A Flowchart Based Finite State Machine Design and Implementation Method for FPGA

Zhongjiang Yan(✉), Hangchao Jiang, Bo Li, and Mao Yang

School of Information and Electronics, Northwestern Polytechnical University, Xi'an 710072, China
{zhjyan,libo.npu,yangmao}@nwpu.edu.cn

Abstract. The design idea of control and data separation is an effective means to realize the complex communication system, and the control part can usually be designed and realized by means of finite state machine (FSM). However, there is no effective method to realize the complex communication system based on finite state machine in the existing research. Aiming at the problem of the existing FPGA design and implementation methods with complex and non-universal communication protocol and algorithm design, a Flowchart based Finite State Machine (F-FSM) design and implementation method for FPGA is proposed, which significantly improves the FPGA development efficiency. This method takes the flowchart describing the complex communication system as input, divides the communication system into modules, and outputs the finite state machine transition diagram and transition matrix of the control module. This method can effectively shorten the design time of the communication system and its control module. Finally, an IP core encapsulated in FPGA is designed. This method can effectively improve the development efficiency of control module, improve the re-usability of control module and reduce the workload of code development.

Keywords: Flowchart · Finite state machine · FPGA

1 Introduction

Communication systems are important parts of the Internet of Things (IoT). For example, wireless local area network (WLAN) can be used as a low-energy means of IoT communication [1]. With the development of communication technology, the complexity of communication networking protocols in communication systems (such as WLAN, 4G/5G/6G, Bluetooth, ZigBee, etc.) are increasingly high, and how to effectively realize these complex communication systems through Field Programmable Gate Array (FPGA) has become an urgent problem to be solved [2–4]. There are two mainstream FPGA design and implementation methods in existing research, i.e., pipeline design idea [5] and control and data separation idea [6]. The idea of pipeline design is only applicable to communication systems with simple logic and clear functions between modules, while the idea of control and data separation is applicable to complex communication systems, in which the control module can be realized by using finite state machine [7,8], and the finite state machine (FSM) describes the overall working condition of the system.

Combined with the actual system design, the general modular system architecture is shown in Fig. 1. This architecture divides the whole system into several related or independent functional modules, and manages the signal interaction and sequence of each functional module by setting the system main module. In the same way, each function module can be divided into sub-modules, and a functional main module can be used to schedule the functions of the overall function module. This method can also be used if the internal structure of subsequent sub-modules is more complex.

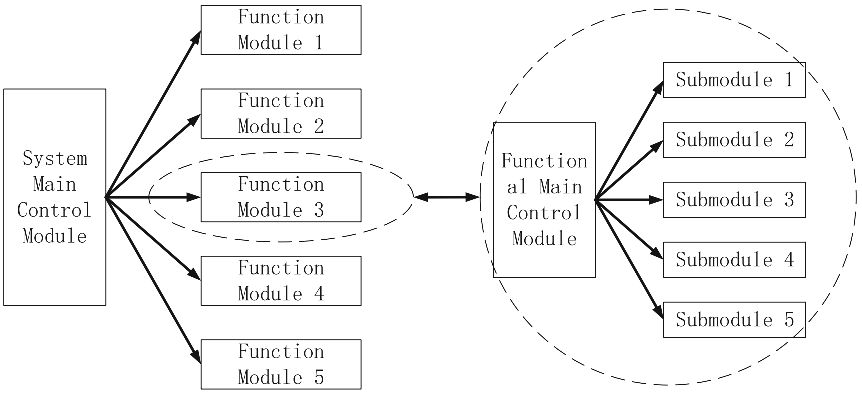


Fig. 1. General modular system architecture

Modular system architecture is very common in practical engineering systems, and its advantages are quite obvious. Ref. [9] applies the modular architecture to design the controller of the machine tool reconstruction. The control module of the top level is responsible for controlling the physical module of the bottom level, and the physical module processes the data. Control is separated from data processing, and this hierarchical control pattern reduces a large amount of overall revalidation. In Ref. [10], IEEE 802.11 multiple access control (MAC) protocol implementation based on FPGA technology divides the entire protocol system into sending and receiving parts, and then realizes the two parts by controlling each function module through main control. Reference [11] applies this architecture to the automatic bag-bagging control strategy based on finite state machine, and verifies the advance of this design method through practical application. In the previously mentioned references on the application of FSM in various fields, the idea of modular system architecture has been used subtly. This paper systematically summarizes this architecture and provides a general design and implementation method for developing a system based on FSM in FPGA.

