K210
FreeRTOS SDK
Programming guide

KENDRYTE

Canaan Inc. ©2019

KENDRYTE.COM

About This Guide

This document provides the programming guide of Kendryte K210 FreeRTOS SDK.

Corresponding SDK version

Kendryte FreeRTOS SDK v0.4.0 (*9c7b0e0d23e46e87a2bfd4dd86d1a1f0d3c899e9*)

Revision History

Date	Ver.	Revision History
2018-10-12	V0.1.0	Initial release

Disclaimer

Information in this document, including URL references, is subject to change without notice. THIS DOCUMENT IS PROVIDED AS IS WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Copyright Notice

Copyright © 2018 Canaan Inc. All rights reserved.

Contents

Chapter 1

FreeRTOS Extension

1.1 Overview

FreeRTOS is a lightweight real-time operating system. This SDK adds some features for the K210.

1.2 Features

The FreeRTOS extension has the following features:

- Get the logical processor Id where the current task is located
- Create tasks on the specified logical processor

The K210 contains 2 logical processors with IDs of 0 and 1 respectively.

1.3 API

Corresponding header file task.h
Provide the following interfaces

- uxTaskGetProcessorId
- xTaskCreateAtProcessor

1.3.1 uxTaskGetProcessorId

1.3.1.1 Description

Get the current logical processor ID.

1.3.1.2 Function prototype

```
UBaseType_t uxTaskGetProcessorId(void);
```

1.3.1.3 Return value

Current logical processor ID.

1.3.2 xTaskCreateAtProcessor

1.3.2.1 Description

Create a task at the specified logical processor.

1.3.2.2 Function prototype

```
BaseType_t xTaskCreateAtProcessor(UBaseType_t uxProcessor, TaskFunction_t pxTaskCode,
    const char * const pcName, const configSTACK_DEPTH_TYPE usStackDepth, void * const
     pvParameters, UBaseType_t uxPriority, TaskHandle_t * const pxCreatedTask);
```

1.3.2.3 Parameter

Parameter name	Description	Input or output
uxProcessor	Logical processor ID	Input
pxTaskCode	Task entry point	Input
pcName	Task name	Input
usStackDepth	Stack depth	Input
pvParameters	Parameters	Input
uxPriority	Priority	Input
pxCreatedTask	Created task handle	Input

1.3.2.4 Return value

Return value	Description
pdPASS	Success
Others	Fail

Chapter 2

Device List

Path	Type	Description
/dev/uart1	UART	
/dev/uart2	UART	
/dev/uart3	UART	
/dev/gpio0	GPIO	High-speed(fast) GPIO
/dev/gpio1	GPIO	
/dev/i2c0	I²C	
/dev/i2c1	I²C	
/dev/i2c2	I²C	
/dev/i2s0	I2S	
/dev/i2s1	I2S	
/dev/i2s2	I2S	
/dev/spi0	SPI	
/dev/spi1	SPI	
/dev/spi3	SPI	
/dev/sccb0	SCCB	
/dev/dvp0	DVP	
/dev/fft0	FFT	
/dev/aes0	AES	
/dev/sha256	SHA256	
/dev/timer0	TIMER	Cannot be used with /dev/pwm0
/dev/timer1	TIMER	Cannot be used with /dev/pwm0
/dev/timer2	TIMER	Cannot be used with /dev/pwm0

Path	Type	Description
/dev/timer3	TIMER	Cannot be used with /dev/pwm0
/dev/timer4	TIMER	Cannot be used with /dev/pwm1
/dev/timer5	TIMER	Cannot be used with /dev/pwm1
/dev/timer6	TIMER	Cannot be used with /dev/pwm1
/dev/timer7	TIMER	Cannot be used with /dev/pwm1
/dev/timer8	TIMER	Cannot be used with /dev/pwm2
/dev/timer9	TIMER	Cannot be used with /dev/pwm2
/dev/timer10	TIMER	Cannot be used with /dev/pwm2
/dev/timer11	TIMER	Cannot be used with /dev/pwm2
/dev/pwm0	PWM	Cannot be used with /dev/timer[0-3]
/dev/pwm1	PWM	Cannot be used with /dev/timer[4-7]
/dev/pwm2	PWM	Cannot be used with /dev/timer[8-11]
/dev/wdt0	WDT	
/dev/wdt1	WDT	
/dev/rtc0	RTC	

Chapter 3

I/O configuration

3.1 Overview

The I/O configuration includes FPIOA and power domain configuration.

FPIOA (Field Programmable I/O Array) allows the user to map 256 internal functions to 48 free I/Os on the chip.

3.2 Features

- Support for I/O's programmable function selection
- 8 driving capability options for I/O output support
- Support I/O internal pull-up resistor selection
- Support I/O internal pull-down resistor selection
- Internal Schmitt trigger settings that support I/O input
- Slew control for I/O output support
- Support level setting of internal input logic
- Support for configuring power domains

3.3 Data type

Corresponding header file `pin_cfg.h`

The relevant data types and data structures are defined as follows:

- fpioa_function_t: The function number of the pin.
- fpioa_cfg_item_t: FPIOA pin configuration.
- fpioa_cfg_t: FPIOA configuration.

- sysctl_power_bank_t: Power domain number.
- sysctl_io_power_mode_t: IO output voltage selection.
- power_bank_item_t: Single power domain configuration.
- power_bank_cfg_t: Power domain configuration.
- pin_cfg_t: Pin configuration.

3.3.1 fpioa_function_t

3.3.1.1 Description

The function number of the pin.

