Exploiting Boundary Scan functions of FPGA on ATE

by

Kunpeng Zeng

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
August 1, 2015

Keywords: Boundary Scan, FPGA, ATE

Abstract

With the increase of integration density at chip level, the controllability and observability of defects become complicated, resulting in limited physical probe access to I/O pins. It is impractical to apply test stimulus to all the input pins and capture outputs from output pins on a chip within a printed circuit board (PCB). Boundary scan is a helpful method to test with reduced physical probe access to IC pins on complex PCBs.

JTAG is mostly synonymous with the term "Boundary scan". By adding some JTAG test logic, including four JTAG pins (TCK, TMS, TDI and TDO), several registers, and a TAP controller, we are able to program the ADVANTEST T2000GS tester to perform functional test of an IC with physical access limited to only JTAG pins. In addition, some other JTAG functions, such as checking the device type and the integrity of JTAG pins, examine the correctness of on-chip logic etc. can also achieved by JTAG.

The proposed procedure for performing functional test using JTAG pins was verified through experiments. The experiments are carried on the Advantest T2000GS ATE located at Auburn University. The device under test (DUT) is a Mercury development board, which contains a XILINX Spartan-3A FPGA. Benchmark circuits of ISCAS'85 and ISCAS'89 are used as the configured on-chip logic. In addition, the verification of FPGA JTAG interface and components are also conducted.

Acknowledgments

First and foremost, I would like to thank my academic advisor Dr. Victor Nelson for his valuable guidance and advice throughout my work and study. He gives me useful suggestions and directions, without which I can barely complete my thesis. I benifit a lot from his enthusiastic and well-structured teaching, along with the analytic thinking in solving problems.

I would like to express my gratitude to Dr. Vishwani Agrawal for being my committee member and his Advance VLSI design course. He is always kind and helpful. I thank Dr. Adit Singh for his wonderful class of VLSI testing, which served as the foundation of my work.

I also would like to thank my friends and classmates. From the academic discussion with them, I learned a lot. I own my gratitude to Baohu Li and Praveen Venkataramani, who taught me how to operate on ATE.

At last, I would like to thank my family for their support all the time. I gratefully dedicate this work and efforts to them.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Very large scale integrated (VLSI) circuit testing

As integrated circuit (IC) technology has developed, since it was born in the 1960s, the electrical industry has changed a lot. VLSI testing has been a part of VLSI technology from the very beginning. From the first design to the final production of a VLSI device, faults and errors may happen in every stage. Many testing methods have been invented, aimed at the emerging new VLSI technology throughout the past few decades.

A manufactured product undergoing test is referred as a device under test (DUT). The principle of testing a DUT is as follows.

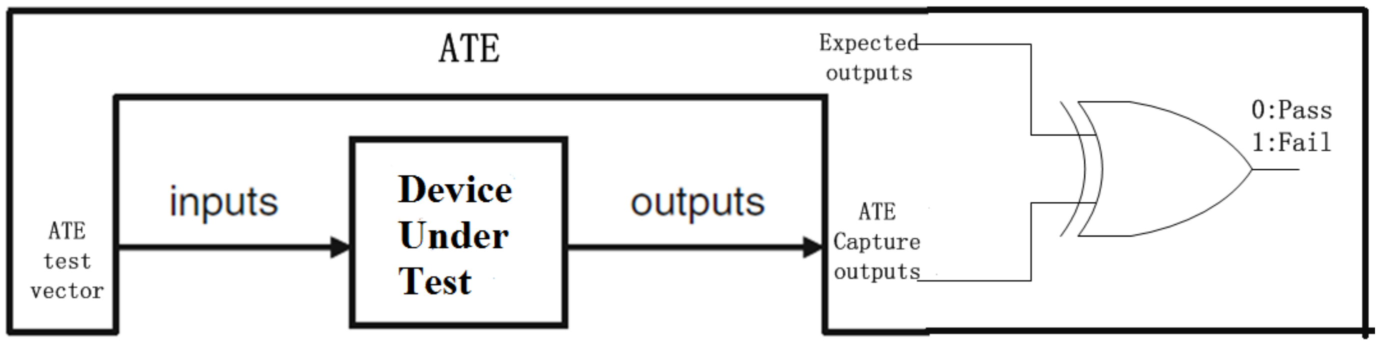


Figure 1.1: Principle of using ATE to test DUT

As shown in Figure 1.1, automatic test equipment (ATE) is used to perform functional test of a given DUT. The DUT has both inputs and outputs. The ATE firstly applies test vectors to the DUT's inputs, and then the outputs are captured by the ATE. The captured outputs are compared to the expected outputs; if they match, we can assume that this DUT functions correctly; if not, we may assume that the DUT is faulty.

Comparing current VLSI technology to that of decades ago, the feature size of transistors has scaled down, resulting in more and more transistors integrated into one die. The main problem in VLSI test is that chips contain many transistors, but can be accessed only via a small number of pins. Similar issues are encountered in printed circuit board (PCB) design and test.

In the 1970s, the mainstream of PCB testing was in-circuit testing (ICT). ICT is mainly used to test PCBs of low-density that consist primarily of circuits in dual in-line packages (DIPs). However, in the 1980s, ICT became ineffective and almost impossible for testing PCBs because of the development of VISI technology. The increasing integration at the chip level complicates controllability and observability of defects, resulting in limited physical probe access of I/O pins.

To provide better access to IC pins on complex PCBs, a long-term solution to testing with reduced physical probe access was to implement a serial shift registers around the boundary of each IC, connected to the IC's I/O pins. Figure 1.2 shows a small device that has a boundary scan cell (BSC) at each of its I/O pins, with the output of each BSC connected to the input of the next BSC. These BSCs together make up a boundary scan register (BSR).

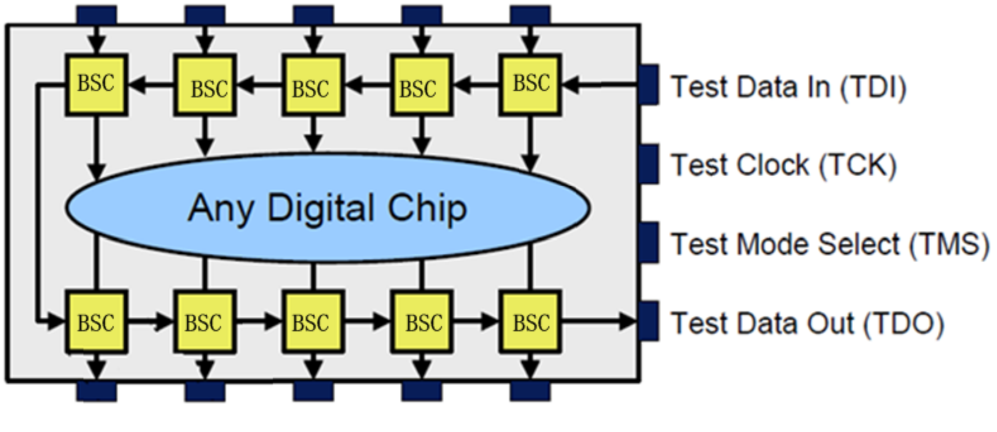


Figure 1.2: A small device and its boundary scan register (BSR) [1]

Figure 1.3 shows a circuit board that contains four boundary-scan devices. The scan path at the board level is formed by chaining all the BSR of each chip together.

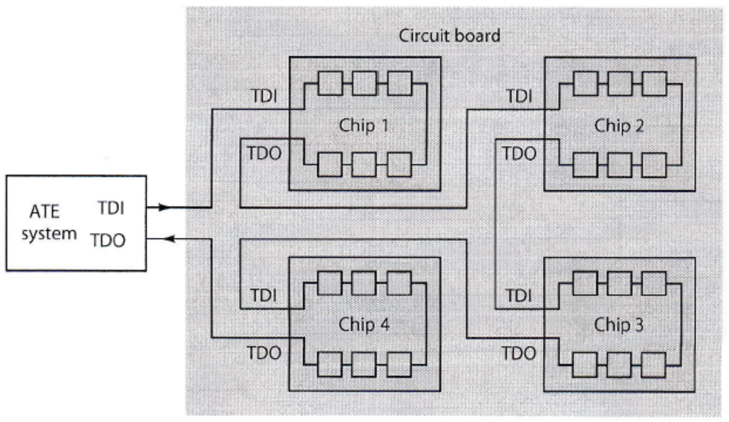


Figure 1.3: A circuit board that has four boundary-scan devices

In 1985, a group of European manufacturers formed the Joint European Test Action Group (JETAG). In 1986, with the addition of North American companies, JETAG became Joint Test Action Group (JTAG) [3]. In 1990, the JTAG 2.0 test standard was approved and formed the basis of the IEEE Standard 1149.1, "Test Port and Boundary-Scan Architecture". In 1994, a supplement that contains a description of the Boundary Scan Description Language (BSDL) was added to the standard to facilitate test development.

Since that time, the JTAG standard has been adopted by most electronic device companies all over the world. JTAG is mostly synonymous with the term "Boundary Scan".

1.2 Application of JTAG

Concerning an assembled product, we have three basic goals:

(1) To confirm that the components are interconnected in the correct manner on the PCB.

(2) To confirm that the components in the product interact correctly and that the product performs its intended function.

(3) To confirm that each component performs its required function.

The IEEE 1149.1 standard defines test logic and the test access port (TAP) that can be included in an integrated circuit to provide standardized approaches to achieve these goals.

Goals 1 and 2 are achieved by testing the interconnections between integrated circuits once they have been assembled onto a printed circuit board (PCB) or other substrate.

As figure 1.4 shows, JTAG can be used to test the interconnections between two devices via the BSR. The output of a BSC on the first chip is connected to the input of a BSC on the second chip.

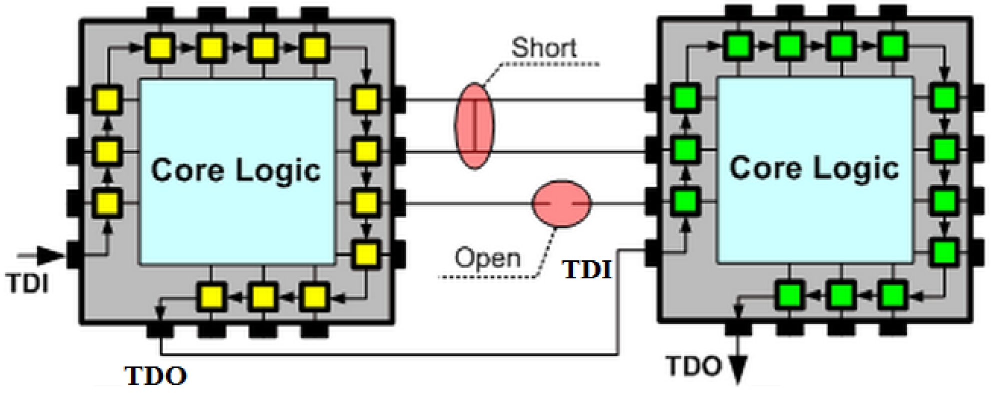


Figure 1.4: Use JTAG to test interconnections between two devices

The JTAG EXTEST instruction can be used to test if the interconnections between the components are intact by sending data from the BSC on one component to the connected BSC on the other component, and verifying that the correct data was received.

Goal 3 testing each IC itself is achieved as follows.

As figure 1.5 shows, the inputs of an on-chip circuit are normally applied through external input pins, and through the BSR. The outputs of an on-chip circuit are sent out to external output pins through the BSR.

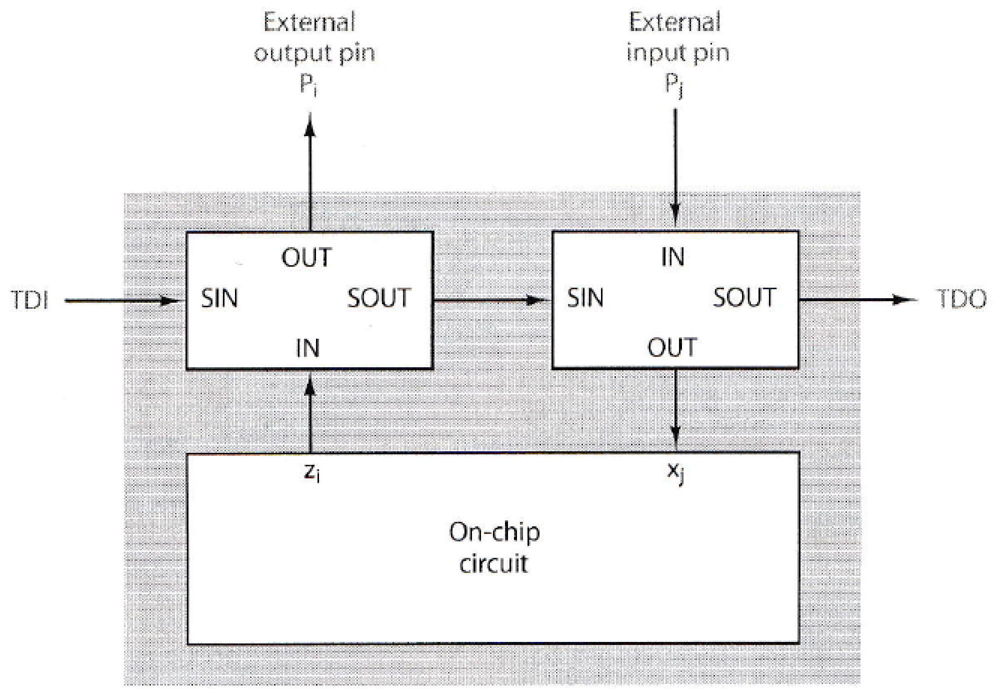


Figure 1.5: Use JTAG to test the function of on-chip circuit

The JTAG INTEST instruction can be used to test the on-chip logic by using the BSC to apply test vectors to the on-chip circuit, and capturing the circuit outputs in the BSR.

In addition to the above, other functions on the IC can be accessed via the JTAG interface. For example, other JTAG instructions, such as IDCODE, can be used to determine what the component is, BYPASS can be used to shorten the scan chain length on the PCB, and so on.

JTAG also supports vendor-defined operations. For example, JTAG can be used to configure many programmable devices. In most FPGAs, when the FPGA mode pins are set for JTAG mode, the FPGA can be configured by downloading configuration data to the device via the JTAG ports after a power-on event. There are vendor-specified configuration registers inside the FPGA that handle the process of processing configuration data. Many FPGA manufacturers provide commercial tools for easy JTAG configuration, such as ISE Design Software from XILINX [12], and Digilent Adept software from Digilent Inc [2].

Many microcontrollers include debug and other functions that can be accessed via a JTAG interface. The CPU processor core is hidden from observation or control by caches, memory, buses, and peripherals, such as I/O blocks. However, the JTAG path provides a direct connection into the logic inside the CPU. With this direct core access, JTAG can be used as a way to debug the hardware with the help of host-based debugger software, by passing data between the microcontroller and the host. JTAG can be also used to program microcontroller flash memory. A flash device can be programmed through JTAG by scanning the address and program data into the device.

1.3 Problem statement

In a PCB board of high density, physical access to every I/O pin is impossible, thus making it difficult to perform testing by conventional methods. This thesis explores a way to perform functional test of an IC with physical access limited to only JTAG ports. The primary focus is to develop test procedures for the ATE that exploits the existing JTAG resources on a device to facilitate testing the device.

In addition, this work also exploits some other JTAG functions of XILINX FPGAs, such as using the IDCODE instruction to check the device type and the integrity of JTAG ports, using the SAMPLE instruction to examine the correctness of on-chip logic through JTAG ports, and so on.

The automatic test equipment (ATE) used in this work is an Advantest T2000GS. The DUT used in the study test procedures is a Mercury Development Board [4], which has a XILINX xc3s50a vq100 FPGA [5] mounted on it. This FPGA contains JTAG components that implement the IEEE 1149.1 standard.

1.4 Organization of this thesis

This thesis is organized as follows. In Chapter 2, basic JTAG hardware architecture and JTAG timing are introduced. Chapter 3 introduces the hardware architecture of the

Advantest T2000GS ATE and the design of test programs that runs on the ATE. Chapter 4 gives detailed information about the verification of the JTAG interface and registers. Chapter 5 describes the procedure to perform functional test of an IC via its JTAG interface with an ATE. Chapter 6 gives examples to show functional tests of combinational and sequential benchmark circuits implemented in the FPGA. Finally, in Chapter 7, conclusions of the current work are drawn and relevant future work is suggested.

Chapter 2

JTAG architecture and timing

In this thesis, the DUT used is the Mercury development board [4], which contains a XILINX Spartan-3A FPGA xc3s50a vq100 [8]. Spartan-3 generation FPGAs are compliant with the IEEE Standard 1149.1 Test Access Port and Boundary-Scan Architecture [20].

In the remaining part of this thesis, this XILINX xc3s50a vq100 FPGA will simply be referred as "the FPGA", and the IEEE 1149.1 standard will be referred as "the JTAG standard".

As shown in Figure 2.1, the JTAG architecture mainly consists of four JTAG pins, which are to be accessed by the ATE, several registers, and a TAP controller. The registers include instruction register, boundary scan register, bypass register and other miscellaneous registers. In addition, the JTAG standard also defines standardizations to implement various instructions.

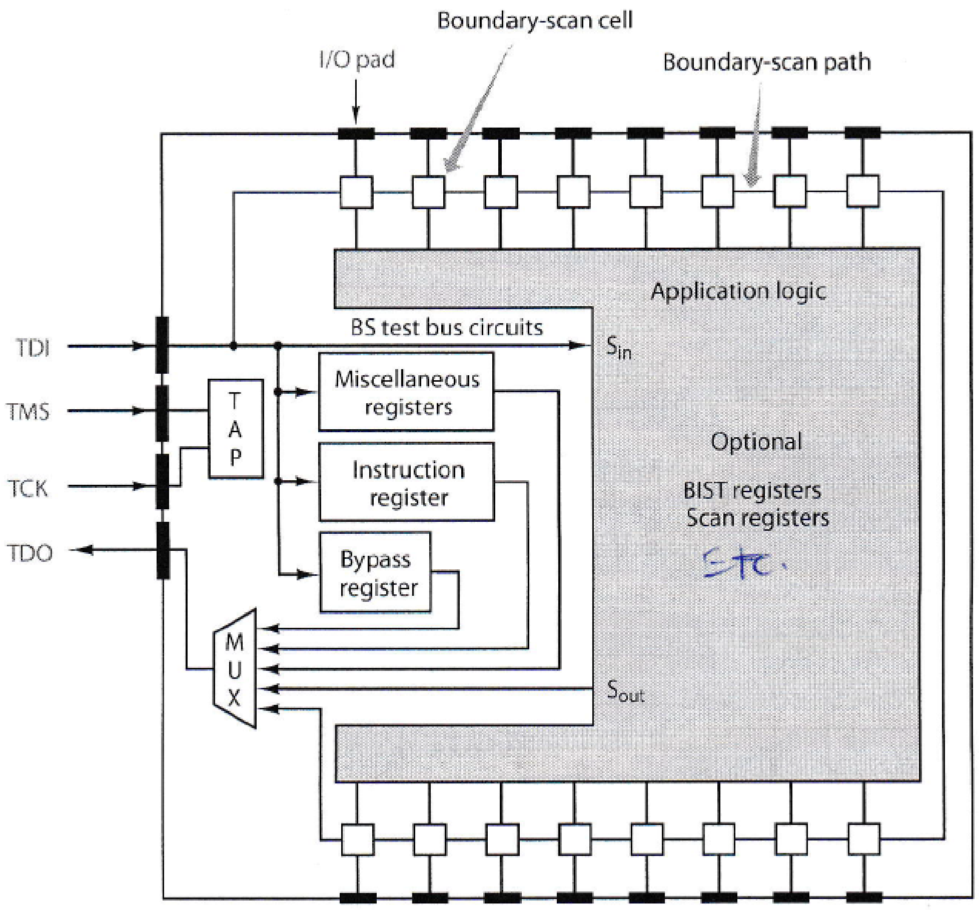


Figure 2.1: Standard JTAG architecture

2.1 JTAG pins

The JTAG port is composed of a minimum of three inputs and one output that are required by the test logic defined for the JTAG standard. The three inputs are: test clock (TCK), test mode select (TMS) and test data input (TDI). The one output is test data output (TDO).

The usage of these four JTAG ports is described concisely as follows:

(1) TCK: The TCK provides the clock for the test logic.

(2) TMS: The signal applied at TMS is used to control the state of TAP controller.

(3) TDI: Test instructions and data are applied at TDI serially to perform test and other operations.

(4) TDO: The output of test instructions and test data from test logic is shifted out serially at TDO.

There is also an optional fourth input pin, test reset (TRST), that can be provided for asynchronous initialization of test logic. The FPGA used in this study does not have a TRST input pin. Through TCK and TMS, the TAP controller can produce the same effect as TRST, as will be described in section 2.4.

2.2 JTAG registers

There are several registers in the JTAG architecture; usually they can be categorized into two types: Instruction Register (IR) and Test Data Register (TDR).

2.2.1 Instruction Register (IR)

IR is crucial to the JTAG architecture. All test operations of JTAG are performed based on a specified instruction and IR is the register that holds the instruction. IR is connected between TDI and TDO, with instruction codes shifted in via TDI. The length of IR varies by device type, and it is the same length as the instruction's binary codes.

In general, the last two bits of the IR are always forced to 01 when a new instruction is to be shifted in, and it will be replaced by the instruction code. Because there are four instructions that are mandatory in JTAG, the length of IR must be at least 2 bits. For Spartan-3 generation FPGAs, the length is 6 bits, so each instruction is also 6 bits.

2.2.2 Test Data Register (TDR)

The TDR category includes several registers that provide data for instructions. These include the boundary scan register (BSR), bypass register, and device identification register (DIR), each of which has different functions selected by different instructions. One of these registers is connected between TDI and TDO at any time for supplying and reading back data.

2.2.2.1 Boundary Scan Register (BSR)

The BSR allows testing of connections to external components and internal logic circuits. The BSR consists of multiple boundary scan cells (BSC) chained together as one path, as shown in Figure 2.1. Each BSC has 4 connections: two as inputs and two as outputs. As shown in Figure 2.2, the two inputs of a BSC are PI and SI, and the two outputs are PO and SO.

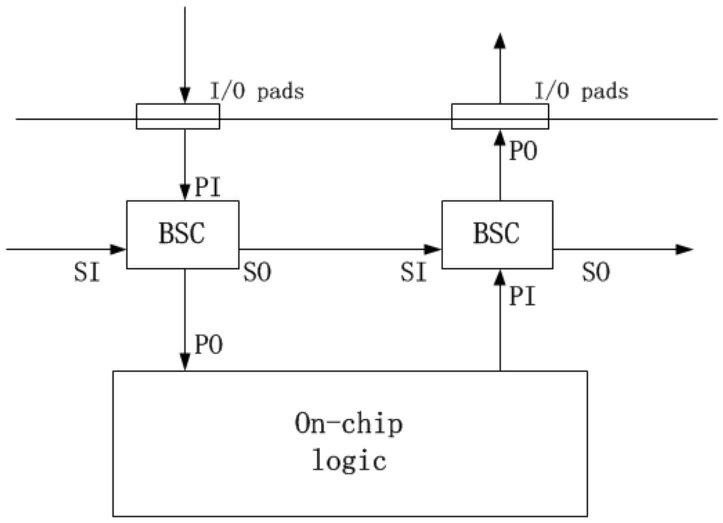


Figure 2.2: Connections of boundary scan cell (BSC)

(1) PI stands for Primary Input; this is the data from an external device or from on-chip logic.

(2) PO stands for Primary Output; this is the data sent to on-chip logic or to an external device.

(3) SI stands for Scan Input; SO stands for Scan Output. They are the data that serially scan from the SO of one BSC to SI of the next BSC to create a scan chain. SI of the first BSC is connected to TDI, SO of the last BSC is connected to TDO. A complete scan chain is shown in Figure 2.1.

Figure 2.3 shows the typical structure of a BSC, which consists of two multiplexers and two D-flip-flops (DFF). The control signals of each component (ClockDR, UpdateDR, ShiftDR, Mode) are provided from the TAP controller and IR.

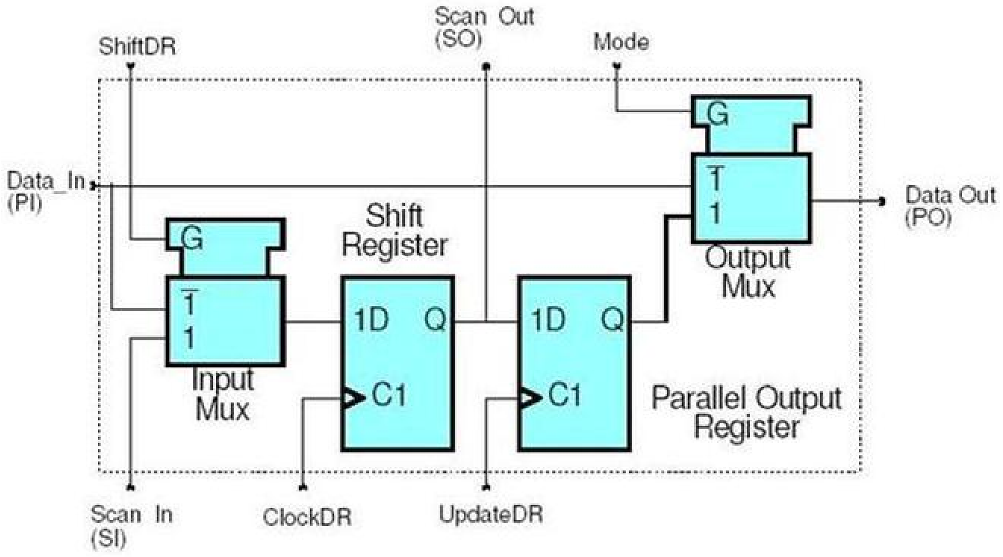


Figure 2.3: Structure of a boundary scan cell (BSC)

In different instructions and TAP controller states, the control signals are also different, which cause four different functional operations of the BSC.

(1) Normal: data transfer from PI to PO. At the output mux, control signal Mode=0. In this mode the BSC is transparent, i.e. data moves normally between on-chip logic and externally connected devices.

(2) Scan: data transfer from SI to shift register. At the input mux, ShiftDR=1, and ClockDR pulses.

(3) Capture: data transfer from PI to shift register. At the input mux, ShiftDR=0, and ClockDR pulses.

(4) Update: data transfer from shift register to parallel output register when UpdateDR pulses.

The number of BSCs is dependent on the specific device. Typically there is one BSC per I/O pin, as shown in Figure 2.1. However, for a bidirectional I/O buffer, as in an FPGA for example, an I/O pin may have three BSCs as shown in Figure 2.4, with one BSC for input data, one for output data, and one for tristate buffer control.

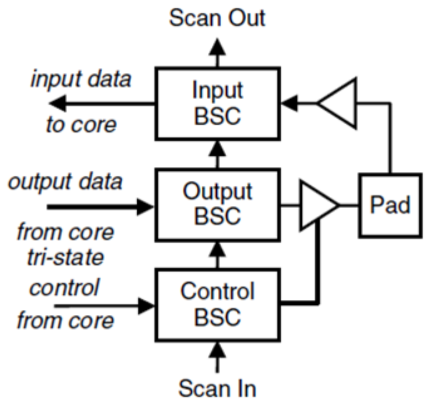


Figure 2.4: Construction of triple BSC bidirectional I/O buffer [6]

The FPGA used in this study has a total of 373 BSCs. Some pins are bidirectional I/Os, each of which has three BSCs: input BSC, output BSC and control BSC. Some I/Os are input only, and only have an input BSC. Besides the BSCs connected to I/Os, there are also some uncontrollable and unobservable internal BSCs. The number of BSCs of different types in the FPGA is listed in Table 2.1.

Table 2.1: Number of BSC of different types in the xc3s50a vq100 FPGA

BSC type	number
input	68
output3	62
controlr	62
internal	181

"input" BSC is a cell that can pass input data to on-chip logic. "output3" BSC is a cell that can drive data to a 3-state output. "controlr" BSC is a control cell that is forced to its disable state in the TEST-LOGIC-RESET controller state. “internal” BSC is an uncontrollable and unobservable cell that is not associated with a device signal.

2.2.2.2 Bypass register

The bypass register is only one bit, and is the default register connected between TDI and TDO if a device identification register (DIR) is not provided. It is mandatory in the JTAG architecture. The bypass register provides a minimum length serial path for the movement of test data between TDI and TDO, bypassing all other on-chip logic.

The architecture of the bypass register is shown in Figure 2.5. It does not perform any system function and has no effect on the on-chip system logic.

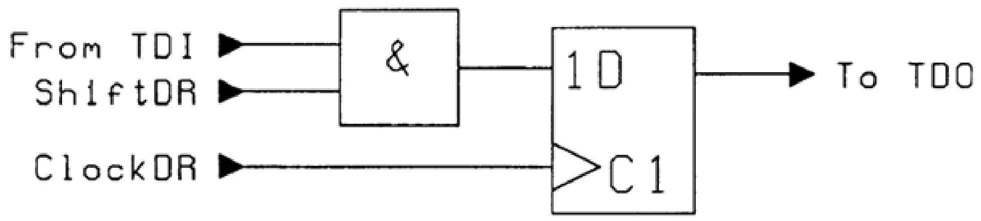


Figure 2.5: The components that form bypass register

Usage of the bypass register in a component can speed the access time to provide data to TDRs in other components during a board level test. An example is shown in Figure 2.6.

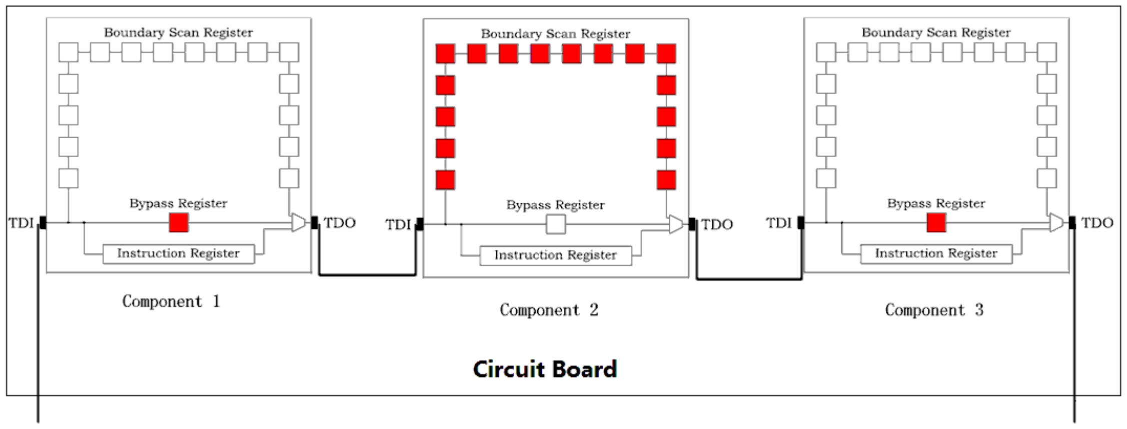


Figure 2.6: Usage of bypass register to bypass some components

Assume that the components are the FPGA used in this study, and that the circuit board has three components, each of which has 373 boundary scan cells (BSCs). Now assume that we just want to test the integrity of the BSR of the second component. If all components are set with BSR between TDI and TDO, it would require 1119 clock cycles to load all BSCs. However, if the two other components are set with bypass register between TDI and TDO, it would require only 375 clock cycles to load the BSC of the second component, which reduces the test time considerably.

2.2.2.3 Device Identification Register (DIR)

DIR is an optional register. If it is provided, DIR should be the default register connected between TDI and TDO when test logic is initialized. The purpose of DIR is to distinguish the manufacturer of components on a board when multiple components that have JTAG are used. DIR can be chosen by default when test logic is initialized, or by executing the IDCODE instruction.

DIR does not perform any system function and has no effect on the on-chip system logic. It is a dedicated part of the test logic. As shown in Figure 2.7, the information differs by vendor and is usually 32 bits, which include the manufacturer, part number, and version of a component. The data in DIR is usually read-only.

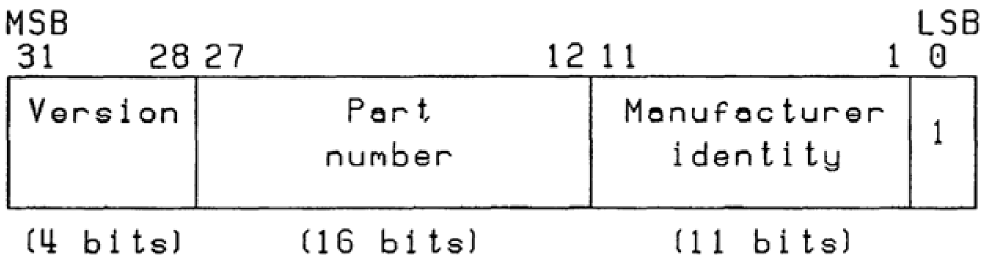


Figure 2.7: The information provided by Device Identification Register (DIR)

An example is shown in Chapter 4, where the IDCODE instruction is executed and the 32 bits of information are shifted out through TDO.

2.2.3 Vendor-defined registers

The JTAG standard allows vendors to provide some vendor-defined registers that utilize the JTAG architecture for various functions that are not part of the standard. Two examples are as follows.

2.2.3.1 Configuration Register

Some devices, such as an FPGA, can configure their internal logic through a configuration register. The configuration of an FPGA is required to define lookup table (LUT) equations, signal routing, flip-flop reset polarity, IOB voltage standards, and various other aspects of the user design. The configuration information, along with configuration instructions, is combined to form a bitstream that will be loaded into the FPGA internal logic.

The JTAG configuration register in the Spartan-3 generation FPGAs is 32 bits, which allows access to the configuration bus through the CFG_IN instruction and read back operations through the CFG_OUT instruction. Read back operations provide for interactive debugging.

2.2.3.2 USERCODE register

If the device component is user-programmable, such as an FPGA for example, then the USERCODE register can be provided and programmed with a design-specific identification code. The USERCODE can be programmed into the device and can be read back in order to verify the configured design.

2.3 JTAG instructions

The JTAG standard defines instructions that can be loaded into IR serially during an instruction-register scan cycle, and then decoded and executed to control test operations. Instructions can be categorized into public instructions and private instructions. They are introduced briefly in this section.

2.3.1 Public instructions

Every device that is compatible with the JTAG standard must support four public instructions: BYPASS, SAMPLE, PRELOAD, and EXTEST. Thus, in order to implement these mandatory instructions, the minimum IR length should be at least two bits.

- BYPASS: When this instruction is executed, the bypass register is connected between TDI and TDO.
- SAMPLE: A snapshot of the normal operation of the on-chip logic can be captured to examine by executing the SAMPLE instruction. The required test data can be loaded in parallel to the BSR during CAPTURE-DR state of the TAP controller, and the test result can be shifted out serially onto TDO during SHIFT-DR state.
- PRELOAD: This instruction allows a data pattern to be placed at the latched parallel outputs of BSCs before the selection of another instruction that uses the BSR, such as INTEST. It has no effect to the internal logic.
- EXTEST: This instruction is mainly used to test interconnects between ICs on a board. Several interconnects can be tested simultaneously by shifting the required data to the BSCs on the device output pins and then capturing data in the BSCs of the device input pins.

2.3.2 Private instructions

Private instructions are not mandatory in the JTAG standard. Some instructions are defined in the JTAG standard, but are optional, while some are defined by the vendor. A component manufacturer can define its own set of private instructions to use the TAP and test logic for design verification, fault diagnosis, etc. The JTAG standard defines instructions that include INTEST, IDCODE, HIGHZ, USERCODE, and so on.

- INTEST: This instruction allows testing of the on-chip system logic while the component is assembled on the board, even with limited physical I/O pin access. The test stimuli are shifted in through TDI and applied by the BSR to the on-chip system logic. Then the test results are captured into the BSR and shifted out through TDO.

- IDCODE: When the IDCODE instruction is used, the vendor's identification code, which contains manufacturer's information, is loaded into the device identification register and scanned out via TDO.

- HIGHZ: This instruction forces all output pins of the chip to an inactive-drive state.

- USERCODE: It allows a user-programmable 32-bit identification code to be loaded into the user-defined register. It is useful when the chip can be programmed with different designs and it is necessary to determine which on-chip design is present.

- CFG_IN: Some FPGAs have a CFG_IN instruction to access the configuration register for FPGA configuration.

2.4 TAP controller

The TAP controller is a synchronous finite state machine that has 16 states, shown in Figure 2.8. The state transition is controlled by the TMS and TCK signals.

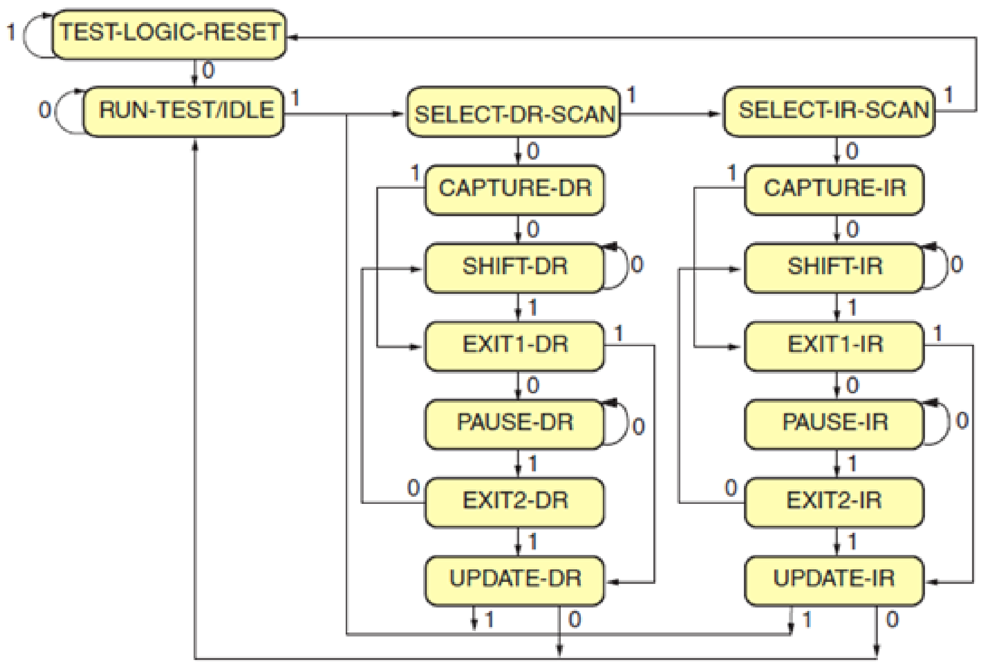


Figure 2.8: TAP state machine

In Figure 2.8, the value shown adjacent to each state transition is the signal value applied at TMS, which is sampled at the rising edge of TCK. The behavior of the TAP controller states is briefly described as follows.

(1) TEST-LOGIC-RESET (TLR): All test logic is disabled, enabling normal operation of the FPGA. We can see from the diagram that no matter what state the TAP controller is in, TLR can be entered by holding TMS high and pulsing TCK five times. This is the reason why the TEST Reset (TRST) pin is optional in the JTAG standard.

(2) RUN-TEST/IDLE (RTI): The test logic is active only when certain instructions are executed; for example the RUNBIST instruction. For instructions that do not have effect in RTI state, the test logic is idle.

(3) SELECT-DR-SCAN: This is a temporary state, from which the TAP controller either enters the SELECT-IR-SCAN state, to provide an instruction, or the CAPTURE-DR to access a data register.

(4) SELECT-IR-SCAN: This is a temporary state, from which the TAP controller either returns to TLR or enters the CAPTURE-IR state.

(5) CAPTURE-DR: Data is parallel-loaded into the TDR selected by the current instruction on the rising edge of TCK.

(6) SHIFT-DR: One of the TDR is connected between TDI and TDO. Test data is shifted into the register serially through TDI and shifted out of the register at TDO.

(7) EXIT1-DR: This is a temporary state, from which the TAP controller either enters the PAUSE-DR state or the UPDATE-DR state.

(8) PAUSE-DR: It allows the test data shifting to be temporarily halted.

(9) EXIT2-DR: This is a temporary state, from which the TAP controller either enters the SHIFT-DR state or the UPDATE-DR state.

(10) UPDATE-DR: Test data is latched into the parallel outputs of the TDR on the falling edge of TCK.

(11) CAPTURE-IR: The shift register in the IR parallel loads a pattern of fixed values on the rising edge of TCK, before entering SHIFT-IR. Notice that the last two bits will always be 01.

(12) SHIFT-IR: The IR is connected between TDI and TDO. Instruction data is shifted into the register serially through TDI and out through TDO.

(13) EXIT1-IR: This is similar to EXIT1-DR.

(14) PAUSE -IR: This is similar to PAUSE -DR.

(15) EXIT2-IR: This is similar to EXIT2-DR.

(16) UPDATE-IR: The instruction shifted into the IR is latched to the parallel outputs on the falling edge of TCK, and becomes the current instruction.

2.5 JTAG timing

From the JTAG standard, the timing of TCK, TMS, TDI, TDO during instruction scan is shown in Figure 2.9, while the timing during data scan is shown in Figure 2.10. The ATE must provide correct timing of these signals to guarantee a successful test.

There are some features that are the same during instruction scan and test data scan.

(1) TCK provides the clock for the test logic. Usually it has a duty cycle close to 50%. The inputs are applied at the rising edge of TCK, while outputs are in response to the falling edge of TCK.

(2) The signal applied at TMS is decoded by the TAP controller to control test operation. The signal at TMS is sampled on the rising edge of TCK.

(3) TDI is the serial input of JTAG instructions and test data. The signal at TDI is sampled on the rising edge of TCK.

(4) TDO is the serial output of the register connected between TDI and TDO. Changes in the state of the signal driven through TDO occur only at the falling edge of TCK. TDO is active only in the SHIFT-DR and SHIFT-IR states. Otherwise, the TDO port is in an inactive state, which is floating high.

