Trusted firmware for A-profile systems

Introduction to TF-A

Trainer: Sumit Garg
Linaro Support and Solutions Engineering

Linaro
Developer Services

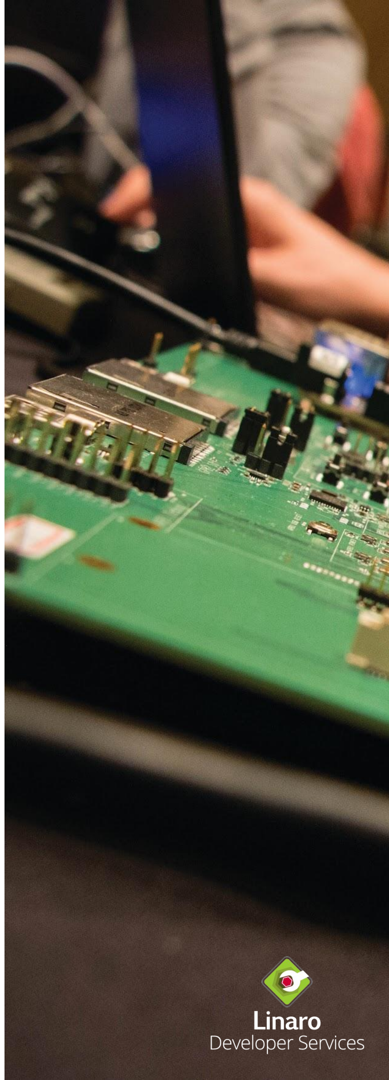
Logistics

These sessions will be recorded. Turn off your camera if you do not want to appear in the recording.

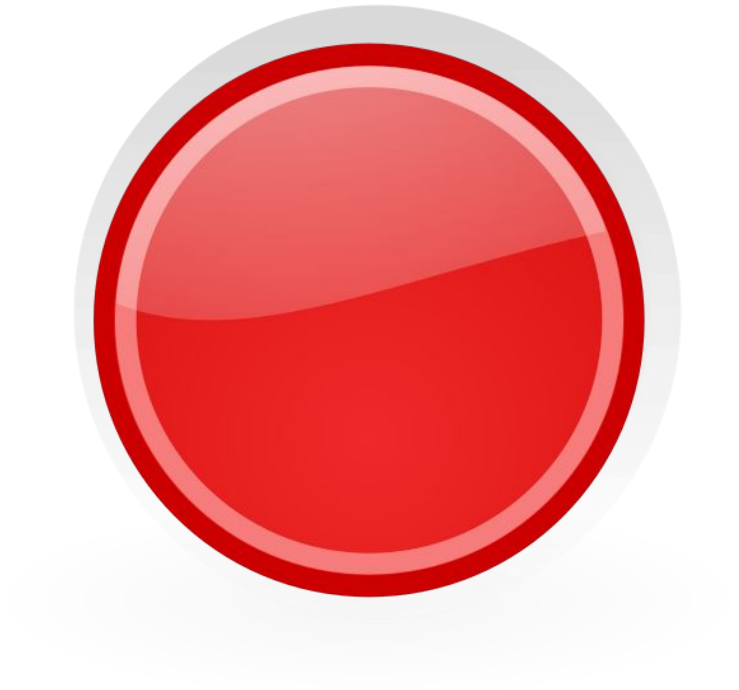
Questions welcome! There is time allocated for Q&A at the end of today's session but you can ask relevant questions verbally or in the chat as we go.

Please keep your microphone muted when not speaking!

Slides and lab resources can be downloaded from:
https://fileserver.linaro.org/s/fE6iBYca8bYqrrk



Linaro
Developer Services



Biography: Sumit Garg

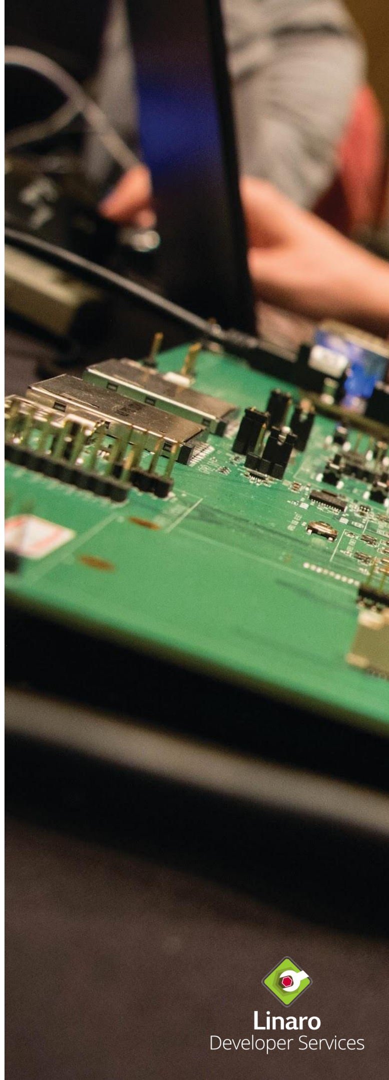
For most of my early career I worked at NXP where I joined the platform team with a primary focus on security aspects across **the entire platform software stack**.

I have been contributing to TF-A and OP-TEE since 2015. The contributions are ranging from **maintaining multiple platforms** to adding **core features** to the role of **Linux's TEE subsystem reviewer** and maintaining **firmware encryption** framework in TF-A.

I joined Linaro in 2018 as a member of the Support and Solutions Engineering team. My work at Linaro is varied but all of it is centred around my background in **platform security** inclusive of **my role as a trainer**. I also help Linaro members/customers in my other areas of interest including toolchains and embedded Linux distributions such as OpenEmbedded where, among other things, I **maintain the meta-arm-toolchain layer**.

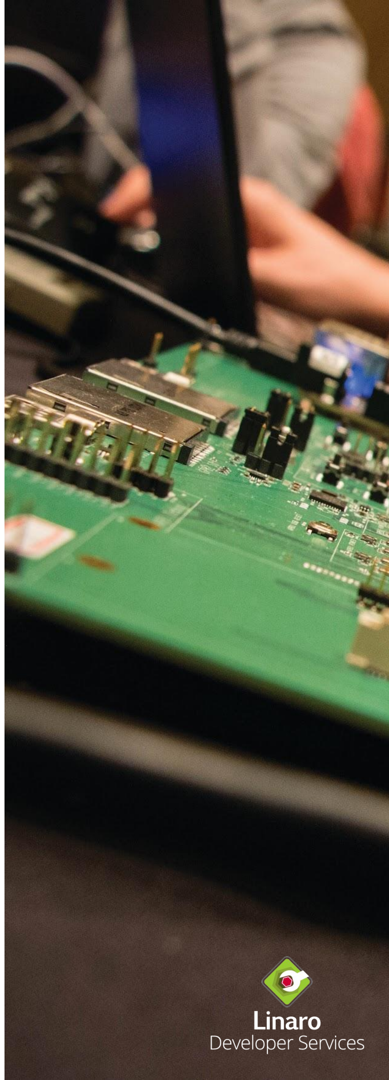
Training modules overview

1. **Introduction to TF-A**
2. Generic boot
3. Firmware security
4. Secure/Realm world interfaces



Introduction to TF-A

- **Arm A-Profile Architecture evolution**
  - **ARMv7-A -> ARMv8-A**
  - **Armv8-A -> ARMv9-A**
- About TF-A project
  - History and origins
- TF-A as EL3 firmware
  - Firmware components
- Handling Secure Monitor Call
- Context management
- Power State Coordination Interface



Arm A-Profile Architecture

The Arm Application-profile (A-profile) architecture targets high performance markets, such as PC, mobile, gaming, and enterprise. The application profile supports:

- A **Virtual Memory System Architecture** (VMSA) based on a Memory Management Unit (MMU).
- The **A64, A32, and T32** instruction sets.

Execution states:

- **AArch64**: Is the 64-bit Execution state, meaning **addresses are held in 64-bit registers**, and instructions in the base instruction set can use 64-bit registers for their processing. AArch64 state supports the **A64 instruction set**.

- **AArch32**: Is the 32-bit Execution state, meaning **addresses are held in 32-bit registers**, and instructions in the base instruction sets use 32-bit registers for their processing. AArch32 state supports the **T32 and A32 instruction sets**.

Arm A-Profile Architecture - II

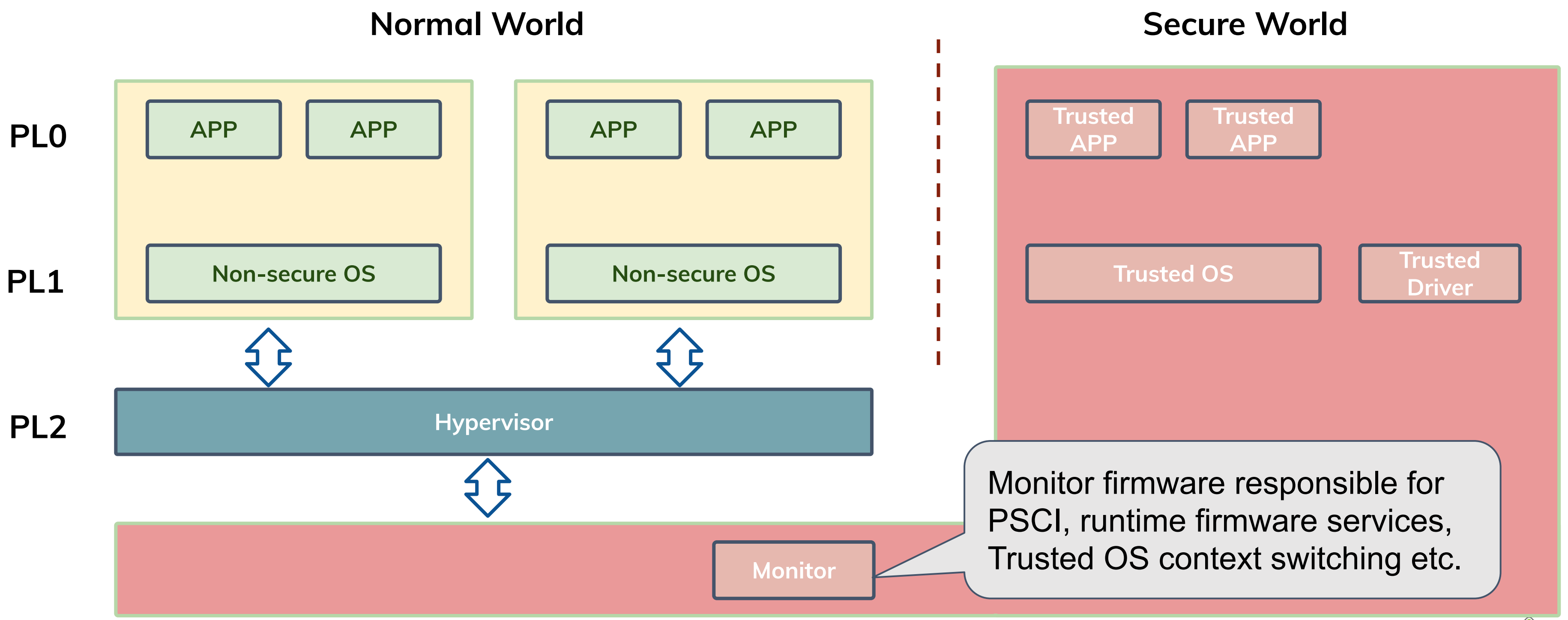
A-profile architecture version mapping for the execution states. Here we will only discuss the last three architecture versions namely, Armv7-A, Armv8-A and Armv9-A.

Armv7-A: Supports **32-bit** execution state with ARM (A32) and Thumb (T32) instruction sets, implies AArch32.

Armv8-A: Supports **both** execution states AArch64 and AArch32 (to be backwards compatible with 32-bit software stack).

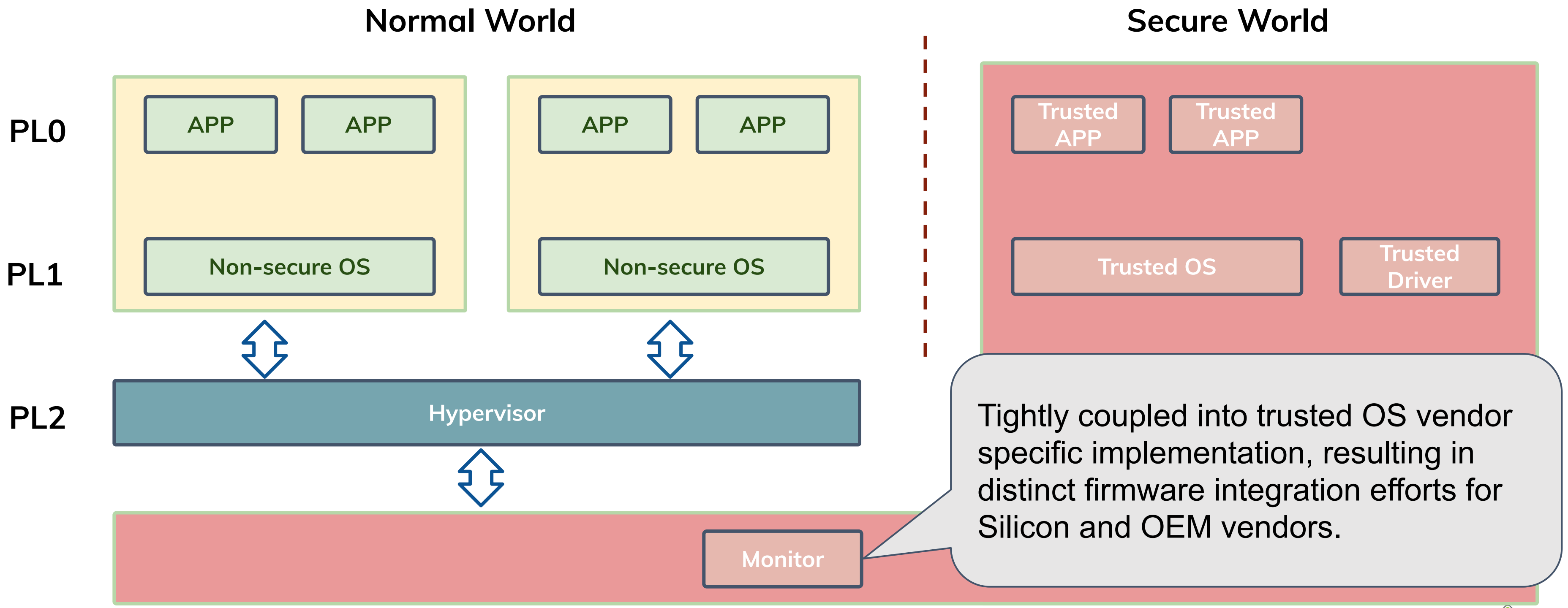
Armv9-A: Supports **AArch64** execution state. The AArch32 state might **optionally** be implemented at EL0. The AArch32 state is **not implemented** at EL1, EL2 and EL3.

