Enabling Time-Synchronized Hybrid Networks With Low-Cost IoT Modules

Alexey M. Romanov, *Senior Member, IEEE*, Francesco Gringoli, *Senior Member, IEEE*, Kamil Alkhouri, Pavel E. Tripolskiy, and Axel Sikora

***Abstract*—Precisely synchronized communication is a major precondition for many industrial applications. At the same time, hardware cost and power consumption need to be kept as low as possible in the Internet of Things (IoT) paradigm. While many wired solutions on the market achieve these requirements, wireless alternatives are an interesting field for research and development. This article presents a novel IEEE802.11n/ac wireless solution, exhibiting several advantages over state-of-the-art competitors. It is based on a market-available wireless System on a Chip with modified low-level communication firmware combined with a low-cost field-programmable gate array. By achieving submicrosecond synchronization accuracy, our solution outperforms the precision of low-cost products by almost four orders of magnitude. Based on inexpensive hardware, the presented wireless module is up to 20 times cheaper than software-defined-radio solutions with comparable timing accuracy. Moreover, it consumes three to five times less power. To back up our claims, we report data that we collected with a high sampling rate (2000 samples per second) during an extended measurement campaign of more than 120 h, which makes our experimental results far more representative than others reported in the literature. Additional support is provided by the size of the testbed we used during the experiments, composed of a hybrid network with nine nodes divided into two independent wireless segments connected by a wired backbone. In conclusion, we believe that our novel Industrial IoT module architecture will have a significant impact on the future technological development of high-precision time-synchronized communication for the cost-sensitive industrial IoT market.**

***Index Terms*—Field-programmable gate array (FPGA), firmware, hybrid networks, Internet of Things (IoT), Jitter, low-cost sensors and devices, real-time systems, sensor and actuator networks, synchronization, WiFi.**

I. Introduction

DURING the last decade, Internet of Things (IoT) has been one of the key technology drivers, being applied in smart-buildings [1], agriculture [2], energy [3], industrial logistics [4], and other application areas. IoT networks of sensors and actuators have been driving the evolution of the automation field: in this context, each application area defines specific requirements in terms of a transmission medium, range, throughput, communication jitter, etc. For example, while in some applications, like smart agriculture, wireless communications are the fundamental architecture [2], in other applications, like industrial automation or smart-building automation, usually wired and wireless solutions are combined to find the best tradeoff between installation costs and reliability [1]. As another example, high delays can be tolerated in sensor networks for agricultural monitoring. In contrast, in other applications, like in phasor measurements for smart grids [3], [5] or in factory automation, very short latencies and high-accuracy synchronization are required. It is, especially, in robot and autonomous vehicle applications, where both throughput and time sensitivity become fundamental.

For wired connectivity, Ethernet has become the one-size-fits-all solution for all new connectivity solutions in its various flavors. The time-sensitive networking (TSN) extension supports high-accuracy synchronization and vendor-independent real-time communication. For wireless applications, instead, a definitive solution still does not exist. Many requirements advocate for the use of 5G [6] and WiFi [7] technologies [4], [8].

5G networks in their different flavors combine several strengths, i.e., long-range, scalability, significant data transmission rates, and even support for time-sensitive communications [9]. However, one challenge with 5G technology is a relatively complex commissioning and operation process, usually provided by a mobile operator. As the underlying infrastructure is shared among all customers, IoT applications that require complete control of network resources are not simply possible. Even though deploying a private 5G system on a specific industrial facility is feasible, the efforts to operate its own 5G infrastructure are still significant—even if new attractive solutions will be available in the near future. Also, an interesting solution to this challenge was proposed by the 3GPP consortium in the 5G NR specification release 16 with the possibility to operate 5G communications over unlicensed spectrum bands, i.e., the same used by IEEE 802.11 (WiFi) networks [6].

IEEE 802.11 is a well-known technology that has undergone more than 20 years of development. The current 6E release can be deployed over 2 GHz of unlicensed spectrum. It can provide multigigabit throughput thanks to 160 MHz channels and MIMO modulations while being cost efficient and

Manuscript received 14 October 2022; revised 16 December 2022; accepted 4 January 2023. Date of publication 9 January 2023; date of current version 23 May 2023. *(Corresponding author: Alexey M. Romanov.)*

Alexey M. Romanov and Pavel E. Tripolskiy are with the Institute of Artificial Intelligent, MIREA-Russian Technological University, 119454 Moscow, Russia (e-mail: romanov@mirea.ru).

Francesco Gringoli is with the National Inter-University Consortium for Telecommunications/Department of Information Engineering, University of Brescia, 25123 Brescia, Italy.

Kamil Alkhouri was with the Institute of Reliable Embedded Systems and Communication Electronics, Offenburg University of Applied Sciences, 77652 Offenburg, Germany. He is now with KUNBUS GmbH, 73770 Denkendorf, Germany.

Axel Sikora is with the Institute of Reliable Embedded Systems and Communication Electronics, Offenburg University of Applied Sciences, 77652 Offenburg, Germany.

Digital Object Identifier 10.1109/JIOT.2023.3235052

keeping the budget for deploying an IoT network very low [7]. The main disadvantages with respect to 5G are relatively short ranges (up to 100 m) and non-real-time characteristics. Different attempts have been made to solve these two issues during the last decade. Directional antennas can easily increase the 802.11 range up to 1 km [10], albeit compromising omnidirectionality. On the standardization side, IEEE introduced the 802.11ah amendment, dedicated to reliable communication in the sub-GHz band with ranges above 1 km and latencies of $\approx$ 100 ms [11]. Another practical way of enhancing range without losing bandwidth was demonstrated with hybrid networks, where several wireless access points (APs) are connected with a wired backbone [12].

The other "major" issue of 802.11 is establishing low-jitter time synchronization between IoT devices. While there are proprietary extensions, like industrial wireless LAN and some IEEE802.11 multimedia extensions in IEEE802.11e and h—trying to improve the best effort networks with a Class-of-Service approach, the Chinese national committee to IEC has also introduced two additional standards around the wireless infrastructure association (WIA). The WIA FA standard for Factory Automation [13] solves this problem by building a dedicated, time-deterministic data link layer on top of the 802.11 physical layer. Even though WIA FA can be profitably used in some robotic IoT applications, like Automated Guided Vehicle collision avoidance [4], it cannot guarantee jitter below 1 ms, which is required for high-dynamic distributed motion control applications. The new standard WIA NR proposes a New Radio to address this issue, but, based on 5G over unlicensed spectrum, it is in the early stages of its deployment roadmap [14].

Software-defined radio (SDR) technology has been considered a demo platform for showcasing time-sensitive wireless communication between IoT nodes multiple times. Based on field-programmable systems-on-chip (FPSoC) equipped with external radio-frequency (RF) hardware, SDR provides unrivaled flexibility in modifying the lower layers of wireless communication stacks. This was exploited for demonstrating pure wireless and hybrid wired–wireless time synchronization with submicrosecond jitter, both in 802.11-compliant implementations [12] and fully customized communication solutions [15]. Despite its flexibility, the dramatically high costs of SDR prevent its adoption as a universal solution for time-synchronized IoT communications: more than 1000 EUR are required per node, even in high quantities.

This article proposes a novel approach to building cost-efficient IoT modules for precisely synchronized wireless-wired industrial networks. In order to avoid the problems mentioned earlier with SDR designs, we propose a solution based on a commercially available wireless System on a Chip (SoC) extended with a low-cost FPGA, where specific modifications manage both communication and time synchronization to the software running in the involved devices.

We proposed a new low-footprint generalized Precision Time Protocol (gPTP) implementation to provide compatibility with Ethernet TSN. It runs on an SoC application processor along with the FreeRTOS operating system, using the same adjustable Real-Time Clock hardware as a wireless synchronization mechanism. The overall production costs of the module are below 70 EUR in small quantities, which is around 20 times lower than competitive solutions with comparable system performance [16], and more than ten times cheaper than market-available EchoRing Ethernet bridge, which provides point-to-point synchronization in 5-ms range. Our solution, thus, reaches the submicrosecond synchronization precision demonstrated with SDR approaches at the cost of a standard mid-end IoT device. Moreover, we validated the performance of our solution during long-term tests on a real prototype of a hybrid network with different network loads for a total of 120 h, which makes our experimental results on jitter analysis more representative than those reported in many recent papers on this topic.

The key contributions of our research to state of the art are as follows.

1) A novel architecture for a low-cost Industrial IoT (IIoT) module, providing precisely synchronized wired and wireless communication and consuming three to five times less power than FPSoC-based SDR solutions.
2) A novel event synchronization interface for modern wireless SoCs, which is suitable for mass production and provides more than five times better synchronization precision than its known predecessors.
3) A software/firmware stack for synchronization and communication in hybrid networks which has low resource consumption and is, thus, compatible with low-cost wireless SoCs and FPGAs, providing better synchronization rate and precision than similar solutions.
4) Experimental results from a specially designed automated testbed show that the proposed solution can guarantee precisely synchronized communication with jitter standard deviation below 200 ns.

