chips

Article

Open-Source FPGA Implementation of an I3C Controller

Jorge André Gastmaier Marques [1,2], Sergiu Arpadi [2] and Maximiliam Luppe [1,*]

[1] Department of Electrical and Computer Engineering, University of São Paulo, São Carlos 13566-590, SP, Brazil
[2] Analog Devices Inc., Wilmington, MA 01887, USA; sergiu.arpadi@analog.com
[*] Correspondence: maxluppe@sc.usp.br

Abstract: Multiple serial interfaces have emerged to meet system requirements across devices, ranging from slower-speed buses, such as I^{2}C, to high throughput serial interfaces, like JESD204. To address the need for a medium-speed protocol and to resolve I^{2}C shortcomings, the MIPI Alliance developed the I3C specification, which is a royalty-free next-generation version of I^{2}C with new features and backward compatibility. Since the MIPI Alliance's I3C work only includes the specifications, it depends on third-party vendors to develop their own cores according to the specifications. Only a few processing systems contain I3C Controllers, each with its own partial implementation of the specification, and there are no open-source controller cores. Thus, this work presents an open-source I3C Controller HDL framework that operates at the maximum specified SDR frequency and is compatible with the Linux kernel. Both the core and Linux kernel drivers are available under permissive open-source licenses. The solution is mostly aimed at development boards with Xilinx Zynq and Intel Cyclone SoC; nevertheless, the structure of the project allows it to be ported to other vendors and carriers.

Keywords: I3C; FPGA; HDL; IP; GNU/Linux

1. Introduction

The communication between a peripheral and a controller in an electronic circuit depends on the system requirements, such as the transmission rate, the ratio between transfer and idle times, and the distance between the devices on the bus. In simpler circuits, the controllers are concentrated within a single processing system; however, a more complex system might have more than one processing system and potentially more than one controller per bus.

The Inter-Integrated Circuit (I^{2}C) interface is sufficient for applications with slower transmission rate requirements and recurring idle times. Since it only uses two traces to communicate with multiple devices, its printed circuit board footprint is considerably smaller when compared with other protocols such as Serial Peripheral Interface (SPI). However, its shortcomings have become increasingly problematic with the performance expectations of new electronic projects and the release of I^{2}C-compatible parts.

With an ever-increasing catalog of parts available from multiple vendors, the fixed identifiers are not unique and often suffer from address collision. Also, if a part lacks the option to modify part of the identifier, for example, by pulling up a peripheral's pin to flip a bit of the address, one cannot connect the same part twice to the same I^{2}C bus.

The I^{2}C maximum rate of 3.4 Mbps in its high-speed mode [1] is insufficient in modern applications that require higher data rates. This rate is bounded by the step response of the open-drain electronic gate on the bus. Thus, higher frequencies could result in transient and undefined logic states when the signal is sampled at the clock edge.

check for updates

Received: 17 November 2024
Revised: 18 January 2025
Accepted: 23 January 2025
Published: 27 January 2025

Citation: Gastmaier Marques, J.A.; Arpadi, S.; Luppe, M. Open-Source FPGA Implementation of an I3C Controller. *Chips* **2025**, *4*, 6. https://doi.org/10.3390/chips4010006

With the intent of overcoming the limitations of the I²C interface, the Mobile Industry Processor Interface (MIPI) Alliance developed the Improved Inter-Integrated Circuit (I3C) interface. This interface defines long and unique identifiers per part and allows transmission rates up to 12.5 Mbps in Single Data Rate (SDR) mode on the grounds of the dynamic push–pull configuration.

Since the MIPI Alliance only defines the interface's specifications, it is up to vendors to implement their own Intellectual Property (IP) blocks. Currently, Silvaco, Cadence, and Synopsys supply paid I3C Controllers and Peripheral IP blocks that are implemented in a few parts and processing systems from NXP, ST, and Texas Instruments. A free I3C Peripheral IP is provided by NXP—a reduced variant of the IP distributed by Silvaco [2]. However, there is no free I3C Controller IP.

Therefore, an engineer who aims to use I3C parts is limited to a few processing systems available with I3C Controllers, which are not necessarily compatible with his project requirements. For example, the solutions from NXP [3], ST [4], and Texas Instruments [5] support only bare metal applications. A project requisite might be to run an operating system on the processing system. This scenario also hinders the adoption and improvement of the protocol, as the paid controllers available are provided with restrictive terms and often under non-disclosure agreements.

Several researchers have contributed to the understanding and validation of I3C, from design and implementation to verification and performance analysis.

Anusha et al. [6] focused on the development of a verification environment following the Universal Verification Methodology guidelines and template, which aims to validate the correct implementation of I3C. Their developed scoreboard seems to lack validation of timing requirements and protocol violations, instead focusing on the monitored data transfers.

Shreyash et al. [7] aimed to develop a platform following the same guidelines, providing more details on the test environment. The test environment was composed of one I3C Peripheral under test, and two agents—one I3C Controller and one I²C Peripheral—intended to mimic a mixed-bus operation. Its sequencer also seems to consider the timing of transactions, beyond the ordering presented by [6].

Gao et al. [8] aimed to address the fixed approach of [6,7] by proposing a flexible hardware architecture. They introduced a configurable structure in the hardware design paired with a software layer to modify the hardware architecture and functionality. Their inclusion of an error injection test to validate whether the target could recover according to protocol requirements is both acknowledged and cherished.

All contributions in [6–8] could be useful for the development of new I3C Peripherals. Ref. [7]'s explicit clarification of the agents and design under test is critical since the logic required to implement an I3C Controller and Peripheral are largely distinct.

Mario et al. [9] created a test environment using the hardware-in-the-loop approach to verify the legacy compatibility of the I3C Controller. Their attention to timing verification is appreciated since the I3C protocol relies heavily on timing to differentiate transfers and states. For example, I3C transfers are hidden from I²C devices by setting the clock's high period to be less than the I²C spike filter.

Yadhu et al. [10] proposed the design and implementation of an I3C Controller simulated and implemented using an AMD Xilinx Spartan 7 Field-programmable Gate Array (FPGA). The presented work stops at the simulation of the I3C Controller; however, it is not clear if the correctness of the implemented protocol was verified and lacks testing on hardware. The source of the implemented controller could not be retrieved for analysis.

