230329.TrustZone

Created: 2023-04-10 19:57:37

Questions

- The ariticle
  - https://genode.org/documentation/articles/trustzone
  - Please summarize the following articles in short.
  - Please compare TrustZone and TPM in a table.
  - What is bootstrap code?
  - What are two lines in the section “Starting point” in the article?
- Basic Concepts
  - Is TrustZone a software or hardware?
  - 2023/4/10: Which types of ARM CPU have TrustZone?
  - 2023/4/10: Does ARM Cortex-R series have TrustZone?
- TrustZone & HSM
  - What is difference between TrustZone and HSM? Please also summarize in a table.
  - What are relationships between TrustZone and HSM?
  - Can TrustZone totally replace HSM?
- Normal World & Secure World
  - Can Normal world and Secure World run at the same time in ARM?
  - But can Normal world and Secure World run at the same time in multi processors of ARM?
- Boot
  - Does boot loader only run in secure world, not in normal world?
  - How to swtich secure world to normal world when booting?
  - What is ATF?
  - 2023/4/10: What is FF-A?
  - 2023/4/10: Does TF-A rely on FF-A?
  - 2023/4/10: Does TF-A rely on TrustZone?
- Memory
  - How to define memory size of secure world?
  - Can memory size of secure world be changed?
  - What is MMU?
  - What is TLB?
- CPU
  - What is SMP?
  - What are difference between primary and seconday CPU cores?
  - Please summarize the two types of CPU cores as a table.
  - What are SCP and AP?
  - Do SCP and AP in ARM share memory?
  - What is differenct between Armv8-A and Armv8-M? Please summarize as a table.
- SM & SPM & SPD
  - What is Secure Monitor?
  - What is SMC?
  - Can secure world request a service in normal world via SMC?
  - What is SPM?
  - What is SPD?
  - Can SPD handle the request from the secure world?
  - What are relationships of SPD, SPM and SMC?

    - Why are SPD and SPM mutually exclusive?
    - Is SPD an optional component?
    - Can I use SPD instead of SM?
    - Does TrustZone require SPM?
- Secure Partition
    - What is SP?
    - Can different SPs communicate together?
    - How to create a SP and when?
    - Can I create a SP in run-time?
    - What is TA?
    - How do TAs and SPs communicate each other?
    - Can a SP host many TAs?
    - Can a TA communicate different SP?
    - Is normal world a secure parition?
    - Is secure world a secure partition?
- Privilege Levels
    - What are EL0, EL1, EL2, S-EL0, S-EL1, S-EL2, S-EL3?
    - 2023/4/10: What is exception level?
    - 2023/4/10: How does a software component know which EL it owns?
    - 2023/4/10: Can a software component change the EL?
    - 2023/4/10: Are ELs of TrustZone?

Q & A

The ariticle

Question: https://genode.org/documentation/articles/trustzone

Answer:

The link you've shared (https://genode.org/documentation/articles/trustzone) directs to an article on Genode's website that discusses TrustZone technology. Genode is an open-source operating system framework focused on creating a secure and scalable environment for running multiple instances of software. TrustZone is a security technology developed by ARM, which is widely used in modern processors.

TrustZone technology aims to provide a secure execution environment for trusted applications, protecting them from potential security threats posed by untrusted applications. It achieves this by dividing the system into two distinct worlds: the "Secure World" and the "Normal World." These two worlds run concurrently on the same hardware, but they are isolated from each other, ensuring that sensitive data and operations remain protected.

The "Secure World" is designed to handle security-sensitive tasks, while the "Normal World" handles general-purpose tasks. To switch between the two worlds, ARM provides a secure monitor call (SMC) instruction that manages the transitions.

Genode leverages TrustZone technology to enhance the security of its operating system framework. By using TrustZone, Genode can isolate critical parts of the system and applications from the rest of the system, reducing the risk of sensitive data leaks or unauthorized access.

If you have any specific questions about TrustZone or Genode, feel free to ask, and I'll do my best to help.

Question: Please summarize the following articles in short.

Answer:

It appears you forgot to provide the articles that you'd like me to summarize. Please provide the links or titles of the articles, and I'll be happy to provide summaries for you.

Question: Please compare TrustZone and TPM in a table.

Answer:

Feature	TrustZone	TPM
Purpose	Hardware-based security solution	Hardware-based security solution
Security functions	Freely programmable	Fixed-function
Execution environment	Secure world on ARM-based SoC	Standalone chip
Isolation	Two worlds: Secure and Non-secure	Separate from CPU
Complexity	Varies, from simple libraries to full OS	Limited, designed for specific functions
Interaction with peripherals	NS bit determines peripheral access control	Communicates via standard bus protocols
Cryptographic capabilities	Software implemented, based on user needs	Built-in cryptographic functions
Secure storage	Depends on SoC implementation	Built-in non-volatile storage
Secure boot	SoC-dependent, can be implemented	Provides support for secure boot process
Virtualization support	Limited, not designed for virtualization	Not applicable
Flexibility	More versatile, programmable functions	Less versatile, fixed-function
Performance impact	Depends on implementation complexity	Minimal, as it operates independently

Please note that this comparison is a high-level overview of TrustZone and TPM. The specific features and capabilities may vary depending on the implementation and the SoC used for TrustZone or the TPM version and vendor.

Question: What is bootstrap code?

Answer:

Bootstrap code, also known as boot code, bootloader, or bootstrapping, is a small program responsible for initializing a computer's hardware components and loading the operating system (OS) into memory. When a computer is powered on or restarted, the bootstrap code is the first code to execute, typically stored in a fixed location, such as a Read-Only Memory (ROM) chip, or firmware like Basic Input/Output System (BIOS) or Unified Extensible Firmware Interface (UEFI).

The primary function of the bootstrap code is to perform hardware checks, initialize essential hardware components (like CPU, memory, and peripherals), and locate the operating system's kernel. Once the kernel is found, the bootstrap code loads it into memory and transfers control to the kernel, enabling the OS to start and manage system resources.

In summary, bootstrap code is responsible for the initial steps in the boot process, laying the groundwork for the operating system to take over and provide a functional environment for the user.

Question: What are two lines in the section "Starting point" in the article?

Answer:

In the context of the "Starting point" section of the article, two lines refer to two distinct approaches to exploring TrustZone technology:

1. Line 1: Utilize TrustZone as a virtualization solution, exploring its potential to host multiple virtual machines or support multiple operating systems in the secure and non-secure worlds. This line of experimentation seeks to understand TrustZone's capabilities as a virtualization mechanism.
2. Line 2: Use TrustZone as an alternative to Trusted Platform Modules (TPM), focusing on its ability to implement security functions and protect sensitive information within the secure world. This line of experimentation aims to investigate the

versatility and programmability of TrustZone as a security mechanism compared to fixed-function TPMs.

