# $\mu$-HAWK: A Hardware-in-the-Loop Framework for Sensor-Rich Embedded Systems Testing

Anderson R. P. Domingues*, Lucas M. Damo*, Thiago S. Zilberknop*, Luigi S. Bos-Mikich*, Sérgio J. Filho*, Fernando G. Moraes*, Luciano Ost†, Ney Calazans‡

*School of Technology – Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil
†Wolfson School - Loughborough University – Loughborough, UK
‡Computing Department (DEC) – Federal University of Santa Catarina (UFSC), Araranguá, Brazil
{lucas.damo, thiago.zilberknop, l.bos}@edu.pucrs.br, l.ost@lboro.ac.uk, {anderson.domingues, sergio.filho, fernando.moraes}@pucrs.br, ney.calazans@ufsc.br

***Abstract*—The processing power and memory size of typical embedded systems hinder the employment of instrumentation-based software testing techniques, as they clash with nonfunctional requirements such as real-time and energy consumption. In such a scenario, hardware-in-the-loop (HILS) methodologies can achieve software quality guarantees while adding minimal resource overhead to the system under test (SUT). However, HILS strategies lack standardization, requiring testing artifacts to be elaborated from scratch, increasing software time to market and cost. This paper proposes a HILS framework named $\mu$-HAWK to test embedded sensor-rich applications using an FPGA in the loop. Validation of $\mu$-HAWK employs a proportional integral derivative (PID) controller deployed on an STM32F411 microcontroller in an unmanned aerial vehicle (UAV).**

***Index Terms*—HILS, FPGA, Robotics, UAV.**

## I. Introduction and Background

Hardware-in-the-loop simulation (HILS) is a tool for validating industrial systems, proven to reduce the time to market for safety-critical embedded applications, such as those in the automotive, avionic, and robotic domains. Projects in both early and advanced stages of system development can adopt HILS. In the early stages, HILS is used with a reference model to allow engineers to produce an operational prototype of the system faster. Projects in later development stages can use HILS for fault removal, bug detection, and calibration activities. HILS often tackles either system-level or module-level validation. Module-level validation refers to the individual components of a system, while system-level validation considers the system as a whole, mostly exposing only I/O elements to the test. HILS can approach the System Under Test (SUT) from top-down or bottom-up perspectives. The bottom-up approach's initial efforts focus on module-level validation, steadily increasing the number of components in the loop to eventually encompass the whole system. In contrast, a top-down approach is helpful in isolating problematic components.

This work focuses on sensor-rich systems, whose requirements for high I/O throughput are similar to those in robotics and control theory domains. The literature presents studies that use HILS testing infrastructure for sensor-rich applications, as discussed below. With a few exceptions [1], most studies focus on validating control software [2]–[7]. In [2], the authors use HILS on a Lyapunov barrier control function (CLBF) for a mobile robot navigation application. An Arduino board runs the CLBF and interacts with a Simulink robot model. Actuation is performed using a pulse width modulation (PWM) signal. Simulink is also used in [3] to run a reference model that interacts with Nucleo F476RG boards for automotive applications. Similar to [2], these boards mimic the behavior of peripherals attached to the SUT, handling $I^2C$, SPI, and CAN communication. In [4], the authors develop proportional-integral-derivative (PID) and pole placement method (PPM) controllers for an altitude control application validated in both LabVIEW software and in a real UAV. In [5], the authors use FlightSim™ to create a realistic simulation, supported by the RTX operating system, for a missile tracking control application, connected to a real turntable controller. In [6], Arduino boards emulate sensors for a PixHawk board and a reference model in Matlab. Last, in [7], the authors introduce an environment to test a proportional-derivative (PD) control system for an altitude controller. Simulink runs the control algorithm on a host connected to a partial UAV hardware.

