White Paper
Randy Johnson
Stewart Christie
Development Tools
Product Marketing
Intel Corporation

JTAG 101

IEEE 1149.x and Software Debug

January 2009

321095

Executive Summary

JTAG (jay-tag) is one of the engineering acronyms that have been transformed into a noun, although arguably it is not so popular as RAM, or CPU. IEEE Std 1149.1-1990 IEEE Standard Test Access Port and Boundary-Scan Architecture is the official name, but JTAG is a bit snappier and is an abbreviation of Joint Test Action Group.

To help a user migrate to IA32, this white paper gives a quick overview of the various implementations, and names, of JTAG debug methods for users familiar with other processors

If you examine the standard's title you may be able to deduce the two use cases for JTAG devices and hardware. This paper deals with using the Test Access Port (TAP) as a means to control the execution of the processor, and to debug software via the TAP.

Not content with JTAG or IEE1149.1 as a name for this feature, most semiconductor vendors have also declared their own brand name version. To help a user migrate to IA32, this white paper gives a quick overview of the various implementations, and names, of JTAG debug methods for users familiar with PowerPC, ARM and MIPS processors and this is compared to the JTAG implementation in the Intel® Atom™ Microprocessor.

Contents

Background

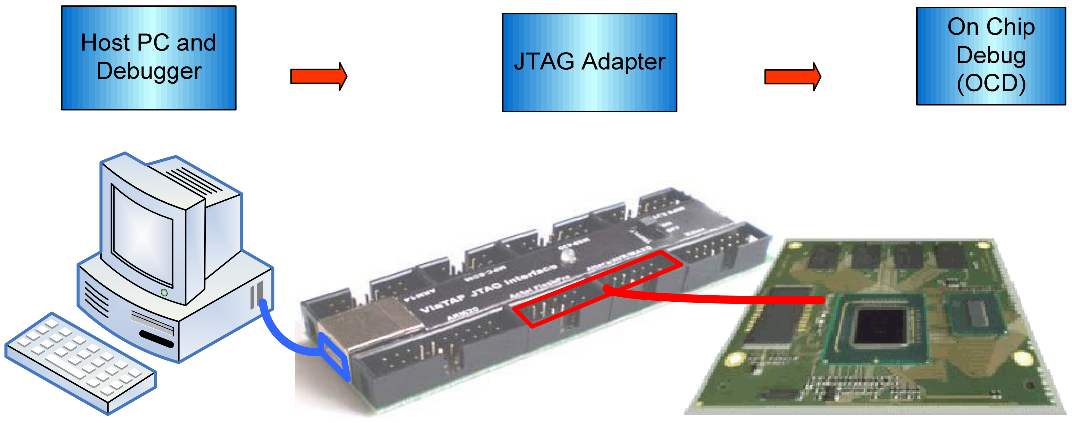
ONCE, BDM, OCD, NEXUS, XDP, are essentially different vendor names or trademark names for the same thing, JTAG, which itself is arguably an abbreviation that means different things to different people. When asked to describe JTAG, depending on your viewpoint and hardware/software experience, you will likely come up with a list that includes:

- An implementation of IEEE 1149.x for Board Test, or Boundary Scan testing.
- An appliance used to program on board flash or eeprom devices on a circuit board.
- A hardware device used to debug microprocessor software.
- A hardware device used to test a board using Boundary Scan.

All of the above are "correct", and can be mapped fairly well to the IEEE 1149.x standard. For the purposes of this paper, however we can exclude the Boundary Scan use cases, and concentrate on the use of JTAG for software debug.

In the general sense, on-chip debugging is a combination of hardware and software, both on and off the chip. The part that resides on the chip is implemented in various ways, and for the purpose of this paper it will be referenced as On-Chip-Debug (OCD). The part that resides off chip is often referred to as an In-Circuit-Emulator (ICE), or, for this paper, as a JTAG Adapter. JTAG Adapters are in effect, PC peripherals, and need a Linux* or Windows*-based PC to control them, and to present a user interface. Figure 1 shows a typical development system setup.

