K210
Standalone SDK 編程指南

關於本手冊

本文檔為用戶提供基於 Kendryte K210 Standalone SDK 開發時的編程指南.

對應 SDK 版本

Kendryte Standalone SDK v0.5.3 (*7ba017918e595714eeeb92ce5695db90e59b8950*)

發佈說明

日期	版本	發佈說明
2018-10-10	V0.1.0	初始版本
2018-10-20	V0.2.0	發佈對應 Standalone SDK v0.4.0 的文檔
2018-11-02	V0.3.0	發佈對應 Standalone SDK v0.5.1 的文檔
2019-01-11	V0.4.0	發佈對應 Standalone SDK v0.5.3 的文檔

免責聲明

本文中的資訊，包括參考的 URL 地址，如有變更，恕不另行通知。文檔“按現狀”提供，不負任何擔保責任，包括對適銷性、適用於特定用途或非侵權性的任何擔保，和任何提案、規格或樣品在他處提到的任何擔保。本文檔不負任何責任，包括使用本文檔內資訊產生的侵犯任何專利權行為的責任。本文檔在此未以禁止反言或其他方式授予任何知識產權使用許可，不管是明示許可還是暗示許可。文中提到的所有商標名稱、商標和註冊商標均屬其各自所有者的財產，特此聲明。

版權公告

目錄

第 1 章

神經網路處理器（KPU）

1.1 概述

KPU 是通用的神經網路處理器，它可以在低功耗的情況下實現捲積神經網路計算，實時獲取被檢測目標的大小、坐標和種類，對人臉或者物體進行檢測和分類。使用 kpu 時，必須結合 model compiler。

1.2 功能描述

KPU 具備以下幾個特點:

- 支持主流訓練框架按照特定限制規則訓練出來的定點化模型
- 對網路層數無直接限制，支持每層捲積神經網路參數單獨配置，包括輸入輸出通道數目、輸入輸出行寬列高
- 支持兩種捲積內核 1x1 和 3x3
- 支持任意形式的激活函數
- 實時工作時最大支持神經網路參數大小為 5.5MiB 到 5.9MiB
- 非實時工作時最大支持網路參數大小為（Flash 容量-軟體體積）

1.3 API 參考

對應的頭文件 kpu.h
為用戶提供以下介面

- kpu_task_init（0.6.0 以後不再支持，請使用 kpu_single_task_init）
- kpu_run（0.6.0 以後不再支持，請使用 kpu_start）
- kpu_get_output_buf（0.6.0 以後不再支持）

- kpu_release_output_buf（0.6.0 以後不再支持）
- kpu_start
- kpu_single_task_init
- kpu_single_task_deinit
- kpu_model_load_from_buffer

1.3.1　kpu_task_init

1.3.1.1　描述

初始化 kpu 任務句柄，該函數具體實現在 model compiler 生成的 gencode_output.c 中。

1.3.1.2　函數定義

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

1.3.1.3　參數

參數名稱	描述	輸入輸出
task	KPU 任務句柄	輸入

1.3.1.4　返回值

KPU 任務句柄。

1.3.2　kpu_run

1.3.2.1　描述

啟動 KPU，進行 AI 運算。

1.3.2.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
task	KPU 任務句柄	輸入
dma_ch	DMA 通道	輸入
src	輸入圖像資料	輸入
dest	運算輸出結果	輸出
callback	運算完成回調函數	輸入

1.3.2.4　返回值

返回值	描述
0	成功
非 0	KPU 忙，失敗

1.3.3　kpu_get_output_buf

1.3.3.1　描述

獲取 KPU 輸出結果的緩存。

1.3.3.2　函數原型

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

1.3.3.3　參數

參數名稱	描述	輸入輸出
task	KPU 任務句柄	輸入

1.3.3.4　返回值

KPU 輸出結果的緩存的指針。

1.3.4　kpu_release_output_buf

1.3.4.1　描述

釋放 KPU 輸出結果緩存。

1.3.4.2　函數原型

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

1.3.4.3　參數

參數名稱	描述	輸入輸出
output_buf	KPU 輸出結果緩存	輸入

1.3.4.4　返回值

無

1.3.5　kpu_start

1.3.5.1　描述

啟動 KPU，進行 AI 運算。

1.3.5.2　函數原型

```
int kpu_start(kpu_task_t *task)
```

1.3.5.3　參數

參數名稱	描述	輸入輸出
task	KPU 任務句柄	輸入

1.3.5.4　返回值

返回值	描述
0	成功
非 0	KPU 忙，失敗

1.3.6　kpu_single_task_init

1.3.6.1　描述

初始化 kpu 任務句柄。

1.3.6.2　函數原型

```
int kpu_single_task_init(kpu_task_t *task)
```

1.3.6.3　參數

參數名稱	描述	輸入輸出
task	KPU 任務句柄	輸入

1.3.6.4　返回值

返回值	描述
0	成功
非 0	失敗

1.3.7　kpu_single_task_deinit

1.3.7.1　描述

註銷 kpu 任務。

1.3.7.2　函數原型

```
int kpu_single_task_deinit(kpu_task_t *task)
```

1.3.7.3　參數

參數名稱	描述	輸入輸出
task	KPU 任務句柄	輸入

1.3.7.4　返回值

返回值	描述
0	成功
非 0	失敗

1.3.8　kpu_model_load_from_buffer

1.3.8.1　描述

解析 kmodel 並初始化 kpu 句柄。

1.3.8.2　函數原型

```
int kpu_model_load_from_buffer(kpu_task_t *task, uint8_t *buffer,
    kpu_model_layer_metadata_t **meta);
```

1.3.8.3　參數

參數名稱	描述	輸入輸出
task	KPU 任務句柄	輸入
buffer	kmodel 資料	輸入
meta	內部測試資料，用戶設置為 NULL	輸出

1.3.8.4　返回值

返回值	描述
0	成功
非 0	失敗

1.3.9　舉例

```
/* 通過MC生成kpu_task_gencode_output_init,設置源資料為g_ai_buf,使用DMA5,kpu完成後調
    用ai_done函數 */
kpu_task_t task;
volatile uint8_t g_ai_done_flag;
static int ai_done(void *ctx)
{
    g_ai_done_flag = 1;
    return 0;
```

```
}

/* 初始化kpu */
kpu_task_gencode_output_init(&task); /* MC生成的函數 */
task.src = g_ai_buf;
task.dma_ch = 5;
task.callback = ai_done;
kpu_single_task_init(&task);

/* 啟動kpu */
kpu_start(&task);
```

1.4　資料類型

相關資料類型、資料結構定義如下：

- kpu_task_t：kpu 任務結構體。

1.4.1　kpu_task_t

1.4.1.1　描述

kpu 任務結構體。

1.4.1.2　定義

```
typedef struct
{
    kpu_layer_argument_t *layers;
    kpu_layer_argument_t *remain_layers;
    plic_irq_callback_t callback;
    void *ctx;
    uint64_t *src;
    uint64_t *dst;
    uint32_t src_length;
    uint32_t dst_length;
    uint32_t layers_length;
    uint32_t remain_layers_length;
    dmac_channel_number_t dma_ch;
    uint32_t eight_bit_mode;
    float output_scale;
    float output_bias;
    float input_scale;
    float input_bias;
} kpu_task_t;
```

1.4.1.3 成員

成員名稱	描述
layers	KPU 參數指針 (MC 初始化，用戶不必關心)
remain_layers	KPU 參數指針（運算過程中使用，用戶不必關心）
callback	運算完成回調函數（需要用戶設置）
ctx	回調函數的參數（非空需要用戶設置）
src	運算源資料（需要用戶設置）
dst	運算結果輸出指針（KPU 初始化賦值，用戶不必關心）
src_length	源資料長度 (MC 初始化，用戶不必關心)
dst_length	運算結果長度 (MC 初始化，用戶不必關心)
layers_length	層數 (MC 初始化，用戶不必關心)
remain_layers_length	剩餘層數（運算過程中使用，用戶不必關心）
dma_ch	使用的 DMA 通道號（需要用戶設置）
eight_bit_mode	是否是 8 比特模式 (MC 初始化，用戶不必關心)
output_scale	輸出 scale 值 (MC 初始化，用戶不必關心)
output_bias	輸出 bias 值 (MC 初始化，用戶不必關心)
input_scale	輸入 scale 值 (MC 初始化，用戶不必關心)
input_bias	輸入 bias 值 (MC 初始化，用戶不必關心)

第 2 章

高級加密加速器 (AES)

2.1 功能描述

K210 內置 AES(高級加密加速器)，相對於軟體可以極大的提高 AES 運算速度。AES 加速器支持多種加密/解密模式 (ECB,CBC,GCM)，多種長度的 KEY(128,192,256) 的運算。

2.2 API 參考

對應的頭文件 aes.h

為用戶提供以下介面

- 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　描述

AES-ECB-128 加密運算。輸入輸出資料都使用 cpu 傳輸。ECB 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。ECB 模式沒有用到向量。

2.2.1.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
input_key	AES-ECB-128 加密的密鑰	輸入
input_data	AES-ECB-128 待加密的明文資料	輸入
input_len	AES-ECB-128 待加密明文資料的長度	輸入
output_data	AES-ECB-128 加密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.1.4　返回值

無。

2.2.2　aes_ecb128_hard_decrypt

2.2.2.1　描述

AES-ECB-128 解密運算。輸入輸出資料都使用 cpu 傳輸。ECB 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。ECB 模式沒有用到向量。

2.2.2.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
input_key	AES-ECB-128 解密的密鑰	輸入
input_data	AES-ECB-128 待解密的密文資料	輸入
input_len	AES-ECB-128 待解密密文資料的長度	輸入

參數名稱	描述	輸入輸出
output_data	AES-ECB-128 解密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.2.4　返回值

無。

2.2.3　aes_ecb192_hard_encrypt

2.2.3.1　描述

AES-ECB-192 加密運算。輸入輸出資料都使用 cpu 傳輸。ECB 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。ECB 模式沒有用到向量。

2.2.3.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
input_key	AES-ECB-192 加密的密鑰	輸入
input_data	AES-ECB-192 待加密的明文資料	輸入
input_len	AES-ECB-192 待加密明文資料的長度	輸入
output_data	AES-ECB-192 加密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.3.4　返回值

無。

2.2.4　aes_ecb192_hard_decrypt

2.2.4.1　描述

AES-ECB-192 解密運算。輸入輸出資料都使用 cpu 傳輸。ECB 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。ECB 模式沒有用到向量。

2.2.4.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
input_key	AES-ECB-192 解密的密鑰	輸入
input_data	AES-ECB-192 待解密的密文資料	輸入
input_len	AES-ECB-192 待解密密文資料的長度	輸入
output_data	AES-ECB-192 解密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.4.4　返回值

無。

2.2.5　aes_ecb256_hard_encrypt

2.2.5.1　描述

AES-ECB-256 加密運算。輸入輸出資料都使用 cpu 傳輸。ECB 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。ECB 模式沒有用到向量。

2.2.5.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
input_key	AES-ECB-256 加密的密鑰	輸入
input_data	AES-ECB-256 待加密的明文資料	輸入
input_len	AES-ECB-256 待加密明文資料的長度	輸入
output_data	AES-ECB-256 加密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.5.4　返回值

無。

2.2.6　aes_ecb256_hard_decrypt

2.2.6.1　描述

AES-ECB-256 解密運算。輸入輸出資料都使用 cpu 傳輸。ECB 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。ECB 模式沒有用到向量。

2.2.6.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
input_key	AES-ECB-256 解密的密鑰	輸入
input_data	AES-ECB-256 待解密的密文資料	輸入

參數名稱	描述	輸入輸出
input_len	AES-ECB-256 待解密密文資料的長度	輸入
output_data	AES-ECB-256 解密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.6.4　返回值

無。

2.2.7　aes_cbc128_hard_encrypt

2.2.7.1　描述

AES-CBC-128 加密運算。輸入輸出資料都使用 cpu 傳輸。CBC 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。

2.2.7.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
context	AES-CBC-128 加密計算的結構體，包含加密密鑰與偏移向量	輸入
input_data	AES-CBC-128 待加密的明文資料	輸入
input_len	AES-CBC-128 待加密明文資料的長度	輸入
output_data	AES-CBC-128 加密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.7.4　返回值

無。

2.2.8　aes_cbc128_hard_decrypt

2.2.8.1　描述

AES-CBC-128 解密運算。輸入輸出資料都使用 cpu 傳輸。CBC 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。

2.2.8.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
context	AES-CBC-128 解密計算的結構體，包含解密密鑰與偏移向量	輸入
input_data	AES-CBC-128 待解密的密文資料	輸入
input_len	AES-CBC-128 待解密密文資料的長度	輸入
output_data	AES-CBC-128 解密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.8.4　返回值

無。

2.2.9　aes_cbc192_hard_encrypt

2.2.9.1　描述

AES-CBC-192 加密運算。輸入輸出資料都使用 cpu 傳輸。CBC 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。

2.2.9.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
context	AES-CBC-192 加密計算的結構體，包含加密密鑰與偏移向量	輸入
input_data	AES-CBC-192 待加密的明文資料	輸入
input_len	AES-CBC-192 待加密明文資料的長度	輸入
output_data	AES-CBC-192 加密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.9.4　返回值

無。

2.2.10　aes_cbc192_hard_decrypt

2.2.10.1　描述

AES-CBC-192 解密運算。輸入輸出資料都使用 cpu 傳輸。CBC 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。

2.2.10.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
context	AES-CBC-192 解密計算的結構體，包含解密密鑰與偏移向量	輸入
input_data	AES-CBC-192 待解密的密文資料	輸入
input_len	AES-CBC-192 待解密密文資料的長度	輸入
output_data	AES-CBC-192 解密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.10.4　返回值

無。

2.2.11　aes_cbc256_hard_encrypt

2.2.11.1　描述

AES-CBC-256 加密運算。輸入輸出資料都使用 cpu 傳輸。CBC 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。

2.2.11.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
context	AES-CBC-256 加密計算的結構體，包含加密密鑰與偏移向量	輸入
input_data	AES-CBC-256 待加密的明文資料	輸入
input_len	AES-CBC-256 待加密明文資料的長度	輸入

參數名稱	描述	輸入輸出
output_data	AES-CBC-256 加密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.11.4　返回值

無。

2.2.12　aes_cbc256_hard_decrypt

2.2.12.1　描述

AES-CBC-256 解密運算。輸入輸出資料都使用 cpu 傳輸。CBC 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。

2.2.12.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
context	AES-CBC-256 解密計算的結構體，包含解密密鑰與偏移向量	輸入
input_data	AES-CBC-256 待解密的密文資料	輸入
input_len	AES-CBC-256 待解密密文資料的長度	輸入
output_data	AES-CBC-256 解密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.12.4　返回值

無。

2.2.13　aes_gcm128_hard_encrypt

2.2.13.1　描述

AES-GCM-128 加密運算。輸入輸出資料都使用 cpu 傳輸。

2.2.13.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
context	AES-GCM-128 加密計算的結構體，包含加密密鑰/偏移向量/aad/aad 長度	輸入
input_data	AES-GCM-128 待加密的明文資料	輸入
input_len	AES-GCM-128 待加密明文資料的長度	輸入
output_data	AES-GCM-128 加密運算後的結果存放在這個 buffer	輸出
gcm_tag	AES-GCM-128 加密運算後的 tag 存放在這個 buffer。這個 buffer 大小需要保證為 16bytes	輸出

2.2.13.4　返回值

無。

2.2.14　aes_gcm128_hard_decrypt

2.2.14.1　描述

AES-GCM-128 解密運算。輸入輸出資料都使用 cpu 傳輸。

2.2.14.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
context	AES-GCM-128 解密計算的結構體，包含解密密鑰/偏移向量/aad/aad 長度	輸入
input_data	AES-GCM-128 待解密的密文資料	輸入
input_len	AES-GCM-128 待解密密文資料的長度	輸入
output_data	AES-GCM-128 解密運算後的結果存放在這個 buffer	輸出
gcm_tag	AES-GCM-128 解密運算後的 tag 存放在這個 buffer。這個 buffer 大小需要保證為 16bytes	輸出

2.2.14.4　返回值

無。

2.2.15　aes_gcm192_hard_encrypt

2.2.15.1　描述

AES-GCM-192 加密運算。輸入輸出資料都使用 cpu 傳輸。

2.2.15.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
context	AES-GCM-192 加密計算的結構體，包含加密密鑰/偏移向量/aad/aad 長度	輸入
input_data	AES-GCM-192 待加密的明文資料	輸入
input_len	AES-GCM-192 待加密明文資料的長度	輸入
output_data	AES-GCM-192 加密運算後的結果存放在這個 buffer	輸出
gcm_tag	AES-GCM-192 加密運算後的 tag 存放在這個 buffer。這個 buffer 大小需要保證為 16bytes	輸出

2.2.15.4　返回值

無。

2.2.16　aes_gcm192_hard_decrypt

2.2.16.1　描述

AES-GCM-192 解密運算。輸入輸出資料都使用 cpu 傳輸。

2.2.16.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
context	AES-GCM-192 解密計算的結構體，包含解密密鑰/偏移向量/aad/aad 長度	輸入
input_data	AES-GCM-192 待解密的密文資料	輸入
input_len	AES-GCM-192 待解密密文資料的長度	輸入
output_data	AES-GCM-192 解密運算後的結果存放在這個 buffer	輸出
gcm_tag	AES-GCM-192 解密運算後的 tag 存放在這個 buffer。這個 buffer 大小需要保證為 16bytes	輸出

2.2.16.4　返回值

無。

2.2.17　aes_gcm256_hard_encrypt

2.2.17.1　描述

AES-GCM-256 加密運算。輸入輸出資料都使用 cpu 傳輸。

2.2.17.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
context	AES-GCM-256 加密計算的結構體，包含加密密鑰/偏移向量/aad/aad 長度	輸入
input_data	AES-GCM-256 待加密的明文資料	輸入
input_len	AES-GCM-256 待加密明文資料的長度	輸入
output_data	AES-GCM-256 加密運算後的結果存放在這個 buffer	輸出
gcm_tag	AES-GCM-256 加密運算後的 tag 存放在這個 buffer。這個 buffer 大小需要保證為 16bytes	輸出

2.2.17.4　返回值

無。

2.2.18　aes_gcm256_hard_decrypt

2.2.18.1　描述

AES-GCM-256 解密運算。輸入輸出資料都使用 cpu 傳輸。

2.2.18.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
context	AES-GCM-256 解密計算的結構體，包含解密密鑰/偏移向量/aad/aad 長度	輸入
input_data	AES-GCM-256 待解密的密文資料	輸入
input_len	AES-GCM-256 待解密密文資料的長度	輸入
output_data	AES-GCM-256 解密運算後的結果存放在這個 buffer	輸出
gcm_tag	AES-GCM-256 解密運算後的 tag 存放在這個 buffer。這個 buffer 大小需要保證為 16bytes	輸出

2.2.18.4　返回值

無。

2.2.19　aes_ecb128_hard_encrypt_dma

2.2.19.1　描述

AES-ECB-128 加密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。ECB 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。ECB 模式沒有用到向量。

2.2.19.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
input_key	AES-ECB-128 加密的密鑰	輸入
input_data	AES-ECB-128 待加密的明文資料	輸入
input_len	AES-ECB-128 待加密明文資料的長度	輸入
output_data	AES-ECB-128 加密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.19.4　返回值

無。

2.2.20　aes_ecb128_hard_decrypt_dma

2.2.20.1　描述

AES-ECB-128 解密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。ECB 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。ECB 模式沒有用到向量。

2.2.20.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
input_key	AES-ECB-128 解密的密鑰	輸入
input_data	AES-ECB-128 待解密的密文資料	輸入
input_len	AES-ECB-128 待解密密文資料的長度	輸入

參數名稱	描述	輸入輸出
output_data	AES-ECB-128 解密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.20.4　返回值

無。

2.2.21　aes_ecb192_hard_encrypt_dma

2.2.21.1　描述

AES-ECB-192 加密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。ECB 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。ECB 模式沒有用到向量。

2.2.21.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
input_key	AES-ECB-192 加密的密鑰	輸入
input_data	AES-ECB-192 待加密的明文資料	輸入
input_len	AES-ECB-192 待加密明文資料的長度	輸入
output_data	AES-ECB-192 加密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.21.4　返回值

無。

2.2.22　aes_ecb192_hard_decrypt_dma

2.2.22.1　描述

AES-ECB-192 解密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。ECB 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。ECB 模式沒有用到向量。

