SoC Debug Interface

Author: Igor Mohor
IgorM@opencores.org

Rev. 2.0

February 1, 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	Mutliple 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.

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 CPU REGISTERS LIST 11
3.2 CPU OPERATION REGISTER 12
3.3 CPU SELECT REGISTER 12

4 .... **14**

OPERATION **14**

4.1 CHAIN SELECTION 14
4.2 WISHBONE SUB-MODULE 15
4.2.1 WISHBONE Read operation **16**
4.2.2 WISHBONE Write operation **17**
4.2.3 WISHBONE Status operation **19**
4.2.4 Data and select signals **20**
4.3 CPU SUB-MODULE 2
4.3.1 CPU Read and CPU Register Read operation **3**
4.3.2 CPU or CPU Register Write operation **4**
4.3.3 Stalling CPU(s) **6**
4.3.4 Resetting CPUs **6**
4.3.5 Selecting different CPUs **6**

5 .... **8**

ARCHITECTURE **8**

5.1 DEBUG INTERFACE 10
5.2 CRC SUB-MODULE 10
5.3 WISHBONE SUB-MODULE 10

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.

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)

Bit #	Access	Description
		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.

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
- 3-bit chain ID
- 32-bit CRC

After this the following is shifted out:

- 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 that is related with the outgoing data

Data that is shifted out before the status is not important (all zeros).

All the data is shifted in/out with the MSB bit shifted first.

See sections 5.1 Debug Interface on page 10 and 5.2 CRC sub-module on page 10 for more details about the CRC.

4.2 WISHBONE Sub-module

There are three types of instruction in the WISHBONE sub-module:

- "Set address/length/type" instruction
- "Go" instruction
- "Read status" instruction

Before some data can be read from or write 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)

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

4.2.1 WISHBONE Read operation

After the debug is enabled and WISHBONE selected (see description on page 15), two instructions need to be executed.

First instruction that sets the address, type of operation and length is performed by shifting the following data through the data scan chain:

- 1-bit with value 0
- 3-bit instruction (READ8, READ16, READ32, depending on the cycle type (32-bit, 16-bit or 8-bit))
- 32-bit address
- 16-bit length (describes data length in bytes)
- 32-bit CRC

After this the following is shifted out:

- 4-bit status
    - 1 if incoming CRC is OK, else 0
    - 3 bit-s of 0
- 32-bit CRC that is related with the outgoing data

Data that is shifted out before the status is not important (all zeros).

All the data is shifted in/out with the MSB bit shifted first.

Second instruction is a "GO" instruction and performs the read operation on the WISHBONE bus. Address, cycle type and data length are specified with the first instruction. Data is latched to the internal buffer. The "GO" instruction would look like this:

- 1-bit with value 0
- 3-bit instruction GO
- 32-bit CRC

After this the following is shifted out:

- 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 that is related with the outgoing data

Data that is shifted out before the status is not important (all zeros).

All the data is shifted in/out with the MSB bit shifted first.

See sections 5.1 Debug Interface on page 10 and 5.2 CRC sub-module on page 10 for more details about the CRC.

4.2.2 WISHBONE Write operation

After the debug is enabled and WISHBONE selected (see description on page 15), two instructions need to be executed.

First instruction that sets the address, type of operation and length is performed by shifting the following data through the data scan chain:

- 1-bit with value 0
- 3-bit instruction (WRITE8, WRITE16, WRITE32, depending on the cycle type (32-bit, 16-bit or 8-bit))
- 32-bit address
- 16-bit length (describes data length in bytes)

- 32-bit CRC

After this the following is shifted out:

- 4-bit status
    - 1 if incoming CRC is OK, else 0
    - 3 bit-s of 0
- 32-bit CRC that is related with the outgoing data

Data that is shifted out before the status is not important (all zeros).

All the data is shifted in/out with the MSB bit shifted first.

Second instruction is a “GO” instruction and performs the write operation on the WISHBONE bus. Address, cycle type and data length are specified with the first instruction. Data that needs to be written to the WISHBONE bus is shifted in with the “GO” instruction. In this case the “GO” instruction looks like this:

- 1-bit with value 0
- 3-bit instruction GO
- data length x 8 bits of data
- 32-bit CRC

After this the following is shifted out:

- 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 over run occurred (data couldn’t be write fast enough), else 0
    - 1 if WISHBONE error occurred
- 32-bit CRC that is related with the outgoing data

Data that is shifted out before the status is not important (all zeros).

All the data is shifted in/out with the MSB bit shifted first.

See sections 5.1 Debug Interface on page 10 and 5.2 CRC sub-module on page 10 for more details about the CRC.

4.2.3 WISHBONE Status operation

After the debug is enabled and WISHBONE selected (see description on page 15), two instructions need to be executed.