ARMv7-A privilege levels



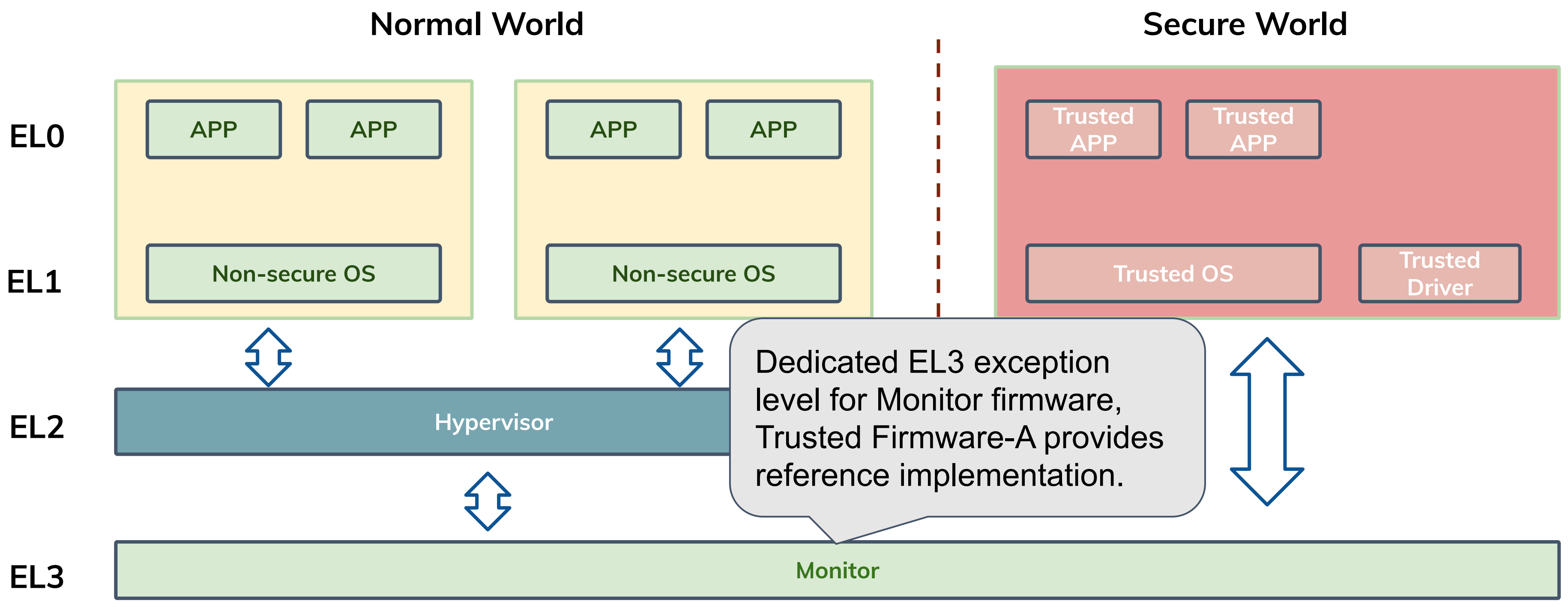
Monitor firmware responsible for PSCI, runtime firmware services, Trusted OS context switching etc.

ARMv7-A: Firmware fragmentation



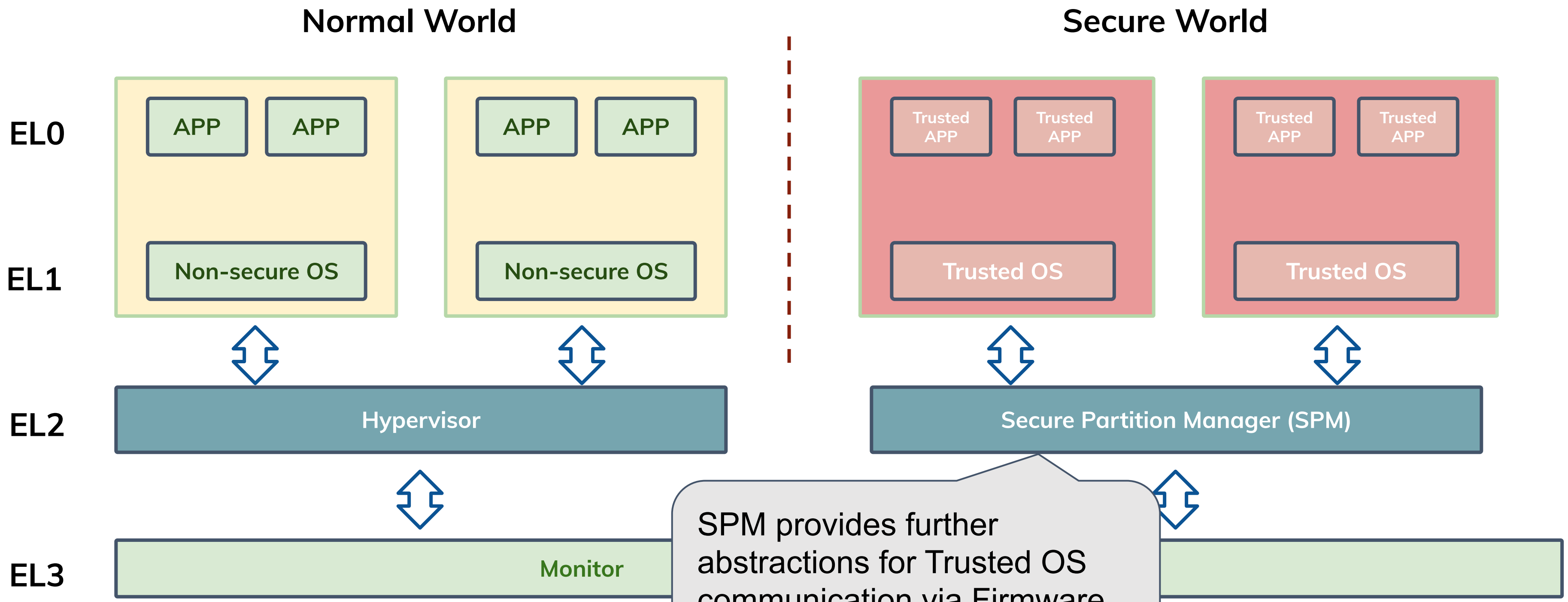
Tightly coupled into trusted OS vendor specific implementation, resulting in distinct firmware integration efforts for Silicon and OEM vendors.

ARMv8-A exception levels



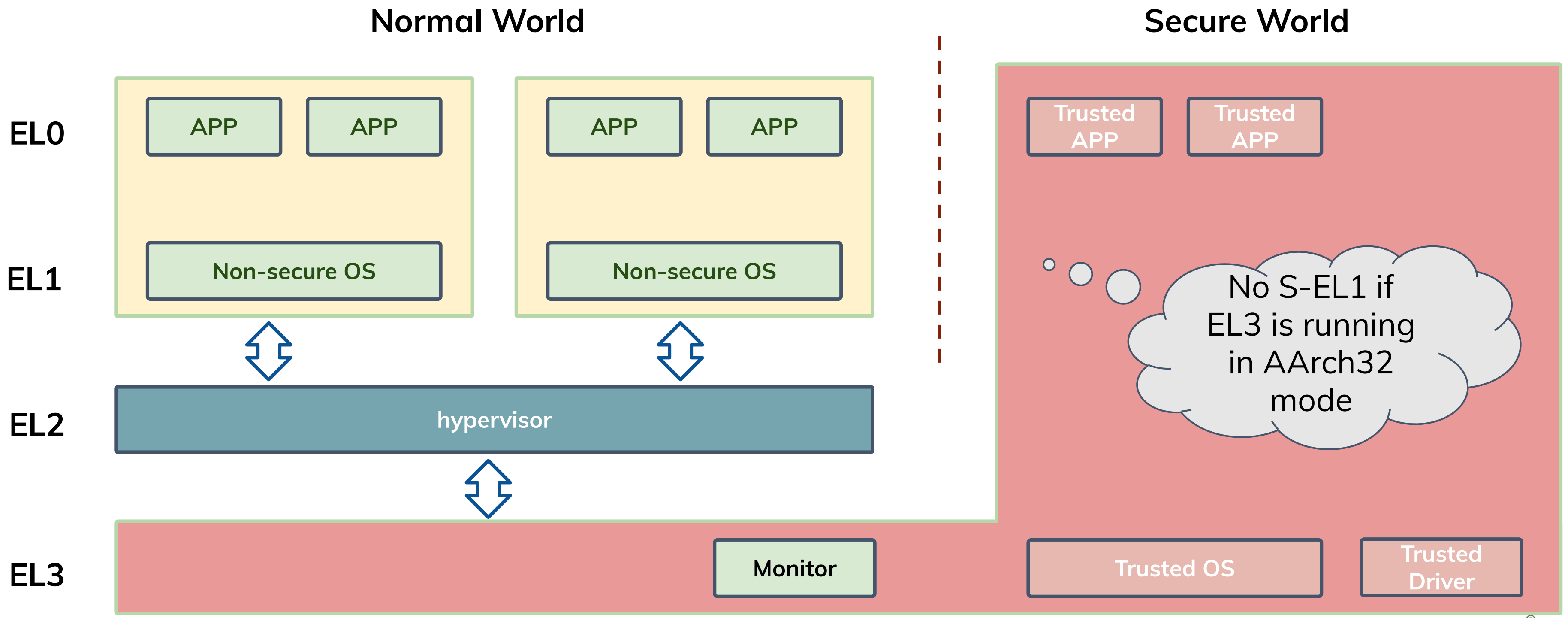
Dedicated EL3 exception level for Monitor firmware, Trusted Firmware-A provides reference implementation.

ARMv8-A exception levels (with S-EL2)



SPM provides further abstractions for Trusted OS communication via Firmware Framework-A (FF-A) interface.

Aside: ARMv8-A exception levels (with 32-bit EL3)



Linaro Developer Services

Aside: A simplified exception model?

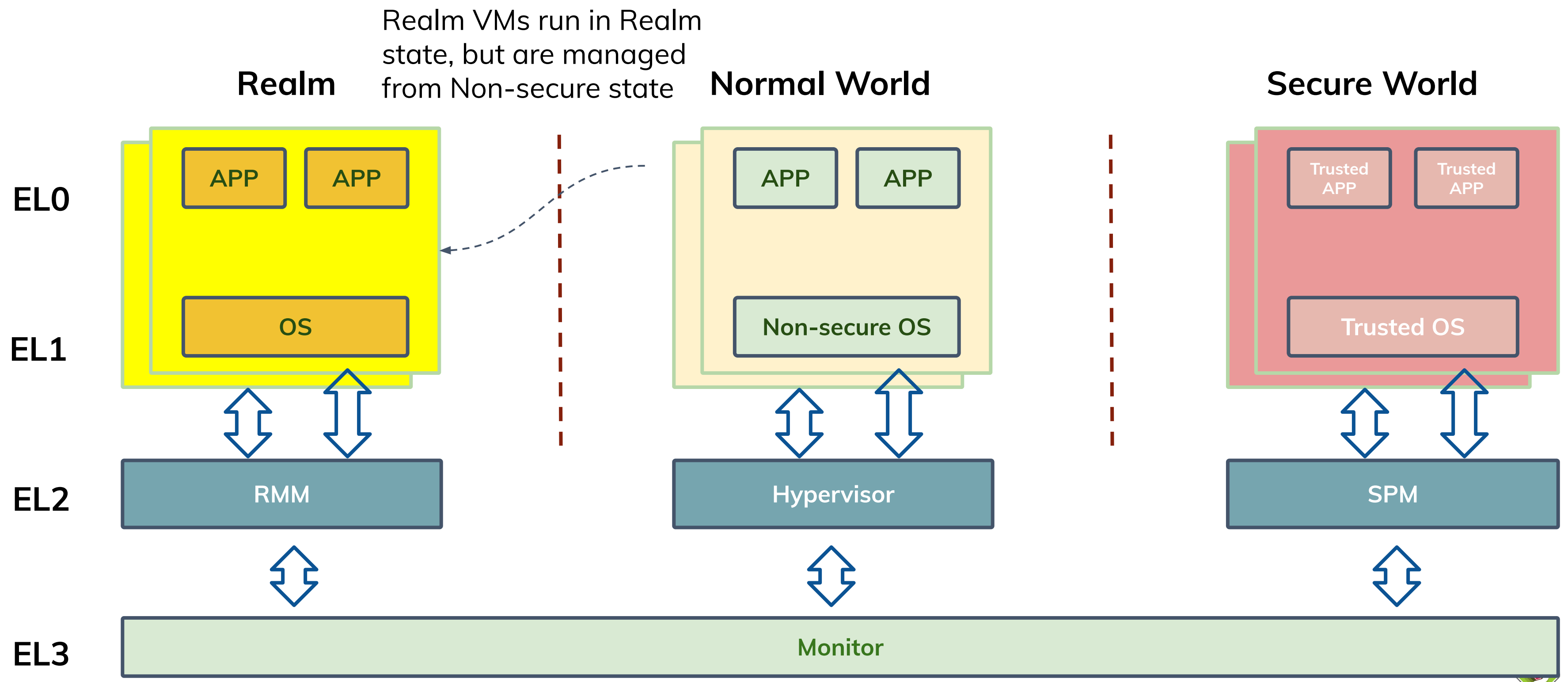
ARMv8-A is often described as having a **simplified exception model** compared to ARMv7-A. Perhaps you are wondering why you haven't seen this yet?

