SoC Debug Interface

Author: Igor Mohor
IgorM@opencores.org

Rev. 3.0

April 14, 2004

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	Multiple CPU support added, WB 16-bit and 8-bit access possible through WBCNTL register use.
2.0	01/02/04	IM	New version of the debug interface. Document

Rev.	Date	Author	Description
			name changed, Document split into two documents, one for TAP and one for debug.
2.1	14/03/04	IM	Missing things added to the documents.
2.2	18/03/04	IM	Table with supported scan chains (sub-modules) added.
3.0	14/04/04	IM	New version of the debug interface. Chain selection is renamed to module selection.

Contents

1 7

INTRODUCTION 7

2 8

IO PORTS 8

2.1 TAP PORTS 8
2.2 CPU PORTS 9
2.3 WISHBONE PORTS 9

3 11

REGISTERS 11

3.1 WISHBONE REGISTERS LIST 11
3.1.1 WB Command Register 12

3.2 CPU REGISTERS LIST 13
3.2.1 CPU Command Register 14

3.2.2 CPU Control Register 15

4 16

OPERATION 16

4.1 MODULE SELECTION 16
4.2 WISHBONE SUB-MODULE 19
4.2.1 WISHBONE Read or Write operation 20

4.2.2 WRITE_COMMAND 21

4.2.3 READ_COMMAND 22

4.2.4 GO_COMMAND 23

4.2.4.1 GO_COMMAND when read cycle is requested 23
4.2.4.2 GO_COMMAND when write cycle is requested 25
4.2.5 Data and select signals 27

4.2.6 Accessing slow devices 28

4.2.6.1 Reading from slow device 28
4.2.6.2 Writing to slow device 29
4.2.7 Errors in the WISHBONE sub-module 30

4.3 CPU SUB-MODULE 31

4.3.1 CPU Read or Write operation **33**
4.3.2 CPU Control Register Read or Write **34**
4.3.3 WRITE_COMMAND **35**
4.3.4 READ_COMMAND **36**
4.3.5 GO_COMMAND **37**
4.3.5.1 GO_COMMAND when read cycle is requested 37
4.3.5.2 GO_COMMAND when write cycle is requested 39
4.3.6 WRITE_CONTROL **41**
4.3.7 READ_CONTROL **42**
4.3.8 Stalling CPU **43**
4.3.9 Resetting CPU **43**
4.3.10 Accessing slow devices **44**
4.3.10.1 Reading from slow CPU 44
4.3.10.2 Writing to slow CPU 45
5 .... **46**
ARCHITECTURE **46**
5.1 DEBUG INTERFACE 48
5.2 CRC SUB-MODULE 48
5.3 WISHBONE SUB-MODULE 49
5.4 CPU SUB-MODULE 49

1 Introduction

The Development Interface is used for debugging purposes and is as such an interface between the processor(s), peripheral cores, and any commercial debugger/emulator or BS testing device. The external debugger or BS tester connects to the core via a fully IEEE 1149.1 compatible JTAG TAP port that is not part of this core. TAP is available at the opencores, too (look for project JTAG Test Access Port (TAP)).

2

IO Ports

2.1 TAP Ports

Debug interface connects to the TAP controller with the following signals:

Port	Width	Direction	Description
tck_i	1	input	Test clock input
tdi_i	1	input	Test data input
tdo_o	1	output	Test data output
shift_dr_i	1	input	TAP controller state “Shift DR”
pause_dr_i	1	input	TAP controller state “Pause DR”
update_dr_i	1	input	TAP controller state “Update DR”
rst_i	1	input	Reset signal.
debug_select_i	1	input	Instruction DEBUG is activated

Table 1: TAP Ports

2.2 CPU Ports

Port	Width	Direction	Description
cpu_clk_i	1	input	CPU clock signal.
cpu_addr_o	32	output	CPU address
cpu_data_i	32	input	CPU data input (data from CPU)
cpu_data_o	32	output	CPU data output (data to CPU)
cpu_bp_i	1	input	CPU breakpoint
cpu_stall_o	1	output	CPU stall (selected CPU is stalled)
cpu_stb_o	1	output	CPU strobe
cpu_we_o	1	output	CPU write enable signal indicates a write cycle when asserted high (read cycle when low).
cpu_ack_i	1	input	CPU acknowledge (signals end of cycle)
cpu_rst_o	1	output	CPU reset output (resets CPU)

Table 2: CPU Ports

2.3 WISHBONE Ports

Port	Width	Direction	Description
wb_clk_i	1	input	WISHBONE clock
wb_ack_i	1	input	WISHBONE acknowledge indicates a normal cycle termination
wb_adr_o	32	output	WISHBONE address output
wb_cyc_o	1	output	WISHBONE cycle encapsulates a valid transfer

Port	Width	Direction	Description
			cycle.
wb_dat_i	32	input	WISHBONE data input (data from WISHBONE)
wb_dat_o	32	output	WISHBONE data output (data to WISHBONE)
wb_err_i	1	input	WISHBONE error acknowledge indicates an abnormal cycle termination
wb_sel_o	4	output	WISHBONE select indicates which bytes are valid on the data bus.
wb_stb_o	1	output	WISHBONE strobe indicates a valid transfer.
wb_we_o	1	output	WISHBONE write enable indicates a write cycle when asserted high (read cycle when low).
wb_cab_o	1	output	WISHBONE consecutive address burst indicates a burst cycle.
wb_cti_o	3	output	WISHBONE cycle type identifier indicates type of cycle (single, burst, end of burst)
wb_bte_o	2	output	WISHBONE burst type extension

Table 3: WISHBONE Ports

3 Registers

This section specifies all registers in the Debug Interface. There are currently two sub-modules in the debug interface, WISHBONE and CPU.

WISHBONE sub-module contains one command register.

CPU sub-module contains one command and one control register.

3.1 WISHBONE Registers List

Name	Width	Access	Description
COMMAND	52	R/W	WISHBONE Command Register

Table 4: WISHBONE Register List

3.1.1 WB Command Register

Bit #	Access	Description
51:48	R/W	Access Type 0x0 = 8-bit WISHBONE write 0x1 = 16-bit WISHBONE write 0x2 = 32-bit WISHBONE write 0x4 = 8-bit WISHBONE read 0x5 = 16-bit WISHBONE read 0x6 = 32-bit WISHBONE read
47:16	R/W	Address
15:0	R/W	Data Size (in bytes) Actual size of the data to be written or read equals to (Data Size + 1)

Table 5: WB Command Register

Reset Value: 0x0

3.2 CPU Registers List

Name	Width	Access	Description
COMMAND	52	R/W	CPU Command Register
CONTROL	52	R/W	CPU Control Register

Table 6: CPU Register List

3.2.1 CPU Command Register

Bit #	Access	Description
51:48	R/W	Access Type 0x2 = CPU write 0x6 = CPU read
47:16	R/W	Address
15:0	R/W	Data Size Size of the data to be written or read equals to (Data Size + 1)

Table 7: CPU Command Register

Reset Value: 0x0

3.2.2 CPU Control Register

Bit #	Access	Description
51	R/W	Reset 0 = cpu_rst_o signal is set to 0 1 = cpu_rst_o signal is set to 1
50	R/W	Stall 0 = cpu_stall_o signal is set to 0 (CPU not stalled) 1 = cpu_stall_o signal is set to 1 (CPU stalled)
49:0	R/W	Reserved (write 0)

Table 8: CPU Control Register

Reset Value: 0x0

4 Operation

This section describes the operation of the Debug Interface and its sub-modules.

4.1 Module Selection

The debug interface is just an interface between the sub-module that is target specific and the TAP controller. Currently three sub-modules are connected to the debug interface, WISHBONE sub-module and two CPU sub-modules. Up to 16 sub-modules can be connected to the debug interface.