Status operation consists of only one instruction that looks like this::

- 1-bit with value 0
- 3-bit instruction STATUS
- 32-bit CRC

After this the following is shifted out:

- 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 over run (under run) occurred (data couldn't be write (read) fast enough), else 0
    - 1 if WISHBONE error occurred
- 32-bit CRC that is related with the outgoing data

Data that is shifted out before the status is not important (all zeros).

All the data is shifted in/out with the MSB bit shifted first.

Errors are always latched and hold until the status operation is performed (i.e. If WISHBONE error occurs, WISHBONE error bit remains set until the status operation is performed). After the status operation, status bits are automatically cleared to zero.

See sections 5.1 Debug Interface on page 10 and 5.2 CRC sub-module on page 10 for more details about the CRC.

4.2.4 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

4.3 CPU Sub-module

CPU sub-module consists of internal registers and the CPU interface.

It can make accesses to both internal CPU registers and the CPU interface.

Internal registers are used for:

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

Before the CPU can be debugged, it must be selected. Selection is made through the write operation to the CPU register (See section 4.3.2 CPU or CPU Register Write operation on page 4 for more details).

CPU interface is an interface to the CPU debug facilities (that are part of the CPU).

Before some data can be read from or write to the CPU (registers or interface), 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)
- CPU sub-module needs to be selected (refer to section 4.1 Chain Selection on page 14 for more details)

There are two types of instruction in the CPU sub-module:

- "Set address/length/type" instruction
- "Go" instruction

CPU operation (read, write, register read or register write) consist of two consequent instructions. First instruction sets the address, type of instruction and length of data that is read or written. Second instruction is a "GO" instruction that actually does what the first instruction requests. Following section describes how read or write operations are performed.

4.3.1 CPU Read and CPU Register Read operation

After the debug is enabled and CPU sub-module selected (see description on page 15), two instructions need to be executed.

First instruction that sets the address and type of operation is performed by shifting the following data through the data scan chain:

- 1-bit with value 0
- 3-bit instruction (CPU_READ8, CPU_READ32 or CPU_READ_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

After this the following is shifted out:

- 4-bit status
    - 1 if incoming CRC is OK, else 0
    - 3 bit-s 010b
- 32-bit CRC that is related with the outgoing data

Data that is shifted out before the status is not important (all zeros).

All the data is shifted in/out with the MSB bit shifted first.

Second instruction is a "GO" instruction and performs the read operation to the CPU interface or CPU registers. Address and cycle type are specified with the first instruction. Data is latched to the internal buffer. The "GO" instruction would look like this:

- 1-bit with value 0
- 3-bit instruction GO
- 32-bit CRC

After this the following is shifted out:

- 8 or 32 bits of data (CPU_READ32 returns 32-bits, CPU_READ8 and CPU_READ_REGreturn 8-bit data)
- 4-bit status
    - 1 if incoming CRC is OK, else 0
    - 3 bit-s 010b
- 32-bit CRC that is related with the outgoing data

Data that is shifted out before the status is not important (all zeros).

All the data is shifted in/out with the MSB bit shifted first.

See section 3.1 CPU Registers List on page 11 for description of the CPU registers.

See sections 5.1 Debug Interface on page 10 and 5.2 CRC sub-module on page 10 for more details about the CRC.

4.3.2 CPU or CPU Register Write operation

After the debug is enabled and CPU selected (see description on page 15), two instructions need to be executed.

First instruction that sets the address and the type of operation is performed by shifting the following data through the data scan chain:

- 1-bit with value 0
- 3-bit instruction (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

After this the following is shifted out:

- 4-bit status
    - 1 if incoming CRC is OK, else 0
    - 3 bit-s of 010b
- 32-bit CRC that is related with the outgoing data

Data that is shifted out before the status is not important (all zeros).

All the data is shifted in/out with the MSB bit shifted first.

Second instruction is a "GO" instruction and performs the write operation to the CPU interface or CPU registers. Address and cycle type are specified with the first instruction. Data that needs to be written to the is shifted in with the "GO" instruction. In this case the "GO" instruction looks like this:

- 1-bit with value 0
- 3-bit instruction GO
- 8 or 32 bits of data
- 8 or 32 bits of data (CPU_WRITE32 needs 32-bits, CPU_WRITE8 and CPU_WRITE_REG need 8-bit data)
- 32-bit CRC

After this the following is shifted out:

- 4-bit status
    - 1 if incoming CRC is OK, else 0
    - 3 bit-s of 010b
- 32-bit CRC that is related with the outgoing data

Data that is shifted out before the status is not important (all zeros).

All the data is shifted in/out with the MSB bit shifted first.