3.3.1.2 Type definition

```
typedef enum _fpioa_function
{
    FUNC_JTAG_TCLK          = 0,      /*!< JTAG Test Clock */
    FUNC_JTAG_TDI           = 1,      /*!< JTAG Test Data In */
    FUNC_JTAG_TMS           = 2,      /*!< JTAG Test Mode Select */
    FUNC_JTAG_TDO           = 3,      /*!< JTAG Test Data Out */
    FUNC_SPI0_D0            = 4,      /*!< SPI0 Data 0 */
    FUNC_SPI0_D1            = 5,      /*!< SPI0 Data 1 */
    FUNC_SPI0_D2            = 6,      /*!< SPI0 Data 2 */
    FUNC_SPI0_D3            = 7,      /*!< SPI0 Data 3 */
    FUNC_SPI0_D4            = 8,      /*!< SPI0 Data 4 */
    FUNC_SPI0_D5            = 9,      /*!< SPI0 Data 5 */
    FUNC_SPI0_D6            = 10,     /*!< SPI0 Data 6 */
    FUNC_SPI0_D7            = 11,     /*!< SPI0 Data 7 */
    FUNC_SPI0_SS0           = 12,     /*!< SPI0 Chip Select 0 */
    FUNC_SPI0_SS1           = 13,     /*!< SPI0 Chip Select 1 */
    FUNC_SPI0_SS2           = 14,     /*!< SPI0 Chip Select 2 */
    FUNC_SPI0_SS3           = 15,     /*!< SPI0 Chip Select 3 */
    FUNC_SPI0_ARB           = 16,     /*!< SPI0 Arbitration */
    FUNC_SPI0_SCLK          = 17,     /*!< SPI0 Serial Clock */
    FUNC_UARTHS_RX          = 18,     /*!< UART High speed Receiver */
    FUNC_UARTHS_TX          = 19,     /*!< UART High speed Transmitter */
    FUNC_CLK_IN1            = 20,     /*!< Clock Input 1 */
    FUNC_CLK_IN2            = 21,     /*!< Clock Input 2 */
    FUNC_CLK_SPI1           = 22,     /*!< Clock SPI1 */
    FUNC_CLK_I2C1           = 23,     /*!< Clock I2C1 */
    FUNC_GPIOHS0            = 24,     /*!< GPIO High speed 0 */
    FUNC_GPIOHS1            = 25,     /*!< GPIO High speed 1 */
    FUNC_GPIOHS2            = 26,     /*!< GPIO High speed 2 */
    FUNC_GPIOHS3            = 27,     /*!< GPIO High speed 3 */
    FUNC_GPIOHS4            = 28,     /*!< GPIO High speed 4 */
    FUNC_GPIOHS5            = 29,     /*!< GPIO High speed 5 */
    FUNC_GPIOHS6            = 30,     /*!< GPIO High speed 6 */
    FUNC_GPIOHS7            = 31,     /*!< GPIO High speed 7 */
```

```
    FUNC_GPIOHS8          = 32,     /*!< GPIO High speed 8 */
    FUNC_GPIOHS9          = 33,     /*!< GPIO High speed 9 */
    FUNC_GPIOHS10         = 34,     /*!< GPIO High speed 10 */
    FUNC_GPIOHS11         = 35,     /*!< GPIO High speed 11 */
    FUNC_GPIOHS12         = 36,     /*!< GPIO High speed 12 */
    FUNC_GPIOHS13         = 37,     /*!< GPIO High speed 13 */
    FUNC_GPIOHS14         = 38,     /*!< GPIO High speed 14 */
    FUNC_GPIOHS15         = 39,     /*!< GPIO High speed 15 */
    FUNC_GPIOHS16         = 40,     /*!< GPIO High speed 16 */
    FUNC_GPIOHS17         = 41,     /*!< GPIO High speed 17 */
    FUNC_GPIOHS18         = 42,     /*!< GPIO High speed 18 */
    FUNC_GPIOHS19         = 43,     /*!< GPIO High speed 19 */
    FUNC_GPIOHS20         = 44,     /*!< GPIO High speed 20 */
    FUNC_GPIOHS21         = 45,     /*!< GPIO High speed 21 */
    FUNC_GPIOHS22         = 46,     /*!< GPIO High speed 22 */
    FUNC_GPIOHS23         = 47,     /*!< GPIO High speed 23 */
    FUNC_GPIOHS24         = 48,     /*!< GPIO High speed 24 */
    FUNC_GPIOHS25         = 49,     /*!< GPIO High speed 25 */
    FUNC_GPIOHS26         = 50,     /*!< GPIO High speed 26 */
    FUNC_GPIOHS27         = 51,     /*!< GPIO High speed 27 */
    FUNC_GPIOHS28         = 52,     /*!< GPIO High speed 28 */
    FUNC_GPIOHS29         = 53,     /*!< GPIO High speed 29 */
    FUNC_GPIOHS30         = 54,     /*!< GPIO High speed 30 */
    FUNC_GPIOHS31         = 55,     /*!< GPIO High speed 31 */
    FUNC_GPIO0            = 56,     /*!< GPIO pin 0 */
    FUNC_GPIO1            = 57,     /*!< GPIO pin 1 */
    FUNC_GPIO2            = 58,     /*!< GPIO pin 2 */
    FUNC_GPIO3            = 59,     /*!< GPIO pin 3 */
    FUNC_GPIO4            = 60,     /*!< GPIO pin 4 */
    FUNC_GPIO5            = 61,     /*!< GPIO pin 5 */
    FUNC_GPIO6            = 62,     /*!< GPIO pin 6 */
    FUNC_GPIO7            = 63,     /*!< GPIO pin 7 */
    FUNC_UART1_RX         = 64,     /*!< UART1 Receiver */
    FUNC_UART1_TX         = 65,     /*!< UART1 Transmitter */
    FUNC_UART2_RX         = 66,     /*!< UART2 Receiver */
    FUNC_UART2_TX         = 67,     /*!< UART2 Transmitter */
    FUNC_UART3_RX         = 68,     /*!< UART3 Receiver */
    FUNC_UART3_TX         = 69,     /*!< UART3 Transmitter */
    FUNC_SPI1_D0          = 70,     /*!< SPI1 Data 0 */
    FUNC_SPI1_D1          = 71,     /*!< SPI1 Data 1 */
    FUNC_SPI1_D2          = 72,     /*!< SPI1 Data 2 */
    FUNC_SPI1_D3          = 73,     /*!< SPI1 Data 3 */
    FUNC_SPI1_D4          = 74,     /*!< SPI1 Data 4 */
    FUNC_SPI1_D5          = 75,     /*!< SPI1 Data 5 */
    FUNC_SPI1_D6          = 76,     /*!< SPI1 Data 6 */
    FUNC_SPI1_D7          = 77,     /*!< SPI1 Data 7 */
    FUNC_SPI1_SS0         = 78,     /*!< SPI1 Chip Select 0 */
    FUNC_SPI1_SS1         = 79,     /*!< SPI1 Chip Select 1 */
    FUNC_SPI1_SS2         = 80,     /*!< SPI1 Chip Select 2 */
    FUNC_SPI1_SS3         = 81,     /*!< SPI1 Chip Select 3 */
    FUNC_SPI1_ARB         = 82,     /*!< SPI1 Arbitration */
    FUNC_SPI1_SCLK        = 83,     /*!< SPI1 Serial Clock */
    FUNC_SPI_SLAVE_D0     = 84,     /*!< SPI Slave Data 0 */
```

```
    FUNC_SPI_SLAVE_SS     = 85,     /*!< SPI Slave Select */
    FUNC_SPI_SLAVE_SCLK   = 86,     /*!< SPI Slave Serial Clock */
    FUNC_I2S0_MCLK        = 87,     /*!< I2S0 Master Clock */
    FUNC_I2S0_SCLK        = 88,     /*!< I2S0 Serial Clock(BCLK) */
    FUNC_I2S0_WS          = 89,     /*!< I2S0 Word Select(LRCLK) */
    FUNC_I2S0_IN_D0       = 90,     /*!< I2S0 Serial Data Input 0 */
    FUNC_I2S0_IN_D1       = 91,     /*!< I2S0 Serial Data Input 1 */
    FUNC_I2S0_IN_D2       = 92,     /*!< I2S0 Serial Data Input 2 */
    FUNC_I2S0_IN_D3       = 93,     /*!< I2S0 Serial Data Input 3 */
    FUNC_I2S0_OUT_D0      = 94,     /*!< I2S0 Serial Data Output 0 */
    FUNC_I2S0_OUT_D1      = 95,     /*!< I2S0 Serial Data Output 1 */
    FUNC_I2S0_OUT_D2      = 96,     /*!< I2S0 Serial Data Output 2 */
    FUNC_I2S0_OUT_D3      = 97,     /*!< I2S0 Serial Data Output 3 */
    FUNC_I2S1_MCLK        = 98,     /*!< I2S1 Master Clock */
    FUNC_I2S1_SCLK        = 99,     /*!< I2S1 Serial Clock(BCLK) */
    FUNC_I2S1_WS          = 100,    /*!< I2S1 Word Select(LRCLK) */
    FUNC_I2S1_IN_D0       = 101,    /*!< I2S1 Serial Data Input 0 */
    FUNC_I2S1_IN_D1       = 102,    /*!< I2S1 Serial Data Input 1 */
    FUNC_I2S1_IN_D2       = 103,    /*!< I2S1 Serial Data Input 2 */
    FUNC_I2S1_IN_D3       = 104,    /*!< I2S1 Serial Data Input 3 */
    FUNC_I2S1_OUT_D0      = 105,    /*!< I2S1 Serial Data Output 0 */
    FUNC_I2S1_OUT_D1      = 106,    /*!< I2S1 Serial Data Output 1 */
    FUNC_I2S1_OUT_D2      = 107,    /*!< I2S1 Serial Data Output 2 */
    FUNC_I2S1_OUT_D3      = 108,    /*!< I2S1 Serial Data Output 3 */
    FUNC_I2S2_MCLK        = 109,    /*!< I2S2 Master Clock */
    FUNC_I2S2_SCLK        = 110,    /*!< I2S2 Serial Clock(BCLK) */
    FUNC_I2S2_WS          = 111,    /*!< I2S2 Word Select(LRCLK) */
    FUNC_I2S2_IN_D0       = 112,    /*!< I2S2 Serial Data Input 0 */
    FUNC_I2S2_IN_D1       = 113,    /*!< I2S2 Serial Data Input 1 */
    FUNC_I2S2_IN_D2       = 114,    /*!< I2S2 Serial Data Input 2 */
    FUNC_I2S2_IN_D3       = 115,    /*!< I2S2 Serial Data Input 3 */
    FUNC_I2S2_OUT_D0      = 116,    /*!< I2S2 Serial Data Output 0 */
    FUNC_I2S2_OUT_D1      = 117,    /*!< I2S2 Serial Data Output 1 */
    FUNC_I2S2_OUT_D2      = 118,    /*!< I2S2 Serial Data Output 2 */
    FUNC_I2S2_OUT_D3      = 119,    /*!< I2S2 Serial Data Output 3 */
    FUNC_RESV0            = 120,    /*!< Reserved function */
    FUNC_RESV1            = 121,    /*!< Reserved function */
    FUNC_RESV2            = 122,    /*!< Reserved function */
    FUNC_RESV3            = 123,    /*!< Reserved function */
    FUNC_RESV4            = 124,    /*!< Reserved function */
    FUNC_RESV5            = 125,    /*!< Reserved function */
    FUNC_I2C0_SCLK        = 126,    /*!< I2C0 Serial Clock */
    FUNC_I2C0_SDA         = 127,    /*!< I2C0 Serial Data */
    FUNC_I2C1_SCLK        = 128,    /*!< I2C1 Serial Clock */
    FUNC_I2C1_SDA         = 129,    /*!< I2C1 Serial Data */
    FUNC_I2C2_SCLK        = 130,    /*!< I2C2 Serial Clock */
    FUNC_I2C2_SDA         = 131,    /*!< I2C2 Serial Data */
    FUNC_CMOS_XCLK        = 132,    /*!< DVP System Clock */
    FUNC_CMOS_RST         = 133,    /*!< DVP System Reset */
    FUNC_CMOS_PWND        = 134,    /*!< DVP Power Down Mode */
    FUNC_CMOS_VSYNC       = 135,    /*!< DVP Vertical Sync */
    FUNC_CMOS_HREF        = 136,    /*!< DVP Horizontal Reference output */
    FUNC_CMOS_PCLK        = 137,    /*!< Pixel Clock */
```

```
    FUNC_CMOS_D0          = 138,    /*!< Data Bit 0 */
    FUNC_CMOS_D1          = 139,    /*!< Data Bit 1 */
    FUNC_CMOS_D2          = 140,    /*!< Data Bit 2 */
    FUNC_CMOS_D3          = 141,    /*!< Data Bit 3 */
    FUNC_CMOS_D4          = 142,    /*!< Data Bit 4 */
    FUNC_CMOS_D5          = 143,    /*!< Data Bit 5 */
    FUNC_CMOS_D6          = 144,    /*!< Data Bit 6 */
    FUNC_CMOS_D7          = 145,    /*!< Data Bit 7 */
    FUNC_SCCB_SCLK        = 146,    /*!< SCCB Serial Clock */
    FUNC_SCCB_SDA         = 147,    /*!< SCCB Serial Data */
    FUNC_UART1_CTS        = 148,    /*!< UART1 Clear To Send */
    FUNC_UART1_DSR        = 149,    /*!< UART1 Data Set Ready */
    FUNC_UART1_DCD        = 150,    /*!< UART1 Data Carrier Detect */
    FUNC_UART1_RI         = 151,    /*!< UART1 Ring Indicator */
    FUNC_UART1_SIR_IN     = 152,    /*!< UART1 Serial Infrared Input */
    FUNC_UART1_DTR        = 153,    /*!< UART1 Data Terminal Ready */
    FUNC_UART1_RTS        = 154,    /*!< UART1 Request To Send */
    FUNC_UART1_OUT2       = 155,    /*!< UART1 User-designated Output 2 */
    FUNC_UART1_OUT1       = 156,    /*!< UART1 User-designated Output 1 */
    FUNC_UART1_SIR_OUT    = 157,    /*!< UART1 Serial Infrared Output */
    FUNC_UART1_BAUD       = 158,    /*!< UART1 Transmit Clock Output */
    FUNC_UART1_RE         = 159,    /*!< UART1 Receiver Output Enable */
    FUNC_UART1_DE         = 160,    /*!< UART1 Driver Output Enable */
    FUNC_UART1_RS485_EN   = 161,    /*!< UART1 RS485 Enable */
    FUNC_UART2_CTS        = 162,    /*!< UART2 Clear To Send */
    FUNC_UART2_DSR        = 163,    /*!< UART2 Data Set Ready */
    FUNC_UART2_DCD        = 164,    /*!< UART2 Data Carrier Detect */
    FUNC_UART2_RI         = 165,    /*!< UART2 Ring Indicator */
    FUNC_UART2_SIR_IN     = 166,    /*!< UART2 Serial Infrared Input */
    FUNC_UART2_DTR        = 167,    /*!< UART2 Data Terminal Ready */
    FUNC_UART2_RTS        = 168,    /*!< UART2 Request To Send */
    FUNC_UART2_OUT2       = 169,    /*!< UART2 User-designated Output 2 */
    FUNC_UART2_OUT1       = 170,    /*!< UART2 User-designated Output 1 */
    FUNC_UART2_SIR_OUT    = 171,    /*!< UART2 Serial Infrared Output */
    FUNC_UART2_BAUD       = 172,    /*!< UART2 Transmit Clock Output */
    FUNC_UART2_RE         = 173,    /*!< UART2 Receiver Output Enable */
    FUNC_UART2_DE         = 174,    /*!< UART2 Driver Output Enable */
    FUNC_UART2_RS485_EN   = 175,    /*!< UART2 RS485 Enable */
    FUNC_UART3_CTS        = 176,    /*!< UART3 Clear To Send */
    FUNC_UART3_DSR        = 177,    /*!< UART3 Data Set Ready */
    FUNC_UART3_DCD        = 178,    /*!< UART3 Data Carrier Detect */
    FUNC_UART3_RI         = 179,    /*!< UART3 Ring Indicator */
    FUNC_UART3_SIR_IN     = 180,    /*!< UART3 Serial Infrared Input */
    FUNC_UART3_DTR        = 181,    /*!< UART3 Data Terminal Ready */
    FUNC_UART3_RTS        = 182,    /*!< UART3 Request To Send */
    FUNC_UART3_OUT2       = 183,    /*!< UART3 User-designated Output 2 */
    FUNC_UART3_OUT1       = 184,    /*!< UART3 User-designated Output 1 */
    FUNC_UART3_SIR_OUT    = 185,    /*!< UART3 Serial Infrared Output */
    FUNC_UART3_BAUD       = 186,    /*!< UART3 Transmit Clock Output */
    FUNC_UART3_RE         = 187,    /*!< UART3 Receiver Output Enable */
    FUNC_UART3_DE         = 188,    /*!< UART3 Driver Output Enable */
    FUNC_UART3_RS485_EN   = 189,    /*!< UART3 RS485 Enable */
    FUNC_TIMER0_TOGGLE1   = 190,    /*!< TIMER0 Toggle Output 1 */
```

```
    FUNC_TIMER0_TOGGLE2   = 191,    /*!< TIMER0 Toggle Output 2 */
    FUNC_TIMER0_TOGGLE3   = 192,    /*!< TIMER0 Toggle Output 3 */
    FUNC_TIMER0_TOGGLE4   = 193,    /*!< TIMER0 Toggle Output 4 */
    FUNC_TIMER1_TOGGLE1   = 194,    /*!< TIMER1 Toggle Output 1 */
    FUNC_TIMER1_TOGGLE2   = 195,    /*!< TIMER1 Toggle Output 2 */
    FUNC_TIMER1_TOGGLE3   = 196,    /*!< TIMER1 Toggle Output 3 */
    FUNC_TIMER1_TOGGLE4   = 197,    /*!< TIMER1 Toggle Output 4 */
    FUNC_TIMER2_TOGGLE1   = 198,    /*!< TIMER2 Toggle Output 1 */
    FUNC_TIMER2_TOGGLE2   = 199,    /*!< TIMER2 Toggle Output 2 */
    FUNC_TIMER2_TOGGLE3   = 200,    /*!< TIMER2 Toggle Output 3 */
    FUNC_TIMER2_TOGGLE4   = 201,    /*!< TIMER2 Toggle Output 4 */
    FUNC_CLK_SPI2         = 202,    /*!< Clock SPI2 */
    FUNC_CLK_I2C2         = 203,    /*!< Clock I2C2 */
    FUNC_INTERNAL0        = 204,    /*!< Internal function signal 0 */
    FUNC_INTERNAL1        = 205,    /*!< Internal function signal 1 */
    FUNC_INTERNAL2        = 206,    /*!< Internal function signal 2 */
    FUNC_INTERNAL3        = 207,    /*!< Internal function signal 3 */
    FUNC_INTERNAL4        = 208,    /*!< Internal function signal 4 */
    FUNC_INTERNAL5        = 209,    /*!< Internal function signal 5 */
    FUNC_INTERNAL6        = 210,    /*!< Internal function signal 6 */
    FUNC_INTERNAL7        = 211,    /*!< Internal function signal 7 */
    FUNC_INTERNAL8        = 212,    /*!< Internal function signal 8 */
    FUNC_INTERNAL9        = 213,    /*!< Internal function signal 9 */
    FUNC_INTERNAL10       = 214,    /*!< Internal function signal 10 */
    FUNC_INTERNAL11       = 215,    /*!< Internal function signal 11 */
    FUNC_INTERNAL12       = 216,    /*!< Internal function signal 12 */
    FUNC_INTERNAL13       = 217,    /*!< Internal function signal 13 */
    FUNC_INTERNAL14       = 218,    /*!< Internal function signal 14 */
    FUNC_INTERNAL15       = 219,    /*!< Internal function signal 15 */
    FUNC_INTERNAL16       = 220,    /*!< Internal function signal 16 */
    FUNC_INTERNAL17       = 221,    /*!< Internal function signal 17 */
    FUNC_CONSTANT         = 222,    /*!< Constant function */
    FUNC_INTERNAL18       = 223,    /*!< Internal function signal 18 */
    FUNC_DEBUG0           = 224,    /*!< Debug function 0 */
    FUNC_DEBUG1           = 225,    /*!< Debug function 1 */
    FUNC_DEBUG2           = 226,    /*!< Debug function 2 */
    FUNC_DEBUG3           = 227,    /*!< Debug function 3 */
    FUNC_DEBUG4           = 228,    /*!< Debug function 4 */
    FUNC_DEBUG5           = 229,    /*!< Debug function 5 */
    FUNC_DEBUG6           = 230,    /*!< Debug function 6 */
    FUNC_DEBUG7           = 231,    /*!< Debug function 7 */
    FUNC_DEBUG8           = 232,    /*!< Debug function 8 */
    FUNC_DEBUG9           = 233,    /*!< Debug function 9 */
    FUNC_DEBUG10          = 234,    /*!< Debug function 10 */
    FUNC_DEBUG11          = 235,    /*!< Debug function 11 */
    FUNC_DEBUG12          = 236,    /*!< Debug function 12 */
    FUNC_DEBUG13          = 237,    /*!< Debug function 13 */
    FUNC_DEBUG14          = 238,    /*!< Debug function 14 */
    FUNC_DEBUG15          = 239,    /*!< Debug function 15 */
    FUNC_DEBUG16          = 240,    /*!< Debug function 16 */
    FUNC_DEBUG17          = 241,    /*!< Debug function 17 */
    FUNC_DEBUG18          = 242,    /*!< Debug function 18 */
    FUNC_DEBUG19          = 243,    /*!< Debug function 19 */
```

```
    FUNC_DEBUG20          = 244,    /*!< Debug function 20 */
    FUNC_DEBUG21          = 245,    /*!< Debug function 21 */
    FUNC_DEBUG22          = 246,    /*!< Debug function 22 */
    FUNC_DEBUG23          = 247,    /*!< Debug function 23 */
    FUNC_DEBUG24          = 248,    /*!< Debug function 24 */
    FUNC_DEBUG25          = 249,    /*!< Debug function 25 */
    FUNC_DEBUG26          = 250,    /*!< Debug function 26 */
    FUNC_DEBUG27          = 251,    /*!< Debug function 27 */
    FUNC_DEBUG28          = 252,    /*!< Debug function 28 */
    FUNC_DEBUG29          = 253,    /*!< Debug function 29 */
    FUNC_DEBUG30          = 254,    /*!< Debug function 30 */
    FUNC_DEBUG31          = 255,    /*!< Debug function 31 */
    FUNC_MAX              = 256,    /*!< Function numbers */
} fpioa_function_t;
```

3.3.1.3 Enumeration element

Element name	Description
FUNC_JTAG_TCLK	JTAG Test Clock
FUNC_JTAG_TDI	JTAG Test Data In
FUNC_JTAG_TMS	JTAG Test Mode Select
FUNC_JTAG_TDO	JTAG Test Data Out
FUNC_SPI0_D0	SPI0 Data 0
FUNC_SPI0_D1	SPI0 Data 1
FUNC_SPI0_D2	SPI0 Data 2
FUNC_SPI0_D3	SPI0 Data 3
FUNC_SPI0_D4	SPI0 Data 4
FUNC_SPI0_D5	SPI0 Data 5
FUNC_SPI0_D6	SPI0 Data 6
FUNC_SPI0_D7	SPI0 Data 7
FUNC_SPI0_SS0	SPI0 Chip Select 0
FUNC_SPI0_SS1	SPI0 Chip Select 1
FUNC_SPI0_SS2	SPI0 Chip Select 2
FUNC_SPI0_SS3	SPI0 Chip Select 3
FUNC_SPI0_ARB	SPI0 Arbitration
FUNC_SPI0_SCLK	SPI0 Serial Clock
FUNC_UARTHS_RX	UART High speed Receiver
FUNC_UARTHS_TX	UART High speed Transmitter
FUNC_RESV6	Clock Input 1
FUNC_RESV7	Clock Input 2

Element name	Description
FUNC_CLK_SPI1	Clock SPI1
FUNC_CLK_I2C1	Clock I2C1
FUNC_GPIOHS0	GPIO High speed 0
FUNC_GPIOHS1	GPIO High speed 1
FUNC_GPIOHS2	GPIO High speed 2
FUNC_GPIOHS3	GPIO High speed 3
FUNC_GPIOHS4	GPIO High speed 4
FUNC_GPIOHS5	GPIO High speed 5
FUNC_GPIOHS6	GPIO High speed 6
FUNC_GPIOHS7	GPIO High speed 7
FUNC_GPIOHS8	GPIO High speed 8
FUNC_GPIOHS9	GPIO High speed 9
FUNC_GPIOHS10	GPIO High speed 10
FUNC_GPIOHS11	GPIO High speed 11
FUNC_GPIOHS12	GPIO High speed 12
FUNC_GPIOHS13	GPIO High speed 13
FUNC_GPIOHS14	GPIO High speed 14
FUNC_GPIOHS15	GPIO High speed 15
FUNC_GPIOHS16	GPIO High speed 16
FUNC_GPIOHS17	GPIO High speed 17
FUNC_GPIOHS18	GPIO High speed 18
FUNC_GPIOHS19	GPIO High speed 19
FUNC_GPIOHS20	GPIO High speed 20
FUNC_GPIOHS21	GPIO High speed 21
FUNC_GPIOHS22	GPIO High speed 22
FUNC_GPIOHS23	GPIO High speed 23
FUNC_GPIOHS24	GPIO High speed 24
FUNC_GPIOHS25	GPIO High speed 25
FUNC_GPIOHS26	GPIO High speed 26
FUNC_GPIOHS27	GPIO High speed 27
FUNC_GPIOHS28	GPIO High speed 28
FUNC_GPIOHS29	GPIO High speed 29
FUNC_GPIOHS30	GPIO High speed 30
FUNC_GPIOHS31	GPIO High speed 31
FUNC_GPIO0	GPIO pin 0