Figure 1. A Complete DeBug Solution Consisting of Three Distinct Components

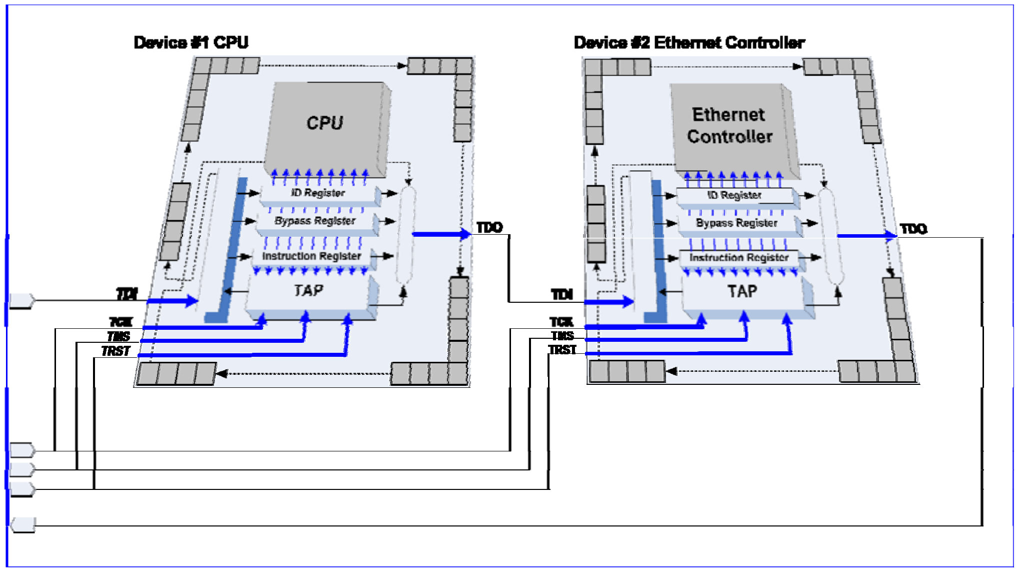


JTAG Overview

Test Access Point

The basic building block of a JTAG OCD is the Test Access Point or TAP controller. This allows access to all the custom features within a specific processor, and must support a minimum set of commands. In addition to a reset, commands exist to read and write registers beyond the TAP controller, and to bypass this device.   As you can see in Figure 2, there is generally only one Test Data In (TDI) and one Test Data Out (TDI) connected between the JTAG adapter and the TAP controller. In simplistic terms, for a 32 bit data word the TCLK line is therefore toggled 32 times to write data, and then 32 times to read data. Because of the dual use capability to perform boundary scan testing, the TDO line may not come directly from this TAP controller back to the JTAG adapter, but is routed serially through other devices on the board. This can add a large delay to the debug process, especially when multiple sets of these 32 bit words are needed to accomplish anything of significance. To minimize this delay the chip designers have often added two TAP controllers to large chips, and s/w debuggers will set the TAP controlling the large boundary scan chain to bypass mode.

Figure 2. Two Devices in a JTAG Scan Chain



Different processors implement OCD via JTAG in different ways. The 600 series of IBM* PowerPC* microprocessors purely use the hardware test chain which winds its way through many of the on-chip resources. Somewhere in the multi-thousand stage serial chain is the instruction register, for example. Debugging with this system is tedious since each core OCD action (modify memory location for instance) may take many trips through the entire JTAG chain. Although the debugger may only be interested in a 32 bit piece of the chain, all elements must be traversed, and multiple times. Downloading user code may be as slow as less than one hundred bytes per second (vs. over 20K per second with other methods). Another drawback to implementations that use a shared hardware test/software debug chain (TI DSP chips, 600 family PowerPC, etc.) is the way the chain is routed during chip design. Since this is typically the least critical path and the least critical part of the chip design/layout (as well it should be), the designers let the silicon auto-router layout the chain's pathway after the rest of the chip has been laid out. This means that each revision of the silicon may have a different JTAG chain; hence the host debugger software must be aware of every revision of silicon. This is a nightmare for software developers who must match the debugger with the silicon rev. TI solves this problem by often updating their OEM emulator software tool kit. This does not help the end-user unless he/she has a very reliable debugger vendor, who keeps up-to-date with the latest changes from the semiconductor vendor.

An alternative method to the JTAG OCD is to use a different chain via the JTAG port. This is allowed for in the IEEE specification. Using this method, one chain is available for the hardware test and debug of the chip, another for software debug. This method is used in the IBM* PowerPC* 400 series as well as in the SHARC* DSP from Analog Devices*. This secondary chain allows access to debug specific registers, usually only two or three are needed. In the IBM* chips, the debug port has access to an instruction stuff buffer, a debug control register and a debug status register. The instruction stuff buffer allows the debugger to stuff any opcode into the core processors' instruction register, in effect causing a single step to occur. By executing the proper instructions, any action needed may be performed. The debug control and status registers allow for the typical debug commands such as single step and run. Since a chain separate from the hardware test chain is used, the length of the chain is typically less than 50 bits long. There is some small overhead with each JTAG action to ensure that the proper chain is being accessed.

Terminology

Before we go any further, a note on terminology. "BDM" is a Motorola* term for a method of debugging. It also refers to a hardware port on their microcontroller chips, the "BDM port". Other chips and other manufacturers use a JTAG port (IBM*), a OnCE port (Motorola*), an MPSD port (Texas Instruments*), etc. We'll discuss these more later. The type of debugging we will be discussing is sometimes known as "BDM debugging" even though it may use a JTAG port. For clarity, it will be referred to as "on-chip debugging" or OCD. This will include all the various methods of using resources on the chip that are put there to enable complete software debug and aid in hardware debug. This includes processors from IBM*, Texas Instruments*, Analog Devices*, Motorola*, and others.

Two more definitions before we get started:

- USER MODE: This is defined as the normal operating mode of the processor, and encompasses all the sub modes a user program may enter, e.g. Interrupt mode, Supervisor mode, or kernel mode. Remember this is an embedded system and this definition of user is different from a Desktop PC and Microsoft* Windows* OS.
- DEBUG MODE: This is defined as the time when a JTAG debugger is "activated". Various vendors may call this Background Debug Mode.

