I4C… Improving I²C's Dynamism and Efficiency

Guangyu Feng
University of California, Berkeley
guangyu_feng@berkeley.edu

Tess Despres
University of California, Berkeley
tdespres@berkeley.edu

Paul de La Sayette
University of California, Berkeley
paul.delasayette@berkeley.edu

Prabal Dutta
University of California, Berkeley
prabal@berkeley.edu

Abstract

I²C is a popular interconnect bus for integrated circuits that employs an open-drain design to support multiple controllers. Originally designed for television motherboards with a static configuration of chips, I²C is now used in nearly every corner of electronics, from wearables to phones to satellites. Some emerging applications, however, are more dynamic and distributed than I²C can support today, limiting its utility. In particular, one typically static, design-time parameter—the value of a pull-up resistor—is chosen to roughly match the total distributed bus and gate capacitance, ensuring a pull-up rise time that satisfies the I²C spec. But with greater dynamism—when many I²C nodes are added or removed at run-time—a static pull-up resistor can lead to a bus that is at best inefficient and at worst inoperable. In this paper, we present a system that can measure the rise time *in situ* and at I²C data rates, allowing us to optimally adjust the pull-up resistor at run-time. With this rise time measurement capability in hand, we show how it can be used for other useful and novel functions, including detecting when I²C nodes are added or removed, enabling an efficient inband-interrupt signaling scheme that eliminates the need for interrupt polling, and even full-duplex reverse data transfer encoded into clock edges. We implement our design, called I4C, using commercial microcontrollers and discrete electronics, and evaluate it on a testbed of several nodes, demonstrating its viability and efficiency.

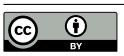
CCS Concepts

• **Computer systems organization → Embedded and cyber-physical systems**; • **Hardware → Communication hardware, interfaces and storage**.

Keywords

I²C, SMBus, Dynamic Bus Management, Plug-and-Play

ACM Reference Format:
Guangyu Feng, Tess Despres, Paul de La Sayette, and Prabal Dutta. 2025. I4C… Improving I²C's Dynamism and Efficiency. In *The 23rd ACM Conference on Embedded Networked Sensor Systems (SenSys '25), May 6–9, 2025, Irvine, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3715014.3722063



SenSys '25, Irvine, CA, USA

ACM ISBN 979-8-4007-1479-5/2025/05
https://doi.org/10.1145/3715014.3722063

1 Introduction

From its humble beginnings in the 1980s as the Inter-Integrated Circuit communications protocol to connect chips on television motherboards, I²C has blossomed into one of the most pervasive board- and system-level interconnect buses in existence today [11]. Wearable devices and environmental sensor systems network modules together with I²C [30, 40, 41]. Computing systems, ranging from phones to gaming consoles to data centers, use I²C, alongside derivative protocols such as SMBus [16, 26]. Robotic systems communicate using I²C, and the protocol has even gone to space with variants such as SPA-1 [23, 29, 32]. I²C is a widely-adopted industry standard because it occupies a sweet spot that balances speed, space, and complexity. I²C offers modest bus speeds (typically 100 kHz), supports multiple controllers that can independently initiate communications (with built-in support for arbitration), and requires only two signal lines which makes it an ideal choice for small devices (with very constrained pin counts and often limited space for routing buses).

A victim of its own success, today's emerging applications increasingly demand more dynamism than I²C can efficiently support, even with hot-swap capable interface chips. Some contemporary applications—like networked smart clothing, environmental sensing systems, and robotics—benefit from modular, dynamic designs [20, 24, 28, 31, 35]. System-level dynamism enables rapid, on-the-fly reconfiguration for different types of conditions, such as daisy-chaining between three and 36 energy-harvesting energy meters in a circuit breaker panel and connecting them to a data logger or radio [22]. Beyond plug-and-play sensors, dynamic systems can also integrate distinct controller devices (*e.g.* microcontrollers) to distribute functionality efficiently. Furthermore, by allowing devices to fail gracefully, dynamism enhances system reliability, particularly in harsh or hostile environments where failure is more common, by adapting operations to the available resources.

Problems arise, however, when unmodified I²C is used in such dynamic systems. I²C was not designed to support a variable number of devices because the open-drain design requires statically configuring the value of a pull-up resistor at design-time. This means that pull-up resistors are often configured for the worst case, anticipating the largest number of devices and highest possible bus capacitance, which results in a higher power dissipation than necessary. Perhaps worse, since I²C was not designed for dynamism, adding or removing devices in a running system risks jolting the bus into a stuck state, halting communications. Fortunately, hot-swap capable interface chips alleviate the stuck-state problem, but they do not solve the general problem of dynamism and efficiency.

In this paper, we introduce I4C, a set of backwards-compatible enhancements to I^2C that better supports dynamism in systems. Since I^2C is already well supported on many devices, we prioritize the ability to share a single physical bus to concurrently support I^2C and I4C, preserving full I^2C functionality while improving the operation of legacy devices by reducing total bus power dissipation, and enabling new capabilities between I4C devices. I4C addresses the dynamism challenge by measuring the bus rise time *in situ*. We focus on rise time because if the bus pull-up resistors are too weak, as additional devices are added, the bus capacitance and rise time can increase beyond the I^2C specification. Sensing rise time makes it possible to adjust the pull-up resistance, to maintain a healthy rise time for a range of dynamic configurations. Adjusting the pull-up resistors can also simultaneously reduce power dissipation when fewer devices are on the bus, because a larger resistor can be used, which limits current when the bus is driven low. I4C measures and adjusts the bus rise time in real-time.

By measuring the bus, we can enable a slew of other features as well. Monitoring the rise time can detect anomalies and identify signal integrity issues. Even in static configurations, I^2C pull-up configuration is difficult to get right. For example, target devices may include unknown internal pull-ups and long wires over large ground planes can introduce unexpected capacitance. I4C automatically handles these cases. Additionally, I4C can detect rise time changes when devices are added or removed from the bus. Detecting a device change is a useful system feature for checking if all devices are operational and for prompting the controller to assign addresses and track system resources when conditions on the bus appear to have changed due to persistent rise time variations.

With a small amount of logic on the target device, we can extend I4C to enable in-band interrupts. In I^2C, interrupts are not built-in and systems need to either poll target devices, wasting cycles when no communication is needed, or implement an interrupt line. In I4C, a key insight is that target devices can use rise time modulation as a side channel signal. Each target device is assigned its own rising edge which it can modulate to signal an interrupt. This allows multiple target devices to signal their pending interrupts within a single I^2C transaction which may be completely unrelated to any pending interrupts. I4C introduces interrupts and full-duplex communication into I^2C without adding additional lines on the bus.

In this paper, we design, implement, and evaluate I4C, which automates a typically static and manual operation—pull-up resistor selection—using rise time detection and modulation circuits. I4C is designed with the high-level goals of supporting dynamism, both at configuration and run-times, prioritizing power efficiency, and enabling full-duplex communication and interrupts. We combine rise time measurement, pull-up resistor adjustment, and rise time modulation to meet these goals and implement our system using a testbed of Raspberry Pi Pico 2 (RP2040) devices [14], custom circuits, and software. We show that we are able to successfully measure and vary rise time across a wide number of target devices. Additionally, our I4C system saves power compared to I^2C systems configured for the same number of devices. Finally, we show detection of dynamic device addition and interrupts using a combination of rise time modulation and sensing. I4C demonstrates that I^2C can be enhanced along several dimensions to enable backwards-compatible, dynamic systems without requiring additional wires.

2 Related Work

We start with a brief introduction to I^2C, its trending modularity, and its major derivatives, then review key bus operations related to dynamism—hot-swapping, device discovery and departure, dynamic address assignment, and service discovery and binding—and end our discussion with pull-up resistor selection strategies.

2.1 I^2C Background

I^2C is a half-duplex serial communication protocol for efficient data exchange between processors and peripheral devices. It transmits data sequentially over only two physical lines: SDA (for data) and SCL (for clock). These lines form a shared bus where multiple controllers and targets communicate. In the communication protocol, a controller initiates data transfer by addressing a specific device, followed by sequential data exchange and acknowledgment signals.

At the physical layer, I^2C uses an open-drain design. Each device has a transistor with its drain connected to the shared bus line and its source grounded, while a pull-up resistor connects the bus line to the supply. This configuration forms an RC circuit, impacting the bus rise time proportional to the product of the pull-up resistance (R_p) and wire/gate capacitance (C_{bus}). As devices are added, the total bus capacitance increases. A device transmits `'0'` by turning on the transistor, thus grounding the bus line, and transmits `'1'` by turning it off, allowing the pull-up resistor to pull the bus to the supply voltage. This design prevents bus contention with wired-and bus arbitration: if multiple devices write conflicting values on the bus simultaneously, the `'0'` dominates. The same principle allows the target of a message to hold the SCL (clock) line low if it needs more time to perform the task—a technique called clock-stretching.

2.2 I^2C Based Systems

Although I^2C does not natively support dynamic device addition and removal, engineers have long explored ways to use it dynamically, both during bus configuration and active operation. This need has become more pressing with the rise of large-scale sensing systems—such as tactile sensing arrays with over 40 accelerometers using I^2C buses [43]—and modular systems which would benefit from a dynamic I^2C bus operation. Companies such as Useful Sensors have offered various sensing modules that can be easily added or swapped on top of the I^2C protocol [19], while modular controller boards are used for robotics where different modules such as sensor and power supply boards can be connected to make a "custom" control board [28]. Similarly, LegoSENSE creates a modular I^2C based IoT platform for sensing [45], and the MASS system, built on the I^2C bus, explicitly trades off power efficiency for enhanced modularity [25]. These developments, along with many other sensor systems that benefit from dynamic modularity [20, 33, 34, 38, 41, 42], underscore the growing demand for dynamic I^2C bus operation, but they do not address the technical challenges that I4C does.