To reduce the unforeseen problems caused by the redundant circuits, Wen proposed a design method of highly efficient FSM and validated it by the circuit diagrams in Ref. [12]. Chen presented an efficient manner described in Verilog HDL in the design of FSM in Ref. [13], and verifies with a synthesizable example the advantages of the design method of FSM in the area and power consumption. Ref. [14] studied different state encoding styles and Verilog descriptions of FSM. Reference [15,16] found out the advantages and disadvantages on circuit, simulation and stability of different FSMs, which showed that the two-always and three-always method are better than the one-always method. It can be seen that the aforementioned related works study how to optimize and coding the FSM based on Verilog HDL. However, few of existing related works study how to design an efficient FSM for a complex communication system. This motivates us to study this topic.

Although there is no general design method of FSM in the existing research, flowchart is a general method to represent the working mechanism of communication protocol or algorithm. Transforming flowchart into FSM is an idea that has never appeared and is worth studying. Based on this observation, in this paper we propose a flowchart-based FSM (F-FSM) design and implement method for FPGA. The main contributions are listed as follows. The communication system is divided into modules and the state transition graph and matrix of the control module are given. This method can effectively shorten the design time of the communication system and its control module. Finally, an IP core encapsulated in FPGA is designed. This method can effectively improve the development efficiency of control module, improve the re-usability of control module and reduce the workload of code development.

The rest of this paper is organized as follows. Section 2 presents the design principle and modelling methods of FSM, as the theoretical foundations of our proposed method. Section 3 presents the proposed F-FSM method. Section 4 gives the IP core design and implementation of F-FSM method. Section 5 verify the correctness and the efficiency of the proposed F-FSM IP core through functional simulation and mapping. Section 6 concludes this paper.

2 Design Principle and Modelling Methods of FSM

The design principle and modelling methods of FSM are presented in this section, which layout the theoretical foundations of our proposed method. Section 2.1 presents the design principle of FSM and Sect. 2.2 presents three modelling and description methods of FSM, which are state transition diagram, state transition matrix and state transition table.

2.1 Design Principle of FSM

As many theories emerge with the needs of science and engineering, FSM is a theory that provides an approximation for physical and abstract phenomena [7].

By using FSM to describe abstract systems, the internal logic of things can be clearly and completely presented to the analyst or designer. Because FSM is easy to build, it is widely used in life, mathematics, engineering and other fields, for example, early Turing machines, grammatical analysis of English sentences, analysis of three states of water. The system which can be described by the design method of FSM has the following characteristics, which are also possessed by the multi-access protocol in the communication system.

1. The overall logic of the system can be divided into a finite number of states with a definite starting state.
2. At any time instant, the system is only in the unique divided state.
3. The triggering conditions for each state and action are clear.
4. Triggering conditions of the system's actions only depend on the current state and current triggering events.

The mechanism of FSM is that the system is divided into finite states according to certain principles to realize the complete functional actions of the original system. The transition is driven by the occurrence of internal or external events of the system. Extended State Machine (EFSM) is a more perfect and universal form of FSM. Due to its completeness of description, EFSM is often used in the modelling of complex communication systems, such as communication protocols and resource allocation algorithms. Its definition is a six-tuple, as shown in Eq. (1).

$$M = \{S, S_0, I, V, O, T\} \tag{1}$$

S refers to the state set of the state machine, i.e., M, and S_0 is the initial state of the state machine. I is the input set of M, and O is the output set. V is the internal variable set, and T is the state transfer condition set of the state machine. Since the state transition condition contains information such as transition direction and trigger condition, T can be represented in the form of six tuples, as shown in Eq. (2).

$$T = \{S_i, S_j, i, o, P_t, A_t\} \tag{2}$$

S_i is the initial state of M. When the input i and the state transition condition P_t of the current variable value are valid, M is triggered to jump to the end state S_j of the corresponding state transition, followed by a series of operations A_t such as output or assignment, and the output result is o.

2.2 Modelling and Description Methods of FSM

Three modelling methods of FSM are presented in this sub-section, which are state transition diagram, state transition matrix and state transition table.

State Transition Diagram. The state transition diagram of an abstract FSM can be represented by circles representing the finite states and state transition directions (directed arrow) between the states. Figure 2 is a state transition diagram of a three-state FSM with initial state $S1$. This description method is visual and intuitive, which is the most commonly used expression at present. However, with the increase of the number of states and transition conditions, the depicted state transition diagram will be relatively chaotic and no longer applicable.