The reviewed studies display some key similarities: ($i$) the SUT is isolated from the remainder of the application at the sensor and actuation interfaces; ($ii$) communication between the SUT and the testing environment is typically controlled by a microcontroller; ($iii$) inter-device communication protocols, such as $I^2C$ and SPI, are often employed; ($iv$) a reference model is included; and ($v$) real-time or time-accurate simulation is required. The similarities enable the design of a generic HILS framework. However, accommodating the wide range of sensors and actuators required by embedded applications remains a challenge.

### A. Context & Goals

This work is part of a broader research project investigating fault-tolerance techniques in avionics. A key objective of the project is early identification of misbehaving subsystems. However, due to the heterogeneity of avionic systems, applying HILS across multiple aircraft subsystems is both error-prone and costly, often demanding the development of test components from the ground up. This work addresses these challenges by introducing $\mu$-HAWK, a framework to test sensor-rich embedded systems. $\mu$-HAWK manages system heterogeneity through the use of FPGAs for test coordination, emphasizing scalability through reusable building blocks that considerably lower the effort and time required to integrate SUTs into the HILS environment. The main contributions of this paper are: ($i$) the architecture of the $\mu$-HAWK framework; ($ii$) module templates for implementing the $\mu$-HAWK building blocks on FPGAs; and ($iii$) a case study on a stabilization control application deployed on a real UAV.

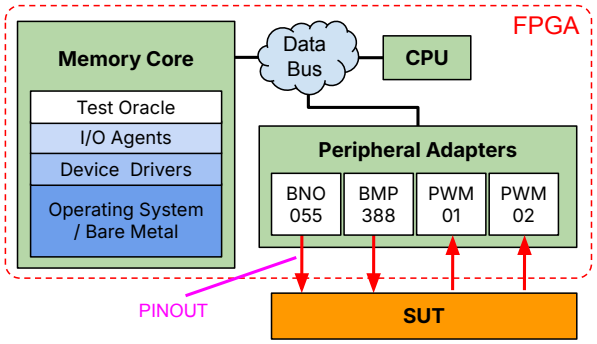


Figure 1. Overview of the $\mu$-HAWK framework, depicting hardware (green) and software components (Test Oracle and I/O Agents). The SUT (orange) is physically connected to the FPGA pins (red arrows). The Test Oracle injects/receives data to/into the SUT through the I/O Agents, comparing received data with a reference model (expected results). The Data Bus (blue cloud) maps Peripheral Adapter registers to the memory, enabling system I/O.

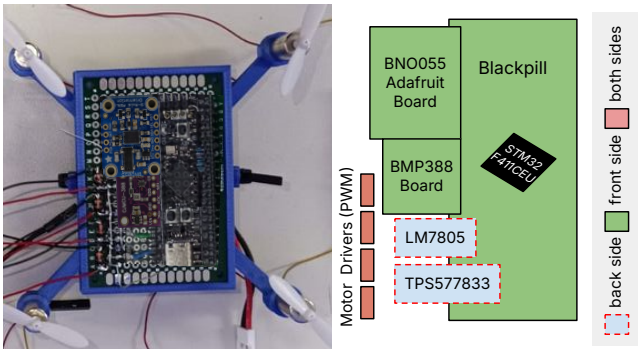


Figure 2. A photograph of the UAV design for swarm applications, with 100g weight and 10cm$^2$ size (left); the flight controller and Blackpill board contents (right). The flight board works as a hub for electronic parts, whereas the Blackpill provides processing power.

## II. The $\mu$-HAWK Framework Architecture

The $\mu$-HAWK framework architecture draws on techniques from Universal Verification Methodology (UVM). It comprises four main components: (*i*) Peripheral Adapters, (*ii*) Data Bus, (*iii*) I/O Agents, and (*iv*) Test Oracle. Figure 1 presents the $\mu$-HAWK architecture. Peripheral Adapters represent simplified hardware models of actuators and sensors. They connect directly to the SUT and emulate real sensors and actuators at the protocol level, enabling the SUT to read data as if it originated from physical devices. In most applications (see Section III), software rarely accesses all functionalities of commercial sensor products; therefore, designers may include only the components required by the SUT, significantly reducing hardware complexity. For example, implementing PWM requires only a few counters to construct a decoder. A single Peripheral Adapter can perform both sensor and actuator functions, depending on the device it emulates. These adapters connect to a Data Bus at the hardware level, allowing software to write/read to/from internal registers. Some of these registers connect to the FPGA pinout in the design, to which the SUT is physically connected. The Data Bus maps Adapter registers to the memory space, allowing the hardware in the FPGA fabric to access the SUT.