On-chip Debug Methods

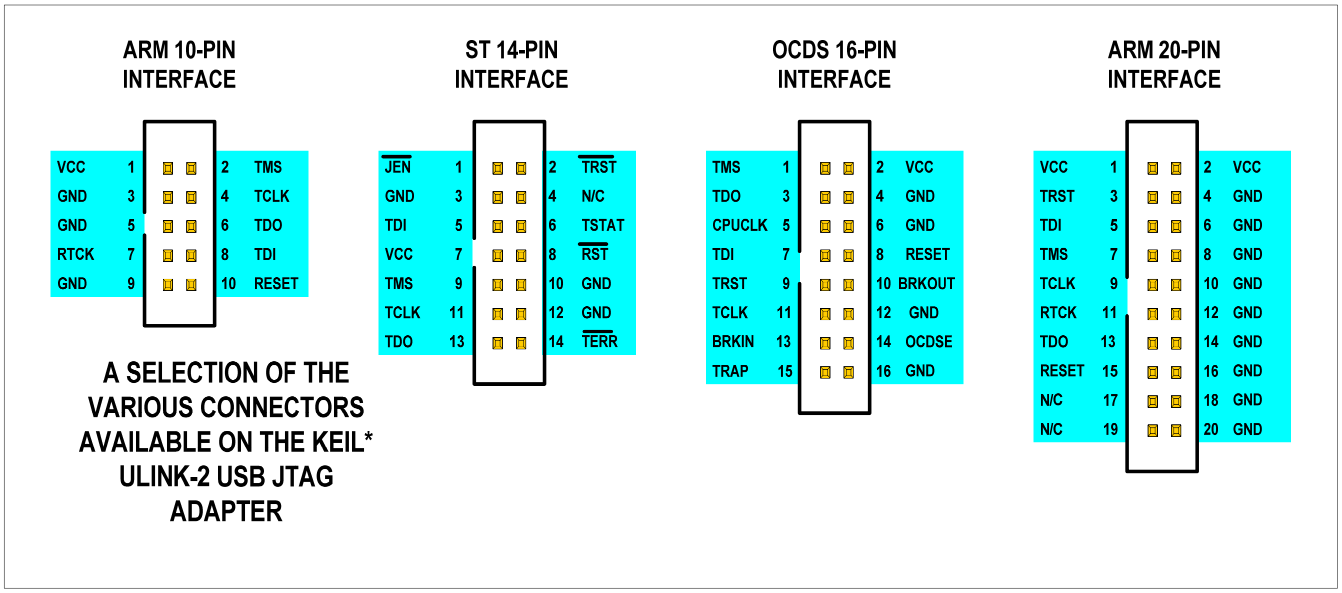
The latest addition to the debugger arsenal is on-chip debugging (OCD). Early on-chip debuggers were basically debug monitors written into the microcode of the target processor (Motorola* CPU32 family of processors). More advanced systems added other features such as real-time reading of the program counter (Analog Devices*' SHARC* processor) and near real-time reading of memory locations (Motorola's ColdFire). The basic OCD allows for code download, reading and writing memory and processor resources, single stepping, processor reset, and status (running or halted). Typically, on-chip peripherals may be set to shut down during OCD (as opposed to while the chip is executing user code).

Some processors enhance their OCD with other resources truly creating complete on-chip debuggers. The IBM* 4xx PowerPC* families of embedded processors have a seven wire interface (“RISCTrace”) in addition to the OCD (“RISCWatch”) that allow for a complete trace of the processor’s execution. Simply capturing these lines in real-time, a debugger can then display a full trace of the last x instructions executed.

In the general sense, on-chip debugging is a combination of hardware and software, both on and off the chip. The part that resides on the chip is implemented in various ways. It may be a microcode based monitor (Motorola* CPU32) or hardware implemented resources (IBM* PPC403). There may be resources used that are available to the end-user’s code such as breakpoint registers (most embedded PowerPC* implementations) or dedicated hardware purely used by the OCD such as instruction stuff buffers (also in embedded PowerPC implementations).

On-chip debugging does require some external hardware, however minimal it may be. There must communication between the chip and debugger host. In most cases this is via a dual-row pin header and several pins on the processor. The IBM* 403 family uses the JTAG port pins, in addition to RESET, power sense and ground, and connects via a 16 pin dual-row header. Motorola* BDM typically uses five dedicated pins (sometimes multiplexed with real-time execution functions), power, ground, and at least one reset, all terminating in a 10 pin dual-row header. Many of the DSP chips use a Texas Instruments*-style standard JTAG interface. Motorola has expanded the interface’s internal definition to include its DSP BDM equivalent, OnCE. ARM* processors use a variety of board connectors. The 10 and 20 pin versions are shown in Figure 3.

Figure 3. Example of ARM, ST and Infineon* JTAG Pinouts


ARM 10-PIN INTERFACE
ST 14-PIN INTERFACE
OCDS 16-PIN INTERFACE
ARM 20-PIN INTERFACE
A SELECTION OF THE VARIOUS CONNECTORS AVAILABLE ON THE KEIL* ULINK-2 USB JTAG ADAPTER

ARM 10-PIN INTERFACE

VCC	1	2	TMS
GND	3	4	TCLK
GND	5	6	TDO
RTCK	7	8	TDI
GND	9	10	RESET

ST 14-PIN INTERFACE

$\overline{\text{JEN}}$	1	2	$\overline{\text{TRST}}$
GND	3	4	N/C
TDI	5	6	TSTAT
VCC	7	8	$\overline{\text{RST}}$
TMS	9	10	GND
TCLK	11	12	GND
TDO	13	14	$\overline{\text{TERR}}$

OCDS 16-PIN INTERFACE

TMS	1	2	VCC
TDO	3	4	GND
CPUCLK	5	6	GND
TDI	7	8	RESET
TRST	9	10	BRKOUT
TCLK	11	12	GND
BRKIN	13	14	OCDSE
TRAP	15	16	GND

ARM 20-PIN INTERFACE

VCC	1	2	VCC
TRST	3	4	GND
TDI	5	6	GND
TMS	7	8	GND
TCLK	9	10	GND
RTCK	11	12	GND
TDO	13	14	GND
RESET	15	16	GND
N/C	17	18	GND
N/C	19	20	GND

JTAG Adapters

On-chip resources are only half the story. A target system with an OCD processor and its dual-row header are useless unless you have a host to communicate with. The host runs your debugger software and interfaces to the OCD header in various ways. The debugger on the host implements the user interface displaying your code, processor resources, target memory, etc. The hardware adapter may be one of many types. The simplest is typically a "wiggler", a device that interfaces the parallel port of an IBM*-type PC to an OCD header. This is both simple and slow. Other interfaces are serial port (RS-232) to JTAG converters, high speed parallel port to JTAG, Ethernet to JTAG, ISA bus card to JTAG and others. USB to JTAG is becoming increasingly popular, as legacy serial and parallel ports on PCs are becoming less popular. Many adapters support multiple families of processor, and often multiple semiconductor vendors with one adapter. Unfortunately each vendor often chooses their own variation of a JTAG board connector. The example shown in figure 2 is from a Keil* Ulink2* USB adapter. It supports two sizes of ARM* interface, an STMicroelectronics* 14 pin interface, and an Infineon* 16 pin interface.

