Design of WirelessHART Protocol Stack Based on Qp Architecture

Houen Li[1], Yunxiang Zhang [1], Zhuyi Rao [1], Wentong Zhang [2], Yue Yu [2] and Sheng Zhang [2]

1 Shenzhen Power Supply Bureau Co., Ltd., Shenzhen 518000, China;
2 Graduate School at Shenzhen, Tsinghua University, Shenzhen 518055, China
Email: wtzhang1995@163.com

Abstract. WirelessHART protocol is mainly used in the field of industrial process control, and is known for its simplicity, reliability and security. The WirelessHART data service is made up of a series of events. The communication between different state machines is also delivered through events. At the same time, each layer of protocol is composed of several state machines. Therefore, the WirelessHART protocol stack can be designed based on the Qp event-driven architecture. In this paper, WirelessHART software protocol stack is divided into seven active objects, in which each object maintains the corresponding functions. It is implemented active objects through the state machine, which can improve the reuse of behavior and save ROM resources.

1. Introduction

WirelessHART is wireless network communication protocol originated from HART[1]. It adopts a half-duplex communication method and is based on FSK modulation technology. It is mainly used in the field of industrial thread automation control with simple, reliable, and safe features that not only reduce the cost of production, but also improve product quality and production efficiency[2]. At present, there are more than 40 million HART devices in the world serving industrial processes, medical monitoring, equipment testing, etc. [3] The use of WirelessHART protocol to design smart electricity communication network[4], real-time monitoring of electricity consumption, energy conservation and emission reduction, with characteristics of smart grid self-healing, security, interaction, high efficiency, integration[5]. The WirelessHART protocol operates at 2.4 GHz and belongs to the ISM (Industry, Science and Medication) band[6]. It uses DSSS technology which is compatible with the IEEE Std 802.15.4-2006 protocol and is a wireless network topology that sends and receives data units with TDMA (Time Division Multiple Access) and FM (Frequency Modulation) methods[1]. The WirelessHART protocol refers to the open communication standard and conforms to the OSI model. The protocol is mainly divided into the physical layer, data link layer, network layer, transport layer, and application layer[7]. The physical layer protocol of the model is defined by IEEE Std 802.15.4-2006 [8]. The protocols of the data link layer, network layer, transport layer, and application layer of the model are defined by IEC_PAS 62591-2009 [9]. There is still no completely open and complete WirelessHART protocol stack. Scholars all over the world have put forward their own views. J Song et al. [10] constructed a simple prototype based on the protocol. Anna N. Kim et al. [11] focused on the media access layer and network manager to implement the WirelessHART network for specific applications. Ye Liu et al [12] designed and implemented the WirelessHART protocol stack based on TinyOS and CC2430; Jianzhong Ling [13] designed and implemented the WirelessHART network using the IEEE802.15 protocol on the hardware and software platforms of Mega128, CC2420 and

TinyOS; W Liu et al [14] designed the sensor network based on the WirelessHART standard to achieve automatic overcoming obstacle transmission and reduced power consumption.

The WirelessHART data service is composed of a series of primitives, and each primitive can be mapped to an event. Communication between different state machines can be transmitted through events. At the same time, each protocol layer is composed of several state machines. The establishment of the message and the processing of the received data can rely on the flow chart to achieve. Therefore, the WirelessHART protocol stack can be implemented based on an event-driven architecture.

2. Overall architecture

Based on the definition of the functions of each layer in the WirelessHART protocol, a software protocol stack architecture as shown in Fig. 1 is designed. The protocol stack consists of five parts: the physical layer, the data link layer, the network layer, the transport layer, and the application layer. The adjacent two parts are connected through a service interface. The application layer consists of a command builder and a parser; the transport layer consists of a message processor, a transmission channel table, and a schedule; the network layer consists of a message processor, a routing module, and an encryption module; and the data link layer is processed by a state machine and a message. Device, link scheduler, timer, communication table, information library and other modules.

