The Tagalyzer - A JTAG Boundary Scan Debug Tool

XAPP 103 January 23, 1998 (Version 1.0) Application Note

Summary

The Tagalyzer is a diagnostic tool that helps debug long JTAG boundary scan chains. It can be modified to adapt to a wide variety of different testing situations, and is made from a single XC9536 CPLD. It can be used to debug JTAG chains made up of any manufacturers parts. The Tagalyzer can be expanded, to support arbitrarily long boundary scan chains and adapted to change its functionality, as needed.

Xilinx Family

XC9500

Introduction

JTAG Boundary Scan designs can become complex and difficult to debug. This can warrant the need for specialized tools capable of isolating and defining the exact state of the JTAG chain. One such tool - the Tagalyzer - is surprisingly inexpensive and easy to use while providing highly useful information to the board level engineer. Using a single Xilinx CPLD, this tool can be built for a few dollars and debug JTAG test circuits of arbitrary length. This paper includes a detailed discussion of the design, operation and modification of the basic instrument. With the design presented in this paper, designers will be able to rapidly debug their JTAG systems in a practical and straightforward way. Design details include complete VHDL code for the Tagalyzer as well as a technical discussion of its use.

JTAG Review and Other Resources

For this discussion, it will be assumed that the reader is an experienced JTAG user. If that is not the case, that problem is easily remedied by studying available resources. Please refer to Xilinx application notes: XAPP068, XAPP069 and XAPP070.

Currently, these application notes are located on Xilinx World Wide Web at:

`http://www.xilinx.com/apps/epld.htm`

The heart of any JTAG test chain (Figure 1) is the Test Access Port (TAP) controller (Figure 2). The TAP controller is a 16 state Moore type state machine that dictates the control of all JTAG activities. JTAG boundary scan testing is accomplished with only four external signals: Test Mode Select (TMS), Test Clock (TCK), Test Data Input (TDI) and Test Data Output (TDO). When parts are interconnected with a JTAG boundary scan chain, their various TMS, TCK, TDI and TDO signals are attached in various configurations. It is difficult to tell from the outside pin view, exactly what is going on inside each chip. In particular, it is hard to tell what state that chip's TAP controller is in. That is where the Tagalyzer comes in.

