Microprocessors and Microsystems 90 (2022) 104509

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Wireless fieldbus networking with precision time synchronization for a low-power WSAN

Ching-Han Chen [a], Ming-Yi Lin [b,*], Wee-Ping Tew [a]

[a] Department of Computer Science and Information Engineering, Machine Intelligence and Automation Technology (MIAT) Laboratory, National Central University, Taoyuan City 32001, Taiwan
[b] Department of Computer Science and Information Engineering, Taipei City University of Science and Technology, Taiwan (R.O.C.)

ARTICLE INFO

Keywords:
Cyber-physical system (CPS)
GRAFCET virtual machine
Precision time synchronization (PTS)
Wireless sensor actuator network (WSAN)
Industrial internet of things (IIoT)

ABSTRACT

A wireless sensor actuator network (WSAN) is an essential part of the industrial Internet of things (IIoT) with a cyber-physical system (CPS). However, the lack of time synchronization support for WSANs to date means that losses in synchronization accuracy have not been overcome. In this manuscript, we design a real-time kernel to resolve the precision time synchronization (PTS) problem and achieve dynamic automatic synthesis and rapid deployment in a complex WSAN system. Furthermore, tight coupling between wireless control and time synchronization motivates a Modbus-based precision time protocol (PTP) codesign that integrates industrial fieldbus networks and a PTS design. The experimental results indicate that our proposed PTP performance can far exceed the time synchronization requirement of an industrial real-time system. Finally, the proposed solution is successfully applied to an IoT-based smart lighting system.

1. Introduction

Whenever an event occurs in the cyber-physical system (CPS) in a wireless sensor actuator network (WSAN), sensors gather this information and forward it to actuators, which are responsible for making prompt decisions and taking appropriate actions in response to the sensed environmental data. Accordingly, WSANs then change the behavior of the physical systems of their environments. Popular WSAN architectures involve distributed wireless sensor networks (WSNs) that have sensors and actuators at the ends of network edges and are interconnected over wireless links. The sensor data are transmitted back to servers through gateways, and actuator responses and input changes are based on received instructions. Gateways monitor and manage the sensors and actuators of nodes. For example, Fig. 1 shows that smart lighting control is a basic Internet of things (IoT) application of WSANs, in which sensors transmit the received gateway instructions to actuators (typically a relay or dimmer) of smart lighting control systems.

Existing WSAN technologies encounter substantial challenges in providing highly efficient integrated time synchronization with the cloud and time synchronization support due to the lack of effective precision time synchronization (PTS) solutions in industrial precision domains. In particular, PTS is crucial for industrial automation control applications such as the industrial IoT (IIoT) system, industrial automation, and industrial wireless control systems. It is also crucial in industrial sensor networks where big data analytics need to be applied to the aggregate with time synchronization. In short, time synchronization in WSAN systems is a fundamental concern for the coordination of industrial embedded devices and event control in wireless communication environments [1]. However, losses in synchronization accuracy impede proper function. For example, multihop communication over mesh networks further increases the time delays in communication. Second, channel conditions in industrial environments can change dynamically due to external interference, moving obstacles, and weather conditions. Furthermore, in most industrial automation applications, the time synchronization requirement of digital inputs/outputs (DI/DO) is 200 μs. High-performance synchronous motion control requires real-time response times shorter than 250 μs [2]. It is therefore challenging to meet the stringent PTS latency requirements of feedback control in WSANs.

Researchers have recently focused on developing IIoT-related technologies for smart factory, smart manufacturing, and other Industry 4.0 solutions [3]. Several studies have explored the industrial applications of WSANs [4–7]–[7]. One study [4] proposed a configuration and processing mechanism in the form of MidSN software architecture. Every

* Corresponding author.
E-mail addresses: pierre.miat@gmail.com (C.-H. Chen), milin@tpcu.edu.tw (M.-Y. Lin), vincent.tew@g.ncu.edu.tw (W.-P. Tew).

https://doi.org/10.1016/j.micpro.2022.104509
Received 1 November 2019; Received in revised form 17 July 2020; Accepted 12 February 2022
Available online 12 March 2022
0141-9331/© 2022 Elsevier B.V. All rights reserved.

node embedded the same uniform application programming interface and provided highly flexible remote configuration and data processing. A WSAN also provided an integration solution for web-based supervisory control and data acquisition in an electrical power grid [5]. However, most studies on IIoT systems have not considered the overall system hardware cost, power consumption, and other performance indices. Second, they have not addressed the interconnection, intercommunication, interaction, and interoperation requirements of a complex CPS. In [7], the authors categorized system-level design into three domains: upstream system specification, midstream algorithm elaboration, and downstream electronic hardware design. The complex embedded design for a CPS is laborious because it must satisfy the following conditions:

a) Intercommunication between midstream and downstream: Coordinated interaction between different standards and the supervision of various relationships is necessary.
b) The interconnection of complex CPS systems makes decomposition challenging due to the tight coupling between subsystems and submodules. Achieving collaborative teamwork of interoperability is difficult.
c) The interdependencies between automation control and real-time communication increase the load on the network.

These factors are believed to affect the overall performance of IIoT systems. Furthermore, if an internal problem is identified in the design flow, the embedded system development cycle will be prolonged and the cost will increase. For example, a WSAN system based on edge computing uses the received or transmitted information (e.g., DIs/DOs, sensors, actuators, and other diverse automation equipment) to change the behavior of the CPSs. While it is well known in DI/DO design that a low sampling rate typically degrades control performance, a high sampling rate may increase resource contention in bandwidth-constrained WSANs, leading to long communication delays that may result in degraded control performance [5] [6]. Therefore, the main challenge in designing a complex PST system lies in addressing the above requirements for a WSAN system. The activities and main contributions of this work are as follows:

1) A consistent and systematic design methodology was adapted for the rapid development of a complex WSAN PTS system. The proposed GRAFCET virtual machine (GVM) of the real-time kernel was designed to realize dynamic software synthesis for a complex embedded system.
2) An architecture-integrated Modbus-based PTP is proposed to achieve time synchronization and timestamp function codes between the master gateway and slave nodes to enhance the PTS of WSANs. We considered a joint estimation of time offset by invoking Modbus-based PTP and modeling the time synchronization message exchange. This was shown to be efficient.
3) The performance of the proposed Modbus-based PTP was evaluated through implementation in real IoT scenarios. Compared the performance with other industry-related products [2,27,28,]-, the Modbus-based PTP achieves higher accuracy, providing a reference for future research in this area.

The remainder of this manuscript is organized as follows. Motivation based on related work is discussed in Section 2. Real-time kernel design for a WSAN PTS system is introduced in Section 3. The overall architectures of the proposed WSAN PTS system are described in Section 4. Detailed implementations and the application of an IoT-based smart lighting system are discussed in Section 5. Finally, conclusions based on the findings are presented in Section 6.

2. Related work

Time synchronization is a procedure that offers a common notion of time for the WSN system. This is important for correlating data between sensor nodes and communication scheduling. In this section, we present an overview of WSNs based on time synchronization technology and then consider state of the art synchronization algorithms in several study cases [8–18]. Over the last decade, researchers have studied the proliferation of synchronization algorithms for WSN systems; these algorithms include Reference Broadcast Synchronization (RBS), Delay Measurement Clock Synchronization (DMTS), the Timing-Sync Protocol for Sensor Networks (TPSN), Tiny-Sync and Mini-Sync, Lightweight Tree Synchronization (LTS), Temperature Compensated Time Synchronization (TCTS), Pulse Synchronization Protocol, Universal Time Signal Receiver, FLOPSYNC-2, Reverse Flooding, Time-of-Flight Aware Time Synchronization Protocol.

In [8], Elson et al. adapted an RBS protocol for WSN applications in which the receiver uses the physical layer broadcasts to compare the clocks. RBS is so named because it exploits the broadcast property of the wireless communication medium. It provides synchronization amongst a group of wireless nodes within the transmission range of the reference

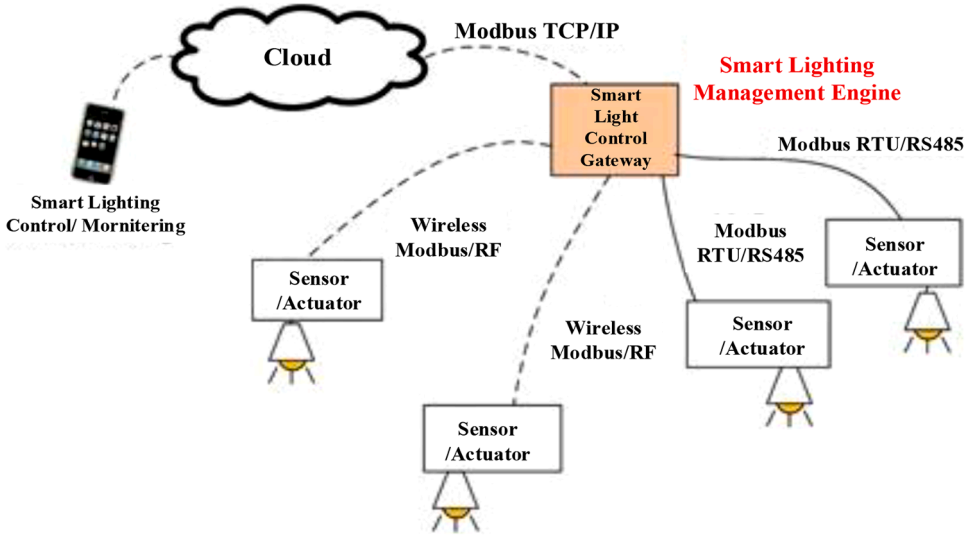


Fig. 1. Smart lighting control with wireless sensor actuator network.

node, which are located within the single-hop broadcast range of a beacon node. The RBS technique entails a receiver with receiver synchronization rather than a sender with receiver synchronization. Therefore, instead of broadcasting a timestamp in a synchronization packet as in protocols such as NTP.RBS, it allows the nodes receiving the synchronization packets to use the packet's time of arrival as a reference node for clock synchronization.