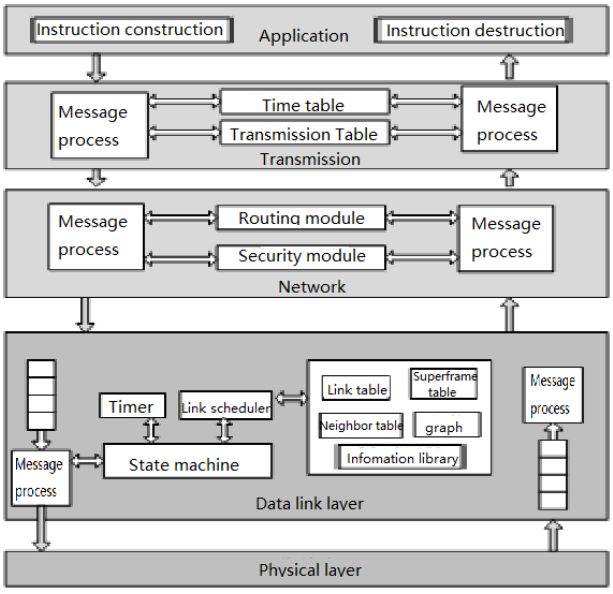

Figure 1 Overall architecture of WirelessHART

3. Design Based on Event-driven Method

The Quantum Platform is a lightweight, open source, portable, state machine-based embedded software development modeling idea. Qp's design is based on a hierarchical state machine that can save developers more time and space. As shown in Figure 2, the Qp platform is mainly composed of a QEP hierarchical event processor, a QF real-time scheduler, a QK preemptive core, and a QS software tracker. QEP supports UML modeling of HSM, allowing state reuse, suppressing unrestricted growth of the state; QF is responsible for the scheduling of events, putting the pointer of the event into the active object's queue, without a complete copy, and the complete event is stored in event pool or static event. The quantum core can support up to 64 active objects, which is more efficient and concise than the traditional preemptive core. The QS tracker group is built in the time scheduler, processor, and kernel, and can record the execution of state functions and the transmission of events, making it convenient for users to view the processing of messages.

Based on the development idea of the Qp platform, we can divide the WirelessHART protocol into several interconnected hierarchical state machines, follow the Qp development style, and deliver the

running rights to the Qp event handler through event-driven. The design ideas of the WirelessHART software stack are: 1) divide active objects, use UML modeling methods, design state machines; 2) allocate event queues for each state machine, determine the type and size of the event pool, define the required events, design Event delivery mechanism; 3) Implement state processing functions, define functions for large-scale state processing functions, implement them in modules, and design flow charts; 4) Strictly encapsulate code, and synchronize code updates with state machines.

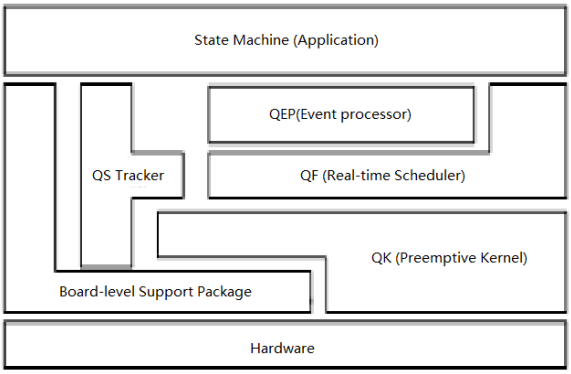

Figure 2 Event-driven System Architecture

4. Design of Protocol Stack

4.1 Division of active objects

Each active object is a state machine, has its own dedicated event queue, and gets a thread of execution. Different active objects are loosely coupled and interact through events. According to the WirelessHART protocol, the protocol divides the protocol into seven active objects: data transmitter, data receiver, MAC scheduler, network layer scheduler, transmission channel, application layer scheduler, and network initiator.

4.1.1 Data Transmitter. The priority of the active object of the data transmitter is 1, which is responsible for evaluating the idle channel, setting the state of the transceiver, setting up the channel, establishing a physical layer protocol data unit, and sending the data through a wireless channel at a designated time. The data transmitter shown in Fig. 3 is based on a UML state machine model.