State Transition Table. In short, a state transition table describes state transitions in a FSM in the form of a table. Table 1 is a state transition table of the FSM state transition diagram shown in Fig. 2. The first row represents all transition conditions between states, and the first column represents all of the finite states of the system. The element at the intersection of a row and a column represents the transition condition from the state indicated by the row to the state indicated by the column. Similar to the disadvantage of the state transition diagram, when the number of states increases, the condition of inter-state transition will increase by a square time, and the number of columns in the state transition table will also increase sharply. Therefore, this description method will no longer be applicable.

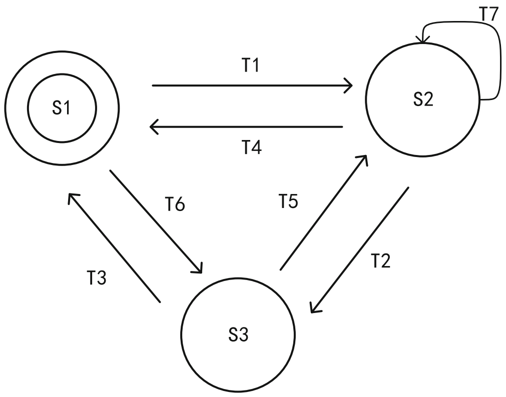


Fig. 2. An example of state transition diagram

Table 1. State transition table

	T_1	T_2	T_3	T_4	T_5	T_6
$S1$	$S2$					$S3$
$S2$		$S3$		$S1$		
$S3$			$S1$		$S2$	

State Transition Matrix. In order to address the shortcomings of the above two methods, the concept of matrix in mathematics is introduced into FSM. Equation (3) shows the state transition matrix of the FSM as shown in Fig. 2. In this matrix, each row and each column represent all states of the system. For each entry of the matrix, the row represents the current working state, and the column represents the state to be transited in the next state transition. The intersection of the row and column represents the trigger condition of the state transition. However, it is not as intuitive as a state transition diagram.

$$\begin{array}{c c} & \begin{array}{ccc} X_1 & X_2 & X_3 \end{array} \\ \begin{array}{c} X_1 \\ X_2 \\ X_3 \end{array} & \begin{bmatrix} & T_1 & T_6 \\ T_4 & & T_2 \\ T_3 & T_5 & \end{bmatrix} \end{array} \tag{3}$$

Above three FSM modelling and description methods are the most commonly used methods in applications, and each has its own advantages and disadvantages. In addition, the combination of the above three methods is also common in FSM modelling and description. In this paper, the FSM is modelled and described mainly by state transition matrix and state transition diagram.

3 Flowchart Based Finite State Machine Designing Method

3.1 Basic Ideas

On the one hand, the modular approach of control and data separation, as shown in Fig. 1, is one that involves the step-by-step splitting of system functions into a number of single functional modules. Among them, the master control module is responsible for implementing the work sequence of these functional modules. On the other hand, flowchart is one of the most common and complete whole project expression method. Therefore, if the flowchart of the judgment conditions can be regarded as state transitions of FSM, and concrete operation can be regarded as the state of FSM, and then connect them with the start and end to form a closed loop, the flowchart can be seen as an FSM state transition diagram. This is the basic idea of the proposed F-FSM. Therefore, the overall engineering design and implementation of FPGA can be carried out based on flowchart.

Figure 3 is the flowchart of the main steps of the proposed F-FSM method. Wherein, the solid frame represents the design step of the method, and the dashed frame represents the finite state machine information obtained by the corresponding step, such as the states and the transition conditions between states. The main steps of the proposed F-FSM is given as follows.

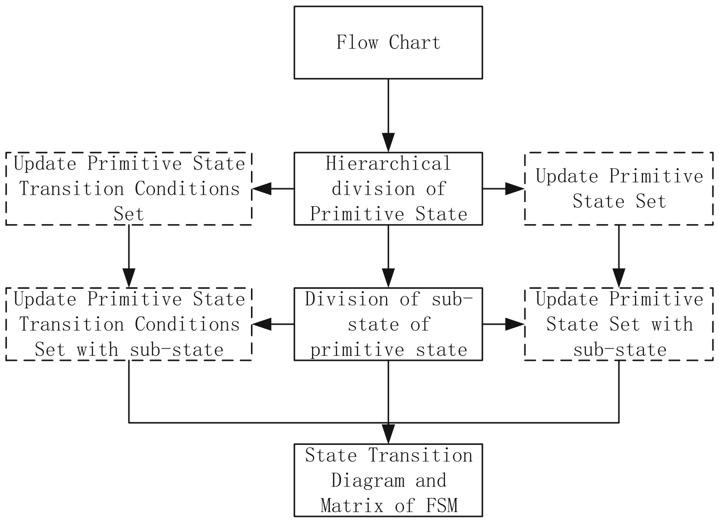

Flow Chart
Update Primitive State Transition Conditions Set
Hierarchical division of Primitive State
Update Primitive State Set
Update Primitive State Transition Conditions Set with sub-state
Division of sub-state of primitive state
Update Primitive State Set with sub-state
State Transition Diagram and Matrix of FSM

Fig. 3. Main designing flow of F-FSM

1. The flowchart is taken as input.
2. According to the flowchart, the set of primitive state and the set of transfer conditions between the primitive states can be obtained through flowchart division procedures.
3. For each primitive state, the sub-states, each with a single function, can be obtained through flowchart division procedures. And then, the finite state set and the state transition conditions set can be updated by including these sub-states.
4. Run above step 3 iteratively until all of the primitive states are divided into sub-states, which can not be divided any more.
5. Finally, the final state transition diagram of FSM can be drawn, and the state transition matrix can be derived.

The proposed F-FSM provides a modular architecture application method for FPGA system design and implementation, based on the traditional flowchart designing method. Therefore, the proposed F-FSM method is a kind of method for complex logic system based on simple traditional flowchart method. It can greatly reduce the design difficulty, and promote the overall design of the structure of the realizability and maintainability.

3.2 Detailed Description of F-FSM

According to the flowchart of the main steps of the proposed F-FSM method, as shown in Fig. 3, the detailed steps are as follows.

Step 1: According to the top-down principle, partition the complex logic into several functional modules. Networking protocols in communication systems are often a complex set of processes, which can be directly divided into several modules according to their functions. For example, the DTRA protocol can be divided into scanning stage, reservation stage and data sending stage in DTRA. The CSMA/CA protocol can be divided into the sending module and receiving module. The purpose of this step is, not only to reduce the complexity of the single flowchart in Step 2, but also to make the overall structure more intuitive.

Step 2: Comb and draw the flowchart of each module. According to the partitioned functional modules in Step 1, draw the flowchart required by this method after understanding the internal principles of each module. The requirements of the proposed F-FSM method for flowchart are as follows. In the required flowchart, only the following frames or box can be accepted.

- START box and END box.
- A decision box with n, $n \geq 1$, inputs and 2 outputs, known as *decision branch.*
- A multi-process box with m, $m \geq 2$, inputs and 1 output, known as *process branch.*
- A single-process box with one input and one output, known as *single process.*

Up to now, the complex logic system can be modelled and described with the traditional flowchart. The purpose of this step is to normalize the flowchart, with clear transition conditions and complete separation of states and conditions. In addition, this step is also to facilitate the implementation of if-else decision logic based on FPGA hardware language. Figure 4 is an example of a flowchart that conforms to this specification. To facilitate further understanding of this approach, the following steps will be described using this flowchart as an example. This flowchart does not correspond to any actual situation, so there is no specific process operation. However, it will not affect the subsequent operation.

Step 3: According to the obtained flowchart, partition the flowchart into primitive states, and then obtain the set of primitive states and the state transition conditions set. After the primitive states are determined, they are removed from the flowchart so that they do not affect subsequent state partitioning procedures. After the primitive states connected by the decision branch are determined, the transition conditions of these states are also determined. That is, the condition of the establishment of the decision branch is the transition condition of these two states. The detailed flow of this step is as follows.

Step 3.1: Take the boxes of "START" and "END" in the flowchart as the states of $S0$ and $S1$, respectively. If the function of the flowchart is a loop, $S0$ and $S1$ can be merged into state $IDLE$ to form a closed loop. Then go to Step 3.2.

Step 3.2: Find all of the "process branches" in the flowchart, and cut the flowchart with all inputs as cutting points. And then go to Step 3.3.

Step 3.3: After cutting, if the process branch contains "single process", then regard "single process" as a primitive state. It is numbered according to the hierarchy and sequence and counted it into the state set S. If the process branch contains decision branch, then the decision branch can be regarded as the state transition condition. After the states connected to "decision branch" is determined, in terms of the previous states, the post-states and state transition conditions, then the state transition conditions are included in the inter-state transition condition set T. Then go to Step 3.4.

Step 3.4: If there is still a decision branch that is not separated from the flowchart, then look for the first decision branch from "START" following the arrow. And then regard all the inputs and outputs of the decision branch as cutting points. Then go back to Step 3.3. Otherwise, when there are no decision branch, the flowchart partition procedure ends.

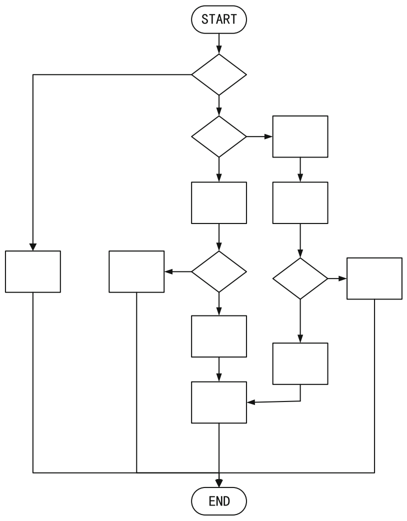
The flowchart, as shown in Fig. 4(a), after the primitive states partition is given in Fig. 4(b) according to Steps 3.2 and 3.3. And the flowchart after the partition of primitive state 4 according to Steps 3.3 and 3.4 is shown in Fig. 4(c).

Finally, the state set S and the inter-state transition condition set T are respectively shown in Eq. (4) and (5), where the transition condition is fictitious and is represented by capital letters, i.e., A, B, C and etc.

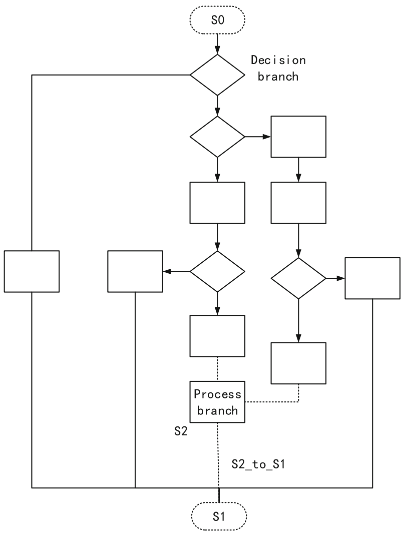
$$\begin{aligned} S &= \{S0, S1, S2, S3, S4\} \\ &= \{S0, S1, S2, S3, \{S4-1, S4-2\}\} \\ &= \left\{S0, S1, S2, S3, \left\{ \begin{array}{l} \{S4-1-1, S4-1-2, S4-1-3\}, \\ \{S4-2-1, S4-2-2, S4-2-3\} \end{array} \right\} \right\} \end{aligned} \tag{4}$$

$$\begin{aligned} T &= \{(S2, S1, A), (S0, S3, B), (S3, S0, C), (S0, S4, \overline{C})\} \\ &= \{(S2, S1, A), (S0, S3, B), (S3, S0, C), (S0, S4-1, \overline{C}+D), (S0, S4-2, \overline{C}+\overline{D})\} \\ &= \{(S2, S1, A), (S0, S3, B), (S3, S0, C), (S0, S4-1-1, \overline{C}+D), \\ &(S4-1-1, S4-1-2, E), (S4-1-1, S4-1-3, \overline{E}), (S4-1-2, S1, F), \\ &(S4-1-3, S2, G), (S0, S4-2-1, \overline{C}+\overline{D}), (S4-2-1, S4-2-2, H), \\ &(S4-2-1, S4-2-3, \overline{H}), (S4-2-2, S2, I), (S4-2-3, S1, J)\} \end{aligned} \tag{5}$$

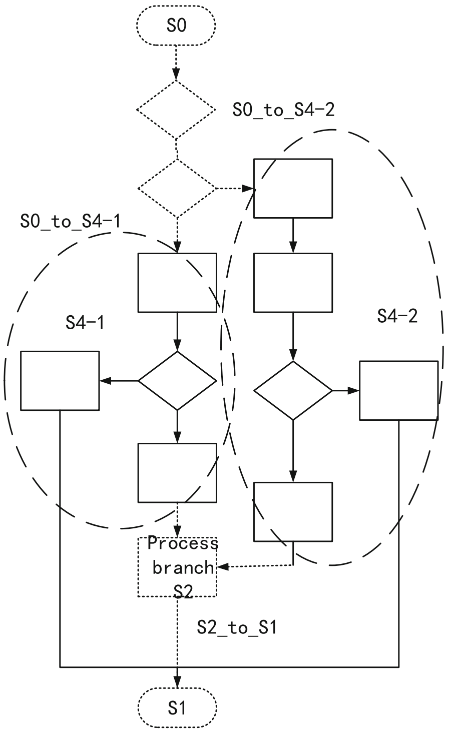
Step 4: Partition the primitive state into sub-states according to the primitive state functions. There is only a "single process" in the initial state partitioned by Step 3, but it may not achieve the goal of single function requirement of the FPGA module designing and implementation. The primitive state is partitioned into sub-states to achieve the requirement of single function sub-state when the functions of the primitive state are multiple. After that, update the set of the states, i.e., S. In addition, some "single process" can also be combined as one according to the coupling degree of functions. For example, if two single processes with the same functions can be merged as one state. There is no branching condition between the sub-states of an primitive state. That is to say that the internal processes of "single process" are in pipeline form, and the next operation begins immediately after the completion of one operation. Finally, update the set of transition conditions, T.



(a) Initial flowchart conforming to the requirements of F-FSM method



(b) Flowchart after procedures of Steps 3.2 and 3.3



(c) Flowchart after procedures of Steps 3.3 and 3.4

Fig. 4. Example of partition the flowchart into primitive states and sub-states

Step 5: Draw the FSM state transition diagram and derive the FSM state transition matrix. The following steps are helpful to draw and derive the state transition diagram and matrix.

Step 5.1: Draw the tree diagram according to the state set S, as shown in Fig. 5.

Step 5.2: Determine the number of FSM states according to the tree diagram.

In general, it is recommended that a FSM have no more than 9 states since when the number of the states is too large the logic of state transition will be too complex to understand and implement. If there are more than 9 states, the number of states can be reduced based on the hierarchy of the tree diagram.

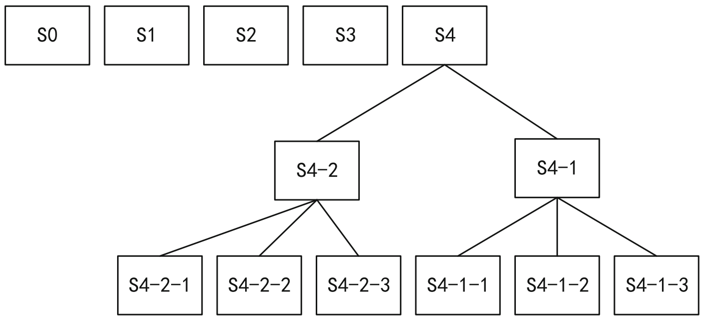


Fig. 5. The tree diagram of the state set S.

As shown in Fig. 5, the first layer has 5 states (namely $S0 - S4$), the second layer has 6 initial states (namely $S0 - S3, S4 - 1$ and $S4 - 2$), and the third layer has 10 initial states (namely $S0 - S3, S4 - 1 - 1/ - 3$ and $S4 - 2 - 1/ - 3$). At this point, $S4 - 1 - 1/ - 3$ can be combined into a state $S4 - 1$ and other states in the third layer can form another FSM, i.e., embedding the new formed FSM into state $S4 - 1$. After that, $S4 - 1 - 1/ - 3$ and the same new state IDLE can be formed into a FSM with four states according to Step 5.2, which can be implemented within $S4 - 1$ in the form of embedding.

Step 5.3: According to the states determined in Step 5.2, the state transition condition set T, draw the FSM state transition diagram according to whether there are transition conditions between states and the direction of state transition. And deduce the transition conditions to output the corresponding position to the state transition matrix.

4 IP Core Design and Implementation of F-FSM Method

The IP core design method of F-FSM is firstly presented in Sect. 4.1 firstly, and then how to use the designed IP core is presented in Sect. 4.2.

4.1 IP Core Design Method of F-FSM

The modular architecture of control and data separation makes the function of the control module become simple, whose function is only to output the enabling signals of other modules according to the changes of input conditions. Therefore, we can design a separate FSM IP core. In the following, we first give the design and implementation principle of the FSM IP core with only 4 states, and then extend it to the design of the finite state IP core with 10 states.

Figure 6 shows an FSM with 4 states and all possible state transitions, where 16 transition conditions can be determined. Table 2 is the parameter definition of the FSM transition condition. In other words, these parameters can be seen as part of the FSM IP core input interfaces.

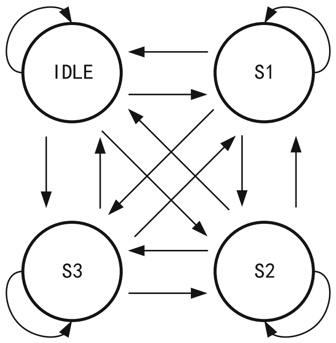


Fig. 6. All possible state transitions in 4-state FSM with $S = \{IDLE, S1, S2, S3\}$, where $S0 = IDLE$.

The inputs of an FSM are directional state transition conditions. That is, a transition from one state to another state. The outputs are enabling signals for corresponding states, indicating that the enabled state module begins to work. Therefore, in the design of the IP core of an FSM with 4 states, the interfaces should include at least the clock and reset signals, 16 transition condition interfaces and 4 state-enabled control signals. Table 2 defines the FSM state transition directions and the state transition parameters. The definition of state transition parameters is named in the direction of the state transition. The state transition condition from $IDLE$ to $S1$ is taken as an example, and its parameter name is *idle-to-s1*.

All of the possible state transition directions and parameters of 4 states can be derived from Eq. (6). Within it, 2 denotes two directional conditions, indicating that any two states among 4 states can jump to each other. 4 means that each state can also transit to itself, so there are altogether 16 transition conditions.

$$2 \times C_4^2 + 4 = 12 + 4 = 16 \tag{6}$$

$$2 \times C_{10}^2 + 10 = 90 + 10 = 100 \tag{7}$$

Table 2. Figure 6's state transition direction and state transition parameter definition

Direction	Parameter	Direction	Parameter
IDLE → S1	idle-to-s1	IDLE → S2	idle-to-s2
IDLE → S3	idle-to-s3	S1 → IDLE	s1-to-idle
S2 → IDLE	s2-to-idle	S3 → IDLE	s3-to-idle
S1 → S2	s1-to-s2	S1 → S3	s1-to-s3
S2 → S1	s2-to-s1	S3 → S1	s3-to-s1
S2 → S3	s2-to-s3	S3 → S2	s3-to-s2
IDLE → IDLE	idle-to-idle	S1 → S1	s1-to-s1
S2 → S2	s2-to-s2	S3 → S3	s3-to-s3

The maximum number of transition conditions that may exist is the number of input ports corresponding to IP core. Therefore, the number of input ports for IP core with 10-states FSM is 100. Due to the excessive number of ports, these ports can be combined and named with certain rules. This is done by merging the 100 ports of the input ports into an input port with a bit width of 100, and similarly merging the output ports into an output interface with a bit width of 10. The corresponding mode of its input and output interfaces is as follows. The transition condition from state i to state j corresponds to the $(10 \times i + j)$th bit of the 100-bit wide input interface, that is, the transition condition from $S2$ to $S3$ is the 23rd bit of the input port. Bit i of the output port with a bit width of 10 is the enabling signal of state i, which controls whether state i is enabled or not.

Figure 7 shows the interface block diagram of 10-state FSM IP core, which encapsulates the traditional three-always style Verilog HDL codes. By combining transition conditions between multiple states as the input interface, the enabling signals of the other functional modules are output. Table 3 is the input/output interface table, where state transition parameters are similar with that defined in Table 2.

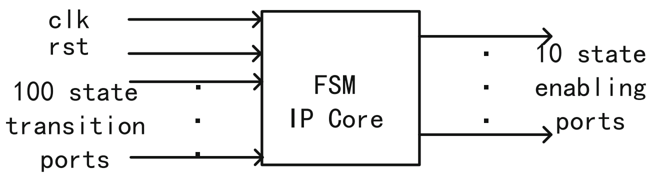


Fig. 7. 10-state FSM IP core interfaces

Table 3. Figure 7's state transition direction and state transition parameter definition

Port name	Width	I/O	Description
clk	1	I	Clock
rst	1	I	Reset
trans-condition	100	I	100 state transitions
state-en	10	O	10 state enabling ports

4.2 System Implementation Method Based on FSM IP Core

According to the modular design architecture of control and data separation idea in Sect. 1, as shown in Fig. 1, the control module and data processing module are effectively separated. The function of the control module is only to control the enabling signals of the other data processing modules and only output the 0/1 control signals of the function modules. Section 4.1 presents the IP core of the designed F-FSM, which can fully meet the single functional requirements of the control module.

In the whole Verilog code implementation process of the system, the design and implementation of the control module can be directly completed by calling the reusable FSM IP core and inputting corresponding state transition conditions, thus reducing a lot of coding time. In addition, the data processing module under the modular architecture is designed as a single functional sub-module. While in different communication systems, some functions are similar or even the same, which makes the functional sub-module under the architecture highly reusable.

5 Performance Evaluation

We implement the proposed F-FSM IP core on FPGA, and verify the correctness and the efficiency of the IP core through functional simulation and mapping.

Figure 8 is the IP core function simulation verification graph of the FSM. In the test case, the state transition conditions between various states were enabled in turn. The functional simulation results show that the implemented F-FSM IP core is correct.