Element name	Description
FUNC_GPIO1	GPIO pin 1
FUNC_GPIO2	GPIO pin 2
FUNC_GPIO3	GPIO pin 3
FUNC_GPIO4	GPIO pin 4
FUNC_GPIO5	GPIO pin 5
FUNC_GPIO6	GPIO pin 6
FUNC_GPIO7	GPIO pin 7
FUNC_UART1_RX	UART1 Receiver
FUNC_UART1_TX	UART1 Transmitter
FUNC_UART2_RX	UART2 Receiver
FUNC_UART2_TX	UART2 Transmitter
FUNC_UART3_RX	UART3 Receiver
FUNC_UART3_TX	UART3 Transmitter
FUNC_SPI1_D0	SPI1 Data 0
FUNC_SPI1_D1	SPI1 Data 1
FUNC_SPI1_D2	SPI1 Data 2
FUNC_SPI1_D3	SPI1 Data 3
FUNC_SPI1_D4	SPI1 Data 4
FUNC_SPI1_D5	SPI1 Data 5
FUNC_SPI1_D6	SPI1 Data 6
FUNC_SPI1_D7	SPI1 Data 7
FUNC_SPI1_SS0	SPI1 Chip Select 0
FUNC_SPI1_SS1	SPI1 Chip Select 1
FUNC_SPI1_SS2	SPI1 Chip Select 2
FUNC_SPI1_SS3	SPI1 Chip Select 3
FUNC_SPI1_ARB	SPI1 Arbitration
FUNC_SPI1_SCLK	SPI1 Serial Clock
FUNC_SPI_SLAVE_D0	SPI Slave Data 0
FUNC_SPI_SLAVE_SS	SPI Slave Select
FUNC_SPI_SLAVE_SCLK	SPI Slave Serial Clock
FUNC_I2S0_MCLK	I2S0 Master Clock
FUNC_I2S0_SCLK	I2S0 Serial Clock(BCLK)
FUNC_I2S0_WS	I2S0 Word Select(LRCLK)
FUNC_I2S0_IN_D0	I2S0 Serial Data Input 0
FUNC_I2S0_IN_D1	I2S0 Serial Data Input 1

Element name	Description
FUNC_I2S0_IN_D2	I2S0 Serial Data Input 2
FUNC_I2S0_IN_D3	I2S0 Serial Data Input 3
FUNC_I2S0_OUT_D0	I2S0 Serial Data Output 0
FUNC_I2S0_OUT_D1	I2S0 Serial Data Output 1
FUNC_I2S0_OUT_D2	I2S0 Serial Data Output 2
FUNC_I2S0_OUT_D3	I2S0 Serial Data Output 3
FUNC_I2S1_MCLK	I2S1 Master Clock
FUNC_I2S1_SCLK	I2S1 Serial Clock(BCLK)
FUNC_I2S1_WS	I2S1 Word Select(LRCLK)
FUNC_I2S1_IN_D0	I2S1 Serial Data Input 0
FUNC_I2S1_IN_D1	I2S1 Serial Data Input 1
FUNC_I2S1_IN_D2	I2S1 Serial Data Input 2
FUNC_I2S1_IN_D3	I2S1 Serial Data Input 3
FUNC_I2S1_OUT_D0	I2S1 Serial Data Output 0
FUNC_I2S1_OUT_D1	I2S1 Serial Data Output 1
FUNC_I2S1_OUT_D2	I2S1 Serial Data Output 2
FUNC_I2S1_OUT_D3	I2S1 Serial Data Output 3
FUNC_I2S2_MCLK	I2S2 Master Clock
FUNC_I2S2_SCLK	I2S2 Serial Clock(BCLK)
FUNC_I2S2_WS	I2S2 Word Select(LRCLK)
FUNC_I2S2_IN_D0	I2S2 Serial Data Input 0
FUNC_I2S2_IN_D1	I2S2 Serial Data Input 1
FUNC_I2S2_IN_D2	I2S2 Serial Data Input 2
FUNC_I2S2_IN_D3	I2S2 Serial Data Input 3
FUNC_I2S2_OUT_D0	I2S2 Serial Data Output 0
FUNC_I2S2_OUT_D1	I2S2 Serial Data Output 1
FUNC_I2S2_OUT_D2	I2S2 Serial Data Output 2
FUNC_I2S2_OUT_D3	I2S2 Serial Data Output 3
FUNC_RESV0	Reserved function
FUNC_RESV1	Reserved function
FUNC_RESV2	Reserved function
FUNC_RESV3	Reserved function
FUNC_RESV4	Reserved function
FUNC_RESV5	Reserved function
FUNC_I2C0_SCLK	I2C0 Serial Clock

Element name	Description
FUNC_I2C0_SDA	I2C0 Serial Data
FUNC_I2C1_SCLK	I2C1 Serial Clock
FUNC_I2C1_SDA	I2C1 Serial Data
FUNC_I2C2_SCLK	I2C2 Serial Clock
FUNC_I2C2_SDA	I2C2 Serial Data
FUNC_CMOS_XCLK	DVP System Clock
FUNC_CMOS_RST	DVP System Reset
FUNC_CMOS_PWDN	DVP Power Down Mode
FUNC_CMOS_VSYNC	DVP Vertical Sync
FUNC_CMOS_HREF	DVP Horizontal Reference output
FUNC_CMOS_PCLK	Pixel Clock
FUNC_CMOS_D0	Data Bit 0
FUNC_CMOS_D1	Data Bit 1
FUNC_CMOS_D2	Data Bit 2
FUNC_CMOS_D3	Data Bit 3
FUNC_CMOS_D4	Data Bit 4
FUNC_CMOS_D5	Data Bit 5
FUNC_CMOS_D6	Data Bit 6
FUNC_CMOS_D7	Data Bit 7
FUNC_SCCB_SCLK	SCCB Serial Clock
FUNC_SCCB_SDA	SCCB Serial Data
FUNC_UART1_CTS	UART1 Clear To Send
FUNC_UART1_DSR	UART1 Data Set Ready
FUNC_UART1_DCD	UART1 Data Carrier Detect
FUNC_UART1_RI	UART1 Ring Indicator
FUNC_UART1_SIR_IN	UART1 Serial Infrared Input
FUNC_UART1_DTR	UART1 Data Terminal Ready
FUNC_UART1_RTS	UART1 Request To Send
FUNC_UART1_OUT2	UART1 User-designated Output 2
FUNC_UART1_OUT1	UART1 User-designated Output 1
FUNC_UART1_SIR_OUT	UART1 Serial Infrared Output
FUNC_UART1_BAUD	UART1 Transmit Clock Output
FUNC_UART1_RE	UART1 Receiver Output Enable
FUNC_UART1_DE	UART1 Driver Output Enable
FUNC_UART1_RS485_EN	UART1 RS485 Enable

Element name	Description
FUNC_UART2_CTS	UART2 Clear To Send
FUNC_UART2_DSR	UART2 Data Set Ready
FUNC_UART2_DCD	UART2 Data Carrier Detect
FUNC_UART2_RI	UART2 Ring Indicator
FUNC_UART2_SIR_IN	UART2 Serial Infrared Input
FUNC_UART2_DTR	UART2 Data Terminal Ready
FUNC_UART2_RTS	UART2 Request To Send
FUNC_UART2_OUT2	UART2 User-designated Output 2
FUNC_UART2_OUT1	UART2 User-designated Output 1
FUNC_UART2_SIR_OUT	UART2 Serial Infrared Output
FUNC_UART2_BAUD	UART2 Transmit Clock Output
FUNC_UART2_RE	UART2 Receiver Output Enable
FUNC_UART2_DE	UART2 Driver Output Enable
FUNC_UART2_RS485_EN	UART2 RS485 Enable
FUNC_UART3_CTS	UART3 Clear To Send
FUNC_UART3_DSR	UART3 Data Set Ready
FUNC_UART3_DCD	UART3 Data Carrier Detect
FUNC_UART3_RI	UART3 Ring Indicator
FUNC_UART3_SIR_IN	UART3 Serial Infrared Input
FUNC_UART3_DTR	UART3 Data Terminal Ready
FUNC_UART3_RTS	UART3 Request To Send
FUNC_UART3_OUT2	UART3 User-designated Output 2
FUNC_UART3_OUT1	UART3 User-designated Output 1
FUNC_UART3_SIR_OUT	UART3 Serial Infrared Output
FUNC_UART3_BAUD	UART3 Transmit Clock Output
FUNC_UART3_RE	UART3 Receiver Output Enable
FUNC_UART3_DE	UART3 Driver Output Enable
FUNC_UART3_RS485_EN	UART3 RS485 Enable
FUNC_TIMER0_TOGGLE1	TIMER0 Toggle Output 1
FUNC_TIMER0_TOGGLE2	TIMER0 Toggle Output 2
FUNC_TIMER0_TOGGLE3	TIMER0 Toggle Output 3
FUNC_TIMER0_TOGGLE4	TIMER0 Toggle Output 4
FUNC_TIMER1_TOGGLE1	TIMER1 Toggle Output 1
FUNC_TIMER1_TOGGLE2	TIMER1 Toggle Output 2
FUNC_TIMER1_TOGGLE3	TIMER1 Toggle Output 3

Element name	Description
FUNC_TIMER1_TOGGLE4	TIMER1 Toggle Output 4
FUNC_TIMER2_TOGGLE1	TIMER2 Toggle Output 1
FUNC_TIMER2_TOGGLE2	TIMER2 Toggle Output 2
FUNC_TIMER2_TOGGLE3	TIMER2 Toggle Output 3
FUNC_TIMER2_TOGGLE4	TIMER2 Toggle Output 4
FUNC_CLK_SPI2	Clock SPI2
FUNC_CLK_I2C2	Clock I2C2
FUNC_INTERNAL0	Internal function 0
FUNC_INTERNAL1	Internal function 1
FUNC_INTERNAL2	Internal function 2
FUNC_INTERNAL3	Internal function 3
FUNC_INTERNAL4	Internal function 4
FUNC_INTERNAL5	Internal function 5
FUNC_INTERNAL6	Internal function 6
FUNC_INTERNAL7	Internal function 7
FUNC_INTERNAL8	Internal function 8
FUNC_INTERNAL9	Internal function 9
FUNC_INTERNAL10	Internal function 10
FUNC_INTERNAL11	Internal function 11
FUNC_INTERNAL12	Internal function 12
FUNC_INTERNAL13	Internal function 13
FUNC_INTERNAL14	Internal function 14
FUNC_INTERNAL15	Internal function 15
FUNC_INTERNAL16	Internal function 16
FUNC_INTERNAL17	Internal function 17
FUNC_CONSTANT	Constant function
FUNC_INTERNAL18	Internal function 18
FUNC_DEBUG0	Debug function 0
FUNC_DEBUG1	Debug function 1
FUNC_DEBUG2	Debug function 2
FUNC_DEBUG3	Debug function 3
FUNC_DEBUG4	Debug function 4
FUNC_DEBUG5	Debug function 5
FUNC_DEBUG6	Debug function 6
FUNC_DEBUG7	Debug function 7

Element name	Description
FUNC_DEBUG8	Debug function 8
FUNC_DEBUG9	Debug function 9
FUNC_DEBUG10	Debug function 10
FUNC_DEBUG11	Debug function 11
FUNC_DEBUG12	Debug function 12
FUNC_DEBUG13	Debug function 13
FUNC_DEBUG14	Debug function 14
FUNC_DEBUG15	Debug function 15
FUNC_DEBUG16	Debug function 16
FUNC_DEBUG17	Debug function 17
FUNC_DEBUG18	Debug function 18
FUNC_DEBUG19	Debug function 19
FUNC_DEBUG20	Debug function 20
FUNC_DEBUG21	Debug function 21
FUNC_DEBUG22	Debug function 22
FUNC_DEBUG23	Debug function 23
FUNC_DEBUG24	Debug function 24
FUNC_DEBUG25	Debug function 25
FUNC_DEBUG26	Debug function 26
FUNC_DEBUG27	Debug function 27
FUNC_DEBUG28	Debug function 28
FUNC_DEBUG29	Debug function 29
FUNC_DEBUG30	Debug function 30
FUNC_DEBUG31	Debug function 31

3.3.2 fpioa_cfg_item_t

3.3.2.1 Description

FPIOA pin configuration.

3.3.2.2 Type definition

```
typedef struct _fpioa_cfg_item
{
    int number;
    fpioa_function_t function;
} fpioa_cfg_item_t;
```

3.3.2.3 Enumeration element

Element name	Description
number	Pin number
function	Function number

3.3.3 fpioa_cfg_t

3.3.3.1 Description

FPIOA configuration.

3.3.3.2 Type definition

```
typedef struct _fpioa_cfg
{
    uint32_t version;
    uint32_t functions_count;
    fpioa_cfg_item_t functions[];
} fpioa_cfg_t;
```

3.3.3.3 Enumeration element

Element name	Description
version	Configuration version, must be set to FPIOA_CFG_VERSION
functions_count	Number of function configurations
functions	Function configurations list

3.3.4 sysctl_power_bank_t

3.3.4.1 Description

Power domain number.

3.3.4.2 Type definition

```
typedef enum _sysctl_power_bank
{
    SYSCTL_POWER_BANK0,
    SYSCTL_POWER_BANK1,
```

```
    SYSCTL_POWER_BANK2,
    SYSCTL_POWER_BANK3,
    SYSCTL_POWER_BANK4,
    SYSCTL_POWER_BANK5,
    SYSCTL_POWER_BANK6,
    SYSCTL_POWER_BANK7,
    SYSCTL_POWER_BANK_MAX,
} sysctl_power_bank_t;
```

3.3.4.3 Enumeration element

Element name	Description
SYSCTL_POWER_BANK0	Power domain 0, control IO0-IO5
SYSCTL_POWER_BANK1	Power domain 0, control IO6-IO11
SYSCTL_POWER_BANK2	Power domain 0, control IO12-IO17
SYSCTL_POWER_BANK3	Power domain 0, control IO18-IO23
SYSCTL_POWER_BANK4	Power domain 0, control IO24-IO29
SYSCTL_POWER_BANK5	Power domain 0, control IO30-IO35
SYSCTL_POWER_BANK6	Power domain 0, control IO36-IO41
SYSCTL_POWER_BANK7	Power domain 0, control IO42-IO47

3.3.5 sysctl_io_power_mode_t

3.3.5.1 Description

I/O output voltage selection。

3.3.5.2 Type definition

```
typedef enum _sysctl_io_power_mode
{
    SYSCTL_POWER_V33,
    SYSCTL_POWER_V18
} sysctl_io_power_mode_t;
```

3.3.5.3 Enumeration element

Element name	Description
SYSCTL_POWER_V33	Set to 3.3V
SYSCTL_POWER_V18	Set to 1.8V

3.3.6 power_bank_item_t

3.3.6.1 Description

Single power domain configuration.

3.3.6.2 Type definition

```
typedef struct _power_bank_item
{
    sysctl_power_bank_t power_bank;
    sysctl_io_power_mode_t io_power_mode;
} power_bank_item_t;
```

3.3.6.3 Enumeration element

Element name	Description
power_bank	Power domain number
io*power*mode	I/O output voltage selection

3.3.7 power_bank_cfg_t

3.3.7.1 Description

Power domain configuration.

3.3.7.2 Type definition

```
typedef struct _power_bank_cfg
{
    uint32_t version;
    uint32_t power_banks_count;
    power_bank_item_t power_banks[];
} power_bank_cfg_t;
```

3.3.7.3 Enumeration element

Element name	Description
version	Configuration version, must be set to FPIOA_CFG_VERSION
power*banks*count	Number of power domain configurations
power_banks	Power domain configuration list

3.3.8 pin_cfg_t

3.3.8.1 Description

Pin configuration.

3.3.8.2 Type definition

```
typedef struct _pin_cfg
{
    uint32_t version;
    bool set_spi0_dvp_data;
} pin_cfg_t;
```

3.3.8.3 Enumeration element

Element name	Description
version	Configuration version, must be set to FPIOA_CFG_VERSION
set-*spi0*dvp_data	Whether to set SPI*0D0-D7 and DVP*D0-D7 for SPI0 data output and DVP data input

3.3.9 Example

```
/* Configuration IO6, IO7 functions to GPIOHS0 and GPIOHS1 */
const fpioa_cfg_t g_fpioa_cfg =
{
    .version = FPIOA_CFG_VERSION,
    .functions_count = 2,
    .functions =
    {
        { .number = 6, .function = FUNC_GPIOHS0 },
        { .number = 7, .function = FUNC_GPIOHS1 }
    }
};
```

Chapter 4

System control (SYSCTL)

4.1 Overview

The system control (SYSCTL) unit can provide configuration functions for the SoC (system on chip).

