




# 基于TrustZone技术的安全嵌入式操作系统的设计与实现

Add Your Company Slogan

导师：郭炜  
学生：温旺庭



# 目录

- 1** Introduction
- 2** Task and design
- 3** Implementation
- 4** My problem
- 5** Future

# Introduction

嵌入式系统，例如智能手机，处理许多敏感数据，例如个人网络银行账户信息、数字版权处理等等。其中还有许多应用可以自动下载应用软件。

通常，安全性要求较低的应用和安全性较高的应用共同运行在一个系统环境中。

ARM提出一种**硬件层的虚拟化技术**，该技术将系统完全划分为两个完全独立空间，即**安全空间**以及**非安全空间**。TrustZone只是一个系统架构，她要求将计算机所有硬件资源（虚拟安全core以及虚拟非安全core, 安全Memory以及非安全memory）以及软件资源(Secure kernel, Non-secure kernel, bootloader)划分为两个独立区间。

# Task division

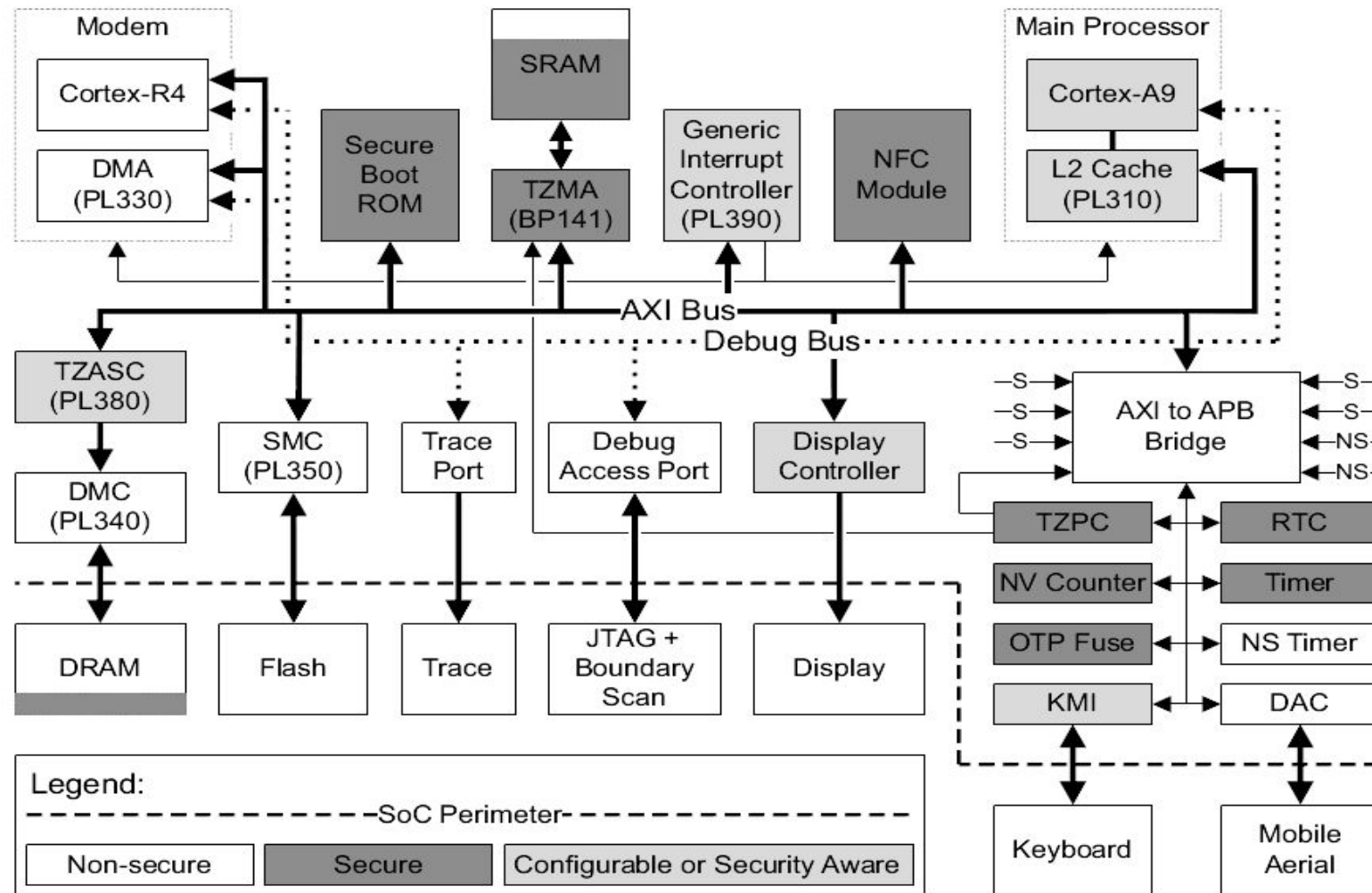
## Trust Zone硬件架构:

- 1 系统架构
- 2 Memory 划分
- 3 processor架构
- 4 Debug架构

## Trust Zone软件架构

- 1 bootloader
- 2 monitor code
- 3 Trust API

# One hardware design Implement TrustZone



## TrustZone hardware

prev: dedicated hardware block  
while TrustZone enables any parts of system to be made secure with system security protocols on the top of TrustZone arch, such as secure bootloader, authenticated debug enable.

➤ **First aspect:**

TrustZone-enabled AMBA3 AXI bus fabric non-secure software cannot access to secure asserts.

➤ **Second aspect:**

extensions implemented in core, which provide two virtual cores in time-sliced fashion.

➤ **Last aspect:**

secure-aware debug architecture( In fact, I not kown what's it)

# System architecture

## ➤ AMBA3 AXI system bus

addition of extra control signal for each read/write channel of main system bus. These bits are known as NS bit.

AWPORT[1] write control low secure and high is non secure

ARPORT[1] read control low secure and high is non secure

Note:

all bus master set these bits while slave interpret them

## ➤ AMBA3 APB peripheral bus

APB attached to bus via AXI-to-APB bridge, APB not carry NS bit, AXI-to-APB bridge manages the security of peripheral.

## ➤ Memory aliasing

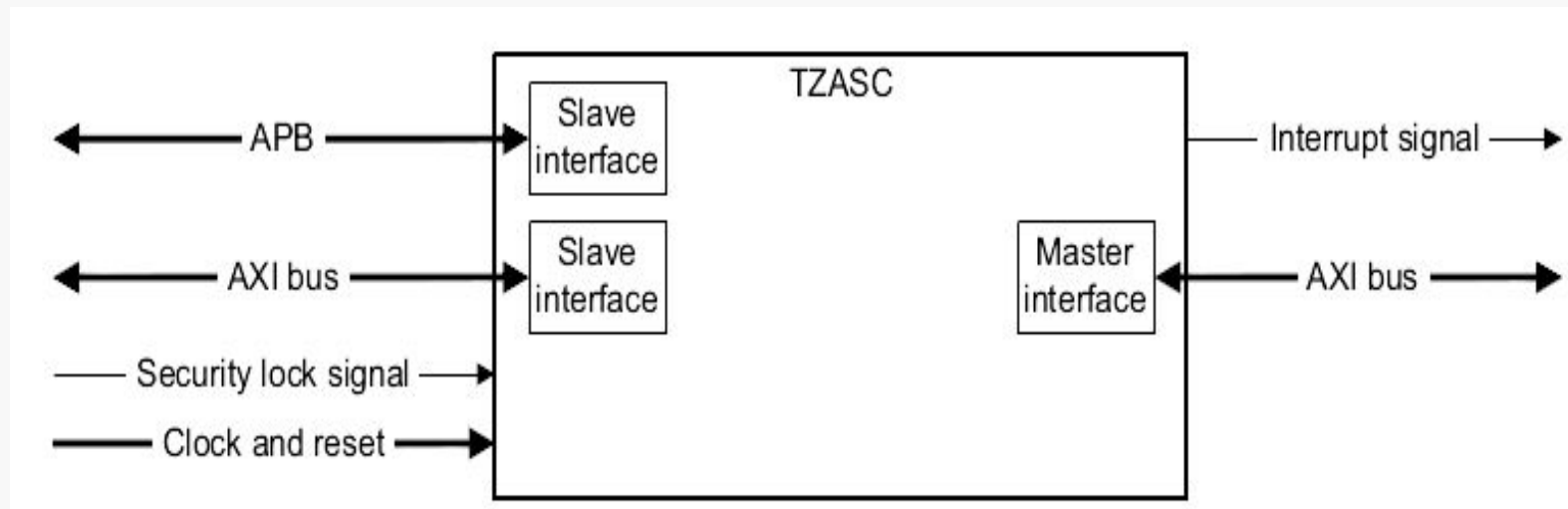
the NS bit added to system bus and cache tags, can be viewed as 33rd address. 32 are phy addr, and NS bit is for secure or non secure space.

