TF-RMM Stage 1 Memory Management

TF-A Tech Forum

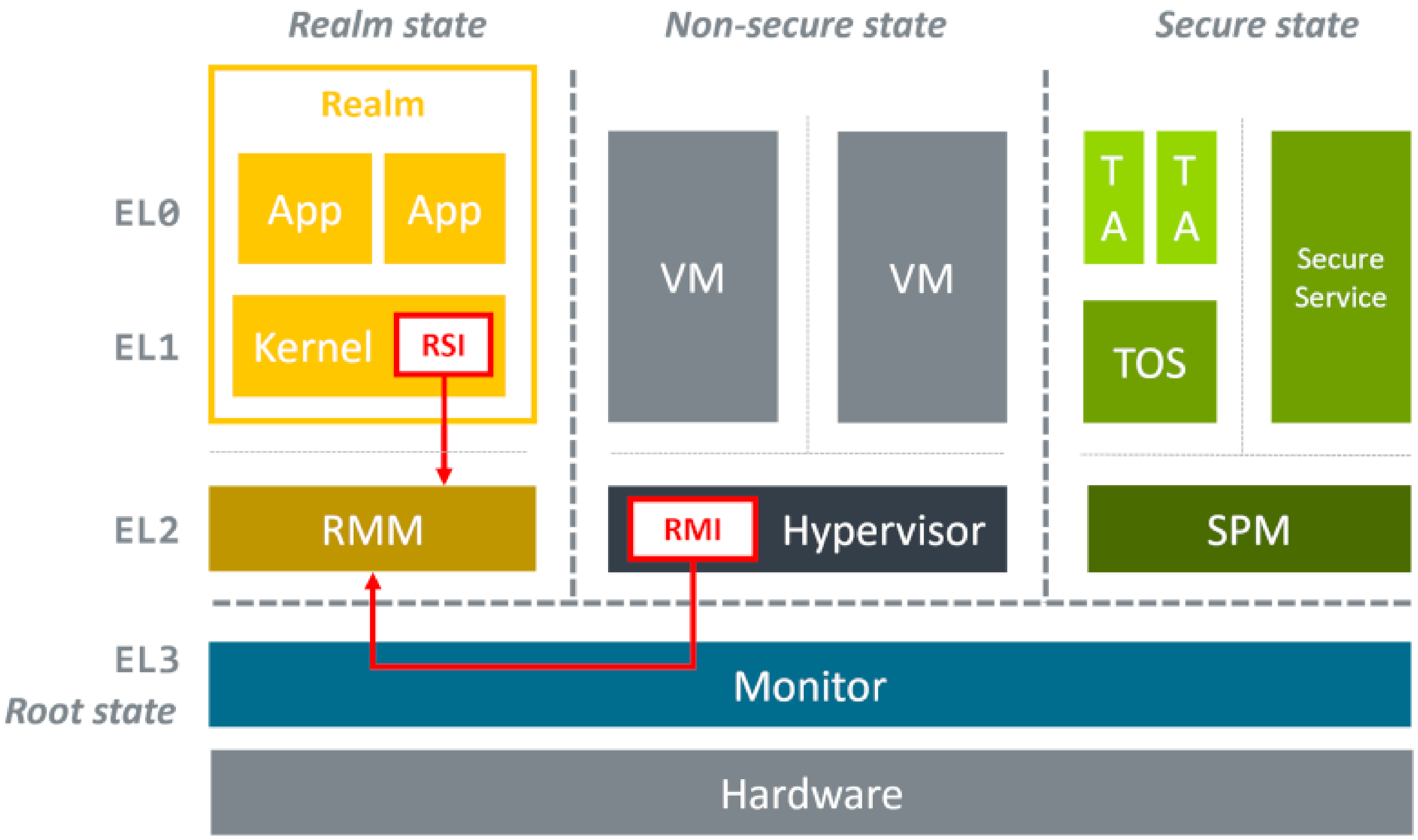
Javier Almansa Sobrino

April 2024

Agenda

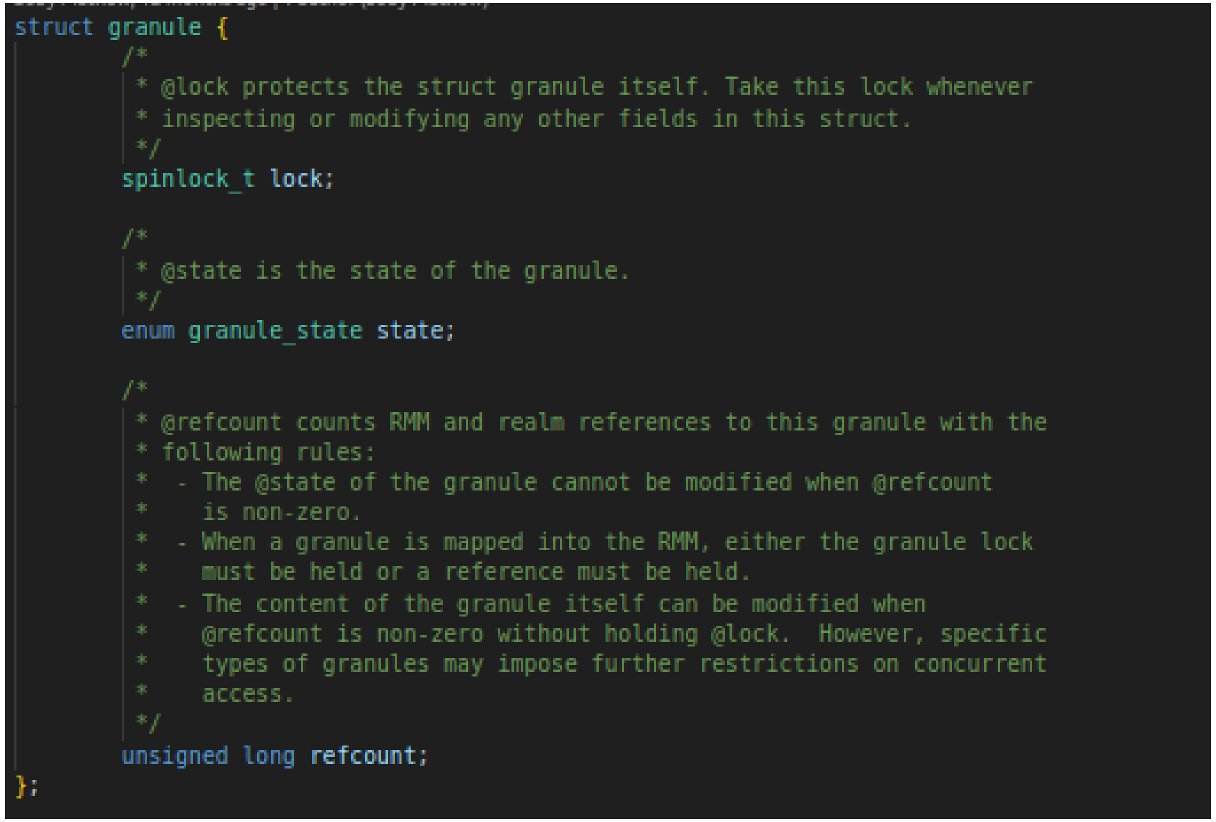
- The physical address space
- Granule state tracking
- Stage 1 translation regime
  - Low VA range
  - High VA range
    - Slot buffers
    - Per-CPU stacks
- The Stage 1 Translation library
- Unittests
- Future work