2.2.22.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
input_key	AES-ECB-192 解密的密鑰	輸入
input_data	AES-ECB-192 待解密的密文資料	輸入
input_len	AES-ECB-192 待解密密文資料的長度	輸入
output_data	AES-ECB-192 解密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.22.4　返回值

無。

2.2.23　aes_ecb256_hard_encrypt_dma

2.2.23.1　描述

AES-ECB-256 加密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。ECB 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。ECB 模式沒有用到向量。

2.2.23.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
input_key	AES-ECB-256 加密的密鑰	輸入
input_data	AES-ECB-256 待加密的明文資料	輸入
input_len	AES-ECB-256 待加密明文資料的長度	輸入
output_data	AES-ECB-256 加密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.23.4　返回值

無。

2.2.24　aes_ecb256_hard_decrypt_dma

2.2.24.1　描述

AES-ECB-256 解密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。ECB 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。ECB 模式沒有用到向量。

2.2.24.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
input_key	AES-ECB-256 解密的密鑰	輸入

參數名稱	描述	輸入輸出
input_data	AES-ECB-256 待解密的密文資料	輸入
input_len	AES-ECB-256 待解密密文資料的長度	輸入
output_data	AES-ECB-256 解密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.24.4　返回值

無。

2.2.25　aes_cbc128_hard_encrypt_dma

2.2.25.1　描述

AES-CBC-128 加密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。CBC 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。

2.2.25.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
context	AES-CBC-128 加密計算的結構體，包含加密密鑰與偏移向量	輸入
input_data	AES-CBC-128 待加密的明文資料	輸入
input_len	AES-CBC-128 待加密明文資料的長度	輸入

參數名稱	描述	輸入輸出
output_data	AES-CBC-128 加密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.25.4　返回值

無。

2.2.26　aes_cbc128_hard_decrypt_dma

2.2.26.1　描述

AES-CBC-128 解密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。CBC 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。

2.2.26.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
context	AES-CBC-128 解密計算的結構體，包含解密密鑰與偏移向量	輸入
input_data	AES-CBC-128 待解密的密文資料	輸入
input_len	AES-CBC-128 待解密密文資料的長度	輸入
output_data	AES-CBC-128 解密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.26.4　返回值

無。

2.2.27　aes_cbc192_hard_encrypt_dma

2.2.27.1　描述

AES-CBC-192 加密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。CBC 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。

2.2.27.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
context	AES-CBC-192 加密計算的結構體，包含加密密鑰與偏移向量	輸入
input_data	AES-CBC-192 待加密的明文資料	輸入
input_len	AES-CBC-192 待加密明文資料的長度	輸入
output_data	AES-CBC-192 加密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.27.4　返回值

無。

2.2.28　aes_cbc192_hard_decrypt_dma

2.2.28.1　描述

AES-CBC-192 解密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。CBC 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。

2.2.28.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
context	AES-CBC-192 解密計算的結構體，包含解密密鑰與偏移向量	輸入
input_data	AES-CBC-192 待解密的密文資料	輸入
input_len	AES-CBC-192 待解密密文資料的長度	輸入
output_data	AES-CBC-192 解密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.28.4　返回值

無。

2.2.29　aes_cbc256_hard_encrypt_dma

2.2.29.1　描述

AES-CBC-256 加密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。CBC 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。

2.2.29.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
context	AES-CBC-256 加密計算的結構體，包含加密密鑰與偏移向量	輸入
input_data	AES-CBC-256 待加密的明文資料	輸入
input_len	AES-CBC-256 待加密明文資料的長度	輸入
output_data	AES-CBC-256 加密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.29.4　返回值

無。

2.2.30　aes_cbc256_hard_decrypt_dma

2.2.30.1　描述

AES-CBC-256 解密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。CBC 加密將明文按照固定大小 16bytes 的塊進行加密的，塊大小不足則進行填充。

2.2.30.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
context	AES-CBC-256 解密計算的結構體，包含解密密鑰與偏移向量	輸入
input_data	AES-CBC-256 待解密的密文資料	輸入
input_len	AES-CBC-256 待解密密文資料的長度	輸入
output_data	AES-CBC-256 解密運算後的結果存放在這個 buffer。這個 buffer 的大小需要保證 16bytes 對齊。	輸出

2.2.30.4　返回值

無。

2.2.31　aes_gcm128_hard_encrypt_dma

2.2.31.1　描述

AES-GCM-128 加密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。

2.2.31.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入

參數名稱	描述	輸入輸出
context	AES-GCM-128 加密計算的結構體，包含加密密鑰/偏移向量/aad/aad 長度	輸入
input_data	AES-GCM-128 待加密的明文資料	輸入
input_len	AES-GCM-128 待加密明文資料的長度。	輸入
output_data	AES-GCM-128 加密運算後的結果存放在這個 buffer。。由於 DMA 搬運資料的最小粒度為 4bytes，所以需要保證這個 buffer 大小至少為 4bytes 的整數倍。	輸出
gcm_tag	AES-GCM-128 加密運算後的 tag 存放在這個 buffer。這個 buffer 大小需要保證為 16bytes	輸出

2.2.31.4　返回值

無。

2.2.32　aes_gcm128_hard_decrypt_dma

2.2.32.1　描述

AES-GCM-128 解密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。

2.2.32.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
context	AES-GCM-128 解密計算的結構體，包含解密密鑰/偏移向量/aad/aad 長度	輸入
input_data	AES-GCM-128 待解密的密文資料	輸入
input_len	AES-GCM-128 待解密密文資料的長度。	輸入
output_data	AES-GCM-128 解密運算後的結果存放在這個 buffer。由於 DMA 搬運資料的最小粒度為 4bytes，所以需要保證這個 buffer 大小至少為 4bytes 的整數倍。	輸出
gcm_tag	AES-GCM-128 解密運算後的 tag 存放在這個 buffer。這個 buffer 大小需要保證為 16bytes	輸出

2.2.32.4　返回值

無。

2.2.33　aes_gcm192_hard_encrypt_dma

2.2.33.1　描述

AES-GCM-192 加密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。

2.2.33.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
context	AES-GCM-192 加密計算的結構體，包含加密密鑰/偏移向量/aad/aad 長度	輸入
input_data	AES-GCM-192 待加密的明文資料	輸入
input_len	AES-GCM-192 待加密明文資料的長度。	輸入
output_data	AES-GCM-192 加密運算後的結果存放在這個 buffer。由於 DMA 搬運資料的最小粒度為 4bytes，所以需要保證這個 buffer 大小至少為 4bytes 的整數倍。	輸出
gcm_tag	AES-GCM-192 加密運算後的 tag 存放在這個 buffer。這個 buffer 大小需要保證為 16bytes	輸出

2.2.33.4　返回值

無。

2.2.34　aes_gcm192_hard_decrypt_dma

2.2.34.1　描述

AES-GCM-192 解密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。

2.2.34.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
context	AES-GCM-192 解密計算的結構體，包含解密密鑰/偏移向量/aad/aad 長度	輸入
input_data	AES-GCM-192 待解密的密文資料	輸入
input_len	AES-GCM-192 待解密密文資料的長度。	輸入
output_data	AES-GCM-192 解密運算後的結果存放在這個 buffer。由於 DMA 搬運資料的最小粒度為 4bytes，所以需要保證這個 buffer 大小至少為 4bytes 的整數倍。	輸出
gcm_tag	AES-GCM-192 解密運算後的 tag 存放在這個 buffer。這個 buffer 大小需要保證為 16bytes	輸出

2.2.34.4　返回值

無。

2.2.35　aes_gcm256_hard_encrypt_dma

2.2.35.1　描述

AES-GCM-256 加密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。

2.2.35.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
context	AES-GCM-256 加密計算的結構體，包含加密密鑰/偏移向量/aad/aad 長度	輸入
input_data	AES-GCM-256 待加密的明文資料	輸入
input_len	AES-GCM-256 待加密明文資料的長度。	輸入
output_data	AES-GCM-256 加密運算後的結果存放在這個 buffer。由於 DMA 搬運資料的最小粒度為 4bytes，所以需要保證這個 buffer 大小至少為 4bytes 的整數倍。	輸出
gcm_tag	AES-GCM-256 加密運算後的 tag 存放在這個 buffer。這個 buffer 大小需要保證為 16bytes	輸出

2.2.35.4　返回值

無。

2.2.36　aes_gcm256_hard_decrypt_dma

2.2.36.1　描述

AES-GCM-256 解密運算。輸入資料使用 cpu 傳輸，輸出資料都使用 dma 傳輸。

2.2.36.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_receive_channel_num	AES 輸出資料的 DMA 通道號	輸入
context	AES-GCM-256 解密計算的結構體，包含解密密鑰/偏移向量/aad/aad 長度	輸入
input_data	AES-GCM-256 待解密的密文資料	輸入
input_len	AES-GCM-256 待解密密文資料的長度。	輸入
output_data	AES-GCM-256 解密運算後的結果存放在這個 buffer。由於 DMA 搬運資料的最小粒度為 4bytes，所以需要保證這個 buffer 大小至少為 4bytes 的整數倍。	輸出
gcm_tag	AES-GCM-256 解密運算後的 tag 存放在這個 buffer。這個 buffer 大小需要保證為 16bytes	輸出

2.2.36.4　返回值

無。

2.2.37　aes_init

2.2.37.1　描述

AES 硬體模組的初始化

2.2.37.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
input_key	待加密/解密的密鑰	輸入
input_key_len	待加密/解密密鑰的長度	輸入
iv	AES 加密解密用到的 iv 資料	輸入
iv_len	AES 加密解密用到的 iv 資料的長度，CBC 固定為 16bytes，GCM 固定為 12bytes	輸出
gcm_aad	AES-GCM 加密解密用到的 aad 資料	輸出
cipher_mode	AES 硬體模組執行的加密解密類型，支持 AES_CBC/AES_ECB/AES_GCM	輸入
encrypt_sel	AES 硬體模組執行的模式：加密或解密	輸入
gcm_aad_len	AES-GCM 加密解密用到的 aad 資料的長度	輸入
input_data_len	待加密/解密的資料長度	輸入

2.2.37.4　返回值

無。

2.2.38　aes_process

2.2.38.1　描述

AES 硬體模組執行加密解密操作

2.2.38.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
input_data	這個 buffer 存放待加密/解密的資料	輸入
output_data	這個 buffer 存放加密/解密的輸出結果	輸出
input_data_len	待加密/解密的資料的長度	輸入
cipher_mode	AES 硬體模組執行的加密解密類型，支持 AES_CBC/AES_ECB/AES_GCM	輸入

2.2.38.4　返回值

無。

2.2.39　gcm_get_tag

2.2.39.1　描述

獲取 AES-GCM 計算結束後的 tag

2.2.39.2　函數原型

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

2.2.39.3　參數

參數名稱	描述	輸入輸出
gcm_tag	這個 buffer 存放 AES-GCM 加密/解密後的 tag，固定為 16bytes 的大小	輸出

2.2.39.4　返回值

無。

2.2.40　舉例

```
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　資料類型

相關資料類型、資料結構定義如下：

- aes_cipher_mode_t：AES 加密/解密的方式。

2.3.1　aes_cipher_mode_t

2.3.1.1　描述

AES 加密/解密的方式。

2.3.1.2　定義

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

- gcm_context_t: AES-GCM 加密/解密時參數用到的結構體

2.3.2　gcm_context_t

2.3.2.1　描述

AES-GCM 參數用到的結構體，包括密鑰、偏移向量、aad 資料、aad 資料長度。

2.3.2.2　定義

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

- cbc_context_t: AES-CBC 加密/解密時參數用到的結構體

2.3.3　cbc_context_t

2.3.3.1　描述

AES-CBC 參數用到的結構體，包括密鑰、偏移向量。

2.3.3.2　定義

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

```
} cbc_context_t;
```

2.3.3.3　成員

成員名稱	描述
AES_ECB	ECB 加密/解密
AES_CBC	CBC 加密/解密
AES_GCM	GCM 加密/解密

第 3 章

中斷 PLIC

3.1 概述

可以將任一外部中斷源單獨分配到每個 CPU 的外部中斷上。這提供了強大的靈活性，能適應不同的應用需求。

3.2 功能描述

PLIC 模組具有以下功能:

- 啟用或禁用中斷
- 設置中斷處理程序
- 配置中斷優先順序

3.3 API 參考

對應頭文件 plic.h
為用戶提供以下介面

- 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　描述

PLIC 初始化外部中斷。

3.3.1.2　函數原型

```
void plic_init(void)
```

3.3.1.3　參數

無。

3.3.1.4　返回值

無。

3.3.2　plic_irq_enable

3.3.2.1　描述

啟動外部中斷。

3.3.2.2　函數原型

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

3.3.2.3　參數

參數名稱	描述	輸入輸出
irq_number	中斷號	輸入

3.3.2.4　返回值

返回值	描述
0	成功
非 0	失敗

3.3.3　plic_irq_disable

3.3.3.1　描述

禁用外部中斷。

3.3.3.2　函數原型

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

3.3.3.3　參數

參數名稱	描述	輸入輸出
irq_number	中斷號	輸入

3.3.3.4　返回值

返回值	描述
0	成功
非 0	失敗

3.3.4　plic_set_priority

3.3.4.1　描述

設置中斷優先順序。

3.3.4.2　函數原型

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

3.3.4.3　參數

參數名稱	描述	輸入輸出
irq_number	中斷號	輸入
priority	中斷優先順序	輸入

3.3.4.4　返回值

返回值	描述
0	成功
非 0	失敗

3.3.5　plic_get_priority

3.3.5.1　描述

獲取中斷優先順序。

3.3.5.2　函數原型

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

3.3.5.3　參數

參數名稱	描述	輸入輸出
irq_number	中斷號	輸入

3.3.5.4　返回值

irq_number 中斷的優先順序。

3.3.6　plic_irq_register

3.3.6.1　描述

註冊外部中斷函數。

3.3.6.2　函數原型

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

3.3.6.3　參數

參數名稱	描述	輸入輸出
irq	中斷號	輸入

參數名稱	描述	輸入輸出
callback	中斷回調函數	輸入
ctx	回調函數的參數	輸入

3.3.6.4　返回值

返回值	描述
0	成功
非 0	失敗

3.3.7　plic_irq_deregister

3.3.7.1　描述

註銷外部中斷函數。

3.3.7.2　函數原型

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

3.3.7.3　參數

參數名稱	描述	輸入輸出
irq	中斷號	輸入

3.3.7.4　返回值

返回值	描述
0	成功
非 0	失敗

3.3.8　舉例

```
/* 設置GPIOHS0的觸發中斷 */
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　資料類型

相關資料類型、資料結構定義如下：

- plic_irq_t：外部中斷號。
- plic_irq_callback_t：外部中斷回調函數。

3.4.1　plic_irq_t

3.4.1.1　描述

外部中斷號。

3.4.1.2　定義

```
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　成員

成員名稱	描述
IRQN_NO_INTERRUPT	不存在
IRQN_SPI0_INTERRUPT	SPI0 中斷
IRQN_SPI1_INTERRUPT	SPI1 中斷
IRQN_SPI_SLAVE_INTERRUPT	從 SPI 中斷
IRQN_SPI3_INTERRUPT	SPI3 中斷
IRQN_I2S0_INTERRUPT	I2S0 中斷
IRQN_I2S1_INTERRUPT	I2S1 中斷
IRQN_I2S2_INTERRUPT	I2S2 中斷
IRQN_I2C0_INTERRUPT	I2C0 中斷
IRQN_I2C1_INTERRUPT	I2C1 中斷
IRQN_I2C2_INTERRUPT	I2C2 中斷
IRQN_UART1_INTERRUPT	UART1 中斷
IRQN_UART2_INTERRUPT	UART2 中斷
IRQN_UART3_INTERRUPT	UART3 中斷
IRQN_TIMER0A_INTERRUPT	TIMER0 通道 0 和 1 中斷
IRQN_TIMER0B_INTERRUPT	TIMER0 通道 2 和 3 中斷
IRQN_TIMER1A_INTERRUPT	TIMER1 通道 0 和 1 中斷
IRQN_TIMER1B_INTERRUPT	TIMER1 通道 2 和 3 中斷
IRQN_TIMER2A_INTERRUPT	TIMER2 通道 0 和 1 中斷
IRQN_TIMER2B_INTERRUPT	TIMER2 通道 2 和 3 中斷
IRQN_RTC_INTERRUPT	RTC 滴答中斷和報警中斷
IRQN_WDT0_INTERRUPT	看門狗 0 中斷
IRQN_WDT1_INTERRUPT	看門狗 1 中斷
IRQN_APB_GPIO_INTERRUPT	普通 GPIO 中斷
IRQN_DVP_INTERRUPT	數位攝像頭（DVP）中斷
IRQN_AI_INTERRUPT	AI 加速器中斷
IRQN_FFT_INTERRUPTFFT	傅立葉加速器中斷
IRQN_DMA0_INTERRUPT	DMA 通道 0 中斷
IRQN_DMA1_INTERRUPT	DMA 通道 1 中斷
IRQN_DMA2_INTERRUPT	DMA 通道 2 中斷
IRQN_DMA3_INTERRUPT	DMA 通道 3 中斷
IRQN_DMA4_INTERRUPT	DMA 通道 4 中斷
IRQN_DMA5_INTERRUPT	DMA 通道 5 中斷
IRQN_UARTHS_INTERRUPT	高速 UART 中斷
IRQN_GPIOHS0_INTERRUPT	高速 GPIO0 中斷

成員名稱	描述
IRQN_GPIOHS1_INTERRUPT	高速 GPIO1 中斷
IRQN_GPIOHS2_INTERRUPT	高速 GPIO2 中斷
IRQN_GPIOHS3_INTERRUPT	高速 GPIO3 中斷
IRQN_GPIOHS4_INTERRUPT	高速 GPIO4 中斷
IRQN_GPIOHS5_INTERRUPT	高速 GPIO5 中斷
IRQN_GPIOHS6_INTERRUPT	高速 GPIO6 中斷
IRQN_GPIOHS7_INTERRUPT	高速 GPIO7 中斷
IRQN_GPIOHS8_INTERRUPT	高速 GPIO8 中斷
IRQN_GPIOHS9_INTERRUPT	高速 GPIO9 中斷
IRQN_GPIOHS10_INTERRUPT	高速 GPIO10 中斷
IRQN_GPIOHS11_INTERRUPT	高速 GPIO11 中斷
IRQN_GPIOHS12_INTERRUPT	高速 GPIO12 中斷
IRQN_GPIOHS13_INTERRUPT	高速 GPIO13 中斷
IRQN_GPIOHS14_INTERRUPT	高速 GPIO14 中斷
IRQN_GPIOHS15_INTERRUPT	高速 GPIO15 中斷
IRQN_GPIOHS16_INTERRUPT	高速 GPIO16 中斷
IRQN_GPIOHS17_INTERRUPT	高速 GPIO17 中斷
IRQN_GPIOHS18_INTERRUPT	高速 GPIO18 中斷
IRQN_GPIOHS19_INTERRUPT	高速 GPIO19 中斷
IRQN_GPIOHS20_INTERRUPT	高速 GPIO20 中斷
IRQN_GPIOHS21_INTERRUPT	高速 GPIO21 中斷
IRQN_GPIOHS22_INTERRUPT	高速 GPIO22 中斷
IRQN_GPIOHS23_INTERRUPT	高速 GPIO23 中斷
IRQN_GPIOHS24_INTERRUPT	高速 GPIO24 中斷
IRQN_GPIOHS25_INTERRUPT	高速 GPIO25 中斷
IRQN_GPIOHS26_INTERRUPT	高速 GPIO26 中斷
IRQN_GPIOHS27_INTERRUPT	高速 GPIO27 中斷
IRQN_GPIOHS28_INTERRUPT	高速 GPIO28 中斷
IRQN_GPIOHS29_INTERRUPT	高速 GPIO29 中斷
IRQN_GPIOHS30_INTERRUPT	高速 GPIO30 中斷
IRQN_GPIOHS31_INTERRUPT	高速 GPIO31 中斷

3.4.2 plic_irq_callback_t

3.4.2.1 描述

外部中斷回調函數。

3.4.2.2　定義

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

第 4 章

通用輸入/輸出（GPIO）

4.1 概述

晶片有 8 個通用 GPIO。

4.2 功能描述

GPIO 模組具有以下功能：

- 可配置上下拉驅動模式

4.3 API 參考

對應的頭文件 `gpio.h`
為用戶提供以下介面

- `gpio_init`
- `gpio_set_drive_mode`
- `gpio_set_pin`
- `gpio_get_pin`

4.3.1 gpio_init

4.3.1.1 描述

初始化 GPIO。

4.3.1.2　函數原型

