K210
Standalone SDK
Programming guide

KENDRYTE

KENDRYTE.COM

About This Guide

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

Corresponding SDK version

Kendryte Standalone SDK v0.5.1 (*414021b7378f7a56babb5ec02666e3cc03af59fc*)

Revision History

Date	Ver.	Revision History
2018-10-10	V0.1.0	Initial release
2018-10-20	V0.2.0	Released Standalone SDK v0.4.0 documentation
2018-11-02	V0.3.0	Released Standalone SDK v0.5.1 documentation

Disclaimer

Copyright Notice

Contents

Chapter 1

KPU

1.1 Overview

Knowledge Processing Unit (KPU, aka Neural network Processing Unit).

KPU is a general-purpose neural network processor that implements convolutional neural network calculations with low power consumption. It can acquire the size, coordinates and types of detected objects in real time, and detect and classify faces or objects. When using kpu, the neural network model must be generated with the model compiler.

1.2 Features

KPU has the following characteristics:

- Supports the fixed training model that the common training framework trains according to specific restriction rules
- There is no direct limit on the number of network layers, which supports separate configuration of each layer of convolutional neural network parameters, including the number of input and output channels, input and output line width and column height.
- Support for two convolution kernels 1x1 and 3x3
- Support for any form of activation function
- The maximum supported neural network parameter size in real-time work is 5.5MiB to 5.9MiB
- Maximum support network parameter size when working in non-real time is (Flash capacity) - (software size)

1.3 API

Corresponding header file kpu.h

Provide the following interfaces

- kpu_task_init
- kpu_run
- kpu_get_output_buf
- kpu_release_output_buf

1.3.1 kpu_task_init

1.3.1.1 Description

Initialize the kpu task handle, which is implemented in the gencode_output.c generated by the model compiler.

1.3.1.2 Function prototype

```
kpu_task_t* kpu_task_init(kpu_task_t* task)
```

1.3.1.3 Parameter

Parameter name	Description	Input or output
task	KPU task handle	Input

1.3.1.4 Return value

KPU task handle.

1.3.2 kpu_run

1.3.2.1 Description

Start the KPU and perform the AI operation.

1.3.2.2 Function prototype

```
int kpu_run(kpu_task_t* v_task, dmac_channel_number_t dma_ch, const void *src, void*
    dest, plic_irq_callback_t callback)
```

1.3.2.3 Parameter

Parameter name	Description	Input or output
task	KPU task handle	Input
dma_ch	DMA channel	Input
src	Input image data	Input
dest	Operation output	Output
callback	Operation completion callback function	Input

1.3.2.4 Return value

Return value	Description
0	Success
Others	KPU is busy. Fail

1.3.3 kpu_get_output_buf

1.3.3.1 Description

Get a buffer of KPU output results.

1.3.3.2 Function prototype

```
uint8_t *kpu_get_output_buf(kpu_task_t* task)
```

1.3.3.3 Parameter

Parameter name	Description	Input or output
task	KPU task handle	Input

1.3.3.4 Return value

A pointer to the buffer of the KPU output.

1.3.4 kpu_release_output_buf

1.3.4.1 Description

Release the KPU output result buffer.

1.3.4.2 Function prototype

```
void kpu_release_output_buf(uint8_t *output_buf)
```

1.3.4.3 Parameter

Parameter name	Description	Input or output
output_buf	KPU output result buffer	Input

1.3.4.4 Return value

None.

1.4 Data type

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

- kpu_task_t: KPU task structure.

1.4.1 kpu_task_t

1.4.1.1 Description

KPU task structure.

1.4.1.2 Type definition

```
typedef struct
{
    kpu_layer_argument_t* layers;
    uint32_t length;
    int dma_ch;
    uint64_t* dst;
    uint32_t dst_length;
    plic_irq_callback_t cb;
```

```
} kpu_task_t;
```

1.4.1.3 Enumeration element

Element name	Description
layers	KPU parameter pointer
length	Number of layers
dma_ch	DMA channel
dst	Operation result output buffer pointer
dst_length	Operation result output buffer length
cb	Operation completion callback function

Chapter 2

AES

2.1 Overview

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

2.2 Features

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

2.3 API

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

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

- aes_cbc192_hard_encrypt
- aes_cbc192_hard_decrypt
- aes_cbc256_hard_encrypt
- aes_cbc256_hard_decrypt
- aes_gcm128_hard_encrypt
- aes_gcm128_hard_decrypt
- aes_gcm192_hard_encrypt
- aes_gcm192_hard_decrypt
- aes_gcm256_hard_encrypt
- aes_gcm256_hard_decrypt
- aes_ecb128_hard_encrypt_dma
- aes_ecb128_hard_decrypt_dma
- aes_ecb192_hard_encrypt_dma
- aes_ecb192_hard_decrypt_dma
- aes_ecb256_hard_encrypt_dma
- aes_ecb256_hard_decrypt_dma
- aes_cbc128_hard_encrypt_dma
- aes_cbc128_hard_decrypt_dma
- aes_cbc192_hard_encrypt_dma
- aes_cbc192_hard_decrypt_dma
- aes_cbc256_hard_encrypt_dma
- aes_cbc256_hard_decrypt_dma
- aes_gcm128_hard_encrypt_dma
- aes_gcm128_hard_decrypt_dma
- aes_gcm192_hard_encrypt_dma
- aes_gcm192_hard_decrypt_dma
- aes_gcm256_hard_encrypt_dma
- aes_gcm256_hard_decrypt_dma
- aes_init
- aes_process
- gcm_get_tag

2.3.1 aes_ecb128_hard_encrypt

2.3.1.1 Description

AES-ECB-128 encryption operation. Input and output data are transmitted using CPU. ECB encryption encrypts the plaintext according to a block of fixed size of

16 bytes, and fills it when the block size is insufficient. The initialization vector(iv) is not used in ECB mode.

2.3.1.2 Function prototype

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

2.3.1.3 Parameter

Parameter name	Description	Input or output
input_key	AES-ECB-128 encryption key	Input
input_data	AES-ECB-128 plaintext data to be encrypted	Input
input_len	AES-ECB-128 length of plaintext data to be encrypted	Input
output_data	The result of the AES-ECB-128 encryption operation is stored in this buffer*1.	Output

2.3.1.4 Return value

None.

2.3.2 aes_ecb128_hard_decrypt

2.3.2.1 Description

AES-ECB-128 decryption operation. Input or output data is transmitted using CPU. ECB encryption encrypts the plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient. The initialization vector(iv) is not used in ECB mode.

2.3.2.2 Function prototype

*1 The size of this buffer needs to be guaranteed to be 16bytes aligned.

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

2.3.2.3 Parameter

Parameter name	Description	Input or output
input_key	AES-ECB-128 decryption key	Input
input_data	AES-ECB-128 ciphertext data to be decrypted	Input
input_len	AES-ECB-128 length of ciphertext data to be decrypted	Input
output_data	The result of the AES-ECB-128 decryption operation is stored in this buffer*2.	Output

2.3.2.4 Return value

None.

2.3.3 aes_ecb192_hard_encrypt

2.3.3.1 Description

AES-ECB-192 encryption operation. Input or output data is transmitted using CPU. ECB encryption encrypts the plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient. The initialization vector(iv) is not used in ECB mode.

2.3.3.2 Function prototype

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

2.3.3.3 Parameter

*2 The size of this buffer needs to be guaranteed to be 16bytes aligned.

Parameter name	Description	Input or output
input_key	AES-ECB-192 encryption key	Input
input_data	AES-ECB-192 plaintext data to be encrypted	Input
input_len	AES-ECB-192 length of plaintext data to be encrypted	Input
output_data	The result of the AES-ECB-192 encryption operation is stored in this buffer*3.	Output

2.3.3.4 Return value

None.

2.3.4 aes_ecb192_hard_decrypt

2.3.4.1 Description

AES-ECB-192 decryption operation. Input or output data is transmitted using CPU. ECB encryption encrypts the plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient. The initialization vector(iv) is not used in ECB mode.

2.3.4.2 Function prototype

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

2.3.4.3 Parameter

Parameter name	Description	Input or output
input_key	AES-ECB-192 decryption key	Input

*3 The size of this buffer needs to be guaranteed to be 16bytes aligned.

Parameter name	Description	Input or output
input_data	AES-ECB-192 ciphertext data to be decrypted	Input
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[*4].	Output

2.3.4.4 Return value

None.

2.3.5 aes_ecb256_hard_encrypt

2.3.5.1 Description

AES-ECB-256 encryption operation. Input or output data is transmitted using CPU. ECB encryption encrypts the plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient. The initialization vector(iv) is not used in ECB mode.

2.3.5.2 Function prototype

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

2.3.5.3 Parameter

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

[*4] The size of this buffer needs to be guaranteed to be 16bytes aligned.

Parameter name	Description	Input or output
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*5.	Output

2.3.5.4 Return value

None.

2.3.6 aes_ecb256_hard_decrypt

2.3.6.1 Description

AES-ECB-256 decryption operation. Input or output data is transmitted using CPU. ECB encryption encrypts the plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient. The initialization vector(iv) is not used in ECB mode.

2.3.6.2 Function prototype

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

2.3.6.3 Parameter

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

*5 The size of this buffer needs to be guaranteed to be 16bytes aligned.

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

2.3.6.4 Return value

None.

2.3.7 aes_cbc128_hard_encrypt

2.3.7.1 Description

AES-CBC-128 encryption operation. Input or output data is transferred using the CPU. CBC encryption encrypts plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient.

2.3.7.2 Function prototype

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

2.3.7.3 Parameter

Parameter name	Description	Input or output
context	AES-CBC-128 encryption operation structure containing encryption key and offset vector	Input
input_data	AES-CBC-128 plaintext data to be encrypted	Input
input_len	AES-CBC-128 length of plaintext data to be encrypted	Input
output_data	The result of the AES-CBC-128 encryption operation is stored in this buffer*7.	Output

*6 The size of this buffer needs to be guaranteed to be 16bytes aligned.

*7 The size of this buffer needs to be guaranteed to be 16bytes aligned.

2.3.7.4 Return value

None.

2.3.8 aes_cbc128_hard_decrypt

2.3.8.1 Description

AES-CBC-128 decryption operation. Input or output data is transferred using the CPU. CBC encryption encrypts plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient.

2.3.8.2 Function prototype

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

2.3.8.3 Parameter

Parameter name	Description	Input or output
context	AES-CBC-128 decryption operation structure, including decryption key and offset vector	Input
input_data	AES-CBC-128 ciphertext data to be decrypted	Input
input_len	AES-CBC-128 length of ciphertext data to be decrypted	Input
output_data	The result of the AES-CBC-128 decryption operation is stored in this buffer[*8].	Output

2.3.8.4 Return value

None.

[*8] The size of this buffer needs to be guaranteed to be 16bytes aligned.

2.3.9 aes_cbc192_hard_encrypt

2.3.9.1 Description

AES-CBC-192 encryption operation. Input or output data is transferred using the CPU. CBC encryption encrypts plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient.

2.3.9.2 Function prototype

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

2.3.9.3 Parameter

Parameter name	Description	Input or output
context	AES-CBC-192 encryption operation structure containing encryption key and offset vector	Input
input_data	AES-CBC-192 plaintext data to be encrypted	Input
input_len	AES-CBC-192 length of plaintext data to be encrypted	Input
output_data	The result of the AES-CBC-192 encryption operation is stored in this buffer*9.	Output

2.3.9.4 Return value

None.

2.3.10 aes_cbc192_hard_decrypt

2.3.10.1 Description

AES-CBC-192 decryption operation. Input or output data is transferred using the CPU. CBC encryption encrypts plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient.

*9 The size of this buffer needs to be guaranteed to be 16bytes aligned.

2.3.10.2 Function prototype

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

2.3.10.3 Parameter

Parameter name	Description	Input or output
context	AES-CBC-192 decryption operation structure, including decryption key and offset vector	Input
input_data	AES-CBC-192 ciphertext data to be decrypted	Input
input_len	AES-CBC-192 length of ciphertext data to be decrypted	Input
output_data	The result of the AES-CBC-192 decryption operation is stored in this buffer*10.	Output

2.3.10.4 Return value

None.

2.3.11 aes_cbc256_hard_encrypt

2.3.11.1 Description

AES-CBC-256 encryption operation. Input or output data is transferred using the CPU. CBC encryption encrypts plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient.

2.3.11.2 Function prototype

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

*10 The size of this buffer needs to be guaranteed to be 16bytes aligned.

2.3.11.3 Parameter

Parameter name	Description	Input or output
context	AES-CBC-256 encryption operation structure containing encryption key and offset vector	Input
input_data	AES-CBC-256 plaintext data to be encrypted	Input
input_len	AES-CBC-256 length of plaintext data to be encrypted	Input
output_data	The result of the AES-CBC-256 encryption operation is stored in this buffer*11.	Output

2.3.11.4 Return value

None.

2.3.12 aes_cbc256_hard_decrypt

2.3.12.1 Description

AES-CBC-256 decryption operation. Input or output data is transferred using the CPU. CBC encryption encrypts plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient.

2.3.12.2 Function prototype

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

2.3.12.3 Parameter

*11 The size of this buffer needs to be guaranteed to be 16bytes aligned.

Parameter name	Description	Input or output
context	AES-CBC-256 decryption operation structure, including decryption key and offset vector	Input
input_data	AES-CBC-256 ciphertext data to be decrypted	Input
input_len	AES-CBC-256 length of ciphertext data to be decrypted	Input
output_data	The result of the AES-CBC-256 decryption operation is stored in this buffer*12.	Output

2.3.12.4 Return value

None.

2.3.13 aes_gcm128_hard_encrypt

2.3.13.1 Description

AES-GCM-128 encryption operation. Input or output data is transferred using the CPU.

2.3.13.2 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)
```

2.3.13.3 Parameter

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

*12 The size of this buffer needs to be guaranteed to be 16bytes aligned.

Parameter name	Description	Input or output
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 the AES-GCM-128 encryption operation is stored in this buffer[*13].	Output

2.3.13.4 Return value

None.

2.3.14 aes_gcm128_hard_decrypt

2.3.14.1 Description

AES-GCM-128 decryption operation. Input or output data is transferred using the CPU.

2.3.14.2 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)
```

2.3.14.3 Parameter

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

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

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

2.3.14.4 Return value

None.

2.3.15 aes_gcm192_hard_encrypt

2.3.15.1 Description

AES-GCM-192 encryption operation. Input or output data is transferred using the CPU.

2.3.15.2 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)
```

2.3.15.3 Parameter

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

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

Parameter name	Description	Input or output
gcm_tag	The tag after the AES-GCM-192 encryption operation is stored in this buffer*15.	Output

2.3.15.4 Return value

None.

2.3.16 aes_gcm192_hard_decrypt

2.3.16.1 Description

AES-GCM-192 decryption operation. Input or output data is transferred using the CPU.

2.3.16.2 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)
```

2.3.16.3 Parameter

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

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

Parameter name	Description	Input or output
gcm_tag	The tag after the AES-GCM-192 decryption operation is stored in this buffer[*16].	Output

2.3.16.4 Return value

None.

2.3.17 aes_gcm256_hard_encrypt

2.3.17.1 Description

AES-GCM-256 encryption operation. Input or output data is transferred using the CPU.

2.3.17.2 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)
```

2.3.17.3 Parameter

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

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

Parameter name	Description	Input or output
gcm_tag	The tag after the AES-GCM-256 encryption operation is stored in this buffer[*17].	Output

2.3.17.4 Return value

None.

2.3.18 aes_gcm256_hard_decrypt

2.3.18.1 Description

AES-GCM-256 decryption operation. Input or output data is transferred using the CPU.

2.3.18.2 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)
```

2.3.18.3 Parameter

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

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

Parameter name	Description	Input or output
gcm_tag	The tag after the AES-GCM-256 decryption operation is stored in this buffer*18.	Output

2.3.18.4 Return value

None.

2.3.19 aes_ecb128_hard_encrypt_dma

2.3.19.1 Description

AES-ECB-128 encryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA. ECB encryption encrypts the plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient. The initialization vector(iv) is not used in ECB mode.

2.3.19.2 Function prototype

```
void aes_ecb128_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num, uint8_t
     *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)
```

2.3.19.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
input_key	AES-ECB-128 encryption key	Input
input_data	AES-ECB-128 plaintext data to be encrypted	Input
input_len	AES-ECB-128 length of plaintext data to be encrypted	Input
output_data	The result of the AES-ECB-128 encryption operation is stored in this buffer*19.	Output

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

*19 The size of this buffer needs to be guaranteed to be 16bytes aligned.

2.3.19.4 Return value

None.

2.3.20 aes_ecb128_hard_decrypt_dma

2.3.20.1 Description

AES-ECB-128 decryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA. ECB encryption encrypts the plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient. The initialization vector(iv) is not used in ECB mode.

2.3.20.2 Function prototype

```
void aes_ecb128_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num, uint8_t
     *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)
```

2.3.20.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
input_key	AES-ECB-128 decryption key	Input
input_data	AES-ECB-128 ciphertext data to be decrypted	Input
input_len	AES-ECB-128 Length ofciphertext data to be decrypted	Input
output_data	The result of the AES-ECB-128 decryption operation is stored in this buffer[*20].	Output

2.3.20.4 Return value

None.

[*20] The size of this buffer needs to be guaranteed to be 16bytes aligned.

2.3.21 aes_ecb192_hard_encrypt_dma

2.3.21.1 Description

AES-ECB-192 encryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA. ECB encryption encrypts the plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient. The initialization vector(iv) is not used in ECB mode.

2.3.21.2 Function prototype

```
void aes_ecb192_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num, uint8_t
     *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)
```

2.3.21.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
input_key	AES-ECB-192 encryption key	Input
input_data	AES-ECB-192 plaintext data to be encrypted	Input
input_len	AES-ECB-192 length of plaintext data to be encrypted	Input
output_data	The result of the AES-ECB-192 encryption operation is stored in this buffer*21.	Output

2.3.21.4 Return value

None.

2.3.22 aes_ecb192_hard_decrypt_dma

2.3.22.1 Description

AES-ECB-192 decryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA. ECB encryption encrypts the plaintext according to a block of fixed size of 16 bytes, and fills it when the

*21 The size of this buffer needs to be guaranteed to be 16bytes aligned.

block size is insufficient. The initialization vector(iv) is not used in ECB mode.

2.3.22.2 Function prototype

```
void aes_ecb192_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num, uint8_t
    *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)
```

2.3.22.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
input_key	AES-ECB-192 decryption key	Input
input_data	AES-ECB-192 ciphertext data to be decrypted	Input
input_len	AES-ECB-192 Length ofciphertext data to be decrypted	Input
output_data	The result of the AES-ECB-192 decryption operation is stored in this buffer[*22].	Output

2.3.22.4 Return value

None.

2.3.23 aes_ecb256_hard_encrypt_dma

2.3.23.1 Description

AES-ECB-256 encryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA. ECB encryption encrypts the plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient. The initialization vector(iv) is not used in ECB mode.

2.3.23.2 Function prototype

```
void aes_ecb256_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num, uint8_t
    *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)
```

[*22] The size of this buffer needs to be guaranteed to be 16bytes aligned.

2.3.23.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
input_key	AES-ECB-256 encryption key	Input
input_data	AES-ECB-256 plaintext data to be encrypted	Input
input_len	AES-ECB-256 length of plaintext data to be encrypted	Input
output_data	The result of the AES-ECB-256 encryption operation is stored in this buffer*[23].	Output

2.3.23.4 Return value

None.

2.3.24 aes_ecb256_hard_decrypt_dma

2.3.24.1 Description

AES-ECB-256 decryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA. ECB encryption encrypts the plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient. The initialization vector(iv) is not used in ECB mode.

2.3.24.2 Function prototype

```
void aes_ecb256_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num, uint8_t
     *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)
```

2.3.24.3 Parameter

*[23] The size of this buffer needs to be guaranteed to be 16bytes aligned.

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
input_key	AES-ECB-256 decryption key	Input
input_data	AES-ECB-256 ciphertext data to be decrypted	Input
input_len	AES-ECB-256 Length ofciphertext data to be decrypted	Input
output_data	The result of the AES-ECB-256 decryption operation is stored in this buffer*24.	Output

2.3.24.4 Return value

None.

2.3.25 aes_cbc128_hard_encrypt_dma

2.3.25.1 Description

AES-CBC-128 encryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA. CBC encryption encrypts plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient.

2.3.25.2 Function prototype

```
void aes_cbc128_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num,
    cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *
    output_data)
```

2.3.25.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input

*24 The size of this buffer needs to be guaranteed to be 16bytes aligned.

Parameter name	Description	Input or output
context	AES-CBC-128 encryption operation structure containing encryption key and offset vector	Input
input_data	AES-CBC-128 plaintext data to be encrypted	Input
input_len	AES-CBC-128 length of plaintext data to be encrypted	Input
output_data	The result of the AES-CBC-128 encryption operation is stored in this buffer[*25].	Output

2.3.25.4 Return value

None.

2.3.26 aes_cbc128_hard_decrypt_dma

2.3.26.1 Description

AES-CBC-128 decryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA. CBC encryption encrypts plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient.

2.3.26.2 Function prototype

```
void aes_cbc128_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num,
    cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *
    output_data)
```

2.3.26.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-CBC-128 decryption operation structure, including decryption key and offset vector	Input

[*25] The size of this buffer needs to be guaranteed to be 16bytes aligned.

Parameter name	Description	Input or output
input_data	AES-CBC-128 ciphertext data to be decrypted	Input
input_len	AES-CBC-128 Length ofciphertext data to be decrypted	Input
output_data	The result of the AES-CBC-128 decryption operation is stored in this buffer*26.	Output

2.3.26.4 Return value

None.

2.3.27 aes_cbc192_hard_encrypt_dma

2.3.27.1 Description

AES-CBC-192 encryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA. CBC encryption encrypts plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient.

2.3.27.2 Function prototype

```
void aes_cbc192_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num,
    cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *
    output_data)
```

2.3.27.3 Parameter

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

*26 The size of this buffer needs to be guaranteed to be 16bytes aligned.

Parameter name	Description	Input or output
output_data	The result of the AES-CBC-192 encryption operation is stored in this buffer*27.	Output

2.3.27.4 Return value

None.

2.3.28 aes_cbc192_hard_decrypt_dma

2.3.28.1 Description

AES-CBC-192 decryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA. CBC encryption encrypts plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient.

2.3.28.2 Function prototype

```
void aes_cbc192_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num,
    cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *
    output_data)
```

2.3.28.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-CBC-192 decryption operation structure, including decryption key and offset vector	Input
input_data	AES-CBC-192 ciphertext data to be decrypted	Input
input_len	AES-CBC-192 Length ofciphertext data to be decrypted	Input
output_data	The result of the AES-CBC-192 decryption operation is stored in this buffer*28.	Output

*27 The size of this buffer needs to be guaranteed to be 16bytes aligned.

*28 The size of this buffer needs to be guaranteed to be 16bytes aligned.

2.3.28.4 Return value

None.

2.3.29 aes_cbc256_hard_encrypt_dma

2.3.29.1 Description

AES-CBC-256 encryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA. CBC encryption encrypts plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient.

2.3.29.2 Function prototype

```
void aes_cbc256_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num,
    cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *
    output_data)
```

2.3.29.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-CBC-256 encryption operation structure containing encryption key and offset vector	Input
input_data	AES-CBC-256 plaintext data to be encrypted	Input
input_len	AES-CBC-256 length of plaintext data to be encrypted	Input
output_data	The result of the AES-CBC-256 encryption operation is stored in this buffer*29.	Output

2.3.29.4 Return value

None.

*29 The size of this buffer needs to be guaranteed to be 16bytes aligned.

2.3.30 aes_cbc256_hard_decrypt_dma

2.3.30.1 Description

AES-CBC-256 decryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA. CBC encryption encrypts plaintext according to a block of fixed size of 16 bytes, and fills it when the block size is insufficient.

2.3.30.2 Function prototype

```
void aes_cbc256_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num, uint8_t
     *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)
```

2.3.30.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-CBC-256 decryption operation structure, including decryption key and offset vector	Input
input_data	AES-CBC-256 ciphertext data to be decrypted	Input
input_len	AES-CBC-256 Length ofciphertext data to be decrypted	Input
output_data	The result of the AES-CBC-256 decryption operation is stored in this buffer[*30].	Output

2.3.30.4 Return value

None.

2.3.31 aes_gcm128_hard_encrypt_dma

2.3.31.1 Description

AES-GCM-128 encryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA.

[*30] The size of this buffer needs to be guaranteed to be 16bytes aligned.

2.3.31.2 Function prototype

```
void aes_gcm128_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num,
    gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *
    output_data, uint8_t *gcm_tag)
```

2.3.31.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	The structure of the AES-GCM-128 encryption operation, including the encryption key / offset vector / aad / aad length	Input
input_data	AES-GCM-128 plaintext data to be encrypted	Input
input_len	AES-GCM-128 length of plaintext data to be encrypted。	Input
output_data	The result of the AES-GCM-128 encryption operation is stored in this buffer[*31].	Output
gcm_tag	The tag after the AES-GCM-128 encryption operation is stored in this buffer[*32].	Output

2.3.31.4 Return value

None.

2.3.32 aes_gcm128_hard_decrypt_dma

2.3.32.1 Description

AES-GCM-128 decryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA.

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

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

2.3.32.2 Function prototype

```
void aes_gcm128_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num,
    gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *
    output_data, uint8_t *gcm_tag)
```

2.3.32.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	The structure of the AES-GCM-128 decryption operation, including the decryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-128 ciphertext data to be decrypted	Input
input_len	AES-GCM-128 Length ofciphertext data to be decrypted。	Input
output_data	The result of the AES-GCM-128 decryption operation is stored in this buffer[*33].	Output
gcm_tag	The tag after the AES-GCM-128 decryption operation is stored in this buffer[*34].	Output

2.3.32.4 Return value

None.

2.3.33 aes_gcm192_hard_encrypt_dma

2.3.33.1 Description

AES-GCM-192 encryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA.

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

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

2.3.33.2 Function prototype

```
void aes_gcm192_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num,
    gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *
    output_data, uint8_t *gcm_tag)
```

2.3.33.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	The structure of the AES-GCM-192 encryption operation, including the 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[*35].	Output
gcm_tag	The tag after the AES-GCM-192 encryption operation is stored in this buffer[*36].	Output

2.3.33.4 Return value

None.

2.3.34 aes_gcm192_hard_decrypt_dma

2.3.34.1 Description

AES-GCM-192 decryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA.

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

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

2.3.34.2 Function prototype

```
void aes_gcm192_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num,
    gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *
    output_data, uint8_t *gcm_tag)
```

2.3.34.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	The structure of the AES-GCM-192 decryption operation, including the decryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-192 ciphertext data to be decrypted	Input
input_len	AES-GCM-192 Length ofciphertext data to be decrypted。	Input
output_data	The result of the AES-GCM-192 decryption operation is stored in this buffer*37.	Output
gcm_tag	The tag after the AES-GCM-192 decryption operation is stored in this buffer*38.	Output

2.3.34.4 Return value

None.

2.3.35 aes_gcm256_hard_encrypt_dma

2.3.35.1 Description

AES-GCM-256 encryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA.

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

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

2.3.35.2 Function prototype

```
void aes_gcm256_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num,
    gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *
    output_data, uint8_t *gcm_tag)
```

2.3.35.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	The structure of the AES-GCM-256 encryption operation, including the encryption key / offset vector / aad / aad length	Input
input_data	AES-GCM-256 plaintext data to be encrypted	Input
input_len	AES-GCM-256 length of plaintext data to be encrypted。	Input
output_data	The result of the AES-GCM-256 encryption operation is stored in this buffer[*39].	Output
gcm_tag	The tag after the AES-GCM-256 encryption operation is stored in this buffer[*40].	Output

2.3.35.4 Return value

None.