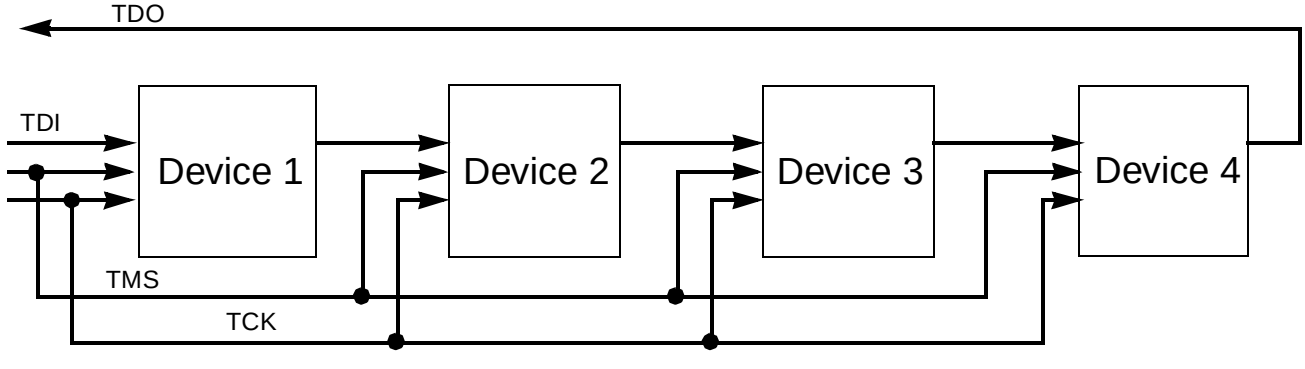

Figure 1: JTAG Boundary Scan Control Signal Connections

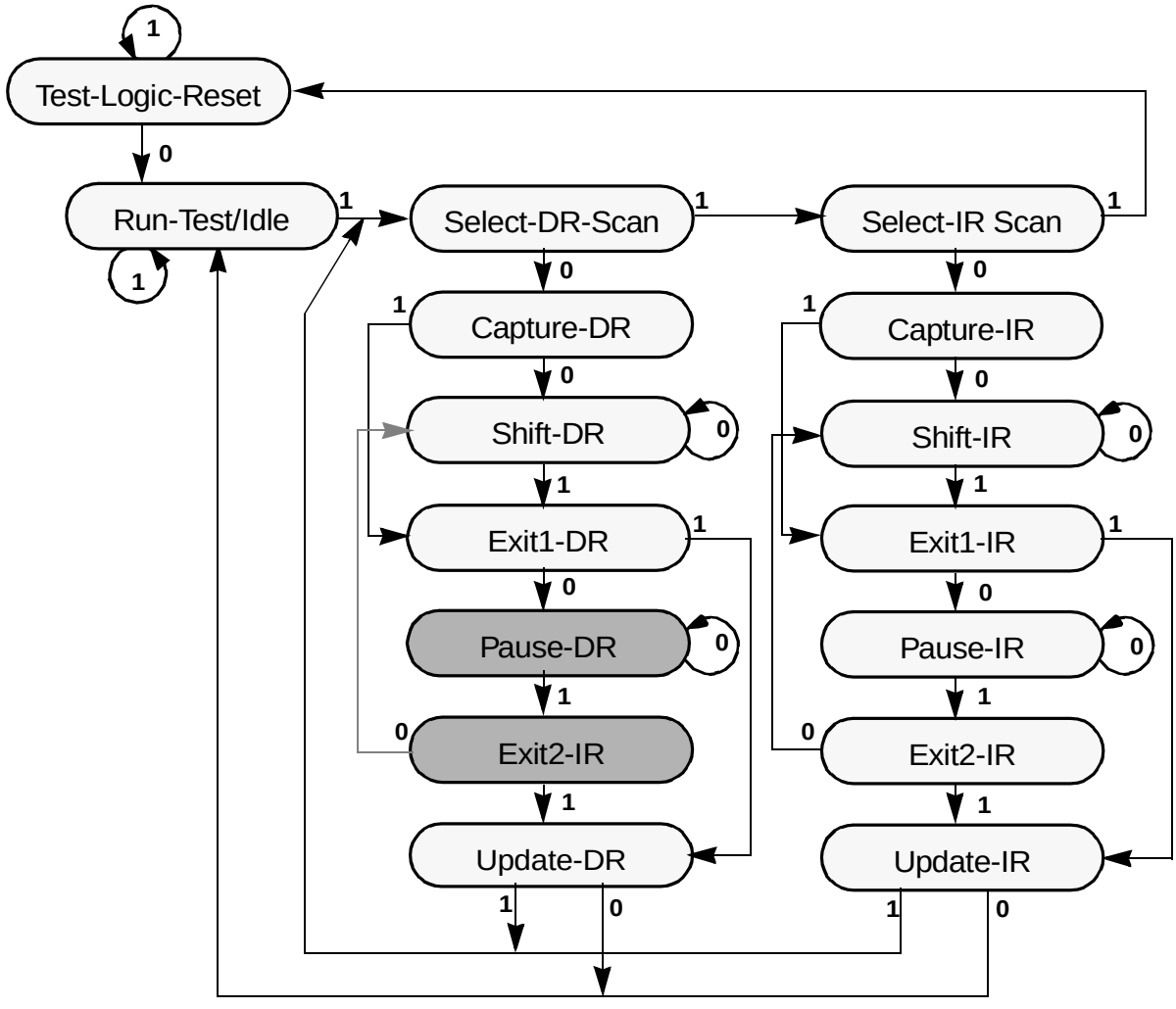

Figure 2: The JTAG TAP CONTROLLER

Note that in Figure 2, there are sixteen states. Each state has a precise meaning and definition, which directly reflects the state of the boundary scan test chain at any point in time. In Figure 2, the upper left two states are key beginning and ending states. The center column of states (starts with Select-DR-Scan) describes bit processing for the JTAG data register. The right column of states (starts with Select-IR-Scan) describes bit processing for the JTAG instruction register. Transitions between all states is done by applying combinations of TMS, TCK and TDI. By attaching to these signals, it is possible to create a separate version of the TAP state machine that tracks the TAP states, and displays them. It is also possible to capture and display the JTAG instruction and data registers. That's what the Tagalyzer does.