The Physical Address Space



Granule State Tracking

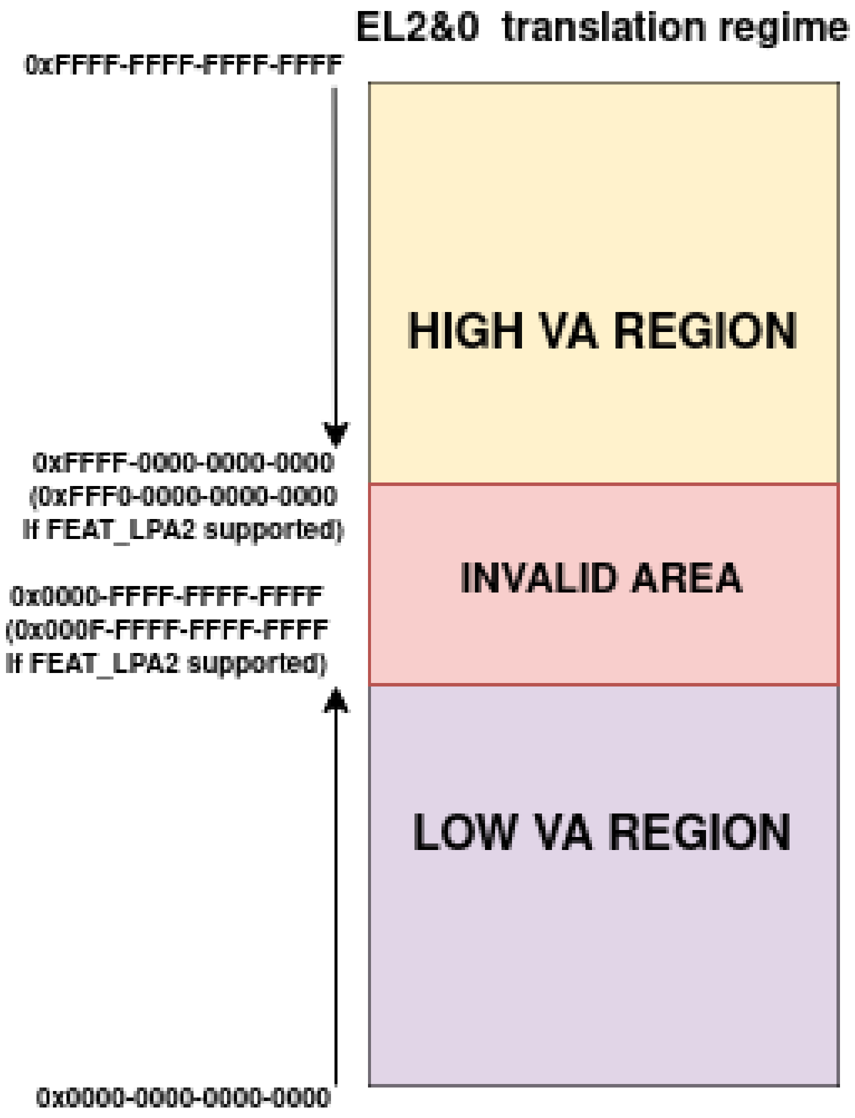
- RMM needs to keep track of all the delegable (Non-Secure PAS) memory available at boot time
- An array of granule structures keep track of the state of the memory.
  - One entry per granule (page) of available memory.
- The state of a granule might be a pre-condition for some RMI SMCs. Likewise, they can undergo transitions as part of the RMI SMCs.



```
struct granule {
        /*
         * @lock protects the struct granule itself. Take this lock whenever
         * inspecting or modifying any other fields in this struct.
         */
        spinlock_t lock;

        /*
         * @state is the state of the granule.
         */
        enum granule_state state;

        /*
         * @refcount counts RMM and realm references to this granule with the
         * following rules:
         *  - The @state of the granule cannot be modified when @refcount
         *    is non-zero.
         *  - When a granule is mapped into the RMM, either the granule lock
         *    must be held or a reference must be held.
         *  - The content of the granule itself can be modified when
         *    @refcount is non-zero without holding @lock.  However, specific
         *    types of granules may impose further restrictions on concurrent
         *    access.
         */
        unsigned long refcount;
};
```