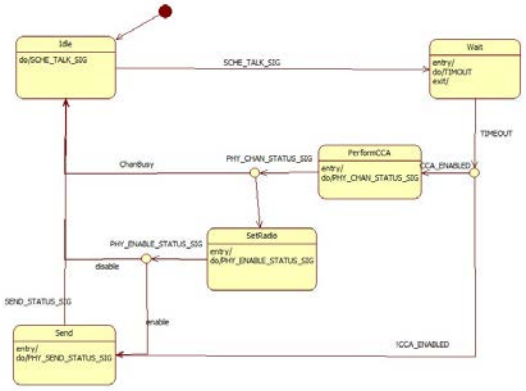

Figure 3 Data Transmitter State Machine

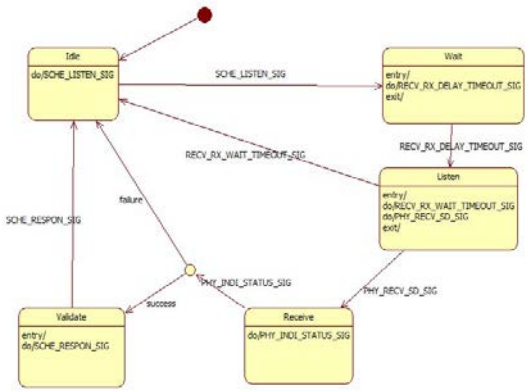

Figure 4 Data Receiver State Machine

4.1.2 Data Receiver. The priority of the active object of the data receiver is set to 2, which is responsible for setting the channel, setting the transceiver state, parsing and verifying the received message, and informing the receiving state and storage address of the message of the MAC scheduler in the form of an event. Fig 4 shows a UML-based state machine for data receiver activity objects.

4.1.3 MAC Scheduler. The priority of the active object of the MAC scheduler is 3, which is mainly responsible for such tasks as link scheduling, maintenance of communication tables and databases, neighbor discovery, and time synchronization. Fig. 5 shows a UML-based state machine for the MAC scheduler activity object.

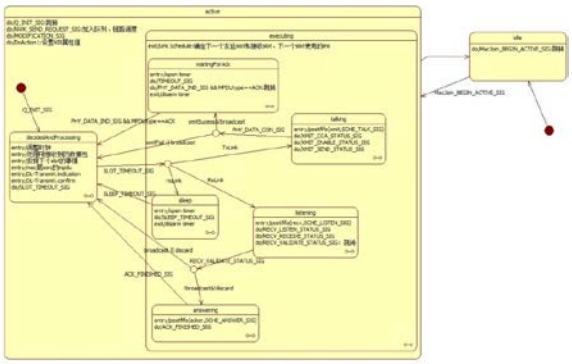

Figure 5 MAC Scheduler

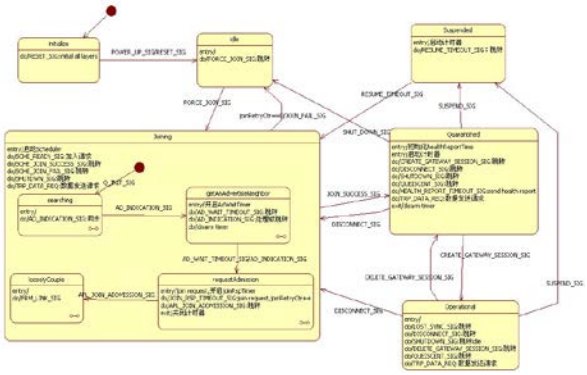

Figure 6 Network Layer Scheduler

When the new WirelessHART time slot arrives, the MAC scheduler is in the IDLE state, inquires the current available link state, and if it is the transmission link, selects the data to be sent, enters the TALKING state, informs the data transmitter through the event, and communicates with the data transmitter. Interaction; if it is a receiving link, it enters the LISTENING state, and informs the data receiver to open the receiving program in the form of an event and updates the neighbor table after receiving the data; if it is a unicast link, the MAC scheduler enters the ANSWERING state and sends an ACK DLPDU. If it is an idle link or a transmission link that does not currently have data transmission, the state machine will enter a sleep state; after the operation of one slot ends, the MAC scheduler returns to the IDLE state to wait for the arrival of the next slot. The MAC scheduler finds new neighbors by discovering the link, periodically keeps in touch with the old neighbors and corrects the clock, and also ensures that the join invitation is periodically issued to invite the new node to join the network.

4.1.4 Network Layer Scheduler. The priority of the active object of the network layer scheduler is 4, responsible for tasks such as joining a network, selecting a route, creating an NPDU, verifying processing of a received packet, maintaining a routing table, maintaining a table, and maintaining a database. Fig. 6 shows a UML-based state machine for the network layer scheduler activity object.

When the system is just started, the node network layer is in the IDLE state. After receiving the command to join the network, it enters the JOINING state and starts the joining mechanism. By default, it enters the SEARCHING substate. By continuously searching for the ADVERTISE data packet, the network parameters are obtained for system upgrade and the time synchronization is completed. The establishment joining command frame is applied for joining the network through the proxy route; after waiting for the network manager to issue the nickname, network key, superframe, link, route, etc., the node enters the isolation state; in the isolation state, the node can work with the network manager. Communication can not communicate with the gateway, but only after the gateway key is obtained is actually added to the network to enter the running state; in the operating state, the node can play all the functions of the network layer. Regardless of the state of the network layer of the node, the handheld device can command the node to enter a pending state, and in the suspended state, the node is in a semi-dormant state.

4.1.5 Transmission Channel. The priority of the active object in the transmission channel is 5, which is responsible for the establishment of the transmission channel, the maintenance of the transmission table and schedule, and the regular reporting of neighbor health status. After the transmission channel receives a data request from the application layer, the TPDU is set up to send a request to the lower layer. After the data is received, the data is decomposed and an indication is given to the upper layer.

Fig. 7 shows a UML-based state machine for the transport channel active object.

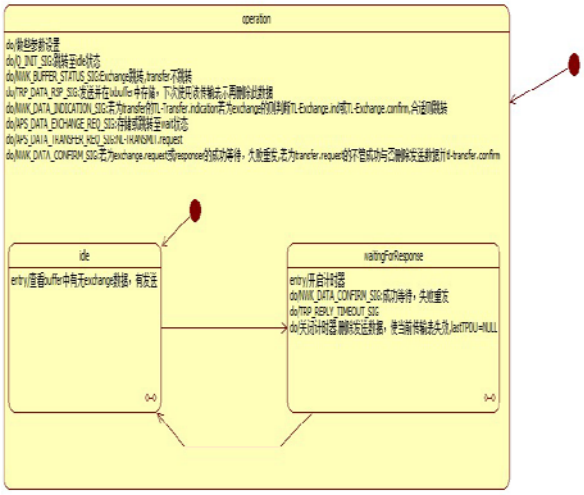

Figure 7 Transmission Channel

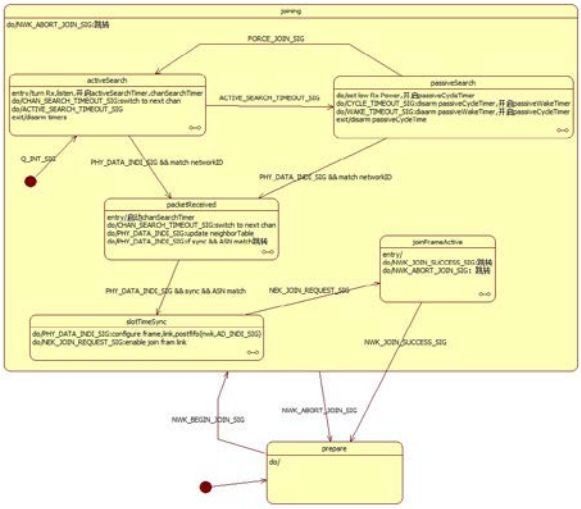

Figure 8 Network Launcher

4.1.6 Application Layer Scheduler. The priority of active objects of the application layer scheduler is 6, which is responsible for the formation and parsing of commands, and realizes the sending and receiving of control data in conjunction with a specific application scenario. The state machine has only one state and no state transition is required.

