Advanced Debug Interface

Author: Nathan Yawn
nathan.yawn@opencores.org

Rev. 1.0

May 13, 2009

History

Rev.	Date	Author	Description
1.0	27/06/08	Nathan Yawn	First Draft

Table of Contents

1

Introduction

The Advanced Debug Interface (ADI) is a hardware module which creates an interface between a JTAG Test Access Port (TAP) and the system bus and CPU debug interface(s) of a System on Chip (SoC). It is part of the system which allows software running on an SoC system to be controlled and debugged by a software debugger such as GDB, running on a separate host PC. This debugging system allows the SoC to be debugged via direct hardware connection, and does not require the "GDB stub" software running on the SoC. A block diagram of this system is shown in Figure 1.

The external interface to the Advanced Debug Interface is based on IEEE Std. 1149.1, Standard Test Access Port and Boundary Scan Architecture. A JTAG TAP is required to link the ADI to an external JTAG cable. This TAP may be a stand-alone TAP[1], or it may be a specialized unit such as an Altera Virtual JTAG or a Xilinx Internal BSCAN unit. No TAP is included with the ADI. The ADI appears as a data register within the TAP.

Internally, the ADI has connections to both a WishBone bus and to one or more CPU debug interfaces. The WishBone interface does not use any of the burst features of the bus; it should therefore be compatible with any version of the bus (B.1 or B.3 as of this writing). The CPU interface is designed to connect to an OR1200 processor, or any other CPU which uses the same debug interface. Note that there are two versions of the OR1200 debug interface; the ADI is designed to use the newer version, which includes the strobe and acknowledge (dbg_stb and dbg_ack) signals.

1 Note that the ADI is incompatible with the stand-alone TAP distributed by OpenCores. Please use the version modified by the author instead.

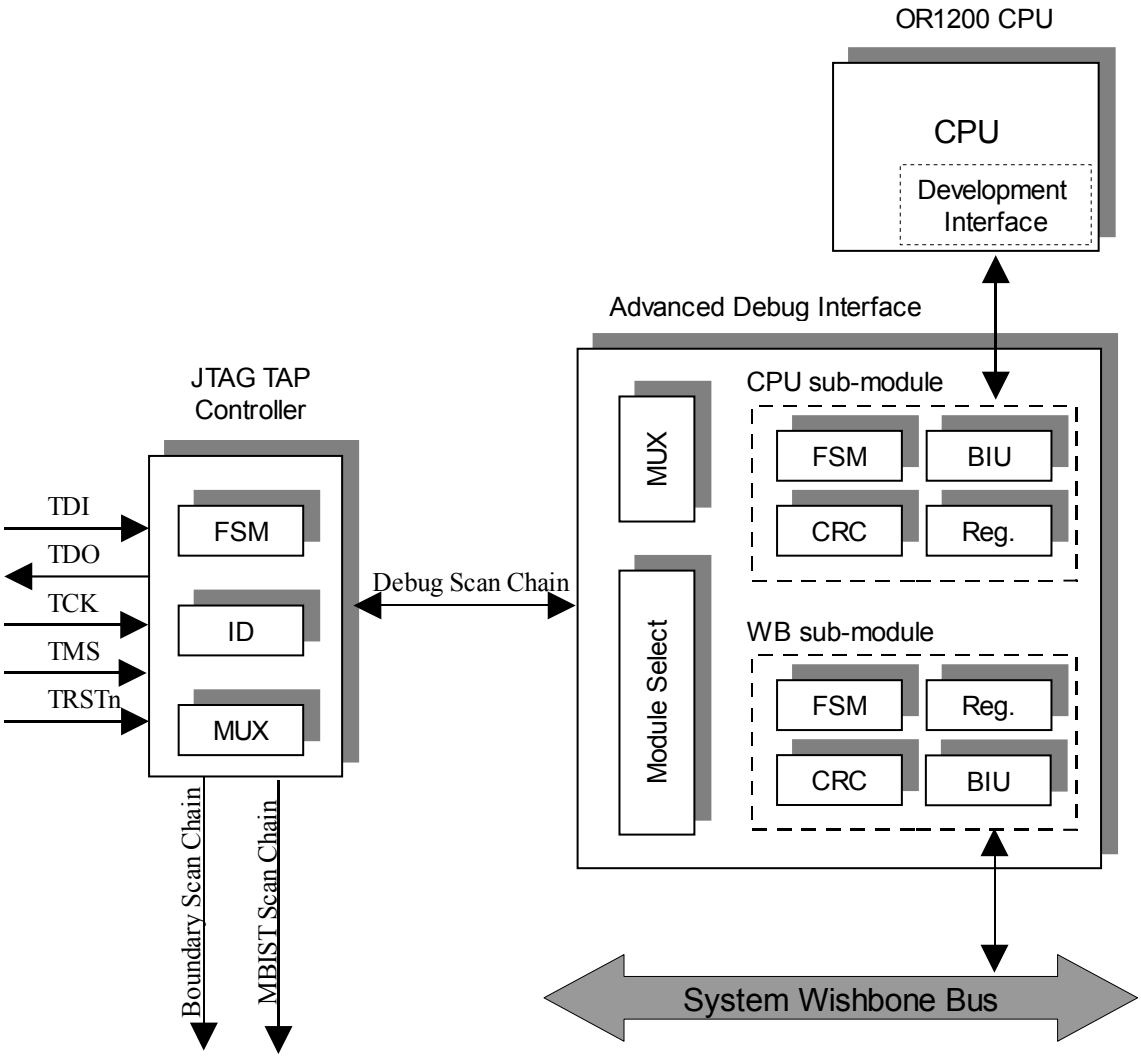


Figure 1: Block diagram of the complete debug system hardware

2

Architecture

The Advanced Debug Interface is built with a modular architecture, for flexibility and expandability. It consists of a top-level module, and several sub-modules designed to interface with individual SoC subsystems. The sub-modules currently include the WishBone module and the OR1200 module.

The top-level module contains the sub-modules, and a register to set the active sub-module. In order to send a command to a sub-module, it must first be made active by setting this top level register; only one sub-module may be active at a time. Zero or more instances of any type of sub-module are valid (the default is one WishBone sub-module and one OR1200 CPU sub-module). The top-level module also contains the input shift register, which holds incoming serial data from the TAP. The value in the input shift register is available to all sub-modules. Note that as per the JTAG specification, all serial transfers are LSB-first.

