The Cost of Convenience: How Vendor Lock-In Is Undermining Scientific Research

Scientific research depends on reproducibility; it is the core mechanism by which results are verified, methods are scrutinized, and human knowledge advances. Yet the tools researchers use to gather data, communicate wirelessly, and control instruments are increasingly shaped not by the needs of the scientific community, but by the profit motives of semiconductor corporations. The proliferation of proprietary hardware, opaque firmware, and vendor-specific toolchains has created a landscape where researchers are not customers to be served, but instead consumers to be captured.

This paper argues that the embedded hardware ecosystem, as dominated by companies like STMicroelectronics and Texas Instruments, is actively hostile to the goals of open science. It further argues that the research community's own purchasing habits (reaching reflexively for whatever is considered "industry standard") reinforce these dynamics. Finally, it examines the radio hardware space (specifically LoRa and the DASH7 Alliance Protocol) as case studies in both the problem and the emerging resistance to it.

The "Industry Standard" Trap

When a researcher or lab technician selects a microcontroller, sensor, or radio module, the decision is rarely made from first principles. More often, the selection follows a familiar pattern: search for what other labs are using, check which parts have the most tutorials and forum posts, and pick whatever shows up most frequently in vendor application notes. The result is a self-reinforcing cycle where the most heavily marketed products become the most widely adopted, and widespread adoption is then mistaken for technical superiority.

The phrase "industry standard" deserves scrutiny. In many contexts it does not mean "best tested" or "most rigorously documented." It means "most money spent on marketing." Companies like Texas Instruments and STMicroelectronics invest heavily in developer evangelism, university partnerships, and sample programs. These are not charitable acts; they are customer acquisition pipelines. A student who learns embedded development on a TI LaunchPad or an ST Nucleo board is far more likely to specify those parts in professional designs for the rest of their career.

For the scientific community this creates a specific problem. Researchers are not designing consumer products for mass manufacture. They are building instruments, sensor networks, and data acquisition systems that need to be understood, modified, and reproduced by other

researchers (often years later, often at different institutions). When the tools used to build those systems are tangled up in proprietary SDKs, vendor-specific IDEs, and opaque driver layers, reproducibility suffers directly.

Pearce (2025) specifically identifies this dynamic, arguing that the proprietary model of scientific hardware produces "black box syndrome," vendor lock-in, limited technology transfer, and high economic costs [1]. Bowman (2023), writing in Nature Reviews Methods Primers, makes the complementary case that open-source hardware practices and open licensing are essential for enabling scientific instruments to be reproduced, scrutinized, and properly recorded [2]. Oellermann et al. (2022) further demonstrate that open electronics offer benefits spanning the individual researcher, the institution, and the scientific community, noting that CERN itself built the electronic components of its particle accelerator using open-source hardware specifically to avoid vendor lock-in [3].

STMicroelectronics: Open in Name, Closed in Practice

STMicroelectronics occupies an interesting position in the embedded world. Their STM32 line of microcontrollers is enormously popular, and to their credit, the HAL (Hardware Abstraction Layer) drivers are released under a BSD open-source license. The STM32CubeMX code generation tool is free to download. On the surface, this looks like a company embracing openness.

The reality is more complicated. The STM32 HAL, while open-source, is widely criticized by experienced embedded developers for fundamental design flaws. The locking mechanism (__HAL_LOCK) has been known to be broken for years; it is not interrupt-safe, which means the mutual exclusion it claims to provide simply does not work in practice [4]. ST has acknowledged this bug repeatedly on their community forums, yet the core issue persisted for years across multiple HAL releases. Engineers at companies like Fluke have documented the problem in detail, noting that race conditions in peripheral drivers like UART and I2C are a direct consequence of this broken locking mechanism [5].

The HAL's design philosophy itself is a form of soft lock-in. It abstracts the hardware to the point where developers stop learning what the registers actually do. When something goes wrong (and with the HAL's known bugs, things do go wrong), the developer is left debugging a thick abstraction layer rather than understanding the hardware directly. This is the opposite of what scientific instrument development requires.

The FlightSense Time-of-Flight Sensors: A Case Study in Opacity

An alarming example of ST's approach to vendor lock-in is their FlightSense family of laser time-of-flight sensors. The VL53L5CX and its successor, the VL53L8CX, are 8x8 multizone

ranging sensors that provide distance measurements across 64 independent zones [6][7]. These are genuinely impressive pieces of hardware. They are also nearly impossible to use without ST's proprietary software stack.

