Intel Debug Technology

Published: 07/28/2021
Last Updated: 07/28/2021

Debug is a critical capability of any system in order to get the system from manufacturing to production to deployment. As a result, Intel silicon contains several different technology capabilities used to debug its silicon. Unlike software debug features which can be excluded in production software, the nature of silicon development precludes hardware features from being removed in the final production silicon. Intel is not unique in this regard; other silicon engineering companies include these or similar types of capabilities in order to bring products to market. The term “debug” spans a wide variety of use models from internal hardware debug through debug of performance issues of software running on the silicon.

Intel recognizes that debug capabilities contained within the hardware can be an attack surface used by adversaries to gain access to secure assets and private user data. Properly controlling these debug capabilities supports the balance of protecting assets as well as providing a critical tool to Intel, Intel's customers, and Intel's developers.

This technical paper describes the general debug capabilities within Intel's silicon as well as the means designed to protect the debug capabilities. Each Intel product may differ in the specific capabilities included as well as the specific protections designed for the debug capabilities.

> Note: the terms “product” and/or “silicon” throughout this document refer specifically to an Intel product and/or Intel silicon.

Intel® Debug Protection Technology

When a person (i.e., a “debugger”) places a given product in a debug mode, additional access privileges that are not available to a person in the standard operating mode (e.g., production system owned by an end user) become available. For example, there are debug registers that expose read-write access to internal states of a system that are not visible in a production mode. A typical system normally contains several assets such as cryptographic keys, configuration data, intellectual property, and sensitive user data, that are stored in registers, memory blocks, fuses and/or otherwise embedded in the silicon. Assets might be compromised due to the privileged access these debug capabilities may provide. Therefore, proper protection measures must be implemented to restrict access to these debug capabilities.

A set of features, referred to as Intel® Debug Protection Technology throughout this paper, are used to control a product’s debug capabilities to help protect secure assets residing on Intel products and private user data being processed within the silicon at run-time. These features are designed to enable a person to perform the necessary debug securely and without compromising or putting assets at risk of exposure to unauthorized entities. The Intel® Debug Protection Technology uses access control mechanisms including authentication of the user to control access to the debug capabilities.

Controlling Debug Capabilities

Debug capabilities are designed to limit for whom or when those capabilities could be used to perform debug of a platform. Debug capabilities may be limited to platforms in certain stages of the manufacturing or development lifecycle or only during certain phases of operation (e.g., early boot). Those limitations are designed to prevent inappropriate access to information or to change behavior of the platform that could be leveraged by malicious or unauthorized users.

Debug capabilities that could be used to expose secure assets or private user data are designed to allow use only when authenticated. Authentication has multiple forms ranging from physical possession of the platform to use of cryptography to ensure the entity using the debug capabilities is permitted by the Platform Manufacturer or Intel. When authorization has not been granted, Intel takes the approach to only allow those debug capabilities that do not expose assets, to meet industry standards or expectations (e.g., IEEE JTAG Boundary Scan), or can be appropriately managed by system firmware or operating system (OS) software.

Available debug capabilities change as the product moves through its lifecycle. Depending on the product, assets may be provisioned by Intel, the Platform Manufacturer, or by the operating system and access to debug capabilities may change to protect those assets. The Protected Secure Assets section explains more about the lifecycles and the addition of assets into the product. Parties (e.g., system manufacturers and/or OS vendors) may provide additional protections beyond what is provided by Intel.

Protected Secure Assets

All Intel products will have, at a minimum, secure assets owned by Intel. These assets consist of intellectual property and stored objects such as keys. In addition, some products may also have secure assets provisioned by the Platform Manufacturer (i.e., the customer purchasing the product from Intel) such as their specific keys, or intellectual property added by the Platform Manufacturer. By design, all assets are protected and only accessible by the owner of the asset. The Intel® Debug Protection Technology needs to be aware of the ownership and only grants access when the user can be identified as the owner of the asset and grant access to only their assets. Debug capabilities that can expose assets not owned by the user are designed to be blocked or restricted.

> Note: The term "Platform Manufacturer" is used in this document to represent the overall entity that manufactures/assembles the platform, provisions secure assets into the product, and provides the platform to the end user. Sometimes these can be separate individual entities such as an Original Device Manufacturer (ODM) versus and Original Equipment Manufacturer (OEM) which may have nuances between them. However, for the purpose of this document, they are simply referred to collectively as the Platform Manufacturer.

As the product moves from silicon manufacturing to production, debug capabilities in general become more restrictive. This is due to the security (and privacy) requirements increasing as assets are provisioned and introduced into the overall system. For Intel products, there are four lifecycle states when considering the Intel® Debug Protection Technology:

1. Silicon Manufacturing
2. Platform Manufacturing
3. Production
4. Product Return

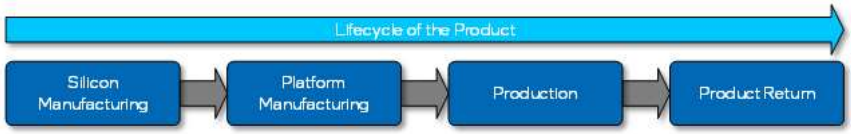

Figure 1: Lifecycle States

During the Silicon Manufacturing lifecycle state, all the debug capabilities are either available or protected by authentication via the hardware-based mechanism only accessible by Intel. Those capabilities protected by the Intel only, hardware-based authentication are those capabilities that can expose Intel intellectual property. As the product moves through the Silicon Manufacturing lifecycle state, Intel assets are provisioned and more debug capabilities become privileged (i.e., require authentication). Some capabilities, specifically for silicon hardware debug primarily used at wafer-level debug or during the early stages of silicon bring-up, are disabled as they are too low-level and not necessary at later states of the product's lifecycle.

During the Platform Manufacturing lifecycle state, the Platform Manufacturer will possibly add more assets to the product. More restrictions may be placed on debug capabilities based on what assets are added. Additional features (explained later) may also be enabled by the Platform Manufacturer. For example, the Platform Manufacturer may add unlock keys to enable more access control of the debug capabilities.

When the product has all the assets provisioned and is ready to be delivered/sold to the end user, the product is transitioned to a Production state where all the necessary protections on the debug capabilities are enabled. The Production lifecycle state is where all the necessary authentication and/or unlocking mechanisms are enabled. The Platform Manufacturer must take a specific action to explicitly cause the silicon to know that the silicon has transitioned into the Production lifecycle state. This is an important step to enable all the required protection mechanisms and should be taken once the Platform Manufacturer has provisioned all necessary assets and completed all validation. The Platform Manufacturer must set the "End of Manufacturing" bit in firmware to properly transition the product to the production lifecycle state. Protection of secure assets may be less hardened than what is possible without this bit being set.

During the Production lifecycle state all the features of the Intel® Debug Protection Technology, except for those not opted in, are functional and intended to protect the secure assets. The mechanisms for authenticating and unlocking (based on what is configured) are available and functional. The entity performing the debug will need to perform authentication and/or unlock to gain access.

The Product Return lifecycle state is like the Production lifecycle state except for the fact that the product is no longer in the field and is in either the Platform Manufacturer's possession or Intel's possession.

> Note that during all states of the product's lifecycle, some debug capabilities are potentially accessible without any hardware authentication and/or unlocking. Sometimes referred to as "public" debug capabilities, these could potentially be used by anyone with physical access to the product. Other parties beyond Intel may add additional security measures that prevent access to "public" debug capabilities implemented within the silicon.

Protecting Private User Data

Protecting private user data requires the system owner to take reasonable measures to avoid providing a path to the debug capabilities contained within the hardware, which includes (but is not limited to) maintaining physical possession, not allowing any unfamiliar external port connections such as USB, and not installing any type of debug applications on the system. If a system owner chooses to return the system to the Platform Manufacturer (or other entity), the system owner is responsible for first removing any private data from the system.

Protection Classes

Intel uses the concept of Protection Classes to define what debug capabilities are available. A Protection Class is a state of the product for which a given person, or entity, debugging can access a given set of capabilities, based on:

1. The set of debuggers (i.e., entities) permitted to debug once a specific Protection Class is established within the hardware.
2. The set of access mechanisms used to enable the use of debug capabilities in each Protection Class.

Access Mechanisms

A product may provide several different access mechanisms for the entity debugging to enable access and gain privilege to debug capabilities. The two current access mechanisms are authentication and unlock.

1. *Authentication* is the act of verifying an entity's identity. The identity of a debugger is important to help classify the entity and know what level of privilege they shall be given.
2. *Unlock* is designed to ensure that a particular entity provides a digital signature or key that is verified before the entity is granted access to a privileged set of debug capabilities based on its identity (via Authentication). Prior to a debugger unlocking, debug capabilities that can expose secure assets are blocked. This blocking of debug capabilities is mainly used to help protect assets based on keys (i.e., block debug once keys are distributed and used in the hardware). Unlocking can be controlled by either the Platform Manufacturer or Intel through separate sets of keys.

Protection Classes Definition

The Protection Classes are defined as a set of classes with both an increasing availability and privilege of debug capabilities and an increasing number of access mechanisms. Protection Class "Public" has the least number of debug capabilities available with no protection mechanisms on those debug capabilities while Protection Class "Intel" has the greatest number of debug capabilities available with many protection mechanisms on those debug capabilities. The available debug capabilities are incremental as the class increases while preserving the proper access to assets by only the respective owner. For example, Protection Class "OxM" has all the debug capabilities of Protection Class "Public" plus additional capabilities.

Protection Class "Public" (historically also known as "Green" or "Locked") is where all sets of debuggers can use the available debug capabilities given by this class. No access mechanisms are used by the debugger and the privilege level on the debug capabilities (i.e., Basic Enabling) are considered public and available to everyone. This also means that these capabilities are designed to not expose or threaten private and/or secure assets. Examples of these Protection Class "Public" capabilities may include IEEE Boundary Scan, basic telemetry, software debugging capabilities provided by the OS, and Crash Log. (Refer to later in the paper for more details.)

