SoC Debug Interface

Author: Igor Mohor
IgorM@opencores.org

Rev. 2.2

March 18, 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 debug interface. Document name

Rev.	Date	Author	Description
			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.

Contents

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”
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_stall_all_o	1	output	CPU stall all (all unselected CPUs are stalled)
cpu_stb_o	1	output	CPU strobe
cpu_sel_o	8	output	CPU select signals (one hot), select the CPU
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_rst_i	1	input	WISHBONE reset
wb_ack_i	1	input	WISHBONE acknowledge indicates a normal cycle

Port	Width	Direction	Description
			termination
wb_adr_o	32	output	WISHBONE address output
wb_cyc_o	1	output	WISHBONE cycle encapsulates a valid transfer 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 doesn't have internal registers.

CPU sub-module does have internal registers and they are described in the following section.

3.1 CPU Registers List

Name	Address	Width	Access	Description
CPU_OP	0x0	8	R/W	CPU Operation Register
CPU_SEL	0x1	8	R/W	CPU Select Register

Table 4: CPU Register List

3.2 CPU Operation Register

Bit #	Access	Description
7:3		Reserved
2	R/W	CPUSTALLALL – Stall all unselected CPUs 0 = normal operation 1 = Stall all unselected CPUs
1	R/W	RESET – Reset CPU 0 = normal 1 = reset
0	R/W	CPUSTALL – CPU Stall 0 = normal operation 1 = Stall CPU. CPU can also set this bit by inserting the cpu_bp_i signal.

Table 5: CPU_OP Register

Reset Value:

CPU_OP: 00h

3.3 CPU Select Register

Bit #	Access	Description
7:0	R/W	CPUSEL – Select one CPU 00000001b = Selects CPU 0 (cpu_sel_o [0] is active) 00000010b = Selects CPU 1 (cpu_sel_o [1] is active) 00000100b = Selects CPU 2 (cpu_sel_o [2] is active) 00001000b = Selects CPU 3 (cpu_sel_o [3] is active) 00010000b = Selects CPU 4 (cpu_sel_o [4] is active) 00100000b = Selects CPU 5 (cpu_sel_o [5] is active) 01000000b = Selects CPU 6 (cpu_sel_o [6] is active) 10000000b = Selects CPU 7 (cpu_sel_o [7] is active) 00000000b = NO CPU selected (cpu_sel_o [7:0] not active)

Table 6: CPU_SEL Register

Reset Value:

CPU_SEL: 00h

4

Operation

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

4.1 Chain Selection

The debug interface is just an interface between the sub-module that is target specific and the TAP controller. Currently two sub-modules are connected to the debug interface, WISHBONE sub-module and CPU sub-module. Up to 8 sub-modules can be connected to the debug interface.

Chain (module) name	Chain Code
CPU Debug Chain	000
WISHBONE Debug Chain	001

Table 7: Supported sub-modules

First thing to do is to select the sub-module. This is done with the chain select instruction. Following needs to be done prior to the chain 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 "chain select" instruction needs to be shifted-in through the TAP data chain:

- 1-bit with value 1 (This bit is treated as a "chain select")
- 3-bit chain ID (LSB 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 "chain select" instruction is shifted-in, the following data is shifted out:

- 36 bits with value 0 (this value should be ignored)
- 4-bit status
    - 1 if incoming CRC is OK, else 0
    - 1 if command was "chain select", else 0
    - 1 if non-existing chain was selected, else 0
    - always 1
- 32-bit CRC (MSB shifted first) that is protecting the outgoing data (four bits). Only status bits are protected with this CRC (first 36 bits are ignored).

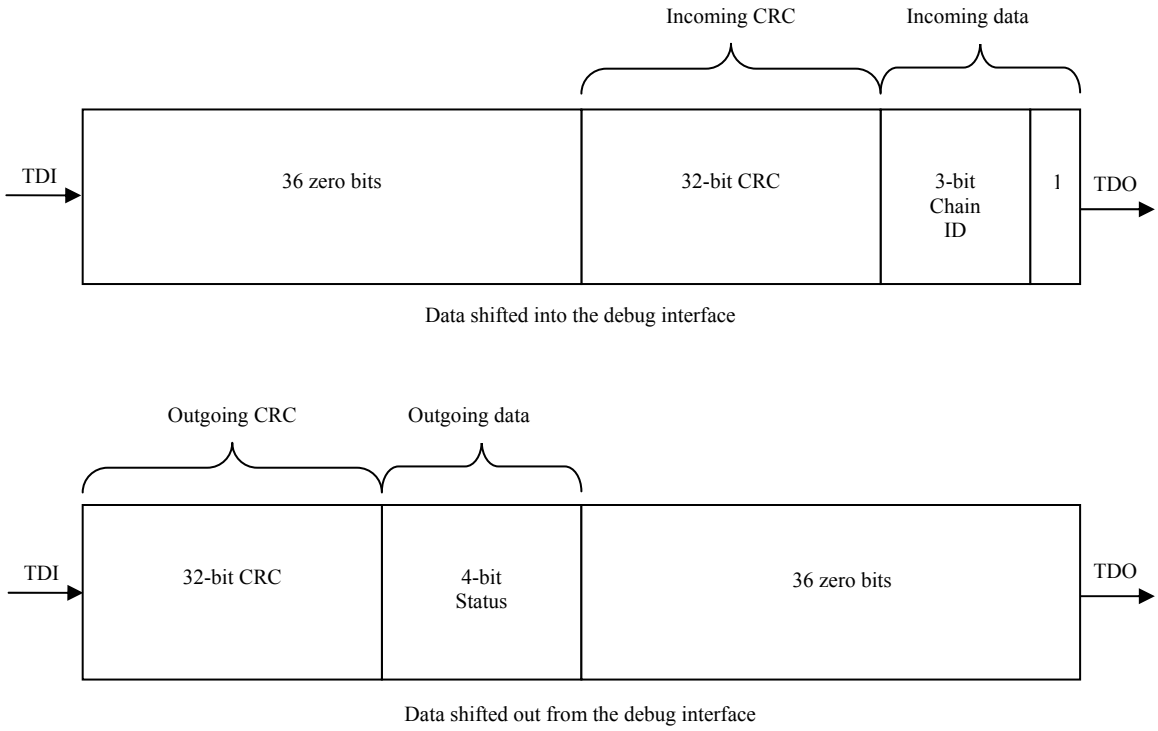

Figure 1: Chain Selection

See section 5.2 CRC sub-module on page 49 for more details about the CRC.