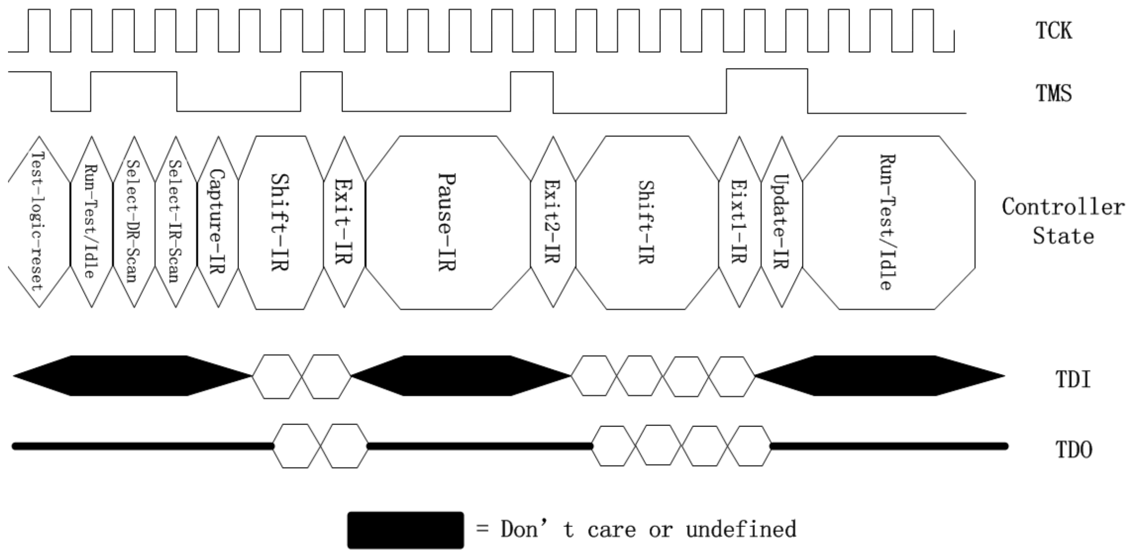


Figure 2.9: Test logic operation: instruction scan

Figure 2.9 illustrates the timing and controller states during instruction scan. To shift in an instruction, the TAP goes through controller states: TLR, RTI, SELECT-DR-SCAN, SELECT-IR-SCAN, CAPTURE-IR, SHIFT-IR. The instruction code is applied at TDI during the SHIFT-IR state.

After the TAP controller leaves SHIFT-IR and enters EXIT1-IR, it can still go back to SHIFT-IR by going through controller states: EXIT1-IR, PAUSE-IR, EXIT2-IR, SHIFT-IR.

Notice that the instruction code applied at TDI has no effect on the test logic in any controller states besides SHIFT-IR. Only when the controller state enters UPDATE-IR, will the instruction be updated to new one at the parallel outputs of IR.

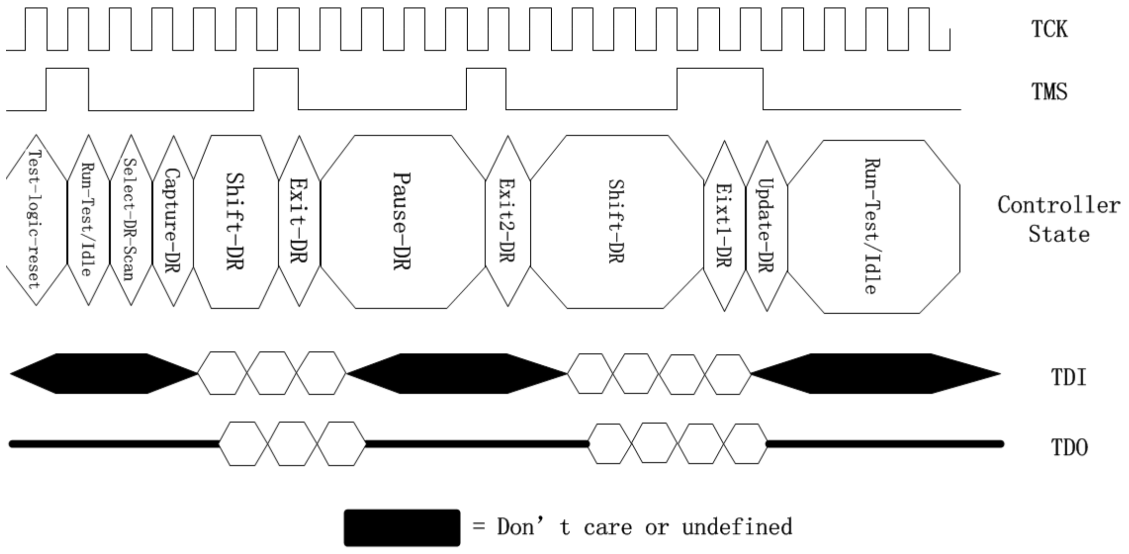


Figure 2.10: Test logic operation: test data scan

Figure 2.10 illustrates the timing and controller states during test data scan. The sequence of TAP controller states is similar to Figure 2.9.

These scan patterns are applied to every operation of JTAG test logic. From these, we can design the timing of test signals on the ATE.

(1) Loading a JTAG instruction is the first step before any test operation. Only after an instruction is loaded into IR, can we perform the test during data scan. In these two

steps, it is important to control the sequence of TMS signal values, in order to force the TAP controller go through several states as illustrated in Figure 2.9 and Figure 2.10.

(2) When the TAP controller is in the SHIFT-DR state for a certain instruction, during every test cycle on the ATE, we should apply test stimulus before the rising edge of TCK, and probe the test result after the falling edge of TCK. This is important to capture and examine the expected results in the ATE test program.

Therefore, several high-level ATE functions are needed to work with the JTAG TAP. The four primary functions are as follows.

- Reset TAP controller to an initial state;
- Send an instruction to the IR;
- Send data to the selected TDR and receive the response;
- Read the IDCODE.

Implementation of these functions on the ATE will be shown in Chapter 3.

Chapter 3

Test Equipment

Auburn University has an Advantest T2000GS automatic test equipment (ATE), housed in the department of Electrical and Computer Engineering. It can perform functional and parametric tests on digital, analog, or mixed signal devices of up to 128 pins.

A XILINX xc3s50a vq100 FPGA device, mounted on a Mercury development board, is used here as an example to illustrate how the ATE works.

3.1 ATE hardware

The ATE mainly consists of three units: mainframe, user interface and test head.

- Mainframe:

  The mainframe comprises a system controller and a site controller. The system controller provides a user GUI to develop and store test plans and patterns, and send commands to the site controller. The site controller executes test plans on the DUT. Each site controller tests one DUT. Our T2000GS model only has a single site controller installed. In addition, the system has various tools, such as oscilloscope and logic analyzer to monitor electrical signals at each DUT pin.

- User interface:

  The user interface consists of a keyboard, mouse and monitor, enabling to write test programs, execute test plans and read the output responses. The operating system is Windows XP.

- Test head:

A test head contains multiple modules to conduct different test operations, such as functional test, parametric test and so on. The modules in our system include a 500mA Device Power Supply module (DPS500mA) and a 250MHz Digital module (250MDM). The latter has the ability to send signals to, and receive signals from up to 128 pins on the DUT, and measure the voltage or current at specified pins. The modules are connected to a test fixture, into which a DUT is inserted.

Figure 3.1 shows the hardware connection between the DUT (FPGA module) and the ATE for our experiments.

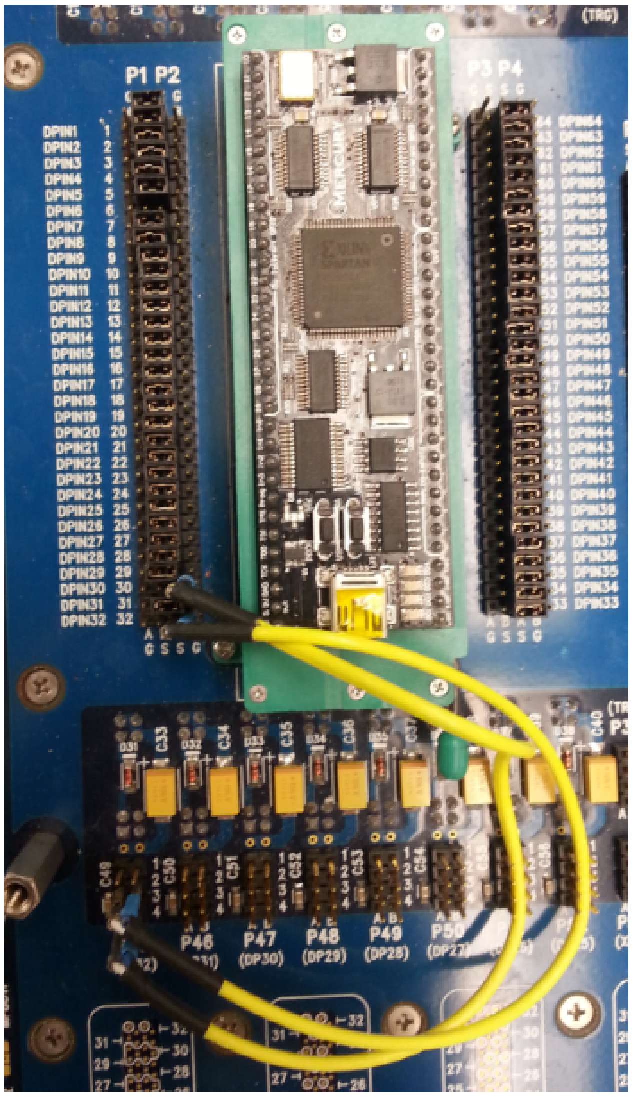


Figure 3.1: Hardware connection between DUT and ATE

As shown in Figure 3.2, wires are used to connect the DUT power pin to DPS500mA module. The two wires are coaxial cables that connect the power pins to the DPS500mA module, which is programmed to provide required supply voltage levels.

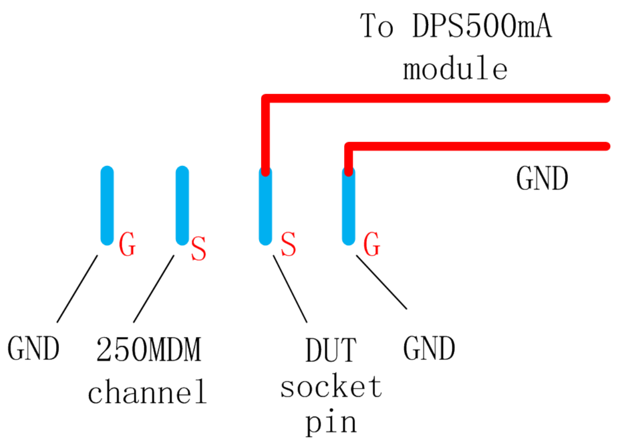


Figure 3.2: DUT power pin connects to DPS 500mA module [19]

For each channel, there are four pins labelled as ‘G’ or ‘S’. ‘G’ stands for Ground, while ‘S’ stands for Source. The S pin next to the DUT is connected to the corresponding DUT pin, while the other ‘S’ pin, farther from the DUT, is connected to the 250MDM module.

As shown in figure 3.3, a shorting plug is inserted between the two ‘S’ pins to connect a DUT I/O pin to a channel of the 250MDM module.

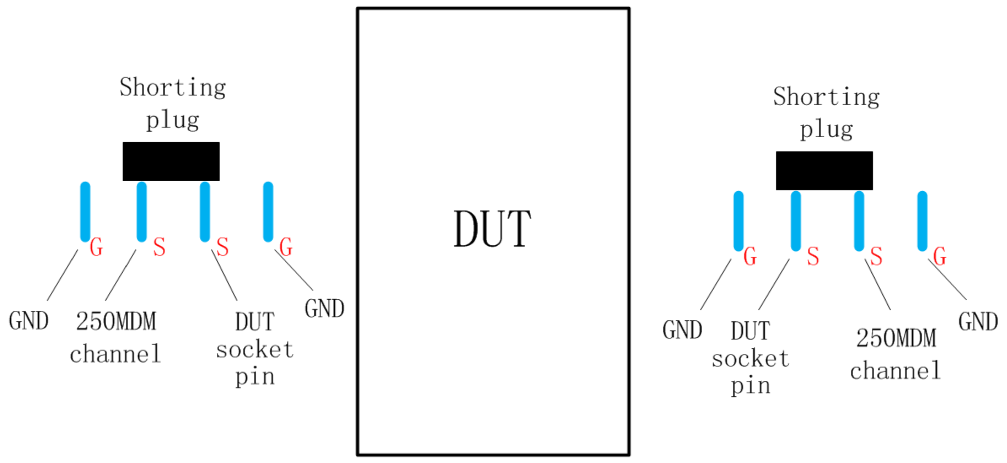


Figure 3.3: DUT I/O pin connects to 250MDM module

3.2 Test program

To conduct a functional test, a test program, based on the OPENSTAR Test Programming Language (OTPL) [7], which is a modular programming language based on C++, needs to be designed for the specific DUT and the on-chip system logic.

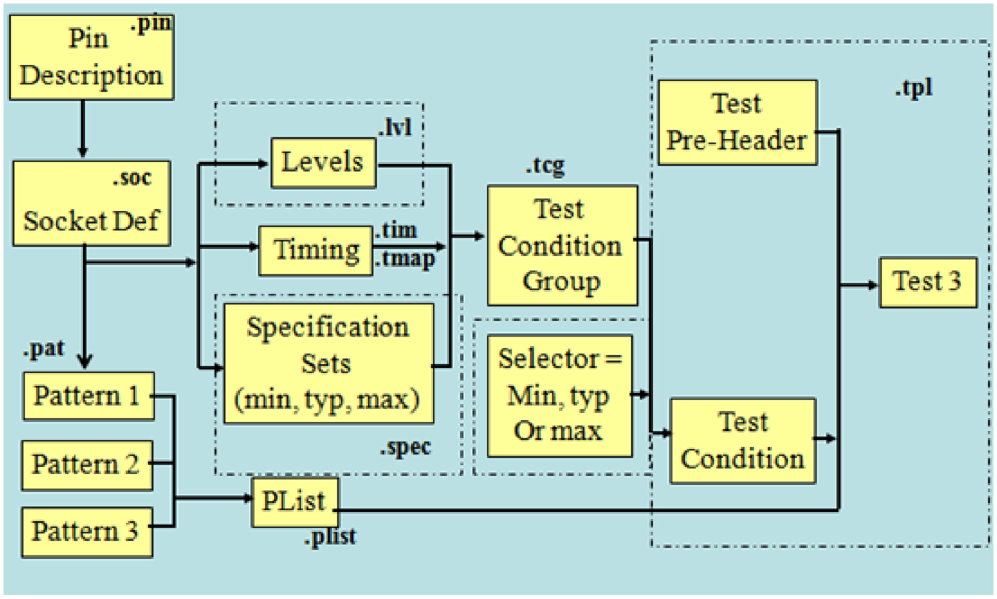


Figure 3.4: Test program structure

A test program comprises the files shown in Figure 3.4. A concise description of each file is provided below.

(1) The pin description file (*.pin) defines the user I/O and power pins available on the DUT. For example,

```
PinDescription {
        Resource AT.Digital.dpin {
                TMS; TDI; TDO; TCK;
                LED_3; LED_2; LED_1; LED_0;
        }
        Resource AT.PowerSupply.dps500mA{
                POWER_5V;
```

```
          }
          Group JTAG_allpins{
                    TMS, TDI, TDO, TCK
          }
}
```

The I/O pin names are defined in the resource file *AT.Digital.dpin*, which is provided for the 250MDM module. The power pin is defined in the resource file *AT.PowerSupply.dps500mA*, which is provided for the DPS500mA module. The variables are the names of the pins that are used in the test.

We can also group some pins in one *Group* for convenience in other files. The variable *JTAG_allpins* stands for the four JTAG pins TMS, TDI, TDO and TCK.

(2) The socket definition file (*.soc) maps DUT pins to pins on the ATE connectors. The *dpin* resource for the 250MDM is provided for user I/O pins, while the *dps500mA* resource for the DPS500mA is provided for power pins. For example:

```
Resource AT.Digital.dpin {
          TMS                 1003.26; # pin 26 on connector 1003
          TDI                 1003.27;
          TDO                 1003.28;
          TCK                 1003.29;
}
Resource AT.PowerSupply.dps500mA{
          POWER_5V            2010.32; # pin 32 on connector 2010
}
```

These DUT pin names are defined in the *.pin file. In the socket file, I/O pins are mapped to channels of the 250MDM module, while power pins are mapped to channels of the DPS500mA. In the above example, *TMS* is connected to 250MDM connector.channel *1003.26*, while *POWER_5V* is connected to DPS500mA connector.channel *2010.32*.

(3) The specification set file (*.spec) defines some device specifications, such as the voltage values for power pins, and voltages VIH, VIL, VOH, and VOL for device I/O pins. These are derived from system parameters found in the XILINX FPGA datasheet [8].

Up to three different values (minimum, typical, maximum) can be defined for each parameter. For example,

```
SpecificationSet functional_Specs(min,typ,max){
        Voltage vih = 1.8V, 2.0V, 2.2V;
        Voltage vil = 0.6V, 0.8V;
        Voltage voh = 2.4V, 2.4V, 2.6V;
        Voltage vol = 0.4V;
}
```

In the example, the *min* value of *vil* is 0.6V, both *typ* and *max* values of *vil* are 0.8V. For *vol*, *min*, *typ*, and *max* are all 0.4V.

(4) The Levels file (*.lvl) defines the sequence of operations to connect DUT pins to ATE connectors by defining *CLOSE* or *OPEN* states of module relays. For example:

```
JTAG_allpins {
        VIH = vih;
        VIL = vil;
        VOH = voh;
        VOL = vol;
        PowerSequence = ON;
        PinOutRelay = CLOSE;
}
```

The pin group *JTAG_allpins* is defined in the *.pin file. An advantage of using *Group* in the *.pin file is that we can apply these parameters to all the pins (TDI, TDO, TCK, TMS) of a group by using just one block.

The values to be assigned to the parameters VIH, VIL, VOH, VOL are provided in the *.spec file. After closing the relay and turning on the power sequence, the electrical signals begin to be applied to input pins and received from output pins.

(5) The timing file (*.tim) sets up the timing edges for input waveforms and defines when to probe the outputs. For example, if we define TCK=11, TMS=10, TDO=LH in the *.pat file, the timing waveforms would be as shown in Figure 3.5.

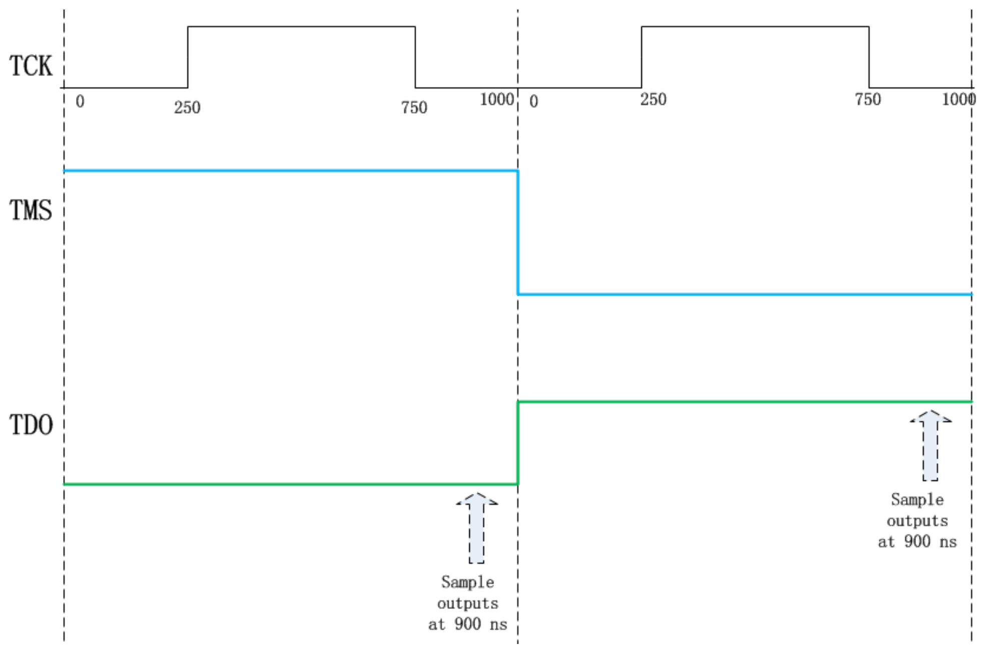


Figure 3.5: Timing waveforms of an example

The following defines the timing cycle to be 1000 ns, and the timing of signals on all pins is defined within that timing cycle. *rate0* is a variable name.

```
PeriodTable{
        Period rate0 {1000nS;}
}
```

The following waveform table definition means that if a '1' is applied to pin TCK in a test pattern, a rising edge on pin TCK is provided at time 250 ns within the test cycle, kept high until 750 ns, and at that time a falling edge occurs and TCK remains low until the end of this timing cycle. *wft1* is a variable name.

```
Pin TCK{
        WaveformTable wft1{
                { 1 {U@250nS; D@750nS;}}
                { 0 {D@0nS;}}
        }
}
```

The following waveform table definition is used to provide a high level signal on pin TMS at time 0 ns within the test cycle if the bit indicating TMS in the pattern is ‘1’, and a low level signal if it is ‘0’. *wft1* is a variable name.

```
Pin TMS{
        WaveformTable wft1{
                { 1 { U@0nS; } }
                { 0 { D@0nS; } }
        }
}
```

The following waveform table definition is used to probe the signal at pin TDO at time 900 ns within the test cycle. In the expected result patterns, ‘H’ means a high level signal is expected, ‘L’ means a low level signal is expected, and ‘X’ means do not care, i.e. the value is ignored by the tester. If the detected signal patterns are the same as the expected values, the test passes. Otherwise, the test fails. *wft1* is a variable name.

```
Pin TDO{
        WaveformTable wft1{
                {H{Z@0nS;H@900nS;}}
                {L{Z@0nS;L@900nS;}}
                { X { Z@0nS; } }
        }
}
```

(6) The timing map file (*.tmap) combines the timing edge settings and period for pins or groups of pins. For example, consider the following map

```
WaveformMap{
        PinFormat{TMS, TCK, TDI, TDO}
        wfs1,rate0,{wft1, wft1, wft1, wft1}
}
```

The variable *wfs1* (waveform set1) indicates the timing for pins TMS, TCK, TDI, and TDO, with test cycle period *rate0* and the timing of each pin defined in waveform table *wft1*. The values for *rate0* and *wft1* are defined in the *.tim file.