The VL53L5CX and VL53L8CX datasheets describe the I2C and SPI electrical interfaces in adequate detail, but they contain no register map [6][7]. This is an extraordinary omission for a sensor IC. Virtually every other I2C or SPI peripheral on the market provides a register map so that developers can write their own drivers and understand exactly how the device operates at the protocol level. ST chose not to do this.

The reason becomes clear when you examine the initialization procedure. These sensors contain an embedded microcontroller (running ST's proprietary STxP70 architecture). During initialization, the host must download approximately 80 to 84 kilobytes of firmware into the sensor over the I2C bus. This firmware is provided as a binary blob in a C header file (vl53l5cx_buffers.h) within ST's "Ultra Lite Driver" (ULD) package. The blob is completely opaque; it cannot be audited, modified, or understood. One researcher who reverse-engineered the firmware identified the instruction set as belonging to ST's secretive STxP70/STHORM processor family, for which no public ISA documentation exists [8].

On ST's own community forums, when a developer asked for register-level documentation of the VL53L5CX, an ST employee responded candidly that the sensor has no traditional registers at all; it is "a full blown MCU" and the register concept is merely used because it fits the I2C protocol. The employee further stated that they had tried to make the API flexible enough that developers would "not be tempted to demand a register map" [9]. Pololu, a well-known robotics supplier that sells breakout boards for these sensors, notes explicitly in their product documentation that ST has not publicly released a register map or any documentation about configuring and controlling the VL53L8CX outside of the provided API [10].

For an earlier sensor in the same family (the VL53L0X), the community frustration was even more vocal. Developers noted that the example code seemed deliberately obfuscated through dense recursive macro redefinition. Another engineer pointed out the absurdity of the situation: the assumption seems to be that engineers are not capable of understanding the complexity, so everything is hidden behind an opaque API. This same engineer noted working with microprocessors that ship with 1800-page reference manuals and 800-page peripheral documents, and questioned how a ranging sensor could possibly be more complex than that [11]. The community frustration grew to the point where an independent, crowd-sourced register map project appeared on GitHub in an attempt to reverse-engineer what ST refused to document [12].

The consequences of this type of obfuscation from the scientific community are severe. A researcher who builds an instrument around a VL53L5CX or VL53L8CX cannot fully characterize the sensor's behavior. They cannot inspect the firmware for bugs. They cannot

modify the ranging algorithms. They cannot verify that the sensor is doing what ST claims it does. When they publish a paper describing their instrument, other researchers cannot independently verify the sensor's operation at a fundamental level. The entire measurement chain passes through a proprietary black box.

Texas Instruments: The Ecosystem Play

Texas Instruments takes a different but equally effective approach to vendor lock-in. Where ST hides details in firmware blobs, TI builds gated ecosystems of tooling and software infrastructure.

The centerpiece of TI's strategy is their SimpleLink platform, a family of wireless microcontrollers (the CC13xx and CC26xx series, among others) paired with the SimpleLink SDK. The SDK provides protocol stacks, drivers, and example applications. The hardware is capable and well-documented at the register level (TI generally does publish reference manuals). However, the software ecosystem strongly encourages (and in some cases effectively requires) the use of TI's own tools.

Code Composer Studio (CCS), TI's IDE, has a history that illustrates the lock-in strategy clearly. Early versions required per-device- family licenses at $3,600 each. Over the years, TI gradually relaxed this; version 7 (released in 2016) finally dropped the paid license requirement entirely [13]. But the relaxation of the license fee was not an act of generosity; it was a recognition that the lock-in had moved upstream. The value was no longer in the IDE license; it was in the ecosystem dependency. Once a project is built around the SimpleLink SDK, TI's RTOS, and TI's proprietary radio stack, switching to another vendor's hardware means rewriting the entire application.

For researchers building sensor networks (a common use case for these wireless MCUs), this ecosystem dependency means that the infrastructure they build is not portable. A sensor network built on TI SimpleLink parts with TI's 15.4 Stack cannot be migrated to another vendor's hardware without a complete software rewrite. The research output becomes entangled with TI's product roadmap.

TI's sub-1 GHz radio solutions are particularly relevant here. The CC13xx family supports proprietary radio protocols in the sub-1 GHz ISM bands. TI markets these as flexible and capable, but the network protocol is TI's own design. There is no interoperability with other vendors' sub-1 GHz solutions. If a lab builds a field sensor network using TI's proprietary protocol, they are locked to TI hardware for every node in that network, permanently.

