A Hierarchical Model of Control Logic for Simplifying Complex Networks Protocol Design

Yi Yang, Wei Quan, Jinli Yan, Lu Tang, Zhigang Sun

A hierarchical model of control logic for simplifying complex networks protocol design

Yi Yang[1] , Wei Quan[1(✉)], Jinli Yan[1], Lu Tang[2], Zhigang Sun[1]

[1] Computer College, National University of Defense Technology
Changsha 410000, Hunan, China
{yangyi14,w.quan}@nudt.edu.cn
[2] HuNan Hua Xin Tong Networks, Changsha 410000, Hunan, China

Abstract. With the increase of network protocols complexity, the finite state machine model commonly used in hardware design is difficult to directly describe and manage complex protocol control logic. The hierarchical approach can simplify the design and implementation of complex logic. However, the control logic of the protocol is a whole, and how to divide the control logic of the protocol hierarchically is a problem that needs to be solved urgently. Therefore, by analyzing the characteristics of complex protocol control logic, this paper proposes the DoubleDeck model. This model divides the state in protocol processing into a global state perceivable by the protocol peer and a local state invisible to the outside. Next, we established a prototype system of the time synchronization protocol (AS6802) on the FPGA array based on the DoubleDeck model, which effectively verified the feasibility of the model.

Keywords: Finite state machine · Control logic · Network protocol · Hierarchical design

1 Introduction

With the development of communication technology [1, 2], the complexity of the protocol has continued to increase. How to implement complex protocol control has become an important issue facing hardware implementation protocols [3]. The complexity of protocol control is mainly reflected in two aspects. One is that while the protocol entity exchanges constantly changing status information with the peer through packets, it also needs to set various types of timers to infer the network status and the behavior of the protocol peer; The second is that the protocol not only needs to monitor asynchronous trigger events, but also needs to perform various synchronous and asynchronous processing operations. In the face of complex protocol control, the finite state machine model used in traditional hardware design is not only poorly readable, but also difficult to directly describe and manage complex control behaviors [4, 5].

In order to simplify the design and implementation of complex control logic, some recent studies have proposed implementation methods based on hierarchical state machines [6]. However, in the existing protocol design schemes, the

control logic of the protocol is a whole, and how to set a common division standard for complex protocols and achieve feasible hierarchical division of control logic is still an urgent problem to be solved [7].

In response to the above-mentioned challenges, this paper proposes a DoubleDeck model that supports the hardware implementation of complex protocol control logic to simplify the complexity of protocol control logic design. The basic idea of DoubleDeck is to divide the state in the protocol processing process into a global state that is external to the protocol entity and a local state that is not external. Among them, the global state and the corresponding conversion logic constitute the top-level state machine to maintain the changes in the protocol processing stage. At the same time, the local state associated with the top state and the corresponding transition logic form a series of bottom state machines to control the detailed processing details of the processing stage.

2 Motivation and Related Works

2.1 Motivation

In different agents of the same protocol, the control logic executes different control state machines that are related to each other. In the process of protocol processing, the protocol entity must obtain the network status and behavior of the opposite end to correctly complete the protocol-related processing. There are two main methods for the protocol entity to obtain the network information of the opposite end. First of all, in view of the characteristics of the packet type that can reflect the processing state of the protocol, the protocol entity exchanges state information with the peer through multiple protocol-related packets. Second, the protocol entity sets up a series of timers to infer the status information of the peer.

Through the analysis of the agreement, we found that in the process of implementation of the agreement, the agreement entities showed a diverse state. Among them, some states are necessary information for the protocol peer to process events (state transitions, etc.), so the protocol peer needs to perceive such states; some states can be set to an externally invisible state because they involve the details of the event processing of the protocol peer. Regarding whether the protocol state can be perceived by the outside world, we divide the protocol state into a global state perceivable by the protocol peer and a local state that is not visible from the outside. Therefore, we can decouple the protocol control logic into global state-related conversion logic and local state-related processing logic. Therefore, in this article, we propose a two-layer state machine design model to simplify the design and implementation of complex protocol control logic by dividing complex protocol control logic into small conversion logic and processing logic.

2.2 Related Works

The finite state machine (FSM) model is widely used in hardware logic design [4]. However, when there are many protocol states and complex state transitions, the

flat finite state machine model will cause problems such as poor model readability and difficulty in later debugging. Hierarchical Finite State Machine (HFSM) [6] adds elements such as state variables and state transitions on the basis of FSM to further express the dynamic behavior of the system in a fine-grained manner. HFSM uses a top-down approach to describe the system. In this method, the hardware logic is divided into multiple smaller sub-modules, which can reduce the complexity of the overall hardware design. However, the sub-modules are highly independent and need to be designed and tested separately, which increases the difficulty of later debugging to a certain extent.

3 DoubleDeck Model Overview

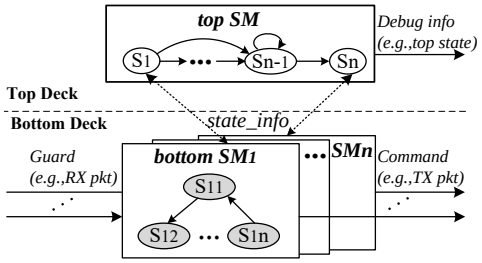

Fig. 1. The overview of DoubleDeck model.