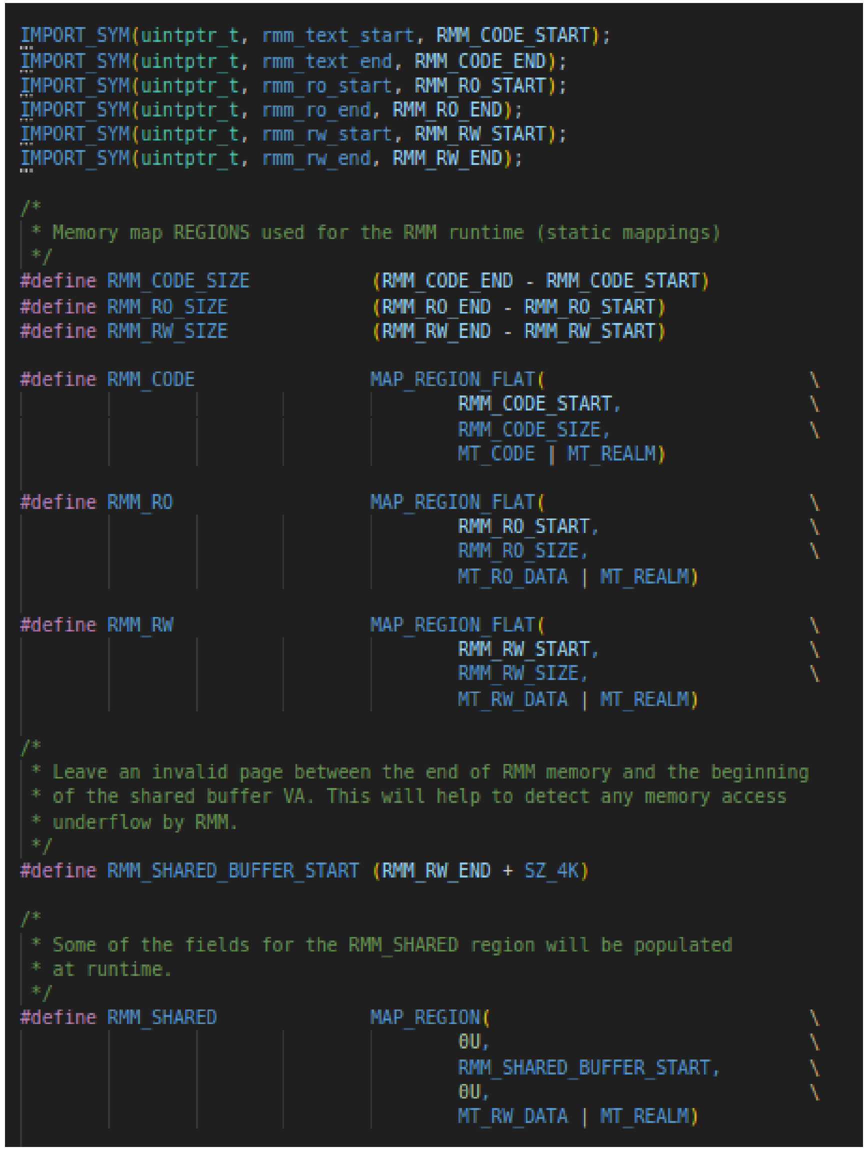
Stage 1 Translation Regime

- RMM Uses FEAT_VHE
  - Splits the 64Bit VA into two different address spaces.
  - Low VA Region
    - Per-CPU tables.
  - High VA Region
    - Shared tables.
- TCR_EL2.TxSZ fields control the maximum VA size of each region
  - VA size in bytes = $2^{64\text{-TCR\_EL2.TxSZ}}$
  - Number of address bits = 64 – TCR_EL2.TxSZ



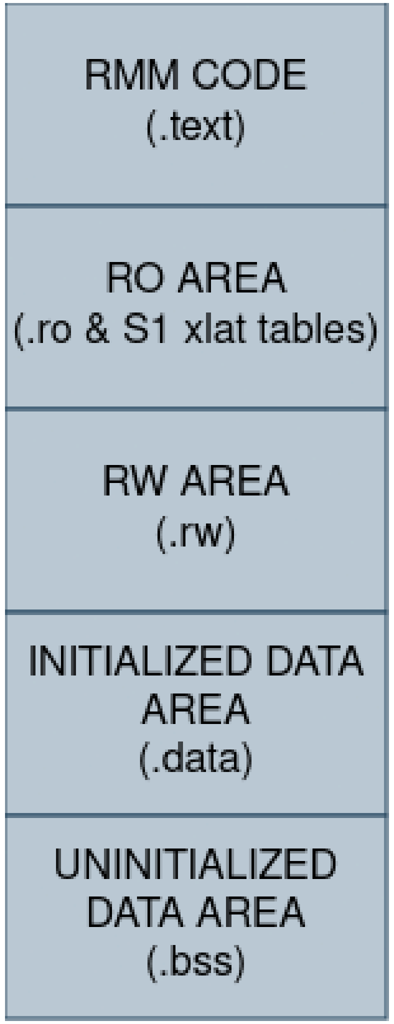
Stage 1 Low VA space

- Shared across all CPUs
- Static (and mostly flat) mappings
  - Symbols from the linker are imported in order to create flat mappings.
  - Other mappings such as the EL3 shared region or per-platform mappings might not be flat.
  - Translation tables are stored into .ro section.
- RMM is compiled as PIE binary
  - GOT and other relocations are fixed by the startup code before the MMU is enabled.



```
IMPORT_SYM(uintptr_t, rmm_text_start, RMM_CODE_START);
IMPORT_SYM(uintptr_t, rmm_text_end, RMM_CODE_END);
IMPORT_SYM(uintptr_t, rmm_ro_start, RMM_RO_START);
IMPORT_SYM(uintptr_t, rmm_ro_end, RMM_RO_END);
IMPORT_SYM(uintptr_t, rmm_rw_start, RMM_RW_START);
IMPORT_SYM(uintptr_t, rmm_rw_end, RMM_RW_END);

/*
 * Memory map REGIONS used for the RMM runtime (static mappings)
 */
#define RMM_CODE_SIZE           (RMM_CODE_END - RMM_CODE_START)
#define RMM_RO_SIZE             (RMM_RO_END - RMM_RO_START)
#define RMM_RW_SIZE             (RMM_RW_END - RMM_RW_START)

#define RMM_CODE                MAP_REGION_FLAT(                \
                                        RMM_CODE_START,         \
                                        RMM_CODE_SIZE,          \
                                        MT_CODE | MT_REALM)

#define RMM_RO                  MAP_REGION_FLAT(                \
                                        RMM_RO_START,           \
                                        RMM_RO_SIZE,            \
                                        MT_RO_DATA | MT_REALM)

#define RMM_RW                  MAP_REGION_FLAT(                \
                                        RMM_RW_START,           \
                                        RMM_RW_SIZE,            \
                                        MT_RW_DATA | MT_REALM)

/*
 * Leave an invalid page between the end of RMM memory and the beginning
 * of the shared buffer VA. This will help to detect any memory access
 * underflow by RMM.
 */
#define RMM_SHARED_BUFFER_START (RMM_RW_END + SZ_4K)

/*
 * Some of the fields for the RMM_SHARED region will be populated
 * at runtime.
 */
#define RMM_SHARED              MAP_REGION(                     \
                                        0U,                     \
                                        RMM_SHARED_BUFFER_START, \
                                        0U,                     \
                                        MT_RW_DATA | MT_REALM)
```

plat_common_init.c



RMM CODE
(.text)

RO AREA
(.ro & S1 xlat tables)