Ping et al [9]. developed DMTS, which is a unidirectional clock synchronization method based on master–slave synchronization and clock correction. The objective of the DMTS method is to avoid the round-trip time. It works to reduce the number of message exchanges in RBS by accurately estimating the delay of the path from the sender to the receiver. The DMTS algorithm has long been established as the most popular clock synchronization algorithm due to its simple principle, low complexity, and energy saving capacity. The DMTS technique is applicable for both single-hop and multi-hop WSNs. One advantage of the DMTS algorithm is that it is quite energy efficient because only one time signal is required to synchronize all the nodes within the time master's transmission range. Furthermore, its computational complexity is lightweight because there are no complex operations involved. Despite the advantages of self-organization and dynamic behavior, DMTS focuses on scalability and flexibility, which means either adapting or being insensitive to changes in the network topology. In addition, synchronization accuracy is traded off against low computational complexity and high energy efficiency. Because the DTMS method involves energy consumption and synchronization accuracy, the latter is weak and is only suitable for WSNs with less precise requirements.

Tiny-Sync and Mini-Sync [10] are two lightweight time synchronization algorithms for WSNs. They are based on the assumption that the frequency of the clock in the sensor networks is constant and has a linear correlation. These two simple algorithms are designed to synchronize a pair of wireless nodes, and then they are extended to synchronize all the network nodes. They use traditional two-way message exchange to collect data points; this estimates the relative clock drift and the relative clock offset. Tiny-Sync uses two-way synchronization multiple times, and the synchronization results are linearly fitted to calculate the frequency offset and phase offset between the two nodes to obtain the clock drift and deviation between them. Mini-Sync is an extension of Tiny-Sync that finds the optimal solution with increasing complexity. The purpose of Mini-Sync is to prevent the algorithm from discarding the data points that may be able to provide strict upper and lower limits. Tiny-Sync and Mini-Sync are both very much in line with the requirements of WSNs, which require storage space and computing resources to be minimal and requirements relating to the bandwidth to be very low. They are extremely suitable for WSNs that are highly constrained in power and bandwidth and are very tolerant of loss message. Both algorithms have a common disadvantage: they are not suitable for mobile WSANs because their networks are logically organized as a hierarchical structure. The convergence time, which is used to achieve time synchronization for the whole network, is also quite high. Finally, the assumptions of these algorithms are difficult to achieve on low-cost sensor nodes.

In WSNs, LTS [11] is a more conservative method of clock synchronization. It is primarily used for global clock synchronization and is based on bidirectional pair synchronization. Its objective is not to maximize accuracy but to reduce the complexity of synchronization. LTS saves energy by sacrificing accuracy and is a relatively simple method. It is therefore more applicable for environments where the accuracy is within the specified range and the complexity is low. The LTS method is divided into two types: centralized multi-hop synchronization and distributed multi-hop synchronization. The advantage of the LTS method is that it saves network overhead and substantial network resources and avoids the network resource consumption caused by communication with other upper-layer nodes. The LTS synchronization algorithm is as important as the aforementioned solutions but, due to its high energy consumption, is not very effective because physical clock correction needs to be performed on the local clock of nodes while achieving synchronization. However, the LTS method is generally precise and can only be used for accuracy requirements in a small environment. Thus, the timing accuracy required is supposed to be given as a constraint and the main objective is to have a synchronization algorithm with minimal complexity to achieve given accuracy. In addition, it is not recommended for use in mobile networks because it requires a hierarchical infrastructure with high mobility.

TCTS exploits the temperature sensor that exists in several sensor network platforms proposed in [12]. The main concept underlying TCTS is to calculate the temperature during frequency error estimation. Having determined the relationship between frequency error and temperature, the node can back off and maximize the time between synchronization slots while adapting its frequency error estimation from temperature measurements. Thus, the protocol comprises two approaches: calibration and compensation. TCTS uses a temperature sensor to autonomously calibrate the local oscillator and eradicate the effects of environmental temperature changes. This allows a time synchronization protocol to increase its resynchronization period without losing synchronization accuracy, saving energy and communication overhead. In addition, TCTS is a starting point in a move towards automatic temperature measurement of a local clock. It provides a stable clock source when radio communication is impaired. Another advantage of TCTS over a legacy time synchronization protocol is that it can be used in scenarios where the connection is lost.

In [13], the authors discussed a new clock synchronization algorithm known as the Pulse Synchronization Protocol (PulseSync) that is asymptotically optimal. The main concept (central idea) of PulseSync is to distribute information on clock values as fast as possible while minimizing the number of messages required to achieve this. The authors evaluated PulseSync on a large number of testbeds and tightly synchronized large networks. The experimental results demonstrated that PulseSync is able to meet the theoretical predictions and offers much better accuracy and latency than the Flooding Time Synchronization Protocol (FTSP). The authors showed that the fast flooding approach of PulseSync achieves significantly improved synchronization accuracy on the same network topology. Furthermore, their experiments confirmed that because the synchronization error to the root node must increase exponentially when using FTSP, PulseSync performs better. The synchronization error of FTSP indicates exponential growth with increasing hop count, whereas the synchronization error of PulseSync increases with the square-root of the hop count.

In [14], the authors presented an inexpensive and ultralow-power mote peripheral known as the Universal Time Signal Receiver (UTSR). UTSR uses broadcast time stations and hardware-assisted clock synchronization; it leverages the availability of time signals transmitted by dedicated radio stations worldwide to provide access to UTC time with millisecond-level accuracy. The authors presented experimental results measuring signal availability, quality of synchronization across motes, and power consumption. These indicated that the proposed UTSR achieves global time synchronization and, for applications where millisecond-level precision is sufficient, consumes up to 1000 times less energy than global positioning system (GPS) or FTSP.

In [15], the authors presented the FLOPSYNC-2 scheme to synchronize the clocks of multiple nodes in a WSN, which requires no additional hardware and is based on the application of theoretical control principles. In FLOPSYNC-2, a message-passing synchronization scheme relies on a tailored controller structure and guarantees low power consumption, low overhead, and synchronization with clock monotonicity. The authors proposed implementing FLOPSYNC-2 on top of the operating system Miosix and demonstrated its validity through experiments on an eight-hop network. The results showed that the average clock difference among nodes was limited to 0.01 ns, with a sub-μs standard deviation.

In [16], the authors proposed a method to estimate and compensate for network propagation delays. Known as the Reverse Flooding protocol, this does not need to maintain the spanning tree of a WSN and

exploits constructive interference, even in the transmission of packets whose contents are slightly different. To demonstrate the validity of the approach, the authors implemented the propagation delay estimator on top of the FLOPSYNC-2 synchronization scheme. The experimental results affirmed the feasibility of measuring propagation delays using off-the-shelf microcontrollers and radio transceivers. They also showed how Reverse Flooding enables submicrosecond clock synchronization to be achieved, even for networks where propagation delays are significant.

Lim et al. recently developed a Time-of-Flight Aware Time Synchronization Protocol (TATS) [17] that combines fast flooding and message delay compensation for wireless embedded systems. As mentioned previously, TATS significantly improves time synchronization accuracy. It achieves a submicrosecond synchronization error but is not involved in isolated nodes. It can even synchronize networks with large diameters of up to 22 hops with submicrosecond accuracy. Experiments conducted on a testbed that closely resemble real deployment scenarios with respect to node distances show that TATS achieves up to 6.9 times greater accuracy than state-of-the-art protocols.

An FTSP was proposed in [18] by Miklós Maróti et al. that provides network-wide internal synchronization with local participating nodes. It assumes that each node has local clock synchronization errors and must be corrected with the message exchange mechanism. The mechanism ensures high accuracy between nodes and maintains synchronized communication. FTSP synchronizes time from a sender to multiple receivers using a single radio message. In FTSP, the root node is the only selected and dynamic node that maintains the overall time for all other nodes to synchronize their clocks. The root node serves as the source of the reference time and periodically broadcasts the network with a synchronization message that contains its local time value. Sensor nodes perform linear regression on the incoming timestamp and time of arrival pairs to estimate the clock offset and the drift from the root node.

As described previously, numerous protocols already have been described in the literature for time synchronizing the sensor nodes in WSNs. A wealth of studies have used synchronization algorithms in the design of WSN PTS systems to improve the performance of the processing system. These algorithms have been improved and innovated in terms of accuracy, robustness, stability, overhead, and so on. In the current study, we propose a synchronization protocol based on the IEEE 1588–2008 precision time protocol (PTP) [19] combined with the Modbus automation standard and refer to it as the Modbus-based PTP. Its performance was tested on a WSAN system used for measuring and correcting time offset and network delays.

3. Real-time kernel design for a WSAN PTS system

This section emphasizes the importance of real-time kernel design and automatic software synthesis in a complex embedded system. It provides an effective and efficient design methodology for complex WSAN PTS system design and dynamic software synthesis. The real-time kernel is set to play an important role in WSANs and is used for the PTS of system operations. In this methodology, which is based on GRAFCET formalism for hierarchical design, behavioral modeling and the automatic software synthesis of a complex WSAN system that satisfies the definition of a PTS system with multiple wireless slaves were performed.

