RMM-EL3 Interface

Javier Almansa Sobrino
23 Jun 2022
Agenda

• Requirements
• Boot process
• Error handling and return values
• Versioning
• Boot Manifest details
• RMM <-> EL3 Runtime calls
• Shared buffer management during EL3 <-> RMM calls
• RMM <-> EL3 World switch register convention
• Runtime Services
• Implementation details
Requirements

- RMM needs to be as platform independent as possible
  - Support for PIE
  - Needs to be able to receive configuration parameters at boot time.
    - RMM has minimal platform specific differentiation at build time
  - RMM does not depend on stages prior to BL31 and its configuration is dependent of the configuration mechanism used for the EL3 firmware
    - This would allow partners to use their own BL2/BL1 image and any configuration mechanism (DT, FCONF, hardcoded parameters). BL31 would parse and extract the relevant info for RMM.
  - RMM should be EL3 firmware agnostic
    - When possible, we should make no assumption about the underlying EL3 software.
    - A contract between EL3 and RMM is needed to allow the former to pass platform information to the latter, such as
      - Number of CPUs
      - Address range for peripherals (e.g. UART)
      - Shared memory buffer (more on this later)
Requirements

- The information will be passed from EL3 to RMM via a register contract between both parties or via a Boot Manifest (or both).
- RMM may require services from EL3 FW (e.g., to get attestation keys or to delegate or undelegate memory granules)
  - A formal spec of the services is required.
- The spec defines the switch register convention between RMM and EL3, part of the SMCCC contract, when NS world is the client.
- Manage compatibility and migration between EL3 and RMM as the interface evolves to cater for future requirements.
Agenda

• Requirements
• Boot process
• Error handling and return values
• Versioning
• Boot Manifest details
• RMM <-> EL3 Runtime calls
• Shared buffer management during EL3 <-> RMM calls
• RMM <-> EL3 World switch register convention
• Runtime Services
• Implementation details
Boot process

The boot process is initiated by the Root world (EL3 Firmware)
## Boot process

RMM accepts up to four arguments, stored in registers x0 to x3. A proposal for the v0.1 Boot Interface (argument usage) is:

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>Linear index for this PE. This index starts from 0 and must be less than PLATFORM_CORE_COUNT</td>
</tr>
<tr>
<td>x1</td>
<td>RMM - EL3 Interface Version (0.1 for this spec)</td>
</tr>
<tr>
<td>x2</td>
<td>PLATFORM_CORE_COUNT</td>
</tr>
<tr>
<td>x3</td>
<td>Base PA for the shared buffer used for communication between EL3 and RMM.</td>
</tr>
<tr>
<td>x4 - x7</td>
<td>RES0</td>
</tr>
</tbody>
</table>

### COLD BOOT

### WARM BOOT

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>Linear index for this PE. This index starts from 0 and must be less than PLATFORM_CORE_COUNT</td>
</tr>
<tr>
<td>x1 - x7</td>
<td>RES0</td>
</tr>
</tbody>
</table>
Agenda

• Requirements
• Boot process
• Error handling and return values
• Versioning
• Boot Manifest details
• RMM <-> EL3 Runtime calls
• Shared buffer management during EL3 <-> RMM calls
• RMM <-> EL3 World switch register convention
• Runtime Services
• Implementation details
Error handling and return values

• After cold/warm boot up and initialization, RMM returns control back to RMMD through `SMC_RMM_BOOT_COMPLETE SMC` call
  • This call only accepts one argument, an error code in x1