```
int gpio_init(void)
```

4.3.1.3　返回值

返回值	描述
0	成功
非 0	失敗

4.3.2　gpio_set_drive_mode

4.3.2.1　描述

設置 GPIO 驅動模式。

4.3.2.2　函數原型

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

4.3.2.3　參數

參數名稱	描述	輸入輸出
pin	GPIO 腳位	輸入
mode	GPIO 驅動模式	輸入

4.3.2.4　返回值

無。

4.3.3　gpio_set_pin

4.3.3.1　描述

設置 GPIO 腳位值。

4.3.3.2　函數原型

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

4.3.3.3　參數

參數名稱	描述	輸入輸出
pin	GPIO 腳位	輸入
value	GPIO 值	輸入

4.3.3.4　返回值

無。

4.3.4　gpio_get_pin

4.3.4.1　描述

獲取 GPIO 腳位值。

4.3.4.2　函數原型

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

4.3.4.3　參數

參數名稱	描述	輸入輸出
pin	GPIO 腳位	輸入

4.3.4.4　返回值

獲取的 GPIO 腳位值。

4.3.5　舉例

```
/* 設置IO13為輸出並置為高 */
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　資料類型

相關資料類型、資料結構定義如下：

- gpio_drive_mode_t：GPIO 驅動模式。
- gpio_pin_value_t：GPIO 值。

4.4.1　gpio_drive_mode_t

4.4.1.1　描述

GPIO 驅動模式。

4.4.1.2　定義

```
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　成員

成員名稱	描述
GPIO_DM_INPUT	輸入
GPIO_DM_INPUT_PULL_DOWN	輸入下拉
GPIO_DM_INPUT_PULL_UP	輸入上拉
GPIO_DM_OUTPUT	輸出

4.4.2　gpio_pin_value_t

4.4.2.1　描述

GPIO 值。

4.4.2.2　定義

```
typedef enum _gpio_pin_value
```

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

4.4.2.3　成員

成員名稱	描述
GPIO_PV_LOW	低
GPIO_PV_HIGH	高

第 5 章

通用高速輸入/輸出（GPIOHS）

5.1 概述

晶片有 32 個高速 GPIO。與普通 GPIO 相似，腳位反轉能力更強。

5.2 功能描述

GPIOHS 模組具有以下功能：

- 可配置上下拉驅動模式
- 支持正緣、負緣和雙緣觸發

5.3 API 參考

對應的頭文件 gpiohs.h
為用戶提供以下介面

- gpiohs_set_drive_mode
- gpiohs_set_pin
- gpiohs_get_pin
- gpiohs_set_pin_edge
- gpiohs_set_irq（0.6.0 後不再支持，請使用 gpiohs_irq_register）
- gpiohs_irq_register
- gpiohs_irq_unregister

5.3.1　gpiohs_set_drive_mode

5.3.1.1　描述

設置 GPIO 驅動模式。

5.3.1.2　函數原型

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

5.3.1.3　參數

參數名稱	描述	輸入輸出
pin	GPIO 腳位	輸入
mode	GPIO 驅動模式	輸入

5.3.1.4　返回值

無。

5.3.2　gpio_set_pin

5.3.2.1　描述

設置 GPIO 腳位值。

5.3.2.2　函數原型

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

5.3.2.3　參數

參數名稱	描述	輸入輸出
pin	GPIO 腳位	輸入
value	GPIO 值	輸入

5.3.2.4　返回值

無。

5.3.3　gpio_get_pin

5.3.3.1　描述

獲取 GPIO 腳位值。

5.3.3.2　函數原型

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

5.3.3.3　參數

參數名稱	描述	輸入輸出
pin	GPIO 腳位	輸入

5.3.3.4　返回值

獲取的 GPIO 腳位值。

5.3.4　gpiohs_set_pin_edge

5.3.4.1　描述

設置高速 GPIO 中斷觸發模式。

5.3.4.2　函數原型

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

5.3.4.3　參數

參數名稱	描述	輸入輸出
pin	GPIO 腳位	輸入
edge	中斷觸發方式	輸入

5.3.4.4　返回值

無。

5.3.5　gpiohs_set_irq

5.3.5.1　描述

設置高速 GPIO 的中斷回調函數。

5.3.5.2　函數原型

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

5.3.5.3　參數

參數名稱	描述	輸入輸出
pin	GPIO 腳位	輸入
priority	中斷優先順序	輸入
func	中斷回調函數	輸入

5.3.5.4　返回值

無。

5.3.6　gpiohs_irq_register

5.3.6.1　描述

設置高速 GPIO 的中斷回調函數。

5.3.6.2　函數原型

```
void gpiohs_irq_register(uint8_t pin, uint32_t priority, plic_irq_callback_t callback,
    void *ctx)
```

5.3.6.3　參數

參數名稱	描述	輸入輸出
pin	GPIO 腳位	輸入
priority	中斷優先順序	輸入
plic_irq_callback_t	中斷回調函數	輸入

參數名稱	描述	輸入輸出
ctx	回調函數參數	輸入

5.3.6.4　返回值

無。

5.3.7　gpiohs_irq_unregister

5.3.7.1　描述

註銷 GPIOHS 中斷。

5.3.8　函數原型

```
void gpiohs_irq_unregister(uint8_t pin)
```

5.3.8.1　參數

參數名稱	描述	輸入輸出
pin	GPIO 腳位	輸入

5.3.8.2　返回值

無。

5.3.9　舉例

```
void irq_gpiohs2(void *ctx)
{
    printf("Hello world\n");
}
/* 設置IO13為高速GPIO，輸出模式並置為高 */
fpioa_set_function(13, FUNC_GPIOHS3);
gpiohs_set_drive_mode(3, GPIO_DM_OUTPUT);
gpiohs_set_pin(3, GPIO_PV_High);
/* 設置IO14為高速GPIO 輸入模式 雙緣觸發中斷時列印Hello world */
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_irq_register(2, 1, irq_gpiohs2, NULL);
sysctl_enable_irq();
```

5.4　資料類型

相關資料類型、資料結構定義如下:

- gpio_drive_mode_t: GPIO 驅動模式。
- gpio_pin_value_t: GPIO 值。
- gpio_pin_edge_t: GPIO 邊緣觸發模式。

5.4.1　gpio_drive_mode_t

5.4.1.1　描述

GPIO 驅動模式。

5.4.1.2　定義

```
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　成員

成員名稱	描述
GPIO_DM_INPUT	輸入
GPIO_DM_INPUT_PULL_DOWN	輸入下拉
GPIO_DM_INPUT_PULL_UP	輸入上拉
GPIO_DM_OUTPUT	輸出

5.4.2　gpio_pin_value_t

5.4.2.1　描述

GPIO 值。

5.4.2.2　定義

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

5.4.2.3　成員

成員名稱	描述
GPIO_PV_LOW	低
GPIO_PV_HIGH	高

5.4.3　gpio_pin_edge_t

5.4.3.1　描述

高速 GPIO 邊緣觸發模式。

5.4.3.2　定義

```
typedef enum _gpio_pin_edge
{
    GPIO_PE_NONE,
    GPIO_PE_FALLING,
    GPIO_PE_RISING,
    GPIO_PE_BOTH,
    GPIO_PE_LOW,
    GPIO_PE_HIGH = 8,
} gpio_pin_edge_t;
```

5.4.3.3　成員

成員名稱	描述
GPIO_PE_NONE	不觸發
GPIO_PE_FALLING	負緣觸發
GPIO_PE_RISING	正緣觸發
GPIO_PE_BOTH	雙緣觸發

成員名稱	描述
GPIO_PE_LOW	低電平觸發
GPIO_PE_HIGH	高電平觸發

第 6 章

現場可編程 IO 陣列 (FPIOA)

6.1 概述

FPIOA（現場可編程 IO 陣列）允許用戶將 255 個內部功能映射到晶片外圍的 48 個自由 IO 上。

6.2 功能描述

- 支持 IO 的可編程功能選擇
- 支持 IO 輸出的 8 種驅動能力選擇
- 支持 IO 的內部上拉電阻選擇
- 支持 IO 的內部下拉電阻選擇
- 支持 IO 輸入的內部施密特觸發器設置
- 支持 IO 輸出的斜率控制
- 支持內部輸入邏輯的電平設置

6.3 API 參考

對應的頭文件 `fpioa.h`
為用戶提供以下介面

- `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　描述

設置 IO0-IO47 腳位復用功能。

6.3.1.2　函數原型

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

6.3.1.3　參數

參數名稱	描述	輸入輸出
number	IO 腳位號	輸入
function	FPIOA 功能號	輸入

6.3.1.4　返回值

返回值	描述
0	成功
非 0	失敗

6.3.2　fpioa_get_io_by_function

6.3.2.1　描述

根據功能號獲取 IO 腳位號。

6.3.2.2　函數原型

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

6.3.2.3　參數

參數名稱	描述	輸入輸出
function	FPIOA 功能號	輸入

6.3.2.4　返回值

返回值	描述
大於等於 0	IO 腳位號
小於 0	失敗

6.3.3　fpioa_get_io

6.3.3.1　描述

獲得 IO 腳位的配置。

6.3.3.2　函數原型

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

6.3.3.3　參數

參數名稱	描述	輸入輸出
number	IO 腳位號	輸入
cfg	腳位功能結構體	輸出

6.3.3.4　返回值

返回值	描述
0	成功
非 0	失敗

6.3.4　fpioa_set_io

6.3.4.1　描述

設置 IO 腳位的配置。

6.3.4.2　函數原型

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

6.3.4.3　參數

參數名稱	描述	輸入輸出
number	IO 腳位號	輸入
cfg	腳位功能結構體	輸入

6.3.4.4　返回值

返回值	描述
0	成功
非 0	失敗

6.3.5　fpioa_set_tie_enable

6.3.5.1　描述

啟動或禁用 FPIOA 功能輸入信號的強制輸入電平功能。

6.3.5.2　函數原型

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

6.3.5.3　參數

參數名稱	描述	輸入輸出
function	FPIOA 功能號	輸入
enable	強制輸入電平啟動位 0: 禁用 1: 啟動	輸入

6.3.5.4　返回值

返回值	描述
0	成功
非 0	失敗

6.3.6　fpioa_set_tie_value

6.3.6.1　描述

設置 FPIOA 功能輸入信號的強制輸入電平高或者低，僅在強制輸入電平功能啟用時生效。

6.3.6.2　函數原型

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

6.3.6.3　參數

參數名稱	描述	輸入輸出
function	FPIOA 功能號	輸入
value	強制輸入電平值 0: 低 1: 高	輸入

6.3.6.4　返回值

返回值	描述
0	成功
非 0	失敗

6.3.7　fpioa_set_pull

6.3.7.1　描述

設置 IO 的上拉下拉。

6.3.7.2　函數原型

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

6.3.7.3　參數

參數名稱	描述	輸入輸出
number	IO 編號	輸入
pull	上下拉值	輸入

6.3.7.4　返回值

返回值	描述
0	成功
非 0	失敗

6.3.8　fpioa_get_pull

6.3.8.1　描述

獲取 IO 腳位上下拉值。

6.3.8.2　函數原型

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

6.3.8.3　參數

參數名稱	描述	輸入輸出
number	IO 編號	輸入

6.3.8.4　返回值

返回值	描述
0	無上下拉
1	下拉
2	上拉

6.3.9　fpioa_set_io_driving

6.3.10　描述

設置 IO 腳位的驅動能力。

6.3.10.1　函數原型

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

6.3.10.2　參數

參數名稱	描述	輸入輸出
number	IO 編號	輸入
driving	驅動能力	輸入

6.3.10.3　返回值

返回值	描述
0	成功
非 0	失敗

6.3.11　fpioa_get_io_driving

6.3.11.1　描述

獲取驅動能力。

6.3.11.2　函數原型

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

6.3.11.3　參數

參數名稱	描述	輸入輸出
number	IO 編號	輸入

6.3.11.4　返回值

返回值	描述
大於等於 0	驅動能力
小於 0	失敗

6.4　資料類型

相關資料類型、資料結構定義如下:

- fpioa_function_t: 腳位的功能編號。
- fpioa_io_config_t: FPIOA 的配置。
- fpioa_pull_t: FPIOA 上拉或者下拉特性。
- fpioa_driving_t: FPIOA 驅動能力編號。

6.4.1　fpioa_io_config_t

6.4.1.1　描述

FPIOA 的配置。

6.4.1.2　定義

```
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　成員

成員名稱	描述
ch_sel	FPIOA 功能編號，從 256 種功能里選擇一種。
ds	驅動電流強度選擇，參照驅動電流強度表。
oe_en	輸出啟動 (OE)（手動模式)。1:啟動 0:禁用
oe_inv	輸出啟動取反。1:啟動 0:禁用
do_sel	輸出信號選擇。1:輸出 OE 信號 0:輸出原始信號
do_inv	輸出信號取反。1:啟動 0:禁用
pu	上拉啟動。1:啟動 0:禁用
pd	下拉啟動。1:啟動 0:禁用
resv0	保留位
sl	輸出擺率控制。1:低擺率，穩定 0: 高擺率，快
ie_en	輸入啟動 (IE)（手動模式)。1:啟動 0:禁用
ie_inv	輸入啟動取反。1:啟動 0:禁用
di_inv	輸入信號取反。1:啟動 0:禁用
st	輸入施密特觸發器。1:啟動 0:禁用
resv1	保留位。
pad_di	當前腳位的輸入電平。

6.4.2　驅動能力選擇表

6.4.2.1　低電平輸入電流

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　高電平輸出電流

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 描述

FPIOA 上拉或者下拉特性。

6.4.3.2 定義

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

6.4.3.3 成員

成員名稱	描述
FPIOA_PULL_NONE	無上下拉
FPIOA_PULL_DOWN	下拉
FPIOA_PULL_UP	上拉

禁止同時啟用晶片內部的上拉與下拉

6.4.4　fpioa_driving_t

6.4.4.1　描述

FPIOA 驅動能力編號，參見驅動能力選擇表。

6.4.4.2　定義

```
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　成員

成員名稱	描述
FPIOA_DRIVING_0	驅動能力 0
FPIOA_DRIVING_1	驅動能力 1
FPIOA_DRIVING_2	驅動能力 2
FPIOA_DRIVING_3	驅動能力 3
FPIOA_DRIVING_4	驅動能力 4
FPIOA_DRIVING_5	驅動能力 5
FPIOA_DRIVING_6	驅動能力 6
FPIOA_DRIVING_7	驅動能力 7

6.4.5　fpioa_function_t

6.4.5.1　描述

腳位的功能編號。

6.4.5.2　定義

```
typedef enum _fpioa_function
```

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

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

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

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

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

6.4.5.3　成員

成員名稱	描述
FUNC_JTAG_TCLK	JTAG 時脈介面
FUNC_JTAG_TDI	JTAG 資料輸入介面
FUNC_JTAG_TMS	JTAG 控制 TAP 狀態機的轉換
FUNC_JTAG_TDO	JTAG 資料輸出介面
FUNC_SPI0_D0	SPI0 資料線 0
FUNC_SPI0_D1	SPI0 資料線 1
FUNC_SPI0_D2	SPI0 資料線 2
FUNC_SPI0_D3	SPI0 資料線 3
FUNC_SPI0_D4	SPI0 資料線 4
FUNC_SPI0_D5	SPI0 資料線 5
FUNC_SPI0_D6	SPI0 資料線 6
FUNC_SPI0_D7	SPI0 資料線 7
FUNC_SPI0_SS0	SPI0 片選信號 0
FUNC_SPI0_SS1	SPI0 片選信號 1
FUNC_SPI0_SS2	SPI0 片選信號 2
FUNC_SPI0_SS3	SPI0 片選信號 3
FUNC_SPI0_ARB	SPI0 仲裁信號
FUNC_SPI0_SCLK	SPI0 時脈
FUNC_UARTHS_RX	UART 高速接收資料介面
FUNC_UARTHS_TX	UART 高速發送資料介面
FUNC_RESV6	保留功能
FUNC_RESV7	保留功能
FUNC_CLK_SPI1	SPI1 時脈
FUNC_CLK_I2C1	I2C1 時脈
FUNC_GPIOHS0	高速 GPIO0
FUNC_GPIOHS1	高速 GPIO1
FUNC_GPIOHS2	高速 GPIO2
FUNC_GPIOHS3	高速 GPIO3
FUNC_GPIOHS4	高速 GPIO4
FUNC_GPIOHS5	高速 GPIO5
FUNC_GPIOHS6	高速 GPIO6
FUNC_GPIOHS7	高速 GPIO7
FUNC_GPIOHS8	高速 GPIO8
FUNC_GPIOHS9	高速 GPIO9
FUNC_GPIOHS10	高速 GPIO10

成員名稱	描述
FUNC_GPIOHS11	高速 GPIO11
FUNC_GPIOHS12	高速 GPIO12
FUNC_GPIOHS13	高速 GPIO13
FUNC_GPIOHS14	高速 GPIO14
FUNC_GPIOHS15	高速 GPIO15
FUNC_GPIOHS16	高速 GPIO16
FUNC_GPIOHS17	高速 GPIO17
FUNC_GPIOHS18	高速 GPIO18
FUNC_GPIOHS19	高速 GPIO19
FUNC_GPIOHS20	高速 GPIO20
FUNC_GPIOHS21	高速 GPIO21
FUNC_GPIOHS22	高速 GPIO22
FUNC_GPIOHS23	高速 GPIO23
FUNC_GPIOHS24	高速 GPIO24
FUNC_GPIOHS25	高速 GPIO25
FUNC_GPIOHS26	高速 GPIO26
FUNC_GPIOHS27	高速 GPIO27
FUNC_GPIOHS28	高速 GPIO28
FUNC_GPIOHS29	高速 GPIO29
FUNC_GPIOHS30	高速 GPIO30
FUNC_GPIOHS31	高速 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 接收資料介面
FUNC_UART1_TX	UART1 發送資料介面
FUNC_UART2_RX	UART2 接收資料介面
FUNC_UART2_TX	UART2 發送資料介面
FUNC_UART3_RX	UART3 接收資料介面
FUNC_UART3_TX	UART3 發送資料介面

成員名稱	描述
FUNC_SPI1_D0	SPI1 資料線 0
FUNC_SPI1_D1	SPI1 資料線 1
FUNC_SPI1_D2	SPI1 資料線 2
FUNC_SPI1_D3	SPI1 資料線 3
FUNC_SPI1_D4	SPI1 資料線 4
FUNC_SPI1_D5	SPI1 資料線 5
FUNC_SPI1_D6	SPI1 資料線 6
FUNC_SPI1_D7	SPI1 資料線 7
FUNC_SPI1_SS0	SPI1 片選信號 0
FUNC_SPI1_SS1	SPI1 片選信號 1
FUNC_SPI1_SS2	SPI1 片選信號 2
FUNC_SPI1_SS3	SPI1 片選信號 3
FUNC_SPI1_ARB	SPI1 仲裁信號
FUNC_SPI1_SCLK	SPI1 時脈
FUNC_SPI_SLAVE_D0	SPI 從模式資料線 0
FUNC_SPI_SLAVE_SS	SPI 從模式片選信號
FUNC_SPI_SLAVE_SCLK	SPI 從模式時脈
FUNC_I2S0_MCLK	I2S0 主時脈（系統時脈）
FUNC_I2S0_SCLK	I2S0 串列時脈（位時脈）
FUNC_I2S0_WS	I2S0 幀時脈
FUNC_I2S0_IN_D0	I2S0 串列輸入資料介面 0
FUNC_I2S0_IN_D1	I2S0 串列輸入資料介面 1
FUNC_I2S0_IN_D2	I2S0 串列輸入資料介面 2
FUNC_I2S0_IN_D3	I2S0 串列輸入資料介面 3
FUNC_I2S0_OUT_D0	I2S0 串列輸出資料介面 0
FUNC_I2S0_OUT_D1	I2S0 串列輸出資料介面 1
FUNC_I2S0_OUT_D2	I2S0 串列輸出資料介面 2
FUNC_I2S0_OUT_D3	I2S0 串列輸出資料介面 3
FUNC_I2S1_MCLK	I2S1 主時脈（系統時脈）
FUNC_I2S1_SCLK	I2S1 串列時脈（位時脈）
FUNC_I2S1_WS	I2S1 幀時脈
FUNC_I2S1_IN_D0	I2S1 串列輸入資料介面 0
FUNC_I2S1_IN_D1	I2S1 串列輸入資料介面 1
FUNC_I2S1_IN_D2	I2S1 串列輸入資料介面 2
FUNC_I2S1_IN_D3	I2S1 串列輸入資料介面 3

成員名稱	描述
FUNC_I2S1_OUT_D0	I2S1 串列輸出資料介面 0
FUNC_I2S1_OUT_D1	I2S1 串列輸出資料介面 1
FUNC_I2S1_OUT_D2	I2S1 串列輸出資料介面 2
FUNC_I2S1_OUT_D3	I2S1 串列輸出資料介面 3
FUNC_I2S2_MCLK	I2S2 主時脈（系統時脈）
FUNC_I2S2_SCLK	I2S2 串列時脈（位時脈）
FUNC_I2S2_WS	I2S2 幀時脈
FUNC_I2S2_IN_D0	I2S2 串列輸入資料介面 0
FUNC_I2S2_IN_D1	I2S2 串列輸入資料介面 1
FUNC_I2S2_IN_D2	I2S2 串列輸入資料介面 2
FUNC_I2S2_IN_D3	I2S2 串列輸入資料介面 3
FUNC_I2S2_OUT_D0	I2S2 串列輸出資料介面 0
FUNC_I2S2_OUT_D1	I2S2 串列輸出資料介面 1
FUNC_I2S2_OUT_D2	I2S2 串列輸出資料介面 2
FUNC_I2S2_OUT_D3	I2S2 串列輸出資料介面 3
FUNC_RESV0	保留功能
FUNC_RESV1	保留功能
FUNC_RESV2	保留功能
FUNC_RESV3	保留功能
FUNC_RESV4	保留功能
FUNC_RESV5	保留功能
FUNC_I2C0_SCLK	I2C0 串列時脈
FUNC_I2C0_SDA	I2C0 串列資料介面
FUNC_I2C1_SCLK	I2C1 串列時脈
FUNC_I2C1_SDA	I2C1 串列資料介面
FUNC_I2C2_SCLK	I2C2 串列時脈
FUNC_I2C2_SDA	I2C2 串列資料介面
FUNC_CMOS_XCLK	DVP 系統時脈
FUNC_CMOS_RST	DVP 系統複位信號
FUNC_CMOS_PWDN	DVP 啟動信號
FUNC_CMOS_VSYNC	DVP 場同步
FUNC_CMOS_HREF	DVP 行參考信號
FUNC_CMOS_PCLK	像素時脈
FUNC_CMOS_D0	像素資料 0
FUNC_CMOS_D1	像素資料 1

成員名稱	描述
FUNC_CMOS_D2	像素資料 2
FUNC_CMOS_D3	像素資料 3
FUNC_CMOS_D4	像素資料 4
FUNC_CMOS_D5	像素資料 5
FUNC_CMOS_D6	像素資料 6
FUNC_CMOS_D7	像素資料 7
FUNC_SCCB_SCLK	SCCB 時脈
FUNC_SCCB_SDA	SCCB 串列資料信號
FUNC_UART1_CTS	UART1 清除發送信號
FUNC_UART1_DSR	UART1 資料裝置準備信號
FUNC_UART1_DCD	UART1 資料載波檢測
FUNC_UART1_RI	UART1 振鈴指示
FUNC_UART1_SIR_IN	UART1 串列紅外輸入信號
FUNC_UART1_DTR	UART1 資料終端準備信號
FUNC_UART1_RTS	UART1 發送請求信號
FUNC_UART1_OUT2	UART1 用戶指定輸出信號 2
FUNC_UART1_OUT1	UART1 用戶指定輸出信號 1
FUNC_UART1_SIR_OUT	UART1 串列紅外輸出信號
FUNC_UART1_BAUD	UART1 時脈
FUNC_UART1_RE	UART1 接收啟動
FUNC_UART1_DE	UART1 發送啟動
FUNC_UART1_RS485_EN	UART1 啟動 RS485
FUNC_UART2_CTS	UART2 清除發送信號
FUNC_UART2_DSR	UART2 資料裝置準備信號
FUNC_UART2_DCD	UART2 資料載波檢測
FUNC_UART2_RI	UART2 振鈴指示
FUNC_UART2_SIR_IN	UART2 串列紅外輸入信號
FUNC_UART2_DTR	UART2 資料終端準備信號
FUNC_UART2_RTS	UART2 發送請求信號
FUNC_UART2_OUT2	UART2 用戶指定輸出信號 2
FUNC_UART2_OUT1	UART2 用戶指定輸出信號 1
FUNC_UART2_SIR_OUT	UART2 串列紅外輸出信號
FUNC_UART2_BAUD	UART2 時脈
FUNC_UART2_RE	UART2 接收啟動
FUNC_UART2_DE	UART2 發送啟動

成員名稱	描述
FUNC_UART2_RS485_EN	UART2 啟動 RS485
FUNC_UART3_CTS	清除發送信號
FUNC_UART3_DSR	資料裝置準備信號
FUNC_UART3_DCD	UART3 資料載波檢測
FUNC_UART3_RI	UART3 振鈴指示
FUNC_UART3_SIR_IN	UART3 串列紅外輸入信號
FUNC_UART3_DTR	UART3 資料終端準備信號
FUNC_UART3_RTS	UART3 發送請求信號
FUNC_UART3_OUT2	UART3 用戶指定輸出信號 2
FUNC_UART3_OUT1	UART3 用戶指定輸出信號 1
FUNC_UART3_SIR_OUT	UART3 串列紅外輸出信號
FUNC_UART3_BAUD	UART3 時脈
FUNC_UART3_RE	UART3 接收啟動
FUNC_UART3_DE	UART3 發送啟動
FUNC_UART3_RS485_EN	UART3 啟動 RS485
FUNC_TIMER0_TOGGLE1	TIMER0 輸出信號 1
FUNC_TIMER0_TOGGLE2	TIMER0 輸出信號 2
FUNC_TIMER0_TOGGLE3	TIMER0 輸出信號 3
FUNC_TIMER0_TOGGLE4	TIMER0 輸出信號 4
FUNC_TIMER1_TOGGLE1	TIMER1 輸出信號 1
FUNC_TIMER1_TOGGLE2	TIMER1 輸出信號 2
FUNC_TIMER1_TOGGLE3	TIMER1 輸出信號 3
FUNC_TIMER1_TOGGLE4	TIMER1 輸出信號 4
FUNC_TIMER2_TOGGLE1	TIMER2 輸出信號 1
FUNC_TIMER2_TOGGLE2	TIMER2 輸出信號 2
FUNC_TIMER2_TOGGLE3	TIMER2 輸出信號 3
FUNC_TIMER2_TOGGLE4	TIMER2 輸出信號 4
FUNC_CLK_SPI2	SPI2 時脈
FUNC_CLK_I2C2	I2C2 時脈
FUNC_INTERNAL0	內部功能 0
FUNC_INTERNAL1	內部功能 1
FUNC_INTERNAL2	內部功能 2
FUNC_INTERNAL3	內部功能 3
FUNC_INTERNAL4	內部功能 4
FUNC_INTERNAL5	內部功能 5

成員名稱	描述
FUNC_INTERNAL6	內部功能 6
FUNC_INTERNAL7	內部功能 7
FUNC_INTERNAL8	內部功能 8
FUNC_INTERNAL9	內部功能 9
FUNC_INTERNAL10	內部功能 10
FUNC_INTERNAL11	內部功能 11
FUNC_INTERNAL12	內部功能 12
FUNC_INTERNAL13	內部功能 13
FUNC_INTERNAL14	內部功能 14
FUNC_INTERNAL15	內部功能 15
FUNC_INTERNAL16	內部功能 16
FUNC_INTERNAL17	內部功能 17
FUNC_CONSTANT	常量
FUNC_INTERNAL18	內部功能 18
FUNC_DEBUG0	調試功能 0
FUNC_DEBUG1	調試功能 1
FUNC_DEBUG2	調試功能 2
FUNC_DEBUG3	調試功能 3
FUNC_DEBUG4	調試功能 4
FUNC_DEBUG5	調試功能 5
FUNC_DEBUG6	調試功能 6
FUNC_DEBUG7	調試功能 7
FUNC_DEBUG8	調試功能 8
FUNC_DEBUG9	調試功能 9
FUNC_DEBUG10	調試功能 10
FUNC_DEBUG11	調試功能 11
FUNC_DEBUG12	調試功能 12
FUNC_DEBUG13	調試功能 13
FUNC_DEBUG14	調試功能 14
FUNC_DEBUG15	調試功能 15
FUNC_DEBUG16	調試功能 16
FUNC_DEBUG17	調試功能 17
FUNC_DEBUG18	調試功能 18
FUNC_DEBUG19	調試功能 19
FUNC_DEBUG20	調試功能 20

成員名稱	描述
FUNC_DEBUG21	調試功能 21
FUNC_DEBUG22	調試功能 22
FUNC_DEBUG23	調試功能 23
FUNC_DEBUG24	調試功能 24
FUNC_DEBUG25	調試功能 25
FUNC_DEBUG26	調試功能 26
FUNC_DEBUG27	調試功能 27
FUNC_DEBUG28	調試功能 28
FUNC_DEBUG29	調試功能 29
FUNC_DEBUG30	調試功能 30
FUNC_DEBUG31	調試功能 31

第 7 章

數位攝像頭介面（DVP）

7.1 概述

DVP 是攝像頭介面模組，支持把攝像頭輸入圖像資料轉發給 AI 模組或者內部儲存。

7.2 功能描述

DVP 模組具有以下功能：

- 支持 RGB565、RGB422 與單通道 Y 灰度輸入模式
- 支持設置幀中斷
- 支持設置傳輸地址
- 支持同時向兩個地址寫資料（輸出格式分別是 RGB888 與 RGB565）
- 支持丟棄不需要處理的幀

7.3 API 參考

對應的頭文件 `dvp.h`
為用戶提供以下介面

- `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
- dvp_sccb_set_clk_rate
- dvp_set_xclk_rate

7.3.1　dvp_init

7.3.1.1　描述

初始化 DVP。

7.3.1.2　函數原型

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

7.3.1.3　參數

參數名稱	描述	輸入輸出
reg_len	sccb 寄存器長度	輸入

7.3.1.4　返回值

無

7.3.2　dvp_set_output_enable

7.3.2.1　描述

設置輸出模式啟動或禁用。

7.3.2.2　函數原型

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

7.3.2.3　參數

參數名稱	描述	輸入輸出
index	圖像輸出至內部儲存或 AI	輸入
enable	0：禁用 1：啟動	輸入

7.3.2.4　返回值

無。

7.3.3　dvp_set_image_format

7.3.3.1　描述

設置圖像接收模式，RGB 或 YUV。

7.3.3.2　函數原型

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

7.3.3.3　參數

參數名稱	描述	輸入輸出
format	圖像模式 DVP_CFG_RGB_FORMAT RGB 模式 DVP_CFG_YUV_FORMAT YUV 模式	輸入

7.3.3.4　返回值

無

7.3.4　dvp_set_image_size

7.3.4.1　描述

設置 DVP 圖像採集尺寸。

7.3.4.2　函數原型

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

7.3.4.3　參數

參數名稱	描述	輸入輸出
width	圖像寬度	輸入
height	圖像高度	輸入

7.3.4.4　返回值

無

7.3.5　dvp_set_ai_addr

7.3.5.1　描述

設置 AI 存放圖像的地址，供 AI 模組進行演算法處理。

7.3.5.2　函數原型

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

7.3.5.3　參數

參數名稱	描述	輸入輸出
r_addr	紅色分量地址	輸入
g_addr	綠色分量地址	輸入
b_addr	藍色分量地址	輸入

7.3.5.4　返回值

無

7.3.6　dvp_set_display_addr

7.3.6.1　描述

設置採集圖像在內部儲存中的存放地址，可以用來顯示。

7.3.6.2　函數原型

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

7.3.6.3　參數

參數名稱	描述	輸入輸出
addr	存放圖像的內部儲存地址	輸入

7.3.6.4　返回值

無

7.3.7　dvp_config_interrupt

配置 DVP 中斷類型。

7.3.7.1　函數原型

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

7.3.7.2　描述

設置圖像開始和結束中斷狀態，啟動或禁用。

7.3.7.3　參數

參數名稱	描述	輸入輸出
interrupt	中斷類型 DVP_CFG_START_INT_ENABLE 圖像開始採集中斷 DVP_CFG_FINISH_INT_ENABLE 圖像結束採集中斷	輸入
enable	0: 禁止 1: 啟動	輸入

7.3.7.4　返回值

無。

7.3.8　dvp_get_interrupt

7.3.8.1　描述

判斷是否是輸入的中斷類型。

7.3.8.2　函數原型

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

7.3.8.3　參數

參數名稱	描述	輸入輸出
interrupt	中斷類型 DVP_CFG_START_INT_ENABLE 圖像開始採集中斷 DVP_CFG_FINISH_INT_ENABLE 圖像結束採集中斷	輸入

7.3.8.4　返回值

返回值	描述
0	否
非 0	是

7.3.9　dvp_clear_interrupt

7.3.9.1　描述

清除中斷。

7.3.9.2　函數原型

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

7.3.9.3　參數

參數名稱	描述	輸入輸出
interrupt	中斷類型 DVP_CFG_START_INT_ENABLE 圖像開始採集中斷 DVP_CFG_FINISH_INT_ENABLE 圖像結束採集中斷	輸入

7.3.9.4　返回值

無。

7.3.10　dvp_start_convert

7.3.10.1　描述

開始採集圖像，在確定圖像採集開始中斷後調用。

7.3.10.2　函數原型

```
void dvp_start_convert(void)
```

7.3.10.3　參數

無。

7.3.10.4　返回值

無。

7.3.11　dvp_enable_burst

7.3.11.1　描述

啟動突發傳輸模式。

7.3.11.2　函數原型

```
void dvp_enable_burst(void)
```

7.3.11.3　參數

無。

7.3.11.4　返回值

無。

7.3.12　dvp_disable_burst

7.3.12.1　描述

禁用突發傳輸模式。

7.3.12.2　函數原型

```
void dvp_disable_burst(void)
```

7.3.12.3　參數

無。

7.3.12.4　返回值

無。

7.3.13　dvp_enable_auto

7.3.13.1　描述

啟動自動接收圖像模式。

7.3.13.2　函數原型

```
void dvp_enable_auto(void)
```

7.3.13.3　參數

無。

7.3.13.4　返回值

無。

7.3.14　dvp_disable_auto

7.3.14.1　描述

禁用自動接收圖像模式。

7.3.14.2　函數原型

```
void dvp_disable_auto(void)
```

7.3.14.3　參數

無。

7.3.14.4　返回值

無。

7.3.15　dvp_sccb_send_data

7.3.15.1　描述

通過 sccb 發送資料。

7.3.15.2　函數原型

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

7.3.15.3　參數

參數名稱	描述	輸入輸出
dev_addr	外部裝置圖像感測器 SCCB 地址	輸入
reg_addr	外部裝置圖像感測器寄存器	輸入
reg_data	發送的資料	輸入

7.3.15.4　返回值

無

7.3.16　dvp_sccb_receive_data

7.3.16.1　描述

通過 SCCB 接收資料。

7.3.16.2　函數原型

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

7.3.16.3　參數

參數名稱	描述	輸入輸出
dev_addr	外部裝置圖像感測器 SCCB 地址	輸入
reg_addr	外部裝置圖像感測器寄存器	輸入

7.3.16.4　返回值

讀取寄存器的資料。

7.3.17　dvp_set_xclk_rate

7.3.17.1　描述

設置 xclk 的速率。

7.3.17.2　函數原型

```
uint32_t dvp_set_xclk_rate(uint32_t xclk_rate)
```

7.3.17.3　參數

參數名稱	描述	輸入輸出
xclk_rate	xclk 的速率	輸入

7.3.17.4　返回值

xclk 的實際速率。

7.3.18　dvp_sccb_set_clk_rate

7.3.18.1　描述

設置 sccb 的速率。

7.3.18.2　函數原型

```
uint32_t dvp_sccb_set_clk_rate(uint32_t clk_rate)
```

7.3.18.3　參數

參數名稱	描述	輸入輸出
clk_rate	sccb 的速率	輸入

7.3.18.4　返回值

返回值	描述
0	失敗，設置的速率太低無法滿足，請使用 I2C
非 0	實際的 sccb 速率

7.3.19　舉例

```
/* 採集320 * 240的RGB圖像資料傳輸至lcd_gram0，及0x40600000 0x40612C00 0x40625800 地址處
    */
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_set_xclk_rate(12000000);
dvp_enable_burst();
dvp_set_output_enable(DVP_OUTPUT_AI, 1);
dvp_set_output_enable(DVP_OUTPUT_DISPLAY, 1);
dvp_set_image_format(DVP_CFG_RGB_FORMAT);
dvp_set_image_size(320, 240);
dvp_set_ai_addr((uint32_t)0x40600000, (uint32_t)0x40612C00, (uint32_t)0x40625800);
dvp_set_display_addr(lcd_gram0);
dvp_config_interrupt(DVP_CFG_START_INT_ENABLE | DVP_CFG_FINISH_INT_ENABLE, 0);
dvp_disable_auto();
plic_set_priority(IRQN_DVP_INTERRUPT, 1);
plic_irq_register(IRQN_DVP_INTERRUPT, on_irq_dvp, NULL);
plic_irq_enable(IRQN_DVP_INTERRUPT);
dvp_clear_interrupt(DVP_STS_FRAME_START | DVP_STS_FRAME_FINISH);
dvp_config_interrupt(DVP_CFG_START_INT_ENABLE | DVP_CFG_FINISH_INT_ENABLE, 1);
sysctl_enable_irq();
/* 通過SCCB向地址0x60的外部裝置0xFF寄存器發送0x01,從寄存器0x1D讀取資料 */
dvp_sccb_send_data(0x60, 0xFF, 0x01);
dvp_sccb_receive_data(0x60, 0x1D)
```

7.4　資料類型

相關資料類型、資料結構定義如下：

- dvp_output_mode_t：DVP 輸出圖像的模式。

7.4.1　dvp_output_mode_t

7.4.1.1　描述

DVP 輸入圖像的模式。

7.4.1.2　定義

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

7.4.1.3　成員

成員名稱	描述
DVP_OUTPUT_AI	AI 輸出
DVP_OUTPUT_DISPLAY	向內部儲存輸出用於顯示

第 8 章

快速傅立葉變換加速器 (FFT)

8.1 概述

FFT 模組是用硬體的方式來實現 FFT 的基 2 時分運算加速。

8.2 功能描述

目前該模組可以支持 64 點、128 點、256 點以及 512 點的 FFT 以及 IFFT。在 FFT 內部有兩塊大小為 512*32bit 的 SRAM，在配置完成後 FFT 會向 DMA 發送 TX 請求，將 DMA 送來的送據放到其中的一塊 SRAM 中去，直到滿足當前 FFT 運算所需要的資料量並開始 FFT 運算，蝶形運算單元從包含有有效資料的 SRAM 中讀出資料，運算結束後將資料寫到另外一塊 SRAM 中去，下次蝶形運算再從剛寫入的 SRAM 中讀出資料，運算結束後並寫入另外一塊 SRAM，如此反覆交替進行直到完成整個 FFT 運算。

8.3 API 參考

對應的頭文件 `fft.h`
為用戶提供以下介面

- `fft_complex_uint16_dma`

8.3.1 fft_complex_uint16_dma

8.3.1.1 描述

FFT 運算。

8.3.1.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_send_channel_num	發送資料使用的 DMA 通道號	輸入
dma_receive_channel_num	接收資料使用的 DMA 通道號	輸入
shift	FFT 模組 16 位寄存器導致資料溢出 (-32768~32767)，FFT 變換有 9 層，shift 決定哪一層需要移位操作 (如 0x1ff 表示 9 層都做移位操作；0x03 表示第第一層與第二層做移位操作)，防止溢出。如果移位了，則變換後的幅值不是正常 FFT 變換的幅值，對應關係可以參考 fft_test 測試 demo 程序。包含了求解頻率點、相位、幅值的示例	輸入
direction	FFT 正變換或是逆變換	輸入
input	輸入的資料序列，格式為 RIRI..，實部與虛部的精度都為 16bit	輸入
point_num	待運算的資料點數，只能為 512/256/128/64	輸入
output	運算後結果。格式為 RIRI..，實部與虛部的精度都為 16bit	輸出

8.3.1.4　返回值

無。

8.3.2　舉例

```
#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　資料類型

相關資料類型、資料結構定義如下：

- fft_data_t: FFT 運算傳入的資料格式。
- fft_direction_t: FFT 變換模式。

8.4.1　fft_data_t

8.4.1.1　描述

FFT 運算傳入的資料格式。

8.4.1.2　定義

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

8.4.1.3　成員

成員名稱	描述
I1	第一個資料的虛部
R1	第一個資料的實部
I2	第二個資料的虛部
R2	第二個資料的實部

8.4.2　fft_direction_t

8.4.2.1　描述

FFT 變換模式

8.4.2.2　定義

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

8.4.2.3　成員

成員名稱	描述
FFT_DIR_BACKWARD	FFT 逆變換
FFT_DIR_FORWARD	FFT 正變換

第 9 章

安全散列演算法加速器 (SHA256)

9.1 功能描述

- 支持 SHA-256 的計算

9.2 API 參考

對應的頭文件 sha256.h

為用戶提供以下介面

- sha256_init
- sha256_update
- sha256_final
- sha256_hard_calculate

9.2.1 sha256_init

9.2.1.1 描述

初始化 SHA256 加速器外部裝置.

9.2.1.2 函數原型

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

9.2.1.3 參數

參數名稱	描述	輸入輸出
context	SHA256 的上下文對象	輸入
input_len	待計算 SHA256 hash 的資訊的長度	輸入

9.2.1.4　返回值

無。

9.2.2　舉例

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

9.2.3　sha256_update

9.2.3.1　描述

傳入一個資料塊參與 SHA256 Hash 計算

9.2.3.2　函數原型

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

9.2.3.3　參數

參數名稱	描述	輸入輸出
context	SHA256 的上下文對象	輸入
input	待加入計算的 SHA256 計算的資料塊	輸入
buf_len	待加入計算的 SHA256 計算資料塊的長度	輸入

9.2.3.4　返回值

無。

9.2.4　sha256_final

9.2.4.1　描述

結束對資料的 SHA256 Hash 計算

9.2.4.2　函數原型

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

9.2.4.3　參數

參數名稱	描述	輸入輸出
context	SHA256 的上下文對象	輸入
output	存放 SHA256 計算的結果，需保證傳入這個 buffer 的大小為 32Bytes 以上	輸出

9.2.4.4　返回值

無。

9.2.5　sha256_hard_calculate

9.2.5.1　描述

一次性對連續的資料計算它的 SHA256 Hash

9.2.5.2　函數原型

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

9.2.5.3　參數

參數名稱	描述	輸入輸出
input	待 SHA256 計算的資料	輸入
input_len	待 SHA256 計算資料的長度	輸入
output	存放 SHA256 計算的結果，需保證傳入這個 buffer 的大小為 32Bytes 以上	輸出

9.2.5.4　返回值

無。

9.2.6　舉例

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

9.3　常式

9.3.1　進行一次計算

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

或者可以直接調用 sha256_hard_calculate 函數

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

9.3.2　進行分塊計算

```
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);
```

第 10 章

通用非同步收發傳輸器（UART）

10.1 概述

嵌入式應用通常要求一個簡單的並且占用系統資源少的方法來傳輸資料。通用非同步收發傳輸器（UART）即可以滿足這些要求，它能夠靈活地與外部裝置進行全雙工資料交換。

10.2 功能描述

UART 模組具有以下功能：

- 配置 UART 參數
- 自動收取資料到緩衝區

10.3 API 參考

對應的頭文件 uart.h
為用戶提供以下介面

- uart_init
- uart_config（0.6.0 後不再支持，請使用 uart_configure）
- uart_configure
- uart_send_data
- uart_send_data_dma
- uart_send_data_dma_irq
- uart_receive_data
- uart_receive_data_dma

- uart_receive_data_dma_irq
- uart_irq_register
- uart_irq_deregister

10.3.1 uart_init

10.3.1.1 描述

初始化 uart。

10.3.1.2 函數原型

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

10.3.1.3 參數

參數名稱	描述	輸入輸出
channel	UART 號	輸入

10.3.1.4 返回值

無。

10.3.2 uart_configure

10.3.2.1 描述

設置 UART 相關參數。該函數已廢棄，替代函數為 uart_configure。

10.3.3 uart_configure

10.3.3.1 描述

設置 UART 相關參數。

10.3.3.2 函數原型

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

10.3.3.3　參數

參數名稱	描述	輸入輸出
channel	UART 編號	輸入
baud_rate	波特率	輸入
data_width	資料位（5-8）	輸入
stopbit	停止位	輸入
parity	校驗位	輸入

10.3.3.4　返回值

無。

10.3.4　uart_send_data

10.3.4.1　描述

通過 UART 發送資料。

10.3.4.2　函數原型

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

10.3.4.3　參數

參數名稱	描述	輸入輸出
channel	UART 編號	輸入
buffer	待發送資料	輸入
buf_len	待發送資料的長度	輸入

10.3.4.4　返回值

已發送資料的長度。

10.3.5　uart_send_data_dma

10.3.5.1　描述

UART 通過 DMA 發送資料。資料全部發送完畢後返回。

10.3.5.2　函數原型

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

10.3.5.3　參數

參數名稱	描述	輸入輸出
uart_channel	UART 編號	輸入
dmac_channel	DMA 通道	輸入
buffer	待發送資料	輸入
buf_len	待發送資料的長度	輸入

10.3.5.4　返回值

無。

10.3.6　uart_send_data_dma_irq

10.3.6.1　描述

UART 通過 DMA 發送資料，並設置 DMA 發送完成中斷函數，僅單次中斷。

10.3.6.2　函數原型

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

10.3.6.3　參數

參數名稱	描述	輸入輸出
uart_channel	UART 編號	輸入
dmac_channel	DMA 通道	輸入
buffer	待發送資料	輸入
buf_len	待發送資料的長度	輸入

參數名稱	描述	輸入輸出
uart_callback	DMA 中斷回調	輸入
ctx	中斷函數參數	輸入
priority	中斷優先順序	輸入

10.3.6.4　返回值

無。

10.3.7　uart_receive_data

10.3.7.1　描述

通過 UART 讀取資料。

10.3.7.2　函數原型

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

10.3.7.3　參數

參數名稱	描述	輸入輸出
channel	UART 編號	輸入
buffer	接收資料	輸出
buf_len	接收資料的長度	輸入

10.3.7.4　返回值

已接收到的資料長度。

10.3.8　uart_receive_data_dma

10.3.8.1　描述

UART 通過 DMA 接收資料。

10.3.8.2　函數原型

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

10.3.8.3　參數

參數名稱	描述	輸入輸出
uart_channel	UART 編號	輸入
dmac_channel	DMA 通道	輸入
buffer	接收資料	輸出
buf_len	接收資料的長度	輸入

10.3.8.4　返回值

無。

10.3.9　uart_receive_data_dma_irq

10.3.9.1　描述

UART 通過 DMA 接收資料，並註冊 DMA 接收完成中斷函數，僅單次中斷。

10.3.9.2　函數原型

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

10.3.9.3　參數

參數名稱	描述	輸入輸出
uart_channel	UART 編號	輸入
dmac_channel	DMA 通道	輸入
buffer	接收資料	輸出
buf_len	接收資料的長度	輸入
uart_callback	DMA 中斷回調	輸入
ctx	中斷函數參數	輸入
priority	中斷優先順序	輸入

10.3.9.4　返回值

無。

10.3.10　uart_irq_register

10.3.10.1　描述

註冊 UART 中斷函數。

10.3.10.2　函數原型

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

10.3.10.3　參數

參數名稱	描述	輸入輸出
channel	UART 編號	輸入
interrupt_mode	中斷類型	輸入
uart_callback	中斷回調	輸入
ctx	中斷函數參數	輸入
priority	中斷優先順序	輸入

10.3.10.4　返回值

無。

10.3.11　uart_irq_deregister

10.3.11.1　描述

註銷 UART 中斷函數。

10.3.11.2　函數原型

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

10.3.11.3　參數

參數名稱	描述	輸入輸出
channel	UART 編號	輸入
interrupt_mode	中斷類型	輸入

10.3.11.4　返回值

無。

10.3.12　舉例

```
/* UART1 波特率115200，8位資料，1位停止位，無校驗位 */
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"};
/* 發送 hello world! */
uart_send_data(UART_DEVICE_1, hel, strlen(v_hel));
/* 接收資料 */
while(uart_receive_data(UART_DEVICE_1, &recv, 1))
{
    printf("%c ", recv);
}
```

10.4　資料類型

相關資料類型、資料結構定義如下：

- uart_device_number_t：UART 編號。
- uart_bitwidth_t：UART 資料位寬。
- uart_stopbits_t：UART 停止位。
- uart_parity_t：UART 校驗位。
- uart_interrupt_mode_t：UART 中斷類型，接收或發送。
- uart_send_trigger_t：發送中斷或 DMA 觸發 FIFO 深度。
- uart_receive_trigger_t：接收中斷或 DMA 觸發 FIFO 深度。

10.4.1　uart_device_number_t

10.4.1.1　描述

UART 編號。

10.4.1.2　定義

```
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　成員

成員名稱	描述
UART_DEVICE_1	UART 1
UART_DEVICE_2	UART 2
UART_DEVICE_3	UART 3

10.4.2　uart_bitwidth_t

10.4.2.1　描述

UART 資料位寬。

10.4.2.2　定義

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

10.4.2.3　成員

成員名稱	描述
UART*BITWIDTH*5BIT	5 比特
UART*BITWIDTH*6BIT	6 比特
UART*BITWIDTH*7BIT	7 比特

成員名稱	描述
UART*BITWIDTH*8BIT	8 比特

10.4.3　uart_stopbits_t

10.4.3.1　描述

UART 停止位。

10.4.3.2　定義

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

10.4.3.3　成員

成員名稱	描述
UART_STOP_1	1 個停止位
UART_STOP_1_5	1.5 個停止位
UART_STOP_2	2 個停止位

10.4.4　uart_parity_t

10.4.4.1　描述

UART 校驗位。

10.4.4.2　定義

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

10.4.4.3　成員

成員名稱	描述
UART_PARITY_NONE	無校驗位
UART_PARITY_ODD	奇校驗
UART_PARITY_EVEN	偶校驗

10.4.5 uart_interrupt_mode_t

10.4.5.1 描述

UART 中斷類型，接收或發送。

10.4.5.2 定義

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

10.4.5.3 成員

成員名稱	描述
UART_SEND	UART 發送
UART_RECEIVE	UART 接收

10.4.6 uart_send_trigger_t

10.4.6.1 描述

發送中斷或 DMA 觸發 FIFO 深度。當 FIFO 中的資料小於等於該值時觸發中斷或 DMA 傳輸。FIFO 的深度為 16 位元組。

10.4.6.2 定義

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

10.4.6.3　成員

成員名稱	描述
UART_SEND_FIFO_0	FIFO 為空
UART_SEND_FIFO_2	FIFO 剩餘 2 位元組
UART_SEND_FIFO_4	FIFO 剩餘 4 位元組
UART_SEND_FIFO_8	FIFO 剩餘 8 位元組

10.4.7　uart_receive_trigger_t

10.4.7.1　描述

接收中斷或 DMA 觸發 FIFO 深度。當 FIFO 中的資料大於等於該值時觸發中斷或 DMA 傳輸。FIFO 的深度為 16 位元組。

10.4.7.2　定義

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

10.4.7.3　成員

成員名稱	描述
UART_RECEIVE_FIFO_1	FIFO 剩餘 1 位元組
UART_RECEIVE_FIFO_4	FIFO 剩餘 2 位元組
UART_RECEIVE_FIFO_8	FIFO 剩餘 4 位元組
UART_RECEIVE_FIFO_14	FIFO 剩餘 8 位元組

第 11 章

高速通用非同步收發傳輸器（UARTHS）

11.1 概述

嵌入式應用通常要求一個簡單的並且占用系統資源少的方法來傳輸資料。高速通用非同步收發傳輸器（UARTHS）即可以滿足這些要求，它能夠靈活地與外部裝置進行全雙工資料交換。目前系統使用該串口做為調試串口，printf 時會調用該串口輸出。

11.2 功能描述

UARTHS 模組具有以下功能:

- 配置 UARTHS 參數
- 自動收取資料到緩衝區

11.3 API 參考

對應的頭文件 uarths.h
為用戶提供以下介面

- 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　描述

初始化 UARTHS，系統默認波特率為 115200 8bit 1 位停止位無檢驗位。因為 uarths 時脈源為 PLL0，在設置 PLL0 後需要重新調用該函數設置波特率，否則會列印亂碼。

11.3.1.2　函數原型

```
void uarths_init(void)
```

11.3.1.3　參數

無。

11.3.1.4　返回值

無。

11.3.2　uarths_config

11.3.2.1　描述

設置 UARTHS 的參數。默認 8bit 資料，無校驗位。

11.3.2.2　函數原型

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

11.3.3　參數

參數名稱	描述	輸入輸出
baud_rate	波特率	輸入
stopbit	停止位	輸入

11.3.3.1　返回值

無。

11.3.4　uarths_receive_data

11.3.4.1　描述

通過 UARTHS 讀取資料。

11.3.4.2　函數原型

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

11.3.4.3　參數

參數名稱	描述	輸入輸出
buf	接收資料	輸出
buf_len	接收資料的長度	輸入

11.3.4.4　返回值

已接收到的資料長度。

11.3.5　uarths_send_data

11.3.5.1　描述

通過 UART 發送資料。

11.3.5.2　函數原型

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

11.3.5.3　參數

參數名稱	描述	輸入輸出
buf	待發送資料	輸入
buf_len	待發送資料的長度	輸入

11.3.5.4　返回值

已發送資料的長度。

11.3.6　uarths_set_irq

11.3.6.1　描述

設置 UARTHS 中斷回調函數。

11.3.6.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
interrupt_mode	中斷類型	輸入
uarths_callback	中斷回調函數	輸入
ctx	回調函數的參數	輸入
priority	中斷優先順序	輸入

11.3.6.4　返回值

無。

11.3.7　uarths_get_interrupt_mode

11.3.7.1　描述

獲取 UARTHS 的中斷類型。接收、發送或接收發送同時中斷。

11.3.7.2　函數原型

```
uarths_interrupt_mode_t uarths_get_interrupt_mode(void)
```

11.3.7.3　參數

無

11.3.7.4　返回值

當前中斷的類型。

11.3.8　uarths_set_interrupt_cnt

11.3.8.1　描述

設置 UARTHS 中斷時的 FIFO 深度。當中斷類型為 UARTHS_SEND_RECEIVE，發送接收 FIFO 中斷深度均為 cnt;

11.3.8.2　函數原型

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

11.3.8.3　參數

參數名稱	描述	輸入輸出
interrupt_mode	中斷類型	輸入
cnt	FIFO 深度	輸入

11.3.8.4　返回值

無。

11.3.9　舉例

```
/* 設置接收中斷 中斷FIFO深度為0，即接收到資料立即中斷並讀取接收到的資料。*/
    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　資料類型

相關資料類型、資料結構定義如下：

- uarths_interrupt_mode_t：中斷類型。
- uarths_stopbit_t：停止位。

11.4.1　uarths_interrupt_mode_t

11.4.2　描述

UARTHS 中斷類型。

11.4.2.1　定義

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

11.4.2.2　成員

成員名稱	描述
UARTHS_SEND	發送中斷
UARTHS_RECEIVE	接收中斷
UARTHS_SEND_RECEIVE	發送接收中斷

11.4.3　uarths_stopbit_t

11.4.3.1　描述：

UARTHS 停止位。

11.4.3.2　定義

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

11.4.3.3　成員

成員名稱	描述
UART_STOP_1	1 位停止位
UART_STOP_2	2 位停止位

第 12 章

看門狗定時器（WDT）

12.1 概述

WDT 提供系統出錯或無響應時的恢復功能。

12.2 功能描述

WDT 模組具有以下功能：

- 配置超時時間
- 手動重啟計時

12.3 API 參考

對應的頭文件 `wdt.h`
為用戶提供以下介面

- `wdt_init`
- `wdt_start`(0.6.0 後不再支持，請使用 `wdt_init`)
- `wdt_stop`
- `wdt_feed`
- `wdt_clear_interrupt`

12.3.1　wdt_init

12.3.1.1　描述

配置參數，啟動看門狗。不使用中斷的話，將 on_irq 設置為 NULL。

12.3.1.2　函數原型

```
uint32_t wdt_init(wdt_device_number_t id, uint64_t time_out_ms, plic_irq_callback_t
    on_irq, void *ctx)
```

12.3.1.3　參數

參數名稱	描述	輸入輸出
id	看門狗編號	輸入
time_out_ms	超時時間（毫秒）	輸入
on_irq	中斷回調函數	輸入
ctx	回調函數參數	輸入

12.3.1.4　返回值

看門狗超時重啟的實際時間（毫秒）。與 time_out_ms 有差異，一般情況會大於這個時間。在外部晶振 26M 的情況下，最大超時時間為 330 毫秒。

12.3.2　wdt_start

12.3.2.1　描述

啟動看門狗。

12.3.2.2　函數原型

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

12.3.2.3　參數

參數名稱	描述	輸入輸出
id	看門狗編號	輸入

參數名稱	描述	輸入輸出
time_out_ms	超時時間（毫秒）	輸入
on_irq	中斷回調函數	輸入

12.3.2.4　返回值

無

12.3.3　wdt_stop

12.3.3.1　描述

關閉看門狗。

12.3.3.2　函數原型

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

12.3.3.3　參數

參數名稱	描述	輸入輸出
id	看門狗編號	輸入

12.3.3.4　返回值

無。

12.3.4　wdt_feed

12.3.4.1　描述

喂狗。

12.3.4.2　函數原型

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

12.3.4.3　參數

參數名稱	描述	輸入輸出
id	看門狗編號	輸入

12.3.4.4　返回值

無。

12.3.5　wdt_clear_interrupt

12.3.5.1　描述

清除中斷。如果在中斷函數中清除中斷，則看門狗不會重啟。

12.3.5.2　函數原型

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

12.3.5.3　參數

參數名稱	描述	輸入輸出
id	看門狗編號	輸入

12.3.5.4　返回值

無。

12.3.6　舉例

```
/* 2秒後進入看門狗中斷函數列印Hello_world,再過2s複位 */
int wdt0_irq(void *ctx)
{
    printf("Hello_world\n");
    return 0;
}
plic_init();
sysctl_enable_irq();
wdt_init(WDT_DEVICE_0, 2000, wdt0_irq, NULL);
```

12.4　資料類型

相關資料類型、資料結構定義如下：

- wdt_device_number_t

12.4.1　wdt_device_number_t

12.4.1.1　描述

看門狗編號。

12.4.1.2　定義

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

12.4.1.3　成員

成員名稱	描述
WDT_DEVICE_0	看門狗 0
WDT_DEVICE_1	看門狗 1

第 13 章

直接內部儲存存取控制器 (DMAC)

13.1 概述

直接存儲訪問 (Direct Memory Access, DMA) 用於在外部裝置與記憶體之間以及記憶體與記憶體之間提供高速資料傳輸。可以在無需任何 CPU 操作的情況下通過 DMA 快速移動資料，從而提高了 CPU 的效率。

13.2 功能描述

DMA 模組具有以下功能:

- 自動選擇一路空閑的 DMA 通道用於傳輸
- 根據源地址和目標地址自動選擇軟體或硬體握手協議
- 支持 1、2、4、8 位元組的元素大小，源和目標大小不必一致
- 非同步或同步傳輸功能
- 循環傳輸功能，常用於刷新屏幕或音頻錄放等場景

13.3 API 參考

對應的頭文件 dmac.h
為用戶提供以下介面

- 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 描述

初始化 DMA。

13.3.1.2 函數原型

```
void dmac_init(void)
```

13.3.1.3 參數

無。

13.3.1.4 返回值

無。

13.3.2 dmac_set_single_mode

13.3.2.1 描述

設置單路 DMA 參數。

13.3.2.2 函數原型

```
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 參數

參數名稱	描述	輸入輸出
channel_num	DMA 通道號	輸入
src	源地址	輸入

參數名稱	描述	輸入輸出
dest	目標地址	輸出
src_inc	源地址是否自增	輸入
dest_inc	目標地址是否自增	輸入
dmac_burst_size	突發傳輸數量	輸入
dmac_trans_width	單次傳輸資料位寬	輸入
block_size	傳輸資料的個數	輸入

13.3.2.4　返回值

無。

13.3.3　dmac_is_done

13.3.3.1　描述

用於 DMAC 啟動後判斷是否完成傳輸。用於 DMAC 啟動傳輸後，如果在啟動前判斷會不准確。

13.3.3.2　函數原型

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

13.3.3.3　參數

參數名稱	描述	輸入輸出
channel_num	DMA 通道號	輸入

13.3.3.4　返回值

返回值	描述
0	未完成
1	已完成

13.3.4　dmac_wait_done

13.3.4.1　描述

等待 DMA 完成工作。

13.3.4.2　函數原型

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

13.3.4.3　參數

參數名稱	描述	輸入輸出
channel_num	DMA 通道號	輸入

13.3.4.4　返回值

無。

13.3.5　dmac_set_irq

13.3.5.1　描述

設置 DMAC 中斷的回調函數

13.3.5.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
channel_num	DMA 通道號	輸入
dmac_callback	中斷回調函數	輸入
ctx	回調函數的參數	輸入
priority	中斷優先順序	輸入

13.3.5.4　返回值

無。

13.3.6　dmac_set_src_dest_length

13.3.6.1　描述

設置 DMAC 的源地址、目的地址和長度，然後啟動 DMAC 傳輸。如果 src 為 NULL 則不設置源地址，dest 為 NULL 則不設置目的地址，len<=0 則不設置長度。

該函數一般用於 DMAC 中斷中，使 DMA 繼續傳輸資料，而不必再次設置 DMAC 的所有參數以節省時間。

13.3.6.2　函數原型

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

13.3.6.3　參數

參數名稱	描述	輸入輸出
channel_num	DMA 通道號	輸入
src	中斷回調函數	輸入
dest	回調函數的參數	輸入
len	傳輸長度	輸入

13.3.6.4　返回值

無。

13.3.7　dmac_is_idle

13.3.7.1　描述

判斷 DMAC 當前通道是否空閑，該函數在傳輸前和傳輸後都可以用來判斷 DMAC 狀態。

13.3.7.2　函數原型

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

13.3.7.3　參數

參數名稱	描述	輸入輸出
channel_num	DMA 通道號	輸入

13.3.7.4　返回值

返回值	描述
0	忙
1	空閑

13.3.8　dmac_wait_idle

13.3.8.1　描述

等待 DMAC 進入空閑狀態。

13.3.8.2　參數

參數名稱	描述	輸入輸出
channel_num	DMA 通道號	輸入

13.3.8.3　返回值

無。

13.3.9　舉例

```
/* I2C通過DMA發送128個int資料 */
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　資料類型

相關資料類型、資料結構定義如下：

- dmac_channel_number_t: DMA 通道編號。

- dmac_address_increment_t：地址增長方式。
- dmac_burst_trans_length_t：突發傳輸數量。
- dmac_transfer_width_t：單次傳輸資料位數。

13.4.1　dmac_channel_number_t

13.4.1.1　描述

DMA 通道編號。

13.4.1.2　定義

```
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　成員

成員名稱	描述
DMAC_CHANNEL0	DMA 通道 0
DMAC_CHANNEL1	DMA 通道 1
DMAC_CHANNEL2	DMA 通道 2
DMAC_CHANNEL3	DMA 通道 3
DMAC_CHANNEL4	DMA 通道 4
DMAC_CHANNEL5	DMA 通道 5

13.4.2　dmac_address_increment_t

13.4.2.1　描述

地址增長方式。

13.4.2.2　定義

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

13.4.2.3　成員

成員名稱	描述
DMAC_ADDR_INCREMENT	地址自動增長
DMAC_ADDR_NOCHANGE	地址不變

13.4.3　dmac_burst_trans_length_t

13.4.3.1　描述

突發傳輸數量。

13.4.3.2　定義

```
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　成員

成員名稱	描述
DMAC_MSIZE_1	單次傳輸數量乘 1
DMAC_MSIZE_4	單次傳輸數量乘 4
DMAC_MSIZE_8	單次傳輸數量乘 8
DMAC_MSIZE_16	單次傳輸數量乘 16
DMAC_MSIZE_32	單次傳輸數量乘 32
DMAC_MSIZE_64	單次傳輸數量乘 64

成員名稱	描述
DMAC_MSIZE_128	單次傳輸數量乘 128
DMAC_MSIZE_256	單次傳輸數量乘 256

13.4.4　dmac_transfer_width_t

13.4.4.1　描述

單次傳輸資料位數。

13.4.4.2　定義

```
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　成員

成員名稱	描述
DMAC_TRANS_WIDTH_8	單次傳輸 8 位
DMAC_TRANS_WIDTH_16	單次傳輸 16 位
DMAC_TRANS_WIDTH_32	單次傳輸 32 位
DMAC_TRANS_WIDTH_64	單次傳輸 64 位
DMAC_TRANS_WIDTH_128	單次傳輸 128 位
DMAC_TRANS_WIDTH_256	單次傳輸 256 位

第 14 章

集成電路內置匯流排 (I²C)

14.1 概述

I2C 匯流排用於和多個外部裝置進行通信。多個外部裝置可以共用一個 I2C 匯流排。

14.2 功能描述

I2C 模組具有以下功能:

- 獨立的 I2C 裝置封裝外部裝置相關參數
- 自動處理多裝置匯流排爭用

14.3 API 參考

對應的頭文件 i2c.h
為用戶提供以下介面

- 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　描述

配置 I²C 器件從地址、寄存器位寬度和 I²C 速率。

14.3.1.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
i2c_num	I²C 號	輸入
slave_address	I²C 器件從地址	輸入
address_width	I²C 器件寄存器寬度 (7 或 10)	輸入
i2c_clk	I²C 速率 (Hz)	輸入

14.3.1.4　返回值

無。

14.3.2　i2c_init_as_slave

14.3.2.1　描述

配置 I²C 為從模式。

14.3.2.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
i2c_num	I²C 號	輸入
slave_address	I²C 從模式的地址	輸入

參數名稱	描述	輸入輸出
address_width	I²C 器件寄存器寬度 (7 或 10)	輸入
handler	I²C 從模式的中斷處理函數	輸入

14.3.2.4　返回值

無。

14.3.3　i2c_send_data

14.3.3.1　描述

寫資料。

14.3.3.2　函數原型

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

14.3.3.3　參數

參數名稱	描述	輸入輸出
i2c_num	I²C 號	輸入
send_buf	待傳輸資料	輸入
send_buf_len	待傳輸資料長度	輸入

14.3.3.4　返回值

返回值	描述
0	成功
非 0	失敗

14.3.4　i2c_send_data_dma

14.3.4.1　描述

通過 DMA 寫資料。

14.3.4.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_channel_num	使用的 dma 通道號	輸入
i2c_num	I²C 號	輸入
send_buf	待傳輸資料	輸入
send_buf_len	待傳輸資料長度	輸入

14.3.4.4　返回值

無

14.3.5　i2c_recv_data

14.3.5.1　描述

通過 CPU 讀資料。

14.3.5.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
i2c_num	I²C 匯流排號	輸入
send_buf	待傳輸資料，一般情況是 i2c 外部裝置的寄存器，如果沒有設置為 NULL	輸入
send_buf_len	待傳輸資料長度，如果沒有則寫 0	輸入
receive_buf	接收資料內部儲存	輸出
receive_buf_len	接收資料的長度	輸入

14.3.5.4　返回值

返回值	描述
0	成功
非 0	失敗

14.3.6　i2c_recv_data_dma

14.3.6.1　描述

通過 dma 讀資料。

14.3.6.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_send_channel_num	發送資料使用的 dma 通道	輸入
dma_receive_channel_num	接收資料使用的 dma 通道	輸入
i2c_num	I²C 匯流排號	輸入
send_buf	待傳輸資料，一般情況是 i2c 外部裝置的寄存器，如果沒有設置為 NULL	輸入
send_buf_len	待傳輸資料長度，如果沒有則寫 0	輸入
receive_buf	接收資料內部儲存	輸出
receive_buf_len	接收資料的長度	輸入

14.3.6.4　返回值

無

14.3.7　舉例

```
/* i2c外部裝置地址是0x32, 7位地址，速率200K */
i2c_init(I2C_DEVICE_0, 0x32, 7, 200000);
uint8_t reg = 0;
```

```
uint8_t data_buf[2] = {0x00,0x01}
data_buf[0] = reg;
/* 向0寄存器寫0x01 */
i2c_send_data(I2C_DEVICE_0, data_buf, 2);
i2c_send_data_dma(DMAC_CHANNEL0, I2C_DEVICE_0, data_buf, 4);
/* 從0寄存器讀取1位元組資料 */
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　資料類型

相關資料類型、資料結構定義如下：

- i2c_device_number_t：i2c 號。
- i2c_slave_handler_t：i2c 從模式的中斷處理函數句柄

14.4.1　i2c_device_number_t

14.4.1.1　描述

i2c 編號。

14.4.1.2　定義

```
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　描述

i2c 從模式的中斷處理函數句柄。根據不同的中斷狀態執行相應的函數操作。

14.4.2.2　定義

```
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　成員

成員名稱	描述
I2C_DEVICE_0	I2C 0
I2C_DEVICE_1	I2C 1
I2C_DEVICE_2	I2C 2

第 15 章

串列外部裝置介面 (SPI)

15.1 概述

SPI 是一種高速的，全雙工，同步的通信匯流排。

15.2 功能描述

SPI 模組具有以下功能:

- 獨立的 SPI 裝置封裝外部裝置相關參數
- 自動處理多裝置匯流排爭用
- 支持標準、雙線、四線、八線模式
- 支持先寫後讀和全雙工讀寫
- 支持發送一串相同的資料幀，常用於清屏、填充存儲扇區等場景

15.3 API 參考

對應的頭文件 `spi.h`
為用戶提供以下介面

- `spi_init`
- `spi_init_non_standard`
- `spi_send_data_standard`
- `spi_send_data_standard_dma`
- `spi_receive_data_standard`
- `spi_receive_data_standard_dma`

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

15.3.1 spi_init

15.3.1.1 描述

設置 SPI 工作模式、多線模式和位寬。

15.3.1.2 函數原型

```
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 參數

參數名稱	描述	輸入輸出
spi_num	SPI 號	輸入
work_mode	極性相位的四種模式	輸入
frame_format	多線模式	輸入
data_bit_length	單次傳輸的資料的位寬	輸入
endian	大小端 0：小端 1：大端	輸入

15.3.1.4 返回值

無。

15.3.2 spi_config_non_standard

15.3.2.1 描述

多線模式下設置指令長度、地址長度、等待時脈數、指令地址傳輸模式。

15.3.2.2 函數原型

```
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　參數

參數名稱	描述	輸入輸出
spi_num	SPI 號	輸入
instruction_length	發送指令的位數	輸入
address_length	發送地址的位數	輸入
wait_cycles	等待時脈個數	輸入
instruction_address_trans_mode	指令地址傳輸的方式	輸入

15.3.2.4　返回值

無

15.3.3　spi_send_data_standard

15.3.3.1　描述

SPI 標準模式傳輸資料。

15.3.3.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
spi_num	SPI 號	輸入
chip_select	片選信號	輸入
cmd_buff	外部裝置指令地址資料，沒有則設為 NULL	輸入
cmd_len	外部裝置指令地址資料長度，沒有則設為 0	輸入
tx_buff	發送的資料	輸入
tx_len	發送資料的長度	輸入

15.3.3.4　返回值

無

15.3.4　spi_send_data_standard_dma

15.3.4.1　描述

SPI 標準模式下使用 DMA 傳輸資料。

15.3.4.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
channel_num	DMA 通道號	輸入
spi_num	SPI 號	輸入
chip_select	片選信號	輸入
cmd_buff	外部裝置指令地址資料，沒有則設為 NULL	輸入
cmd_len	外部裝置指令地址資料長度，沒有則設為 0	輸入
tx_buff	發送的資料	輸入
tx_len	發送資料的長度	輸入

15.3.4.4　返回值

無

15.3.5　spi_receive_data_standard

15.3.5.1　描述

標準模式下接收資料。

15.3.5.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
spi_num	SPI 號	輸入
chip_select	片選信號	輸入
cmd_buff	外部裝置指令地址資料，沒有則設為 NULL	輸入
cmd_len	外部裝置指令地址資料長度，沒有則設為 0	輸入
rx_buff	接收的資料	輸出
rx_len	接收資料的長度	輸入

15.3.5.4　返回值

無

15.3.6　spi_receive_data_standard_dma

15.3.6.1　描述

標準模式下通過 DMA 接收資料。

15.3.6.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_send_channel_num	發送指令地址使用的 DMA 通道號	輸入
dma_receive_channel_num	接收資料使用的 DMA 通道號	輸入
spi_num	SPI 號	輸入
chip_select	片選信號	輸入
cmd_buff	外部裝置指令地址資料，沒有則設為 NULL	輸入
cmd_len	外部裝置指令地址資料長度，沒有則設為 0	輸入
rx_buff	接收的資料	輸出
rx_len	接收資料的長度	輸入

15.3.6.4　返回值

無

15.3.7　spi_send_data_multiple

15.3.7.1　描述

多線模式發送資料。

15.3.7.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
spi_num	SPI 號	輸入
chip_select	片選信號	輸入
cmd_buff	外部裝置指令地址資料，沒有則設為 NULL	輸入
cmd_len	外部裝置指令地址資料長度，沒有則設為 0	輸入
tx_buff	發送的資料	輸入
tx_len	發送資料的長度	輸入

15.3.7.4　返回值

無

15.3.8　spi_send_data_multiple_dma

15.3.8.1　描述

多線模式使用 DMA 發送資料。

15.3.8.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
channel_num	DMA 通道號	輸入
spi_num	SPI 號	輸入
chip_select	片選信號	輸入
cmd_buff	外部裝置指令地址資料，沒有則設為 NULL	輸入
cmd_len	外部裝置指令地址資料長度，沒有則設為 0	輸入
tx_buff	發送的資料	輸入
tx_len	發送資料的長度	輸入

15.3.8.4　返回值

無

15.3.9　spi_receive_data_multiple

15.3.9.1　描述

多線模式接收資料。

15.3.9.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
spi_num	SPI 號	輸入
chip_select	片選信號	輸入
cmd_buff	外部裝置指令地址資料，沒有則設為 NULL	輸入
cmd_len	外部裝置指令地址資料長度，沒有則設為 0	輸入
rx_buff	接收的資料	輸出
rx_len	接收資料的長度	輸入

15.3.9.4　返回值

無

15.3.10　spi_receive_data_multiple_dma

15.3.10.1　描述

多線模式通過 DMA 接收。

15.3.10.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
dma_send_channel_num	發送指令地址使用的 DMA 通道號	輸入
dma_receive_channel_num	接收資料使用的 DMA 通道號	輸入
spi_num	SPI 號	輸入
chip_select	片選信號	輸入
cmd_buff	外部裝置指令地址資料，沒有則設為 NULL	輸入
cmd_len	外部裝置指令地址資料長度，沒有則設為 0	輸入
rx_buff	接收的資料	輸出
rx_len	接收資料的長度	輸入

15.3.10.4　返回值

無

15.3.11　spi_fill_data_dma

15.3.11.1　描述

通過 DMA 始終發送同一個資料，可以用於刷新資料。

15.3.11.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
channel_num	DMA 通道號	輸入
spi_num	SPI 號	輸入
chip_select	片選信號	輸入
tx_buff	發送的資料，僅發送 tx_buff 這一個資料，不會自動增加	輸入
tx_len	發送資料的長度	輸入

15.3.11.4　返回值

無

15.3.12　spi_send_data_normal_dma

15.3.12.1　描述

通過 DMA 發送資料。不用設置指令地址。

15.3.12.2　函數原型

```
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　參數

參數名稱	描述	輸入輸出
channel_num	DMA 通道號	輸入
spi_num	SPI 號	輸入
chip_select	片選信號	輸入
tx_buff	發送的資料，僅發送 tx_buff 這一個資料，不會自動增加	輸入
tx_len	發送資料的長度	輸入
spi_transfer_width	發送資料的位寬	輸入

15.3.12.4　返回值

無

15.3.13　舉例

```
/* SPI0 工作在MODE0模式 標準SPI模式 單次發送8位資料 */
spi_init(SPI_DEVICE_0, SPI_WORK_MODE_0, SPI_FF_STANDARD, 8, 0);
uint8_t cmd[4];
cmd[0] = 0x06;
cmd[1] = 0x01;
cmd[2] = 0x02;
cmd[3] = 0x04;
uint8_t data_buf[4] = {0,1,2,3};
/* SPI0 使用片選0 發送指令0x06 向地址0x010204 發送0，1，2，3 四個位元組資料 */
spi_send_data_standard(SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 4, data_buf, 4);
/* SPI0 使用片選0 發送指令0x06 地址0x010204 接收4個位元組的資料 */
spi_receive_data_standard(SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 4, data_buf, 4);

/* SPI0 工作在MODE0模式 四線SPI模式 單次發送8位資料 */
spi_init(SPI_DEVICE_0, SPI_WORK_MODE_0, SPI_FF_QUAD, 8, 0);
/* 8位指令長度 32位地址長度 發送指令地址後等待4個clk，指令通過標準SPI方式發送，地址通過
    四線方式發送 */
spi_init_non_standard(SPI_DEVICE_0, 8, 32, 4, SPI_AITM_ADDR_STANDARD);
uint32 cmd[2];
cmd[0] = 0x06;
cmd[1] = 0x010204;
uint8_t data_buf[4] = {0,1,2,3};
/* SPI0 使用片選0 發送指令0x06 向地址0x010204 發送0，1，2，3 四個位元組資料 */
spi_send_data_multiple(SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 2, data_buf, 4);
/* SPI0 使用片選0 發送指令0x06 地址0x010204 接收4個位元組的資料 */
spi_receive_data_multiple(SPI_DEVICE_0, SPI_CHIP_SELECT_0, cmd, 2, data_buf, 4);

/* SPI0 工作在MODE2模式 八線SPI模式 單次發送32位資料 */
spi_init(SPI_DEVICE_0, SPI_WORK_MODE_2, SPI_FF_OCTAL, 32, 0);
/* 無指令 32位地址長度 發送指令地址後等待0個clk，指令地址通過8線發送 */
spi_init_non_standard(SPI_DEVICE_0, 0, 32, 0, SPI_AITM_AS_FRAME_FORMAT);
uint32_t data_buf[256] = {0};
/* 使用DMA通道0 片選0 發送256個int資料*/
spi_send_data_normal_dma(DMAC_CHANNEL0, SPI_DEVICE_0, SPI_CHIP_SELECT_0, data_buf, 256,
    SPI_TRANS_INT);
uint32_t data = 0x55AA55AA;
/* 使用DMA通道0 片選0 連續發送256個 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　描述

設置 SPI 的時脈頻率

15.3.14.2　函數原型

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

15.3.14.3　參數

參數名稱	描述	輸入輸出
spi_num	SPI 號	輸入
spi_clk	目標 SPI 裝置的時脈頻率	輸入

15.3.14.4　返回值

設置完後的 SPI 裝置的時脈頻率

15.4　資料類型

相關資料類型、資料結構定義如下：

- spi_device_num_t: SPI 編號。
- spi_mode_t: SPI 模式。
- spi_frame_format_t: SPI 幀格式。
- spi_instruction_address_trans_mode_t: SPI 指令和地址的傳輸模式。

15.4.1　spi_device_num_t

15.4.1.1　描述

SPI 編號。

15.4.1.2　定義

```
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　成員

成員名稱	描述
SPI*DEVICE*0	SPI 0 做為主裝置
SPI*DEVICE*1	SPI 1 做為主裝置
SPI*DEVICE*2	SPI 2 做為從裝置
SPI*DEVICE*3	SPI 3 做為主裝置

15.4.2　spi_mode_t

15.4.2.1　描述

SPI 模式。

15.4.2.2　定義

```
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　成員

成員名稱	描述
SPI_WORK_MODE_0	SPI 模式 0
SPI_WORK_MODE_1	SPI 模式 1
SPI_WORK_MODE_2	SPI 模式 2
SPI_WORK_MODE_3	SPI 模式 3

15.4.3　spi_frame_format_t

15.4.3.1　描述

SPI 幀格式。

15.4.3.2　定義

```
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　成員

成員名稱	描述
SPI_FF_STANDARD	標準
SPI_FF_DUAL	雙線
SPI_FF_QUAD	四線
SPI_FF_OCTAL	八線 (SPI3 不支持)

15.4.4　spi_instruction_address_trans_mode_t

15.4.4.1　描述

SPI 指令和地址的傳輸模式。

15.4.4.2　定義

```
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　成員

成員名稱	描述
SPI_AITM_STANDARD	均使用標準幀格式
SPI_AITM_ADDR_STANDARD	指令使用配置的值，地址使用標準幀格式
SPI_AITM_AS_FRAME_FORMAT	均使用配置的值

第 16 章

集成電路內置音頻匯流排（I2S）

16.1 概述

I2S 標準匯流排定義了三種信號：時脈信號 BCK、聲道選擇信號 WS 和串列資料信號 SD。一個基本的 I2S 資料匯流排有一個主機和一個從機。主機和從機的角色在通信過程中保持不變。I2S 模組包含獨立的發送和接收聲道，能夠保證優良的通信性能。

16.2 功能描述

I2S 模組具有以下功能：

- 根據音頻格式自動配置裝置（支持 16、24、32 位深，44100 採樣率，1 - 4 聲道）
- 可配置為播放或錄音模式
- 自動管理音頻緩衝區

16.3 API 參考

對應的頭文件 `i2s.h`
為用戶提供以下介面

- `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 設置採樣率。
- i2s_set_dma_divide_16：設置 dma*divide*16，16 位資料時設置 dma*divide*16，DMA 傳輸時自動將 32 比特 INT32 資料分成兩個 16 比特的左右聲道資料。
- i2s_get_dma_divide_16：獲取 dma*divide*16 值。用於判斷是否需要設置 dma*divide*16。

16.3.1　i2s_init

16.3.1.1　描述

初始化 I2S。

16.3.1.2　函數原型

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

16.3.1.3　參數

成員名稱	描述	輸入輸出
device_num	I2S 號	輸入
rxtx_mode	接收或發送模式	輸入
channel_mask	通道掩碼	輸入

16.3.1.4　返回值

無。

16.3.2　i2s_send_data_dma

16.3.2.1　描述

I2S 發送資料。

16.3.2.2　函數原型

```
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　參數

成員名稱	描述	輸入輸出
device_num	I2S 號	輸入
buf	發送資料地址	輸入
buf_len	資料長度	輸入
channel_num	DMA 通道號	輸入

16.3.2.4　返回值

無。

16.3.3　i2s_recv_data_dma

16.3.3.1　描述

I2S 接收資料。

16.3.3.2　函數原型

```
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　參數

成員名稱	描述	輸入輸出
device_num	I2S 號	輸入
buf	接收資料地址	輸出
buf_len	資料長度	輸入
channel_num	DMA 通道號	輸入

16.3.3.4　返回值

無

16.3.4　i2s_rx_channel_config

16.3.4.1　描述

設置接收通道參數。

16.3.4.2　函數原型

```
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　參數

成員名稱	描述	輸入輸出
device_num	I2S 號	輸入
channel_num	通道號	輸入
word_length	接收資料位數	輸出
word_select_size	單個資料時脈數	輸入
trigger_level	DMA 觸發時 FIFO 深度	輸入
word_mode	工作模式	輸入

16.3.4.4　返回值

無。

16.3.5　i2s_tx_channel_config

16.3.5.1　描述

設置發送通道參數。

16.3.5.2　函數原型

```
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　參數

成員名稱	描述	輸入輸出
device_num	I2S 號	輸入
channel_num	通道號	輸入
word_length	接收資料位數	輸出

成員名稱	描述	輸入輸出
word_select_size	單個資料時脈數	輸入
trigger_level	DMA 觸發時 FIFO 深度	輸入
word_mode	工作模式	輸入

16.3.5.4　返回值

無。

16.3.6　i2s_play

16.3.6.1　描述

發送 PCM 資料，比如播放音樂

16.3.6.2　函數原型

```
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　參數

成員名稱	描述	輸入輸出
device_num	I2S 號	輸入
channel_num	通道號	輸入
buf	PCM 資料	輸入
buf_len	PCM 資料長度	輸入
frame	單次發送數量	輸入
bits_per_sample	單次採樣位寬	輸入
track_num	聲道數	輸入

16.3.6.4　返回值

無。

16.3.7　i2s_set_sample_rate

16.3.7.1　描述

設置採樣率。

16.3.7.2　函數原型

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

16.3.7.3　參數

成員名稱	描述	輸入輸出
device_num	I2S 號	輸入
sample_rate	採樣率	輸入

16.3.7.4　返回值

實際的採樣率。

16.3.8　i2s_set_dma_divide_16

16.3.8.1　描述

設置 dma_divide_16，16 位資料時設置 dma_divide_16，DMA 傳輸時自動將 32 比特 INT32 資料分成兩個 16 比特的左右聲道資料。

16.3.8.2　函數原型

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

16.3.8.3　參數

成員名稱	描述	輸入輸出
device_num	I2S 號	輸入
enable	0:禁用 1: 啟動	輸入

16.3.8.4　返回值

返回值	描述
0	成功
非 0	失敗

16.3.9　i2s_get_dma_divide_16

16.3.9.1　描述

獲取 dma_divide_16 值。用於判斷是否需要設置 dma_divide_16。

16.3.9.2　函數原型

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

16.3.9.3　參數

成員名稱	描述	輸入輸出
device_num	I2S 號	輸入

16.3.9.4　返回值

返回值	描述
1	啟動
0	禁用
<0	失敗

16.3.10　舉例

```
/* I2S0 通道0 設置為接收通道，接收16位資料，單次傳輸32個時脈，FIFO深度為4，標準模式。接
    收8組資料*/
/* I2S2 通道1 設置為發送通道，發送16位資料，單次傳輸32個時脈，FIFO深度為4，右對齊模式。
    發送8組資料*/
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　資料類型

相關資料類型、資料結構定義如下：

- i2s_device_number_t：I2S 編號。
- i2s_channel_num_t：I2S 通道號。
- i2s_transmit_t：I2S 傳輸模式。
- i2s_work_mode_t：I2S 工作模式。
- i2s_word_select_cycles_t：I2S 單次傳輸時脈數。
- i2s_word_length_t：I2S 傳輸資料位數。
- i2s_fifo_threshold_t：I2S FIFO 深度。

16.4.1　i2s_device_number_t

16.4.1.1　描述

I2S 編號。

16.4.1.2　定義

```
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　成員

成員名稱	描述
I2S_DEVICE_0	I2S 0
I2S_DEVICE_1	I2S 1
I2S_DEVICE_2	I2S 2

16.4.2　i2s_channel_num_t

16.4.2.1　描述

I2S 通道號。

16.4.2.2　定義

```
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　成員

成員名稱	描述
I2S_CHANNEL_0	I2S 通道 0
I2S_CHANNEL_1	I2S 通道 1
I2S_CHANNEL_2	I2S 通道 2
I2S_CHANNEL_3	I2S 通道 3

16.4.3　i2s_transmit_t

16.4.3.1　描述

I2S 傳輸模式。

16.4.3.2　定義

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

16.4.3.3　成員

成員名稱	描述
I2S_TRANSMITTER	發送模式
I2S_RECEIVER	接收模式

16.4.4　i2s_work_mode_t

16.4.4.1　描述

I2S 工作模式。

16.4.4.2　定義

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

16.4.4.3　成員

成員名稱	描述
STANDARD_MODE	標準模式
RIGHT_JUSTIFYING_MODE	右對齊模式
LEFT_JUSTIFYING_MODE	左對齊模式

16.4.5　i2s_word_select_cycles_t

16.4.5.1　描述

I2S 單次傳輸時脈數。

16.4.5.2　定義

```
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　成員

成員名稱	描述
SCLK_CYCLES_16	16 個時脈

成員名稱	描述
SCLK_CYCLES_24	24 個時脈
SCLK_CYCLES_32	32 個時脈

16.4.6　i2s_word_length_t

16.4.6.1　描述

I2S 傳輸資料位數。

16.4.6.2　定義

```
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　成員

成員名稱	描述
IGNORE_WORD_LENGTH	忽略長度
RESOLUTION_12_BIT	12 位資料長度
RESOLUTION_16_BIT	16 位資料長度
RESOLUTION_20_BIT	20 位資料長度
RESOLUTION_24_BIT	24 位資料長度
RESOLUTION_32_BIT	32 位資料長度

16.4.7　i2s_fifo_threshold_t

16.4.7.1　描述

I2S FIFO 深度。

16.4.7.2　定義

```
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　成員

成員名稱	描述
TRIGGER_LEVEL_1	1 位元組 FIFO 深度
TRIGGER_LEVEL_2	2 位元組 FIFO 深度
TRIGGER_LEVEL_3	3 位元組 FIFO 深度
TRIGGER_LEVEL_4	4 位元組 FIFO 深度
TRIGGER_LEVEL_5	5 位元組 FIFO 深度
TRIGGER_LEVEL_6	6 位元組 FIFO 深度
TRIGGER_LEVEL_7	7 位元組 FIFO 深度
TRIGGER_LEVEL_8	8 位元組 FIFO 深度

成員名稱	描述
TRIGGER_LEVEL_9	9 位元組 FIFO 深度
TRIGGER_LEVEL_10	10 位元組 FIFO 深度
TRIGGER_LEVEL_11	11 位元組 FIFO 深度
TRIGGER_LEVEL_12	12 位元組 FIFO 深度
TRIGGER_LEVEL_13	13 位元組 FIFO 深度
TRIGGER_LEVEL_14	14 位元組 FIFO 深度
TRIGGER_LEVEL_15	15 位元組 FIFO 深度
TRIGGER_LEVEL_16	16 位元組 FIFO 深度

第 17 章

定時器（TIMER）

17.1 概述

晶片有 3 個定時器，每個定時器有 4 路通道。可以配置為 PWM，詳見 PWM 說明。

17.2 功能描述

TIMER 模組具有以下功能：

- 啟用或禁用定時器
- 配置定時器觸發間隔
- 配置定時器觸發處理程序

17.3 API 參考

對應的頭文件 `timer.h`
為用戶提供以下介面

- `timer_init`
- `timer_set_interval`
- `timer_set_irq`（0.6.0 後不再支持，請使用 `timer_irq_register`）
- `timer_set_enable`
- `timer_irq_register`
- `timer_irq_deregister`

17.3.1　timer_init

17.3.1.1　描述

初始化定時器。

17.3.1.2　函數原型

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

17.3.1.3　參數

參數名稱	描述	輸入輸出
timer_number	定時器號	輸入

17.3.1.4　返回值

無。

17.3.2　timer_set_interval

17.3.2.1　描述

設置定時間隔。

17.3.2.2　函數原型

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

17.3.2.3　參數

參數名稱	描述	輸入輸出
timer_number	定時器號	輸入
channel	定時器通道號	輸入
nanoseconds	時間間隔（納秒）	輸入

17.3.2.4　返回值

實際的觸發間隔（納秒）。

17.3.3　timer_set_irq

17.3.3.1　描述

設置定時器觸發中斷回調函數，該函數已廢棄，替代函數為 timer_irq_register。

17.3.3.2　函數原型

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

17.3.3.3　參數

參數名稱	描述	輸入輸出
timer_number	定時器號	輸入
channel	定時器通道號	輸入
func	回調函數	輸入
priority	中斷優先順序	輸入

17.3.3.4　返回值

無。

17.3.4　timer_set_enable

17.3.4.1　描述

啟動禁用定時器。

17.3.4.2　函數原型

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

17.3.4.3　參數

參數名稱	描述	輸入輸出
timer_number	定時器號	輸入
channel	定時器通道號	輸入
enable	啟動禁用定時器 0：禁用 1：啟動	輸入

17.3.4.4　返回值

無。

17.3.5　timer_irq_register

17.3.5.1　描述

註冊定時器觸發中斷回調函數。

17.3.5.2　函數原型

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

17.3.5.3　參數

參數名稱	描述	輸入輸出
device	定時器號	輸入
channel	定時器通道號	輸入
is_single_shot	是否單次中斷	輸入
priority	中斷優先順序	輸入
callback	中斷回調函數	輸入
ctx	回調函數參數	輸入

17.3.5.4　返回值

返回值	描述
0	成功
非 0	失敗

17.3.6　timer_irq_deregister

17.3.6.1　描述

註銷定時器中斷函數。

17.3.6.2　函數原型

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

17.3.6.3　參數

參數名稱	描述	輸入輸出
device	定時器號	輸入
channel	定時器通道號	輸入

17.3.6.4　返回值

返回值	描述
0	成功
非 0	失敗

17.3.7　舉例

```
/* 定時器0 通道0 定時1秒列印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　資料類型

相關資料類型、資料結構定義如下：

- timer_device_number_t: 定時器編號。
- timer_channel_number_t: 定時器通道號。
- timer_callback_t: 定時器回調函數。

17.4.1　timer_device_number_t

17.4.1.1　描述

定時器編號

17.4.1.2　定義

```
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　成員

成員名稱	描述
TIMER_DEVICE_0	定時器 0
TIMER_DEVICE_1	定時器 1
TIMER_DEVICE_2	定時器 2

17.4.2　timer_channel_number_t

17.4.2.1　描述

定時器通道號。

17.4.2.2　定義

```
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　成員

成員名稱	描述
TIMER_CHANNEL_0	定時器通道 0
TIMER_CHANNEL_1	定時器通道 1
TIMER_CHANNEL_2	定時器通道 2
TIMER_CHANNEL_3	定時器通道 3

17.4.3　timer_callback_t

17.4.3.1　描述

定時器回調函數。

17.4.3.2　定義

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

第 18 章

實時時脈 (RTC)

18.1 概述

RTC 是用來計時的單元，在設置時間後具備計時功能。

註意 RTC 模組僅當 PLL0 啟動，並且 CPU 頻率大於 30MHz 時使用

18.2 功能描述

RTC 模組具有以下功能：

- 獲取當前日期時刻
- 設置當前日期時刻

18.3 API 參考

對應的頭文件 rtc.h

為用戶提供以下介面

- rtc_init
- rtc_timer_set
- rtc_timer_get

18.3.1 rtc_init

18.3.1.1 描述

初始化 RTC。

18.3.1.2　函數原型

```
int rtc_init(void)
```

18.3.1.3　參數

無。

18.3.1.4　返回值

返回值	描述
0	成功
非 0	失敗

18.3.2　rtc_timer_set

18.3.2.1　描述

設置日期時間。

18.3.2.2　函數原型

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

18.3.2.3　參數

參數名稱	描述	輸入輸出
year	年	輸入
month	月	輸入
day	日	輸入
hour	時	輸入
minute	分	輸入
second	秒	輸入

18.3.2.4　返回值

無

18.3.3　rtc_timer_get

18.3.3.1　描述

獲取日期時間。

18.3.3.2　函數原型

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

18.3.3.3　參數

參數名稱	描述	輸入輸出
year	年	輸出
month	月	輸出
day	日	輸出
hour	時	輸出
minute	分	輸出
second	秒	輸出

18.3.3.4　返回值

返回值	描述
0	成功
非 0	失敗

18.3.4　舉例

```
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);
```

第 19 章

脈衝寬度調製器 (PWM)

19.1 概述

PWM 用於控制脈衝輸出的占空比。其本質是一個定時器，所以註意設置 PWM 號與通道時不要與 TIMER 定時器衝突。

19.2 功能描述

PWM 模組具有以下功能:

- 配置 PWM 輸出頻率
- 配置 PWM 每個腳位的輸出占空比

19.3 API 參考

對應頭文件 pwm.h
為用戶提供以下介面

- pwm_init
- pwm_set_frequency
- pwm_set_enable

19.3.1 pwm_init

19.3.1.1 描述

初始化 PWM。

19.3.1.2　函數原型

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

19.3.1.3　參數

參數名稱	描述	輸入輸出
pwm_number	pwm 號	輸入

19.3.1.4　返回值

無。

19.3.2　pwm_set_frequency

19.3.2.1　描述

設置頻率及占空比。

19.3.2.2　函數原型

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

19.3.2.3　參數

參數名稱	描述	輸入輸出
pwm_number	PWM 號	輸入
channel	PWM 通道號	輸入
frequency	PWM 輸出頻率	輸入
duty	占空比	輸入

19.3.2.4　返回值

實際輸出頻率。

19.3.3　pwmsetenable

19.3.3.1　描述

啟動禁用 PWM。

19.3.3.2　函數原型

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

19.3.3.3　參數

參數名稱	描述	輸入輸出
pwm_number	PWM 號	輸入
channel	PWM 通道號	輸入
enable	啟動禁用 PWM0：禁用 1：啟動	輸入

19.3.3.4　返回值

無。

19.3.4　舉例

```
/* pwm0 channel 1 輸出 200KHZ占空比為0.5的方波 */
/* 設置IO13作為PWM的輸出腳位 */
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　資料類型

- pwm_device_number_t：pwm 號。
- pwm_channel_number_t：pwm 通道號。

19.4.1　pwm_device_number_t

19.4.1.1　描述

pwm 號。

19.4.1.2　定義

```
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　成員

成員名稱	描述
PWM_DEVICE_0	PWM0
PWM_DEVICE_1	PWM1
PWM_DEVICE_2	PWM2

19.4.2　pwm_channel_number_t

19.4.2.1　描述

pwm 通道號。

19.4.2.2　定義

```
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　成員

成員名稱	描述
PWM_CHANNEL_0	PWM 通道 0
PWM_CHANNEL_1	PWM 通道 1
PWM_CHANNEL_2	PWM 通道 2
PWM_CHANNEL_3	PWM 通道 3

第 20 章

系統控制

20.1 概述

系統控制模組提供對操作系統的配置功能。

20.2 功能描述

系統控制模組具有以下功能：

- 設置 PLL CPU 時脈頻率。
- 設置各個模組時脈的分頻值。
- 獲取各個模組的時脈頻率。
- 啟動、禁用、複位各個模組。
- 設置 DMA 請求源。
- 啟動禁用系統中斷。

20.3 API 參考

對應的頭文件 `sysctl.h`
為用戶提供以下介面

- `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
- sysctl_get_time_us
- sysctl_get_reset_status

20.3.1 sysctl_cpu_set_freq

20.3.1.1 描述

設置 CPU 工作頻率。是通過修改 PLL0 的頻率實現的。

20.3.1.2 函數原型

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

20.3.1.3 參數

參數名稱	描述	輸入輸出
freq	要設置的頻率（Hz）	輸入

20.3.1.4 返回值

設置後的實際頻率（Hz）。

20.3.2 sysctl_set_pll_frequency

20.3.2.1 描述

設置 PLL 頻率。

20.3.2.2　函數原型

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

20.3.2.3　參數

參數名稱	描述	輸入輸出
pll	PLL 編號	輸入
pll_freq	要設置的頻率（Hz）	輸入

20.3.2.4　返回值

設置後的實際頻率（Hz）。

20.3.2.5　函數原型

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

20.3.2.6　參數

參數名稱	描述	輸入輸出
pll	PLL 編號	輸入

20.3.2.7　返回值

對應 PLL 的頻率（Hz）。

20.3.3　sysctl_pll_enable

20.3.3.1　描述

啟動對應的 PLL。

20.3.3.2　函數原型

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

20.3.3.3　參數

參數名稱	描述	輸入輸出
pll	PLL 編號	輸入

20.3.3.4　返回值

返回值	描述
0	成功
非 0	失敗

20.3.4　sysctl_pll_disable

20.3.4.1　描述

禁用對應 PLL。

20.3.4.2　函數原型

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

20.3.4.3　參數

參數名稱	描述	輸入輸出
pll	PLL 編號	輸入

20.3.4.4　返回值

返回值	描述
0	成功
非 0	失敗

20.3.5　sysctl_clock_set_threshold

20.3.5.1　描述

設置對應時脈的分頻值。

20.3.5.2　函數原型

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

20.3.5.3　參數

參數名稱	描述	輸入輸出
which	設置的時脈	輸入
threshold	分頻值	輸入

20.3.5.4　返回值

返回值	描述
0	成功
非 0	失敗

20.3.6　sysctl_clock_get_threshold

20.3.6.1　描述

獲取對應時脈的分頻值。

20.3.6.2　函數原型

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

參數名稱	描述	輸入輸出
which	時脈	輸入

20.3.6.3　返回值

對應時脈的分頻值。

20.3.7　sysctl_clock_set_clock_select

20.3.7.1　描述

設置時脈源。

20.3.7.2　函數原型

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

20.3.7.3　參數

參數名稱	描述	輸入輸出
which	時脈	輸入
select	時脈源	輸入

20.3.7.4　返回值

返回值	描述
0	成功
非 0	失敗

20.3.8　sysctl_clock_get_clock_select

20.3.8.1　描述

獲取時脈對應的時脈源。

20.3.8.2　函數原型

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

20.3.8.3　參數

參數名稱	描述	輸入輸出
which	時脈	輸入

20.3.8.4　返回值

時脈對應的時脈源。

20.3.9　sysctl_clock_get_freq

20.3.9.1　描述

獲取時脈的頻率。

20.3.9.2　函數原型

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

20.3.9.3　參數

參數名稱	描述	輸入輸出
clock	時脈	輸入

20.3.9.4　返回值

時脈的頻率（Hz）

20.3.10　sysctl_clock_enable

20.3.10.1　描述

啟動時脈。PLL 要使用 sysctl_pll_enable。

20.3.10.2　函數原型

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

20.3.10.3　參數

參數名稱	描述	輸入輸出
clock	時脈	輸入

20.3.10.4　返回值

返回值	描述
0	成功
非 0	失敗

20.3.11　sysctl_clock_disable

20.3.11.1　描述

禁用時脈，PLL 使用 sysctl_pll_disable。

20.3.11.2　函數原型

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

20.3.11.3　參數

參數名稱	描述	輸入輸出
clock	時脈	輸入

20.3.11.4　返回值

返回值	描述
0	成功
非 0	失敗

20.3.12　sysctl_reset

20.3.12.1　描述

複位各個模組。

20.3.12.2　函數原型

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

20.3.12.3　參數

參數名稱	描述	輸入輸出
reset	預複位模組	輸入

20.3.12.4　返回值

無。

20.3.13　sysctl_dma_select

20.3.13.1　描述

設置 DMA 請求源。與 DMAC 的 API 配合使用。

20.3.13.2　函數原型

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

20.3.13.3　參數

參數名稱	描述	輸入輸出
channel	DMA 通道號	輸入
select	DMA 請求源	輸入

20.3.13.4　返回值

返回值	描述
0	成功
非 0	失敗

20.3.14　sysctl_set_power_mode

20.3.14.1　描述

設置 FPIOA 的對應電源域的電壓。

20.3.14.2　原型

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

20.3.14.3　參數

參數名稱	描述	輸入輸出
power_bank	IO 電源域編號	輸入
io_power_mode	設置的電壓值 1.8V 或 3.3V	輸入

20.3.14.4　返回值

無。

20.3.15　sysctl_enable_irq

20.3.15.1　描述

啟動系統中斷，如果使用中斷一定要開啟系統中斷。

20.3.15.2　函數原型

```
void sysctl_enable_irq(void)
```

20.3.15.3　參數

無。

20.3.15.4　返回值

無。

20.3.16　sysctl_disable_irq

20.3.16.1　描述

禁用系統中斷。

20.3.16.2　函數原型

```
void sysctl_disable_irq(void)
```

20.3.16.3　參數

無。

20.3.16.4　返回值

無。

20.3.17　sysctl_get_time_us

20.3.17.1　描述

開機至今的時間（微秒）。

20.3.17.2　函數原型

```
uint64_t sysctl_get_time_us(void)
```

20.3.17.3　參數

無。

20.3.17.4　返回值

開機至今的時間（微秒）。

20.3.18　sysctl_get_reset_status

20.3.18.1　描述

獲取複位狀態。參見 sysctl*reset*enum*status*t 說明。

20.3.18.2　函數原型

```
sysctl_reset_enum_status_t sysctl_get_reset_status(void)
```

20.3.18.3　參數

無。

20.3.18.4　返回值

複位狀態。

20.4　資料類型

相關資料類型、資料結構定義如下：

- sysctl_pll_t: PLL 編號。
- sysctl_threshold_t: 設置分頻值時各模組編號。
- sysctl_clock_select_t: 設置時脈源時各模組編號。
- sysctl_clock_t: 各個模組的編號。
- sysctl_reset_t: 複位時各個模組的編號。
- sysctl_dma_channel_t: DMA 通道號。
- sysctl_dma_select_t: DMA 請求源編號。
- sysctl_power_bank_t: 電源域編號。
- sysctl_io_power_mode_t: IO 輸出電壓值。
- sysctl_reset_enum_status_t: 複位狀態。

20.4.1 sysctl_pll_t

20.4.1.1 描述

PLL 編號。

20.4.1.2 定義

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

20.4.1.3 成員

成員名稱	描述
SYSCTL_PLL0	PLL0
SYSCTL_PLL1	PLL1
SYSCTL_PLL2	PLL2

20.4.2 sysctl_threshold_t

20.4.2.1 描述

設置分頻值各模組編號。

20.4.2.2　定義

```
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　成員

成員名稱	描述
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

成員名稱	描述
SYSCTL_THRESHOLD_ROM	ROM
SYSCTL_THRESHOLD_SPI0	SPI0
SYSCTL_THRESHOLD_SPI1	SPI1
SYSCTL_THRESHOLD_SPI2	SPI2
SYSCTL_THRESHOLD_SPI3	SPI3
SYSCTL_THRESHOLD_TIMER0	TIMER0
SYSCTL_THRESHOLD_TIMER1	TIMER1
SYSCTL_THRESHOLD_TIMER2	TIMER2
SYSCTL_THRESHOLD_I2S0	I2S0
SYSCTL_THRESHOLD_I2S1	I2S1
SYSCTL_THRESHOLD_I2S2	I2S2
SYSCTL_THRESHOLD_I2S0_M	I2S0 MCLK
SYSCTL_THRESHOLD_I2S1_M	I2S1 MCLK
SYSCTL_THRESHOLD_I2S2_M	I2S2 MCLK
SYSCTL_THRESHOLD_I2C0	I2C0
SYSCTL_THRESHOLD_I2C1	I2C1
SYSCTL_THRESHOLD_I2C2	I2C2
SYSCTL_THRESHOLD_WDT0	WDT0
SYSCTL_THRESHOLD_WDT1	WDT1

20.4.3 sysctl_clock_select_t

20.4.3.1 描述

設置時脈源時各模組編號。

20.4.3.2 定義

```
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　成員

成員名稱	描述
SYSCTL_CLOCK_SELECT_PLL0_BYPASS	PLL0_BYPASS
SYSCTL_CLOCK_SELECT_PLL1_BYPASS	PLL1_BYPASS
SYSCTL_CLOCK_SELECT_PLL2_BYPASS	PLL2_BYPASS
SYSCTL_CLOCK_SELECT_PLL2	PLL2
SYSCTL_CLOCK_SELECT_ACLK	ACLK
SYSCTL_CLOCK_SELECT_SPI3	SPI3
SYSCTL_CLOCK_SELECT_TIMER0	TIMER0
SYSCTL_CLOCK_SELECT_TIMER1	TIMER1
SYSCTL_CLOCK_SELECT_TIMER2	TIMER2
SYSCTL_CLOCK_SELECT_SPI3_SAMPLE	SPI3 資料採樣時脈沿選擇

20.4.4　sysctl_clock_t

20.4.4.1　描述

各個模組的編號。

20.4.4.2　定義

```
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　成員

成員名稱	描述
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

成員名稱	描述
SYSCTL_CLOCK_FFT	FFT
SYSCTL_CLOCK_GPIO	GPIO
SYSCTL_CLOCK_SPI0	SPI0
SYSCTL_CLOCK_SPI1	SPI1
SYSCTL_CLOCK_SPI2	SPI2
SYSCTL_CLOCK_SPI3	SPI3
SYSCTL_CLOCK_I2S0	I2S0
SYSCTL_CLOCK_I2S1	I2S1
SYSCTL_CLOCK_I2S2	I2S2
SYSCTL_CLOCK_I2C0	I2C0
SYSCTL_CLOCK_I2C1	I2C1
SYSCTL_CLOCK_I2C2	I2C2
SYSCTL_CLOCK_UART1	UART1
SYSCTL_CLOCK_UART2	UART2
SYSCTL_CLOCK_UART3	UART3
SYSCTL_CLOCK_AES	AES
SYSCTL_CLOCK_FPIOA	FPIOA
SYSCTL_CLOCK_TIMER0	TIMER0
SYSCTL_CLOCK_TIMER1	TIMER1
SYSCTL_CLOCK_TIMER2	TIMER2
SYSCTL_CLOCK_WDT0	WDT0
SYSCTL_CLOCK_WDT1	WDT1
SYSCTL_CLOCK_SHA	SHA
SYSCTL_CLOCK_OTP	OTP
SYSCTL_CLOCK_RTC	RTC
SYSCTL_CLOCK_ACLK	ACLK
SYSCTL_CLOCK_HCLK	HCLK
SYSCTL_CLOCK_IN0	外部輸入時脈 IN0

20.4.5 sysctl_reset_t

20.4.5.1 描述

複位時各個模組的編號。

20.4.5.2 定義

```
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　成員

成員名稱	描述
SYSCTL_RESET_SOC	晶片複位
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

成員名稱	描述
SYSCTL_RESET_SPI2	SPI2
SYSCTL_RESET_SPI3	SPI3
SYSCTL_RESET_I2S0	I2S0
SYSCTL_RESET_I2S1	I2S1
SYSCTL_RESET_I2S2	I2S2
SYSCTL_RESET_I2C0	I2C0
SYSCTL_RESET_I2C1	I2C1
SYSCTL_RESET_I2C2	I2C2
SYSCTL_RESET_UART1	UART1
SYSCTL_RESET_UART2	UART2
SYSCTL_RESET_UART3	UART3
SYSCTL_RESET_AES	AES
SYSCTL_RESET_FPIOA	FPIOA
SYSCTL_RESET_TIMER0	TIMER0
SYSCTL_RESET_TIMER1	TIMER1
SYSCTL_RESET_TIMER2	TIMER2
SYSCTL_RESET_WDT0	WDT0
SYSCTL_RESET_WDT1	WDT1
SYSCTL_RESET_SHA	SHA
SYSCTL_RESET_RTC	RTC

20.4.6 sysctl_dma_channel_t

20.4.6.1 描述

DMA 通道號。

20.4.6.2 定義

```
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　成員

成員名稱	描述
SYSCTL_DMA_CHANNEL_0	DMA 通道 0
SYSCTL_DMA_CHANNEL_1	DMA 通道 1
SYSCTL_DMA_CHANNEL_2	DMA 通道 2
SYSCTL_DMA_CHANNEL_3	DMA 通道 3
SYSCTL_DMA_CHANNEL_4	DMA 通道 4
SYSCTL_DMA_CHANNEL_5	DMA 通道 5

20.4.7　sysctl_dma_select_t

20.4.7.1　描述

DMA 請求源編號。

20.4.7.2　定義

```
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　成員

成員名稱	描述
SYSCTL_DMA_SELECT_SSI0_RX_REQ	SPI0 接收
SYSCTL_DMA_SELECT_SSI0_TX_REQ	SPI0 發送
SYSCTL_DMA_SELECT_SSI1_RX_REQ	SPI1 接收
SYSCTL_DMA_SELECT_SSI1_TX_REQ	SPI1 發送
SYSCTL_DMA_SELECT_SSI2_RX_REQ	SPI2 接收
SYSCTL_DMA_SELECT_SSI2_TX_REQ	SPI2 發送
SYSCTL_DMA_SELECT_SSI3_RX_REQ	SPI3 接收
SYSCTL_DMA_SELECT_SSI3_TX_REQ	SPI3 發送
SYSCTL_DMA_SELECT_I2C0_RX_REQ	I2C0 接收
SYSCTL_DMA_SELECT_I2C0_TX_REQ	I2C0 發送
SYSCTL_DMA_SELECT_I2C1_RX_REQ	I2C1 接收
SYSCTL_DMA_SELECT_I2C1_TX_REQ	I2C1 發送
SYSCTL_DMA_SELECT_I2C2_RX_REQ	I2C2 接收
SYSCTL_DMA_SELECT_I2C2_TX_REQ	I2C2 發送
SYSCTL_DMA_SELECT_UART1_RX_REQ	UART1 接收
SYSCTL_DMA_SELECT_UART1_TX_REQ	UART1 發送
SYSCTL_DMA_SELECT_UART2_RX_REQ	UART2 接收
SYSCTL_DMA_SELECT_UART2_TX_REQ	UART2 發送
SYSCTL_DMA_SELECT_UART3_RX_REQ	UART3 接收
SYSCTL_DMA_SELECT_UART3_TX_REQ	UART3 發送
SYSCTL_DMA_SELECT_AES_REQ	AES
SYSCTL_DMA_SELECT_SHA_RX_REQ	SHA 接收
SYSCTL_DMA_SELECT_AI_RX_REQ	AI 接收
SYSCTL_DMA_SELECT_FFT_RX_REQ	FFT 接收
SYSCTL_DMA_SELECT_FFT_TX_REQ	FFT 發送
SYSCTL_DMA_SELECT_I2S0_TX_REQ	I2S0 發送
SYSCTL_DMA_SELECT_I2S0_RX_REQ	I2S0 接收

成員名稱	描述
SYSCTL_DMA_SELECT_I2S1_TX_REQ	I2S1 發送
SYSCTL_DMA_SELECT_I2S1_RX_REQ	I2S1 接收
SYSCTL_DMA_SELECT_I2S2_TX_REQ	I2S2 發送
SYSCTL_DMA_SELECT_I2S2_RX_REQ	I2S2 接收

20.4.8 sysctl_power_bank_t

20.4.8.1 描述

電源域編號。

20.4.8.2 定義

```
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 成員

成員名稱	描述
SYSCTL_POWER_BANK0	電源域 0，控制 IO0-IO5
SYSCTL_POWER_BANK1	電源域 1，控制 IO6-IO11
SYSCTL_POWER_BANK2	電源域 2，控制 IO12-IO17
SYSCTL_POWER_BANK3	電源域 3，控制 IO18-IO23
SYSCTL_POWER_BANK4	電源域 4，控制 IO24-IO29
SYSCTL_POWER_BANK5	電源域 5，控制 IO30-IO35
SYSCTL_POWER_BANK6	電源域 6，控制 IO36-IO41
SYSCTL_POWER_BANK7	電源域 7，控制 IO42-IO47

20.4.9　sysctl_io_power_mode_t

20.4.9.1　描述

IO 輸出電壓值。

20.4.9.2　定義

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

20.4.9.3　成員

成員名稱	描述
SYSCTL_POWER_V33	設置為 3.3V
SYSCTL_POWER_V18	設置為 1.8V

20.4.10　sysctl_reset_enum_status_t

20.4.10.1　描述

複位狀態。

20.4.10.2　定義

```
typedef enum _sysctl_reset_enum_status
{
    SYSCTL_RESET_STATUS_HARD,
    SYSCTL_RESET_STATUS_SOFT,
    SYSCTL_RESET_STATUS_WDT0,
    SYSCTL_RESET_STATUS_WDT1,
    SYSCTL_RESET_STATUS_MAX,
} sysctl_reset_enum_status_t;
```

20.4.10.3　成員

成員名稱	描述
SYSCTL_RESET_STATUS_HARD	硬體複位，重新上電或觸發 reset 腳位
SYSCTL_RESET_STATUS_SOFT	軟體複位

成員名稱	描述
SYSCTL_RESET_STATUS_WDT0	看門狗 0 複位
SYSCTL_RESET_STATUS_WDT1	看門狗 1 複位

第 21 章

平臺相關（BSP）

21.1 概述

平臺相關的通用函數，核之間鎖的相關操作。

21.2 功能描述

提供獲取當前運行程序的 CPU 核編號的介面以及啟動第二個核的入口。

21.3 API 參考

對應的頭文件 `bsp.h`

為用戶提供以下介面

- `register_core1`
- `current_coreid`
- `read_cycle`
- `corelock_lock`
- `corelock_trylock`
- `corelock_unlock`

21.3.1 register_core1

21.3.1.1 描述

向 1 核註冊函數，並啟動 1 核。

21.3.1.2　函數原型

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

21.3.1.3　參數

參數名稱	描述	輸入輸出
func	向 1 核註冊的函數	輸入
ctx	函數的參數，沒有設置為 NULL	輸入

21.3.1.4　返回值

返回值	描述
0	成功
非 0	失敗

21.3.2　current_coreid

21.3.2.1　描述

獲取當前 CPU 核編號。

21.3.2.2　函數原型

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

21.3.2.3　參數

無。

21.3.2.4　返回值

當前所在 CPU 核的編號。

21.3.2.5　read_cycle

21.3.2.6　描述

獲取 CPU 開機至今的時脈數。可以用使用這個函數精準的確定程序運行時脈。可以配合 sysctl_clock_get_freq(SYSCTL_CLOCK_CPU) 計算運行的時間。

21.3.2.7　函數原型

```
#define read_cycle()        read_csr(mcycle)
```

21.3.2.8　參數

無。

21.3.2.9　返回值

開機至今的 CPU 時脈數。

21.3.3　corelock_lock

21.3.3.1　描述

獲取核間鎖，核之間互斥的鎖，同核內該鎖會嵌套，只有異核之間會阻塞。不建議在中斷使用該函數，中斷中可以使用 corelock_trylock。

21.3.3.2　函數原型

```
void corelock_lock(corelock_t *lock)
```

21.3.3.3　參數

核間鎖，要使用全局變數，參見舉例。

21.3.3.4　返回值

無。

21.3.4　corelock_trylock

21.3.4.1　描述

獲取核間鎖，同核時鎖會嵌套，異核時非阻塞。成功獲取鎖會返回 0，失敗返回-1。

21.3.4.2　函數原型

```
corelock_trylock(corelock_t *lock)
```

21.3.4.3　參數

核間鎖，要使用全局變數，參見舉例。

21.3.4.4　返回值

無。

21.3.5　corelock_unlock

21.3.5.1　描述

核間鎖解鎖。

21.3.5.2　函數原型

```
void corelock_unlock(corelock_t *lock)
```

21.3.5.3　參數

核間鎖，要使用全局變數，參見舉例。

21.3.5.4　返回值

無。

21.3.6　舉例

```
/* 1核在0核第二次釋放鎖的時候才會獲取到鎖，通過讀cycle計算時間 */
#include <stdio.h>
#include "bsp.h"
#include <unistd.h>
#include "sysctl.h"

corelock_t lock;

uint64_t get_time(void)
{
```

```
    uint64_t v_cycle = read_cycle();
    return v_cycle * 1000000 / sysctl_clock_get_freq(SYSCTL_CLOCK_CPU);
}

int core1_function(void *ctx)
{
    uint64_t core = current_coreid();
    printf("Core %ld Hello world\n", core);
    while(1)
    {
        uint64_t start = get_time();
        corelock_lock(&lock);
        printf("Core %ld Hello world\n", core);
        sleep(1);
        corelock_unlock(&lock);
        uint64_t stop = get_time();
        printf("Core %ld lock time is %ld us\n",core, stop - start);
        usleep(10);
    }
}

int main(void)
{
    uint64_t core = current_coreid();
    printf("Core %ld Hello world\n", core);
    register_core1(core1_function, NULL);
    while(1)
    {
        corelock_lock(&lock);
        sleep(1);
        printf("1> Core %ld sleep 1\n", core);
        corelock_lock(&lock);
        sleep(2);
        printf("2> Core %ld sleep 2\n", core);
        printf("2> Core unlock\n");
        corelock_unlock(&lock);
        sleep(1);
        printf("1> Core unlock\n");
        corelock_unlock(&lock);
        usleep(10);
    }
}
```

21.4　資料類型

相關資料類型、資料結構定義如下：

- core_function：CPU 核調用的函數。
- spinlock_t：自旋鎖。
- corelock_t：核間鎖。

21.4.1　core_function

21.4.1.1　描述

CPU 核調用的函數。

21.4.1.2　定義

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

21.4.2　spinlock_t

自旋鎖。

21.4.2.1　定義

```
typedef struct _spinlock
{
    int lock;
} spinlock_t;
```

21.4.3　corelock_t

核間鎖。

21.4.3.1　定義

```
typedef struct _corelock
{
    spinlock_t lock;
    int count;
    int core;
} corelock_t;
```