4.1.7 Network Launcher. The priority of the active object of the network initiator is 7, and the behavior of the data link layer of the control node is added to the network. Fig. 8 shows a UML-based state machine. After the node is powered on, the application layer scheduler is in the IDLE state. After receiving the join command from the network layer scheduler, it enters the network initiator state and starts some network access processes. The node periodically scans the channel in the active search state and enters the data after receiving the data. PACKET_RECEIVED state; continue to scan the channel, parse the packet after receiving the ADVERTISE DLPDU, configure the data superframe and join the link, perform passive time synchronization, complete the ASN synchronization and clock synchronization, and enter the slot enable state; receive the network layer scheduling. After the join request of the layer, the superframe is activated, and the join request message is formed. After the node joins, the IDLE state is entered. When the node needs to rejoin after the node leaves the network, the join process is newly opened.

4.2 Defining events and event pools

The WirelessHART software stack asynchronously communicates between different active objects in the form of event delivery. The definition of the event inherits the QEvent type [15]. Dynamic events can store different parameters such as data pointers and state flags. The static time is directly defined by the QEvent structure and cannot store any parameters. This protocol stack defines two event pools, SMALL_EVENT (small size) and LARGE_EVENT (large size). The size of the small size event pool is within 8 bytes, and the size of the large size event pool is not larger than 16 bytes.

4.3 Event Delivery Mechanism

The protocol stack uses two types of event delivery mechanisms: direct delivery mechanism and publication subscription mechanism. The direct delivery mechanism publishes the event directly through the functions QActive_postFIFO() and QAcive_postLIFO(), and directly publishes this event to the event queue that is the active object of this event consumer. This is a one-to-one delivery mechanism, suitable for Unicast events; the publishing subscription mechanism publishes events to the QF framework through the function QF_publish(). Multiple active objects subscribe to this event through the function QActive_subscribe(). This is a one-to-many delivery mechanism suitable for

broadcast events.

4.4 Define State Processing Functions

The input to the state handler is a pointer to the active object and a pointer to the event, and the return value is the execution state. The switch-case branch structure is used inside the function, and corresponding processing is performed according to different types of events. The return value of the branch is already executed or it is turned to another state processing function. It should be noted that state transitions cannot be performed at the entry and exit signals of the state, that is, the return value of the function can only be executed.

5. Test of software protocol stack

After verifying the logic function of the protocol stack through the software simulation platform, in order to actually test the networking time and network power consumption, the MSP430 and CC1100E are used to actually test the protocol stack.

5.1 Test Platform

The CPU adopts TI's MSP430, the ROM size is 64KB, the RAM size is 6KB, the system clock is 16MHz, and the crystal frequency used during sleep is 32.768KHz. The current consumption when the CPU is in sleep is only 3uA. The wireless transceiver uses TI's CC1100E. CC1100E band is 470-510MHz, supports 250Kbps data transmission rate, supports channel evaluation, supports programmable transmit power, and consumes only 300nA during sleep.

5.2 Test results

The actual test of the mesh topology formed by the seven nodes and the network manager. View the logs through the serial port, analyze the data, and verify the overall functionality of the protocol stack. The logical functions of the protocol stack meet the requirements of the protocol. After the path is blocked, the node can automatically select the backup path for data transmission. After the node leaves the network, it can rejoin the network. The average re-join time is 3.8s. The average network update time is 6.8s. The data retransmission mechanism works normally and the packet loss rate is high. In the case of 93.5%, the transmission reliability of the data packet is 99.98%. Through long-term observation and calculation, the power consumption of the system can be represented by Fig. 9. After adopting the dormant scheduling mechanism, the system can reduce the average working time, thereby greatly saving energy and improving the life cycle of the network. The average power consumption of a normal node is 1.1 mA, and the power consumption of the center node is 1.3 mA.