2.3 I^2C Derivatives

A variety of I^2C-derivative protocols have emerged with the goal of either increasing the capabilities of the I^2C bus or avoiding some of its more problematic constraints. These derivatives include SMBus, PMBus, ACCESS.bus and I3C, closely-related siblings like TWI, and purely mechanical adaptations like Qwiic and STEMMA.

SMBus, introduced by Intel in 1995, is designed for system management in computers and enhances I²C reliability and efficiency through features such as built-in timeout, standardized commands, and low-power modes [16]. It is interoperable with I²C, but requires that devices acknowledge their own addresses, utilize a minimum SCL clock frequency of 10 kHz, support additional commands, and include a low-power mode. Similarly, PMBus was designed for power-management by defining a set of standardized commands and response formats for power management tasks, including commands to control and set output voltages [15]. Atmel introduced the Two Wire Interface (TWI) [7], which limits certain advanced I²C features to reduce hardware complexity and resource commands. ACCESS.bus, a collaboration between DEC and Philips, enhances I²C modularity by adding dedicated power and ground lines that enable hot-swapping and daisy-chaining of peripheral devices on a host PC [21], but it never gained market traction and was superceded by USB. Sparkfun developed the Qwiic system to reduce the mechanical complexity of wiring and enable plug-and-play sensors over I²C [17]. Adafruit has also adopted this method for making quick prototypes using I²C [8]. I3C, under development by the MIPI Alliance I3C Working Group, combines features of I²C and SPI to get a faster (10 Mbps), two-wire, low-power communication channel targeting smartphones [13]. These protocol extensions and derivatives illustrate that evolving demands have long pushed I²C into new regimes, sometimes possible within the I²C spec and at other times requiring new buses to meet needs.

However, these adaptations fail to address key challenges related to dynamic operations on an I²C bus. SMBus, PMBus, TWI, Qwiic, Stemma, and ACCESS.bus remain within the I²C framework, but lack support for dynamic configuration, while SPI and I3C introduce more fundamental changes that require hardware modifications. Compared with I4C, SPI does not support dynamism, and I3C, while offering partial dynamism support for I3C targets—such as address resolution protocol (ARP) and in-band interrupts—neither supports multiple active controllers on the bus nor enhances operations on legacy I²C devices. Moreover, I3C does not detect I3C device departures nor does it address the static pull-up resistor issue when communicating with traditional I²C devices. These limitations highlight an unmet need for increased dynamic capabilities for legacy I²C devices—a gap that I4C aims to address.

2.4 Dynamism in I²C and Derivatives

In this section, we discuss the existing support for dynamism in I²C and derivative protocols, and how I4C extends these capabilities.

Hot-Swapping. Hot-swapping devices—adding and removing devices while the bus is powered—is a valuable feature for modern dynamic systems that I²C was not originally designed to support. Hot-swapping involves a few key challenges. First, devices are designed such that power pins receive power before signal pins to prevent data corruption [39]. Second, conflicting transactions need to be avoided to prevent data corruption. This can be prevented by ensuring that devices only transmit after detecting a stop condition, or bus idle condition. Some buses, such as SMBus, incorporate additional timeout safeguards, to reset the bus if a transaction does not complete within the time limit. I4C adopts timeout, retransmissions, bus idle/stop detection, and wired-and bus arbitration to handle such conflicts. Third, when devices are first connected to the bus, their sudden surge in current draw from the bus and sudden increase in bus capacitance can result in bus voltage dips. Pre-charging the device's internal capacitance can help prevent the resulting device disruption or bus data errors. Alternatively, rise time accelerators can help by injecting current into the line when in transition and resulting in a faster rise time, but they can cause signal distortion when multiple devices are connected. Another mechanism to support hot-swapping is buffering different segments of the bus with an IC such as the TCA9511A [10] to reduce the total capacitance and prevent spikes. The trade-off is that buffering requires a pull-up resistor on every bus segment, increasing power draw. To address this challenge, I4C takes a different approach to support hot-swapping, by detecting rise time variations, adjusting the pull-up resistance, and regulating the bus current as needed, while avoiding rise time accelerators and buffers.

Device Discovery and Departure. Tracking device arrivals to and departures from the bus helps a system track available resources. In I²C, the standard approach relies on polling, where the controller repeatedly polls the bus to discover new devices and check if known devices are still present. To improve upon polling, some systems make target devices temporarily assume a controller role and send heartbeats to announce their arrival and presence—though most I²C targets lack the hardware capability to do so. However, both polling and heartbeat-based methods can be expensive, and during stable periods of bus configuration—when there is not device churn—they waste bandwidth, which could be better spent on communications. I4C eliminates this inefficiency by detecting rise time changes, which indicate a change in bus capacitance when devices are added or removed. Only when a change is detected will I4C poll to discover new devices or remove departed ones, reducing the bandwidth and power overhead of existing approaches.

Dynamic Address Assignment. For a newly added device, the controller on the bus must identify its address while ensuring that it does not have address conflicts with existing devices. SMBus addresses this by using an ARP, where the controller collects all devices' addresses via a broadcast command and dynamically reassigns new, unique addresses to devices with conflicts [16]. However, ARP only works for devices responding to the broadcast commands—typically controllers—and traditionally, needs to be triggered by polling to detect device changes or timeouts. In contrast, triggers for I4C ARP are efficient—executed only when an I4C controller detects that the bus configuration has changed.

Service Discovery and Binding. When devices are added to the bus, understanding their capabilities is important for the system to function efficiently. SMBus and PMBus reserve certain addresses for certain devices such as smart batteries [16]. It is not practical, however, to assign addresses for a large set of device types, because I²C addresses are short (7-bit or 10-bit), limiting the number of possibilities. Buses like Display Data Channel (DDC), developed for display interfaces, solve this problem by including an Extended Display Identification Data (EDID) structure which is communicated automatically upon device connection [18]. I4C initiates service discovery after detecting a change in the bus rise time and uses predefined encoded values to represent a device's services and capabilities, removing the need to map addresses or data structures to services, but permitting that level of indirection if useful.

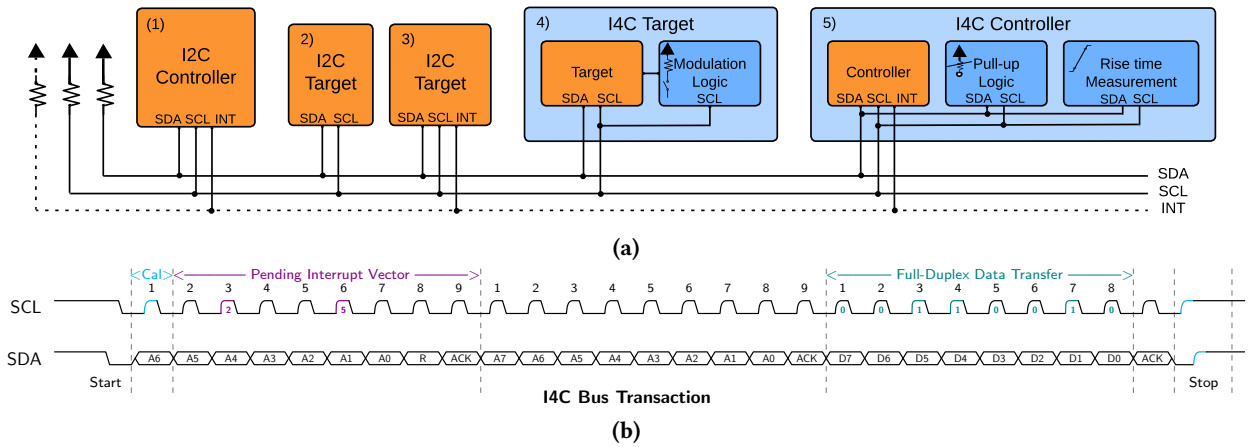


Figure 1: (a) Overview of the I²C/I4C system. I4C supports (i) traditional I²C devices and (ii) I4C-enabled ones, all on the same physical bus. **(b) Example I4C bus transaction.** I4C: (i) controllers automatically adjust pull-ups after measuring reference edge rise times (⟨ Cal ⟩), (ii) targets signal pending interrupts by modulating SCL rising edge rise times (⟨ Pending Interrupt Vector ⟩), and (iii) support full-duplex data transfers using the same scheme with targets modulating clock edges (⟨ Full-Duplex Data Transfer ⟩) during the data byte(s) transfer phase from the controller to the target during an I²C write command.

2.5 Pull-up Resistor Selection

Adding or removing devices at run-time makes static pull-up resistor values suboptimal. The open-drain bus design of I²C requires pull-up resistors whose resistance values strike a balance. With a static configuration of chips, designers must satisfy I²C system specifications [11], including maximum rise time (1,000 ns in standard mode and 300 ns in fast mode), limiting the maximum resistor value. Conversely, the spec limits the maximum allowable sink current to 3 mA, limiting the minimum pull-up resistance.

Power-sensitive designs employ static values closer to the maximum to limit unnecessary power dissipation, but this leaves less margin for additional capacitance from new devices or wiring [46]. Some researchers have even proposed toggling off resistors to save power [27]. Since fixed pull-up resistors cannot adapt to dynamic changes in bus capacitance, a standard approach in larger and modular systems is to segment the bus with buffers, which help isolate capacitance in each segment and maintain signal integrity. However, this solution introduces new challenges: each segment must have its own pull-up resistors, increasing power dissipation, and the buffers themselves add capacitance. Additionally, the fundamental challenge of selecting optimal pull-up resistance remains—now applied to each segment instead of the entire bus.

Another alternative is to assign each device (or group of devices) its own single pull-up resistor, matched to its gate capacitance, to offset the increase in bus capacitance and maintain a constant rise time. However, precisely pairing resistors to gate capacitance is challenging, and the method fails if any device lacks a dedicated pull-up—as is common with most I²C devices. Moreover, this approach does not account for cable length or other extra capacitance. By dynamically selecting the pull-up resistance value in real-time, I4C eliminates the static resistance trade-offs, offering both low-power operation and sufficient margin for newly added devices, while avoiding the complexity and power inefficiency of precautionarily-segmented networks.

3 System Overview

Figure 1a (top) shows a typical I²C network of controllers and targets, augmented with I4C-enabled devices. SCL (clock) and SDA (data) lines connect all the devices together. An INT (interrupt) line is sometimes used (outside of the I²C specification) to allow targets to notify a controller of a pending interrupt. Static pull-up resistors (shown left) are used to pull the bus voltage to a high value (`'1'`) while controllers and targets drive the bus to a low value (`'0'`), and controllers drive SCL even when a target is transmitting data.

An I4C controller includes a hardware and software subsystem that can: (i) measure the rise time of the I²C SCL and SDA signals on a clock cycle-by-cycle basis, (ii) detect the arrivals or departures of other devices, (iii) process the stream of rise time measurements and I²C protocol states, (iv) control the values of programmable pull-up resistances on the SCL and SDA lines to balance speed and efficiency, and ensure correctness, (v) extract a pending interrupt request number embedded in an SCL rising clock edge by a target, potentially in response to a shared interrupt line being asserted, (vi) communicate in (nearly) full-duplex with a target, and (vii) execute an ARP process to dynamically select the bus controller.

An I4C device can signal a pending interrupt by modulating the rise time of a particular clock pulse's rising edge during the address phase of an I²C bus transaction, or by modulating the clock's rising edges during the subsequent bytes, to transmit full-duplex data. In both cases, the target device's address uniquely identifies the edges that the target device is permitted to modulate.

Figure 1b (bottom) shows a typical I²C bus transaction with I4C's sensing and modulation of clock edges highlighted. The I4C controller measures the rise times shown in cyan to adjust the pull-ups on the SCL and SDA lines. Two devices indicate pending interrupts by modulating the rise times shown in indigo (SCL rising edges 3 and 6). A target transfers data to the controller (`0b00110010`) by modulating the SCL rise times shown in teal whiles the controller is simultaneously transmitting data to the target on the SDA line.

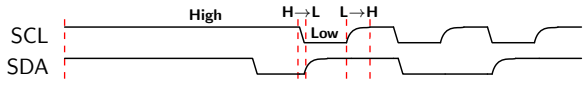


Figure 2: I²C signals traverse four stages. The stages, High, H→L, Low, and L→H, yield distinct power draws.

4 Design

I4C's primary design goal is to support greater dynamism in low-power I²C networks while achieving high efficiency and strong adherence to the I²C spec. I4C achieves these goals by introducing new capabilities to I²C controllers in the form of high-resolution signal rise time measurement and coarse-grained pull-up resistor control, and to I²C targets in the form of fine-grained pull-up control on select clock edges. These capabilities enable *new side-channels* that convey crucial information, including bus status, interrupt flags, and even full-duplex data streams hidden in plain sight.

4.1 Design Goals and Constraints

Motivated by the vision of an *ad hoc* group of a few—or many—I²C-compatible devices connected together and automatically configured in plug-and-play manner, we share our design considerations.

Greater Dynamism. We envision supporting as few I²C devices as may be needed for an application, or the maximum number of devices possible on an I²C segment, limited only by bus constraints (or actual bus failure, if so desired). When devices are added or removed, the changes are quickly detected on the very next I²C transaction, triggering algorithmic adjustment of pull-up resistor values to ensure performance, efficiency, and correctness. Device churn also initiates higher-level protocols like address resolution, service discovery, and resource binding immediately thereafter. And, if the bus becomes loaded beyond its specified operating regime, signaling that such a condition has occurred and the bus is operating in a "red zone" prior to bus failure.

Higher Efficiency. With the ability to detect signal rise time changes due to changes in bus capacitance or pull-up resistance, many avenues for efficiency become possible. Device churn is quickly detected without any periodic polling overhead (*e.g.* energy, latency, or bandwidth). The pull-up resistor values on I²C's SCL and SDA lines are tuned to the actual (and dynamic) bus capacitance rather than typically far more conservative estimates, which minimizes power dissipation. The energy and latency required to identify the source of a pending interrupt is greatly reduced compared to polling when, using I4C's approach, the target can raise an interrupt flag by modulating the rise time of a target address-specific clock edge in the next transaction. The latency can be reduced even further when the target asserts a shared interrupt line (INT), which immediately triggers the next I²C transaction. Finally, full-duplex communication, similar to SPI's MISO/MOSI model, is possible when the controller transmits data to the target using SDA (like MOSI) while the target transmits data to the controller by modulating the rise time of SCL clock edges (like MISO).

I²C Protocol Compliance. For maximum utility, I4C's support for greater dynamism with higher efficiency requires strict adherence to the I²C bus specification. In particular, I²C standard mode requires the following conditions are met:

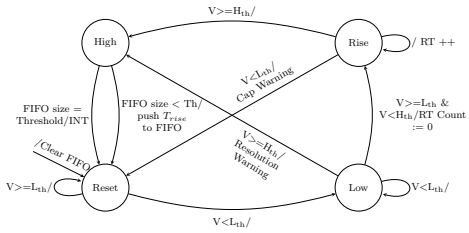


Figure 3: FSM for rise time measurement. The SCL and SDA rise times are measured by counting the number of fast clock cycles that occur between the low and high states. This FSM enables I4C to measure and report the rise time during I²C communications.

- $C_{bus} \leq 400$ pF (the total bus capacitance including wire and gate loads cannot exceed this figure).
- $R_p \geq V_{dd}/I_{max}$ (the pull-up resistor must limit current to no more than 3.3 mA independent of the bus supply voltage, which can be either 3.3 V or 5.0 V):
  $\Rightarrow$ If $V_{dd} = 3.3$ V, $R_p \geq 1.1$ kΩ
  $\Rightarrow$ If $V_{dd} = 5.0$ V, $R_p \geq 1.67$ kΩ
- $T_{rise} = 0.85\tau = 0.85 \cdot R_p \cdot C_{bus} \leq$1,000 ns (the 30%-to-70% rise-time for a bus with C_{bus} capacitance and R_p pull-up resistance, is further constrained as follows):
  $\Rightarrow$ If $C_{bus} = 50$ pF, $R_p \leq 23.5$ kΩ
  $\Rightarrow$ If $C_{bus} = 400$ pF, $R_p \leq 2.9$ kΩ

Note that I²C's fast mode uses somewhat different values.

4.2 I²C Power Model

With the key bus parameters and their constraints presented, we now discuss how these parameters contribute to I²C's power dissipation. Figure 2 shows the four stages that I²C's SCL and SDA transition through. The power dissipated in these states are as follow, emphasizing the critical importance of minimizing C_{bus} and maximizing R_p and f_{SCL}:

High (Idle): There is minimal power dissipation in this state other than very small leakage currents.

High→Low: The energy stored in C_{bus}, the bus capacitance representing the total gate and wire load, is dissipated by the device that drives the bus low. The energy lost as heat is $E_{H\rightarrow L} = \frac{1}{2}C_{bus}V_{dd}^2$ for a single SCL or SDA edge transition and $P_{H\rightarrow L} = \frac{1}{2}C_{bus}V_{dd}^2 f_{SCL}$, when SCL is clocked at f_{SCL}.

Low: Power is dissipated in the pull-up resistor and to a lesser extent by the device driving the bus low. This loss is approximated by $E_{low} = (\alpha_{low}/f_{SCL}) \cdot (V_{dd}^2/R_p)$, where α_{low} represents the fraction of each clock cycle that the signal is driven low (a factor since I²C's SCL may be unbalanced).

Low→High: The bus capacitance, C_{bus}, is charged via the pull-up resistor, R_p, which dissipates the same energy per transition independent of the value of R_p. This is because the current in the resistor is given by $I(t) = \frac{V_{dd}}{R_p}e^{-t/R_pC_{bus}}$. The energy lost as heat in the pull-up resistor is given by $E_{L\rightarrow H} = \int_0^\infty I^2(t)R_p dt = \frac{1}{2}C_{bus}V_{dd}^2$, which is equal to the energy stored in the capacitor, independent of the actual value of R_p.

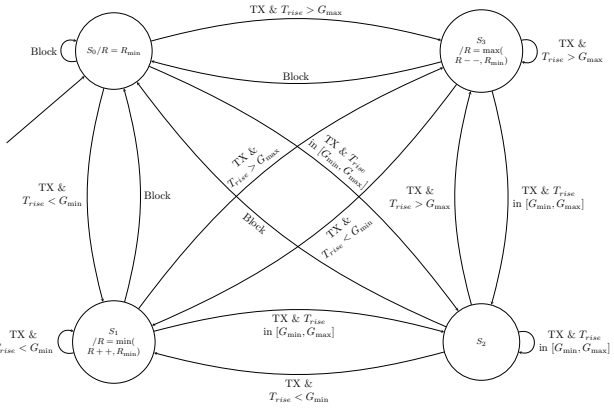


Figure 4: FSM for controller rise time adjustment. R is the pull-up resistance value, R_{min} is the minimum available pull-up resistance value, and $[G_{min}, G_{max}]$ is the target rise time range. This FSM allows the controller to regulate the rise time of the bus.

