NXP Semiconductors
Application Notes

Document Number: AN13276
Rev. 0, 08/2021

Linux Integration Example of S32V234 A53 SCST Library

by: NXP Semiconductors

Contents

1. Introduction

This application note describes an integration example of Cortex-A53® SCST Library in a Linux environment on S32V234 product, which provides an implementation hint when Cortex-A53 SCST Library is required to be deployed as a safety measure for Cortex-A53 cores. It is intended to assist system integrators to design and develop some applicable methods to adapt Cortex-A53 SCST Library to their OS environment.

This is supplementary to the S32V234 Cortex-A53 SCST User Manual available in NXP RTM release of Cortex-A53 SCST Library for S32V234. Please read Cortex-A53 S32V234 SCST User Manual in conjunction with the application note for a comprehensive understanding.

The integration example is based on S32V234 Linux BSP and some proprietary software components:

- S32V234 Linux BSP RTM v23.1
  - Linux v4.19.59
  - U-Boot v2018.07
- Secure Monitor patches from S32V234 VSDK v1.7.0
  - U-Boot patch for Secure Monitor support.
  - Linux device tree patch to reserve

memory for Secure Monitor firmware and SCST environment.

- Proprietary Cortex-A53 SCST for S32V234, 1.0.2 RTM
- Proprietary Secure Monitor Linux driver and firmware sample for SCST
- GCC compiler (BLD = 1620) 6.3.1 20170509 (build.sh rev=g01b30c3)

NOTE

For additional information refer to Reference.

2. Terms and Abbreviations

Terms and abbreviations	Meaning
Cortex-A53	The Cortex-A53 processor is a mid-range, low-power processor that implements the Arm® v8-A architecture.
SCST	Structural Core Self-Test
OS	Operating System, for example Linux
AArch64	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.
RTM	Released To Market
BSP	Board Support Package
VSDK	Vision Software Development Kit
CPU	A hardware implementation of the ARM Architecture.
EL0	The lowest Exception level. The Exception level that is used to execute user applications in Non-secure state.
EL1	The privileged Exception level. The Exception level that is used to execute operating systems in Non-secure state.
EL1h	Indicates use of the SP_EL1 Stack Pointer in the Exception Level 1. Compared with EL1t which indicates use of the SP_EL0 Stack Pointer.
EL2	The hypervisor Exception level. The Exception level that is used to execute hypervisor code in Non-secure state.
EL3	The Secure Monitor Exception level. The Exception level that is used to execute Secure Monitor code, which handles the transitions between Non-secure and Secure states. EL3 is always in Secure state.
MMU	Memory Management Unit
GPL	GNU General Public License
SM	The Secure Monitor is software that executes at the EL3 Exception level.
Firmware	Software that provides platform specific services. Firmware typically operates at an exception level higher than the operating system or Hypervisor which makes use of the firmware services.
GIC	Generic Interrupt Controller

Terms and abbreviations	Meaning
SMC	Secure Monitor Call. An Arm assembler instruction that causes an exception that is taken synchronously into EL3.
ASIL	Automotive Safety Integrity Level

3. Integration Limitations

Cortex-A53 SCST Library is a software self-test method and can be integrated into an application running in a Linux environment with the following limitations:

- Core-tests from Cortex-A53 SCST Library cannot be executed within ordinary Linux user space application as these applications run in Exception Level EL0, which lacks privileges required by the tests (e.g. some system registers are not writable in this exception level, some instructions cannot be executed, etc.). This is also the case of applications executed with root privileges, which give the application additional permissions to access files and commands on Unix systems, but do not provide extended access to hardware resources. Therefore, the SCST Library is designed to be executed from Exception Level EL1, which is normally used by the Linux Kernel and which provides accesses to most of the hardware features from software.
- Another specific of the Linux environment is that it uses MMU to separate address spaces of running processes from each other and from kernel address space. On 64-bit system, each process has its 64-bit address space, where part of the address space is reserved for the application's data (program, data, heap, stack, linked libraries…) and part of the address is reserved for kernel's data (shared across all processes within the system). The memory mapping of a program file with all its sections into an address space is done by an operating system on program startup. The SCST Library currently cannot use this feature, but rather relies on static mapping of individual sections to target memory based on the definitions in the linker file and configuration header file. It is possible to run the SCST Library with MMU enabled, but virtual to physical address translation is still needed to be known in advance as it must be provided by the user at compile time.

4. Possibilities of integration

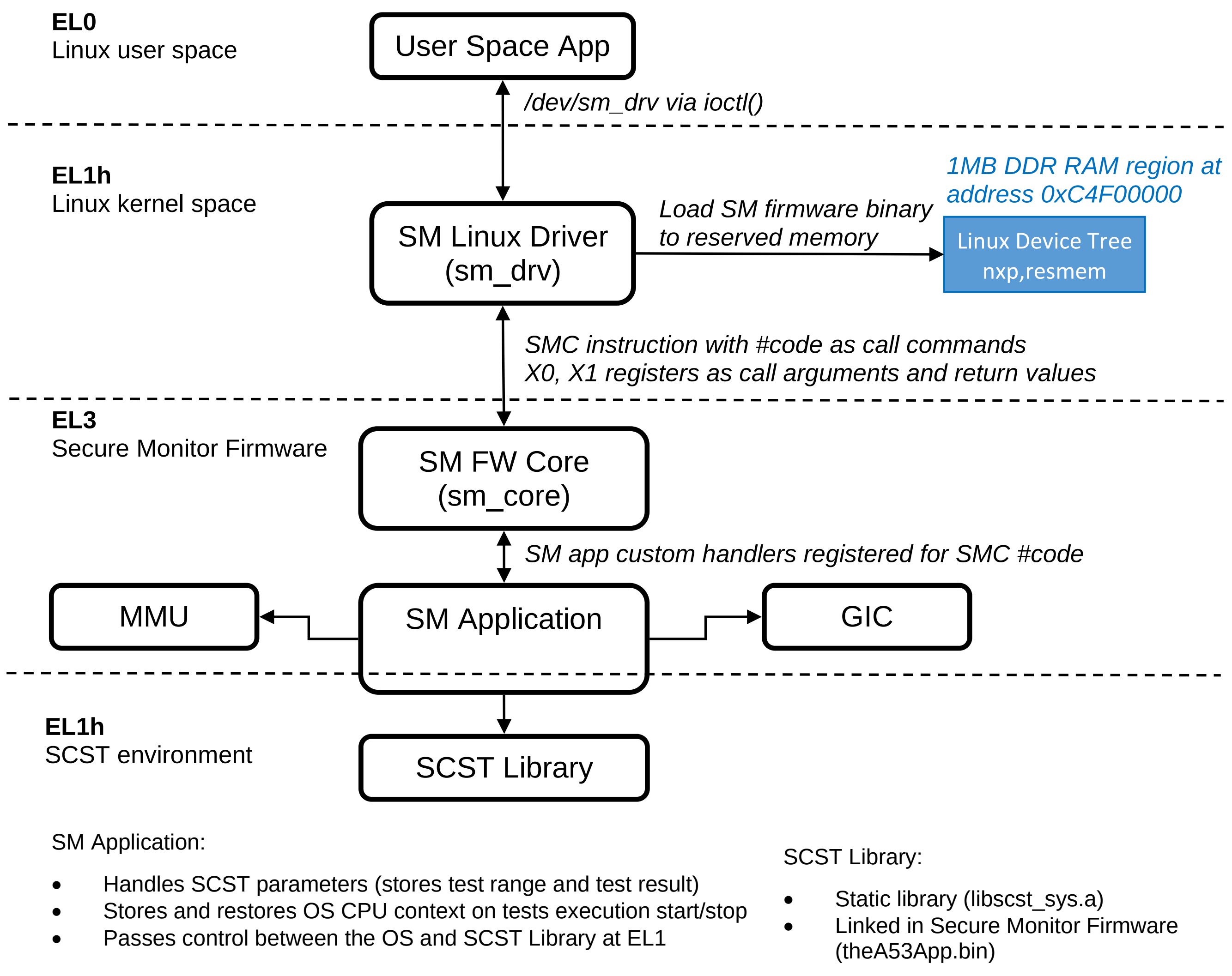
There are two technical possibilities of how to integrate SCST Library into an application running with a Linux environment:

- Integration of the SCST Library directly into Linux kernel, thus linking the kernel with the library, or using binary loadable module. This scenario is not applicable because of the legal issue associated with the fact that the Linux kernel falls under the GPL, while SCST Library is a proprietary product.
- Using a Linux kernel module to interface with custom firmware running in Secure Monitor (Exception Level EL3), that will execute the tests from SCST Library at EL1.

The second option was considered in the demo application, which was proved to be working.

5. Description of demo application

The structure of the demo application is previewed in the figure below. It consists of three executable files. The idea is that on request from Linux environment, custom firmware running in Secure Monitor mode (EL3) is instructed to switch between Linux environment and SCST environment. Once the system has switched into SCST environment, the specified tests can be executed and result is passed back to the Linux environment. In Linux environment, a Kernel module is used to interface with custom firmware. Linux user space application accesses functions of the kernel module, thus enabling it to run the SCST tests and get the result.


EL0
Linux user space
User Space App
/dev/sm_drv via ioctl()
EL1h
Linux kernel space
SM Linux Driver (sm_drv)
Load SM firmware binary to reserved memory
1MB DDR RAM region at address 0xC4F00000
Linux Device Tree nxp,resmem
SMC instruction with #code as call commands
X0, X1 registers as call arguments and return values
EL3
Secure Monitor Firmware
SM FW Core (sm_core)
SM app custom handlers registered for SMC #code
MMU
SM Application
GIC
EL1h
SCST environment
SCST Library

SM Application:

- Handles SCST parameters (stores test range and test result)
- Stores and restores OS CPU context on tests execution start/stop
- Passes control between the OS and SCST Library at EL1

SCST Library:

- Static library (libscst_sys.a)
- Linked in Secure Monitor Firmware (theA53App.bin)

Figure 1. **Structure of Linux SCST integration demo**

5.1. Prerequisites

The Secure Monitor firmware is a bare-metal firmware, that is loaded by the Linux kernel module sm_drv upon initialization into reserved region of DRAM memory. The address, that the firmware is loaded to, must match the address that the firmware is linked to, otherwise the whole system hangs. The DRAM memory reservation is done by Device Tree Blob (DTB) file for target device found in the arch/arm64/boot/dts sub-directory of the Linux kernel source tree. An example of how to reserve 1MB of memory at address 0xC4F00000 is presented below. For detailed description of DTB files please see Reference.

```
reserved-memory {
        #address-cells = <2>;
        #size-cells = <2>;
        ranges;
    resmem: rmem@C4F00000 {
                reg = <0 0xC4F00000 0 0x100000>;
                no-map;
    };
};

nxpresmem: themem@C4F00000 {
        status = "okay";
        compatible = "nxp,resmem";
        memory-region = <&resmem>;
        interrupts = <0 0 4>;
};
```

The example DTS patch included in the S32V234 Yocto VSDK build can be found in Reference.

During the boot process of the Linux environment, it's necessary to enable the Secure Monitor Call (SMC) instructions from Non-secure EL1 to be handled by Secure Monitor firmware running at EL3. To enable this feature, HCR_EL2.TSC and SCR_EL3.SMD bits of the system registers shall be cleared to zero just before passing control to the Linux kernel image. Further, address of the exception vector table used at EL3, which is part of the Secure Monitor firmware and known in advance, must be written to the VBAR_EL3 system register. This is best achieved by modifying the bootloader that is used to boot the Linux environment, such as U-Boot, because it has sufficient privileges to access the system registers on startup. This setup shall be done on all A53 cores that are supposed to run SCST environment. In this example, the boot core #0 is configured as described and it will be the only core running SCST environment.

The example patch for U-Boot Yocto VSDK build can be found in Reference.

5.2. Secure Monitor Firmware

The Secure Monitor firmware runs in AArch64 execution state in Exception Level EL3 and consists of two major components – the sm_core component and the sm_application component.

The sm_core component is a generic component that provides interface between software running at lower Exception Levels on the CPU and custom applications running in Secure Monitor. It handles incoming SMC calls from lower Exception Levels (in this case from Linux environment), processes call arguments passed in registers X0 (parameter_id) and X1 (parameter_value) and then invokes corresponding application. Applications in Secure Monitor register their callbacks within sm_core component on initialization to handle specific SMC exceptions as distinguished by SMC exception syndrome (i.e. “#code” in SMC instruction in Figure 1).

The sm_application component is custom application running partially at EL3 and partially at EL1h. In EL3 it is responsible for processing of SCST parameters, storing and restoring Linux CPU context, passing control to the SCST environment and then back to the Linux environment. It works like a simple hypervisor, that switches between two guest operating systems on a single hardware. In EL1h, it executes a single procedure `scst_execute_core_tests()`, that represents SCST environment. This function acquires start and end test indexes from Secure Monitor and then directly calls `a53_sys_scst_execute_core_tests()` function from SCST Library.

There are currently several commands and parameters supported by the sm_application. They are listed in Table 1 *and* Table 2 as implementation example.

Table 1. Commands supported by sm_application

Command	SMC #code	Description
`SC_CMD_INIT`	0x1001	Initialize the sm_application
`SC_CMD_START`	0x1002	Switch from Linux to SCST environment and execute core tests
`SC_CMD_STOP`	0x1003	Switch from SCST back to Linux environment (only used by SCST environment)
`SC_CMD_CONFIG_SET`	0x1004	Change SCST configuration parameter (must be writable)
`SC_CMD_CONFIG_GET`	0x1005	Read SCST configuration parameter (must be readable)

Table 2. Parameters supported by sm_application

Parameter	Readable	Writeable	Description
APP_KEY_PRM_START_IDX	Yes	Yes	Index of the first SCST test to execute
APP_KEY_PRM_END_IDX	Yes	Yes	Index of the last SCST test to execute
APP_KEY_PRM_SCST_RESULT	Yes	No	Returned signature of the last SCST run

The sm_application also makes use of GIC controller and MMU controller. Using the GIC controller, it disables external interrupts once the control is passed to the SCST environment and restores them back when returning to Linux environment. This is necessary, because SCST environment cannot handle interrupts, that were previously setup by Linux environment. The MMU controller can be used when SCST Library is configured to use virtual addresses for testing purposes. The sm_application is then responsible for proper MMU re-configuration and page table setup before switching into SCST environment.

The sm_core, sm_application and SCST Library together correspond to a single executable file. Therefore, the SCST environment, as displayed in Figure 1, is de-facto built into the Secure Monitor firmware, but it is executed independently in a different Exception Level (EL1h).

5.3. Linux Kernel Module

The Linux kernel module sm_drv provides means to load the Secure Monitor firmware from Linux environment to reserved memory partition and to communicate with it by using dedicated SMC instructions plus core registers X0 and X1. It implements the communication protocol used by sm_core module on the Linux side.

Once the driver module is successfully loaded into Linux kernel, it creates a new device /dev/sm_drv within the Linux filesystem. This device provides ioctl interface to support communication between Linux user space application and the Secure Monitor firmware. Following commands are defined with this interface.

Table 3. sm_drv ioctl interface commands

Command	Description
SM_DRV_IOCTL_INIT	Initialize the firmware
SM_DRV_IOCTL_START	Start firmware operation
SM_DRV_IOCTL_STOP	Stop firmware operation
SM_DRV_IOCTL_SET_CFG	Set configuration parameter
SM_DRV_IOCTL_GET_CFG	Get configuration parameter
SM_DRV_IOCTL_REG_SIG	Register asynchronous signal listener
SM_DRV_IOCTL_UNREG_SIG	Cancel registration of asynchronous signal listener
SM_DRV_IOCTL_ENABLE_EVENTS	Enable/Disable asynchronous events reporting by firmware

These commands are either handled directly by kernel module or are internally translated to SMC calls and handled by Secure Monitor firmware.

5.4. Linux User Space Application

A command line application can be developed to execute SCST tests from Linux user space. It may only be executed once the sm_drv kernel module was successfully loaded using *insmod* Linux command. The user space application can use two arguments - the first and last SCST test indexes. It then forwards these indexes to the SCST environment and requests test execution, both using sm_drv ioctl interface. When control is returned to the application, it reads the resulting signature and prints it to the standard output. The demo application currently uses only subset of commands defined in *Table 3* to accomplish this task.

5.5. Execution Time

All 60 atomic tests can be invoked through a minimum call to the test shell executed at EL1h:

```
/* Execute tests */
test_result = a53_sys_scst_execute_core_tests( 0, 59 );
```

The typical execution time of a complete Linux user space SCST call is around 6.6 milliseconds.

6. Summary

The demo application described in this application note uses Secure Monitor firmware to implement the context switch between Linux OS environment and SCST environment. It provides an idea on how to integrate SCST Library in OS scenario and it’s system integrator’s responsibility to develop application specific integration method to meet system safety requirements.

Please note that the Linux Integration Example can be provided “AS IS” with NXP’s approval for demo purpose only. Also note that the example itself could be used to detect hardware faults in the processor cores, but it’s not sufficient to achieve required ASIL of an application running in the Linux environment.

7. Reference

[1] A53-S32V234 AArch64 Structural Core Self Test Library User Manual, Rev. 1.4, 24 October 2019

[2] S32V234 Cortex A53 Structural Core Self-Test Software

[3] Automotive SW - Linux BSP

[4] Vision SDK Software

[5] Arm® Cortex®-A53 MPCore Processor, Technical Reference Manual, Revision: r0p4

[6] Embedded Linux Wiki - Device Tree Reference

[7] DTS patch for reserved memory: 0001-s32v234-dts-Add-VSDK-specific-configuration.patch

[8] U-Boot patch for Secure Monitor support: 0001-secure-monitor-enable-Secure-Monitor-support-2018.07.patch

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN13276
Rev. 0
08/2021