Received 16 August 2024, accepted 23 August 2024, date of publication 27 August 2024, date of current version 6 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3450801

RESEARCH ARTICLE

Microsecond-Level Real-Time Ethernet Deterministic Bus (REDBUS): Architecture and Motor Control Experiments

GABRIELE BRUGNONI AND LUDOVICO MINATI, (Senior Member, IEEE)
DVE Progettazione Elettronica, 21020 Casciago, Italy

Corresponding authors: Gabriele Brugnoni (info@developembedded.com) and Ludovico Minati (lminati@ieee.org)

The authors contributed in equal measures to this study. L.M. was also with the Tokyo Institute of Technology (Tokyo, Japan), and is now with the University of Electronic Science and Technology of China (Chengdu, Sichuan, China).

ABSTRACT For decades, real-time digital control systems have enabled countless applications across industry, transportation, defense, and science. A more recent trend is enhancing their interconnections using hardware and protocols, delivering timing performance comparable with that of the controllers themselves, thus enabling considerably more complex forms of real-time coordination over multiple devices, such as robot axes. Several industry standards have emerged, primarily leveraging existing IEEE802.3 Ethernet hardware and differing levels of the associated stack. However, these are generally proprietary and involve complex software libraries, which are desirable in terms of abstraction and multi-vendor interoperability but are not well-suited for situations where low-cost, small-size, and high-reliability remotization of individual sensors and actuators are required. In this paper, we introduce a complementary approach known as the Real-time Ethernet Deterministic Bus (REDBUS), which is characterized by an atypical usage of the IEEE802.3 physical layer, rewired according to a daisy-chain topology, yielding a sort of token-passing network. The cascaded devices replace control values with sensor readings on the fly at predetermined frame locations under a minimalist framework devoid of any software layers. The concepts and implementation are described in detail, an example code is provided, and representative results on the real-time control of high-speed stepper and brushless motors are presented. Due to the lack of abstraction, security, scalability, diagnostics, and multi-protocol integration features, REDBUS cannot replace the existing industrial Ethernet standards on the factory floor. Rather, it aims to complement them within predetermined designs, such as individual pieces of equipment that represent closed ecosystems, where complexity, size, and timing requirements are more pressing and where the entire data flow allocation and timings can be fixed ab initio. Diverse engineering fields can directly benefit from this simple and unconstrained architecture for meeting demanding experiment control and data acquisition requirements.

INDEX TERMS Ethernet networks, field programmable gate arrays, IEEE802.3, industrial automation, real-time control systems.

I. INTRODUCTION
Industrial networks emerged in the 1960s as a way to automate and control industrial processes, primarily in manufacturing and process scenarios. The early industrial networks mainly consisted of hardwired connections between sensors, controllers, and actuators. However, with the advent of microprocessors and the diffusion of digital technologies, industrial networks started to evolve rapidly and gradually merge, in terms of the underlying technologies and paradigms, with other types of embedded networks facing similar purposes as requirements, such as in transportation and defense applications. In the 1970s and 1980s, the development of programmable logic controllers (PLCs) revo-

The associate editor coordinating the review of this manuscript and approving it for publication was Binit Lukose.

lutionized industrial automation, as these devices allowed for far more complex control algorithms, higher flexibility, and real-time monitoring and control. Importantly, the introduction of the IEEE802 standard in 1980 also paved the way for the development of industrial Ethernet, which is now widely used as the backbone of industrial networks [1], [2], [3], [4]. The 1990s saw the emergence of fieldbus technologies, such as Profibus, which allowed for more efficient communication between devices and reduced wiring costs. In more recent years, a further transition has been taking place from legacy interconnection standards such as RS-485 to Ethernet, and this has, to some extent, been mirrored in other fields such as aerospace, defense and automotive, with ever-increased integration between, for example, CAN, MIL-STD-1553, and ARINC standards, and Ethernet [5], [6], [7], [8], [9], [10].

On the other hand, at its inception, Ethernet was initially devised primarily for office networking applications and lacked several of the timing and availability features essential to industrial and other critical applications [1], [2]. In particular, deterministic real-time operation is a fundamental prerequisite for machine control, process automation, safety management, maintenance, and, in general, all applications wherein fixed physical time constraints are imposed by direct interaction with a dynamical real-world scenario [11], [12]. To address this challenge, multiple solutions and standards have been proposed, including proprietary extensions such as PROFINET, and particularly PROFINET RT and IRT, from Siemens, other protocols such as Modbus TCP and, more recently, EtherCAT, developed by Beckhoff Automation in the early 2000s [5], [6], [7], [8]. To respond systematically to the ever-increasing usage of Ethernet in real-time applications, throughout the years 2016-2020, the Time Sensitive Network (TSN) extensions to the IEEE802.1 standard were published [9], [10], [13].

Today, developers are offered a range of options to deploy Ethernet with different levels of timing performance and determinism in control applications. EtherCAT uses a master-slave architecture and a distributed clock mechanism to achieve cycle times of less than 1 ms [8]. PROFINET supports both real-time and non-real-time communication and offers a wide range of device profiles [6]. Both were designed for large-scale interoperability between devices and vendors and, as such, implement a significant level of abstraction, resulting in a firmware stack footprint on the order of 10 KBytes or more, rendering them not well-suited for low-level hardware implementation [14], [15]. On the other hand, TSN is an IEEE standard that allows multiple industrial protocols to share the same network infrastructure, reducing complexity and cost and enabling different classes of traffic to coexist on the same media. However, the TSN extensions require specific hardware-level capabilities that must be supported by all the devices involved [13].

For completeness, it is worth mentioning that the need to enhance timing performance and determinism in Ethernet connectivity has also been encountered in other application domains, notably the usage of Ethernet infrastructure for long-distance, high-aggregation connectivity. The term carrier-grade Ethernet, reflecting a service-oriented concept rather than a defined set of specifications, has been introduced to refer to all applications of Ethernet at the urban level, particularly having the purpose of interconnecting geographically dispersed buildings and workers. Compared to regular office-grade Ethernet, the thrust is towards lean, reduced protocol stacks combined with traffic engineering and hierarchical planning to attain quality-of-service comparable to connection-oriented architectures, such as synchronous optical networks. While industrial networking and carrier-grade Ethernet share a common pursuit for deterministic timings and reliability, the requirement profiles are different in that, typically, industrial networks require tighter timing margins and leaner software stacks. This is due to the physical nature of the controlled processes and the fact that the interconnected equipment primarily consists of large numbers of resource-constrained embedded devices rather than high-performance routers. Consequently, the facilities supporting quality-of-service, security, and fault management are profoundly different between industrial and carrier-grade Ethernet [16], [17].

Consequently, there remained a gap between the capabilities of these standards and the need for a high-speed, low-complexity, low-abstraction, and low-cost solution for use in interconnecting the sensors and actuators within fully self-contained and proprietary devices. For example, EtherCAT and PROFINET RT/IRT are widely used in large multi-axis machines. However, the cost and complexity of their implementation make them poorly suited for equipment such as entry-level robots, small autonomous vehicles, non-critical equipment such as complex lighting systems, etcetera. The abstraction and integration of these standards may well represent an unnecessary burden for such applications. On the other hand, the majority of off-the-shelf microprocessor and microcontroller boards available to date do not yet support the TSN extensions, which, moreover, require a capable switch [13], [18].

In this paper, we present a new concept, grounded on an atypical use of mainstream IEEE802.3 Ethernet physical layer hardware, whereby devices are cascaded by daisy-chaining their transmit and receive signal pairs so that the transmitter of the previous device is connected to the receiver of the next device to form a chain (Fig. 1a). While unorthodox according to the 100BASE-T specifications, which prescribe a star-based physical interconnection architecture using switches (Fig. 1b), this topology is viable and proves to have numerous advantages, realizing in effect a sort of token-passing system wherein the Ethernet frame sent by the controller can traverse all peripheral nodes and return to it [4], [19].

Each peripheral replaces the intended control data on the fly with sensor readings and other returned information. This architecture has been named the Real-time Ethernet Deterministic BUS (REDBUS). As visible in Fig. 2, the highly elementary nature of the architecture required to

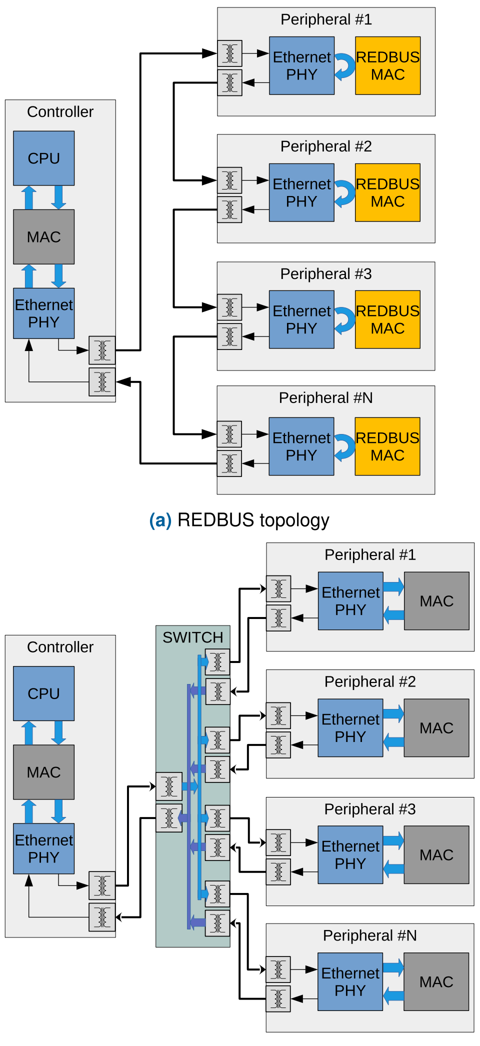


FIGURE 1. Comparison between the REDBUS and conventional IEEE802.3 standard physical interconnection topology.

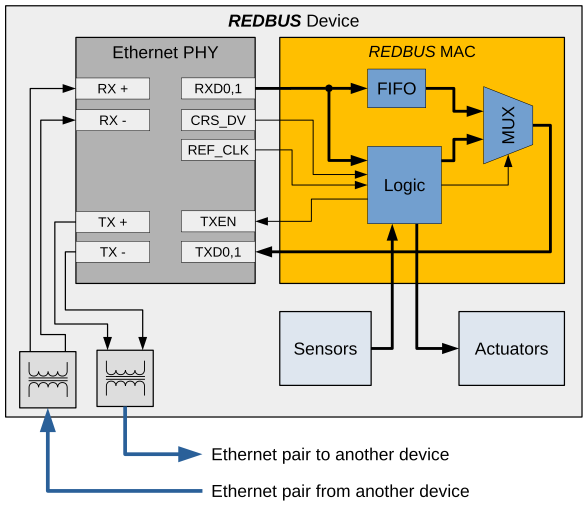


FIGURE 2. REDBUS peripheral node architecture. Data is extracted from the Ethernet payload, used to drive the actuators, and replaced on the fly with incoming information from sensors or generic inputs.

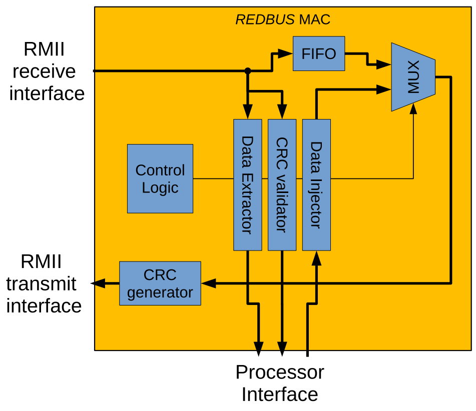
implement each node, not requiring any software stack, renders it well-suited for hardware-level implementation. It simply consists of a first-in, first-out (FIFO) buffer that repeats the received data, replacing part of the content when necessary, with additional logic adjusting the Ethernet frame fields to retain compliance with the physical and link layers of the IEEE 802.3 standard. For the avoidance of doubt, it is underlined that, unlike other industrial Ethernet standards, REDBUS cannot coexist with other Ethernet-based protocols. Due to the lack of abstraction, security, scalability, diagnostics, and multi-protocol integration features, REDBUS cannot replace the existing industrial Ethernet standards on the factory floor. Instead, it aims to complement them within predetermined designs where complexity, size, and timing requirements in the remotization of one or more sensors and actuators are more pressing and where the entire data flow allocation and timings can be fixed ab initio.

II. SYSTEM ARCHITECTURE AND PERFORMANCE

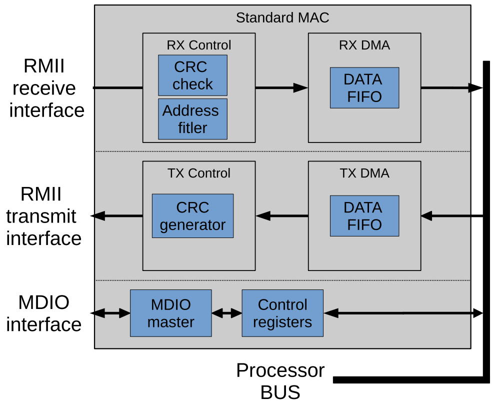
A. OVERVIEW

As said, the REDBUS architecture comprises a controller and one or more peripheral nodes, interconnected by leveraging standard IEEE802.3 Ethernet physical layers in an atypical topology forming a ring, comprising a chain of devices and closed by a controller [4], [19]. As said, the key distinguishing feature is that no switch is required, allowing the latency to be precisely predetermined and minimized and the throughput to be maximized without requiring any dedicated protocol extensions or dedicated hardware. As will be shown below, while the proposed interconnections deviate from the standard Ethernet implementation, they are readily accepted by all existing, commercially available physical layer network devices (PHYs). Moreover, it is possible to realize the controller utilizing standard media access controllers (MACs) without necessitating any dedicated infrastructure, thus substantially simplifying the deployment to existing programmable logic controllers (PLCs) and other embedded devices. Peripherals, on the other hand, comprise two fundamental components: a standard Ethernet PHY and a dedicated, special MAC capable of handling the uncommon chain topology, configured at the hardware level for the target application and typically realized using a field-programmable gate array (FPGA) device [4], [19].

