10.48047/jocaaa.2025.34.10.16

A Comparative Overview of I2C and I3C Protocols for Server Platform Management and Diagnostics

Maheswara Kurapati
Independent Researcher, USA

Abstract

The development of server architectures requires better communication protocols that can handle the growing complexity of the platforms without compromising the power efficiency and bandwidth, among others. During the time since its introduction, Inter-Integrated Circuit (I2C) has been the stalwart of peripheral communication; however, the requirements of modern server platforms are far from what I2C requirements allow. Improved Inter-Integrated Circuit (I3C) is a transformative solution that provides significant performance advantages by providing push-pull signaling, dynamic address assignment, and in-band interrupt support, and is still compatible with legacy devices. Protocol transition covers the severe issues of data center infrastructure, where energy consumption, thermal control, and telemetry reactiveness have a direct influence on the performance. Server platforms using I3C have simpler board designs, higher polling rates, and lower standby power of dispersed sensor networks based on voltage regulators, temperature sensors, and power management units. Integration factors include electrical attributes, signal integrity limits, multi-domain power management, and firmware implementation plans based on Linux kernel interfaces and device tree configurations. I2C to I3C architectural evolution of embedded communication technology is a great enhancement in the field of embedded communication technology, and it allows the server technology to support the growing needs of performance as well as energy efficiency in a growingly heterogeneous computing environment.

Keywords: Inter-Integrated Circuit, Improved Inter-Integrated Circuit, Server Platform Management, Embedded Communication Protocols, Power Management Systems

Introduction

Servers have been developed to have more and more complex architectures, so more advanced communication protocols are required to support their more complex telemetry, component management, and diagnostic operations. Serial communication protocols, especially Inter-Integrated Circuit (I2C) and Improved Inter-Integrated Circuit (I3C), have become the technology of implementation of efficient device-to-device interaction in such systems. Although I2C has continued to be the de facto standard in low-speed peripheral communication since its inception by Philips Semiconductor in 1982, I3C is an alternative next-generation standard because of the increased bandwidth demands of modern server platforms, their more stringent power constraints, and a broader feature set. I3C was introduced by the MIPI Alliance to overcome the shortcomings of I2C-based implementations and provide backwards compatibility, as well as provide significant performance improvement to meet the needs of the current embedded system demands [1].

The background of this protocol development is not only technical requirements but also includes the general trends in industry in terms of power consumption and computational density. Power delivery and thermal management challenges are unprecedented in data center infrastructure that has undergone an explosion owing to the workload in cloud computing and artificial intelligence. As it stands, it is estimated that as of the year 2026, the electricity consumption of data centers may rise to anywhere

10.48047/jocaaa.2025.34.10.16

between 460 and 1,000 terawatt-hours of electricity usage in a year, or around 3.5 percent of the global electricity needs, due to artificial intelligence workloads, as well [2]. In this landscape, all the elements of the server architecture have to play roles in the optimization of efficiency, even the communication buses that connect management and monitoring subsystems. The classical I2C designs, with topology, current draw, and timing limits of the slow 3.4 Mbit/s open-drain implementation, put power penalties of constant pull-up resistor current and timing constraints that restricted polling frequencies of distributed sensors across server chassis.

I3C addresses these fundamental limitations through architectural innovations documented in the MIPI specification, achieving data throughput of 12.5 Mbit/s in Single Data Rate mode and extending to 33 Mbit/s in High Data Rate Double Data Rate configurations [1]. The protocol's push-pull signaling methodology eliminates static power consumption associated with pull-up resistors during active communication phases, while its integrated In-Band Interrupt capability removes the necessity for dedicated interrupt lines connecting dozens of peripheral devices to management controllers. For server platforms deploying 30 to 50 monitoring devices across voltage regulators, temperature sensors, fan controllers, and power management integrated circuits, these enhancements translate directly to reduced board complexity, lower standby power consumption, and improved telemetry responsiveness. The protocol's dynamic address assignment mechanism further simplifies system integration by eliminating the address conflicts that plague I2C deployments in complex multi-device topologies, particularly relevant as server designs incorporate increasing numbers of sensors to support predictive maintenance algorithms and real-time thermal optimization strategies responding to the variable power profiles characteristic of modern heterogeneous computing workloads [1][2].

