IEEE 1149.1 Test Access Port

Author: Igor Mohor
IgorM@opencores.org

Revised: Nathan Yawn
nathan.yawn@opencores.org

Rev. 2.1

February 21, 2008

Copyright (C) 2001 - 2009 OPENCORES.ORG and Authors.

This document is free; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This document is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Revision History

Rev.	Date	Author	Description
0.1	02/02/01	Igor Mohor	First Draft
0.2	05/04/01	IM	Trace port added
0.3	16/04/01	IM	WP and BP number changed, trace modified
0.4	01/05/01	IM	Title changed, DEBUG instruction added, scan chains changed, IO ports changed
0.5	05/05/01	IM	TSEL and QSEL register changed
0.6	06/05/01	IM	Ports connected to the OpenRISC changed
0.7	14/05/01	IM	MODER register changed, trace scan chain changed; SSEL register added
0.8	18/05/01	IM	RESET bit and signal added; STALLR changed to RISCOP
0.9	23/05/01	IM	RISC changed to OpenRISC; WISHBONE interface added, SPR and memory access added
0.10	01/06/01	IM	Meaning of Instruction status and Load/store status changed in all registers; more details added to Appendix A
0.11	10/09/01	IM	Register and OpenRISC scan chain operation changed
1.0	19/09/01	IM	Some registers deleted
1.1	15/10/01	IM	WISHBONE interface added; RISC Stall signal is set by breakpoint and reset by writing 0 to RISCOP register
1.2	03/12/01	IM	Chain length changed so additional CRC checking can be performed
1.3	21/01/02	Jeanne Wiegelmann	Document revised.
1.4	07/05/02	IM	Register MONCNTL added.
1.5	10/10/02	IM	WISHBONE Scan Chain changed to show state of the access.
1.6	06/11/02	IM	TRST_PAD_I changed from active low signal to active low signal.
1.7	23/09/03	Simon Srot	Mutliple CPU support added, WB 16-bit and 8-bit access possible through WBCNTL register use.
2.0	30/01/04	IM	JTAG and debug interface split into two different cores (and documents).
2.1	05/16/09	Nathan Yawn	Document revised for Adv. Debug I/F.

Contents

List of Figures

List of Tables

1

Introduction

The JTAG TAP controller is used for development purposes (Boundary Scan testing, Memory BIST and debugging). It functions as an interface between the processor(s), peripheral cores, and any commercial debugger/emulator or Boundary Scan (BS) testing device. The external debugger or BS tester connects to the core via an IEEE 1149.1 compatible JTAG port such as this one. This core was originally designed to connect to the original OpenCores debug interface ("dbg_interface" on the OpenCores web page). However, this version has been modifed for use with the Advanced Debug Interface (adv_dbg_if), and is not compatible with the older dbg_interface core.

2

IO Ports

2.1 JTAG Ports

Port	Width	Direction	Description
tck_pad_i	1	input	Test clock input
tms_pad_i	1	input	Test mode select input
tdi_pad_i	1	input	Test data input
tdo_pad_o	1	output	Test data output
tdo_padoe_o	1	output	TDO output enable
trstn_pad_i	1	input	Test reset input (asynchronous; active low)

Table 1: JTAG Ports

2.2 Debug Ports

Port	Width	Direction	Description
test_logic_reset_o	1	output	TAP controller state “Test Logic Reset”
run_test_idle_o	1	output	TAP controller state “Run Test / Idle”
shift_dr_o	1	output	TAP controller state “Shift DR”
pause_dr_o	1	output	TAP controller state “Pause DR”
update_dr_o	1	output	TAP controller state “Update DR”
capture_dr_o	1	output	TAP controller state “Capture DR”

Port	Width	Direction	Description
extest_select_o	1	output	Boundary Scan external test select
sample_preload_ select_o	1	output	Sample/Preload select
mbist_select_o	1	output	Memory BIST select
debug_select_o	1	output	Debug select
tdi_o	1	output	TDI signals that connects to all TDI signals of sub-modules
debug_tdo_i	1	input	TDO signal from debug module
bs_chain_tdo_i	1	input	TDO signal from boundary scan chain module
mbist_tdo_i	1	input	TDO signal from memory BIST

Table 2: Debug Ports

3

Registers

This section specifies all registers in the JTAG Interface.

