Where there's a JTAG, there's a way: obtaining full system access via USB

Maxim Goryachy and Mark Ermolov

Everyone makes mistakes. These words are certainly true for developers involved in low-level coding, where such common tools as print debugging and software debuggers run into limits. To solve this problem, hardware developers use in-circuit emulators or, if available on the target platform, the JTAG debugging interface (IEEE1149.1 [1]). Such debugging mechanisms first appeared in the 1980s [2]. Over time, microchip vendors extended the functionality of these interfaces. This allowed developers to obtain detailed information on power consumption, find bottlenecks in high-performance algorithms, and perform many other useful tasks.

Hardware debugging tools are also of interest to security researchers. These tools grant low-level system access and bypass important security protections, making it easier for researchers to study a platform's behavior and undocumented features. Unsurprisingly, these abilities have attracted the attention of intelligence services as well [3].

For many years only a limited audience had access to these technologies for Intel processors, due to the need to own expensive specialized equipment. But with Skylake processors, the situation changed in a big way: debugging mechanisms were built into the Platform Controller Hub (PCH) [4], which opened up this powerful tool to ordinary users, including malicious ones, who could use it to gain total control over the processor. For security reasons, these mechanisms are not activated by default, but as we show in this article, they can be activated on the equipment sold in common computer stores.

Figure 1. Intel I2ICE, one of the first in-system debuggers for Intel 80386 processors (recycledgoods.com/intel-series-iv-emul-system-iii514b.html)

EVOLUTION OF DEBUGGING TOOLS ON INTEL PROCESSORS

From in-circuit emulator to JTAG

Initially, an in-circuit emulator (ICE) for Intel 80286 processors was a separate computer ("the big blue box" [5]). The ICE was connected instead of the processor in the system to be debugged, and emulated its behavior. This emulator allowed setting breakpoints, modifying memory and processor registers, and performing reads and writes.

Later, Intel introduced the I2ICE hardware debugger (Figure 1), which did not require replacing the integrated processor. Instead, developers connected the I2ICE to the debugged system using special adapters. Communication with the host computer involved a standard serial connection at the speed of 9600 baud [5].

As technology progressed and bitrates increased, Intel ceased to develop stand-alone full-featured debuggers and started to partially relocate them inside the processor in the form of a special undocumented ICE mode. (In design, ICE resembled another mode called System Management Mode (SMM), and some developers at the time strongly believed that SMM was nothing more than a documented and extended ICE mode [6].) Standardization of debugging mechanisms in electronics, in turn, resulted in the IEEE1149.1 (JTAG) test interface supported on some Intel 80486 processors [7].

Joint Test Action Group (JTAG) is the name of the team that developed the Standard for Test Access Port and Boundary-Scan Architecture (IEEE1149.1 [1]). This document describes standardized testing and debugging equipment for a wide range of devices. Eventually, the JTAG abbreviation began to be associated with the IEEE1149 standard. JTAG is widespread in modern industrial microchips and is used for testing, installing firmware, debugging, and final factory-line inspection. In terms of physical implementation, JTAG is four or five dedicated pins that form the Test Access Port (TAP). The JTAG standard supports device chaining, which allows access to any connected device (Figure 2).

Hardware developers often extend JTAG functionality with new features, and Intel processors are no exception. Starting with Pentium processors, Intel introduced a more affordable and powerful stand-alone debugger that uses a special probe mode.

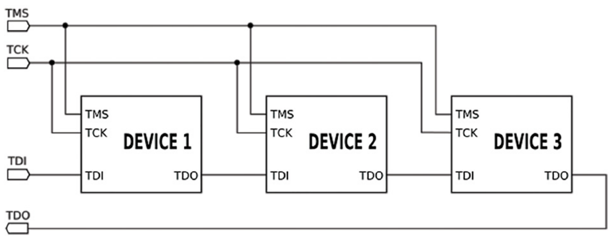


Figure 2. Connecting debugged devices in a JTAG chain

Probe mode

Probe mode is yet another undocumented mode of Intel processors. It is used to perform diagnostics and debugging. This mode is impossible to activate without access to the JTAG registers of a CPU. Probe mode allows the processor only to modify memory and read/write using I/O ports. In this mode, normal execution of instructions is interrupted and the processor switches to a dormant state while awaiting JTAG commands. This behavior differentiates probe mode from ICE mode, in which the processor continues to perform instructions. When probe mode is entered, prefetching and decoding of commands stops. Commands from JTAG to modify or read are fed directly into the processor execution units, bypassing the prefetch and decode stages [8], which allows access to a number of registers not accessible in standard modes.

Probe mode is implemented as a JTAG extension with several added registers and signals (R/S#, PRDY) (see [8] and [9] for more details on probe mode implementation). Third-party companies also produce JTAG adapters for x86 processors supporting this extension, but only Intel debuggers will be considered in this article.