Module name	Module Code
WISHBONE Debug Module	0x0
CPU 0 Debug Module	0x1
CPU 1 Debug Module	0x2

Table 9: Supported sub-modules

First thing to do is to select the sub-module. This is done with the module select instruction. Following needs to be done prior to the module select operation:

- instruction DEBUG needs to be activated in the TAP (refer to the IEEE 1149.1 Test Access Port documentation for more information)

Then the "module select" instruction needs to be shifted-in through the TAP data chain:

- 1-bit with value 1 (This bit is treated as a "module select")
- 4-bit module ID (MSB shifted first)
- 32-bit CRC (MSB shifted first) that is protecting the incoming data (first four bits).
- 36 bits with value 0 (these bits are ignored in the debug interface)

While the "module select" instruction is shifted-in, the following data is shifted out:

- 37 bits with value 0 (this value should be ignored)
- 4-bit status (MSB shifted first)
  - 0 if incoming CRC is OK, 1 when CRC error occurs
  - 0 when existing module was selected, 1 when non-existing module was selected
  - always 0
  - always 0
- 32-bit CRC (MSB shifted first) that is protecting the outgoing data (four bits). Only status bits are protected with this CRC (first 37 bits are ignored).

Note: MSB bit is always shifted first.

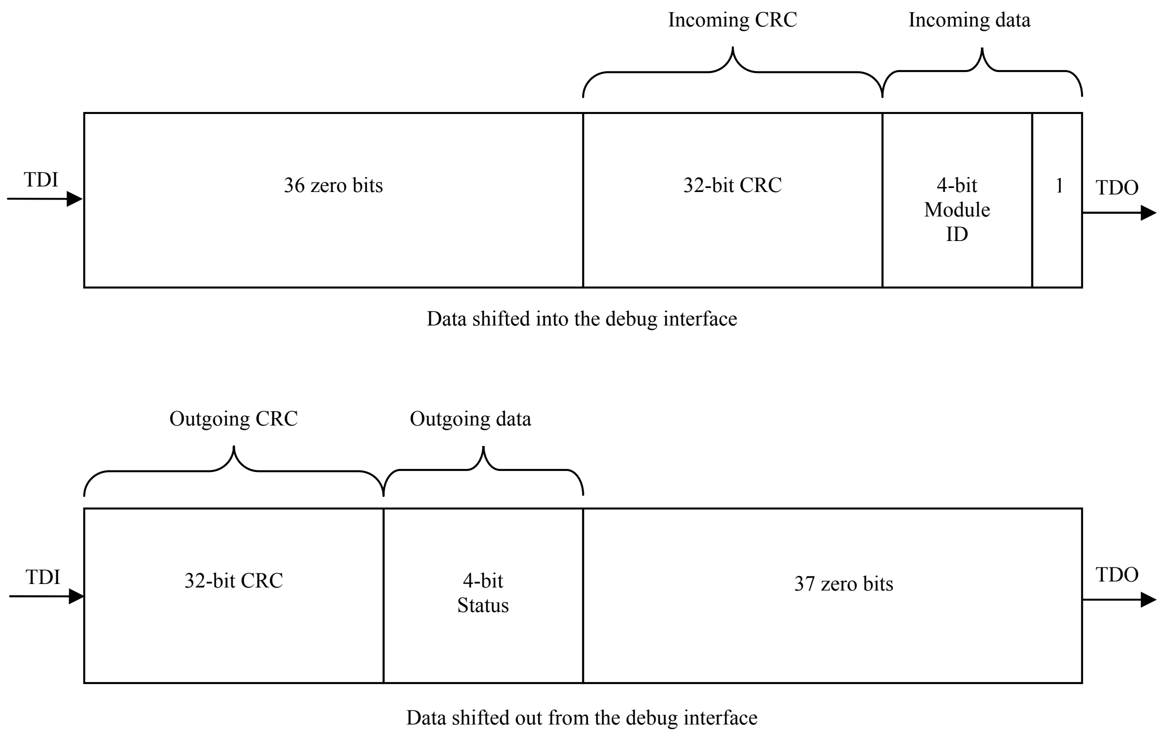


Figure 1: Module Selection

See section 5.2 CRC sub-module on page 48 for more details about the CRC.

4.2 WISHBONE Sub-module

WISHBONE sub-module is used for the WISHBONE devices debugging. It connects to the WISHBONE bus through the WB interface and can perform 8, 16 and 32-bit read and write accesses (single accesses).

There are two steps needed to write/read the data to/from the WISHBONE device:

- Address of the device, data size and the type of operation need to be written to the Command Register (see section 3.1.1 WB Command Register on page 12 for more details about the command register). This is done by using the **WRITE _COMMAND**.
- WISHBONE read or write operation (8, 16 or 32-bit) is executed with the **GO_ COMMAND**. Operation type is selected in the previous step (written in the WB Command Register).

READ_COMMAND is used for reading the content of the WB Command Register.

Note: Address written in the Command Register is incremented automatically after each successful access. At the end of the operation it points to the first data after the last accessed. To improve the download or upload performance, many "GO_COMMAND" instructions can be executed in a row. Command Register in that case needs to be set only once at the startup. In case of an error on the WB bus, address is not incremented.

The following table shows all supported commands and their codes:

Command	Code
GO_COMMAND	0x0
READ_COMMAND	0x1
WRITE_COMMAND	0x2

Table 10: WISHBONE sub-module: Supported commands

4.2.1 WISHBONE Read or Write operation

To perform the WISHBONE read or write operation, the debug must be enabled (instruction DEBUG needs to be activated in the TAP (refer to the IEEE 1149.1 Test Access Port documentation for more information)) and the WISHBONE sub-module selected (see description on page 16 for more details).

WISHBONE read or write operation is performed in two steps:

- Writing the address, data size and type of operation to the WB Command Register. This is done by issuing the WRITE_COMMAND (see section 4.2.2 WRITE_COMMAND on page 21 for more details)
- Issuing the GO_COMMAND (see section 4.2.4.1 GO_COMMAND when read cycle is requested on page 23 for more details when performing a read operation or 4.2.4.2 GO_COMMAND when write cycle is requested on page 25 when performing write operation)

First instruction sets the address, type of operation and data size that needs to be read/written.

Second instruction performs the read/write operation on the WISHBONE bus.

Both instructions (WRITE_COMMAND and GO_COMMAND) return 4-bit status. The status should be checked on-the-fly to verify that the instruction was completed successfully. Status bits are also protected with the 32-bit CRC. In case of errors different steps need to be performed:

- CRC error: WB sub-module device didn't accept the instruction from the debugger through the JTAG. Instruction needs to be repeated.
- WISHBONE error: GO_COMMAND needs to be repeated. Address is already set to the right value.
- Overrun/underrun:
    - When performing write operation the WB device was to slow. Data was not written on time and overrun occurred.
    - When performing read operation the WB device was too slow. Data was not read out on time and underrun occurred.

  In both cases GO_COMMAND instruction needs to be repeated. See also section 4.2.6 Accessing slow devices on page 28.

4.2.2 WRITE_COMMAND

WRITE_COMMAND writes the address, the type of operation and the size of the data that will be read or written to the WB Command register. Command is performed by shifting the following data through the data scan chain:

- 1-bit with value 0
- 4-bit instruction WRITE_COMMAND (0x2) (MSB shifted first)
- 4-bit access type (read or write, 8-bit, 16-bit or 32-bit) (MSB shifted first)
- 32-bit address (MSB shifted first)
- 16-bit size (MSB shifted first)
- 32-bit CRC (MSB shifted first) that is protecting the incoming data (first 57 bits).
- 36 bits with value 0 (this value is ignored in the debug interface)

While the WRITE_COMMAND instruction is shifted-in, the following data is shifted out:

- 89 bits with value 0 (this value should be ignored)
- 4-bit status (MSB shifted first)
    - 1 when CRC error occurs
    - always 0
    - always 0 (1 when WB error occurs. This can only occur with the "GO_COMMAND")
    - always 0 (1 when overrun/underrun occurs. This can only occur with the "GO_COMMAND")
- 32-bit CRC (MSB shifted first) that is protecting the outgoing data (four bits). Only status bits are protected with this CRC (first 89 bits are ignored).

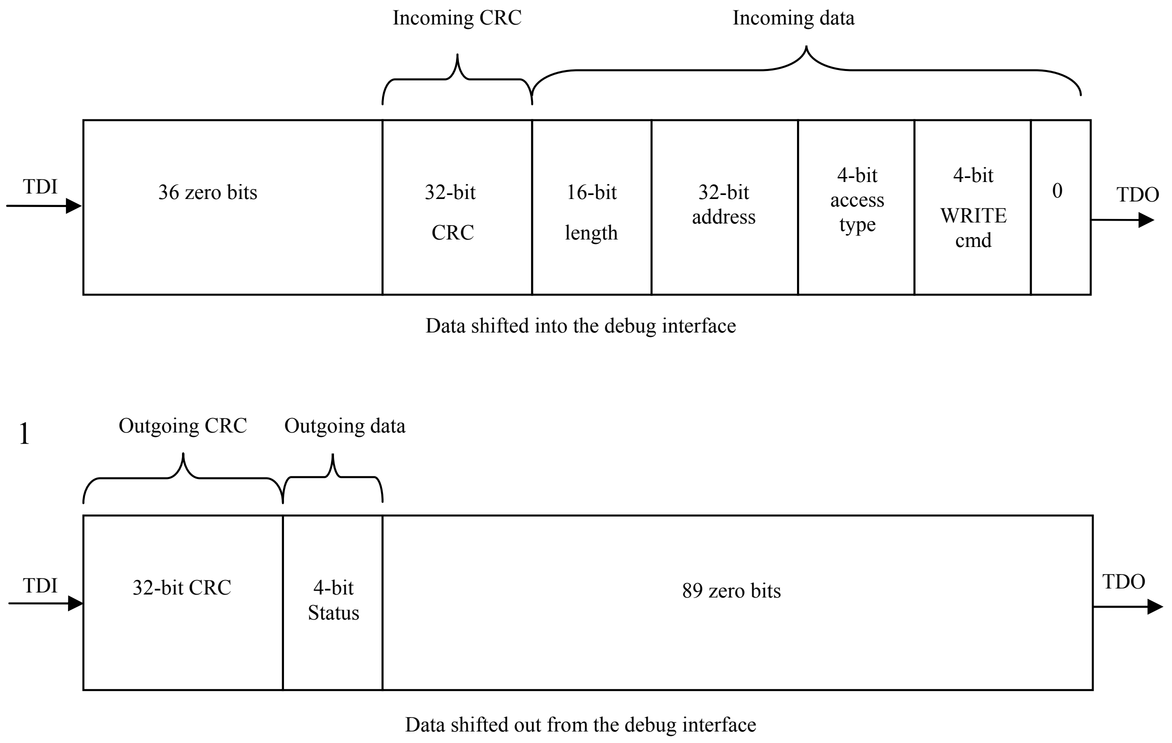


Figure 2: WRITE_COMMAND

4.2.3 READ_COMMAND

READ_COMMAND reads the address, the type of operation and the size of the data from the WB Command register. Command is identical to the WRITE_COMMAND with one exception, READ_COMMAND (0x1) needs to be used instead of the WRITE_COMMAND. See section 4.2.2 WRITE_COMMAND for more details.

4.2.4 GO_COMMAND

GO_COMMAND executes what is written in the Command register. The structure of the GO_COMMAND differs depending on the access type written in the WB Command register. There are two possibilities:

- WISHBONE read cycle is requested (8, 16 or 32-bit)
- WISHBONE write cycle is requested (8, 16 or 32-bit)

4.2.4.1 GO_COMMAND when read cycle is requested

GO_COMMAND when read cycle is requested performs the read operation on the WISHBONE bus. Address, cycle type and data size are specified in the WB Command register. The GO_COMMAND is performed by shifting the following data through the data scan chain:

- 1-bit with value 0 (This bit is treated as a "module select" when set to 1)
- 4-bit instruction GO_COMMAND (0x0) (MSB shifted first)
- 32-bit CRC (MSB shifted first) that is protecting the incoming data (first 5 bits).
- ((data size + 1) x 8 + 36) bits with value 0 (this value is ignored in the debug interface).

While the GO_COMMAND is shifted-in, the following data is shifted out:

- 37 bits with value 0 (this value should be ignored)
- ((data size + 1) x 8) bits of data
- 4-bit status (MSB shifted first)
    - 1 when CRC error occurs
    - always 0
    - 1 when WB error occurs.
    - 1 when overrun/underrun occurs.

- 32-bit CRC (MSB shifted first) that is protecting the outgoing data ((data size + 1) x 8 + 5 bits). Only outgoing data bits are protected with this CRC (first 37 bits are ignored).

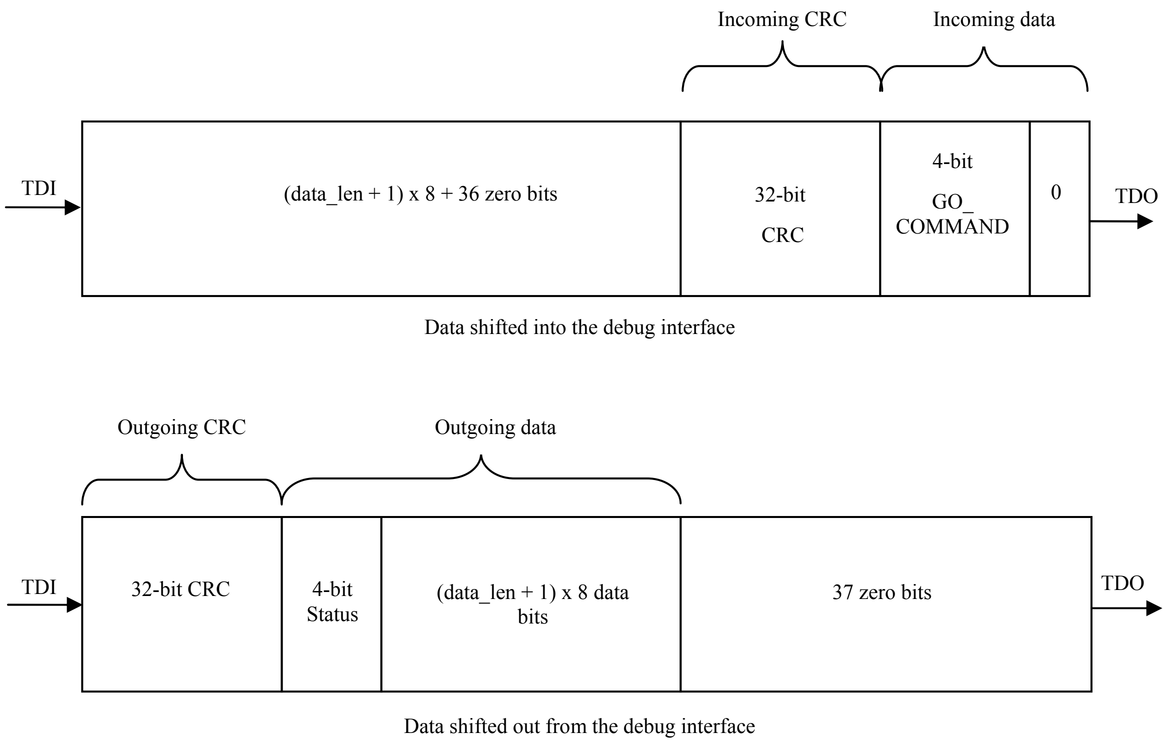

Incoming CRC
Incoming data
TDI
(data_len + 1) x 8 + 36 zero bits
32-bit CRC
4-bit GO_ COMMAND
0
TDO
Data shifted into the debug interface
Outgoing CRC
Outgoing data
TDI
32-bit CRC
4-bit Status
(data_len + 1) x 8 data bits
37 zero bits
TDO
Data shifted out from the debug interface

Figure 3: GO_COMMAND when read cycle is requested

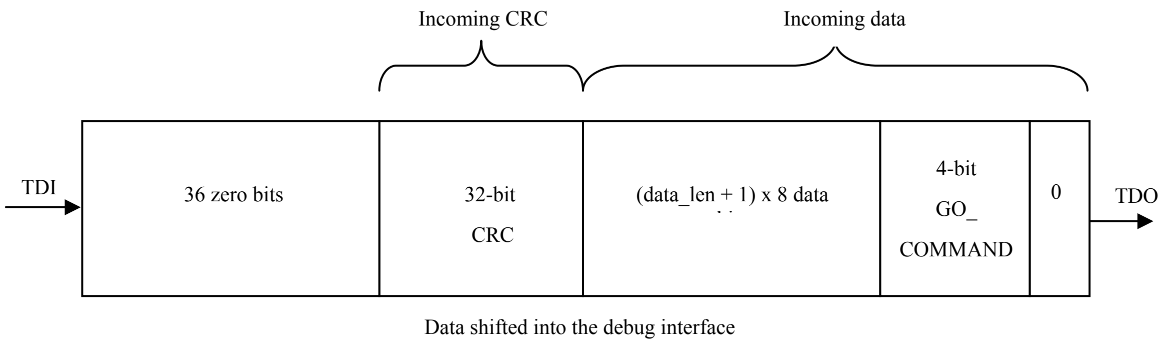
4.2.4.2 GO_COMMAND when write cycle is requested

GO_COMMAND when write cycle is requested performs the write operation on the WISHBONE bus. Address, cycle type and data size are specified in the WB Command register. The GO_COMMAND is performed by shifting the following data through the data scan chain:

- 1-bit with value 0
- 4-bit instruction GO_COMMAND (0x0) (MSB shifted first)
- ((data size + 1) x 8) bits of data (MSB shifted first)
- 32-bit CRC (MSB shifted first) that is protecting the incoming data ((first data + 1) size x 8 + 5 bits).
- 36 bits with value 0 (these bits are ignored in the debug interface)

While the GO_COMMAND is shifted-in, the following data is shifted out:

- ((data size +1) x 8 + 37) bits with value 0 (this value should be ignored)
- 4-bit status
  - 1 when CRC error occurs
  - always 0
  - 1 when WB error occurs.
  - 1 when overrun/underrun occurs.
- 32-bit CRC (MSB shifted first) that is protecting the outgoing data (4 status bits). Only outgoing data bits are protected with this CRC (first (data size + 1) x 8 + 37 bits are ignored).



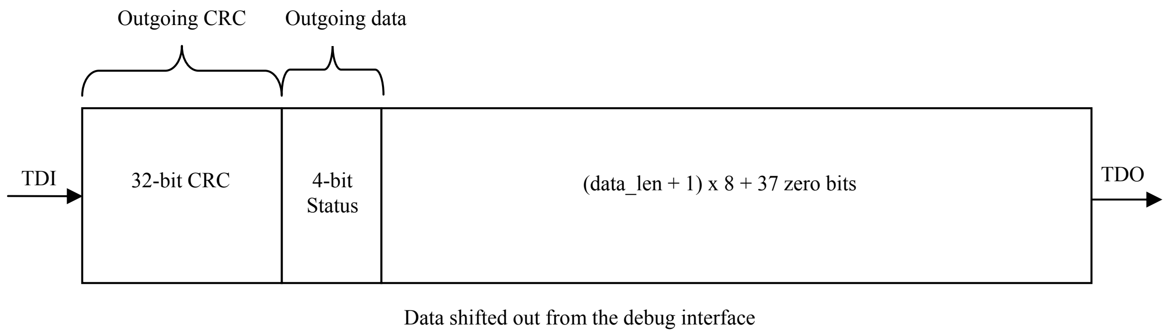

Outgoing CRC
Outgoing data
TDI
32-bit CRC
4-bit Status
(data_len + 1) x 8 + 37 zero bits
TDO
Data shifted out from the debug interface

Figure 4: GO_COMMAND when write cycle is requested

4.2.5 Data and select signals

Data in the WISHBONE sub-module is organized in the big endian byte ordering. Following section describes the data and select signals depending on the address and type of operation (32-bit, 16-bit and 8-bit).

32-bit access (wb_adr_o[1:0] = 00b):

wb_sel_o[3:0] = 1111b

wb_dat_x[31:0] are used

16-bit access (wb_adr_o[1:0] = 00b):

wb_sel_o[3:0] = 1100b

wb_dat_x[31:16] are used

16-bit access (wb_adr_o[1:0] = 10b):

wb_sel_o[3:0] = 0011b

wb_dat_x[15:0] are used

8-bit access (wb_adr_o[1:0] = 00b):

wb_sel_o[3:0] = 1000b

wb_dat_x[31:24] are used

8-bit access (wb_adr_o[1:0] = 01b):

wb_sel_o[3:0] = 0100b

wb_dat_x[23:16] are used

8-bit access (wb_adr_o[1:0] = 10b):

wb_sel_o[3:0] = 0010b

wb_dat_x[15:8] are used

8-bit access (wb_adr_o[1:0] = 11b):

wb_sel_o[3:0] = 0001b

wb_dat_x[7:0] are used

4.2.6 Accessing slow devices

Usually the WISHBONE clock (wb_clk_i) is much faster than the JTAG clock (tck_i). In that case read or write accesses are finished on time. However it is possible to do a read or write access to a WISHBONE device that is not fast enough to complete the desired operation on time.

On time means:

- Read operation needs to be finished before the data is shifted out through the JTAG
- Write operation must be finished before the next write is started.

4.2.6.1 Reading from slow device

Following needs to be done to read the data from a slow device:

- Perform the WRITE_COMMAND normally.
- Perform only first part of the GO_COMMAND. After the first 37 bits are shifted out, force TAP state machine to go to the PAUSE_DR state. This means that the tms_i signal needs to be driven high after the 36th bit.
- Once in the PAUSE_DR state, tdo_o signal reflects the state of the WISHBONE bus. While bus is busy (read cycle not finished), tdo_o is set to 1. Once the read cycle is finished, tdo_o goes to zero. Loop in the PAUSE_DR state until tdo_o goes to zero. Then go to the SHIFT_DR state and continue like nothing happened. When reading more data, go to the PAUSE_DR state after each word (half or byte) is shifted out (depending on the type of access (8, 16 or 32 bit)).

CRC is not calculated when not in the SHIFT_DR state.

Note: TAP state machine is described in the documentation that is part of the project “JTAG Test Access Port (TAP)” that is available on the opencores website.

4.2.6.2 Writing to slow device

Following needs to be done to write the data to a slow device:

- Perform the WRITE_COMMAND normally.
- Perform only part of the GO_COMMAND. After the first 5 bits are shifted in, shift in the first data word (half or byte). Then force the TAP state machine to go to the PAUSE_DR state. This means that the tms_i signal needs to be driven high after the 36th bit for 32-bit access, 20th bit for 16-bit access or 12th bit for 8-bit access.
- Once in the PAUSE_DR state, tdo_o signal reflects the state of the WISHBONE bus. While bus is busy (write cycle not finished), tdo_o is set to 1. Once the write cycle is finished, tdo_o goes to zero. Loop in the PAUSE_DR state until tdo_o goes to zero. Then go to the SHIFT_DR state and continue like nothing happened. When writing more data, go to the PAUSE_DR state after each data word/half/byte is shifted in.
- Check the busy status also after the last data word (half or byte). Then shift out the status and the CRC

CRC is not calculated when not in the SHIFT_DR state.

Note: TAP state machine is described in the documentation that is part of the project “JTAG Test Access Port (TAP)” that is available on the opencores website.

4.2.7 Errors in the WISHBONE sub-module

Both, the data that is shifted-in and the data that is shifted-out are protected with the 32-bit CRC. The incoming-CRC is checked by the WISHBONE sub-module. Whenever the CRC error is detected, status bit status[3] is set to 1. CRC is checked for all commands.

Outgoing CRC should be checked by the debugger.

When the module select command is performed and a non-existing module is selected, status bit status[2] is set to 1. For all other commands this bit is always set to 0.

When a WISHBONE cycle is terminated with an error (signal wb_err_i is set to 1), the status bit [1] is set to 1. This can only happen when the GO_COMMAND is performed. For all other commands this bit is set to 0. When WB error occurs, address is not incremented but points to the address when the error occured.

When the addressed WISHBONE device replies to late to the request, overrun (for write cycles) or underrun (for read cycles) occurs. The status bit [0] is set to 1. If this happens too often, see the section 4.2.6 Accessing slow devices on page 28.

4.3 CPU Sub-module

CPU sub-module is an interface to the CPU debug facilities (that are part of the CPU). It consists of the Command Register, Control Register and the CPU interface. The CPU sub-module can access them all. CPU sub-module can only perform 32-bit read and write accesses.

There are two steps needed to write/read the data to/from the CPU that is connected to the CPU sub-module:

- Address, data size and the type of operation need to be written to the Command Register (see section 3.2.1 CPU Command Register on page 14 for more details about the CPU command register). This is done by using the **WRITE_COMMAND**.
- CPU read or write operation (32-bit) is executed with the **GO_COMMAND**. Operation type is selected in the previous step.

READ_COMMAND is used for reading the content of the CPU Command Register.

Note: Address written in the Command Register is incremented automatically after each successful access. At the end of the operation it points to the first data after the last accessed. To improve the download or upload performance, many GO_COMMAND instructions can be executed in a row. Command Register in that case needs to be set only once at the startup. In case of an error, address is not incremented.

Stalling and resetting of the CPU is done through the CPU Control register:

- **WRITE_CONTROL** command is used to change the value in the CPU Control register.
- **READ_CONTROL** command is used to read the content of the CPU Control Register.

The following table shows all supported commands and their codes.

Command	Code
GO_COMMAND	0x0
READ_COMMAND	0x1
WRITE_COMMAND	0x2
READ_CONTROL	0x3
WRITE_CONTROL	0x4

Table 11: CPU sub-module: Supported commands

4.3.1 CPU Read or Write operation

To perform a read or write operation from/to a CPU, the debug must be enabled (instruction DEBUG needs to be activated in the TAP (refer to the IEEE 1149.1 Test Access Port documentation for more information)) and the CPU sub-module selected (see description on page 16 for more details).

CPU read or write operation is performed in two steps:

- Writing the address, data size and type of operation to the CPU Command Register. This is done by issuing the WRITE_COMMAND (see section 4.3.3 WRITE_COMMAND on page 35 for more details)
- Issuing the GO_COMMAND (see section 4.3.5.1 GO_COMMAND when read cycle is requested on page 37 for more details when performing a read operation or 4.3.5.2 GO_COMMAND when write cycle is requested on page 39 when performing write operation)

First instruction sets the address, type of operation and data size that needs to be read/written.

Second instruction performs the read/write operation from-to the CPU.

Both instructions (WRITE_COMMAND and GO_COMMAND) return 4-bit status. The status should be checked on-the-fly to verify that the instruction was completed successfully. Status bits are also protected with the 32-bit CRC. In case of errors different steps need to be performed:

- CRC error: CPU sub-module didn’t accept the instruction from the debugger through JTAG. Instruction needs to be repeated.
- Overrun/underrun:
    - When performing write operation the CPU was to slow. Data was not written on time and overrun occurred.
    - When performing read operation the CPU was too slow. Data was not read out on time and underrun occurred.

  In both cases GO_COMMAND needs to be repeated. See also section 4.3.10 Accessing slow devices on page 44.

4.3.2 CPU Control Register Read or Write

To perform a read or write operation from/to the CPU CONTROL Register, the debug must be enabled (instruction DEBUG needs to be activated in the TAP (refer to the IEEE 1149.1 Test Access Port documentation for more information)) and the CPU sub-module selected (see description on page 16 for more details).

CPU Control Register read or write operation is performed by using the WRITE_CONTROL instruction (see section 4.3.3 WRITE_COMMAND on page 35 for more details)

Instruction WRITE_CONTROL returns a 4-bit status (3 bits are reserved). The status should be checked on-the-fly to verify that the instruction was completed successfully. Status bits are also protected with the 32-bit CRC. In case of errors different steps need to be performed:

- CRC error: CPU sub-module didn't accept the instruction from the debugger through JTAG. Instruction needs to be repeated.

4.3.3 WRITE_COMMAND

WRITE_COMMAND writes the address, the type of operation and the size of the data that will be read or written to the CPU Command register. Command is performed by shifting the following data through the data scan chain:

- 1-bit with value 0
- 4-bit instruction WRITE_COMMAND (0x2) (MSB shifted first)
- 4-bit access type (read or write) (MSB shifted first)
- 32-bit address (MSB shifted first)
- 16-bit size (MSB shifted first)
- 32-bit CRC (MSB shifted first) that is protecting the incoming data (first 57 bits).
- 36 bits with value 0 (this value is ignored in the debug interface)

While the WRITE_COMMAND is shifted-in, the following data is shifted out:

- 89 bits with value 0 (this value should be ignored)
- 4-bit status (MSB shifted first)
  - 1 when CRC error occurs
  - always 0
  - always 0
  - always 0 (1 when overrun/underrun occurs. This can only occur with the “GO_COMMAND”)
- 32-bit CRC (MSB shifted first) that is protecting the outgoing data (four bits). Only status bits are protected with this CRC (first 89 bits are ignored).

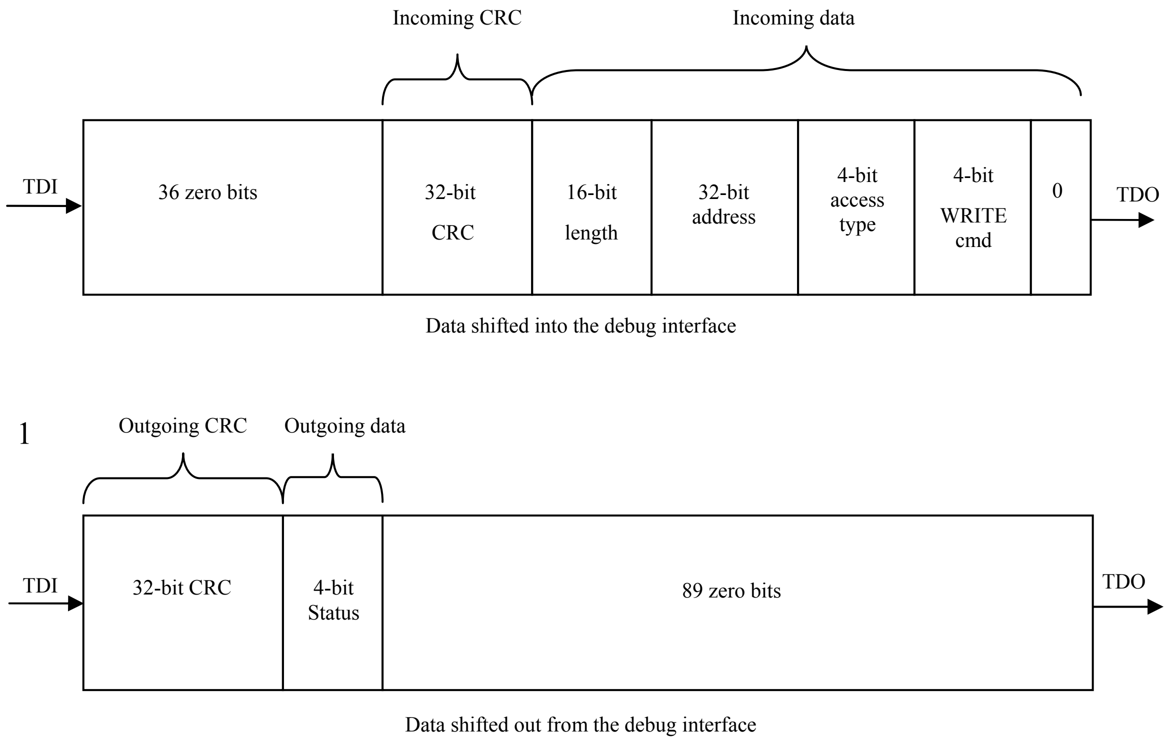


Figure 5: WRITE_COMMAND

4.3.4 READ_COMMAND

READ COMMAND reads the address, the type of operation and the size of the data from the CPU Command register. Command is identical to the WRITE_COMMAND with one exception, READ_COMMAND (0x1) needs to be used instead of the WRITE_COMMAND. See section 4.3.3 WRITE_COMMAND for more details.

4.3.5 GO_COMMAND

GO_COMMAND executes what is written in the Command register. The structure of the GO_COMMAND differs depending on the access type written in the CPU Command register. There are two possibilities:

- Read cycle is requested
- Write cycle is requested

4.3.5.1 GO_COMMAND when read cycle is requested

GO_COMMAND when read cycle is requested performs the read operation to the CPU. Address, cycle type and data size are specified in the CPU Command register. The GO_COMMAND is performed by shifting the following data through the data scan chain:

- 1-bit with value 0 (This bit is treated as a “module select” when set to 1)
- 4-bit instruction GO_COMMAND (0x0) (MSB shifted first)
- 32-bit CRC (MSB shifted first) that is protecting the incoming data (first 5 bits).
- ((data size + 1) x 8 + 36) bits with value 0 (this value is ignored in the debug interface).

While the GO_COMMAND is shifted-in, the following data is shifted out:

- 37 bits with value 0 (this value should be ignored)
- ((data size + 1) x 8) bits of data
- 4-bit status (MSB shifted first)
    - 1 when CRC error occurs
    - always 0
    - always 0
    - 1 when overrun/underrun occurs.

- 32-bit CRC (MSB shifted first) that is protecting the outgoing data ((data size + 1) x 8 + 5 bits). Only outgoing data bits are protected with this CRC (first 37 bits are ignored).

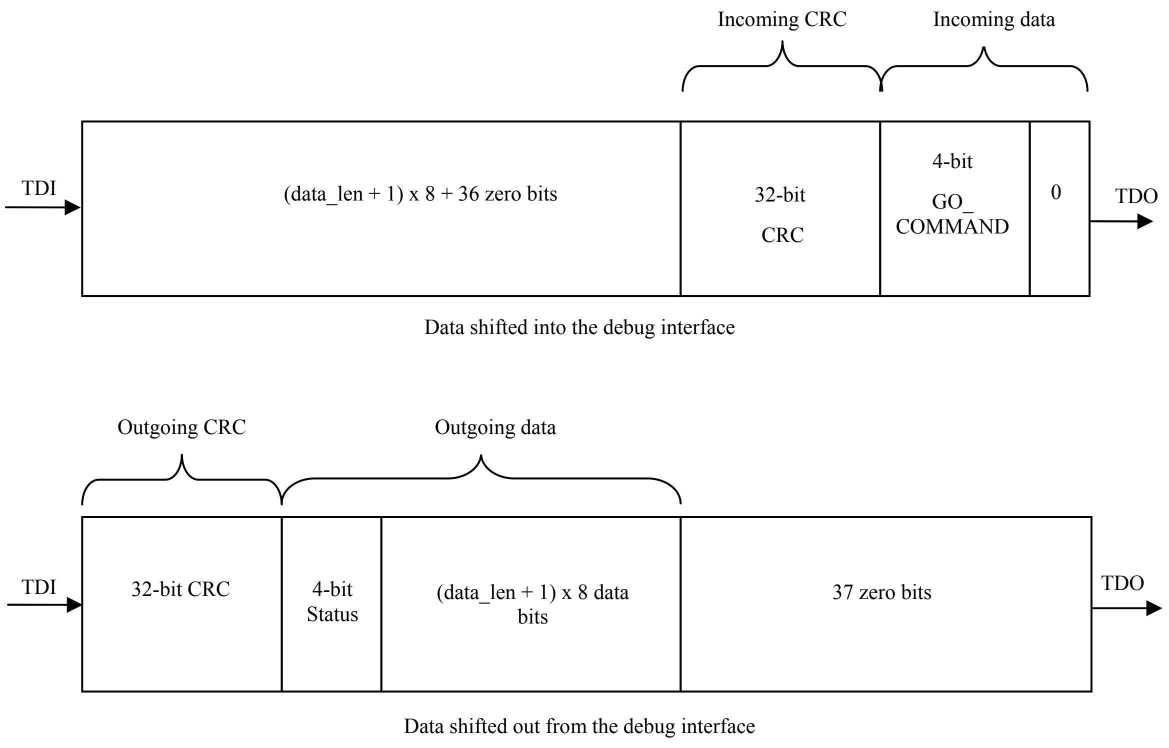

Incoming CRC
Incoming data
TDI
(data_len + 1) x 8 + 36 zero bits
32-bit CRC
4-bit GO_ COMMAND
0
TDO
Data shifted into the debug interface
Outgoing CRC
Outgoing data
TDI
32-bit CRC
4-bit Status
(data_len + 1) x 8 data bits
37 zero bits
TDO
Data shifted out from the debug interface

Figure 6: GO_COMMAND when read cycle is requested

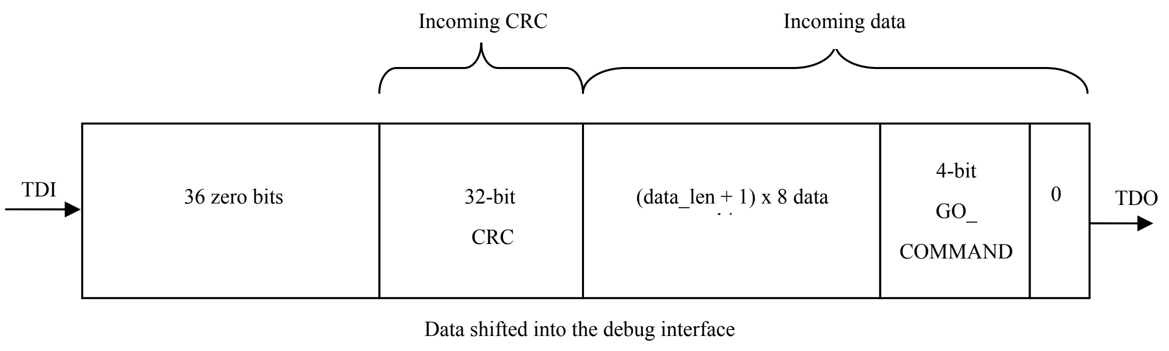
4.3.5.2 GO_COMMAND when write cycle is requested

GO_COMMAND when write cycle is requested performs the write operation to the CPU. Address, cycle type and data size are specified in the CPU Command register. The GO_COMMAND is performed by shifting the following data through the data scan chain:

- 1-bit with value 0
- 4-bit instruction GO_COMMAND (0x0) (MSB shifted first)
- ((data size + 1) x 8) bits of data (MSB shifted first)
- 32-bit CRC (MSB shifted first) that is protecting the incoming data ((first data + 1) size x 8 + 5 bits).
- 36 bits with value 0 (these bits are ignored in the debug interface)

While the GO_COMMAND is shifted-in, the following data is shifted out:

- ((data size +1) x 8 + 37) bits with value 0 (this value should be ignored)
- 4-bit status
  - 1 when CRC error occurs
  - always 0
  - always 0
  - 1 when overrun/underrun occurs.
- 32-bit CRC (MSB shifted first) that is protecting the outgoing data (4 status bits). Only outgoing data bits are protected with this CRC (first (data size + 1) x 8 + 37 bits are ignored).



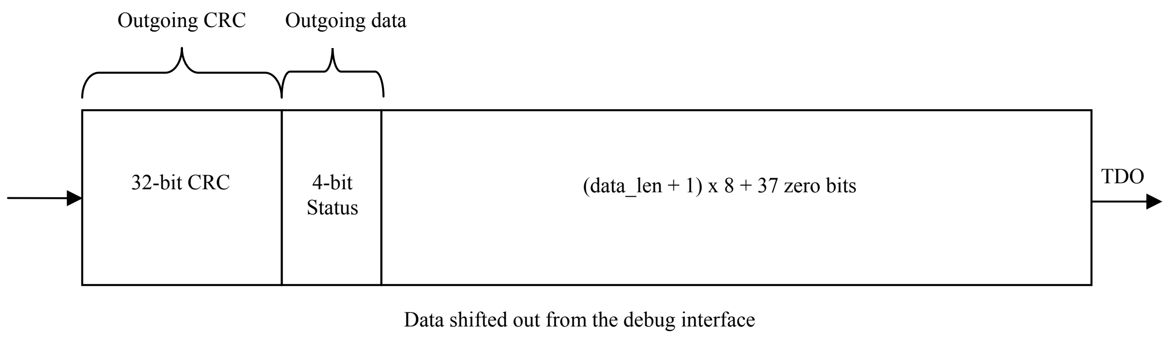

Outgoing CRC
Outgoing data
32-bit CRC
4-bit Status
(data_len + 1) x 8 + 37 zero bits
TDO
Data shifted out from the debug interface

Figure 7: GO_COMMAND when write cycle is requested

4.3.6 WRITE_CONTROL

WRITE_CONTROL writes the data to the CPU Control register (see 3.2.2 CPU Control Register on page 15). Command is performed by shifting the following data through the data scan chain:

- 1-bit with value 0
- 4-bit instruction WRITE_CONTROL (0x4) (MSB shifted first)
- 52-bit data (MSB shifted first)
- 32-bit CRC (MSB shifted first) that is protecting the incoming data (first 57 bits).
- 36 bits with value 0 (this value is ignored in the debug interface)

While the WRITE_CONTROL is shifted-in, the following data is shifted out:

- 89 bits with value 0 (this value should be ignored)
- 4-bit status (MSB shifted first)
    - 1 when CRC error occurs
    - always 0
    - always 0
    - always 0
- 32-bit CRC (MSB shifted first) that is protecting the outgoing data (four bits). Only status bits are protected with this CRC (first 89 bits are ignored).

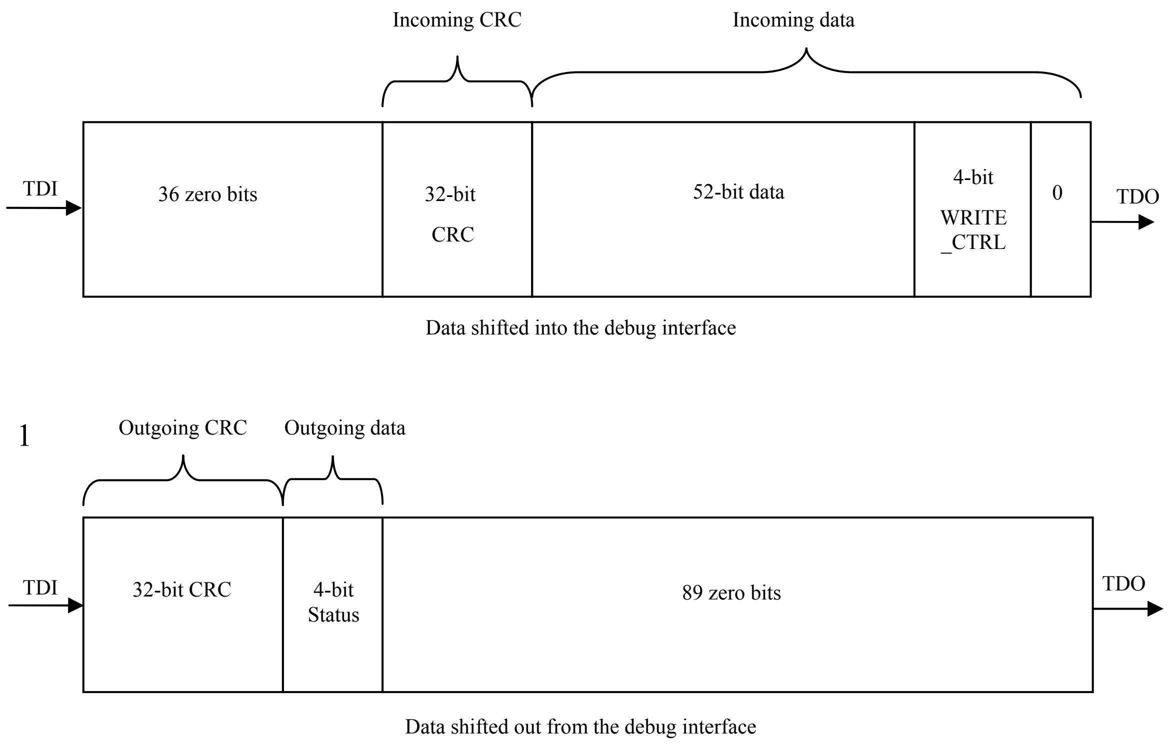


Figure 8: WRITE_CONTROL

4.3.7 READ_CONTROL

READ_CONTROL reads the data from the CPU Control register (see 3.2.2 CPU Control Register on page 15). Command is identical to the WRITE_CONTROL with one exception, READ_CONTROL (0x3) needs to be used instead of the WRITE_CONTROL. See section 4.3.6 WRITE_CONTROL for more details.

4.3.8 Stalling CPU

The CPU can be stalled in two ways:

- By deliberately setting bit STALL bit in the CPU Control Register to 1 (see section 3.2.2 CPU Control Register on page 15 for more details). Clearing this bit again unstalls the CPU.
- An input breakpoint signal (cpu_bp_i) automatically stops the CPU and sets bit STALL of the CPU Control Register to 1. Clearing this bit again unstalls the CPU.

For more information about changing the value of the CPU Control register go to the section 4.3.6 WRITE_CONTROLon page 41.

Reading the value of the CPU Control register is described in section 4.3.7 READ_CONTROL on page 42.

For more information about the breakpoint generation refer to the CPU manual (i.e. OpenRISC 1000 System Architecture Manual).

4.3.9 Resetting CPU

The Debug Interface puts the CPU to reset by setting the RESET bit in the CPU Control Register to 1. Clearing this bit to 0 deactivates the reset signal.

For more information about changing the value of the CPU Control register go to the section 4.3.6 WRITE_CONTROLon page 41.

Reading the value of the CPU Control register is described in section 4.3.7 READ_CONTROL on page 42.

4.3.10 Accessing slow devices

Usually the CPU clock (cpu_clk_i) is much faster than the JTAG clock (tck_i). In that case read or write accesses are finished on time. However it is possible to do a read or write access to a CPU that is not fast enough to complete the desired operation on time.

On time means:

- Read operation needs to be finished before the data is shifted out through the JTAG
- Write operation must be finished before the next write is started.

4.3.10.1 Reading from slow CPU

Following needs to be done to read the data from a slow CPU:

- Perform the WRITE_COMMAND normally.
- Perform only first part of the GO_COMMAND. After the first 37 bits are shifted out, force TAP state machine to go to the PAUSE_DR state. This means that the tms_i signal needs to be driven high after the 36th bit.
- Once in the PAUSE_DR state, tdo_o signal reflects the state of the CPU bus. While bus is busy (read cycle not finished), tdo_o is set to 1. Once the read cycle is finished, tdo_o goes to zero. Loop in the PAUSE_DR state until tdo_o goes to zero. Then go to the SHIFT_DR state and continue like nothing happened. When reading more data, go to the PAUSE_DR state after each word is shifted out.

CRC is not calculated when not in the SHIFT_DR state.