I/O Agents are software components that utilize device driver APIs to transmit data between the Test Oracle and the SUT through Peripheral Adapters. They implement a message queue subsystem to filter and format I/O messages, similar to sequencer components in UVM. The Test Oracle can instantiate multiple I/O Agents, enabling tests to compare the values of multiple sensors against one or more reference models, similar to scoreboards in UVM.

The $\mu$-HAWK architecture enables the creation of complex test suites, by providing templates for the Test Oracle, I/O Agents, and Peripheral Adapter components. Templates consist of function prototypes in C language and sequential blocks in SystemVerilog. Users can develop customized HILS environments or directly employ the base platform provided by $\mu$-HAWK, whose components remain open-source and thoroughly validated.

### A. The $\mu$-HAWK Base Platform

The $\mu$-HAWK framework includes a minimal implementation on a base platform. The platform builds on the RS5 SoC [8], [9] and targets the Nexys A7 board, which incorporates a Xilinx Artix-7 FPGA. The required lookup tables (LUTs) and block RAMs (BRAMs) depend on the number and size of Peripheral Adapters as well as on the size of the companion SoC. The base implementation employs the RS5 CPU, the AXI bus, and a dual-port scratchpad memory. Other boards, such as the ZC706 evaluation kit with an AMD Zynq 7000 SoC, feature hard IP CPUs and memory that interface with the FPGA fabric. In such cases, the framework can operate with the on-board SoC; however, the FPGA fabric remains necessary to implement the Peripheral Adapters.

The $\mu$-HAWK framework base implementation includes adapters for the following sensors: (*i*) a BNO055-like sensor via I$^2$C, (*ii*) a BMP388-like sensor via I$^2$C, and (*iii*) a PWM raw decoder. If the target FPGA offers insufficient I/O pins to connect multiple sensors, the design allows multiple Peripheral Adapters to share pins using e. g. I$^2$C splitters. The base platform includes a software package for programming the RS5 SoC bare-metal layer and a port of ZephyrOS [10], which provides device drivers for accessing Peripheral Adapters through AXI. It also contains example applications, together with the Test Oracle and the environment employed in this work, as described in Section III.

## III. A $\mu$-HAWK Case Study: UAV Stability Control

The $\mu$-HAWK framework validation employed a UAV stabilization application. The goal is to provide a reference setup for $\mu$-HAWK users and to evaluate a stabilization algorithm within a flight system for a robotic swarm application.

The UAV features a plastic frame made of printed polylactic acid (PLA) and two boards, as depicted in Figure 2. The first board, a WeAct Blackpill [11] carrying an STM32F411CEU6 microcontroller [12], provides the computing infrastructure. The second board, a custom UAV flight board, connects to the Blackpill. Power is supplied by a single-cell 3.7 V LiPo battery via a TPS577833 (3.3V) regulator. The UAV board also includes an LM7805 (5V) regulator to increase the range of supported electronic components, such as analog sensors not discussed here.

The UAV board has 2 sensors: ($i$) a BNO055 9-DOF IMU [13] and ($ii$) a BMP388 barometric sensor, for altitude tracking. An HC-12 SI4463 low-current transceiver enables remote control. The board also contains four PWM drivers, each consisting of an SI2300 N-channel MOS transistor, a 10k Ω resistor, and a 1N4148 diode. Drivers connect to DC rotors with dimensions (7x16)mm and 0.8 mm axis.