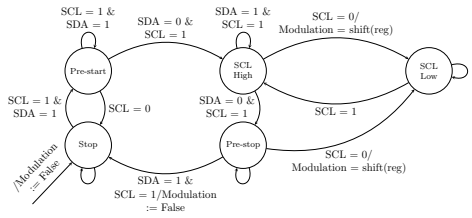


Figure 5: FSM for target device rise time modulation. When the target detects a start condition, it modulates rising SCL edges based on the shifted out bit from the shift register. This FSM shows that a target can toggle the rise times of specific SCL edges.

4.3 I4C Controller

The I4C controller measures the rise time of every single rising edge of I^{2}C's SCL and SDA lines. For clarity, the following discussion only covers SCL, but identical logic also tracks SDA. The measurement chain begins with two comparators; one is configured to trigger when $V_{SCL} \geq 0.3 \cdot V_{dd}$ and the other when $V_{SCL} \geq 0.7 \cdot V_{dd}$. The time difference between the low-to-high transitions of the two comparators captures the SCL rise time. The FSM shown in Figure 3 takes these two comparator signals as asynchronous inputs and counts the number of cycles of a fast clock that elapse between them to measure the SCL rise time. The FSM also identifies abnormal behavior via two warning conditions: (i) a capacitance warning checks for a sudden voltage drop during a rising edge which indicates bus capacitance has exceeded the I^{2}C limit and (ii) a resolution warning which identifies rise time events in which where the rise time is shorter than the minimum interval that the FSM can measure.

Conversely, the FSM fast clock's minimum frequency is determined by the resolution needed to measure the SCL rise time. Recall that we use rise time measurements to detect device arrivals and departures, which we expect change the bus capacitance by at least ±10 pF since I^{2}C loads are restricted to 10 pF and PCB wire traces are typically around 1 pF/cm. The fastest rise time occurs when $R_p = 1.1$ kΩ and $C_{bus} = 10$ pF, yielding $\tau = 11$ ns and $T_{rise} = 0.85\tau = 9.3$ ns, implying $f_{CLK} \geq 107$ MHz. Because rise time is linear in C_{bus}, the same f_{CLK} is needed to resolve rise times when C_{bus} increases from 390 pF to 400 pF, assuming R_p is fixed.

However, R_p is not fixed. Rather, it is adjustable under software control for minimizing power, subject to the I^{2}C protocol constraints, as shown in Figure 4. This FSM selects the best pull-up resistor from a set of choices that fall within an acceptable range of values. When greater resolution is needed than f_{CLK} provides, increasing R_p within this acceptable range provides proportionately greater resolution without requiring higher clock rates.

Beyond the basic rise time measurement and modulation functions, the controller also takes actions depending on the particular pattern of changes in the rising edges. For example, when a new device joins or leaves the bus, controller software (not shown here) detects a change in the rise time of the first rising edge of SCL across consecutive transactions. This discovery then triggers address resolution, service discovery, and resource binding, to identify the device that may have just joined or left. Or, if the controller detects a change in the rise time of any of the 2$^{\text{nd}}$ through N^{th} rising edges in an I^{2}C transaction (for $N \leq 10$), then one (or more) interrupt(s) are pending, which triggers targeted polling of the device(s).

In some networks, there may be more than one controller, leading to conflicting rise time modulation actions. To prevent such conflicts, we can use the ARP protocol to assign unique local addresses, and designate a special address which is assigned to the controller responsible for the modulation.

We conclude our discussion of the I4C controller design with a note about estimating the actual bus capacitance in the presence of pull-up resistors outside of I4C's control. First, observe that if the only pull-up is the one under the control of the FSM in Figure 4, then determining C_{bus} is straightforward. Since rise-time (30%-to-70%) is 0.85τ, and $\tau = R_p C_{bus} = T_{rise}/0.85$, we see $C_{bus} = 1.18 \cdot T_{rise}/R_p$, allowing us to ensure both R_p and C_{bus} remain within the I^{2}C spec. The challenge arises if there are other pull-ups on the bus, since $T_{rise} = 0.85 \cdot (R_{unknown} || R_p) \cdot C_{bus}$. In that case, we have two unknowns and only one equation, which makes ensuring $R_{unknown} || R_p \geq V_{dd}/I_{max}$ difficult. We address this issue by selecting two different known values of R_p, termed R_1 and R_2, and generating two equations with two unknowns: $\tau_1 = (R_{unknown} + R_1)C_{bus}$ and $\tau_2 = (R_{unknown} + R_2)C_{bus}$, which can yield both $R_{unknown}$ and C_{bus}, letting I4C ensure protocol invariants in presence of unknown or varying pull-up resistances.

4.4 I4C Target

An I4C target is comparatively much simpler than the I4C controller and offers only two key functions—interrupt signaling and full-duplex data transmission. These functions are built on the ability to modulate any particular rising edge(s) of the SCL line by adding and removing a controlled impedance from the line. An I4C target modulates any one of the 2$^{\text{nd}}$ through N^{th} rising edges of the SCL line using the FSM shown in Figure 5 ($N = 10$ for 7-bit I^{2}C addresses). The index of the rising edge that a target should modulate is chosen so index = I2C_ADDR mod $(N-1)+1$. Adding a shift register to this FSM allows multiple SCL clock edges to be modulated, enabling full-duplex communications.

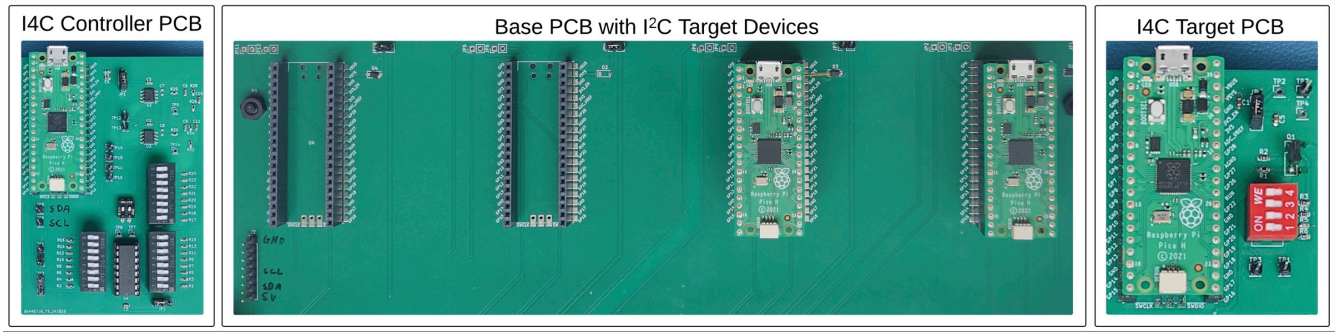


Figure 6: I4C testbed. The testbed includes: (i) an I4C *controller* with switchable resistance modulation arrays and a comparator rise time measurement circuit, (ii) a *base* with an I²C backplane and several I²C target sockets, and an I4C *target* with a rise time modulation circuit.

5 Implementation

In this section, we present the details of our I4C prototype implementation, which is built around the Raspberry Pi Pico (RP2040) and three custom printed circuit boards (PCBs), as Figure 6 shows.

5.1 I4C Controller

The I4C controller implements rise time measurement, pull-up resistor control, interrupt processing, and device/service discovery.

5.1.1 Rise Time Measurement. I²C rise time is defined as the elapsed time between the SCL (or SDA) voltage crossing a low and high threshold (30% and 70%, respectively, of V_{dd}). Each threshold is generated by a resistive voltage divider connected to V_{dd} and GND. TLV3202 comparators compare the SCL voltage to the thresholds, and the comparator outputs are wired to GPIO pins on the RP2040 controller. The controller determines the rise time by measuring the elapsed time between the low-to-high transitions of the two comparators, as Figure 7 shows.

The RP2040's Programmable IO (PIO) module executes the rise time measurement program in about 30 lines of PIO assembly, implementing the FSM shown in Figure 3. Each assembly instruction executes in a single RP2040 clock cycle. The program stays in the reset state until V_{SCL} or V_{SDA} becomes low. Upon a low threshold, the voltage comparator output changes, and the program transitions to the rise state and starts counting the number of clock cycles elapsed until the high threshold voltage is detected. The rise time is the clock cycle count multiplied by the RP2040 clock period (8 ns).

The PIO memory and FIFO buffer to the main program are limited to 2 and 8 bytes, respectively, while I²C rise time events occur at 100 kHz. To strike a balance between interrupting the main program too frequently and overflowing buffers, our implementation generates an interrupt when the FIFO is half full, which triggers the main program to drain the FIFO and clear the interrupt.

5.1.2 Pull-Up Resistor Control. The pull-up resistor control keeps the rise time below the maximum, and recovers the bus in case of a stuck-low state, by setting a low resistance. To implement this, we use a selectable array of pull-up resistor paths. We could also use a digital potentiometer as an alternative implementation. The pull-up resistor control circuit is shown in Figure 7. Resistor step values were carefully chosen to give an appropriate rise time range of $[max_{rt} \times a, max_{rt}]$ where a is a design parameter ($a < 1$) and max_{rt} is the maximum rise time (set to 1,000 ns according to the I²C specification). When a is determined, the circuit resistors can be picked to provide logarithmic steps, where $R_i/R_{i+1} = a^{1/(n-1)}$ with n being the number of resistance levels available. The resistor path is chosen using a dual 4-channel analog multiplexer, Analog Devices DG409 [2], which is manipulated by the controller's two GPIO pins. We control the multiplexer by implementing the FSM shown in Figure 4 in C code on the RP2040. The FSM switches between the four resistance levels to control the rise time.

5.1.3 Interrupt Processing. The I4C controller is also responsible for receiving and processing interrupts from targets. The first 10 SCL rising edges are reserved for targets to signal pending interrupts in our I4C implementation because I²C transactions must have at least ten rising edges. The first SCL rise time is reserved for calibration—to provide a reference rise time—and it is never modulated by any target. In our current implementation, if a bus has more than 9 devices, multiple devices will be assigned to a single interrupt index, so the controller may need to poll all devices whose addresses would map to that index. The controller subtracts each rise time from the first reference rise time to test whether any targets have pending interrupts, which is currently implemented in C code on the RP2040. The identity of interrupting target, in I4C, is discovered immediately by the controller upon receipt of the modulated SCL edge (as long as the number of targets is ≤ 9).

5.1.4 Device and Service Discovery. The controller in I4C can detect device churn when the rise time at an unmodulated SCL index changes beyond an empirically determined threshold. The change triggers ARP and subsequent service discoveries. In I4C ARP, devices initially have a default I4C address of `0x55` and respond with an unique per-device ID. The ARP procedure starts with the controller broadcasting a discovery command to `0x55`. If this command succeeds, the controller reads the unique ID at `0x55`, takes advantage of I²C bus arbitration to isolate one responding device, and then assigns the isolated device a new address using its unique ID. This process repeats until no device responds to the default address. Once addresses are assigned by the controller, service discovery starts. The target device resources are tracked in a "service registry" which is a table data structure that tracks the target registration, type, address, and custom field which can be used for priority.

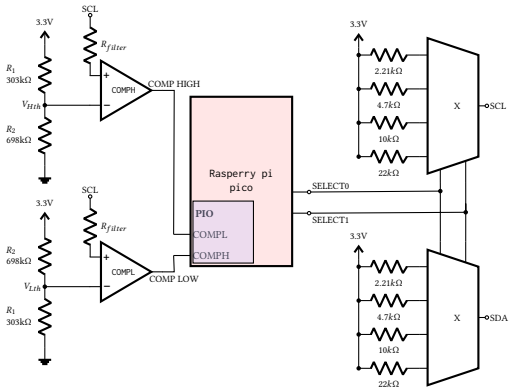


Figure 7: Rise time measurement circuit. The controller uses two comparators to determine when the bus lines pass the threshold voltages, enabling the controller to track the I4C bus rise time.

5.2 Target

The target device implementation includes a modulation circuit and modulation logic.

5.2.1 Rise Time Modulation. There are two aspects to rise time modulation on the target—the mechanics of modulating a single bit and the process of modulating the correct set of bits, which we discuss in this section.

Modulation mechanism. A target modulates the rise time of a bit by turning on a switch to connect a pull-up resistor in parallel with the existing bus pull-up resistor(s). This is implemented with a simple modulation circuit, shown in Figure 8b. The circuit consists of a PMOS transistor connected to V_{dd} that is used to gate a pull-up resistor on SCL. The PMOS itself is controlled by a modulation signal, which is triggered under either target software control or separately with discrete logic.

The target controls the modulation signal using the FSM shown in Figure 5. We implement this FSM on the RP2040 device in approximately 30 lines of PIO assembly. After an I^{2}C start condition is detected, the PIO shift register, preloaded by the main program, shifts out a bit at every SCL falling edge. If the bit is '1', the RP2040 target device pulls the modulation GPIO pin low, connecting the pull-up resistors in parallel to modulate the incoming rising edge. The modulation process is complete when the I^{2}C stop condition is detected. Both interrupt and full-duplex communication can be simultaneously implemented with this modulation scheme.

Clock edge modulation assignment. To prevent multiple targets from concurrently modulating the same clock edges, each device is assigned its own edge for interrupt signaling, while full-duplex communication modulates clock edges distinct from those reserved for interrupts. Additionally, a target can initiate a full-duplex message only when it has been specifically addressed. In our implementation, to initiate full-duplex communications, the controller initiates a write operation with full-duplex command code followed by a read, a common pattern in many I^{2}C communications. Upon receiving the command code, the target prepares the data for transmission to the controller by loading it into the PIO shift register, thereby initiating full-duplex communication.

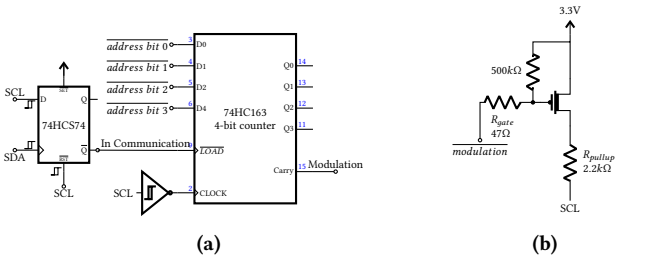


Figure 8: Rise time modulation circuits. Circuit (a) generates a control signal for modulation, replicating the PIO-based interrupting logic, and (b) uses a PMOS transistor to modulate rise time on SCL edges, based on the control signal. Circuit (a) can act as an alternative to a PIO-based approach, allowing the target device to modulate the rise time precisely on SCL edges.

Power optimization and modulation alternatives. Unlike PIO-based rise time *sensing*, which needs a high-frequency clock for resolution, the PIO-based rise time *modulation* does not inherently require one. A potential power optimization could be scaling down the clock frequency. We developed a circuit-based method, as an alternative to the PIO-based implementation, to generate the control signal for interrupt rise time modulation, as shown in Figure 8a. The left part of the circuit detects active I^{2}C communication using a Schmitt-triggered D flip-flop (74HCS74 [9]) clocked by SDA, with the D input and the negated asynchronous reset connected to SCL. As per I^{2}C specifications, SCL remains low at SDA edges except at the stop condition, resulting in Q', the indicator of communication status, set to high at the first SDA rising edge, remaining high throughout the transaction, and being set back to low during the I^{2}C stop condition.

The right part of the circuit tracks the number of SCL falling edges elapsed during the current I^{2}C bus transaction (*i.e.* since the most recent I^{2}C start condition) using a 4-bit counter (74HC163) [1]. The flip-flop's Q' output is connected to the negated synchronous LOAD pin of the counter. Upon detecting a stop condition, the counter resets and preloads $X - 1$ with all bits inverted, where X is the index of the rising edge to be modulated (counting from 1). During the next transaction, when the X^{th} SCL falling edge occurs, the counter overflows and triggers the modulation control signal to modulate the following X^{th} rising edge. This circuit, however, does not support I4C's full-duplex communication.

Table 1: I4C custom commands. Our implementation of I4C uses the following custom commands to facilitate device discovery, interrupt handling, and data processes.

Command	Description
COMMAND_DISCOVER	Request target
COMMAND_ASSIGN	Assign address
COMMAND_GET_INTERRUPT	Get interrupt status
COMMAND_CLEAR_INTERRUPT	Clear interrupt status
COMMAND_READ_SENSOR	Read sensor data
COMMAND_SEND_RADIO	Send radio data
COMMAND_STORE_DATA	Request save data
COMMAND_REQUEST_DEVICE_TYPE	Request device type

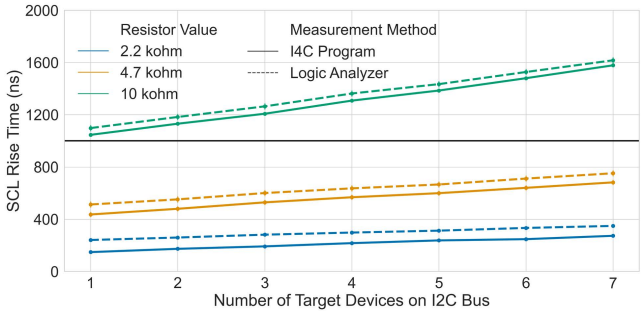


Figure 9: I4C program measured SCL rise times compared with SCL rise times measured with a logic analyzer for ground truth. The specifications maximum rise time is notated at 1000 ns. As the number of devices increase, we see an increase in the measured rise times, showing that we are able to accurately measure rise time using I4C.

5.2.2 Communication Scheme with a Controller. Our I4C implementation includes a set of application level commands that are used to interact with device discovery, address assignment, data processing, and interrupt operations. The process begins when a controller writes commands to a target device. After receiving a command, a target will wait until the next I²C stop condition occurs before it processes the command. Table 1 lists the set of I4C commands. When a rise time change is detected, the discovery, address assignment, and device type request commands are invoked to identify new targets and assign them addresses. We also implement commands for receiving and clearing interrupt information. For our application demo, we implement commands to read sensor data, store data, and send data over the radio. These commands demonstrate the basic I4C primitives, but additional commands could be added to satisfy a variety of other application needs.

5.3 System Integration

Controller and target RP2040 devices, programmed for I4C communication, can co-exist with our custom I4C devices to form a full system. Testbed implementation includes comparators and switchable pull-up resistor arrays on the I4C controller PCB, along with a resistor modulation circuit on the I4C target PCB. The full hardware testbed is shown in Figure 6.

Bus recovery is an important I4C feature for handling errors or misconfiguration. I4C detects abnormal behavior, such as a sudden voltage spike or drop on the line, and can issue a retransmit. I4C can also detect if the system is operating outside of the I²C rise time specification due to extra bus capacitance or misconfigured resistors. A common failure mode in I²C is that the bus becomes stuck low, which our implementation automatically detects using a timeout. If a stuck-bus is detected, I4C attempts to recover the bus by setting the lowest possible resistance to pull-up the bus and then retransmitting any failed transactions.

6 Evaluation

In this section, we use our prototype implementation to evaluate our claims and characterize the performance of the I4C design.

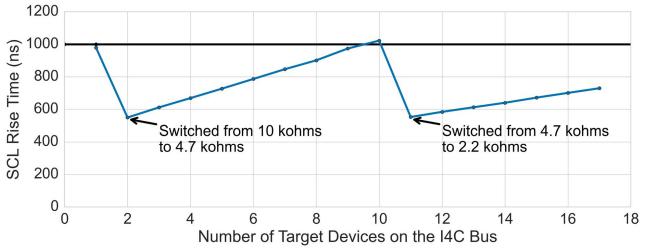


Figure 10: Program measured I4C SCL rise times. The I4C program keeps the rise time below, or close to the I²C specification of 1,000 ns. When the number of devices reaches 2 and 11 respectively, the program picks a lower resistor to lower the rise time. This shows I4C can dynamically adjust the rise time as the number of devices changes.

6.1 Measurement Accuracy

We validate that our system is able to accurately measure rise times across a range of different conditions. We measured with three different pull-up resistance values on our testbed. For each resistance, we varied the number of targets on the bus from one to seven devices, with one controller device. We used our I4C system to measure the rise time, and also captured the analog waveforms using a Saleae logic analyzer with a 50 MS/s sampling rate.

Our I4C implementation uses comparators to trigger the high and low voltage thresholds. Our low voltage threshold was 0.97 V and our high voltage threshold was 2.25 V, approximating the 30% to 70% rise time for a V_{dd} of 3.3 V. We then recorded over 1,000 SCL rising events using a logic analyzer. By post-processing the waveform and measuring the time between the signals crossing the low and high thresholds, we established a ground truth for the signal rise time behavior. In parallel, our I4C program measures the rise time, with a resolution of 40 ns. Figure 9 compares the I4C program's measured rise time with the post-processed ground truth data. We measured rise times both below and above the maximum rise time specified in the I²C standard and found that the I4C program accurately tracked the rise time trend across the entire range of test points that we evaluated.

6.2 Dynamism in Action

To test rise time adjustment, we configured our bus with 4 different resistor choices and let the I4C controller pick the appropriate resistor to keep the rise time close to the maximum allowed by the I²C standard. We set the I4C program to keep the rise time below the specified maximum of 1,000 ns, while varying the number of devices on the bus from 1 to 17. When the capacitance increases, and therefore the rise time also increases, the program dynamically switches to a lower resistance value to keep the rise time within the specification. Figure 10 shows the rise time, measured using 30%-to-70% rise time on the Saleae logic analyzer, varying as more devices are added to the bus. When a 2nd device is added to the bus, the rise time drops, which corresponds to a change from 10 kΩ to 4.7 kΩ. When an 11th device is added to the bus, the rise time again becomes too high, and the I4C controller automatically readjusts the bus pull-up resistance.

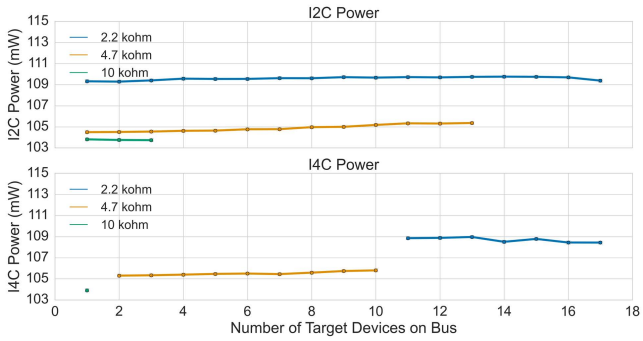


Figure 11: Power dissipation I²C vs I4C. Power dissipation is higher with smaller resistors because there is more current flowing through the resistors when the bus is driven low. This shows that I4C is able to dynamically utilize a larger resistor, saving power.

6.3 Power Savings

Another benefit of dynamic resistor adjustment is that it offers a chance to tune the power draw to better match the number of devices on the bus. We tested the power draw of the controller, with different pull-up resistor values, as the number of devices is varied. We used a bread board set up with 17 target devices and a Keithley 2401 source meter to measure the power draw of a single controller and circuits, including the processor, pull-up resistors, and rise time measurement circuit. To establish a representative I²C baseline power draw, we removed the comparator circuit and I4C code. Meanwhile, in the I4C system, there is a pull-up resistor in series with an analog multiplexer which increases the resistance and decreases power dissipation. I4C switches resistor values to remain at or below a 1,000 ns maximum rise time. Figure 11 compares the I4C power to I²C power. I²C data points are only shown for the number of devices which stay within the rise time specification of 1,000 ns. In a dynamic bus designed for 17 targets, if only one device is connected, I4C will use a 10 kΩ pull-up resistor while I²C uses a 2.2 kΩ one, which results in a savings of 5.4±0.2 mW.

6.4 Interrupt Timing

Another benefit of I4C is that target devices can send interrupts by varying the rise time. I²C itself lacks mechanisms to send interrupts at all, requiring controllers to poll periodically to check for target device requests. A common way to add interrupts to I²C is to add a third line—a shared interrupt (INT)—between all devices on the bus. When a target needs to signal an interrupt, it drives INT low. Since all the target devices share a single line, the controller knows a device is requesting interrupt service, but it does not know which one. Therefore, the controller has to check each device's interrupt status, usually in a round robin fashion, until the interrupt line is clear. For each device polled, the controller addresses the target, sends a request payload, and receives response payload from the target. This method both requires three signal lines, and interrupt service latency scales with the number of devices on the bus. The total communication overhead for polling a single device amounts to 27 bits, consisting of three data frames of 9 bits each. In contrast, the I4C protocol can identify interrupting devices immediately after decoding the modulated clock edges in the first data frame. For a scenario of 9 devices, I4C can achieve a 13.5× improvement in interrupt time compared to polling in the best case, and 27× improvement compared to the worst case, when the target is the last device accessed. Figure 12 captures a trace of an I4C target device sending an interrupt by modulating the rise time of SCL.

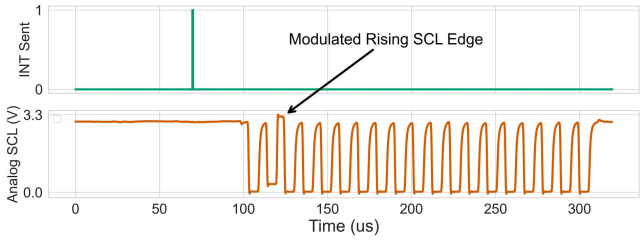


Figure 12: Interrupt signaling via rise time modulation. On the next I²C transaction, the target modulates the second rising edge of SCL to indicate an interrupt. The timeline shows that target devices are able to modulate the SCL rise time precisely and send interrupt signals to the controller.

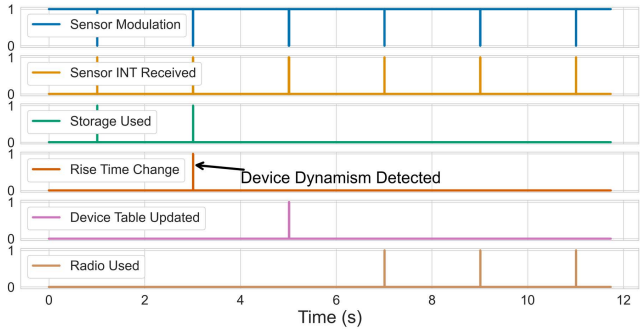


Figure 13: Digital waveform showing a "radio" device dynamically added to the bus. The I4C program detects a change in rise time, updates the device table, and then uses the preferred radio target instead of the "storage" target. This illustrates I4C's ability to dynamically detect and discover new devices using the rise time.

6.5 Putting it All Together

Motivated by one of our opening examples—daisy-chaining energy-harvesting energy meters in a circuit breaker panel and connecting them to a data logger or radio [22]—we now show how such an application with extreme dynamism could be supported by our I4C system. To test overall dynamism, we configure three Raspberry Pi Pico devices: a sensor, storage, and radio device. Our I4C program prioritizes the radio over local storage for data transmission. The sensor periodically sends in-band interrupts to the controller. At the beginning, the controller writes sensor data to the wired storage device. Afterward, we dynamically plug in a radio device, causing a change in rise time which is detected by I4C. I4C then runs ARP and service discovery and updates the device table. Once the device table is updated, our I4C program switches to using the radio instead of the storage device. This sequence is shown in Figure 13.

Table 2: I²C feature enhancements with I4C integration. We compare different combinations of I²C and I4C controllers and targets across key features: pull-up resistance adjustment, dynamism detection, dynamic device service management (*e.g.* ARP, service discovery, and resource binding), bus load monitoring, in-band interrupt, worst-case interrupt delay (9-device configuration), interrupt source ID, in-band full-duplex communication, and target-to-controller full-duplex initialization delay. "1:1" and "1:N" indicate the ratio of interrupt lines to total interrupt-capable devices (N). Parameter definitions: T_{tx_gap} (time until next I²C transaction), T_{byte} (duration of an 8-bit byte transaction), and T_{poll_gap} (time until next polling transaction, $\geq T_{tx\_gap}$). Additional I²C targets on the I4C bus operate as I²C baseline devices, without affecting I4C target device functionalities.