LoRa: Promise and Compromise

LoRa (Long Range) technology, developed by Semtech, occupies a complicated position in this discussion. It is widely used in scientific sensor networks, environmental monitoring, agricultural research, and similar long-range, low-power applications. It operates in unlicensed ISM bands, which eliminates spectrum licensing costs. The LoRaWAN protocol (the network layer built on top of the LoRa physical layer) has public documentation and is managed by the LoRa Alliance.

However, the LoRa physical layer itself is proprietary to Semtech. The chirp spread spectrum modulation scheme that gives LoRa its characteristic long range and noise immunity is patented and not publicly documented [14]. For years, Semtech was the sole source of LoRa radio silicon, creating a single-vendor dependency for any project that adopted the technology. ST has since licensed the LoRa IP and integrated it into their STM32WL system-on-chip, providing a second source; but the underlying technology remains proprietary.

The research community has pushed back against this in notable ways. Knight and Seeber (2016) presented the first reverse-engineered LoRa PHY implementation at the GNU Radio Conference, demonstrating that the proprietary modulation could be decoded using software-defined radio [15]. Tapparel et al. (2020) built on this work with an open- source, standard-compatible LoRa PHY prototype on GNU Radio, including carrier frequency offset estimation and compensation [16]. Most recently, Busacca et al. (2024) published SDR-LoRa, a full-fledged open-source SDR implementation of a complete LoRa transceiver, demonstrating that their implementation performs comparably to commercial LoRa hardware while enabling research applications (localization, interference cancellation, link reliability enhancement) that are impossible on proprietary silicon [17]. The LoLRa project took a different approach entirely, demonstrating that LoRa packets can be transmitted using general-purpose microcontrollers with no radio hardware at all [18].

The Meshtastic project represents another form of resistance: a fully open-source mesh networking stack built on top of LoRa hardware [19]. While it still depends on Semtech or ST radio silicon at the physical layer, everything above that layer is open, auditable, and community- driven.

These efforts point toward a healthier model, but they also highlight the fundamental tension. LoRa's popularity in research stems from the same "industry standard" dynamic described earlier. Researchers reach for it because other researchers use it, and because Semtech (and now ST) have invested in making it easy to adopt. The proprietary nature of the PHY is accepted as a reasonable tradeoff. Whether that tradeoff is truly reasonable depends on whether the research community values long-term independence over short-term convenience.

DASH7: The Open Alternative Nobody Uses

While the research community debates the acceptable level of proprietary lock-in in LoRa, a fully open alternative has existed for over a decade: the DASH7 Alliance Protocol (D7AP).

DASH7 is an open standard for wireless sensor and actuator networks operating in the sub-1 GHz unlicensed ISM bands (433 MHz, 868 MHz, and 915 MHz). It originated from the ISO/IEC 18000-7 standard for active RFID, which was primarily used in military logistics; in 2009, the U.S. Department of Defense awarded a $429 million contract for DASH7 devices [20]. The DASH7 Alliance, a non-profit consortium, re-purposed the technology in 2011 for commercial wireless sensor network applications and has since released multiple revisions of the specification, most recently version 1.2 [21].

The protocol's design philosophy is fundamentally different from LoRaWAN. Where LoRaWAN is optimized for uplink-heavy, star-topology sensor networks, DASH7 was designed around the concept of BLAST: Bursty, Light, Asynchronous, Stealth, and Transitional communications [20]. This makes it well-suited to the kind of event-driven, bidirectional sensor-actuator networks common in scientific field deployments. DASH7 supports a built-in query protocol that allows the network to be addressed as a distributed database (querying by data content rather than node address), peer-to-peer communication, and multi-hop networking [22].

Critically, the DASH7 Alliance policy explicitly forbids the addition of proprietary or licensable modulation techniques in the official protocol specification [20]. The physical layer uses standard (non-proprietary) modulation schemes that can be implemented on commodity radio transceivers from multiple vendors. This is the exact opposite of LoRa's approach, where the PHY is the proprietary component that locks you to Semtech silicon.

The protocol has a complete open-source reference implementation called OSS-7, licensed under the Apache License 2.0, which was developed with a focus on completeness, correctness, and clarity [23]. Weyn et al. (2013) published a comprehensive survey of the protocol's architecture, covering the OSI layer implementations and introducing the OSS-7 stack [24]. Ergeerts et al. later described practical deployment scenarios and the communication trade-offs available within the protocol [25]. Researchers have also developed SDR-based tools for DASH7, enabling physical-layer analysis using GNU Radio [26].

The layered architecture of DASH7 also allows integration of alternative modulation schemes beneath its network layer. This means DASH7 can run on top of LoRa hardware, providing an open network stack while using LoRa's proprietary PHY for the radio link [20]. Haystack Technologies has demonstrated this hybrid approach, claiming significant performance improvements over LoRaWAN by combining DASH7's network intelligence with LoRa's radio capabilities [27].

Despite all of this, DASH7 has gained minimal traction in the research community compared to LoRa. Ayoub et al. (2019) identified LoRaWAN, DASH7, and NB-IoT as the three main LPWAN standards, yet LoRaWAN dominates adoption [28]. The reasons relate directly to the "industry standard" problem: Semtech has invested heavily in marketing, the LoRa Alliance has over 400 member companies, and the availability of cheap LoRa development boards and gateway infrastructure makes it the path of least resistance. DASH7, despite being technically superior in several dimensions (true bidirectionality, lower latency, query-based addressing, fully open specification), lacks the marketing budget and ecosystem momentum to compete.

This is a case where the scientific community is actively choosing vendor lock-in over an open alternative, not because the open alternative is inadequate, but because it is less convenient. The researchers making this choice are, in most cases, not even aware that the choice exists.

What Open Looks Like

The counterexamples exist. The RISC-V instruction set architecture is fully open and gaining traction in embedded systems. Projects like Zephyr RTOS provide vendor-neutral real-time operating systems that support hardware from multiple manufacturers. The SX1262 and similar LoRa transceivers can be driven by open-source stacks like LoRaMac-node. GNU Radio provides a complete open-source signal processing framework that can replace proprietary radio implementations entirely when paired with appropriate SDR hardware.

In the sensor space, alternatives to ST's opaque FlightSense modules exist. The Garmin LIDAR-Lite series provides ranging with documented interfaces. The TFmini and similar modules from Benewake offer documented serial protocols. These sensors may not match every specification of the VL53L8CX, but they can be fully characterized and independently verified.

The path forward for the scientific community requires a deliberate shift in purchasing and design culture. Selection criteria for research instrumentation should include: availability of complete register-level documentation, absence of required proprietary firmware blobs, support by open-source toolchains (GCC, OpenOCD, etc.), interoperability with vendor-neutral communication protocols, and a credible path to second-sourcing.

This is not a call for researchers to avoid all commercial hardware. It is a call for researchers to recognize that "industry standard" often means "most effectively marketed" and that the convenience of a vendor's integrated ecosystem comes at a real cost to reproducibility, transparency, and scientific independence.

A Week in New Orleans

In December 2025, my team presented research at the American Geophysical Union Fall Meeting in New Orleans. AGU is the largest gathering of Earth and space scientists in the world; over 25,000 attendees from more than 100 countries. Walking the poster halls, I was struck by a concerning pattern: the same technology stack, assembled in roughly the same way, presented over and over again by groups who had each built it independently.

Poster after poster described sensor networks built on the same vendor development boards, running the same vendor SDKs, and collecting data through the same proprietary pipelines. The hardware choices were nearly identical across groups that had never spoken to each other. An ESP32 here, a TI LaunchPad there, an ST Nucleo somewhere else; each wired to the same sensors, each running vendor-generated (or Arduino) initialization code, and each dumping CSV files into a cloud service. A few groups gave in to spending a large amount of their research funds on expensive proprietary equipment. The novelty in each poster was the science (the field site, the measured quantity, the research question) but the instrumentation was treated as disposable scaffolding. Nobody, other than large corporate entities, was building tools. Everyone was building single-purpose data collection rigs, and most of them were building the same one.

What troubled me was not that the systems worked; most of them did, at least well enough to produce meaningful data. What troubled me was that almost none of them were designed to be reproduced. The firmware, if I could get any information at all, was a patchwork of vendor examples and Stack Overflow snippets. The hardware was undocumented beyond a photograph and maybe bill of materials. The data pipelines depended on commercial cloud services with high, and frequently fluctuating, prices. If another group wanted to replicate the measurement, they would have to start from scratch, making the same vendor choices, encountering the same undocumented quirks, and writing the same throwaway code.

There is a deeper problem here than just reproducibility; the scientific community is not investing in research software development as a discipline. We are not building tools for making tools. We are not developing reusable, well-documented frameworks for sensor integration, data acquisition, or field communication that other researchers could pick up and extend. Instead, we are pasting together vendor modules with a specific data collection goal in mind, treating the instrument as a means to a publication rather than as a contribution in its own right. The result is thousands of nearly identical systems, each built from scratch, each with the same vendor dependencies, none of them advancing the state of the art in scientific instrumentation.