Note: TAP state machine is described in the documentation that is part of the project "JTAG Test Access Port (TAP)" that is available on the opencores website.

4.3.10.2 Writing to slow CPU

Following needs to be done to write the data to a slow CPU:

- Perform the WRITE_COMMAND normally.
- Perform only part of the GO_COMMAND. After the first 5 bits are shifted in, shift in the first data word. Then force the TAP state machine to go to the PAUSE_DR state. This means that the tms_i signal needs to be driven high after the 36th bit.
- Once in the PAUSE_DR state, tdo_o signal reflects the state of the CPU bus. While bus is busy (write cycle not finished), tdo_o is set to 1. Once the write cycle is finished, tdo_o goes to zero. Loop in the PAUSE_DR state until tdo_o goes to zero. Then go to the SHIFT_DR state and continue like nothing happened. When writing more data, go to the PAUSE_DR state after each data word is shifted in.
- Check the busy status also after the last data word (half or byte). Then shift out the status and the CRC

CRC is not calculated when not in the SHIFT_DR state.

Note: TAP state machine is described in the documentation that is part of the project "JTAG Test Access Port (TAP)" that is available on the opencores website.

5 Architecture

The SoC Debug Interface architecture is based on IEEE Std. 1149.1 Standard Test Access Port and Boundary Scan Architecture. Other signals are added to provide additional flexibility.

The interface consists of several parts (blocks):

- Logic that selects one of the connected sub modules. Currently three sub-modules are available, CPU0, CPU1 and WISHBONE.
- CRC sub-module that checks incoming data.
- CRC sub-module that calculates the CRC for the outgoing data.
- WISHBONE sub-module
- 2 CPU sub-modules

As seen on the following figure, debug interface is just one part of the complete debugging system. For more information about the TAP controller, go to the opencores web site and look for the project "JTAG Test Access Port (TAP)". There is a complete IP core with test bench and documentation available.

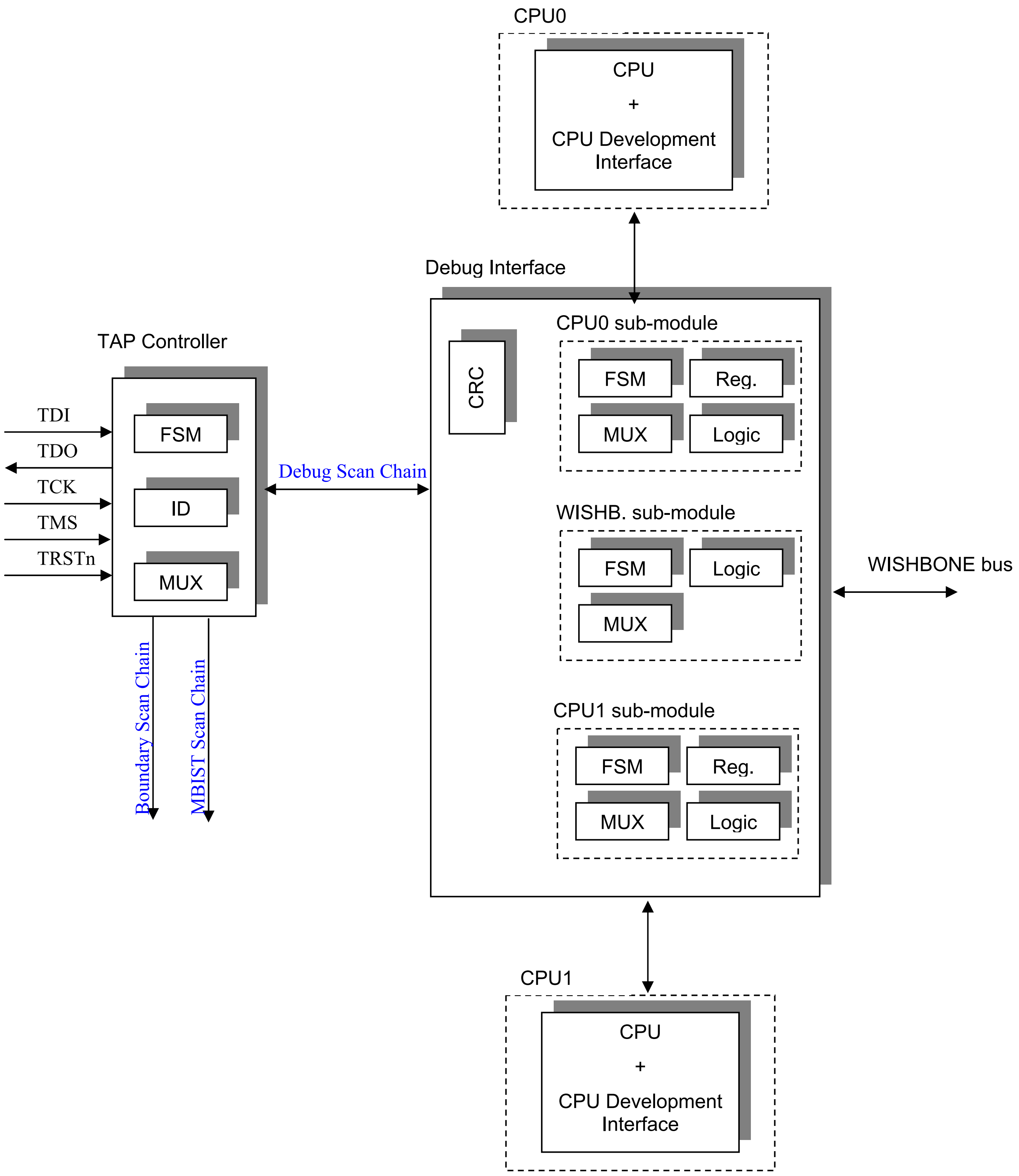


Figure 9: Complete system

5.1 Debug Interface

Debug Interface is an interface between the TAP controller and the sub-modules that are target specific (CPU, WISHBONE...). It receives data from the TAP whenever the DEBUG instruction is active (see IEEE 1149.1 Test Access Port documentation).

Data can hold two kinds of instructions:

- Module select instruction
- Sub-module instruction (of the selected sub-module)

First bit of the instruction is used to distinguish between the module select instruction (first bit is 1) **and the sub-module instruction** (first bit is 0).

Module select instruction is used for selecting/enabling the sub-module.

Sub-module instructions are sub-module specific. Each sub-module can use different instructions. Because of this, it is very easy to add additional sub-modules.

All the data (in both directions) is protected with the 32-bit CRC (see section 5.2 CRC sub-module on page 48 for more information). Both CRC engines (one for incoming data and one for outgoing data) are located in the debug interface. None of the sub-modules have their own CRC engine.

5.2 CRC sub-module

There are two CRC sub-modules in the debug interface. One is checking the incoming data, while the other is calculating the CRC from the outgoing data.

The following polynomial is used for 32-bit CRC calculation:

1 + x1 + x2 + x4 + x5 + x7 + x8 + x10 + x11 + x12 + x16 + x22 + x23 + x26 + x32

1-bit data input is used for CRC calculation. CRC is initialized to the value 0xffffffff before the actual CRC calculation starts. The CRC is received/send with the MSB shifted first.

Incoming CRC is calculated from the incoming data.

Outgoing CRC is calculated from the outgoing data. CRC calculation does not include zero bits that are shifted out while incoming data and incoming CRC are shifting in (See **Figure 1** on page 18 for example).

5.3 WISHBONE sub-module

Is capable of doing the 8-bit, 16-bit and 32-bit read and write accesses. All accesses are single accesses since the data flow through the TAP (JTAG) is slow and there is no need for bursts. Wishbone clock frequency must be higher than the TCK frequency. See section 4.2 WISHBONE Sub-module on page 19 for more information about the WISHBONE sub-module.

5.4 CPU sub-module

Is capable of doing the 32-bit read and write accesses. CPU clock frequency must be higher than the TCK frequency. See section 4.3 CPU Sub-module on page 31 for more information about the CPU sub-module.