4.2 Features

The system control module has the following features:

- Set the CPU frequency
- Install custom drivers

4.3 API

Corresponding header file `hal.h`
Provide the following interfaces

- system_set_cpu_frequency
- system_install_custom_driver

4.3.1 system_set_cpu_frequency

4.3.1.1 Description

Set the CPU frequency.

4.3.1.2 Function prototype

```
uint32_t system_set_cpu_frequency(uint32_t frequency);
```

4.3.1.3 Parameter

Parameter name	Description	Input or output
frequency	Frequency to be set (Hz)	Input

4.3.1.4 Return value

The actual frequency (Hz) after setting.

4.3.2 system_install_custom_driver

4.3.2.1 Description

Install a custom driver.

4.3.2.2 Function prototype

```
void system_install_custom_driver(const char *name, const custom_driver_t *driver);
```

4.3.2.3 Parameter

Parameter name	Description	Input or output
name	Specify the path to access the device	Input
driver	Custom driver implementation	Input

4.3.2.4 Return value

None.

4.3.3 Example

```
/* Set the CPU frequency to 400MHz */
system_set_cpu_frequency(400000000);
```

4.4 Data type

The relevant data types and data structures are defined as follows:

- driver_base_t: The base class for the driver implementation.
- custom_driver_t: Custom drive implementation.

4.4.1 driver_base_t

4.4.1.1 Description

The base class for the driver implementation.

4.4.1.2 Type definition

```
typedef struct _driver_base
{
    void *userdata;
    void (*install)(void *userdata);
    int (*open)(void *userdata);
    void (*close)(void *userdata);
} driver_base_t;
```

4.4.1.3 Enumeration element

Element name	Description
userdata	User data
install	Called during installation
open	Called when opened
close	Called when closed

4.4.2 custom_driver_t

4.4.2.1 Description

Custom drive implementation.

4.4.2.2 Type definition

```
typedef struct _custom_driver
```

```
{
    driver_base_t base;
    int (*io_control)(uint32_t control_code, const uint8_t *write_buffer, size_t
        write_len, uint8_t *read_buffer, size_t read_len, void *userdata);
} custom_driver_t;
```

4.4.2.3 Enumeration element

Element name	Description
base	Driver implementation base class
io_control	Called when control information is received

Chapter 5

PIC

5.1 Overview

Programmable Interrupt Controller (PIC).

Any external interrupt source can be individually assigned to an external interrupt on each CPU. This provides great flexibility to adapt to different application needs.

5.2 Features

The PIC module has the following features:

- Enable or disable interrupts
- Set the interrupt handler
- Configure interrupt priority

5.3 API

Corresponding header file `hal.h`

Provide the following interfaces

- pic_set_irq_enable
- pic_set_irq_handler
- pic_set_irq_priority

5.3.1 pic_set_irq_enable

5.3.1.1 Description

Set whether IRQ is enabled.

5.3.1.2 Function prototype

```
void pic_set_irq_enable(uint32_t irq, bool enable);
```

5.3.1.3 Parameter

Parameter name	Description	Input or output
irq	IRQ number	Input
enable	Whether to enable	Input

5.3.1.4 Return value

None.

5.3.2 pic_set_irq_handler

5.3.2.1 Description

Set the IRQ Handler.

5.3.2.2 Function prototype

```
void pic_set_irq_handler(uint32_t irq, pic_irq_handler_t handler, void *userdata);
```

5.3.2.3 Parameter

Parameter name	Description	Input or output
irq	IRQ number	Input
handler	Handler	Input
userdata	Handler user data	Input

5.3.2.4 Return value

None.

5.3.3 pic_set_irq_priority

5.3.3.1 Description

Set the IRQ priority.

5.3.3.2 Function prototype

```
void pic_set_irq_priority(uint32_t irq, uint32_t priority);
```

5.3.3.3 Parameter

Parameter name	Description	Input or output
irq	IRQ number	Input
priority	Priority	Input

5.3.3.4 Return value

None.

5.4 Data type

The relevant data types and data structures are defined as follows:

- pic_irq_handler_t: IRQ Handler。

5.4.1 pic_irq_handler_t

5.4.1.1 Description

IRQ Handler。

5.4.1.2 Type definition

```
typedef void (*pic_irq_handler_t)(void *userdata);
```

5.4.1.3 Parameter

Parameter name	Description	Input or output
userdata	User data	Input

Chapter 6

DMA

6.1 Overview

Direct Memory Access (DMA) controller is used to provide high-speed data transfer between peripherals and memory and between memory and memory. This improves CPU efficiency by quickly moving data through DMA without any CPU operation.

6.2 Features

The DMA controller has the following features:

- Automatically select an idle DMA channel for transmission
- Automatic selection of software or hardware handshake protocol based on source and destination addresses
- Supports 1, 2, 4, and 8 byte element sizes, source and target sizes do not have to be consistent
- Asynchronous or synchronous transfer function
- Loop transmission function, often used to refresh scenes such as screen or audio recording and playback

6.3 API

Corresponding header file `hal.h`
Provide the following interfaces

- dma_open_free

- dma_close
- dma_set_request_source
- dma_transmit_async
- dma_transmit
- dma_loop_async

6.3.1 dma_open_free

6.3.1.1 Description

Open an available DMA device.

6.3.1.2 Function prototype

```
handle_t dma_open_free();
```

6.3.1.3 Return value

DMA device handle.

6.3.2 dma_close

6.3.2.1 Description

Close the DMA device.

6.3.2.2 Function prototype

```
void dma_close(handle_t file);
```

6.3.2.3 Parameter

Parameter name	Description	Input or output
file	DMA device handle	Input

6.3.2.4 Return value

None.

6.3.3 dma_set_request_source

6.3.3.1 Description

Set the DMA request source.

6.3.3.2 Function prototype

```
void dma_set_request_source(handle_t file, uint32_t request);
```

6.3.3.3 Parameter

Parameter name	Description	Input or output
file	DMA device handle	Input
request	Request source number	Input

6.3.3.4 Return value

None.

6.3.4 dma_transmit_async

6.3.4.1 Description

Perform DMA asynchronous transfer.

6.3.4.2 Function prototype

```
void dma_transmit_async(handle_t file, const volatile void *src, volatile void *dest,
    int src_inc, int dest_inc, size_t element_size, size_t count, size_t burst_size,
    SemaphoreHandle_t completion_event);
```

6.3.4.3 Parameter

Parameter name	Description	Input or output
file	DMA device handle	Input
src	Source address	Input
dest	Destination address	Output
src_inc	Whether the source address is increasing	Input

Parameter name	Description	Input or output
dest_inc	Whether the destination address is increasing	Input
element_size	Element size (bytes)	Input
count	Element count	Input
burst_size	Burst transfer size	Input
completion_event	Transfer completion event	Input

6.3.4.4 Return value

None.

6.3.5 dma_transmit

6.3.5.1 Description

Perform DMA synchronous transfer.

6.3.5.2 Function prototype

```
void dma_transmit(handle_t file, const volatile void *src, volatile void *dest, int
    src_inc, int dest_inc, size_t element_size, size_t count, size_t burst_size);
```

6.3.5.3 Parameter

Parameter name	Description	Input or output
file	DMA device handle	Input
src	Source address	Input
dest	Destination address	Output
src_inc	Whether the source address is increasing	Input
dest_inc	Whether the destination address is increasing	Input
element_size	Element size (bytes)	Input
count	Element count	Input
burst_size	Burst transfer size	Input

6.3.5.4 Return value

None.

6.3.6 dma_loop_async

6.3.6.1 Description

Perform DMA asynchronous loop transfer.

6.3.6.2 Function prototype

```
void dma_loop_async(handle_t file, const volatile void **srcs, size_t src_num, volatile
     void **dests, size_t dest_num, int src_inc, int dest_inc, size_t element_size,
    size_t count, size_t burst_size, dma_stage_completion_handler_t
    stage_completion_handler, void *stage_completion_handler_data, SemaphoreHandle_t
    completion_event, int *stop_signal);
```

6.3.6.3 Parameter

Parameter name	Description	Input or output
file	DMA device handle	Input
srcs	Source address list	Input
src_num	Source address count	Input
dests	Destination address list	Output
dest_num	Destination address count	Input
src_inc	Whether the source address is increasing	Input
dest_inc	Whether the destination address is increasing	Input
element_size	Element size (bytes)	Input
count	Element count	Input
burst_size	Burst transfer size	Input
stage_completion_handler	Stage completion handler [*1]	Input
stage_completion_handler_data	Stage completion handler user data	Input
completion_event	Transfer completion event	Input
stop_signal	Stop signal	Input

[*1] Stage completion refers to the completion of the transfer of a single source to a destination N elements. N is elements count.

6.3.6.4 Return value

None.

6.3.7 Example

```
int src[256] = { [0 ... 255] = 1 };
int dest[256];
handle_t dma = dma_open_free();
dma_transmit(dma, src, dest, true, true, sizeof(int), 256, 4);
assert(dest[0] == src[0]);
dma_close(dma);
```

6.4 Data type

The relevant data types and data structures are defined as follows:

- dma_stage_completion_handler_t: DMA stage completion handler。

6.4.1 dma_stage_completion_handler_t

6.4.1.1 Description

DMA stage completion handler。

6.4.1.2 Type definition

```
typedef void (*dma_stage_completion_handler_t)(void *userdata);
```

6.4.1.3 Parameter

Parameter name	Description	Input or output
userdata	User data	Input

Chapter 7

Standard IO

7.1 Overview

The standard IO module is the basic interface for accessing peripherals.

7.2 Features

The standard IO module has the following features:

- Find peripherals based on the path
- Unified read and write and control interface

7.3 API

Corresponding header file `devices.h`
Provide the following interfaces

- io_open
- io_close
- io_read
- io_write
- io_control

7.3.1 io_open

7.3.1.1 Description

Open a device.

7.3.1.2 Function prototype

```
handle_t io_open(const char *name);
```

7.3.1.3 Parameter

Parameter name	Description	Input or output
name	Device path	Input

7.3.1.4 Return value

Return value	Description
0	Fail
Others	device handle

7.3.2 io_close

7.3.2.1 Description

Close a device.

7.3.2.2 Function prototype

```
int io_close(handle_t file);
```

7.3.2.3 Parameter

Parameter name	Description	Input or output
file	device handle	Input

7.3.2.4 Return value

Return value	Description
0	Success
Others	Fail

7.3.3 io_read

7.3.3.1 Description

Read from the device.

7.3.3.2 Function prototype

```
int io_read(handle_t file, uint8_t *buffer, size_t len);
```

7.3.3.3 Parameter

Parameter name	Description	Input or output
file	Device handle	Input
buffer	Destination buffer	Output
len	Maximum number of bytes read	Input

7.3.3.4 Return value

The number of bytes actually read.

7.3.4 io_write

7.3.4.1 Description

Write to the device.

7.3.4.2 Function prototype

```
int io_write(handle_t file, const uint8_t *buffer, size_t len);
```

7.3.4.3 Parameter

Parameter name	Description	Input or output
file	Device handle	Input
buffer	Source buffer	Input
len	The number of bytes to write	Input

7.3.4.4 Return value

Return value	Description
len	Success
Others	Fail

7.3.5 io_control

7.3.5.1 Description

Send control information to the device.

7.3.5.2 Function prototype

```
int io_control(handle_t file, uint32_t control_code, const uint8_t *write_buffer,
    size_t write_len, uint8_t *read_buffer, size_t read_len);
```

7.3.5.3 Parameter

Parameter name	Description	Input or output
file	Device handle	Input
control_code	Control code	Input
write_buffer	Source buffer	Input
write_len	The number of bytes to write	Input
read_buffer	Destination buffer	Output
read_len	Maximum number of bytes read	Input

7.3.5.4 Return value

The number of bytes actually read.

7.3.6 Example

```
handle_t uart = io_open("/dev/uart1");
io_write(uart, "hello\n", 6);
io_close(uart);
```

Chapter 8

UART

8.1 Overview

Universal Asynchronous Receiver/Transmitter (UART).

Embedded applications typically require a simple method that consumes less system resources to transfer data. The Universal Asynchronous Receiver/Transmitter (UART) meets these requirements with the flexibility to perform full-duplex data exchange with external devices.

8.2 Features

The UART module has the following features:

- Configuring UART parameters
- Automatically charge data to the buffer

8.3 API

Corresponding header file `devices.h`

Provide the following interfaces

- uart_config

8.3.1 uart_config

8.3.1.1 Description

Configure the UART device.

8.3.1.2 Function prototype

```
void uart_config(handle_t file, uint32_t baud_rate, uint32_t databits, uart_stopbits_t
    stopbits, uart_parity_t parity);
```

8.3.1.3 Parameter

Parameter name	Description	Input or output
file	UART device handle	Input
baud_rate	Baud rate	Input
databits	Data bits (5-8)	Input
stopbits	Stop bit	Input
parity	Parity bit	Input

8.3.1.4 Return value

None.

8.3.2 Example

```
handle_t uart = io_open("/dev/uart1");

uint8_t b = 1;
/* Write 1 byte */
io_write(uart, &b, 1);
/* Read 1 byte */
while (io_read(uart, &b, 1) != 1);
```

8.4 Data type

The relevant data types and data structures are defined as follows:

- uart_stopbits_t: UART stop bits.

- uart_parity_t: UART parity bit.

8.4.1 uart_stopbits_t

8.4.1.1 Description

UART stop bits.

8.4.1.2 Type definition

```
typedef enum _uart_stopbits
{
    UART_STOP_1,
    UART_STOP_1_5,
    UART_STOP_2
} uart_stopbits_t;
```

8.4.1.3 Enumeration element

Element name	Description
UART_STOP_1	1 stop bit
UART_STOP_1_5	1.5 stop bits
UART_STOP_2	2 stop bits

8.4.2 uart_parity_t

8.4.2.1 Description

UART parity bit.

8.4.2.2 Type definition

```
typedef enum _uart_parity
{
    UART_PARITY_NONE,
    UART_PARITY_ODD,
    UART_PARITY_EVEN
} uart_parity_t;
```

8.4.2.3 Enumeration element

Element name	Description
UART_PARITY_NONE	No parity bit
UART_PARITY_ODD	Odd parity
UART_PARITY_EVEN	Even parity

Chapter 9

GPIO

9.1 Overview

GPIO stands for General Purpose Input Output. The chip has 32 high-speed GPIOs and 8 general-purpose GPIOs.

9.2 Features

The GPIO unit has the following features:

- Configurable up and down drive mode
- Support for rising edge, falling edge and double edge trigger

9.3 API

Corresponding header file `devices.h`
Provide the following interfaces

- gpio_get_pin_count
- gpio_set_drive_mode
- gpio_set_pin_edge
- gpio_set_on_\changed
- gpio_get_pin_value
- gpio_set_pin_value

9.3.1 gpio_get_pin_count

9.3.1.1 Description

Get the GPIO pin count.

9.3.1.2 Function prototype

```
uint32_t gpio_get_pin_count(handle_t file);
```

9.3.1.3 Parameter

Parameter name	Description	Input or output
file	GPIO controller handle	Input

9.3.1.4 Return value

Pin count.

9.3.2 gpio_set_drive_mode

9.3.2.1 Description

Set the GPIO pin drive mode.

9.3.2.2 Function prototype

```
void gpio_set_drive_mode(handle_t file, uint32_t pin, gpio_drive_mode_t mode);
```

9.3.2.3 Parameter

Parameter name	Description	Input or output
file	GPIO controller handle	Input
pin	Pin number	Input
mode	Drive mode	Input

9.3.2.4 Return value

None.

9.3.3 gpio_set_pin_edge

9.3.3.1 Description

Set the GPIO pin edge trigger mode.

Note: /dev/gpio1 is not supported this function.

9.3.3.2 Function prototype

```
void gpio_set_pin_edge(handle_t file, uint32_t pin, gpio_pin_edge_t edge);
```

9.3.3.3 Parameter

Parameter name	Description	Input or output
file	GPIO controller handle	Input
pin	Pin number	Input
edge	Edge trigger mode	Input

9.3.3.4 Return value

None.

9.3.4 gpio_set_on_changed

9.3.4.1 Description

Set the GPIO pin edge trigger handler.

Note: /dev/gpio1 is not supported this function.

9.3.4.2 Function prototype

```
void gpio_set_on_changed(handle_t file, uint32_t pin, gpio_on_changed_t callback, void
    *userdata);
```

9.3.4.3 Parameter

Parameter name	Description	Input or output
file	GPIO controller handle	Input

Parameter name	Description	Input or output
pin	Pin number	Input
callback	Handler	Input
userdata	Handler user data	Input

9.3.4.4 Return value

None.

9.3.5 gpio_get_pin_value

9.3.5.1 Description

Get the value of the GPIO pin.

9.3.5.2 Function prototype

```
gpio_pin_value_t gpio_get_pin_value(handle_t file, uint32_t pin);
```

9.3.5.3 Parameter

Parameter name	Description	Input or output
file	GPIO controller handle	Input
pin	Pin number	Input

9.3.5.4 Return value

The value of the GPIO pin.

9.3.6 gpio_set_pin_value

9.3.6.1 Description

Set the value of the GPIO pin.

9.3.6.2 Function prototype

```
void gpio_set_pin_value(handle_t file, uint32_t pin, gpio_pin_value_t value);
```

9.3.6.3 Parameter

Parameter name	Description	Input or output
file	GPIO controller handle	Input
pin	Pin number	Input
value	The value to set	Input

9.3.6.4 Return value

None.

9.3.7 Example

```
handle_t gpio = io_open("/dev/gpio0");

gpio_set_drive_mode(gpio, 0, GPIO_DM_OUTPUT);
gpio_set_pin_value(gpio, 0, GPIO_PV_LOW);
```

9.4 Data type

The relevant data types and data structures are defined as follows:

- gpio_drive_mode_t: GPIO drive mode.
- gpio_pin_edge_t: GPIO edge trigger mode.
- gpio_pin_value_t: GPIO value.
- gpio_on_changed_t: GPIO edge trigger handler。

9.4.1 gpio_drive_mode_t

9.4.1.1 Description

GPIO drive mode.

9.4.1.2 Type definition

```
typedef enum _gpio_drive_mode
{
    GPIO_DM_INPUT,
    GPIO_DM_INPUT_PULL_DOWN,
    GPIO_DM_INPUT_PULL_UP,
    GPIO_DM_OUTPUT
} gpio_drive_mode_t;
```

9.4.1.3 Enumeration element

Element name	Description
GPIO_DM_INPUT	Input
GPIO_DM_INPUT_PULL_DOWN	Input pull down
GPIO_DM_INPUT_PULL_UP	Input pull up
GPIO_DM_OUTPUT	Output

9.4.2 gpio_pin_edge_t

9.4.2.1 Description

GPIO edge trigger mode.

9.4.2.2 Type definition

```
typedef enum _gpio_pin_edge
{
    GPIO_PE_NONE,
    GPIO_PE_FALLING,
    GPIO_PE_RISING,
    GPIO_PE_BOTH
} gpio_pin_edge_t;
```

9.4.2.3 Enumeration element

Element name	Description
GPIO_PE_NONE	Do not trigger
GPIO_PE_FALLING	Falling edge trigger
GPIO_PE_RISING	Rising edge trigger
GPIO_PE_BOTH	Double edge trigger

9.4.3 gpio_pin_value_t

9.4.3.1 Description

GPIO value.

9.4.3.2 Type definition

```
typedef enum _gpio_pin_value
{
    GPIO_PV_LOW,
    GPIO_PV_HIGH
} gpio_pin_value_t;
```

9.4.3.3 Enumeration element

Element name	Description
GPIO_PV_LOW	Low
GPIO_PV_HIGH	High

9.4.4 gpio_on_changed_t

9.4.4.1 Description

GPIO edge trigger handler。

9.4.4.2 Type definition

```
typedef void (*gpio_on_changed_t)(uint32_t pin, void *userdata);
```

9.4.4.3 Parameter

Parameter name	Description	Input or output
pin	Pin number	Input
userdata	User data	Input

Chapter 10

I²C

10.1 Overview

The I²C (Inter-Integrated Circuit) bus is used to communicate with multiple external I²C devices. Multiple external I²C devices can share an I²C bus.

10.2 Features

The I²C unit has the following features:

- Independent I²C controller
- Automatic processing of multi-device bus contention
- Support slave mode

10.3 API

Corresponding header file `devices.h`
Provide the following interfaces

- i2c_get_device
- i2c_dev_set_clock_rate
- i2c_dev_transfer_sequential
- i2c_config_as_slave
- i2c_slave_set_clock_rate

10.3.1 i2c_get_device

10.3.1.1 Description

Register and open an I²C device.

10.3.1.2 Function prototype

```
handle_t i2c_get_device(handle_t file, const char *name, uint32_t slave_address,
    uint32_t address_width);
```

10.3.1.3 Parameter

Parameter name	Description	Input or output
file	I²C controller handle	Input
name	Specify the path to access the device	Input
slave_address	Slave address	Input
address_width	Slave address width	Input

10.3.1.4 Return value

I²C device handle.

10.3.2 i2c_dev_set_clock_rate

10.3.2.1 Description

Configure the clock rate of the I²C device.

10.3.2.2 Function prototype

```
double i2c_dev_set_clock_rate(handle_t file, double clock_rate);
```

10.3.2.3 Parameter

Parameter name	Description	Input or output
file	I²C device handle	Input
clock_rate	Expected clock rate	Input

10.3.2.4 Return value

The actual clock rate after setting.

10.3.3 i2c_dev_transfer_sequential

10.3.3.1 Description

Read the I²C device, then write it.

10.3.3.2 Function prototype

```
int i2c_dev_transfer_sequential(handle_t file, const uint8_t *write_buffer, size_t
    write_len, uint8_t *read_buffer, size_t read_len);
```

10.3.3.3 Parameter

Parameter name	Description	Input or output
file	I²C device handle	Input
write_buffer	Source buffer	Input
write_len	The number of bytes to write	Input
read_buffer	Destination buffer	Output
read_len	Maximum number of bytes read	Input

10.3.3.4 Return value

The number of bytes actually read.

10.3.4 i2c_config_as_slave

10.3.4.1 Description

Configure the I²C controller to be in slave mode.

10.3.4.2 Function prototype

```
void i2c_config_as_slave(handle_t file, uint32_t slave_address, uint32_t address_width,
     i2c_slave_handler_t *handler);
```

10.3.4.3 Parameter

Parameter name	Description	Input or output
file	I²C controller handle	Input
slave_address	Slave address	Input
address_width	Slave address width	Input
handler	Slave device handler	Input

10.3.4.4 Return value

None.

10.3.5 spi_dev_set_clock_rate

10.3.5.1 Description

Configure the clock rate for the I²C Slave mode.

10.3.5.2 Function prototype

```
double i2c_slave_set_clock_rate(handle_t file, double clock_rate);
```

10.3.5.3 Parameter

Parameter name	Description	Input or output
file	I²C controller handle	Input
clock_rate	Expected clock rate	Input

10.3.5.4 Return value

The actual clock rate after setting.

10.3.6 Example

```
handle_t i2c = io_open("/dev/i2c0");
/* The i2c peripheral address is 0x32, 7-bit address, rate 200K */
handle_t dev0 = i2c_get_device(i2c, "/dev/i2c0/dev0", 0x32, 7);
i2c_dev_set_clock_rate(dev0, 200000);

uint8_t reg = 0;
uint8_t data_buf[2] = { 0x00,0x01 };
data_buf[0] = reg;
```

```
/* Write 0x01 to the 0 register */
io_write(dev0, data_buf, 2);
/* Read 1 byte data from 0 register */
i2c_dev_transfer_sequential(dev0, &reg, 1, data_buf, 1);
```

10.4 Data type

The relevant data types and data structures are defined as follows:

- i2c_event_t: I²C event.
- i2c_slave_handler_t: I²C slave device handler.

10.4.1 i2c_event_t

10.4.1.1 Description

I²C event.

10.4.1.2 Type definition

```
typedef enum _i2c_event
{
    I2C_EV_START,
    I2C_EV_RESTART,
    I2C_EV_STOP
} i2c_event_t;
```

10.4.1.3 Enumeration element

Element name	Description
I2C_EV_START	Received start signal
I2C_EV_RESTART	Received restart signal
I2C_EV_STOP	Received stop signal

10.4.2 i2c_slave_handler_t

10.4.2.1 Description

I²C slave device handler.

10.4.2.2 Type definition

```
typedef struct _i2c_slave_handler
{
    void (*on_receive)(uint32_t data);
    uint32_t (*on_transmit)();
    void (*on_event)(i2c_event_t event);
} i2c_slave_handler_t;
```

10.4.2.3 Enumeration element

Element name	Description
on_receive	Called when data is received
on_transmit	Called when data needs to be sent
on_event	Called when an event occurs

Chapter 11

I²S

11.1 Overview

The Integrated Inter-IC Sound Bus (I²S) is a serial bus interface standard used for connecting digital audio devices together.

The I²S bus defines three types of signals: the clock signal BCK, the channel selection signal WS, and the serial data signal SD. A basic I²S bus has one master and one slave. The roles of the master and slave remain unchanged during the communication process. The I²S unit includes separate transmit and receive channels for excellent communication performance.

11.2 Features

The I²S module has the following features:

- Automatically configure the device according to the audio format (supports 16, 24, 32 bit depth, 44100 sample rate, 1 - 4 channels)
- Configurable for playback or recording mode
- Automatic management of audio buffers

11.3 API

Corresponding header file `devices.h`
Provide the following interfaces

- i2s_config_as_render

- i2s_config_as_capture
- i2s_get_buffer
- i2s_release_buffer
- i2s_start
- i2s_stop

11.3.1 i2s_config_as_render

11.3.1.1 Description

Configure the I²S controller to be in output mode.

11.3.1.2 Function prototype

```
void i2s_config_as_render(handle_t file, const audio_format_t *format, size_t delay_ms,
     i2s_align_mode_t align_mode, size_t channels_mask);
```

11.3.1.3 Parameter

Parameter name	Description	Input or output
file	I²S controller handle	Input
format	Audio format	Input
delay_ms	Buffer delay time	Input
align_mode	Align mode	Input
channels_mask	Channel mask	Input

11.3.1.4 Return value

None.

11.3.2 i2s_config_as_capture

11.3.2.1 Description

Configure the I²S controller to capture mode.

11.3.2.2 Function prototype

```
void i2s_config_as_capture(handle_t file, const audio_format_t *format, size_t delay_ms
    , i2s_align_mode_t align_mode, size_t channels_mask);
```

11.3.2.3 Parameter

Parameter name	Description	Input or output
file	I²S controller handle	Input
format	Audio format	Input
delay_ms	Buffer delay time	Input
align_mode	Align mode	Input
channels_mask	Channel mask	Input

11.3.2.4 Return value

None.

11.3.3 i2s_get_buffer

11.3.3.1 Description

Get the audio buffer.

11.3.3.2 Function prototype

```
void i2s_get_buffer(handle_t file, uint8_t **buffer, size_t *frames);
```

11.3.3.3 Parameter

Parameter name	Description	Input or output
file	I²S controller handle	Input
buffer	Buffer	Output
frames	Number of buffer frames	Output

11.3.3.4 Return value

None.

11.3.4 i2s_release_buffer

11.3.4.1 Description

Release the audio buffer.

11.3.4.2 Function prototype

```
void i2s_release_buffer(handle_t file, size_t frames);
```

11.3.4.3 Parameter

Parameter name	Description	Input or output
file	I²S controller handle	Input
frames	Confirm the number of frames that have been read or written	Input

11.3.4.4 Return value

None.

11.3.5 i2s_start

11.3.5.1 Description

Start playing or recording.

11.3.5.2 Function prototype

```
void i2s_start(handle_t file);
```

11.3.5.3 Parameter

Parameter name	Description	Input or output
file	I²S controller handle	Input

11.3.5.4 Return value

None.

11.3.6 i2s_stop

11.3.6.1 Description

Stop playing or recording.

11.3.6.2 Function prototype

```
void i2s_stop(handle_t file);
```

11.3.6.3 Parameter

Parameter name	Description	Input or output
file	I²S controller handle	Input

11.3.6.4 Return value

None.

11.3.7 Example

```
/* Loop play PCM audio */
handle_t i2s = io_open("/dev/i2s0");
audio_format_t audio_fmt = { .type = AUDIO_FMT_PCM, .bits_per_sample = 16, .sample_rate
     = 44100, .channels = 2 };
i2s_config_as_render(i2s, &audio_fmt, 100, I2S_AM_RIGHT, 0b11);
i2s_start(i2s);

while (1)
{
    uint8_t *buffer;
    size_t frames;
    i2s_get_buffer(i2s, &buffer, &frames);
    memcpy(buffer, pcm, 4 * frames);
    i2s_release_buffer(i2s, frames);
    pcm += frames;
    if (pcm >= pcm_end)
        pcm = pcm_start;
}
```

11.4 Data type

The relevant data types and data structures are defined as follows:

- audio_format_type_t: Audio format type.
- audio_format_t: Audio format.
- i2s_align_mode_t: I²S align mode.

11.4.1 audio_format_type_t

11.4.1.1 Description

Audio format type.

11.4.1.2 Type definition

```
typedef enum _audio_format_type
{
    AUDIO_FMT_PCM
} audio_format_type_t;
```

11.4.1.3 Enumeration element

Element name	Description
AUDIO_FMT_PCM	PCM

11.4.2 audio_format_t

11.4.2.1 Description

Audio format.

11.4.2.2 Type definition

```
typedef struct _audio_format
{
    audio_format_type_t type;
    uint32_t bits_per_sample;
    uint32_t sample_rate;
    uint32_t channels;
} audio_format_t;
```

11.4.2.3 Enumeration element

Element name	Description
type	Audio format type
bits_per_sample	Sampling bits per sample
sample_rate	Sample rate

Element name	Description
channels	Number of channels

11.4.3 i2s_align_mode_t

11.4.3.1 Description

I²S align mode.

11.4.3.2 Type definition

```
typedef enum _i2s_align_mode
{
    I2S_AM_STANDARD,
    I2S_AM_RIGHT,
    I2S_AM_LEFT
} i2s_align_mode_t;
```

11.4.3.3 Enumeration element

Element name	Description
I2S_AM_STANDARD	Standard mode
I2S_AM_RIGHT	Right justified mode
I2S_AM_LEFT	Left justified mode

Chapter 12

SPI

12.1 Overview

The Serial Peripheral Interface (SPI) is a synchronous serial communication interface. It is a high speed, full duplex, synchronous communication interface.

12.2 Features

The SPI module has the following features:

- Independent SPI device interface with peripheral related parameters
- Automatic processing of multi-device bus contention
- Support standard, two-wire, four-wire, eight-wire mode
- Supports write-before-read and full-duplex read and write
- Supports sending a series of identical data frames, often used for clearing screens, filling storage sectors, etc.

12.3 API

Corresponding header file `devices.h`
Provide the following interfaces

- spi_get_device
- spi_dev_config_non_standard
- spi_dev_set_clock_rate
- spi_dev_transfer_full_duplex

- spi_dev_transfer_sequential
- spi_dev_fill

12.3.1 spi_get_device

12.3.1.1 Description

Register and open an SPI device.

12.3.1.2 Function prototype

```
handle_t spi_get_device(handle_t file, const char *name, spi_mode mode,
    spi_frame_format frame_format, uint32_t chip_select_mask, uint32_t data_bit_length
    );
```

12.3.1.3 Parameter

Parameter name	Description	Input or output
file	SPI controller handle	Input
name	Specify the path to access the device	Input
mode	SPI mode	Input
frame_format	Frame format	Input
chip_select_mask	Chip select mask	Input
data_bit_length	Data bit length	Input

12.3.1.4 Return value

SPI device handle.

12.3.2 spi_dev_config_non_standard

12.3.2.1 Description

Configure non-standard frame format parameters for the SPI device.

12.3.2.2 Function prototype

```
void spi_dev_config_non_standard(handle_t file, uint32_t instruction_length, uint32_t
    address_length, uint32_t wait_cycles, spi_inst_addr_trans_mode_t trans_mode);
```

12.3.2.3 Parameter

Parameter name	Description	Input or output
file	SPI device handle	Input
instruction_length	Instruction length	Input
address_length	Address length	Input
wait_cycles	Number of waiting cycles	Input
trans_mode	Command and address transfer mode	Input

12.3.2.4 Return value

None.

12.3.3 spi_dev_set_clock_rate

12.3.3.1 Description

Configure the clock rate of the SPI device.

12.3.3.2 Function prototype

```
double spi_dev_set_clock_rate(handle_t file, double clock_rate);
```

12.3.3.3 Parameter

Parameter name	Description	Input or output
file	SPI device handle	Input
clock_rate	Expected clock rate	Input

12.3.3.4 Return value

The actual rate after setting.

12.3.4 spi_dev_transfer_full_duplex

12.3.4.1 Description

Full-duplex transmission of SPI devices.

Note: Only standard frame formats are supported.

12.3.4.2 Function prototype

```
int spi_dev_transfer_full_duplex(handle_t file, const uint8_t *write_buffer, size_t
    write_len, uint8_t *read_buffer, size_t read_len);
```

12.3.4.3 Parameter

Parameter name	Description	Input or output
file	SPI device handle	Input
write_buffer	Source buffer	Input
write_len	The number of bytes to write	Input
read_buffer	Target buffer	Output
read_len	Maximum number of bytes read	Input

12.3.4.4 Return value

The number of bytes actually read.

12.3.5 spi_dev_transfer_sequential

12.3.5.1 Description

Write the SPI device first and then read it (write-before-read).

Note: Only standard frame formats are supported.

12.3.5.2 Function prototype

```
int spi_dev_transfer_sequential(handle_t file, const uint8_t *write_buffer, size_t
    write_len, uint8_t *read_buffer, size_t read_len);
```

12.3.5.3 Parameter

Parameter name	Description	Input or output
file	SPI device handle	Input
write_buffer	Source buffer	Input
write_len	The number of bytes to write	Input
read_buffer	Target buffer	Output

Parameter name	Description	Input or output
read_len	Maximum number of bytes read	Input

12.3.5.4 Return value

The number of bytes actually read.

12.3.6 spi_dev_fill

12.3.6.1 Description

Fill the SPI device with a sequence of identical frames.

Note: Only standard frame formats are supported.

12.3.6.2 Function prototype

```
void spi_dev_fill(handle_t file, uint32_t instruction, uint32_t address, uint32_t value
    , size_t count);
```

12.3.6.3 Parameter

Parameter name	Description	Input or output
file	SPI device handle	Input
instruction	Instruction (ignored in standard frame format)	Input
address	Address (ignored in standard frame format)	Input
value	Frame data	Output
count	Number of frames	Input

12.3.6.4 Return value

None.

12.3.7 Example

```
handle_t spi = io_open("/dev/spi0");
/* dev0 works in MODE0 (Standard SPI mode), single transmission, 8-bit data and using
    chip select 0 */
handle_t dev0 = spi_get_device(spi, "/dev/spi0/dev0", SPI_MODE_0, SPI_FF_STANDARD, 0b1,
     8);
uint8_t data_buf[] = { 0x06, 0x01, 0x02, 0x04, 0, 1, 2, 3 };
```

```
/* Send instruction 0x06. Send 0,1,2,3 four bytes of data to address 0x010204 */
io_write(dev0, data_buf, sizeof(data_buf));
/* Send instruction 0x06. Receive four bytes of data from address 0x010204 */
spi_dev_transfer_sequential(dev0, data_buf, 4, data_buf, 4);
```

12.4 Data type

The relevant data types and data structures are defined as follows:

- spi_mode_t: SPI mode.
- spi_frame_format_t: SPI frame format.
- spi_inst_addr_trans_mode_t: The transfer mode of the SPI instruction and address.

12.4.1 spi_mode_t

12.4.1.1 Description

SPI mode.

12.4.1.2 Type definition

```
typedef enum _spi_mode
{
    SPI_MODE_0,
    SPI_MODE_1,
    SPI_MODE_2,
    SPI_MODE_3,
} spi_mode_t;
```

12.4.1.3 Enumeration element

Element name	Description
SPI_MODE_0	SPI mode 0
SPI_MODE_1	SPI mode 1
SPI_MODE_2	SPI mode 2
SPI_MODE_3	SPI mode 3

12.4.2 spi_frame_format_t

12.4.2.1 Description

SPI frame format.

12.4.2.2 Type definition

```
typedef enum _spi_frame_format
{
    SPI_FF_STANDARD,
    SPI_FF_DUAL,
    SPI_FF_QUAD,
    SPI_FF_OCTAL
} spi_frame_format_t;
```

12.4.2.3 Enumeration element

Element name	Description
SPI_FF_STANDARD	Standard
SPI_FF_DUAL	Dual wire (2 wires)
SPI_FF_QUAD	Duad wire (4 wires)
SPI_FF_OCTAL	Octal wire (8 wires, /dev/spi3 is not supported)

12.4.3 spi_inst_addr_trans_mode_t

12.4.3.1 Description

The transfer mode of the SPI instruction and address.

12.4.3.2 Type definition

```
typedef enum _spi_inst_addr_trans_mode
{
    SPI_AITM_STANDARD,
    SPI_AITM_ADDR_STANDARD,
    SPI_AITM_AS_FRAME_FORMAT
} spi_inst_addr_trans_mode_t;
```

12.4.3.3 Enumeration element

Element name	Description
SPI_AITM_STANDARD	All use the standard frame format
SPI_AITM_ADDR_STANDARD	The instruction uses the configured value and the address uses the standard frame format
SPI_AITM_AS_FRAME_FORMAT	All use the configured values

Chapter 13

DVP

13.1 Overview

Digital Video Port (DVP) unit is a camera interface unit that supports forwarding camera input image data to KPU or memory.

13.2 Features

The DVP unit has the following features:

- Support RGB565, RGB422 and single channel Y gray scale input mode
- Support for setting frame interrupt
- Support setting transfer address
- Supports writing data to two addresses at the same time (output format is RGB888 and RGB565 respectively)
- Support for discarding frames that do not need to be processed

13.3 API

Corresponding header file `devices.h`
Provide the following interfaces

- dvp_xclk_set_clock_rate
- dvp_config
- dvp_enable_frame
- dvp_get_output_num

- dvp_set_signal
- dvp_set_output_enable
- dvp_set_output_attributes
- dvp_set_frame_event_enable
- dvp_set_on_frame_event

13.3.1 dvp_xclk_set_clock_rate

13.3.1.1 Description

Configure the frequency of DVP XCLK.

13.3.1.2 Function prototype

```
double dvp_xclk_set_clock_rate(handle_t file, double clock_rate);
```

13.3.1.3 Parameter

Parameter name	Description	Input or output
file	DVP device handle	Input
clock_rate	Configure the frequency of XCLK, such as OV5640 configured to 20MHz	Input

13.3.1.4 Return value

The actual frequency after setting.

13.3.2 dvp_config

13.3.2.1 Description

Configure the DVP device.

13.3.2.2 Function prototype

```
void dvp_config(handle_t file, uint32_t width, uint32_t height, bool auto_enable);
```

13.3.2.3 Parameter

Parameter name	Description	Input or output
file	DVP device handle	Input
width	Frame width	Input
height	Frame height	Input
auto_enable	Automatically enable frame processing	Input

13.3.2.4 Return value

None.

13.3.3 dvp_enable_frame

13.3.3.1 Description

Enable processing of the current frame.

13.3.3.2 Function prototype

```
void dvp_enable_frame(handle_t file);
```

13.3.3.3 Parameter

Parameter name	Description	Input or output
file	DVP device handle	Input

13.3.3.4 Return value

None.

13.3.4 dvp_get_output_num

13.3.4.1 Description

Get the number of outputs of the DVP device.

13.3.4.2 Function prototype

```
uint32_t dvp_get_output_num(handle_t file);
```

13.3.4.3 Parameter

Parameter name	Description	Input or output
file	DVP device handle	Input

13.3.4.4 Return value

Output number.

13.3.5 dvp_set_signal

13.3.5.1 Description

Set the DVP signal status.

13.3.5.2 Function prototype

```
void dvp_set_signal(handle_t file, dvp_signal_type_t type, bool value);
```

13.3.5.3 Parameter

Parameter name	Description	Input or output
file	DVP device handle	Input
type	Signal type	Input
value	Status value	Input

13.3.5.4 Return value

None.

13.3.6 dvp_set_output_enable

13.3.6.1 Description

Set whether DVP output is enabled.

13.3.6.2 Function prototype

```
void dvp_set_output_enable(handle_t file, uint32_t index, bool enable);
```

13.3.6.3 Parameter

Parameter name	Description	Input or output
file	DVP device handle	Input
index	Output index	Input
enable	Whether to enable	Input

13.3.6.4 Return value

None.

13.3.7 dvp_set_output_attributes

13.3.7.1 Description

Set the DVP output attributes.

13.3.7.2 Function prototype

```
void dvp_set_output_attributes(handle_t file, uint32_t index, video_format_t format,
    void *output_buffer);
```

13.3.7.3 Parameter

Parameter name	Description	Input or output
file	DVP device handle	Input
index	Output index	Input
format	Video format	Input
output_buffer	Output buffer	Output

13.3.7.4 Return value

None.

13.3.8 dvp_set_frame_event_enable

13.3.8.1 Description

Set whether DVP frame events are enabled.

13.3.8.2 Function prototype

```
void dvp_set_frame_event_enable(handle_t file, dvp_frame_event_t event, bool enable);
```

13.3.8.3 Parameter

Parameter name	Description	Input or output
file	DVP device handle	Input
event	Frame event	Input
enable	Whether to enable	Input

13.3.8.4 Return value

None.

13.3.9 dvp_set_on_frame_event

13.3.9.1 Description

Set up a DVP frame event handler.

13.3.9.2 Function prototype

```
void dvp_set_on_frame_event(handle_t file, dvp_on_frame_event_t handler, void *userdata
    );
```

13.3.9.3 Parameter

Parameter name	Description	Input or output
file	DVP device handle	Input
handler	Handler	Input
userdata	Handler user data	Input

13.3.9.4 Return value

None.

13.3.10 Example

```
handle_t dvp = io_open("/dev/dvp0");
```

```
dvp_config(dvp, 320, 240, false);
dvp_set_on_frame_event(dvp, on_frame_isr, NULL);
dvp_set_frame_event_enable(dvp, VIDEO_FE_BEGIN, true);
dvp_set_output_attributes(dvp, 0, VIDEO_FMT_RGB565, lcd_gram0);
dvp_set_output_enable(dvp, 0, true);
```

13.4 Data type

The relevant data types and data structures are defined as follows:

- video_format_t: Video format.
- dvp_frame_event_t: DVP frame event.
- dvp_signal_type_t: DVP signal type.
- dvp_on_frame_event_t: The handler when the frame event is triggered.

13.4.1 video_format_t

13.4.1.1 Description

Video format.

13.4.1.2 Type definition

```
typedef enum _video_format
{
    VIDEO_FMT_RGB565,
    VIDEO_FMT_RGB24_PLANAR
} video_format_t;
```

13.4.1.3 Enumeration element

Element name	Description
VIDEO_FMT_RGB565	RGB565
VIDEO_FMT_RGB24_PLANAR	RGB24 Planar

13.4.2 dvp_frame_event_t

13.4.2.1 Description

DVP frame event.

13.4.2.2 Type definition

```
typedef enum _video_frame_event
{
    VIDEO_FE_BEGIN,
    VIDEO_FE_END
} dvp_frame_event_t;
```

13.4.2.3 Enumeration element

Element name	Description
VIDEO_FE_BEGIN	Frame start
VIDEO_FE_END	End of frame

13.4.3 dvp_signal_type_t

13.4.3.1 Description

DVP signal type.

13.4.3.2 Type definition

```
typedef enum _dvp_signal_type
{
    DVP_SIG_POWER_DOWN,
    DVP_SIG_RESET
} dvp_signal_type_t;
```

13.4.3.3 Enumeration element

Element name	Description
DVP_SIG_POWER_DOWN	Power down signal
DVP_SIG_RESET	Reset signal

13.4.4 dvp_on_frame_event_t

13.4.4.1 Description

The handler when the frame event is triggered.

13.4.4.2 Type definition

```
typedef void (*dvp_on_frame_event_t)(dvp_frame_event_t event, void *userdata);
```

13.4.4.3 Parameter

Parameter name	Description	Input or output
userdata	User data	Input

Chapter 14

SCCB

14.1 Overview

Serial Camera Control Bus (SCCB) interface. The SCCB protocol is very similar to the I²C protocol.

14.2 Features

The SCCB module has the following features:

- Independently packaged peripheral related parameters
- Automatic processing of multi-device bus contention

14.3 API

Corresponding header file `devices.h`
Provide the following interfaces

- sccb_get_device
- sccb_dev_read_byte
- sccb_dev_write_byte

14.3.1 sccb_get_device

14.3.1.1 Description

Register and open an SCCB device.

14.3.1.2 Function prototype

```
handle_t sccb_get_device(handle_t file, const char *name, size_t slave_address, size_t
    reg_address_width);
```

14.3.1.3 Parameter

Parameter name	Description	Input or output
file	SCCB controller handle	Input
name	Specify the path to access the device	Input
slave_address	Slave address	Input
reg_address_width	Register address width	Input

14.3.1.4 Return value

SCCB device handle.

14.3.2 sccb_dev_read_byte

14.3.2.1 Description

Read a byte from the SCCB device.

14.3.2.2 Function prototype

```
uint8_t sccb_dev_read_byte(handle_t file, uint16_t reg_address);
```

14.3.2.3 Parameter

Parameter name	Description	Input or output
file	SCCB device handle	Input
reg_address	Register address	Input

14.3.2.4 Return value

The byte read.

14.3.3 sccb_dev_write_byte

14.3.3.1 Description

Write a byte to the SCCB device.

14.3.3.2 Function prototype

```
void sccb_dev_write_byte(handle_t file, uint16_t reg_address, uint8_t value);
```

14.3.3.3 Parameter

Parameter name	Description	Input or output
file	SCCB device handle	Input
reg_address	Register address	Input
value	The byte to be written	Input

14.3.3.4 Return value

None.

14.3.4 Example

```
handle_t sccb = io_open("/dev/sccb0");
handle_t dev0 = sccb_get_device(sccb, "/dev/sccb0/dev0", 0x60, 8);

sccb_dev_write_byte(dev0, 0xFF, 0);
uint8_t value = sccb_dev_read_byte(dev0, 0xFF);
```

Chapter 15

TIMER

15.1 Overview

The timer peripheral provides high-precision timing. The chip has 3 timers, each with 4 channels. Can be configured as PWM, see PWM description for details.

15.2 Features

The TIMER module has the following features:

- Enable or disable the timer
- Configure timer trigger interval
- Configure timer trigger handler

15.3 API

Corresponding header file `devices.h`
Provide the following interfaces

- timer_set_interval
- timer_set_on_tick
- timer_set_enable

15.3.1 timer_set_interval

15.3.1.1 Description

Set the timer trigger interval.

15.3.1.2 Function prototype

```
size_t timer_set_interval(handle_t file, size_t nanoseconds);
```

15.3.1.3 Parameter

Parameter name	Description	Input or output
file	TIMER device handle	Input
nanoseconds	Interval (nanoseconds)	Input

15.3.1.4 Return value

Actual trigger interval (nanoseconds).

15.3.2 timer_set_on_tick

15.3.2.1 Description

Set the handler when the timer is triggered.

15.3.2.2 Function prototype

```
void timer_set_on_tick(handle_t file, timer_on_tick_t on_tick, void *userdata);
```

15.3.2.3 Parameter

Parameter name	Description	Input or output
file	TIMER device handle	Input
on_tick	Handler	Input
userdata	Handler user data	Input

15.3.2.4 Return value

None.

15.3.3 timer_set_enable

15.3.3.1 Description

Enable or disable the timer.

15.3.3.2 Function prototype

```
void timer_set_enable(handle_t file, bool enable);
```

15.3.3.3 Parameter

Parameter name	Description	Input or output
file	TIMER device handle	Input
enable	Whether to enable	Input

15.3.3.4 Return value

None.

15.3.4 Example

```
/* Timer 0 Channel 0 timed 1 second print "Time OK!" */
void on_tick(void *unused)
{
    printf("Time OK!\n");
}

handle_t timer = io_open("/dev/timer0");

timer_set_interval(timer, 1e9);
timer_set_on_tick(timer, on_tick, NULL);
timer_set_enable(timer, true);
```

15.4 Data type

The relevant data types and data structures are defined as follows:

- timer_on_tick_t: The handler when the timer is triggered.

15.4.1 timer_on_tick_t

15.4.1.1 Description

The handler when the timer is triggered.

15.4.1.2 Type definition

```
typedef void (*timer_on_tick_t)(void *userdata);
```

15.4.1.3 Parameter

Parameter name	Description	Input or output
userdata	User data	Input

Chapter 16

PWM

16.1 Overview

A pulse width modulator (PWM) is used to control the duty cycle of the pulse output. It is essentially a timer, so be careful not to conflict with the timer when setting the PWM number and channel.

16.2 Features

The PWM module has the following features:

- Configure the PWM output frequency
- Configure the output duty cycle of each pin of the PWM

16.3 API

Corresponding header file `devices.h`
Provide the following interfaces

- `pwm_get_pin_count`
- `pwm_set_frequency`
- `pwm_set_active_duty_cycle_percentage`
- `pwm_set_enable`

16.3.1 pwm_get_pin_count

16.3.1.1 Description

Get the number of PWM pins.

16.3.1.2 Function prototype

```
uint32_t pwm_get_pin_count(handle_t file);
```

16.3.1.3 Parameter

Parameter name	Description	Input or output
file	PWM device handle	Input

16.3.1.4 Return value

The number of PWM pins.

16.3.2 pwm_set_frequency

16.3.2.1 Description

Set the PWM frequency.

16.3.2.2 Function prototype

```
double pwm_set_frequency(handle_t file, double frequency);
```

16.3.2.3 Parameter

Parameter name	Description	Input or output
file	PWM device handle	Input
frequency	Expected frequency (Hz)	Input

16.3.2.4 Return value

The actual frequency (Hz) after setting.

16.3.3 pwm_set_active_duty_cycle_percentage

16.3.3.1 Description

Set the PWM pin duty cycle.

16.3.3.2 Function prototype

```
double pwm_set_active_duty_cycle_percentage(handle_t file, uint32_t pin, double
    duty_cycle_percentage);
```

16.3.3.3 Parameter

Parameter name	Description	Input or output
file	PWM device handle	Input
pin	Pin number	Input
duty_cycle_percentage	Expected duty cycle	Input

16.3.3.4 Return value

The actual duty cycle after setting.

16.3.4 pwm_set_enable

16.3.4.1 Description

Set whether the PWM pin is enabled.

16.3.4.2 Function prototype

```
void pwm_set_enable(handle_t file, uint32_t pin, bool enable);
```

16.3.4.3 Parameter

Parameter name	Description	Input or output
file	PWM device handle	Input
pin	Pin number	Input
enable	Whether to enable	Input

16.3.4.4 Return value

The actual duty cycle after setting.

16.3.5 Example

```
/* pwm0 pin0 output 200KHz square wave with duty cycle of 0.5 */
handle_t pwm = io_open("/dev/pwm0");
pwm_set_frequency(pwm, 200000);
pwm_set_active_duty_cycle_percentage(pwm, 0, 0.5);
pwm_set_enable(pwm, 0, true);
```

Chapter 17

WDT

17.1 Overview

A watchdog timer (WDT) is a hardware timer that automatically generates a system reset if the main program neglects to periodically service it.

17.2 Features

The WDT module has the following features:

- Configure timeout
- Manually set the restart time
- Configure to reset or enter interrupt after timeout
- Clear the interrupt after entering the interrupt to cancel the reset, otherwise wait for the second timeout after reset

17.3 API

Corresponding header file `devices.h`
Provide the following interfaces

- wdt_set_response_mode
- wdt_set_timeout
- wdt_set_on_timeout
- wdt_restart_counter
- wdt_set_enable

17.3.1 wdt_set_response_mode

17.3.1.1 Description

Set the WDT response mode.

17.3.1.2 Function prototype

```
void wdt_set_response_mode(handle_t file, wdt_response_mode_t mode);
```

17.3.1.3 Parameter

Parameter name	Description	Input or output
file	WDT device handle	Input
mode	Response mode	Input

17.3.1.4 Return value

None.

17.3.2 wdt_set_timeout

17.3.2.1 Description

Set the WDT timeout period.

17.3.2.2 Function prototype

```
size_t wdt_set_timeout(handle_t file, size_t nanoseconds);
```

17.3.2.3 Parameter

Parameter name	Description	Input or output
file	WDT device handle	Input
nanoseconds	Expected timeout (nanoseconds)	Input

17.3.2.4 Return value

The actual timeout (nanoseconds) after setting.

17.3.3 wdt_set_on_timeout

17.3.3.1 Description

Set the WDT timeout handler.

17.3.3.2 Function prototype

```
void wdt_set_on_timeout(handle_t file, wdt_on_timeout_t handler, void *userdata);
```

17.3.3.3 Parameter

Parameter name	Description	Input or output
file	WDT device handle	Input
handler	Handler	Input
userdata	Handler user data	Input

17.3.3.4 Return value

None.

17.3.4 wdt_restart_counter

17.3.4.1 Description

Let the WDT restart counting.

17.3.4.2 Function prototype

```
void wdt_restart_counter(handle_t file);
```

17.3.4.3 Parameter

Parameter name	Description	Input or output
file	WDT device handle	Input

17.3.4.4 Return value

None.

17.3.5 wdt_set_enable

17.3.5.1 Description

Set whether WDT is enabled.

17.3.5.2 Function prototype

```
void wdt_set_enable(handle_t file, bool enable);
```

17.3.5.3 Parameter

Parameter name	Description	Input or output
file	WDT device handle	Input
enable	Whether to enable	Input

17.3.5.4 Return value

None.

17.3.6 Example

```
/*
 * After 2 seconds, enter the watchdog interrupt function to print Hello_world,
 * and then reset after 2 seconds.
 */
void on_timeout(void *unused)
{
    printf("Timeout\n");
}

handle_t wdt = io_open("/dev/wdt0");

wdt_set_response_mode(wdt, WDT_RESP_INTERRUPT);
wdt_set_timeout(wdt, 2e9);
wdt_set_on_timeout(wdt, on_timeout, NULL);
wdt_set_enable(wdt, true);
```

17.4 Data type

The relevant data types and data structures are defined as follows:

- wdt_response_mode_t: WDT Response mode.
- wdt_on_timeout_t: WDT timeout handler.

17.4.1 wdt_response_mode_t

17.4.1.1 Description

WDT Response mode.

17.4.1.2 Type definition

```
typedef enum _wdt_response_mode
{
    WDT_RESP_RESET,
    WDT_RESP_INTERRUPT
} wdt_response_mode_t;
```

17.4.1.3 Enumeration element

Element name	Description
WDT_RESP_RESET	Reset system after timeout
WDT_RESP_INTERRUPT	Enter the interrupt after timeout, reset the system if it times out again

17.4.2 wdt_on_timeout_t

17.4.2.1 Description

WDT timeout handler.

17.4.2.2 Type definition

```
typedef int (*wdt_on_timeout_t)(void *userdata);
```

17.4.2.3 Parameter

Parameter name	Description	Input or output
userdata	User data	Input

17.4.2.4 Return value

Return value	Description
0	The interrupt is not cleared and the system will reset
1	Clear interrupt, system does not reset

Chapter 18

FFT

18.1 Overview

Fast Fourier Transform (FFT) Accelerator.

The FFT accelerator implements the radix-2 decimation-in-time (DIT) Cooley-Tukey FFT algorithm[*1] acceleration in hardware.

18.2 Features

The FFT accelerator currently supports 64-point, 128-point, 256-point, and 512-point FFTs and IFFTs. Inside the FFT accelerator, there are two SRAMs with a size of 512 * 32 bits. After the configuration is completed, the FFT sends a TX request to the DMA, and the data sent by the DMA is placed in one of the SRAMs until the data volume satisfies the current FFT operation needs, and the FFT operation begins at this point. The butterfly operation unit reads data from the SRAM which containing the valid data, and writes the data to another SRAM after the end of the operation. The next butterfly operation reads the data from the SRAM just written, when the operation ends write to another SRAM. This process alternates this way until the entire FFT operation is completed.

18.3 API

Corresponding header file `fft.h`

Provide the following interfaces

[*1] https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm

- fft_complex_uint16

18.3.1 fft_complex_uint16

18.3.1.1 Description

FFT operation.

18.3.1.2 Function prototype

```
void fft_complex_uint16(uint16_t shift, fft_direction_t direction, const uint64_t *
    input, size_t point_num, uint64_t *output);
```

18.3.1.3 Parameter

Parameter name	Description	Input or output
shift	Data shift setting*2	Input
direction	FFT or IFFT	Input
input	Input data sequence, format is RIRI...*3	Input
point_num	The number of data points to be calculated can only be 512/256/128/64	Input
output	The result after the operation. The format is RIRI...*4	Output

18.3.2 Example

```
#define FFT_N                512U
#define FFT_FORWARD_SHIFT    0x0U
#define FFT_BACKWARD_SHIFT   0x1ffU
#define PI                   3.14159265358979323846
```

*2 The 16-bit register (-32768~32767) of the FFT accelerator may overflow during the operation, and the FFT transform has 9 layers. Shift setting determines which layer needs to be shifted to prevent overflow, such as 0x1ff means that 9 layers are all shifted; 0x03 means The first layer and the second layer are shifted. If it is shifted, the transformed amplitude is not the amplitude of the normal FFT transform. For the corresponding relationship, refer to the fft_test test demo program, they contain examples of solving frequency points, phases, and amplitudes.

*3 The precision of both the real part and the imaginary part is 16 bit.

*4 The precision of both the real part and the imaginary part is 16 bit.

```
for (i = 0; i < FFT_N; i++)
{
    tempf1[0] = 0.3 * cosf(2 * PI * i / FFT_N + PI / 3) * 256;
    tempf1[1] = 0.1 * cosf(16 * 2 * PI * i / FFT_N - PI / 9) * 256;
    tempf1[2] = 0.5 * cosf((19 * 2 * PI * i / FFT_N) + PI / 6) * 256;
    data_hard[i].real = (int16_t)(tempf1[0] + tempf1[1] + tempf1[2] + 10);
    data_hard[i].imag = (int16_t)0;
}
for (int i = 0; i < FFT_N / 2; ++i)
{
    input_data = (fft_data_t *)&buffer_input[i];
    input_data->R1 = data_hard[2 * i].real;
    input_data->I1 = data_hard[2 * i].imag;
    input_data->R2 = data_hard[2 * i + 1].real;
    input_data->I2 = data_hard[2 * i + 1].imag;
}
fft_complex_uint16(FFT_FORWARD_SHIFT, FFT_DIR_FORWARD, buffer_input, FFT_N,
    buffer_output);
for (i = 0; i < FFT_N / 2; i++)
{
    output_data = (fft_data_t*)&buffer_output[i];
    data_hard[2 * i].imag = output_data->I1 ;
    data_hard[2 * i].real = output_data->R1 ;
    data_hard[2 * i + 1].imag = output_data->I2 ;
    data_hard[2 * i + 1].real = output_data->R2 ;
}
for (int i = 0; i < FFT_N / 2; ++i)
{
    input_data = (fft_data_t *)&buffer_input[i];
    input_data->R1 = data_hard[2 * i].real;
    input_data->I1 = data_hard[2 * i].imag;
    input_data->R2 = data_hard[2 * i + 1].real;
    input_data->I2 = data_hard[2 * i + 1].imag;
}
fft_complex_uint16(FFT_BACKWARD_SHIFT, FFT_DIR_BACKWARD, buffer_input, FFT_N,
    buffer_output);
for (i = 0; i < FFT_N / 2; i++)
{
    output_data = (fft_data_t*)&buffer_output[i];
    data_hard[2 * i].imag = output_data->I1 ;
    data_hard[2 * i].real = output_data->R1 ;
    data_hard[2 * i + 1].imag = output_data->I2 ;
    data_hard[2 * i + 1].real = output_data->R2 ;
}
```

18.4 Data type

The relevant data types and data structures are defined as follows:

- fft_data_t: The data format passed in by the FFT operation.

- fft_direction_t: FFT transform mode.

18.4.1 fft_data_t

18.4.1.1 Description

The data format passed in by the FFT operation.

18.4.1.2 Type definition

```
typedef struct tag_fft_data
{
    int16_t I1;
    int16_t R1;
    int16_t I2;
    int16_t R2;
} fft_data_t;
```

18.4.1.3 Enumeration element

Element name	Description
I1	The imaginary part of the first data
R1	The real part of the first data
I2	The imaginary part of the second data
R2	The real part of the second data

18.4.2 fft_direction_t

18.4.2.1 Description

FFT transform mode.

18.4.2.2 Type definition

```
typedef enum tag_fft_direction
{
    FFT_DIR_BACKWARD,
    FFT_DIR_FORWARD,
    FFT_DIR_MAX,
} fft_direction_t;
```

18.4.2.3 Enumeration element

Element name	Description
FFT_DIR_BACKWARD	FFT inverse transform (FFT)
FFT_DIR_FORWARD	FFT positive transform (IFFT)

Chapter 19

SHA256

19.1 Overview

Secure Hash Algorithm (SHA) accelerator supports hardware acceleration of the sha256 algorithm.

19.2 Features

- Supports hardware acceleration of the sha256 algorithm

19.3 API

Corresponding header file `sha256.h`
Provide the following interfaces

- sha256_hard_calculate

19.3.1 sha256_hard_calculate

19.3.1.1 Description

SHA256 hardware accelerated hash for data

19.3.1.2 Function prototype

```
void sha256_hard_calculate(const uint8_t *input, size_t input_len, uint8_t *output);
```

19.3.1.3 Parameter

Parameter name	Description	Input or output
input	Input data to be hashed	Input
input_len	Length of input data	Input
output	SHA256 hash result. The size of buffer must be greater than or equal to 32 bytes	Output

19.3.2 Example

```
uint8_t hash[32];
sha256_hard_calculate((uint8_t *)"abc", 3, hash);
```

Chapter 20

AES

20.1 Overview

Advanced Encryption Standard (AES) acceleration engine. The AES module uses hardware to implement AES operation acceleration.

20.2 Features

K210 have a built-in AES(Advanced Encryption Standard acceleration engine). Compared with software, it can greatly improve the speed of AES operation. The AES accelerator supports multiple encryption/decryption modes (ECB, CBC, GCM) and multiple length of KEY (128, 192, 256).

20.3 API

Corresponding header file `aes.h`
Provide the following interfaces

- aes_ecb128_hard_encrypt
- aes_ecb128_hard_decrypt
- aes_ecb192_hard_encrypt
- aes_ecb192_hard_decrypt
- aes_ecb256_hard_encrypt
- aes_ecb256_hard_decrypt
- aes_cbc128_hard_encrypt
- aes_cbc128_hard_decrypt

- aes_cbc192_hard_encrypt
- aes_cbc192_hard_decrypt
- aes_cbc256_hard_encrypt
- aes_cbc256_hard_decrypt
- aes_gcm128_hard_encrypt
- aes_gcm128_hard_decrypt
- aes_gcm192_hard_encrypt
- aes_gcm192_hard_decrypt
- aes_gcm256_hard_encrypt
- aes_gcm256_hard_decrypt

20.3.1 aes_ecb128_hard_encrypt

20.3.1.1 Description

AES-ECB-128 encryption operation, when the amount of encrypted data is less than or equal to 896 bytes, it will use the CPU to transmit data. When it is greater than 896 bytes, it will use DMA to transmit data, thus improving the calculation efficiency.

20.3.1.2 Function prototype

```
void aes_ecb128_hard_encrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len,
     uint8_t *output_data)
```

20.3.1.3 Parameter

Parameter name	Description	Input or output
input_key	AES-ECB-128 encryption key	Input
input_data	AES-ECB-128 plaintext data to be encrypted	Input
input_len	AES-ECB-128 length of plaintext data to be encrypted	Input

Parameter name	Description	Input or output
output_data	The result of the AES-ECB-128 encryption operation is stored in this buffer*1.	Output

20.3.1.4 Return value

None.

20.3.2 aes_ecb128_hard_decrypt

20.3.2.1 Description

AES-ECB-128 decryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.2.2 Function prototype

```
void aes_ecb128_hard_decrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len,
    uint8_t *output_data)
```

20.3.2.3 Parameter

Parameter name	Description	Input or output
input_key	AES-ECB-128 decryption key	Input
input_data	AES-ECB-128 ciphertext data to be decrypted	Input
input_len	AES-ECB-128 length of ciphertext data to be decrypted	Input

*1 This buffer size needs to be at least an integer multiple of 16bytes.

Parameter name	Description	Input or output
output_data	The result of the AES-ECB-128 decryption operation is stored in this buffer*2.	Output

20.3.2.4 Return value

None.

20.3.3 aes_ecb192_hard_encrypt

20.3.3.1 Description

AES-ECB-192 encryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.3.2 Function prototype

```
void aes_ecb192_hard_encrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len,
     uint8_t *output_data)
```

20.3.3.3 Parameter

Parameter name	Description	Input or output
input_key	AES-ECB-192 encryption key	Input
input_data	AES-ECB-192 plaintext data to be encrypted	Input
input_len	AES-ECB-192 length of plaintext data to be encrypted	Input

*2 This buffer size needs to be at least an integer multiple of 16bytes.

Parameter name	Description	Input or output
output_data	The result of the AES-ECB-192 encryption operation is stored in this buffer*3.	Output

20.3.3.4 Return value

None.

20.3.4 aes_ecb192_hard_decrypt

20.3.4.1 Description

AES-ECB-192 decryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.4.2 Function prototype

```
void aes_ecb192_hard_decrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len,
     uint8_t *output_data)
```

20.3.4.3 Parameter

Parameter name	Description	Input or output
input_key	AES-ECB-192 decryption key	Input
input_data	AES-ECB-192 ciphertext data to be decrypted	Input
input_len	AES-ECB-192 length of ciphertext data to be decrypted	Input

*3 This buffer size needs to be at least an integer multiple of 16bytes.

Parameter name	Description	Input or output
output_data	The result of the AES-ECB-192 decryption operation is stored in this buffer*4.	Output

20.3.4.4 Return value

None.

20.3.5 aes_ecb256_hard_encrypt

20.3.5.1 Description

AES-ECB-256 encryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.5.2 Function prototype

```
void aes_ecb256_hard_encrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len,
     uint8_t *output_data)
```

20.3.5.3 Parameter

Parameter name	Description	Input or output
input_key	AES-ECB-256 encryption key	Input
input_data	AES-ECB-256 plaintext data to be encrypted	Input
input_len	AES-ECB-256 length of plaintext data to be encrypted	Input

*4 This buffer size needs to be at least an integer multiple of 16bytes.

Parameter name	Description	Input or output
output_data	The result of the AES-ECB-256 encryption operation is stored in this buffer*5.	Output

20.3.5.4 Return value

None.

20.3.6 aes_ecb256_hard_decrypt

20.3.6.1 Description

AES-ECB-256 decryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.6.2 Function prototype

```
void aes_ecb256_hard_decrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len,
     uint8_t *output_data)
```

20.3.6.3 Parameter

Parameter name	Description	Input or output
input_key	AES-ECB-256 decryption key	Input
input_data	AES-ECB-256 ciphertext data to be decrypted	Input
input_len	AES-ECB-256 length of ciphertext data to be decrypted	Input

*5 This buffer size needs to be at least an integer multiple of 16bytes.

Parameter name	Description	Input or output
output_data	The result of the AES-ECB-256 decryption operation is stored in this buffer*6.	Output

20.3.6.4 Return value

None.

20.3.7 aes_cbc128_hard_encrypt

20.3.7.1 Description

AES-CBC-128 encryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.7.2 Function prototype

```
void aes_cbc128_hard_encrypt(cbc_context_t *context, uint8_t *input_data, size_t
    input_len, uint8_t *output_data)
```

20.3.7.3 Parameter

Parameter name	Description	Input or output
context	AES-CBC-128 encryption operation structure containing encryption key and offset vector	Input
input_data	AES-CBC-128 plaintext data to be encrypted	Input
input_len	AES-CBC-128 length of plaintext data to be encrypted	Input

*6 This buffer size needs to be at least an integer multiple of 16bytes.

Parameter name	Description	Input or output
output_data	The result of the AES-CBC-128 encryption operation is stored in this buffer*7.	Output

20.3.7.4 Return value

None.

20.3.8 aes_cbc128_hard_decrypt

20.3.8.1 Description

AES-CBC-128 decryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.8.2 Function prototype

```
void aes_cbc128_hard_decrypt(cbc_context_t *context, uint8_t *input_data, size_t
    input_len, uint8_t *output_data)
```

20.3.8.3 Parameter

Parameter name	Description	Input or output
context	AES-CBC-128 decryption operation structure, including decryption key and offset vector	Input
input_data	AES-CBC-128 ciphertext data to be decrypted	Input
input_len	AES-CBC-128 length of ciphertext data to be decrypted	Input

*7 This buffer size needs to be at least an integer multiple of 16bytes.

Parameter name	Description	Input or output
output_data	The result of the AES-CBC-128 decryption operation is stored in this buffer*8.	Output

20.3.8.4 Return value

None.

20.3.9 aes_cbc192_hard_encrypt

20.3.9.1 Description

AES-CBC-192 encryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.9.2 Function prototype

```
void aes_cbc192_hard_encrypt(cbc_context_t *context, uint8_t *input_data, size_t
    input_len, uint8_t *output_data)
```

20.3.9.3 Parameter

Parameter name	Description	Input or output
context	AES-CBC-192 encryption operation structure containing encryption key and offset vector	Input
input_data	AES-CBC-192 plaintext data to be encrypted	Input
input_len	AES-CBC-192 length of plaintext data to be encrypted	Input

*8 This buffer size needs to be at least an integer multiple of 16bytes.

Parameter name	Description	Input or output
output_data	The result of the AES-CBC-192 encryption operation is stored in this buffer*9.	Output

20.3.9.4 Return value

None.

20.3.10 aes_cbc192_hard_decrypt

20.3.10.1 Description

AES-CBC-192 decryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.10.2 Function prototype

```
void aes_cbc192_hard_decrypt(cbc_context_t *context, uint8_t *input_data, size_t
    input_len, uint8_t *output_data)
```

20.3.10.3 Parameter

Parameter name	Description	Input or output
context	AES-CBC-192 decryption operation structure, including decryption key and offset vector	Input
input_data	AES-CBC-192 ciphertext data to be decrypted	Input
input_len	AES-CBC-192 length of ciphertext data to be decrypted	Input

*9 This buffer size needs to be at least an integer multiple of 16bytes.

Parameter name	Description	Input or output
output_data	The result of the AES-CBC-192 decryption operation is stored in this buffer*10.	Output

20.3.10.4 Return value

None.

20.3.11 aes_cbc256_hard_encrypt

20.3.11.1 Description

AES-CBC-256 encryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.11.2 Function prototype

```
void aes_cbc256_hard_encrypt(cbc_context_t *context, uint8_t *input_data, size_t
    input_len, uint8_t *output_data)
```

20.3.11.3 Parameter

Parameter name	Description	Input or output
context	AES-CBC-256 encryption operation structure containing encryption key and offset vector	Input
input_data	AES-CBC-256 plaintext data to be encrypted	Input
input_len	AES-CBC-256 length of plaintext data to be encrypted	Input

*10 This buffer size needs to be at least an integer multiple of 16bytes.

Parameter name	Description	Input or output
output_data	The result of the AES-CBC-256 encryption operation is stored in this buffer*11.	Output

20.3.11.4 Return value

None.

20.3.12 aes_cbc256_hard_decrypt

20.3.12.1 Description

AES-CBC-256 decryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.12.2 Function prototype

```
void aes_cbc256_hard_decrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len,
     uint8_t *output_data)
```

20.3.12.3 Parameter

Parameter name	Description	Input or output
context	AES-CBC-256 decryption operation structure, including decryption key and offset vector	Input
input_data	AES-CBC-256 ciphertext data to be decrypted	Input
input_len	AES-CBC-256 length of ciphertext data to be decrypted	Input

*11 This buffer size needs to be at least an integer multiple of 16bytes.

Parameter name	Description	Input or output
output_data	The result of the AES-CBC-256 decryption operation is stored in this buffer*12.	Output

20.3.12.4 Return value

None.

20.3.13 aes_gcm128_hard_encrypt

20.3.13.1 Description

AES-GCM-128 encryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.13.2 Function prototype

```
void aes_gcm128_hard_encrypt(gcm_context_t *context, uint8_t *input_data, size_t
    input_len, uint8_t *output_data, uint8_t *gcm_tag)
```

20.3.13.3 Parameter

Parameter name	Description	Input or output
context	The structure of the AES-GCM-128 encryption operation, including the encryption key / offset vector / aad / aad length	Input
input_data	AES-GCM-128 plaintext data to be encrypted	Input
input_len	AES-GCM-128 length of plaintext data to be encrypted	Input

*12 This buffer size needs to be at least an integer multiple of 16bytes.

Parameter name	Description	Input or output
output_data	The result of the AES-GCM-128 encryption operation is stored in this buffer*13.	Output
gcm_tag	The tag after the AES-GCM-128 encryption operation is stored in this buffer*14.	Output

20.3.13.4 Return value

None.

20.3.14 aes_gcm128_hard_decrypt

20.3.14.1 Description

AES-GCM-128 decryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.14.2 Function prototype

```
void aes_gcm128_hard_decrypt(gcm_context_t *context, uint8_t *input_data, size_t
    input_len, uint8_t *output_data, uint8_t *gcm_tag)
```

20.3.14.3 Parameter

Parameter name	Description	Input or output
context	The structure of the AES-GCM-128 decryption operation, including the decryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-128 ciphertext data to be decrypted	Input

*13 Since the minimum granularity of DMA handling data is 4bytes, therefore, you need to ensure that the buffer size is at least an integer multiple of 4 bytes.

*14 This buffer size needs to be determined to be 16bytes.

Parameter name	Description	Input or output
input_len	AES-GCM-128 length of ciphertext data to be decrypted。	Input
output_data	The result of the AES-GCM-128 decryption operation is stored in this buffer*15.	Output
gcm_tag	The tag after the AES-GCM-128 decryption operation is stored in this buffer*16.	Output

20.3.14.4 Return value

None.

20.3.15 aes_gcm192_hard_encrypt

20.3.15.1 Description

AES-GCM-192 encryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.15.2 Function prototype

```
void aes_gcm192_hard_encrypt(gcm_context_t *context, uint8_t *input_data, size_t
    input_len, uint8_t *output_data, uint8_t *gcm_tag)
```

20.3.15.3 Parameter

Parameter name	Description	Input or output
context	The structure of the AES-GCM-192 encryption operation, including the encryption key / offset vector / aad / aad length	Input

*15 Since the minimum granularity of DMA handling data is 4bytes, therefore, you need to ensure that the buffer size is at least an integer multiple of 4 bytes.

*16 This buffer size needs to be determined to be 16bytes.

Parameter name	Description	Input or output
input_data	AES-GCM-192 plaintext data to be encrypted	Input
input_len	AES-GCM-192 length of plaintext data to be encrypted。	Input
output_data	The result of the AES-GCM-192 encryption operation is stored in this buffer*17.	Output
gcm_tag	The tag after the AES-GCM-192 encryption operation is stored in this buffer*18.	Output

20.3.15.4 Return value

None.

20.3.16 aes_gcm192_hard_decrypt

20.3.16.1 Description

AES-GCM-192 decryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.16.2 Function prototype

```
void aes_gcm192_hard_decrypt(gcm_context_t *context, uint8_t *input_data, size_t
    input_len, uint8_t *output_data, uint8_t *gcm_tag)
```

20.3.16.3 Parameter

*17 Since the minimum granularity of DMA handling data is 4bytes, therefore, you need to ensure that the buffer size is at least an integer multiple of 4 bytes.

*18 This buffer size needs to be determined to be 16bytes.

Parameter name	Description	Input or output
context	The structure of the AES-GCM-192 decryption operation, including the decryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-192 ciphertext data to be decrypted	Input
input_len	AES-GCM-192 length of ciphertext data to be decrypted。	Input
output_data	The result of the AES-GCM-192 decryption operation is stored in this buffer[*19].	Output
gcm_tag	The tag after the AES-GCM-192 decryption operation is stored in this buffer[*20].	Output

20.3.16.4 Return value

None.

20.3.17 aes_gcm256_hard_encrypt

20.3.17.1 Description

AES-GCM-256 encryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.17.2 Function prototype

```
void aes_gcm256_hard_encrypt(gcm_context_t *context, uint8_t *input_data, size_t
    input_len, uint8_t *output_data, uint8_t *gcm_tag)
```

20.3.17.3 Parameter

[*19] Since the minimum granularity of DMA handling data is 4bytes, therefore, you need to ensure that the buffer size is at least an integer multiple of 4 bytes.

[*20] This buffer size needs to be determined to be 16bytes.

Parameter name	Description	Input or output
context	The structure of the AES-GCM-256 encryption operation, including the encryption key / offset vector / aad / aad length	Input
input_data	AES-GCM-256 plaintext data to be encrypted	Input
input_len	AES-GCM-256 length of plaintext data to be encrypted。	Input
output_data	The result of the AES-GCM-256 encryption operation is stored in this buffer*21.	Output
gcm_tag	The tag after the AES-GCM-256 encryption operation is stored in this buffer*22.	Output

20.3.17.4 Return value

None.

20.3.18 aes_gcm256_hard_decrypt

20.3.18.1 Description

AES-GCM-256 decryption operation. When the amount of encrypted data is less than or equal to 896 bytes, cpu is used to transmit data. When it is greater than 896 bytes, dma is used to transmit data, thereby improving the efficiency of calculation.

20.3.18.2 Function prototype

```
void aes_gcm256_hard_decrypt(gcm_context_t *context, uint8_t *input_data, size_t
    input_len, uint8_t *output_data, uint8_t *gcm_tag)
```

20.3.18.3 Parameter

*21 Since the minimum granularity of DMA handling data is 4bytes, therefore, you need to ensure that the buffer size is at least an integer multiple of 4 bytes.

*22 This buffer size needs to be determined to be 16bytes.

Parameter name	Description	Input or output
context	The structure of the AES-GCM-256 decryption operation, including the decryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-256 ciphertext data to be decrypted	Input
input_len	AES-GCM-256 length of ciphertext data to be decrypted。	Input
output_data	The result of the AES-GCM-256 decryption operation is stored in this buffer[*23].	Output
gcm_tag	The tag after the AES-GCM-256 decryption operation is stored in this buffer[*24].	Output

20.3.18.4 Return value

None.

20.3.19 Example

```
cbc_context_t cbc_context;
cbc_context.input_key = cbc_key;
cbc_context.iv = cbc_iv;
aes_cbc128_hard_encrypt(&cbc_context, aes_input_data, 16L, aes_output_data);
memcpy(aes_input_data, aes_output_data, 16L);
aes_cbc128_hard_decrypt(&cbc_context, aes_input_data, 16L, aes_output_data);
```

20.4 Data type

The relevant data types and data structures are defined as follows:

- aes_cipher_mode_t: AES encryption and decryption mode

[*23] Since the minimum granularity of DMA handling data is 4bytes, therefore, you need to ensure that the buffer size is at least an integer multiple of 4 bytes.

[*24] This buffer size needs to be determined to be 16bytes.

20.4.1 aes_cipher_mode_t

20.4.1.1 Description

AES encryption and decryption mode

20.4.1.2 Type definition

```
typedef enum _aes_cipher_mode
{
    AES_ECB = 0,
    AES_CBC = 1,
    AES_GCM = 2,
    AES_CIPHER_MAX
} aes_cipher_mode_t;
```

20.4.1.3 Enumeration element

Element name	Description
AES_ECB	ECB encryption and decryption
AES_CBC	CBC encryption and decryption
AES_GCM	GCM encryption and decryption