A peculiarity of the proposed architecture is that the PHYs of the peripheral devices use unidirectional communication towards the PHYs of the adjacent devices along the chain. Unlike the classic topology of modern Ethernet networks wherein each PHY is connected to a counterpart through both receive and transmit signals, in the REDBUS topology, only one signal pair is wired from one PHY to the next: the signal emitted by the transmit port of one device enters the receive port of the next device, and so on, until a ring including the controller itself is formed [4], [19]. In other



(a) REDBUS MAC architecture



(b) Standard IEEE802.3 MAC architecture

FIGURE 3. **Comparison between the REDBUS and standard IEEE802.3 Media Access Controller (MAC) internal architectures.**

words, as shown in Fig. 1a, according to this particular interconnection scheme, the head of the chain of peripherals will have its receive port left open, while the tail of the chain will have its transmit port left open. These two ports are then connected to the controller and seen by the same indistinguishably as if they represented a conventional point-to-point link between two Ethernet PHYs and MACs. The extended nature of the chain is effectively concealed.

As detailed below, compared with the existing industrial Ethernet standards described above, the proposed architecture is logically simpler and, especially, features lower latency. A European patent covers usage of the REDBUS architecture in commercial products, however, unlimited permission for use in realizing the system disclosed herein towards academic research by not-for-profit institutions is granted under all the terms of the GNU General Public License version 3; besides, the underlying concepts are universal and can be straightforwardly applied by anyone to develop other architectures not subject of the patent [20].

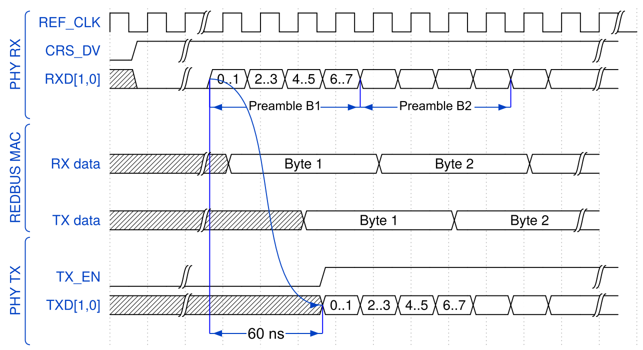


FIGURE 4. **Representative PHY RMII digital loopback timings derived from RTL simulations.**

B. MEDIA ACCESS CONTROLLER (MAC) DESIGN

The REDBUS MAC has an uncomplicated architecture and can preferably be realized using register transfer level (RTL) logic specified using the VHDL or Verilog languages and integrated into FPGA or application-specific integrated circuit (ASIC) components. Due to the peculiar nature of the network topology, standard MACs are poorly suited for implementing REDBUS peripheral connections [4], [19]. The REDBUS MAC is considerably simpler and has a datapath architecture different from the standard Ethernet MAC. As shown in Fig. 3a, its distinctive feature is the presence of a FIFO and multiplexer (MUX) for rapid frame forwarding with hardware-level on-the-fly data replacement, connected to a processor interface feeding data extraction and injection submodules. By contrast, the standard Ethernet MAC, visible in Fig. 3b, being general-purpose, possesses two independent media interfaces with entirely separate buffering and clocking. Consequently, it is unable to perform on-the-fly frame forwarding and data replacement which need to be implemented at a higher level, resulting in longer latency [4].

The primary task of the REDBUS peripheral MAC is to repeat the Ethernet frame content received from the physical receive channel to its transmit channel, passing it on to the next device along the chain. Data can be received by the MAC in groups of two or four bits, depending on the type of connection with the Ethernet PHY; media-independent interfaces (MII, four bits at a time), or reduced MII interfaces (RMII, two bits at a time) are prevalently used. The existing RTL implementation of the REDBUS MAC comprises a total of $\approx$ 2000 lines of VHDL code, defining 9 entities. Behavioral-level simulation using the ModelSim environment (Lattice FPGA Edition 2020.3, Mentor Graphics Inc., Wilsonville OR) allows demonstrating that the bits are forwarded by the PHYs nearly instantaneously, with the typical latency introduced by each device being on the order of 60 ns (interface clock at 50 MHz), as visible in Fig. 4.

The basis of the REDBUS protocol is that the data contained within the Ethernet payload are divided into predetermined and dedicated slots, each of which has a target address, meaning a fixed device along the chain that it is intended for. When each MAC intercepts a frame, the bits flowing through it corresponding to its address are instantaneously replaced with the data to be sent back from the peripheral to the controller without introducing

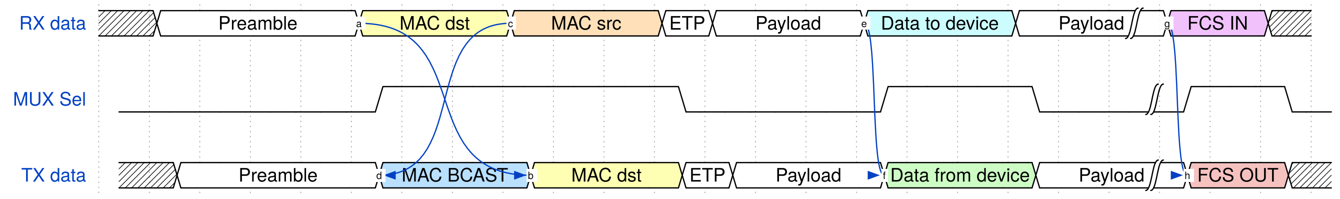


FIGURE 5. Example of REDBUS frame retransmission and multiplexer (MUX) selection.

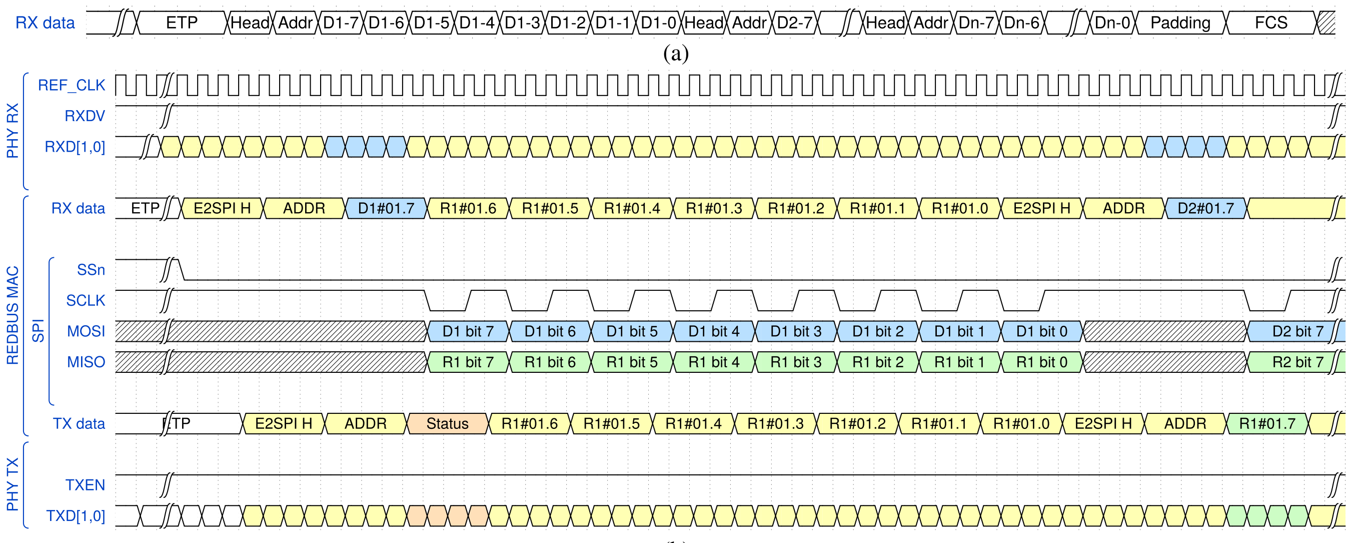


FIGURE 6. Payload data organization into addressed slots. Each slot contains one byte for each device in the same address group. (a) Layout of the slots within the payload. (b) REDBUS MAC SPI data exchange example involving the device with subaddress 7 within a group of eight devices per address. In the first slot of eight bytes, each byte is assigned to a device.

any additional software-related latency. As the experiments reported below demonstrate, a typical example involves a motor interface peripheral receiving current set-points and returning corresponding instantaneous current and voltage readings.

Notably, the first peripheral device along the chain replaces the source MAC coming from the controller with its unique REDBUS MAC address (an internally defined magic number), and, as the destination address, it inserts either the one corresponding to the source MAC, or the broadcast address, depending on a configuration setting. At the end of the transmission, since some bytes in the Ethernet frame have been changed, the frame check sequence (FCS) field is replaced with the checksum computed on the fly during the flow of data transit. Owing to this arrangement, the outgoing Ethernet frame from the last device contains a destination MAC address such that the controller that has sent or is still transmitting the original frame, recognizes it as valid. The mechanism is diagrammed in Fig. 5.

Depending on the application, a REDBUS MAC may be realized and deployed as a stand-alone integrated circuit, or combined with other functions. On the one hand, an FPGA or ASIC can be interfaced directly with external analog-to-digital and digital-to-analog (ADC, DAC) converters or logic signals. On the other hand, real-time communication with a microcontroller, microprocessor, or other target device can be conveniently attained through a synchronous serial interface (SSI or SPI) [11], [21]. A minimalist implementation of the MAC is based on external data exchange using an SPI-type interface and can be, for instance, deployed in the smallest MACHXO2 type FPGA (Lattice Semiconductor Inc., Portland OR, USA), resulting in an occupancy of 194 registers, 125 slices and 248 look-up tables (LUT4) [22].

C. PHYSICAL LAYER (PHY) INTERFACE AND LATENCY

Interfacing between a REDBUS MAC and an Ethernet PHY can be accomplished through any of the specialized interfaces specified by the IEEE802.3 standard and its updates [4]. However, it should be borne in mind that the nature of the topology is suitable for realization using 100BASE-T media and its variants [19] but not, for example, using 1000BASE-T, since the presence of two fully separate transmit and receive twisted pairs is a fundamental prerequisite. For this reason, only implementations based on the MII and RMII local interfaces can be considered.

The MII interface provisions two separate clocks for transmission and reception. The data stream received by the PHY originates from an external clock domain, which the

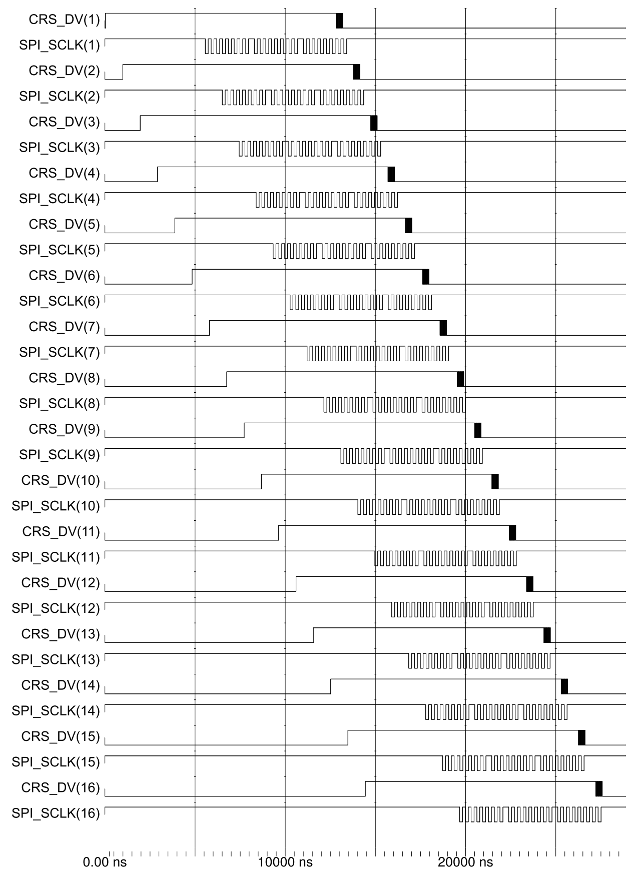


FIGURE 7. RTL-level behavioral simulation of an Ethernet frame flowing through 16 cascaded devices, assuming type KSZ8081 physical layers (see Table 2). The `CRS_DV` signals, output by the PHY devices and asserted in the presence of valid Ethernet frames, and the `SPI_SCLK` signals, namely the clocks synchronous with the data sent through SPI, are shown. In this example, three bytes are exchanged per device.

PHY reconstructs from the carrier signal and provides to the MAC. On the other hand, the local application clock of the MAC is used for transmission. By contrast, the RMII interface uses the same clock to provide received data to the MAC and transmit data from the MAC so that all signals between the PHY and MAC are synchronous in a unified clock domain. This requires the PHY to include a dedicated FIFO buffer to absorb small frequency differences and jitter between the remote and local clock domains, which introduces some latency. Following initial analyses about the typical requirements for motor control, in the current REDBUS implementation, the RMII interface was chosen to minimize logic occupancy at the expense of a slightly higher latency [23], [24]. Future work will include transitioning fully to the MII interface while implementing dynamic clock-shifting techniques within the MAC to ensure that timing requirements are met while avoiding the additional latency associated with a FIFO buffer. Mainstream implementations of Ethernet connectivity do not place as much emphasis on latency as REDBUS, therefore, this approach to latency minimization remains to be developed.