• Upon error, whether it happens during cold or warm boot, RMM will abort the boot process and it will be made unavailable to all the CPUs as to present a symmetric view to the entire system.
# Error handling and return values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_RMM_BOOT_SUCCESS</td>
<td>Boot successful</td>
<td>0</td>
</tr>
<tr>
<td>E_RMM_BOOT_ERR_UNKNOWN</td>
<td>Unknown error</td>
<td>-1</td>
</tr>
<tr>
<td>E_RMM_IFC_VERSION_INVALID</td>
<td>Boot interface version reported by RMMD is not supported by RMM</td>
<td>-2</td>
</tr>
<tr>
<td>E_RMM_BOOT_CPUS_OUT_OF_RANGE</td>
<td>RMMD reported a maximum number of CPUs larger than the maximum supported by RMM</td>
<td>-3</td>
</tr>
<tr>
<td>E_RMM_BOOT_CPU_ID_OUT_OF_RANGE</td>
<td>Current CPU ID is higher than the maximum reported by RMMD</td>
<td>-4</td>
</tr>
<tr>
<td>E_RMM_BOOT_INVALID_SHARED_BUFFER</td>
<td>Invalid pointer to shared buffer area</td>
<td>-5</td>
</tr>
<tr>
<td>E_RMM_BOOT_MANIFEST_VERSION_NOT_SUPPORTED</td>
<td>Version reported by the boot manifest not supported by RMM</td>
<td>-6</td>
</tr>
<tr>
<td>E_RMM_BOOT_MANIFEST_DATA_ERROR</td>
<td>Error parsing the core boot manifest</td>
<td>-7</td>
</tr>
</tbody>
</table>
Agenda

• Requirements
• Boot process
• Error handling and return values
• Versioning
  • Boot Manifest details
  • RMM <-> EL3 Runtime calls
  • Shared buffer management during EL3 <-> RMM calls
  • RMM <-> EL3 World switch register convention
• Runtime Services
• Implementation details
Interface (and manifest) versioning

- The EL3 – RMM interface is versioned to ease compatibility between versions.
- Version number (passed through x1) is 32 bits-wide with bit 31 set as RES0
  - `VERSION_MAJOR` (Bits [16:30])
    - This value is increased iff the changes to the Boot Interface ABI break compatibility with previous versions.
  - `VERSION_MINOR` (Bits [0:15])
    - This value is increased iff the changes to the Boot Interface ABI do not break backwards compatibility with previous versions or
    - It is reset to 0 upon `VERSION_MAJOR` update.
- `RES0` field
  - For consistency with other modules' versioning on RMM.
Agenda

• Requirements
• Boot process
• Error handling and return values
• Versioning
• Boot Manifest details
• RMM <-> EL3 Runtime calls
• Shared buffer management during EL3 <-> RMM calls
• RMM <-> EL3 World switch register convention
• Runtime Services
• Implementation details
Boot Manifest details

- Common to all platforms.
- First field corresponds to the version number (mandatory).
- Can grow up to 1 page size.
- Must be allocated inside the RMM – EL3 Shared buffer.

- It caters for per-platform data.
  - The platform part of the manifest must fall inside the shared area and not overlap with the core manifest.
    - Macros will be provided on TF-A to get a valid pointer for the platform manifest data.
- The offset of each component on the manifest is enforced by the spec.

```
/* Boot manifest core structure as per v0.1 */
You_lastmonth1 author(You)
typdef struct rmm_manifest {
    uint32_t version;  /* Manifest version */
    uintptr_t plat_data; /* Manifest platform data */
} rmm_manifest_t;

CASSERT((offsetof(rmm_manifest_t, version) == 0, 
    rmm_manifest_t_version_unaligned));
CASSERT((offsetof(rmm_manifest_t, plat_data) == 8, 
    rmm_manifest_t_plat_data_unaligned));
```
Agenda

• Requirements
• Boot process
• Error handling and return values
• Versioning
• Boot Manifest details
• RMM <-> EL3 Runtime calls
• Shared buffer management during EL3 <-> RMM calls
• RMM <-> EL3 World switch register convention
• Runtime Services
• Implementation details
RMM <-> EL3 Runtime calls – RMM initiated
RMM <-> EL3 Runtime calls – NS Initiated

• The specification defines the implementation defined register switch convention between RMM and EL3 when NS world is the client.
• EL3 FW is seen as a service provider.
• Steps 3 & 4 are optional.
Agenda

