Using MA35D1 TSI Monitor through OP-TEE

Example Code Introduction for 64-bit NuMicro® Family

Document Information

Application	This example code implements the secure monitoring functionality of memory blocks using the MA35D1 TSI through OP-TEE.
BSP Version	Linux-5.10.x
Hardware	NuMaker-HMI-MA35D1 V1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. Overview

This example code uses the TSI within the MA35D1 series microprocessor to perform integrity checks on specific memory areas.

TSI stands for Trusted Secure Island and is part of the MA35D1 SoC. Its purpose is to provide secure encryption and key storage services. In the MA35D1, the TSI operates as an independent subsystem and runs separately from the Cortex®-A35 main system. The only way for programs running on the MA35D1 Cortex-A35 to access the TSI is through Wormhole1, a hardware component with security attributes that provides bidirectional communication between the Cortex-A35 main system and the TSI subsystem. The TSI operates passively, constantly waiting for requests from the Cortex-A35. It runs in an infinite loop, receiving requests, executing them, and responding with an ACK through Wormhole1. To conserve power, the TSI automatically enters power-saving mode if no requests are received for a period.

OP-TEE (Open Portable Trusted Execution Environment) is an open-source Trusted Execution Environment (TEE, https://optee.readthedocs.io/en/latest/general/about.html) developed by Linaro, designed specifically for ARM TrustZone® technology. Its purpose is to provide a secure execution environment in embedded systems for running sensitive applications and handling sensitive data. It has the following application scenarios:

- Digital signature and verification: OP-TEE can be used to protect key operations, ensuring that the private key for digital signatures is only used in a secure environment, preventing its leakage.
- Data encryption and decryption: Encryption and decryption operations are handled through the TEE to ensure the security of sensitive data.
- Authentication: TEE is used to store and verify identity credentials, preventing credentials from being stolen or tampered with.
- Payment systems: TEE can protect sensitive financial transaction data, preventing security vulnerabilities during the transaction process.

This example code implements a client application on the Linux side based on OP-TEE, which sends requests to a trusted application (TA) in the secure world (OP-TEE OS). The TA then calls the TSI to perform memory integrity verification. TSI can access the entire memory space of the MA35D1 and periodically calculates the hash of the monitored memory area using the Crypto SHA Engine, ensuring that the memory space has not been tampered with. The starting address and size of the areas can be configured by the user, with up to four areas being monitored. Since TSI is an independent subsystem, it ensures the independence and reliability of its computations.

1.1 Principle

Cortex-A35 can only access the TSI When CPU running on the TrustZone Secure (TZS) mode. Since Linux operates in the non-secure world, it does not have the rights to access Wormhole1 and TSI. Therefore, a secure operating system like OP-TEE, running in secure mode, is required as a communication bridge for Linux to access TSI. MA35D1's Linux BSP supports

OP-TEE's Crypto, Key Store, and TRNG client drivers to help Linux access the TSI in the secure world. The communication between Linux and OP-TEE can be referred to as Figure 1-1.

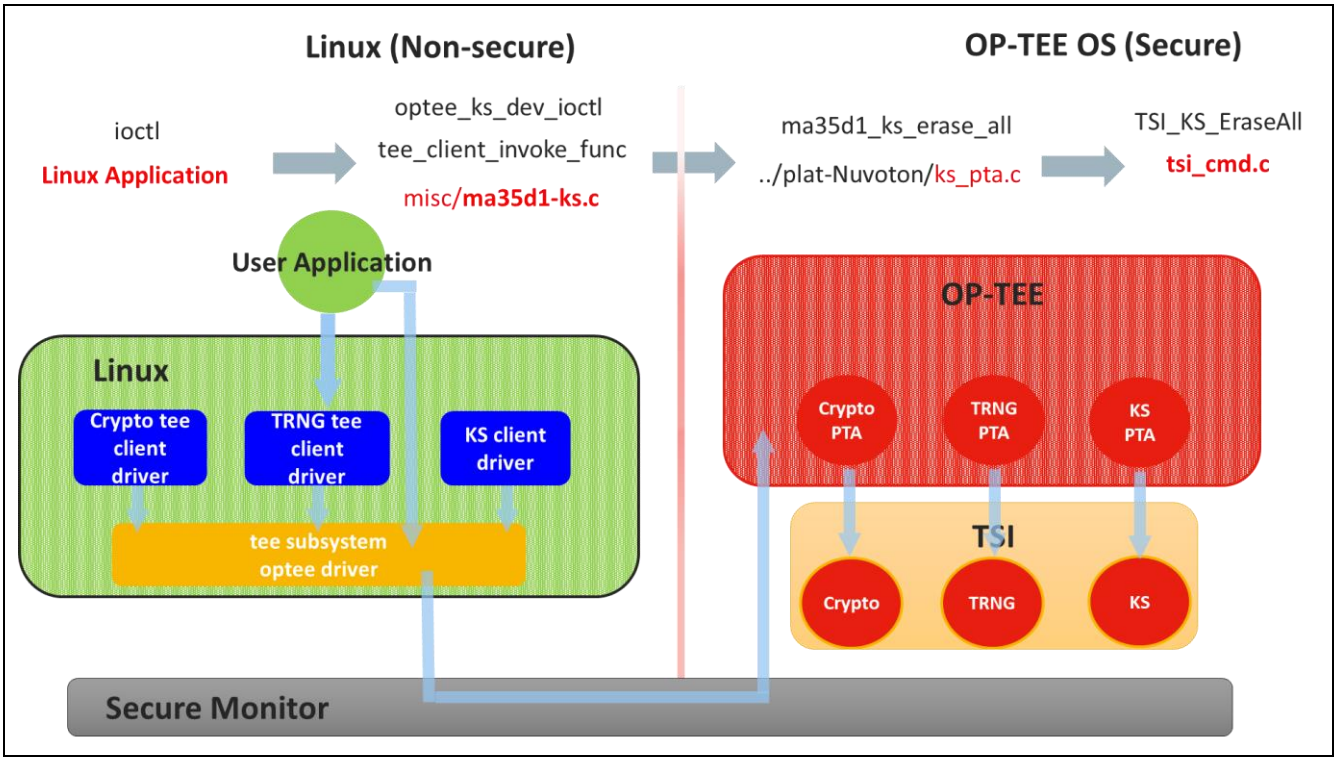


Figure 1-1 Communication between Linux and OP-TEE

The attribute of Wormhole1 is TrustZone Secure (TZS). Only code running in secure mode can access Wormhole1. A35 switches the security attribute (NS->S) through the Secure Monitor to enter secure mode and run OP-TEE OS, while the Linux kernel runs in non-secure mode. The Crypto, TRNG, and Key Store drivers in the MA35D1 Linux kernel are implemented as client-side drivers of OP-TEE and require OP-TEE to help access TSI. TSI receives requests from Wormhole1, executes them, and responds with an ACK indicating the execution results, as shown in Figure 1-2.

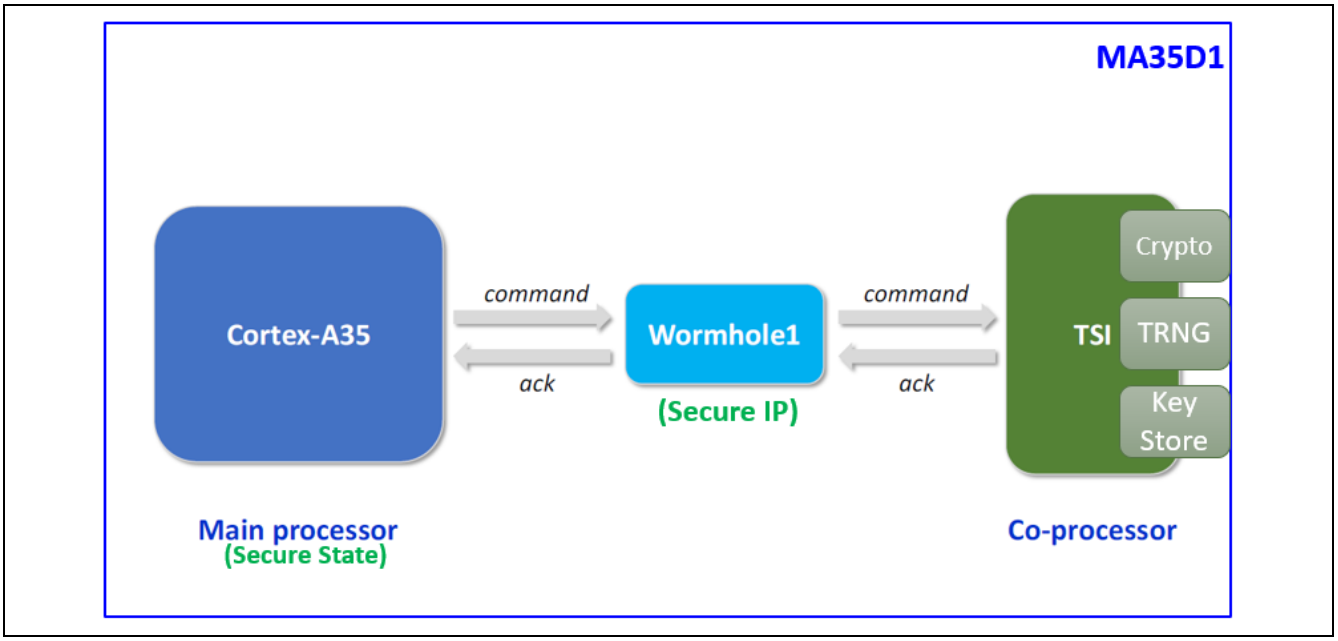


Figure 1-2 Communication A35 and TSI

TSI provides the commands TSI_Monitor_Set, TSI_Monitor_Status, and TSI_Monitor_Status, which allow TSI to monitor the integrity of specified memory areas. TSI can simultaneously support up to four monitors in operation.

TSI checks and compares the memory areas using the Crypto SHA engine. At any time, MA35D1 can use the TSI_Monitor_Status command to check the integrity status of the memory. The monitor will periodically check the memory until the TSI_Monitor_Close command is received.

- TSI_Monitor_Set

  Set the starting position, size (maximum: 32 MB), and time interval for the memory monitoring.

- TSI_Monitor_Status

  Request the TSI to report the status of the monitored segment.

- TSI_Monitor_Close

  Close the periodic monitoring.

1.2 Demo Result

The execution results of the example code are shown in Figure 1-3. After the MA35D1 starts successfully and running in the Linux, you can find OP-TEE related examples in the usr/bin directory. Except for optee_example_TSI_Monitor, all other examples are provided by OP-TEE by default. To execute optee_example_TSI_Monitor, input the TSI_Monitor_Set related parameters for the first time (optee_example_TSI_Monitor cmd set, time interval, base address, size). After successful setup, you will see TSI returning the ID (0x35). When executing

TSI_Monitor_Status and TSI_Monitor_Close, this ID value must be passed as a parameter to successfully execute the example.

In the next step, input the TSI_Monitor_Status related parameters (optee_example_TSI_Monitor cmd status, ID). After successful execution, you will see TSI returning the status of the monitored region (0x14300035). The value 143 corresponds to 0x0143, indicating that the input command is TSI_Monitor_Status, 00 corresponds to 0x00 indicating no abnormal status, and 35 represents the ID 0x35 corresponding to the monitored region.

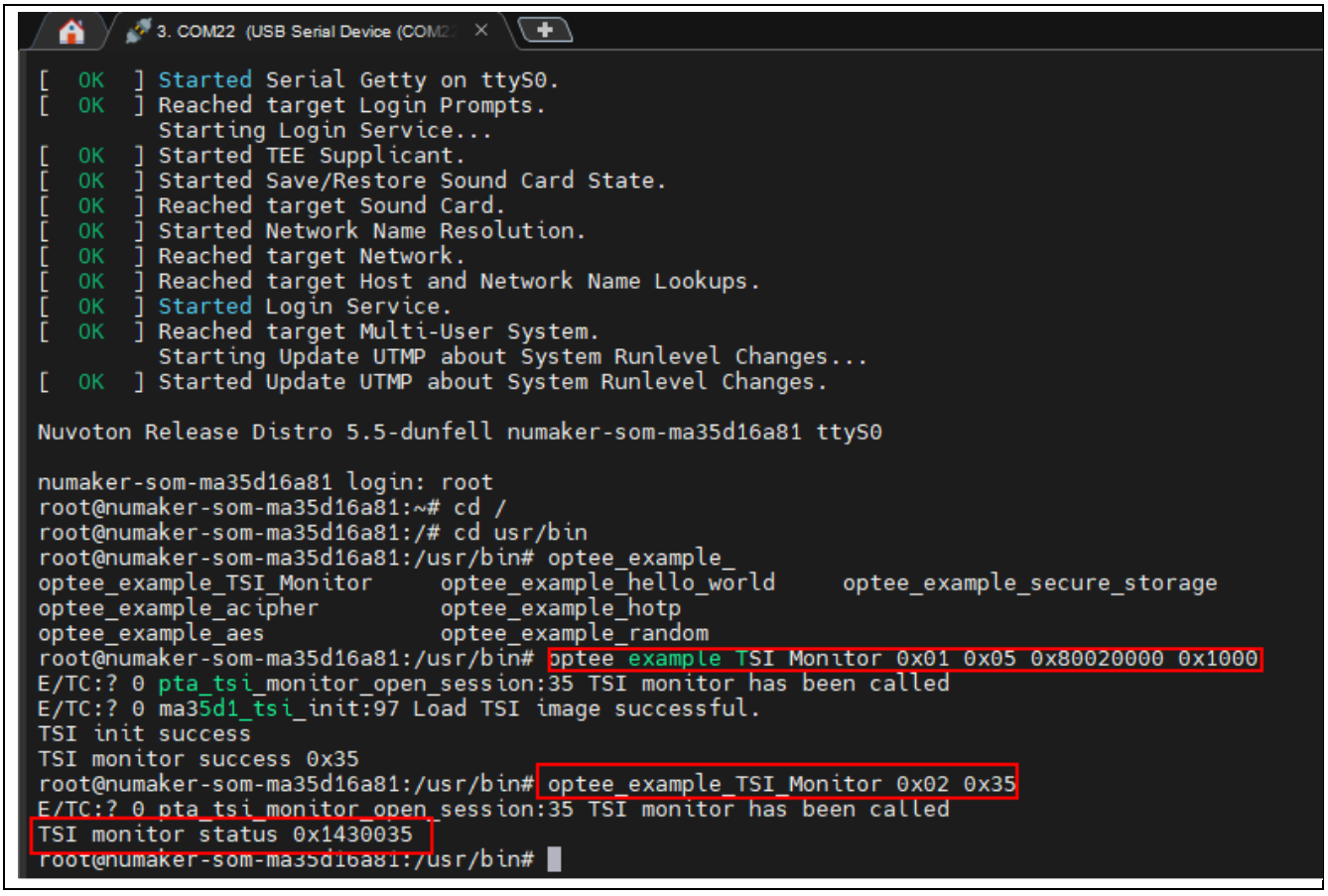


Figure 1-3 Execution Results

2. Code Description

The TSI Monitor example is divided into two parts:

- optee_example

This is the application running on the Linux side, also known as the Client Application (CA). The CA interacts with the secure side through APIs provided by the OP-TEE driver. Each command in the CA corresponds to an operation on the OP-TEE OS side. The actual operations are performed in the TEE OS or in the Trusted Application (TA) on the application layer, and the results are then returned to CA side.

- optee_os_ma35d1

The architecture is divided into OS layer and application layer. The OS layer handles secure function calls from Linux, while the application layer contains the Trusted Application for implementing main function such as encryption/decryption, secure storage, and interaction with the secure hardware.

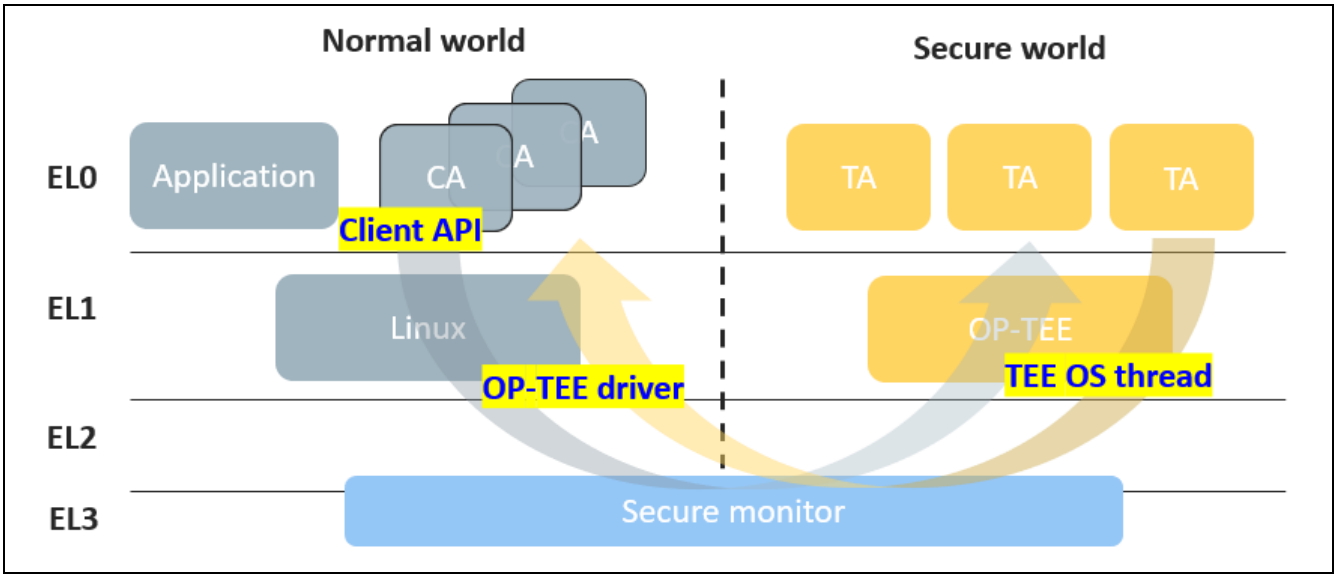


Figure 2-1 OP-TEE Architecture

The first part, optee_example, initially calls Prepare_tee_session to confirm whether the TSI monitor session is successfully established. Then, use sscanf, which reads the application parameters to determine the actual operation to be performed. The code can be found in main.c.

```
int main(int argc, char *argv[])
{
    struct test_ctx ctx;
    uint32_t monitor_argv1,monitor_argv2,monitor_argv3,monitor_argv4;

    /* Open tee application */
    Prepare_tee_session(&ctx);

    /*
```

```
     * argv[1]: TSI monitor command
     * argv[2]:
     * For TSI monitor set, it represents monitor time interval
     *     unit: 1 second, Maximum is 255
     * For TSI monitor status/ TSI monitor close, it represents monitor ID
     * argv[3]: Base Address
     * argv[4]: Monitor size, Maximum is 32 MB (0x2000000)
     */

    sscanf (argv[1], "%x", &monitor_argv1);

    /*
     * Check tsi operation:
     * Monitor_Set
     * Monitor_Status
     * Monitor_Close
     */
    switch (monitor_argv1)
    {
    case monitor_set:
         sscanf (argv[2], "%x", &monitor_argv2);
         sscanf (argv[3], "%x", &monitor_argv3);
         sscanf (argv[3], "%x", &monitor_argv4);
         Monitor_Set(&ctx, monitor_argv2, monitor_argv3, monitor_argv4);
         break;

    case monitor_status:
         sscanf (argv[2], "%x", &monitor_argv2);
         Monitor_Status(&ctx, monitor_argv2);
         break;

    case monitor_close:
         sscanf (argv[2], "%x", &monitor_argv2);
         Monitor_Close(&ctx, monitor_argv2);
         break;

    default:
         break;
    }

    /*Close tee application*/
    Terminate_tee_session(&ctx);

    return 0;
}
```

The related operation APIs are as follows, providing functions for monitoring setup (Monitor_Set), status reporting (Monitor_Status), and closing the monitoring (Monitor_Close).

```
void Monitor_Set(struct test_ctx *ctx, uint32_t interval, uint32_t baseaddress, uint32_t
size)
{
    TEEC_Operation op;
    TEEC_Result res;
    uint32_t err_origin;
    /* Clear the TEEC_Operation structure */
```

```
    memset(&op, 0, sizeof(op));

    /*
     * Invokes a Command within the specified Session.
     * Command: TSI initial
     */
    res = TEEC_InvokeCommand(&ctx->sess, TA_TSI_INIT, &op,
              &err_origin);
    if (res != TEEC_SUCCESS)
         printf("TSI init fail\n");
    else
         printf("TSI init success\n");


    /* Set TSI monitor parameter */
    memset(&op, 0, sizeof(op));
    op.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INPUT, TEEC_VALUE_OUTPUT,
TEEC_VALUE_INPUT, TEEC_NONE);
    op.params[0].value.a = interval;
    op.params[0].value.b = baseaddress;
    op.params[2].value.a = size;

    /*
     * Invokes a Command within the specified Session.
     * Command: TA_TSI_MONITOR_SET
     */
     res = TEEC_InvokeCommand(&ctx->sess, TA_TSI_MONITOR_SET, &op,
     &err_origin);
     if (res != TEEC_SUCCESS)
             errx(1, "TEEC_InvokeCommand failed with code 0x%x origin 0x%x",
             res, err_origin);
         else
         printf("TSI monitor success%x\n", op.params[1].value.a );

}

void Monitor_Status(struct test_ctx *ctx, uint32_t id)
{
    TEEC_Operation op;
    TEEC_Result res;
    uint32_t err_origin;
    /* Clear the TEEC_Operation structure */
    memset(&op, 0, sizeof(op));
    /* Set TSI monitor parameter */
    op.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INPUT, TEEC_VALUE_OUTPUT,
                      TEEC_NONE, TEEC_NONE);
    op.params[0].value.a = id;            /* Monitor ID */
    /*
     * Invokes a Command within the specified Session.
     * Command: TA_TSI_MONITOR_STATUS
     */
     res = TEEC_InvokeCommand(&ctx->sess, TA_TSI_MONITOR_STATUS, &op,
     &err_origin);
         printf("TSI monitor status%x\n",op.params[1].value.a);

}
```

```
void Monitor_Close(struct test_ctx *ctx, uint32_t id)
{
    TEEC_Operation op;
    TEEC_Result res;
    uint32_t err_origin;
    /* Close TSI monitor 1 status */
    memset(&op, 0, sizeof(op));
    op.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INPUT, TEEC_VALUE_OUTPUT,
                     TEEC_NONE, TEEC_NONE);
    op.params[0].value.a = id;            /* Monitor ID */
    /*
     * Invokes a Command within the specified Session.
     * Command: TA_TSI_MONITOR_CLOSE
     */
     res = TEEC_InvokeCommand(&sess, TA_TSI_MONITOR_CLOSE, &op,
     &err_origin);
     if (res != TEEC_SUCCESS)
             errx(1, "TEEC_InvokeCommand failed with code 0x%x origin 0x%x",
             res, err_origin);
         else
         printf("TSI Close status%x\n",op.params[1].value.a);
}
```

The CA operations have corresponding functionalities in TSI_Monitor_pta.c under the optee-os-ma35d1 directory. These operations involve calling the TSI via the PTA (Pseudo Trusted Application), which executes the requested tasks and then returns the results to the CA.

```
// SPDX-License-Identifier: GPL-2.0
/*
 * Copyright (C) 2020 Nuvoton Technology Corp. All rights reserved.
 *
 */

#include <kernel/pseudo_ta.h>
#include <mm/core_memprot.h>
#include <platform_config.h>
#include <io.h>
#include <platform_config.h>
#include <tsi_cmd.h>
#include <TSI_Monitor_ta.h>

#define PTA_NAME "tsi_monitor.pta"

/*
 * Open TSI monitor session
 */

TEE_Result pta_tsi_monitor_open_session(uint32_t param_types,
     TEE_Param __maybe_unused params[4],
     void __maybe_unused **sess_ctx)
{
    uint32_t exp_param_types = TEE_PARAM_TYPES(TEE_PARAM_TYPE_NONE,
                           TEE_PARAM_TYPE_NONE,
```

```
                        TEE_PARAM_TYPE_NONE,
                        TEE_PARAM_TYPE_NONE);

    EMSG("TSI monitor has been called");

    if (param_types != exp_param_types)
     return TEE_ERROR_BAD_PARAMETERS;

    /* Unused parameters */
    (void)&params;
    (void)&sess_ctx;

    /* If return value != TEE_SUCCESS the session will not be created. */
    return TEE_SUCCESS;
}

/*
 * Close TSI monitor session
 */
void pta_tsi_monitor_close_session(void __maybe_unused *sess_ctx)
{
    (void)&sess_ctx; /* Unused parameter */
    IMSG("Goodbye!\n");
}

/*
 * TSI initialization
 */

static TEE_Result tsi_init(void)
{
    vaddr_t sys_base = core_mmu_get_va(SYS_BASE, MEM_AREA_IO_SEC);
    vaddr_t tsi_base = core_mmu_get_va(TSI_BASE, MEM_AREA_IO_SEC);
     if (!(io_read32(sys_base + SYS_CHIPCFG) & TSIEN))
         return ma35d1_tsi_init();
}

/*
 * Send monitor command to TSI
 */

static TEE_Result tsi_monitor_set(uint32_t types,
                    TEE_Param params[TEE_NUM_PARAMS])
{
    vaddr_t sys_base = core_mmu_get_va(SYS_BASE, MEM_AREA_IO_SEC);
    vaddr_t tsi_base = core_mmu_get_va(TSI_BASE, MEM_AREA_IO_SEC);
    int ret;
    uint32_t *status_buff;
    uint32_t size;

    /* Check types of parameters */
    if (types != TEE_PARAM_TYPES(TEE_PARAM_TYPE_VALUE_INPUT,
                    TEE_PARAM_TYPE_VALUE_OUTPUT,
                    TEE_PARAM_TYPE_VALUE_INPUT,
                    TEE_PARAM_TYPE_NONE)) {
        EMSG("bad parameters types: 0x%" PRIx32, types);
```

```
        return TEE_ERROR_BAD_PARAMETERS;
    }

    /* Check TSI is enable and do monitor set*/
    if (!(io_read32(sys_base + SYS_CHIPCFG) & TSIEN)) {
        ret = TSI_Monitor_Set(params[0].value.a,        /* Interval      */
            params[0].value.b,                          /* Base address  */
            params[2].value.a,                          /* Size */
            &params[1].value.a                          /* Status return */
            );
        if (ret != ST_SUCCESS) {
              EMSG("TSI Monitor Set failed! 0x%x\n", ret);
            return ret;
        }
        return TEE_SUCCESS;
    }


    return 0x00000002;
}

/*
 * Send status check command to TSI
 */
static TEE_Result tsi_monitor_status(uint32_t types,
                     TEE_Param params[TEE_NUM_PARAMS])
{
    vaddr_t sys_base = core_mmu_get_va(SYS_BASE, MEM_AREA_IO_SEC);
    vaddr_t tsi_base = core_mmu_get_va(TSI_BASE, MEM_AREA_IO_SEC);
    int ret;

    /* Check types of parameters*/
    if (types != TEE_PARAM_TYPES(TEE_PARAM_TYPE_VALUE_INPUT,
                    TEE_PARAM_TYPE_VALUE_OUTPUT,
                     TEE_PARAM_TYPE_NONE,
                     TEE_PARAM_TYPE_NONE)) {
        EMSG("bad parameters types: 0x%" PRIx32, types);
        return TEE_ERROR_BAD_PARAMETERS;
    }


    /* Check TSI is enable and do monitor set */
    if (!(io_read32(sys_base + SYS_CHIPCFG) & TSIEN)) {
        ret = TSI_Monitor_Status(params[0].value.a,       /* Monitor ID */
            &params[1].value.a);                          /* Status return*/
        if (ret != ST_SUCCESS) {
              EMSG("TSI Monitor Status fail");
            return ret;
        }
        return TEE_SUCCESS;
    }


    return 0x00000002;
}
```

```
/*
 * Send monitor close command to TSI
 */
static TEE_Result tsi_monitor_close(uint32_t types,
                    TEE_Param params[TEE_NUM_PARAMS])
{
    vaddr_t sys_base = core_mmu_get_va(SYS_BASE, MEM_AREA_IO_SEC);
    vaddr_t tsi_base = core_mmu_get_va(TSI_BASE, MEM_AREA_IO_SEC);
    int ret;

    /* Check types of parameters */
    if (types != TEE_PARAM_TYPES(TEE_PARAM_TYPE_VALUE_INPUT,
                   TEE_PARAM_TYPE_VALUE_OUTPUT,
                    TEE_PARAM_TYPE_NONE,
                    TEE_PARAM_TYPE_NONE)) {
        EMSG("bad parameters types: 0x%" PRIx32, types);
        return TEE_ERROR_BAD_PARAMETERS;
    }

    /* Close monitor */
    if (!(io_read32(sys_base + SYS_CHIPCFG) & TSIEN)) {
        ret = TSI_Monitor_Close(params[0].value.a,       /* Monitor ID */
            &params[1].value.a);                         /* Status return*/
        if (ret != ST_SUCCESS) {
              EMSG("TSI Monitor Close fail");
            return ret;
        }
        return TEE_SUCCESS;
    }


    return 0x00000002;
}



static TEE_Result pta_tsi_monitor_invoke_command(void *pSessionContext __unused,
              uint32_t nCommandID, uint32_t nParamTypes,
              TEE_Param pParams[TEE_NUM_PARAMS])
{
    switch (nCommandID) {
    case TA_TSI_INIT:
            return tsi_init();
    case TA_TSI_MONITOR_SET:
            return tsi_monitor_set(nParamTypes, pParams);
    case TA_TSI_MONITOR_STATUS:
            return tsi_monitor_status(nParamTypes, pParams);
    case TA_TSI_MONITOR_CLOSE:
            return tsi_monitor_close(nParamTypes, pParams);

    }

    return TEE_ERROR_NOT_IMPLEMENTED;
}

pseudo_ta_register(.uuid = PTA_TSI_MONITOR_UUID, .name = PTA_NAME,
```

```
        .flags = PTA_DEFAULT_FLAGS | TA_FLAG_DEVICE_ENUM,
        .open_session_entry_point = pta_tsi_monitor_open_session,
        .close_session_entry_point = pta_tsi_monitor_close_session,
        .invoke_command_entry_point = pta_tsi_monitor_invoke_command);
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - Linux-5.10.x

3.2 Hardware Requirements

- Circuit components
  - NuMaker-HMI-MA35D1 V1.0

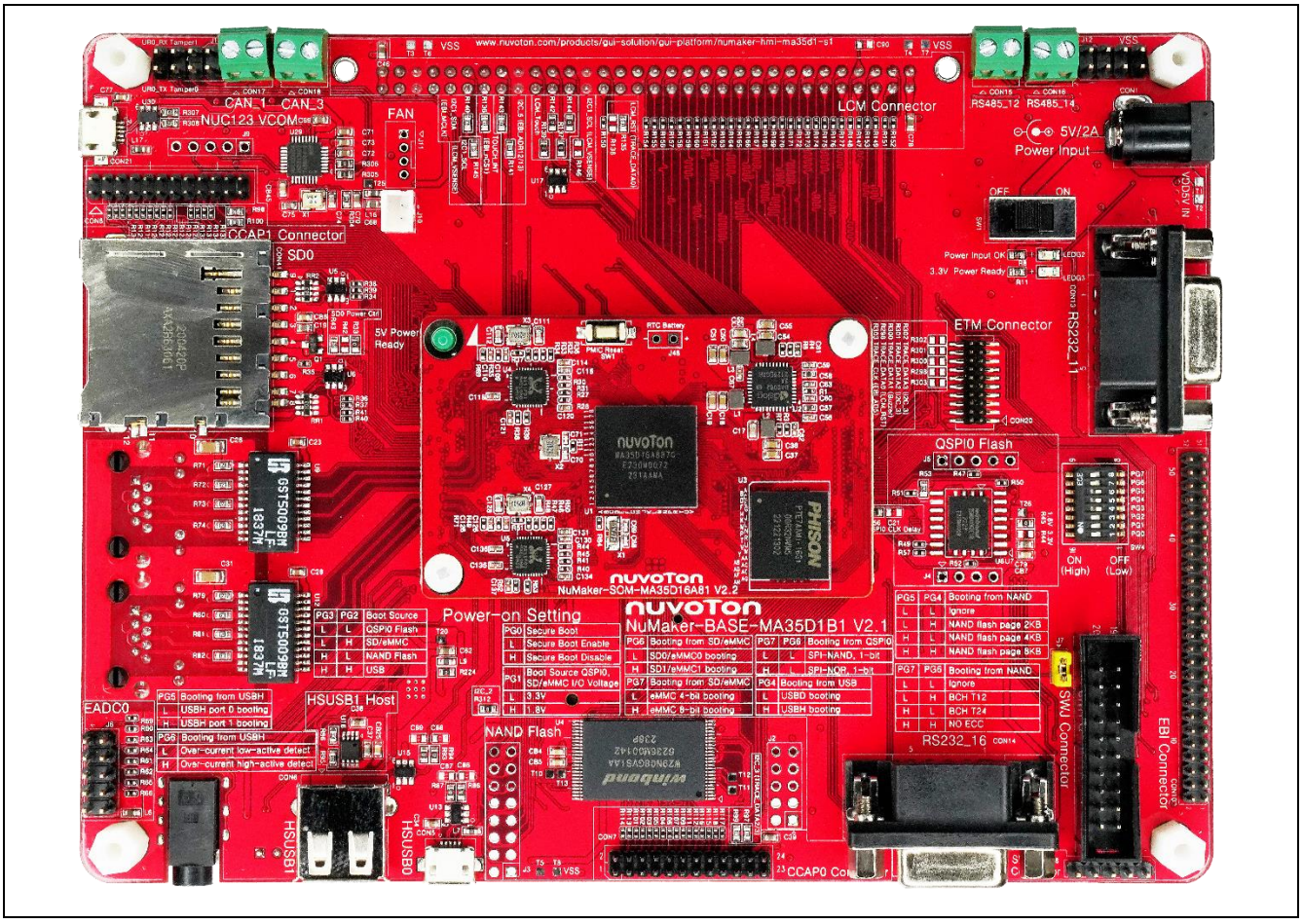


Figure 3-1 NuMaker-HMI-MA35D1 V1.0

4. Directory Information

The directory structure is shown below.

EC_MA35D1_TSI_Monitor_through_OP-TEE_V1.00	
optee_os_ma35d1	Trusted application header and source files
optee_example	Client application header and source files

Figure 4-1 Directory Structure

5. Example Code Execution

For the MA35D1 platform, the default image includes both optee-os-ma35d1 and optee-examples. To successfully run the example, you need to modify the relevant source code in optee-os-ma35d1 and add the Linux-side application under optee-examples. Below are the steps for Yocto:

1. Enter the MA35D1 Yocto project path (which may vary based on individual settings): *user@662f85abcbe8:~/shared/NuMaker-SOM-MA35D1/yocto/build$* .
2. Enter:

```
devtool modify optee-os-ma35d1
```

3. Enter:

```
devtool modify optee-examples
```

4. Confirm that the path: *NuMaker-SOM-MA35D1/yocto/build/workspace/sources* contains the folders optee-examples and optee-os-ma35d1.
5. Enter the following path in the optee-os-ma35d1 folder: *NuMaker-SOM-MA35D1/yocto/build/workspace/sources/optee-os-ma35d1/core/arch/arm/plat-nuvoton*. Replace the corresponding files in the original path with tsi_cmd.c and tsi_cmd.h from the optee_os folder in the example code folder, as well as TSI_Monitor_pta.c and TSI_Monitor_ta.h from the pta folder.
6. Open sub.mk in the pta folder and enter the following command in the end:

```
srcs-y += TSI_Monitor_pta.c
```

7. Rebuild optee-os-ma35d1 by entering the following command:

```
bitbake optee-os-ma35d1
```

8. Confirm that the compilation is successful.

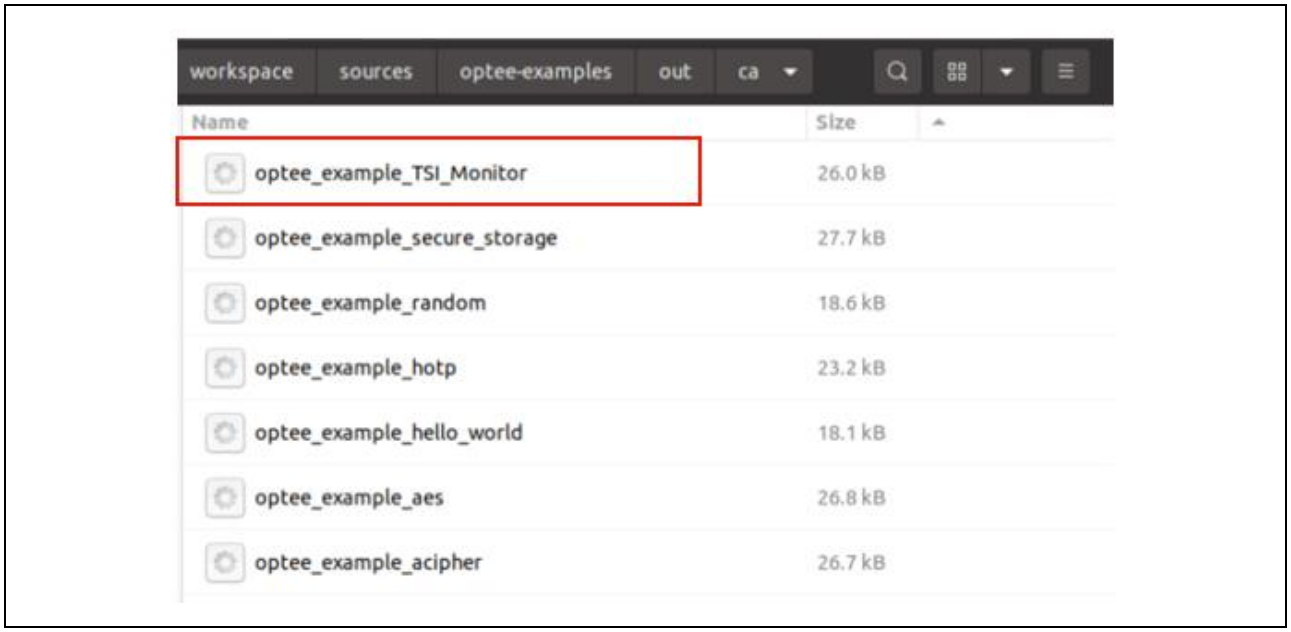
```
user@662f85abcbe8:~/shared/NuMaker-SOM-MA35D1/yocto/build$ bitbake optee-os-ma35d1
Loading cache: 100% |#########################################################################################################
Loaded 3521 entries from dependency cache.
Parsing recipes: 100% |#######################################################################################################
Parsing of 2384 .bb files complete (2382 cached, 2 parsed). 3523 targets, 405 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION           = "1.46.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "ubuntu-20.04"
TARGET_SYS           = "aarch64-poky-linux"
MACHINE              = "numaker-som-ma35d16a81"
DISTRO               = "nvt-ma35d1-directfb"
DISTRO_VERSION       = "5.5-dunfell"
TUNE_FEATURES        = "aarch64 cortexa35 crc crypto"
TARGET_FPU           = ""
meta
meta-poky            = "HEAD:251639560dd29bd2da5e694c1d5958ca26b12d0d"
meta-oe
meta-multimedia
meta-python
meta-networking
meta-filesystems
meta-gnome           = "HEAD:430ef96fe65f62d8da995f446d5b9b093544f031"
meta-qt5             = "HEAD:b4d24d70aca75791902df5cd59a4f4a54aa4a125"
meta-ma35d1          = "HEAD:cdd0aa40ccfe84051f66c32f1d849d0331017d64"
workspace            = "<unknown>:<unknown>"
meta-virtualization  = "HEAD:89abc62b47f6f20db9d00a8ec9b2c1b6b60ac3f9"

Initialising tasks: 100% |####################################################################################################
Sstate summary: Wanted 0 Found 0 Missed 0 Current 78 (0% match, 100% complete)
NOTE: Executing Tasks
NOTE: Tasks Summary: Attempted 540 tasks of which 540 didn't need to be rerun and all succeeded.
user@662f85abcbe8:~/shared/NuMaker-SOM-MA35D1/yocto/build$
```

9. Enter the following path in the optee-examples folder: *NuMaker-SOM-MA35D1/yocto/build/workspace/sources/examples* and copy the TSI_Monitor example to this path.

10. Rebuild optee-example by entering the following command:

```
bitbake optee-examples
```

11. Confirm that the compilation is successful and that the example optee_example_TSI_Monitor has been generated in the ca folder.



12. Build MA35D1 image by entering the following command:

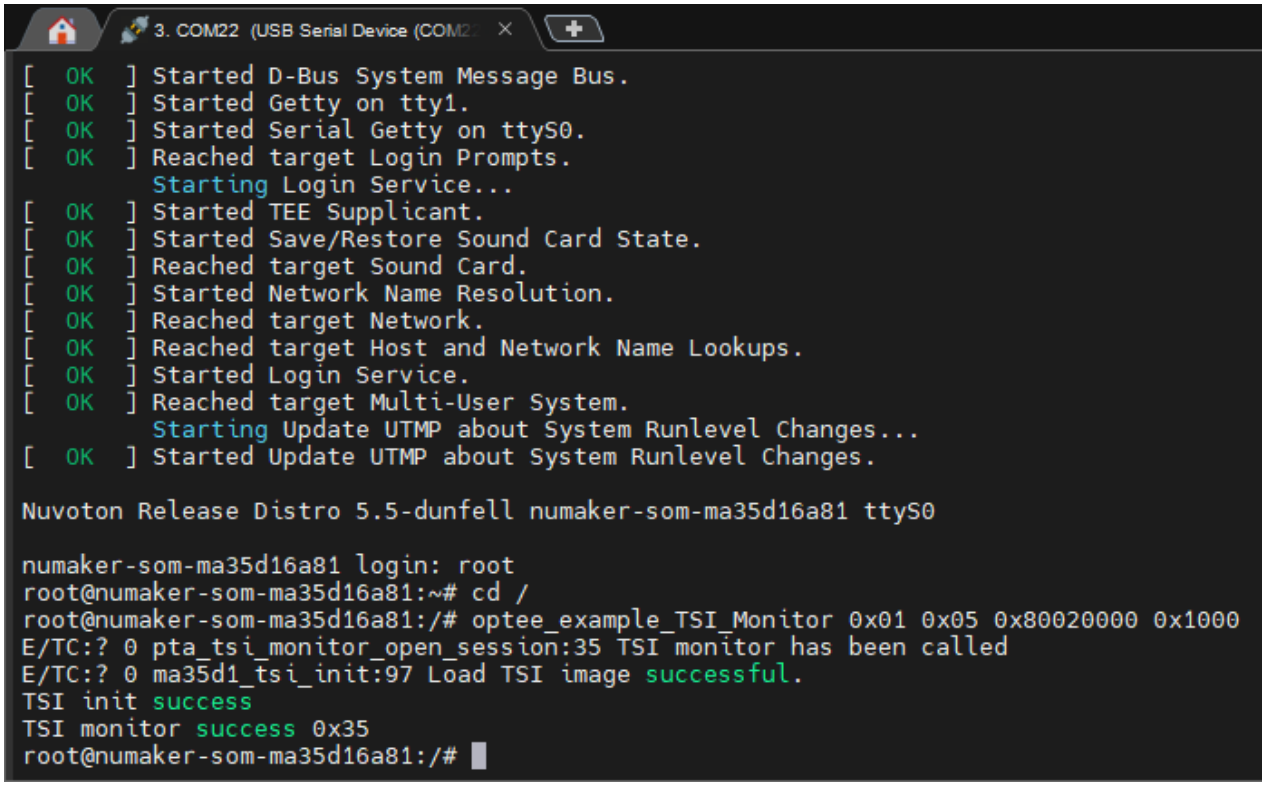
```
bitbake core-image-minimal
```

13. After the compilation is complete, program the image of core-image-minimal-numaker-som-ma35d16a81-202xxxxxxxxxx.rootfs.sdcard to the SD card used for booting.

14. Set the Power-on setting switch on the evaluation board to boot from SD0 then power on the MA35D1 evaluation board. The following demonstrates the operation of the secure monitor. First, configure the monitor region by entering the command: optee_example_TSI_Monitor 0x01 0x05 0x80020000 0x1000. Each parameter represents:

    - TSI_Monitor_Set command (0x01)
    - Monitor time interval (0x05)
    - Monitor base address (0x80020000)
    - Monitor size (0x1000)

15. After successful configuration, TSI will return a status code of 0x35, which serves as the identifier for that monitoring area.



16. Next, attempt to request TSI to report the status of the monitored region by entering the command: optee_example_TSI_Monitor 0x02 0x35, where each parameter represents:

- TSI_Monitor_Status command (0x02)
- Monitor ID (0x35)

If the setup is successful, TSI will return a status code (0x1430035), where 143 corresponds to 0x0143, indicating that the command entered is TSI_Monitor_Status. The 00 indicates that there are no abnormal statuses, and 35 corresponds to the ID for the monitored area.

```
Nuvoton Release Distro 5.5-dunfell numaker-som-ma35d16a81 ttyS0

numaker-som-ma35d16a81 login: root
root@numaker-som-ma35d16a81:~# cd /
root@numaker-som-ma35d16a81:/# optee_example_TSI_Monitor 0x01 0x05 0x80020000 0x1000
E/TC:? 0 pta_tsi_monitor_open_session:35 TSI monitor has been called
E/TC:? 0 ma35d1_tsi_init:97 Load TSI image successful.
TSI init success
TSI monitor success 0x35
root@numaker-som-ma35d16a81:/# optee_example_TSI_Monitor 0x02 0x35
E/TC:? 0 pta_tsi_monitor_open_session:35 TSI monitor has been called
TSI monitor status 0x1430035
root@numaker-som-ma35d16a81:/#
```

17. Use devmem2 to attempt to modify the memory value in the monitored area. First, enter: *devmem2 0x80020000*, to check the current data then input: *devmem2 0x80020000 w 0x12345678*, to modify the value at the address 0x80020000 to 0x12345678. Afterward, request TSI to report the status of the monitored area again. You will receive 0x1434335, where the error code 0x43 indicates that the data in the monitored memory area has been detected as modified.

```
root@numaker-som-ma35d16a81:/# optee_example_TSI_Monitor 0x02 0x35
E/TC:? 0 pta_tsi_monitor_open_session:35 TSI monitor has been called
TSI monitor status 0x1430035
root@numaker-som-ma35d16a81:/# devmem2 0x80020000
/dev/mem opened.
Memory mapped at address 0xffff8eff2000.
Read at address  0x80020000 (0xffff8eff2000): 0xAAAAAAAA
root@numaker-som-ma35d16a81:/# devmem2 0x80020000 w 0x12345678
/dev/mem opened.
Memory mapped at address 0xffffa0ae1000.
Read at address  0x80020000 (0xffffa0ae1000): 0xAAAAAAAA
Write at address 0x80020000 (0xffffa0ae1000): 0x12345678, readback 0x12345678
root@numaker-som-ma35d16a81:/# optee_example_TSI_Monitor 0x02 0x35
E/TC:? 0 pta_tsi_monitor_open_session:35 TSI monitor has been called
E/TC:? 0 tsi_monitor_status:136 TSI Monitor Status fail
TSI monitor status 0x1434335
root@numaker-som-ma35d16a81:/#
```

18. When TSI monitoring is no longer needed, enter the following command to disable the monitoring function: optee_example_TSI_Monitor 0x03 0x35, where each parameter represents:

- TSI_Monitor_Close command (0x03)
- Monitor ID (0x35)

When the TSI closes successfully, it will return a status code of 0x1440035, where 144 corresponds to 0x0144, indicating that the command executed is TSI_Monitor_Close, 00 indicates 0x00, signifying that the closure is successful, and 35 represents the ID corresponding to the area that has been closed.

```
root@numaker-som-ma35d16a81:/# optee_example_TSI_Monitor 0x03 0x35
E/TC:? 0 pta_tsi_monitor_open_session:35 TSI monitor has been called
TSI Close status 0x1440035
root@numaker-som-ma35d16a81:/#
```

19. When the number of monitor regions set exceeds four, an error code 0x41 (All monitors are used) will be received in response.

```
numaker-som-ma35d16a81 login: root
root@numaker-som-ma35d16a81:~# cd /
root@numaker-som-ma35d16a81:/# optee_example_TSI_Monitor 0x01 0x05 0x80020000 0x1000
E/TC:? 0 pta_tsi_monitor_open_session:35 TSI monitor has been called
E/TC:? 0 ma35d1_tsi_init:97 Load TSI image successful.
TSI init success
TSI monitor success 0x35
root@numaker-som-ma35d16a81:/# optee_example_TSI_Monitor 0x01 0x05 0x80030000 0x1000
E/TC:? 0 pta_tsi_monitor_open_session:35 TSI monitor has been called
E/TC:? 0 ma35d1_tsi_init:97 Load TSI image successful.
TSI init success
TSI monitor success 0x36
root@numaker-som-ma35d16a81:/# optee_example_TSI_Monitor 0x01 0x05 0x80040000 0x1000
E/TC:? 0 pta_tsi_monitor_open_session:35 TSI monitor has been called
E/TC:? 0 ma35d1_tsi_init:97 Load TSI image successful.
TSI init success
TSI monitor success 0x37
root@numaker-som-ma35d16a81:/# optee_example_TSI_Monitor 0x01 0x05 0x80050000 0x1000
E/TC:? 0 pta_tsi_monitor_open_session:35 TSI monitor has been called
E/TC:? 0 ma35d1_tsi_init:97 Load TSI image successful.
TSI init success
TSI monitor success 0x38
root@numaker-som-ma35d16a81:/# optee_example_TSI_Monitor 0x01 0x05 0x80060000 0x1000
E/TC:? 0 pta_tsi_monitor_open_session:35 TSI monitor has been called
E/TC:? 0 ma35d1_tsi_init:97 Load TSI image successful.
TSI init successE/TC:? 0 tsi_monitor_set:99 TSI Monitor Set failed! 0x41

optee_example_TSI_Monitor: TEEC_InvokeCommand failed with code 0x41 origin 0x4
root@numaker-som-ma35d16a81:/#
```

20. When an incorrect monitor ID is entered, an error code 0x42 (Invalid monitor ID) will be received in response.

```
root@numaker-som-ma35d16a81:/# optee_example_TSI_Monitor 0x03 0x41
E/TC:? 0 pta_tsi_monitor_open_session:35 TSI monitor has been called
E/TC:? 0 tsi_monitor_close:170 TSI Monitor Close fail
optee_example_TSI_Monitor: TEEC_InvokeCommand failed with code 0x42 origin 0x4
root@numaker-som-ma35d16a81:/#
```

6. Revision History

Date	Revision	Description
2024.09.24	1.00	Initial version.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.