To date, a wide variety of off-the-shelf PHY devices with RMII interfaces are available on the market. While largely functionally equivalent, as detailed in Table 1, these devices feature substantial variation in latency performance, stemming from internal circuit architectural and manufacturing process differences. For instance, the PHY type TLK105 provides a balanced latency $<$ 180 ns in both directions, whereas the type TJA1100 may incur almost an order of magnitude longer delay in data propagation. It follows that, for applications such as REDBUS, careful analysis of PHY timings is necessary to drive optimal decisions.

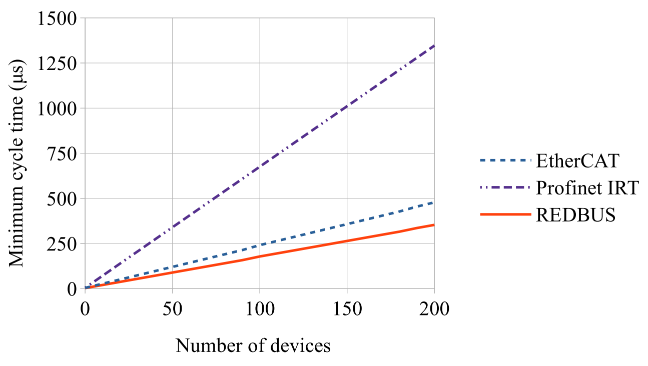
TABLE 1. Manufacturer-declared latencies of representative commercial PHY devices equipped with the RMII interface. The MDI to RMII value denotes the latency between the analog signal on the media-dependent (MDI) input and the first edge of the RMII clock accompanied by the assertion of the `RX_DV` signal. Vice-versa for the transmitter. The parameters reported with the same value in the min and max columns must be understood as typical. Devices without known parameters were omitted.

Device	Brand	MDI to RMII		RMII to MDI	
		min (ns)	max (ns)	min (ns)	max (ns)
TJA1100 [25]	NXP Semiconductors	700	1070	190	540
DP83620 [26]	Texas Instruments	185	185	110	110
DP83826I [27]	Texas Instruments	268	288	88	96
TLK105 [28]	Texas Instruments	176	176	120	120
ADIN1300 [29]	Analog Devices	328	368	72	92
STE101P [30]	STMicroelectronics	200	200	300	300

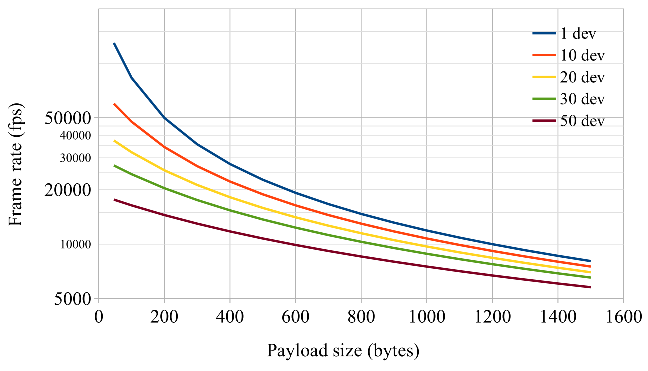
D. REDBUS TO SYNCHRONOUS SERIAL CONVERTER

This subsection describes some design considerations concerning the realization of a complete bridge between REDBUS and a synchronous serial interface such as SPI. It should, at first, be noted that except for multi-lane versions geared at some specific applications, serial interfaces such as SPI have significantly lower data transfer rates compared to the Ethernet interface, on the order of 10 Mbps vs. 100 Mbps [21]. As the REDBUS protocol was conceived to interconnect a plurality of devices, it is nevertheless possible and appropriate to group data belonging to several peripherals into each Ethernet frame. In other words, device-specific data can be exchanged with the external processor or other devices at a lower speed, while at the same time, information destined for other devices travels in the same Ethernet frame at a higher speed.

Using a prescaler, the raw Ethernet data bitrate of 100 Mbps can be divided to reduce the serialization clock on the synchronous serial interface. For instance, three suitable different division factors are 32, 16, and 8, which yield endpoint bitrates of 3.125 Mbps, 6.25 Mbps, and 12.5 Mbps. Let us consider the lowest bitrate: while data is serialized at a rate 32 times lower, the Ethernet data continues to advance through each MAC device. As long as the data in the Ethernet frame payload are organized so that each byte becomes available as soon as the previous one has completed the serialization on the SPI interface, it is easy to understand that the various bytes of data must be spaced by at least 32 bytes. Since the Ethernet payload is intended to carry data for multiple devices, the required bytes

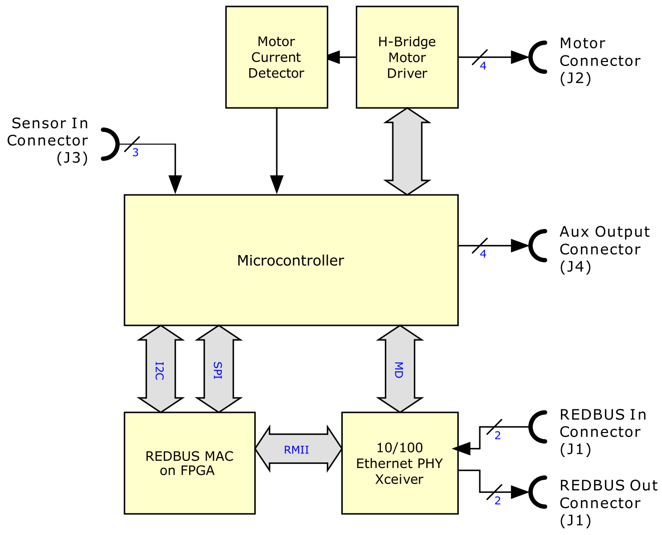


(a) Comparsion with other industrial Ethernet standards

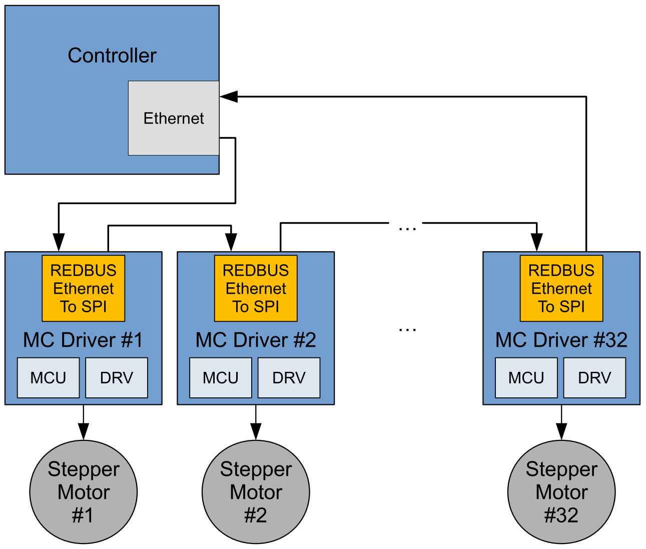


(b) Frame rate under different conditions.

FIGURE 8. Key performance figures. bf(a) Minimum achievable cycle times at a bandwidth of 100 Mbps for EtherCAT, PROFINET IRT, and REDBUS (assuming PHY type DP83826I, as per Table 1) on a chain network with 16 bytes payload per device, as a function of the number of devices. (b) Frame rate vs. payload size for different numbers of peripheral devices in the loop.



(a) Block diagram of the individual motor driver board



(b) A controller interfaced to 32 peripherals, each corresponding to a stepper motor. MCU: Microcontroller, DRV: Power Driver.

FIGURE 9. Representative implementation of a system for synchronously controlling multiple stepper motors. (a) Block diagram of the motor driver board. The REDBUS MAC is implemented within an FPGA, attached to the Ethernet PHY via the RMII interface, and to the host microcontroller by means of I2C and SPI interfaces, respectively, for configuration and data exchange. (b) Block diagram of a controller board arranged to control 32 stepper motor drivers.

of minimum spacing can be filled with data intended for other devices. Assuming, without loss of generality, to have exactly 32 REDBUS devices connected in the chain, the data in the payload will be organized as represented in Table 2. While the first device is serializing the data, the following devices can begin serialization almost simultaneously. The Ethernet flow that travels through a device is affected by a propagation delay, therefore, additional improvements in the overall serialization latency could be obtained by inverting the device data order (not shown).

Considering the data in Table 2, each row represents a slot, and each slot is characterized by a header that defines the properties of the slot. One important property is the address. In the Ethernet-to-SPI MAC, each slot contains one byte for each of the destination devices. The slot has a single-address property, so each device whose data is included should have the same address. For this reason, the Ethernet-to-SPI MAC is provided with both address and sub-address properties. The sub-address range depends on the clock division ratio of the SPI: if divided by 32, then the sub-address will range from 0 to 31, if divided by 8, the sub-address will range from 0 to 7. Depending on the serial port clock rate, the device

TABLE 2. Data arrangement in the Ethernet frame payload for serialization towards an SPI interface. Each row in the table is inserted into the payload in succession after the previous one.

Device#1 B1	Device#2 B1	Device#3 B1	...	Device#32 B1
Device#1 B2	Device#2 B2	Device#3 B2	...	Device#32 B2
...	...	...	...	...
Device#1 Bn	Device#2 Bn	Device#3 Bn	...	Device#32 Bn

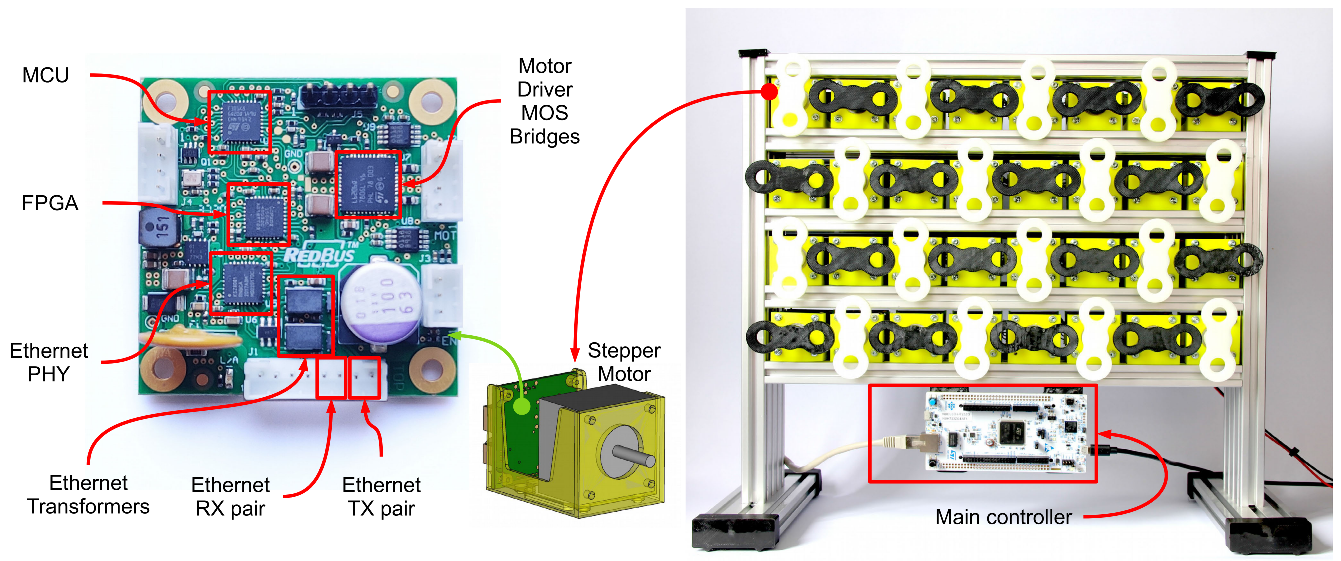


FIGURE 10. Hardware verification system comprising 32 driver boards, each connected to a stepper motor. The interlocking propellers successfully rotate synchronously at 5500 RPM under real-time open-loop control via REDBUS.

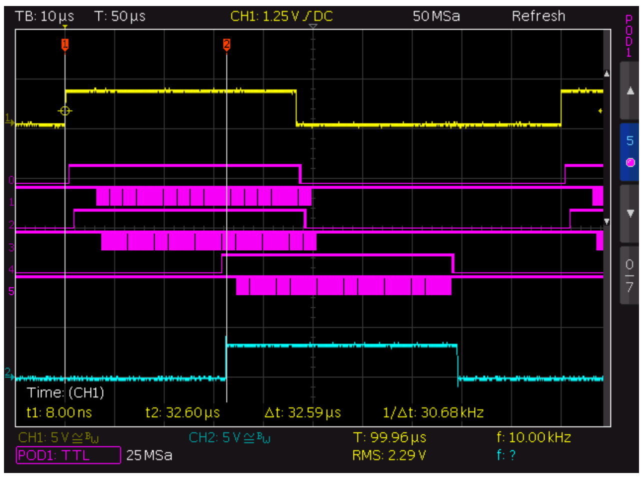


FIGURE 11. Experimentally-measured round-trip latency of 32.6 μs between the transmission of the Ethernet frame by the controller (yellow trace rising edge) and its reception back by the same (cyan). Magenta traces (top to bottom): RMII data available and SPI clock on the driver boards number 1, 2, and 32.

chain will comprise multiple device groups, each owning the same address and an incremental sub-address. The address field is an 8-bit value, with the values 0 and 255 considered reserved. The group size determines the overall number of devices addressable, as reported in Table 3.