3.1. Methodology for real-time kernel design

Fig. 2 shows the top-level schema of the high-level modeling and synthesis methodology and presents the workflow model, which comprises five macrosteps: system specification, discrete-event modeling, dynamic software synthesis, function simulation and verification, and hardware verification. The GVM forms an important core in the gateway; it manages the running of the coding tables sent by users and assigns tasks to slave nodes for operation Fig. 3. presents the GVM workflow of the real-time kernel for deploying and configuring servers, gateway, and DIs/DOs of embedded devices with operations at the beginning of the deployment, thus providing configuration flexibility prior to adjustment with remote configuration.

As shown in Fig. 3, the GVM workflow of the real-time kernel for the constructed IIoT system includes a cloud service module for system function definition using a graphical language consisting of the GRAFCET Application (GPP) editor and GRAFCET translator. Users can map out system specifications using the GPP editor while the GRAFCET translator automatically parses graphics or text into important coding tables. Additionally, the gateway module uses the GVM as a real-time kernel for the rapid deployment and configuration of the IIoT system with the cloud. As illustrated in Fig. 3, we defined system specifications that must be satisfied for the functions and behaviors of an IIoT system. The behavior of each independent functional module is represented as a discrete-event system (DES) using the GRAFCET discrete-event modeling tool and then is synthesized as a coding table. The GPP editor is then adopted to model the individual DES module and the hierarchical, pipelined behaviors of the system. The GRAFCET translator transmits the coding table to the GVM through the cloud. Finally, the user can communicate and control the system remotely with sensors, actuators, electronic devices, and industrial equipment using middleware provided by the smart gateway middle tier. However, a compiler is not needed to achieve the requirements of effective reusable design.

3.2. Discrete-Event modeling using GRAFCET

The DES is a discrete-state and event-driven system that includes the space of discrete states and the events of state changes. Our design methodology for the GVM used GRAFCET because this can describe the evolution order of numerous states in the DES. In addition, GRAFCET

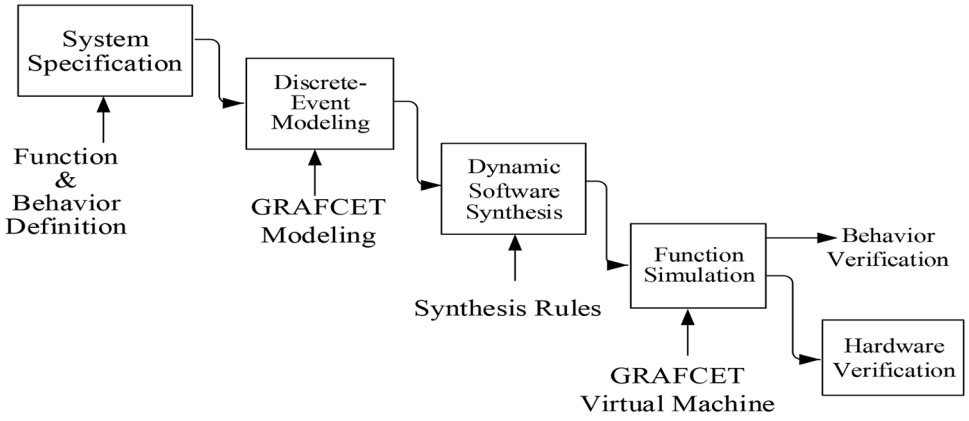


Fig. 2. High-level modeling and synthesis methodology.

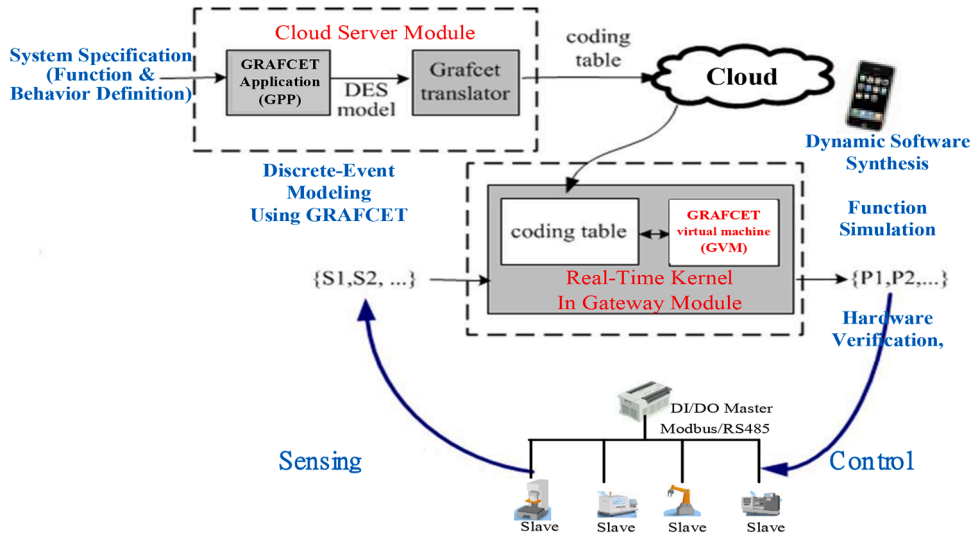


Fig. 3. GVM workflow of the real-time kernel for an IIoT system.

provides a simple and intuitive formalism for DESs. It contains a sequences of steps, transitions to connect consecutive steps, and directed connections.

The initial step in a DES is represented by a double-bordered square associated with a unique number. A single-bordered square represents a normal step (see Fig. 4). A normal step is either active or inactive and is typically connected to an action. The triggered action is executed only when the corresponding step is activated; the square is then marked by a black dot indicating token acquisition. However, if the step is not activated, the token is transferred to the next step and the action is not allowed. A vertical line is used for connecting two consecutive steps, and a transition represents the possibility of changes in system behaviors. A condition is represented by a horizontal line between steps or transitions. Each directed connection links a step to a transition or a transition to a step; steps and transitions always appear alternately. When a given step is active and the transition condition is fulfilled, the doAction step and the doTransition step become simultaneously inactive and active, respectively. A condition can be associated with these actions, and the action is executed only if the step is active and the condition is satisfied.

A hierarchical GRAFCET model can be represented by a specific hierarchical structure which in this software architecture is similar to that of a main program and its subfunctions. The hierarchical structure in Fig. 5 can be extended consecutively to the lower layer following the aforementioned protocol. The system is therefore divided into coarse-to-fine submodules, and each submodule is independently designed. Due to the simple master–slave communication protocol, the design of a complex computing system can be divided into a set of decoupled tasks whose results are ultimately integrated. Because GRAFCET is a rigorous language structure with states and transitions, formulating a set of synthesis rules to map each GRAFCET block into the software program architecture is straightforward.

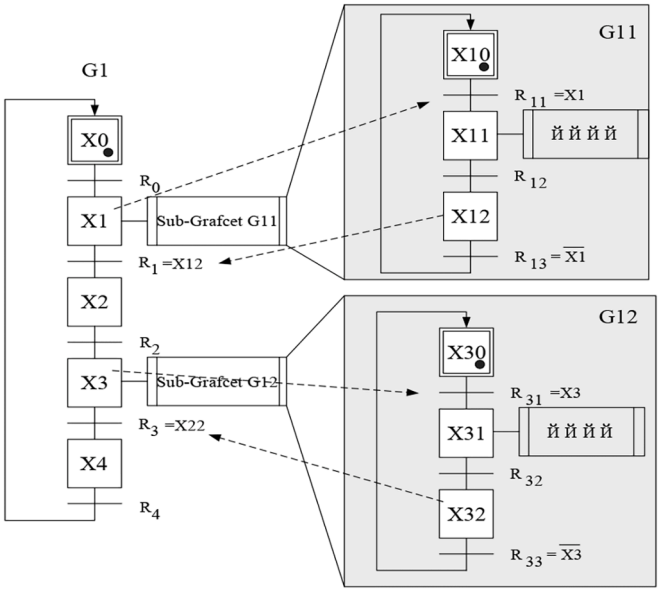


Fig. 5. Hierarchical discrete-event modeling by GRAFCET.

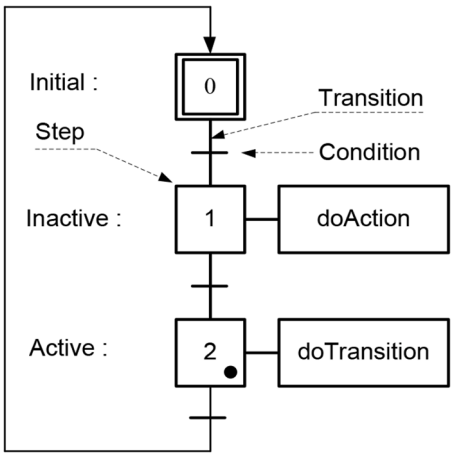


Fig. 4. Elements of the GRAFCET graphical language.

The GVM can therefore be used within process tasks and is divided into three parts: doAction, doTransition, and doCondition. The GVM process tasks run as directed by the coding table; the automaton undergoes a state transition and is instructed to take the appropriate action (see Fig. 4). The doAction part implements the current state of the action description of the recorded function. If the state has an action description then the system can run the corresponding action when the system is in that state. The doCondition task determines the conditions; the answer determines the address of the next state to which the automaton is directed.