DoubleDeck consists of top deck and bottom deck, as shown in Fig.1. There is a top state machine on the Top deck, which is responsible for maintaining the global state of the protocol and its corresponding transition logic. The top state machine controls the conversion of the global state and sends the state information to the CPU and other external devices to facilitate later debugging and maintenance. In addition, the top state machine controls the bottom state machine to jump from the waiting state to the active state through the *state_info* signal. At the same time, the state transition of the top state depends on the feedback information of the bottom state machine. The bottom state machine that completes the processing task transfers the next stage of the protocol processing (global state) to the top state machine through the *state_info* signal. Bottom deck consists of a series of bottom state machines. The bottom state machine implements the processing logic in the global state and provides control information for the state transition of the global state. The bottom state machine array processes external trigger events based on flow rules. The bottom state machine has three states, which are waiting state, active state and end state. Among them, the active state has multiple processing states, that is, the partial state mentioned above. When the device receives protocol-related traffic, the external receiving module sends the protocol-related information to the bottom state machine array. The bottom state machine in the active state receives

the information and enters the appropriate local state according to the protocol settings to generate command events. When the command event contains global state jump information, the bottom state machine enters the end state and uses the *state_info* signal to notify the top state machine to perform state transition.

4 Experiment

The AS6802 protocol describes a fault-tolerant high-precision time synchronization algorithm [8]. SM and CM are two protocol roles of AS6802. We implemented the IP core of the AS6802 protocol based on the DoubleDeck model, and implemented the prototype system of the AS6802 protocol on a network processing platform composed of multiple Arria10 SoCs (Altera FPGAs). We instantiate the IP core as three SMs and one CM, as shown in Fig.2. In addition, the oscilloscope in Fig.2 is used to detect the synchronization pulse signal of each node and calculate the error between the synchronization pulses in real time. The controller forms an out-of-band configuration network with all nodes through switches to realize the configuration and status monitoring of each node.

In the first experiment, we used a CM to exchange data with multiple SMs to achieve time synchronization between nodes. As shown in Fig.3, we tested the stay time of the node in the intermediate state during the synchronization establishment process through three sub-experiments. Experimental results show that with the increase in the number of nodes, the test network can quickly synchronize the time of the entire network within 465 microseconds. In addition, the state transition conditions of the nodes are consistent with the description in the protocol.

In the second experiment, we evaluated the performance of the AS6802 IP core using 3 SMs and 1 CM. After all nodes enter the SYNC state, we obtain the maximum clock deviation of the four nodes through an oscilloscope, that is, the synchronization accuracy of the test network. We collected 50 sets of data, as shown in Fig.4. Analyzing the experimental results, it can be found that the synchronization accuracy of the four nodes can be controlled within 25ns, which fully meets the requirements of high-precision time synchronization.

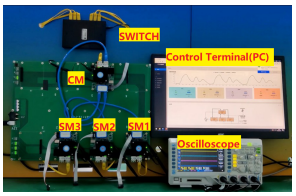

Fig. 2. An experiment environment with 4 nodes(3 SM and 1 CM).

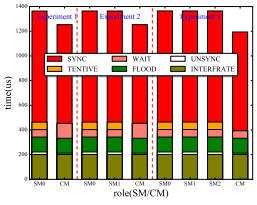

Fig. 3. The time used by the nodes during the startup phase.

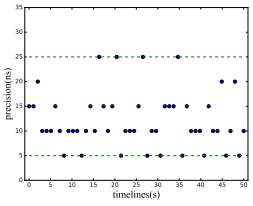

Fig. 4. Synchronization precision of the four-node network.

5 Conclusion

The traditional finite state machine model is difficult to solve the complexity problem of hardware implementation of complex protocol control logic. Although the hierarchical method has become an important means to simplify the design of complex logic, the existing work fails to provide developers with a standard for dividing complex protocol control logic. This paper decouples the control logic of the protocol into conversion logic and processing logic, and maps it to the double-layer state machine of the DoubleDeck model, which effectively simplifies the design of complex protocol control logic.

Acknowledgments. This work is supported by the Defense Industrial Technology Development Program (Grant NO.WDZC20205500110). We also thanks HuNan Hua Xin Tong Networks for their equipments to carry out our experiments.

References

1. Kotulski, Z., Nowak, T.W., Sepczuk, M., Tunia, M.A.: 5g networks: Types of isolation and their parameters in ran and cn slices. Computer Networks 171, 107135 (2020)
2. Nasrallah, A., Thyagaturu, A.S., Alharbi, Z., Wang, C., Shao, X., Reisslein, M., ElBakoury, H.: Ultra-low latency (ull) networks: The ieee tsn and ietf detnet standards and related 5g ull research. IEEE Communications Surveys & Tutorials 21(1), 88–145 (2018)
3. Steinhammer, K., Ademaj, A.: Hardware implementation of the time-triggered ethernet controller. In: Embedded System Design: Topics, Techniques and Trends, pp. 325–338. Springer (2007)
4. Qi, Y., Zhong, J., Jiang, R., Jia, Y., Li, A., Huang, L., Han, W.: Fsm-based cyber security status analysis method. In: 2019 IEEE Fourth International Conference on Data Science in Cyberspace (DSC). pp. 510–515. IEEE (2019)
5. Moshref, M., Bhargava, A., Gupta, A., Yu, M., Govindan, R.: Flow-level state transition as a new switch primitive for sdn. In: Proceedings of the third workshop on Hot topics in software defined networking. pp. 61–66 (2014)
6. Oliveira, A., Melo, A., Sklyarov, V.: Specification, implementation and testing of hfsms in dynamically reconfigurable fpgas. In: International Workshop on Field Programmable Logic and Applications. pp. 313–322. Springer (1999)
7. Fragal, V.H., Simao, A., Mousavi, M.R.: Hierarchical featured state machines. Science of Computer Programming 171, 67–88 (2019) and Applications. pp. 313–322. Springer (1999)
8. AS6802, S.: Time-triggered ethernet. SAE International (2011)