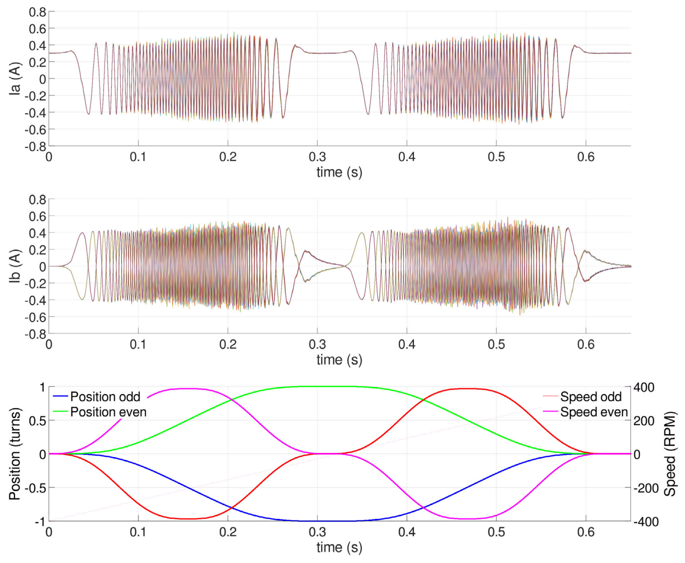
Fig. 6a shows an example of data organization for 12.5 MHz serialization over eight devices, and Fig. 6b illustrates the data exchange mechanism for the device with address 1 and sub-address 7. The MAC collects the two-bit pairs incoming from the PHY device signals `RXD1` and `RXD0` until a full byte has been received. Next, the MAC starts to serialize each bit of the received byte on the SPI port, connected to an external processor.

TABLE 3. Number of addressable devices, data size, and sub-address size versus SPI clock frequency.

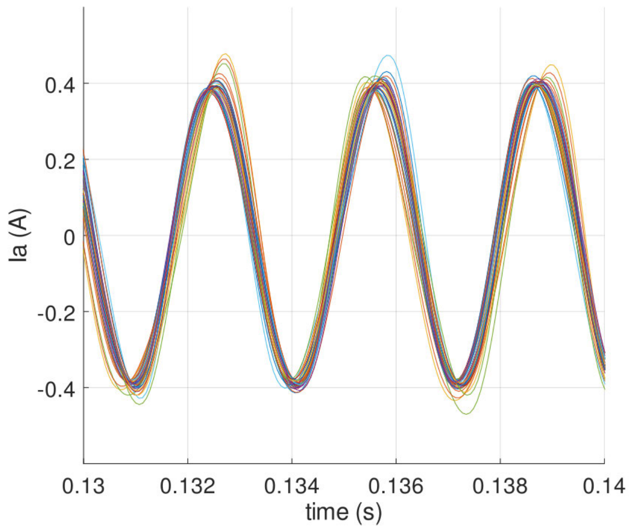
Clock Freq	Sub-address size	Max bytes per device inside a single Ethernet frame	Number of addressable devices
12.5 MHz	8	150	2,024
6.25 MHz	16	83	4,048
3.125 MHz	32	44	8,096

The SPI interface provides a bidirectional method to exchange data. On each rising edge of the SPI clock, the MAC outputs a bit to an external processor and receives another from it. After serializing the byte from the Ethernet payload, a byte from the processor becomes available to the MAC. This byte, highlighted in green in Fig. 6b, is placed into the next available slot of the transmitted Ethernet frame. Each byte in the received frame is substituted with the data coming from the external processor. It should be noted that the first byte cannot be replaced by data coming from SPI because the serialization is not yet complete. Consequently, a MAC status byte is instead inserted as the first byte, and the reply is shorter by one byte compared to the received data length.

The response data provided by the first device enters the second device and is repeated without modification, then enters the third device and is repeated without modification, and so on, until the last device sends it to the controller that initiated the communication. In this way, each device "fills" its boxes by replacing (overwriting) the original content, and the collected data returns to the main processor after having traveled through all the devices along the chain. A timing diagram example of data exchange through 16 devices is visible in Fig. 7.



(a) Winding currents, position and speed recorded during continuous rotation.



(b) Magnified view of the I_a plot.

FIGURE 12. Representative current recordings from the stepper motor experiment. (a) Real-time current for all 32 motors, captured after fast rotation direction reversal. In the I_a plot, all the currents present the same phase. In the I_b plot, the currents have a phase shift of 180° depending on the motor direction (odd-numbered motors: counter-clockwise, even-numbered motors: clockwise). (b) Overlapping waveforms confirm the excellent phase synchronization level between the digitized winding currents and read back from nodes.

E. TIMING COMPARISON WITH EtherCAT AND PROFINET IRT

In this section, the timing performance and scalability of REDBUS and two other standards built upon IEEE802.3 Ethernet, namely EtherCAT and PROFINET IRT, are compared according to the following simplifier model, adapted from Ref. [7]. It must be underlined that there are profound functional differences, because REDBUS is, by design, devoid of the abstraction, security, scalability, diagnostics, and versatility provided by the other two standards. The comparison of timings, therefore, has a practical value only with reference to scenarios where both REDBUS and the other preexisting standards could be used, such as predesigned equipment that only requires sensor and actuator remotization, without the other high-level features required for factory-level integration.

The attainable cycle time t_{cycle} was expressed as

$$t_{\mathrm{cycle}} = t_{\mathrm{overhead}} + t_{\mathrm{payload}} + t_{\mathrm{propagation}}, \tag{1}$$

where t_{overhead} denotes the overhead time for transmitting the required number of Ethernet frames, t_{payload} represents the actual payload transmission time, and $t_{\mathrm{propagation}}$ is the time necessary for the Ethernet frames to propagate through all the devices on the bus.

Assuming that each device requires a fixed amount of actuation and sensing data d_{node} to be carried, the total payload size in bytes d_{payload} was calculated as

$$d_{\mathrm{payload}} = n_{\mathrm{nodes}} d_{\mathrm{node}}. \tag{2}$$

Consequently, the required number of Ethernet frames n_{frames} was obtained as

$$n_{\mathrm{frames}} = \lceil d_{\mathrm{payload}} / d_{\mathrm{max}} \rceil, \tag{3}$$

where d_{max} denotes the maximum number of payload bytes per Ethernet frame according to each standard. In turn,

$$t_{\mathrm{overhead}} = n_{\mathrm{frames}} t_{\mathrm{frame}}, \tag{4}$$

where t_{frame} denotes the minimum time required to send a frame, given by

$$t_{\mathrm{frame}} = d_{\mathrm{fixed}} t_{\mathrm{byte}}, \tag{5}$$

where d_{fixed} is the fixed number of bytes, comprising the preamble, header, and check fields of all encapsulation layers, as well as the inter-frame gap, and t_{byte} is the time needed to transmit a byte. Similarly, for the payload,

$$t_{\mathrm{payload}} = d_{\mathrm{payload}} t_{\mathrm{byte}}. \tag{6}$$

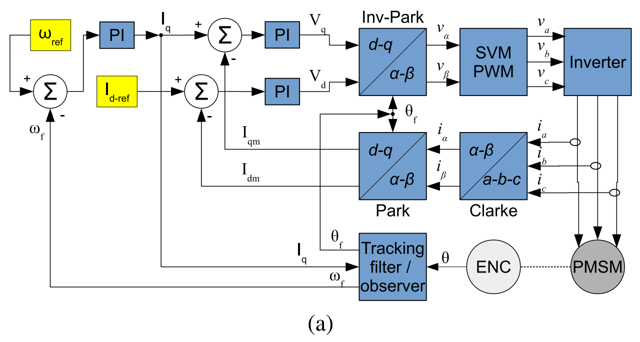
Finally, the time necessary for the Ethernet frames to propagate through all the devices was given by

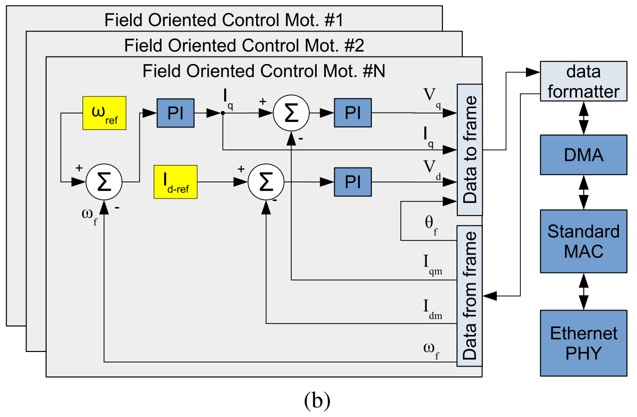
$$t_{\mathrm{propagation}} = n_{\mathrm{nodes}} t_{\mathrm{forward}}, \tag{7}$$

where t_{forward} denotes the time needed for a node to receive, process and forward an Ethernet frame (determined, among other factors, by FIFO buffers, firmware processing, etc.). For clarity, this parameter could be further decomposed as

$$t_{\mathrm{forward}} = t_{\mathrm{PHY}} + t_{\mathrm{electrical}} + t_{\mathrm{internal}}, \tag{8}$$

where t_{PHY} denotes the maximum latency introduced by the Ethernet PHY in use (Tx+Rx time), $t_{\mathrm{electrical}}$ represents the electrical propagation time (group delay of the channel), and




Field Oriented Control Mot. #1
Field Oriented Control Mot. #2
Field Oriented Control Mot. #N
Data to frame
Data from frame
data formatter
DMA
Standard MAC
Ethernet PHY
(b)

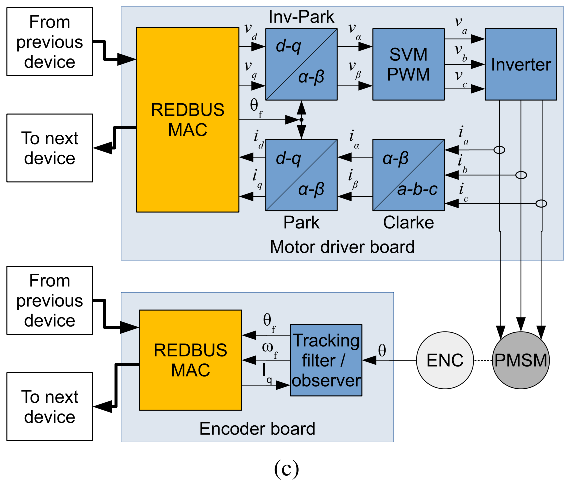


FIGURE 13. Control schemes and their possible mapping over multiple REDBUS nodes. (a) A typical brushless motor driver based on field-oriented control (FOC). (b) A set of PI controllers used for FOC are executed in the main controller processor. The driving vectors output data V_d and V_q are organized inside the Ethernet frame and sent to the driver device. The current values I_d and I_q and the encoder data θ and ω are returned back after the frame has passed through the driver and encoder devices. (c) The inverter device receives real-time data from the REDBUS node and returns back information about position, speed, and winding currents. See text for description. ENC: Encoder. PMSM: Permanent Magnet Synchronous Motor.

t_internal represents the internal latency due to buffering and processing.

To compare the three standards, the attainable cycle time t_cycle was charted as a function of the number of nodes on the bus n_nodes, assuming the appropriate constant values according to the specifications of each standard [7].

Across the three standards, the fixed amount of actuation and sensing data per node was set to $d_\mathrm{node} = 16$ byte, the time needed to transmit a byte was assumed to be $t_\mathrm{byte} = 80$ ns according to 100BASE-T signaling, and the electrical propagation time was taken as $t_\mathrm{electrical} = 10$ ns. Following the differences in implementation indicated above and Ref. [7], the maximum latency introduced by the Ethernet PHY was assumed to be $t_\mathrm{PHY} = 960$ ns for EtherCAT and PROFINET IRT, and $t_\mathrm{PHY} = 370$ ns for REDBUS. The internal forwarding latency was taken as $t_\mathrm{internal} = 80$ ns for EtherCAT and $t_\mathrm{internal} = 60$ ns for PROFINET IRT and REDBUS. Based on these parameters alone, it could be seen that the node forwarding time was considerably larger for EtherCAT and PROFINET IRT, namely, $t_\mathrm{forward} = 1\ \mu$s, than for REDBUS, namely, $t_\mathrm{forward} = 0.44\ \mu$s.

Due to the different packet formatting, the maximum number of payload bytes per Ethernet frame was considerably larger for EtherCAT and REDBUS, respectively $d_\mathrm{max} = 1488$ byte and $d_\mathrm{max} = 1500$ byte, than PROFINET IRT, namely $d_\mathrm{max} = 16$ byte. The fixed number of bytes per frame was, in order, $d_\mathrm{fixed} = 50$ byte, $d_\mathrm{fixed} = 38$ byte and $d_\mathrm{fixed} = 55$ byte.

Fig. 8a provides a comparison between the cycle times characteristics of REDBUS, EtherCAT, and PROFINET IRT. It can be seen that the cycle time scales considerably more favorably for REDBUS than for both EtherCAT and PROFINET IRT, owing to both the higher payload-carrying capability and lower forwarding latency. The frame rate across various application scenarios, namely, the number of peripheral nodes, has also been charted in Fig. 8b.

As revealed by these figures, from an architectural and performance perspective, EtherCAT and REDBUS are considerably more similar to each other than PROFINET IRT. Both operate at the data link level of Ethernet frames without requiring higher stack layers, and both are grounded on inserting (or replacing) process data on the fly. The

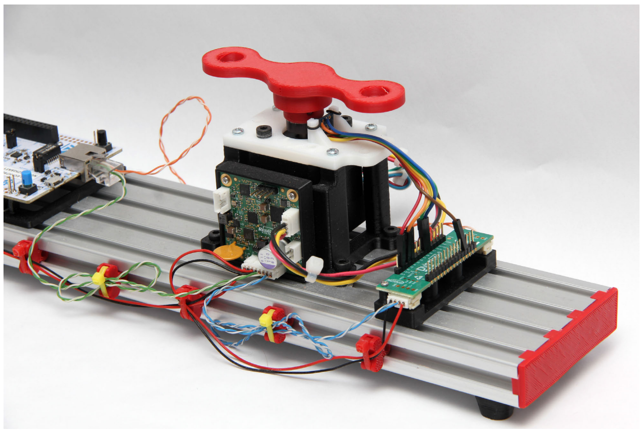


FIGURE 14. Experimental setup for testing the brushless DC (BLDC) motor control system. Left: controller board, Center: motor and driver board attached to it, Right: encoder board. Green/white twisted pair: controller to driver board, blue/white pair: driver to encoder board, orange/white pair: encoder board back to controller.

fundamental difference is the topology, since EtherCAT also supports line, tree, and star networks, with a pass-through approach that is compliant with the standard wiring of IEEE802.3 Ethernet connections, thus necessitating two PHYs per node, whereas the key notion behind REDBUS is that of rewiring the nodes into a ring, reducing the number of PHYs per node to one [5], [7], [8], [14], [15]. REDBUS thus reduces the latency at the price of flexibility. Furthermore, it is devoid of the higher-level standardization layers offered by EtherCAT, thus only being suitable for ad-hoc applications and not multi-vendor integration. PROFINET, including PROFINET RT and IRT, is substantially different in that it operates at the application layer of the OSI stack and is, therefore, built upon TCP and UDP packet protocols. It thus incurs a considerably higher level of software complexity and latency while having important advantages in terms of integration with other and preexisting information technology systems [6], [7]. It should also be noted that EtherCAT can leverage TSN extensions and, therefore, deliver excellent timing performance. At the same time, interconnections are realized using switches instead of point-to-point, and PROFINET, too, can be implemented over TSN networks [9], [13], [18]. Moreover, both EtherCAT and PROFINET networks can be created using 1000BASE-T hardware, whereas the atypical daisy-chaining of the twisted pairs confines REDBUS to 100BASE-T. Overall, the three standards have different and complementary profiles in terms of performance and complexity [4], [19].

It is also essential to remember the fundamental difference between EtherCAT, PROFINET, and REDBUS: while the former two can share the same physical network as other Ethernet protocols, the latter cannot. One of the defining features of modern industrial Ethernet standards is the ability to deliver consistent timing performance while coexisting with other non-time-sensitive protocols. Due to the interconnection topology, this is inherently impossible with REDBUS, and its timing performance, therefore, exclusively depends on hardware factors.

III. EXPERIMENTAL CONFIRMATION

A. OPEN-LOOP DRIVING A STEPPER MOTOR ENSEMBLE

Owing to its highly deterministic nature, minimal overhead, and suitability for hardware-level peripheral implementation, REDBUS is well-adapted for applications in remote motor control, wherein a central processor calculates in real-time the currents through a multitude of motors in tight synchrony. To demonstrate these applications, as shown in Fig. 9a, a dedicated peripheral board was designed, including a Cortex-M3 microcontroller (type STM32F301K, 72 MHz clock; STMicroelectronics SpA, Agrate Brianza, Italy) also implementing pulse-width modulation (PWM) signal generation and current readout, an FPGA (type LCMXO2-256HC-4; Lattice Semiconductor Inc., Hillsboro OR) realizing the REDBUS MAC and interfaced via SPI to the microcontroller, a standard Ethernet PHY (type KSZ8081RNB; Microchip Technology Inc., Chandler AZ), and bridge driver (type L6206Q; STMicroelectronics SpA) connected to power MOSFETs capable of handling up to 3 A of winding current.

As visible in Fig. 10, a mechanical assembly presenting particularly severe synchronization challenges was purposedly designed in the form of a 4×8 array of actuators, each one comprising a stepper motor directly attached to a propeller-like blade. In the past, arrangements of this kind were considered for naval propulsion. However, they were rapidly abandoned due to the catastrophic consequences of even transient loss of synchronization between the propeller blades [31]. The motors (type 16HS0014; Shanghai Moons Electric Co. Ltd., China) had 50 magnetic pole pairs, measured $39 \times 39 \times 26$ mm, were rated for 0.65 A and 5 V, and had a step angle of 1.8°. The array pitch was 50×68 mm, and the blade length was 74 mm, consequently achieving a tight interlock that allowed a maximum angle error on the order of $\pm 20°$. The 3D blade design files are provided as Supplementary Material. Due to the interlocking design, any deviation from perfect synchronization results in a domino effect of collisions, readily appreciable to the operator. The corresponding interconnection topology, shown in Fig. 9b, involved the frame traveling through all the actuators, each motor being provisioned with a fully separate driver board in the form of a REDBUS peripheral, physically attached behind it.

The data were distributed to the 32 driver boards through a single Ethernet frame. Each node replicated the incoming frame while replacing the position information content with the real-time measures of winding currents and other diagnostic information. The overall size of the data for each driver board was 18 bytes. More specifically, the controller sent to each driver board 2 bytes encoding real-time commands, 2 bytes encoding non-real-time commands, 4 bytes representing the angle increment $\Delta\theta$, and 2 bytes of CRC code. Each driver board returned 1 byte for identification, 1 byte for a tick counter, 2+2 bytes representing the absolute angle θ used for sine-cosine generation and its estimation based on the currents, plus 2 bytes for the modulus of each of the currents I_d and I_q based on the estimated angle, alongside the voltages V_d and V_q, followed by a 2 bytes CRC code, and 2 bytes padding. Following the arrangement scheme detailed in Section II-D, each byte was carried in a slot having a size of 32 bytes (one per peripheral node) plus a header of 2 bytes. To the resulting 612 bytes, a debug field of 10 bytes, 26 bytes for the preamble and check field, and 2 bytes padding had to be added, bringing the total to 650 bytes. The entire frame consequently took 52 μs to be transferred. Each driver board and the PHY of the controller itself introduced about $\approx$ 1 μs of delay, so the frame started to be received back after $\approx$ 30 μs.

The speed at which data were interchanged between the controller and peripherals made it possible to minimize the amount of computation taking place on the driver boards. To exemplify the level of centralization that can be attained using REDBUS, the calculation of position data

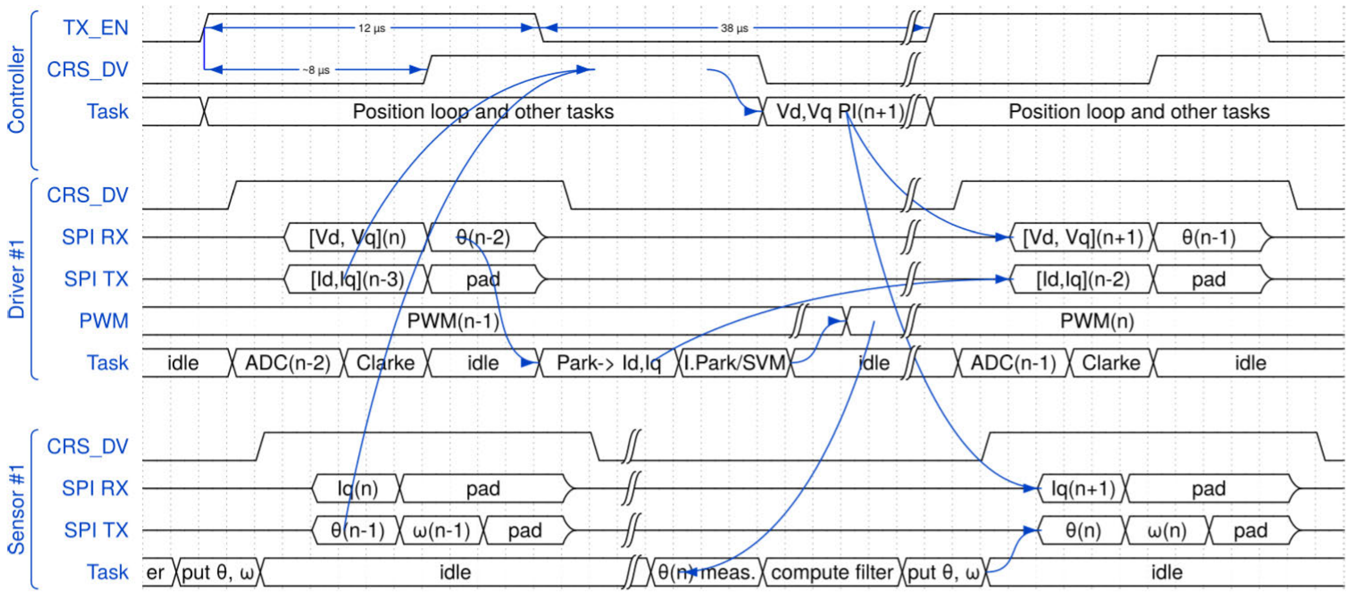


FIGURE 15. **Timing diagrams of the data exchange and processor tasks. The RMII signals** TX_EN **and** CRS_DV **are, respectively, asserted high during transmission and reception of an Ethernet frame by the PHY.**

and motion trajectories for all 32 motors was therefore carried out centrally by a single Cortex-M7 processor (type STM32H723Z, 550 MHz clock; STMicroelectronics SpA). Importantly, the controller does not require any specific hardware for managing the REDBUS network; consequently, it was possible to employ an off-the-shelf processor evaluation board containing a standard Ethernet MAC and PHY.

On the controller board, a custom firmware developed under the FreeRTOS operating system (v. 10.2.1) was run, and a hardware timer triggered the generation of an Ethernet frame every 100 μs, with jitter on the order of 10 ns. On the peripheral boards, a custom firmware peripheral ran directly on the hardware without an operating system.

To achieve adequate synchronization among the 32 motors and avoid propeller collisions, it turned out to be necessary to individually synchronize their PWM cycles so that each new position would be processed after each PWM cycle in a fully synchronous manner over all the devices. The average deviation due to the propagation delays was negligible compared to the cycle period, which was fixed at a frequency of 20 kHz, corresponding to 50 μs. The driver board firmware compared the precise time of the start of reception of the Ethernet frames to a local counter register used for PWM signal generation, and, for each frame, the value of this register was adjusted to ensure continued synchronization. In this way, all driver boards could generate PWM signals at a frequency exactly in unison with all the others. The position information was fed into the Ethernet frame so that all the positions of the 32 motors would be contained within a single frame and updated exactly every two PWM cycles synchronously over all the driver boards.

This experiment, devoid of closed-loop control, was designed primarily to allow measuring, under realistic circumstances, the propagation latencies of the Ethernet data frame along a large chain of peripherals. The data exchange between the MAC and the processor occurred almost simultaneously over all the peripherals, the time lag between one device and the next being consistently < 1 μs, as visible in Fig. 11. The setup could be operated at up to 5500 RPM, limited by the stepper motor electromechanical characteristics, without any collision or mechanical conflicts ever being observed, even after several hours of continuous rotation.

During each PWM cycle, each driver board measured the current through the motor windings, and the collected measurements were fed back into the Ethernet frame overwriting the received position information, and then sent to the controller board, which was used to implement continuous logging. An excellent level of synchronization stability could be appreciated among the measured currents: in Fig. 12a, they appeared superimposed to the point of being essentially indistinguishable from a single signal, whereas in Fig. 12b, magnifying a very short time interval, they could be distinguished. A quantitative analysis of the level of phase synchronization is given below, and a video recording of the experiment is provided as Supplementary Material.

B. FIELD-ORIENTED CONTROL

To further demonstrate the capabilities of REDBUS in a more complex scenario wherein the feedback information is used in real-time, we next considered the closed-loop control of a brushless DC (BLDC) motor. Typically, field-oriented control (FOC) of these motors requires the usage of multiple calculations to maintain the vectors of the driving currents in quadrature against the magnetic flux of the rotor. For example, a proportional-integral (PI)-based approach may be used to estimate the driving current and, from it, the torque to maintain the motor at a given speed set-point. The driving current I_q derived from the first PI controller, is then compared to the calculated I_{qm}, derived from a set of winding current measures, to generate the driving voltage vector module V_q. The I_{qm} value is derived from the two Clarke and Park calculations. Another PI is used to determine the V_d voltage vector modulus, comparing the calculated I_d from the measurements. The two voltage vector modules are transformed to modulation pulses with another set of calculations [23], [24], [32], [33], [34]. A block diagram implementing this algorithm is visible in Fig. 13a.

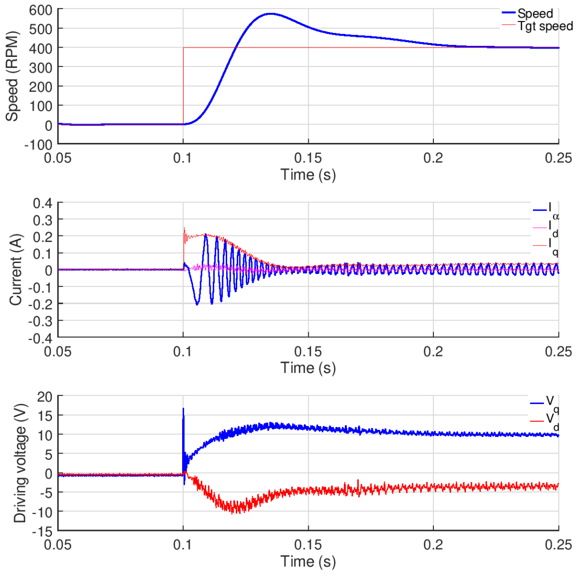


FIGURE 16. Spin-up of the stepper motor driven with field-oriented control (FOC).

The maximum cycle time acceptable for FOC algorithms depends on the modulation frequency of the inverter. Here, the same setting as above, namely 20 kHz, was maintained. The control calculations were distributed over several processors interconnected via a REDBUS network and controlled by a single unit to demonstrate the possibility of dividing the workload of algorithmic composition. Namely, the FOC block algorithm represented in Fig. 13a was split into two main blocks, the first, visible in Fig. 13b executed on the controller side, while the drive and sense portions of the second block were executed over two peripheral devices, visible in Fig. 13c.

The same Cortex-M7 processor indicated above was used to realize the controller and implement in fixed-point arithmetic the PI controllers determining the V_d, V_q, and I_q vector modules for up to eight motors. The control data were exchanged every 50 μs, corresponding to one period used in the modulation PWM by the inverters that drive the motors, as opposed to two periods in the previous section.

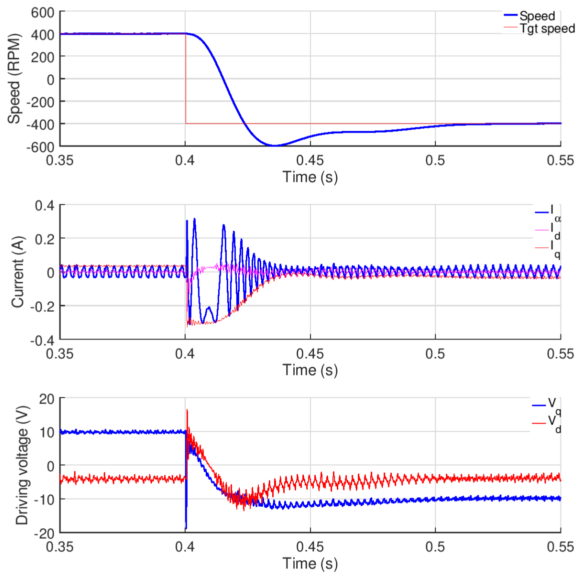


FIGURE 17. Sudden rotation direction reversal for the stepper motor driven with field-oriented control (FOC).

The driver boards were also the same as used in the previous subsection. However, in this case, as visible in Fig. 14, an additional board was used to interface an analog encoder (type ADA4570; Analog Devices Inc., Norwood MA) based on a magnetic ring and magnetoresistive sensors that convert the magnetic field into analog sine-cosine signals. Excerpts of the microcontroller source codes running on three nodes are provided for reference as Supplementary Materials.

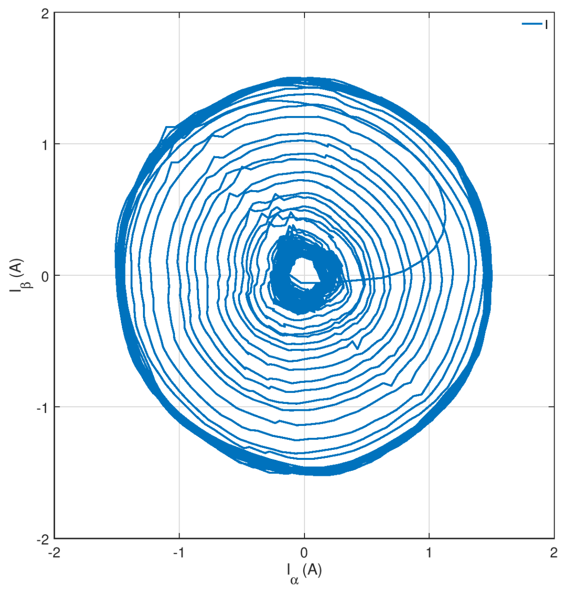


FIGURE 18. Winding currents, shown after applying the Clarke transform, immediately following direction reversal of the brushless DC (BLDC) motor and during spin-up to 4000 RPM.

The motor driver board received from the controller the V_d, V_q driving parameters, and the θ angle, derived from the tracking filter running on the encoder board, such that the subsequent PWM cycle would be updated based on the new values of these parameters. The driver board returned to the controller the I_d and I_q currents after computing the Clarke and Park transforms to convert the three measured stationary currents into two orthogonal vectors in the rotating reference frame. In turn, the encoder board received the commanding current I_q, which was used by the tracking filter to predict the next position, corrected by the measured angle θ. The filtered angle θ and the angular speed ω were sent back to the controller. The data from the driver board and the encoder board were used by the PI controllers running in the controller to compute the subsequent values of V_d and V_q. The task timing diagrams are shown in Fig. 15.

To demonstrate the versatility of this control approach and its robustness in the implementation based on REDBUS, two types of motors were driven: the same stepper motor described above, used as a brushless motor with 50 magnetic pole pairs, and an actual brushless motor having 4 pairs of magnetic poles. These two types of permanent magnet synchronous motors are structurally similar, and stepper motors are, in effect, specialized brushless motors. They can be controlled either in an open or closed loop. A typical stepper motor presents 50 magnetic pole pairs, and each pole has an angle of 7.2 degrees. Using a magnetic ring with the same number of radial poles, measuring the flux angle for each "step" of the motor is possible. Here, a special magnet with radial poles was been used to measure the flux angle. The magnet was directly clamped onto the motor shaft. The number of poles was the same as the poles of the motor so that the magnetic flux angle of the rotor could be precisely measured.

The FOC scheme used was conceptually the same. By measuring the magnetic flux angle of the rotor, it aimed to constantly keep the winding current vector 90 degrees ahead of the magnetic flux vector, maximizing the output torque in this way. The brushless motor had three phases and was controlled using space-vector-modulation, which generated three driving signals directly from V_α and V_β. Two-phase stepper motors can be controlled in a similar mode, with the exception that the Clarke transform is not needed, as the I_a and I_b measure are already orthogonal, thus they coincide with $I\alpha$ and $I\beta$. Moreover, the V_α and V_β values are used directly to modulate the two PWM signals for the two phases without the need for space-vector-modulation. The current measures were taken by the driver board and sent to the controller, as said, every 50 μs.

The encoder board inserted the magnetic flux angle measure within each Ethernet frame. It furthermore provided a speed value based on a tracking filter that estimated the angle based on the driving I_q current and corrected it with the measured one. For controlling the stepper motor, the PI coefficients were $K_\mathrm{p} = 8$ and $K_\mathrm{i} = 0.2$ for V_d and V_q, and $K_\mathrm{p} = 0.007$ and $K_\mathrm{i} = 0.00001$ for the speed.

When changing the target speed from 0 to a positive value, the PI reacted quickly, with I_q growing almost instantly, as visible in Fig. 16. As could be seen, the PIs of the FOC algorithm successfully processed the V_d and V_q driving voltages so as to keep I_d to a null value and I_q to the target value. The motor torque had the maximum efficiency, and the target speed was reached in $\approx$ 0.02 s, implying an acceleration of $\approx$ 19000 RPM/s. When the target speed was instantaneously inverted in sign, as shown in Fig. 17, the system immediately reacted, and in a very short time, the motor reversed its rotating direction and reached the new target speed with minimal overshoot. A high I_q value was used for braking the rotating load and quickly reaching the new target value. The motor took $\approx$ 23.5 ms to reverse direction, implying an acceleration of about $\approx$ 34100 RPM/s.

The brushless motor (type QBL4208-41-04-006; Trinamic AG, Hamburg, Germany) had 4 magnetic pole pairs, 3 phases, and was rated for 1.8 A and 24 V, maximum speed 4000 RPM. It was driven following the same scheme, however, replacing the ring with one containing 4 poles as the motor itself. For this experiment, the PI coefficients were $K_\mathrm{p} = 0.3$ and $K_\mathrm{i} = 0.012$ for V_d and V_q, and $K_\mathrm{p} = 0.4$ and $K_\mathrm{i} = 0.0005$ for the speed. The stability of the currents is shown in Fig. 18 at the nominal spinning rate. Stable operation could be attained up to twice this rate.

C. LEVEL OF TIMING DETERMINISM

To confirm the level of determinism attained in these realistic experimental scenarios, some oscilloscope measurements were taken on the test systems visible in Fig. 10 and Fig. 14. The frame synchronization signals were probed on the one side, at the PHY of the controller (type LAN8742A, Ref. [35]) and on the other side, at the PHY of a selected peripheral (type KSZ8081RNB, Ref. [36]). As reported in Table 4, the propagation latency between adjacent devices was consistently under 1 μs and, accordingly, the turn-around time over the 32 peripherals was $\approx$ 32 μs. Corresponding representative waveforms are shown in Fig. 11 and Fig. 19. As discussed above, given the low-level nature of REDBUS, all timings are predetermined by design, there are no load fluctuations possible nor external influences by other traffic, so the amount of jitter is purely determined by hardware factors, primarily the PHY performance as exemplified in Table 4. Due to the daisy-chaining architecture, jitter scales summatively with the number of nodes on the bus.

TABLE 4. Experimentally measured latencies between PHY devices, representing the time-delay from the `RMII_TXEN` signal rising edge on the source PHY to the `RMII_CRS/DV` rising edge on a target PHY. The "span" column represents the number of peripherals between source and destination.

Source PHY type	Target PHY type	Span	Min (ns)	Average (ns)	Max (ns)	Jitter (ns)
LAN8742A	LAN8742A	none	466	497	528	62
LAN8742A	KSZ8081RNB	none	747	756	765	18
KSZ8081RNB	KSZ8081RNB	none	972	991	1010	38
LAN8742A	KSZ8081RNB	32	32130	32330	32530	400

Further confirmation of the attained level of timing stability was obtained by analyzing the raw motor current

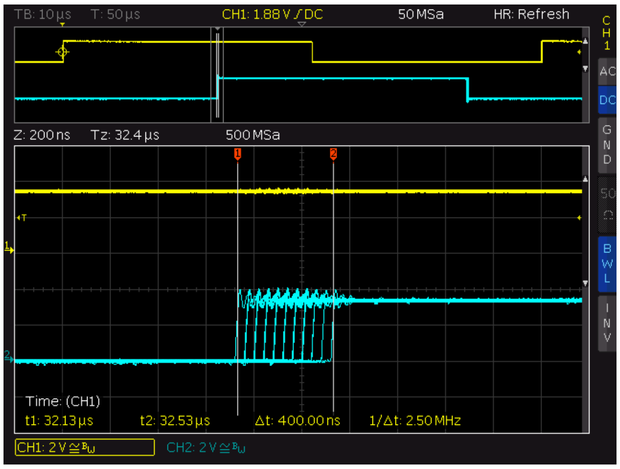


FIGURE 19. Jitter measurement in the REDBUS interconnection topology. Infinite-persistence capture, triggered on the transmission of the frame by the controller, depicting the fluctuation in the arrival time of the same. After traversing all 32 devices, the accumulated jitter is ≈ 0.4 μs.

waveforms, as directly sampled by the analog-to-digital converters and returned in the Ethernet frames. In particular, phase synchronization, a measure commonly used for analyzing the temporal dynamics of non-linear systems, was calculated. After removal of the average, each current $I_k(t)$, with $k = 1 \ldots N$ where N is the number of signals, was turned into the corresponding analytic signal with

$$\psi_k(t) = I_k(t) + j\tilde{I}_k(t) = A_k(t)e^{j\theta_k(t)}, \tag{9}$$

where $j = \sqrt{-1}$, $\tilde{u}(t)$ denotes the Hilbert transform of $u(t)$

$$\tilde{u}(t) = \frac{1}{\pi}\text{p.v.}\left[\int_{-\infty}^{\infty} \frac{u(\tau)}{t-\tau}\mathrm{d}\tau\right], \tag{10}$$

with p.v. signifying the Cauchy principal value of the integral, and $A_k(t)$ and $\theta_k(t)$ denote, finally, the instantaneous amplitude and phase. As detailed in Ref. [37], from the latter, the phase synchronization value, ranging between 0 for complete asynchrony to 1 for perfect synchronization, can be calculated between two signals i and k, with

$$r_{i,k} = |\langle e^{j(\theta_i - \theta_k)}\rangle_t|. \tag{11}$$

For the experiment involving the ensemble of $N = 32$ stepper motors driven in an open loop configuration, the instantaneous phase $\theta_k(t)$ was calculated for each motor, and a 32 × 32 matrix R of phase synchronization values was obtained. Excluding the diagonal elements, the average and standard deviation were, respectively, 0.991 and 0.007, while the minimum was 0.964. For the experiments with field-oriented control, the r value was calculated only once, between the A and B phase currents of the motor. The corresponding phase synchronization values were 0.993 and 0.989, confirming a near-perfect entrainment attained and maintained over the Ethernet-based control medium.

D. COMPARISON TO CONVENTIONAL ARCHITECTURE

Finally, an additional experiment was performed to provide a representative baseline of the performance of standard IEEE802.3 Ethernet under the same situation. It involved one controller exchanging frames with 8 peripheral nodes, not according to the REDBUS daisy-chain configuration but to the standard star-based physical interconnection architecture according to the 100BASE-T specifications, as visible in Fig. 1b. The same hardware (type STM32H723Z Cortex-M7 processor with PHY type LAN8742A) was retained for all the devices (controller and peripherals), and a high-end 1000BASE-T capable switch (type GO-SW-16G; D-Link Corporation, Taipei) operating in store-and-forward mode was selected. For this additional experiment, only 8 peripheral nodes were used instead of 32, as in Section III-A (as detailed below, even with a lower number of peripherals, the gap in performance was well-evident).