This is not a failure of individual researchers. It is a structural problem. The incentive system in academic science rewards papers and data, not tools. A well-designed, reusable sensor framework is harder to publish than a novel dataset. A graduate student who spends six months writing robust, portable instrument firmware will have less to show a tenure committee than one

who spent that time collecting data with a quick-and-dirty rig. The vendors understand this perfectly; their entire business model depends on researchers who need something that works right now, who do not have time to evaluate alternatives, and who will reach for whatever the last person in their lab used.

The conversations I had in the presentation hall confirmed this. Researchers knew their systems were fragile. They knew the code was not portable. Several admitted to using LLMs in lieu of finding a research software developer. But seemingly, nobody had the time or the institutional support to do it differently. The pressure to produce data, to publish, to justify the next grant cycle; it all pointed toward the fastest possible path from sensor to spreadsheet. And that path runs straight through the vendor ecosystem. This is not a sustainable pattern.

Conclusion

Corporate and capitalistic interests have no business driving the tools of scientific exploration. The purpose of research instrumentation is to produce trustworthy, reproducible measurements; not to generate recurring revenue for semiconductor companies. Every proprietary firmware blob, every undocumented register, every vendor- locked SDK represents a compromise in the integrity of the scientific process.

The embedded hardware industry will not voluntarily change course. Companies like ST and TI are behaving rationally within their own incentive structures. The change must come from the research community itself: in procurement decisions, in how labs train new students, in what we consider acceptable in a datasheet, and in our willingness to invest the additional effort that open tools sometimes require. The alternative is a future where the instruments of science are black boxes, where reproducibility depends on the continued goodwill and product support of corporations, and where the foundations of our measurements rest on code we are not permitted to read.

Glossary

API (Application Programming Interface): A defined set of rules and protocols that allows one piece of software to communicate with another. When this paper criticizes "opaque APIs," it means the vendor provides a way to *use* the sensor but not a way to *understand* what the sensor is actually doing internally.

Arduino: An open-source electronics platform built around simple microcontroller boards and an easy-to-use programming environment. Widely used in education and prototyping. In this

paper, "Arduino initialization code" refers to the use of Arduino's simplified programming libraries to set up hardware, which trades ease of use for limited control and portability.

Binary blob: A block of compiled machine code distributed without its human-readable source code. The user must load and execute it as-is, with no ability to inspect, audit, or modify what the code actually does. In the context of the VL53L5CX sensor, the binary blob is firmware that must be uploaded to the sensor before it can operate.

BLAST (Bursty, Light, Asynchronous, Stealth, Transitional): A design philosophy used by the DASH7 protocol. It describes a communication pattern where devices send short, infrequent messages without needing a persistent connection, where transmissions are difficult to detect, and where devices can move between different operating states. This contrasts with always-on or scheduled communication patterns.

Chirp spread spectrum (CSS): A radio modulation technique where the signal frequency sweeps (or "chirps") across a range of frequencies over time. This makes the signal resistant to noise and interference and allows it to travel long distances at low power. It is the core physical-layer technology behind LoRa.

Code Composer Studio (CCS): Texas Instruments' proprietary integrated development environment for programming their microcontrollers. Historically required expensive per-device licenses; now free, but still designed to keep developers working within TI's software ecosystem.

D7AP (DASH7 Alliance Protocol): An open wireless communication standard for sensor and actuator networks operating in sub-1 GHz radio bands. Managed by a non-profit consortium. Unlike LoRa, DASH7's entire specification (including the physical layer) is open and can be implemented on commodity hardware from multiple manufacturers.

Driver: Software that allows an operating system or application to communicate with a specific piece of hardware. A "vendor-specific driver" is one written by the hardware manufacturer that may only work with their proprietary tools, as opposed to an open-source driver that anyone can inspect and modify.

Firmware: Permanent software programmed into a hardware device (such as a sensor or microcontroller) that controls how the device operates at a low level. Unlike application software, firmware typically runs directly on the hardware without a full operating system.

GCC (GNU Compiler Collection): A free, open-source set of compilers that can translate human-readable source code into machine code for a wide variety of processor architectures. It is the standard open-source compiler used in embedded development as an alternative to vendor-proprietary compilers.

GNU Radio: A free, open-source software toolkit that provides signal processing building blocks for implementing software-defined radios (SDRs). Researchers use it to build custom radio systems, analyze wireless signals, and reverse-engineer proprietary radio protocols like LoRa.