4.2 WISHBONE Sub-module

There are three types of instruction in the WISHBONE sub-module (see Table 8: WISHBONE sub-module: Supported Instructions below for more details):

- READx/WRITEx instructions
- GO instruction
- STATUS instruction

The following table describes all supported instructions and their codes:

Instruction	Code	Meaning
STATUS	000	Reads the status of the previous command(s)
WRITE8	001	Requests write of the 8-bit data
WRITE16	010	Requests write of the 16-bit data
WRITE32	011	Requests write of the 32-bit data
GO	100	Performs the requested instruction
READ8	101	Requests read of the 8-bit data
READ16	110	Requests read of the 16-bit data
READ32	111	Requests read of the 32-bit data

Table 8: WISHBONE sub-module: Supported Instructions

All WISHBONE operations (except STATUS) consist of two consequent instructions. First instruction is the READx/WRITEx instruction. It sets the address, type of instruction (read or write) and length of data that is read or written. Second instruction is the GO instruction that actually does what the first instruction requests. Following section describes how read, write or status operations are performed.

Note: Before some data can be read from or written to the WISHBONE, the following needs to be done:

- instruction DEBUG needs to be activated in the TAP (refer to the IEEE 1149.1 Test Access Port documentation for more information)
- WISHBONE sub-module needs to be selected (refer to section 4.1 Chain Selection on page 14 for more details)

4.2.1 WISHBONE Read operation

To perform the WISHBONE read 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 14 for more details).

Read operation is performed in two steps:

- Issuing the READx instruction (see section 4.2.3 READx/WRITEx instruction on page 20 for more details)
- Issuing the GO instruction (see section 4.2.4.1 GO with READx requested on page 22 for more details)

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

Second instruction performs the read operation on the WISHBONE bus.

Both instructions (READx and GO) return status. This status should be checked on-the-fly to verify that the instruction was completed successfully. However it is also possible to check the status after the operation by issuing the STATUS instruction. Go to the section 4.2.5 STATUS instruction on page 26 for more information about the STATUS.

Whenever an error occurs, both steps (READx and GO) need to be repeated.

4.2.2 WISHBONE Write operation

To perform the WISHBONE 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 14 for more details).

Write operation is performed in two steps:

- Issuing the WRITEx instruction (see section 4.2.3 READx/WRITEx instruction on page 20 for more details)
- Issuing the GO instruction (see section 4.2.4.2 GO with WRITEx requested on page 24 for more details)

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

Second instruction performs the write operation on the WISHBONE bus.

Both instructions (WRITEx and GO) return status. This status should be checked on-the-fly to verify that the instruction was completed successfully. However it is also possible to check the status after the operation by issuing the STATUS instruction. Go to the section 4.2.5 STATUS instruction on page 26 for more information about the STATUS.

Whenever an error occurs, both steps (WRITEx and GO) need to be repeated.

4.2.3 READx/WRITEx instruction

Read/write instruction sets the address, the type of operation and the length of the data that will be read or written. It is performed by shifting the following data through the data scan chain:

- 1-bit with value 0
- 3-bit instruction (READ8, READ16, READ32, WRITE8, WRITE16 or WRITE32, depending on the cycle type (read or write) and the data width (8-bit, 16-bit or 32-bit))
- 32-bit address
- 16-bit length (describes data length in bytes)
- 32-bit CRC (MSB shifted first) that is protecting the incoming data (first 52 bits).
- 36 bits with value 0 (this value is ignored in the debug interface)

While the "read/write" instruction is shifted-in, the following data is shifted out:

- 84 bits with value 0 (this value should be ignored)
- 4-bit status
    - 1 if incoming CRC is OK, else 0
    - 3 bit-s of 0
- 32-bit CRC (MSB shifted first) that is protecting the outgoing data (four bits). Only status bits are protected with this CRC (first 84 bits are ignored).

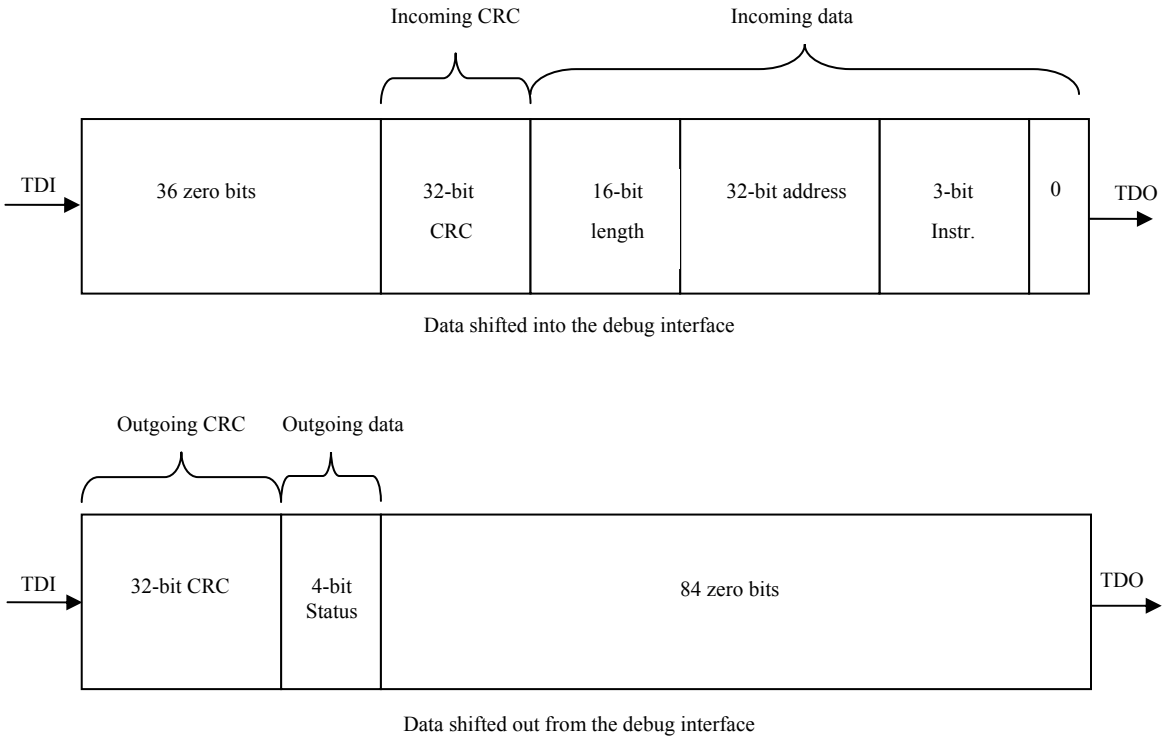

Figure 2: READx/WRITEx instruction

4.2.4 GO instruction

GO instruction actually performs the instruction that was requested with the previous READx/WRITEx instruction. The structure of the GO instruction differs depending on the previous requested instruction. There are two possibilities:

- READx instruction was previously requested
- WRITEx instruction was previously requested

4.2.4.1 GO with READx requested

GO instruction with READx previously requested performs the read operation on the WISHBONE bus. Address, cycle type and data length are specified with the READx instruction. The GO instruction is performed by shifting the following data through the data scan chain:

- 1-bit with value 0
- 3-bit instruction GO
- 32-bit CRC (MSB shifted first) that is protecting the incoming data (first 4 bits).
- (data length x 8) bits with value 0 (this value is ignored in the debug interface). Number x is number of bits that are read out.

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

- 36 bits with value 0 (this value should be ignored)
- (data length x 8) bits of data
- 4-bit status
    - 1 if incoming CRC is OK, else 0
    - 1 if WISHBONE cycle didn't finish (still in progress), else 0. This is important only for the first data byte
    - 1 if under run occurred (data couldn't be read fast enough), else 0
    - 1 if WISHBONE error occurred

- 32-bit CRC (MSB shifted first) that is protecting the outgoing data (data length x 8 + 4 bits). Only outgoing data bits are protected with this CRC (first 36 bits are ignored).

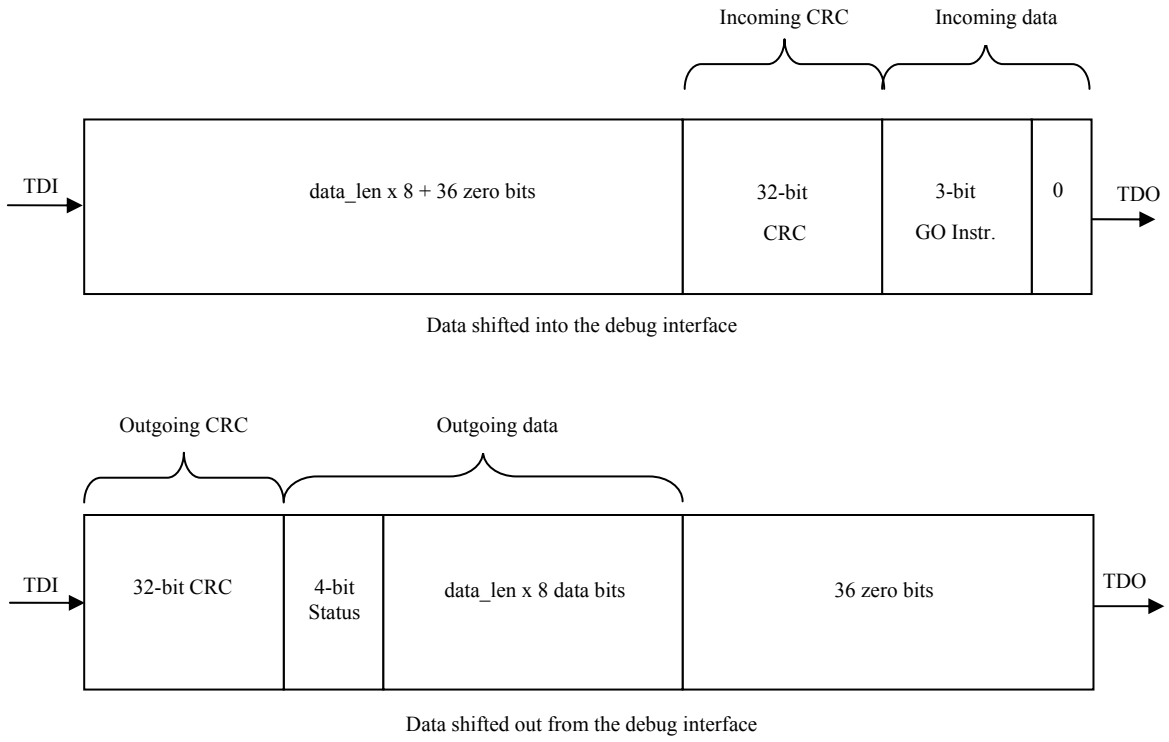

Figure 3: GO with READx requested

4.2.4.2 GO with WRITEx requested

GO instruction with WRITEx previously requested performs the write operation on the WISHBONE bus. Address, cycle type and data length are specified with the WRITEx instruction. The GO instruction is performed by shifting the following data through the data scan chain:

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

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

- (length x 8 + 36) bits with value 0 (this value should be ignored)
- 4-bit status
    - 1 if incoming CRC is OK, else 0
    - 1 if WISHBONE cycle didn't finish (still in progress), else 0. This is important only for the first data byte
    - 1 if overrun occurred (data couldn't be write fast enough), else 0
    - 1 if WISHBONE error occurred
- 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 length x 8 + 36 bits are ignored).

