SMC CALLING CONVENTION
System Software on ARM® Platforms

Document number: ARM DEN 0028A

SMC Calling Convention
System Software on ARM Platforms

Release information

The Release History table lists the releases of this document.

Table 1-1 Release history

Date	Issue	Confidentiality	Change
June 2013	A	Non-Confidential	First release

1 ABOUT THIS DOCUMENT

1.1 Introduction

This document defines a common calling mechanism for use with the Secure Monitor Call (SMC) instruction in both the ARMv7 and ARMv8 architectures.

The SMC instruction is used to generate a synchronous exception that is handled by Secure Monitor code running in EL3. The arguments are passed in registers and then used to select which Secure function to execute. These calls may then be passed on to a Trusted OS in S-EL1.

This specification aims to ease integration and reduce fragmentation between software layers, such as Operating Systems, Hypervisors, Trusted OS, Secure Monitor and System Firmware.

Note: This document is defined with respect to the ARMv8 Exception levels, EL0 to EL3.
The relationship between these and the 32-bit ARMv7 Exception levels is described in [2].

1.2 References

This document refers to the following documents.

Ref	Doc No	Author(s)	Title
[1]	ARM DDI 0406	ARM	ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition
[2]	ARM DDI 0487	ARM	ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile ***Note:*** *Document under development, release expected Q3-2013*
[3]	ARM IHI 0042	ARM	Procedure Call Standard for the ARM 32-bit Architecture
[4]	ARM IHI 0055	ARM	Procedure Call Standard for the ARM 64-bit Architecture
[5]	ARM DEN 022	ARM	Power State Coordination Interface
[6]	http://tools.ietf.org/html/rfc4122	IETF	RFC 4122 - A Universally Unique IDentifier (UUID) URN Namespace

1.3 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
AArch32 state	The ARM 32-bit execution state that uses 32-bit general purpose registers, and a 32-bit program counter (PC), stack pointer (SP), and link register (LR). AArch32 execution state provides a choice of two instruction sets, A32 and T32, previously called the ARM and Thumb instruction sets.
AArch64 state	The ARM 64-bit execution state that uses 64-bit general purpose registers, and a 64-bit program counter (PC), stack pointer (SP), and exception link registers (ELR). AArch64 execution state provides a single instruction set, A64.
EL0	The lowest exception level. The exception level used to execute user applications, in Non-secure state.
EL1	Privileged exception level. The exception level used to execute operating systems, in Non-secure state.
EL2	Hypervisor exception level. The exception level used to execute hypervisor code. EL2 is always in Non-secure state.
EL3	Secure Monitor exception level. The exception level used to execute Secure Monitor code, which handles the transitions between Non-secure and Secure states. EL3 is always in Secure state.
Non-secure state	The ARM execution state that restricts access to only the Non-secure system resources such as: memory, peripherals and system registers.
OEM	Original Equipment Manufacturer. In this document the final device manufacturer.
S-EL0	The Secure EL0 Exception level, the Exception level used to execute trusted application code in Secure state.
S-EL1	The Secure EL1 Exception level, the Exception level used to execute Trusted OS code in Secure state.
Secure Monitor	The Secure Monitor is software that executes at the EL3 Exception level. It receives and handles Secure Monitor exceptions, and provides transitions between Secure state and Non-secure state.
Secure state	The ARM execution state that enables access to the Secure and Non-secure systems resources, such as: memory, peripherals and system registers.
SiP	Silicon Partner. In this document, the silicon manufacturer.
SMC	Secure Monitor Call. An ARM assembler instruction that causes an exception that is taken synchronously into EL3.
SMC32	32-bit SMC calling convention
SMC64	64-bit SMC calling convention
SMC Function Identifier	A 32-bit integer which identifies which function is being invoked by this SMC call. Passed in R0 or W0 into every SMC call.
Trusted OS	The secure operating system running in the Secure EL1 Exception level. It supports the execution of trusted applications in Secure EL0.

2 SMC CALLING CONVENTIONS

In the ARM architecture, synchronous control is transferred between the normal Non-secure state to Secure state through System Monitor Call exceptions [1][2]. SMC exceptions are generated by the SMC instruction [1][2] and handled by the Secure Monitor. The operation of the Secure Monitor is determined by the parameters passed in through registers.