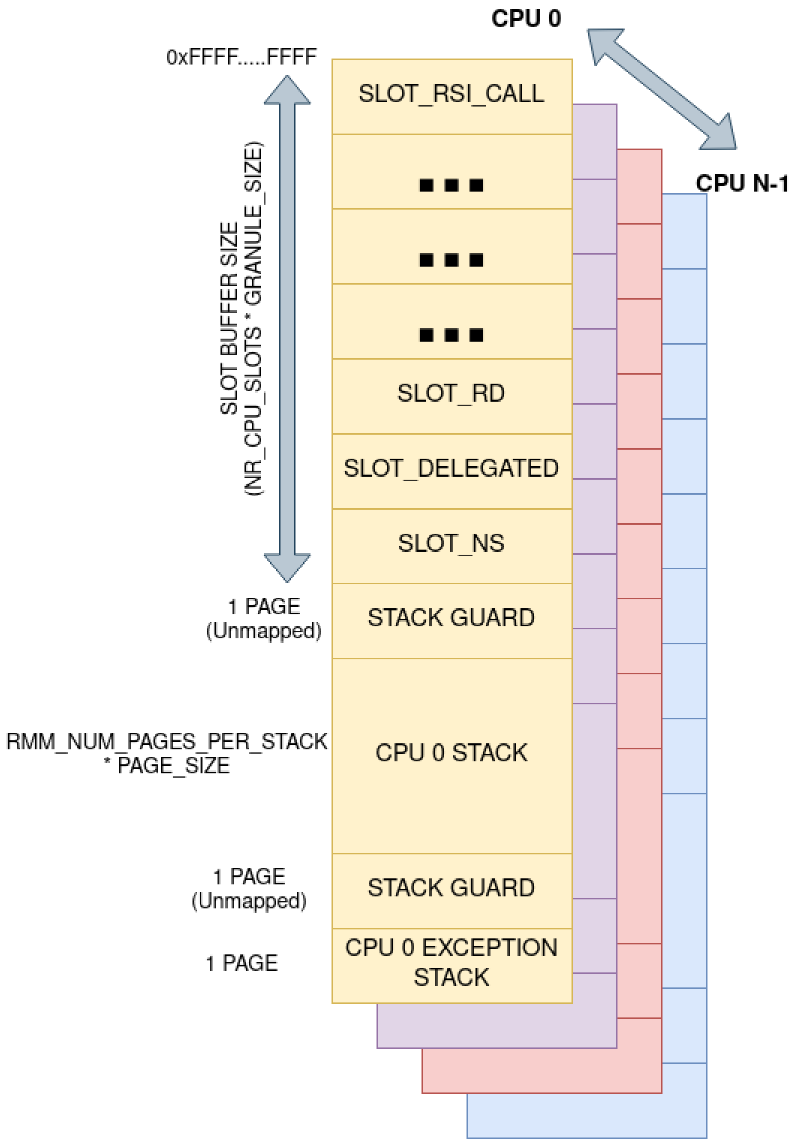
RW AREA
(.rw)

INITIALIZED DATA AREA
(.data)

UNINITIALIZED DATA AREA
(.bss)

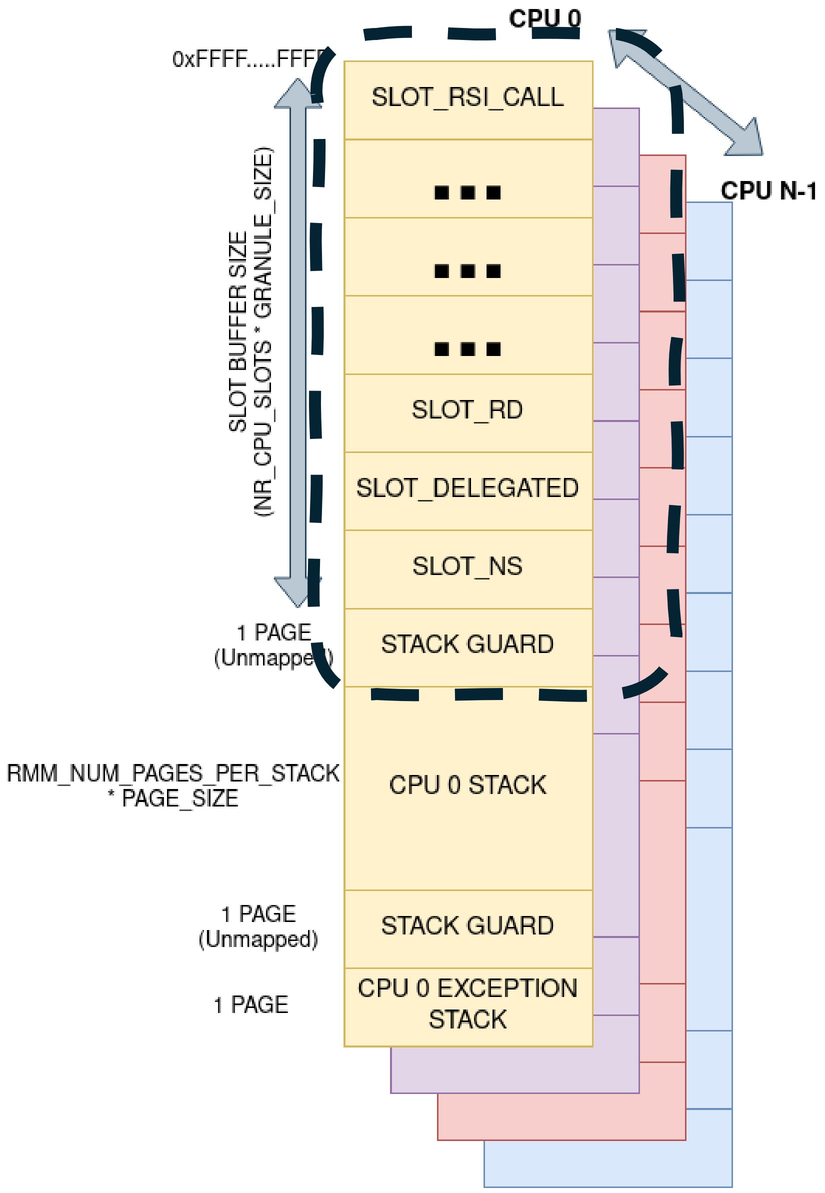
Stage 1 High VA space

- Per-CPU set of translation tables
- Contains mappings for the slot buffers, mapped a fixed VAs.
  - Any CPU can map/unmap any granule on any slot buffer.
- Contains mappings for the per-CPU stacks
- This space is managed by *xlat_high_va.{c, h}*