Protection Class "OxM" (historically also known as "Orange" or "OEM Unlocked") is where the debugger must authorize and unlock before the class is established and additional debug capabilities are enabled. Since this is limited to only using the Platform Manufacturer's authentication key, the only assumed set of debuggers is the Platform Manufacturer themselves. It is assumed that the Platform Manufacturer will not share their authentication key with any other set of debuggers.

Protection Class "Intel" (historically also known as "Red" or "Intel Unlocked") is like Protection Class "OxM" but the debugger must authenticate and unlock using Intel's authentication key. The debugger must authenticate and unlock before the class is established and additional debug capabilities are enabled. Since this is limited to only using Intel's authentication key, the only assumed set of debuggers is Intel. It is assumed that Intel will not share their authentication key with any other set of debuggers.

Debug Protection Features

The different debug capabilities enabled for a given entity performing the debug on a given product will be associated with a particular Protection Class. The result of authentication and/or unlocking (i.e., which results in entering a certain Protection Class) will determine what capabilities are available for the entity to access in order to debug the product and/or platform. Intel products will have different features of the Intel® Debug Protection Technology that are used to protect secure assets and private data as well as provide the mechanisms to perform the authentication, authorization, and unlocking.

The following subsections describe the main features of the Intel® Debug Protection Technology.

State Aggregation and Privilege Generation

The State Aggregation and Privilege Generation feature brings together the aggregated state of the product and determines the necessary protection needed. The state consists of whether the product is in the Production lifecycle state as opposed to the Silicon Manufacturing or Platform Manufacturing lifecycle states and if the entity performing the debug has authenticated and/or unlocked. This information is processed by the associated logic, and the entity's privilege is determined. The entity's privilege consists of the debug security policy, or what secure assets should be protected, and the debug capabilities enabling, or what debug capabilities are enabled based on the authentication and unlock state. This is a fundamental feature of the Intel® Debug Protection Technology and is internally known as the "DFx Security Aggregator". This feature (and the Hardware-based Policy Protection feature) is part of the Debug Security Framework, which is an architectural framework integrated into Intel's silicon.

A conceptual diagram of the logic in the product for the state aggregation and privilege generation logic is shown below.

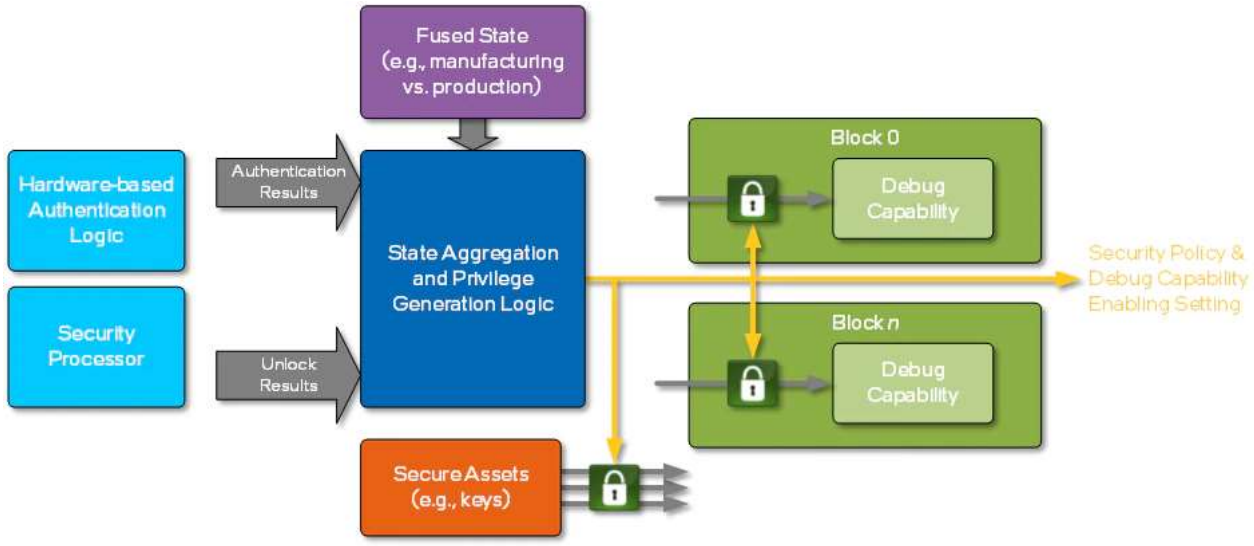

Figure 2: Conceptual Diagram of State Aggregation and Privilege Generation Logic

The State Aggregation and Privilege Generation logic generates the security policy and debug capabilities enabling setting. This hardware-based aggregator takes in the fused state and various results from logic that processes the authentication and unlocking flows. The fused state indicates where in the lifecycle the product currently resides. The other states can be from the hardware-based authentication logic, a security processor (e.g., dedicated microcontroller running firmware), or in the hardware of the State Aggregation and Privilege Generation logic itself.