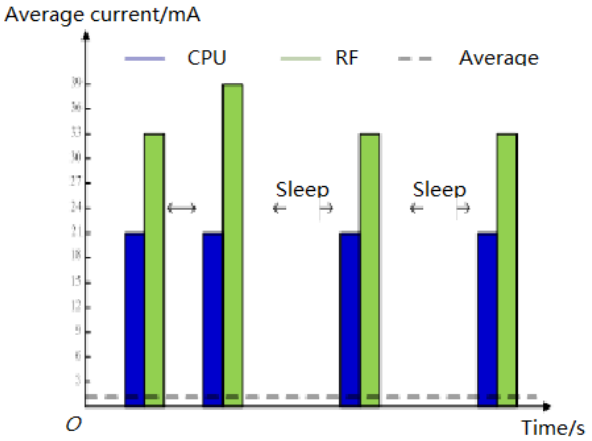

Figure 9 Sleep-based scheduling system

6. Conclusion

In this paper, based on WirelessHART protocol, we designs the overall architecture of the WirelessHART software protocol stack based on the design philosophy of the Qp event-driven

architecture, and provides functional definitions and interface definitions for each module of the protocol stack. The WirelessHART software protocol stack is divided into 7 active objects, each object maintains corresponding functions, and active objects are implemented through a hierarchical state machine to improve behavior reuse, save ROM resources, and reduce power consumption.

7. References

[1] HCF-HART Communication Foundation, HART7 Specification [Z], 2007:4-13.
[2] David Guatafsson. WirelessHART-Implementation and evaluation on wireless sensors [J]. KTH Electrical Engineering, 2009: 13-48.
[3] Kim A N, Hekland F, Petersen S, et al. When HART goes wireless: Understanding and implementing the WirelessHART standard[C]// IEEE International Conference on Emerging Technologies and Factory Automation. IEEE, 2014:899-907.
[4] Zhang Wen-liang, Liu Zhuang-zhi, Wang Ming-jun, et al. Research Status and Development Trend of Smart Grid[J]. Power System Technology, 2009, 33(13):1-11.
[5] Yang De-chang, Li Yong, Liu Ze-hong, et al. Study on the Structure and the Development Planning of Smart Grid in China[J]. Power System Technology, 2009, 33(20):13-20.
[6] Aaro Lehto. WirelessHART smart wireless solutions. HART Communication Foundation, 9390 Research Blvd, USA, 2009:2-39.
[7] Tomas Lennvall. A comparison of WirelessHART and ZigBee for industrial applications. KTH Electrical Engineering, 2009.
[8] IEEE Std 802.15.4-2006. Wireless Medium Access Control and Physical Layer Specifications for Low-Rate Wireless Personal Area Networks. Institute of electrical and electronics engineers, USA, 2006.
[9] Piscataway N. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications[J]. IEEE D3, 2012:C1-1184.
[10] Song J, Han S, Mok A K, et al. WirelessHART: Applying Wireless Technology in Real-Time Industrial Process Control[J]. IEEE Real Time & Embedded Technology & Applications Symposium, 2008:377-386.
[11] Kim A N, Hekland F, Petersen S, et al. When HART goes wireless: Understanding and implementing the WirelessHART standard[C]// IEEE International Conference on Emerging Technologies and Factory Automation. IEEE, 2014:899-907.
[12] Liu Yu. Design and Implementation of WirelessHART Protocol Stack Based on TinyOS and CC2430[D]. University of Electronic Science and Technology of China, 2011.
[13] Ling Jian-zhong. Design and Implementation of WirelessHART Protocol Stack[D]. University of Electronic Science and Technology of China, 2013.
[14] Liu W, Sheng X U, Zhang X, et al. Design and Implementation of Wireless Sensor Network System Based on WirelessHART[J]. Chinese Journal of Electron Devices, 2017.
[15] MIRO SAMEK PH.D Quantum Platform Programmer's Mannual[EB/OL].http://www.quantum-leeps.com/doc/QP_mannual.pdf.