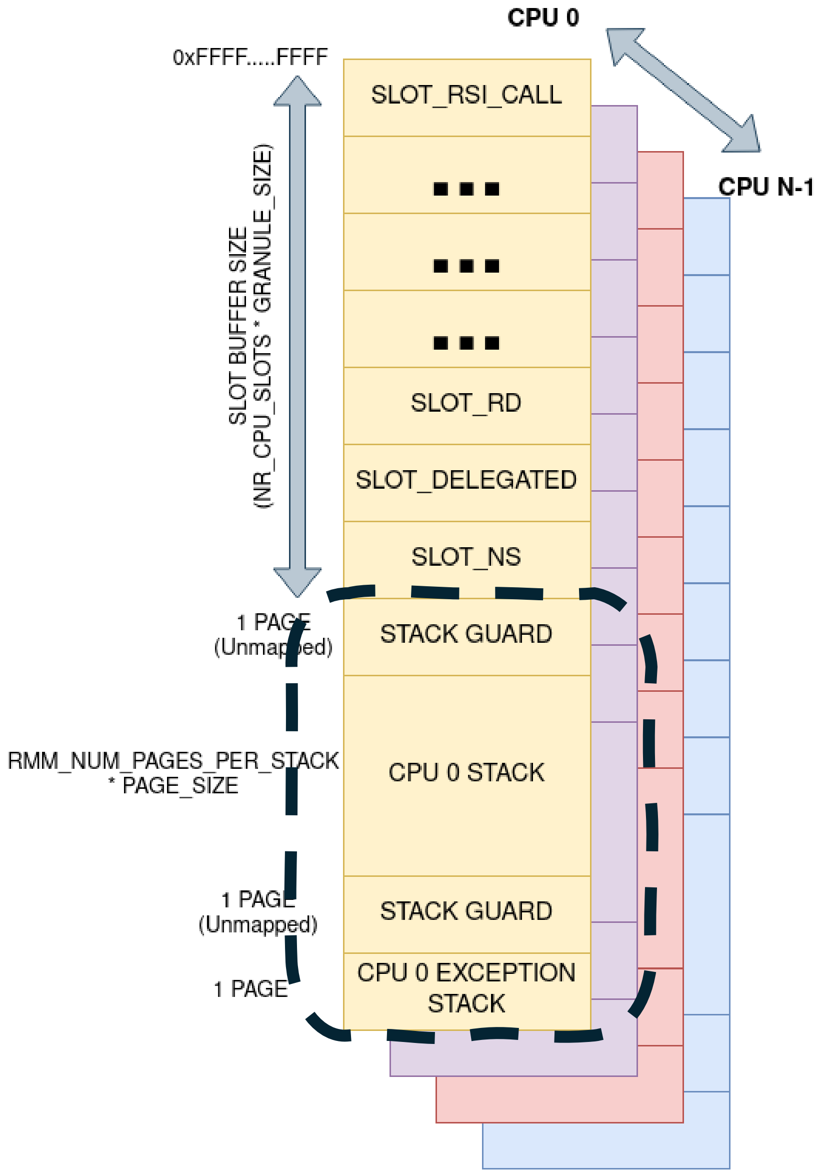
Stage 1 High VA space – Slot buffers

- Fixed number of slots per CPU
- Each slot is used to map a granule in a particular state
  - RMM uses the *granule_state* to ensure that granules are mapped to the right slot
  - *enum buffer_slot* in *buffer.h*
- Each CPU has its own set of translation tables
  - Same type of slot has same VA across all the CPUs
  - Ease the migration of vCPUs
- The Slot Buffer component includes optimizations to increase map/unmap performace.



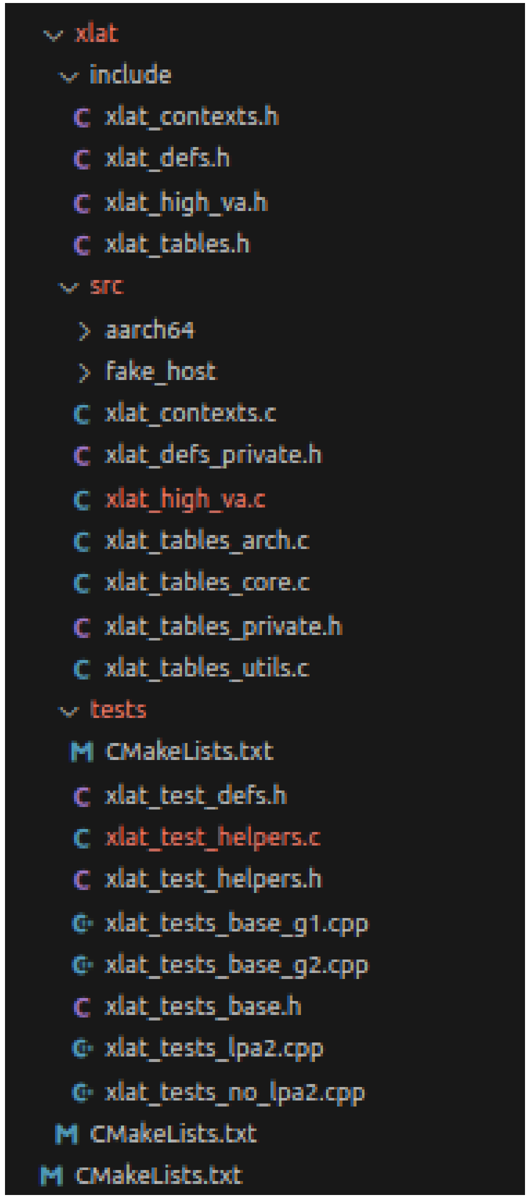
Stage 1 High VA space – Per CPU Stack

- Stack size configurable at build time
  - RMM_NUM_PAGES_PER_STACK
- The stack start for each CPU is calculated at boot time and the mapping updated
- An unmapped page guard protects against stack underflows.
- There is a special stack used to handle stack overflow faults.