The Tagalyzer Structure

Although not required, the Tagalyzer is conveniently constructed on the XC9536 Customer Demo Card (Figure 3). This card can be obtained for a nominal fee from Xilinx Customer Support or Xilinx Sales. It combines an XC9536 with power connections making it easy to use with a single 9V battery. Delivered, the card has a set of 10 LEDs installed that will be used as part of the Tagalyzer. Additional LED displays will need to be installed to replicate the unit described here. Basically, the idea is simple - build a copy of the TAP controller in an XC9536 so that it externally displays the current TAP controller state. Additionally, it is possible to display (current LED column) the contents of the JTAG instruction or data register. VHDL code for the TAP controller is included in the appendix of this application note.

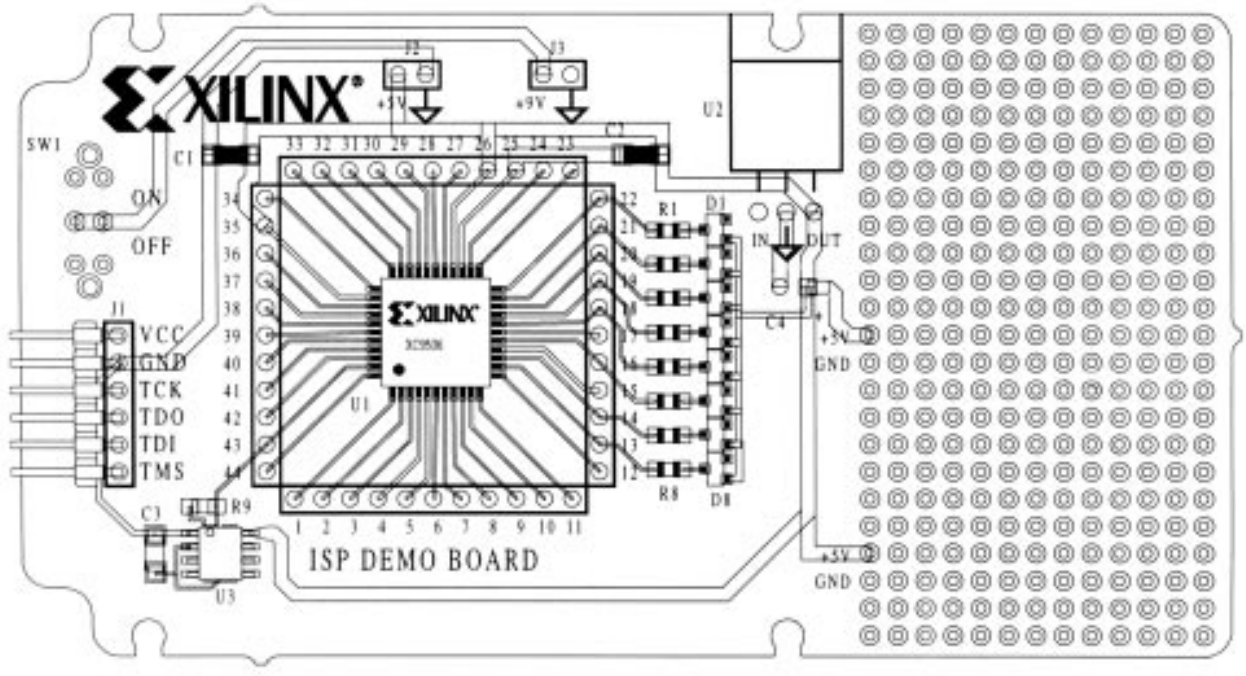

Figure 3: XC9536 Customer Demo Board

Figure 4 shows how the Tagalyzer can be connected to various points along a 4 device JTAG boundary scan chain. At each site, 3 external leads must be attached to the respective TCK, TMS and TDI points of the chain. At that position, the Tagalyzer will track the internal TAP controller and display its matching state. This permits debuggers the ability to quickly identify misconnections and correct any TAP controller misconceptions about internal operation. It also permits, to some degree, the tracking of TDO (for subset conditions like Bypass configurations). It is also useful to display the current data and instruction registers. A point to remember is that you won't be attaching the XC9536 JTAG pins at these points, but rather pins which are attached to ordinary XC9536 I/O points.