The result of the state aggregation is a given privilege state of the assets and debug capabilities. As mentioned earlier, this privilege state consists of what secure assets should be protected, or the security policy, and what debug capabilities are enabled, or the debug capabilities enabling setting.

Hardware-based Policy Protection

The Hardware-based Policy Protection feature is used to help secure the protected assets and private data in the product. This is also referred to as another aspect of the Debug Security Framework since the protection is based on an architectural framework integrated into Intel silicon. Each logic block in the product receives the generated privilege state, or the security policy and/or the debug capabilities enabling setting. A conceptual diagram of this policy protection:

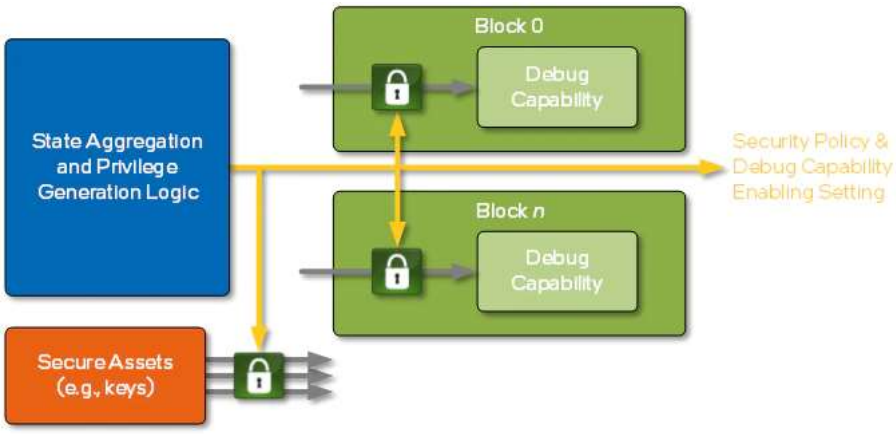

Figure 3: Conceptual Diagram of Hardware-based Policy Protection

Secure assets are protected based solely on the value of the security policy while the availability of debug capabilities is based on both the security policy and the debug capabilities enabling setting. The determination of what assets are protected and/or what debug capabilities are accessible is fixed in the silicon hardware. The value of the debug security policy and/or the debug capabilities enabling setting will dictate which setting is used. In other words, there exists a look-up table in the silicon hardware of what assets are protected and/or what debug capabilities are accessible for a given value of the security policy and/or the debug capabilities enabling setting.

Hardware-based Authentication

The Hardware-based Authentication feature is designed to allow Intel to access Intel-privileged debug capabilities to perform hardware debug of Intel's technology in the product while blocking secure assets from the entity performing the debug. This also provides the earliest access to such debug capabilities of the product. There is a limited time during the boot for which this feature can be used by an Intel entity (i.e., an entity authenticated by a per-part Intel key hash stored in the product). The entity must perform the authentication (and unlocking) through this hardware-based mechanism before any secure assets are distributed from its rest location (note: "rest location" refers to where an asset is stored).

The flow diagram below is a summary of the behavior for the hardware-based authentication feature. The entity will enter a per-part "password" into the product through the debug interface (i.e., JTAG, I3C, USB) which is used to authenticate (and unlock).