### A. Control Application for Quadrotors Stabilization

The Blackpill board runs a PID controller on top of the UCX/OS real-time kernel [14]. Equation 1 shows the PID controller used to compute the control signal ($u_\alpha$), which drives the PWM rotors to match the desired height and orientation for the UAV (roll $\alpha_{sp}$, pitch $\beta_{sp}$, and yaw $\theta_{sp}$). Parameters $K_p$, $K_i$ and $K_d$ represent the proportional, integral, and derivative gains, respectively. The roll setpoint corresponds to $\alpha_{sp}$; pitch and yaw employ the same controller structure, with $\beta_{sp}$ and $\theta_{sp}$ replacing $\alpha_{sp}$. The parameters $\alpha$, $\beta$, and $\theta$ originate from BNO055 readings. Height measurements rely on both the BNO055 and BMP388 sensors, while a Kalman filter implementation [15], [16] performs sensor fusion and filtering. A PWM library configures an internal timer of the STM32F411 SoC to generate PWM signals on the pins connected to the PWM drivers.

$$u_\alpha = K_p\left(\alpha_{sp} - \alpha\right) + K_i \int_0^t \left(\alpha_{sp} - \alpha\right)dt + K_d \frac{d}{dt}\left(\alpha_{sp} - \alpha\right) \quad (1)$$

In practice, commands emitted by a remote control set the target height and orientation, while the controller determines the adjustments required to maneuver the UAV. See [15] for additional information on PID controllers and on the dynamics of X-type quadrotors.

### B. The Verification Loop

The SUT connects to the FPGA pinout to interact with the $\mu$-HAWK test infrastructure, as Figure 3 illustrates. During the execution of a test case, the Test Oracle ① configures the BNO055 and BMP388 adapters ② with the proper input. Peripheral Adapters connect the SoC design to the SUT via the NexysA7 I/O pins ③, respectively connected to the pins B6-B7 of the Blackpill board ④. The SUT reads from these pins to acquire data from sensors, computing the control algorithm (PID), whose output determines the power of rotors ⑤. The output of the control algorithm is passed to the rotors as a PWM signal. Since the actual rotors were removed, the PWM pins connect directly to the FPGA ⑥. The PWM adapters ⑦ then decode the signals, translating the pulses into integer values between 0 and 100 that represent rotor power (%). Finally, the Test Oracle compares the received values with the expected results in the test case, closing the validation loop.

### C. The Test Oracle

The Test Oracle employs an algorithm that evaluates whether the UAV keeps responding correctly to altitude and attitude variations during stabilization flight mode. In this mode, the UAV must hover while maintaining a constant height. The objective is to verify whether the UAV activates the appropriate rotors to preserve its position.

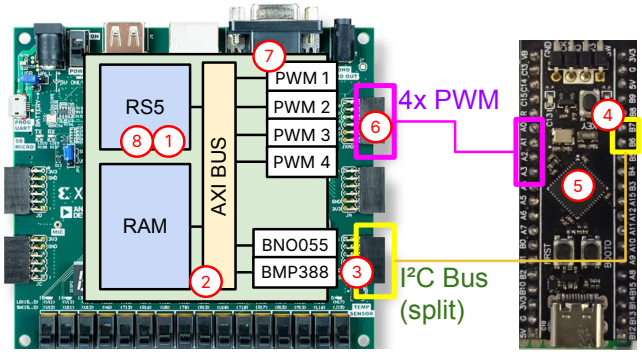


Figure 3. Connection between the Nexy A7 board and the Blackpill board. The SUT generates PWM stimuli to pins A0-A3, and receives sensor data through I$^2$C pins B6 (serial data line, SDA) and B7 (serial clock line, SCL).

Because the proposed setup cannot measure the actual motor speed, the Test Oracle computes changes in rotor power between successive control iterations and determines whether the power increased or decreased. One way of implementing the Test Oracle is as a dedicated hardware component in the SoC, without the need for a CPU. However, implementing the Test Oracle as software eliminates the need for reprogramming the FPGA when exploring multiple test cases.