A trip through most s/w labs using JTAG will show the importance of using polarized connectors and cables. Burned cables, damaged by reversing the connection, is arguably the most common failure mode of a new JTAG board bring up, and a source of many burnt fingers.

Many vendors now support GDB as a debugger option, including providing Eclipse* plugins. A fair number of vendors have proprietary solutions, bundled with their hardware adapter and the cost of host software runs from around $49 to several thousand dollars. Several vendors, for example Macraigor Systems*, provide free open source debugger software, but also license other debugger vendors, and semiconductor vendors to provide compatible solutions. The hardware typically costs from $100 to upwards of $5000 or more, depending on feature set and speeds.

JTAG Debugger Usage Models

Most users can recognize the obvious use for a JTAG Debugger when powering up a new board design for the first time. There will be no code in the ROM to initialize peripherals, and possibly no RAM either. A JTAG system can be used therefore to initialize devices connected to the processor, debug and test the RAM initialization code and parameters, and even to program a boot loader into ROM.

Similarly, device driver developers, or even OS developers use JTAG, especially in the initial stages. It can be extremely difficult to get the debug messages out an Ethernet port, or just maintain a link, while you are still debugging the TCP/IP stack.

For embedded device developers, as opposed to desktop programmers, a JTAG connection is often the only debug channel that can be relied upon, or that will not interfere with the regular devices operational mode. Let's examine, for example, a Las Vegas gaming machine developer's needs.

Regulatory Requirements

State gaming authorities would be highly suspect if developers could connect over a serial cable or Ethernet connection and debug a production model of a one-armed bandit. Medical devices and most aerospace applications, in fact any embedded device that is certified, are "sealed" from outside tampering. However, tampering is what is needed to debug code. JTAG can provide this, while still allowing full and secure use of an Ethernet connections used for the regular application. It isn't recommended to sneak a JTAG Adapter into a casino, as not only is it illegal but physical security and other methods are also used to prevent tampering.

Limited or Dedicated Resources

Many embedded devices are built with a single purpose in mind, and to save costs, unnecessary devices are not installed in a production unit. Many embedded devices do not even have a display. In a similar manner, again for a gaming device, the display is a dedicated device. The display cannot be used to display diagnostics or even a windowed debugger, while the user or the QA department, is trying to experience the user interface.

To get around the restrictions imposed by a missing or dedicated display or communication component, a JTAG device with a separate Host PC can extract the information needed, display source code, even single step through a display driver, all without disrupting the images shown on the user interface. The extra cost of a JTAG connector and some resistors need to be compared against the cost of holding up a production line, or the cost and time of debugging a system in the field, without the JTAG capabilities.

An advantage to using the JTAG port for software debug is that it does not need any additional pins on the processor for separate hardware and software debug. A disadvantage is the added overhead needed for each basic action.

A Review of Various Types of OCD

Overview

Several processor vendors have extended JTAG software debug, often to make it faster, or to add hardware trace capabilities. The NEXUS* consortium for example defines an auxiliary port containing a variable bit width MDO (message data out) bus. Examples of other technologies are explained in the different implementations below.

BDM (Motorola* CPU16, CPU32, ColdFire*, HC08, HC11, HC16)

As mentioned previously, Motorola* coined the term BDM (Background Mode Debug) with its CPU32 family of microcontrollers. This was followed by the CPU16 family, and then ColdFire*, and now encompasses the HC08, and HC11 families. These BDMs are extremely similar. They build upon the concept of a ROM monitor and have a similar command set. The core of the hardware interface consists of a serial data in, serial data out, serial clock/breakpoint, and freeze status signal. The commands are shifted into the chip serially and are 17 bits in length. The command set for the CPU32 is as follows:

Table 1. CPU32 Command Set

Command	Description
RAREG/RDREG	Read address or data register

Command	Description
WAREG/WDREG	Write address or data register
RSREG	Read system control register
WSREG	Write system control register
READ	Read memory
WRITE	Write memory
DUMP	Read memory block
FILL	Write memory block
GO	Run CPU
CALL	Call user patch code
RST	CPU reset instruction
NOP	Null command

These commands closely mirror those that have been used for years in ROM monitors. Single stepping is accomplished via hardware control of the BDM port or by placing a software breakpoint type of instruction in the code stream.

The processor is not aware of the BDM engine, it is not seen as an exception or interrupt. There is a “background” instruction, “BGND” which causes the processor to enter BDM when it is executed. BDM is left, and real-time code execution is resumed, upon the GO command being executed.

Embedded PowerPC* BDM (Motorola* MPC5xx, MPC8xx)

This BDM works quite differently from the CPU32 type of BDM. The hardware interface is similar with serial in, serial out, clock, reset, and status signals. The difference is that there is not a specific command set. Any serial stream entered into the chip is either 7 or 32 bits in length (not counting start, control, and length bits). Thirty-two-bit data streams go into the instruction stuff register and come out of the debug data register. What actually happens is that the host debugger stuffs PowerPC* opcodes into the processor and they are executed. This is actually a very powerful design allowing for all system resources to be accessible since this method gives the debug port the same power as executing system code. Seven bit data streams are used to control on chip breakpoint functions. Debug control registers exist to enable single stepping and other special controls.