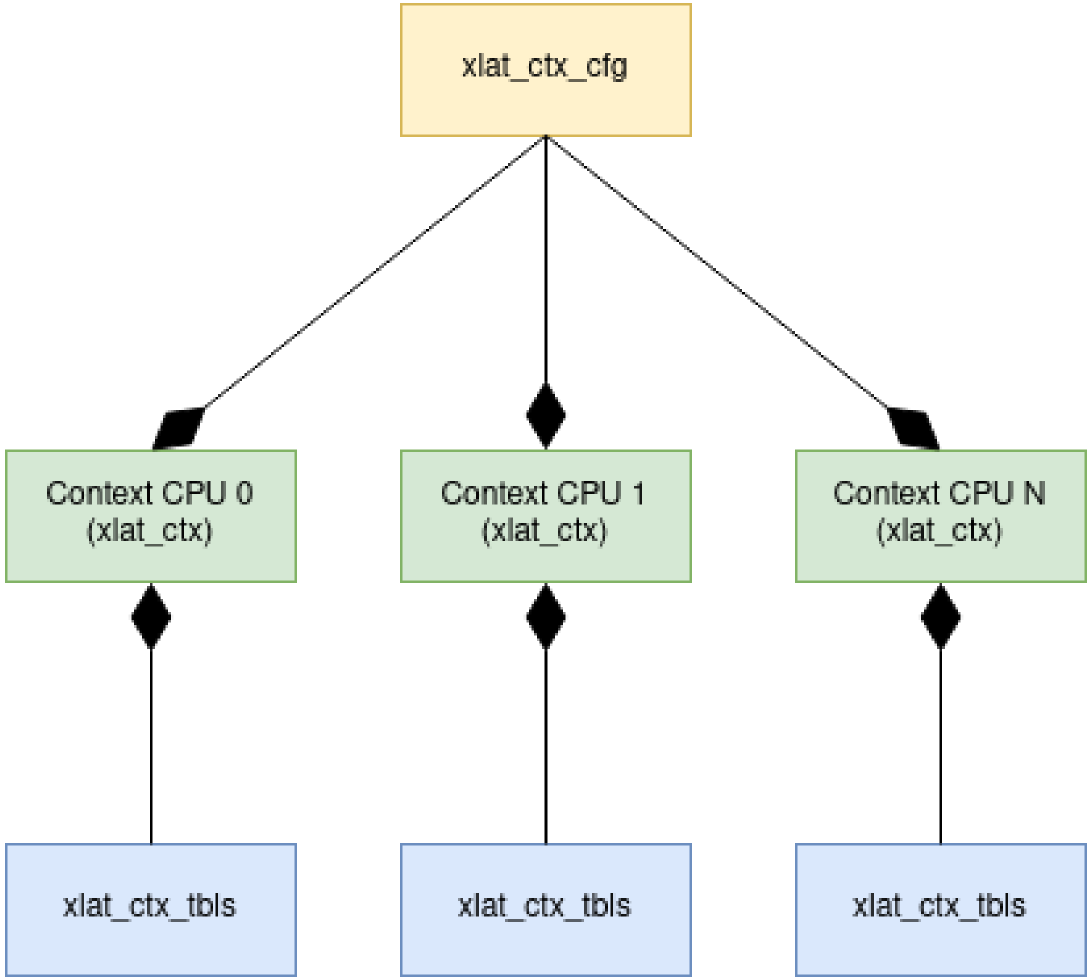
Stage 1 Translation Library

- Used TF-A xlat-v2 library as baseline
- Supports up to 52 bit-wide addresses and up to 5 levels of translation (when FEAT_LPA2 is enabled).
- Stateless. Uses the abstraction of a "context" to store status.
  - One context per CPU per VA Region*.
  - Contexts can be shared across CPUs.
- Uses TRANSIENT TTEs for dynamic mappings
  - It uses a bit flag to mark an invalid TTE as TRANSIENT.
  - An ordinary invalid TTE cannot be used on a mapping by the library.

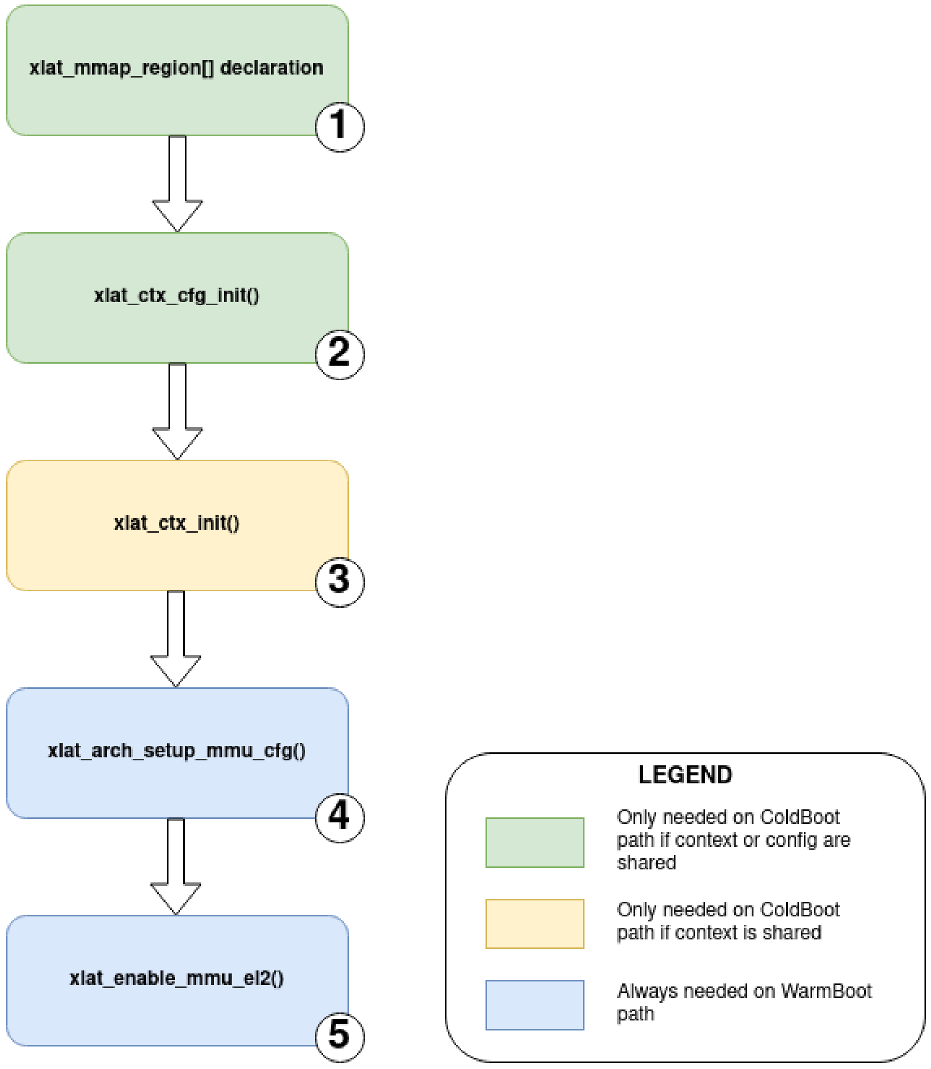


```
xlat
  include
    xlat_contexts.h
    xlat_defs.h
    xlat_high_va.h
    xlat_tables.h
  src
    aarch64
    fake_host
    xlat_contexts.c
    xlat_defs_private.h
    xlat_high_va.c
    xlat_tables_arch.c
    xlat_tables_core.c
    xlat_tables_private.h
    xlat_tables_utils.c
  tests
    CMakeLists.txt
    xlat_test_defs.h
    xlat_test_helpers.c
    xlat_test_helpers.h
    xlat_tests_base_g1.cpp
    xlat_tests_base_g2.cpp
    xlat_tests_base.h
    xlat_tests_lpa2.cpp
    xlat_tests_no_lpa2.cpp
  CMakeLists.txt
CMakeLists.txt
```

Stage 1 Translation Library – xlat_ctx



Stage 1 Translation Library – Initialization (I)



xlat_mmap_region[] declaration 1

xlat_ctx_cfg_init() 2

xlat_ctx_init() 3

xlat_arch_setup_mmu_cfg() 4

xlat_enable_mmu_el2() 5

LEGEND

🟩 Only needed on ColdBoot path if context or config are shared

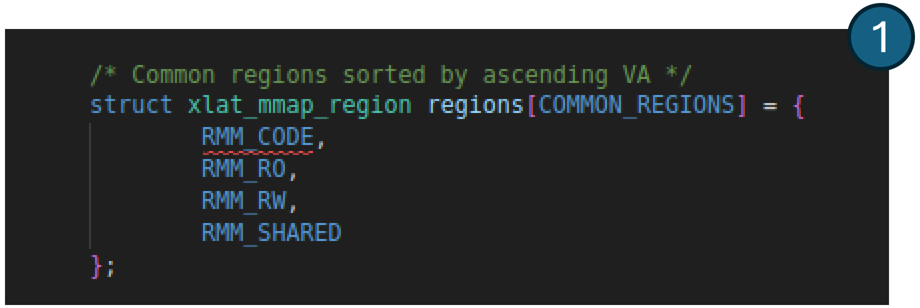
🟨 Only needed on ColdBoot path if context is shared

🟦 Always needed on WarmBoot path

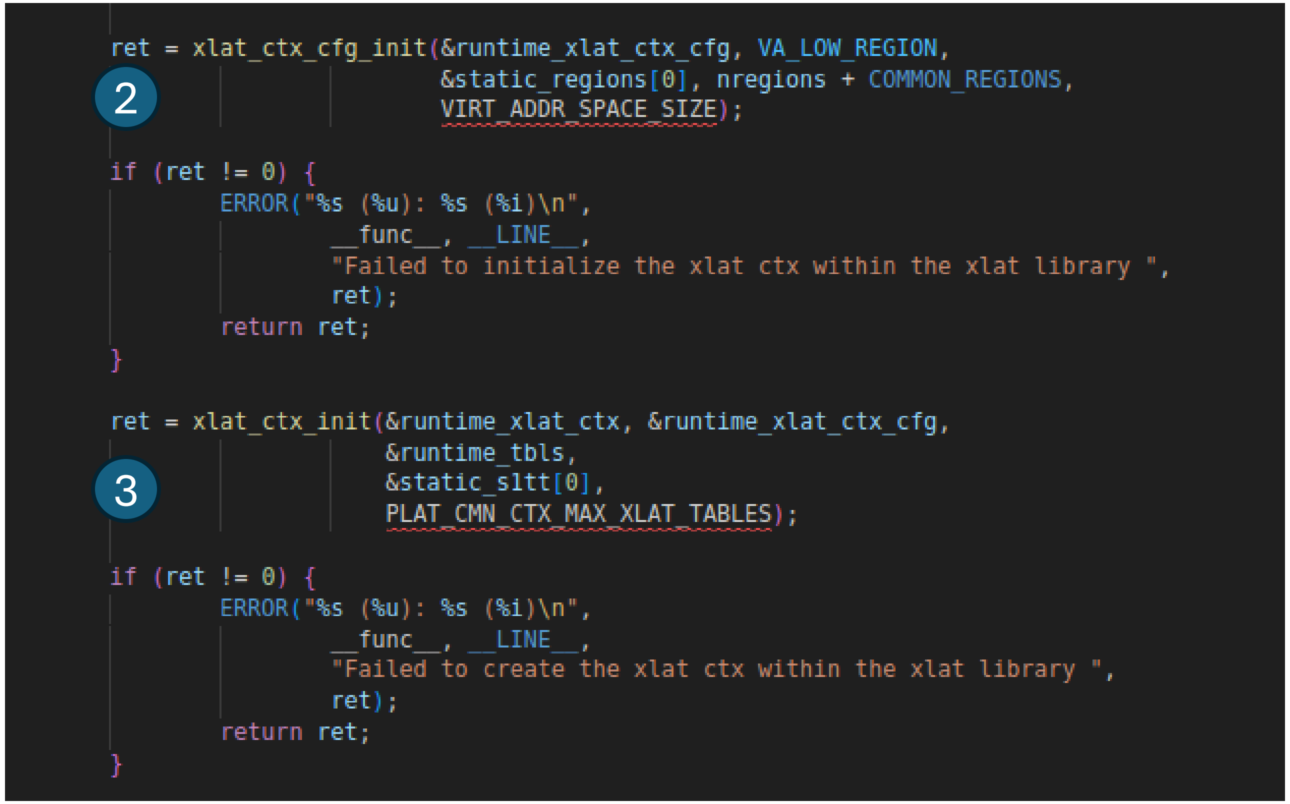
- Except for steps 4 & 5, which always needs to be done in WarmBoot path by every CPU, all the steps can be done either during ColdBoot or WarmBoot.
- Both VA regions must be created and configured before step 5.

Stage 1 Translation Library – Initialization (I)

1



```
/* Common regions sorted by ascending VA */
struct xlat_mmap_region regions[COMMON_REGIONS] = {
        RMM_CODE,
        RMM_RO,
        RMM_RW,
        RMM_SHARED
};
```



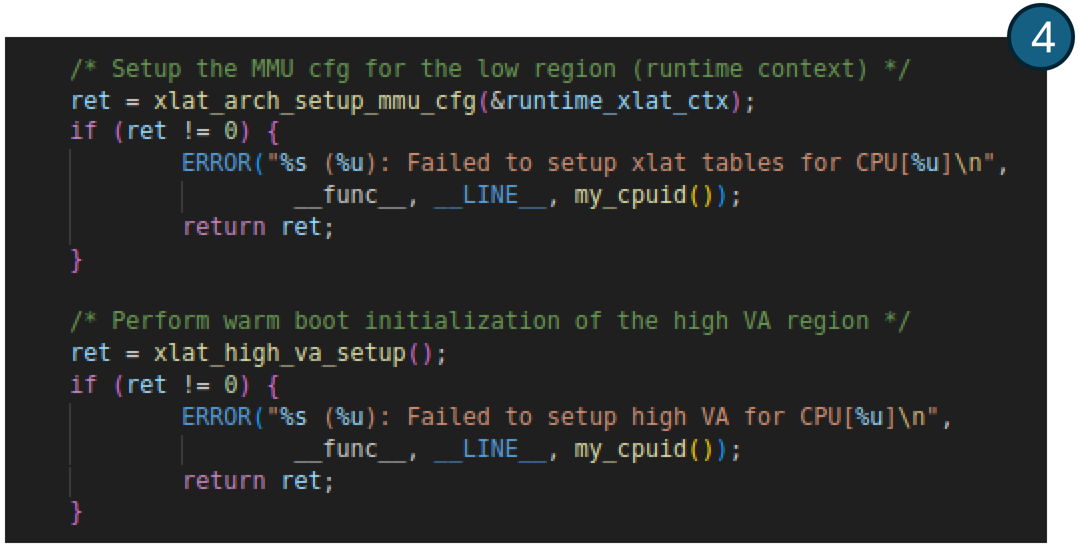
2

```
ret = xlat_ctx_cfg_init(&runtime_xlat_ctx_cfg, VA_LOW_REGION,
                        &static_regions[0], nregions + COMMON_REGIONS,
                        VIRT_ADDR_SPACE_SIZE);

if (ret != 0) {
        ERROR("%s (%u): %s (%i)\n",
                __func__, __LINE__,
                "Failed to initialize the xlat ctx within the xlat library ",
                ret);
        return ret;
}
```

3

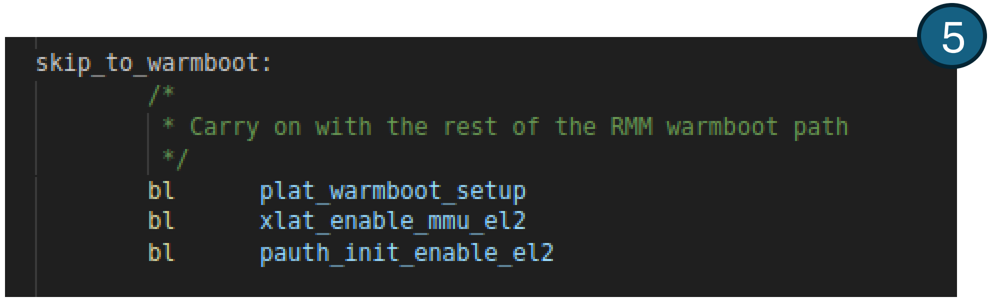
```
ret = xlat_ctx_init(&runtime_xlat_ctx, &runtime_xlat_ctx_cfg,
                    &runtime_tbls,
                    &static_s1tt[0],
                    PLAT_CMN_CTX_MAX_XLAT_TABLES);

if (ret != 0) {
        ERROR("%s (%u): %s (%i)\n",
                __func__, __LINE__,
                "Failed to create the xlat ctx within the xlat library ",
                ret);
        return ret;
}
```

4



```
/* Setup the MMU cfg for the low region (runtime context) */
ret = xlat_arch_setup_mmu_cfg(&runtime_xlat_ctx);
if (ret != 0) {
        ERROR("%s (%u): Failed to setup xlat tables for CPU[%u]\n",
                __func__, __LINE__, my_cpuid());
        return ret;
}

/* Perform warm boot initialization of the high VA region */
ret = xlat_high_va_setup();
if (ret != 0) {
        ERROR("%s (%u): Failed to setup high VA for CPU[%u]\n",
                __func__, __LINE__, my_cpuid());
        return ret;
}
```

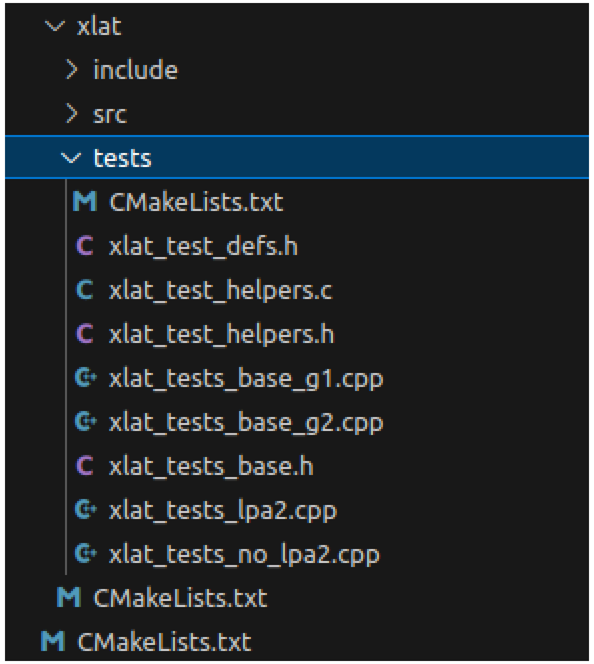
5



```
skip_to_warmboot:
        /*
         * Carry on with the rest of the RMM warmboot path
         */
        bl      plat_warmboot_setup
        bl      xlat_enable_mmu_el2
        bl      pauth_init_enable_el2
```

Unittests

- Support for unittests (CppUTest) using the *fake_host* architecture
- Different test groups run same tests with different configurations:
  - *xlat_tests_LPA2: FEAT_LPA2* Enabled
  - *xlat_tests_no_LPA2: FEAT_LPA2* Disabled
- Tests both regions



```
v xlat
  > include
  > src
  v tests
    M CMakeLists.txt
    C xlat_test_defs.h
    C xlat_test_helpers.c
    C xlat_test_helpers.h
    G xlat_tests_base_g1.cpp
    G xlat_tests_base_g2.cpp
    C xlat_tests_base.h
    G xlat_tests_lpa2.cpp
    G xlat_tests_no_lpa2.cpp
  M CMakeLists.txt
M CMakeLists.txt
```

Future work

- Remove recursive calls on some of the table creation APIs
- General code optimizations to improve efficiency
- Returned error codes need to be revisited
- The library can generate panic() under certain circumstances. We need to return an error code instead to the caller.
- Break the stage 1 translation library API into context manipultion APIs and general TTE manipulation APIs

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు

arm