AN12448

EdgeLock^{TM} SE05x Plug & Trust Middleware porting guidelines

Rev. 1.1 — 7 December 2020 Application note

546711

Document information

Information	Content
Keywords	EdgeLock SE05x, secure element, middleware, porting
Abstract	This document provides guidelines to port the EdgeLock SE05x Plug & Trust Middleware to your host MCU/MPU. It details the layers and software components that must be adapted to use the EdgeLock SE05x Plug & Trust Middleware in your host platform and host operating system.

Revision history

Revision history

Revision number	Date	Description
1.0	2019-07-17	First release
1.1	2020-01-20	Updated product name and other minor corrections.
1.2	2020-12-04	Updated template and product name

1 EdgeLock SE05x Plug & Trust Middleware architecture

The EdgeLock SE05x Plug & Trust Middleware translates function calls into APDUs that are transferred through T=1 protocol over an I^{2}C interface to the EdgeLock SE05x security IC. The EdgeLock SE05x executes the different APDUs and gives back the results to the EdgeLock SE05x Plug & Trust Middleware through the same interface. The complete set of EdgeLock SE05x Plug & Trust Middleware functions can be called from communication stacks like TLS or an application running on the host. Therefore, the EdgeLock SE05x Plug & Trust Middleware behaves as the interface between a host application and the EdgeLock SE05x security IC. Figure 1 gives a simplified view of the generic architecture of a system using EdgeLock SE05x Plug & Trust Middleware.

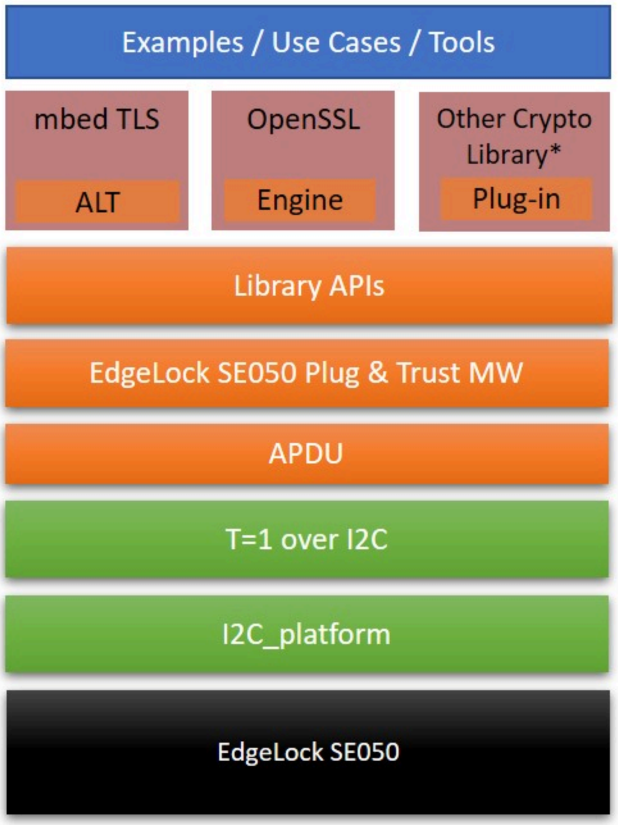

Figure 1. Simplified EdgeLock SE05x Plug & Trust Middleware architecture

1.1 Platform drivers

The `<i2c_platform>` layer depicted in Figure 1 acts as a link between the T=1 protocol and I^{2}C platform infrastructure. It is platform specific and must be adapted to your host platform. The EdgeLock SE05x Plug & Trust Middleware provides the `<i2c_platform>` layer adapted for various NXP MCU / MPU platforms and can be ported to multiple host platforms and host operating systems if these files are modified accordingly:

- **`i2c_<platform>.c`**: Platform specific I^{2}C code. The file `i2c_a7.c` is in the `/hostLib/platform/imx/` directory and the file `i2c_<platform>.c` is in the `/hostLib/platform/ksdk/` directory. As of MW v02.09.01, the following platforms are supported and have standalone files for I^{2}C implementation:
  - **frdm**: `i2c_frdm.c`
  - **imxrt10xx**: `i2c_imxrt10xx.c`
  - **lpc55sxx**: `i2c_lpc55sxx.c`
- **`sm_timer.c`**: This file defines the sleep functionality. It must be implemented according to the timers of the target platform. Located in the folder `/hostLib/platform/generic/`.
- **`timer_kinetis_<platform>.c`**: These files define the implementation of the sleep functions for different platforms. Located in the /`hostLib/platform/ksdk` directory.

There are other files that, if needed, can be optionally modified:

- **`ax_reset.c:`** This file defines low level reset functionality and DEEP Power-Down mode. If ENA pin is permanently pulled up, this file does not need to be implemented. If ENA pin is connected to the host, this file can be either kept as it is or re-implemented to match the user's needs. It is located in the `/hostLib/platform/ksdk` directory.
- **`se05x_reset.c`**: This file is the high level reset functionality to SE05x variants. Just like `ax_reset.c`, this is also an optional file depending on ENA pin. This file is located in directory `/hostLib/platform/ksdk` in case of Kinetis target, and `/hostLib/platform/imx` in case of Linux.
- **`sm_printf.c:`** Printf implementation. This is usually platform independent but in case the target platform does not support standard libraries or has special conditions, it needs to be adapted. It is located in the `/hostLib/platform/generic` directory.

Figure 2 shows the distribution of the files mentioned above. In the following sections, information about implementing and adapting these files is given.

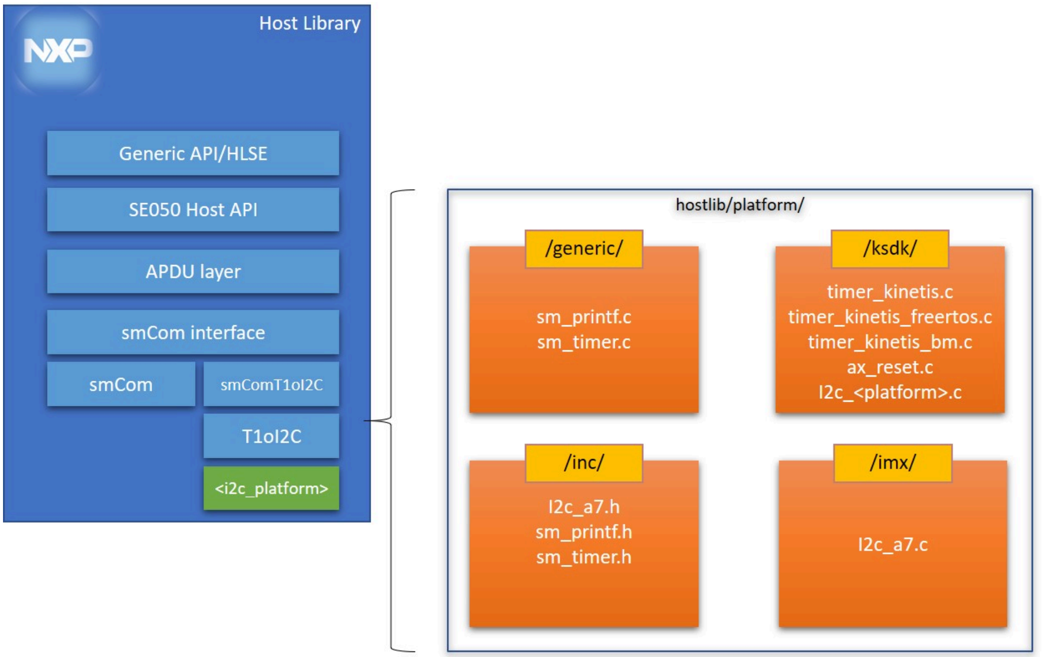

Figure 2. Folder structure and location of each of the files.

2 EdgeLock SE05x Plug & Trust Middleware porting

The main module to port in the EdgeLock SE05x Plug & Trust Middleware is I^{2}C. The host platform may either feature an I^{2}C-dedicated hardware or needs to implement a bit-banging mechanism. The I^{2}C driver does not need to support specific features, as communication with the secure element is based in T=1 protocol. Its implementation is written in pure ‘C’ code and should compile on any system with or without OS as it is platform independent. More details about T=1 over I^{2}C can be found in UM11225 - NXP SE05x T=1 Over I2C Specification.

However, as mentioned in Section 1, the user must adapt:

- The I^{2}C platform specific code.
- The timers.
- The reset (Optional: The reset default implementation is functional, but some targets may need extra work performed during the reset).
- sm_printf (Optional: The sm_printf is usually platform independent but there may be special target platforms that don’t support it and need to be adapted).

2.1 Adapting EdgeLock SE05x Plug & Trust Middleware to your Linux platform

This section describes how to port the EdgeLock SE05x Plug & Trust Middleware to your Linux environment. The EdgeLock SE05x Plug & Trust Middleware Linux integration in i.MX6UltraLite platform is used as a reference for this section.

2.1.1 Required header files to import

The location of the header to implement the I^{2}C driver is inside `platform/inc` folder of the EdgeLock SE05x Plug & Trust Middleware. The name of that file is `i2c_a7.h`. The contents of the file are depicted here.

Note: *Always read the documentation in this header file in case it is updated in the future. The different functions that must be implemented depen on the target platform. For your convenience, important sections in this header file have been highlighted in bold.*

```
/**
 * @file i2c_a7.h
 * @author NXP Semiconductors
 * @version 1.0
 * @par License
 * Copyright 2017 NXP
 *
 * This software is owned or controlled by NXP and may only be used
 * strictly in accordance with the applicable license terms.  By expressly
 * accepting such terms or by downloading, installing, activating and/or
 * otherwise using the software, you are agreeing that you have read, and
 * that you agree to comply with and are bound by, such license terms.  If
 * you do not agree to be bound by the applicable license terms, then you
 * may not retain, install, activate or otherwise use the software.
 *
 * @par Description
 *
 * I2C API used by SCI2C & T=1 over I2C protocol implementation.
 *
 * - SCIIC / SCI2C is the protocol used by A71CH / A71CL family of secure elements.
 *
 * - T=1 over I2C is the protocol used by SE050 family of secure elements.
 *
 * These APIs are to be implemented when porting the Middleware stack to a new
 * host platform.
 *
 * @note Few APIs are only required for the SCI2C protocol and few are only
 *       needed for T=1 over I2C Protocol.  They are marked by the defines
 *       ``I2C`` and  ``T1oI2C``
 *
 * # Convention of the APIs.
 *
 *
 * APIs for which a buffer is input. e.g.::
 *
 *   i2c_error_t axI2CWrite(unsigned char bus, unsigned char addr,
 *   unsigned char * pTx, unsigned short txLen);
 *
 *
 * In the above case :samp:`pTx` is a buffer input.  It is assumed that
 * the lengh as set in :samp:`txLen` is same as that pointed to by
 * :samp:`pTx`.  This parameter is used as is and any mistake by the
 * calling/implemented API will have unpredictable errors.
 *
 *
 * APIs for which a buffer is output. e.g.::
 *
 * i2c_error_t axI2CWriteRead(unsigned char bus,
 *     unsigned char addr,
 *     unsigned char *pTx,
 *     unsigned short txLen,
 *     unsigned char *pRx,
 *     unsigned short *pRxLen);
 *
 *
 * In the above case :samp:`pRx` is a buffer output and :samp:`pRxLen`
 * is both input and output. It is assumed that the lengh as set in
 * :samp:`pRxLen` is set to the maximum as available to the pointer
 * pointed by :samp:`pRx`.  This parameter is used as is and any mistake
 * by the calling/implemented API will have unpredictable errors.
 *
 * @par History
 *
 **/

#ifndef _I2C_A7_H
#define _I2C_A7_H

#include "sm_types.h"

#define SCI2C_T_CMDG 180 //!< Minimum delay between stop of Wakeup command and start of subsequent command (Value in micro seconds)

#define I2C_IDLE              0
#define I2C_STARTED           1
#define I2C_RESTARTED         2
#define I2C_REPEATED_START    3
#define DATA_ACK              4
#define DATA_NACK             5
#define I2C_BUSY              6
#define I2C_NO_DATA           7
#define I2C_NACK_ON_ADDRESS   8
#define I2C_NACK_ON_DATA      9
#define I2C_ARBITRATION_LOST  10
#define I2C_TIME_OUT          11
#define I2C_OK                12
#define I2C_FAILED            13

typedef unsigned int i2c_error_t;
#define I2C_BUS_0   (0)

/** Initialize the I2C platform HW/Driver*/

i2c_error_t axI2CInit( void );

/** Terminate / de-initialize the I2C platform HW/Driver
 *
 *
 * @param[in] mode Can be either 0 or 1.
 *
 *            Where applicable, and implemented a value of 0 corresponds
 *            to a 'light-weight' terminate.
 *
 *            In genral, this is not used for most of the porting
 *            platforms and use cases.
 *
 *
 */
void axI2CTerm(int mode);

#if AX_EMBEDDED
/** Smarter handling of back off logic
 *
 *  When we get a NAK from SE, we back off and keep on increasing the delay for next I2C Read/Write.
 *
 *  When we get an ACK from SE, we reset this back off delay.
 */
void axI2CResetBackoffDelay( void );
#endif /* FREEDOM */

#if defined(I2C) /* Means SCI2C SCIIC */ || defined(T1oI2C)
/** Write a frame.
 *
 * Needed for SCI2C and T=1 over I2C */
i2c_error_t axI2CWrite(unsigned char bus, unsigned char addr, unsigned char * pTx, unsigned short txLen);
#endif

#ifdef T1oI2C
/** Read a byte.
 *
 * Needed only for T=1 over I2C */
i2c_error_t axI2CRead(unsigned char bus, unsigned char addr, unsigned char * pRx, unsigned short rxLen);
#endif /* T1oI2C */

#endif // _I2C_A7_H
```

The I²C driver implementation in Linux is layered. The top-level layer is contained in a file called `i2c-dev.c` and is referred to as I²C device driver. The bottom layer deals with the specific hardware of the I²C controller, thus being specific for the target platform. It is called I²C bus driver.

The I²C device driver is defined in `i2c-dev.c` and `i2c.c` files. These files represent an I²C adapter implementing IOCTL functions (system call for device specific input/output operations and other operations which cannot be expressed by regular system calls). Therefore, it is required to include them in the implementation of the `i2c_a7.c` driver. As can be observed here, both `i2c_dev.h` and `i2c.h` header files have been included in `i2c_a7.c`.

```
#include "i2c_a7.h"
#include <stdio.h>
#include <string.h>

#include <fcntl.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <linux/i2c-dev.h>
      #include <linux/i2c.h>
      #include <time.h>
```

As a summary, the header `i2c_a7.h` provides the definitions of the I²C driver functions (in **bold**), while `i2c_dev.h` and `i2c.h` header files provide the definitions and implementation of the Linux native I²C device driver (in *italics*). It is not necessary to implement the functions of the I²C device driver as they are implemented natively by the Linux system.

The last part to be specified in `i2c_a7.c` is a reference to the specific I^{2}C master (device node) the EdgeLock SE05x is connected to. To do this, assign the correct device node to the variable “devName” as depicted here. In this case (iMX6UltraLite and OM-SE050ARD), the I^{2}C interface used is `i2c-1` but, depending on the manufacturer and the system specifications, the direction may be different (default values usually are “i2c-0” or “i2c-1”).

```
static int axSmDevice;
static int axSmDevice_addr = 0x48;      // 7-bit address
static char devName[] = "/dev/i2c-1";   // Change this when connecting to another host i2c master port
```

2.1.2 Adapting timers

The timers are defined by the system and are used to manage interrupts. The timers are usually natively implemented in Linux or RTOS platforms and defined by the manufacturer in bare metal systems. Figure 3 depicts the timers implementation architecture. For Linux, `sm_timer.h` defines three different functions to implement interruptions:

- `sm_initSleep` function to initialize the system tick counter.
- `sm_sleep` function to block the calling thread for a number of milliseconds.
- `sm_usleep` function to block the calling thread for `microsec` microseconds.

```
/* function used for delay loops */
uint32_t sm_initSleep(void);
void sm_sleep(uint32_t msec);
void sm_usleep(uint32_t microsec);
```

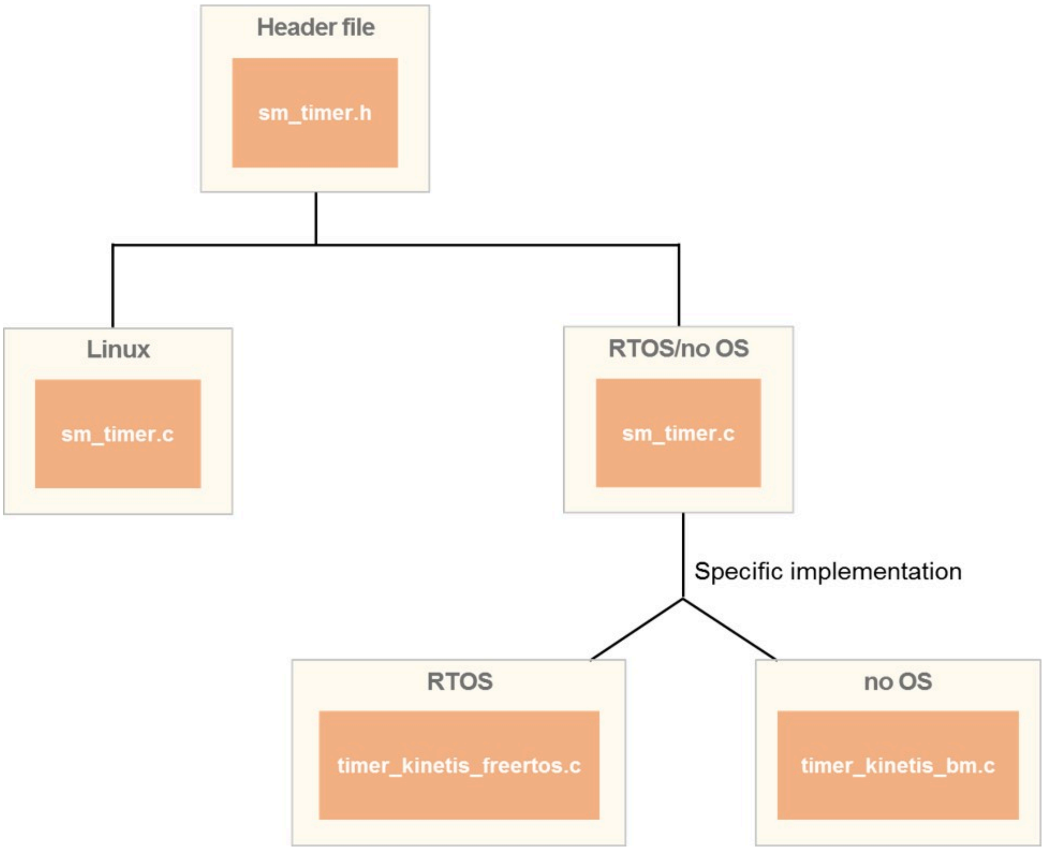

Figure 3. Sm_timer.c implementation in each target platform

2.2 Adapting EdgeLock SE05x Plug & Trust Middleware to your RTOS or bare metal platform

This section describes how to port the EdgeLock SE05x Plug & Trust Middleware to your RTOS or bare metal environment. The EdgeLock SE05x Plug & Trust Middleware RTOS integration in FRDM-K64F platform is used as a reference for this section.

2.2.1 Required header files to import

The location of the header to implement the I^{2}C driver is inside `platform/inc` folder of the EdgeLock SE05x Plug & Trust Middleware. The name of that file is `i2c_a7.h`. The contents of the file and the different functions that must be implemented are depicted here.

Note: *Always read the documentation in this header file in case it is updated in the future. The different functions that must be implemented depen on the target platform. For your convenience, important sections in this header file have been highlighted in bold.*

```
/**
 * @file i2c_a7.h
 * @author NXP Semiconductors
 * @version 1.0
 * @par License
  * Copyright 2017 NXP
  *
  * This software is owned or controlled by NXP and may only be used
  * strictly in accordance with the applicable license terms.  By expressly
  * accepting such terms or by downloading, installing, activating and/or
  * otherwise using the software, you are agreeing that you have read, and
  * that you agree to comply with and are bound by, such license terms.  If
  * you do not agree to be bound by the applicable license terms, then you
  * may not retain, install, activate or otherwise use the software.
  *
  * @par Description
  *
  * I2C API used by SCI2C & T=1 over I2C protocol implementation.
  *
  * - SCIIC / SCI2C is the protocol used by A71CH / A71CL family of secure elements.
  *
  * - T=1 over I2C is the protocol used by SE050 family of secure elements.
  *
  * These APIs are to be implemented when porting the Middleware stack to a new
  * host platform.
  *
  * @note Few APIs are only required for the SCI2C protocol and few are only
  *       needed for T=1 over I2C Protocol.  They are marked by the defines
  *       ``I2C`` and  ``T1oI2C``
  *
  * # Convention of the APIs.
  *
  *
  * APIs for which a buffer is input. e.g.::
  *
  *   i2c_error_t axI2CWrite(unsigned char bus, unsigned char addr,
  *   unsigned char * pTx, unsigned short txLen);
  *
  *
  * In the above case :samp:`pTx` is a buffer input.  It is assumed that
  * the lengh as set in :samp:`txLen` is same as that pointed to by
  * :samp:`pTx`.  This parameter is used as is and any mistake by the
  * calling/implemented API will have unpredictable errors.
  *
  *
  * APIs for which a buffer is output. e.g.::
  *
  * i2c_error_t axI2CWriteRead(unsigned char bus,
  *     unsigned char addr,
  *     unsigned char *pTx,
  *     unsigned short txLen,
  *     unsigned char *pRx,
  *     unsigned short *pRxLen);
  *
  *
  * In the above case :samp:`pRx` is a buffer output and :samp:`pRxLen`
  * is both input and output. It is assumed that the lengh as set in
  * :samp:`pRxLen` is set to the maximum as available to the pointer
  * pointed by :samp:`pRx`.  This parameter is used as is and any mistake
  * by the calling/implemented API will have unpredictable errors.
  *
  * @par History
  *
  **/

 #ifndef _I2C_A7_H
 #define _I2C_A7_H

 #include "sm_types.h"

 #define SCI2C_T_CMDG 180 //!< Minimum delay between stop of Wakeup command and start of subsequent command (Value in micro seconds)

 #define I2C_IDLE              0
 #define I2C_STARTED           1
 #define I2C_RESTARTED         2
 #define I2C_REPEATED_START    3
 #define DATA_ACK              4
 #define DATA_NACK             5
 #define I2C_BUSY              6
 #define I2C_NO_DATA           7
 #define I2C_NACK_ON_ADDRESS   8
 #define I2C_NACK_ON_DATA      9
 #define I2C_ARBITRATION_LOST  10
 #define I2C_TIME_OUT          11
 #define I2C_OK                12
 #define I2C_FAILED            13

 typedef unsigned int i2c_error_t;
 #define I2C_BUS_0   (0)

 /** Initialize the I2C platform HW/Driver*/

 i2c_error_t axI2CInit( void );

 /** Terminate / de-initialize the I2C platform HW/Driver
  *
  *
  * @param[in] mode Can be either 0 or 1.
  *
  *            Where applicable, and implemented a value of 0 corresponds
  *            to a 'light-weight' terminate.
  *
  *              In genral, this is not used for most of the porting
  *              platforms and use cases.
  *
  *
  */
  void axI2CTerm(int mode);

  #if AX_EMBEDDED
  /** Smarter handling of back off logic
  *
  *  When we get a NAK from SE, we back off and keep on increasing the delay for next I2C Read/Write.
  *
  *  When we get an ACK from SE, we reset this back off delay.
  */
  void axI2CResetBackoffDelay( void );
  #endif /* FREEDOM */

  #if defined(I2C) /* Means SCI2C SCIIC */ || defined(T1oI2C)
  /** Write a frame.
  *
  * Needed for SCI2C and T=1 over I2C */
  i2c_error_t axI2CWrite(unsigned char bus, unsigned char addr, unsigned char * pTx, unsigned short txLen);
  #endif

  #ifdef T1oI2C
  /** Read a byte.
  *
  * Needed only for T=1 over I2C */
  i2c_error_t axI2CRead(unsigned char bus, unsigned char addr, unsigned char * pRx, unsigned short rxLen);
  #endif /* T1oI2C */

  #endif // _I2C_A7_H
```

An RTOS operative system can manage and schedule events and manage interruptions (e.g. FreeRTOS). The necessary files to include in the implementation of the `i2c_a7.c` are shown here.

The header file `i2c_a7.h` provides the definitions of the functions to be implemented to port the EdgeLock SE05x Plug & Trust Middleware (in **bold**). In addition, the following files should be provided by the manufacturer (in *italics*):

- If the system is compiled as a bare metal platform, `fsl_i2c.h` file is used.
- If the system is compiled using FreeRTOS, `fsl_i2c_freertos.h` file is used.

```
#include "i2c_a7.h"
#include "fsl_clock.h"
#include "fsl_i2c.h"
#if defined(SDK_OS_FREE_RTOS) && SDK_OS_FREE_RTOS == 1
#include "fsl_i2c_freertos.h"
#endif
#include "fsl_port.h"
#include "sm_timer.h"
#include <stdio.h>
```

```
#include "fsl_gpio.h"
#include "sci2c_cfg.h"
```

2.2.2 Adapting timers

The timers are defined by the system and are used to manage interrupts. Figure 4 illustrates the example in this guide, using as target platform Linux, RTOS or bare metal. It is more straightforward to implement “sm_timer.c” in Linux or RTOS versions as those timers are usually implemented natively. In a bare metal system, the timers are defined by the MCU manufacturer.

The timers are defined by the system and are used to manage interrupts. The timers are usually natively implemented in Linux or RTOS platforms and defined by the manufacturer in bare metal systems. Figure 4 depicts the timers implementation architecture. For RTOS or bare metal, the EdgeLock SE05x Plug & Trust Middleware provides several files already adapted for the Kinetis family: `timer_kinetis.c` with common implementation, `timer_kinetis_freertos.c` for FreeRTOS and `timer_kinetis_bm.c` for bare metal.

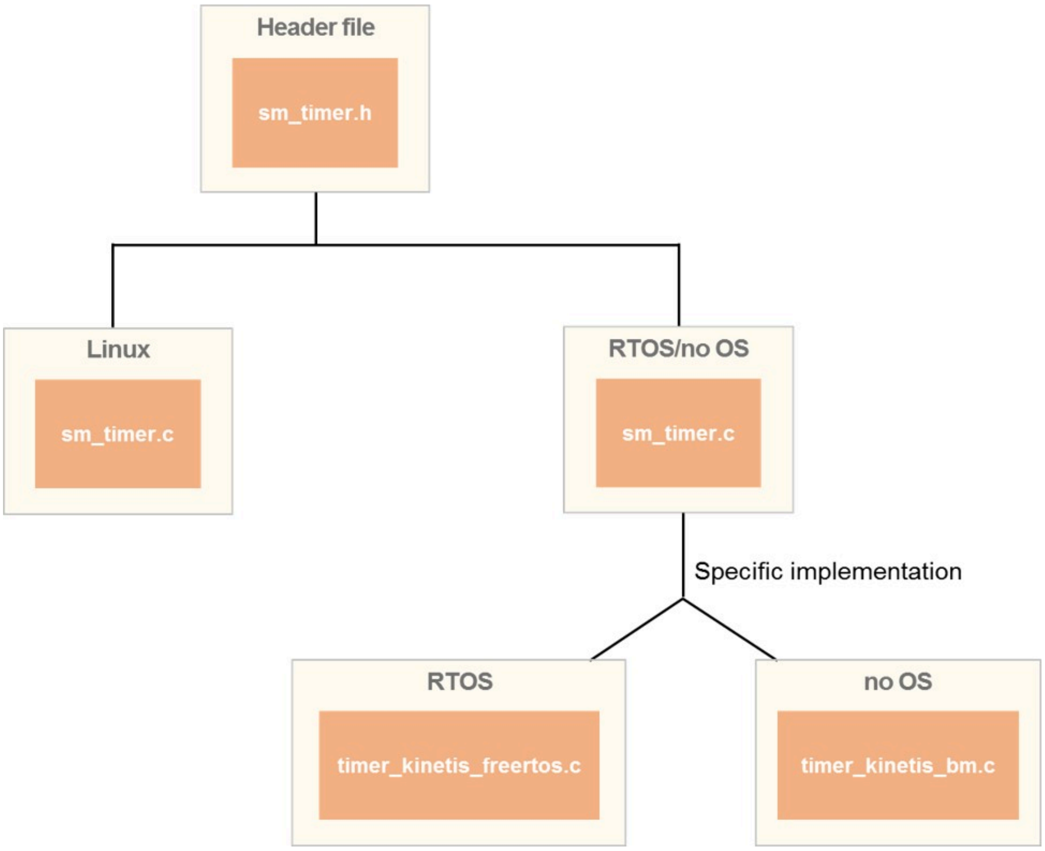

Figure 4. Sm_timer.c implementation in each target platform

The implementation of the `sm_unsleep()` function is shown here. As can be observed, the sleep time is defined by the macro `CORRECTION_TOLERENCE`. This macro depends on the compiler used and the core clock frequency of the MCU used.

```
void sm_usleep(uint32_t microsec) {
    gusleep_delay = microsec * CORRECTION_TOLERENCE;
    while (gusleep_delay-- ) {
        __NOP();
    }6 }
```

The current code uses microsecond level delays using software loop (in **bold**). Depending on the clock and compiler, a very fast host MCU could violate the protocol and result in a non-responsive system. If available, it is recommended to use hardware timers. However, a hardware clock may lead to resource conflicts for system integration and, hence, they are not used in this implementation. In addition, there are two possible sub-scenarios:

- **RTOS**: RTOS has its own timers defined. In FreeRTOS, these libraries are included in `sm_timer.c` . This time is defined by a number of ‘systicks’ (system ticks). In the current integration, one ‘systick’ is configured to last 1 millisecond. The FreeRTOS function “vTaskDelay” is employed to handle the interruption. As an example, the `sm_timer.c` implementation in a Kinetis board with FreeRTOS as shown here.

```
/* initializes the system tick counter
 * return 0 on succes, 1 on failure */
uint32_t sm_initSleep() {
    return 0;
}

/**
 * Implement a blocking (for the calling thread) wait for a number of milliseconds.
 */
void sm_sleep(uint32_t msec) {
    vTaskDelay(msec);
}

void vApplicationTickHook() {
    gtimer_kinetis_msticks++;22 }
```

- **Bare metal**: A bare metal system is not controlled by any operating system (no OS). The timers should be manually implemented according to the specifications of the MCU and the manufacturer. As an example, the implementation of the function `sm_sleep` in a Kinetis board (FRDM-K64F) without OS is shown here. The function highlighted in bold is implemented according to the board specifications of the target platform to implement the sleep functionality. In bare metal systems, the systick timer is used to be triggered every millisecond, and that is how the system keeps track of the delay.

```
static void systick_delay(const uint32_t delayTicks) {
    uint32_t currentTicks;
    assert(delayTicks < 0x7FFFFFFFu);

    __disable_irq();

    if ((gtimer_kinetis_msticks) & 0x80000000u)
    {
       /* gtimer_kinetis_msticks has increased drastically (MSB is set),
         * So, reset gtimer_kinetis_msticks before it's too late to detect an
         * overflow. */
        gtimer_kinetis_msticks = 0;
    }

    currentTicks = gtimer_kinetis_msticks;   // read current tick counter

    __DSB();
    __enable_irq();

    // Now loop until required number of ticks passes
    while ((gtimer_kinetis_msticks - currentTicks) <= delayTicks) {
#ifdef __WFI
        __WFI();
#endif
    }
}

/* interrupt handler for system ticks */
void SysTick_Handler(void) {
    gtimer_kinetis_msticks++;
}


/* initializes the system tick counter
 * return 0 on succes, 1 on failure */
uint32_t sm_initSleep() {
    gtimer_kinetis_msticks = 0;
    SysTick_Config(SystemCoreClock / 1000);
    __enable_irq();
    return 0;
}

/**
 * Implement a blocking (for the calling thread) wait for a number of milliseconds.
 */
void sm_sleep(uint32_t msec) {
    systick_delay(msec); }
```

