ARM Security Technology

Building a Secure System using TrustZone® Technology

ARM®

PRD29-GENC-009492C

ARM Security Technology

Building a Secure System using TrustZone Technology

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Change
December 2008	A	First release
January 2009	B	Minor language clarifications Fixed monitor latency calculation onpage 5-12
April 2009	C	Added information related to multiprocessor systems: *Accelerator Coherency Port* on page 3-10 *Multiprocessor systems with the Security Extensions* on page 3-13 *Multiprocessor debug control* on page 3-18 *Secure software and multiprocessor systems* on page 5-13 *Hardware design checklist* on page 7-3 *Software design checklist* on page 7-5

Proprietary Notice

Words and logos marked with or   are registered trademarks or trademarks owned by ARM Limited, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Unrestricted Access is an ARM internal classification.

Product Status

This document is an informative whitepaper related to ARM security technology, and is not directly related to any individual product.

Web Address

`http://www.arm.com`

Contents
ARM Security Technology

Preface

Chapter 1 Introduction

Chapter 2 System Security

Chapter 3 TrustZone Hardware Architecture

Chapter 4 TrustZone Hardware Library

Chapter 5 TrustZone Software Architecture

Chapter 6 TrustZone System Design

Chapter 7 Design Checklists

Preface

This preface introduces the *ARM Security Technology* document. It contains the following sections:

- *About this document* on page viii
- *Using this document* on page ix
- *Further reading* on page x
- *Feedback* on page xi

About this document

This document provides an overview of the ARM TrustZone technology and how this can provide a practical level of security through careful System-on-a-Chip (SoC) configuration and software design.

ARM TrustZone technology includes the ARM Security Extensions to the processor, the security signals added to the AMBA®3 bus infrastructure, and a number of pieces of peripheral Intellectual Property (IP) which can be used to build security on top of the processor architecture and system architecture.

Caution

This document does not provide a full specification of any of the technology involved. See the appropriate Technical Reference Manuals.

Intended audience

This whitepaper is suitable for all developers who are making use of the ARM TrustZone technology, whether they are writing security requirements, designing a SoC, developing software, or auditing a design for security.

This document assumes that you are familiar with the ARM processor architecture and common hardware and software terminology.

Using this document

This document is organized into the following chapters:

Chapter 1 ***Introduction***

Background material on the embedded security ecosystem, a discussion on who might attack a device, and what form the attack may take.

Chapter 2 ***System Security***

An overview of some of the existing security technologies that are deployed into embedded devices, including some of their strengths and weaknesses. This chapter also outlines the design philosophy of the ARM TrustZone technology, and how it encompasses the strengths of many of the alternative solutions.

Chapter 3 ***TrustZone Hardware Architecture***

A detailed description of the ARM TrustZone technology, and how it impacts the fundamental system components.

This chapter is split into three parts: the first looks at the impact of TrustZone technology on the system infrastructure, the second looks at the impact of the technology to the ARM processor core, and the last part looks at the changes to the debug architecture.

Chapter 4 ***TrustZone Hardware Library***

An overview of the TrustZone-aware peripheral IP that is available from ARM.

This section also includes some design suggestions which enable limited integration of existing IP based on AMBA2 AHB™ interfaces within a system design.

Chapter 5 ***TrustZone Software Architecture***

An introduction to some of the possible software design choices when using an ARM processor implementing the ARM Security Extensions.

Chapter 6 ***TrustZone System Design***

An example system design using Digital Rights Management and Mobile Payment as example use cases.

Chapter 7 ***Design Checklists***

Use the checklists in this chapter for prompts when designing or reviewing a system using TrustZone technology.

Further reading

This section lists publications from both ARM Limited and third parties that provide additional information.

ARM periodically provides updates and corrections to its documentation. See `http://infocenter.arm.com` for the latest versions of the Technical Reference Manuals, User Guides, and the ARM Frequently Asked Questions list.

ARM publications

Refer to the following publications from ARM Limited:

- *ARM Architecture Reference Manual ARM®v7-A and ARM®v7-R edition* (ARM DDI 0406)

External publications

Refer to the following publications from third parties:

- Trusted Computing Group: Securing Mobile Devices on Converged Networks

  https://www.trustedcomputinggroup.org/groups/mobile/Final_iGR_mobile_security_white_paper_sept_2006.pdf
- Reuters UK: R u getting the msg about stolen mobiles?

  http://uk.reuters.com/article/domesticNews/idUKL2175341320070621
- US Department of Transportation: The Incidence Rate of Odometer Fraud

  http://www.nhtsa.dot.gov/cars/rules/regrev/evaluate/pdf/809441.pdf
- Washington Post: Lost a BlackBerry? Data Could Open a Security Breach

  http://www.washingtonpost.com/wp-dyn/content/article/2005/07/24/AR2005072401135.html
- Securing Java, G. McGraw and E. Felten

  http://www.securingjava.com

Feedback

ARM Limited welcomes feedback on this document.

Feedback on this document

If you have any comments on this document, send email to `errata@arm.com` giving:

- the document title
- the document number
- the page number(s) to which your comments apply
- a concise explanation of your comments.

General suggestions for additions and improvements are also welcome.

Chapter 1
Introduction

The chapter provides some of the background related to security in embedded systems.

This chapter includes the following sections:

1.1 What is security?

In very abstract terms, the term *security* can be used to cover a number of very different underlying features of a design. However, it is essentially a property of the system which ensures that resources of value cannot be copied, damaged, or made unavailable to genuine users. Every system design will require a different set of security properties, depending on the type and value of the *assets* it is trying to *defend* against malicious *attack*.

Asset A resource of value which is worth protecting. An asset may be a tangible object, such as a user password, or may be an intangible asset, such as network availability.

To Attack The act of intentionally trying to acquire, damage or disrupt an asset which you do not have permission to access.

An attack may include the use of malicious software, hardware monitoring and hardware tampering.

To Defend The act of designing a system that includes hardware or software mechanisms which provide countermeasures against attacks.

1.1.1 Fundamental security properties

The fundamental security properties on which nearly every higher level property can be based are those of *confidentiality* and *integrity.*

Confidentiality An asset that is confidential cannot be copied or stolen by a defined set of attacks.

This property is essential for assets such as passwords and cryptographic keys.

Integrity An asset that has its integrity assured is defended against modification by a defined set of attacks.

This property is essential for some of the on-device root secrets on which the rest of the system security is based, and for the security software once it is running.

Authenticity In some circumstances a design cannot provide integrity, so instead provides the property of *authenticity.* In this case the value of the asset can be changed by an attacker, but the defender will be able to detect the change before the asset is used and hence before the attack can cause a security fault.

This property is essential for security software. If an attacker can tamper with the program code before it is loaded into a safe execution location, without being detected, then the security provided by the software can be bypassed.

1.1.2 Limitations of security solutions

All security solutions are designed to defend against only a subset of the possible attacks that they may experience. Defending against all possible attacks is an impossible task; there is always someone willing to spend a significant amount of time and money to break any security scheme using very complex attacks. A design must therefore decide which assets it wants to protect, and which of the possible attacks it wants to protect the assets against. This is perhaps the most critical part of the design process; a design that protects the wrong assets against an incorrect or incomplete list of attacks can be easily broken.

Taking the position that all security can be broken with enough time and money, the security requirements for a design should not be described as “impossible to bypass” but should be described in value terms: “attack A on asset B should take at least Y days and Z dollars”. If a set of countermeasures mean that a successful attack will take too long or will cost too much, then the defense is a success. Most attackers will move on to a different target if they find themselves in this situation.

1.2 The need for security

Embedded devices are handling data of increasing value, such as consumer banking credentials. They are also steadily becoming open software platforms with high levels of off-device connectivity that enable the consumer to download arbitrary third-party applications. This puts these devices in a high risk position. In the desktop environment the combination of open software platforms, arbitrary application download, and valuable assets present on the device, have been shown to significantly increase the risk of a security vulnerability being exploited. Additionally, if the value of data passing through these systems is increasing, it is more likely that attackers will invest in breaking them.

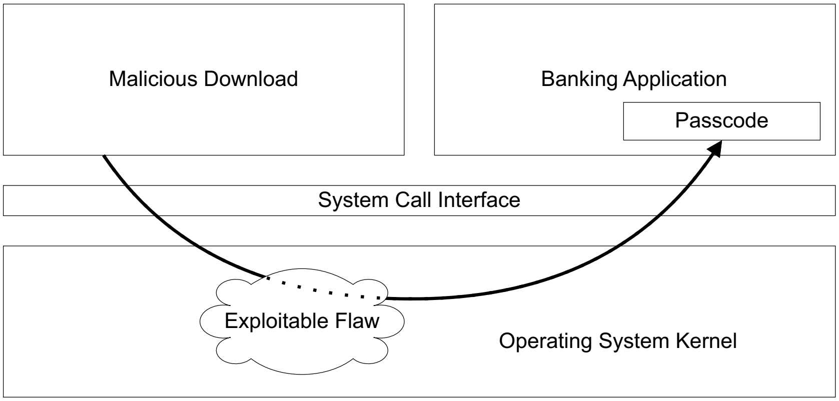


Figure 1-1 : Attacking an application

The aim of the ARM TrustZone technology is to enable a device to benefit from both a feature-rich open operating environment and a robust security solution. A well designed system hardware architecture and appropriate secure software design can ensure that the sensitive data remains safe, no matter what the less trusted operating environment does.

1.2.1 Hardware enforced security

Making an embedded product safe from malicious attacks has consequences for the entire system design. The hardware and software present in the device must work together to enable robust security countermeasures against the correct types of attack. The Trusted Computing Group whitepaper "Securing Mobile Devices on Converged Networks" identifies that the best protected systems have dedicated hardware security measures included from the beginning of their design process, starting with the specification for the processor core and the SoC infrastructure. Careful consideration of hardware security at this early stage allows protection to be built into the device that would be impossible to add later in the design process.

ARM TrustZone technology enables this level of system-wide security by integrating protective measures into the ARM processor, bus fabric, and system peripheral IP. Through a combination of integrated hardware and software components, TrustZone technology provides a framework that allows a diverse range of secure system architectures to be implemented with minimal impact on the cost of the device.

1.3 What are the threats?

Before exploring the details of the TrustZone hardware architecture it is important to understand what is meant by security in this context and how the risks posed by the attacks compare with the cost of prevention. Only with this information can a system designer make justified design choices regarding what to protect and how much to invest in hardware or software defenses.

There are many examples, spanning multiple applications and industries, of the costs associated with the failure of embedded systems to resist attack. Some attacks, such as payment fraud, cause a direct cost which has to be covered by the service provider, while others, such as the addition of a modified hardware chip to a set-top box, can cause a prolonged loss of revenues over the lifetime of the device.

1.3.1 Market sector overview

The sensitive assets that each market sector tries to protect against attack are diverse. For example, mobile handsets aim to protect the integrity of the radio network, while television set-top boxes prevent unauthorized access to subscription channels. The varied type and value of the assets being protected, combined with the different underlying system implementations, mean that the attacks experienced by each are also diverse. This section aims to outline a small part of the security history in some of the market sectors where TrustZone technology might be deployed.

Mobile sector

Two critical parts of a GSM handset are the International Mobile Equipment Identity (IMEI) code, a unique 15-digit code used to identify an individual handset when it connects to the network, and the low level SIMLock protocol which is used to bind a particular device to SIM cards of a particular network operator.

Both of these components are used to provide a security feature: the IMEI is used to block stolen handsets from accessing the network, and the SIMLock protocol is used to tie the device to the operator for the duration of a contract. On many handsets both of these protection mechanisms can be bypassed with little effort, typically using a USB cable and a reprogramming tool running on a desktop workstation.

The result of these insecurities in the implementation is an opportunity for fraud to be committed on such a large scale that statistics reported by Reuters UK suggest it is driving half of all street crime through mobile phone thefts, and costs the industry billions of dollars every year.

Security requirements placed on new mobile devices no longer relate only to the network, but also to content and services available locally on the device. Protection of digital media content through Digital Rights Management (DRM), and protection of confidential user data, such as synchronized email accounts, is becoming critical as both operators and users try to obtain more value from their devices.

Consumer electronics and embedded sector

The requirements placed on consumer electronics, such as portable games consoles and home movie players, are converging with those seen in the mobile market. Increasing wired and wireless connectivity, greater storage of user data, dynamic download of programmable content, and handling of higher value services, all suggest the need for a high-performance and robust security environment.

Security attacks are not limited to open systems with user-extensible software stacks. Within the automotive market most systems are closed or deeply embedded, yet odometer fraud, in which the mileage reading is rolled back to inflate the price of a second-hand vehicle, is still prevalent. The U.S. Department of Transportation reports that this fraud alone costs American consumers hundreds of millions of dollars every year in inflated vehicle prices.

Security features typically encountered in these embedded systems are those that verify firmware updates are authentic, and those that ensure that debug mechanisms cannot be used maliciously. These requirements can be met using a well designed system built using ARM TrustZone technology.

1.3.2 Economic value in security issues

Almost every security design is justified using a risk analysis that balances the probability of a successful attack, the cost to the business if an attack is successful, and the cost of preventing that attack. In some cases the risk analysis will decide that the probability of an attack is too low to be worth defending. In other cases the cost of a suitable defense is too high in comparison with the value of the asset being protected. However, in most cases some form of asset defense can be justified. The next few chapters will show that ARM TrustZone technology can potentially lower the cost of implementing a security solution; the architecture allows reduced system complexity, which leads to reduced design costs, development costs and testing costs. This enables system designs to protect a much wider set of assets than they have historically been able to do, if only for economic reasons.

Many of the possible attackers can also be considered to be motivated by economic reasoning. The biggest losses in industry originate from attacks perpetrated by professional hackers rather than end users, and professional hackers are predominantly motivated by financial gain. They will not invest more in a hack than they can reasonably expect to see in financial return.

Class-breaking attacks

Many device attacks are driven by professional criminals, academic security researchers, and enthusiasts working at home. Whether for financial gain, academic prestige, or just for fun, these groups can focus significant amounts of time into analyzing a system and developing complex attacks designed to break it.

The most sought after result of these attacks is a *class-break*; an easily reproducible attack that can be used to break a whole generation, or class, of devices. The most widely published attacks that fall in to this category are those deployed against consumer entertainment devices, such as the attacks that break the software restrictions on games consoles and the content protection schemes on DVD movies.

In many class-break scenarios in which the device hardware is attacked, the first investigative research by the attackers may cost a significant amount. This money is used to fund access to tools, such as electron microscopes and transistor stimulating lasers, which are used for silicon-level analysis. The objective of such an attack is to discover security weaknesses that can subsequently be exploited on multiple devices without significant cost. These class-breaking attacks can dramatically shift the balance of the economic argument in the favour of the attacker if the number of attackable devices is sufficiently high.

Positive economics

The benefits of a secure environment are not constrained to only the technical. It has been shown in studies that increased consumer confidence in easy to use and secure payment systems can boost consumer spending by up to twenty percent, while also enabling faster uptake of new revenue streams and different business models.

Manufacturers of devices are increasingly in a world where consumers have awareness of security issues, and security features of devices are becoming an issue of competitive differentiation. Devices with good security reviews and useful security functionality are being chosen in preference to less well specified devices. This is especially true for devices deployed into corporate enterprise environments, where mobile devices can connect to the internal network. The ability to synchronize confidential e-mail and calendar appointments to a smart phone is very useful but can, as the Washington Post reports, expose the owning corporation to high levels of risk if the device is lost or stolen. Many companies now restrict access to corporate networks, reducing the value of smart devices, or require additional security mechanisms on the devices themselves.

1.3.3 How are devices attacked?

The next most important factor in the design is the mechanisms that are used to perform the attacks. Different mechanisms of performing attacks are known as attack vectors, and these break down into three classes defined in this document– *hack attacks*, *shack attacks* and *lab attacks*.

Hack attack

A hack attack is one where the hacker is only capable of executing a software attack. Examples of hack attacks include viruses and malware which are downloaded to the device via a physical or a wireless connection.

In many cases of a successful hack attack the device user inadvertently approves the installation of the software that then executes the attack. This is either because the malware pretends to be a piece of the software that the user does want to install, or because the user does not understand the warning messages displayed by the operating environment.

In the book “Securing Java” there is a section which sums up the decision making capability of the typical user when it comes to choosing between security and desirable functionality:

“Given a choice between dancing pigs and security, users will pick dancing pigs every time.”

Shack attack

A shack attack is a low-budget hardware attack, using equipment that could be bought on the high street from a store such as Radio Shack. In these scenarios the attackers have physical access to the device, but not enough equipment or expertise to attack within the integrated circuit packages.

The attackers can attempt to connect to the device using JTAG debug, boundary scan I/O, and built-in self test facilities. They can passively monitor the system using logic probes and network analyzers to snoop bus lines, pins and system signals. The attackers may also be able to perform simple active hardware attacks, such as forcing pins and bus lines to be at a high or low voltage, reprogramming memory devices, and replacing hardware components with malicious alternatives.

Lab attack

The lab attack vector is the most comprehensive and invasive. If the attacker has access to laboratory equipment, such as electron microscopes, they can perform unlimited reverse engineering of the device. It must be assumed that the attacker can reverse engineer transistor-level detail for any sensitive part of the design - including logic and memories.

Attackers can reverse engineer a design, attach microscopic logic probes to silicon metal layers, and glitch a running circuit using lasers or other techniques. Attackers can also monitor analog signals, such as device power usage and electromagnetic emissions, to perform attacks such as cryptographic key analysis.

In most cases, considering the rule of thumb that states every device can be broken, a device should not try and defend against lab attack directly, but should take measures which limit the damage when a device is broken and therefore make the lab attack uneconomical. Use of per-device unique secrets is one example where reverse engineering a single device provides the attacker with no useful information; they have the secret for the device that they already own, but not any of the other devices in that class.

——— **Note** ———

TrustZone technology is designed to provide a hardware-enforced logical separation between security components and the rest of the SoC infrastructure.

Lab attacks are outside of the scope of the protection provided TrustZone technology, although a SoC using TrustZone can be used in conjunction with an ARM SecurCore$^{©}$ smartcard if protection against physical attacks is needed for some assets.

———————————

1.3.4 Who attacks devices?

Once a designer has identified the assets, and the possible attacks, it is important to identify the possible attackers. Different attackers can deploy different types of attack, and certain assets will only attract certain attackers. This analysis can help rationalize what attacks each asset needs to be protected against.

The analysis should also include a description of who is explicitly trusted with access to assets stored on the device. This can highlight weaknesses in the security model. There have been a number of published cases in which consumer data has been stolen from devices by maintenance or repair technicians, and subsequently published on the internet.

Remote attacker

The classical view of an attacker is a distributor of the malicious software that we see on desktop workstations: viruses and other malware. These attackers have no physical access to the device they are attacking, although may have access to a similar device to develop the attack, and rely on exploiting software vulnerabilities and user error to gain access to sensitive resources.

The increasing software complexity of embedded devices means that there are statistically more bugs for the attackers to exploit. The capability to install third-party code, including the execution of browser-based content, makes it easier to get the malicious code onto the devices to perform the attack.

Security specialist

The most technically capable attackers are criminal gangs, security experts, and users attacking devices for fun. These groups are capable of executing a lab attack, described on page 1-10.

This class of attacker is generally attempting to discover a class-break attack that can be reproduced across a large number of devices. This secondary attack may be deployed by another attacker.

Trusted developer

An often unconsidered attack is that executed, or at least assisted, by a person who could be considered trusted. Company employees have ended up in the news because they have stolen secret information related to designs, or purposefully damaged an internal computer network. In many cases these employees would have been considered explicitly trusted on the day before their attack was discovered.

Using the economic argument to justify defense against these kinds of attacks, it is usually cheaper, and faster, to bribe someone with access to design information than it is to reverse engineer a piece of silicon.

Although this type of attack can be difficult to protect against, some defensive measures in business process can be used. Processes that restrict access to sensitive materials, and audits who has accessed them, can be used.

Device owner

The last group of attackers against devices that we will discuss in this document are the device owners themselves. This group of attackers have a typical aim of gaining free access to services and content. In general the device owner is motivated to perform the attack, but is not technically competent. A technical expert will have developed an attack and published the details of it online; all that the device owner has to do is follow the recipe and reproduce it.

It is interesting to note that device owners in this category expose themselves to a higher than usual risk of attack. In their attempts to break the protections that are in place they mistakenly download prepackaged attack software from websites that are hosted by hackers; these sites often include viruses and other malware which the users then inadvertently install on their system.

Chapter 2
System Security

The chapter outlines some of the existing options for security in embedded systems, before introducing the ARM TrustZone hardware architecture.

This chapter includes the following sections:

- *System security* on page 2-2
- *TrustZone hardware security* on page 2-7

2.1 System security

System designs for embedded devices are complicated, including multiple independent processor cores, secondary bus masters such as DMA engines, and large numbers of memory and peripheral bus slaves. In addition to these functional components there is typically a parallel system infrastructure that provides invasive and non-invasive debug capabilities, as well as component boundary scan and Built-In-Self-Test (BIST) facilities.

Each of these subsystems in the platform has to be designed and integrated in such a way that it works with the security solution, rather than developing each sub-system independently of the security requirements. If the threat model for a device indicates that it needs to protect against shack attacks, there is no point securing only the functional part of the system. An attacker with unrestricted access to a debug port can bypass many of the functional protections that may exist.

This section aims to look at some of the security architectures that have historically been available on the market, and where they have strengths and weaknesses.

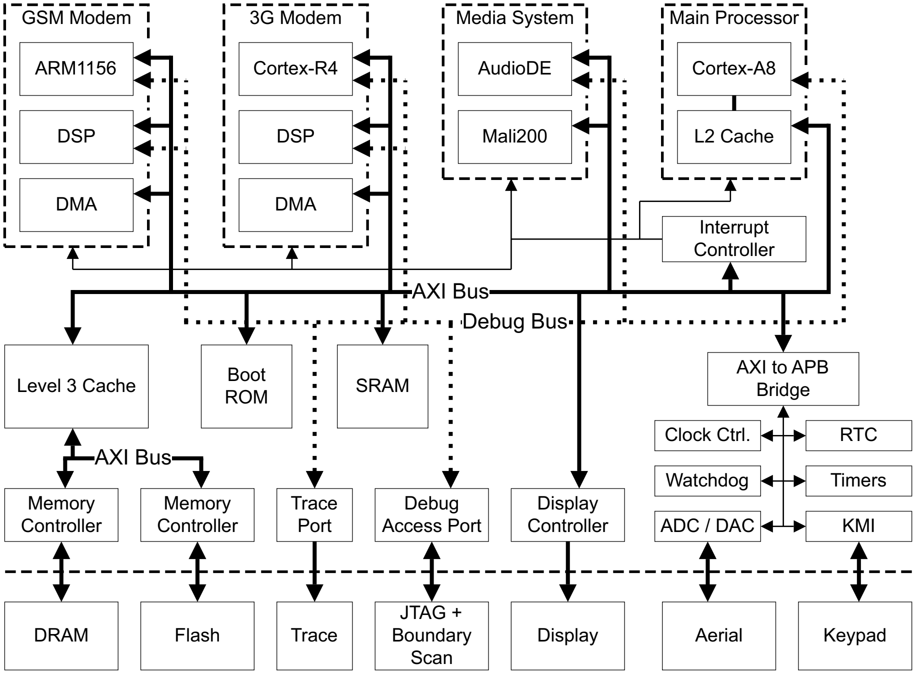


Figure 2-1 : A simplified schematic of a typical cellular handset SoC design

2.1.1 External hardware security module

The classic security solution for embedded applications is the inclusion of a dedicated hardware security module, or trusted element, that is outside of the main SoC. For example, a SIM card in a mobile handset, or a conditional access smartcard in a set-top box.

Advantages

The external security devices encapsulate the assets inside a physical device designed for robust security.

The use of a completely separate design and manufacturing flow allows use of techniques and silicon processes that can give high levels of tamper resistance and physical security.

The smartcard manufacture and personalization processes have frequently been formally certified through approved evaluation schemes. This makes them suitable for use in use cases that need a high degree of security assurance, such as credit card and debit card payment schemes.

Disadvantages

The hardware techniques and processes used for smartcards are impractical for standard SoC designs, as they would add significant effort to the design process and compromise the area, power efficiency and performance of the device.

The manufacturing methods that give the physical security also force a lower processor performance, in the region of 5-20 Mhz, and small quantities of RAM within the device. These design characteristics restrict the possible use cases which can be deployed on a smartcard to relatively static programs which can be run from non-volatile memory, and which do not need continuous high-bandwidth traffic across the security perimeter.

Another issue with smartcards is that they only provide a processing and secure storage function; they have no direct access to any of the user-interfaces of the parent device. A smartcard is reliant on software running outside of its security perimeter to provide these facilities, and consequently cannot protect all of the assets of interest. For example, a user's entry of a Personal Identification Number (PIN) must be managed by the less secure software outside of the smartcard, making it vulnerable to attack.

The commercial disadvantages of smartcards can be a more significant obstacle than the technical ones. Providing a smartcard alongside the main SoC is expensive, and is consequently uneconomic for most assets because they are not valuable enough.

2.1.2 Internal hardware security module

Hardware security modules that are located within the SoC sacrifice the hardware defenses provided by a smartcard for the cost reduction and convenience of being an integrated part of the system.

The precise form of integrated modules varies, but there are two dominant forms. The first is a hardware block which manages cryptographic operations and key storage. The second is a general purpose processing engine that is placed alongside the main processor, and which uses custom hardware logic to prevent unauthorized access to sensitive resources.

Advantages

The main advantage of these systems is the significant cost reduction and performance improvements over the smartcard option.

The systems that provide a dedicated general purpose processor for the security sub-system are comparable to the TrustZone hardware solution in terms of security.

Disadvantages

The cryptographic module implementations have a similar problem to the smartcards in that they have a restricted perimeter and as such are only capable of securing the cryptographic key material. Cryptography is a tool and not an objective in its own right; it is inevitable that the resources protected by the cryptographic methods will need to be used outside of the cryptographic module, where they are no longer protected by it and can be attacked.

The designs that provide a secondary general-purpose processor that is dedicated to the security sub-system have minor disadvantages when compared to a TrustZone system.

One issue is that the design needs a separate physical security processor. This processor is typically less powerful than the main applications processor, and also consumes significant silicon area. Additionally, communication between the two processors requires any data to be flushed to coherent memory, which is normally the external memory. This operation is time consuming and can use a significant amount of energy.

Another issue is that any separation of resources needs to be implemented in proprietary hardware extensions within the SoC. This requires significant design and test effort, and migrating or extending the system becomes more difficult.

The cryptographic module and secondary processor solutions presented here normally attempt to secure functional aspects of the system. The system security risks associated with the debug and test mechanisms available in the SoC are often ignored, and provide a vulnerable interface which can be attacked. In order to avoid this problem debug must be completely disabled, making it difficult to diagnose software issues in the field.

2.1.3 Software virtualization

Virtualization is a software security mechanism in which a highly trusted management layer, known as a hypervisor, runs in a privileged mode of a general purpose processor. The hypervisor separates multiple independent software platforms running on top of it using the Memory Management Unit (MMU), placing each inside a virtual machine controlled by the hypervisor software.

The compact nature of some hypervisors means that, if they can be thoroughly tested, the software running within one of the virtual machines it controls can have confidence that it is safe from attack by software running in one of the other virtual machines.

Note

This section describes *paravirtualization*; which is the type of virtualization typically found in the embedded market. Processor architectures also exist which have a special processor mode for hypervisors. These architectures allow a hypervisor to host a full operating system which can then subdivide the memory space it has been given by the hypervisor using an MMU.

Advantages

Any processor with an MMU can be used to implement a virtualization solution, and some of the common rich operating systems have been ported to run on top of them. There is also no requirement for additional hardware to implement a hypervisor.

Security sensitive applications can be ported to run in a secure environment running on top of the hypervisor, but outside of the view of the rich operating environment. The hypervisor can provide communications mechanisms which can be used to allow software virtual machines to communicate.

Disadvantages

The isolation provided by virtualization technology is restricted to the processor implementing the hypervisor. Any other bus masters in the system, such as DMA engines and Graphics Processing Units (GPUs), can bypass the protections provided by the hypervisor and thus must also be managed by the hypervisor to enforce the required security policy.

This is difficult to achieve without damaging the performance of the system; validating the complex inputs to a programmable GPU without reducing graphics performance is beyond the scope of most virtualization solutions.

Virtualization ignores the security issues associated with hardware attacks, such as threats that use the debug or test infrastructure. To secure a virtualized system against such an attack you need to disable debug and test visibility completely, which makes software development and diagnosis of defects in the field very difficult.

2.2 TrustZone hardware security

The problems associated with the security systems discussed in the previous section exist because the design can only protect some of the assets in a restricted part of the system, or because the security solution ignores big parts of the attack problem space.

In many cases the restrictive nature of the security design means that the wrong assets are protected. Cryptographic hardware blocks are primarily designed to protect the keys, which are undoubtedly valuable assets, but if an attacker can repeatedly steal decrypted content when it is consumed in the media players outside of the cryptographic hardware, the protection of the key will be of little consolation to the content rights owner.

Attacks on real devices deployed today are wide and varied, but can be shown to span all of the attack profiles outlined earlier in this chapter. Any security solution that only attempts to meet one of these threat areas is ignoring more than half of the problem and will not be able to defend against many attacks that exist in the real world. This may place the level of security that many solutions achieve beneath the level that they intended to provide, and consequently means that many assets are inadequately protected.

2.2.1 System-wide security

The ARM approach to enabling trusted computing within the embedded world is based on the concept of a trusted platform; a hardware architecture that extends the security infrastructure throughout the system design. Instead of protecting assets in a dedicated hardware block, the TrustZone architecture enables any part of the system to be made secure, enabling an end-to-end security solution that includes functional units and the debug infrastructure.

With suitable use of security protocols built on top of the TrustZone architecture, such as secure boot and authenticated debug enable, many of the possible hack and shack attack threats can have some form of countermeasure constructed. If these defenses can be used in combination with methods that mitigate the risks associated with lab attacks, for example, by making every device use statistically unique secrets, a very powerful solution begins to emerge.

The TrustZone hardware architecture is covered in detail in Chapter 3.

Chapter 3
TrustZone Hardware Architecture

The chapter provides details of the TrustZone hardware architecture, and its impacts on the system fabric, the processor, and the debug infrastructure.

This chapter includes the following sections:

3.1 Overview

The TrustZone hardware architecture aims to provide a security framework that enables a device to counter many of the specific threats that it will experience. Instead of providing a fixed one-size-fits-all security solution, TrustZone technology provides the infrastructure foundations that allow a SoC designer to choose from a range of components that can fulfil specific functions within the security environment.

The primary security objective of the architecture is actually rather simple; to enable the construction of a programmable environment that allows the confidentiality and integrity of almost any asset to be protected from specific attacks. A platform with these characteristics can be used to build a wide ranging set of security solutions which are not cost-effective with traditional methods.

The security of the system is achieved by partitioning all of the SoC's hardware and software resources so that they exist in one of two worlds - the Secure world for the security subsystem, and the Normal world for everything else. Hardware logic present in the TrustZone-enabled AMBA3 AXI™ bus fabric ensures that no Secure world resources can be accessed by the Normal world components, enabling a strong security perimeter to be built between the two. A design that places the sensitive resources in the Secure world, and implements robust software running on the secure processor cores, can protect almost any asset against many of the possible attacks, including those which are normally difficult to secure, such as passwords entered using a keyboard or touch-screen.

The second aspect of the TrustZone hardware architecture is the extensions that have been implemented in some of the ARM processor cores. These additions enable a single physical processor core to safely and efficiently execute code from both the Normal world and the Secure world in a time-sliced fashion. This removes the need for a dedicated security processor core, which saves silicon area and power, and allows high performance security software to run alongside the Normal world operating environment.

The final aspect of the TrustZone hardware architecture is a security-aware debug infrastructure which can enable control over access to Secure world debug, without impairing debug visibility of the Normal world.

Each of these three aspects are discussed in more detail in the following sections of this chapter.

——— **Note** ———

The ARM Architecture Reference Manual and many of the hardware component Technical Reference Manuals use the terms *Secure* and *Non-secure* – these are equivalent to the Secure world and the Normal world.

When referring solely to hardware this document will use the Secure and Non-secure naming conventions to avoid confusion. The terms Secure world and Normal world will be used to describe the combination of hardware and software that forms each execution environment.

3.2 System architecture

The architectural changes to the system IP provide the mechanisms to separate the hardware resources into the two worlds. This section looks at the impact of these changes and the components that have been modified.

3.2.1 The AMBA3 AXI system bus

The most significant feature of the extended bus design is the addition of an extra control signal for each of the read and write channels on the main system bus. These bits are known as the Non-Secure, or NS bits, and are defined in the public AMBA3 Advanced eXtensble Interface (AXI) bus protocol specification.

- **AWPROT[1]**: Write transaction – low is Secure and high is Non-secure.
- **ARPROT[1]**: Read transaction – low is Secure and high is Non-secure.

All bus masters set these signals when they make a new transaction, and the bus or slave decode logic must interpret them to ensure that the required security separation is not violated. All Non-secure masters must have their NS bits set high in the hardware, which makes it impossible for them to access Secure slaves. The address decode for the access will not match any Secure slave and the transaction will fail.

If a Non-secure master attempts to access a Secure slave it is implementation defined whether the operation fails silently or generates an error. An error may be raised by the slave or the bus, depending on the hardware peripheral design and bus configuration, consequently a **SLVERR** (slave error) or a **DECERR** (decode error) may occur.

3.2.2 The AMBA3 APB peripheral bus

One of the most useful features of the TrustZone architecture is the ability to secure peripherals, such as interrupt controllers, timers, and user I/O devices. This enables the security environment to be extended so that it can solve some of the wider security issues which need more than just a secure data processing environment. A secure interrupt controller and timer allows a non-interruptible secure task to monitor the system, a secure clock source enables robust DRM, and a securable keyboard peripheral enables secure entry of a user password.

The AMBA3 specification includes a low gate-count low-bandwidth peripheral bus known as the Advanced Peripheral Bus (APB), which is attached to the system bus using an AXI-to-APB bridge. The APB bus does not carry an equivalent of the NS bits. This ensures that existing AMBA2 APB peripherals are compatible with systems implementing TrustZone technology. The AXI-to-APB bridge hardware is responsible for managing the security of the APB peripherals; the bridge must reject transactions of inappropriate security setting and must not forward these requests to the peripherals.

3.2.3 Memory aliasing

The addition of the NS bit to the bus transactions, and to any cache tags in the system, can be viewed as providing a 33rd address bit. There is a 32-bit physical address space for Secure transactions and a 32-bit physical address space for Non-secure transactions.

As with any address space, including those without TrustZone technology, care must be taken to ensure that the 33-bit address space is used in such a way that data remains coherent in all of the locations that it is stored, otherwise data corruption may result.

Consider the case where a Secure world master wants to access a Non-secure slave that is cached. A design may implement either of the following choices:

- The master makes a Non-secure access to the slave.
- The master makes a Secure access to the slave and the Non-secure slave accepts the Secure transaction. The slave treats these accesses as Non-secure.

In the second design the hardware must support address space aliasing. In this aliased memory system the same memory location appears as two distinct locations in the address map, one Secure and one Non-secure. As a result, it is possible to have multiple values representing the same data present in the cache simultaneously. For modifiable data this aliasing causes coherency problems; if one copy of the data is modified while the other exists in the cache you will have versions of the data but both will be different. System designers must be aware of potential data coherency problems, and must take steps to avoid them.

3.3 Processor architecture

The most significant architectural changes apply to the ARM processors that implement the architectural Security Extensions. Currently these are the:

- ARM1176JZ(F)-S™ processor
- Cortex™-A8 processor
- Cortex-A9 processor
- Cortex-A9 MPCore™ processor

Each of the physical processor cores in these designs provides two virtual cores, one considered Non-secure and the other Secure, and a mechanism to robustly context switch between them, known as monitor mode. The value of the NS bit sent on the main system bus is indirectly derived from the identity of the virtual core that performed the instruction or data access. This enables trivial integration of the virtual processors into the system security mechanism; the Non-secure virtual processor can only access Non-secure system resources, but the Secure virtual processor can see all resources.

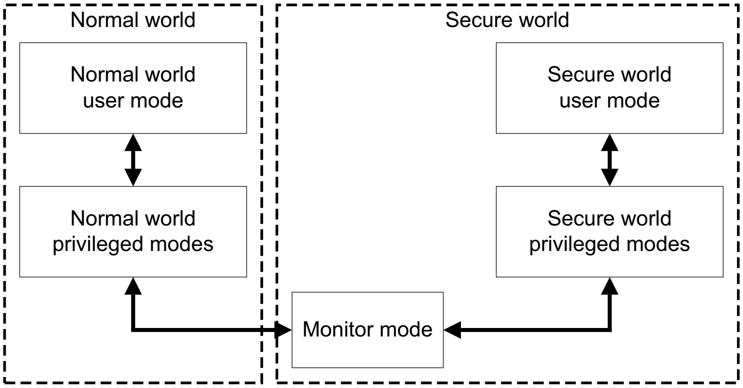


Figure 3-1 : Modes in an ARM core implementing the Security Extensions

3.3.1 Switching worlds

The two virtual processors execute in a time-sliced fashion, context switching through a new core mode called monitor mode when changing the currently running virtual processor.

The mechanisms by which the physical processor can enter monitor mode from the Normal world are tightly controlled, and are all viewed as exceptions to the monitor mode software. The entry to monitor can be triggered by software executing a dedicated instruction, the *Secure Monitor Call* (SMC) instruction, or by a subset of the hardware exception mechanisms. The IRQ, FIQ, external Data Abort, and external Prefetch Abort exceptions can all be configured to cause the processor to switch into monitor mode.

The software that executes within monitor mode is implementation defined, but it generally saves the state of the current world and restores the state of the world being switched to. It then performs a return-from-exception to restart processing in the restored world.

The world in which the processor is executing is indicated by the NS-bit in the Secure Configuration Register (SCR) in CP15, the system control coprocessor, unless the processor is in monitor mode. When in monitor mode, the processor is always executing in the Secure world regardless of the value of the SCR NS-bit, but operations on banked CP15 registers will access Normal world copies if the SCR NS-bit is set to 1.

Note

If Secure world software sets the SCR NS-bit to 1 when the processor is not in monitor mode, the processor will immediately switch to run in the Normal world. This will give less trusted software visibility of the execution of instructions still in the pipeline, and visibility of any data held in processor registers. This can cause a security violation if the instructions, or the data in the registers, is sensitive. For this reason it is recommended that only monitor mode software directly modifies the NS-bit.

Normal world software is not able to access the contents of the SCR.

3.3.2 Securing the level one memory system

The memory infrastructure outside of the core separates the system into two worlds, and a similar partitioning needs to be applied within the core to separate the data used and stored within the components of the level one (L1) memory system.

Memory Management Unit

The major component of the L1 memory system in an ARM applications profile processor is the Memory Management Unit (MMU), which is capable of mapping the virtual address space that is seen by the software running on the processor on to the physical address space that exists outside of the processor. The address translation is managed using a software-controlled translation table, which details which virtual address corresponds to each physical address, and some other attributes about the memory access, such as cacheability and access permissions.

In an ARM core which has an MMU but not the Security Extensions, such as the ARM926EJ-S™ processor, there is a single mapping of virtual address to physical address at any instant in time. Privileged mode code typically rewrites, or points the hardware at a new set of, tables on process context switch to provide multiple independent virtual memory spaces. Within a TrustZone processor the hardware provides two virtual MMUs, one for each virtual processor. This enables each world to have a local set of translation tables, giving them independent control over their virtual address to physical address mappings.

The ARMv6 and ARMv7 L1 translation table descriptor format includes an NS field which is used by the Secure virtual processor to determine the value of the NS-bit to use when accessing the physical memory locations associated with that table descriptor. The Non-secure virtual processor hardware ignores this field, and the memory access is always made with NS=1. This design enables the Secure virtual processor to access either Secure or Non-secure memory.

To enable efficient context switching between worlds the ARM processor implementations may tag entries in the Translation Lookaside Buffers (TLBs), which cache the results of translation table walks, with the identity of the world that performed the walk. This allows Non-secure and Secure entries to co-exist in the TLBs, enabling faster world switching as there is then no need to flush TLB entries.

Note

The presence of the identity of the owning world in the TLB tag is not mandated by the ARM architecture; it is implementation defined. Some processor hardware may flush some or all of the TLB entries on world switch.

Caches

It is a desirable feature of any high performance design to support data of both security states in the caches. This removes the need for a cache flush when switching between worlds, and enables high performance software to communicate over the world boundary. To enable this the L1, and where applicable level two and beyond, processor caches have been extended with an additional tag bit which records the security state of the transaction that accessed the memory.

The content of the caches, with regard to the security state, is dynamic. Any non-locked down cache line can be evicted to make space for new data, regardless of its security state. It is possible for a Secure line load to evict a Non-secure line, and for a Non-secure line load to evict a Secure line.

Example

Putting all of the concepts described above together, Figure 3-2 shows how the L1 memory system of a theoretical ARM processor might handle the state associated with Security Extensions when accessing the memory system.

1. The core processing logic attempts a data load, a data store, or an instruction prefetch. The hardware passes the Virtual Address (VA) and the current world (Non-Secure Table Identifier, or NSTID) to the TLB to enable it to perform address translation.

2. The TLB loads the Physical Address (PA) and the NS-bit associated with the VA and NSTID it was passed, performing a page-table walk and forcing NS=1 if NSTID=1 if necessary. The TLB then passes this information to the cache to perform the actual data or instruction access.

3. The cache attempts the match the PA and the NS-bit from the TLB with the tag of an existing cache line. If this succeeds it will return the data from that cache line, otherwise it will load the cache line from the external memory system.

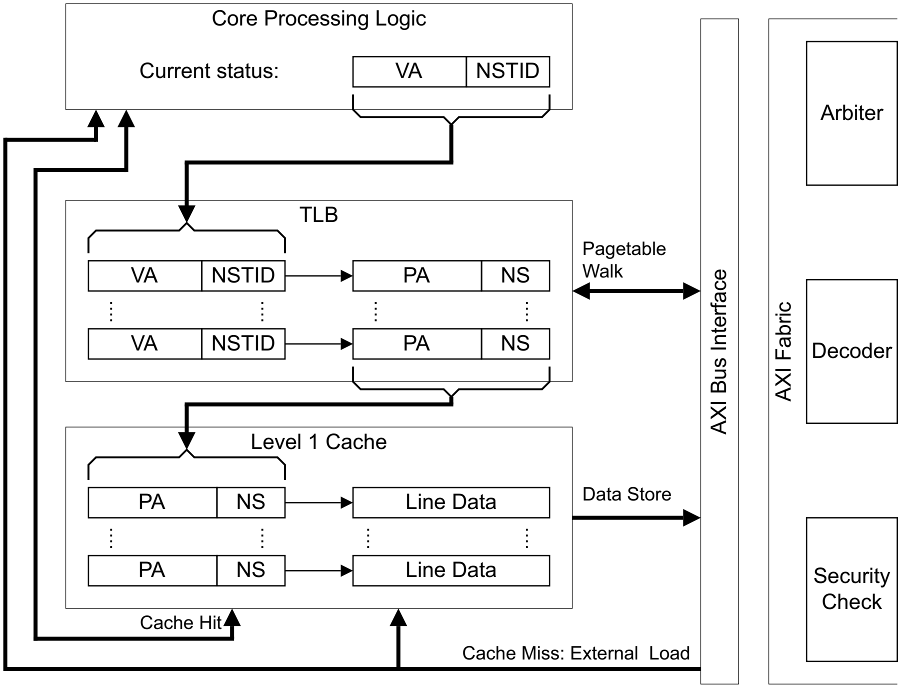


Figure 3-2 : Level one memory system for a theoretical ARM core

In a media application where encrypted audio content is loaded in the Normal world media player, and decrypted in the Secure world, the Secure world software can map the Non-secure memory containing the data belonging to the media player in the Secure world translation tables. This allows the Secure world to directly access the Non-secure cache lines containing the audio content that needs to be decrypted; this type of memory is known as World-shared memory. A Normal world application can therefore pass data to a companion task in the Secure world though any level in the cache hierarchy. This enables a high performance system in comparison to solutions that require cached data to be flushed out of the cache and in to external memory.

Tightly Coupled Memories

The ARM1176JZ(F)-S processor supports Tightly Coupled Memories (TCMs), which are blocks of high performance SRAM that exist at the same level in the memory hierarchy as the L1 cache. There are up to two blocks of TCM present on each of the instruction and data interfaces, depending on the total size of TCM configured at synthesis time. Software can configure each block of the TCM to be made either Secure access only, or Non-secure access only, with independent control of the base addresses of each block.

Accelerator Coherency Port

Some ARM processors, such as the ARM Cortex-A9 MPCore processor, include an optional Accelerator Coherence Port (ACP). The ACP is an AXI slave interface on the processor which enables any peripheral master that is connected to it to access a physical memory map which is coherent with the processor. This allows the external peripheral to access data located within the cache hierarchy of the ARM processor. This technology reduces the number of use cases which need data to be cleaned and/or invalidated out of the ARM processor's caches, improving the performance of software which has to closely share data with an external peripheral such as a DMA controller.

The security state of memory accesses made by the ARM processor memory system on behalf of an external master using the ACP will be same as the security state of the bus transaction accessing the ACP. The value of the **ARPROT[1]** signal read at the ACP interface will be used for reads and the value of the **AWPROT[1]** signal will be used for writes.

It may be possible for an AXI slave to distinguish between a transaction made by the ARM processor on behalf of an internal processing unit and one made on behalf of an ACP transaction. This is dependent on the microarchitecture of the processor and may not be available in all implementations. The Cortex-A9 MPCore processor sets the following AXI signals in accordance with the originator of the memory access:

- **ARIDMx[2]**: Read transaction – low if originator is the ARM processor, high if it is the ACP.
- **AWIDMx[2]**: Write transaction – low if originator is the ARM processor, high if it is the ACP.

This signal can be used by the address decode logic of security sensitive slaves to determine if an access has originated from a trusted master. This technique can, for example, be used to hide part of the ARM processor's physical memory map from other masters which have access to the ACP.

Note

The ACP can be made Secure-access only just like any other AXI slave.

3.3.3 Secure interrupts

The ability to trap IRQ and FIQ directly to the monitor, without intervention of code in either world, allows for the creation of a flexible interrupt model for secure interrupt sources. Once the execution reaches the monitor, the trusted software can route the interrupt request accordingly. When combined with a security aware interrupt controller this allows a design to provide secure interrupt sources which cannot be manipulated by the Normal world software.

The model recommended by ARM is the use of IRQ as a Normal world interrupt source, and FIQ as the Secure world source. IRQ is the most common interrupt source in use in most operating environments, so the use of FIQ as the secure interrupt should mean the fewest modifications to existing software. If the processor is running the correct virtual core when an interrupt occurs there is no switch to the monitor and the interrupt is handled locally in the current world. If the core is in the other world when an interrupt occurs the hardware traps to the monitor, the monitor software causes a context switch and jumps to the restored world, at which point the interrupt is taken.

Note

It is recommended that the monitor always executes with interrupts masked.

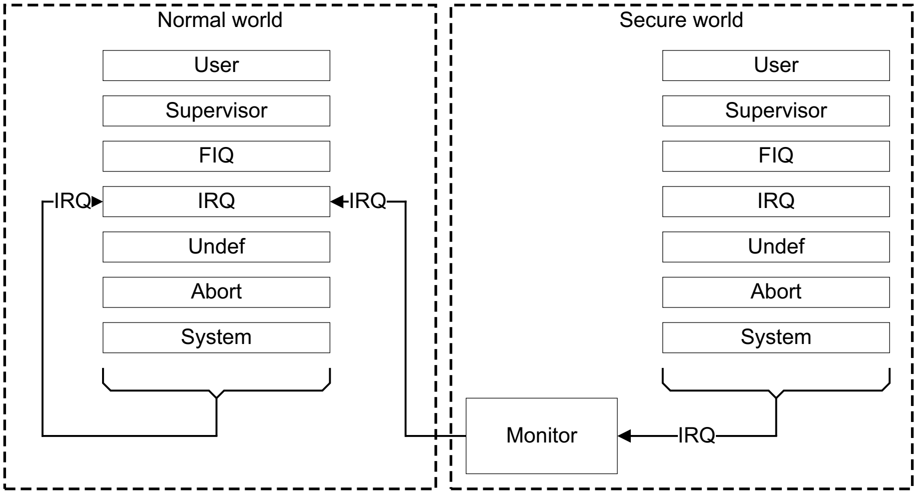


Figure 3-3 : One possible IRQ routing in a design with IRQ configured as a non-secure interrupt

To prevent malicious Normal world software masking sensitive Secure world interrupts the processor hardware includes a configuration register in CP15 which can be used to prevent any Normal world software modifying the F (FIQ mask) and A (external abort mask) bits in the CPSR. This control register can only be accessed by Secure world software. Note that there is no option to prevent the Normal world masking IRQ interrupts.

Processor exception vector tables

To provide the exception behavior described above, a TrustZone-enabled processor implements three sets of exception vector tables. One of these tables is for the Normal world, one is for the Secure world, and the other is for Monitor mode.

The base address of the Secure world table at reset is in accordance with the setting of the **VINITHI** processor input signal; `0x00000000` if it is not asserted, `0xFFFF0000` if it is. The base address of the other tables is undefined, and should be set by software before use.

Unlike previous generations of ARM processors, the location of each of the tables can be moved at run-time. This is achieved by programming the appropriate Vector Base Address Register (VBAR) in CP15.

Note

The use of High vectors can be enabled or disabled at run-time by setting the V bit in the CP15 Control Register. If the V bit is set processor exceptions are always taken from a table starting at `0xFFFF0000`, regardless of the value stored in the VBAR. The value of the V bit is banked, enabling independent configuration of the Secure world and the Normal world vector tables.

The V-bit only applies to the respective Secure world and the Normal world exception tables; the Monitor exception table is always located at the memory address specified in Monitor Vector Base Address Register.

3.3.4 Secure processor configuration

To enable independent execution of code on the virtual CPUs, the hardware strictly manages the configuration options present in CP15. The configuration options that are considered sensitive, or that apply globally to the core, can only be written by Secure world software, although most can be read by Normal world software.

Settings that are not globally sensitive, and as such can be applied locally in each world, are normally banked in the hardware. This gives each world independent control over the settings that would impact their implementation.

Some of the global configuration options which only the Secure world can modify have some impact on the Normal world implementation, in particular in relation to access to some of the low-level hardware features such as cache lockdown or system coprocessors. However, legacy software typically requires minimal or no changes to execute in the Normal world.

3.3.5 Multiprocessor systems with the Security Extensions

The ARM architecture includes support for multiprocessor designs with between one and four processors in a cluster. The processors in the cluster can be configured to execute either in Symmetric Multi-Processing (SMP) mode, or in Asymmetric Multi-Processing (AMP) mode.

When a processor is executing in SMP mode the cluster’s Snoop Control Unit (SCU) will transparently keep data which is shared across the SMP processors coherent[1] in the L1 data cache. When a processor is executing in AMP mode the executing software must manually maintain memory coherency if it is needed.

[1] Only a subset of the memory types supported by the MMU are kept coherent by the SCU. Refer to the appropriate processor Technical Reference Manual for details.

These multiprocessor systems may implement the ARM Security Extensions, giving each processor in the cluster the programmer's model features described earlier in this chapter. The ARM processor which currently implements both the multiprocessor features and the security features is the Cortex-A9 MPCore processor

Note

Multiprocessor systems often include an *Accelerator Coherency Port* which allows an external bus master to access the same physical memory view as the processor cluster. See page 3-10 for further details.

Two worlds per processor

Each of the processors within the multiprocessor cluster has a Normal world and a Secure world. This gives a four processor cluster a total of eight virtual processors, each with independent control over their MMU configuration.

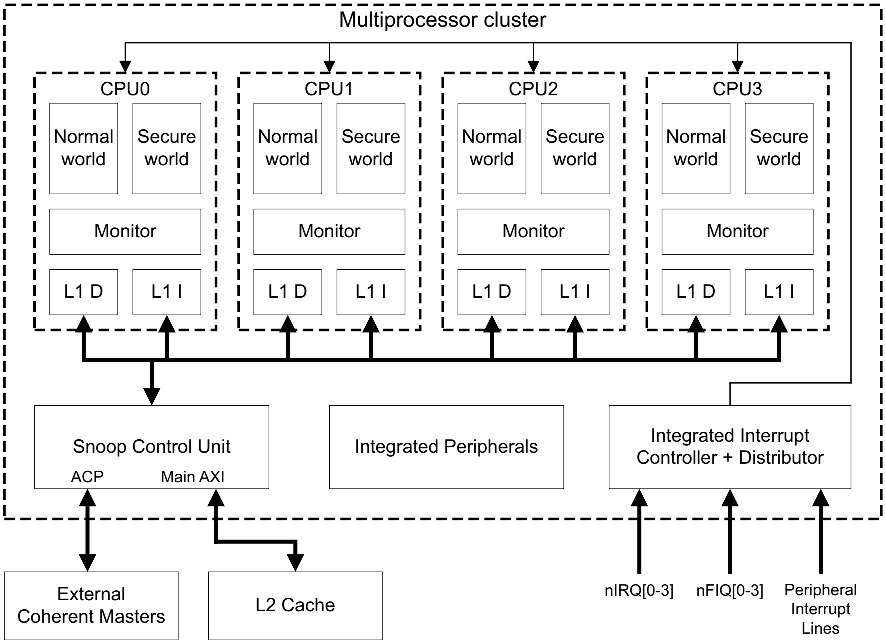


Figure 3-4 ARM multiprocessor cluster

Any number of the processors in the cluster may be in the Secure world at any point in time, and the processors can transition between the worlds independently of other processors in the cluster. A specific software implementation may choose to restrict the concurrent execution of Secure world software to reduce the security risks associated with complex software designs.

Note

The potential impact of multiprocessing on Secure world software design is discussed in *Secure software and multiprocessor systems* on page 5-13.

As described in *Caches* section on page 3-8, each of the cache lines in the cluster stores the security state of the data it contains as part of its tag. This enables the concurrent storage of Secure and Non-secure data within the L1 processor data caches when the processors within the cluster are executing in SMP mode. The coherency hardware uses the whole cache tag when performing coherency operations, allowing it to keep both Secure and Non-secure data coherent simultaneously.

Snoop Control Unit configuration

The SCU includes a number of configuration registers which determine the configuration of the SCU itself, the configuration of each of the ARM processors in the cluster, and the accessibility of the processor-local timers to Non-secure memory transactions.

- The *SCU Access Control Register* determines which processors in the cluster can reprogram the SCU's configuration registers.
- The *SCU Secure Access Control Register* determines if Non-secure accesses can reprogram the SCU configuration registers or access the processor-local timers.

Interrupt handling

The Cortex A-profile multiprocessor systems include an integrated interrupt controller based on the same technology as the *PrimeCell Generic Interrupt Controller (PL390)*, described on page 4-6. This interrupt controller provides a flexible interrupt model which is capable of distributing prioritized interrupts across the multiprocessor cluster, interrupting lower priority interrupt handlers which are already executing if a higher priority interrupt is received.

In a multiprocessor system which also implements the Security Extensions this interrupt controller is TrustZone-aware. This allows it to manage Secure and Non-secure interrupts and prevent Non-secure memory accesses from reading or modifying the configuration of a Secure interrupt.

An interrupt managed by the integrated interrupt controller can be configured as a Secure interrupt by programming the appropriate bits in the *Interrupt Security Register*. Once and interrupt has been made Secure, no Non-secure access can modify its configuration.

All interrupts managed by the integrated interrupt controller are assigned a priority to determine whether they are allowed to interrupt an exception which is already being handled by the ARM processor. The hardware ensures that a lower priority interrupt will wait until a higher priority interrupt has been cleared before it issued to the processor. The priority space is partitioned to ensure that Secure interrupts can always be configured with a higher priority than the Non-secure interrupts. Assigning the Secure world a high priority interrupt can be used to prevent the Non-secure world performing a denial-of-service attack against the Secure world using interrupts.

The integrated interrupt controller can support the model described earlier in this chapter, causing Secure interrupts it controls to generate an FIQ exception and Non-secure interrupts it controls to generate an IRQ exception. In this case all interrupts are managed by the integrated interrupt controller, and no direct interrupt generation from an external interrupt controller is possible. The integrated interrupt controller can also support a number of legacy configurations which cause the FIQ and/or the IRQ exceptions to be generated by an external interrupt trigger, bypassing the integrated interrupt controller completely.

It is possible to independently configure the legacy interrupt generation for FIQ and IRQ exceptions.

- If legacy mode is enabled only for FIQ exceptions then the integrated controller will route both Secure and Non-secure interrupts it controls to the IRQ exception vector.
- If legacy mode is enabled only for IRQ exceptions then the integrated controller becomes unable to generate exceptions for Non-secure interrupts, and Secure interrupts will be routed to the FIQ exception vector.
- If legacy mode is enabled for both FIQ and IRQ exceptions then the integrated interrupt controller is bypassed completely.

Note

If they are used in a design, the legacy interrupt input signals to the processor cluster are typically generated by one or more external interrupt controllers. These external devices can be secured using the same methods as any other external AXI or APB slaves in a TrustZone system.

3.4 Debug architecture

The final part of the system infrastructure that integrates with the Security Extensions is the debug provision. The debug solutions provided by ARM split into two parts: processor debug components and system debug components.

3.4.1 Processor debug control

ARM processors prior to the introduction of the Security Extensions have included a single debug control signal that globally enables or disables debugger access to the processor. Deploying security sensitive software alongside a rich operating environment in these designs implies that debug must be globally disabled, otherwise you risk significant threat from simple hardware attacks.

As operating environments become larger and more complex this starts to become a far from ideal situation, especially when there is a large community of developers who desire to develop software for the rich operating environment. These people need to debug their applications, typically after the device is in the field and the security software has been enabled.

The TrustZone debug extensions separate the debug access control into independently configurable views of each of the following aspects:

- Secure privileged invasive (JTAG) debug
- Secure privileged non-invasive (trace) debug
- Secure user invasive debug
- Secure user non-invasive debug

The Secure privileged debug access is controlled by two input signals to the core, **SPIDEN** (invasive) and **SPNIDEN** (non-invasive). The Secure user mode debug access is controlled by two bits, **SUIDEN** (invasive) and **SUNIDEN** (non-invasive) in a Secure privileged access only CP15 register. These settings enable a TrustZone processor to give control over the debug visibility once the device is deployed. It is, for example, possible to give full Normal world debug visibility while also preventing all Secure world debug.

Note

ARM processors also include global debug enable input signals: **DBGEN** and, on cores which implement the ARMv7 architecture, **NIDEN**. These signals can be used to disable all debug visibility of a core, including Normal world debug.

- ARMv6: **DBGEN** – global invasive and non-invasive debug enable.
- ARMv7: **DBGEN** – global invasive debug enable.
- ARMv7: **NIDEN** – global non-invasive debug enable.

Multiprocessor debug control

In ARM processors implementing the multiprocessor extensions, described in *Multiprocessor systems with the Security Extensions* section on page 3-13, each processor in the cluster is provided with independent **DBGEN**, **NIDEN**, **SPIDEN** and **SPNIDEN** input signals. This allows a subset of the processors in the cluster to be debugged in a subset of the possible ways.

Note

System designers must be aware that the SMP data coherency hardware may allow a processor with invasive debug enabled to modify the data being used by a processor with invasive debug disabled.

Performance analysis

To enable low level benchmarking of code, the ARMv6 and ARMv7 applications-grade processors include a Performance Monitor in CP15. This hardware unit can be used for timing code execution and counting processor events which can occur at run-time, such as cache line evictions.

To prevent the Performance Monitor being used in attacks against the Secure world software, a secure CP15 configuration option can be used to prevent Normal world and user mode access to these counters.

3.4.2 System debug control

The ARM system debug solution is the ARM CoreSight™ on-chip debug and trace technology. It provides a debug and trace solution for the entire SoC, enabling debug of multiple processors and other system components. The CoreSight debug infrastructure can be accessed both from off-device tools and from on-device components.

The parts of the CoreSight infrastructure that are accessible from on-SoC hardware and software are implemented as APB peripherals. To reduce the number of components needed the CoreSight peripherals are designed not to use the standard per-peripheral protection mechanisms provided by an AXI-to-APB bridge; CoreSight components should be accessible to Non-secure memory transactions.

As an alternative to the protection provided by the AXI-to-APB bridge, the CoreSight components include a number of control signals which are used to enable or disable Secure debug. These signals are known as the CoreSight authentication interface, and include **SPIDEN**, **SPNIDEN** and **DBGEN** signals, which perform a similar role to the signals of the same name described for the processor core.

If external debug hardware, or on-target Normal world software, attempts to set a system breakpoint on a secure address when **SPIDEN** is deasserted, the CoreSight hardware will fail to create a breakpoint. For instrumentation solutions, Secure trace information will simply be discarded by the peripheral if **SPNIDEN** is not asserted.

Note

A consequence of the debug security architecture is that Normal world software may be able to directly affect or monitor the Secure world execution in a system when **SPIDEN** or **SPNIDEN** are asserted.

Secure debug should therefore only be enabled when the device is located in a trusted environment.

Chapter 4
TrustZone Hardware Library

This chapter outlines the TrustZone-enabled IP that is available from ARM.

This chapter includes the following sections:

4.1 System IP

This section outlines some of the system IP that is available from ARM which has in-built support for the Security Extensions.

Note

Any APB peripheral can be secured using the Security Extensions if you use an appropriate AXI-to-APB bridge.

4.1.1 PrimeCell® High-Performance Matrix - PL301

The primary component in implementing the system-wide isolation is the AMBA3 AXI compliant bus matrix, which connects all of the system components together. ARM provides this in the PrimeCell High-Performance Matrix product, PL301.

To support the requirements placed on modern SoC infrastructure, the AXI bus generator is complemented by a range of support components. Register slices for timing isolation, width scaling downsizers for reducing bus width to low bandwidth SoC regions, and synchronous or asynchronous bridges for linking clock domains are all available:

- PrimeCell Infrastructure AMBA3 AXI Register Slice - BP130
- PrimeCell Infrastructure AMBA3 AXI Downsizer - BP131
- PrimeCell Infrastructure AMBA3 AXI to AXI Bridges - BP132-4

PrimeCell Infrastructure AMBA3 AXI to APB Bridge - BP135

In a typical ARM system most peripherals are connected to the APB bus. APB is a simpler, lower power, bus than the main AXI bus. The APB protocol does not carry the bits related to the TrustZone security state of the bus transactions. This makes it possible for existing peripheral designs to used on the AMBA3 APB bus, and places responsibility for managing the security state onto the AXI-to-APB Bridge that provides the interface between the high-speed AXI domain and the low-power APB domain.

Each AXI-to-APB bridge provides an AXI slave interface and can mediate accesses for up to 16 peripherals on its local APB bus. The bridge contains address decode logic that generates the APB peripheral select based on the incoming AXI transaction. The bridge includes a single **TZPCDECPROT** input signal for each peripheral that is located on the bus. This signal is used to determine if the peripheral is configured as Secure or Non-secure; the bridge will reject Non-secure transactions to Secure peripheral address ranges.

These bridge input signals can be tied persistently at synthesis time or can be dynamically controlled via a trusted peripheral, such as the TrustZone Protection Controller (TZPC), to allow dynamic switching of security state at run-time.

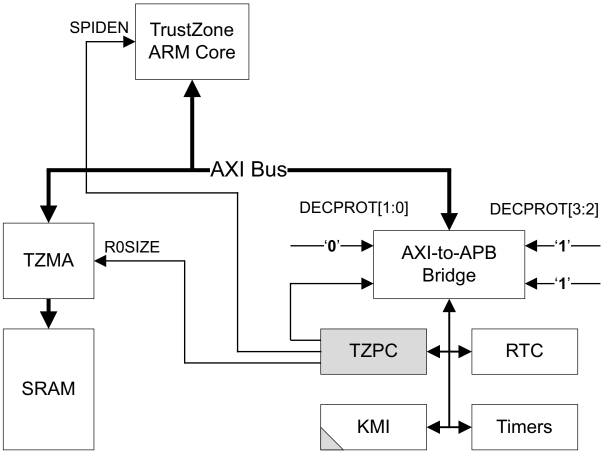


Figure 4-1 : Using a TZPC to control security signals on a SoC

Figure 4-1 shows an AXI-to-APB bridge controlling 4 peripherals. The TZPC is configured as always Secure, the Timers and Real-Time Clock (RTC) as always Non-secure, and the Keyboard and Mouse Interface (KMI) has a programmable security state under software control. Secure world software can program the TZPC at run-time to change the signal input to the AXI-to-APB bridge to switch the KMI from Secure to Non-secure or visa versa.

——— **Note** ———

Shaded blocks in the diagram represent Secure peripherals, and blocks with a shaded corner represent peripherals which may be switched from Secure to Non-secure.

This design allows peripherals to exist in the Normal world for most of the time, but lets them be temporarily switched into the Secure world for a short duration. The KMI peripheral is a good example; it will commonly be used as the general purpose Normal world keypad, but might be switched to become a Secure input device for a short period to allow the entry of a user password in a safe environment. As this figure also shows, the addition of the TZPC allows other signals on the SoC to be controlled dynamically, for example the **SPIDEN** debug control input to the ARM core.

PrimeCell Infrastructure AMBA3 AXI to AMBA 2 AHB Bridge - BP136-7

The AXI-to-AHB and AHB-to-AXI bridges allow two subsystems which implement the AMBA3 and AMBA2 specifications to be joined together. A separate bridge is needed for each direction of communication; the AXI-to-AHB bridge allows an AXI transaction to master on the AHB bus, and the AHB-to-AXI bridge allows an AHB transaction to master on the AXI bus.

The AHB bus does not provide any mechanism to carry the security state of the memory transactions that pass across it, so all of the security implementation must be managed in the bridge itself.

- The AXI-to-AHB bridge allows the whole of the AHB slave domain to be made either Secure or Non-secure.
- The AHB-to-AXI bridge allows the whole of the AHB master domain to be made either Secure or Non-secure.

If a design requires a mixture of Secure and Non-secure AHB masters and slaves it is recommended that Secure components and Non-secure components are never placed behind the same bridge.

——— **Note** ———

The security within the AHB domain is managed entirely by the AHB configuration and is outside of the scope of the Security Extensions.

———————————

4.1.2 PrimeCell Level 2 Cache Controller - PL310

As processor clock frequencies rise it is increasingly desirable to include a level 2 cache between the core and the external memory system. This reduces the number of processor pipeline stalls experienced because of delays accessing external memory, enabling significantly faster performance in some applications, and typically reducing power consumption. Each of the caches in a TrustZone system needs to tag each cache line with the security state of the data that it contains, enabling concurrent storage of data from both worlds.

The ARM1176JZ(F)-S processor and the Cortex-A9 processors can make use of a separate level 2 cache controller, which must implement appropriate tagging of the security state. The ARM high performance PrimeCell Level 2 Cache Controller can be used for this purpose.

——— **Note** ———

The Cortex-A8 processor includes an integrated L2 cache controller.

———————————

4.1.3 PrimeCell DMA Controller - PL330

Using a processor to move data around the system can be an inefficient use of power and processor execution time. For this reason many systems include a Direct Memory Access Controller (DMAC), which is a dedicated engine for moving data around the physical memory system.

The PrimeCell DMA Controller is a multi-channel AXI engine, with a microcoded job description to enable transfers of complex structures. The DMAC can support concurrent Secure and Non-secure channels, each with independent interrupt events and controlled by a dedicated APB interface. A Non-secure transaction trying to program a DMA transfer to or from Secure memory will result in the DMA transfer failing.

4.1.4 PrimeCell TrustZone Address Space Controller - PL380

The TrustZone Address Space Controller (TZASC) is an AXI component which partitions its slave address range into a number of memory regions. The TZASC can be programmed by Secure software to configure these regions as Secure or Non-secure, and will reject Non-secure transactions to a region that is configured as Secure.

The number of memory regions, and the bus widths of the TZASC AXI interfaces, are configurable when the design is synthesized.

The main reason to use a TZASC is to partition a single AXI slave, such as an off-SoC DRAM, into multiple security domains. Off-SoC RAM is a good example as memory devices have significant cost associated with them due to extra pin-out, printed circuit board area, and the cost of the memory itself. It is therefore desirable for a system to partition a single external memory so that it can contain both Secure and Non-secure regions; this is typically less expensive than placing two smaller memory devices.

The ARM AXI Dynamic Memory Controller (DMC) family is a range of high performance controllers which do not internally support the creation of Secure and Non-secure partitions. To enable security partitions to be created a TZASC can be placed between the DMC and the on-SoC masters that need to access it. The TZASC is designed to work with dynamic memory and allows burst accesses to travel through it with minimal impact on memory latencies.

Note

The TZASC can only be used for partitioning memory mapped devices; in particular it cannot be used for partitioning block-based devices, such as NAND flash.

4.1.5 PrimeCell Infrastructure AMBA3 AXI TrustZone Memory Adapter - BP141

The TrustZone Memory Adapter (TZMA) enables a design to secure a region within an on-SoC static memory such as a ROM or a SRAM. It is typically less expensive to place a single large memory device and partition it into Secure and Non-secure regions than it is to provide separate dedicated memories for each world. The TZMA allows a single static memory of up to 2MB to be partitioned into two regions where the lower part is Secure, and the upper part Non-secure.

The location of the partition between the Secure and Non-secure regions is always a multiple of 4KB and is controlled via the **R0SIZE** input signals to the TZMA. These signals may be dynamically configured by connecting the signals to the **TZPCR0SIZE** output from a TZPC peripheral, or can be tied off at synthesis time to partition the memory in a fixed fashion.

The TZMA cannot be used for partitioning dynamic memories, or memories that require more than one Secure region; for these designs the TZASC must be used.

4.1.6 PrimeCell Generic Interrupt Controller - PL390

To support the robust management of Secure and Non-secure interrupts, the underlying interrupt controller must prevent the Normal world modifying the configuration of Secure world interrupt sources. This means that a single interrupt controller must support TrustZone technology using internal partitioning, or two interrupt controllers must be placed in the system.

The Generic Interrupt Controller (GIC) is a single hardware device that supports both Secure and Non-secure prioritized interrupt sources. Attempts by Normal world software to modify the configuration of an interrupt line configured as a Secure source will be prevented by the GIC hardware. Additionally, Non-secure software can only configure interrupts in the lower half of the priority range, preventing denial-of-service attacks.

Note

The ARM Cortex-A9 MPCore incorporates its own interrupt controller with the same programmer's model as the GIC, and therefore does not require an external interrupt controller. More details about the integrated interrupt controller can be found in the section called *Interrupt handling* on page 3-15.

4.1.7 PrimeCell Infrastructure AMBA3 TrustZone Protection Controller - BP147

As described in the AXI-to-APB bridge section, the TrustZone Protection Controller (TZPC) is a configurable signal control block which can be placed on the APB bus to supply control signals to other components on the SoC. The TZPC includes three general purpose registers, **TZPCDECPROT{2:0}**, which can each control 8 signals within the SoC. It also includes a single register, **TZPCR0SIZE**, that can be used to provide the partition location control signals for the TrustZone Memory Adapter.

The power-on state of the TZPC is for all of the TZPCDECROT register bits to be set to 0, which is Secure, and the TZPCR0SIZE register to be set to `0x200`, which makes the entire 2MB memory range supported by the TZMA Secure. Boot code can relax the security settings, making components Non-secure as needed.

4.2 Processor IP

There are a number of processors which implement an AMBA3 AXI memory interface, but not all implement the same feature set. This section outlines the major features of these processors, and how they might be used in a TrustZone system.

This section also introduces the ARM SecurCore family of smartcard processors, which could be used alongside a TrustZone-enabled SoC to provide higher levels of physical security.

4.2.1 ARM1176JZ(F)-S processor

The ARM1176JZ(F)-S processor was the first of the TrustZone processors. It implements the ARMv6Z architecture and provides an 8-stage single-issue integer pipeline capable of 320MHz in 90nm process using the ARM Artisan® Metro standard cell libraries. It consumes 1mm^2 die area, excluding caches, when implemented with this silicon process.

Interesting features related to TrustZone System design:

- Optional Tightly Coupled Memories (TCMs).
- Optional external level 2 cache controller (PL310).

4.2.2 Cortex-A8 processor

The Cortex-A8 processor was the first of the ARMv7 A-profile processors. It implements a 13-stage dual-issue integer pipeline and an additional 10-stage NEON media SIMD pipeline. It is capable of 650MHz in 65nm LP process using the ARM Artisan® Advantage libraries, consuming less than 3mm^2 die area, excluding NEON and cache RAMs.

Interesting features related to TrustZone system design:

- No TCMs.
- Integrated level 2 cache controller.

4.2.3 Cortex-A9 processor and Cortex-A9 MPCore processor

The Cortex-A9 is an ARMv7 A-profile processor, available as a unicore and as a 1-4 way multicore implementation. It implements an out-of-order, multi-issue superscalar pipeline, with an optional Vector Floating Point (VFP) or NEON pipeline.

Interesting features related to TrustZone system design:

- First multi-processing platform to implement the ARM Security Extensions, giving a total of eight virtual processors in a four-core design.
- No TCMs.
- Designed to work with the PrimeCell Level 2 Cache Controller (PL310).

  Designed to work with the Generic Interrupt Controller (PL390).
  This is only a requirement for the uniprocessor variant of the Cortex-A9; the multi-processor MPCore version includes a built-in interrupt controller.

4.2.4 ARM1156T2(F)-S™ processor

The ARM1156T2(F)-S processor is an ARMv6 processor with a Memory Protection Unit (MPU), designed for embedded applications. The processor itself does not implement the Security Extensions, but it does implement an AMBA3 AXI bus interface which enables it to be placed directly within a TrustZone-enabled SoC design without additional logic or bridges.

In most designs the ARM1156T2(F)-S will be statically tied to either generate Secure memory transactions or Non-secure memory transactions, enabling the ARM1156T2(F)-S to act as an asymmetric processing engine alongside the main applications processor as either parts of the Secure world, or as part of the Normal world.

4.2.5 Cortex-R4 processor

The Cortex-R4 processor is an ARMv7 R-profile processor for embedded applications. Similarly to the ARM1156T2(F)-S processor, the Cortex-R4 processor does not implement the Security Extensions, but it does implement an AMBA3 AXI bus interface which enables it to be placed directly within a TrustZone-enabled SoC design without additional logic or bridges.

4.2.6 SecurCore smartcard processors

The ARM SecurCore family provides a range of processor IP designed for use in smartcards and tamper-resistant integrated circuit deployments. For security applications that need a higher level of physical security than a TrustZone system can provide, a SecurCore-based smartcard can be used alongside a TrustZone system. The TrustZone system can provide functionality that the smartcard cannot, such as secure user interfaces and software that can handle high performance, less sensitive, security tasks.

Summary of processor features:

- SC100: Supports ARM and Thumb® instruction sets.
- SC200: Supports ARM and Thumb instruction sets and integrated Jazelle™ technology for accelerating Java Card 2.x applications.
- SC300: Based on the Cortex™ -M3, SC300 supports multiple technologies, including the Thumb®-2 instruction set, to reduce memory requirements and increase software performance.

4.3 Reuse of AMBA2 AHB IP

A system designer may have a need to reuse legacy AMBA2 AHB masters and slaves within a design that also uses AMBA3 and the Security Extensions. These legacy devices may include non-processor masters, such as DMA controllers, and complex sub-systems including masters such as the ARM926EJ-S processor.

4.3.1 Reuse of AHB masters

In these designs any AHB masters must connect to the AXI domain by passing their transactions through an AHB-to-AXI bridge, described on page 4-4. These bridges implement a simple mapping of the AXI NS-bits, based on a bridge configuration signal. This allows the entire AHB domain behind the bridge to be made Secure or Non-secure from the AXI domain's point of view.

If only a sub-set of the AHB masters behind a single bridge need to be available as Secure masters in the AXI domain, additional custom logic is needed to generate the configuration input to the bridge that determines the NS-bit, based on the AHB master ID.

Note

It is only possible to directly address 4GB of physical memory per AHB master. It is not possible to address the full 8GB of the Secure and Non-secure physical address space in one AHB master's memory map.

4.3.2 Reuse of AHB slaves

ARM recommends that separate AHB buses are used to contain the Secure and the Non-secure AHB slaves. This enables use of the single NS-bit control signal provided by the AXI-to-AHB bridge; the bridge will reject any transactions which have invalid security permissions.

In situations where this is not possible, a TrustZone Address Space Controller can be placed before the AXI-to-AHB bridge, enabling address-based checks of the NS-bit to be performed.

Chapter 5
TrustZone Software Architecture

This chapter looks at some of the possible software architectures that make use of the ARM Security Extensions.

This chapter includes the following sections:

5.1 Software overview

The implementation of a Secure world in the SoC hardware needs some secure software to run within it and to make use of the sensitive assets stored there.

The Security Extensions are an open component of the ARM architecture, so any developer can create a custom Secure world software environment to meet their requirements. This section presents some of the possibilities that a software architect might want to consider when designing a Secure world software stack.

Note

Each of the software options outlined here places a different set of requirements on the Secure world hardware. For example, a design which has a standalone pre-emptive Secure world operating system will require a secure timer and a security-aware interrupt controller.

5.1.1 Secure world processing resources

The overall structure of the software architecture will be heavily influenced by the nature of the available Secure world processing resource. A system may provide a TrustZone-enabled core, such as the ARM1176JZ(F)-S processor, or may provide a dedicated processor for the Secure world, such as a Cortex-R4 processor.

The design with two physical processors is a classical embedded design, and is not heavily impacted by the addition of the Security Extensions. The software running on the Secure world processor must be self contained and provide its own local operating environment.

It is expected that most designs will choose to use a TrustZone-enabled processor. This typically gives the Secure world higher software performance and requires less silicon area than a dedicated security processor. It is this case that we will focus on for the remainder of this chapter.

5.1.2 Software architecture

There are many possible software architectures which a Secure world software stack on a TrustZone-enabled processor core could implement. The most complex is a dedicated Secure world operating system: the simplest is a synchronous library of code placed in the Secure world. There are many intermediate options between these two extremes.

Secure operating system

A dedicated operating system in the Secure world is a complex, but powerful, design. It can simulate concurrent execution of multiple independent Secure world applications, run-time download of new security applications, and Secure world tasks that are completely independent of the Normal world environment.

The most extreme version of these designs closely resembles the software stacks that would be seen in a SoC with two separate physical processors in an Asymmetric Multi-Processing (AMP) configuration. The software running on each virtual processor is a standalone operating system, and each world uses hardware interrupts to preempt the currently running world and acquire processor time.

A tightly integrated design, which uses a communications protocol that associates Secure world tasks with the Normal world thread that requested them, can provide many of the benefits of a Symmetric Multi-Processing (SMP) design. In these designs a Secure world application could, for example, inherit the priority of the Normal world task that it is assisting. This would enable some form of soft real-time response for media applications.

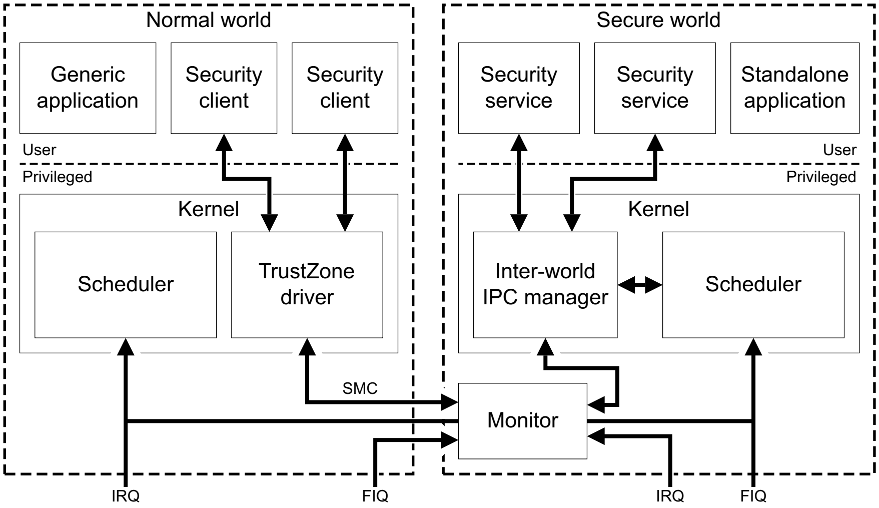


Figure 5-1 : A possible architecture with an independent Secure world OS

One of the advantages of a design based on operating system principles is the use of the processor MMU to separate the Secure world memory space into multiple user-space sandboxes. Provided that the Secure world kernel software is correctly implemented, security tasks from independent stakeholders can execute at the same time without needing to trust each other. The kernel design can enforce the logical isolation of secure tasks from each other, preventing one secure task from tampering with the memory space of another.

Synchronous library

Many use cases do not need the complexity of a Secure world operating system. A simple library of code in the Secure world which can handle one task at a time is sufficient for many applications. This code library is entirely scheduled and managed using software calls from the Normal world operating system.

The Secure world in these systems is a slave to the Normal world and cannot operate independently, but can consequently have a much lower level of complexity.

Intermediate options

There is a range of options that lies between these two extremes. For example, a Secure world multi-tasking operating system may be designed to have no dedicated interrupt source, and as such could be provided with a virtual interrupt by the Normal world. This design would be vulnerable to a denial-of-service attack if the Normal world operating system stopped providing the virtual interrupt, but for many cases this is not a problematic attack. Alternatively, the MMU could be used to statically separate different components of an otherwise synchronous Secure world library.

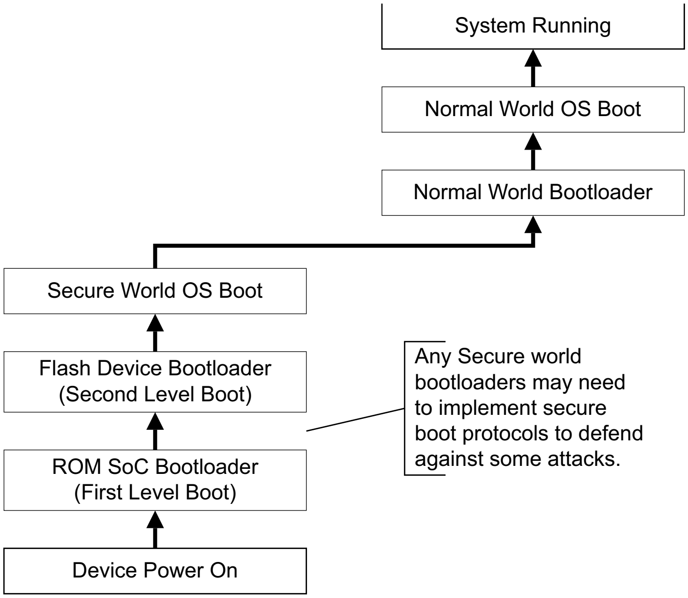
5.2 Booting a secure system

One of the critical points during the lifetime of a secure system is at boot time. Many attackers attempt to break the software while the device is powered down, performing an attack that, for example, replaces the Secure world software image in flash with one that has been tampered with. If a system boots an image from flash, without first checking that is it authentic, the system is vulnerable.

One of the principles applied here is the generation of a chain of trust for all Secure world software, and potentially Normal world software, established from a root of trust that cannot easily be tampered with. This is known as a secure boot sequence. See *Secure boot* on page 5-6.

5.2.1 Boot sequence

A TrustZone-enabled processor starts in the Secure world when it is powered on. This enables any sensitive security checks to run before the Normal world software has an opportunity to modify any aspect of the system.



Any Secure world bootloaders may need to implement secure boot protocols to defend against some attacks.

Figure 5-2 : A typical boot sequence of a TrustZone-enabled processor

After power-on most SoC designs will start executing a ROM-based bootloader which is responsible for initializing critical peripherals such as memory controllers, before switching to a device bootloader located in external non-volatile storage such as flash memory. The boot sequence will then progress through the Secure world operating environment initialization stages, before passing control to the Normal world bootloader. This will progress to starting the Normal world operating system, at which point the system can be considered running.

System control coprocessor lockdown

Systems that want an additional level of protection can use a signal input into the processor core to lock-down some of the critical Secure world configuration options in CP15. Asserting the **CP15SDISABLE** processor input signal will cause some of the Secure world CP15 settings to become unmodifiable, even if the modification is attempted by Secure world privileged software.

It is expected that designs using **CP15SDISABLE** will configure the sensitive settings during the boot process and assert the signal before passing control to the Normal world software.

Note that a system must boot with **CP15SDISABLE** set low to enable Secure world boot code to configure the CP15 registers will appropriate settings. The method used to to change the signal should only be available to the Secure world, and other protections may be suitable, such as using a latching signal generator which can only be reset to a low state by resetting the device.

5.2.2 Secure boot

A secure boot scheme adds cryptographic checks to each stage of the Secure world boot process. This process aims to assert the integrity of all of the Secure world software images that are executed, preventing any unauthorized or maliciously modified software from running.

Cryptographic signature protocol

The most logical cryptographic protocol to apply is one based on a public-key signature algorithm, such as RSA-PSS (Rivest, Shamir and Adleman - Probabilistic Signature Scheme).

In these protocols a trusted vendor uses their Private Key (PrK) to generate a signature of the code that they want to deploy, and pushes this to the device alongside the software binary. The device contains the Public Key (PuK) of the vendor, which can be used to verify that the binary has not been modified and that it was provided by the trusted vendor in question.

The PuK does not need to be kept confidential, but it does need to be stored within the device in a manner which means it cannot be replaced by a PuK that belongs to an attacker.

Chain of trust

The secure boot process implements a chain of trust. Starting with an implicitly trusted component, every other component can be authenticated before being executed. The ownership of the chain can change at each stage - a PuK belonging to the device OEM might be used to authenticate the first bootloader, but the Secure world OS binary might include a secondary PuK that is used to authenticate the applications that it loads.

Unless a design can discount hardware shack attacks the foundations of the secure boot process, known as the root of trust, must be located in the on-SoC ROM. The SoC ROM is the only component in the system that cannot be trivially modified or replaced by simple reprogramming attacks.

Storage of the PuK for the root of trust can be problematic; embedding it in the on-SoC ROM implies that all devices use the same PuK. This makes them vulnerable to class-break attacks if the PrK is stolen or successfully reverse-engineered. On-SoC One-Time-Programmable (OTP) hardware, such as poly-silicon fuses, can be used to store unique values in each SoC during device manufacture. This enables a number of different PuK values to be stored in a single class of devices, reducing risk of class break attacks.

Note

OTP memory can consume considerable silicon area, so the number of bits are that available is typically limited. A RSA PuK is over 1024-bits long, which is typically too large to fit in the available OTP storage. However, as the PuK is not confidential it can be stored in off-SoC storage, provided that a cryptographic hash of the PuK is stored on-SoC in the OTP. The hash is much smaller than the PuK itself (256-bits for a SHA256 hash), and can be used to authenticate the value of the PuK at run-time.

On-SoC Secure world or Off-SoC Secure world

The simplest defense against shack attacks is to keep any Secure world resource execution located in on-SoC memory locations. If the code and data is never exposed outside of the SoC package it becomes significantly more difficult to snoop or modify data values; a physical attack on the SoC package is much harder than connecting a logic probe to a PCB track or a package pin.

The secure boot code is generally responsible for loading code into the on-SoC memory, and it is critical to correctly order the authentication to avoid introducing a window of opportunity for an attacker. Assuming the running code and required cryptographic hashes are already in safe on-SoC memory, the binary or PuK being verified should be copied to a secure location before being authenticated using cryptographic methods. A design that authenticates an image, and then copies it into the safe memory location risks attack. The attacker can modify the image in the short window between the check completing and the copy taking place.

5.3 Monitor mode software

The role of the monitor mode software in a design is to provide a robust gatekeeper which manages the switches between the Secure and Non-secure processor states. In most designs its functionality will be similar to a traditional operating system context switch, ensuring that state of the world that the processor is leaving is safely saved, and the state of the world the processor is switching to is correctly restored.

Normal world entry to monitor mode is tightly controlled. It is only possible via the following exceptions: an interrupt, an external abort, or an explicit call via an SMC instruction. The Secure world entry to the monitor mode is a little more flexible, and can be achieved by directly writing to CPSR, in addition to the exception mechanisms available to the Normal world.

The monitor is a security critical component, as it provides the interface between the two worlds. For robustness reasons it is suggested that the monitor code executes with interrupts disabled; writing a re-entrant monitor would add complexity and is unlikely to provide significant benefits over a simpler design.

5.3.1 Context switching

As mentioned previously, the primary role of the monitor is to context switch resources that are needed in both worlds. Any secure state saved by the monitor should be saved into a region of Secure memory, so that the Normal world cannot tamper with it.

Exactly what needs to be saved and restored for each switch depends on the hardware design, and the software model used for inter-world communications. The list typically includes:

- All general purpose ARM registers.
- Any coprocessor registers, such as NEON or VFP.

  Note: Only required if coprocessor is used in both worlds.
- Any world-dependant processor configuration state in CP15.

Hardware exceptions: IRQ, FIQ, external abort

When the processor hardware is configured to trap exceptions (IRQ, FIQ and external abort) to monitor, the state of the interrupted context is arbitrary.

This means a full context switch of all state is required, unless some can be lazily context switched. See *Lazy context switching* on page 5-10 for more details.

Software exception: SMC

In many circumstances a design will use an SMC instruction as an simple inter-world yield, which causes a full context switch in a similar manner to the hardware exceptions described above. However, in some circumstances it is beneficial for an SMC driven world switch to carry a message payload in some of the processor registers. In these circumstances a full context switch is not desirable.

Efficient software communications protocols can be built between the two worlds using SMC initiated context switches and World-shared memory.

Lazy context switching

Some of the hardware coprocessors connected to the ARM coprocessor interface, such as the ARM VFP and NEON execution units, can support a mechanism called lazy context switching. This allows the context of the coprocessor to be saved only when necessary, rather than on every operating system context switch or TrustZone world switch. The state associated with the VFP and NEON units can be quite large, so lazy context switching can result in a significant decrease in average switch overheads.

To implement lazy context switching in a TrustZone system the Secure world can block Normal world access to each of the coprocessor interfaces using bit settings in the CP15 Non-Secure Access Control Register (NSACR). When Normal world or user-mode software tries to use a coprocessor configured as Secure in NSACR an undefined instruction exception will be raised. The exception must be trapped by the Normal world kernel, and a handler must issue an SMC to the monitor to request the required coprocessor context switch. As soon as the coprocessor context has been swapped, the monitor can make the coprocessor accessible to Non-secure software using the NSACR configuration, and return to the Normal world handler.

Note

Many Secure world implementations will not require floating point or SIMD operations. In a design where the Secure world does not use the coprocessors, the system boot code can make all coprocessors accessible to Non-secure software. In these cases there is no sensitive data in the coprocessors, and the monitor code does not need to context switch their state.

5.3.2 Interrupt model - monitor requirements

The interrupt model outlined in *Secure interrupts* on page 3-11, in which IRQ is configured as a Normal world interrupt and FIQ is configured as a Secure world interrupt, requires some core configuration by the monitor software on world switch.

The model proposes that if the processor is already executing in the correct world when an interrupt occurs, the hardware does not trap to monitor and jumps directly to the local world's vector table. This avoids the overhead of a switch to the monitor and keeps the monitor software design simpler.

Note

In a design that traps exceptions to the monitor you will trap to the appropriate entry in the monitor mode vector table; however the core will be in monitor mode, not the respective exception mode.

The Secure Configuration Register (SCR) in CP15 contains the settings which determine whether to trap IRQ, FIQ or external aborts to the monitor hardware. To implement the model proposed here the monitor needs to modify the content of the SCR on every world switch. When switching to the Normal world the SCR IRQ-bit must be cleared, and the SCR FIQ-bit must be set. When switching to the Secure world the SCR IRQ-bit must be set, and the SCR FIQ-bit must be cleared.

Note

This is only one of many possible models for using interrupts in a TrustZone processor.

5.3.3 Interrupt latency impact

In any design that has to switch worlds to handle an interrupt, the monitor becomes part of the critical path which defines the system's worst case interrupt latency. Unlike ARM R-profile and M-profile processors, which are typically deployed into deeply embedded systems, software deployed to an ARM A-profile applications processor typically does not need low interrupt latency. However, designs should take any additional latency introduced by the monitor software into account to ensure that the worst case behavior does not cause violation of any timing constraints.

The interrupt overheads added by the monitor may be higher than just the cost of a single world switch. For example, if the processor has just entered the monitor when an interrupt occurs it will transition through the monitor before handling it, if the processor then needs to switch back to the other world to handle the interrupt it will need to perform a second monitor transition.

In designs that want to minimize the interrupt latency impact, the code and data used by the monitor should be placed in fast memory close to the core. In systems using the ARM1176JZ(F)-S processor, the monitor can be placed in TCM. In systems which do not provide TCM, the monitor can be placed in locked-down L2 cache lines or fast on-SoC SRAM.

Example interrupt latency increases

A simple monitor which stacks the general purpose registers, and performs the SCR setting reconfiguration required for the interrupt model described previously in this section, may incur the following overheads:

- ARM1176JZ(F)-S:
  - Monitor code and data located in TCM.
  - Upper bound is 200 cycles per switch, or 400 cycles total.
  - 400 cycles at 300MHz is 1.3us.
- Cortex-A8:
  - Monitor code and data located in locked L2 cache.
  - Approximate overhead is 1200 cycles per switch, or 2400 cycles total.
  - 2400 cycles at 600MHz is 4us.

Note

Total interrupt latency will include overheads due to the Normal world software, the Secure world software, and aspects of the system design such as external memory performance. These overheads are not indicated in the figures in this list.

For comparison, the interrupt latency seen by a Linux driver running on an ARM1176JZ(F)-S with two levels of cache, and without any Secure world software, is approximately 5000 cycles. This is caused by overheads in the operating system itself.

5.4 Secure software and multiprocessor systems

Some designs which wish to make use of a secure execution environment may be based on multi-processor ARM designs, such as the Cortex-A9 MPCore processor. These systems allow higher software performance by concurrently executing threads on multiple hardware processors, which makes them ideal for processing intensive tasks such as software media CODEC processing.

The Secure world software architect has an important decision to make here – how will the Secure world software make use of multiple processors?

Most security systems aim to use a simple software design because simplicity means that there is less risk of a software bug which exposes a security vulnerability. Genuine multi-threading adds a layer of software complexity which is often hard to test because of the timing-sensitivity it introduces. For this reason many Secure world software implementations may choose to implement a single-processor Secure world which makes little or no use of the SMP processor features, even if the Normal world is running using full SMP mode.

5.4.1 Secure world processor affinity

If the choice is taken to implement a Secure world which is not capable of multi-processing then a design needs to be able to synchronize the communications between an SMP Normal world and the Secure world.

A design might choose to fix the Secure world execution on to one specific processor, which makes Secure interrupt routing simple, but also means that the Secure world may make the Normal world thread scheduling less efficient because it is using processor time which the Normal world cannot easily load balance. In this design the Normal world driver which communications with the Secure world usually needs to route requests to use the Secure world to the correct processor using inter-processor communications. In addition, the monitor software on the processors which the Secure world is not using must prevent the Normal world causing a malicious world switch. This architecture is shown in Figure 5-3 on page 5-14, in which the Secure world software is only using CPU0.

An alternative design may choose to let the Secure world migrate around the multiple processors in the system, restricting it so that it is only executing on one processor at any single point in time. This makes the Secure world more efficient because it can run on the same processor as the Normal world application which is making use of it, and allows the Normal world to load balance its scheduling, but it makes the routing of the Secure interrupts to the necessary processor more complicated.

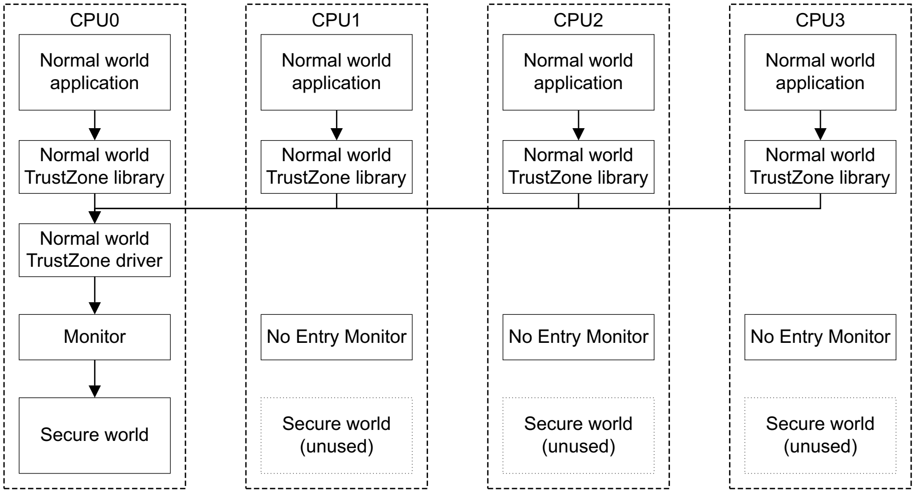


Figure 5-3 An example multiprocessor software architecture

5.4.2 Secure world interrupt usage

After deciding how the Secure world will make use of the SMP processor, a developer must then choose how to integrate any Secure interrupt sources in to the design.

In an architecture which executes the Secure world on a single fixed processor the interrupt routing is straightforward. The per-processor *Interrupt Controller Processor Interface Control Register* must be configured to suitably cause Secure interrupts to raise FIQ or IRQ exceptions, depending on how the software is designed to make use of the hardware features.

The interrupt routing model for an individual processor in a multi-processor system is described in more detail in the *Interrupt model - monitor requirements* on page 5-10. As that section shows, the monitor software for the processor which has to handle Secure or Non-secure interrupts needs to be able to route them to the appropriate world so that they can be handled by the correct exception handler.

In an SMP system where the Secure world is executing on a fixed single processor within the cluster, the FIQ interrupt can be made available for use by Non-secure interrupts on the processors not running the Secure world software. To prevent the hardware routing these interrupts to the Secure world the secure bootloader or the monitor software executing on these processors must ensure that the Secure Configuration Register in CP15 is appropriately programmed.

Note

In a design where the Secure world only executes on a single processor it is usually necessary for the monitor software running on the other processors to reject Normal world attempts to switch world using the SMC instruction, interrupts, or external aborts.

5.5 The TrustZone API

To encourage the development of security solutions ARM have produced a standardized software API, called the TrustZone API (TZAPI), which defines a software interface which client applications running in the rich operating environment can use to interact with a security environment.

The API is predominantly a communications API, enabling a client to send command requests to a security service, and to enable the client to efficiently exchange data with the services it is connected to. This communications interface is designed to support the principles of World-shared memory, for high performance bulk data transfer.

Secondary features of the API allow Normal world client applications to authenticate themselves with a secure service, query the properties of installed services, and allow trusted Normal world code to perform run-time download of new security services.

Note

The communications API supports synchronous and (optional) asynchronous calling conventions to support all of the common Normal world operating systems found on embedded devices.

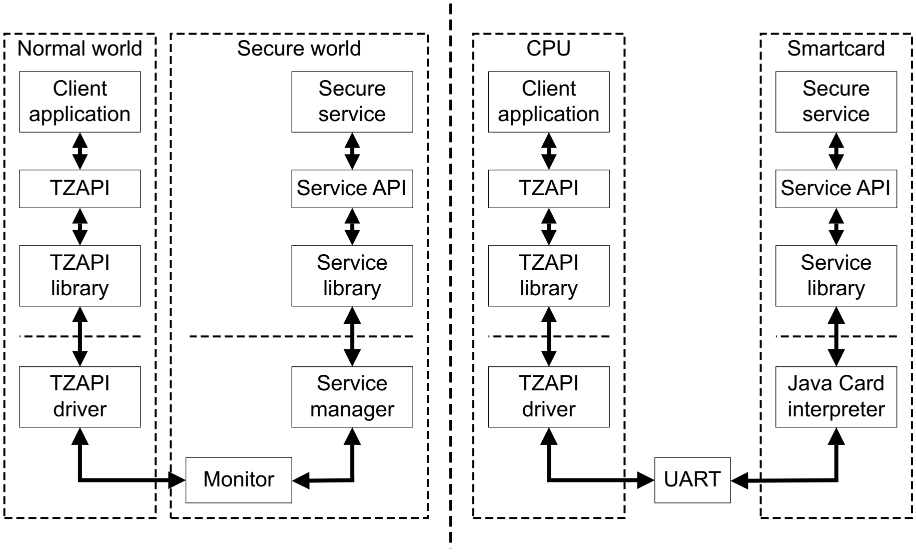


Figure 5-4 : Two example systems that might make use of TrustZone API

Although the TrustZone API is targeted at systems using a TrustZone-enabled processor, and tries to take advantage of the available hardware features such as World-shared memory, it is designed to be portable to almost any implementation of a secure environment. A system using an extensible security framework running on a smartcard would also be a suitable candidate for an implementation of TrustZone API. In these non-TrustZone systems the implementation of the TZAPI World-shared memory constructs might require a copy, but they should still be portable.

5.5.1 API availability

Recognizing that development of a security software ecosystem has been hindered by the lack of common standards for software development, ARM has released the TrustZone API as a public specification that can be downloaded and used free of charge by any software developer as an interface to their underlying security solution.

The TrustZone API can be downloaded for free from the ARM website. See `http://www.arm.com/TrustZone` for more details.

Chapter 6
TrustZone System Design

The chapter takes some possible security use cases and creates a SoC design for an imaginary mobile device, the Gadget2008.

This chapter contains the following sections:

6.1 Gadget2008 product design brief

The Gadget2008 product aims to provide a portable cellular handset, with a feature-rich operating system capable executing user downloaded applications.

To support the current trend for portable media, the handset will support audio and video playback. License deals with major content providers require the addition of a DRM content protection scheme in the software environment.

To provide additional revenues a companion content provision service, GadgetStore, will be developed. GadgetStore will allow the user to pay for downloaded content using standard banking facilities. The design should aim to reduce card transaction cost to the content provision service and minimise risk of the consumer being exposed to fraud.

6.2 Example use cases

As stated in the previous chapters of this document, creating a system that has the necessary security properties requires analysis of the assets that need defending, the types of attackers that a platform expects to encounter, and the details of the attacks that the design needs to provide countermeasures against.

This section outlines at a few use cases which will be used later in the chapter to demonstrate how TrustZone technology might be applied to defend the sensitive assets involved.

6.2.1 Content management

Many of the current security use cases in mobile devices are related to content management. The role of a Digital Rights Management (DRM) agent is to enforce the access to media content based on the rights that the user has purchased. The options for the rights are diverse; they may limit the user based on expiry date, number of plays, number of minutes of playback, and the playing device, amongst others. When a user selects a DRM protected file for playback the agent must ensure that the user's rights are still valid and, if they are, decrypt the file and pass it to the media player.

Integration in a system using TrustZone technology

The integration of a DRM agent within a TrustZone-enabled system is architecturally relatively simple. The design in Gadget2008 will place the DRM agent component, which is responsible for rights storage, rights validation and content decryption, in the Secure world. The design will place the CODEC and media player in the Normal world. Although this exposes the decrypted content to a degree of risk, mechanisms such as anti-virus can be used to detect unauthorized changes to the Normal world operating system and the media player, at which point the Secure world can stop sending content to the Normal world.

Placing the entire software CODEC stack in the Secure world is possible, but raises other security problems due to the size and complexity of a typical software implementation. Larger pieces of software have statistically more security vulnerabilities than a simple compact software environment, and could threaten the security of the more sensitive assets.

Note

Use of an external Secure world hardware block, such as a DSP or dedicated accelerator, could be used to decode the content files without exposing them to the Normal world. Additional hardware protections such as master ID filtering can be used to isolate the complex CODEC hardware from Secure world assets it does not need to access.

Assets

The assets associated with a DRM scheme are outlined in the table below:

Table 6-1 : Assets in an example DRM scheme

Asset	Security	Description
Rights Data	Authenticity	The rights data contains the information about the user's rights; for example, the number of plays remaining. It must be possible to check the authenticity of a rights object at run time.
Rights Checking Code	Authenticity	The code that checks the rights object must be authentic and executed from a secure location. If an attacker can modify the code for the rights check to always return `valid` then the DRM can be trivially bypassed.
Device Key	Confidentiality, Integrity	The device key is the root secret that is used to decrypt the rights data when transmitted from the content store. The device key must be kept confidential. If an attacker can steal the device key they can decrypt all rights objects to acquire all of the content keys. This would allow the attacker to decrypt all of the stored content files.
Content Key	Confidentiality	The content key is the secret key which is stored in a rights object, and which is used to decrypt a specific piece of content before playback. It an attacker could steal the content key they could decrypt the single piece of content that it relates to.
Content Data	Confidentiality	The content data is decrypted content that is passed to media player. System integrity monitoring and Normal world anti-virus can be used to reduce risk of theft.

Attackers

The primary attacker associated with a DRM scheme is the device owner; they have a desire to access free content and services.

Most attacks against content protection schemes on the desktop personal computer are software hack attacks, so we must defend against malicious software. The user also has physical access to the device, which means that simple hardware attacks, such as reprogramming attacks, must form part of the risk analysis.

6.2.2 Mobile Payment

Many embedded devices are now storing a large amount of user data, including sensitive information such as synchronised email, mobile banking details, and mobile payment credentials. This user data can be protected, requiring the entry of passcode before it can be used, however once unlocked it is vulnerable to any weakness in the underlying software environment.

Migrating the data storage, data manipulation, and even the passcode entry, into the Secure world makes sense for many applications that make use of user data. While all of these use cases have subtly different assets, they all share a similar security requirements. For the purposes of this example, Gadget2008 will use mobile payment which has stricter requirements than most of the other use cases.

Overview

There are two common payment models: *card-holder present* and *card-holder not present*. Card-holder present payments require some form of authentication that shows that both the user and their card are present, typically a user signature or Chip-and-PIN PIN entry. Card-holder not present transactions, such as those used when purchasing goods online, cannot authenticate the card and consequently have a higher degree of fraud. Card-holder not present transactions may cost the merchant more in transaction fees, and can also bear a significant risk of charge-back penalty fees against the merchant if a card transaction turns out to be fraudulent.

To meet the Gadget2008 design brief of reducing transaction cost and risk of fraud, it is desirable that the device can support a remote card-holder present transaction. For this purpose the design must be able to demonstrate that both the user and their card are present.

Note

Deployment of any new payment scheme will require the approval of the card issuers and card schemes. This example just tries to show some of the possibilities for a TrustZone enabled system.

Integration in a system using TrustZone technology

A TrustZone-enabled SoC provides a platform which allows the deployment of trusted software which can access a trusted keypad and a trusted display. Using this technology a payment application can securely display details of a remote purchase, and the user can enter a passcode on a trusted keypad to approve the transaction displayed on the screen.

For additional levels of security the passcode and transaction information could be passed to a smartcard, such as those found on Chip-an-PIN banking cards. Software running on the banking card can verify the correctness of the user passcode, and generate a signed approval for the remote transaction if the passcode is correct. The combination of secure user passcode, and a transaction approval which can only come from the user's banking card, allows the device to achieve a level of security much higher than card-holder not present transactions, and starting to approach the levels achieved by existing card-holder present transaction mechanisms. The smartcard can provide a higher level of tamper-resistance than the Gadget2008 handset, making it suitable for the storage of root payment credentials, but relies on the handset to provide the secure user interfaces.

The handset hardware will not have the same security as a FIPS 140-2 certified Chip-and-PIN terminal, which requires a degree of tamper resistance, but it is unlikely that a user would make physical modifications to their own device to steal their own banking passcode.

Note

It is likely that the Secure world hardware and software involved with the payment infrastructure would need to undergo a security evaluation process to meet the requirements of the payment schemes.

Assets

The assets associated with a payment scheme are outlined in the table below:

Table 6-2 : Assets in an example payment scheme

Asset	Security	Description
Transaction Information	Authenticity	When the user approves a remote transaction it is important that they have confidence that they are approving the correct information. Allowing a user to think that they have approved the purchase of a $10 book, only to later find out that they have paid $1000 for a fake watch, would be very damaging for consumer confidence in the payment scheme. This places a requirement that the information that the user actually approves, which is essentially what they see on the device display, comes from a trusted source.
User and Card Information	Confidentiality	The user's card details, such as their card number and its date of issue, and other user details, such at their home address, are all of value to an attacker attempting to perform card-holder not present transactions. These assets must be protected, as card-holder not present fraud is one of the fastest growing financial frauds in the market today.
User Passcode	Confidentiality	The user's passcode is of value if an attacker can also steal the handset and the corresponding banking card. The value of the passcode must not be stored on the handset, otherwise it could be recovered by an attacker who could then perform fraudulent card-holder present transactions.
Root Payment Credentials	Confidentiality	The root cryptographic secrets that are used to approve a transaction must be kept confidential, otherwise an attacker could approve arbitrary fraudulent transactions. The security of these root secrets is critical to the security of the whole payment scheme and, in this example, will rely on the security provided by a banking grade smartcard.

Attackers

If the security of a payment scheme can be broken, the fraud that can be committed is worth a lot of money. Payment schemes therefore attract a large number of attackers, from those attacking the hardware, to those performing e-mail scams trying to steal the users’ details. Most attacks against these devices will be remote, implying software-only hack attacks, but stolen devices and devices sent in for repair may be attacked using physical shack attacks too.

6.3 Gadget2008 specification

Now that there is an outline of the use cases, and the security assets involved, it is possible to start designing a platform that meets the requirements.

6.3.1 General specification

There are a number of requirements which are not specific to the use cases described in the previous section. These generic requirements are covered in this section.

Secure boot

As outlined in *Booting a secure system* on page 5-5, one of the common methods of attacking mobile devices is reprogramming the device firmware with a modified copy. A Secure boot implementation is needed to protect against this.

Gadget2008 will include:

- Secure boot code located in on-SoC ROM,
- 256-bits of OTP fuse which can contain a SHA256 hash of the RSA public key owned by the Secure world software developer,
- and a statistically unique secret key located in an on-SoC cryptographic accelerator.

Secure boot investigations have shown that sufficient software cryptography for image decryption and signature verification can fit into around 4KB of ROM. There is an additional requirement for 8KB of on-SoC RAM to contain any dynamic boot state, otherwise the secure boot secrets are vulnerable to snooping attacks on external memory interfaces.

The use of OTP fuse to store a cryptographic hash of the RSA public key means that we can distribute devices with a variety of root keys, reducing the risk of a successful class break attack.

The inclusion of a statistically unique secret, available only to Secure software, in the on-SoC cryptographic hardware means that confidential data can be encrypted and bound to the device. The only mechanism to recover the secret key is to perform side-channel analysis of the cryptography, which the hardware accelerator design should prevent, or to dismantle the silicon to recover the key. This is likely to be too expensive given the value of the assets that the Gadget2008 contains, so this is a good security countermeasure.

Multi-tasking Secure world software

The Gadget2008 design requires multiple pieces of software in the Secure world: a DRM agent provided for management of the content purchased from the GadgetStore, a secure GadgetStore shopping backend, and a payment application provided by the user's card issuer.

In a real-world scenario it is unlikely that the stakeholders that provide this software will actually trust each other enough to exist without some form of isolation. For this reason many Secure world software stacks will need to include a privileged kernel component that is trusted by all of the stakeholders, and this kernel must provide robust isolation between the security services that are running.

The Gadget2008 will include:

- A Secure world operating system using the MMU to separate user tasks.

Securing the debug channels

The debug scheme for the Gadget2008 will split the production devices into two batches. The initial batch will be a small development run which will allow full debug access, including Secure world access. These devices will only be shipped to trusted parties, and the OTP fuse will be set to a known invalid key. The second batch of devices will enable Normal world debug, but permanently disable all Secure world debug using a fuse.

Other system aspects

There are many aspects of a SoC design that this document has not covered. These areas include the use of multiple clock domains, multiple power domains, and power saving states such as system hibernation in which the power to the SoC is removed.

All of these could have an impact on the security of the system, depending on the use case and the assets involved. For example, a power controller might be able to power down areas of the SoC performing background security checks, and clock frequency changes might expose a cryptographic algorithm to side-channel analysis. Any design must review, and take into account, the possible influences of any system design aspect and how they might affect the system as a whole, given the level of security that is desired.

Power management is one area in particular where the Secure world software may need to provide a secure protocol implementation. Many modern cellphone SoCs aggressively power save, turning off portions of the SoC while still giving the user the impression that the device is fully active. In these designs the state of the system is saved to external DRAM, which is then put in a state-retaining low power mode, and then the SoC is powered down. On power up, the power controller informs the device that this is a return from hibernate, or a warm boot, and the device restores the running state from the external DRAM. If the Secure world in such a system uses on-SoC memory to store its resources, the hibernate and warm boot cycle must be taken into account in the security analysis. The main reason for using on-SoC memory is to prevent snooping attacks, so this implies that the content of hibernated state stored in external memory must be encrypted. If the design is also concerned with replay attacks, in which an attacker tries to restore from an old hibernation state, it may need to incorporate a non-volatile count value in the encrypted state.

6.3.2 Content management specification

The content management scheme has the following requirements:

Software video decoding

The Gadget2008's most performance intensive task is handling software decode of the video files that the user can purchase from the GadgetStore. Media CODEC algorithms can be relatively processor intensive so the design will use a uniprocessor Cortex-A9 processor implementing the NEON media processing engine.

In summary, the device will include:

- An ARM Cortex-A9 processor.
- AMBA3 AXI High Performance Bus Matrix (PL301).

Note

The total processor bandwidth required for a TrustZone-enabled processor is the sum of the Normal world processing requirement and the Secure world processing requirement. In this case there must be enough processor frequency and memory bandwidth to handle the concurrent execution of the media CODEC and the DRM decryption task.

Soft real-time performance

Any use case involving streaming media will require a soft real-time response from any component involved in the media processing, including the DRM agent. If content is not provided to the media CODEC when it is needed, the quality of the output will degrade and the end-user experience will be poor.

To support these requirements the Secure world software will be a multi-tasking pre-emptive operating system, using a secure interrupt source to preempt other tasks running on the processor. This enables the higher priority DRM task to be efficiently scheduled, and should ensure that the CODEC does not run out of content to decode. In addition, the data rates associated with video playback are judged to require a level cache in the system.

The Gadget2008 will include:

- An ARM Dual-Timer Module (SP804).
- A PrimeCell Generic Interrupt Controller (PL390).
- A PrimeCell Level 2 Cache Controller (PL310).

Non-volatile counter

One of the attacks that a content management system is vulnerable to is a replay attack, in which an attacker replaces a file system image containing an expired rights object with an older version containing the rights object before it had expired. Even encrypting the file system image is not a protection in its own right; an attacker can simply swap the encrypted images without even needing to break the cryptographic protections.

The addition of a non-volatile counter to the design allows a hardware enforced version count to be embedded in the stored data. Building a secure storage protocol enforced by strong cryptographic measures around this counter value can enable the system to identify and prevent replay attacks.

The Gadget2008 will include:

- A 32-bit on-SoC non-volatile counter.

Secure real-time clock source

Many content schemes implement some form of time-based rights issue, so it is a requirement for the platform to support a Real-Time Clock (RTC) which cannot be modified by the Normal world software.

Synchronizing the on-device RTC with a more trusted time managed by the back-end content server enables the on-device RTC to be corrected for minor clock drift. The design will also store the last received server-time in a file protected by the Non-volatile counter so that it can also protect against replay attacks.

The Gadget2008 will include:

- A Secure world Real-Time Clock.

Memory requirements

Many of the content management schemes are complex, requiring significant quantities of Secure world code and data to implement a full agent. The cost of sufficient on-SoC memory to contain all of the code and data would be prohibitive, so Gadget2008 will use a TrustZone Address Space Controller to partition a single off-SoC DRAM.

The Gadget2008 will include:

- A TrustZone Address Space Controller (PL380)

6.3.3 Mobile payment specification

Mobile payment places the following requirements on the device:

Payment sub-system

The security requirements placed on a payment system are normally significantly higher than those required for a content management agent. To provide additional protection against hardware attacks it is desirable for the Secure world code and data related to payment, and the security kernel code and data that separates payment from other Secure world services, to be located in on-SoC RAM rather than off-SoC RAM.

Initial investigations have shown that the code and data needed for a light-weight Secure world kernel, with a small number of security services, running could fit into a 96KB on-SoC RAM. Using the MMU, the Secure world kernel could make use of the larger off-SoC memory for less critical use cases such as content management. The Gadget2008 design will include a 128KB SRAM, and use a TrustZone Memory Adapter to allow the bottom 96KB to be made Secure. This leaves 32KB available for Normal world software.

To allow for some flexibility in the design we will use a TrustZone Protection Controller to provide the inputs to the TrustZone Memory Adapter, enabling the partition size to be optimally positioned at boot-time.

The Gadget2008 will include:

- 128KB of on SoC SRAM,
- A PrimeCell Infrastructure AMBA3 TrustZone Memory Adapter (BP141).
- A PrimeCell Infrastructure AMBA3 TrustZone Protection Controller (BP147).

Secure keypad

The external keypad controller in the Gadget2008 design communicates with the main SoC using the PS/2 protocol, so we will use a standard APB peripheral to handle this. The design will use a TrustZone Protection Controller to set the security setting of the peripheral within the AXI-to-APB bridge, enabling the keypad to be switched in and out of the Secure world at run-time.

The Gadget2008 will include:

- PrimeCell PS/2 Keyboard/Mouse Interface (PL050)

Secure display

The Gadget2008 design uses simple framebuffer graphics, in which a display controller copies the content of the framebuffer memory to an external LCD interface every frame.

To support a securable display the hardware design will include two framebuffers, one Secure and one Non-secure. The display controller will read the content of both framebuffers each frame, and will overlay the visible portions of the Secure display on top of the Non-secure display. This enables a display with protected integrity, which is the important security property required for the payment transaction approval screen.

The Gadget2008 will include:

- A custom display controller

Near-Field Communication interface

To economically enable a level of security which is similar to a card-holder present transaction the handset must be able to communicate with a banking card. This will be achieved by including a Near-Field Communication (NFC) reader interface in the device, allowing it to communicate with banking cards that implement the passive NFC card interface.

Banks and credit card companies that support our payment scheme will need deploy an application onto their cards. A Secure world software service can communicate with this smartcard application to approve a payment transaction. The Secure world service will manage access to the keypad and the secure display, but the banking card application is responsible for consuming the user’s passcode to actually approve the financial transaction.

6.3.4 Putting the hardware together

Putting all of the features discussed in this section together, we can produce the following SoC design:

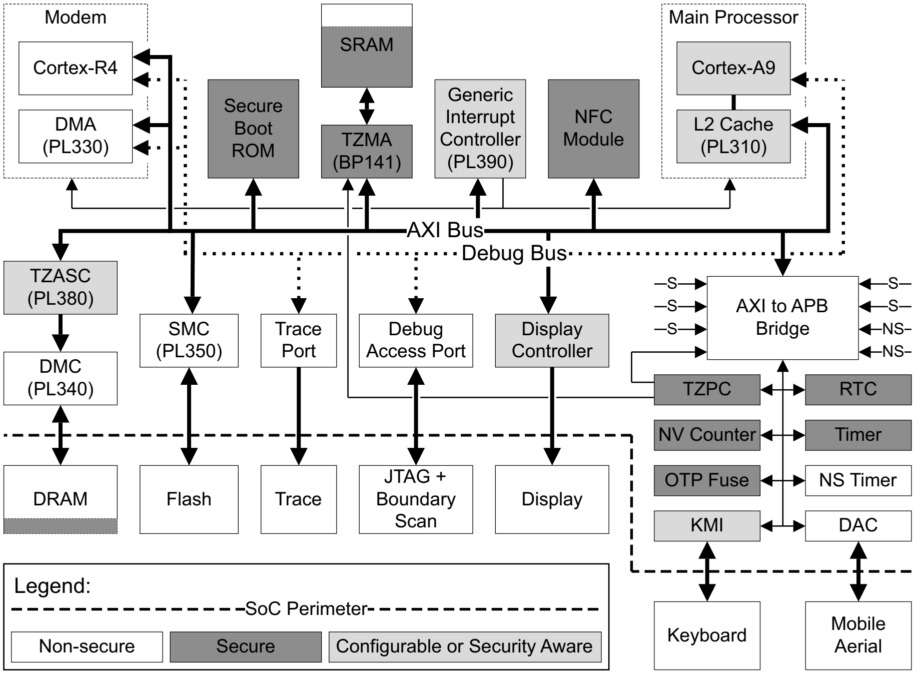


Figure 6-1 : The Gadget2008 SoC design

Note

Note that this design only shows the aspects that are relevant to, or overlap with, the security of the system. Most Normal world peripherals, and all system peripherals, such as power controllers, are not shown.

Putting the software together

The software architecture can similarly be outlined, as shown in Figure 6-2 on page 6-16. In this design the Secure world is implemented around microkernel principles and pushes critical system components, such as device drivers, into unique user-space tasks.

Three separate security services are installed into the Secure world, one for each of the DRM, GadgetStore and mobile payment use cases. These services are responsible for managing all of the sensitive assets associated with each use case and making use of hardware, via the secure device drivers, to adequately protect them.

The Normal world communicates with the Secure world using the TrustZone API, and implements a driver component which manages concurrent communication sessions from Normal world client applications such as the media player and the GadgetStore front-end.

Finally, the Secure world is able to use a NFC interface to communicate with a payment application running on a banking card. This works in conjunction with the on-device payment service to approve financial transactions.

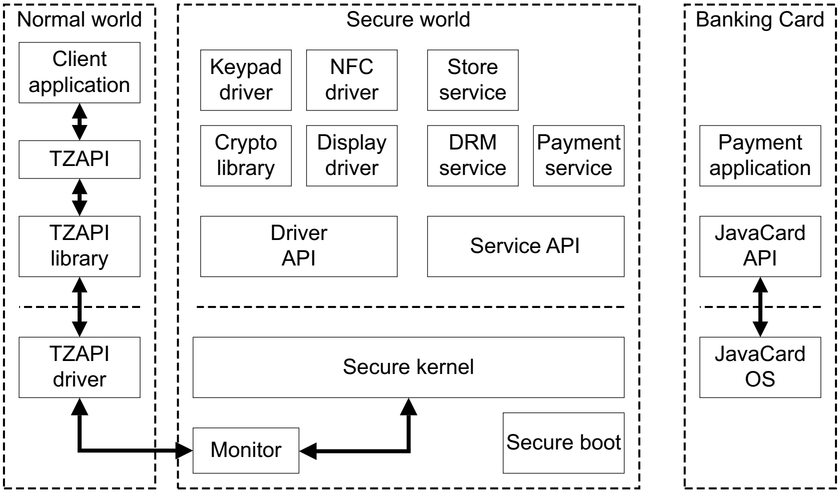


Figure 6-2 : The Gadget2008 software architecture

Note

To allow the Normal software to make use of the device's external communications interfaces, such as NFC or GPRS, a design may choose to make the lower levels of the communications protocol stacks run in the Normal world environment.

It is possible to use cryptographic methods to create a secure tunnel between the Secure world software and the device on the other side of the communications link, without exposing the content of the traffic to the less trusted Normal world.

Chapter 7
Design Checklists

The chapter provides some checklists which can be used when designing a SoC using TrustZone technology, or when reviewing such a design for security facilities.

This chapter contains the following sections:

7.1 Use case checklist

This checklist covers the first part of the design phase; identifying use cases, assets and the threats.

- What are the use cases for the SoC?
- What are the assets that exist in each use case?
- What security properties, such as confidentiality or integrity, does the asset require?
- Who are the expected attackers of each asset?
- What attack methods will each attacker be able to use?
- Who are the security stakeholders and who do they trust?

7.2 Hardware design checklist

This checklist covers the second part of the design phase; identifying the fundamental hardware requirements:

- Where are assets stored and handled on the system and how are the corresponding hardware components secured?

  Does the location of assets sufficiently match the physical hardware isolation barriers, such as the SoC packaging, which are implemented on the system?

- Are assets belonging to each stakeholder protected against attack from other stakeholders?

- Is the security configuration, including master filtering, of any bus slave, bridge, or custom logic, correct?

- Is the security configuration of any APB peripheral correct, including APB configuration ports of larger components such as AXI memory controllers?

- Are any secret values used in the design unique enough to prevent class break attacks?

  Should the use of one-time-programmable resources, such as metal-layer fuse, be considered to enable a per-device identity to be installed?

- Should particular assets be kept on-SoC to enhance protection against shack attacks?

  This analysis should include the less tangible assets, such as the control of on-SoC signals like processor debug enable. If debug enable is controlled from outside of the SoC it would be easy for an attacker to manipulate the SoC pin input to enable debug.

  Is there enough on-SoC memory for the software code and data that is needed to run there? An 8KB SRAM is sufficient for Secure boot and the Monitor, but 96KB should be considered if OS-like functionality is needed in the Secure world to separate multiple concurrent tasks or stakeholders.

- Is the design fixed function, or would dynamic control of configuration be useful?

  If dynamic control of the security of hardware blocks is useful then consider the use of a TZMA, TZPC, or TZASC to allow boot-time or run-time configuration of security configuration.

- How is invasive debug and the built-in self test infrastructure protected against malicious use?

- How is non-invasive trace debug protected against malicious use?

- Does it need to be possible to re-enable debug in the field? How is this mechanism secured?
- Are system components, such as clock controllers and power controllers, suitably protected?
- If the **CP15SDISABLE** signal of the ARM processor is used, how is it secured?

7.2.1 Multiprocessor design

This section of the checklist asks additional questions which are only relevant to designs making use of an ARM core implementing the multi-processor extensions.

- Are the debug control signals for each of the processors in the cluster secure?
- If the cluster-wide **CFGSDISABLE** signal of the ARM processor is used, how is it secured?
- Can Secure interrupts be routed to the correct processor, given the expected software usage model?

  Are legacy secure **nFIQ** interrupts, if any, routed to the correct processor, given the expected usage model?

7.3 Software design checklist

This checklist covers the third part of the design phase, software requirements, and the impact on the hardware design which they might have.

- Does the design need, now or in the future, more than one service in the Secure world?

  If no, then a single threaded library is a suitable design, otherwise consider a Secure world operating system with MMU separation between service tasks.

- Is any part of the Secure world required to handle soft real-time tasks, such as media processing?

  If yes, then consider making Secure world scheduling preemptive. This places requirements on the design to include a hardware interrupt source for the Secure world.

- Is any part of the Secure world or the Normal world sensitive to interrupt latency?

  If yes, then consider placing the monitor code and data in TCM (if available), locked cache lines, or fast on-SoC SRAM.

- Does any part of the Secure world or the Normal world require hard real-time behavior?

  If yes, then the design of both worlds, and their interaction, will need review to ensure that any path which disables interrupts is sufficiently time bounded.

- Does the monitor software correctly context switch all of the appropriate state for each possible path through it?

- Does only the monitor software modify the NS-bit in the CP15 SCR?

- Does the Secure world care about direct or indirect Normal world visibility of its execution?

  If yes, then consider obfuscating interrupt timing, disabling Non-secure access to the Performance Counters, and performing selective cache maintenance on critical address ranges on world switch.

7.3.1 Multiprocessor design

This section of the checklist asks additional questions which are only relevant to designs making use of an ARM core implementing the multi-processor extensions.

- Does the software need to make use of the SMP cluster, or should it only run on a single processor to benefit from simpler software design?

- If the software is SMP, or single-threaded but migrates around based on processor usage, how are interrupts routed to ensure that they can be taken reliably?
- If the software is single-threaded and only runs on a specific processor in the cluster does the monitor software running on the other processors correctly reject attempts by Non-secure software to perform a world-switch?

  This should include attempts to switch using the SMC instruction, interrupts, and external aborts.

Glossary

The items in this glossary are listed in alphabetical order, with any symbols and numerics appearing at the end.

ACP Accelerator Coherency Port. See page 3-10.

AHB Advanced Hardware Bus. The main system bus interface in the AMBA2 specification.

AMBA Advanced Microcontroller Bus Architecture.

AMP See *Asymmetric Multi-Processing*.

APB Advanced Peripheral Bus. The simple peripheral bus interface in the AMBA2 and AMBA3 specifications.

Asset A resource of value which needs protecting in a secure system.

Asymmetric Multi-Processing Execution within a multi-processor cluster without utilizing hardware enforced data coherency.

Attack The act of trying to acquire, damage or disrupt an asset.

Attacker A person creating or performing an attack.

Authenticity Authentic data is data which can be shown to be in a trusted state.

AXI Advanced eXtensible Interface. The main SoC level system bus interface in the AMBA3 specification.

Class-break A single attack that can be used to break a significant number of similar devices.

CODEC enCOder / DECoder algorithms used for compressing and decompressing data. Typically used for audio and video media files.

Confidentiality An asset that is confidential cannot be copied or stolen by an attacker.

CP15 The ARM processor system control coprocessor.

Defend The act of protecting an asset against an attack.

DRM Digital Rights Management.

GIC Generic Interrupt Controller. See page 4-6.

Hack attack An attack which is performed using software-only methods.

Hacker See *Attacker*.

Integrity An asset which has integrity cannot be modified.

IP Intellectual Property.

Lab attack An attack that is capable of observing and altering all aspects or a device.

Monitor mode A new processor mode added as part of the Security Extensions to facilitate the context switching between the two virtual processors.

Monitor, the The implementation defined software that runs in Monitor mode to context switch between the two virtual processors.

Non-secure (1) A device (master or slave) that exists in an untrusted part of the system.

(2) The untrusted virtual processor in an ARM processor implementing the Security Extensions.

(3) A transaction on the bus trying to access a Non-secure device.

Normal world The system environment that encompasses all Non-secure devices and software.

One-Time-Programmable memory A memory device that can only be programmed once, but which allows per-device customization. Typically implemented using fuse technology, such as poly-silicon fuse or metal layer fuse.

OTP memory See *One-Time-Programmable memory*.

SCU See *Snoop Control Unit*.

Secure (1) A device (master or slave) that exists in the trusted part of the system.

(2) The trusted virtual processor in an ARM processor implementing the Security Extensions.

(3) A transaction on the bus trying to access a Secure device. Only Secure masters can create Secure bus transactions.

Secure world The system environment that encompasses all Secure devices and software. Some Secure devices may allow access to Non-secure devices to be performed, for example, an ARM processor using World-shared memory.

Secure Monitor Call An ARM instruction added to the ARM cores implementing the Security Extensions. This instruction allows privileged code in the Normal world and the Secure world to switch the processor into monitor mode.

Security Extensions The extensions made to the ARM processor cores to enable the TrustZone technology. This primarily encompasses the creation of two virtual processors within a single physical processor core.

Shack attack An attack performed with simple hardware, such as logic analyzers, but which cannot access resources within a integrated circuit package.

SMP See *Symmetric Multi-Processing*.

Snoop Control Unit Part of a multi-processor cluster which maintains data coherency between the data caches of processor executing in Symmetric Multi-Processing mode.

SoC See *System-on-a-Chip*.

Symmetric Multi-Processing Execution within a multi-processor cluster utilizing hardware enforced coherency of the L1 data cache.

System-on-a-Chip An integrated device containing the majority of a device's logic, including processors, memory controllers, and peripherals.

TCM See *Tightly Coupled Memory*.

Tightly Coupled Memory Fast SRAM located at the same level of the memory hierarchy as the level one cache.

TrustZone The security technology from ARM that enables the construction of a Normal world and a Secure world.

TZASC TrustZone Address Space Controller. See page 4-5.

TZMA TrustZone Memory Adapter. See page 4-6.

TZPC TrustZone Protection Controller. See page 4-6.

World-shared memory Non-secure memory mapped into the Secure world using the MMU of the ARM processor.