An ongoing initiative by the Chips Alliance seeks to implement an open-source I3C Controller [11]. The proposed design requires a 500 MHz system clock, which is unsuitable for low-cost FPGAs [12], as these typically operate at system frequencies of up to 200 MHz, excluding hard cores such as high-speed transceivers. This limitation is illustrated by the use of an FPGA platform, the AMD Xilinx Zynq ZCU106, which is priced at USD 3234 [13,14].

Although there are ongoing efforts to develop open-source I3C Controllers, no finalized design targeting low-cost existing FPGAs has been successfully implemented and distributed. The lack of such a framework limits accessibility for further development and practical adoption—an unfortunate pitfall when considering that the specification is royalty-free.

This paper presents an open-source I3C Controller HDL framework that operates at the specified maximum SDR frequency and is compatible with the Linux kernel. This paper is organized as follows: Section 2 explores the electrical characteristics and features of the I3C protocol, followed by the development of the HDL implementation and the corresponding Linux driver. Section 3 briefly summarizes the simulated results, focusing on the experimental results with the controller deployed on an FPGA and communicating with an I3C test device. FPGA resource utilization is also provided and compared with that of an SPI solution for a similar use case. Finally, Section 4 concludes the paper with a summary of the accomplishments of the implemented design, highlights its shortcomings, and provides pointers for future improvements.

2. Materials and Methods

The development of the I3C Controller is divided into two major phases, as illustrated in the flowchart in Figure 1.

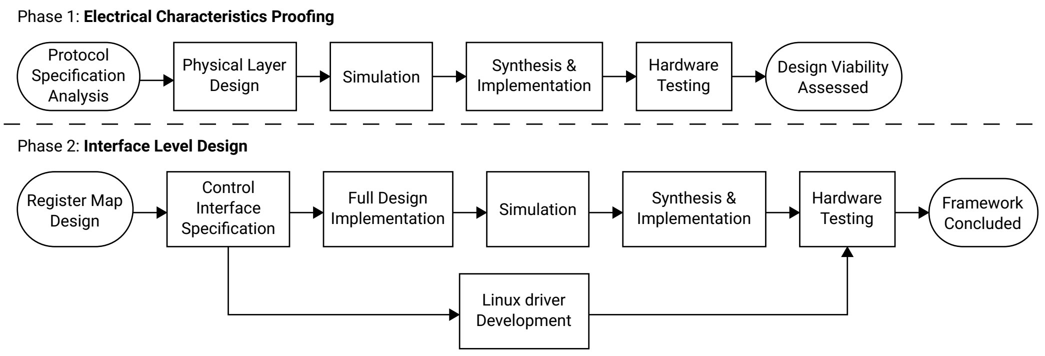

Phase 1: Electrical Characteristics Proofing
Protocol Specification Analysis
Physical Layer Design
Simulation
Synthesis & Implementation
Hardware Testing
Design Viability Assessed
Phase 2: Interface Level Design
Register Map Design
Control Interface Specification
Full Design Implementation
Simulation
Synthesis & Implementation
Hardware Testing
Framework Concluded
Linux driver Development

Figure 1. Development process of the I3C Controller.

In Phase 1, the focus was on validating the feasibility of implementing the controller on existing, low-cost FPGAs. This phase began with a protocol specification analysis, during which, the detailed protocol requirements and constraints were studied. The conclusions from this stage are detailed in the background section of Section 2.1 and provide a foundation for understanding the electrical characteristics and features of the I3C protocol, which guide the implementation decisions.

The phase continued with the Hardware Description Language (HDL) implementation of the core elements of the protocol, including the open-drain arbitration process, the transition to 12.5 MHz push–pull transfers, and backward compatibility with I^{2}C. Assessing the viability of the design at this stage was a critical decision point in determining whether to proceed to Phase 2.

In Phase 2, the protocol was further analyzed to identify key features for implementation, and the host interface was designed. This included the development of the register map and the control interface. A complete system design was created, incorporating the processing system, memory, buses, and auxiliary IP cores.

Along with the HDL development, a Linux driver was implemented, adhering to established abstractions and code quality standards. Both the protocol implementation and the Linux kernel I3C subsystem were carefully considered during the design of the host interface to ensure compatibility.

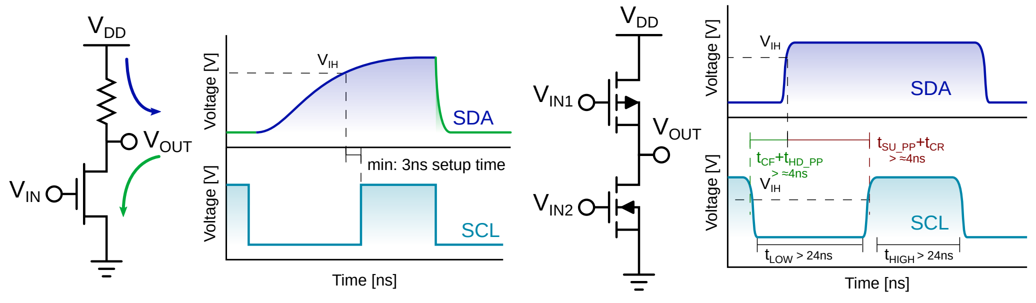
2.1. Background

The I3C protocol is meant to substitute I²C as the industry standard for mid-speed serial communication. This section explores its electrical characteristics and features, providing a basis for the desired implementation.

2.1.1. Standard Data Rate

To achieve higher speeds than its predecessor, the I3C interface specifies a clever combination of open-drain and push–pull modes that define the SDR.

The open-drain mode, Figure 2a, shares the same configuration as I²C. The logical high-level set bit is achieved using a weak pull-up resistor, and the logical low-level unset bit is created by a short to ground. This configuration allows multiple peripherals to safely drive the Serial Data (SDA) lane, allowing them to acknowledge transfers at the Acknowledge bit (ACK-bit) and send data on a single bidirectional lane. However, the weak pull-up has a high rise time, which limits the maximum frequency.



(**a**) Open-drain mode. (**b**) Push–pull mode.

Figure 2. I3C electronic configurations (schematics for illustrative purposes only).

To increase the maximum operating frequency, the I3C interface allows the SDA line to be switched from open-drain to push–pull mode. The push–pull mode uses a totem pole driver, Figure 2b, and the handoff occurs in a timed pattern to ensure that no other device attempts to drive the SDA line while it is being driven in push–pull mode, as this would result in a short circuit.

The specified minimum high and low clock states are 24 ns each; together with the minimum setup/hold (t_{HD_PP}/t_{SU_PP}) times and rise/falling times (t_{CR}/t_{CF}), the maximum operating frequency is 12.5 MHz.

The I3C specification disallows clock stretching, and only the active controller can drive the Serial Clock (SCL) line; this is advantageous in that the SCL line can be driven only in push–pull mode, eliminating the possibility of a bus hang due to a malformed SCL state, which is a common issue in I²C buses.

2.1.2. In-Band Interrupt

In-Band Interrupt (IBI) is a feature that allows peripherals to trigger interrupts to the I3C Controller. The priority level of the interrupt is encoded in the device address, with lower addresses having higher priority [15]. This is enabled by the arbitration process that occurs during the broadcast address phase of an I3C transfer.

The arbitration process works by having each device compare the bits transmitted during the broadcast address phase against their own addresses. At each address bit, devices wanting to send an interrupt transmit their addresses. If the current bit is unset and the device's own address bit is set, then the device has lost the arbitration to another device with a higher priority. The process continues until the broadcast address phase is complete, at which point the controller shall acknowledge the interrupt request and receive the Mandatory Data Byte (MDB).

The arbitration process occurs in two scenarios:

1. The controller initiates a transfer with a start, followed by the I3C broadcast address or a target device address; the interrupt sender drives the SDA with its own address.
2. The bus is in the bus-available condition and the interrupt issuer pulls the SDA line low; the controller notices, pulls the SCL line low (completing a start), and provides clock cycles for the peripheral to send its address.

Figure 3 illustrates an arbitration process in which three devices attempt to issue an IBI during the broadcast address phase. The controller initiates a transfer with a start condition.

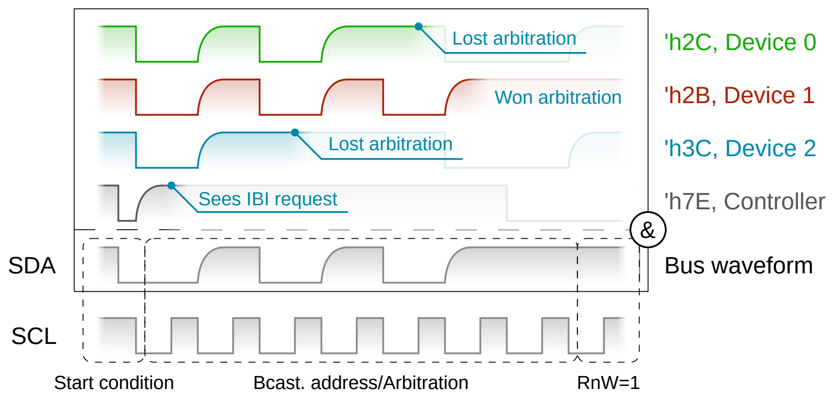


Figure 3. Representation of the arbitration process during the broadcast address phase.

Note that the arbitration process can only occur after a start; a peripheral cannot request an interrupt during a transfer, even after a repeated start.

The MDB contains values specified by MIPI, which are kept up to date in the I3C MDB Values Implementers table [16].

2.1.3. Dynamic Address Assignment

Unlike I²C, the I3C interface allows device discovery through a process called Dynamic Address Assignment (DAA) during bus initialization. The assigned addresses are the same length as in the I²C bus, 7 bits, but since they are allocated by the controller, there is no address collision as there is with I²C's fixed addresses.

The bus enters the DAA procedure when the controller issues an "ENTDAA" Common Command Code (CCC). It then sends the I3C broadcast address (0x7E), starting an arbitration process similar to the one shown in Figure 3, where each device that has not yet been assigned an address yields its Provisioned ID, bus characteristics register, and device characteristics register. The controller assigns a dynamic address and the peripheral that won the arbitration acknowledges it. The controller then repeats this process until no peripheral acknowledges the I3C broadcast address, indicating that every device on the bus has an address (dynamic or not), including itself [15].

Some phases of the DAA occur in push–pull mode, but since the speed grade of the peripherals is assumed to be unknown, it is reasonable to perform the entire procedure in open-drain mode.

In the scenario involving peripherals with fixed addresses, either I3C or I^{2}C, the controller must know them in advance; for a Linux implementation, this information is described in the devicetree [17].

2.1.4. Hot-Join

The Hot-Join mechanism allows devices to join the bus after it has been configured. A device issues an IBI with the reserved target address, but instead of transferring a data payload, the controller enters the DAA procedure and assigns a dynamic address to the device [15,18].

This feature exists to enable two use cases:

1. The device is connected to the bus and remains unpowered until it is needed. This is useful in applications where power usage management is critical.
2. The device is connected physically to the bus after the bus has been configured.

The second scenario is similar to enumeration within a Universal Serial Bus system when a new device is connected to the bus.

2.1.5. High Data Rate Modes

High data rate modes allow for higher throughput compared to the SDR mode. The basic specification [15] includes two modes:

1. Double data rate.
2. Bulk transfer.

The full specification includes two additional high data rate modes that are not discussed in this work.

In the double data rate mode, the data change on both clock edges, whereas in the SDR mode, the data change when SCL is unset and is sampled on the rising edge.

The bulk transfer mode extends the typical bus topology by leveraging multiple SDA lanes. It foresees quad and dual-lane configurations, but can also operate with a single lane. The transport form of bulk transfer is a pure byte stream.

2.2. Implementation

This section describes the development of the HDL implementation and the Linux solution for the project.

2.2.1. Project Structure

The project is divided into three publicly available version-controlled (`git`) repositories: “HDL reference designs”; “HDL testbenches”l and “Linux kernel”. A script-based workflow is used to call all relevant toolchains to synthesize, compile, package, and deploy the project to the development board.

