고려대학교
KOREA UNIVERSITY

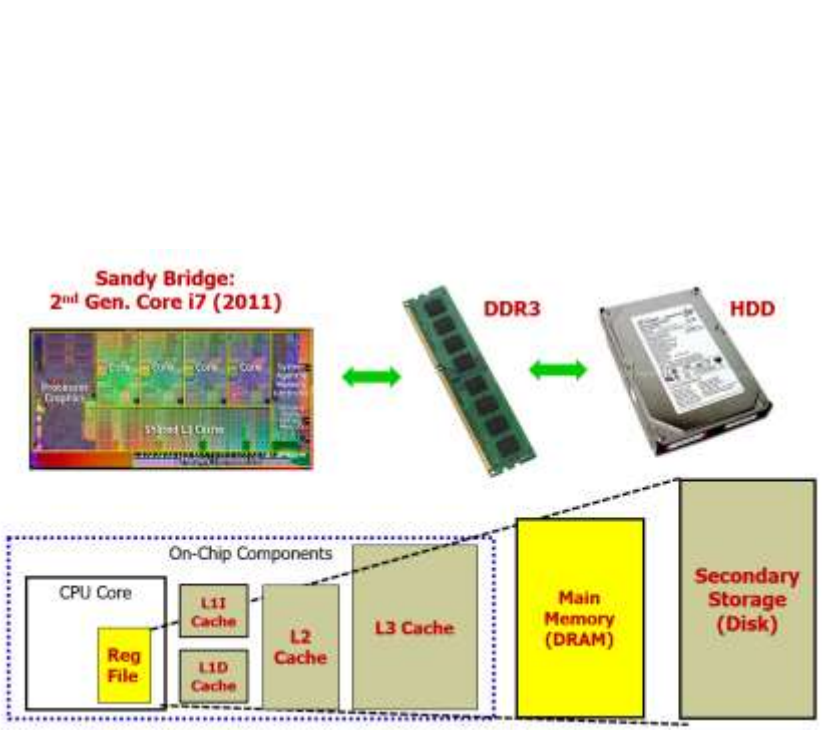
COSE321 Computer Systems Design

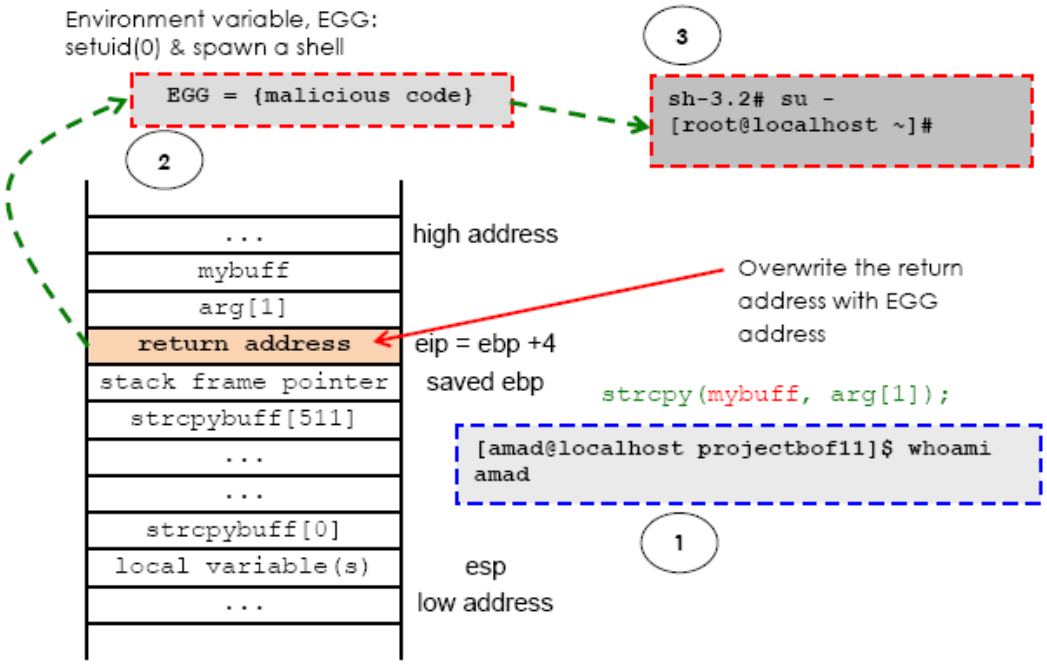
Lecture 12. TrustZone

Prof. Taeweon Suh
Computer Science & Engineering
Korea University

Hacking

- Exploit vulnerabilities in software
  - Classic buffer overflow
  - Heap-based overflow
  - Function pointer overflow …



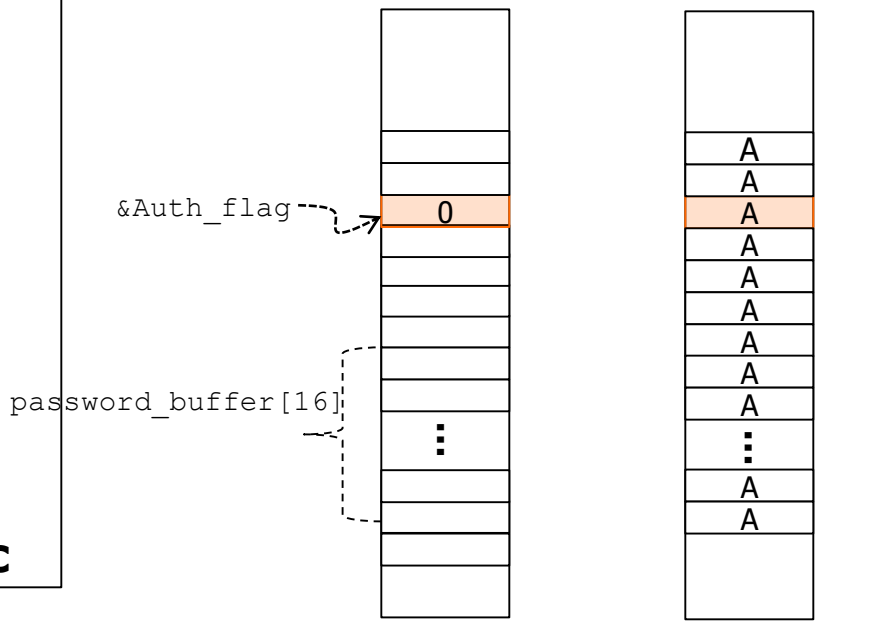


Buffer Overflow Example

```
int check_authentication(char *password) {
        int auth_flag = 0;
        char password_buffer[16];

        strcpy(password_buffer, password);

        if (strcmp(password_buffer, "Tom") == 0)     auth_flag = 1;

        if (strcmp(password_buffer, "Jerry") == 0)     auth_flag = 1;

        return auth_flag;
}


int main(int argc, char *argv[]) {

        if ( check_authentication(argv[1]) )  {
                printf("       Access Granted.\n");
        } else {
                printf("\nAccess Denied.\n");
        }

}
```

auth_overflow.c

```
$./auth_overflow   test
  Access Denied

$./auth_overflow   Tom
  Access Granted

$./auth_overflow   AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
  Access Granted
```



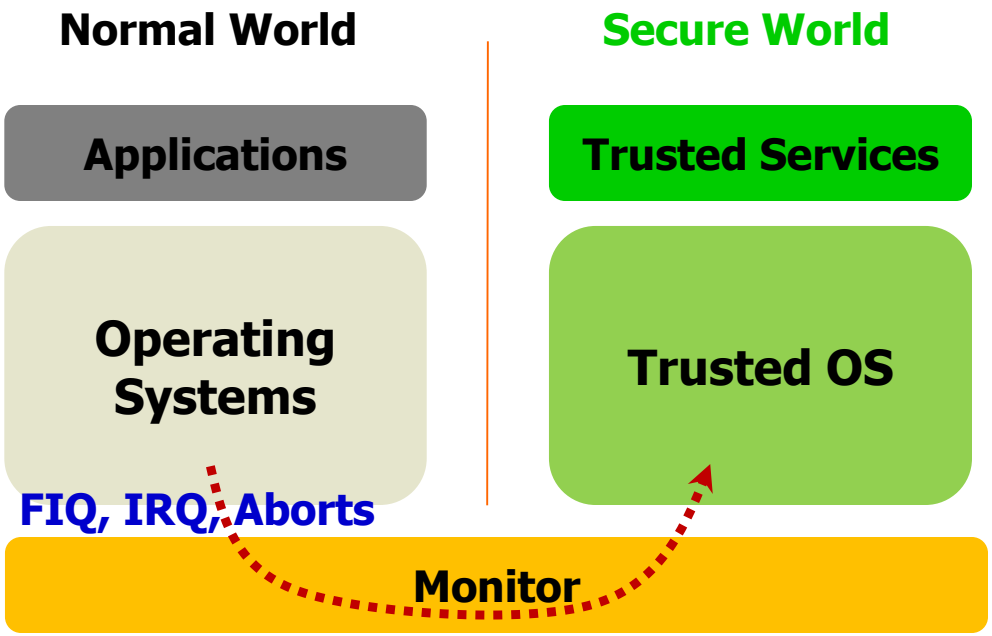
Why TrustZone?

- Modern mobile devices do a lot more than making phone calls
  - Music, movie, games, internet surfing, banking services and so on
- Complex open software is historically much more vulnerable to hacking attacks
  - How do you make sure that your newly installed applications are safe to use?
- Nevertheless, some activities require us to trust the system
  - Banking and payment services
- TrustZone provides a hardware mechanism for trusted execution environment (TEE)

TrustZone

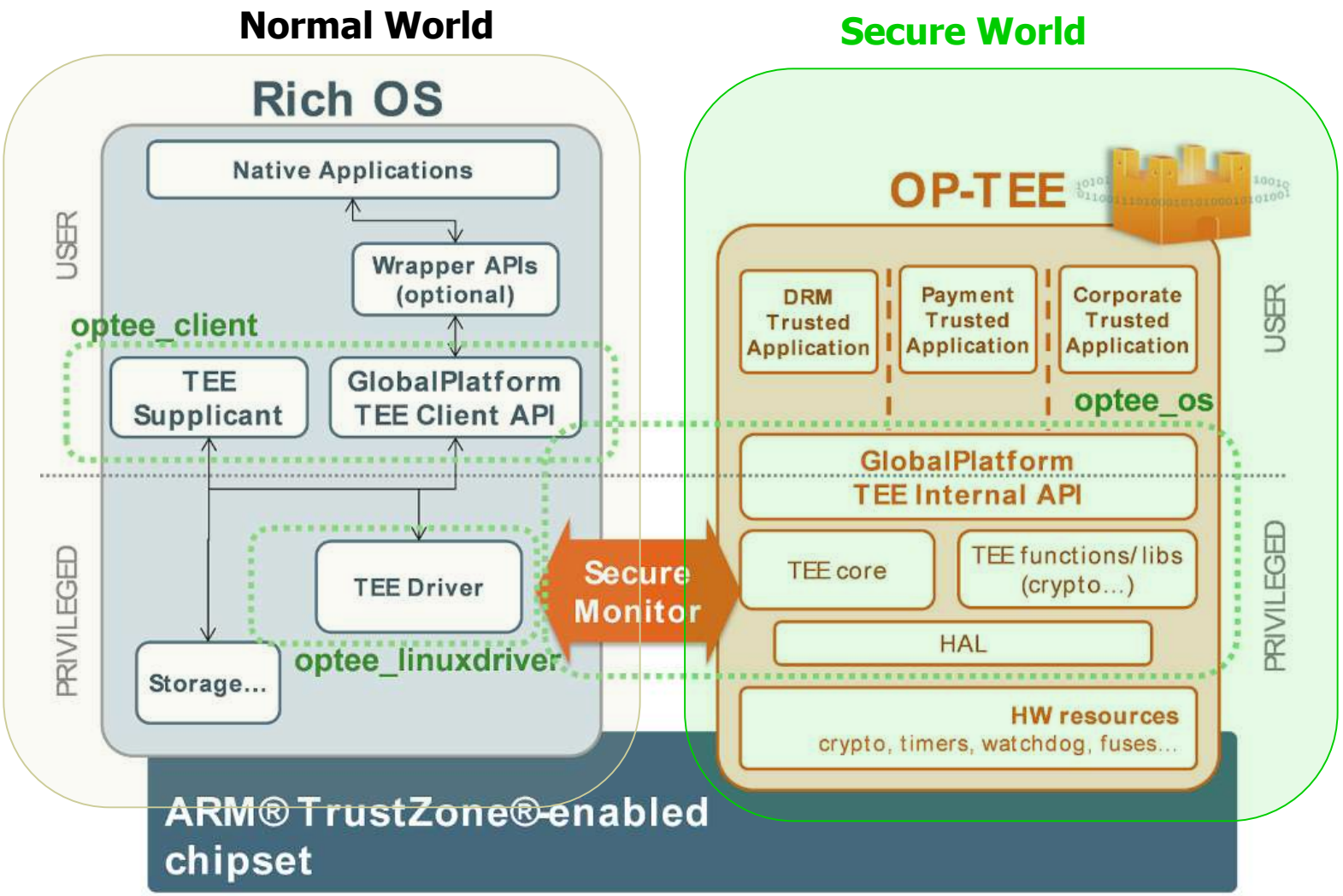
- 2 worlds: **Secure** and **Normal** Worlds
  - Each world has its own vector table, page tables (each world has its own Virtual address space) and system configuration registers
  - Monitor mode acts as a gatekeeper between 2 worlds