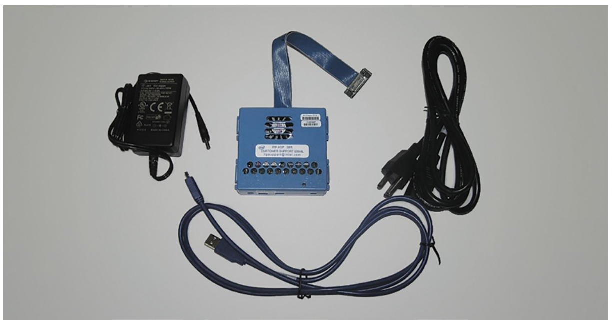


Figure 3. ITP-XDP

Modern hardware and technologies for Intel processor debugging

Modern Intel processors offer JTAG via three interfaces:

+ Intel In-Target Probe eXtended Debug Port (ITP-XDP) (Figure 3)
+ Intel Direct Connect Interface (DCI) makes JTAG available via USB 3.0 by means of either of the following connection types (Figure 4):
    + USB3 Hosting DCI (USB debug cable)—an ordinary DbC cable
    + BSSB Hosting DCI (Intel SVT Closed Chassis Adapter)—a specialized adapter (Figure 5)

Intel ITP-XDP has a closed protocol and requires a special board socket and specialized software (Intel System Studio, a trial version of which is available on the vendor's website). Other disadvantages are high price (around USD $3,000) and the necessity to sign a Corporate Non-Disclosure Agreement [10]. The high price and CNDA put this debugger beyond the reach of ordinary developers and home users.

However, starting with Skylake processors, Intel introduced Direct Connect Interface (DCI) technology. A rather superficial description is available in the Intel documentation [4]. DCI aims to simplify mobile device development, which results in the following drawback: it can be activated without hardware modifications. The Intel SVT adapter connection uses USB 3.0 pins but

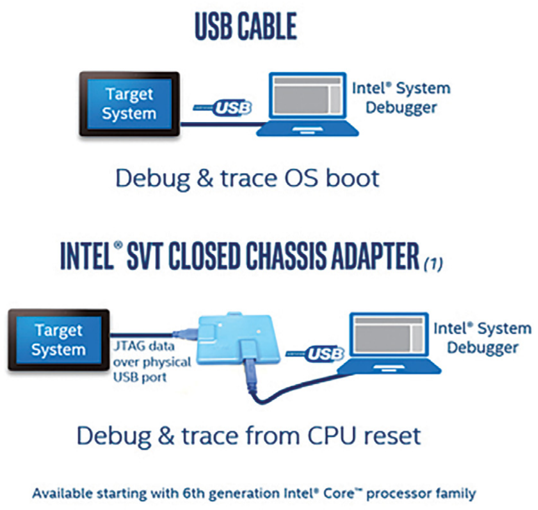

USB CABLE
Target System
USB
Intel® System Debugger
Debug & trace OS boot
INTEL® SVT CLOSED CHASSIS ADAPTER (1)
Target System
JTAG data over physical USB port
USB
Intel® System Debugger
Debug & trace from CPU reset
Available starting with 6th generation Intel® Core™ processor family

Figure 4. DCI connection types

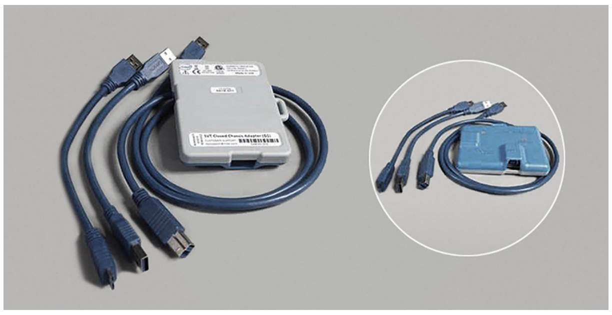


Figure 5. Intel SVT Closed Chassis Adapter

implements its own protocol, making it possible to work with the target system in deep sleep mode. Unfortunately, the SVT adapter, though relatively inexpensive ($390), is also available for purchase only after a CNDA is signed.

The most attractive option for an ordinary programmer (and one that does not involve signing any documents) is USB3-hosted DCI. The JTAG interface is accessed over a common USB 3.0 debug cable. As soon as DCI is activated on the target PC, the USB 3.0 port switches to slave mode and starts accepting commands from the host system.

One important question about USB3-hosted DCI is whether any external USB 3.0 port is suitable for DCI connections or a debug port available only on special development boards is required. Let's review this issue in detail.

There is confusion among system developers caused by the fact that debugging over USB has existed since long ago (USB 2.0) and is currently widely used by developers for software debugging of OS kernels and UEFI applications. However, software debugging via USB (in windbg or UEFI debug agent) has nothing to do with hardware debugging via JTAG, with the exception of transport. The USB 2.0 controller specification (Enhanced Host Controller Interface, EHCI) includes a debug port (PCI capability) for interaction between a server (software or hardware) on the debugged computer and the host client. In particular, the Windows kernel supports debugging via the EHCI Debug Port, which requires a USB 2.0 debug cable with integrated USB 2.0 device. In this case, not every external USB 2.0 port could work as a debug port: this option was assigned to certain ports which might, depending on the hardware manufacturer, be only on the inside of the case. Therefore, developers deliberately looked for hardware with an external debug port for debugging over USB. So Debug Port ability is an attribute of a specific USB port.

However, the situation changed entirely with USB 3.0 and its eXtended Host Controller Interface (XHCI) specification. This specification also supports debugging over USB, but it has undergone fundamental changes and is now called USB Debug Capability (DbC). According to the specification, DbC is not a port attribute, but a property of a specific XHCI controller. So if an XHCI controller supports DbC, then debugging over USB 3.0 is possible over any USB 3.0 port, including external ports. DbC automatically selects the first port to which the client implementing USB 3.0 transactions connects a debug cable.

It is important to note that the first XHCI controllers did not support DbC, making these controllers useless for debugging over USB. Then, starting with the 100 series (for Skylake), Intel introduced their own XHCI controller supporting DbC on PCH. Intel DCI (in Skylake and later) uses USB 3.0 DbC as transport for the JTAG client connection. It does not use USB 2.0 Debug Port technology.

What does this all mean? Any USB 3.0 port can be used for access to DCI and JTAG debugging.

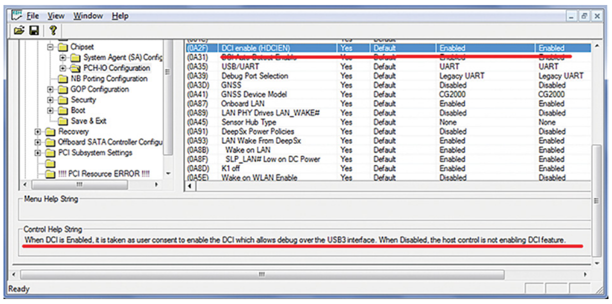
DCI ACTIVATION

How can we activate DCI? We found three solutions:

+ EFI Human Interface Infrastructure
+ PCH Strap (Intel Flash Image Tool)
+ P2SB device

Activation via EFI Human Interface Infrastructure

EFI Human Interface Infrastructure (HII) is a special mechanism for creating a user interface in the UEFI, as well as processing and managing user input. The architectures of modern UEFI BIOS software include many hidden options that are user-inaccessible but processed nonetheless. This is the basis for our first solution.



(0A2F)	DCI enable (HDCIEN)	Yes	Default	Enabled	Enabled
(0A31)	~~DCI Auto Detect Enable~~	~~Yes~~	~~Default~~	~~Enabled~~	~~Enabled~~
(0A35)	USB/UART	Yes	Default	UART	UART
(0A39)	Debug Port Selection	Yes	Default	Legacy UART	Legacy UART
(0A3D)	GNSS	Yes	Default	Disabled	Disabled
(0A41)	GNSS Device Model	Yes	Default	CG2000	CG2000
(0A87)	Onboard LAN	Yes	Default	Enabled	Enabled
(0A89)	LAN PHY Drives LAN_WAKE#	Yes	Default	Disabled	Disabled
(0A45)	Sensor Hub Type	Yes	Default	None	None
(0A91)	DeepSx Power Policies	Yes	Default	Disabled	Disabled
(0A93)	LAN Wake From DeepSx	Yes	Default	Enabled	Enabled
(0A8B)	Wake on LAN	Yes	Default	Enabled	Enabled
(0A8F)	SLP_LAN# Low on DC Power	Yes	Default	Enabled	Enabled
(0A8D)	K1 off	Yes	Default	Enabled	Enabled
(0A5E)	Wake on WLAN Enable	Yes	Default	Disabled	Disabled

Control Help String
When DCI is Enabled, it is taken as user consent to enable the DCI which allows debug over the USB3 interface. When Disabled, the host control is not enabling DCI feature.

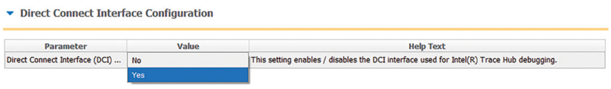
Figure 6

EFI HII identifies default values for all options, including the hidden ones. As soon as the option related to DCI is found, it can be activated for the default configuration, and DCI can be enabled by restoring the BIOS factory settings. The utility AMI BIOS Configuration Program 5.0 makes it possible to edit these settings. The edited image is programmed in SPI flash by the programmer or via the standard BIOS firmware mechanism, if privileges allow.

However, this way has one drawback: the system will not boot if Boot Guard is activated, since the utility modifies the EFI module.

Activation via Flash Descriptor Region

DCI can also be enabled by setting the PCH configuration bits either manually (they are in the Flash Descriptor Region) or using Flash Image Tool. This solution works even if Boot Guard is enabled.


Direct Connect Interface Configuration

Parameter	Value	Help Text
Direct Connect Interface (DCI) ...	No [✓ Yes]	This setting enables / disables the DCI interface used for Intel(R) Trace Hub debugging.

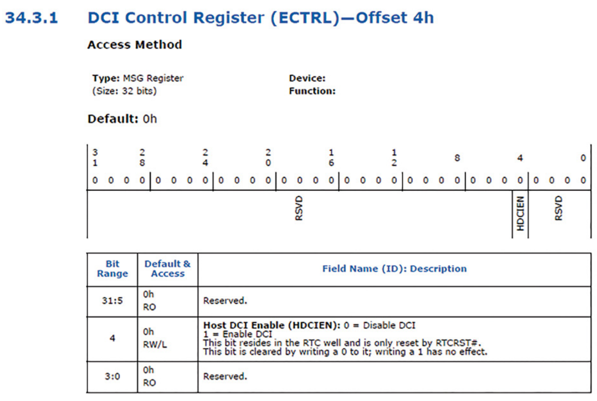
Figure 7

Activation via P2SB device

Finally, we can try to act directly via a P2SB device. A special index and register can be found in the documentation for different PCH generations, using which DCI can be enabled on the fly unless the BIOS has blocked DCI modification.

This solution exploits a vulnerability, because if the BIOS does not block writes to the ECTRL register, then (by using the ability to save the configuration between restarts and after power down) we can enable DCI once and then use the JTAG interface as a hardware backdoor to the system (and bypass the lock screen, for example).

We conducted research [12] and found that major motherboard manufacturers do not block this register. This is worrisome since this method allows enabling DCI and reprogramming the BIOS while bypassing any protection, including verification of digital signatures.



34.3.1 DCI Control Register (ECTRL)—Offset 4h

Access Method

Type: MSG Register (Size: 32 bits)

Device:

Function:

Default: 0h

Bit Range	Default & Access	Field Name (ID): Description
31:5	0h RO	Reserved.
4	0h RW/L	**Host DCI Enable (HDCIEN):** 0 = Disable DCI 1 = Enable DCI This bit resides in the RTC well and is only reset by RTCRST#. This bit is cleared by writing a 0 to it; writing a 1 has no effect.
3:0	0h RO	Reserved.

Figure 8

SUMMARY

Debugging technologies found on modern Intel processors facilitate development of UEFI modules, operating systems, and hypervisors. Security researchers use this low-level mechanism to obtain privileged access to hardware, in order to search for malware and study undocumented hardware and driver features. But as with any debugging mechanism, DCI can also be used by malicious users to gain unauthorized access to data.

To defend against such attacks, we advise that users activate Boot Guard, verify the status of the DCI enable bit, and disable debugging in the IA32_DEBUG_INTERFACE register (even if the register is disabled, DCI can still run but it is unable to interrupt and therefore access to memory and registers is impossible).

REFERENCES

1. 1149.1-1990—IEEE Standard Test Access Port and Boundary-Scan Architecture // standards.ieee.org/findstds/standard/1149.1-1990.html
2. jtag.com/en/content/about-jtag-technologies
3. resources.infosecinstitute.com/close-look-nsa-monitor-catalog-server-hacking
4. 6th Generation Intel Core Processor I/O Datasheet. Vol. 2 // www-ssl.intel.com/content/www/us/en/processors/core/6th-gen-core-pch-u-y-io-datasheet-vol-2.html
5. In-Circuit Emulation, Robert R. Collins // rcollins.org/ddj/Jul97.
6. Intel's System Management Mode, Robert R. Collins // Dr. Dobb's Journal. January 1997.
7. Guk M. Protsessory Intel ot 8086 do Pentium II [Intel Processors from the 8086 to Pentium II]. St. Petersburg: Piter, 1998. (In Russian)
8. Overview of Pentium Probe Mode, Robert R. Collins // rcollins.org/articles/probemd/ProbeMode.html.
9. Guk M. Protsessory Pentium II, Pentium Pro i prosto Pentium [Pentium II, Pentium Pro, and Plain Pentium Processors]. St. Petersburg: Piter, 1999. (In Russian)
10. www-ssl.intel.com/content/www/us/en/forms/design/registration-privileged.html
11. asset-intertech.com/products/jtag-interposers-and-arium-jtag-adapters
12. blog.ptsecurity.com/2017/01/intel-debugger-interface-open-to.html