See sections 5.1 Debug Interface on page 10 and 5.2 CRC sub-module on page 10 for more details about the CRC.

4.3.3 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 the breakpoint generation refer to the CPU manual (i.e. OpenRISC 1000 System Architecture Manual).

4.3.4 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.

4.3.5 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 muxing 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 muxing 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.

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

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. 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 Figure 1: Complete system on page 9). 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 signals.

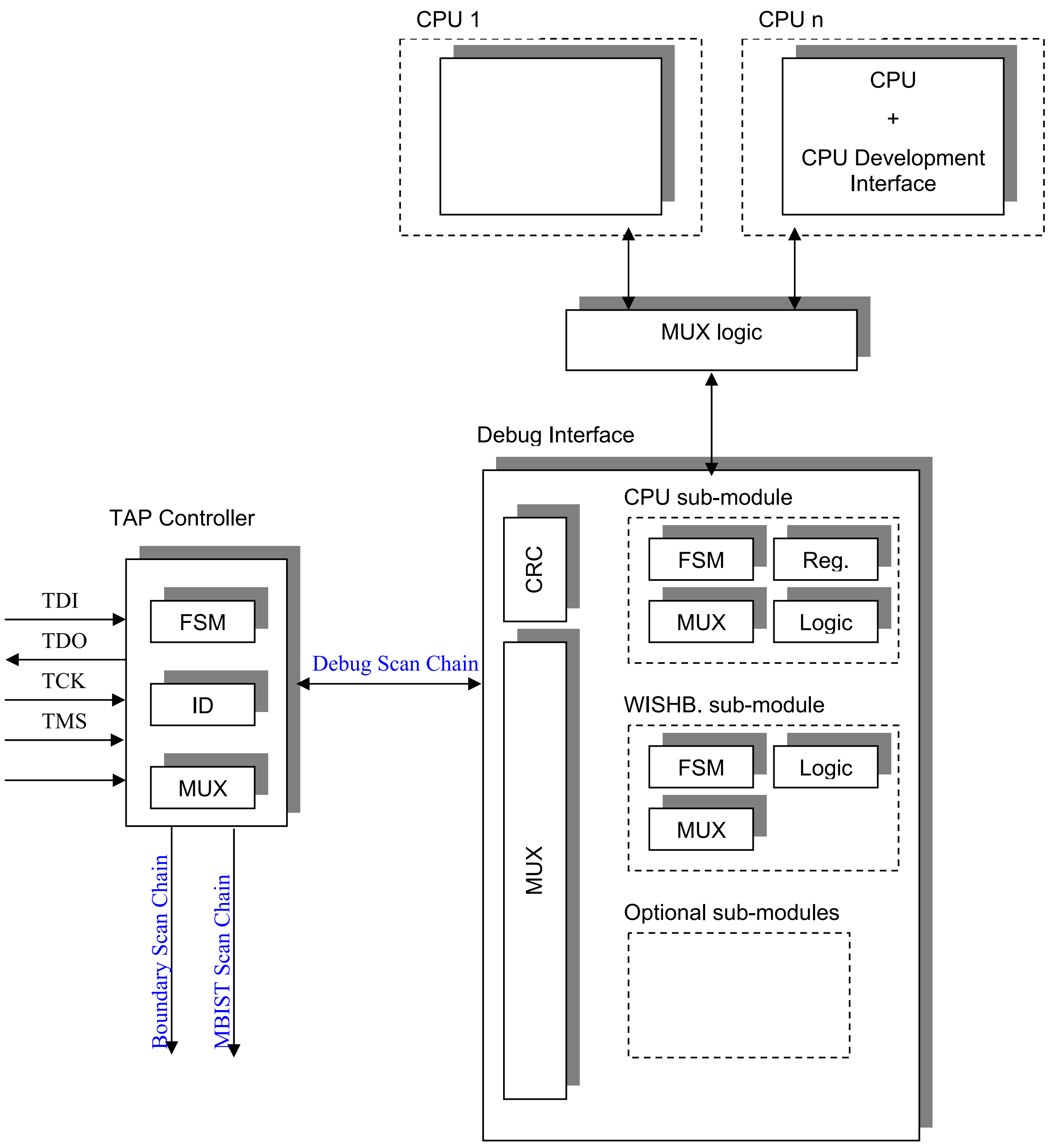

Figure 1: Complete system

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

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 10 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 its 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. The MSB bit of data is shifted in/out first. The CRC is also received/send with MSB first.

There is more information about the CRC available in the 5.1 Debug Interface section on page 10).

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. The length of the WISHBONE scan chain depends on the instruction (and their combination). See section 4.2 WISHBONE Sub-module on page 15 for more information about the chain length.