Two types of calls are defined:

- Fast Calls used to execute atomic Secure operations.
- Standard Calls used to start pre-emptible Secure operations.

The additional asynchronous infrastructure required for pre-emptible Standard Calls is outside of the scope of this specification.

Two calling conventions for the SMC instruction are defined:

- **SMC32**: A wholly 32-bit interface which can be used by either 32-bit or 64-bit client code and which passes up to six 32-bit arguments
- **SMC64**: A 64-bit interface which can be used only by 64-bit client code and which passes up to six 64-bit arguments

The SMC Function Identifier is defined.

It is passed into every SMC call in register R0 or W0 and it determines:

- The call type in use.
- The calling convention is in use.
- The secure function to be invoked.

2.1 SMC Function Identifiers

An SMC Function Identifier is a 32-bit integer value which indicates which function is being requested by the caller. It is always passed as the first argument to every SMC call in R0 or W0.

Specified bits within the 32-bit value have defined meanings as shown in table **Table 2-1**.

Table 2-1 Bit usage within the SMC Function Identifier

Bit Numbers	Bit Mask	Description
31	0x80000000	If set to 0 then this is Standard call (pre-emptible) If set to 1 then this is a Fast Call (atomic)
30	0x40000000	If set to 0 then this is the SMC32 calling convention. If set to 1 then this is the SMC64 calling convention.
29:24	0x3F000000	These ranges are further defined in section 6.
23:16	0x00FF0000	Must be zero (MBZ), for all Fast Calls, when bit[31] == 1. All other values reserved for future use **Note:** Some ARMv7 legacy Trusted OS Fast Call implementations have all bits set to 1.
15:0	0x0000FFFF	Function number within the range call type defined by bits[29:24].

Owning Entity Number	Bit Mask	Description
0	0x00000000	ARM Architecture Calls
1	0x01000000	CPU Service Calls
2	0x02000000	SIP Service Calls
3	0x03000000	OEM Service Calls
4	0x04000000	Standard Service Calls
5-47	0x05000000 – 0x2F000000	Reserved for future use
48-49	0x30000000 – 0x31000000	Trusted Application Calls
50-63	0x32000000 – 0x3F000000	Trusted OS Calls

2.2 SMC32 Argument passing

When the SMC32 convention is used, the SMC instructions take up to seven 32-bit arguments in registers and can return up to four 32-bit values in registers.

When an SMC32 call is made from AArch32:

- Arguments are passed in registers R0-R6.
- Results are returned in R0-R3.
- Registers R4-R14 are callee-saved and must be preserved over the SMC call.

When an SMC32 call is made from AArch64:

- Arguments are passed in registers W0-W6.
- Results are returned in W0-W3.
- Registers X18-X30 and stack pointers SP_EL0 and SP_ELx are callee-saved and must be preserved over the SMC call.

Note: Unused result and scratch registers can leak information after an SMC call. An implementation can mitigate this risk by either preserving the register state over the call, or returning a constant value, such as zero, in each register.

Note: SMC32 calls from AArch32 and AArch64 use the same physical registers for arguments and results, since registers W0-W7 in AArch64 are equivalent to R0-R7 in AArch32.

2.3 SMC64 Argument passing

When the SMC64 convention is used, the SMC instructions take up to seven 64-bit arguments in registers and can return up to four 64-bit values in registers.

When an SMC64 call is made from AArch64:

- Arguments are passed in registers X0-X6.
- Results are returned in X0-X3.
- Registers X18-X30 and stack pointers SP_EL0 and SP_ELx are callee-saved and must be preserved over the SMC call.

This calling convention cannot be used by code executing AArch32 state.

- Any SMC64 calls from AArch32 state will receive the Unknown SMC Function Identifier result, see section 5.1.

Note: Unused result and scratch registers can leak information after an SMC call. An implementation can mitigate this risk by either preserving the register state over the call, or returning a constant value, such as zero, in each register.

2.4 SIMD and Floating-point registers

SIMD and floating-point registers must not be used to pass arguments to or receive results from any SMC call.