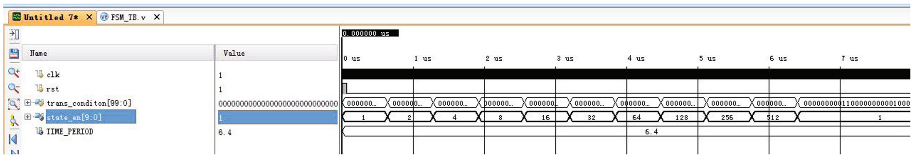


Fig. 8. 10-state FSM IP core functional simulation

Table 4 shows the resource utilization of the implemented F-FSM IP core, among which LUT, register and clock resources account for a relatively low proportion, while input-output interfaces use a large amount of input conditions as the interface in this scheme design.

Table 4. Resource utilization of the implemented F-FSM IP core

Resources	Utilized	Available	Percentage of utilization
Slice LUTs	132	303600	0.04
Slice registers	10	607200	0.00
IO	101	700	14.43
Clocking	1	32	3.12

To sum up, for the implemented F-FSM IP core, the functional simulation result is correct and the resource utilization is low, which meets the design requirements.

6 Conclusion

Finite state machine (FSM) is an important design method of control module in the design of control and data separation modular architecture. However, with the increasing complexity of communication system, how to realize the separation of control and data efficiently and how to design an efficient FSM become a very significant problem. This paper proposes a flowchart-based FSM FPGA design and implementation method for complex communication system, which significantly improves the FPGA development efficiency. This method takes the flowchart describing the complex communication system as input, partition the communication system into modules, and output the FSM transition diagram and transition matrix of the control module. This method can effectively shorten the design time of the communication system and its control module. Finally, an IP core encapsulated in FPGA is designed. This method can effectively improve the development efficiency of control module, and improve the re usability of control module and reduce the workload of code development.

Acknowledgments. This work was supported in part by the National Natural Science Foundations of China (Grant No. 61771392, No. 61771390, No. 61871322 and No. 61501373), and Science and Technology on Avionics Integration Laboratory and the Aeronautical Science Foundation of China (Grant No. 201955053002, No. 20185553035).

References

1. Pirayesh, H., Sangdeh, P.K., Zeng, H.: EE-IoT: an energy-efficient IoT communication scheme for WLANs. In: Proceedings of the IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019, pp. 361–369 (2019)
2. Yan, Z., Li, B., Gao, T., et al.: Design and implementation of FPGA-based transmitter memory management system. In: IEEE International Symposium on Consumer Electronics. IEEE (2014)
3. Li, S., Li, B., Yan, Z., et al.: Design and implementation of DSR routing table entries for FPGA. Appl. Electron. Tech. **44**(12), 89–92 (2018)
4. Jiang, H., Li, B., Yan, Z., et al.: Design and implementation of a frequency hopping hybrid multiple access protocol on FPGA. In: 2018 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC). IEEE (2018)
5. Davis, I.E., Wong, A.: Pipeline method and system for switching packets. United States Patent, 24 March 2015
6. Syed, I., Roh, B.H.: Delay analysis of IEEE 802.11e EDCA with enhanced QoS for delay sensitive applications. In: Performance Computing & Communications Conference (2017)
7. Gill, A.: Introduction to the Theory of Finite-State Machines. McGraw-Hill, New York (1962)
8. Minns, P., Elliott, I.: FSM-based digital design using Verilog HDL (2008). https://doi.org/10.1002/9780470987629:1-22
9. Xu, H., Tang, R., Cheng, Y.: Modular design method for control of reconfigurable machine tools. J. Zhejiang Univ. (Eng. Sci.) **38**(1), 5–10 (2004)
10. Zhang, J.: Design and Implementation of MAC prototype based on FPGA for Wireless Local Area Network. Master Degree Thesis, Northwestern Polytechnical University, Xi'an, China (2015)
11. Ren, P.: Design and implementation of frequency synthesizer based on DDS and PLL. Master Degree Thesis, National University of Defense Technology, Changsha, China (2009)
12. Wen, G.: Design of high efficient state machine based on verilog HDL. Electron. Eng. **32**(6), 4–7 (2006)
13. Chen, Y.: Modelling and optimized design of finite state machine. J. Chongqing Inst. Technol. (Nat. Sci. Ed.) **21**(5), 55–58 (2007)
14. Yu, L., Fu, Y.: Verilog design and research of finite state machine. Microelectron. Comput. **21**(11), 146–148+157 (2004)
15. Luo, X., Li, J., Tian, Z.: Optimization design of FSM based on Verilog HDL. Electron. Qual. **3**(1), 36–38+42 (2012)
16. Feng, K., Li, Y., Zhang, J., Li, J.: Design and implementation of control system based on FSM. Shipboard Electron. Coutermeasure **38**(5), 94–98 (2015)