3.1 Registers List

Name	Width	Access	Description
IR	4	R/W	Instruction Register
IDCODE	32	RO	IDCODE Register
BYPASS	1	R/W	Bypass Register

Table 3: Register List

3.2 IR (Instruction Register)

Bit #	Access	Description
3:0	R/W	INSTR – Instruction 0000 = EXTEST 0001 = SAMPLE_PRELOAD 0010 = IDCODE 1000 = DEBUG 1001 = MBIST 1111 = BYPASS

Table 4: IR Register

Value from this register is always read as 0101b.

3.3 IDCODE register

Bit #	Access	Description
31:0	RO	ID – Identification

Table 5: IDCODE Register

Default value of the IDCODE register is set in the tap_defines.v file. This is a compile-time option.

3.4 BYPASS register

Bit #	Access	Description
0	R/W	1-bit bypass register; always read as 0

Table 6: IDCODE Register

4

Operation

This section describes the operation of the JTAG interface, TAP controller, supported instructions, and connection of the sub-modules. Knowledge of the basic JTAG TAP standard is assumed for this section.

4.2 JTAG Interface and the TAP Controller

The JTAG Interface and TAP Controller are fully IEEE Std.1149.1 compliant. The JTAG interface consists of six signals:

- tck_pad_i (TCK)
- tms_pad_i (TMS)
- tdi_pad_i (TDI)
- trstn_pad_i (TRST)
- tdo_pad_o (TDO)
- tdo_padoe_o

tdo_padoe_o is an enable signal for the TDO pad. tdo_pad_o and tdo_padoe_o should be combined together to create the tri-state TDO signal at a higher layer.

trstn_pad_i is the TRST reset, which is active low. This signal is asynchronous, its effects do not depend on an edge of TCK. The TAP Controller will not leave the Test Logic Reset state while this signal is asserted.

The following instructions are supported by the TAP (see section Table 3: Register List for instruction coding):

- EXTEST
- SAMPLE/PRELOAD
- IDCODE
- DEBUG
- MBIST
- BYPASS

All supported instructions are shifted to the instruction register with the LSB bit shifted first.

Supported instructions are described in the following sections.

4.2.1 EXTEST (IR=0000)

The EXTEST instruction connects the boundary scan chain between TDI and TDO. During the EXTEST instruction, the boundary-scan register is accessed to drive test data off-chip via the boundary outputs and to receive test data in-chip via the boundary inputs. The bit code of this instruction is defined as all zeroes by IEEE Std. 1149.1.

- CaptureDR state: The outputs from the system logic (test vector) are captured (copied into the scan chain registers).
- ShiftDR state: The captured test vector is shifted out via the TDO output, while a new test vector is shifted in via the TDI input.
- UpdateDR state: The data shifted in via TDI is applied to Output and Control cells (output pins are driven with 0, 1, or highZ).

4.2.2 SAMPLE/PRELOAD (IR=0001)

The SAMPLE/PRELOAD instruction allows the IC to remain in its functional mode while selecting the boundary-scan chain to be connected between TDI and TDO. During this instruction, you can access the boundary-scan chain registers via a data scan operation to take a sample of the functional data entering and leaving the IC. The instruction is also used to preload test data into the boundary-scan register before loading an EXTEST instruction. This instruction should only be used for production tests.

- CaptureDR state: The inputs from the system logic (test vector) are captured.
- ShiftDR state: The captured test vector is shifted out via TDO output while a new test vector is shifted in via TDI input.
- UpdateDR state: No changes.

4.2.3. IDCODE (IR=0010)

The IDCODE instruction allows the IC to remain in its functional mode while selecting the device identification register (ID register) to be connected between TDI and TDO. The device identification register is a 32-bit read-only register containing information regarding the IC manufacturer, device type, and version code. Accessing the device identification register does not interfere with the operation of the IC. Also, access to the device identification register should be immediately available via a TAP data-scan operation after power-up of the IC, or after the TAP has been reset by using the optional trstn_pad_i pin or by otherwise moving to the Test-Logic-Reset state.

- CaptureDR state: The ID value is captured from the ID register.
- ShiftDR state: The captured ID value is shifted out via TDO output.
- UpdateDR state: No changes (ID is a read-only register).

4.2.4 DEBUG (IR=1000)

The DEBUG instruction is used to communicate with in-circuit debugging hardware. When this instruction is activated, the debug_tdi_i signal is connected to TDO. Signal tdo_o should be connected to the TDI signal of the debug hardware (specifically, the OpenCores adv_dbg_if module). For more details see documentation for the Advanced Debug Interface system available on the OpenCores website.

Note that this instruction is not required or defined by the JTAG standard. It is present in the core to simplify the creation of a system using the Advanced Debug Interface. If your design does not use this interface, you may connect your own JTAG debug module in its place, or you may remove this instruction from the core. If you wish to add custom scan chains to the JTAG core, it is recommended that you use the implementation of the debug instruction and chain as your template.

4.2.5 MBIST (IR=1001)

The MBIST instruction is used for internal memory BIST. When active, mbist_tdi_i is connected to the TDO. Please refer to the MBIST documentation for more details about MBIST.

4.2.6 BYPASS (IR=1111)

The BYPASS instruction keeps the IC in a functional mode and selects the 1-bit bypass register to be connected between TDI and TDO. It allows communication with other devices on the same JTAG scan chain with a minimum of extra bits. The bit code of this instruction is defined as all ones by IEEE Std. 1149.1. Usage of an unimplemented instruction will result in the BYPASS instruction.

- CaptureDR state: A logical 0 is captured in the bypass register.
- ShiftDR state: Input data shifted in via TDI is shifted out via TDO via the bypass register (one clock cycle delay between input and output).
- UpdateDR state: No changes.

4.3 Scan Chains

There are four scan chains connected to the TAP controller:

- ID scan chain
- Debug interface scan chain
- Global boundary scan chain
- Memory BIST scan chain

The last three scan chains are inputs to the TAP controller, while the ID scan chain is internal. Switching between these four scan chains is done each time a new instruction is shifted into the Instruction Register (see section Table 4: IR Register for more details).

The ID scan chain is connected to TDO when the IDCODE instruction was previously shifted into the IR register. The Debug interface scan chain is connected to TDO when the DEBUG instruction was previously shifted into the IR register. The Global Boundary Scan chain is connected to TDO when the EXTEST or SAMPLE/PRELOAD instructions were previously shifted into the IR register. The Memory BIST scan chain is connected to TDO when the MBIST instruction was previously shifted in the IR register. Changes to the selected scan chain occur on the falling edge of the TCK clock signal when the TAP is in the Update IR mode.

All scan chains capture data at TDI on the rising edge of TCK; TDO is updated on the falling edge of the TCK clock signal.

4.3.1 ID Scan Chain

The ID scan chain is an internal 32-bit chain used for reading out the internal IDCODE. The least significant bit (LSB) is shifted out first. This chain is automatically selected after reset.

4.3.2 Debug Scan Chain

The Debug Interface scan chain is an external chain used for interfacing to a hardware debug module (CPU, wishbone…). The debug module is not part of this core. Please refer to the "adv_dbg_if" core documentation for an example of a debug scan chain. This chain should use the capture_dr_o, shift_dr_o, and update_dr_o outputs as control inputs for latching and shifting data; the test_logic_reset_o signal may be used to reset the debug logic. The debug_select_o signal acts as a chip select to the debug unit.

4.3.3 Global BS (Boundary Scan) Chain

This external chain allows access to the entire SoC periphery and is used for the boundary scan testing (interconnect test). This chain should use the capture_dr_o, shift_dr_o, and update_dr_o outputs as control inputs for latching and shifting data. This chain may be constructed using the "cells" modules included with this core.

WARNING: This version of the core has been modified for use with the Advanced Debug Interface. It has NOT been tested with the included "cells" modules. Those wishing to create a boundary scan chain should insure that the cells are compatible with the signals expected by this modified TAP.

4.3.4 Memory BIST Scan Chain

This external chain allows access to the Memory built-in self test scan chain. MBIST is not part of this project, TAP only has connection ports for it. This chain should use the capture_dr_o, shift_dr_o, and update_dr_o outputs as control inputs for latching and shifting data, as well as the run_test_idle_o signal to control when to run the self-test. The test_logic_reset_o signal may optionally be used to reset the self test logic. Note that to use this feature, an MBIST scan chain must be constructed. This is beyond the scope of this document.

