RZ/N1x Group (D, S, L)

Bare Metal Drivers

R11AN0282EJ0014
Rev.0.14
August 1, 2019

Introduction

This application note explains APIs of bare metal drivers for RZ/N1x Group.

Target Device

RZ/N1L, RZ/N1D, RZ/N1S

Related Documents

User's Manual: System Introduction, Multiplexing, Electrical and Mechanical Information (R01UH0750EJ)
User's Manual: System Control and Peripheral (R01UH0751EJ)
User's Manual: Peripherals (R01UH0752EJ)

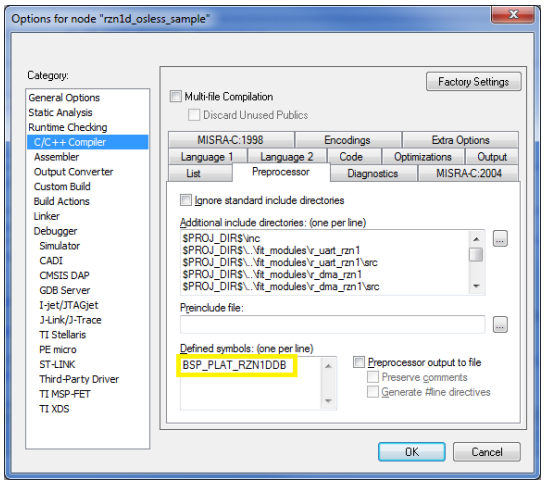
Contents

1. Overview

The RZN1 BSP contains a suite of drivers for the RZN1 family of device covering RZN1L, S and D variants.

The Included project workspace also includes a sample application (LED Blink) that utilizes the UART, GPIO, and I2C drivers, as well as demonstrating the system initialization.

The target device is configured by selecting the project to build within the IAR Workbench, and choosing Project -> Options from the menu, or by “right clicking” on the project



The define from the project, selects the required board, and includes the relevant hwsetup.c, and pin mux settings

1.1 BSP functions and modules

This package supports functions and modules shown below.

Driver function	Module name	Description
A5PSW	r_a5psw	Enables and Configures the 5 Port Switch
ADC	r_adc	ADC controller driver
Base system	r_bsp	System timer, board setup, etc.
IO multiplexer	r_iomux	Select multiplexed IO function
CAN	r_can	Configure and control CAN interface independently.
DMA	r_dma	Configure and control DMA interface independently.
GMAC	r_gmac	GMAC driver
GPIO	r_gpio	Configure direction, and control pin level, and control interrupt input function
I2C	r_iic	Configure baud-rate, and receive/transmit via I2C
MSEBI	r_msebi	Provides access to the MSEBI peripheral
QSPI	r_qspi	Configure QSPI peripheral and receive/transmit data from/to Flash memory vis QSPI.
RTC	r_rtc	RealTime Clock driver
SDIO	r_sdio	Control and configure SDIO interface.
Semaphore	r_semaphore	Provides access to the system semaphore module
SPI	r_spi	Configure baud-rate, and receive/transmit via SPI
TIMER	r_timer	Configure and control system Timer.
UART	r_uart	Configure baud-rate, and receive/transmit via UART
USB Function CDC	r_usb	Provides a CDC Class driver and USB Function driver
USB Host CDC & HUB	r_usb	Provides support for USB Host CDC & Hub classes
WDT	r_wdt	Configure (for all available cores).

1.2 BSP system architecture and hierarchy and initialization

1.2.1 CM3

The system initializes, and calls functions in __iar_program_start ( ) `in the file` **`start_m3.c`**

- bsp_interrupts_open();
    - Initialises the interrupt handlers
- SystemInit();
    - Configures the CM3 SHCSR
- If execution on the D board, then initialize the DDR RAM.
- __iar_data_init3();
    - This initializes the memory in the data sections
- SCB->VTOR = (uint32_t)__vector_table;
    - This replaces the vector table with the one defined in vector_table.c
- hardware_setup();
- main();

1.2.2 CA7

The system initializes, and calls functions in __iar_program_start ( ) `in the file` **`startup_ARMCA7.s`**

- Disables interrupts (“CPSID if”, Change Processor State Interrupt or abort disable IRQ & FIQ)
- Sets up the clocks
- Put any cores other than 0 to sleep
- Reset SCTLR Settings
- Configure ACTLR
- Set Vector Base Address Register (VBAR) to point to this application's vector table
- Setup Stack for each exception mode
- set_mmu
- enable_caches
- Initialise the GIC
- enableVGICPhysicalCPUInterface
- Enable interrupts (“CPSIE if”, Change Processor State Interrupt or abort enable IRQ & FIQ)
- SystemInit()
    - If execution on the D board, then initialize the DDR RAM.
    - The function SystemInit() is called in system_RZN1.c. This calls hardware_setup() in the device specific hardware setup file, this contains the hardware setup routine for the target platform.
    - bsp_interrupts_open();
        - Initialises the interrupt handlers
    - __iar_data_init3();
        - This initializes the memory in the data sections
    - hardware_setup();
    - main();

1.2.3 hwsetup

Hardware setup calls;

- clock_init();
- sys_timer_start();
    - The timer module is initialized to enable access to a timer, used by drivers for any non-blocking timer related functionality.
- R_SYSCTRL_EnableIOMUXLV2();
    - This enables the clock supply to enable access to level 2 multiplexed functionality (such as the UART).
- rzn1_board_pinmux(-1);
    - The pinmux is then configured as per the header file from the auto-generation tool for the board. Using a parameter value of -1 sets the board pinmux settings to the defaults in from the header from the PinMux Tool.
- output_ports_configure();
- interrupts_configure();
- peripheral_modules_enable();
- dma_mux_configure();

Now all drivers may be used, as their pins are pre-configured.

The IOMUX module contains R_IOMUX_PinCtrl(). This function may be used to re-define any pin functions.

1.3 Folder structure

Folder structure is shown below. There are drivers in ‘fit_modules’ folder.

```
Connect_it_RZN/Software
  |
  +---BaremetalDriver
       |
       |---r11an0282ejxxxx-rzn1-baremetal-drivers.pdf
       |
       +---+---src
       |   +---fit_modules
       |   |   +---r_a5psw_rzn1
       |   |   +---r_adc_rzn1
       |   |   +---r_bsp
       |   |   |   +---board
       |   |   |   |   +---rzn1d-db
       |   |   |   |   +---rzn1l-db
       |   |   |   |   +---rzn1s-db
       |   |   |   |   \---user
       |   |   |   +---inc
       |   |   |   |   \---iodefines
       |   |   |   \---mcu
       |   |   |       +---all
       |   |   |       \---rzn1
       |   |   |           +---ca7
       |   |   |           +---cm3
       |   |   |           +---ddr
       |   |   +---r_can_rzn1
       |   |   +---r_config
       |   |   +---r_dma_rzn1
       |   |   +---r_gmac_rzn1
       |   |   +---r_gpio_rzn1
       |   |   +---r_iic_rzn1
       |   |   +---r_iomux_rzn1
       |   |   +---r_lcdc_rzn1
       |   |   +---r_msebi_rzn1
       |   |   +---r_qspi_rzn1
       |   |   +---r_rtc_rzn1
       |   |   +---r_sdio_rzn1
       |   |   +---r_semaphore_rzn1
       |   |   +---r_spi_rzn1
       |   |   +---r_timer_rzn1
       |   |   +---r_uart_rzn1
       |   |   +---r_usb_rzn1
       |   |   |   +---r_usb_common_rzn1
       |   |   |   +---r_usbf_basic_rzn1
       |   |   |   +---r_usbf_cdc_rzn1
       |   |   |   +---r_usbh_basic_rzn1
       |   |   |   +---r_usbh_cdc_rzn1
       |   |   |   +---r_usbh_hub_rzn1
       |   |   \---r_wdt_rzn1
       |   +---osless_sample
       |   |   |   RZN1D_CA7.icf
       |   |   |   RZN1D_CM3.icf
       |   |   |   RZN1S_CA7.icf
       |   |   |   RZN1S_CM3.icf
       |   |   |   RZN1_CM3.icf
       |   |   \---src
```

```
|   |   |       adc_tests.c
|   |   |       can_tests.c
|   |   |       dma_tests.c
|   |   |       gmac_tests.c
|   |   |       gpio_tests.c
|   |   |       i2c_tests.c
|   |   |       lcdc_tests.c
|   |   |       log_serial_io.c
|   |   |       msebi_tests.c
|   |   |       qspi_tests.c
|   |   |       rtc_tests.c
|   |   |       r_usb_pcdc_apl.c
|   |   |       r_usb_pcdc_descriptor.c
|   |   |       sample_app.c
|   |   |       sdio_tests.c
|   |   |       semaphore_tests.c
|   |   |       spi_tests.c
|   |   |       system_tests.c
|   |   |       uart_tests.c
|   |   |       usbf_tests.c
|   |   |       usbh_tests.c
|   |   |       wdt_tests.c
|   |   +---rzn1d_debug
|   |   +---rzn1d_debug - CA7
|   |   +---rzn1l_debug
|   |   +---rzn1s_debug
|   |   +---rzn1S_debug - CA7
|   \---renesas_app
|       \---led_blink
|           +---log_serial_io.c
|           +---main.c
|           `---iar
|               `---7_70
|                   +---rzn1d_demo_board
|                   +---rzn1d_demo_board_ca7
|                   +---rzn1s_demo_board
|                   +---rzn1s_demo_board_ca7
|                   `---rzn1l_demo_board
|
\---utilities
    \---USB_CDC_WindowsDriver
        +---32bit
        \---64bit
```

2. API Information

This Driver API follows the Renesas FIT API naming standards.

2.1 Hardware Requirements

This BSP provides support the following MCU features:

- Interval timer peripheral, (Also used as system timer)
- A5PSW
- ADC
- CAN
- DMA
- GMAC
- GPIO
- I2C
- LCDC
- MSEBI
- QSPI
- RTC
- SDIO
- Semaphore
- SPI
- USB FUNCTION (CDC)
- USB HOST (CDC & Hub Class Support)
- UART
- WDT

2.2 Hardware Resource Requirements

This section details the hardware peripherals that this driver requires. Unless explicitly stated, these resources must be reserved for the driver and the user cannot use them.

2.2.1 IOMUX

This API provides a user-friendly API to the Pin MUX setting functionality.

The user can set pin function, drive strength, internal Pull-Up/down etc config.

Pins can be pre-configured using the PinMux tool (Connect_it_RZN\Tools\PinMux) by loading in the relevant rzn1[s/d/l]-db-pinmux.h containing the board specific code (\bare_metal_dev\src\fit_modules\r_bsp\board\rzn1[s/d/l]-db

2.2.2 System Control

Located under the mcu/rzn1 folder, this module is typically called from within other drivers to enable peripheral blocks prior to the drivers being used.

This driver enables/disables the clock to the peripherals, and configures peripheral clock dividers.

2.2.3 A5PSW

This driver enables the functionality of the 5 Port Switch peripheral.

2.2.4 ADC

This driver enables the functionality of the ADC peripheral.

2.2.5 CAN

This driver enables the functionality of the CAN peripheral.

2.2.6 DMA

This driver allows to use DMA independently (e.g. memory to memory transfers). In order to use DMA with other peripherals, the initialization and the DMA channel opening have to be done through this driver.

2.2.7 GMAC

This driver enables the functionality of the Independent GMAC peripheral.

2.2.8 GPIO

External GPIO pins can be multiplexed for various peripherals such as UART. This driver allows for common GPIO operations on external pins only when multiplexed for GPIO operation. Available GPIO pins are defined in r_gpio_rzn1(l, s, or d).h

2.2.9 I2C

This driver enables the functionality of the I2C peripheral.

2.2.10 LCDC

This driver enables functionality of the LCDC peripheral.

2.2.11 MSEBI

This driver enables the functionality of the MSEBI peripheral.

2.2.12 QSPI

This driver makes use of the QSPI peripheral to communicate with Flash memory.

The r_qspi_rzn1_config.h file allows the user to configure the default page size, and the clock divider.

2.2.13 RTC

This driver enables the functionality of the Real Time Clock peripheral.

2.2.14 SPI

This driver makes use of the SPI peripheral.

Interrupt threshold values can be modified in the “r_spi_rzn1_config.h” file.

2.2.15 Semaphore

This driver makes use of the Semaphore block.

2.2.16 SDIO

This driver makes use of the SDIO peripheral to communicate with external SD card.

2.2.17 Timer

This driver makes use of the Timer peripheral.

2.2.18 UART

This driver makes use of the UART peripheral

2.2.19 USB Function CDC Driver

This driver enables the functionality of the USB Function peripheral.

The USB function bare metal driver includes:

- USBf Basic module (HAL and USBf Core)
- USBf CDC module (CDC class driver for USBf)
- USB Common module (definitions and code common to all USB modules including USB Class modules which may be added in the future)

The USB Function CDC Sample application is included to demonstrate how to use the USBf driver

2.2.20 USB Host (CDC & Hub Class Support)

This driver enables the functionality of the USB Host peripheral.

The USB Host bare metal driver includes:

- USBh Basic module - Host stack with support for OHCI (Full-speed only) and EHCI (High-speed)
- USBh CDC class support
- USBh Hub class support

The USB Host CDC Sample application includes demonstrations of how to use with the the USBf CDC driver, stand-alone CDC devices and via a Hub.

2.2.21 WDT

This driver makes use of the Watchdog peripheral.

2.3 Limitations

No software limitations.

2.4 Development Environment

This driver is tested and working with the following:

Item	Description
Board	Renesas Electronics • RZ/N1L-DB Development Board • RZ/N1S-DB Development Board + Expansion Board • RZ/N1D-DB Development Board + Expansion Board
IDE	IAR Systems Embedded Workbench® for ARM (IAR EWARM 8.30.2) or later is installed, with the IAR EWARM compiler
Emulator	IAR Systems I-jet debugger

2.5 Header Files

All UART API calls and their supporting interface definitions are located in r_uart_rzn1_if.h. Likewise, all GPIO API calls and their supporting interface definitions are located in r_gpio_rzn1_if.h. Compile time configurable options are located in r_uart_rzn1_config.h and r_gpio_rzn1_config.h. All of these files should be included by the User's application.

2.6 Variable Types

2.6.1 Integer Types

This project uses ANSI C99 "Exact width integer types" in order to make the code clearer and more portable. These types are defined in stdint.h.

2.6.2 A5PSW Types

The 5 Port Switch is configured using the GMAC configuration defines

2.6.3 ADC Types

ADC Channel States

```
typedef enum
{
    ADC_STATE_NOT_INIT = 0,         /* ADC un-initialised */
    ADC_STATE_INIT,                 /* ADC initialised */
    ADC_STATE_CHANNEL_OPEN,         /* Channel initialised and idle */
    ADC_STATE_CHANNEL_CLOSED,       /* Channel uninitialised */
    ADC_STATE_CHANNEL_BUSY,         /* Channel currently being used for reading sample */
}
ADC_IF_STATE_E;
```

ADC Control Request

```
typedef enum
{
    ADC_CONTROL_SET_CHAN_CALLBACK,          /* Set channel callback for ADC interrupts */
    ADC_CONTROL_SET_CHAN_CONFIG,            /* Set channel configuration */
    ADC_CONTROL_GET_CHAN_CONFIG,            /* Get channel configuration */
    ADC_CONTROL_SET_CHAN_TRIGGER,           /* Set channel trigger */
    ADC_CONTROL_GET_CHAN_STATE,             /* Get channel state(see ADC_VC_CONFIG_T) */
    ADC_CONTROL_SET_DMA_CAPABILITY,         /* Set Global DMA Capability */
    ADC_CONTROL_SET_DMA_CALLBACK,           /* Set DMA callback */
    ADC_CONTROL_SET_SAMP_HOLD_CAPABILITY,   /* Set Global Sample & Hold Capability */
```

```
}
ADC_CONTROL_REQUEST_E;
```

Virtual Channel Trigger Enable

```
typedef struct
{
    ADC_VC_TRIG_E trig_trig_enable;
    ADC_VC_TRIG_SEL_E trig_select;
} ADC_VC_TRIG_CONFIG_T;
```

Buffer for ADC channels

```
typedef struct
{
        uint32_t channel[ADC_MAX_LOGICAL_CHANNELS];
} ADC_ALL_CHANNEL_BUFFER_T; /* Buffer size is 32bits to allow usage with DMA mode */
```

DMA Controller

```
typedef enum
{
    ADC_DMA_DISABLE = 0, ADC_DMA_ENABLE = 1,
} ADC_DMA_CONTROL_E;
```

ADC Sample Hold Controller

```
typedef enum
{
    ADC_SAMP_HOLD_DISABLE = 0, ADC_SAMP_HOLD_ENABLE = 7, /* Enable capability on all 3 Physical
Sample & Hold Channels */
} ADC_SAMP_HOLD_CONTROL_E;
```

Virtual Channel trigger select

```
typedef enum
{
    ADC_NOT_USED = 0x00,
    ADC_EOC_VC0 = 0x10,
    ADC_EOC_VC1 = 0x11,
    ADC_EOC_VC2 = 0x12,
    ADC_EOC_VC3 = 0x13,
    ADC_EOC_VC4 = 0x14,
    ADC_EOC_VC5 = 0x15,
    ADC_EOC_VC6 = 0x16,
    ADC_EOC_VC7 = 0x17,
    ADC_EOC_VC8 = 0x18,
    ADC_EOC_VC9 = 0x19,
    ADC_EOC_VC10 = 0x1a,
```

```
    ADC_EOC_VC11 = 0x1b,
    ADC_EOC_VC12 = 0x1c,
    ADC_EOC_VC13 = 0x1d,
    ADC_EOC_VC14 = 0x1e,
    ADC_EOC_VC15 = 0x1f,
} ADC_VC_TRIG_SEL_E;
```

Virtual Channel Trigger Enable

```
typedef enum
{
    ADC_TRIG_DISABLE = 0, ADC_TRIG_ENABLE = 1,
} ADC_VC_TRIG_E;
```

ADC Virtual Channel Configuration

```
typedef union
{
    uint32_t value;

    struct
    {
        unsigned long adc1_chan_sel :3;
        unsigned long adc2_chan_sel :3;
        unsigned long mode :2;
        unsigned long trig_select :5;
        unsigned long trig_enable :1;
        unsigned long run_mode :1;
        unsigned long adc_module :2;
        unsigned long dma_request :2;
    };
} ADC_VC_CONFIG_T;
```

ADC Virtual Channel

```
typedef union
{
    uint16_t value;

    struct
    {
        unsigned long vc0 :1;
        unsigned long vc1 :1;
        unsigned long vc2 :1;
        unsigned long vc3 :1;
        unsigned long vc4 :1;
        unsigned long vc5 :1;
        unsigned long vc6 :1;
```

```
        unsigned long vc7 :1;
        unsigned long vc8 :1;
        unsigned long vc9 :1;
        unsigned long vc10 :1;
        unsigned long vc11 :1;
        unsigned long vc12 :1;
        unsigned long vc13 :1;
        unsigned long vc14 :1;
        unsigned long vc15 :1;
    };
} ADC_VC_T;
```

ADC Transfer mode

```
typedef enum
{
    ADC_ACQ_MODE_BLOCKING = 0, /* Wait for conversion to complete */
    ADC_ACQ_MODE_CONTINUOUS, /* Same as blocking but if read at lower speed than that of the
acquisition time then a value should always be instantaneously available*/
    ADC_ACQ_MODE_NON_BLOCKING, /* Invoke callback when acquisition has completed */
    ADC_ACQ_MODE_DMA, /* Same as non-blocking but data is written to or read from DMA channel */
} ADC_ACQ_MODE_E;
```

ADC Required Transfer Information

```
typedef struct
{
    ADC_ACQ_MODE_E acq_mode; /* The mode of the acquisition */
    uint32_t * acq_buf; /* Where acquisition data is stored **if** using a blocking mode*/
} ADC_ACQ_CMD_T;
```

ADC Channel Transfer Completion Call-back Function Prototype for Non-Blocking mode

```
typedef void (*adc_chan_complete_callback_t)(uint8_t chan_num,
        uint32_t acq_value);
```

ADC DMA Completion Call-back Function Prototype

```
typedef void (*adc_dma_complete_callback_t)(void);
```

2.6.4 CAN Types

CAN Channel States

```
typedef enum
{
```

```
    CAN_STATE_NOT_INIT = 0,              /* CAN un-initialised */
    CAN_STATE_INIT,                      /* CAN initialised */
    CAN_STATE_CHANNEL_OPEN,              /* Channel initialised and idle */
    CAN_STATE_CHANNEL_CLOSED,            /* Channel uninitialised */
    CAN_STATE_CHANNEL_BUSY,              /* Channel currently being used for reading sample */
} CAN_IF_STATE_E;
```

CAN Control Request

```
typedef enum
{
    CAN_CONTROL_SET_CHAN_CALLBACK,       /* Set channel callback for CAN interrupts */
    CAN_CONTROL_GET_CHAN_STATE,          /* Get channel state*/
    CAN_CONTROL_GET_CHAN_CONFIG,         /* Get channel configuration */
    CAN_CONTROL_SET_CHAN_RESET,          /* Set channel into Reset state */
    CAN_CONTROL_CLEAR_CHAN_RESET,        /* Exit channel from Reset state */
    CAN_CONTROL_SET_CHAN_BITRATE,        /* Set channel Bitrate */
    CAN_CONTROL_SET_CHAN_FILTER,         /* Set channel Filter */
    CAN_CONTROL_START_CHAN_SYNC_PULSE,   /* Start channel Synch pulse (CANopen) */
    CAN_CONTROL_STOP_CHAN_SYNC_PULSE,    /* Stop channel Synch pulse (CANopen) */
    CAN_CONTROL_GET_CHAN_STATS,          /* Get channel  Communication Statistics */
    CAN_CONTROL_SET_TEST_MODE,           /* Enable/disable channel Test modes */
} CAN_CONTROL_REQUEST_E;
```

CAN Events

```
typedef enum
{
    CAN_EVENT_RX = 0,                    /* Message Received */
    CAN_EVENT_TX,                        /* Message Transmitted */
    CAN_EVENT_ERR_BUSOFF,                /* Bus is Off */
    CAN_EVENT_ERR_PASSIVE,               /* No acknowledge */
    CAN_EVENT_ERR_ACTIVE,                /* Bus OK */
    CAN_EVENT_ERR_OVERRUN                /* Receive message lost */
} CAN_EVENT_T;
```

CAN Bit-rate

```
typedef enum
{
    CAN_BR_20_KBITS   = 0,
    CAN_BR_50_KBITS   = 1,
    CAN_BR_100_KBITS  = 2,
    CAN_BR_125_KBITS  = 3,
    CAN_BR_250_KBITS  = 4,
    CAN_BR_500_KBITS  = 5,
    CAN_BR_800_KBITS  = 6,
    CAN_BR_1000_KBITS = 7,
```

```
    CAN_BR_INVALID    = 8
} CAN_BITRATE_T;
```

CAN Test Modes

```
typedef enum
{
    CAN_TMODE_NONE           = 0,       /* No tests mode active */
    CAN_TMODE_TX_NACK_OK     = 1,       /* TX requires no ACK to be seen as successful */
    CAN_TMODE_TX_SINGLE_SHOT = 2,       /* Do not attempt to repeat unsuccessful messages */
    CAN_TMODE_MONITOR_ONLY   = 4,       /* Listen to messages on the bus without issuing an ACK */
    CAN_TMODE_SELF_RX        = 8,      /* Receive own Tx messages */
} CAN_TEST_MODE_T;
```

CAN RTR Filter

```
typedef enum
{
    CAN_FRTR_LOW       = 0,                    /* Filter for non-RTR messages */
    CAN_FRTR_HIGH      = 1,                    /* Filter for RTR messages */
    CAN_FRTR_DONT_CARE = 2,                    /* Don't care if RTR is set or not */
} CAN_FILTER_RTR_T;
```

CAN Message Structure

```
typedef struct
{
    uint32_t id;                               /* Message identifier */
    bool rtr_flag;                             /* Remote Transmission requested */
    bool extended_flag;                        /* Extended Identifier */
    uint16_t data_len;                         /* Length of data */
    uint8_t data[CAN_DATA_LEN_MAX];            /* Message data */
} CAN_MSG_T;
```

CAN Synch Pulse Setup Structure

```
typedef struct
{
    CAN_MSG_T sync_msg;
    uint16_t sync_period_in_bits;
} CAN_SYNC_PULSE_T;
```

CAN Acceptance Filter Setup Structure

```
typedef struct
{
    uint32_t accept;                /* 1st Acceptance Filter */
    uint32_t mask;                  /* 1st Acceptance Filter Mask */
    bool is_extended;               /* Defines if filter relates to extended or standard ID */
```

```
    CAN_FILTER_RTR_T is_rtr;          /* Defines if also filtered on RTR flag */
} CAN_FILTER_T;
```

CAN Bus Metrics Structure

```
typedef struct
{
    uint32_t tx_msgs_total;
    uint32_t tx_msgs_std;
    uint32_t tx_msgs_ext;
    uint32_t rx_msgs_total;
    uint32_t rx_msgs_std;
    uint32_t rx_msgs_ext;
    uint32_t bus_off_errors;
    uint32_t bus_passive_errors;
    uint32_t bus_active_errors;
    uint32_t data_overrun_errors;
} CAN_STATS_T;
```

CAN Config Structure

```
typedef struct
{
    CAN_BITRATE_T bitrate;
    CAN_FILTER_T filter;
    CAN_SYNC_PULSE_T sync_msg;
    uint32_t test_mode;
} CAN_CONFIG_T;
```

CAN Event callback

```
typedef void (*can_event_callback_t)(uint8_t chan_num, CAN_EVENT_T event, CAN_MSG_T *ptr_msg);
```

2.6.5 DMA Types

DMA channel state

```
typedef enum
{
        DMA_CHANNEL_STATE_CLOSED = 0, /* Channel uninitialised */
        DMA_CHANNEL_STATE_OPEN, /* Channel initialised and idle */
        DMA_CHANNEL_STATE_BUSY /* Channel currently being used for transfer */
} DMA_CHANNEL_STATE;
```

DMA transfer mode and flow controller

```
typedef enum
{
        MTM = 0, /* Memory to memory - Flow controller: DMAC */
```

```
        MTP_DMAC = 1, /* Memory to peripheral - Flow controller: DMAC */
        PTM_DMAC = 2, /* Peripheral to memory - Flow controller: DMAC */
        PTP_DMAC = 3, /* Peripheral to peripheral - Flow controller: DMAC */
        PTM_P = 4, /* Peripheral to memory - Flow controller: Peripheral  */
        PTP_SP = 5, /* Peripheral to peripheral - Flow controller: Source Peripheral */
        MTP_P = 6, /* Memory to peripheral - Flow controller: Peripheral */
        PTP_DP = 7, /* Peripheral to peripheral - Flow controller: Destination Peripheral */
} DMA_TR_FLOW;
```

DMA transfer width

```
typedef enum
{
        BITS8 = 0, /* 1 Byte */
        BITS16 = 1, /* 2 Bytes */
        BITS32 = 2, /* 4 Bytes */
        BITS64 = 3, /* 8 Bytes */
} DMA_TR_WIDTH;
```

DMA transfer burst length (of width: DMA_TR_WIDTH)

```
typedef enum
{
        DATA_ITEMS_1 = 0, /* Single transaction per transaction request */
        DATA_ITEMS_4 = 1, /* 4 Data items per transaction request */
        DATA_ITEMS_8 = 2, /* 8 Data items per transaction request */
        DATA_ITEMS_16 = 3, /* 16 Data items per transaction request */
        DATA_ITEMS_32 = 4, /* 32 Data items per transaction request */
        DATA_ITEMS_64 = 5, /* 64 Data items per transaction request */
        DATA_ITEMS_128 = 6, /* 128 Data items per transaction request */
        DATA_ITEMS_256 = 7, /* 256 Data items per transaction request */
} DMA_BURST_LEN;
```

DMA handshaking type

```
typedef enum
{
        HARDWARE = 0, SOFTWARE = 1
} DMA_HS_TYPE;
```

DMA address incrementation

```
typedef enum
{
        INCREMENT = 0, DECREMENT = 1, NO_CHANGE = 2
} DMA_INCREMENTATION;
```

DMA handshaking setup

```
typedef struct
{
	DMA_HS_TYPE TYPE; /* software handshaking or hardware handshaking */
	bool INTERFACE_POLARITY; /* true = active low, false = active high */
	uint8_t HS_INTERFACE; /* Which DMA request channel to map the DMA channel to (see DMA Multiplexing register) To do: implement user manipulation of this register */
} DMA_HANDSHAKING;
```

DMA channel configuration

```
typedef struct
{
	uint8_t PRIORITY;
	DMA_HANDSHAKING SRC_HS;
	DMA_HANDSHAKING DST_HS;
} DMA_CHAN_CTRL;
```

DMA transaction configuration

```
typedef struct
{
	bool INT_EN;
	DMA_TR_FLOW TR_FLOW;
	uint32_t SRC;
	uint32_t DST;
	DMA_INCREMENTATION SRC_INCR;
	DMA_INCREMENTATION DST_INCR;
	DMA_TR_WIDTH SRC_TR_WIDTH;
	DMA_TR_WIDTH DST_TR_WIDTH;
	DMA_BURST_LEN SRC_BRST_LEN;
	DMA_BURST_LEN DST_BRST_LEN;
	uint16_t BLOCK_SIZE;
} DMA_TRANS_CTRL;
```

DMA Trasnfer request type

```
typedef enum
{
	DMA_TRANSFER, /* When hardware handshaking is enabled or memory to memory transfer is selected, use this to enable the channel for transfer*/
	DMA_SRC, /* Source software transaction request */
	DMA_SRC_SINGLE, /* Single source software transaction request */
	DMA_SRC_LAST, /* Last source transaction request */
	DMA_DST, /* Destination software transaction request */
	DMA_DST_SINGLE, /* Single destination software transaction request */
	DMA_DST_LAST, /* Last destination transaction request */
} DMA_TRANS_REQ_TYPE;
```

DMA control request

```
typedef enum
{
        DMA_CONTROL_GET_DRIVER_VERSION, /* Get the driver version (returns one byte) */
        DMA_CONTROL_SET_CALL_BACK, /* Set the call back function for interrupts */
        DMA_CONTROL_SET_CHAN_CONFIG, /* Set the configuration for the channel: flow control,
((priority)), handshaking */
        DMA_CONTROL_GET_CHAN_CONFIG, /* Get the configuration for the channel */
        DMA_CONTROL_SET_TRANS_CONFIG, /* Set the configuration for the transaction: source and
destination locations, incrementation, transfer widths, burst lengths, block size */
        DMA_CONTROL_GET_TRANS_CONFIG, /* Get the configuration for the transaction */
        DMA_CONTROL_GET_CHAN_STATE, /* Get the state of the channel: Open, Closed, Busy */
        DMA_CONTROL_PAUSE_TRANS, /* Cleanly suspend all DMA transfers on the channel (channel state
remains busy) */
        DMA_CONTROL_RESUME_TRANS, /* Unsuspend DMA transfers on the channel */
        DMA_CONTROL_FREE_CHAN /* Disables channel and sets the state to open */
} DMA_CONTROL_REQUEST;
```

DMA Callback function

```
typedef void (*dma_trans_complete_callback)(uint8_t DMAC_number, uint8_t channel);
```

2.6.6 GMAC

GMAC channel state

```
typedef enum
{
    GMAC_STATE_CLOSED = 0, /* Channel uninitialised */
    GMAC_STATE_OPEN, /* Channel initialised and idle */
} GMAC_STATE;
```

GMAC Control State

```
typedef enum {
    SET_SPEED,          /* Change speed in PHY, GMAC an RGMII/MII interface */
    SET_FILTER,         /* Change filtering setting for GMAC*/
    ETH_MUX_CTRL        /* Change control settings for ethernet switch mux */
} gmac_control_state_t;
```

GMAC Enable / Disable

```
typedef enum{
     DISABLE = 0,
     ENABLE = 1,
} gmac_control;
```

GMAC Filter MAC address should be used for source address filtering or destination

```
typedef enum{
     Destination_Address = 0,
     Source_Address = 1,
} gmac_comparison_field;
```

GMAC Operation modes: FD - full duplex, HD - half duplex

```
typedef enum {
    FD_1G,
    HD_1G,
    FD_100M,
    HD_100M,
    FD_10M,
    HD_10M,
} gmac_operation_mode_t;
```

GMAC Pointer to data of the last received packet

```
typedef struct {
    char *payload;
    uint64_t *dst_mac;
    uint64_t *src_mac;
    uint16_t *type;
} gmac_receive_buf;
```

GMAC Stores the speed of the operation on the link and received data of the last received packet

```
typedef struct {
    uint8_t chan_num;
    uint8_t port;
    gmac_operation_mode_t mode;
    gmac_receive_buf receive_buf;
    uint16_t *payload_length;
} gmac_transfer_data;
```

GMAC Filtering configurations

```
typedef struct {
    gmac_control receive_all;
    gmac_control sa_filter;
    gmac_control sa_inverse_filter;
    gmac_control da_inverse_filter;
    gmac_control disable_broadcast;
    gmac_control promiscuous_mode;
    gmac_comparison_field mac_comparison_field[17];     /*Tells whether the filter MAC address
should be used for source address filtering or destination */
```

```
    uint16_t msw_mac_addr[17];                          /*Most significant word of MAC address which
is desired to be used for filtering (18 address can be used for filtering, 1 of them is hard coded
with device MAC address*/
    uint32_t lsw_mac_addr[17];                          /*Most significant word of MAC address which
is desired to be used for filtering*/
    uint8_t number_of_filtered_addr;                    /*Number of added addresses*/
} gmac_filter_config;
```

GMAC Ethernet frame data

```
typedef struct {
    uint16_t dst_mac_msw;
    uint32_t dst_mac_lsw;
    uint16_t type;
    char *payload;
    uint16_t size;
} gmac_eth_frame;
```

GMAC Link Status

```
typedef enum
{
    LINK_DOWN = 0, LINK_UP = 1,
} link_status;
```

2.6.7 GPIO Types

gpio_pin_states_t

State of the GPIO pin (Whether or not it has been enabled as gpio).

```
typedef enum
{
    GPIO_PIN_STATE_CLOSED = 0, /* Pin uninitialized */
    GPIO_PIN_STATE_OPEN, /* Pin initialized */
} gpio_pin_states_t;
```

gpio_dir_t

Options that can be used with the R_GPIO_PortDirectionSet () and R_GPIO_PinDirectionSet () functions. Also the output of the R_GPIO_PinDirectionGet () function.

```
typedef enum
{
    GPIO_DIRECTION_INPUT = 0u,     /* GPIO Input */
    GPIO_DIRECTION_OUTPUT = 1u  /* GPIO Output */
} gpio_dir_t;
```

gpio_int_en_t

Used with the R_GPIO_InterruptEnable () function to Enable and Disable interrupts for a GPIO pin.

```
typedef enum
{
    GPIO_INTTERRUPT_DISABLED = 0u,    /* GPIO Pin Interrupt Disable */
```

```
    GPIO_INTTERRUPT_ENABLED = 1u      /* GPIO Pin Interrupt Enable */
} gpio_int_en_t;
```

gpio_int_mask_t

Used with the R_GPIO_InterruptMask () to mask or unmask interrupts for a GPIO pin.

```
typedef enum
{
    GPIO_INTTERRUPT_MASK_DISABLED = 0u,    /* GPIO Pin Interrupt Mask Disable (unmasked) */
    GPIO_INTTERRUPT_MASK_ENABLED = 1u      /* GPIO Pin Interrupt Mask Enable (masked) */
} gpio_int_mask_t;
```

gpio_int_trigger_type_t

Used with the R_GPIO_InterruptTriggerType () function to configure the GPIO pin interrupt to be edge triggered or level triggered.

```
typedef enum
{
    GPIO_INTTERRUPT_TRIGGER_LEVEL = 0u,    /* GPIO Pin Interrupt level triggered */
    GPIO_INTTERRUPT_TRIGGER_EDGE = 1u       /* GPIO Pin Interrupt edge triggered */
} gpio_int_trigger_type_t;
```

gpio_int_polarity_t

Used with the R_GPIO_InterruptTriggerPolarity () function to configure the polarity of the GPIO pin interrupt to be high or low. If the trigger type is edge triggered then low polarity refers to a falling edge and high polarity refers to a rising edge.

```
typedef enum
{
    GPIO_INTTERRUPT_POLARITY_LOW = 0u,    /* GPIO Pin Interrupt Polarity Low */
    GPIO_INTTERRUPT_POLARITY_HIGH = 1u    /* GPIO Pin Interrupt polarity High */
} gpio_int_polarity_t;
```

gpio_level_t

The logic level of the GPIO pins.

```
typedef enum
{
    GPIO_LEVEL_LOW = 0u,    /* Output/Input Level low */
    GPIO_LEVEL_HIGH = 1u    /* Output/Input Level high */
} gpio_level_t;
```

gpio_cmd_t

Used with the R_GPIO_PinControl () function to configure the properties of the GPIO pins.

```
typedef enum
{
    GPIO_CONTROL_GET_PIN_STATE,         /* Get the pin state */
    GPIO_CONTROL_SET_INT,               /* Enable or disable interrupts */
    /* A maximum of 8 A-Port pins can be enabled for interrupts at a time */
    GPIO_CONTROL_SET_INT_MASK,          /* Setup interrupt mask */
    GPIO_CONTROL_SET_INT_TRIG_TYPE,     /* Set the interrrupt trigger type */
    GPIO_CONTROL_SET_INT_TRIG_POL       /* Set the interrupt trigger polarity */
} gpio_cmd_t;
```

gpio_callback

Call back function for interrupts.

```
typedef void (*gpio_callback) (gpio_port_pin_t pin);
```

2.6.8 I2C Types

I2C Channel States

```
typedef enum
{
    I2C_CHANNEL_STATE_CLOSED = 0, /* Channel uninitialized */
    I2C_CHANNEL_STATE_OPEN, /* CHANNEL initialized and idle */
    I2C_CHANNEL_STATE_BUSY /* Channel currently being used for Read or Write */
} I2C_IF_CHANNEL_STATE_E;
```

I2C Transfer States

```
typedef enum
{
    I2C_TRANSFER_STATE_IDLE = 0, /* Not started */
    I2C_TRANSFER_STATE_IN_PROGRESS, I2C_TRANSFER_STATE_END, /* Transfer completed OK, transfer status not yet requested */
    I2C_TRANSFER_STATE_ERROR /* Transfer error, transfer status not yet requested */
} I2C_IF_TRANSFER_STATE_E;
```

I2C transfer mode

```
typedef enum
{
    I2C_TRANSFER_MODE_BLOCKING = 1, /* Wait for rx or tx transfer to complete (I2C interrupt disabled) */
    I2C_TRANSFER_MODE_NON_BLOCKING, /* Invoke callback when rx or tx transfer has completed (I2C interrupt enabled )*/
} I2C_TRANSFER_MODE_E;
```

I2C transfer structure

```
typedef struct
{
        I2C_TRANSFER_MODE_E transfer_mode; /* blocking or non-blocking i.e. interrupt disabled or enabled */
        uint32_t timeout_ms; /* max time to wait for data to be sent or received */
} i2c_transfer_config_t;
```

I2C configuration structure

```
typedef struct
{
       /* i2c channel configuration */
       bool master; /* TRUE if master mode, FALSE if slave mode*/
       bool restart_enable; /* True if Restart conditions may be sent - Master mode only */
```

```
        uint8_t addr_mode; /* 0: 7-bit addressing. 1: 10-bit addressing */
        uint8_t addr_mode_response_slave; /* 0: responds to 7-bit addressing. 1: responds to 10-bit
addressing - Slave mode only */
        uint8_t speed; /* 1: standard mode (100 kbits/sec). 2: fast mode (<= 400 kbits per sec) -
Master mode only */
        uint32_t slave_addr; /* Slave address - Slave mode only */
} I2C_CHANNEL_CONFIG_T;
```

I2C Control Request

```
typedef enum
{
    I2C_CONTROL_GET_DRIVER_VERSION, /* Get I2C Driver version */
    I2C_CONTROL_SET_CHAN_CONFIG, /* Set channel configuration */
    I2C_CONTROL_GET_CHAN_CONFIG, /* Get channel configuration */
    I2C_CONTROL_GET_CHAN_STATE, /* Get channel state (see I2C_IF_CHANNEL_STATE_E) */
    I2C_CONTROL_SET_COMPLETION_CALLBACK, /* Set I2C operation completion callback */
    I2C_CONTROL_RESET /* Reset the I2C channel */
} I2C_CONTROL_REQUEST_E;
```

Define for I2C Information structure type.

```
typedef struct
{
        uint8_t *p_tx_data; /* Pointer to transmit data buffer */
        uint8_t *p_rx_data; /* Pointer to receive data buffer */
        uint32_t data_len; /* Number of byes to send or receive */
} i2c_info_t;
```

I2C callback

```
typedef void (*i2c_transfer_complete_callback_t) (uint8_t chan_num, uint32_t total_bytes);
```

I2C Parameters Type

```
typedef struct
{
        uint32_t size; /* Total flash size */
        uint32_t page_size; /* Write (page) size */
} I2C_PARAMETERS_TYPE_T;
```

2.6.9 IOMUX Types

iomux_pin_properties_t

```
typedef struct
{
    uint8_t function;
    uint8_t drive_strength;
    uint8_t pull_level;
}iomux_pin_properties_t;
```

iomux_cmd_t

```
typedef enum
{
    IOMUX_CONTROL_GET_PIN_PROPERTIES,
    IOMUX_CONTROL_SET_PIN_PROPERTIES
} iomux_cmd_t;
```

2.6.10 LCDC Types

LCDC States

```
typedef enum
{
        LCDC_PORT_STATE_CLOSED = 0,     /* LCDC Port not configured and not enabled */
        LCDC_PORT_STATE_OPEN,           /* LCDC Port configured and enabled */
        LCDC_PORT_STATE_BUSY            /* LCDC port display currently being updated */
} lcdc_port_state_e;
```

Touch screen driver states

```
typedef enum
{
        TS_STATE_CLOSED = 0,            /* touch screen driver not configured & not enabled */
        TS_STATE_OPEN,                  /* touch screen driver configured and enabled */
} ts_state_e;
```

Touch screen events

```
typedef enum
{
        TS_EVENT_NONE = 0,              /* No touch screen event */
        TS_EVENT_PEN_DOWN,              /* first touch on touch screen */
        TS_EVENT_PEN_DRAG,              /* drag on touch screen */
    TS_EVENT_PEN_UP                 /* final touch on touch screen */
} ts_event_e;
```

LCDC Control Request

```
typedef enum
{
    LCDC_CONTROL_SET_PORT_CONFIG,           /* Set port configuration */
    LCDC_CONTROL_GET_PORT_CONFIG,           /* Get port configuration */
    LCDC_CONTROL_GET_PORT_STATE,
    LCDC_CONTROL_RESET                     /* Reset the LCDC  */
} lcdc_control_request_e;
```

LCDC Blink Mode

```
typedef enum
```

```
{
    LCDC_BLINK_MODE_OFF = 0,            /* Disable blink */
    LCDC_BLINK_MODE_SLOW,               /* Enable blink SLOW speed*/
    LCDC_BLINK_MODE_MEDIUM,             /* Enable blink MEDIUM speed*/
    LCDC_BLINK_MODE_FAST                /* Enable blink FAST */
} lcdc_blink_mode_e;
```

Pixel format

```
typedef enum
{
    LCD_RGB888,
    LCD_RGB666,
    LCD_RGB565,
    LCD_RGB555,
    LCD_BGR888,
    LCD_BGR666,
    LCD_BGR565,
    LCD_BGR555
} lcd_pixel_format_e;
```

LCDC Port Configuration

```
typedef struct
{
    uint32_t            bpp;            /* bits per pixel */
    lcd_pixel_format_e  pixel_format;
    uint32_t            x_res;          /* max pixels horizontally */
    uint32_t            y_res;          /* max pixels vertically */
    uint16_t           *palette;        /* palette lookup table values , required for
LCDC_DRIVER_BPP_1, LCDC_DRIVER_BPP_2, LCDC_DRIVER_BPP_4, LCDC_DRIVER_BPP_8 */
} lcdc_port_config_t;
```

Point pixel coordinates

```
typedef struct
{
    uint32_t lcd_point_x;
    uint32_t lcd_point_y;
} lcd_point_t;

/* Rectangle pixel coordinates  */
typedef struct
{
    uint32_t lcd_rectangle_left;
    uint32_t lcd_rectangle_top;
    uint32_t lcd_rectangle_right;
    uint32_t lcd_rectangle_bottom;
} lcd_rectangle_t;
```

Display buffer

```
typedef struct
{
    uint8_t            *display_buf_mem;        /* pointer to mem area for display buffer */
    uint32_t            display_buf_mem_size;
    uint32_t            display_x_res;          /* display buffer width in pixels */
    uint32_t            display_y_res;          /* display buffer height in pixels */
} lcdc_display_buffer_t;
```

Touch Screen event information

```
typedef struct
{
    ts_event_e          event;                  /* touch screen event */
    lcd_point_t         position;               /* touch position on the display */
} tsd_touch_event_info;
```

Touch Screen event callback

```
typedef void (*ts_event_callback_t)(tsd_touch_event_info *touch_screen_event, uint8_t
num_positions);
```

2.6.11 MSEBI TYPES

MSEBI Master/Slave mode

```
typedef enum {
        MSEBIM = 0,     /* MSEBI Master */
        MSEBIS          /* MSEBI Slave */
} msebi_mode_t;
```

State of the MSEBI --> Whether or not it has been opened

```
typedef enum
{
    MSEBI_STATE_CLOSED = 0,     /* MSEBI un-opened */
    MSEBI_STATE_OPEN            /* MSEBI opened */
} msebi_states_t;
```

Callback Mode

```
typedef enum
{
    TRANSFER = 0, /* Register callback function only for received transfer interrupts */
    RECEIVE = 1, /* Register callback function only for received receive interrupts */
    TRANFER_RECEIVE = 2  /* Register the same callback function for received transfer/receive
interrupts */
} msebi_callback_mode_t;
```

State of the MSEBI chip select --> Whether or not it has been opened

```
typedef enum
{
    MSEBI_CS_STATE_CLOSED = 0,  /* Chip select un-opened */
    MSEBI_CS_STATE_OPEN         /* Chip select opened */
} msebi_cs_states_t;
```

Cycle sizes of the different MSEBI phases

```
typedef struct {
    uint8_t write_dle_data_nb;  /* Size of data latch phase (0(1 MSEBI_CLK) - 255(256 MSEBI_CLK)) on
write cycle in no burst mode */
    uint8_t read_dle_data_nb;   /* Size of data latch phase (0(1 MSEBI_CLK) - 255(256 MSEBI_CLK)) on
read cycle in no burst mode */
    uint8_t write_dle_data_b;   /* Size of data latch phase (0(1 MSEBI_CLK) - 3(4 MSEBI_CLK)) on
write cycle in burst mode */
    uint8_t read_dle_data_b;    /* Size of data latch phase (0(1 MSEBI_CLK) - 3(4 MSEBI_CLK)) on
read cycle in burst mode */
    bool cle_data;           /* Size of control latch phase (False(1 MSEBI_CLK) - True(2 MSEBI_CLK))
*/
    bool ale_data;           /* Size of address latch phase (False(1 MSEBI_CLK) - True(2 MSEBI_CLK))
*/
} msebi_cyclesize_t;
```

Setup and hold timings for MSEBI data phase

```
typedef struct {
    uint8_t write_dle_setup;    /* Size of data phase setup (1(1 MSEBI_CLK) - 63(63 MSEBI_CLK)) on
write cycle */
    uint8_t read_dle_setup;     /* Size of data phase setup (1(1 MSEBI_CLK) - 63(63 MSEBI_CLK)) on
read cycle */
    uint8_t m_write_dle_hold;   /* Size of data phase hold (0(0 MSEBI_CLK) - 63(63 MSEBI_CLK)) on
write cycle (Master only) */
    uint8_t m_read_dle_hold;    /* Size of data phase hold (0(0 MSEBI_CLK) - 63(63 MSEBI_CLK)) on
read cycle (Master only) */
} msebi_setuphold_t;
```

The operating modes of MSEBI

```
typedef enum {
    MSEBIM_16_ASY = 0,   /* Asynchronous, 16 bit mode, no burst (Master only) */
    MSEBIM_16_SYN = 1,   /* Synchronous, 16 bit mode, burst available */
    MSEBIM_32_ASY = 2,   /* Asynchronous, 32 bit mode, no burst (Master only) */
    MSEBIM_32_SYN = 3,   /* Synchronous, 32 bit mode, burst available */
    MSEBIM_08_ASY = 4,   /* Asynchronous, 8 bit mode, no burst (Master only) */
    MSEBIM_08_SYN = 5,    /* Synchronous, 8 bit mode, burst available */
    MSEBIM_TRANS_MAX = 5,

    MSEBIS_16_SYN = 0,   /* Synchronous, 16 bit mode, burst available */
    MSEBIS_32_SYN = 1,   /* Synchronous, 32 bit mode, burst available */
```

```
    MSEBIS_08_SYN = 2,    /* Synchronous, 8 bit mode, burst available */
    MSEBIS_TRANS_MAX = 2
} msebi_trans_func_t;
```

MSEBI wait management

```
typedef enum {
    MSEBI_WAIT_CSN = 1,     /* Wait management on MSEBI_WAIT[n]_N pin */
    MSEBI_WAIT_CS0 = 2      /* Wait management on MSEBI_WAIT0_N pin */
} msebi_wait_mode_t;
```

MSEBI DMA access type

```
typedef enum {
    MSEBI_DMA_TRANSMIT = 1,
    MSEBI_DMA_RECEIVE = 2,
    MSEBI_DMA_WRITE = 1,
    MSEBI_DMA_READ = 2
} msebi_dma_access_t;


#define MSEBI_MAX_FIFO_LEVEL 32
#define MSEBI_MAX_DMA_TRANSFER_SIZE 8191
```

Max burst sizes between Master CPU FIFO and MSEBI bus

```
typedef enum {
    MSEBIM_BURST_1_WORD = 0,    /* 1 word burst max */
    MSEBIM_BURST_2_WORDS = 1,   /* 2 words burst max */
    MSEBIM_BURST_4_WORDS = 2,   /* 4 words burst max */
    MSEBIM_BURST_8_WORDS = 3,   /* 8 words burst max */
    MSEBIM_BURST_16_WORDS = 4,  /* 16 words burst max */
    MSEBIM_BURST_UNLIMITED = 5,  /* Not limited */
    MSEBIM_BURST_MAX = 5
} msebim_burstsize;
```

Max burst sizes between Slave CPU FIFO and MSEBI bus

```
typedef enum {
    MSEBIS_BURST_1_WORD = 0,    /* 1 word burst max */
    MSEBIS_BURST_4_WORDS = 1,   /* 4 words burst max */
    MSEBIS_BURST_8_WORDS = 2,   /* 8 words burst max */
    MSEBIS_BURST_16_WORDS = 3,   /* 16 words burst max */
    MSEBIS_BURST_MAX = 3
} msebis_burstsize;
```

MSEBI Master common configurations

```
typedef struct {
    uint8_t rx_fifo_data_level; /* Current recieve CPU FIFO level 1 word is 32 bits. READ FROM
REGISTER */
```

```
    uint8_t tx_fifo_data_level; /* Current transmit CPU FIFO level 1 word is 32 bits. READ FROM
REGISTER */

    uint8_t msebim_clk_low;     /* Time duration in units of MSEBIM_HCLK of MSEBIM_CLK level low
(0(1 MSEBIM_HCLK) - 3(4 MSEBIM_HCLK)) */

    uint8_t msebim_clk_high;    /* Time duration in units of MSEBIM_HCLK of MSEBIM_CLK level high
(0(1 MSEBIM_HCLK) - 3(4 MSEBIM_HCLK)) */

    msebim_burstsize cpu_write_burst; /* Max write burst size from CPU Tx FIFO to MSEBI bus */

    msebim_burstsize cpu_read_burst;  /* Max read burst size from MSEBI bus to CPU Rx FIFO */

} msebim_config_t;
```

MSEBI Slave common configurations

```
typedef struct {

    uint8_t rx_fifo_data_level; /* Current recieve CPU FIFO level 1 word is 32 bits. READ FROM
REGISTER */

    uint8_t tx_fifo_data_level; /* Current transmit CPU FIFO level 1 word is 32 bits. READ FROM
REGISTER */

    msebis_burstsize cpu_read_burst;  /* Max burst size of reads of Slave CPU Tx FIFO by MSEBI
Master */

    msebis_burstsize cpu_write_burst; /* Max burst size of writes from Slave CPU Rx FIFO to AHB
master */

    bool buffer_data;   /* true: data is bufferable, false: data is not bufferable */

    bool cache_data;    /* true; data is cacheable, false: data is not cacheable */

    uint8_t cfg_reg_timeout_delay_x4;   /* Timeout of register access delay = 4 *
(cfg_reg_timeout_delay_x4 + 1),(0(4 MSEBIS_HCLK) - 15(64 MSEBIS_HCLK)) */

} msebis_config_t;
```

General MSEBI chip select configurations

```
typedef struct {

    msebi_trans_func_t trans_func;

    msebi_wait_mode_t wait_func;

    bool burst_enable;        /* true: burst mode enable (Synchronous mode only), false: burst mode
disabled */

    bool addr_route_cs1;      /* true: map MSEBI_CS1_N on address line, false: no address routing */

    bool addr_route_cs2;      /* true: map MSEBI_CS2_N on address line, false: no address routing */

    bool addr_route_cs3;      /* true: map MSEBI_CS3_N on address line, false: no address routing */

} msebi_config_cs_t;
```

Master MSEBI chip select configurations

```
typedef struct {

    msebi_config_cs_t m_s_shared_config;

    bool wait_management;       /* true: wait management as defined in m_s_shared_config, false: No
wait management */

    uint8_t num_ale_used;       /* The number of phase MSEBI_ALE used (0-4) */

    bool parallel_ale_mode;     /* true: parallel mode, false: serial mode (only MSEBIM_ALE used for
all ALE cycles) */

    bool multi_dle;             /* true: multi DLE mode enabled, false: multi DLE mode disabled */

    uint8_t extend_addr;        /* Least significant 5 bits: b5 to b0 - MSEBI_A31 to MSEBI_A27 */

} msebim_config_cs_t;
```

Slave MSEBI chip select configurations

```
typedef struct {
    msebi_config_cs_t m_s_shared_config;
    bool write_access;  /* true: enable write on device, false: disable write on device */
    bool mmu_addr_mode; /* true: address management MMU mode, false: address anagement direct mode
*/
} msebis_config_cs_t;
```

Master MSEBI DMA control configurations for both transmit and receive.

```
typedef struct {
    bool single_dest_width;        /* Size of single transaction can be configured. READ FROM
REGISTER */
    uint16_t current_block_size;   /* Outstanding single transactions. READ FROM REGISTER */
    uint16_t block_size;           /* Destination/Source block transfer size 13bits 0-8191*/
    msebim_burstsize burst_size;   /* Max burst transaction size */
    bool enable_dma;               /* true: enable the DMA  false: disable the DMA */
} msebim_dma_control_cs_t;
```

Master MSEBI DMA data level configurations for both transmit and receive.

```
typedef struct {
    bool use_ext_pin;                  /* true: enable the control of DMA channel by external pins,
false: DMA starts immediately */
    msebim_burstsize fifo_burst_size;  /* Max burst size for access between DMA FIFO and MSEBI bus.
*/
    uint8_t current_fifo_data_level;   /* Current FIFO data level. 1 word is 64 bits. READ FROM
REGISTER */
    uint8_t fifo_data_level;           /* FIFO data level control. 0-31 with 28 being an optimal
value. */
} msebim_dma_datalevel_cs_t;
```

Slave MSEBI DMA data level configurations for both transmit and receive.

```
typedef struct {
    uint16_t current_fifo_data_level;  /* Current FIFO data level. 1 word is 32 bits. READ FROM
REGISTER */
    bool flow_control;                 /* true: enable the DMA flow control for write signal */
    bool burst_size_optimised;         /* Burst size optimization true: enabled */
    msebis_burstsize fifo_burst_size;  /* Max burst size for access between DMA FIFO and MSEBI bus.
*/
} msebis_dma_datalevel_cs_t;
```

Slave MSEBI from CPU interrupt status

```
typedef struct {
    bool eob_detected_dma_tx[MSEBI_CS_DMA_COUNT];  /* true: end of block detected for DMA transmit
chip 0,1 */
    bool eob_detected_dma_rx[MSEBI_CS_DMA_COUNT];  /* true: end of block detected for DMA receive
chip 0,1 */
    bool eob_detected_cpu_tx[MSEBI_CS_COUNT];  /* true: end of block detected for CPU transmit chip
0,3 */
    bool eob_detected_cpu_rx[MSEBI_CS_COUNT];  /* true: end of block detected for DMA receive chip
0,3 */
} msebis_interrupt_status_t;
```

Slave MSEBI from CPU interrupt control of mask and clear

```
typedef struct {
    bool eob_interrupt_dma_tx[MSEBI_CS_DMA_COUNT];  /* true: Mask or clear interrupt for DMA
transmit chip 0,1 */
    bool eob_interrupt_dma_rx[MSEBI_CS_DMA_COUNT];  /* true: Mask or clear interrupt for DMA receive
chip 0,1 */
    bool eob_interrupt_cpu_tx[MSEBI_CS_COUNT];  /* true: eMask or clear interrupt for CPU transmit
chip 0,3 */
    bool eob_interrupt_cpu_rx[MSEBI_CS_COUNT];  /* true: Mask or clear interrupt for DMA receive
chip 0,3 */
} msebis_interrupt_control_t;
```

Master MSEBI DMA read or write block address configurations.

```
typedef struct {
    bool align_first_block_32bit;       /* true: align the first block address as 32 bit, false:
align 64-bit */
    uint32_t address_dma_read;          /* First block address used by DMA controller to start a DMA
transfer. Only bits 3 to 31 used.*/
} msebim_dma_block_address_cs_t;


typedef struct {
    bool enable;                  /* true: enable the DMA ready to transmit or recieve */
    bool force;                   /* true: drive the DMA flow control signals */
} msebis_dma_request_cs_t;
```

MSEBI Control Request

```
typedef enum {
    MSEBI_CONTROL_SET_CONFIG,               /* Set the common configurations */
    MSEBI_CONTROL_GET_CONFIG,               /* Get the current common configurations */
    MSEBI_CONTROL_MASTER_CLK_ENABLE,        /* Enable/Disable MSEBIM_CLK clock generation */
    MSEBI_CONTROL_SET_CS_CYCLESIZE,         /* Set the chip select cycle size */
    MSEBI_CONTROL_GET_CS_CYCLESIZE,         /* Get the chip select cycle size */
    MSEBI_CONTROL_SET_CS_SETUPHOLD,         /* Set the chip select setup and hold times */
    MSEBI_CONTROL_GET_CS_SETUPHOLD,         /* Get the chip select setup and hold times */
    MSEBI_CONTROL_SET_CS_CONFIG,            /* Set the chip select configuration */
    MSEBI_CONTROL_GET_CS_CONFIG,            /* Get the chip select configuration */
    MSEBI_CONTROL_SET_CS_DMA_TDATALEVEL,    /* Set the DMA transmit data level */
    MSEBI_CONTROL_GET_CS_DMA_TDATALEVEL,    /* Get the DMA transmit data level  */
    MSEBI_CONTROL_SET_CS_DMA_RDATALEVEL,    /* Set the DMA receive data level */
    MSEBI_CONTROL_GET_CS_DMA_RDATALEVEL,    /* Get the DMA receive data level  */
    MSEBI_CONTROL_SET_CS_DMA_TCONTROL,      /* Set the DMA transmit control */
    MSEBI_CONTROL_GET_CS_DMA_TCONTROL,      /* Get the DMA transmit control  */
    MSEBI_CONTROL_SET_CS_DMA_RCONTROL,      /* Set the DMA receive control */
    MSEBI_CONTROL_GET_CS_DMA_RCONTROL,      /* Get the DMA receive control */
    MSEBIM_CONTROL_SET_CS_DMA_READADDR,     /* Set the DMA read address register */
    MSEBIM_CONTROL_GET_CS_DMA_READADDR,     /* Get the DMA read address register   */
```

```
    MSEBIM_CONTROL_GET_CS_DMA_CURR_READADDR, /* Get the DMA current read address register */
    MSEBIM_CONTROL_SET_CS_DMA_WRITEADDR,     /* Set the DMA write address register */
    MSEBIM_CONTROL_GET_CS_DMA_WRITEADDR,     /* Get the DMA write address register   */
    MSEBIM_CONTROL_GET_CS_DMA_CURR_WRITEADDR,/* Get the DMA current write address register */
    MSEBIM_CONTROL_FLUSH_CPU_FIFO,           /* Flush CPU Receive FIFO */
    MSEBIS_CONTROL_SLAVE_CS_ENABLE,          /* Enable/Disable MSEBI_CS[n]_N master request
receiving */
    MSEBIS_CONTROL_SET_CS_MMU_ADDRESS,       /* Set the MMU base address register */
    MSEBIS_CONTROL_GET_CS_MMU_ADDRESS,       /* Get the MMU base address register  */
    MSEBIS_CONTROL_SET_CS_MMU_ADDRESS_MASK,  /* Set the MMU base address mask register */
    MSEBIS_CONTROL_GET_CS_MMU_ADDRESS_MASK,  /* Get the MMU base address mask register */
    MSEBIS_CONTROL_SET_CS_DMA_TRANSMIT_REQ,  /* Set the DMA transmit request register */
    MSEBIS_CONTROL_GET_CS_DMA_TRANSMIT_REQ,  /* Get the DMA transmit request register */
    MSEBIS_CONTROL_SET_CS_DMA_RECEIVE_REQ,   /* Set the DMA receive request register */
    MSEBIS_CONTROL_GET_CS_DMA_RECEIVE_REQ,   /* Get the DMA receive request register */
    MSEBIS_CONTROL_GET_INT_STATUS,           /* Get interrupt status register values */
    MSEBIS_CONTROL_GET_MASKED_INT_STATUS,    /* Get masked interrupt status register values */
    MSEBIS_CONTROL_SET_MASK_INTERRUPT,       /* Set interrupt mask */
    MSEBIS_CONTROL_GET_MASK_INTERRUPT,       /* Get interrupt mask */
    MSEBIS_CONTROL_CLEAR_INTERRUPT,          /* Clear slave interrupt */
    MSEBIS_CONTROL_SET_CS_CPU_EOB_ADDRESS,   /* Set CPU End of Block address register value */
    MSEBIS_CONTROL_GET_CS_CPU_EOB_ADDRESS,   /* Get CPU End of Block address register value */
    MSEBIS_CONTROL_SET_CS_DMA_EOB_ADDRESS,   /* Set DMA End of Block address register value */
    MSEBIS_CONTROL_GET_CS_DMA_EOB_ADDRESS,   /* Get DMA End of Block address register value */

} msebi_cmd_t;
```

MSEBI callback function prototype

```
typedef void (*msebi_callback) (msebi_mode_t mode);
```

2.6.12 QSPI Types

QSPI Channel States

```
typedef enum {
    QSPI_CHANNEL_STATE_CLOSED = 0,           /* Channel uninitialised */
    QSPI_CHANNEL_STATE_OPEN,                 /* CHANNEL initialised and idle */
    QSPI_CHANNEL_STATE_BUSY                  /* Channel currently being used for Read or Write */
} QSPI_IF_CHANNEL_STATE_E;
```

QSPI Control Request

```
typedef enum {
    QSPI_CONTROL_GET_DRIVER_VERSION = 1,    /* Get QSPI driver version */
    QSPI_CONTROL_SET_TRANSFER_CONFIG,
    QSPI_CONTROL_GET_TRANSFER_CONFIG,
    QSPI_CONTROL_GET_CHAN_STATE,            /* Get channel state (see QSPI_IF_CHANNEL_STATE_E) */
    QSPI_CONTROL_RESET,                     /* Reset the QSPI channel */
} QSPI_CONTROL_REQUEST_E;
```

Enumeration for values of SPI operation mode.

```
typedef enum e_qspi_opmode
{
    QSPI_OPMODE_SINGLE_SPI_WRITE = 0,      /* Single-SPI mode for writing             */
    QSPI_OPMODE_SINGLE_SPI_READ,           /* Single-SPI mode for reading             */
    QSPI_OPMODE_SINGLE_SPI_WRITEREAD,      /* Single-SPI mode for writing and reading */
    QSPI_OPMODE_DUAL_SPI,                  /* Dual-SPI mode                           */
    QSPI_OPMODE_QUAD_SPI                   /* Quad-SPI mode                           */
} QSPI_OPMODE_E;
```

Define for data transfer mode.

```
typedef enum e_qspi_transfer_mode
{
    QSPI_TRANSFER_MODE_STIG = 0,    /* Software Triggered Instruction Generator (STIP) required to
access Config & Status SPI registers  & perform ERASE */
    QSPI_TRANSFER_MODE_DAC,         /* Direct Access Controller, mem-mapped-AHB accesses directly
trigger a read or write to flash memory */
    QSPI_TRANSFER_MODE_INDIRECT     /* Indirect mode */
} QSPI_TRANSFER_MODE_E;
```

Define for QSPI Information structure type.

```
typedef struct
{
    uint32_t                page_size;      /* Flash page size      */
    uint32_t                sector_size;    /* Flash sector size    */
    uint32_t                flash_size;     /* Flash total size     */
    QSPI_OPMODE_E           op_mode;        /* SPI operating mode   */
    QSPI_TRANSFER_MODE_E    tran_mode;      /* Data transfer mode   */
} qspi_info_t;
```

2.6.13 RTC

RTC States

```
typedef enum
{
    RTC_STATE_NOT_INIT = 0,         /* RTC un-initialised */
    RTC_STATE_INIT,                 /* RTC initialised */
    RTC_STATE_OPEN,                 /* RTC open */
    RTC_STATE_CLOSED,               /* RTC closed */
}
RTC_IF_STATE_E;
```

RTC Control Request

```
typedef enum
{
    RTC_CONTROL_GET_DRIVER_VERSION,         /* Get RTC Driver version */
    RTC_CONTROL_SET_CLOCK_CONFIG,           /* Set Clock configuration */
    RTC_CONTROL_GET_STATE,                  /* Get state (see RTC_IF_STATE_E) */
```

```
    RTC_CONTROL_SET_ALARM_CALLBACK,         /* Set RTC Alarm Callback */
    RTC_CONTROL_SET_PERIODIC_CALLBACK,      /* Set RTC One Second Callback */
    RTC_CONTROL_SET_ONE_SEC_CALLBACK,       /* Set RTC One Second Callback */
    RTC_CONTROL_RESET                       /* Reset the RTC */
} RTC_CONTROL_REQUEST_E;
```

RTC Hour Modes

```
typedef enum
{
    RTC_HOURS_AMPM = 0, /* hours reported in am/pm mode */
    RTC_HOURS_24HR = 1, /* hours reported in 24 hour mode */
} RTC_HOURS_MODE_E;
```

RTC Events

```
typedef enum
{
    RTC_ALARM_EVENT = 0, RTC_PERIODIC_EVENT, RTC_ONE_SEC_EVENT,
} RTC_EVENT_E;
```

RTC Periodic Events

```
typedef enum
{
    RTC_PERIODIC_NONE = 0,
    RTC_PERIODIC_QTR_SEC = 1,
    RTC_PERIODIC_HALF_SEC = 2,
    RTC_PERIODIC_ONE_SEC = 3,
    RTC_PERIODIC_ONE_MIN = 4,
    RTC_PERIODIC_ONE_HOUR = 5,
    RTC_PERIODIC_ONE_DAY = 6,
    RTC_PERIODIC_ONE_MTH = 7,
} RTC_PERIODIC_EVENT_E;
```

RTC Clock Calibration Modes

```
typedef enum
{
    RTC_CLK_CALIB_NONE = 0, RTC_CLK_CALIB_FINE, RTC_CLK_CALIB_COARSE,
} RTC_CLOCK_CALIB_MODE_E;
```

RTC Alarm Setup

```
typedef struct
{
        uint8_t minutes;
        uint8_t hours;
```

```
        union
        {
                uint8_t activeDays;

                struct
                {
                        bool sunday :1;
                        bool monday :1;
                        bool tuesday :1;
                        bool wednesday :1;
                        bool thursday :1;
                        bool friday :1;
                        bool saturday :1;
                };
        };
} RTC_ALARM_SETUP_T;
```