10.48047/jocaaa.2025.34.10.16

Element	POSIX Specification	Linux Driver Implementation
Interface definition	System calls and library functions	ioctl commands and character devices
Portability approach	Standard API across UNIX systems	Kernel driver abstraction
Architecture support	Cross-platform specifications	Hardware-specific adaptations
I2C access method	File descriptor operations	I2C_RDWR and I2C_SLAVE ioctl
SPI access method	Standard file operations	SPI mode and frequency ioctl
Code organization	Portable C implementation	Layered driver architecture
Device tree integration	File system interface	fdt library API functions
Compiler compatibility	Standard C compliance	GCC and Clang toolchains
Memory management	Standard allocation functions	Kernel memory subsystems

Table 1: I2C and I3C Protocol Evolution and Energy Efficiency [1][2]
Legend: I2C = Inter-Integrated Circuit, I3C = Improved Inter-Integrated Circuit

2. Technical Architecture and Protocol Comparison

The fundamental architectural differences between I2C and I3C reflect the evolutionary trajectory of embedded communication requirements across more than four decades of technological advancement. I2C operates as a multi-master, multi-slave protocol utilizing a two-wire interface consisting of Serial Data (SDA) and Serial Clock (SCL) lines, with the protocol employing open-drain outputs with pull-up resistors that enable wired-AND bus arbitration through a deterministic collision detection mechanism where multiple masters monitor the bus state during transmission and withdraw when detecting conflicts [3]. The protocol's speed modes span multiple orders of magnitude, with Standard-mode operating at 100 kbit/s for basic sensor interfaces, Fast-mode supporting up to 400 kbit/s for more demanding applications, Fast-mode Plus reaching 1 Mbit/s for higher throughput requirements, and High-speed mode achieving 3.4 Mbit/s under specific conditions involving dedicated master codes and current-source pull-ups, though practical implementations in server platforms typically remain constrained to Fast-mode or Fast-mode Plus due to signal integrity limitations on printed circuit boards with trace lengths exceeding 20 centimeters and capacitive loading from multiple device connections [3]. The inherent limitations of I2C become particularly evident in power consumption metrics, where continuous current flow through pull-up resistors during idle high states and slow rise times resulting from RC time constants contribute to both static and dynamic power dissipation that scales unfavorably with bus capacitance and operating frequency [4].

I3C, standardized by the MIPI Alliance, represents a paradigm shift in serial bus design while maintaining backward compatibility with legacy I2C devices through sophisticated protocol detection mechanisms that automatically identify device capabilities during initialization sequences and seamlessly transition between I2C and I3C timing domains. The protocol introduces push-pull signaling on both SDA and SCL lines during I3C transfers, fundamentally transforming the electrical characteristics by actively driving high and low states with CMOS output stages rather than relying on passive pull-up resistors, which eliminates the continuous static current path and enables significantly faster edge rates with controlled slew characteristics that reduce electromagnetic interference [4]. I3C achieves substantially higher data

10.48047/jocaaa.2025.34.10.16

rates through multiple operational modes, with Single Data Rate (SDR) mode supporting up to 12.5 Mbit/s using conventional binary encoding, High Data Rate Double Data Rate (HDR-DDR) mode extending performance to 33 Mbit/s by sampling data on both rising and falling clock edges similar to DDR memory interfaces, and High Data Rate Ternary (HDR-Ternary) mode achieving even higher spectral efficiency through three-level signaling that encodes 1.58 bits per symbol transition [3][4]. These performance enhancements translate directly to improved system responsiveness in server platforms, enabling polling frequencies exceeding 1 kilohertz for distributed sensor networks compared to the 10-100 hertz rates typical of I2C implementations, thereby reducing control loop latency in thermal management systems that must respond to rapid transient power events characteristic of modern heterogeneous processor architectures combining general-purpose cores with specialized accelerators [3].

Bus arbitration and address management represent fundamental areas where I3C diverges from I2C design philosophy, addressing long-standing limitations in scalability and flexibility. I2C relies on static 7-bit or 10-bit addresses determined through hardware strapping pins or factory programming, with the 7-bit address space supporting 112 usable device addresses after reserving addresses for special purposes, including the general call address and 10-bit addressing prefixes [3]. This addressing scheme creates significant constraints in complex server platforms where identical sensor models from a single manufacturer share common default addresses, often necessitating external address translation circuits or hierarchical bus multiplexer topologies that increase component count, board area, and firmware complexity while introducing additional propagation delays in time-critical monitoring paths [4]. I3C fundamentally transforms this paradigm through dynamic address assignment orchestrated by the bus controller during initialization, where devices initially respond using provisional addresses derived from I2C-compatible static addresses or randomly generated values, then receive unique 7-bit dynamic addresses assigned by the controller after reading mandatory device characteristic registers that identify device type, capabilities, and required features [3][4].

Protocol Feature	I2C Architecture	I3C Architecture
Output Stage Design	Open-drain requiring an external pull-up	Push-pull CMOS output stages
Arbitration Method	Wired-AND collision detection	Dynamic address-based coordination
Address Allocation	Static hardware-determined values	Controller-orchestrated dynamic assignment
Rise Time Behavior	RC time constant limited	Actively controlled transitions
Multi-Master Support	Collision detection with withdrawal	Enhanced coordination mechanisms
Speed Mode Diversity	Multiple modes with different configurations	Unified modes with enhanced throughput
Encoding Schemes	Binary signaling only	Binary and ternary options
Power State Management	Limited native support	Integrated low-power modes

Table 2: Architectural Distinctions and Timing Characteristics [3][4]

Legend: CMOS = Complementary Metal-Oxide-Semiconductor, RC = Resistor-Capacitor

10.48047/jocaaa.2025.34.10.16

3. Application in Server Platform Management

Server platforms leverage I2C and I3C buses extensively for interfacing with diverse peripheral components essential to platform monitoring and control, with implementation complexity increasing proportionally as server architectures integrate specialized accelerators, high-bandwidth memory subsystems, and sophisticated power delivery networks. Temperature sensors distributed throughout the chassis provide thermal telemetry at critical monitoring points including processor die temperatures measured through digital thermal sensors with resolution of 1 degree Celsius, memory module thermal margins monitored via thermal sensor devices integrated into DIMM SPD EEPROMs, voltage regulator hot spots tracked through dedicated temperature monitoring integrated circuits, and ambient air temperature measurements at inlet and exhaust locations using thermistor-based or silicon bandgap sensor technologies [5]. These thermal monitoring systems enable real-time fan control algorithms that adjust cooling capacity based on instantaneous thermal loads, with modern baseboard management controllers implementing closed-loop control strategies that maintain processor junction temperatures within specifications while minimizing acoustic noise levels through variable-speed fan operation coordinated via I2C or SMBus interfaces operating at standard 100 kbit/s rates for compatibility with legacy thermal management infrastructure [6]. Voltage regulators communicate status information through standardized PMBus protocol implementations layered atop SMBus physical and electrical specifications, which themselves derive from I2C Fast-mode standards operating at 400 kbit/s maximum clock frequencies, enabling digital power management capabilities, including real-time voltage and current telemetry with 12-bit analog-to-digital converter resolution translating to millivolt-level voltage accuracy and milliampere-level current measurement precision across multi-phase buck converters delivering hundreds of amperes to processor and memory subsystems [6].

The deployment of these protocols in server environments extends beyond simple sensor interfaces to encompass sophisticated subsystems coordinating platform-level operations across multiple operational domains. Power management integrated circuits coordinate multi-rail power sequencing through SMBus commands conforming to PMBus specifications, orchestrating precise timing of voltage rail activation during boot sequences where processor core rails, input-output interface supplies, memory voltages, and peripheral power domains must energize in manufacturer-specified sequences with programmable delay intervals to prevent electrical stress conditions and ensure reliable system initialization [6]. Server platforms typically implement hierarchical SMBus topologies where a baseboard management controller serves as a master device communicating with multiple slave devices, including voltage regulator modules, hot-swap controllers managing redundant power supply insertion and removal, field-replaceable unit EEPROMs storing manufacturing data and failure history logs, and temperature monitoring devices distributed across critical thermal zones [5]. Fan controllers implement proportional-integral-derivative control algorithms processing temperature inputs from multiple sensor sources and generating pulse-width modulation outputs driving variable-speed cooling fans, with sophisticated implementations supporting acoustic optimization strategies that balance thermal requirements against noise constraints, particularly important in office environments where acoustic emissions specifications limit continuous sound pressure levels to 35-40 decibels at operator positions [5].

The transition from I2C to I3C in server platforms presents both opportunities and challenges influencing architectural decisions across successive product generations. The higher bandwidth of I3C operating at 12.5 Mbit/s in Single Data Rate mode proves advantageous when polling multiple sensors at elevated frequencies, reducing latency in telemetry collection loops and enabling more responsive thermal management algorithms that can react to transient thermal events within milliseconds rather than the tens

10.48047/jocaaa.2025.34.10.16

or hundreds of milliseconds characteristic of traditional I2C-based monitoring implementations [6]. The in-band interrupt capability inherent to I3C protocol architecture eliminates requirements for separate GPIO interrupt lines connecting slave devices to management controllers, simplifying board routing complexity in space-constrained server designs where printed circuit board real estate costs and connector pin count limitations significantly influence platform architecture decisions [5][6].

Device Category	Protocol Usage	Primary Function	Management Interface
Thermal Sensors	I2C/SMBus	Temperature monitoring	Digital sensor interfaces
Voltage Regulators	PMBus over SMBus	Power delivery control	Standardized power management
Memory Modules	I2C for SPD	Configuration and monitoring	EEPROM data access
Cooling Systems	I2C/SMBus	Fan speed regulation	PWM control interfaces
Management Controllers	SMBus master operation	Platform coordination	Multi-device communication
Power Supplies	SMBus with hot-swap	Redundant power management	Current monitoring and control

Table 3: Server Platform Device Integration [5][6]

4. Integration Challenges and System-Level Considerations

The integration of I2C and I3C buses into heterogeneous server architectures introduces several technical challenges that warrant careful consideration during system design and firmware development, with complexity scaling non-linearly as device counts increase and topology requirements expand to support distributed management subsystems. Bus topology complexity increases substantially in platforms featuring multiple management controllers operating as independent masters on shared bus segments, hot-plug capabilities enabling field-replaceable unit insertion and removal without system power cycling, and redundant communication paths implemented through bus multiplexers or dedicated backup interfaces to ensure continued operation during partial system failures [7]. Electrical characteristics such as capacitive loading, which accumulates from printed circuit board trace capacitance typically ranging from 1-2 picofarads per centimeter, component pin capacitance contributing 5-10 picofarads per device connection, and connector interface capacitance adding 2-5 picofarads per mated contact pair, must be meticulously analyzed to ensure that total bus capacitance remains within I2C specification limits of 400 picofarads for Fast-mode operation or I3C limits allowing 50 picofarads for optimal 12.5 Mbit/s performance [7]. Signal integrity concerns become particularly acute in server platforms where bus traces may extend 20-30 centimeters across motherboards connecting distributed sensor networks, with reflections from impedance discontinuities at connector interfaces and stub lengths from multi-drop topologies potentially causing signal distortion that violates timing specifications for setup and hold requirements, which mandate minimum 100 nanosecond data setup times before clock falling edges and 0 nanosecond hold times after clock transitions for I2C Fast-mode, tightening to 10 nanosecond setup requirements and 5 nanosecond hold times for I3C operating at maximum SDR data rates [8].

I3C's backward compatibility with I2C devices, while advantageous for gradual migration strategies enabling phased introduction of next-generation components without requiring complete platform

10.48047/jocaaa.2025.34.10.16

redesigns, imposes protocol-specific constraints that complicate both hardware and firmware implementations. The bus controller must accurately detect device types through analysis of response timing characteristics and acknowledge signaling patterns, then dynamically switch between I3C timing with push-pull transitions completing in 20-50 nanosecond edge rates and I2C timing with open-drain rise times extending 100-300 nanoseconds depending on bus capacitance and pull-up resistor values typically ranging from 1 kilohm to 10 kilohms [7]. Address assignment procedures require sophisticated orchestration, particularly in systems where devices may be added or removed during runtime through hot-plug operations, necessitating dynamic address allocation algorithms that track available addresses from the 112 usable values in the 7-bit address space after reserving addresses for broadcast commands, special function codes, and protocol-specific reserved addresses [8]. The dynamic address assignment feature, while eliminating static address conflicts that plague I2C implementations where identical device models share common factory-programmed addresses, necessitates robust address management algorithms implementing persistent storage mechanisms such as battery-backed SRAM maintaining 256 bytes to 1 kilobyte of address mapping tables or non-volatile EEPROM storage with write endurance specifications supporting 100,000 to 1,000,000 programming cycles to maintain consistent device identification across power cycles, warm resets, and firmware updates [7].

Power domain considerations add another layer of complexity, reflecting the aggressive power management strategies employed in modern server architectures, where idle power consumption directly impacts operational expenses in large-scale data center deployments. Server platforms typically partition devices across multiple power domains operating at voltages ranging from 1.2 volts for low-power sensors to 3.3 volts for legacy peripheral interfaces and 5.0 volts for certain fan controllers and hot-swap circuits, enabling selective component activation where non-critical monitoring functions may be disabled during low-power states while essential thermal management and fault detection capabilities remain active [8]. I2C/I3C buses spanning multiple domains require level shifters implementing bidirectional voltage translation with propagation delays typically contributing 5-15 nanoseconds per translation stage or bus buffers providing signal regeneration at domain boundaries with similar latency penalties that accumulate across multi-stage topologies, potentially totaling 50-100 nanoseconds end-to-end delay affecting maximum achievable bus speeds and protocol timing margins [7]. The management of these multi-domain topologies demands careful firmware coordination implementing state machines that sequence bus transactions around power domain transitions, prevent bus lockup conditions where devices in powered-down domains present high-impedance states that could be misinterpreted as stuck-low fault conditions, and handle communication failures gracefully when target devices become inaccessible during domain power-down sequences that may complete within 1-10 milliseconds depending on voltage regulator slew rate specifications [8].

Error handling and fault isolation represent critical aspects of robust bus implementation, distinguishing production-quality firmware from prototype implementations lacking the resilience required for long-term field deployment in mission-critical server environments. Firmware must implement comprehensive error detection including parity checking for protocols supporting it, timeout mechanisms with durations calibrated to device-specific maximum response times ranging from 10 milliseconds for simple sensors to 100 milliseconds for EEPROM write operations requiring internal charge pump stabilization, and recovery procedures implementing graduated response strategies that first attempt simple transaction retries with exponential backoff intervals doubling from initial 1 millisecond delays, then escalate to bus reset procedures toggling bus signals to break potential deadlock conditions, and ultimately trigger fault

10.48047/jocaaa.2025.34.10.16

isolation measures disabling problematic bus segments or devices to prevent single-point failures from disrupting entire management subsystems [7][8].

Design Consideration	I2C Requirements	I3C Requirements
Bus Capacitance Management	Limited to specification constraints	Optimized for higher speeds
Timing Margin Allocation	Conservative for open-drain operation	Tighter for push-pull operation
Multi-Domain Interfacing	Level shifters with voltage translation	Enhanced cross-domain capabilities
Address Space Management	Static allocation with potential conflicts	Dynamic assignment prevents conflicts
Hot-Plug Operation	Complex coordination required	Simplified through dynamic addressing
Signal Integrity	Sensitive to trace length and loading	Improved through active driving
Power Domain Coordination	Manual state machine implementation	Enhanced protocol-level support

Table 4: Integration and Electrical Implementation [7][8]

5. Linux User-Space Tools and Firmware Implementation

The Linux ecosystem provides a mature set of user-space tools and kernel interfaces for I2C and I3C device interaction, facilitating development, debugging, and runtime diagnostics through standardized command-line utilities and programmatic interfaces that have evolved throughout kernel development history. The i2c-tools package encompasses essential utilities for I2C bus management and device communication, with the i2cdetect utility scanning I2C buses to enumerate connected devices by probing addresses in the 7-bit address space, systematically issuing transactions and recording acknowledgment responses to identify populated locations, proving invaluable during board bring-up phases where engineers must verify physical connectivity before attempting software integration [9]. The i2cget and i2cset utilities enable direct register-level access for debugging and prototyping, with i2cget supporting read operations specifying bus number, device address, register offset, and optional data width parameters including byte mode for 8-bit registers and word mode reading 16-bit values with appropriate endianness handling, while i2cset allows writing configuration values to device registers supporting immediate mode for single-byte writes and block mode for multi-byte sequential transfers [9]. The i2cdump utility provides comprehensive register dumps spanning device address spaces, typically configured to read contiguous register blocks with output formatted in hexadecimal representation showing register addresses and corresponding data values in tabular format facilitating visual inspection and comparison against device datasheets during hardware validation activities [10].

For I3C devices, the Linux kernel provides evolving support through dedicated subsystems with subsequent enhancements continuing through development branches, though user-space tooling remains less mature compared to I2C counterparts due to the protocol's relatively recent standardization and licensing considerations that have limited widespread tool availability. The i3c-tools package, developed

10.48047/jocaaa.2025.34.10.16

primarily for Linux-based platforms, represents one of the few available user-space utility collections for I3C device interaction, providing functionality analogous to the established i2c-tools ecosystem [9]. The i3ctransfer utility serves as the I3C equivalent to i2ctransfer, enabling combined read-write transactions with I3C devices through command-line interfaces that specify dynamic addresses assigned during bus initialization, supporting both private I3C transfers utilizing push-pull signaling and legacy I2C transfers for backward-compatible device communication [10]. Unlike I2C tools with decades of refinement and comprehensive documentation across multiple Linux distributions, i3c-tools availability remains constrained to specific hardware platforms and kernel configurations, with limited distribution packaging and sparse documentation reflecting both the protocol's recent adoption and potential intellectual property considerations surrounding MIPI Alliance specifications that may restrict open-source tool development and distribution [9].

Device enumeration and communication interfaces follow patterns similar to I2C but accommodate I3C-specific features such as dynamic addressing where device addresses may change between boot cycles based on topology discovery sequences, in-band interrupts enabling slave-initiated communication without dedicated interrupt lines, and higher data rate mode capabilities requiring negotiation between master and slave devices during runtime [9]. The i3ctransfer tool supports these I3C-native features through command-line parameters specifying transfer modes including Single Data Rate operations at standard throughput levels and potential High Data Rate modes depending on controller and device capabilities, though practical implementations often require direct kernel driver interaction through sysfs interfaces or custom applications utilizing I3C subsystem APIs when user-space tools prove insufficient for complex initialization sequences or protocol-specific commands [10]. Firmware developers must navigate the developing state of I3C software infrastructure while maintaining compatibility with established I2C workflows, often requiring adaptation periods as driver support matures across kernel versions, with the limited availability of comprehensive user-space tools necessitating greater reliance on kernel-level debugging techniques including dynamic kernel logging, protocol analyzer hardware for bus-level signal inspection, and custom test applications built against I3C kernel APIs for validation scenarios beyond the capabilities of available command-line utilities [9][10].

Device tree configuration serves as the primary mechanism for describing I2C/I3C bus topologies to the Linux kernel through declarative device tree source files compiled into binary device tree blob format loaded during system initialization. These specifications define bus controllers with properties including register base addresses located in platform-specific memory-mapped regions, clock source assignments, interrupt line connections, and timing parameters such as clock frequency properties expressed in Hertz units for different speed modes, with I3C device tree bindings requiring additional properties specifying dynamic address assignment capabilities, in-band interrupt support, and high data rate mode enablement flags that distinguish I3C controllers from legacy I2C implementations [9]. Device nodes describe attached peripherals with mandatory properties including compatible strings identifying device type and driver binding, reg properties specifying provisional I2C-compatible static addresses used during initial enumeration before dynamic address assignment, and optional properties including interrupt specifications for legacy GPIO-based interrupts coexisting with in-band interrupt mechanisms, along with device-specific configuration parameters documented in kernel binding documentation that varies significantly between I2C devices with stable, well-documented bindings and I3C devices where binding specifications continue evolving alongside protocol adoption and driver maturation [10]. Proper device tree construction requires thorough understanding of both hardware capabilities and kernel driver expectations, with validation tools detecting syntax errors though semantic correctness requires runtime

10.48047/jocaaa.2025.34.10.16

testing, particularly critical for I3C configurations where improper timing parameters, incorrect dynamic addressing setup, or incompatible mixed I2C-I3C device combinations may result in bus initialization failures or intermittent communication errors difficult to diagnose without comprehensive protocol-level debugging capabilities [9].

Driver integration follows the Linux kernel's layered architecture implementing separation of concerns, with bus controller drivers managing low-level timing and signaling including clock generation, dynamic address assignment orchestration, in-band interrupt handling, and protocol mode switching between I3C and I2C timing domains for backward compatibility, while device-specific drivers provide abstracted interfaces for particular component types exposing standardized kernel APIs such as framework interfaces for sensors, voltage regulators, and input devices [10]. The I3C subsystem architecture differs from I2C in its handling of dynamic addressing and device lifecycle management, requiring controller drivers to implement ENTDAA (Enter Dynamic Address Assignment) command sequences, maintain dynamic address mapping tables across power cycles, and coordinate hot-plug operations where devices may join or leave the bus during runtime without requiring full bus re-initialization [9]. Best practices for robust bus communication encompass transaction atomicity preserved through appropriate kernel locking mechanisms preventing concurrent access from multiple contexts, timeout values calibrated to device-specific maximum response times accounting for both I2C legacy device delays and faster I3C response characteristics, error handling implementing logging with appropriate verbosity levels that distinguish between expected I2C slow response patterns and anomalous I3C timing violations, and power management integration coordinating between bus controller callbacks managing bus power states and device-specific handlers ensuring proper sequencing during system state transitions where I3C devices may enter ultra-low-power modes with sub-microampere standby currents between active communication phases [9][10].

Tool/Component	Functionality	Application Context
i2cdetect	Device enumeration	Board bring-up and validation
i2cget/i2cset	Register access	Debugging and prototyping
i2cdump	Memory visualization	Hardware validation
i3ctransfer	I3C transaction execution	Dynamic address communication
i3c-tools suite	I3C device interaction	Platform-specific BMC environments
Device Tree	Topology description	Kernel configuration
Bus Controller Drivers	Hardware abstraction	Low-level communication
Device Drivers	Component interfaces	Framework integration
Kernel Subsystems	Protocol support	I2C and I3C management

Table 5: Linux Development Tools and Driver Framework [9][10]