The remainder of this article is organized as follows. Section II contains a brief literature review of known solutions. Section III presents the novel IIoT module architecture and describes the modifications of the wireless SoC firmware that provide the functionalities necessary to implement the new event synchronization interface. It also introduces the FreeRTOS-based implementation of the novel communication stack, including the design of the gPTP module and the extensions for wireless synchronization, in addition to the description of the designed FPGA intellectual property (IP) cores. Experimental results and analysis are reported in Section IV. Finally, Section V summarizes the main research results.

II. State of the Art

As IoT devices play a key role in different areas of modern industry, a significant number of projects contributed to their development [17], [18]. Even though [17] determined three categories of systems (low-, middle-, and high-end), developers usually balance cost and performance by combining different devices: i.e., low-end to collect data and control actuator, and high-end to perform intelligent control [19]. The same approach and hardware solutions are used in service robotics, where hierarchical control systems are already

state of the art [20]. While the majority of IoT devices are developed around a microprocessor, FPGAs are routinely chosen to process data at high sample rates or to implement custom interfaces [21]. FPSoC [22] approaches are also considered, but they are as expensive as high-end IoT modules despite combining FPGA and microprocessor in one integrated circuit.

Currently, Ethernet TSN is one of the most promising contenders for vendor-independent real-time wired communication between IIoT nodes. However, to perform gPTP synchronization [23], devices should be equipped with TSN-compatible Ethernet controllers like the Intel I210. In contrast, [24] demonstrates that TSN can be supported with standard Ethernet controllers coupled with dedicated FPGAs, while an FPSoC-based approach is reported [25]. Implementing the gPTP stack requires high-end IoT devices running Linux in these cases.

Reference [26] demonstrates a gPTP stack on an STM32 microcontroller running FreeRTOS and equipped with an Ethernet interface supporting hardware timestamping. However, it does not discuss the software stack at all. An STM32 solution featuring an FPGA for precise timestamping is described in [27]. Unfortunately, [27] does not reveal many implementation details and reports a synchronization precision several times worse than [26].

A survey [28] by Mahmood et al. describes most of the approaches for clock synchronization over IEEE 802.11 introduced before 2017. Since then, most novel approaches have adopted FPSoC SDRs with external radio front-ends.

In [16], the modification of the 802.11 timing synchronization function (TSF) on the OpenWiFi FPGA platform[1] is presented. Using PTP, it achieves a clock synchronization error below 1.4 μs in 90% of the cases. Val et al. [12] developed an SDR-based solution for synchronization in wired–wireless hybrid networks compatible with TSN. Finally, Seijo et al. [25] reported a hybrid TSN proof of concept composed of FPSoC SDR devices. The main drawback of these solutions is their cost: e.g., [12] uses ADRV9361-Z7035 Evaluation Kits based on 20-layer PCBs with a unit price exceeding 1500 EUR, which is ten times the average cost of high-end commercial IIoT devices [17].

Liang et al. [15] introduced an innovative SDR solution for time-sensitive wireless IoT networks that reports 90th percentile of jitter below 100 ns. To achieve such results, however, the authors use three nodes, where each node includes a USRP X310 high-performance SDR and a powerful workstation equipped with 64 GB of RAM and a 16-core 3.4-GHz AMD 1950X CPU. This configuration is far more powerful and expensive than the majority of IoT high-end devices [17] available on the market, i.e., a single USRP X310 costs more than 6500 EUR.

To be precise, SDR is usually not considered as a platform for final industrial products, due to high cost and high power consumption. Instead, it is often used as a prototyping platform for technologies that are then assumed to be implemented in an application-specific integrated circuit (ASIC). However, the overall process of new ASIC development and production may take years, while the so-called "microchip crisis" started during the COVID-19 pandemic made it even longer and harder to plan [29]. In this case, in the short term, SDR becomes one of a few ways to introduce high-precision wireless synchronization in new products. The other way is described as follows.

In principle, synchronization approaches based on commercial IEEE 802.11 chipsets are possible as they adopt reprogrammable real-time microcontrollers to run time-critical operations: for instance, the 802.11 TSF is already sub-μs accurate [16]. Unfortunately, manufacturers do not release toolchains and documentation for modifying the underlying drivers and firmware. Recently, several open-source projects, such as Nexmon [30] and PicoScenes [31], changed this trend and released toolchains for customizing the entire stack of selected 802.11 chipsets.

In 2021, our team introduced the first synchronization approach based on Nexmon [32]. In that research, we showed how to precisely timestamp IEEE 802.11 beacons on a Raspberry Pi node with the help of an external low-cost FPGA and custom firmware running in the 802.11 wireless transceiver of the RP. While being priced below 30 EUR, our solution provides very promising results, i.e., it achieves a synchronization error with a standard deviation below 500 ns. Still, our solution exhibits a few significant drawbacks: 1) it does not fully meet submicrosecond synchronization precision requirements: maximum jitter, in fact, can exceed 1 μs; 2) the RP's Ethernet interface is unsuitable for real-time communication; and 3) to deploy the event synchronization interface originally proposed in [33] and enhanced in [32]: a) the shielding of the IEEE 802.11 chipset has to be removed and b) additional wires must be soldered to the tiny pad of the wireless receiver input. These operations affect the system's overall reliability and are inappropriate for mass production. Finally, the synchronization algorithm requires transmitting a wireless frame to trigger the FPGA action, which adds unnecessary load to the channel. To overcome these problems, we significantly revised our solution in this article to make it 1) easy to manufacture and 2) even more precise. In the remainder of this article, we present our design methodology and describe the architecture along with the implementation of our novel IIoT module architecture.

III. Novel IIoT Module Architecture

In this section, we introduce a novel IIoT architecture inspired by [32], but overcoming its main limitations. It is based on a commercial industry-ready wireless SoC combined with a low-cost FPGA, features both precise wired and wireless synchronization capabilities, introduces a novel event synchronization interface suitable for mass production, and does not require transmitting additional radio signals. In our design, all the user-controlled interfaces of the SoC are connected to the FPGA, which interacts with sensors and actuators, performs real-time data processing, and includes the hardware for both wired and wireless synchronization. While we focused on two specific products, i.e., Cypress IoT

[1] https://github.com/open-sdr/openwifi

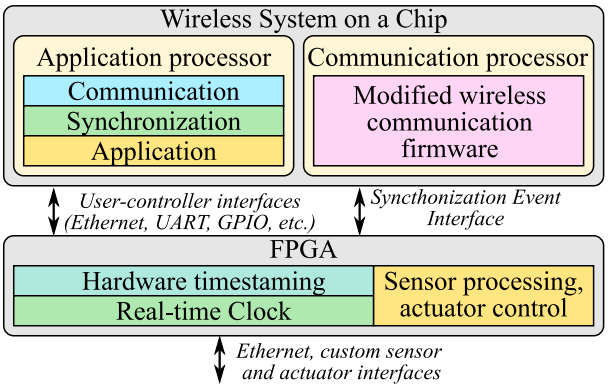

Fig. 1. Proposed IIoT module architecture.

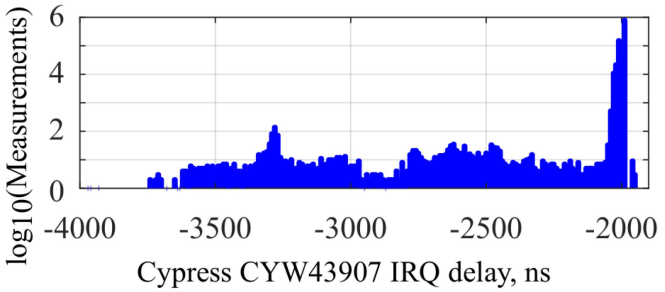

Fig. 2. Logarithmic histogram of CYW43907 IRQ latency.

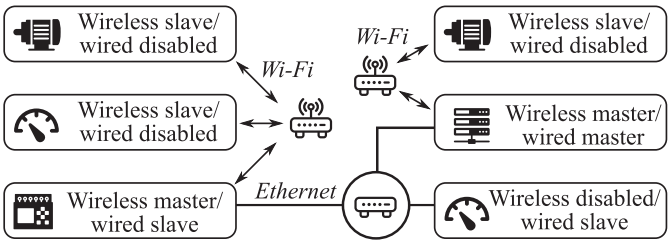

Fig. 3. IIoT module communication through hybrid network concept.

SoCs CYW43907 and CYW54907, that embed a WiFi 4 and a WiFi 5 *communication processor*, respectively, the same results can be obtained with any wireless SoC equipped with separate application and communication processors whose firmware can be customized by direct rebuilding or patching.

The implementation of the proposed architecture (Fig. 1) assumes that the firmware of the *communication processor* can be modified in order to transmit FPGA messages containing the sequence numbers (SNs) of the WiFi beacons received from the AP. To this end, an interface directly controlled by the *communication processor* should be defined. More specifically, it should provide a very low latency between beacon reception and message transmission and a way for discriminating received messages from random interferences. As in [32], we call it synchronization event interface (SEI). We show in the following section how we successfully adopted the serial enhanced coexistence interface (SECI) available in the chosen SoCs as SEI. While the modifications to the *communication processor* described above are the only ones required by the proposed architecture, the possibility to customize its firmware makes the entire platform very flexible. Among many promising options, Pizarro et al. [34] introduced submeter accurate localization of a wireless node without any additional hardware on the IIoT module.

In the proposed architecture, the *application processor* is mainly used to implement the communication and synchronization protocols for hybrid wired and wireless networks. In addition, it can process state machines for application tasks. All real-time data processing and specific sensor and actuator interfaces are implemented on the FPGA. First, within the FPGA, we can easily implement digital interfaces for almost all devices available on the market, providing flexibility close to that of FPSoC-based platforms. Second, the CPU of the *application processor* runs the real-time synchronization stack, where any additional computational load can negatively affect its performance. Instead, the FPGA executes time-critical tasks in parallel, so they are isolated from each other. Third, concurrent tasks running on the *application processor* can significantly deteriorate latency. According to our measurements, the latency of digital input interrupts the processing of CYW43907 ranges from 1.95 to 4.06 μs (Fig. 2). Running the *Real-Time Clock* on the FPGA significantly reduces the synchronization error between devices and applications. Finally, when additional computational resources are needed on the FPGA, replacing the current with a new one is straightforward and does not require code refactoring [35], [36]. Changing the *application processor* of the SoC, instead, is not really possible.

Regarding the synchronization methodology, we follow the hybrid scheme described in [32], which we modified to be compatible with the hardware solutions proposed in this article. First, we equip each IIoT module that can act as a master (the source providing the clock) or slave, depending on its configuration, with both wired and wireless interfaces. The configuration defines the synchronization role of each interface: for instance, when a module is synchronized through the wired network, it cannot act as a wireless synchronization slave (and vice versa) but can redistribute synchronization information toward the wireless domain. In addition, a module can be synchronized with an external clock source distributed through an Ethernet TSN network, or it can act as a clock source (grandmaster) endpoint itself.

In summary, the generalized hybrid network of IIoT modules can be represented as an Ethernet TSN backbone connecting several wireless networks, all synchronized to a single clock source as in Fig. 3. It is also worth mentioning that our solution does not require Ethernet switches or IEEE 802.11 APs to be TSN compliant: they just need to be compatible with the standard. Needless to say, the implementation of additional features (like WiFi-based IIoT module localization [34]) may introduce some constraints.

The two main challenges of the proposed architecture are: 1) a low-latency SEI that must be both reliable and cheap to manufacture and 2) a wired and wireless synchronization stack that matches the limited resources of the *application processor*. In the next sections, we introduce the hardware solutions and the software stack that we developed in order to satisfy these two requirements: we report in the Appendixes additional details for readers interested in the implementation.

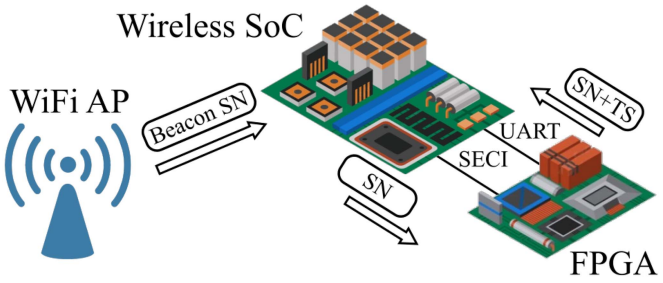

Fig. 4. Illustration of the beacon timestamping procedure performed by each node involved in wireless synchronization.

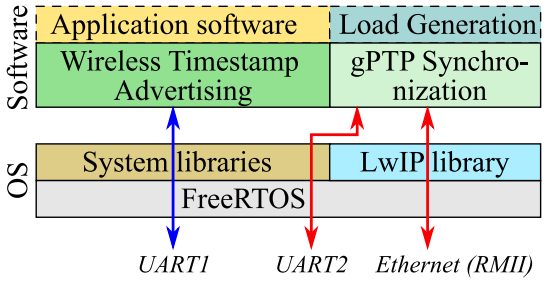

Fig. 5. Software structure.

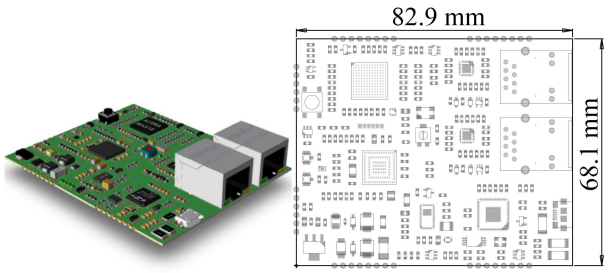

Fig. 6. IIoT module prototype (left) and IIoT module PCB layout (right).

A. Novel SECI-Based Synchronization Event Interface

The SECI is an arbitration mechanism adopted in many combo chipsets to avoid collisions between Bluetooth and WiFi transmissions in the 2.4-GHz band. Despite the scarcely available documentation, some information was recently presented in [37], which showed how SECI is fundamental to maintaining communication performance and demonstrated some underlying vulnerabilities. We modified the behavior of the SECI interface and adapted it for sharing synchronization information between the *communication processor* and the FPGA. As our module design does not include a Bluetooth controller, our modifications do not introduce any side effects.

As our approach uses the beacon transmitted from the WiFi AP as a reference, we activate an SECI transmission as soon as a beacon is received (Fig. 4). Thanks to the very low latency and negligible jitter of the SECI interface (see the Appendix), the timestamp (TS) produced by the FPGA upon detecting the SECI transmission approximates the timestamp of the beacon with very high accuracy. By embedding the SN of the beacon in the SECI message, an integer of 12 bits, we can use the same beacon as a common reference at an arbitrary number of receivers. To match the specifications of the SECI interface, we encode the message with the 6-to-8 encoding from [38], which also gives the FPGA the ability to detect spurious SECI transmissions. We provide details about the internals of such solution in Appendix A.

B. Software Stack for Synchronization and Communication in Hybrid Networks

Our new software stack for synchronization and communication in hybrid networks runs on the *application processor* of the SoC. It gets extra hardware support from a specialized FPGA core that we describe further in Section III-C. The software consists of three main modules, shown in Fig. 5: 1) *Wireless Timestamp Advertising*; 2) *gPTP Synchronization*; and 3) *Load Generation*. These modules run over a *FreeRTOS* operating system in parallel with *Application software* and use the lightweight Internet protocol (*LwIP*) library as a network stack in addition to other *System libraries* from the WICED toolchain.

Since most of the wireless synchronization tasks are performed by the *communication processor* firmware and by the FPGA, the only purpose of the *Wireless Timestamp Advertising* module is to distribute the master clock to other nodes. On a wireless synchronization master node, this module receives the timestamp of the last WiFi beacon from the FPGA and broadcasts it as user datagram protocol (UDP) messages. On the wireless slave side, these UDP messages are received and relayed to the FPGA, which completes the synchronization procedure. When the IIoT module is not synchronized through the wireless network, the *Wireless Timestamp Advertising* module can be stopped.

The *gPTP Synchronization* module implements IEEE 802.1AS (gPTP) as a profile of PTP from IEEE 1588 and provides the functionality required by both gPTP master and slave endpoints for full compliance with IEEE 802.1AS. We provide additional details about this module in Appendix B.

Finally, the *Load Generation* module controls the network load. It is used only during experiments but will not be used in later operations.

C. Hardware Module for Hybrid Networks Implementation

The proposed IIoT module architecture is implemented as a PCB module with a WiFi module Murata Type 1GC/1PS and an Intel MAX10 FPGA (Fig. 6). The choice of the WiFi module is motivated by compatibility with the software described in Sections III-A and III-B, while the Intel MAX10 is one of the cheapest FPGA with integrated memory. The application processor of the Murata wireless module is loaded with firmware based on the wired–wireless synchronization stack described in the previous section. The D11 communication processor firmware is patched to send the 12-bit beacon's SN through the SECI interface every time it receives a beacon from the 802.11 AP. The FPGA design, which provides synchronization hardware support (Fig. 7), consists of three main parts: 1) wired synchronization; 2) wireless synchronization; and 3) real-time clock adjustment.