HAL (Hardware Abstraction Layer): A software layer that sits between the application code and the raw hardware registers of a microcontroller. It provides simplified function calls (e.g., "send data over UART") so the programmer does not need to manipulate individual hardware registers directly. ST's HAL is criticized in this paper for being buggy and for discouraging developers from understanding the underlying hardware.

I2C (Inter-Integrated Circuit): A widely used communication protocol that allows multiple chips on a circuit board to talk to each other using just two wires (a clock line and a data line). Sensors, displays, and other peripherals commonly use I2C to communicate with a microcontroller.

IDE (Integrated Development Environment): A software application that combines a code editor, compiler, debugger, and other tools into a single interface for writing and testing programs. Examples include TI's Code Composer Studio and ST's STM32CubeIDE.

ISM bands (Industrial, Scientific, and Medical bands): Portions of the radio spectrum that governments have set aside for use without requiring an individual license. Common ISM frequencies include 433 MHz, 868 MHz, 915 MHz, and 2.4 GHz. Wi-Fi, Bluetooth, LoRa, and DASH7 all operate in ISM bands.

LoRa (Long Range): A proprietary radio modulation technology developed by Semtech that enables long-distance, low-power wireless communication. "LoRa" refers specifically to the physical radio layer. It is distinct from LoRaWAN, which is the network protocol built on top of LoRa.

LoRaWAN: The open network-layer protocol that defines how devices communicate over LoRa radio links, including how they join networks, send and receive data, and manage encryption. Managed by the LoRa Alliance. While LoRaWAN is openly documented, it still depends on Semtech's proprietary LoRa radio technology underneath.

LPWAN (Low-Power Wide-Area Network): A category of wireless network technologies designed for devices that need to send small amounts of data over long distances while using very little battery power. LoRa, DASH7, and NB-IoT are all LPWAN technologies.

MCU / Microcontroller: A small, self-contained computer on a single chip, typically including a processor, memory, and input/output peripherals. Microcontrollers are the "brains" inside embedded devices like sensors, instruments, and IoT nodes. Examples include ST's STM32 family and TI's CC13xx/CC26xx series.

Modulation: The technique used to encode digital information onto a radio signal for wireless transmission. Different modulation schemes (such as chirp spread spectrum, FSK, or OFDM) offer different tradeoffs in range, data rate, power consumption, and noise resistance.

NB-IoT (Narrowband Internet of Things): A cellular-based LPWAN technology standardized by 3GPP that operates on existing mobile network infrastructure. Unlike LoRa and DASH7, NB-IoT requires a cellular carrier and licensed spectrum.

OpenOCD (Open On-Chip Debugger): A free, open-source tool used to program and debug microcontrollers through standard hardware debug interfaces. It is the open-source alternative to vendor-specific programming and debugging tools.

OSI layers: The Open Systems Interconnection model divides network communication into seven layers, from the physical radio signal (Layer 1) up through the application (Layer 7). When this paper discusses "physical layer" (PHY) versus "network layer," it is referring to different levels in this stack. The physical layer defines how raw bits are transmitted over the air; the network layer defines how devices find each other and route messages.

PHY (Physical Layer): The lowest layer of a communication system, responsible for the actual transmission and reception of raw data over a physical medium (in this case, radio waves). When this paper says LoRa's PHY is proprietary, it means the fundamental method by which LoRa encodes and sends radio signals is owned and patented by Semtech.

Race condition: A software bug that occurs when two or more processes or interrupts try to access or modify the same data at the same time, and the outcome depends on the unpredictable timing of their execution. The STM32 HAL locking bug described in this paper is a race condition: the lock mechanism meant to prevent simultaneous access does not actually work when hardware interrupts are involved.

Register / Register map: In a microcontroller or sensor, a register is a small, named memory location that controls or reports on a specific hardware function (e.g., "start a measurement," "read the temperature value"). A register map is the complete listing of all such registers, their addresses, and what each bit means. It is the fundamental documentation that allows a developer to control the hardware directly without relying on vendor software.

RFID (Radio-Frequency Identification): A technology that uses radio waves to automatically identify and track tags attached to objects. DASH7 evolved from the ISO/IEC 18000-7 active RFID standard, which was originally designed for military logistics and asset tracking.

RISC-V: A free and open instruction set architecture (ISA) for designing computer processors. Unlike proprietary architectures (such as ARM, which requires licensing), anyone can design and manufacture a RISC-V processor without paying royalties. It is gaining adoption as an open alternative in embedded systems.

