PHYCHIPS

RED FWM
2020-11-01

Firmware Development
User Manual

Rev 1.0.4 2020-11-01

Contents

1 Revision History

Version	Date	Description
1.0.0	2015.03.05	Initial Release
1.0.1	2015.06.11	Rename doc's name, fixed some error
1.0.2	2015.06.22	Fixed typos
1.0.3	2016.11.25	Modified in section 9.1 Wakeup Interrupt
1.0.4	2020.11.01	Added in section 6.10 Bootloader Added API functions in section 7.1 HAL

2 RED4S Firmware Overview

This document describes architecture and function of RED4S firmware. This document intends to help engineers who are currently developing MAC layer or embedded function of MCU. The outline of firmware architecture is as shown below.

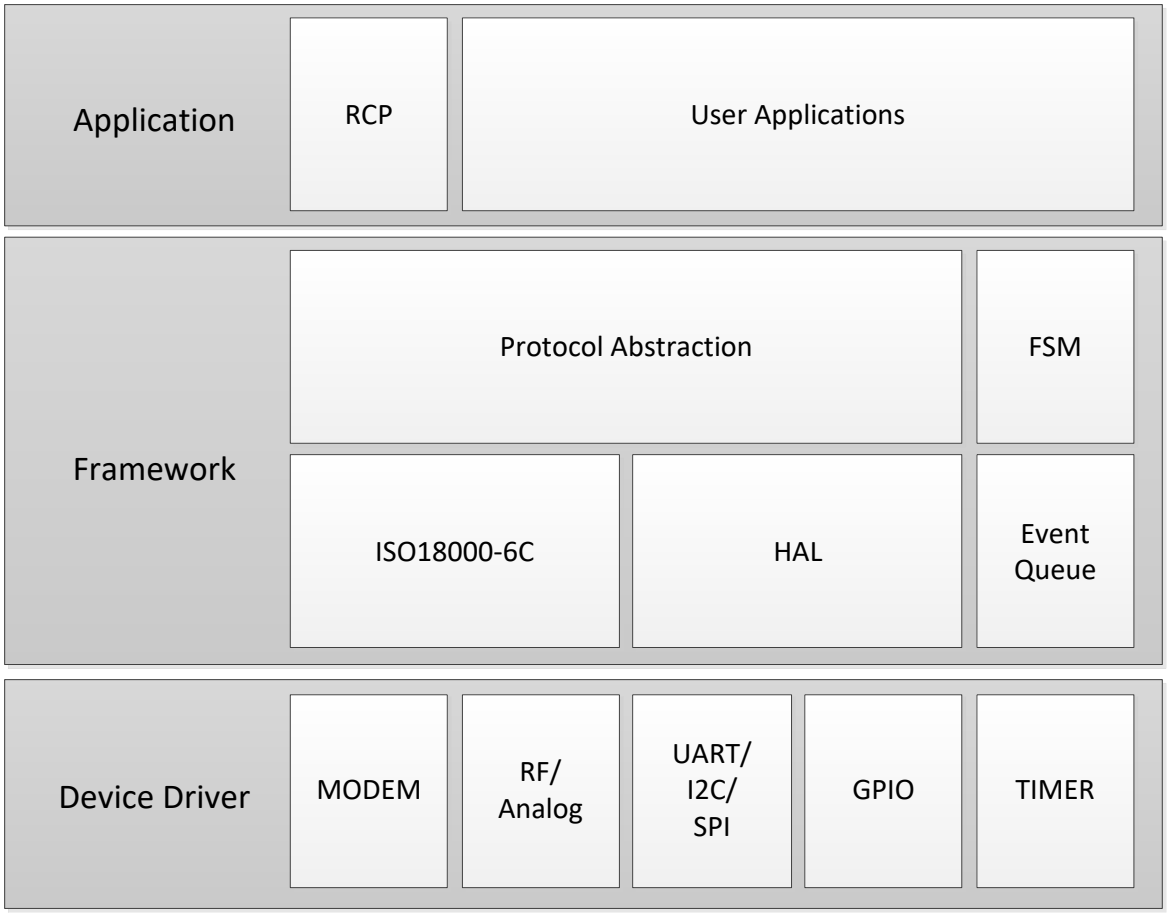


Figure 1 Firmware Architecture

3 Naming Conventions

Naming conventions are used to make codes easier to read and understand.

3.1 Constants

Use upper case letters for defined constant.

```
#define TYPEC_QFP100                    (400)  // slotted ALOHA Qfp value 4.0
```

3.2 Type names

Append suffix _type to type names of structure, union, and enum.

```
typedef struct
{
        typec_pc_uii_type target;
} typec_tag_type;
```

3.3 Enum members

Use upper case letters for enum members.

```
typedef enum
{
        MEM_RESERVE   = 0x00,
        MEM_EPC         = 0x01,
        MEM_TID         = 0x02,
        MEM_USER        = 0x03,
        MEM_TID_USER= 0x04
} typec_membank_type;
```

3.4 Functions

Use lower case letters and under bar _ as word separators.

[module name]_[verb]_[objective] or rcp_parse_cmd

4 Directory Structure

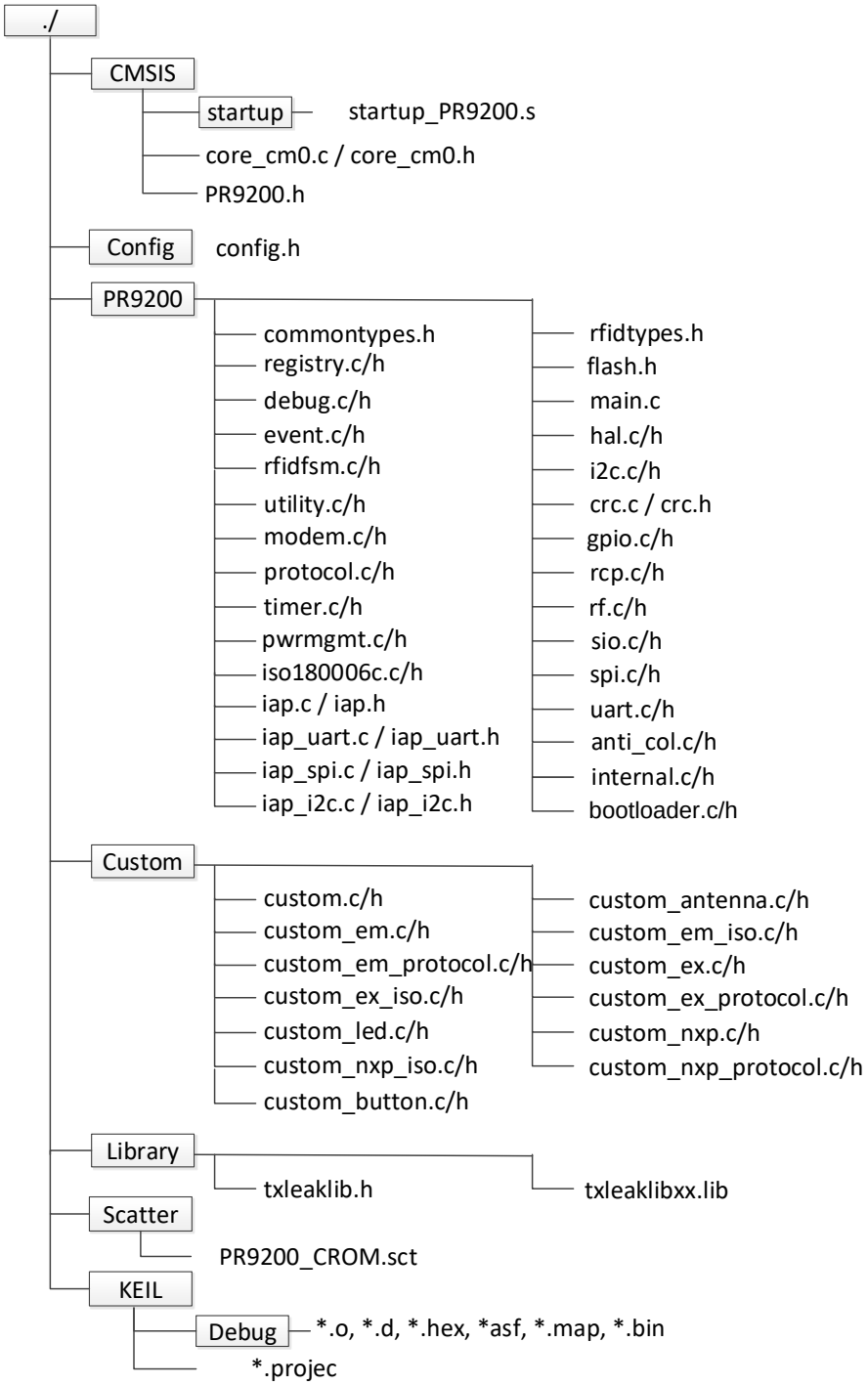


Figure 2 RED4S Directory Structure

4.1 CMSIS (Cortex Microcontroller Software Interface Standard)

CMSIS provides simple software interface for a Cortex-M series processor-based system. Cortex-M0 core definitions and static function are in core_cm0.c/core_cm0.h. This core file includes many common features. A PR9200.h is top-level header file defining the interrupt number and peripheral registers of RED4S. Also a startup_PR9200.s file in startup directory provides a system start-up method

4.2 Config

A config.h file declares for compile options.

4.3 PR9200

PR9200 directory includes all files to operate the RED4S. Brief description of source files and functions is shown below.

RCP
rcp.c / rcp.h: Packet Handler for RCP (Reader Control Protocol)
sio.c / sio.h: Implementation of serial interface

Framework
rfidtypes.h: The common data type for HAL and Protocol.
hal.c / hal.h: Implementation of hardware abstraction layer
protocol.c / protocol.h: Implementation of RFID protocol
iso180006c.c / iso180006c.h: Handler for Air Interface Protocol packets (EPC Gen2 Protocol stack)
rfidfsm.c / rfidfsm.h: Inventory/LBT/FH time scheduling
event.c /event.h: implementation of event handler
anti_col.c / anti_col.h: implementation of anti-collision algorithms

Device drivers
uart.c / uart.h: Handler for inbound UART serial packet. UART event generation
spi.c / spi.h: Handler for inbound SPI serial packet. SPI event generation
i2c.c / i2c.h: Handler for inbound I^2C serial packet. I2C event generation
gpio.c / gpio.h: Implementation of GPIO driver.
timer.c / timer.h: Implementation of system timer driver
modem.c / modem.h: Implementation of Interface for MODEM registers
rf.c / rf.h: Implementation of Interface for RF registers
pwrmgnt.c / pwrmgnt.h: Implementation of power management
registry.c / registry.h: Implementation of registry management

Bootloader
bootloader.c / bootloader.h: Implementation of bootloader

IAP (Download)
Iap.c / iap.h: Implementation of download service
Iap_uart.c / iap_uart.h: Handler for inbound UART serial packet for IAP
Iap_spi.c / iap_spi.h: Handler for inbound SPI serial packet for IAP
Iap_i2c.c / iap_i2c.h: Handler for inbound I^2C serial packet for IAP

Misc. services
debug.c / debug.h: Implementation of debug service
utility.c/ utility.h: Implementation of utility service

4.4 Custom

Custom provides application to use the LED and Beep with GPIO for user. In addition to this, user can add user application freely.

4.5 Library

Library provides Tx leakage cancelation algorithm

4.6 Scatter

PR9200_CROM.sct is a scatter load description file. The scatter file determines how the memory layout of user's controller is organized. In essence, user can allocate objects to specific memory regions, determine the mapping of load regions to execution regions.

4.7 KEIL

The project file for uVision is in KEIL directory. Also all output files of compile is in Debug sub directory

5 Firmware Memory Map

Firmware memory map is shown below. RED4S has 64K Flash memory and 16K SRAM. Flash memory is divided into some regions. 57K of 64K Flash memory is used RED4S code and user's code and 7K is used for special purpose. The lowest 3K byte is used for Bootloader for firmware update. The highest 1K byte is used for registry to store internal parameters. 3K byte under the Registry is used for firmware update and codes are loaded.

Memory layout is decided by scatter-load description file(*.sct). The scatter-load description file for PR9200 is provided(./Scatter/PR9200_CROM.sct). To use this file, user must add the scatter-load description file to linker option. For details about the memory map, refer to PR9200_CROM.sct file.

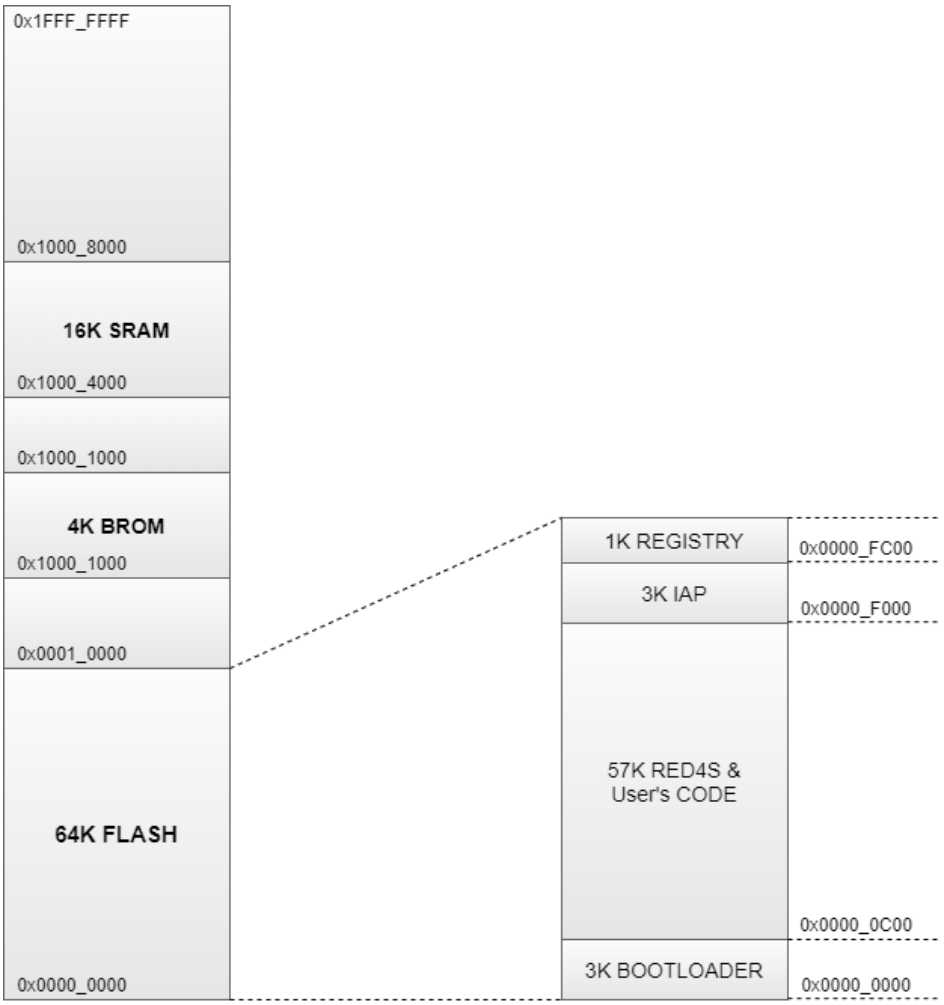


Figure 3 Firmware Memory Map

6 Module

6.1 Data types

To recognize bit size of each data type, data types are re-defined in system.h and described as below.

```
typedef signed char         int8;     //    -128 to +127
typedef unsigned char       uint8;    //    0 to 255
typedef signed short        int16;    //    -32768 to +32767
typedef unsigned short      uint16;   //    0 to 65535
typedef signed int          int32;    //    -2147483648 to +2147483647
typedef unsigned int        uint32;   //    0 to 4 294 967 295
typedef long long           int64;    //    -9223372036854775808 to 9223372036854775807
typedef unsigned long long  uint64;   //    0 to 18446744073709551615
typedef unsigned char       BOOL;
```

6.2 Compile options

Compile options are coded in config.h file in Config directory. To use debug codes, define below statement in config.h

#define __DEBUG__

Select one of three statements as described in below. To use UART port, select the first statement. To use SPI port, select the second statement. To use I2C port, select the third statement. Three serial ports are assigned to RCP protocol. To use serial ports for other purpose, the user should change each driver code

#define __FEATURE_UART0__

or

#define __FEATURE_SPI_SLAVE_RCP__

or

#define __FEATURE_I2C_SLAVE_RCP__

Code feature is used for activating specific codes. Unused code should be de-activated to reduce code size.

6.3 RCP

RED4S is controlled through RCP (Reader Control Protocol,) which is using the UART, SPI or I2C serial interface. RCP protocol is programmed in rcp.c. If received RCP packet, event to decode RCP packet is occurred. RCP packet is stored in rcp_req_pkt_c. The RCP packet format is shown in the figure below. Preamble and end mark have constant values. 0xBB is used for preamble and 0x7E is used for end mark. Header consists of 3 fields: Message Type, Code, and Payload Length. Message Type field indicates packet types; command (0x00), response (0x01), notification (0x02). Code field is used to indicate control command type or response type. Payload Length field is used to inform RED4S about payload length. Payload contains either data or control information. More details are described in 'RED-RCPXX Reader Control Protocol' document.

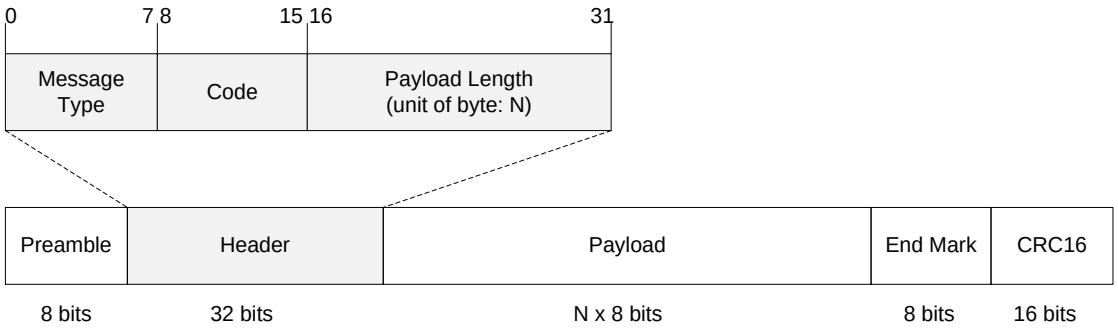


Figure 4 RCP Format

RCP interact with serial interface through SIO layer instead of direct. As abstraction layer between RCP and serial interface, SIO provides functions to send or receive various packets and common buffer. RCP can use same functions regardless of the underlying serial interface. That is, RCP just use one send function to response or notification message. Also serial interface reuses the common behavior of SIO.

6.4 Protocol

Protocol layer provides EPC GEN2 or ISO18000-6 air interface protocol. This layer is simple interface to access the air interface.
Protocol provides following functions:

- Protocol parameter configuration (such as Select and Query)
- Modulation mode configuration
- Anti-collision algorithm configuration
- Perform inventory
- Tag memory read/write
- Tag lock and Kill
- Others

User can easily configure parameters and access to air interface by using Protocol API programmed in protocol.c. For complete details about the Protocol, refer to Protocol API Reference chapter.
RED4S provides some anti-collision algorithms and user can select algorithm. Algorithm is divided on how to adjust session (Select) or Q value(Query). Considering the collision or empty slot, Q value is changed in various ways.

6.5 HAL

The HAL(Hardware Abstraction Layer) provides a simple device driver interface for programs to connect to the underlying hardware. The HAL interfaces to device drivers by abstracting hardware details from the software application. This abstraction minimizes or eliminates the need to access hardware registers directly to connect to and control peripherals. Therefore, the HAL allow user to write programs using a consistent API regardless of the underlying implementation of the device hardware.
HAL provides following functions:

- Region, Channel and Tx power level configuration
- FH and LBT parameter configuration
- Reader power control
- GPIO, Timer and Serial IO control
- Registry data management
- Others.

The HAL supports available RED4S processor implementations. The HAL defines a set of functions that user uses to initialize. The user calls the HAL API to access device such as timers, modem and rf for air interface, and I/O peripheral. For complete details about the HAL, refer to HAL API Reference chapter.

6.6 Event

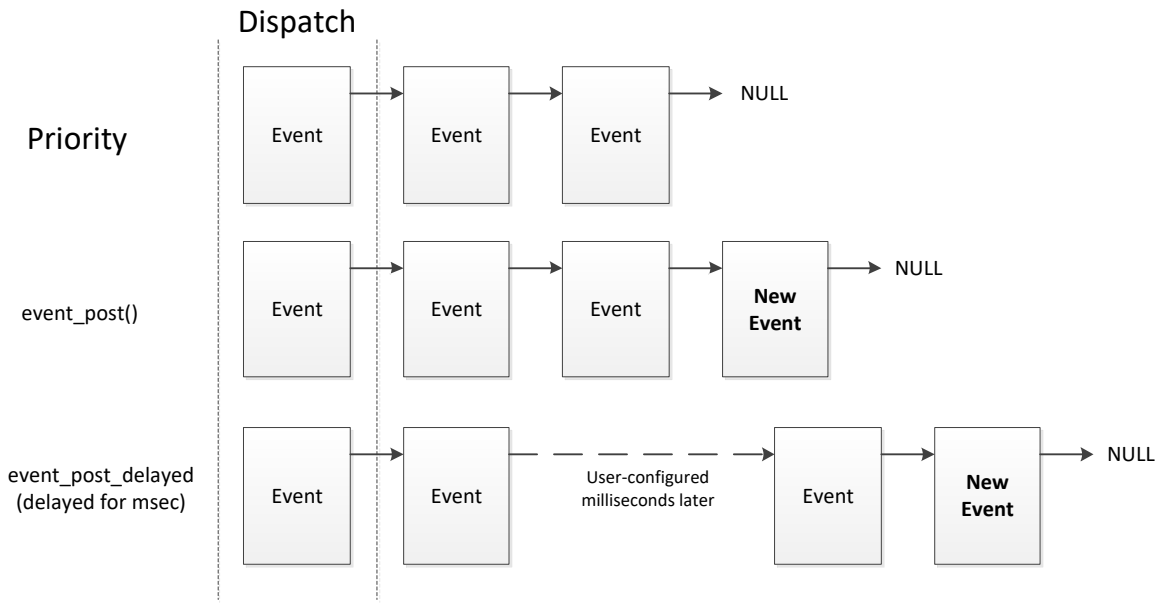
RED4S provides event queue. This event queue has several event block and use linked list to connect each event block. Every event block consists of event handler and parameter. The most important attribute of event block is event handler to describe behavior specific to the event. Event parameter indicates event-type. Some event type is predefined and user can define additional event type. Events are handled uniformly by event handlers, which is function pointer.

```
typedef void(*event_handler)(struct EVENT e);

typedef uint8 EVENT_PARAM;

typedef struct EVENT
{
        event_handler handler;
        EVENT_PARAM param;
} EVENT;
```

First event in event list has the highest priority. New event is connected at the end of event and has the lowest priority. Event is dispatched in order of priority. For using event, two functions are provided: event_post() and event_post_delayed(). event_post() is normally used to generate and post the new event. The event_post_delayed() can post new event user-configured milliseconds later.

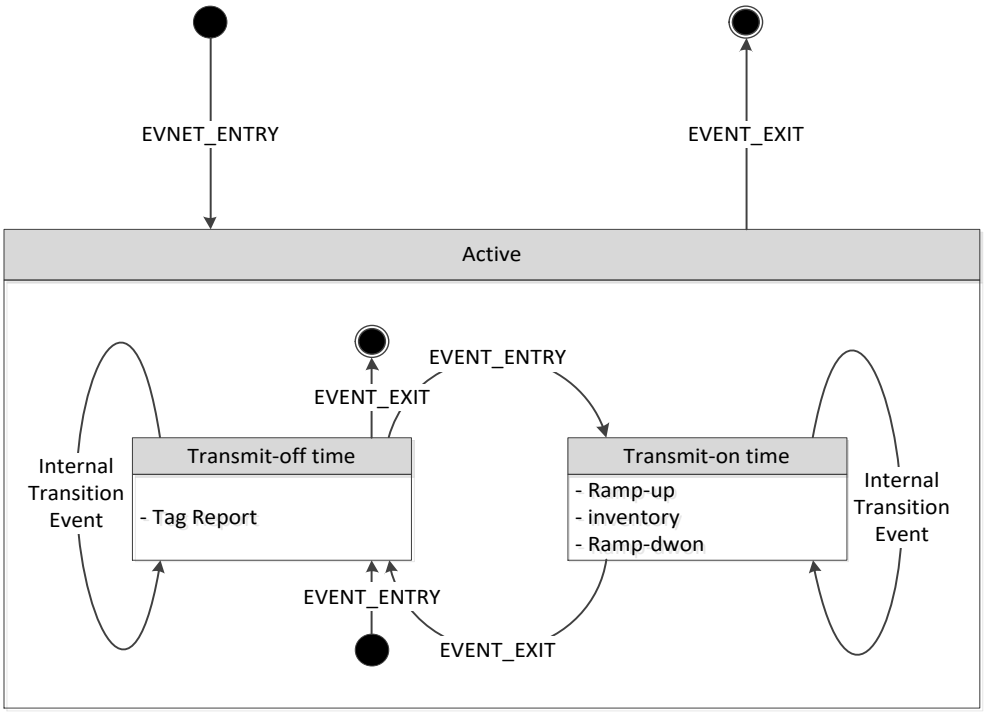


In case LED turn on and off 1sec later, user can use the event as following. First, declare a new event, event handler, and event type. Then, put EVENT_LED_ON event into the event list by using event_post() and LED will turn on immediately. EVENT_LED_OFF event is posted 1sec later by using event_post_delayed() with delay time and LED will turn off 1sec later.

```
// event generation
 void led_event(EVENT e)
{
        switch(e.param)
        {
        case EVENT_LED_ON:
                /* do something */
                break;
        case EVENT_LED_OFF:
                /* do something */
                break;
        }
}
// event post
void led_blink(void) // blink 1 time
{
        EVENT e;
        e.handler = led_event;
        e.param = EVENT_LED_ON;
        event_post(e);
        e.param = EVENT_LED_OFF;
        event_post_delayed(e,1000);
}
```

6.7 FSM (Finite State Machine)

RED4S provides event-driven FSM. FSM is composed of a finite number of states, transitions between those states, and event. Every FSM has entry/exit point. Entry and exit action and default transitions are implemented inside the event handler in response to the pre-defined events EVENT_ENTRY and EVENT_EXIT. The FSM generates and dispatches these events to appropriate handlers upon state transitions.
RED4S FSM has hierarchical three states. In Active state, there are two sub-states Transmit-off time state and Transmit-off time state. Entry event of Active state is occurred to perform inventory and exit event is occurred when inventory is discontinued. If enter the Active state, state transition is repeatedly happened between Transmit-off time state and Transmit-on time state whenever Transmit-off time and transmit-on time is expired. Transmit-off time state and Transmit-on time state transition use event_post_delayed() function to use Internal time. Each states use various events and switches internal events. In transmit-off time state, event is used to report tag ID and perform frequency hopping and LBT (it is only enabled). In transmit-n time state, event is used to ramp-up, ramp-down and inventory.



6.8 Registry

Registry is some specific section of flash memory. Registry is located from 0000_FC00h to 0000_FFFFh in flash memory. Registry allows memory to store the data without downloading program code. Registry includes registry version, firmware date, parameters such as band, modulation and tx power. Data format is defined in rfidtypes.h. During the boot-up sequence, active data is loaded for initializing parameters instead of default settings. Stored data are not selectable. Thus, all parameters should be checked carefully before the user stores settings into Registry.

6.9 IAP (In-Application Programming)

By using IAP feature, firmware can be updated. Firmware code in Host can be transferred to RED4S through UART, SPI or I²C. During download process, keep away from other operation. Source code is programmed in iap.c. IAP code includes RCP, UART, SPI, I²C, Timer and Flash control. iap_main() is loaded at 0x0000_F000. To start download, call iap_main (0x0000_F000) with parameter indicating serial interface. Downloading is executed from 0x0000_0000 to 0x0000_EFFF and Registry and IAP code is unchanged.

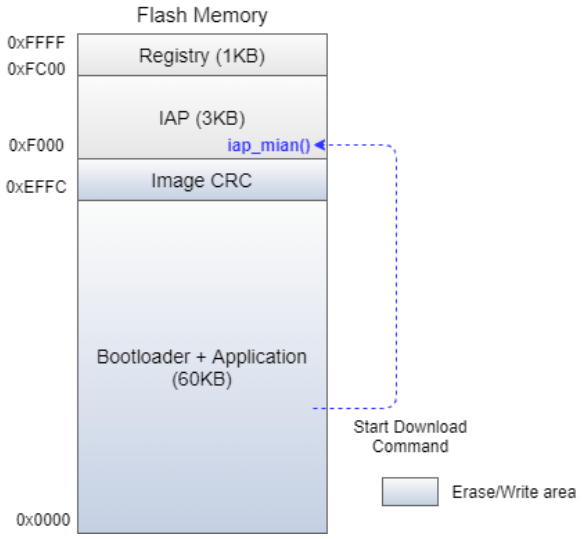


Figure 5 IAP Memory Map

If the reader receives start download command from Host, iap_main() function is called in order to move pointer to IAP area. Flash is erased and wrote according to the defined procedure. After firmware update complete, reset to start new firmware.

6.10 Bootloader

The bootloader is stored in start area of flash memory(0x000_0000) that can initialize a device, update firmware images, and integrity checks. Firmware image in Host can be transferred to RED4S through UART. During download process, keep away from other operation. Source code is programmed in bootloader.c. Bootloader code includes RCP, UART, Timer and Flash control.

If the reader receives start download command from Host, the reader enters the bootloader through reset and performs firmware update. Firmware update is performed only in the Application area (The Bootloader, IAP and Registry are unchanged) and even if a problem occurs during download, it can be recovered.

Also, every time booting, the image status is checked, and if there is an error in the image, the bootloader is automatically executed. When entering the bootloader, recovery is possible by updating a new application.

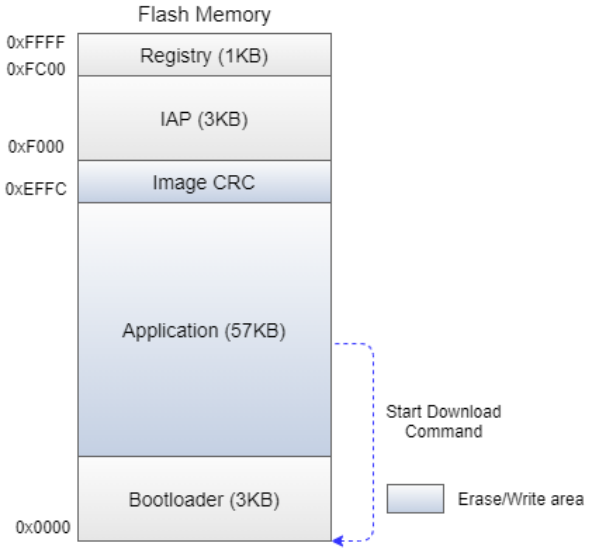


Figure 6 Bootloader Memory Map

6.11 Device drivers

6.11.1 MODEM

MODEM driver modem.c is used for loading MODEM registers settings. Initial register values are programmed at function modem_init. Some functions in modem.c are associated with air interface protocol and should be called in real time depending on user preference.

6.11.2 RF

RF driver rf.c is used for loading RF registers settings. Initial register values are programmed at function rf_init. Some functions are provided with rf.c to set registers associated with channel, transmitting power, and baseband filter bandwidth.

6.11.3 GPIO

By default, all port is used general purpose input, outputs or external interrupt. If you want to use alternative function, change the corresponding GPIO line by setting the alternative function select register. RED4S provides 5 external interrupts. Among them, external interrupt 0 and external interrupt 1 can be used for wakeup interrupt. Some GPIO is not available. Because module such as RED4 and RED5 use GPIOs for internal purpose. Please refer to each module spec sheet in order to check GPIO usage.

GPIO	Function	GPIO	Function
P00	UART0_RXD	P10	Ext.2
P01	UART0_TXD	P11	I2C SDA
P02	Ext.0	P12	I2C SCL
P03	Ext.1	P13	UART1_RXD
P04	SSP_TXDS	P14	UART1_TXD
P05	SSP_RXDS	P15	Ext.3
P06	SSP_SCK	P16	Ext.4
P07	SSP_SEL	P17	Ext.5

Table 1 GPIO Map

6.11.4 UART

RCP incoming packet calls UART interrupt handler. MCU calls UART interrupt service routine for every incoming byte. Incoming bytes are stored in buffer. UART interrupt service routine checks packet integrity using preamble, endmark, and payload length. After checking integrity, UART ISR generates a message or discards packet. If received packet has no error, post buffered packet to RCP. If certain error is occurred, received packet is discarded.
Below table indicates UART pin information.

Pin	Function
P00	UART0_RXD
P01	UART0_TXD
P13	UART1_RXD
P14	UART1_TXD

Table 2 UART Pin Information

For change serial interface or baudrate, rerogram new firmware after modify and compile the firmware.

- Modify firmware config.h file

```
051 // RCP Path
052 //#define __FEATURE_GPIO_SIO_SEL__
053 #define __FEATURE_UART_RCP__
054 //#define __FEATURE_SPI_SLAVE_RCP__
055 //#define __FEATURE_I2C_SLAVE_RCP__
```

Remove comment!

- Change baudrate

```
073 ////////////////////////////////////////////////////////////
074 // UART Feature
075 ////////////////////////////////////////////////////////////
076 #define UART0_BAUDRATE (115200)
```

Modify baudrate!

6.11.5 SPI

SPI operate as slave for RCP. RCP incoming packet calls SPI interrupt handler. MCU calls SPI interrupt service routine for every incoming byte. Incoming bytes are stored in buffer. SPI interrupt service routine checks packet integrity using preamble, endmark, and payload length. After checking integrity, SPI ISR generates a message or discards packet. If received packet has no error, post buffered packet to RCP. If certain error is occurred, received packet is discarded. To use slave master for other purpose, the user should change spi driver code spi.c.
Below table indicates SPI pin information.

Pin	Function
P04	SPI_TXDS
P05	SPI_RXDS
P06	SPI_CLK
P07	SPI_SEL
P16	IRQ (output, this pin is controlled by RED4S)

Table 3 SPI Pin Information

For change serial interface, reprogram new firmware after modify and compile the firmware

- Modify firmware config.h file

```
051 // RCP Path
052 //#define __FEATURE_GPIO_SIO_SEL__
053 //#define __FEATURE_UART_RCP__
054 #define __FEATURE_SPI_SLAVE_RCP__
055 //#define __FEATURE_I2C_SLAVE_RCP__
```

Remove comment!

[Notice] For the detail SPI procedure, refer to RED_RCP.pdf document.

6.11.6 I2C

I2C operate as slave for RCP. RCP incoming packet calls I2C interrupt handler. MCU calls I2C interrupt service routine for every incoming byte. Incoming bytes are stored in buffer. I2C interrupt service routine checks packet integrity using preamble, endmark, and payload length. After checking integrity, I2C ISR generates a message or discards packet. If received packet has no error, post buffered packet to RCP. If certain error is occurred, received packet is discarded. To use i2c master for other purpose, the user should re-initialize i2c as master.
Below table indicates I2C pin information.

Pin	Function
P11	I2C_SDA
P12	I2C_SCL
P16	IRQ (output, this pin is controlled by RED4S)

Table 4 I2C Pin Information

For change serial interface, reprogram new firmware after modify and compile the firmware

- Modify firmware config.h file

```
051 // RCP Path
052 //#define __FEATURE_GPIO_SIO_SEL__
053 //#define __FEATURE_UART_RCP__
054 //#define __FEATURE_SPI_SLAVE_RCP__
055 #define __FEATURE_I2C_SLAVE_RCP__
```

Remove comment!

[Notice] For the detail SPI procedure, refer to RED_RCP.pdf document.

6.11.7 Timer

RED4S has four programmable dual(16-bit, 32-bit) timer – Timer 0, Timer 1, Timer2 and Timer 3. Timer3 is used for user timer (application) such as LED time interval. To use timer3, user has to register and deregister callback function by using HAL function. Registered callback function is called when timer interrupt occurs. Timer 1 and Timer 2 are used for internal delay.

Timer	Function
TIMER 0	IAP, Bootloader
TIMER 1	Delay
TIMER 2	Delay for T1 time
TIMER 3	-

Table 5 Timer Map

6.12 Misc. services

6.12.1 Debug

Debug service provides text based user message which can be read through RCP. This function uses RCP message field as 0x02 (notification), and code as 0xDF (debug message). This service would be useful for monitoring internal variables in real time. Generated messages are shown in external software in PC. It supports the use of printf()-style argument (limitation of 50 bytes). Usage follows below.

debug_msg_str0 ("Counter Enable ");

or

debug_msg_str1 ("Counter : %d ", (int) cnt);

It is not recommended to use prototype directly. Instead of debug_msg_str_, use below functions such as debug_msg_str0, debug_msg_str1, debug_msg_str2, debug_msg_str2. If __DEBUG__ option is not defined, binary code size would be reduced.

```
#ifdef __DEBUG__
#define debug_msg_str0(a)          debug_msg_str_(a);
#define debug_msg_str1(a,b)        debug_msg_str_(a,b);
#define debug_msg_str2(a,b,c)      debug_msg_str_(a,b,c);
#define debug_msg_str3(a,b,c,d)    debug_msg_str_(a,b,c,d);
#else
#define debug_msg_str0(a)
#define debug_msg_str1(a,b)
#define debug_msg_str2(a,b,c)
#define debug_msg_str3(a,b,c,d)
#endif
```

7 API Reference

7.1 HAL

7.1.1 hal_init()

Prototype	SYS_S hal_init(void)
Description	Initialize peripheral, modem and rf module
Parameters	None
Returns	SYS_S_OK
Example	

7.1.2 hal_set_sys_deep_sleep()

Prototype	SYS_S hal_set_sys_deep_sleep (void)
Description	Set the system to the deep sleep mode.
Parameters	None
Returns	SYS_S_OK SYS_S_HAL_ERR_DEEP_SLEEP
Example	

7.1.3 hal_set_sys_sleep()

Prototype	SYS_S hal_set_sys_deep (void)
Description	Set the system to the deep sleep mode.
Parameters	None
Returns	SYS_S_OK
Example	

7.1.4 hal_get_region ()

Prototype	SYS_S hal_get_region (uint8 region)	
Description	Retrieves current region	
Parameters	region	0x11 – Korea 0x21 – US WIDE 0x22 – US NARROW 0x31 – EUROPE 0x41 – JAPAN 0x52 – CHINA2 0x61 - BRAZIL
Returns	SYS_S_OK	
Example		

7.1.5 hal_set_region ()

Prototype	SYS_S hal_set_region (uint8 region)	
Description	Set current region	
Parameters	region	0x11 – Korea 0x21 – US WIDE 0x22 – US NARROW 0x31 – EUROPE 0x41 – JAPAN 0x52 – CHINA2 0x61 - BRAZIL
Returns	SYS_S_OK SYS_S_HAL_ERR_SET_REGION	
Example		

``

7.1.6 hal_get_ch ()

Prototype	SYS_S hal_get_ch (uint8 * ch, uint8 * ch_ext)	
Description	Retrieves current channel	
Parameters	ch	Channel number. The range of channel number depends on regional settings
	ch_ext	Channel number offset for miller subcarrier. Japanese channel only
Returns	SYS_S_OK	
Example		

7.1.7 hal_set_ch ()

Prototype	SYS_S hal_set_ch (uint8 * ch, uint8 * ch_ext)	
Description	Configures current channel	
Parameters	ch	Channel number. The range of channel number depends on regional settings
	ch_ext	Channel number offset for miller subcarrier. Japanese channel only
Returns	SYS_S_OK SYS_S_HAL_ERR_SET_CH	
Example		

7.1.8 hal_get_fhlbt_prm ()

```
typedef struct hal_fhlbt_prm_type
{
        uint16   tx_on_time;
        uint16   tx_off_time;
        uint16   sense_time;
        int16    lbt_rf_level;
        uint8    fh_enable;
        uint8    lbt_enable;
        uint8    cw_enable;
} hal_fhlbt_prm_type;
```

Field	Description
tx_on_time	Indicates tx on time (1 = 1ms)
tx_off_time	Indicates tx off time (1 = 1ms)
sense_time	CW sense time for LBT (1 = 1ms)
lbt_rf_level	RF level for LBT (-dBm x 10)
fh_enable	Frequency hopping enable (1: enable, 0: disable)
lbt_enable	LBT enable (1: enable, 0:disable)
cw_enable	Use CW instead of modulation during inventory period(1: enable, 0: disable)

Prototype	SYS_S hal_get_fhlbt_prm(hal_fhlbt_prm_type *prm)	
Description	Retrieves FH and LBT parameters	
Parameters	*prm	Pointer of FH and LBT parameters
Returns	SYS_S_OK	
Example		

7.1.9 hal_set_fhlbt_prm ()

Prototype	SYS_S hal_set_fhlbt_prm (const hal_fhlbt_prm_type *prm)	
Description	Sets FH and LBT parameters	
Parameters	*prm	Pointer of FH and LBT parameters
Returns	SYS_S_OK SYS_S_HAL_ERR_SET_REGION	

Example	hal_fhlbt_prm_type prm;
	
	prm.tx_on_time = 400; //400ms
	prm.tx_off_time = 100; //100ms
	prm.sense_time = 10’ // 10ms	

``

	prm.lbt_rf_level = -740; // -74.0 dBm
	prm.fh_enable = HAL_FH_ON; // frequency hopping enable
	prm.lbt_enable = HAL_LBT_OFF; // lbt disable
	prm.cw_enable = HAL_CW_OFF; // cw disable
	
	hal_set_fhlbt_prm(&prm);

7.1.10 hal_set_rfidblk_pwr ()

Prototype	SYS_S hal_set_rfidblk_pwr (uint8 mode)	
Description	Enable/Disable the RFID block	
Parameters	mode	HAL_RFIDBLK_ON – RFID block on HAL_RFIDBLK_OFF – RFID block off
Returns	SYS_S_OK SYS_S_HAL_ERR_RFIDBLK_CTRL	
Example		

7.1.11 hal_set_cw ()

Prototype	SYS_S hal_set_cw (uint8 mode)	
Description	On/Off the CW	
Parameters	mode	HAL_CW_ON – cw on HAL_CW_OFF – cw off
Returns	SYS_S_OK SYS_S_HAL_ERR_CW_CTRL	
Example		

7.1.12 hal_get_tx_pwr_lvl ()

Prototype	SYS_S hal_get_tx_pwr_lvl (uint16 *pwr_pa)	
Description	Get current tx power level	
Parameters	*pwr_pa	Output power value
Returns	SYS_S_OK	
Example		

7.1.13 hal_set_tx_pwr_lvl ()

Prototype	SYS_S hal_set_tx_pwr_lvl (uint16 pwr_pa)	
Description	Set current tx power level	
Parameters	pwr_pa	Output power value
Returns	SYS_S_OK	
Example		

7.1.14 hal_get_freq_hopping_table ()

```
typedef struct hal_rf_freq_hopping_table_type
{
        uint8 size;
        uint8* table_ptr;
} hal_rf_freq_hopping_table_type;
```

Field	Description
size	Size of frequency hopping table
table_ptr	Pointer of frequency hopping table

Prototype	SYS_S hal_get_freq_hopping_table (hal_rf_freq_hopping_table_type *tbl)	
Description	Get channels from frequency hopping table	
Parameters	*tbl	Pointer of frequency hopping table

Returns	SYS_S_OK
Example	

7.1.15 hal_set_freq_hopping_table ()

Prototype	SYS_S hal_set_freq_hopping_table (hal_rf_freq_hopping_table_type *tbl)	
Description	Set channels to frequency hopping table	
Parameters	*tbl	Pointer of frequency hopping table
Returns	SYS_S_OK	
Example		

7.1.16 hal_get_rssi ()

Prototype	SYS_S hal_get_rssi (uint16 * rssi16)	
Description	Gets current rssi value	
Parameters	rssi16	RSSI level (-dBm x 10, decimal value)
Returns	SYS_S_OK	
Example		

7.1.17 hal_get_registry ()

Prototype	SYS_S hal_get_registry(uint16 reg_address, uint8 *ret_len, uint8 *ret_byte)	
Description	Retrieves a item from the registry	
Parameters	reg_address	Registry address
	*ret_len	Registry data length to get
	*ret_byte	Rigistry data
Returns	SYS_S_OK SYS_S_HAL_ERR_REG_ADD_NOT_EXIST	
Example		

7.1.18 hal_update_registry ()

Prototype	SYS_S hal_update_registry (void)
Description	Update the registry. Executes erase, write and verify operation sequentially
Parameters	None
Returns	SYS_S_OK SYS_S_HAL_ERR_ERASE_REGISTRY
Example	

7.1.19 hal_erase_registry ()

Prototype	SYS_S hal_erase_registry(void)
Description	Erase the entire registry.
Parameters	None
Returns	SYS_S_OK SYS_S_HAL_ERR_ERASE_REGISTRY
Example	

7.1.20 hal_start_download ()

Prototype	SYS_S hal_start_download(void)
Description	Start IAP process
Parameters	None
Returns	SYS_S_OK
Example	

7.1.21 hal_i2c_master_init ()

Prototype	SYS_S hal_i2c_master_init (void)
Description	Initialize I2C master
Parameters	None
Returns	SYS_S_OK
Example	

7.1.22 hal_i2c_master_read ()

Prototype	SYS_S hal_i2c_master_read (uint8 slave_add, uint8 *data, uint16 length)	
Description	Read a block of data from the I2C Interface	
Parameters	slave_add	i2c slave address
	data	A pointer of data to read
	length	Data length
Returns	SYS_S_OK SYS_S_HAL_ERR_I2C_NACK	
Example	uint8 data[2] = {0,}; hal_i2c_master_read(SLAVE_ADD, & data[0], 2);	

7.1.23 hal_i2c_master_write ()

Prototype	SYS_S hal_i2c_master_write (uint8 slave_add, uint8 *data, uint16 length)	
Description	Write a block of data to the I2C Interface	
Parameters	slave_add	i2c slave address
	data	A pointer of data to read
	length	Data length
Returns	SYS_S_OK OK SYS_S_HAL_ERR_I2C_NACK	
Example	uint8 data[2] = {0x01, 0x02}; hal_i2c_master_write(SLAVE_ADD, & data[0], 2);	

7.1.24 hal_timer_register ()

```
typedef struct hal_timer_type
{
        hal_timer_pf start_cb;
        hal_timer_pf stop_cb;
        hal_timer_load_pf load_cb;
        hal_timer_pf handler_cb;
}hal_timer_type;
```

Field	Description
start_cb	Callback function to start timer
stop_cb	Callback function for stop timer
load_cb	Callback function to configure time value
handler_cb	Callback function for timer interrupt service routine

Prototype	SYS_S hal_timer_register_cb(hal_timer_type* timer)	
Description	Register the timer callback function.	
Parameters	timer	Callback functions to use timer
Returns	SYS_S_OK	

	SYS_S_ERR_TIMER3_CB_EXIST

Example	hal_timer_type timer3 = {NULL, NULL, NULL, NULL};
	
	void led_init()
	{
	timer3.handler_cb = led_isr;
	
	if(hal_timer_register_cb(&timer3) != SYS_S_OK)
	return;
	
	…
	
	timer3.load_cb(LED_TIME_1S);
	timer3.start_cb();
	}
	
	void led_isr()
	{
	/* do something */
	}

7.1.25 hal_timer_deregister ()

Prototype	SYS_S hal_timer_deregister_cb()
Description	Deregister timer callback function
Parameters	None
Returns	SYS_S_OK
Example	

7.1.26 hal_get_temperature ()

Prototype	SYS_S hal_get_temperature(int8 *temp)	
Description	Retrieves current temperature	
Parameters	temp	Current temperature
Returns	SYS_S_OK	
Example		

7.1.27 hal_get_ant_port_bit ()

Prototype	SYS_S hal_get_ant_port_bit(int8 *port_bit)	
Description	Retrieves current antenna port bit	
Parameters	*port_bit	Antenna port bit
Returns	SYS_S_OK SYS_S_HAL_ERR_NULL_DEV_TYPE	
Example		

7.1.28 hal_set_ant_port_bit ()

Prototype	SYS_S hal_set_ant_port_bit(uint8 port_bit)	
Description	Set antenna port bit	
Parameters	port_bit	Antenna port bit
Returns	SYS_S_OK SYS_S_HAL_ERR_NULL_DEV_TYPE	
Example		

7.1.29 hal_rcp_uart_baudrate_control ()

Prototype	SYS_S hal_rcp_uart_baudrate_control(int speed)

Description	Change UART baudrate	
Parameters	Speed	baudrate
Returns	SYS_S_OK	
Example		

7.1.30 hal_set_opt_fh_table ()

Prototype	SYS_S hal_set_opt_fh_table(void)
Description	Set optimum frequency hopping table
Parameters	None
Returns	SYS_S_OK SYS_S_HAL_ERR_INVALID_PARAM
Example	

7.1.31 hal_get_fh_mode ()

Prototype	SYS_S hal_get_fh_mode(uint8 *data)	
Description	Retrieves current frequency hopping mode	
Parameters	*data	0 - Normal mode 1 - FH optimum mode
Returns	SYS_S_OK	
Example		

7.1.32 hal_set_fh_mode ()

Prototype	SYS_S hal_set_fh_mode(uint8 data)	
Description	Set frequency hopping mode	
Parameters	Data	0 - Normal mode 1 - FH optimum mode
Returns	SYS_S_OK SYS_S_HAL_ERR_INVALID_PARAM	
Example		

7.1.33 hal_get_gain_mode ()

Prototype	SYS_S hal_get_gain_mode(uint8 *mode)	
Description	Retrieves current gain mode	
Parameters	*mode	0 - High gain mode 1 - Low gain mode
Returns	SYS_S_OK	
Example		

7.1.34 hal_set_gain_mode ()

Prototype	SYS_S hal_set_gain_mode(uint8 mode)	
Description	Set gain mode	
Parameters	mode	0 - High gain mode 1 - Low gain mode
Returns	SYS_S_OK SYS_S_HAL_ERR_INVALID_PARAM	
Example		

7.1.35 hal_get_report_mode ()

Prototype	SYS_S hal_get_report_mode(uint8 *mode)	
Description	Retrieves current report mode	
Parameters	*mode	0 - Report in idle 1 - Report in read operation

Returns	SYS_S_OK
Example	

7.1.36 hal_set_report_mode ()

Prototype	SYS_S hal_set_report_mode(uint8 mode)	
Description	Set report mode	
Parameters	mode	0 - Report in idle 1 - Report in read operation
Returns	SYS_S_OK SYS_S_HAL_ERR_INVALID_PARAM	
Example		

7.1.37 hal_get_fh_ref_level ()

Prototype	SYS_S hal_get_fh_ref_level(uint8 *data)	
Description	Retrieves current FH reference level	
Parameters	*data	Current reference level
Returns	SYS_S_OK	
Example		

7.1.38 hal_set_fh_ref_level ()

Prototype	SYS_S hal_set_fh_ref_level(uint8 data)	
Description	Set FH reference level	
Parameters	data	Reference level
Returns	SYS_S_OK	
Example		

7.1.39 hal_get_multiant_seq ()

Prototype	SYS_S hal_get_multiant_seq(hal_ant_sequence_type *tbl)	
Description	Retrieves current antenna sequence	
Parameters	*tbl	Pointer of antenna sequence table
Returns	SYS_S_OK	
Example		

7.1.40 hal_set_multiant_seq ()

Prototype	SYS_S hal_set_multiant_seq(hal_ant_sequence_type *tbl)	
Description	Set antenna sequence	
Parameters	*tbl	Pointer of antenna sequence table
Returns	SYS_S_OK SYS_S_HAL_ERR_INVALID_PARAM	
Example		

7.2 RFID Protocol

7.2.1 protocol_init ()

Prototype	SYS_S protocol_init (void)
Description	Initialize the RFID protocol parameters
Parameters	None
Returns	SYS_S_OK
Example	

7.2.2 protocol_register_tag_report_cb ()

typedef void(*protocol_tag_report_cb_type)(uint8 rssi, uint8 len, uint8* data);

Field	Description
rssi	Tag rssi value
len	Tag ID length
data	Pointer of tag ID

Prototype	void protocol_register_tag_report_cb(protocol_tag_report_cb_type cb)	
Description	Register callback function to get tag ID.	
Parameters	cb	Callback function to register
Returns	SYS_S_OK	
Example		

7.2.3 protocol_register_tag_completed_cb ()

typedef void(*protocol_tag_completed_cb_type)(uint8 discontinued, uint8 detected);

Field	Description
discontinued	Indicates whether read operation is completed
detected	Indicates whether tag is detected

Prototype	void protocol_register_tag_completed_cb(protocol_tag_completed_cb_type cb)	
Description	Register callback function to inform whether inventory is completed.	
Parameters	cb	Callback function to register
Returns	SYS_S_OK	
Example		

7.2.4 protocol_get_c_sel_param ()

```
typedef struct protocol_c_prm_sel_type
{
   uint8   target     ;
   uint8   action     ;
   uint8   mem_bank    ;
   uint16  pointer    ;
   uint8   len     ;
   uint8   mask[32]   ;
   uint8   truncate   ;
} protocol_c_prm_sel_type;
```

Field	Description
target	Indicates what Tag's SL flag or inventoried flag, and specifies one of four sessions
action	Indicates whether matching Tags SL flag or inventoried flag
mem_bank	Specifies whether mask applies to EPC, TID, or User memory
pointer	References a memory bit address
len	memory range
mask	To determine if it is matching or non-matching.
truncate	Asserts truncate

Prototype	SYS_S protocol_get_c_sel_param(protocol_c_prm_sel_type **param)
Description	Retrieves ISO18000-6C select command paramters
Parameters	param | A pointer of parameter for SELECT command
Returns	SYS_S_OK
Example	protocol_c_prm_sel_type *sel; protocol_get_c_sel_param(&sel);

7.2.5 protocol_set_c_sel_param ()

Prototype	SYS_S protocol_set_c_sel_param (const protocol_c_prm_sel_type *param)
Description	Configures ISO18000-6C select command paramters.
Parameters	param | A pointer of parameter for SELECT command
Returns	SYS_S_OK

Example	protocol_c_prm_sel_type sel = {0,};
	
	sel.target = 0x00;
	sel.action = 0x00;
	sel.mem_bank = 0x01;
	sel.pointer = 0x0010;
	sel.truncate = 0x00;
	
	protocol_set_c_sel_param(&sel);

7.2.6 protocol_get_c_qry_param ()

```
typedef struct protocol_c_prm_qry_type
{
   uint8   dr         ;
   uint8   m          ;
   uint8   trext      ;
   uint8   sel        ;
   uint8   session    ;
   uint8   target     ;
   uint8   q          ;
} protocol_c_prm_qry_type;
```

Field	Description
dr	Sets the T=>R link frequency
m	Sets the T=>R data rate and modulation format
trext	Chooses the T=>R preamble is prepended with a pilot tone
sel	Chooses which Tags respond to query
session	Chooses a session for the inventory round
target	Selects whether inventory flag in the inventory round
q	Sets the number of slots in the round

Prototype	SYS_S protocol_get_c_qry_param(protocol_c_prm_qry_type **param)
Description	Retrieves ISO18000-6C query command paramters
Parameters	param | A pointer of parameter for QUERY command
Returns	SYS_S_OK
Example	protocol_c_prm_qry_type *qry;

	protocol_get_c_qry_param(&qry);

7.2.7 protocol_set_c_qry_param ()

Prototype	SYS_S protocol_set_c_qry_param(const protocol_c_prm_qry_type *param)	
Description	Retrieves ISO18000-6C query command paramters	
Parameters	param	A pointer of parameter for QUERY command
Returns	SYS_S_OK	

Example	protocol_c_prm_qry_type qry = {0,};
	
	qry.dr = 0x00;
	qry.m = 0x00;
	qry.trext = 0x01;
	qry.sel = 0x00;
	qry.session = 0x00;
	qry.target = 0x00;
	qry.q = 0x00;
	
	protocol_set_c_qry_param(&qry);	

7.2.8 protocol_get_modulation ()

Prototype	SYS_S protocol_get_modulation(uint8 *mode)	
Description	Retrieves current modulation mode	
Parameters	mode	0 - M4, 250KHz 1 - M8, 250KHz
Returns	SYS_S_OK	
Example		

7.2.9 protocol_set_modulation ()

Prototype	SYS_S protocol_set_modulation(const uint8 mode)	
Description	Configures modulation mode	
Parameters	mode	0 - M4, 250KHz 1 - M8, 250KHz
Returns	SYS_S_OK	
Example		

7.2.10 protocol_get_anticol_mode ()

Prototype	SYS_S protocol_get_anticol_mode(uint8 *mode)	
Description	Retrieves current anti-collision algorithm	
Parameters	mode	1 - Manual mode 3 - Auto mode
Returns	SYS_S_OK	
Example		

7.2.11 protocol_set_anticol_mode ()

Prototype	SYS_S protocol_set_anticol_mode(const uint8 mode)	
Description	Configures current anti-collision algorithm	
Parameters	mode	1 - Manual mode 3 - Auto mode
Returns	SYS_S_OK	
Example		

``

7.2.12 protocol_perform_inventory ()

```
typedef enum protocol_command_code_type
{
        CMD_INVENTORY_SINGLE,
        CMD_INVENTORY_MULTIPLE_CYCLE,
        CMD_INVENTORY_MULTIPLE_CYCLETIMENUM,
        CMD_INVENTORY_ONE_TAG,
        CMD_WRITE,
        CMD_BLOCK_WRITE,
        CMD_READ,
        CMD_BLOCK_ERASE,
        CMD_CODE_MAX
} protocol_command_code_type;

typedef struct rfid_inventory_prm_type
{
        protocol_command_code_type          cmd_sel;
        uint16   count;
        uint8    mtnu;
        uint8    mtime;
} rfid_inventory_prm_type;
```

Field	Description
count	The number of inventory round
cmd_sel	Command type
mtnu	maximum number of tag to read
mtime	maximum elapsed time to tagging

Prototype	SYS_S protocol_perform_inventory (const rfid_inventory_prm_type *param)	
Description	Performs inventory.	
Parameters	param	A pointer of parameter for inventory operation type
Returns	SYS_S_OK SYS_S_PRT_ERR_BUSY SYS_S_PRT_ERR_INVALID_PARAM	

Example	rfid_inventory_prm_type param;
	
	param.cmd_sel = CMD_INVENTORY_MULTIPLE_CYCLE;
	param.count = 10;
	param.mtnu = INVENTORY_MTNU_INFINITE;
	param.mtime = 0; // INVENTORY_MTIME_INFINITE;
	
	protocol_perform_inventory(&param);	

7.2.13 protocol_discontinue_inventory ()

Prototype	SYS_S protocol_discontinue_inventory (void)
Description	Discontinues inventory
Parameters	None
Returns	SYS_S_OK
Example	

7.2.14 protocol_convert_tagerrcode ()

Prototype	SYS_S protocol_convert_tagerrcode(uint8 err_code)
Description	Convert he backscattered an tag error code to system error code

``

Parameters	err_code	The backscattered an error code from Tag
Returns	SYS_S_PRT_ERR_MEM_OVERRUN SYS_S_PRT_ERR_MEM_LOCK SYS_S_PRT_ERR_TAG_LOW_POWER SYS_S_PRT_ERR_TAG_KNOWN	
Example		

7.2.15 protocol_read_or_block_erase_memory ()

```
typedef enum protocol_c_membank_type
{
        MEM_RESERVE = 0x00,
        MEM_EPC     = 0x01,
        MEM_TID     = 0x02,
        MEM_USER    = 0x03,
        MEM_TID_USER = 0x04
} protocol_c_membank_type;

typedef struct protocol_c_prm_read_erase_type
{
        uint8 target_id_len;
        uint8 *target_id_ptr;
        uint32 access_pw;
        protocol_c_membank_type mem_bank;
        uint16 start_add;
        uint8 word_count;
        uint8 cmd_sel;
        uint8 retry;
} protocol_c_prm_read_erase_type;
```

Field	Description
target_id_len	EPC length of target tag_buf
target_id_ptr	EPC pointer of target tag_buf
access_pw	Access password
mem_bank	Memory bank
start_add	Start address
word_count	Word count
cmd_sel	Command type (Read or BlockErase)
retry	The number of retry

Prototype	SYS_S protocol_read_or_block_erase_memory(const protocol_c_prm_read_erase_type *param, rfid_data_type *ret_data)	
Description	Read part or all of a tag's momory	
Parameters	param	A pointer of parameter for READ command
	ret_data	A pointer of backscattered data form TAG
Returns	SYS_S_OK SYS_S_PRT_ERR_BUSY SYS_S_PRT_ERR_INVALID_PARAM SYS_S_PRT_ERR_NO_TAG SYS_S_PRT_ERR_ACCESS_TAG SYS_S_PRT_ERR_READ_MEM	

Example	protocol_c_prm_read_erase_type param;
	rfid_data_type ret_data;
	SYS_S ret;
	
	param.access_pw = 0xACCEC0DE;
	param.target_id_len = 0x0C;
	param.mem_bank = MEM_EPC;
	param.start_add = 0x0000;
	param.word_count =0x02;	

``

	param.target_id_ptr = &target_id_data[0];
	param.cmd_code = CMD_READ;
	param.retry = 0;
	
	ret = protocol_read_or_block_erase_memory(&param,&ret_data);
	
	if(ret == SYS_S_OK)
	{
	/* Success */
	}
	else
	{
	/* Error */
	}

7.2.16 protocol_write_memory ()

```
typedef struct protocol_c_prm_write_type
{
        uint8 target_id_len;
        uint8 *target_id_ptr;
        uint32 access_pw;
        protocol_c_membank_type mem_bank;
        uint16 start_add;
        uint8 word_count;
        uint16 *word_data_ptr;
        uint8 cmd_sel;
        uint8 retry;
} protocol_c_prm_write_type;
```

Field	Description
target_id_len	EPC length of target tag
target_id_ptr	EPC pointer of target tag
access_pw	access password
mem_bank	memory bank
start_add	start address
word_count	word count
word_data_ptr	word data pointer
cmd_code	Command type (Write or BlockWrite)
retry	The number of retry

Prototype	SYS_S protocol_write_memory (const protocol_c_prm_write_type *param, rfid_data_type *ret_data)	
Description	Write a word in a tag's momory	
Parameters	*param	A pointer of parameter for WRITE command
	*ret_data	A pointer of target epc
Returns	SYS_S_OK SYS_S_PRT_ERR_BUSY SYS_S_PRT_ERR_INVALID_PARAM SYS_S_PRT_ERR_NO_TAG SYS_S_PRT_ERR_ACCESS_TAG SYS_S_PRT_ERR_WRITE_MEM	
Example	protocol_c_prm_write_type param; rfid_data_type ret_data; SYS_S ret; param.access_pw = 0xACCEC0DE;	

``

	param.target_id_len = 0x0C;
	param.mem_bank = MEM_EPC;
	param.start_add = 0x0000;
	param.word_count =0x02;
	param.target_id_ptr = &target_id_data[0]
	param.word_data_ptr = &word_data[0];
	param.cmd_sel = CMD_WRITE;;
	param.retry = 0;
	
	ret = protocol_write_memory(&param, &ret_data);

7.2.17 protocol_kill_memory ()

```
typedef struct protocol_c_prm_kill_type
{
        uint8 target_id_len;
        uint8 *target_id_ptr;
        uint32 kill_pw;
        uint8 recom;
        uint8 retry;
} protocol_c_prm_kill_type;
```

Field	Description
target_id_len	EPC length of target tag
target_id_ptr	EPC pointer of target tag
kill_pw	Kill password
recom	Recommissioning bits
retry	The number of retry

Prototype	SYS_S protocol_kill_memory (const protocol_c_prm_kill_type *param)	
Description	Kill a tag to permanently disable	
Parameters	param	A pointer of parameter for KILL command
Returns	SYS_S_OK SYS_S_PRT_ERR_BUSY SYS_S_PRT_ERR_INVALID_PARAM SYS_S_PRT_ERR_NO_TAG SYS_S_PRT_ERR_KILL_TAG	

Example	protocol_c_prm_kill_type param;
	SYS_S ret;
	
	param.kill_pw = 0xDEADC0DE;
	param.target_id_len = 0x0C;
	param.target_id_ptr = &target_id_data[0];
	param.recom = 0; // TBD
	param.retry = 0;
	
	ret = protocol_kill_memory(&param);	

7.2.18 protocol_lock_memory ()

```
typedef struct protocol_c_prm_lock_type
{
        uint8 target_id_len;
        uint8 *target_id_ptr;
        uint32 access_pw;
```

``

```
        uint32 mask_action;
        uint8 retry;
} protocol_c_prm_lock_type;
```

Field	Description
target_id_len	EPC length of target tag
target_id_ptr	EPC pointer of target tag
access_pw	access password
mask_action	Mask and action data
retry	The number of retry

Prototype	SYS_S protocol_lock_memory (const protocol_c_prm_lock_type *param)	
Description	Lock a tag to change lock status	
Parameters	param	A pointer of parameter for LOCK command
Returns	SYS_S_OK SYS_S_PRT_ERR_BUSY SYS_S_PRT_ERR_INVALID_PARAM SYS_S_PRT_ERR_NO_TAG SYS_S_PRT_ERR_ACCESS_TAG SYS_S_PRT_ERR_LOCK_TAG	

Example	protocol_c_prm_lock_type param;
	SYS_S ret;
	
	param.access_pw = 0xACCEC0DE;
	param.target_id_len = 0x0C;
	param.target_id_ptr = &target_id_data[0];
	param.mask_action = mask_action;
	param.retry = 0;
	
	ret = protocol_lock_memory(&param);	

``

7.3 Reader Control Protocol (RCP)

7.3.1 RCP Packet Structure

```
typedef struct
{
        uint8    preamble;
        uint8    msg_type;
        uint8    cmd_code;
        uint8    pl_length[2];
        uint8    payload[RCP_PKT_MAX_BUF_SIZ];
} rcp_pkt_type;

typedef union
{
        rcp_pkt_type pkt;
        uint8 byte_data[RCP_PKT_MAX_BUF_SIZ];
} rcp_rsp_type;

typedef rcp_rsp_type rcp_req_type;
```

7.3.2 rcp_init ()

Prototype	void rcp_init(void)
Description	Initialize RCP parameters
Parameters	None
Returns	None
Example	

7.3.3 rcp_dispatch_event ()

Prototype	void rcp_dispatch_event(EVENT e)
Description	Parses the RCP Packet Check CRC of RCP packet and call functions according to command code
Parameters	None
Returns	None
Example	

7.3.4 rcp_clear_req_buf ()

Prototype	void rcp_clear_req_buf(void)
Description	Clear the received rcp packet
Parameters	None
Returns	None
Example	

7.3.5 rcp_write_msg_nack ()

Prototype	void rcp_write_msg_nack(const uint8 msg_type, const uint8 fail_code)	
Description	Create the NACK message and send the message	
Parameters	msg_type	Indicate reader-to-user RCP packets type (Response or Notification)
	fail_code	A error code value when fail
Returns	None	

Example	if(SYS_S_OK != protocol_set_inventory_mode
	((typec_inventory_type)rcp_req_pkt_c.pkt.payload[0]))
	{
	rcp_write_msg_nack(RCP_MSG_RSP, FAIL_INVALID_PARM);	

	}
	else
	{
	rcp_write_msg_ack(RCP_MSG_RSP,RCP_CMD_SET_INV_MODE);
	}

7.3.6 rcp_write_msg_ack ()

Prototype	void rcp_write_msg_ack(const uint8 msg_type, const uint8 cmd_code)	
Description	Create the ACK message and send the message	
Parameters	msg_type	Indicate reader-to-user RCP packets type (Response or Notification)
	cmd_code	A code value of a corresponding command when success
Returns	None	

Example	if(SYS_S_OK != protocol_set_inventory_mode
	((typec_inventory_type)rcp_req_pkt_c.pkt.payload[0]))
	{
	rcp_write_msg_nack(RCP_MSG_RSP, FAIL_INVALID_PARM);
	}
	else
	{
	rcp_write_msg_ack(RCP_MSG_RSP,RCP_CMD_SET_INV_MODE);
	}	

7.3.7 rcp_write_string ()

Prototype	void rcp_write_string(const uint8 msg_type, const uint8 cmd_code, const uint8 *payload, const uint16 payload_size)	
Description	Create the ACK message and send the message	
Parameters	msg_type	Indicate reader-to-user RCP packets type (Response or Notification)
	cmd_code	A code value of a corresponding command when success
	payload	A pointer of data to be inserted into payload field of RCP packet
	payload_size	A Payload length
Returns	None	

Example	void rcp_get_rf_ch()
	{
	uint8 param[2];
	
	hal_get_ch(&param[0], &param[1]);
	
	rcp_write_string(RCP_MSG_RSP, RCP_CMD_GET_CH, &param[0], sizeof(param));
	}	

7.4 Event

7.4.1 event_init()

Prototype	void event_init(void)
Description	Initialize event.
Parameters	None
Returns	None
Example	

7.4.2 event_post ()

Prototype	void event_post(EVENT e)	
Description	Report tag ID of all inventoried tags	
Parameters	e	Event to be posted
Returns	None	
Example		

7.4.3 event_post_delayed ()

Prototype	void event_post_delayed(EVENT e, uint32 ms)	
Description	Used to post event after delayed for few millisecond	
Parameters	e	Event to be posted
	ms	Delay time (millisecond)
Returns	None	
Example		

7.4.4 event_flush ()

Prototype	void event_flush(event_handler h)	
Description	Flush all event corresponding to event handler	
Parameters	h	Event handler to remove
Returns	None	
Example		

8 Error Codes

Code	Value	Description
SYS_S_OK	0	Successfully complete.
SYS_S_HAL_ERR_INVALID_PARAM	-125	One of the HAL function parameters is invalid
SYS_S_HAL_ERR_RFIDBLK_CTRL	-123	The RFID block failed to control
SYS_S_HAL_ERR_CW_CTRL	-122	The CW failed to control
SYS_S_HAL_ERR_DEEP_SLEEP	-121	The device failed to deep sleep mode
SYS_S_HAL_ERR_SET_REGION	-120	Failed to set the region
SYS_S_HAL_ERR_SET_CH	-119	Failed to set the RF channel
SYS_S_HAL_ERR_ERASE_REGISTRY	-118	Erasing the all registry item failed
SYS_S_HAL_ERR_WRITE_REGISTRY	-117	Writing the registry failed
SYS_S_HAL_ERR_REG_ADD_NOT_EXIST	-116	Specified registry address is out of range
SYS_S_HAL_ERR_I2C_NACK	-115	Not acknowledged during I2C communication
SYS_S_HAL_ERR_NULL_DEV_TYPE	-114	The device type is not supported
SYS_S_HAL_ERR_GPIO_RESERVED	-113	The GPIO has already been reserved.
SYS_S_HAL_ERR_NULL_EXTINT_TYPE	-112	The external interrupt type is not supported
SYS_S_HAL_ERR_TIMER_CB_EXIST	-111	The Timer has already been registered
SYS_S_PRT_ERR_INVALID_PARAM	-59	One of the Protocol function parameters is invalid
SYS_S_PRT_ERR_BUSY	-58	Currently busy executing a previous request.
SYS_S_PRT_ERR_NO_TAG	-57	There is no responding tag
SYS_S_PRT_ERR_ACCESS_TAG	-56	Failed to access the tag memory
SYS_S_PRT_ERR_READ_MEM	-55-	Reading the tag memory failed
SYS_S_PRT_ERR_MEM_OVERRUN	-54	Memory overrun or unsupported PC value
SYS_S_PRT_ERR_MEM_LOCK	-53	Memory locked
SYS_S_PRT_ERR_TAG_LOW_POWER	-52	Insufficient powers
SYS_S_PRT_ERR_TAG_KNOWN	-51	Known error
SYS_S_PRT_ERR_WRITE_MEM	-50	Writing the tag memory failed
SYS_S_PRT_ERR_KILL_TAG	-49	Failed to kill the tag memory
SYS_S_PRT_ERR_LOCK_TAG	-48	Failed to lock the tag memory
SYS_S_PRT_ERR_READPROTECT_TAG	-47	Failed to execute the ReadProtect command
SYS_S_PRT_ERR_RESET_READPROTECT_TAG	-46	Failed to execute the Reset ReadProtect command
SYS_S_PRT_ERR_CHANGE_EAS_TAG	-45	Failed to execute the Change-EAS command
SYS_S_PRT_ERR_EAS_ALARM_TAG	-44	Failed to execute the EAS Alarm command
SYS_S_PRT_ERR_CALIBRATE_TAG	-43	Failed to execute the Calibrate command
SYS_S_PRT_ERR_ERASE_MEM	-42	Failed to erase the memory
SYS_S_PRT_ERR_INVALID_CMD_SEL	-41	Invalid command selection
SYS_S_PRT_ERR_TRSNPORT	-40	Failed to execute the Transport command
SYS_S_PRT_ERR_AUTHENTICATE_TAG	-39	Failed to execute the Authenticate command
SYS_S_PRT_ERR_NOT_SUPPORTED	-38	Not support error-specific codes
SYS_S_PRT_ERR_INVALID_CS	-37	Errors specified by the cryptographic suite
SYS_S_PRT_ERR_INSUFFICIENT_PWR	-36	The Tag has insufficient power to the operation
SYS_S_PRT_ERR_CRYPTO_SUITE	-35	Errors specified by the cryptographic suite
SYS_S_PRT_ERR_INSUFF_POWER	-34	The Tag has insufficient power to the operation
SYS_S_PRT_ERR_UNTRACEABLE_TAG	-33	Failed to execute the Untraceable command
SYS_S_PRT_ERR_READSIGNATURE_TAG	-32	Failed to execute the Readsignature command
SYS_S_PRT_ERR_CHANGESTATUSWORD_TAG	-31	Failed to execute the ChangeStatusWord command
SYS_S_PRT_ERR_GETUID_TAG	-30	Failed to execute the GetUid command
SYS_S_PRT_ERR_GETSENSORDATA_TAG	-29	Failed to execute the GetSensorData command

9 Appendix

9.1 Wakeup Interrupt

RED4S provides sleep mode. In case of sleep mode, not only internal interrupt like timer interrupt but external interrupt can be used. Below describes how to configure wakeup interrupt pin.

9.1.1 GPIO Configuration for wakeup from sleep mode

RED4S can use only P02 and P17 for wakeup. Because P02 is used to control External PA, P17 is used as default. How to configure wakeup interrupt pin is contained in the following table. Level trigger interrupt has to be used and IEV register can select whether trigger mode is high level or low level.

GPIO Register	Configuration	Note
Data Direction (DIR)	0: Input	
Interrupt Sense (IS)	1: Level Trigger	0: Edge Trigger, this is cannot used for wakeup
Interrupt Event (IEV)	0: Low Level Trigger Interrupt 1: High Level Trigger Interrupt	User selectable

Below a port of “pwrmgmt.c” in RED4S Firmware. If user want to modify wakeup interrupt, modify GPIO register value in red box.

```
void pwrmgmt_deepsleep()
{
    uint8 gpio_val[4];

    // Set DeppSleep Mode
    SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
    // Clock Control (STOP1 = 1 : StopOsc)
    SYSCON->CLOCKCON = (SYSCON->CLOCKCON & 0x3FFF) | 0x8000;
    // ClockCnt Delay  (temp value)
    SYSCON->CLOCKCNT = 0x00FF;

    // xp03 -> extint0
    gpio_val[0] = GPIO0->DIR;
    gpio_val[1] = GPIO0->IS;
    gpio_val[2] = GPIO0->IEV;
    gpio_val[3] = GPIO0->IE;

    GPIO0->DIR &= ~BIT3;     // Input
    GPIO0->IS |= BIT3;       // (1)Level Trigger, (0)Edge Trigger
    //GPIO0->IEV  &= ~BIT3; // (0)Low Level Trigger
    GPIO0->IEV  |= BIT3;     // (1)High Level Trigger
    GPIO0->IE |= BIT3;       // Interrupt Enable

    NVIC_EnableIRQ(WAKEINT_TX_IRQn);
    NVIC_EnableIRQ(EXT1_IRQn);

    __WFI();

    NVIC_DisableIRQ(EXT1_IRQn);
    NVIC_DisableIRQ(WAKEINT_TX_IRQn);


    GPIO0->DIR = gpio_val[0];
    GPIO0->IS |= gpio_val[1];
    GPIO0->IEV = gpio_val[2];
    GPIO0->IE = gpio_val[3];

}
```

9.2 Flash Copy Proection

In order to prevent copying the data of flash memory through SWD, user can copy protection macro.

9.2.1 How to protect firmware with SWD

Please call protection macro in flash.h

EFLASH_CHIP_ERASE ( )
- erase all sector of flash memory.
- Initialize copy protection function.

ELFASH_PROTECT_JTAG( )
- protect read operation of flash memory

10 Address Information

PHYCHIPS Inc.
#104, 187 Techno 2-ro, Yuseong-gu, Daejeon, Korea (Yongsan-dong, Migun Technoworld 2), 34025
http://www.phychips.com
sales@phychips.com
TEL: +82-42-864-2402
FAX: +82-42-864-2403