The processor is “aware” of this BDM in that it is a CPU exception. BDM may be entered upon one of any number of exception causing events (invalid opcode, address bus misalignment, non-maskable interrupt, etc.) To resume real-time execution, the debugger stuffs a “return from exception” instruction, “RFI” into the processor’s instruction register.

OnCE (On-Chip Emulation)

The OnCE (On-Chip Emulation) interface is found on the Motorola* family of DSP chips. It allows for all the same type of debugging as the BDM interface. On most of the chips, the OnCE interface is implemented via dedicated pins. On the more recent parts, the OnCE engine is accessed via the JTAG port pins. The OnCE port is more complex than the BDM port in that it is a state machine controlled by the external debugger. The capabilities of OnCE include:

- Interrupt/break into debug mode on program memory address
- Interrupt/break into debug node on data memory address
- Interrupt/break into debug mode on an on-chip peripheral access
- Enter debug mode using a DSP microprocessor instruction
- Read/write any DSP core register
- Read/write peripheral memory mapped registers
- Read/write program or data memory
- Step one or more instructions
- Trace one or more instructions
- Save or restore current chip pipeline
- Read real-time instruction trace buffer
- Exit debug mode

ARM*

ARM* Licensees fall into two broad camps, architecture licensees and foundry/design/implementation licensees. The former are granted the rights to design their own CPU implementation compliant with the ARM Instruction Set Architecture. Intel has such a license, and used it to develop the Intel Xscale® processor family, and by implication, Intel's own variation on ARM OCD hardware that is covered in the next section.

Regular licensees can choose to implement various versions of the (ETM) Embedded Trace Macrocell*, or its big brother CoreSight*. ETM includes a FIFO trace buffer with pre and post trigger capture capabilities, and also include compression of the trace data, to shrink the amount of data needing extraction via the JTAG port. In addition to the standard JTAG single data bit, the ETM can be enhanced with a parallel trace port configurable from 1 to 32 bits. This high speed port has to be connected to a special Adapter such as the RealView* Trace 2.

Various options are also available for systems designers to reduce this pin count while still maintaining the capability to trace program execution.

Instead of an off chip trace buffer, the ETM can be extended by using an ETB (Embedded Trace Buffer) Macrocell*. The execution trace can be stored at full speed in the ETB, and read out a slower rate, at a later time through the JTAG interface. For customers who cannot even dedicate five pins to JTAG, ARM* now offers the SWB (Serial Wire Debug) a 2 pin interface that connects internally to standard JTAG TAP controllers.

Intel Xscale® Core-based Processors

For the family of Intel XScale® core processors, instead of a fixed function ROM-based implementation, the debugger downloads a debug handler into the mini-instruction cache of the target processor. The debug handler is code that runs on the target processor and communicates with the debugger through a set of JTAG registers. Normally the target code for debug, the application code for the target processor is typically stored in the Flash of the target circuit board, but since flash access is slower than RAM access, the application code normally decompresses itself into RAM and then runs from there. The mini-instruction cache is only used by the debug handler and cannot be accessed by the application code running on the circuit board. The debugger, the debug handler, and the application code have distinct roles that are illustrated in Table 2 below.

Table 2. Summary of Debugger, Debug Handler and Application Code Functionality

Debugger	Debug Handler	Application
Located on a separate computer	Located in the mini-instruction cache	Normally resides in the Flash at startup. Can run out of RAM after initialization
Selects Halt or Monitor Mode in the Debug Controls and Status Register (DCSR)	Performs commands issued from the debugger	Initializes the processor and circuit board
Performs and external debug break to connect to the target processor	The code in the debug handler uses exception traps and breakpoints to stop application code at desired locations	The code that is being debugged
Downloads the debug handler		
Issues commands to the debug handler		

MIPS*

Processors offer several different build time options for the chip hardware designers to choose from. In addition to regular hardware breakpoints and single stepping options MIPS* also has two optional trace options. MIPS original trace support is called PDtrace*, and a newer instructions trace only implementation is supplied by the iFlowtrace* hardware. PDTrace can be implemented using an on chip FIFO that records the program flow by saving a compressed version of the addresses executed. This method is also known as Branch Trace Message recording. The FIFO is emptied by the JTAG probe, and the debugger then derives the executed code stream by working forwards and backwards from a so called "sync message" that has a record of a complete address. Another novel feature of MIPS* trace is a sampling version that can be used to profile execution, or record cache hits or misses in the trace FIFO. One of the problems of using a JTAG interface to extract the FIFO data serially is the ever increasing execution speeds of modern processors. A back of the napkin calculation for a 1GHz Processor with a 32 bit address bus and an average of 2.5 addresses per instruction, clocks along at about 400M instructions per second. Even without any overhead that means a serial extraction of just the addresses needs around a 12Ghz bit clock. A reasonable maximum speed for a JTAG clock is around 100Mhz, and that's quite a bit slower than 12GHz. MIPS customers have the option of implementing a feature known as iFlowtrace*, that uses a double data rate (DDR) clock and a four bit parallel interface to extract data from the FIFO. When this is coupled with the ability to enable or disable the trace using on chip counters and registers, it can aid a user in ensuring that a code section of interest can be captured.

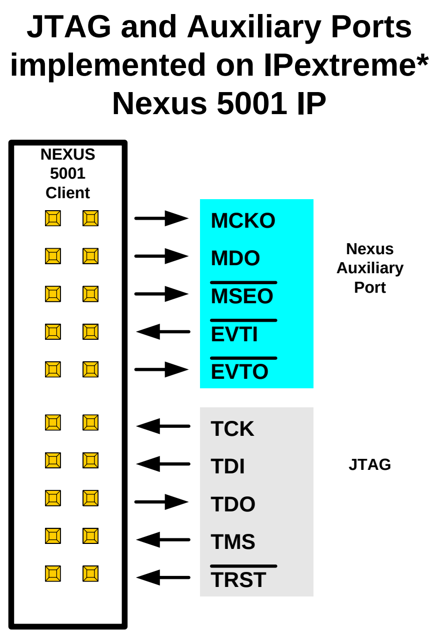
The Nexus 5001* Forum

The Nexus 5001* Forum initiative came about because of the unique needs of the auto industry's engine control development, and unlike the other examples is not processor or even semiconductor vendor specific. Engine manufacturers can tune the dynamic response of engines and drive components, and develop and test different fuel saving algorithms using chips with the NEXUS interface. For example, the combo of chip and JTAG adapter gives the designers the capability to swap two complete sets of calibration numbers as the cylinder hits Top Dead Center (TDC). Without this "instant' swap capability, a regular JTAG implementation would swap one constant at a time, and risk damaging the rotating engine components. Nexus is an open standard developed under the auspices of the Nexus 5001 Forum Standard for a Global Embedded Processor Debug Interface (IEEE-ISTO 5001 – 2003). A copy of the standard can be freely downloaded at www.nexus.org. A summary of the different levels of compliance is shown in Table 3 below. Most implementations of OCD that include trace are generally compliant up to class 2. Class 3 includes the capability to read or write memory while the processor is running, and Class 4 can remap memory or IO ports to the JTAG Adapter Device. Other features are optional, for example using the watchpoint signal to trigger the trace on or off.

Table 3. Debug Features Supported by the Different Classes of the Nexus 5001* Standard

Feature	Class 1	Class 2	Class 3	Class 4
Read/write user regs and memory in debug mode:	X	X	X	X
Enter Debug Mode from Reset or User Mode.	X	X	X	X
Exit Debug mode	X	X	X	X
Single step in user mode.	X	X	X	X
Enter Debug mode on instruction/data breakpoint.	X	X	X	X
Raise signal when watchpoint detected	X	X	X	X
Process ID Trace		X	X	X
Execution Trace		X	X	X
Real Time Data Write Trace			X	X
Real Time Data Read Trace			X	X
Real Time Memory Read and Write			X	X
Trigger Trace on Watchpoint				X
Trace Overflow Control				X

Figure 4. JTAG and Auxiliary Ports Implemented on IPExtreme* Nexus 5001 IP



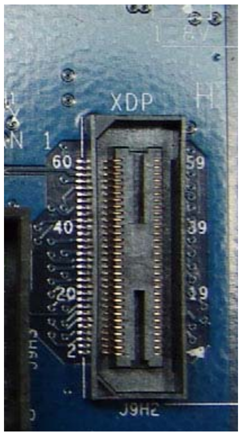
In addition to the regular five JTAG pins, the higher class features in class 3 or 4 may make it necessary to add the NEXUS* Auxiliary Port pins. The MDO (Message Data Out) bus can be expanded to multiple bits as necessary. For example the 56MHz PPC-based Freescale* MPC565 part has either a 9 pin or 16 pin debug interface, because the MDO bus can set to be 1bit or 8 bits wide.

Intel® 64 and IA32 Architectures

Intel incorporates logic in all of its processors and some of its chipset components to support the IEEE 1149.1-2001 standard for Test Access Port and Boundary-Scan Architecture. All of the traditional usages of the JTAG port, such as boundary scan, run control, BIST and debug hooks, are available on the Intel implementation as well. While the scope of this paper is JTAG and software debug, another white paper, *Designing Embedded Systems for Debugging and Testing* covers some of these other topics.

Intel's approach to the debug of the target system environment is to put what is called an extended debug port or XDP down on the target board. All current Intel reference designs and CRB's have an XDP for use by internal Intel design and validation teams. The standard XDP connector is a Samtec* 60-pin BSH-030-01 series connector or equivalent.

Figure 5. 60-pin XDP Connector



The XDP provides additional silicon /system debug resources compared to other debug-port implementations and also for expansion for future capabilities. There are additional proprietary capabilities built into the XDP interface accessible only to Intel personnel. Using XDP could save valuable time if help in debug by Intel is required. As a result, Intel supplies Debug Port Design Guides to OEM's and board manufacturers. These guides provide detailed information on how to design in the debug port capability.

The table below shows the pin outs for an Intel® Atom™-based Moorestown platform's XDP.

Table 4. CPU XDP Connector Pin Out for Intel® Atom™-based Moorestown Platform

Pin	XDP Signal Name	Target Signal	I/O	Device	Pin	XDP Signal Name	Target Signal	I/O	Device
1	GND	GND	NA		2	GND	GND	NA	
3	OBSFN_A0	XXPREQ_B	I/O	CPU	4	OBSFN_C0	Open	NA	
5	OBSFN_A1	XXPRDY_B	I/O	CPU	6	OBSFN_C1	Open	NA	
7	GND	GND	NA		8	GND	GND	NA	
9	OBSDATA_A0	XXBPM[0]_B	I/O	CPU	10	OBSDATA_C0	Open	NA	
11	OBSDATA_A1	XXBPM[1]_B	I/O	CPU	12	OBSDATA_C1	Open	NA	
13	GND	GND	NA		14	GND	GND	NA	
15	OBSDATA_A2	XXBPM[2]_B	I/O	CPU	16	OBSDATA_C2	Open	NA	
17	OBSDATA_A3	XXBPM[3]_B	I/O	CPU	18	OBSDATA_C3	Open	NA	

Pin	XDP Signal Name	Target Signal	I/O	Device	Pin	XDP Signal Name	Target Signal	I/O	Device
19	GND	GND	NA		20	GND	GND	NA	
21	OBSFN_B0	Open	NA		22	OBSFN_D0	Open	NA	
23	OBSFN_B1	Open	NA		24	OBSFN_D1	Open	NA	
25	GND	GND	NA		26	GND	GND	NA	
27	OBSDATA_B0	Open	NA		28	OBSDATA_D0	Open	NA	
29	OBSDATA_B1	Open	NA		30	OBSDATA_D1	Open	NA	
31	GND	GND	NA		32	GND	GND	NA	
33	OBSDATA_B2	Open	NA		34	OBSDATA_D2	Open	NA	
35	OBSDATA_B3	Open	NA		36	OBSDATA_D3	Open	NA	
37	GND	GND	NA		38	GND	GND	NA	
39	HOOK0	PWRMODE[0]1	I	system	40	HOOK4	XXBCLK[0]1	I	CPU
41	HOOK1	PWRMODE[1]1	I	system	42	HOOK5	XXBCLK[1]1	I	CPU
43	VCC_OBS_AB	VCCPAOAC	NA	system	44	VCC_OBS_CD	CPU VCCPAOAC	NA	system
45	HOOK2	PWRMODE[2]1	I	system	46	HOOK6	PMIC GPIO 0	O	PMIC
47	HOOK3	Open	NA		48	HOOK7/DBR#	DBR#	O	system
49	GND	GND	NA		50	GND	GND	NA	
51	SDA	SDA1	I/O	system	52	TDO	XXTDO	I	CPU
53	SCL	SCL1	I/O	system	54	TRSTn	XXTRST_B	O	CPU
55	TCK1	Open	NA		56	TDI	XXTDI	O	CPU
57	TCK0	XXTCK	O	CPU	58	TMS	XXTMS	O	CPU
59	GND	GND	NA		60	GND (XDP_PRESENT #)	-	NA	

Note: All the JTAG signals are included on the XDP along with other signals used by Intel in test and validation.

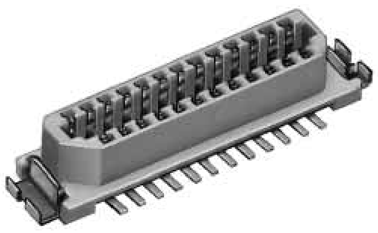
An interesting implementation feature of the Intel® XDP is the presence of a second TCLK, TCK1. The implantation allows for two scan chains to be run simultaneously through the XDP. Generally speaking the processor(s) will run at a higher clock rate than other components on the board which may be in the scan chain. Having the processor(s) on one scan chain and the other devices on the second will allow for faster through put times on test scans. Because only one of the scan chains is active at any given instance, it is possible to multiplex the remaining signals on the port.

Many mobile or embedded designs do not readily support the XDP design on the board, due to real estate issues involving the number of signals evolved.

As a result, Intel has provided alternative designs for the XDP with lower pin counts.

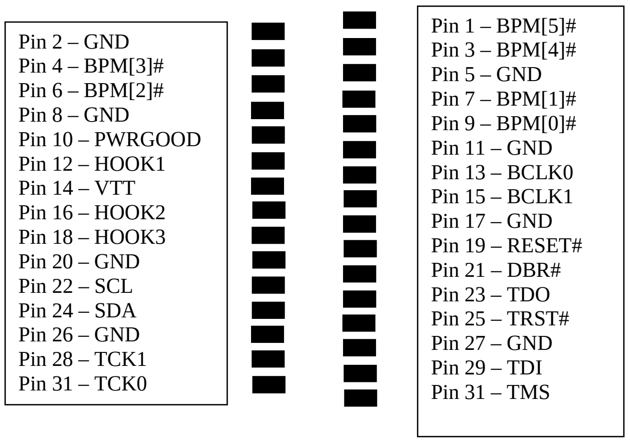
One is the XDP Secondary-Side Attach XDP. This option is best suited for customers who cannot place an XDP on the primary side of a platform and where the debug port will not be populated in the standard bill of materials configuration. In this case a connector and adapter solution is available. This solution will reduce the risk of hand soldering an XDP on the secondary side of a product. Historically, fine-pitch 60-pin XDPs installed on the secondary side have been prone to failure due to the difficulty of hand soldering the connector onto the board. The XDP Secondary-Side Attach (SSA) connector has a much wider pitch compared to the standard XDP to ease the task of hand soldering. The XDP-SSA connector, placed on the target system, is a Hirose* 31-pin DF9C-31S series connector. Specific plating types, mounting pads, and alignment pins versions of this connector can be obtained from Hirose.

Figure 6. XDP SSA Connector



An adapter is required to convert the 31-pin XDP-SSA connector to a standard 60-pin XDP, for use with tools like the American Arium* ECM-XDP3 In-Target Probe.

Figure 7. XDP-SSA Connector Pin Layout

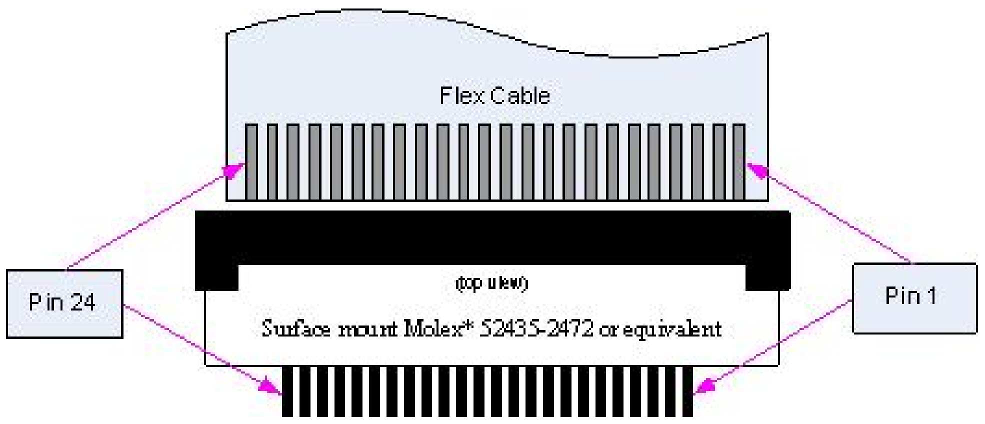


Another option is the XDP-SFF-24 Pin which is a slightly smaller connector than the XDP connector; however it has a much smaller keep-out volume because of the connection methodology (flex edge connector). As such many users prefer this connection method for systems that are extremely tight on physical space such as mobile systems and blade servers.

The XDP-SFF-24Pin connector is a 24-pin surface mount device, Molex* 52435-2472 or equivalent, which is RoHS compliant or equivalent. Specific plating types, ELV and RoHS compliant information of this connector can be obtained from Molex.

An adapter is also required to convert the 24-pin XDP-SFF-24Pin connector to a standard 60-pin XDP. As there are fewer pins, both the XDP-SFF-24Pin and the XDP-SSA have fewer capabilities than an XDP; however the JTAG functionality is the same for all three.

Figure 8. top view of Surface Mount XDP SFF-24 Pin Connector



Intel works with several run-control vendors to create tools including American Arium* and ASSET InterTech Inc*.

The American Arium* Sourcepoint* debugger software and their ECM-XDP3 In-Target probe provide excellent run control and debug capabilities on Intel processors via the JTAG interface.

ASSET InterTech* provides tools for boundary scan, BSDL validation. Intel QPI IBIST testing is conducting using ASSET InterTech's ScanWorks* tool suite.

Other software and hardware debugger developers are planning to come on line with new JTAG software debug capabilities for the IA32 platform in 2009.

Additional Reading

Many of the JTAG Adapter vendors and semiconductor suppliers have excellent articles that explain the intricacies, and recommend tips and tricks to help you get the most out of their solutions. Several recommendations are listed below.

JTAG Standard

http://standards.ieee.org/reading/ieee/std_public/description/testtech/1149.1-1990_desc.html

Macraigor Systems* original article

The Zen of BDM by Craig Haller, Managing Partner of Macraigor Systems* granted permission for use of a large section of his article in writing this white paper.

http://www.macraigor.com/zenofbdm.pdf

IPextreme* Inc.: Freescale* Nexus 5001 Software Debug Interfaces

IPextreme* Inc. brings intellectual property (IP) from large semiconductor companies to Chip (SOC) designers. They are currently supporting Freescale's implementation of the NEXUS standard.

http://www.ip-extreme.com/IP/nexus_5001.html

Debugging Intel® Xscale™ Platforms

This application note describes the usage models of the debug handler loaded in the mini-cache of Intel Xscale® processors to enable JTAG debuggers.

http://www.arium.com/support/pdf/XScale_app_note.pdf

Boundary Scan Tutorial 2007.

An 87 page boundary scan PDF tutorial for board developers and test developers, with a good deal of interest to software developers also. ASSET InterTech* is a company specializing in the Boundary Scan usage of JTAG.

http://www.asset-intertech.com/pdfs/Boundary-Scan_Tutorial_2007.pdf

Conclusion

JTAG features and functionality continue to be added to processors by silicon vendors. Companies such as FS2* and IP-Extreme* are offering ready made solutions that can be used to enhance what was once a simple run-control interface by adding enhanced logic-analyzer style features. As multicore processors move deeper into the embedded space features that once were built into the adapter are moving on-chip, to enable closer synchronization of start and stop commands, or to timestamp and correlate execution trace for each core.

JTAG for board test was introduced when signals between chips were slower and discrete pins on parallel busses were the norm. With high speed, self training, serial interfaces such as SATA and PCI-Express*, changes are underway to enhance JTAG for boundary scan also.

The combination of these two uses for the JTAG interface, coupled with the addition of more features on the chip mean that users need to be able to depend on their JTAG Adapter to support future requirements. Some vendors offer modular extensions, e.g. a trace module, while others can upgrade features by downloading new logic into FPGAs and new firmware.

The intent of this paper is to help you negotiate the various silicon and hardware adapter vendor solutions and to make sense of the options available.

Readers with specific questions, or feedback, are encouraged to submit topics to the Embedded Device Forum located at http://communities.intel.com/community/embedded.

Authors

Randy Johnson is a Product Marketing Engineer with the Embedded and Communications Group at Intel Corporation.

Stewart Christie is a Product Marketing Engineer with the Embedded and Communications Group at Intel Corporation.

Acronyms

ONCE	ON Chip Emulation – Freescale* DSP
BDM	Background Debug Mode – Freescale* Coldfire*
OCD	On-Chip Debugging – Macraigor's JTAG Adapter (OCDemon), Wind River Workbench, On-Chip Debugging Edition, Olimex JTAG Adapter (ARM-USB-OCD)
Nexus*	IEEE-ISTO 5001™-2003 is a standard initially developed for the debug and development of automotive powertrain applications and Engine Control Units.
XDP	Intel® eXtended Debug Port http://download.intel.com/design/Pentium4/guides/31337301.pdf