Sub-modules generally consist of two parts: internal registers, and a bus interface. Internal module registers may contain information about the status of the module (such as the error register in the WishBone module), or they may control external I/O lines (such as the reset and stall lines from the OR1200 module). Each sub-module contains an index register, which enables one internal register at a time for reading or writing. Internal registers are selected, read, and written by sending commands to a sub-module through the TAP.

The bus interface of a sub-module is designed to allow the TAP to read or write data from or to a bus as quickly as possible. Note that 'bus' in this case does not necessarily mean a WishBone bus; the bus interface of the OR1200 module connects to the processor's SPR bus. All bus transactions are 'burst' transactions from the external / TAP side: a setup command is first sent to a sub-module, then the entire block of data is streamed into or out of the sub-module without further control action. Different sub-modules may provide burst transactions using various word lengths. Burst data is CRC-protected. A block diagram of the general module structure is shown in Figure 2.

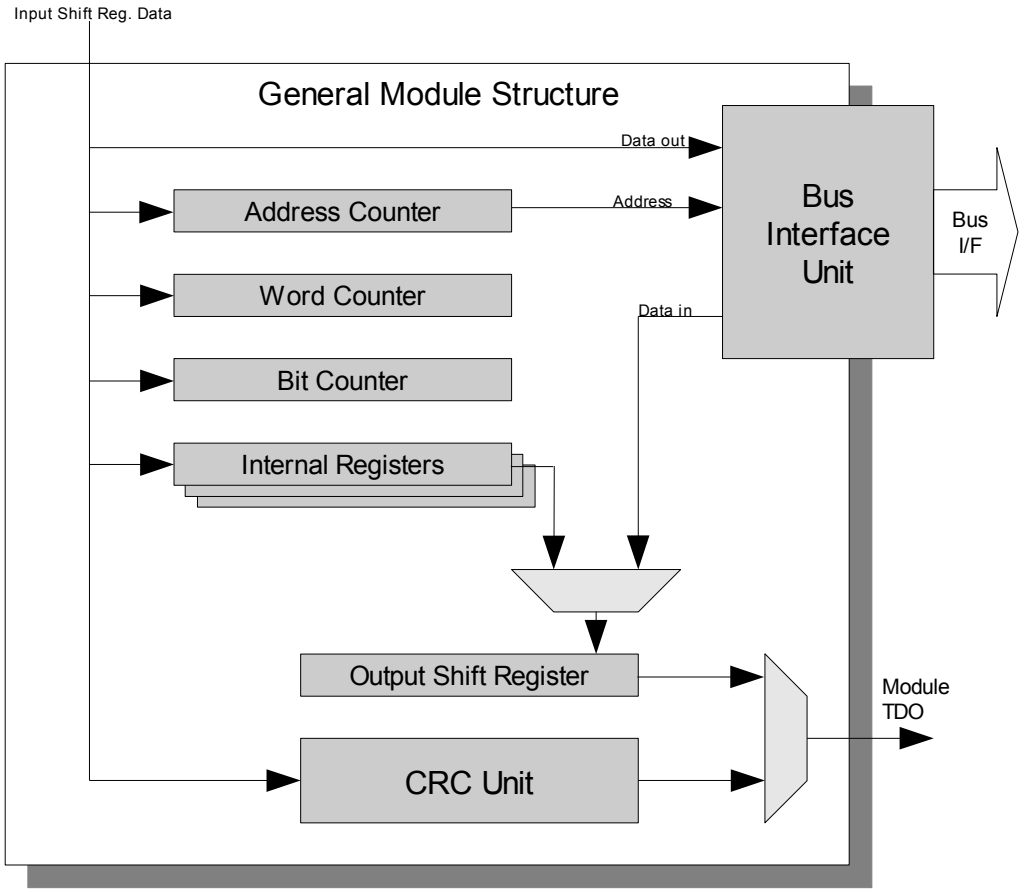


Figure 2: General module structure

In order to support JTAG scan chains with more than one device, commands are usually executed by the sub-modules when the TAP moves through the UPDATE_DR state. This allows a software driver to add the necessary bits to the end of a serial bitstream to position the command at the correct place in the scan chain.

The exception to this is burst data, due to its unknown (and potentially very large) size. To do a burst transaction, a burst command is first sent to a module, and executed by moving the TAP through the UPDATE_DR state. The next time the TAP goes into the SHIFT_DR state, 'burst mode' is active. In burst mode, bus data is immediately clocked into or out of the module, and the next bus transaction (or the end of the transaction) is determined by internal counters. In order to support multi-device chains, a "start bit" feature was added to burst mode. During burst writes, the module will not start its counters or collect write data until after the first '1' (a "start bit") is encountered in the bitstream. Since TAP devices in BYPASS mode will initially shift out a '0', this means that these extra bits from other devices will be ignored by the ADI module. Once the

module sees the start bit, it begins to capture write data (the start bit is discarded). Burst reads require no such added feature, as a software cable driver may simply discard the appropriate number of bits before beginning to capture read data. However, the first status bit of a burst read may be used as a start bit during burst reads, see the API sections on burst reads for details.

There are two more things to consider when using the ADI in a scan chain with multiple devices. First, all other devices should be in BYPASS mode when using the ADI – otherwise, a false start bit could get sent to the ADI during a burst write, corrupting the data. Second, the ADI data register is not a through shift register – that is, the serial TDI input of the ADI is not directly connected to the TDO output. This means that data shifted into the ADI will never appear at the output. As such, the ADI should never be active when other devices on the chain are in use.

Three modules are currently implemented. The following sections give details for each type.

2.1 Top Module

The top-level module is the simplest of the modules. It does not have a bus interface, and has only a single register. This register is called the "module select register", and is used to select the active sub-module. The top module does not use command opcodes the way the sub-modules do. Instead, a single bit in the input shift register (the MSB) indicates whether the command is a write to the select register, or a command to a sub-module (in which case the command is ignored by the top-level). The value in the select register cannot be read back.

The top-level module provides enable signals to all sub-modules, based on the value in the module select register. The value of the input shift register is also provided to all modules. Finally, the serial TDO output of the ADI is selected from the sub-module TDO outputs, based on the module select register. A block diagram of the top-level module is shown in Figure 3.

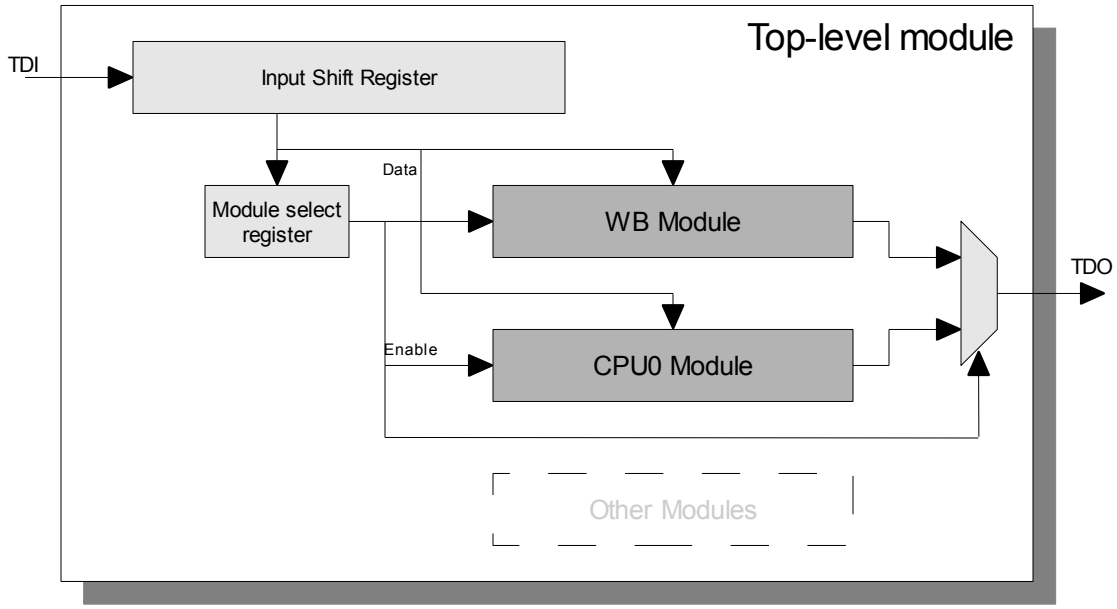


Figure 3: Block diagram of top-level module

2.2 WishBone Module

The purpose of the WishBone module is to provide a software debugger access to the SoC's memory system, allowing it to load code, examine and change program data, and set software breakpoints. The WishBone module has a bus interface which follows the WishBone standard. Because the JTAG interface is relatively slow, burst accesses on the WishBone bus interface are not used, therefore the interface should be compatible with all versions of the WishBone standard (through B.3 as of this writing). The WishBone module allows 8-, 16-, and 32-bit reads and writes over the WishBone bus interface. The WishBone Module uses a 32-bit address and a 32-bit data bus, and allows burst transfers of up to 65535 words (262140 bytes).

Since the JTAG clock is asynchronous to the WishBone bus clock, transactions are synchronized between clock domains. This clock differential may cause other problems, however, if the JTAG clock is much faster than the WishBone clock. For efficient operation, it must be possible to complete a WishBone write (plus 4 JTAG-domain clock cycles for control and synchronization) in less time than it takes to shift the next word in via the TAP. The problem is magnified when using 8-bit words. If the bus is not ready by the time the next word has arrived, then the attempt to write the next word will fail. This overflow condition can be detected by reading back a status bit after writing each word in burst mode. If the status bit is true, then the transaction has succeeded and the burst should continue. If the status bit is false, then an overflow has occurred, and the software driver should retry part of the burst, starting with last word written.

Underflow conditions during burst reads are avoided by use of a “ready” bit, which is read by the software driver before each word is shifted out of the ADI. When data is not yet available, the WishBone module shifts out zeros. As soon as a data word is read from the WishBone bus, the module shifts out a one, indicating that data is ready (the driver should discard this '1' and all preceding ready bits). Bus data follows immediately after a true ready bit is sent.

The WishBone module contains a single 33-bit internal register, called the “error register,” which is used to detect errors on the bus interface. During a burst read or write, the WishBone error (wb_err) bit is sampled at the end of each bus transaction. If the error bit is ever true, then the error bit (bit 0) in the error register is set, and the address of the failed transaction is captured into the other 32 bits of the error register. Once the error bit has been set, it can only be cleared by performing an internal register write to the WishBone module, and the addresses of subsequent errors will not be captured. Thus, when the error bit is set, the error register contains the address of the first bus error encountered since the bit was last reset.

The error register should be reset before each bus transaction (or after each time the error bit is found set), then checked after each burst transaction. Because the error bit is the least-significant bit, a software driver may read only 1 bit in order to determine whether or not an error condition exists. If true, then the driver may read out the 32-bit error address, and retry the part of the burst starting from that address.

The WishBone sub-module includes a CRC calculation, which is used to protect burst data. During burst transactions, an internal word counter determines when all of the data for a burst has been transferred. If a burst read has just completed, the module then begins to shift out a 32-bit CRC, which the host may compare to a locally-generated CRC. If the completed transaction is a burst write, the module accepts a 32-bit CRC from the host, then shifts out a single bit indicating whether or not the CRC received matched its internal CRC computation. Note that the CRC is done only on the burst data; the preceding burst command, and all start and ready bits, are ignored.

The CRC polynomial is 0xEDB88320. This is bitwise-reversed from many implementations; this is because we compute the CRC LSB-first (also reversed from other implementations). The LSB-first calculation was used to reduce hardware and routing requirements in the CRC module.

2.3 OR1200 CPU Module

The OR1200 CPU sub-module is designed to allow a software debugger to access the internal registers of a CPU, to stall and reset the processor, and to take control when a breakpoint occurs in software. The OR1200 module may also allow access to the CPU's

hardware breakpoint and watchpoint configuration and trace buffer, if the CPU has been synthesized with these features – see the OR1000 architecture specification and the OR1200 implementation document for details.

The OR1200 sub-module is based on the WishBone module, and is therefore similar. The bus interface connects to the debug interface of the OR1200, which allows access to the processor's SPR bus. Accesses to this bus are performed by ADI burst transactions. Reads, writes, clock synchronization, ready bits, status bits, overflow, underflow, and CRC computations are all handled exactly as they are in the WishBone module. The OR1200 debug interface bus does not have an error indicator bit. Thus, the OR1200 module does not have an internal "bus error" register.

The OR1200 module allows the user to set and clear the CPU reset bit, to stall the CPU, and to capture breakpoints in the CPU. This is done through a module internal register, called the "CPU status register." Bit 1 of this register is the reset bit. When this bit is true, the cpu_rst_o output bit is set true, and vice-versa. This bit is set and cleared only by internal register writes via the ADI. Note that this bit is synchronized between clock domains.