RTOS (Real-Time Operating System): A lightweight operating system designed for embedded devices that must respond to events within strict, predictable time constraints. Examples include Zephyr (open-source, vendor-neutral) and TI's proprietary RTOS. A real-time OS ensures that critical tasks like reading a sensor or sending a radio packet happen on schedule.

SDK (Software Development Kit): A collection of software tools, libraries, code examples, and documentation provided by a vendor to help developers build applications for their hardware. SDKs from companies like TI and ST are designed to make development easier, but they also create dependency on that vendor's ecosystem.

SDR (Software-Defined Radio): A radio system where functions traditionally performed by dedicated hardware (filters, modulators, decoders) are instead implemented in software running on a general-purpose computer. SDRs allow researchers to analyze, decode, and even transmit radio signals without being locked to a specific vendor's radio chip.

Second-sourcing: The availability of functionally equivalent components from more than one manufacturer. If a sensor or chip is only made by one company, that company has monopoly control over supply, pricing, and documentation. Second-sourcing provides alternatives and reduces vendor dependency.

SoC (System on Chip): A single integrated circuit that combines a microcontroller, radio transceiver, memory, and other components onto one chip. ST's STM32WL is a SoC that combines an STM32 microcontroller with a LoRa radio.

SPI (Serial Peripheral Interface): A communication protocol used between chips on a circuit board, similar in purpose to I2C but typically faster. SPI uses four wires and supports full-duplex communication (sending and receiving data simultaneously).

Sub-1 GHz: Radio frequencies below 1 gigahertz, typically referring to the 433 MHz, 868 MHz, and 915 MHz ISM bands. Sub-1 GHz signals travel farther and penetrate buildings better than higher-frequency signals (like 2.4 GHz Wi-Fi), making them well-suited for long-range sensor networks.

Time-of-Flight (ToF) sensor: A sensor that measures distance by emitting a pulse of light (typically a laser) and measuring how long it takes for the reflected light to return. ST's VL53L5CX and VL53L8CX are ToF sensors that measure distances across an 8×8 grid of zones simultaneously.

Toolchain: The complete set of software tools needed to write, compile, and load programs onto a microcontroller. This typically includes a compiler (translates code to machine language), a linker (combines code modules), and a programmer/debugger (loads code onto the chip and allows step-by-step testing). An "open-source toolchain" means all of these tools are freely available, such as GCC + OpenOCD.

UART (Universal Asynchronous Receiver-Transmitter): A simple, widely used hardware communication protocol for sending data between devices one bit at a time over a serial connection. It is one of the most basic and common ways to connect a microcontroller to sensors, GPS modules, computers, and other peripherals.

Vendor lock-in: A situation where a customer becomes dependent on a specific vendor's products and cannot easily switch to a competitor without significant cost or effort. In embedded systems, lock-in typically occurs through proprietary software tools, undocumented hardware, or ecosystem dependencies that make it impractical to use alternative components.

Zephyr RTOS: An open-source, vendor-neutral real-time operating system maintained by the Linux Foundation. It supports microcontrollers from many different manufacturers (ST, TI, Nordic, NXP, and others), allowing developers to write portable code that is not tied to any single vendor's software ecosystem.

References

[1] A. Gibb Seidle and J. M. Pearce, "Overcoming Limitations of Proprietary Scientific Hardware Funding," Journal of the Knowledge Economy, Springer, 2025. doi:10.1007/s13132-025-02783-w

[2] R. W. Bowman, "Improving instrument reproducibility with open source hardware," Nature Reviews Methods Primers, vol. 3, no. 27, 2023. doi:10.1038/s43586-023-00218-x

[3] M. Oellermann et al., "Open Hardware in Science: The Benefits of Open Electronics," Integrative and Comparative Biology, vol. 62, no. 4, pp. 1061-1075, 2022. doi:10.1093/icb/icac043

[4] STMicroelectronics Community Forum, "[BUG] STM32 HAL driver lock mechanism is not interrupt safe," community.st.com, thread ID 289837.

[5] STMicroelectronics Community Forum, "Questions surrounding __HAL_LOCK," community.st.com, thread ID 480382. (Post by Fluke engineer, April 2016.)

[6] STMicroelectronics, "VL53L5CX Datasheet: Time-of-Flight 8x8 multizone ranging sensor with wide field of view," DS13754, Rev. 5, 2022.

[7] STMicroelectronics, "VL53L8CX Datasheet: Low-power high- performance 8x8 multizone Time-of-Flight sensor," DS14161, 2023.