(7) The test condition group file (*.tcg) combines Levels, Timing, DCParametrics and Specification Set objects under one name, to be used in a test plan *.tpl file. For example, consider the following test condition groups, named *mercury_jtag_tcg* and *power_off_tcg*, respectively.

```
TestConditionGroup mercury_jtag_tcg{
        SpecificationSet functional_Specs;
        Levels mercury_jtag_levels;
        Timings{
                Timing=mercury_jtag_timing;
                TimingMap=mercury_jtag_tmap;
        }
}


TestConditionGroup power_off_tcg{
        Levels end_levels;
}
```

mercury_jtag_tcg is used for a functional test, and *power_off_tcg* is used to terminate all the signals after all the tests are done.

(8) The pattern file (*.pat) contains the test patterns to be applied during functional tests. For example, the following defines four patterns.

```
Domain default{
        NOP{V{TCK=1;TMS=1;TDI=0;TDO=X;}
                W{TCK=wfs1; TMS=wfs1; TDI=wfs1; TDO=wfs1;}}
        NOP{V{TCK=1;TMS=1;TDI=0;TDO=X;}}
        NOP{V{TCK=1;TMS=1;TDI=0;TDO=X;}}
        NOP{V{TCK=1;TMS=1;TDI=0;TDO=X;}}
}
```

In each line, *V* means the following information is sequential vector data. *W* indicates the waveform set to be used for the pattern. Waveform set *wfs1*, defined in the *.tmap file, is to be used for the four JTAG pins for all four patterns in this example.

(9) The pin list file (*.plist) combines one or more sets of patterns together under one name. For example,

```
GlobalPList mercury_jtag_plist{
        Pat mercury_jtag_pat;
}
```

Only one pattern list is needed for the test in this thesis. However, several pattern files can be defined in the pattern list file.

(10) The test plan file (*.tpl) organizes the test flow and imports all the test condition and resource files needed for a test (functional test and/or parametric test). For example,

```
TestCondition tc_mercury_jtag_typ{
        TestConditionGroup=mercury_jtag_tcg;
        Selector=typ;
}


Test FunctionalTest tfunc_mercury_jtag_typ{
        PListParam=mercury_jtag_plist;
        TestConditionParam=tc_mercury_jtag_typ;
}
```

The test condition *tc_mercury_jtag_typ* uses *mercury_jtag_tcg* from the *.tcg file and the *typ* (typical) value from the *spec file. A test is conducted, which is indicated by keyword *Test*. The test type is functional test, which is indicated by keyword *FunctionalTest*. During the test, the used test condition is *tc_mercury_jtag_typ* defined above, and the used pattern list is *mercury_jtag_plist* from the *.plist file.

For a specific DUT, the *.pin, *.soc, *.spec and *tcg files would normally remain the same for different tests, because these files only specify the hardware and electrical information about the DUT. The *.lvl, *.tim, *.tmap, *.pat, *.plist and *.tpl files are dependent on the on-chip system logic implemented on the DUT and the type of test to be performed, because different pins may be used as inputs or outputs and different patterns may need to be tested.

The ATE software system provides many useful GUI and tools to facilitate testing, such as Flow Editor, Pattern Editor, Shmoo tool etc.

The Flow Editor allows the user to design and conduct multiple tests sequentially with one test program.

The Pattern Editor allows the user to change the applied test patterns and probed output values in the GUI, rather than having to edit and recompile the pattern files.

The Shmoo tool allows the user to modify the value of variables over a range while executing a functional test. Take the example shown in Figure 3.6. We can use the Shmoo tool to test a chip's maximum operating frequencies at different working voltages. The green and red blocks indicate voltage/frequency pairs for which the test passed and failed, respectively.

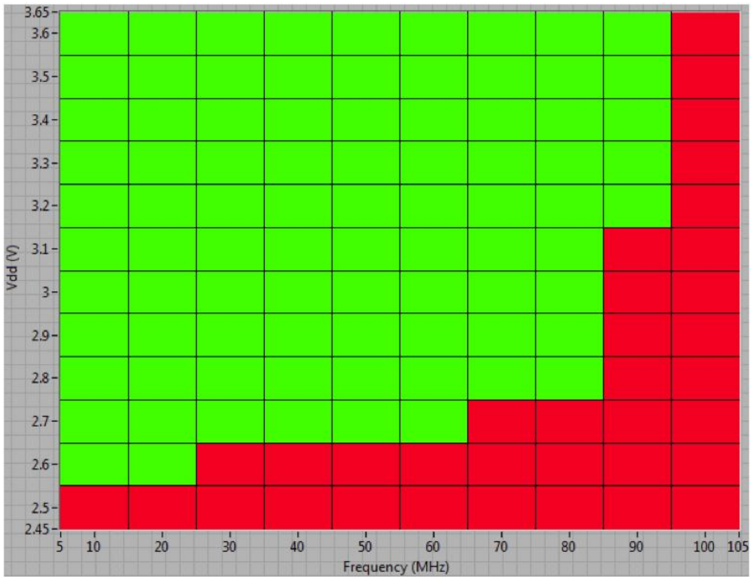


Figure 3.6: Shmoo plot of operating frequencies and working voltages

Chapter 4

Verification of FPGA JTAG interface and components on the T2000GS ATE

In this chapter, a simple two-bit adder is configured as the on-chip logic of an FPGA to illustrate the operation of instructions such as IDCODE, BYPASS, HIGHZ, SAMPLE, and PRELOAD. The adder function is C2C1C0 = A1A0 + B1B0, and each bit stands for a binary value, either 0 or 1.

4.1 Verification of instruction register

Table 4.1 shows some JTAG instructions and their corresponding binary codes for Spartan-3A FPGAs. The length of the IR is 6 bits in the Spartan-3A FPGA.

Table 4.1: JTAG instructions and binary codes of Spartan-3A FPGA

JTAG instruction	binary code
IDCODE	001001
BYPASS	111111
HIGHZ	001010
SAMPLE	000001
PRELOAD	000001
INTEST	000111
EXTEST	001111

The components of the instruction register are shown in Figure 4.1.

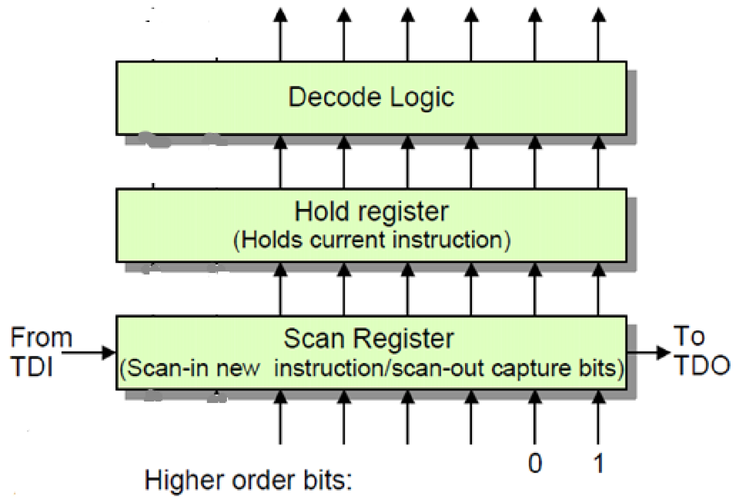

Decode Logic
Hold register
(Holds current instruction)
From TDI
Scan Register
(Scan-in new instruction/scan-out capture bits)
To TDO
Higher order bits:
0
1

Figure 4.1: Components of Instruction Register

The binary code of an instruction is shifted into the scan register through TDI, and is held in the IR. Because the least significant bit (LSB) is shifted in through TDI first, the binary code should be loaded into IR LSB first. For example, to load the INTEST instruction, 000111, into IR, we need to apply vector 111000 on TDI.

If the number of cycles in the SHIFT-IR state is more than the length of IR, in this case 6 bits, the last 6 bits shifted into IR are taken as the instruction. For example, if we apply TDI=010111111 in the SHIFT-IR state, the instruction BYPASS (binary code is 111111) is loaded, and the first three bits, 010, are shifted out of TDO.

When the TAP controller state enters UPDATE-IR, the scan register bits are latched into the hold register in parallel as the current instruction.

Along with the TAP controller state, which is decided by TMS and TCK signals, the decode logic of the IR will generate several internal control signals: ClockDR, ShiftDR, UpdateDR, Reset, Select, ClockIR, ShiftIR, UpdateIR and Enable, as shown in Figure 4.2.

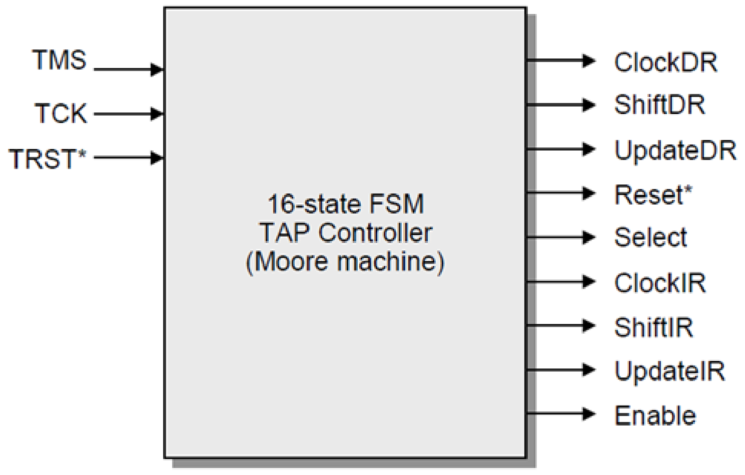


Figure 4.2: Internal control signals of JTAG

These nine signals are distributed to each register and multiplexer to control the behavior of JTAG logic. For different instructions at different states, these internal control signals are different. The correct operation of the test logic is maintained through these internal control signals.

Part of the test pattern file is shown as follow. It is used to test the integrity of instruction register, and the waveform is shown in Figure 4.3.

```
# '#' is used for single line comment.
# at first, the TAP controller state is TEST-LOGIC-RESET
NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}  # TMS=0, enter RUN-TEST/IDLE state.
NOP{V{TCK=1;TMS=1;TDI=0;TDO=X;}}  # TMS=1, enter SELECT-DR-SCAN state.
NOP{V{TCK=1;TMS=1;TDI=0;TDO=X;}}  # TMS=1, enter SELECT-IR-SCAN state.
# TMS=0, enter CAPTURE-IR state.
# the last two bits of IR is initialized to 01.
NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}
# TMS=0, enter SHIFT-IR state.
# TDI=0 is shifted in IR. TDO=H is used to check the LSB.
NOP{V{TCK=1;TMS=0;TDI=0;TDO=H;}}
# TDO=L is used to check the penultimate bit of initial IR value.
```

```
NOP{V{TCK=1;TMS=0;TDI=1;TDO=L;}}
NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}
NOP{V{TCK=1;TMS=0;TDI=1;TDO=X;}}
NOP{V{TCK=1;TMS=0;TDI=1;TDO=X;}}
NOP{V{TCK=1;TMS=0;TDI=1;TDO=X;}}
# the value of TDI=1 (six TCK cycles ago) is passed to TDO.
# TDO=H is used to check it.
NOP{V{TCK=1;TMS=0;TDI=1;TDO=H;}}
NOP{V{TCK=1;TMS=0;TDI=1;TDO=L;}}
# The last bit of code (TDI=1) is shifted in.
# While TMS=1 is used to leave SHIFT-IR state.
NOP{V{TCK=1;TMS=1;TDI=1;TDO=X;}}
```

Figure 4.3 shows the load and read back of an instruction in the experiment on the ATE.

The little triangles at TDO (at the bottom of the figure) are the ATE probe points. The normal triangle means to check that the signal value is high (H), while the inverted triangle means to check that the signal value is low (L).

If the probe value matches the actual value, the triangle will be green; otherwise it will be red. If they match, it means the DUT functions correctly; if there is any discrepancy, it means there may be some defects in the DUT or erroneous operation by the user. The user can debug their design or test operation based on the specified incorrect bits.

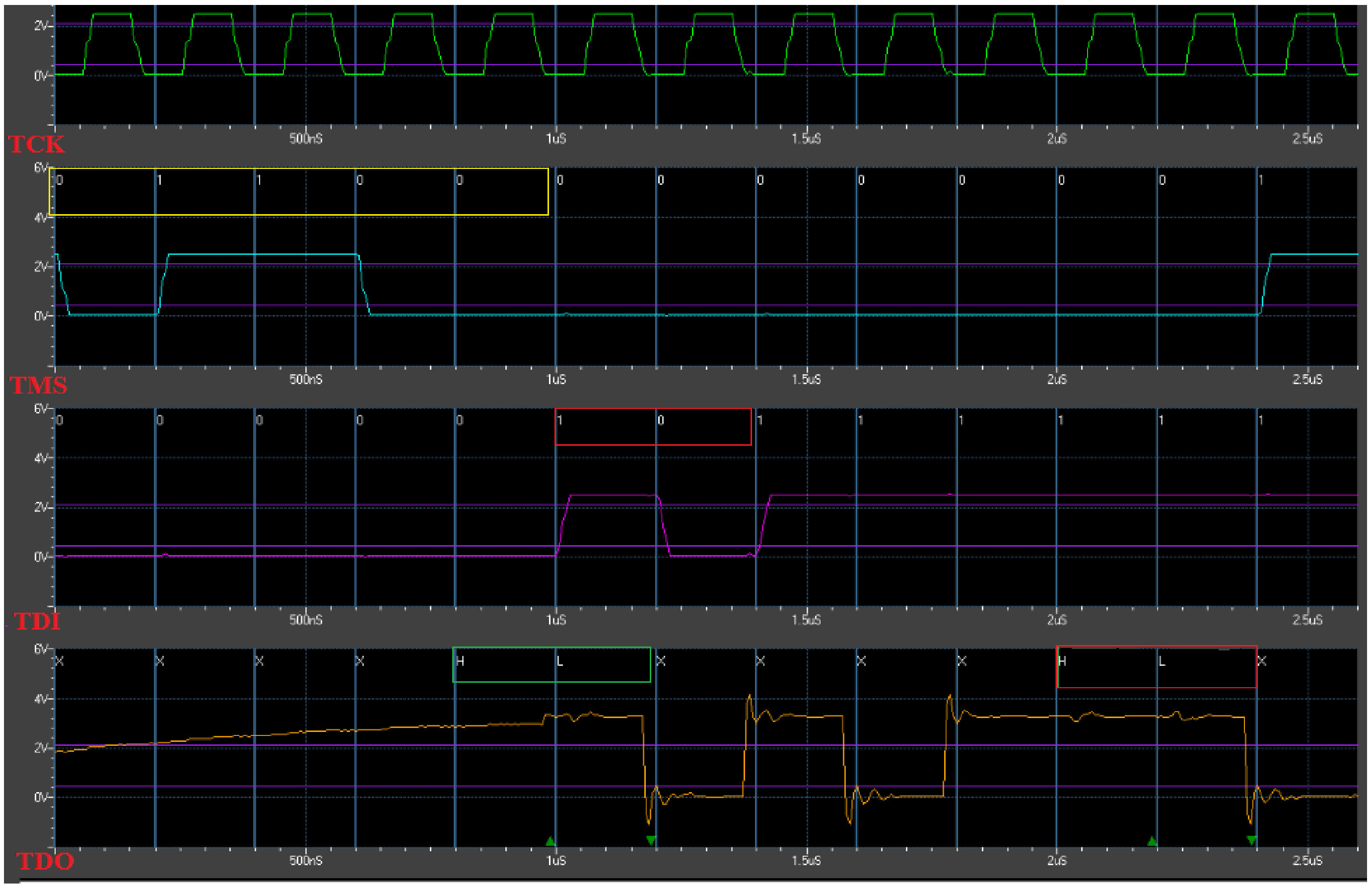


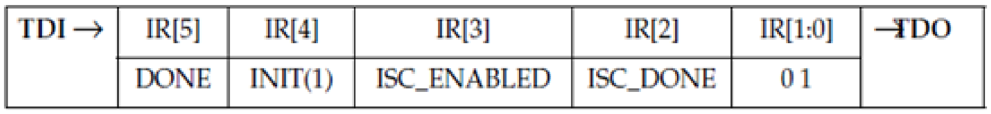
Figure 4.3: The timing of load and read back of instruction

From Figure 4.3, there are some features we can conclude as folows:

(1) The probe is at the falling edge of TCK.

(2) At the beginning, the TAP controller is in Test-Logic-Reset (TLR) state. In figure 4.3, TMS=01100 are applied (shown in the yellow rectangle), and the TAP controller goes into the SHIFT-IR state and remains for several timing cycles.

The 6-bit data shifted out on TDO are the initial value in the Instruction Register (IR), which are loaded automatically when the CAPTURE-IR state is entered. The default value is dependent on the type of FPGA and the situation of FPGA, as shown in Figure 4.4.



TDI →	IR[5]	IR[4]	IR[3]	IR[2]	IR[1:0]	→TDO
	DONE	INIT(1)	ISC_ENABLED	ISC_DONE	0 1	

Figure 4.4: Values captured at CAPTURE-IR state

Notice that the first two bits will be always 10, as the green rectangle shows in Figure 4.3, because in the standard, the last two bits inside the IR are hardwired to b1b0=01.

- ISC_ENABLED indicates whether the in-system configuration is enabled.
- ISC_DONE indicates whether the in-system configuration is done.
- INIT(1) indicates whether the FPGA programming is initialized.
- DONE indicates whether the FPGA programming is done.

(3) The signals at TDO follow the signals at TDI with 6 TCK cycles delay, as the red rectangle shows in Figure 4.3. Although the IR is 6 bits, we can remain in the SHIFT-IR state indefinitely. However, to load an instruction, 6 timing cycles is sufficient. In the earlier example of Figure 2.6 where three FPGAs are chained as one scan path, it would take 18 cycles to load the instruction codes into IRs of the three FPGAs.

If the instruction load and read back are both good, then we can assume the instruction register is good.

4.2 Verification of correct device

To verify the type of device, the IDCODE instruction is used. The IDCODE of this FPGA is 32 bits. By shifting out the IDCODE, we can make sure that the current device is the DUT we want in the test. If the code does not match, then we have the wrong device for the test or other operation.

There are two ways to shift out IDCODE.

(1) Enter the SHIFT-DR state directly without entering the SHIFT-IR state after TLR. The device identification register is connected between TDI and TDO by default. Therefore, IDCODE would be shifted out through TDO in reverse order.

(2) Enter the SHIFT-IR state by applying the binary code of the IDCODE instruction, 001001, at TDI. Then the IDCODE can be shifted out through TDO in the shift-DR state.

As the yellow rectangle shows in Figure 4.5, the value shifted into TDI is 100100, which is the reverse order of the IDCODE instruction. Notice that when the last bit of the IDCODE instruction is shifted in, TMS is set to 1 to enter the EXIT1-IR state at the next timing cycle.

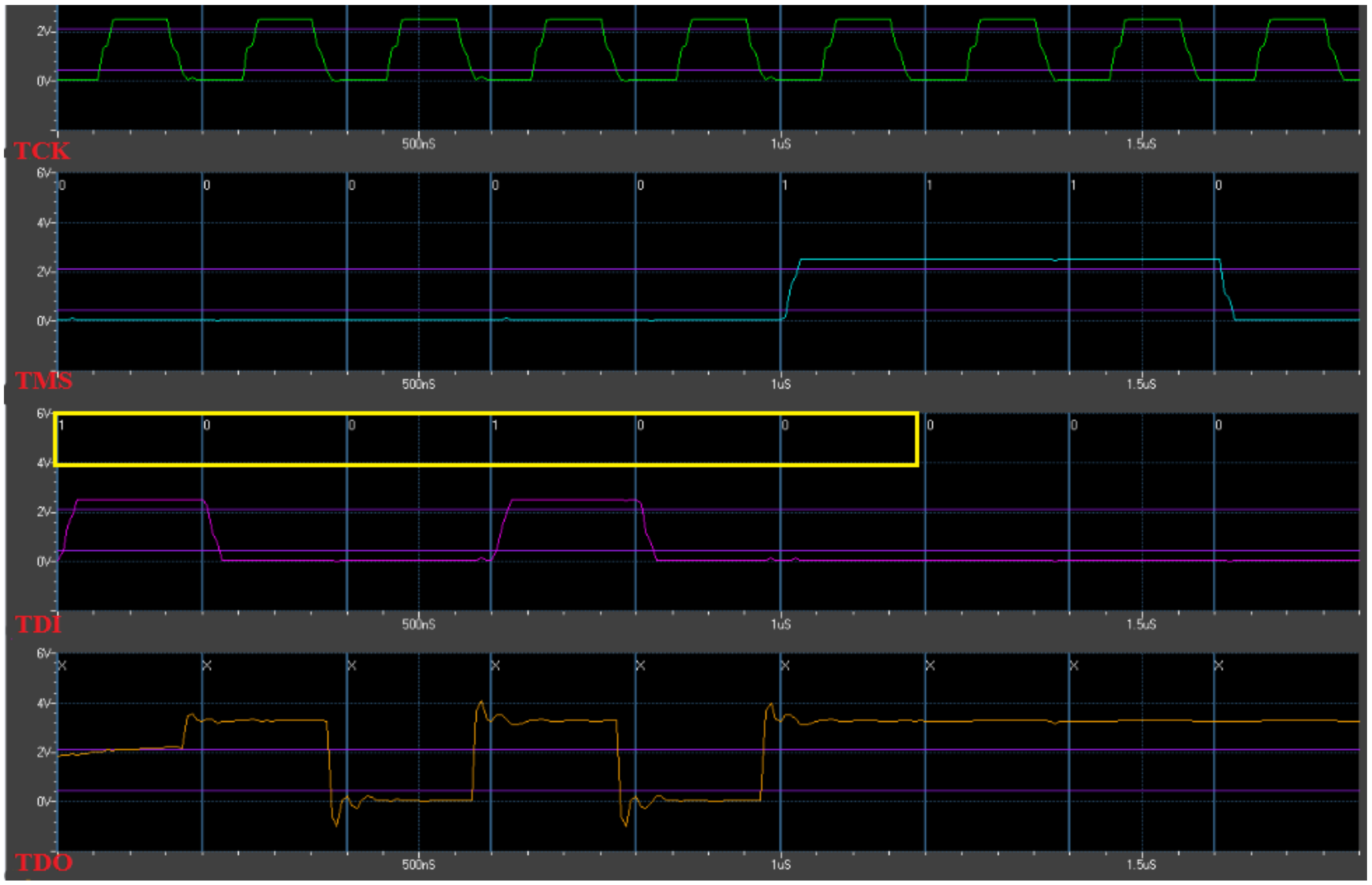


Figure 4.5: IDCODE instruction 001001 is shifted into TDI during SHIFT-IR state

From Figure 4.6, the value shifted out at TDO is HHLLHLLHLLLLLLLLHLLLLH-LLLHLLLLLL.

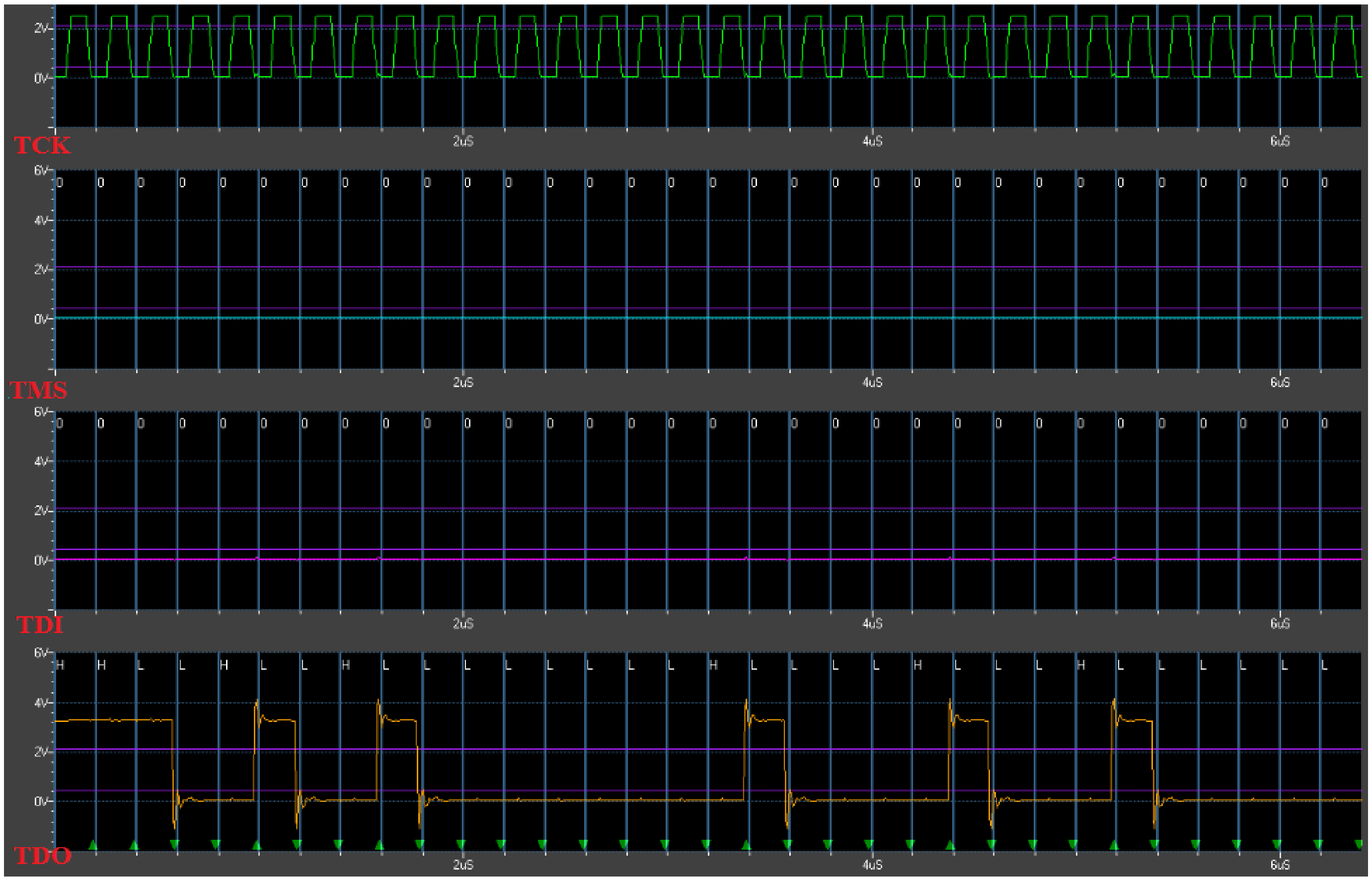


Figure 4.6: IDCODE is shifted out at TDO during SHIFT-DR state

In binary code, this is 11001001000000001000010001000000. From the BSDL file, we know the reverse IDCODE of xc3s50a vq100 FPGA is as shown in Figure 4.7.

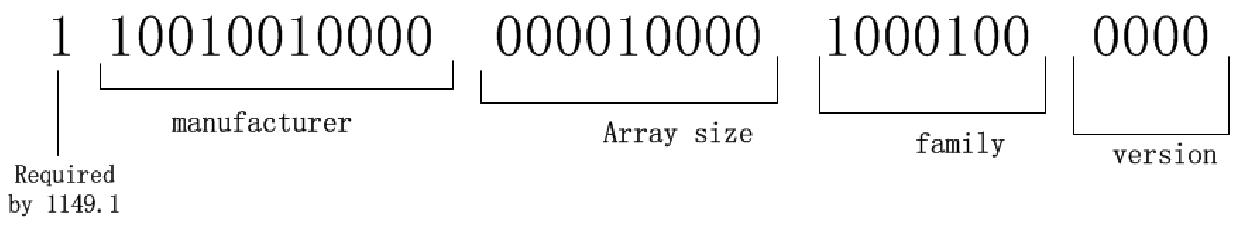


Figure 4.7: Reverse IDCODE of FPGA xc3s50a vq100

Comparing the result to the expected value, which is exactly the reverse order of the FPGA's IDCODE, the ATE will verify that this is the correct FPGA device that should be used for proceeding with the rest of the test.

4.3 Verification of JTAG pins

Since JTAG pins may be faulty, we need to make sure the JTAG pins are free of defects before proceeding with further tests. Consider the faults shown in Figure 4.8.

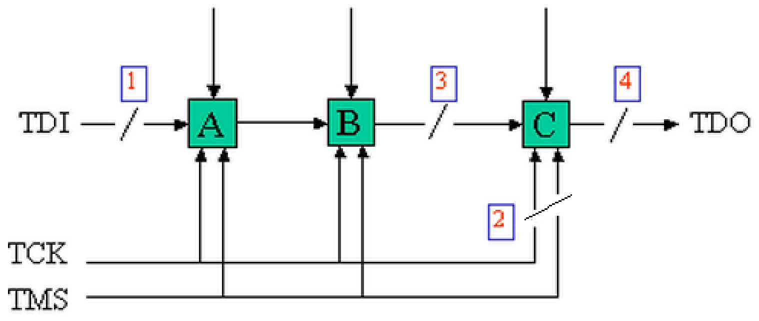


Figure 4.8: JTAG components may have faults in these places [9]

- Fault 1: defect (open circuit) on the TDI pin.

  In this situation, the valid IDCODE can be read out through TDO if the SHIFT-DR state is entered directly. But when any other instructions is applied to TDI in the SHIFT-IR state, the BYPASS instruction (111111) will be loaded, and what is read out from TDO in the shift-DR state is always 1, because the TDI signal will be read as 1 if open circuited.

- Fault 2: defect on TCK or TMS pin

  If there are errors at TCK or TMS pins, no matter whether the error is stuck-at-0 or stuck-at-1, the TDO will not be driven, so the ATE can't probe valid signals on TDO.

  A fault at TCK means the TAP controller clock can't be pulsed. A fault at TMS means the TAP controller can't switch among the useful states. The TAP controller state would be at TLR and the component is in normal function mode all the time. It will never enter test mode.

- Fault 3: defect in a JTAG register

Errors in the connections of the JTAG registers inside the device,(bypass register, boundary-scan register, device identification register, instruction register, etc.) will prevent the corresponding instruction that connects this register between TDI and TDO from operating correctly.

- Fault 4: defect on TDO pin

  Faults on the TDO pin would prevent valid data from being read out from TDO. It will be always high if open circuited for FPGA in general.

There are two ways to check the integrity of JTAG pins.

(1) The FPGA has three supply voltages: VCCO, VCCINT, and VCCAUX. The JTAG components are powered by VCCAUX. By default all JTAG pins of the FPGA connect to VCCAUX through pull-up resistors internally. Therefore, after power up, a test can probe the signals at these four pins to measure their voltage by using the ATE to do the DC parametric tests. For an intact FPGA, all voltages at these JTAG ports should be the same as VCCAUX minus drop across pull-up resistors. If there is any difference at a certain pin, it means the pin is faulty and will make the JTAG functions incorrect.

(2) A test can enter the SHIFT-IR state and shift a data pattern through TDI to TDO. Because the TDO driver is active in the SHIFT-IR state, the values at TDO would follow the values at TDI after some cycles delay (the delay depends on the length of the instruction register), as shown at section 3.2. If the pattern on TDO matches that applied to TDI, we can assume that the JTAG pins are functioning correctly; if not, then either the JTAG pins or the register are faulty.

4.4 Verification of bypass register

Both BYPASS and HIGHZ instructions utilize the bypass register to perform tests.

4.4.1 BYPASS instruction

The BYPASS instruction selects the bypass register to be connected between TDI and TDO. The BYPASS instruction does not perform any system logic operation and has no effect on the on-chip system logic.

Figure 4.9 shows the ATE waveform when the BYPASS instruction is executed. TDO follows TDI after one timing cycle as shown in the green rectangle. The initial value of the bypass register is 0.

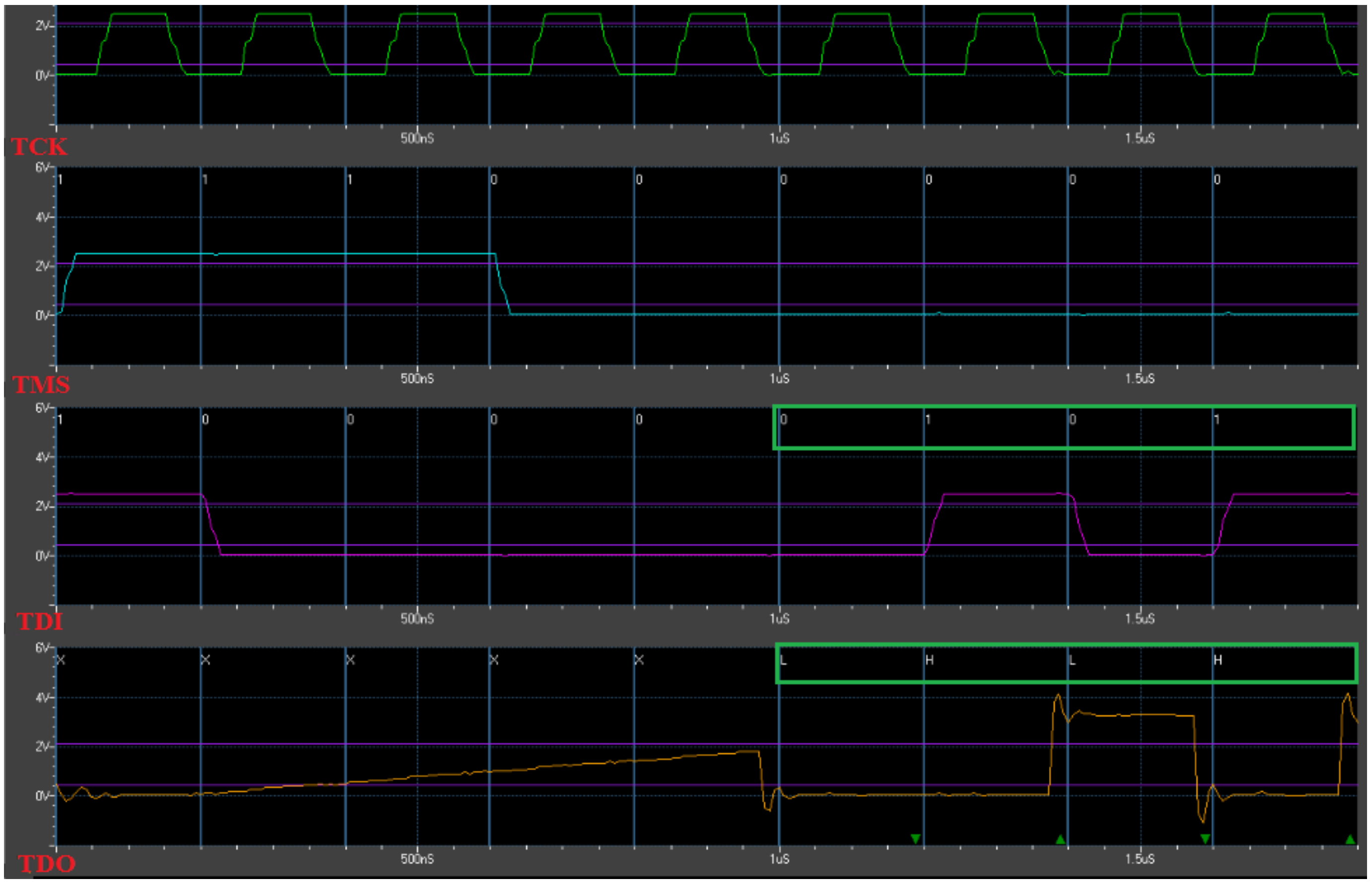
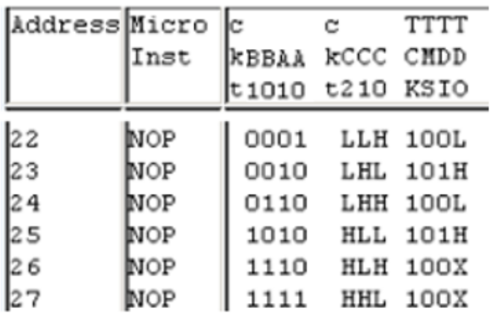


Figure 4.9: Waveform of executing BYPASS instruction

In this test, the FPGA is configured as a simple two-bit adder. The function is C2C1C0 = A1A0 + B1B0. The patterns for a functional test are shown in Figure 4.10, assuming that bypass register is selected between TDI and TDO.



Address	Micro	c	c	TTTT
	Inst	kBBAA	kCCC	CMDD
		t1010	t210	KSIO
22	NOP	0001	LLH	100L
23	NOP	0010	LHL	101H
24	NOP	0110	LHH	100L
25	NOP	1010	HLL	101H
26	NOP	1110	HLH	100X
27	NOP	1111	HHL	100X

Figure 4.10: Patterns of BYPASS instruction

From Figure 4.10, we can see that the expected value at TDO is the same as the value at TDI after one clock cycle. It indicates that bypass register is connected between TDI and TDO. In addition, the system logic also functions correctly. The two-bit adder works normally. For example, when A1A0 + B1B0 is 10+01, the result is C2C1C0 = LHH.

4.4.2 HIGHZ instruction

HIGHZ instruction is similar to the BYPASS instruction. It also connects the bypass register between TDI and TDO. However, HIGHZ also disables the system logic outputs, while for the BYPASS instruction, the on-chip system logic would perform as normal.

As shown in Figure 4.11, the enable signal of tristate multiplexer is from the instruction decoder.

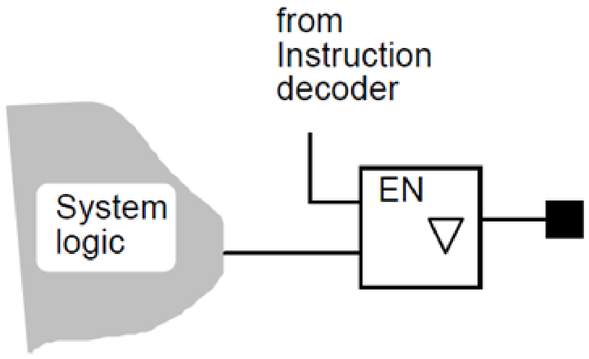
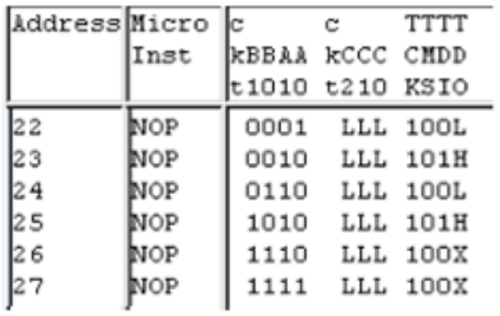


Figure 4.11: Output is in inactive state when executing HIGHZ instruction

From figure 4.12, we can see that the HIGHZ instruction also connected bypass register between TDI and TDO. However, the outputs of on-chip logic C2C1C0 are all ‘L’. When the TDO pin is in inactive state, its value will be floating low.



Address	Micro Inst	c kBBAA t1010	c kCCC t210	TTTT CMDD KSIO
22	NOP	0001	LLL	100L
23	NOP	0010	LLL	101H
24	NOP	0110	LLL	100L
25	NOP	1010	LLL	101H
26	NOP	1110	LLL	100X
27	NOP	1111	LLL	100X

Figure 4.12: Patterns of HIGHZ instruction

4.5 Verification of boundary scan register (BSR)

4.5.1 BSR as one scan path

In order to execute instructions that utilize BSR, firstly we need to make sure that BSR works correctly and is connected between TDI and TDO. To verify the integrity of BSR, we can chain BSR as one scan path and send test data through TDI, and verify the value through TDO in SHIFT-DR state. The following pattern file performs this test.

```
# use five TMS=1 to make sure it's at TLR state.
NOP{V{TCK=1;TMS=1;TDI=X;TDO=X;}
      W{TCK=wfs1; TMS=wfs1; TDI=wfs1; TDO=wfs1;}}
NOP{V{TCK=1;TMS=1;TDI=X;TDO=X;}}
NOP{V{TCK=1;TMS=1;TDI=X;TDO=X;}}
NOP{V{TCK=1;TMS=1;TDI=X;TDO=X;}}
NOP{V{TCK=1;TMS=1;TDI=X;TDO=X;}}
NOP{V{TCK=1;TMS=0;TDI=X;TDO=X;}} # TMS=0, enter RTI state
NOP{V{TCK=1;TMS=1;TDI=X;TDO=X;}} # TMS=1, enter SELECT-DR-SCAN state
NOP{V{TCK=1;TMS=1;TDI=X;TDO=X;}} # TMS=1, enter SELECT-IR-SCAN state
```

```
NOP{V{TCK=1;TMS=0;TDI=X;TDO=X;}} # TMS=0, enter CAPTURE-IR state
NOP{V{TCK=1;TMS=0;TDI=1;TDO=X;}} # TMS=0, enter SHIFT-IR state
NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}
NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}
NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}
NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}

# The SAMPLE/PRELOAD instruction, 000001, is loaded into IR.
# and state enters EXIT1-IR.
NOP{V{TCK=1;TMS=1;TDI=0;TDO=X;}}

NOP{V{TCK=1;TMS=1;TDI=X;TDO=X;}} # TMS=1, enter UPDATE-IR state.
NOP{V{TCK=1;TMS=1;TDI=X;TDO=X;}} # TMS=1, enter SELECT-DR-SCAN state.
NOP{V{TCK=1;TMS=0;TDI=X;TDO=X;}} # TMS=0, enter CAPTURE-DR state.
NOP{V{TCK=1;TMS=0;TDI=X;TDO=X;}} # TMS=0, enter SHIFT-DR state.

# the data loaded into TDI will shifted out at TDO after 373 clock cycles,
# if the BSR is intact and connected as one path.
NOP{V{TCK=1;TMS=0;TDI=1;TDO=X;}}
NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}
NOP{V{TCK=1;TMS=0;TDI=1;TDO=X;}}
NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}
NOP{V{TCK=1;TMS=0;TDI=1;TDO=X;}}
NOP{V{TCK=1;TMS=0;TDI=1;TDO=X;}}
NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}
NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}

# ''IDXI 363'' means to execute the current ATE instruction 364 times
IDXI 363 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
```

```
# verify the values at TDO, which is the same as TDI 373 clock cycles ago.
NOP{V{TCK=1;TMS=0;TDI=X;TDO=H;}}
NOP{V{TCK=1;TMS=0;TDI=X;TDO=L;}}
NOP{V{TCK=1;TMS=0;TDI=X;TDO=H;}}
NOP{V{TCK=1;TMS=0;TDI=X;TDO=L;}}
NOP{V{TCK=1;TMS=0;TDI=X;TDO=H;}}
NOP{V{TCK=1;TMS=0;TDI=X;TDO=H;}}
NOP{V{TCK=1;TMS=0;TDI=X;TDO=L;}}
NOP{V{TCK=1;TMS=0;TDI=X;TDO=L;}}

# exit the ATE test
EXIT{V{TCK=1;TMS=1;TDI=X;TDO=X;}}
```

In the patterns above, we first set TMS=11111 to make sure thr TAP controller is in the TLR state, and then set TMS=0110 to prepare to enter the SHIFT-IR state. In order to chain the BSR as one scan path, we need to use a JTAG instruction that utilizes BSR. Here, SAMPLE/PRELOAD instruction, 000001, is used. After the instruction is loaded, we set TMS=110 to prepare to enter SHIFT-DR state.

After entering SHIFT-DR state, we apply some values, which are 10101100 in the example, at TDI. "IDXI num" is an ATE instruction that repeatedly executes the current instruction *num+1* times. Since the number of BSCs is 373 in the FPGA used in the study, the first value of TDI is passed to TDO after 373 clock cycles.

We can see that the output at TDO is HLHLHHLL, which is exactly the same as the value shifted into TDI 373 clock cycles earlier. Therefore, the BSR is intact and connected between TDI and TDO.

4.5.2 SAMPLE instruction

The SAMPLE instruction allows a snapshot of the normal operation of the component to be taken and examined, while it has no effect on the system logic. The data flow of SAMPLE instruction is as shown in Figure 4.13.

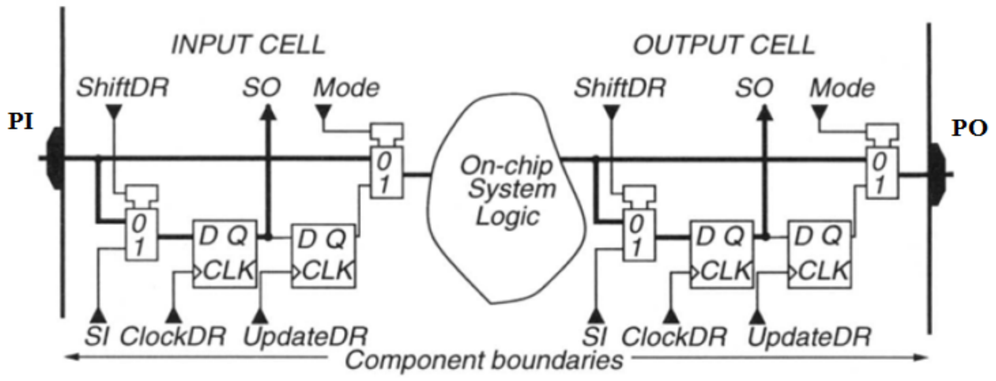


Figure 4.13: Data flow inside JTAG of SAMPLE instruction [18]

Three features of SAMPLE instruction are shown by the bold paths in Figure 4.13:

(1) In the input cell, the PI input pin states are captured to SO, and also forwarded to the on-chip system logic.

(2) In the output cell, the response of the on-chip system logic is captured to SO, and also forwarded to the output pins, PO.

(3) The normal system function is maintained.

The SAMPLE instruction connects the BSR between TDI and TDO. The signal values that are captured in the CAPTURE-DR state at the rising edge of TCK then can be shifted out through TDO.

4.5.3 PRELOAD instruction

The PRELOAD instruction allows data values to be loaded into the BSR before the selection of other instructions that use the BSR data. From the bold path in Figure 4.14, we can see that the data flows from SI (the first SI is TDI) to SO (the last SO is TDO) in each BSC, while the normal function of the on-chip system logic is maintained.

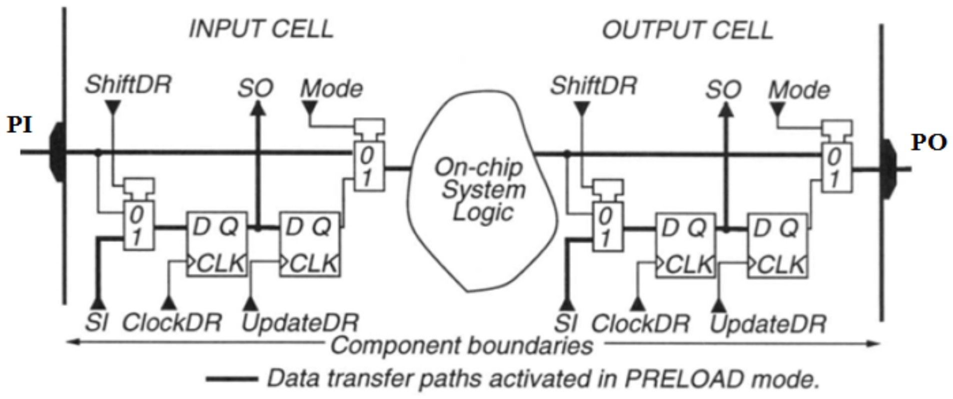


Figure 4.14: Data flow inside JTAG of PRELOAD instruction [18]

Notice that the binary code of the PRELOAD instruction is the same as SAMPLE instruction in this FPGA, which is actually recommended by the JTAG standard. Figure 4.15 shows an example of the steps of executing the SAMPLE/PRELOAD instruction.

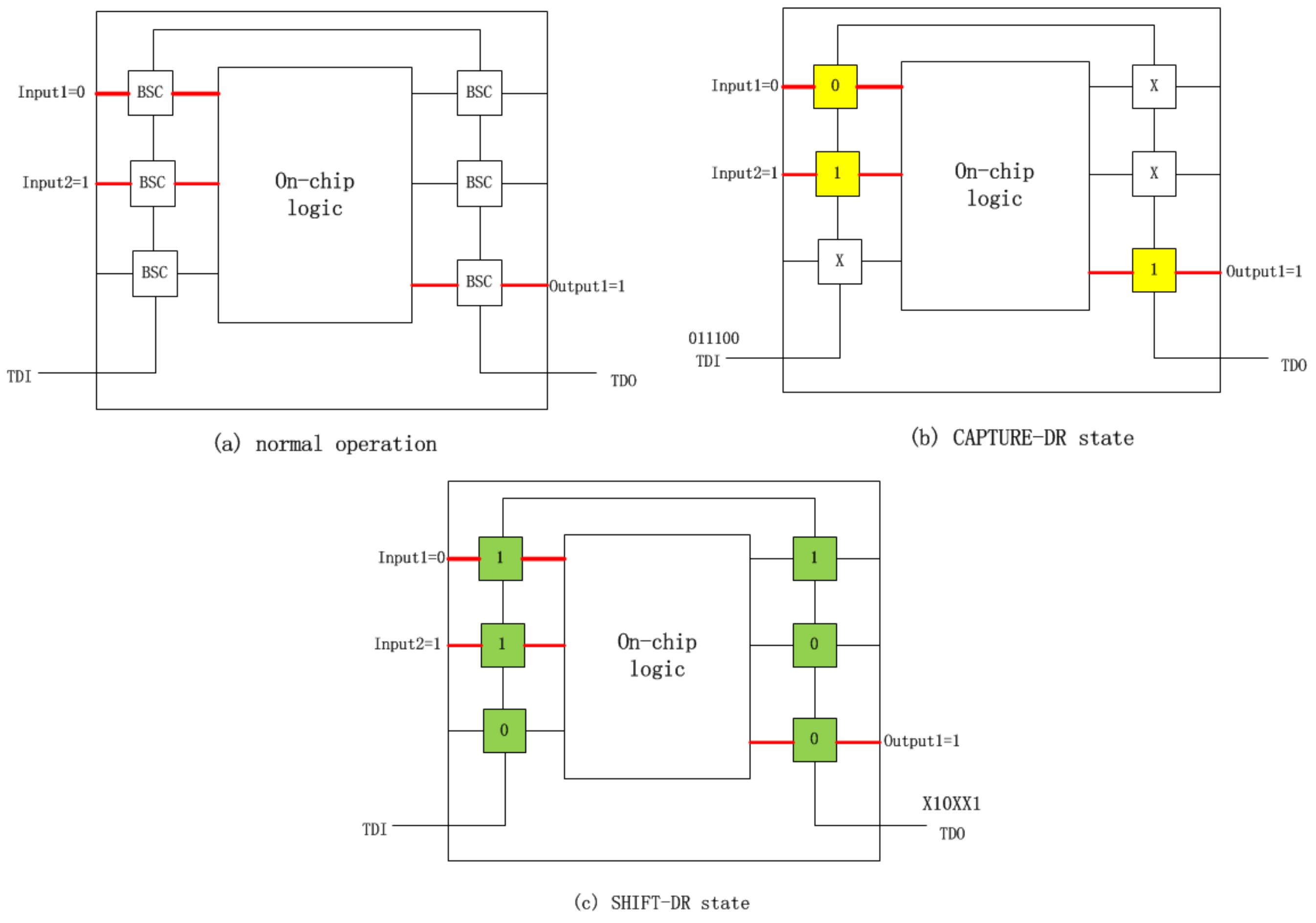


Figure 4.15: Steps of executing SAMPLE/PRELOAD instruction

After the binary code, 000001, is shifted into IR, the normal operation is also maintained.

Firstly, as shown in Figure 4.15(a), during the normal operation, there are two on-chip logic inputs (input1=0, input2=1) and one on-chip logic output (output1=1). The red lines show the data flow.

Secondly, as shown in Figure 4.15(b), when the TAP controller state enters CAPTURE-DR, the data of on-chip logic inputs/outputs are captured into BSC, as shown in the yellow BSC blocks. This is the capture operation of SAMPLE instruction.

Thirdly, as shown in Figure 4.15(c), when the TAP controller state enters SHIFT-DR, the data in BSCs are shifted out to TDO, which is 1XX01X. This is the shift operation of SAMPLE instruction. At the same time, the data, 011100, applied at TDI is shifted into

BSCs through TDI, as shown in the green BSC blocks. Now each BSC holds a value. This is the execution of the PRELOAD instruction.

The operations of SAMPLE and PRELOAD instructions are mutually exclusive. Therefore, they can share the same binary code without causing a conflict.

Chapter 5

Experiment procedures for functional test

This chapter introduces the basic experimental procedures for functional test of an IC. To perform functional test with JTAG, several benchmark circuits of both ISCAS85 [10] (combinational circuits) and ISCAS89 [11] (sequential circuits) are tested. For simplicity in demonstrating the process, the example concerning combinational circuits is c17.v from ISCAS85, while the example concerning sequential circuits is s27.v from ISCAS89.

For this experiment, we want to test circuits that use scan design, so the overall procedures are shown in Figure 5.1. First, the design needs to be converted to scan design if it's sequential, and then FastScan is used to produce the test patterns for both combinational and sequential circuits. Second, we need to implement the circuit and generate the bitstream to be downloaded to the FPGA. Finally, an ATE test pattern generator written in Perl is used to convert the ATPG test patterns to the format of the ATE pattern file.

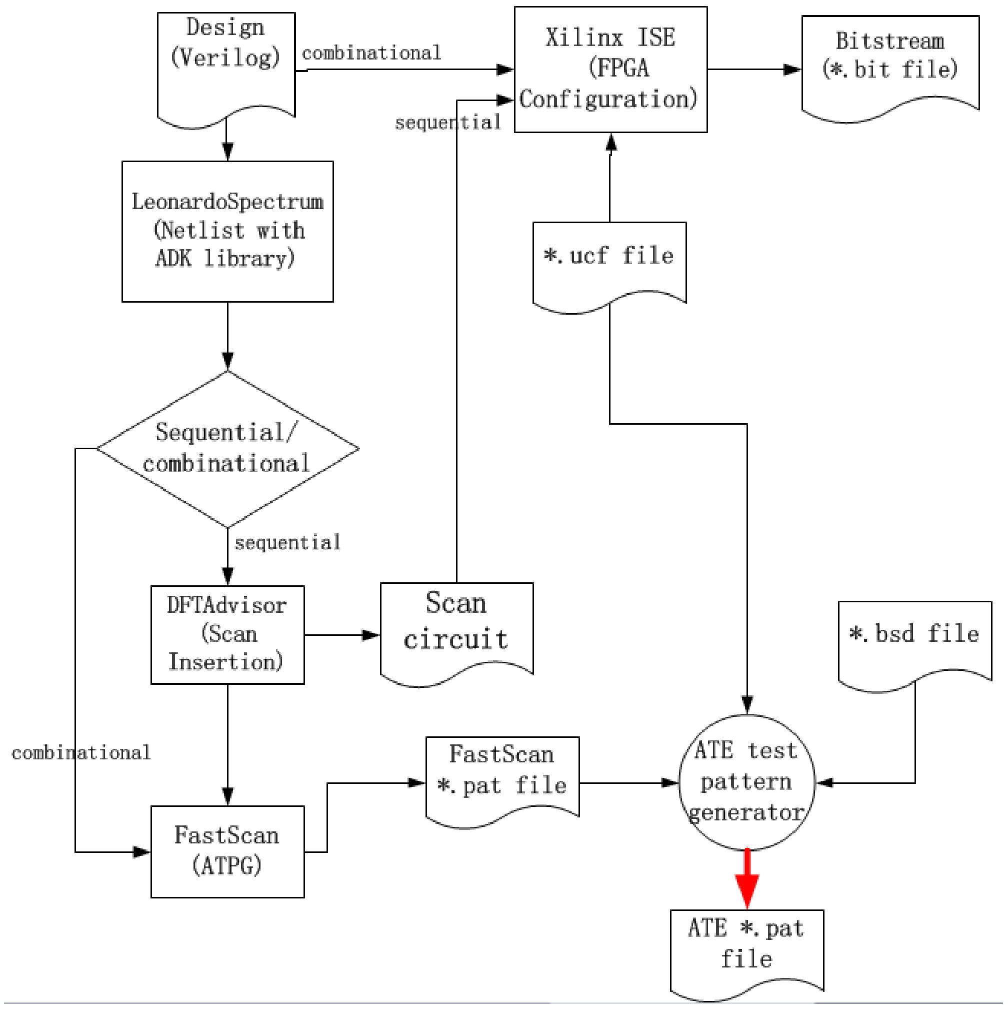


Figure 5.1: Flow of the overall procedures

5.1 FPGA implementation

Before any test of a circuit, firstly we need to configure it for the FPGA. Since the FPGA is from XILINX, ISE WebPACK Design Software 14.2 [12] is used to synthesize the circuit and generate the bitstream to be downloaded to the FPGA.

5.1.1 Convert non-scan design to scan design

The benchmark circuit is written in Verilog as an RTL design. For the combinational circuit, nothing needs to be changed. But for the sequential circuit, we need to convert it to scan design first. Figure 5.2 is a simple example of a non-scan sequential circuit. There are two primary inputs (PI), two primary outputs (PO), and two DFF.

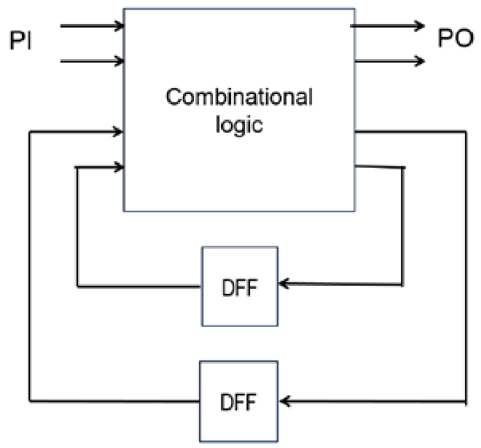


Figure 5.2: Non-scan sequential circuit with two D-flip-flops (DFF)

Firstly, Leonardo Spectrum [13] is used to generate the netlist with ASIC Design Kit (ADK), and DFTAdvisor is used to perform the conversion. The DFTAdvisor manual [14] can be referred for information on working with scan designs.

After the conversion, the non-scan design of Figure 5.2 becomes the scan design of Figure 5.3. Notice that two new inputs are introduced: scan in (SI) and scan enable (SE), and one new output is introduced: scan out (SO). In some case, some SO signals share the same pins with PO signals.

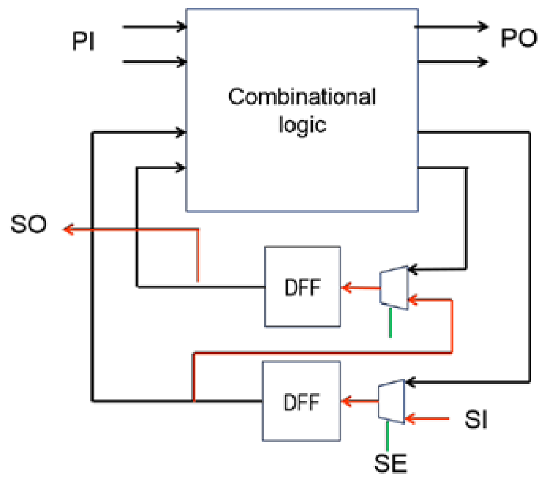
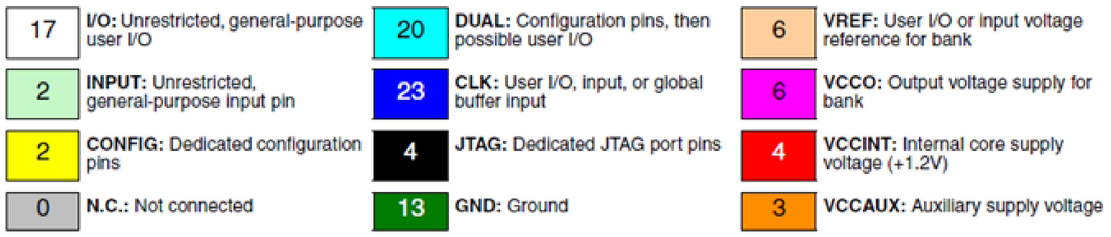


Figure 5.3: Scan sequential circuit with two D-flip-flop (DFF)

The scan-inserted netlist is the Verilog module imported into ISE for the sequential circuit, while the original gate level netlist is used for the combinational benchmark. Both files have the extension *.v. The ISE software does synthesis, mapping, place and route, and then generates the configuration bitstream.

5.1.2 Mapping design inputs/outputs to FPGA pins

After the Verilog design is ready, we need to map the design inputs/output to FPGA pins. This is also done in ISE software. The pins of an FPGA have many types, which function differently. For the xc3s50a vq100 FPGA, the types of pins are shown as Figure 5.4.