Basic Concepts

Question: Is TrustZone a software or hardware?

Answer:

TrustZone is a hardware-based security technology implemented in ARM processors. It provides a security extension that enables the creation of a secure environment alongside the normal environment, isolating the execution of sensitive code and data from potential threats in the normal environment. TrustZone itself is not a software; however, it requires software support to fully utilize its features, such as secure operating systems, trusted applications, and secure bootloaders.

Question: 2023/4/10: Which types of ARM CPU have TrustZone?

Answer:

TrustZone technology is available in various ARM CPU cores, particularly in the ARM Cortex-A and ARM Cortex-M series.

1. ARM Cortex-A series: TrustZone is supported in many Cortex-A cores, including Cortex-A5, Cortex-A7, Cortex-A8, Cortex-A9, Cortex-A12, Cortex-A15, Cortex-A17, Cortex-A32, Cortex-A35, Cortex-A53, Cortex-A55, Cortex-A57, Cortex-A72, Cortex-A73, Cortex-A75, Cortex-A76, and Cortex-A77.
2. ARM Cortex-M series: The ARMv8-M architecture introduces TrustZone technology for Cortex-M processors. Examples of Cortex-M cores with TrustZone support include Cortex-M23 and Cortex-M33.

Keep in mind that ARM continues to develop new processor cores, and the list above might not be exhaustive. Always refer to the official ARM documentation for the most up-to-date information on specific cores and their features.

Question: 2023/4/10: Does ARM Cortex-R series have TrustZone?

Answer:

Yes, some ARM Cortex-R series processors also have TrustZone support. TrustZone for Cortex-R processors is designed for real-time systems that require high performance, safety, and security. One example of a Cortex-R processor with TrustZone support is the Cortex-R52.

As with other series, it's essential to consult the official ARM documentation or the specific processor datasheet to verify TrustZone support for each Cortex-R processor.

TrustZone & HSM

Question: What is difference between TrustZone and HSM? Please also summarize in a table.

Answer:

TrustZone and HSM (Hardware Security Module) are both technologies that provide security features, but they are designed for different purposes and have distinct characteristics. Here is a comparison of the two:

TrustZone:

- A security extension of ARM architecture.
- Creates two separate worlds: Secure and Normal worlds.
- Provides isolation and protection for sensitive operations and data.
- Can be implemented on a general-purpose CPU.
- Usually lower cost than HSM.
- Lower performance for cryptographic operations compared to HSM.
- Suitable for a wide range of applications, including mobile devices, IoT, and embedded systems.

HSM:

- A dedicated hardware device for secure cryptographic operations and key management.
- Designed specifically for high-level security.
- Offers higher performance for cryptographic operations compared to TrustZone.
- Physically tamper-resistant.
- FIPS 140-2 and other security certifications.
- Higher cost than TrustZone.
- Suitable for high-security applications like banking, government, and critical infrastructure.

Here is a summary table:

Feature	TrustZone	HSM
Type	ARM architecture extension	Dedicated hardware device
Purpose	Isolation and protection	Cryptographic operations & key management
Implementation	General-purpose CPU	Specialized hardware
Cost	Lower	Higher
Performance	Lower for crypto operations	Higher for crypto operations
Tamper-resistance	Limited	High (physically tamper-resistant)
Security certifications	Varies	FIPS 140-2 and others
Use cases	Mobile, IoT, embedded systems	Banking, government, critical infrastructure

Please note that the comparison above is a generalization, and specific implementations of TrustZone and HSMs may have different features and performance characteristics.

Question: What are relationships between TrustZone and HSM?

Answer:

TrustZone and HSM (Hardware Security Module) can be used together or independently to provide security features in a system. They are not mutually exclusive and can complement each other in various ways. Here are some relationships between TrustZone and HSM:

1. Complementary Technologies: TrustZone can be used to create a secure environment for running sensitive software, while HSM can be used for secure cryptographic operations and key management. Both technologies can work together in a system to provide a higher level of security.
2. TrustZone as a lightweight alternative: In some cases, TrustZone can be used as a lightweight alternative to HSM for less critical applications or where the cost and complexity of HSM are not justifiable. TrustZone provides a good balance between security and cost in such cases.
3. HSM inside TrustZone: HSM can be used within the secure world of TrustZone to provide high-performance cryptographic operations and secure key storage. In this setup, TrustZone isolates the HSM and its operations from the normal world, while HSM performs its dedicated security functions.
4. Communication: TrustZone and HSM can be used together to establish secure communication channels between different components of a system. TrustZone can ensure the confidentiality and integrity of the data exchanged between the secure and normal worlds, while HSM can be used for cryptographic operations like encryption, decryption, and digital signatures.

In summary, TrustZone and HSM can be seen as complementary technologies that can work together to provide a comprehensive security solution. TrustZone provides isolation and protection for sensitive operations and data, while HSM delivers high-performance cryptographic operations and secure key management. Depending on the application and security requirements, they can be used together or independently.

Question: Can TrustZone totally replace HSM?

Answer:

TrustZone cannot entirely replace an HSM because they serve different purposes and have different strengths and limitations. While TrustZone can provide a secure execution environment for sensitive applications, it does not offer the same level of security, performance, and tamper-resistance as a dedicated HSM.

Here are some reasons why TrustZone cannot fully replace an HSM:

1. Hardware-level Security: HSMs are designed with specialized hardware to provide a high level of tamper-resistance, secure key storage, and cryptographic performance. TrustZone, on the other hand, relies on the processor's hardware features and provides software-based isolation. It is not designed to offer the same level of hardware-level security as an HSM.
2. Cryptographic Performance: HSMs are optimized for cryptographic operations and can provide better performance for encryption, decryption, and other cryptographic functions compared to TrustZone, which relies on the processor's general-purpose hardware.
3. Secure Key Storage: HSMs are designed to store cryptographic keys securely and protect them from unauthorized access, even in the presence of physical attacks. TrustZone can provide some level of secure key storage, but it may not be as robust as the key storage in an HSM.
4. Certification: HSMs often come with certifications, such as FIPS 140-2 or Common Criteria, that guarantee a certain level of security and assurance. TrustZone, being a part of a general-purpose processor, may not have the same certifications, making it less suitable for highly regulated industries or applications with strict security requirements.

In summary, while TrustZone can provide a secure environment for sensitive applications, it cannot fully replace an HSM in terms of security, performance, and tamper-resistance. Both technologies have their use cases, and depending on the security requirements, one or both can be employed in a system.

Normal World & Secure World

Question: Can Normal world and Secure World run at the same time in ARM?