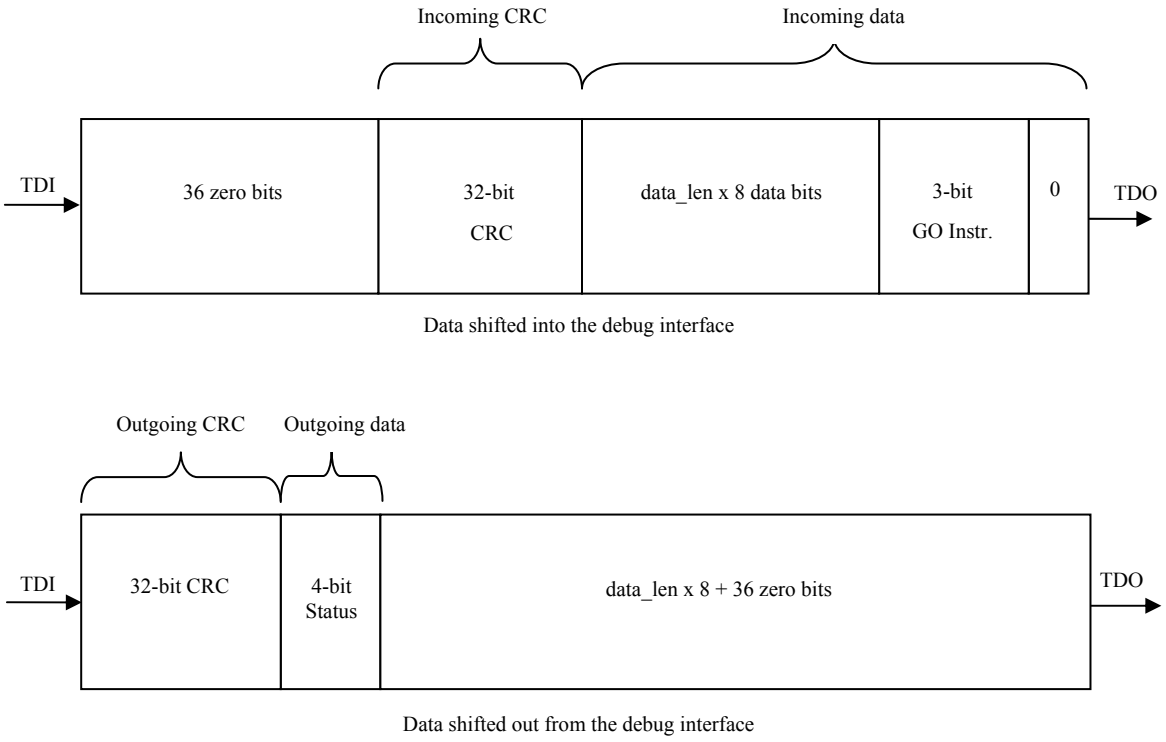

Figure 4: GO with WRITEx requested

4.2.5 STATUS instruction

STATUS instruction is used for reading the value of the status register. Status register consists of four bits. The MSB bit is always shifted out first. The meaning of the bits is described in the following table.

Status bit #	Meaning
3	0 = Incoming CRC error 1 = Incoming CRC OK
2	0 = WISHBONE cycle finished 1 = WISHBONE cycle not finished
1	0 = no overrun/underrun 1 = overrun/underrun occurred
0	0 = No WISHBONE error 1= WISHBONE error occurred

Table 9: Status bit description

Bit 0 is set when at least one WISHBONE error occurred since the last STATUS instruction was performed. Once set, it can be cleared only with the STATUS instruction.

Bit 1 is set when at least one underrun (for read operation) or overrun (for write) happened since the last STATUS instruction was performed. Underrun is when data is not available for reading on time. Overrun is when data is not written to the device on time (new data arrives before the previous data is written). Once set, it can be cleared only with the STATUS instruction.

Bit 2 is set while the WISHBONE cycle is in progress (not finished, yet). Go to section 4.2.7 Accessing slow devices on page 30 to read how to access data from slow devices. This bit changes when the access is finished (is not latched).

Bit 3 is cleared to zero when the CRC error occurs while shifting the data into the debug interface. This bit is valid only for the last instruction. STATUS instruction does not change the value of this bit.

The STATUS instruction is performed by shifting the following data through the data scan chain:

- 1-bit with value 0
- 3-bit instruction STATUS
- 32-bit CRC (MSB shifted first) that is protecting the incoming data (first 4 bits).
- 36 bits with value 0 (these bits are ignored in the debug interface)

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

- 36 bits with value 0 (this value should be ignored)
- 4-bit status (see **Table 9** above for more details)
- 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 36 bits are ignored).

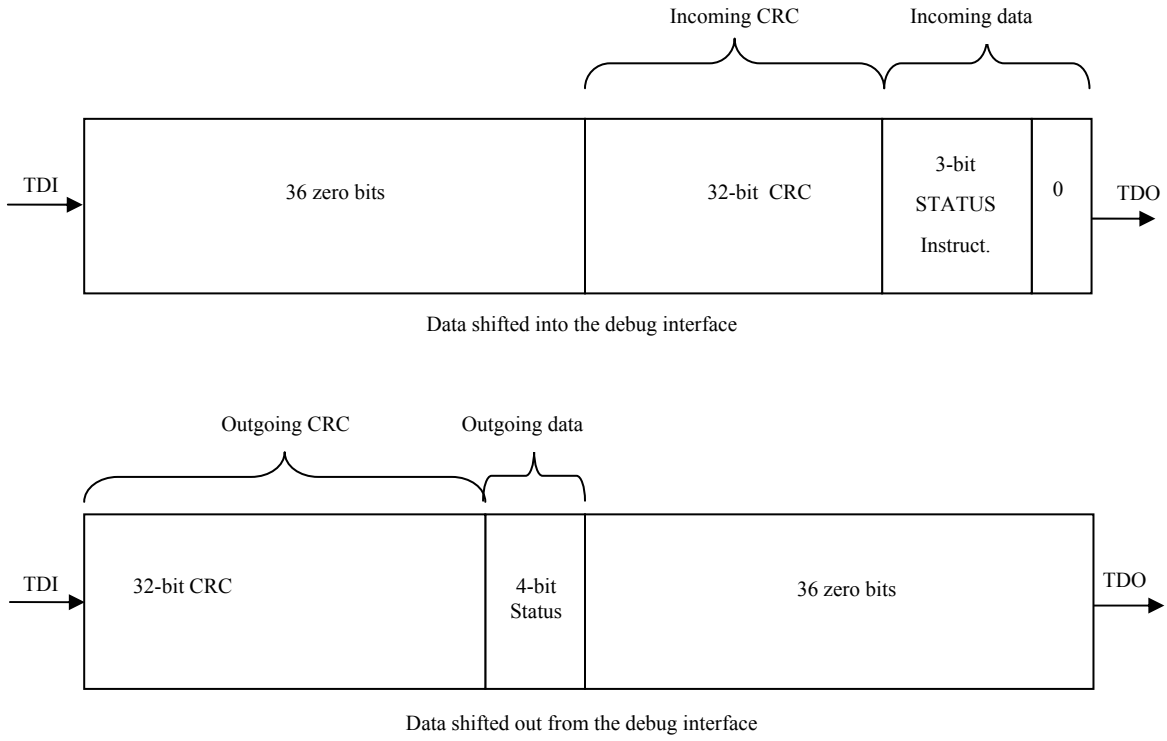

Figure 5: STATUS Instruction

4.2.6 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

4.2.7 Accessing slow devices

Usually the WISHBONE clock (wb_clk_i) is much slower 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.