Bit 0 of the CPU status register is the stall bit. When set, the cpu_stall_o output of the ADI is also set, and vice versa. This bit may be set and cleared via internal register access to the ADI. This bit may also be set by the CPU: when the cpu_bp_i input from the CPU goes high (indicating a breakpoint), the stall bit in the register is set, and the stall output set high. This effectively transfers control of the CPU to the ADI, which may perform various debugging operation before clearing the stall bit via internal register access, allowing the CPU to continue normal execution.

3

API

This section gives information on the commands, opcodes, and data formats of the Advanced Debug Interface. There are three major sections: the top-level ADI, the WishBone module, and the CPU module. All commands and registers are shown with the least-significant bit to the right, and all commands are shifted out LSB-first. Data is also shifted in to and out of the ADI LSB-first. All commands are interpreted when the TAP passes through state UPDATE_DR. All commands are interpreted as MSB-aligned. This means that the minimum number of bits for a command may be shifted into the ADI, without consideration for the length of the data register.

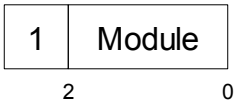
Note that the ADI must be reset before any commands will be accepted or executed. If the system TAP has an output indicating the "Test Logic / Reset" state, it is recommended that this be used as the ADI reset input. If this signal is not available, it is recommended that the ADI reset input be connected to the OR1200 or WishBone bus reset signal.

3.1 Top-level Commands

3.1.1 Module Select command

This command is used to select which one of the sub-modules is active. Only the active sub-module will process commands sent to the ADI. This command should be sent before any other command is sent to any sub-module.

Figure 4: Module Select Command format



Bit #	Access	Description
2	W	**Top-Level Select** Set to '1' to select the top level module
1:0	W	**Module** Number of the module to select. The following modules are valid: 0x0 = WishBone module 0x1 = CPU0 module 0x2 = CPU1 module (optional)

Table 1: Module Select command format

3.2 WishBone Commands

The WishBone sub-module uses a standardized command format. Each command must have a zero as the MSBit, to differentiate it from a module select command. In the next four most-significant bit positions is a 4-bit opcode, which indicates the operation to be performed. Following the opcode are zero or more data values, whose length and meaning are command-specific. Table 2 summarizes the opcodes supported in the WishBone module.

OPCODE	Operation
0x0	NOP
0x1	Burst Setup Write, 8-bit words
0x2	Burst Setup Write, 16-bit words
0x3	Burst Setup Write, 32-bit words
0x5	Burst Setup Read, 8-bit words
0x6	Burst Setup Read, 16-bit words
0x7	Burst Setup Read, 32-bit words
0x9	Internal register write
0xD	Internal register select

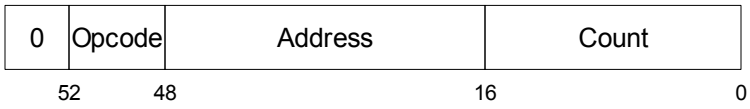
Table 2: WishBone module command opcode summary

3.2.1 Burst Setup

A burst setup command prepares the WishBone module to do either a read or write burst, in order to move data to or from a consecutive sequence of addresses on the WishBone bus. The word size of the burst is decoded from the opcode. The WishBone module can use an 8-, 16-, or 32-bit word size. After each individual word transfer during a burst, the address counter in the ADI is incremented according to the word size: it will be incremented by 1, 2, or 4, for 8-, 16-, and 32-bit words respectively.

After a burst setup command has been executed (in the UPDATE_DR state), the WishBone module will enter 'burst read' or 'burst write' mode, the next time the TAP enters SHIFT_DR mode. Whether read or write mode is used depends on the opcode sent with the burst setup command. Details on burst read and burst write modes are in the following sections.

Figure 5: Burst Setup command



Bit #	Access	Description
52	W	**Top-Level Select** Set to '0' for all sub-module commands
48:51	W	**Opcode** Operation to perform. The following are valid burst setup operations: 0x1 = Burst Write, 8-bit words 0x2 = Burst Write, 16-bit words 0x3 = Burst Write, 32-bit words 0x5 = Burst Read, 8-bit words 0x6 = Burst Read, 16-bit words 0x7 = Burst Read, 32-bit words
47:16	W	**Address** The first WishBone address which will be read from or written to
0:15	W	**Count** Total number of *words* to be transferred. Note that this means that the total number of *bits* transferred depends both on this field, and on the opcode. Must be greater than 0.

Table 3: WishBone module Burst Setup command format

3.2.2 Burst Write

The next time the ADI is accessed after sending a valid burst write command, the WishBone module will be in burst write mode. In this mode, commands and data are not interpreted or executed on transition through UPDATE_DR. Instead, counters are used to determine position in the bitstream, and a word is written to the WishBone as soon as it has been transferred in via JTAG. The total length of a burst write transfer depends on the word size (set by the opcode) and the count fields in the burst setup command; for a word size of n and a transfer of m words, the total length will be $((n + 1) * m) + 34$.

The first bit transferred in a burst write is a '1' start bit. This tells the WishBone module to begin counting bits, and is required due to the possibility of multiple devices on the JTAG chain. After the start bit, one word of data is transferred into the ADI, followed by a status bit, which is transferred *out* of the ADI, and should be read by the software driver. The status bit tells the user whether the WishBone was ready to accept the word just transferred; when true, the bus was ready. When false, the word was not written to the WishBone, and the software driver should retry the transfer, starting from

the failed word. Note that timing of the actual reception of the status bit may vary, depending on the number of other devices on the JTAG chain.

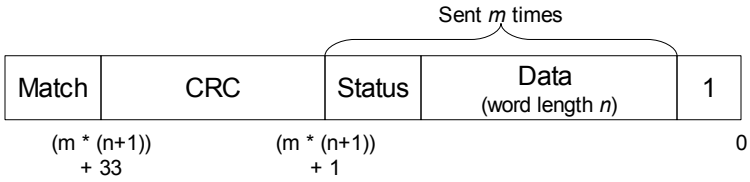
Data transmission continues to alternate data word and status bit until all words have been transferred. Immediately following the last status bit, a 32-bit CRC code is transferred into the WishBone module. This CRC is compared with a CRC computed internally to the WishBone module. After the CRC is transferred in, a single bit is transferred out of the ADI, indicating whether or not the CRC written matched the CRC calculated. A burst write transaction may be aborted at any time by moving the TAP through the UPDATE_DR state.

The CRC protects only the data bits in a burst transaction; commands and status bits are not included in the CRC computation. The CRC resets before each burst transaction. For more information on the CRC calculation, see chapter 6.

WishBone bus errors are captured during a burst, but the information is not transferred during the burst transaction. After a burst, the user should check the WishBone module error register to see if a bus error occurred during the burst, and if so, at what address. See the section on WishBone sub-module internal registers for details on the error register.

Note that extra bits sent at the end of a burst write are ignored; thus, the user need not worry about sending a valid or safe operation/opcode at the end of the burst transaction.

Figure 6: Burst Write format



Bits	Access	Description
1 bit	R	**Match** '1' if CRC sent matches internal CRC computation, '0' if not
32 bits	W	**CRC** 32-bit CRC computed on all of the data bits of the burst
1 bit	R	**Status** '1' if the most recently sent data word was written to the WishBone, '0' if the bus was not ready. Sent *m* times, once after each data word.
n bits	W	**Data** Data word. Length specified by the opcode in the burst setup command. Sent *m* times.
1 bit	W	**Start Bit** Set to '1' to indicate the start of a burst write.

Table 4: WishBone module burst write format

3.2.3 Burst Read

The next time the ADI is accessed after sending a valid burst read command, the WishBone module will be in burst read mode. In this mode, commands and data are not interpreted or executed on transition through UPDATE_DR. Instead, counters are used to determine position in the bitstream, and a word is read from the WishBone while the previous data word is transferred out via JTAG. The total length of a burst read transfer depends on the word size (set by the opcode) and the count fields in the burst setup command; for a word size of *n* and a transfer of *m* words, the total length will be $((n + 1) * m) + 32$.

The first bit (or bits) transferred during a burst read is a status bit. This bit indicates whether or not data from the WishBone is ready to be transferred out via JTAG. The WishBone module will send '0' bits until a word is ready, then send a single '1' bit. One data word will follow a '1' status bit. The status bit is used to prevent data underruns and retries; a retry should never be necessary due to a data underrun in the WishBone module.

A status bit (or bits) is transferred before each data word. Data transmission continues to alternate status bits and data words until all words have been transferred. Immediately following the last data word, a 32-bit CRC code is sent from the WishBone module to the driver software. The driver software should compare the CRC received

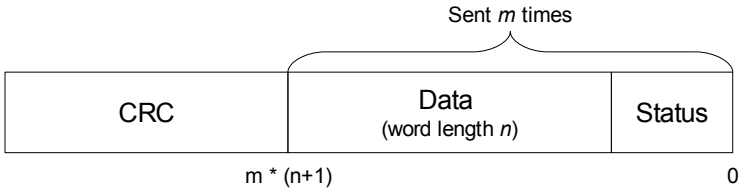
with one computed internally, to determine whether or not the complete transaction must be retried.

The CRC protects only the data bits in a burst transaction; commands and status bits are not included in the CRC computation. The CRC resets before each burst transaction. For more information on the CRC calculation, see chapter 6.

WishBone bus errors are captured during a burst, but the information is not transferred during the burst transaction. After a burst, the user should check the WishBone module error register to see if a bus error occurred during the burst, and if so, at what address. See the section on WishBone sub-module internal registers for details on the error register.

Note that extra bits sent at the end of a burst read are ignored; thus, the user need not worry about sending a valid or safe operation/opcode at the end of the burst transaction.

Figure 7: Burst Read format



Bits	Access	Description
32 bits	R	**CRC** 32-bit CRC computed on all of the data bits of the burst
n bits	R	**Data** Data word. Length specified by the opcode in the burst setup command. Sent *m* times.
1 bit	R	**Status** Read '0' until a word is ready to be sent, then a single '1' bit is sent before the data word.

Table 5: WishBone module burst read format

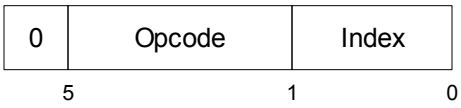
3.2.4 Register Select

A register select command will make the module-internal register with the given index active (in the currently selected sub-module). While any register in the current module can be written with a single command, only the active register can be read.

When the TAP enters CAPTURE_DR mode, the WishBone module captures the value of the active register into the output shift register, allowing the value to be read when the TAP is in SHIFT_DR mode. This will happen each time a command is sent to the WishBone module, unless the previous command was a burst setup.

The register select command uses the same top-level select bit and opcode format as the other WishBone module commands. In this case, the opcode is followed by a 1-bit value, which is the index of the register which should be made active. See the API section on registers for a complete listing of all registers in the WishBone module, and the meaning of each.

Figure 8: Register Select command format



Bit #	Access	Description
5	W	**Top-Level Select** Set to '0' for all sub-module commands
4:1	W	**Opcode** Operation to perform. 0xD = Internal Register Select
0	W	**Index** Index of the register to make active. The WishBone module uses a 1-bit index.

Table 6: WishBone module Register Select command format

3.2.5 Register Read

There is no specific command to read a module internal register; the value of the active register is shifted out every time a command is shifted in. In order to read a particular register, make that register active using the register select command, then read out the value of the register while sending a NOP command. The length of each register may vary, and the meaning of each bit is also register-specific. The register data will be LSB-aligned; that is, the LSB of the register data will be the first bit shifted out of the ADI. This allows the minimum number of JTAG bits to be transferred. It is legal to read more bits than the active register has – the additional bits will have undefined value, and should be discarded. Note that only register data will be transferred out, no command, index, opcodes, start, or status bits will be sent with it. It is legal to abort a register read at any time, provided that a valid command (probably a NOP) is in the correct position in the input shift register.

3.2.6 Register Write

A register write command contains both a register index, and data to be written to the register at that index. The register with the given index will become the active register after this command is executed. Note that the value of the *previously* active register will be shifted out as this command is shifted in. The length of the data field is variable, and depends on the particular register being written. See the API section on registers for a complete description of the registers, their lengths, and their meanings.

Figure 9: Register Write command format

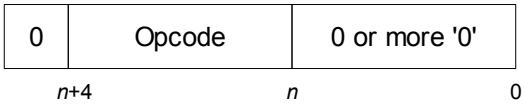
Bit #	Access	Description
5 + *n*	W	**Top-Level Select** Set to '0' for all sub-module commands
(4:1) + *n*	W	**Opcode** Operation to perform. 0x9 = Internal Register Write
n	W	**Index** Index of the register to make active. The WishBone module uses a 1-bit index.
n-1:0	W	**Data** *n* bits of data to write to the register specified by Index. *n* depends on the register being written.

Table 7: WishBone module Register Write command format

3.2.7 NOP

A NOP command will perform no operation. It is included as a "safe" command to shift into the WishBone module while shifting out internal register data. A NOP command consists of five or more zeros, making it easy to send for any length of data read.

Figure 10: NOP command format



Bit #	Access	Description
4 + *n*	W	**Top-Level Select** Set to '0' for all sub-module commands
(3:0) + *n*	W	**Opcode** Operation to perform. 0x0 = NOP
n:0	W	**Zero** Zero or more '0' bits

Table 8: WishBone module NOP command format

3.3 CPU Commands

The CPU sub-module uses the same standardized command format as the WishBone module. Each command must have a zero as the MSB, to differentiate it from a module select command. In the next four most-significant bit positions is a 4-bit opcode, which indicates the operation to be performed. Following the opcode are zero or more data values, whose length and meaning are command-specific. Table 9 summarizes the opcodes supported in the CPU module.

OPCODE	Operation
0x0	NOP
0x3	Burst Setup Write, 32-bit words
0x7	Burst Setup Read, 32-bit words
0x9	Internal register write
0xD	Internal register select

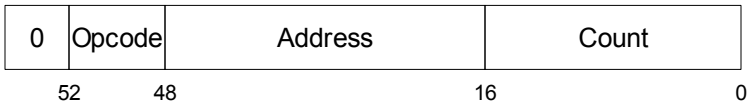
Table 9: CPU module command opcode summary

3.3.1 Burst Setup

A burst setup command prepares the CPU module to do either a read or write burst, in order to move data to or from a consecutive sequence of address on the OR1200's SPR bus. Because all SPRs are 32-bit registers, all burst transfers in the CPU module use 32-bit words. After each individual word transfer during a burst, the address counter in the CPU module is incremented by 1. Note that while the OR1000 architecture defines an SPR address as 16 bits, the OR1200 implementation uses a 32-bit address in its external debug interface. The ADI is designed to use the 32-bit OR1200 implementation.

After a burst setup command has been executed (in the UPDATE_DR state), the CPU module will enter 'burst read' or 'burst write' mode, the next time the TAP enters SHIFT_DR mode. Whether read or write mode is used depends on the opcode sent with the burst setup command. Details on burst read and burst write modes are in the following sections.

Figure 11: Burst Setup command



Bit #	Access	Description
52	W	**Top-Level Select** Set to '0' for all sub-module commands
48:51	W	**Opcode** Operation to perform. The following opcodes are valid burst setup operations for the CPU module: 0x3 = Burst Write, 32-bit words 0x7 = Burst Read, 32-bit words
47:16	W	**Address** The first OR1200 SPR address which will be read or written
0:15	W	**Count** Total number of 32-bit words to be transferred.

Table 10: CPU module Burst Setup command format

3.3.2 Burst Write

The next time the ADI is accessed after sending a valid burst write command, the CPU module will be in burst write mode. In this mode, commands and data are not interpreted or executed on transition through UPDATE_DR. Instead, counters are used to determine position in the bitstream, and a word is written to the CPU SPR bus as soon as it has been transferred in via JTAG. The total length of a burst write transfer depends on the count field in the burst setup command; for a transfer of *m* words, the total length will be (33 * *m*) + 34.

The first bit transferred in a burst write is a '1' start bit. This tells the CPU module to begin counting bits, and is required due to the possibility of multiple devices on the JTAG chain. After the start bit, one word of data is transferred into the ADI, followed by a status bit, which is transferred *out* of the ADI, and should be read by the software driver. The status bit tells the user whether the OR1200 was ready to accept the word just transferred; when true, the bus was ready. When false, the word was not written to the SPR bus, and the software driver should retry the transfer, starting from the failed word. Note that timing of the actual reception of the status bit may vary, depending on the number of other devices on the JTAG chain.

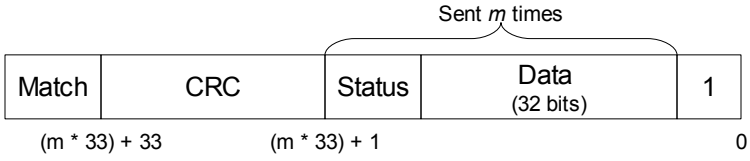
Data transmission continues to alternate data word and status bit until all words have been transferred. Immediately following the last status bit, a 32-bit CRC code is transferred into the CPU module. This CRC is compared with a CRC computed internally to the CPU module. After the CRC is transferred in, a single bit is transferred out of the ADI, indicating whether or not the CRC written matched the CRC calculated. A burst write transaction may be aborted at any time by moving the TAP through the UPDATE_DR state.

The CRC protects only the data bits in a burst transaction; commands and status bits are not included in the CRC computation. The CRC resets before each burst transaction. For more information on the CRC calculation, see chapter 6.

The OR1200 SPR bus does not provide any sort of error indication beyond ready / not ready. As such, there is no error register to be tested after a burst (as opposed to the WishBone module, which has such a register).

Note that extra bits sent at the end of a burst write are ignored; thus, the user need

Figure 12: Burst Write format



Bits	Access	Description
1 bit	R	**Match** '1' if CRC sent matches internal CRC computation, '0' if not
32 bits	W	**CRC** 32-bit CRC computed on all of the data bits of the burst
1 bit	R	**Status** '1' if the most recently sent data word was written to the SPR bus, '0' if the bus was not ready. Sent *m* times, once after each data word.
n bits	W	**Data** 32-bit data word. Sent *m* times.
1 bit	W	**Start Bit** Set to '1' to indicate the start of a burst write.

Table 11: CPU module burst write format

3.3.3 Burst Read

The next time the ADI is accessed after sending a valid burst read command, the CPU module will be in burst read mode. In this mode, commands and data are not interpreted or executed on transition through UPDATE_DR. Instead, counters are used to determine position in the bitstream, and a word is read from the CPU SPR bus while the previous data word is transferred out via JTAG. The total length of a burst read transfer depends on the count field in the burst setup command; for a transfer of *m* words, the total length will be (33 * *m*) + 32.

The first bit (or bits) transferred during a burst read is a status bit. This bit indicates whether or not data from the SPR bus is ready to be transferred out via JTAG. The CPU module will send '0' bits until a word is ready, then send a single '1' bit. One data word will follow a '1' status bit. The status bit is used to prevent data underruns and retries; a retry should never be necessary due to a data underrun in the CPU module.

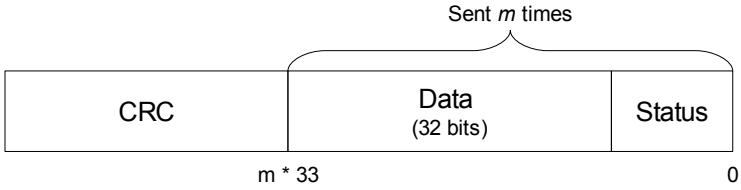
A status bit (or bits) is transferred before each data word. Data transmission continues to alternate status bits and data words until all words have been transferred. Immediately following the last status bit, a 32-bit CRC code is sent from the CPU module to the driver software. The driver software should compare the CRC received with one computed internally, to determine whether or not the complete transaction must be retried.

The CRC protects only the data bits in a burst transaction; commands and status bits are not included in the CRC computation. The CRC resets before each burst transaction. For more information on the CRC calculation, see chapter 6.

The OR1200 SPR bus does not provide any error indication for failed transactions beyond ready / not ready. Thus, there is no error register to check after a burst is completed (as opposed to the WishBone module).

Note that extra bits sent at the end of a burst read are ignored; thus, the user need not worry about sending a valid or safe operation/opcode at the end of the burst transaction.

Figure 13: Burst Read format



Bits	Access	Description
32 bits	R	**CRC** 32-bit CRC computed on all of the data bits of the burst
n bits	R	**Data** Data word. Length specified by the opcode in the burst setup command. Sent *m* times.
1 bit	R	**Status** Read '0' until a word is ready to be sent, then a single '1' bit is sent before the data word.

Table 12: CPU module burst read format

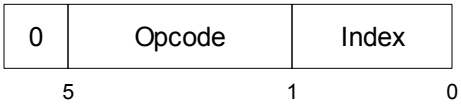
3.3.4 Register Select

A register select command will make the module-internal register with the given index active (in the currently selected sub-module). While any register in the current module can be written with a single command, only the active register can be read.

When the TAP enters CAPTURE_DR mode, the CPU module captures the value of the active register into the output shift register, allowing the value to be read when the TAP is in SHIFT_DR mode. This will happen each time a command is sent to the CPU module, unless the previous command was a burst setup.

The register select command uses the same top-level select bit and opcode format as the other module commands. In this case, the opcode is followed by a 1-bit value, which is the index of the register which should be made active. See the API section on register for a complete listing of all registers in the CPU module, and the meaning of each.

Figure 14: Register Select command format



Bit #	Access	Description
5	W	**Top-Level Select** Set to '0' for all sub-module commands
4:1	W	**Opcode** Operation to perform. 0xD = Internal Register Select
0	W	**Index** Index of the register to make active. The CPU module uses a 1-bit index.

Table 13: CPU module Register Select command format

3.3.5 Register Read

There is no specific command to read a module internal register; the value of the active register is shifted out every time a command is shifted in. In order to read a particular register, make that register active using the register select command, then read out the value of the register while sending a NOP command. The length of each register may vary, and the meaning of each bit is also register-specific. The register data will be LSB-aligned; that is, the LSB of the register data will be the first bit shifted out of the

ADI. This allows the minimum number of bits to be transferred over the JTAG, and allows the user to ignore the total length of the output shift register. It is legal to read more bits than the active register has – the additional bits will have undefined values, and should be discarded. Note that only register data will be transferred out; no command, index, opcodes, start, or status bits will be sent with it. It is legal to abort a register read at any time, provided that a valid command (probably a NOP) is in the correct position in the input shift register.

3.3.6 Register Write

A register write command contains both a register index and data to be written to the register at that index. The register with the given index will become the active register after this command is executed. Note that the value of the *previously* active register will be shifted out as this command is shifted in. The length of the data field is variable, and depends on the particular register being written. See the API section on registers for a complete description of the registers, their lengths, and their meanings.

Figure 15: Register Write command format

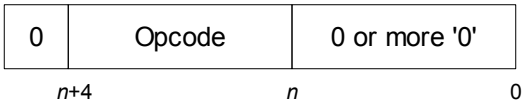
Bit #	Access	Description
5 + *n*	W	**Top-Level Select** Set to '0' for all sub-module commands
(4:1) + *n*	W	**Opcode** Operation to perform. 0x9 = Internal Register Write
n	W	**Index** Index of the register to make active. The CPU module uses a 1-bit index.
n-1:0	W	**Data** *n* bits of data to write to the register specified by Index. *n* depends on the register being written.

Table 14: CPU module Register Write command format

3.3.7 NOP

A NOP command will perform no operation. It is included as a "safe" command to shift into the CPU module while shifting out internal register data. A NOP command consists of five or more zeros, making it easy to send for any length of data read.

Figure 16: NOP command format



Bit #	Access	Description
4 + *n*	W	**Top-Level Select** Set to '0' for all sub-module commands
(3:0) + *n*	W	**Opcode** Operation to perform. 0x0 = NOP
n:0	W	**Zero** Zero or more '0' bits

Table 15: CPU module NOP command format

3.4 WishBone Module Registers

Table 16 summarizes all of the registers contained within the WishBone sub-module. Note that the data format of a register may be different depending on whether it is read or written; this saves the user from having to shift in extra bits to fill read-only values when writing.

Index	Register name
0x0	Error register

Table 16: WishBone module register summary

3.4.1 Error Register

The error register captures WishBone bus errors during burst transactions. Each time a bus access is completed, the WishBone error indicator bit (wb_err) is tested. If an error is present, then the error bit in the error register is set to '1', and the address of the failed access is stored in the rest of the error register. Once the error bit is set, the error register will retain its value and further WishBone errors will be ignored until the error bit is reset. The error bit may only be reset by writing a '1' to the error bit via an internal register write.

When read, the error bit is the first bit shifted out. This allows transferring the minimum number of bits when testing whether an error has occurred (5 bits must be transferred in order to send a valid NOP command while reading). If an error has

occurred, then an error handling routine in the driver software can read the error register again to get the 32-bit error address – the value will not change until the error bit is reset.

When written, the error register consists of a single bit, the error bit. This should be written as '1' in order to clear the error bit and re-enable error detection.

Figure 17: Wishbone module Error Register, as read

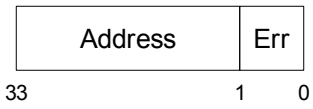
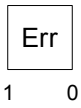


Figure 18: Wishbone module Error Register, as written



Bit #	Access	Description
33:1	R	**Address** When error bit = '1', contains the address of the failed transaction
0	R	**Err** *(when read)* Error bit. Set to '1' when a WishBone error has occurred since the last time the error bit was reset.
0	W	**Err** *(when written)* Write as '1' to reset the error bit to '0' and re-enable error detection

Table 17: WishBone module Error Register format

3.5 CPU Module Registers

Table 18 summarizes all of the registers present within the CPU sub-module.

Index	Register name
0x0	Status register

Table 18: CPU module register summary

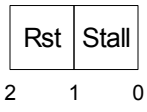
3.5.1 Status Register

The status register is used to control the reset line to the CPU, and to detect and control breakpoint conditions. The format is the same whether read or written.

Bit 1 of the register controls the reset line to the CPU. When written as a '1', reset to the CPU is active, and the CPU is put into a reset state. When written '0', the reset line is negated, and the CPU may run normally. The reset bit can only be changed by an internal register write in the CPU module.

Bit 0 of the status register detects breakpoints, and controls the stall line to the CPU. When written '1' via internal register write, the stall line to the CPU is made active, and the CPU stops executing instructions (but retains the ability to resume). When written '0', the stall line is negated, and the CPU resumes execution. The stall bit will also be set when the breakpoint output of the CPU goes active. The breakpoint output will be registered, the stall bit will be set, and the CPU will be held in the stall state by the ADI. This condition must be detected by polling in the software driver – once stalled due to a breakpoint, the CPU cannot resume execution until the stall bit is reset to '0' by an internal register access via JTAG.

Figure 19: CPU module Status register



Bit #	Access	Description
1	R/W	**Reset** Set to '1' to put the CPU in reset state. Set to '0' to allow the CPU to restart execution.
0	R/W	**Stall** Set to '1' to suspend execution in the CPU. Set to '0' to resume. Will be set to '1' automatically when a breakpoint indicator arrives from the CPU.

Table 19: CPU module Status Register format

4

IO Ports

4.1 TAP Ports

The Advanced Debug Interface connects to the TAP controller with the signals shown in Table 20.

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”
capture_dr_i	1	Input	TAP controller state “Capture DR”
rst_i	1	input	Reset signal.
debug_select_i	1	input	Instruction DEBUG is activated

Table 20: TAP Ports

4.2 CPU Ports

For each CPU module included, one set of I/O lines for that module will be present. The '*n*' in the port name will be replaced with the CPU module number, always starting from 0. These lines are shown in Table 21.

Port	Width	Direction	Description
cpu*n*_clk_i	1	input	CPU clock signal.
cpu*n*_addr_o	32	output	CPU address
cpu*n*_data_i	32	input	CPU data input (data from CPU)
cpu*n*_data_o	32	output	CPU data output (data to CPU)
cpu*n*_bp_i	1	input	CPU breakpoint
cpu*n*_stall_o	1	output	CPU stall (selected CPU is stalled)
cpu*n*_stb_o	1	output	CPU strobe
cpu*n*_we_o	1	output	CPU write enable signal indicates a write cycle when asserted high (read cycle when low).
cpu*n*_ack_i	1	input	CPU acknowledge (signals end of cycle)
cpu*n*_rst_o	1	output	CPU reset output (resets CPU)

Table 21: CPU Ports

4.3 WISHBONE Ports

The WishBone module, if included, will add a set of WishBone interface signals to the top-level IO, as described in Table 22.

Port	Width	Direction	Description
wb_clk_i	1	input	WISHBONE clock
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. (always false)
wb_cti_o	3	output	WISHBONE cycle type identifier indicates type of cycle (single, burst, end of burst) (always single)
wb_bte_o	2	output	WISHBONE burst type extension (always 0)

Table 22: WISHBONE Ports

5

Module Configuration

The Advanced Debug Interface supports three options, which allow the user to configure which sub-modules will be included when the design is synthesized.

Option: `DBG_WISHBONE_SUPPORTED`

Use: Define this option if you want to include a WishBone module in the ADI. Default is defined.

Option: `DBG_CPU0_SUPPORTED`

Use: Define this option if you want to include one or more OR1200 CPU debug modules. Default is defined.

Option: `DBG_CPU1_SUPPORTED`

Use: Define this option if you want to include a second OR1200 CPU debug module. Default is undefined.

One other WishBone-specific option can be found in adbg_wb_defines.v:

Option: `DBG_WB_LITTLE_ENDIAN`

Use: Define this option when the system CPU uses big-endian byte ordering. When left undefined, little-endian byte ordering (the OR1200 default) will be used. Default is undefined.

6

CRC Module

A CRC calculation module is contained within each WishBone and CPU sub-module. The CRC module is active only during burst read and write transactions. It calculates a 32-bit CRC on only the data bits of the transaction, starting with the LSB of the first word transferred, and ending with the MSB of the last word transferred.

In order to simplify the hardware, it was desirable to have the shift performed during the CRC calculation be in the same direction as the shift required to read out the CRC serially, LSB-first. This is the reverse of previous CRC implementations. As such, the CRC polynomial used is also the bitwise-reverse of the standard Ethernet CRC-32 polynomial: `0xEDB88320`. A C-language routine for computing a compatible CRC, up to 32 bits at a time, is reproduced below. When computing a CRC for multiple words, the output of the last call to `compute_crc()` should be passed to the next call as `crc_in`. When calling `compute_crc()` for the first word of a burst, `crc_in` should be set to `0xFFFFFFFF`.

```
#define CRC_POLY 0xEDB88320


uint32_t compute_crc(uint32_t crc_in, uint32_t data_in,
                        int length_bits)
{
uint32_t crc_out, c, d;
int i;

      crc_out = crc_in;
      for(i = 0; i < length_bits; i++) {
            d = ((data_in >> i) & 0x1) ? 0xffffffff : 0x0;
            c = (crc_out & 0x1) ? 0xffffffff : 0x0;
            crc_out = crc_out >> 1;
            crc_out = crc_out ^ ((d ^ c) & CRC_POLY);
      }

return crc_out;
}
```