10.48047/jocaaa.2025.34.10.16

Conclusion

The comparative evaluation of I2C and I3C protocols reveals fundamental architectural differences that directly influence suitability for contemporary server platform management applications. I2C maintains relevance through extensive device ecosystem support, mature software infrastructure, and proven reliability across decades of deployment, yet faces inherent constraints in bandwidth, power efficiency, and address space scalability that increasingly limit applicability in modern server architectures. I3C addresses these fundamental limitations through architectural innovations, including push-pull signaling, eliminating static power consumption, dynamic address assignment, resolving address conflicts in complex multi-device topologies, and in-band interrupt capabilities, reducing pin count requirements while enhancing system responsiveness. The protocol achieves substantial performance improvements, enabling higher polling frequencies for distributed sensor networks, reduced latency in thermal management feedback loops, and simplified board designs through the elimination of dedicated interrupt lines. Issues to implementation include the management of backward compatibility where dynamic protocol switching is needed, electrical issues such as capacitive loading, signal integrity over very long trace lengths, multi-domain power management coordination, and software development to deal with changes in Linux kernel support of I3C subsystems. I2C to I3C server platform transition requires thoughtful consideration of such factors as the investments already made in existing infrastructure, availability of devices, performance needs, power limits, and the long-term plan of platform evolution. I3C has excellent benefits over the complexity of software integration and emerging software ecosystems when used in advanced server designs that focus on aggressive performance and efficiency goals. Legacy I2C implementations remain viable for incremental platform updates and cost-sensitive applications where proven validation efforts reduce development risk. The continued maturation of I3C device ecosystems, Linux kernel infrastructure, and industry adoption trajectories will influence protocol selection decisions across successive server generations. Proficiency in both protocols becomes essential for firmware engineers operating in server platform domains, as heterogeneous implementations combining I2C legacy devices with I3C components characterize transitional architectures bridging current and future platform requirements. The architectural evolution from I2C to I3C represents a significant milestone in embedded communication technology, enabling server platforms to address escalating computational density, thermal management complexity, and power efficiency requirements characteristic of contemporary data center infrastructure supporting cloud computing and artificial intelligence workloads.

10.48047/jocaaa.2025.34.10.16

References

[1] NXP Semiconductors, "INTRODUCTION TO MIPI I3C," 2022. [Online]. Available: https://www.nxp.com/docs/en/training-reference-material/TIP-1109-MIPI-I3C-CN-1.pdf

[2] McKinsey & Company, "How data centers and the energy sector can sate AI's hunger for power," 2024. [Online]. Available: https://www.mckinsey.com/industries/private-capital/our-insights/how-data-centers-and-the-energy-sector-can-sate-ais-hunger-for-power

[3] Anusha Mahale; Kariyappa B.S., "Architecture Analysis and Verification of I3C Protocol," IEEE, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8822121

[4] Srirenganayagi, et al., "Understanding of MIPI I3C protocol and usage for sensor, other communication interface applications," Maxvytech, 2018. [Online]. Available: https://maxvytech.com/blogs/document/understanding_of_mipi_i3c_white_paper_v0.95.pdf

[5] Intel Corporation, "Technical Product Specification for the Intel® Server Board M50CYP2SB Family," 2024. [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000058779/server-products/single-node-servers.html

[6] Jessica Hopkins, "How Do PMBus vs SMBus vs I2C Compare?" Total Phase, 2021. [Online]. Available: https://www.totalphase.com/blog/2021/01/how-pmbus-smbus-i2c-compare/

[7] NXP Semiconductors, "UM10204 I2C-bus specification and user manual," NXP User Manual, 2021. [Online]. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf

[8] ABHINAV BODDUPALLI, "DESIGN AND IMPLEMENTATION OF I2C BUS PROTOCOL ON FPGA USING VERILOG FOR EEPROM," International Research Journal of Engineering and Technology, 2018. [Online]. Available: https://www.iraj.in/journal/journal_file/journal_pdf/11-426-151815309396-100.pdf

[9] Jonathan Corbet, et al., "Linux Device Drivers, 3rd Edition," O'Reilly Media, 2005. [Online]. Available: https://www.oreilly.com/library/view/linux-device-drivers/0596005903/

[10] M.Rama Mohan, "Development of I2C Device Driver for Open Power Architecture A2O Processor," International Journal of Scientific Research in Engineering and Management, 2023. [Online]. Available: https://ijsrem.com/download/development-of-i2c-device-driver-for-open-power-architecture-a2o-processor/

[11] AspeedTech BMC, "i3c-tools - User-space tools for I3C," GitHub Repository. [Online]. Available: https://github.com/AspeedTech-BMC/i3c-tools