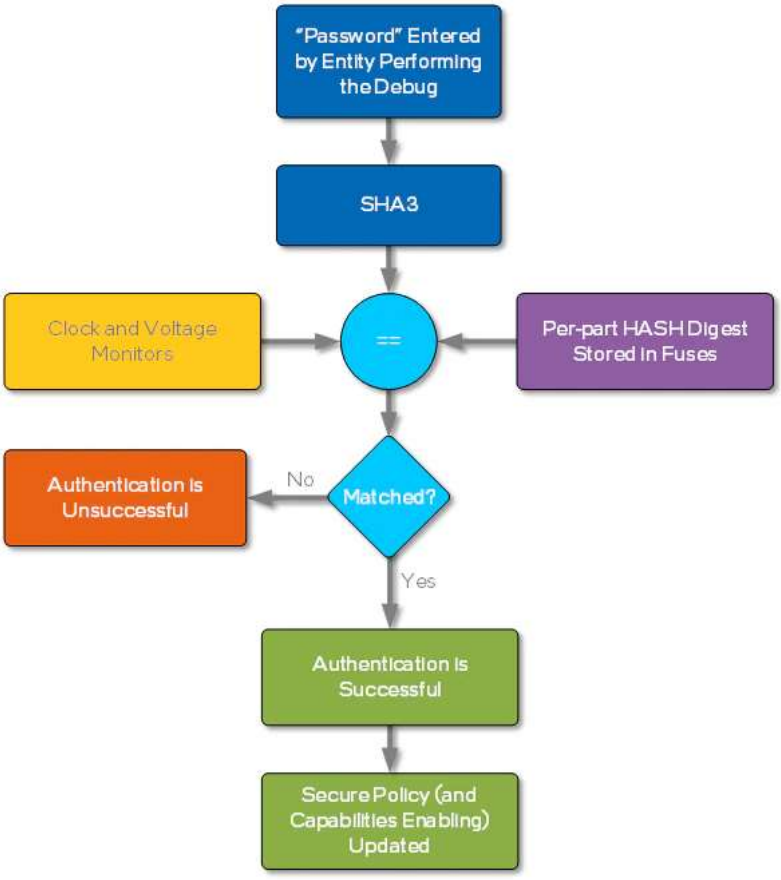

Figure 4: Hardware-based Authentication

This feature is always enabled, including in non-production products, to help protect assets and technology of Intel. This feature also has protections against brute force attacks of an adversary sending in consecutive passwords by enforcing NIST compliant password key lengths as well as limiting password attempts to once per boot. Finally, the password is unique to each Intel part and is stored in one-time programmable fuses, thereby avoiding the use of global passwords and limiting Break Once Repeat Everywhere (BORE) attacks.

Disable CPU Debug (DCD)

The Disable CPU Debug (DCD) feature, when enabled, forces the entity performing the debug to unlock before gaining access to the Probe Mode debug capabilities. Probe Mode is a set of capabilities to enable debugging of software and firmware modules specific to the X86 CPU microcode designs. Probe Mode is accessible through the various debug interfaces over the internal TAP network to the CPU cores. Probe Mode specifically allows software/application and hardware/processor execution control including breakpoints (e.g., code, data, and several architectural and microarchitectural conditions), instruction single-step, and access to state (e.g., MSR/CSR access, MMIO space) and resources (e.g., cache, system memory) used by the CPU.

By default, the Probe Mode debug capabilities are available to any entity encompassing all the Protection Classes. However, due to the privileged nature of Probe Mode and its ability to gain access to state and resources used by the CPU, the Platform Manufacturer may desire to protect the use of Probe Mode and only permit Probe Mode's use once the entity has unlocked (which also requires authentication). This means that when the Disable CPU Debug feature is enabled, Probe Mode can only be accessed in Protection Classes "OxM" and "Intel". The entity must either be the Platform Manufacturer or Intel to use Probe Mode and the associated debug capabilities.

Disable CPU Debug (DCD) is an opt-in feature available and requires the Platform Manufacturer to program the DCD fuse. By default, the Probe Mode debug capabilities are available in all Protection Classes when the DCD fuse is not set. The Platform Manufacturer must set the DCD fuse to only permit Probe Mode debug capabilities in Protection Classes "OxM" and "Intel".

Intel recommends protecting the components of the platform boot and set the DCD fuse to enable the Disable CPU Debug feature. If the product has Intel® Boot Guard enabled, then the Platform Manufacturer must program the DCD fuse to enable the Disable CPU Debug feature to protect against Boot Guard attacks via Probe Mode.

Delayed Authentication Mode

The Delayed Authentication Mode feature allows the entity performing the debug to protect secure assets before any debug is performed. The Delayed Authentication Mode causes the product to use debug versions of the assets and not the actual assets themselves, allowing the product to behave as closely to production (using debug keys/assets) before debug is initiated. The behavior of the product most likely will be different than in production but should closely resemble a production enough to allow some (more difficult) scenarios and/or situations to be debugged. Delayed Authentication Mode enabled the entity to later authenticate (and unlock) to perform debug beyond the timeline as described in the Hardware Authentication section. Changes in when the hardware may be unlocked when using Delayed Authentication Mode and when assets at rest are or are not distributed are shown in the diagram below.

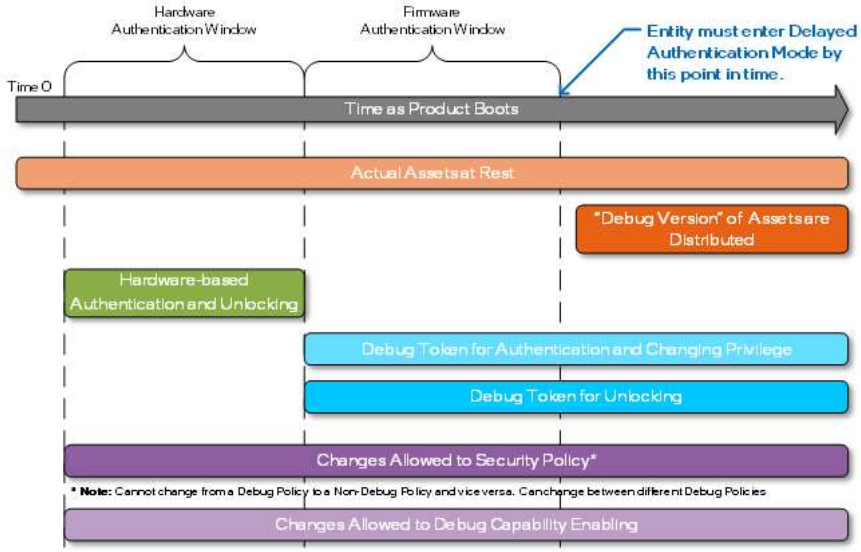

Figure 5: Delayed Authentication Mode Timeline

To later authenticate (and unlock), it is required to first enter the Delayed Authentication Mode before any secure assets are distributed. It is required that the hardware-based policy protection inform the storage location of the secure assets to not distribute assets and to instead distribute debug versions of the assets. Delayed Authentication Mode is considered a debug Security Policy and once in this Security Policy, the authenticated entity will be allowed to change to a more privileged Security Policy (based on the entity's identity). The result of being in Delayed Authentication Mode is that the application of the Debug Token may happen after the original firmware unlocking window would normally occur.