- 2 physical address spaces (controller by `SCR'NS`)
  - S:0x1000 treated as different physical location from NS:0x1000



Example

- OP-TEE: Open Portable Trusted Execution Environment



https://linuxgizmos.com/files/linaro_optee_arch.jpg

https://www.op-tee.org/

Privilege Levels & Security States

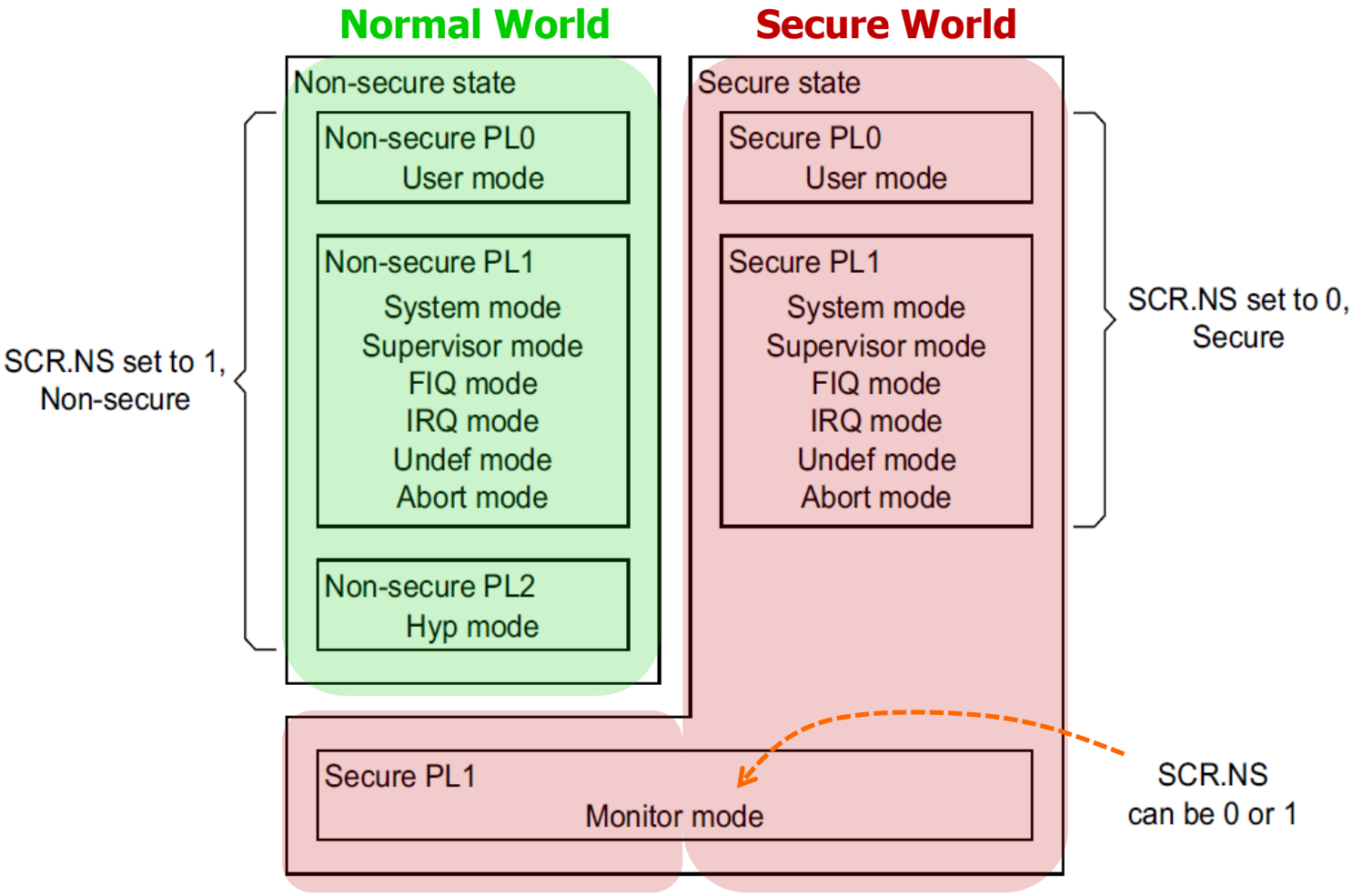


Figure B1-1 Modes, privilege levels, and security states

Source: ARMv7 Architecture Reference Manual

ARM Modes

Format of the CPSR and SPSRs

The CPSR and SPSR bit assignments are:

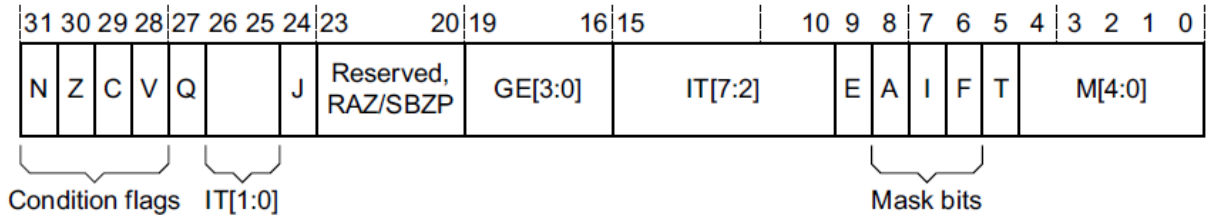
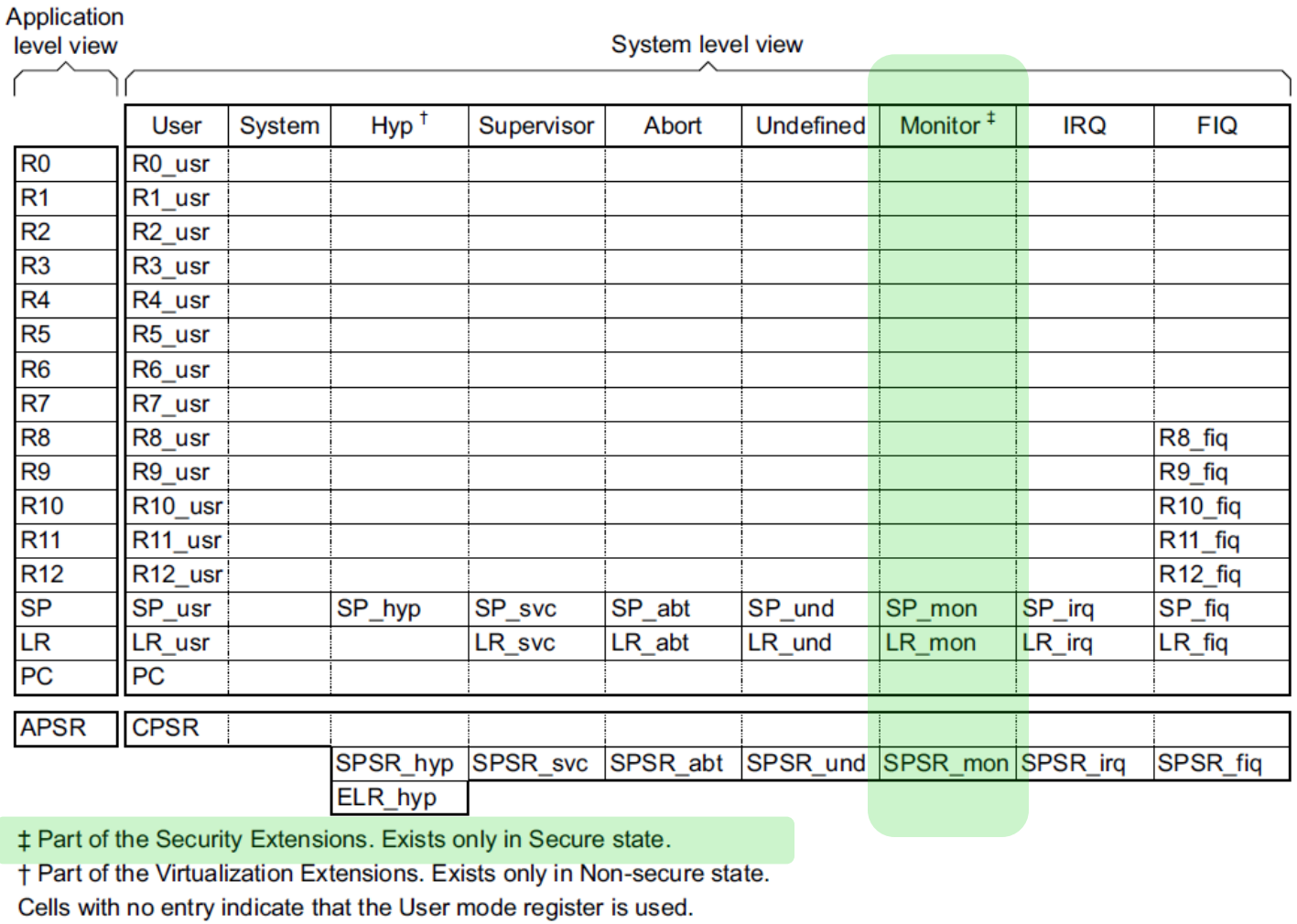


Table B1-1 ARM processor modes

Processor mode		Encoding	Privilege level	Implemented	Security state
User	usr	10000	PL0	Always	Both
FIQ	fiq	10001	PL1	Always	Both
IRQ	irq	10010	PL1	Always	Both
Supervisor	svc	10011	PL1	Always	Both
Monitor	mon	10110	PL1	With Security Extensions	Secure only
Abort	abt	10111	PL1	Always	Both
Hyp	hyp	11010	PL2	With Virtualization Extensions	Non-secure only
Undefined	und	11011	PL1	Always	Both
System	sys	11111	PL1	Always	Both

ARM Registers



Application level view

System level view

	User	System	Hyp †	Supervisor	Abort	Undefined	Monitor ‡	IRQ	FIQ
R0	R0_usr								
R1	R1_usr								
R2	R2_usr								
R3	R3_usr								
R4	R4_usr								
R5	R5_usr								
R6	R6_usr								
R7	R7_usr								
R8	R8_usr								R8_fiq
R9	R9_usr								R9_fiq
R10	R10_usr								R10_fiq
R11	R11_usr								R11_fiq
R12	R12_usr								R12_fiq
SP	SP_usr		SP_hyp	SP_svc	SP_abt	SP_und	SP_mon	SP_irq	SP_fiq
LR	LR_usr			LR_svc	LR_abt	LR_und	LR_mon	LR_irq	LR_fiq
PC	PC								
APSR	CPSR								
			SPSR_hyp	SPSR_svc	SPSR_abt	SPSR_und	SPSR_mon	SPSR_irq	SPSR_fiq
			ELR_hyp						

‡ Part of the Security Extensions. Exists only in Secure state.

† Part of the Virtualization Extensions. Exists only in Non-secure state.

Cells with no entry indicate that the User mode register is used.

Figure B1-2 ARM core registers, PSRs, and ELR_hyp, showing register banking

Source: ARMv7 Architecture Reference Manual

Vector Tables

- In a TrustZone system, there are 3 vector tables:
  - Tables for Secure world (VBAR), Normal world (VBAR), and Secure Monitor (MVBAR): so, VAR is a banked register
  - Only the Secure vector table has a defined reset value (0x0 or 0xFFFF_0000)
  - `SMC` always vectors to the Secure Monitor table

- FIQ (0x1C), IRQ (0x18), and Abort (Data abort (0x10 or prefetch abort (0x0C)) exceptions can vector to the **current world's table** or the **Secure Monitor's table**
  - It is configured via SCR (Secure Configuration Register)

Vector Tables

Table B1-3 The vector tables

Offset	Vector tables			
	Hyp[a]	Monitor[b]	Secure	Non-secure
0x00	Not used	Not used	Reset	Not used
0x04	Undefined Instruction, from Hyp mode	Not used	Undefined Instruction	Undefined Instruction
0x08	Hypervisor Call, from Hyp mode	Secure Monitor Call	Supervisor Call	Supervisor Call
0x0C	Prefetch Abort, from Hyp mode	Prefetch Abort	Prefetch Abort	Prefetch Abort
0x10	Data Abort, from Hyp mode	Data Abort	Data Abort	Data Abort
0x14	Hyp Trap, or Hyp mode entry[c]	Not used	Not used	Not used
0x18	IRQ interrupt	IRQ interrupt	IRQ interrupt	IRQ interrupt
0x1C	FIQ interrupt	FIQ interrupt	FIQ interrupt	FIQ interrupt

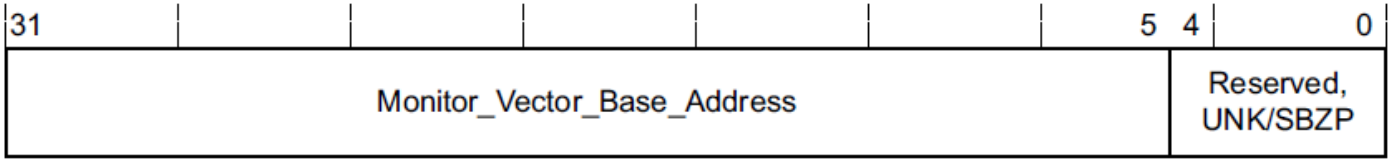
a. Non-secure state only. Implemented only if the implementation includes the Virtualization Extensions.
b. Secure state only. Implemented only if the implementation includes the Security Extensions.
c. See *Use of offset 0x14 in the Hyp vector table* on page B1-1167.

Table B1-4 Modes for taking exceptions using the Secure or Non-secure vector table

MVBAR

- Monitor Vector Base Address Register (MVBAR) holds the exception base address for all exceptions taken to Monitor mode
  - Only accessible from Secure PL1 modes

The MVBAR bit assignments are:



Monitor_Vector_Base_Address, bits[31:5]

Bits[31:5] of the base address of the exception vectors for exceptions that are taken to Monitor mode. Bits[4:0] of an exception vector is the exception offset, see Table B1-3 on page B1-1166.

Bits[4:0] Reserved, UNK/SBZP.

```
MRC p15, 0, <Rt>, c12, c0, 1  ; Read MVBAR into Rt
MCR p15, 0, <Rt>, c12, c0, 1  ; Write Rt to MVBAR
```

Source: ARMv7 Architecture Reference Manual

Secure/Non-Secure Vector Tables

- VBAR (Vector Base Address Register) is a banked register

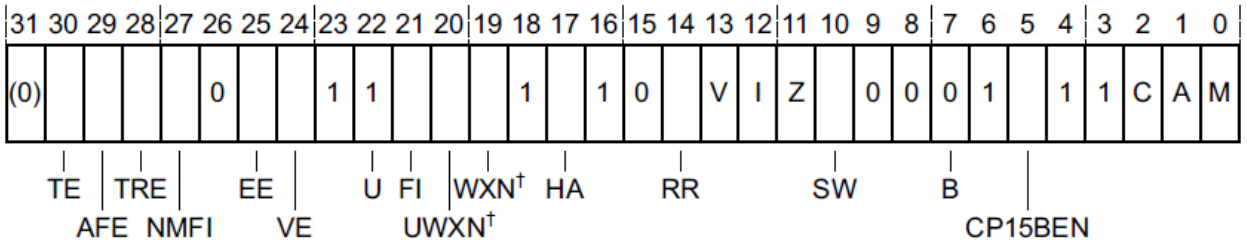
- The Secure SCTLR.V determines the exception base address
  - If V == 0, the secure VBAR holds the base
  - If V == 1, base = 0xFFFF_0000

- The Non-Secure SCTLR.V determines the exception base address
  - If V == 0, the non-secure VBAR holds the base
  - If V == 1, base = 0xFFFF_0000

System Control Register (SCTLR)

- SCTLR provides the top level control of the system

In a VMSAv7 implementation, the SCTLR bit assignments are:

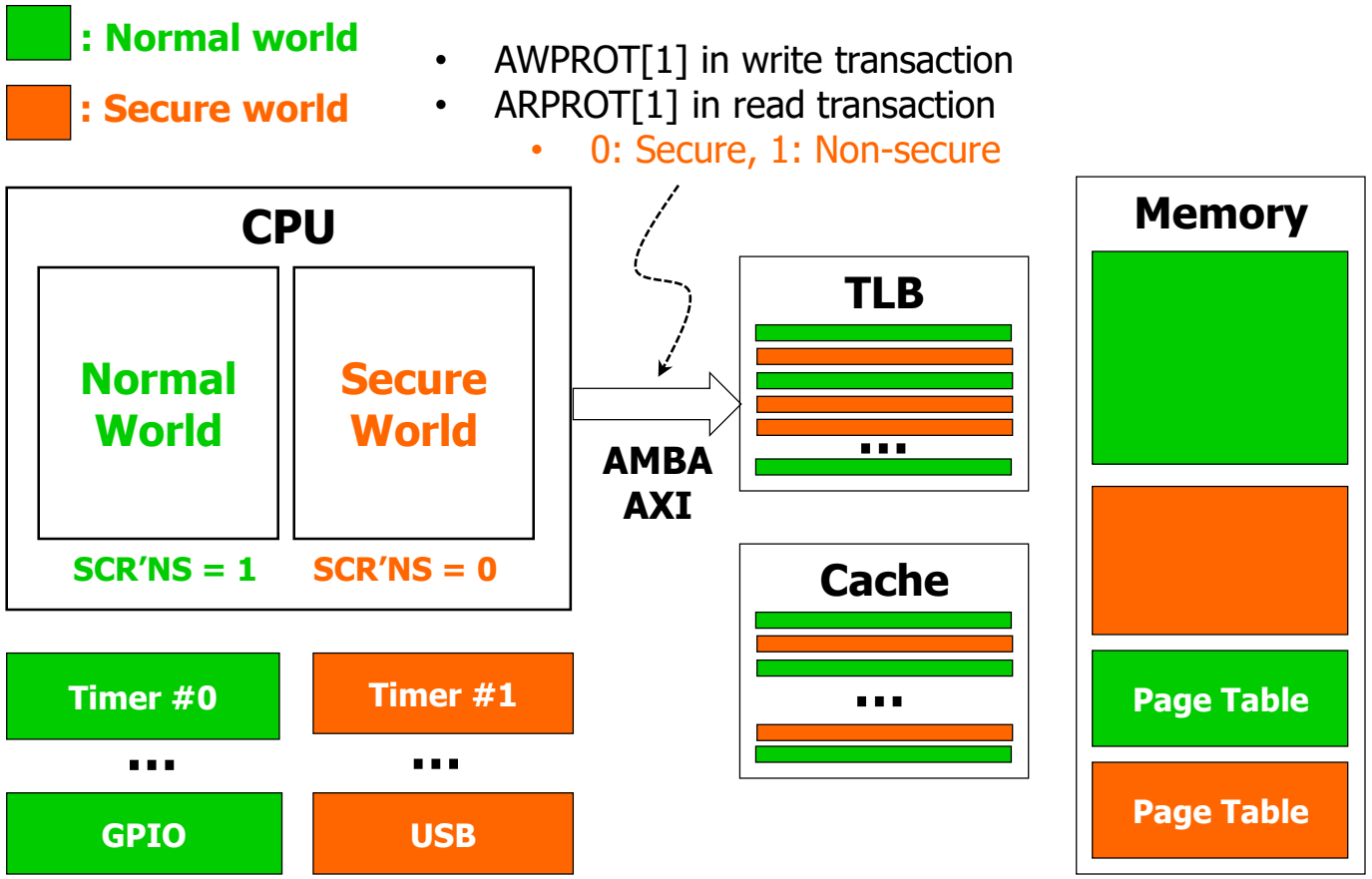


† Reserved before the introduction of the Virtualization Extensions, see text for more information.

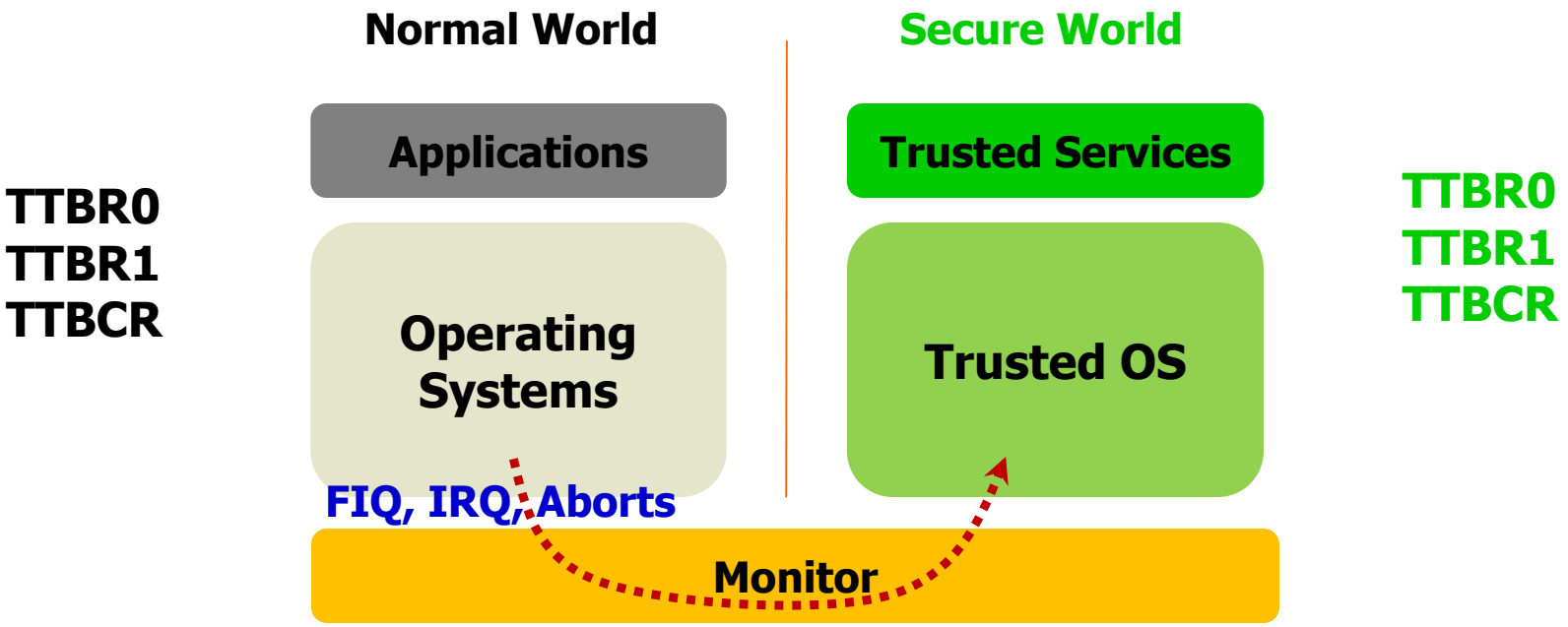
- **TE: Thumb Exception Enable**
- **AFE: Access Flag Enable**
  0: In the translation table descriptors, AP[0] is an access permission bit
  1: In the translation table descriptors, AP[0] is an access flag
- **TRE: TEX remap enable**
  0: TEX remap disabled. TEX[2:0] are used with the C and B bits to describe memory region attributes
  1: TEX remap enabled. TEX[2:1] are reassigned for use as bits managed by OS. The TEX[0], C and B bits, with the MMU remap registers describe the memory region attributes
- **VE: Interrupt Vectors Enable**
- **RR: Round Robin select. Cache replacement policy**
- **V: Vectors bit. 0: Low vectors 0x0, 1: High vectors (Hivecs): 0xFFFF0000**
- **I: I$ enable**
- **Z: Branch prediction enable**
- **CP15BEN: CP15 barrier enable**
- **C: $ enable**
- **A: ~~Alignment check enable~~**
- **M: MMU enable**

TrustZone Overview

- AWPROT[2], ARPROT[2]: 0: data access, 1: Instruction access
- AWPROT[1], ARPROT[1]: 0: Secure, 1: Non-secure
- AWPROT[0], ARPROT[0]: 0: Unprivileged access, 1: Privileged Access

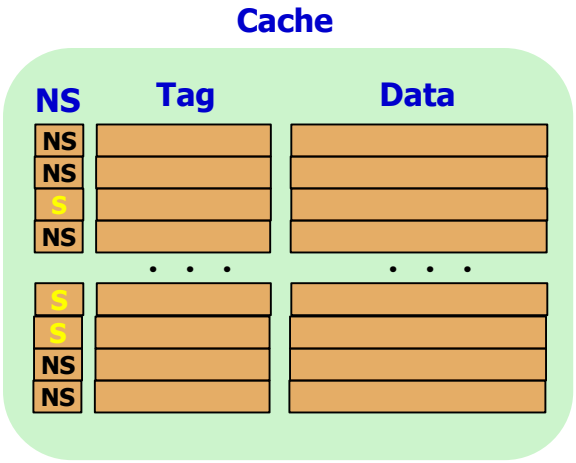


Banked Registers



TrustZone in Caches & TLBs

- Secure and Non-secure entries can coexist within caches and TBLs
  - No need to flush on transition between worlds
- Caches record which world a line belongs to
  - Normal world can only generate non-secure accesses, so it will hit only on cache lines marked as non-secure
  - Secure world can generate both secure and non-secure accesses
- TLBs record which world generated an entry
  - TLB entries are NEVER shared between worlds



Memory Management

- TTBR0, TTRB1, TTBCR, and SCTLR.M (MMU enable) are **banked** between the worlds (see backup slides for more banked systems registers)
  - Each world has independent translation tables and its own view of virtual memory
  - Page table walks are made to the physical address space corresponding to the security state
- TrustZone introduces the NS (non-secure) memory attribute
  - It distinguishes between secure and non-secure physical address spaces
  - It is reflected on the bus using `AXI AxPROT[1]`
- **Normal world** signals all accesses as **non-secure**
- **Secure world** can signal access as **secure** or **non-secure**, controlled by the translation table entries (page tables) – NS bit in each entry

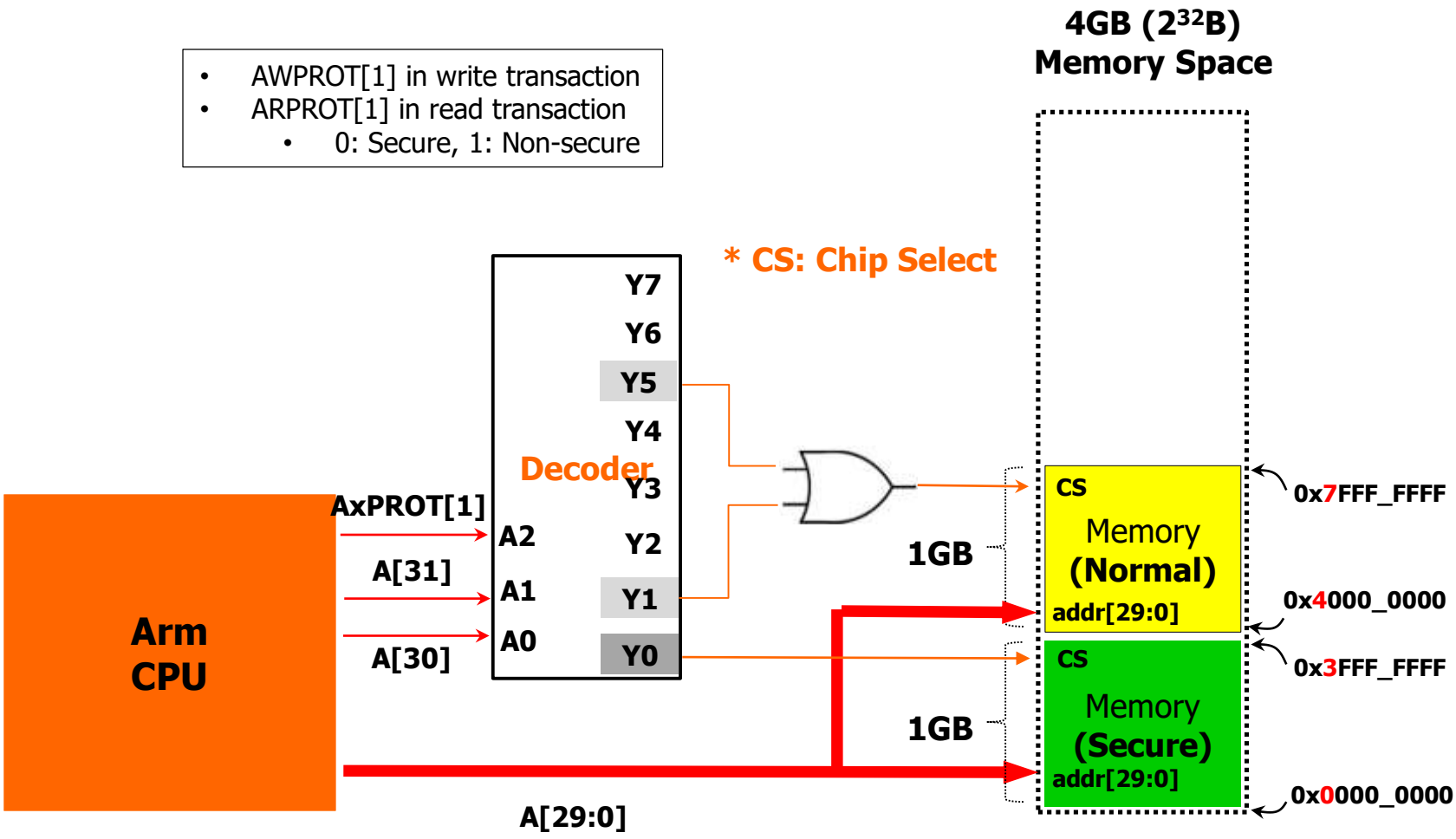
TrustZone in Memory and I/Os

- Secure/Non-Secure Configuration in Zynq

- In addition to Cortex-A9, the Zynq also provides a Secure/Non-Secure configurable option for all the hard I/O peripherals and AXI interconnects

- The TrustZone control registers can be used to **configure** the Secure or Non-Secure mode for the following PS devices:
  - All I/O Peripherals (IOP)
  - OCM RAM
  - DMA Controller and Interrupts
  - DDR Controller
  - All PS Interconnects to PL
    - General-purpose (GP) Master/Slave Interconnects
    - High-performance (HP) Slaves
    - Accelerator Coherency Ports (ACP)

TrustZone Hardware System Example: Memory



Moving Between the Worlds

- Transitions between the worlds should pass through the Monitor mode

  1. `SMC` (Secure Monitor Call) instruction always causes core to enter Monitor mode
  2. Monitor mode is normally entered via an exception
     - IRQs, FIQs, and some aborts are configured to enter the Monitor mode (see SCR)

  - Secure world can move directly to the Normal world by setting the SCR.NS bit, but it is **not** recommended

- Monitor mode is in Secure state regardless of the SCR.NS bit
  - Except in Monitor mode and Hyp mode, the security state is controlled by SCR.NS
  - Operations on banked CP15 registers will access Normal world if SCR'NS == 1

- Context switching is typically performed in the Monitor mode
  - Most **system registers** are automatically banked by the hardware
  - **General-purpose registers** must be handled in the Monitor mode

Source: ARMv7 Architecture Reference Manual, p1157

Moving Between the Worlds

- The usual mechanism for changing **from Secure to Non-Secure** is an exception return
  - To return to Non-Secure state, software executing in Monitor mode sets SCR.NS to 1, and then performs the exception return (`movs pc, lr`)

Source: ARMv7 Architecture Reference Manual, p1157

SMC (Secure Monitor Call)

B9.3.14 SMC (previously SMI)

Secure Monitor Call causes a Secure Monitor Call exception. For more information see *Secure Monitor Call (SMC) exception* on page B1-1211.

SMC is available only from software executing at PL1 or higher. It is UNDEFINED in User mode.

Encoding A1 Security Extensions

SMC<c> #<imm4>

31 30 29 28	27 26 25 24	23 22 21	20	19 18 17 16 15 14 13 12 11 10 9 8	7 6 5 4	3 2 1 0
cond	0 0 0 1	0 1 1	0	(0) (0) (0) (0) (0) (0) (0) (0) (0) (0) (0) (0)	0 1 1 1	imm4

For the case when cond is 0b1111, see *Unconditional instructions* on page A5-216.

```
imm32 = ZeroExtend(imm4, 32);
// imm32 is for assembly/disassembly only and is ignored by hardware
```

A Secure Monitor Call exception is taken to Monitor mode.

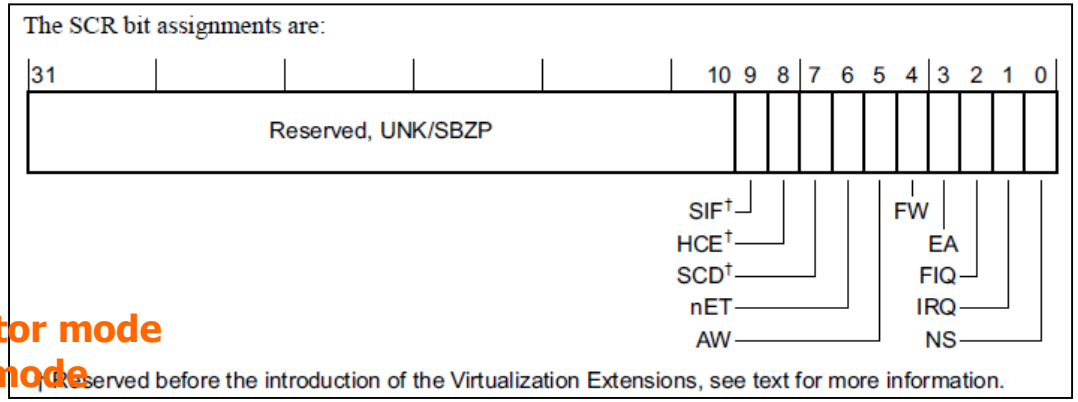
The preferred return address for a Secure Monitor Call exception is the address of the next instruction after the SMC instruction. This return is performed using the SPSR and LR_mon values generated by the exception entry, using an exception return instruction without a subtraction.

For more information, see *Exception return* on page B1-1194.

Secure Configuration Register (SCR)

- SCR defines the configuration of the current security state
  - Security state of the CPU: Secure or Non-secure
  - What mode CPU branches to if IRQ, FIQ or external abort occurs
  - Whether the CPSR.{F,A} can be modified when SCR.NS == 1
- **Only accessible from Secure PL1**

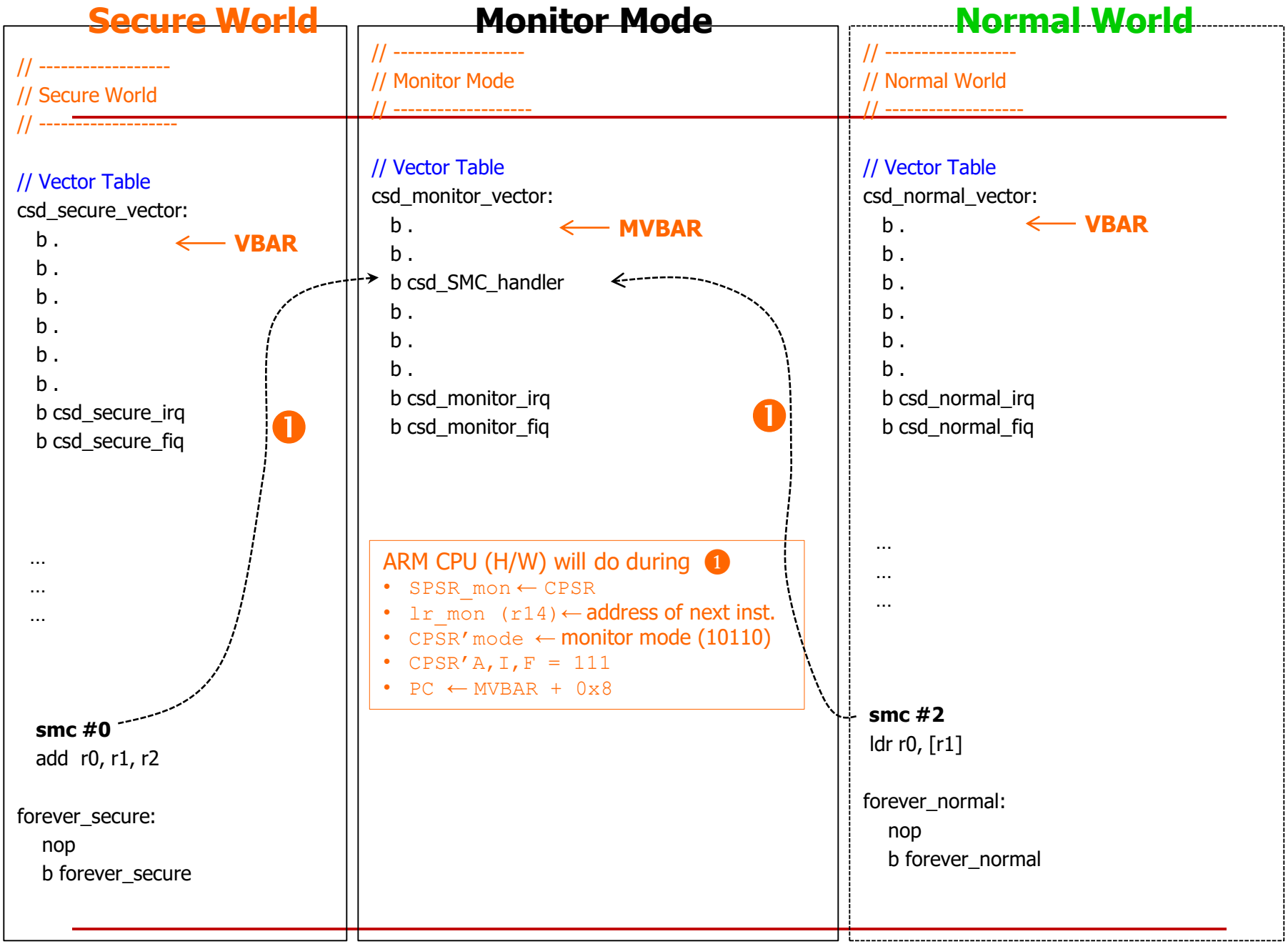
- **SIF: Secure Instruction Fetch**
- **HCE: Hyp Call Enable**
- **SCD: Secure Monitor Call disable**
- **nET: Not Early Termination**
- **AW: A (CPSR'A) bit writable**
- **FW: F (CPSR'F) bit writable**
- **EA: External Abort handler**
  - **0: External aborts not taken to Monitor mode**
  - **1: External aborts taken to Monitor mode**
- **FIQ: FIQ handler**
  - **0: FIQs not taken to Monitor mode**
  - **1: FIQs taken to Monitor mode**
- **IRQ: IRQ handler**
  - **0: IRQs not taken to Monitor mode**
  - **1: IRQs taken to Monitor mode**
- **NS: Non-Secure bit**