4.2.7.1 Reading from slow device

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

- Perform the READx instruction normally
- Perform only first part of the GO instruction. After the first 36 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 35th 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/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.

4.2.7.2 Writing to slow device

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

- Perform the WRITEx instruction normally.
- Perform only part of the GO instruction. After the first 4 bits are shifted in, shift in the first data word/half/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 35th bit for 32-bit access, 19th bit for 16-bit access or 11th 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/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.

4.2.8 Error and retry on the WISHBONE

When a WISHBONE cycle is terminated with an error (signal wb_err_i is set to 1), the status bit [0] is set to 1. This bit remains set until the status is read with the STATUS instruction. This is useful to detect errors that occurred during several consecutive cycles (reading/writing data from/to consecutive addresses).

When the addresses WISHBONE device replies to late to the request, status bit [1] (overrun/underrun) is set. This bit remains set until the status is read with the STATUS instruction. This is useful to detect errors that occurred during several consecutive cycles.

WISHBONE retry function is not supported.

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 internal registers and the CPU interface.

The CPU sub-module can access both, internal CPU registers and the CPU interface (CPU's on the other side of the debug interface).

Internal registers are used for:

- selecting one of the connected CPUs
- resetting the CPU(s)
- stalling the selected CPU
- stalling all unselected CPUs

Before the CPU sub-module can be accessed, several steps need to be taken:

- 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))
- CPU sub-module must be selected (see description of the 4.1 Chain Selection section on page 14 for more details).

The following table describes all supported instructions and their codes.

Instruction	Code	Meaning
CPU_WRITE8	001	Requests write of the 8-bit data to the CPU.
CPU_WRITE32	010	Requests write of the 32-bit data to the CPU.
CPU_WRITE_REG	011	Requests write of the 8-bit data to the internal CPU register.
CPU_GO	100	Performs the requested instruction.
CPU_READ8	101	Requests read of the 8-bit data from the CPU.
CPU_READ32	110	Requests read of the 32-bit data from the CPU.
CPU_READ_REG	111	Requests read of the 8-bit data from the internal CPU register.

Table 10: CPU sub-module: Supported Instructions

4.3.1 CPU Register Read operation

To read the data from the CPU register (internal register that is part of the CPU sub-module), 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 of the 4.1 Chain Selection section on page 14 for more details).

Read operation is performed in two steps:

- Issuing the CPU_READ_REG instruction (see section 4.3.5 CPU_READx/CPU_WRITEx operation on page 39 for more details)
- Issuing the CPU_GO instruction (see section 4.3.6.1 GO with CPU_READx requested on page 41 for more details)

First instruction sets the address of the register. Second instruction performs the read operation to the addressed internal CPU register.

Both instructions (CPU_READ_REG and CPU_GO) return status. This status should be checked on-the-fly to verify that the instruction was completed successfully.

Whenever an error occurs, both steps (CPU_READ_REG and CPU_GO) need to be repeated.

4.3.2 CPU Register Write operation

To write the data from the CPU register (internal register that is part of the CPU sub-module), 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 of the 4.1 Chain Selection section on page 14 for more details).

Write operation is performed in two steps:

- Issuing the CPU_WRITE_REG instruction (see section 4.3.5 CPU_READx/CPU_WRITEx operation on page 39 for more details)
- Issuing the CPU_GO instruction (see section 4.3.6.2 GO with CPU_WRITEx requested on page 43 for more details)

First instruction sets the address of the register. Second instruction performs the write operation to the addressed internal CPU register.

Both instructions (CPU_WRITE_REG and CPU_GO) return status. This status should be checked on-the-fly to verify that the instruction was completed successfully.

Whenever an error occurs, both steps (CPU_WRITE_REG and CPU_GO) need to be repeated.

4.3.3 CPU Read operation

To read the data from the CPU, following needs to be done:

- 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))
- CPU sub-module must selected (see description of the 4.1 Chain Selection section on page 14 for more details)
- CPU must be selected. Selection is made through the write operation to the internal CPU register (See section 4.3.7 Selecting different CPUs on page 45 for more details).

Read operation is performed in two steps:

- Issuing the CPU_READ8 instruction when 8-bit read is needed or CPU_READ32 instruction when 32-bit read is needed (see section 4.3.5 CPU_READx/CPU_WRITEx operation on page 39 for more details).
- Issuing the CPU_GO instruction (see section 4.3.6.1 GO with CPU_READx requested on page 41 for more details)

First instruction sets the address of the CPU where the data will be read from. Second instruction performs the read operation to the addressed CPU space.

Both instructions (CPU_READ8/32 and CPU_GO) return status. This status should be checked on-the-fly to verify that the instruction was completed successfully.

Whenever an error occurs, both steps (CPU_READ8/32 and CPU_GO) need to be repeated.

4.3.4 CPU Write operation

To write the data to the CPU, following needs to be done:

- 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))
- CPU sub-module must selected (see description of the 4.1 Chain Selection section on page 14 for more details)
- CPU must be selected. Selection is made through the write operation to the internal CPU register (See section 4.3.7 Selecting different CPUs on page 45).

Write operation is performed in two steps:

- Issuing the CPU_WRITE8 instruction when 8-bit write is needed or CPU_WRITE32 instruction when 32-bit read is needed (see section 4.3.5 CPU_READx/CPU_WRITEx operation on page 39 for more details).
- Issuing the CPU_GO instruction (see section 4.3.6.2 GO with CPU_WRITEx requested on page 43 for more details)

First instruction sets the address of the CPU where the data will be written to. Second instruction performs the write operation to the addressed CPU space.

Both instructions (CPU_WRITE8/32 and CPU_GO) return status. This status should be checked on-the-fly to verify that the instruction was completed successfully.

Whenever an error occurs, both steps (CPU_WRITE8/32 and CPU_GO) need to be repeated.

4.3.5 CPU_READx/CPU_WRITEx operation

CPU_READ8, CPU_READ32, CPU_WRITE8 and CPU_WRITE32 instructions set the address of the CPU where the data that will be read from or written to.