There is no limit on the number of parts that can be tested with the Tagalyzer. It will track the state of whatever part to which it is attached. Also, because the XC9536 can be easily modified, the design inside the Tagalyzer can be altered to suit the specific problem addressed. For instance, the column can display instruction in one version and data in another. For larger problems, the design can be migrated to larger XC9500 parts, if needed, but it must be rebuilt on a larger card.

Software Support

Xilinx M1.4 Design Software includes a JTAGProgrammer module. The JTAGProgrammer module has a special feature that is appropriate to use with the Tagalyzer, which permits low level delivery of a prescribed number of clocks and delivery of TDI and capture of TDO.

The debugger (see Figure 5) provides you with a method to apply boundary-scan test access port stimulus. This feature allows you to set TDI and TMS, then pulse TCK a specified number times. You can monitor TDO, TDI and TMS using the Tagalyzer to see if the boundary scan chain is operating correctly. The debugger also displays the current TAP state and allows you to reset the chain to Run Test Idle.

To access the debugger:

File ‡ Debug Chain

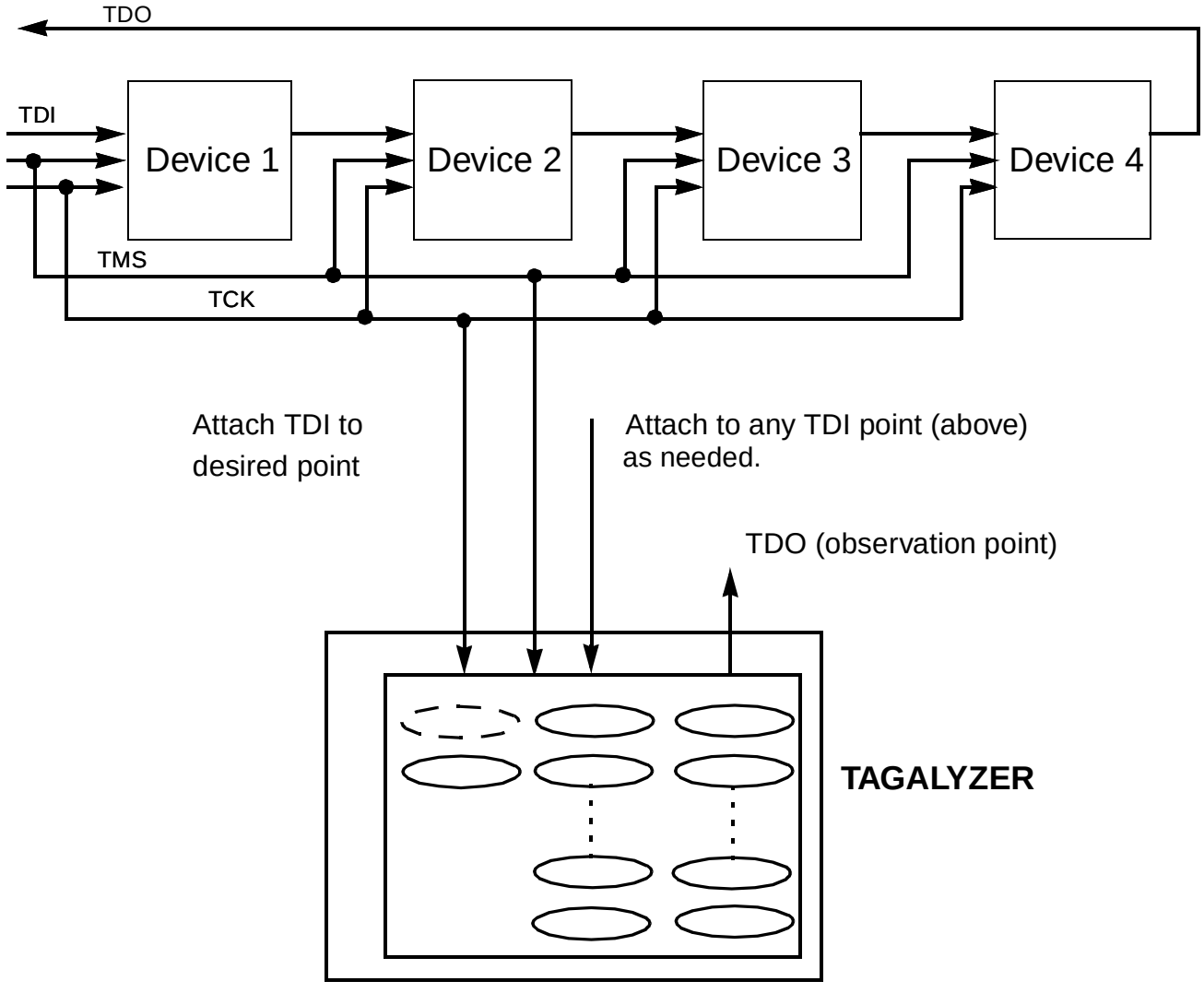

Figure 4: Possible Tagalyzer Attachments

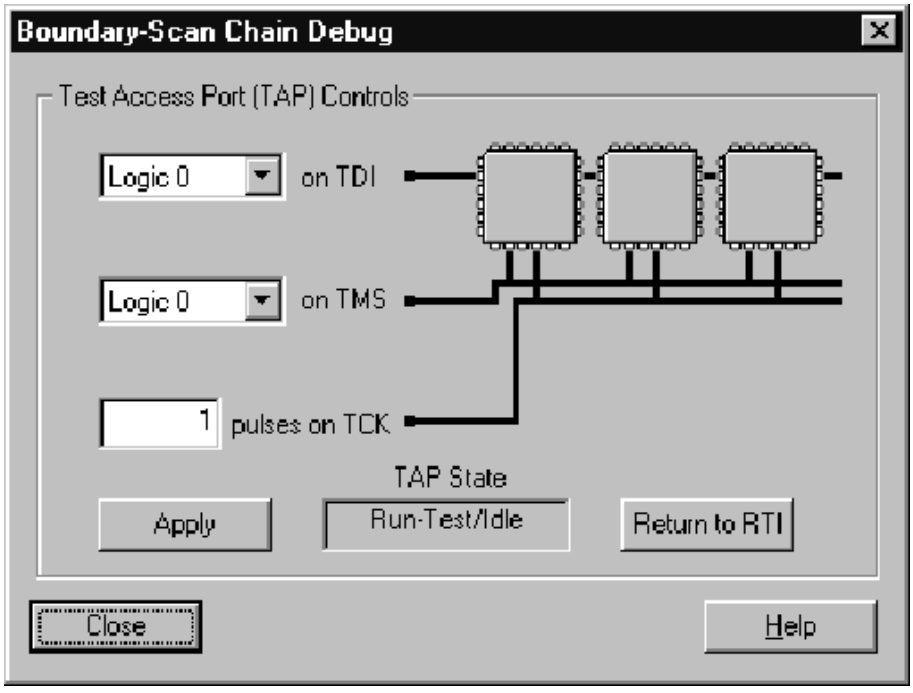

Figure 5: M1 JTAG Downloader Debug GUI

The features of this dialog box operate as follows:

1. The first selection box allows you to set a logic state for TDI. This state will not be set until you click on the Apply button.
2. The second selection box allows you to set a logic state for TMS. This state will not be set until you click on the Apply button.
3. The third selection box allows you to set a number of pulses to apply to TCK. These pulses will not be sent until you click on the Apply button. If you want to see the pulses again, click the Apply button as often as you want.
4. The TAP State window displays the current state of the controller.

5. The Return to RTI (Run Test Idle) button executes a Test Logic Reset, then returns to Run Test Idle.

Conclusion

The Tagalyzer is an easy to use, very inexpensive but powerful diagnostic tool. It can be used to debug JTAG chains of any length and virtually any manufacturer's parts. It is easily constructed using Xilinx XC9536 customer demo board and Xilinx M1.4 Design Software.