ARMv7-A has a **16 register ISA but it actually has 31 general purpose registers** (excluding Trustzone state) . Most of the **extra registers are banked** and depends on whether you are running in user mode (PL0) or one of the five exception modes (PL1). There are banked registers for each exception mode.

ARMv8-A does introduces a new exception level but it also replaces the exception modes with exception vectors which drastically reduces the number of banked registers. **ARMv8-A/A64 has a 32 register ISA and 35 general purpose registers**.

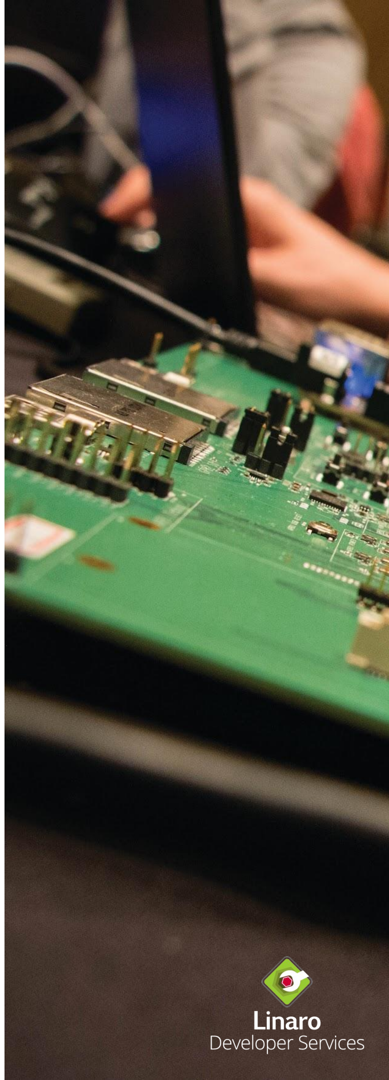
(roughly… and depending how you define general purpose)

ARMv9-A exception levels



Introduction to TF-A

- Arm A-Profile Architecture evolution
  - ARMv7-A -> ARMv8-A
  - Armv8-A -> ARMv9-A
- **About TF-A project**
  - **History and origins**
- TF-A as EL3 firmware
  - Firmware components
- Handling Secure Monitor Call
- Context management
- Power State Coordination Interface



What is Trusted firmware-A?

A reference implementation of:

- **EL3 monitor firmware** for Armv8-A and Armv9-A.
- **Early bootloaders** including BootROM for Arm based application processors.

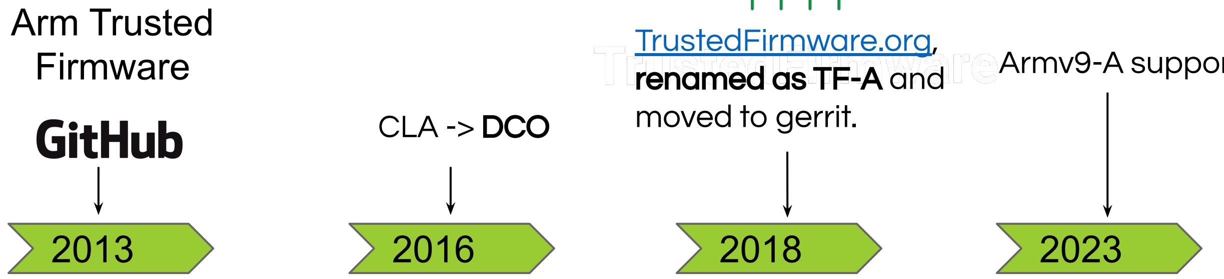
TF-A provides foundation to build a Trusted Execution Environment (**TEE**). It is supported by **35+ platform ports** upstream originating from **20+ different vendors.**

Open source project since **October 2013:**

- **BSD-3-Clause** license
- Contributions accepted under the term of **Developer Certificate of Origin**
- Open governance model on **trustedfirmware.org**
- **6-monthly** releases

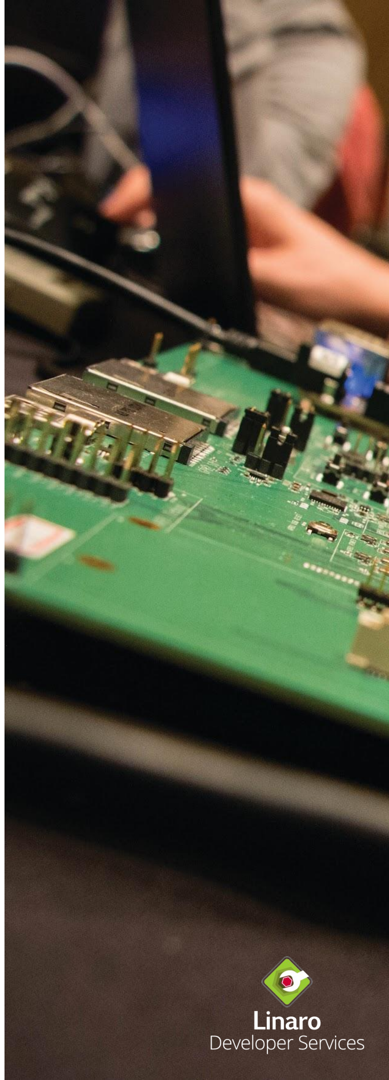
History and origins

- **Arm** open sourced it in October 2013 as "**Arm Trusted Firmware**" on Github
- Been developed for >9 years
- Became widely adopted EL3 firmware



Introduction to TF-A

- Arm A-Profile Architecture evolution
  - ARMv7-A -> ARMv8-A
  - Armv8-A -> ARMv9-A
- About TF-A project
  - History and origins
- **TF-A as EL3 firmware**
  - **Firmware components**
- Handling Secure Monitor Call
- Context management
- Power State Coordination Interface



Role of EL3

EL3 is the **most privileged** exception level where the **runtime resident** monitor firmware executes. TF-A provides **reference implementation** for EL3 monitor firmware.

Provide **runtime services**, such as:

- Arm architectural services
- Standard services such as **PSCI**, **SDEI**, **MM**, **TRNG** etc.
- SiP/OEM specific services
- Foundation to build:
  - Trusted Execution Environment (**TEE**)
  - **Realms** for Confidential Compute Architecture (**CCA**)

to lower exception levels (EL1/EL2 and S-EL1/S-EL2)

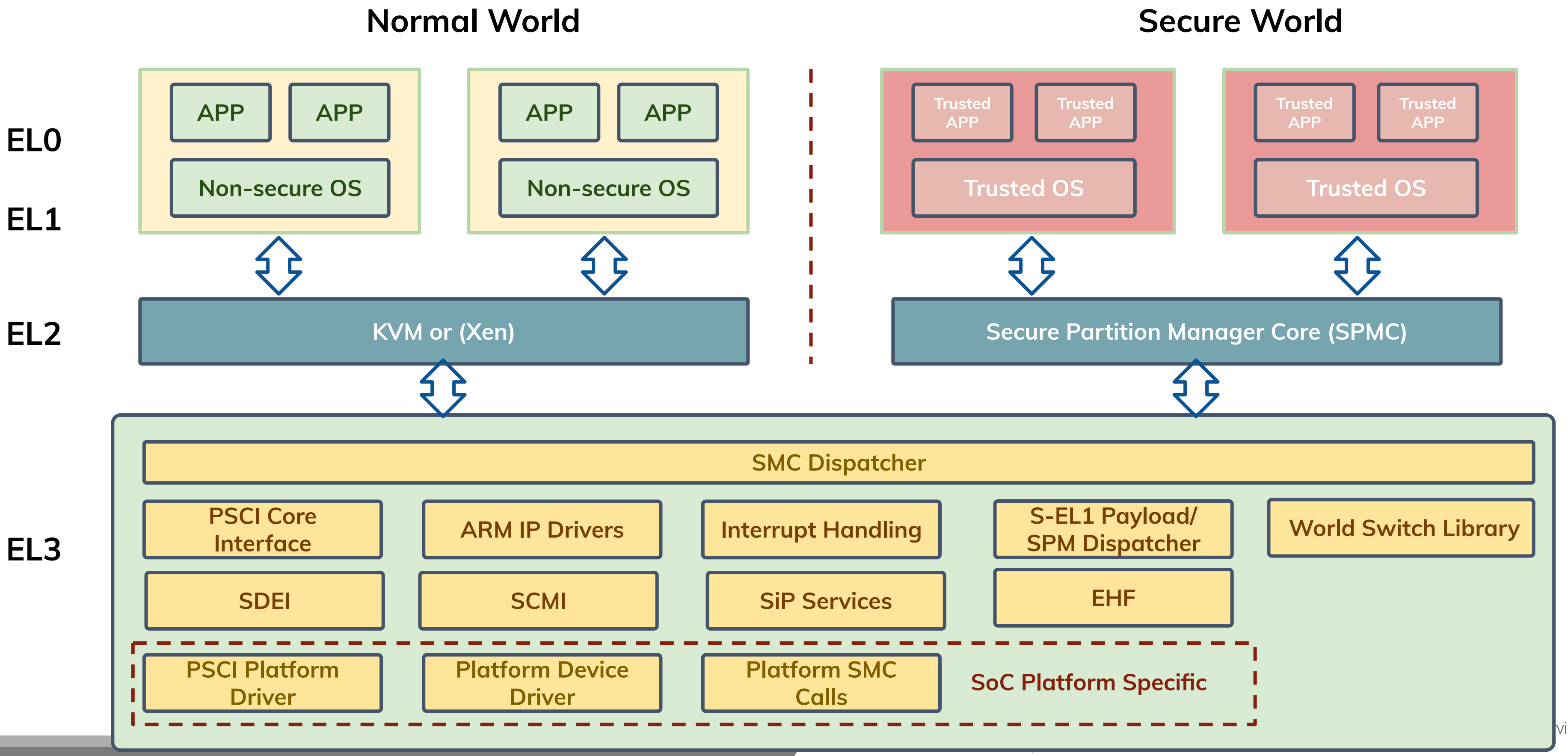
- **EL1 -> Rich OS** or **EL2 -> Hypervisor**
- **S-EL1 -> Trusted OS** or **S-EL2 -> Secure hypervisor**

TF-A EL3 monitor firmware design goals

Some of the major design goals for TF-A EL3 monitor firmware are:

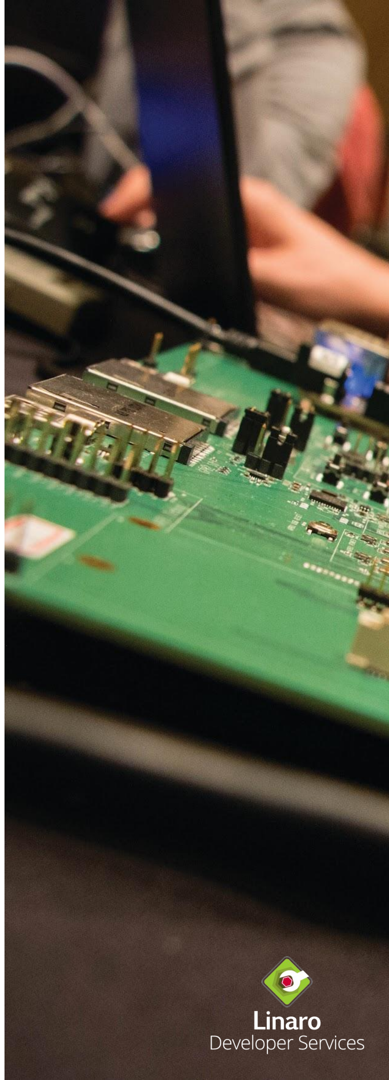
- Aims to have a **minimal and generic** EL3 firmware
  - Move secure and platform specific services to lower exception levels such as S-EL0, S-EL1 and S-EL2.
- Aims to reduce firmware attack surface, has its own **generic firmware threat model**.
- Aims to **reduce firmware complexity** to ease auditing and certification process.
- Promotes standardized interfaces such as **Firmware Framework-A** for communication among normal and secure world.

TF-A: EL3 firmware components



Introduction to TF-A

- Arm A-Profile Architecture evolution
  - ARMv7-A -> ARMv8-A
  - Armv8-A -> ARMv9-A
- About TF-A project
  - History and origins
- TF-A as EL3 firmware
  - Firmware components
- **Handling Secure Monitor Call**
- Context management
- Power State Coordination Interface



Linaro Developer Services

Handling Secure Monitor Call

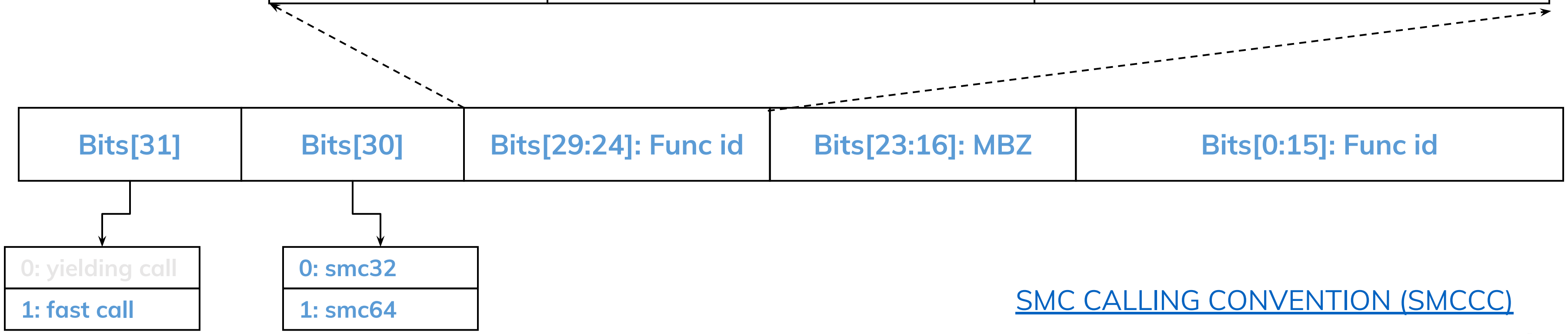
- SMCs may originate from **both secure and non-secure** worlds

- Some SMCs are serviced by TF-A itself, others are **forwarded** to the Secure EL1/EL2 payload
    - **S-EL1/EL2 payload dispatcher** is linked into TF-A and is responsible for forwarding calls
    - Dispatcher copies register values between the worlds and, using library calls, **saves context and controls the world switch**

- SMC call type may be either **fast or yielding**
    - Fast calls are **atomic and uninterruptible**.
    - Yielding calls can be **pre-empted**, and the call can return before the requested operation has completed.
    - All **TF-A services calls are fast calls** (thus TF-A does not need any machinery to track thread states).