Delayed Authentication Mode is also required in scenarios where one will apply and use debug-signed patches. Debug-signed patches are intended to quickly resolve issues and may not necessarily protect the secure assets. Therefore, debug-signed patches are designed to only be used when true assets are not distributed.

Once the Delayed Authentication Mode is entered, then the State Aggregation and Privilege Logic is designed to only (based on the entity's identity) move between different privileged debug Security Policies. Delayed Authentication Mode is designed to prevent an entity from going back to a non-debug policy (where the actual assets will be distributed from the assets' storage location) until a full power-cycle reset has occurred and Delayed Authentication Mode has not been entered following that full power-cycle reset.

Debug Token

The Debug Token feature allows the entity performing the debug to inject a digitally signed data blob into firmware, referred to as the Debug Token, to authenticate and/or unlock the product and allow for more fine grain control of the debug features. The Debug Token is a firmware blob that is signed by either the Platform Manufacturer (in Protection Class "OxM") or Intel (in Protection Class "Intel"). The signing key is provisioned by the Platform Manufacturer and/or Intel at the time of manufacturing (i.e., silicon manufacturing for Intel and platform manufacturing for the Platform Manufacturer).

Debug Tokens are specific to a given product instance (i.e., per part) and can be time bound.

Debug Capabilities Within Intel Hardware

Most of the capabilities used to debug silicon are intended for Intel internal use, primarily while in the silicon prototyping phase, and are assigned to the "Intel" Protection Class. However, since issues may arise after a product has gone into Production lifecycle state, these capabilities may be used in specific circumstances to help root cause issues in a system in the Production lifecycle state, assuming proper authentication and unlock described above, and some of these capabilities may be classified as "OxM" or "Public" Protection Classes.

Debug capabilities can broadly be classified into two basic categories: items that aid in the control or observation of the silicon, and items that are used for tracing and/or triggering of events within the silicon. Debug capabilities are typically accessed through the Debug Port of the silicon.

Control and Observation

The following capabilities exist within Intel silicon:

1. Hardware-based Execution Run Control. This capability provides basic execution control of cores.
    1. For Intel Architecture (IA) based cores, this includes halt, go, step, break, accessing architectural resources (specifically registers defined in the Intel Software Developers Manual), software source-level debug, instruction injection, and microarchitectural state action (typically registers referred to in an External Design Specification, such as Model Specific Register (MSR).) This functionality is referred to as "Probe Mode". This functionality is available in all Protection Classes, though a subset of functionality is available only in Protection Class "Intel" state, and the capability is not available when the DCD bit is set in the Protection Class "Public" state.
    2. For non-Intel Architecture embedded cores provided by other IP companies similar hooks exist that allow for execution control. Those hooks are core specific as determined by the core vendors. Those capabilities may be available in all Protection Classes or may be restricted to Production Class "Intel" or Protection Class "OxM".
2. TAP Access to Registers. This capability allows an external Debug and Test System (DTS) to perform both read and write access configuration and status register (CSR) contents within the Target System (TS) via the IEEE 1149.1 Test Access Port (TAP) using a hardware probe. These registers are typically microarchitectural state and are silicon generation specific, and the specific Protection Class applied to each register is product specific.
3. Scandump and Array Freeze & Dump. This capability provides read access to almost all state elements within the design, regardless of how the state elements are implemented. This capability is also used for High Volume Manufacturing (HVM) to find silicon defects. This capability is only available in the Protection Class "Intel" state.
4. Silicon Reset Control. The silicon contains hooks to "stall" or "break" execution at various parts of the reset sequence. These are typically implemented with a combination of external pins and internal configuration registers. Once stalled, the debugger may examine or change state visible in configuration registers. While in the Protection Class "Intel" state , all hooks are available to the user.
5. Firmware Patching. Firmware patching is a capability that allows for modification of internal firmware (e.g., microcode, power management firmware code, etc.). Firmware patches come in two basic flavors: production patches, which are signed and loaded into a production system (i.e., a product in the Production lifecycle state), and debug patches, which require the silicon to be in a Protection Class "Intel" state.
6. Fuse Overrides. In a Protection Class "Intel" state, the value of some fuses within the hardware may have their values temporarily overridden.
7. Intel® Crash Log Technology is a capability that automatically collects and stores data when a failure occurs. Crash Log contains only public information and thus available in all Protection Classes. Some silicon may have additional features that collect more data to provide to Platform Manufacturers once specifically authorized (i.e., Production Class "OxM"). There is no "Intel-only" version of Crash Log today, because capability #2 can be used to collect any information once the silicon is in the Protection Class "Intel" state.
8. Debug Read & Debug Write macro-instructions. These instructions are used to read and write resources within the product in the Protection Class "Intel" state.
9. Memory (e.g., DRAM, MMIO) read and/or write. When in Production Class "Intel" state, hardware allows access to most memory within the system.
10. PSMI. PSMI is the term for a specific technology used to capture signals within the chip and replay those signals on a hardware emulator or simulator of the silicon. This capability is available only in the Protection Class "Intel" state.
11. IEEE-defined boundary scan may be available in all Protection Classes.