Controller	Target	Pull-Up Res. Adj.	Dynamism Detection	Dynamic Svc. Mgmt.	Bus Load Monitor	In-Band INT	WC INT Delay	INT Src ID	In-Band FD Comm	Target FD Init Delay
I4C Controller	I4C + 1:1 INT Ln	✓	✓	✓	✓	✓	0	✓	✓	0
	I4C + 1:N INT Ln	✓	✓	✓	✓	✓	T_{byte}	✓	✓	0
	I4C	✓	✓	✓	✓	✓	T_{tx_gap} + T_{byte}	✓	✓	T_{tx_gap}
	I2C + 1:1 INT Ln	✓	✓	-*	✓	×	0	✓	×	
	I2C + 1:N INT Ln	✓	✓	-*	✓	×	$27 * T_{byte}$	×	×	
	I2C	✓	✓	-*	✓	×	T_{poll_gap}+ $27 * T_{byte}$	×	×	
I2C Baseline		×	×	-*	×	×	T_{poll_gap}+ $27 * T_{byte}$	×	×	

* Depends on the type of I2C targets used (*e.g.* whether they support address assignment).

7 Discussion

In this section, we discuss I²C/I4C coexistence, I4C power overhead and potential mitigations, and other limitations and future work.

7.1 Legacy Device Support

I4C can accommodate both I²C devices and I4C-capable devices. While I4C targets benefit fully from I4C features, such as faster interrupts and dynamic addressing, legacy I²C device performance is still enhanced by I4C. Non-programmable legacy devices cannot generate interrupts via resistor modulation, but they can still be detected by an I4C controller. Programmable legacy devices can only do address assignment if their software supports ARP, but they can still be augmented to use an INT pin or GPIO to modulate rise time for in-band interrupts or full-duplex communication. Despite the limitations of legacy I²C devices, the core dynamism features of I4C are still retained, even in environments with only legacy targets. Table 2 presents a comprehensive overview of I4C features across different combinations of I²C and I4C controllers and targets.

7.2 Power Overhead

Our PIO-based implementation of I4C requires the use of a high frequency clock, resulting in power-inefficiencies. To address this challenge, we explore two low-power oriented design paths: leveraging more efficient high frequency clocks, and using a circuit design that eliminates the need for a high frequency clock.

7.2.1 High-Frequency Low-Power Clocks. Recent advances in oscillator design have enabled high-frequency clocks to operate with minimal power draw. For instance, Naing et al. developed 61 MHz oscillator that draws as little as 43 µW [36]. Wu et al. introduced a 204 MHz oscillator with a power draw of 47 µW [44], while Nelson et al. developed a 2 GHz oscillator that draws 22 µW [37]. These advances show the potential of integrating low-power, high-frequency clocks with off-the-shelf MCUs to perform rise time measurement.

7.2.2 Alternative Circuit Designs. For controllers not equipped with high-frequency clocks, a circuit-based solution could be used as an alternative to the PIO program. We introduce both an ADC-based concept, shown in Figure 14, and a comparator-based concept, shown in Figure 15, that could perform rise time measurement.

In the ADC-based circuit, rise time is converted to an analog voltage signal using a ramp generator. In the ramp generator, a switch connects a current source to a capacitor which is controlled by the output of the low and high threshold voltage comparators on SCL. The capacitor voltage reflects the rise time measurement and is the output of the ramp generator, which is passed to an ADC and read using the SPI input of a microcontroller.

The comparator-based circuit uses a similar ramp generator approach, but also includes three comparator circuits, with three references voltages V_{th1}, V_{th2}, V_{th3}. These threshold voltages correspond to the minimum acceptable rise time, maximum acceptable rise time, and rise time threshold for modulation detection, respectively. The minimum and maximum rise time comparator (Comp 1 and Comp 2) outputs are connected to a flip-flop triggered at the first rising clock event of an I²C transaction, ensuring that only the unmodulated first rise time is measured and fed to the microcontroller. The third comparator circuit (Comp 3) compares V_{th3} with bus voltage to detect the rise-time modulation imposed by any device. It stores the modulation information in a shift register, which will be read by the microcontroller.

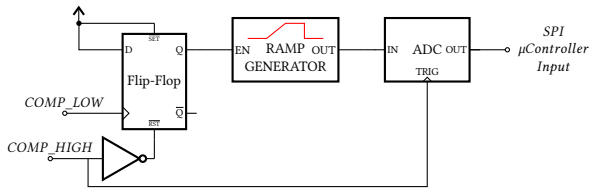


Figure 14: ADC-based rise time measurement circuit. The digitalized rise time is captured and transferred to the microcontroller via a high-speed communication protocol such as SPI. This circuit could act as a low-power solution for rise time measurement, without relying on low-power high-frequency clocks.

The controller uses a multiplexer to set the shift register's mode (read/write) and corresponding clock source—SCL for writing and controller clock for reading. However, because there is no mechanism for the shift register to signal whether it is full, the controller will not be able to read until completing the transaction, limiting modulation status detection to the last 8 clock edges. To address this challenge, a modification to the I4C protocol is needed—disabling full-duplex communication and requiring interrupting devices to modulate the corresponding edge once every byte instead of every transaction. Despite sacrificing the full-duplex interrupt feature, the comparator-based approach can still directly operate with GPIO pins on a microcontroller.

We estimate the power draw of both the ADC- and comparator-based approach using datasheet specifications of potential components [3–6, 9, 12]. Both approaches are estimated to draw roughly 400 μW. This is approximately two orders of magnitude lower than the power draw of the RP2040 when used for rise time sensing and one order of magnitude lower than the power savings achieved through pull-up resistance adjustments presented in the evaluation. We expect that the power draw would be even lower if the circuits were integrated on a chip.

7.3 Limitations and Future Work

I4C and our implementation of it have a number of limitations as discussed in this section. One major one is the small scale of our testbed, which limits the number of devices we could test.

Implementation Anomalies. The RP2040 stretches the frequency of the bus as rise time increases, which could affect power draw and bandwidth with larger rise times. The rise time noise did not pose a problem in our tests, but with other resistor values (*e.g.* larger modulation resistor) and in larger systems, rise time noise could vary.

Scalability. We chose a relatively small modulation resistor to ensure that the SCL rise time was sufficiently modulated, but there is a trade-off with the number of devices which can be assigned to modulate the edge. Many devices toggling small resistors in parallel may prevent other devices from pulling the bus low, given I^{2}C's 3 mA current sink limit, which could break communications.

Cost of Discretes. I4C requires modifications to the I^{2}C system in terms of additional circuitry, which adds cost and space to the system, although the number of discrete components is relatively small. However, if cost and space were to be sufficient concerns, this functionality could be incorporated into integrated circuits.

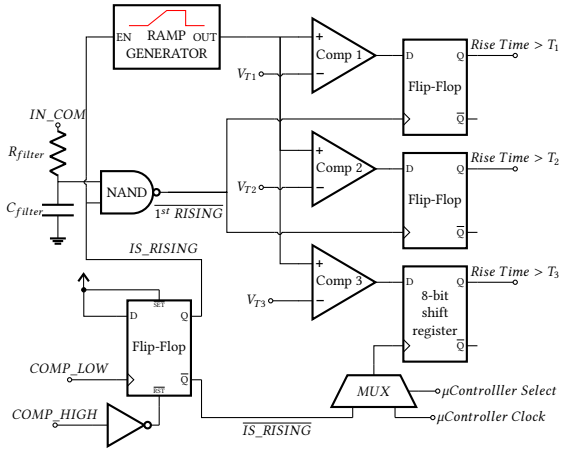


Figure 15: Comparator-based rise time measurement circuit. Three comparators capture bus device dynamism and clock edge modulation status, which is parsed by the microcontroller. This circuit could act as a low-power solution for rise time measurement, without relying on low-power high-frequency clocks or SPI.

Standard Speed. In this paper, we only demonstrate that I4C works up to the standard speed of an I^{2}C bus (100 kHz). In the future, we could use the rise time values from multiple resistors to gain more information about the bus capacitance, or measure internal pull-up devices on target devices.

Fault Tolerance. When many devices are in the same system, we assign unique addresses, but this does not prevent misconfigured or malicious devices from conflicting; we could incorporate an abort process in the case of conflicts, or implement orthogonal codes to handle multiple simultaneous transmissions.

8 Conclusion

We foresee a future in which I^{2}C is enhanced to support dynamic systems. With this in mind, we design and implement I4C, which leverages rise time measurement, automatic pull-up adjustment, and rise time modulation to enable dynamism and power efficiency in I^{2}C based systems. Building on rise time measurement and modulation, we also demonstrate automatic device detection, address assignment, and service discovery. With a small amount of logic on the target device, I4C can reduce interrupt latency by roughly 30× vs I^{2}C. And with its I^{2}C-compatibility, anyone can compare I4C against their legacy I^{2}C system, and even get SPI-like full-duplex with little additional cost and full invisibility to I^{2}C operations!

Acknowledgments

We would like to thank the anonymous reviewers and our shepherd for their insightful comments and feedback, Edward Lee for brainstorming and support, Lab11 members for their helpful comments, and collaborators who helped make this work possible, including Robert Shi for assisting with software implementation, Jiahui Cheng for contributing to hardware development, and Vivian Chan for assisting with testing. This material is based upon work supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the award number DE-EE0008220.

References

[1] 2003. TI 74HCT163N. https://www.ti.com/product/CD74HCT163.

[2] 2008. Analog Multiplexer DG409. https://www.analog.com/media/en/technical-documentation/data-sheets/DG408-DG409.pdf.

[3] 2010. SN74AUP2G00 Datasheet. https://www.ti.com/product/SN74AUP2G00.

[4] 2014. LT6020 Datasheet. https://www.analog.com/en/products/lt6020.html#documentation.

[5] 2015. ADS7040 Ultra-Low-Power, Ultra-Small Size, 8-Bit, 1-MSPS, SARADC. https://www.ti.com/lit/ds/symlink/ads7040.pdf.

[6] 2015. MAX6018 Datasheet. https://www.analog.com/en/products/max6018.html.

[7] 2018. What is TWI? How to Configure the TWI for I2C Communication. ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/SupportingCollateral/90003181A.pdf.

[8] 2019. Adafruit STEMMA. https://learn.adafruit.com/introducing-adafruit-stemma-qt/what-is-stemma.

[9] 2019. TI SN74HCS74 Datasheet. https://www.ti.com/product/SN74HCS74.

[10] 2020. TCA9511A. https://www.ti.com/product/TCA9511A.

[11] 2021. I2C Bus Specification Rev. 7.0. https://www.nxp.com/docs/en/user-guide/UM10204.pdf.

[12] 2023. TLV7032-Q1 Datasheet. https://www.ti.com/product/TLV7032-Q1.

[13] 2024. MIPI I3C and I3C Basic. https://www.mipi.org/specifications/i3c-sensor-specification.

[14] 2024. Pi Pico 2. https://www.raspberrypi.com/products/raspberry-pi-pico-2/.

[15] 2024. PMBus. https://pmbus.org/about-pmbus/.

[16] 2024. SMBus. https://www.smbus.org/.

[17] 2024. Sparkfun Qwiic. https://www.sparkfun.com/qwiic.

[18] 2024. Understanding EDID - Extended Display Identification Data. www.extron.com/article/uedid.

[19] 2024. Useful Sensors Person Sensor. https://www.sparkfun.com/products/21231.

[20] Joshua Adkins, Bradford Campbell, Branden Ghena, Neal Jackson, Pat Pannuto, and Prabal Dutta. 2016. The Signpost Network: Demo Abstract. *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems* (2016).

[21] Gary Berline. 1993. Ports of ACCESS. *PC Magazine* (1993).

[22] Campbell Bradford, Ye-Sheng Kuo, and Prabal Dutta. 2018. From Energy Audits to Monitoring Megawatt Loads: A Flexible and Deployable Power Metering System. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI) (IoTDI'18)*. 189–200. doi:10.1109/IoTDI.2018.00027

[23] Weijia Cai, Le Zhang, Lei Huang, Xinran Yu, and Zhengbo Zou. 2022. TEA-bot: a thermography enabled autonomous robot for detecting thermal leaks of HVAC systems in ceilings. *Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation* (2022).

[24] Prabal Dutta, Jay Taneja, Jaein Jeong, Xiaofan Fred Jiang, and David E. Culler. 2008. A building block approach to sensornet systems. In *ACM International Conference on Embedded Networked Sensor Systems*.

[25] Nicholas Gerard Edmonds, Douglas Stark, and Jesse M. Davis. 2005. MASS: modular architecture for sensor systems. *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.* (2005), 393–397.

[26] Nathan Farrington, Alex Forencich, George Porter, P-C Sun, Joseph E Ford, Yeshaiahu Fainman, George C Papen, and Amin Vahdat. 2013. A multiport microsecond optical circuit switch for data center networking. *IEEE Photonics Technology Letters* 25, 16 (2013), 1589–1592.

[27] Birte Friesel and Olaf Spinczyk. 2019. I2C considered wasteful: saving energy with host-controlled pull-up resistors: poster abstract. *Proceedings of the 18th International Conference on Information Processing in Sensor Networks* (2019).

[28] Alvin Jacob, Wan Nurshazwani Wan Zakaria, and Mohd Razali Md Tomari. 2016. Evaluation of I2C communication protocol in development of modular controller boards. *ARPN Journal of Engineering and Applied Sciences* 11, 8, 4991–4996.

[29] Chuanwen Lin, Gang Chen, Songsong He, and Linghong Chi. 2020. Overview of Space Plug-and-Play Avionics Interface Technology. *2020 International Conference on Computer Engineering and Application (ICCEA)* (2020), 34–38.

[30] Xinjian Liu, Anjali Agrawal, Akiyoshi Tanaka, and Benton H. Calhoun. 2024. Distributed Energy Harvesting and Power Management Units for Self-Powered In-Fabric Sensing Networks. *2024 IEEE 67th International Midwest Symposium on Circuits and Systems (MWSCAS)* (2024).

[31] Xinjian Liu, Zhenghong Chen, Nugaira Gahan Mim, Anjali Agrawal, and Benton H. Calhoun. 2023. A 1pJ/bit Bypass-SPI Interconnect Bus with I2C Conversion Capability and 2.3nW Standby Power for Fabric Sensing Networks. *2023 IEEE Biomedical Circuits and Systems Conference (BioCAS)* (2023), 1–5.

[32] James Lyke, Jesse K. Mee, Fredrik C. Bruhn, Gael Chosson, Robert Lindegren, Henrik Lofgren, Jan Schulte, Scott R. Cannon, Jacob Christensen, Bryan Hansen, Robin Vick, Alonzo Vera, and Josette Calixte-Rosengren. 2010. A Plug-and-play Approach Based on the I2C Standard. *24th AIAA/USU Conference on Small Satellites*.

[33] Dimitrios Lymberopoulos, Bodhi Priyantha, and Feng Zhao. 2007. mPlatform: A Reconfigurable Architecture and Efficient Data Sharing Mechanism for Modular Sensor Nodes. *2007 6th International Symposium on Information Processing in Sensor Networks* (2007), 128–137.

[34] Konstantin Mikhaylov, Juha Petäjäjärvi, Marko Mäkeläinen, Anton Paatelma, and Tuomo Hänninen. 2015. Demo: Modular Multi-radio Wireless Sensor Platform for IoT Trials with Plug&Play Module Connection. *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking* (2015).

[35] Konstantin Mikhaylov, Tomi Pitkäaho, and Jouni K. Tervonen. 2013. Plug-and-play mechanism for plain transducers with wired digital interfaces attached to wireless sensor network nodes. *Int. J. Sens. Networks* 14 (2013), 50–63.

[36] Thura Lin Naing, Tristan O Rocheleau, Elad Alon, and Clark T-C Nguyen. 2020. Low-power MEMS-based pierce oscillator using a 61-MHz capacitive-gap disk resonator. *IEEE transactions on ultrasonics, ferroelectrics, and frequency control* 67, 7 (2020), 1377–1391.

[37] Andrew Nelson, Julie Hu, Jyrki Kaitila, Richard Ruby, and Brian Otis. 2011. A 22μw, 2.0 GHz fbar oscillator. In *2011 IEEE Radio Frequency Integrated Circuits Symposium*. IEEE, 1–4.

[38] Kevin Nelson and Kamran Mohseni. 2017. An artificial fish lateral line sensory system composed of modular pressure sensor blocks. *2017 IEEE International Conference on Robotics and Automation (ICRA)* (2017), 4914–4919.

[39] Duy Nguyen. 2023. TI I2C Solutions for Hot Swap Applications. https://www.ti.com/lit/an/scpa058a/scpa058a.pdf.

[40] Akihito Noda and Hiroyuki Shinoda. 2018. Inter-IC for Wearables (I2We): Power and Data Transfer Over Double-Sided Conductive Textile. *IEEE Transactions on Biomedical Circuits and Systems* 13 (2018).

[41] Olivier Pieters, Emiel Deprost, Jonas Van Der Donckt, Lore Brosens, Pieter Sanczuk, Pieter Vangansbeke, Tom De Swaef, Pieter De Frenne, and Francis Wyffels. 2021. MIRRA: A Modular and Cost-Effective Microclimate Monitoring System for Real-Time Remote Applications. *Sensors (Basel, Switzerland)* 21 (2021).

[42] William Richards, John Selker, and Chet Udell. 2024. Loom: A Modular Open-Source Approach to Rapidly Produce Sensor, Actuator, Datalogger Systems. *Sensors (Basel, Switzerland)* 24 (2024).

[43] Yitian Shao, Hui Hu, and Yon Visell. 2019. A Wearable Tactile Sensor Array for Large Area Remote Vibration Sensing in the Hand. *IEEE Sensors Journal* 20 (2019), 6612–6623.

[44] Xiaotie Wu, Chengjie Zuo, Milin Zhang, Jan Van der Spiegel, and Gianluca Piazza. 2013. A 47μW 204MHz AlN Contour-Mode MEMS based tunable oscillator in 65nm CMOS. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1757–1760.

[45] Minghui Zhao, Stephen Xia, Jingping Nie, Kaiyuan Hou, Avik Dhupar, and Xiaofan Jiang. 2023. LegoSENSE: An Open and Modular Sensing Platform for Rapidly-Deployable IoT Applications. *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation* (2023).

[46] Josip Zidar, Ivan Aleksi, and Tomislav Matić. 2023. Analysis of energy consumption for SPI and I2C communications in ultra-low power embedded systems. *2023 46th MIPRO ICT and Electronics Convention (MIPRO)* (2023), 213–217.