# Dynamic control of configuration

make use of TZMA, TZPC, TZASC to allow boot-time and run-time security configured.

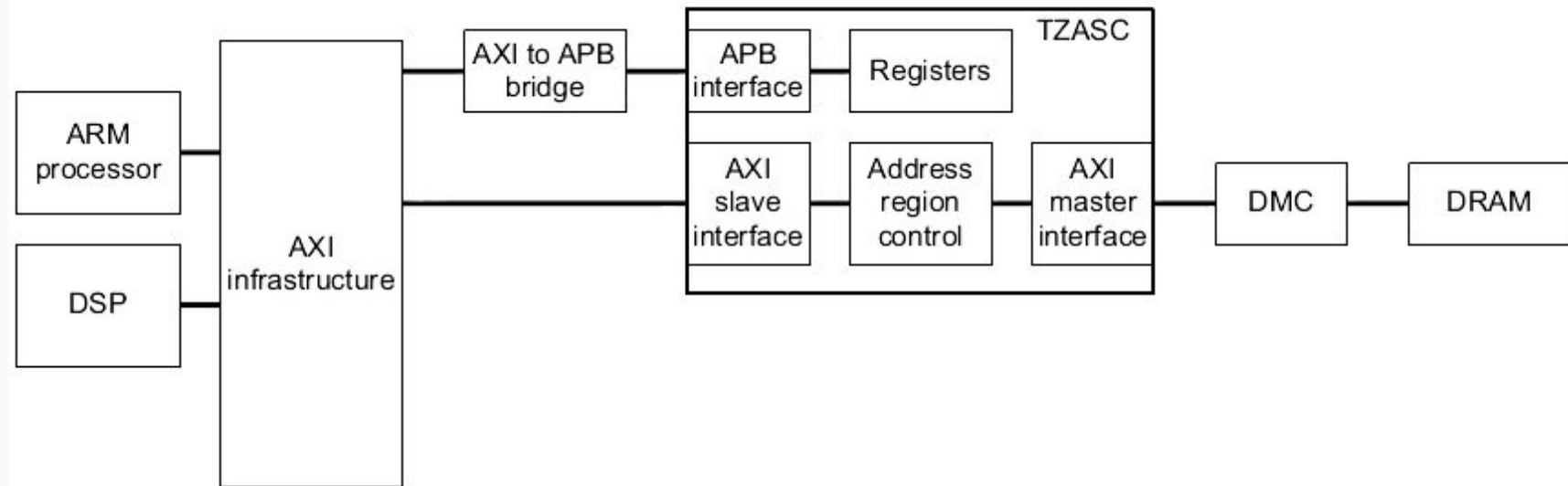
**TZASC:**

**an peripheral with AMBA interfaces(address space controller)**





# TZASC(TrustZone address space control)



## TZASC features:

- 1 enable programming security access of each address region
- 2 data transfer between master and slave(while security access valid)
- 3 prevent write access to registers after secure\_boot\_lock

# TZASC interfaces

**TZASC interfaces:**

**AXI bus interfaces:** AXI slave interface AXI master interface

**AXI channels:**

**Write Addr/Write Data/Write Response/Read Addr/Read Data**

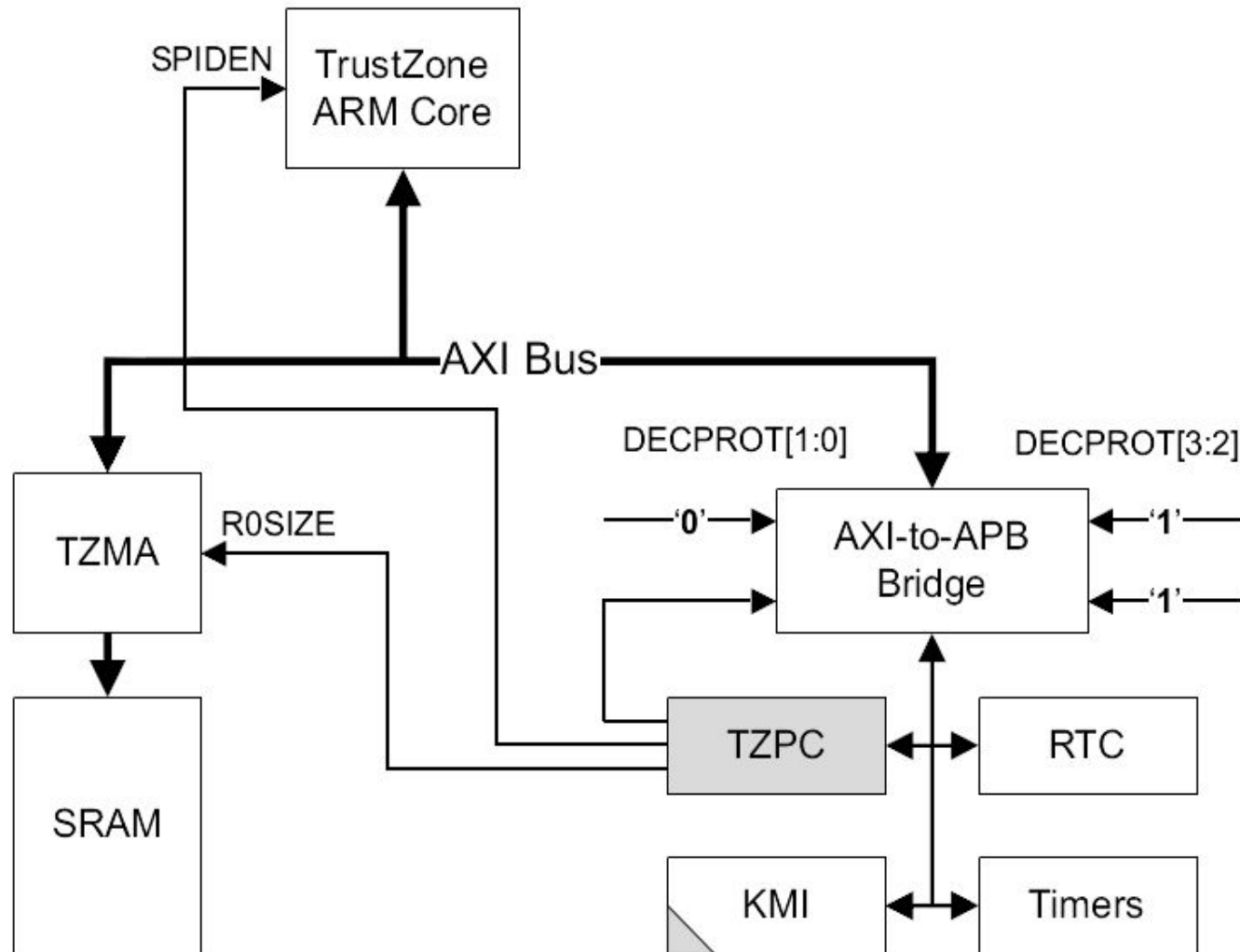
**key signal: secure\_boot\_lock and tzasc\_int**

**TZASC programmable:**

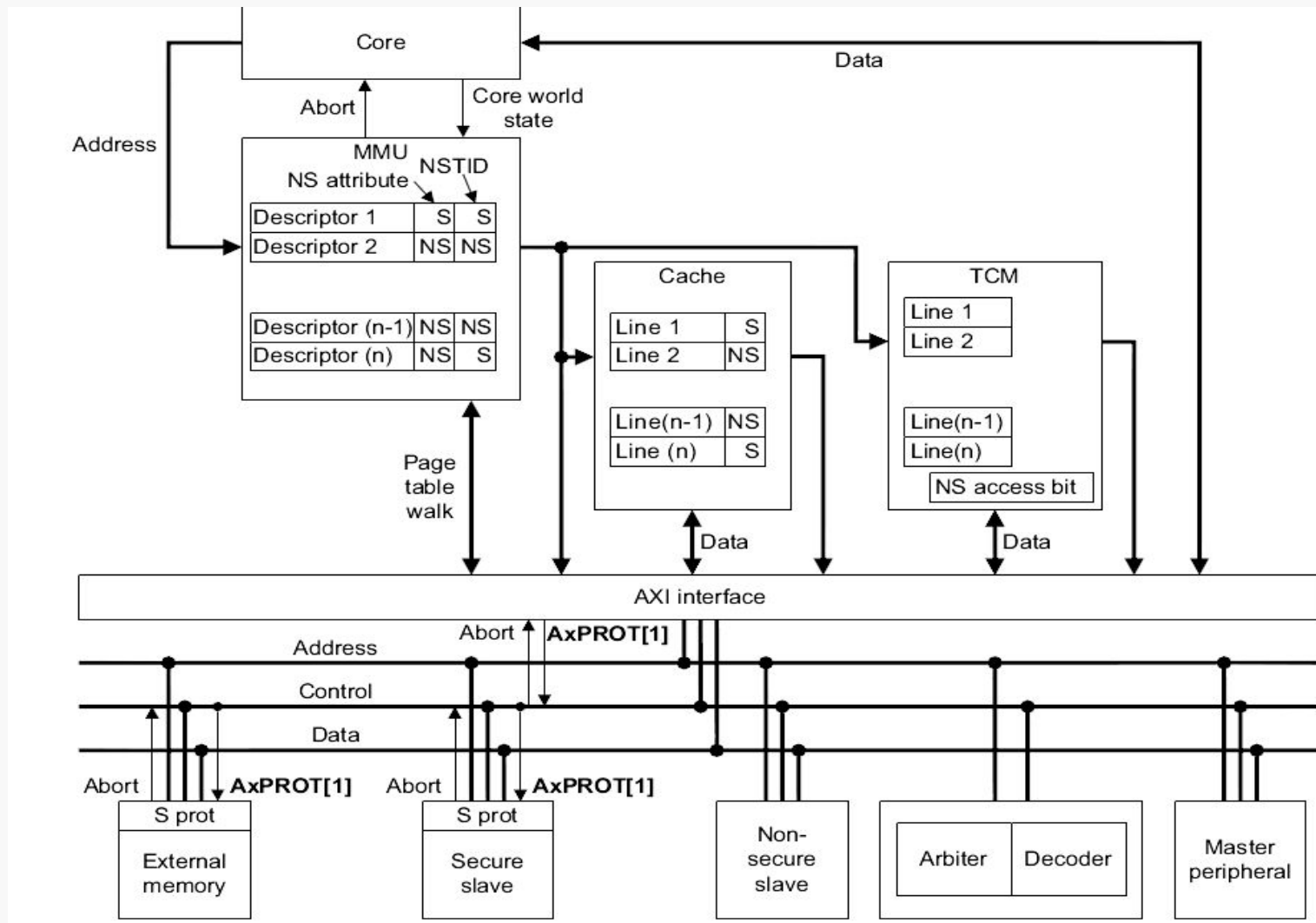
**size, and base address and enable and security**



# TZPC() control security signals



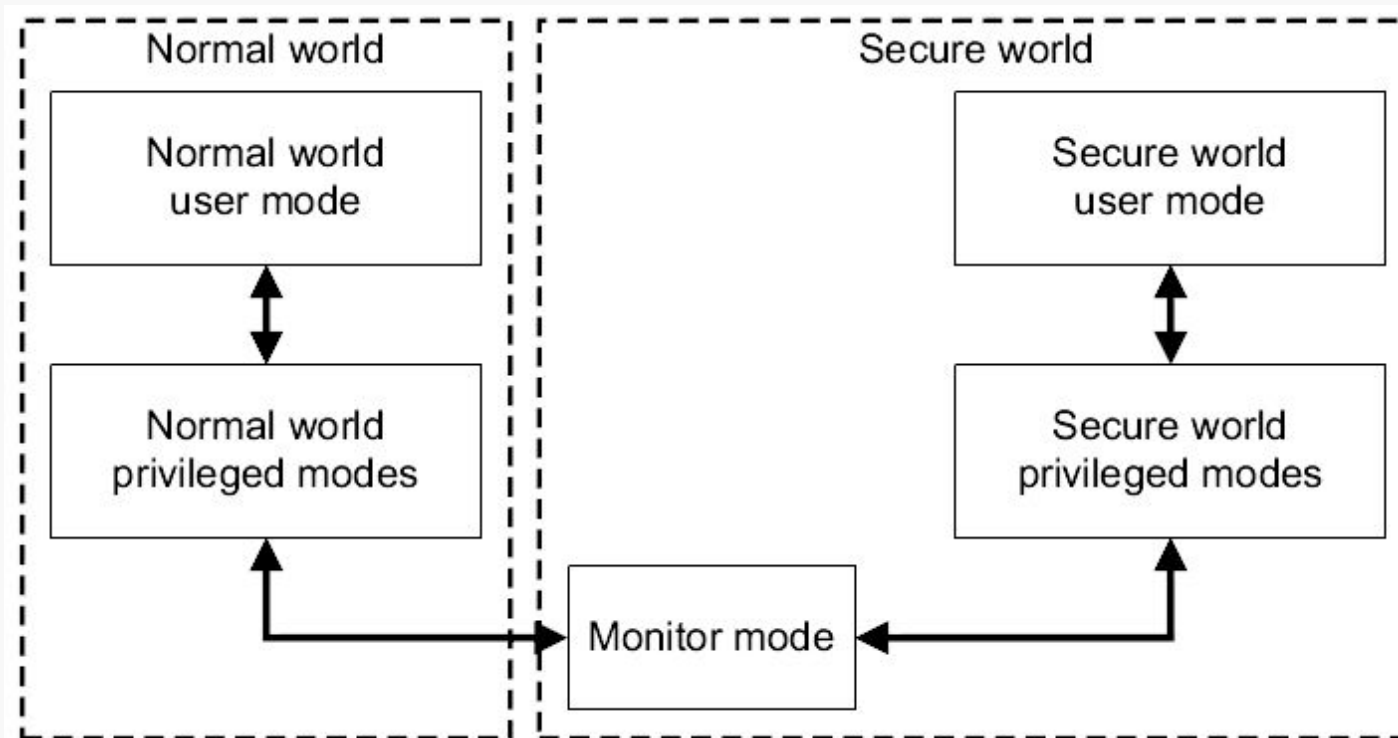
# ARM trustZone安全区间以及非安全区间Memory



## Processor architecture

Each phy core provides two vitual cores and a mechanism to context swtichs between them.

Implemented core: ARM1176 cortex-A8 cortex-A9



## Processor architecture

we need to separate the data used and stored in L1 memory system.

the major component of in arm is MMU, mapping virtual address to phy address , address translation is managed by a software-control,

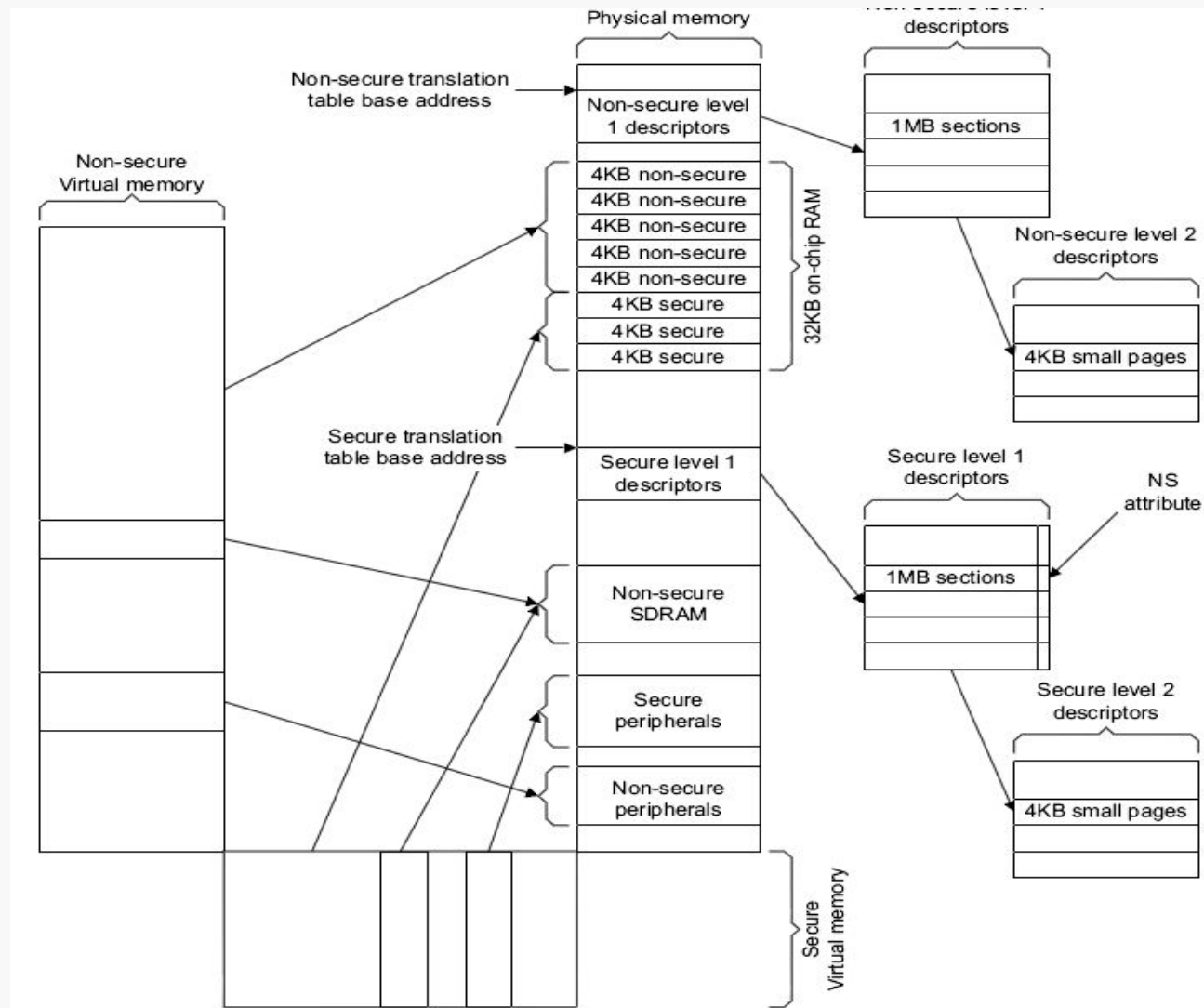
Within a TrustZone processor the hardware provides two virtual MMUS, one for each world, so two world have virtual address space.

**Addition:**

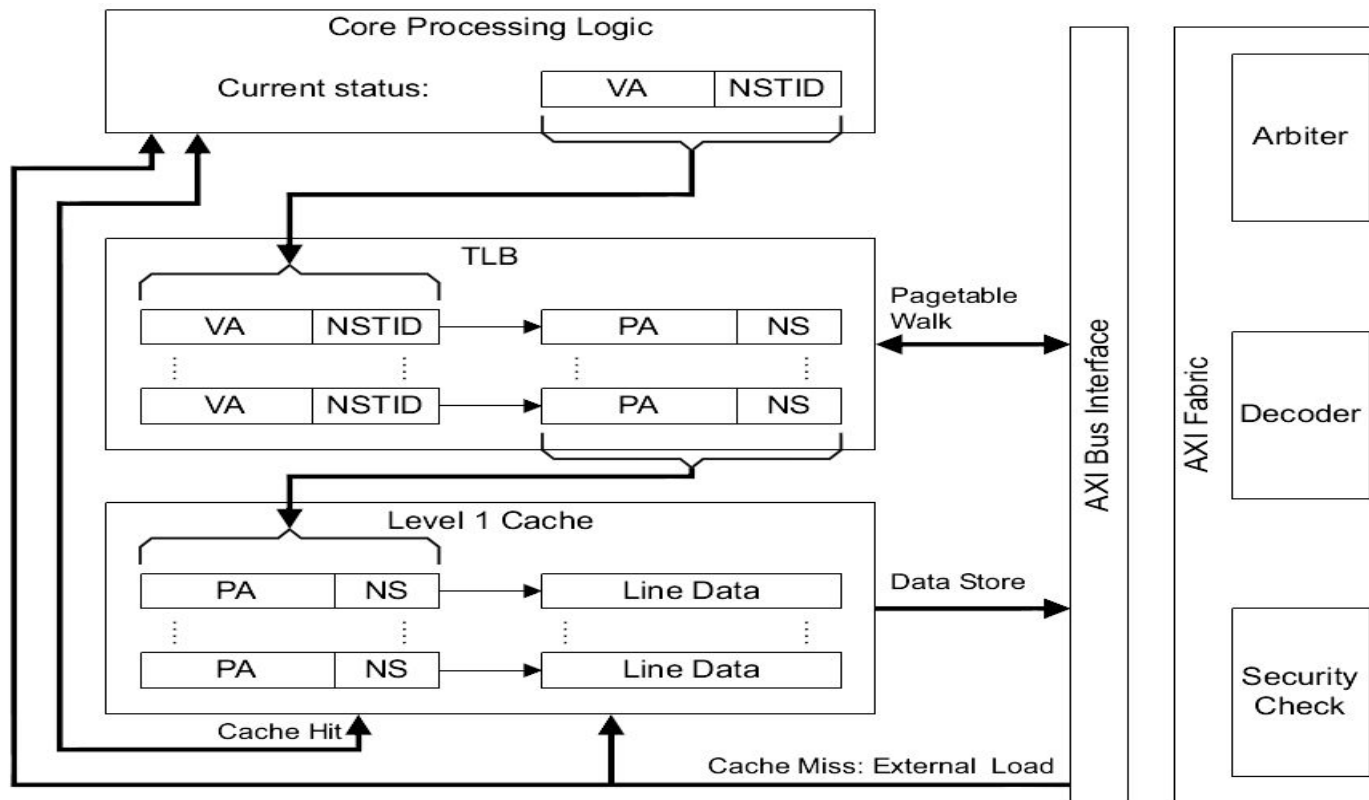
1 ARM may add tags to entries of TLBs(Translation Lookside Buffers)

2 Cache, add tag bit of security state, non flush when context switching

# 安全memory以及非安全memory映射图



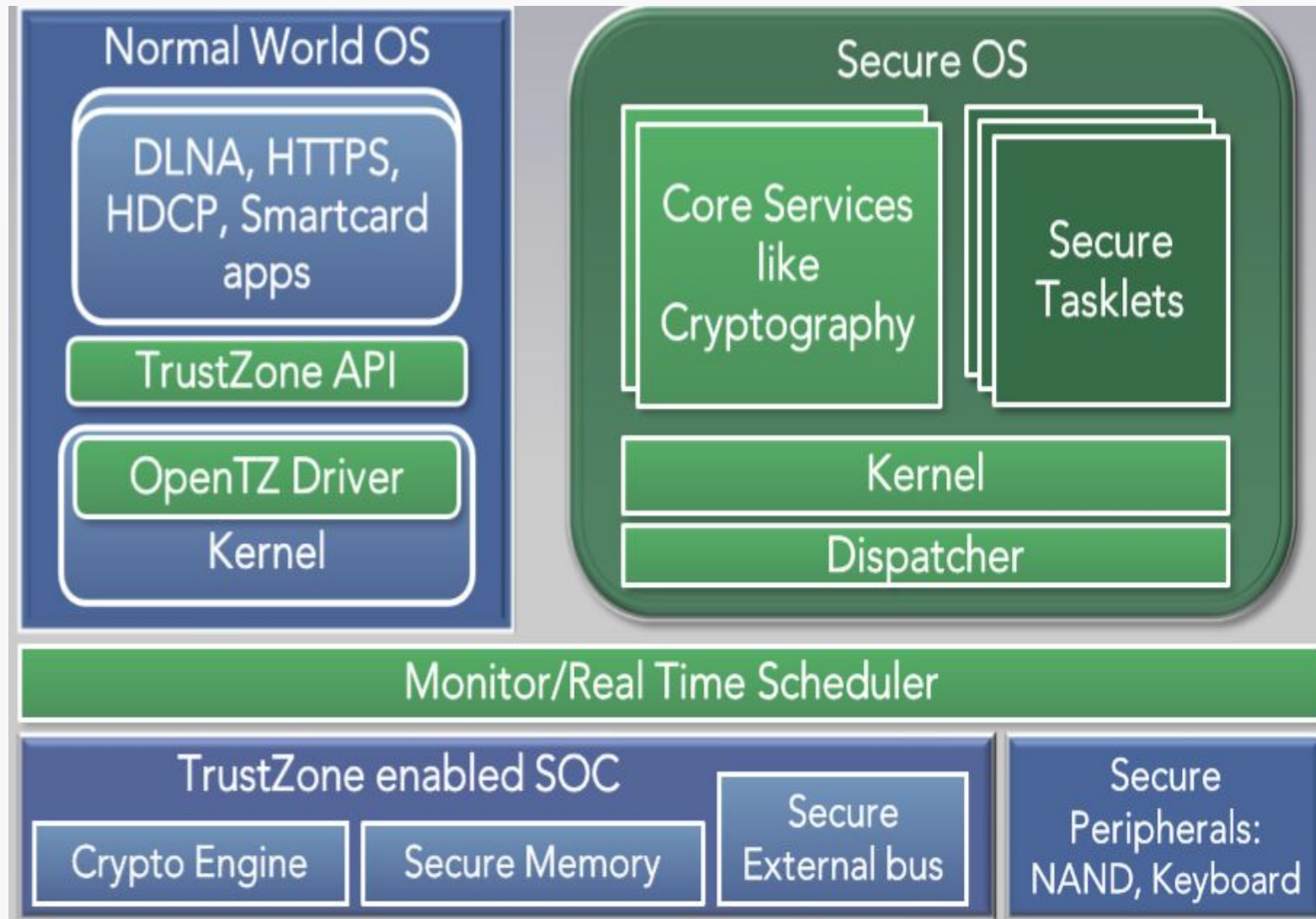
# Example



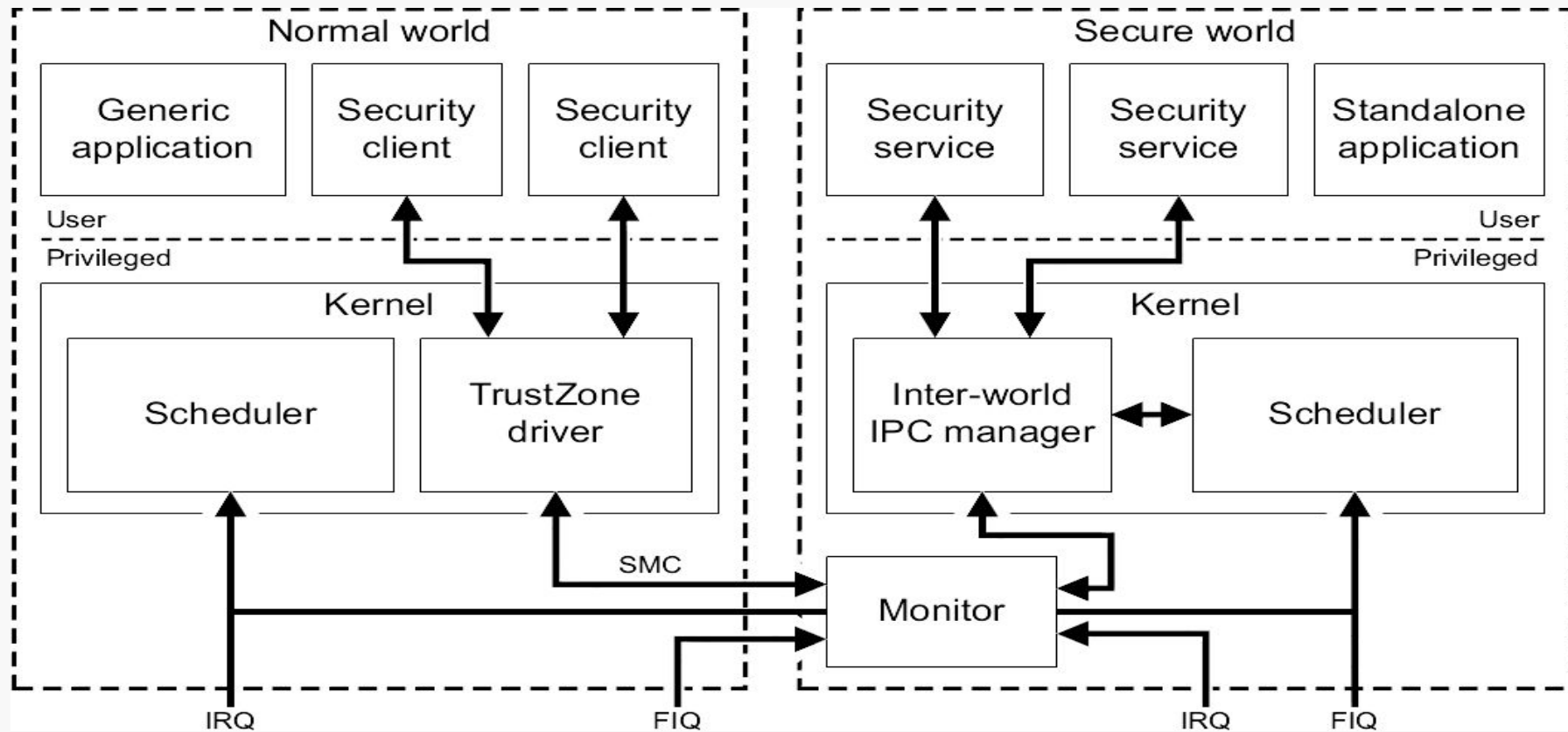
- 1 core processing logic attempts data load, HW passes VAddr and NSTID to TLBs
- 2 TLBs loads PhyAddr and NS associated with Vaddr and NSTID
- 3 cache return data or load data from external memory.