All SIMD and floating-point registers are callee-saved and must be preserved over all SMC calls.

2.5 SMC immediate value

The SMC instruction encodes an immediate value as defined by the ARM architecture [1][2]. The size of this and mechanism to access the immediate value differ between the ARM instruction sets. Additionally, it is time consuming for 32-bit Secure Monitor code to access this immediate value. Consequently:

- An SMC immediate value of Zero must be used.
- All other SMC immediate values are reserved.

2.6 Hypervisor Client ID

If an implementation includes a hypervisor or similar supervisory software executing at EL2 then it may be necessary to identify which client operating system the SMC call originated from.

- A 32-bit hypervisor client ID parameter is defined for SMC calls.
- In AArch32, the hypervisor client ID is passed in the R7 register.
- In AArch64, the hypervisor client ID is passed in the W7 register.
- The hypervisor client ID of 0x00000000 is designated for SMC calls from the hypervisor itself.

The hypervisor client ID is expected to be created within the hypervisor and used to register, reference and de-register client operating systems to a Trusted OS. Is not expected to correspond to the VMIDs used by the MMU.

All SMC calls generated by software executing at EL1 should be trapped by the hypervisor. Identification information should be inserted into R7 or W7 before forwarding any SMC call on to the Secure Monitor.

2.7 Trusted OS Session ID (optional)

To support multiple sessions with in the Trusted OS, it may be necessary to identify multiple instances of the same SMC call.

- An optional 32-bit Session ID is defined for SMC calls.
- In AArch32, the Session ID is passed in the R6 register.
- In AArch64, the Session ID is passed in the W6 register.

It is expected that the Session ID is provided by the Trusted OS, and is used by its clients in subsequent calls.

3 AARCH64 SMC CALLING CONVENTIONS

This specification defines two common calling mechanisms for use with the SMC instruction from the AArch64 state, known as SMC32 and SMC64.

For ARM AArch64 systems all Trusted OS and Secure Monitor implementations must conform to this specification.

3.1 Register use in AArch64 SMC calls

The same architectural registers, R0-R7, are used for the two AArch64 calling conventions, SMC32 and SMC64.

The working size of the register is identified by its name:

Xn	All 64-bits used.
Wn	Lower 32-bits used, upper 32-bits are zero.

Table 3-1 Register Usage in AArch64 SMC32 and SMC64 calls

Register Name		Role during SMC call		
SMC32	**SMC64**	**Calling values**	**Modified**	**Return state**
SP_ELx		ELx Stack Pointer	No	Unchanged, Registers are saved/restored
SP_EL0		EL0 Stack Pointer	No	
X30		The Link Register	No	
X29		The Frame Pointer	No	
X19…X28		Callee-saved registers	No	
X18		The Platform Register	No	
X17		The second intra-procedure-call scratch register.	Yes	Unpredictable, Scratch registers
X16		The first intra-procedure-call scratch register.	Yes	
X9…X15		Temporary registers	Yes	
X8		Indirect result location register	Yes	
W7	W7	Hypervisor Client ID register	Yes	
W6	X6 (or W6)	Parameter register Optional Session ID register	Yes	
W4…W5	X4…X5	Parameter registers	Yes	
W1…W3	X1…X3	Parameter registers	Yes	SMC Result registers
W0	X0	SMC Function ID	Yes	

For more information see [4] Procedure Call Standard for the ARM 64-bit Architecture

4 ARCH32 SMC CALLING CONVENTION

This specification defines a common calling mechanism for use with the SMC instruction from the AArch32 state, also known as SMC32.

Note: ARM recognizes that a number of vendors already use a proprietary calling convention and won't be able to meet all of the following requirements.

4.1 Register use in AArch32 SMC calls

Table 4-1 Register usage in AArch32 SMC Calls

Register SMC32	Role during SMC call		
	Calling values	**Modified**	**Return state**
R15	The Program Counter	Yes	Next instruction
R14	The Link Register	No	Unchanged, Registers are saved/restored
R13	The Stack Pointer	No	
R12	The Intra-Procedure-call scratch register	No	
R11	Variable-register 8	No	
R10	Variable-register 7	No	
R9	Platform register.	No	
R8	Variable-register 5	No	
R7	Hypervisor Client ID register	No	
R6	Parameter register 6 Optional Session ID	No	
R5	Parameter register 5	No	
R4	Parameter register 4	No	
R3	Parameter register 3	Yes	SMC results registers
R2	Parameter register 2	Yes	
R1	Parameter register 1	Yes	
R0	SMC Function Identifier	Yes	

For more information see [3] Procedure Call Standard for the ARM 32-bit Architecture.

5 SMC STANDARD RESULTS

5.1 Unknown SMC Function Identifiers

The Unknown SMC Function Identifier is a 32-bit value of `0xFFFFFFFF` returned in R0. The same return value is used by SMC32 and SMC64 calls.

An implementation must return this value when it receives an:

- SMC call with an unknown function identifier
- SMC call for a removed function identifier
- SMC64 call from AArch32 state

Note: The Unknown SMC Function Identifier should not be used to discover the presence, or lack of, an SMC Function. SMC Function Identifiers should be determined from the UID and Revision information.

5.2 Unique Identification (UID) format

This value identifies the owner of a particular sub-range of the API, and therefore who controls the actions of SMCs in that sub-range.

The UID is a UUID as defined by RFC 4122 [6]. These UUIDs must be generated by any method defined by RFC 4122 [6], and are 16 bytes strings.

UIDs are returned as a single128-bit value using the SMC32 calling convention. This is mapped to argument registers as shown in Table 5-1.

Table 5-1: UUID register mapping

Register		Value
AArch32	**AArch64**	
R0	W0	Bytes 0…3 with byte 0 in the low order bits
R1	W1	Bytes 4…7 with byte 4 in the low order bits
R2	W2	Bytes 8…11 with byte 8 in the low order bits
R3	W3	Bytes 12…15 with byte 12 in the low order bits

UIDs with the first 32-bits set to `0xFFFFFFFF` (i.e. the value of R0 or W0) should be avoided since they are indistinguishable from Unknown SMC Function Identifiers (see section 0).

5.3 Revision information format

The revision information for a sub-range is defined by a 32-bit major version and a 32-bit minor version.

Different major revision values indicate possibly incompatible SMC APIs, for the affected SMC range.

For two revisions, *A* and *B*, for which the major revision values are identical, if the minor revision value of revision *B* is greater than the minor revision value of revision *A*, then every SMC in the affected range that works in revision *A* must also work, with a compatible effect, in revision *B*.

When returned by a call, the major version is returned in R0 or W0 and the minor version is returned in R1 or W1. Such an SMC must use the SMC32 calling convention.

The rules for interface updates are:

- An SMC function identifier once issued must never be re-used.
- Additional SMC calls must take a new unused SMC identifier.
- Calls to removed SMC identifiers must return the Unknown SMC Function Identifier value.
- Incompatible argument changes cannot be made to an existing SMC call, a new call is required.
- Major revision number must be incremented when:
    - Any SMC call is removed.
- Minor revision number must be incremented when:
    - Any SMC call is added.
    - Backwards compatible changes are made to existing function arguments

6 SMC IDENTIFIER RANGES

6.1 Allocation of Values

The following tables show the recommended allocation of SMC identifier value ranges for different entities and purposes. The owner of a range is the entity who is responsible for that function in a specific SoC. The same entity can be responsible for multiple sub-ranges.

Table 6-1: SMC Identifier Sub-range ownership

SMC Function Identifier	SMC sub-range ownership	Notes
0x00000000-0x0100FFFF	Reserved for existing APIs	This region is already in use by ARMv7 devices on the field.
0x02000000-0x7FFFFFFF	Trusted OS	Trusted OS Standard Calls
0x80000000-0x8000FFFF	SMC32: ARM Architecture Calls	
0x81000000-0x8100FFFF	SMC32: CPU Service Calls	
0x82000000-0x8200FFFF	SMC32: SiP Service Calls	
0x83000000-0x8300FFFF	SMC32: OEM Service Calls	
0x84000000-0x8400FFFF	SMC32: Standard Service Calls	
0x85000000-0xAF00FFFF	Reserved for future expansion	
0xB0000000-0xB100FFFF	SMC32: Trusted Application Calls	
0xB2000000-0xBF00FFFF	SMC32: Trusted OS Calls	
0xC0000000-0xC000FFFF	SMC64: ARM Architecture Calls	
0xC1000000-0xC100FFFF	SMC64: CPU Service Calls	
0xC2000000-0xC200FFFF	SMC64: SiP Service Calls	
0xC3000000-0xC300FFFF	SMC64: OEM Service Calls	
0xC4000000-0xC400FFFF	SMC64: Standard Service Calls	
0xC5000000-0xEF00FFFF	Reserved for future expansion	
0xF0000000-0xF100FFFF	SMC64: Trusted Application Calls	
0xF2000000-0xFF00FFFF	SMC64: Trusted OS Calls	

All Function Identifier ranges not listed in this table are reserved.

Table 6-2: Trusted OS SMC range

SMC Function Identifier	Reserved use and sub-range ownership	Notes
	Owner: Trusted OS	
0x02000000-0x1FFFFFFF	General Trusted OS	Trusted OS dependent SMC usage. Typically this channel is used to create an asynchronous API to Trusted Services.
0x20000000-0x7FFFFFFF	Reserved for future expansion	

These values are used in a Trusted OS specific way to implement the Standard Call functionality.

Note: Trusted OS identification and revision details can be discovered through the Trusted OS Fast Call identification and revision interface – see Table 6-10.

Table 6-3: ARM Architecture Call range

SMC Function Identifier	Reserved use and sub-range ownership	Notes
	Owner: ARM Service Calls	
0x8000000-0x8000FEFF	SMC32: ARM Service Calls	
0x8000FF00	SMC32: ARM Architecture Call Count	This call returns a 32-bit count of the available Service Calls. A return value of zero means no services are available.
0x8000FF01	SMC32: ARM Architecture Call UID	Each implementation of ARM Architecture Calls must provide a unique Identifier (UID).
0x8000FF02	Reserved	
0x8000FF03	SMC32: ARM Architecture Call Revision details	Each variant of a UID implementation must provide revision details.
0x8000FF04-0x8000FFFF	Reserved for future expansion	
0xC0000000-0xC000FFFF	SMC64: ARM Architecture Calls	

The ARM Architecture Calls provide interfaces to generic services for the ARM Architecture.

Table 6-4: CPU Service Calls range

SMC Function Identifier	Reserved use and sub-range ownership	Notes
	Owner: CPU Service Calls	
0x81000000-0x8100FEFF	SMC32: CPU Service Calls	
0x8100FF00	SMC32: CPU Service Call Count	This call returns a 32-bit count of the available Service Calls. A return value of zero means no services are available.
0x8100FF01	SMC32: CPU Service Call UID	Each Implementation of CPU Service Calls must provide a unique Identifier.
0x8100FF02	Reserved	
0x8100FF03	SMC32: CPU Service Call Revision details	Each update may provide revision details. The structure of this data is CPU dependent.
0x8100FF04-0x8100FFFF	Reserved for future expansion	
0xC100FF00-0xC100FFFF	SMC64: CPU Service Calls	

The CPU Service Calls provide interfaces to CPU implementation-specific services for this platform. Such as access to errata work-arounds.

Table 6-5: SiP Service Calls range

SMC Function Identifier	Reserved use and sub-range ownership	Notes
	Owner: SiP Service Calls	
0x82000000-0x8200FEFF	SMC32: SiP Service Calls	
0x8200FF00	SMC32: SiP Service Call Count	This call returns a 32-bit count of the available Service Calls. A return value of zero means no services are available.
0x8200FF01	SMC32: SiP Service Call UID	Each Implementation of SiP Service Calls must provide a unique Identifier.
0x8200FF02	Reserved	
0x8200FF03	SMC32: SiP Service Call Revision details	Each update can provide revision details. The structure of this data is SiP dependent.
0x8200FF04-0x8200FFFF	Reserved for future expansion	
0xC200FF00-0xC200FFFF	SMC64: SiP Service Calls	

The SiP Service Calls provide interfaces to SoC implementation specific services on this platform.
For example, Secure platform initialization, configuration and some power control.