5

Architecture

The TAP controller consists of several parts (blocks):

- JTAG interface with the TAP Controller
- Instruction register (described in section Table 3: Register List on page 10)
- Multiplexer that connects one of the scan chains to TDO.

5.1 TAP Controller

The TAP controller provides a connection to the external debugger and / or Boundary Scan testing equipment through the standard JTAG interface (IEEE Std. 1149.1 compliant). The external interface consists of five signals (TCK, TMS, TDI, TDO and TRST). The TAP controller is a 16-state finite state machine. The TAP moves from one state to another on the rising edge of TCK. The next state depends only on the TMS input. State outputs of the module (e.g. shift_dr_o) are active when the TAP controller is in the corresponding state. For a full description of TAP controller operation, see IEEE Standard 1149.1.

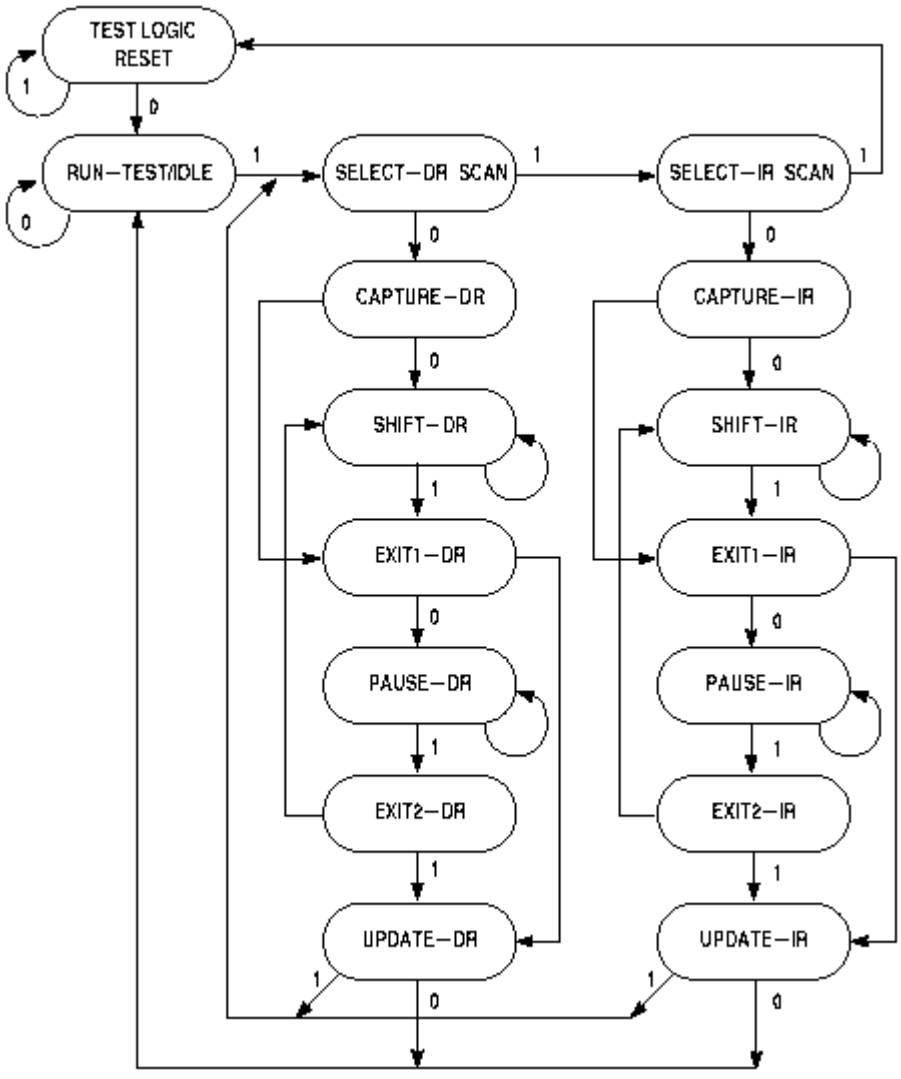

Figure 1: TAP Controller

5.1 TAP Controller in the system

As seen on the following figure, the TAP controller is just one part of the complete debugging system. For more information about the Advanced Debug Interface, go to the OpenCores web site. There you will find a complete debug IP core (adv_dbg_if), with test bench and documentation available.