Appendix A Tagalyzer VHDL Source Code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use WORK.tag_defs.ALL;
use WORK.ALL;

entity tagalize is
port
(
my_tdi:instd_logic;
my_tck:instd_logic;
my_tms:instd_logic;
my_tdo:instd_logic;

Instruction:buffer std_logic_vector (7 downto 0);
Data:buffer std_logic_vector (2 downto 0);

tdo_output:buffer std_logic_vector (2 downto 0);
TLR:outstd_logic;
RTI:outstd_logic;
SDR:outstd_logic;
CDR:outstd_logic;
SHDR:outstd_logic;
E1DR:outstd_logic;
PDR:outstd_logic;
E2DR:outstd_logic;
UDR:outstd_logic;
SIR:outstd_logic;
CIR:outstd_logic;
SHIR:outstd_logic;
E1IR:outstd_logic;
PIR:outstd_logic;
E2IR:outstd_logic;
UIR:outstd_logic
);
end tagalize;

architecture behavior of tagalize is
signal jtag_state:std_logic_vector (15 downto 0);

begin
TLR <= jtag_state(0);
RTI <= jtag_state(1);
SDR <= jtag_state(2);
SIR <= jtag_state(3);
CDR <= jtag_state(4);
CIR <= jtag_state(5);
SHDR <= jtag_state(6);
SHIR <= jtag_state(7);
E1DR <= jtag_state(8);
E1IR <= jtag_state(9);
PDR <= jtag_state(10);
PIR <= jtag_state(11);
E2DR <= jtag_state(12);
E2IR <= jtag_state(13);
UDR <= jtag_state(14);
UIR <= jtag_state(15);
process

begin
wait until (my_tck'event and my_tck = '1');
case jtag_state is
when test_logic_reset =>
if my_tms = '1' then jtag_state <= test_logic_reset;
else jtag_state <= run_test_idle;
end if;

when run_test_idle =>
if my_tms = '1' then jtag_state <= sel_dr_scan;
else jtag_state <= run_test_idle;
end if;

when sel_dr_scan =>
if my_tms = '1' then jtag_state <= sel_ir_scan;
else jtag_state <= capture_dr;
end if;
```

```
when sel_ir_scan =>
if my_tms = '1' then jtag_state <= test_logic_reset;
else jtag_state <= capture_ir;
end if;

when capture_dr =>
if my_tms = '1' then jtag_state <= exit1_dr;
else jtag_state <= shift_dr;
end if;

when capture_ir =>
if my_tms = '1' then jtag_state <= exit1_ir;
else jtag_state <= shift_ir;
end if;

when shift_dr =>
if my_tms = '1' then jtag_state <= exit1_dr;
else jtag_state <= shift_dr;
end if;

when shift_ir =>
if my_tms = '1' then jtag_state <= exit1_ir;
else jtag_state <= shift_ir;
end if;

when exit1_dr =>
if my_tms = '1' then jtag_state <= update_dr;
else jtag_state <= pause_dr;
end if;

when exit1_ir =>
if my_tms = '1' then jtag_state <= update_ir;
else jtag_state <= pause_ir;
end if;

when pause_dr =>
if my_tms = '1' then jtag_state <= exit2_dr;
else jtag_state <= pause_dr;
end if;

when pause_ir =>
if my_tms = '1' then jtag_state <= exit2_ir;
else jtag_state <= pause_ir;
end if;

when exit2_dr =>
if my_tms = '1' then jtag_state <= update_dr;
else jtag_state <= shift_dr;
end if;

when exit2_ir =>
if my_tms = '1' then jtag_state <= update_ir;
else jtag_state <= shift_ir;
end if;

when update_dr =>
if my_tms = '1' then jtag_state <= sel_dr_scan;
else jtag_state <= run_test_idle;
end if;

when update_ir =>
if my_tms = '1' then jtag_state <= sel_dr_scan;
else jtag_state <= run_test_idle;
end if;

when others =>
jtag_state <= test_logic_reset;
end case;
end process;

process
begin
wait until (my_tck'event and my_tck = '1');

if jtag_state = shift_dr then
Data (1 downto 0) <= Data (2 downto 1);
Data (2) <= my_tdi;
tdo_output (1 downto 0) <= tdo_output (2 downto 1);
tdo_output (2) <= my_tdo;
end if;
```

```
if jtag_state = shift_ir then
Instruction (6 downto 0) <= Instruction (7 downto 1);
Instruction (7) <= not my_tdi;
tdo_output (1 downto 0) <= tdo_output (2 downto 1);
tdo_output (2) <= my_tdo;
end if;

end process;
end behavior;
```