Fast SMC calling convention

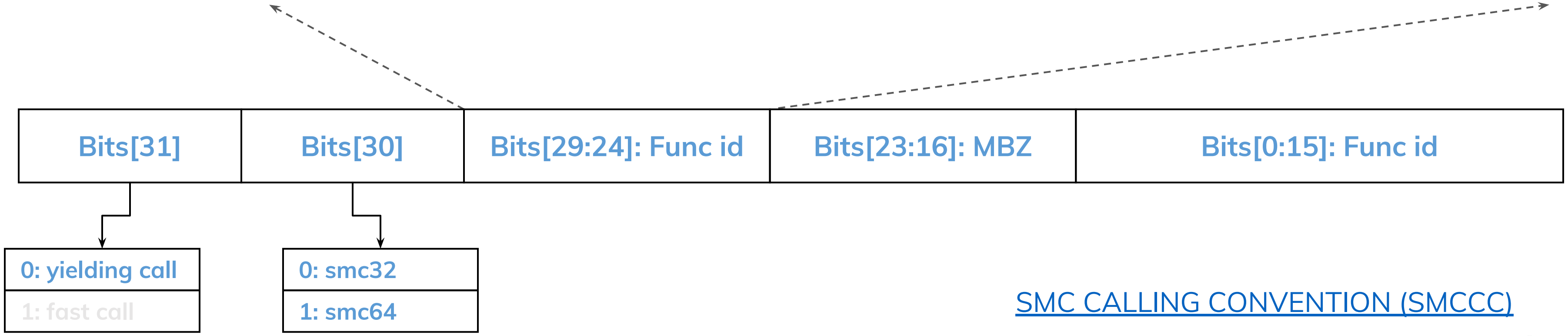
Entity Number	Bit Mask	Description
0	0x00000000	ARM Architecture Calls
1	0x01000000	CPU Service Calls
2	0x02000000	SIP Service Calls
3	0x03000000	OEM Service Calls
4	0x04000000	Standard Service Calls
5 - 47	0x05000000 – 0x2F000000	Reserved for future use
48 - 49	0x30000000 – 0x31000000	Trusted Application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



SMC CALLING CONVENTION (SMCCC)

Yielding SMC calling convention

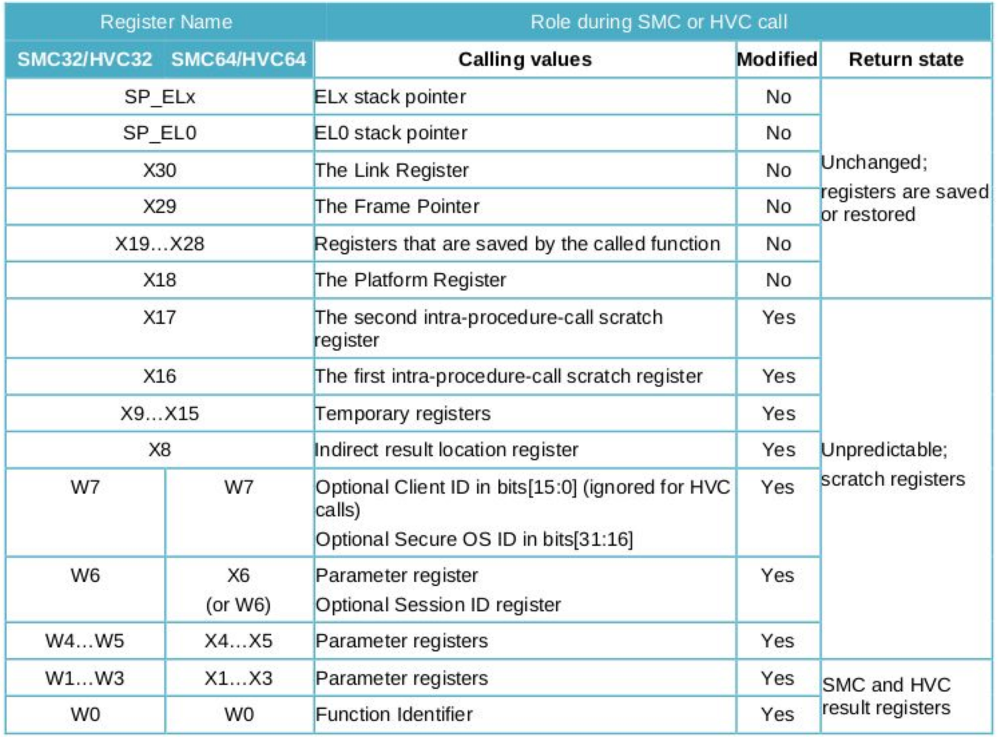
Entity Number	Bit Mask	Description
0-1	0x00000000 – 0x0100FFFF	Reserved because this region is already in use by ARMv7 devices on the field. Strictly speaking this is reserved only when Bit[30] is set for smc32.
2-31	0x02000000 – 0x1FFFFFF	Trusted OS Calls
32-63	0x20000000 – 0x3F000000	Reserved for future expansion of Trusted OS Yielding Calls



[SMC CALLING CONVENTION (SMCCC)]

Linaro Developer Services

AArch64 ABI for SMC calls



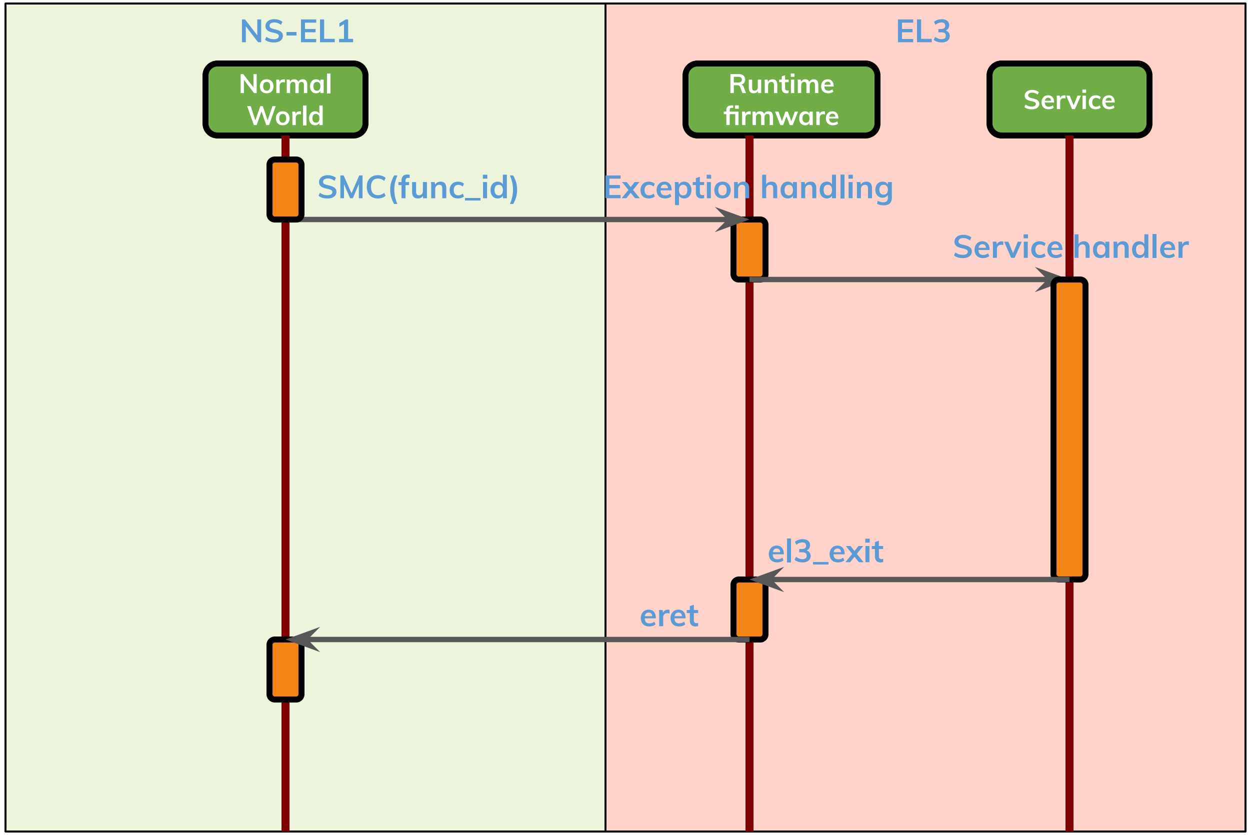
Register Name		Role during SMC or HVC call		
SMC32/HVC32	SMC64/HVC64	Calling values	Modified	Return state
SP_ELx		ELx stack pointer	No	Unchanged; registers are saved or restored
SP_EL0		EL0 stack pointer	No	
X30		The Link Register	No	
X29		The Frame Pointer	No	
X19…X28		Registers that are saved by the called function	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	Optional Client ID in bits[15:0] (ignored for HVC calls) Optional Secure OS ID in bits[31:16]	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 and HVC result registers
W0	W0	Function Identifier	Yes	

Same as *Procedure Call Standard for the ARM 64-bit Architecture* except that LR is also preserved

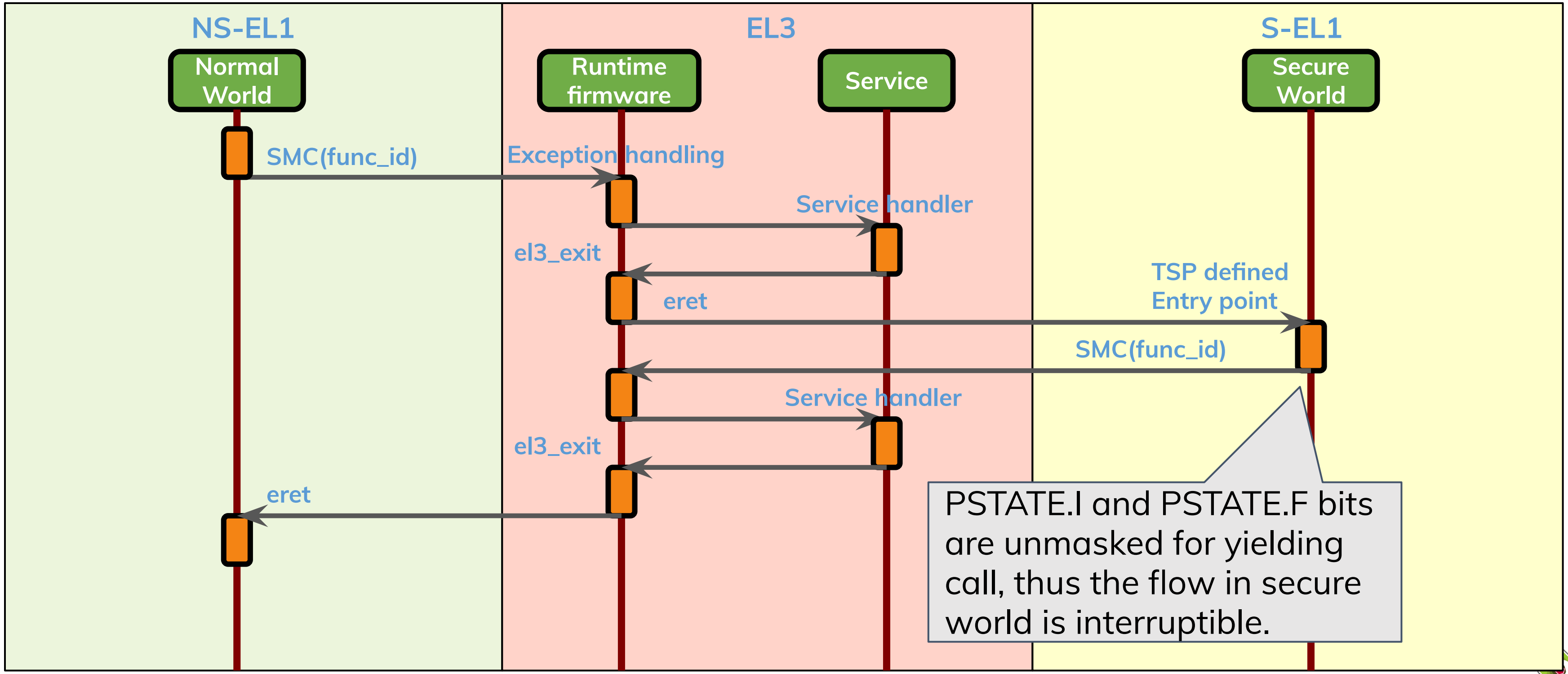
SMC CALLING CONVENTION (SMCCC)

Linaro Developer Services

TF-A runtime service calls handling



TF-A Trusted OS calls forwarding



PSTATE.I and PSTATE.F bits are unmasked for yielding call, thus the flow in secure world is interruptible.

Commercial break

Based on feedback from previous courses…

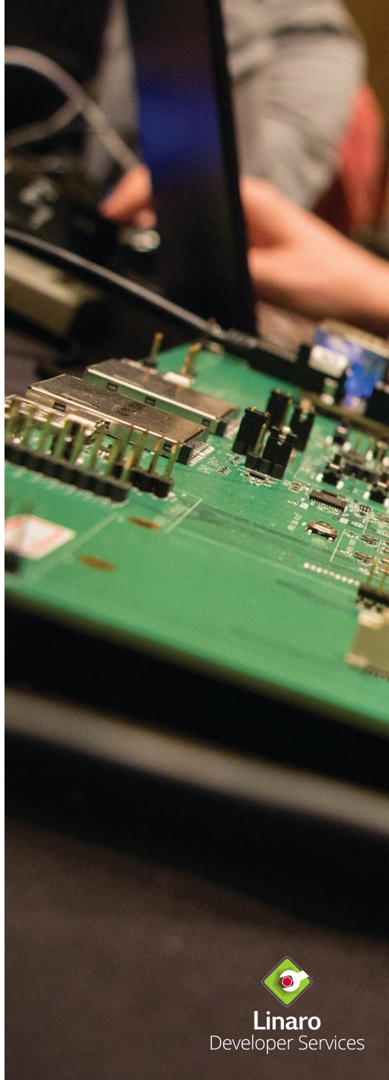
… we moved this to the middle…

… because it is important …

… and after two hours it is hard to remember this bit

Remember this for any post-training support…

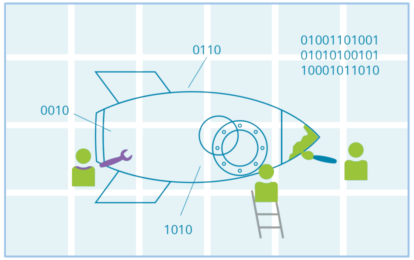
… even simple things like not being able to find the slides



Developer Services help build, test and deploy your products on Arm

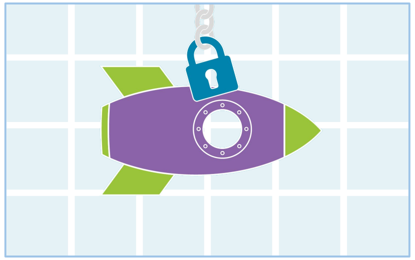
Leverage our Arm expertise

As part of Linaro, Developer Services has access to some of the world's **leading Arm Software experts.** All of this expertise and experience is made available to you on your project.



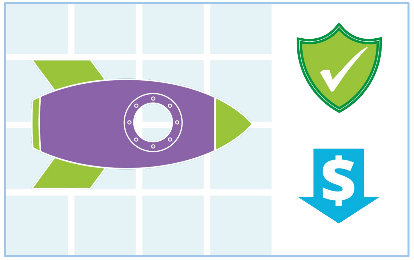
Secure your product

Specialists in security and Trusted Execution Environment (TEE) on Arm, we leverage open source to ensure you **benefit from the latest upstream features and security fixes**.



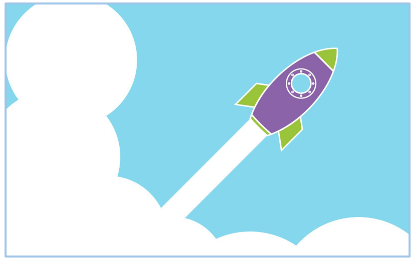
Maintain quality cost-effectively

We upstream code to **reduce the cost and effort needed to maintain your product**. We offer continuous integration (CI) and automated validation for your product software, ensuring the **highest possible quality**.



Build, Test and deploy faster

We support every aspect of product delivery, from building secure board support packages (BSPs), product validation and long-term maintenance - we help get your **products to market faster**.



Linaro
Developer Services

Linaro Training & Educational Services

Linaro provides **hands-on training for software developers** who **work on Arm**

Expert Instructors who are **real world engineers** deliver hands-on training

Training delivered **online or onsite**

Training structured to your **team needs** with mentoring available

Our downloadable catalogue includes:

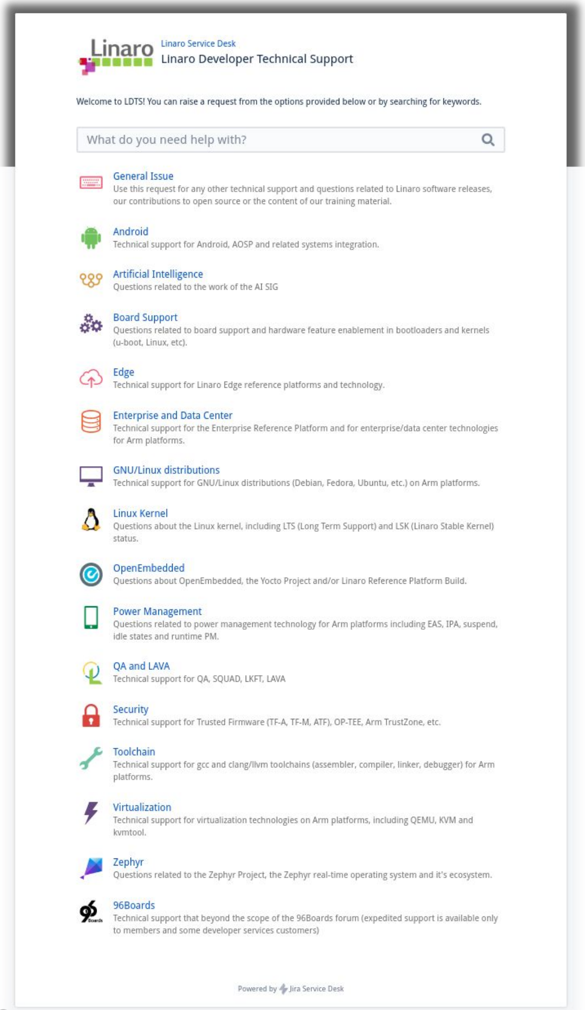
- Advanced kernel debugging techniques and tools
- Upstream Kernel Development
- Arm Trusted Firmware, Secure boot and PSCI
- Power Management - Energy Aware Scheduler (EAS) Introduction and tuning
- Introduction to OP-TEE
- Building Custom Systems with OE/Yocto
- Automatic Validation with LAVA

Linaro Developer Technical Support

LDTS means **developer to developer** technical support. Our **support team** is **staffed by** experienced **developers**... and is **reinforced by** the many domain **experts** employed by Linaro.

“The I3C subsystem does not currently implement the features you need to achieve this, largely because there are no drivers in the upstream kernel that would require anything like this. The most effective way to influence the evolution of this subsystem would be to share a driver that demonstrates the gaps (if that is possible for you). If you need advice on how to work wit

Ask us about: Android system integration, board support and enablement, embedded GNU/Linux distributions, Linux Kernel, Yocto/OpenEmbedded, testing using LAVA, TF-A, OP-TEE, GNU and clang compilers, Zephyr and more.



Linaro Service Desk
Linaro Developer Technical Support

Welcome to LDTS! You can raise a request from the options provided below or by searching for keywords.

What do you need help with?

General Issue
Use this request for any other technical support and questions related to Linaro software releases, our contributions to open source or the content of our training material.

Android
Technical support for Android, AOSP and related systems integration.

Artificial Intelligence
Questions related to the work of the AI SIG

Board Support
Questions related to board support and hardware feature enablement in bootloaders and kernels (u-boot, Linux, etc).

Edge
Technical support for Linaro Edge reference platforms and technology.

Enterprise and Data Center
Technical support for the Enterprise Reference Platform and for enterprise/data center technologies for Arm platforms.

GNU/Linux distributions
Technical support for GNU/Linux distributions (Debian, Fedora, Ubuntu, etc.) on Arm platforms.

Linux Kernel
Questions about the Linux kernel, including LTS (Long Term Support) and LSK (Linaro Stable Kernel) status.

OpenEmbedded
Questions about OpenEmbedded, the Yocto Project and/or Linaro Reference Platform Build.

Power Management
Questions related to power management technology for Arm platforms including EAS, IPA, suspend, idle states and runtime PM.

QA and LAVA
Technical support for QA, SQUAD, LKFT, LAVA

Security
Technical support for Trusted Firmware (TF-A, TF-M, ATF), OP-TEE, Arm TrustZone, etc.

Toolchain
Technical support for gcc and clang/llvm toolchains (assembler, compiler, linker, debugger) for Arm platforms.

Virtualization
Technical support for virtualization technologies on Arm platforms, including QEMU, KVM and kvmtool.

Zephyr
Questions related to the Zephyr Project, the Zephyr real-time operating system and it's ecosystem.

96Boards
Technical support that beyond the scope of the 96Boards forum (expedited support is available only to members and some developer services customers)

Powered by Jira Service Desk

How to get support

By e-mail

support@linaro.org

Any e-mail to this address will raise a support ticket. Accounts are created automatically based on the reporter's e-mail address.

Your developers must use their company e-mail addresses to raise tickets!

From a web browser

https://support.linaro.org

Developers can self-enroll using their company e-mail address.

Entitlements to the different levels of service are determined using this e-mail address.

With a search engine

"linaro support"

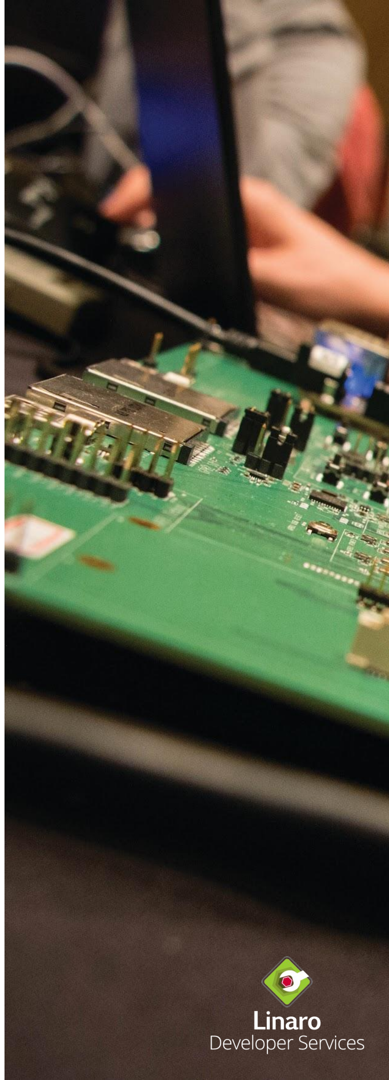
Busy developers sometimes forget things. The search term "linaro support" will bring up a web page containing reminders about what LDTS is and how to raise support tickets both by e-mail and from a web browser.

Linaro Developer Services

support@linaro.org

Introduction to TF-A

- Arm A-Profile Architecture evolution
  - ARMv7-A -> ARMv8-A
  - Armv8-A -> ARMv9-A
- About TF-A project
  - History and origins
- TF-A as EL3 firmware
  - Firmware components
- Handling Secure Monitor Call
- **Context management**
- Power State Coordination Interface



Context Management

The context management library in TF-A provides the **basic CPU context** initialization and management routines for use by different components in EL3 firmware.

The EL3 runtime firmware is responsible for **managing register context** during switch between Normal and Secure worlds. The register context to be saved and restored **depends on the mechanism** used to trigger the world switch.

Mechanisms to trigger the world switch:

- **Synchronous** mechanism via **SMC** exception.
- **Asynchronous** mechanism via **interrupt handling**.

Context Management - II

The context management library

- Will force to save and restore **general purpose registers**

- Provides helper functions for **system register** save and restore
    - Different scenarios will save and restore different context
    - If don’t switch to another world, then doesn’t save EL3’s context
    - If don’t turn off CPU, then doesn’t save GIC and arch timer’s context

- Context management for **secure payload dispatcher**
    - Standard dispatcher: Op-TEE dispatcher (OPTEED), Test secure payload dispatcher (TSPD), trusty, TLK dispatcher (tlkd) in firmware
    - Save and restore context during world switch
    - Route interrupt handling and execute context switch
    - If integrate own dispatcher, need manually save and restore context

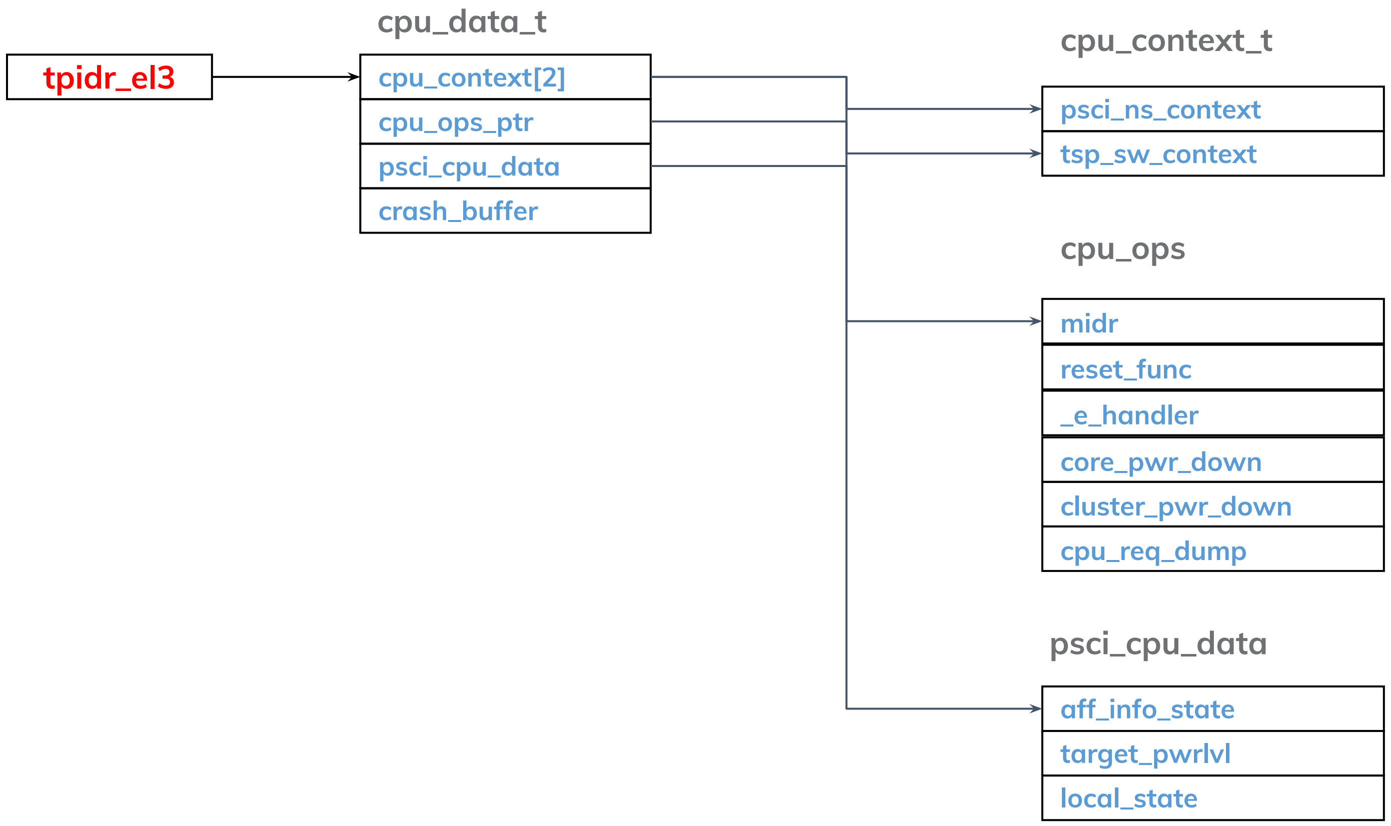
Aarch64 CPU registers

- Aarch64 execution state has:
  - Only one copy of general purpose registers
  - System registers (TTBR_ELx, VBAR_ELx, etc)
  - Permission: SMP bit ONLY can be accessed from EL3