Accessing the SCR

To access the SCR, software reads or writes the CP15 r
and <opc2> set to 0. For example:

```
MRC p15, 0, <Rt>, c1, c1, 0     ; Read SCR into Rt
MCR p15, 0, <Rt>, c1, c1, 0     ; Write Rt to SCR
```


Secure World
// ------------------
// Secure World
// ------------------
// Vector Table
csd_secure_vector:
b .
b .
b .
b .
b .
b .
b csd_secure_irq
b csd_secure_fiq
VBAR
...
...
...
smc #0
add r0, r1, r2
forever_secure:
nop
b forever_secure
Monitor Mode
// ------------------
// Monitor Mode
// ------------------
// Vector Table
csd_monitor_vector:
b .
b .
b csd_SMC_handler
b .
b .
b .
b csd_monitor_irq
b csd_monitor_fiq
MVBAR
1
ARM CPU (H/W) will do during 1
• SPSR_mon ← CPSR
• lr_mon (r14) ← address of next inst.
• CPSR'mode ← monitor mode (10110)
• CPSR'A,I,F = 111
• PC ← MVBAR + 0x8
Normal World
// ------------------
// Normal World
// ------------------
// Vector Table
csd_normal_vector:
b .
b .
b .
b .
b .
b .
b csd_normal_irq
b csd_normal_fiq
VBAR
...
...
...
smc #2
ldr r0, [r1]
forever_normal:
nop
b forever_normal

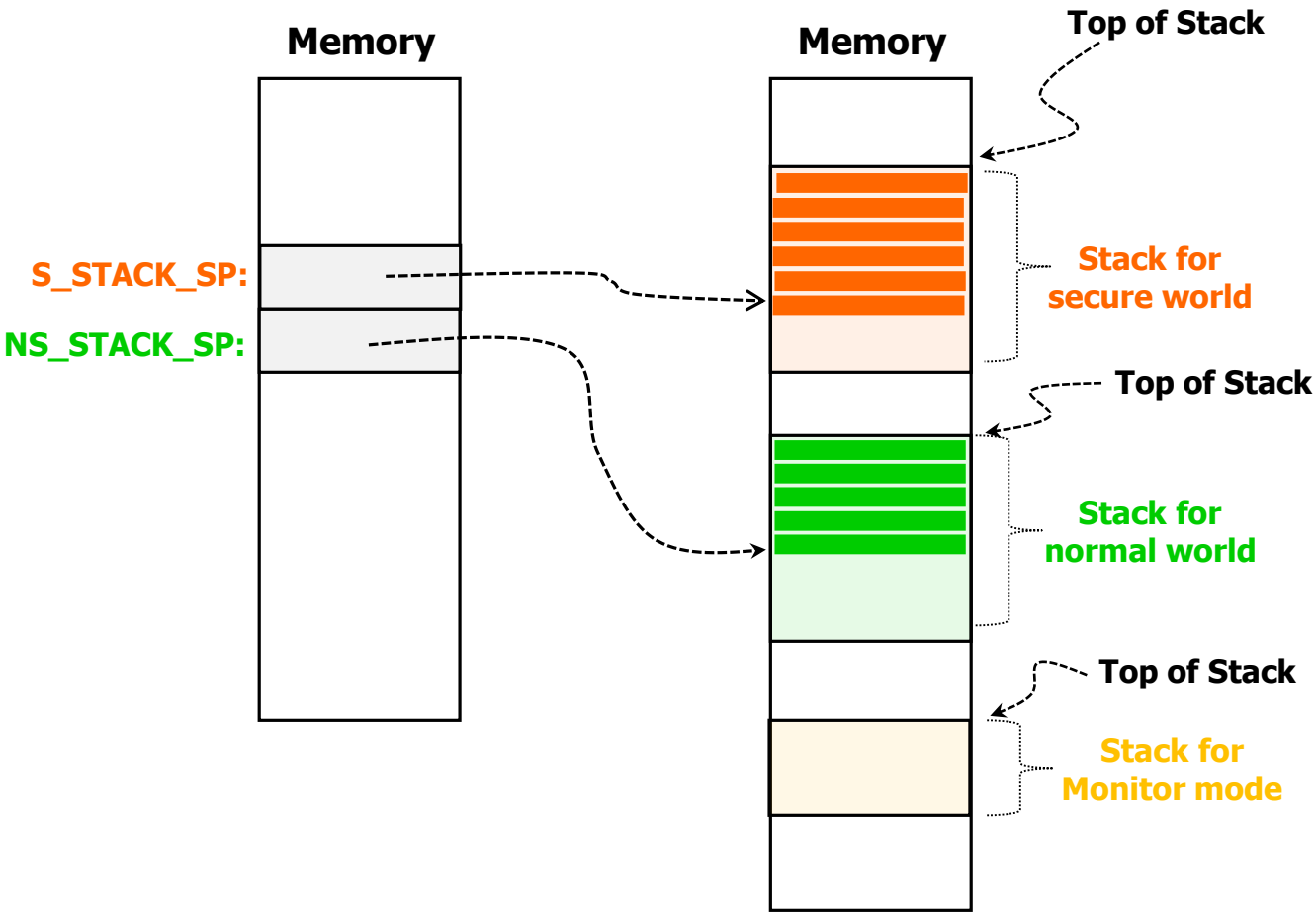
Korea Univ

Example Code (SMC Handler in Monitor Mode)

```
// ------------------------------------------------------------
// SMC Handler
//
// 1. Detect which world executed SMC
// 2. Switch world
// 3. Load saved stack pointers from 2 worlds
// 4. Push world's state & Pop the other worlds state
// 5. Exception return
// ------------------------------------------------------------
SMC_Handler:

 PUSH  {r0-r3} // r0-r3 contain args to be passed between worlds
                  // here use it as scratch regs

// Which world have we come from
  MRC     p15, 0, r0, c1, c1, 0  // Read SCR          (1)
  TST     r0, #0x1               // Test (and) NS bit → N,Z,C updated
  EOR     r0, r0, #0x1           // Toggle NS bit
  MCR     p15, 0, r0, c1, c1, 0  // Write to  SCR     (2)

// Load the saved SP for push
  LDREQ   r0, =S_STACK_SP        (3)
  LDRNE   r0, =NS_STACK_SP
  LDR     r2, [r0]               // r2 has SP
```

```
// Load the saved SP for pop
 LDREQ   r1, =NS_STACK_SP       (3)
 LDRNE   r1, =S_STACK_SP
 LDR     r3, [r1]         // r3 has SP

 // Push to r2
 // Save general purpose registers, SPSR and LR
 STMFD   r2!, {r4-r12}      // Save r4 to r12      (4)
 MRS     r4, spsr
 STMFD   r2!, {r4, lr}      // Save original SPSR and LR
 STR     r2, [r0]           // Save updated SP (r0 and r2 now free)

 // Pop from r3
 // Restore the other world's registers, SPSR and LR
 LDMFD   r3!, {r0, lr}      // Get SPSR and LR from (4)
 MSR     spsr_all, r0       // Restore SPSR
 LDMFD   r3!, {r4-r12}      // Restore registers r4 to r12
 STR     r3, [r1]           // Save updated SP (r1 and r3 now free)

 // Now restore args (r0-r3)
 POP     {r0-r3}

 // Perform exception return
 MOVS    pc, lr             (5)
```

Memory Layout for Example Code



Example:
World Switching via Interrupts

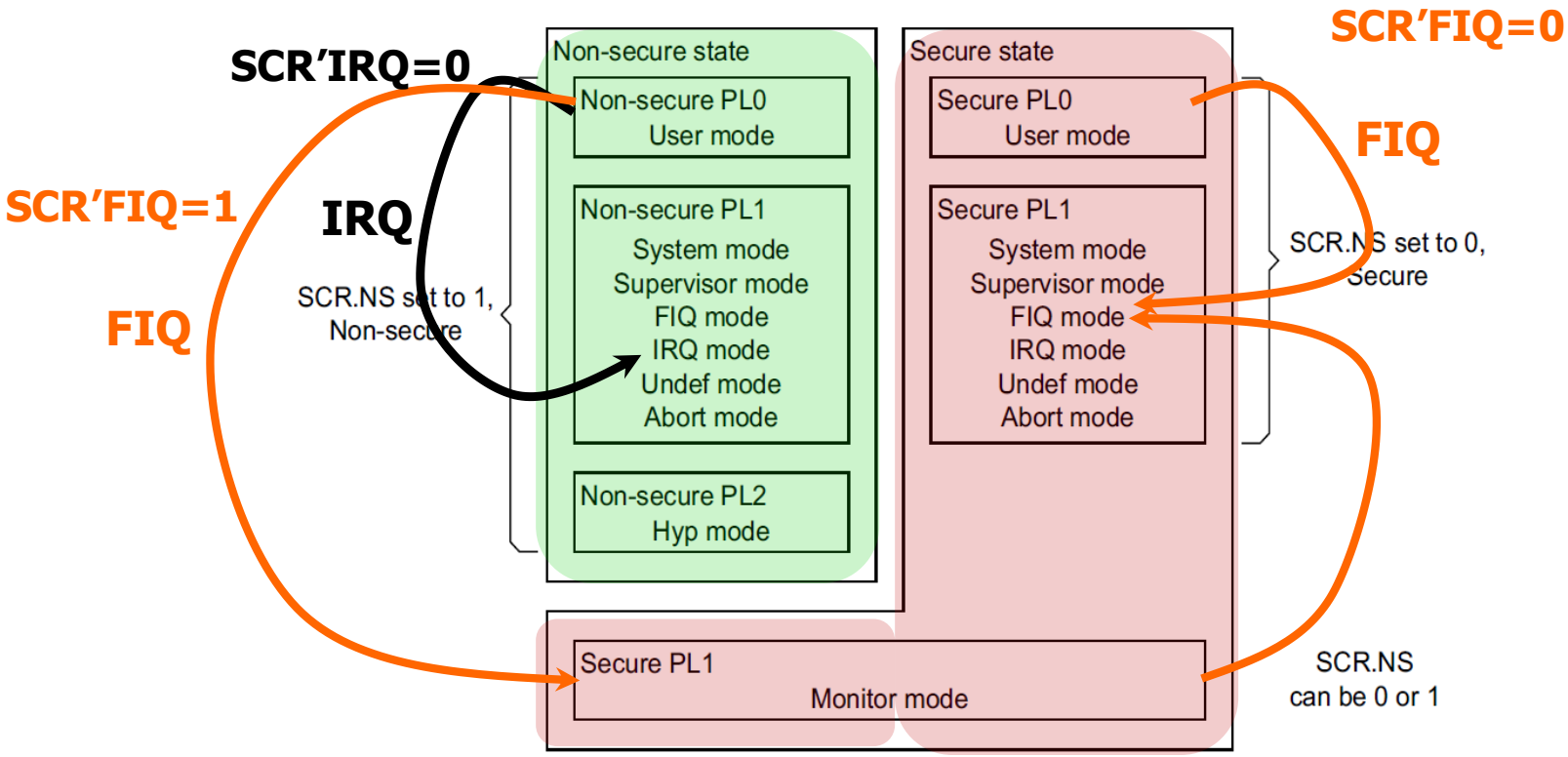


Figure B1-1 Modes, privilege levels, and security states

Source: ARM Security Technology – Building a Secure System using TrustZone Technology: p44

Example:
World Switching via Interrupts

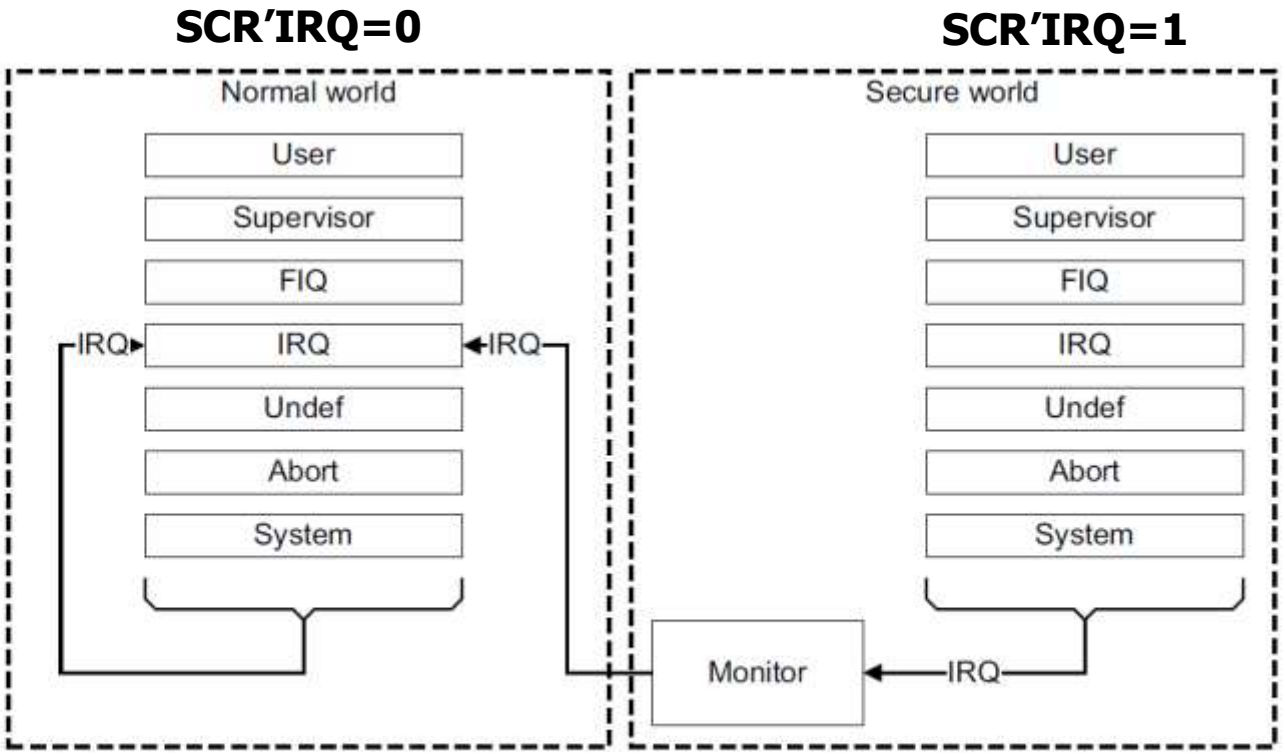
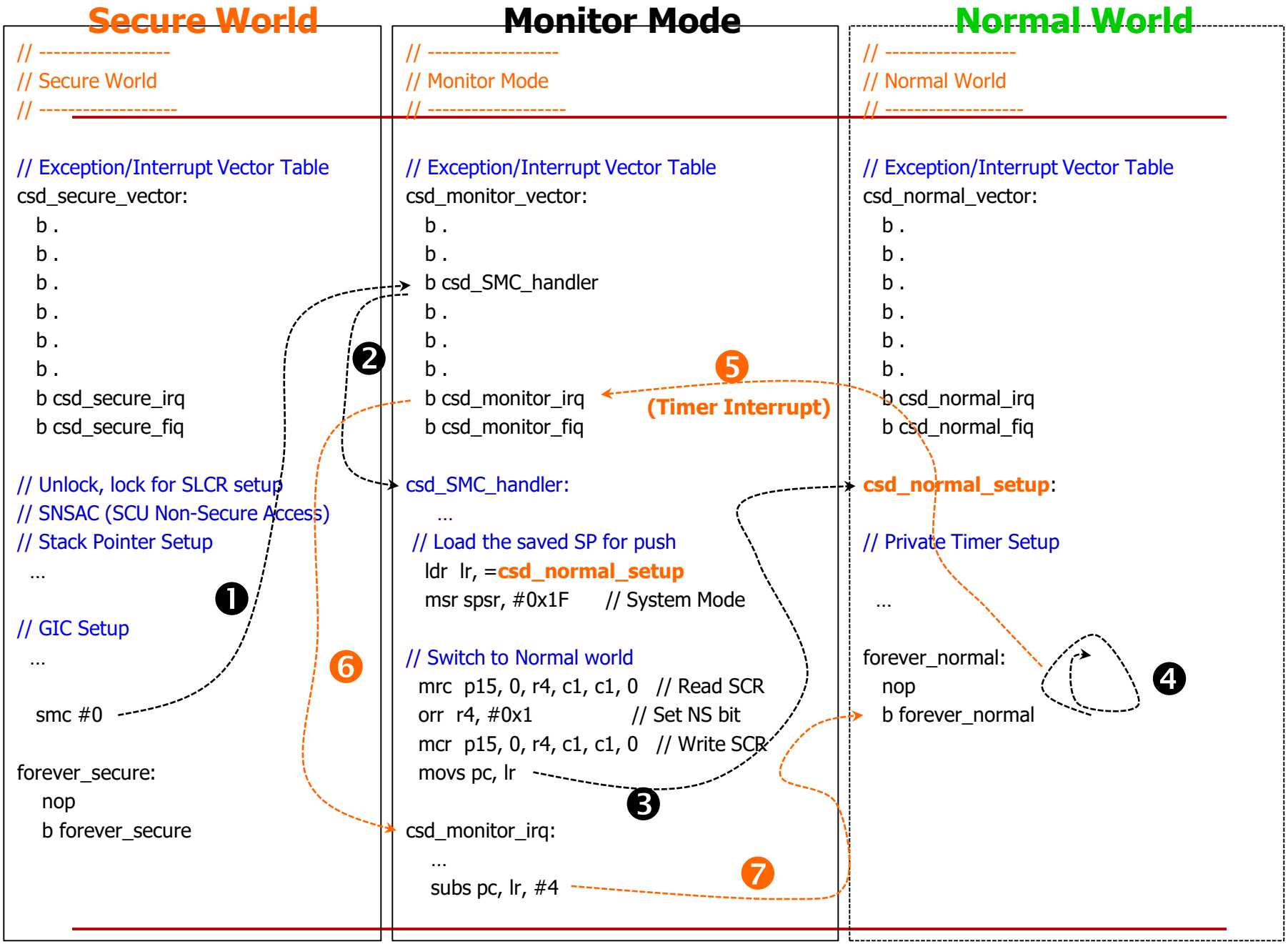


Figure 3-3 : One possible IRQ routing in a design with IRQ configured as a non-secure interrupt

Source: ARM Security Technology – Building a Secure System using TrustZone Technology: p44


Secure World
// ----------
// Secure World
// ----------
// Exception/Interrupt Vector Table
csd_secure_vector:
b .
b .
b .
b .
b .
b .
b csd_secure_irq
b csd_secure_fiq
// Unlock, lock for SLCR setup
// SNSAC (SCU Non-Secure Access)
// Stack Pointer Setup
...
// GIC Setup
...
smc #0
forever_secure:
nop
b forever_secure
Monitor Mode
// ----------
// Monitor Mode
// ----------
// Exception/Interrupt Vector Table
csd_monitor_vector:
b .
b .
b csd_SMC_handler
b .
b .
b .
b csd_monitor_irq
b csd_monitor_fiq
csd_SMC_handler:
...
// Load the saved SP for push
ldr lr, =csd_normal_setup
msr spsr, #0x1F // System Mode
// Switch to Normal world
mrc p15, 0, r4, c1, c1, 0 // Read SCR
orr r4, #0x1 // Set NS bit
mcr p15, 0, r4, c1, c1, 0 // Write SCR
movs pc, lr
csd_monitor_irq:
...
subs pc, lr, #4
Normal World
// ----------
// Normal World
// ----------
// Exception/Interrupt Vector Table
csd_normal_vector:
b .
b .
b .
b .
b .
b .
b csd_normal_irq
b csd_normal_fiq
csd_normal_setup:
// Private Timer Setup
...
forever_normal:
nop
b forever_normal
1
2
3
4
5
(Timer Interrupt)
6
7

Backup Slides

Taking SMC…

Pseudocode description of taking the Secure Monitor Call exception

The TakeSMCException() pseudocode procedure describes how the processor takes the exception:

```
// TakeSMCException()
// ==================

TakeSMCException()
    // Determine return information. SPSR is to be the current CPSR, after changing the IT[]
    // bits to give them the correct values for the following instruction, and LR is to be
    // the current PC minus 0 for Thumb or 4 for ARM, to change the PC offsets of 4 or 8
    // respectively from the address of the current instruction into the required address of
    // the next instruction (with the SMC instruction always being 4 bytes in length).
    ITAdvance();
    new_lr_value = if CPSR.T == '1' then PC else PC-4;
    new_spsr_value = CPSR;
    vect_offset = 8;

    // Ensure Secure state if initially in Monitor mode.
    // This affects the Banked versions of various registers accessed later in the code.
    if CPSR.M == '10110' then SCR.NS = '0';

    EnterMonitorMode(new_spsr_value, new_lr_value, vect_offset);
```

B1.9.14 Additional pseudocode functio

The EnterMonitorMode() pseudocode function changes the processor mode to Monitor mode, with the required state changes:

```
// EnterMonitorMode()
// ==================

EnterMonitorMode(bits(32) new_spsr_value, bits(32) new_lr_value, integer vect_offset)
    CPSR.M  = '10110';
    SPSR[]  = new_spsr_value;
    R[14]   = new_lr_value;
    CPSR.J  = '0';
    CPSR.T  = SCTLR.TE;
    CPSR.E  = SCTLR.EE;
    CPSR.A  = '1';
    CPSR.F  = '1';
    CPSR.I  = '1';
    CPSR.IT = '00000000';
    BranchTo(MVBAR + vect_offset);
```

The EnterHypMode() pseudocode function changes the processor mode to Hyp mode, with the required state changes:

Banked System Registers

Banked system control registers

In an implementation that includes the Security Extensions, some system control registers are Banked. Banked system control registers have two copies, one Secure and one Non-secure. The SCR.NS bit selects the Secure or Non-secure copy of the register. Table B3-33 shows which CP15 registers are Banked in this way, and the permitted access to each register. No CP14 registers are Banked.

Table B3-33 Banked CP15 registers

CRn[a]	Banked register	Permitted accesses[b]
c0	CSSELR, Cache Size Selection Register	Read/write only at PL1 or higher
c1	SCTLR, System Control Register[c]	Read/write only at PL1 or higher
	ACTLR, Auxiliary Control Register[d]	Read/write only at PL1 or higher
c2	TTBR0, Translation Table Base 0	Read/write only at PL1 or higher
	TTBR1, Translation Table Base 1	Read/write only at PL1 or higher
	TTBCR, Translation Table Base Control	Read/write only at PL1 or higher
c3	DACR, Domain Access Control Register	Read/write only at PL1 or higher
c5	DFSR, Data Fault Status Register	Read/write only at PL1 or higher
	IFSR, Instruction Fault Status Register	Read/write only at PL1 or higher
	ADFSR, Auxiliary Data Fault Status Register[d]	Read/write only at PL1 or higher
	AIFSR, Auxiliary Instruction Fault Status Register[d]	Read/write only at PL1 or higher

http://esca.korea.ac.kr/teaching/cose321_CSD/labs/DDI0406C_C_armv7AR_architecture_reference_manual.pdf

Banked System Registers

c6	DFAR, Data Fault Address Register	Read/write only at PL1 or higher
	IFAR, Instruction Fault Address Register	Read/write only at PL1 or higher
c7	PAR, Physical Address Register	Read/write only at PL1 or higher
c10	PRRR, Primary Region Remap Register	Read/write only at PL1 or higher
	NMRR, Normal Memory Remap Register	Read/write only at PL1 or higher
c12	VBAR, Vector Base Address Register	Read/write only at PL1 or higher
c13	FCSEIDR, FCSE PID Register[e]	Read/write only at PL1 or higher
	CONTEXTIDR, Context ID Register	Read/write only at PL1 or higher
	TPIDRURW, User Read/Write Thread ID	Read/write at all privilege levels, including PL0
	TPIDRURO, User Read-only Thread ID	Read-only at PL0 Read/write at PL1 or higher
	TPIDRPRW, PL1 only Thread ID	Read/write only at PL1 or higher

a. For accesses to 32-bit registers. More correctly, this is the primary coprocessor register.

b. Any attempt to execute an access that is not permitted results in an Undefined Instruction exception.

c. Some bits are common to the Secure and the Non-secure copies of the register, see *SCTLR, System Control Register, VMSA* on page B4-1707.

d. See *ADFSR and AIFSR, Auxiliary Data and Instruction Fault Status Registers, VMSA* on page B4-1523. Register is IMPLEMENTATION DEFINED.

e. Banked only in an implementation that includes the FCSE. The FCSE PID Register is RAZ/WI if the FCSE is not implemented.

A Banked CP15 register can contain a mixture of:

- fields that are Banked
- fields that are read-only in Non-secure PL1 or PL2 modes but read/write in the Secure state.

The System Control Register SCTLR is an example of a register of that contains this mixture of fields.

The Secure copies of the Banked CP15 registers are sometimes referred to as the Secure Banked CP15 registers. The Non-secure copies of the Banked CP15 registers are sometimes referred to as the Non-secure Banked CP15 registers.

http://esca.korea.ac.kr/teaching/cose321_CSD/labs/DDI0406C_C_armv7AR_architecture_reference_manual.pdf

External Aborts

B3.12 VMSA memory aborts

In a VMSAv7 implementation, the following mechanisms cause a processor to take an exception on a failed memory access:

Debug exception	An exception caused by the debug configuration, see *About debug exceptions* on page C4-2090.
Alignment fault	An Alignment fault is generated if the address used for a memory access does not have the required alignment for the operation. For more information see *Unaligned data access* on page A3-108 and *Alignment faults* on page B3-1402.
MMU fault	An MMU fault is a fault generated by the fault checking sequence for the current translation regime.
External abort	Any memory system fault other than a Debug exception, an Alignment fault, or an MMU fault.

Collectively, these mechanisms are called *aborts*. Chapter C4 *Debug Exceptions* describes Debug exceptions, and the remainder of this section describes Alignment faults, MMU faults, and External aborts.

The exception generated on a synchronous memory abort:

- on an instruction fetch is called the Prefetch Abort exception
- on a data access is called the Data Abort exception.

External Aborts

B3.12.6 External aborts

The ARM architecture defines external aborts as errors that occur in the memory system, other than those that are detected by the MMU or Debug hardware. External aborts include parity errors detected by the caches or other parts of the memory system. An external abort is one of:

- synchronous
- precise asynchronous
- imprecise asynchronous.

Normally, external aborts are rare. An imprecise asynchronous external abort is likely to be fatal to the process that is running. An example of an event that might cause an external abort is an uncorrectable parity or ECC failure on a Level 2 Memory structure.

It is IMPLEMENTATION DEFINED which external aborts, if any, are supported.

VMSAv7 permits external aborts on data accesses, translation table walks, and instruction fetches to be either synchronous or asynchronous. The reported fault code identifies whether the external abort is synchronous or asynchronous.

External Aborts

External aborts

External aborts are defined as errors that occur in the memory system other than those that are detected by the MMU or Debug hardware. They include parity errors detected by the caches or other parts of the memory system. An external abort is one of:

- synchronous
- precise asynchronous
- imprecise asynchronous.

For more information, see *Terminology for describing exceptions*.

The ARM architecture does not provide a method to distinguish between precise asynchronous and imprecise asynchronous aborts.

The ARM architecture handles asynchronous aborts in a similar way to interrupts, except that they are reported to the processor using the Data Abort exception. Setting the CPSR.A bit to 1 masks asynchronous aborts, see *Program Status Registers (PSRs)*.

Normally, external aborts are rare. An imprecise asynchronous external abort is likely to be fatal to the process that is running. An example of an event that might cause an external abort is an uncorrectable parity or ECC failure on a Level 2 Memory structure.

It is implementation defined which external aborts, if any, are supported.

VMSAv7 permits external aborts on data accesses, translation table walks, and instruction fetches to be either synchronous or asynchronous. The DFSR indicates whether the external abort is synchronous or asynchronous, see *c5, Data Fault Status Register (DFSR)*.

Note

Because imprecise external aborts are normally fatal to the process that caused them, ARM recommends that implementations make external aborts precise wherever possible.

More information about possible external aborts is given in the subsections:

- *External abort on instruction fetch*
- *External abort on data read or write*
- *External abort on a translation table walk*
- *Behavior of external aborts on a translation table walk caused by a VA to PA translation*
- *Parity error reporting*.

https://developer.arm.com/documentation/ddi0406/b/System-Level-Architecture/Virtual-Memory-System-Architecture--VMSA-/VMSA-memory-aborts/External-aborts

Security Extensions Support

- When a GIC that implements the GIC Security Extensions is connected to a processor that implement the ARM Security Extensions:
  - Group0 interrupt are Secure interrupts, and Group 1 interrupts are Non-secure interrupts
  - Secure system software individually defines each implemented interrupt as either Secure or Non-secure
  - A Secure interrupt can signal either an IRQ or an FIQ interrupt request to a target processor
  - A Non-secure interrupt signals an IRQ interrupt request to a target processor

Source: ARM GIC Architecture Specification, page# 16

Cortex A9-MP Interrupt Controller

- Permit all implemented interrupts to be individually defined as secure or non-secure
- Non-secure interrupts are always signaled using IRQ
- Secure interrupts can be programmed to use either FIQ or IRQ through the FIQen bit in the ICCICR (CPU Interface Control register)

[3]	FIQEn	Controls whether the GIC signals Secure interrupts to a target processor using the FIQ or the IRQ signal. **0** Signal Secure interrupts using the IRQ signal. **1** Signal Secure interrupts using the FIQ signal. The GIC always signals Non-secure interrupts using the IRQ signal.

Source: Programming ARM TrustZone Architecture on Zynq-7000, page# 35

Recommendations

- Use **IRQ** as **Normal world interrupt source**, and **FIQ** as **Secure world interrupt source**
  - IRQ is the most common interrupt source in use in most OSs
  - It incurs fewest modifications to existing software
- If the core is in the other world (i.e., normal world) when an interrupt occurs,
  - The hardware traps to the monitor, the monitor software causes a context switch, and jumps to the restored world (secure world), at which point the interrupt is taken
- To prevent malicious Normal world software masking sensitive Secure world interrupts, SCR'AW and SCR'FW are provided, which can only be accessed by Secure world

Source: ARM Security Technology – Building a Secure System using TrustZone Technology: p44

Example

- Group0
  - Secure interrupt
  - Signaled as FIQ
- Group1
  - Non-secure interrupts
  - Signaled as IRQ
- Require the processor to
  - Route FIQ to be taken in Secure Monitor mode
  - Prevent Non-secure software from masking FIQs
  - Ensure that IRQ is masked whenever operating in Secure state (CPSR'I?: can't get it though)

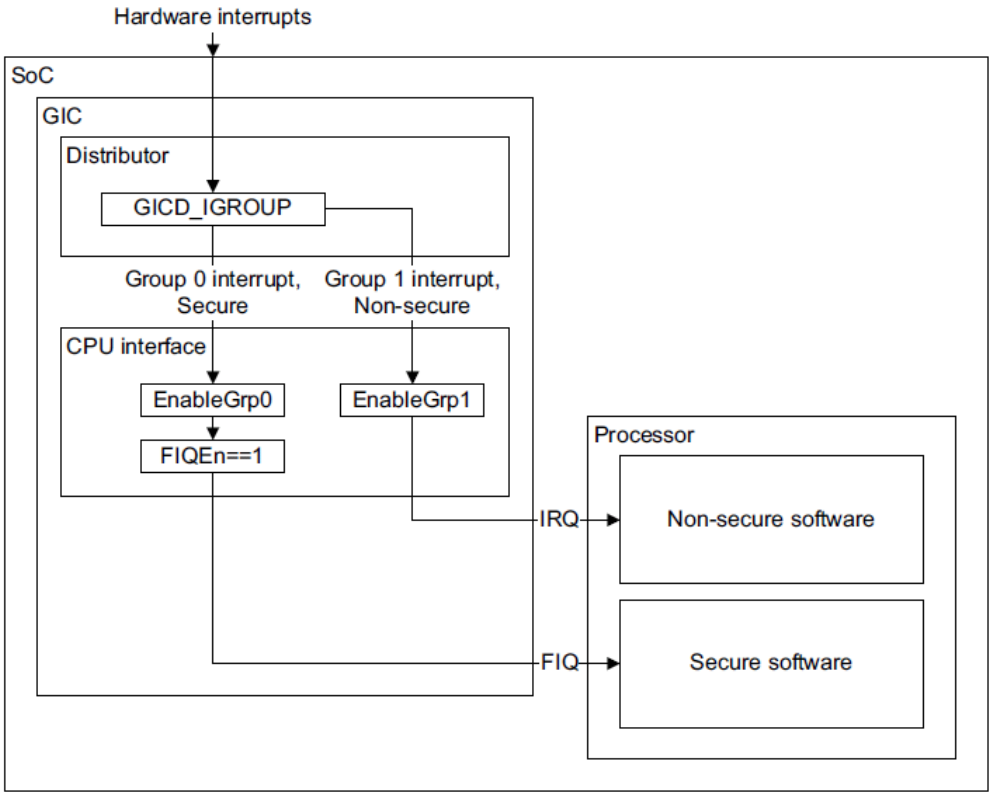


Figure 3-9 Using the GIC to route Secure and Non-secure interrupts

http://esca.korea.ac.kr/teaching/cose321_CSD/labs/IHI0048B_b_gic_architecture_specification.pdf

Notes

- When a processor that implements the ARM Security Extensions is connected to the GIC, the Secure software executing on the processor usually accesses the GIC using only Secure accesses.

Source: ARM GIC Architecture version 2.0 page 4-80

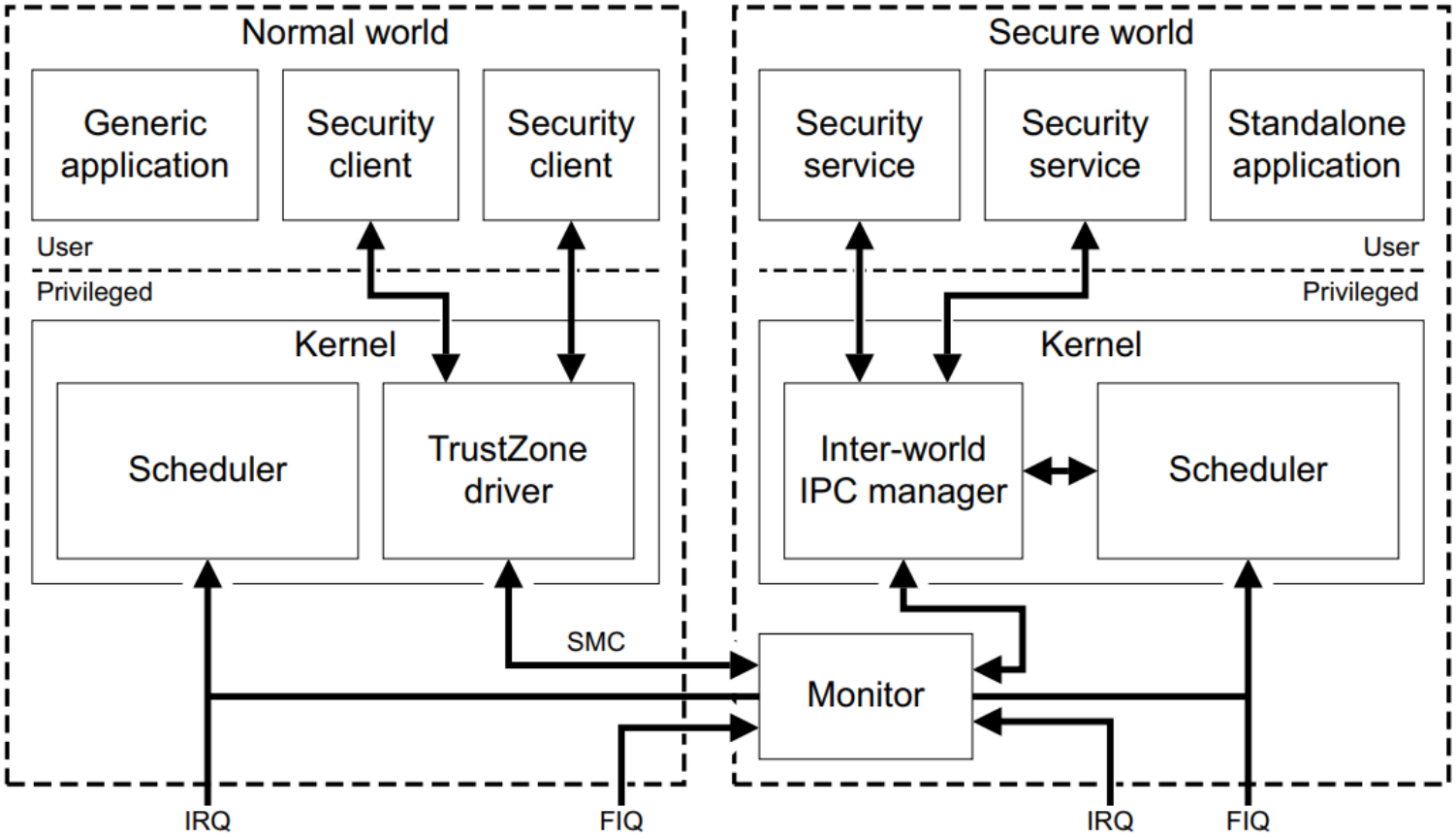


Figure 5-1 : A possible architecture with an independent Secure world OS

http://esca.korea.ac.kr/teaching/cose321_CSD/labs/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

Interrupt Taken to …

	Group0 (Secure) via FIQ				Group1 (Normal) via IRQ			
	SCR'F				SCR'I			
	SCR'F = 0 (to current world)		SCR'F = 1 (to monitor)		SCR'I = 0 (to current world)		SCR'I = 1 (to monitor)	
CPU's Current Status	in Normal world	In Secure World	In Normal World	In Secure World	In Normal World	In Secure World	In Normal World	In Secure World
Interrupt taken to…	Seems… Normal World (breakpoint doesn't work, but when suspended, CPU is in FIQ mode	Secure World	Monitor Mode	Monitor Mode	Normal World	Secure World	Monitor Mode	Monitor Mode

- Private timer interrupt to Group0 (GICD_IGROUP0, bit29):
- GICC_CTLR (FIQen, G1, G0 enables) = 1x11; Group0 uses FIQ. GICD_CTRL has G1 and G0 enables
- SCR (F, I, NS)= 3'b000 ; F = 0 meaning interrupt taken to current world

- Private timer interrupt to Group1 (GICD_IGROUP0, bit29):
- GICC_CTLR (FIQen, G1, G0 enables) = 1x11; Group0 uses FIQ
- SCR (F, I, NS)= 3'b000 ; F = 0 meaning interrupt taken to current world

TrustZone-related Registers in Zynq

Register Summary

Register Name	Address	Width	Type	Reset Value	Description
security2_sdio0	0xE0200008	1	WO	0x00000000	SDIO0 slave security setting.
security3_sdio1	0xE020000C	1	WO	0x00000000	SDIO1 slave security setting.
security4_qspi	0xE0200010	1	WO	0x00000000	QSPI slave security setting.
security6_apb_slaves	0xE0200018	15	WO	0x00000000	APB slave security setting.
security7_smc	0xE020001C	1	WO	0x00000000	SMC slave security setting.
DMAC_RST_CTRL	0xF800020C	32	RW	0x00000000	DMA Controller SW Reset Control
TZ_OCM_RAM0	0xF8000400	32	RW	0x00000000	OCM RAM TrustZone Config 0
TZ_OCM_RAM1	0xF8000404	32	RW	0x00000000	OCM RAM TrustZone Config 1
TZ_OCM	0xF8000408	32	RW	0x00000000	OCM ROM TrustZone Config
TZ_DDR_RAM	0xF8000430	32	RW	0x00000000	DDR RAM TrustZone Config

Register Name	Address	Width	Type	Reset Value	Description
TZ_DMA_NS	0xF8000440	32	RW	0x00000000	DMAC TrustZone Config
TZ_DMA_IRQ_NS	0xF8000444	32	RW	0x00000000	DMAC TrustZone Config for Interrupts
TZ_DMA_PERIPH_NS	0xF8000448	32	RW	0x00000000	DMAC TrustZone Config for Peripherals
TZ_GEM	0xF8000450	32	RW	0x00000000	Ethernet TrustZone Config
TZ_SDIO	0xF8000454	32	RW	0x00000000	SDIO TrustZone Config
TZ_USB	0xF8000458	32	RW	0x00000000	USB TrustZone Config
TZ_FPGA_M	0xF8000484	32	RW	0x00000000	FPGA master ports TrustZone Disable
TZ_FPGA_AFI	0xF8000488	32	RW	0x00000000	FPGA AFI AXI ports TrustZone Disable
security_fssw_s0	0xF890001C	1	WO	0x00000000	M_AXI_GP0 security setting
security_fssw_s1	0xF8900020	1	WO	0x00000000	M_AXI_GP1 security setting
security_apb	0xF8900028	6	WO	0x00000000	APB boot secure ports setting.

Source: Programming ARM TrustZone Architecture on Zynq-7000 page# 15

TZ_DDR_RAM

TZ_DDR_RAM

Name	TZ_DDR_RAM
Relative Address	0xF8000430
Absolute Address	0xF8000430
Width	32 bits
Access Type	RW
Reset Value	0x00000000
Description	DDR RAM TrustZone Config

Field Name	Bit	Type	Reset Value	Description
TZ_DDR_RAM	0	RW	0x0	Each bit represents the TrustZone status for a 64 MB section n at nMB: • 0: Secure, reset value • 1: Non-secure

SLCR_LOCK

Register (slcr) SLCR_LOCK

Name	SLCR_LOCK
Relative Address	0x00000004
Absolute Address	0xF8000004
Width	32 bits
Access Type	wo
Reset Value	0x00000000
Description	SLCR Write Protection Lock

Register SLCR_LOCK Details

Field Name	Bits	Type	Reset Value	Description
reserved	31:16	wo	0x0	Reserved. Writes are ignored, read data is zero.
LOCK_KEY	15:0	wo	0x0	Write the lock key, 0x767B, to write protect the slcr registers: all slcr registers, 0xF800_0000 to 0xF800_0B74, are write protected until the unlock key is written to the SLCR_UNLOCK register. A read of this register returns zero.

- **SLCR: System-level Control Register**
- **SLCRs are used to control the PS behavior**

Zynq-7000 TRM: Page 1493 ~ 1498

Source: Zynq-7000 TRM

SLCR_UNLOCK

SLCR: System-level Control Register

Register (slcr) SLCR_UNLOCK

Name	SLCR_UNLOCK
Relative Address	0x00000008
Absolute Address	0xF8000008

Width	32 bits
Access Type	wo
Reset Value	0x00000000
Description	SLCR Write Protection Unlock

Register SLCR_UNLOCK Details

Field Name	Bits	Type	Reset Value	Description
reserved	31:16	wo	0x0	Reserved. Writes are ignored, read data is zero.
UNLOCK_KEY	15:0	wo	0x0	Write the unlock key, 0xDF0D, to enable writes to the slcr registers. All slcr registers, 0xF800_0000 to 0xF800_0B74, are writeable until locked using the SLCR_LOCK register. A read of this register returns zero.

Source: Zynq-7000 TRM

Zynq-7000 TRM: Page 1493 ~ 1498

SLCR_LOCK Status

SLCR: System-level Control Register

Register (slcr) SLCR_LOCKSTA

Name	SLCR_LOCKSTA
Relative Address	0x0000000C
Absolute Address	0xF800000C
Width	32 bits
Access Type	ro
Reset Value	0x00000001
Description	SLCR Write Protection Status

Register SLCR_LOCKSTA Details

Field Name	Bits	Type	Reset Value	Description
reserved	31:1	ro	0x0	Reserved. Writes are ignored, read data is zero.
LOCK_STATUS	0	ro	0x1	Current state of write protection mode of SLCR: 0: Registers are writeable. Use the slcr.SLCR_LOCK register to lock the slcr registers. 1: Registers are not writeable. Any attempt to write to an slcr register is ignored, but reads will return valid register values. Use the slcr.SLCR_UNLOCK register to unlock the slcr registers.

Source: Zynq-7000 TRM

SCU Non-Secure Access Control Register

Address: 0xF8F0_0054

Check out Zynq-7000 TRM and Cortex-A9 MPCore TRM

Figure 2-8 shows the SNSAC register bit assignments.

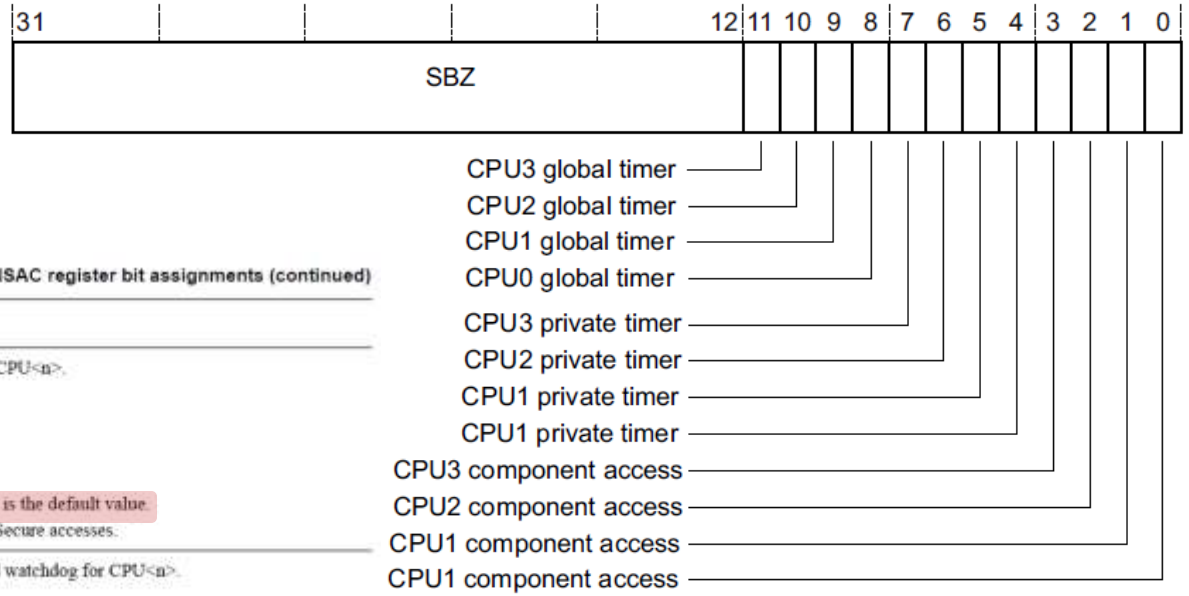


Figure 2-8 SNSAC register bit assignments

Table 2-9 SNSAC register bit assignments (continued)

Bits	Name	Function
[11]	CPU3 global timer	Non-secure access to the global timer for CPU<n>. • <n> is 3 for bit[11] • <n> is 2 for bit[10] • <n> is 1 for bit[9] • <n> is 0 for bit[8]. 0 Secure accesses only. This is the default value. 1 Secure accesses and Non-Secure accesses.
[10]	CPU2 global timer	
[9]	CPU1 global timer	
[8]	CPU0 global timer	
[7]	Private timers for CPU<n>	Non-secure access to the private timer and watchdog for CPU<n>. • <n> is 3 for bit[7] • <n> is 2 for bit[6]] • <n> is 1 for bit[5] • <n> is 0 for bit[4]. 0 Secure accesses only. Non-secure reads return 0. This is the default value. 1 Secure accesses and Non-secure accesses.
[6]		
[5]		
[4]		
[3]	Register access for CPU<n>	Non-secure access to the registers for CPU<n>. • <n> is 3 for bit[3] • <n> is 2 for bit[2]] • <n> is 1 for bit[1] • <n> is 0 for bit[0]. 0 CPU cannot write the registers[a]. 1 CPU can access the registers[a].
[2]		
[1]		
[0]		

a. The accessible registers are the SAC Register, the SCU Control Register, the SCU CPU Status Register, the filtering registers, and the SCU CPU Power Status Register.

Zynq-7000 TRM: Page# 1376

Confusion in SPECs

Source: Zynq-7000 TRM -- WO

Register Summary

Register Name	Address	Width	Type	Reset Value	Description
security_gp0_axi	0x0000001C	1	wo	0x00000000	M_AXI_GP0 security setting
security_gp1_axi	0x00000020	1	wo	0x00000000	M_AXI_GP1 security setting

Register (nic301_addr_region_ctrl_registers) security_gp0_axi

Name	security_gp0_axi
Relative Address	0x0000001C
Absolute Address	0xF890001C
Width	1 bits
Access Type	wo
Reset Value	0x00000000
Description	M_AXI_GP0 security setting

Register security_gp0_axi Details

Field Name	Bits	Type	Reset Value	Description
gp0_axi	0	wo	0x0	Controls the transactions from M_AXI_GP0 to PL: 0 - Always secure 1 - Always non-secure.

Source: Programming ARM TrustZone Architecture on the Xilinx Zynq-7000 -- RO

security_fssw_s0

Name	security_fssw_s0
Relative Address	0xF890001C
Absolute Address	0xF890001C
Width	1 bit
Access Type	RO
Reset Value	0x00000000
Description	M_AXI_GP0 security setting

Field Name	Bit	Type	Reset Value	Description
fssw_s0	0	RO	0x0	This register defines whether S or NS request propagates out to the logic: • 0: NS requests do not propagate to the logic • 1: Both NS and S requests are propagated to the logic

TTBCR

B4.1.153 TTBCR, Translation Table Base Control Register, VMSA

The TTBCR characteristics are:

Purpose

TTBCR determines which of the Translation Table Base Registers, TTBR0 or TTBR1, defines the base address for a translation table walk required for the stage 1 translation of a memory access from any mode other than Hyp mode.

If the implementation includes the Large Physical Address Extension, the TTBCR also:

- Controls the translation table format.
- When using the Long-descriptor translation table format, holds cacheability and shareability information for the accesses.

Note

When using the Short-descriptor translation table format, TTBR0 and TTBR1 hold this cacheability and shareability information.

This register is part of the Virtual memory control registers functional group.

Usage constraints

Only accessible from PL1 or higher.

Configurations

The Large Physical Address Extension adds an alternative format for the register. If an implementation includes the Large Physical Address Extension then the current translation table format determines which format of the register is used.

If the implementation includes the Security Extensions, this register:

- is Banked
- has write access to the Secure copy of the register disabled when the **CP15SDISABLE** signal is asserted HIGH.

Attributes

A 32-bit RW register that resets to zero. If the implementation includes the Security Extensions this defined reset value applies only to the Secure copy of the register, except for the EAE bit in an implementation that includes the Large Physical Address Extension. For more information see the field descriptions. See also *Reset behavior of CP14 and CP15 registers* on page B3-1450.

Table B3-45 on page B3-1493 shows the encodings of all of the registers in the Virtual memory control registers functional group.

TTBR0

B4.1.154 TTBR0, Translation Table Base Register 0, VMSA

The TTBR0 characteristics are:

Purpose

TTBR0 holds the base address of translation table 0, and information about the memory it occupies. This is one of the translation tables for the stage 1 translation of memory accesses from modes other than Hyp mode.

This register is part of the Virtual memory control registers functional group.

Usage constraints

Only accessible from PL1 or higher.

Used in conjunction with the TTBCR. When the 64-bit TTBR0 format is used, cacheability and shareability information is held in the TTBCR, not in TTBR0.

Configurations

The Multiprocessing Extensions change the TTBR0 32-bit register format.

The Large Physical Address Extension extends TTBR0 to a 64-bit register. In an implementation that includes the Large Physical Address Extension, TTBCR.EAE determines which TTBR0 format is used:

EAE==0 32-bit format is used. TTBR0[63:32] are ignored.

EAE==1 64-bit format is used.

If the implementation includes the Security Extensions, this register:

- is Banked
- has write access to the Secure copy of the register disabled when the **CP15SDISABLE** signal is asserted HIGH.

Attributes

A 32-bit or 64-bit RW register with a reset value that depends on the register implementation. For more information see the register bit descriptions. See also *Reset behavior of CP14 and CP15 registers* on page B3-1450.

Table B3-45 on page B3-1493 shows the encodings of all of the registers in the Virtual memory control registers functional group.