The *Adjustable Real-Time Clock* serves as the primary clock source for all synchronization-related IP cores and application

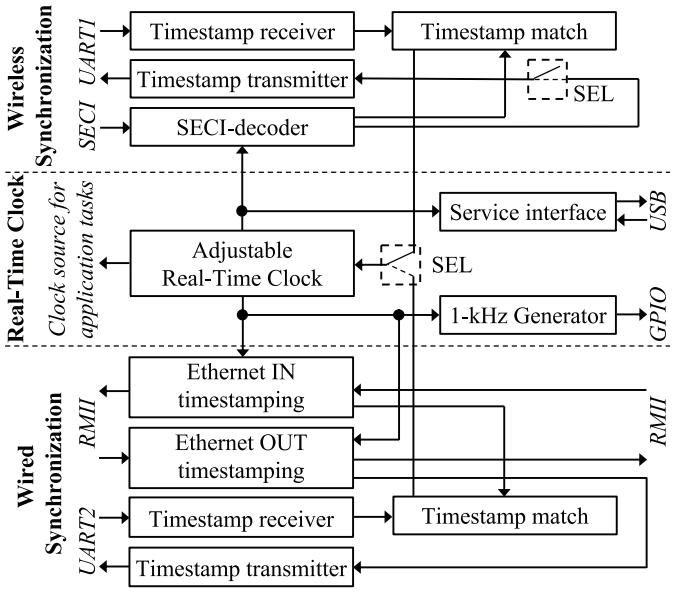

Fig. 7. FPGA firmware structure.

tasks. It is adjusted by one of the two *Timestamp match* modules, which depends on the state of the *SEL* switch configured during the synchronization startup phase. Even though it would be possible to reduce FPGA area utilization by using only one *Timestamp match* module and switch its inputs with selectors similar to *SEL*, this would negatively influence FPGA timing results as timestamps are 80 bits-wide.

The FPGA also includes a 1-kHz Generator and a USB-UART Service interface for debugging and monitoring jitter: these blocks can be removed in real-life applications. For wireless synchronization, the *SECI-decoder* receives and timestamps SECI messages, checks their validity, and decodes the beacon's SNs. Beacon's SNs and their corresponding timestamps are sent to the *Timestamp match* core. The second input of this core is connected to the *Timestamp receiver* that forwards the last beacon timestamps received by the Murata SoC from the wireless synchronization master. *Timestamp match* has two separate, three unit-long queues connected to its inputs. The beacon's SNs stored in these queues are compared when a new pair of beacon's SNs and timestamps arrives at one of the module's inputs. If it finds a match between SNs from the different inputs, the corresponding pair of timestamps will be sent to *SEL* to adjust the Real-Time Clock. If the IIoT module is configured as wireless synchronization master, the *SEL* selector simultaneously switches the clock adjustment source to the Wired Synchronization *Timestamp match* IP core and connects its output of *SECI-decoder* to *Timestamp transmitter*. The latter sends the SN, and the timestamp of the received beacons to the synchronization stack on the Murata SoC application processor that forwards them to the other wireless IIoT modules.

The wired synchronization hardware is similar to the wireless one and includes identical *Timestamp receiver*, *transmitter*, and *match* modules. Instead of the *SECI-decoder*, there are two Ethernet timestamping modules that, besides timestamping Ethernet frames, forward the corresponding traffic to the wireless SoC and vice versa. These modules are connected directly to the Reduce Media Independent Interfaces of the Ethernet PHY and the SoC. The *Ethernet IN timestamping* core timestamps all incoming frames and embeds the 80-bit timestamps inside the frames in front of the Ethernet checksum. The *Ethernet OUT timestamping* core timestamps all outgoing frames and sends them back to the SoC each time it receives a corresponding command via the UART.

While the IIoT module supports up to two independent Ethernet ports, only one was used in this research. A second port can be used for application tasks or to implement a two-port switch for a daisy-chain connection. Both options can be easily implemented using available FPGA resources.

TABLE I
FPGA FIRMWARE RESOURCE UTILIZATION BY ENTITY OF DIFFERENT TYPES

IP core	LUT	RAM	DSP	PLL
Adjustable Real-time Clock	1590	0	0	0
Wired Synchronization	1923	2	0	1
Wireless Synchronization	1991	0	0	0
1kHz Generator	121	0	12	0
Service Interface	337	0	0	0
Total:	**5962**	**2**	**12**	**1**

TABLE II
FPGA FIRMWARE SYNTHESIS RESULTS

Target FPGA	Number of entities			
	LUT	RAM	DSP	PLL
Cyclone IV (EP4CE22F17C6)	27%	<1%	9%	25%
MAX10 (10M08SCU169C8G)	77%	<1%	25%	100%
MAX10 (10M16SCU169C8G)	39%	<1%	13%	100%

IV. EXPERIMENTAL RESULTS

A. Firmware Resource Usage

The PCB of the developed prototype can be equipped with two different wireless SoCs from Murata (Murata 1GC based on Cypress CYW43907 and Murata 1PS based on Cypress CYW54907) and with several FPGAs versions of the Intel MAX10 series. Since the Murata 1PS module was still under development during experimental studies, we chose the 1GC version. The FPGA firmware was synthesized with Intel Quartus Prime Version 17.0.0 Build 595: according to the resource consumption that we report in Table I, two out of five Intel MAX10 pin-compatible FPGAs can be considered (Table II). Finally, the efficiency of our design is proven by the low resource usage for Cyclone IV FPGA on the popular low-cost development kit Terasic DE0-Nano.

Service Interface and *1-kHz Generator* are used only for debugging and monitoring jitter and are not required in real industrial applications. Still, our design keeps fitting into low-cost FPGAs even if these modules are included, and we have enough unused resources for application-related tasks or Specialized Computational Cores [36].

Our SoC software from Section III-B with the *gPTP Synchronization* module requires only 27.6 kB (Table III),[2] which is eight times smaller than *ptp4l* daemon in Linux-based

[2]The Load Generation module is not listed in Table III as it is used only to perform experimental studies.

TABLE III
MEMORY USAGE OF THE SOFTWARE

Software module	Memory usage
Wireless Timestamp Advertising module	712B
gPTP Synchronization module	27599B
LwIP	48388B
System libraries	256553B
FreeRTOS	6655B
Total:	**339907B**

TABLE IV
IIOT MODULE PROTOTYPE MANUFACTURING COSTS

FPGA	Batch-size	Compo-nents, EUR	PCB, EUR	Manufac-turing, EUR	Total, EUR
Module v1	1	75.78	0.72	62.4	138.9
Module v2	1	91.38	0.72	62.4	154.5
Module v1	1000	59.86	0.45	6.46	66.77
Module v2	1000	73.74	0.45	6.46	80.65

implementations. Additionally, we used LwIP instead of standard sockets implementation, which decreased memory usage even more. Consequently, we managed to fit it into the built-in memory of the wireless SoC, which is only 640 kB. The total size of our software is 499 kB, including a compressed image of the *communication processor* firmware, which leaves more than 140 kB for application-specific functions.

B. Hardware Costs

For our experiments, we implemented modules with a 10M08SCU169C8G FPGA (Module v1). The total cost, including components and manufacturing, is provided in Table IV. For comparison reasons, we also show the cost of an alternative version (Module v2) based on a 10M16SCU169C8G FPGA as well as the cost of these modules if produced in batches of 1000 pieces, which can be considered a small quantity for such type of electronics. Numbers do not include 190 EUR nonrecurring engineering cost for manufacturing preparations and materials.

Even in single quantities, these costs are around ten times lower than the FPSoC-based hardware in [12]. Module v2, equipped with the larger FPGA, has a cost comparable to the average price of a high-end IoT device [17]. At the same time, with mass production, this cost can be reduced (to below 70 EUR), which is comparable with standard mid-end devices, where no FPSoC device with IEEE 802.11 compatible interface is available. Thanks to the proposed module architecture, an 8-layer PCB is entirely sufficient and reduces the cost of the PCB below 1 EUR. In addition, all components (except RJ-45 connectors) are suitable for fully automatic pick-and-place.

It is worth mentioning that manufacturing costs (Table IV) are only part of the final product cost [39]. The other parts are dedicated to expenses on development, marketing, customer support, etc. These costs differ from product to product. In the industrial automation market, their total sum usually ranges from 60% to 150% of mass-production manufacturing costs. Even though pricing and marketing are out of the scope of this research, using the above-mentioned ratios, we can estimate that the total cost of the proposed solution will be below 167 EUR for Module v1 and below 202 EUR for Module v2. Thus, it is still several times lower than market-available solutions, like R3 EchoRing Ethernet Bridge.

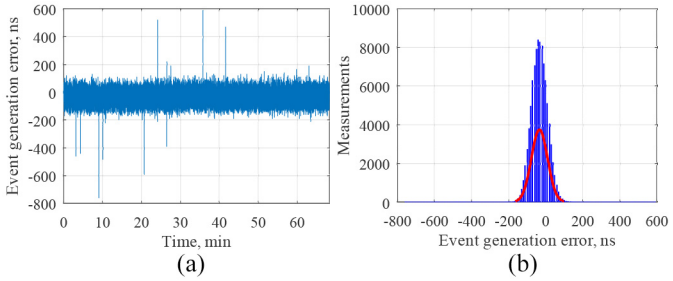

Fig. 8. SECI event generation error: (a) transient process and (b) histogram.

We believe these cost considerations, which are often disregarded in the literature, are fundamental indeed. High equipment cost, in fact, is one of the key barriers to the wide deployment of wireless time synchronization. As it was mentioned above, SDR FPSoC hardware prices exceed 1500 EUR. Similarly, one of the few industrial low-latency wireless products available on the market, the R3 EchoRing Ethernet Bridge, costs above 750 EUR and still, as reported in the datasheet, its synchronization precision lies in 5-ms range [40]. In contrast, our solution provides a significant drop in hardware costs and performance, and we show next that its performance is close to real-time wired solutions used in the industry nowadays.

C. Synchronization Precision

As synchronization is one of the key features of the proposed IIoT module architecture, we performed a series of experiments to properly estimate its precision and compare it with state of the art.

First, we estimated the jitter of our SECI-based SEI. During this experiment, we put two IIoT modules close to each other, keeping the same distance between their antennas and the antenna of the AP. The firmware of the *communication Processor* was configured to send a short SECI message each time it receives a WiFi beacon. After receiving this message, the FPGA generates an edge. An FPGA-based logic analyzer from [41] measures the SECI jitter, i.e., the time differences between the edges from both FPGAs (Fig. 8). Measurements ran for approximately 68 min. The results show a maximum SECI jitter below 730 ns with a standard deviation below 50 ns and 90-percentile (P_{90}) of 70 ns outperforming [32], [33].

In the second experiment, we set up one module as a wireless synchronization master and the other as a slave. We implemented another slave device as in [32] with a Raspberry Pi 3B and a Lattice iCE40 FPGA. The only significant modification was the use of timestamps advertised by our master device as a synchronization source instead of IEEE 802.11 TSF. The jitter was monitored for 10 h by the same logic analyzer as in the previous experiments. Fig. 9 shows that the SEI proposed in this article has much less jitter compared to the state of the art. In our prototype, the jitter

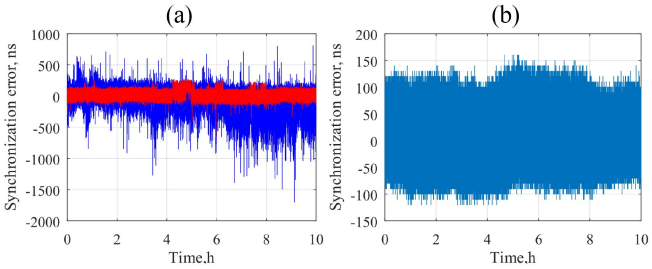

Fig. 9. (a) Wireless synchronization precision is provided by the method proposed in this article (red) and [32] (blue). (b) Wired synchronization precision is provided by the method proposed in this article.

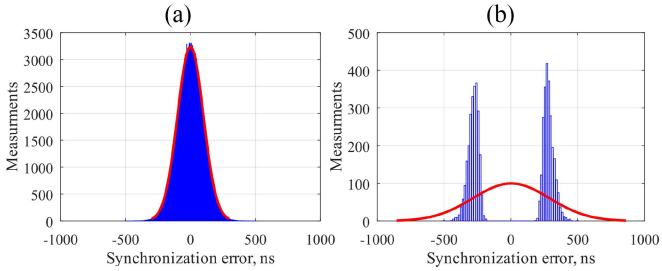

Fig. 10. Histograms of (a) normally distributed synchronization error and (b) normally distributed synchronization error processed by 128× prescaler.

TABLE V
SYNCHRONIZATION METHODS' ERRORS UNDER NO LOAD IN IEEE802.11

Method	Sdv., ns	P_{90}, ns	P_{99}, ns
A. Mahmood et al., 2014, [42]	360	620	–
G. Cena et al., 2015, [43]	107	–	464
A. Mahmood et al., 2016, [44]	341	6440	–
A. Romanov et al., 2021, [32]	189.1	280	560
M. Aslam et al., 2021, [16]	820	1400	–
I. Val et al., 2021, [12]	1.9	–	–
This research	**72.67**	**110**	**170**

P_{90} is 110 ns with a maximum of 690 ns and a standard deviation (Sdv.) of 72.67 ns, while in [32], these values are 280, 1620, and 189.1 ns, respectively.

A fair comparison of the different Wi-Fi synchronization methods is very complicated, as all conditions of the experiments should be similar. In fact, the test results available in the literature have been executed under different electromagnetic conditions, with different numbers of nodes, and with various network loads. At the same time, in most of the papers, at least in one experiment, the results were collected with point-to-point synchronization of two wireless nodes in a regular office environment without additional interference or network load. This article is no exception (see the second experiment). Table V compares our solution with the other methods for this test case, as it is a reasonable and realistic use case. As can be seen, our solution outperforms most of the others, excluding [12]. At the same time, it can be implemented with market-available components at a much lower cost than the ones required by [12]. Meanwhile, being fair and wide (in terms of compared methods), this comparison only covers some actual use cases, excluding synchronization of multiple nodes in hybrid networks or operation in crowded radio areas. Thus, we performed additional experiments to deeply analyze the performance of our solution. These experiments are described below.

In the third experiment, we verified our method using wired synchronization. We used one of our prototypes as a grandmaster and the other as a slave endpoint. Their 1 kHz signals were connected to the logic analyzer. The output of the grandmaster endpoint was used as a reference. As in previous experiments, the logic analyzer was configured with a 128× prescaler. Experimental data was collected for 10 h. Our software/firmware stack guarantees synchronization errors below 160 ns [Fig. 9(b)]. Moreover, the synchronization jitter was below 90 ns with a standard deviation of only 60.5 ns in 90% of the cases. Thus, our results are comparable to the known FreeRTOS-based PTP implementation [26] with an integrated IEEE 1588-compliant Ethernet controller and much better than [27] using FPGA-based Ethernet frame timestamping.

The logic analyzer was configured to send the maximum jitter value that it measured every 64 ms (128 consecutive measurements) to reduce the amount of data recorded during long-term measurements in the last two experiments [41]. Histograms of the normally distributed synchronization error are shown in Fig. 10(a) and the same error after a prescaling in the logic analyzer in Fig. 10(b). Values around zero rarely become worst case measurements during the prescaling interval, which causes a gap in the middle of the second histogram. These values do not impact the maximum jitter but make jitter standard deviation and 90-percentile bigger than their actual value (more than two times compared to the non-prescaled case). Thus, our wired and wireless synchronization approach's standard deviation and P_{90} values are even smaller than the shown values.

Finally, we evaluated the performance in hybrid networks under specific traffic load conditions using our automated physical TestBed (APTB) [45] and keeping out any external EM interference. The antenna ports of nine devices in shield boxes were linked with waveguides interconnected via RF splitters to measure the synchronization accuracy between devices in the different segments. The structure in Fig. 11 includes two separate wireless networks, with six devices in one network and three in the other. The WiFi network uses a TL-WR1043ND AP. One device in each network is configured as a wireless master, with the others being slaves. The wireless masters are synchronized through a wired network, using a Linux-based third-party TSN switch, and using the DE1-SoC development kit as their grandmaster. All IIoT modules are connected to a logic analyzer with their 1 kHz ports. As the third-party TSN switch does not provide a synchronized 1-kHz output, a wireless master (Module 1) is used as a reference. As in the previous experiments, the logic analyzer prescaler is configured to send the worst case jitter values of subsequent 128 measurements. Measurements with network loads from 0 to 10 Mb/s were performed for 24 h for each load. The experimental results are presented in Table VI. For the synchronization error values, the P_{90} metrics are compared with [16] results supplementing it with 99-percentile (P_{99}). In each row of Table VI, worst case values are marked with the semi-bold font.

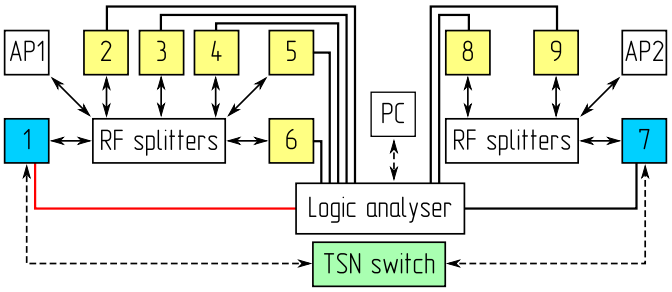

Fig. 11. APTB testbed structure for the third experiment. Solid arrows—WiFi connections through waveguides. Dashed arrows—wired Ethernet connections. Solid lines—1-kHz digital signals connected to the logic analyzer. Red line—logic analyzer's reference 1-kHz signal. Device color scheme: green—wired synchronization master; blue—wired synchronization slave/wireless synchronization master; yellow—wireless synchronization slave; and white—not synchronized.

TABLE VI
SYNCHRONIZATION ERROR IN A HYBRID NETWORK

Load, Mbps	Metric	Module ID 2	3	4	5	6	7	8	9
0	P_{90}, ns	90	120	**360**	160	90	90	110	120
	P_{99}, ns	120	200	**410**	190	120	110	150	150
3	P_{90}, ns	110	110	110	100	**120**	100	110	110
	P_{99}, ns	230	190	250	210	**260**	120	190	160
5	P_{90}, ns	130	130	**140**	130	**140**	50	110	120
	P_{99}, ns	190	180	190	220	**240**	70	150	160
8	P_{90}, ns	150	140	140	**170**	140	90	150	150
	P_{99}, ns	410	400	390	**450**	340	120	320	310
10	P_{90}, ns	190	210	240	250	**290**	100	110	110
	P_{99}, ns	520	480	620	610	**700**	140	170	170

During 99% of the time, the error does not exceed 700 ns (cf. Table VI). Moreover, for 90% of the time, it is below 360 ns, which is approximately four times better than [16]. With increased network load, both P_{90} and P_{99} are rising because of the higher loss of UDP broadcasts with master clock information. Still, even in these cases, the values of P_{90} and P_{99} continue to be below 1 μs. The synchronization results achieved in [12] are generally better than ours but were obtained using 20 times more expensive equipment. In addition, these works do not investigate any dependency on the network load. Experiments in [12], [16], [25], and [32] were performed with only two wireless nodes. Moreover, in [12] and [16], the duration of the experiments was around 20 min, and the jitter was estimated once per second by analyzing the pulse per second signal at the output of each device. In our work, instead, we performed long-term tests by collecting jitter 2000 times per second with different network loads for a total of 120 h. And most importantly, we evaluated a more complex hybrid network consisting of nine modules with independent wireless segments and a third-party TSN switch as the master synchronization clock source. This indeed makes our experimental results on jitter analysis more representative than those reported by recent papers.

We performed another experiment to ensure that experimental results achieved on APTB are also representative of real conditions of the noisy radio spectrum. We set up our WiFi AP in a crowded part of the building where several neighboring public WiFi networks were present. We placed two nodes, one master and one slave, as close as possible to the AP; we placed a second slave at approximately 10 m from the AP in a position where the signal strength of the neighboring networks was very high. Fig. 12 presents synchronization errors of both slave devices measured from 10:00 to 19:00 on a business day. As it can be seen, the error of the distant slave device is biased in the interval between 11:00 and 16:00: this can be explained by considering typical working patterns with people being more active at that moment and, hence, generating more traffic at the neighboring WiFi networks. During this time interval, the synchronization error also exhibits several peaks with amplitude above 1μs that are not present, instead, on the master and slave nodes placed near the AP. Thus, these deviations can be considered caused by interference from public WiFi networks. Numerical analysis of the results shows that the slave placed near AP provided synchronization error Sdv. of 86.1 ns, $P_{90} = 130$ ns, and $P_{99} = 190$ ns. The distant slave, instead, provided Sdv. of 134.4 ns, $P_{90} = 190$ ns, and $P_{99} = 260$ ns. Those values lay in the same range previously measured in laboratory conditions (Table VI). Thus, the experimental studies performed on APTB represent real applications in a noisy environment.

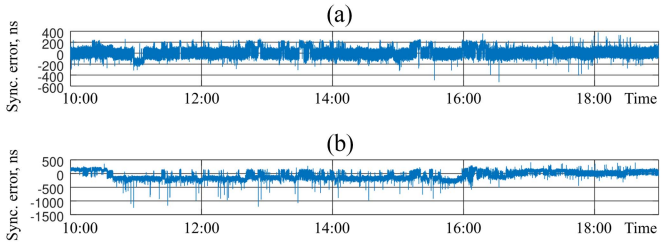

Fig. 12. Synchronization error of slave nodes measured in a crowded area with several public WiFi networks: (a) slave device located near AP and (b) slave device distanced from AP by approximately 10 m.

As can be seen, regular interference from other WiFi networks does not significantly impact interdevice synchronization. At the same time, our approach does not introduce additional features for increasing WiFi reliability, and, in fact, cannot provide good performance if the radio signal is jammed. To prevent this, well-known approaches to achieve Electromagnetic Compatibility should be used, including the proper isolation of communication part from power electronics as well as IoT device shielding, placement, and antenna positioning (the proposed IoT module can be equipped with two external antennas) [46]. Meanwhile, radio conditions in typical industrial applications can be much better than those in the described-above experiment, thanks to proper spectrum planning and frequency allocation, which can minimize radio interference.

D. Power Consumption

We evaluated the power consumption of our prototype during standby and communication, with and without network load, and we compared it with R3 EchoRing Ethernet Bridge, and OpenWiFi SDR-solution implemented on Xilinx ZC706

TABLE VII
PROPOSED IIoT MODULE POWER CONSUMPTION COMPARED TO THE SDR SOLUTION

Module	Mode	Average Voltage	Average Current	Average Power
SDR-based (OpenWiFi)	Standby	12.15 V	0.71 A	8.63 W
SDR-based (OpenWiFi)	Communication (5 Mbps network load)	12.11 V	0.82 A	9.93 W
EchoRing Ethernet Bridge	Typical (according manual)	24 V	0.042 A	1 W
EchoRing Ethernet Bridge	Maximum (according manual)	24 V	0.146 A	3.5 W
Proposed IIoT module (Wireless master)	Standby	5.5 V	0.442 A	2.431 W
Proposed IIoT module (Wireless slave)	Standby	5.5 V	0.298 A	1.639 W
Proposed IIoT module (Wireless master)	Wired synchronized, Wireless standby	5.5 V	0.442 A	2.772 W
Proposed IIoT module (Wireless master)	Wired, wireless synchronized	5.5 V	0.575 A	3.163 W
Proposed IIoT module (Wireless slave)	Wireless synchronized	5.5 V	0.373 A	2.052 W
Proposed IIoT module (Wireless master)	Wired, wireless synchronized, Communication (5 Mbps network load)	5.5 V	0.589 A	3.24 W
Proposed IIoT module (Wireless slave)	Wireless synchronized, Communication (5 Mbps network load)	5.5 V	0.419 A	2.305 W

development board and Analog devices FMCOMMS2-EBZ high-speed analog module. For our prototypes and OpenWiFi, power consumption was estimated by measuring current and voltage with meters connected to their power supply chains. For EchoRing, instead, we used information from the vendor's datasheets as we did not have access to the actual product. As it can be seen in Table VII, our IIoT module consumes up to five times less power in standby and up to three times less power during active synchronized communication compared to the FPSoC-based SDR solution. With respect to the EchoRing Ethernet Bridge product, our solution's power consumption is very similar, but its synchronization precision achieves 700 ns, compared to 5 ms.

V. SUMMARY AND CONCLUSION

In this article, we have designed, implemented, and validated a novel architecture for synchronizing hybrid WiFi-Ethernet networks that combine the requirements of an ultralow-cost approach with low power consumption and significant performance expectations. The proposed architecture includes market-available wireless SoC with modified low-level communication firmware connected with a low-cost FPGA. It provides a complete solution for industrial communication in hybrid wired–wireless networks with synchronization precision below 1 μs (worst case 99-percentile of synchronization error does not exceed 700 ns) at the cost of standard mid-end IoT device, reaching total cost below 167 EUR. Furthermore, it is characterized by up to 20 times lower prices and up to five times lower power consumption than competitive SDR-based solutions. To back up our claims, we reported data that we collected with a high sampling rate (2000 samples per second) during an extremely long (more than 120 h) measurement campaign performed on a testbed composed of a hybrid network with nine nodes divided into two independent wireless segments connected by a wired backbone. Additionally, we verified that experimental results achieved on the testbed match those measured in the area with several public WiFi networks, acting as a source of the RF interference.

In conclusion, we believe that our novel Industrial IoT module architecture will have a significant impact on the future technological development of high-precision time-synchronized communication for the industrial IoT market.

APPENDIX A
SECI INTERFACE

While the proposed modifications to the SECI interface could be implemented on all devices that use it, we restrict the following analysis to the Cypress chipset used in our prototypes.

Like all the other time-critical operations, the Cypress chipset manages the IO of the SECI interface in the D11 core. This microcontroller executes tiny real-time firmware and controls operations like arbitration of channel access and scheduling of transmission [47]. To add functionalities to the D11 core, we modify its firmware, called *ucode*, by using the same binary-patching approach introduced by the Nexmon framework.[3] We then replace the modified ucode in the software image of the SoC that we write into its flash memory: the new code is loaded into the microcontroller memory at the next reboot.

In our implementation, we exploit the real-time capabilities of the D11 core to transmit an SECI message to our FPGA almost exactly at the end of a received beacon. During a reception, the microcontroller knows how many bytes of the current frame has been received and decoded. It can, hence, filter beacons coming from the AP from other frames. When it detects one, it waits until it is completely received by spinning on the hardware signal named `COND_RX_COMPLETE`: when this happens, it starts the transmission of the SECI message by writing register `BTCX_Transmit_Control`. As we implement spinning by using a single compare-and-jump instruction, and since the D11 core does not support interrupts, the delay and, hence, the jitter introduced between the end of the beacon's reception and the start of the SECI message depends on the D11 clock frequency. In our case, the 200-MHz clock frequency translates to 5 ns of maximum jitter. Unfortunately,

[3]The Nexmon project offers tools and resources for binary patching, several drivers, by Broadcom/Cypress.

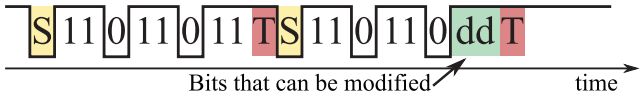

Fig. 13. SECI-frame structure. S-start bit, T-stop bit, and dd are used to transmit part of beacon's SN.

much more significant uncertainty is introduced by the sampling process of the OFDM receiver: the actual jitter is, hence, dominated by this process when operating on a 20-MHz channel, the receiving hardware samples the incoming signal at 20 MS/s, which corresponds to an uncertainty on the frame detection process of 50 ns.

The transfer of the beacon's SN is a bit more complex. The first reason is that the signal transmitted over the SECI is designed specifically for sharing channel access information from a WiFi chipset to a Bluetooth controller (cf. Fig. 13). An SECI frame is a fixed bit sequence where only the two bits marked with `d` can be chosen by the D11 core via two specific bits of register `BTCX_Transmit_Control`: multiple SECI frames must be transmitted for transferring the entire beacon's SN. The second reason is that the SECI frame is transmitted only if at least one of the two bits is modified. Otherwise, it would not be transmitted. Hence, we adopt the encoding from [38], choosing 6-to-8 encoding, where the 12-bit SN is encoded into 16 bits with always different consecutive pairs. This also allows the rejection of SNs affected by spurious SECI transmissions. Finally, we split the encoded bit sequence into pairs and transmit each separately. The first pair is embedded into the first SECI frame, transmitted right after the reception of the beacon, which is used by the FPGA to sample the timestamp of the beacon. All other pairs are transmitted asynchronously with other frames.

APPENDIX B
GPTP SYNCHRONIZATION MODULE

The source code of the *gPTP Synchronization* module is based on LinuxPTP implementation and PTP demon.[4] With respect to that source code, we introduced in our work some significant contributions.

1) We ported STM32 PTP demon to the wireless SoC and upgraded it from PTP to gPTP.
2) We implemented additional features such as gPTP time-length-value (TLV) structures to be exchanged between ports to communicate information related to Follow_Up messages (FOLLOW_UP_INFO_TLV) and to track the path, over which Announce messages are transmitted (PATH_TRACE_TLV).
3) We implemented driver support for the FPGA-based hardware timestamping of gPTP messages over the Ethernet interface.

Timestamps are appended at the end of the frame to keep track of incoming frames. On the other side, the UART interface reads the timestamps of outgoing frames upon request. The messages timestamped at Tx and Rx are Sync, pDelay_Req, and pDelay_Resp. Compared to other known PTP and gPTP solutions, all clock adjustments, including calculating phase and frequency offset, are done by FPGA based on the timestamp of the last Sync message and its original timestamp at the master's clock side estimated by our software. Thus, the *gPTP Synchronization* module performs everything around the wired gPTP synchronization, while the FPGA only performs timestamping and adjustment. The communication via the standard UART interface between the FPGA firmware and the developed software is based on FreeRTOS drivers.

[4]https://github.com/mpthompson/stm32_f4_ptpd

REFERENCES

[1] R. Khan et al., "A hybrid approach for seamless and interoperable communication in the Internet of Things," *IEEE Netw.*, vol. 35, no. 6, pp. 202–208, Nov./Dec. 2021.
[2] T. Ojha, S. Misra, and N. S. Raghuwanshi, "Internet of Things for agricultural applications: The state of the art," *IEEE Internet Things J.*, vol. 8, no. 14, pp. 10973–10997, Jul. 2021.
[3] G. Bedi, G. K. Venayagamoorthy, R. Singh, R. R. Brooks, and K.-C. Wang, "Review of Internet of Things (IoT) in electric power and energy systems," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 847–870, Apr. 2018.
[4] H. Shi, M. Zheng, W. Liang, J. Zhang, K. Wang, and S. Liu, "Transmission scheduling with order constraints in WIA-FA-based AGV systems," *IEEE Internet Things J.*, vol. 8, no. 1, pp. 381–392, Jan. 2021.
[5] M. Agustoni, P. Castello, and G. Frigo, "Phasor measurement unit with digital inputs: Synchronization and interoperability issues," *IEEE Trans. Instrum. Meas.*, vol. 71, pp. 1–10, 2022. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9775053
[6] S. Parkvall et al., "5G NR release 16: Start of the 5G evolution," *IEEE Commun. Stand. Mag.*, vol. 4, no. 4, pp. 56–63, Dec. 2020.
[7] *IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications—Redline*, IEEE Standard 802.11-2020 (Revision IEEE Std 802.11-2016), 2021.
[8] B. Li, Z. Fei, and Y. Zhang, "UAV communications for 5G and beyond: Recent advances and future trends," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2241–2263, Apr. 2019.
[9] B. Bertenyi, "5G evolution: What's next?" *IEEE Wireless Commun.*, vol. 28, no. 1, pp. 4–8, Feb. 2021.
[10] Y. Gu, M. Zhou, S. Fu, and Y. Wan, "Airborne WiFi networks through directional antennae: An experimental study," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2015, pp. 1314–1319.
[11] A. Seferagic, I. Moerman, E. De Poorter, and J. Hoebeke, "Evaluating the suitability of IEEE 802.11ah for low-latency time-critical control loops," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7839–7848, Oct. 2019.
[12] I. Val, Ó. Seijo, R. Torrego, and A. Astarloa, "IEEE 802.1AS clock synchronization performance evaluation of an integrated wired–wireless TSN architecture," *IEEE Trans. Ind. Informat.*, vol. 18, no. 5, pp. 2986–2999, May 2022.
[13] W. Liang et al., "WIA-FA and its applications to digital factory: A wireless network solution for factory automation," *Proc. IEEE*, vol. 107, no. 6, pp. 1053–1073, Jun. 2019.
[14] H. Yu, P. Zeng, and C. Xu, "Industrial wireless control networks: From WIA to the future," *Engineering*, vol. 8, no. 1, pp. 18–24, 2022.
[15] J. Liang, H. Chen, and S. C. Liew, "Design and implementation of time-sensitive wireless IoT networks on software-defined radio," *IEEE Internet Things J.*, vol. 9, no. 3, pp. 2361–2374, Feb. 2022.
[16] M. Aslam et al., "Hardware efficient clock synchronization across Wi-Fi and Ethernet based network using PTP," *IEEE Trans. Ind. Informat.*, vol. 18, no. 6, pp. 3808–3819, Jun. 2022.
[17] M. O. Ojo, S. Giordano, G. Procissi, and I. N. Seitanidis, "A review of low-end, middle-end, and high-end IoT devices," *IEEE Access*, vol. 6, pp. 70528–70554, 2018.
[18] A. Zrelli, "Hardware, software platforms, operating systems and routing protocols for Internet of Things applications," *Wireless Pers. Commun.*, vol. 122, pp. 3889–3912, Sep. 2021.
[19] A. Zielonka, A. Sikora, M. Wozniak, W. Wei, Q. Ke, and Z. Bai, "Intelligent Internet of Things system for smart home optimal convection," *IEEE Trans. Ind. Informat.*, vol. 17, no. 6, pp. 4308–4317, Jun. 2021.

[20] A. M. Romanov, "A review on control systems hardware and software for robots of various scale and purpose. Part 2. Service robotics," *Russ. Technol. J.*, vol. 7, no. 6, pp. 68–86, 2019.

[21] L. Zhao, I. B. M. Matsuo, Y. Zhou, and W.-J. Lee, "Design of an industrial IoT-based monitoring system for power substations," *IEEE Trans. Ind. Appl.*, vol. 55, no. 6, pp. 5666–5674, Nov./Dec. 2019.

[22] R. F. Molanes, L. Costas, J. J. Rodríguez-Andina, and J. Fariña, "Comparative analysis of processor-FPGA communication performance in low-cost FPSoCs," *IEEE Trans. Ind. Informat.*, vol. 17, no. 6, pp. 3826–3835, Jun. 2021.

[23] D. Bruckner et al., "An introduction to OPC UA TSN for industrial communication systems," *Proc. IEEE*, vol. 107, no. 6, pp. 1121–1131, Jun. 2019.

[24] M. Ulbricht, J. Acevedo, S. Krdoyan, and F. H. P. Fitzek, "Precise fruits: Hardware supported time-synchronisation on the RaspberryPI," in *Proc. Int. Conf. Smart Appl., Commun. Netw. (SmartNets)*, 2021, pp. 1–6.

[25] O. Seijo, X. Iturbe, and I. Val, "Tackling the challenges of the integration of wired and wireless TSN with a technology proof-of-concept," *IEEE Trans. Ind. Informat.*, vol. 18, no. 10, pp. 7361–7372, Oct. 2022.

[26] M. Xuan, F. Yu, and Q.-C. Fan, "Research of distributed acquisition system based on clock synchronization," in *Proc. Int. Conf. Sens. Instrum. (ICSI)*, 2021, Art. no. 1188720.

[27] K. Yuan, X. Guo, and J. Tian, "Research and implementation of clock synchronization technology based on PTP," in *Proc. J. Phys. Conf. Series*, 2021, Art. no. 12139.

[28] A. Mahmood, R. Exel, H. Trsek, and T. Sauter, "Clock synchronization over IEEE 802.11—A survey of methodologies and protocols," *IEEE Trans. Ind. Informat.*, vol. 13, no. 2, pp. 907–922, Apr. 2017.

[29] J. Kamasa, *Microchips: Small and Demanded: CSS Analyses in Security Policy*, Center Security Stud. (CSS), ETH Zürich, Zürich, Switzerland, 2021.

[30] M. Schulz, D. Wegemer, and M. Hollick, "The Nexmon firmware analysis and modification framework: Empowering researchers to enhance Wi-Fi devices," *Comput. Commun.*, vol. 129, pp. 269–285, Sep. 2018.

[31] Z. Jiang et al., "Eliminating the barriers: Demystifying Wi-Fi baseband design and introducing the picoscenes Wi-Fi sensing platform," *IEEE Internet Things J.*, vol. 9, no. 6, pp. 4476–4496, Mar. 2022.

[32] A. M. Romanov, F. Gringoli, and A. Sikora, "A precise synchronization method for future wireless TSN networks," *IEEE Trans. Ind. Informat.*, vol. 17, no. 5, pp. 3682–3692, May 2021.

[33] A. M. Romanov, F. Gringoli, and A. Sikora, "High precision synchronization between commercial WiFi-ICs and external device," in *Proc. 16th Workshop Position. Navig. Commun. (WPNC)*, 2019, pp. 1–6.

[34] A. B. Pizarro, J. P. Beltrán, M. Cominelli, F. Gringoli, and J. Widmer, "Accurate ubiquitous localization with off-the-shelf IEEE 802.11ac devices," in *Proc. 19th Annu. Int. Conf. Mobile Syst. Appl. Services*, 2021, pp. 241–254.

[35] A. M. Romanov, M. P. Romanov, and E. I. Shestakov, "A novel architecture for control systems of modular reconfigurable robots," in *Proc. IEEE 2nd Int. Conf. Control Tech. Syst. (CTS)*, 2017, pp. 131–134.

[36] A. M. Romanov et al., "Modular reconfigurable robot distributed computing system for tracking multiple objects," *IEEE Syst. J.*, vol. 15, no. 1, pp. 802–813, Mar. 2021.

[37] J. Ruge, J. Classen, F. Gringoli, and M. Hollick, "Frankenstein: Advanced wireless fuzzing to exploit new Bluetooth escalation targets," in *Proc. 29th USENIX Security Symp.*, 2020, pp. 19–36.

[38] A. M. Romanov and F. Gringoli, "Enhanced self-synchronized reduced media-independent interface for robotic and automotive applications," *IEEE Trans. Ind. Informat.*, vol. 18, no. 4, pp. 2274–2286, Apr. 2022.

[39] R. DiResta, B. Forrest, and R. Vinyard, *The Hardware Startup: Building Your Product, Business, and Brand*. Beijing, China: O'Reilly Media, 2015.

[40] C. Dombrowski and J. Gross, "EchoRing: A low-latency, reliable token-passing MAC protocol for wireless industrial networks," in *Proc. 21st Eur. Wireless Conf.*, 2015, pp. 1–8.

[41] A. M. Romanov, "An easy to implement logic analyzer for long-term precise measurements," *HardwareX*, vol. 9, Apr. 2021, Art. no. e00164.

[42] A. Mahmood, R. Exel, and T. Sauter, "Delay and jitter characterization for software-based clock synchronization over WLAN using PTP," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1198–1206, May 2014.

[43] G. Cena, S. Scanzio, A. Valenzano, and C. Zunino, "Implementation and evaluation of the reference broadcast infrastructure synchronization protocol," *IEEE Trans. Ind. Informat.*, vol. 11, no. 3, pp. 801–811, Jun. 2015.

[44] A. Mahmood, R. Exel, and T. Sauter, "Performance of IEEE 802.11's timing advertisement against SyncTSF for wireless clock synchronization," *IEEE Trans. Ind. Informat.*, vol. 13, no. 1, pp. 370–379, Feb. 2017.

[45] A. Sikora, E. J. Sebastian, A. Yushev, E. Schmitt, and M. Schappacher, "Automated physical testbeds for emulation of wireless networks," in *Proc. ICMIE*, vol. 75, 2016, pp. 1–5.

[46] C. Ziegler, C. Paulwitz, S. Weber, and H. Bachmaier, "Challenges of industry 4.0 for the assessment of electromagnetic compatibility (EMC)," in *Proc. PCIM Europe Digit. Days Int. Exhibition Conf. Power Electron. Intell. Motion Renew. Energy Energy Manage.*, 2021, pp. 1–8.

[47] M. Schulz, J. Link, F. Gringoli, and M. Hollick, "Shadow Wi-Fi: Teaching smartphones to transmit raw signals and to extract channel state information to implement practical covert channels over Wi-Fi," in *Proc. 16th ACM Int. Conf. Mobile Syst. Appl. Services (MobiSys)*, 2018, pp. 256–268.

Alexey M. Romanov (Senior Member, IEEE) received the graduation degree (Hons.) in mechatronic engineering, the Ph.D. degree in electrical and electronics engineering, and the Habilitation degree in electrical and electronic engineering from MIREA-Russian Technological University, Moscow, Russia, in 2010, 2014, and 2021, respectively.

Since 2022, he has been a Full Professor with MIREA-Russian Technological University. During his career, he participated in a wide range of scientific and industrial projects and coauthored more than 80 papers and patents. His current research interests include image processing, signal processing, real-time sensor networks, robotics, and FPGA design.

Francesco Gringoli (Senior Member, IEEE) received the Laurea degree in telecommunications engineering from the University of Padua, Padua, Italy, in 1998, and the Ph.D. degree in information engineering from the University of Brescia, Brescia, Italy, in 2002.

He is a Full Professor with the University of Brescia. His research interests include security assessment, performance evaluation, and medium access control in wireless LANs.

Kamil Alkhouri received the M.Sc. degree in communication and media engineering from Offenburg University of Applied Sciences, Offenburg, Germany, in 2018.

From 2018 to 2022, he was an Academic Engineer with Offenburg University of Applied Sciences, participating several projects on time-sensitive networking. He is currently an Embedded Software Developer with KUNBUS GmbH, Denkendorf, Germany. His research interests include embedded software development, synchronization, and real-time communication.

Pavel E. Tripolskiy received the engineering and Ph.D. degrees from MIREA-Russian Technological University, Moscow, Russia, in 1991 and 1998, respectively.

He is currently an Associate Professor with MIREA-Russian Technological University. His research interests include PCB design, embedded computing, Internet of Things, and robotics.

Axel Sikora received the Dipl.-Ing. degree in electrical engineering and the Dipl.-Wirt.-Ing. (equivalent to M.B.A.) in 1993 from RWTH Aachen University, Aachen, Germany, and the Ph.D. (Dr.-Ing.) degree in electrical engineering from the Fraunhofer Institute of Microelectronics Circuits and Systems, Duisburg, Germany, in 1995.

After various positions in the telecommunications and semiconductor industry, he became a Professor with Baden-Wuerttemberg Cooperative State University Loerrach, Lörrach, Germany, in 1999. In 2011, he joined Offenburg University of Applied Sciences, Offenburg, Germany, where he currently leads the Institute of Reliable Embedded Systems and Communication Electronics (ivESK). Since 2016, he has also been the Deputy Member of the Board to Hahn-Schickard Association of Applied Research, where he leads the Engineering Divisions and "Software Solutions." He is the author, coauthor, editor, and co-editor of several textbooks and more than 205 peer-reviewed papers in the field of embedded design and wireless–wired networking.