To enable a fair comparison, the controller transmitted its frame in broadcast mode simultaneously to all the peripheral nodes in this experiment. The size of this frame was kept equal to 650 bytes, that is, the same as in Section III-A. Oscilloscope-based timing measurements of the frame synchronization signals between the controller and the peripheral nodes readily demonstrated that, as a consequence of its internal architecture, the switch introduced a strikingly longer lag compared to the propagation latency between adjacent REDBUS devices. First, each frame had to be completely received before it could be forwarded, incurring a delay corresponding to its transmission time, in this case, ≈ 52 μs. Second, an internal processing time of ≈ 10 μs had to be added. Since these measurements pertained only to the frame from the controller to the peripheral nodes, which was broadcast to all, they were inherently unaffected by the number of peripheral nodes actually present. The resulting measured forwarding time, ≈ 62 μs, was, therefore, drastically longer not only compared to the propagation latency between adjacent REDBUS devices, namely < 1 μs, but even compared to the entire REDBUS chain of 32 devices, namely ≈ 32 μs.

Each peripheral node responded with the same amount of payload data, namely 18 bytes, considered in Section III-A. However, in this case, each response had to be encapsulated individually: considering the requirement of a minimum payload size of 64 bytes set by the IEEE802.3 standard and the preamble and check field for a total of 26 bytes, this resulted in a frame size of 90 bytes. Therefore, the total amount of data returned to the controller, namely 720 bytes, was eventually larger, even though only 8 instead of 32 peripheral nodes were present. According to oscilloscope-based measurements of the frame synchronization signals, the first response started to be received by the controller on average after ≈ 139 μs, denoting a latency over 4 times longer than that observed using the REDBUS topology. The time lag until complete reception of the response from all peripheral nodes, which was ≈ 84 μs for REDBUS using 32 nodes, was elevated to $\approx$ 196 μs for 8 nodes, and would have been $\approx$ 367 μs for 32 nodes.

Not less importantly, as shown in Fig. 20, the accumulated jitter considering the time of starting to receive the first response from the peripheral nodes was $\approx$ 2.9 μs, that is, more than 7 times what was observed using the REDBUS topology. Furthermore, the order of reception of the peripheral node responses, albeit not fully random owing to the internal arbitration system of the switch, exhibited considerable stochasticity. For example, the first reply to be received belonged 38%, 24%, 22%, 13%, 1%, 2%, <1%, and <1% of the times to the peripheral nodes attached at ports 1 through 8, respectively.

Therefore, using the standard topology in Fig. 1b compared to REDBUS as in Fig. 1a, resulting not only in considerably worse overall timing performance but also in a severely degraded level of determinism.

To date, store-and-forward Ethernet switching is vastly predominant, owing to its advantages in terms of error checking and prevention of error propagation, and the majority of industrial Ethernet switches operate in this mode (e.g., Ref. [38] for an example of a contemporary high-end series of industrial Ethernet switches). However, for completeness, it should be mentioned that cut-through switching is also possible, whereby the switch initiates frame forwarding immediately after receiving the destination MAC address, thus drastically reducing the latency. Such a mode, however, remains primarily confined to data centers and high-performance computing systems and, owing to the reliability issues, is not used in industrial scenarios [39], [40]. Other techniques to improve the forwarding performance, such as frame preemption, are also available but rely on specific TSN extensions of the IEEE802.1 standard that, as discussed above, incur considerable additional complexity [9], [18], [41]. For these reasons, the comparison was conducted using a conventional switch representative of current commercial and industrial devices, thus operating in store-and-forward mode.

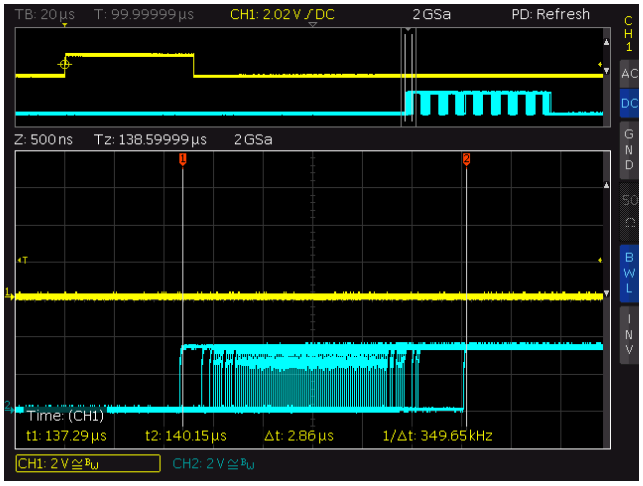


FIGURE 20. **Jitter measurement in the standard IEEE802.3 Ethernet star-based physical interconnection. Infinite-persistence capture, triggered on the transmission of the frame by the controller, depicting the fluctuation in the arrival time of the same. After traversing the switch twice and one peripheral device, the accumulated jitter is $\approx$ 2.9 μs.**

IV. DISCUSSION

This work has demonstrated the possibility of obtaining compelling real-time remote control performance under a minimalist configuration, whereby the drive and sense signals are directly embedded in Ethernet frames, which flow through a chain topology, closed by a controller device, thus forming a ring. While the present experiments only provide a proof of concept, the timing performance recorded in realistic experiments on multi-axis and multi-sensor motor control was, in principle, compatible with the requirements of the most demanding scenarios encountered in the design of industrial machinery and high-speed vehicles [11], [12]. At the same time, the implementation could be accomplished using low-cost, off-the-shelf hardware consisting of broad-market Ethernet PHY integrated circuits alongside some of the smallest FPGAs available commercially [9], [22], [36]. These two aspects hallmark the key advantages of the proposed architecture, which were further underlined by experimental comparison with the performance of standard IEEE802.3 star-based physical interconnection.

It appears, in particular, worthwhile to highlight three aspects and consider their impact on scaling. First, when using the standard IEEE802.3 topology, there is a timing penalty with respect to REDBUS because the entire frame must be received before it can start to be retransmitted. This penalty scales linearly with the frame size. Second, in the case of typical store-and-forward architectures, additional latency is introduced by the internal buffering, resulting in a lag that increases linearly with the number of switches that need to be transversed. Third, each Ethernet frame requires a fixed number of bytes as preamble and check fields and imposes a minimum payload size, implying a data overhead that increases with the number of peripheral devices, potentially in a steep manner if the actual number of payload bytes to be returned is small. Crucially, the REDBUS architecture avoids all these three penalties since retransmission occurs on the fly, no switches are required, and a single Ethernet frame, in principle, carries the payload for all devices, thus sharing the associated overhead and typically consuming all of the minimum number of payload bytes.

It needs to be noted that the above comes at the price of versatility since the architecture implies that the bus timings are homogeneous across all nodes, as reflected in the two experiments presented in this study. While the presence of manifold sensors and actuators of the same type is common within individual equipment, for example, in multi-axis robots, the inability to handle heterogeneous timing requirements underlines the differences between REDBUS and proper fieldbus protocols such as EtherCAT and PROFINET. In summary, the scalability profile of REDBUS is primarily driven by the required frame rate and latency, which are straightforwardly determined by the number of devices on the chain and the associated amount of data. The jitter is dominated by hardware factors, each node introducing a relatively fixed amount. Given its small entity, jitter is likely to have less impact on most applications, but it should also be considered when determining the maximum number of nodes for a hypothetical application. Given the low-level nature of REDBUS and its intended use, it appears implausible that instances with more than a few tens of nodes would be practically relevant.

The logical notion underlying REDBUS is closely related to that of token-passing, which was the basis of media access negotiation in some early generations of computer networks, such as the IEEE 802.5 Token Ring and ANSI 3T9.5 Fiber Distributed Data Interface (FDDI) standards [42], [43]. However, REDBUS realizes this scheme using contemporary, widely available IEEE 802.3 Ethernet hardware, wired in an atypical manner such that, instead of being distributed through a switch, frames flow along a chain connection [4], [19]. In this paper, it was shown that the media sensing and other PHY-level features of this standard perfectly support such an interconnection scheme, the only changes being required at the MAC level to realize in an optimal way the required interchange and addressing operations. Crucially, the chain of peripherals appears to the controller as fully indistinguishable from an individual device, hallmarking one advantage of this scheme, that is, the possibility to attach the chain to any network card based on a standard MAC.

Conceptually, it is worth noting that REDBUS and EtherCAT, through inserting (or replacing) process data on the fly at fixed locations in an Ethernet frame, essentially implement a form of time-domain multiplexing, though the timing of the frame generation is not isochronous. Time-domain multiplexing, originally introduced in isochronous telecommunications, of which synchronous optical networks are an example, finds applications in countless standards and contexts [44]. For instance, in the IEEE 802.15.4 wireless standard, superframes are used as a means of efficiently pacing the transmission of data from multiple sensors while avoiding contention, and the aggregation of multiple frames has been proposed as a means of increasing the throughput for critical monitoring applications [45], [46]. In IEEE 802.3 Ethernet, the usage of jumbo frames to enhance bandwidth efficiency is a possibility [4].

Unlike complex standards such as PROFINET and EtherCAT, and to a lesser extent also Modbus TCP, REDBUS is entirely devoid of features related to device abstraction, identification, interoperability, and security [5], [6]. As such, and due to its topology, it is inherently unsuitable for use as a large-scale fieldbus, and is not intended for such uses. Rather than a disadvantage, this should be viewed as the distinguishing feature of this protocol, which is complementary rather than adversarial to them. Owing to its architecture, REDBUS has low flexibility and requires the entire system, the related signals, and their encapsulation to be defined a priori and embedded in the configuration of the MAC of each peripheral. At the same, its complexity and cost of implementation are drastically lower. For example, REDBUS could be conveniently utilized as a processor bus extension within a multi-board system, which is, in turn, attached to the rest of an industrial network through a regular large-scale field bus. It is noteworthy that the highly deterministic timings together with the implementation simplicity of an archaic network standard based on token passing, ARCNet, running at 2.5 Mbps, support its continued popularity in industrial automation, particularly in Japan [47].

From the standpoint of security, REDBUS, being a low-level bus, is entirely devoid of authentication, encryption, and trust management features. It should be noted that its security profile has favorable aspects because, unlike other Ethernet-based protocols, REDBUS at the basis prevents the use of other Ethernet-based protocols, due to the atypical and hardwired topology and the hardly predetermined payload structure. Moreover, using a tailored MAC handling the data formatting in hardware renders it impossible to attack the processor by crafting malformed frames. As such, it appears ill-suited as a means of propagating an attack. Insofar as peripheral nodes are expected to be fully self-contained microcontrollers or FPGAs devoid of other interconnection means, the security of the REDBUS chains depends on the integrity of the controller. Man-in-the-middle type attacks with malicious on-the-fly data replacement are possible but would require physical access and the addition of a node in place with the equipment where the bus is installed. While the data integrity profile appears comparable with other low-level protocols, including mainstream implementations of CAN, Modbus, and custom links over RS232 and RS485, formal modeling of the security aspects is always necessary for critical applications [48], [49].

Two fundamental aspects of any control system are safety and reliability [11], [12], [50]. On the one hand, REDBUS is a very low-level communication scheme and, therefore, does not implement specific functional safety functions in itself. On the other hand, it should be compared, at least at the physical level, to existing standards in terms of reliability. The chain/ring topology inherently introduces a vulnerability since the failure of any node leads to the entire network becoming inoperable. While star topologies are more resilient to node failure except for the hub, similar situations arise, for example, in chains of EtherCAT devices. As indicated above, REDBUS is inherently unsuitable for factory-floor integration level but primarily intended to integrate homogeneous sensors and actuators within predesigned equipment. In most such scenarios, failure of a single node, for example, loss of control of one axis in a robot, inherently leads to machine downtime. Therefore, the reliability penalty introduced by the interconnection topology is limited. Moreover, while there is no recovery mechanism in case a frame is lost, the fast update rate inherently mitigates the consequences. In situations where the probability of node failure or frame loss cannot be sufficiently lowered, two possible mitigation strategies are available. One involves having an automatic shunting mechanism directly shorting the transmit and receive pairs to bypass a failed node, recalling what has been implemented for other standards such as FDDI [51], [52]. Another requires having two PHY devices per node so that direct connectivity is established with both first and second neighbors along the chain, analogously to dual-port EtherCAT nodes [5], [15]. In mission-critical applications, the reliability impact of the interconnection topology needs to be formally modeled and mitigated.

In summary, REDBUS offers properties and advantages that are complementary to the existing Ethernet standards currently in use in the industrial and associated fields. The ease of its implementation also renders it well-suited for control and sensing applications in the realization of research equipment. More generally, the demonstration that daisy-chaining Ethernet PHYs delivers highly deterministic communication performance should inspire further research and the development of new standards in this area.

ACKNOWLEDGMENT

The authors would like to thank Gianluca Mariani (Lattice Semiconductor Srl, Assago, Italy) for helpful discussions and Roberto Zinna (Elettroprogetti SAS, Rho, Italy) for helpful discussions and assistance in hardware design and realization. Parts of the architecture and code described herein are covered by European patent EP2930896 and its national filings and the Italian patent IT0001423488: permission for use in realizing the system disclosed herein toward academic research by not-for-profit institutions is granted under all the terms of the GNU General Public License version 3. Ludovico Minati contributed to this work during the period from March 2020 to May 2023, while intermittently collaborating with DVE Progettazione Elettronica as an Independent Researcher and an External Advisor and in part also collaborating with a subsidiary of Lattice Semiconductor Inc.

REFERENCES

[1] P. Marshall and J. Rinaldi, *Industrial Ethernet: How to Plan, Install and Maintain TCP/IP Ethernet Networks, the Basic Reference Guide for Automation and Process Control Engineers*, 2nd ed., Pittsburgh PA, USA: International Society of Automation, 2005.