CPU_READ_REG and CPU_WRITE_REG instructions set the address of the internal CPU register (that is part of the CPU sub-module) where the data that will be read from or written to.

All mentioned instructions are performed by shifting the following data through the data scan chain:

- 1-bit with value 0
- 3-bit instruction (CPU_READ8, CPU_READ32, CPU_READ_REG, CPU_WRITE8, CPU_WRITE32 or CPU_WRITE_REG) depending on the cycle type (32-bit or 8-bit access to the CPU or 8-bit access to the CPU register))
- 32-bit address
- 32-bit CRC (MSB shifted first) that is protecting the incoming data (first 36 bits).
- 36 bits with value 0 (this value is ignored in the debug interface)

While the “CPU_READx/CPU_WRITEx” instruction is shifted-in, the following data is shifted out:

- 68 bits with value 0 (this value should be ignored)
- 4-bit status
    - 1 if incoming CRC is OK, else 0
    - 3 bit-s 010b
- 32-bit CRC (MSB shifted first) that is protecting the outgoing data (four bits). Only status bits are protected with this CRC (first 68 bits are ignored).

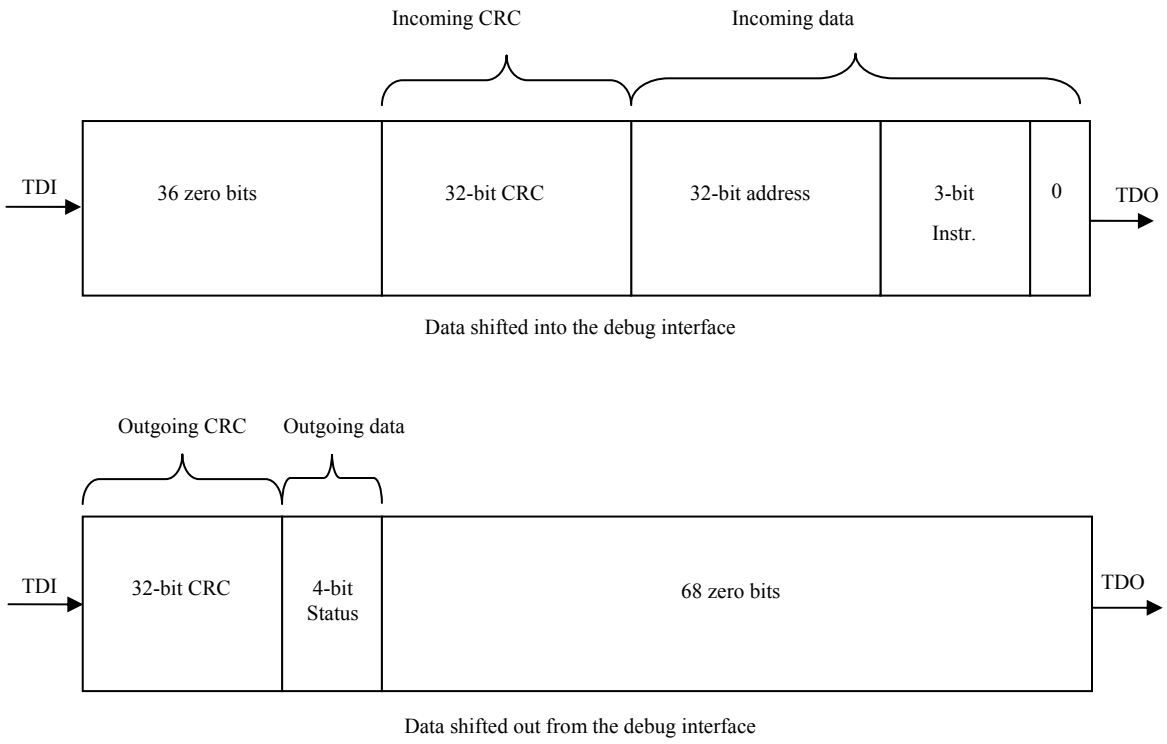

Figure 6: CPU_READx/CPU_WRITEx instruction

4.3.6 GO instruction

GO instruction actually performs the instruction that was requested with the previous CPU_READx/CPU_WRITEx instruction. The structure of the GO instruction differs depending on the previous requested instruction. There are two possibilities:

- CPU_READx instruction was previously requested
- CPU_WRITEx instruction was previously requested

4.3.6.1 GO with CPU_READx requested

GO instruction with CPU_READx previously requested performs the read operation to the:

- internal CPU register (that is part of the CPU sub-module)

or

- CPU that is connected to the CPU sub-module (CPU must be previously selected. See section 4.3.7 Selecting different CPUs on page 45)

Address of the internal CPU register or the location in the CPU is specified with the previous CPU_READx instruction. The GO instruction is performed by shifting the following data through the data scan chain:

- 1-bit with value 0
- 3-bit instruction GO
- 32-bit CRC (MSB shifted first) that is protecting the incoming data (first 4 bits).
- x bits with value 0 (this value is ignored in the debug interface). Number x is number of bits that are read out.
    - 36 + 32 for CPU_READ32
    - 36 + 8 for CPU_READ8 or CPU_READ_REG

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

- 36 bits with value 0 (this value should be ignored)

- x bits of data
  - x is 32 for CPU_READ32
  - x is 8 for CPU_READ8 or CPU_READ_REG
- 4-bit status
  - 1 if incoming CRC is OK, else 0
  - 3 bit-s 010b
- 32-bit CRC (MSB shifted first) that is protecting the outgoing data (32(8) + 4 bits). Only outgoing data bits are protected with this CRC (first 36 bits are ignored).

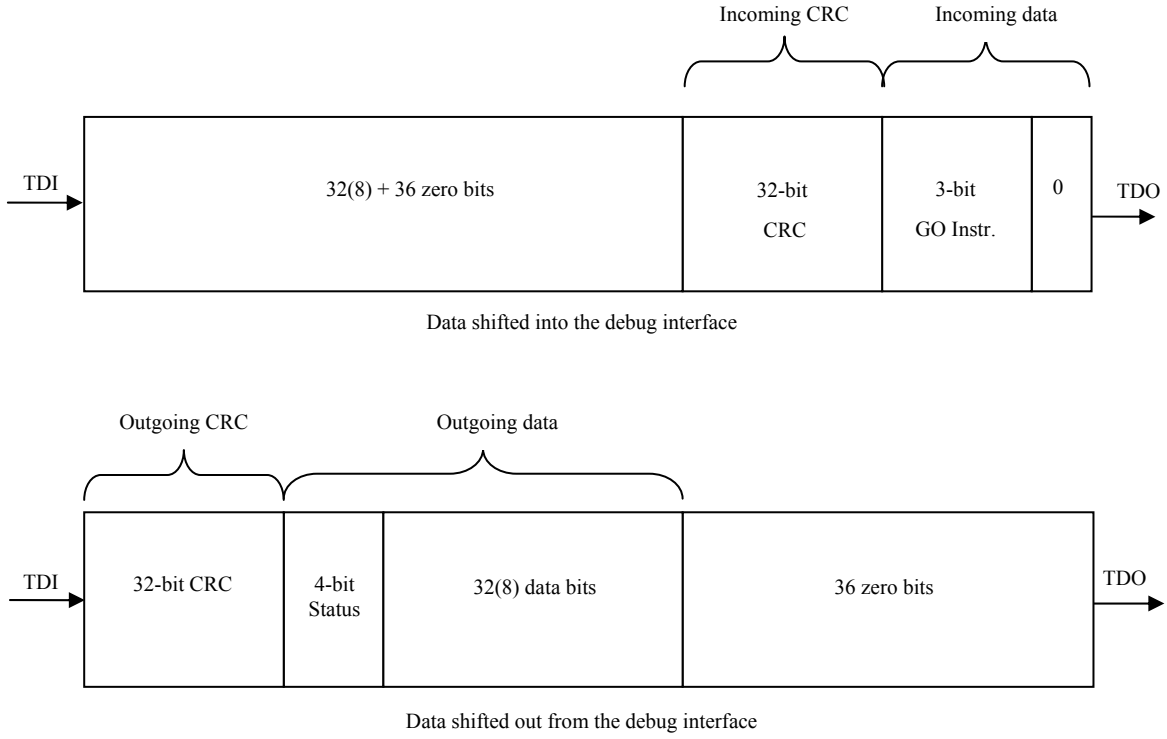

Figure 7: GO with CPU_READx requested

4.3.6.2 GO with CPU_WRITEx requested

GO instruction with CPU_WRITEx previously requested performs the write operation to the:

- internal CPU register (that is part of the CPU sub-module)

or

- CPU that is connected to the CPU sub-module (CPU must be previously selected. See section 4.3.7 Selecting different CPUs on page 45)

Address of the internal CPU register or the location in the CPU is specified with the previous CPU_WRITEx instruction. The GO instruction is performed by shifting the following data through the data scan chain:

- 1-bit with value 0
- 3-bit instruction GO
- x bits of data
    - x is 32 for CPU_READ32
    - x is 8 for CPU_READ8 or CPU_READ_REG
- 32-bit CRC (MSB shifted first) that is protecting the incoming data (first 32(8) + 4 bits).
- 36 bits with value 0 (these bits are ignored in the debug interface)

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

- 36 + 32(8) bits with value 0 (this value should be ignored)
- 4-bit status
    - 1 if incoming CRC is OK, else 0
    - 3 bit-s 010b
- 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 36 + 32(8) bits are ignored).

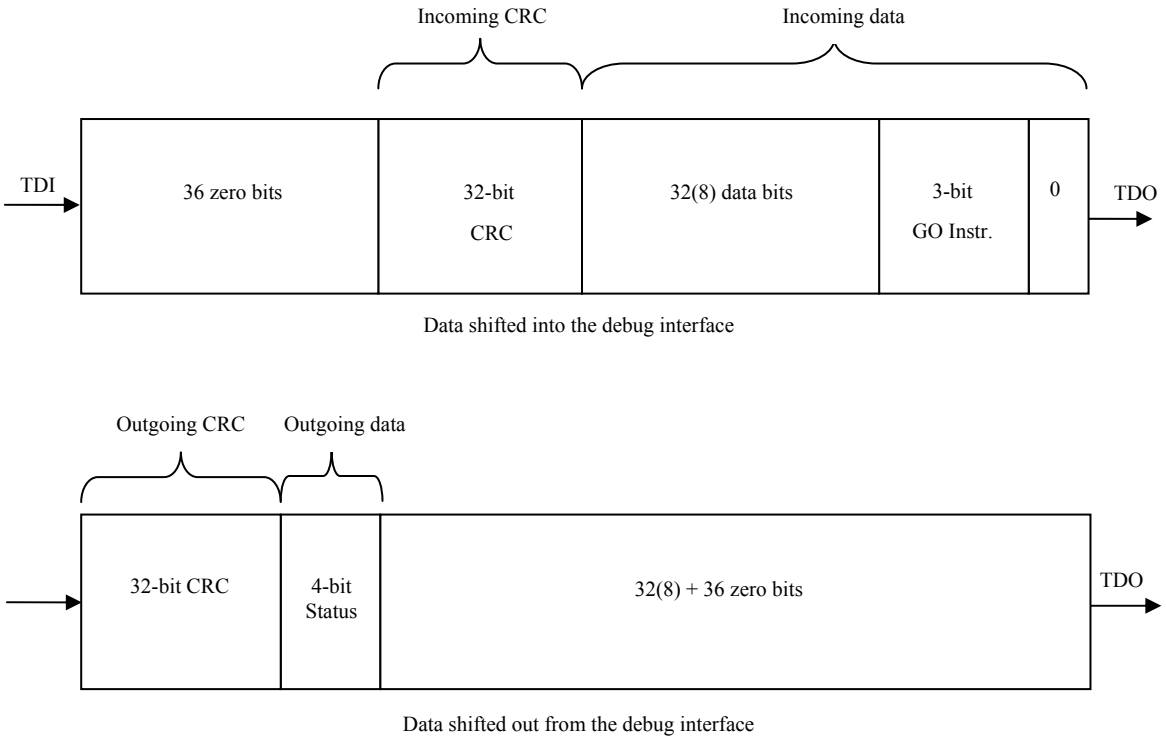

Figure 8: GO with CPU_WRITEx requested

4.3.7 Selecting different CPUs

cpu_sel_o and cpu_stall_all_o signals have been added to the Debug Interface to support more than one CPU. It is meant, that this signals are used to enable multiplexing of all the signals going from/to CPU and Debug Interface (cpu_bp_i, cpu_data_i, cpu_data_o, cpu_addr_o, cpu_stall_o, cpu_ack_i). The multiplexing of signals is not part of the Debug Interface and is the responsibility of the system integrator.

cpu_sel_o signals are controlled from CPU_SEL register and cpu_stall_all_o signal is controlled from CPU_OP register.

For more information about changing the value of the internal CPU register go to the section 4.3.2 CPU Register Write operation on page 36.

Reading the value of the internal CPU register is described in section 4.3.1 CPU Register Read operation on page 35.

Normally, if just one CPU is connected to the Debug Interface these signals are not necessary and can be ignored.

4.3.8 Stalling CPU(s)

The selected CPU can be stalled in two ways:

- By deliberately setting bit CPUSTALL in the CPU_OP register to 1 (see section 3.2 CPU Operation Register on page 12 for more details). Clearing this bit again restarts the CPU.
- An input breakpoint signal (cpu_bp_i) automatically stops the CPU and sets bit 0 of the CPU_OP register to 1. Clearing this bit again restarts the CPU.

When CPUSTALLALL bit is set in the CPU_OP register, all unselected CPUs are stalled.

For more information about changing the value of the internal CPU register go to the section 4.3.2 CPU Register Write operation on page 36.

Reading the value of the internal CPU register is described in section 4.3.1 CPU Register Read operation on page 35.

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

4.3.9 Resetting CPUs

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

For more information about changing the value of the internal CPU register go to the section 4.3.2 CPU Register Write operation on page 36.

Reading the value of the internal CPU register is described in section 4.3.1 CPU Register Read operation on page 35.

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 scan chains (from sub-modules). Currently two sub-modules are available, CPU and WISHBONE.
- CRC sub-module that checks incoming data.
- CRC sub-module that calculates the CRC for the outgoing data.
- WISHBONE sub-module
- CPU sub-module

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.

If there are more than 1 CPU in the system, then additional external logic is needed (marked as MUX logic in the Figure 9: Complete system on page 48). The function of this logic is:

- Multiplexes data that comes from CPUs to data that goes to the debug interface.
- Defines stall signals that are connected to the CPUs from cpu_stall_o, cpu_stall_all_o and cpu_sel_o signals.

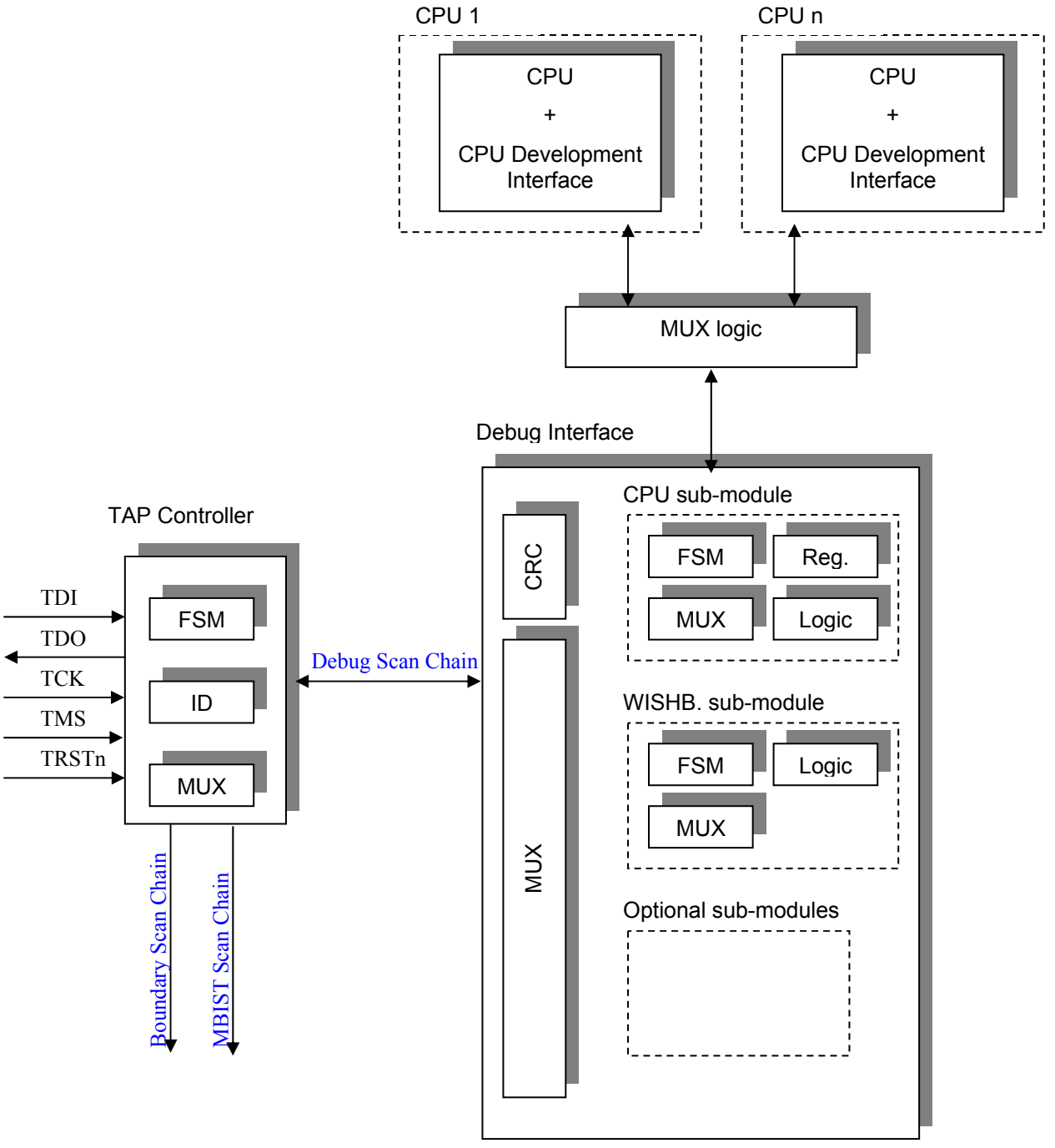

Figure 9: Complete system

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:

- Chain select instruction
- Sub-module instruction

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

Chain 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 49 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 16 for example).

5.3 WISHBONE sub-module

Is capable of doing the 8-bit, 16-bit and 32-bit WISHBONE 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 16 for more information about the WISHBONE sub-module.