3.3. GRAFCET virtual machine for real-time kernel

The GVM of a real-time kernel allows users to quickly build system models using GRAFCET. This solution also obviates the practice of burning completely static and node-specific code into each device. Instead, rapid configuration commands are provided remotely over the WSAN system. A web-based GPP can provide dynamic software synthesis because its development environment is not confined to the desktop. Application system development can be performed remotely using mobile phones or tablets. With the GPP, remote users can simply describe application system behaviors using GRAFCET. The GVM is a meta-program that interprets the GRAFCET system. Users can describe the system with GRAFCET through the GPP editor, convert the GRAFCET graphics into a coding table, and allow interpretation using the GVM (see Fig. 6). Further details on the GVM can be found in [20–23]. In [20], the author designed the embedded system architecture of the GVM and implements it with a microcontroller. Base on this software and hardware, the author proposed a fast prototyping development platform for the smart DES and CTS hybrid controller. The author applied embedded GVM as the embedded robot development platform in [21]. The author implemented the embedded part of this platform on a microcontroller called SIOC. A GUI based application for the Android mobile device platform was also implemented. A developer can easily use this platform as a rapid development platform for developing robot applications. In [23], the author designed four embedded middleware modules based on GVM: display, communications, sensing and actuation for reducing significantly the develop-debug-deploy cycle. Besides, the author used two tested applications: the electronic label network and IOT-based digital Signage to demonstrate the practicability and rapidity of system development by using the proposed rapid prototyping platform. The author proposed an IoT embedded system development platform using GVM as a microkernel by [23]. It allows users to quickly build the system model. The author designed a Web GRAFCET Application (GPP) editor, making the development environment is no longer confined to the desktop computer. Application system development can be made remotely by mobile phones or tablets. With GPP, remote developers can simply describe application system behaviors by using GRAFCET.

4. System design of the WSAN PTS system

In this section, we first introduce our design concept and the overall WSAN PTS system architecture design. The Modbus-based PTP messages and timestamp exchange model are then described in Section 4.2. Finally, Section 3.3 describes GRAFCET modeling for the WSAN PTS system. The main concept of our proposed Modbus-based PTP protocol in this manuscript is based on an IEEE 1588–2008 PTP combined with the Modbus automation standard of the application layer. Unlike most applications of an IEEE 1588 PTP in the literature, the proposed protocol does not through User Datagram Protocol (UDP) of the network layer or Ethernet of data link layer. The Modbus-based PTP protocol adopts PTP packets embedded with Modbus function code through which to transmit and record timestamp and master and slave and to write/read an instruction-driven sensor and actuator. It not only implements the time synchronization between master and slave, it can also monitor slave nodes, manipulate actuators, and report data through an industrial fieldbus communication protocol in the perception layer.

4.1. Overall architecture of the WSAN PTS system

Fig. 7 portrays the overall architecture of the proposed WSAN PTS system, which comprises three parts: cloud server, master gateway, and slave nodes. The cloud server is a networking module at the management layer, a remote monitor, and supervises all the control and environment information. A server is connected for system function definition using of GPP editor. It provides a graphical user interface (GUI) enabling users to manage and store the lowest-level actuator and sensor information. It primarily allows users to design and observe the behaviors of slave nodes. Due to the increasing prevalence of smartphones and mobile platform systems, the server GUI works as a web-based GPP so that the system only needs to manage one application. Thus, there is no need to design several applications for different embedded system development platforms.

The master gateway module uses the GVM as a real-time kernel for the rapid deployment of the IIoT system with the cloud and distributed slave nodes for the time synchronization operation and IIoT management. As illustrated in Fig. 8, the Modbus transmission control protocol/internet protocol (TCP/IP) model is responsible for server–master gateway communication. The intermediary master gateway adopts the Modbus remote terminal unit (RTU) mode to communicate with slave nodes. The Modbus adaptor is therefore designed in the master gateway to oversee server–gateway–slave communications.

As described in Section 3, users use the GRAFCET graphical interface to describe the discrete event behaviors of all system specifications in the WSAN PTS system. The master gateway uses the GVM to control the discrete event behaviors for the rapid configuration of the WSAN

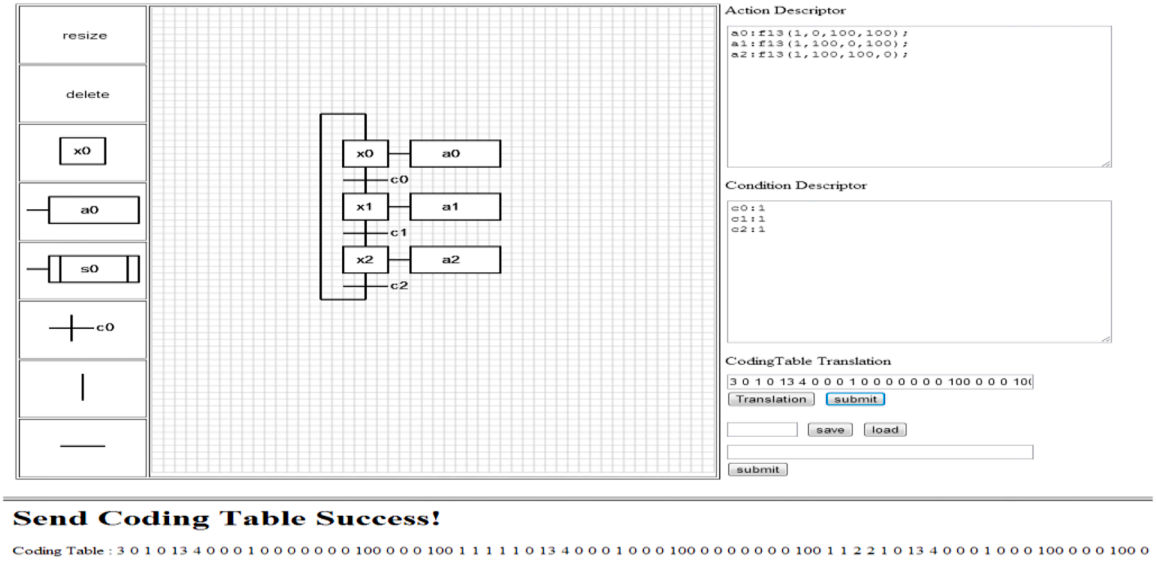


Fig. 6. GRAFCET virtual machine.

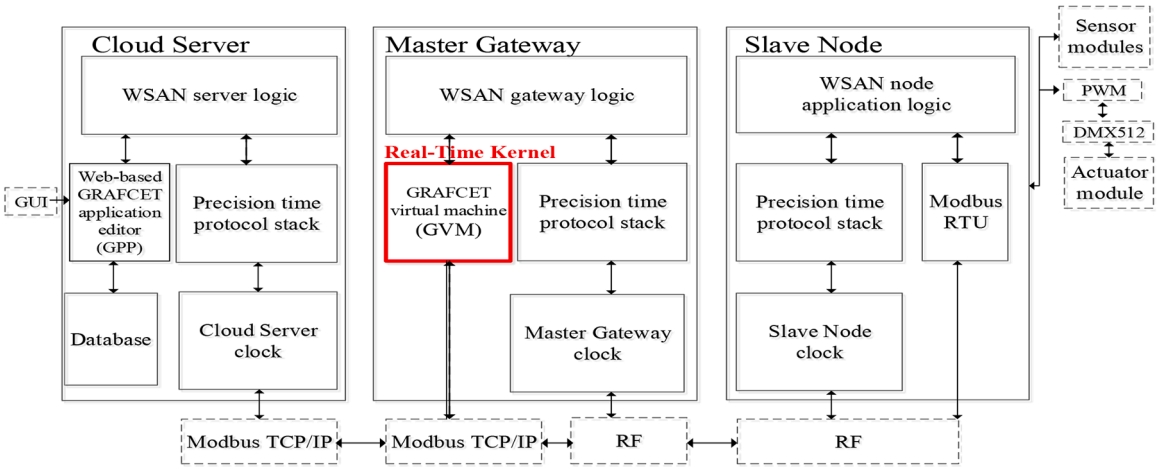


Fig. 7. WSAN PTS system architecture.

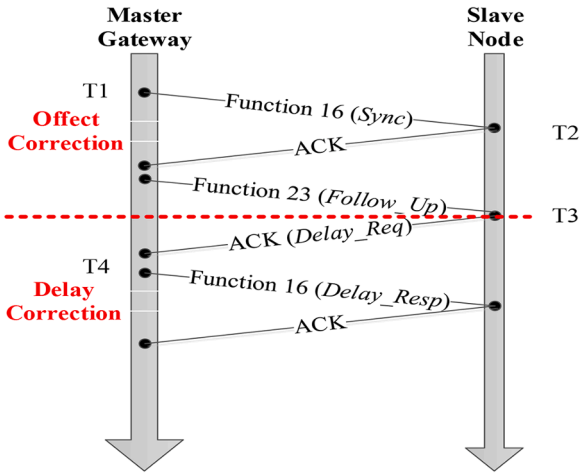


Fig. 8. Messages and timestamp exchange mode of the synchronization algorithm.

system. As a real-time kernel, the GVM is set to play an important role in the gateway with PTS and is used for numerous major operations. The master gateway decodes the Modbus TCP/IP packet transmitted from the server, converts it into corresponding behaviors through the GVM, converts the information obtained into Modbus RTU packet format, and eventually transmits the converted information to slave nodes through a 2.4 GHz ISM (Industrial, Scientific, Medical Band) radio frequency (RF) module. The slave nodes also adopt a digital multiplex standard (DMX512) through the pulse width modulation (PWM) signal to control actuators and equipment on node devices, allowing such devices to respond appropriately with precise timing.