- Per CPU registers context
  - General purpose registers (X0 ~ X30, LR, SP_ELx)
  - EL1/EL2/EL3's system control registers
  - VFP/NEON registers
  - CVE registers (optional)
  - Pointer authentication registers (optional)
  - GIC IF registers
  - Arch timer registers

- Some registers are banked
  - GIC, arch timer, etc

CPU Context



cpu_data_t
tpidr_el3
cpu_context[2]
cpu_ops_ptr
psci_cpu_data
crash_buffer
cpu_context_t
psci_ns_context
tsp_sw_context
cpu_ops
midr
reset_func
_e_handler
core_pwr_down
cluster_pwr_down
cpu_req_dump
psci_cpu_data
aff_info_state
target_pwrlvl
local_state

CPU Context

gp_regs_t

CTX_GPREG_X0 – X29
CTX_GPREG_LR
CTX_GPREG_SP_EL0

fp_regs_t

CTX_FP_Q0 – Q31
CTX_FP_FPSR
CTX_FP_FPCR

el3_state_t

CTX_SCR_EL3
CTX_ESR_EL3
CTX_RUNTIME_SP
CTX_SPSR_EL3
CTX_ELR_EL3
CTX_PMCR_EL0
CTX_IS_IN_EL3
CTX_CPTR_EL3
CTX_ZCR_EL3

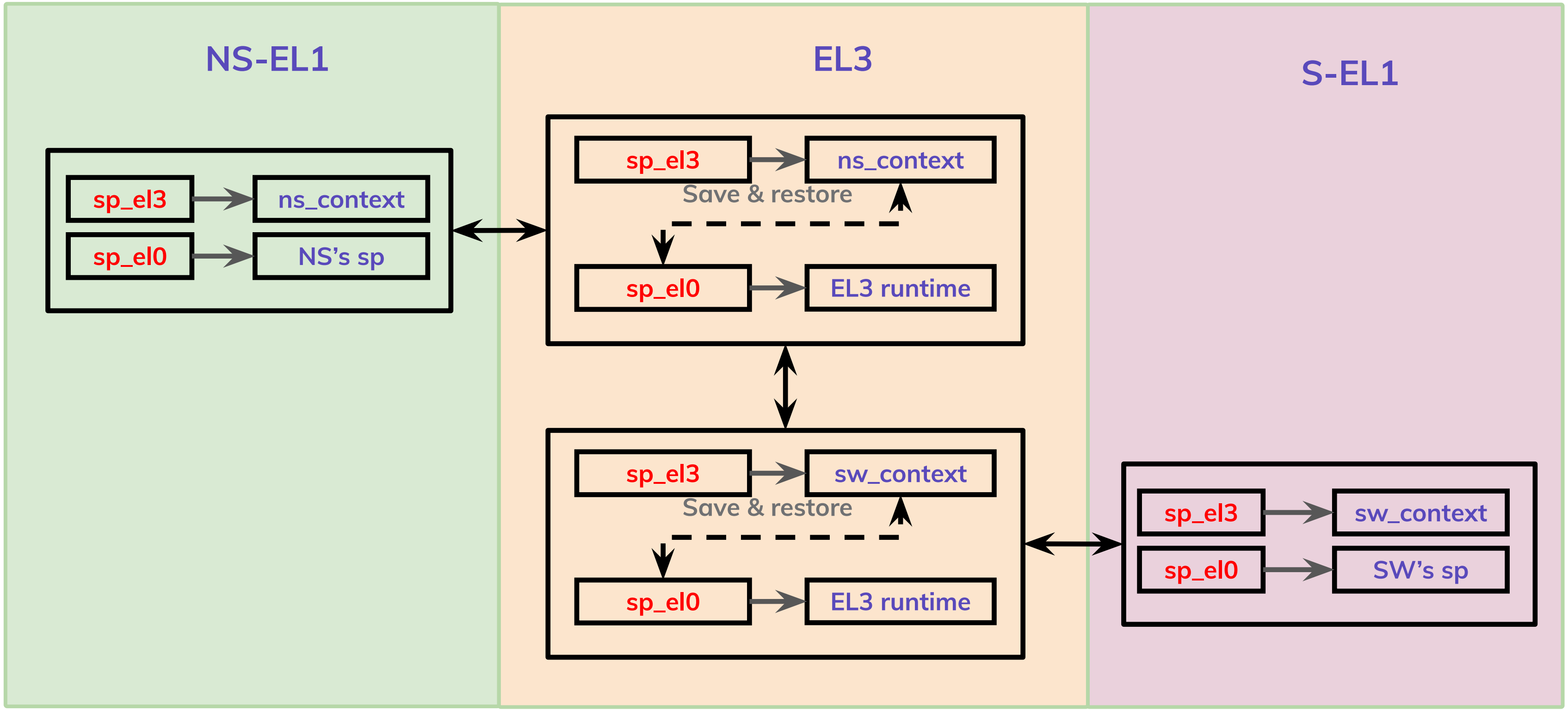
el2_sysregs_t

CTX_ACTLR_EL2	CTX_AFSR0_EL2	CTX_AFSR1_EL2	CTX_AMAIR_EL2	CTX_CNTHCTL_EL2	CTX_CNTVOFF_EL2
CTX_CPTR_EL2	CTX_DBGVCR32_EL2	CTX_ELR_EL2	CTX_ESR_EL2	CTX_FAR_EL2	CTX_HACR_EL2
CTX_HCR_EL2	CTX_HPFAR_EL2	CTX_HSTR_EL2	CTX_ICC_SRE_EL2	CTX_ICH_HCR_EL2	CTX_ICH_VMCR_EL2
CTX_MAIR_EL2	CTX_MDCR_EL2	CTX_PMSCR_EL2	CTX_SCTLR_EL2	CTX_SPSR_EL2	CTX_SP_EL2
CTX_TCR_EL2	CTX_TPIDR_EL2	CTX_TTBR0_EL2	CTX_VBAR_EL2	CTX_VMPIDR_EL2	CTX_VPIDR_EL2
CTX_VTCR_EL2	CTX_VTTBR_EL2	CTX_TFSR_EL2	CTX_MPAM2_EL2	CTX_MPAMHCR_EL2	CTX_MPAMVPM[0..7]_EL2
CTX_MPAMVPMV_EL2	CTX_HAFGRTR_EL2	CTX_HDFGRTR_EL2	CTX_HDFGWTR_EL2	CTX_HFGITR_EL2	CTX_HFGRTR_EL2
CTX_HFGWTR_EL2	CTX_CNTPOFF_EL2	CTX_CONTEXTIDR_EL2	CTX_SDER32_EL2	CTX_TTBR1_EL2	CTX_VDISR_EL2
CTX_VNCR_EL2	CTX_VSESR_EL2	CTX_VSTCR_EL2	CTX_VSTTBR_EL2	CTX_TRFCR_EL2	

el1_sysregs_t

CTX_SPSR_EL1	CTX_ELR_EL1	CTX_SCTLR_EL1	CTX_TCR_EL1	CTX_CPACR_EL1
CTX_CSSELR_EL1	CTX_SP_EL1	CTX_ESR_EL1	CTX_TTBR0_EL1	CTX_TTBR1_EL1
CTX_MAIR_EL1	CTX_AMAIR_EL1	CTX_ACTLR_EL1	CTX_TPIDR_EL1	CTX_TPIDR_EL0
CTX_TPIDRRO_EL0	CTX_PAR_EL1	CTX_FAR_EL1	CTX_AFSR0_EL1	CTX_AFSR1_EL1
CTX_CONTEXTIDR_EL1	CTX_VBAR_EL1	CTX_CNTP_CTL_EL0	CTX_CNTP_CVAL_EL0	CTX_CNTV_CTL_EL0
CTX_CNTV_CVAL_EL0	CTX_CNTKCTL_EL1	CTX_TFSRE0_EL1	CTX_TFSR_EL1	CTX_RGSR_EL1
CTX_GCR_EL1				

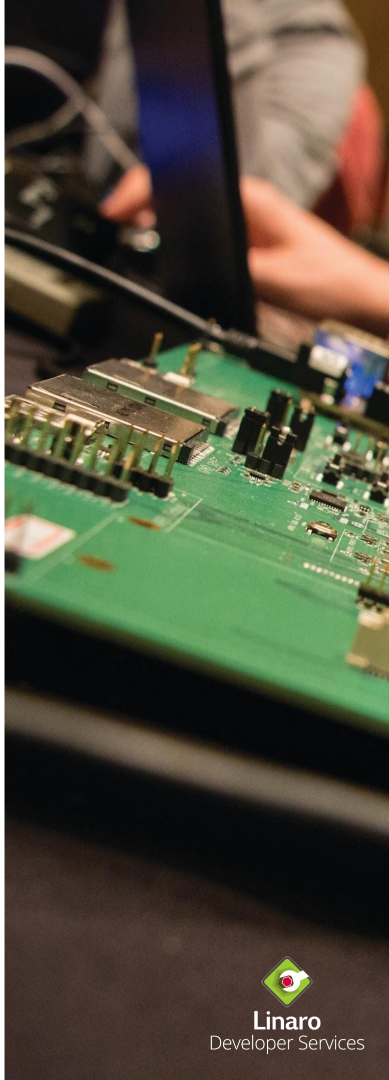
Stack pointer (SP) refers context



Linaro
Developer Services

Introduction to TF-A

- Arm A-Profile Architecture evolution
  - ARMv7-A -> ARMv8-A
  - Armv8-A -> ARMv9-A
- About TF-A project
  - History and origins
- TF-A as EL3 firmware
  - Firmware components
- Handling Secure Monitor Call
- Context management
- **Power State Coordination Interface**



Power State Coordination Interface

- Power State Coordination Interface (PSCI)
- Defines a standard interface for making power management requests across exception levels and operating systems
- Allows secure firmware to arbitrate power management requests from secure and non-secure software
- Default method for power control in Linux AArch64 kernel

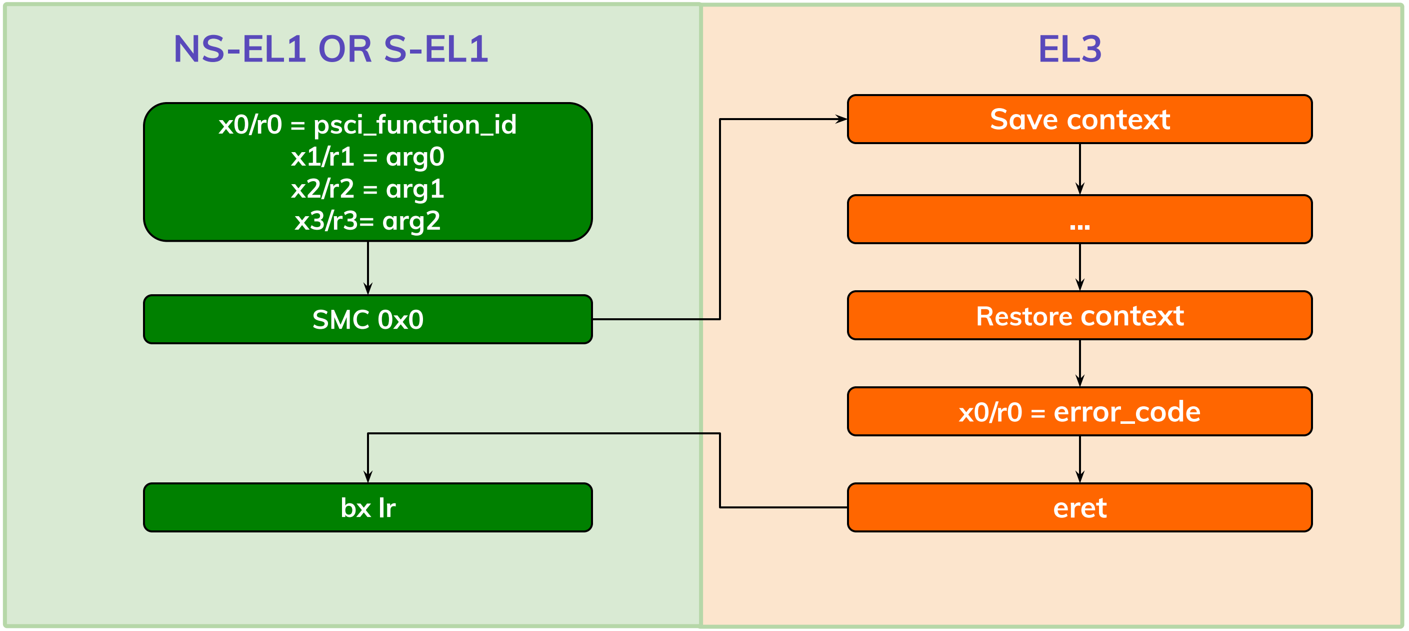
https://developer.arm.com/documentation/den0022/c/

Before, when there was no PSCI...

- The state machine for power management is not easily expressed
- Misunderstanding between silicon and software architects
  - Sometimes silicon design engineer don't understand well software implementation, vice versa
  - State machine mismatch between hardware and software
  - Bad thing: surprises after tapeout
- Surprising changes with product families
  - Total different solution within SoC series, why?
  - It's hard for upstreaming, such like workaround, have no clean code layout, etc
- ARM64 maintainers likely to resist integrating alternatives to PSCI

Linaro
Developer Services

SMC Calls For PSCI



Linaro
Developer Services

PSCI CPU context management

PSCI library is in charge of initializing/restoring the **non-secure CPU system registers** according to PSCI specification during cold/warm boot. Registers that need to be preserved across **CPU power down/power up** cycles are maintained in **cpu_context_t** data structure.

