Design and Implementation of a JTAG Boundary-Scan Interface Controller

Shen XuBang , Liang SongHai

ShaanXi Microelectronics Research Institute

Abstract

In this paper we present an architecture for JTAG boundary-scan Interface Controller which we have implemented as a basic RISC microprocessor chip. We also present a JTAG test language which makes the interface between machine and users very friendly.

1. Introduction

New technologies, such as surface mounting and VLSI, have brought difficulty to test chips and digital systems with the traditional bed-of-nails and probe techniques. So designer's attention was attracted to JTAG boundary scan approach.With the appearance of the proposed P1149.1 standard, JTAG testability design has been widely applied. Many manufactures is producting components which support boundary-scan function defined in the standard. JTAG projection can support IC test, board test, IC on board test and system test. So research on Interface Controller design which is part of JTAG testability design is meaningful. In this paper, we present an architecture for JTAG boundary-scan Interface Controller. We have implemented it as a basic RISC microprocessor chip. We also present a JTAG test language. We can construct a simple stand-alone test instrument with the chip and proper memory or assemble them on a board to be used in PC.

2.Architecture

The Interface Controller's function is to generate P1149.1 signals:TDO,TMS,TCK,TRST and TDI according to testing arrangment. Its logic structure has three parts: command set interpreter, momory interface and P1149.1 interface to scan path, as shown in fig.1. Memory interface consists of DATA[15:0](16-bit data), ADR[23:0](24-bit address), WT(write signal), RD(read signal), RESET and CLOCK signals. P1149.1 interface consists of 5 signals in P1149.1 procotol: TDI, TDO, TMS, TCK and TRST. Command set is selected according to P1149.1 protocol. Command format is shown infig.2(a). Command field only occupies 3 bits. Other bits are control bit C and data field(12 bits). So code efficiency is great. We can see that if command format is expanded to 32 bits code efficiency will be greater. In command set there are eight commands: SCAN, COMP, TMS, COUNT, BRANCH, CBRANCH, STORE and CONTROL, as shown in fig.2(b).

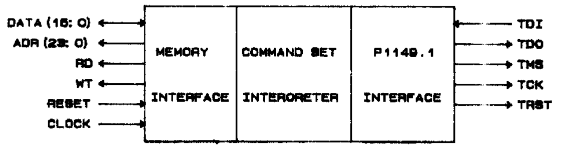
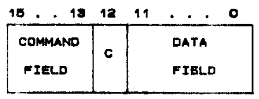


fig.1 architecture structure of Interface Controller



COMMAND FIELD	COMMAND NAME
0 0 0	CONTROL
0 0 1	SCAN
0 1 0	COMP
0 1 1	TMS
1 0 0	BRANCH
1 0 1	CBRANCH
1 1 0	STORE
1 1 1	COUNT

(a) command format (b) command field encode

fig.2 command set

SCAN command writes data to scan ring and reads response data from scan ring. That's to say, it can write to and read from scan ring. It shifts test command code or test data code into scan ring in the meantime it shifts test response code out of scan ring. The length of its efficient data code varies from 0 to 12. So this command can be used to support data or instruction scan in any length.

TMS command sends 13-bit test mode code to test mode register TR to define scan mode as IRSCAN or DRSCAN.

COMP command compares expected result data defined in its data field with test result data shifted out by scan

0-8186-3930-X/93 $3.00 © 1993 IEEE

operation. If the two are equal, test result data is correct and next command turns into excution. Otherwise, test result data is error, test halts.

CONTROL command is a composite command. Its lower 12 bits part is not data field. Together with control bit C, it is used as expanded command field which denotes 13 subcommands. Here we use higher 7 bits as shown in fig.3.

15: 13	12	11	10	9	8	7	6	5: 0	SUBCOMMAND
000	1	0	0	0	0	0	0	0..0	END
	0	1	0	0	0	0	0	0..0	CLEAR
	0	0	1	0	0	0	0	0..0	CAPTURE
	0	0	0	1	0	0	0	0..0	EXIT2
	0	0	0	0	1	0	0	0..0	TRST
	0	0	0	0	0	1	0	0..0	PAUSE
	0	0	0	0	0	0	1	0..0	CLEAROR

fig.3 composite command CONTROL

CONTROL command is used to control scan operation. Its function is defined by the subcommand: 1.CAPTURE subcommand indicates state transition is from state CAPTURE to state SHIFT. It settes CAPTURE flag to denote scan operation is starting; 2. EXIT2 subcommand indicates state transition is from state EXIT2 to state SHIFT.It settes EXIT2 flag to denote scan operation is starting; 3.CLEAR subcommand unconditionally clears CAPTURE and EXIT2 flagges and ends scan operation; 4.END subcommand clears CAPTURE and EXIT2 flagges and ends scan operation when scan operation is finished; 4.CLEAROR subcommand clears OR register; 6.TRST subcommand generates reset signal TRST to reset the logic under test; 7.PAUSE subcommand blocks clock signal of Interface Controller. In fact, lower 13-bit part of CONTROL command code can denotes 13 kinds of controll manner. Here we only use higher 7 bits [12:6]. Setting the corresponding bit to "1", subcommands can be merged into one command.

BRANCH, CBRANCH and COUNT commands are used to control the flow of command program. COUNT command sends a 13-bit number to loop count register LC. JUMP commands BRANCH and CBRANCH are used to send new instruction address to program counter(PC). BRANCH is an unconditional JUMP command. CBRANCH is a conditional JUMP command. When LC isn't equal to 0, JUMP operation is executed. If LC is equal to 0, program goes on to next command following CBRANCH command. To simplify the design only the lower 12-bit part of PC register has count function. So 16M memory space is divided into 4K subareas in 4K location. The data field of JUMP command denotes JUMP address. When control bit C is "1", 12-bit address will be sent to a buffer. When control bit C is "0", 12-bit address will be sent to lower 12-bit part of PC directly and higher 12-bit address will be sent from buffer to higher 12-bit part of PC in the meantime. So 24-bit JUMP address is defined in two JUMP commands. If the address is within a single subarea, we need only one JUMP command which changs the number in lower 12-bits of PC. CBRANCH has the function of subtracting 1 from LC. SO it's also a close command. We use CBRANCH and COUNT to realize the function of LOOP statement defined in test language.

STORE command stores test result data shifted out by scan operation into some location of memory for further analysis. Store address is defined in its data field. So 24-bit store address is also defined in two STORE commands. In STORE command the definition of data field and control bit C is the same as that in JUMP command. If store address is within a subarea, the defininition of store address only needs one STORE command.

3.Logic structure

From the discussion on command set we can see that Interface Controller is a basic RISC microprocessor, as shown in fig.4. It has a simple logic structure which is composed of data path and control logic.

Kernal elements of data path are eight registers. 1.AR is an address-buffer register, 24-bit long(not shown in fig.4). AR receives store address defined in JUMP command; 2.PC is a program counter, 24-bit long. Its lower 12-bit part has the function of adding 1 to its content. PC receives the JUMP address defined in JUMP command. It outputs command address when command is fetched from memory. Store address used in store operation is different from command address defined in PC. In order to fetch command correctly, we buffer store address in AR and use 24 2-to-1 multiplexers to select the output between PC and AR; 3.LC is a loop counter, 13-bit long. LC has the function of subtracting 1 from its content. It receives loop number defined in COUNT command; 4.TR is a test mode register, 13 bit long. In the manner of serial right shift it sends TMS signal to scan ring.It parallel receives test mode code defined in TMS command; 5. IR is a scan-in register, 12 bit long. In the manner of serial left shift it receives test result data scanned out from scan ring; 6. CPR is a result comparision register, 12-bit long. It receives expected result data defined in COMP command. We need to compare its content with test result data received in IR. So we place a compare circuit between CPR and IR; 7.OR is a scan-out register, 13-bit long. It serially sends test instruction code or test data code into scan ring. It parallel receives test instruction or test data defined in SCAN command; 8. CR is a command register, 16-bit long. Once a command has been fetched from memory, it will be sent to CR at first.

Control logic interprets command and generates control signals for data path according to current command and status fed back from data path. It is composed of a clock pulse generator, a command interpreter and combinational control logic. Its structure is very simple.

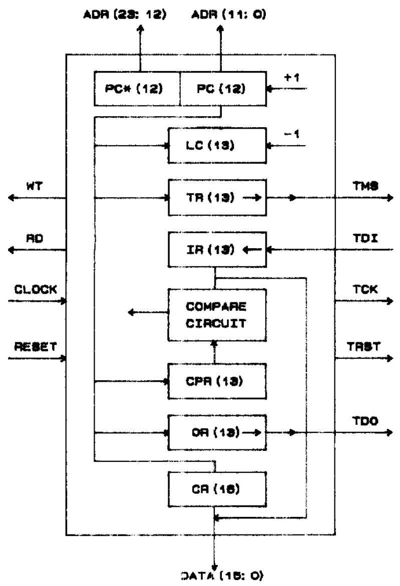


fig.4 logic structure of Interface Controller

To simplify the design we have solved following three problems. The first one is the the definition of test mode code in TMS command. If generating one bit of test mode code needs one shift operation in TR as shown in JTAG TAP state map, the length of TR will always greater than 13. Here we solve it in two steppes. Firstly, we use code "000" to denote those five zero steady states in TAP state map; secondly, the test mode code in any TMS command only contains a NO SHIFT steady state (RUN-TEST/IDLE, PAUSE-DR, PAUSE-IR) or a SHIFT steady state (SHIFT-DR, SHIFT-IR) and a NO SHIFT steady state. When entering those steady states, TR stops doing any shift operation and the TMS code it generates is always "0". If current steady state is SHIFT, it will be released when the corresponding scan operation is finished, and TR will come into a NO SHIFT steady state at last. To realse TR from those NO SHIFT steady states, a new TMS command is needed to send new test mode code into TR. In this manner we only need one TMS command to generate all test mode code when we use SCAN commands to shift test instruction code or test data code of any length into scan ring.

The second one is the control of scan length. Because test instruction and data is defined in SCAN command sequence, their length is defined by the length of the sequence. So we don't need a counter.

The third one is the succession of scan operation. No matter how long the scan sequence is, we want to do scan operation in a successive pulse series to improve scan efficiency. We can find that scan is a serial operation, but fetching command is a parallel operation. So pipeline is used here. When a SCAN command executes scan operation, next command is fetched into CR. That's to say, when a SCAN command turns into operation, in the meantime the next command is fetched from memory. If it is a SCAN command, it will be sent into CR to wait for current scan operation being finished. If it is a COMP or STORE command, it will turn into execution at once. When its operation is finished, the next SCAN command is fetched into CR. Because when SCAN command needs to execute 12-bit serial shift operation, it is possible to fetch two commands in this period. In fact, all commands except SCAN and END are executed in two paces: P0 fetching, P1 excuting. The more paces which SCAN or END command needs are generated by adding more P1. From P1149.1 interface point of view all scan signals are generated successively and scan rate is the highest.

4. Test language

If we write test program in a assembler language of the command set, we will find it is a boring work. So we present a concise JTAG test language. It only has four statements: SCAN OPERATION, COMP OPERATION, STORE OPERATION and LOOP OPERATION. The program writed in JTAG test language is translated by a compiler into commands, which directly executes on Interface Controller.

The operand of scan operation is test instruction or test data. In IRSCAN operation or DRSCAN operation they are sent to the instruction register or data register in the chip under test. SCAN OPERATION statement is defined as follows:

```
SCAN OPERATION::=IRSCAN(test instruction)
                 ¦DRSCAN(test data)
```

There are seven test instructions defined in P1149.1 protocol openly. we use their names to denote them, as shown in below:

```
test instruction::= Bypass | Extest ¦ IDcode ¦ Intest¦
                    RunBIST ¦ Sample ¦ Usercode
```

They are encoded by the compiler. Test data are expressed in binary, octadic or hexadecimal number.

Test result data received from TDI port can either be compared directly or be stored for further analysis. The corresponding operation are compare operation and store operation. They are always used in company with scan operation. The corresponding statements are defined as follows:

```
COMP OPERATION ::=COMP(expected instruction
                        |expected data)

STORE OPRRATION::=STORE[address1:address2]
```

The expression of expected instruction and expected data is the same as that of test instruction and test data. Address1 and address2 are expressed in binary, octadic or hexadecimal number.

To define and control the circular execution of scan operation, we defined a LOOP OPERATION statement. This statement can only be placed behind those SCAN OPERATION statements which need to be executed looply. Its definition is shown as follows:

```
LOOP OPERATION::=LOOP[loop-begin address:loop
                        times]
```

To make the program more readable, we define the SCAN OPERATION statements sequence which has integrity meaning as STATE i. So we can use STATE i to denote loop-begin address. For example, the LOOP OPERATION statement shown in fig.5 indicats the statements sequence from STATE i to the statement just before it will be excuted 25 times.

```
STATE i       IRscan(......)
              DRscan(......)
                 .
                 .
                 .
              DRscan(......)
STATE i+1     IRscan(......)
              DRscan(......)
                 .
                 .
                 .
STATE k       IRscan(......)
              DRscan(......)
              LOOP(State i, 25)
STATE k+1     IRscan(......)
```

fig.5 An example for test procedure

5.Conclusion

We have implemented Interface Controller in Microelectronic Centre with gate array technology which has 9690 paires of PMOS and NMOS transistor in a chip. Package of the chip is PGA68. The chip operates at 20MHZ.

Interface Controller is designed as a basic RISC microprocessor. With the techniques of code compression and pipeline, we simplify its data path and control logic. Its command set has great code efficiency and provides plentiful control function. The command set can be further expanded.

The way in which Interface Controller is designed and implemented makes its application more flexible. It supports many kinds of hardware assemble projection: the first one is to assemble the Interface Controller chip into the application system under test; the second one is that the Interface Controller chip can be used to construct a stand-alone test instrument; the third one is that we can assemble the Interface Controller chip and proper RAM on a board and put the board under the supervision of the host computer. From application point of view, JTAG boundary-scan Interface Controller designed as a RISC microprocessor is better than the one designed in [3], which interfaces parallel host microprocessor data to P1149.1 test bus.

The test language simplifies the design of test programs. Users don't need to consider the command set when they write their test program. They can directly write in the test language. The concise language makes the interface between mathine and customer very friendly.

Now we are working at a test instrument which is composed of Interface Controller, memory and host computer.

REFERENCE

[1] Shen XuBang, The Strategy Of JTAG Testability Design, MINI-MICRO SYSTEMS, 1990, Vol.12, No.1, PP.10-18.

[2] Shen XuBang, Liang SongHai, An Implementation Design For The Boundary-Scan Architecture, MINI-MICRO SYSTEMS, 1991, Vol.12, No.11, PP.1-8,32.

[3] Sue Vining, Tradeoff Decisions Made For A P1149.1 Controller Design, International Test Conference 1989 Proceeding, PP.47-54.