ADVANCED DEBUG INTERFACE Implementation in Bluespec

A Project Report

Submitted In partial fulfillment of the requirements for the award of the degree of

MASTER OF TECHNOLOGY

In

Microelectronics and VLSI Design

(Electrical Engineering)

by

BHARATI MOHAN DATTAPRASAD
(EE14M050)

Under the guidance of

Dr. V. Kamakoti

DEPARTMENT OF ELECTRICAL ENGINEERING INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

JUNE 2016

THESIS CERTIFICATE

This is to certify that the thesis titled **ADVANCED DEBUG INTERFACE implementation in bluespec**, submitted by **Bharati Mohan Dattaprasad**, **(EE14M050)** to the **Indian Institute of Technology Madras**, for the award of the degree of **Master of Technology** in **microelectronics and VLSI design**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. V. Kamakoti
(Project guide)

Dept. of Computer Science
IIT-Madras,
Chennai 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my guide, **Dr. V.Kamakoti** for his valuable guidance, encouragement and advice. His immense motivation helped me in making firm commitment towards my project work.

My special thanks to Mr. G.S. Madhusudan for his encouragement and motivation throughout the project. His valuable suggestions and constructive feedback were very helpful in moving ahead with my project work.

I would like to thank my co-guide Dr.Nitin Chandrachoodan and faculty advisor **Dr.Deleep R Nair** who have patiently listened, evaluated, and guided us throughout the program.

My special thanks to my project team members Neel Gala, **Arjun Menon**, Rahul, Arnab for their help and support.

ABSTRACT

Key Words: JTAG Test Access Port (TAP), system AXI4 bus, OR1200 CPU SPR bus, System on Chip (SoC)

As VLSI engineer designs some circuits it needed to be verified. This verification can be pre and post fabrication. In verification we need to check weather SoC is working properly or not or if is there any error coming. Now when your processor is ready on SoC we need to check whether it is working or not. For that we use Software debugger which check whether our design under test (DUT) is working properly or not. But there should be some hardware which 'translate' debugger's request to our DUT.

Main objective of this project is to design this hardware which will help debugger to read from or write to SoC. Main purpose is accessing burst data.

ADF hardware contains JTAG TAP which is connected to SoC, OR1200 CPU SPR bus will connect to debug interface and ADI also uses AXI4 bus.

Table of Contents

1. INTRODUCTOIN

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 a SoC 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 interface has initially been developed for OpenRISC based processors, but it is universal and can also be used with other cores. In the PULP project it is used for both OpenRISC and RISC-V based cores.

Changes compared to the initial version:

- Replaced WishBone memory interface with AXI
- Support for 32 and 64 bit wide memory interfaces
- Support for up to 16 cores
- Intelligent handling of multi core debugging

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. The ADI appears as a data register within the TAP.

Internally, the ADI has connections to both a AXI4 bus and CPU debug interface. The AXI4 interface does not use any of the burst features of the bus; it should therefore be compatible with any version of the bus. The CPU interface is designed to connect to an OR1ON processor, or any other CPU which uses the same debug ports. Note that there are two versions of the OR1ON debug interface; the ADI is designed to use the newer version, which includes the strobe and acknowledge (dbg_stb and dbg_ack) signals