Note that the capabilities described above are typically only accessed by one or more "debug ports" in the silicon. Intel debug ports generally operate on the principle of allowing the connection and using the internal protection functions with respect to Protection Classes described above. Examples of Intel debug ports for control & observation include the IEEE 1149.1 TAP, Intel® Direct Connect Interface (DCI, also known as Debug via Universal Serial Bus (USB) and/or Debug Class (DbC)), an Extensible Debug Port (XDP), and in the future Debug over I3C. Some Intel products that contain an Arm core may contain the Arm Serial Wire Debug (SWD) port.

Trace and Trigger

Trace and trigger are a more advanced form of observation. Trace is used to observe events within the silicon over time. Trigger is used to detect and respond to events that occur within the silicon. Some additional features within Intel silicon, such as performance monitoring and telemetry, are examples of trace and trigger that are beyond the debug domain and are not included in this document.

In general, the protection mechanisms for trace and trigger are applied at the source. The amount of visibility is thus limited by the protection state of the silicon and how the protection state is treated by the source of the information. In Protection Class "Intel" state, 100% of trace signals and trigger events and responses are available to the debugger with no restrictions.

1. VISA (Visualization of Internal Signal Architecture) is the internal name for internal hardware signal-level trace. Various functional signals are collected and may be observed while the silicon is operating. The location and number of signals collected, as well as the Protection Class of the individual signals, varies by function and product.
2. Debug Trace Fabric (DTF) is an internal transport of trace, but the term is used to describe various "hardware-based trace sources" that are not strictly VISA tracing described above. These types of sources may include internal fabric tracing, memory command tracing, or external Input/Output (I/O) tracing. The specific Protection Class of the individual sources varies by function and product.

3. Internal logic can generate microarchitectural “events” that occur within the hardware (e.g., detection of an unexpected error condition) and/or implement microarchitectural “responses” within the hardware (e.g., implement a temporary logic stall). In addition, trigger events may be sourced out of tracing signals and responses may be used to control the tracing system (e.g., start and/or stop trace). The specific Protection Class of the individual events and responses varies by function and product.
4. Firmware engines (i.e., code running on an embedded microcontroller) may produce trace messages (sometimes referred to as “debug print”) that provide visibility into the firmware engine's progress. Some firmware engines contain messages that can be collected in a Protection Class “Public” state; in the Protection Class "Intel” state all messages can be visible.
5. Intel Architecture (IA) Microcode Tracing is implemented via capability is known as Architectural Event Trace (AET). Most messages are available in a Production Class “Public” state after “Probe Mode” is entered.
6. Software Tracing is provided by Intel Architecture “Intel® Processor Trace”. This is provided via hardware within the IA core to capture IA instruction execution and other interesting events (please refer to the Software Development Manual chapter about Intel® Processor Trace for more details.) In addition, Intel Processor Trace may allow application software to create its own, debug-print style messages. Finally, IA software may write messages directly to the Intel Trace Hub's “Software Trace Hub”. These messages are typically available in all Protection Classes.
7. Viewpins. Viewpins is a hardware signal tracing technology used for very high-speed digital and analog signals using dedicated pins on the silicon. This capability is generally available only in Protection Class “Intel” state and is used for high speed analog debug (for example, with high speed I/Os).

The capabilities above generally flow into hardware that is known as “Intel® Trace Hub”. Other hardware elements exist within the silicon as well. The silicon may contain external trace ports, such as the MIPI Parallel Trace Interface (PTI) port or the VISA port, to observe trace data in “real time”.

Debug Ports and Interfaces

The typical setup for hardware debug is to connect a Debug and Test System (DTS) to a Target System (TS, which may be referred to as System Under Test). The debug tool software runs on the DTS and is connected through a physical connection to the TS.

Each product contains one or more interfaces (i.e., “Debug Connections”) that are used for debug of that product. The debug interfaces are grouped into four capabilities: control, observation, trace, and trigger. Debug interfaces are either shared with functional interfaces (i.e., not dedicated for debug usage) or dedicated interfaces for debug usage (which are generally connected to a DTS). The table below summarizes the potential product interfaces implemented in current Intel products. Not all products have all these interfaces.