The proposed Modbus-based PTP provides time synchronization in WSANs. In this network architecture, running the best master clock (BMC) algorithm of the IEEE 1588 PTP means that the highest-property clock is considered to be the grandmaster clock and can create a master–slave hierarchy. Both the master clock and slave clock can be synchronized from an external grandmaster clock in the WSAN PTS system. To synchronize the master clock and slave clock with the grandmaster clock, extremely accurate timestamps need to be provided for the Modbus-based PTP protocol.

4.2. The Modbus-based PTP messages and timestamp exchange model

The proposed Modbus-based PTP protocol based on the IEEE 1588 PTP was combined with the Modbus standard to provide precision time synchronization in WSANs. A typical conventional time synchronization system uses the IEEE 1588 PTP as its core standard. IEEE 1588 PTP defines the following four packet message types: *Sync, Follow_Up, Delay_Req,* and *Delay_Resp*. These packets are used by ordinary and boundary clocks and communicate the time-related information used to synchronize clocks across the network [24]. The PTP uses a specific BMC algorithm that determines which clock among the communicating clocks within the network is of the highest property. The algorithm performs a distributed selection of the best candidate clock based on the following clock properties: identifier, quality, priority, and variance. By running the BMC algorithm, each clock compares the aforementioned properties and identifies the property clock to build the synchronization distribution master–slave hierarchy among the PTP clocks in the network. The highest-ranking property clock is called the grandmaster clock and synchronizes all other clocks in the network. The BMC algorithm is then run continuously to quickly adjust for changes in network configuration. If the grandmaster clock is removed from the network or is determined by the BMC algorithm to no longer be the highest-quality clock, the algorithm provides a means for participating clocks to automatically determine the new grandmaster clock. The PTP makes the following key assumptions that affect clock accuracy: transmission delays are symmetrical between master and slave and transmission delays are almost constant or at least change slowly.

Among all the fieldbus communication protocols, Modbus was selected on the basis of advantages such as ease of use, openness, high price-to-performance ratio, and high reliability. Thus, Modbus has become a de facto industry standard. It is an application-layer messaging protocol, positioned at level 7 of the Open Systems Interconnection model, that provides master–slave architecture between devices connected on different types of buses or networks. Modbus offers two basic communication mechanisms: question/answer (polling) and broadcast. It can easily be used to monitor and program slave nodes and allow intelligent devices to communicate with sensors and actuators. In [3, 7, 25], and [26], the authors demonstrated that the Modbus protocol is suitable for use in the sensor network of the IIoT system. Moreover, Modbus supports many variant modes of different types of buses or networks, such as Modbus RTU, Modbus TCP/IP, Modbus UDP, Modbus Plus, Modbus ASCII, and so on. In this work, we use Modbus RTU mode and Modbus TCP/IP mode for server–gateway–node communication. Our proposed Modbus-based PTP protocol, based on Modbus TCP/IP and the Modbus RTU, enables mutual correspondence using a Modbus adaptor. Additionally, Modbus's point of departure is the concept of the master–slave principle that, like PTP, describes a hierarchical master–slave architecture for clock distribution and appears to show they are both symmetrical.

$$\text{Offset} = ((T2 - T1) - (T4 - T3))/2 \quad (1)$$

$$\text{Delay} = ((T2 - T1) + (T4 - T3))/2 \quad (2)$$

As illustrated in Fig. 8, our proposed Modbus-based PTP protocol operates between the master gateway and slave nodes. First, the master sends a *Sync* message through Modbus function code 16 and records the master timestamp (T1). On receiving the packet, the slave node records the receipt of timestamp (T2) and sends a Modbus acknowledgement (ACK) back to the master gateway. The master gateway sends the *Follow_Up* message through Modbus function code 23 and sends T1 while reading the *Delay_Req* packet of the slave node. The slave node then calculates the time offset (1) between itself and the master, sends the *Delay_Req* packet, and records the slave node timestamp (T3). The master gateway receives the *Delay_Req* packet sent by the slave and then uses Modbus function code 16 to transmit the *Delay_Resp* packet of time received by the master gateway (T4) to the slave. Eventually, the slave node calculates the time delay (2) and sends the Modbus ACK back to the master gateway, marking the completion of the master–slave time synchronization.

The Modbus-based PTP protocol provides a method of offset measurement that can be used to synchronize all participating clocks to the highest-property clock in the network. Because the time difference between the master gateway clock and slave node clock is a combination of the clock offset and message transmission delay.The time synchronization is divided into the offset correction and delay correction of two phases. In the first phase, the slave nodes correct their time difference with the offset measurement. In so doing, the master gateway sends the message to the connected slave node cyclically at a defined synchronization period of 2 s. This *Sync* message contains the current time of the master gateway clock. Thus, the actual transmission time of the message is measured as close as possible to the transmitter-receiver of the physical interface. This information is then sent to the slave node by means of a second *Follow_Up* message. Upon receipt of this message, the slave node calculates the correction value of the offset to the master gateway by considering the reception timestamp of the *Sync* message. The clock of the slave node must then be corrected by this offset. If there is no delay on the transmission line, both the master gateway clock and the slave node clocks will be in synchronization. The difference between the two timestamps represents the offset of the slave plus the message transmission delay. The slave node clock then adjusts the local clock in line with this difference. To correct for the message transmission delay, the slave node uses a second set of *Sync* and *Follow_Up* messages with its corrected clock to calculate the master-to-slave delay. The second phase is the delay measurement of the synchronization process, which determines the delay time between the slave node and the master gateway. The second set of messages is needed to account for variations in network delays. To achieve this, the slave node sends a *Delay_Req* packet to the master gateway and determines the exact transmission time of the message. On receiving the packet, the master gateway generates a timestamp and sends the reception time back to the slave in a *Delay_Resp* packet. From the local transmission and reception timestamp of the master gateway, the slave determines the delay time between the slave node and the master gateway. The difference between the timestamps is the slave node-to-master gateway delay. The slave node averages the bidirectional delays and then adjusts the clock by the length of the delay to synchronize the two clocks.

4.3. GRAFCET modeling for the WSAN PTS system

This section describes the synchronization algorithm of GRAFCET modeling for the WSAN PTS system. As illustrated in Fig. 8, our proposed Modbus-based PTP protocol operates between the master gateway and slave nodes. The protocol based on the Modbus TCP/IP and Modbus RTU enables mutual correspondence using the Modbus adaptor in the master gateway. The synchronization algorithm of the protocol is divided into three parts: the master gateway, slave nodes, and Modbus adaptor. In this section, for each independent DES module, we present the GRAFCET modeling for translation into a software architecture for the WSAN PST system.

The core of the WSAN is the master gateway with an embedded GVM that synchronizes and controls the perception-layer slave nodes. Enables big data collection by the slave network and manipulates interfaces for instruction exchange. The GVM forms the kernel of the gateway; it manages the running of the coding tables sent by users and assigns tasks to slave nodes for operation. If the packet consists of a coding table, the GVM renews its current runtime behavior. The GRAFCET modeling for the master gateway can be divided into several steps, as shown in Fig. 9. The **TimeSync** function submodule is responsible for every PTP type message exchange between the master and slave ports. It is implemented the time synchronization process between the master gateway and slave nodes.

The multiple slave nodes of the perception layer are connected to the master gateway through the 2.4 G ISM (Industrial, Scientific, Medical Band) RF module. When the master gateway makes a request to implement the process of time synchronization (TimeSync), it sends a PTP packet (*Sync* message) containing a timestamp to the slave nodes. The slave nodes record the timestamp received in the PTP packet through Modbus function codes into their RTC. After receiving the *Sync* message, the slave nodes send a *Delay_Req* message back to the master gateway. If this is not a PTP packet, slave nodes realize the corresponding behavior of the DMX512 standard using the PWM control signal. If the packet is not received, slave nodes provide data processing for the upper-layer network, including data reports, and manipulate actuator responses. The GRAFCET modeling for the slave nodes can be divided into several steps, as shown in Fig. 10. The **TimeSync** function submodule is responsible for every PTP type message exchange between the master and slave ports. It is implemented the time synchronization process between the master gateway and slave nodes.

As illustrated in Fig. 8 and described in relation to the master gateway in Section 4.3, the proposed WSAN PTS system uses the Modbus TCP/IP and Modbus RTU for system communication; the Modbus adaptor in the master gateway is designed to be in charge of server–gateway–slave communications. The GRAFCET modeling for the Modbus adaptor is illustrated in Fig. 11. When a received Modbus TCP packet is converted into an RTU packet (see Fig. 11(a)), the Modbus TCP header is recorded. This is followed by placement of the unit identification number as well as content into the RTU packet before the cyclic redundancy check (CRC) code calculations are performed and the RTU packet reply length is sent. When a received Modbus RTU packet is converted into a TCP packet (see Fig. 11(b)), the previously received Modbus TCP header is recorded to analyze the length of the TCP packet. This is followed by verification of the length of the received RTU packet. If the length is confirmed to be accurate, the Modbus TCP header is developed before the content of the body is written. The length of the TCP packet is then sent back.