2.3.36 aes_gcm256_hard_decrypt_dma

2.3.36.1 Description

AES-GCM-256 decryption operation. The Input data is transmitted using the CPU, and the Output data is transmitted using the DMA.

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

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

2.3.36.2 Function prototype

```
void aes_gcm256_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num,
    gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *
    output_data, uint8_t *gcm_tag)
```

2.3.36.3 Parameter

Parameter name	Description	Input or output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	The structure of the AES-GCM-256 decryption operation, including the decryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-256 ciphertext data to be decrypted	Input
input_len	AES-GCM-256 Length ofciphertext data to be decrypted。	Input
output_data	The result of the AES-GCM-256 decryption operation is stored in this buffer[*41].	Output
gcm_tag	The tag after the AES-GCM-256 decryption operation is stored in this buffer[*42].	Output

2.3.36.4 Return value

None.

2.3.37 aes_init

2.3.37.1 Description

Initialization of the AES hardware module

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

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

2.3.37.2 Function prototype

```
void aes_init(uint8_t *input_key, size_t input_key_len, uint8_t *iv,size_t iv_len,
    uint8_t *gcm_aad,aes_cipher_mode_t cipher_mode, aes_encrypt_sel_t encrypt_sel,
    size_t gcm_aad_len, size_t input_data_len)
```

2.3.37.3 Parameter

Parameter name	Description	Input or output
input_key	Encryption/decryption key	Input
input_key_len	The length of the key to be used for encryption/decryption	Input
iv	AES encryption and decryption iv data	Input
iv_len	The length of the iv data used for AES encryption and decryption, CBC is fixed to 16bytes, and GCM is fixed to 12bytes.	Output
gcm_aad	aad data used by AES-GCM encryption and decryption	Output
cipher_mode	The type of encryption and decryption performed by the AES hardware module, supporting AES_CBC/AES_ECB/AES_GCM	Input
encrypt_sel	Execution mode of the AES hardware module: encryption or decryption	Input
gcm_aad_len	Length of aad data used by AES-GCM encryption and decryption	Input
input_data_len	Length of data to be encrypted/decrypted	Input

2.3.37.4 Return value

None.

2.3.38 aes_process

2.3.38.1 Description

AES hardware module performs encryption and decryption operations

2.3.38.2 Function prototype

```
void aes_process(uint8_t *input_data, uint8_t *output_data, size_t input_data_len,
    aes_cipher_mode_t cipher_mode)
```

2.3.38.3 Parameter

Parameter name	Description	Input or output
input_data	This buffer stores the data to be encrypted/decrypted	Input
output_data	This buffer stores the output result of encryption/decryption	Output
input_data_len	Length of data to be encrypted/decrypted	Input
cipher_mode	Encryption and decryption type performed by AES hardware module, supporting AES_CBC/AES_ECB/AES_GCM	Input

2.3.38.4 Return value

None.

2.3.39 gcm_get_tag

2.3.39.1 Description

Get the tag after the AES-GCM calculation is completed

2.3.39.2 Function prototype

```
void gcm_get_tag(uint8_t *gcm_tag)
```

2.3.39.3 Parameter

Parameter name	Description	Input or output
gcm_tag	This buffer stores the AES-GCM encrypted/decrypted tag, fixed to a size of 16bytes.	Output

2.3.39.4 Return value

None.

2.3.40 Example

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

2.4 Data type

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

- aes_cipher_mode_t : AES encryption/decryption mode.

2.4.1 aes_cipher_mode_t

2.4.1.1 Description

AES encryption/decryption mode.

2.4.1.2 Type definition

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

- gcm_context_t : AES-GCM structure used for parameters during encryption/decryption

2.4.2 gcm_context_t

2.4.2.1 Description

The structure used by the AES-GCM parameters, including the key, offset vector, aad data, and aad data length.

2.4.2.2 Type definition

```
typedef struct _gcm_context
{
    uint8_t *input_key;
    uint8_t *iv;
    uint8_t *gcm_aad;
    size_t gcm_aad_len;
} gcm_context_t;
```

- cbc_context_t : AES-CBC structure used for parameters during encryption/decryption

2.4.3 cbc_context_t

2.4.3.1 Description

The structure used by the AES-CBC parameter, including the key and offset vector.

2.4.3.2 Type definition

```
typedef struct _cbc_context
{
    uint8_t *input_key;
    uint8_t *iv;
} cbc_context_t;
```

2.4.3.3 Enumeration element

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

Chapter 3

PLIC

3.1 Overview

Platform-Level Interrupt Controller (PLIC).

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.

3.2 Features

The PLIC module has the following features:

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

3.3 API

Corresponding header file `plic.h`

Provide the following interfaces

- plic_init
- plic_irq_enable
- plic_irq_disable
- plic_set_priority
- plic_get_priority

- plic_irq_register
- plic_irq_deregister

3.3.1 plic_init

3.3.1.1 Description

Initializes PLIC external interrupt.

3.3.1.2 Function prototype

```
void plic_init(void)
```

3.3.1.3 Parameter

None.

3.3.1.4 Return value

None.

3.3.2 plic_irq_enable

3.3.2.1 Description

Enable external interrupts.

3.3.2.2 Function prototype

```
int plic_irq_enable(plic_irq_t irq_number)
```

3.3.2.3 Parameter

Parameter name	Description	Input or output
irq_number	Interrupt number	Input

3.3.2.4 Return value

Return value	Description
0	Success

Return value	Description
Others	Fail

3.3.3 plic_irq_disable

3.3.3.1 Description

Disable external interrupts.

3.3.3.2 Function prototype

```
int plic_irq_disable(plic_irq_t irq_number)
```

3.3.3.3 Parameter

Parameter name	Description	Input or output
irq_number	Interrupt number	Input

3.3.3.4 Return value

Return value	Description
0	Success
Others	Fail

3.3.4 plic_set_priority

3.3.4.1 Description

Set the interrupt priority.

3.3.4.2 Function prototype

```
int plic_set_priority(plic_irq_t irq_number, uint32_t priority)
```

3.3.4.3 Parameter

Parameter name	Description	Input or output
irq_number	Interrupt number	Input
priority	Interrupt priority	Input

3.3.4.4 Return value

Return value	Description
0	Success
Others	Fail

3.3.5 plic_get_priority

3.3.5.1 Description

Get the interrupt priority.

3.3.5.2 Function prototype

```
uint32_t plic_get_priority(plic_irq_t irq_number)
```

3.3.5.3 Parameter

Parameter name	Description	Input or output
irq_number	Interrupt number	Input

3.3.5.4 Return value

The priority of the interrupt whose interrupt number is irq_number.

3.3.6 plic_irq_register

3.3.6.1 Description

Register an external interrupt function.

3.3.6.2 Function prototype

```
int plic_irq_register(plic_irq_t irq, plic_irq_callback_t callback, void* ctx)
```

3.3.6.3 Parameter

Parameter name	Description	Input or output
irq	Interrupt number	Input
callback	Interrupt callback function	Input
ctx	Parameter of callback function	Input

3.3.6.4 Return value

Return value	Description
0	Success
Others	Fail

3.3.7 plic_irq_deregister

3.3.7.1 Description

Deregister the external interrupt function.

3.3.7.2 Function prototype

```
int plic_irq_deregister(plic_irq_t irq)
```

3.3.7.3 Parameter

Parameter name	Description	Input or output
irq	Interrupt number	Input

3.3.7.4 Return value

Return value	Description
0	Success
Others	Fail

3.3.8 Example

```
/* Set the trigger interrupt of GPIOHS0 */
int count = 0;
int gpiohs_pin_onchange_isr(void *ctx)
{
    int *userdata = (int *)ctx;
    *userdata++;
}
plic_init();
plic_set_priority(IRQN_GPIOHS0_INTERRUPT, 1);
plic_irq_register(IRQN_GPIOHS0_INTERRUPT, gpiohs_pin_onchange_isr, &count);
plic_irq_enable(IRQN_GPIOHS0_INTERRUPT);
sysctl_enable_irq();
```

3.4 Data type

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

- plic_irq_t: External interrupt number.
- plic_irq_callback_t: External interrupt callback function.

3.4.1 plic_irq_t

3.4.1.1 Description

External interrupt number.

3.4.1.2 Type definition

```
typedef enum _plic_irq
{
    IRQN_NO_INTERRUPT        = 0, /*!< The non-existent interrupt */
    IRQN_SPI0_INTERRUPT      = 1, /*!< SPI0 interrupt */
    IRQN_SPI1_INTERRUPT      = 2, /*!< SPI1 interrupt */
    IRQN_SPI_SLAVE_INTERRUPT = 3, /*!< SPI_SLAVE interrupt */
    IRQN_SPI3_INTERRUPT      = 4, /*!< SPI3 interrupt */
    IRQN_I2S0_INTERRUPT      = 5, /*!< I2S0 interrupt */
    IRQN_I2S1_INTERRUPT      = 6, /*!< I2S1 interrupt */
    IRQN_I2S2_INTERRUPT      = 7, /*!< I2S2 interrupt */
    IRQN_I2C0_INTERRUPT      = 8, /*!< I2C0 interrupt */
    IRQN_I2C1_INTERRUPT      = 9, /*!< I2C1 interrupt */
    IRQN_I2C2_INTERRUPT      = 10, /*!< I2C2 interrupt */
    IRQN_UART1_INTERRUPT     = 11, /*!< UART1 interrupt */
    IRQN_UART2_INTERRUPT     = 12, /*!< UART2 interrupt */
```

```
    IRQN_UART3_INTERRUPT     = 13, /*!< UART3 interrupt */
    IRQN_TIMER0A_INTERRUPT   = 14, /*!< TIMER0 channel 0 or 1 interrupt */
    IRQN_TIMER0B_INTERRUPT   = 15, /*!< TIMER0 channel 2 or 3 interrupt */
    IRQN_TIMER1A_INTERRUPT   = 16, /*!< TIMER1 channel 0 or 1 interrupt */
    IRQN_TIMER1B_INTERRUPT   = 17, /*!< TIMER1 channel 2 or 3 interrupt */
    IRQN_TIMER2A_INTERRUPT   = 18, /*!< TIMER2 channel 0 or 1 interrupt */
    IRQN_TIMER2B_INTERRUPT   = 19, /*!< TIMER2 channel 2 or 3 interrupt */
    IRQN_RTC_INTERRUPT       = 20, /*!< RTC tick and alarm interrupt */
    IRQN_WDT0_INTERRUPT      = 21, /*!< Watching dog timer0 interrupt */
    IRQN_WDT1_INTERRUPT      = 22, /*!< Watching dog timer1 interrupt */
    IRQN_APB_GPIO_INTERRUPT  = 23, /*!< APB GPIO interrupt */
    IRQN_DVP_INTERRUPT       = 24, /*!< Digital video port interrupt */
    IRQN_AI_INTERRUPT        = 25, /*!< AI accelerator interrupt */
    IRQN_FFT_INTERRUPT       = 26, /*!< FFT accelerator interrupt */
    IRQN_DMA0_INTERRUPT      = 27, /*!< DMA channel0 interrupt */
    IRQN_DMA1_INTERRUPT      = 28, /*!< DMA channel1 interrupt */
    IRQN_DMA2_INTERRUPT      = 29, /*!< DMA channel2 interrupt */
    IRQN_DMA3_INTERRUPT      = 30, /*!< DMA channel3 interrupt */
    IRQN_DMA4_INTERRUPT      = 31, /*!< DMA channel4 interrupt */
    IRQN_DMA5_INTERRUPT      = 32, /*!< DMA channel5 interrupt */
    IRQN_UARTHS_INTERRUPT    = 33, /*!< Hi-speed UART0 interrupt */
    IRQN_GPIOHS0_INTERRUPT   = 34, /*!< Hi-speed GPIO0 interrupt */
    IRQN_GPIOHS1_INTERRUPT   = 35, /*!< Hi-speed GPIO1 interrupt */
    IRQN_GPIOHS2_INTERRUPT   = 36, /*!< Hi-speed GPIO2 interrupt */
    IRQN_GPIOHS3_INTERRUPT   = 37, /*!< Hi-speed GPIO3 interrupt */
    IRQN_GPIOHS4_INTERRUPT   = 38, /*!< Hi-speed GPIO4 interrupt */
    IRQN_GPIOHS5_INTERRUPT   = 39, /*!< Hi-speed GPIO5 interrupt */
    IRQN_GPIOHS6_INTERRUPT   = 40, /*!< Hi-speed GPIO6 interrupt */
    IRQN_GPIOHS7_INTERRUPT   = 41, /*!< Hi-speed GPIO7 interrupt */
    IRQN_GPIOHS8_INTERRUPT   = 42, /*!< Hi-speed GPIO8 interrupt */
    IRQN_GPIOHS9_INTERRUPT   = 43, /*!< Hi-speed GPIO9 interrupt */
    IRQN_GPIOHS10_INTERRUPT  = 44, /*!< Hi-speed GPIO10 interrupt */
    IRQN_GPIOHS11_INTERRUPT  = 45, /*!< Hi-speed GPIO11 interrupt */
    IRQN_GPIOHS12_INTERRUPT  = 46, /*!< Hi-speed GPIO12 interrupt */
    IRQN_GPIOHS13_INTERRUPT  = 47, /*!< Hi-speed GPIO13 interrupt */
    IRQN_GPIOHS14_INTERRUPT  = 48, /*!< Hi-speed GPIO14 interrupt */
    IRQN_GPIOHS15_INTERRUPT  = 49, /*!< Hi-speed GPIO15 interrupt */
    IRQN_GPIOHS16_INTERRUPT  = 50, /*!< Hi-speed GPIO16 interrupt */
    IRQN_GPIOHS17_INTERRUPT  = 51, /*!< Hi-speed GPIO17 interrupt */
    IRQN_GPIOHS18_INTERRUPT  = 52, /*!< Hi-speed GPIO18 interrupt */
    IRQN_GPIOHS19_INTERRUPT  = 53, /*!< Hi-speed GPIO19 interrupt */
    IRQN_GPIOHS20_INTERRUPT  = 54, /*!< Hi-speed GPIO20 interrupt */
    IRQN_GPIOHS21_INTERRUPT  = 55, /*!< Hi-speed GPIO21 interrupt */
    IRQN_GPIOHS22_INTERRUPT  = 56, /*!< Hi-speed GPIO22 interrupt */
    IRQN_GPIOHS23_INTERRUPT  = 57, /*!< Hi-speed GPIO23 interrupt */
    IRQN_GPIOHS24_INTERRUPT  = 58, /*!< Hi-speed GPIO24 interrupt */
    IRQN_GPIOHS25_INTERRUPT  = 59, /*!< Hi-speed GPIO25 interrupt */
    IRQN_GPIOHS26_INTERRUPT  = 60, /*!< Hi-speed GPIO26 interrupt */
    IRQN_GPIOHS27_INTERRUPT  = 61, /*!< Hi-speed GPIO27 interrupt */
    IRQN_GPIOHS28_INTERRUPT  = 62, /*!< Hi-speed GPIO28 interrupt */
    IRQN_GPIOHS29_INTERRUPT  = 63, /*!< Hi-speed GPIO29 interrupt */
    IRQN_GPIOHS30_INTERRUPT  = 64, /*!< Hi-speed GPIO30 interrupt */
    IRQN_GPIOHS31_INTERRUPT  = 65, /*!< Hi-speed GPIO31 interrupt */
```

```
    IRQN_MAX
} plic_irq_t;
```

3.4.1.3 Enumeration element

Element name	Description
IRQN_NO_INTERRUPT	The non-existent interrupt
IRQN_SPI0_INTERRUPT	SPI0 interrupt
IRQN_SPI1_INTERRUPT	SPI1 interrupt
IRQN_SPI_SLAVE_INTERRUPT	SPI_SLAVE interrupt
IRQN_SPI3_INTERRUPT	SPI3 interrupt
IRQN_I2S0_INTERRUPT	I2S0 interrupt
IRQN_I2S1_INTERRUPT	I2S1 interrupt
IRQN_I2S2_INTERRUPT	I2S2 interrupt
IRQN_I2C0_INTERRUPT	I2C0 interrupt
IRQN_I2C1_INTERRUPT	I2C1 interrupt
IRQN_I2C2_INTERRUPT	I2C2 interrupt
IRQN_UART1_INTERRUPT	UART1 interrupt
IRQN_UART2_INTERRUPT	UART2 interrupt
IRQN_UART3_INTERRUPT	UART3 interrupt
IRQN_TIMER0A_INTERRUPT	TIMER0 channel 0 or 1 interrupt
IRQN_TIMER0B_INTERRUPT	TIMER0 channel 2 or 3 interrupt
IRQN_TIMER1A_INTERRUPT	TIMER1 channel 0 or 1 interrupt
IRQN_TIMER1B_INTERRUPT	TIMER1 channel 2 or 3 interrupt
IRQN_TIMER2A_INTERRUPT	TIMER2 channel 0 or 1 interrupt
IRQN_TIMER2B_INTERRUPT	TIMER2 channel 2 or 3 interrupt
IRQN_RTC_INTERRUPT	RTC tick and alarm interrupt
IRQN_WDT0_INTERRUPT	Watching dog timer0 interrupt
IRQN_WDT1_INTERRUPT	Watching dog timer1 interrupt
IRQN_APB_GPIO_INTERRUPT	APB GPIO interrupt
IRQN_DVP_INTERRUPT	Digital video port interrupt
IRQN_AI_INTERRUPT	AI accelerator interrupt
IRQN_FFT_INTERRUPTFFT	FFT accelerator interrupt
IRQN_DMA0_INTERRUPT	DMA channel0 interrupt
IRQN_DMA1_INTERRUPT	DMA channel1 interrupt
IRQN_DMA2_INTERRUPT	DMA channel2 interrupt
IRQN_DMA3_INTERRUPT	DMA channel3 interrupt

Element name	Description
IRQN_DMA4_INTERRUPT	DMA channel4 interrupt
IRQN_DMA5_INTERRUPT	DMA channel5 interrupt
IRQN_UARTHS_INTERRUPT	Hi-speed UART0 interrupt
IRQN_GPIOHS0_INTERRUPT	Hi-speed GPIO0 interrupt
IRQN_GPIOHS1_INTERRUPT	Hi-speed GPIO1 interrupt
IRQN_GPIOHS2_INTERRUPT	Hi-speed GPIO2 interrupt
IRQN_GPIOHS3_INTERRUPT	Hi-speed GPIO3 interrupt
IRQN_GPIOHS4_INTERRUPT	Hi-speed GPIO4 interrupt
IRQN_GPIOHS5_INTERRUPT	Hi-speed GPIO5 interrupt
IRQN_GPIOHS6_INTERRUPT	Hi-speed GPIO6 interrupt
IRQN_GPIOHS7_INTERRUPT	Hi-speed GPIO7 interrupt
IRQN_GPIOHS8_INTERRUPT	Hi-speed GPIO8 interrupt
IRQN_GPIOHS9_INTERRUPT	Hi-speed GPIO9 interrupt
IRQN_GPIOHS10_INTERRUPT	Hi-speed GPIO10 interrupt
IRQN_GPIOHS11_INTERRUPT	Hi-speed GPIO11 interrupt
IRQN_GPIOHS12_INTERRUPT	Hi-speed GPIO12 interrupt
IRQN_GPIOHS13_INTERRUPT	Hi-speed GPIO13 interrupt
IRQN_GPIOHS14_INTERRUPT	Hi-speed GPIO14 interrupt
IRQN_GPIOHS15_INTERRUPT	Hi-speed GPIO15 interrupt
IRQN_GPIOHS16_INTERRUPT	Hi-speed GPIO16 interrupt
IRQN_GPIOHS17_INTERRUPT	Hi-speed GPIO17 interrupt
IRQN_GPIOHS18_INTERRUPT	Hi-speed GPIO18 interrupt
IRQN_GPIOHS19_INTERRUPT	Hi-speed GPIO19 interrupt
IRQN_GPIOHS20_INTERRUPT	Hi-speed GPIO20 interrupt
IRQN_GPIOHS21_INTERRUPT	Hi-speed GPIO21 interrupt
IRQN_GPIOHS22_INTERRUPT	Hi-speed GPIO22 interrupt
IRQN_GPIOHS23_INTERRUPT	Hi-speed GPIO23 interrupt
IRQN_GPIOHS24_INTERRUPT	Hi-speed GPIO24 interrupt
IRQN_GPIOHS25_INTERRUPT	Hi-speed GPIO25 interrupt
IRQN_GPIOHS26_INTERRUPT	Hi-speed GPIO26 interrupt
IRQN_GPIOHS27_INTERRUPT	Hi-speed GPIO27 interrupt
IRQN_GPIOHS28_INTERRUPT	Hi-speed GPIO28 interrupt
IRQN_GPIOHS29_INTERRUPT	Hi-speed GPIO29 interrupt
IRQN_GPIOHS30_INTERRUPT	Hi-speed GPIO30 interrupt
IRQN_GPIOHS31_INTERRUPT	Hi-speed GPIO31 interrupt

3.4.2 plic_irq_callback_t

3.4.2.1 Description

External interrupt callback function.

3.4.2.2 Type definition

```
typedef int (*plic_irq_callback_t)(void *ctx);
```

Chapter 4

GPIO

4.1 Overview

GPIO stands for General Purpose Input Output. The chip has 8 general purpose GPIOs.

4.2 Features

The GPIO unit has the following features:

- Configurable up and down drive mode

4.3 API

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

- gpio_init
- gpio_set_drive_mode
- gpio_set_pin
- gpio_get_pin

4.3.1 gpio_init

4.3.1.1 Description

Initialize GPIO.

4.3.1.2 Function prototype

```
int gpio_init(void)
```

4.3.1.3 Return value

Return value	Description
0	Success
Others	Fail

4.3.2 gpio_set_drive_mode

4.3.2.1 Description

Set GPIO drive mode.

4.3.2.2 Function prototype

```
void gpio_set_drive_mode(uint8_t pin, gpio_drive_mode_t mode)
```

4.3.2.3 Parameter

Parameter name	Description	Input or output
pin	GPIO pin	Input
mode	GPIO driver mode	Input

4.3.2.4 Return value

None.

4.3.3 gpio_set_pin

4.3.3.1 Description

Set GPIO pin value.

4.3.3.2 Function prototype

```
void gpio_set_pin(uint8_t pin, gpio_pin_value_t value)
```

4.3.3.3 Parameter

Parameter name	Description	Input or output
pin	GPIO pin	Input
value	GPIO value	Input

4.3.3.4 Return value

None.

4.3.4 gpio_get_pin

4.3.4.1 Description

Get GPIO pin value.

4.3.4.2 Function prototype

```
gpio_pin_value_t gpio_get_pin(uint8_t pin)
```

4.3.4.3 Parameter

Parameter name	Description	Input or output
pin	GPIO pin	Input

4.3.4.4 Return value

The result of GPIO pin value.

4.3.5 Example

```
/* Set IO13 to Output and set it high */
gpio_init();
fpioa_set_function(13, FUNC_GPIO3);
gpio_set_drive_mode(3, GPIO_DM_OUTPUT);
gpio_set_pin(3, GPIO_PV_High);
```

4.4 Data type

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

- gpio_drive_mode_t: GPIO drive mode.
- gpio_pin_value_t: GPIO value.

4.4.1 gpio_drive_mode_t

4.4.1.1 Description

GPIO drive mode.

4.4.1.2 Type definition

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

4.4.1.3 Enumeration element

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

4.4.2 gpio_pin_value_t

4.4.2.1 Description

GPIO value.

4.4.2.2 Type definition

```
typedef enum _gpio_pin_value
```

```
{
    GPIO_PV_LOW,
    GPIO_PV_HIGH
} gpio_pin_value_t;
```

4.4.2.3 Enumeration element

Element name	Description
GPIO_PV_LOW	Low
GPIO_PV_HIGH	High

Chapter 5

GPIOHS

5.1 Overview

GPIO stands for General Purpose Input Output. HS stands for high speed. The chip has 32 high speed GPIOs. Similar to normal GPIO, but faster.

5.2 Features

The GPIO unit has the following features:

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

5.3 API

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

- gpiohs_set_drive_mode
- gpiohs_set_pin
- gpiohs_get_pin
- gpiohs_set_pin_edge
- gpiohs_set_irq

5.3.1 gpiohs_set_drive_mode

5.3.1.1 Description

Set GPIO drive mode.

5.3.1.2 Function prototype

```
void gpiohs_set_drive_mode(uint8_t pin, gpio_drive_mode_t mode)
```

5.3.1.3 Parameter

Parameter name	Description	Input or output
pin	GPIO pin	Input
mode	GPIO driver mode	Input

5.3.1.4 Return value

None.

5.3.2 gpio_set_pin

5.3.2.1 Description

Set GPIO pin value.

5.3.2.2 Function prototype

```
void gpiohs_set_pin(uint8_t pin, gpio_pin_value_t value)
```

5.3.2.3 Parameter

Parameter name	Description	Input or output
pin	GPIO pin	Input
value	GPIO value	Input

5.3.2.4 Return value

None.

5.3.3 gpio_get_pin

5.3.3.1 Description

Get GPIO pin value.

5.3.3.2 Function prototype

```
gpio_pin_value_t gpiohs_get_pin(uint8_t pin)
```

5.3.3.3 Parameter

Parameter name	Description	Input or output
pin	GPIO pin	Input

5.3.3.4 Return value

The result of GPIO pin value.

5.3.4 gpiohs_set_pin_edge

5.3.4.1 Description

Set the high speed GPIO interrupt trigger mode.

5.3.4.2 Function prototype

```
void gpiohs_set_pin_edge(uint8_t pin, gpio_pin_edge_t edge)
```

5.3.4.3 Parameter

Parameter name	Description	Input or output
pin	GPIO pin	Input
edge	Interrupt trigger mode	Input

5.3.4.4 Return value

None.

5.3.5 gpiohs_set_irq

5.3.5.1 Description

Set the interrupt callback function for high speed GPIO.

5.3.5.2 Function prototype

```
void gpiohs_set_irq(uint8_t pin, uint32_t priority, void(*func)());
```

5.3.5.3 Parameter

Parameter name	Description	Input or output
pin	GPIO pin	Input
priority	Interrupt priority	Input
func	Interrupt callback function	Input

5.3.5.4 Return value

None.

5.3.6 Example

```
void irq_gpiohs2(void)
{
    printf("Hello world\n");
}
/* Set IO13 to high speed GPIO and set the output mode to high. */
fpioa_set_function(13, FUNC_GPIOHS3);
gpiohs_set_drive_mode(3, GPIO_DM_OUTPUT);
gpiohs_set_pin(3, GPIO_PV_High);
/* Set IO14 to high speed GPIO Input mode. Print Hello world when double edge triggers
    interrupt. */
plic_init();
fpioa_set_function(14, FUNC_GPIOHS2);
gpiohs_set_drive_mode(2, GPIO_DM_INPUT);
gpiohs_set_pin_edge(2, GPIO_PE_BOTH);
gpiohs_set_irq(2, 1, irq_gpiohs2);
sysctl_enable_irq();
```

5.4 Data type

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

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

5.4.1 gpio_drive_mode_t

5.4.1.1 Description

GPIO drive mode.

5.4.1.2 Type definition

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

5.4.1.3 Enumeration element

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

5.4.2 gpio_pin_value_t

5.4.2.1 Description

GPIO value.

5.4.2.2 Type definition

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

5.4.2.3 Enumeration element

Element name	Description
GPIO_PV_LOW	Low
GPIO_PV_HIGH	High

5.4.3 gpio_pin_edge_t

5.4.3.1 Description

GPIO edge trigger mode.

5.4.3.2 Type definition

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

5.4.3.3 Enumeration element

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

Chapter 6

FPIOA

6.1 Overview

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

6.2 Features

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

6.3 API

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

- fpioa_set_function
- fpioa_get_io_by_function
- fpioa_set_io
- fpioa_get_io
- fpioa_set_tie_enable

- fpioa_set_tie_value
- fpioa_set_io_pull
- fpioa_get_io_pull
- fpioa_set_io_driving
- fpioa_get_io_driving

6.3.1 fpioa_set_function

6.3.1.1 Description

Set the IO0-IO47 pin multiplexing function.