Answer:

In ARM TrustZone, the Normal world and Secure world cannot run concurrently. The processor can only execute code in either the Normal world or the Secure world at any given time. The two worlds are isolated from each other, and the processor switches between them based on events such as interrupts or requests for secure services. TrustZone uses a secure monitor mode to facilitate the transition between the two worlds, ensuring that the secure and non-secure states remain separate and protected.

Question: But can Normal world and Secure World run at the same time in multi processors of ARM?

Answer:

Yes, in a multi-processor ARM system, it is possible for the Normal world and Secure world to run concurrently on separate cores. Each core in the system can be executing in either the Normal world or the Secure world independently. This allows for parallel processing of tasks in both worlds. However, it is important to note that the isolation between the two worlds is still maintained, and each core must follow the same TrustZone mechanisms to transition between the worlds when required.

Boot

Question: Does boot loader only run in secure world, not in normal world?

Answer:

Bootloaders can run in both the Secure World and the Normal World, depending on the specific use case and platform. In a secure boot process, the initial bootloader stages run in the Secure World to ensure the integrity and authenticity of the

system’s firmware and subsequent software components.

Once the initial validation is complete, the bootloader may hand over control to the Normal World, allowing the operating system and applications to boot up and run.

However, in some cases, a bootloader may run entirely in the Normal World if the platform doesn’t require secure boot or if the secure boot process is handled differently. Ultimately, whether the bootloader runs in the Secure World or the Normal World depends on the specific hardware, firmware, and security requirements of the system.

Question: How to swtich secure world to normal world when booting?

Answer:

When booting a system with TrustZone technology, the processor starts in the Secure World by default. To switch from the Secure World to the Normal World during the boot process, you can follow these general steps:

1. Initialize the Secure World: In the Secure World, the secure bootloader initializes the necessary hardware components, sets up the secure environment, and configures the memory and peripherals access permissions.
2. Validate and load the Normal World software: The secure bootloader validates the Normal World software, such as the operating system or bootloader, using cryptographic techniques like digital signatures. Once validation is successful, it loads the software into the memory designated for the Normal World.
3. Configure the processor: Before switching to the Normal World, the secure bootloader sets up the processor’s state, registers, and exception levels. It also configures the system registers, such as the Secure Configuration Register (SCR) and the Monitor Vector Base Address Register (MVBAR), to define the behavior of the system when switching between the Secure World and the Normal World.
4. Switch to the Normal World: The secure bootloader performs a Secure Monitor Call (SMC) or an Exception Return (ERET) to switch the processor from the Secure World to the Normal World. This transition is facilitated by the Secure Monitor, which is responsible for handling the state switch between the two worlds.

After these steps, the processor enters the Normal World, and the Normal World software, such as the operating system or bootloader, starts executing.

Please note that these steps are a general outline and may vary depending on the specific platform, hardware, and firmware implementations.

Question: What is ATF?

Answer:

ATF stands for ARM Trusted Firmware, which is an open-source project that provides a reference implementation of secure software for ARMv8-A-based platforms. The ARM Trusted Firmware provides a set of software components that are required to implement security features on ARMv8-A processors, such as those that support ARM TrustZone technology.

The main components of ARM Trusted Firmware include:

1. BL1 (Boot Loader 1): The initial boot loader responsible for setting up a secure execution environment and loading the next stage of the boot process, usually BL2 or BL31.
2. BL2 (Boot Loader 2): Responsible for loading and authenticating the remaining firmware images, such as BL31, BL32, and BL33. It typically performs additional platform-specific initialization and sets up memory partitioning.
3. BL31 (EL3 Runtime Software): The runtime firmware that runs at Exception Level 3 (EL3) and provides services for managing the system power state, system resets, and the handling of SMC (Secure Monitor Call) between the secure and non-secure worlds.
4. BL32 (Secure-EL1 Payload): Optional firmware that provides services to the secure world, such as a Secure Partition Manager (SPM) or a Trusted Execution Environment (TEE) like OP-TEE.
5. BL33 (Non-secure world bootloader): The bootloader for the non-secure world, typically a bootloader like U-Boot or an operating system like Linux.

The ARM Trusted Firmware provides a foundation for developing secure software on ARM-based platforms and serves as a reference implementation for platform vendors and developers to build upon or customize for their specific needs.

Question: 2023/4/10: What is FF-A?

Answer:

FF-A, or Firmware Framework for Armv8-A, is a specification that defines a standard interface for Armv8-A-based systems to support secure firmware components, such as Secure Partitions and Secure Partition Managers. This framework aims to promote interoperability between different firmware components from different vendors and provides a consistent method for handling security, communication, and resource management.

FF-A enables the development of secure, modular firmware components that can be used across multiple platforms and SoCs, simplifying firmware development and reducing the need for custom solutions. It supports various use cases, including secure boot, trusted execution environments, and virtualization.

Question: 2023/4/10: Does TF-A rely on FF-A?

Answer:

Yes, Arm Trusted Firmware-A (TF-A) relies on and implements the Firmware Framework for Armv8-A (FF-A) specification. TF-A provides a reference implementation of secure world software for Armv8-A platforms, including support for Secure Partition Manager (SPM) and Secure Partitions (SPs) as defined by the FF-A specification.

By relying on FF-A, TF-A ensures that it follows a standardized interface for secure firmware components, enabling compatibility and interoperability with various platforms and SoCs. This helps simplify firmware development and promote the use of secure, modular firmware components across different Armv8-A-based systems.

Question: 2023/4/10: Does TF-A rely on TrustZone?

Answer:

Yes, Arm Trusted Firmware-A (TF-A) relies on and utilizes the TrustZone technology for providing a secure execution environment on Armv8-A platforms. TrustZone is a hardware-based security feature that creates a secure and isolated environment (Secure World) for running sensitive applications and handling secure data.

TF-A is a reference implementation of secure world software that provides various security services, such as secure boot, runtime firmware services, and a Secure Partition Manager (SPM) for managing Secure Partitions (SPs). By leveraging TrustZone, TF-A ensures that these security services and sensitive applications are protected from the less secure, normal world applications and potential vulnerabilities.

Memory

Question: How to define memory size of secure world?

Answer:

Defining the memory size for the secure world in an ARM TrustZone-enabled system is highly dependent on the specific System-on-Chip (SoC) being used. Each SoC has different characteristics and mechanisms for allocating and partitioning memory resources between the secure and non-secure worlds.

Typically, the memory size for the secure world is defined during the system's initialization or configuration phase. The process involves configuring TrustZone-specific hardware components such as TrustZone Protection Controller (TZPC) and TrustZone Address Space Controller (TZASC) to set access permissions and memory partitions.

To define the memory size of the secure world, follow these general steps:

1. Determine the available memory resources: Consult your SoC's technical reference manual or datasheet to identify the available memory resources and their respective addresses.

2. Configure TZPC and TZASC: Based on the desired memory size for the secure world, configure the TZPC and TZASC registers to partition the memory accordingly. This may involve setting up specific regions and access permissions for secure and non-secure worlds. Some SoCs might provide alternative or additional mechanisms for memory partitioning.
3. Verify and update your secure world software: Ensure your secure world software is aware of the allocated memory size and adapts its memory management accordingly.

Please note that these steps are generalized, and you should refer to the documentation of your specific SoC for exact procedures and register configurations.

Question: Can memory size of secure world be changed?

Answer:

In most ARM TrustZone-enabled systems, the memory size of the secure world is typically defined during the system's initialization or configuration phase. However, whether the memory size of the secure world can be changed at runtime depends on the specific System-on-Chip (SoC) being used and its support for dynamic memory partitioning.

Some SoCs may allow for dynamic reconfiguration of TrustZone components like the TrustZone Protection Controller (TZPC) and TrustZone Address Space Controller (TZASC) during runtime, enabling changes to memory partitioning between the secure and non-secure worlds. In such cases, the secure world software would also need to support and be aware of such changes to adapt its memory management accordingly.

To determine if your specific SoC supports changing the memory size of the secure world at runtime, consult the technical reference manual or datasheet provided by the manufacturer. It's important to consider the potential security implications and any restrictions when dynamically changing memory partitioning, as it could introduce new attack surfaces or vulnerabilities.

Question: What is MMU?

Answer:

MMU stands for Memory Management Unit. It is a hardware component found in many computer systems, including those based on the ARM architecture. The primary function of the MMU is to manage the system's memory by translating virtual memory addresses to physical memory addresses, enabling memory protection, and controlling access permissions for different regions of memory.

Key functions of an MMU include:

1. Virtual-to-physical address translation: MMUs use a system of page tables to translate virtual memory addresses, used by software, to physical memory addresses, which correspond to the actual location of data in the memory hardware. This allows software to access memory in a uniform and consistent manner, even when the actual memory layout is fragmented or changes dynamically.
2. Memory protection: MMUs enable separation and isolation of different memory regions based on the access permissions set for each region. This helps prevent unauthorized access to memory and protects the system from bugs, crashes, or security vulnerabilities that could result from unintended memory access.
3. Caching and cache management: MMUs often work closely with the CPU's cache to ensure efficient and consistent access to memory. By managing cache attributes and maintaining a consistent view of memory, MMUs help improve overall system performance.
4. Supporting virtual memory: MMUs play a crucial role in implementing virtual memory, which allows multiple applications and processes to run simultaneously while maintaining the illusion that each has access to a large, contiguous block of memory. Virtual memory allows the system to use disk space as an extension of RAM, effectively increasing the available memory and enabling better multitasking and memory management.

Overall, the MMU is an essential component in modern computer systems that helps manage memory efficiently, enhances system performance, and provides a foundation for implementing advanced memory-related features such as virtual memory and memory protection.

Question: What is TLB?

Answer:

TLB stands for Translation Lookaside Buffer. It is a specialized cache used by a computer's memory management unit (MMU) to speed up the translation of virtual memory addresses to physical memory addresses. The TLB stores a small number of recently-used virtual-to-physical address mappings, allowing the MMU to quickly look up the physical address for a given virtual address without having to consult the full page table stored in main memory.

Since accessing the main memory is generally slower than accessing the cache, the TLB helps improve the overall performance of the system. When the MMU receives a virtual address to translate, it first checks the TLB to see if the required mapping is already present. If the mapping is found in the TLB (a TLB hit), the MMU can quickly obtain the corresponding physical address. If the mapping is not in the TLB (a TLB miss), the MMU must access the page table in main memory, which takes more time.

To maintain the efficiency of the TLB, the system uses various algorithms to decide which mappings should be stored in the TLB and which ones should be evicted when the TLB becomes full. These algorithms aim to optimize the TLB hit rate by keeping the most frequently or recently used mappings in the buffer.

In summary, the TLB is a critical hardware component that works with the MMU to accelerate virtual-to-physical address translation, thus improving overall system performance and reducing memory access latency.

CPU

Question: What is SMP?

Answer:

SMP stands for Symmetric Multiprocessing. It is an architecture used in multi-core and multi-processor computer systems where two or more identical processors or processor cores are connected to a single shared memory and are controlled by a single operating system. In an SMP system, each processor or core can execute different tasks independently, but they share the same memory resources and I/O devices.

SMP allows for more efficient processing and better utilization of system resources, as multiple processors or cores can work simultaneously on different tasks, improving overall system performance. This architecture also provides better fault tolerance and load balancing, as tasks can be distributed among the available processors, reducing the workload on any single processor and minimizing the impact of a processor failure on the overall system.

In an SMP system, the operating system and applications must be designed to take advantage of multiple processors, using techniques such as multithreading and parallelism to distribute tasks among the available processors. Modern operating systems, such as Linux, Windows, and macOS, are capable of managing SMP systems and can automatically distribute workloads across multiple processors or cores.

Question: What are difference between primary and seconday CPU cores?

Answer:

In the context of multi-core or multi-processor systems, primary and secondary CPU cores generally refer to the roles these cores play in the system, with the primary core being responsible for the initialization process and secondary cores assisting in processing tasks once the system is up and running.

Here are the main differences between primary and secondary CPU cores:

1. Initialization and Boot-up: The primary CPU core (sometimes called the boot core or master core) is responsible for initializing the system during boot-up. It executes the initial instructions, loads the firmware or BIOS, and configures the hardware components. Once the system is initialized, the primary core starts the operating system and enables other secondary cores.
2. Secondary Core Activation: Secondary cores (also called slave cores or worker cores) are enabled by the primary core after the system initialization process is complete. They start to execute tasks assigned by the primary core or the operating system.

3. Task Management: The primary core typically manages tasks and distributes them among secondary cores. However, modern operating systems can manage tasks independently and distribute them across all available cores (including the primary core) for efficient processing.
4. Power Management: In some systems, the primary core may have more advanced power management capabilities compared to secondary cores, enabling it to control the power states of secondary cores, such as turning them off during periods of low workload to save power.

It is essential to note that these differences apply primarily to systems where a distinction is made between primary and secondary cores. In modern symmetric multiprocessing (SMP) systems, all cores are often treated as equals once the system is up and running, and the operating system manages tasks and power states across all cores without any significant difference in their roles.

Question: Please summarize the two types of CPU cores as a table.

Answer:

Aspect	Primary Core (Boot/Master Core)	Secondary Core (Slave/Worker Core)
Initialization	Responsible for system initialization during boot-up	Activated by the primary core after system initialization
Task Management	Manages and distributes tasks among secondary cores	Executes tasks assigned by the primary core or OS
Power Management	May have advanced power management capabilities	Power states controlled by the primary core or OS
Core Activation	Always active during boot-up and system operation	Activated and deactivated as needed for workload demands

Please note that in modern symmetric multiprocessing (SMP) systems, these distinctions become less relevant, as all cores are treated as equals once the system is up and running, and the operating system manages tasks and power states across all cores without any significant difference in their roles.

Question: What are SCP and AP?

Answer:

SCP (System Control Processor) and AP (Application Processor) are two different types of processors often found in a system-on-chip (SoC) or embedded systems.

1. SCP (System Control Processor): The SCP is a low-power, dedicated processor responsible for managing system-level functions such as power management, thermal management, and other low-level tasks. The SCP typically runs a simple firmware, often referred to as the system control firmware (SCFW), which is in charge of controlling and monitoring various system components. The SCP usually runs at a lower frequency and consumes less power compared to the AP, as it is designed to handle background tasks and ensure efficient operation of the overall system.
2. AP (Application Processor): The AP, on the other hand, is a more powerful processor responsible for running the operating system (OS) and executing user applications. In a typical SoC, the AP will consist of one or more CPU cores, often accompanied by GPU, memory controllers, and other peripherals required for running the OS and applications. The AP manages more complex and compute-intensive tasks, making it the primary processing unit in the system.

In summary, the SCP is responsible for managing low-level system functions, while the AP is responsible for running the OS and executing user applications. Together, they enable efficient and seamless operation of the system.

Question: Do SCP and AP in ARM share memory?

Answer:

In an ARM-based system-on-chip (SoC), the System Control Processor (SCP) and Application Processor (AP) may share

memory, depending on the specific SoC design and implementation. Shared memory is often used as a means of communication and data exchange between the two processors.

In many cases, the SCP and AP will have access to a region of shared memory, typically implemented as SRAM, DRAM, or on-chip memory. This shared memory space allows the SCP and AP to exchange information, synchronize operations, or pass messages between them. The shared memory region can be protected by appropriate access control mechanisms, such as TrustZone or memory protection units (MPUs), to prevent unauthorized access or data corruption.

However, it is essential to note that the exact memory architecture and sharing mechanism depend on the specific SoC design and implementation. In some cases, the SCP and AP may have separate memory spaces, with communication occurring through dedicated hardware channels or interfaces.

In summary, SCP and AP in ARM-based SoCs can share memory depending on the specific design and implementation, enabling efficient communication and data exchange between the two processors.

Question: What is differenct between Armv8-A and Armv8-M? Please summarize as a table.

Answer:

Here is a summary of the differences between Armv8-A and Armv8-M in a table format:

Feature	Armv8-A	Armv8-M
Target Applications	High-performance systems (smartphones, tablets, servers, networking equipment)	Low-power, resource-constrained embedded systems (microcontrollers, IoT devices, wearables)
Instruction Set	AArch64 (64-bit) and AArch32 (32-bit)	AArch32 (32-bit)
TrustZone Support	Yes, for secure environment in complex systems	Yes, adapted for microcontrollers
Memory Management	Memory Management Unit (MMU) with virtual memory support	Memory Protection Unit (MPU) without virtual memory support
Exception Levels and Privilege Modes	Multiple levels (EL0-EL3) and privilege modes	Simpler model with fewer privilege modes

Armv8-A is designed for high-performance systems and supports both 32-bit and 64-bit instruction sets, while Armv8-M is targeted at low-power microcontrollers and supports only the 32-bit instruction set. The memory management, TrustZone implementation, and exception handling mechanisms are also tailored to the specific needs of their target applications.

SM & SPM & SPD

Question: What is Secure Monitor?

Answer:

The Secure Monitor is a critical component of the ARM TrustZone security technology that acts as an intermediary between the Secure and Non-Secure worlds. It operates at the highest privilege level (S-EL3) within the Secure world, ensuring the integrity and confidentiality of the Secure environment.

The primary responsibilities of the Secure Monitor include:

1. Handling transitions between Secure and Non-Secure states: When a Secure Monitor Call (SMC) instruction is executed by the Non-Secure world, the processor switches to the Secure state, and the Secure Monitor takes control to process the request. It can either perform the requested operation or forward the request to other components, such as the Secure Partition Manager (SPM) or a specific Secure Partition (SP).
2. Managing resources and access control: The Secure Monitor oversees access to secure system resources, such as memory regions, peripherals, and cryptographic keys, ensuring that only authorized Secure software components can access them.

3. Enforcing the security policy: The Secure Monitor is responsible for implementing and enforcing the overall security policy for the system. This includes defining the security boundaries between the Secure and Non-Secure worlds, as well as ensuring that the Secure world remains protected from potential attacks originating from the Non-Secure world.

In essence, the Secure Monitor is a vital component of ARM TrustZone that provides a trusted execution environment for security-sensitive applications and data, while also managing the transitions and interactions between the Secure and Non-Secure worlds.

Question: What is SMC?

Answer:

SMC stands for Secure Monitor Call, which is an ARM instruction used in TrustZone-enabled systems. The SMC instruction is used to transfer control between the non-secure (Normal) world and the secure world. When the non-secure world needs to access a secure resource or service, it uses the SMC instruction to trigger a switch to the secure world. The secure world then handles the request, performs the necessary operations, and returns control back to the non-secure world using the ERET (Exception Return) instruction.

The SMC instruction ensures that transitions between the two worlds are securely managed and controlled, allowing the secure world to maintain its isolation and integrity. Secure Monitor Calls are essential for implementing security services and trusted applications that require isolation from the non-secure world while still providing a controlled interface for communication between the two worlds.

Question: Can secure world request a service in normal world via SMC?

Answer:

Yes, the secure world can request a service in the normal world via SMC. Although it's more common for the normal world to request services from the secure world, the secure world may still need to access certain resources or services in the normal world. In such cases, the secure world can use SMC to make a request to the normal world.

However, it's important to note that the secure world should carefully manage any interactions with the normal world to avoid compromising its own security. When the secure world needs to access services or resources in the normal world, it should do so with the understanding that the normal world is potentially untrusted and might attempt to exploit any vulnerabilities in the communication process. Therefore, the implementation of such communication should be done cautiously, with a focus on maintaining the isolation and integrity of the secure world.

Question: What is SPM?

Answer:

SPM stands for Secure Partition Manager. In the context of ARM TrustZone or systems with secure partitions, the SPM is a component responsible for managing and orchestrating the execution of Secure Partitions (SPs). The SPM's role includes controlling access to resources, managing communication between SPs, ensuring the isolation of different secure partitions, and enforcing security policies.

In a nutshell, the Secure Partition Manager provides a framework for the secure world to manage the lifecycle and interactions of multiple secure partitions, ensuring that the security and isolation properties of each partition are maintained.

Question: What is SPD?

Answer:

In the context of ARM TrustZone, SPD (Secure Payload Dispatcher) is a component responsible for managing and dispatching Secure Partition (SP) requests. An SPD serves as an intermediary layer between the normal world and secure world, allowing for the secure processing of service requests from the normal world.

When a normal world application requires services provided by the secure world, it issues a request via SMC (Secure Monitor Call). The SPD receives this request, identifies the appropriate secure partition to handle the request, and dispatches the

service execution to the corresponding partition. After the secure partition has processed the request, the SPD returns the result to the normal world.

By using an SPD, the secure world can maintain better isolation and security, as the SPD ensures that the normal world does not have direct access to secure partitions or their resources.

Question: Can SPD handle the request from the secure world?

Answer:

Yes, the Secure Payload Dispatcher (SPD) can handle requests from both the normal world and the secure world. Although it is more common for the normal world to make service requests to the secure world, there might be cases where the secure world needs to access resources or services available in the normal world.

In such scenarios, the SPD plays a similar role in managing and dispatching these requests. It serves as an intermediary between the two worlds, ensuring that the communication is secure and that the appropriate services are invoked to handle the requests. This way, the secure world can access resources in the normal world without compromising its security and isolation.

Question: What are relationships of SPD, SPM and SMC?

Answer:

The Secure Partition Manager (SPM), Secure Partition Dispatcher (SPD), and Secure Monitor Call (SMC) are related concepts in the context of ARM TrustZone technology, which provides a secure and isolated environment for sensitive applications and data. Here's a brief explanation of each concept and their relationships:

1. Secure Monitor Call (SMC): SMC is a special instruction used to switch between the Secure and Non-Secure states in ARM TrustZone. When SMC is executed, the processor transitions to the Secure state, and the Secure Monitor (typically running at S-EL3) is responsible for handling the call. The Secure Monitor can then perform the requested operation or forward the request to a more specific component, such as the SPM or an SPD.
2. Secure Partition Manager (SPM): The SPM is responsible for managing Secure Partitions (SPs), which are isolated execution environments that host Trusted Applications (TAs) or other secure services. The SPM handles tasks such as creating, initializing, and terminating SPs, as well as managing access to secure resources. When the Non-Secure world needs to access a service provided by an SP, it can use the SMC instruction to request the SPM's assistance.
3. Secure Partition Dispatcher (SPD): The SPD is an optional component that facilitates communication between the Non-Secure world and the Secure world, particularly when using an SMC. The SPD can handle requests from the Non-Secure world, forward them to the appropriate SP, and return the results to the Non-Secure caller. In some implementations, the SPM and SPD can be combined into a single component.

In summary, SMC is an instruction used to transition between Secure and Non-Secure states, while the SPM and SPD are components that manage Secure Partitions and facilitate communication between the two worlds. When an SMC is issued, the request is

---

Question: Why are SPD and SPM mutually exclusive?

Answer:

The Secure Partition Manager (SPM) and the Secure Partition Dispatcher (SPD) are two different approaches to managing secure partitions in the ARMv8.4-A architecture, and they are mutually exclusive because they serve different purposes and are intended for different use cases.

1. Secure Partition Manager (SPM): SPM is designed for the ARMv8.4-A architecture and is responsible for managing Secure Partitions (SPs). SPs are isolated execution environments running inside the secure world, which can host multiple security services or Trusted Applications (TAs). The SPM is responsible for creating, initializing, scheduling, and managing communication between SPs. It enables fine-grained partitioning of resources and services within the secure world.
2. Secure Partition Dispatcher (SPD): SPD is a part of ARM Trusted Firmware (ATF) and is used to manage secure

services in ARMv8-A architecture. SPD provides an interface between the Normal World (NW) and the Secure World (SW). It is responsible for dispatching secure service requests from the NW to the appropriate secure execution environment, such as an SP or a TA, and returning the results back to the NW.

The reason they are mutually exclusive is that they cater to different architectural versions and use cases. SPM is designed specifically for ARMv8.4-A and provides advanced features for managing SPs, while SPD is used in ARMv8-A to manage secure services.

Using both SPM and SPD simultaneously would lead to conflicts in terms of resource allocation, scheduling, and communication between the secure world and the normal world. To avoid confusion and ensure proper secure partition management, either SPM or SPD is used, depending on the target ARM architecture and the specific use case.

Question: Is SPD an optional component?

Answer:

Yes, the Secure Partition Dispatcher (SPD) is an optional component within the ARM TrustZone security architecture. The SPD is responsible for managing and scheduling the execution of Secure Partitions (SPs) in the Secure world. It is a more advanced way of handling Secure services compared to the traditional monolithic Secure Monitor, which directly handles Secure world requests.

Using SPD allows for a more modular and flexible approach to implementing Secure services. However, if your use case does not require the functionality provided by Secure Partitions, you can choose not to include the SPD in your system. In that case, you will rely on the Secure Monitor to handle Secure world requests directly or use a different mechanism for managing Secure services.

Ultimately, the choice of whether to include an SPD in your system depends on your specific security requirements and the desired level of flexibility and modularity in your Secure software stack.

Question: Can I use SPD instead of SM?

Answer:

Yes, you can use the Secure Partition Dispatcher (SPD) instead of the traditional Secure Monitor (SM) in your ARM TrustZone-enabled system. The SPD provides a more flexible and modular approach to managing Secure services by handling and scheduling the execution of Secure Partitions (SPs) in the Secure world.

When using the SPD, the responsibility for managing Secure services is shifted from the monolithic Secure Monitor to the individual Secure Partitions, which are isolated from each other and can be developed and updated independently.

However, keep in mind that this decision depends on your specific security requirements and the desired level of flexibility and modularity in your Secure software stack. If you choose to use the SPD, it will replace the SM as the primary mechanism for managing Secure services in your system.

Question: Does TrustZone require SPM?

Answer:

TrustZone itself does not inherently require a Secure Partition Manager (SPM). TrustZone is a hardware-based security feature in ARM processors that provides a secure execution environment (Secure World) and isolates it from the less secure environment (Normal World).

However, when using ARMv8.4-A architecture or later, which introduces the concept of Secure Partitions (SPs) to improve isolation and modularity of the Secure World components, an SPM becomes an important part of the system. The SPM is responsible for managing the lifecycle of Secure Partitions and handling inter-partition communication and resource management.

So, while TrustZone doesn’t require an SPM in its basic form, utilizing SPM can enhance the security and modularity of the system when working with Secure Partitions in newer ARM architectures.

Secure Partition

Question: What is SP?

Answer:

SP stands for Secure Partition, which is a software component or execution environment running within the secure world of ARM TrustZone. Secure Partitions are designed to provide isolation and protection for sensitive code and data, such as cryptographic operations, key management, or secure storage. The secure world hosts these SPs, which can communicate with the normal world through a well-defined interface, typically via SMC (Secure Monitor Call) instructions.

SPs help to ensure that critical operations and information remain protected from potentially malicious or insecure code running in the normal world. By isolating these components, TrustZone enhances the overall security of a system.

Question: Can different SPs communicate together?

Answer:

Yes, different Secure Partitions (SPs) can communicate with each other within the secure world. However, this communication should be carefully designed and managed to maintain the isolation and security properties of each SP. Typically, an intermediary component like a secure partition manager or the secure monitor itself is responsible for facilitating communication between different SPs, ensuring that information sharing adheres to the security policies and requirements of each partition.

To enable secure communication between different SPs, developers should implement well-defined interfaces and access control policies to prevent unauthorized access or information leaks between partitions, maintaining the overall security goals of the system.

Question: How to create a SP and when?

Answer:

Creating a Secure Partition (SP) typically involves a few steps and is usually done during the system initialization or when a secure application needs to be launched. Here are the steps to create an SP:

1. Design: Begin by identifying the secure functionality or application you want to isolate in a separate SP. Determine the security requirements, resource access, and communication interfaces needed for the secure partition.
2. Implementation: Implement the secure functionality or application, adhering to the security requirements and following best practices for secure programming. This may involve writing code and creating any necessary data structures, as well as implementing communication interfaces to interact with other SPs or the normal world.
3. Configuration: Configure the system to support the SP. This might involve configuring memory regions, peripherals, and access controls to meet the security requirements of the SP. This is typically done in the system's bootloader, firmware, or Secure Partition Manager (SPM), depending on the platform.
4. Initialization: Load and initialize the SP during system startup or when required. This process includes loading the secure partition's code and data into the protected memory regions and initializing the partition's state. The SPM or another trusted component may handle this initialization process.
5. Execution: Once initialized, the SP can be executed, providing its secure functionality to other components or applications in the system. The SPM or a trusted scheduler will manage the execution of the SP, ensuring proper isolation and adherence to security policies.

Creating an SP depends on the specific platform or system you are working with, as well as the requirements of your secure application. Ensure that you consult the relevant documentation and guidelines for your platform to create and manage secure partitions effectively.

Question: Can I create a SP in run-time?

Answer:

Creating a Secure Partition (SP) during runtime is generally not recommended, as it could introduce security risks and may not be supported by some platforms. Secure Partitions are typically initialized during system startup or secure application launch to ensure proper isolation and security.

However, if your platform supports creating an SP during runtime and you have a valid use case, it might be possible. You would need to ensure that you take the necessary precautions to maintain the security of the system and the SPs, including:

1. Ensuring proper isolation between the newly created SP and the rest of the system, including other SPs.
2. Configuring memory regions, peripherals, and access controls correctly.
3. Verifying the integrity and authenticity of the code and data being loaded into the new SP.

Keep in mind that creating an SP during runtime may be more complex and error-prone than creating one during system startup or secure application launch. You should carefully consider the security implications and consult your platform's documentation to determine if this approach is supported and appropriate for your use case.

Question: What is TA?

Answer:

TA stands for Trusted Application. In the context of ARM TrustZone or Trusted Execution Environments (TEEs), a Trusted Application is a secure piece of software running in the secure world. These applications are designed to handle sensitive data or perform security-critical operations, isolated from the normal world or non-secure applications.

Trusted Applications typically run on top of a TEE Operating System (TEE OS), which provides services like secure storage, cryptographic operations, and secure communication with the normal world. The TEE OS is responsible for managing and isolating multiple Trusted Applications, ensuring that they don't interfere with each other or leak sensitive data. Examples of TEE OSs include OP-TEE and Trustonic Kinibi.

Trusted Applications are usually loaded and instantiated by the TEE OS based on requests from the normal world or other secure world components. They can be used to perform a wide range of tasks, such as digital rights management, secure payment processing, biometric authentication, and more.

Question: How do TAs and SPs communicate each other?

Answer:

TAs (Trusted Applications) and SPs (Secure Partitions) can communicate with each other using Inter-Partition Communication (IPC) mechanisms, which are platform-specific and depend on the underlying TEE (Trusted Execution Environment) OS or ARMv8.4-A architecture implementation.

When TAs and SPs need to communicate, they typically follow these steps:

1. Establish a communication channel: Both TAs and SPs should have a well-defined communication interface, such as shared memory or message passing. The implementation details depend on the TEE OS for TAs and the platform's support for SPs.
2. Secure the communication channel: To ensure the confidentiality and integrity of data exchanged between TAs and SPs, use cryptographic techniques, such as encryption and authentication.
3. Exchange messages: TAs and SPs can exchange messages through the established communication channel. These messages may include requests for services, data to be processed, or responses.
4. Synchronize communication: Depending on the IPC mechanism used, it may be necessary to implement synchronization primitives, such as semaphores or mutexes, to coordinate communication between TAs and SPs.
5. Handle errors and exceptions: Implement error handling and exception mechanisms to manage communication failures, timeouts, or other unexpected events.

It is essential to note that the communication between TAs and SPs requires careful implementation to ensure security and prevent potential attacks. Therefore, following best practices and guidelines for secure IPC is critical when developing communication between TAs and SPs.

Question: Can a SP host many TAs?

Answer:

A Secure Partition (SP) can host multiple Trusted Applications (TAs) in a single isolated execution environment within the secure world. The Secure Partition Manager (SPM) is responsible for managing and scheduling these TAs. Each TA can provide a specific security service, and they can all run within the same SP. By hosting multiple TAs within an SP, you can share resources among TAs, such as memory or peripherals, which can improve overall efficiency and reduce the overhead of managing individual TAs in separate SPs.

Question: Can a TA communicate different SP?

Answer:

A Trusted Application (TA) can communicate with different Secure Partitions (SPs) through a secure communication channel, which is typically established by the Secure Partition Manager (SPM) or another trusted component in the secure world. The communication between TAs and different SPs is usually facilitated by a well-defined interface using messages or function calls. This allows TAs to request services or share data with other SPs in a controlled and secure manner, without directly accessing each other's memory or resources. However, it is important to note that the level of isolation between TAs and SPs may vary depending on the specific implementation and platform.

Question: Is normal world a secure parition?