(a) Modbus TCP packet converted into an RTU packet

(b) Modbus RTU packet converted into a TCP packet

The WSAN system also consists of a master gateway controlling the communication and a number of slave nodes. The Modbus adaptor of the master gateway is used to monitor the slave nodes of the perception layer through the Unit Identifier of the Modbus Application Protocol header. To verify the network functions, the Modbus adaptor of the master gateway provides 14 types of error report signals, which are shown in Table 1. The WSAN system can monitor real-time sensor node states. For example, if the sensor nodes on the network are removed, error signals occur: (0 × 14, MB_EX_SENSOR_REEOR). Thus, the proposed WSAN PTS system not only meets the precision time synchronization definition of PTP but also satisfies the definition of an IIoT smart sensor.

M = Master, S = Slave

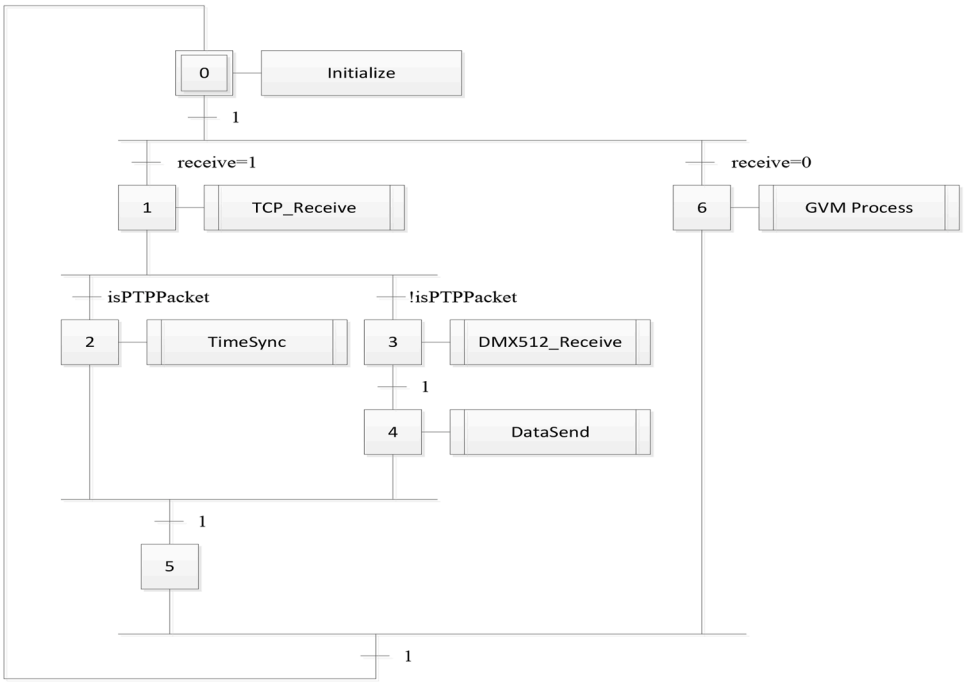


Fig. 9. GRAFCET modeling for the master gateway.

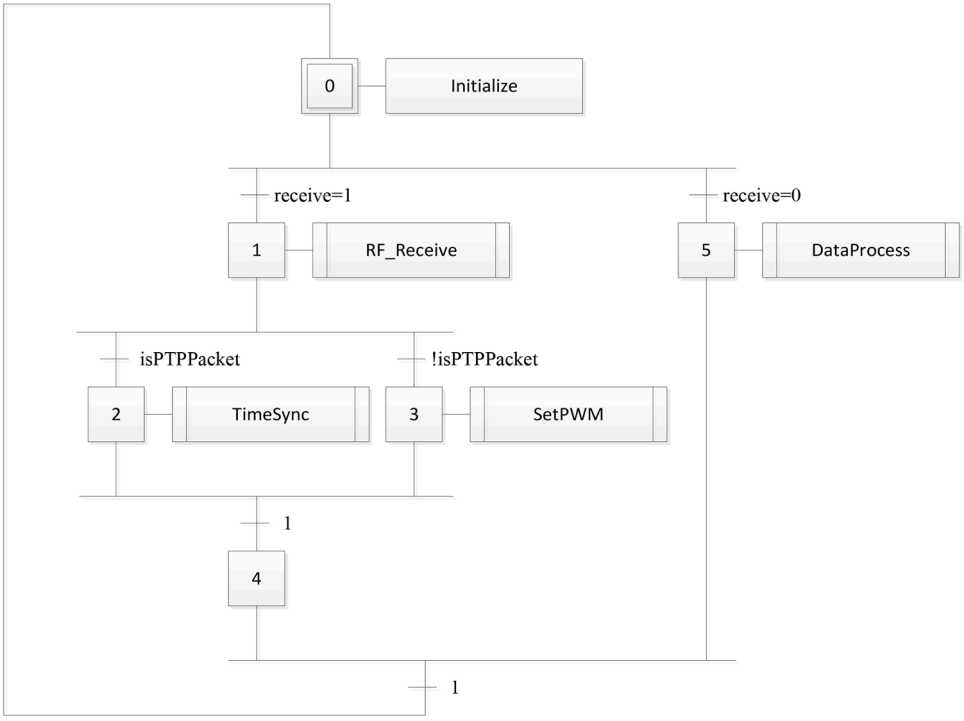


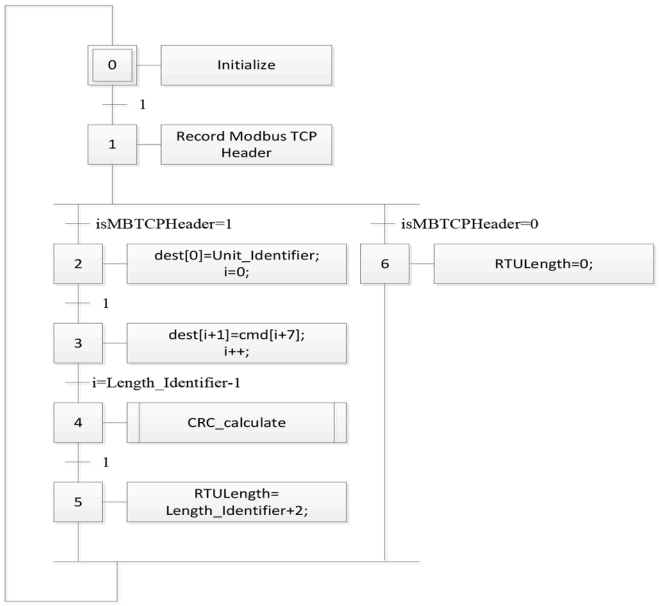
Fig. 10. GRAFCET modeling for the slave node.

5. Implementation

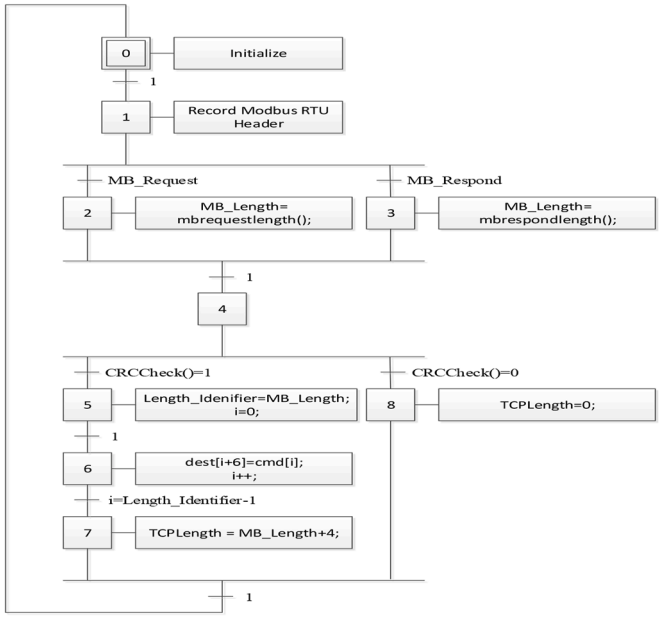
After calculating Offset (1) and Delay (2), adjust the clock of the slave node according to these two values to complete a time synchronization process. As shown in Fig. 8, a complete PTP process involves polling (defined synchronization period of 2 s) using the master gateway to two slave nodes and the reception of response messages (ACK) returned from the slave. Both the master gateway and the slave nodes were implemented and tested on a real ARM Cortex-M3 microcontroller. For validation purposes, individual tests of time synchronization were conducted 1000 times, following which the mean time of the clock offsets and the network delays were measured by Fig. 8 Messages and timestamp exchange mode.

5.1. Performance evaluation

The synchronization results of offset and delay measurements are presented in Table 2 and Fig. 12. As shown in Table 2, we enumerate 10



(a) Modbus TCP packet converted into an RTU packet



(b) Modbus RTU packet converted into a TCP packet

Fig. 11. GRAFCET modeling for the Modbus adaptor.

sets comprising the number of sampling times, time offset, time delay, success rate, packet loss rate, and retransmission rate. In the results, the success rates of transmission were 100% with a packet loss rate and retransmission rate of 0. A distribution diagram of offset is shown in Fig. 12; the mean time of offset is 9.61 ms. A distribution diagram of delay is shown in Fig. 13; the mean time of delay is 2.502 ms.

Finally, the clocks shift 2.502 ms to adjust the delay error. By adjusting clocks, the delay error in Table 3 was obtained, the success rates of transmission were 100%, and the retransmission rate was 0. To verify our results, Fig. 14 shows that the standard deviation of the synchronization delay Table 3. shows that the overall mean delay time of the final synchronization results.

In most industrial automation applications, the time synchronization requirement of DI/DO is 200 μs, which was declared in [2]. In this work, our experimental evaluation compared the performance with other industry-related products (PCIE-1203L[2], PCI-L221-B1D0 [27],

Table 1
Smart sensor function codes of a Modbus adaptor in a master gateway.