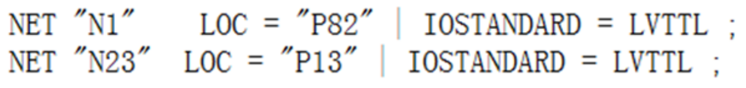


17	**I/O:** Unrestricted, general-purpose user I/O	20	**DUAL:** Configuration pins, then possible user I/O	6	**VREF:** User I/O or input voltage reference for bank
2	**INPUT:** Unrestricted, general-purpose input pin	23	**CLK:** User I/O, input, or global buffer input	6	**VCCO:** Output voltage supply for bank
2	**CONFIG:** Dedicated configuration pins	4	**JTAG:** Dedicated JTAG port pins	4	**VCCINT:** Internal core supply voltage (+1.2V)
0	**N.C.:** Not connected	13	**GND:** Ground	3	**VCCAUX:** Auxiliary supply voltage

Figure 5.4: Pin type of the xc3s50a vq100 FPGA (digits indicate the number of each pin type)

The pins labelled as CONFIG are dedicated to FPGA configuration. The pins labelled as GND, VREF, VCCO, VCCINT are VCCAUX are dedicated power pins. The pins labelled as I/O, DUAL, CLK can be configured as general purpose user I/O, either input or output. The pins labelled as INPUT can be configured only as input, not output.

In XILINX, a Universal Constraint File (UCF) is used to map design inputs/outputs to FPGA pins. The UCF file is specified with the extension *.ucf. An example is shown in Figure 5.5.



```
NET "N1"    LOC = "P82" | IOSTANDARD = LVTTL ;
NET "N23"  LOC = "P13" | IOSTANDARD = LVTTL ;
```

Figure 5.5: Part of the *.ucf file, which shows the mapping information between nets and FPGA pins [8]

NET is a keyword to specify a design's inputs or outputs. *LOC* is a keyword to specify an FPGA pin. Notice that in the UCF file we cannot tell if a pin is input or output. Figure 5.5 shows UCF statements to map the design input/output net *N1* to the FPGA pin *P82*, and net *N23* to pin *P13*.

Notice that the design inputs/outputs can only map to the FPGA pins that can be configured as user I/O. In addition, the design outputs can not be mapped to the FPGA pins labelled as INPUT. Otherwise, errors would occur during the process of bitstream generation.

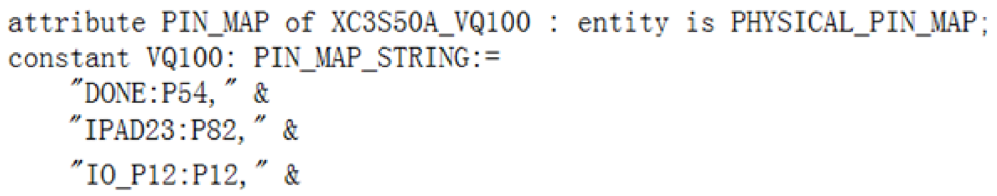
When we have the scan design file *.v and the *.ucf file, ISE performs placement and routing, and then generates a bitstream file *.bit. The ISE manual [15] can be referenced for information on generating bitstreams step by step.

If no error occurs, it means the system logic is successfully implemented on the FPGA. Now, we can design the ATE test for testing.

5.2 Get FPGA information from its BSDL file

BSDL provides a machine-readable means of representing some parts of the information about JTAG in a device. Usually, a vendor of a component that implements the IEEE 1149.1 standard would provide a BSDL description of the component. In this thesis, the DUT is a XILINX xc3s50a vq100 FPGA, so the boundary scan description (BSD) file used is xc3s50a_vq100.bsd [16], which is provided by XILINX.

The BSDL file provides much information, such as the binary code of each JTAG instruction, the type of each FPGA pin, and so on. To perform functional test with INTEST, the most important information is about the BSCs. Figure 5.6 shows the BSDL file information for several pins.



```
attribute PIN_MAP of XC3S50A_VQ100 : entity is PHYSICAL_PIN_MAP;
constant VQ100: PIN_MAP_STRING:=
    "DONE:P54," &
    "IPAD23:P82," &
    "IO_P12:P12," &
```

Figure 5.6: Device package pin mappings of xc3s50a vq100

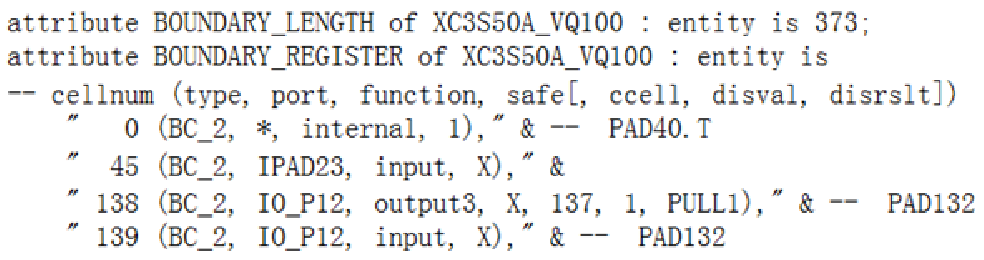
The format of each line is

PIN_MAP : PHYSICAL_PIN_MAP.

PIN_MAP is the name inside the FPGA, while *PHYSICAL_PIN_MAP* is the name of an FPGA pin, which is used for the mapping in the *.ucf file as illustrated in section 5.1.2.

Combining Figure 5.6 and Figure 5.5, we know that P82 maps to IPAD23, which is an input only pin that can only be configured as input. P12 maps to IO_P12, which is a general purpose I/O pin that can be configured as either input or output.

The descriptions of several BSCs are as shown in Figure 5.7.



```
attribute BOUNDARY_LENGTH of XC3S50A_VQ100 : entity is 373;
attribute BOUNDARY_REGISTER of XC3S50A_VQ100 : entity is
-- cellnum (type, port, function, safe[, ccell, disval, disrslt])
    "   0 (BC_2, *, internal, 1)," & --  PAD40.T
    "  45 (BC_2, IPAD23, input, X)," &
    " 138 (BC_2, IO_P12, output3, X, 137, 1, PULL1)," & --  PAD132
    " 139 (BC_2, IO_P12, input, X)," & --  PAD132
```

Figure 5.7: Boundary-scan register description of xc3s50a vq100

From Figure 5.7, we can see that the BOUNDARY_LENGTH is 373, which means the FPGA has a total of 373 BSCs in its BSR. The *cellnum* is the position of each BSC within the BSR. The BSC with *cellnum* 0 is the nearest one to TDO, while the BSC with the highest *cellnum*, 373, is the one nearest to TDI. The *cellnum* of *IPAD23* is 45, which means the BSC at position 45 is connected between FPGA pin 82 and on-chip system logic. This BSC is used for the data load from TDI.

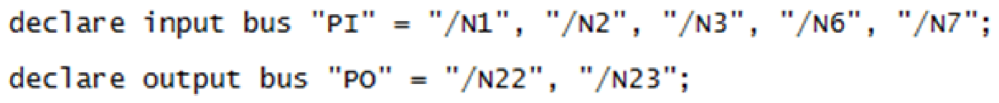
Because IO_P12 is a general purpose I/O pin, two BSCs are involved in the data load and read back. If FPGA pin P12 is configured as input, then BSC at position 139 is connected between FPGA pin 12 and on-chip system logic, and this BSC is used for the data load from TDI. If FPGA pin P12 is configured as output, then the BSC at position 139 is connected between on-chip system logic and FPGA pin 12, and this BSC is used for the data read back through TDO.

5.3 Generate test vectors in FastScan

5.3.1 Combinational benchmark circuits

We import the gate level netlist into FastScan to get the test patterns to detect stuck at faults. The output file has the extension *.pat. The reader can refer to the FastScan manual [17] to determine how to generate test patterns step by step. In the FastScan *.pat file, we can get information such as the test coverage, number of test patterns, inputs, outputs and so on.

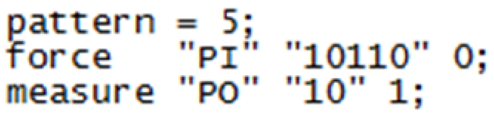
Figure 5.8 shows the name of input and output signals in the pattern file. Notice the order of the signals listed is important, as they correspond to positions of bits in test patterns.



```
declare input bus "PI" = "/N1", "/N2", "/N3", "/N6", "/N7";
declare output bus "PO" = "/N22", "/N23";
```

Figure 5.8: Input signals and output signals of the system logic of combinational benchmark c17.v

Figure 5.9 shows the format of a FastScan test pattern for a combinational circuit.



```
pattern = 5;
force   "PI" "10110" 0;
measure "PO" "10" 1;
```

Figure 5.9: A FastScan test pattern for combinational benchmark c17.v

Combining Figure 5.8 and Figure 5.9, we can determine what value should be applied to the system logic inputs, and what values should be probed at the system logic outputs.

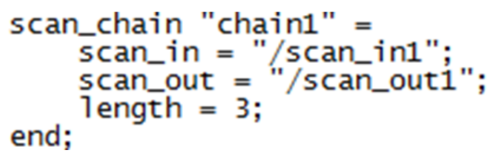
Taking these two figures as example, the *force* statement indicates that value 1 should be applied to input *N1*, value 0 to input *N2*, value 1 to input *N3*, value 1 to input *N6*, and value 0 to input *N7*. The *measure* statement indicates that we should probe primary output *N22* to see if it matches value 1, and *N23* to see if it matches value 0.

In conclusion, the pattern file defines the mapping between on-chip system logic inputs and test data and between on-chip system logic outputs and probe data.

5.3.2 Sequential benchmark circuits

The netlist with inserted scan chains was imported into FastScan to get the test patterns for stuck at faults.

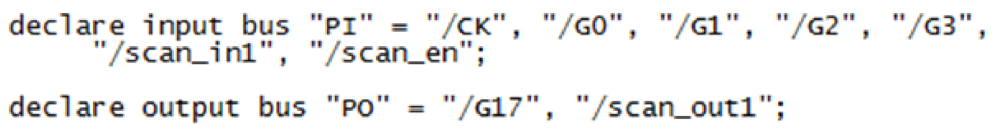
In Figure 5.10, *scan_in* and *scan_out* are the key words in FastScan to specify the scan_in pin and scan_out pin. scan chain *length* is also a key word in FastScan.



```
scan_chain "chain1" =
    scan_in = "/scan_in1";
    scan_out = "/scan_out1";
    length = 3;
end;
```

Figure 5.10: Scan chain input, output and chain length of the scan design of sequential benchmark s27.v

Figure 5.11 defines the input bus and output bus of the scan design of sequential benchmark s27.v.

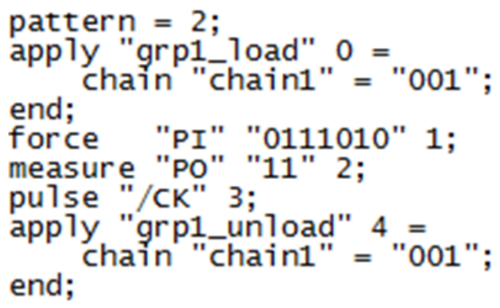


```
declare input bus "PI" = "/CK", "/G0", "/G1", "/G2", "/G3",
     "/scan_in1", "/scan_en";

declare output bus "PO" = "/G17", "/scan_out1";
```

Figure 5.11: Inputs and outputs of the scan design of sequential benchmark s27.v

Figure 5.12 shows the procedure to apply a FastScan test pattern.



```
pattern = 2;
apply "grp1_load" 0 =
    chain "chain1" = "001";
end;
force   "PI" "0111010" 1;
measure "PO" "11" 2;
pulse "/CK" 3;
apply "grp1_unload" 4 =
    chain "chain1" = "001";
end;
```

Figure 5.12: A test pattern of the scan design of sequential benchmark s27.v

The FastScan *.pat file is where the sequential benchmark is more complicated than for the combinational benchmark. To get the ATE test pattern, we need to make clear which signal values we need to apply and probe.

For each FastScan pattern, four steps are required:

(1) load the scan inputs via the scan_in pin;

(2) apply values to all inputs (primary inputs and scan inputs);

(3) measure the primary outputs;

(4) measure the scan outputs from the scan_out pin.

Take Figure 5.12 as an example. We need to:

(1) load value 001 at scan_in serially during the first three clock cycles, with each bit applied before the rising edge of the clock.

(2) Apply signal values 0111010 to all the PI concurrently.

(3) Just after step (2), measure the primary outputs to see if they match the pattern 11.

(4) Measure the scan outputs serially after the rising edge of each clock, to verify that the scan_out signal matches 001, which needs three clock cycles.

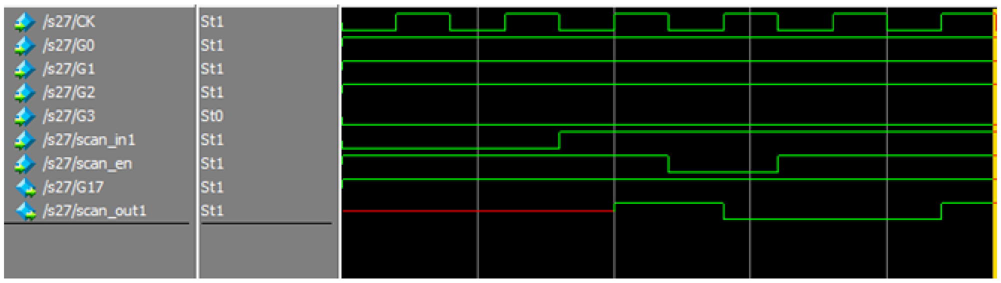


Figure 5.13: Simulation of a test pattern from Figure 5.12 of the scan design of sequential benchmark s27.v in Modelsim

Figure 5.13 gives a simulation example to show the timing to apply one test pattern to the scan design of sequential benchmark circuit s27.v. The test pattern is shown in Figure 5.12 . Before the fourth rising edge of CK, G17=1 and scan_out1=1, which is step (3).

Checking the scan cells (step 4):

- After the fourth rising edge of CK, scan_out1=0;
- After the fifth rising edge of CK, scan_out1 =0;
- After the sixth rising edge of CK, scan_out1=1.

5.4 Create the ATE test program

In the ATE *.pat file, for the input signals, use '0' to represent electrical value less than Input Voltage Threshold VIL, '1' to represent electrical value more than Input Voltage Threshold VOH, and 'X' to represent do not care. For the output signals, use 'L' to represent electrical value less than Output Voltage Threshold VOL, and 'H' to represent electrical value more than Output Voltage Threshold VOH, and 'X' to represent do not care.

An ATE pattern generator program was written in Perl to automatically convert FastScan test patterns to ATE test patterns. The generator takes three files as input files: *.bsd file, *.ucf file, FastScan *.pat file. The *.bsd file was decribed in section 5.2. The *.ucf file was explained in section 5.1, while the FastScan *.pat file was described in section 5.3.

In performing tests with JTAG on the ATE, because the value applied on TDI in shifted in serially, we need to get the information about what test data values should be applied at each BSC position.

Now what the information we have is:

(1) FPGA pins => BSC position, this is from the *.bsd file;

(2) system logic pins => FPGA pins, this is from the *.ucf file;

(3) system logic pins => test data values, this is from FastScan the *.pat file.

A hash data structure is used in the Perl program. From the information contained in the three files listed above, we can get the information about: which FPGA pins are used, and BSC position => test data values.

5.4.1 Combinational benchmark circuits

Figures 5.5- 5.9 illustrate the example. We know that:

(1) N1 => P82; N23 => P12. This is from *.ucf file; (Figure 5.5)

(2) P82 => value 1, and this is an input; P12 => value 0, and this is an output. This is from FastScan *.pat file. (Figure 5.8, Figure 5.9)

(3) P82 => cellnum 45; P12 => cellnum 138. This is from the *.bsd file. (Figure 5.6, Figure 5.7)

Therefore, we know the information that FPGA pins P82 and P12 are used, so *cellnum* 45 should be set to value 1, and *cellnum* 138 should capture value L.

5.4.2 Sequential benchmark circuits

The information from the *.ucf file and *.bsd file is the same as for the combinational benchmark. However, the information from the the FastScan *.pat file is different.

The key point lies on how to implement the system clock signal for the JTAG INTEST instruction. Because the application of stimulus and capture of the response of an INTEST instruction is a single-step operation, each BSC can only hold one certain electrical value at a time. Therefore, the transition of the system clock, such as from state 0 to state 1, can not be applied to a BSC in only one step.

In fact, there are four options defined in the JTAG IEEE 1149.1 standard for different devices. After trying all four options on ATE, we see that the xc3s50a vq100 FPGA implements the d) method defined in the IEEE 1149.1 standard.

- "d) Clock signals can be shifted in via the boundary-scan path in the same manner in which nonclock signals for the on-chip system logic are supplied. Note that this will require the boundary-scan register to be shifted for each distinct clock signal state (e.g., twice for a single-phase clock)." [20]

Figure 5.13 is redrawn in Figure 5.14 to illustrate the idea. For just one test pattern, there are 12 different sets of signal values, as shown in the yellow lines.

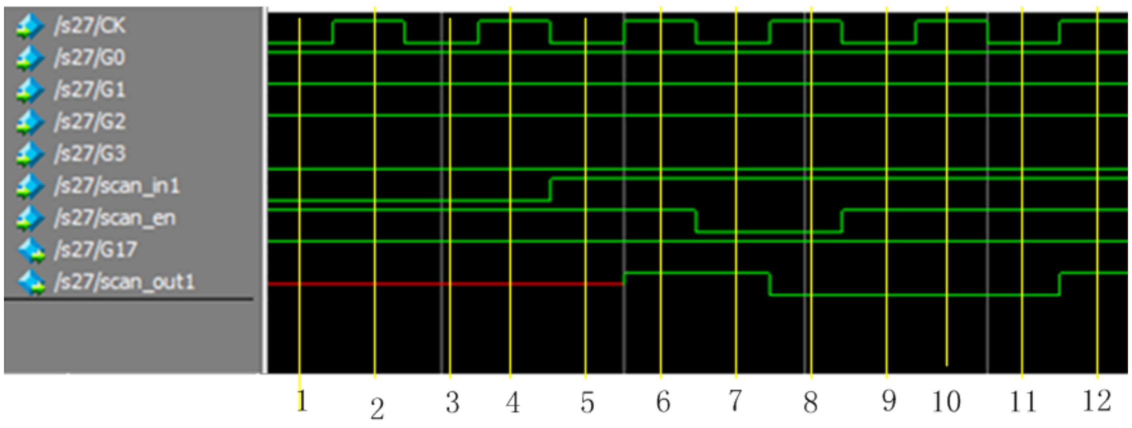


Figure 5.14: One test pattern of Figure 5.12 contains 12 different test vectors

To verify this one pattern, 12 test vectors should be shifted into BSR through TDI in the exact order shown in Figure 5.14, and verified through TDO. The 12 test vectors are as follows.

(CK, G0, G1, G2, G3, scan_in1, scan_en) are inputs, (G17, scan_out1) are outputs.

(1) (0111001) (XX)

(2) (1111001) (XX)

(3) (0111001) (XX)

(4) (1111001) (XX)

(5) (0111011) (XX)

(6) (1111011) (XX)

(7) (0111010) (HH)

(8) (1111010) (XL)

(9) (0111011) (XX)

(10) (1111011) (XL)

(11) (0111011) (XX)

(12) (1111011) (XH)

The results are presented in the next chapter.

Chapter 6

Functional test of the configured circuit with JTAG

6.1 Using the INTEST instruction for functional test

To conduct a functional test through JTAG ports, the INTEST instruction is used, which selects BSR to be connected between TDI and TDO.

The advantage of using JTAG to perform functional test is that there is no need to access input/output pins of the system logic. The ATE can test the system logic just with four JTAG ports, no matter how many inputs/outputs the circuit has, and no matter if these inputs/outputs are accessible.

The disadvantage is the test time, which would increase considerably, since the test data is applied at TDI and examined at TDO by shifting them serially.

There are three features of the INTEST instruction that we can see from Figure 6.1:

(1) The normal functions of on-chip system logic are suspended.

(2) The test inputs are scanned into the BSCs at SI, through TDI, in the first phase.

(3) These inputs are applied by the BSCs to on-chip system logic.

(4) The response of the on-chip system logic captured in BSCs.

(5) These results are scanned out through TDO for verification in the second phase.

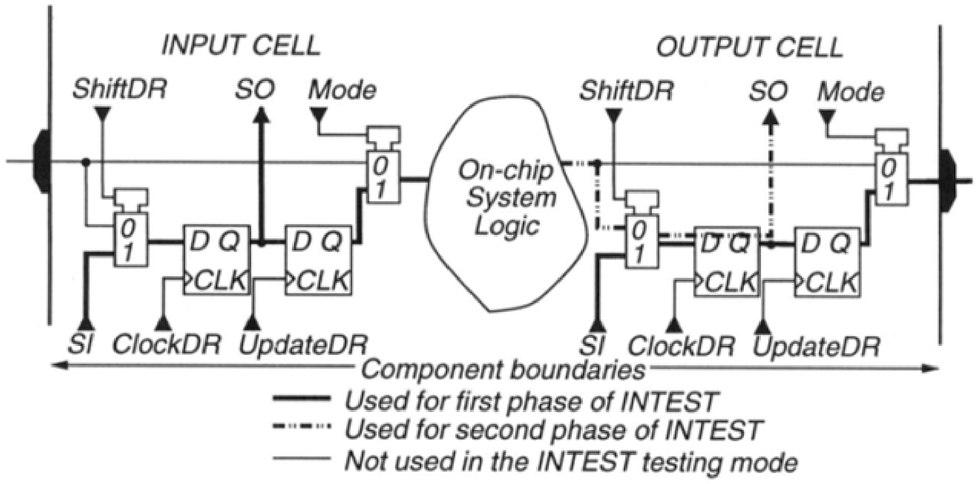


Figure 6.1: Data flow inside JTAG of INTEST instruction

Figure 6.2 shows the operation of each step when executing the INTEST instruction.

(1) During the SHIFT-DR state, test data are shifted into the BSCs.

(2) The TAP controller state goes to UPDATE-DR, and the shifted data are applied to internal logic.

(3) The TAP controller state goes to CAPTURE-DR, and the response of the internal logic is captured in the BSCs.

(4) The TAP controller state goes to SHIFT-DR again, and the test results are shifted out through TDO for verification.

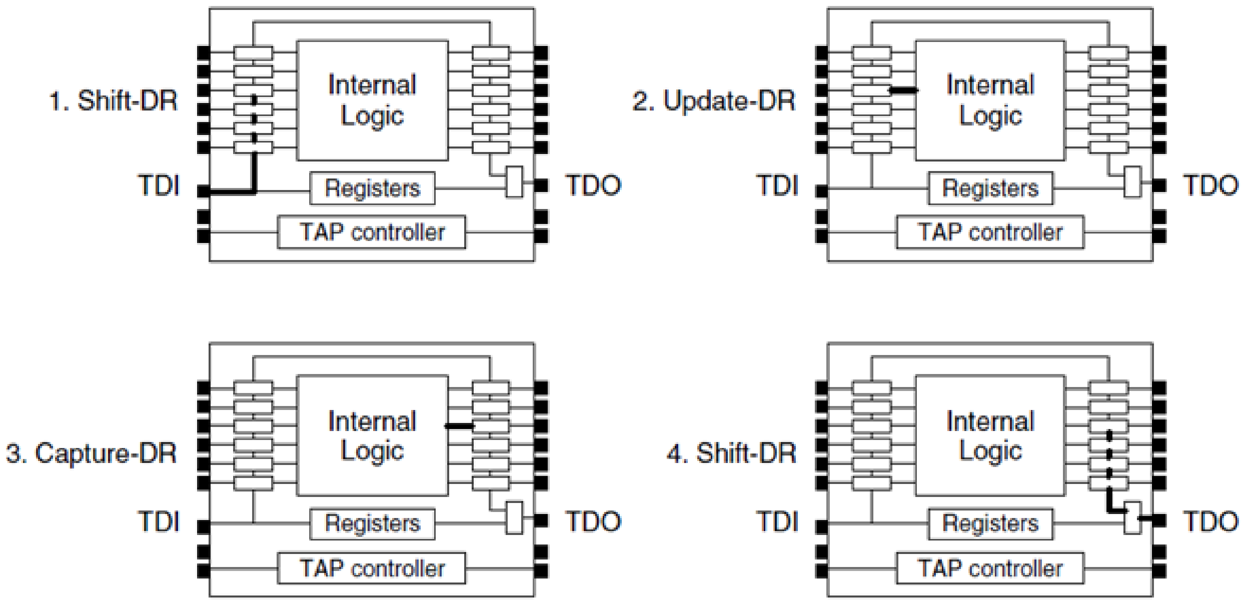
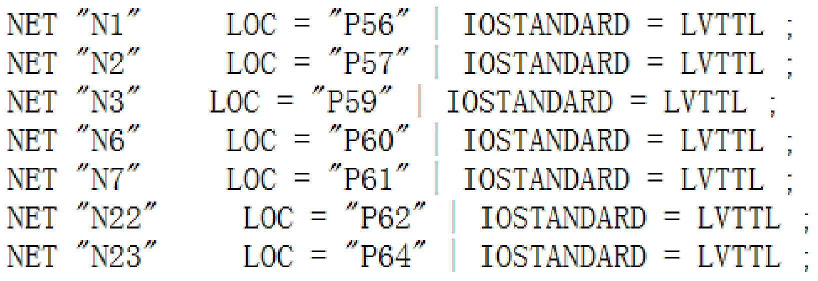


Figure 6.2: Steps when executing INTEST instruction

6.2 Load test vectors to ATE and verification of the result

This section uses some patterns of benchmark c17.v to show the load of test vectors by the ATE and the verification of the result.

Figure 6.3 is the *.ucf file of benchmark c17.v, which shows the mapping of benchmark's inputs/outputs to FPGA pins.



```
NET "N1"     LOC = "P56" | IOSTANDARD = LVTTL ;
NET "N2"     LOC = "P57" | IOSTANDARD = LVTTL ;
NET "N3"    LOC = "P59" | IOSTANDARD = LVTTL ;
NET "N6"     LOC = "P60" | IOSTANDARD = LVTTL ;
NET "N7"     LOC = "P61" | IOSTANDARD = LVTTL ;
NET "N22"     LOC = "P62" | IOSTANDARD = LVTTL ;
NET "N23"     LOC = "P64" | IOSTANDARD = LVTTL ;
```

Figure 6.3: *.ucf file of benchmark c17.v

Figure 6.4 shows part of the FastScan test patterns for benchmark c17.v, which are used as the example to produce the ATE test patterns.

```
SETUP =
    declare input bus "PI" = "/N1", "/N2", "/N3", "/N6", "/N7";
    declare output bus "PO" = "/N22", "/N23";
end;

SCAN_TEST =
    pattern = 0;
    force   "PI" "10001" 0;
    measure "PO" "01" 1;

    pattern = 1;
    force   "PI" "01010" 0;
    measure "PO" "11" 1;
end;
```

Figure 6.4: Part of FastScan *.pat file of benchmark c17.v

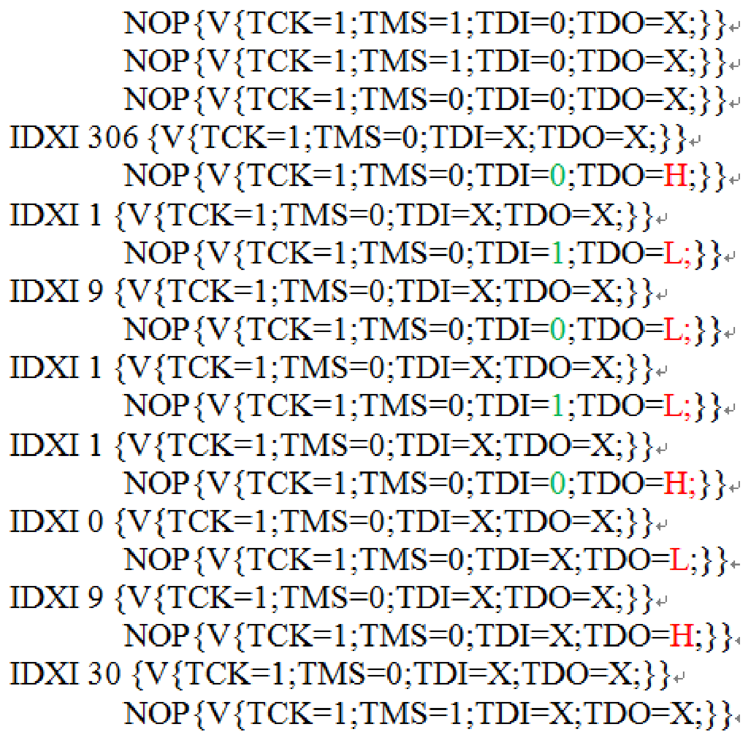
In Figure 6.5, ‘NOP’ is an ATE instruction that means to execute the instruction once. ‘IDXI num’ is an ATE instruction that means to execute the instruction repeatedly for num+1 times.

```
IDXI 306 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
         NOP{V{TCK=1;TMS=0;TDI=1;TDO=X;}}
IDXI 1 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
         NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}
IDXI 9 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
         NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}
IDXI 1 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
         NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}
IDXI 1 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
         NOP{V{TCK=1;TMS=0;TDI=1;TDO=X;}}
IDXI 43 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
         NOP{V{TCK=1;TMS=1;TDI=X;TDO=X;}}
```

Figure 6.5: Load PI of *pattern=0* of Figure 6.4 to ATE

Mixing Figure 6.5 and Figure 6.4,the pattern bits applied to TDI in Figure 6.5, colored red, correspond to *pattern=0* of Figure 6.4 for the primary input (PI=“10001”), according to the cellnum from BSDL file, which is introduced in section 5.2. The first test vector is loaded into BSR through TDI, while no verification is at TDO, so TDO is specified as ’X’ here.

In Figure 6.6, after the first shifting is done, the TAP controller state needs to go through UPDATE-DR, CAPTURE-DR, SHIFT-DR to verify the first test vector, and load the next test vector. This is explained in section 6.1, as Figure 6.2 shows.



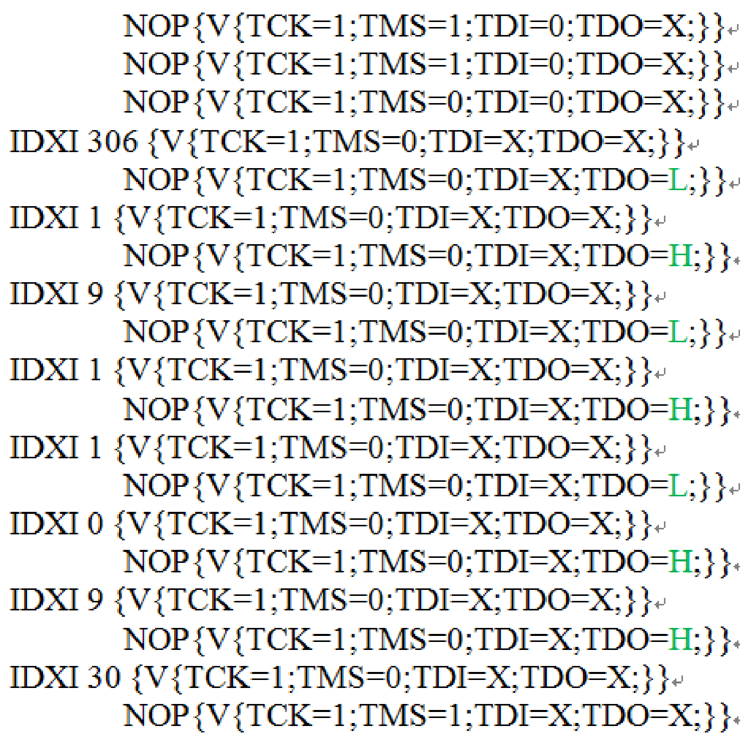
```
          NOP{V{TCK=1;TMS=1;TDI=0;TDO=X;}}
          NOP{V{TCK=1;TMS=1;TDI=0;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}
IDXI 306 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=0;TDO=H;}}
IDXI 1 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=1;TDO=L;}}
IDXI 9 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=0;TDO=L;}}
IDXI 1 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=1;TDO=L;}}
IDXI 1 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=0;TDO=H;}}
IDXI 0 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=X;TDO=L;}}
IDXI 9 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=X;TDO=H;}}
IDXI 30 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=1;TDI=X;TDO=X;}}
```

Figure 6.6: Load PI of *pattern=1* of Figure 6.4 to ATE and verify PO of *pattern=0* of Figure 6.4 in ATE

In Figure 6.6, the TDO pattern values of ‘H’ and ‘L’, colored red, are the verification of *pattern=0*. The first five values, HLLLH, is the PI of *pattern=0*, which proves that BSR is connected between TDI and TDO as one scan path. The last two values, LH, are the PO of *pattern=0*, which proves that the result of first test vector is verified.

When verifying the result of the first test vector on TDO, the load of the second test vector, 01010, on TDI is at the same time, which are the TDI value of ’0’ and ’1’ colored green in Figure 6.6.

In Figure 6.7, since this is the last test pattern, all values at TDI are specified as ’X’.



```
          NOP{V{TCK=1;TMS=1;TDI=0;TDO=X;}}
          NOP{V{TCK=1;TMS=1;TDI=0;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=0;TDO=X;}}
IDXI 306 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=X;TDO=L;}}
IDXI 1 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=X;TDO=H;}}
IDXI 9 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=X;TDO=L;}}
IDXI 1 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=X;TDO=H;}}
IDXI 1 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=X;TDO=L;}}
IDXI 0 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=X;TDO=H;}}
IDXI 9 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=0;TDI=X;TDO=H;}}
IDXI 30 {V{TCK=1;TMS=0;TDI=X;TDO=X;}}
          NOP{V{TCK=1;TMS=1;TDI=X;TDO=X;}}
```

Figure 6.7: Verify PO of *pattern=1* of Figure 6.4 in ATE

The 'H' and 'L' values colored green at TDO are the verification of *pattern=1*. The first five values, LHLHL, are the PI of *pattern=1*, which proves that BSR is connected between TDI and TDO as one scan path. The last two values, HH, are the PO of *pattern=1*, which proves that the result of the second test vector is verified.

6.3 Analysis of the test time

Let's assume the number of FastScan test patterns is M, the number of BSC in the BSR is N, and the longest scan chain length is L in the test circuit, and C is a constant. Then the number of test timing cycles is:

timing cycles = M * (L*4) * (N+3) + N + C (equation 1)

The meaning of each term is explained below.

- The second term (L*4):

For each of the test patterns, firstly we need L shifting cycles to shift the scan inputs in. Notice that the shifting cycles are controlled by the rising edge of the system clock, which is a transition from state 0 to state 1. From section 5.4.2, we know that it requires the value of BSR to be shifted for each distinct clock signal state, which means twice for loading one bit. Thus we need $L*2$ patterns for the data load. Similarly, we also need L shifting cycles to shift the scan outputs out, so another $L*2$ test patterns are needed. Therefore, each test operation requires $L*4$ patterns in total.

Additionally, notice that L represents the longest scan chain length. The number of scan chains is specified when using the DFTAdvisor software to convert the non-scan design to a scan design. Let's say the longest scan chain length of one scan chain design is L1, and the longest scan chain length of a multiple scan chain design is L2. It is obvious that L2 is less than L1. Therefore, using multiple scan chains in a design would reduce the test time proportionally. The disadvantage is that for each scan chain, a separate scan_in input is required, and an optional scan_out output is also required. So we need at least one extra user I/O pin, and at most two extra user I/O pins, for each extra scan chain. It is a tradeoff between the pin resources of the FPGA and the test time.

- The third term (N+3):

  From the first two terms in equation 1, we know that $M*(L*4)$ patterns are needed for a complete functional test. We need to remember that the process of test data applied is realized by shifting data into BSR in the SHIFT-DR state, and thus the BSR length of an FPGA is crucial. The xc3s50A vq100 FPGA has N=373 BSCs in its BSR, which means we need to pulse TCK 373 times to shift all the data bits to their corresponding positions for each pattern.

  N timing cycles are referred to as one shifting cycle. After one shifting cycle, the TAP controller state needs to go to UPDATE-DR to update the values held by the BSR.

Secondly it needs to go to CAPTURE-DR to capture the response, which is the set of outputs that need to be checked against the expected response. Thirdly it needs to enter SHIFT-DR again to shift the outputs out through TDO, to be verified.

TMS=1 to enter EXIT1-DR while the last bit of input is shifted in. Therefore, after the N timing cycles in SHIFT-IR, we need to apply three more clocks for TMS=110 to go through: UPDATE-DR, SELECT-DR-SCAN, CAPTURE-DR, followed by the next shifting cycle.

- The fourth term N:

  The reason why we need another N timing cycles is that the inputs shifted in through TDI and the corresponding outputs shifted out through TDO do not occur at the same time. In the shifting cycle, only the first set of inputs are shifted in through TDI. Only after entering the CAPTURE-DR state can we then check the outputs at next shifting cycle. During the last shifting cycle, only the response of last set of inputs is shifted out through TDO. Between the first and last shifting cycles, for example, the K^{th} set of inputs are shifted in through TDI at shifting cycle, while the $(K\text{-}1)^{th}$ set of outputs are shifted out at the same time.

- The fifth term C:

  C is just a small constant. At first before the shifting of the test patterns, we need to set TMS=11111 to make sure the TAP controller is in the TLR state. Then we set TMS=0110 to prepare to go into the SHIFT-IR state. Then we set TMS=000001, while TDI=111000 at the same time, to shift the INTEST binary code into IR. Then we set TMS=110 to prepare to go into SHIFT-DR state. At last, after the completion of the functional test, we need to reset the component to the normal system logic by entering the TLR state. Therefore, TMS=1111 is applied at the end. In total, C is at least 22.

For example: N is 373 for FPGA xc3s50a vq100. The scan design of s27.v has a scan length of 3, so L is 3. The number of FastScan test patterns is 9, so M is 9.

Therefore, the number of test timing cycles is: 9*(3*4) *(373+3) +373+22=41006 cycles.

Chapter 7

Conclusions and future works

7.1 Conclusions

In this thesis, the use of JTAG pins in functional test on an ATE is proposed, which can be used to test chips compatible with the IEEE 1149.1 standard that have no I/O pins accessible to the tester. ISCAS'85 and ISCAS'89 benchmark circuits were configured as the on-chip logic on a XILINX xc3s50a vq100 FPGA to demonstrate this method of testing. The key lies in loading test data to, and reading the result from, the correct BSCs. An ATE test pattern generator program is written in Perl to convert ATPG patterns to ATE test patterns to be used in the JTAG-based test. The results show that functional test through JTAG is a trade off between device I/O resources and test time.

In addition, some other aspects of JTAG instructions are also exploited on the ATE. The IDCODE instruction can be used to determine the device type and to check the integrity of JTAG pins. The BYPASS instruction provides a minimum length of one to connect TDI to TDO. The SAMPLE instruction can take a snapshot of on-chip logic function to be examined through JTAG.

7.2 Future work

Since the DUT in this work has only one component that is compatible with the IEEE 1149.1 standard, the EXTEST instruction was not tested. However, the test of circuitry external to the chips on a PCB is important, typically interconnects between chips and between boards. If a board has several components that are compatible with the IEEE

1149.1 standard, future work can be performed to test interconnects between each chip using the EXTEST instruction, implemented on the ATE.

Furthermore, as technology developed rapidly, many advanced techniques are emerged to increase the test difficult for the IEEE 1149.1 standard. For example, because serial data I/O have increased to gigabits per second rates, signals such as differential and AC-coupled networks have been introduced. An extended boundary scan standard IEEE 1149.6 has been approved. Because of increasing board complexity, system-on-chip (SOC) technology faces test challenges in addition to those of normal VLSI devices. A new test standard IEEE 1500 has been approved for connecting cores on SOC devices. Exploitation of these new standards in testing with ATE could be a good future work.

Bibliography

[1] Ben Bennetts, “IEEE 1149.1 JTAG and Boundary-Scan Tutorial”, ASSET InterTech, Inc., 2007.

[2] *Digilent Plug-in for Xilinx 12.x Tools User Manual*, Digilent Inc., 2011.

[3] M. Smith, *Application-Specific-Integrated-Circuit.* Boston, Addison-Wesley, 1997.

[4] *Mercury FPGA module Reference Manual.* micronova., 2012.

[5] *Spartan-3 Generation FPGA User Guide.* Xilinx Inc., San Jose, CA, 2011.

[6] L. Wang, C. Wu, X. Wen, *VLSI Test Principles and Architecture: Design for Testability.* San Francisco, morgan kaufmann, 2006.

[7] Advantest R&D Center Inc., “OPENSTAR Test Programming Language (OTPL)”. Advantest Corporation technical publication. 2003.

[8] *Spartan-3A FPGA Family: Data Sheet.* Xilinx Inc., San Jose, CA, 2010.

[9] “JTAG Chain Debugging,” http://www.xjtag.com/sup-kb-Q3.php (accessed Feb. 14, 2015)

[10] “ISCAS85 Sequential Benchmark Circuits.,” http://www.pld.ttu.ee/ maksim/benchmarks/iscas85/verilog/ (accessed Mar. 19, 2015)

[11] “ISCAS89 Sequential Benchmark Circuits.,” http://www.pld.ttu.ee/ maksim/benchmarks/iscas89/verilog/ (accessed Mar. 27, 2015)

[12] “ISE WebPACK Design Software,” http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.html (accessed Mar.20, 2015)

[13] *Leonardo Spectrum User Guide. Mentor Graphics Corp*, Wilsonville, OR, 2011.

[14] *DFTAdvisor Reference Manual.* Mentor Graphics Corp, Wilsonville, OR, 2011.

[15] *ISE In-Depth Tutorial.* Xilinx Inc., San Jose, CA, 2011.

[16] *BSDL file for device XC3S50A_VQ100.* Xilinx Inc., San Jose, CA, 2008.

[17] *Scan and ATPG Process Guide (DFTAdvisor and Tessent FastScan).* Mentor Graphics Corp, Wilsonville, OR, 2011

[18] M.L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits*. Boston, Springer, 2000.

[19] V. P. Nelson, "Functional IC test with the ADVANTEST T2000 GS system", Auburn University, Auburn, AL, Jan 15, 2014.

[20] *IEEE Std. 1149.1-2001, IEEE Standard Test Access Port and Boundary-Scan Architecture*, IEEE Press, New York, 2001.