[2] J.-D. Decotignie, "The many faces of industrial Ethernet [past and present," *IEEE Ind. Electron. Mag.*, vol. 3, no. 1, pp. 8–19, Mar. 2009.

[3] J. A. Kay, R. A. Entzminger, and D. C. Mazur, "Industrial Ethernet-overview and best practices," in *Proc. Conf. Rec. Annu. Pulp Paper Ind. Tech. Conf.*, Jun. 2014, pp. 18–27.

[4] *IEEE Standard for Ethernet*, Standard 802.3-2022 (Revision of IEEE Std 802.3-2018), 2022, pp. 1–7025.

[5] D. Jansen and H. Buttner, "Real-time Ethernet: The EtherCAT solution," *Comput. Control Eng.*, vol. 15, no. 1, pp. 16–21, Feb. 2004.

[6] J. Jasperneite and J. Feld, "PROFINET: An integration platform for heterogeneous industrial communication systems," in *Proc. IEEE Conf. Emerg. Technol. Factory Autom.*, vol. 1, Catania, Italy, Sep. 2005, pp. 815–822.

[7] G. Prytz, "A performance analysis of EtherCAT and PROFINET IRT," in *Proc. IEEE Int. Conf. Emerg. Technol. Factory Autom.*, Hamburg, Germany, Sep. 2008, pp. 408–415.

[8] G. Cena, I. C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "Evaluation of EtherCAT distributed clock performance," *IEEE Trans. Ind. Informat.*, vol. 8, no. 1, pp. 20–29, Feb. 2012.

[9] Y. Seol, D. Hyeon, J. Min, M. Kim, and J. Paek, "Timely survey of time-sensitive networking: Past and future directions," *IEEE Access*, vol. 9, pp. 142506–142527, 2021.

[10] Z. Satka, M. Ashjaei, H. Fotouhi, M. Daneshtalab, M. Sjödin, and S. Mubeen, "A comprehensive systematic review of integration of time sensitive networking and 5G communication," *J. Syst. Archit.*, vol. 138, May 2023, Art. no. 102852.

[11] P. Marwedel, *Embedded System Design: Foundations of Cyber-Physical Systems, and the Internet of Things*. Cham, Switzerland: Springer, 2021.

[12] M. Maggio, "Real-time implementation of control systems," in *Cyber-Physical Systems: A Reference*. Berlin, Germany: Springer, 2013.

[13] *IEEE Standard for Local and Metropolitan Area Networks-Timing and Synchronization for Time-Sensitive Applications*, Standard 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011, 2020, pp. 1–421.

[14] Microchip Technology Inc. *EtherCAT Software Framework User's Guide (DS50003044A)*. Microchip Technol. Inc. Accessed: Aug. 16, 2024. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/EtherCATSoftware-Framework-Users-Guide-50003044A.pdf

[15] Beckhoff Automation GmbH. *Application Note ET9300 (EtherCAT Slave Stack Code)*. [Online]. Available: https://download.beckhoff.com/download/Document/io/ethercatdevelopment-products/an_et9300_v1i10.pdf

[16] R. Sanchez, L. Raptis, and K. Vaxevanakis, "Ethernet as a carrier grade technology: Developments and innovations," *IEEE Commun. Mag.*, vol. 46, no. 9, pp. 88–94, Sep. 2008.

[17] K. Fouli and M. Maier, "The road to carrier-grade Ethernet," *IEEE Commun. Mag.*, vol. 47, no. 3, pp. S30–S38, Mar. 2009.

[18] N. Finn, "Introduction to time-sensitive networking," *IEEE Commun. Standards Mag.*, vol. 2, no. 2, pp. 22–28, Jun. 2018.

[19] *IEEE Standards for Local and Metropolitan Area Networks: Supplement-Media Access Control (MAC) Parameters, Physical Layer, Medium Attachment Units, and Repeater for 100 Mb/s Operation, Type 100BASE-T (Clauses 21-30)*, Standard 802.3u-1995 (Supplement to ISO/IEC 8802-3: 1993; ANSI/IEEE Std 802.3, 1995, pp. 1–415.

[20] G. Brugnoni, "Ethernet network device and local Ethernet network," European Patent Patent EP 2 930 896, Oct. 1, 2015.

[21] Analog Devices Inc. *ADI-SPI Technical Specification—Serial Control Interface Standard*. Analog Devices Inc. Accessed: Aug. 16, 2024. [Online]. Available: https://wiki.analog.com/_media/resources/technical-guides/adispi_rev_1p0_customer.pdf

[22] Lattice Semiconductor. *MachXO2 Family Data Sheet (FPGA-DS-02056-4.1)*. Accessed: Aug. 16, 2024. [Online]. Available: https://www.latticesemi.com/-/media/LatticeSemi/Documents/DataSheets/MachXO23/FPGA-DS-02056-4-1-MachXO2-Family-Data-Sheet.ashx

[23] R. Marino, P. Tomei, and C. M. Verrelli, *Induction Motor Control Design*. London, U.K.: Springer, 2010.

[24] Texas Instruments Inc. *Field Orientated Control of 3-Phase AC-Motors*. Texas Instrum. Inc. [Online]. Available: https://www.ti.com/lit/an/bpra073/bpra073.pdf

[25] NXP Semiconductors Inc. *TJA1100 PHY Datasheet*. Accessed: Aug. 16, 2024. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/TJA1100.pdf

[26] Texas Instruments Inc. *DP83620 PHY Datasheet*. Texas Instrum. Inc. Accessed: Aug. 16, 2024. [Online]. Available: https://www.ti.com/lit/ds/symlink/dp83620.pdf

[27] Texas Instruments Inc. *DP83826 Deterministic, Low-Latency, Low-Power, 10/100 Mbps, Industrial Ethernet PHY*. Texas Instrum. Inc. Accessed: Aug. 16, 2024. [Online]. Available: https://www.ti.com/lit/ds/symlink/dp83826i.pdf

[28] Texas Instruments Inc. *TLK10xL PHY Datasheet*. Texas Instrum. Inc. Accessed: Aug. 16, 2024. [Online]. Available: https://www.ti.com/lit/ds/symlink/tlk106l.pdf

[29] Analog Devices Inc. *ADIN1130 PHY Datasheet*. Analog Devices Inc. Accessed: Aug. 16, 2024. [Online]. Available: https://www.analog.com/media/en/technical-documentation/

[30] STMicroelectronics SpA. *STE101P PHY Datasheet*. Accessed: Aug. 16, 2024. [Online]. Available: https://www.st.com/resource/en/datasheet/ste101p.pdf

[31] T. Munk, "Tests with interlocking and overlapping propellers," Maritime Archive TU Delft Library, Delft, The Netherlands, Tech. Rep. Hy-12, 1969.

[32] O. Barambones and P. Alkorta, "A robust vector control for induction motor drives with an adaptive sliding-mode control law," *J. Franklin Inst.*, vol. 348, no. 2, pp. 300–314, Mar. 2011.

[33] P. Ramesh and R. Prathyusha, "Field oriented control of permanent magnet synchronous motor," *Int. J. Comput. Sci. Mobile Comput.*, vol. 3, no. 3, pp. 269–275, Mar. 2014.

[34] D. Wilson. *Motor Control Compendium*. Accessed: Aug. 16, 2024. [Online]. Available: https://www.ti.com/download/trng/docs/c2000/TI_MotorControlCompendium_2010.pdf

[35] Microchip Technology Inc. *LAN8742A PHY Datasheet*. Microchip Technol. Inc. Accessed: Aug. 16, 2024. [Online]. Available: https://www.microchip.com/en-us/product/LAN8742A

[36] Microchip Technology Inc. *KSZ8081RNB PHY Datasheet*. Microchip Technol. Inc. Accessed: Aug. 16, 2024. [Online]. Available: https://ww1.microchip.com/downloads/en/devicedoc/ksz8081mnx-rnb.pdf

[37] S. Boccaletti, J. Kurths, G. Osipov, D. L. Valladares, and C. S. Zhou, "The synchronization of chaotic systems," *Phys. Rep.*, vol. 366, nos. 1–2, pp. 1–101, Aug. 2002.

[38] Cisco Systems Inc. *Data Sheet, Cisco Catalyst IE3400 Rugged Series*. Cisco Syst. Inc. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-ie3400-heavy-duty-series/datasheet-c78-742313.html

[39] H. Ahmadi and W. E. Denzel, "A survey of modern high-performance switching techniques," *IEEE J. Sel. Areas Commun.*, vol. 7, no. 7, pp. 1091–1103, Sep. 1989.

[40] N. Zilberman, L. Dudziak, M. Jadczak, T. Parks, A. Rietmann, V. Safronov, and D. Zuo, "Cut-through network switches: Architecture, design and implementation," Univ. Cambridge Computer Lab., Cambridge, U.K., Tech. Rep. UCAM-CL-TR-928, 2018.

[41] M. Ashjaei, M. Sjödin, and S. Mubeen, "A novel frame preemption model in TSN networks," *J. Syst. Archit.*, vol. 116, Jun. 2021, Art. no. 102037.

[42] N. C. Strole, "The IBM token-ring network—A functional overview," *IEEE Netw.*, vol. 1, no. 1, pp. 23–30, Jan. 1987.

[43] D. Chen, V. C. S. Lee, and E. Chan, "On the ability of the FDDI-M protocol to support real-time traffic," in *Proc. 5th Int. Conf. Real-Time Comput. Syst. Appl.*, Hiroshima, Japan, 1998, pp. 51–57.

[44] D. Zheng and E. F. Y. Young, "An integrated circuit partitioning and TDM assignment optimization framework for multi-FPGA systems," in *Proc. 28th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Tokyo, Japan, Jan. 2023, pp. 522–526.

[45] M. Akbar, H. Yu, and S. Cang, "IEEE 802.15.4 frame aggregation enhancement to provide high performance in life-critical patient monitoring systems," *Sensors*, vol. 17, no. 2, p. 241, Jan. 2017.

[46] H. Amirinia and R. Liscano, "Optimization of the IEEE 802.15.4 superframe for clustered WSNs using differential evolution," in *Proc. IEEE 32nd Annu. Int. Symp. Pers., Indoor Mobile Radio Commun. (PIMRC)*, Helsinki, Finland, Sep. 2021, pp. 1316–1322.

[47] G. Thomas, "ARCNET never received enough credit [history]," *IEEE Ind. Appl. Mag.*, vol. 23, no. 5, pp. 7–13, Sep. 2017.

[48] F. D. Fagundes and M. J. da Cunha, "Industrial network security," *J. Control, Autom. Elect. Syst.*, vol. 33, pp. 1177–1187, Aug. 2022.

[49] A. Alfardus and D. B. Rawat, "Evaluation of CAN bus security vulnerabilities and potential solutions," in *Proc. 6th Int. Conf. Women Data Sci. Prince Sultan Univ. (WiDS PSU)*, Riyadh, Saudi Arabia, Mar. 2023, pp. 90–97.

[50] G. Peserico, A. Morato, F. Tramarin, and S. Vitturi, "Functional safety networks and protocols in the industrial Internet of Things era," *Sensors*, vol. 21, no. 18, p. 6073, Sep. 2021.

[51] S. F. Ralph, O. J. Ukrainsky, R. H. Schellack, and L. Weinberg, "Alternate path FDDI topology," in *Proc. 17th Conf. Local Comput. Netw.*, Nov. 1992, pp. 168–177.

[52] B. Chen, S. Kamat, and W. Zhao, "Fault-tolerant real-time communication in FDDI-based networks," in *Proc. 16th IEEE Real-Time Syst. Symp.*, Pisa, Italy, Dec. 1995, pp. 141–150.

GABRIELE BRUGNONI completed state technical training in electrotechnics and industrial electronics, in 1988, and then established DVE Progettazione Elettronica, in 2000. He has provided technical consulting and development services for multiple clients in Italy and Europe, specializing in highly multi-axis motor control systems, particularly for electro-optic applications and real-time control and high-reliability embedded systems using field-programmable gate array (FPGA) technology from multiple vendors. His current research interest includes FPGA-based systems for real-time control applications in critical applications.

LUDOVICO MINATI (Senior Member, IEEE) received the Ph.D. degree in neuroscience from the Brighton and Sussex Medical School, Falmer, U.K., in 2012, the D.Sc. (Doktor Habilitowany) degree in physics from the Institute of Nuclear Physics, Polish Academy of Sciences, Kraków, Poland, in 2017, and the M.B.A. degree in technology management from The Open University, Milton Keynes, U.K., in 2021. Until 2023, he was a Specially Appointed Associate Professor with the Institute of Innovative Research, Tokyo Institute of Technology, Tokyo, an Affiliate Research Fellow with the Center for Mind/Brain Sciences, University of Trento, Trento, Italy, and a Freelance Research and Development Consultant. He is currently a Professor, an Outstanding Young Talent, and the Director of the Interdisciplinary Nonlinear Dynamics Laboratory, School of Life Science and Technology, University of Electronic Science and Technology of China, Sichuan, China. He has authored more than 160 articles and several patents. His research interests include nonlinear dynamical systems, chaotic oscillators, reconfigurable analog and digital computing, analog integrated circuits, advanced techniques for biosignal analysis, brain-machine/computer interfaces, and robotics. He is an European Engineer (Eur. Ing.), a Chartered Engineer (C.Eng.), and a member of the Institution of Engineering and Technology, U.K. He is also a member of the Institute of Electronics, Information, and Communication Engineers (IEICE) and the Institute of Electrical Engineers (IEE) of Japan.