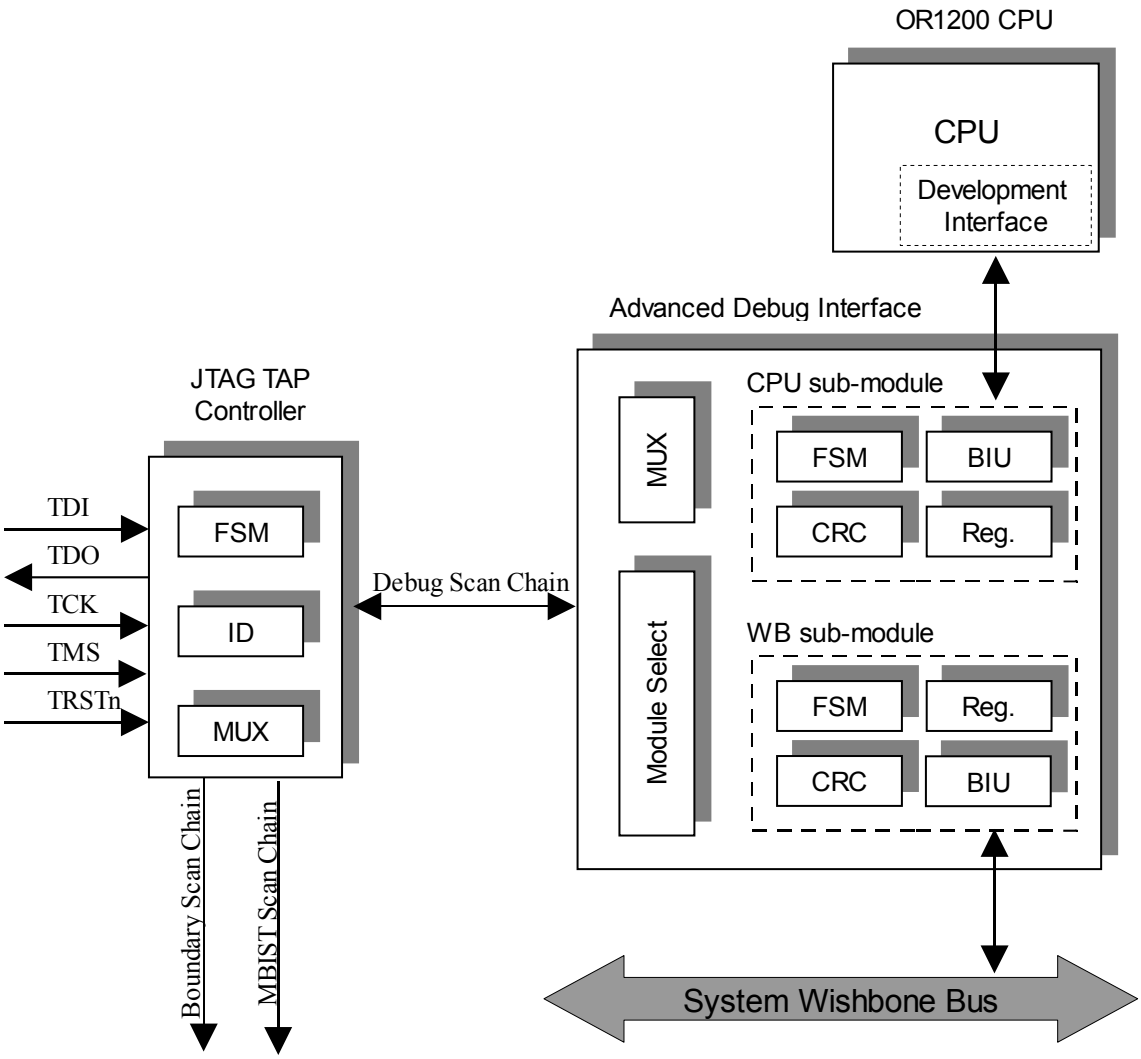

Figure 2: Complete System