2.2.2. Selected I3C Features

Considering that not all I3C features are mandatory to bring-up an I3C bus, essential features were selected to be implemented:

- Single Data Rate mode.
- In-Band Interrupt.
- Dynamic address.
- I^{2}C backward compatibility.

The objectives are to transfer data at the maximum rate allowed in the SDR mode (12.5 MHz), execute the DAA procedure, and handle IBIs.

The IBI feature is partially implemented, that is, only the MDB is polled by the controller (additional data bytes are not acknowledged). This decision was made to save FPGA resources since an additional memory queue would be required to store arbitrary-length IBI payloads.

High data rate modes, Hot-Join, and secondary mastership are not implemented.

2.2.3. Hardware and Tools

The main targets are Digilent Cora Z7s and Terasic DE10-Nano. Both contain FPGA and Advanced RISC Machine (ARM) processing systems in their System-on-a-Chip (SoC). The Cora Z7s features an AMD Xilinx Zynq-7000 [19], which integrates an Artix 7-based FPGA with a single-core ARM Cortex-A9 processing system [20]. The Terasic DE10-Nano features an Alera Cyclone [21] that integrates a Cyclone V SE FPGA with a dual-core ARM Cortex-A9 processing system [22].

Both development kits are low-cost, off-the-shelf solutions with good tooling and Linux support. The development tools used were AMD Vivado for the Zynq target and Altera Quartus for the Altera target. No paid or professional licenses are required for either SoC, making them end-user-accessible solutions.

ADALM2000 (Analog Devices Inc., Wilmington, MA, USA) was used as a digital oscilloscope. It is powered by an AD9963, a 10/12-bit low-power broadband mixed-signal front end, and a Zynq 7000 SoC [23].

2.2.4. HDL Implementation

The controller is divided into two IP blocks, consisting of five and three modules, respectively. The general controller architecture is illustrated in Figure 4, which shows the two IP blocks, namely, the host interface and core.

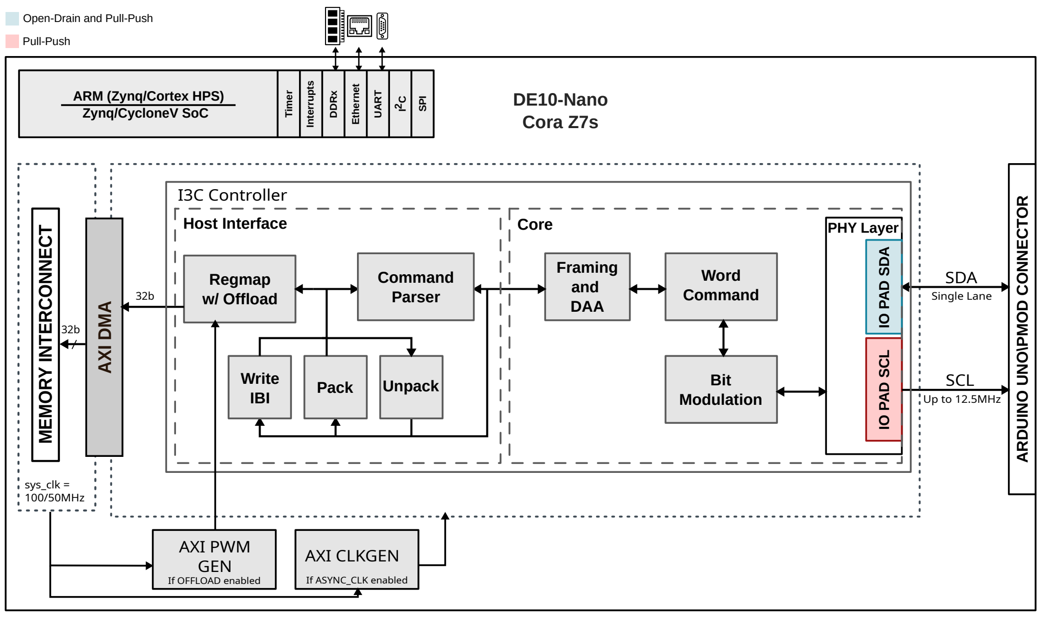


Figure 4. General controller architecture.

The host interface is the gateway between the processing system and the controller, managing internal registers, interrupts, and processing instructions. The core, in turn, receives parsed processed instructions and translates them into modulated I3C signals.

While the host interface contains almost generic logic designed to provide an instruction interface to the processing system, the core contains highly specialized logic designed to implement the I3C interface according to the specification.

The controller includes configurable parameters that allow for various features and configurations, which affect resource utilization.

Primarily, the asynchronous clock (`ASYNC_CLK`) parameter allows the controller to be driven by a dedicated clock source. The clock modifier (`CLK_MOD`) parameter tunes the internal logic to accept either 50 MHz or 100 MHz clock sources when operating at nominal bus speeds. The second is particularly useful for avoiding the consumption of FPGA clock resources such as phase-locked loops.

The offload (`OFFLOAD`) parameter enables a stream interface within the register map module. Conceptually, it is based on the SPI engine offload [24], but it uses fewer resources than the original implementation and is integrated into the register map module instead of requiring its own dedicated IP.

The offload functionality allows I3C commands and data streams to be stored and executed when a trigger signal is asserted. This enables the execution of I3C transactions with minimal delay in response to an event. A common use case involves providing a constant interval trigger using a Pulse-Width Modulation (PWM), as illustrated in Figure 4. For instance, it can be used to trigger and read back data from a sensor or peripheral connected to the I3C bus.

The stream interface works by setting up instructions into block random-access memory and then asserting a trigger to execute them. To ensure constant trigger intervals, the trigger source can be a pulse-width modulation signal. Since the instructions are written in advance of the transfer and are reused at every trigger, the interface allows minimal latency.

The stream interface is connected to a Direct Memory Access (DMA). The DMA used is the AXI DMAC IP, a high-speed DMA controller [25], and is configured to generate an interrupt when its buffer is full.

During the design phase, it is important to be aware of the resources available. Particular attention is paid to the bidirectional three-state switch required for open-drain and push–pull modulation, as described in the I3C specifications.

For the AMD Xilinx solution, the IOBUF (OBUFT + IBUF) design element implements a bidirectional tri-state buffer [26], but it is located in the device fabric logic instead of the Logic I/O, resulting in worse timing for the SDA and SCL lanes.