RTC Clock Calibration

```
typedef struct
{
        int32_t clkFrequency;
        RTC_CLOCK_CALIB_MODE_E mode;
} RTC_CLOCK_CALIB_T;
```

RTC Callback Function Prototype

```
typedef void (*rtc_event_callback_t) (RTC_EVENT_E event);
```

RTC Callback Request

```
typedef struct
{
        RTC_EVENT_E eventType;
        rtc_event_callback_t callback;

        union
        {
                RTC_ALARM_SETUP_T alarmSetup;
                RTC_PERIODIC_EVENT_E periodicSetup;
        };
} RTC_CALLBACK_REQUEST_T;
```

2.6.14 SDIO Types

SDIO Host controller state

```
typedef enum
```

```
{
    SDIO_STATE_CLOSED = 0, /* Controller uninitialised */
    SDIO_STATE_OPEN, /* Initialised controller, not busy */
    SDIO_STATE_BUSY /* Initialised and busy controller */
} SDIO_STATE;
```

Clock dividers for external SD clock - Base frequency = 50MHz

```
typedef enum
{
    SDIO_CLK_DIV_2046 = 11, /* Base frequency / 2046*/
    SDIO_CLK_DIV_1024 = 10, /* Base frequency / 1024*/
    SDIO_CLK_DIV_512 = 9, /* Base frequency / 512*/
    SDIO_CLK_DIV_256 = 8, /* Base frequency / 256*/
    SDIO_CLK_DIV_128 = 7, /* Base frequency / 128*/
    SDIO_CLK_DIV_64 = 6, /* Base frequency / 64*/
    SDIO_CLK_DIV_32 = 5, /* Base frequency / 32*/
    SDIO_CLK_DIV_16 = 4, /* Base frequency / 16*/
    SDIO_CLK_DIV_8 = 3, /* Base frequency / 8*/
    SDIO_CLK_DIV_4 = 2, /* Base frequency / 4*/
    SDIO_CLK_DIV_2 = 1, /* Base frequency / 2*/
    SDIO_CLK_DIV_BASE_FREQ = 0 /* Base frequency */
} SDIO_CLK_DIV;
```

Command response type

```
typedef enum
{
    SDIO_RESPONSE_NONE, /* No response */
    SDIO_RESPONSE_R1, /* Resopnse: R1 */
    SDIO_RESPONSE_R1b, /* Resopnse: R1b */
    SDIO_RESPONSE_R2, /* Resopnse: R2 */
    SDIO_RESPONSE_R3, /* Resopnse: R3 */
    SDIO_RESPONSE_R4, /* Resopnse: R4 */
    SDIO_RESPONSE_R5, /* Resopnse: R5 */
    SDIO_RESPONSE_R5b, /* Resopnse: R5b */
    SDIO_RESPONSE_R6, /* Resopnse: R6 */
    SDIO_RESPONSE_R7 /* Resopnse: R7 */
} SDIO_RESPONSE_TYPE;
```

Direction of data in transfer

```
typedef enum
{
    SDIO_WRITE = 0, /* Write to card */
    SDIO_READ = 1 /* Read from card */
} SDIO_DIRECTION;
```

Data transfer width

```
typedef enum
{
    SDIO_WIDTH_1_BIT, /* 1-bit mode */
```

```
    SDIO_WIDTH_4_BIT, /* 4-bit mode */
    SDIO_WIDTH_8_BIT /* 8-bit mode <- for embedded devices only */
} SDIO_TRANS_WIDTH;
```

Type of issued command to the card

```
typedef enum
{
    SDIO_CMD_NON_DATA = 0, /* Data line not used with command */
    SDIO_CMD_DATA = 1 /* Command uses data line */
} SDIO_COMMAND_TYPE;
```

Card type

```
typedef enum
{
    SD, /* SDSC card */
    SDHC_SDXC, /* SDHC or SDXC */
    SDIO, /* SDIO card */
    MMC, /* eMMC */
    INVALID_CARD /* Unknown card type */
} SDIO_CARD_TYPE;
```

SD card physical spec. version

```
typedef enum
{
    Version_1_0, /* SD physical spec. ver. 1.00 and 1.01 */
    Version_1_1, /* SD physical spec. ver. 1.10 */
    Version_2, /* SD physical spec. ver. 2.00 */
    Version_3, /* SD physical spec. ver. 3.0x */
    Version_4, /* SD physical spec. ver. 4.xx */
    Version_5, /* SD physical spec. ver. 5.xx */
    Version_6, /* SD physical spec. ver. 6.xx */
    Version_Unknown /* Unknown SD physical spec. version */
} SDIO_CARD_VERSION;
```

Information about the card sent by the card

```
typedef struct
{
        uint32_t OCR; /* Operating Conditions Register */
        uint32_t RCA; /* Relative Card Address */
        uint32_t CID[4]; /* Card ID Numbers */
        uint32_t CSD[4]; /* Card specific data */
        uint32_t SCR[2]; /* SD Config Register */
        SDIO_CARD_VERSION Card_Version; /* physical spec version */
} SDIO_CARD_PARAMETERS;
```

SDIO control request

```
typedef enum
```

```
{
    SDIO_CONTROL_GET_DRIVER_VERSION, /* Get the driver version number */
    SDIO_CONTROL_SEND_CMD, /* Send a command */
    SDIO_CONTROL_SEND_APP_CMD, /* Send an application command */
    SDIO_CONTROL_SET_INSERT_EJECT_CALLBACK, /* Set call back function */
    SDIO_CONTROL_GET_CARD_TYPE, /* Get the card type */
    SDIO_CONTROL_SET_CLK_DIV, /* Set the external clock frequency divider */
    SDIO_CONTROL_SD_INIT, /* Initialise card in SD mode */
    SDIO_CONTROL_SPI_INIT, /* Initialise card in SPI mode */
    SDIO_CONTROL_SET_BUS_WIDTH, /* Set the data bus width */
    SDIO_CONTROL_GET_CARD_PARAMS /* Returns a populated SDIO_CARD_PARAMETERS struct */
} SDIO_CONTROL_REQUEST;
```

Callback functions for interrupts

Transfer complete/error callback

typedef void (*sdio_callback) (uint16_t SDIO_NormalIntStatus, uint16_t SDIO_ErrorIntStatus);

Card inserted/ejected callback

typedef void (*sdio_insert_eject_callback) (bool SDIO_card_inserted, bool SDIO_card_ejected);

Transfer control

```
typedef struct
{
    void *buffer; /* Address of buffer to store data */
    uint16_t block_count; /* Number of blocks to be transfered */
    SDIO_DIRECTION direction; /* Direction of data */
    sdio_callback trans_complete_callback; /* Address of the callback function */
} SDIO_TRANS_CTRL;
```

Command info

```
typedef struct
{
        uint8_t command_index; /* CMD index */
        uint32_t argument; /* CMD argument */
        SDIO_RESPONSE_TYPE response_type; /* Expected response for CMD index */
        SDIO_COMMAND_TYPE command_type; /* Data or non-data command */
        SDIO_TRANS_CTRL command_data; /* Transfer control */
        uint32_t response[4]; /* CMD response buffer */
} SDIO_COMMAND;
```

2.6.15 Semaphore

Semaphore States

```
typedef enum
{
    SEMAPHORE_STATE_NOT_INIT = 0,           /* Semaphore un-initialised */
```

```
    SEMAPHORE_STATE_INIT,                   /* Semaphore initialised */
} SEMAPHORE_IF_STATE_E;
```

Semaphore Control Request

```
typedef enum
{
    SEMAPHORE_CONTROL_GET_STATE,      /* Get state (see SEMAPHORE_IF_STATE_E) */
    SEMAPHORE_CONTROL_GET_STATUS,     /* Get status of a Semaphore */
    SEMAPHORE_CONTROL_GET_CPU_ID,     /* Get CPU Id of processor currently running this code */
    SEMAPHORE_CONTROL_RESET           /* Reset the Semaphore */
} SEMAPHORE_CONTROL_REQUEST_E;
```

Semaphore Status Request

```
typedef struct
{
    SEMAPHORE_ID_E semId;
    SEMAPHORE_CPU_E cpuId;
} SEMAPHORE_STATUS_REQUEST_IF_T;
```

2.6.16 SPI Types

SPI Channel States

```
typedef enum {
    SPI_CHANNEL_STATE_CLOSED = 0,      /* Channel uninitialised */
    SPI_CHANNEL_STATE_OPEN,            /* Channel initialised and idle */
    SPI_CHANNEL_STATE_BUSY             /* Channel currently being used for Read or Write */
} spi_channel_state_t;
```

SPI Transfer mode

```
typedef enum {
    SPI_TRANSFER_MODE_BLOCKING = 0,    /* Wait for rx or tx transfer to complete */
    SPI_TRANSFER_MODE_NON_BLOCKING,    /* Invoke callback when rx or tx transfer has completed */
    SPI_TRANSFER_MODE_DMA,             /* Same as non-blocking but data is written to or read from
dma channel */
} spi_transfer_mode_t;
```

SPI Transfer type

```
typedef enum {
    SPI_TRANSFER_TYPE_TX_RX = 0,        /* Transmit and receive */
    SPI_TRANSFER_TYPE_TX = 1,           /* Transmit only */
    SPI_TRANSFER_TYPE_RX = 2,           /* Receive only */
} spi_transfer_type_t;
```

SPI Frame Format

```
typedef enum {
```

```
    spi_motorola = 0,                /* Motorola SPI frame format */
    spi_texas_instruments = 1,       /* Texas Instruments Synchronous Serial Protocol */
    spi_microwire = 2                /* National Semiconductor Microwire */
} spi_frame_format_t;
```

Motorola frame format serial clock phase

```
typedef enum {
    toggle_clk_mid_data_bit = 0,     /* Serial clock toggles in middle of first data bit */
    toggle_clk_start_data_bit = 1    /* Serial clock toggles at start of first data bit */
} spi_motorola_clk_phase_t;
```

Motorola frame format serial clock polarity

```
typedef enum {
    clk_inactive_low = 0,            /* Inactive state of serial clock is low */
    clk_inactive_high = 1            /* Inactive state of serial clock is high */
} spi_motorola_clk_pol_t;
```

SPI Motorola frame format config

```
typedef struct {
    spi_motorola_clk_phase_t    clk_phase;      /* serial clock phase */
    spi_motorola_clk_pol_t      clk_polarity;   /* serial clock polarity */
} spi_motorola_config_t;
```

Microwire frame format handshaking

```
typedef enum {
    microwire_handshaking_disabled = 0,         /* handshaking interface is disabled */
    microwire_handshaking_enabled = 1           /* handshaking interface is enabled */
} spi_microwire_master_handshaking_t;
```

Microwire frame format transfer mode

```
typedef enum {
    microwire_non_sequential_trans = 0,         /* Non-sequential transfer */
    microwire_sequential_trans = 1              /* Sequential transfer */
} spi_microwire_trans_mode_t;
```

Microwire frame format data direction

```
typedef enum {
    microwire_receive_data = 0,       /* The data word is sent to the external device */
    microwire_send_data = 1           /* The data word is received from the external device */
} spi_microwire_data_dir_t;
```

SPI Microwire frame format config

```
typedef struct {
    uint8_t control_frame_size;                         /* Control frame size: 1-16 bits */
    spi_microwire_master_handshaking_t  handshaking;    /* Handshaking configuration */
    spi_microwire_data_dir_t            data_direction; /* Data direction configuration */
    spi_microwire_trans_mode_t          transfer_mode;  /* Transfer mode configuration */
} spi_microwire_config_t;
```

SPI master channel config

```
typedef struct {
    uint32_t baudrate;          /* Divides SPI_SCLK, values: (even numbers) between 2 and 65534 */
    uint8_t slave_select;       /* Slave Select(0 to 3): Corresponds to SPI_SS_N lines [0:3] */
    uint8_t RXD_sample_delay;   /* SPI MISO sample delay in unit of SPI_SCLK 0-64 */
} spi_master_channel_config_t;
```

SPI Channel configuration

```
typedef struct {
    uint8_t                             data_frame_size;        /* Data frame size: 4-16 bits */
    spi_frame_format_t                  frame_format;           /* SPI Frame Format */
    spi_motorola_config_t               motorola_config;        /* Motorola frame format config --
Ignored when frame format is not Motorola */
    spi_microwire_config_t              microwire_config;       /* Microwire frame format config --
Ignored when frame format is not Microwire */
    spi_master_channel_config_t         master_config;          /* SPI master channel config --
Ignored by slave channels */
} spi_channel_config_t;
```

SPI Required transfer information

```
typedef struct {
    uint16_t *read_buf;                         /* Where data received is stored */
    uint16_t *write_buf;                        /* Data to be written */
    /* Each element in the read/write buffers is assumed to equal the data frame size.
       Thus, it is the user's responsibility to right-justify write data in the write buffer */
    uint16_t data_frames;                      /* Number of data frames to read/write -- if dma
mode is used, this is limited to 8191 */
    spi_transfer_mode_t transfer_mode_rx;       /* The transfer mode of the read */
    spi_transfer_mode_t transfer_mode_tx;       /* The transfer mode of the write */
    spi_transfer_type_t transfer_type;          /* The type of transfer being requested */
} spi_transfer_data_t;
```

SPI Transfer completion call-back function prototype

```
typedef void (*spi_transfer_complete_callback_t)(uint8_t chan_num);
```

SPI Control Request

```
typedef enum {
    SPI_CONTROL_SET_CALL_BACK,                  /* Set the call back function for SPI interrupts */
    SPI_CONTROL_SET_CHAN_CONFIG,          /* Set channel configuration */
    SPI_CONTROL_GET_CHAN_CONFIG,          /* Get channel configuration */
    SPI_CONTROL_RESERVE_DMA_CHAN_RX,    /* Reserve a DMA channel for use with the SPI channel Rx
transfers */
    SPI_CONTROL_RESERVE_DMA_CHAN_TX,    /* Reserve a DMA channel for use with the SPI channel Tx
transfers */
    SPI_CONTROL_RELEASE_DMA_CHAN_RX,      /* Release any reserved DMA channels from the SPI channel
Rx transfers */
    SPI_CONTROL_RELEASE_DMA_CHAN_TX,      /* Release any reserved DMA channels from the SPI channel
Tx transfers */
    SPI_CONTROL_GET_CHAN_STATE,           /* Get channel state (see spi_channel_state_t) */
    SPI_CONTROL_CANCEL_TRANSFER           /* Cancels an ongoing transfer */
} spi_cmd_t;
```

2.6.17 Sys Control

The SYSCTRL_Type defines the bit fields for the system control register. For the definition, refer to the SYSCTRL_SFR_table.h file.

2.6.18 Timer Types

Timer DMA Info

```
typedef struct
{
    uint8_t DMAC_number;
    uint8_t HS_Interface;
    uint8_t channel;
    bool dma_chan_reserved;
} timer_dma_info;
```

TIMER call-back function prototype

```
typedef void (*timer_interval_callback) (unsigned long interval_count);
```

2.6.19 UART Types

UART_IF_CHANNEL_STATE_E

```
UART channel states
typedef enum {
    UART_CHANNEL_STATE_CLOSED = 0,          /* Channel uninitialized */
    UART_CHANNEL_STATE_OPEN,                /* CHANNEL initialized and idle */
    UART_CHANNEL_STATE_BUSY                 /* Channel currently being used for Read or Write */
} UART_IF_CHANNEL_STATE_E;
```

UART_TRANSFER_MODE_E

```
UART transfer mode
typedef enum {
    UART_TRANSFER_MODE_BLOCKING = 1,        /* Wait for rx or tx transfer to complete */
    UART_TRANSFER_MODE_NON_BLOCKING,        /* Invoke callback when rx or tx transfer has completed
*/
    UART_TRANSFER_MODE_DMA,        /* Same as non-blocking but data is written to or read from dma
channel */
} UART_TRANSFER_MODE_E;
```

UART_PARITY_E

```
UART Parity settings
typedef enum {
    UART_PARITY_NONE = 0,
    UART_PARITY_ODD,
    UART_PARITY_EVEN
} UART_PARITY_E;
```

UART_FLOW_CONTROL_E

UART Flow control settings

```
typedef enum {
    UART_FLOW_CONTROL_NONE = 0,
    UART_FLOW_CONTROL_HW =1
} UART_FLOW_CONTROL_E;
```

UART number of stop bits

```
typedef enum
{
        NO_OF_STOP_BITS_1 = 0, /*!< 1 stop bits */
        NO_OF_STOP_BITS_2 = 1, /*!< 2 stop bits
         - If 5 bits per char, this is 1.5 stop bits */
} UART_STOP_BITS_E;
```

UART number of data bits per character

```
typedef enum
{
        DATA_BITS_PER_CHAR_5 = 0, /*!< 5 data bits per character */
        DATA_BITS_PER_CHAR_6 = 1, /*!< 6 data bits per character */
        DATA_BITS_PER_CHAR_7 = 2, /*!< 7 data bits per character */
        DATA_BITS_PER_CHAR_8 = 3, /*!< 8 data bits per character */
} UART_DATA_BITS_E;
```

UART_CHANNEL_CONFIG_T

Channel configuration structure

```
typedef struct
{
        UART_PARITY_E parity; /*!< Parity configuration */
        UART_FLOW_CONTROL_E flow_control; /*!< Flow control */
        UART_STOP_BITS_E stopbits; /*!< Number of stop bits */
        UART_DATA_BITS_E databits; /*!< Number of data bits per character */
        uint32_t baudrate; /*!< Transfer baudrate */
} UART_CHANNEL_CONFIG_T;
```

UART transfer complete callback

```
typedef void (*uart_transfer_complete_callback_t) (uint8_t chan_num, uint32_t total_bytes_received);
```

UART_CONTROL_REQUEST_E

UART Control Request

```
/* UART Control Request */
typedef enum {
    UART_CONTROL_GET_DRIVER_VERSION = 1,    /* Get UART driver version */
    UART_CONTROL_SET_CHAN_CONFIG,           /* Set channel configuration parity, flow_control,
stopbits, databits, baudrate */
    UART_CONTROL_GET_CHAN_CONFIG,           /* Get channel configuration */
    UART_CONTROL_SET_DMA_TX_CONFIG,         /* Set DMA block size and burst size for transmit mode
(Memory to peripheral)*/
```

```
    UART_CONTROL_SET_DMA_RX_CONFIG,         /* Set DMA block size and burst size for receive mode
(Peripheral to memory)*/
    UART_CONTROL_GET_DMA_TX_CONFIG,         /* Get DMA configuration for transfer mode */
    UART_CONTROL_GET_DMA_RX_CONFIG,         /* Get DMA configuration for receive mode */
    UART_CONTROL_GET_CHAN_STATE,            /* Get channel state (see UART_IF_CHANNEL_STATE_E) */
    UART_CONTROL_TRANSFER_CANCEL,           /* Cancel current transfer */
    UART_CONTROL_RESET,                     /* Reset the UART channel */
} UART_CONTROL_REQUEST_E;
```

2.6.20 USB FUNCTION CDC TYPES

USB Basic

USB error Type

```
typedef uint16_t        USB_ER_t;
```

USB UTR Type

```
typedef struct USB_UTR  USB_UTR_t;
```

USB Standard Callback Type

```
typedef void (*USB_UTR_CB_t)(USB_UTR_t *);
```

USB Class Callback Type

```
typedef void (*USB_CB_t)(uint16_t);
```

USB Request Type

```
typedef struct
{
    uint16_t        ReqType;            /* Request type */
    uint16_t        ReqTypeType;        /* Request type TYPE */
    uint16_t        ReqTypeRecip;       /* Request type RECIPIENT */
    uint16_t        ReqRequest;         /* Request */
    uint16_t        ReqValue;           /* Value */
    uint16_t        ReqIndex;           /* Index */
    uint16_t        ReqLength;          /* Length */
    uint16_t        align;              /* align structure on 32-bit boundary */
} USB_REQUEST_t;


struct USB_UTR
{
    uint16_t        keyword;        /* Rootport / Device address / Pipe number */
    USB_UTR_CB_t    complete;       /* Call Back Function Info */
    void            *tranadr;       /* Transfer data Start address */
    uint32_t        tranlen;        /* Transfer data length */
    uint16_t        status;         /* Status */
    uint16_t        pipectr;        /* Pipe control register */
};
```

Class request processing function type

```
typedef void (*USB_CB_TRN_t)(USB_REQUEST_t* preq, uint16_t ctsq);
```

Driver Registration

```
typedef struct USB_PCDREG
{
    uint16_t        **pipetbl;      /* Pipe Define Table address */
    uint8_t         *devicetbl;     /* Device descriptor Table address */
    uint8_t         *qualitbl;      /* Qualifier descriptor Table address */
    uint8_t         **configtbl;    /* Configuration descriptor Table address */
    uint8_t         **othertbl;     /* Other configuration descriptor Table address */
    uint8_t         **stringtbl;    /* String descriptor Table address */
    USB_CB_t        devdefault;     /* Device default callback */
    USB_CB_t        devconfig;      /* Device configured callback */
    USB_CB_t        devdetach;      /* Device detach callback */
    USB_CB_t        devsuspend;     /* Device suspend callback */
    USB_CB_t        devresume;      /* Device resume callback */
    USB_CB_t        interface;      /* Interface changed callback */
    USB_CB_TRN_t    ctrltrans;      /* Control Transfer callback */
    uint16_t        registered;     /* set true when a PCD application has been registered */
} USB_PCDREG_t;
```

1.1.1.1 CDC Class Driver

```
typedef struct st_unist
{
    union
    {
        unsigned long WORD;
        struct
        {
                    /* DCD signal */
                    uint16_t                    bRxCarrier:1;
                    /* DSR signal */
                    uint16_t                    bTxCarrier:1;
                    /* State of break detection mechanism of teh device */
                    uint16_t                    bBreak:1;
                    /* State of ring signal detection of the device */
                    uint16_t                    bRingSignal:1;
                    /* Framing error */
                    uint16_t                    bFraming:1;
                    /* Parity error */
                    uint16_t                    bParity:1;
                    /* Over Run error */
                    uint16_t                    bOverRun:1;
                    /* reserve */
                    uint16_t                    rsv:9;
                } BIT;

       }u_USB_SCI_SerialState;
```

```
}
USB_SCI_SerialState_t;

typedef struct
{
    /* Data terminal rate, in bits per second */
    uint32_t                    dwDTERate;
    /* Stop bits */
    uint8_t                     bCharFormat;
    /* Parity */
    uint8_t                     bParityType;
    /* Data bits */
    uint8_t                     bDataBits;
    /* reserve */
    uint8_t                     rsv;
}
USB_PCDC_LineCoding_t;          /* note: make sure is 32-bit aligned */

typedef struct
{
    /* DTR */
    uint16_t                    bDTR:1;
    /* RTS */
    uint16_t                    bRTS:1;
    /* reserve */
    uint16_t                    rsv:14;
}
USB_PCDC_ControlLineState_t;
```

2.6.21 USB Host CDC Class Types

USB Host CDC Class Events

```
typedef enum
{
    EVENT_HCDC_NONE,
    EVENT_HCDC_CONFIGURED,
    EVENT_HCDC_DETACH,
    EVENT_HCDC_CLASS_REQUEST_START,
    EVENT_HCDC_CLASS_REQUEST_COMPLETE,
    EVENT_HCDC_RD_START,
    EVENT_HCDC_RD_COMPLETE,
    EVENT_HCDC_WR_START,
    EVENT_HCDC_WR_COMPLETE,
    EVENT_HCDC_COM_NOTIFY_RD_START,
    EVENT_HCDC_COM_NOTIFY_RD_COMPLETE,
}
```

```
EVENT_HCDC_t;
```

USB Host CDC Control Operations

```
typedef enum
{
    USB_HCDC_CONTROL_SET_CALLBACK,
    USB_HCDC_CONTROL_SET_CHAN_CONFIG,
    USB_HCDC_CONTROL_GET_CHAN_CONFIG,
    USB_HCDC_CONTROL_SEND_BREAK,
}
USB_HCDC_CONTROL_REQUEST_t;
```

USB Host CDC Class Event Callback

```
typedef void (* USB_HCDC_CB_t)(uint16_t devadr, EVENT_HCDC_t event,
                               const uint8_t * data, uint16_t length);
```

2.6.22 USB Host Hub Class Types

USB Host Hub Class Events

```
typedef enum
{
    EVENT_HHUB_NONE,
    EVENT_HHUB_CONFIGURED,
    EVENT_HHUB_DETACH,
    EVENT_HHUB_DEVICE_ATTACH,
    EVENT_HHUB_DEVICE_DETACH,
}


EVENT_HHUB_t;
```

USB Host Hub Class Event Callback

```
typedef void (* USB_HHUB_CB_t)(uint16_t devadr, EVENT_HHUB_t event,
const void * data, uint16_t length);
```

2.6.23 WDT Types

WDT State

```
typedef enum
{
    WDT_STATE_CLOSED = 0, /* WDT uninitialised */
    WDT_STATE_OPEN, /* WDT initialised and idle */
    WDT_STATE_BUSY /* WDT is curently counting */
} WDT_STATE;
```

WDT prescaler configuration

```
typedef enum
{
  WDT_CLOCK_DIVIDE_2 = 0,
  WDT_CLOCK_DIVIDE_2EXP14 = 1
} WDT_PRESCALER;
```

WDT Output Type

```
typedef enum
{
  WDT_SYSTEM_RESET = 0,
  WDT_INTERRUPT
} WDT_OUTPUT;
```

WDT Control Request

```
typedef enum
{
  WDT_SET_PRESCALER,
  WDT_GET_PRESCALER,
  WDT_SET_RELOAD_VALUE,
  WDT_GET_RELOAD_VALUE,
  WDT_SET_OUTPUT,
  WDT_GET_OUTPUT,
  WDT_SET_CALLBACK
} WDT_CONTROL_REQUEST;
```

2.7 Configuration Overview

2.7.1 Platform Configuration

Supported boards are RZ/N1L-DB, RZ/N1D-DB, RZ/N1S-DB

Macro definition	Value	Description
BSP_PLAT_RZN1DDB	1	For RZ/N1D-DB board
BSP_PLAT_RZN1SDB	1	For RZ/N1S-DB board
BSP_PLAT_RZN1LDB	1	For RZ/N1L-DB board

The IAR project defines a macro for compilation for both the Core;

CORE_CM3 or CORE_CA7

And for the device (used to include the correct SFR header files)

RZN1S, RZN1D, RZN1L

This is used in src\fit_modules\r_bsp\mcu/rzn1/rzn1.h to include the correct header file to the project for the core

```
#ifdef CORE_CM3
#include "core_cm3.h"                    /* Cortex-M3 processor and core peripherals
*/
#endif
#ifdef CORE_CA7
#include "core_ca.h"                     /* Cortex-A processor and core peripherals
*/
#endif
```

And for the Platform in src\fit_modules\r_bsp\platform.h ;

BSP_PLAT_RZN1SDB, BSP_PLAT_RZN1DDB, BSP_PLAT_RZN1LDB

```
#ifdef BSP_PLAT_RZN1DDB
/* rzn1d-db */
#include "./board/rzn1d-db/r_bsp.h"
#endif

#ifdef BSP_PLAT_RZN1SDB
/* rzn1s-db */
#include "./board/rzn1s-db/r_bsp.h"
#endif

#ifdef BSP_PLAT_RZN1LDB
/* rzn1l-db */
#include "./board/rzn1l-db/r_bsp.h"
#endif
```

These include the platform (development board) specific header files for the target.

2.7.2 Driver Configuration

All configurable options that can be set at build time are in the files in the r_config folder.

A summary of these settings is provided in the following table:

Interrupt Priority

The default interrupt priority used during opening a channel and registering the interrupt and is defined in the r_xxxx_rzn1_config.h file;

Macro definition	Default Value
ADC_DEFAULT_INT_PRIORITY	9
CAN_DEFAULT_INT_PRIORITY	9
DMA_DEFAULT_INT_PRIORITY	7
GMAC_DEFAULT_INT_PRIORITY	7
GPIO_DEFAULT_INT_PRIORITY	7
I2C_DEFAULT_INT_PRIORITY	12
MSEBI_DEFAULT_INT_PRIORITY	7
RTC_DEFAULT_INT_PRIORITY	12
SDIO_DEFAULT_INT_PRIORITY	10
SPI_DEFAULT_INT_PRIORITY	9
TIMER_DEFAULT_INT_PRIORITY	0
UART_DEFAULT_INT_PRIORITY	1
USB_INT_PRIORITY	9
USB_HOST_INT_PRIORITY	9

2.8 Code Size

The code size is based on optimization level low for EWARM 8.32. The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options set in the module configuration header file.

The code size is reported at the end of each MAP file in

RZN1S CA7

```
\src\osless_sample\rzn1S_debug - CA7\List\rzn1S_osless_sample - CA7.map
    179 860 bytes of readonly  code memory
     66 403 bytes of readonly  data memory
  3 745 678 bytes of readwrite data memory
```

RZN1S CM3

```
\src\osless_sample\rzn1s_debug\List\rzn1s_osless_sample.map
    178 006 bytes of readonly  code memory
     65 176 bytes of readonly  data memory
  3 727 289 bytes of readwrite data memory
```

RZN1L CM3

```
\src\osless_sample\rzn1l_debug\List\rzn1l_osless_sample.map
  112 532 bytes of readonly  code memory
   43 978 bytes of readonly  data memory
  546 861 bytes of readwrite data memory
```

RZN1D CA7

```
src\osless_sample\rzn1d_debug - CA7\List\rzn1d_osless_sample - CA7.map
    182 056 bytes of readonly  code memory
     67 870 bytes of readonly  data memory
  3 745 681 bytes of readwrite data memory
```

RZN1D CM3

```
\src\osless_sample\rzn1d_debug\List\rzn1d_osless_sample.map
    180 150 bytes of readonly  code memory
     66 652 bytes of readonly  data memory
  3 727 292 bytes of readwrite data memory
```

2.9 API Data Structures

The API data structures are located in the file “r_xxxx_rzn1_if.h” and discussed in Section 3.

3. IOMUX API Functions

3.1 Summary

Function	Description
R_IOMUX_PinCtrl	Sets and gets multiplexing configuration for the pin
R_IOMUX_PortCtrl	Sets and gets multiplexing configuration for all pins of a port

3.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
```

3.3 R_IOMUX_PinCtrl

Format

ER_RET R_IOMUX_PinCtrl (uint8_t pin, iomux_cmd_t cmd, iomux_pin_properties_t *properties)

Parameters

pin	Which pin to configure
cmd	The command to execute (Set/Get pin properties)
properties	The properties of the pin * (function, drive strength, pull level)

Return Values

ER_OK	Success
ER_PARAM	Invalid command (Get or Set) or NULL pointer to properties

Properties

Prototyped in file “r_iomux_rzn1_if”

Description

Sets or Gets multiplexing configuration for specified pin.

Reentrant

No

Example:

```
iomux_pin_properties_t pin_mux = {.function = RZN1_FUNC_GPIO,
  .drive_strength = RZN1_MUX_DRIVE_4MA,
  .pull_level = RZN1_MUX_PULL_UP};
…
/* GPIO_PORT_3A_PIN_16 */
ret_val = R_IOMUX_PinCtrl (138, IOMUX_CONTROL_SET_PIN_PROPERTIES, &pin_mux);
```

Special Notes:

None

3.4 R_IOMUX_PortCtrl

Format

ER_RET R_IOMUX_ PortCtrl (gpio_port_t port, uint32_t port_mask, iomux_cmd_t cmd, iomux_pin_properties_t *properties)

Parameters

port	Port to set or get multiplexing configuration of pins

port_mask	Indicates which pins on the port are implemented
cmd	The command to execute (Set/Get pin properties)
properties	Pointer to array (for max 32 entries) for the properties of each pin (function, drive strength, pull level)

Return Values

ER_OK	Success
ER_PARAM	Invalid command (Get or Set) or NULL pointer to properties

Properties

Prototyped in file “r_iomux_rzn1_if.h”

Description

Sets or Gets multiplexing configuration for all pins of specified port.

Reentrant

No

Example:

```
ret_val = R_IOMUX_PortCtrl(test_4_port, test_4_port_mask, IOMUX_CONTROL_GET_PIN_PROPERTIES,
saved_pin_properties);
```

Special Notes:

None

4. SYSCTRL API Functions

4.1 Summary

Function	Description
R_SYSCTRL_EnableIOMUXLV2	Enables UDL to enable level 2 of pin multiplexing
R_SYSCTRL_EnableBGPIO	Configure clock enabling and interconnection for GPIO ports
R_SYSCTRL_DisableBGPIO	Disable clock and interconnection for GPIO ports
R_SYSCTRL_SetSPIClkDivider	Configure clock divider for SPI IP blocks
R_SYSCTRL_EnableMSEBI	Enables the MSEBI clock
R_SYSCTRL_DisableMSEBI	Disables the MSEBI clock
R_SYSCTRL_EnableSPI	Configure clock multiplexers for SPI channels
R_SYSCTRL_DisableSPI	Disable clock multiplexers for SPI channels
R_SYSCTRL_EnableUART	Enable clock multiplexers for UART channels
R_SYSCTRL_SetUARTClkDivider	Configure clock divider for UART IP blocks
R_SYSCTRL_DisableUART	Disable clock multiplexers for UART channels
R_SYSCTRL_EnableQSPI	Enables QSPI controller
R_SYSCTRL_DisableQSPI	Disables QSPI controller
R_SYSCTRL_ConfigDMAMux	Configure the DMA interconnect
R_SYSCTRL_EnableADC	Enables ADC
R_SYSCTRL_SetADCClkDivider	Configure the ADC
R_SYSCTRL_DisableADC	Disables ADC
R_SYSCTRL_EnableI2C	Enables I2C controller
R_SYSCTRL_DisableI2C	Disables I2C controller
R_SYSCTRL_SetI2CClkDivider	Setup I2C Clock divider
R_SYSCTRL_EnableGMAC	Enables GMAC controller
R_SYSCTRL_DisableGMAC	Disables GMAC controller
R_SYSCTRL_Switch	Power Management Control for R-IN Engine and A5PSW Accessory Registers
R_SYSCTRL_EthReg	Power Management Control for Ethernet Accessory Register
R_SYSCTRL_EnableDMA	Activate the DMA interconnect
R_SYSCTRL_DisableDMA	Disable the DMA interconnect
R_SYSCTRL_EnableSDIO	Activate the SDIO interconnect
R_SYSCTRL_DisableSDIO	Deactivate the SDIO interconnect
R_SYSCTRL_EnableUSBf	Enables the USB Function clock
R_SYSCTRL_DisableUSBf	Disables the USB Function clock
R_SYSCTRL_EnableUSBh	Enables the USB Host clock
R_SYSCTRL_DisableUSBh	Disables the USB Host clock
R_SYSCTRL_EnableTimer	Enable clock multiplexers for system timer.
R_SYSCTRL_WDTResetConfig	Enable or mask the Watchdog timer's reset ability
R_SYSCTRL_WDTReset	Unmask watchdog reset generation (To be called when a reset is pending
R_SYSCTRL_SystemReset	Resets the system
R_SYSCTRL_SetSDIOBaseClk	Set the SDIO base Clock Frequency
R_SYSCTRL_RTCReset	Reset the RTC IP
R_SYSCTRL_EnableRTC	Enables the RTC clock
R_SYSCTRL_DisableRTC	Disables the RTC clock

R_SYSCTRL_EnableCAN	Enables the CAN clock
R_SYSCTRL_DisableCAN	Disables the CAN clock
R_SYSCTRL_EnableLCDC	Enables LCDC controller
R_SYSCTRL_DisableLCDC	Disables LCDC controller
R_SYSCTRL_EnableSemaphore	Configure clock multiplexers for Semaphore Peripheral
R_SYSCTRL_DisableSemaphore	Turn off Semaphore, Disable clock
R_SYSCTRL_SemaphoreReset	Reset the Semaphore Peripheral
R_Get_CPUID	Gets the ID of the ARM CA7 core

4.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK       ((ER_RET)0)         /*!< Error code for Normal end (no error)    */
#define ER_NG       ((ER_RET)-1)        /*!< Error code for Abnormal end (error)     */
#define ER_SYS      ((ER_RET)(2 * ER_NG))  /*!< Error code for Undefined error       */
#define ER_PARAM    ((ER_RET)(3 * ER_NG))  /*!< Error code for Invalid parameter     */
#define ER_NOTYET   ((ER_RET)(4 * ER_NG))  /*!< Error code for Incomplete processing */
#define ER_NOMEM    ((ER_RET)(5 * ER_NG))  /*!< Error code for Out of memory         */
#define ER_BUSY     ((ER_RET)(6 * ER_NG))  /*!< Error code for Busy                  */
#define ER_INVAL    ((ER_RET)(7 * ER_NG))  /*!< Error code for Invalid state         */
#define ER_TIMEOUT  ((ER_RET)(8 * ER_NG))  /*!< Error code for Timeout occurs        */
```

4.3 R_SYSCTRL_EnableIOMUXLV2

Format

void R_SYSCTRL_EnableIOMUXLV2(void)

Parameters

NONE

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Enables the Level 2 multiplexing.

This is only called from the IOMUX_Init function during R_GPIO_Init at the start of Main().

Reentrant

NO

Example:

```
R_SYSCTRL_EnableIOMUXLV2();
```

Special Notes:

None

4.4 R_SYSCTRL_SystemReset

Format

void R_SYSCTRL_SystemReset (void);

Parameters

NONE

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Resets the system (Software reset).

Reentrant

No

Example:

```
R_SYSCTRL_SystemReset ();
```

Special Notes:

None

4.5 R_SYSCTRL_EnableADC

Format

ER_RET R_SYSCTRL_EnableADC (void);

Parameters

NONE

Return Values

ER_OK	No error
ER_SYS	System error when configuring ADC clock

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Enable clock and interconnect for ADC.

Called as part of adc_init().

Reentrant

YES

Example:

```
ret_val = R_SYSCTRL_EnableADC();
```

Special Notes:

NoneR_SYSCTRL_DisableADC

Format

ER_RET R_SYSCTRL_DisableADC (void);

Parameters

NONE

Return Values

ER_OK	No error
ER_SYS	System error when turning off ADC clock

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Disable clock and interconnect for ADC.

Called as part of adc_uninitialise ().

Reentrant

YES

Example:

```
ret_val = R_SYSCTRL_DisableADC();
```

Special Notes:

None

4.6 R_SYSCTRL_SetADCClkDivider

Format

ER_RET R_SYSCTRL_SetADCClkDivider (uint8_t clock_divider)

Parameters

NONE

Return Values

ER_OK	No error
ER_SYS	System error when configuring ADC clock divider

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Configure clock divider for ADC interconnect.

Called as part of adc_init().

Reentrant

YES

Example:

```
ret_val = R_SYSCTRL_SetADCClkDivider(ADC_CLOCK_DIVIDER);
```

Special Notes:

None

4.7 R_SYSCTRL_EnableBGPIO

Format

void R_SYSCTRL_EnableBGPIO(void)

Parameters

NONE

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Configure clock enabling and interconnection for GPIO ports.

Reentrant

No

Example:

```
R_SYSCTRL_EnableBGPIO();
```

Special Notes:

None

4.8 R_SYSCTRL_DisableBGPIO

Format

void R_SYSCTRL_DisableBGPIO(void)

Parameters

NONE

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Disable clock and interconnection for GPIO ports.

Reentrant

No

Example:

```
R_SYSCTRL_DisableBGPIO ();
```

Special Notes:

None

4.9 R_SYSCTRL_EnableCAN

Format

ER_RET R_SYSCTRL_EnableCAN(void);

Parameters

NONE

Return Values

ER_OK	No error
ER_SYS	System error when configuring RTC clock

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Enable clock and interconnect for CAN.

Called as part of can_init().

Reentrant

YES

Example:

```
ret_val = R_SYSCTRL_EnableCAN();
```

Special Notes:

None

4.10 R_SYSCTRL_DisableCAN

Format

void R_SYSCTRL_DisableCAN(void);

Parameters

NONE

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Disable clock and interconnect for CAN.

Called as part of can_uninitialise ().

Reentrant

YES

Example:

```
R_SYSCTRL_DisableCAN();
```

Special Notes:

None

4.11 R_SYSCTRL_ EnableDMA

Format

ER_RET R_SYSCTRL_EnableDMA (uint8_t DMAC);

Parameters

DMAC	DMA controller

Return Values

ER_OK	No error
ER_PARAM	Invalid parameters

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Activate the DMA interconnect.

Called as part of dma_init.

Reentrant

YES

Example:

```
R_SYSCTRL_EnableDMA(0);
```

Special Notes:

None

4.12 R_SYSCTRL_DisableDMA

Format

ER_RET R_SYSCTRL_DisableDMA (uint8_t DMAC);

Parameters

DMAC	DMA controller

Return Values

ER_OK	No error
ER_PARAM	Invalid parameters

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Disable the DMA interconnect.

Called as part of dma_uninit.

Reentrant

YES

Example:

```
R_SYSCTRL_DisableDMA (0);
```

Special Notes:

Non

4.13 R_SYSCTRL_ ConfigDMAMux

Format

ER_RET R_SYSCTRL_ConfigDMAMux (uint8_t function_request);

Parameters

DMAC	DMA controller

Return Values

ER_OK	No error

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Configure the DMA interconnect.

Called as part of rzn1_dma_mux.

Reentrant

YES

Example:

```
for (i = 0; i < 32; i++)
R_SYSCTRL_ConfigDMAMux(DMA_MUX_CONFIG[i]);
```

Special Notes:

None

4.14 R_SYSCTRL_EthReg();

Format

void R_SYSCTRL_EthReg();

Parameters

NONE

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Enables the 5 Port Switch.

Called as part of switch_init ().

Reentrant

YES

Example:

```
R_SYSCTRL_EthReg(); /*Enable clock for protected Ethernet register*/
R_SYSCTRL_Switch();
```

Special Notes:

None

4.15 R_SYSCTRL_ EnableTimer

Format

void R_SYSCTRL_ EnableTimer ();

Parameters

None

Return Values

void

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Configure clock multiplexers for Timer.

Called as part of timer1_subtimer_init.

Reentrant

YES

Example:

```
R_SYSCTRL_EnableTimer();
```

Special Notes:

None

4.16 R_SYSCTRL_EnableI2C

Format

ER_RET **R_SYSCTRL_EnableI2C** (uint8_t chan_num)

Parameters

chan_num	I2C channel to enable

Return Values

ER_OK	Success
ER_SYS	Invalid channel number or IP timeout

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Enables an I2C channel. Called as part of i2c driver initialisation

Reentrant

No

Example:

```
ret_val = R_SYSCTRL_EnableI2C(chan_num);
```

Special Notes:

None

4.17 R_SYSCTRL_SetI2CClkDivider

Format

ER_RET R_SYSCTRL_SetI2CClkDivider (uint8_t i2c_clock_divider)

Parameters

i2c_clock_divider	I2C clock divider value to set value can be in range 12-64 RZ/N1l main PLL freq / PWRCTRL_PG0_I2CDIV gives I2C clock

Return Values

ER_OK	Success
ER_SYS	PWRCTRL_PG0_I2CDIV BUSY bit timeout

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Sets the I2C clock divider

Reentrant

No

Example:

```
ret_val = R_SYSCTRL_SetI2CClkDivider(I2C_CLOCK_DIVIDER);
```

Special Notes:

None

4.18 R_SYSCTRL_ EnableI2C

Format

```
ER_RET R_SYSCTRL_EnableI2C (uint8_t chan_num)
```

Parameters

chan_num	I2C channel to enable

Return Values

ER_OK	Success
ER_SYS	Invalid channel number or IP timeout

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Enables an I2C channel.

Reentrant

No

Example:

```
ret_val = R_SYSCTRL_EnableI2C(channel);
if (ER_OK != ret_val)
{
    break;
}
```

Special Notes:

None

4.19 R_SYSCTRL_ DisableI2C

Format

```
ER_RET R_SYSCTRL_ DisableI2C (uint8_t chan_num)
```

Parameters

chan_num	I2C channel to disable

Return Values

ER_OK	Success
ER_SYS	Invalid channel number or IP timeout

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Enables an I2C channel.

Reentrant

No

Example:

```
ret_val = R_SYSCTRL_DisableI2C(chan_num);
```

Special Notes:

None

4.20 R_SYSCTRL_EnableLCDC

Format

ER_RET R_SYSCTRL_EnableLCDC (uint8_t clock_divider)

Parameters

NONE

Return Values

ER_OK	No error
ER_SYS	System error (timeout) when configuring the LCDC

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Configure clock multiplexers for LCDC

Called as part of lcdc_driver_init ().

Reentrant

YES

Example:

```
/* 33.3 MHz LCDC Control CLK
 * 1GHz / PWRCTRL_PG0_LCDCDIV gives LCDC clock
 * Divider will be set to 30 for a 33.33 MHz LCDC Clock frequency
 * Divider value can be 12-64                                    */
#define LCDC_CLOCK_DIVIDER      (30)
    ret_val = R_SYSCTRL_EnableLCDC(LCDC_CLOCK_DIVIDER);
```

Special Notes:

None

4.21 R_SYSCTRL_DisableLCDC

Format

void R_SYSCTRL_DisableLCDC (void);

Parameters

NONE

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Disable clocks and interconnect for LCDC.

Called as part of lcdc_driver_uninit ().

Reentrant

YES

Example:

```
ret_val = R_SYSCTRL_DisableLCDC();
```

Special Notes:

None

4.22 R_SYSCTRL_EnableMSEBI

Format

void R_SYSCTRL_DisableMSEBI (bool master);

Parameters

bool master True for Master, False for Slave

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Disable clock and interconnect from MSEBI master or slave.

Reentrant

No

Example:

Called within MSEBI_Open

```
/* Enable bus interconnect and PCLK to the MSEBI */
R_SYSCTRL_EnableMSEBI(master);
```

Special Notes:

None

4.23 R_SYSCTRL_DisableMSEBI

Format

void R_SYSCTRL_DisableMSEBI (bool master);

Parameters

bool master True for Master, False for Slave

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Enables the MSEBI peripheral.

Reentrant

No

Example:

Called within MSEBI_Close

```
/* Disable bus interconnect and PCLK from the MSEBI */
R_SYSCTRL_DisableMSEBI(master);
```

Special Notes:

None

4.24 R_SYSCTRL_ EnableQSPI

Format

void R_SYSCTRL_EnableQSPI (uint8_t chan_num);

Parameters

chan_num	QSPI channel

Return Values

void

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Enables QSPI controller.

Called as part of qspi_channel_open.

Reentrant

YES

Example:

```
R_SYSCTRL_EnableQSPI(chan_num);
```

Special Notes:

None

4.25 R_SYSCTRL_ DisableQSPI

Format

void R_SYSCTRL_DisableQSPI (uint8_t chan_num);

Parameters

chan_num	QSPI channel

Return Values

void

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Disables QSPI controller.

Called as part of qspi_uninit (void).

Reentrant

YES

Example:

```
R_SYSCTRL_DisableQSPI(chan_num);
```

Special Notes:

None

4.26 R_SYSCTRL_EnableRTC

Format

ER_RET R_SYSCTRL_EnableRTC (void);

Parameters

NONE

Return Values

ER_OK	No error

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Enable clock and interconnect for RTC.

Called as part of rtc_init().

Reentrant

YES

Example:

```
ret_val = R_SYSCTRL_EnableRTC();
```

Special Notes:

None

4.27 R_SYSCTRL_DisableRTC

Format

void R_SYSCTRL_DisableRTC (void);

Parameters

NONE

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Disable clock and interconnect for RTC.

Called as part of rtc_uninitialise ().

Reentrant

YES

Example:

```
R_SYSCTRL_DisableRTC();
```

Special Notes:

None

4.28 R_SYSCTRL_RTCReset

Format

void R_SYSCTRL_RTCReset (void)

Parameters

NONE

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Reset RTC interconnect.

Called as part of rtc_reset ().

Reentrant

YES

Example:

```
R_SYSCTRL_RTCReset();
```

Special Notes:

None

4.29 R_SYSCTRL_ EnableSDIO

Format

void R_SYSCTRL_ EnableSDIO (uint8_t sdio_instance);

Parameters

sdio_instance	Which sdio host controller to enable

Return Values

ER_OK	Success
ER_PARAM	Invalid parameter

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Activate the SDIO interconnect.

Called as part of sdio_open.

Reentrant

YES

Example:

```
R_SYSCTRL_EnableSDIO(SDIO_number);
```

Special Notes:

None

4.30 R_SYSCTRL_ DisableSDIO

Format

void R_SYSCTRL_ DisableSDIO (uint8_t sdio_instance);

Parameters

sdio_instance	Which sdio host controller to enable

Return Values

ER_OK	Success
ER_PARAM	Invalid parameter

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Deactivate the SDIO interconnect.

Called as part of sdio_close.

Reentrant

YES

Example:

```
R_SYSCTRL_ DisableSDIO (SDIO_number);
```

Special Notes:

None

4.31 R_SYSCTRL_ SetSDIOBaseClk

Format

void R_SYSCTRL_ SetSDIOBaseClk (uint8_t sdio_instance, uint8_t base_clock);

Parameters

sdio_instance	Which sdio host controller to enable
base_clock	The base clock frequency for the SDIO controller (MHz)

Return Values

ER_OK	Success
ER_PARAM	Invalid parameter

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Set the SDIO base Clock Frequency.

Called as part of sdio_open.

Reentrant

YES

Example:

```
R_SYSCTRL_SetSDIOBaseClk(SDIO_number, SDIO_BASE_CLOCK_FREQ);
```

Special Notes:

None

4.32 R_SYSCTRL_EnableSemaphore

Format

ER_RET R_SYSCTRL_EnableSemaphore (void)

Parameters

NONE

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Configure clock multiplexers for Semaphore Peripheral

Called as part of semaphore_init().

Reentrant

YES

Example:

```
ret_val = R_SYSCTRL_EnableSemaphore();
if (ER_OK == ret_val)
{
    m_semaphore_ready = true;
}
```

Special Notes:

None

4.33 R_SYSCTRL_DisableSemaphore

Format

ER_RET R_SYSCTRL_DisableSemaphore (void)

Parameters

NONE

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Turn off Semaphore, Disable clock

Called as part of semaphore_uninitialise().

Reentrant

YES

Example:

```
R_SYSCTRL_DisableSemaphore();
```

Special Notes:

None

4.34 R_SYSCTRL_SemaphoreReset

Format

void R_SYSCTRL_SemaphoreReset (void)

Parameters

NONE

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Reset the Semaphore Peripheral.

Called as part of semaphore_reset ().

Reentrant

YES

Example:

```
R_SYSCTRL_SemaphoreReset();
```

Special Notes:

None

4.35 R_SYSCTRL_Switch

Format

void R_SYSCTRL_Switch ()

Parameters

NONE

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Enables the 5 Port Switch.

Called as part of switch_init ().

Reentrant

YES

Example:

```
R_SYSCTRL_EthReg(); /*Enable clock for protected Ethernet register*/
R_SYSCTRL_Switch();
```

Special Notes:

None

4.36 R_SYSCTRL_ EnableSPI

Format

void R_SYSCTRL_ EnableSPI (uint8_t chan_num);

Parameters

chan_num	SPI channel number to enable

Return Values

void

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Enable clock and interconnect for SPI channel.

Called as part of spi_channelopen(uint8_t chan_num).

Reentrant

YES

Example:

```
R_SYSCTRL_ EnableSPI (chan_num);
```

Special Notes:

None

4.37 R_SYSCTRL_ DisableSPI

Format

void R_SYSCTRL_ DisableSPI (uint8_t chan_num);

Parameters

chan_num	SPI channel number to enable

Return Values

void

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Enable clock and interconnect for SPI channel.

Called as part of spi_channelclose(uint8_t chan_num).

Reentrant

YES

Example:

```
R_SYSCTRL_ DisableSPI (chan_num);
```

Special Notes:

None

4.38 R_SYSCTRL_SetSPIClkDivider

Format

ER_RET R_SYSCTRL_SetUARTClkDivider (uint8_t clock_divider)

Parameters

clock_divider	Clock divider for the SPI (8 - 128)

Return Values

ER_OK	Success
ER_SYS	Undefined error

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Configure clock divider for SPI IP blocks.

Called as part of `spi_driverinit.`

Reentrant

YES

Example:

```
ret_val = R_SYSCTRL_SetSPIClkDivider(SPI_CLOCK_DIVIDER);
```

Special Notes

None

4.39 R_SYSCTRL_EnableUART

Format

void R_SYSCTRL_EnableUART (uint8_t chan_num);

Parameters

chan_num	UART channel number to open

Return Values

None

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Enables a UART channel. Enables the clock supply to the UART channel, and de-asserts the reset.

Called as part of uart_channel_open ()

Reentrant

YES

Example:

```
R_SYSCTRL_EnableUART(chan_num);
```

Special Notes:

None

4.40 R_SYSCTRL_DisableUART

Format

void R_SYSCTRL_DisableUART (uint8_t chan_num)

Parameters

chan_num	UART channel number to disable

Return Values

None

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Disables a UART channel. Disables the clock supply to the channel.

Called as part of uart_channel_close ()

Reentrant

No

Example:

```
/* Disable power control & Clock for UART channel */
R_SYSCTRL_DisableUART(chan_num);
```

Special Notes:

None

4.41 R_SYSCTRL_SetUARTClkDivider

Format

ER_RET R_SYSCTRL_SetUARTClkDivider (uint8_t chan_num, uint8_t clock_divider)

Parameters

chan_num	UART channel number to configure
clock_divider	Clock divider for the UART

Return Values

ER_OK	Success
ER_SYS	PWRCTRL_PG0_UARTDIV BUSY bit timeout

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Sets UART clock divider.

Called as part of `R_UART_Open`

Reentrant

YES

Example:

```
ret_val = R_SYSCTRL_SetUARTClkDivider(chan_num, UART_CLOCK_DIVIDER);
```

Special Notes:

None

4.42 R_SYSCTRL_EnableUSBf

Format

ER_RET R_SYSCTRL_EnableUSBf (void)

Parameters

NONE

Return Values

ER_RET	Values defined in errcodes.h

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Setup USB 2.0 Function controller.

Reentrant

No

Example:

Called within R_USBf_Init

```
/* Initialise USB function controller  */
ret_val = R_SYSCTRL_EnableUSBf ();
```

Special Notes:

None

4.43 R_SYSCTRL_DisableUSBf

Format

void R_SYSCTRL_DisableUSBf (void);

Parameters

NONE

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Disables USB Function controller.

Reentrant

No

Example:

```
R_SYSCTRL_DisableUSBf ( );
```

Special Notes:

None

4.45 R_SYSCTRL_EnableUSBh

Format

ER_RET R_SYSCTRL_EnableUSBh(void)

Parameters

NONE

Return Values

ER_OK	No error
ER_SYS	System error when configuring USB Host clock

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Setup USB 2.0 Host.

Reentrant

No

Example:

Called within R_USBh_Init

```
/* Initialise USB Host  */
ret_val = R_SYSCTRL_EnableUSBh ();
```

Special Notes:

None

4.46 R_SYSCTRL_DisableUSBh

Format

void R_SYSCTRL_DisableUSBfh(void);

Parameters

NONE

Return Values

NONE

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Disables USB Host.

Reentrant

No

Example:

```
R_SYSCTRL_DisableUSBh ( );
```

Special Notes:

None

4.47 R_SYSCTRL_WDTReset

Format

void R_SYSCTRL_WDTReset (void);

Parameters

None

Return Values

void

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Unmask watchdog reset generation (To be called when a reset is pending).

Called as part of handle_wdt_isr.

Reentrant

YES

Example:

```
R_SYSCTRL_WDTReset();
```

Special Notes:

None

4.48 R_SYSCTRL_WDTResetConfig

Format

void R_SYSCTRL_WDTResetConfig (bool reset_en);

Parameters

reset_en	Enable or Disable

Return Values

void

Properties

Prototyped in file “r_sysctrl_rzn1_if.h”

Description

Mask or unmask the Watchdog timer's reset ability.

Called as part of wdt_setoutput.

Reentrant

YES

Example:

```
R_SYSCTRL_WDTResetConfig(true);
```

Special Notes:

None

5. BSP API Functions

5.1 Summary

The BSP functions relate to registering and configuring the interrupts. They are contained in the mcu_interrupts.c file, and included in the drivers using the mcu_interrupts_if.h file

Function	Description
R_BSP_InterruptRegisterCallback	Register a callback function
R_BSP_InterruptGetRegisteredCallback	Get a registered callback function pointer
R_BSP_InterruptControl	Execute an interrupt control command

5.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

5.3 R_BSP_InterruptRegisterCallback

Format

MCU_INT_ERR_E R_BSP_InterruptRegisterCallback (IRQn_Type vector, bsp_int_cb_t callback);

Parameters

vector	Which interrupt to register a callback for
callback	Pointer to function to call when interrupt occurs

Return Values

MCU_INT_SUCCESS	No error
MCU_INT_ERR_INVALID_ARG	Invalid address or invalid interrupt vector number

Properties

Prototyped in file “mcu_interrupts_if.h”

Description

Registers a callback function for supported interrupts. If FIT_NO_FUNC, NULL, or any other invalid function address is passed for the callback argument then any previously registered callbacks are unregistered.

Reentrant

No

Example:

```
 if (MCU_INT_SUCCESS == R_BSP_InterruptRegisterCallback((IRQn_Type) irq_num, (bsp_int_cb_t)
isr_handler[DMAC_number])
 {…
```

Special Notes:

None

5.4 R_BSP_InterruptGetRegisteredCallback

Format

MCU_INT_ERR_E R_BSP_InterruptGetRegisteredCallback (IRQn_Type vector, bsp_int_cb_t * callback);

Parameters

vector	Which interrupt to read the callback for
callback	Pointer of where to store callback address

Return Values

MCU_INT_SUCCESS	No error
MCU_INT_ERR_INVALID_ARG	Invalid address or invalid interrupt vector number
MCU_INT_ERR_NO_REGISTERED_CALLBACK	No Registered Callback found

Properties

Prototyped in file “mcu_interrupts_if.h”

Description

Returns the callback function address for an interrupt.

Reentrant

No

Example:

```
bsp_int_cb_t * callback ;
if (MCU_INT_SUCCESS == R_BSP_InterruptGetRegisteredCallback ((IRQn_Type) vector, callback);
 {…
```

Special Notes:

None

5.5 R_BSP_InterruptControl

Format

MCU_INT_ERR_E R_BSP_InterruptControl (IRQn_Type vector, MCU_INT_CMD_E cmd, void *pdata);

Parameters

vector	Which vector is being used
cmd	Which command to execute
pdata	Pointer to data to use with command

Return Values

MCU_INT_SUCCESS	No error
MCU_INT_ERR_INVALID_ARG	Invalid address or invalid interrupt vector number
MCU_INT_ERR_NO_REGISTERED_CALLBACK	No Registered Callback found

Properties

Prototyped in file “mcu_interrupts_if.h”

Description

Executes the specified command “cmd”.

Reentrant

No

Example:

```
ret_val = R_BSP_InterruptControl(irq_num, MCU_INT_CMD_SET_INTERRUPT_PRIORITY, &int_priority);
```

Special Notes:

None

6. ADC API Functions

6.1 Summary

Function	Description
R_ADC_GetVersion	Returns driver version.
R_ADC_Init	Initialise ADC driver and channels interface state
R_ADC_Uninitialise	Un-Initialise ADC IP
R_ADC_Open	Opens an ADC channel
R_ADC_Close	Closes an ADC channel
R_ADC_Read	Reads data from an ADC channel after sampling. If BLOCKING (synchronous) mode then returns when all the requested data has been transferred or a timeout occurs. If NON-BLOCKING/DMA (asynchronous) mode then initiates the conversion and returns. In this case a callback parameter is required to be passed through a previous call to R_ADC_Control().
R_ADC_ReadGroup	Reads data from selected ADC channels after sampling. If BLOCKING (synchronous) mode then returns when all the requested data If NON-BLOCKING/DMA (asynchronous) mode then initiates the conversion and returns. In this case a callback parameter is required to be passed through a previous call to R_ADC_Control().
R_ADC_Control	Configure channel parameters, set transfer parameters or get channel or transfer info, depending on control_request
R_ADC_GetConversionStatus	Returns EBUSY if an ADC acquisition is in progress otherwise returns EOK NOTE: If a channel is in continuous mode then EBUSY is always returned.

6.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

6.3 R_ADC_GetVersion

Format

void R_ADC_GetVersion(void *buf);

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

void

Properties

Prototyped in file “r_adc_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
R_ADC_GetVersion((void*)&adc_driver_version);
```

Special Notes:

None

6.4 R_ADC_Init

Format

ER_RET R_ADC_Init(void);

Parameters

None

Return Values

ER_OK	No error
ER_INVAL	Driver already initialised.

Properties

Prototyped in file “r_adc_rzn1_if.h”

Description

Initialise ADC driver and channels interface state.

Reentrant

No

Example:

```
R_ADC_Init();
```

Special Notes:

None

6.5 R_ADC_Uninitialise

Format

ER_RET R_ADC_Uninitialise(void);

Parameters

None

Return Values

ER_OK	No error
ER_INVAL	Driver already not Initialised or Closed.

Properties

Prototyped in file “r_adc_rzn1_if.h”

Description

Un-Initialise ADC IP.

Reentrant

No

Example:

```
R_ADC_Uninitialise(void);
```

Special Notes:

None

6.6 R_ADC_Open

Format

ER_RET R_ADC_Open(uint8_t chan_num, ADC_VC_CONFIG_T * ptr_adc_ctrl);

Parameters

chan_num	ADC channel number to open
ptr_adc_ctrl	ADC Configuration parameters

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_INVAL	Driver is not initialised

Properties

Prototyped in file “r_adc_rzn1_if.h”

Description

Opens an ADC channel.

Reentrant

No

Example:

```
ret_status = R_ADC_Open(chan, &vc_setup);
```

Special Notes:

None

6.7 R_ADC_Close

Format

ER_RET R_ADC_Close(uint8_t chan_num);

Parameters

chan_num	ADC channel number to close

Return Values

void

Properties

Prototyped in file “r_adc_rzn1_if.h”

Description

Closes an ADC channel.

Reentrant

No

Example:

```
ret_status = R_ADC_Close(chan_num);
```

Special Notes:

None

6.8 R_ADC_Read

Format

ER_RET R_ADC_Read (uint8_t chan_num, ADC_ACQ_CMD_T * ptr_acq_cmd)

Parameters

chan_num	ADC channel number
ptr_acq_cmd	A pointer to a ADC_ACQ_CMD_T struct containing an acquisition mode and somewhere to store the data

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_INVAL	Driver is not initialised
ER_BUSY	Channel acquisition is already in progress via another request
ER_NG	Conversion has not completed correctly
ER_NOTYET	Channel not ready to read data

Properties

Prototyped in file “r_adc_rzn1_if.h”

Description

Reads data from an ADC channel after sampling.

If BLOCKING (synchronous) mode then returns when all the requested data has been transferred or a timeout occurs.

If NON-BLOCKING/DMA (asynchronous) mode then initiates the conversion and returns.

In this case a callback parameter is required to be passed through a previous call to R_ADC_Control().

Reentrant

No

Example:

```
ret_status  = R_ADC_Read(ADC_VC5, &acq_cmd);
```

Special Notes:

None

6.9 R_ADC_ReadGroup

Format

ER_RET R_ADC_ReadGroup(ADC_VC_T chan_group, ADC_ACQ_CMD_T * ptr_acq_cmd)

Parameters

chan_group	ADC channel group selection via bit fields
ptr_acq_cmd	A pointer to a ADC_ACQ_CMD_T struct containing an acquisition mode and somewhere to store the data

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_INVAL	Driver is not initialised
ER_BUSY	Channel acquisition is already in progress via another request
ER_NG	Conversion has not completed correctly
ER_NOTYET	Channel not ready to read data

Properties

Prototyped in file “r_adc_rzn1_if.h”

Description

Reads data from selected ADC channels after sampling.

If BLOCKING (synchronous) mode then returns when all the requested data has been transferred or a timeout occurs.

If NON-BLOCKING/DMA (asynchronous) mode then initiates the conversion and returns.

In this case a callback parameter is required to be passed through a previous call to R_ADC_Control().

Reentrant

No

Example:

```
ret_status = R_ADC_ReadGroup(chan_group, &acq_cmd);
```

Special Notes:

None

6.10 R_ADC_Control

Format

ER_RET R_ADC_Control(uint8_t chan_num, ADC_CONTROL_REQUEST_E control_request, void * ptr_adc_ctrl)

Parameters

chan_num	ADC channel number to control
control_request	request to configure channel or get channel info.
ptr_adc_ctrl	configuration data (to set or get)

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_INVAL	Driver is not initialised
ER_BUSY	Channel acquisition is already in progress via another request
ER_NG	Conversion has not completed correctly
ER_NOTYET	Channel not ready to read data

Properties

Prototyped in file “r_adc_rzn1_if.h”

Description

Configure channel parameters, set transfer parameters or get channel or transfer info, depending on the configuration data

Reentrant

No

Example:

```
vc_setup.dma_request   = ADC_VC_DMA_DISABLE;
vc_setup.mode          = ADC_PHYS_CHANNEL_CONVERT;
vc_setup.adc1_chan_sel = ADC1_IN0;
vc_setup.run_mode      = ADC_SINGLE_CONVERSION;
ret_status = R_ADC_Control(ADC_VC5, ADC_CONTROL_SET_CHAN_CONFIG, &vc_setup);
```

Special Notes:

None

6.11 R_ADC_GetConversionStatus

Format

ER_RET R_ADC_GetConversionStatus(void)

Parameters

None

Return Values

ER_OK	No error
ER_BUSY	Channel acquisition is already in progress via another request

Properties

Prototyped in file “r_adc_rzn1_if.h”

Description

Returns EBUSY if an ADC acquisition is in progress otherwise returns EOK

NOTE: If a channel is in continuous mode then EBUSY is always returned..

Reentrant

No

Example:

```
status = R_ADC_GetConversionStatus();
```

Special Notes:

None

7. CAN API Functions

7.1 Summary

Function	Description
R_CAN_GetVersion	Returns driver version.
R_CAN_Init	Reset all channels states to closed. Initializes CAN driver
R_CAN_Uninitialise	Uninitializes CAN driver
R_CAN_Open	Opens a CAN channel & initializes channel state
R_CAN_Control	Controls & gets information for the CAN: • Gets channel state • Gets/sets Channel configuration • Reset the channel • Sets event callback function • Set Acceptance Filter • Configures Synchronisation pulse
R_CAN_Write	Writes data to CAN Bus channel
R_CAN_Close	Closes an CAN channel

7.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

7.3 R_CAN_GetVersion

Format

void R_CAN_GetVersion(void *buf)

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

void

Properties

Prototyped in file “r_can_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
R_CAN_GetVersion ((void*) &can_driver_version);
```

Special Notes:

None

7.4 R_CAN_Init

Format

ER_RET R_CAN_Init (void);

Parameters

None

Return Values

ER_OK	No error
ER_INVAL	Already Initialised
ER_SYS	System error when enabling CAN peripheral clock

Properties

Prototyped in file “r_can_rzn1_if.h”

Description

Initialize CAN driver and CAN channel states

Reentrant

No

Example:

```
R_CAN_Init ();
```

Special Notes:

None

7.5 R_CAN_Uninitialise

Format

ER_RET R_CAN_ Uninitialise(void);

Parameters

None

Return Values

ER_OK	No error
ER_INVAL	Not Initialised or a channel is still open
ER_SYS	System error when enabling CAN peripheral clock

Properties

Prototyped in file “r_can_rzn1_if.h”

Description

Uninitialize CAN driver and CAN channel states

Reentrant

No

Example:

```
R_CAN_Uninitialise();
```

Special Notes:

None

7.6 R_CAN_Open

Format

ER_RET R_CAN_Open (uint8_t chan_num);

Parameters

chan_num	The CAN Channel to open

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels or is already open
ER_INVAL	Channel is not already Initialised
ER_SYS	System error when configuring CAN interrupts

Properties

Prototyped in file “r_can_rzn1_if.h”

Description

Open CAN channel

Reentrant

No

Example:

```
R_CAN_Open (CAN_CHAN_1);
```

Special Notes:

None

7.7 R_CAN_Close

Format

ER_RET R_CAN_Close (uint8_t chan_num);

Parameters

chan_num	The CAN Channel to close

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels or channel is already closed
ER_INVAL	Channel is not Open

Properties

Prototyped in file “r_can_rzn1_if.h”

Description

Closes a CAN channel

Reentrant

No

Example:

```
R_CAN_Close (CAN_CHAN_1);
```

Special Notes:

None

7.8 R_CAN_Control

Format

ER_RET R_CAN_Control (uint8_t chan_num, CAN_CONTROL_REQUEST_E control_request,
void * ptr_can_ctrl);

Parameters

chan_num	The CAN Channel to close
control_request	A define for the type of control request being performed
ptr_can_ctrl	A pointer to a structure for the control data

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels or an invalid request
ER_INVAL	Channel is not Open
ER_SYS	System error when disabling or enabling the CAN channel

Properties

Prototyped in file “r_can_rzn1_if.h”

Description

Configures the CAN driver

Reentrant

No

Example:

Configure the driver;

```
CAN_SYNC_PULSE_T sync_setup;

sync_setup.sync_msg.extended_flag = true;
 sync_setup.sync_msg.id = 0x12345678;
 sync_setup.sync_msg.data_len = 3;
 sync_setup.sync_msg.data[0] = 0xAA;
 sync_setup.sync_msg.data[1] = 0xBB;
 sync_setup.sync_msg.data[2] = 0xCC;


 ret_status = R_CAN_Control(CAN_CHAN_2, CAN_CONTROL_START_CHAN_SYNC_PULSE,&sync_setup);
```

Special Notes:

None

7.9 R_CAN_Write

Format

ER_RET R_CAN_Write(uint8_t chan_num, CAN_MSG_T * ptr_msg);

Parameters

chan_num	The CAN Channel to write to
ptr_msg	Pointer to description of message to transmit

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels, or invalid parameter, NULL pointer
ER_INVAL	Channel is not Open
ER_SYS	System error when disabling or enabling the CAN channel
ER_BUSY	BUS transmission on the CAN channel currently in progress

Properties

Prototyped in file “r_can_rzn1_if.h”

Description

Writes data to a CAN channel.

If a RTR type of message (request for data) then a callback parameter is required to have been passed in a previous call to R_CAN_Control() to handle reception of the response.

Reentrant

No

Example:

```
ret_status = R_CAN_Write(CAN_CHAN_2, &can_msg);
```

Special Notes:

None

8. DMA API Functions

8.1 Summary

Function	Description
R_DMA_GetVersion	Returns driver version.
R_DMA_Init	Reset all channels states to closed. Initializes DMA driver
R_DMA_Uninitialise	Uninitializes DMA driver
R_DMA_Open	Opens an DMA channel & initializes channel state
R_DMA_Control	Controls & gets information for the DMA: • Gets channel state • Gets/sets channel configuration • Gets/sets transaction configuration • Pause/Resume transaction • Reset the channel • Sets interrupt callback function
R_DMA_Write	Writes data from to DMA channel
R_DMA_Close	closes an DMA channel

8.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

8.3 R_DMA_GetVersion

Format

void R_DMA_GetVersion(void *buf)

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

void

Properties

Prototyped in file “r_dma_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
R_DMA_GetVersion( (void*) &dma_driver_version);
```

Special Notes:

None

8.4 R_DMA_Init

Format

void R_DMA_Init (void);

Parameters

None

Return Values

void

Properties

Prototyped in file “r_dma_rzn1_if.h”

Description

Initialize DMA driver and DMA channel states

Reentrant

No

Example:

```
R_DMA_Init ();
```

Special Notes:

None

8.5 R_DMA_Uninitialise

Format

void R_DMA_ Uninitialise(void);

Parameters

None

Return Values

None

Properties

Prototyped in file “r_dma_rzn1_if.h”

Description

Uninitialise DMA driver and DMA channel states

Reentrant

No

Example:

```
R_DMA_Uninitialise();
```

Special Notes:

None

8.6 R_DMA_Open

Format

ER_RET R_DMA_Open (uint8_t DMAC_number, uint8_t channel);

Parameters

DMAC_number	The number of DMAC instance
channel	Channel number

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_INVAL	Channel is not Open

Properties

Prototyped in file “r_dma_rzn1_if.h”

Description

Open a DMA channel.

Reentrant

No

Example:

```
ret_val = R_DMA_Open(DMAC1, DMA_CHAN_3);
```

Special Notes:

None

8.7 R_DMA_Close

Format

ER_RET R_DMA_Close (uint8_t DMAC_number, uint8_t channel);

Parameters

DMAC_number	The number of DMAC instance
channel	Channel number

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_INVAL	Channel is not Open

Properties

Prototyped in file “r_dma_rzn1_if.h”

Description

Closes an DMA channel

Reentrant

No

Example:

```
R_DMA_Close (DMAC1, DMA_CHAN_1);
```

Special Notes:

None

8.8 R_DMA_Control

Format

ER_RET R_DMA_Control (uint8_t DMAC_number, uint8_t channel, DMA_CONTROL_REQUEST control_request, void *control);

Parameters

DMAC_number	DMAC instance to be controlled
channel	DMA channel
control_request	A define for the type of control request being performed
control	A pointer to a structure for the control data

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_INVAL	Channel is not Open, driver not initialized
ER_SYS	System error when disabling or enabling the DMA channel

Properties

Prototyped in file “r_dma_rzn1_if.h”

Description

Configures the DMA driver

Reentrant

No

Example:

Set callback function:

```
ret_val = R_DMA_Control(DMAC1, DMA_CHAN_7, DMA_CONTROL_SET_CALL_BACK, (void *) &dmac1_callback);
```

Special Notes:

None

8.9 R_DMA_Write

Format

ER_RET R_DMA_Write (uint8_t DMAC_number, uint8_t channel, DMA_TRANS_REQ_TYPE req_type);

Parameters

DMAC_number	DMAC instance to be controlled
channel	DMA channel
req_type	Transaction requirement type

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels, or invalid parameter, NULL pointer
ER_INVAL	Channel is not Open, driver not initialized

Properties

Prototyped in file “r_dma_rzn1_if.h”

Description

Enables transaction from DMA channel.

Reentrant

No

Example:

```
ret_val = R_DMA_Write(DMAC1, DMA_CHAN_7, DMA_TRANSFER);
```

Special Notes:

None

9. GMAC API Functions

9.1 Summary

Function	Description
R_GMAC_GetVersion	Returns the version of this driver
R_GMAC_Init	Initialises GMAC channels
R_GMAC_Uninit	Disables GMAC
R_GMAC_Open	Opens GMAC channels and sets it to listen for upcoming data
R_GMAC_Close	Closes GMAC channel
R_GMAC_Write	Sends out data
R_GMAC_Read	Reads data
R_GMAC_Control	Configures GMAC channel
R_GMAC_LinkStatus	Determines the link status for a port.

9.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

9.3 R_GMAC_GetVersion

Format

void R_GMAC_GetVersion(void *buf);

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

None

Properties

Prototyped in file “r_gmac_rzn1_if.h”

Description

Returns the version of this driver.

Reentrant

No

Example:

R_GMAC_GetVersion((void*) &gmac_driver_version);

Special Notes:

None

9.4 R_GMAC_Init

Format

ER_RET R_GMAC_Init (void);

Parameters

None

Return Values

ER_OK	No error

Properties

Prototyped in file “r_gmac_rzn1_if.h”

Description

Enables transaction from DMA channel.

Reentrant

No

Example:

```
ret_val = R_GMAC_Init();
```

Special Notes:

None

9.5 R_GMAC_Uninit

Format

void R_GMAC_Uninit (void);

Parameters

None

Return Values

None

Properties

Prototyped in file “r_gmac_rzn1_if.h”

Description

Enables transaction from DMA channel.

Reentrant

No

Example:

```
ret_val = R_GMAC_Init();
```

Special Notes:

None

9.6 R_GMAC_Open

Format

```
ER_RET R_GMAC_Open (const gmac_transfer_data *data)
```

Parameters

data	Structure that stores all information about the channel

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_SYS	Driver is not initialised

Properties

Prototyped in file “r_gmac_rzn1_if.h”

Description

Opens GMAC channels and sets it to listen for upcoming data

Reentrant

No

Example:

```
ret_val = R_GMAC_Open(&sending_chan);
```

Special Notes:

None

9.7 R_GMAC_Close

Format

ER_RET R_GMAC_Close (const gmac_transfer_data *data)

Parameters

data	Structure that stores all information about the channel

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_SYS	Driver failed to close

Properties

Prototyped in file “r_gmac_rzn1_if.h”

Description

Closes GMAC channel.

Reentrant

No

Example:

```
ret_val = R_GMAC_Close(&receiving_chan);
```

Special Notes:

None

9.8 R_GMAC_Write

Format

```
ER_RET R_GMAC_Write (const gmac_transfer_data *data, const gmac_eth_frame
*eth_frame)
```

Parameters

data	Structure that stores all information about the channel
eth_frame	structure with ethernet header info

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_INVAL	The GMAC channel or port is CLOSED

Properties

Prototyped in file “r_gmac_rzn1_if.h”

Description

Sends out data

Reentrant

No

Example:

```
ret_val = R_GMAC_Write(&sending_chan, &frame);
```

Special Notes:

None

9.9 R_GMAC_Read

Format

```
ER_RET R_GMAC_Read (const gmac_transfer_data *data, gmac_receive_buf * pDataStruct)
```

Parameters

data	pointer to a structure containing the MAC channel
pDataStruct	pointer a buffer to read data into

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_INVAL	The GMAC channel or port is CLOSED

Properties

Prototyped in file “r_gmac_rzn1_if.h”

Description

Reads data

Reentrant

No

Example:

```
R_GMAC_Read(&receiving_chan, &receive_buf);
```

Special Notes:

None

9.10 R_GMAC_Control

Format

```
ER_RET R_GMAC_Control(const gmac_transfer_data *data, gmac_control_state_t
control_request, uint32_t control_options)
```

Parameters

data	Structure that stores all information about the channel
control_request	control request
control_options	address of a structure dedicated for a certain control request

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_SYS	Driver failed to close

Properties

Prototyped in file “r_gmac_rzn1_if.h”

Description

Configures GMAC channel.

Reentrant

No

Example:

```
ret_val = R_GMAC_Control(&sending_chan, SET_SPEED, 0);
ret_val = R_GMAC_Control(&receiving_chan, SET_FILTER, (uint32_t) &filtering_options);
```

Special Notes:

None

9.11 R_GMAC_LinkStatus

Format

```
uint8_t R_GMAC_LinkStatus(uint8_t port);
```

Parameters

port	PHY port number

Return Values

0 if the link is down and 1 if the link is up.

Properties

Prototyped in file “r_gmac_rzn1_if.h”

Description

Configures GMAC channel.

Reentrant

No

Example:

```
while((!R_GMAC_LinkStatus(PORT1) && !R_GMAC_LinkStatus(PORT2)))  /*Wait till the link is up*/
```

Special Notes:

None

10. GPIO API Functions

10.1 Summary

The GPIO driver provides an API to use the functionality of the GPIO pins.

The GPIO Driver checks that the PIN has been configured as a GPIO pin by checking the Pin Function bit.

Function	Description
R_GPIO_GetVersion	Returns the driver version number
R_GPIO_Init	Initializes the GPIO driver and IP interconnect
R_GPIO_Uninitialise	Uninitializes the GPIO driver and IP interconnect
R_GPIO_PortOpen	Activates pins of a port as GPIO pins
R_GPIO_PinOpen	Activates a single pin as a GPIO pin
R_GPIO_PortClose	Deactivates GPIO pins of a port
R_GPIO_PinClose	Deactivates a single GPIO pin
R_GPIO_PortWrite	Writes the logic levels to pins of a port
R_GPIO_PortRead	Reads the logic levels of a port
R_GPIO_PortDirectionGet	Returns the contents of the Data Direction Register for the port
R_GPIO_PortDirectionSet	Sets the Data Direction of pins of a port
R_GPIO_PinWrite	Writes the logic level to a pin
R_GPIO_PinRead	Reads the logic level of a pin
R_GPIO_PinDirectionGet	Returns the content of the Data Direction Register for the pin
R_GPIO_PinDirectionSet	Sets the Data Direction of the pin
R_GPIO_RegInterruptCallBack	Registers an interrupt call back function
R_GPIO_PinControl	Controls the properties of a GPIO pin

10.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

10.3 R_GPIO_GetVersion

Format

void R_GPIO_GetVersion (void *buf);

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

void

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

Returns the driver version number. The major version number is located in the two most significant bytes, whereas the minor version number is located in the 2 least significant bytes.

Reentrant

Yes

Example:

```
uint8_t            gpio_driver_version[2];
R_GPIO_GetVersion((void *)gpio_driver_version);
```

Special Notes:

None

10.4 R_GPIO_Init

Format

ER_RET R_GPIO_Init (void);

Parameters

None

Return Values

ER_OK	No error
ER_SYS	Failure to register interrupts through mcu_interrupts

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

initializes the GPIO driver. Sets up the driver internal data, register base addresses, and registers the interrupts.

Reentrant

No

Example:

```
ret_val = R_GPIO_Init();
```

Special Notes:

None

10.5 R_GPIO_Uninitialise

Format

ER_RET R_GPIO_Uninit (void);

Parameters

None

Return Values

ER_OK	No error
ER_SYS	Failure to disable interrupts.

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

Uninitialises the GPIO drivers.

Reentrant

No

Example:

```
ret_val = R_GPIO_Uninit();
```

Special Notes:

None

10.6 R_GPIO_PortOpen

Format

ER_RET R_GPIO_PortOpen (gpio_port_t port, uint32_t mask);

Parameters

port	The GPIO port to open
mask	Selects the pins on the port to open

Return Values

ER_OK	No error
ER_PARAM	Parameter error (Non-existent pins)
ER_INVAL	One or more of the selected pins have already been opened. None of the pins will be opened.
ER_BUSY	One or more of the selected pins have not been multiplexed for GPIO operation. None of the pins will be opened.

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

Enables GPIO operation of the selected pins (i.e. writing to pins and enabling interrupts).

The “port” specified when opening should be

```
GPIO_PORT_1A
GPIO_PORT_1B
GPIO_PORT_2A
GPIO_PORT_2B
GPIO_PORT_3A
```

Reentrant

No

Example:

```
ret_val = R_GPIO_PortOpen (GPIO_PORT_1A, 0xF0F0F0F0);
```

Special Notes:

None

10.7 R_GPIO_PinOpen

Format

ER_RET R_GPIO_PinOpen (gpio_port_pin_t pin);

Parameters

pin	The GPIO pin to open

Return Values

ER_OK	No error
ER_PARAM	Parameter error (Non-existent pin)
ER_INVAL	The pin has already been opened. The pin will not be opened.
ER_BUSY	The pin has not been multiplexed for GPIO operation. The pin will not be opened.

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

Enables GPIO operation of this pin (i.e. writing to the pin and enabling interrupts).

The “pin” specified should a define from gpio_port_pin_t in the target header file in the GPIO driver under src/targets/rzn1X/r_gpio_rzn1X.h

Reentrant

No

Example:

```
ret_val = R_GPIO_PinOpen (GPIO_PORT_1A_PIN_01);
```

Special Notes:

None

10.8 R_GPIO_PortClose

Format

ER_RET R_GPIO_PortClose (gpio_port_t port, uint32_t mask);

Parameters

port	The GPIO port to close
mask	Selects the pins on the port to close

Return Values

ER_OK	No error
ER_PARAM	Parameter error (Non-existent pins)

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

Deactivates GPIO pins of a port preventing further GPIO operation of any of the selected pins. Also disables interrupts for any interrupt-enabled pins.

Reentrant

No

Example:

```
ret_val = R_GPIO_PortClose(GPIO_PORT_1A, 0xF0F0F0F0);
```

Special Notes:

None

10.9 R_GPIO_PinClose

Format

ER_RET R_GPIO_PinClose(gpio_port_pin_t pin);

Parameters

pin	The GPIO pin to close

Return Values

ER_OK	No error
ER_PARAM	Parameter error (Non-existent pin)

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

Deactivates a single GPIO pin preventing further GPIO operation of the pin. Also disables interrupts for the pin.

Reentrant

No

Example:

```
ret_val  = R_GPIO_PinClose(GPIO_PORT_1A_PIN_00);
```

Special Notes:

None

10.10 R_GPIO_PortWrite

Format

ER_RET GPIO_PortWrite (gpio_port_t port, uint32_t value, uint32_t mask);

Parameters

port	Which port to write to.
value	The value to write to the port. Each bit corresponds to a pin on the port (e.g. bit 0 of value will be written to pin 0 on supplied port)
mask	Mask to allow certain pins of the port to be changed. A pin will only change value when the corresponding mask bit is 1.

Return Values

ER_OK	No error
ER_PARAM	Parameter error (Non-existent pins)
ER_INVAL	One or more of the selected pins is not open or has been set to input.

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

Writes the levels of all selected pins on a port.

Reentrant

No

Example:

```
ret_val = R_GPIO_PortWrite(GPIO_PORT_1A, 0x55555555, 0xF0F0F0F0);
```

Special Notes:

None

10.11 R_GPIO_PortRead

Format

ER_RET R_GPIO_PortRead (gpio_port_t port, uint32_t *value);

Parameters

port	Which port to write to.
value	The value of the port

Return Values

ER_OK	No error
ER_PARAM	Parameter error (Non-existent pins)

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

Reads the logic levels of a port.

Reentrant

Yes

Example:

```
uint32_t input_value;
ret_val = R_GPIO_PortRead(GPIO_PORT_3A, &input_value);
```

Special Notes:

None

10.12 R_GPIO_PortDirectionGet

Format

ER_RET R_GPIO_PortDirectionGet (gpio_port_t port, uint32_t *direction);

Parameters

port	Which port to retrieve the direction from.
direction	The value of the Data Direction Register

Return Values

ER_OK	No error
ER_PARAM	Parameter error (Non-existent pins)

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

Returns the contents of the Data Direction Register for the port (1 = Output, 0 = Input).

Reentrant

Yes

Example:

```
uint32_t Port_Dir_Get ;
ret_val = R_GPIO_PortDirectionGet(GPIO_PORT_1A, &Port_Dir_Get);
```

Special Notes:

None

10.13 R_GPIO_PortDirectionSet

Format

ER_RET R_GPIO_PortDirectionSet (gpio_port_t port, gpio_dir_t dir, uint32_t mask);

Parameters

port	Which port to set.
dir	The data direction of the pins
mask	Mask to allow certain pins of the port to be changed. A pin's direction will only change value when the corresponding mask bit is 1

Return Values

ER_OK	No error
ER_PARAM	Parameter error (Non-existent pins)
ER_INVAL	One or more of the selected pins is not open. None of the pins will have their direction set.

Properties

Prototyped in file "r_gpio_rzn1_if.h"

Description

Sets the Data Direction of pins of a port**.**

Reentrant

No

Example:

```
ret_val = R_GPIO_PortDirectionSet(GPIO_PORT_1A, GPIO_DIRECTION_OUTPUT, 0x55555555);
```

Special Notes:

None

10.14 R_GPIO_PinWrite

Format

ER_RET R_GPIO_PinWrite (gpio_port_pin_t pin, gpio_level_t level);

Parameters

pin	Which pin to write to.
level	The logic level to write to the pin

Return Values

ER_OK	No error
ER_PARAM	Parameter error (Non-existent pin)
ER_INVAL	The pin is not open or has been set to input.

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

Writes the logic level to a pin.

Reentrant

No

Example:

```
ret_val = R_GPIO_PinWrite(GPIO_PORT_3A_PIN_14, GPIO_LEVEL_HIGH);
```

Special Notes:

None

10.15 R_GPIO_PinRead

Format

ER_RET R_GPIO_PinRead (gpio_port_pin_t pin, gpio_level_t *level);

Parameters

pin	Which pin to read from
level	The logic level of the pin

Return Values

ER_OK	No error
ER_PARAM	Parameter error (Non-existent pin)
ER_INVAL	The pin is not open or has been set to output.

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

Reads the logic level of a pin.

Reentrant

No

Example:

```
gpio_level_t input_level;
ret_val = R_GPIO_PinRead(GPIO_PORT_3A_PIN_16, &input_level);
```

Special Notes:

None

10.16 R_GPIO_PinDirectionGet

Format

ER_RET R_GPIO_PinDirectionGet (gpio_port_pin_t pin, gpio_dir_t *dir);

Parameters

pin	Which pin to retrieve the direction from.
dir	The Data Direction of the pin

Return Values

ER_OK	No error
ER_PARAM	Parameter error (Non-existent pin)
ER_INVAL	The pin is not open.

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

Returns the content of the Data Direction Register for the pin.

Reentrant

No

Example:

```
gpio_dir_t Pin_Dir_Get;
ret_val = R_GPIO_PinDirectionGet(GPIO_PORT_1A_PIN_01, &Pin_Dir_Get);
```

Special Notes:

None

10.17 R_GPIO_PinDirectionSet

Format

ER_RET R_GPIO_PinDirectionSet (gpio_port_pin_t pin, gpio_dir_t dir);

Parameters

pin	Which pin to set.
dir	The data direction of the pin

Return Values

ER_OK	No error
ER_PARAM	Parameter error (Non-existent pin)
ER_INVAL	The pin is not open.

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

Sets the Data Direction of the pin.

Reentrant

No

Example:

```
ret_val = R_GPIO_PinDirectionSet(GPIO_PORT_1A_PIN_01, GPIO_DIRECTION_OUTPUT);
```

Special Notes:

None

10.18 R_GPIO_RegInterruptCallBack

Format

ER_RET R_GPIO_RegisterInterruptCallBack(gpio_callback gpio_call_back);

Parameters

gpio_call_back	The call back function to register.

Return Values

ER_OK	No error
ER_PARAM	Call back function has been passed through as NULL

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

Registers an interrupt call back function. This single, common, call back function will be called when any of the interrupt pins generate an interrupt.

Reentrant

No

Example:

```
volatile gpio_port_pin_t pin ;
…
static void gpio_interrupt_callback (gpio_port_pin_t GPIO_PIN);
…
main (void)
{
    ER_RET ret_val;
    …
    ret_val = R_GPIO_RegisterInterruptCallBack(&gpio_interrupt_callback);
    …
}
static void gpio_interrupt_callback (gpio_port_pin_t GPIO_PIN)
{
    pin = GPIO_PIN;
    …
}
```

Special Notes:

None

10.19 R_GPIO_PinControl

Format

ER_RET R_GPIO_PinControl(gpio_port_pin_t pin, gpio_cmd_t cmd, void *gpio_ctrl);

Parameters

pin	The GPIO pin to configure
cmd	Which pin control function to execute
gpio_ctrl	Command data for the pin control function

Return Values

ER_OK	No error
ER_PARAM	Parameter error (Non-existent pin)
ER_INVAL	If cmd is **not** GPIO_CONTROL_GET_PIN_STATE: - The pin is not open If cmd is GPIO_CONTROL_SET_INT: - The pin belongs to a B port - Attempting to enable an interrupt with no free IRQ line (Only 8 interrupts can be enabled at a time) - Attempting to disable interrupts on an already interrupt-disabled pin

Properties

Prototyped in file “r_gpio_rzn1_if.h”

Description

GPIO_CONTROL_GET_PIN_STATE:

Retrieves the state of the GPIO pin (GPIO_PIN_STATE_OPEN, GPIO_PIN_STATE_CLOSED). Parameter *gpio_ctrl must be a pointer to a variable of type gpio_pin_states_t.

GPIO_CONTROL_SET_INT:

Enables or disables interrupts for a GPIO pin. Parameter *gpio_ctrl must be a pointer to a variable of type gpio_int_en_t.

GPIO_CONTROL_SET_INT_MASK:

Enables or disables an interrupt mask for a GPIO pin. Parameter *gpio_ctrl must be a pointer to a variable of type gpio_int_mask_t.

GPIO_CONTROL_SET_INT_TRIG_TYPE:

Sets the trigger type for an interrupt on a GPIO pin. Parameter *gpio_ctrl must be a pointer to a variable of type gpio_int_trigger_type_t.

GPIO_CONTROL_SET_INT_TRIG_POL:

Sets the trigger polarity for interrupts on a GPIO pin. Parameter *gpio_ctrl must be a pointer to a variable of type gpio_int_polarity_t.

Reentrant

No

Example:

```
gpio_pin_states_t pin_state;
gpio_int_trigger_type_t trigger_type = GPIO_INTTERRUPT_TRIGGER_EDGE;
gpio_int_polarity_t trigger_polarity = GPIO_INTTERRUPT_POLARITY_LOW;
gpio_int_mask_t mask = GPIO_INTTERRUPT_MASK_DISABLED;
gpio_int_en_t gpio_int = GPIO_INTTERRUPT_ENABLED;
…
ret_val = R_GPIO_PinControl(GPIO_PORT_3A_PIN_14, GPIO_CONTROL_GET_PIN_STATE, &pin_state);
…
/* Set to level or edge */
    ret_val = R_GPIO_PinControl(GPIO_PORT_3A_PIN_16, GPIO_CONTROL_SET_INT_TRIG_TYPE, &trigger_type);
…
/* Interrupt set to trigger when pin reads Low */
    ret_val= R_GPIO_PinControl(GPIO_PORT_3A_PIN_16, GPIO_CONTROL_SET_INT_TRIG_POL,
&trigger_polarity);
…
/* Unmask interrupt */
    ret_val = R_GPIO_PinControl(GPIO_PORT_3A_PIN_16, GPIO_CONTROL_SET_INT_MASK, &mask);
…
/* Enable interrupt */
    ret_val = R_GPIO_PinControl(GPIO_PORT_3A_PIN_16, GPIO_CONTROL_SET_INT, &gpio_int);
```

Special Notes:

None

11. I2C API Functions

11.1 Summary

Function	Description
R_I2C_GetVersion	Returns driver version.
R_I2C_Init	Reset all channels states to closed. Initializes I2C driver
R_I2C_Uninitialise	Uninitializes I2C driver
R_I2C_Open	Opens an I2C channel & initializes channel state
R_I2C_Control	Controls & gets information for the I2C: • Gets channel state • Gets/sets Channel configuration • Reset the channel • Sets interrupt callback function
R_I2C_Read	Reads data from a I2C channel
R_I2C_Write	Writes data from to I2C channel
R_I2C_Close	closes an I2C channel

11.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

11.3 R_I2C_GetVersion

Format

void R_I2C_GetVersion(void *buf)

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

void

Properties

Prototyped in file “r_i2c_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
R_I2C_GetVersion ((void*) &i2c_driver_version);
```

Special Notes:

None

11.4 R_I2C_Init

Format

ER_RET R_I2C_Init (void);

Parameters

None

Return Values

void

Properties

Prototyped in file “r_i2c_rzn1_if.h”

Description

Initialize I2C driver and I2C channel states

Reentrant

No

Example:

```
R_I2C_Init ();
```

Special Notes:

None

11.5 R_I2C_Uninitialise

Format

ER_RET R_I2C_ Uninitialise(void);

Parameters

None

Return Values

void

Properties

Prototyped in file “r_i2c_rzn1_if.h”

Description

Uninitialize I2C driver and I2C channel states

Reentrant

No

Example:

```
R_I2C_ Uninitialise();
```

Special Notes:

None

11.6 R_I2C_Open

Format

ER_RET R_I2C_Open (uint8_t i2c_chan_num);

Parameters

i2c_chan_num	The I2C Channel to open

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_INVAL	Channel is not Open

Properties

Prototyped in file “r_i2c_rzn1_if.h”

Description

Initialize I2C driver and I2C channel states

Reentrant

No

Example:

```
R_I2C_Open (I2C_CHAN_1);
```

Special Notes:

None

11.7 R_I2C_Close

Format

ER_RET R_I2C_Close (uint8_t i2c_chan_num);

Parameters

i2c_chan_num	The I2C Channel to close

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_INVAL	Channel is not Open

Properties

Prototyped in file “r_i2c_rzn1_if.h”

Description

Closes an I2C channel

Reentrant

No

Example:

```
R_I2C_Close (I2C_CHAN_1);
```

Special Notes:

None

11.8 R_I2C_Control

Format

ER_RET R_I2C_Control (uint8_t i2c_chan_num, I2C_CONTROL_REQUEST_E control_request, uint8_t *buf);

Parameters

i2c_chan_num	The I2C Channel to close
control_request	A define for the type of control request being performed
buf	A pointer to a structure for the control data

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_INVAL	Channel is not Open, driver not initialized
ER_SYS	System error when disabling or enabling the I2C channel

Properties

Prototyped in file “r_i2c_rzn1_if.h”

Description

Configures the I2C driver

Reentrant

No

Example:

Configure the driver;

```
        i2c_config.speed = I2C_SPEED_FAST_MODE;
        i2c_config.master = 1;
        i2c_config.slave = 0;
        i2c_config.restart_enable = 1;
        i2c_config.addr_mode = 1;
ret_status = R_I2C_Control(I2C_CHAN_2, I2C_CONTROL_SET_CHAN_CONFIG, (uint8_t *) &i2c_config);
```

Special Notes:

None

11.9 R_I2C_Write

Format

ER_RET R_I2C_Write (uint8_t i2c_chan_num, uint32_t i2c_slave_addr, uint8_t *addr, uint32_t addr_len, uint8_t *buf, size_t data_len, I2C_TRANSFER_MODE_E mode);

Parameters

i2c_chan_num	The I2C Channel to Write to
i2c_slave_addr	A define for the type of control request being performed
addr	Address internal to the device
addr_len	Number bytes in addr
buf	Data buffer
data_len	Number of bytes to be written
mode	BLOCKING or NON-BLOCKING write

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels, or invalid parameter, NULL pointer
ER_INVAL	Channel is not Open, driver not initialized
ER_SYS	System error when disabling or enabling the I2C channel

Properties

Prototyped in file “r_i2c_rzn1_if.h”

Description

Writes data to an i2c channel.

If BLOCKING (synchronous) mode then returns when all the requested data has been read or a timeout occurs.

If NON-BLOCKING (asynchronous) mode then initiates the read process and returns.

In this case a callback parameter is required to have been passed in a previous call to R_I2C_Control()

Reentrant

No

Example:

```
ret_val = R_I2C_Write(I2C_CHAN_2, I2C_PORT_EXPANDER_SLAVE_ADDR, &addr, 1, &data, 1,
I2C_TRANSFER_MODE_BLOCKING);
```

Special Notes:

None

11.10 R_I2C_Read

Format

ER_RET R_I2C_Read (uint8_t i2c_chan_num, uint32_t i2c_slave_addr, uint8_t *addr, uint32_t addr_len, uint8_t *buf, size_t data_len, I2C_TRANSFER_MODE_E mode);

Parameters

i2c_chan_num	The I2C Channel to Read from
i2c_slave_addr	A define for the type of control request being performed
addr	Address internal to the device
addr_len	Number bytes in addr
buf	Data buffer
data_len	number of bytes to be read
mode	BLOCKING or NON-BLOCKING read

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_INVAL	Channel is not Open, driver not initialized
ER_SYS	System error when disabling or enabling the I2C channel

Properties

Prototyped in file “r_i2c_rzn1_if.h”

Description

Reads data from a i2c channel.

If BLOCKING (synchronous) mode then returns when all the requested data has been written or a timeout occurs.

If NON-BLOCKING (asynchronous) mode then initiates the write process and returns.

In this case a callback parameter is required to have been passed in a previous call to R_I2C_Control().

Reentrant

No

Example:

```
ret_val = R_I2C_Read(I2C_CHAN_2, I2C_PORT_EXPANDER_SLAVE_ADDR, &addr, 1, &data, 1,
I2C_TRANSFER_MODE_BLOCKING);
```

Special Notes:

None

12. LCDC API Functions

12.1 Summary

Function	Description
R_LCDC_GetVersion	Returns driver version.
R_LCDC_Init	Initialises LCD controller IP
R_LCDC_Uninitialise	Close down LCD controller
R_TSD_Init	Initialise the Touch Screen driver
R_TSD_Uninitialise	Close down the Touch Screen driver
R_LCDC_Open	Opens and configures the LCD port
R_LCDC_Close	Closes the LCD port
R_LCDC_Control	Handles requests to set or get port status and configuration
R_LCDC_DisplayUpdate	Update a section of the display with the given image
R_LCDC_FillRectangle	Set a section of the display to a single colour
R_LCDC_Blank	Clear (blank) the whole display with given colour
R_LCDC_Blink	Blink a section of the display

12.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

12.3 R_LCDC_GetVersion

Format

void R_LCDC_GetVersion (void *buf)

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

void

Properties

Prototyped in file “r_lcdc_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
R_LCDC_GetVersion ((void*) &lcdc_driver_version);
```

Special Notes:

None

12.4 R_LCDC_Init

Format

ER_RET R_LCDC_Init (void);

Parameters

None

Return Values

Error status

Properties

Prototyped in file “r_lcdc_rzn1_if.h”

Description

Initialises the LCD controller

Reentrant

No

Example:

```
ret_val = R_LCDC_Init();
```

Special Notes:

None

12.5 R_LCDC_Uninitialise

Format

ER_RET R_LCDC_Uninitialise(void);

Parameters

None

Return Values

Error status

Properties

Prototyped in file “r_lcdc_rzn1_if.h”

Description

Close down the LCD controller

Reentrant

No

Example:

Ret_val = `R_LCDC_Uninitialise();`

Special Notes:

None

12.6 R_TSD_Init

Format

ER_RET R_TSD_Init (gpio_callback ts_int_callback);

Parameters

gpio_callback callback function for
ts interrupts

Return Values

Error status

Properties

Prototyped in file “r_lcdc_rzn1_if.h”

Description

Initialises the Touch Screen driver

Reentrant

No

Example:

ret_val = R_TSD_Init(&ts_event_callback);

Special Notes:

IOMUX , GPIO and LCDC drivers need to be initialized before calling this function.

After calling R_TSD_Init , the I2C driver needs to be initialized and opened to read the touch screen inputs

12.7 R_TSD_Uninitialise

Format

ER_RET R_TSD_Uninitialise(void);

Parameters

None

Return Values

Error status

Properties

Prototyped in file “r_lcdc_rzn1_if.h”

Description

Close down the Touch Screen Driver

Reentrant

No

Example:

```
ret_val = R_TSD_Uninitialise();
```

Special Notes:

None

12.8 R_LCDC_Open

Format

ER_RET R_LCDC_Open (lcdc_port_config_t *port_config);

Parameters

port_config	pointer to display configuration

Return Values

ER_OK	No error
ER_PARAM	Channel number is higher than the max number of channels
ER_INVAL	Channel is not Open

Properties

Prototyped in file “r_lcdc_rzn1_if.h”

Description

Opens and configures the LCD port

Reentrant

No

Example:

```
lcdc_config.bpp = NHD_LCD_SCREEN_BPP;
lcdc_config.pixel_format = LCD_RGB565;
lcdc_config.x_res = NHD_LCD_SCREEN_WIDTH;
lcdc_config.y_res = NHD_LCD_SCREEN_HEIGHT;
lcdc_config.palette = (void *)0;
ret_val = R_LCDC_Open(&lcdc_config);
```

Special Notes:

None

12.9 R_LCDC_Close

Format

ER_RET R_LCDC_Close (void);

Parameters

none

Return Values

ER_OK	No error
ER_INVAL	Port is not Open

Properties

Prototyped in file “r_lcdc_rzn1_if.h”

Description

Closes the LCD Port

Reentrant

No

Example:

```
ret_val = R_LCDC_Close();
```

Special Notes:

None

12.10 R_LCDC_Control

Format

ER_RET R_LCDC_Control (lcdc_control_request_e control_request, uint8_t *buf);

Parameters

control_request	request to configure port or get port info
buf	A pointer to a structure for the control data

Return Values

ER_OK	No error
ER_PARAM	Buf is NULL or Control Request is invalid
ER_INVAL	Driver is not initialised or Port is not open
ER_SYS	System error

Properties

Prototyped in file “r_lcdc_rzn1_if.h”

Description

Requests to set or get port status or configuration

Reentrant

No

Example:

```
ret_val = R_LCDC_Control(LCDC_CONTROL_GET_PORT_CONFIG, config_buf);
```

Special Notes:

None

12.11 R_LCDC_DisplayUpdate

Format

ER_RET R_LCDC_DisplayUpdate (lcd_rectangle_t *display_update_area, uint32_t *buf, uint32_t buf_len);

Parameters

display_update_area	pointer to rectangle of display to update
buf	pointer to display data to be written
buf_len	data length

Return Values

ER_OK	No error
ER_PARAM	display_update_area or buf are NULL, co-ordinates of display_update_area are invalid
ER_INVAL	driver not initialized or LCD port not open
ER_SYS	System error

Properties

Prototyped in file “r_lcdc_rzn1_if.h”

Description

Updates an area of the display.

Reentrant

No

Example:

```
ret_val = R_LCDC_DisplayUpdate (lcd_rectangle_t *display_update_area, uint32_t *buf, uint32_t
buf_len);
```

Special Notes:

- for bpp <= 8, rectangle co-ordinates must be on an 8-pixel boundary
- for bpp >= 16 have 1 pixel data per 32-bit entry of buf

  for bpp = 8 have 1 pixel index per 32-bit entry of buf

  for bpp = 4 have 2 pixel indices per 32-bit entry of buf

  for bpp = 2 have 4 pixel indices per 32-bit entry of buf

  for bpp = 1 have 8 pixel indices per 32-bit entry of buf

12.12 R_LCDC_FillRectangle

Format

ER_RET R_LCDC_FillRectangle (lcd_rectangle_t *display_update_area, uint32_t colour);

Parameters

display_update_area	pointer to rectangle of display to update
colour	Fill colour

Return Values

ER_OK	No error
ER_PARAM	display_update_area is NULL,
ER_INVAL	driver not initialized or LCD port not open
ER_SYS	System error

Properties

Prototyped in file “r_lcdc_rzn1_if.h”

Description

Updates a rectangular section of the display with the specified colour

Reentrant

No

Example:

```
ret_val = R_LCDC_FillRectangle(&horiz_line_top_coords, LCDC_PINK_RGB888);
```

Special Notes:

colour should be in correct format depending on configured bpp and colour mode e.g.

for bpp < 16 , colour is palette index

for bpp = 16, RGB 5:6:5 , colour is 0b0000 0000 0000 0000 RRRR RGGG GGGB BBBB

for bpp = 16, BGR 5:6:5 , colour is 0b0000 0000 0000 0000 BBBB BGGG GGGR RRRR

for bpp = 16, RGB 5:5:5 , colour is 0b0000 0000 0000 0000 0RRR RRGG GGGB BBBB

for bpp = 16, BGR 5:5:5 , colour is 0b0000 0000 0000 0000 0BBB BBGG GGGR RRRR

for bpp = 24, RGB 8:8:8 , colour is 0b0000 0000 RRRR RRRR GGGG GGGG BBBB BBBB

for bpp = 24, BGR 8:8:8 , colour is 0b0000 0000 BBBB BBBB GGGG GGGG RRRR RRRR

12.13 R_LCDC_Blank

Format

ER_RET R_LCDC_Blank (uint32_t colour);

Parameters

colour	Set blank display to this colour

Return Values

ER_OK	No error
ER_INVAL	driver not initialized or LCD port not open
ER_SYS	System error

Properties

Prototyped in file “r_lcdc_rzn1_if.h”

Description

Clear (blank) the display with the given colour

Reentrant

No

Example:

```
ret_val = R_LCDC_Blank (LCDC_BLACK_RGB565);
```

Special Notes:

colour should be in correct format depending on configured bpp and colour mode e.g.

for bpp < 16 , colour is palette index

for bpp = 16, RGB 5:6:5 , colour is 0b0000 0000 0000 0000 RRRR RGGG GGGB BBBB

for bpp = 16, BGR 5:6:5 , colour is 0b0000 0000 0000 0000 BBBB BGGG GGGR RRRR

for bpp = 16, RGB 5:5:5 , colour is 0b0000 0000 0000 0000 0RRR RRGG GGGB BBBB

for bpp = 16, BGR 5:5:5 , colour is 0b0000 0000 0000 0000 0BBB BBGG GGGR RRRR

for bpp = 24, RGB 8:8:8 , colour is 0b0000 0000 RRRR RRRR GGGG GGGG BBBB BBBB

for bpp = 24, BGR 8:8:8 , colour is 0b0000 0000 BBBB BBBB GGGG GGGG RRRR RRRR

12.14 R_LCDC_Blink

Format

ER_RET R_LCDC_Blink (lcd_rectangle_t *display_blink_area, lcdc_blink_mode_e blink_mode);

Parameters

display_blink_area	pointer to rectangle of display to blink
blink_mode	LCDC_BLINK_MODE_OFF
	LCDC_BLINK_MODE_SLOW
	LCDC_BLINK_MODE_MEDIUM
	LCDC_BLINK_MODE_FAST

Return Values

ER_OK	No error
ER_PARAM	display_update_area is NULL, blink_mode is invalid
ER_INVAL	driver not initialized or LCD port not open
ER_SYS	System error

Properties

Prototyped in file “r_lcdc_rzn1_if.h”

Description

Blink a section of the display

Reentrant

No

Example:

```
blink_area.lcd_rectangle_bottom = 31;
blink_area.lcd_rectangle_top = 16;
blink_area.lcd_rectangle_right = 31;
blink_area.lcd_rectangle_left = 16;
ret_val = R_LCDC_Blink(&blink_area, LCDC_BLINK_MODE_ON);
```

Special Notes:

Co-ordinates of the blink area should be on a 4-byte boundary

13. MSEBI API Functions

13.1 Summary

Function	Description
R_DMA_GetVersion	Returns driver version.
R_MSEBI_Init	Initialise MSEBI driver and chip select states
R_MSEBI_Open	Opens an MSEBI mode and/or chip select
R_MSEBI_Control	Configure MSEBI parameters, set chip select parameters or get info, depending on control_request
R_MSEBI_Close	Closes an MSEBI mode and/or chip select(s)

13.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK      ((ER_RET)0)                /*!< Error code for Normal end (no error) */
#define ER_NG      ((ER_RET)-1)                /*!< Error code for Abnormal end (error)  */
#define ER_SYS     ((ER_RET)(2 * ER_NG))    /*!< Error code for Undefined error       */
#define ER_PARAM   ((ER_RET)(3 * ER_NG))    /*!< Error code for Invalid parameter     */
#define ER_NOTYET  ((ER_RET)(4 * ER_NG))    /*!< Error code for Incomplete processing */
#define ER_NOMEM   ((ER_RET)(5 * ER_NG))    /*!< Error code for Out of memory         */
#define ER_BUSY    ((ER_RET)(6 * ER_NG))    /*!< Error code for Busy                  */
#define ER_INVAL   ((ER_RET)(7 * ER_NG))    /*!< Error code for Invalid state         */
#define ER_TIMEOUT ((ER_RET)(8 * ER_NG))    /*!< Error code for Timeout occurs        */
```

13.3 R_MSEBI_GetVersion

Format

void R_MSEBI_GetVersion (void *buf);

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

void

Properties

Prototyped in file “r_msebi_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
R_MSEBI_GetVersion((void *)msebi_driver_version);
```

Special Notes:

None

13.4 R_MSEBI_Init

Format

void MSEBI_Init(void)

Parameters

None

Return Values

None

Properties

Prototyped in file “r_msebi_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
/* Initialise MSEBI module, before GPIO */
R_MSEBI_Init();
```

Special Notes:

None

13.5 R_MSEBI_Open

Format

ER_RET R_MSEBI_Open(msebi_mode_t mode, uint8_t chip_sel)

Parameters

msebi_mode_t mode	MSEBIM for Master or MSEBIS for Slave Mode
uint8_t chip_sel	chip_sel - The desired chip select

Return Values

ER_RET Error Value. ER_OK on success.

Properties

Prototyped in file “r_msebi_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
ret_val = R_MSEBI_Open(MSEBIM, MSEBI_NO_CS);
```

Special Notes:

None

13.6 R_MSEBI_Close

Format

ER_RET R_MSEBI_Close(msebi_mode_t mode, uint8_t chip_sel)

Parameters

msebi_mode_t mode	MSEBIM for Master or MSEBIS for Slave Mode
uint8_t chip_sel	chip_sel - The desired chip select

Return Values

ER_RET	Error Value. ER_OK on success.

Properties

Prototyped in file “r_msebi_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
R_MSEBI_Close(MSEBIM, MSEBI_NO_CS);
```

Special Notes:

None

13.7 R_MSEBI_Control

Format

ER_RET R_MSEBI_Control(msebi_mode_t mode, uint8_t chip_sel, msebi_cmd_t control_request, void *msebi_ctrl)

Parameters

msebi_mode_t mode	MSEBIM for Master or MSEBIS for Slave Mode
uint8_t chip_sel	The desired chip select
msebi_cmd_t control_request	request to configure MSEBI or get MSEBI info.
void *msebi_ctrl	configuration data (to set or get).

Return Values

ER_RET	Error Value. ER_OK on success.

Properties

Prototyped in file “r_msebi_rzn1_if.h”

Description

Configure MSEBI parameters, set chip select parameters or get info, depending on control_request.

NOTE: API function checks that msebi_ctrl is non-zero. The upper layer should check that the buffer lies within a valid address range.

Reentrant

No

Example:

```
R_MSEBI_Control(MSEBIM ,MSEBI_NO_CS ,MSEBI_CONTROL_MASTER_CLK_ENABLE, &clock_enable);
```

Special Notes:

None

13.8 R_MSEBIS_SetCallback

Format

ER_RET R_MSEBIS_SetCallback (msebi_mode_t mode, msebi_callback_mode_t callback_mode, uint8_t chip_sel, msebi_callback callback)

Parameters

mode	MSEBI master/slave mode
callback_mode	mode which determines for which interrupt callback function stands
chip_sel	The chip select to configure 0-3
callback	callback function

Return Values

ER_RET	Error Value. ER_OK on success.

Properties

Prototyped in file “r_msebi_rzn1_if.h”

Description

Register callback function for transmit, receive or both interrupt types.

Reentrant

No

Example:

```
R_MSEBIS_SetCallback(MSEBISLAVE, RECEIVE, MSEBI_CS0_N, (msebi_callback) &rx_interrupt_callback);
```

Special Notes:

None

14. QSPI API Functions

14.1 Summary

Function	Description
R_QSPI_GetVersion	Returns driver version.
R_QSPI_Init	Resets all channels states to closed and initializes QSPI IP.
R_QSPI_Uninit	Resets all channels states to closed and uninitializes QSPI IP.
R_QSPI_Open	Opens a QSPI channel.
R_QSPI_Control	Controls/gets information for the QSPI driver/channel: • Sets/gets channel transfer configuration • Gets Channel State • Resets the channel
R_QSPI_Read	Reads data from Flash memory via QSPI channel.
R_QSPI_Write	Writes data to Flash memory via QSPI channel.
R_QSPI_Erase	Erases serial flash.
R_QSPI_Close	Closes a QSPI channel

14.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

14.3 R_QSPI_GetVersion

Format

void R_QSPI_GetVersion(void *buf)

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

void

Properties

Prototyped in file “r_qspi_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
R_QSPI_GetVersion ((void*) &qspi_driver_version);
```

Special Notes:

None

14.4 R_QSPI_ Init

Format

ER_RET R_QSPI_Init (void) ;

Parameters

None

Return Values

void

Properties

Prototyped in file “r_qspi_rzn1_if.h”

Description

Initialise QSPI IP and channel states

Reentrant

No

Example:

```
ret_val = R_QSPI_Init();
```

Special Notes:

None

14.5 R_QSPI_ Uninit

Format

ER_RET R_QSPI_Uninit (void) ;

Parameters

None

Return Values

void

Properties

Prototyped in file “r_qspi_rzn1_if.h”

Description

Uninitialise QSPI IP and channel states

Reentrant

No

Example:

```
ret_val = R_QSPI_Uninit();
```

Special Notes:

None

14.6 R_QSPI_ Open

Format

ER_RET R_QSPI_Open (uint8_t chan_num) ;

Parameters

chan_num	QSPI channel number to open

Return Values

ER_OK	No error
ER_INVAL	Invalid state

Properties

Prototyped in file “r_qspi_rzn1_if.h”

Description

Opens a QSPI channel.

Reentrant

No

Example:

```
ret_val = R_QSPI_Open(QSPI_CHAN_1);
```

Special Notes:

None

14.7 R_QSPI_ Control

Format

ER_RET R_QSPI_Control (uint8_t chan_num, QSPI_CONTROL_REQUEST_E control_request, uint8_t *buf);

Parameters

chan_num	QSPI channel number to open
control_request	request to configure channel or get channel info.
buf	configuration data (to set or get).

Return Values

ER_OK	No error
ER_INVAL	Invalid state
ER_PARAM	Invalid parameter

Properties

Prototyped in file “r_qspi_rzn1_if.h”

Description

Controls an QSPI instance.

Reentrant

No

Example:

```
uint8_t qspi_driver_version[2];
ret_val = R_QSPI_Control( QSPI_CHAN_1, QSPI_CONTROL_GET_DRIVER_VERSION, qspi_driver_version);

QSPI_IF_CHANNEL_STATE_E qspi_chan_state;
ret_val = R_QSPI_Control(QSPI_CHAN_1, QSPI_CONTROL_GET_CHAN_STATE, (uint8_t *) &qspi_chan_state);
```

Special Notes:

None

14.8 R_QSPI_ Read

Format

ER_RET R_QSPI_Read (uint8_t chan_num, uint32_t offset, uint8_t *buf, uint32_t block_len) ;

Parameters

chan_num	QSPI channel number to open
offset	Offset in bytes from start of Flash from where data should be read
buf	Pointer to buffer where data read from Flash should be stored
block_len	Number of bytes to read

Return Values

ER_OK	No error
ER_INVAL	Invalid state
ER_PARAM	Invalid parameter

Properties

Prototyped in file “r_qspi_rzn1_if.h”

Description

Reads data from Flash memory via QSPI channel.

Reentrant

No

Example:

```
uint32_t flash_page = 2048;
ret_val = R_QSPI_Read(QSPI_CHAN_1, (flash_page * QSPI_PAGE_SIZE), flash_buf, QSPI_PAGE_SIZE);
```

Special Notes:

None

14.9 R_QSPI_ Erase

Format

ER_RET R_QSPI_Erase (uint8_t chan_num, uint32_t erase_block_offset, uint32_t erase_size) ;

Parameters

chan_num	QSPI channel number to open
erase_block_offset	Offset in bytes from start of Flash from where data should be read
erase_size	Pointer to buffer where data read from Flash should be stored

Return Values

ER_OK	No error
ER_INVAL	Invalid state

Properties

Prototyped in file “r_qspi_rzn1_if.h”

Description

Erases serial flash.

Note1: Minimum erasable block size is 4k bytes (0x1000 or QSPI_ERASE_BLOCK_SIZE_4K)

Note2: If erase_size is 0xffffffff then whole flash is erased

Reentrant

No

Example:

```
uint32_t flash_page = 2048;
erase_block_offset = (flash_page * QSPI_PAGE_SIZE);
ret_val = R_QSPI_Erase(QSPI_CHAN_1, erase_block_offset, QSPI_MIN_ERASE_BLOCK_SIZE);
```

Special Notes:

None

14.10 R_QSPI_ Write

Format

ER_RET R_QSPI_ Write (uint8_t chan_num, uint32_t offset, uint8_t *buf, uint32_t block_len) ;

Parameters

chan_num	QSPI channel number to open
offset	Offset in bytes from start of Flash from where data should be read
buf	Pointer to buffer where data read from Flash should be stored
block_len	Number of bytes to read

Return Values

ER_OK	No error
ER_INVAL	Invalid state
ER_PARAM	Invalid parameter

Properties

Prototyped in file “r_qspi_rzn1_if.h”

Description

Writes data to Flash memory via QSPI channel.

Reentrant

No

Example:

```
uint32_t flash_page = 2048;
ret_val = R_QSPI_Write(QSPI_CHAN_1, (flash_page * QSPI_PAGE_SIZE), flash_buf, QSPI_PAGE_SIZE);
```

Special Notes:

None

14.11 R_QSPI_ Close

Format

ER_RET R_QSPI_ Close (uint8_t chan_num) ;

Parameters

chan_num	QSPI channel number to open

Return Values

ER_OK	No error
ER_INVAL	Invalid state
ER_PARAM	Invalid parameter

Properties

Prototyped in file “r_qspi_rzn1_if.h”

Description

Closes a qspi channel.

Reentrant

No

Example:

```
ret_val = R_QSPI_Close(QSPI_CHAN_1);
```

Special Notes:

None

15. RTC API Functions

15.1 Summary

Function	Description
R_RTC_GetVersion	Returns the RTC driver version number
R_RTC_Init	Initialise RTC IP
R_RTC_Uninitialise	Un-Initialise RTC IP
R_RTC_Open	Opens the RTC
R_RTC_Close	Closes the RTC
R_RTC_Control	Set Alarm/Periodic/Interval Callbacks, reset, get RTC info, depending on control_request
R_RTC_Write	Writes time to the RTC
R_RTC_Read	Reads time from the RTC

15.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

15.3 R_RTC_GetVersion

Format

void R_RTC_GetVersion(void *buf)

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

None

Properties

Prototyped in file "r_rtc_rzn1_if.h"

Description

Returns the RTC driver version number

Reentrant

No

Example:

```
uint8_t rtc_driver_version[2];
sample_app_printf("\n **** RTC Driver Version ****\n");
R_RTC_GetVersion((void*)&rtc_driver_version);
```

Special Notes:

None

15.4 R_RTC_Init

Format

ER_RET R_RTC_Init(void)

Parameters

None

Return Values

ER_OK	No error
ER_INVAL	Driver already initialised

Properties

Prototyped in file "r_rtc_rzn1_if.h"

Description

Initialise RTC IP

Reentrant

No

Example:

```
ret_status = R_RTC_Init();
```

Special Notes:

None

15.5 R_RTC_Uninitialise

Format

ER_RET R_RTC_Uninitialise(void)

Parameters

None

Return Values

ER_OK	No error
ER_INVAL	Driver is not initialised

Properties

Prototyped in file "r_rtc_rzn1_if.h"

Description

Un-Initialise RTC IP

Reentrant

No

Example:

```
ret_status = R_RTC_Uninitialise ();
```

Special Notes:

None

15.6 R_RTC_Open

Format

ER_RET R_RTC_Open(void)

Parameters

None

Return Values

ER_OK	No error
ER_INVAL	Driver is not initialized or closed

Properties

Prototyped in file "r_rtc_rzn1_if.h"

Description

Opens the RTC

Reentrant

No

Example:

```
/* Open the RTC*/
ret_status = R_RTC_Open();
```

Special Notes:

None

15.7 R_RTC_Close

Format

ER_RET R_RTC_Close(void)

Parameters

None

Return Values

ER_OK	No error
ER_INVAL	Driver is not open

Properties

Prototyped in file "r_rtc_rzn1_if.h"

Description

Closes the RTC

Reentrant

No

Example:

```
/* Close the RTC*/
ret_status = R_RTC_Close();
```

Special Notes:

None

15.8 R_RTC_Control

Format

ER_RET R_RTC_Control(RTC_CONTROL_REQUEST_E control_request, uint8_t *buf)

Parameters

control_request	request to configure or get existing configuration
buf	configuration data (to set or get)

NOTE: API function checks that buf is non-zero.
The upper layer should check that the buffer lies within a valid address range.

Return Values

ER_OK	No error
ER_INVAL	RTC not in Open State
ER_PARAM	Invalid parameter used

Properties

Prototyped in file "r_rtc_rzn1_if.h"

Description

Set Alarm/Periodic/Interval Callbacks, reset, get RTC info, depending on control_request

Reentrant

No

Example:

```
alarmEvent.eventType = RTC_ALARM_EVENT;
alarmEvent.callback     = rtc_alarm_callback;
alarmEvent.alarmSetup.friday  = true;
alarmEvent.alarmSetup.hours      = 9;
alarmEvent.alarmSetup.minutes = 16;
ret_status = R_RTC_Control(RTC_CONTROL_SET_ALARM_CALLBACK, (void *)&alarmEvent);
```

Special Notes:

None

15.9 R_RTC_Read

Format

ER_RET R_RTC_Read(tm_t * p_current)

Parameters

p_current	Returns RTC time in standard 'C' time format

Return Values

ER_OK	No error
ER_INVAL	RTC not in Open State, or ready
ER_PARAM	Invalid parameter used

Properties

Prototyped in file "r_rtc_rzn1_if.h"

Description

Reads time from the RTC

Reentrant

No

Example:

```
tm_t currentTime;
/* Read time again and check seconds have elasped */
ret_status = R_RTC_Read(&currentTime);
```

Special Notes:

None

15.10 R_RTC_Write

Format

ER_RET R_RTC_Write(tm_t * p_current)

Parameters

p_current	Gives RTC time in standard 'C' time format

Return Values

ER_OK	No error
ER_INVAL	RTC not in Open State, or ready
ER_PARAM	Invalid parameter used

Properties

Prototyped in file "r_rtc_rzn1_if.h"

Description

Writes time to the RTC

Reentrant

No

Example:

```
/* Set time */
prevTime.tm_sec     = 15;
prevTime.tm_min     = 15;
prevTime.tm_hour    = 15;
prevTime.tm_wday    = 5;
prevTime.tm_mday    = 15;
prevTime.tm_mon     = 4;
prevTime.tm_year    = 2018 - TM_STRUCT_EPOCH;

/* Write test time */
ret_status = R_RTC_Write(&prevTime);
```

Special Notes:

None

16. Semaphore API Functions

16.1 Summary

Function	Description
R_SEMAPHORE_GetVersion	Returns the Semaphore driver version number.
R_SEMAPHORE_Init	Initialise Semaphore IP
R_ SEMAPHORE_Uninitialise	Un-Initialise Semaphore IP
R_SEMAPHORE_Control	Get the status (if and what CPU is the owner) of the given Semaphore Request to configure or get existing configuration
R_SEMAPHORE_Lock	Locks the requested Semaphore
R_SEMAPHORE_Release	Unlocks the requested Semaphore
R_SEMAPHORE_AutoCpuLock	Locks the requested Semaphore Automatically using the correct CPU Id
R_SEMAPHORE_AutoCpuRelease	Unlocks the requested Semaphore Automatically using the correct CPU Id

16.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

16.3 R_SEMAPHORE_GetVersion

Format

void R_SEMAPHORE_GetVersion (void *buf);

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

void

Properties

Prototyped in file “r_semaphore_rzn1_if.h”

Description

Returns the Semaphore driver version number.

Reentrant

No

Example:

```
R_SEMAPHORE_GetVersion((void*)&semaphore_driver_version);
```

Special Notes:

None

16.4 R_SEMAPHORE_Init

Format

ER_RET R_SEMAPHORE_Init(void);

Parameters

void

Return Values

ER_OK	No error
ER_INVAL	Driver already initialised

Properties

Prototyped in file “r_semaphore_rzn1_if.h”

Description

Initialise Semaphore IP.

Reentrant

No

Example:

```
ret_status = R_SEMAPHORE_Init();
```

Special Notes:

None

16.5 R_SEMAPHORE_Uninitialise

Format

ER_RET R_SEMAPHORE_Uninitialise(void);

Parameters

void

Return Values

ER_OK	No error
ER_INVAL	Driver not initialised

Properties

Prototyped in file “r_semaphore_rzn1_if.h”

Description

Un-Initialise Semaphore IP.

Reentrant

No

Example:

```
ret_status = R_SEMAPHORE_Uninitialise();
```

Special Notes:

None

16.6 R_SEMAPHORE_Control

Format

ER_RET R_SEMAPHORE_Control(SEMAPHORE_CONTROL_REQUEST_E control_request, void *buf);

Parameters

control_request	request to configure or get existing configuration.
buf	configuration data (to set or get).

Return Values

ER_OK	No error
ER_PARAM	Invalid parameter used

Properties

Prototyped in file “r_semaphore_rzn1_if.h”

Description

Get the status (if and what CPU is the owner) of the given Semaphore

Request to configure or get existing configuration.

Reentrant

No

Example:

```
statusReq.semId = SEMAPHORE_ID_63;
statusReq.cpuId = SEMAPHORE_CPU_NONE;
ret_status = R_SEMAPHORE_Control(SEMAPHORE_CONTROL_GET_STATUS, (void *)&statusReq);
```

Special Notes:

None

16.7 R_SEMAPHORE_Lock

Format

ER_RET R_SEMAPHORE_Lock(SEMAPHORE_CPU_E cpuId, SEMAPHORE_ID_E semId);

Parameters

cpuId	ID of the CPU that is making requesting
semId	ID of the Semaphore that request is being made against

Return Values

ER_OK	No error
ER_INVAL	Driver not initialised

Properties

Prototyped in file “r_semaphore_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
/* Get the Semaphore */
ret_status = R_SEMAPHORE_Lock(SEMAPHORE_CPU_1, SEMAPHORE_ID_63);
```

Special Notes:

None

16.8 R_SEMAPHORE_Release

Format

ER_RET R_SEMAPHORE_Release(SEMAPHORE_CPU_E cpuId, SEMAPHORE_ID_E semId);

Parameters

cpuId	ID of the CPU that is making requesting
semId	ID of the Semaphore that request is being made against

Return Values

ER_OK	No error
ER_INVAL	Driver not initialised

Properties

Prototyped in file “r_semaphore_rzn1_if.h”

Description

Unlocks the requested Semaphore.

Reentrant

No

Example:

```
/* Check Semaphore can not be released via another CPU entry point*/
ret_status = R_SEMAPHORE_Release(SEMAPHORE_CPU_4, SEMAPHORE_ID_63);
```

Special Notes:

None

16.9 R_SEMAPHORE_AutoCpuLock

Format

ER_RET R_SEMAPHORE_AutoCpuLock (SEMAPHORE_ID_E semId);

Parameters

semId	ID of the Semaphore that request is being made against

Return Values

ER_OK	No error
ER_INVAL	Driver not initialised

Properties

Prototyped in file “r_semaphore_rzn1_if.h”

Description

Locks the requested Semaphore Automatically using the correct CPU Id.

Reentrant

No

Example:

```
/* Get the Semaphore */
ret_status = R_SEMAPHORE_AutoCpuLock (SEMAPHORE_ID_63);
```

Special Notes:

None

16.10 R_SEMAPHORE_AutoCpuRelease

Format

ER_RET R_SEMAPHORE_AutoCpuRelease (SEMAPHORE_ID_E semId);

Parameters

semId	ID of the Semaphore that request is being made against

Return Values

ER_OK	No error
ER_INVAL	Driver not initialised

Properties

Prototyped in file “r_semaphore_rzn1_if.h”

Description

Unlocks the requested Semaphore Automatically using the correct CPU Id.

Reentrant

No

Example:

```
/* Get the Semaphore */
ret_status = R_SEMAPHORE_ AutoCpuRelease (SEMAPHORE_ID_63);
```

Special Notes:

None

17. SPI API Functions

17.1 Summary

Function	Description
R_SPI_GetVersion	Returns the SPI driver version number
R_SPI_Init	Reset all channels states to closed. Initializes SPI driver.
R_SPI_Uninit	Un-initialise SPI driver and channels interface state
R_SPI_Open	Opens a SPI channel & initializes channel state.
R_SPI_Control	Controls/gets information for the SPI driver/channel: • Gets channel state • Sets interrupt callback function • Gets/sets Channel configuration • Reserves/releases UART channel for/from DMA transmit/receive transaction • Cancels a SPI Tx or Rx transfer
R_SPI_Read	Reads data from a SPI channel in BLOCKING or NON-BLOCKING mode (only for half-duplex mode).
R_SPI_Write	Writes data to a SPI channel in BLOCKING or NON-BLOCKING mode (only for half-duplex mode).
R_SPI_ReadWrite	Writes/reads data to/from a SPI channel in BLOCKING or NON-BLOCKING mode (only for full-duplex mode).
R_SPI_Close	Closes a SPI channel

17.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

17.3 R_SPI_GetVersion

Format

void R_SPI_GetVersion (void *buf);

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

void

Properties

Prototyped in file “r_spi_rzn1_if.h”

Description

Returns the SPI driver version number

Reentrant

No

Example:

```
R_SPI_GetVersion((void *)spi_driver_version);
```

Special Notes:

None

17.4 R_SPI_Init

Format

void R_SPI_Init (void)

Parameters

None

Return Values

None

Properties

Prototyped in file “r_spi_rzn1_if.h”

Description

Initialise SPI driver and channels interface state

Reentrant

No

Example:

```
R_SPI_Init ();
```

Special Notes:

None

17.5 R_SPI_Uninit

Format

void R_SPI_Uninit (void)

Parameters

None

Return Values

None

Properties

Prototyped in file “r_spi_rzn1_if.h”

Description

UnInitialise SPI driver and channels interface state

Reentrant

No

Example:

```
R_SPI_Uninit ();
```

Special Notes:

None

17.6 R_SPI_ Open

Format

ER_RET R_SPI_ Open (uint8_t chan_num)

Parameters

chan_num	SPI channel number to open

Return Values

ER_OK	No error
ER_PARAM	Parameter error
ER_INVAL	Channel has not been initialized

Properties

Prototyped in file “r_spi_rzn1_if.h”

Description

Opens an SPI channel

Reentrant

No

Example:

```
ret_val = R_SPI_ Open (SPI_CHAN_1);
```

Special Notes:

None

17.7 R_SPI_ Control

Format

ER_RET R_SPI_ Control (uint8_t chan_num, spi_cmd_t control_request, void *spi_ctrl)

Parameters

chan_num	SPI channel number to control
control_request	request to configure channel or get channel info.
spi_ctrl	configuration data (to set or get).

Return Values

ER_OK	No error
ER_PARAM	Parameter error
ER_INVAL	Channel has not been initialized
ER_BUSY	Channel in use (unavailable)

Properties

Prototyped in file “r_spi_rzn1_if.h”

Description

Controls & gets information for the SPI driver or for a SPI channel:

- Configures channel parameters,
- Gets Channel State and Channel configuration
- Sets Channel configuration
- Sets SPI-DMA Rx and Tx configuration
- Cancels a SPI Tx or Rx transfer

Reentrant

No

Example:

```
ret_val = R_SPI_Control(SPI_CHAN_1, SPI_CONTROL_SET_CHAN_STATE, &spi_chan_state);
    /* Set SPI config with baudrate 9600 */
spi_channel_config_t spi_set_chan_config
spi_set_chan_config.master_config.baudrate = 9600;
spi_set_chan_config.microwire_config.control_frame_size = 8;
spi_set_chan_config.master_config.slave_select = 0;
spi_set_chan_config.frame_format = spi_texas_instruments;
spi_set_chan_config.data_frame_size = 8;

ret_val = R_SPI_Control(SPI_CHAN_1, SPI_CONTROL_SET_CHAN_CONFIG, &spi_set_chan_config);
```

Special Notes:

None

17.8 R_SPI_ Read

Format

ER_RET R_SPI_ Read (uint8_t chan_num, spi_transfer_data_t transfer_data)

Parameters

chan_num	SPI channel number
transfer_data:	
read_buf	Where data received is stored
write_buf	Data to be written
data_frames	Number of data frames to read/write
transfer_mode_tx	The transfer mode of the write
transfer_mode_rx	The transfer mode of the read
transfer_type	The type of transfer being requested

Return Values

ER_OK	No error
ER_PARAM	Parameter error
ER_INVAL	Channel has not been initialized

Properties

Prototyped in file “r_spi_rzn1_if.h”

Description

Only reads data from an SPI channel. Do not work for full duplex mode.

If BLOCKING (synchronous) mode, then returns when all the requested data

has been transferred or a timeout occurs.

If NON-BLOCKING/DMA (asynchronous) mode, then initiates the transfer process and returns.

In this case a callback parameter is required to be passed through a previous call to R_SPI_Control().

Reentrant

No

Example:

```
spi_transfer_data_t SPI_CHAN_TRANS_DATA[MAX_SPI_CHANNELS]
ret_val = R_SPI_Read(SPI_CHAN_1, SPI_CHAN_TRANS_DATA[chan_num]);
```

Special Notes:

None

17.9 R_SPI_ Write

Format

ER_RET R_SPI_ Write (uint8_t chan_num, spi_transfer_data_t transfer_data)

Parameters

chan_num		SPI channel number
transfer_data:	read_buf	Where data received is stored
	write_buf	Data to be written
	data_frames	Number of data frames to read/write
	transfer_mode_tx	The transfer mode of the write
	transfer_mode_rx	The transfer mode of the read
	transfer_type	The type of transfer being requested

Return Values

ER_OK	No error
ER_PARAM	Parameter error
ER_INVAL	Channel has not been initialized

Properties

Prototyped in file “r_spi_rzn1_if.h”

Description

Only writes data to an SPI channel. Do not work for full duplex mode.

If BLOCKING (synchronous) mode, then returns when all the requested data

has been transferred or a timeout occurs.

If NON-BLOCKING/DMA (asynchronous) mode, then initiates the transfer process and returns.

In this case a callback parameter is required to be passed through a previous call to R_SPI_Control().

Reentrant

No

Example:

```
spi_transfer_data_t SPI_CHAN_TRANS_DATA[MAX_SPI_CHANNELS]
ret_val = R_SPI_Write(SPI_CHAN_1, SPI_CHAN_TRANS_DATA[chan_num]);
```

Special Notes:

None

17.10 R_SPI_ ReadWrite

Format

ER_RET R_SPI_ ReadWrite(uint8_t chan_num, spi_transfer_data_t transfer_data)

Parameters

chan_num		SPI channel number
transfer_data:	read_buf	Where data received is stored
	write_buf	Data to be written
	data_frames	Number of data frames to read/write
	transfer_mode_tx	The transfer mode of the write
	transfer_mode_rx	The transfer mode of the read
	transfer_type	The type of transfer being requested

Return Values

ER_OK	No error
ER_PARAM	Parameter error
ER_INVAL	Channel has not been initialized

Properties

Prototyped in file “r_spi_rzn1_if.h”

Description

Writes/reads data to/from an SPI channel. Do not work for full duplex mode.

If BLOCKING (synchronous) mode, then returns when all the requested data

has been transferred or a timeout occurs.

If NON-BLOCKING/DMA (asynchronous) mode, then initiates the transfer process and returns.

In this case a callback parameter is required to be passed through a previous call to R_SPI_Control().

Reentrant

No

Example:

```
spi_transfer_data_t SPI_CHAN_TRANS_DATA[MAX_SPI_CHANNELS]
ret_val = R_SPI_ ReadWrite(SPI_CHAN_1, SPI_CHAN_TRANS_DATA[chan_num]);
```

Special Notes:

None

17.11 R_SPI_ Close

Format

ER_RET R_SPI_ Close (uint8_t chan_num)

Parameters

chan_num	SPI channel number to close.

Return Values

ER_OK	No error
ER_PARAM	Parameter error
ER_INVAL	Channel has not been initialized

Properties

Prototyped in file “r_spi_rzn1_if.h”

Description

Opens an SPI channel

Reentrant

No

Example:

```
ret_val = R_SPI_ Close (SPI_CHAN_1);
```

Special Notes:

None

18. SDIO API Functions

18.1 Summary

Function	Description
R_SDIO_GetVersion	Returns driver version.
R_SDIO_Init	Initialises the SDIO IP and instance states
R_SDIO_Open	Opens an SDIO controller.
R_SDIO_Control	Controls an SDIO instance • Sends (application) command • Sets clock divider • Sets insert/eject interrupt callback function • Initialises card in SD mode • Sets the data bus width • Gets card parameters
R_SDIO_Read	Reads data from a Card.
R_ SDIO_Write	Writes data to a Card.
R_SDIO_Close	Closes a SDIO controller

18.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

18.3 R_SDIO_GetVersion

Format

void R_SDIO_GetVersion(void *buf)

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

void

Properties

Prototyped in file “r_sdio_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
R_SDIO_GetVersion ((void*) &sdio_driver_version);
```

Special Notes:

None

18.4 R_SDIO_Init

Format

void R_SDIO_Init (void);

Parameters

None

Return Values

None

Properties

Prototyped in file “r_sdio_rzn1_if.h”

Description

Initialises the SDIO IP and instance states

Reentrant

No

Example:

```
R_SDIO_Init();
```

Special Notes:

None

18.5 R_SDIO_Open

Format

ER_RET R_SDIO_Open (uint8_t SDIO_number);

Parameters

SDIO_number	Which SDIO controller to configure.

Return Values

ER_OK	No error
ER_INVAL	Invalid state

Properties

Prototyped in file “r_sdio_rzn1_if.h”

Description

Opens an SDIO controller.

Reentrant

No

Example:

```
ret_val = R_SDIO_Open(SDIO0);
```

Special Notes:

None

18.6 R_SDIO_Control

Format

ER_RET R_SDIO_Control (uint8_t SDIO_number, SDIO_CONTROL_REQUEST control_request, void *control);

Parameters

SDIO_number	Which SDIO controller to configure.
control	Which control action to implement.
ctrl_param	The control parameter that is used along with the control action to configure the controller instance a certain way.

Return Values

ER_OK	No error
ER_INVAL	Invalid state
ER_PARAM	Invalid parameter

Properties

Prototyped in file “r_sdio_rzn1_if.h”

Description

Controls an SDIO instance.

Reentrant

No

Example:

```
SDIO_CARD_PARAMETERS parameters;
ret_val = R_SDIO_Control(SDIO0, SDIO_CONTROL_GET_CARD_PARAMS, &parameters);

SDIO_TRANS_WIDTH width = SDIO_WIDTH_4_BIT;
ret_val = R_SDIO_Control(SDIO0, SDIO_CONTROL_SET_BUS_WIDTH, &width);
```

Special Notes:

None

18.7 R_SDIO_Read

Format

ER_RET R_SDIO_Read (uint8_t SDIO_number, SDIO_TRANS_CTRL trans_control, uint32_t start_address);

Parameters

SDIO_number	Which SDIO instance to read from.
trans_control	transfer control of the read transfer.
start_address	The address in the card to start reading from

Return Values

ER_OK	No error
ER_INVAL	Invalid state
ER_PARAM	Invalid parameter

Properties

Prototyped in file “r_sdio_rzn1_if.h”

Description

Reads data from a Card.

Reentrant

No

Example:

```
uint32_t flash_page = 2048;
ret_val = R_QSPI_Read(QSPI_CHAN_1, (flash_page * QSPI_PAGE_SIZE), flash_buf, QSPI_PAGE_SIZE);
```

Special Notes:

None

18.8 R_SDIO_Write

Format

ER_RET R_SDIO_Write (uint8_t SDIO_number, SDIO_TRANS_CTRL trans_control, uint32_t start_address);

Parameters

SDIO_number	Which SDIO instance to read from.
trans_control	transfer control of the read transfer.
start_address	The address in the card to start reading from

Return Values

ER_OK	No error
ER_INVAL	Invalid state
ER_PARAM	Invalid parameter

Properties

Prototyped in file “r_sdio_rzn1_if.h”

Description

Writes data to a Card.

Reentrant

No

Example:

```
SDIO_TRANS_CTRL trans_control ={ .block_count = 1, .buffer =
&SD_write_buffer, .trans_complete_callback = &sdio_interrupt_callback };


ret_val = R_SDIO_Write(SDIO0, trans_control, 0xA0000);
```

Special Notes:

None

18.9 R_SDIO_Close

Format

ER_RET R_SDIO_Close (uint8_t SDIO_number);

Parameters

SDIO_number	Which SDIO controller to configure.

Return Values

ER_OK	No error
ER_INVAL	Invalid state

Properties

Prototyped in file “r_qspi_rzn1_if.h”

Description

Close an SDIO controller.

Reentrant

No

Example:

```
ret_val = R_SDIO_Close(SDIO0);
```

Special Notes:

None

19. Timer API Functions

19.1 Summary

Function	Description
R_TIMER_Init	Initialize interval timer, set initial interval timer counter and enable timer interrupts for subtimer 'chan' of timer block1
R_TIMER_DMAConfigure	Unpacks dma information about the channel.
R_TIMER_DMADisable	Disables dma requests.
R_TIMER_Start	Start the Timer for this Timer channel
R_TIMER_Stop	Stop the Timer for this Timer channel
Delay_usec	Timer delay using SYS_TIMER_CHANNEL
R_TIMER_Delay	Timer delay on a specific channel
R_TIMER_GetVersion	Returns the Timer driver version

19.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

19.3 R_TIMER_Init

Format

ER_RET R_TIMER_Init (uint8_t chan, uint8_t time_base, timer_interval_callback callback_function);

Parameters

chan	channel number (i.e. sub-timer)
time_base	Time base
callback_function	Pointer tot he callback function.

Return Values

ER_OK	No error
ER_INVAL	Invalid state
ER_PARAM	Invalid parameter

Properties

Prototyped in file “r_timer_rzn1_if.h”

Description

Initialize interval timer, set initial interval timer counter and enable timer interrupts for subtimer 'chan' of timer block1.

Reentrant

No

Example:

```
ret_val = R_TIMER_Init(timer_chan, TIMER_TIME_BASE_1MHz, &dma_timer1_callback);
```

Special Notes:

None

19.4 R_TIMER_DMAConfigure

Format

ER_RET R_TIMER_DMAConfigure (uint8_t chan, uint8_t dma_chan);

Parameters

chan	channel number (i.e. sub-timer)
dma_chan	DMA channel number

Return Values

ER_OK	No error
ER_INVAL	Invalid state
ER_PARAM	Invalid parameter

Properties

Prototyped in file “r_timer_rzn1_if.h”

Description

Unpacks dma information about the channel.

Reentrant

No

Example:

```
ret_val = R_TIMER_DMAConfigure (timer_chan, DMA_CHAN_7);
```

Special Notes:

None

19.5 R_TIMER_DMADisable

Format

ER_RET R_TIMER_DMADisable (uint8_t chan) ;

Parameters

chan	channel number (i.e. sub-timer)

Return Values

ER_OK	No error
ER_INVAL	Invalid state

Properties

Prototyped in file “r_timer_rzn1_if.h”

Description

Disables DMA requests.

Reentrant

No

Example:

```
ret_val = R_TIMER_DMADisable (timer_chan);
```

Special Notes:

None

19.6 R_TIMER_Start

Format

ER_RET R_TIMER_Start (uint8_t chan, uint32_t i_time) ;

Parameters

chan	channel number (i.e. sub-timer)
i_time	interval time (ticks)

Return Values

ER_OK	No error
ER_PARAM	Invalid parameter

Properties

Prototyped in file “r_timer_rzn1_if.h”

Description

Set the interval time and start the interval timer counter.

Reentrant

No

Example:

```
ret_val = R_TIMER_Start ( SUB_TIMER_CHANNEL7, 20000ul);
```

Special Notes:

None

19.7 R_TIMER_Stop

Format

ER_RET R_TIMER_Stop (uint8_t chan) ;

Parameters

chan	channel number (i.e. sub-timer)

Return Values

ER_OK	No error
ER_PARAM	Invalid parameter

Properties

Prototyped in file “r_timer_rzn1_if.h”

Description

Stop the Timer for this Timer channel.

Reentrant

No

Example:

```
ret_val = R_TIMER_Stop (SYS_TIMER_CHANNEL);
```

Special Notes:

None

19.8 Delay_usec

Format

ER_RET Delay_usec(uint32_t usecs_delay);

Parameters

usecs_delay	length of time to delay in usecs

Return Values

None

Properties

Prototyped in file “r_timer_rzn1_if.h”

Description

Timer delay using SYS_TIMER_CHANNEL.

Reentrant

No

Example:

```
ret_val = Delay_usec (1000000);
```

Special Notes:

None

19.9 R_TIMER_Delay

Format

ER_RET R_TIMER_Delay (uint32_t chan, uint32_t delay_interval_count) ;

Parameters

chan	channel number (i.e. sub-timer)
delay_interval_count	count of timer intervals to delay

Return Values

ER_OK	No error
ER_INVAL	Invalid state
ER_PARAM	Invalid parameter

Properties

Prototyped in file “timer.h”

Description

Timer delay on a specific channel.

If time base is 1MHz then 1 interval count will represent 1us.

If time base is 25MHz then 1 interval count will represent 40ns.

Reentrant

No

Example:

```
ret_val = R_TIMER_Delay (SYS_TIMER_CHANNEL, 500);
```

Special Notes:

None

19.10 R_TIMER_GetVersion

Format
void R_TIMER_GetVersion (void *buf);

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values
void

Properties
Prototyped in file “r_timer_rzn1_if.h”

Description
Returns driver version.

Reentrant
No

Example:
```
*(uint32_t *)timer_driver_version = R_TIMER_GetVersion();
```

Special Notes:
None

20. UART API Functions

20.1 Summary

Function	Description
R_UART_GetVersion	Returns driver version.
R_UART_Init	Reset all channels states to closed. Initializes UART driver
R_UART_Open	Opens a UART channel & initializes channel state
R_UART_Control	Controls & gets information for the UART driver or for a UART channel: • Sets/gets channel configuration • Gets channel state • Reserves/releases UART channel for/from DMA transmit/receive transaction • Cancel UART transfer • Resets a channel.
R_UART_Read	Reads data from a UART channel in BLOCKING or NON-BLOCKING mode
R_UART_Write	Writes data to a UART channel in BLOCKING or NON-BLOCKING mode
R_UART_Close	closes a UART channel

20.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

20.3 R_UART_GetVersion

Format

void R_UART_GetVersion(void *buf)

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

void

Properties

Prototyped in file “r_uart_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
R_UART_ GetVersion ((void*) &gpio_driver_version);
```

Special Notes:

None

20.4 R_UART_Init

Format

void R_UART_Init (void)

Parameters

None

Return Values

void

Properties

Prototyped in file “r_uart_rzn1_if.h”

Description

Initialize UART driver and channels interface state

Reentrant

No

Example:

```
R_UART_Init();
```

Special Notes:

None

20.5 R_UART_Open

Format

ER_RET R_UART_open(uint8_t chan_num);

Parameters

chan_num	UART channel number to open

Return Values

ER_OK	No error
ER_PARAM	Parameter error
ER_INVAL	Channel has not been initialized

Properties

Prototyped in file “r_uart_rzn1_if.h”

Description

Opens a UART channel

Reentrant

No

Example:

```
ret_val = R_UART_Open(UART_CHAN_2);
```

Special Notes:

None

20.6 R_UART_Control

Format

ER_RET R_UART_Control(uint8_t chan_num, UART_CONTROL_REQUEST_E control_request, uint8_t *buf);

Parameters

chan_num	UART channel number to control
control_request	Request to configure channel or get channel info.
buf	Configuration data (to set or get)

Return Values

ER_OK	No error
ER_PARAM	Invalid Channel Number or Control Request
ER_INVAL	Channel state incorrect for configuration request

Properties

Prototyped in file “r_uart_rzn1_if.h”

Description

Set and Get the Configuration for the UART driver or for a UART channel

- Configures channel parameters,
- Gets UART driver version number, Channel State and Channel configuration
- Sets Channel configuration
- Sets UART-DMA Rx and Tx configuration
- Cancel a UART Tx or Rx transfer
- Resets the UART Driver

Reentrant

No

Example:

```
ret_val = R_UART_Control(UART_CHAN_1, UART_CONTROL_GET_DRIVER_VERSION, uart_driver_version);

ret_val = R_UART_Control(UART_CHAN_1, UART_CONTROL_GET_CHAN_STATE, (uint8_t *) &uart_chan_state);

    /* Set UART config with baudrate 9600 */
    uart_config.baudrate = 9600;
    uart_config.parity = UART_PARITY_ODD;
    uart_config.flow_control = UART_FLOW_CONTROL_NONE;
    uart_config.stopbits = 1;
```

```
    uart_config.databits = 8;
    ret_val = R_UART_Control(UART_CHAN_1, UART_CONTROL_SET_CHAN_CONFIG, (uint8_t *) &uart_config);
```

Special Notes:

None

20.7 R_UART_Read

Format

ER_RET R_UART_Read(uint8_t chan_num, uint8_t *buf, size_t *num_chars_ptr, UART_TRANSFER_MODE_E mode, uart_transfer_complete_callback_t rx_complete_callback);

Parameters

chan_num	UART channel number to read
buf	where data received is stored
num_chars_ptr	num chars to read
mode	BLOCKING or NON-BLOCKING read
rx_complete_callback	function to call when all data has been read (for NON-BLOCKING read)

Return Values

ER_OK	No error
ER_PARAM	Invalid Channel Number
ER_INVAL	Channel state incorrect for Read

Properties

Prototyped in file “r_uart_rzn1_if.h”

Description

Reads data from a UART channel.

If BLOCKING (synchronous) mode then returns when all the requested data when it has been read or a timeout occurs.

If NON-BLOCKING (asynchronous) mode then initiates the read process and returns. In this case a callback parameter is required to be passed.

Reentrant

No

Example:

```
ret_val = R_UART_Read(UART_CHAN_1, (uint8_t *) uart_buf, &max_chars,
UART_TRANSFER_MODE_NON_BLOCKING, &rx_complete_callback);
```

Special Notes:

None

20.8 R_UART_Write

Format

ER_RET R_UART_Write(uint8_t chan_num, const uint8_t *buf, size_t *num_chars_ptr,
UART_TRANSFER_MODE_E mode, uart_transfer_complete_callback_t tx_complete_callback);

Parameters

chan_num	UART channel number to write to
buf	Data to be written
num_chars_ptr	num chars to write
mode	BLOCKING or NON-BLOCKING write
tx_complete_callback	function to call when all data has been written (for NON-BLOCKING write)

Return Values

ER_OK	No error
ER_PARAM	Invalid Channel Number
ER_INVAL	Channel state incorrect for Write

Properties

Prototyped in file “r_uart_rzn1_if.h”

Description

Writes data to a UART channel.

If BLOCKING (synchronous) mode then returns when all the requested data has been written or a timeout occurs.

If NON-BLOCKING (asynchronous) mode then initiates the write process and returns.

In this case a callback parameter is required to be passed

Reentrant

No

Example:

Blocking Write

```
ret_val = R_UART_Write(UART_CHAN_1, (const uint8_t *) uart_buf, &max_chars,
UART_TRANSFER_MODE_BLOCKING, (void *) 0);
```

Non-Blocking Write

```
ret_val = R_UART_Write(UART_CHAN_1, (const uint8_t *) uart_buf, &max_chars,
UART_TRANSFER_MODE_NON_BLOCKING, &tx_complete_callback);
```

Special Notes:

None

20.9 R_UART_Close

Format

ER_RET R_UART_Close(uint8_t chan_num);

Parameters

chan_num	UART channel number to close

Return Values

ER_OK	No error
ER_PARAM	Invalid Channel Number
ER_INVAL	Channel is not Open, state is not UART_CHANNEL_STATE_OPEN

Properties

Prototyped in file “r_uart_rzn1_if.h”

Description

closes a UART channel.

Reentrant

No

Example:

```
ret_val = R_UART_Close(UART_CHAN_1);
```

Special Notes:

None

21. USB CDC Function Driver

The RZ/N1 USB function bare metal driver code and sample application provide a basis for a developer to add USB device functionality to an RZ/N1-based system.

The USB function bare metal driver includes:

- USBf Basic module (HAL and USBf Core)
- USBf CDC module (CDC class driver for USBf)
- USB Common module (definitions and code common to all USB modules including USB Class modules which may be added in the future)
- USB CDC Sample application demonstrating how to use the USBf driver

The relationship between the modules is shown in Section 3 Fig.2

The software is available as source written in ANSI C.

This manual includes an outline description of the USBf Basic and Common modules and the USBf CDC module. Further details about these modules may be found in the RZ/N1 design specifications.

This manual fully describes the USB function driver APIs (all modules) and the USBf CDC Sample Application.

21.1 Project Description

The RZN1 bare metal drivers project contains a suite of drivers for the RZN1 family of devices including RZ/N1d, RZ/N1s and RZN1l.

The project also includes a Sample Application which implements a menu-driven set of functional tests for each bare metal driver. The USB function test menu provides selections to run functional tests or a CDC sample application which connects to a USB Host and exchanges serial data. The structure of the Sample Test application is shown in Figure 1 **RZ/N1 bare metal drivers - Sample Test application structure**

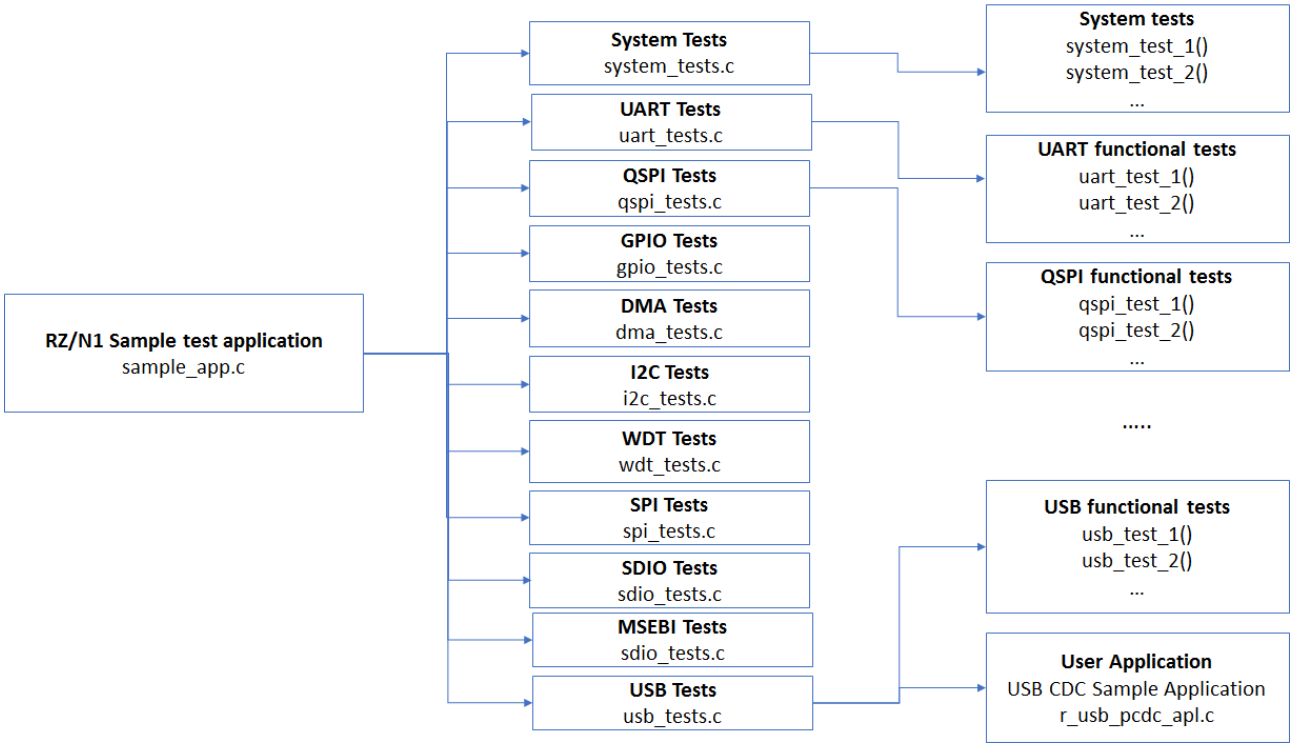


Figure 1 RZ/N1 bare metal drivers - Sample Test application structure

The USB test application and how it interacts with the USB bare metal drivers is shown in Figure 2 **RZ/N1 bare metal drivers project, USB modules**.

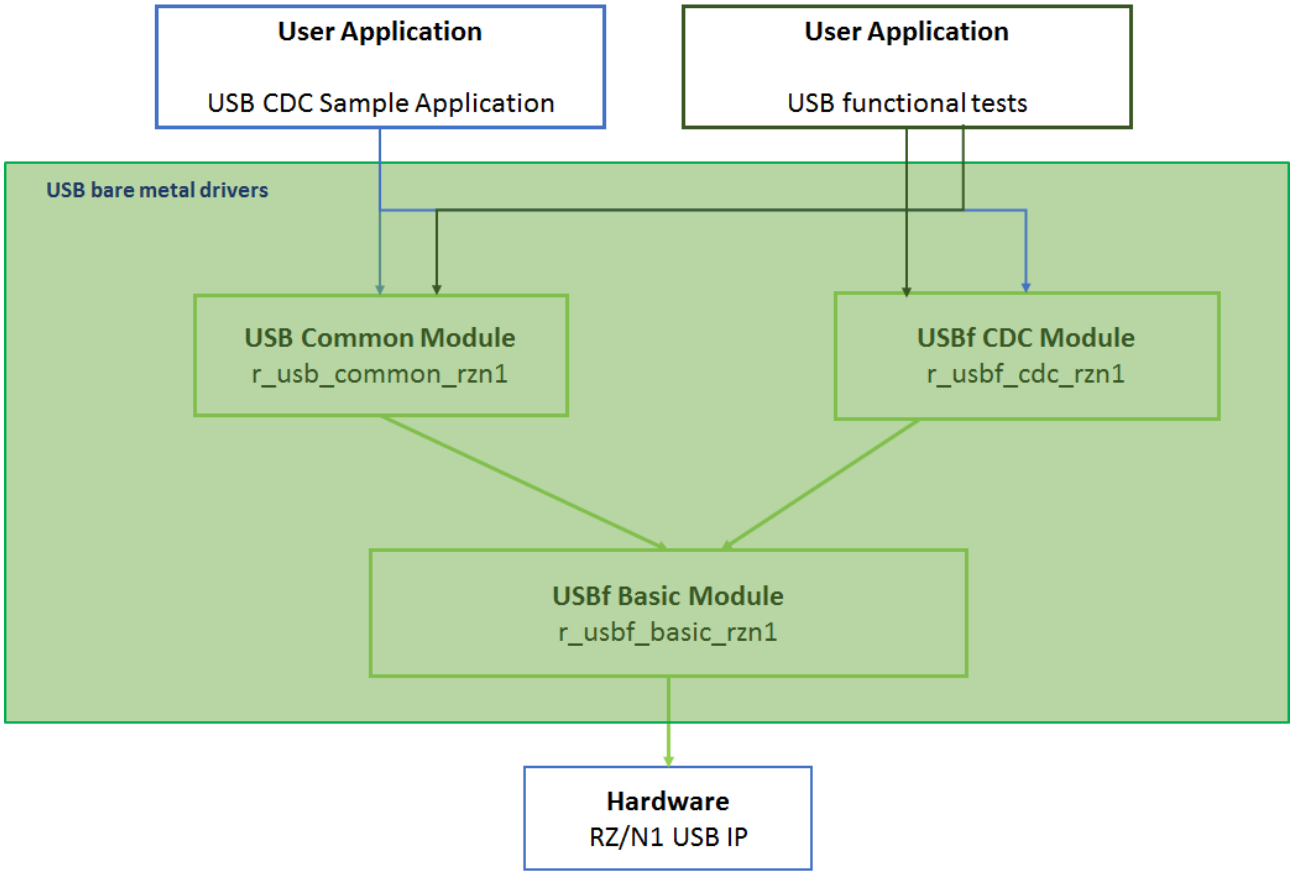


Figure 2 RZ/N1 bare metal drivers project, USB modules

21.1.1 File structure

The folder structure and naming of the USB modules is compatible with the RZ/N1 Bare Metal Drivers FIT folder structure.

Three FIT modules form the USB Function + CDC Class driver (under r_usb_rzn1 FIT folder):

(i) r_usb_common_rzn1

The ‘r_usb_common_rzn1’ module provides a way to add any generic USB API’s:

- where a Basic layer function needs to be called directly from the User application
- for generic USB APIs
- for common definitions

(ii) r_usbf_basic_rzn1

The USBf basic module functions and definitions. This consists of the USBf Core and HAL layers of the USBf stack

(iii) r_usbf_cdc_rzn1

USBf CDC functions

The CDC Sample application is in the project folder (osless_sample) src sub-folder where the menu-driven test application and functional tests for all Bare Metal drivers are located.

21.2 USB Common Module

This module provides a Class-independent interface to the USBf driver for any API functions which may apply to any Class.

The USB Common module functions will also be used to call USB Host driver functions (when the Host driver is implemented) as well as USB Function driver functions, where the API is the same.

All USB IP register definition files are included in this module.

USB definition files common to all USB Classes and common to both Host and Function drivers are also included in this module.

The USB Common module consists of the following files: -

r_usb_rzn1_if.c

r_usb_common.h

r_usb_rzn1_reg_defs.h

r_usb_rzn1_if.h

21.3 USB Common API

21.3.1 R_USB_GetVersion

Format

void R_USB_GetVersion (void *buf)

Parameters

buf	void *	Pointer to buffer where USB Major & Minor version numbers are returned

Return Values

None

Properties

Prototyped in file “r_usb_rzn1_if.h”

Description

Called by user application.

Returns the USB Function driver version number.

Reentrant

No

Example:

```
R_USB_GetVersion( (void*) &usb_driver_version);
```

21.3.2 R_USB_Init

Format

ER_RET R_USB_Init (void)

Parameters

None

Return Values

ER_RET	Error Value. ER_OK on success. ER_SYS for USB system error

Properties

Prototyped in file “r_usb_rzn1_if.h”

Description

Initialise USB bare metal driver.

Reentrant

No

Example:

```
/* Initialise USB module */
ret_val = R_USB_Init();
```

Special Notes:

None

21.4 USBf CDC Module

Communication Device Class
The CDC class, ACM sub-class allows a host to see a device as a standard serial (COM) port. This is particularly useful when working with legacy applications that use serial communications. Bulk IN and Bulk OUT transfers are used for all non-setup data.
The CDC module utilises the USB Core layer for the processing of all standard requests. In addition, it processes the following class requests:
GET_LINE_CODING
SET_LINE_CODING (As required by MS HyperTerminal)
SET_CONTROL_LINE_STATE
The CDC class is supported by MS Windows so there is no need to develop a custom Windows kernel driver. However, a custom ‘inf’ file is required, the sample CDC application includes such a file. When a CDC ACM device is plugged into a Windows PC an additional (virtual) COM port will become available that applications can use just like a standard COM port.

The CDC module consists of the following files:-

r_usb_pcdc_api.c

r_usb_pcdc_driver.c

r_usb_pcdc_local.h

r_usb_pcdc_if.h

21.5 USBf CDC API

21.5.1 R_usb_pcdc_SendData

Format

void R_usb_pcdc_SendData(uint8_t *Table, uint32_t size, USB_UTR_CB_t complete);

Parameters

uint8_t	*Table	Pointer to Data stored buffer
uint32_t	size	Data size
USB_CB_t	complete	Pointer to Callback function

Return Values

None

Properties

Prototyped in file “r_usb_pcdc_if.h”

Description

Handles request to send data to Host. Called from cdc_data_transfer() as part of the CDC application in r_usb_pcdc_apl.c .

Reentrant

No

Example:

```
R_usb_pcdc_SendData( cdc_trans_data, cdc_trans_len, &cdc_write_complete );
```

Special Notes:

None

21.5.2 R_usb_pcdc_ReceiveData

Format

void R_usb_pcdc_ReceiveData (uint8_t *Table, uint32_t size, USB_UTR_CB_t complete)

Parameters

uint8_t *table	Pointer to buffer for received data
uint32_t size	Data size in bytes
USB_UTR_CB_t complete	Pointer to callback function

Return Values

None

Properties

Prototyped in file “r_usb_pcdc_if.h”

Description

Handles request to receive data from Host. Called from cdc_data_transfer() as part of the CDC application in r_usb_pcdc_apl.c .

Reentrant

No

Example:

```
R_usb_pcdc_ReceiveData( cdc_trans_data, CDC_DATA_LEN, &cdc_read_complete );
```

Special Notes:

None

21.5.3 R_usb_pcdc_SerialStateNotification

Format

void R_usb_pcdc_SerialStateNotification (USB_SCI_SerialState_t serial_state, USB_UTR_CB_t complete)

Parameters

USB_SCI_SerialState_t serial_state	State of Serial Port
USB_UTR_CB_t complete	Pointer to callback function

Return Values

None

Properties

Prototyped in file “r_usb_pcdc_if.h”

Description

Send state of Serial port to Host

Reentrant

No

21.5.4 R_usb_pcdc_ctrltrans

Format

void R_usb_pcdc_ctrltrans(USB_REQUEST_t *preq, uint16_t ctsq)

Parameters

USB_REQUEST_t * preq	Class request information
uint16_t ctsq	Control Transfer stage or state

Return Values

None

Properties

Prototyped in file “r_usb_pcdc_if.h”

Description

Processing for next stage of a Control Transfer

Called the USBf Basic module Core layer to handle the next stage of a Control Transfer, depending on the Control Transfer state.

Reentrant

No

21.6 USBf Basic module

This module provides the USB stack for the USB function driver

It incorporates the USBf HAL and USBf Core.

21.6.1 USBf HAL

The HAL is a hardware specific layer that provides a non-hardware specific API. The HAL supports the following transfer modes:
Control (Setup, Data IN/OUT, Status)
Bulk (IN and OUT)

The HAL manages the Control Endpoint (EP0) state to establish and configure the connection to a Host. It makes calls back to the Core layer to handle Class requests and responses.

The HAL module consists of the following files:-

r_usbf_hal.c	-	provides a hardware independent API to the USB peripheral
r_usbf_hal.h	-	definitions for HAL layer
r_usbf_driver.c	-	hardware dependent functions and interrupt handling
r_usbf_driver.h	-	definitions for HAL hardware functions
r_usbf_dataio.c	-	USBf data transfer functions
r_usb_cdefusbip.h	-	register and state definitions for HAL layer
r_usbf_controlrw.c	-	Handles data IO for Control Transfers

21.6.2 USBf Core

The USB Core layer handles standard USB requests common to all USB devices during the enumeration stage. The USB Core requires initialising with the descriptors for the class or vendor specific implementation for the device being implemented. It uses the USB HAL, which it initializes, to access the USB IP.

The Core module consists of the following files:-

r_usbf_core.c

r_usbf_core.h

The following Standard Requests are handled:

- Get_Status
- Clear_Feature
- Set_Feature
- Get_Descriptor
- Get_Configuration
- Set_Configuration
- Get_Interface
- Set_Interface

The following Get_Status requests are handled:

- Recipient Device
- Recipient Interface
- Recipient End point

A Get_Status Standard request can be directed at the device, interface or endpoint. When directed to a device it returns flags indicating the status of remote wakeup and if the device is self-powered.

However, if the same request is directed at the interface it always returns zero, or should it be directed at an endpoint will return the halt flag for the endpoint.
Clear_Feature and Set_Feature requests can be used to set boolean features. These commands are implemented only for endpoint recipient. Only endpoint feature selector values may be used when the recipient is an endpoint.

Get_Descriptor returns the specified descriptor if the descriptor exists.
The Get_Desriptor command is handled for following descriptor types:

- Device
- Configuration
- String
    - Language ID (Only a single language ID is supported, and this is currently English)
    - Manufacturer
    - Product
    - Serial Number
    - Device Qualifier
    - Other Speed Configuration

Get_Configuration request returns the current device configuration value. A byte will be returned during the data stage indicating the devices status. A zero value means the device is not configured and a non-zero value indicates the device is configured.
The Get_Interface request should return the selected alternate setting for the specified interface.
The Set_Interface request set the Alternative Interface setting. If USB devices have configurations with interfaces that have mutually exclusive settings, then Set_Interface request allows the host to select the desired alternate setting. This stack only supports a default setting for the specified interface, so a STALL will be returned in the Status stage of the request.
The USB CDC API consists of a single function called ‘USBCORE_Init’. This initialises the USBCore and the HAL. In addition to passing device descriptors to this function it also requires call back functions for the following conditions:

- A Setup packet has been received that this layer can’t handle. This enables a higher layer to handle class or vender specific requests.
- A Control Data Out has completed following a setup packet that is being handled by the layer above.
- The USB cable has been connected or disconnected.
- An unhandled error has occurred.

21.7 USBf Basic API

21.7.1 R_USBf_Init

Format

ER_RET R_USBf_Init (void)

Parameters

None

Return Values

ER_RET	Error Value. ER_OK on success. ER_SYS for USB system error

Properties

Prototyped in file “r_usbf_basic_rzn1_if.h”

Description

Initialise USB function bare metal driver

Reentrant

No

21.7.2 R_USB_Open

Format

ER_RET R_USB_Open (void)

Parameters

None

Return Values

ER_RET	Error Value ER_OK on success.

Properties

Prototyped in file “r_usbf_basic_rzn1_if.h”

Description

Prepare USB function port for connection to USB Host

Reentrant

No

21.7.3 R_USB_Close

Format

ER_RET R_USB_Close (void)

Parameters

None

Return Values

ER_RET	Error Value ER_OK on success.

Properties

Prototyped in file “r_usbf_basic_rzn1_if.h”

Description

Close USB function connection

Reentrant

No

21.7.4 R_usb_pstd_TransferStart

Format

USB_ER_t R_usb_pstd_TransferStart(USB_UTR_t *ptr)

Parameters

USB_UTR_t * ptr	Pipe information and details of data to be transferred

Return Values

USB_ER_t	USB_QOVR Transfer start overlap
	USB_ERROR Pipe not configured or invalid
	USB_OK Success

Properties

Prototyped in file “r_usbf_basic_rzn1_if.h”

Description

Initiate data transfer for a Pipe

Called by USBf CDC module function usb_pcdc_DataTrans()

Reentrant

No

21.7.5 R_usb_pstd_TransferEnd

Format

USB_ER_t R_usb_pstd_TransferEnd(uint16_t pipe)

Parameters

uint16_t pipe	USB_PIPE1 to USB_PIPE15

Return Values

USB_ER_t	USB_ERROR Pipe not configured or invalid
	USB_OK Success

Properties

Prototyped in file “r_usbf_basic_rzn1_if.h”

Description

Terminate data transfer on a Pipe

Reentrant

No

21.7.6 R_usb_pstd_ChangeDeviceState

Format

void R_usb_pstd_ChangeDeviceState (uint16_t state, uint16_t keyword, USB_UTR_CB_t complete)

Parameters

uint16_t State	Requested state
uint16_t keyword	Pipe number
USB_CB_t complete	Callback function

Return Values

None

Properties

Prototyped in file “r_usbf_basic_rzn1_if.h”

Description

Change status of USB device

Reentrant

No

21.7.7 R_usb_pstd_DriverRegistration

Format

USB_ER_t R_usb_pstd_DriverRegistration(USB_PCDREG_t *registinfo)

Parameters

USB_PCDREG_t * registinfo	USB_PIPE1 -USB_PIPE15

Registration information - descriptor tables, callback functions.

Return Values

USB_ER_t	USB_OK Success
	ER_INVAL A USB device class app has already been registered for this driver

Properties

Prototyped in file “r_usbf_basic_rzn1_if.h”

Description

Register a USB device Class application with the USB peripheral driver

Reentrant

No

21.7.8 R_usb_pstd_driver_deregister

Format

void R_usb_pstd_ driver_deregister (void)

Parameters

None

Return Values

void

Properties

Prototyped in file “r_usbf_basic_rzn1_if.h”

Description

De-Register a USB device Class application with the USB peripheral driver

Reentrant

No

21.7.9 R_usb_pstd_SetPipeStall

Format

void R_usb_pstd_SetPipeStall (uint16_t pipe)

Parameters

uint16_t pipe	Pipe number

Return Values

None

Properties

Prototyped in file “r_usbf_basic_rzn1_if.h”

Description

Called by USBf CDC module function when an error occurs transferring data on a pipe to stall a pipe.

Reentrant

No

21.7.10 R_usb_pstd_ControlRead

Format

uint16_t R_usb_pstd_ControlRead(uint32_t bsize, uint8_t *table)

Parameters

uint32_t bsize	Read size in bytes
uint8_t * table	Start address of read data buffer

Return Values

uint16_t Return value	Control Read end status

Properties

Prototyped in file “r_usbf_basic_rzn1_if.h”

Description

Called by USBf Basic module, HAL function to read data during a Control Transfer. Start Control Read transfer.

Reentrant

No

21.7.11 R_usb_pstd_ControlWrite

Format

void R_usb_pstd_ControlWrite(uint32_t bsize, uint8_t *table)

Parameters

uint32_t bsize	Write size in bytes
uint8_t * table	Start address of write data buffer

Return Values

None

Properties

Prototyped in file “r_usbf_basic_rzn1_if.h”

Description

Called by USBf Basic module, HAL function to write data during a Control Transfer to Start Control Write transfer.

Reentrant

No

21.7.12 R_usb_pstd_ControlEnd

Format

void R_usb_pstd_ControlEnd(uint16_t status)

Parameters

uint16_t status	Control Transfer status

Return Values

None

Properties

Prototyped in file “r_usbf_basic_rzn1_if.h”

Description

Called by USBf Basic module, HAL function to end a Control Transfer. End Control transfer.

Reentrant

No

21.7.13 R_usb_pstd_poll

Format

void R_usb_pstd_poll(void)

Parameters

None

Return Values

None

Properties

Prototyped in file “r_usbf_basic_rzn1_if.h”

Description

For OS-less system, calls the USBf driver task to check for queued interrupts. Called by user application

Reentrant

No

Example:

```
while(1)
{
   R_usb_pstd_poll();                    /* USB Driver(Peripheral CDC) */

   switch( cdc_dev_info.state )          /* Check application state */
   {
   case STATE_DATA_TRANSFER:             /* Data transfer state */
      cdc_data_transfer();
   break;
   case STATE_ATTACH:                    /* Wait Connect(USB Configured) state */
      cdc_connect_wait();
   break;
   case STATE_DETACH:
      cdc_detach_device();               /* Detach process */
   break;

   default:
   break;
```

```
        }

        /* Check to see if should exit the application */


    }
```

Special Notes:

None

21.8 Communications Device Class Sample Application

The CDC sample application demonstrates communication with a Windows PC using a standard terminal program e.g. Tera Term.

The CDC Sample Application consists of the following files:-

r_usb_pcdc_apl.c	-	Sample application functions to setup the USB peripheral CDC driver and exchange data with a USB Host via a serial comms port.
r_usb_pcdc_descriptor.c	-	USBf CDC descriptor data which defines CDC endpoint configurations

These files are located with the sample Test application see Fig. 4.

The CDC test application is started when the function usbf_main() is called from run_usb_tests() in usb_tests.c (see Fig.1).

- **Steps to run the CDC Sample application.**

The following steps are relevant to running the CDC Sample application on the RXZ/N1d board.

(i) Connect the I-jet probe to J2 (to download and debug).

(ii) Attach jumper to CN6 (ON) for power over USB (CN10).

(iii) Remove jumper from CN2 (OFF) to set USB Device mode.

(iv) Connect a USB cable form CN9 (RZ/N1 USB device) to host pc USB port (USB host).

(v) Connect a USB cable from CN10 to a pc, this will power-on the board.

(vi) Open Tera Term (or HyperTerminal) on the pc and select the serial port to see output from UART Channel 1 of RZ/N1, via CN10

(vii) Build the image and download to the RZ/N1d using IAR IDE and I-jet probe. Run the code.

(viii) The sample application Test menu will appear on the connected Tera Term Window

(ix) Open Device manager on the pc to see connected COM ports

(x) Choose Option ‘A’ to run USB tests

(xi) Choose option ‘2’ to run the CDC Sample application and follow the instructions to connect the USB cable.

(xii) The RZ/N1d USB CDC COM port should be detected by Windows.

(xiii) If prompted, find the Windows device driver for the USB CDC Host side at https://duesvn2.ad.ree.renesas.com/systems/svnSYS_RZN1drv/branches/bare_metal_dev/utilities/USB_CDC_WindowsDriver. Choose 32bit or 64bit depending on Windows version.

(xiv) If not prompted, then install the Windows device driver from the Device Manager window for the newly-detected COM port.

(xv) Now the COM port should be recognized by Windows as ‘CDC USB Demonstration Driver’

(xvi) Open another instance of Ter Term (or HyperTerminal) and connect to the serial port corresponding to CDC USB Demonstration Driver.

(xvii) Select Baud rate 115200, 8-bit dtaa No parity, 1 stop bit no flow control

(xviii) Type some characters into the Tera term window. Initially a message will be displayed ‘Echo mode’ Then characters typed-in should be echoed back from the CDC application running on the RZ/N1 board.

22. USB Host Basic Functions

22.1 Summary

Function	Description
R_USBh_GetVersion	Returns driver version.
R_USBh_Init	Initialise USB Host driver.
R_USBh_Open	Opens the USB Host.
R_USBh_Close	Closes the USB Host.
R_USBh_PollTask	Function that needs to be polled regualry to process USB host events.
R_USB_ConvertUnicodeStrDscrToAsciiStr	Helper function to extract and convert the UTF string from the given string descriptor to a standard ‘C’ string.

22.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

22.3 R_USBh_GetVersion

Format

void R_USBh_GetVersion (void *buf);

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

void

Properties

Prototyped in file “r_usb_rzn1_if.h”

Description

Returns USB host driver version.

Reentrant

No

Example:

```
R_USBh_GetVersion((void*)&host_driver_version);
```

Special Notes:

None

22.4 R_USBh_Init

Format

ER_RET R_USBh_Init(void);

Parameters

None

Return Values

ER_OK	No error
ER_SYS	Problem registering the host interrupt

Properties

Prototyped in file “r_usb_basic_if.h”

Description

Initialise USB Host stack.

Reentrant

No

Example:

```
R_USBh_Init();
```

Special Notes:

None

22.5 R_USBh_Open

Format

ER_RET R_ R_USBh_Open (void);

Parameters

None

Return Values

ER_OK	No error
ER_SYS	Error enabling USB Host interrupt

Properties

Prototyped in file “r_usb_basic_if.h”

Description

Opens the USB Host.

Reentrant

No

Example:

```
ret_status = R_USBh_Open();
```

Special Notes:

None

22.6 R_USBh_Close

Format

ER_RET R_USBh_Close (void);

Parameters

None

Return Values

ER_OK	No error
ER_SYS	Error disabling USB Host interrupt

Properties

Prototyped in file “r_usb_basic_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
ret_status = R_USBh_Close();
```

Special Notes:

None

22.7 R_USBh_PollTask

Format

void R_USBh_PollTask (void);

Parameters

None

Return Values

None

Properties

Prototyped in file “r_usb_basic_if.h”

Description

Called by user application to process host events. Should be called as regular as possible.

Reentrant

No

Example:

```
R_USBh_PollTask();
```

Special Notes:

None

22.8 R_USB_ConvertUnicodeStrDscrToAsciiStr

Format

void R_USB_ConvertUnicodeStrDscrToAsciiStr(const uint8_t * const p_str_dscr, uint8_t * p_ascii_str);

Parameters

p_str_dscr	Pointer to a USB String Descriptor (Descriptr type, length and UTF String)
p_ascii_str	Standard ‘C’ ASCII character buffer

Return Values

None

Properties

Prototyped in file “r_usb_rzn1_if.h”

Description

Produces standard ‘C’ string from given USB string descriptor.

Reentrant

No

Example:

```
/* Copy Manufacturer Id string data */
R_USB_ConvertUnicodeStrDscrToAsciiStr((uint8_t *)mess->tranadr,
                                      (uint8_t *)usb_ghcdc_strings.manufacturer_id);
```

Special Notes:

None

23. USB Host CDC API Functions

23.1 Summary

Function	Description
R_USB_HCDC_Init	Initialise USB Host CDC driver.
R_USB_HCDC_InterfaceTask	Task Function that needs to be polled regualry to process USB host CDC events.
R_USB_HCDC_Write	Write data to an attached CDC device to then be transmitted on the serial communication bus.
R_USB_HCDC_RegisterCallback	Register a call-back to to receive notification of host CDC events
R_USB_HCDC_Control	Control function to configure USB host CDC driver.

23.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

23.3 R_USB_HCDC_Init

Format

void R_USB_HCDC_Init (void);

Parameters

None

Return Values

None

Properties

Prototyped in file “r_usb_hcdc_rzn1_if.h”

Description

Initialise USB Host CDC driver.

Reentrant

No

Example:

```
R_USB_HCDC_Init();
```

Special Notes:

Called from `R_USBh_Open()` subject to `USB_CFG_HCDC_USE` being defined.

23.4 R_USB_HCDC_InterfaceTask

Format

void R_USB_HCDC_InterfaceTask (void);

Parameters

None

Return Values

None

Properties

Prototyped in file “r_usb_hcdc_rzn1_if.h”

Description

Called by USB Host stack to process host CDC events.

Reentrant

No

Example:

```
R_USB_HCDC_InterfaceTask();
```

Special Notes:

Called from `R_USBh_PollTask()` subject to `USB_CFG_HCDC_USE` being defined.

23.5 R_USB_HCDC_Write

Format

ER_RET R_USB_HCDC_Write (uint16_t devadr, const uint8_t * const data, size_t length);

Parameters

devadr	USB device address given to attached device when it was enumerated.
data	Pointer to data to be transmitted
length	Length of data to be transmitted

Return Values

ER_OK	No error
ER_PARAM	Length of data to be transmitted is too long
ER_INVAL	Given device address is not valid
ER_BUSY	Previous write request to same device address has not yet completed

Properties

Prototyped in file “r_usb_hcdc_rzn1_if.h”

Description

Writes data to an attached CDC device.

Reentrant

No

Example:

```
ret_status = R_USB_HCDC_Write(1, pTxMsg, 64);
```

Special Notes:

None

23.6 R_USB_HCDC_RegisterCallback

Format

ER_RET R_USB_HCDC_RegisterCallback (USB_HCDC_CB_t cdc_callback);

Parameters

hub_callback	Callback to allow client knowledge of device attach, detachment, read and write events.

Return Values

ER_OK	No error

Properties

Prototyped in file “r_usb_hcdc_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
ret_val = R_USB_HCDC_RegisterCallback(cdc_host_callback);
if (ER_OK != ret_val)
{
    sample_app_printf("\n\nTEST: USBh TEST6 : USB Host Couldn't Register CDC Callback\n");
    test6_fails++;
}
```

Special Notes:

None

23.7 R_USB_HCDC_Control

Format

ER_RET R_USB_HCDC_Control (uint8_t devadr, USB_HCDC_CONTROL_REQUEST_t control_request, void *buf))

Parameters

devadr	USB device address given to attached device when it was enumerated.
control_request	Request to configure channel or get channel info.
buf	Pointer to configuration data (to set or get)

Return Values

ER_OK	No error
ER_PARAM	Given device address is not in the valid range or an invalid request has been made.
ER_INVAL	Given device address is invalid (not currently in use). Note to set the event callback use of any address within the valid range is permissable.

Properties

Prototyped in file “r_usb_hcdc_rzn1_if.h”

Description

Configure channel parameters, retrieve channel parameters, set the event callback or send a serial ‘break’.

Reentrant

No

Example:

```
line_coding.dwDTERate   = USB_HCDC_SPEED_9600;
line_coding.bDataBits   = USB_HCDC_DATA_BIT_7;
line_coding.bCharFormat = USB_HCDC_STOP_BIT_2;
line_coding.bParityType = USB_HCDC_PARITY_BIT_ODD;

ret_val = R_USB_HCDC_Control(devadr, USB_HCDC_CONTROL_SET_CHAN_CONFIG, &line_coding);
if (ret_val != ER_OK)
{
    sample_app_printf("\n\nTEST: USBh TEST5: USB Host Failed to Set the Line Configuration\n");
    test5_fails++;
}
```

Special Notes:

None

24. USB Host Hub API Functions

24.1 Summary

Function	Description
R_USB_HHUB_Registration	Register class callbacks.
R_USB_HHUB_Task	Task Function that needs to be polled regualry to process USB host Hub events.
R_USB_HHUB_RegisterCallback	Register a call-back to to receive notification of host Hub events.

24.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

24.3 R_USB_HHUB_Registration

Format

void R_USB_HHUB_Registration(USB_HCDREG_t * callback);

Parameters

* callback	Callback to allow client knowledge of hub registration event.

Return Values

None

Properties

Prototyped in file “r_usb_hhub_rzn1_if.h”

Description

Register bespoke callback for Hub class.

Reentrant

No

Example:

```
R_USB_HHUB_Registration(USB_NULL);
```

Special Notes:

Called from R_USBh_Init.

24.4 R_USB_HHUB_Task

Format

void R_USB_HHUB_Task (void);

Parameters

None

Return Values

None

Properties

Prototyped in file “r_usb_hhub_rzn1_if.h”

Description

Called by USB Host stack to process host Hub events.

Reentrant

No

Example:

```
R_USB_HHUB_Task();
```

Special Notes:

Called from `R_USBh_PollTask()`.

24.5 R_USB_HHUB_RegisterCallback

Format

ER_RET R_USB_HHUB_RegisterCallback(USB_HHUB_CB_t hub_callback);

Parameters

hub_callback	Callback to allow client knowledge of hub/device attach and detachment events.

Return Values

ER_OK	No error

Properties

Prototyped in file “r_usb_hhub_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
ret_val = R_USB_HHUB_RegisterCallback(hub_host_callback);
if (ER_OK != ret_val)
{
    sample_app_printf("\n\nTEST: USBh TEST6 : USB Host Couldn't Register Hub Callback\n");
    test6_fails++;
}
```

Special Notes:

None

25. WDT API Functions

25.1 Summary

Function	Description
R_WDT_GetVersion	Returns driver version.
R_WDT_Open	Initialises the WDT.
R_WDT_Control	Controls a DMA channel • Sets/gets prescaler configuration • Sets/gets reload configuration • Sets/gets WDT output configuration • Sets callback function
R_WDT_Start	Start the WDT countdown.
R_WDT_Kick	Refreshes the countdown value
R_WDT_Close	Uninitialises the WDT.

25.2 Return Values

API function returns with specified values. These values are defined by macro found in errcodes.h, which the values and definitions are shown below.

```
/*!< Function return type will return the error code */
typedef int ER_RET;

/* define of error code */
#define ER_OK         ((ER_RET)0)     /* Normal end (no error) */
#define ER_NG         ((ER_RET)-1)    /* Abnormal end (system call failed)) */
#define ER_SYS        ((ER_RET)-2)    /* Undefined error */
#define ER_PARAM      ((ER_RET)-3)    /* Invalid parameter */
#define ER_NOMEM      ((ER_RET)-4)    /* Out of memory         */
#define ER_BUSY       ((ER_RET)-5)    /* Channel in use (unavailable) */
#define ER_INVAL      ((ER_RET)-6)    /* Invalid state */
#define ER_TIMEOUT    ((ER_RET)-7)    /* Timeout occurs */
```

25.3 R_WDT_GetVersion

Format

void R_WDT_GetVersion(void *buf)

Parameters

buf	A pointer to a variable where the version number is stored.

Return Values

void

Properties

Prototyped in file “r_wdt_rzn1_if.h”

Description

Returns driver version.

Reentrant

No

Example:

```
R_WDT_GetVersion ((void*) &wdt_driver_version);
```

Special Notes:

None

25.4 R_WDT_Open

Format

ER_RET R_WDT_Open (void);

Parameters

None

Return Values

ER_OK	No error
ER_BUSY	Channel in use (unavailable)

Properties

Prototyped in file “r_wdt_rzn1_if.h”.

Description

Initialises the WDT.

Reentrant

No

Example:

```
R_WDT_Open();
```

Special Notes:

None

25.5 R_WDT_Control

Format

ER_RET R_WDT_Control (WDT_CONTROL_REQUEST control_request, void *control);

Parameters

control_request	Which control action to implement.
control	The control parameter that is used along with the control action to configure the controller instance a certain way.

Return Values

ER_OK	No error
ER_INVAL	Invalid state
ER_PARAM	Invalid parameter

Properties

Prototyped in file “r_wdt_rzn1_if.h”

Description

Controls a WDT instance.

Reentrant

No

Example:

```
WDT_PRESCALER prescale = WDT_CLOCK_DIVIDE_2EXP14;
ret_val = R_WDT_Control(WDT_SET_PRESCALER, &prescale);

WDT_OUTPUT output = WDT_SYSTEM_RESET;
ret_val = R_WDT_Control(WDT_SET_OUTPUT, &output);
```

Special Notes:

None

25.6 R_WDT_Start

Format

ER_RET R_WDT_Start (void);

Parameters

None

Return Values

ER_OK	No error
ER_INVAL	Invalid state

Properties

Prototyped in file “r_wdt_rzn1_if.h”.

Description

Start the WDT countdown.

Reentrant

No

Example:

```
R_WDT_Start();
```

Special Notes:

None

25.7 R_WDT_Kick

Format

ER_RET R_WDT_Kick (void);

Parameters

None

Return Values

ER_OK	No error
ER_INVAL	Invalid state

Properties

Prototyped in file “r_wdt_rzn1_if.h”.

Description

Refreshes the countdown value.

Reentrant

No

Example:

```
R_WDT_Kick();
```

Special Notes:

None

25.8 R_WDT_Close

Format

ER_RET R_WDT_Close (void);

Parameters

None

Return Values

ER_OK	No error
ER_INVAL	Invalid state

Properties

Prototyped in file “r_wdt_rzn1_if.h”.

Description

Uninitialises the WDT.

Reentrant

No

Example:

```
R_WDT_Close();
```

Special Notes:

None

26. Demo Projects

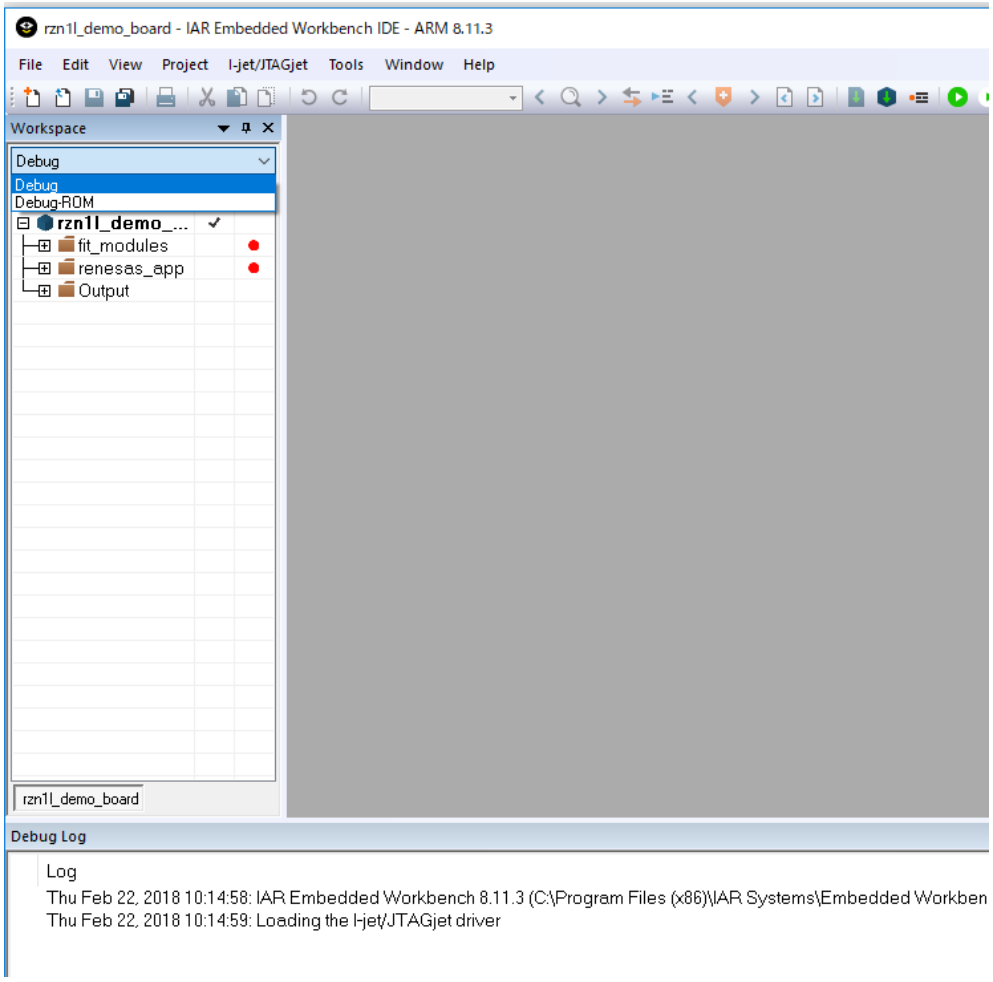
This chapter describes how to demonstrate application sample.

Procedure to Build the Application

There are two ways to run program. The one way lets a program run in RAM, and the other way lets a program run in ROM (e.g. SPI flash ROM).

Program runs in RAM case:

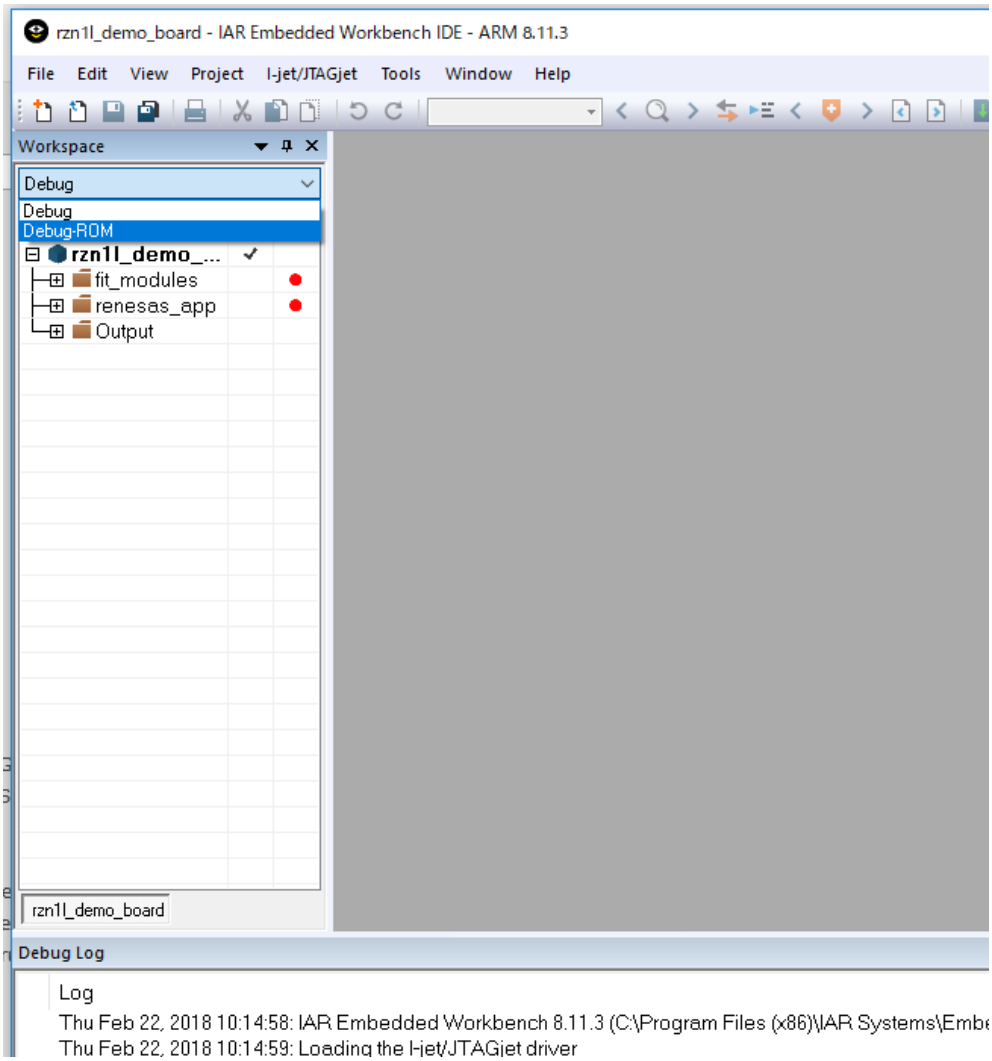
1. Start any EWARM workspace and choose build configuration ‘Debug’ which is on top left as pull down list - See following figure.
   Example workspace: *BaremetalDriver\renesas_app\led_blink\iar\rzn1l_demo_board\rzn1l_demo_board.eww*



2. Start build
3. Download the program into RAM, and start debugging

Program runs in ROM case:

1. Start any EWARM workspace and choose build configuration ‘Debug-ROM’ which is on top left as pull down list - See following figure. This build configuration already has a flash loader setting.
   Example workspace: *BaremetalDriver\renesas_app\led_blink\iar\rzn1l_demo_board\rzn1l_demo_board.eww*



2. Start build
3. Download the program into ROM, and start debugging

26.1 led_blink

This sample application demonstrates terminal I/O operation using UART, and LED control and monitoring dip switch on boards using I2C. I2C initialization and pin settings for LED control have been done in hardware_setup(), before main(). hardware_setup() is defined in C source file hwsetup.c.

Procedure to Run the Application

1. Connect a PC to UART connector of RZ/N1x-DB (x=D, S, or L) board.
2. Power up the board.
3. Open terminal emulator (e.g. putty, teraterm), and open property COM port. The line ending code of terminal emulator should be set ‘CR’.
4. Run program, and check if application works fine.

Result Log

If the program works fine, initial messages as shown below are displayed on your terminal window.

```
*************** Sample Application Starts ***************

Enter a key to test
 0-7: Target LED is toggled
   i: Print Dip Switch settings
   b: Change UART baud-rate
   x: Exit
   ?: Help
```

After displayed these messages, put any key to check if application works correctly.

27. Known Limitation and Restrictions

- USBf does not run on RZ/N1L – cable connection to Host is not detected (a hardware modification is required for the RZ/N1L board).
- The Timer driver currently supports only Timer Block1.
- Generic interrupt controller function R_BSP_InterruptControl should disable interrupts on entry and re-enable on exit as it is re-entrant.
- RZ/N1L has the ability to run a USB stack but as there is no expansion board for it there is no TypeA host port to plug a device in with. For this reason the host has not been tested on this platform and the USB host tests have been removed from it's project. However, the driver and API code has been left in the case that a customer wants to experitment with their own hardware.

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/contact/

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
0.01	Feb 28, 2018	—	First edition issued
0.02	Oct 01, 2018	---	Updated
0.03	Oct 20, 2018	—	Core Initialisation updated
0.04	Oct 25, 2018	—	Corrected USBF API prototypes
0.05	Oct 31, 2018	—	Added drivers for SPI, QSPI, SDIO, WDT, Timer, DMA, MSEBI, and USB CDC Function.
		—	Adapted to RZ/N1D and RZ/N1S, with Cortex-A7.
		—	Upgraded EWARM version to 8.22.1.
		—	Added new chapter for Known Limitation and Restriction.
0.06	Dec 18, 2018	---	Updated with new peripheral drivers
0.07	Feb 15, 2019	---	Updated with new peripheral drivers,
0.08	Feb 19, 2019	--	LCDC added. IOMUX API updated
0.09	March 4, 2019	--	USB Host Basic, CDC & Hub added, USB Host OHCI removed
0.10	March 27, 2019	--	Minor corrections and updates.
0.11	April 10, 2019	--	Minor updates
0.12	April 15, 2019	--	Update footers so are consistent. ‘Extension’ board corrected to ‘Expansion’ board
0.13	July 19, 2019	--	Semaphore driver added
0.14	August 1, 2019	151	Documented new GMAC Read function

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other disputes involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawing, chart, program, algorithm, application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics products.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (space and undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. When using the Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat radiation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions or failure or accident arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please ensure to implement safety measures to guard them against the possibility of bodily injury, injury or damage caused by fire, and social damage in the event of failure or malfunction of Renesas Electronics products, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures by your own responsibility as warranty for your products/system. Because the evaluation of microcomputer software alone is very difficult and not practical, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please investigate applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive carefully and sufficiently and use Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall not use Renesas Electronics products or technologies for (1) any purpose relating to the development, design, manufacture, use, stockpiling, etc., of weapons of mass destruction, such as nuclear weapons, chemical weapons, or biological weapons, or missiles (including unmanned aerial vehicles (UAVs)) for delivering such weapons, (2) any purpose relating to the development, design, manufacture, or use of conventional weapons, or (3) any other purpose of disturbing international peace and security, and you shall not sell, export, lease, transfer, or release Renesas Electronics products or technologies to any third party whether directly or indirectly with knowledge or reason to know that the third party or any other party will engage in the activities described above. When exporting, selling, transferring, etc., Renesas Electronics products or technologies, you shall comply with any applicable export control laws and regulations promulgated and administered by the governments of the countries asserting jurisdiction over the parties or transactions.
10. Please acknowledge and agree that you shall bear all the losses and damages which are incurred from the misuse or violation of the terms and conditions described in this document, including this notice, and hold Renesas Electronics harmless, if such misuse or violation results from your resale or making Renesas Electronics products available any third party.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.3.0-1 November 2016)

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 6.0