Answer:

No, the Normal World is not a Secure Partition. The Normal World and Secure World are separate execution environments in ARM TrustZone-enabled systems. The Secure World is designed for running security-sensitive code and handling sensitive data, whereas the Normal World is intended for running general-purpose applications and operating systems.

Secure Partitions (SPs) are isolated execution environments within the Secure World. These partitions allow the development and management of security services and Trusted Applications (TAs) independently from each other. They provide additional separation and isolation for sensitive code and data within the Secure World.

In contrast, the Normal World does not provide the same level of protection and isolation as the Secure World and its Secure Partitions, making it unsuitable for hosting security-critical components.

Question: Is secure world a secure partition?

Answer:

No, the Secure World is not a Secure Partition, but it contains Secure Partitions. The Secure World is a separate, protected execution environment enabled by ARM TrustZone technology, designed for running security-sensitive code and handling sensitive data.

Secure Partitions (SPs) are isolated execution environments within the Secure World. They provide additional separation and isolation for sensitive code and data, allowing the development and management of security services and Trusted Applications (TAs) independently from each other.

So, while the Secure World provides a protected environment, Secure Partitions offer even more granular isolation within the Secure World itself.

Privilege Levels

Question: What are EL0, EL1, EL2, S-EL0, S-EL1, S-EL2, S-EL3?

Answer:

In the ARMv8-A architecture, the privilege levels (EL) define different modes of operation for the processor, with each level

having different access rights and capabilities. There are four Exception Levels (EL0, EL1, EL2, and EL3), and each level can operate in both Secure and Non-Secure states. Here's a brief explanation of each:

1. EL0: This is the least privileged level, where user-space applications run. It has limited access to system resources and cannot execute privileged instructions.
2. EL1: This level is where the operating system kernel runs. It has more privileges than EL0, enabling it to manage system resources, configure the hardware, and isolate user-space applications from each other.
3. EL2: This level is intended for hypervisors, which manage multiple virtual machines or operating systems. It provides the necessary isolation and control to manage virtualization.
4. EL3: This is the most privileged level, typically used for low-level firmware, such as a bootloader or a Trusted Execution Environment (TEE). It is responsible for setting up the system and ensuring a secure boot process.

The "S-" prefix indicates that the exception level is operating in the Secure state, which is part of ARM's TrustZone technology. TrustZone creates a separate, isolated environment for secure applications and data, ensuring that sensitive information remains protected from potential attacks or unauthorized access:

1. S-EL0: Secure User mode, where secure user-space applications run.
2. S-EL1: Secure Kernel mode, where the secure operating system kernel runs.
3. S-EL2: Secure Hypervisor mode, where a secure hypervisor can manage secure virtual machines.
4. S-EL3: Secure Monitor mode, where low-level firmware like a secure bootloader or a TEE operates.

In summary, the ARMv8-A architecture organizes the privilege levels (EL0, EL1, EL2, and EL3) and the Secure (S-EL0, S-EL1, S-EL2, and S-EL3) and Non-Secure states to create a robust and flexible environment for running a mix of secure and non-secure applications, operating systems, and hypervisors.

Question: 2023/4/10: What is exception level?

Answer:

Exception level (EL) is a concept in the ARM architecture that defines different privilege levels for executing software. The ARMv8-A architecture, for instance, supports four exception levels: EL0, EL1, EL2, and EL3. Each level has different access rights and capabilities, which provide a mechanism for isolating different types of software and ensuring that less privileged software cannot interfere with more privileged software.

Here's a brief overview of the exception levels in ARMv8-A architecture:

- EL0: This is the least privileged level and is used for running user applications. Software at this level has limited access to system resources and cannot directly access privileged hardware operations.
- EL1: This level is used for running operating system kernels and is more privileged than EL0. Software at this level can access system resources and control hardware operations but is still restricted from certain sensitive operations.
- EL2: This level is used for running hypervisors, which manage and control virtual machines. Software at this level has more privileged access to hardware resources and can control the operation of multiple EL1 instances (virtual machines).
- EL3: This is the most privileged level and is typically used for secure monitor or firmware code, which manages the TrustZone security features. Software at this level has full control over the hardware resources and can enforce secure access policies across the entire system.

These exception levels provide a structured way to separate software components based on their required privileges and ensure a secure execution environment.

Question: 2023/4/10: How does a software component know which EL it owns?

Answer:

A software component does not inherently "know" its exception level (EL). However, when it is executed, it runs at the EL designated by the system configuration and the control registers set by more privileged software.

Typically, software components are designed to operate at specific exception levels. The boot process and the system's

firmware, which run at the highest privilege level (e.g., EL3 in ARMv8-A), are responsible for setting up the initial exception levels and transitioning to lower-privileged levels as needed. During this process, they load and configure less privileged software components (such as the hypervisor, operating system kernel, and user applications) to execute at the appropriate EL.

At runtime, a software component can determine its current exception level by querying the CurrentEL register. This register contains a field that indicates the current exception level, allowing the software to determine its current privilege level and act accordingly.

It's important to note that this query can only be performed if the software component is designed to check its own exception level, and it is assumed that the component has been correctly loaded and configured by the more privileged software in the system.

Question: 2023/4/10: Can a software component change the EL?

Answer:

A software component can't directly change its own exception level (EL), but it can initiate a process that results in a change of EL. This typically happens through a specific event or instruction that triggers a change in privilege level, such as an exception, interrupt, or a special instruction like the "Secure Monitor Call" (SMC) in ARM systems.

For example, when an exception occurs, the processor switches to a higher EL to handle the exception. Once the exception has been dealt with, the software running at the higher EL can return to the previous, lower EL using the "Exception Return" (ERET) instruction in ARM systems or a similar mechanism in other architectures.

In the ARM architecture, a lower EL software can also request services from a higher EL software (e.g., from the normal world to the secure world in TrustZone) using the SMC instruction. The SMC instruction causes a switch to a higher EL, where the requested service is provided, and then the control is returned back to the lower EL.

Keep in mind that a software component can only initiate a change to a higher EL if it has been designed to do so and the system is configured to allow such transitions. The actual EL transition is managed by the hardware and the more privileged software components.

Question: 2023/4/10: Are ELs of TrustZone?

Answer:

Exception Levels (ELs) are a concept in the ARM architecture, not specific to TrustZone. However, TrustZone does make use of the ELs to separate and manage privileges between the secure and non-secure worlds.

In ARMv8-A architecture, there are four ELs (EL0 to EL3). TrustZone operates on EL3 and S-EL1. EL3 is the Secure Monitor level, responsible for managing the transition between the secure and non-secure worlds. S-EL1 is used for secure operating systems or hypervisors.

In summary, while ELs are part of the ARM architecture, TrustZone does make use of them to manage privileges and ensure security between the two worlds (secure and non-secure).