For comparison, IOB and IOB_TRI_REG are used to guarantee well-defined I/O timings; however, the first places a two-state register and the second is only available on more expensive UltraScale+ devices [27]. Similarly, for the Altera solution, the ALTIOBUF primitive is used, which is also a tri-state buffer [28].

To ensure a weak pull-up during the high impedance state, the PULLUP primitive was considered for both solutions. It establishes a set logic level for open-drain elements when all drivers are unset [26]. However, its response times and driver strengths are not adequate to maintain a high state. Instead, a passive pull-up resistor is used, undermining the power-saving benefits of the I3C bus.

The I3C bus clock frequency depends on the peripherals that may be attached to the bus, which is only determined during the driver's initialization process. Thus, it is a software-adjustable clock.

For the Zynq family, the mixed-mode clock manager clock source can be reconfigured by software using the dynamic reconfiguration port [29]. It could be connected directly to the processing system with an interface adapter, but this approach is not ideal because the configuration needs to be polled and would require a carrier-specific code in the device driver. It is preferable to control it with a state machine managed by register access instead. It is also possible to generate predefined clock sources with BUFR, and multiplex them with BUFGMUX [26].

However, it is not strictly necessary for the modulated SCL clock to be a clock network. Therefore, a more straightforward implementation can be achieved by modulating the SCL wave with logic registers instead. This approach was selected because it allows for the implementation of a software-adjustable clock in a fairly simple approach, resulting in frequencies of 12.5 MHz, 6.25 MHz, 3.125 MHz, and 1.56 MHz (the open-drain mode is locked at the lowest speed).

To exchange data between the cores within the I3C Controller and with the processing system, a control interface—composed of six stream interfaces similar to Advanced eXtensible Interface (AXI) streams, and two custom interfaces—is used. It carries instructions, synchronizes payloads, streams data, and provides access to registers.

The I3C Controller's main instruction set is simply referred to as "command" or `cmd` and is either a single or a pair of 32-bit instructions, depending on the contents of the first instruction (`cmd 0`).

The format tries to fit all corner cases of the I3C specification and is inspired by the GPL-licensed "mainland" Linux kernel drivers, especially the i3c-master-cdns.c [30] and spi-axi-spi-engine.c [31] Linux drivers. The intent is to reduce the friction from integrating the driver with the "mainland" Linux kernel source code. The structure of command 0 is shown in Table 1.

Table 1. Command 0 descriptor.

Name	Range	Description
	31:23	Reserved.
Is CCC	22:22	Indicate if it is a CCC transfer (1) or not (0).
Bcast. header	21:21	Include the broadcast header in private transfer (1) or not (0).
Sr	20:20	Yield a repeated start (1) or stop (0) at the end of the transfer.
Buffer length	19:08	Unsigned 12-bit payload length; direction depends on RnW-bit value.
DA	07:01	7-bit device address (does not care in broadcast mode).
RnW	00:00	It should retrieve data from the device (1) or not (0).

The CCC instructions define a CCC transfer; see Table 2. To start a CCC transfer, bit 22 of command 0 is set. Considering the DAA procedure and how the Linux I3C abstraction is defined, the controller enters the DAA procedure with the ENTDAA CCC instruction, even though the DAA procedure does not follow the same state flow as private and CCC transfers and requires specialized logic.

Table 2. Command 1 descriptor; CCC instruction.

Name	Range	Description
Type	07:07	Direct (1) or broadcast (0), except SETXTIME and VENDOR.
ID	06:00	CCC to transfer; identifier matches the payload to be sent in the bus.

The private transfer instruction sets a private transfer. To start a private transfer, bit 22 of command 0 is unset and command 1 is unused.

For each command descriptor (`cmd`) executed, a feedback descriptor named command receipts (`cmdr`) is created.

In the command receipt, the buffer length is updated with the number of bytes actually transferred, an incrementing sync field keeps track of the execution, and an error field is present to notify the processing system of the bus integrity.

The I3C specification defines errors CE0 to CE3; however, CE1 is optional and not implemented. CE0 occurs when an unexpected number of bytes is received by the controller during a CCC. CE2 and NACK_RESP are similar, but CE2 is restricted to the ACK-bit of the broadcast address (7'h7e), while the latter is for any other acknowledgment, even the read of the Parity bit (T-bit).

IBIs are autonomously accepted when the bus is available if the feature is enabled in the register map.

The accepted IBIs fill a dedicated first-in, first-out memory and generate an interrupt to the processing system if the interrupt bit is not masked.

Currently, only the MDB is supported and the controller will not acknowledge additional data bytes.

If the IBI feature is disabled, the controller will not acknowledge IBI requests. If enabled, the controller will acknowledge the IBI request and receive the MDB if the peripheral provides one. In both cases, if the request occurred during the header broadcast phase, the controller will proceed with the command transfer after the IBI request is resolved.

The full interface specification is available in Analog Devices Inc.'s HDL repository documentation [32].

To test the developed design, the Vivado and QuestaSim simulation tools are used. These tools allow simulation of the vendors' resources, providing rich debugging features, such as behavioral checks for the AXI bus. This design verification approach is the standard format for the Analog Devices Inc. testbench repository.

The simulation uses a gray box probing approach, as follows: internal state machines are used to wait for the desired test conditions, and the physical layer module's serial data out signal is overwritten to simulate the peripheral transferring data to the controller. Verilog header files are used to keep the testbench human-readable, for example, by using `‘CCC_ENTDAA` instead of its hexadecimal counterpart in the register write method.

2.2.5. Linux Solution

Table 3 provides the source files created to implement the controller and test device driver in the Linux kernel, followed by a brief description.

Table 3. New Linux kernel source files.

File	Description
adi-i3c-controller.c	I3C Controller driver; methods for the I3C Linux abstractions.
dev/dev_core.c	Test device driver, agnostic to the communication protocol.
dev/dev_i3c.c	Test device driver, specific to the I3C variant.
adi,adi-i3c-controller.yaml	I3C Controller driver documentation.
--dev-i3c.dts	devicetree file (template).

Even though this work focuses on I3C parts, the device driver is divided into "core" and "i3c" modules to meet modern Linux driver standards. The "core" module contains all methods that are agnostic to the communication protocol (for example, to compute a register value present in all part variants), and the "i3c" contains methods that are specific to the I3C variant, for example, to instantiate the device after it was allocated by the I3C Controller driver and provide the callback method executed when an IBI is received.

The Regmap abstraction is used to execute read and write register access on the target, regardless of the protocol [33].

The development is divided into two parts, namely, the controller driver and the peripheral driver. The controller driver must implement I3C Linux abstraction methods while being compatible with the instructions designed. The Linux abstraction is intended to create a unified interface that handles multiple controllers and reduces code repetition in the Linux kernel.

The integration process is done by generating uImage/zImage, BOOT.bin/u-boot.spf, and devicetree.dtb. First, a memory card is flashed with Kuiper Linux [34] (creates all partitions needed), and then the files are copied to the boot partition. If the image boots successfully in the hardware, the following deployments can easily be done via `scp` (copies files between hosts on a network) [35].

In the case of a kernel panic, `kgdb`, the Kernel GNU Debugger, is used to debug the exception. Since the text console and the debug session share the same serial port, agent-proxy is used. It allows the host to access both the text console and the debug session, for example, by using telnet and gdb, respectively.

From the user space, the part can be interacted with via libIIO [36]; debug signals can be added to the design using the Integrated Logic Analyzer and SignalTap to analyze the signals of the designs. For example, debug signals can be added to the paths of the SCL and SDA ports and a transfer from the user space can be triggered with a libIIO raw channel read.

3. Results and Discussion

The success of Phase 1 was confirmed after evaluating the results obtained from the hardware tests. Using ADALM2000 to acquire analog signals, it was shown that the HDL implementation, combined with the default passive weak pull-up resistor of 2.2 kΩ, achieved an open-drain rise time (90%) of 250 ns. This setup time allows operating at the maximum open-drain speed of 1.56 MHz.

Using the ADALM2000's digital logic analyzer, the signal integrity was verified at the maximum SDR data rate of 12.5 Mbps. It was confirmed that the passive weak pull-up on the lanes did not hinder the operating rate. It is worth noting that depending on the number of devices on the bus and the length of the traces, the system may require tuning to ensure signal integrity, similar to the adjustments commonly made for I^{2}C buses.

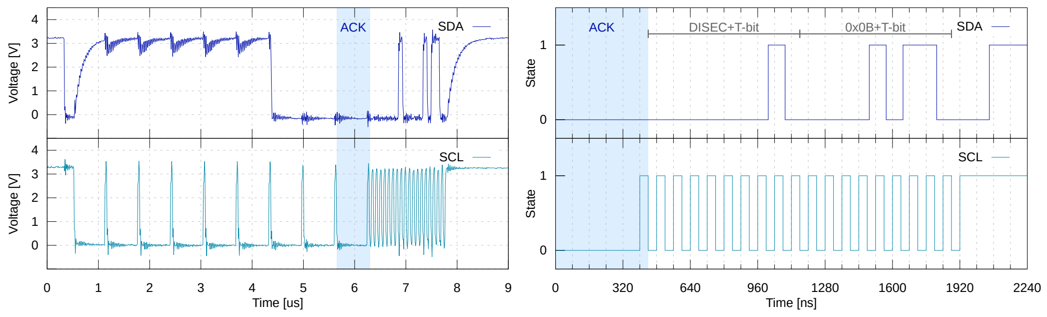
The primary component evaluated at this stage was the IOBUF, which demonstrated fast and consistent switching times. It successfully transitioned from an open-drain operation to the 12.5 Mbps push–pull mode, validating its suitability for the intended application.

For Phase 2, the first successful result obtained was that the controller passed all five implemented SystemVerilog testbenches, encompassing all features implemented, and identified corner cases of the specification. The testbench is critical for validating the logic before testing on hardware, and for protecting the design from regression as new features are integrated.

No simulated Linux target was developed to ease the Linux integration; after the testbench succeeded, it was deployed on hardware. Despite this, temporary tests were conducted on the driver to verify register access, such as logging the controller version and the assigned dynamic address. The values were logged using `printk`, a method to print messages to the Linux kernel log buffer, and visible via `dmesg`, a command to print and control the kernel ring buffer. The driver's functionality was accessed through kernel–user space application programming interfaces; debugging was conducted with `kgdb`.

The I3C test device was used to test the communication with the I3C Controller, successfully completing the DAA procedures, CCC transfers, and IBIs. SDR transfers are transmitted at the maximum rated speed of 12.5 MHz.

Figure 5a shows a sampled CCC transaction of the command "DISEC" (0x01). The waveform was acquired using an ADALM2000 at a sample rate of 100 Msps. After the controller sends the I3C broadcast address, the target successfully acknowledges the transfer at the ACK bit by driving the SDA lane low, and the controller starts transferring data in push–pull mode. The spikes on the SDA line during the open-drain section are caused by cross-talk introduced by the analog probe itself and are not present when sampling each lane individually. Additionally, as long as this capacitive effect does not trigger a false state during the sampling edge, it is considered safe.

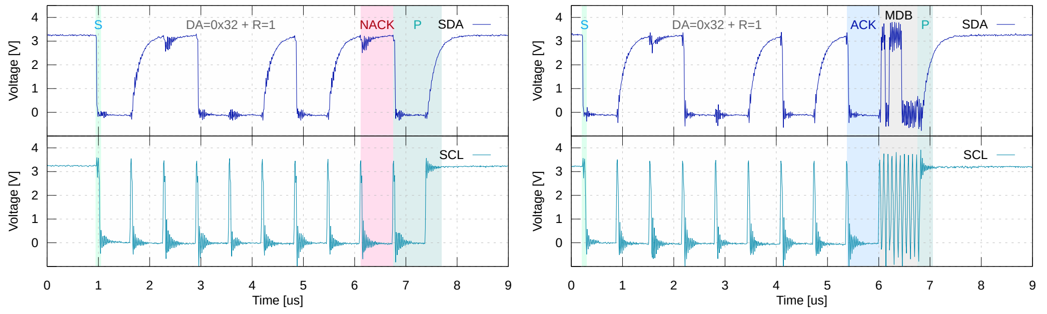


(**a**) Full transaction. (**b**) Push–pull section.

Figure 5. CCC DISEC 0x0B at 12.5 MHz.

Captured by the logic analyzer at 100 Msps, the push–pull section of the transfer is clearly shown in Figure 5b, where the controller sends two bytes at 12.5 MHz—the CCC command and the payload—both followed by the T-bit. By design, the setup time is at least 30 ns, always 10 ns after the SCL falling edge. And the hold time is at least 40 ns for the data bits, considering that the speed grade is configurable.

Figure 6a shows an IBI request not acknowledged by the controller; Figure 6b shows the same request acknowledged by the controller. The controller has prior knowledge that the I3C test device yields an MDB during IBI and, therefore, provides the clock cycle to receive this byte.



(**a**) Not acknowledged. (**b**) Acknowledged.

Figure 6. I3C IBI transaction, dynamic address = 0 × 32, bus characteristics register, bit 2 = 1.

Resource utilization for a common use case with DMA is shown in Table 4, where it is compared against a similar reference design using the SPI engine as the protocol [37]. The first column lists the total number of resources available on the Cora Z7s FPGA. The second and third columns show the number of resources utilized by each controller, followed by the corresponding percentages.

Table 4. Resource utilization (Cora Z7s with I3C + DMA vs. SPI + DMA).

Resource	Available Resources	I3C Utilization	SPI Utilization	I3C Utilization %	SPI Utilization %
Slice LUTs	14,400	1714	678	11.9%	4.7%
Slice Registers	28,800	677	899	2.3%	3.1%
Block RAM Tile	50	1.5	3.5	3%	7%
MMCME_ADV	2	0	1	0%	50%

The higher LUT usage in the I3C implementation is justified by the more complex state machines of the I3C protocol compared to SPI. A low FPGA utilization ensures shorter routing paths and better timing performance, leaving sufficient slack for users to implement additional logic alongside the controller.

4. Conclusions

The I3C Controller IP has been successfully implemented; it is able to communicate and evaluate I3C parts and be used to alter the protocol itself. The full-stack solution allows for rapid deployment and provides a starting point for users with any level of expertise, from high-level programming languages through libIIO language bindings to low-level Linux kernel and HDL development.

This work also proves that existing FPGAs are capable of deploying I3C devices; however, it is important to note that the power consumption improvement from I^{2}C to I3C is not measured in this work and it is not expected to be met; at the very least, the passive pull-up used on the bus would have to be replaced with an active one.

It is worth noting that the implemented solution does not foresee the I3C stall times beyond the DAA "First Bit of Assigned Address". The solution is conceptualized so that transfer instructions are fully written first and then executed, allowing for a precise estimation of the number of clock cycles required for a transfer, but it is acknowledged that this decision may be an issue depending on the user's needs.

Future work will include developing optional I3C features such as high data rate modes and Hot-Join, enhancing the resilience of the IP against malformed instructions, and implementing bus monitoring and the non-implemented stall times.

Author Contributions: Conceptualization, J.M.; methodology, J.A.G.M.; formal analysis, S.A.; writing—original draft preparation, J.A.G.M. and M.L.; review and editing, M.L. and S.A; supervision, M.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The source code for the developed I3C Controller, along with the commit SHA used in this paper, is available at the following repositories: HDL (library) (accessed on 26 January 2025): https://github.com/analogdevicesinc/hdl/tree/main/library/i3c_controller (9c5d04a70ba0225cc14d7f81c60376d1191df0e7) HDL (sample project) (accessed on 20 January 2025): https://github.com/gastmaier/hdl-i3c-bus/tree/i3c/projects/i3c_bus (d111174b5d29dcba7063fe88635f703c1cf73501) HDL testbench (accessed on 20 January 2025): https://github.com/analogdevicesinc/testbenches/blob/main/testbenches/ip/i3c_controller/tests/test_program.sv (bb7e91bdf71798e982355ca8a41d83f84b569ead) Linux driver (accessed on 20 January 2025): https://github.com/analogdevicesinc/linux/blob/i3c/drivers/i3c/master/adi-i3c-master.c (f6860a0fa9bdf36a269326553b849ba7a87defe6). The repository containing the sample project also contains the library; the sample project is not on the former HDL repository because it does not target a specific evaluation board or peripheral, as required. The HDL library documentation is available at https://analogdevicesinc.github.io/hdl/library/i3c_controller/index.html (accessed on 20 January 2025).

Conflicts of Interest: Authors Jorge André Gastmaier Marques, Sergiu Arpadi are employed by the company Analog Devices Inc.. Maximiliam Luppe declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACK-bit	acknowledge bit
ARM	Advanced RISC Machine
AXI	Advanced eXtensible Interface
CCC	common command code
DAA	dynamic address assignment
DMA	direct memory access
FPGA	field-programmable gate array
HDL	hardware description language
I^2C	inter-integrated circuit
I3C	improved inter-integrated circuit
IBI	in-band interrupt
IP	intellectual property
LUT	lookup table
MDB	mandatory data byte
MIPI	mobile industry processor interface
PID	provisioned ID
PWM	pulse-width modulation
RnW-bit	read-or-write bit
SCL	serial clock
SDA	serial data
SDR	single data rate
SoC	system-on-a-chip
SPI	serial peripheral interface
T-bit	parity bit.

References

1. I²C Quick Guide. Available online: https://www.analog.com/media/en/technical-documentation/product-selector-card/i2Cb.pdf (accessed on 12 October 2024).
2. MIPI I3C Slave. Available online: https://github.com/NXP/i3c-slave-design (accessed on 12 October 2024).
3. LPC553x. Available online: https://www.nxp.com/docs/en/data-sheet/LPC553x.pdf (accessed on 12 October 2024).
4. STM32H503xx. Available online: https://www.st.com/resource/en/datasheet/stm32h503eb.pdf (accessed on 12 October 2024).
5. DRA829 Jacinto Processors. Available online: https://www.ti.com/lit/ds/symlink/dra829v.pdf (accessed on 12 October 2024).
6. Mahale, A.; Kariyappa, B.S. Architecture Analysis and Verification of I3C Protocol. In Proceedings of the 2019 3rd International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 12–14 June 2019; pp. 930–935. [CrossRef]
7. Chauhan, S.N.; Andurkar, G.K. Development of UVM Testbench for I3C protocol. In Proceedings of the 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), Delhi, India, 6–8 July 2023; pp. 1–4. [CrossRef]
8. Gao, P.; Xu, N.; Zheng, X.; Gong, J.; Zhong, X. Design Implementation and Verification of a Flexible I3C Hardware Architecture. In Proceedings of the 2024 13th International Conference on Communications, Circuits and Systems (ICCCAS), Xiamen, China, 10–12 May 2024; pp. 148–153. [CrossRef]
9. Golubic, M.; Kundrata, J.; Baric, A., Verification of the Legacy Compatibility of the MIPI I3C Master. In Proceedings of the 2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia, 27 September–1 October 2021; pp. 160–165. [CrossRef]
10. Krishnan, Y.S; Bhakthavatchalu, R. Design and Implementation of MIPI I3C master controller SubSystems. In Proceedings of the 2023 3rd International Conference on Intelligent Technologies (CONIT), Hubli, India, 23–25 June 2023; pp. 1–6. [CrossRef]
11. I3C Core. Available online: https://github.com/chipsalliance/i3c-core (accessed on 18 January 2024).
12. I3C Core—Documentation. Clock Speed and Oversampling. Available online: https://chipsalliance.github.io/i3c-core/timings.html#clock-speed-and-oversampling (accessed on 18 January 2024).
13. I3C Core—Documentation. FPGA Validation Platform. Available online: https://chipsalliance.github.io/i3c-core/testing.html (accessed on 18 January 2024).

14. Zynq UltraScale+ MPSoC ZCU106 Evaluation Kit. Available online: https://www.xilinx.com/products/boards-and-kits/zcu106.html (accessed on 18 January 2024).
15. MIPI I3C Basic Specification Version 1.1.1. Available online: https://www.mipi.org/specifications/i3c-sensor-specification (accessed on 12 October 2024).
16. I3C MDB Values—Implementers Table. Available online: https://www.mipi.org/MIPI_I3C_mandatory_data_byte_values_public (accessed on 29 July 2023).
17. Generic Device Tree Bindings for I3C Busses. Available online: https://github.com/torvalds/linux/blob/54820b4a6627e87afc0425c8b4ce338d3dbdbb80/Documentation/devicetree/bindings/i3c/i3c.txt (accessed on 27 October 2024).
18. I3C—Improved Inter-Integrated Circuit Module, 37.2.8 Hot-Join Mechanism. Available online: https://onlinedocs.microchip.com/oxy/GUID-598A6CC5-BA9B-433D-BAFE-893E2A72A7A3-en-US-14/GUID-25CE5360-2033-447B-BF99-7739E1365410.html?hl=hot-join (accessed on 12 October 2024).
19. Coraz7s Reference Manual. Available online: https://digilent.com/reference/programmable-logic/cora-z7/reference-manual (accessed on 12 October 2024).
20. Zynq-7000 SoC Data Sheet: Overview. Available online: https://docs.xilinx.com/v/u/en-US/ds190-Zynq-7000-Overview (accessed on 12 October 2024).
21. DE10-Nano User Manual (rev. B2/C Hardware). Available online: https://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=1046&FID=f1f656bb5f040121c36f2f93f6b107ff (accessed on 12 October 2024).
22. Cyclone V Device Overview. Available online: https://cdrdv2.intel.com/v1/dl/getContent/666729?fileName=cv_51001-683694-666729.pdf (accessed on 12 October 2024).
23. ADALM2000—Advanced Active Learning Module. Available online: https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/ADALM2000.html (accessed on 12 October 2024).
24. SPI Engine Offload FPGA Peripheral. Available online: https://analogdevicesinc.github.io/hdl/library/spi_engine/spi_engine_offload.html (accessed on 12 October 2024).
25. Interrupts, High-Speed DMA Controller Peripheral. Available online: https://analogdevicesinc.github.io/hdl/library/axi_dmac/index.html (accessed on 12 October 2024).
26. Vivado Design Suite 7 Series FPGA and Zynq 7000 SoC Libraries Guide (UG953). Available online: https://docs.xilinx.com/r/en-US/ug953-vivado-7series-libraries/OBUFT (accessed on 12 October 2024).
27. Vivado Design Suite Properties Reference Guide (UG912). Available online: https://docs.xilinx.com/r/en-US/ug912-vivado-properties/IOB (accessed on 12 October 2024).
28. ALTIOBUF IP Core User Guide. Available online: https://cdrdv2-public.intel.com/666402/ug_altiobuf-6PWM83471-666402.pdf (accessed on 12 October 2024).
29. MMCM and PLL Dynamic Reconfiguration (XAPP888). Available online: https://docs.xilinx.com/v/u/en-US/xapp888_7Series_DynamicRecon (accessed on 12 October 2024).
30. i3c-master-cdns.c, Linux Kernel. Available online: https://github.com/torvalds/linux/blob/master/drivers/i3c/master/i3c-master-cdns.c (accessed on 29 July 2023)
31. SPI-Engine SPI Controller Driver, Linux Kernel. Available online: https://github.com/torvalds/linux/blob/master/drivers/spi/spi-axi-spi-engine.c (accessed on 26 January 2024).
32. Control Interface, I3C Controller. Available online: http://analogdevicesinc.github.io/hdl/library/i3c_controller/interface.html (accessed on 12 October 2024).
33. Register Map Access API, Linux Kernel. Available online: https://github.com/torvalds/linux/blob/master/include/linux/regmap.h (accessed on 12 October 2024).
34. SD Card Flashing, Kuiper Linux. Available online: https://analogdevicesinc.github.io/documentation/linux/kuiper/sdcard/index.html (accessed on 12 August 2024).
35. scp(1)—Linux Man Page. Available online: https://linux.die.net/man/1/scp (accessed on 12 October 2024).

36. Libiio, Analog Devices Inc. Available online: https://analogdevicesinc.github.io/documentation/software/libiio/index.html (accessed on 27 October 2024).
37. PULSAR_ADC HDL Project, Analog Devices Inc. Available online: https://github.com/analogdevicesinc/hdl/tree/main/projects/pulsar_adc (accessed on 15 January 2024).