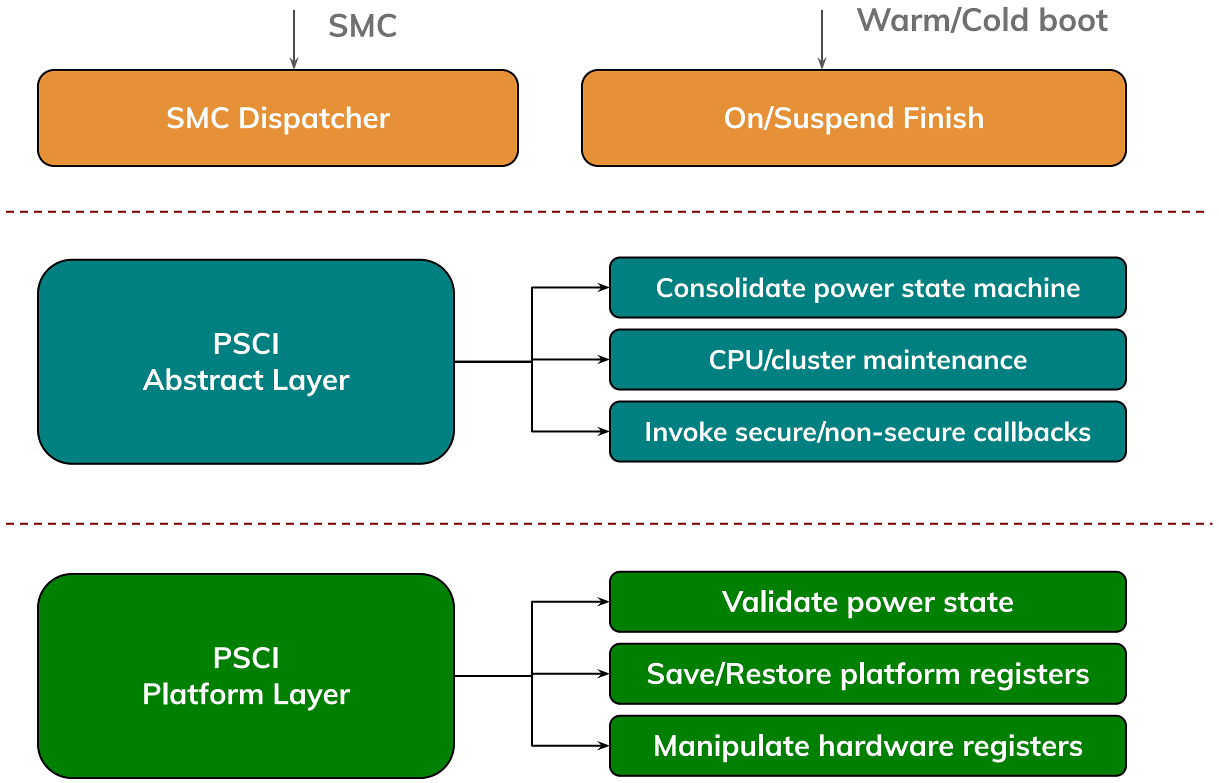
PSCI library internally uses CPU Context management API:

- **cm_set_context_by_index()**
- **cm_get_context()**
- **cm_get_context_by_index()**

PSCI context management can be split into following categories:

- **PSCI core layer** will help save system registers
- **Platform code** need save and restore GIC and arch timer context

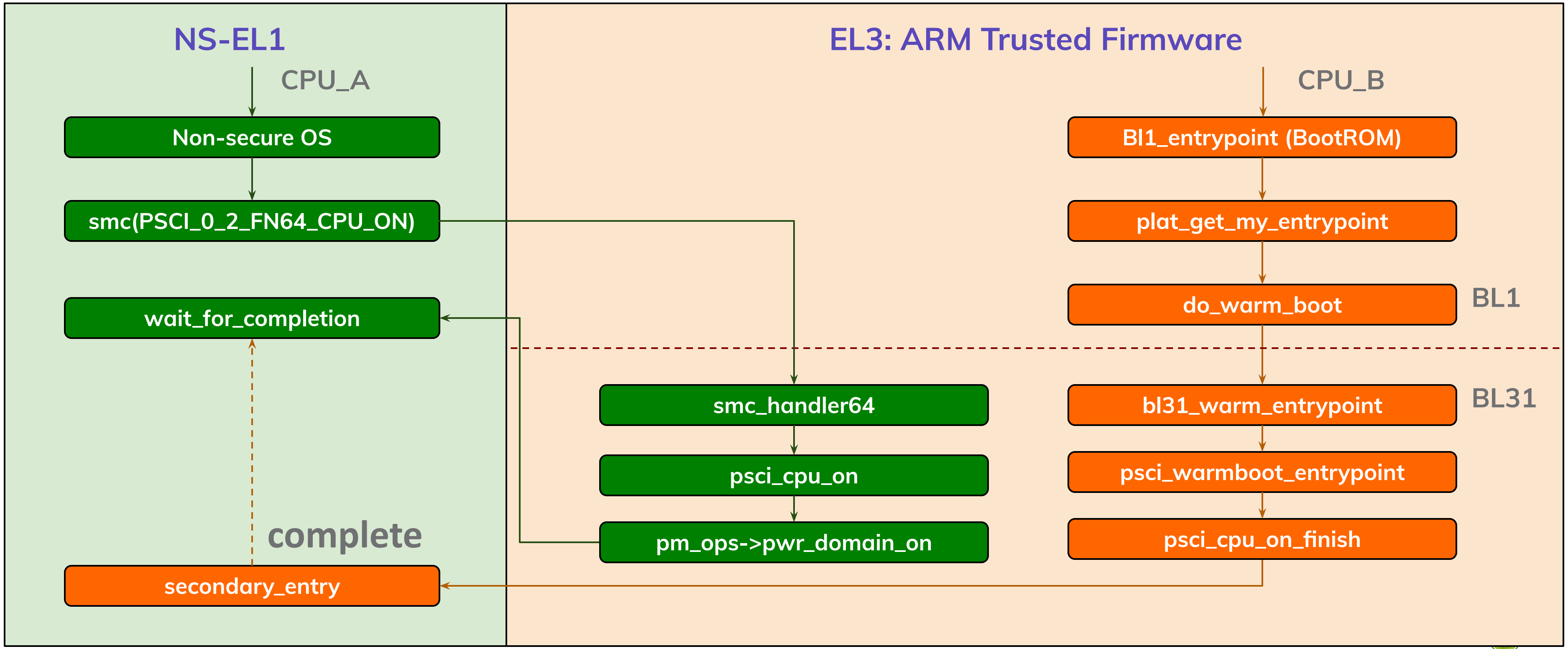
PSCI Core Layer



PSCI Definition - 1

No	Function	Description	X0/R0	X1/R1	X2/R2	X3/R3	Result	PSCI 0.2	PSCI 1.0
1	**PSCI_VERSION**	**Return the version of PSCI implemented**	0x8400 0000	N/A	N/A	N/A	Bits [31:16] Major Version Bits [15:0] Minor Version	**Mandatory**	**Mandatory**
2	**CPU_SUSPEND**	**Suspend execution on a core**	ARM32: 0x8400 0001 ARM64: 0xC400 0001	power_state	entry_Point	context_id	SUCCESS INVALID_PARAMETERS INVALID_Address DENIED	**Mandatory**	**Mandatory**
3	**CPU_OFF**	**Hotplug the calling core**	0x8400 0002	N/A	N/A	N/A	No return DENIED	**Mandatory**	**Mandatory**
4	**CPU_ON**	**Power up a core**	ARM32: 0x8400 0003 ARM64: 0xC400 0003	target_cpu	entry_point	context_id	SUCCESS INVALID_PARAMETERS ALREADY_ON ON_PENDING INTERNAL_FAILURE	**Mandatory**	**Mandatory**
5	**AFFINITY_INFO**	**Enables the caller to request status of an affinity level**	ARM32: 0x8400 0004 ARM64: 0xC400 0004	Target_affinity	lowest_affinity_level	N/A	2 ON_PENDING 1 OFF 0 ON INVALID_PARAMETERS NOT_PRESENT DISABLED	**Mandatory**	**Mandatory**

Aside: CPU Power On Main Flow



PSCI Definition - 2

No	Function	Description	X0/R0	X1/R1	X2/R2	X3/R3	Result	PSCI 0.2	PSCI 1.0
6	**MIGRATE**	Migrate its context to a specific core	ARM32: 0x8400 0005 ARM64: 0xC400 0005	Target_cpu	N/A	N/A	SUCCESS NOT_SUPPORTED INVALID_PARAMETERS DENIED INTERNAL_FAILURE NOT_PRESENT	Optional	Optional
7	**MIGRATE_INFO_TYPE**	Identify the level of multicore support present in the Trusted OS	0x8400 0006	N/A	N/A	context_id	0 1 2 NOT_SUPPORTED	Optional	Optional
8	**MIGRATE_INFO_UP_CPU**	Returns the current resident core	ARM32: 0x8400 0007 ARM64: 0xC400 0007	N/A	N/A	N/A	UNDEFINED MPIDR based value	Optional	Optional
9	**SYSTEM_OFF**	Shutdown the system	0x8400 0008	N/A	N/A	N/A	N/A	**Mandatory**	**Mandatory**
10	**SYSTEM_RESET**	Reset the system	0x8400 0009	N/A	N/A	N/A	N/A	**Mandatory**	**Mandatory**

PSCI Definition - 3

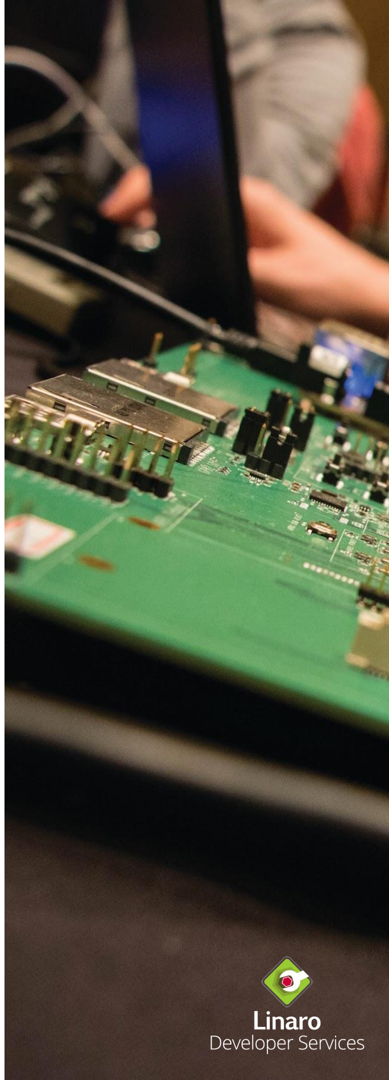
No	Function	Description	X0/R0	X1/R1	X2/R2	X3/R3	Result	PSCI 0.2	PSCI 1.0
11	**PSCI_FEATURES**	**Discover whether a specific function is implemented**	0x8400 000A	Function Id	N/A	N/A	NOT_SUPPORTED Feature Flag	Not Applicable	**Mandatory**
12	**CPU_FREEZE**	**In an IMPLEMENTATION DEFINED low-power state, Still valid for interrupts**	0x8400 000B	N/A	N/A	N/A	NOT_SUPPORTED DENIED	Not Applicable	Optional
13	**CPU_DEFAULT_SUSPEND**	**Unlike CPU_SUSPEND the caller need not specify a power_state parameter**	ARM32: 0x8400 000C ARM64: 0xC400 000C	Entry_point	context_id	N/A	SUCCESS INVALID_ADDRESS	Not Applicable	Optional
14	**NODE_HW_STATE**	**return the true HW state of a node**	ARM32: 0x8400 000D ARM64: 0xC400 000D	target_cpu	target_level	N/A	2 HW_STANDBY 1 HW_OFF 0 HW_ON NOT_SUPPORTED INVALID_PARAMETERS	Not Applicable	Optional
15	**SYSTEM_SUSPEND**	**Implement suspend to RAM**	ARM32: 0x8400 000E ARM64: 0xC400 000E	Entry_point	context_id	N/A	NOT_SUPPORTED INVALID_ADDRESS ALREADY_ON	Not Applicable	Optional

Linaro
Developer Services

PSCI Definition - 4

No	Function	Description	X0/R0	X1/R1	X2/R2	X3/R3	Result	PSCI 0.2	PSCI 1.0
16	**PSCI_SET_ SUSPEND_MODE**	**Setting the mode used by CPU_SUSPEND to coordinate power states.**	0x8400 000F	Function Id	N/A	N/A	SUCCESS NOT_SUPPORTED INVALID_PARAMETERS DENIED	Not Applicable	Optional
17	**PSCI_STAT_RESIDENCY**	**Returns the amount of time the platform has spent in the given power state since cold boot**	ARM32: 0x8400 0010 ARM64: 0xC400 0010	target_cpu	power_state	N/A	residency time	Not Applicable	Optional
18	**PSCI_STAT_COUNT**	**Return the number of times the platform has used the given power state since cold boot.**	ARM32: 0x8400 0011 ARM64: 0xC400 0011	target_cpu	power_state	N/A	count	Not Applicable	Optional

Lab sessions



Linaro
Developer Services

Preparation: Install host packages

For Debian/Ubuntu:

```
$ sudo apt install build-essential bc bison flex git libelf-dev libncurses-dev libssl-dev
```

For Fedora and RHEL/CentOS/RockyLinux:

```
$ sudo dnf install gcc git flex make bison \
                   elfutils-libelf-devel ncurses-devel openssl-devel
```

(or perhaps `sudo` *`yum`* `install gcc …`)

Preparation: Downloadable content

The assets that we will use to provide a development environment that can be used for the lab exercises can be downloaded from here:

https://fileserver.linaro.org/s/fE6iBYca8bYqrrk

You will need to download:

- `poky-glibc-x86_64-core-image-tfa-cortexa57-qemuarm64-secureboot-toolchain-4.0.10.sh`
- `flash.bin.xz`
- `core-image-tfa-qemuarm64-secureboot.wic.qcow2.xz`