### D. Implementing Peripheral Adapters

The BNO055 Adapter implements 45 of 80+ registers of the actual sensor specification. These registers provide raw data for acceleration, gravitation, gyroscope measurements, and attitude in both Euler and quaternion representations. The adapter employs an internal state machine that manages I$^2$C communication in slave mode. For a read operation, the SUT writes the target register address to the bus and then resets the transaction in read mode. For a write operation, the master device places the slave device ID on the bus, followed by the register address and the register value.

The BMP388 Adapter acts similarly to the BNO055 Adapter, implementing only some of the registers in the actual sensor specification. As the Blackpill has an internal thermometer, the temperature sensor from BMP388 is not used. The application requires implementing I$^2$C-specific registers (e.g. device ID) and pressure sensor registers (0x4, 0x5, and 0x6 in the BMP388 datasheet).

The PWM Adapters decode PWM pulses using edge detector and divider blocks. At startup, they configure the duty cycle (i.e. the percentage of time in which the PWM signal remains high during each of the PWM periods) to match the one in the SUT.

The PWM Adapter measures the number of FPGA clock cycles during which the PWM signal remains high and divides this value by the total length of the PWM period. This calculation produces an accurate representation of the rotor power, expressed as a percentage, which the Test Oracle can use to validate the control algorithm.

Designers can create new Peripheral Adapters based on existing off-the-shelf sensors and actuators. However, adapters must comply with the interface specifications of the corresponding sensor or actuator. When using the base platform, adapters must also present AXI compatibility.

## IV. Discussion and Conclusion

The $\mu$-HAWK framework addresses the lack of tools for debugging sensor-rich applications in the literature. It employs FPGAs to provide sensor data to the SUT, capture actuation, and compare the measured results with expected values. Unlike previous approaches, which primarily validate control algorithms, $\mu$-HAWK delivers reusable assets and emphasizes peripheral simulation. The framework supports deployment on a variety of FPGA boards and accommodates different SoC architectures. Validation on a real UAV application demonstrated that $\mu$-HAWK operates as an effective and practical testing tool. As faults hidden in inter-device communication protocols cannot be detected by simulation-only environments (e.g., Gazebo-ROS), $\mu$-HAWK provides a new validation layer to contribute to the robustness of mission-critical systems design.

The main benefits of $\mu$-HAWK include: $(i)$ well-defined building blocks; $(ii)$ straightforward replacement of Peripheral Adapters; $(iii)$ support for software-based Test Oracles; $(iv)$ reproducible HILS setups across diverse technologies, including boards, SoCs, and software; and $(v)$ the ability to validate various embedded systems provided the FPGA meets the I/O requirements of the SUT. The framework provides exclusively open-source components, which is another advantage.

A current limitation of $\mu$-HAWK is the number of supplied Peripheral Adapters. Systems may interact with several different types of sensors and actuators, and covering every single possibility takes a long development time. Although $\mu$-HAWK allows engineers to add Peripheral Adapters through templates, they still need to dominate inter-device communication protocols to create new Adapters. $\mu$-HAWK currently provides a base implementation that supplies example Peripheral Adapters for the $I^2C$ protocol and a PWM decoder.

## V. Research Overlook and Further Work

The $\mu$-HAWK framework is operational in its current state. Its assets are published as open-source components, to help practitioners create their own test environments. Assets include source code and documentation of software and hardware components, as well as tutorials. Currently, code assets can be found as a branch[1] of the companion RS5 SoC repository [9].

However, $\mu$-HAWK requires further development to support a broader range of applications. Planned improvements aim to address issues observed during UAV testing in this study, as well as support future applications. The roadmap for enhancing the framework includes, but is not limited to:

- Further research to qualitatively assess the time required to test a system with and without the framework, since one of its primary goals is to reduce the time-to-market of sensor-rich embedded systems.
- Add more Peripheral Adapters to cover a broader range of automotive and avionic applications, supporting protocols used by off-the-shelf sensors such as UART, SPI, and CAN.
- Extend this research by experimenting with Ardupilot and Pixhawk firmware, targeting both open-source and proprietary UAV designs. While many commercial UAV exist on the market, most rely on one of these two firmware platforms.
- Integration of $\mu$-HAWK with ground station software, including support for QGroundControl and MissionPlanner. This effort is driven by a broader research direction: exploring fault prediction in UAVs.
- Develop a test suite for quadrotor UAV applications, incorporating additional sensors such as GPS modules and battery monitors. The goal is to create tests for UAVs out-of-the-box with $\mu$-HAWK, by including a software benchmark in the distribution.
- Verify the possibility to use the $\mu$-HAWK framework for sensor calibration. Examples of this are to be added to the framework soon.
- A console tool to instantiate the framework for different FPGA targets and enable users to switch test cases more efficiently. This is currently under development. Another use of this tool is for it to act as a component wizard to generate hardware targeting AXI interfaces, similarly to the ones available in environments like AMD Vivado.

[1] $\mu$-HAWK source code is available at https://github.com/gaph-pucrs/RS5.


## Acknowledgments

This work is partially financed by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq, grants 305621/2024-6, 311587/2022-4 and 407477/2022-5); and Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS, grant 23/2551-0002200-1).


## References


[1] S. Khaliq, S. Ahsan, and M. D. Nisar, "Multi-Platform Hardware In The Loop (HIL) Simulation for Decentralized Swarm Communication Using ROS and GAZEBO," in *WoWMoM*, 2021, pp. 310– 315.

[2] G. H. Widiarta et al., "Control Lyapunov – Barrier Function Implementation for Mobile Robot Model with Hardware in the Loop," in *MoRSE*, 2021, pp. 1–6.

[3] A. Medgyesi and G. Harja, "Lightweight Hardware in the Loop Setup for Research and Rapid Prototyping," in *AQTR*, 2024, pp. 1–5.

[4] A. Irfan, M. G. Khan, and S. A. Mohsin, "Quadcopter Dynamic Modeling and Stability Control Design using Hardware in Loop," in *RAAICON*, 2021, pp. 56–59.

[5] P. Wang et al., "Design of Hardware-in-the-loop Simulation System based on RTX and FlightSim™," in *CIS/RAM*, 2019, pp. 233–238.

[6] N. Lilansa, Y. Rajib, and J. A. Mawardi, "Interface for Quadcopter Drone Physical Model Hardware in the Loop (HIL)," in *ISMEE*, 2021, pp. 268–271.

[7] M. K. Bayrakceken, M.Kemal et al., "HIL Simulation Setup for Attitude Control of a Quadrotor," in *ICM*, 2011, pp. 354–357.

[8] Nunes et al., "RS5: An Integrated Hardware and Software Ecosystem for RISC- V Embedded Systems," in *LASCAS*, 2024, pp. 1–5.

[9] R. Faccenda et al., "RS5-SoC: A Flexible Open-Source RISC-V Platform for Embedded Systems," in *SBCCI*, 2025, pp. 1–5.

[10] Linux Foundation, "Zephyr Project," Apr. 2025. [Online]. Available: https://www.zephyrproject.org

[11] WeAct Studio, "STM32F4x1 MiniF4," Apr. 2025. [Online]. Available: https://github.com/WeActStudio/WeActStudio.MiniSTM32F4x1

[12] STMicroelectronics, "STM32F411," 2025. [Online]. Available: http://www.st.com/en/microcontrollers-microprocessors/stm32f411.html

[13] Bosch Sensortec, "Smart Sensor BNO055," Apr. 2025. [Online]. Available: https://www.bosch-sensortec.com/products/smart-sensor-systems/bno055

[14] S. Johann, "UCX/OS: A Preemptive Nanokernel for Microcontrollers," 2025. [Online]. Available: https://github.com/sjohann81/ucx-os

[15] P. H. Vancin et al., "Towards an Integrated Software Development Environment for Robotic Applications in MPSoCs with Support for Energy Estimations," in *ISCAS*, 2020, pp. 1–5.

[16] P. Vancin, "Software Framework of Control Systems on an MPSoC Platform," Ph.D. dissertation, PPGCC/PUCRS, 2023.