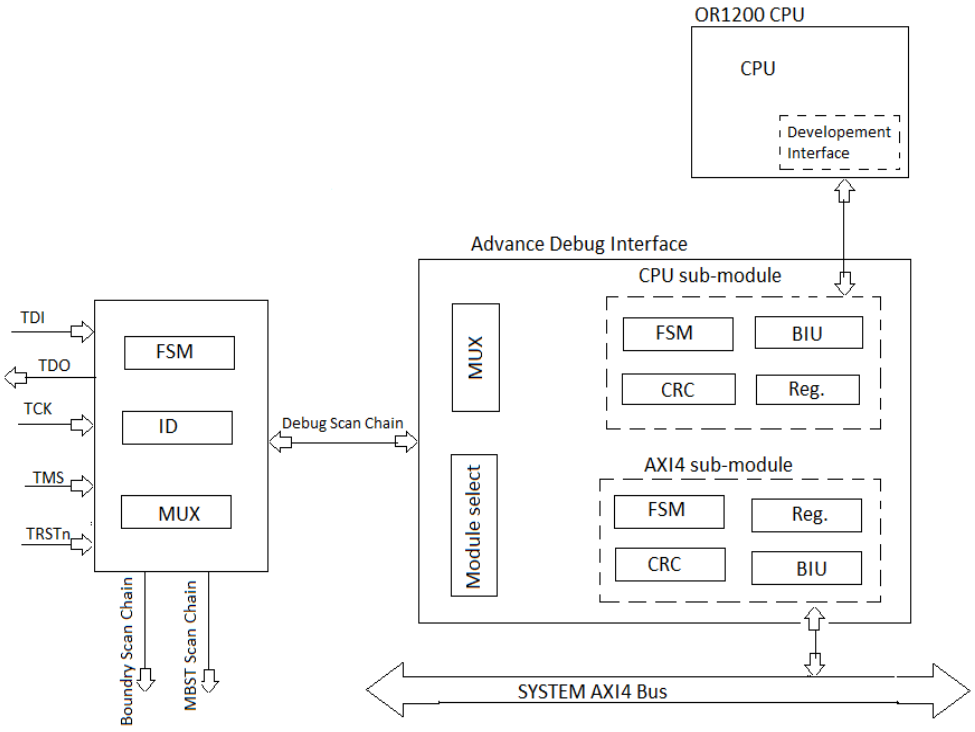


Fig. 1: Block diagram of the complete debug system hardware

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 AXI4 module and the OR1ON 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 AXI4 sub-module and one OR1ON 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:

1) *Internal module registers:*

   Contain information about the status of the module such as the error register in the AXI4 module, or they may control external I/O lines such as the reset and stall lines from the OR1ON module.

   Each sub-module using one or more registers 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.

2) *Bus interface:*

   The bus interface of most sub-modules is designed to allow the TAP to read or write data from or to a bus as quickly as possible.

   The bus interface of the OR1ON 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, and 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*.

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.

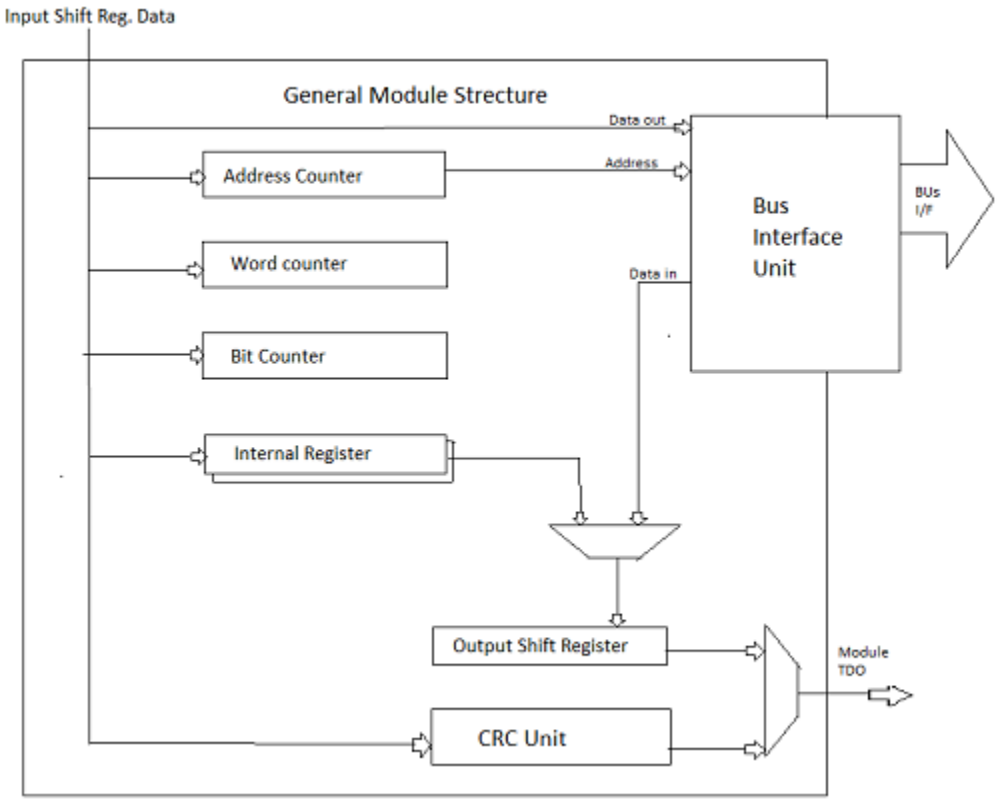


Figure 2: General Module structure

Burst transaction:

1) 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
2) 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 is determined by internal counters. In order to support multi-device chains, a “start bit” feature was added to burst mode.
3) 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 and the start bit is discarded.
4) *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. Next sections will elaborate them

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. 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.*

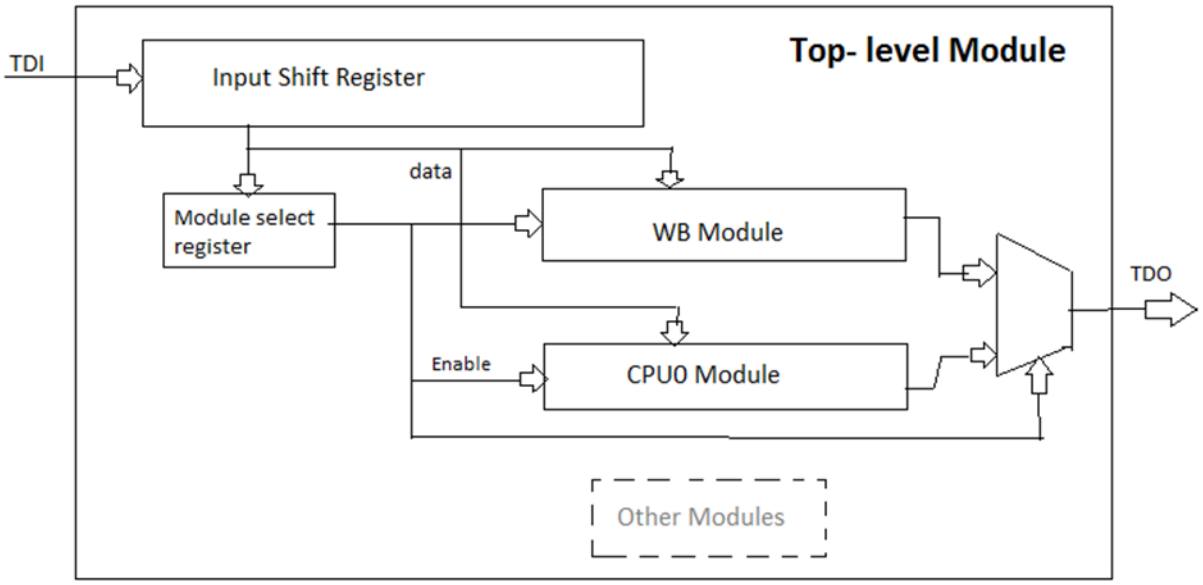


Figure 3: Block diagram of top- level module

2.3 AXI4 Module

The purpose of the AXI4 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 AXI4 module has a bus interface which follows the AXI4 standard.

Because the JTAG interface is relatively slow, burst accesses on the AXI4 bus interface are not used, therefore the interface should be compatible with all versions of the AXI4 standard.

The AXI4 module allows 8-, 16-, 32-bit and 64-bit reads and writes over the AXI4 bus interface. The AXI4 Module uses a 32-bit address and a 64-bit data bus, and allows burst transfers of up to 65535 words (524 KB).

The ADI may be synthesized in one of two modes: Which mode is used affects how overflow conditions are detected.

1) *Legacy mode:*

   In legacy mode, an overflow condition during a burst write can be detected by reading back a status bit after writing each word.

   If the status bit is true, then the transaction has succeeded (the AXI4 bus was ready to accept the word) 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.

2) *Hi-speed mode:*

   There is no status bit between data bytes, and overflows cannot be detected until after the burst transaction is finished. In hi-speed mode, an overflow is detected and treated in the same way as a AXI4 bus error, and is captured by the module's "error register".

In legacy mode, underflow conditions during burst reads are avoided by use of a "ready" bit. As soon as a data word is read from the AXI4 bus, the module shifts out a one, indicating that data is ready. Bus data follows immediately after a true ready bit is sent.

Hi-speed mode eliminates all ready bits in a burst read except the first one – driver software must only wait for a single valid ready bit at the start of a read transaction before transferring all of the data.

Which mode you select (legacy or hi-speed) depends on the relative clock speeds of your system's AXI4 bus and JTAG interface. Systems with AXI4 clock slower than the JTAG clock may require legacy mode. Most systems should select hi-speed mode, especially if a USB-based JTAG cable is used. A system in hi-speed mode using a USB JTAG cable is generally able to transfer AXI4 data an order of magnitude faster than a system with a legacy-mode ADI. I used hi speed mode since we use USB-based JTAG interface.

Error register: A 33-bit internal register, which is used to detect errors on the bus interface.

During a burst read or write, the AXI4 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. 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.

The AXI4 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 and if 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 LSB-first calculation was used to reduce hardware and routing requirements in the CRC module.

2.3 OR1ON CPU Module

The OR1ON CPU sub-module is designed to allow a software debugger to access the internal registers of a CPU, to stall the processor, and to take control when a breakpoint occurs in software.

The OR1ON sub-module is based on the AXI4 module, and is therefore similar. The bus interface connects to the debug interface of the OR1ON, 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 AXI4 module.

The OR1ON debug interface bus does not have an error indicator bit. Thus, the OR1ON module does not have an internal “bus error” register.

Since error register is not present, in hi-speed mode, overflows and underflows during burst transactions cannot be detected. So, we use legendary mode.

The OR1ON 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.

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 four major sections: the top-level ADI, the AXI4 module, and the CPU modules. 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.

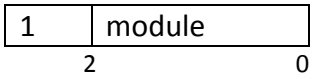
The ADI must be reset before any commands will be accepted or executed

3.1.1 Top module:

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
5	W	Top Level Select Set to '1' to select the top module
4:0	W	Module Number of the module to select. The following modules are valid: 0X0 = AXI4 module 0X1 = CPU module

Table 1: Module Select command format

3.2 AXI4 Commands:

A standardized command format in which, 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 summarize it.

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
0X4	Burst Setup Write, 64-bit words
0X5	Burst Setup Read, 8-bit words
0X6	Burst Setup Read, 16-bit words
0X7	Burst Setup Read, 32-bit words
0X8	Burst Setup Read, 64-bit words
0X9	Internal register write
0XD	Internal register select

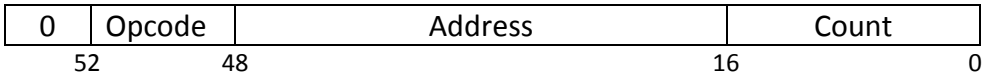
Table 2: AXI module command opcode summary

3.2.1 Burst Setup

A burst setup command prepares the AXI4 module to do either a read or write burst, in order to move data to or from a consecutive sequence of addresses on the AXI4 bus. The word size of the burst is decoded from the opcode. the address counter in the ADI is incremented by 1, 2, 4 or 8, for 8-, 16-, 32- and 64-bit words respectively.

After a burst setup command has been executed (in the UPDATE_DR state), the AXI4 module will enter 'burst read' or 'burst write' mode, the next time the TAP enters SHIFT_DR mode.

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 0X4 = Burst Write, 64-bit words 0X5 = Burst Read, 8-bit words 0X6 = Burst Read, 16-bit words 0X7 = Burst Read, 32-bit words 0X8 = Burst Read, 64-bit words
47:16	W	Address The first AXI4 address which will be read from or written to
0:15	W	Count Total no of words to be transferred

Table 3: AXI4 module Burst Setup commands formats

3.2.2 Burst Write

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 AXI4 as soon as it has been transferred in via JTAG. For a word size of *n* and a transfer of *m* words in hi-speed mode, the total length will be (*n* * *m*) + 34.

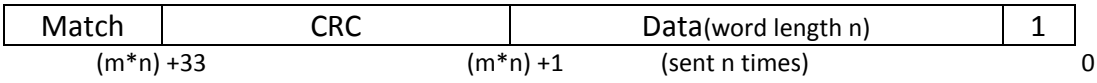
The first bit transferred in a burst write is a '1' start bit. This tells the AXI4 module to begin counting bits, and is required due to the possibility of multiple devices on the JTAG chain. After the start bit, one word of data is transferred into the ADI. The status bit tells the user whether the AXI4 was ready to accept the word just transferred; when true, the bus was ready. When false, the word was not written to the AXI4, and the software driver should retry the transfer, starting from the failed word.

Each data word is immediately followed by another data word, until all words have been transferred. Immediately following the last data word, a 32-bit CRC code is transferred into the AXI4 module. This CRC is compared with a CRC computed internally to the AXI4 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.

AXI4 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 AXI4 module error register to see if a bus error occurred during the burst, and if so, at what address.

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
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.

Table4: AXI4 module burst write format

3.2.3 Burst Read

In this mode, counters are used to determine position in the bitstream, and a word is read from the AXI4 while the previous data word is transferred out via JTAG. For a word size of *n* and a transfer of *m* words, the total length will be (*n* * *m*) + 33.

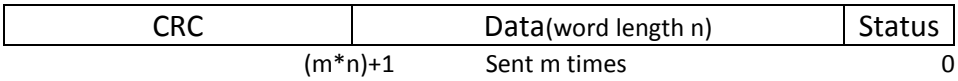
The first bit (or bits) transferred during a burst read, whether legacy or hi-speed mode, is a status bit. This bit indicates whether or not data from the AXI4 is ready to be transferred out via JTAG. The AXI4 module will send '0' bits until a word is ready, then send a single '1' bit. One data word will follow a '1' status bit.

Immediately following the last data word, a 32-bit CRC code is sent from the AXI4 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.

Same treatment for error bit as in burst write.

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. In hi-speed mode, this is only sent before the first data word. In legacy mode, this is sent before each data word.

Table5: AXI4 module burst read format

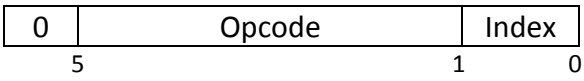
3.2.4 Register Select

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 AXI4 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. If command is of burst type then only one time command needed to give.

The register select command uses the same top-level select bit and opcode format as the other AXI4 module commands. The opcode is followed by a 1-bit value, which is the index of the register which should be made active.

Figure 8: Register Select commands formats



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 AXI4 module uses a 1-bit index.

Table 6: AXI4 module Register Select command format

3.2.5 Register Read

No specific command to read a module internal register.

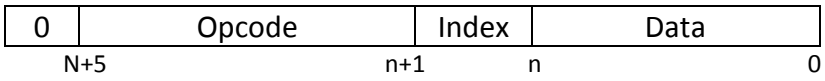
In order to read a particular register, make that register active using the register select command, then read out the value of the register while sending a NOP command. The register data will be LSB-aligned; that is, the LSB of the register data will be the first bit shifted out of the ADI. This allows the minimum number of JTAG bits to be transferred.

Only register data will be transferred out, no command, index, opcodes, start, or status bits will be sent with it. You can abort register read with command like NOP in 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. The value of the *previously* active register will be shifted out as this command is shifted in.

Figure 8: Register Select commands formats



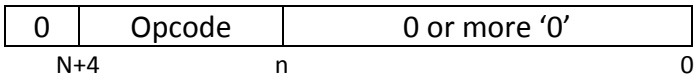
Bit#	Access	Description
5+ *n*	W	Top-Level Select Set to '0' for all sub-module commands
(4:1)+ *n*	W	Opcode Operation to perform. 0x9 = Internal Register Write
n	W	Index Index of the register to make active. The AXI4 module uses a 1-bit index.
(n-1):0	W	Data *n* bits of data to write to the register specified by Index. *n* depends on the register being written.