6.3.1.2 Function prototype

```
int fpioa_set_function(int number, fpioa_function_t function)
```

6.3.1.3 Parameter

Parameter name	Description	Input or output
number	I/O pin number	Input
function	FPIOA function number	Input

6.3.1.4 Return value

Return value	Description
0	Success
Others	Fail

6.3.2 fpioa_get_io_by_function

6.3.2.1 Description

Get the I/O pin number according to the function number.

6.3.2.2 Function prototype

```
int fpioa_get_io_by_function(fpioa_function_t function)
```

6.3.2.3 Parameter

Parameter name	Description	Input or output
function	FPIOA function number	Input

6.3.2.4 Return value

Return value	Description
Greater than or equal to 0	I/O pin number
Less than 0	Fail

6.3.3 fpioa_get_io

6.3.3.1 Description

Get the configuration of the I/O pin.

6.3.3.2 Function prototype

```
int fpioa_get_io(int number, fpioa_io_config_t *cfg)
```

6.3.3.3 Parameter

Parameter name	Description	Input or output
number	I/O pin number	Input
cfg	Pin function structure	Output

6.3.3.4 Return value

Return value	Description
0	Success
Others	Fail

6.3.4 fpioa_set_io

6.3.4.1 Description

Set the configuration of the I/O pin.

6.3.4.2 Function prototype

```
int fpioa_set_io(int number, fpioa_io_config_t *cfg)
```

6.3.4.3 Parameter

Parameter name	Description	Input or output
number	I/O pin number	Input
cfg	Pin function structure	Input

6.3.4.4 Return value

Return value	Description
0	Success
Others	Fail

6.3.5 fpioa_set_tie_enable

6.3.5.1 Description

Enable or disable force the input level function of FPIOA function input signal.

6.3.5.2 Function prototype

```
int fpioa_set_tie_enable(fpioa_function_t function, int enable)
```

6.3.5.3 Parameter

Parameter name	Description	Input or output
function	FPIOA function number	Input
enable	Force input level enable setting, 0: Disable 1: Enable	Input

6.3.5.4 Return value

Return value	Description
0	Success
Others	Fail

6.3.6 fpioa_set_tie_value

6.3.6.1 Description

Set the input level to be high or low of FPIOA function input signal, only valid when the forced input level function is enabled.

6.3.6.2 Function prototype

```
int fpioa_set_tie_value(fpioa_function_t function, int value)
```

6.3.6.3 Parameter

Parameter name	Description	Input or output
function	FPIOA function number	Input
value	Forced input level value 0: Low 1: High	Input

6.3.6.4 Return value

Return value	Description
0	Success
Others	Fail

6.3.7 fpioa_set_pull

6.3.7.1 Description

Set the pullup or pulldown of I/O.

6.3.7.2 Function prototype

```
int fpioa_set_io_pull(int number, fpioa_pull_t pull)
```

6.3.7.3 Parameter

Parameter name	Description	Input or output
number	I/O number	Input
pull	Pull up or pull down	Input

6.3.7.4 Return value

Return value	Description
0	Success
Others	Fail

6.3.8 fpioa_get_pull

6.3.8.1 Description

Get the I/O pin pull-up or pull-down status.

6.3.8.2 Function prototype

```
int fpioa_get_io_pull(int number)
```

6.3.8.3 Parameter

Parameter name	Description	Input or output
number	I/O number	Input

6.3.8.4 Return value

Return value	Description
0	None
1	Pull down
2	Pull up

6.3.9 fpioa_set_io_driving

6.3.10 Description

Set the driving strength of the I/O pin.

6.3.10.1 Function prototype

```
int fpioa_set_io_driving(int number, fpioa_driving_t driving)
```

6.3.10.2 Parameter

Parameter name	Description	Input or output
number	I/O number	Input
driving	Driving strength	Input

6.3.10.3 Return value

Return value	Description
0	Success
Others	Fail

6.3.11 fpioa_get_io_driving

6.3.11.1 Description

Get driving strength

6.3.11.2 Function prototype

```
int fpioa_get_io_driving(int number)
```

6.3.11.3 Parameter

Parameter name	Description	Input or output
number	I/O number	Input

6.3.11.4 Return value

Return value	Description
Greater than or equal to 0	Driving strength
Less than 0	Fail

6.4 Data type

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

- fpioa_function_t: The function number of the pin.
- fpioa_io_config_t: FPIOA configuration.
- fpioa_pull_t: FPIOA pull-up or pull-down feature.
- fpioa_driving_t: FPIOA driving strength number.

6.4.1 fpioa_io_config_t

6.4.1.1 Description

FPIOA configuration.

6.4.1.2 Type definition

```
typedef struct _fpioa_io_config
{
    uint32_t ch_sel : 8;
    uint32_t ds : 4;
    uint32_t oe_en : 1;
    uint32_t oe_inv : 1;
    uint32_t do_sel : 1;
    uint32_t do_inv : 1;
    uint32_t pu : 1;
    uint32_t pd : 1;
    uint32_t resv0 : 1;
    uint32_t sl : 1;
    uint32_t ie_en : 1;
    uint32_t ie_inv : 1;
    uint32_t di_inv : 1;
    uint32_t st : 1;
    uint32_t resv1 : 7;
    uint32_t pad_di : 1;
} __attribute__((packed, aligned(4))) fpioa_io_config_t;
```

6.4.1.3 Enumeration element

Element name	Description
ch \ _sel	FPIOA function number, choose one of 256 functions
ds	Driving current strength selection. Check driving current table.
oe \ _en	Output Enable (OE) (Manual Mode). 1: Enable 0: Disable
oe \ _inv	Output enable inversion. 1: Enable 0: Disable
do \ _sel	Output signal selection. 1: Output OE signal 0: Output original signal
do\ _inv	Output signal inversion. 1: Enable 0: Disable
pu	Pull-up enable. 1: Enable 0: Disable
pd	Pull-down enable.1: Enable 0: Disable
resv0	Reserved bits
sl	Output slew rate control. 1: Low slew rate, stable 0: High slew rate, fast
ie \ _en	Input Enable (IE) (manual mode). 1: Enable 0: Disable
that is \ _inv	Input enable inversion. 1: Enable 0: Disable
di \ _inv	Input signal inversion. 1: Enable 0: Disable
st	Input Schmitt trigger. 1: Enable 0: Disable
resv1	Reserved bits
pad \ _di	Current pin input value

6.4.2 Driving strength selection table

6.4.2.1 Low level sinking (input) current

ds	Min(mA)	Typ(mA)	Max(mA)
0	3.2	5.4	8.3
1	4.7	8.0	12.3
2	6.3	10.7	16.4
3	7.8	13.2	20.2
4	9.4	15.9	24.2
5	10.9	18.4	28.1
6	12.4	20.9	31.8
7	13.9	23.4	35.5

6.4.2.2 High level sourcing (output) current

ds	Min(mA)	Typ(mA)	Max(mA)
0	5.0	7.6	11.2
1	7.5	11.4	16.8
2	10.0	15.2	22.3
3	12.4	18.9	27.8
4	14.9	22.6	33.3
5	17.4	26.3	38.7
6	19.8	30.0	44.1
7	22.3	33.7	49.5

6.4.3 fpioa_pull_t

6.4.3.1 Description

FPIOA pull-up or pull-down feature.

6.4.3.2 Type definition

```
typedef enum _fpioa_pull
{
    FPIOA_PULL_NONE,
    FPIOA_PULL_DOWN,
    FPIOA_PULL_UP,
    FPIOA_PULL_MAX
} fpioa_pull_t;
```

6.4.3.3 Enumeration element

Element name	Description
FPIOA_PULL_NONE	None.
FPIOA_PULL_DOWN	Pull down.
FPIOA_PULL_UP	Pull up.

It is forbidden to enable pull-up and pull-down inside the chip at the same time.

6.4.4 fpioa_driving_t

6.4.4.1 Description

FPIOA driving strength number, see drive strength selection table.

6.4.4.2 Type definition

```
typedef enum _fpioa_driving
{
    FPIOA_DRIVING_0,
    FPIOA_DRIVING_1,
    FPIOA_DRIVING_2,
    FPIOA_DRIVING_3,
    FPIOA_DRIVING_4,
    FPIOA_DRIVING_5,
    FPIOA_DRIVING_6,
    FPIOA_DRIVING_7,
} fpioa_driving_t;
```

6.4.4.3 Enumeration element

Element name	Description
FPIOA_DRIVING_0	Driving strength selection 0
FPIOA_DRIVING_1	Driving strength selection 1
FPIOA_DRIVING_2	Driving strength selection 2
FPIOA_DRIVING_3	Driving strength selection 3
FPIOA_DRIVING_4	Driving strength selection 4
FPIOA_DRIVING_5	Driving strength selection 5
FPIOA_DRIVING_6	Driving strength selection 6
FPIOA_DRIVING_7	Driving strength selection 7

6.4.5 fpioa_function_t

6.4.5.1 Description

The function number of the pin.

6.4.5.2 Type definition

```
typedef enum _fpioa_function
```

```
{
    FUNC_JTAG_TCLK          = 0,      /*!< JTAG Test Clock */
    FUNC_JTAG_TDI           = 1,      /*!< JTAG Test Data In */
    FUNC_JTAG_TMS           = 2,      /*!< JTAG Test Mode Select */
    FUNC_JTAG_TDO           = 3,      /*!< JTAG Test Data Out */
    FUNC_SPI0_D0            = 4,      /*!< SPI0 Data 0 */
    FUNC_SPI0_D1            = 5,      /*!< SPI0 Data 1 */
    FUNC_SPI0_D2            = 6,      /*!< SPI0 Data 2 */
    FUNC_SPI0_D3            = 7,      /*!< SPI0 Data 3 */
    FUNC_SPI0_D4            = 8,      /*!< SPI0 Data 4 */
    FUNC_SPI0_D5            = 9,      /*!< SPI0 Data 5 */
    FUNC_SPI0_D6            = 10,     /*!< SPI0 Data 6 */
    FUNC_SPI0_D7            = 11,     /*!< SPI0 Data 7 */
    FUNC_SPI0_SS0           = 12,     /*!< SPI0 Chip Select 0 */
    FUNC_SPI0_SS1           = 13,     /*!< SPI0 Chip Select 1 */
    FUNC_SPI0_SS2           = 14,     /*!< SPI0 Chip Select 2 */
    FUNC_SPI0_SS3           = 15,     /*!< SPI0 Chip Select 3 */
    FUNC_SPI0_ARB           = 16,     /*!< SPI0 Arbitration */
    FUNC_SPI0_SCLK          = 17,     /*!< SPI0 Serial Clock */
    FUNC_UARTHS_RX          = 18,     /*!< UART High speed Receiver */
    FUNC_UARTHS_TX          = 19,     /*!< UART High speed Transmitter */
    FUNC_CLK_IN1            = 20,     /*!< Clock Input 1 */
    FUNC_CLK_IN2            = 21,     /*!< Clock Input 2 */
    FUNC_CLK_SPI1           = 22,     /*!< Clock SPI1 */
    FUNC_CLK_I2C1           = 23,     /*!< Clock I2C1 */
    FUNC_GPIOHS0            = 24,     /*!< GPIO High speed 0 */
    FUNC_GPIOHS1            = 25,     /*!< GPIO High speed 1 */
    FUNC_GPIOHS2            = 26,     /*!< GPIO High speed 2 */
    FUNC_GPIOHS3            = 27,     /*!< GPIO High speed 3 */
    FUNC_GPIOHS4            = 28,     /*!< GPIO High speed 4 */
    FUNC_GPIOHS5            = 29,     /*!< GPIO High speed 5 */
    FUNC_GPIOHS6            = 30,     /*!< GPIO High speed 6 */
    FUNC_GPIOHS7            = 31,     /*!< GPIO High speed 7 */
    FUNC_GPIOHS8            = 32,     /*!< GPIO High speed 8 */
    FUNC_GPIOHS9            = 33,     /*!< GPIO High speed 9 */
    FUNC_GPIOHS10           = 34,     /*!< GPIO High speed 10 */
    FUNC_GPIOHS11           = 35,     /*!< GPIO High speed 11 */
    FUNC_GPIOHS12           = 36,     /*!< GPIO High speed 12 */
    FUNC_GPIOHS13           = 37,     /*!< GPIO High speed 13 */
    FUNC_GPIOHS14           = 38,     /*!< GPIO High speed 14 */
    FUNC_GPIOHS15           = 39,     /*!< GPIO High speed 15 */
    FUNC_GPIOHS16           = 40,     /*!< GPIO High speed 16 */
    FUNC_GPIOHS17           = 41,     /*!< GPIO High speed 17 */
    FUNC_GPIOHS18           = 42,     /*!< GPIO High speed 18 */
    FUNC_GPIOHS19           = 43,     /*!< GPIO High speed 19 */
    FUNC_GPIOHS20           = 44,     /*!< GPIO High speed 20 */
    FUNC_GPIOHS21           = 45,     /*!< GPIO High speed 21 */
    FUNC_GPIOHS22           = 46,     /*!< GPIO High speed 22 */
    FUNC_GPIOHS23           = 47,     /*!< GPIO High speed 23 */
    FUNC_GPIOHS24           = 48,     /*!< GPIO High speed 24 */
    FUNC_GPIOHS25           = 49,     /*!< GPIO High speed 25 */
    FUNC_GPIOHS26           = 50,     /*!< GPIO High speed 26 */
    FUNC_GPIOHS27           = 51,     /*!< GPIO High speed 27 */
```

```
    FUNC_GPIOHS28         = 52,     /*!< GPIO High speed 28 */
    FUNC_GPIOHS29         = 53,     /*!< GPIO High speed 29 */
    FUNC_GPIOHS30         = 54,     /*!< GPIO High speed 30 */
    FUNC_GPIOHS31         = 55,     /*!< GPIO High speed 31 */
    FUNC_GPIO0            = 56,     /*!< GPIO pin 0 */
    FUNC_GPIO1            = 57,     /*!< GPIO pin 1 */
    FUNC_GPIO2            = 58,     /*!< GPIO pin 2 */
    FUNC_GPIO3            = 59,     /*!< GPIO pin 3 */
    FUNC_GPIO4            = 60,     /*!< GPIO pin 4 */
    FUNC_GPIO5            = 61,     /*!< GPIO pin 5 */
    FUNC_GPIO6            = 62,     /*!< GPIO pin 6 */
    FUNC_GPIO7            = 63,     /*!< GPIO pin 7 */
    FUNC_UART1_RX         = 64,     /*!< UART1 Receiver */
    FUNC_UART1_TX         = 65,     /*!< UART1 Transmitter */
    FUNC_UART2_RX         = 66,     /*!< UART2 Receiver */
    FUNC_UART2_TX         = 67,     /*!< UART2 Transmitter */
    FUNC_UART3_RX         = 68,     /*!< UART3 Receiver */
    FUNC_UART3_TX         = 69,     /*!< UART3 Transmitter */
    FUNC_SPI1_D0          = 70,     /*!< SPI1 Data 0 */
    FUNC_SPI1_D1          = 71,     /*!< SPI1 Data 1 */
    FUNC_SPI1_D2          = 72,     /*!< SPI1 Data 2 */
    FUNC_SPI1_D3          = 73,     /*!< SPI1 Data 3 */
    FUNC_SPI1_D4          = 74,     /*!< SPI1 Data 4 */
    FUNC_SPI1_D5          = 75,     /*!< SPI1 Data 5 */
    FUNC_SPI1_D6          = 76,     /*!< SPI1 Data 6 */
    FUNC_SPI1_D7          = 77,     /*!< SPI1 Data 7 */
    FUNC_SPI1_SS0         = 78,     /*!< SPI1 Chip Select 0 */
    FUNC_SPI1_SS1         = 79,     /*!< SPI1 Chip Select 1 */
    FUNC_SPI1_SS2         = 80,     /*!< SPI1 Chip Select 2 */
    FUNC_SPI1_SS3         = 81,     /*!< SPI1 Chip Select 3 */
    FUNC_SPI1_ARB         = 82,     /*!< SPI1 Arbitration */
    FUNC_SPI1_SCLK        = 83,     /*!< SPI1 Serial Clock */
    FUNC_SPI_SLAVE_D0     = 84,     /*!< SPI Slave Data 0 */
    FUNC_SPI_SLAVE_SS     = 85,     /*!< SPI Slave Select */
    FUNC_SPI_SLAVE_SCLK   = 86,     /*!< SPI Slave Serial Clock */
    FUNC_I2S0_MCLK        = 87,     /*!< I2S0 Master Clock */
    FUNC_I2S0_SCLK        = 88,     /*!< I2S0 Serial Clock(BCLK) */
    FUNC_I2S0_WS          = 89,     /*!< I2S0 Word Select(LRCLK) */
    FUNC_I2S0_IN_D0       = 90,     /*!< I2S0 Serial Data Input 0 */
    FUNC_I2S0_IN_D1       = 91,     /*!< I2S0 Serial Data Input 1 */
    FUNC_I2S0_IN_D2       = 92,     /*!< I2S0 Serial Data Input 2 */
    FUNC_I2S0_IN_D3       = 93,     /*!< I2S0 Serial Data Input 3 */
    FUNC_I2S0_OUT_D0      = 94,     /*!< I2S0 Serial Data Output 0 */
    FUNC_I2S0_OUT_D1      = 95,     /*!< I2S0 Serial Data Output 1 */
    FUNC_I2S0_OUT_D2      = 96,     /*!< I2S0 Serial Data Output 2 */
    FUNC_I2S0_OUT_D3      = 97,     /*!< I2S0 Serial Data Output 3 */
    FUNC_I2S1_MCLK        = 98,     /*!< I2S1 Master Clock */
    FUNC_I2S1_SCLK        = 99,     /*!< I2S1 Serial Clock(BCLK) */
    FUNC_I2S1_WS          = 100,    /*!< I2S1 Word Select(LRCLK) */
    FUNC_I2S1_IN_D0       = 101,    /*!< I2S1 Serial Data Input 0 */
    FUNC_I2S1_IN_D1       = 102,    /*!< I2S1 Serial Data Input 1 */
    FUNC_I2S1_IN_D2       = 103,    /*!< I2S1 Serial Data Input 2 */
    FUNC_I2S1_IN_D3       = 104,    /*!< I2S1 Serial Data Input 3 */
```

```
    FUNC_I2S1_OUT_D0      = 105,    /*!< I2S1 Serial Data Output 0 */
    FUNC_I2S1_OUT_D1      = 106,    /*!< I2S1 Serial Data Output 1 */
    FUNC_I2S1_OUT_D2      = 107,    /*!< I2S1 Serial Data Output 2 */
    FUNC_I2S1_OUT_D3      = 108,    /*!< I2S1 Serial Data Output 3 */
    FUNC_I2S2_MCLK        = 109,    /*!< I2S2 Master Clock */
    FUNC_I2S2_SCLK        = 110,    /*!< I2S2 Serial Clock(BCLK) */
    FUNC_I2S2_WS          = 111,    /*!< I2S2 Word Select(LRCLK) */
    FUNC_I2S2_IN_D0       = 112,    /*!< I2S2 Serial Data Input 0 */
    FUNC_I2S2_IN_D1       = 113,    /*!< I2S2 Serial Data Input 1 */
    FUNC_I2S2_IN_D2       = 114,    /*!< I2S2 Serial Data Input 2 */
    FUNC_I2S2_IN_D3       = 115,    /*!< I2S2 Serial Data Input 3 */
    FUNC_I2S2_OUT_D0      = 116,    /*!< I2S2 Serial Data Output 0 */
    FUNC_I2S2_OUT_D1      = 117,    /*!< I2S2 Serial Data Output 1 */
    FUNC_I2S2_OUT_D2      = 118,    /*!< I2S2 Serial Data Output 2 */
    FUNC_I2S2_OUT_D3      = 119,    /*!< I2S2 Serial Data Output 3 */
    FUNC_RESV0            = 120,    /*!< Reserved function */
    FUNC_RESV1            = 121,    /*!< Reserved function */
    FUNC_RESV2            = 122,    /*!< Reserved function */
    FUNC_RESV3            = 123,    /*!< Reserved function */
    FUNC_RESV4            = 124,    /*!< Reserved function */
    FUNC_RESV5            = 125,    /*!< Reserved function */
    FUNC_I2C0_SCLK        = 126,    /*!< I2C0 Serial Clock */
    FUNC_I2C0_SDA         = 127,    /*!< I2C0 Serial Data */
    FUNC_I2C1_SCLK        = 128,    /*!< I2C1 Serial Clock */
    FUNC_I2C1_SDA         = 129,    /*!< I2C1 Serial Data */
    FUNC_I2C2_SCLK        = 130,    /*!< I2C2 Serial Clock */
    FUNC_I2C2_SDA         = 131,    /*!< I2C2 Serial Data */
    FUNC_CMOS_XCLK        = 132,    /*!< DVP System Clock */
    FUNC_CMOS_RST         = 133,    /*!< DVP System Reset */
    FUNC_CMOS_PWND        = 134,    /*!< DVP Power Down Mode */
    FUNC_CMOS_VSYNC       = 135,    /*!< DVP Vertical Sync */
    FUNC_CMOS_HREF        = 136,    /*!< DVP Horizontal Reference output */
    FUNC_CMOS_PCLK        = 137,    /*!< Pixel Clock */
    FUNC_CMOS_D0          = 138,    /*!< Data Bit 0 */
    FUNC_CMOS_D1          = 139,    /*!< Data Bit 1 */
    FUNC_CMOS_D2          = 140,    /*!< Data Bit 2 */
    FUNC_CMOS_D3          = 141,    /*!< Data Bit 3 */
    FUNC_CMOS_D4          = 142,    /*!< Data Bit 4 */
    FUNC_CMOS_D5          = 143,    /*!< Data Bit 5 */
    FUNC_CMOS_D6          = 144,    /*!< Data Bit 6 */
    FUNC_CMOS_D7          = 145,    /*!< Data Bit 7 */
    FUNC_SCCB_SCLK        = 146,    /*!< SCCB Serial Clock */
    FUNC_SCCB_SDA         = 147,    /*!< SCCB Serial Data */
    FUNC_UART1_CTS        = 148,    /*!< UART1 Clear To Send */
    FUNC_UART1_DSR        = 149,    /*!< UART1 Data Set Ready */
    FUNC_UART1_DCD        = 150,    /*!< UART1 Data Carrier Detect */
    FUNC_UART1_RI         = 151,    /*!< UART1 Ring Indicator */
    FUNC_UART1_SIR_IN     = 152,    /*!< UART1 Serial Infrared Input */
    FUNC_UART1_DTR        = 153,    /*!< UART1 Data Terminal Ready */
    FUNC_UART1_RTS        = 154,    /*!< UART1 Request To Send */
    FUNC_UART1_OUT2       = 155,    /*!< UART1 User-designated Output 2 */
    FUNC_UART1_OUT1       = 156,    /*!< UART1 User-designated Output 1 */
    FUNC_UART1_SIR_OUT    = 157,    /*!< UART1 Serial Infrared Output */
```

```
    FUNC_UART1_BAUD       = 158,    /*!< UART1 Transmit Clock Output */
    FUNC_UART1_RE         = 159,    /*!< UART1 Receiver Output Enable */
    FUNC_UART1_DE         = 160,    /*!< UART1 Driver Output Enable */
    FUNC_UART1_RS485_EN   = 161,    /*!< UART1 RS485 Enable */
    FUNC_UART2_CTS        = 162,    /*!< UART2 Clear To Send */
    FUNC_UART2_DSR        = 163,    /*!< UART2 Data Set Ready */
    FUNC_UART2_DCD        = 164,    /*!< UART2 Data Carrier Detect */
    FUNC_UART2_RI         = 165,    /*!< UART2 Ring Indicator */
    FUNC_UART2_SIR_IN     = 166,    /*!< UART2 Serial Infrared Input */
    FUNC_UART2_DTR        = 167,    /*!< UART2 Data Terminal Ready */
    FUNC_UART2_RTS        = 168,    /*!< UART2 Request To Send */
    FUNC_UART2_OUT2       = 169,    /*!< UART2 User-designated Output 2 */
    FUNC_UART2_OUT1       = 170,    /*!< UART2 User-designated Output 1 */
    FUNC_UART2_SIR_OUT    = 171,    /*!< UART2 Serial Infrared Output */
    FUNC_UART2_BAUD       = 172,    /*!< UART2 Transmit Clock Output */
    FUNC_UART2_RE         = 173,    /*!< UART2 Receiver Output Enable */
    FUNC_UART2_DE         = 174,    /*!< UART2 Driver Output Enable */
    FUNC_UART2_RS485_EN   = 175,    /*!< UART2 RS485 Enable */
    FUNC_UART3_CTS        = 176,    /*!< UART3 Clear To Send */
    FUNC_UART3_DSR        = 177,    /*!< UART3 Data Set Ready */
    FUNC_UART3_DCD        = 178,    /*!< UART3 Data Carrier Detect */
    FUNC_UART3_RI         = 179,    /*!< UART3 Ring Indicator */
    FUNC_UART3_SIR_IN     = 180,    /*!< UART3 Serial Infrared Input */
    FUNC_UART3_DTR        = 181,    /*!< UART3 Data Terminal Ready */
    FUNC_UART3_RTS        = 182,    /*!< UART3 Request To Send */
    FUNC_UART3_OUT2       = 183,    /*!< UART3 User-designated Output 2 */
    FUNC_UART3_OUT1       = 184,    /*!< UART3 User-designated Output 1 */
    FUNC_UART3_SIR_OUT    = 185,    /*!< UART3 Serial Infrared Output */
    FUNC_UART3_BAUD       = 186,    /*!< UART3 Transmit Clock Output */
    FUNC_UART3_RE         = 187,    /*!< UART3 Receiver Output Enable */
    FUNC_UART3_DE         = 188,    /*!< UART3 Driver Output Enable */
    FUNC_UART3_RS485_EN   = 189,    /*!< UART3 RS485 Enable */
    FUNC_TIMER0_TOGGLE1   = 190,    /*!< TIMER0 Toggle Output 1 */
    FUNC_TIMER0_TOGGLE2   = 191,    /*!< TIMER0 Toggle Output 2 */
    FUNC_TIMER0_TOGGLE3   = 192,    /*!< TIMER0 Toggle Output 3 */
    FUNC_TIMER0_TOGGLE4   = 193,    /*!< TIMER0 Toggle Output 4 */
    FUNC_TIMER1_TOGGLE1   = 194,    /*!< TIMER1 Toggle Output 1 */
    FUNC_TIMER1_TOGGLE2   = 195,    /*!< TIMER1 Toggle Output 2 */
    FUNC_TIMER1_TOGGLE3   = 196,    /*!< TIMER1 Toggle Output 3 */
    FUNC_TIMER1_TOGGLE4   = 197,    /*!< TIMER1 Toggle Output 4 */
    FUNC_TIMER2_TOGGLE1   = 198,    /*!< TIMER2 Toggle Output 1 */
    FUNC_TIMER2_TOGGLE2   = 199,    /*!< TIMER2 Toggle Output 2 */
    FUNC_TIMER2_TOGGLE3   = 200,    /*!< TIMER2 Toggle Output 3 */
    FUNC_TIMER2_TOGGLE4   = 201,    /*!< TIMER2 Toggle Output 4 */
    FUNC_CLK_SPI2         = 202,    /*!< Clock SPI2 */
    FUNC_CLK_I2C2         = 203,    /*!< Clock I2C2 */
    FUNC_INTERNAL0        = 204,    /*!< Internal function signal 0 */
    FUNC_INTERNAL1        = 205,    /*!< Internal function signal 1 */
    FUNC_INTERNAL2        = 206,    /*!< Internal function signal 2 */
    FUNC_INTERNAL3        = 207,    /*!< Internal function signal 3 */
    FUNC_INTERNAL4        = 208,    /*!< Internal function signal 4 */
    FUNC_INTERNAL5        = 209,    /*!< Internal function signal 5 */
    FUNC_INTERNAL6        = 210,    /*!< Internal function signal 6 */
```

```
    FUNC_INTERNAL7          = 211,    /*!< Internal function signal 7 */
    FUNC_INTERNAL8          = 212,    /*!< Internal function signal 8 */
    FUNC_INTERNAL9          = 213,    /*!< Internal function signal 9 */
    FUNC_INTERNAL10         = 214,    /*!< Internal function signal 10 */
    FUNC_INTERNAL11         = 215,    /*!< Internal function signal 11 */
    FUNC_INTERNAL12         = 216,    /*!< Internal function signal 12 */
    FUNC_INTERNAL13         = 217,    /*!< Internal function signal 13 */
    FUNC_INTERNAL14         = 218,    /*!< Internal function signal 14 */
    FUNC_INTERNAL15         = 219,    /*!< Internal function signal 15 */
    FUNC_INTERNAL16         = 220,    /*!< Internal function signal 16 */
    FUNC_INTERNAL17         = 221,    /*!< Internal function signal 17 */
    FUNC_CONSTANT           = 222,    /*!< Constant function */
    FUNC_INTERNAL18         = 223,    /*!< Internal function signal 18 */
    FUNC_DEBUG0             = 224,    /*!< Debug function 0 */
    FUNC_DEBUG1             = 225,    /*!< Debug function 1 */
    FUNC_DEBUG2             = 226,    /*!< Debug function 2 */
    FUNC_DEBUG3             = 227,    /*!< Debug function 3 */
    FUNC_DEBUG4             = 228,    /*!< Debug function 4 */
    FUNC_DEBUG5             = 229,    /*!< Debug function 5 */
    FUNC_DEBUG6             = 230,    /*!< Debug function 6 */
    FUNC_DEBUG7             = 231,    /*!< Debug function 7 */
    FUNC_DEBUG8             = 232,    /*!< Debug function 8 */
    FUNC_DEBUG9             = 233,    /*!< Debug function 9 */
    FUNC_DEBUG10            = 234,    /*!< Debug function 10 */
    FUNC_DEBUG11            = 235,    /*!< Debug function 11 */
    FUNC_DEBUG12            = 236,    /*!< Debug function 12 */
    FUNC_DEBUG13            = 237,    /*!< Debug function 13 */
    FUNC_DEBUG14            = 238,    /*!< Debug function 14 */
    FUNC_DEBUG15            = 239,    /*!< Debug function 15 */
    FUNC_DEBUG16            = 240,    /*!< Debug function 16 */
    FUNC_DEBUG17            = 241,    /*!< Debug function 17 */
    FUNC_DEBUG18            = 242,    /*!< Debug function 18 */
    FUNC_DEBUG19            = 243,    /*!< Debug function 19 */
    FUNC_DEBUG20            = 244,    /*!< Debug function 20 */
    FUNC_DEBUG21            = 245,    /*!< Debug function 21 */
    FUNC_DEBUG22            = 246,    /*!< Debug function 22 */
    FUNC_DEBUG23            = 247,    /*!< Debug function 23 */
    FUNC_DEBUG24            = 248,    /*!< Debug function 24 */
    FUNC_DEBUG25            = 249,    /*!< Debug function 25 */
    FUNC_DEBUG26            = 250,    /*!< Debug function 26 */
    FUNC_DEBUG27            = 251,    /*!< Debug function 27 */
    FUNC_DEBUG28            = 252,    /*!< Debug function 28 */
    FUNC_DEBUG29            = 253,    /*!< Debug function 29 */
    FUNC_DEBUG30            = 254,    /*!< Debug function 30 */
    FUNC_DEBUG31            = 255,    /*!< Debug function 31 */
    FUNC_MAX                = 256,    /*!< Function numbers */
} fpioa_function_t;
```

6.4.5.3 Enumeration element

Element name	Description
FUNC_JTAG_TCLK	JTAG Test Clock
FUNC_JTAG_TDI	JTAG Test Data In
FUNC_JTAG_TMS	JTAG Test Mode Select
FUNC_JTAG_TDO	JTAG Test Data Out
FUNC_SPI0_D0	SPI0 Data 0
FUNC_SPI0_D1	SPI0 Data 1
FUNC_SPI0_D2	SPI0 Data 2
FUNC_SPI0_D3	SPI0 Data 3
FUNC_SPI0_D4	SPI0 Data 4
FUNC_SPI0_D5	SPI0 Data 5
FUNC_SPI0_D6	SPI0 Data 6
FUNC_SPI0_D7	SPI0 Data 7
FUNC_SPI0_SS0	SPI0 Chip Select 0
FUNC_SPI0_SS1	SPI0 Chip Select 1
FUNC_SPI0_SS2	SPI0 Chip Select 2
FUNC_SPI0_SS3	SPI0 Chip Select 3
FUNC_SPI0_ARB	SPI0 Arbitration
FUNC_SPI0_SCLK	SPI0 Serial Clock
FUNC_UARTHS_RX	UART High speed Receiver
FUNC_UARTHS_TX	UART High speed Transmitter
FUNC_RESV6	Clock Input 1
FUNC_RESV7	Clock Input 2
FUNC_CLK_SPI1	Clock SPI1
FUNC_CLK_I2C1	Clock I2C1
FUNC_GPIOHS0	GPIO High speed 0
FUNC_GPIOHS1	GPIO High speed 1
FUNC_GPIOHS2	GPIO High speed 2
FUNC_GPIOHS3	GPIO High speed 3
FUNC_GPIOHS4	GPIO High speed 4
FUNC_GPIOHS5	GPIO High speed 5
FUNC_GPIOHS6	GPIO High speed 6
FUNC_GPIOHS7	GPIO High speed 7
FUNC_GPIOHS8	GPIO High speed 8
FUNC_GPIOHS9	GPIO High speed 9
FUNC_GPIOHS10	GPIO High speed 10

Element name	Description
FUNC_GPIOHS11	GPIO High speed 11
FUNC_GPIOHS12	GPIO High speed 12
FUNC_GPIOHS13	GPIO High speed 13
FUNC_GPIOHS14	GPIO High speed 14
FUNC_GPIOHS15	GPIO High speed 15
FUNC_GPIOHS16	GPIO High speed 16
FUNC_GPIOHS17	GPIO High speed 17
FUNC_GPIOHS18	GPIO High speed 18
FUNC_GPIOHS19	GPIO High speed 19
FUNC_GPIOHS20	GPIO High speed 20
FUNC_GPIOHS21	GPIO High speed 21
FUNC_GPIOHS22	GPIO High speed 22
FUNC_GPIOHS23	GPIO High speed 23
FUNC_GPIOHS24	GPIO High speed 24
FUNC_GPIOHS25	GPIO High speed 25
FUNC_GPIOHS26	GPIO High speed 26
FUNC_GPIOHS27	GPIO High speed 27
FUNC_GPIOHS28	GPIO High speed 28
FUNC_GPIOHS29	GPIO High speed 29
FUNC_GPIOHS30	GPIO High speed 30
FUNC_GPIOHS31	GPIO High speed 31
FUNC_GPIO0	GPIO pin 0
FUNC_GPIO1	GPIO pin 1
FUNC_GPIO2	GPIO pin 2
FUNC_GPIO3	GPIO pin 3
FUNC_GPIO4	GPIO pin 4
FUNC_GPIO5	GPIO pin 5
FUNC_GPIO6	GPIO pin 6
FUNC_GPIO7	GPIO pin 7
FUNC_UART1_RX	UART1 Receiver
FUNC_UART1_TX	UART1 Transmitter
FUNC_UART2_RX	UART2 Receiver
FUNC_UART2_TX	UART2 Transmitter
FUNC_UART3_RX	UART3 Receiver
FUNC_UART3_TX	UART3 Transmitter

Element name	Description
FUNC_SPI1_D0	SPI1 Data 0
FUNC_SPI1_D1	SPI1 Data 1
FUNC_SPI1_D2	SPI1 Data 2
FUNC_SPI1_D3	SPI1 Data 3
FUNC_SPI1_D4	SPI1 Data 4
FUNC_SPI1_D5	SPI1 Data 5
FUNC_SPI1_D6	SPI1 Data 6
FUNC_SPI1_D7	SPI1 Data 7
FUNC_SPI1_SS0	SPI1 Chip Select 0
FUNC_SPI1_SS1	SPI1 Chip Select 1
FUNC_SPI1_SS2	SPI1 Chip Select 2
FUNC_SPI1_SS3	SPI1 Chip Select 3
FUNC_SPI1_ARB	SPI1 Arbitration
FUNC_SPI1_SCLK	SPI1 Serial Clock
FUNC_SPI_SLAVE_D0	SPI Slave Data 0
FUNC_SPI_SLAVE_SS	SPI Slave Select
FUNC_SPI_SLAVE_SCLK	SPI Slave Serial Clock
FUNC_I2S0_MCLK	I2S0 Master Clock
FUNC_I2S0_SCLK	I2S0 Serial Clock(BCLK)
FUNC_I2S0_WS	I2S0 Word Select(LRCLK)
FUNC_I2S0_IN_D0	I2S0 Serial Data Input 0
FUNC_I2S0_IN_D1	I2S0 Serial Data Input 1
FUNC_I2S0_IN_D2	I2S0 Serial Data Input 2
FUNC_I2S0_IN_D3	I2S0 Serial Data Input 3
FUNC_I2S0_OUT_D0	I2S0 Serial Data Output 0
FUNC_I2S0_OUT_D1	I2S0 Serial Data Output 1
FUNC_I2S0_OUT_D2	I2S0 Serial Data Output 2
FUNC_I2S0_OUT_D3	I2S0 Serial Data Output 3
FUNC_I2S1_MCLK	I2S1 Master Clock
FUNC_I2S1_SCLK	I2S1 Serial Clock(BCLK)
FUNC_I2S1_WS	I2S1 Word Select(LRCLK)
FUNC_I2S1_IN_D0	I2S1 Serial Data Input 0
FUNC_I2S1_IN_D1	I2S1 Serial Data Input 1
FUNC_I2S1_IN_D2	I2S1 Serial Data Input 2
FUNC_I2S1_IN_D3	I2S1 Serial Data Input 3

Element name	Description
FUNC_I2S1_OUT_D0	I2S1 Serial Data Output 0
FUNC_I2S1_OUT_D1	I2S1 Serial Data Output 1
FUNC_I2S1_OUT_D2	I2S1 Serial Data Output 2
FUNC_I2S1_OUT_D3	I2S1 Serial Data Output 3
FUNC_I2S2_MCLK	I2S2 Master Clock
FUNC_I2S2_SCLK	I2S2 Serial Clock(BCLK)
FUNC_I2S2_WS	I2S2 Word Select(LRCLK)
FUNC_I2S2_IN_D0	I2S2 Serial Data Input 0
FUNC_I2S2_IN_D1	I2S2 Serial Data Input 1
FUNC_I2S2_IN_D2	I2S2 Serial Data Input 2
FUNC_I2S2_IN_D3	I2S2 Serial Data Input 3
FUNC_I2S2_OUT_D0	I2S2 Serial Data Output 0
FUNC_I2S2_OUT_D1	I2S2 Serial Data Output 1
FUNC_I2S2_OUT_D2	I2S2 Serial Data Output 2
FUNC_I2S2_OUT_D3	I2S2 Serial Data Output 3
FUNC_RESV0	Reserved function
FUNC_RESV1	Reserved function
FUNC_RESV2	Reserved function
FUNC_RESV3	Reserved function
FUNC_RESV4	Reserved function
FUNC_RESV5	Reserved function
FUNC_I2C0_SCLK	I2C0 Serial Clock
FUNC_I2C0_SDA	I2C0 Serial Data
FUNC_I2C1_SCLK	I2C1 Serial Clock
FUNC_I2C1_SDA	I2C1 Serial Data
FUNC_I2C2_SCLK	I2C2 Serial Clock
FUNC_I2C2_SDA	I2C2 Serial Data
FUNC_CMOS_XCLK	DVP System Clock
FUNC_CMOS_RST	DVP System Reset
FUNC_CMOS_PWDN	DVP Power Down Mode
FUNC_CMOS_VSYNC	DVP Vertical Sync
FUNC_CMOS_HREF	DVP Horizontal Reference output
FUNC_CMOS_PCLK	Pixel Clock
FUNC_CMOS_D0	Data Bit 0
FUNC_CMOS_D1	Data Bit 1

Element name	Description
FUNC_CMOS_D2	Data Bit 2
FUNC_CMOS_D3	Data Bit 3
FUNC_CMOS_D4	Data Bit 4
FUNC_CMOS_D5	Data Bit 5
FUNC_CMOS_D6	Data Bit 6
FUNC_CMOS_D7	Data Bit 7
FUNC_SCCB_SCLK	SCCB Serial Clock
FUNC_SCCB_SDA	SCCB Serial Data
FUNC_UART1_CTS	UART1 Clear To Send
FUNC_UART1_DSR	UART1 Data Set Ready
FUNC_UART1_DCD	UART1 Data Carrier Detect
FUNC_UART1_RI	UART1 Ring Indicator
FUNC_UART1_SIR_IN	UART1 Serial Infrared Input
FUNC_UART1_DTR	UART1 Data Terminal Ready
FUNC_UART1_RTS	UART1 Request To Send
FUNC_UART1_OUT2	UART1 User-designated Output 2
FUNC_UART1_OUT1	UART1 User-designated Output 1
FUNC_UART1_SIR_OUT	UART1 Serial Infrared Output
FUNC_UART1_BAUD	UART1 Transmit Clock Output
FUNC_UART1_RE	UART1 Receiver Output Enable
FUNC_UART1_DE	UART1 Driver Output Enable
FUNC_UART1_RS485_EN	UART1 RS485 Enable
FUNC_UART2_CTS	UART2 Clear To Send
FUNC_UART2_DSR	UART2 Data Set Ready
FUNC_UART2_DCD	UART2 Data Carrier Detect
FUNC_UART2_RI	UART2 Ring Indicator
FUNC_UART2_SIR_IN	UART2 Serial Infrared Input
FUNC_UART2_DTR	UART2 Data Terminal Ready
FUNC_UART2_RTS	UART2 Request To Send
FUNC_UART2_OUT2	UART2 User-designated Output 2
FUNC_UART2_OUT1	UART2 User-designated Output 1
FUNC_UART2_SIR_OUT	UART2 Serial Infrared Output
FUNC_UART2_BAUD	UART2 Transmit Clock Output
FUNC_UART2_RE	UART2 Receiver Output Enable
FUNC_UART2_DE	UART2 Driver Output Enable

Element name	Description
FUNC_UART2_RS485_EN	UART2 RS485 Enable
FUNC_UART3_CTS	UART3 Clear To Send
FUNC_UART3_DSR	UART3 Data Set Ready
FUNC_UART3_DCD	UART3 Data Carrier Detect
FUNC_UART3_RI	UART3 Ring Indicator
FUNC_UART3_SIR_IN	UART3 Serial Infrared Input
FUNC_UART3_DTR	UART3 Data Terminal Ready
FUNC_UART3_RTS	UART3 Request To Send
FUNC_UART3_OUT2	UART3 User-designated Output 2
FUNC_UART3_OUT1	UART3 User-designated Output 1
FUNC_UART3_SIR_OUT	UART3 Serial Infrared Output
FUNC_UART3_BAUD	UART3 Transmit Clock Output
FUNC_UART3_RE	UART3 Receiver Output Enable
FUNC_UART3_DE	UART3 Driver Output Enable
FUNC_UART3_RS485_EN	UART3 RS485 Enable
FUNC_TIMER0_TOGGLE1	TIMER0 Toggle Output 1
FUNC_TIMER0_TOGGLE2	TIMER0 Toggle Output 2
FUNC_TIMER0_TOGGLE3	TIMER0 Toggle Output 3
FUNC_TIMER0_TOGGLE4	TIMER0 Toggle Output 4
FUNC_TIMER1_TOGGLE1	TIMER1 Toggle Output 1
FUNC_TIMER1_TOGGLE2	TIMER1 Toggle Output 2
FUNC_TIMER1_TOGGLE3	TIMER1 Toggle Output 3
FUNC_TIMER1_TOGGLE4	TIMER1 Toggle Output 4
FUNC_TIMER2_TOGGLE1	TIMER2 Toggle Output 1
FUNC_TIMER2_TOGGLE2	TIMER2 Toggle Output 2
FUNC_TIMER2_TOGGLE3	TIMER2 Toggle Output 3
FUNC_TIMER2_TOGGLE4	TIMER2 Toggle Output 4
FUNC_CLK_SPI2	Clock SPI2
FUNC_CLK_I2C2	Clock I2C2
FUNC_INTERNAL0	Internal function 0
FUNC_INTERNAL1	Internal function 1
FUNC_INTERNAL2	Internal function 2
FUNC_INTERNAL3	Internal function 3
FUNC_INTERNAL4	Internal function 4
FUNC_INTERNAL5	Internal function 5

Element name	Description
FUNC_INTERNAL6	Internal function 6
FUNC_INTERNAL7	Internal function 7
FUNC_INTERNAL8	Internal function 8
FUNC_INTERNAL9	Internal function 9
FUNC_INTERNAL10	Internal function 10
FUNC_INTERNAL11	Internal function 11
FUNC_INTERNAL12	Internal function 12
FUNC_INTERNAL13	Internal function 13
FUNC_INTERNAL14	Internal function 14
FUNC_INTERNAL15	Internal function 15
FUNC_INTERNAL16	Internal function 16
FUNC_INTERNAL17	Internal function 17
FUNC_CONSTANT	Constant function
FUNC_INTERNAL18	Internal function 18
FUNC_DEBUG0	Debug function 0
FUNC_DEBUG1	Debug function 1
FUNC_DEBUG2	Debug function 2
FUNC_DEBUG3	Debug function 3
FUNC_DEBUG4	Debug function 4
FUNC_DEBUG5	Debug function 5
FUNC_DEBUG6	Debug function 6
FUNC_DEBUG7	Debug function 7
FUNC_DEBUG8	Debug function 8
FUNC_DEBUG9	Debug function 9
FUNC_DEBUG10	Debug function 10
FUNC_DEBUG11	Debug function 11
FUNC_DEBUG12	Debug function 12
FUNC_DEBUG13	Debug function 13
FUNC_DEBUG14	Debug function 14
FUNC_DEBUG15	Debug function 15
FUNC_DEBUG16	Debug function 16
FUNC_DEBUG17	Debug function 17
FUNC_DEBUG18	Debug function 18
FUNC_DEBUG19	Debug function 19
FUNC_DEBUG20	Debug function 20

Element name	Description
FUNC_DEBUG21	Debug function 21
FUNC_DEBUG22	Debug function 22
FUNC_DEBUG23	Debug function 23
FUNC_DEBUG24	Debug function 24
FUNC_DEBUG25	Debug function 25
FUNC_DEBUG26	Debug function 26
FUNC_DEBUG27	Debug function 27
FUNC_DEBUG28	Debug function 28
FUNC_DEBUG29	Debug function 29
FUNC_DEBUG30	Debug function 30
FUNC_DEBUG31	Debug function 31

Chapter 7

DVP

7.1 Overview

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

7.2 Features

The DVP unit has the following features:

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

7.3 API

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

- dvp_init
- dvp_set_output_enable
- dvp_set_image_format
- dvp_set_image_size

- dvp_set_ai_addr
- dvp_set_display_addr
- dvp_config_interrupt
- dvp_get_interrupt
- dvp_clear_interrupt
- dvp_start_convert
- dvp_enable_burst
- dvp_disable_burst
- dvp_enable_auto
- dvp_disable_auto
- dvp_sccb_send_data
- dvp_sccb_receive_data

7.3.1 dvp_init

7.3.1.1 Description

Initialize DVP.

7.3.1.2 Function prototype

```
void dvp_init(uint8_t reg_len)
```

7.3.1.3 Parameter

Parameter name	Description	Input or output
reg_len	SCCB register length	Input

7.3.1.4 Return value

None.

7.3.2 dvp_set_output_enable

7.3.2.1 Description

Set the output mode to enable or disable.

7.3.2.2 Function prototype

```
void dvp_set_output_enable(dvp_output_mode_t index, int enable)
```

7.3.2.3 Parameter

Parameter name	Description	Input or output
index	Image output to memory or KPU	Input
enable	0: Disable 1: Enable	Input

7.3.2.4 Return value

None.

7.3.3 dvp_set_image_format

7.3.3.1 Description

Set the image receiving mode, RGB or YUV.

7.3.3.2 Function prototype

```
void dvp_set_image_format(uint32_t format)
```

7.3.3.3 Parameter

Parameter name	Description	Input or output
format	Image format selection*1	Input

7.3.3.4 Return value

None.

*1 DVP_CFG_RGB_FORMAT is RGB format, DVP_CFG_YUV_FORMAT is YUV format.

7.3.4 dvp_set_image_size

7.3.4.1 Description

Set the DVP image capture size.

7.3.4.2 Function prototype

```
void dvp_set_image_size(uint32_t width, uint32_t height)
```

7.3.4.3 Parameter

Parameter name	Description	Input or output
width	Image width	Input
height	Image height	Input

7.3.4.4 Return value

None.

7.3.5 dvp_set_ai_addr

7.3.5.1 Description

Set the image address required by the KPU, in order to facilitate the KPU to perform algorithm processing.

7.3.5.2 Function prototype

```
void dvp_set_ai_addr(uint32_t r_addr, uint32_t g_addr, uint32_t b_addr)
```

7.3.5.3 Parameter

Parameter name	Description	Input or output
r_addr	Red component address	Input
g_addr	Green component address	Input
b_addr	Blue component address	Input

7.3.5.4 Return value

None.

7.3.6 dvp_set_display_addr

7.3.6.1 Description

Set the storage address of the captured image in the memory, which can be used for display.

7.3.6.2 Function prototype

```
void dvp_set_display_addr(uint32_t addr)
```

7.3.6.3 Parameter

Parameter name	Description	Input or output
addr	Memory address to store images	Input

7.3.6.4 Return value

None.

7.3.7 dvp_config_interrupt

Configure the DVP interrupt type.

7.3.7.1 Function prototype

```
void dvp_config_interrupt(uint32_t interrupt, uint8_t enable)
```

7.3.7.2 Description

Set image capture start and end interrupt status, enable or disable.

7.3.7.3 Parameter

Parameter name	Description	Input or output

Parameter name	Description	Input or output
interrupt	DVP interrupt type*2	Input
enable	0: Disable 1: Enable	Input

7.3.7.4 Return value

None.

7.3.8 dvp_get_interrupt

7.3.8.1 Description

Determine if it is the specified interrupt type.

7.3.8.2 Function prototype

```
int dvp_get_interrupt(uint32_t interrupt)
```

7.3.8.3 Parameter

Parameter name	Description	Input or output
interrupt	DVP interrupt type*3	Input

7.3.8.4 Return value

Return value	Description
0	False
Others	True

*2 DVP_CFG_START_INT_ENABLE for the interrupt of image capture begin; DVP_CFG_FINISH_INT_ENABLE for the interrupt of image capture end.

*3 DVP_CFG_START_INT_ENABLE for the interrupt of image capture begin; DVP_CFG_FINISH_INT_ENABLE for the interrupt of image capture end.

7.3.9 dvp_clear_interrupt

7.3.9.1 Description

Clear the interrupt.

7.3.9.2 Function prototype

```
void dvp_clear_interrupt(uint32_t interrupt)
```

7.3.9.3 Parameter

Parameter name	Description	Input or output
interrupt	DVP interrupt type[*4]	Input

7.3.9.4 Return value

None.

7.3.10 dvp_start_convert

7.3.10.1 Description

Start capturing images and call them after determining that the image capture begins.

7.3.10.2 Function prototype

```
void dvp_start_convert(void)
```

7.3.10.3 Parameter

None.

7.3.10.4 Return value

None.

[*4] DVP_CFG_START_INT_ENABLE for the interrupt of image capture begin; DVP_CFG_FINISH_INT_ENABLE for the interrupt of image capture end.

7.3.11 dvp_enable_burst

7.3.11.1 Description

Enable burst transfer mode.

7.3.11.2 Function prototype

```
void dvp_enable_burst(void)
```

7.3.11.3 Parameter

None.

7.3.11.4 Return value

None.

7.3.12 dvp_disable_burst

7.3.12.1 Description

Disable burst transfer mode.

7.3.12.2 Function prototype

```
void dvp_disable_burst(void)
```

7.3.12.3 Parameter

None.

7.3.12.4 Return value

None.

7.3.13 dvp_enable_auto

7.3.13.1 Description

Enable automatic image mode reception.s

7.3.13.2 Function prototype

```
void dvp_enable_auto(void)
```

7.3.13.3 Parameter

None.

7.3.13.4 Return value

None.

7.3.14 dvp_disable_auto

7.3.14.1 Description

Disable automatic image reception mode.

7.3.14.2 Function prototype

```
void dvp_disable_auto(void)
```

7.3.14.3 Parameter

None.

7.3.14.4 Return value

None.

7.3.15 dvp_sccb_send_data

7.3.15.1 Description

Send data via scbb.

7.3.15.2 Function prototype

```
void dvp_sccb_send_data(uint8_t dev_addr, uint16_t reg_addr, uint8_t reg_data)
```

7.3.15.3 Parameter

Parameter name	Description	Input or output
dev_addr	Image sensor SCCB address	Input
reg_addr	Image sensor register	Input
reg_data	Sent data	Input

7.3.15.4 Return value

None.

7.3.16 dvp_sccb_receive_data

7.3.16.1 Description

Receive data through the SCCB.

7.3.16.2 Function prototype

```
uint8_t dvp_sccb_receive_data(uint8_t dev_addr, uint16_t reg_addr)
```

7.3.16.3 Parameter

Parameter name	Description	Input or output
dev_addr	Image sensor SCCB address	Input
reg_addr	Image sensor register	Input

7.3.16.4 Return value

Read the data of the register.

7.3.17 Example

```
/*
 * Capture 320 * 240 RGB image data transfer to lcd_gram0,
 * and 0x40600000 0x40612C00 0x40625800 address
 */
uint32_t lcd_gram0[38400] __attribute__((aligned(64)));

int on_irq_dvp(void* ctx)
{
    if (dvp_get_interrupt(DVP_STS_FRAME_FINISH))
    {
```

```
        dvp_clear_interrupt(DVP_STS_FRAME_FINISH);
    }
    else
    {
        dvp_start_convert();
        dvp_clear_interrupt(DVP_STS_FRAME_START);
    }
    return 0;
}
plic_init();
dvp_init(8);
dvp_enable_burst();
dvp_set_output_enable(DVP_OUTPUT_AI, 1);
dvp_set_output_enable(DVP_OUTPUT_DISPLAY, 1);
dvp_set_image_format(DVP_CFG_RGB_FORMAT);
dvp_set_image_size(320, 240);
dvp_set_ai_addr((uint32_t)0x40600000, (uint32_t)0x40612C00, (uint32_t)0x40625800);
dvp_set_display_addr(lcd_gram0);
dvp_config_interrupt(DVP_CFG_START_INT_ENABLE | DVP_CFG_FINISH_INT_ENABLE, 0);
dvp_disable_auto();
plic_set_priority(IRQN_DVP_INTERRUPT, 1);
plic_irq_register(IRQN_DVP_INTERRUPT, on_irq_dvp, NULL);
plic_irq_enable(IRQN_DVP_INTERRUPT);
| dvp_clear_interrupt(DVP_STS_FRAME_START       | DVP_STS_FRAME_FINISH);         |
| dvp_config_interrupt(DVP_CFG_START_INT_ENABLE | DVP_CFG_FINISH_INT_ENABLE, 1); |
sysctl_enable_irq();
/*
 * Send 0x01 to the peripheral 0xFF register of address 0x60 through SCCB,
 * read data from register 0x1D.
 */
dvp_sccb_send_data(0x60, 0xFF, 0x01);
dvp_sccb_receive_data(0x60, 0x1D)
```

7.4 Data type

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

- dvp_output_mode_t: DVP output image mode.

7.4.1 dvp_output_mode_t

7.4.1.1 Description

DVP output image mode.

7.4.1.2 Type definition

```
typedef enum _dvp_output_mode
{
    DVP_OUTPUT_AI,
    DVP_OUTPUT_DISPLAY,
} dvp_output_mode_t;
```

7.4.1.3 Enumeration element

Element name	Description
DVP_OUTPUT_AI	Output to KPU
DVP_OUTPUT_DISPLAY	Output to memory for display

Chapter 8

FFT

8.1 Overview

Fast Fourier Transform (FFT) Accelerator.

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

8.2 Features

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

8.3 API

Corresponding header file `fft.h`

Provide the following interfaces

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

- fft_complex_uint16_dma

8.3.1 fft_complex_uint16_dma

8.3.1.1 Description

FFT operation.

8.3.1.2 Function prototype

```
void fft_complex_uint16_dma(dmac_channel_number_t dma_send_channel_num,
    dmac_channel_number_t dma_receive_channel_num, uint16_t shift, fft_direction_t
    direction, const uint64_t *input, size_t point_num, uint64_t *output);
```

8.3.1.3 Parameter

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

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

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

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

8.3.1.4 Return value

None.

8.3.2 Example

```
#define FFT_N                512U
#define FFT_FORWARD_SHIFT    0x0U
#define FFT_BACKWARD_SHIFT   0x1ffU
#define PI                   3.14159265358979323846
complex_hard_t data_hard[FFT_N] = {0};
for (i = 0; i < FFT_N; i++)
{
    tempf1[0] = 0.3 * cosf(2 * PI * i / FFT_N + PI / 3) * 256;
    tempf1[1] = 0.1 * cosf(16 * 2 * PI * i / FFT_N - PI / 9) * 256;
    tempf1[2] = 0.5 * cosf((19 * 2 * PI * i / FFT_N) + PI / 6) * 256;
    data_hard[i].real = (int16_t)(tempf1[0] + tempf1[1] + tempf1[2] + 10);
    data_hard[i].imag = (int16_t)0;
}
for (int i = 0; i < FFT_N / 2; ++i)
{
    input_data = (fft_data_t *)&buffer_input[i];
    input_data->R1 = data_hard[2 * i].real;
    input_data->I1 = data_hard[2 * i].imag;
    input_data->R2 = data_hard[2 * i + 1].real;
    input_data->I2 = data_hard[2 * i + 1].imag;
}
fft_complex_uint16_dma(DMAC_CHANNEL0, DMAC_CHANNEL1, FFT_FORWARD_SHIFT, FFT_DIR_FORWARD
    , buffer_input, FFT_N, buffer_output);
for (i = 0; i < FFT_N / 2; i++)
{
    output_data = (fft_data_t*)&buffer_output[i];
    data_hard[2 * i].imag = output_data->I1 ;
    data_hard[2 * i].real = output_data->R1 ;
    data_hard[2 * i + 1].imag = output_data->I2 ;
    data_hard[2 * i + 1].real = output_data->R2 ;
}
for (int i = 0; i < FFT_N / 2; ++i)
{
    input_data = (fft_data_t *)&buffer_input[i];
    input_data->R1 = data_hard[2 * i].real;
    input_data->I1 = data_hard[2 * i].imag;
    input_data->R2 = data_hard[2 * i + 1].real;
    input_data->I2 = data_hard[2 * i + 1].imag;
}
fft_complex_uint16_dma(DMAC_CHANNEL0, DMAC_CHANNEL1, 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 ;
}
```

8.4 Data type

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

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

8.4.1 fft_data_t

8.4.1.1 Description

The data format passed in by the FFT operation.

8.4.1.2 Type definition

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

8.4.1.3 Enumeration element

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

8.4.2 fft_direction_t

8.4.2.1 Description

FFT transform mode.

8.4.2.2 Type definition

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

8.4.2.3 Enumeration element

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

Chapter 9

SHA256

9.1 Overview

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

9.2 Features

- Supports hardware acceleration of the sha256 algorithm

9.3 API

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

- sha256_init
- sha256_update
- sha256_final
- sha256_hard_calculate

9.3.1 sha256_init

9.3.1.1 Description

Initialize the SHA256 accelerator.

9.3.1.2 Function prototype

```
void sha256_init(sha256_context_t *context, size_t input_len)
```

9.3.1.3 Parameter

Parameter name	Description	Input or output
context	SHA256 context	Input
input_len	Length of data to be SHA256 hash	Input

9.3.1.4 Return value

None.

9.3.2 Example

```
sha256_context_t context;
sha256_init(&context, 128U);
```

9.3.3 sha256_update

9.3.3.1 Description

Pass in a data block for SHA256 Hash

9.3.3.2 Function prototype

```
void sha256_update(sha256_context_t *context, const void *input, size_t input_len)
```

9.3.3.3 Parameter

Parameter name	Description	Input or output
context	SHA256 context	Input
input	Input data block to be hashed	Input
buf_len	Length of input data	Input

9.3.3.4 Return value

None.

9.3.4 sha256_final

9.3.4.1 Description

End the SHA256 Hash calculation and get the result

9.3.4.2 Function prototype

```
void sha256_final(sha256_context_t *context, uint8_t *output)
```

9.3.4.3 Parameter

Parameter name	Description	Input or output
context	SHA256 context	Input
output	SHA256 hash result. The size of buffer must be greater than or equal to 32 bytes	Output

9.3.4.4 Return value

None.

9.3.5 sha256_hard_calculate

9.3.5.1 Description

Directly get the sha256 hash of the data

9.3.5.2 Function prototype

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

9.3.5.3 Parameter

Parameter name	Description	Input or output
input	Input data to be hashed	Input

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

9.3.5.4 Return value

None.

9.3.6 Example

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

- Single sha256 hash

```
sha256_context_t context;
sha256_init(&context, input_len);
sha256_update(&context, input, input_len);
sha256_final(&context, output);
```

Or you can call the `sha256_hard_calculate` function directly

```
sha256_hard_calculate(input, input_len, output);
```

- Multiple sha256 hashes

```
sha256_context_t context;
sha256_init(&context, input_piece1_len + input_piece2_len);
sha256_update(&context, input_piece1, input_piece1_len);
sha256_update(&context, input_piece2, input_piece2_len);
sha256_final(&context, output);
```

Chapter 10

UART

10.1 Overview

Universal Asynchronous Receiver/Transmitter (UART).

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

10.2 Features

The UART module has the following features:

- Configuring UART parameters
- Automatically charge data to the buffer

10.3 API

Corresponding header file `uart.h`

Provide the following interfaces

- uart_init
- uart_config (deprecated)
- uart_configure
- uart_send_data
- uart_send_data_dma

- uart_send_data_dma_irq
- uart_receive_data
- uart_receive_data_dma
- uart_receive_data_dma_irq
- uart_irq_register
- uart_irq_deregister

10.3.1 uart_init

10.3.1.1 Description

Initialize uart.

10.3.1.2 Function prototype

```
void uart_init(uart_device_number_t channel)
```

10.3.1.3 Parameter

Parameter name	Description	Input or output
channel	UART number	Input

10.3.1.4 Return value

None.

10.3.2 uart_configure

10.3.2.1 Description

Set the UART related parameters. This function is deprecated and the recommended replacement function is uart_configure.

10.3.3 uart_configure

10.3.3.1 Description

Set the UART related parameters.

10.3.3.2 Function prototype

```
void uart_configure(uart_device_number_t channel, uint32_t baud_rate, uart_bitwidth_t
    data_width, uart_stopbit_t stopbit, uart_parity_t parity)
```

10.3.3.3 Parameter

Parameter name	Description	Input or output
channel	UART number	Input
baud_rate	Baud rate	Input
data_width	Data bits (5-8)	Input
stopbit	Stop bit	Input
parity	Parity bit	Input

10.3.3.4 Return value

None.

10.3.4 uart_send_data

10.3.4.1 Description

Send data through the UART.

10.3.4.2 Function prototype

```
int uart_send_data(uart_device_number_t channel, const char *buffer, size_t buf_len)
```

10.3.4.3 Parameter

Parameter name	Description	Input or output
channel	UART number	Input
buffer	Data to be sent	Input
buf_len	Length of data to be sent	Input

10.3.4.4 Return value

The length of the data that has been sent.

10.3.5 uart_send_data_dma

10.3.5.1 Description

The UART sends data via DMA. After all the data has been sent, return.

10.3.5.2 Function prototype

```
void uart_send_data_dma(uart_device_number_t uart_channel, dmac_channel_number_t
    dmac_channel, const uint8_t *buffer, size_t buf_len)
```

10.3.5.3 Parameter

Parameter name	Description	Input or output
uart_channel	UART number	Input
dmac_channel	DMA channel	Input
buffer	Data to be sent	Input
buf_len	Length of data to be sent	Input

10.3.5.4 Return value

None.

10.3.6 uart_send_data_dma_irq

10.3.6.1 Description

The UART sends data through DMA and sets the DMA transmission completion interrupt function, which is only a single interrupt.

10.3.6.2 Function prototype

```
void uart_send_data_dma_irq(uart_device_number_t uart_channel, dmac_channel_number_t
    dmac_channel,
                            const uint8_t *buffer, size_t buf_len, plic_irq_callback_t
                                uart_callback,
                            void *ctx, uint32_t priority)
```

10.3.6.3 Parameter

Parameter name	Description	Input or output
uart_channel	UART number	Input
dmac_channel	DMA channel	Input
buffer	Data to be sent	Input
buf_len	Length of data to be sent	Input
uart_callback	DMA interrupt callback	Input
ctx	Interrupt function parameter	Input
priority	Interrupt priority	Input

10.3.6.4 Return value

None.

10.3.7 uart_receive_data

10.3.7.1 Description

Read data through the UART.

10.3.7.2 Function prototype

```
int uart_receive_data(uart_device_number_t channel, char *buffer, size_t buf_len);
```

10.3.7.3 Parameter

Parameter name	Description	Input or output
channel	UART number	Input
buffer	Receive data	Output
buf_len	Length of received data	Input

10.3.7.4 Return value

The length of the data that has been received.

10.3.8 uart_receive_data_dma

10.3.8.1 Description

The UART receives data via DMA.

10.3.8.2 Function prototype

```
void uart_receive_data_dma(uart_device_number_t uart_channel, dmac_channel_number_t
    dmac_channel, uint8_t *buffer, size_t buf_len)
```

10.3.8.3 Parameter

Parameter name	Description	Input or output
uart_channel	UART number	Input
dmac_channel	DMA channel	Input
buffer	Receive data	Output
buf_len	Length of received data	Input

10.3.8.4 Return value

None.

10.3.9 uart_receive_data_dma_irq

10.3.9.1 Description

The UART receives data via DMA and registers the DMA receive completion interrupt function with a single interrupt.

10.3.9.2 Function prototype

```
void uart_receive_data_dma_irq(uart_device_number_t uart_channel, dmac_channel_number_t
     dmac_channel,
                                    uint8_t *buffer, size_t buf_len,
                                        plic_irq_callback_t uart_callback,
                                    void *ctx, uint32_t priority)
```

10.3.9.3 Parameter

Parameter name	Description	Input or output
uart_channel	UART number	Input
dmac_channel	DMA channel	Input
buffer	Receive data	Output

Parameter name	Description	Input or output
buf_len	Length of received data	Input
uart_callback	DMA interrupt callback	Input
ctx	Interrupt function parameter	Input
priority	Interrupt priority	Input

10.3.9.4 Return value

None.

10.3.10 uart_irq_register

10.3.10.1 Description

Register the UART interrupt function.

10.3.10.2 Function prototype

```
void uart_irq_register(uart_device_number_t channel, uart_interrupt_mode_t
    interrupt_mode, plic_irq_callback_t uart_callback, void *ctx, uint32_t priority)
```

10.3.10.3 Parameter

Parameter name	Description	Input or output
channel	UART number	Input
interrupt_mode	Interrupt type	Input
uart_callback	Interrupt callback	Input
ctx	Interrupt function parameter	Input
priority	Interrupt priority	Input

10.3.10.4 Return value

None.

10.3.11 uart_irq_deregister

10.3.11.1 Description

Deregister the UART interrupt function.

10.3.11.2 Function prototype

```
void uart_irq_deregister(uart_device_number_t channel, uart_interrupt_mode_t
    interrupt_mode)
```

10.3.11.3 Parameter

Parameter name	Description	Input or output
channel	UART number	Input
interrupt_mode	Interrupt type	Input

10.3.11.4 Return value

None.

10.3.12 Example

```
/* UART1 with baud rate 115200, 8 data bits, 1 stop bit, no parity bit */
uart_init(UART_DEVICE_1);
uart_config(UART_DEVICE_1, 115200, UART_BITWIDTH_8BIT, UART_STOP_1, UART_PARITY_NONE);
char *v_hel = {"hello world!\n"};
/* Send data "hello world!" */
uart_send_data(UART_DEVICE_1, hel, strlen(v_hel));
/* Receive data */
while(uart_receive_data(UART_DEVICE_1, &recv, 1))
{
    printf("%c ", recv);
}
```

10.4 Data type

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

- uart_device_number_t: UART number.
- uart_bitwidth_t: UART data bit width.
- uart_stopbits_t: UART stop bit.
- uart_parity_t: UART parity bit.
- uart_interrupt_mode_t: UART interrupt type, send or receive.
- uart_send_trigger_t: Send interrupt or DMA trigger FIFO depth.

- uart_receive_trigger_t: Receive interrupt or DMA trigger FIFO depth.

10.4.1 uart_device_number_t

10.4.1.1 Description

UART number.

10.4.1.2 Type definition

```
typedef enum _uart_device_number
{
    UART_DEVICE_1,
    UART_DEVICE_2,
    UART_DEVICE_3,
    UART_DEVICE_MAX,
} uart_device_number_t;
```

10.4.1.3 Enumeration element

Element name	Description
UART_DEVICE_1	UART 1
UART_DEVICE_2	UART 2
UART_DEVICE_3	UART 3

10.4.2 uart_bitwidth_t

10.4.2.1 Description

UART data bit width.

10.4.2.2 Type definition

```
typedef enum _uart_bitwidth
{
    UART_BITWIDTH_5BIT = 0,
    UART_BITWIDTH_6BIT,
    UART_BITWIDTH_7BIT,
    UART_BITWIDTH_8BIT,
} uart_bitwidth_t;
```

10.4.2.3 Enumeration element

Element name	Description
UART*BITWIDTH*5BIT	5 bits
UART*BITWIDTH*6BIT	6 bits
UART*BITWIDTH*7BIT	7 bits
UART*BITWIDTH*8BIT	8 bits

10.4.3 uart_stopbits_t

10.4.3.1 Description

UART stop bit.

10.4.3.2 Type definition

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

10.4.3.3 Enumeration element

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

10.4.4 uart_parity_t

10.4.4.1 Description

UART parity bit.

10.4.4.2 Type definition

```
typedef enum _uart_parity
{
    UART_PARITY_NONE,
    UART_PARITY_ODD,
    UART_PARITY_EVEN
```

```
} uart_parity_t;
```

10.4.4.3 Enumeration element

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

10.4.5 uart_interrupt_mode_t

10.4.5.1 Description

UART interrupt type, send or receive.

10.4.5.2 Type definition

```
typedef enum _uart_interrupt_mode
{
    UART_SEND = 1,
    UART_RECEIVE = 2,
} uart_interrupt_mode_t;
```

10.4.5.3 Enumeration element

Element name	Description
UART_SEND	UART send
UART_RECEIVE	UART receive

10.4.6 uart_send_trigger_t

10.4.6.1 Description

Transmit interrupt or DMA trigger FIFO depth. An interrupt or DMA transfer is triggered when the data in the FIFO is less than or equal to this value. The depth of the FIFO is 16 bytes.

10.4.6.2 Type definition

```
typedef enum _uart_send_trigger
{
    UART_SEND_FIFO_0,
    UART_SEND_FIFO_2,
    UART_SEND_FIFO_4,
    UART_SEND_FIFO_8,
} uart_send_trigger_t;
```

10.4.6.3 Enumeration element

Element name	Description
UART_SEND_FIFO_0	FIFO is empty
UART_SEND_FIFO_2	FIFO remaining 2 Bytes
UART_SEND_FIFO_4	FIFO remaining 4 Bytes
UART_SEND_FIFO_8	FIFO remaining 8 Bytes

10.4.7 uart_receive_trigger_t

10.4.7.1 Description

Receive interrupt or DMA trigger FIFO depth. An interrupt or DMA transfer is triggered when the data in the FIFO is greater than or equal to this value. The depth of the FIFO is 16 bytes.

10.4.7.2 Type definition

```
typedef enum _uart_receive_trigger
{
    UART_RECEIVE_FIFO_1,
    UART_RECEIVE_FIFO_4,
    UART_RECEIVE_FIFO_8,
    UART_RECEIVE_FIFO_14,
} uart_receive_trigger_t;
```

10.4.7.3 Enumeration element

Element name	Description
UART_RECEIVE_FIFO_1	FIFO remaining 1 Bytes
UART_RECEIVE_FIFO_4	FIFO remaining 2 Bytes
UART_RECEIVE_FIFO_8	FIFO remaining 4 Bytes

Element name	Description
UART_RECEIVE_FIFO_14	FIFO remaining 8 Bytes

Chapter 11

UARTHS

11.1 Overview

High Speed Universal Asynchronous Receiver/Transmitter (UART).

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

At present, the system uses this high speed serial port as the debugging serial port, and the serial port output is called when using printf.

11.2 Features

The UART peripheral has the following features:

- Configuring UART parameters
- Automatically charge data to the buffer
- 8-N-1 and 8-N-2 formats: 8 data bits, no parity bit, 1 start bit, 1 or 2 stop bits
- 8-entry transmit and receive FIFO buffers with programmable watermark interrupts
- 16× Rx oversampling with 2/3 majority voting per bit The UART peripheral does not support hardware flow control or other modem control signals, or synchronous serial data tranfesrs.

11.3 API

Corresponding header file uarths.h

Provide the following interfaces

- uarths_init
- uarths_config
- uarths_receive_data
- uarths_send_data
- uarths_set_irq
- uarths_get_interrupt_mode
- uarths_set_interrupt_cnt

11.3.1 uarths_init

11.3.1.1 Description

Initialize UARTHS, the system default baud rate is 115200 and 8-N-1 formats (8 data bits, no parity bit, 1 start bit, 1 stop bits). Because the clock source of the uarths is PLL0, you need to call this function to set the baud rate after setting PLL0, otherwise it will print garbled characters.

11.3.1.2 Function prototype

```
void uarths_init(void)
```

11.3.1.3 Parameter

None.

11.3.1.4 Return value

None.

11.3.2 uarths_config

11.3.2.1 Description

Set the parameters of the UARTHS. Default is 8-N-1 formats (8 data bits, no parity bit, 1 start bit, 1 stop bits).

11.3.2.2 Function prototype

```
void uarths_config(uint32_t baud_rate, uarths_stopbit_t stopbit)
```

11.3.3 Parameter

Parameter name	Description	Input or output
baud_rate	Baud rate	Input
stopbit	Stop bit	Input

11.3.3.1 Return value

None.

11.3.4 uarths_receive_data

11.3.4.1 Description

Read data through UARTHS.

11.3.4.2 Function prototype

```
size_t uarths_receive_data(uint8_t *buf, size_t buf_len)
```

11.3.4.3 Parameter

Parameter name	Description	Input or output
buf	Receive data	Output
buf_len	Length of received data	Input

11.3.4.4 Return value

The length of the data that has been received.

11.3.5 uarths_send_data

11.3.5.1 Description

Send data through the UART.

11.3.5.2 Function prototype

```
size_t uarths_send_data(const uint8_t *buf, size_t buf_len)
```

11.3.5.3 Parameter

Parameter name	Description	Input or output
buf	Data to be sent	Input
buf_len	Length of data to be sent	Input

11.3.5.4 Return value

The length of the data that has been sent.

11.3.6 uarths_set_irq

11.3.6.1 Description

Set the UARTHS interrupt callback function.

11.3.6.2 Function prototype

```
void uarths_set_irq(uarths_interrupt_mode_t interrupt_mode, plic_irq_callback_t
    uarths_callback, void *ctx, uint32_t priority)
```

11.3.6.3 Parameter

Parameter name	Description	Input or output
interrupt_mode	Interrupt type	Input
uarths_callback	Interrupt callback function	Input
ctx	Callback function parameters	Input
priority	Interrupt priority	Input

11.3.6.4 Return value

None.

11.3.7 uarths_get_interrupt_mode

11.3.7.1 Description

Get the interrupt type of UARTHS. The same interrupt is used by received, sent, or received.

11.3.7.2 Function prototype

```
uarths_interrupt_mode_t uarths_get_interrupt_mode(void)
```

11.3.7.3 Parameter

None.

11.3.7.4 Return value

The type of current interrupt.

11.3.8 uarths_set_interrupt_cnt

11.3.8.1 Description

Set the FIFO depth when the UARTHS interrupt is generated. When the interrupt type is UARTHS_SEND_RECEIVE, the transmit and receive FIFO interrupt depths are both cnt.

11.3.8.2 Function prototype

```
void uarths_set_interrupt_cnt(uarths_interrupt_mode_t interrupt_mode, uint8_t cnt)
```

11.3.8.3 Parameter

Parameter name	Description	Input or output
interrupt_mode	Interrupt type	Input
cnt	FIFO depth	Input

11.3.8.4 Return value

None.

11.3.9 Example

```
/*
 * Set the receive interrupt. The interrupt FIFO depth is 0, that is, the
 * received data is immediately interrupted and the received data is read.
 */
    int uarths_irq(void *ctx)
    {
        if(!uarths_receive_data((uint8_t *)&receive_char, 1))
            printf("Uarths receive ERR!\n");
        return 0;
    }

    plic_init();
    uarths_set_interrupt_cnt(UARTHS_RECEIVE , 0);
    uarths_set_irq(UARTHS_RECEIVE ,uarths_irq, NULL, 4);
    sysctl_enable_irq();
```

11.4 Data type

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

- uarths_interrupt_mode_t: UARTHS interrupt type.
- uarths_stopbit_t: UARTHS stop bit.

11.4.1 uarths_interrupt_mode_t

11.4.2 Description

UARTHS interrupt type.

11.4.2.1 Type definition

```
typedef enum _uarths_interrupt_mode
{
    UARTHS_SEND = 1,
    UARTHS_RECEIVE = 2,
    UARTHS_SEND_RECEIVE = 3,
} uarths_interrupt_mode_t;
```

11.4.2.2 Enumeration element

Element name	Description
UARTHS_SEND	Send interrupt
UARTHS_RECEIVE	Receive interrupt
UARTHS_SEND_RECEIVE	Send or Receive interrupt

11.4.3 uarths_stopbit_t

11.4.3.1 Description

UARTHS stop bit.

11.4.3.2 Type definition

```
typedef enum _uarths_stopbit
{
    UART_STOP_1,
    UART_STOP_2
} uarths_stopbit_t;
```

11.4.3.3 Enumeration element

Element name	Description
UART_STOP_1	1 Stop bit
UART_STOP_2	2 Stop bits

Chapter 12

WDT

12.1 Overview

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

12.2 Features

The WDT module has the following features:

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

12.3 API

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

- wdt_start
- wdt_stop
- wdt_feed
- wdt_clear_interrupt

12.3.1 wdt_start

12.3.1.1 Description

Start the watchdog.

12.3.1.2 Function prototype

```
void wdt_start(wdt_device_number_t id, uint64_t time_out_ms, plic_irq_callback_t on_irq
    )
```

12.3.1.3 Parameter

Parameter name	Description	Input or output
id	Watchdog number	Input
time_out_ms	Timeout (ms)	Input
on_irq	Interrupt callback function	Input

12.3.1.4 Return value

None.

12.3.2 wdt_stop

12.3.2.1 Description

Stop the watchdog.

12.3.2.2 Function prototype

```
void wdt_stop(wdt_device_number_t id)
```

12.3.2.3 Parameter

Parameter name	Description	Input or output
id	Watchdog number	Input

12.3.2.4 Return value

None.

12.3.3 wdt_feed

12.3.3.1 Description

Feed the watchdog (clear the counter of watchdog).

12.3.3.2 Function prototype

```
void wdt_feed(wdt_device_number_t id)
```

12.3.3.3 Parameter

Parameter name	Description	Input or output
id	Watchdog number	Input

12.3.3.4 Return value

None.

12.3.4 wdt_clear_interrupt

12.3.4.1 Description

Clear the interrupt. If the interrupt is cleared in the interrupt function, the watchdog will not restart.

12.3.4.2 Function prototype

```
void wdt_clear_interrupt(wdt_device_number_t id)
```

12.3.4.3 Parameter

Parameter name	Description	Input or output
id	Watchdog number	Input

12.3.4.4 Return value

None.

12.3.5 Example

```
/*
 * After 2 seconds, enter the watchdog interrupt function to print Hello_world,
 * and then reset after 2 seconds.
 */
int wdt0_irq(void)
{
    printf("Hello_world\n");
    return 0;
}
plic_init();
sysctl_enable_irq();
wdt_start(WDT_DEVICE_0, 2000, wdt0_irq);
```

12.4 Data type

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

- wdt_device_number_t

12.4.1 wdt_device_number_t

12.4.1.1 Description

Watchdog number。

12.4.1.2 Type definition

```
typedef enum _wdt_device_number
{
    WDT_DEVICE_0,
    WDT_DEVICE_1,
    WDT_DEVICE_MAX,
} wdt_device_number_t;
```

12.4.1.3 Enumeration element

Element name	Description
WDT_DEVICE_0	Watchdog 0
WDT_DEVICE_1	Watchdog 1

Chapter 13

DMA

13.1 Overview

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

13.2 Features

The DMA controller has the following features:

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

13.3 API

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

- dmac_init

- dmac_set_single_mode
- dmac_is_done
- dmac_wait_done
- dmac_set_irq
- dmac_set_src_dest_length
- dmac_is_idle
- dmac_wait_idle

13.3.1 dmac_init

13.3.1.1 Description

Initialize the DMA.

13.3.1.2 Function prototype

```
void dmac_init(void)
```

13.3.1.3 Parameter

None.

13.3.1.4 Return value

None.

13.3.2 dmac_set_single_mode

13.3.2.1 Description

Set a single channel DMA parameter.

13.3.2.2 Function prototype

```
void dmac_set_single_mode(dmac_channel_number_t channel_num, const void *src, void *
    dest, dmac_address_increment_t src_inc, dmac_address_increment_t dest_inc,
    dmac_burst_trans_length_t dmac_burst_size, dmac_transfer_width_t dmac_trans_width,
     size_t block_size)
```

13.3.2.3 Parameter

Parameter name	Description	Input or output
channel_num	DMA channel number	Input
src	Source address	Input
dest	Destination address	Output
src_inc	Whether the source address is increasing	Input
dest_inc	Whether the destination address is increasing	Input
dmac_burst_size	Burst transfer size	Input
dmac_trans_width	Single transfer data bit width	Input
block_size	The size of the transmitted data	Input

13.3.2.4 Return value

None.

13.3.3 dmac_is_done

13.3.3.1 Description

It is used to determine whether the transmission is completed after the DMAC starts the transmission. It can only be used after the DMAC starts transmitting. If you use it before the DMAC starts transmitting, you will get an incorrect result.

13.3.3.2 Function prototype

```
int dmac_is_done(dmac_channel_number_t channel_num)
```

13.3.3.3 Parameter

Parameter name	Description	Input or output
channel_num	DMA channel number	Input

13.3.3.4 Return value

Return value	Description
0	Undone
1	Done

13.3.4 dmac_wait_done

13.3.4.1 Description

Wait for the DMA to finish working.

13.3.4.2 Function prototype

```
void dmac_wait_done(dmac_channel_number_t channel_num)
```

13.3.4.3 Parameter

Parameter name	Description	Input or output
channel_num	DMA channel number	Input

13.3.4.4 Return value

None.

13.3.5 dmac_set_irq

13.3.5.1 Description

Set the callback function of the DMAC interrupt.

13.3.5.2 Function prototype

```
void dmac_set_irq(dmac_channel_number_t channel_num , plic_irq_callback_t dmac_callback
    , void *ctx, uint32_t priority)
```

13.3.5.3 Parameter

Parameter name	Description	Input or output
channel_num	DMA channel number	Input
dmac_callback	Interrupt callback function	Input
ctx	Callback function parameters	Input
priority	Interrupt priority	Input

13.3.5.4 Return value

None.

13.3.6 dmac_set_src_dest_length

13.3.6.1 Description

Set the source address, destination address, and length of the DMAC, and then start DMA transfer. If src is NULL, the source address is not set. If dest is NULL, the destination address is not set. If len<=0, the length is not set.

This function is typically used in DMAC interrupts to allow the DMA to continue transferring data without having to set all the parameters of the DMAC again to save time.

13.3.6.2 Function prototype

```
void dmac_set_src_dest_length(dmac_channel_number_t channel_num, const void *src, void
    *dest, size_t len)
```

13.3.6.3 Parameter

Parameter name	Description	Input or output
channel_num	DMA channel number	Input
src	Interrupt callback function	Input
dest	Callback function parameters	Input
len	Transmission length	Input

13.3.6.4 Return value

None.

13.3.7 dmac_is_idle

13.3.7.1 Description

Determine whether the current channel of the DMAC is idle. This function can be used to determine the DMAC status before and after transmission.

13.3.7.2 Function prototype

```
int dmac_is_idle(dmac_channel_number_t channel_num)
```

13.3.7.3 Parameter

Parameter name	Description	Input or output
channel_num	DMA channel number	Input

13.3.7.4 Return value

Return value	Description
0	Busy
1	Idle

13.3.8 dmac_wait_idle

13.3.8.1 Description

Wait for the DMAC to enter the idle state.

13.3.8.2 Parameter

Parameter name	Description	Input or output
channel_num	DMA channel number	Input

13.3.8.3 Return value

None.

13.3.9 Example

```
/* I2C sends 128 int data via DMA */
uint32_t buf[128];
dmac_wait_idle(SYSCTL_DMA_CHANNEL_0);
sysctl_dma_select(SYSCTL_DMA_CHANNEL_0, SYSCTL_DMA_SELECT_I2C0_TX_REQ);
dmac_set_single_mode(SYSCTL_DMA_CHANNEL_0, buf, (void*)(&i2c_adapter->data_cmd),
    DMAC_ADDR_INCREMENT, DMAC_ADDR_NOCHANGE, DMAC_MSIZE_4, DMAC_TRANS_WIDTH_32, 128);
dmac_wait_done(SYSCTL_DMA_CHANNEL_0);
```

13.4 Data type

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

- dmac_channel_number_t: DMA channel number.
- dmac_address_increment_t: Address self-increasing behavior.
- dmac_burst_trans_length_t: Burst transfer size.
- dmac_transfer_width_t: The bit width of a single transfer of data.

13.4.1 dmac_channel_number_t

13.4.1.1 Description

DMA channel number.

13.4.1.2 Type definition

```
typedef enum _dmac_channel_number
{
    DMAC_CHANNEL0 = 0,
    DMAC_CHANNEL1 = 1,
    DMAC_CHANNEL2 = 2,
    DMAC_CHANNEL3 = 3,
    DMAC_CHANNEL4 = 4,
    DMAC_CHANNEL5 = 5,
    DMAC_CHANNEL_MAX
} dmac_channel_number_t;
```

13.4.1.3 Enumeration element

Element name	Description
DMAC_CHANNEL0	DMA channel 0
DMAC_CHANNEL1	DMA channel 1
DMAC_CHANNEL2	DMA channel 2
DMAC_CHANNEL3	DMA channel 3
DMAC_CHANNEL4	DMA channel 4
DMAC_CHANNEL5	DMA channel 5

13.4.2 dmac_address_increment_t

13.4.2.1 Description

Address self-increasing behavior.

13.4.2.2 Type definition

```
typedef enum _dmac_address_increment
{
    DMAC_ADDR_INCREMENT = 0x0,
    DMAC_ADDR_NOCHANGE  = 0x1
} dmac_address_increment_t;
```

13.4.2.3 Enumeration element

Element name	Description
DMAC_ADDR_INCREMENT	Automatic address increase
DMAC_ADDR_NOCHANGE	Fixed address

13.4.3 dmac_burst_trans_length_t

13.4.3.1 Description

Burst transfer size.

13.4.3.2 Type definition

```
typedef enum _dmac_burst_trans_length
{
    DMAC_MSIZE_1   = 0x0,
    DMAC_MSIZE_4   = 0x1,
    DMAC_MSIZE_8   = 0x2,
    DMAC_MSIZE_16  = 0x3,
    DMAC_MSIZE_32  = 0x4,
    DMAC_MSIZE_64  = 0x5,
    DMAC_MSIZE_128 = 0x6,
    DMAC_MSIZE_256 = 0x7
} dmac_burst_trans_length_t;
```

13.4.3.3 Enumeration element

Element name	Description
DMAC_MSIZE_1	Single transmission number multiplied by 1
DMAC_MSIZE_4	Single transmission number multiplied by 4
DMAC_MSIZE_8	Single transmission number multiplied by 8
DMAC_MSIZE_16	Single transmission number multiplied by 16
DMAC_MSIZE_32	Single transmission number multiplied by 32
DMAC_MSIZE_64	Single transmission number multiplied by 64
DMAC_MSIZE_128	Single transmission number multiplied by 128
DMAC_MSIZE_256	Single transmission number multiplied by 256

13.4.4 dmac_transfer_width_t

13.4.4.1 Description

The bit width of a single transfer of data.

13.4.4.2 Type definition

```
typedef enum _dmac_transfer_width
{
    DMAC_TRANS_WIDTH_8   = 0x0,
    DMAC_TRANS_WIDTH_16  = 0x1,
    DMAC_TRANS_WIDTH_32  = 0x2,
    DMAC_TRANS_WIDTH_64  = 0x3,
    DMAC_TRANS_WIDTH_128 = 0x4,
    DMAC_TRANS_WIDTH_256 = 0x5
} dmac_transfer_width_t;
```

13.4.4.3 Enumeration element

Element name	Description
DMAC_TRANS_WIDTH_8	Single transmission of 8 bits
DMAC_TRANS_WIDTH_16	Single transmission of 16 bits
DMAC_TRANS_WIDTH_32	Single transmission of 32 bits
DMAC_TRANS_WIDTH_64	Single transmission of 64 bits
DMAC_TRANS_WIDTH_128	Single transmission of 128 bits
DMAC_TRANS_WIDTH_256	Single transmission of 256 bits

Chapter 14

I²C

14.1 Overview

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

14.2 Features

The I²C unit has the following features:

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

14.3 API

Corresponding header file `i2c.h`
Provide the following interfaces

- i2c_init
- i2c_init_as_slave
- i2c_send_data
- i2c_send_data_dma
- i2c_recv_data
- i2c_recv_data_dma

14.3.1 i2c_init

14.3.1.1 Description

Configure the I²C device slave address, register bit width, and I²C rate.

14.3.1.2 Function prototype

```
void i2c_init(i2c_device_number_t i2c_num, uint32_t slave_address, uint32_t
    address_width, uint32_t i2c_clk)
```

14.3.1.3 Parameter

Parameter name	Description	Input or output
i2c_num	I²C number	Input
slave_address	I²C device slave address	Input
address_width	I²C device register width (7 or 10)	Input
i2c_clk	I²C clock rate (Hz)	Input

14.3.1.4 Return value

None.

14.3.2 i2c_init_as_slave

14.3.2.1 Description

Configure the I²C controller to be in slave mode.

14.3.2.2 Function prototype

```
void i2c_init_as_slave(i2c_device_number_t i2c_num, uint32_t slave_address, uint32_t
    address_width, const i2c_slave_handler_t *handler)
```

14.3.2.3 Parameter

Parameter name	Description	Input or output
i2c_num	I²C number	Input
slave_address	I²C slave address	Input

Parameter name	Description	Input or output
address_width	I²C device register width (7 or 10)	Input
handler	I²C slave interrupt handler	Input

14.3.2.4 Return value

None.

14.3.3 i2c_send_data

14.3.3.1 Description

Write data.

14.3.3.2 Function prototype

```
int i2c_send_data(i2c_device_number_t i2c_num, const uint8_t *send_buf, size_t
    send_buf_len)
```

14.3.3.3 Parameter

Parameter name	Description	Input or output
i2c_num	I²C number	Input
send_buf	Data to be transmitted	Input
send_buf_len	Length of data to be transmitted	Input

14.3.3.4 Return value

Return value	Description
0	Success
Others	Fail

14.3.4 i2c_send_data_dma

14.3.4.1 Description

Write data through the DMA.

14.3.4.2 Function prototype

```
void i2c_send_data_dma(dmac_channel_number_t dma_channel_num, i2c_device_number_t
    i2c_num, const uint8_t *send_buf, size_t send_buf_len)
```

14.3.4.3 Parameter

Parameter name	Description	Input or output
dma_channel_num	DMA channel number used	Input
i2c_num	I²C number	Input
send_buf	Data to be transmitted	Input
send_buf_len	Length of data to be transmitted	Input

14.3.4.4 Return value

None.

14.3.5 i2c_recv_data

14.3.5.1 Description

Read data through the CPU.

14.3.5.2 Function prototype

```
int i2c_recv_data(i2c_device_number_t i2c_num, const uint8_t *send_buf, size_t
    send_buf_len, uint8_t *receive_buf, size_t receive_buf_len)
```

14.3.5.3 Parameter

Parameter name	Description	Input or output
i2c_num	I²C number	Input
send_buf	Data to be transmitted[*1]	Input
send_buf_len	Length of data to be transmitted. If not use, write 0	Input
receive_buf	Receive data buffer	Output
receive_buf_len	Length of received data	Input

[*1] The general case corresponds to the register of the I²C peripheral, if it is not set to NULL.

14.3.5.4 Return value

Return value	Description
0	Success
Others	Fail

14.3.6 i2c_recv_data_dma

14.3.6.1 Description

Read data through the DMA.

14.3.6.2 Function prototype

```
void i2c_recv_data_dma(dmac_channel_number_t dma_send_channel_num,
    dmac_channel_number_t dma_receive_channel_num,
    i2c_device_number_t i2c_num, const uint8_t *send_buf, size_t send_buf_len, uint8_t
        *receive_buf, size_t receive_buf_len)
```

14.3.6.3 Parameter

Parameter name	Description	Input or output
dma_send_channel_num	Dma channel used to send data	Input
dma_receive_channel_num	Dma channel for receiving data	Input
i2c_num	I²C number	Input
send_buf	Data to be transmitted[*2]	Input
send_buf_len	Length of data to be transmitted. If not use, write 0	Input
receive_buf	Receive data buffer	Output
receive_buf_len	Length of received data	Input

14.3.6.4 Return value

None.

14.3.7 Example

[*2] The general case corresponds to the register of the I²C peripheral, if it is not set to NULL.

```
/* The i2c peripheral address is 0x32, 7-bit address, rate 200K */
i2c_init(I2C_DEVICE_0, 0x32, 7, 200000);
uint8_t reg = 0;
uint8_t data_buf[2] = {0x00,0x01}
data_buf[0] = reg;
/* Write 0x01 to the 0 register */
i2c_send_data(I2C_DEVICE_0, data_buf, 2);
i2c_send_data_dma(DMAC_CHANNEL0, I2C_DEVICE_0, data_buf, 4);
/* Read 1 byte data from 0 registe */
i2c_receive_data(I2C_DEVICE_0, &reg, 1, data_buf, 1);
i2c_receive_data_dma(DMAC_CHANNEL0, DMAC_CHANNEL1, I2C_DEVICE_0,&reg, 1, data_buf, 1);
```

14.4 Data type

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

- i2c_device_number_t: I²C device number.
- i2c_slave_handler_t: I²C slave mode interrupt handler function handle.

14.4.1 i2c_device_number_t

14.4.1.1 Description

I²C device number.

14.4.1.2 Type definition

```
typedef enum _i2c_device_number
{
    I2C_DEVICE_0,
    I2C_DEVICE_1,
    I2C_DEVICE_2,
    I2C_DEVICE_MAX,
} i2c_device_number_t;
```

14.4.2 i2c_slave_handler_t

14.4.2.1 Description

I²C slave mode interrupt handler function handle. The corresponding function operations are performed according to different interrupt states.

14.4.2.2 Type definition

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

14.4.2.3 Enumeration element

Element name	Description
I2C_DEVICE_0	I²C 0
I2C_DEVICE_1	I²C 1
I2C_DEVICE_2	I²C 2

Chapter 15

SPI

15.1 Overview

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

15.2 Features

The SPI module has the following features:

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

15.3 API

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

- spi_init
- spi_init_non_standard
- spi_send_data_standard
- spi_send_data_standard_dma

- spi_receive_data_standard
- spi_receive_data_standard_dma
- spi_send_data_multiple
- spi_send_data_multiple_dma
- spi_receive_data_multiple
- spi_receive_data_multiple_dma
- spi_fill_data_dma
- spi_send_data_normal_dma
- spi_set_clk_rate

15.3.1 spi_init

15.3.1.1 Description

Set the SPI mode of operation, multi-line mode, and transfer bit width.

15.3.1.2 Function prototype

```
void spi_init(spi_device_num_t spi_num, spi_work_mode_t work_mode, spi_frame_format_t
    frame_format, size_t data_bit_length, uint32_t endian)
```

15.3.1.3 Parameter

Parameter name	Description	Input or output
spi_num	SPI number	Input
work_mode	Four working modes	Input
frame_format	Frame format	Input
data_bit_length	Data bit length	Input
endian	Endian. 0: Little endian 1: Big endian	Input

15.3.1.4 Return value

None.

15.3.2 spi_config_non_standard

15.3.2.1 Description

Used to set the instruction length, address length, wait clock count, and instructio/ address transfer mode in multi-wire mode.

15.3.2.2 Function prototype

```
void spi_init_non_standard(spi_device_num_t spi_num, uint32_t instruction_length,
    uint32_t address_length, uint32_t wait_cycles,
    spi_instruction_address_trans_mode_t instruction_address_trans_mode)
```

15.3.2.3 Parameter

Parameter name	Description	Input or output
spi_num	SPI number	Input
instruction_length	Instruction length	Input
address_length	Address length	Input
wait_cycles	Number of waiting cycles	Input
instruction_address_trans_mode	Instruction/address transfer mode	Input

15.3.2.4 Return value

None.

15.3.3 spi_send_data_standard

15.3.3.1 Description

The SPI transfers data in standard mode.

15.3.3.2 Function prototype

```
void spi_send_data_standard(spi_device_num_t spi_num, spi_chip_select_t chip_select,
    const uint8_t *cmd_buff, size_t cmd_len, const uint8_t *tx_buff, size_t tx_len)
```

15.3.3.3 Parameter

Parameter name	Description	Input or output
spi_num	SPI number	Input
chip_select	Chip select	Input
cmd_buff	Peripheral instruction/address data, set to NULL if not used	Input

Parameter name	Description	Input or output
cmd_len	Peripheral instruction/address data length, set to 0 if not used	Input
tx_buff	Sent data buffer	Input
tx_len	Length of data sent	Input

15.3.3.4 Return value

None.

15.3.4 spi_send_data_standard_dma

15.3.4.1 Description

Data is transferred via DMA using SPI standard mode.

15.3.4.2 Function prototype

```
void spi_send_data_standard_dma(dmac_channel_number_t channel_num, spi_device_num_t
    spi_num, spi_chip_select_t chip_select, const uint8_t *cmd_buff, size_t cmd_len,
    const uint8_t *tx_buff, size_t tx_len)
```

15.3.4.3 Parameter

Parameter name	Description	Input or output
channel_num	DMA channel number	Input
spi_num	SPI number	Input
chip_select	Chip select	Input
cmd_buff	Peripheral instruction/address data, set to NULL if not used	Input
cmd_len	Peripheral instruction/address data length, set to 0 if not used	Input
tx_buff	Sent data buffer	Input
tx_len	Length of data sent	Input

15.3.4.4 Return value

None.

15.3.5 spi_receive_data_standard

15.3.5.1 Description

Receive data using standard mode.

15.3.5.2 Function prototype

```
void spi_receive_data_standard(spi_device_num_t spi_num, spi_chip_select_t chip_select,
     const uint8_t *cmd_buff, size_t cmd_len, uint8_t *rx_buff, size_t rx_len)
```

15.3.5.3 Parameter

Parameter name	Description	Input or output
spi_num	SPI number	Input
chip_select	Chip select	Input
cmd_buff	Peripheral instruction/address data, set to NULL if not used	Input
cmd_len	Peripheral instruction/address data length, set to 0 if not used	Input
rx_buff	Received data buffer	Output
rx_len	Length of received data	Input

15.3.5.4 Return value

None.

15.3.6 spi_receive_data_standard_dma

15.3.6.1 Description

Data is received via DMA using standard mode.

15.3.6.2 Function prototype

```
void spi_receive_data_standard_dma(dmac_channel_number_t dma_send_channel_num,
    dmac_channel_number_t dma_receive_channel_num, spi_device_num_t spi_num,
    spi_chip_select_t chip_select, const uint8_t *cmd_buff, size_t cmd_len, uint8_t *
    rx_buff, size_t rx_len)
```

15.3.6.3 Parameter

Parameter name	Description	Input or output
dma_send_channel_num	The DMA channel number used to send the instructio/ address	Input
dma_receive_channel_num	DMA channel number used to receive data	Input
spi_num	SPI number	Input
chip_select	Chip select	Input
cmd_buff	Peripheral instruction/address data, set to NULL if not used	Input
cmd_len	Peripheral instruction/address data length, set to 0 if not used	Input
rx_buff	Received data buffer	Output
rx_len	Length of received data	Input

15.3.6.4 Return value

None.

15.3.7 spi_send_data_multiple

15.3.7.1 Description

Send data using multi-wire mode.

15.3.7.2 Function prototype

```
void spi_send_data_multiple(spi_device_num_t spi_num, spi_chip_select_t chip_select,
    const uint32_t *cmd_buff, size_t cmd_len, uint8_t *tx_buff, size_t tx_len)
```

15.3.7.3 Parameter

Parameter name	Description	Input or output
spi_num	SPI number	Input
chip_select	Chip select	Input

Parameter name	Description	Input or output
cmd_buff	Peripheral instruction/address data, set to NULL if not used	Input
cmd_len	Peripheral instruction/address data length, set to 0 if not used	Input
tx_buff	Sent data buffer	Input
tx_len	Length of data sent	Input

15.3.7.4 Return value

None.

15.3.8 spi_send_data_multiple_dma

15.3.8.1 Description

Data is sent by DMA using multi-line mode.

15.3.8.2 Function prototype

```
void spi_send_data_multiple_dma(dmac_channel_number_t channel_num,spi_device_num_t
    spi_num, spi_chip_select_t chip_select, const uint32_t *cmd_buff, size_t cmd_len,
    const uint8_t *tx_buff, size_t tx_len)
```

15.3.8.3 Parameter

Parameter name	Description	Input or output
channel_num	DMA channel number	Input
spi_num	SPI number	Input
chip_select	Chip select	Input
cmd_buff	Peripheral instruction/address data, set to NULL if not used	Input
cmd_len	Peripheral instruction/address data length, set to 0 if not used	Input
tx_buff	Sent data buffer	Input
tx_len	Length of data sent	Input

15.3.8.4 Return value

None.

15.3.9 spi_receive_data_multiple

15.3.9.1 Description

Receive data using multi-wire mode.

15.3.9.2 Function prototype

```
void spi_receive_data_multiple(spi_device_num_t spi_num, spi_chip_select_t chip_select,
     const uint32_t *cmd_buff, size_t cmd_len, uint8_t *rx_buff, size_t rx_len)
```

15.3.9.3 Parameter

Parameter name	Description	Input or output
spi_num	SPI number	Input
chip_select	Chip select	Input
cmd_buff	Peripheral instruction/address data, set to NULL if not used	Input
cmd_len	Peripheral instruction/address data length, set to 0 if not used	Input
rx_buff	Received data buffer	Output
rx_len	Length of received data	Input

15.3.9.4 Return value

None.

15.3.10 spi_receive_data_multiple_dma

15.3.10.1 Description

Receive data via DMA using multi-wire mode.

15.3.10.2 Function prototype

```
void spi_receive_data_multiple_dma(dmac_channel_number_t dma_send_channel_num,
    dmac_channel_number_t dma_receive_channel_num, spi_device_num_t spi_num,
```

```
    spi_chip_select_t chip_select, uint32_t const *cmd_buff, size_t cmd_len, uint8_t *
    rx_buff, size_t rx_len);
```

15.3.10.3 Parameter

Parameter name	Description	Input or output
dma_send_channel_num	The DMA channel number used to send the instructio/ address	Input
dma_receive_channel_num	DMA channel number used to receive data	Input
spi_num	SPI number	Input
chip_select	Chip select	Input
cmd_buff	Peripheral instruction/address data, set to NULL if not used	Input
cmd_len	Peripheral instruction/address data length, set to 0 if not used	Input
rx_buff	Received data buffer	Output
rx_len	Length of received data	Input

15.3.10.4 Return value

None.

15.3.11 spi_fill_data_dma

15.3.11.1 Description

Always send the same data through DMA, can be used to refresh data.

15.3.11.2 Function prototype

```
void spi_fill_data_dma(dmac_channel_number_t channel_num,spi_device_num_t spi_num,
    spi_chip_select_t chip_select, const uint32_t *tx_buff, size_t tx_len);
```

15.3.11.3 Parameter

Parameter name	Description	Input or output
channel_num	DMA channel number	Input
spi_num	SPI number	Input
chip_select	Chip select	Input
tx_buff	Sent data buffer, send only tx_buff this data, it will not increase automatically	Input
tx_len	Length of data sent	Input

15.3.11.4 Return value

None.

15.3.12 spi_send_data_normal_dma

15.3.12.1 Description

Send data by DMA. There is no need to set the instruction/address.

15.3.12.2 Function prototype

```
void spi_send_data_normal_dma(dmac_channel_number_t channel_num, spi_device_num_t
    spi_num, spi_chip_select_t chip_select, const void *tx_buff, size_t tx_len,
    spi_transfer_width_t spi_transfer_width)
```

15.3.12.3 Parameter

Parameter name	Description	Input or output
channel_num	DMA channel number	Input
spi_num	SPI number	Input
chip_select	Chip select	Input
tx_buff	Sent data buffer, send only tx_buff this data, it will not increase automatically	Input
tx_len	Length of data sent	Input
spi_transfer_width	The bit width of the transmitted data	Input

15.3.12.4 Return value

None.

15.3.13 Example

```
/* SPI0 operates in MODE0 (standard SPI mode) for single transmission of 8-bit data */
spi_init(SPI_DEVICE_0, SPI_WORK_MODE_0, SPI_FF_STANDARD, 8, 0);
uint8_t cmd[4];
cmd[0] = 0x06;
cmd[1] = 0x01;
cmd[2] = 0x02;
cmd[3] = 0x04;
uint8_t data_buf[4] = {0,1,2,3};
/* SPI0 sends the instruction 0x06 using chip select 0, and sends 0, 1, 2, 3 four bytes
     of data to address 0x010204. */
spi_send_data_standard(SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 4, data_buf, 4);
/* SPI0 sends a 0x06 instruction using chip select 0 and then receives 4 bytes of data
    from Address 0x010204. */
spi_receive_data_standard(SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 4, data_buf, 4);

/* SPI0 operates in MODE0 (4-wire SPI mode) single-transmission of 8-bit data */
spi_init(SPI_DEVICE_0, SPI_WORK_MODE_0, SPI_FF_QUAD, 8, 0);
/* 8-bit instruction length 32-bit address length wait for 4 clk after sending the
    instruction/address, the instruction is sent by standard SPI mode, and the address
     is sent by four-wire mode. */
spi_init_non_standard(SPI_DEVICE_0, 8, 32, 4, SPI_AITM_ADDR_STANDARD);
uint32 cmd[2];
cmd[0] = 0x06;
cmd[1] = 0x010204;
uint8_t data_buf[4] = {0,1,2,3};
/* SPI0 sends the instruction 0x06 using chip select 0, and sends 0, 1, 2, 3 four bytes
     of data to address 0x010204. */
spi_send_data_multiple(SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 2, data_buf, 4);
/* SPI0 sends a 0x06 instruction using chip select 0 and then receives 4 bytes of data
    from Address 0x010204. */
spi_receive_data_multiple(SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 2, data_buf, 4);

/* SPI0 works in MODE2 (8-wire SPI mode) single-transmission 32-bit data */
spi_init(SPI_DEVICE_0, SPI_WORK_MODE_2, SPI_FF_OCTAL, 32, 0);
/* No instruction 32-bit address length Wait for 0 clk after sending the instruction/
    address, the instruction/address is sent through 8-wires */
spi_init_non_standard(SPI_DEVICE_0, 0, 32, 0, SPI_AITM_AS_FRAME_FORMAT);
uint32_t data_buf[256] = {0};
/* Send 256 int data using DMA channel 0 and chip select 0 */
spi_send_data_normal_dma(DMAC_CHANNEL0, SPI_DEVICE_0, SPI_CHIP_SELECT_0, data_buf, 256,
     SPI_TRANS_INT);
uint32_t data = 0x55AA55AA;
/* Use DMA channel 0 and chip select 0 to send 256 consecutively 0x55AA55AA */
spi_fill_data_dma(DMAC_CHANNEL0, SPI_DEVICE_0, SPI_CHIP_SELECT_0,&data, 256);
```

15.3.14 spi_set_clk_rate

15.3.14.1 Description

Set the clock frequency of the SPI

15.3.14.2 Function prototype

```
uint32_t spi_set_clk_rate(spi_device_num_t spi_num, uint32_t spi_clk)
```

15.3.14.3 Parameter

Parameter name	Description	Input or output
spi_num	SPI number	Input
spi_clk	Target SPI device clock frequency	Input

15.3.14.4 Return value

The actual clock frequency of the SPI device after setup

15.4 Data type

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

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

15.4.1 spi_device_num_t

15.4.1.1 Description

SPI number.

15.4.1.2 Type definition

```
typedef enum _spi_device_num
```

```
{
    SPI_DEVICE_0,
    SPI_DEVICE_1,
    SPI_DEVICE_2,
    SPI_DEVICE_3,
    SPI_DEVICE_MAX,
} spi_device_num_t;
```

15.4.1.3 Enumeration element

Element name	Description
SPI*DEVICE*0	SPI 0 as the master device
SPI*DEVICE*1	SPI 1 as the master device
SPI*DEVICE*2	SPI 2 as the slave device
SPI*DEVICE*3	SPI 3 as the master device

15.4.2 spi_mode_t

15.4.2.1 Description

SPI mode.

15.4.2.2 Type definition

```
typedef enum _spi_mode
{
    SPI_WORK_MODE_0,
    SPI_WORK_MODE_1,
    SPI_WORK_MODE_2,
    SPI_WORK_MODE_3,
} spi_mode_t;
```

15.4.2.3 Enumeration element

Element name	Description
SPI_WORK_MODE_0	SPI mode 0
SPI_WORK_MODE_1	SPI mode 1
SPI_WORK_MODE_2	SPI mode 2
SPI_WORK_MODE_3	SPI mode 3

15.4.3 spi_frame_format_t

15.4.3.1 Description

SPI frame format.

15.4.3.2 Type definition

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

15.4.3.3 Enumeration element

Element name	Description
SPI_FF_STANDARD	Standard
SPI_FF_DUAL	Dual wire (2 wires)
SPI_FF_QUAD	Duad wire (4 wires)
SPI_FF_OCTAL	Octal wire (8 wires, SPI3 is not supported)

15.4.4 spi_instruction_address_trans_mode_t

15.4.4.1 Description

The transfer mode of the SPI instruction and address.

15.4.4.2 Type definition

```
typedef enum _spi_instruction_address_trans_mode
{
    SPI_AITM_STANDARD,
    SPI_AITM_ADDR_STANDARD,
    SPI_AITM_AS_FRAME_FORMAT
} spi_instruction_address_trans_mode_t;
```

15.4.4.3 Enumeration element

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

Chapter 16

I²S

16.1 Overview

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

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

16.2 Features

The I²S module has the following features:

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

16.3 API

Corresponding header file `i2s.h`
Provide the following interfaces

- i2s_init

- i2s_send_data_dma
- i2s_recv_data_dma
- i2s_rx_channel_config
- i2s_tx_channel_config
- i2s_play
- i2s_set_sample_rate: I²S set the sampling rate.
- i2s_set_dma_divide_16: Set dma*divide*16, set dma*divide*16 for 16-bit data, and automatically divide 32-bit INT32 data into two 16-bit left and right channel data during DMA transfer.
- i2s_get_dma_divide_16: Get the dma*divide*16 value. Used to determine if dma*divide*16 needs to be set.

16.3.1 i2s_init

16.3.1.1 Description

Initialize I²S.

16.3.1.2 Function prototype

```
void i2s_init(i2s_device_number_t device_num, i2s_transmit_t rxtx_mode, uint32_t
    channel_mask)
```

16.3.1.3 Parameter

Element name	Description	Input or output
device_num	I²S number	Input
rxtx_mode	Receive or transmit mode	Input
channel_mask	Channel mask	Input

16.3.1.4 Return value

None.

16.3.2 i2s_send_data_dma

16.3.2.1 Description

I²S sends data over DMA.

16.3.2.2 Function prototype

```
void i2s_send_data_dma(i2s_device_number_t device_num, const void *buf, size_t buf_len,
     dmac_channel_number_t channel_num)
```

16.3.2.3 Parameter

Element name	Description	Input or output
device_num	I²S number	Input
buf	Send data address	Input
buf_len	Data length	Input
channel_num	DMA channel number	Input

16.3.2.4 Return value

None.

16.3.3 i2s_recv_data_dma

16.3.3.1 Description

I²S receives data.

16.3.3.2 Function prototype

```
void i2s_recv_data_dma(i2s_device_number_t device_num, uint32_t *buf, size_t buf_len,
    dmac_channel_number_t channel_num)
```

16.3.3.3 Parameter

Element name	Description	Input or output
device_num	I²S number	Input
buf	Receive data address	Output
buf_len	Data length	Input
channel_num	DMA channel number	Input

16.3.3.4 Return value

None.

16.3.4 i2s_rx_channel_config

16.3.4.1 Description

Set the receive channel parameters.

16.3.4.2 Function prototype

```
void i2s_rx_channel_config(i2s_device_number_t device_num, i2s_channel_num_t
    channel_num, i2s_word_length_t word_length, i2s_word_select_cycles_t
    word_select_size, i2s_fifo_threshold_t trigger_level, i2s_work_mode_t word_mode)
```

16.3.4.3 Parameter

Element name	Description	Input or output
device_num	I²S number	Input
channel_num	Channel number	Input
word_length	Word length (bits)	Output
word_select_size	Word select size	Input
trigger_level	FIFO depth when DMA is triggered	Input
word_mode	Word mode	Input

16.3.4.4 Return value

None.

16.3.5 i2s_tx_channel_config

16.3.5.1 Description

Set the send channel parameters.

16.3.5.2 Function prototype

```
void i2s_tx_channel_config(i2s_device_number_t device_num, i2s_channel_num_t
    channel_num, i2s_word_length_t word_length, i2s_word_select_cycles_t
    word_select_size, i2s_fifo_threshold_t trigger_level, i2s_work_mode_t word_mode)
```

16.3.5.3 Parameter

Element name	Description	Input or output
device_num	I²S number	Input
channel_num	Channel number	Input
word_length	Word length (bits)	Output
word_select_size	Word select size	Input
trigger_level	FIFO depth when DMA is triggered	Input
word_mode	Word mode	Input

16.3.5.4 Return value

None.

16.3.6 i2s_play

16.3.6.1 Description

Send PCM data, such as playing music

16.3.6.2 Function prototype

```
void i2s_play(i2s_device_number_t device_num, dmac_channel_number_t channel_num,
              const uint8_t *buf, size_t buf_len, size_t frame, size_t bits_per_sample,
                   uint8_t track_num)
```

16.3.6.3 Parameter

Element name	Description	Input or output
device_num	I²S number	Input
channel_num	Channel number	Input
buf	PCM data	Input
buf_len	PCM data length	Input
frame	Single send quantity	Input
bits_per_sample	Single sampling bit width	Input
track_num	Number of channels	Input

16.3.6.4 Return value

None.

16.3.7 i2s_set_sample_rate

16.3.7.1 Description

Set the sampling rate.

16.3.7.2 Function prototype

```
uint32_t i2s_set_sample_rate(i2s_device_number_t device_num, uint32_t sample_rate)
```

16.3.7.3 Parameter

Element name	Description	Input or output
device_num	I²S number	Input
sample_rate	Sampling Rate	Input

16.3.7.4 Return value

Actual sampling rate.

16.3.8 i2s_set_dma_divide_16

16.3.8.1 Description

Set dma_divide_16, set dma_divide_16 for 16-bit data, and automatically split 32-bit INT32 data into two 16-bit left and right channel data during DMA transfer.

16.3.8.2 Function prototype

```
int i2s_set_dma_divide_16(i2s_device_number_t device_num, uint32_t enable)
```

16.3.8.3 Parameter

Element name	Description	Input or output
device_num	I²S number	Input
enable	0: Disable 1: Enable	Input

16.3.8.4 Return value

Return value	Description
0	Success
Others	Fail

16.3.9 i2s_get_dma_divide_16

16.3.9.1 Description

Get the dma_divide_16 value. Used to determine if you need to set dma_divide_16.

16.3.9.2 Function prototype

```
int i2s_get_dma_divide_16(i2s_device_number_t device_num)
```

16.3.9.3 Parameter

Element name	Description	Input or output
device_num	I²S number	Input

16.3.9.4 Return value

Return value	Description
1	Enable
0	Disable
<0	Fail

16.3.10 Example

```
/*
 * I2S0 Channel 0 is set to receive channel, receive 16-bit data, 32 clocks in a
 * single transmission, FIFO depth is 4, standard mode. Receive 8 sets of data.
 * I2S2 Channel 1 is set to transmit channel, transmitting 16-bit data, 32
 * clocks in a single transmission, FIFO depth is 4, right-aligned mode. Send 8
 * sets of data
 */
uint32_t buf[8];
i2s_init(I2S_DEVICE_0, I2S_RECEIVER, 0x3);
```

```
i2s_init(I2S_DEVICE_2, I2S_TRANSMITTER, 0xC);
i2s_rx_channel_config(I2S_DEVICE_0, I2S_CHANNEL_0, RESOLUTION_16_BIT, SCLK_CYCLES_32,
    TRIGGER_LEVEL_4, STANDARD_MODE);
i2s_tx_channel_config(I2S_DEVICE_2, I2S_CHANNEL_1, RESOLUTION_16_BIT, SCLK_CYCLES_32,
    TRIGGER_LEVEL_4, RIGHT_JUSTIFYING_MODE);
i2s_recv_data_dma(I2S_DEVICE_0, rx_buf, 8, DMAC_CHANNEL1);
i2s_send_data_dma(I2S_DEVICE_2, buf, 8, DMAC_CHANNEL0);
```

16.4 Data type

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

- i2s_device_number_t: I²S device number.
- i2s_channel_num_t: I²S channel number.
- i2s_transmit_t: I²S transmission mode.
- i2s_work_mode_t: I²S working mode.
- i2s_word_select_cycles_t: The number of I²S single transmission clocks.
- i2s_word_length_t: The number of data bits transmitted by I²S.
- i2s_fifo_threshold_t: I²S FIFO depth.

16.4.1 i2s_device_number_t

16.4.1.1 Description

I²S device number.

16.4.1.2 Type definition

```
typedef enum _i2s_device_number
{
    I2S_DEVICE_0 = 0,
    I2S_DEVICE_1 = 1,
    I2S_DEVICE_2 = 2,
    I2S_DEVICE_MAX
} i2s_device_number_t;
```

16.4.1.3 Enumeration element

Element name	Description
I2S_DEVICE_0	I²S 0
I2S_DEVICE_1	I²S 1

Element name	Description
I2S_DEVICE_2	I²S 2

16.4.2 i2s_channel_num_t

16.4.2.1 Description

I²S channel number.

16.4.2.2 Type definition

```
typedef enum _i2s_channel_num
{
    I2S_CHANNEL_0 = 0,
    I2S_CHANNEL_1 = 1,
    I2S_CHANNEL_2 = 2,
    I2S_CHANNEL_3 = 3
} i2s_channel_num_t;
```

16.4.2.3 Enumeration element

Element name	Description
I2S_CHANNEL_0	I²S channel 0
I2S_CHANNEL_1	I²S channel 1
I2S_CHANNEL_2	I²S channel 2
I2S_CHANNEL_3	I²S channel 3

16.4.3 i2s_transmit_t

16.4.3.1 Description

I²S transmission mode.

16.4.3.2 Type definition

```
typedef enum _i2s_transmit
{
    I2S_TRANSMITTER = 0,
    I2S_RECEIVER = 1
} i2s_transmit_t;
```

16.4.3.3 Enumeration element

Element name	Description
I2S_TRANSMITTER	Send mode
I2S_RECEIVER	Receive mode

16.4.4 i2s_work_mode_t

16.4.4.1 Description

I2S working mode.

16.4.4.2 Type definition

```
typedef enum _i2s_work_mode
{
    STANDARD_MODE = 1,
    RIGHT_JUSTIFYING_MODE = 2,
    LEFT_JUSTIFYING_MODE = 4
} i2s_work_mode_t;
```

16.4.4.3 Enumeration element

Element name	Description
STANDARD_MODE	Standard mode
RIGHT_JUSTIFYING_MODE	Right justified mode
LEFT_JUSTIFYING_MODE	Left justified mode

16.4.5 i2s_word_select_cycles_t

16.4.5.1 Description

The number of I²S single transmission clocks.

16.4.5.2 Type definition

```
typedef enum _word_select_cycles
{
    SCLK_CYCLES_16 = 0x0,
    SCLK_CYCLES_24 = 0x1,
```

```
    SCLK_CYCLES_32 = 0x2
} i2s_word_select_cycles_t;
```

16.4.5.3 Enumeration element

Element name	Description
SCLK_CYCLES_16	16 clock cycles
SCLK_CYCLES_24	24 clock cycles
SCLK_CYCLES_32	32 clock cycles

16.4.6 i2s_word_length_t

16.4.6.1 Description

The number of data bits transmitted by I²S.

16.4.6.2 Type definition

```
typedef enum _word_length
{
    IGNORE_WORD_LENGTH = 0x0,
    RESOLUTION_12_BIT = 0x1,
    RESOLUTION_16_BIT = 0x2,
    RESOLUTION_20_BIT = 0x3,
    RESOLUTION_24_BIT = 0x4,
    RESOLUTION_32_BIT = 0x5
} i2s_word_length_t;
```

16.4.6.3 Enumeration element

Element name	Description
IGNORE_WORD_LENGTH	Ignore word length
RESOLUTION_12_BIT	12 bit data length
RESOLUTION_16_BIT	16 bit data length
RESOLUTION_20_BIT	20 bit data length
RESOLUTION_24_BIT	24 bit data length
RESOLUTION_32_BIT	32 bit data length

16.4.7 i2s_fifo_threshold_t

16.4.7.1 Description

I²S FIFO depth.

16.4.7.2 Type definition

```
typedef enum _fifo_threshold
{
    /* Interrupt trigger when FIFO level is 1 */
    TRIGGER_LEVEL_1 = 0x0,
    /* Interrupt trigger when FIFO level is 2 */
    TRIGGER_LEVEL_2 = 0x1,
    /* Interrupt trigger when FIFO level is 3 */
    TRIGGER_LEVEL_3 = 0x2,
    /* Interrupt trigger when FIFO level is 4 */
    TRIGGER_LEVEL_4 = 0x3,
    /* Interrupt trigger when FIFO level is 5 */
    TRIGGER_LEVEL_5 = 0x4,
    /* Interrupt trigger when FIFO level is 6 */
    TRIGGER_LEVEL_6 = 0x5,
    /* Interrupt trigger when FIFO level is 7 */
    TRIGGER_LEVEL_7 = 0x6,
    /* Interrupt trigger when FIFO level is 8 */
    TRIGGER_LEVEL_8 = 0x7,
    /* Interrupt trigger when FIFO level is 9 */
    TRIGGER_LEVEL_9 = 0x8,
    /* Interrupt trigger when FIFO level is 10 */
    TRIGGER_LEVEL_10 = 0x9,
    /* Interrupt trigger when FIFO level is 11 */
    TRIGGER_LEVEL_11 = 0xa,
    /* Interrupt trigger when FIFO level is 12 */
    TRIGGER_LEVEL_12 = 0xb,
    /* Interrupt trigger when FIFO level is 13 */
    TRIGGER_LEVEL_13 = 0xc,
    /* Interrupt trigger when FIFO level is 14 */
    TRIGGER_LEVEL_14 = 0xd,
    /* Interrupt trigger when FIFO level is 15 */
    TRIGGER_LEVEL_15 = 0xe,
    /* Interrupt trigger when FIFO level is 16 */
    TRIGGER_LEVEL_16 = 0xf
} i2s_fifo_threshold_t;
```

16.4.7.3 Enumeration element

Element name	Description
TRIGGER_LEVEL_1	1 Byte FIFO depth

Element name	Description
TRIGGER_LEVEL_2	2 Byte FIFO depth
TRIGGER_LEVEL_3	3 Byte FIFO depth
TRIGGER_LEVEL_4	4 Byte FIFO depth
TRIGGER_LEVEL_5	5 Byte FIFO depth
TRIGGER_LEVEL_6	6 Byte FIFO depth
TRIGGER_LEVEL_7	7 Byte FIFO depth
TRIGGER_LEVEL_8	8 Byte FIFO depth
TRIGGER_LEVEL_9	9 Byte FIFO depth
TRIGGER_LEVEL_10	10 Byte FIFO depth
TRIGGER_LEVEL_11	11 Byte FIFO depth
TRIGGER_LEVEL_12	12 Byte FIFO depth
TRIGGER_LEVEL_13	13 Byte FIFO depth
TRIGGER_LEVEL_14	14 Byte FIFO depth
TRIGGER_LEVEL_15	15 Byte FIFO depth
TRIGGER_LEVEL_16	16 Byte FIFO depth

Chapter 17

TIMER

17.1 Overview

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

17.2 Features

The TIMER module has the following features:

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

17.3 API

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

- timer_init
- timer_set_interval
- timer_set_irq (deprecated)
- timer_set_enable
- timer_irq_register
- timer_irq_deregister

17.3.1 timer_init

17.3.1.1 Description

Initialize timer.

17.3.1.2 Function prototype

```
void timer_init(timer_device_number_t timer_number)
```

17.3.1.3 Parameter

Parameter name	Description	Input or output
timer_number	Timer number	Input

17.3.1.4 Return value

None.

17.3.2 timer_set_interval

17.3.2.1 Description

Set the timer trigger interval.

17.3.2.2 Function prototype

```
size_t timer_set_interval(timer_device_number_t timer_number, timer_channel_number_t
    channel, size_t nanoseconds)
```

17.3.2.3 Parameter

Parameter name	Description	Input or output
timer_number	Timer number	Input
channel	Timer channel number	Input
nanoseconds	Interval (nanoseconds)	Input

17.3.2.4 Return value

Actual trigger interval (nanoseconds).

17.3.3 timer_set_irq

17.3.3.1 Description

Set the timer to trigger the interrupt callback function. This function is deprecated. The recommended replacement function is timer_irq_register.

17.3.3.2 Function prototype

```
void timer_set_irq(timer_device_number_t timer_number, timer_channel_number_t channel,
    void(*func)(),  uint32_t priority)
```

17.3.3.3 Parameter

Parameter name	Description	Input or output
timer_number	Timer number	Input
channel	Timer channel number	Input
func	Callback	Input
priority	Interrupt priority	Input

17.3.3.4 Return value

None.

17.3.4 timer_set_enable

17.3.4.1 Description

Enable or disable the timer.

17.3.4.2 Function prototype

```
void timer_set_enable(timer_device_number_t timer_number, timer_channel_number_t
    channel, uint32_t enable)
```

17.3.4.3 Parameter

Parameter name	Description	Input or output
timer_number	Timer number	Input
channel	Timer channel number	Input
enable	Whether it is enabled, 0: disable 1: enable	Input

17.3.4.4 Return value

None.

17.3.5 timer_irq_register

17.3.5.1 Description

Register the timer interrupt callback function.

17.3.5.2 Function prototype

```
int timer_irq_register(timer_device_number_t device, timer_channel_number_t channel,
    int is_single_shot, uint32_t priority, timer_callback_t callback, void *ctx);
```

17.3.5.3 Parameter

Parameter name	Description	Input or output
device	Timer number	Input
channel	Timer channel number	Input
is_single_shot	Whether it is a single shot interruption	Input
priority	Interrupt priority	Input
callback	Interrupt callback function	Input
ctx	Callback function parameter	Input

17.3.5.4 Return value

Return value	Description
0	Success
Others	Fail

17.3.6 timer_irq_deregister

17.3.6.1 Description

Deregister the timer interrupt callback function.

17.3.6.2 Function prototype

```
int timer_irq_deregister(timer_device_number_t device, timer_channel_number_t channel)
```

17.3.6.3 Parameter

Parameter name	Description	Input or output
device	Timer number	Input
channel	Timer channel number	Input

17.3.6.4 Return value

Return value	Description
0	Success
Others	Fail

17.3.7 Example

```
/* Timer 0 Channel 0 timed 1 second print "Time OK!" */
void irq_time(void)
{
    printf("Time OK!\n");
}
plic_init();
timer_init(TIMER_DEVICE_0);
timer_set_interval(TIMER_DEVICE_0, TIMER_CHANNEL_0, 1e9);
timer_set_irq(TIMER_CHANNEL_0, TIMER_CHANNEL_0, irq_time, 1);
timer_set_enable(TIMER_CHANNEL_0, TIMER_CHANNEL_0, 1);
sysctl_enable_irq();
```

17.4 Data type

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

- timer_device_number_t: Timer number.
- timer_channel_number_t: Timer channel number.
- timer_callback_t: Timer callback function.

17.4.1 timer_device_number_t

17.4.1.1 Description

Timer number.

17.4.1.2 Type definition

```
typedef enum _timer_deivce_number
{
    TIMER_DEVICE_0,
    TIMER_DEVICE_1,
    TIMER_DEVICE_2,
    TIMER_DEVICE_MAX,
} timer_device_number_t;
```

17.4.1.3 Enumeration element

Element name	Description
TIMER_DEVICE_0	Timer 0
TIMER_DEVICE_1	Timer 1
TIMER_DEVICE_2	Timer 2

17.4.2 timer_channel_number_t

17.4.2.1 Description

Timer channel number.

17.4.2.2 Type definition

```
typedef enum _timer_channel_number
{
    TIMER_CHANNEL_0,
    TIMER_CHANNEL_1,
    TIMER_CHANNEL_2,
    TIMER_CHANNEL_3,
    TIMER_CHANNEL_MAX,
```

```
} timer_channel_number_t;
```

17.4.2.3 Enumeration element

Element name	Description
TIMER_CHANNEL_0	Timer channel 0
TIMER_CHANNEL_1	Timer channel 1
TIMER_CHANNEL_2	Timer channel 2
TIMER_CHANNEL_3	Timer channel 3

17.4.3 timer_callback_t

17.4.3.1 Description

Timer callback function.

17.4.3.2 Type definition

```
typedef int (*timer_callback_t)(void *ctx);
```

Chapter 18

RTC

18.1 Overview

The Real Time Clock (RTC) is a unit for timing and has a timing function after the set time.

Note The RTC unit is only used when PLL0 is enabled and the CPU frequency is greater than 30MHz.

18.2 Features

The RTC unit has the following features:

- Get current date and time
- Set the current date and time
- Set timing interrupt
- Set alarm interrupt

18.3 API

Corresponding header file `rtc.h`

Provide the following interfaces

- rtc_init
- rtc_timer_set
- rtc_timer_get

18.3.1 rtc_init

18.3.1.1 Description

Initialize the RTC.

18.3.1.2 Function prototype

```
int rtc_init(void)
```

18.3.1.3 Parameter

None.

18.3.1.4 Return value

Return value	Description
0	Success
Others	Fail

18.3.2 rtc_timer_set

18.3.2.1 Description

Set the RTC date and time.

18.3.2.2 Function prototype

```
int rtc_timer_set(int year, int month, int day, int hour, int minute, int second)
```

18.3.2.3 Parameter

Parameter name	Description	Input or output
year	Year	Input
month	Month	Input
day	Day	Input
hour	Hour	Input
minute	Minute	Input

Parameter name	Description	Input or output
second	Second	Input

18.3.2.4 Return value

None.

18.3.3 rtc_timer_get

18.3.3.1 Description

Get the RTC date and time.

18.3.3.2 Function prototype

```
int rtc_timer_get(int *year, int *month, int *day, int *hour, int *minute, int *second)
```

18.3.3.3 Parameter

Parameter name	Description	Input or output
year	Year	Output
month	Month	Output
day	Day	Output
hour	Hour	Output
minute	Minute	Output
second	Second	Output

18.3.3.4 Return value

Return value	Description
0	Success
Others	Fail

18.3.4 Example

```
rtc_init();
rtc_timer_set(2018, 9, 12, 23, 30, 29);
```

```
int year;
int month;
int day;
int hour;
int minute;
int second;
rtc_timer_get(&year, &month, &day, &hour, &minute, &second);
printf("%4d-%d-%d %d:%d:%d\n", year, month, day, hour, minute, second);
```

Chapter 19

PWM

19.1 Overview

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

19.2 Features

The PWM module has the following features:

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

19.3 API

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

- pwm_init
- pwm_set_frequency
- pwm_set_enable

19.3.1 pwm_init

19.3.1.1 Description

Initialize PWM.

19.3.1.2 Function prototype

```
void pwm_init(pwm_device_number_t pwm_number)
```

19.3.1.3 Parameter

Parameter name	Description	Input or output
pwm_number	PWM number	Input

19.3.1.4 Return value

None.

19.3.2 pwm_set_frequency

19.3.2.1 Description

Set the frequency and duty cycle.

19.3.2.2 Function prototype

```
double pwm_set_frequency(pwm_device_number_t pwm_number, pwm_channel_number_t channel,
    double frequency, double duty)
```

19.3.2.3 Parameter

Parameter name	Description	Input or output
pwm_number	PWM number	Input
channel	PWM channel number	Input
frequency	PWM output frequency	Input
duty	Duty cycle	Input

19.3.2.4 Return value

Actual output frequency.

19.3.3 pwmsetenable

19.3.3.1 Description

Enable or disable the PWM.

19.3.3.2 Function prototype

```
void pwm_set_enable(pwm_device_number_t pwm_number, uint32_t channel, int enable)
```

19.3.3.3 Parameter

Parameter name	Description	Input or output
pwm_number	PWM number	Input
channel	PWM channel number	Input
enable	Whether to enable, 0: Disable 1: Enable	Input

19.3.3.4 Return value

None.

19.3.4 Example

```
/* pwm0 pin0 output 200KHz square wave with duty cycle of 0.5 */
/* Set IO13 as the output pin of PWM */
fpioa_set_function(13, FUNC_TIMER0_TOGGLE1);
pwm_init(PWM_DEVICE_0);
pwm_set_frequency(PWM_DEVICE_0, PWM_CHANNEL_1, 200000, 0.5);
pwm_set_enable(PWM_DEVICE_0, PWM_CHANNEL_1, 1);
```

19.4 Data type

- pwm_device_number_t: PWM device number.
- pwm_channel_number_t: PWM channel number.

19.4.1 pwm_device_number_t

19.4.1.1 Description

PWM device number.

19.4.1.2 Type definition

```
typedef enum _pwm_device_number
{
    PWM_DEVICE_0,
    PWM_DEVICE_1,
    PWM_DEVICE_2,
    PWM_DEVICE_MAX,
} pwm_device_number_t;
```

19.4.1.3 Enumeration element

Element name	Description
PWM_DEVICE_0	PWM0
PWM_DEVICE_1	PWM1
PWM_DEVICE_2	PWM2

19.4.2 pwm_channel_number_t

19.4.2.1 Description

PWM channel number.

19.4.2.2 Type definition

```
typedef enum _pwm_channel_number
{
    PWM_CHANNEL_0,
    PWM_CHANNEL_1,
    PWM_CHANNEL_2,
    PWM_CHANNEL_3,
    PWM_CHANNEL_MAX,
} pwm_channel_number_t;
```

19.4.2.3 Enumeration element

Element name	Description
PWM_CHANNEL_0	PWM channel 0
PWM_CHANNEL_1	PWM channel 1
PWM_CHANNEL_2	PWM channel 2
PWM_CHANNEL_3	PWM channel 3

Chapter 20

SYSCTL

20.1 Overview

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

20.2 Features

The system control module has the following features:

- Set the PLL CPU clock frequency.
- Set the division value of each module clock.
- Get the clock frequency of each module.
- Enable, disable, reset each module.
- Set the DMA request source.
- Enable or disable system interrupts.

20.3 API

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

- sysctl_cpu_set_freq
- sysctl_pll_set_freq
- sysctl_pll_get_freq
- sysctl_pll_enable

- sysctl_pll_disable
- sysctl_clock_set_threshold
- sysctl_clock_get_threshold
- sysctl_clock_set_clock_select
- sysctl_clock_get_clock_select
- sysctl_clock_get_freq
- sysctl_clock_enable
- sysctl_clock_disable
- sysctl_reset
- sysctl_dma_select
- sysctl_set_power_mode
- sysctl_enable_irq
- sysctl_disable_irq

20.3.1 sysctl_cpu_set_freq

20.3.1.1 Description

Set the CPU operating frequency. It is achieved by modifying the frequency of PLL0.

20.3.1.2 Function prototype

```
uint32_t sysctl_cpu_set_freq(uint32_t freq)
```

20.3.1.3 Parameter

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

20.3.1.4 Return value

The actual frequency (Hz) after setting.

20.3.2 sysctl_set_pll_frequency

20.3.2.1 Description

Set the PLL frequency.

20.3.2.2 Function prototype

```
uint32_t sysctl_pll_set_freq(sysctl_pll_t pll, uint32_t pll_freq)
```

20.3.2.3 Parameter

Parameter name	Description	Input or output
pll	PLL number	Input
pll_freq	Frequency to be set (Hz)	Input

20.3.2.4 Return value

The actual frequency (Hz) after setting.

20.3.2.5 Function prototype

```
uint32_t sysctl_pll_get_freq(sysctl_pll_t pll)
```

20.3.2.6 Parameter

Parameter name	Description	Input or output
pll	PLL number	Input

20.3.2.7 Return value

The frequency of the PLL (Hz).

20.3.3 sysctl_pll_enable

20.3.3.1 Description

Enable the PLL.

20.3.3.2 Function prototype

```
int sysctl_pll_enable(sysctl_pll_t pll)
```

20.3.3.3 Parameter

Parameter name	Description	Input or output
pll	PLL number	Input

20.3.3.4 Return value

Return value	Description
0	Success
Others	Fail

20.3.4 sysctl_pll_disable

20.3.4.1 Description

Disable the PLL.

20.3.4.2 Function prototype

```
int sysctl_pll_disable(sysctl_pll_t pll)
```

20.3.4.3 Parameter

Parameter name	Description	Input or output
pll	PLL number	Input

20.3.4.4 Return value

Return value	Description
0	Success
Others	Fail

20.3.5 sysctl_clock_set_threshold

20.3.5.1 Description

Set the division value of the clock.

20.3.5.2 Function prototype

```
void sysctl_clock_set_threshold(sysctl_threshold_t which, int threshold)
```

20.3.5.3 Parameter

Parameter name	Description	Input or output
which	The clock to be select	Input
threshold	Frequency division value	Input

20.3.5.4 Return value

Return value	Description
0	Success
Others	Fail

20.3.6 sysctl_clock_get_threshold

20.3.6.1 Description

Get the division value of the clock.

20.3.6.2 Function prototype

```
int sysctl_clock_get_threshold(sysctl_threshold_t which)
```

Parameter name	Description	Input or output
which	The clock to be select	Input

20.3.6.3 Return value

The division value of the clock.

20.3.7 sysctl_clock_set_clock_select

20.3.7.1 Description

Set the clock source.

20.3.7.2 Function prototype

```
int sysctl_clock_set_clock_select(sysctl_clock_select_t which, int select)
```

20.3.7.3 Parameter

Parameter name	Description	Input or output
which	The clock to be select	Input
select	The clock source to be select	Input

20.3.7.4 Return value

Return value	Description
0	Success
Others	Fail

20.3.8 sysctl_clock_get_clock_select

20.3.8.1 Description

Get the clock source corresponding to the clock.

20.3.8.2 Function prototype

```
int sysctl_clock_get_clock_select(sysctl_clock_select_t which)
```

20.3.8.3 Parameter

Parameter name	Description	Input or output
which	The clock to be select	Input

20.3.8.4 Return value

The clock source corresponding to the clock.

20.3.9 sysctl_clock_get_freq

20.3.9.1 Description

Get the frequency of the clock.

20.3.9.2 Function prototype

```
uint32_t sysctl_clock_get_freq(sysctl_clock_t clock)
```

20.3.9.3 Parameter

Parameter name	Description	Input or output
clock	The clock to be select	Input

20.3.9.4 Return value

Clock frequency (Hz)

20.3.10 sysctl_clock_enable

20.3.10.1 Description

Enable the clock. If you enable the PLL you need to use sysctl_pll_enable.

20.3.10.2 Function prototype

```
int sysctl_clock_enable(sysctl_clock_t clock)
```

20.3.10.3 Parameter

Parameter name	Description	Input or output
clock	The clock to be select	Input

20.3.10.4 Return value

Return value	Description
0	Success
Others	Fail

20.3.11 sysctl_clock_disable

20.3.11.1 Description

Disable the clock. If you disable the PLL you need to use sysctl_pll_disable.

20.3.11.2 Function prototype

```
int sysctl_clock_disable(sysctl_clock_t clock)
```

20.3.11.3 Parameter

Parameter name	Description	Input or output
clock	The clock to be select	Input

20.3.11.4 Return value

Return value	Description
0	Success
Others	Fail

20.3.12 sysctl_reset

20.3.12.1 Description

Reset a peripheral.

20.3.12.2 Function prototype

```
void sysctl_reset(sysctl_reset_t reset)
```

20.3.12.3 Parameter

Parameter name	Description	Input or output
reset	The peripheral to be reset	Input

20.3.12.4 Return value

None.

20.3.13 sysctl_dma_select

20.3.13.1 Description

Set the DMA request source. Used in conjunction with the DMAC API.

20.3.13.2 Function prototype

```
int sysctl_dma_select(sysctl_dma_channel_t channel, sysctl_dma_select_t select)
```

20.3.13.3 Parameter

Parameter name	Description	Input or output
channel	DMA channel number	Input
select	DMA request source	Input

20.3.13.4 Return value

Return value	Description
0	Success
Others	Fail

20.3.14 sysctl_set_power_mode

20.3.14.1 Description

Select the voltage of the corresponding power domain of the FPIOA.

20.3.14.2 Function prototype

```
void sysctl_set_power_mode(sysctl_power_bank_t power_bank, sysctl_io_power_mode_t
    io_power_mode)
```

20.3.14.3 Parameter

Parameter name	Description	Input or output
power_bank	IO power domain number	Input
io_power_mode	Select the voltage. 0: 3.3V, 1: 1.8V	Input

20.3.14.4 Return value

None.

20.3.15 sysctl_enable_irq

20.3.15.1 Description

Enable system interrupts, if you use interrupts, you must turn on system interrupts.

20.3.15.2 Function prototype

```
void sysctl_enable_irq(void)
```

20.3.15.3 Parameter

None.

20.3.15.4 Return value

None.

20.3.16 sysctl_disable_irq

20.3.16.1 Description

Disable system interrupts.

20.3.16.2 Function prototype

```
void sysctl_disable_irq(void)
```

20.3.16.3 Parameter

None.

20.3.16.4 Return value

None.

20.4 Data type

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

- sysctl_pll_t: PLL number.
- sysctl_threshold_t: The peripheral number when setting the divider value.
- sysctl_clock_select_t: The peripheral number when setting the clock source.
- sysctl_clock_t: The peripheral number when setting the clock.
- sysctl_reset_t: The peripheral number when reset.
- sysctl_dma_channel_t: DMA channel number.
- sysctl_dma_select_t: DMA request source number.
- sysctl_power_bank_t: Power domain number.
- sysctl_io_power_mode_t: IO power domain voltage selection.

20.4.1 sysctl_pll_t

20.4.1.1 Description

PLL number.

20.4.1.2 Type definition

```
typedef enum _sysctl_pll_t
{
    SYSCTL_PLL0,
    SYSCTL_PLL1,
    SYSCTL_PLL2,
    SYSCTL_PLL_MAX
} sysctl_pll_t;
```

20.4.1.3 Enumeration element

Element name	Description
SYSCTL_PLL0	PLL0
SYSCTL_PLL1	PLL1

Element name	Description
SYSCTL_PLL2	PLL2

20.4.2 sysctl_threshold_t

20.4.2.1 Description

The peripheral number when setting the divider value.

20.4.2.2 Type definition

```
typedef enum _sysctl_threshold_t
{
    SYSCTL_THRESHOLD_ACLK,
    SYSCTL_THRESHOLD_APB0,
    SYSCTL_THRESHOLD_APB1,
    SYSCTL_THRESHOLD_APB2,
    SYSCTL_THRESHOLD_SRAM0,
    SYSCTL_THRESHOLD_SRAM1,
    SYSCTL_THRESHOLD_AI,
    SYSCTL_THRESHOLD_DVP,
    SYSCTL_THRESHOLD_ROM,
    SYSCTL_THRESHOLD_SPI0,
    SYSCTL_THRESHOLD_SPI1,
    SYSCTL_THRESHOLD_SPI2,
    SYSCTL_THRESHOLD_SPI3,
    SYSCTL_THRESHOLD_TIMER0,
    SYSCTL_THRESHOLD_TIMER1,
    SYSCTL_THRESHOLD_TIMER2,
    SYSCTL_THRESHOLD_I2S0,
    SYSCTL_THRESHOLD_I2S1,
    SYSCTL_THRESHOLD_I2S2,
    SYSCTL_THRESHOLD_I2S0_M,
    SYSCTL_THRESHOLD_I2S1_M,
    SYSCTL_THRESHOLD_I2S2_M,
    SYSCTL_THRESHOLD_I2C0,
    SYSCTL_THRESHOLD_I2C1,
    SYSCTL_THRESHOLD_I2C2,
    SYSCTL_THRESHOLD_WDT0,
    SYSCTL_THRESHOLD_WDT1,
    SYSCTL_THRESHOLD_MAX = 28
} sysctl_threshold_t;
```

20.4.2.3 Enumeration element

Element name	Description
SYSCTL_THRESHOLD_ACLK	ACLK
SYSCTL_THRESHOLD_APB0	APB0
SYSCTL_THRESHOLD_APB1	APB1
SYSCTL_THRESHOLD_APB2	ACLK
SYSCTL_THRESHOLD_SRAM0	SRAM0
SYSCTL_THRESHOLD_SRAM1	SRAM1
SYSCTL_THRESHOLD_AI	AI
SYSCTL_THRESHOLD_DVP	DVP
SYSCTL_THRESHOLD_ROM	ROM
SYSCTL_THRESHOLD_SPI0	SPI0
SYSCTL_THRESHOLD_SPI1	SPI1
SYSCTL_THRESHOLD_SPI2	SPI2
SYSCTL_THRESHOLD_SPI3	SPI3
SYSCTL_THRESHOLD_TIMER0	TIMER0
SYSCTL_THRESHOLD_TIMER1	TIMER1
SYSCTL_THRESHOLD_TIMER2	TIMER2
SYSCTL_THRESHOLD_I2S0	I2S0
SYSCTL_THRESHOLD_I2S1	I2S1
SYSCTL_THRESHOLD_I2S2	I2S2
SYSCTL_THRESHOLD_I2S0_M	I2S0 MCLK
SYSCTL_THRESHOLD_I2S1_M	I2S1 MCLK
SYSCTL_THRESHOLD_I2S2_M	I2S2 MCLK
SYSCTL_THRESHOLD_I2C0	I2C0
SYSCTL_THRESHOLD_I2C1	I2C1
SYSCTL_THRESHOLD_I2C2	I2C2
SYSCTL_THRESHOLD_WDT0	WDT0
SYSCTL_THRESHOLD_WDT1	WDT1

20.4.3 sysctl_clock_select_t

20.4.3.1 Description

The peripheral number when setting the clock source.

20.4.3.2 Type definition

```
typedef enum _sysctl_clock_select_t
{
    SYSCTL_CLOCK_SELECT_PLL0_BYPASS,
    SYSCTL_CLOCK_SELECT_PLL1_BYPASS,
    SYSCTL_CLOCK_SELECT_PLL2_BYPASS,
    SYSCTL_CLOCK_SELECT_PLL2,
    SYSCTL_CLOCK_SELECT_ACLK,
    SYSCTL_CLOCK_SELECT_SPI3,
    SYSCTL_CLOCK_SELECT_TIMER0,
    SYSCTL_CLOCK_SELECT_TIMER1,
    SYSCTL_CLOCK_SELECT_TIMER2,
    SYSCTL_CLOCK_SELECT_SPI3_SAMPLE,
    SYSCTL_CLOCK_SELECT_MAX = 11
} sysctl_clock_select_t;
```

20.4.3.3 Enumeration element

Element name	Description
SYSCTL_CLOCK_SELECT_PLL0_BYPASS	PLL0_BYPASS
SYSCTL_CLOCK_SELECT_PLL1_BYPASS	PLL1_BYPASS
SYSCTL_CLOCK_SELECT_PLL2_BYPASS	PLL2_BYPASS
SYSCTL_CLOCK_SELECT_PLL2	PLL2
SYSCTL_CLOCK_SELECT_ACLK	ACLK
SYSCTL_CLOCK_SELECT_SPI3	SPI3
SYSCTL_CLOCK_SELECT_TIMER0	TIMER0
SYSCTL_CLOCK_SELECT_TIMER1	TIMER1
SYSCTL_CLOCK_SELECT_TIMER2	TIMER2
SYSCTL_CLOCK_SELECT_SPI3_SAMPLE	SPI3 data sampling clock edge selection

20.4.4 sysctl_clock_t

20.4.4.1 Description

The peripheral number when setting the clock.

20.4.4.2 Type definition

```
typedef enum _sysctl_clock_t
{
    SYSCTL_CLOCK_PLL0,
    SYSCTL_CLOCK_PLL1,
    SYSCTL_CLOCK_PLL2,
    SYSCTL_CLOCK_CPU,
```

```
    SYSCTL_CLOCK_SRAM0,
    SYSCTL_CLOCK_SRAM1,
    SYSCTL_CLOCK_APB0,
    SYSCTL_CLOCK_APB1,
    SYSCTL_CLOCK_APB2,
    SYSCTL_CLOCK_ROM,
    SYSCTL_CLOCK_DMA,
    SYSCTL_CLOCK_AI,
    SYSCTL_CLOCK_DVP,
    SYSCTL_CLOCK_FFT,
    SYSCTL_CLOCK_GPIO,
    SYSCTL_CLOCK_SPI0,
    SYSCTL_CLOCK_SPI1,
    SYSCTL_CLOCK_SPI2,
    SYSCTL_CLOCK_SPI3,
    SYSCTL_CLOCK_I2S0,
    SYSCTL_CLOCK_I2S1,
    SYSCTL_CLOCK_I2S2,
    SYSCTL_CLOCK_I2C0,
    SYSCTL_CLOCK_I2C1,
    SYSCTL_CLOCK_I2C2,
    SYSCTL_CLOCK_UART1,
    SYSCTL_CLOCK_UART2,
    SYSCTL_CLOCK_UART3,
    SYSCTL_CLOCK_AES,
    SYSCTL_CLOCK_FPIOA,
    SYSCTL_CLOCK_TIMER0,
    SYSCTL_CLOCK_TIMER1,
    SYSCTL_CLOCK_TIMER2,
    SYSCTL_CLOCK_WDT0,
    SYSCTL_CLOCK_WDT1,
    SYSCTL_CLOCK_SHA,
    SYSCTL_CLOCK_OTP,
    SYSCTL_CLOCK_RTC,
    SYSCTL_CLOCK_ACLK = 40,
    SYSCTL_CLOCK_HCLK,
    SYSCTL_CLOCK_IN0,
    SYSCTL_CLOCK_MAX
} sysctl_clock_t;
```

20.4.4.3 Enumeration element

Element name	Description
SYSCTL_CLOCK_PLL0	PLL0
SYSCTL_CLOCK_PLL1	PLL1
SYSCTL_CLOCK_PLL2	PLL2
SYSCTL_CLOCK_CPU	CPU
SYSCTL_CLOCK_SRAM0	SRAM0

Element name	Description
SYSCTL_CLOCK_SRAM1	SRAM1
SYSCTL_CLOCK_APB0	APB0
SYSCTL_CLOCK_APB1	APB1
SYSCTL_CLOCK_APB2	APB2
SYSCTL_CLOCK_ROM	ROM
SYSCTL_CLOCK_DMA	DMA
SYSCTL_CLOCK_AI	AI
SYSCTL_CLOCK_DVP	DVP
SYSCTL_CLOCK_FFT	FFT
SYSCTL_CLOCK_GPIO	GPIO
SYSCTL_CLOCK_SPI0	SPI0
SYSCTL_CLOCK_SPI1	SPI1
SYSCTL_CLOCK_SPI2	SPI2
SYSCTL_CLOCK_SPI3	SPI3
SYSCTL_CLOCK_I2S0	I2S0
SYSCTL_CLOCK_I2S1	I2S1
SYSCTL_CLOCK_I2S2	I2S2
SYSCTL_CLOCK_I2C0	I2C0
SYSCTL_CLOCK_I2C1	I2C1
SYSCTL_CLOCK_I2C2	I2C2
SYSCTL_CLOCK_UART1	UART1
SYSCTL_CLOCK_UART2	UART2
SYSCTL_CLOCK_UART3	UART3
SYSCTL_CLOCK_AES	AES
SYSCTL_CLOCK_FPIOA	FPIOA
SYSCTL_CLOCK_TIMER0	TIMER0
SYSCTL_CLOCK_TIMER1	TIMER1
SYSCTL_CLOCK_TIMER2	TIMER2
SYSCTL_CLOCK_WDT0	WDT0
SYSCTL_CLOCK_WDT1	WDT1
SYSCTL_CLOCK_SHA	SHA
SYSCTL_CLOCK_OTP	OTP
SYSCTL_CLOCK_RTC	RTC
SYSCTL_CLOCK_ACLK	ACLK
SYSCTL_CLOCK_HCLK	HCLK

Element name	Description
SYSCTL_CLOCK_IN0	External input clock IN0

20.4.5 sysctl_reset_t

20.4.5.1 Description

The peripheral number when reset.

20.4.5.2 Type definition

```
typedef enum _sysctl_reset_t
{
    SYSCTL_RESET_SOC,
    SYSCTL_RESET_ROM,
    SYSCTL_RESET_DMA,
    SYSCTL_RESET_AI,
    SYSCTL_RESET_DVP,
    SYSCTL_RESET_FFT,
    SYSCTL_RESET_GPIO,
    SYSCTL_RESET_SPI0,
    SYSCTL_RESET_SPI1,
    SYSCTL_RESET_SPI2,
    SYSCTL_RESET_SPI3,
    SYSCTL_RESET_I2S0,
    SYSCTL_RESET_I2S1,
    SYSCTL_RESET_I2S2,
    SYSCTL_RESET_I2C0,
    SYSCTL_RESET_I2C1,
    SYSCTL_RESET_I2C2,
    SYSCTL_RESET_UART1,
    SYSCTL_RESET_UART2,
    SYSCTL_RESET_UART3,
    SYSCTL_RESET_AES,
    SYSCTL_RESET_FPIOA,
    SYSCTL_RESET_TIMER0,
    SYSCTL_RESET_TIMER1,
    SYSCTL_RESET_TIMER2,
    SYSCTL_RESET_WDT0,
    SYSCTL_RESET_WDT1,
    SYSCTL_RESET_SHA,
    SYSCTL_RESET_RTC,
    SYSCTL_RESET_MAX = 31
} sysctl_reset_t;
```

20.4.5.3 Enumeration element

Element name	Description
SYSCTL_RESET_SOC	SoC
SYSCTL_RESET_ROM	ROM
SYSCTL_RESET_DMA	DMA
SYSCTL_RESET_AI	AI
SYSCTL_RESET_DVP	DVP
SYSCTL_RESET_FFT	FFT
SYSCTL_RESET_GPIO	GPIO
SYSCTL_RESET_SPI0	SPI0
SYSCTL_RESET_SPI1	SPI1
SYSCTL_RESET_SPI2	SPI2
SYSCTL_RESET_SPI3	SPI3
SYSCTL_RESET_I2S0	I2S0
SYSCTL_RESET_I2S1	I2S1
SYSCTL_RESET_I2S2	I2S2
SYSCTL_RESET_I2C0	I2C0
SYSCTL_RESET_I2C1	I2C1
SYSCTL_RESET_I2C2	I2C2
SYSCTL_RESET_UART1	UART1
SYSCTL_RESET_UART2	UART2
SYSCTL_RESET_UART3	UART3
SYSCTL_RESET_AES	AES
SYSCTL_RESET_FPIOA	FPIOA
SYSCTL_RESET_TIMER0	TIMER0
SYSCTL_RESET_TIMER1	TIMER1
SYSCTL_RESET_TIMER2	TIMER2
SYSCTL_RESET_WDT0	WDT0
SYSCTL_RESET_WDT1	WDT1
SYSCTL_RESET_SHA	SHA
SYSCTL_RESET_RTC	RTC

20.4.6 sysctl_dma_channel_t

20.4.6.1 Description

DMA channel number.

20.4.6.2 Type definition

```
typedef enum _sysctl_dma_channel_t
{
    SYSCTL_DMA_CHANNEL_0,
    SYSCTL_DMA_CHANNEL_1,
    SYSCTL_DMA_CHANNEL_2,
    SYSCTL_DMA_CHANNEL_3,
    SYSCTL_DMA_CHANNEL_4,
    SYSCTL_DMA_CHANNEL_5,
    SYSCTL_DMA_CHANNEL_MAX
} sysctl_dma_channel_t;
```

20.4.6.3 Enumeration element

Element name	Description
SYSCTL_DMA_CHANNEL_0	DMA channel 0
SYSCTL_DMA_CHANNEL_1	DMA channel 1
SYSCTL_DMA_CHANNEL_2	DMA channel 2
SYSCTL_DMA_CHANNEL_3	DMA channel 3
SYSCTL_DMA_CHANNEL_4	DMA channel 4
SYSCTL_DMA_CHANNEL_5	DMA channel 5

20.4.7 sysctl_dma_select_t

20.4.7.1 Description

DMA request source number.

20.4.7.2 Type definition

```
typedef enum _sysctl_dma_select_t
{
    SYSCTL_DMA_SELECT_SSI0_RX_REQ,
    SYSCTL_DMA_SELECT_SSI0_TX_REQ,
    SYSCTL_DMA_SELECT_SSI1_RX_REQ,
    SYSCTL_DMA_SELECT_SSI1_TX_REQ,
    SYSCTL_DMA_SELECT_SSI2_RX_REQ,
    SYSCTL_DMA_SELECT_SSI2_TX_REQ,
    SYSCTL_DMA_SELECT_SSI3_RX_REQ,
    SYSCTL_DMA_SELECT_SSI3_TX_REQ,
    SYSCTL_DMA_SELECT_I2C0_RX_REQ,
    SYSCTL_DMA_SELECT_I2C0_TX_REQ,
```

```
    SYSCTL_DMA_SELECT_I2C1_RX_REQ,
    SYSCTL_DMA_SELECT_I2C1_TX_REQ,
    SYSCTL_DMA_SELECT_I2C2_RX_REQ,
    SYSCTL_DMA_SELECT_I2C2_TX_REQ,
    SYSCTL_DMA_SELECT_UART1_RX_REQ,
    SYSCTL_DMA_SELECT_UART1_TX_REQ,
    SYSCTL_DMA_SELECT_UART2_RX_REQ,
    SYSCTL_DMA_SELECT_UART2_TX_REQ,
    SYSCTL_DMA_SELECT_UART3_RX_REQ,
    SYSCTL_DMA_SELECT_UART3_TX_REQ,
    SYSCTL_DMA_SELECT_AES_REQ,
    SYSCTL_DMA_SELECT_SHA_RX_REQ,
    SYSCTL_DMA_SELECT_AI_RX_REQ,
    SYSCTL_DMA_SELECT_FFT_RX_REQ,
    SYSCTL_DMA_SELECT_FFT_TX_REQ,
    SYSCTL_DMA_SELECT_I2S0_TX_REQ,
    SYSCTL_DMA_SELECT_I2S0_RX_REQ,
    SYSCTL_DMA_SELECT_I2S1_TX_REQ,
    SYSCTL_DMA_SELECT_I2S1_RX_REQ,
    SYSCTL_DMA_SELECT_I2S2_TX_REQ,
    SYSCTL_DMA_SELECT_I2S2_RX_REQ,
    SYSCTL_DMA_SELECT_MAX
} sysctl_dma_select_t;
```

20.4.7.3 Enumeration element

Element name	Description
SYSCTL_DMA_SELECT_SSI0_RX_REQ	SPI0 receive
SYSCTL_DMA_SELECT_SSI0_TX_REQ	SPI0 transmit
SYSCTL_DMA_SELECT_SSI1_RX_REQ	SPI1 receive
SYSCTL_DMA_SELECT_SSI1_TX_REQ	SPI1 transmit
SYSCTL_DMA_SELECT_SSI2_RX_REQ	SPI2 receive
SYSCTL_DMA_SELECT_SSI2_TX_REQ	SPI2 transmit
SYSCTL_DMA_SELECT_SSI3_RX_REQ	SPI3 receive
SYSCTL_DMA_SELECT_SSI3_TX_REQ	SPI3 transmit
SYSCTL_DMA_SELECT_I2C0_RX_REQ	I2C0 receive
SYSCTL_DMA_SELECT_I2C0_TX_REQ	I2C0 transmit
SYSCTL_DMA_SELECT_I2C1_RX_REQ	I2C1 receive
SYSCTL_DMA_SELECT_I2C1_TX_REQ	I2C1 transmit
SYSCTL_DMA_SELECT_I2C2_RX_REQ	I2C2 receive
SYSCTL_DMA_SELECT_I2C2_TX_REQ	I2C2 transmit
SYSCTL_DMA_SELECT_UART1_RX_REQ	UART1 receive
SYSCTL_DMA_SELECT_UART1_TX_REQ	UART1 transmit

Element name	Description
SYSCTL_DMA_SELECT_UART2_RX_REQ	UART2 receive
SYSCTL_DMA_SELECT_UART2_TX_REQ	UART2 transmit
SYSCTL_DMA_SELECT_UART3_RX_REQ	UART3 receive
SYSCTL_DMA_SELECT_UART3_TX_REQ	UART3 transmit
SYSCTL_DMA_SELECT_AES_REQ	AES
SYSCTL_DMA_SELECT_SHA_RX_REQ	SHA receive
SYSCTL_DMA_SELECT_AI_RX_REQ	AI receive
SYSCTL_DMA_SELECT_FFT_RX_REQ	FFT receive
SYSCTL_DMA_SELECT_FFT_TX_REQ	FFT transmit
SYSCTL_DMA_SELECT_I2S0_TX_REQ	I2S0 transmit
SYSCTL_DMA_SELECT_I2S0_RX_REQ	I2S0 receive
SYSCTL_DMA_SELECT_I2S1_TX_REQ	I2S1 transmit
SYSCTL_DMA_SELECT_I2S1_RX_REQ	I2S1 receive
SYSCTL_DMA_SELECT_I2S2_TX_REQ	I2S2 transmit
SYSCTL_DMA_SELECT_I2S2_RX_REQ	I2S2 receive

20.4.8 sysctl_power_bank_t

20.4.8.1 Description

Power domain number.

20.4.8.2 Type definition

```
typedef enum _sysctl_power_bank
{
    SYSCTL_POWER_BANK0,
    SYSCTL_POWER_BANK1,
    SYSCTL_POWER_BANK2,
    SYSCTL_POWER_BANK3,
    SYSCTL_POWER_BANK4,
    SYSCTL_POWER_BANK5,
    SYSCTL_POWER_BANK6,
    SYSCTL_POWER_BANK7,
    SYSCTL_POWER_BANK_MAX,
} sysctl_power_bank_t;
```

20.4.8.3 Enumeration element

Element name	Description
SYSCTL_POWER_BANK0	Power domain 0, contain IO0-IO5
SYSCTL_POWER_BANK1	Power domain 1, contain IO6-IO11
SYSCTL_POWER_BANK2	Power domain 2, contain IO12-IO17
SYSCTL_POWER_BANK3	Power domain 3, contain IO18-IO23
SYSCTL_POWER_BANK4	Power domain 4, contain IO24-IO29
SYSCTL_POWER_BANK5	Power domain 5, contain IO30-IO35
SYSCTL_POWER_BANK6	Power domain 6, contain IO36-IO41
SYSCTL_POWER_BANK7	Power domain 7, contain IO42-IO47

20.4.9 sysctl_io_power_mode_t

20.4.9.1 Description

IO power domain voltage selection.

20.4.9.2 Type definition

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

20.4.9.3 Enumeration element

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

Chapter 21

Architecture support package (BSP)

21.1 Overview

Architecture level support function of k210 SoC.

21.2 Features

Provides an interface to get the CPU core ID of the currently running program and an entry to start the second core.

21.3 API

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

- register_core1
- current_coreid

21.3.1 register_core1

21.3.1.1 Description

Register the function with core 1 and start core 1.

21.3.1.2 Function prototype

```
int register_core1(core_function func, void *ctx)
```

21.3.1.3 Parameter

Parameter name	Description	Input or output
func	Function registered to core 1	Input
ctx	The parameter of the function, set to NULL means not used	Input

21.3.1.4 Return value

Return value	Description
0	Success
Other	Fail

21.3.2 current_coreid

21.3.2.1 Description

Get the current CPU core ID.

21.3.2.2 Function prototype

```
#define current_coreid()       read_csr(mhartid)
```

21.3.2.3 Parameter

None.

21.3.2.4 Return value

The ID of the current CPU core.

21.3.3 Example

```
/* Core 0 gets the CPU core ID and prints Hello world, then starts core 1 to get the
    CPU core ID and prints Hello world */
#include <stdio.h>
#include "bsp.h"

int core1_function(void *ctx)
{
    uint64_t core = current_coreid();
```

```
    printf("Core %ld Hello world\n", core);
    while(1);
}

int main()
{
    uint64_t core = current_coreid();
    printf("Core %ld Hello world\n", core);
    register_core1(core1_function, NULL);
    while(1);
}
```

21.4 Data type

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

- core_function: The function called by the CPU core.

21.4.1 core_function

21.4.1.1 Description

The function called by the CPU core.

21.4.1.2 Type definition

```
typedef int (*core_function)(void *ctx);
```