Preparation: Install the development tools

```
$ sh poky-glibc-x86_64-core-image-tfa-cortexa57-qemuarm64-secureboot-toolchain-4.0.10.sh
Poky (Yocto Project Reference Distro) SDK installer version 4.0.10
==================================================================
Enter target directory for SDK (default: /opt/poky/4.0.10): ~/build/ts/tfa/sdk/
You are about to install the SDK to "/home/sumit/build/ts/tfa/sdk". Proceed [Y/n]? Y
Extracting SDK............................................................done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the
environment setup script e.g.
 $ . /home/sumit/build/ts/tfa/sdk/environment-setup-cortexa57-poky-linux

$ . /home/sumit/build/ts/tfa/sdk/environment-setup-cortexa57-poky-linux
$ aarch64-poky-linux-gcc --version
aarch64-poky-linux-gcc (GCC) 11.3.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
$ qemu-system-aarch64 --version
QEMU emulator version 6.2.0
Copyright (c) 2003-2021 Fabrice Bellard and the QEMU Project developers
```

Preparation: Unpack and boot kernel and rootfs

- Decompress

```
unxz -T0 flash.bin.xz
unxz -T0 core-image-tfa-qemuarm64-secureboot.wic.qcow2.xz
```

- Boot

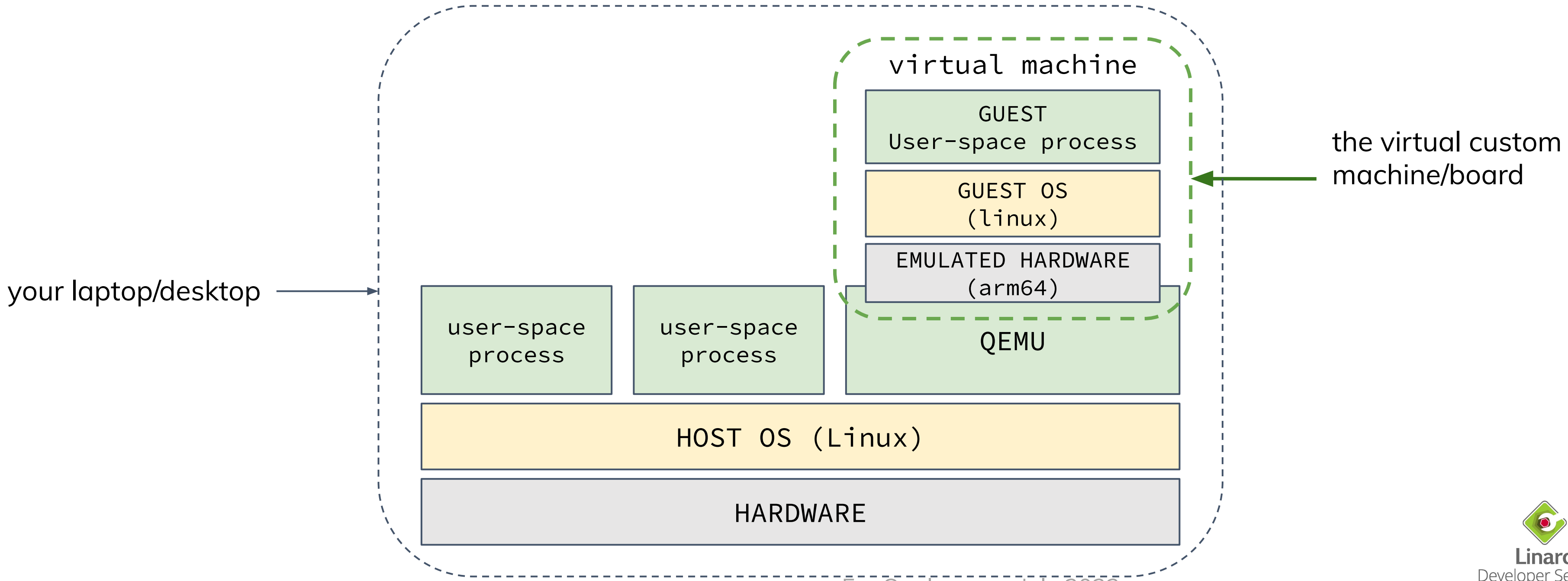
```
qemu-system-aarch64 \
    -machine virt,secure=on -cpu cortex-a57 \
    -smp 4 -nographic -m 1G -bios flash.bin \
    -drive \
     file=./core-image-tfa-qemuarm64-secureboot.wic.qcow2,if=virtio,format=qcow2 \
    -netdev user,id=eth0,hostfwd=tcp::2222-:22 \
    -device virtio-net-device,netdev=eth0
```

Don’t forget to source the SDK environment setup script!

- Explore
  - root user has no password set
  - Use `poweroff` command to shutdown cleanly or press Ctrl-A x to force immediate off

Aside: QEMU

Qemu is a free and open source emulator providing hardware virtualization. It will allow us to work with an arm64 machine by emulating the processor(s) and its peripherals.

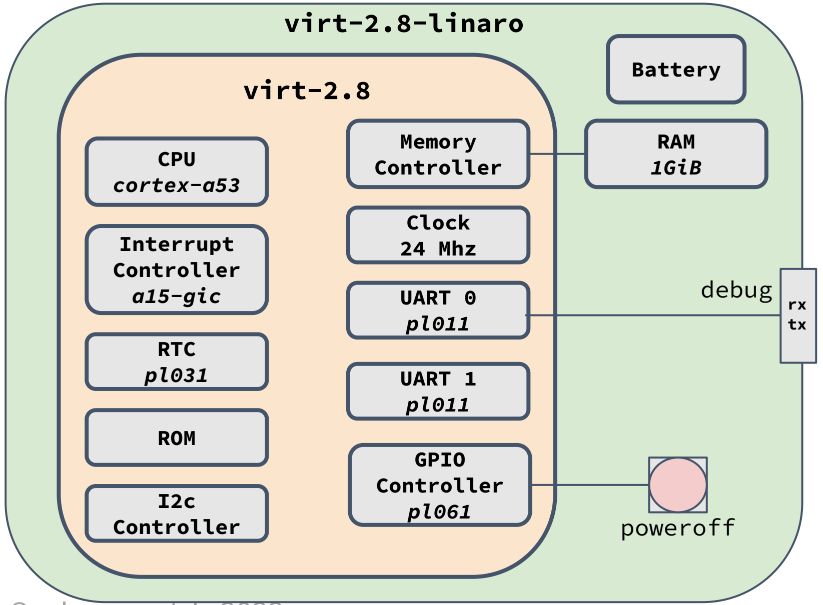


Aside: “Hardware” information

- We decide to create a custom board for an embedded IoT project.
- This board is **built around** the “**virt,secure”** virtual SoC. The simplified SoC devices memory mapping is given below:

Device	Address	Size
VIRT_ROM	0x00000000	0x08000000
VIRT_GIC	0x08000000	0x00010000
VIRT_UART (UART 0)	0x09000000	0x00001000
VIRT_RTC	0x09010000	0x00001000
VIRT_I2C	0x0901A000	0x00001000
VIRT_FW_CFG	0x09020000	0x00000018
VIRT_GPIO	0x09030000	0x00001000
VIRT_SECURE_UART (UART 1)	0x09040000	0x00001000
VIRT_MEM	0x40000000	0x08000000

- Our custom board ‘**virt-2.8-linaro**’ is populated with **virt-2.8 SoC**, **1 GiB of RAM**, a connector routed to UART for **debug console** and a **power off button**.



Aside: Understanding the virtual network

```
qemu-system-aarch64 \
    -machine virt,secure=on -cpu cortex-a57 \
    -smp 4 -nographic -m 1G -bios flash.bin \
    -drive \
     file=./core-image-tfa-qemuarm64-secureboot.wic.qcow2,if=virtio,format=qcow2 \
    -netdev user,id=eth0,hostfwd=tcp::2222-:22 \
    -device virtio-net-device,netdev=eth0
```

Emulate a private network (in **user**space) as part of the qemu process. The emulated network includes the host workstation (10.0.2.2), the VM (10.0.2.15) and a DNS server (10.0.2.3).

The emulated network uses **NAT routing** to communicate with other devices. **hostfwd allows** us to configure **port forwarding**, in this case routing port 2222 on your host workstation to port 22 on the VM (port 22 is secure shell).

Preparation: Test the networking and NAT routing

- Boot the VM
  - Use `ifconfig` to check the networking is active (the VM is pre-configured to bring up the network interface and launch a DHCP request)
- Open a new Terminal emulator window on your workstation and try:
  - Connecting to the target interactively (lower case -p)
    ```
    ssh -p 2222 root@localhost
    ```
  - Copying files from host to target (upper case -P)
    ```
    scp -P 2222 my_file root@localhost:
    ```
  - Copying files from target to host (upper case -P)
    ```
    scp -P 2222  root@localhost:/proc/config.gz .
    ```
- From the target try:
  - Pinging your workstation
    ```
    ping 10.0.2.2
    ```
  - Test NAT routing (if you are behind a company firewall replace https://linaro.org with something served internally)
    ```
    wget https://linaro.org
    ```

LAB1 - Build and boot your own TF-A

Build TF-A from source code

```
$ . /poky/sdk/path/environment-setup-cortexa57-poky-linux

$ git clone https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git
$ cd trusted-firmware-a
$ git checkout -b v2.9.0 v2.9.0

$ LDFLAGS= make CFLAGS= PLAT=qemu DEBUG=1 \
      BL33=$SDKTARGETSYSROOT/boot/u-boot.bin BL32_RAM_LOCATION=tdram \
      all fip
$ dd if=build/qemu/debug/bl1.bin of=flash-src.bin bs=4096 conv=notrunc
$ dd if=build/qemu/debug/fip.bin of=flash-src.bin seek=64 bs=4096 conv=notrunc
```

Linaro
Developer Services

Update with “flash-src.bin” and boot

Launch Qemu with cmdline:

```
qemu-system-aarch64 \
    -machine virt,secure=on -cpu cortex-a57 \
    -smp 4 -nographic -m 1G -bios trusted-firmware-a/flash-src.bin \
    -drive \
     file=./core-image-tfa-qemuarm64-secureboot.wic.qcow2,if=virtio,format=qcow2 \
    -netdev user,id=eth0,hostfwd=tcp::2222-:22 \
    -device virtio-net-device,netdev=eth0
```

Modify TF-A source and rebuild

Modify TF-A source code and see if you can confirm if its the one that you booted?

```
diff --git a/bl31/bl31_main.c b/bl31/bl31_main.c
index e70eb5584..86a1e4f01 100644
--- a/bl31/bl31_main.c
+++ b/bl31/bl31_main.c
@@ -112,7 +112,7 @@ void bl31_setup(u_register_t arg0, u_register_t arg1, u_register_t arg2,
  ******************************************************************************/
 void bl31_main(void)
 {
-       NOTICE("BL31: %s\n", version_string);
+       NOTICE("<Your name> built this BL31: %s\n", version_string);
        NOTICE("BL31: %s\n", build_message);

 #if FEATURE_DETECTION
```

Linaro
Developer Services

LAB2 - Exploring PSCI flow

Attach GDB during TF-A boot

Launch Qemu with cmdline:

```
qemu-system-aarch64 \
    -machine virt,secure=on -cpu cortex-a57 \
    -smp 2 -nographic -m 1G -bios trusted-firmware-a/flash-src.bin \
    -drive \
     file=./core-image-tfa-qemuarm64-secureboot.wic.qcow2,if=virtio,format=qcow2 \
    -netdev user,id=eth0,hostfwd=tcp::2222-:22 \
    -device virtio-net-device,netdev=eth0 \
    -gdb tcp::1234 -S
```

“**-gdb**” option: Accept gdb connection over “**tcp**” port no. “**1234**”. QEMU defaults to starting the guest without waiting for gdb to connect; use “**-S**” too if you want it to not start execution.

“**-S**” option: Freeze CPU at startup (use 'c' to start execution).

Attach debugger during TF-A boot

Launch GDB in another terminal:

```
$ . /poky/sdk/path/environment-setup-cortexa57-poky-linux
$ cd trusted-firmware-a
$ aarch64-poky-linux-gdb
```

GDB setup commands:

```
(gdb) target remote localhost:1234
(gdb) add-symbol-file build/qemu/debug/bl31/bl31.elf
```

Exploring PSCI_CPU_ON flow

```
(gdb) hbreak psci_cpu_on_start
Hardware assisted breakpoint 1 at 0xe044584: file lib/psci/psci_on.c, line 64.
(gdb) c
Continuing.

Thread 1 hit Breakpoint 1, psci_cpu_on_start (target_cpu=target_cpu@entry=1, ep=ep@entry=0xe04c038) at lib/psci/psci_on.c:64
64            int ret = plat_core_pos_by_mpidr(target_cpu);
(gdb) set print pretty
(gdb) p /x *ep
$1 = {
  h = {
    type = 0x1,
    version = 0x1,
    size = 0x58,
    attr = 0x1
  },
  pc = 0x40f8a254,
  spsr = 0x3c5,
…
```

Set hardware breakpoint

Primary CPU calls PSCI to power on secondary CPU

This is the non-secure entry point

Exploring PSCI_CPU_ON_FINISH flow

```
(gdb) hbreak *0x40f8a254
Hardware assisted breakpoint 2 at 0x40f8a254
(gdb) hbreak bl31_warm_entrypoint
Hardware assisted breakpoint 3 at 0xe040114: file bl31/aarch64/bl31_entrypoint.S, line 166.
(gdb) c
Continuing.
[Switching to Thread 1.2]

Thread 2 hit Breakpoint 3, bl31_warm_entrypoint () at bl31/aarch64/bl31_entrypoint.S:166
166          el3_entrypoint_common                                \
(gdb) hbreak psci_cpu_on_finish
Hardware assisted breakpoint 4 at 0xe04448c: file lib/psci/psci_on.c, line 177.
(gdb) c
Continuing.

Thread 2 hit Breakpoint 4, psci_cpu_on_finish (cpu_idx=1, state_info=0xe04d0b0) at lib/psci/psci_on.c:177
177          psci_plat_pm_ops->pwr_domain_on_finish(state_info);
(gdb) c
Continuing.

Thread 2 hit Breakpoint 2, 0x0000000040f8a254 in ?? ()
```

Set hardware breakpoint for non-secure and secure entry point

Set hardware breakpoint for PSCI CPU ON finish entry

CPU hit the non-secure entry point meaning that it has jumped to kernel

Thank you

support@linaro.org

Linaro
Developer Services