Table 7: AXI4 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 AXI4 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: AXI4 module NOP command format

3.3 CPU Commands

The CPU sub-module uses the same standardized command format as the AXI4 module. Each command must have a zero as the MSBit, to differentiate it from a module select command. In the next four MSB 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.

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

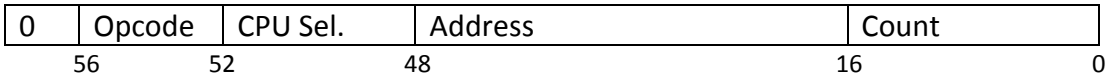
Table 9: CPU module command opcode summary

3.3.1 Burst Setup

A burst setup commands are used for CPU module to either read or write. . Since 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 OR1ON implementation uses a 32-bit address in its external debug interface. The ADI is designed to use the 32-bit OR1ON 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.

Figure 11: Burst Setup command



Bits #	Access	Description
56	W	Top-Level Select Set to '0' for all sub-module commands
52:55	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
48:51	W	CPU Select Select the CPU to write to
47:16	W	Address The first OR10N 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

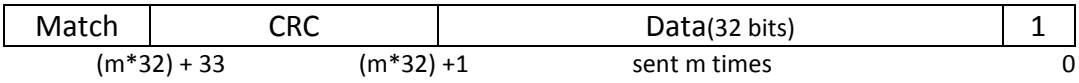
In this mode, counters are used to determine position in the bitstream, and a word is written to the CPU SPR bus as soon as it has been transferred in via JTAG. For a transfer of m words, the total length will be (32 * m) + 38.

In legacy mode, the first bit transferred in a burst write is a '1' start bit. This tells the AXI4 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. The status bit tells the user whether the CPU was ready to accept the word just transferred; when true, the bus was ready. When false, the word was not written to the SPR, and the software driver should retry the transfer, starting from the failed word.

In legacy mode, data transmission continues to alternate data word and status bit until all words have been transferred. In hi-speed mode, a data word is followed immediately by another data word, until all words are transferred.

A burst write transaction may be aborted at any time by moving the TAP through the UPDATE_DR state. Since no error bit, SPR bus does not provide any error indications beyond Start or Ready.

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
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

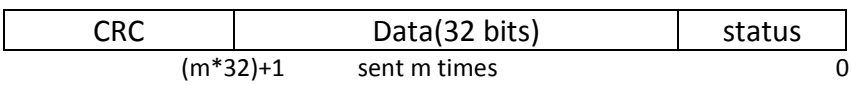
In this mode, counters are used to determine position in the bitstream, and a word is read from the CPU SPR bus while the previous data word is transferred out via JTAG. For a word size of *n* and a transfer of *m* words, the total length will be (*n* * *m*) + 33.

The first bit (or bits) transferred during a burst read, whether legacy or hi-speed mode, 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, and then send a single '1' bit. One data word will follow a '1' status bit.

Immediately following the last data word, 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.

Figure 13: Burst Read format



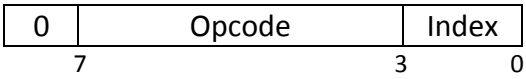
bits	Access	Description
32 bits	R	CRC 32-bit CRC computed on all of the data bits of the burst
n bits	R	Data Data word. Length specified by the opcode in the burst setup command. Sent *m* times.
1 bit	R	Status Read '0' until a word is ready to be sent, then a single '1' bit is sent before the data word.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

Table 12: CPU module burst read format

3.3.4 Register Select

This is same as AXI4 module Register select except The CPU module uses a 3-bit index.

Figure 14: Register Select command format



Bit #	Access	Description
7	W	Top-Level Select Set to '0' for all sub-module commands
6:3	W	Opcode Operation to perform. 0xD = Internal Register Select
2:0	W	Index Index of the register to make active. The CPU module uses a 3-bit index.

Table 13: CPU module Register Select command format

3.3.5 Register Read

This is same as AXI4 module Register read.

3.3.6 Register Write

This is same as AXI4 module Register select except The CPU module uses a 3-bit index.