# Secure OS frame diagram



# Switching of two world



# Fastmodels-ARM建模软件

The screenshot displays the Fastmodels-ARM software interface. The main window is titled "RTSM\_VE\_Cortex\_A15x1.lisa". It features a "Components" tab, which is active, showing a list of components. The list includes various ARM components, with "ARM1176CT" highlighted. The interface also includes a "Source" tab and a "Block Diagram" tab. A status bar at the bottom indicates the loading progress: "Loading project file", "Loading components and protocols", and "Loading of models completed".

Component Name	Version	Type	File
AMBAPV2PVBUS	7.1.71	Bus	/home/himiko/ARM/Fast...
AMBAPVACE2PVBUS	7.1.71	Bus	/home/himiko/ARM/Fast...
AMBAPVSignal2SGSignal	7.1.71	Other	/home/himiko/ARM/Fast...
AMBAPVSignalState2SGStateSignal	7.1.71	Other	/home/himiko/ARM/Fast...
AMBAPVValue2SGValue	7.1.71	Other	/home/himiko/ARM/Fast...
AMBAPVValue2SGValue64	7.1.71	Other	/home/himiko/ARM/Fast...
AMBAPVValueState2SGValueState	7.1.71	Other	/home/himiko/ARM/Fast...
AMBAPVValueState2SGValueState64	7.1.71	Other	/home/himiko/ARM/Fast...
ARM926CT	7.1.71	Core	/home/himiko/ARM/Fast...
ARM968CT	7.1.71	Core	/home/himiko/ARM/Fast...
ARM1136CT	7.1.71	Core	/home/himiko/ARM/Fast...
ARM1176CT	7.1.71	Core	/home/himiko/ARM/Fast...
ARMCortexA5CT	7.1.71	Core	/home/himiko/ARM/Fast...
ARMCortexA5MPx1CT	7.1.71	Core	/home/himiko/ARM/Fast...
ARMCortexA5MPx2CT	7.1.71	Core	/home/himiko/ARM/Fast...
ARMCortexA5MPx4CT	7.1.71	Core	/home/himiko/ARM/Fast...
ARMCortexA7x1CT	7.1.71	Core	/home/himiko/ARM/Fast...

Source Block Diagram

Loading project file  
Loading components and protocols  
Loading of models completed

# **My work** Emulating arm TrustZone in QEMU

## **Emulating arm TrustZone in QEMU**

- **we want to emulate trust zone in QEMU, but Trust zone is hardware virtualization found in real target.**

**so we need to answer:**

- **1 how to extend arm emulator(QEMU) to support trust zone feature to develop quality operating secure system**
- **2 how to do with secure monitor call and support context switching context.**

# My answer for prev problem

## 1 Modeling arm trust zone peripherals in an QEMU platform:

- ARM Security Technology (PRD29-GENC-009492C)
- ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition (DDI0406C)
- **ARM TrustZone Protection Controller (BP147)** Technical Overview (DTO0015)

# My answer for prev problem

## 2 critical problem

we need to Implement addition to kernel at least

- Secure boot
- Montor code

1 context swiching

2 interrupt model

more changes to kernel

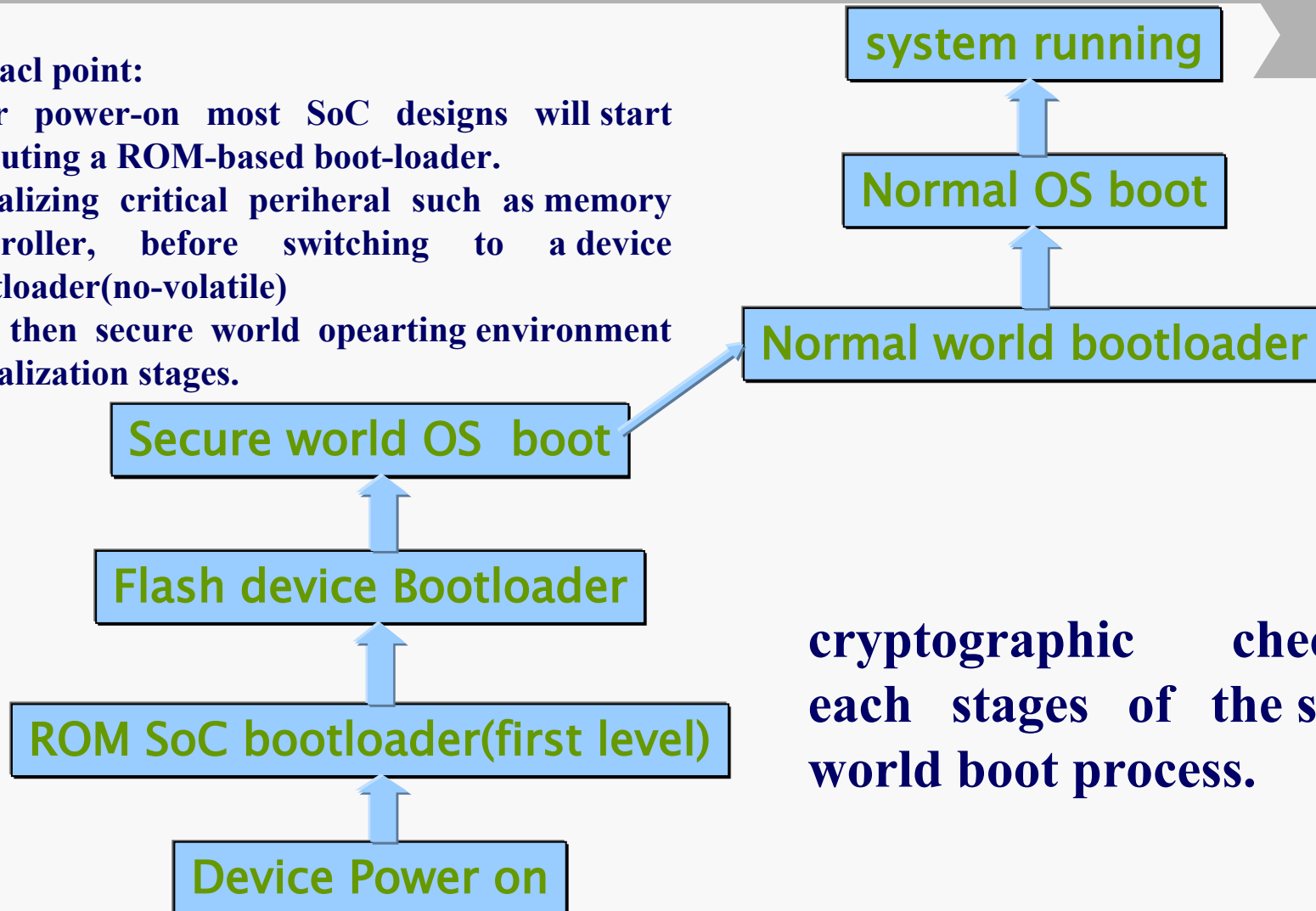
# Secure boot

critical point:

after power-on most SoC designs will start executing a ROM-based boot-loader.

initializing critical peripheral such as memory controller, before switching to a device bootloader(no-volatile)

and then secure world operating environment initialization stages.



cryptographic checks in each stages of the secure world boot process.

# Secure boot-QEMU Implement

## ➤ secure boot implementation function

### void smc\_nscpu\_context\_init(void)

```
for (k = 0; k < blocks; k++) {
    ret_blkcnt = block_dev->block_read(block_dev->dev,
                                       k+SECURE_OS_MMC_START_SECTOR, 1,
                                       (void*)((u32)g_buffer + k * 512));

    if(ret_blkcnt != 1) {
        sw_printf("error in reading block 0x%x\n", k);
    }

#ifdef SW_BL_DBG
    if( k == 0 || k == blocks -1) {
        for(i=0; i<32; i++){
            sw_printf(" %02x |", *(g_buffer + k * 512 + i));
            if(j++ == 4){
                j=0;
                sw_printf("!\\n");
            }
        }
        sw_printf("!\\n");
    }
#endif
}
```



# Monitor design

In most designs, monitor code is similar to OS context-switch, monitor make sure that state of current world is safely saved (cannot be accessed by normal world), then switches to world correctly restored.

rule we'd better follow:

1 normal world entry to monitor mode is tightly only possible entry exceptions,

➤ processor hardware configured to trap exceptions (such as external abort, or an explicit call or an interrupt).

➤ software exception: SMC, compared to full context switch of hardware, SMC can carry some message in processor. Efficient software communications can be built via SMC initiated context switches and shared-memory.

while secure world entry to monitor mode are more flexible, for secure software can directly write to CPSR.

## Monitor mode

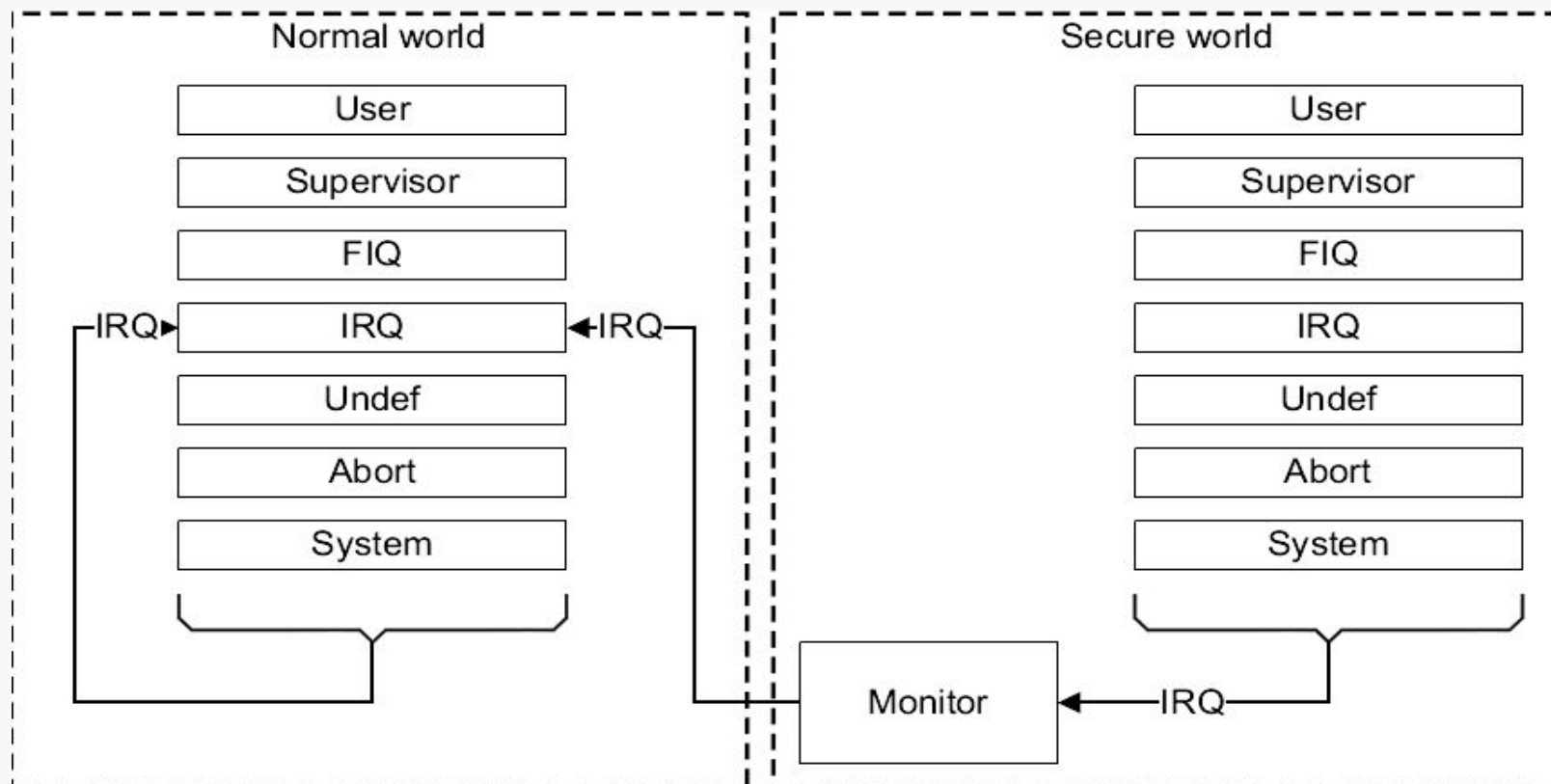
During the transfer of data from normal memory region to secure memory region, we need to do two things.

- check for authenticity of the code, i.e. whether application is trusted one or not
- If application is trusted, then change the mode of the bit and cp15 register.

	Monitor Mode	Supervisor Mode	Non Secure bit	Secure bit
Application in Non secure region	0	1	1	0
Application in Secure region	1	1	0	1

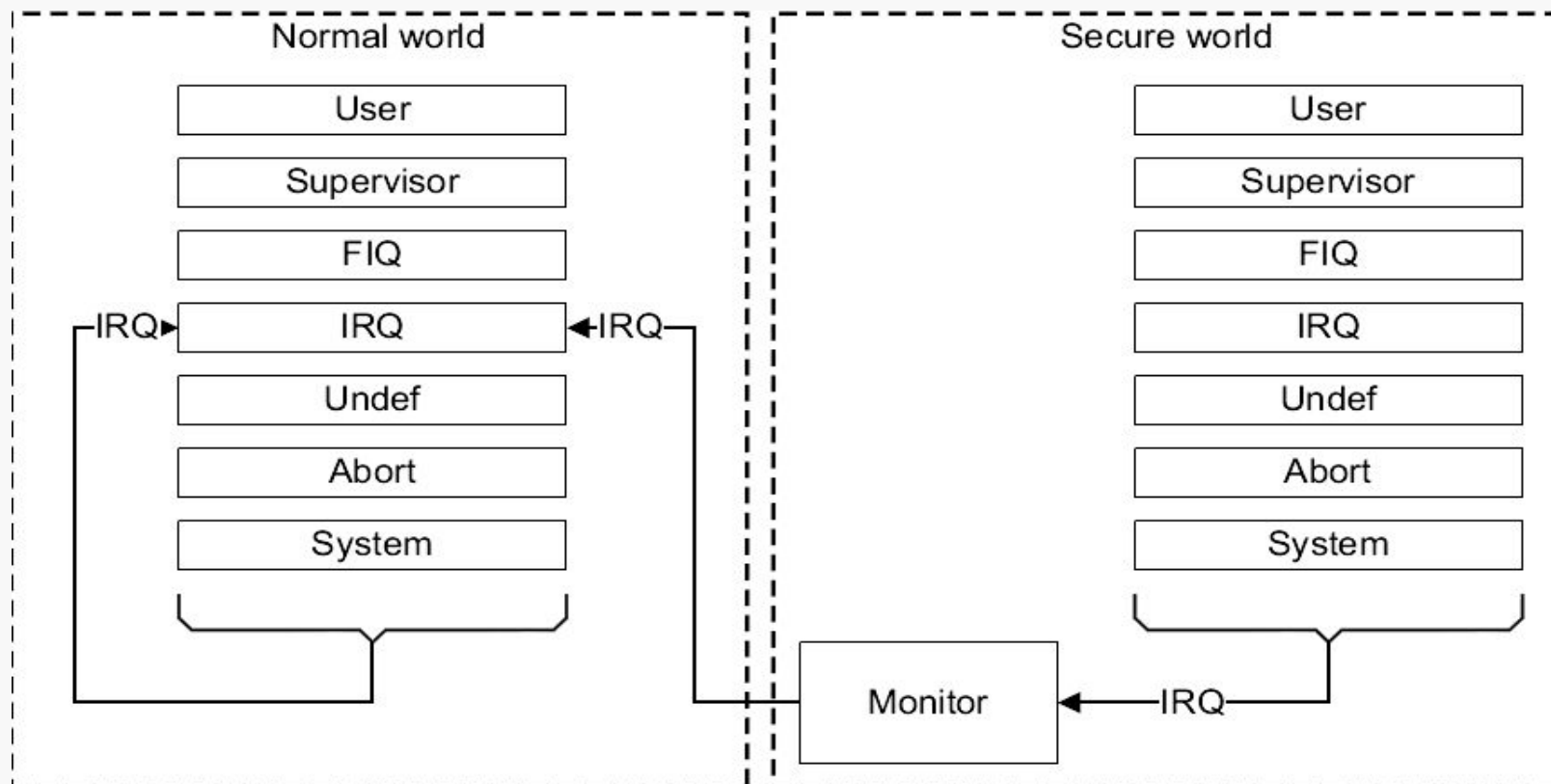
# Monitor - Interrupt model

we use mode ARM recommended that IRQ serve as Normal world interrupt source, while FIQ serve as Secure world interrupt source.



# Monitor - Interrupt model

we use mode ARM recommended that IRQ serve as Normal world interrupt source, while FIQ serve as Secure world interrupt source.



## Monitor - Interrupt model

Normal world cannot access to configuration register(F and A) in CPSR, Monitor mode and Secure mode can modifying F(FIQ) and A(Abort).

### Processor exception vector table:

TrustZone-enabled processor implements three sets of exception vector tables. One for normal world, one is for Secure world, and the other is for Monitor mode.

we can program the Vector Base Address Register to change location of each table.

## Monitor SMC call- QEMU Implement

In QEMU emulator, we implement SMC not in a SMC instruction, but make use of a function to implemtn function.

```
int scm_call (unsigned char cmd_addr){
    unsigned char context_id;
    unsigned char r0 = 1;
    unsigned char r1 = (unsigned char) & context_id;
    unsigned char r2 = cmd_addr;
    printf ("r0: %d\n", r0);
    printf ("r1: %d\n", r1);
    printf("r2:%d\n", r2);
    //__asm__ ("1: smc #0 @ switch to secure world\n" "cmp    r0, #1\n"
    "beq 1b\n": "=r"(r0): "r"(r0), "r"(r1),    "r"(r2):"r3","cc");
    return r0;
}
```

# Monitor - QEMU Implement

new Instruction “SMC” helps us to switch between secure and non secure world

➤ `smc_id.h` Header file for SMC identifiers

➤ `smc_wrapper.c`

`void __execute_smc (u32 smc_arg)`

smc system call implementation

`void invoke_ns_kernel (void)`

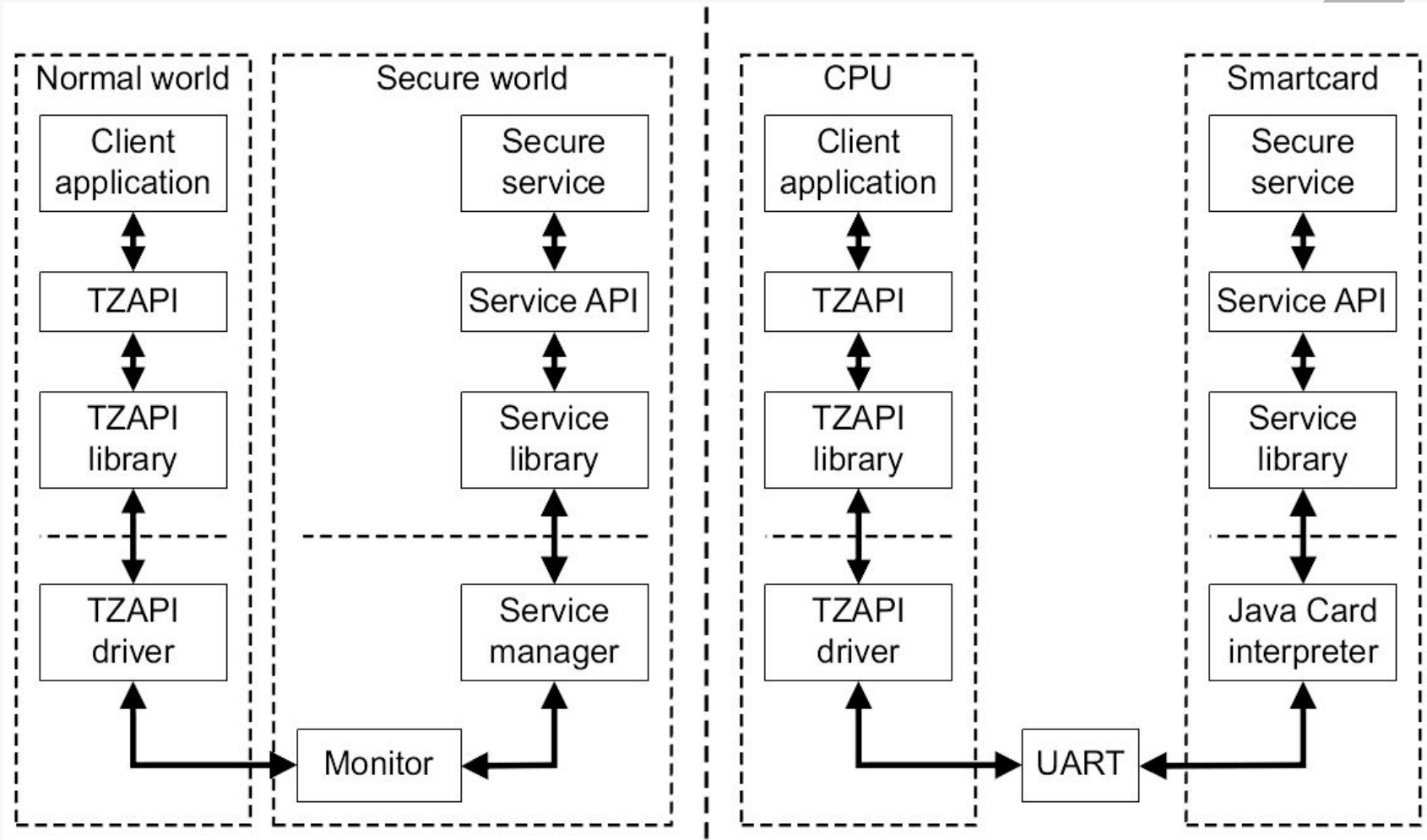
Calls smc function with appropriate argument and hence non-secure kernel gets invoked on reaching the non-secure side.

`void return_secure_api (u32 retval)`

Calls smc function with appropriate return value obtained by executing the secure api

`void smc_nscpu_context_init (void)`

# TrustZone API





# ARM trust zone Emulated Architecture

	ARM Trust Zone	Emulated Architecture
Secure memory Management	Creating during booting	Creating when SCM call comes
Moniter	NS and S bit value changes according to transit between the worlds	NS and S bit value changes according to transit between the worlds
trusted apps	Loaded during boot time and it's not dynamic	Loaded during execution time and its dynamic
Interrupts	Operating system take care of it	Operating system take care of it
SCM call	ARM instruction	Procedure call
Debugging	We cannot debug application since its already compiled	
Registers	Registers	Variables

# Work summary and Future work:

## Work summary:

- **Systematic analysis of TrustZone Technology**
- **Design architecture of Seucore Embedded system**  
**hardware architecture**  
**software architecture**
- **Specific analysis of hardware subsystem, TZASC and son.**  
**Implement hardware design in emulator (OVP). and write**  
**small program to test.**
- **Implement of software subsystem, secure bootloader and**  
**monitor code and more changes to Linux OS.**

## Future work:

- **More features of TrustZone to system.**
- **Implement Security Check with RSA.**