Table 1: Debug Interface Summary

Interface	Group	Open or Closed Chassis Usage?	Dedicated Debug or Shared Interface?	Description/Notes
JTAG	Control	Open	Shared	Primary and simplest bare-metal connection used for both debug and test/HVM. Connected to the MIPI-60 Debug Connector on platform (may be shared with other components). Note that each silicon instance typically has a dedicated JTAG interface; sharing sometimes occurs at the platform level between different components.
Probe Mode (PREQ, PRDY pins)	Trigger	Open	Dedicated	Probe Mode Request and Response (IA core run control); pins used to coordinate between sockets.
DCI OOB (a.k.a., BSSB)	Control, Trace, Trigger	Closed	Shared or Dedicated	A 2-wire or 4-wire out-of-band, point-to-point, bare-metal interface for debug communication. The 4-wire variant is multiplexed over the superspeed (SS) pins on the USB 3.x port or USB Type-C receptacle. The port is USB 3.x and the receptacle is Type-C. The 2-wire variant is typically multiplexed over the sideband (SBU) pins of a USB Type-C receptacle. Sometimes, they are used over a dedicated receptacle.
I2C	Control	Open	Shared	Debug tools can directly drive a functional I2C interface/protocol. Multiple different usages/protocols.
USB	Control, Trace, Trigger	Closed	Shared	Intel Direct Connect Interface (DCI). High-speed debug communication and trace interface using the USB 2.0 interface and the Debug Class (DbC). DCI uses the Intel proprietary protocol and has endpoints to support DFx (i.e., JTAG and run control), Trace (streaming from North Peak), and DMA to/from system memory. Operates as a device (UFP on the TS).
USB	Control, Trace, Trigger	Closed	Shared	Intel Direct Connect Interface (DCI). High-speed debug communication and trace interface using the USB 3.x interface and the Debug Class (DbC). It uses the Intel-proprietary ExI Protocol and has endpoints to support DFx (i.e., JTAG and run control), Kernel Mode Debug (KMD), Trace, and Direct Memory Access (DMA) to/from system memory. Operates as a host (DFP) on the TS. Not all products support all given endpoints.

Interface	Group	Open or Closed Chassis Usage?	Dedicated Debug or Shared Interface?	Description/Notes
PTI	Trace	Open	Shared or Dedicated	High-bandwidth, parallel trace interface. Uses the MIPI System Trace Protocol (STP). Typically connected to the MIPI-60 Debug Connector on platform (may be shared with other interfaces, typically VISA
Trigger Pins	Trigger	Open	Dedicated	Cross-trigger interface between platform components and connection to MIPI-60 Debug Connector.
UART	Control	Open	Shared or Dedicated	Legacy capability used by software or firmware engines to send messages instead of sending messages to Intel Trace Hub.
Viewpins	Observation	Open	Dedicated	High-speed digital and analog pin used for PHY and hard IP debug.
VISA	Observation	Open	Shared or Dedicated	Signal level trace, supports clocked and asynchronous operation, captured on external Logic Analyzer. May be shared with PTI.
HTI	Trace	Both	Shared	Out-of-band high-speed trace interface utilizing DP supported physical interfaces (PHYs). Used in a subset of products, and interface is typically shared with USB Type-C and/or Display Port.

Additional References

Intel is not unique in the debug capabilities within the silicon. Other example public industry whitepapers that describe debug capabilities include:

1. MIPI Alliance Architecture Overview for Debug
2. IEEE System Validation and Debug Technology Committee (SVDTC) Debug Infrastructure for System-on-chips
3. IEEE System Validation and Debug Technology Committee whitepaper on scandump

Public Intel References:

1. Intel Platform Analysis Library
2. Intel® Crash Log Technology

Appendix: Acronyms

This document uses the following acronyms.

Table 2: Acronyms

Acronym	Definition
AET	Architectural Event Trace
CSR	Control and Status Register
CPU	Central Processing Unit
DbC	Debug Class
DCD	Disable CPU Debug
DTF	Debug Trace Fabric
DCI	Direct Connect Interface
DFP	Downstream Facing Port
DFx	Design for Debug, Test, Manufacturing, and/or Validation

Acronym	Definition
DMA	Direct Memory Access
DRAM	Dynamic Random-Access Memory
DTS	Debug and Test System
EOM	End of Manufacturing
EXI	Embedded DFx Interface
IA	Intel Architecture
I2C	Inter-Integrated Circuit
I3C	MIPI I3C bus (https://en.wikipedia.org/wiki/I3C_(bus)
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
HTI	High-speed Trace Interface
HVM	High Volume Manufacturing
JTAG	Joint Test Action Group
KMD	Kernel Mode Debug
MIPI	Mobile Industry Processor Interface
MMIO	Memory-Mapped Input/Output
ODM	Original Device Manufacturer
OEM	Original Equipment Manufacturer
OxM	Original Device and/or Equipment Manufacturer
OOB	Out of Band
OS	Operating System
PHY	External Physical Interface
PREQ	Probe Mode Request Pin
PRDY	Probe Mode Ready Pin
PSMI	Capture & Replay Technology
PTI	Parallel Trace Interface
STP	System Trace Protocol

Acronym	Definition
SWD	Serial Wire Debug
SBU	Sideband USB pins
SS	Superspeed
TAP	Test Access Port
TS	Target System
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
VISA	Visualization of Internal Signals Architecture
XDP	Extensible Debug Port
X86	Intel x86 Architecture

Product and Performance Information

[1] Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex .