K210

FreeRTOS SDK Programming Guide

Translate by Sipeed

KENDRYTE

Jia Nan Technology
KENDRYTE.COM

About this manual

This document provides users with programming guidelines based on the FreeRTOS SDK development.

Corresponding to the sdk version

Kendryte FreeRTOS SDK v0.4.0 (9c7b0e0d23e46e87a2bfd4dd86d1a1f0d3c899e9)

Release notes

date	version	Release notes
2018-10-12	V0.1.0	initial version

Disclaimer

The information in this document, including the referenced url address, is subject to change without notice. The Documentation is provided "as is" without warranty of any kind, including any warranties of merchantability, fitness for a particular purpose, or non-infringement, and any warranties referred to elsewhere by any proposal, specification or sample. This document is not responsible for any infringement of any patent rights arising out of the use of the information in this document. No license, express or implied, by estoppel or otherwise, is hereby granted. All trademark names, trademarks and registered trademarks mentioned are the property of their respective owners and are hereby acknowledged.

Copyright notice

Copyright © 2018 Jia Nan Technology. all rights reserved.

table of Contents

Chapter 1

FreeRTOS Expansion

1.1 Overview

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

1.2 Functional description

The FreeRTOS extension has the following features:

- Get the logical processor ID of the current task
- Create a task at a specified logical processor

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

1.3 Api reference

Corresponding header file task.h

Provide users with 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 Parameters

parameter name	description	input Output
uxProcessor	Logical processor Id	Input
pxTaskCode	Task entry point	Input
pcName	mission name	Input
usStackDepth	Stack space	Input
pvParameters	parameter	Input
uxPriority	priority	Input
pxCreatedTask	Created task handle	Input

1.3.2.4 Return value

return value	description
pdPASS	Succ ess other fail ure

chapter 2

Device List

path	Types of	Remarks
/dev/uart1	UART	
/dev/uart2	UART	
/dev/uart3	UART	
/dev/gpio0	GPIO	High speed 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	Types of	Remarks
/dev/timer3	TIMER	Not available /dev/pwm0 is used at the same time
/dev/timer4	TIMER	Not available /dev/pwm1 is used at the same time
/dev/timer5	TIMER	Not available /dev/pwm1 is used at the same time
/dev/timer6	TIMER	Not available /dev/pwm1 is used at the same time
/dev/timer7	TIMER	Not available /dev/pwm1 is used at the same time
/dev/timer8	TIMER	Not available /dev/pwm2 is used at the same time
/dev/timer9	TIMER	Not available /dev/pwm2 is used at the same time
/dev/timer10	TIMER	Not available /dev/pwm2 is used at the same time
/dev/timer11	TIMER	Not available /dev/pwm2 is used at the same time
/dev/pwm0	PWM	Not available /dev/timer[0-3] Use at the same time
/dev/pwm1	PWM	Not available /dev/timer[4-7] Also use
/dev/pwm2	PWM	Not available /dev/timer[8-11] Simultaneous use
/dev/wdt0	WDT	
/dev/wdt1	WDT	
/dev/rtc0	RTC	

Chapter 3

Pin configuration

3.1 Overview

The pin configuration includes fpioa and power domain configuration.

3.2 Functional description

- Support for io's programmable function selection
- Configuring the power domain

3.3 type of data

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 value.
- Power_bank_item_t: A 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 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 I2S_RECEIVER */
    FUNC_UARTHS_TX      = 19, /*!< UART High speed I2S_TRANSMITTER */
    FUNC_RESV6          = 20, /*!< Reser ved function */
    FUNC_RESV7          = 21, /*!< Reser ved function */
    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 I2S_RECEIVER */
    FUNC_UART1_TX          = 65 , /*!< UART1 I2S_TRANSMITTER */
    FUNC_UART2_RX          = 66 , /*!< UART2 I2S_RECEIVER */
    FUNC_UART2_TX          = 67 , /*!< UART2 I2S_TRANSMITTER */
    FUNC_UART3_RX          = 68 , /*!< UART3 I2S_RECEIVER */
    FUNC_UART3_TX          = 69 , /*!< UART3 I2S_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 */ F
    UNC_I2S0_IN_D1         =  91 , /*!< I2S0 Serial Data Input 1 */ F
    UNC_I2S0_IN_D2         =  92 , /*!< I2S0 Serial Data Input 2 */ F
    UNC_I2S0_IN_D3         =  93 , /*!< I2S0 Serial Data Input 3 */ F
    UNC_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_PWDN        = 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  I2S_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  I2S_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  I2S_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 member

Member name	description
FUNC_JTAG_TCLK	Jtag clock interface
FUNC_JTAG_TDI	Jtag data input
interface func_jtag_tms	Jtag controls
the conversion of the tap state machine	
func_jtag_tdo	Jtag data output
interface	
FUNC_SPI0_D0	Spi0 data line 0
FUNC_SPI0_D1	Spi0 data line 1
FUNC_SPI0_D2	Spi0 data line 2
FUNC_SPI0_D3	Spi0 data line 3
FUNC_SPI0_D4	Spi0 data line 4
FUNC_SPI0_D5	Spi0 data line 5
FUNC_SPI0_D6	Spi0 data line 6
FUNC_SPI0_D7	Spi0 data line 7
FUNC_SPI0_SS0	Spi0 chip select signal 0
FUNC_SPI0_SS1	Spi0 chip select signal 1
FUNC_SPI0_SS2	Spi0 chip select signal 2
FUNC_SPI0_SS3	Spi0 chip select signal 3
FUNC_SPI0_ARB	Spi0 arbitration signal
FUNC_SPI0_SCLK	Spi0 clock
FUNC_UARTHS_RX	Uart high speed receiving data interface
FUNC_UARTHS_TX	Uart high speed send data interface
FUNC_RESV6	Reserved function
FUNC_RESV7	Reserved function
FUNC_CLK_SPI1	Spi1 clock
FUNC_CLK_I2C1	I2c1 clock
FUNC_GPIOHS0	High speed gpio0
FUNC_GPIOHS1	High speed gpio1
FUNC_GPIOHS2	High speed gpio2
FUNC_GPIOHS3	High speed gpio3
FUNC_GPIOHS4	High speed gpio4

Member name	description
FUNC_GPIOHS5	High speed gpio5
FUNC_GPIOHS6	High speed gpio6
FUNC_GPIOHS7	High speed gpio7
FUNC_GPIOHS8	High speed gpio8
FUNC_GPIOHS9	High speed gpio9
FUNC_GPIOHS10	High speed gpio10
FUNC_GPIOHS11	High speed gpio11
FUNC_GPIOHS12	High speed gpio12
FUNC_GPIOHS13	High speed gpio13
FUNC_GPIOHS14	High speed gpio14
FUNC_GPIOHS15	High speed gpio15
FUNC_GPIOHS16	High speed gpio16
FUNC_GPIOHS17	High speed gpio17
FUNC_GPIOHS18	High speed gpio18
FUNC_GPIOHS19	High speed gpio19
FUNC_GPIOHS20	High speed gpio20
FUNC_GPIOHS21	High speed gpio21
FUNC_GPIOHS22	High speed gpio22
FUNC_GPIOHS23	High speed gpio23
FUNC_GPIOHS24	High speed gpio24
FUNC_GPIOHS25	High speed gpio25
FUNC_GPIOHS26	High speed gpio26
FUNC_GPIOHS27	High speed gpio27
FUNC_GPIOHS28	High speed gpio28
FUNC_GPIOHS29	High speed gpio29
FUNC_GPIOHS30	High speed gpio30
FUNC_GPIOHS31	High speed gpio31
FUNC_GPIO0	GPIO0
FUNC_GPIO1	GPIO1
FUNC_GPIO2	GPIO2
FUNC_GPIO3	GPIO3
FUNC_GPIO4	GPIO4
FUNC_GPIO5	GPIO5
FUNC_GPIO6	GPIO6
FUNC_GPIO7	GPIO7

Member name	description
FUNC_UART1_RX	Uart1 receive data interface
FUNC_UART1_TX	Uart1 send data interface
FUNC_UART2_RX	Uart2 receive data interface
FUNC_UART2_TX	Uart2 send data interface
FUNC_UART3_RX	Uart3 receive data interface
FUNC_UART3_TX	Uart3 send data interface
FUNC_SPI1_D0	Spi1 data line 0
FUNC_SPI1_D1	Spi1 data line 1
FUNC_SPI1_D2	Spi1 data line 2
FUNC_SPI1_D3	Spi1 data line 3
FUNC_SPI1_D4	Spi1 data line 4
FUNC_SPI1_D5	Spi1 data line 5
FUNC_SPI1_D6	Spi1 data line 6
FUNC_SPI1_D7	Spi1 data line 7
FUNC_SPI1_SS0	Spi1 chip select signal 0
FUNC_SPI1_SS1	Spi1 chip select signal 1
FUNC_SPI1_SS2	Spi1 chip select signal 2
FUNC_SPI1_SS3	Spi1 chip select signal 3
FUNC_SPI1_ARB	Spi1 arbitration signal
FUNC_SPI1_SCLK	Spi1 clock
Func_spi_slave_d0	spi slave mode data line 0
func_spi_slave_ss	spi slave mode chip select signal
func_spi_slave_sclk	spi slave mode clock
FUNC_I2S0_MCLK	I2s0 master clock (system clock)
func_i2s0_sclk	I2s0 serial clock (bit clock)
func_i2s0_ws	I2s0 frame clock
FUNC_I2S0_IN_D0	I2s0 serial input data interface 0
func_i2s0_in_d1	I2s0 serial input data interface 1
func_i2s0_in_d2	I2s0 serial input data interface 2
func_i2s0_in_d3	I2s0 serial input data interface 3
func_i2s0_out_d0	i2s0 serial output data interface 0
func_i2s0_out_d1	i2s0 serial output data interface 1
func_i2s0_out_d2	i2s0 serial output data interface 2
func_i2s0_out_d3	i2s0 serial output data interface 3
func_i2s1_mclk	I2s1 master clock (system clock)

Member name	description
FUNC_I2S1_SCLK	I2s1 serial clock (bit clock)
FUNC_I2S1_WS	I2s1 frame clock
FUNC_I2S1_IN_D0	I2s1 serial input data interface 0 func_i2s1_in_d1 I2s1 serial input data interface 1 func_i2s1_in_d2 I2s1 serial input data interface 2 func_i2s1_in_d3 I2s1 serial input data interface 3 func_i2s1_out_d0 i2s1 serial output data interface 0 func_i2s1_out_d1 i2s1 serial output data interface 1 func_i2s1_out_d2 i2s1 serial output data interface 2 func_i2s1_out_d3 i2s1 serial output data interface 3 func_i2s2_mclk I2s2 master clock (system clock) func_i2s2_sclk I2s2 serial clock (bit clock) func_i2s2_ws I2s2 frame clock
FUNC_I2S2_IN_D0	I2s2 serial input data interface 0 func_i2s2_in_d1 I2s2 serial input data interface 1 func_i2s2_in_d2 I2s2 serial input data interface 2 func_i2s2_in_d3 I2s2 serial input data interface 3 func_i2s2_out_d0 i2s2 serial output data interface 0 func_i2s2_out_d1 i2s2 serial output data interface 1 func_i2s2_out_d2 i2s2 serial output data interface 2 func_i2s2_out_d3 i2s2 serial output data interface 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
FUNC_I2C0_SDA	I2c0 serial data interface
FUNC_I2C1_SCLK	I2c1 serial clock
FUNC_I2C1_SDA	I2c1 serial data interface
FUNC_I2C2_SCLK	I2c2 serial clock
FUNC_I2C2_SDA	I2c2 serial data interface
FUNC_CMOS_XCLK	Dvp system clock
FUNC_CMOS_RST	Dvp system reset signal

Member name	description
FUNC_CMOS_PWDN	Dvp enable signal
FUNC_CMOS_VSYNC	Dvp field sync
FUNC_CMOS_HREF	Dvp line reference signal
FUNC_CMOS_PCLK	Pixel clock
FUNC_CMOS_D0	Pixel data 0
FUNC_CMOS_D1	Pixel data 1
FUNC_CMOS_D2	Pixel data 2
FUNC_CMOS_D3	Pixel data 3
FUNC_CMOS_D4	Pixel data 4
FUNC_CMOS_D5	Pixel data 5
FUNC_CMOS_D6	Pixel data 6
FUNC_CMOS_D7	Pixel data 7
FUNC_SCCB_SCLK	Sccb clock
FUNC_SCCB_SDA	Sccb serial data
signal func_uart1_cts	Uart1 clears the
send signal func_uart1_dsr	Uart1 data
device preparation signal func_uart1_dcd	
	Uart1 data carrier
detection func_uart1_ri	Uart1
ringing indication	
FUNC_UART1_SIR_IN	Uart1 serial infrared
input signal func_uart1_dtr	Uart1 data
terminal preparation signal func_uart1_rts	
	Uart1 sends request
signal func_uart1_out2	Uart1 user specified
output signal 2 func_uart1_out1	Uart1
user specified output signal 1	
func_uart1_sir_out	Uart1 serial infrared
output signal func_uart1_baud	Uart1 clock
FUNC_UART1_RE	Uart1 receive enable
FUNC_UART1_DE	Uart1 send enable
FUNC_UART1_RS485_EN	Uart1 enable rs485
FUNC_UART2_CTS	Uart2 clears the
send signal func_uart2_dsr	Uart2 data
device preparation signal func_uart2_dcd	
	Uart2 data carrier
detection func_uart2_ri	Uart2
ringing indication	
Func_uart2_sir_in	uart2 serial infrared
input signal func_uart2_dtr	Uart2 data
terminal preparation signal func_uart2_rts	
	Uart2 sends a
request signal	

Member name	description
FUNC_UART2_OUT2	Uart2 user specified output signal 2 func_uart2_out1 Uart2 user specified output signal 1 func_uart2_sir_out Uart2 serial infrared output signal func_uart2_baud Uart2 clock
FUNC_UART2_RE	Uart2 receive enable
FUNC_UART2_DE	Uart2 send enable
FUNC_UART2_RS485_EN	Uart2 enable rs485
FUNC_UART3_CTS	Clear send signal
FUNC_UART3_DSR	Data device preparation signal func_uart3_dcd Uart3 data carrier detection func_uart3_ri Uart3 ringing indication
FUNC_UART3_SIR_IN	Uart3 serial infrared input signal func_uart3_dtr Uart3 data terminal preparation signal func_uart3_rts Uart3 sends request signal func_uart3_out2 Uart3 user specified output signal 2 func_uart3_out1 Uart3 user specified output signal 1 func_uart3_sir_out Uart3 serial infrared output signal func_uart3_baud Uart3 clock
FUNC_UART3_RE	Uart3 receive enable
FUNC_UART3_DE	Uart3 send enable
func_uart3_rs485_en	Uart3 enable rs485 func_timer0_toggle1 Timer0 output signal 1 func_timer0_toggle2 Timer0 output signal 2 func_timer0_toggle3 Timer0 output signal 3 func_timer0_toggle4 Timer0 output signal 4 func_timer1_toggle1 Timer1 output signal 1 func_timer1_toggle2 Timer1 output signal 2 func_timer1_toggle3 Timer1 output signal 3 func_timer1_toggle4 Timer1 output signal 4 func_timer2_toggle1 Timer2 output signal 1 func_timer2_toggle2 Timer2 output signal 2 func_timer2_toggle3 Timer2 output signal 3 func_timer2_toggle4 Timer2 output signal 4 func_clk_spi2 Spi2 clock
FUNC_CLK_I2C2	I2c2 clock

Member name	description
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
FUNC_INTERNAL18	Internal function 18
FUNC_DEBUG0	Debug function 0
FUNC_DEBUG1	Debug function 1
FUNC_DEBUG2	Debugging function 2
FUNC_DEBUG3	Debugging function 3
FUNC_DEBUG4	Debugging function 4
FUNC_DEBUG5	Debugging function 5
FUNC_DEBUG6	Debugging function 6
FUNC_DEBUG7	Debugging function 7
FUNC_DEBUG8	Debugging function 8
FUNC_DEBUG9	Debugging function 9
FUNC_DEBUG10	Debugging function 10
FUNC_DEBUG11	Debugging function 11
FUNC_DEBUG12	Debugging function 12
FUNC_DEBUG13	Debugging function 13
FUNC_DEBUG14	Debugging function 14

Member name	description
FUNC_DEBUG15	Debugging function 15
FUNC_DEBUG16	Debugging function 16
FUNC_DEBUG17	Debugging function 17
FUNC_DEBUG18	Debugging function 18
FUNC_DEBUG19	Debugging function 19
FUNC_DEBUG20	Debugging function 20
FUNC_DEBUG21	Debugging function 21
FUNC_DEBUG22	Debugging function 22
FUNC_DEBUG23	Debugging function 23
FUNC_DEBUG24	Debugging function 24
FUNC_DEBUG25	Debugging function 25
FUNC_DEBUG26	Debugging function 26
FUNC_DEBUG27	Debugging function 27
FUNC_DEBUG28	Debugging function 28
FUNC_DEBUG29	Debugging function 29
FUNC_DEBUG30	Debugging function 30
FUNC_DEBUG31	Debugging function 31

3.3.2 fpioa_cfg_item_t

3.3.2.1 description

Fpioa pin configuration.

3.3.2.2 definition

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

3.3.2.3 Members

Member name	description
number	Pin
function	number
	function
	number

3.3.3 fpioa_cfg_t

3.3.3.1 description

Fpioa configuration.

3.3.3.2 definition

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

3.3.3.3 member

Member name	description
version	Configuration version, must be set to fpioa_cfg_version
functions_count	Number of function configurations
functions	Feature configuration list

3.3.4 sysctl_power_bank_t

3.3.4.1 Desc

ribe the

power domain

number.

3.3.4.2 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 member

Member 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

Io Output voltage value.

3.3.5.2 definition

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

3.3.5.3 member

Member 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 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 member

Member name	description
power_bank	Power domain number
iopowermode	Io output voltage value

3.3.7 power_bank_cfg_t

3.3.7.1 Desc

ribe the power domain configuratio n.

3.3.7.2 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 member

Member name	description
version	Configuration version, must be set to fpioa_cfg_version
powerbankscount	Number of power domain configurations
power_banks	Power domain configuration list

3.3.8 pin_cfg_t

3.3.8.1 Desc

ribe the pin configuratio n.

3.3.8.2 definition

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

```
} pin_cfg_t ;
```

3.3.8.3 member

Member name	description
version	Configuration version, must be set to fpioa_cfg_version
setspi0dvp_data	Whether to set spi0d0-d7 dvpd0-d7 for spi0 or dvp data

input

3.3.9 Example

```
/* Configure io 6 and io7 functions as 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

4.1 Overview

The system control module provides configuration functions for the operating system.

4.2 Functional description

The system control module has the following features:

- Set cpu frequency
- Install a custom driver

4.3 Api reference

Corresponding header file hal.h

Provide users with 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 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 Output
name	Specify the path to access the device	Input
driver	Custom driver implementation	Input

4.3.2.4 The return value is none.

4.3.3 Example

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

4.4 type of data

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

- Driver_base_t: The driver implements the base class.
- Custom_driver_t: Custom driver implementation.

4.4.1 driver_base_t

4.4.1.1 Descri

be the driver
implementation
base class.

4.4.1.2 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 member

Member 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 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 member

Member name	description
base	Driver implementation base class
io_control	Called when control information is received

Chapter 5

Programmable interrupt controller (PIC)

5.1 Overview

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 Functional description

The pic module has the following features:

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

5.3 Api reference

Corresponding header file hal.h

Provide users with 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 Output
irq	Irq number	Input
enable	Whether to enable	Input

5.3.1.4 The return value is none.

5.3.2 pic_set_irq_handler

5.3.2.1 description

Set up 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 Output
irq	Irq number	Input
handler	Handler	Input
userdata	Handler user data	Input

5.3.2.4 The return value is 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 Parameters

parameter name	description	input Output
irq	Irq number	Input
priority	priority	input

5.3.3.4 Return value

no.

5.4 type of data

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 definition

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

5.4.1.3 parameter

parameter name	description	input Output
userdata	User data	Input

Chapter 6

Direct storage access(DMA)

6.1 Overview

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

6.2 Functional description

The dma module has the following features:

- Automatically select an idle dma channel for transmission
- Automatically select software or hardware handshake protocol based on source and destination addresses
- Supports element sizes of 1, 2, 4, and 8 bytes, source and destination 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 reference

Corresponding header file hal.h

Provide users with 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

Turn off the dma device.

6.3.2.2 Function prototype

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

6.3.2.3 parameter

parameter name	description	input Output
file	Dma device handle	Input

6.3.2.4 The return value is 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 Output
file	Dma device handle	Input
request	Request source number	Input

6.3.3.4 The return value is 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, S
    emaphoreHandle_t completion_event);
```

6.3.4.3 parameter

parameter name	description	input Output
file	Dma device handle	Input
src	source address	Input
dest	target address	Output
src_inc	Whether the source address is increasing	Input
dest_inc	Whether the target address is increasing	Input
element_size	Element size (bytes)	Input
count	Number of elements	Input
burst_size	Burst transmission quantity	Input
completion_event	Transfer completion event	Input

6.3.4.4 The

return value

is none.

6.3.5 dma_transmit

6.3.5.1 description

Perform dma synchronous transmission.

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 Output
file	Dma device handle	Input
src	source address	input,
dest	target address	output,
src_inc	Whether the source address is incremented by the target address, whether it is self-incrementing	input,
dest_inc		input,
element_size	element size (bytes)	input,
count	Burst transmission quantity	input
burst_size		

6.3.5.4 return value

no.

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 Output
file	Dma device handle	Input
srcs	Source address list	Input
src_num	Number of source addresses	Input
dests	Destination address list	Output
dest_num	Number of destination addresses	Input
src_inc	Whether the source address is increasing	Input
dest_inc	Whether the target address is increasing	Input
element_size	Element size (bytes)	Input
count	Number of elements	Input
burst_size	Burst transmission quantity	Input
stage_completion_handler	Stage completion handler	Input
stage_completion_handler_data	Stage completion handler user data	Input
completion_event	Transfer completion event	Input
stop_signal	Stop signal	Input

Note: Phase completion refers to the completion of the transfer of a single source to the target count element.

6.3.6.4 The return value is none.

6.3.7 Example

```
int src[256] = { [0 ... 255] = 1 };
int dest[256];
handle_t dma = dm a_ open_ free ();
dma_transmit(dma, src, dest, true, true, sizeof(int), 256, 4);
assert(dest[0] == src[0]);
dm a_close ( dma );
```

6.4 type of data

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

- Dma_stage_completion_handler_t: The DMA stage completes the handler.

6.4.1 dma_stage_completion_handler_t

6.4.1.1 description

The dma stage completes the handler.

6.4.1.2 definition

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

6.4.1.3 parameter

parameter name	description	input 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 Functional description

The standard io module has the following features:

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

7.3 Api reference

Corresponding header file devices.h

Provide users with the following interfaces:

- io_open
- io_close
- io_read
- io_write
- io_control

7.3.1 io_open

7.3.1.1 Descri

ption 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 Output
name	Device path	Input

7.3.1.4 return value

return value	description
0	failure
other	Device handle

7.3.2 io_close

7.3.2.1 Descri

ption Turns

off a device.

7.3.2.2 Function prototype

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

7.3.2.3 parameter

parameter name	description	input Output
file	Device handle	Input

7.3.2.4 return value

return value	description
0	success
other	failure

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 Output
file	Device handle	Input
buffer	Target 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 Output
file	Device handle	Input
buffer	Source buffer	Input

parameter name	description	input Output
len	The number of bytes to write	Input

7.3.4.4 return value

return value	description
len	success
other	failure

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 Parameters

parameter name	description	input Output
file	Device handle	Input
control_code	control code	input,
write_buffer	source buffer	input,
write_len	Number of bytes to write to the target buffer	input and output
read_buffer	Maximum number of bytes read	output
read_len		

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

Universal asynchronous transceiver (UART)

8.1 Overview

Embedded applications typically require a simple method that consumes less system resources to transfer data.Universal Asynchronous Transceiver (uart)

To meet these requirements, it has the flexibility to perform full-duplex data exchange with external devices.

8.2 Functional description

The uart module has the following features:

- Configuring uart parameters
- Automatically collect data into the buffer

8.3 Api reference

Corresponding header file devices.h

Provide users with 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 Output
file	Uart device handle	Input
baud_rate	Baud rate	Input
databits	Data bits (5-8)	Input
stopbits	Stop bit	Input
parity	Check Digit	Input

8.3.1.4 The return value is 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 type of data

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

- Uart_stopbits_t: UART stop bit.
- Uart_parity_t: UART check digit.

8.4.1 uart_stopbits_t

8.4.1.1 description

Uart stop bit.

8.4.1.2 definition

```
typedef enum _uart_ stopbits
{
    UART_STOP_1 ,
    UART_STOP_1_5 ,
    UART_STOP_2
} uart_stopbits_t ;
```

8.4.1.3 member

Member 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 check digit.

8.4.2.2 definition

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

8.4.2.3 Members

Member name	description
UART_PARITY_NONE	No
UART_PARITY_ODD	parity
UART_PARITY_EVEN	check
	parity
	check

Chapter 9

General purpose input/output (GPIO)

9.1 Overview

The chip has 32 high-speed gpio and 8 universal gpio.

9.2 Functional description

The gpio module has the following features:

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

9.3 Api reference

Corresponding header file devices.h

Provide users with 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 number of gpio pins.

9.3.1.2 Function prototype

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

9.3.1.3 parameter

parameter name	description	input Output
file	Gpio controller handle	Input

9.3.1.4 The number of return value pins.

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 Parameters

parameter name	description	input Output
file	Gpio controller handle	Input
pin	pin number	input
mode	Drive mode	input

9.3.2.4 Return value

no.

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 at this time.

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 Output
file	Gpio controller handle	Input
pin	Pin number	Input
edge	Edge trigger mode	Input

9.3.3.4 The return value is 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 at this time.

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 Parameters

parameter name	description	input Output
file	Gpio controller	Inpu
pin	handle pin	t
callback	number	inpu
	Handler	t

parameter name	description	input Output
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 Parameters

parameter name	description	input Output
file	Gpio controller	Inpu
pin	handle pin	t
	number	inpu
		t

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 Output
file	Gpio controller handle	Input

parameter name	description	input Output
pin	Pin number	Input
value	The value to set	Input

9.3.6.4 Return value

no.

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 type of data

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 definition

```
typedef enum _gpio_drive_mode
{
    GPIO_ DM_
    INPUT , GPIO_DM_INP
    UT_PULL_DOWN, GPIO_D
    M_INPUT_PULL_UP, GPIO
```

9.4.1.3 member

Member name	description
GPIO_DM_INPUT	Input
GPIO_DM_INPUT_PULL_DOWN	Input drop 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 definition

```
typedef enum _gpio_pin_edge
{
    GPIO_
    PE_NONE ,
    GPIO_PE_
    FALLING,
```

9.4.2.3 member

Member 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 definition

```
typedef enum _gpio_pin_value
{
    GPIO_PV_L
    OW , GPIO_P
      V_HIGH
```

9.4.3.3 member

Member name	description
GPIO_PV_LOW	low
GPIO_PV_HIGH	high

9.4.4 gpio_on_changed_t

9.4.4.1 description

The gpio edge trigger handler.

9.4.4.2 definition

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

9.4.4.3 Parameters

parameter name	description	input Output
pin	Pin number	Input
userdata	user data	input

Chapter 10

Integrated circuit built-in bus (i2c)

10.1 Overview

The i2c bus is used to communicate with multiple external devices. Multiple external devices can share an i2c bus.

10.2 Functional description

The i2c module has the following features:

- Independent i2c device package peripheral related parameters
- Automatic processing of multi-device bus contention
- Support slave mode

10.3 Api reference

Corresponding header file devices.h

Provide users with 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 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 Parameters

parameter name	description	input Output
file clock_rate	I2c device handle expected clock rate	Input input

10.3.2.4 Return value

The actual rate after setting.

10.3.3 i2c_dev_transfer_sequential

10.3.3.1 description

Read and write to the i2c device first.

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 Output
file	I2c 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

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 Output
file	I2c controller handle	Input

	parameter name	description	input Output
	slave_address	Slave address	Inpu
	address_width	Slave device	t
	handler	address width	inpu
		from device	t
		handler	

10.3.4.4 return value

no.

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 Parameters

parameter name	description	input Output
file clock_rate	I2c controller	Inpu
	handle	t
	expected clock	inpu
	rate	t

10.3.5.4 Return value

The actual rate after setting.

10.3.6 Example

```
handle_t i2c = io_open("/dev/i2c0");
/* i2c Peripheral address is 0x32, 7-bit address, rate 200K */
handle_t dev0 = i2c_get_device(i2c, "/dev/i2c0/dev0", 0x32, 7); i2c_dev_se
t_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 of data from the 0 register*/
```

10.4 type of data

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

- I2c_event_t: I2C event.
- I2c_slave_handler_t: I2C slave handler.

10.4.1 i2c_event_t

10.4.1.1 description

I2c event.

10.4.1.2 definition

```
typedef enum _i2c_event
{
    I2C_EV_
    START , I2C_
    EV_RESTART ,
    I2C_EV_STOP
```

10.4.1.3 member

Member name	description
I2C_EV_START	Received Start signal
I2C_EV_RESTART	Received a Restart signal
I2C_EV_STOP	Received a Stop signal

10.4.2 i2c_slave_handler_t

10.4.2.1 description

I2c slave device handler.

10.4.2.2 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 member

Member 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

Integrated circuit built-in audio bus (i2s)

11.1 Overview

The i2s standard bus defines three types of signals: the clock signal bck, the channel selection signal ws, and the serial data signal sd.a basic

The i2s data bus has one master and one slave.The roles of the master and slave remain unchanged during the communication process.The i2s module includes separate transmit and receive channels for excellent communication performance.

11.2 Functional description

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
- Automatically manage audio buffers

11.3 Api reference

Corresponding header file devices.h

Provide users with 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 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 Output
file format	I2s controller	Inpu
delay_ms	handle audio	t
align_mode	format	inpu
channels_mask	Buffer	t
	length	inpu
	alignment	t
	mode	inpu
	Channel mask	t

11.3.1.4 return value

no.

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 Parameters

parameter name	description	input Output
file format	I2s controller	Inpu
delay_ms	handle audio	t
	format	inpu
	Buffer length	t

parameter name	description	input Output
align_mode	Align mode	Input
channels_mask	Channel mask	Input

11.3.2.4 Return

value is 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 Output
file	I2s controller handle	Input
buffer	Buffer	Output
frames	Number of buffer frames	Output

11.3.3.4 The return value is 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 Output
file	I2s controller handle	Input
frames	Confirm the number of frames that have been read or written	Input

11.3.4.4 The return value is 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 Output
file	I2s controller handle	Input

11.3.5.4 The return value is 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 Output
file	I2s controller handle	Input

11.3.6.4 The return value is none.

11.3.7 Example

```
/* Loop the 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_s
tart(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 (
```

11.4 type of data

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

- Audio_format_type_t: The audio format type.
- Audio_format_t: Audio format.
- I2s_align_mode_t: I2S alignment mode.

11.4.1 audio_format_type_t

11.4.1.1 Describe the audio format type.

11.4.1.2 definition

```
typedef enum _audio_format_type
{
    AUDIO_FMT_
        PCM
```

11.4.1.3 member

Member name	description
AUDIO_FMT_PCM	PCM

11.4.2 audio_format_t

11.4.2.1 Desc
ribe the
audio format.

11.4.2.2 definition

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

11.4.2.3 member

Member name	description
type bits_per_sample	Audio format
sample_rate	type
channels	sampling
	depth
	Sampling Rate
	Number of channels

11.4.3 i2s_align_mode_t

11.4.3.1 description
I2s alignment mode.

11.4.3.2 definition

```
typedef enum _i2s_align_mode
{
    I2S_AM_
    STANDARD,
    I2S_AM_ RIGHT ,I
    2S_AM_LEFT
```

11.4.3.3 member

Member name	description
I2S_AM_STANDARD	Standard mode
I2S_AM_RIGHT	Align right
I2S_AM_LEFT	Align left

Chapter 12

Serial peripheral interface (spi)

12.1 Overview

Spi is a high speed, full duplex, synchronous communication bus.

12.2 Functional description

The spi module has the following features:

- Independent spi device package 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 reference

Corresponding header file devices.h

Provide users with 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 a spi device.

12.3.1.2 Function prototype

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

12.3.1.3 Parameters

parameter name	description	input 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

Sp 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 Output
file	Spi device handle	Input

parameter name	description	input Output
instruction_length	Instructio	Inpu
address_length	n length	t
wait_cycles trans_mode	address	inpu
	length	t
	waiting	inpu
	period	t
	Command and address transfer mode	

12.3.2.4 return value

no.

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 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 Parameters

parameter name	description	input Output
file write_buffer	Spi device	Inpu
write_len	handle	t
read_buffer	source	inpu
read_len	buffer	t,
	Number of	inpu
	bytes to write	t
	to the target	and
	buffer	outp
	Maximum number of bytes read	ut

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.

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 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

12.3.5.4 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 string 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 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 The return value is none.

12.3.7 Example

```
handle_t spi = io_open("/dev/spi0");
/* dev0 works in MODE0 mode Standard SPI mode Single-send 8-bit data 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 0 x06 Send 0, 1 , 2 , 3 four bytes of data to address 0 x010204 * /
io_write(dev0, data_buf, sizeof(data_buf));
/* Send instruction 0 x06 Address 0 x010204 Receive four bytes of data */
spi_dev_transfer_sequential(dev0, data_buf, 4, data_buf, 4);
```

12.4 type of data

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: Transmission mode of the SPI instruction and address.

12.4.1 spi_mode_t

12.4.1.1 description

Spi mode.

12.4.1.2 definition

```
typedef enum _spi_mode
{
    SPI_MODE_
    0          ,
    SPI_MODE_
    1          ,
```

12.4.1.3 member

	Member 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 definition

```
typedef enum _spi_frame_format
{
    SPI_FF_
    STANDARD,
    SPI_FF_DUAL ,
    SPI_FF_QUAD ,
```

12.4.2.3 member

Member name	description
SPI_FF_STANDARD	standard
SPI_FF_DUAL	Double line
SPI_FF_QUAD	Four lines
SPI_FF_OCTAL	Eight lines (not supported by /dev/spi3)

12.4.3 spi_inst_addr_trans_mode_t

12.4.3.1 description

The transmission mode of the spi instruction and address.

12.4.3.2 definition

```
typedef enum _spi_inst_addr_trans_mode
{
    SPI_AITM_STANDARD, SPI_AITM_ADDR_STANDARD, SPI_AITM_AS_
    FRAME_FORMAT
```

12.4.3.3 member

Member name	description
SPI_AITM_STANDARD	Use 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	Use configured values

Chapter 13

Digital camera interface (dvp)

13.1 Overview

Dvp is a camera interface module that supports forwarding camera input image data to the ai module or memory.

13.2 Functional description

The dvp module has the following features:

- 2 video data output ports for RGB565 and RGB24Planar
- Support for dropping frames that do not need to be processed

13.3 Api reference

Corresponding header file devices.h

Provide users with the following interfaces:

- 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_config

13.3.1.1 Describe the configuration of the dvp device.

13.3.1.2 Function prototype

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

13.3.1.3 parameter

parameter name	description	input Output
file	Dvp device handle	Input
width	Frame width	Input
height	Frame height	Input
auto_enable	Automatically enable frame processing	Input

13.3.1.4 The return value is none.

13.3.2 dvp_enable_frame

13.3.2.1 description

Enable processing of the current frame.

13.3.2.2 Function prototype

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

13.3.2.3 parameter

parameter name	description	input Output
file	Dvp device handle	Input

13.3.2.4 The return value is

none.

13.3.3 dvp_get_output_num

13.3.3.1 description

Get the number of outputs of the dvp device.

13.3.3.2 Function prototype

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

13.3.3.3 parameter

parameter name	description	input Output
file	Dvp device handle	Input

13.3.3.4 Retu

rns the

number of

output.

13.3.4 dvp_set_signal

13.3.4.1 description

Set the dvp signal status.

13.3.4.2 Function prototype

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

13.3.4.3 Parameters

parameter name	description	input Output
file	Dvp device	Inpu
type	handle	t
value	signal type	inpu
	Status value	t

13.3.4.4 Return value

no.

13.3.5 dvp_set_output_enable

13.3.5.1 description

Set whether dvp output is enabled.

13.3.5.2 Function prototype

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

13.3.5.3 parameter

parameter name	description	input Output
file	Dvp device handle	Input
index	Output index	Input
enable	Whether to enable	Input

13.3.5.4 The return value is none.

13.3.6 dvp_set_output_attributes

13.3.6.1 description

Set the dvp output characteristics.

13.3.6.2 Function prototype

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

13.3.6.3 Parameters

parameter name	description	input Output
file index	Dvp device handle	Input
format	output index	input
output_buffer	Video format	input and output
	Output buffer	

13.3.6.4 Return

value None.

13.3.7 dvp_set_frame_event_enable

13.3.7.1 description

Sets whether dvp frame events are enabled.

13.3.7.2 Function prototype

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

13.3.7.3 parameter

parameter name	description	input Output
file	Dvp device handle	Input
event	Frame event	Input
enable	Whether to enable	Input

13.3.7.4 The

return

value is

none.

13.3.8 dvp_set_on_frame_event

13.3.8.1 description

Set the dvp frame event handler.

13.3.8.2 Function prototype

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

13.3.8.3 Parameters

parameter name	description	input Output
file handler	Dvp device	Inpu
userdata	handle	t
	handler	inpu
	Handler user data	t

13.3.8.4 Return value None.

13.3.9 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_eve
nt_enable(dvp, VIDEO_FE_BEGIN, true);
dvp_set_output_attributes(dvp, 0, VIDEO_FMT_RGB565, lcd_gram0); dv
```

13.4 type of data

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: DVP frame event handler.

13.4.1 video_format_t

13.4.1.1 Describe the video format.

13.4.1.2 definition

```
typedef enum _video_format
{
    VIDEO_ FMT_
    RGB565 , V IDEO_FMT
     RGB24_PLANAR
```

13.4.1.3 member

Member 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 definition

```
typedef enum _video_frame_event
{
    VIDEO_FE_
    BEGIN, VIDEO_
    FE_END
```

13.4.2.3 member

Member 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 definition

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

13.4.3.3 Members

Member name	description
DVP_SIG_POWER_DOWN	Power down
DVP_SIG_RESET	reset

13.4.4 dvp_on_frame_event_t

13.4.4.1 description

The handler when the timer is triggered.

13.4.4.2 definition

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

13.4.4.3 parameter

parameter name	description	input Output
userdata	User data	Input

Chapter 14

Serial camera control bus (sccb)

14.1 Overview

Sccb is a serial camera control bus.

14.2 Functional description

The sccb module has the following features:

- Independent sccb device package peripheral related parameters
- Automatic processing of multi-device bus contention

14.3 Api reference

Corresponding header file devices.h

Provide users with 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_addre
    ss_width);
```

14.3.1.3 Parameters

parameter name	description	input Output
file	Sccb controller handle	Inpu
name	Specify the path	t
slave_address	from the device to	inpu
reg_address_width	access the device	t
	Register address width	inpu t

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 scbb 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 Output
file	Sccb device handle	Input
reg_address	Register address	Input

14.3.2.4 Returns the byte read by the value.

14.3.3 sccb_dev_write_byte

14.3.3.1 description

Write a byte to the sccc 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 Output
file	Sccb device handle	Input
reg_address	Register address	Input
value	The byte to be written	Input

14.3.3.4 The return value is 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 provides high-precision timing.

15.2 Functional description

The timer module has the following features:

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

15.3 Api reference

Corresponding header file devices.h

Provide users with 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 Parameters

parameter name	description	input Output
file nanoseconds	Timer device handle	Inpu t
	interval (nanoseconds)	inpu t

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 fires.

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 Output
file	Timer device handle	Input
on_tick	Handler	Input
userdata	Handler user data	Input

15.3.2.4 The return value is none.

15.3.3 timer_set_enable

15.3.3.1 description

Set whether timer is enabled.

15.3.3.2 Function prototype

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

15.3.3.3 parameter

parameter name	description	input Output
file	Timer device handle	Input
enable	Whether to enable	Input

15.3.3.4 The return value is none.

15.3.4 Example

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

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

timer_set_interval(timer, 1e9);        t imer_set_
on_tick(timer, on_tick, NULL); timer_set_ena
```

15.4 type of data

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

- Timer_on_tick_t: The handler when 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 definition

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

15.4.1.3 parameter

parameter name	description	input Output
userdata	User data	Input

Chapter 16

Pulse width modulator (pwm)

16.1 Overview

Pwm is used to control the duty cycle of the pulse output.

16.2 Functional description

The pwm module has the following features:

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

16.3 Api reference

Corresponding header file devices.h

Provide users with 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 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 Descri

ption Sets the

pwm frequency.

16.3.2.2 Function prototype

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

16.3.2.3 Parameters

parameter name	description	input 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 Parameters

parameter name	description	input Output
file	Pwm device	Inpu
pin	handle pin	t
duty_cycle_percentage	number	inpu
	Expected duty cycle	t

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 Parameters

parameter name	description	input Output
file pin	Pwm device	Inpu
enable	handle pin	t
	number	inpu
	Whether to enable	t

16.3.4.4 Return value

The actual duty cycle after setting.

16.3.5 Example

```
/* pwm0 pin0 outputs 200 KHZ 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 ,
```

```
pwm_set_enable(pwm, 0, true);
```

Chapter 17

Watchdog timer (wdt)

17.1 Overview

Wdt provides recovery when the system is in error or not responding.

17.2 Functional description

The wdt module has the following features:

- Configuration timeout
- Manual restart timing
- Configured 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 reference

Corresponding header file devices.h

Provide users with 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 Output
file	Wdt device handle	Input
mode	Response mode	Input

17.3.1.4 The return value is none.

17.3.2 wdt_set_timeout

17.3.2.1 description

Set the wdt timeout.

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 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 Output
file	Wdt device handle	Input
handler	Handler	Input
userdata	Handler user data	Input

17.3.3.4 The return value is none.

17.3.4 wdt_restart_counter

17.3.4.1 description

Cause wdt to restart counting.

17.3.4.2 Function prototype

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

17.3.4.3 parameter

parameter name	description	input Output
file	Wdt device handle	Input

17.3.4.4 The return value is 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 Output
file	Wdt device handle	Input
enable	Whether to enable	Input

17.3.5.4 The return value is none.

17.3.6 Example

```
/* After 2 seconds, enter the watchdog interrupt function to print the imeout, and then reset it in 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);
```

17.4 type of data

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 definition

```
typedef enum _wdt_response_mode
{
    WDT_ RESP_
    RESET , WDT_RES
    P_INTERRUPT
```

17.4.1.3 member

Member name	description
WDT_RESP_RESET	Reset system after timeout
WDT_RESP_INTERRUPT	Enter the interrupt after timeout, reset the system again after timeout

17.4.2 wdt_on_timeout_t

17.4.2.1 description

Wdt timeout handler.

17.4.2.2 definition

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

17.4.2.3 parameter

parameter name	description	input Output
userdata	User data	Input

17.4.2.4 return value

return value	description
0	The system will reset if the interrupt is not cleared.

return value	description
1	Clear interrupt, system does not reset

Chapter 18

Fast Fourier Transform Accelerator (fft)

18.1 Overview

The fft module is hardware-based to implement the base 2 time-division acceleration of fft.

18.2 Functional description

The module currently supports 64-point, 128-point, 256-point, and 512-point FFTs as well as IFFT. Inside the FFT, 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 DMA sent the data is placed in one of the SRAMs until the current FFT operation is satisfied. The amount of data starts and the FFT operation begins. The butterfly unit reads the data from the SRAM containing the valid data. After the operation ends, the data is written to another SRAM, and the next butterfly operation is performed from the SRAM just written. The data is read out, and after the operation is completed, another SRAM is written, and thus iteratively repeats until the entire FFT operation is completed.

18.3 Api reference

Corresponding header file fft.h

Provide users with the following interfaces:

- 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 Output
shift	Fft module 16-bit register causes data overflow (-32768~32767), the FFT transform has 9 layers, shift determines which layer needs to be shifted (such as 0x1ff means that 9 layers are all shifted; 0x03 means the first layer and the second layer do shift operations), Prevent spillage.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.Contains examples of solving frequency points, phases, and amplitudes	Input
direction	Fft positive or inverse	Input
input	The input data sequence, in the format rri.., the real and imaginary parts t is 16 bit	Inpu
point_num 512/256/128/64.	The number of data points to be calculated can only be	Input
output	The result after the operation.The format is rri.. , the precision of the real and imaginary parts output is 16 bit	The

18.3.2 Example

```
#define FFT_N                   512U
#define FFT_FORWARD_SHIFT       0x0U
#define FFT_BACKWARD_SHIFT      0x1ffU
#define PI                      3.14159265358979323846
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_har
    d[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_har
        d[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 type of data

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

- fft_data_t:fft Operates the incoming data format.
- fft_direction_t:fft operation mode.

18.4.1 fft_data_t

18.4.1.1 description

Fft computes the incoming data format.

18.4.1.2 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 member

Member 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 operation mode

18.4.2.2 definition

```
typedef enum tag_fft_direction
{
    FFT_DIR_
    BACKWARD ,FFT_
    DIR_FORWARD,
```

18.4.2.3 member

Member name	description
FFT_DIR_BACKWARD	Fft inverse transform
FFT_DIR_FORWARD	Fft positive transform

Chapter 19

Secure Hash Algorithm Accelerator (sha256)

19.1 Overview

The sha256 module uses hardware to implement the time division operation acceleration of sha256.

19.2 Functional description

- Support for sha-256 calculations

19.3 Api reference

Corresponding header file sha256.h

Provide users with the following interfaces:

- sha256_hard_calculate

19.3.1 sha256_hard_calculate

19.3.1.1 description

Shabrating the 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 Output
input	Data to be calculated by sha256	Input
input_len	Waiting for sha256 to calculate the length of the data	Input
output	To store the result of the SHA256 calculation, ensure that the size of the incoming buffer is 32 bytes.	Output

19.3.2 Example

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

Chapter 20

Advanced crypto accelerator (aes)

20.1 Overview

The aes module is hardware-based to implement the time-division acceleration of aes.

20.2 Functional description

The k210 has built-in aes (Advanced Encryption Accelerator), which can greatly improve the speed of aes operation compared to software.The aes accelerator supports multiple encryption/decryption modes (ecb, cbc, gcm) and multiple length keys (128, 192, 256).

20.3 Api reference

Corresponding header file aes.h

Provide users with 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

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 Output
input_key	Aes-ecb-128 encrypted 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
output_data	The result of the AES-ECB-128 encryption operation is stored in this buffer.	Output

20.3.1.4 The return value is none.

20.3.2 aes_ecb128_hard_decrypt

20.3.2.1 description

Aes-ecb-128 decryption operation

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 Output
input_key	Aes-ecb-128 decrypted key	Inpu
input_data	Aes-ecb-128 ciphertext data to be decrypted	t
input_len	Aes-ecb-128 Length of ciphertext data to be decrypted	inpu
output_data	The result of the AES-ECB-128 decryption operation is stored in this buffer.	t and outp ut

20.3.2.4 return value

no.

20.3.3 aes_ecb192_hard_encrypt

20.3.3.1 description

Aes-ecb-192 encryption operation

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 Output
input_key	Aes-ecb-192 encrypted key	Inpu
input_data	Aes-ecb-192 plaintext data to be encrypted	t
input_len	Aes-ecb-192 Length of plaintext data to be encrypted	inpu
output_data	The result of AES-ECB-192 encryption operation is stored in this buffer.	t and outp ut

20.3.3.4 return value

no.

20.3.4 aes_ecb192_hard_decrypt

20.3.4.1 description

Aes-ecb-192 decryption operation

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 Output
input_key	Aes-ecb-192 decrypted key	Input
input_data	Aes-ecb-192 ciphertext data to be decrypted	
input_len	Aes-ecb-192 Length of ciphertext data to be decrypted	input
output_data	The result of the AES-ECB-192 decryption operation is stored in this buffer.	and output

20.3.4.4 return value

no.

20.3.5 aes_ecb256_hard_encrypt

20.3.5.1 description

Aes-ecb-256 encryption operation

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 Output
input_key	Aes-ecb-256 encrypted 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
output_data	The result of the AES-ECB-256 encryption operation is stored in this buffer.	Output

20.3.5.4 The return value is none.

20.3.6 aes_ecb256_hard_decrypt

20.3.6.1 description

Aes-ecb-256 decryption operation

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 Output
input_key	Aes-ecb-256 decrypted key	Inpu
input_data	Aes-ecb-256 ciphertext data to be decrypted	t
input_len	Aes-ecb-256 Length of ciphertext data to be decrypted	inpu
output_data	The result of the AES-ECB-256 decryption operation is stored in this buffer.	t
		and
		outp
		ut

20.3.6.4 return value

no.

20.3.7 aes_cbc128_hard_encrypt

20.3.7.1 description

Aes-cbc-128 encryption operation

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
Parameters

parameter name	description	input Output
context	Aes-cbc-128 Encrypted computed 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

parameter name	description	input Output
output_data	The result of AES-CBC-128 encryption operation is stored in this buffer.	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

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
Parameters

parameter name	description	input Output
context	Aes-cbc-128 decrypts the computed structure, including the 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
output_data	The result of the AES-CBC-128 decryption operation is stored in this buffer.	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

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 Output
context	Aes-cbc-192 Encrypted computed 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
output_data	The result of AES-CBC-192 encryption operation is stored in this buffer.	Output

20.3.9.4 The return value is none.

20.3.10 aes_cbc192_hard_decrypt

20.3.10.1 description

Aes-cbc-192 decryption operation

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 Output
context	Aes-cbc-192 Decrypted computed structure containing 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
output_data	The result of the AES-CBC-192 decryption operation is stored in this buffer.	Output

20.3.10.4 The return value is none.

20.3.11 aes_cbc256_hard_encrypt

20.3.11.1 description

Aes-cbc-256 encryption operation

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 Output
context	Aes-cbc-256 Encrypted computed 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
output_data	The result of the AES-CBC-256 encryption operation is stored in this buffer.	Output

20.3.11.4 The return value is none.

20.3.12 aes_cbc256_hard_decrypt

20.3.12.1 description

Aes-cbc-256 decryption operation

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 Output
context	Aes-cbc-256 Decrypted computed structure containing 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
output_data	The result of the AES-CBC-256 decryption operation is stored in this buffer.	Output

20.3.12.4 The return value is none.

20.3.13 aes_gcm128_hard_encrypt

20.3.13.1 description

Aes-gcm-128 encryption operation

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 Output
context	AES-GCM-128 Encrypted computed structure containing 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
output_data	The result of the AES-GCM-128 encryption operation is stored in this buffer.	Output
gcm_tag	The tag after AES-GCM-128 encryption operation is stored in this buffer.	Output

20.3.13.4 The return value is

none.

20.3.14 aes_gcm128_hard_decrypt

20.3.14.1 description

Aes-gcm-128 decryption operation

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 Output
context	AES-GCM-128 decrypts the computed structure, including the decryption key/offset vector/aad/aad length	Input
input_data	Aes-gcm-128 ciphertext data to be decrypted	Input
input_len	Aes-gcm-128 Length of ciphertext data to be decrypted	Input

parameter name	description	input Output
output_data	The result of the AES-GCM-128 decryption operation is stored in this buffer.	Output
gcm_tag	The tag after the decryption operation of AES-GCM-128 is stored in this buffer.	Output

20.3.14.4 The return value is none.

20.3.15 aes_gcm192_hard_encrypt

20.3.15.1 description

Aes-gcm-192 encryption operation

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 Output
context	AES-GCM-192 Encrypted computed structure containing encryption key/offset vector/aad/aad length	Input
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.	Output
gcm_tag	The AES-GCM-192 encryption operation tag is stored in this buffer.	Output

20.3.15.4 The return value is none.

20.3.16 aes_gcm192_hard_decrypt

20.3.16.1 description

Aes-gcm-192 decryption operation

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

parameter name	description	input Output
context	AES-GCM-192 decrypts the computed structure, 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.	Output
gcm_tag	AES-GCM-192 decrypted operation of the tag stored in this buffer	Output

20.3.16.4 The return value is none.

20.3.17 aes_gcm256_hard_encrypt

20.3.17.1 description

Aes-gcm-256 encryption operation

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

parameter name	description	input Output
context	AES-GCM-256 Encrypted computed structure containing 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 AES-GCM-256 encryption operation is stored in this buffer.	Output
gcm_tag	The tag after AES-GCM-256 encryption operation is stored in this buffer.	Output

20.3.17.4 The return value is none.

20.3.18 aes_gcm256_hard_decrypt

20.3.18.1 description

Aes-gcm-256 decryption operation

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

parameter name	description	input Output
context	AES-GCM-256 decrypted computed structure containing 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 buffer.	The result of the AES-GCM-256 decryption operation is stored in this	Output
gcm_tag buffer.	The tag after the AES-GCM-256 decryption operation is stored in this	Output

20.3.18.4 The return value is none.

20.3.19 Example

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

20.4 type of data

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

- aes_cipher_mode_t:aes The way to encrypt/decrypt.

20.4.1 aes_cipher_mode_t

20.4.1.1 description

Aes The way to encrypt/decrypt.

20.4.1.2 definition

```
typedef enum _aes_cipher_mode
{
    AES_ECB = 0,
    AES_CBC = 1,
    AES_ GCM = 2, A
    ES_CIPHER_MAX
} aes_cipher_mode_t;
```

20.4.1.3 Members

Member name	description
AES_ECB	Ecb encryption/decryption
AES_CBC	Cbc encryption/decryption
AES_GCM	Gcm encryption/decryption