[8] "STxP70/STHORM/P2012: ST's secret microcontroller," the6p4c.github.io, June 2022. https://the6p4c.github.io/2022/06/13/stxp70-sthorm-p2012.html

[9] STMicroelectronics Community Forum, "VL53L5CX Register description," community.st.com, thread ID 114785, March 2022.

[10] Pololu, "VL53L8CX Time-of-Flight 8x8-Zone Distance Sensor Carrier with Voltage Regulators, 400cm Max," product page #3419. https://www.pololu.com/product/3419

[11] STMicroelectronics Community Forum, "VL53L0X Register Map?" community.st.com, thread ID 453873, June 2016.

[12] GrimbiXcode, "VL53L0X-Register-Map: A crowd sourced extended API documentation for the ToF sensor VL53L0X from ST," GitHub. https://github.com/GrimbiXcode/VL53L0X-Register-Map

[13] "Code Composer Studio," Wikipedia, citing TI licensing history. CCS v7 dropped paid license requirement in September 2016.

[14] F. Busacca et al., note that "only its medium access control (MAC) layer, known as LoRaWAN, is open and its physical and logical link control layers [...] are still only partially understood." See [17].

[15] M. Knight and B. Seeber, "Decoding LoRa: Realizing a Modern LPWAN with SDR," Proceedings of the GNU Radio Conference 2016, Boulder, CO, September 2016.

[16] J. Tapparel, O. Afisiadis, P. Mayoraz, A. Balatsoukas- Stimming, and A. Burg, "An Open-Source LoRa Physical Layer Prototype on GNU Radio," 2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), Atlanta, GA, May 2020. doi:10.1109/SPAWC48557.2020.9154273

[17] F. Busacca, S. Mangione, S. Palazzo, F. Restuccia, and I. Tinnirello, "SDR-LoRa, an open-source, full-fledged implementation of LoRa on Software-Defined-Radios: Design and potential exploitation," Computer Networks, vol. 241, 110194, March 2024. doi:10.1016/j.comnet.2024.110194

[18] CNLohr, "LoLRa: Transmitting LoRa packets without radio using CH32V003, ESP8266, or ESP32-S2 MCU," GitHub, 2024.

[19] Meshtastic Project, "Introduction," meshtastic.org. https://meshtastic.org/docs/introduction/

[20] "DASH7," Wikipedia. Citing DASH7 Alliance Protocol specification, ISO/IEC 18000-7, and DoD $429M contract announcement (January 2009).

[21] DASH7 Alliance, "DASH7 Alliance Protocol Specification v1.2," freely available at https://www.dash7-alliance.org

[22] "OSS-7: an opensource DASH7 stack," FOSDEM 2018 presentation. https://archive.fosdem.org/2018/schedule/event/oss7_dash7/

[23] OSS-7 Open Source Stack, Apache License 2.0, GitHub. https://github.com/MOSAIC-LoPoW/dash7-ap-open-source-stack

[24] M. Weyn, G. Ergeerts, L. Wante, C. Vercauteren, and P. Hellinckx, "Survey of the DASH7 Alliance Protocol for 433 MHz Wireless Sensor Communication," International Journal of Distributed Sensor Networks, vol. 9, no. 12, 2013. doi:10.1155/2013/870430

[25] G. Ergeerts, M. Nikodem, D. Subotic, T. Surmacz, B. Wojciechowski, P. De Meulenaere, and M. Weyn, "DASH7 Alliance Protocol in Monitoring Applications," 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), pp. 623-628, 2015. doi:10.1109/3PGCIC.2015.93

[26] DASH7 Alliance, "Software-Defined Radio-Based Internet of Things Communication Systems: An Application for the DASH7 Alliance Protocol," Whitepapers and Publications. https://www.dash7-alliance.org/whitepapers-and-publications/

[27] Haystack Technologies, "Haystack for LoRa," haystacktechnologies.com. Also: P. Burns, "How To Save Big Money On Semtech's LoRa," Medium/HackerNoon, 2018.

[28] W. Ayoub, A. E. Samhat, F. Nouvel, M. Mroue, and J.-C. Preevotet, "Internet of Mobile Things: Overview of LoRaWAN, DASH7, and NB-IoT in LPWANs Standards and Supported Mobility," IEEE Communications Surveys & Tutorials, vol. 21, no. 2, pp. 1561-1581, 2019. doi:10.1109/COMST.2018.2877382 (Also available: hal.science/hal-01901612)