Table 6-6: OEM Service Call range

SMC Function Identifier	Reserved use and sub-range ownership	Notes
	Owner: OEM Service Calls	
0x83000000-0x8300FEFF	SMC32: OEM Service Calls	
0x8300FF00	SMC32: OEM Service Call Count	This call returns a 32-bit count of the available Service Calls. A return value of zero means no services are available.
0x8300FF01	SMC32: OEM Service Call UID	Each Implementation of OEM Service Calls must provide a unique Identifier. Typically it is expected that there is one UID per OEM.
0x8300FF02	Reserved	
0x8300FF03	SMC32: OEM Service Call Revision details	Each update can provide revision details. The structure of this data is OEM dependent.
0x8300FF04-0x8300FFFF	Reserved for future expansion	
0xC300FF00-0xC300FFFF	SMC64: OEM Service Calls	

The OEM Service Calls provide interfaces to OEM-specific services on this platform.

Table 6-7: Standard Service Call range

SMC Function Identifier	Reserved use and sub-range ownership	Notes
	Owner: Standard Service Calls	
0x84000000-0x8400001F	PSCI SMC32 bit Calls	A range of SMC calls. See [5] for details of functions and arguments.
0x84000020-0x8400FEFF	SMC32: Standard Service Calls	Service calls defined by ARM standards. The arguments are defined by the relevant ARM standard.
0x8400FF00	SMC32: Standard Service Call Count	This call returns a 32-bit count of the available Service Calls. A return value of zero means no services are available.
0x8400FF01	SMC32: Standard Service Call UID	Each Implementation of Standard Service Calls must provide a unique Identifier (UID).
0x8400FF02	Reserved	
0x8400FF03	SMC32: Standard Service Call Revision details	This SMC returns the revision information for the Standard service calls.
0x8400FF04-0x8400FFFF	Reserved for future expansion	
0xC4000000-0xC400001F	PSCI SMC64 bit Calls	A range of SMC calls. See [5] for details of functions and arguments.
0xC4000004-0xC400FEFF	SMC64: Standard Service Calls	Service calls defined by ARM standards. The arguments are defined by the relevant ARM standard.
0xC4FFFF00-0xC4FFFFFF	Reserved for future expansion	

ARM intends to define a set of standard Service Calls for the management of the overall system. By standardizing such calls the job of implementing Operating Systems on ARM will be made easier.

The first of these standards is the Power State Coordination Interface [5].

Note: Standard Service identifiers need to be understood by a Hypervisor when it traps SMC calls because it must know which SMC calls are for power control and similar operations so that it can emulate these calls for its clients.

Table 6-8: Reserved for future expansion

SMC Function Identifier	Reserved use and sub-range ownership	Notes
	Reserved for future expansion	
0x85000000-0xEF00FFFF	Reserved for future expansion	

Table 6-9: Trusted Application Call range

SMC Function Identifier	Reserved use and sub-range ownership	Notes
	Owner: Trusted Application Calls	
0xB0000000-0xB100FFFF	SMC32: Trusted Application Calls	
0xF0000000-0xF100FFFF	SMC64: Trusted Application Calls	

Note: It is the responsibility of a Trusted OS to identify and describe services provided by Trusted Applications

Table 6-10: Trusted OS Call range

SMC Function Identifier	Reserved use and sub-range ownership	Notes
	Owner: Trusted OS calls	
0xB2000000-0xBF00FEFF	SMC32: Trusted OS Calls	
0xBF00FF00	SMC32: Trusted OS Calls Count	This call returns a 32-bit count of the available Service Calls. A return value of zero means no services are available.
0xBF00FF01	SMC32: Trusted OS Calls UID	Each Implementation of a Trusted OS Call must provide a unique Identifier. A return value of 0 indicates that no Trusted OS is present.
0xBF00FF02	Reserved	
0xBF00FF03	SMC32: Trusted OS Call Revision details	Each update can provide revision details. The structure of this data is OEM-dependent.
0xBF00FF04-0xBF00FFFF	Reserved for future expansion	
0xF2000000-0xFF00FFFF	SMC64: Trusted OS calls	