K

Standalone SDK Programming Guide

KENDRYTE

Kan Chi

KENDRYTE.COM

About this manual

This document provides users based programming guide when Kendryte K210 Standalone SDK development.

The corresponding SDK version

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

Release Notes

date	Release notes
2018-10-10	V0.1.0 initial release
2018-10-20	V0.2.0 release corresponds to the document Standalone SDK v0.4.0 2018-11-02 V0.3.0 release of the document corresponding to the Standalone SDK v0.5.1

Disclaimer

Copyright Notice

table of Contents

The first 1 chapter

Neural network processor (KPU)

1.1 Overview

KPU is generic neural network processor, which may implement a convolutional neural network calculation in the case of low-power, real-time access is detected mesh Target size, type and the coordinates of the face or an object detection and classification. When using kpu, it must be combined model compiler.

1.2 Functional Description

KPU has the following characteristics:

- Support mainstream training framework trained in accordance with specific rules to limit fixed-point model
- No direct limit network layers, each supporting separate convolutional neural network configuration parameters, including the number of input and output channels, input and output High travel wide columns
- Convolution kernel supports two 1x1 and 3x3
- Support any form of activation function
- The maximum supported size of the neural network parameters 5.5MiB to 5.9MiB real-time work
- When the non-real-time network parameters to support the work of the largest size (Flash capacity - Software volume)

1.3 API Reference

Corresponding header file kpu.h

To provide users with 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

Kpu initialization task handle, the function embodied in gencode_output.c model compiler generated.

1.3.1.2 function definition

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

1.3.1.3 Parameters

Parameter Name	Description	input 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 KPU, were AI operations.

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 Parameters

Parameter Name	Description	input Output
task	KPU task handle	Entry
dma_ch	DMA channel	Entry
src	The input image data	Entry
dest	Output calculation	Export
callback	function input operation completion callback	

1.3.2.4 Return value

Return Value	Description
0	success
Non-0	KPU busy, failed

1.3.3 kpu_get_output_buf

1.3.3.1 Description

Gets a cached output result of the KPU.

1.3.3.2 function prototype

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

1.3.3.3 Parameters

Parameter Name	Description	input Output
task	KPU task handle	input

1.3.3.4 Return value

KPU result output buffer pointer.

1.3.4 kpu_release_output_buf

1.3.4.1 Description

KPU release the output cache.

1.3.4.2 function prototype

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

1.3.4.3 Parameters

Parameter Name	Description	input Output
The output buffer	input output_buf KPU	

1.3.4.4 Return value None

1.4 Data Types

Data types, 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 definitions

```
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 members

Member Name	Description
layers	KPU argument pointer
length	The number of layers
dma_ch	DMA channel
dst	Pointer calculation result output
dst_length	length calculation result cb
	Operation completion callback function

The first 2 chapter

Advanced Encryption Accelerator (AES)

2.1 Functional Description

K210 built-in AES (Advanced Encryption Accelerator), with respect to the AES software can greatly improve computing speed. AES accelerator supports multiple encryption / decryption mode (ECB, CBC, GCM), more KEY (128,192,256) the length of the operations.

2.2 API Reference

Corresponding header file aes.h

To provide users with the following interfaces

- aes_ecb128_hard_encrypt
- aes_ecb128_hard_decrypt
- aes_ecb192_hard_encrypt
- aes_ecb192_hard_decrypt
- aes_ecb256_hard_encrypt
- aes_ecb256_hard_decrypt
- aes_cbc128_hard_encrypt
- aes_cbc128_hard_decrypt
- aes_cbc192_hard_encrypt
- aes_cbc192_hard_decrypt
- aes_cbc256_hard_encrypt
- aes_cbc256_hard_decrypt
- aes_gcm128_hard_encrypt
- aes_gcm128_hard_decrypt
- aes_gcm192_hard_encrypt

- aes_gcm192_hard_decrypt
- aes_gcm256_hard_encrypt
- aes_gcm256_hard_decrypt
- 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

2.2.1 aes_ecb128_hard_encrypt

2.2.1.1 Description

AES-ECB-128 encryption algorithm. Cpu use input and output data transfer. ECB encrypts the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling. ECB did not use vector mode.

2.2.1.2 function prototype

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

2.2.1.3 Parameters

parameter name	description	input Output
input_key	AES-ECB-128 encryption key	Entry
input_data	Plaintext data AES-ECB-128 to be encrypted	Entry
input_len	The length of the AES-ECB-128 plaintext data to be encrypted	Entry
output_data	The result of the encryption computation AES-ECB-128 stored in this buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.1.4 Return Value None.

2.2.2 aes_ecb128_hard_decrypt

2.2.2.1 Description

AES-ECB-128 decryption operation. Cpu use input and output data transfer. ECB encrypts the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling. ECB did not use vector mode.

2.2.2.2 function prototype

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

2.2.2.3 Parameters

parameter name	description	input Output
input_key	AES-ECB-128 key decrypting	Entry
input_data	The ciphertext data AES-ECB-128 to be decrypted	Entry
input_len	The length of the AES-ECB-128 of the ciphertext data to be decrypted	Entry
output_data	Result of decryption AES-ECB-128 stored in this buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.2.4 Return Value None.

2.2.3 aes_ecb192_hard_encrypt

2.2.3.1 Description

AES-ECB-192 encryption algorithm. Cpu use input and output data transfer. ECB encrypts the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling. ECB did not use vector mode.

2.2.3.2 function prototype

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

2.2.3.3 Parameters

parameter name	description	input Output
input_key	AES-ECB-192 encryption key	Entry
input_data	Plaintext data AES-ECB-192 to be encrypted	Entry
input_len	The length of the AES-ECB-192 plaintext data to be encrypted	Entry
output_data	The result of the encryption computation AES-ECB-192 stored in this buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.3.4 Return Value None.

2.2.4 aes_ecb192_hard_decrypt

2.2.4.1 Description

AES-ECB-192 decryption operation. Cpu use input and output data transfer. ECB encrypts the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling. ECB did not use vector mode.

2.2.4.2 function prototype

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

2.2.4.3 Parameters

parameter name	description	input Output
input_key	Key AES-ECB-192 decryption	Entry
input_data	AES-ECB-192 the ciphertext data to be decrypted	Entry
input_len	The length of the AES-ECB-192 of the ciphertext data to be decrypted	Entry
output_data	Result of decryption AES-ECB-192 stored in this buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.4.4 Return Value None.

2.2.5 aes_ecb256_hard_encrypt

2.2.5.1 Description

AES-ECB-256 encryption algorithm. Cpu use input and output data transfer. ECB encrypts the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling. ECB did not use vector mode.

2.2.5.2 function prototype

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

2.2.5.3 Parameters

parameter name	description	input Output
input_key	AES-ECB-256 encryption key	Entry
input_data	Plaintext data AES-ECB-256 to be encrypted	Entry
input_len	The length of the AES-ECB-256 plaintext data to be encrypted	Entry

parameter name	description	input Output
output_data	The result of the encryption computation AES-ECB-256 stored in this buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.5.4 Return Value None.

2.2.6 aes_ecb256_hard_decrypt

2.2.6.1 Description

AES-ECB-256 decryption operations. Cpu use input and output data transfer. ECB encrypts the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling. ECB did not use vector mode.

2.2.6.2 function prototype

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

2.2.6.3 Parameters

parameter name	description	input Output
input_key	Key AES-ECB-256 decryption	Entry
input_data	The ciphertext data AES-ECB-256 to be decrypted	Entry
input_len	The length of the AES-ECB-256 of the ciphertext data to be decrypted	Entry
output_data	The result of decryption AES-ECB-256 is stored in the buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.6.4 Return Value None.

2.2.7 aes_cbc128_hard_encrypt

2.2.7.1 Description

AES-CBC-128 encryption algorithm. Cpu use input and output data transfer. CBC encrypting plaintext block in a fixed size 16bytes

Encrypted, the block size is less than the filling.

2.2.7.2 function prototype

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

2.2.7.3 Parameters

parameter name	description	input Output
context	AES-CBC-128 encryption calculation structure, comprising the encryption key and the offset vector	Entry
input_data	Plaintext data AES-CBC-128 to be encrypted	Entry
input_len	The length of the AES-CBC-128 is plaintext data to be encrypted	Entry
output_data	The result of the encryption computation AES-CBC-128 is stored in the buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.7.4 Return Value None.

2.2.8 aes_cbc128_hard_decrypt

2.2.8.1 Description

AES-CBC-128 decryption operation. Cpu use input and output data transfer. CBC encrypting plaintext block in a fixed size 16bytes

Encrypted, the block size is less than the filling.

2.2.8.2 function prototype

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

2.2.8.3 Parameters

parameter name	description	input Output
context	AES-CBC-128 decrypts the structure calculation, the offset comprises a decryption key vector	Entry
input_data	The ciphertext data AES-CBC-128 to be decrypted	Entry
input_len	The length of the AES-CBC-128 of the ciphertext data to be decrypted	Entry
output_data	Result of decryption AES-CBC-128 is stored in the buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.8.4 Return Value None.

2.2.9 aes_cbc192_hard_encrypt

2.2.9.1 Description

AES-CBC-192 encryption algorithm. Cpu use input and output data transfer. CBC encrypting plaintext block in a fixed size 16bytes

Encrypted, the block size is less than the filling.

2.2.9.2 function prototype

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

2.2.9.3 Parameters

parameter name	description	input Output
context	AES-CBC-192 encryption calculation structure comprising the encryption key and the offset vector	Entry
input_data	Plaintext data AES-CBC-192 to be encrypted	Entry
input_len	The length of the AES-CBC-192 is plaintext data to be encrypted	Entry
output_data	The result of the encryption computation AES-CBC-192 is stored in the buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.9.4 Return Value None.

2.2.10 aes_cbc192_hard_decrypt

2.2.10.1 Description

AES-CBC-192 decryption operation. Cpu use input and output data transfer. CBC encrypting plaintext block in a fixed size 16bytes Encrypted, the block size is less than the filling.

2.2.10.2 function prototype

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

2.2.10.3 Parameters

parameter name	description	input Output
context	AES-CBC-192 decrypts the structure calculation, the offset comprises a decryption key vector	Entry
input_data	The ciphertext data AES-CBC-192 to be decrypted	Entry
input_len	The length of the AES-CBC-192 of the ciphertext data to be decrypted	Entry

parameter name	description	input Output
output_data	Result of decryption AES-CBC-192 is stored in the buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.10.4 Return Value None.

2.2.11 aes_cbc256_hard_encrypt

2.2.11.1 Description

AES-CBC-256 encryption algorithm. Cpu use input and output data transfer. CBC encrypting plaintext block in a fixed size 16bytes Encrypted, the block size is less than the filling.

2.2.11.2 function prototype

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

2.2.11.3 Parameters

parameter name	description	input Output
context	AES-CBC-256 encryption calculation structure comprising the encryption key and the offset vector	Entry
input_data	Plaintext data AES-CBC-256 to be encrypted	Entry
input_len	The length of the AES-CBC-256 is plaintext data to be encrypted	Entry
output_data	The result of the encryption computation AES-CBC-256 is stored in the buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.11.4 Return Value None.

2.2.12 aes_cbc256_hard_decrypt

2.2.12.1 Description

AES-CBC-256 decryption operation. Cpu use input and output data transfer. CBC encrypting plaintext block in a fixed size 16bytes

Encrypted, the block size is less than the filling.

2.2.12.2 function prototype

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

2.2.12.3 Parameters

parameter name	description	input Output
context	AES-CBC-256 decrypts the structure calculation, the offset comprises a decryption key vector	Entry
input_data	The ciphertext data AES-CBC-256 to be decrypted	Entry
input_len	The length of the AES-CBC-256 of the ciphertext data to be decrypted	Entry
output_data	Result of decryption AES-CBC-256 is stored in the buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.12.4 Return Value None.

2.2.13 aes_gcm128_hard_encrypt

2.2.13.1 Description

AES-GCM-128 encryption algorithm. Cpu use input and output data transfer.

2.2.13.2 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)
```

2.2.13.3 Parameters

Parameter Name	Description	input Output
context	Structure AES-GCM-128 encryption calculation, comprising an encryption key / an offset vector / aad / aad length	Entry
Plaintext data input_data	AES-GCM-128 to be encrypted	Entry
input_len	The length of the AES-GCM-128 plaintext data to be encrypted	Entry
Results after output_data	AES-GCM-128 encryption algorithm stored in this buffer	Export
gcm_tag	tag after AES-GCM-128 encryption algorithm stored in this buffer. The buffer size is 16bytes need to ensure output	

2.2.13.4 Return Value None.

2.2.14 aes_gcm128_hard_decrypt

2.2.14.1 Description

AES-GCM-128 decryption operation. Cpu use input and output data transfer.

2.2.14.2 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)
```

2.2.14.3 Parameters

Parameter Name	Description	input Output
context	Structure AES-GCM-128 decryption calculation, comprising a decryption key / offset vector / aad / aad length	Entry
input_data	AES-GCM-128 the ciphertext data to be decrypted	Entry
input_len	The length of the AES-GCM-128 of the ciphertext data to be decrypted	Entry
Results after output_data	AES-GCM-128 decryption stored in this buffer	Export
gcm_tag	AES-GCM-128 tag after decryption stored in this buffer. The buffer size is 16bytes need to ensure output	

2.2.14.4 Return Value None.

2.2.15 aes_gcm192_hard_encrypt

2.2.15.1 Description

AES-GCM-192 encryption algorithm. Cpu use input and output data transfer.

2.2.15.2 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)
```

2.2.15.3 Parameters

Parameter Name	Description	input Output
context	Structure AES-GCM-192 encryption calculation, comprising an encryption key / an offset vector / aad / aad length	Entry
input_data	Plaintext data AES-GCM-192 to be encrypted	Entry
input_len	The length of the AES-GCM-192 plaintext data to be encrypted	Entry
output_data	Results after AES-GCM-192 encryption algorithm stored in this buffer	Export
gcm_tag	AES-GCM-192 tag after encryption algorithms stored in this buffer. The buffer size is 16bytes need to ensure output	

2.2.15.4 Return Value None.

2.2.16 aes_gcm192_hard_decrypt

2.2.16.1 Description

AES-GCM-192 decryption operation. Cpu use input and output data transfer.

2.2.16.2 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)
```

2.2.16.3 Parameters

Parameter Name	Description	input Output
context	Structure AES-GCM-192 decryption calculation, comprising a decryption key / offset vector / aad / aad length	Entry
input_data	AES-GCM-192 the ciphertext data to be decrypted	Entry
input_len	The length of the AES-GCM-192 of the ciphertext data to be decrypted	Entry

Parameter Name	Description	input Output
	Results after output_data AES-GCM-192 decryption stored in this buffer	Export
gcm_tag	AES-GCM-192 tag after decryption stored in this buffer. The buffer size is 16bytes need to ensure output	

2.2.16.4 Return Value None.

2.2.17 aes_gcm256_hard_encrypt

2.2.17.1 Description

AES-GCM-256 encryption algorithm. Cpu use input and output data transfer.

2.2.17.2 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)
```

2.2.17.3 Parameters

Parameter Name	Description	input Output
context	Structure AES-GCM-256 encryption calculation, comprising an encryption key / an offset vector / aad / aad length	Entry
	Plaintext data input_data AES-GCM-256 to be encrypted	Entry
input_len	The length of the AES-GCM-256 plaintext data to be encrypted	Entry
	Results after output_data AES-GCM-256 encryption algorithm stored in this buffer	Export
gcm_tag	AES-GCM-256 tag after encryption algorithms stored in this buffer. The buffer size is 16bytes need to ensure output	

2.2.17.4 Return Value None.

2.2.18 aes_gcm256_hard_decrypt

2.2.18.1 Description

AES-GCM-256 decryption operations. Cpu use input and output data transfer.

2.2.18.2 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)
```

2.2.18.3 Parameters

Parameter Name	Description	input Output
context	Structure AES-GCM-256 decryption calculation, comprising a decryption key / offset vector / aad / aad length	Entry
The ciphertext data input_data AES-GCM-256 to be decrypted		Entry
input_len	The length of the AES-GCM-256 of the ciphertext data to be decrypted	Entry
Results after output_data AES-GCM-256 decryption stored in this buffer		Export
gcm_tag	AES-GCM-256 tag after decryption is stored in the buffer. The buffer size is 16bytes need to ensure output	

2.2.18.4 Return Value None.

2.2.19 aes_ecb128_hard_encrypt_dma

2.2.19.1 Description

AES-ECB-128 encryption algorithm. Cpu input data using the transmission, output data transmission dma use. ECB encrypts the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling. ECB did not use vector mode.

2.2.19.2 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)
```

2.2.19.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
input_key	AES-ECB-128 encryption key	Entry
input_data	Plaintext data AES-ECB-128 to be encrypted	Entry
input_len	The length of the AES-ECB-128 plaintext data to be encrypted	Entry
output_data	The result of the encryption computation AES-ECB-128 stored in this buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.19.4 Return Value None.

2.2.20 aes_ecb128_hard_decrypt_dma

2.2.20.1 Description

AES-ECB-128 decryption operation. Cpu input data using the transmission, output data transmission dma use. ECB encrypts the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling. ECB did not use vector mode.

2.2.20.2 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)
```

2.2.20.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
input_key	AES-ECB-128 key decrypting	Entry
input_data	The ciphertext data AES-ECB-128 to be decrypted	Entry
input_len	The length of the AES-ECB-128 of the ciphertext data to be decrypted	Entry
output_data	Result of decryption AES-ECB-128 stored in this buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.20.4 Return Value None.

2.2.21 aes_ecb192_hard_encrypt_dma

2.2.21.1 Description

AES-ECB-192 encryption algorithm. Cpu input data using the transmission, output data transmission dma use. ECB encrypts the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling. ECB did not use vector mode.

2.2.21.2 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)
```

2.2.21.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
input_key	AES-ECB-192 encryption key	Entry
input_data	Plaintext data AES-ECB-192 to be encrypted	Entry
input_len	The length of the AES-ECB-192 plaintext data to be encrypted	Entry
output_data	The result of the encryption computation AES-ECB-192 stored in this buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.21.4 Return Value None.

2.2.22 aes_ecb192_hard_decrypt_dma

2.2.22.1 Description

AES-ECB-192 decryption operation. Cpu input data using the transmission, output data transmission dma use. ECB encrypts the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling. ECB did not use vector mode.

2.2.22.2 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)
```

2.2.22.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
input_key	Key AES-ECB-192 decryption	Entry

parameter name	description	input Output
input_data	AES-ECB-192 the ciphertext data to be decrypted	Entry
input_len	The length of the AES-ECB-192 of the ciphertext data to be decrypted	Entry
output_data	Result of decryption AES-ECB-192 stored in this buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.22.4 Return Value None.

2.2.23 aes_ecb256_hard_encrypt_dma

2.2.23.1 Description

AES-ECB-256 encryption algorithm. Cpu input data using the transmission, output data transmission dma use. ECB encrypts the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling. ECB did not use vector mode.

2.2.23.2 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)
```

2.2.23.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
input_key	AES-ECB-256 encryption key	Entry
input_data	Plaintext data AES-ECB-256 to be encrypted	Entry
input_len	The length of the AES-ECB-256 plaintext data to be encrypted	Entry
output_data	The result of the encryption computation AES-ECB-256 stored in this buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.23.4 Return Value None.

2.2.24 aes_ecb256_hard_decrypt_dma

2.2.24.1 Description

AES-ECB-256 decryption operations. Cpu input data using the transmission, output data transmission dma use. ECB encrypts the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling. ECB did not use vector mode.

2.2.24.2 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)
```

2.2.24.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
input_key	Key AES-ECB-256 decryption	Entry
input_data	The ciphertext data AES-ECB-256 to be decrypted	Entry
input_len	The length of the AES-ECB-256 of the ciphertext data to be decrypted	Entry
output_data	The result of decryption AES-ECB-256 is stored in the buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.24.4 Return Value None.

2.2.25 aes_cbc128_hard_encrypt_dma

2.2.25.1 Description

AES-CBC-128 encryption algorithm. Cpu input data using the transmission, output data transmission dma use. CBC encrypting the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling.

2.2.25.2 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)
```

2.2.25.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
context	AES-CBC-128 encryption calculation structure, comprising the encryption key and the offset vector	Entry
input_data	Plaintext data AES-CBC-128 to be encrypted	Entry
input_len	The length of the AES-CBC-128 is plaintext data to be encrypted	Entry
output_data	The result of the encryption computation AES-CBC-128 is stored in the buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.25.4 Return Value None.

2.2.26 aes_cbc128_hard_decrypt_dma

2.2.26.1 Description

AES-CBC-128 decryption operation. Cpu input data using the transmission, output data transmission dma use. CBC encrypting the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling.

2.2.26.2 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)
```

2.2.26.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
context	AES-CBC-128 decrypts the structure calculation, the offset comprises a decryption key vector	Entry
input_data	The ciphertext data AES-CBC-128 to be decrypted	Entry
input_len	The length of the AES-CBC-128 of the ciphertext data to be decrypted	Entry
output_data	Result of decryption AES-CBC-128 is stored in the buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.26.4 Return Value None.

2.2.27 aes_cbc192_hard_encrypt_dma

2.2.27.1 Description

AES-CBC-192 encryption algorithm. Cpu input data using the transmission, output data transmission dma use. CBC encrypting the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling.

2.2.27.2 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)
```

2.2.27.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
context	AES-CBC-192 encryption calculation structure comprising the encryption key and the offset vector	Entry
input_data	Plaintext data AES-CBC-192 to be encrypted	Entry

parameter name	description	input Output
input_len	The length of the AES-CBC-192 is plaintext data to be encrypted	Entry
output_data	The result of the encryption computation AES-CBC-192 is stored in the buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.27.4 Return Value None.

2.2.28 aes_cbc192_hard_decrypt_dma

2.2.28.1 Description

AES-CBC-192 decryption operation. Cpu input data using the transmission, output data transmission dma use. CBC encrypting the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling.

2.2.28.2 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)
```

2.2.28.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
context	AES-CBC-192 decrypts the structure calculation, the offset comprises a decryption key vector	Entry
input_data	The ciphertext data AES-CBC-192 to be decrypted	Entry
input_len	The length of the AES-CBC-192 of the ciphertext data to be decrypted	Entry

parameter name	description	input Output
output_data	Result of decryption AES-CBC-192 is stored in the buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.28.4 Return Value None.

2.2.29 aes_cbc256_hard_encrypt_dma

2.2.29.1 Description

AES-CBC-256 encryption algorithm. Cpu input data using the transmission, output data transmission dma use. CBC encrypting the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling.

2.2.29.2 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)
```

2.2.29.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
context	AES-CBC-256 encryption calculation structure comprising the encryption key and the offset vector	Entry
input_data	Plaintext data AES-CBC-256 to be encrypted	Entry
input_len	The length of the AES-CBC-256 is plaintext data to be encrypted	Entry
output_data	The result of the encryption computation AES-CBC-256 is stored in the buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.29.4 Return Value None.

2.2.30 aes_cbc256_hard_decrypt_dma

2.2.30.1 Description

AES-CBC-256 decryption operation. Cpu input data using the transmission, output data transmission dma use. CBC encrypting the plaintext to encrypt fixed size 16bytes per block, the block size is less than the filling.

2.2.30.2 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)
```

2.2.30.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
context	AES-CBC-256 decrypts the structure calculation, the offset comprises a decryption key vector	Entry
input_data	The ciphertext data AES-CBC-256 to be decrypted	Entry
input_len	The length of the AES-CBC-256 of the ciphertext data to be decrypted	Entry
output_data	Result of decryption AES-CBC-256 is stored in the buffer. The buffer size needs to ensure 16bytes alignment.	Export

2.2.30.4 Return Value None.

2.2.31 aes_gcm128_hard_encrypt_dma

2.2.31.1 Description

AES-GCM-128 encryption algorithm. Cpu input data using the transmission, output data transmission dma use.

2.2.31.2 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)
```

2.2.31.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
context	Structure AES-GCM-128 encryption calculation, comprising an encryption key / an offset vector / aad / aad length	Entry
input_data	Plaintext data AES-GCM-128 to be encrypted	Entry
input_len	AES-GCM-128 the length of plaintext data to be encrypted.	Entry
output_data	Result of AES-GCM-128 encryption algorithm stored in this buffer. . Since the minimum particle size of 4bytes DMA data transfer, it is necessary to ensure that the buffer size is at least the whole of 4bytes Several times.	Export
gcm_tag	tag after AES-GCM-128 encryption algorithm stored in this buffer. The buffer size needs to guarantee 16bytes	Export

2.2.31.4 Return Value None.

2.2.32 aes_gcm128_hard_decrypt_dma

2.2.32.1 Description

AES-GCM-128 decryption operation. Cpu input data using the transmission, output data transmission dma use.

2.2.32.2 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)
```

2.2.32.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
context	Structure AES-GCM-128 decryption calculation, comprising a decryption key / offset vector / aad / aad length	Entry
input_data	AES-GCM-128 the ciphertext data to be decrypted	Entry
input_len	AES-GCM-128 to be the length of the ciphertext data decrypted.	Entry
output_data	Result of decryption AES-GCM-128 stored in this buffer. Since the minimum particle size of 4bytes DMA data transfer, it is necessary to ensure that the buffer size is at least the whole of 4bytes Several times.	Export
gcm_tag	AES-GCM-128 tag after decryption stored in this buffer. The buffer size needs to guarantee 16bytes	Export

2.2.32.4 Return Value None.

2.2.33 aes_gcm192_hard_encrypt_dma

2.2.33.1 Description

AES-GCM-192 encryption algorithm. Cpu input data using the transmission, output data transmission dma use.

2.2.33.2 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)
```

2.2.33.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
context	Structure AES-GCM-192 encryption calculation, comprising an encryption key / an offset vector / aad / aad length	Entry
input_data	Plaintext data AES-GCM-192 to be encrypted	Entry
input_len	AES-GCM-192 the length of plaintext data to be encrypted.	Entry
output_data	The result of the encryption computation AES-GCM-192 stored in this buffer. Since the minimum particle size of 4bytes DMA data transfer, it is necessary to ensure that the buffer size is at least the whole of 4bytes Several times.	Export
gcm_tag	AES-GCM-192 tag after encryption algorithms stored in this buffer. The buffer size needs to guarantee 16bytes	Export

2.2.33.4 Return Value None.

2.2.34 aes_gcm192_hard_decrypt_dma

2.2.34.1 Description

AES-GCM-192 decryption operation. Cpu input data using the transmission, output data transmission dma use.

2.2.34.2 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)
```

2.2.34.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
context	Structure AES-GCM-192 decryption calculation, comprising a decryption key / offset vector / aad / aad length	Entry
input_data	AES-GCM-192 the ciphertext data to be decrypted	Entry
input_len	The length of the AES-GCM-192 of the ciphertext data to be decrypted.	Entry
output_data	Result of decryption AES-GCM-192 stored in this buffer. Since the minimum particle size of 4bytes DMA data transfer, it is necessary to ensure that the buffer size is at least the whole of 4bytes Several times.	Export
gcm_tag	AES-GCM-192 tag after decryption stored in this buffer. The buffer size needs to guarantee 16bytes	Export

2.2.34.4 Return Value None.

2.2.35 aes_gcm256_hard_encrypt_dma

2.2.35.1 Description

AES-GCM-256 encryption algorithm. Cpu input data using the transmission, output data transmission dma use.

2.2.35.2 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)
```

2.2.35.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
context	Structure AES-GCM-256 encryption calculation, comprising an encryption key / an offset vector / aad / aad length	Entry
input_data	Plaintext data AES-GCM-256 to be encrypted	Entry
input_len	AES-GCM-256 the length of plaintext data to be encrypted.	Entry
output_data	The result of the encryption computation AES-GCM-256 stored in this buffer. Since the minimum particle size of 4bytes DMA data transfer, it is necessary to ensure that the buffer size is at least the whole of 4bytes Several times.	Export
gcm_tag	AES-GCM-256 tag after encryption algorithms stored in this buffer. The buffer size needs to guarantee 16bytes	Export

2.2.35.4 Return Value None.

2.2.36 aes_gcm256_hard_decrypt_dma

2.2.36.1 Description

AES-GCM-256 decryption operations. Cpu input data using the transmission, output data transmission dma use.

2.2.36.2 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)
```

2.2.36.3 Parameters

parameter name	description	input Output
dma_receive_channel_num	AES outputs DMA channel number data	Entry
context	Structure AES-GCM-256 decryption calculation, comprising a decryption key / offset vector / aad / aad length	Entry
input_data	The ciphertext data AES-GCM-256 to be decrypted	Entry
input_len	The length of the AES-GCM-256 of the ciphertext data to be decrypted.	Entry
output_data	The result of decryption AES-GCM-256 stored in the buffer. Since the minimum particle size of 4bytes DMA data transfer, it is necessary to ensure that the buffer size is at least the whole of 4bytes Several times.	Export
gcm_tag	AES-GCM-256 tag after decryption is stored in the buffer. The buffer size needs to guarantee 16bytes	Export

2.2.36.4 Return Value None.

2.2.37 aes_init

2.2.37.1 described AES hardware module

initialization

2.2.37.2 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)
```

2.2.37.3 Parameters

parameter name	description	input Output
input_key	Be the encryption key / decryption	Entry
Length input_key_len be encryption / decryption key		Entry
iv	AES encryption and decryption of data used iv	Entry
iv_len	Iv AES encryption and decryption data used length, CBC fixed 16bytes, GCM is fixed to an output 12bytes	
gcm_aad	AES-GCM encryption and decryption data used aad	Export
cipher_mode	Type AES hardware encryption and decryption module executed, support AES_CBC / AES_ECB / AES_GCM	Entry
encrypt_sel	Mode AES hardware module executed: encryption or decryption	Entry
gcm_aad_len	The length of the AES-GCM encryption and decryption data used aad	Entry
Input_data_len length data to be encrypted / decrypted		Entry

2.2.37.4 Return Value None.

2.2.38 aes_process

2.2.38.1 Description

AES hardware modules perform cryptographic operations

2.2.38.2 function prototype

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

2.2.38.3 Parameters

parameter name	description	input Output
input_data	The buffer storage to be encrypted / decrypted data	Entry
The buffer storage output_data encryption / decryption output		Export
Input_data_len length to be encrypted / decrypted data		Entry

parameter name	description	input Output
cipher_mode	Type AES encryption and decryption performed by hardware modules, support AES_CBC / AES_ECB / AES_GCM	input

2.2.38.4 Return Value None.

2.2.39 gcm_get_tag

2.2.39.1 Description

The tag after the end AES-GCM computing

2.2.39.2 function prototype

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

2.2.39.3 Parameters

Parameter Name	Description	input Output
	The buffer store gcm_tag AES-GCM encryption / tag decrypted, 16bytes fixed to the size of the input	

2.2.39.4 Return Value None.

For example 2.2.40

```
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);
```

2.3 Data Type

Data types, data structures are defined as follows:

- aes_cipher_mode_t: AES encryption / decryption mode.

2.3.1 aes_cipher_mode_t

2.3.1.1 Description

AES encryption / decryption mode.

2.3.1.2 definitions

```
typedef enum _ aes_cipher_mode {

    AES_ECB = 0, AES_CBC = 1,
AES_GCM = 2,
AES_CIPHER_MAX}
aes_cipher_mode_t;
```

- gcm_context_t: structure when AES-GCM encryption / decryption parameters used

2.3.2 gcm_context_t

2.3.2.1 Description

AES-GCM structure parameters used, comprising a key, an offset vector, AAD data, the data length of AAD.

2.3.2.2 definitions

```
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: structure when AES-CBC encryption / decryption parameters used

2.3.3 cbc_context_t

2.3.3.1 Description

AES-CBC parameters used in the structure, including key offset vector.

2.3.3.2 definitions

```
typedef struct _ cbc_context {
```

```
    uint8_t * input_key; uint8_t * iv;}
cbc_context_t;
```

2.3.3.3 members

Member Name	Description
AES_ECB	ECB encryption / decryption
AES_CBC	CBC encryption / decryption
AES_GCM	GCM encryption / decryption

The first 3 chapter

Interrupt PLIC

3.1 Overview

It may be any of a single external interrupt source assigned to each CPU of the external interrupt. This provides great flexibility, can adapt to different Application requirements.

3.2 Functional Description

PLIC module has the following features:

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

3.3 API Reference

The corresponding header file plic.h

To provide users with 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

PLIC initialize external interrupt.

3.3.1.2 function prototype

```
void plic_init ( void)
```

3.3.1.3 Parameters

None.

3.3.1.4 Return Value None.

3.3.2 plic_irq_enable

3.3.2.1 Description Enable

external interrupt.

3.3.2.2 function prototype

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

3.3.2.3 Parameters

Parameter Name	Description	Input Output
irq_number	interrupt number	input

3.3.2.4 Return value

Return Value	Description
0	success
Non-0	failure

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 Parameters

Parameter Name	Description	Input Output
irq_number	interrupt number	input

3.3.3.4 Return value

Return Value	Description
0	success
Non-0	failure

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 Parameters

Parameter Name	Description	input Output
irq_number	interrupt number	Entry
priority	Interrupt priority	input

3.3.4.4 Return value

Return Value	Description
0	success
Non-0	failure

3.3.5 plic_get_priority

3.3.5.1 Description Get

interrupt priority.

3.3.5.2 function prototype

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

3.3.5.3 Parameters

Parameter Name	Description	Input Output
irq_number	interrupt number	input

3.3.5.4 Return value

irq_number interrupt priority.

3.3.6 plic_irq_register

3.3.6.1 Description Registering

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 Parameters

Parameter Name	Description	input Output
irq	Interrupt number	Entry
interrupt callback function callback		Entry
ctx	Input parameters callback function	

3.3.6.4 Return value

Return Value	Description
0	success
Non-0	failure

3.3.7 plic_irq_deregister

3.3.7.1 Description cancellation of

external interrupt function.

3.3.7.2 function prototype

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

3.3.7.3 Parameters

Parameter Name	Description	Input	Output
irq	IRQ input		

3.3.7.4 Return value

Return Value	Description
0	success
Non-0	failure

3.3.8 Examples

```
/ * Set GPIOHS0 triggers an interrupt * /
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 Types

Data types, 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 definitions

```
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 members

Member name	description
IRQN_NO_INTERRUPT	does not exist
IRQN_SPI0_INTERRUPT	SPI0 interrupt
IRQN_SPI1_INTERRUPT	SPI1 break
IRQN_SPI_SLAVE_INTERRUPT from SPI interrupt	
IRQN_SPI3_INTERRUPT	SPI3 break
IRQN_I2S0_INTERRUPT	I2S0 break
IRQN_I2S1_INTERRUPT	I2S1 break

Member name	description
IRQN_I2S2_INTERRUPT	I2S2 break
IRQN_I2C0_INTERRUPT	I2C0 break
IRQN_I2C1_INTERRUPT	I2C1 break
IRQN_I2C2_INTERRUPT	I2C2 break
IRQN_UART1_INTERRUPT	UART1 interrupt
IRQN_UART2_INTERRUPT	UART2 break
IRQN_UART3_INTERRUPT	UART3 break
IRQN_TIMER0A_INTERRUPT	TIMER0 interrupt channels 0 and 1
IRQN_TIMER0B_INTERRUPT	TIMER0 interrupt channels 2 and 3
IRQN_TIMER1A_INTERRUPT	TIMER1 interrupt channels 0 and 1
IRQN_TIMER1B_INTERRUPT	TIMER1 interrupt channels 2 and 3
IRQN_TIMER2A_INTERRUPT	TIMER2 interrupt channels 0 and 1
IRQN_TIMER2B_INTERRUPT	TIMER2 channel 2 and 3 interrupt
IRQN_RTC_INTERRUPT	RTC tick interrupts and alarm interrupts
IRQN_WDT0_INTERRUPT	Watchdog Interrupt 0
IRQN_WDT1_INTERRUPT	Watchdog interrupt 1
IRQN_APB_GPIO_INTERRUPT ordinary GPIO interrupt	
IRQN_DVP_INTERRUPT	Digital camera (DVP) interrupts
IRQN_AI_INTERRUPT	AI interrupt accelerator
IRQN_FFT_INTERRUPTFFT	Fourier interrupt accelerator
IRQN_DMA0_INTERRUPT	DMA channel 0 interrupt
IRQN_DMA1_INTERRUPT	DMA channel 1 interrupt
IRQN_DMA2_INTERRUPT	DMA Channel 2 Interrupt
IRQN_DMA3_INTERRUPT	DMA Channel 3 Interrupt
IRQN_DMA4_INTERRUPT	DMA Channel 4 Interrupt
IRQN_DMA5_INTERRUPT	DMA Channel 5 Interrupt
IRQN_UARTHS_INTERRUPT	High-speed UART interrupt
IRQN_GPIOHS0_INTERRUPT	High-speed interrupt GPIO0
IRQN_GPIOHS1_INTERRUPT	High-speed interrupt GPIO1
IRQN_GPIOHS2_INTERRUPT	High-speed interrupt GPIO2
IRQN_GPIOHS3_INTERRUPT	High-speed interrupt GPIO3
IRQN_GPIOHS4_INTERRUPT	High-speed interrupt GPIO4
IRQN_GPIOHS5_INTERRUPT	High-speed interrupt GPIO5
IRQN_GPIOHS6_INTERRUPT	High-speed interrupt GPIO6
IRQN_GPIOHS7_INTERRUPT	High-speed interrupt GPIO7

Member name	description
IRQN_GPIOHS8_INTERRUPT	High-speed interrupt GPIO8
IRQN_GPIOHS9_INTERRUPT	High-speed interrupt GPIO9
IRQN_GPIOHS10_INTERRUPT	high-speed GPIO10 interrupt
IRQN_GPIOHS11_INTERRUPT	high-speed GPIO11 interrupt
IRQN_GPIOHS12_INTERRUPT	high-speed GPIO12 interrupt
IRQN_GPIOHS13_INTERRUPT	high-speed GPIO13 interrupt
IRQN_GPIOHS14_INTERRUPT	high-speed GPIO14 interrupt
IRQN_GPIOHS15_INTERRUPT	high-speed GPIO15 interrupt
IRQN_GPIOHS16_INTERRUPT	high-speed GPIO16 interrupt
IRQN_GPIOHS17_INTERRUPT	high-speed GPIO17 interrupt
IRQN_GPIOHS18_INTERRUPT	high-speed GPIO18 interrupt
IRQN_GPIOHS19_INTERRUPT	high-speed GPIO19 interrupt
IRQN_GPIOHS20_INTERRUPT	high-speed GPIO20 interrupt
IRQN_GPIOHS21_INTERRUPT	high-speed GPIO21 interrupt
IRQN_GPIOHS22_INTERRUPT	high speed GPIO22 interrupt
IRQN_GPIOHS23_INTERRUPT	high-speed GPIO23 interrupt
IRQN_GPIOHS24_INTERRUPT	high-speed high-speed GPIO25 GPIO24
interrupt IRQN_GPIOHS25_INTERRUPT	interrupt
IRQN_GPIOHS26_INTERRUPT	high-speed GPIO26 interrupt
IRQN_GPIOHS27_INTERRUPT	high-speed GPIO27 interrupt
IRQN_GPIOHS28_INTERRUPT	high-speed GPIO28 interrupt
IRQN_GPIOHS29_INTERRUPT	high-speed GPIO29 interrupt
IRQN_GPIOHS30_INTERRUPT	high-speed GPIO30 interrupt
IRQN_GPIOHS31_INTERRUPT	high-speed interrupt GPIO31

3.4.2 plic_irq_callback_t

3.4.2.1 Description external interrupt

callback function.

3.4.2.2 definitions

```
typedef int ( * Plic_irq_callback_t) ( void * Ctx);
```

The first 4 chapter

General Purpose Input / Output (GPIO)

4.1 Overview

There are eight general-purpose chip GPIO.

4.2 Functional Description

GPIO module has the following features:

- With configurable pull-down driving mode

4.3 API Reference

Corresponding header file gpio.h

To provide users with 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
Non-0	failure

4.3.2 gpio_set_drive_mode

4.3.2.1 Set GPIO drive mode is

described.

4.3.2.2 function prototype

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

4.3.2.3 Parameters

Parameter Name	Description	input Output
pin	GPIO pins	Entry
mode	GPIO drive mode input	

4.3.2.4 Return Value None.

4.3.3 gpio_set_pin

4.3.3.1 Set the value GPIO

pin.

4.3.3.2 function prototype

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

4.3.3.3 Parameters

Parameter Name	Description	input Output
pin	GPIO input pin	
value	GPIO input value	

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 Parameters

Parameter Name	Description	input Output
pin	GPIO input pin	

4.3.4.4 GPIO pin return value

acquired.

4.3.5 Examples

```
/ * Set to IO13 and the output is made 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 Types

Data types, 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 GPIO drive mode is

described.

4.4.1.2 definitions

```
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 members

Member name	description
GPIO_DM_INPUT	Entry
GPIO_DM_INPUT_PULL_DOWN	input pulldown
GPIO_DM_INPUT_PULL_UP	Input pull
GPIO_DM_OUTPUT	Export

4.4.2 gpio_pin_value_t

4.4.2.1 Description

GPIO value.

4.4.2.2 definitions

```
typedef enum _ gpio_pin_value {

    GPIO_PV_LOW,
  GPIO_PV_HIGH}
  gpio_pin_value_t;
```

4.4.2.3 members

Member name	description
GPIO_PV_LOW	low
GPIO_PV_HIGH	high

The first 5 chapter

High-speed general-purpose input / output (GPIOHS)

5.1 Overview

Chip has 32 high speed GPIO. Ordinary GPIO similar pin reversal stronger.

5.2 Functional Description

GPIOHS module has the following features:

- With configurable pull-down driving mode
- Support rising, falling, and double-edge trigger

5.3 API Reference

Corresponding header file gpiohs.h

To provide users with 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 Set GPIO drive mode is

described.

5.3.1.2 function prototype

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

5.3.1.3 Parameters

Parameter Name	Description	input Output
pin	GPIO pins	Entry
mode	GPIO drive mode input	

5.3.1.4 Return Value None.

5.3.2 gpio_set_pin

5.3.2.1 Set the value GPIO

pin.

5.3.2.2 function prototype

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

5.3.2.3 Parameters

Parameter Name	Description	input Output
pin	GPIO input pin	
value	GPIO input value	

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 Parameters

Parameter Name	Description	input Output
pin	GPIO input pin	

5.3.3.4 GPIO pin return value

acquired.

5.3.4 gpiohs_set_pin_edge

5.3.4.1 Description

Setting 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 Parameters

Parameter Name	Description	input Output
pin	GPIO input pin	
edge	Interrupt trigger input	

5.3.4.4 Return Value None.

5.3.5 gpiohs_set_irq

5.3.5.1 Description

Setting high-speed GPIO interrupt callback function.

5.3.5.2 function prototype

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

5.3.5.3 Parameters

Parameter Name	Description	input Output
pin	GPIO input pin	
priority	interrupt priority	input
func	Interrupt callback function	input

5.3.5.4 Return Value None.

5.3.6 Examples

```
void irq_gpiohs2 ( void)
{
        printf ( "Hello _ world \ n ");}

/ * Set IO13 high-speed GPIO, and the output mode is set to high * /
fpioa_set_function (13, FUNC_GPIOHS3); gpiohs_set_drive_mode (3,
GPIO_DM_OUTPUT); gpiohs_set_pin (3, GPIO_PV_High);

/ * Print Hello world setting IO14 interrupt the high-speed double-edge triggered GPIO input mode * /
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 Types

Data types, 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-triggered mode.

5.4.1 gpio_drive_mode_t

5.4.1.1 GPIO drive mode is

described.

5.4.1.2 definitions

```
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 members

Member name	description
GPIO_DM_INPUT	Entry
GPIO_DM_INPUT_PULL_DOWN	input pulldown
GPIO_DM_INPUT_PULL_UP	Input pull
GPIO_DM_OUTPUT	Export

5.4.2 gpio_pin_value_t

5.4.2.1 Description

GPIO value.

5.4.2.2 definitions

```
typedef enum _ gpio_pin_value {

    GPIO_PV_LOW,
  GPIO_PV_HIGH}
  gpio_pin_value_t;
```

5.4.2.3 members

Member name	description
GPIO_PV_LOW	low
GPIO_PV_HIGH	high

5.4.3 gpio_pin_edge_t

5.4.3.1 Description

High-speed GPIO edge-triggered mode.

5.4.3.2 definitions

```
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 members

Member name	description
GPIO_PE_NONE	Does not trigger
GPIO_PE_FALLING	falling trigger
GPIO_PE_RISING	trigger
GPIO_PE_BOTH	Double-edge trigger

The first 6 chapter

A field programmable array IO (FPIOA)

6.1 Overview

FPIOA (Field Programmable Array IO) 255 allows the user to map the internal to the free 48 IO chip periphery.

6.2 Functional Description

- IO supports programmable function selection
- 8 kinds of output drive capability to support IO selection
- Select pull-up resistors on the internal support IO
- IO support internal pull-down resistor selection
- IO support internal Schmitt trigger input settings
- IO support output slew rate control
- Support internal setting input logic level

6.3 API Reference

Corresponding header file fpioa.h

To provide users with 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

Provided IO0-IO47 alternate functions.

6.3.1.2 function prototype

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

6.3.1.3 Parameters

Parameter Name	Description	input Output
number	IO pin number input	
No function pin function input		

6.3.1.4 Return value

Return Value	Description
0	success
Non-0	failure

6.3.2 fpioa_get_io_by_function

6.3.2.1 Description

Gets IO pin number based on the function number.

6.3.2.2 function prototype

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

6.3.2.3 Parameters

Parameter Name	Description	input Output
function	function input pin number	

6.3.2.4 Return value

return value	description
Greater than or equal 0	IO pin number is
less than 0	failure

6.3.3 fpioa_get_io

6.3.3.1 Description obtain

configuration IO pins.

6.3.3.2 function prototype

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

6.3.3.3 Parameters

Parameter Name	Description	input Output
number	IO Pin Number	Entry
cfg	Pin function output structure	

6.3.3.4 Return value

Return Value	Description
0	success
Non-0	failure

6.3.4 fpioa_set_io

6.3.4.1 Description Settings

Configure IO pins.

6.3.4.2 function prototype

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

6.3.4.3 Parameters

Parameter Name	Description	input Output
number	IO Pin Number	Entry
cfg	Input pin functional structure	

6.3.4.4 Return value

Return Value	Description
0	success
Non-0	failure

6.3.5 fpioa_set_tie_enable

6.3.5.1 Description

Enable disable function pins TIE.

6.3.5.2 function prototype

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

6.3.5.3 Parameters

Parameter Name	Description	input Output
No function pin function		Entry
enable	TIE Enable Bit 0: Disabled 1: enable input	

6.3.5.4 Return value

Return Value	Description
0	success
Non-0	failure

6.3.6 fpioa_set_tie_value

6.3.6.1 Description

On the Set Feature pull-down pin TIE.

6.3.6.2 function prototype

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

6.3.6.3 Parameters

Parameter Name	Description	input Output
No function pin function		Entry
value	TIE values 0: 1 pull-down: the pull-input	

6.3.6.4 Return value

Return Value	Description
0	success
Non-0	failure

6.3.7 fpioa_set_pull

6.3.7.1 Set the IO of the

pull-down.

6.3.7.2 function prototype

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

6.3.7.3 Parameters

Parameter Name	Description	input Output
number	Enter the number of IO	
pull	On the drop-down value input	

6.3.7.4 Return value

Return Value	Description
0	success
Non-0	failure

6.3.8 fpioa_get_pull

6.3.8.1 Description

Drop-down on the value of acquired IO pins.

6.3.8.2 function prototype

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

6.3.8.3 Parameters

Parameter Name	Description	Input	Output
number	Enter the number of IO		

6.3.8.4 Return value

Return Value	Description
0	Supreme
dropdown 1	2
pull-down	pull up

6.3.9 fpioa_set_io_driving

6.3.10 Description

Set drive capability IO pins.

6.3.10.1 function prototype

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

6.3.10.2 Parameters

Parameter Name	Description	input Output
number	Enter the number of IO	
driving	input drive capability	

6.3.10.3 return value

Return Value	Description
0	success
Non-0	failure

6.3.11 fpioa_get_io_driving

6.3.11.1 Description Get

driving ability.

6.3.11.2 function prototype

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

6.3.11.3 Parameters

Parameter Name	Description	Input Output
number	Enter the number of IO	

6.3.11.4 return value

return value	description
Driving capability than 0	less than 0 failure

6.4 Data Type

Data types, data structures are defined as follows:

- fpioa_function_t: function number pin.
- fpioa_io_config_t: Configure FPIOA of.

- fpioa_pull_t: pull-down function value FPIOA pull pin.
- fpioa_driving_t: FPIOA drive capability numbers.

6.4.1 fpioa_io_config_t

6.4.1.1 FPIOA described

configuration.

6.4.1.2 definitions

```
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 members

Member Name	Description
Pin function configuration ch_sel number ds	
	Select drive capability, the ability to choose the reference driving table
oe_en	1: Output Enable 0: Output Off
oe_inv	1: Enable inverted output 0: Disable inverted output
do_sel	1: 0 output high: low output
The output level of pu do_inv	
	1: 0 pull can enable: pull-off
pd	1: 0 down enable: Close dropdown
resv0	Reserved bits
sl	Slew rate control enabled

Member Name	Description
ie_en	1: Input Enable 0: Input Close
ie_inv	1: Enable inverted input 0: Disable inverting input
di_inv	inverted input data st
	Schmitt trigger
resv1	Reserved bits
pad_di	reads the current input pin level

6.4.2 drive capacity selection table

6.4.2.1 Low-level 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 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

Value pull-down function FPIOA pull pin.

6.4.3.2 definitions

```
typedef enum _ fpioa_pull {

    FPIOA_PULL_NONE,
FPIOA_PULL_DOWN,
FPIOA_PULL_UP,
FPIOA_PULL_MAX} fpioa_pull_t;
```

6.4.3.3 members

Member name	description
FPIOA_PULL_NONE	Supreme drop-down
FPIOA_PULL_DOWN	drop-down
FPIOA_PULL_UP	pull up

6.4.4 fpioa_driving_t

6.4.4.1 Description

FPIOA drive capability numbers, see the drive capacity selection table.

6.4.4.2 definitions

```
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 members

Member name	description
FPIOA_DRIVING_0	driving capability 0
FPIOA_DRIVING_1	1 FPIOA_DRIVING_2 driving ability of the driving capability of the driving capability of 3 FPIOA_DRIVING_4 2 FPIOA_DRIVING_3 4 FPIOA_DRIVING_5 driving ability of the driving capability of the driving capability of 5 FPIOA_DRIVING_6 drive capability 7 6 FPIOA_DRIVING_7

6.4.5 fpioa_function_t

6.4.5.1 Description The pin function numbers.

6.4.5.2 definitions

```
typedef enum _ fpioa_function {

    FUNC_JTAG_TCLK          = 0, /*! <JTAG Test Clock */
    FUNC_JTAG_TDI           = 1 /*! <JTAG Test Data In */
    FUNC_JTAG_TMS           = 2 /*! <JTAG Test Mode Select */
    FUNC_JTAG_TDO           = 3, /*! <JTAG Test Data Out */
    FUNC_SPI0_D0            = 4 /*! <SPI0 Data 0 */
    FUNC_SPI0_D1            = 5 /*! <SPI0 Data 1 */
    FUNC_SPI0_D2            = 6 /*! <SPI0 Data 2 */
    FUNC_SPI0_D3            = 7 /*! <SPI0 Data 3 */
    FUNC_SPI0_D4            = 8 /*! <SPI0 Data 4 */
    FUNC_SPI0_D5            = 9, /*! <SPI0 Data 5 */
    FUNC_SPI0_D6            = 10, /*! <SPI0 Data 6 */
    FUNC_SPI0_D7            = 11 /*! <SPI0 Data 7 */
    FUNC_SPI0_SS0           = 12, /*! <SPI0 Chip Select 0 */
    FUNC_SPI0_SS1           = 13 /*! <SPI0 Chip Select 1 */
    FUNC_SPI0_SS2           = 14, /*! <SPI0 Chip Select 2 */
    FUNC_SPI0_SS3           = 15, /*! <SPI0 Chip Select 3 */
    FUNC_SPI0_ARB           = 16, /*! <SPI0 Arbitration */
    FUNC_SPI0_SCLK          = 17 /*! <SPI0 Serial Clock */
    FUNC_UARTHS_RX          = 18, /*! <UART High speed I2S_RECEIVER */
    FUNC_UARTHS_TX          = 19 /*! <UART High speed I2S_TRANSMITTER */
    FUNC_RESV6              = 20, /*! <Reserved function */
    FUNC_RESV7              = 21 /*! <Reserved function */
    FUNC_CLK_SPI1           = 22, /*! <Clock SPI1 */
    FUNC_CLK_I2C1           = 23 /*! <Clock I2C1 */
    FUNC_GPIOHS0            = 24, /*! <GPIO High speed 0 */
    FUNC_GPIOHS1            = 25 /*! <GPIO High speed 1 */
    FUNC_GPIOHS2            = 26 /*! <GPIO High speed 2 */
    FUNC_GPIOHS3            = 27 /*! <GPIO High speed 3 */
```

```
    FUNC_GPIOHS4                = 28, /*! <GPIO High speed 4 */
    FUNC_GPIOHS5                = 29 /*! <GPIO High speed 5 */
    FUNC_GPIOHS6                = 30, /*! <GPIO High speed 6 */
    FUNC_GPIOHS7                = 31 /*! <GPIO High speed 7 */
    FUNC_GPIOHS8                = 32, /*! <GPIO High speed 8 */
    FUNC_GPIOHS9                = 33, /*! <GPIO High speed 9 */
    FUNC_GPIOHS10               = 34, /*! <GPIO High speed 10 */
    FUNC_GPIOHS11               = 35, /*! <GPIO High speed 11 */
    FUNC_GPIOHS12               = 36, /*! <GPIO High speed 12 */
    FUNC_GPIOHS13               = 37, /*! <GPIO High speed 13 */
    FUNC_GPIOHS14               = 38, /*! <GPIO High speed 14 */
    FUNC_GPIOHS15               = 39, /*! <GPIO High speed 15 */
    FUNC_GPIOHS16               = 40, /*! <GPIO High speed 16 */
    FUNC_GPIOHS17               = 41, /*! <GPIO High speed 17 */
    FUNC_GPIOHS18               = 42, /*! <GPIO High speed 18 */
    FUNC_GPIOHS19               = 43, /*! <GPIO High speed 19 */
    FUNC_GPIOHS20               = 44, /*! <GPIO High speed 20 */
    FUNC_GPIOHS21               = 45 /*! <GPIO High speed 21 */
    FUNC_GPIOHS22               = 46, /*! <GPIO High speed 22 */
    FUNC_GPIOHS23               = 47 /*! <GPIO High speed 23 */
    FUNC_GPIOHS24               = 48 /*! <GPIO High speed 24 */
    FUNC_GPIOHS25               = 49 /*! <GPIO High speed 25 */
    FUNC_GPIOHS26               = 50, /*! <GPIO High speed 26 */
    FUNC_GPIOHS27               = 51, /*! <GPIO High speed 27 */
    FUNC_GPIOHS28               = 52, /*! <GPIO High speed 28 */
    FUNC_GPIOHS29               = 53 /*! <GPIO High speed 29 */
    FUNC_GPIOHS30               = 54, /*! <GPIO High speed 30 */
    FUNC_GPIOHS31               = 55 /*! <GPIO High speed 31 */
    FUNC_GPIO0                  = 56, /*! <GPIO pin 0 */
    FUNC_GPIO1                  = 57 /*! <GPIO pin 1 */
    FUNC_GPIO2                  = 58 /*! <GPIO pin 2 */
    FUNC_GPIO3                  = 59 /*! <GPIO pin 3 */
    FUNC_GPIO4                  = 60, /*! <GPIO pin 4 */
    FUNC_GPIO5                  = 61 /*! <GPIO pin 5 */
    FUNC_GPIO6                  = 62, /*! <GPIO pin 6 */
    FUNC_GPIO7                  = 63 /*! <GPIO pin 7 */
    FUNC_UART1_RX               = 64, /*! <UART1 I2S_RECEIVER */
    FUNC_UART1_TX               = 65 /*! <UART1 I2S_TRANSMITTER */
    FUNC_UART2_RX               = 66 /*! <UART2 I2S_RECEIVER */
    FUNC_UART2_TX               = 67 /*! <UART2 I2S_TRANSMITTER */
    FUNC_UART3_RX               = 68 /*! <UART3 I2S_RECEIVER */
    FUNC_UART3_TX               = 69 /*! <UART3 I2S_TRANSMITTER */
    FUNC_SPI1_D0                = 70, /*! <SPI1 Data 0 */
    FUNC_SPI1_D1                = 71, /*! <SPI1 Data 1 */
    FUNC_SPI1_D2                = 72, /*! <SPI1 Data 2 */
    FUNC_SPI1_D3                = 73 /*! <SPI1 Data 3 */
    FUNC_SPI1_D4                = 74 /*! <SPI1 Data 4 */
    FUNC_SPI1_D5                = 75 /*! <SPI1 Data 5 */
    FUNC_SPI1_D6                = 76 /*! <SPI1 Data 6 */
    FUNC_SPI1_D7                = 77 /*! <SPI1 Data 7 */
    FUNC_SPI1_SS0               = 78 /*! <SPI1 Chip Select 0 */
    FUNC_SPI1_SS1               = 79 /*! <SPI1 Chip Select 1 */
    FUNC_SPI1_SS2               = 80, /*! <SPI1 Chip Select 2 */
```

```
    FUNC_SPI1_SS3               = 81 /*! <SPI1 Chip Select 3 */
    FUNC_SPI1_ARB               = 82, /*! <SPI1 Arbitration */
    FUNC_SPI1_SCLK              = 83 /*! <SPI1 Serial Clock */
    FUNC_SPI_SLAVE_D0           = 84 /*! <SPI Slave Data 0 */
    FUNC_SPI_SLAVE_SS           = 85 /*! <SPI Slave Select */
    FUNC_SPI_SLAVE_SCLK         = 86 /*! <SPI Slave Serial Clock */
    FUNC_I2S0_MCLK              = 87 /*! <I2S0 Master Clock */
    FUNC_I2S0_SCLK              = 88 /*! <I2S0 Serial Clock (BCLK) */
    FUNC_I2S0_WS                = 89 /*! <I2S0 Word Select (LRCLK) */
    FUNC_I2S0_IN_D0             = 90 /*! <I2S0 Serial Data Input 0 */
    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_PWDN          = 134,      /*! <DVP Power Down Mode */
FUNC_CMOS_VSYNC         = 135,      /*! <DVP Vertical Sync */
FUNC_CMOS_HREF          = 136,      /*! <DVP Horizontal Reference output */
FUNC_CMOS_PCLK          = 137       /*! <Pixel Clock */
FUNC_CMOS_D0            = 138,      /*! <Data Bit 0 */
FUNC_CMOS_D1            = 139,      /*! <Data Bit 1 */
FUNC_CMOS_D2            = 140,      /*! <Data Bit 2 */
FUNC_CMOS_D3            = 141,      /*! <Data Bit 3 */
FUNC_CMOS_D4            = 142,      /*! <Data Bit 4 */
FUNC_CMOS_D5            = 143       /*! <Data Bit 5 */
FUNC_CMOS_D6            = 144,      /*! <Data Bit 6 */
FUNC_CMOS_D7            = 145,      /*! <Data Bit 7 */
FUNC_SCCB_SCLK          = 146       /*! <SCCB Serial Clock */
FUNC_SCCB_SDA           = 147,      /*! <SCCB Serial Data */
FUNC_UART1_CTS          = 148       /*! <UART1 Clear To Send */
FUNC_UART1_DSR          = 149,      /*! <UART1 Data Set Ready */
FUNC_UART1_DCD          = 150,      /*! <UART1 Data Carrier Detect */
FUNC_UART1_RI           = 151,      /*! <UART1 Ring Indicator */
FUNC_UART1_SIR_IN       = 152,      /*! <UART1 Serial Infrared Input */
FUNC_UART1_DTR          = 153,      /*! <UART1 Data Terminal Ready */
FUNC_UART1_RTS          = 154,      /*! <UART1 Request To Send */
FUNC_UART1_OUT2         = 155,      /*! <UART1 User -designated Output 2 */
FUNC_UART1_OUT1         = 156,      /*! <UART1 User -designated Output 1 */
FUNC_UART1_SIR_OUT      = 157,      /*! <UART1 Serial Infrared Output */
FUNC_UART1_BAUD         = 158       /*! <UART1 Transmit Clock Output */
FUNC_UART1_RE           = 159       /*! <UART1 I2S_RECEIVER Output Enable */
FUNC_UART1_DE           = 160,      /*! <UART1 Driver Output Enable */
FUNC_UART1_RS485_EN     = 161       /*! <UART1 RS485 Enable */
FUNC_UART2_CTS          = 162,      /*! <UART2 Clear To Send */
FUNC_UART2_DSR          = 163,      /*! <UART2 Data Set Ready */
FUNC_UART2_DCD          = 164,      /*! <UART2 Data Carrier Detect */
FUNC_UART2_RI           = 165,      /*! <UART2 Ring Indicator */
FUNC_UART2_SIR_IN       = 166,      /*! <UART2 Serial Infrared Input */
FUNC_UART2_DTR          = 167       /*! <UART2 Data Terminal Ready */
FUNC_UART2_RTS          = 168       /*! <UART2 Request To Send */
FUNC_UART2_OUT2         = 169,      /*! <UART2 User -designated Output 2 */
FUNC_UART2_OUT1         = 170,      /*! <UART2 User -designated Output 1 */
FUNC_UART2_SIR_OUT      = 171,      /*! <UART2 Serial Infrared Output */
FUNC_UART2_BAUD         = 172,      /*! <UART2 Transmit Clock Output */
FUNC_UART2_RE           = 173,      /*! <UART2 I2S_RECEIVER Output Enable */
FUNC_UART2_DE           = 174,      /*! <UART2 Driver Output Enable */
FUNC_UART2_RS485_EN     = 175,      /*! <UART2 RS485 Enable */
FUNC_UART3_CTS          = 176       /*! <UART3 Clear To Send */
FUNC_UART3_DSR          = 177,      /*! <UART3 Data Set Ready */
FUNC_UART3_DCD          = 178,      /*! <UART3 Data Carrier Detect */
FUNC_UART3_RI           = 179       /*! <UART3 Ring Indicator */
FUNC_UART3_SIR_IN       = 180,      /*! <UART3 Serial Infrared Input */
FUNC_UART3_DTR          = 181,      /*! <UART3 Data Terminal Ready */
FUNC_UART3_RTS          = 182       /*! <UART3 Request To Send */
FUNC_UART3_OUT2         = 183,      /*! <UART3 User -designated Output 2 */
FUNC_UART3_OUT1         = 184,      /*! <UART3 User -designated Output 1 */
FUNC_UART3_SIR_OUT      = 185,      /*! <UART3 Serial Infrared Output */
FUNC_UART3_BAUD         = 186,      /*! <UART3 Transmit Clock Output */
```

```
    FUNC_UART3_RE               = 187,      /*! <UART3 I2S_RECEIVER Output Enable */
    FUNC_UART3_DE               = 188,      /*! <UART3 Driver Output Enable */
    FUNC_UART3_RS485_EN         = 189,      /*! <UART3 RS485 Enable */
    FUNC_TIMER0_TOGGLE1         = 190       /*! <TIMER0 Toggle Output 1 */
    FUNC_TIMER0_TOGGLE2         = 191,      /*! <TIMER0 Toggle Output 2 */
    FUNC_TIMER0_TOGGLE3         = 192,      /*! <TIMER0 Toggle Output 3 */
    FUNC_TIMER0_TOGGLE4         = 193,      /*! <TIMER0 Toggle Output 4 */
    FUNC_TIMER1_TOGGLE1         = 194,      /*! <TIMER1 Toggle Output 1 */
    FUNC_TIMER1_TOGGLE2         = 195,      /*! <TIMER1 Toggle Output 2 */
    FUNC_TIMER1_TOGGLE3         = 196,      /*! <TIMER1 Toggle Output 3 */
    FUNC_TIMER1_TOGGLE4         = 197,      /*! <TIMER1 Toggle Output 4 */
    FUNC_TIMER2_TOGGLE1         = 198,      /*! <TIMER2 Toggle Output 1 */
    FUNC_TIMER2_TOGGLE2         = 199       /*! <TIMER2 Toggle Output 2 */
    FUNC_TIMER2_TOGGLE3         = 200,      /*! <TIMER2 Toggle Output 3 */
    FUNC_TIMER2_TOGGLE4         = 201,      /*! <TIMER2 Toggle Output 4 */
    FUNC_CLK_SPI2               = 202,      /*! <Clock SPI2 */
    FUNC_CLK_I2C2               = 203,      /*! <Clock I2C2 */
    FUNC_INTERNAL0              = 204,      /*! <Internal function signal 0 */
    FUNC_INTERNAL1              = 205,      /*! <Internal function signal 1 */
    FUNC_INTERNAL2              = 206,      /*! <Internal function signal 2 */
    FUNC_INTERNAL3              = 207,      /*! <Internal function signal 3 */
    FUNC_INTERNAL4              = 208,      /*! <Internal function signal 4 */
    FUNC_INTERNAL5              = 209,      /*! <Internal function signal 5 */
    FUNC_INTERNAL6              = 210,      /*! <Internal function signal 6 */
    FUNC_INTERNAL7              = 211,      /*! <Internal function signal 7 */
    FUNC_INTERNAL8              = 212,      /*! <Internal function signal 8 */
    FUNC_INTERNAL9              = 213,      /*! <Internal function signal 9 */
    FUNC_INTERNAL10             = 214,      /*! <Internal function signal 10 */
    FUNC_INTERNAL11             = 215,      /*! <Internal function signal 11 */
    FUNC_INTERNAL12             = 216,      /*! <Internal function signal 12 */
    FUNC_INTERNAL13             = 217,      /*! <Internal function signal 13 */
    FUNC_INTERNAL14             = 218,      /*! <Internal function signal 14 */
    FUNC_INTERNAL15             = 219,      /*! <Internal function signal 15 */
    FUNC_INTERNAL16             = 220,      /*! <Internal function signal 16 */
    FUNC_INTERNAL17             = 221       /*! <Internal function signal 17 */
    FUNC_CONSTANT               = 222,      /*! <Constant function */
    FUNC_INTERNAL18             = 223       /*! <Internal function signal 18 */
    FUNC_DEBUG0                 = 224,      /*! <Debug function 0 */
    FUNC_DEBUG1                 = 225,      /*! <Debug function 1 */
    FUNC_DEBUG2                 = 226       /*! <Debug function 2 */
    FUNC_DEBUG3                 = 227,      /*! <Debug function 3 */
    FUNC_DEBUG4                 = 228       /*! <Debug function 4 */
    FUNC_DEBUG5                 = 229       /*! <Debug function 5 */
    FUNC_DEBUG6                 = 230,      /*! <Debug function 6 */
    FUNC_DEBUG7                 = 231,      /*! <Debug function 7 */
    FUNC_DEBUG8                 = 232,      /*! <Debug function 8 */
    FUNC_DEBUG9                 = 233       /*! <Debug function 9 */
    FUNC_DEBUG10                = 234,      /*! <Debug function 10 */
    FUNC_DEBUG11                = 235,      /*! <Debug function 11 */
    FUNC_DEBUG12                = 236,      /*! <Debug function 12 */
    FUNC_DEBUG13                = 237,      /*! <Debug function 13 */
    FUNC_DEBUG14                = 238       /*! <Debug function 14 */
    FUNC_DEBUG15                = 239,      /*! <Debug function 15 */
```

```
    FUNC_DEBUG16                = 240,      /*! <Debug function 16 */
    FUNC_DEBUG17                = 241,      /*! <Debug function 17 */
    FUNC_DEBUG18                = 242,      /*! <Debug function 18 */
    FUNC_DEBUG19                = 243       /*! <Debug function 19 */
    FUNC_DEBUG20                = 244       /*! <Debug function 20 */
    FUNC_DEBUG21                = 245       /*! <Debug function 21 */
    FUNC_DEBUG22                = 246,      /*! <Debug function 22 */
    FUNC_DEBUG23                = 247,      /*! <Debug function 23 */
    FUNC_DEBUG24                = 248       /*! <Debug function 24 */
    FUNC_DEBUG25                = 249,      /*! <Debug function 25 */
    FUNC_DEBUG26                = 250,      /*! <Debug function 26 */
    FUNC_DEBUG27                = 251,      /*! <Debug function 27 */
    FUNC_DEBUG28                = 252       /*! <Debug function 28 */
    FUNC_DEBUG29                = 253,      /*! <Debug function 29 */
    FUNC_DEBUG30                = 254       /*! <Debug function 30 */
    FUNC_DEBUG31                = 255,      /*! <Debug function 31 */
    FUNC_MAX                    = 256,      /*! <Function numbers */
} Fpioa_function_t;
```

6.4.5.3 members

Member name	description
FUNC_JTAG_TCLK	JTAG clock interface
FUNC_JTAG_TDI	JTAG data input interface
FUNC_JTAG_TMS	JTAG TAP state machine controls the conversion
FUNC_JTAG_TDO	JTAG data output interface
FUNC_SPI0_D0	SPI0 data line 0
FUNC_SPI0_D1	SPI0 data line 1
FUNC_SPI0_D2	SPI0 data line 2
FUNC_SPI0_D3	Data line 3 SPI0
FUNC_SPI0_D4	SPI0 data line 4
FUNC_SPI0_D5	SPI0 data line 5
FUNC_SPI0_D6	SPI0 data line 6
FUNC_SPI0_D7	Data lines 7 SPI0
FUNC_SPI0_SS0	SPI0 chip select signal 0
FUNC_SPI0_SS1	A chip select signal SPI0
FUNC_SPI0_SS2	SPI0 chip select signal 2
FUNC_SPI0_SS3	SPI0 chip select signal 3
FUNC_SPI0_ARB	SPI0 arbitration signals
FUNC_SPI0_SCLK	SPI0 clock
FUNC_UARTHS_RX	Receiving a high-speed data interface UART
FUNC_UARTHS_TX	UART transmit high-speed data interface

Member name	description
FUNC_RESV6	Hold feature
FUNC_RESV7	Hold feature
FUNC_CLK_SPI1	SPI1 clock
FUNC_CLK_I2C1	I2C1 clock
FUNC_GPIOHS0	High-speed GPIO0
FUNC_GPIOHS1	High-speed GPIO1
FUNC_GPIOHS2	High-speed GPIO2
FUNC_GPIOHS3	High-speed GPIO3
FUNC_GPIOHS4	High-speed GPIO4
FUNC_GPIOHS5	High-speed GPIO5
FUNC_GPIOHS6	High-speed GPIO6
FUNC_GPIOHS7	High-speed GPIO7
FUNC_GPIOHS8	High-speed GPIO8
FUNC_GPIOHS9	High-speed GPIO9
FUNC_GPIOHS10	High-speed GPIO10
FUNC_GPIOHS11	High-speed GPIO11
FUNC_GPIOHS12	High-speed GPIO12
FUNC_GPIOHS13	High-speed GPIO13
FUNC_GPIOHS14	High-speed GPIO14
FUNC_GPIOHS15	High-speed GPIO15
FUNC_GPIOHS16	High-speed GPIO16
FUNC_GPIOHS17	High-speed GPIO17
FUNC_GPIOHS18	High-speed GPIO18
FUNC_GPIOHS19	High-speed GPIO19
FUNC_GPIOHS20	High-speed GPIO20
FUNC_GPIOHS21	High-speed GPIO21
FUNC_GPIOHS22	High-speed GPIO22
FUNC_GPIOHS23	High-speed GPIO23
FUNC_GPIOHS24	High-speed GPIO24
FUNC_GPIOHS25	High-speed GPIO25
FUNC_GPIOHS26	High-speed GPIO26
FUNC_GPIOHS27	High-speed GPIO27
FUNC_GPIOHS28	High-speed GPIO28
FUNC_GPIOHS29	High-speed GPIO29
FUNC_GPIOHS30	High-speed GPIO30

Member name	description
FUNC_GPIOHS31	High-speed GPIO31
FUNC_GPIO0	GPIO0
FUNC_GPIO1	GPIO1
FUNC_GPIO2	GPIO2
FUNC_GPIO3	GPIO3
FUNC_GPIO4	GPIO4
FUNC_GPIO5	GPIO5
FUNC_GPIO6	GPIO6
FUNC_GPIO7	GPIO7
FUNC_UART1_RX	UART1 receive data interface
FUNC_UART1_TX	Transmitting data interface UART1
FUNC_UART2_RX	UART2 receive data interface
FUNC_UART2_TX	UART2 transmit data interface
FUNC_UART3_RX	Receiving data interface UART3
FUNC_UART3_TX	Transmitting data interface UART3
FUNC_SPI1_D0	SPI1 data line 0
FUNC_SPI1_D1	SPI1 data line 1
FUNC_SPI1_D2	SPI1 data line 2
FUNC_SPI1_D3	SPI1 data line 3
FUNC_SPI1_D4	SPI1 data line 4
FUNC_SPI1_D5	SPI1 data line 5
FUNC_SPI1_D6	SPI1 data line 6
FUNC_SPI1_D7	SPI1 data line 7
FUNC_SPI1_SS0	SPI1 chip select signal 0
FUNC_SPI1_SS1	SPI1 chip select signal 1
FUNC_SPI1_SS2	SPI1 chip select signal 2
FUNC_SPI1_SS3	SPI1 chip select signal 3
FUNC_SPI1_ARB	SPI1 arbitration signals
FUNC_SPI1_SCLK	SPI1 clock
FUNC_SPI_SLAVE_D0	SPI mode from the data line 0
FUNC_SPI_SLAVE_SS	SPI chip select signal from the mode
FUNC_SPI_SLAVE_SCLK	SPI mode clock from
FUNC_I2S0_MCLK	I2S0 master clock (system clock)
FUNC_I2S0_SCLK	I2S0 serial clock (bit clock)
FUNC_I2S0_WS	I2S0 frame clock

Member name	description
FUNC_I2S0_IN_D0	I2S0 serial input data interface 0
FUNC_I2S0_IN_D1	I2S0 serial input data interface 1
FUNC_I2S0_IN_D2	I2S0 serial input data interface 2
FUNC_I2S0_IN_D3	I2S0 serial input data interface 3
FUNC_I2S0_OUT_D0	I2S0 serial output data interface 0
FUNC_I2S0_OUT_D1	I2S0 serial output data interface 1
FUNC_I2S0_OUT_D2	I2S0 serial output data interface 2
FUNC_I2S0_OUT_D3	I2S0 serial output data interface 3
FUNC_I2S1_MCLK	I2S1 master clock (system clock)
FUNC_I2S1_SCLK	I2S1 serial clock (bit clock)
FUNC_I2S1_WS	I2S1 frame clock
FUNC_I2S1_IN_D0	I2S1 serial input data interface 0
FUNC_I2S1_IN_D1	I2S1 serial input data interface 1
FUNC_I2S1_IN_D2	I2S1 serial input data interface 2
FUNC_I2S1_IN_D3	I2S1 serial input data interface 3
FUNC_I2S1_OUT_D0	I2S1 serial output data interface 0
FUNC_I2S1_OUT_D1	I2S1 serial output data interface 1
FUNC_I2S1_OUT_D2	I2S1 serial output data interface 2
FUNC_I2S1_OUT_D3	I2S1 serial output data interface 3
FUNC_I2S2_MCLK	I2S2 master clock (system clock)
FUNC_I2S2_SCLK	I2S2 serial clock (bit clock)
FUNC_I2S2_WS	I2S2 frame clock
FUNC_I2S2_IN_D0	I2S2 serial input data interface 0
FUNC_I2S2_IN_D1	I2S2 serial input data interface 1
FUNC_I2S2_IN_D2	I2S2 serial input data interface 2
FUNC_I2S2_IN_D3	I2S2 serial input data interface 3
FUNC_I2S2_OUT_D0	I2S2 serial output data interface 0
FUNC_I2S2_OUT_D1	I2S2 serial output data interface 1
FUNC_I2S2_OUT_D2	I2S2 serial output data interface 2
FUNC_I2S2_OUT_D3	I2S2 serial output data interface 3
FUNC_RESV0	Hold feature
FUNC_RESV1	Hold feature
FUNC_RESV2	Hold feature
FUNC_RESV3	Hold feature
FUNC_RESV4	Hold feature

Member name	description
FUNC_RESV5	Hold feature
FUNC_I2C0_SCLK	I2C0 Serial Clock
FUNC_I2C0_SDA	I2C0 serial data interface
FUNC_I2C1_SCLK	I2C1 Serial Clock
FUNC_I2C1_SDA	I2C1 serial data interface
FUNC_I2C2_SCLK	I2C2 Serial Clock
FUNC_I2C2_SDA	I2C2 serial data interface
FUNC_CMOS_XCLK	DVP system clock
FUNC_CMOS_RST	DVP system reset signal
FUNC_CMOS_PWDN	DVP enable signal
FUNC_CMOS_VSYNC	DVP field synchronization
FUNC_CMOS_HREF	Line Reference signal DVP
FUNC_CMOS_PCLK	Pixel Clock
FUNC_CMOS_D0	Pixel data 0
FUNC_CMOS_D1	1 pixel data
FUNC_CMOS_D2	Pixel data 2
FUNC_CMOS_D3	3 pixel data
FUNC_CMOS_D4	4 pixel data
FUNC_CMOS_D5	Pixel data 5
FUNC_CMOS_D6	Pixel data of 6
FUNC_CMOS_D7	Pixel data 7
FUNC_SCCB_SCLK	SCCB clock
FUNC_SCCB_SDA	SCCB serial data signal
FUNC_UART1_CTS	UART1 send a clear signal
FUNC_UART1_DSR	UART1 device prepares a data signal
FUNC_UART1_DCD	Data Carrier Detect UART1
FUNC_UART1_RI	UART1 Ring Indicator
FUNC_UART1_SIR_IN	UART1 serial infrared input signal
FUNC_UART1_DTR	Data Terminal Ready signal UART1
FUNC_UART1_RTS	UART1 transmission request signal
FUNC_UART1_OUT2	User-specified output signal UART1 2
FUNC_UART1_OUT1	A user-specified output signal UART1
FUNC_UART1_SIR_OUT	Infrared UART1 serial output signal
FUNC_UART1_BAUD	UART1 clock
FUNC_UART1_RE	UART1 receive enable

Member name	description
FUNC_UART1_DE	UART1 transmit enable
FUNC_UART1_RS485_EN	UART1 Enable RS485
FUNC_UART2_CTS	UART2 send a clear signal
FUNC_UART2_DSR	UART2 device prepares a data signal
FUNC_UART2_DCD	UART2 Data Carrier Detect
FUNC_UART2_RI	UART2 Ring Indicator
FUNC_UART2_SIR_IN	UART2 serial infrared input signal
FUNC_UART2_DTR	UART2 Data Terminal Ready signal
FUNC_UART2_RTS	UART2 transmit request signal
FUNC_UART2_OUT2	User-specified output signal UART2 2
FUNC_UART2_OUT1	A user-specified output signal UART2
FUNC_UART2_SIR_OUT	Infrared UART2 serial output signal
FUNC_UART2_BAUD	UART2 clock
FUNC_UART2_RE	UART2 receive enable
FUNC_UART2_DE	UART2 transmit enable
FUNC_UART2_RS485_EN	UART2 enable RS485
FUNC_UART3_CTS	Clear To Send signal
FUNC_UART3_DSR	Data equipment ready signal
FUNC_UART3_DCD	Data Carrier Detect UART3
FUNC_UART3_RI	UART3 Ring Indicator
FUNC_UART3_SIR_IN	UART3 serial infrared input signal
FUNC_UART3_DTR	Data Terminal Ready signal UART3
FUNC_UART3_RTS	Transmission request signal UART3
FUNC_UART3_OUT2	User-specified output signal UART3 2
FUNC_UART3_OUT1	A user-specified output signal UART3
FUNC_UART3_SIR_OUT	UART3 infrared serial output signal
FUNC_UART3_BAUD	UART3 clock
FUNC_UART3_RE	UART3 receive enable
FUNC_UART3_DE	UART3 transmit enable
FUNC_UART3_RS485_EN	UART3 enable RS485
FUNC_TIMER0_TOGGLE1	TIMER0 1 FUNC_TIMER0_TOGGLE2
TIMER0 outputs an output signal 2 FUNC_TIMER0_TOGGLE3	
TIMER0 4 FUNC_TIMER1_TOGGLE1	TIMER1 output signal of the
output signal of a signal output 3 FUNC_TIMER0_TOGGLE4	
TIMER0	

Member name	description
FUNC_TIMER1_TOGGLE2 TIMER1 2 FUNC_TIMER2_TOGGLE3	
TIMER2 1 FUNC_TIMER2_TOGGLE2 TIMER2 output signal of the	
output signal of the output signal 4 FUNC_TIMER2_TOGGLE1	
TIMER2 2 FUNC_TIMER1_TOGGLE3 TIMER1 3	
FUNC_TIMER1_TOGGLE4 TIMER1 output signal of the output	
signal of the output signal 3 FUNC_TIMER2_TOGGLE4 TIMER2 4	
FUNC_CLK_SPI2	
	SPI2 clock
FUNC_CLK_I2C2	I2C2 clock
FUNC_INTERNAL0	Internal functions 0
FUNC_INTERNAL1	Internal Function 1
FUNC_INTERNAL2	2 internal functions
FUNC_INTERNAL3	Internal function 3
FUNC_INTERNAL4	4 internal functions
FUNC_INTERNAL5	5 internal functions
FUNC_INTERNAL6	Internal functions 6
FUNC_INTERNAL7	Internal functions 7
FUNC_INTERNAL8	8 internal functions
FUNC_INTERNAL9	Internal functions 9
FUNC_INTERNAL10	10 internal functions
FUNC_INTERNAL11	11 internal functions
FUNC_INTERNAL12	12 internal functions
FUNC_INTERNAL13	13 internal functions
FUNC_INTERNAL14	14 internal functions
FUNC_INTERNAL15	15 internal functions
FUNC_INTERNAL16	16 internal functions
FUNC_INTERNAL17	17 internal functions
FUNC_CONSTANT	constant
FUNC_INTERNAL18	18 internal functions
FUNC_DEBUG0	Debugging functions 0
FUNC_DEBUG1	Debugging Function 1
FUNC_DEBUG2	Debugging Function 2
FUNC_DEBUG3	Debugging features 3
FUNC_DEBUG4	Debugging Function 4
FUNC_DEBUG5	Debugging features 5

Member name	description
FUNC_DEBUG6	Debugging functions 6
FUNC_DEBUG7	7 debug function
FUNC_DEBUG8	Debugging features 8
FUNC_DEBUG9	Debugging functions 9
FUNC_DEBUG10	Debugging features 10
FUNC_DEBUG11	Debugging functions 11
FUNC_DEBUG12	Debugging features 12
FUNC_DEBUG13	Debugging features 13
FUNC_DEBUG14	Debugging features 14
FUNC_DEBUG15	Debugging features 15
FUNC_DEBUG16	Debugging features 16
FUNC_DEBUG17	Debugging features 17
FUNC_DEBUG18	Debugging functions 18
FUNC_DEBUG19	Debugging features 19
FUNC_DEBUG20	Debugging features 20
FUNC_DEBUG21	Debugging features 21
FUNC_DEBUG22	Debugging features 22
FUNC_DEBUG23	Debugging features 23
FUNC_DEBUG24	Debugging features 24
FUNC_DEBUG25	Debugging features 25
FUNC_DEBUG26	Debugging features 26
FUNC_DEBUG27	Debugging features 27
FUNC_DEBUG28	Debugging features 28
FUNC_DEBUG29	Debugging features 29
FUNC_DEBUG30	Debugging functions 30
FUNC_DEBUG31	Debugging features 31

The first 7 chapter

Digital camera interface (DVP)

7.1 Overview

DVP is a camera interface module supports the camera input image data forwarded to the AI module or memory.

7.2 Functional Description

DVP module has the following features:

- RGB565 RGB24Planar of 2 and the video data output port
- Frame discard unwanted process

7.3 API Reference

Corresponding header file dvp.h

To provide users with 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

initialization DVP.

7.3.1.2 function prototype

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

7.3.1.3 Parameters

Parameter Name	Description	input Output
length	register input reg_len sccb	

7.3.1.4 Return value None

7.3.2 dvp_set_output_enable

7.3.2.1 Description

Set the output mode is enabled or disabled.

7.3.2.2 function prototype

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

7.3.2.3 Parameters

Parameter Name	Description	input Output
index	Image is output to memory or input AI	

Parameter Name	Description	input Output
enable	0: Disable 1: Enable	Entry

7.3.2.4 Return Value None.

7.3.3 dvp_set_image_format

7.3.3.1 Description

Setting an image receiving mode, RGB, or YUV.

7.3.3.2 function prototype

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

7.3.3.3 Parameters

Parameter Name	Description	input Output
DVP_CFG_RGB_FORMAT	RGB format image mode mode mode input DVP_CFG_YUV_FORMAT YUV	

7.3.3.4 Return value None

7.3.4 dvp_set_image_size

7.3.4.1 Description

DVP size setting image acquisition.

7.3.4.2 function prototype

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

7.3.4.3 Parameters

Parameter Name	Description	input Output
width	Input image width	
height	image height input	

7.3.4.4 Return value None

7.3.5 dvp_set_ai_addr

7.3.5.1 Description

Set image storage address AI, AI module for arithmetic 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 Parameters

Parameter Name	Description	input Output
r_addr	red component input address g_addr	
	green component blue component b_addr	
	address input address entered.	

7.3.5.4 Return value None

7.3.6 dvp_set_display_addr

7.3.6.1 Description

Provided captured image storage address in the memory, can be used to display.

7.3.6.2 function prototype

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

7.3.6.3 Parameters

Parameter Name	Description	input Output
addr	Image storage memory address input	

7.3.6.4 Return value None

7.3.7 dvp_config_interrupt

Configuring 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 the image start and end interrupt status, enabled or disabled.

7.3.7.3 Parameters

parameter name	description	input Output
interrupt	Interrupt type DVP_CFG_START_INT_ENABLE image began collecting interrupt DVP_CFG_FINISH_INT_ENABLE image capture end interrupt	Entry
enable	0: Disable 1: Enable	Entry

7.3.7.4 Return Value None.

7.3.8 dvp_get_interrupt

7.3.8.1 Description

Determining whether the input interrupt type.

7.3.8.2 function prototype

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

7.3.8.3 Parameters

parameter name	description	input Output
interrupt	Interrupt type DVP_CFG_START_INT_ENABLE image began collecting interrupt DVP_CFG_FINISH_INT_ENABLE image capture end interrupt	Entry

7.3.8.4 Return value

Return Value	Description
0	no
Non-0	Yes

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 Parameters

parameter name	description	input Output
interrupt	Interrupt type DVP_CFG_START_INT_ENABLE image began collecting interrupt DVP_CFG_FINISH_INT_ENABLE image capture end interrupt	Entry

7.3.9.4 Return Value None.

7.3.10 dvp_start_convert

7.3.10.1 Description

He began collecting images, called after determining the image acquisition start interrupt.

7.3.10.2 function prototype

```
void dvp_start_convert ( void)
```

7.3.10.3 Parameters None.

7.3.10.4 Return Value None.

7.3.11 dvp_enable_burst

7.3.11.1 enabling burst transfer mode

is described.

7.3.11.2 function prototype

```
void dvp_enable_burst ( void)
```

7.3.11.3 Parameters None.

7.3.11.4 Return Value None.

7.3.12 dvp_disable_burst

7.3.12.1 disable the burst transfer

mode is described.

7.3.12.2 function prototype

```
void dvp_disable_burst ( void)
```

7.3.12.3 Parameters None.

7.3.12.4 Return Value None.

7.3.13 dvp_enable_auto

7.3.13.1 Description

Enabling automatic reception mode image.

7.3.13.2 function prototype

```
void dvp_enable_auto ( void)
```

7.3.13.3 Parameters None.

7.3.13.4 Return Value None.

7.3.14 dvp_disable_auto

7.3.14.1 Description

Disable automatic receiving mode image.

7.3.14.2 function prototype

```
void dvp_disable_auto ( void)
```

7.3.14.3 Parameters None.

7.3.14.4 Return Value None.

7.3.15 dvp_sccb_send_data

7.3.15.1 sccb described

transmission data.

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 Parameters

Parameter Name	Description	input Output
dev_addr	peripheral address entered image sensor SCCB	

Parameter Name Description	input Output
The image sensor reg_addr peripheral register	Entry
Data transmitted reg_data	Entry

7.3.15.4 Return value None

7.3.16 dvp_sccb_receive_data

7.3.16.1 description SCCB

receiving data.

7.3.16.2 function prototype

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

7.3.16.3 Parameters

Parameter Name Description	input Output
dev_addr peripheral image sensor SCCB address entered image sensor registers the peripheral reg_addr	Entry

7.3.16.4 return value of the read

data register.

For example 7.3.17

```
/ * Acquired RGB image data to the lcd_gram0 320 * 240, and address 0x40600000 0x40612C00 0x40625800
        * /
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) 0 x40625800); 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 ();

/ * Sent to the address 0x60 0x01 0xFF peripheral registers SCCB, reading data from the register 0x1D * /
dvp_sccb_send_data (0x60, 0xFF, 0x01); dvp_sccb_receive_data
(0x60, 0x1D)
```

7.4 Data Type

Data types, 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 Model Description DVP

input image.

7.4.1.2 definitions

```
typedef enum _ dvp_output_mode {

    DVP_OUTPUT_AI,
DVP_OUTPUT_DISPLAY,}
dvp_output_mode_t;
```

7.4.1.3 members

Member name	description
DVP_OUTPUT_AI	AI output
A memory to an output display DVP_OUTPUT_DISPLAY	

The first 8 chapter

Fast Fourier Transform accelerator (FFT)

8.1 Overview

FFT module is a hardware approach to achieve time-division 2-yl FFT computation acceleration.

8.2 Functional Description

At present, the module can support 64-, 128-, 256- and 512-point FFT and IFFT. There are two internal FFT size of 512 * 32bit SRAM, TX FFT send request, send DMA data fed into the SRAM to which one, until the FFT operation to meet the current need in the DMA after completion of the configuration starts FFT operation and the amount of data, the butterfly with valid data

The SRAM data is read, data is written to another operation after the end of an SRAM to go, and then the next butterfly operation from the SRAM just written in

Read data, after the end of the operation and an additional write SRAM, thus, are alternately repeated until the entire FFT computation.

8.3 API Reference

Corresponding header file fft.h

To provide users with the following interfaces

- 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 Parameters

parameter name	description	input Output
dma_send_channel_num	Transmitting data using DMA channel numbers of	Entry
dma_receive_channel_num	DMA channel number of the received data used	Entry
shift	The FFT module 16 cause data overflow register (-32768 ~ 32767), FFT converting layer 9, shift needs to determine which one shift operation (e.g., 9 layer do 0x1ff represents a shift operation; 0x03 represents a first layer Do shift operation with the second layer), preventing overflow Out. If the shift is transformed pieces The normal value is not the amplitude of the FFT transform, the corresponding relationship can reference fft_test test demo program. It includes solving frequencies, Phase, amplitude examples	Entry
direction	FFT positive transformation or inverse transformation	Entry
input	Input data sequence, the format RIRI .., the accuracy of the real and imaginary parts are 16bit	Entry
point_num	The number of data points to be operational only as 512/256/128/64	Entry
output	After the operation result. Format RIRI .., the accuracy of the real and imaginary parts are 16bit	Export

8.3.1.4 Return Value None.

8.3.2 Examples

```
# 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

Data types, data structures are defined as follows:

- fft_data_t: FFT computation incoming data format.

- fft_direction_t: FFT transform mode.

8.4.1 fft_data_t

8.4.1.1 Description

FFT operation incoming data format.

8.4.1.2 definitions

```
typedef struct tag_fft_data {

    int16_t I1; int16_t R1;
int16_t I2; int16_t R2;}
fft_data_t;
```

8.4.1.3 members

Member Name	Description
I1	The imaginary portion of the first data
R1	The real portion of the first data
I2	The imaginary portion of the second data
R2	The real portion of the second data

8.4.2 fft_direction_t

8.4.2.1 Model Description

FFT transform

8.4.2.2 definitions

```
typedef enum _ fft_direction {

    FFT_DIR_BACKWARD,
FFT_DIR_FORWARD,
FFT_DIR_MAX,} fft_direction_t;
```

8.4.2.3 members

Member name	description
FFT_DIR_BACKWARD	FFT inverse transform
FFT_DIR_FORWARD	FFT positive transformation

The first 9 chapter

Accelerator Secure Hash Algorithm (SHA256)

9.1 Functional Description

- Calculation of SHA-256 support

9.2 API Reference

Corresponding header file sha256.h

To provide users with the following interfaces

- sha256_init
- sha256_update
- sha256_final
- sha256_hard_calculate

9.2.1 sha256_init

9.2.1.1 Description

Initialization SHA256 accelerator peripherals.

9.2.1.2 function prototype

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

9.2.1.3 Parameters

Parameter Name	Description	input Output
context	SHA256 context object	Entry
Input_len be calculated length of the input SHA256 hash of the message		

9.2.1.4 Return Value None.

9.2.2 Examples

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

9.2.3 sha256_update

9.2.3.1 Description

Passing a block of data involved in SHA256 Hash calculation

9.2.3.2 function prototype

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

9.2.3.3 Parameters

Parameter Name	Description	input Output
context	SHA256 context object	Entry
input	Calculation of the data block to be added to the calculated SHA256	Entry
SHA256 calculated length of the input data block to be added calculated buf_len		

9.2.3.4 Return Value None.

9.2.4 sha256_final

9.2.4.1 Description

The end of the data computing SHA256 Hash

9.2.4.2 function prototype

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

9.2.4.3 Parameters

Parameter Name	Description	input Output
context	SHA256 context object	Entry
	SHA256 output calculation result stored, the need to ensure that the buffer size is passed over the output 32Bytes	

9.2.4.4 Return Value None.

9.2.5 sha256_hard_calculate

9.2.5.1 Description

Its continuous-time calculation data SHA256 Hash

9.2.5.2 function prototype

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

9.2.5.3 Parameters

Parameter Name	Description	input Output
input	SHA256 be calculated data	Entry
Input_len	length of data to be calculated SHA256	Entry
output	SHA256 calculation result is stored, the need to ensure that the buffer size is passed over the output 32Bytes	

9.2.5.4 Return Value None.

9.2.6 Examples

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

9.3 Routine

9.3.1 evaluated once

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

Or you can call the function directly sha256_hard_calculate

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

9.3.2 Calculation block

```
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);
```

The first 10 chapter

Universal Asynchronous Receiver Transmitter (UART)

10.1 Overview

Embedded applications often require a small footprint and simple method for system resources to transmit data. Universal Asynchronous Receiver Transmitter (UART)

That is to meet these requirements, it is possible to flexibly perform full duplex data exchange with external devices.

10.2 Functional Description

UART module has the following features:

- UART configuration parameters
- To automatically collect data buffer

10.3 API Reference

Corresponding header file uart.h

To provide users with the following interfaces

- uart_init
- uart_config
- uart_send_data
- uart_receive_data

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 Parameters

Parameter Name	Description	Input Output
No.	input channel UART	

10.3.1.4 Return Value None.

10.3.2 uart_config

10.3.2.1 described in the UART

parameters.

10.3.2.2 function prototype

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

10.3.2.3 Parameters

Parameter Name	Description	input Output
channel	UART serial number input	
baud_rate	baud rate	Entry
data_width	data bit (5-8) input	
stopbit	Stop bits	Entry
parity	Check Digit	Entry

10.3.2.4 Return Value None.

10.3.3 uart_send_data

Description 10.3.3.1 UART

transmit data.

10.3.3.2 function prototype

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

10.3.3.3 Parameters

Parameter Name Description	input Output
channel UART serial number	Entry
buffer data to be sent	Entry
Length of the input data to be transmitted buf_len	

10.3.3.4 The return value is the length

of the transmitted data.

10.3.4 uart_receive_data

10.3.4.1 described by reading

data UART.

10.3.4.2 function prototype

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

10.3.4.3 Parameters

Parameter Name Description	input Output
channel UART serial number	Entry
receive data buffer	Export
Length of the input data received buf_len	

10.3.4.4 return value to the data length of

the received.

10.3.5 Examples

```
/ * The UART1 115200 baud, 8 data bits, 1 stop bit, no parity bits * /
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 hello world! * /
uart_send_data (UART_DEVICE_1, hel, strlen (v_hel));
/ * Received data * /
while ( uart_receive_data (UART_DEVICE_1, & recv, 1)) {

        printf ( "% c _ ", recv);}
```

10.4 Data Type

Data types, data structures are defined as follows:

- uart_device_number_t: UART number.
- uart_bitwidth_t: UART data bits wide.
- uart_stopbits_t: UART stop bit.
- uart_parity_t: UART parity bit.

10.4.1 uart_device_number_t

10.4.1.1 describes the

UART number.

10.4.1.2 definitions

```
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 member

Member name	description
UART_DEVICE_1	UART 1
UART_DEVICE_2	UART 2
UART_DEVICE_3	UART 3

10.4.2 uart_bitwidth_t

Description 10.4.2.1

UART data bit.

10.4.2.2 definitions

```
typedef enum _ uart_bitwidth {

    UART_BITWIDTH_5BIT = 0,
  UART_BITWIDTH_6BIT,
  UART_BITWIDTH_7BIT,
  UART_BITWIDTH_8BIT,} uart_bitwidth_t;
```

10.4.2.3 member

Member name	description
UART *BITWIDTH* 5BIT	5-bit UART *BITWIDTH*
6BIT	6-bit UART *BITWIDTH* 7BIT 7-bit
UART *BITWIDTH* 8BIT	8 bits

10.4.3 uart_stopbits_t

10.4.3.1 Description

UART stop bit.

10.4.3.2 definitions

```
typedef enum _ uart_stopbits {

    UART_STOP_1,
  UART_STOP_1_5,
  UART_STOP_2}
  uart_stopbits_t;
```

10.4.3.3 member

Member name	description
UART_STOP_1	1 stop bit
UART_STOP_1_5	1.5 stop bits
UART_STOP_2	2 stop bits

10.4.4 uart_parity_t

Description 10.4.4.1

UART parity bit.

10.4.4.2 definitions

```
typedef enum _ uart_parity {
    UART_PARITY_NONE,
    UART_PARITY_ODD,
    UART_PARITY_EVEN}
    uart_parity_t;
```

10.4.4.3 member

Member name	description
UART_PARITY_NONE	no parity bit
UART_PARITY_ODD	odd parity
UART_PARITY_EVEN	even parity

The first 11 chapter

High speed Universal Asynchronous Receiver Transmitter (UARTHS)

11.1 Overview

Embedded applications often require a small footprint and simple method for system resources to transmit data. High speed Universal Asynchronous Receiver Transmitter (UARTHS) i.e. to meet these requirements, it is possible to flexibly perform full duplex data exchange with external devices. Currently the system uses the serial port As debugging serial port, the serial output printf will call upon.

11.2 Functional Description

UARTHS module has the following features:

- Configuration parameters UARTHS
- To automatically collect data buffer

11.3 API Reference

Corresponding header file uarths.h

To provide users with 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

Initialization UARTHS, the system default baud rate is 115200 8bit 1 stop bit no parity bit. Because uarths clock source PLL0, after setting PLL0 need to call this function to set the baud rate, otherwise it will print garbage.

11.3.1.2 function prototype

```
void uarths_init ( void)
```

11.3.1.3 Parameters None.

11.3.1.4 Return Value None.

11.3.2 uarths_config

11.3.2.1 Description

Setting the parameters UARTHS. The default 8bit data, no parity.

11.3.2.2 function prototype

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

11.3.3 Parameters

Parameter Name	Description	Input	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 UARTHS.

11.3.4.2 function prototype

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

11.3.4.3 Parameters

Parameter Name	Description	input Output
buf	Receive data	Export
Length of the input data received buf_len		

11.3.4.4 return value to the data length of

the received.

11.3.5 uarths_send_data

Description 11.3.5.1 UART

transmit data.

11.3.5.2 function prototype

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

11.3.5.3 Parameters

Parameter Name	Description	input Output
buf	Data to be sent	Entry
Length of the input data to be transmitted buf_len		

11.3.5.4 The return value is the length

of the transmitted data.

11.3.6 uarths_set_irq

11.3.6.1 Description

Set 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 Parameters

parameter name	description	input Output
interrupt_mode	interrupt type	Entry
uarths_callback	interrupt callback function input ctx Input parameters callback function	
priority	Interrupt priority	Entry

11.3.6.4 Return Value None.

11.3.7 uarths_get_interrupt_mode

11.3.7.1 Description

UARTHS get interrupt type. Receive, send or receive simultaneously transmit interrupt.

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 type of the

current interruption.

11.3.8 uarths_set_interrupt_cnt

11.3.8.1 Description

Set FIFO depth of UARTHS interrupted. When the interrupt type UARTHS_SEND_RECEIVE, both transmit and receive FIFO interrupt CNT depth;

11.3.8.2 function prototype

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

11.3.8.3 Parameters

parameter name	description	input Output
interrupt_mode	interrupt type	input
cnt	Input FIFO depth	

11.3.8.4 Return Value None.

11.3.9 Examples

```
/ * Set interrupt reception FIFO depth is 0, i.e. the data received immediately interrupt and read the received data. * /
    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 Types

Data types, data structures are defined as follows:

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

11.4.1 uarths_interrupt_mode_t

11.4.2 Description

UARTHS interrupt type.

11.4.2.1 definitions

```
typedef enum _ uarths_interrupt_mode {

    UARTHS_SEND = 1, UARTHS_RECEIVE =
2, UARTHS_SEND_RECEIVE = 3,}
uarths_interrupt_mode_t;
```

11.4.2.2 member

Member name	description
UARTHS_SEND	Transmit Interrupt
UARTHS_RECEIVE	Receive Interrupt
UARTHS_SEND_RECEIVE	interrupt transmission and reception

11.4.3 uarths_stopbit_t

11.4.3.1 Description:

UARTHS stop bit.

11.4.3.2 definitions

```
typedef enum _ uarths_stopbit {

    UART_STOP_1,
UART_STOP_2}
uarths_stopbit_t;
```

11.4.3.3 member

Member name	description
UART_STOP_1	1 stop bit
UART_STOP_2	2 stop bits

The first 12 chapter

Watchdog Timer (WDT)

12.1 Overview

WDT provides recovery when system error or no response.

12.2 Functional Description

WDT module has the following features:

- Configuring timeout
- Manually restart timing

12.3 API Reference

Corresponding header file wdt.h

To provide users with the following interfaces

- wdt_start
- wdt_stop
- wdt_feed
- wdt_clear_interrupt

12.3.1 wdt_start

12.3.1.1 Description Start

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 Parameters

parameter name	description	input Output
id	Watchdog number	Entry
time_out_ms	timeout (ms)	input
on_irq	Interrupt callback function	input

12.3.1.4 Return None

12.3.2 wdt_stop

Description 12.3.2.1

closed watchdog.

12.3.2.2 function prototype

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

12.3.2.3 Parameters

Parameter Name	Description	input Output
id	Watchdog number	input

12.3.2.4 Return Value None.

12.3.3 wdt_feed

12.3.3.1 describe

dogs.

12.3.3.2 function prototype

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

12.3.3.3 Parameters

Parameter Name	Description	input Output
id	Watchdog number input	

12.3.3.4 Return Value None.

12.3.4 wdt_clear_interrupt

12.3.4.1 Description

Clears the interrupt. If you clear the interrupt 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 Parameters

Parameter Name	Description	input Output
id	Watchdog number input	

12.3.4.4 Return Value None.

12.3.5 Examples

```
After / * Interrupt 2 seconds to enter the watchdog function to print Hello_world, another reset 2s * /
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 Types

Data types, data structures are defined as follows:

- wdt_device_number_t

12.4.1 wdt_device_number_t

12.4.1.1 Description The

Watchdog number.

12.4.1.2 definitions

```
typedef enum _ wdt_device_number {

    WDT_DEVICE_0,
  WDT_DEVICE_1,
  WDT_DEVICE_MAX,}
  wdt_device_number_t;
```

12.4.1.3 member

Member name	description
WDT_DEVICE_0	Watchdog
Watchdog 1	0 WDT_DEVICE_1

The first 13 chapter

Direct memory access controller (DMAC)

13.1 Overview

Direct memory access (Direct Memory Access, DMA) for providing high speed data transfer between the memory and the memory, and in between peripherals and memories. In the case can move without any CPU operation through the DMA data quickly, thereby improving the CPU effectiveness.

13.2 Functional Description

DMA module has the following features:

- Automatically selecting a free way for the DMA channel transmission
- According to the source and destination addresses to automatically select software or hardware handshake protocol
- 1,2,4,8-byte support element size, source and destination do not have the same size
- Asynchronous or synchronous transmission function
- Loop transfer function, commonly used to refresh the screen or audio recorders and other scenes

13.3 API Reference

Corresponding header file dmac.h

To provide users with 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 described

initialization DMA.

13.3.1.2 function prototype

```
void dmac_init ( void)
```

13.3.1.3 Parameters None.

13.3.1.4 Return Value None.

13.3.2 dmac_set_single_mode

13.3.2.1 DMA single parameter set is

described.

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 Parameters

parameter name	description	input Output
channel_num	DMA channel number	Entry
src	source address	Entry
dest	target address	Export
src_inc	Whether the source address increment input	

parameter name	description	input Output
dest_inc	If the destination address increment input	
Transmission burst number dmac_burst_size		Entry
dmac_trans_width single transmission of data bit input block_size		
	The number of input transmission data	

13.3.2.4 Return Value None.

13.3.3 dmac_is_done

13.3.3.1 Description

After determining whether to complete the transfer to start DMAC. After a start DMAC transfer, if it is determined before the start inaccurate.

13.3.3.2 function prototype

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

13.3.3.3 Parameters

parameter name	description	input Output
channel_num DMA channel number input		

13.3.3.4 Return Values

Return Value	Description
0	Unfinished
1	completed

13.3.4 dmac_wait_done

13.3.4.1 describing wait for DMA

completion.

13.3.4.2 function prototype

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

13.3.4.3 Parameters

parameter name	description	input 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 DMAC interrupt callback function

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 Parameters

parameter name	description	input Output
channel_num	DMA channel number	Entry
dmac_callback	interrupt callback function input ctx Input parameters callback function	
priority	Interrupt priority	Entry

13.3.5.4 Return Value None.

13.3.6 dmac_set_src_dest_length

13.3.6.1 Description

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

This function is commonly used in the DMAC interrupt the DMA data transfer to continue, without having to set all the parameters 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 Parameters

parameter name	description	input Output
channel_num	DMA channel number	Entry
src	Interrupt callback function input	
dest	Input parameters callback function	
len	Interrupt priority	Entry

13.3.6.4 Return Value None.

13.3.7 dmac_is_idle

13.3.7.1 Description

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

13.3.7.2 function prototype

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

13.3.7.3 Parameters

parameter name	description	input Output
channel_num	DMA channel number input	

13.3.7.4 Return Values

Return Value	Description
0	Busy
1	idle

13.3.8 dmac_wait_idle

13.3.8.1 Description

DMAC waiting to enter an idle state.

13.3.8.2 Parameters

parameter name	description	input Output
channel_num	DMA channel number	input

13.3.8.3 Return Value None.

13.3.9 Examples

```
/ * I2C int 128 transmits data through 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 Types

Data types, data structures are defined as follows:

- dmac_channel_number_t: DMA channel number.
- dmac_address_increment_t: address growth.
- dmac_burst_trans_length_t: number of burst transfers.
- dmac_transfer_width_t: single transmission of data bits.

13.4.1 dmac_channel_number_t

13.4.1.1 DMA channel number

is described.

13.4.1.2 definitions

```
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 member

Member 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 channels 4
DMAC_CHANNEL5	DMA channel 5

13.4.2 dmac_address_increment_t

13.4.2.1 Description

address growth.

13.4.2.2 definitions

```
typedef enum _ dmac_address_increment {

        DMAC_ADDR_INCREMENT = 0x0,
    DMAC_ADDR_NOCHANGE = 0x1}
    dmac_address_increment_t;
```

13.4.2.3 member

Member name	description
DMAC_ADDR_INCREMENT	address automatically increase
DMAC_ADDR_NOCHANGE	address unchanged

13.4.3 dmac_burst_trans_length_t

13.4.3.1 number of burst

transmission is described.

13.4.3.2 definitions

```
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 member

Member name	description
DMAC_MSIZE_1	Number of single transmission by 1
DMAC_MSIZE_4	Number of single transmission by 4
DMAC_MSIZE_8	Number of single transmission by 8
DMAC_MSIZE_16 single number of transmission by a single number 16 DMAC_MSIZE_32 32 DMAC_MSIZE_64 transmission by a single transmission by the number 64 DMAC_MSIZE_128 single number 128 DMAC_MSIZE_256 single transmission by the number of transmission by 256	

13.4.4 dmac_transfer_width_t

13.4.4.1 describes a single

transmission of data bits.

13.4.4.2 definitions

```
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 member

Member name	description
DMAC_TRANS_WIDTH_8	A single transmission 8
DMAC_TRANS_WIDTH_16	single transmission 16
DMAC_TRANS_WIDTH_32	single transmission 32
DMAC_TRANS_WIDTH_64	single transmission 64
DMAC_TRANS_WIDTH_128	single transmission
DMAC_TRANS_WIDTH_256	a single transmission 128 256

The first 14 chapter

Built-in integrated circuit bus (I²C)

14.1 Overview

I2C bus and a plurality of external devices for communication. A plurality of external devices can share I2C bus.

14.2 Functional Description

I2C module has the following features:

- Peripheral device package independent parameters I2C
- Automatic processing of multi-device bus contention

14.3 API Reference

Corresponding header file i2c.h

To provide users with 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

Configuration from the address register and a bit width I²C rate I²C device.

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 Parameters

parameter name	description	input Output
i2c_num	I²C No.	Entry
slave_address	I²C slave address	Entry
address_width	I²C width register device (7 or 10) input i2c_clk	
	I²C rate (Hz)	Entry

14.3.1.4 Return Value None.

14.3.2 i2c_init_as_slave

14.3.2.1 Configuration I²C slave mode is

described.

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 Parameters

parameter name	description	input Output
i2c_num	I²C No.	Entry
Address slave_address	I²C slave mode,	Entry
address_width	I²C width register device (7 or 10) input	

parameter name	description	input Output
handler	I²C slave mode interrupt handler	Entry

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 Parameters

Parameter Name	Description	input Output
i2c_num	I²C No.	Entry
send_buf	Data to be transmitted	Entry
send_buf_len	length of the input data to be transmitted	

14.3.3.4 Return Values

Return Value	Description
0	success
Non-0	failure

14.3.4 i2c_send_data_dma

Description 14.3.4.1 write

data 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 Parameters

parameter name	description	input Output
dma channel number using the input i2c_num dma_channel_num		
	I²C No.	Entry
send_buf	Data to be transmitted	Entry
send_buf_len	Length of the input data to be transmitted	

14.3.4.4 Return None

14.3.5 i2c_recv_data

14.3.5.1 CPU reads

description data.

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 Parameters

parameter name	description	input Output
i2c_num	I²C bus number	Entry
send_buf	Data to be transmitted, the general situation is i2c peripheral register, if the input is not set to NULL	
send_buf_len	Length of data to be transmitted, if the write is not 0	Entry
receive_buf	Receive data memory	Export
The length of the received data receive_buf_len		Entry

14.3.5.4 Return Values

Return Value	Description
0	success
Non-0	failure

14.3.6 i2c_recv_data_dma

14.3.6.1 described dma

read data.

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 Parameters

parameter name	description	input Output
dma_send_channel_num	Transmitting data using the channel dma	Entry
dma_receive_channel_num reception data using channel dma		Entry
i2c_num	I²C bus number	Entry
send_buf	Data to be transmitted, the general situation is i2c peripheral register, if the input is not set to NULL	
send_buf_len	Length of data to be transmitted, if the write is not 0	Entry
receive_buf	Receive data memory	Export
receive_buf_len	The length of the received data	Entry

14.3.6.4 Return None

14.3.7 Examples

```
/ * I2c peripheral address is 0x32, 7 bit address, a rate of 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 0 to register 0x01 * /
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 the register 0 * /
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 Types

Data types, data structures are defined as follows:

- i2c_device_number_t: i2c number.
- i2c_slave_handler_t: i2c slave mode interrupt handler handler

14.4.1 i2c_device_number_t

14.4.1.1 description i2c

number.

14.4.1.2 definitions

```
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 handles. Performing the corresponding function depending on the operation state of the interrupt.

14.4.2.2 definitions

```
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 member

Member name	description
I2C_DEVICE_0	I2C 0
I2C_DEVICE_1	I2C 1

Member name	description
I2C_DEVICE_2	I2C 2

The first 15 chapter

Serial Peripheral Interface (SPI)

15.1 Overview

SPI is a high-speed, full-duplex, synchronous communication bus.

15.2 Functional Description

SPI module has the following features:

- Peripheral device package separate SPI parameters
- Automatic processing of multi-device bus contention
- Supports standard, two-wire, four-wire, eight-line mode
- Support to read after write and write full-duplex
- Transmitting a series of data to support the same frame used to clear the screen, filling storage sector and other scenes

15.3 API Reference

Corresponding header file spi.h

To provide users with 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_config

15.3.1.1 Description

Setting the SPI mode, multi-line pattern and 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 Parameters

parameter name	description	input Output
spi_num	SPI No.	Entry
work_mode	Four modes polarity of the phase of the input	
frame_format	Multi-line mode	Entry
Data_bit_length bit wide data input single transmission 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

The multi-line mode setting instruction length, address length, the number of wait clocks, the instruction address transmission 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 Parameters

parameter name	description	input Output
spi_num	SPI No.	Entry
instruction_length	The number of bits sent instructions	Entry
address_length	The number of bits transmitted address	Entry
wait_cycles	The number of wait clocks	Entry
instruction_address_trans_mode instruction address transmission mode input		

15.3.2.4 Return None

15.3.3 spi_send_data_standard

15.3.3.1 Description

Standard SPI mode to transmit data.

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 Parameters

parameter name	description	input Output
spi_num	SPI No.	Entry
chip_select chip select signal		Entry
cmd_buff	Peripheral instruction address data is not set to NULL	Entry
cmd_len	Peripheral length instruction address data, no input is set to 0	
tx_buff	Data transmission	Entry
tx_len	The length of the transmission data	Entry

15.3.3.4 Return None

15.3.4 spi_send_data_standard_dma

15.3.4.1 Description

DMA transferred data using the standard SPI 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 Parameters

parameter name	description	input Output
channel_num DMA channel number		Entry
spi_num	SPI No.	Entry
chip_select chip select signal		Entry
cmd_buff	Peripheral instruction address data is not set to NULL	Entry
cmd_len	Peripheral length instruction address data, no input is set to 0	
tx_buff	Data transmission	Entry
tx_len	The length of the transmission data	Entry

15.3.4.4 Return None

15.3.5 spi_receive_data_standard

Receiving data in the standard mode

described 15.3.5.1.

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 Parameters

parameter name	description	input Output
spi_num	SPI No.	Entry
chip_select chip select signal		Entry
cmd_buff	Peripheral instruction address data is not set to NULL	Entry
cmd_len	Peripheral length instruction address data, no input is set to 0	
rx_buff	Receiving data	Export
rx_len	The length of the received data	Entry

15.3.5.4 Return None

15.3.6 spi_receive_data_standard_dma

15.3.6.1 Description

Standard mode to receive data via DMA.

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 Parameters

parameter name	description	input Output
dma_send_channel_num	DMA channel number used for transmission instruction address	Entry
dma_receive_channel_num receive DMA channel number data used		Entry
spi_num	SPI No.	Entry
chip_select	Chip select signal	Entry
cmd_buff	Peripheral instruction address data is not set to NULL	Entry
cmd_len	Peripheral length instruction address data, no input is set to 0	
rx_buff	Receiving data	Export
rx_len	The length of the received data	Entry

15.3.6.4 Return None

15.3.7 spi_send_data_multiple

15.3.7.1 described multi-line mode

data transmission.

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 Parameters

parameter name	description	input Output
spi_num	SPI No.	Entry
chip_select chip select signal		Entry
cmd_buff	Peripheral instruction address data is not set to NULL	Entry
cmd_len	Peripheral length instruction address data, no input is set to 0	
tx_buff	Data transmission	Entry
tx_len	The length of the transmission data	Entry

15.3.7.4 Return None

15.3.8 spi_send_data_multiple_dma

15.3.8.1 Description

DMA transfer mode using multi-line data.

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 Parameters

parameter name	description	input Output
channel_num DMA channel number		Entry

parameter name	description	input Output
spi_num	SPI No.	Entry
chip_select chip select signal		Entry
cmd_buff	Peripheral instruction address data is not set to NULL	Entry
cmd_len	Peripheral length instruction address data, no input is set to 0	
tx_buff	Data transmission	Entry
tx_len	The length of the transmission data	Entry

15.3.8.4 Return None

15.3.9 spi_receive_data_multiple

15.3.9.1 described multi-line mode to

receive data.

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 Parameters

parameter name	description	input Output
spi_num	SPI No.	Entry
chip_select chip select signal		Entry
cmd_buff	Peripheral instruction address data is not set to NULL	Entry
cmd_len	Peripheral length instruction address data, no input is set to 0	
rx_buff	Receiving data	Export
rx_len	The length of the received data	Entry

15.3.9.4 Return None

15.3.10 spi_receive_data_multiple_dma

15.3.10.1 described multi-line mode

received via DMA.

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 parameters

parameter name	description	input Output
dma_send_channel_num	DMA channel number used for transmission instruction address	Entry
dma_receive_channel_num receive DMA channel number data used		Entry
spi_num	SPI No.	Entry
chip_select	Chip select signal	Entry
cmd_buff	Peripheral instruction address data is not set to NULL	Entry
cmd_len	Peripheral length instruction address data, no input is set to 0	
rx_buff	Receiving data	Export
rx_len	The length of the received data	Entry

15.3.10.4 Return None

15.3.11 spi_fill_data_dma

15.3.11.1 Description

By DMA always sends the same data can be used to refresh the 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 parameters

parameter name	description	input Output
channel_num	DMA channel number	Entry
spi_num	SPI No.	Entry
chip_select	chip select signal	Entry
tx_buff	Data transmission, a data transmission only tx_buff this does not automatically increase the input	
tx_len	The length of the transmission data	Entry

15.3.11.4 Return None

15.3.12 spi_send_data_normal_dma

15.3.12.1 Description

Transmitting data via DMA. Do not 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 parameters

parameter name	description	input Output
channel_num	DMA channel number	Entry
spi_num	SPI No.	Entry
chip_select	Chip select signal	Entry
tx_buff	Data transmission, a data transmission only tx_buff this does not automatically increase the input	
tx_len	The length of the transmission data	Entry
Bit wide transmission data spi_transfer_width		Entry

15.3.12.4 Return None

For example 15.3.13

```
/ * Work the SPI0 single transmission mode, 8-bit data in MODE0 standard SPI mode * /
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};
/ * The SPI0 Chip Select 0 0x06 sends an instruction to send the address 0x010204 0, 1, 2, 3 four-byte data * /
spi_send_data_standard (SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 4, data_buf, 4);
/ * Send the SPI0 Chip Select 0 0x06 instruction address 0x010204 receives 4-byte data * /
spi_receive_data_standard (SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 4, data_buf, 4);

/ * Work the SPI0 single transmit 8 bits of data in four-wire mode MODE0 SPI mode * /
spi_init (SPI_DEVICE_0, SPI_WORK_MODE_0, SPI_FF_QUAD, 8, 0);
/ * Length of the eight 32-bit address of the instruction address of the instruction length of the transmission waiting four CLK, command transmission standard SPI mode, address
        Four-wire transmission * /
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};
/ * The SPI0 Chip Select 0 0x06 sends an instruction to send the address 0x010204 0, 1, 2, 3 four-byte data * /
spi_send_data_multiple (SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 2, data_buf, 4);
/ * Send the SPI0 Chip Select 0 0x06 instruction address 0x010204 receives 4-byte data * /
spi_receive_data_multiple (SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 2, data_buf, 4);

/ * Work the SPI0 single transmission mode, 32-bit data in eight-wire SPI mode MODE2 * /
spi_init (SPI_DEVICE_0, SPI_WORK_MODE_2, SPI_FF_OCTAL, 32, 0);
/ * 32-bit address length is no command transmission instruction address after waiting 0 clk, instruction address sent through line 8 * /
spi_init_non_standard (SPI_DEVICE_0, 0, 32, 0, SPI_AITM_AS_FRAME_FORMAT); uint32_t data_buf [256] = {0};

/ * Use the selected DMA channel 0 0 256 transmits data int * /
spi_send_data_normal_dma (DMAC_CHANNEL0, SPI_DEVICE_0, SPI_CHIP_SELECT_0, data_buf, 256,
        SPI_TRANS_INT); uint32_t data =
0x55AA55AA;
/ * DMA Channel 0 is selected from 0 continuous transmission 256 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 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 parameters

Parameter Name	Description	input Output
spi_num	SPI No.	Entry
	SPI clock frequency input target device spi_clk	

15.3.14.4 Return Values

After setting the clock frequency of the SPI device

15.4 Data Type

Data types, 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: SPI transmission mode commands and addresses.

15.4.1 spi_device_num_t

Description 15.4.1.1 SPI

number.

15.4.1.2 definitions

```
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 member

Member Name	Description
SPI *DEVICE* 0	SPI 0 do SPI-based devices *DEVICE* 1 SPI 1 SPI master device to do *DEVICE* 2 SPI 2 as SPI devices from *DEVICE* 3 SPI 3-based devices do

15.4.2 spi_mode_t

15.4.2.1 SPI mode is

described.

15.4.2.2 definitions

```
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 member

Member name	description
SPI_WORK_MODE_0	SPI mode 0
SPI_WORK_MODE_1	SPI mode 1
SPI_WORK_MODE_2	SPI Mode 3 Mode 2
SPI_WORK_MODE_3	SPI

15.4.3 spi_frame_format_t

15.4.3.1 SPI frame format

is described.

15.4.3.2 definitions

```
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 member

Member name	description
SPI_FF_STANDARD	standard

Member name	description
SPI_FF_DUAL	Wire
SPI_FF_QUAD	Four-wire
SPI_FF_OCTAL	Eight-line (SPI3 not supported)

15.4.4 spi_instruction_address_trans_mode_t

15.4.4.1 Description

Transmission mode and SPI instruction address.

15.4.4.2 definitions

```
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 member

Member name	description
SPI_AITM_STANDARD	Use standard frame format
SPI_AITM_ADDR_STANDARD	Values using the configuration instructions, addresses the use of a standard frame format
SPI_AITM_AS_FRAME_FORMAT are configured using the values	

The first 16 chapter

Built-in audio IC bus (I2S)

16.1 Overview

I2S bus standard defines three kinds of signals: the clock signal BCK, channel selection signal WS and the serial data signal SD. A basic I2S data bus has a master and a slave. The role of master and slave remains unchanged in the communication process. I2S module contains independent Transmission and reception channels, to ensure good communication performance.

16.2 Functional Description

I2S module has the following features:

- Automatic configuration apparatus according to the audio format (16,24,32 supports bit depth, sampling rate 44100, 1--4 channel)
- You may be configured to play or record mode
- Automatic management Audio Buffer

16.3 API Reference

Corresponding header file i2s.h

To provide users with 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 sample rate.
- **i2s_set_dma_divide_16: Set dma *divide* When bit data 16, 16 is provided dma *divide* 16, the 32-bit automatic INT32 16-bit data into two left** and right channels of DMA transfer data.
- **i2s_get_dma_divide_16: get dma *divide* 16 value. For determining whether to set dma *divide* 16.**

16.3.1 i2s_init

16.3.1.1 described

initialization 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 Parameters

Member name	description	input Output
device_num	No. I2S	Entry
rxtx_mode	Reception or transmission mode input	
channel_mask	channel mask	Entry

16.3.1.4 Return Value None.

16.3.2 i2s_send_data_dma

16.3.2.1 I2S described

transmission data.

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 Parameters

Member name	description	input Output
device_num	No. I2S	Entry
buf	Transmission data address entered	
buf_len	Data length	Entry
channel_num DMA channel number input		

16.3.2.4 Return Value None.

16.3.3 i2s_recv_data_dma

Description 16.3.3.1 I2S

receive 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 Parameters

Member name	description	input Output
device_num	No. I2S	Entry
buf	Receives the address output data	
buf_len	Data length	Entry
channel_num DMA channel number input		

16.3.3.4 Return None

16.3.4 i2s_rx_channel_config

Description 16.3.4.1 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 Parameters

Member name	description	input Output
device_num	No. I2S	Entry
channel_num	Channel number	Entry
word_length	Receiving data bits	Export
word_select_size single data clock count		Entry
trigger_level	When DMA trigger input FIFO depth	
word_mode	Operating mode	Entry

16.3.4.4 Return Value None.

16.3.5 i2s_tx_channel_config

Set the transmission channel

parameters 16.3.5.1.

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 Parameters

Member name	description	input Output
device_num	No. I2S	Entry
channel_num	Channel number	Entry
word_length	Receiving data bits	Export
word_select_size single data clock count		Entry
trigger_level	When DMA trigger input FIFO depth	
word_mode	Operating mode	Entry

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 Parameters

Member name	description	input Output
device_num	No. I2S	Entry
channel_num	Channel number	Entry
buf	PCM data	Entry
buf_len	PCM input data length	
frame	Number of transmit input single	
bits_per_sample	single sample bit width of input	
track_num	Number of channels	Entry

16.3.6.4 Return Value None.

16.3.7 i2s_set_sample_rate

16.3.7.1 description set the

sample 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 Parameters

Member name	Description Input Output
device_num	No. I2S input
sample_rate	input sampling rate

16.3.7.4 return value of the

actual sampling rate.

16.3.8 i2s_set_dma_divide_16

16.3.8.1 Description

Setting When setting dma_divide_16,16 dma_divide_16 bit data, 32-bit INT32 automatically left and right channel data into two 16-bit data when the 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 Parameters

Member Name	Description	input Output
device_num	number I2S	Entry
enable	0: Disable 1: Enable input	

16.3.8.4 Return Values

Return Value	Description
0	success
Non-0	failure

16.3.9 i2s_get_dma_divide_16

16.3.9.1 Description

Get dma_divide_16 value. For determining whether 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 Parameters

Member Name	Description	Input Output
device_num	number I2S	input

16.3.9.4 Return Values

Return Value	Description
1	Enable
0	Disable
<0	failure

For example 16.3.10

```
/ * I2S0 arranged to receive the channel 0 channel, 16-bit data is received, a single transmission clocks 32, the FIFO depth is 4, the standard mode. Meet
        Group 8 received data * /
/ * I2S2 transmission channel is set to channel 1, 16-bit data transmission, single transmission clocks 32, the FIFO depth is 4, right alignment mode.
        * 8 sets of data transmitted /
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 Types

Data types, data structures are defined as follows:

- i2s_device_number_t: I2S number.
- i2s_channel_num_t: I2S channel number.
- i2s_transmit_t: I2S transmission mode.
- i2s_work_mode_t: I2S mode.
- i2s_word_select_cycles_t: I2S number of clocks per transfer.
- i2s_word_length_t: I2S transmission of data bits.
- i2s_fifo_threshold_t: I2S FIFO depth.

16.4.1 i2s_device_number_t

Description 16.4.1.1 I2S

number.

16.4.1.2 definitions

```
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 member

Member name	description
I2S_DEVICE_0	I2S 0
I2S_DEVICE_1	I2S 1
I2S_DEVICE_2	I2S 2

16.4.2 i2s_channel_num_t

Description 16.4.2.1 I2S

channel number.

16.4.2.2 definitions

```
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 member

Member name	description
I2S_CHANNEL_0	I2S channel 0

Member name	description
I2S_CHANNEL_1	I2S channel 1
I2S_CHANNEL_2	I2S Channel 3 Channel 2 I2S_CHANNEL_3 I2S

16.4.3 i2s_transmit_t

16.4.3.1 I2S described

transmission mode.

16.4.3.2 definitions

```
typedef enum _ i2s_transmit {

    I2S_TRANSMITTER = 0,
I2S_RECEIVER = 1} i2s_transmit_t;
```

16.4.3.3 member

Member name	description
I2S_TRANSMITTER	transmission mode
I2S_RECEIVER	Receive mode

16.4.4 i2s_work_mode_t

Description 16.4.4.1

I2S mode.

16.4.4.2 definitions

```
typedef enum _ i2s_work_mode {

    STANDARD_MODE = 1,
  RIGHT_JUSTIFYING_MODE = 2,
  LEFT_JUSTIFYING_MODE = 4} i2s_work_mode_t;
```

16.4.4.3 member

Member name	description
STANDARD_MODE	Standard Mode
RIGHT_JUSTIFYING_MODE	right-justified left-aligned
mode mode LEFT_JUSTIFYING_MODE	

16.4.5 i2s_word_select_cycles_t

Description 16.4.5.1 I2S single

transmission clock count.

16.4.5.2 definitions

```
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 member

Member name	description
SCLK_CYCLES_16	16 clocks
SCLK_CYCLES_24	24 clocks
SCLK_CYCLES_32	32 clocks

16.4.6 i2s_word_length_t

Description 16.4.6.1 I2S the

transmitted data bits.

16.4.6.2 definitions

```
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 member

Member name	description
IGNORE_WORD_LENGTH	ignore 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 I2S FIFO depth

description.

16.4.7.2 definitions

```
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 member

Member name	description
TRIGGER_LEVEL_1	1 byte FIFO depth
TRIGGER_LEVEL_2	The 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

The first 17 chapter

Timer (TIMER)

17.1 Overview

Chip has three timers, each timer has 4 channels. You may be configured to PWM, PWM detailed description.

17.2 Functional Description

TIMER module has the following features:

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

17.3 API Reference

Corresponding header file timer.h

To provide users with the following interfaces

- timer_init
- timer_set_interval
- timer_set_irq
- timer_set_enable

17.3.1 timer_init

17.3.1.1 description

initialize the timer.

17.3.1.2 function prototype

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

17.3.1.3 Parameters

parameter name	description	input 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

timing 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 Parameters

parameter name	description	input Output
timer_number	timer number	Entry
channel	Timer channel number	input
Interval	nanoseconds (ns)	input

17.3.2.4 returns the value of the actual trigger

interval (ns).

17.3.3 timer_set_irq

17.3.3.1 Description

Set the timer interrupt trigger callback function.

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 Parameters

parameter name	description	input Output
timer_number	timer number	Entry
channel	Timer channel number input	
func	Callback	Entry
priority	Interrupt priority input	

17.3.3.4 Return Value None.

17.3.4 timer_set_enable

17.3.4.1 Description enable

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 Parameters

parameter name	description	input Output
timer_number	timer number	Entry
channel	Timer channel number	Entry
enable	Enable disable Timer 0: Disable 1: Enable input	

17.3.4.4 Return Value None.

17.3.5 Examples

```
/ * Timer 0 channel 0 timing of 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 Types

Data types, data structures are defined as follows:

- timer_device_number_t: Timer number.
- timer_channel_number_t: Timer channel number.

17.4.1 timer_device_number_t

17.4.1.1 Description Timer

number

17.4.1.2 definitions

```
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 member

Member name	description
TIMER_DEVICE_0	Timer 0

Member name	description
TIMER_DEVICE_1	Timer 2 Timer 1
TIMER_DEVICE_2	

17.4.2 timer_channel_number_t

Description 17.4.2.1 timer

channel number.

17.4.2.2 definitions

```
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 member

Member name	description
TIMER_CHANNEL_0	timer channel 0
TIMER_CHANNEL_1	timer channel 1
TIMER_CHANNEL_2	Timer Timer Channel 3
Channel 2 TIMER_CHANNEL_3	

The first 18 chapter

Real-time clock (RTC)

18.1 Overview

Is used to clock RTC unit, the timer function is provided after the set time.

note RTC PLL0 module only when enabled, and CPU frequency greater than 30MHz

18.2 Functional Description

RTC module has the following features:

- Get the current date and time
- Set the current date and time

18.3 API Reference

Corresponding header file rtc.h

To provide users with the following interfaces

- rtc_init
- rtc_timer_set
- rtc_timer_get

18.3.1 rtc_init

18.3.1.1 description

initialize RTC.

18.3.1.2 function prototype

```
int rtc_init ( void)
```

18.3.1.3 Parameters None.

18.3.1.4 Return Values

Return Value	Description
0	success
Non-0	failure

18.3.2 rtc_timer_set

18.3.2.1 Set the 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 Parameters

Parameter Name	Description	Input Output
year	Year input	
month	Enter the month	
day	Day input	
hour	Input	
second	input second input	
minute	minutes	

18.3.2.4 Return None

18.3.3 rtc_timer_get

18.3.3.1 Description Get the

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 Parameters

Parameter Name	Description	Input Output
year	In Output	
month	May output	
day	Daily output	
hour	Output	
minutes seconds minute		
output second output		

18.3.3.4 Return Values

Return Value	Description
0	success
Non-0	failure

18.3.4 Examples

```
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);
```

The first 19 chapter

A pulse width modulator (PWM)

19.1 Overview

PWM duty cycle for controlling a pulse output. Its essence is a timer, so do not pay attention to the PWM and TIMER and channel number when

Timer conflict.

19.2 Functional Description

PWM module has the following features:

- Configuring the PWM output frequency
- Configuration PWM output duty of each pin

19.3 API Reference

The corresponding header file pwm.h

To provide users with the following interfaces

- pwm_init
- pwm_set_frequency
- pwm_set_enable

19.3.1 pwm_init

19.3.1.1 described

initialization PWM.

19.3.1.2 function prototype

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

19.3.1.3 Parameters

Parameter Name	Description	Input Output
pwm_number	pwm signal input	

19.3.1.4 Return Value None.

19.3.2 pwm_set_frequency

19.3.2.1 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 Parameters

Parameter Name	Description	input Output
pwm_number	PWM No.	Entry
channel	PWM input channel number	
frequency	PWM output frequency input	
duty	Duty Cycle	Entry

19.3.2.4 return value of the

actual output frequency.

19.3.3 pwm *set* enable

Description 19.3.3.1 enable

disable 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 Parameters

Parameter Name	Description	input Output
pwm_number	PWM No.	Entry
channel	PWM channel number	Entry
enable	Enable Disable PWM0: Disable 1: Enable input	

19.3.3.4 Return Value None.

19.3.4 Examples

```
/ * Pwm0 channel 1 duty cycle square wave output 200KHZ 0.5 * / / * set the PWM output pin
as the IO13 * /
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 Types

- pwm_device_number_t: pwm number.
- pwm_channel_number_t: pwm channel number.

19.4.1 pwm_device_number_t

19.4.1.1 describe pwm

number.

19.4.1.2 definitions

```
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 member

Member 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 described pwm

channel number.

19.4.2.2 definitions

```
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 member

Member name	description
PWM_CHANNEL_0	PWM channel 0
PWM_CHANNEL_1	PWM channel 1
PWM_CHANNEL_2	PWM Channel 2 Channel 2 PWM_CHANNEL_3 PWM

The first 20 chapter

System Control

20.1 Overview

Configuring the system control module provides the functionality of the operating system.

20.2 Functional Description

The system control module has the following features:

- Set PLL CPU clock frequency.
- Provided each module clock divider value.
- Acquiring the clock frequency of each module.
- Enable, disable, each module is reset.
- Set DMA request source.
- The system can disable interrupts.

20.3 API Reference

Corresponding header file sysctl.h

To provide users with 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. PLL0 is achieved by modifying the frequency.

20.3.1.2 function prototype

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

20.3.1.3 Parameters

Parameter Name	Description	input Output
freq	Frequency (Hz) to set	input

20.3.1.4 returns the actual frequency (Hz)

after setting.

20.3.2 sysctl_set_pll_frequency

20.3.2.1 described PLL

frequency setting.

20.3.2.2 function prototype

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

20.3.2.3 Parameters

Parameter Name	Description	input Output
pll	PLL No.	Entry
Frequency (Hz)	pll_freq to set input	

20.3.2.4 returns 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 Parameters

Parameter Name	Description	input Output
pll	Enter the number of PLL	

20.3.2.7 return Frequency (Hz) corresponding

to the value of the PLL.

20.3.3 sysctl_pll_enable

20.3.3.1 described enable the

corresponding PLL.

20.3.3.2 function prototype

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

20.3.3.3 Parameters

Parameter Name	Description	input Output
pll	Enter the number of PLL	

20.3.3.4 Return Values

Return Value	Description
0	success
Non-0	failure

20.3.4 sysctl_pll_disable

Description 20.3.4.1 disable

the corresponding PLL.

20.3.4.2 function prototype

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

20.3.4.3 Parameters

Parameter Name	Description	input Output
pll	Enter the number of PLL	

20.3.4.4 Return Values

Return Value	Description
0	success
Non-0	failure

20.3.5 sysctl_clock_set_threshold

20.3.5.1 Description

Set value corresponding to the frequency division clock.

20.3.5.2 function prototype

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

20.3.5.3 Parameters

Parameter Name	Description	input Output
which	Setting the clock input	

Parameter Name	Description	input Output
division threshold value		Entry

20.3.5.4 Return Values

Return Value	Description
0	success
Non-0	failure

20.3.6 sysctl_clock_get_threshold

20.3.6.1 Description

Acquiring a corresponding clock divider value.

20.3.6.2 function prototype

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

Parameter Name	Description	Input Output
which	Clock input	

20.3.6.3 return clock division value

corresponding to the value.

20.3.7 sysctl_clock_set_clock_select

20.3.7.1 description set

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 Parameters

Parameter Name	Description	Input/Output
which	Clock input select the input clock source	

20.3.7.4 Return Values

Return Value	Description
0	success
Non-0	failure

20.3.8 sysctl_clock_get_clock_select

20.3.8.1 Description

Get clock corresponding to the clock source.

20.3.8.2 function prototype

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

20.3.8.3 Parameters

Parameter Name	Description	Input/Output
which	Clock input	

20.3.8.4 return value corresponding to

the clock source clock.

20.3.9 sysctl_clock_get_freq

Description 20.3.9.1 acquisition

frequency clock.

20.3.9.2 function prototype

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

20.3.9.3 Parameters

Parameter Name	Description	Input Output
clock	Clock input	

20.3.9.4 return value of the clock

frequency (Hz)

20.3.10 sysctl_clock_enable

20.3.10.1 Description

Enable clock. PLL to use sysctl_pll_enable.

20.3.10.2 function prototype

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

20.3.10.3 parameters

Parameter Name	Description	Input Output
clock	Clock input	

20.3.10.4 Return Values

Return Value	Description
0	success
Non-0	failure

20.3.11 sysctl_clock_disable

20.3.11.1 Description

Disable clock, PLL use sysctl_pll_disable.

20.3.11.2 function prototype

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

20.3.11.3 parameters

Parameter Name	Description	Input Output
clock	Clock input	

20.3.11.4 Return Values

Return Value	Description
0	success
Non-0	failure

20.3.12 sysctl_reset

20.3.12.1 description of each

module is reset.

20.3.12.2 function prototype

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

20.3.12.3 parameters

Parameter Name	Description	input Output
reset	The pre-reset input module	

20.3.12.4 Return Value None.

20.3.13 sysctl_dma_select

20.3.13.1 Description

Set DMA request source. DMAC is used in conjunction with the 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 parameters

Parameter Name	Description	input Output
channel	DMA channel number select input	
	Input DMA request source	

20.3.13.4 Return Values

Return Value	Description
0	success
Non-0	failure

20.3.14 sysctl_set_power_mode

20.3.14.1 Description

Provided corresponding to the power domain voltage FPIOA.

20.3.14.2 prototype

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

20.3.14.3 parameters

parameter name	description	input Output
power_bank	IO power domains Number	Entry
The voltage value of 3.3V or 1.8V io_power_mode set input		

20.3.14.4 Return Value None.

20.3.15 sysctl_enable_irq

20.3.15.1 Description

Enabling interrupt, if the interrupt must open system outage.

20.3.15.2 function prototype

```
void sysctl_enable_irq ( void)
```

20.3.15.3 parameters None.

20.3.15.4 Return Value None.

20.3.16 sysctl_disable_irq

20.3.16.1 describe disable

system interrupts.

20.3.16.2 function prototype

```
void sysctl_disable_irq ( void)
```

20.3.16.3 parameters None.

20.3.16.4 Return Value None.

20.4 Data Types

Data types, data structures are defined as follows:

- sysctl_pll_t: PLL number.
- sysctl_threshold_t: Set each of the frequency dividing module ID value.
- sysctl_clock_select_t: each module ID clock source settings.
- sysctl_clock_t: number of each module.
- sysctl_reset_t: number of each module reset.
- sysctl_dma_channel_t: DMA channel number.
- sysctl_dma_select_t: DMA request source ID.
- sysctl_power_bank_t: power domain number.
- sysctl_io_power_mode_t: IO output voltage value.

20.4.1 sysctl_pll_t

20.4.1.1 described PLL

number.

20.4.1.2 definitions

```
typedef enum _ sysctl_pll_t {

    SYSCTL_PLL0,
SYSCTL_PLL1,
SYSCTL_PLL2,
SYSCTL_PLL_MAX}
sysctl_pll_t;
```

20.4.1.3 member

Member name	description
SYSCTL_PLL0	PLL0
SYSCTL_PLL1	PLL1
SYSCTL_PLL2	PLL2

20.4.2 sysctl_threshold_t

20.4.2.1 Description

Each frequency division value setting module number.

20.4.2.2 definitions

```
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 member

Member 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	

Member name	description
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

Each clock source settings Module Number.

20.4.3.2 definitions

```
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 member

Member 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

Member name	description
SYSCTL_CLOCK_SELECT_TIMER2	TIMER2
SYSCTL_CLOCK_SELECT_SPI3_SAMPLE	SPI3 selected data sampling clock edge

20.4.4 sysctl_clock_t

Description 20.4.4.1 number of

each module.

20.4.4.2 definitions

```
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 member

Member name	description
SYSCTL_CLOCK_PLL0	PLL0
SYSCTL_CLOCK_PLL1	PLL1
SYSCTL_CLOCK_PLL2	PLL2
SYSCTL_CLOCK_CPU	CPU
SYSCTL_CLOCK_SRAM0	SRAM0
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

Member name	description
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
SYSCTL_CLOCK_IN0	External clock input IN0

20.4.5 sysctl_reset_t

20.4.5.1 Description

Reset number of each module.

20.4.5.2 definitions

```
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 member

Member name	description
SYSCTL_RESET_SOC	Chip reset
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

Member name	description
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 DMA channel

number is described.

20.4.6.2 definitions

```
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 member

Member 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 channels 4
SYSCTL_DMA_CHANNEL_5	DMA channel 5

20.4.7 sysctl_dma_select_t

20.4.7.1 DMA request source

number is described.

20.4.7.2 definitions

```
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 member

Member name	description
SYSCTL_DMA_SELECT_SSI0_RX_REQ	SPI0 reception
SYSCTL_DMA_SELECT_SSI0_TX_REQ	Send SPI0
SYSCTL_DMA_SELECT_SSI1_RX_REQ	SPI1 reception
SYSCTL_DMA_SELECT_SSI1_TX_REQ	Send SPI1
SYSCTL_DMA_SELECT_SSI2_RX_REQ	SPI2 reception
SYSCTL_DMA_SELECT_SSI2_TX_REQ	Send SPI2

Member name	description
SYSCTL_DMA_SELECT_SSI3_RX_REQ	SPI3 reception
SYSCTL_DMA_SELECT_SSI3_TX_REQ	Send SPI3
SYSCTL_DMA_SELECT_I2C0_RX_REQ	I2C0 reception
SYSCTL_DMA_SELECT_I2C0_TX_REQ	Send I2C0
SYSCTL_DMA_SELECT_I2C1_RX_REQ	I2C1 reception
SYSCTL_DMA_SELECT_I2C1_TX_REQ	Send I2C1
SYSCTL_DMA_SELECT_I2C2_RX_REQ	I2C2 reception
SYSCTL_DMA_SELECT_I2C2_TX_REQ	Send I2C2
SYSCTL_DMA_SELECT_UART1_RX_REQ UART1 receiving	
SYSCTL_DMA_SELECT_UART1_TX_REQ UART1 reception	
SYSCTL_DMA_SELECT_UART2_TX_REQ UART2 transmit	
SYSCTL_DMA_SELECT_UART2_RX_REQ UART2 transmit	
SYSCTL_DMA_SELECT_UART3_RX_REQ UART3 receiving	
SYSCTL_DMA_SELECT_UART3_TX_REQ UART3 transmitted	
SYSCTL_DMA_SELECT_AES_REQ	AES
SYSCTL_DMA_SELECT_SHA_RX_REQ	SHA reception
SYSCTL_DMA_SELECT_AI_RX_REQ	AI reception
SYSCTL_DMA_SELECT_FFT_RX_REQ	FFT reception
SYSCTL_DMA_SELECT_FFT_TX_REQ	Send FFT
SYSCTL_DMA_SELECT_I2S0_TX_REQ	Send I2S0
SYSCTL_DMA_SELECT_I2S0_RX_REQ	I2S0 reception
SYSCTL_DMA_SELECT_I2S1_TX_REQ	Send I2S1
SYSCTL_DMA_SELECT_I2S1_RX_REQ	I2S1 reception
SYSCTL_DMA_SELECT_I2S2_TX_REQ	Send I2S2
SYSCTL_DMA_SELECT_I2S2_RX_REQ	I2S2 reception

20.4.8 sysctl_power_bank_t

20.4.8.1 described power

domain number.

20.4.8.2 definitions

```
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 member

Member name	description
SYSCTL_POWER_BANK0	power domain 0, the control IO0-IO5
SYSCTL_POWER_BANK1	power domain 0, the control IO6-IO11
SYSCTL_POWER_BANK2	power domain 0, the control IO12-IO17
SYSCTL_POWER_BANK3	power domain 0, the control IO18-IO23
SYSCTL_POWER_BANK4	power domain 0, the control IO24-IO29
SYSCTL_POWER_BANK5	power domain 0, control IO30-IO35
SYSCTL_POWER_BANK6	power domain 0, IO36-IO41
SYSCTL_POWER_BANK7	power control field 0, the control IO42-IO47

20.4.9 sysctl_io_power_mode_t

20.4.9.1 IO described output

voltage value.

20.4.9.2 definitions

```
typedef enum _ sysctl_io_power_mode {

    SYSCTL_POWER_V33,
  SYSCTL_POWER_V18}
  sysctl_io_power_mode_t;
```

20.4.9.3 member

Member name	description
SYSCTL_POWER_V33	set to 1.8V to 3.3V
SYSCTL_POWER_V18	

The first twenty one chapter

Platform-dependent (BSP)

21.1 Overview

The relevant generic function platform.

21.2 Functional Description

Providing access to the currently running program CPU core number of interface and start a second nuclear entrance.

21.3 API Reference

Corresponding header file bsp.h

To provide users with the following interfaces

- register_core1
- current_coreid

21.3.1 register_core1

21.3.1.1 Description

Sign up to 1 nuclear function, and starts a nucleus.

21.3.1.2 function prototype

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

21.3.1.3 Parameters

Parameter Name	Description	input Output
func	1 nuclear function to register	Entry
ctx	Function parameters, input is not set to NULL	

21.3.1.4 Return Values

Return Value	Description
0	success
Non-0	failure

21.3.2 current_coreid

21.3.2.1 Description Get the current CPU

core number.

21.3.2.2 function prototype

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

21.3.2.3 Parameters None.

21.3.2.4 returns the number of the current

value where CPU core.

21.3.3 Examples

```
/ * 0 to obtain the nuclear CPU core number print Hello world, and then start acquiring nuclear CPU core number 1 print 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 Types

Data types, data structures are defined as follows:

- core_function: CPU core function call.

21.4.1 core_function

21.4.1.1 Description CPU core

function call.

21.4.1.2 definitions

```
typedef int ( * Core_function) ( void * Ctx);
```