cost in terms of runtime speed in systems such as the embedded controllers, due to limited memory and processing power. While the project implemented various techniques to optimize the utilization of limited memory and computational resources, it became evident that additional enhancements and refactoring were required to satisfy the latency requirements of the system. A crucial balance had to

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-MO2BC006

be struck between optimizing runtime performance and the level of modularization by optimizing function call overheads, data transfer overheads, initialization costs, memory fragmentation, and layers of indirection in the code.

Increased Initial Time Investment

Implementing a CI/CD pipeline for embedded controllers demands a considerable initial time investment. This includes extensive scripting and configuring for automation, establishing test environments which replicate the production setup, fine-tuning the pipeline for optimal performance and developing custom solutions for HIL testing. However, this initial increased investment will lead to a more efficient, and responsive development process in the long run.

FUTURE DIRECTIONS

The MOR DUAL-AMP pilot project provided several insights and lessons throughout the planning, implementation, and testing phases. As the project moves towards full productization and wider deployment of these approaches in NIF, plans for future approaches and enhancements were proposed, a few of which are described below.

Integration of Direct Network Update

The deployment of software to the Hardware-in-the-Loop (HIL) test systems from the CI/CD pipeline made use of a dedicated intermediary network connected server. A direct over-the-network update to the embedded controller (without the intermediary server and serial/USB interfaces) will simplify deployment and reduce the infrastructure complexity. The plan, therefore, is to transition to a direct network update mechanism over Ethernet to achieve the above goals, but this would require updated bootloaders in the controller and a different server infrastructure to serve the deployable software images. However, this upgrade might not be feasible for all embedded controllers, particularly those with extremely limited memory and computational capabilities, where implementing network capability may not be realistic. Therefore, in practice, a mixed solution employing both the existing and network update mechanisms is deemed to be the best long-term solution.

Enhancing Hardware-in-the-Loop (HIL) Testing

The current HIL test setup, although quite extensive in coverage, operates within a limited scope in terms of exercising various input stimulations, such as optical feeds, at levels comparable to production environments. To bridge this gap, plans are in place to integrate external hardware configurations capable of generating additional input conditions, thereby creating a more production-like test setup.

Harmonization of Hardware Platforms

Plans are in place for harmonizing embedded controller hardware platforms to enable a more unified and simplified development ecosystem. By focusing on a singular, or more realistically, a reduced set of platforms, the plan is to eliminate the complexities arising from the management of multiple disparate systems. This strategy will help in simplifying development cycles, enhance supply chain efficiency, address hardware obsolescence, and thereby becoming a vital part of the overarching NIF sustainment initiative [14].

CONCLUSION

As embedded controllers continue to perform critical functions in the NIF control system and grow in complexity, the necessity of robust and efficient development methodologies becomes increasingly critical. In this context, the application of software engineering best practices into embedded controllers' development represents a significant step forward, ensuring a more agile, reliable, and effective process for developing, testing, and deploying NIF embedded controllers.

ACKNOWLEDGEMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Document number: LLNL-CONF-854611.

REFERENCES

[1] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison-Wesley, 2010.

[2] M. L. Spaeth *et al.*, "Description of the NIF Laser", *Fusion Sci, Technol.,* vol. 69, 2016. doi:10.13182/FST15-144

[3] P. J. Wisoff *et al.*, "NIF injection laser system", *Proc. SPIE* vol. 5341, of *Lasers Appl. Sci. Eng.*, 2004, San Jose, CA, USA. doi:10.1117/12.538466

[4] Microchip AVR microcontrollers, https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/avr-mcus

[5] Microchip PIC microcontrollers, https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/pic-mcus

[6] Microchip Development Tools and Software, https://www.microchip.com/en-us/tools-resources

[7] GitLab, https://gitlab.com

[8] Docker, https://www.docker.com

[9] Git, https://git-scm.com

[10] avr-gcc – GCC Wiki, https://gcc.gnu.org/wiki/avr-gcc

[11] Cppcheck, https://cppcheck.sourceforge.io

[12] GoogleTest, http://google.github.io/googletest

[13] GitLab Runner, https://docs.gitlab.com/runner/

[14] M. Fedorov *et al.*, "Status of the National Ignition Facility (NIF) Integrated Computer Control and Information Systems", in *Proc. ICALEPCS'21*, Shanghai, China, Oct. 2021, pp. 9-13. doi:10.18429/JACoW-ICALEPCS2021-MOAL02