Function Name	Function code	Explanation	Note
MB_EX_NONE	0×00	No error	M/S
MB_EX_ILLEGAL_FUNCTION	0×01	Illegal function code	S
MB_EX_ILLEGAL_DATA_ADDRESS	0×02	Illegal memory address	S
MB_EX_ILLEGAL_DATA_VALUE	0×03	Illegal data	S
MB_EX_MEMORY_PARITY_ERROR	0×08	Memory error	M
MB_EX_COMMUNICATION_FAILURE	0×0C	Communication failure	M
MB_EX_NOT_CONNECT_DEVICE	0×0D	Lost device	M
MB_EX_ID_NO_INCONSISTENCY	0×0E	ID inconsistency	M
MB_EX_FUNCTION_CODE_ERROR	0×0F	Code inconsistency	M
MB_EX_FORMAT_TOO_LONG	0×10	Format too long	M/S
MB_EX_FORMAT_TOO_SHORT	0×11	Format too short	M/S
MB_EX_RESPONSE_TIMEOUT	0×12	Response time out	M
MB_EX_CRC_ABNORMAL	0×13	CRC code error	M/S
MB_EX_SENSOR_REEOR	0×14	Sensor node error	S

Table 2
Experiment results of offset and delay measurements.

Synchronization (100 Mean time)	Offset (ms)	Delay (ms)	Success rate (%)	Packet loss rate (%)	Retransmission rate (%)
1	15.7	2.48	100	0	0
2	12.02	2.86	100	0	0
3	11.2	2.51	100	0	0
4	10.7	2.41	100	0	0
5	9.26	2.52	100	0	0
6	8.63	2.59	100	0	0
7	8.03	2.44	100	0	0
8	8	2.2	100	0	0
9	6.4	2.57	100	0	0
10	6.2	2.44	100	0	0
Average	9.61	2.502	100	0	0

MMC-EtherCAT [28]). The result presented that the standard deviation of the synchronization delay remains below 4 μs, far exceeding the industrial real-time system requirement. Moreover, different noise sources affect pocket timestamping capabilities of RF transceiver on a real wireless sensor node. We present a simple method to improve both accuracy and reliability. This highlights the accuracy of the end-to-end clock of the proposed Modbus-based PTP solution, providing a reference for future research in this area.

5.2. Implementation of the smart lighting system

The focus of this section is on the smart lighting system: the DMX standard was adopted, and light-emitting diode (LED) control commands were converted into digital signals through DMX512 to control the color and brightness of the LEDs. The protocol can reduce the burden of lighting designers and operators and enable remote network monitoring. The WSAN of the smart lighting system consists of one master gateway and four slave nodes (see Fig. 15). In this study, the prototype of the smart lighting system was implemented using the hardware resources presented in Table 4.

The master gateway acted as the communication interface for application and node devices through Modbus. By integrating with the GVM rapid development platform and web-based GPP on the cloud server, developers were able to obtain the desired flow by editing. This was transmitted to the GVM on the gateway via Ethernet for implementation; control messages were then sent to each node through the 2.4 G RF module and each node was controlled remotely. The web-based GPP of the cloud server provides an editing interface where users can manage and deploy node behaviors to adjust LED lighting and

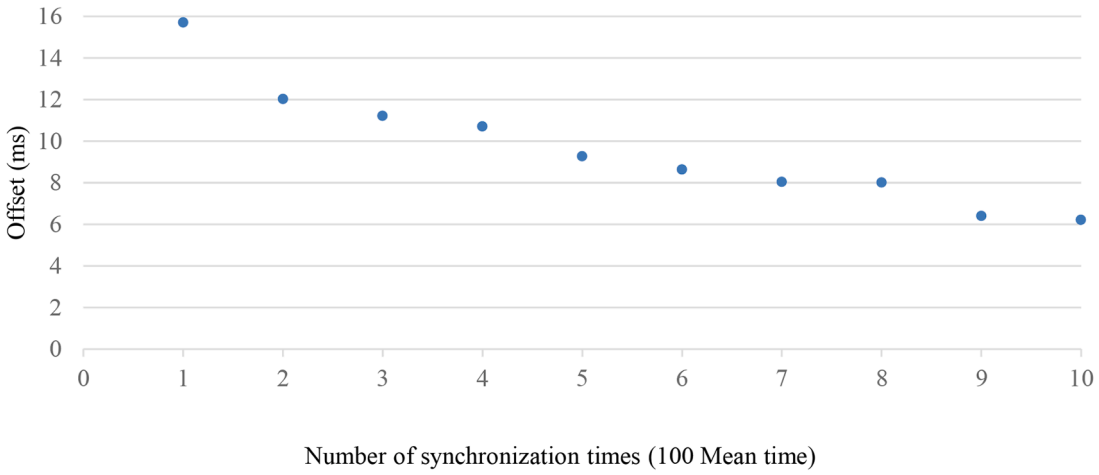


Fig. 12. Distribution diagram of offset measurements.

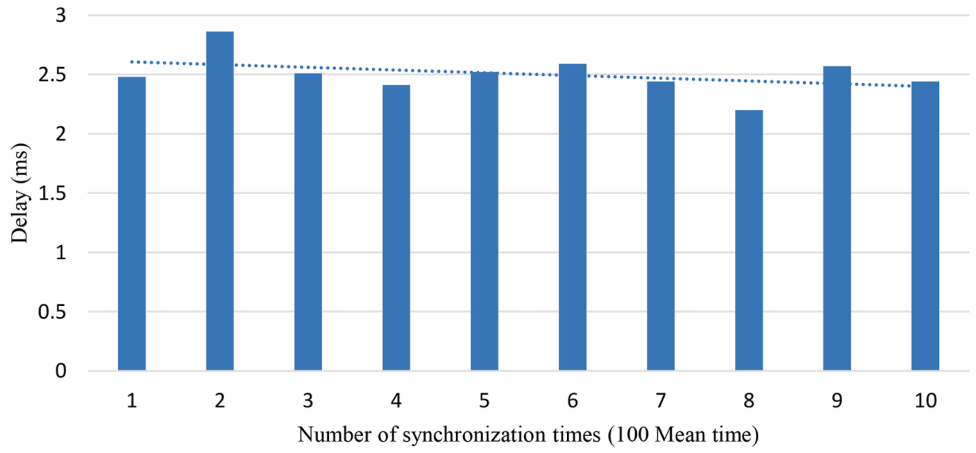


Fig. 13. Distribution diagram of delay measurements .

Table 3
The final synchronization results.

Synchronization (100 Mean time)	Delay (ms)	Success rate (%)	Packet loss rate (%)	Retransmission rate (%)
1	0.001070	100	0	0
2	0.001180	100	0	0
3	0.003450	100	0	0
4	-0.003880	100	0	0
5	0.001050	100	0	0
6	-0.000909	100	0	0
7	-0.002170	100	0	0
8	-0.003010	100	0	0
9	0.000731	100	0	0
10	0.000911	100	0	0
11	0.000951	100	0	0
Average	-0.000057	100	0	0

environmental brightness.

After the system was initialized, the master gateway would first analyze whether the receiving packet required time synchronization. If the master gateway sends the *Sync* message to request the process of time synchronization, the GVM forms the core of the gateway; it manages the running of the coding tables sent by users and assigns tasks to node devices for operation. If the packet consists of a coding table, the GVM renews its current runtime behavior. The behavior control of nodes varies with the function code of the Modbus packet sent by the gateway, and the corresponding behaviors are then performed on the basis of the instructions transmitted. In addition, users can use the GVM to adjust LED lighting and environmental brightness. The lighting control gateway receives a coding table sent from servers via Ethernet and transmits it to nodes through the RF module. Node devices receive messages sent from the gateway through the RF module and then send these to the LEDs of actuators in node devices. The present results suggest that the GPP editor can be used for the rapid remote development of application systems and the renewal of node device programs. Developers can easily develop and monitor processes. Unlike traditional intelligent lighting systems, developers can alter lighting system behaviors using the flow described by GRAFCET rather than simply sending variables to change the brightness levels of node lights.

6. Conclusion

In this work, the Modbus-based PTP provides a synchronization scheme to synchronize nodes on a network with precision and efficiency. The Modbus-based PTP codesign integrates industrial fieldbus networks and IEEE 1588 in a WSAN PTS system. The designed WSAN is extremely suitable for real- time synchronization and the requirements of industrial applications such as multiaxis motion or high-speed DI and DO function modules. The GVM application of the real-time kernel simplified the deployment and configuration of WSANs, and the GPP solution rendered embedded device development highly flexible and extensible. Finally, the proposed system achieved satisfactory results in an IoT-

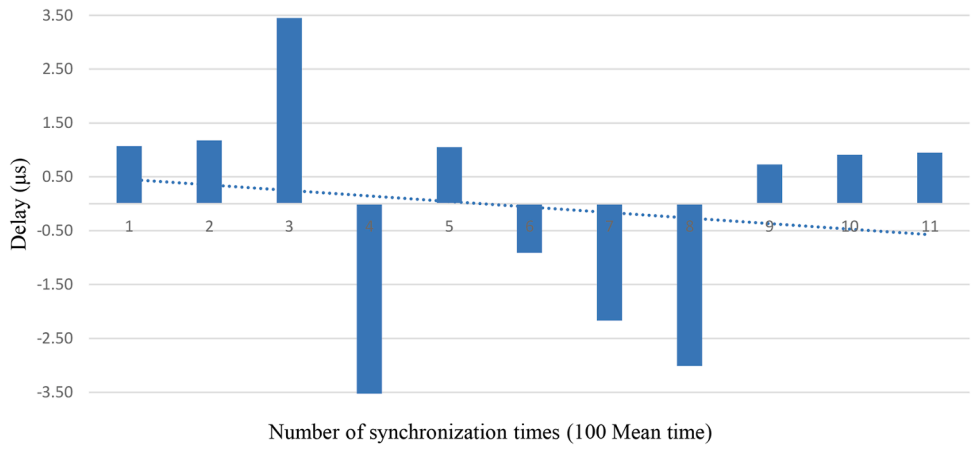


Fig. 14. Distribution diagram of delay measurements.

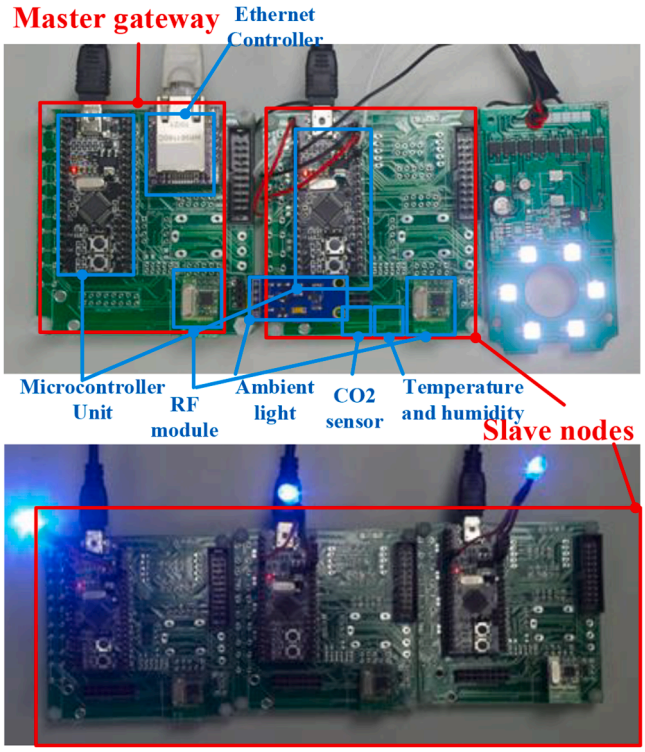


Fig. 15. . Prototype architecture of smart lighting system.

Table 4
Hardware specifications of the IIoT gateway system.

Hardware	Specification
Master MCU(Gateway)	STM32F103C8T6
Slave MCU(Nodes)	STM32F103C8T6
Ambient light sensor	BH1750FVI
Temperature sensor	SHT10
Humidity sensor	SHT10
2.4 G ISM RF module	nRF24L01
Ethernet Controller	ENC28J60
CO_2 sensor	MG811
LED module	SMD LEDs module

based smart lighting system.

In future work, since PWM control signals are used for a wide variety of IIoT applications. The proposed Modbus-based PTP system is no only realizes the precision time synchronization but also achieves the definition of a smart sensor network of IIoT.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that cdocould have appeared to influence the work reported in this paper.

References

[1] Gyorgy Kálmán, Gunner Prytz, "Time synchronization challenges in industrial networks," ABB corporate research, Norway, Mar. 2014. Available: https://www.ia ria.org/conferences2014/filesICDT14/kalman-Time_Synchronization_Challenges. pdf.
[2] Advantech Co., Ltd. : PCIE-1203L http://advdownload.advantech.com/ productfile/PIS/PCIE-1203L/Product%20-%20Datasheet/PCIE-1203L20180910103050.pdf.
[3] C.-.H. Chen, M.-.Y. Lin, C.-.C. Liu, Edge computing gateway of the industrial internet of things using multiple collaborative microcontrollers, IEEE Netw. 32 (1) (2018) 24–32.
[4] J. Cecilio, P. Furtado, Architecture for uniform (re) configuration and processing over embedded sensor and actuator networks, IEEE Trans. Ind. Inf. 10 (1) (Feb. 2014) 53–60.
[5] A. Grilo, J. Chen, M. Diaz, D. Garrido, A. Casaca, An integrated WSAN and SCADA system for monitoring a critical infrastructure, IEEE Trans. Ind. Inf. 10 (3) (Aug. 2014) 1755–1764.
[6] C. Lu, A. Saifullah, B. Li, M. Sha, H. Gonzalez, D. Gunatilaka, C. Wu, L. Nie, Y. Chen, Real-time wireless sensor-actuator networks for industrial cyber-physical systems, Proceedings of the IEEE 104 (5) (May 2016) 1013–1024.
[7] C.-.H. Chen, M.-.Y. Lin & X.-.C. Guo "High-level modeling and synthesis of smart sensor networks for industrial internet of things," J. Comput. Electrical Eng. (CAEE), DOI: 10.1016/j.compeleceng.2017.06.005.
[8] Jeremy Elson, Lewis Girod, Deborah Estrin, Fine-grained network time synchronization using reference broadcasts, SIGOPS Oper. Syst. Rev 36 (2002) 147–163.
[9] S. Ping, Delay Measurement Time Synchronization For Wireless Sensor Networks, 6, Intel Research Berkeley Lab, 2003, pp. 1–10.
[10] M.L. Sichitiu, C. Veerarittiphan, Simple, accurate time synchronization for wireless sensor networks, in: In 2003 IEEE Wireless Communications and Networking, 2003. WCNC 2003 2, IEEE, 2003, pp. 1266–1273.
[11] J. Van Greunen, J. Rabaey, Lightweight time synchronization for sensor networks, in: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications, 2003, September, pp. 11–19.
[12] T. Schmid, Z. Charbiwala, R. Shea, M. Srivastava, Temperature compensated time synchronization, IEEE Embedded Syst. Lett 1 (2009) 37–41.
[13] Christoph LENZEN, Philipp SOMMER, Roger WATTENHOFER, PulseSync: an efficient and scalable clock synchronization protocol, IEEE/ACM Trans. Netw. 23 (3) (2014) 717–727.
[14] Y. Chen, Q. Wang, M. Chang, A. Terzis, Ultra-low power time synchronization using passive radio receivers, in: Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks, IEEE, 2011, April, pp. 235–245.
[15] F. Terraneo, L. Rinaldi, M. Maggio, A.V. Papadopoulos, A. Leva, FLOPSYNC-2: efficient monotonic clock synchronisation, in: 2014 IEEE Real-Time Systems Symposium, IEEE, 2014, December, pp. 11–20.
[16] F. Terraneo, A. Leva, S. Seva, M. Maggio, A.V. Papadopoulos, Reverse flooding: exploiting radio interference for efficient propagation delay compensation in WSN

clock synchronization, in: 2015 IEEE Real-Time Systems Symposium, IEEE, 2015, December, pp. 175–184.
[17] Lim, R., Maag, B., & Thiele, L. (2016, February). Time-of-flight aware time synchronization for wireless embedded systems. In EWSN (pp. 149–158).
[18] M. Maroti, B. Kusy, G. Simon, A. Ledeczi, The flooding time synchronization protocol, in: Proceedings of the 2nd international conference on Embedded networked sensor systems, 2004, pp. 39–49.
[19] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," in IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002), vol., no., pp.1–300, 24 July 2008.
[20] Hsin-Pei Hsieh, M.S. thesis, Department of Computer Science & Information Engineering, National Central University, 2011.
[21] Tzu-Hai Chung, M.S. thesis, Department of Computer Science & Information Engineering, National Central University, 2011.
[22] Ming-Cong Guo, M.S. thesis, Department of Computer Science & Information Engineering, National Central University, 2012.
[23] Ching-An Cho, M.S. thesis, Department of Computer Science & Information Engineering, National Central University, 2013.
[24] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," in IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002), vol., no., pp.1–300, 24 July 2008.
[25] C.-.H. Chen, M.-.Y. Lin, W.-.H. Lin, Designing and implementing a lightweight WSN MAC protocol for smart home networking applications, J Circuit Syst. Comp. 26 (Mar. 2017). DOI: https://doi.org/10.1142/S0218126617500438.
[26] C.-.H. Chen, M.-.Y. Lin, X.-.C. Guo, High-performance fieldbus application-specific integrated circuit design for industrial smart sensor networks, The J. Supercomputing, S.I. (Mar. 2017), https://doi.org/10.1007/s11227-017-2010-1.
[27] Delta Electronics, 2000 Inc.: PCI-L221-B1D0 http://www.deltaww.com/products/CategoryListT1.aspx.
[28] RS Automation.: 2000 MMC-EtherCAT http://rsautomation.biz/product/product_list.php?sca=60.



Ching-Han Chen received the Ph.D. degree in 1995 from the Franche-Comte University, France. He was an Associate Professor in the department of electrical engineering, I-Shou University, Taiwan, R.O.C. from 1995. He is now a Professor in the department of computer science and information engineering. His research interests include embedded system design, multimedia signal processing, robotics.



Ming-Yi Lin received Master degree from Yuan-Ze University and received Ph.D. degree of computer science from National Central University in 2011 and 2016, respectively. He is now an Assistant Professor in the department of computer science and information engineering at Taipei City University of Science and Technology, Taiwan. His research interests include embedded systems, smart sensor networks, and Internet of Things.



Wee-Ping Tew received Master degree in computer science from National Central University in 2014. His-research interests include Modbus embedded systems and the Internet of Things.