• Requirements
• Boot process
• Error handling and return values
• Versioning
• Boot Manifest details
• RMM <-> EL3 Runtime calls
• Shared buffer management during EL3 <-> RMM calls
• RMM <-> EL3 World switch register convention
• Runtime Services
• Implementation details
Shared buffer management during EL3 <-> RMM interaction

- The shared buffer is meant to be used during EL3 <-> RMM communications, to pass large data structures.
  - Platform tokens or keys
- RMM sees EL3 as a service provider. Only RMM can initiate communications via SMC.
- When the shared buffer is needed in a communication (regardless of the data direction), RMM is responsible of locking and own the buffer to avoid concurrent accesses or other race conditions.
- EL3 assumes that the PE making the service call has exclusive access to the shared buffer.
- It is RMM responsibility to unlock and free the shared buffer upon request termination.
Shared buffer management during EL3 <-> RMM comms.

The SMC can pass a pointer to a PA to ELC as long as it belongs to the shared buffer.
Agenda

• Requirements
• Boot process
• Error handling and return values
• Versioning
• Boot Manifest details
• RMM <-> EL3 Runtime calls
• Shared buffer management during EL3 <-> RMM calls
• RMM <-> EL3 World switch register convention
• Runtime Services
• Implementation details
RMM <-> EL3 world switch register convention

- EL3 is expected to maintain a register context specific to each world and it will save and restore the register appropriately.

- EL3 must maintain a separate register context for
  - GPRs (x0 – x30) as well as sp_el0 and sp_el2 stack pointers.
  - EL2 system register context for all enabled features by EL3, including registers with _EL2 prefix.
  - EL2 physical and virtual timer registers must not be included in the register context.

- EL3 will not save some registers as mentioned below. It is responsibility of RMM to save them if the Realm World makes use of them.
  - FP/SIMD registers
  - SVE registers
  - SME registers
  - EL1/0 registers
Agenda

- Requirements
- Boot process
- Error handling and return values
- Versioning
- Boot Manifest details
- RMM <-> EL3 Runtime calls
- Shared buffer management during EL3 <-> RMM calls
- RMM <-> EL3 World switch register convention
- Runtime Services
- Implementation details
Runtime services – SMC_RMMMD_GTSI_DELEGATE

0xC4001B0

- Request EL3 to delegate a memory granule

<table>
<thead>
<tr>
<th>Name</th>
<th>Register</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FID</td>
<td>x0</td>
<td>[63:0]</td>
<td>Uint64</td>
<td>Command FID</td>
</tr>
<tr>
<td>PA</td>
<td>x1</td>
<td>[63:0]</td>
<td>Address</td>
<td>Physical Base Address of the granule to delegate</td>
</tr>
</tbody>
</table>

**Input Values:**

**Output Values:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Register</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>x0</td>
<td>[63:0]</td>
<td>Error Code</td>
<td>Command return status</td>
</tr>
</tbody>
</table>
Runtime services – SMC_RMMD_GTSI_UNDELEGATE

0xC4001B1

- Request EL3 to undelegate a memory granule

<table>
<thead>
<tr>
<th>Name</th>
<th>Register</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FID</td>
<td>x0</td>
<td>[63:0]</td>
<td>Uint64</td>
<td>Command FID</td>
</tr>
<tr>
<td>PA</td>
<td>x1</td>
<td>[63:0]</td>
<td>Address</td>
<td>Physical Base Address of the granule to undelegate</td>
</tr>
</tbody>
</table>

Input Values:

Output Values:

<table>
<thead>
<tr>
<th>Name</th>
<th>Register</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>x0</td>
<td>[63:0]</td>
<td>Error Code</td>
<td>Command return status</td>
</tr>
</tbody>
</table>
Runtime services – SMC_RMM_GET_REALM_ATTEST_KEY
0xC4001B2

• Retrieve the Realm Attestation Key from EL3

Input Values:

<table>
<thead>
<tr>
<th>Name</th>
<th>Register</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FID</td>
<td>x0</td>
<td>[63:0]</td>
<td>Uint64</td>
<td>Command FID</td>
</tr>
<tr>
<td>PA</td>
<td>x1</td>
<td>[63:0]</td>
<td>Address</td>
<td>PA where to store the Realm Attestation Key. The PA must belong to the shared buffer</td>
</tr>
<tr>
<td>BSize</td>
<td>x2</td>
<td>[63:0]</td>
<td>Size</td>
<td>Size in bytes of the Realm Attestation Key Buffer</td>
</tr>
<tr>
<td>Curve</td>
<td>x3</td>
<td>[63:0]</td>
<td>Enum</td>
<td>Type of the elliptic curve to which the requested attestation key belongs to</td>
</tr>
</tbody>
</table>

Output Values:

<table>
<thead>
<tr>
<th>Name</th>
<th>Register</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>x0</td>
<td>[63:0]</td>
<td>Error Code</td>
<td>Command return status</td>
</tr>
<tr>
<td>PTSize</td>
<td>x1</td>
<td>[63:0]</td>
<td>Size</td>
<td>Size of the Realm Attestation Key</td>
</tr>
</tbody>
</table>
## Runtime services – SMC_RMM_GET_REALM_TOKEN

0xC4001B3

- Retrieve the platform token from EL3

### Input Values:

<table>
<thead>
<tr>
<th>Name</th>
<th>Register</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FID</td>
<td>x0</td>
<td>[63:0]</td>
<td>Uint64</td>
<td>Command FID</td>
</tr>
<tr>
<td>PA</td>
<td>x1</td>
<td>[63:0]</td>
<td>Address</td>
<td>PA of the platform attestation token. The challenge object is passed in this buffer. The PA must belong to the shared buffer</td>
</tr>
<tr>
<td>BSize</td>
<td>x2</td>
<td>[63:0]</td>
<td>Size</td>
<td>Size in bytes of the platform attestation token buffer</td>
</tr>
<tr>
<td>CSize</td>
<td>x3</td>
<td>[63:0]</td>
<td>Size</td>
<td>Size in bytes of the challenge object. It corresponds to the size of one of the defined SHA algorithms</td>
</tr>
</tbody>
</table>

### Output Values:

<table>
<thead>
<tr>
<th>Name</th>
<th>Register</th>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>x0</td>
<td>[63:0]</td>
<td>Error Code</td>
<td>Command return status</td>
</tr>
<tr>
<td>PTSize</td>
<td>x1</td>
<td>[63:0]</td>
<td>Size</td>
<td>Size of the platform token</td>
</tr>
</tbody>
</table>
Agenda

• Requirements
• Boot process
• Error handling and return values
• Versioning
• Boot Manifest details
• RMM <-> EL3 Runtime calls
• Shared buffer management during EL3 <-> RMM calls
• RMM <-> EL3 World switch register convention
• Runtime Services
• Implementation details
Implementation details – TF-A Boot process

**EL3/Root World**
- `std_svc`
- `RMM`
- `context_mgmt.c`
- `bl31_main`

**Legend**
- RMM Setup
- RMM Cold Boot Entry

**Realm World**
- RMM

Pre-requisite: EL3 firmware has allocated and mapped a memory area to share with RMM. EL3 has mapped this area into REALM space.

Generate boot manifest into the shared area

Setup boot parameters for RMM in x0 to x3

Before finishing the boot process, RMM can request Runtime Services from EL3 firmware.

Read boot manifest and perform any critically early initialization

HW Initialization (e.g. UART, MMU)

Continue with the rest of the initialization

Return control back to EL3 (context switch)
Implementation details

- The shared buffer area is a single page of statically allocated memory. It can be used by any CPU.
- BL2 maps the REALM area to load the RMM image.
- BL31 (Root) maps the shared buffer area with the Realm PAS attributes.
- The Realm and shared buffer areas are mapped as a single GPT block with same attributes (PAS_REALM) as both areas can only be accessed by the Realm world (and from Root).
- The shared buffer area is available for the whole lifecycle of the system.
• `plat_rmmdd_load_manifest()` must be implemented by the platform provider.

• It receives a pointer to a manifest, which it can populate the boot parameters.

• The platform is responsible for defining a platform manifest data structure and populate it if necessary.
Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה