AN10993

Pegoda Software Design Guide

Rev. 1.0 — 21 March 2011
196110

Application note
COMPANY PUBLIC

Document information

Info	Content
Keywords	Pegoda software design guide, firmware documentation, software documentation, firmware update, Pegoda firmware architecture, Pegoda functional overview, PC/SC mode
Abstract	This document provides a comprehensive architectural overview of the Pegoda software.

Revision history

Rev	Date	Description
1.0	20110321	First release
	20101101	Draft version

Contact information

For additional information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

1.1 Scope

The scope of this document covers the software architecture coming with the Pegoda package. This includes a description of the integrated firmware stack on the LPC1768 controller as well as a high level overview of available modes in RD710/RX852.

The Basic Function Library (BFL) architecture and modules are covered in a separate document. Examples on how to setup the BFL stack on PC environment to communicate in different SAM modes and card products are covered in the Example Project Document [5].

The firmware/BFL may be changed individually to its own needs. An adequate tool chain for loading the firmware project file and doing further programming/debugging is introduced in the documents [7].

Updates for future firmware releases may be provided by NXP and can be easily downloaded by USB.

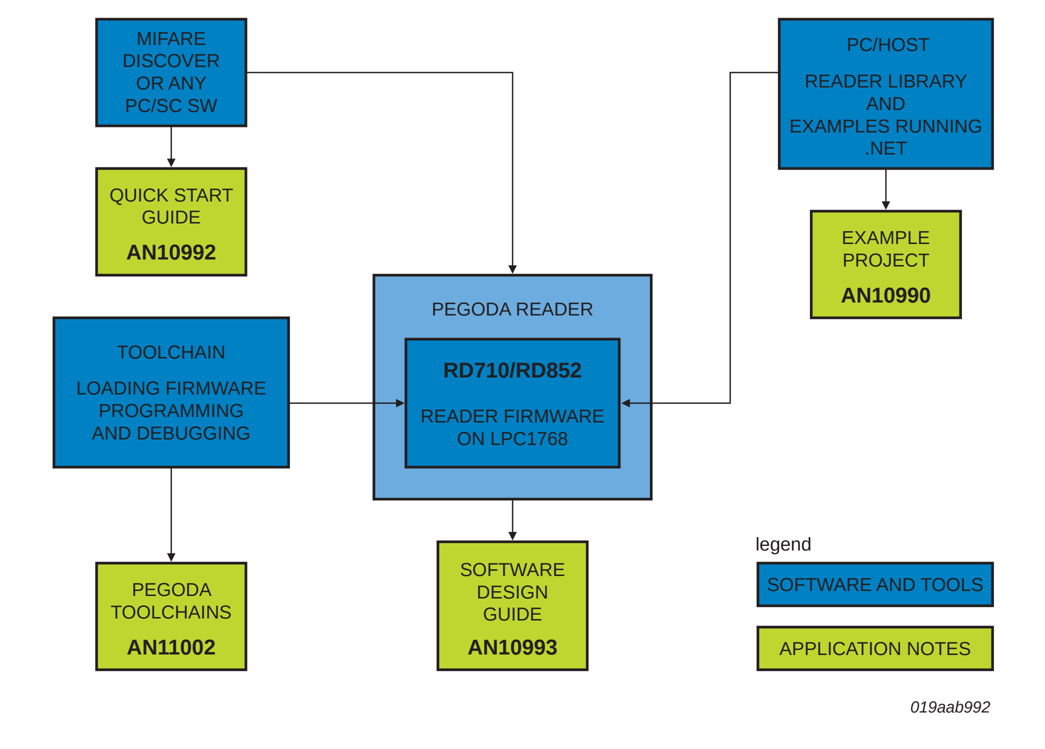

MIFARE DISCOVER OR ANY PC/SC SW
QUICK START GUIDE AN10992
PC/HOST READER LIBRARY AND EXAMPLES RUNNING .NET
EXAMPLE PROJECT AN10990
PEGODA READER
RD710/RD852 READER FIRMWARE ON LPC1768
TOOLCHAIN LOADING FIRMWARE PROGRAMMING AND DEBUGGING
PEGODA TOOLCHAINS AN11002
SOFTWARE DESIGN GUIDE AN10993
legend
SOFTWARE AND TOOLS
APPLICATION NOTES
019aab992

Fig 1. Flowchart

1.2 Audience

This document is intended for use by manufacturers wanting to develop applications based on the software stack delivered within the Pegoda package.

Note that understanding parts of this document requires knowledge on microcontroller and their underlying systems.

1.3 Applicable documents or references

[1] [ISO/IEC 14443]

[2] PC/SC Workgroup Specifications (www.nxp.com/redirect/pcscworkgroup.com/specifications/specdownload)

[3] onARM: see www.nxp.com/redirect/onarm for more information

[4] See www.nxp.com/redirect/freetos for more information

[5] www.nxp.com/redirect/en.wikipedia.org/wiki/Interrupt_handler

[6] AN10990 Example Project Pegoda

[7] ANxxxx ISP Programming Toolchain

[8] www.nxp.com/redirect/ics.nxp.com/products/lpc1000/all/~LPC1768 or on CD

For information on availability of samples as well as documentation, please refer to the application note ‘Pegoda EV710/EV852 Documentation and Sampling guide’.

1.4 Acronyms and abbreviations

SAM	Secure Access Module
S	SAM in S-Mode
N	no SAM
X	SAM in X-Mode
BFL	Basic Function Library

2. Functional overview and mode of operation

The reader firmware can perform contact and contactless communication. The contact part is used to communicate with a SAM. The contactless part performs the polling and activation sequence according to PC/SC part 2, ISO/IEC 14443-3 and 14443-4 standards.

The polling and activation sequence can be turned off and on with the use of escape (PC/SC part 3) commands. The polling and activation sequence is automatically disabled if escape commands are executed that would interfere with the normal operations. The firmware also interprets the ADPUs for MIFARE which are defined in PC/SC part 3.

The main modes of operation are being set by DIP switches which are located on the reader. Some configuration parameters can be set with the escape commands. The reader will store them in non-violate memory. The user will be able to reset the configuration back to default state.

The firmware will construct the product name – which is returned by USB descriptor – to easily identify the reader/SAM configuration. Depending on the DIP switches, there are three possible configurations

1. No SAM (Pegoda 2 N)
2. SAM in X-Mode (Pegoda 2 X)
3. SAM in S-Mode (Pegoda 2 S)

The user will be able to flash the board with custom or original firmware with three methods:

1. Over USB (IAP)
2. Over serial port (ISP)
3. Over JTAG with the use of external program

The Pegoda reader provides four modes of operation:

- Demo Mode
- PCSC Mode
- Overwrite Configuration Mode
- Enter Secondary Bootloader Mode

2.1 Demo mode

The demo mode is used to showcase in an autonomous way the functionality of the reader. In this mode, ISO14443-3A activation loop is performed and based on ***SAK*** blink the LEDs according to the next table:

Table 1. Card type according to SAK and number of beep

Card type	sound
MIFARE 1K (0x08)	1
MIFARE Classic 4K (0x18)	2

Card type	sound
MIFARE Ultralight (0x00)	3
MIFARE DESFire	4
MIFARE Plus	5

The demo mode can run in all three operation modes (non SAM, SAM in S, SAM in X mode).

The demo mode is implemented in ***src/p2_fw_demo_mode.c*** in the task:

```
    void p2_fw_task_demo_mode (void * param);
```

it distinguishes between BFL and SAM driven demo.

The BFL mode is implemented in

```
    static void p2_fw_task_demo_mode_bfl (void * param);
```

for the usage of the BFL library.

The SAM mode is implemented in

```
    static void p2_fw_task_demo_mode_sam (void * param);
```

for the usage of SAM APDUs and T=1 protocol.

2.2 PC/SC Mode

In this mode the reader acts as a fully compliant PC/SC reader. The PC/SC mode provides two sub modes called

- Standard mode
- Direct mode.

The standard mode provides routines and commands according to the PC/SC standard. The user can also directly control the reader (e.g. manual activation of cards, settings IC registers) by using the direct mode.

The mode is split into two tasks:

```
    void p2_fw_task_pcsc_execute (void * param);
```

which takes care of scheduling the bottom halfs of interrupt functions and

```
    void p2_fw_task_pcsc_poll_and_act_loop (void * param);
```

which performs activation and pooling of cards.

There are three types of operating modes within the PC/SC mode:

- No SAM mode
- SAM in non X-mode
- SAM in X-Mode

2.2.1 No SAM mode

The most important aspect of this mode is performing activation and polling sequence for ISO 14443 A type cards as defined in PC/SC part 3. Selected cards are put in slot manager and notification is sent to PC/SC driver. There can be only one ISO 14443-3 card or multiple (maximum 14) ISO 14443-4 cards in the field.

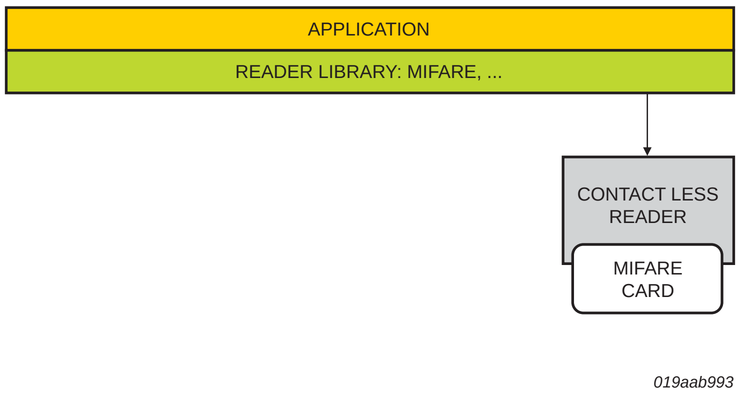


Fig 2. No SAM configuration

2.2.2 SAM in non X-mode

SAM in non X-mode is similar as 2.2.1 No SAM mode. The only difference is that slot 0 is always occupied with SAM, which can be used as key store, cryptographic engine, etc.

The reader chip can not be controlled by SAM in this mode.

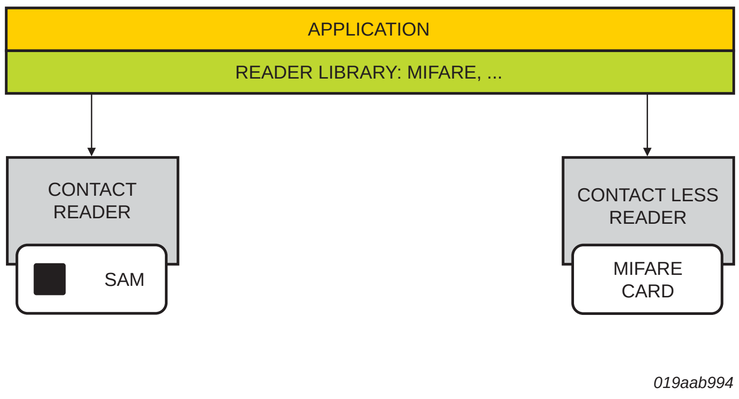


Fig 3. SAM in non X configuration

2.2.3 SAM in X-mode

In this mode communication is only through SAM (slot 0). Only limited number of proprietary commands can be executed.

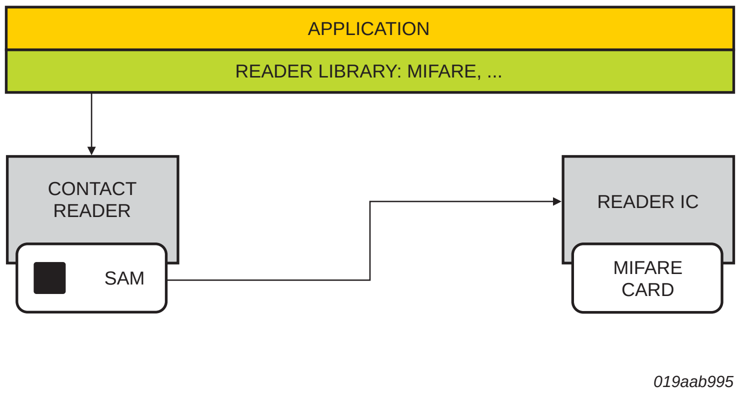


Fig 4. SAM in X-mode configuration

2.2.4 The Interrupt Driven Part

Interrupts from external interfaces are serviced by interrupt services which mainly read-in data sent from external device to the reader. After the data is read-in to the communication buffer we call:

```
void p2_fw_ccid_top_half_dispatch(void);
```

in ***src/p2_fw_ccid.c*** which acts as a dispatcher for the CCID messages. For serial and ethernet messages (if selected interface is other then USB then the reader goes in to direct mode) the interrupt service always provides a CCID XFER Block command.

In the dispatcher we check if the requested command is supported and allowed and then call one of the dedicated functions:

- for CCID ICC_POWER_ON we call
  ```
  void p2_fw_ccid_icc_power_on_top_half (void);
  ```
- for CCID GET_SLOT_STATUS we call
  ```
  void p2_fw_ccid_get_slot_status_top_half (void);
  ```
- for CCID ICC_POWER_OFF we call
  ```
  void p2_fw_ccid_icc_power_off_top_half (void);
  ```
- for CCID XFR_BLOCK we call
  ```
  void p2_fw_ccid_xfr_block_top_half (void);
  ```
- for CCID GET_PARAMETERS we call
  ```
  void p2_fw_ccid_get_parameters_top_half (void);
  ```
- for CCID SET_PARAMETERS we call
  ```
  void p2_fw_ccid_set_parameters_top_half (void);
  ```
- for CCID ESCAPE we call

```
void p2_fw_ccid_escape_top_half (void);
```

These top half handlers do the smallest amount of work because we are still in the interrupt service routine. Mostly they only check if the frame is valid and of correct size and at the end schedule a bottom half which is run by execute task and which does the real work.

The bottom half functions are implemented in src/p2_fw_ccid.c with the following routines:

Get Status Bottom half

```
Bool p2_fw_ccid_get_slot_status_bottom_half (uint8_t slot_idx);
```

Returns status for slot defined by slot_idx.

Power On Bottom Half

```
Bool p2_fw_ccid_icc_power_on_bottom_half (uint8_t slot_idx);
```

Powers on the SAM.

Power Off Bottom Half

```
Bool p2_fw_ccid_icc_power_off_bottom_half (uint8_t slot_idx);
```

Powers off the SAM card or Halts a contactless card if one is present.

Get Parameters Bottom Half

```
Bool p2_fw_ccid_get_parameters_bottom_half (uint8_t slot_idx);
```

Returns parameters of a card in slot.

Set Parameters Bottom Half

```
Bool p2_fw_ccid_set_parameters_bottom_half (uint8_t slot_idx);
```

Sets parameters for a card in a slot.

Escape Bottom Half

```
Bool p2_fw_ccid_escape_bottom_half (uint8_t slot_idx);
```

Can be used to execute some of the direct mode commands in other modes.

Xfer Bottom Half

```
Bool p2_fw_ccid_xfr_block_bottom_half (uint8_t slot_idx);
```

This is the most complex function which performs multiple actions depending on the sub mode.

2.2.5 Standard Submode

In standard mode the reader sends APDUs to an ISO14443-4 compliment card:

```
static Bool p2_fw_ccid_xfr_bh_l4_xfer (uint8_t slot_idx);
```

and to the SAM:

```
static Bool p2_fw_ccid_xfr_bh_sam (uint8_t slot_idx);
```

The firmware also interprets the APDUs for MIFARE which are defined in PC/SC part 3. The list of supported APDUs is as following:

- **Get Data**
  ```
  static Bool p2_fw_pcsc_ext_cmd_get_data(uint8_t slot_idx);
  ```
- **Load Keys**
  ```
  static Bool p2_fw_pcsc_ext_cmd_load_key(uint8_t slot_idx);
  ```
- **General Authenticate**
  ```
  static Bool p2_fw_pcsc_ext_cmd_auth(uint8_t slot_idx);
  ```
- **Read binary**
  ```
  static Bool p2_fw_pcsc_ext_cmd_read_bin(uint8_t slot_idx);
  ```
- **Update binary**
  ```
  static Bool p2_fw_pcsc_ext_cmd_update_bin(uint8_t slot_idx);
  ```

2.2.6 Direct Submode

After we enter the direct mode the payload of xfer command is interpreted by the firmware. The main dispatcher function for direct mode:

```
Bool p2_fw_dm (uint8_t message_type, uint16_t allowed_cmds);
```

Every subset of direct mode is coded in its own file:

Reader Operations

Implements commands for Reader Operations. Entry point:

```
Bool p2_fw_dm_ro (uint8_t message_type);
```

HAL Commands

Implements commands for HAL Commands. Entry point:

```
Bool p2_fw_dm_hal (uint8_t message_type);
```

ISO14443-3

Implements commands for ISO14443-3. Entry point:

```
Bool p2_fw_dm_l3 (uint8_t message_type );
```

ISO14443-4Activation

Implements commands for ISO14443-4Activation. Entry point:

```
Bool p2_fw_dm_l4a (uint8_t message_type);
```

ISO14443-4

Implements commands for ISO14443-4 layer. Entry point:

```
Bool p2_fw_dm_l4 (uint8_t message_type );
```

MIFARE Exchange

Implements commands for MIFARE Exchange. Entry point:

```
Bool p2_fw_dm_xchg (uint8_t message_type);
```

CID Manager

Implements commands for CID Manager. Entry point:

```
Bool p2_fw_dm_cid (uint8_t message_type);
```

Contact Card Communication

Implements commands for Contact Card Communication. Entry point:

```
Bool p2_fw_dm_cc (uint8_t message_type);
```

Key Store

Implements commands for Key Store. Entry point:

```
Bool p2_fw_dm_key_store (uint8_t message_type);
```

Pooling and Activation Part

Pooling and activation is implemented in three functions. It has its own task:

```
void p2_fw_task_pcsc_poll_and_act_loop (void * param);
```

This function performs the activation for ISO14443-3 level. If a card supports an ISO14443-4 level card then it calls:

```
static Bool p2_fw_task_int_do_l4 (uint8_t * atq);
```

For pooling it uses:

```
static void p2_fw_task_int_do_poll (void);
```

2.3 Overwrite Configuration Mode

In this mode we issue:

```
void p2_fw_flash_erase_config(void);
```

from ***src/p2_fw_flash_utils.c*** which erases configuration from flash and then we call:

```
void p2_fw_invoke_error_mode(uint32_t error_code);
```

to inform the user that the overwrite was done.

2.4 Enter Secondary Bootloader Mode

In this mode we set a configuration option P2_FW_CFG_ENT_SEC_BOOT_MODE to buff[0] = 1 with the use of

```
Bool p2_fw_flash_set_config(uint32_t cfg_id, uint8_t * buff);
```

and then we issue a reset with the use of CMSIS function.

3. Library and source code overview

The firmware is composed from different libraries:

• BFL – NXP Basic Function Library 4.8 (can be found in bfl directory in the source tree) – provides all functions related to interaction with cards

• BFL Extensions – Extensions to the BFL provided by NXP – provides key store implementation required by the direct mode reader operation

• CMSIS - Cortex Microcontroller Software Interface Standard with extra peripheral drivers (can be found in cmsis directory in the source tree) – provides startup code and peripheral drivers and easier porting to other hardware platforms [3]

• FreeRTOS operating system (can be found in FreeRTOS_6_0_0 directory in the source tree) – provides multitasking and tasks intercommunication and easier porting to other hardware platforms

• lpcusb – USB stack designed for NXP LPC family of microcontrollers (can be found in lpcusb directory in the source tree)

• debug – simple debug library (can be found in debug directory in the source tree)

• firmware code (can be found in src and include directory in the source tree)

3.1 Architecture of the Pegoda 2 Firmware

The firmware is interrupt driven and can be split in two logical parts:

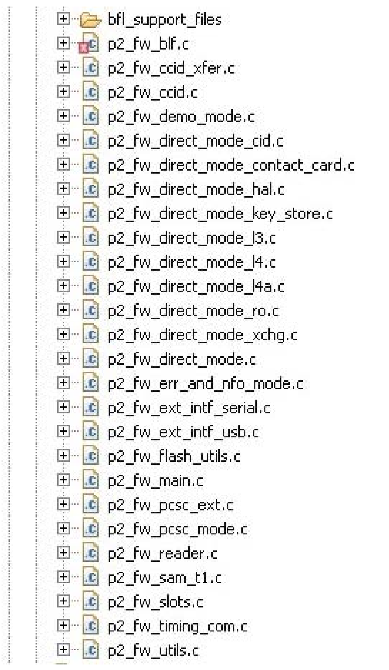
- the interrupt handling
- scheduled tasks

All interrupts are processed through top/bottom half (see **Error! Reference source not found.**) interfaces. Basically the interrupt service routine (also called top half) schedules bottom half which does not run in interrupt execution space but in “normal” execution space.

The scheduled tasks are described in the following chapters.

3.2 Description of firmware code

The following firmware source files (see Fig 5) can be found in the project. The main functionality will be described in the following section.



```
bfl_support_files
p2_fw_blf.c
p2_fw_ccid_xfer.c
p2_fw_ccid.c
p2_fw_demo_mode.c
p2_fw_direct_mode_cid.c
p2_fw_direct_mode_contact_card.c
p2_fw_direct_mode_hal.c
p2_fw_direct_mode_key_store.c
p2_fw_direct_mode_l3.c
p2_fw_direct_mode_l4.c
p2_fw_direct_mode_l4a.c
p2_fw_direct_mode_ro.c
p2_fw_direct_mode_xchg.c
p2_fw_direct_mode.c
p2_fw_err_and_nfo_mode.c
p2_fw_ext_intf_serial.c
p2_fw_ext_intf_usb.c
p2_fw_flash_utils.c
p2_fw_main.c
p2_fw_pcsc_ext.c
p2_fw_pcsc_mode.c
p2_fw_reader.c
p2_fw_sam_t1.c
p2_fw_slots.c
p2_fw_timing_com.c
p2_fw_utils.c
```

019aab996

Fig 5. Firmware source files In project

3.2.1 P2_fw_bfl.c

This file contains functions related to old BFL initialization and usage.

- bool **p2_fw_bfl_init** (void) – performs initialization of required BFL structures. We initialize the hardware interface, the required subsystems (IO and OpCtrl) and ISO14443 layer 3, 4A and 4 components.

  PARAMETERS: NONE

  RETURN: NONE

- void **p2_fw_bfl_set_up_rc_type_a_reading** (void) – sets up registers for ISO14443-3A card type reading.

  PARAMETERS: NONE

  RETURN: NONE

- bool **p2_fw_bfl_reset_reader** (void) – resets reader chip.

PARAMETERS: NONE

RETURN: NONE

- **void p2_fw_bfl_set_timeout** (uint16_t qsec, uint8_t aFlags) – sets timeout to the reader chip timer in milliseconds or microseconds depending on the flags setting. Please note that values larger than 39590ms will cause overflow.

  PARAMETERS: **qsec** − uint16_t value

  aFlags − can be:

  • P2_FW_TMR_MS − qsec is in milliseconds

  • P2_FW_TMR_US − qsec is in microseconds

  • P2_FW_TMR_START_NOW − force timer start

  RETURN: NONE

- **void p2_fw_bfl_set_com_speed**(uint8_t dri, uint8_t dsi) – set card communication speed parameters to the reader.

  PARAMETERS: **dri** − DRI parameter to set

  dsi − DSI parameter to set

  RETURN: NONE

- **void p2_fw_bfl_change_rc523_baud_rate** (uint32_t baudrate) – changes the reader UART speed to higher one from the default (9600

  PARAMETERS: **baudrate** − the baudrate value (9600 − 12880000)

  RETURN: NONE

- **static void p2fw_pcd_utils_wtx_cb** (phcsBflI3P4_CbSetWtxParam_t* wtx_param) − a callback for ISO14443-4 layer which sets the new WTX value.

  PARAMETERS: **wtx_param** − internal BFL structure

  RETURN: NONE

3.2.2 P2_fw_ccid.c

The file contains implementation of the USB CCID 1.1 messages. For every message there is a top and bottom half function.

The following messages are implemented in this file:

ICC Power On

- **void p2_fw_ccid_icc_power_on_top_half** (**void**);

  Bool **p2_fw_ccid_icc_power_on_bottom_half** (uint8_t slot_idx);

ICC Power Off

- **void p2_fw_ccid_icc_power_off_top_half** (**void**);

  Bool **p2_fw_ccid_icc_power_off_bottom_half** (uint8_t slot_idx);

Get Slot Status

- **void p2_fw_ccid_get_slot_status_top_half** (**void**);

  Bool **p2_fw_ccid_get_slot_status_bottom_half** (uint8_t slot_idx);

Get Parameters

- **void p2_fw_ccid_get_parameters_top_half** (**void**);

  Bool **p2_fw_ccid_get_parameters_bottom_half** (uint8_t slot_idx);

Set Parameters

- **void p2_fw_ccid_set_parameters_top_half** (**void**);

  Bool **p2_fw_ccid_set_parameters_bottom_half** (uint8_t slot_idx);

Escape

- **void p2_fw_ccid_escape_top_half** (**void**);

  Bool **p2_fw_ccid_escape_bottom_half** (uint8_t slot_idx);

Please refer to the source code and USB CCID 1.1 documentation for more information.

3.2.3 p2_fw_ccid_xfer.c

The file contains implementation of the USB CCID 1.1 XfrBlock message for contact and contactless cards which support APDUs.

The top half is implemented in**: void p2_fw_ccid_xfr_block_top_half (void) function**.

The bottom half function**: Bool p2_fw_ccid_xfr_block_bottom_half (uint8_t slot_idx**) calls the correct sub functions:

• for contactless cards:

static Bool p2_fw_ccid_xfr_bh_l4_xfer (uint8_t slot_idx) – function implements the ISO1444-4 protocol with chaining support, for more details refer to the source file

Bool p2_fw_pcsc_commands (uint8_t slot_idx) – function implements the APDUs

which are part PC/SC

• for contact cards:

static Bool p2_fw_ccid_xfr_bh_sam (uint8_t slot_idx) – function uses T=1 to exchange data with a contact SAM card, or more details refer to the source file

3.2.4 p2_fw_demo_mode.c

The demo mode is a simple demonstration on how to query for a card. The mode performs a selection of an NXP card according to ISO/IEC14443. Depending on the SAK byte of the selected card, the antenna LEDs start blinking and an acoustic signal is generated.

The whole demo mode is implemented as one FreeRTOS task:

void p2_fw_task_demo_mode (void * param) – main task function which calls the correct sub task according to requested mode:

- **use reader chip directly**:
  - o **static void p2_fw_task_demo_mode_bfl (void * param)** – the flow of this function is as following:

1. set up type ISO/IEC14443-3A card type reading
2. set up communication speed to 106kbps
3. turn on the antenna LEDs
4. enter infinity loop
5. set up timeout for REQA (pool for card) function
6. set up parameters for REQA function
7. execute REQA function
8. if REQA returns error, return to step 5.
9. REQA /ATQA returned success – card is in the field
10. set up timeout for AnticollSelect function

11. set up parameters for AnticollSelect function
12. perform AnticollSelect function
13. if selection of card is successful, beep according to reported SAK byte:

- if SAK == 0x08 → one time (MIFARE Classic Card 1k)
- if SAK == 0x18 → two times (MIFARE Classic Card 4k)
- if SAK == 0x00 → three times (MIFARE Ultralight/Ultralight C Card)
- if SAK == 0x20 → four times (ISO14443-4 Type Card (MIFARE Plus, MIFARE DESFire, ...))

14. halt the card and return to step 5

- **use SAM:**
  - o **static void p2_fw_task_demo_mode_sam (void * param)** – to use the SAM to control the chip, the following steps are required:

1. enter an endless loop (a FreeRTOS requirement)
2. reset the SAM by performing reinitialization
3. check SAM version and correct mode of operation
4. initialize the reader chip by performing Rc_Init command on SAM
5. after the reader chip has been successfully initialized, set up the RF field by issuing RC_RF_Control command
6. activate a card by performing Activate Card command on SAM
7. after successful selection of a card, an acoustic signal is generated as described in the direct reader chip usage step 13.

3.2.5 Functions for Direct mode

The following files contain functions for so called direct mode of operation of the Pegoda reader. This mode is meant to be used by the NXPRdLib. Functions found in this file bridge the difference between the NXPRdLib and the old BFL which is used as foundation of the current Pegoda firmware. If you are interested in how this mode works please refer to the source code.

p2_fw_direct_mode.c

p2_fw_direct_mode_cid.c

p2_fw_direct_mode_contact_card.c

p2_fw_direct_mode_hal.c

p2_fw_direct_mode_key_store.c

p2_fw_direct_mode_l3.c

p2_fw_direct_mode_l4a.c

p2_fw_direct_mode_l4.c

p2_fw_direct_mode_ro.c

p2_fw_direct_mode_xchg.c

3.2.6 p2_fw_err_and_nfo_mode.c

Contains a FreeRTOS task:

void p2_fw_task_err_and_nfo_loop (void * param) – which is used to inform the user of an error or that some action was completed. First the antenna LEDs blinks for five seconds with two blinks per second and then with rate of one blink and one beep per second informs the user of the error or that some action is completed. Please refer to source code file for more information.

3.2.7 p2_fw_ext_intf_serial.c

This file contains implementation of the serial interface (RS232, RS485) for LPC1768 UART for communication with a client. The serial communication can be only used in Direct Mode.

For more information please refer to

- LPC1768 user manual – Chapter 15. LPC17xx UART1 [8]
- the source file

3.2.8 p2_fw_ext_intf_usb.c

This file contains implementation of the USB interface (USB CCID 1.1) for communication with a client.

For more information please refer to:

- LPCUSB documentation (http://sourceforge.net/projects/lpcusb/)
- USB CCID 1.1 specification (http://www.usb.org/developers/devclass_docs/DWG_Smart-Card_CCID_Rev110.pdf)
- source file

3.2.9 p2_fw_flash_utils.c

The p2_fw_flash_utils.c file contains functions which perform actions over the IAP interface of LPC1768. For more information on IAP please refer to NXP LPC1768 User Manual. These actions are:

- **static Bool p2_fw_flash_utils_erase(int sec_from, int sec_to)** – performs erase of the flash from sec_from to sec_to

  PARAMETERS: - sec_from – int – start erase at this sector

  - sec_to – int – stop erase at this sector

  RETURN: -TRUE if success

  -FALSE if error

- **static Bool p2_fw_flash_utils_flash(int sec, uint8_t * buff, int size)** – performs flashing of the internal flash

PARAMETERS: - sec – int – sector to flash

- buff – uint8[] - data to flash

- size – int – size of buff (should be 128, 512 or 1024)

RETURN: - TRUE if success

- FALSE if error

- **Bool p2_fw_flash_read_serial(uint32_t * sernum)** – returns the CPU serial

  PARAMETERS: - sernum – uint32_t * - pointer to a uint32_t for storing the read CPU serial number

  RETURN: - TRUE if success

  - FALSE if error

3.2.10 p2_fw_main.c

The firmware starts in this file. In the main loop (int main()), initialization of hardware and software parts of the firmware are performed:

1. first we set up hardware (see p2_fw_reader.c for more information about setting up hardware)
2. then we initialize the debugging framework – this depends on type of build that we are building
3. next come reading the configuration from DIP switches and flash (see p2_fw_reader.c for more information about reading configuration)
4. now we can setup the external interface
5. now we have to set up FreeRTOS queue – used for communication between top bottom halfs and semaphore for protecting access to reader chip – so that we do not use it from two tasks at the same time
6. next we try to set up the reader chip
7. if we find a supported chip we set up the correct tasks for particular mode – if not found we enter into error and information task with correct error code
8. next we have to start the FreeRTOS scheduler (please refer to FreeRTOS documentation for more information) and loop forever

For more detailed information please refer to the source code file.

3.2.11 p2_fw_pcsc_ext.c

In this file we implement PC/SC extensions for memory cards. The implemented commands are:

- Get Data – see **static Bool p2_fw_pcsc_ext_cmd_get_data(uint8_t slot_idx);**
- Load Keys – see **static Bool p2_fw_pcsc_ext_cmd_load_key(uint8_t slot_idx);**
- General Authenticate – see **static Bool p2_fw_pcsc_ext_cmd_auth(uint8_t slot_idx);**
- Read Binary – see **static Bool p2_fw_pcsc_ext_cmd_read_bin(uint8_t slot_idx);**

- Update Binary – see **static Bool p2_fw_pcsc_ext_cmd_update_bin(uint8_t slot_idx);**

For more information please refer to the source file.

3.2.12 p2_fw_pcsc_mode.c

This file contains functions related to PC/SC mode of operation by Pegoda 2 reader. We define two FreeRTOS tasks:

- **void p2_fw_task_pcsc_execute (void * param)**

This task provides executions of bottom halfs which are scheduled by top halfs. Most of time this task blocks on a queue waiting for interrupts to schedule a bottom half which is then executed by this task.

- **void p2_fw_task_pcsc_poll_and_act_loop (void * param)**

This task provides polling and activation mechanism required by the PC/SC specification and it is split in three parts:

- main polling and ISO14443-3A activation (found in this function)
- ISO14443-4A activation function (see **static Bool p2_fw_task_int_do_l4 (uint8_t * atq, phcsBflI3P3A_AnticollSelectParam_t * ans_p)**)
- existing cards polling function (see **static void p2_fw_task_int_do_poll (void)**)

Please refer to the source file for more information.

3.2.13 p2_fw_reader.c

In this file we initialize the hardware parts of the Pegoda 2 reader:

- **void p2_fw_reader_setup_hardware(void) function:**
  1. runs SystemInit() function provided by the CMSIS library
  2. sets the Interrupt Vector table pointer to the correct address
  3. sets up the pin function of used GPIOs – first the DIP switches, next configuration pins and lastly LEDs and beeper pin
  4. next we set the correct direction and initial value (only for output pins) for this pins
- **void p2_fw_reader_read_config(void)** function reads the value of the DIP switches and stores a local copy of it
- **void p2_fw_reader_set_up_external_interface (void)** function sets up the correct external interface according to the configuration that was set by DIP switches
- **Bool p2_fw_reader_set_up_reader_chip(void)** function checks if the set configuration is possible – if it can find the correct reader chip and/or SAM – and sets up the old BFL if needed

Please refer to the source code for more information.

3.2.14 p2_fw_sam_t1.c

This file contains implementation of the ISO/IEC 7816/3 standard:

- **void p2_fw_sam_t1_init(uint8_t chip, uint8_t mode)** – initialize the processors I/O pins according to chip and mode parameter to work with SAM card.

  PARAMETERS: chip – P2_FW_READER_CHIP_RC523 or

  P2_FW_READER_CHIP_SAM

  mode – P2_FW_READER_MODE_SAM_IN_X or

  P2_FW_READER_MODE_SAM_NON_X

- **void p2_fw_sam_t1_start**(**void**) – it perform activation sequence according ISO 7816-3:2006 (Figure 1). The function returns immediately but the user should wait until the module is busy (see p2_fw_sam_t1_is_busy) then he can read the ATR of inserted SAM card with p2_fw_sam_t1_get_atr.

- **void p2_fw_sam_t1_deinit**(**void**) – the function perform Deactivation sequence according to ISO 7816/3:2006 (Figure 6).

- **void p2_fw_sam_t1_warm_reset**(**void**) – the function perform warm reset according to ISO 7816-3:2006 (Figure 1). The function returns immediately but the user should wait until the module is busy (see p2_fw_sam_t1_is_busy) then he can read the ATR of inserted SAM card with p2_fw_sam_t1_get_atr.

- **void p2_fw_sam_t1_send(uint8_t *data, uint32_t len, uint8_t frame_type)** The function sends len bytes of data to SAM.

  PARAMETERS: data – uint8_t * - pointer to data to send

  len – uint32_t – length of data to send

  frame_type:

  - ***P2_FW_SAM_FRAME_APDU*** the function prepends NAD, PCB and LEN bytes, and appends LRC to data parameter (according ISO 7816-3:2006).
  - ***P2_FW_SAM_FRAME_T1*** the function just send data to SAM card.

The function returns immediately but the user should wait until the module is busy (see p2_fw_sam_t1_is_busy) then he can read the response with p2_fw_sam_t1_receive.

- **uint32_t p2_fw_sam_t1_receive(uint8_t *data, uint8_t frame_type)** – the function returns the number of bytes received from SAM card. If frame_type equals P2_FW_SAM_FRAME_APDU only the INF field is returned.

  PARAMETERS: data – uint8_t * - pointer to received data

  frame_type – uint8_t – frame type

  RETURN: number of bytes received

- **Bool p2_fw_sam_t1_is_busy(void) –** the function return true if the SAM module is busy and false otherwise.

- **void p2_fw_sam_t1_get_atr(uint8_t * buffer, uint8_t *max_length)** – the function returns ATR of currently inserted SAM card.

  PARAMETERS: buffer – uint8_t * - buffer in which to store ATR max_length – uint8_t * - the size of the buffer/length of ATR (on return)

- **void p2_fw_sam_t1_set_etu(uint8_t fi_di)** – function sets the module's internal etu according to fi_di parameter. fi_di is encoded as TA1 of ATR (ISO 7816/3:2006; 8.3).

The module uses TIMER0, TIMER2, GPIO falling edge interrupt on SAM's I/O pin and PWM0 for SAM's CLK pin. When the user calls ***p2_fw_sam_t1_start*** or ***p2_fw_sam_t1_warm_reset*** the SAM card is in reset state and after 400 clocks (ISO 7816-3:2006) the TIME0 interrupt triggers and sets the RST pin high. Then the module is in read state and waits for GPIO failing edge interrupt (start bit). After receiving start bit TIMER0 is used to capture data bits and parity bit. TIMER2 is used to timeout if no start bit is received. All that time the function ***p2_fw_sam_t1_is_busy*** returns true. For sending data to SAM, TIMER0 is used to shift each bit to I/O pin. When the answer is received the module is not busy anymore and it can be read with ***p2_fw_sam_t1_receive*** function.

3.2.15 p2_fw_slots.c

We define multiple slots in which cards can reside. At one time you can use only one ISO144443 card or fourteen ISO14443 cards. Every card has its own communication settings and this are stored in a slot. All slot and CID management functions can be found in this source file:

- **void p2_fw_slots_init(void)** – initialize slots and prepare them for work
- **Bool p2_fw_slots_free_cid (uint8_t cid)** – frees occupied CID channel for ISO14443-4 communication

  PARAMETERS: cid – uint8_t – CID channel to free

  RETURN: TRUE if success, FALSE in case of ERROR
- **Bool p2_fw_slots_get_free_cid (uint8_t * cid)** – returns a free CID channel for ISO14443-4 communication

  PARAMETERS: cid – uint8_t * – pointer to uint8_t where to store CID channel

  RETURN: TRUE if success, FALSE in case of ERROR
- **Bool p2_fw_slots_get_free_slot (uint8_t * slot)** – returns a free slot to store a card (CID channel and slot index are two different things and should not be mixed up)

  PARAMETERS: slot – uint8_t * - pointer to uint8_t where to store slot index

  RETURN: TRUE if success, FALSE in case of ERROR
- **Bool p2_fw_slots_add_new_l4_card (phcsBflI3P4AAct_RatsParam_t * rat_p, uint8_t cid_index, uint8_t sak, uint8_t * atq, uint8_t * uid, uint8_t uid_len)** – adds a new ISO14443-4 card to a slot

  PARAMETERS:

  - rat_p – phcsBflI3P4AAct_RatsParam_t * – pointer to RATS structure used for card activation
  - cid_index – uint8_t – CID channel used for communication

- sak – uint8_t – SAK that card returned at ISO14443-3A activation
- atq – uint8_t * - pointer to buffer containing ATQ
- uid – uint8_t * - pointer to buffer containing UID
- uid_len – uint8_t – uid buffer length

RETURN: TRUE if success, FALSE in case of ERROR

- **Bool p2_fw_slots_add_new_l3_card (uint8_t sak, uint8_t * atq, uint8_t * uid, uint8_t uid_len) – adds a new ISO14443-3 card to a slot**

  PARAMETERS:

  - sak – uint8_t – SAK that card returned at ISO14443-3A activation
  - atq – uint8_t * - pointer to buffer containing ATQ
  - uid – uint8_t * - pointer to buffer containing UID
  - uid_len – uint8_t – uid buffer length

  RETURN: TRUE if success, FALSE in case of ERROR

- **Bool p2_fw_slots_add_new_sam_card (void)** – adds a new SAM to a slot

  PARAMETERS: NONE

  RETURN: TRUE if success, FALSE in case of ERROR

- **Bool p2_fw_slots_is_known_l3_card (uint8_t * uid, uint8_t uid_len)** – check if the card is already occupying a slot

  PARAMETERS:

  - uid – uint8_t * - pointer to buffer containing UID
  - uid_len – uint8_t – uid buffer length

  RETURN: TRUE if success, FALSE in case of ERROR

- **Bool p2_fw_slots_get_atr (uint8_t slot_index, uint8_t * buffer, uint8_t *max_length)** – returns an ATR for a particular slot (by calling p2_fw_slots_atr_l4_cards or p2_fw_slots_atr_l3_card function)

  PARAMETERS:

  - slot_index – uint8_t – index of the slot for which we would like to get ATR
  - buffer – uint8_t * - pointer where to save the ATR
  - max_length – maximum length of the ATR that can be stored to the buffer

  RETURN: TRUE if success, FALSE in case of ERROR

- **static Bool p2_fw_slots_atr_l4_cards (uint8_t slot_index, uint8_t * buffer, uint8_t *max_length)** – returns correctly formatted ATR for ISO14443-4 type card

  PARAMETERS:

  - slot_index – uint8_t – index of the slot for which we would like to get ATR
  - buffer – uint8_t * - pointer where to save the ATR
  - max_length – maximum length of the ATR that can be stored to the buffer

  RETURN: TRUE if success, FALSE in case of ERROR

- **static Bool p2_fw_slots_atr_l3_cards (uint8_t slot_index, uint8_t * buffer, uint8_t *max_length)** – returns correctly formatted ATR for ISO14443-3 type card

  PARAMETERS:

  - slot_index – uint8_t – index of the slot for which we would like to get ATR
  - buffer – uint8_t * - pointer where to save the ATR
  - max_length – maximum length of the ATR that can be stored to the buffer

  RETURN: TRUE if success, FALSE in case of ERROR

- **void p2_fw_slots_remove_card (uint8_t slot_index)** – removes card from a slot

  PARAMETERS:

  - slot_index – uint8_t – index of the slot from which we would like to remove the card

  RETURN: NONE

Please refer to the source code for more information.

3.2.16 p2_fw_timing_com.c

This file contains functions for timing of operations. The supported timing modes are:

- FTD – frame time delay – time between the end or transmission to the card and start of reply (this time is calculated from the RC523 timer)
- COM – whole command time – from start of the command until the end (this time is calculated by using LPC1768 timer TIM1)

Please refer to the source code for more information.

3.2.17 p2_fw_utils.c

Contains simple helpful functions:

- **uint8_t p2_fw_utils_get_dri(uint8_t ta1)** – returns DRI parameter from cards ATS

  PARAMETERS: ta1 – cards ATSs TA1

- **uint8_t p2_fw_utils_get_dsi(uint8_t ta1)** – returns DSI parameter from cards ATS

  PARAMETERS: ta1 – cards ATSs TA1

- **void p2_fw_utils_blink(int count)** – blinks antenna LEDs count times

  PARAMETERS: count – times to blink the antenna LEDs

- **void p2_fw_utils_field_off (void)** – turns off the RF field
- **void p2_fw_utils_field_on (uint16_t wFiledRecoveryTime)** – turns on the RF field

  PARAMETERS: wFiledRecoveryTime – time to wait fro field to settle

- **void p2_fw_utils_reg_write (uint8_t addr, uint8_t val)** – write to a RC523 register

  PARAMETERS: addr – uint8_t – register address value – uint8_t – value to write

- **void p2_fw_utils_reg_read (uint8_t addr, uint8_t * val)** – read from RC523 register

  PARAMETERS: addr – uint8_t – register address

  value – uint8_t * – pointer to store the value

4. Code execution overview

4.1 Initialization

On entry, after main we first setup the hardware by calling:

```
void p2_fw_reader_setup_hardware(void);
```

in ***src/p2_fw_reader.c***. This function initializes the ARM CMSIS library and sets up hardware.

Next we setup the slots with a call to:

```
void p2_fw_slots_init(void);
```

Now we are ready to read configuration – this configuration tells the currently set operating mode of the Pegoda.

```
void p2_fw_reader_read_config(void);
```

Now that we have read the configuration we can set up the external interface defined by PINs connected to DIP switch 3 and 4 by calling:

```
void p2_fw_reader_set_up_external_interface (void);
```

Depending on the configuration the correct interface is initialized:

- USB:

```
void p2_fw_usb_init_usb(void);
```

- serial (RS232 and RS485):

```
void p2_fw_usb_init_serial(void);
```

- ethernet:

```
void p2_fw_usb_init_ethernet(void);
```

Now we need to initialize the timer for timing services by calling:

```
void p2_fw_timing_init(void);
```

Now we only have to set up the reader chip by calling:

```
Bool p2_fw_reader_set_up_reader_chip(void);
```

This function – depending on the configuration and chip type – initializes the correct subsystems.

Initialization can be performed on BFL or/and SAM depending on configuration.

For initialization of the BFL we call:

```
Bool p2_fw_bfl_init (void);
```

For SAM initialization we have to call the correct sequence of:

```
void p2_fw_sam_t1_init(uint8_t chip, uint8_t mode);
void p2_fw_sam_t1_start(void);
Bool p2_fw_sam_t1_is_busy(void);
Bool p2_fw_sam_t1_is_sam_inserted(void)
void p2_fw_sam_t1_deinit(void);
```

Now we can check which mode is requested and start the correct tasks for it.

5. Legal information

5.1 Definitions

Draft — The document is a draft version only. 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 herein and shall have no liability for the consequences of use of such information.

5.2 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.

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 medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts 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.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on a weakness or default in the customer application/use or the application/use of customer's third party customer(s) (hereinafter both referred to as "Application"). It is customer's sole responsibility to check whether the NXP Semiconductors product is suitable and fit for the Application planned. Customer has to do all necessary testing for the Application in order to avoid a default of the Application and the product. NXP Semiconductors 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 national authorities.

5.3 Licenses

Purchase of NXP ICs with ISO/IEC 14443 type B functionality

This NXP Semiconductors IC is ISO/IEC 14443 Type B software enabled and is licensed under Innovatron's Contactless Card patents license for ISO/IEC 14443 B.

The license includes the right to use the IC in systems and/or end-user equipment.

RATP/Innovatron Technology

5.4 Trademarks

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

MIFARE — is a trademark of NXP B.V.

6. Index

7. List of figures

8. List of tables

9. Contents

Date of release: 21 March 2011
196110
Document identifier: AN10993