Figure 15: Register Write command format

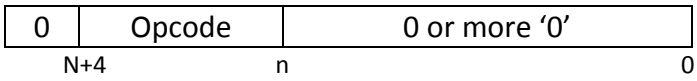
Bit #	Access	Description
7+ *n*	W	Top-Level Select Set to '0' for all sub-module commands
(6:3)+ *n*	W	Opcode Operation to perform. 0x9 = Internal Register Write
(2:0)+*n*	W	Index Index of the register to make active. The CPU module uses a 3-bit index.
(*n*-1):0	W	Data *n* bits of data to write to the 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 AXI4 Module Registers

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: AXI4 module register summary

3.4.1 Error Register

The error register captures AXI4 bus errors during burst transactions. Each time a bus access is completed, the AXI4 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 AXI4 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

Error in read, 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: AXI4 module Error Register as read

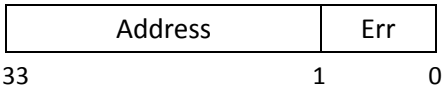
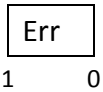


Figure 18: AXI4 module Error Register, as written



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 AXI4 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 16: AXI4 module Error Register format

3.5 CPU Module Register

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

Index	Register name
0x0	Status register
0x1	Group1 OR mask
0x2	Group1 AND mask
0x3	Group2 OR mask
0x4	Group2 AND mask

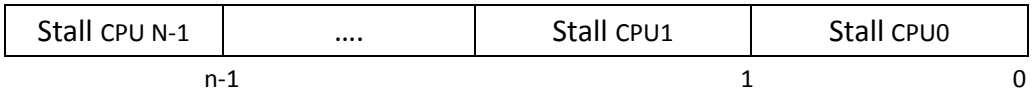
Table 17: CPU module register summary

3.5.1 Status Register

The status register is NB_CORES bit long and each bit of the status register detects breakpoint condition for a particular core, and controls the stall line to the same CPU. When written '1' via internal register write, the stall line to the CPU is made active, and the CPU stops executing instructions. 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



Bits#	Access	Description
15:0	R/W	Stall CPUn Set to '1' to suspend execution in the CPUn. Set to '0' to resume. Will be set to '1' automatically when a breakpoint indicator arrives from the CPU.

Table 18: 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.

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 AXI4 Ports

The AXI4 module will add a set of AXI4 initiator interface signals to the top-level IO

Port	Width	Direction	Description
wb_clk_i	1	input	AXI4 clock
wb_rst_i	1	Input	AXI4 reset signal
wb_ack_i	1	input	AXI4 acknowledge indicates a normal cycle termination
wb_adr_o	32	output	AXI4 address output
wb_cyc_o	1	output	AXI4 cycle encapsulates a valid transfer cycle.
wb_dat_i	32	input	AXI4 data input (data from AXI4)
wb_dat_o	32	output	AXI4 data output (data to AXI4)
wb_err_i	1	input	AXI4 error acknowledge indicates an abnormal cycle termination
wb_sel_o	4	output	AXI4 select indicates which bytes are valid on the data bus.
wb_stb_o	1	output	AXI4 strobe indicates a valid transfer.
wb_we_o	1	output	AXI4 write enable indicates a write cycle when asserted high (read cycle when low).
wb_cab_o	1	output	AXI4 consecutive address burst indicates a burst cycle. (always false)
wb_cti_o	3	output	AXI4 cycle type identifier indicates type of cycle (single, burst, end of burst) (always single)
wb_bte_o	2	output	AXI4 burst type extension (always 0)

Table 22: AXI4 Ports

5. Module Configuration

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

1) Option: DBG_AXI4_SUPPORTED:
   Used to include AXI4 module in the ADI.
2) Option: DBG_CPU_SUPPORTED:
   Used to include one or more OR1ON CPU debug modules.
3) Option: DBG_CPU_NUM:
   Used to define the number of OR1ON CPU debug module to be instantiated. Default is 4.
4) Option: ADBG_USE_HISPEED:
   Used to define this option to synthesize the AXI4 and OR1ON modules in hi-speed mode.

6. CRC Module

A CRC calculation module is contained within each AXI4 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.