2.3 Adapting EdgeLock SE05x Plug & Trust Middleware printing layer (sm_printf.h)

This module is used to print messages in a console in a debug mode. Usually, the printing layer is platform independent as it only needs two standard libraries. The implementation of “sm_printf” platform is shown here.

```
#include <stdio.h>
#include <stdarg.h>

#include "sm_printf.h"

```

```
#ifdef FREEDOM
#   include "fsl_device_registers.h"
#   include "fsl_debug_console.h"
#   include "board.h"
#else
#   define PRINTF printf
#endif

#define MAX_SER_BUF_SIZE  (1024)

void sm_printf(uint8_t dev, const char * format, ...)
{
    uint8_t  buffer[MAX_SER_BUF_SIZE + 1];
    va_list   vArgs;

    dev = dev; // avoids warning; dev can be used to determine output channel

    va_start(vArgs, format);
#ifdef _WIN32
    vsnprintf_s((char *)buffer, MAX_SER_BUF_SIZE, MAX_SER_BUF_SIZE, (char const *)format, vArgs);
#else
    vsnprintf((char *)buffer, MAX_SER_BUF_SIZE, (char const *)format, vArgs);
#endif
    va_end(vArgs);

    PRINTF("%s", buffer); }
```

The `stdarg.h` library provides the functions needed to use a `varArg` list. Note that Windows has its own implementation of `vsnprintf_s`.

The `stdio.h` library provides the **printf** function needed to print output data in a console. If the target platform does not support the `stdio.h` or `starg.h`, the developer should implement the `sm_printf` function according to the target platform specifications.

2.4 Adapting EdgeLock SE05x Plug & Trust Middleware reset module (ax_reset.c)

The EdgeLock SE05x Plug & Trust Middleware reset module handles resets on the EdgeLock SE05x. The reference implementation is already platform dependent, as can be seen here. If needed, the developer can extend the implementation with extra functionality for its host platform.

```
#include <board.h>
#include "ax_reset.h"

#include "fsl_gpio.h"
#include "sm_timer.h"
#include "sm_types.h"
#include "fsl_common.h"
#include "se05x_apis.h"
#include "se_reset_config.h"

/*
 * Where applicable, Configure the PINs on the Host
 *
 */
void axReset_HostConfigure()
{
#if defined(CPU_MIMXRT1052DVL6B)
    gpio_pin_config_t reset_pin_cfg = {kGPIO_DigitalOutput, 0, SE_RESET_LOGIC};
#else
    gpio_pin_config_t reset_pin_cfg = {kGPIO_DigitalOutput, SE_RESET_LOGIC};
#endif
#if defined(CPU_LPC55S69JBD100)
    GPIO_PortInit(GPIO, (uint32_t)SE05X_ENA_HOST_PORT);
    GPIO_PinInit(GPIO, (uint32_t)SE05X_ENA_HOST_PORT, SE05X_ENA_HOST_PIN, &reset_pin_cfg);
#else
    GPIO_PinInit(SE05X_ENA_HOST_PORT, SE05X_ENA_HOST_PIN, &reset_pin_cfg);
#endif
    return;
}

/*
 * Where applicable, PowerCycle the SE
 *
 * Pre-Requistie: @ref axReset_Configure has been called
 */
void axReset_ResetPluseDUT()
{
    axReset_PowerDown();
    sm_usleep(2000);
    axReset_PowerUp();
    return;
}

/*
 * Where applicable, put SE in low power/standby mode
 *
 * Pre-Requistie: @ref axReset_Configure has been called
 */
void axReset_PowerDown()
{
#if defined(CPU_LPC55S69JBD100)
    GPIO_PinWrite(GPIO, (uint32_t)SE05X_ENA_HOST_PORT, SE05X_ENA_HOST_PIN, !SE_RESET_LOGIC);
#else
    GPIO_PinWrite(SE05X_ENA_HOST_PORT, SE05X_ENA_HOST_PIN, !SE_RESET_LOGIC);
#endif
    return;
}

/*
 * Where applicable, put SE in powered/active mode
 *
 * Pre-Requistie: @ref axReset_Configure has been called
 */
void axReset_PowerUp()
{
#if defined(CPU_LPC55S69JBD100)
```

```
    GPIO_PinWrite(GPIO, (uint32_t)SE05X_ENA_HOST_PORT,
 SE05X_ENA_HOST_PIN, SE_RESET_LOGIC);
#else
    GPIO_PinWrite(SE05X_ENA_HOST_PORT, SE05X_ENA_HOST_PIN,
 SE_RESET_LOGIC);
#endif
    return;
}
```

The developer should extend the implementation of `ax_reset.c` function according to the target platform specifications.

3 Legal information

3.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

3.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors. In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory. Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification. Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products. NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer. In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages. Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

3.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

Figures

Contents

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 7 December 2020
Document identifier: AN12448
Document number: 546711