

## 1 简介

本产品高度集成了 AFE、OFDM 调制解调、高处理能力的 CPU，协议处理器、电源模块等功能模块的电力线载波通信信息。采用低功耗，最高性能的设计理念，与高性能处理器内核技术相结合。实现满足各个领域的应用需求。

## 2 特征

- CPU 最高频率可达 200MHz
- 频率动态切换，有效降低功耗
- 内置加解密引擎
- 支持 JATG 在线调试
- 支持全局可屏蔽中断
- 内置 1.5M SRAM
- 支持 SPI，支持 CPU 从 SPI 口 BOOT
- 支持 UART 以及调试功能
- 支持通用 GPIO 功能
- 支持 32 位计数器（定时器）功能
- 以太网芯片 PHY，配有 MDIO 和 RMII 端口
- 支持 DMA 功能
- 配有硬件看门狗
- 一次性写入 eFuse 功能
- 国家电网双模(有线-无线)通信协议

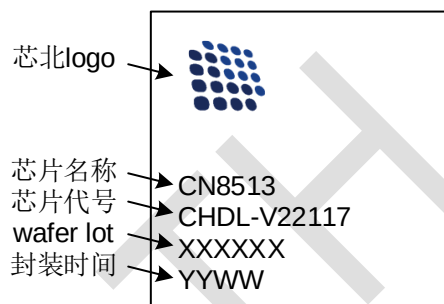
## 3 应用领域

- 抄表系统
- 智能楼宇
- 智能家具
- 工业自动化
- 智慧社区
- 远程监控系统

## 4 丝印

STA 芯片名称: CN8513

CCO 芯片名称: CN8514



注\*: YY/Y=Year; WW/W=Week。

绿色 (RoHS&HF): 芯北科技将“绿色”定义为无铅（符合 RoHS 标准）且不含卤素物质。如果您有其他意见或问题，请直接联系您的芯北代表。

湿敏等级(MSL): 3

## 5 订购信息

| 产品编号   | 封装    | 数量/编带     |
|--------|-------|-----------|
| CN8513 | QFN68 | 3480/Tray |
| CN8514 | QFN88 | 1680/Tray |

## 5.1 引脚排列

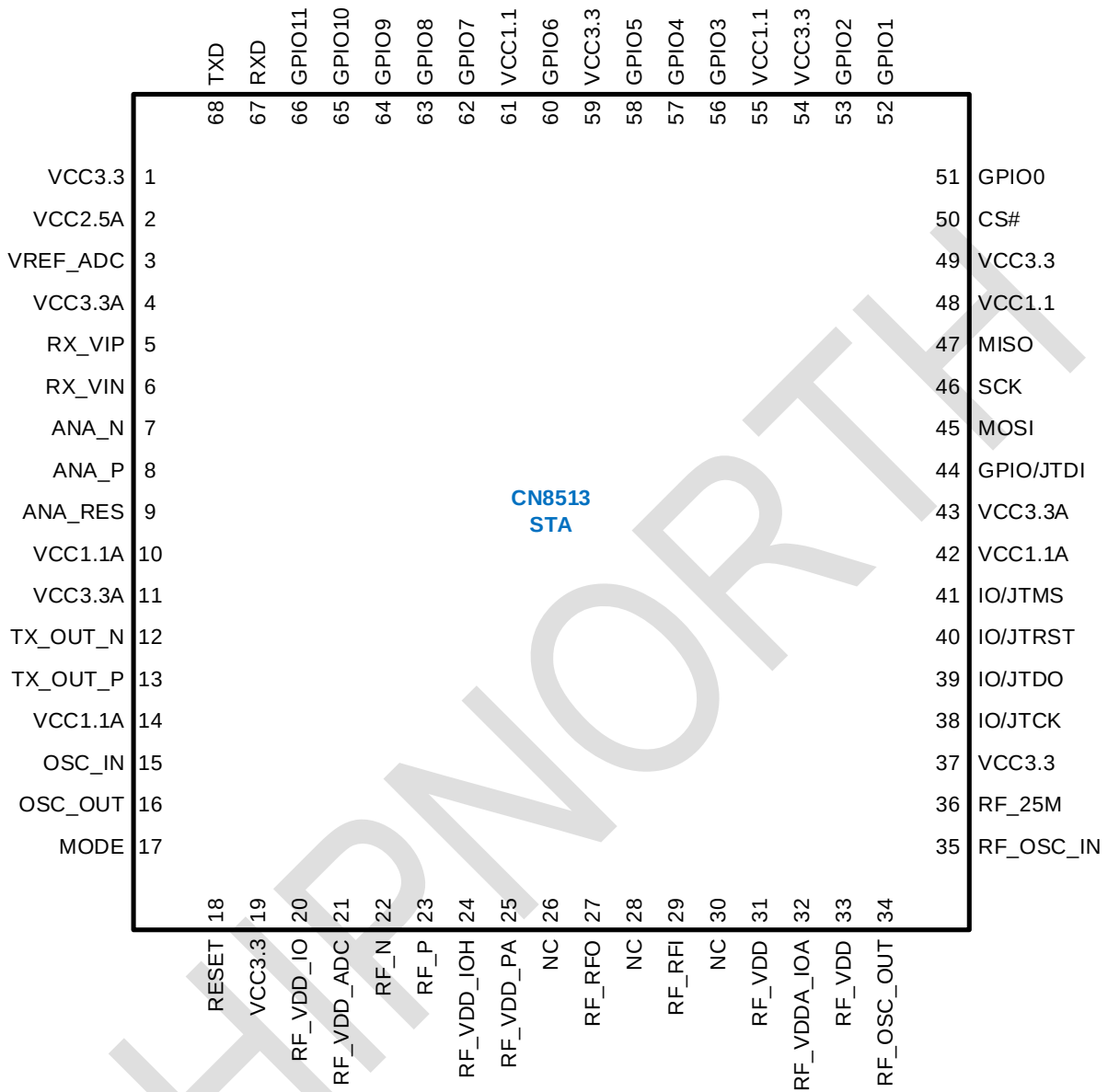


图 5-1 CN8513 STA 引脚图

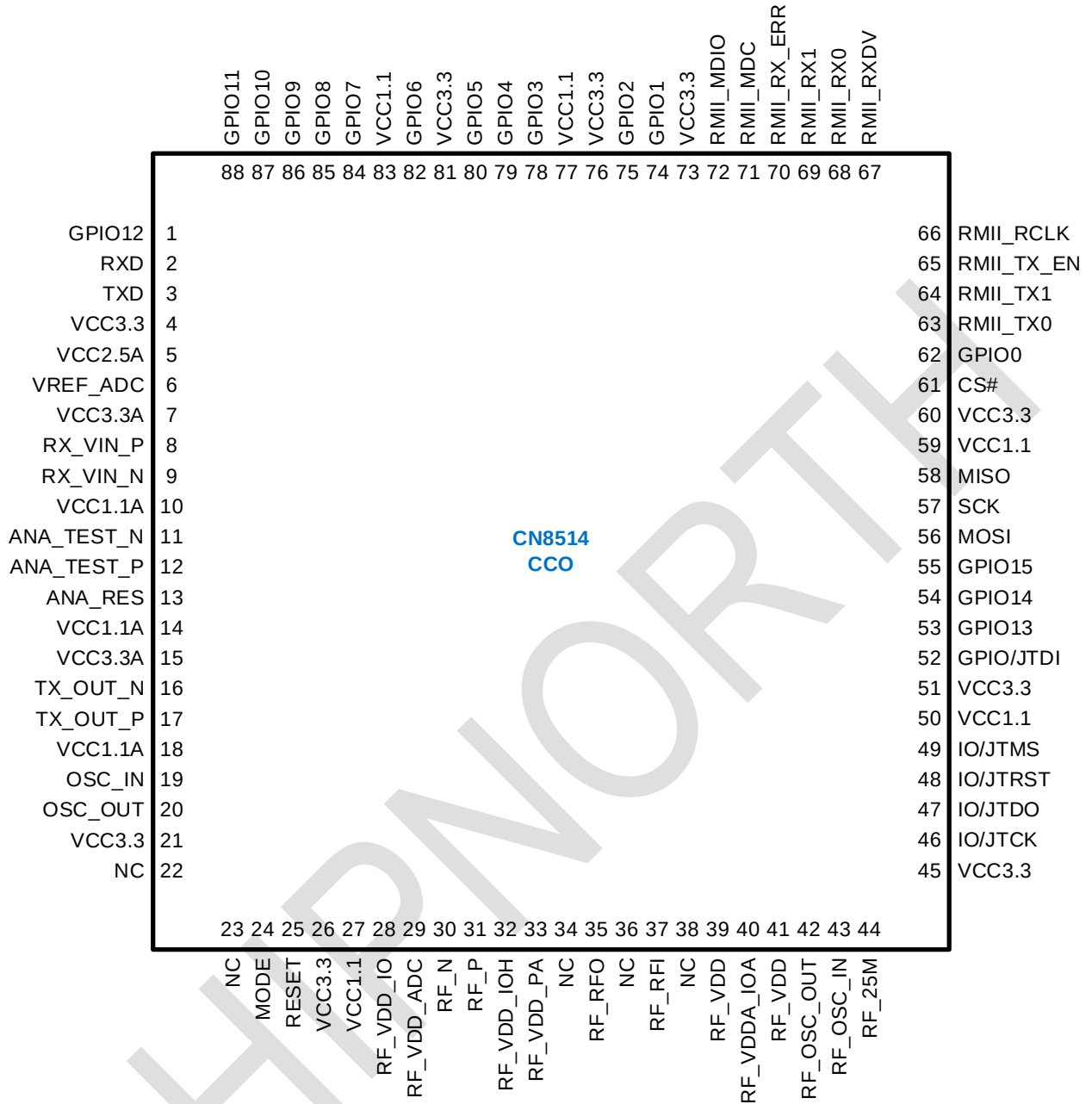


图 5-2 CN8514 CCO 引脚图

## 5.2 引脚描述

### 5.2.1 CN8513 芯片引脚定义列表

| 引脚 | 名称          | 类型  | 描述                    |
|----|-------------|-----|-----------------------|
| 1  | VCC3.3      | P   | 3.3V                  |
| 2  | VCC2.5A     | P   | 2.5V                  |
| 3  | VREF_ADC    | P   | ADC 参考电压              |
| 4  | VCC3.3A     | P   | 3.3V                  |
| 5  | RX_VIP      | I   | 模拟输入，模拟接收             |
| 6  | RX_VIN      | I   | 模拟输入，模拟接收             |
| 7  | ANA_N       | O   | 模拟测试                  |
| 8  | ANA_P       | O   | 模拟测试                  |
| 9  | ANA_RES     | I   | 10K 电阻接地              |
| 10 | VCC1.2A     | P   | 1.2V                  |
| 11 | VCC3.3A     | P   | 3.3V                  |
| 12 | TX_OUT_N    | O   | 模拟输出，模拟发送             |
| 13 | TX_OUT_P    | O   | 模拟输出，模拟发送             |
| 14 | VCC1.2A     | P   | 1.2V                  |
| 15 | OSC_IN      | I   | 晶振输入，25MHZ            |
| 16 | OSC_OUT     | O   | 晶振输出                  |
| 17 | MODE        | I   | 芯片模式                  |
| 18 | RESET       | I   | 复位信号，低电平有效            |
| 19 | VCC3.3      | P   | 3.3V                  |
| 20 | RF_VDD_IO   | I/O | RF VDD_IO1            |
| 21 | RF_VDD_ADC  | I/O | RF VDD_ADC            |
| 22 | RF_N        | I/O | RF 输入输出通道             |
| 23 | RF_P        | I/O | RF 输入输出通道             |
| 24 | RF_VDD_IOH  | I   | LDO 输入                |
| 25 | RF_VDD_PA   | O   | LDO 输出                |
| 26 | NC          | /   | 空                     |
| 27 | RF_RFO      | O   | RF RFO                |
| 28 | NC          | /   | 空                     |
| 29 | RF_RFI      | I   | RF RFI                |
| 30 | NC          | /   | 空                     |
| 31 | RF_VDD      | P   | RF 电源                 |
| 32 | RF_VDDA_IOA | I/O | RF VDD_IOA1 & VDD_IOA |
| 33 | RF_VDD      | P   | RF 电源                 |
| 34 | RF_OSC_OUT  | O   | RF 晶振输出               |
| 35 | RF_OSC_IN   | I   | RF 晶振输入               |
| 36 | RF_25M      | O   | RF 25M clock out      |
| 37 | VCC3.3      | P   | 3.3V                  |
| 38 | IO/JTCK     | I/O | JATG 时钟，GPIO          |
| 39 | IO/JTDO     | I/O | JATG 输出，GPIO          |
| 40 | IO/JTRST    | I/O | JATG 复位，GPIO          |

| 引脚 | 名称        | 类型  | 描述                 |
|----|-----------|-----|--------------------|
| 41 | IO/JTMS   | I/O | JATG 测试模式选择, GPIO  |
| 42 | VCC1.2A   | P   | 1.2V               |
| 43 | VCC3.3A   | P   | 3.3V               |
| 44 | GPIO/JTDI | I/O | JATG 输入, GPIO      |
| 45 | MOSI      | O   | SPI 主机输出           |
| 46 | SCK       | O   | SPI 主机时钟输出         |
| 47 | MISO      | I   | SPI 主机输入           |
| 48 | VCC1.2    | P   | 1.2V               |
| 49 | VCC3.3    | P   | 3.3V               |
| 50 | CS#       | I   | SPI 片选信号           |
| 51 | GPIO0     | I/O | 通用 GPIO 端口, 编号: 0  |
| 52 | GPIO1     | I/O | 通用 GPIO 端口, 编号: 1  |
| 53 | GPIO2     | I/O | 通用 GPIO 端口, 编号: 2  |
| 54 | VCC3.3    | P   | 3.3V               |
| 55 | VCC1.2    | P   | 1.2V               |
| 56 | GPIO3     | I/O | 通用 GPIO 端口, 编号: 3  |
| 57 | GPIO4     | I/O | 通用 GPIO 端口, 编号: 4  |
| 58 | GPIO5     | I/O | 通用 GPIO 端口, 编号: 5  |
| 59 | VCC3.3    | P   | 3.3V               |
| 60 | GPIO6     | I/O | 通用 GPIO 端口, 编号: 6  |
| 61 | VCC1.2    | P   | 1.2V               |
| 62 | GPIO7     | I/O | 通用 GPIO 端口, 编号: 7  |
| 63 | GPIO8     | I/O | 通用 GPIO 端口, 编号: 8  |
| 64 | GPIO9     | I/O | 通用 GPIO 端口, 编号: 9  |
| 65 | GPIO10    | I/O | 通用 GPIO 端口, 编号: 10 |
| 66 | GPIO11    | I/O | 通用 GPIO 端口, 编号: 11 |
| 67 | RXD       | I   | UART 数据接收          |
| 68 | TXD       | O   | UART 数据发送          |

注: DCOUT1.1: 芯片内部输出电压, VCC3.3: 芯片供电电压, VCC1.2: 芯片数字电压, VCC2.5A: UFUSE 供电, VCC1.2A: 模拟电路 1.2V 电源, VCC3.3A: 模拟电路 3.3V 电源。

### 5.2.2 CN8514 芯片引脚定义列表

| 引脚 | 名称          | 类型  | 描述                    |
|----|-------------|-----|-----------------------|
| 1  | GPIO12      | I/O | 通用 GPIO 端口，编号：12      |
| 2  | RXD         | I   | UART 数据接收             |
| 3  | TXD         | O   | UART 数据发送             |
| 4  | VCC3.3      | P   | 3.3V                  |
| 5  | VCC2.5A     | P   | 2.5V                  |
| 6  | VREF_ADC    | P   | ADC 参考电压              |
| 7  | VCC3.3A     | P   | 3.3V                  |
| 8  | RX_VIN_P    | I   | 模拟输入，模拟接收             |
| 9  | RX_VIN_N    | I   | 模拟输入，模拟接收             |
| 10 | VCC1.2A     | P   | 1.2V                  |
| 11 | ANA_N       | P   | 模拟测试                  |
| 12 | ANA_P       | P   | 模拟测试                  |
| 13 | ANA_RES     | I   | 10K 电阻接地              |
| 14 | VCC1.2A     | P   | 1.2V                  |
| 15 | VCC3.3A     | P   | 3.3V                  |
| 16 | TX_OUT_N    | O   | 模拟输出，模拟发送             |
| 17 | TX_OUT_P    | O   | 模拟输出，模拟发送             |
| 18 | VCC1.2A     | P   | 1.2V                  |
| 19 | OSC_IN      | I   | 晶振输入，25MHZ            |
| 20 | OSC_OUT     | O   | 晶振输出                  |
| 21 | VCC3.3      | P   | 3.3V                  |
| 22 | NC          | /   | 空                     |
| 23 | NC          | /   | 空                     |
| 24 | MODE        | I   | 芯片模式                  |
| 25 | RESET       | I   | 复位信号，低电平有效            |
| 26 | VCC3.3      | P   | 3.3V                  |
| 27 | VCC1.2      | P   | 1.2V                  |
| 28 | RF_VDD_IO   | I   | LDO 输入                |
| 29 | RF_VDD_ADC  | I/O | RF VDD_ADC            |
| 30 | RF_N        | I/O | RF 输入输出通道             |
| 31 | RF_P        | I/O | RF 输入输出通道             |
| 32 | RF_VDD_IOH  | I   | LDO 输入                |
| 33 | RF_VDD_PA   | O   | LDO 输出                |
| 34 | NC          | /   | 空                     |
| 35 | RF_RFO      | O   | RF_RFO                |
| 36 | NC          | /   | 空                     |
| 37 | RF_RFI      | I   | RF_RFI                |
| 38 | NC          | /   | 空                     |
| 39 | RF_VDD      | P   | RF 电源                 |
| 40 | RF_VDDA_IOA | I/O | RF VDD_IOA1 & VDD_IOA |
| 41 | RF_VDD      | P   | RF 电源                 |

| 引脚 | 名称          | 类型  | 描述                 |
|----|-------------|-----|--------------------|
| 42 | RF_OSC_OUT  | O   | RF 晶振输出            |
| 43 | RF_OSC_IN   | I   | RF 晶振输入            |
| 44 | RF_25M      | O   | RF 25M clock out   |
| 45 | VCC3.3      | P   | 3.3V               |
| 46 | IO/JTCK     | I/O | JATG 时钟, GPIO      |
| 47 | IO/JTDO     | I/O | JATG 输出, GPIO      |
| 48 | IO/JTRST    | I/O | JATG 复位, GPIO      |
| 49 | IO/JTMS     | I/O | JATG 测试模式选择, GPIO  |
| 50 | VCC1.2      | P   | 1.2V               |
| 51 | VCC3.3      | P   | 3.3V               |
| 52 | GPIO/JTDI   | I/O | JATG 输入, GPIO      |
| 53 | GPIO13      | I/O | 通用 GPIO 端口, 编号: 13 |
| 54 | GPIO14      | I/O | 通用 GPIO 端口, 编号: 14 |
| 55 | GPIO15      | I/O | 通用 GPIO 端口, 编号: 15 |
| 56 | MOSI        | O   | SPI 主机输出           |
| 57 | SCK         | O   | SPI 主机时钟输出         |
| 58 | MISO        | I   | SPI 主机输入           |
| 59 | VCC1.2      | P   | 1.2V               |
| 60 | VCC3.3      | P   | 3.3V               |
| 61 | CS#         | I   | SPI 片选信号           |
| 62 | GPIO0       | I/O | 通用 GPIO 端口, 编号: 0  |
| 63 | RMII_TX0    | O   | RMII 发送数据 0        |
| 64 | RMII_TX1    | O   | RMII 发送数据 1        |
| 65 | RMII_TX_EN  | O   | RMII 发送使能信号        |
| 66 | RMII_RCLK   | I   | RMII 接收时钟          |
| 67 | RMII_RXDV   | I   | RMII 接收数据有效        |
| 68 | RMII_RX0    | I   | RMII 接收数据 0        |
| 69 | RMII_RX1    | I   | RMII 接收数据 1        |
| 70 | RMII_RX_ERR | I   | RMII 接收错误标志        |
| 71 | RMII_MDC    | O   | MDIO 时钟输出          |
| 72 | RMII_MDIO   | I/O | MDIO 数据信号          |
| 73 | VCC3.3      | P   | 3.3V               |
| 74 | GPIO1       | I/O | 通用 GPIO 端口, 编号: 1  |
| 75 | GPIO2       | I/O | 通用 GPIO 端口, 编号: 2  |
| 76 | VCC3.3      | P   | 3.3V               |
| 77 | VCC1.2      | P   | 1.2V               |
| 78 | GPIO3       | I/O | 通用 GPIO 端口, 编号: 3  |
| 79 | GPIO4       | I/O | 通用 GPIO 端口, 编号: 4  |
| 80 | GPIO5       | I/O | 通用 GPIO 端口, 编号: 5  |
| 81 | VCC3.3      | P   | 3.3V               |
| 82 | GPIO6       | I/O | 通用 GPIO 端口, 编号: 6  |
| 83 | VCC1.2      | P   | 1.2V               |

| 引脚 | 名称     | 类型  | 描述               |
|----|--------|-----|------------------|
| 84 | GPIO7  | I/O | 通用 GPIO 端口，编号：7  |
| 85 | GPIO8  | I/O | 通用 GPIO 端口，编号：8  |
| 86 | GPIO9  | I/O | 通用 GPIO 端口，编号：9  |
| 87 | GPIO10 | I/O | 通用 GPIO 端口，编号：10 |
| 88 | GPIO11 | I/O | 通用 GPIO 端口，编号：11 |

注：DCOUT1.1：芯片内部输出电压，VCC3.3：芯片供电电压，VCC1.2：芯片数字电压，VCC2.5A：UFUSE 供电，VCC1.2A：模拟电路 1.2V 电源，VCC3.3A：模拟电路 3.3V 电源。



## 5.3 典型应用图

### 5.3.1 典型应用--智能电表 HPLC & HRF 双模网络

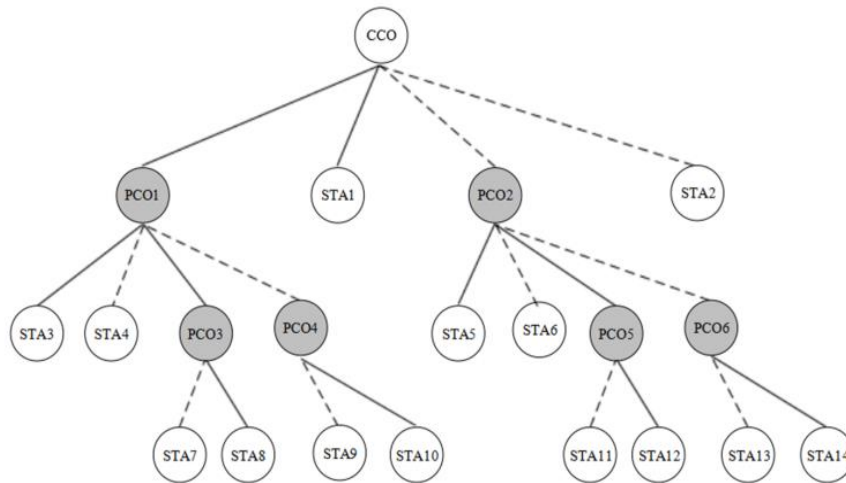


图 5-3 典型应用网络

图 1-3 所示，在 HPLC & HRF 双模网络中，CCO 为主节点载波通信单元(Central Coordinator, CCO)，模组主控芯片为 CN8514，STA 为子节点载波通信单元，主控芯片为 CN8513，PCO 为中继，在既定通信协议下完成数据通信，主要应用在智能电表，智能家居等行业。

### 5.3.2 典型应用--透传模组 485 网络

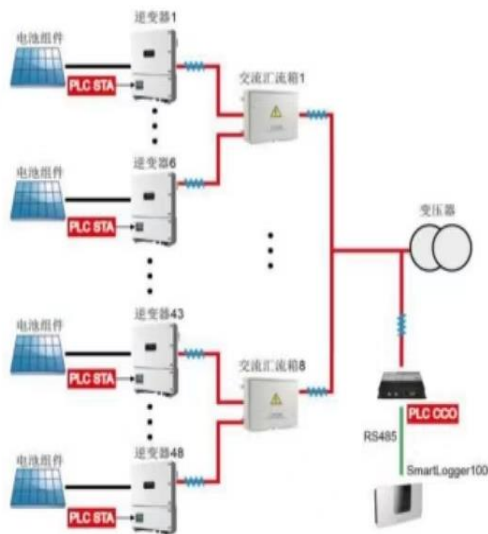


图 5-4 PLC 典型应用网络--RS485 透传通信

PLC CCO 具有 HPLC&HRF 双模通信通道。其工作原理如下：

通过高低压隔离耦合电路，隔离低频三相交流电与单相低压，保证单板安全，提供 PLC 信号输入和输出的高频通路。同时具有无线发射和接收通路。

通过接收链路，滤波、处理从交流电力线上解调的 PLC 信号。同时也可以通过无线信号进行数据的接收。

通过发送链路，对 PLC 信号进行功率放大，然后向交流电力线发射 PLC 信号。同时也可以通过无线信号进行数据的发送。

通过 PLC 控制器，实现 PLC 信号和 RS485 信号双向转换。

通过 RS485 收发接口电路，提供 RS485 信号接收和发送的通路，实现与数据采集器通信。

### 5.3.3 典型应用--数据采集物联网

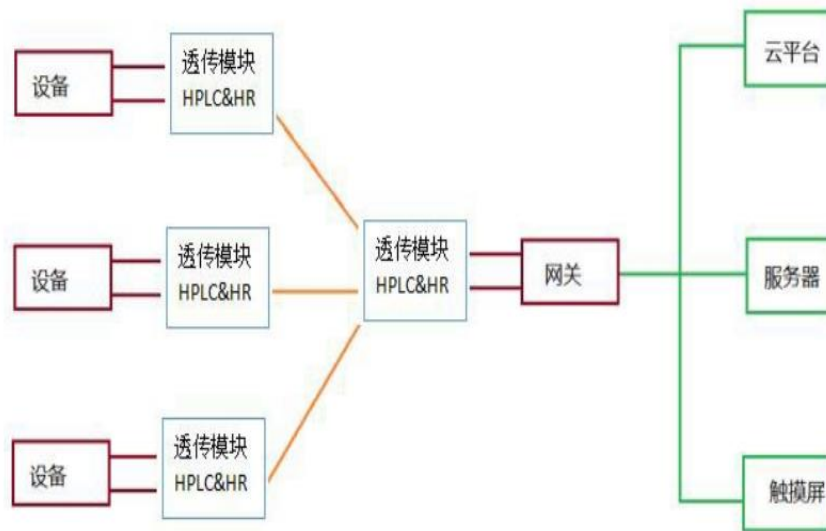


图 5-5 HPLC&HRF 典型应用网络--物联网

物联网（IOT）是近年比较热门的通信方式，基于互联网，实现智能化识别、定位、跟踪、监管等功能，引入 HPLC&HRF 双模模块，简化通信硬件，两条电力线即可完成通讯，无需额外布线即可入网通信，节省成本，并且在很多方面有广阔的应用前景。

## 6 电气参数

### 6.1 工作电流参数

| 名称      | 概述          | 静态     | 动态      | 单位 |
|---------|-------------|--------|---------|----|
| VCC3.3  | 外围数字电压 3.3V | 13     | 27      | mA |
| VCC3.3A | 模拟电压 3.3V   | 10     | 16      | mA |
| VCC1.2  | 内核数字电压 1.2V | 13     | 36      | mA |
| VCC1.2A | 模拟电压 1.2V   | 9      | 30      | mA |
| VCC1.8  | RF 部分电压     | 27（接收） | 148（发射） | mA |

注：静态：CPU 工作但物理层未接收，动态：CPU 工作而且物理层接收。

### 6.2 工作功率参数

| 名称    | 供电功耗   | 单位 |
|-------|--------|----|
| 芯片启动时 | 100.01 | mW |
| 物理层接收 | 201.3  | mW |

### 6.3 电源参数

| 名称             | 概述                    | 最小值 | 典型值 | 最大值 | 单位 |
|----------------|-----------------------|-----|-----|-----|----|
| VCC3.3、VCC3.3A | 外围数字电压 3.3V、模拟电压 3.3V | 3.1 | 3.3 | 3.5 | V  |
| VCC1.2、VCC1.2A | 内核数字电压 1.2V、模拟电压 1.2V | 1.1 | 1.2 | 1.3 | V  |
| VCC1.8         | RF 部分电压               | 1.8 | 1.8 | 1.9 | V  |
| VCC2.5A        | EFUSE 电源              | 2.3 | 2.5 | 2.7 | V  |

## 7 电源及复位

### 7.1 电源系统

芯片需要使用 3.3V 和 1.1V 两种直流电源供电。3.3V 电源分为模拟电源(VDDA)；引脚 IO(VDD33)。1.1V 电源 (VDD) 用于芯片内部数字电路供电，此 1.1V 电源可使用外部直接供给。

### 7.2 复位系统

芯片支持如下 6 种复位：

- 软件复位
- 外部复位
- 上电复位
- 欠压复位
- 看门狗复位
- 模拟复位

### 7.3 芯片上电时序

芯片的 3.3V、1.1V、RESET 信号上电过程时序如下图所示：

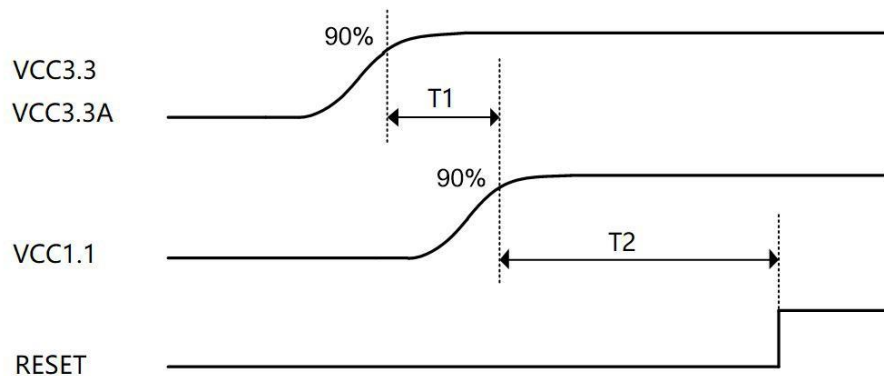


图 7-1 芯片上电启动时序

T1 和 T2 参数：

| 名称             | 概述                    | 最小值 | 最大值 | 单位 |
|----------------|-----------------------|-----|-----|----|
| VCC3.3、VCC3.3A | 外围数字电压 3.3V、模拟电压 3.3V | 3.1 | 3.5 | V  |
| VCC1.2、VCC1.2A | 内核数字电压 1.2V、模拟电压 1.2V | 1.1 | 1.3 | V  |
| VCC1.8         | RF 部分电压               | 1.8 | 1.9 | V  |
| VCC2.5A        | EFUSE 电源              | 2.3 | 2.7 | V  |

## 8 系统结构

芯片由数字和模拟两部分组成，系统结构图如下：

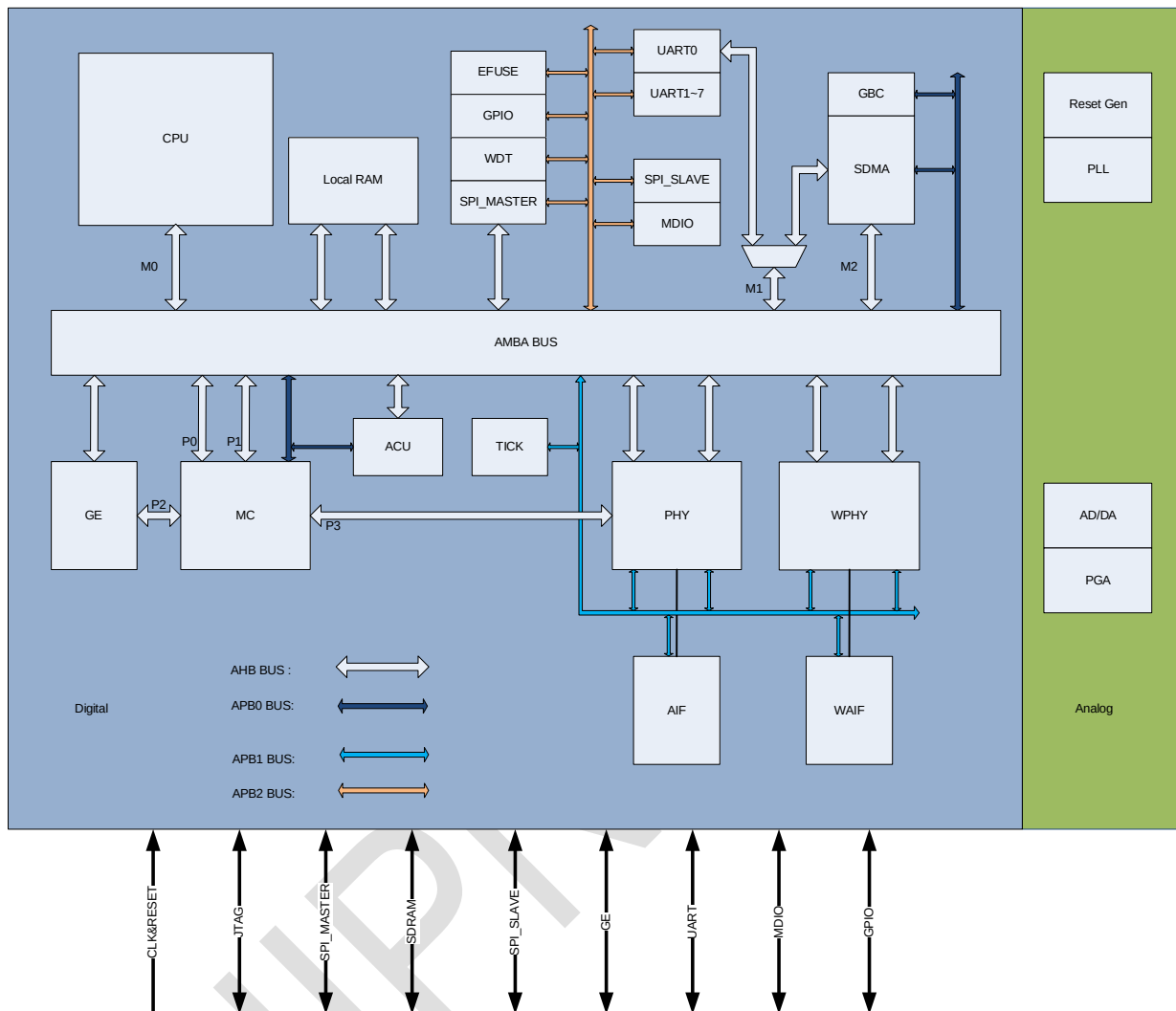


图 8-1 SOC 系统结构

## 8.1 数字功能模块

数字部分主要组成：

|    |                                               |
|----|-----------------------------------------------|
| 1  | CPU，主频最高 200MHz                               |
| 2  | 双口 SRAM，AHB 时钟，1.5M 内存                        |
| 3  | AHB 和 APB 总线，最高频率 100MHz                      |
| 4  | ACU，加密引擎                                      |
| 5  | SPI master 系统从此口读取 FLASH 并启动，启动后此端口用作通用 SPI 口 |
| 6  | RMII MAC(GE)，最高频率 50MHz                       |
| 7  | 内存 SDRAM 控制器 MC，最高频率 100MHz                   |
| 8  | 8 通道 DMA                                      |
| 9  | TICK 模块，提供 4 组 timer 功能                       |
| 10 | EFUSE 支持一次性写入唯一 CHIP ID                       |
| 11 | PLC 功能 MAC 以及 PHY                             |
| 12 | GPIO 功能                                       |
| 13 | 看门狗 WDT                                       |
| 14 | UART 接口，其中 UART0 支持调试功能                       |
| 15 | MIDO                                          |
| 16 | SPI SLAVE                                     |

## 8.2 模拟功能模块

模拟部分主要组成：

|   |                 |
|---|-----------------|
| 1 | RESET 控制电路      |
| 2 | PLL             |
| 3 | DCDC3.3V 和 1.2V |

一般的工作原理如下：传输有线信号作接收模式时，先通过低噪声放大器放大微弱的小信号，然后通过滤波器将频带外的干扰信号衰减抑制，最后通过模数转换器 ADC 将模拟信号转换成离散的数字信号给到数字基带做处理。传输有线信号作发射模式时，数字基带将调制好的 OFDM 信号给到数模转换器，通过滤波器、输出驱动放大器后去驱动有线接口。

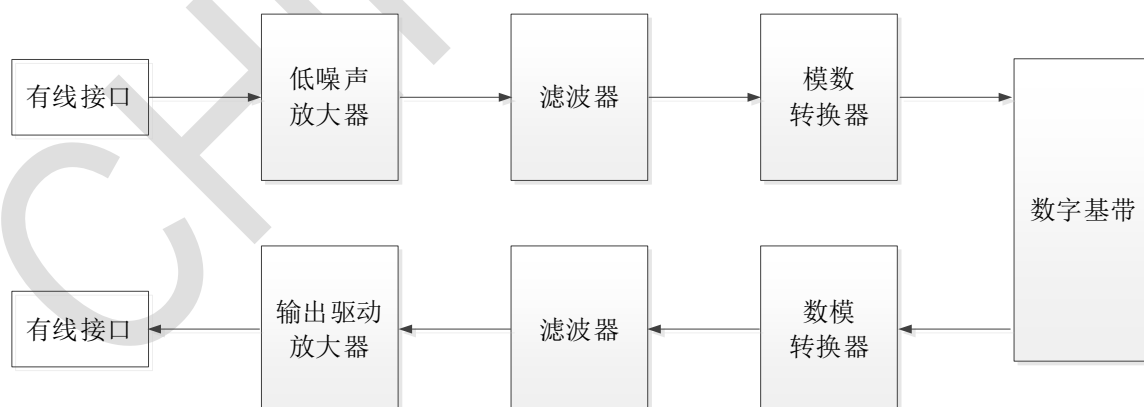


图 8-2 模拟功能原理图

### 8.3 中断控制系统

芯片控制系统如下图：

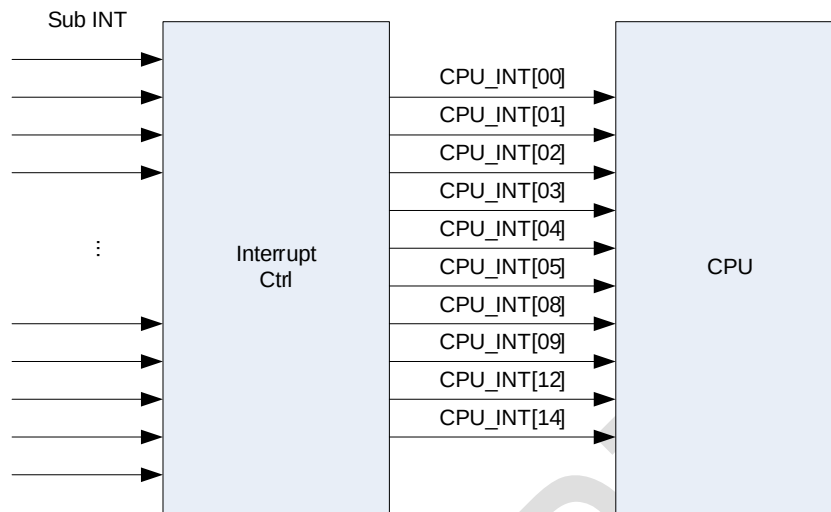


图 8-3 中断控制框架

中断控制模块集成在 GBC 模块中，SOC 系统中各子模块的中断，先送入中断控制器，处理后，再送入 CPU 处理器，各中断都拥有不同的优先级以及使能控制。

SOC 支持如下的模块的中断：

|   |                  |
|---|------------------|
| 1 | PHY 中断           |
| 2 | ACU 中断           |
| 3 | TICK timer 中断    |
| 4 | SDMA 中断          |
| 5 | GE, GEI, MDIO 中断 |
| 6 | GOIO 中断          |
| 7 | UART, SPI 中断     |

SOC 子模块发起中断，对应的中断信号拉起，中断控制器接收到子模块中断信号，并将子模块中断映射到对应的 CPU 中断上，CPU 接收到中断，并触发对应的中断处理函数，中断处理函数需要查询并清理子模块中断，然后清理 GBC 中的中断控制器的状态，以撤销 CPU 的中断。

#### 8.3.1 中断处理

芯片内部的中断处理流程如下：

- 子模块发起中断，拉起对应的中断信号
- 中断控制器接收到子模块中断信号，并将子模块中断映射到对应的 CPU 中断上
- CPU 接收到中断，并触发对应的中断处理函数

中断处理函数需要查询并清理子模块中断，然后清理 GBC 中的中断控制器的状态，以撤销 CPU 的中断。

## 8.4 芯片内部寻址空间

| 功能单元                    | 起始地址       | 结束地址         | 最大空间     |
|-------------------------|------------|--------------|----------|
| BOOT Address            | 0xFE000000 | 0xFEFFFFFF   | 16MByte  |
| Local RAM Port0         | 0x10000000 | 0x1FFFFFFF   | 256MByte |
| Local RAM Port1         | 0x20000000 | 0x2FFFFFFF   | 256MByte |
| Local RAM Port2         | 0x50000000 | 0x5FFFFFFF   | 256MByte |
| GE Memory and Registers | 0xD0000000 | 0xD0FFFFFF   | 16MByte  |
| ACU Memory              | 0xD1000000 | 0xD1FFFFFF   | 16MByte  |
| PMA Memory              | 0xD2000000 | 0xD2FFFFFF   | 16MByte  |
| PMD Memory              | 0xD3000000 | 0xD3FFFFFF   | 16MByte  |
| WPMA Memory             | 0xD4000000 | 0xD4FFFFFF   | 16MByte  |
| WPMD Memory             | 0xD5000000 | 0xD53FFFFFFF | 16MByte  |
| MC Port0                | 0x00000000 | 0x0FFFFFFF   | 256Mbyte |
| MC Port1                | 0xa0000000 | 0xaFFFFFFF   | 256Mbyte |
| MC Port2                | 0x80000000 | 0x8FFFFFFF   | 256Mbyte |
| MC Port3                | 0x90000000 | 0x9FFFFFFF   | 256Mbyte |
| GBC                     | 0xF0000000 | 0xF00FFFFF   | 1Mbyte   |
| SDMA                    | 0xF0100000 | 0xF01FFFFF   | 1Mbyte   |
| MC Registers            | 0xF0200000 | 0xF02FFFFF   | 1Mbyte   |
| ACU Registers           | 0xF0300000 | 0xF03FFFFF   | 1Mbyte   |
| MON                     | 0xF0400000 | 0xF04FFFFF   | 1Mbyte   |
| AIF                     | 0xF1000000 | 0xF10FFFFF   | 1MByte   |
| PMD                     | 0xF1100000 | 0xF11FFFFF   | 1MByte   |
| PMA                     | 0xF1200000 | 0xF12FFFFF   | 1MByte   |
| TICK                    | 0xF1300000 | 0xF13FFFFF   | 1MByte   |
| WAIF                    | 0xF1400000 | 0xF14FFFFF   | 1MByte   |
| WPMD                    | 0xF1500000 | 0xF15FFFFF   | 1MByte   |
| WPMA                    | 0xF1600000 | 0xF16FFFFF   | 1MByte   |
| SPI                     | 0xF0500000 | 0xF05FFFFF   | 1MByte   |
| SPI1                    | 0xF0600000 | 0xF06FFFFF   | 1MByte   |
| SPI2                    | 0xF0700000 | 0xF07FFFFF   | 1MByte   |
| SPI3                    | 0xF0800000 | 0xF08FFFFF   | 1MByte   |
| SPI4                    | 0xF0900000 | 0xF09FFFFF   | 1MByte   |
| SPI5                    | 0xF0A00000 | 0xF0AFFFFF   | 1MByte   |
| GPIO                    | 0xF2000000 | 0xF20FFFFF   | 1MByte   |
| SPI Slave               | 0xF2100000 | 0xF21FFFFF   | 1MByte   |
| SPI Slave1              | 0xF2200000 | 0xF22FFFFF   | 1MByte   |
| WDT                     | 0xF2300000 | 0xF23FFFFF   | 1MByte   |
| MDIO                    | 0xF2400000 | 0xF24FFFFF   | 1MByte   |
| UART0                   | 0xF2500000 | 0xF25FFFFF   | 1MByte   |
| UART1                   | 0xF2600000 | 0xF26FFFFF   | 1MByte   |
| UART2                   | 0xF2700000 | 0xF27FFFFF   | 1MByte   |
| UART3                   | 0xF2800000 | 0xF28FFFFF   | 1MByte   |

| 功能单元  | 起始地址       | 结束地址       | 最大空间   |
|-------|------------|------------|--------|
| UART4 | 0xF2900000 | 0xF29FFFFF | 1MByte |
| UART5 | 0xF2A00000 | 0xF2AFFFFF | 1MByte |
| UART6 | 0xF2B00000 | 0xF2BFFFFF | 1MByte |
| UART7 | 0xF2C00000 | 0xF2CFFFFF | 1MByte |
| EFUSE | 0xF2D00000 | 0xF2DFFFFF | 1MByte |
| I2C   | 0xF2E00000 | 0xF2EFFFFF | 1MByte |

## 9 时钟系统

芯片外部链接 25MHz 晶振，芯片内部 PLL 将其倍频以后供片内系统使用，软件根据具体的需要可以进行不同的分频配置，芯片内部功能模块时钟可单独关闭，以便降低功耗。

芯片时钟结构图如下：

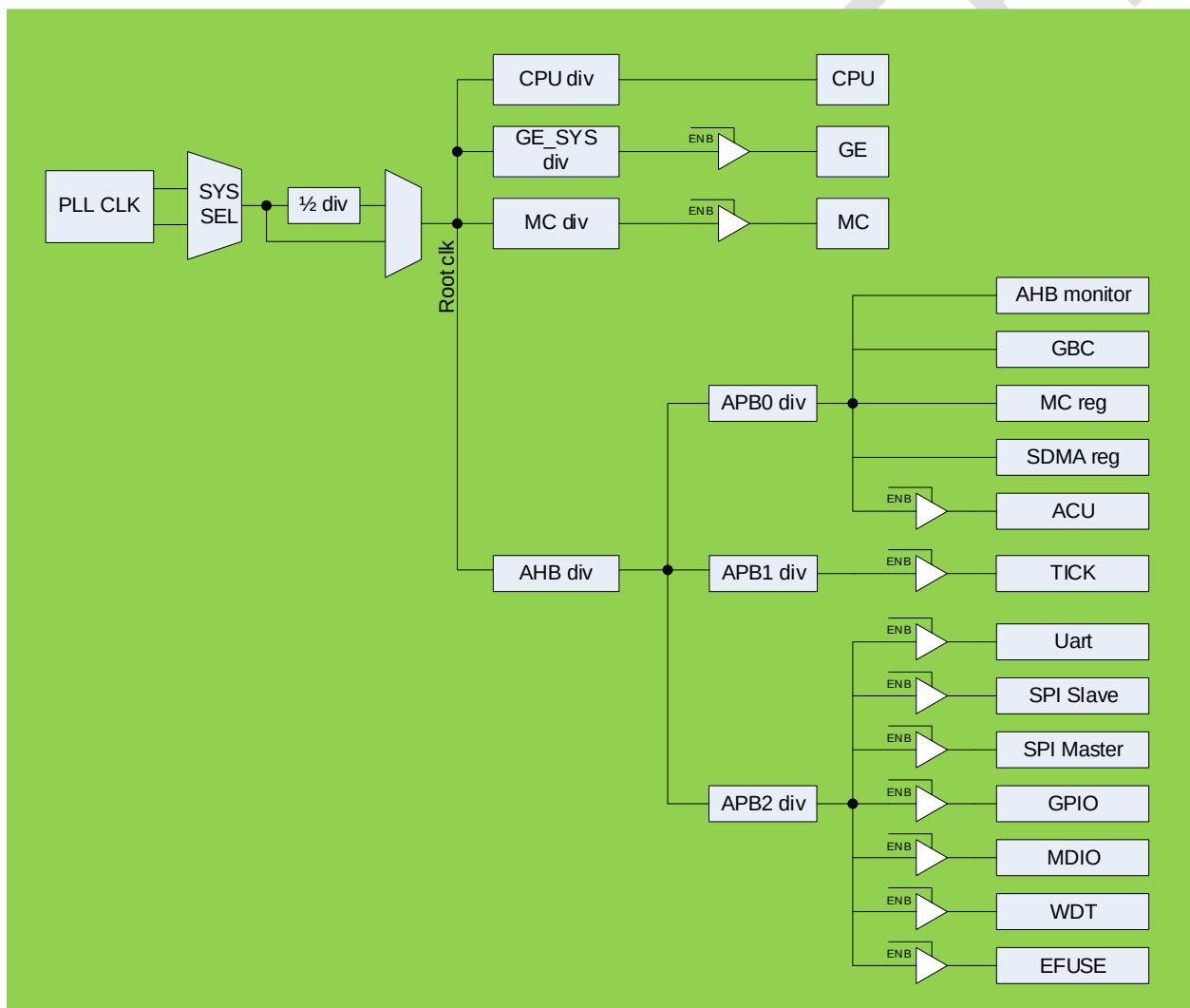


图 9-1 芯片时钟结构



## 10 GBC芯片全局控制

GBC 是此芯片的全局控制单元，主要由时钟控制、复位控制、DMA 映射、中断控制总线控制几个部分组成。

### 10.1 全局时钟控制

时钟控制寄存器地址：

| 地址/名称                     | 域                            | 初值  | 类型 | 说明                                                                                            |
|---------------------------|------------------------------|-----|----|-----------------------------------------------------------------------------------------------|
| 0xF0000000<br>SYS_CLK_CFG | [3]: SYS_PLL_SRC             | 0x0 | RW | 系统时钟选择 PLL 源<br>0: SYS_PLL<br>1: GE_PLL                                                       |
|                           | [2]: SYS_PLL_DIV             | 0x0 | RW | 对所选的系统 PLL 时钟二分频                                                                              |
|                           | [1]: SYS_CLK_SW              | 0x0 | RW | 切换系统时钟，此寄存器中所配置的系统时钟参数不是立即生效，而是在此域由 0 变为 1 的时候生效                                              |
|                           | [0]: SYS_CLK_PLL             | 0x0 | RW | 系统时钟选择<br>0: OSC (25MHz)<br>1: PLL                                                            |
| 0xF0000004<br>SYS_CLK_DIV | [31: 24]:<br>DEBUG_BAUD_RATE | 0x7 | RW | UART0 debug 模式下的波特率配置，默认波特率 115200，在 debug 模式下配置系统时钟分频的时候需要根据分频比来重新计算波特率。计算方式与 UART 波特率计算方式相同 |
|                           | [23]: GE_SYS_CLK_OPEN        | 0x1 | RW | GE 模块时钟使能                                                                                     |
|                           | [21]: MCLK_OPEN              | 0x1 | RW | MC 模块时钟使能                                                                                     |
|                           | [19: 16]: GE_SYS_CLK_DIV     | 0x0 | RW | GE 时钟分频                                                                                       |
|                           | [11: 8]: CPU_CLK_DIV         | 0x0 | RW | CPU 时钟分频                                                                                      |
|                           | [7: 4]: MCLK_DIV             | 0x0 | RW | MC 时钟分频                                                                                       |
|                           | [3: 0]: AHB_CLK_DIV          | 0x1 | RW | AHB 时钟分频                                                                                      |
| 0xF0000008<br>APB_CLK_DIV | [11: 8]: APB2_CLK_DIV        | 0x0 | RW | APB2 时钟分频                                                                                     |
|                           | [7: 4]: APB1_CLK_DIV         | 0x0 | RW | APB1 时钟分频                                                                                     |
|                           | [3: 0]: APB0_CLK_DIV         | 0x0 | RW | APB0 时钟分频                                                                                     |

| 地址/名称                  | 域               | 初值      | 类型 | 说明                                                                                                                                                                                                                                                                                                                                  |
|------------------------|-----------------|---------|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0xF000000C<br>CLK_GATE | [21: 0]: CLK_EN | 0x3ffff | RW | 1: enable; 0: disable<br>[21]: WPMA<br>[20]: WPMD<br>[19]: WAIF<br>[18]: UART4<br>[17]: UART3<br>[16]: SPI2<br>[15]: SPI1<br>[14]: I2C<br>[13]: EFUSE<br>[12]: TICK<br>[11]: ACU<br>[10]: UART2<br>[9]: UART1<br>[8]: UART0<br>[7]: SPI_SLAVE<br>[6]: MIDO<br>[5]: GPIO<br>[4]: WDT<br>[3]: SPI<br>[2]: PMA<br>[1]: PMD<br>[0]: AIF |

### 10.1.1 时钟配置说明

- MC、CPU、GE 时钟频率必须是 AHB 时钟的整数倍
- 典型应用下 APB0 时钟频率与 AHB 相同
- 为方便物理层控制，APB1 时钟频率默认为 50MHz
- 子模块的时钟可通过 CLK\_EN 寄存器单独关闭

## 10.2 全局复位控制

芯片支持软件对系统进行全局复位及对各个子模块独立复位操作，具体复位寄存器见下表：

| 地址         | 寄存器名称           |
|------------|-----------------|
| 0xF0000010 | SOFT_RST[31: 0] |
| 0xF0000014 | RST_STA[31: 0]  |

SOFT\_RST[31: 0]寄存器：

| 位                | 功能                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 类型 |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| [31]: SYS_RST_EN | 初始值：0x0，写 1 将对芯片进行全局复位                                                                                                                                                                                                                                                                                                                                                                                                                                                    | RW |
| [30: 0]: RST_EN  | 初始值：0x0，1：复位；0：撤销复位<br>[30]: WPMA<br>[29]: WPMD<br>[28]: WAIF<br>[27]: UART4<br>[26]: UART3<br>[25]: SPI2<br>[24]: SPI1<br>[23]: I2C<br>[22]: efuse<br>[21]: tick<br>[20]: ge_cg<br>[19]: ge_mii_rx<br>[18]: ge_mii_tx<br>[17]: mon<br>[16]: acu<br>[15]: uart2<br>[14]: uart1<br>[13]: uart0<br>[12]: gpio<br>[11]: spi_slave<br>[10]: wdt<br>[9]: mdio<br>[8]: spi<br>[7]: ge<br>[6]: pma_rx<br>[5]: pma_tx<br>[4]: pma<br>[3]: pmd<br>[2]: aif<br>[1]: mc<br>[0]: sdma | RW |

RST\_STA[31: 0]寄存器:

| 位                     | 功能                                    | 类型 |
|-----------------------|---------------------------------------|----|
| [15]: SFT_GLB_RST_GEN | 初始值: 0x0, 软件复位记录                      | RO |
| [14]: DT_GLB_RST_GEN  | 初始值: 0x0, 看门狗全局复位记录                   | RO |
| [13]: EXT_WDT_RST_GEN | 初始值: 0x0, 看门狗外部复位记录                   | RO |
| [12]: EXT_RST_GEN     | 初始值: 0x1, 芯片外部复位记录(RESET_IN 管脚)       | RO |
| [11]: ANA_PAD_RST_GEN | 初始值: 0x1, 模拟复位记录(RESET_MR 管脚)         | RO |
| [10]: NA_WDT_RST_GEN  | 初始值: 0x0, 看门狗模拟复位记录                   | RW |
| [9]: ANA_LP_RST_GEN   | 初始值: 0x0, 欠压复位记录                      | RO |
| [8]: ANA_POR_RST_GEN  | 初始值: 0x1, 上电复位记录                      | RO |
| [0]: RST_GEN_STA_CLR  | 初始值: 0x0, 清除复位记录, 完成后需要软件写 0 以便恢复复位记录 | RW |

注: 上电的时候有一定的概率检测到欠压复位, 如果遇到上电复位和欠压复位同时产生的情况, 可以直接屏蔽欠压复位状态, 认为上一次复位是上电复位。

SYS\_RST\_EN 全局复位将对芯片所有的模块进行复位, 如非必要尽量不要使用此功能。RST\_EN 可对各个子模块进行单独复位, 软件需要在复位完成后在将对应模块的比特位配置为 0, 以撤销复位。

RST\_STA 寄存器主要用于记录芯片的异常复位情况, 软件可以在上电后读取该寄存器的值, 以便确定上一次复位的原因, 根据不同的复位源进行相应的操作。一般在读取完此寄存器后, 需要对 RST\_GEN\_STA\_CLR 进行写 1 和写 0 操作, 以清除复位记录。

### 10.3 DMA 硬件连接映射

SDMA 模块支持两种硬件相关的 DMA 调度方式, 一种是由子模块自己发起 DMA 启动、停止请求, 软件只要进行简单的配置即可; 另外一种是由软件发起 DMA 连接, 子模块给 SDMA 模块提供是否有数据(流控)信息, 以便 SDMA 发起一次总线 burst 操作。第二种 DMA 方式能够最有效的利用总线带宽, 且软件的参与度也比较高, 控制起来比较容易。两种方式均需要通过寄存器配置, 以便将不同的子模块和不同的 SDMA 通道连接。DMA 硬件连接相关的寄存器如下表所示:

| 地址         | 寄存器名称           |
|------------|-----------------|
| 0xF0000020 | DRQ_EN[31: 0]   |
| 0xF0000024 | DFC_EN[31: 0]   |
| 0xF0000028 | DFC_EN_1[31: 0] |

DRQ\_EN[31: 0]寄存器:

| 位                       | 功能                       | 类型 |
|-------------------------|--------------------------|----|
| [15: 12]: DRQ_MDIO_MGDI | 初始值 0x8, MDIO 模块 DMA 通道  | RW |
| [11: 8]: DRQ_UART2_RX   | 初始值 0x8, UART2 模块 DMA 通道 | RW |
| [7: 4]: DRQ_UART1_RX    | 初始值 0x8, UART1 模块 DMA 通道 | RW |
| [3: 0]: DRQ_UART0_RX    | 初始值 0x8, UART0 模块 DMA 通道 | RW |

**DFC\_EN[31: 0]寄存器:**

| 位                          | 功能                           | 类型 |
|----------------------------|------------------------------|----|
| [23: 20]: DFC_GE_GEI_RX    | 初始值 0x8, GE 发送流控 DMA 通道      | RW |
| [23: 20]: DFC_GE_GEI_RX    | 初始值 0x8, GE 接收流控 DMA 通道      | RW |
| [19: 16]: DFC_ACU_CRC_RX   | 初始值 0x8, ACU CRC 写入流控 DMA 通道 | RW |
| [15: 12]: DFC_ACU_CRYPT_RX | 初始值 0x8, ACU 加解密写入流控 DMA 通道  | RW |
| [11: 8]: DFC_ACU_CRYPT_TX  | 初始值 0x8, ACU 加解密读取流控 DMA 通道  | RW |
| [7: 4]: DFC_PMA_PMI_RX     | 初始值 0x8, PMI 接收流控 DMA 通道     | RW |
| [3: 0]: DFC_PMA_PMI_TX     | 初始值 0x8, PMI 发送流控 DMA 通道     | RW |

**DFC\_EN\_1[31: 0]寄存器:**

| 位                        | 功能                             | 类型 |
|--------------------------|--------------------------------|----|
| [19: 16]: DFC_WPMD_TX    | 初始值 0x8, WPMD 发送流控 DMA 通道      | RW |
| [15: 12]: DFC_WPMA_RX    | 初始值 0x8, WPMA 接收流控 DMA 通道      | RW |
| [11: 8]: DFC_WPMA_TX     | 初始值 0x8, WPMA 发送流控 DMA 通道      | RW |
| [7: 4]: DFC_SPI_SLAVE_RX | 初始值 0x8, SPI_SLAVE 接收流控 DMA 通道 | RW |
| [3: 0]: DFC_SPI_SLAVE_TX | 初始值 0x8, SPI_SLAVE 发送流控 DMA 通道 | RW |

注：有效的 DMA 通道编号为 0 到 7；其他值均无效。

DRQ\_EN 用于上文提到的第一种硬件 DMA 连接。

DFC\_EN\DFC\_EN\_1 用于上文提到的第二种硬件 DMA 连接。

## 10.4 全局中断控制

中断控制寄存器如下表所示：

| 地址         | 寄存器名称             |
|------------|-------------------|
| 0xF0000030 | CINTR_EN[31: 0]   |
| 0xF0000034 | CINTR_STA[31: 0]  |
| 0xF0000040 | MINTR_EN[31: 0]   |
| 0xF0000044 | MINTR_STA[31: 0]  |
| 0xF0000048 | MINTR_CLR[31: 0]  |
| 0xF0000050 | MINRT_PRI0[31: 0] |
| 0xF0000054 | MINRT_PRI1[31: 0] |
| 0xF0000058 | MINRT_PRI2[31: 0] |
| 0xF000005C | MINRT_PRI3[31: 0] |

CINTR\_EN[31: 0]寄存器：

| 位           | 功能                                                                                                                                                                                   | 类型 |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| [9: 0]: CIE | 初始值 0x0，CPU 中断使能标识<br>[9]: 中断 14 使能<br>[8]: 中断 12 使能<br>[7]: 中断 9 使能<br>[6]: 中断 8 使能<br>[5]: 中断 5 使能<br>[4]: 中断 4 使能<br>[3]: 中断 3 使能<br>[2]: 中断 2 使能<br>[1]: 中断 1 使能<br>[0]: 中断 0 使能 | RW |

CINTR\_STA[31: 0]寄存器：

| 位           | 功能                             | 类型 |
|-------------|--------------------------------|----|
| [9: 0]: CIS | 初始值 0x0，CPU 中断状态标识，对应于 CIE 寄存器 | RO |

## MINTR\_EN[31: 0]寄存器:

| 位                   | 功能                         | 类型 |
|---------------------|----------------------------|----|
| [30]: MIE_UART4     | 初始值 0x0, UART4 全局中断使能      | RW |
| [29]: MIE_UART3     | 初始值 0x0, UART3 全局中断使能      | RW |
| [28]: MIE_SPI2      | 初始值 0x0, SPI2 全局中断使能       | RW |
| [27]: MIE_SPI1      | 初始值 0x0, SPI1 全局中断使能       | RW |
| [26]: MIE_I2C       | 初始值 0x0, I2C 全局中断使能        | RW |
| [25]: MIE_HTMR      | 初始值 0x0, TICK TIMER 全局中断使能 | RW |
| [24]: MIE_GE_GEI    | 初始值 0x0, GEI 全局中断使能        | RW |
| [23]: MIE_PMA_PMI   | 初始值 0x0, PMI 全局中断使能        | RW |
| [22]: MIE_ACU       | 初始值 0x0, ACU 全局中断使能        | RW |
| [21]: MIE_UART2     | 初始值 0x0, UART2 全局中断使能      | RW |
| [20]: MIE_UART1     | 初始值 0x0, UART1 全局中断使能      | RW |
| [19]: MIE_MDIO_MGDI | 初始值 0x0, MDIO 全局中断使能       | RW |
| [18]: MIE_GE        | 初始值 0x0, GE 全局中断使能         | RW |
| [17]: MIE_SPI_SLAVE | 初始值 0x0, SPI_SLAVE 全局中断使能  | RW |
| [16]: MIE_SDMA_CH7  | 初始值 0x0, SDMA 通道 7 全局中断使能  | RW |
| [16]: MIE_SDMA_CH7  | 初始值 0x0, SDMA 通道 7 全局中断使能  | RW |
| [15]: MIE_SDMA_CH6  | 初始值 0x0, SDMA 通道 6 全局中断使能  | RW |
| [14]: MIE_SDMA_CH5  | 初始值 0x0, SDMA 通道 5 全局中断使能  | RW |
| [13]: MIE_SDMA_CH4  | 初始值 0x0, SDMA 通道 4 全局中断使能  | RW |
| [12]: MIE_SDMA_CH3  | 初始值 0x0, SDMA 通道 3 全局中断使能  | RW |
| [11]: MIE_SDMA_CH2  | 初始值 0x0, SDMA 通道 2 全局中断使能  | RW |
| [10]: MIE_SDMA_CH1  | 初始值 0x0, SDMA 通道 1 全局中断使能  | RW |
| [9]: MIE_SDMA_CH0   | 初始值 0x0, SDMA 通道 0 全局中断使能  | RW |
| [8]: MIE_UART0      | 初始值 0x0, UART0 全局中断使能      | RW |
| [7]: MIE_GPIO_EXT   | 初始值 0x0, GPIO 外部中断全局使能     | RW |
| [6]: MIE_SPI        | 初始值 0x0, SPI_MASTER 全局中断使能 | RW |
| [5]: MIE_MC         | 初始值 0x0, MC 全局中断使能         | RW |
| [4]: MIE_PMA_ERR    | 初始值 0x0, PMA 错误全局中断使能      | RW |
| [3]: MIE_PMA_HDR    | 初始值 0x0, PMA Header 全局中断使能 | RW |
| [2]: MIE_PMD_ERR    | 初始值 0x0, PMD 错误全局中断使能      | RW |
| [1]: MIE_TICK       | 初始值 0x0, TICK 全局中断使能       | RW |
| [0]: MIE_PMD_PHY    | 初始值 0x0, PMD 物理层全局中断使能     | RW |

MINTR\_STA[31: 0]寄存器:

| 位                   | 功能                         | 类型 |
|---------------------|----------------------------|----|
| [30]: MIS_UART4     | 初始值 0x0, UART4 全局中断状态      | RO |
| [29]: MIS_UART3     | 初始值 0x0, UART3 全局中断状态      | RO |
| [28]: MIS_SPI2      | 初始值 0x0SPI2 全局中断状态         | RO |
| [27]: MIS_SPI1      | 初始值 0x0, SPI1 全局中断状态       | RO |
| [26]: MIS_I2C       | 初始值 0x0, I2C 全局中断状态        | RO |
| [25]: MIS_HTMR      | 初始值 0x0, TICK TIMER 全局中断状态 | RO |
| [24]: MIS_GE_GEI    | 初始值 0x0, GEI 全局中断状态        | RO |
| [23]: MIS_PMA_PMI   | 初始值 0x0, PMI 全局中断状态        | RO |
| [22]: MIS_ACU       | 初始值 0x0, ACU 全局中断状态        | RO |
| [21]: MIS_UART2     | 初始值 0x0, UART2 全局中断状态      | RO |
| [20]: MIS_UART1     | 初始值 0x0, UART1 全局中断状态      | RO |
| [19]: MIS_MDIO_MGDI | 初始值 0x0, MDIO 全局中断状态       | RO |
| [18]: MIS_GE        | 初始值 0x0, GE 全局中断状态         | RO |
| [17]: MIS_SPI_SLAVE | 初始值 0x0, SPI_SLAVE 全局中断状态  | RO |
| [16]: MIS_SDMA_CH7  | 初始值 0x0, SDMA 通道 7 全局中断状态  | RO |
| [15]: MIS_SDMA_CH6  | 初始值 0x0, SDMA 通道 6 全局中断状态  | RO |
| [14]: MIS_SDMA_CH5  | 初始值 0x0, SDMA 通道 5 全局中断状态  | RO |
| [13]: MIS_SDMA_CH4  | 初始值 0x0, SDMA 通道 4 全局中断状态  | RO |
| [12]: MIS_SDMA_CH3  | 初始值 0x0, SDMA 通道 3 全局中断状态  | RO |
| [11]: MIS_SDMA_CH2  | 初始值 0x0, SDMA 通道 2 全局中断状态  | RO |
| [10]: MIS_SDMA_CH1  | 初始值 0x0, SDMA 通道 1 全局中断状态  | RO |
| [9]: MIS_SDMA_CH0   | 初始值 0x0, SDMA 通道 0 全局中断状态  | RO |
| [8]: MIS_UART0      | 初始值 0x0, UART0 全局中断状态      | RO |
| [7]: MIS_GPIO_EXT   | 初始值 0x0, GPIO 外部中断全局状态     | RO |
| [6]: MIS_SPI        | 初始值 0x0, SPI_MASTER 全局中断状态 | RO |
| [5]: MIS_MC         | 初始值 0x0, MC 全局中断状态         | RO |
| [4]: MIS_PMA_ERR    | 初始值 0x0, PMA 错误全局中断状态      | RO |
| [3]: MIS_PMA_HDR    | 初始值 0x0, PMA Header 全局中断状态 | RO |
| [2]: MIS_PMD_ERR    | 初始值 0x0, PMD 错误全局中断状态      | RO |
| [1]: MIS_TICK       | 初始值 0x0, TICK 全局中断状态       | RO |
| [0]: MIS_PMD_PHY    | 初始值 0x0, PMD 物理层全局中断状态     | RO |



MINTR\_CLR[31: 0]寄存器:

| 位                   | 功能                      | 类型 |
|---------------------|-------------------------|----|
| [30]: MIC_UART4     | 初始值 0x0，写 1 清理对应的全局中断状态 | A0 |
| [29]: MIC_UART3     |                         |    |
| [28]: MIC_SPI2      |                         |    |
| [27]: MIC_SPI1      |                         |    |
| [26]: MIC_I2C       |                         |    |
| [25]: MIC_HTMR      |                         |    |
| [24]: MIC_GE_GEI    |                         |    |
| [23]: MIC_PMA_PMI   |                         |    |
| [22]: MIC_ACU       |                         |    |
| [21]: MIC_UART2     |                         |    |
| [20]: MIC_UART1     |                         |    |
| [19]: MIC_MDIO_MGDI |                         |    |
| [18]: MIC_GE        |                         |    |
| [17]: MIC_SPI_SLAVE |                         |    |
| [16]: MIC_SDMA_CH7  |                         |    |
| [15]: MIC_SDMA_CH6  |                         |    |
| [14]: MIC_SDMA_CH5  |                         |    |
| [13]: MIC_SDMA_CH4  |                         |    |
| [12]: MIC_SDMA_CH3  |                         |    |
| [11]: MIC_SDMA_CH2  |                         |    |
| [10]: MIC_SDMA_CH1  |                         |    |
| [9]: MIC_SDMA_CH0   |                         |    |
| [8]: MIC_UART0      |                         |    |
| [7]: MIC_GPIO_EXT   |                         |    |
| [6]: MIC_SPI        |                         |    |
| [5]: MIC_MC         |                         |    |
| [4]: MIC_PMA_ERR    |                         |    |
| [3]: MIC_PMA_HDR    |                         |    |
| [2]: MIC_PMD_ERR    |                         |    |
| [1]: MIC_TICK       |                         |    |
| [0]: MIC_PMD_PHY    |                         |    |

MINRT\_PRI0[31: 0]寄存器：

| 位                      | 功能                           | 类型 |
|------------------------|------------------------------|----|
| [31: 28]: MIP_GPIO_EXT | 初始值 0x0，各个子模块中断到 CPU 中断的映射配置 | RW |
| [27: 24]: MIP_SPI      |                              |    |
| [23: 20]: MIP_MC       |                              |    |
| [19: 16]: MIP_PMA_ERR  |                              |    |
| [15: 12]: MIP_PMA_HDR  |                              |    |
| [11: 8]: MIP_PMD_ERR   |                              |    |
| [7: 4]: MIP_TICK       |                              |    |
| [3: 0]: MIP_PMD_PHY    |                              |    |

MINRT\_PRI1[31: 0]寄存器：

| 位                      | 功能                           | 类型 |
|------------------------|------------------------------|----|
| [31: 28]: MIP_SDMA_CH6 | 初始值 0x0，各个子模块中断到 CPU 中断的映射配置 | RW |
| [27: 24]: MIP_SDMA_CH5 |                              |    |
| [23: 20]: MIP_SDMA_CH4 |                              |    |
| [19: 16]: MIP_SDMA_CH3 |                              |    |
| [15: 12]: MIP_SDMA_CH2 |                              |    |
| [11: 8]: MIP_SDMA_CH1  |                              |    |
| [7: 4]: MIP_SDMA_CH0   |                              |    |
| [3: 0]: MIP_UART0      |                              |    |

MINRT\_PRI2[31: 0]寄存器：

| 位                       | 功能                           | 类型 |
|-------------------------|------------------------------|----|
| [31: 28]: MIP_PMA_PMI   | 初始值 0x0，各个子模块中断到 CPU 中断的映射配置 | RW |
| [27: 24]: MIP_ACU       |                              |    |
| [23: 20]: MIP_UART2     |                              |    |
| [19: 16]: MIP_UART1     |                              |    |
| [15: 12]: MIP_MDIO_MGDI |                              |    |
| [11: 8]: MIP_GE         |                              |    |
| [7: 4]: MIP_SPI_SLAVE   |                              |    |
| [3: 0]: MIP_SDMA_CH7    |                              |    |

MINRT\_PRI3[31: 0]寄存器：

| 位                   | 功能                           | 类型 |
|---------------------|------------------------------|----|
| [27: 24]: MIP_UART4 | 初始值 0x0，各个子模块中断到 CPU 中断的映射配置 | RW |
| [23: 20]: MIP_UART3 |                              |    |
| [19: 16]: MIP_SPI2  |                              |    |
| [15: 12]: MIP_SPI1  |                              |    |
| [11: 8]: MIP_I2C    |                              |    |
| [7: 4]: MIP_HTMIR   |                              |    |
| [3: 0]: MIP_GE_GEI  |                              |    |

不同 CPU 中断号对应于不同的中断优先级。具体关系如下：

| CPU 中断编号 | 中断优先级 | 等级    |
|----------|-------|-------|
| 14       | NMI   | 最高优先级 |
| 12       | 4     | 中等优先级 |
| 9        | 3     | 中等优先级 |
| 8        | 2     | 中等优先级 |
| 7        | 1     | 最低优先级 |
| 6        | 1     | 最低优先级 |
| 5        | 1     | 最低优先级 |
| 4        | 1     | 最低优先级 |
| 3        | 1     | 最低优先级 |
| 2        | 1     | 最低优先级 |
| 1        | 1     | 最低优先级 |
| 0        | 1     | 最低优先级 |

## 10.5 总线控制

芯片内部支持对 AHB 总线的错误进行控制，以防止总线挂死。支持 Local RAM 的两个端口进行优先级控制。控制寄存器如下：

| 地址         | 寄存器名称                |
|------------|----------------------|
| 0xF0000060 | AHB_ERR_EN[31: 0]    |
| 0xF0000064 | AHB_TIMEOUT[31: 0]   |
| 0xF0000068 | LRAM_BANK_PRI[31: 0] |

AHB\_ERR\_EN[31: 0]寄存器：

| 位               | 功能                   | 类型 |
|-----------------|----------------------|----|
| [0]: AHB_ERR_EN | 初始值 0x0，AHB 总线错误检测使能 | RW |

AHB\_TIMEOUT[31: 0]寄存器：

| 位                    | 功能                                  | 类型 |
|----------------------|-------------------------------------|----|
| [31: 0]: AHB_TIMEOUT | 初始值 0x10000000，AHB 总线超时周期，单位 AHB 时钟 | RW |

LRAM\_BANK\_PRI[31: 0]寄存器：

| 位                     | 功能                                                                       | 类型 |
|-----------------------|--------------------------------------------------------------------------|----|
| [1: 0]: LRAM_BANK_PRI | 初始值 0x0，<br>[1]: 1: BANK1 端口 1 优先级高于端口 0<br>[0]: 1: BANK0 端口 1 优先级高于端口 0 | RW |

## 11 存储控制

芯片存储主要由两部分组成：片内存储和片外存储，其中片内存储对应高速小容量的特点主要满足运行期间的需要，本设计提供 1.5MB 的高速片内存储即 LRAM，片外存储相对是低速大容量的特点，本设计提供两种方式：带有 MC 控制器的 SDRAM 和高速低管脚数量的 PSRAM。

### 11.1 MC 内存控制器

MC 是一个通用的 SDRAM 内存控制器，MC 提供 4 组 AHB slave 接口供不同的设备进行访问，4 组 AHB slave 端口优先级相同，采用 round robin 方式进行访问。4 组 AHB slave 接口其中两组用于 AHB 总线连接，还有两组分别用于 GE 和 PLC PHY 的 PDMA 连接，具体的连接关系请参考 SOC 系统结构图。

MC 基础特征：

|   |                                     |
|---|-------------------------------------|
| 1 | 最高工作于 200MHz                        |
| 2 | 支持 1MB 到 16MB SDRAM                 |
| 3 | 支持 SDRAM 低功耗功能                      |
| 4 | 支持调整 SDRAM 刷新周期，最小 0.976us 最大 125us |
| 5 | 支持 2Bank 或 4Bank SDRAM              |
| 6 | CAS 支持 1 到 3                        |
| 7 | burst length 支持 1、2、4、8 和 Full page |

由于内封 SDRAM，不同的 SDRAM 可能需要不同的配置，所以请务必使用芯片 SDK 提供的 MC 配置，用户不能随意修改 MC 的配置。

### 11.2 LRAM 片内存储

LRAM 片内存储，直接连接到 AMBA 总线上，特性如下：

|   |                                           |
|---|-------------------------------------------|
| 1 | 由 32 个 32KB 大小的 BANK 组成                   |
| 2 | 提供两个通用的访问接口                               |
| 3 | 支持同时访问两个 BANK                             |
| 4 | 支持 BANK 对访问端口的优先级控制，控制寄存器请参考表格 19：总线控制寄存器 |

## 12 SDMA系统DMA控制器

SDMA 模块主要用于在芯片内部总线上进行高效的数据搬移，以节约大量的 CPU 处理时间。SDMA 支持在 SDRAM 到 SDRAM，SDRAM 到 LRAM，GE 到 RAM，ACU 到 RAM，PHY 到 RAM 及部分低速 APB 总线设备之间搬移数据。

SDMA 支持如下特性：

|    |                                    |
|----|------------------------------------|
| 1  | 双 AHB master 接口，接口连接方式请参考 SOC 系统结构 |
| 2  | 每个通道可独立配置使用几个 AHB master 接口        |
| 3  | 8 条独立通道，可并行工作                      |
| 4  | 每个通道包含 32 个 word 缓冲区，支持流水线配置       |
| 5  | 支持自动切分数据总长到多个总线 burst              |
| 6  | 支持 byte、half_word、word 传输          |
| 7  | 支持通道优先级配置                          |
| 8  | 支持地址递增、锁定、跳变、环回模式                  |
| 9  | 支持硬件握手模式                           |
| 10 | 支持通道级联功能                           |
| 11 | 支持自动重载通道配置                         |
| 12 | 支持硬件数据流控模式（需要对应模块的支持），能有效提高总线利用效率  |
| 13 | 支持 DMA 通道错误检测                      |

### 12.1 SDMA 通道传输控制

SDMA 通道典型传输框图如下所示：

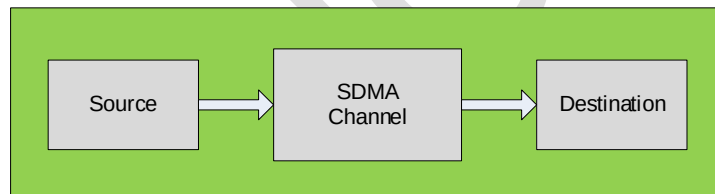


图 12-1 SDMA 通道传输示意图

各通道 DMA 功能寄存器如下表所示：

| 地址         | 寄存器名称       |
|------------|-------------|
| 0xF0100X00 | SA[31: 0]   |
| 0xF0100X04 | DA[31: 0]   |
| 0xF0100X08 | SOA[31: 0]  |
| 0xF0100X0C | DOA[31: 0]  |
| 0xF0100X20 | LEN[31: 0]  |
| 0xF0100X24 | CTRL[31: 0] |
| 0xF0100X28 | STA[31: 0]  |
| 0xF0100X40 | IE[31: 0]   |
| 0xF0100X44 | IS[31: 0]   |
| 0xF0100X64 | TO[31: 0]   |
| 0xF0100X68 | BUF[31: 0]  |

SA[31: 0]寄存器:

| 位           | 功能                                | 类型 |
|-------------|-----------------------------------|----|
| [31: 0]: SA | 初始值: 0x0, 源数据地址, 读取值为当前正在搬移的源数据地址 | RW |

DA[31: 0]寄存器:

| 位           | 功能                                  | 类型 |
|-------------|-------------------------------------|----|
| [31: 0]: DA | 初始值: 0x0, 目的数据地址, 读取值为当前正在搬移的目的数据地址 | RW |

SOA[31: 0]寄存器:

| 位            | 功能                                                                                | 类型 |
|--------------|-----------------------------------------------------------------------------------|----|
| [31: 0]: SOA | 初始值: 0x0, 特定模式源地址附加配置:<br>跳变模式下, 高 16 位表示块大小, 低 16 位表示偏移量<br>环回模式下, 低 16 位表示环回块大小 | RW |

DOA[31: 0]寄存器:

| 位            | 功能                                                                                 | 类型 |
|--------------|------------------------------------------------------------------------------------|----|
| [31: 0]: DOA | 初始值: 0x0, 特定模式目的地址附加配置:<br>跳变模式下, 高 16 位表示块大小, 低 16 位表示偏移量<br>环回模式下, 低 16 位表示环回块大小 | RW |

LEN[31: 0]寄存器:

| 位            | 功能                                 | 类型 |
|--------------|------------------------------------|----|
| [15: 0]: LEN | 初始值: 0x0, DMA 传输数据长度, 读取值为本次传输剩余长度 | RW |

CTRL[31: 0]寄存器：

| 位                 | 功能                                                                                                                 | 类型 |
|-------------------|--------------------------------------------------------------------------------------------------------------------|----|
| [29]: DFC_EN      | 初始值：0x0，目的端硬件流控使能                                                                                                  | RW |
| [28]: SFC_EN      | 初始值：0x0，源端硬件流控使能                                                                                                   | RW |
| [26]: RELOAD_LEN  | 初始值：0x0，完成后自动重载 LEN 寄存器                                                                                            | RW |
| [25]: RELOAD_DA   | 初始值：0x0，完成后自动重载 DA 寄存器                                                                                             | RW |
| [24]: RELOAD_SA   | 初始值：0x0，完成后自动重载 SA 寄存器                                                                                             | RW |
| [23: 20]: SW_HS   | 初始值：0x0，软件握手模式 DMA 指令：<br>4'b0000：无效<br>4'b0001：通道启动<br>4'b0010：通道暂停<br>4'b0100：强制停止改通道                            | A0 |
| [19]: PAUSER      | 初始值：0x0，可暂停通道端口选择<br>0：源端<br>1：目的端                                                                                 | RW |
| [18]: STOPPER     | 初始值：0x0，可停止通道端口选择<br>0：源端<br>1：目的端                                                                                 | RW |
| [17: 16]: HS_MODE | 初始值：0x0，握手模式选择<br>0：软件模式<br>1：硬件模式<br>2：软硬件混合模式                                                                    | RW |
| [15: 12]: DA_MODE | 初始值：0x0，目的地址产生模式<br>0：递增<br>1：固定<br>2：跳变<br>3：环回                                                                   | RW |
| [11: 8]: SA_MODE  | 初始值：0x0，源地址产生模式<br>0：递增<br>1：固定<br>2：跳变<br>3：环回                                                                    | RW |
| [7]: CHAN_EN      | 初始值：0x0，通道级联使能                                                                                                     | RW |
| [6: 4]: CHAN_TO   | 初始值：0x0，上一级通道号                                                                                                     | RW |
| [3: 2]: PT_CFG    | 初始值：0x0，源端和目的端端口配置<br>0：AHB0->源端；AHB1->目的端<br>1：AHB1->源端；AHB0->目的端<br>2：AHB0->源端；AHB0->目的端<br>3：AHB1->源端；AHB1->目的端 | RW |
| [1]: CH_RST       | 初始值：0x0，通道复位                                                                                                       | RW |
| [0]: CH_EN        | 初始值：0x0，通道使能，通道不使用的時候請關閉該通道                                                                                        | RW |

**STA[31: 0]寄存器:**

| 位                | 功能                                                                                                                                                                                  | 类型 |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| [15: 8]: ERR_STA | 初始值: 0x0, 通道错误状态<br>8'b0000_0000: 无错误<br>8'bxxxx_xxx1: 源端口错误<br>8'bxxxx_xx1x: 目的端口错误<br>8'bxxxx_x1xx: 软件握手错误<br>8'bxxxx_1xxx: 硬件握手错误<br>8'bxxx1_xxxx: FIFO 错误<br>8'bxx1x_xxxx: 超时错误 | RO |
| [3: 0]: CH_STA   | 初始值: 0x0, 通道状态<br>4'b0000: 当前无传输操作<br>4'b0001: 正在传输<br>4'b0010: 传输暂停                                                                                                                | RO |

**IE[31: 0]寄存器:**

| 位                  | 功能                    | 类型 |
|--------------------|-----------------------|----|
| [8]: IE_TRANS_ERR  | 初始值: 0x0, 通道错误中断      | RW |
| [4]: IE_TRANS_DONE | 初始值: 0x0, 通道传输结束中断    | RW |
| [3]: IE_BLK_DONE   | 初始值: 0x0, 通道传输块传输结束中断 | RW |
| [2]: IE_STOP       | 初始值: 0x0, 通道停止中断      | RW |
| [1]: IE_PAUSE      | 初始值: 0x0, 通道暂停中断      | RW |
| [0]: IE_START      | 初始值: 0x0, 通道启动中断      | RW |

**IS[31: 0]寄存器:**

| 位                  | 功能                      | 类型 |
|--------------------|-------------------------|----|
| [8]: IS_TRANS_ERR  | 初始值: 0x0, 通道错误中断状态      | RC |
| [4]: IS_TRANS_DONE | 初始值: 0x0, 通道传输结束中断状态    | RC |
| [3]: IS_BLK_DONE   | 初始值: 0x0, 通道传输块传输结束中断状态 | RC |
| [2]: IS_STOP       | 初始值: 0x0, 通道停止中断状态      | RC |
| [1]: IS_PAUSE      | 初始值: 0x0, 通道暂停中断状态      | RC |
| [0]: IS_START      | 初始值: 0x0, 通道启动中断状态      | RC |

**TO[31: 0]寄存器:**

| 位                | 功能                      | 类型 |
|------------------|-------------------------|----|
| [31: 0]: TIMEOUT | 初始值: 0x100000, 通道传输超时计数 | RW |



BUF[31: 0]寄存器:

| 位                | 功能                                                                                                                                                                              | 类型 |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| [19: 16]: WT_WM  | 初始值: 0x4, 目的端写出 FIFO 水线控制<br>0: 只要有数据就启动写出<br>1: 1/8 深度以上启动写出<br>2: 2/8 深度以上启动写出<br>3: 3/8 深度以上启动写出<br>4: 4/8 深度以上启动写出<br>5: 5/8 深度以上启动写出<br>6: 6/8 深度以上启动写出<br>7: 7/8 深度以上启动写出 | RW |
| [11: 8]: RD_WM   | 初始值: 0x4, 源端读入 FIFO 水线控制<br>0: 只要有空间就启动读入<br>1: 1/8 深度以下启动读入<br>2: 2/8 深度以下启动读入<br>3: 3/8 深度以下启动读入<br>4: 4/8 深度以下启动读入<br>5: 5/8 深度以下启动读入<br>6: 6/8 深度以下启动读入<br>7: 7/8 深度以下启动读入  | RW |
| [3: 0]: BLK_SIZE | 初始值: 0x5, 单次总线 burst 传输最大长度 0:<br>burst_block_size=LEN<br>other: burst_block_size=2 <sup>BLK_SIZE</sup> Bytes                                                                   | RW |

注: 寄存器地址中的 X 为 0 到 7, 分别代表 8 条通道地址。

### 12.1.1 通道地址及长度控制

通道源端和目的端的初始地址有 SA 和 DA 两个寄存器配置。传输长度由 LEN 寄存器控制。源端和目的端的地址控制方式相同, 分别有 SA\_MODE 和 DA\_MODE 配置。四种地址控制方式如下:

|   |                                                                                                                                                                                                           |
|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | 递增: 按总线位宽实际操作地址递增                                                                                                                                                                                         |
| 2 | 固定: 地址锁定模式, 传输过程中不会变化, 常用于搬移 FIFO 中的数据                                                                                                                                                                    |
| 3 | 跳变: 基地址由寄存器配置, 当传输数据长度达到 SOA 或 DOA 的高 16 位的时候, 将下一个传输地址加上 SOA 或 DOA 的低 16 位的数值, 然后开始继续传输。当继续传输数据的长度再次到达 SOA 或 DOA 的高 16 位数字的时候, 将传输地址继续加上 SOA 或 DOA 的低 16 位的数字。后续不断重复上述流程, 直到总传输长度到达寄存器 LEN 的配置, 然后结束通道传输 |
| 4 | 环回: 基地址由寄存器配置, 当传输数据长度达到 SOA 或 DOA 的低 16 位的时候, 将下个传输数据地址重新设置为寄存器配置的 SA 或 DA 的地址, 然后继续重复上述流程, 直到总传输长度到达寄存器 LEN 的配置, 然后结束通道传输                                                                               |

### 12.1.2 握手模式

SDMA 支持软件和硬件两种握手模式，软件握手模式下 SDMA 有软件控制启动；硬件握手模式下，SDMA 由软件配置地址和传输模式等信息，由硬件模块根据硬件的调度启动传输。此芯片只有 MDIO 和 UART 支持硬件握手。

硬件握手模式下软件参与度较少，调度不够灵活，一般情况下我们使用软件握手模式即可完成所有 DMA 操作需求。

软件握手模式比较灵活，且软件握手模式支持硬件流控，使能硬件流控后不会降低 DMA 搬移速度，但是将有效提高总线利用效率。

### 12.1.3 通道级联

在某些特殊的应用下我们需要一条 DMA 通道完成之后继续调用另外一条通道来搬移数据，SDMA 中的 CHAIN\_EN 和 CHAIN\_TO 两个寄存器用来完成此功能。第一条通道不需要配置这两个寄存器，后续通道均需要配置 CHAIN\_EN 为 1，CHAIN\_TO 为上一条通道的编号。全部配置完成之后，软件只需要启动第一条通道即可，当第一条通道完成之后硬件会自动启动下一条通道，直到所有级联的通道完成。

## 12.2 SDMA 通用配置

### 12.2.1 SDMA 通道优先级

SDMA8 条通道支持配置为不同的优先级，软件可以根据实际应用需要将不同的通道配置为不同的优先级。SDMA 通道优先级配置寄存器如下：

| 地址         | 寄存器名称      |
|------------|------------|
| 0xF0100X60 | ARB[31: 0] |

ARB[31: 0]寄存器：

| 位                           | 功能                                                                                                       | 类型 |
|-----------------------------|----------------------------------------------------------------------------------------------------------|----|
| [15]: CH_ARB_FIX            | 初始值：0x0，通道优先级方案调整                                                                                        | RW |
| [10: 8]: PRI                | 初始值：0x0，通道优先级，值越大优先级越高                                                                                   | RW |
| [1]: CH_ARB<br>[0]: BLK_ARB | 初始值：2'b00<br>2'b00：同等优先级下，不同通道按总线 burst block 切换<br>2'b01：锁定总线仲裁器，一次完成整个通道传输<br>2'b1x：允许 word 级别，总线优先级切换 | RW |

注：寄存器地址中的 X 为 0 到 7，分别代表 8 条通道地址。

## 13 UART串行接口

芯片内置 8 组通用 UART 接口，此接口支持如下特性：

|    |                                     |
|----|-------------------------------------|
| 1  | 支持奇偶校验                              |
| 2  | 支持不同的波特率配置                          |
| 3  | 支持 5、6、7、8 数据位                      |
| 4  | 支持 1 或 2 个停止位                       |
| 5  | 支持帧错误检测                             |
| 6  | 内置 256Byte 发送 FIFO                  |
| 7  | 内置 256Byte 接收 FIFO                  |
| 8  | 支持可配置频率及占空比的方波调制，用于支持红外传输           |
| 9  | 支持波特率速度检测，可用于自适应波特率                 |
| 10 | 支持可屏蔽中断                             |
| 11 | 支持接收数据 DMA 硬件握手传输模式                 |
| 12 | 支持 UART MASTER DEBUG 模式（仅 UART0 支持） |

UART 模块系统结构图如下：

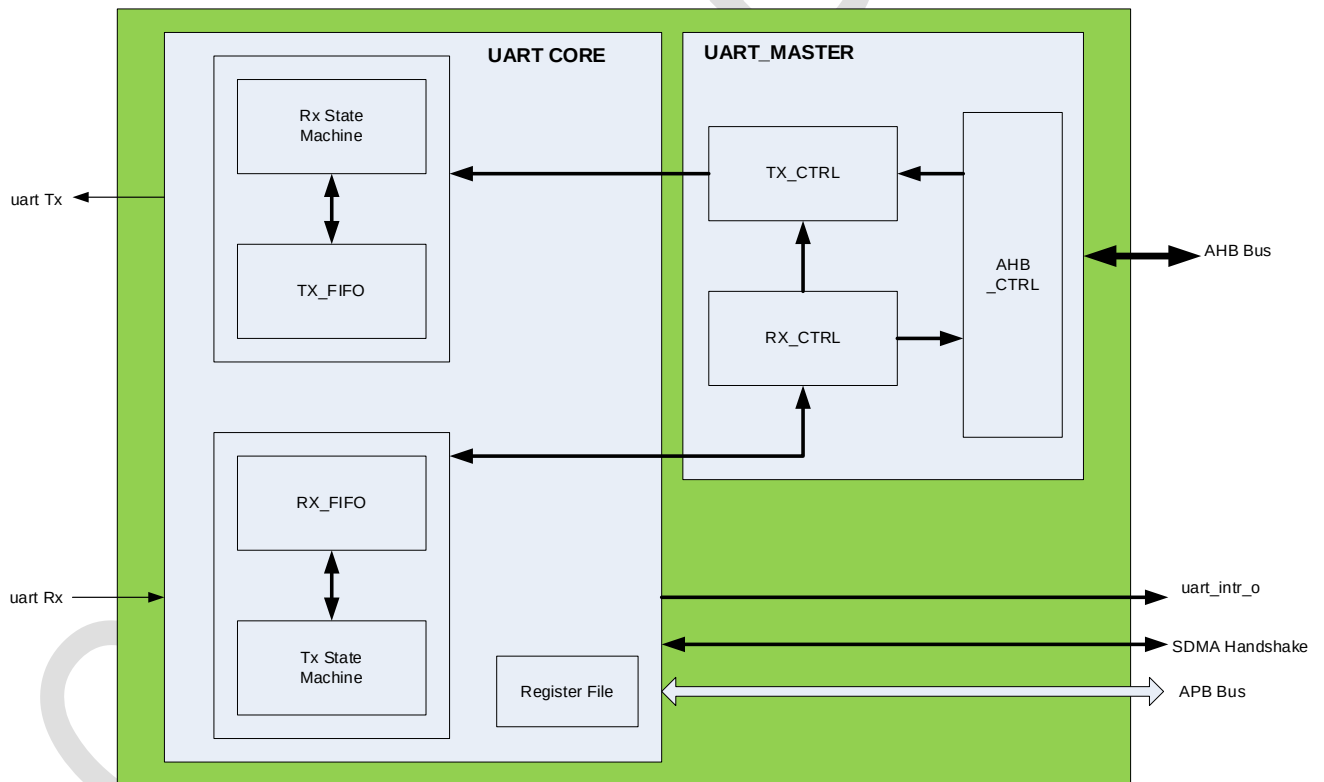


图 13-1 UART 系统结构图

UART 模块相关寄存器如下表：

| 地址         | 寄存器名称                     |
|------------|---------------------------|
| 0xF2X00000 | BUFR[31: 0]               |
| 0xF2X00004 | IER[31: 0]                |
| 0xF2X00008 | IIFCR[31: 0]              |
| 0xF2X0000C | LCR[31: 0]                |
| 0xF2X00014 | LSR[31: 0]                |
| 0xF2X003C0 | TF_CNT[31: 0]             |
| 0xF2X003C4 | RF_CNT[31: 0]             |
| 0xF2X003D0 | BAUD_DET_CTRL[31: 0]      |
| 0xF2X003D4 | BAUD_MAX_CNT[31: 0]       |
| 0xF2X003D8 | BAUD_MIN_CNT[31: 0]       |
| 0xF2X003DC | BAUD_DET_CNT[31: 0]       |
| 0xF2X003E0 | IR_CTRL[31: 0]            |
| 0xF2X003E4 | CARRIER_PERIOD_CFG[31: 0] |

BUFR[31: 0]寄存器：

| 位            | 功能                                     | 类型 |
|--------------|----------------------------------------|----|
| [7: 0]: FIFO | 初始值：0x0，写操作：填充发送 FIFO<br>读操作：读取接收 FIFO | RW |

IER[31: 0]寄存器：

| 位            | 功能                       | 类型 |
|--------------|--------------------------|----|
| [6]: IE_TWM  | 初始值：0x0，发送 FIFO 低于水线中断使能 | RW |
| [4]: IE_RTO  | 初始值：0x0，接收 FIFO 非空超时中断使能 | RW |
| [2]: IE_RLS  | 初始值：0x0，接收帧错误中断使能        | RW |
| [1]: IE_THRE | 初始值：0x0，发送 FIFO 空中断使能    | RW |
| [0]: IE_RDA  | 初始值：0x0，接收 FIFO 超过水线中断使能 | RW |

IIFCR[31: 0]寄存器：

| 位              | 功能                                                                                                                                                                                                                                                                                                                 | 类型 |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| [31: 16]: IIFT | 初始值：0x3ff，接收 FIFO 非空中断超时时间，PCLK2 在 50MHz，默认值为 3072us                                                                                                                                                                                                                                                               | RW |
| [15: 8]: ITWM  | 初始值：0x10，发送 FIFO 中断水线                                                                                                                                                                                                                                                                                              | RW |
| [7: 0]: IIFC   | 初始值：0xe0，此寄存器读取时：<br>[0]：为 0 时中断状态有效，为 1 时中断状态无效<br>[3: 1]=3'b011：接收帧错误中断状态<br>[3: 1]=3'b010：接收数据超过水线中断状态<br>[3: 1]=3'b001：发送 FIFO 为空中断状态<br>[7: 4]：接收 FIFO 中断触发水线配置<br>此寄存器写入时：<br>[0]：写 1 时清空接收 FIFO<br>[0]：写 1 时清空发送 FIFO<br>[7: 4]：接收 FIFO 中断触发水线，中断数目为<br>[7: 4]<4；下面两个特殊值例外：4'b0000：1Byte<br>4'b1111：256Byte | RW |

**LCR[31: 0]寄存器:**

| 位               | 功能                                                                                                                                                                                                                                                                                   | 类型 |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| [7]: LC_DL      | 初始值: 0x0, 寄存器第二功能开关, 当此寄存器配置为 0 时, 寄存器维持原始功能。当此寄存器为 1 时: 写 BUFR 寄存器将配置波特率分频参数的低 8 位写 IER 寄存器配置波特率分频参数高 8 位。波特率分频参数计算公式为 $DIV = PCLK2 / \text{BaudRate} / 16$ 写 IIFT 寄存器低 8 位, 将配置接收超时单位时间单元, 默认值为 149, $PCLK2 = 50\text{MHz}$ 时一个时间单元是 3us。接收超时默认时间 $3us * IIFT(0x3ff + 1) = 3072us$ | RW |
| [6]: LC_BC      | 初始值: 0x0, 1: 强制 TX 输出为 0<br>0: TX 正常输出                                                                                                                                                                                                                                               | RW |
| [5]: LC_SP      | 初始值: 0x0, 0: 使用数据 0 进行校验<br>1: 使用数据 1 进行校验                                                                                                                                                                                                                                           | RW |
| [4]: LC_EP      | 初始值: 0x0, 0: 奇(odd)校验<br>1: 偶(even)校验                                                                                                                                                                                                                                                | RW |
| [3]: LC_PE      | 初始值: 0x0, 校验使能                                                                                                                                                                                                                                                                       | RW |
| [2]: LC_SB      | 初始值: 0x0, 0: 使用 1bit 停止位<br>1: 使用 2bit 停止位                                                                                                                                                                                                                                           | RW |
| [1: 0]: LC_BITS | 初始值: 0x0, 2'b00: 使用 5bit 数据<br>2'b01: 使用 6bit 数据<br>2'b10: 使用 7bit 数据<br>2'b11: 使用 8bit 数据                                                                                                                                                                                           | RW |

**LSR[31: 0]寄存器:**

| 位                 | 功能                         | 类型 |
|-------------------|----------------------------|----|
| [31: 16]: LS_RFTO | 初始值: 0x0, 接收 FIFO 非空计时器数值  | RO |
| [9]: LS_TWM       | 初始值: 0x0, 发送 FIFO 低于水位线状态  | RC |
| [8]: LS_RTO       | 初始值: 0x0, 接收 FIFO 非空超时状态   | RC |
| [7]: LS_EI        | 初始值: 0x0, 接收 FIFO 数据中有校验错误 | RC |
| [6]: LS_TE        | 初始值: 0x0, 发送完成             | RO |
| [5]: LS_TFE       | 初始值: 0x0, 发送 FIFO 空        | RO |
| [4]: LS_BI        | 初始值: 0x0, UART break 中断状态  | RC |
| [3]: LS_FE        | 初始值: 0x0, 帧错误中断状态          | RC |
| [2]: LS_PE        | 初始值: 0x0, 校验错误中断状态         | RC |
| [1]: LS_OE        | 初始值: 0x0, 接收 FIFO 溢出中断状态   | RC |
| [0]: LS_DR        | 初始值: 0x0, 接收 FIFO 中有数据     | RO |

**TF\_CNT[31: 0]寄存器:**

| 位             | 功能                       | 类型 |
|---------------|--------------------------|----|
| [8: 0]: T_CNT | 初始值: 0x0, 发送 FIFO 中的数据长度 | RW |

**RF\_CNT[31: 0]寄存器:**

| 位             | 功能                       | 类型 |
|---------------|--------------------------|----|
| [8: 0]: R_CNT | 初始值: 0x0, 接收 FIFO 中的数据长度 | RW |

AUD\_DET\_CTRL[31: 0]寄存器:

| 位                 | 功能                   | 类型 |
|-------------------|----------------------|----|
| [1]: BAUD_DET_CLR | 初始值: 0x0, 清空当前波特率检测值 | AO |
| [0]: BAUD_DET_EN  | 初始值: 0x0, 使能波特率检测功能  | RW |

BAUD\_MAX\_CNT[31: 0]寄存器:

| 位                     | 功能                          | 类型 |
|-----------------------|-----------------------------|----|
| [31: 0]: BAUD_MAX_CNT | 初始值: 0xffff ffff, 波特率检测的最大值 | RW |

BAUD\_DET\_CNT[31: 0]寄存器:

| 位                     | 功能                  | 类型 |
|-----------------------|---------------------|----|
| [31: 0]: BAUD_DET_CNT | 初始值: 0x0, 波特率检测的最小值 | RW |

BAUD\_MIN\_CNT[31: 0]寄存器:

| 位                     | 功能                                       | 类型 |
|-----------------------|------------------------------------------|----|
| [31: 0]: BAUD_MIN_CNT | 初始值: 0xffff ffff, 检测到的波特率值, 按 PCLK2 数目计算 | RO |

IR\_CTRL[31: 0]寄存器:

| 位                   | 功能                                             | 类型 |
|---------------------|------------------------------------------------|----|
| [5]: UART_RX_INVERT | 初始值: 0x0, IR 模式下反转 RX 信号                       | RW |
| [4]: UART_TX_INVERT | 初始值: 0x0, IR 模式下反转 TX 信号                       | RW |
| [1]: IR_CARRIER_MOD | 初始值: 0x0, 1: 载波信号与 TX 信号相与<br>0: 载波信号与 TX 信号相或 | RW |
| [0]: IR_CARRIER_EN  | 初始值: 0x0, 红外载波使能                               | RW |

CARRIER\_PERIOD\_CFG[31: 0]寄存器:

| 位                            | 功能                                  | 类型 |
|------------------------------|-------------------------------------|----|
| [31: 16]: CARRIER_CLK_PERIOD | 初始值: 0x522, 红外载波周期, PCLK2 时钟数目      | RW |
| [15: 0]: CARRIER_HIGH_PERIOD | 初始值: 0x1b5, IR 红外载波高电平持续 PCLK2 周期数目 | RW |

注: X=5-C, 分别代表 UART0~7。

### 13.1 UART 波特率

UART 波特率配置流程如下:

- `SET\_UART\_LC\_DL(1, uart\_idx)
- `SET\_UART\_FIFO(baud\_div[7: 0], uart\_idx) //baud\_rate set
- `WR\_UART\_IER(baud\_div[15: 8], uart\_idx)//baud\_rate set
- `SET\_UART\_LC\_DL(0, uart\_idx)

波特率分频参数配置如寄存器说明所示, 例如目标波特率为 115200, PCLK2 工作于 50MHz。  
 $\text{baud\_div} = 50 \times 106 / 16 / 115200 = 27.12$ 。实际配置的时候我们取 27 配入寄存器中, 这样系统的实际波特率为  $50 \times 106 / 16 / 27 = 115740$ 。

### 13.1.1 波特率检测

UART 模块内置波特率检测电路，可用于在不知道 `uart` 另外一端设备波特率的时候，自适应的调整此设备的波特率。此检测电路通过检测 `RX` 信号上的波特率来实现此功能。也就是说如果要使用此功能话，`UART RX` 信号线必须要有信号输入。使用流程如下：

- 打开 `BAUD_DET_EN`。
- 配置 `BAUD_MAX_CNT` 及 `BAUD_MIN_CNT`，确认检测波特率的范围。
- 配置 `BAUD_DET_CLR` 为 1，清空波特率统计值。
- 等待一段时间，确保这段时间中，`UART` 能够接收到数据。
- 读取 `BAUD_DET_CNT`，计算波特率，可将此值直接右移 4 位（除 16）并配置为新的波特率。

注：受限于 `UART` 的通信模式，此模块在 50MHz `PCLK2` 时钟下最大可稳定支持检测 57600 波特率。如需要支持更高的波特率，需要软件对检测的结果进行修正。

### 13.2 UART 中断

`UART` 模块支持多个中断源，具体中断源如寄存器中 `IER` 所示。中断使能及中断状态查询关系表如下：

| 中断号                  | <code>IIFCR[3: 0]</code> read | <code>LSR[9: 0]</code> | 说明                                                       |
|----------------------|-------------------------------|------------------------|----------------------------------------------------------|
| <code>IE_RDA</code>  | 4'b0100                       | NA                     |                                                          |
| <code>IE_THRE</code> | 4'b0010                       | 10'b00_0010_0000       |                                                          |
| <code>IE_RLS</code>  | 4'b0110                       | 10'b00_0001_1110       | <code>LSR[4: 1]</code> 只要有一个为 1 就会触发 <code>RLS</code> 中断 |
| <code>IE_RTO</code>  | 4'b1000 无效                    | 10'b01_0000_0000       |                                                          |
| <code>IE_TWM</code>  | 4'b1000 无效                    | 10'b10_0000_0000       |                                                          |

软件进入中断后先查询 `IIFCR[3: 0]`中是否有中断状态，没有中断状态的话，继续查询 `LSR` 中的相关状态。

`IE_RTO` 为接收 `FIFO` 非空超时中断，当 `UART` 接收到数据之后就会开始计时，当有新的数据进入后会清零超时计数器。超时计时器由两个部分组成，`LC_DL` 为 1 时 `IIFT` 低 8 位有效，配置的是计时单位，默认值为 149，`PCLK2` 为 50MHz 时，一个时间单位是 3us；当 `LC_DL` 为 0 时 `IIFT` 为 16 位计数器。`RTO` 超时时间为这两个计数器的乘积。

### 13.3 红外载波控制

`UART` 模块支持将 `UART TX` 发送信号进行红外载波调制。支持对 `TX` 信号的高电平或低电平调制；并支持将 `TX` 信号反转后再进行调制。具体的调制模式需要根据具体的外围硬件电路来确定。具体配置说明请参考寄存器说明。

### 13.4 UART DEBUG 模式

`UART0` 模块支持硬件 `DEBUG` 模式，主要用于工程调试，默认波特率 115200bps。当此功能使能的时候，可以通过 `uart` 端口直接访问芯片内部的寄存器。软件启动后，此功能关闭。

`UART DEBUG` 模式支持如下两条命令：

```
read    0xFFFFFFFF          // read reg
write   0xFFFFFFFF 0xFFFFFFFF // write reg
```

## 14 SPI MASTER

芯片内置 6 组 SPI MASTER 模块，支持如下特性：

|   |                                       |
|---|---------------------------------------|
| 1 | 支持 FLASH 用于 CPU 启动                    |
| 2 | 最大支持 4 个 SLAVE 设备                     |
| 3 | 支持 SCK 时钟占空比频率调整                      |
| 4 | 支持 SPI 模式调整                           |
| 5 | 支持聚合 MISO、MOSI 信号线，使用 3 根信号线完成 SPI 操作 |
| 6 | 支持 SD(TF)卡 SPI 模式读写                   |
| 7 | 支持硬件中断                                |

SPI master 相关寄存器如下表：

| 地址         | 寄存器名称            |
|------------|------------------|
| 0xF0X00000 | INTR_EN[31: 0]   |
| 0xF0X00004 | INTR_STA[31: 0]  |
| 0xF0X00010 | BASIC_CFG[31: 0] |
| 0xF0X00014 | CS_CFG[31: 0]    |
| 0xF0X00020 | OP_CFG[31: 0]    |
| 0xF2000024 | SPI_WDATA[31: 0] |
| 0xF0X00028 | SPI_RDATA[31: 0] |

INTR\_EN[31: 0]寄存器：

| 位                | 功能                                     | 类型 |
|------------------|----------------------------------------|----|
| [0]: SPI_INTR_EN | 初始值：0x0，SPI 中断使能，使能之后，当 SPI 操作结束后会产生中断 | RW |

INTR\_STA[31: 0]寄存器：

| 位             | 功能              | 类型 |
|---------------|-----------------|----|
| [0]: SPI_BUSY | 初始值：0x0，SPI 忙状态 | RO |

BASIC\_CFG[31: 0]寄存器：

| 位                 | 功能                                                                                                            | 类型 |
|-------------------|---------------------------------------------------------------------------------------------------------------|----|
| [31: 16]BASIC_CFG | 初始值：0x0，SPI sck 生成方式：<br>$Sck = spi\_sys\_clk / ((spi\_sck\_low\_period + 1) + (spi\_sck\_high\_period + 1))$ | RW |
| [13: 12]BASIC_CFG | 初始值：0x0，Spi 输入数据延时控制<br>0: delay 0 cycle 1: delay half cycle 2: delay 1 cycle                                 | RW |
| [11: 4]BASIC_CFG  | 初始值：0x03，基于 spi_sys_clk，Spi cs 保持的周期数                                                                         | RW |
| [3]: BASIC_CFG    | 初始值：0x0，0: 标准 SPI 4 线模式<br>1: SPI 3 线模式                                                                       | RW |
| [2]: BASIC_CFG    | 初始值：0x0，MOSI/MISO 数据极性选择                                                                                      | RW |
| [1]: BASIC_CFG    | 初始值：0x0，SCK 时钟极性选择                                                                                            | RW |
| [0]: BASIC_CFG    | 初始值：0x1，SPI master 功能使能                                                                                       | RW |



**CS\_CFG[31: 0]寄存器:**

| 位              | 功能                                                                             | 类型 |
|----------------|--------------------------------------------------------------------------------|----|
| CS_CFG[15: 14] | 初始值: 0x01, Spi hold 使能<br>Bit[0]=0, 选择 Spi_hold 为使能<br>Bit[0]=1, 选择 bit[1]为使能  | RW |
| CS_CFG[13: 12] | 初始值: 0x01, Spi wp 使能<br>Bit[0]=0, 选择 Spi_wp 为使能<br>Bit[0]=1, sel bit[1]为使能     | RW |
| CS_CFG[11: 10] | 初始值: 0x03, Spi miso 使能<br>Bit[0]=0, 选择 Spi_miso 为使能<br>Bit[0]=1, sel bit[1]为使能 | RW |
| CS_CFG[9: 8]   | 初始值: 0x01, Spi mosi 使能<br>Bit[0]=0, 选择 Spi_mosi 为使能<br>Bit[0]=1, 选择 bit[1]为使能  | RW |
| CS_CFG[5]      | 初始值: 0x0, SPI CS 设置值                                                           | RW |
| CS_CFG[4]      | 初始值: 0x0, 如果 bit=1'b1, CS 片选由 bit[5]驱动                                         | RW |
| CS_CFG[3: 0]   | 初始值: 0x1, SPI CS0-3 选择                                                         | RW |

**OP\_CFG[31: 0]寄存器:**

| 位              | 功能                                                                                                         | 类型 |
|----------------|------------------------------------------------------------------------------------------------------------|----|
| [11]: OP_CFG   | 初始值: 0x0, Spi_hold 方向控制<br>1: MOSI 输入<br>0: MOSI 输出                                                        | RW |
| [11]: OP_CFG   | 初始值: 0x0, Spi_wp 方向控制<br>1: MOSI 输入<br>0: MOSI 输出                                                          | RW |
| [9]: OP_CFG    | 初始值: 0x1, Spi_miso 方向控制<br>1: MOSI 输入<br>0: MOSI 输出                                                        | RW |
| [8]: OP_CFG    | 初始值: 0x0, Spi_mosi 方向控制<br>1: MOSI 输入<br>0: MOSI 输出                                                        | RW |
| [7]: OP_CFG    | 初始值: 0x0, Spi quad mode 使能                                                                                 | RW |
| [6]: OP_CFG    | 初始值: 0x0, Spi dual mode 使能                                                                                 | RW |
| [5: 4]: OP_CFG | 初始值: 0x0, Spi 传输字节数<br>0: 4-bytes 1: 1-bytes 2: 2-bytes 3: 3-bytes                                         | RW |
| [3]: OP_CFG    | 初始值: 0x0, 传输字节 MSB<br>0: wdata={wdata[7: 0], wdata[15: 8],<br>wdata[23: 16], wdata[31: 24]} 1: wdata=wdata | RW |
| [2]: OP_CFG    | 初始值: 0x0, 如果该位设置为 1'b1, 则在此操作之后, cs 将断言                                                                    | RW |
| [1]: OP_CFG    | 初始值: 0x0, SPI 最低位先传输                                                                                       | RW |
| [0]: OP_CFG    | 初始值: 0x0, SPI 最高位先传输                                                                                       | RO |

SPI\_WDATA[31: 0]寄存器:

| 位                  | 功能                    | 类型 |
|--------------------|-----------------------|----|
| [31: 0]: SPI_WDATA | 初始值: 0x0, SPI 移位数据寄存器 | RW |

SPI\_RDATA[31: 0]寄存器:

| 位                  | 功能                 | 类型 |
|--------------------|--------------------|----|
| [31: 0]: SPI_RDATA | 初始值: 0x0, SPI 读取数据 | RW |

### 14.1 SPI SCK 及模式控制

SPI\_SCK\_LOW\_PERIOD 和 SPI\_SCK\_HIGH\_PERIOD 两个寄存器用于控制 SCK 信号的时钟频率, 具体的计算方式请参考寄存器描述。

CPOL 及 CPHA 两个寄存器用于控制 SPI 模式, 具体的时序图如下:

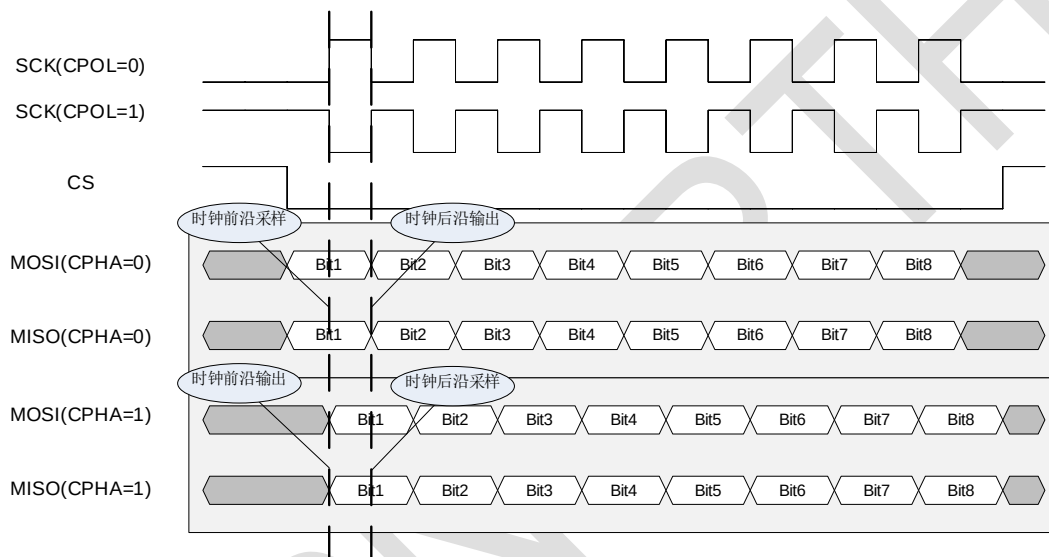


图 14-1 SPI 模式选择时序图

### 14.2 SPI CS 片选控制

SPI master CS 信号支持两种控制方式:

- 硬件自动控制
- 软件直接控制

硬件自动控制模式需要配置 SPI\_CFG\_CSS\_EN 为 0, SPI 在发送第一个数据的时候将 CS 信号拉低, 如果 SPI\_CSS\_STOP 寄存器为 0, 那么在这个数据发送完成后 CS 信号保持为低 (片选中状态), 软件在发送最后一个数据之前将 SPI\_CSS\_STOP 配置为 1, 那么硬件在最后一个数据发送完成后将 CS 拉高 (撤销片选)。

软件直接控制模式需要配置 SPI\_CFG\_CSS\_EN 为 1, SPI 的片选信号受 SPI\_CFG\_CSS 直接控制, SPI\_CFG\_CSS 为 1 的时候片选信号为低 (选中状态)。

SPI master 支持 4 个 CS 信号, 由 SPI\_CSS\_EN 寄存器控制, 每次 SPI 操作可以选择一个或者多个, 选择多个的时候需要注意 MISO 的冲突情况。

### 14.3 SPI BOOT FLASH 器件限制

此 SPI 端口在系统启动的时候用于 CPU 读取程序内容。对 SPI FLASH 器件的限制如下：

- FLASH 读数据指令为 0x03，读取时序必须如下图所示：

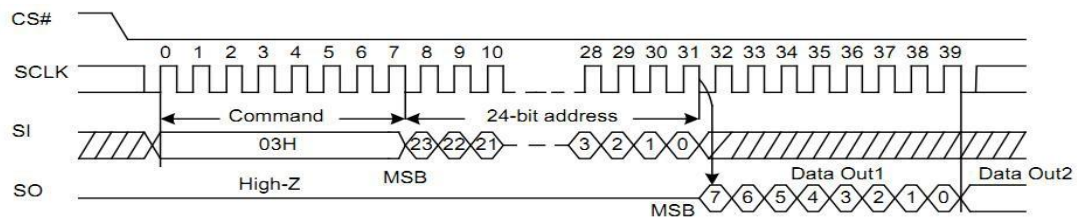


图 14-2 SPI Boot Flash 读取时序

- FLASH 需要支持 CPOL=0，CPHA=0 模式
- FLASH SCK 需要支持到 25MHz 以上。

## 15 SPI\_SLAVE

芯片集成 2 组通用的 SPI\_SLAVE 模块，可以与 SPI\_MASTER 直接通信。此 SPI\_SLAVE 模块的特性如下：

|   |                         |
|---|-------------------------|
| 1 | 支持 8/16/32bits apb 总线配置 |
| 2 | 独立的屏蔽控制位对应相关的设置         |
| 3 | 可配置的接收延时保证数据采样的正确       |
| 4 | 可配置的接收数据帧大小             |
| 5 | 支持中断                    |

SPI\_SLAVE 系统框图如下所示：

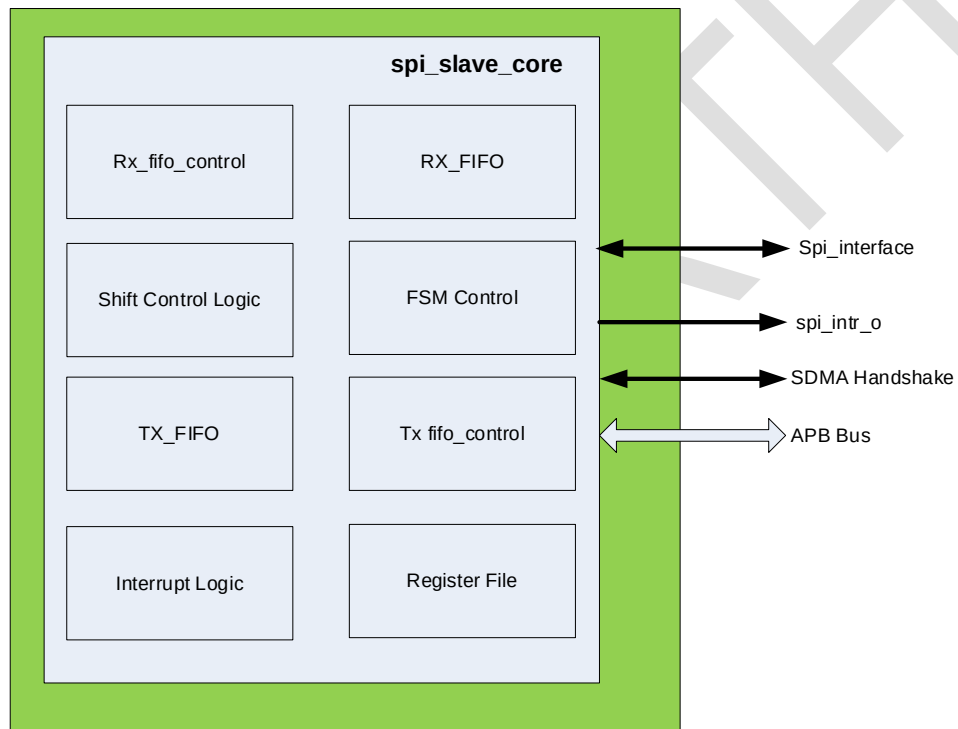


图 15-1 SPI\_SLAVE 系统结构图

SPI\_SLAVE 功能相关寄存器如下：

| 地址         | 寄存器名称         |
|------------|---------------|
| 0xF2200000 | CTRLR[31: 0]  |
| 0xF2200008 | SSIENR[31: 0] |
| 0xF2200018 | TXFTLR[31: 0] |
| 0xF220001C | RXFTLR[31: 0] |
| 0xF2200020 | TXFLR[31: 0]  |
| 0xF2200024 | RXFLR[31: 0]  |
| 0xF2200028 | SR[31: 0]     |
| 0xF220002C | IMR[31: 0]    |
| 0xF2200034 | RISR[31: 0]   |

**CTRLR[31: 0]寄存器:**

| 位               | 功能                                                                                       | 类型 |
|-----------------|------------------------------------------------------------------------------------------|----|
| [15: 12]: CTRLR | 初始值: 0x0, 选择 Microwire frame format 控制字的传送长度                                             |    |
| [11]: SRL       | 初始值: 0x0, 1: Loop back 测试模式<br>0: 常规模式<br>1: 测试模式                                        | RW |
| [10]: SLV_OE    | 初始值: 0x0, Slave output 使能<br>0: TXD 使能<br>1: TXD 关闭                                      | RW |
| [9: 8]: TMOD    | 初始值: 0x0, 2'b00: 发送接收同时使能<br>2'b00: 发送、接收<br>2'b01: 仅发送<br>2'b10: 仅接收<br>2'b11: EEPROM 读 | RW |
| [7]: SCPOL      | 初始值: 0x0, Spi 时钟极性选择<br>0: 时钟初始状态为低电平<br>1: 时钟初始状态为高电平                                   | RW |
| [6]: SCPH       | 初始值: 0x0, Spi 时钟相位选择<br>0: SCK 时钟的第二个边沿进行数据的采样<br>1: SCK 时钟的第一个边沿进行数据的采样                 | RW |
| [5: 4]: SFS     | 初始值: 0x0, Frame 格式选择<br>2'b00: Motorola spi 2'b01: TI SSP<br>2'b10: microwire            |    |
| [3: 0]: DFS     | 初始值: 0x7, Frame 数据大小<br>Data size=value+1, 不满足 16bits 数据右对齐, 高位补 0                       | RW |

**SSIENR[31: 0]寄存器:**

| 位           | 功能                                         | 类型 |
|-------------|--------------------------------------------|----|
| [0]: SSI_EN | 初始值: 0x0, SPI SLAVE 功能使能<br>0: 关闭<br>1: 使能 | RW |

**TXFTLR[31: 0]寄存器:**

| 位           | 功能                   | 类型 |
|-------------|----------------------|----|
| [4: 0]: TFT | 初始值: 0x0, 发送 FIFO 阈值 | RW |

**RXFTLR[31: 0]寄存器:**

| 位           | 功能                   | 类型 |
|-------------|----------------------|----|
| [4: 0]: RFT | 初始值: 0x0, 接收 FIFO 阈值 | RW |

**TXFLR[31: 0]寄存器:**

| 位             | 功能                      | 类型 |
|---------------|-------------------------|----|
| [5: 0]: TXTFL | 初始值: 0x0, 发送 FIFO 中的数据量 | RO |

## RXFLR[31: 0]寄存器:

| 位             | 功能                      | 类型 |
|---------------|-------------------------|----|
| [5: 0]: RXTFL | 初始值: 0x0, 接收 FIFO 中的数据量 | RO |

## SR[31: 0]寄存器:

| 位         | 功能                                       | 类型 |
|-----------|------------------------------------------|----|
| [5]: TXE  | 初始值: 0x0, 发送错误                           | RC |
| [4]: RFF  | 初始值: 0x0, 接收 FIFO 满                      | RO |
| [3]: RFNE | 初始值: 0x0, 接收 FIFO 空                      | RO |
| [2]: TFE  | 初始值: 0x0, 发送 FIFO 空                      | RO |
| [1]: TFNF | 初始值: 0x0, 发送 FIFO 满                      | RO |
| [0]: BUSY | 初始值: 0x0, SPI SLAVE 忙状态<br>0: 不忙<br>1: 忙 | RO |

## IMR[31: 0]寄存器:

| 位          | 功能                          | 类型 |
|------------|-----------------------------|----|
| [4]: RXFIM | 初始值: 0x1, 接收 FIFO Full 中断掩码 | RW |
| [3]: RXOIM | 初始值: 0x1, 接收 FIFO 溢出中断掩码    | RW |
| [2]: RXUIM | 初始值: 0x1, 接收 FIFO 向下溢出中断掩码  | RW |
| [1]: TXOIM | 初始值: 0x1, 发送 FIFO 溢出中断掩码    | RW |
| [0]: TXEIM | 初始值: 0x1, 发送 FIFO 空中断掩码     | RW |

## RISR[31: 0]寄存器:

| 位          | 功能                       | 类型 |
|------------|--------------------------|----|
| [4]: RXFIR | 初始值: 0x0, 对应于上一个寄存器的中断状态 | RO |
| [3]: RXOIR |                          |    |
| [2]: RXUIR |                          |    |
| [1]: TXOIR |                          |    |
| [0]: TXEIR |                          |    |

## 16 MDIO

芯片内置 MDIO 模块，用于控制 RMII 接口的 PHY 芯片。此模块支持如下特性：

- MDC 时钟周期可调整
- 支持 PHY 和 MAC 两种模式

MDIO 模块寄存器如下：

| 地址         | 寄存器名称         |
|------------|---------------|
| 0xF2400020 | MCFG[31: 0]   |
| 0xF2400024 | MCMD[31: 0]   |
| 0xF2400028 | MADR[31: 0]   |
| 0xF240002C | MWTD[31: 0]   |
| 0xF2400030 | MRDD[31: 0]   |
| 0xF2400034 | MIND[31: 0]   |
| 0xF240004C | PHYID[31: 0]  |
| 0xF2400050 | PHYSTA[31: 0] |
| 0xF2400078 | MGDI[31: 0]   |
| 0xF240007C | MGDO[31: 0]   |

MCFG[31: 0]寄存器：

| 位            | 功能                                                 | 类型 |
|--------------|----------------------------------------------------|----|
| MCFG[15]     | 初始值：0x0，1：复位 MDIO 模块                               | RW |
| MCFG[14: 10] | 初始值：0x0，MAC 模式读写结束之后持续等待 MDC 时钟周期                  | RW |
| MCFG[9]      | 初始值：0x1，MDIO 低速模式，CLKS 寄存器大于 0 时，可以配置此寄存器          | RW |
| MCFG[8]      | 初始值：0x0，<br>0：MDIO PHY 模式<br>1：MDIO MAC 模式         | RW |
| MCFG[4: 2]   | 初始值：0x5，MDC 时钟分频设置， $MDC = PCLK2 / (2^{(CLKS+1)})$ | RW |
| MCFG[1]      | 初始值：0x0，1：关闭 32bit 的 preamble 域                    | RW |
| MCFG[0]      | 初始值：0x0，1：轮询读取模式下，FIAD 寄存器自动递增                     | RW |

MCMD[31: 0]寄存器：

| 位       | 功能               | 类型 |
|---------|------------------|----|
| MCMD[1] | 初始值：0x0，1：轮询读取模式 | RW |
| MCMD[0] | 初始值：0x0，1：单次读取   | RW |

MADR[31: 0]寄存器：

| 位           | 功能                          | 类型 |
|-------------|-----------------------------|----|
| MADR[12: 8] | 初始值：0x0，5bit PHY address 参数 | RW |
| MADR[4: 0]  | 初始值：0x0，5bit REG address 参数 | RW |

MWTD[31: 0]寄存器:

| 位           | 功能                           | 类型 |
|-------------|------------------------------|----|
| MWTD[15: 0] | 初始值: 0x0, 写入到 PHY 的 16bit 数据 | RW |

MRDD[31: 0]寄存器:

| 位           | 功能                            | 类型 |
|-------------|-------------------------------|----|
| MRDD[15: 0] | 初始值: 0x0, 从 PHY 读取到的 16bit 数据 | RW |

MIND[31: 0]寄存器:

| 位       | 功能                                    | 类型 |
|---------|---------------------------------------|----|
| MIND[3] | 初始值: 0x0, 1: MDIO MAC 模式连接失败          | RW |
| MIND[2] | 初始值: 0x0, 1: MDIO MAC 读取未完成, 读取数据暂时无效 | RW |
| MIND[1] | 初始值: 0x0, 1: SCAN 模式                  | RW |
| MIND[0] | 初始值: 0x0, 1: MDIO MAC 状态机忙状态          | RW |

PHYID[31: 0]寄存器:

| 位           | 功能                           | 类型 |
|-------------|------------------------------|----|
| PHYID[5]    | 初始值: 0x0, 测试模式               | RW |
| PHYID[4: 0] | 初始值: 0x0, 5bit 的 PHY address | RW |

PHYSTA[31: 0]寄存器:

| 位             | 功能                  | 类型 |
|---------------|---------------------|----|
| PHYSTA[15: 0] | 初始值: 0xfe00, PHY 状态 | RW |

MGDI[31: 0]寄存器:

| 位           | 功能                                             | 类型 |
|-------------|------------------------------------------------|----|
| MGDI[15: 0] | 初始值: 0x0, MAC 写入到 PHY REG address 0x10 的数据, 读清 | RW |

MGDO[31: 0]寄存器:

| 位           | 功能                                         | 类型 |
|-------------|--------------------------------------------|----|
| MGDO[15: 0] | 初始值: 0x0, PHY REG address 0x14 输出到 MAC 的数据 | RW |



## 16.1 MAC 模式编程实例

函数一：初始化

```
mdio_mac_mode_init( ) {  
    `SET_MDIO_CLKS(wdata);    //set mdio clock frequency  
    `SET_MDIO_MDIO_MAC(1);    //set '1' MAC mode.  
}
```

函数二：MAC READ

```
mdio_mac_read ( ) {  
    `SET_MDIO_FIAD(phy_addr);    // set 5-bits PHY Address to be used by MDIO MAC  
    `SET_MDIO_RGAD(reg_addr);    // 5-bits Reg Address to be used by MDIO MAC  
    `SET_MDIO_RSTAT(1);    // 1: cause the MDIO MAC to perform a single read cycles.  
    While ( `GET_MDIO_BUSY(rdata) );    // wait mdio bus is not busy  
    `GET_MDIO_PRSD(rdata);    //get phy 16-bit rdata  
    `SET_MDIO_RSTAT(0);    // 0: disable the MDIO MAC to perform a single read cycles.  
}
```

函数三：MAC WRITE

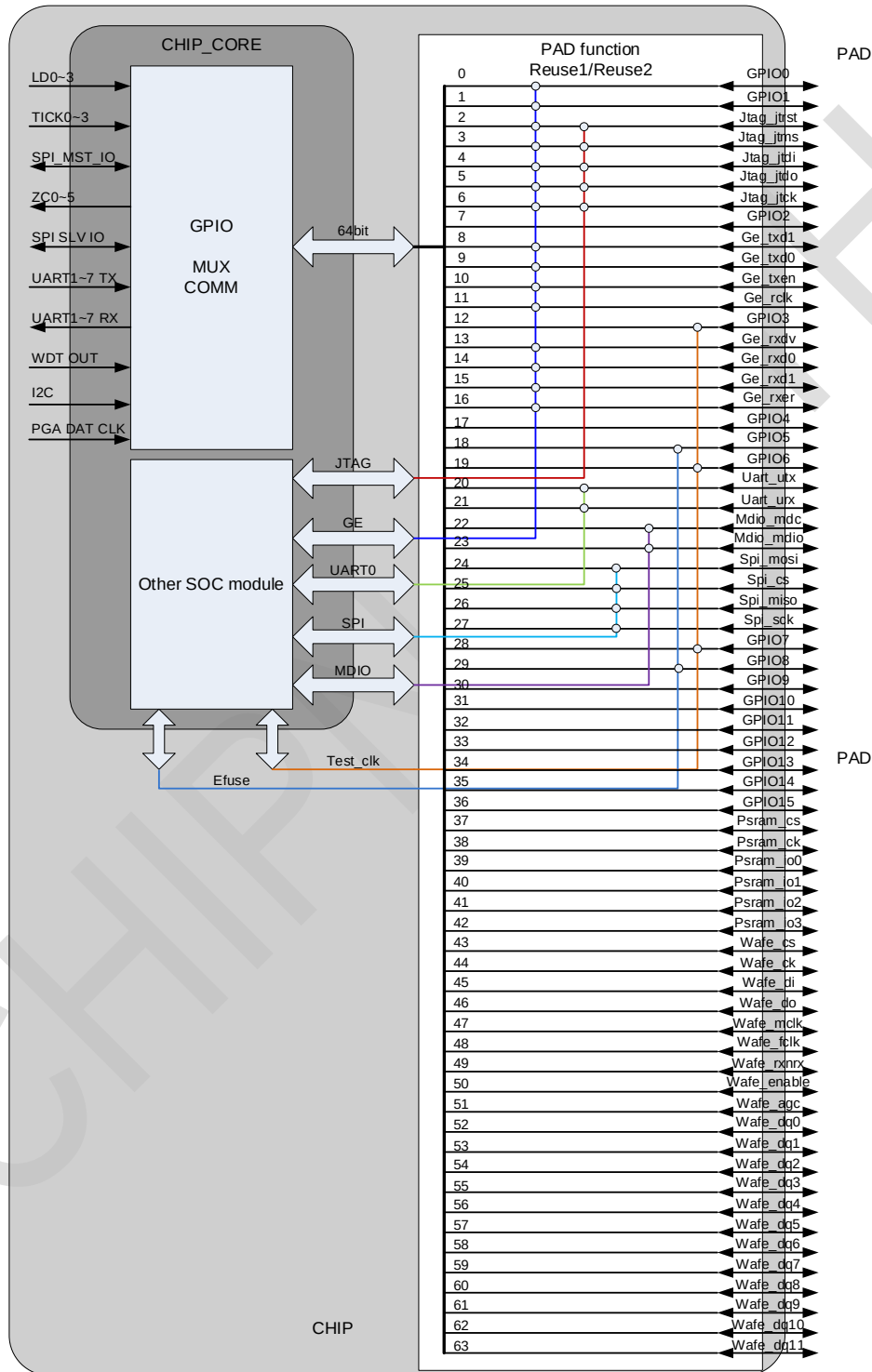
```
mdio_mac_write ( ) {  
    `SET_MDIO_FIAD(phy_addr);    // set 5-bits PHY Address to be used by MDIO MAC  
    `SET_MDIO_RGAD(reg_addr);    // 5-bits Reg Address to be used by MDIO MAC  
    `SET_MDIO_CTLD(wdata);    //16 bits MDIO Write Data, send to PHY  
    While ( `GET_MDIO_BUSY(rdata) );    //wait mdio is not busy, data has transmit done  
    Return();  
}
```

## 17 GPIO通用输入输出

芯片内部供 64 个 GPIO，全部的 GPIO 支持功能复用，支持外部中断。

### 17.1 GPIO PAD 复用

芯片 GPIO PAD 采用两级复用设计，具体复用图如下：



GPIO PAD 第一级复用集成在 GPIO 模块中，上图中 GPIO 模块的左侧信号（LD0~1、TICK0~3、SPI\_MST\_IO 等等）为第一级复用信号，这些信号可以映射到任意的 GPIO 管脚，以替换 GPIO 输入输出功能。

第二级复用在 PAD function 模块中，上图中的 Other SOC module 所连接的信号（JTAG、GE、UART0、SPI、MDIO、TEST\_CLK、EFUSE）为第二级复用信号，这些信号映射到固定的 GPIO PAD 端口上，软件不能修改管脚位置，只能打开或者关闭映射关系，一旦使能第二级映射，对应管脚的 GPIO 功能或第一级映射之后的功能均失效。

二级复用中 JTAG 端口存在两次复用，可用作 GE 的 MII 功能的部分端口或者 JTAG 端口，如果 GPIO\_JTAG\_EN 和 GPIO\_GE\_MII\_EN 功能同时打开，JTAG 端口依然用作为 JTAG 端口。即越靠近 PAD 端口的复用优先级越高。

理论上如果将所有的复用关闭，那么 64 个 GPIO PAD 均为普通的 GPIO 端口。

GPIO PAD 复用配置寄存器如下：

| 地址         | 寄存器名称                 |
|------------|-----------------------|
| 0xF2000090 | GPIO_SHARE_EN0[31: 0] |
| 0xF2000094 | GPIO_SHARE_EN1[31: 0] |
| 0xF2000098 | GPIO_SHARE_EN2[31: 0] |
| 0xF000009C | GPIO_SHARE_EN3[31: 0] |
| 0xF00000A0 | GPIO_SAMP_0[31: 0]    |
| 0xF00000A4 | GPIO_SAMP_1[31: 0]    |
| 0xF20000A8 | GPIO_SAMP_2[31: 0]    |
| 0xF20000AC | GPIO_SAMP_3[31: 0]    |
| 0xF20000B0 | GPIO_SAMP_4[31: 0]    |
| 0xF20000B4 | GPIO_SAMP_5[31: 0]    |
| 0xF20000B8 | GPIO_SAMP_6[31: 0]    |
| 0xF20000BC | GPIO_SAMP_7[31: 0]    |
| 0xF20000C0 | GPIO_SAMP_8[31: 0]    |
| 0xF20000C4 | GPIO_SAMP_9[31: 0]    |
| 0xF20000C8 | GPIO_SAMP_10[31: 0]   |
| 0xF20000CC | GPIO_SAMP_11[31: 0]   |
| 0xF20000D0 | GPIO_SAMP_12[31: 0]   |
| 0xF20000D4 | GPIO_SAMP_13[31: 0]   |
| 0xF20000D8 | GPIO_SAMP_14[31: 0]   |
| 0xF20000DC | GPIO_SAMP_15[31: 0]   |
| 0xF20000E0 | GPIO_SAMP_16[31: 0]   |
| 0xF20000E4 | GPIO_SAMP_17[31: 0]   |
| 0xF20000E8 | GPIO_SAMP_18[31: 0]   |

## GPIO\_SHARE\_EN0[31: 0]寄存器:

| 位                      | 功能                                                | 类型 |
|------------------------|---------------------------------------------------|----|
| [28]: GPIO_SPI_SLV1_EN | 初始值: 0x0, SPI_SLAVE1 接口复用使能                       | RW |
| [27]: GPIO_GE_MII_EN   | 初始值: 0x0, 复用 GPIO0、GPIO1、JTAG、GE 端口为 MII、RGMII 接口 | RW |
| [26]: GPIO_GE_RGMII_EN | 初始值: 0x0, 复用 GPIO0、GPIO1、JTAG、GE 端口为 MII、RGMII 接口 | RW |
| [25]: GPIO_ZC5_EN      | 初始值: 0x0, 过零信号复用使能                                | RW |
| [24]: GPIO_ZC4_EN      |                                                   |    |
| [23]: GPIO_ZC3_EN      |                                                   |    |
| [22]: GPIO_ZC2_EN      |                                                   |    |
| [21]: GPIO_ZC1_EN      |                                                   |    |
| [20]: GPIO_ZC0_EN      |                                                   |    |
| [19]: GPIO_WDT_EN      | 初始值: 0x0, WDT GPIO 输出 IO 复用使能                     | RW |
| [18]: GPIO_MDIO_EN     | 初始值: 0x0, MDIO GPIO 复用使能                          | RW |
| [17]: GPIO_JTAG_EN     | 初始值: 0x0, JTAG GPIO 复用使能                          | RW |
| [16]: GPIO_GE_EN       | 初始值: 0x0, GE 端口复用使能, 默认支持 RMII 模式                 | RW |
| [15]: GPIO_TICK3_EN    | 初始值: 0x0, TICK IO output 信号复用使能                   | RW |
| [14]: GPIO_TICK2_EN    |                                                   |    |
| [13]: GPIO_TICK1_EN    |                                                   |    |
| [12]: GPIO_TICK0_EN    |                                                   |    |
| [11]: GPIO_LD3_EN      | 初始值: 0x0, 模拟接口 LD 信号复用使能                          | RW |
| [10]: GPIO_LD2_EN      |                                                   |    |
| [9]: GPIO_LD1_EN       |                                                   |    |
| [8]: GPIO_LD0_EN       |                                                   |    |
| [7]: GPIO_SPI_SLV_EN   | 初始值: 0x0, SPI_SLAVE 接口复用使能                        | RW |
| [6]: GPIO_UART2_EN     | 初始值: 0x0, UART 端口复用使能                             | RW |
| [5]: GPIO_UART1_EN     |                                                   |    |
| [4]: GPIO_UART0_EN     |                                                   |    |

GPIO\_SHARE\_EN1[31: 0]寄存器：

| 位                       | 功能                                                                               | 类型 |
|-------------------------|----------------------------------------------------------------------------------|----|
| [12: 11]: GPIO_2ND_BOOT | 初始值：0x1，Bit0 0：由 bit1 控制启动<br>1：由 gpio0 控制启动<br>Bit1 0：从 spi 启动<br>1：由 gpio 控制启动 | RW |
| [10]: GPIO_EFUSE_EN     | 初始值：0x0，EFUSE 信号复用使能                                                             | RW |
| [9]: GPIO_TCK_W_DAC_EN  | 初始值：0x0，GPIO3 输出 DAC 时钟使能                                                        | RW |
| [8]: GPIO_TCK_W_ADC_EN  | 初始值：0x0，GPIO3 输出 ADC 时钟使能                                                        | RW |
| [5]: GPIO_TCK_DAC_EN    | 初始值：0x0，GPIO3 输出 DAC 时钟使能                                                        | RW |
| [4]: GPIO_TCK_ADC_EN    | 初始值：0x0，GPIO3 输出 ADC 时钟使能                                                        | RW |
| [3]: GPIO_TCK_OSC_EN    | 初始值：0x0，GPIO3 输出 OSC 时钟使能                                                        | RW |
| [2]: GPIO_TCK_GE_EN     | 初始值：0x0，GPIO7 输出 GE_PLL 时钟使能                                                     | RW |
| [1]: GPIO_TCK_SYS_EN    | 初始值：0x0，GPIO6 输出 SYS_PLL 时钟使能                                                    | RW |
| [0]: GPIO_TCK_MC_SYN_EN | 初始值：0x0，GPIO3 输出 MC SYN 时钟使能                                                     | RW |

GPIO\_SHARE\_EN2[31: 0]寄存器：

| 位                      | 功能                      | 类型 |
|------------------------|-------------------------|----|
| [16]: GPIO_SPI5_CS1_EN | 初始值：0x0，SPI5 CS 信号复用使能  | RW |
| [15]: GPIO_SPI5_CS0_EN | 初始值：0x0，SPI5 CS 信号复用使能  | RW |
| [14]: GPIO_SPI4_CS1_EN | 初始值：0x0，SPI4 CS 信号复用使能  | RW |
| [13]: GPIO_SPI4_CS0_EN | 初始值：0x0，SPI4 CS 信号复用使能  | RW |
| [12]: GPIO_UART7_EN    | 初始值：0x0，UART7 端口复用使能    | RW |
| [11]: GPIO_UART6_EN    | 初始值：0x0，UART6 端口复用使能    | RW |
| [10]: GPIO_UART5_EN    | 初始值：0x0，UART5 端口复用使能    | RW |
| [9]: GPIO_SPI1_QUAL_EN | 初始值：0x0，SPI1 qual 使能    | RW |
| [8]: GPIO_ANT_CTR1_EN  | 初始值：0x0，ANT_CTR1 信号复用使能 | RW |
| [7]: GPIO_ANT_CTR0_EN  | 初始值：0x0，ANT_CTR0 信号复用使能 | RW |
| [6]: GPIO_SPI1_CS3_EN  | 初始值：0x0，SPI1 CS 信号复用使能  | RW |
| [5]: GPIO_SPI1_CS2_EN  |                         |    |
| [4]: GPIO_SPI1_CS1_EN  |                         |    |
| [3]: GPIO_SPI1_CS0_EN  |                         |    |
| [2]: GPIO_UART4_EN     | 初始值：0x0，UART4 端口复用使能    | RW |
| [1]: GPIO_UART3_EN     | 初始值：0x0，UART3 端口复用使能    | RW |
| [0]: GPIO_I2C_EN       | 初始值：0x0，I2C 端口复用使能      | RW |

GPIO\_SAMP\_0[31: 0]寄存器:

| 位                    | 功能                          | 类型 |
|----------------------|-----------------------------|----|
| [5: 0]: GPIO_WDT_MAP | 初始值: 0xe, WDT GPIO 输出复用管脚配置 | RW |

GPIO\_SAMP\_1[31: 0]寄存器:

| 位                       | 功能                            | 类型 |
|-------------------------|-------------------------------|----|
| [29: 24]: GPIO_URX2_MAP | 初始值: 0x15, UART1、UART2 复用管脚配置 | RW |
| [21: 16]: GPIO_UTX2_MAP | 初始值: 0x14, UART1、UART2 复用管脚配置 | RW |
| [13: 8]: GPIO_URX1_MAP  | 初始值: 0x15, UART1、UART2 复用管脚配置 | RW |
| [5: 0]: GPIO_UTX1_MAP   | 初始值: 0x14, UART1、UART2 复用管脚配置 | RW |

GPIO\_SAMP\_2[31: 0]寄存器:

| 位                      | 功能                   | 类型 |
|------------------------|----------------------|----|
| [29: 25]: GPIO_ZC5_MAP | 初始值: 0xC, 过零信号复用管脚配置 | RW |
| [24: 20]: GPIO_ZC4_MAP |                      |    |
| [19: 15]: GPIO_ZC3_MAP |                      |    |
| [14: 10]: GPIO_ZC2_MAP |                      |    |
| [9: 5]: GPIO_ZC1_MAP   |                      |    |
| [4: 0]: GPIO_ZC0_MAP   |                      |    |

GPIO\_SAMP\_3[31: 0]寄存器:

| 位                     | 功能                    | 类型 |
|-----------------------|-----------------------|----|
| [13: 8]: GPIO_ZC5_MAP | 初始值: 0x3f, 过零信号复用管脚配置 | RW |
| [5: 0]: GPIO_ZC4_MAP  |                       |    |

GPIO\_SAMP\_4[31: 0]寄存器:

| 位                      | 功能                       | 类型 |
|------------------------|--------------------------|----|
| [29: 24]: GPIO_LD3_MAP | 初始值: 0x3f, LD 控制信号复用管脚配置 | RW |
| [21: 16]: GPIO_LD2_MAP |                          |    |
| [13: 8]: GPIO_LD1_MAP  |                          |    |
| [5: 0]: GPIO_LD0_MAP   |                          |    |

GPIO\_SAMP\_5[31: 0]寄存器:

| 位                        | 功能                                 | 类型 |
|--------------------------|------------------------------------|----|
| [29: 24]: GPIO_TICK3_MAP | 初始值: 0x3f, TICK IO output 信号复用管脚配置 | RW |
| [21: 16]: GPIO_TICK2_MAP |                                    |    |
| [13: 8]: GPIO_TICK1_MAP  |                                    |    |
| [5: 0]: GPIO_TICK0_MAP   |                                    |    |

GPIO\_SAMP\_6[31: 0]寄存器:

| 位                               | 功能                              | 类型 |
|---------------------------------|---------------------------------|----|
| [29: 24]: GPIO_SPI_SLV_MISO_MAP | 初始值: 0x3f, SPI SLAVE 相关信号复用管脚配置 | RW |
| [21: 16]: GPIO_SPI_SLV_MOSI_MAP |                                 |    |
| [13: 8]: GPIO_SPI_SLV_CS_MAP    |                                 |    |
| [5: 0]: GPIO_SPI_SLV_SCK_MAP    |                                 |    |

GPIO\_SAMP\_7[31: 0]寄存器:

| 位                                | 功能                               | 类型 |
|----------------------------------|----------------------------------|----|
| [29: 24]: GPIO_SPI_SLV1_MISO_MAP | 初始值: 0x3f, SPI SLAVE1 相关信号复用管脚配置 | RW |
| [21: 16]: GPIO_SPI_SLV1_MOSI_MAP |                                  |    |
| [13: 8]: GPIO_SPI_SLV1_CS_MAP    |                                  |    |
| [5: 0]: GPIO_SPI_SLV1_SCK_MAP    |                                  |    |

GPIO\_SAMP\_8[31: 0]寄存器:

| 位                         | 功能                        | 类型 |
|---------------------------|---------------------------|----|
| [13: 8]: GPIO_I2C_SDA_MAP | 初始值: 0x3f, I2C 相关信号复用管脚配置 | RW |
| [5: 0]: GPIO_I2C_SCL_MAP  |                           |    |

GPIO\_SAMP\_9[31: 0]寄存器:

| 位                      | 功能                          | 类型 |
|------------------------|-----------------------------|----|
| [13: 8]: GPIO_URX3_MAP | 初始值: 0x3f, UART3 相关信号复用管脚配置 | RW |
| [5: 0]: GPIO_UTX3_MAP  |                             |    |

GPIO\_SAMP\_10[31: 0]寄存器:

| 位                      | 功能                          | 类型 |
|------------------------|-----------------------------|----|
| [13: 8]: GPIO_URX4_MAP | 初始值: 0x3f, UART4 相关信号复用管脚配置 | RW |
| [5: 0]: GPIO_UTX4_MAP  |                             |    |

GPIO\_SAMP\_11[31: 0]寄存器:

| 位                           | 功能                         | 类型 |
|-----------------------------|----------------------------|----|
| [29: 24]: GPIO_SPI1_CS3_MAP | 初始值: 0x3f, SPI1 片选信号复用管脚配置 | RW |
| [21: 16]: GPIO_SPI1_CS2_MAP |                            |    |
| [13: 8]: GPIO_SPI1_CS1_MAP  |                            |    |
| [5: 0]: GPIO_SPI1_CS0_MAP   |                            |    |

GPIO\_SAMP\_12[31: 0]寄存器:

| 位                            | 功能                         | 类型 |
|------------------------------|----------------------------|----|
| [20: 16]: GPIO_SPI1_MISO_MAP | 初始值: 0x3f, SPI1 相关信号复用管脚配置 | RW |
| [12: 8]: GPIO_SPI1_MOSI_MAP  |                            |    |
| [4: 0]: GPIO_SPI1_SCK_MAP    |                            |    |

GPIO\_SAMP\_13[31: 0]寄存器:

| 位                          | 功能                            | 类型 |
|----------------------------|-------------------------------|----|
| [13: 8]: GPIO_ANT_CTR0_MAP | 初始值: 0x3f, ANT CTR 相关信号复用管脚配置 | RW |
| [5: 0]: GPIO_ANT_CTR0_MAP  |                               |    |

GPIO\_SAMP\_14[31: 0]寄存器:

| 位                      | 功能                          | 类型 |
|------------------------|-----------------------------|----|
| [13: 8]: GPIO_URX5_MAP | 初始值: 0x3f, UART5 相关信号复用管脚配置 | RW |
| [5: 0]: GPIO_UTX5_MAP  |                             |    |

GPIO\_SAMP\_15[31: 0]寄存器:

| 位                      | 功能                          | 类型 |
|------------------------|-----------------------------|----|
| [13: 8]: GPIO_URX6_MAP | 初始值: 0x3f, UART6 相关信号复用管脚配置 | RW |
| [5: 0]: GPIO_UTX6_MAP  |                             |    |

GPIO\_SAMP\_16[31: 0]寄存器:

| 位                      | 功能                          | 类型 |
|------------------------|-----------------------------|----|
| [13: 8]: GPIO_URX7_MAP | 初始值: 0x3f, UART7 相关信号复用管脚配置 | RW |
| [5: 0]: GPIO_UTX7_MAP  |                             |    |

GPIO\_SAMP\_17[31: 0]寄存器:

| 位                            | 功能                         | 类型 |
|------------------------------|----------------------------|----|
| [29: 24]: GPIO_SPI4_MISO_MAP | 初始值: 0x3f, SPI4 相关信号复用管脚配置 | RW |
| [23: 18]: GPIO_SPI4_MOSI_MAP |                            |    |
| [17: 12]: GPIO_SPI4_SCK_MAP  |                            |    |
| [11: 6]: GPIO_SPI4_CS1_MAP   |                            |    |
| [5: 0]: GPIO_SPI4_CS0_MAP    |                            |    |

GPIO\_SAMP\_18[31: 0]寄存器:

| 位                            | 功能                         | 类型 |
|------------------------------|----------------------------|----|
| [29: 24]: GPIO_SPI5_MISO_MAP | 初始值: 0x3f, SPI5 相关信号复用管脚配置 | RW |
| [23: 18]: GPIO_SPI5_MOSI_MAP |                            |    |
| [17: 12]: GPIO_SPI5_SCK_MAP  |                            |    |
| [11: 6]: GPIO_SPI5_CS1_MAP   |                            |    |
| [5: 0]: GPIO_SPI5_CS0_MAP    |                            |    |



### 17.1.1 GPIO PAD 复用实例

实例一：将 JTAG 端口用作通用 GPIO 端口

关闭 GPIO\_GE\_MII\_EN。

关闭 GPIO\_GE\_RGMII\_EN。

关闭 GPIO\_JTAG\_EN。

解除所有一级复用对 JTAG 端口的复用。

按 GPIO 规则使用 JTAG 端口。

实例二：将 JTAG 端口复用为 SPI\_SLAVE 端口

关闭 GPIO\_GE\_MII\_EN

关闭 GPIO\_GE\_RGMII\_EN

关闭 GPIO\_JTAG\_EN

解除所有一级复用对 JTAG（GPIO MAP 3、4、5、6）端口的复用

配置 GPIO\_SSI\_SCK\_MAP 为 3，使用 JTMS 作为 SCK 信号

配置 GPIO\_SSI\_CS\_MAP 为 4，使用 JTDI 作为 CS 信号

配置 GPIO\_SSI\_MISO\_MAP 为 5，使用 JTDO 作为 MISO 信号

配置 GPIO\_SSI\_MOSI\_MAP 为 6，使用 JTCK 作为 MOSI 信号

配置 GPIO\_SSI\_EN 为 1，使能 SPI\_SLAVE 端口信号复用

使用 SPI\_SLAVE 器件

实例三：使用 GPIO 输出 SYS\_PLL 的 50MHz 分频信号

配置 GPIO\_TCK\_SYS\_EN 信号为 1，打开 SYS\_PLL 分频输出

GPIO6 输出 50MHz 信号

注：GPIO\_TCK\_SYS\_EN 配置后会屏蔽 GPIO6 上的其他复用。实际使用中，为了安全考虑还是建议关闭 GPIO6 的其他复用。

## 17.2 GPIO 输入输出功能

当 GPIO 没有被复位为特定的功能时，GPIO 就是一个通用的输入输出端口。软件可以通过寄存器将 GPIO 端口配置为不同的状态，GPIO 输入输出功能相关寄存器如下：

| 地址         | 寄存器名称                  |
|------------|------------------------|
| 0xF2000000 | GPIO_DIN[31: 0]        |
| 0xF2000004 | GPIO_DIN[31: 0]        |
| 0xF2000008 | GPIO_DOUT[31: 0]       |
| 0xF200000C | GPIO_DOUT1[31: 0]      |
| 0xF2000010 | GPIO_CTRL[31: 0]       |
| 0xF2000014 | GPIO_CTRL1[31: 0]      |
| 0xF2000018 | GPIO_IN_EN[31: 0]      |
| 0xF200001C | GPIO_IN_EN1[31: 0]     |
| 0xF2000020 | GPIO_PD_EN[31: 0]      |
| 0xF2000024 | GPIO_PD_EN1[31: 0]     |
| 0xF2000028 | GPIO_PU_EN[31: 0]      |
| 0xF200002C | GPIO_PU_EN1[31: 0]     |
| 0xF2000030 | GPIO_SMIT_EN[31: 0]    |
| 0xF2000034 | GPIO_SMIT_EN1[31: 0]   |
| 0xF2000038 | GPIO_PAD_DRV_0_[31: 0] |
| 0xF200003C | GPIO_PAD_DRV_1_[31: 0] |
| 0xF2000040 | GPIO_PAD_DRV_2_[31: 0] |
| 0xF2000044 | GPIO_PAD_DRV_3_[31: 0] |

GPIO\_DIN[31: 0]寄存器：

| 位                 | 功能                        | 类型 |
|-------------------|---------------------------|----|
| [31: 0]: GPIO_DIN | 初始值: N/A, GPIO 输入值[31: 0] | RO |

GPIO\_DIN1[31: 0]寄存器：

| 位                  | 功能                         | 类型 |
|--------------------|----------------------------|----|
| [31: 0]: GPIO_DIN1 | 初始值: N/A, GPIO 输入值[63: 32] | RO |

GPIO\_DOUT[31: 0]寄存器：

| 位                  | 功能                          | 类型 |
|--------------------|-----------------------------|----|
| [31: 0]: GPIO_DOUT | 初始值: 0x0, GPIO 输出值配置[31: 0] | RW |

GPIO\_DOUT1[31: 0]寄存器：

| 位                   | 功能                           | 类型 |
|---------------------|------------------------------|----|
| [31: 0]: GPIO_DOUT1 | 初始值: 0x0, GPIO 输出值配置[63: 32] | RW |

GPIO\_CTRL[31: 0]寄存器：

| 位                  | 功能                                                  | 类型 |
|--------------------|-----------------------------------------------------|----|
| [31: 0]: GPIO_CTRL | 初始值: 0xffff ffff, GPIO[31: 0]方向配置<br>1: 输入<br>0: 输出 | RW |

GPIO\_CTRL1[31: 0]寄存器：

| 位                   | 功能                                                   | 类型 |
|---------------------|------------------------------------------------------|----|
| [31: 0]: GPIO_CTRL1 | 初始值: 0xffff ffff, GPIO[63: 32]方向配置<br>1: 输入<br>0: 输出 | RW |

GPIO\_IN\_EN[31: 0]寄存器：

| 位                   | 功能                                      | 类型 |
|---------------------|-----------------------------------------|----|
| [31: 0]: GPIO_IN_EN | 初始值: 0x0, GPIO[31: 0]强制输入使能, 默认情况下不需要配置 | RW |

GPIO\_IN\_EN1[31: 0]寄存器：

| 位                    | 功能                                       | 类型 |
|----------------------|------------------------------------------|----|
| [31: 0]: GPIO_IN_EN1 | 初始值: 0x0, GPIO[63: 32]强制输入使能, 默认情况下不需要配置 | RW |

GPIO\_PD\_EN[31: 0]寄存器：

| 位                   | 功能                                    | 类型 |
|---------------------|---------------------------------------|----|
| [31: 0]: GPIO_PD_EN | 初始值: 0x5, GPIO[31: 0]下拉功能使能, 下拉电 44KΩ | RW |

GPIO\_PD\_EN1[31: 0]寄存器：

| 位                    | 功能                                     | 类型 |
|----------------------|----------------------------------------|----|
| [31: 0]: GPIO_PD_EN1 | 初始值: 0x5, GPIO[63: 32]下拉功能使能, 下拉电 44KΩ | RW |

GPIO\_PU\_EN[31: 0]寄存器：

| 位                   | 功能                                             | 类型 |
|---------------------|------------------------------------------------|----|
| [31: 0]: GPIO_PU_EN | 初始值: 0x0f00 0000, GPIO[31: 0]上拉功能使能, 上拉电阻 44KΩ | RW |

GPIO\_PU\_EN1[31: 0]寄存器：

| 位                    | 功能                                              | 类型 |
|----------------------|-------------------------------------------------|----|
| [31: 0]: GPIO_PU_EN1 | 初始值: 0x0f00 0000, GPIO[63: 32]上拉功能使能, 上拉电阻 44KΩ | RW |

GPIO\_SMIT\_EN[31: 0]寄存器：

| 位                     | 功能                                    | 类型 |
|-----------------------|---------------------------------------|----|
| [31: 0]: GPIO_SMIT_EN | 初始值: 0x0, GPIO[31: 0]上拉功能使能, 上拉电 44KΩ | RW |

GPIO\_SMIT\_EN1[31: 0]寄存器：

| 位                      | 功能                                     | 类型 |
|------------------------|----------------------------------------|----|
| [31: 0]: GPIO_SMIT_EN1 | 初始值: 0x0, GPIO[63: 32]上拉功能使能, 上拉电 44KΩ | RW |

GPIO\_PAD\_DRV\_0 [31: 0]寄存器：

| 位                    | 功能                                                                                               | 类型 |
|----------------------|--------------------------------------------------------------------------------------------------|----|
| [31: 0]: GPIO_DRV_0_ | 初始值：0xffffffff，GPIO 0~15 输出强度调整，每个 pad 占用两个 bit：<br>0: 2.4mA<br>1: 4.8mA<br>2: 7.2mA<br>3: 9.6mA | RW |

GPIO\_PAD\_DRV\_1 [31: 0]寄存器：

| 位               | 功能                                                                                                | 类型 |
|-----------------|---------------------------------------------------------------------------------------------------|----|
| GPIO_PAD_DRV_1_ | 初始值：0xffffffff，GPIO 16~31 输出强度调整，每个 pad 占用两个 bit：<br>0: 2.4mA<br>1: 4.8mA<br>2: 7.2mA<br>3: 9.6mA | RW |

GPIO\_PAD\_DRV\_2 [31: 0]寄存器：

| 位               | 功能                                                                                                | 类型 |
|-----------------|---------------------------------------------------------------------------------------------------|----|
| GPIO_PAD_DRV_2_ | 初始值：0xffffffff，GPIO 32~47 输出强度调整，每个 pad 占用两个 bit：<br>0: 2.4mA<br>1: 4.8mA<br>2: 7.2mA<br>3: 9.6mA | RW |

GPIO\_PAD\_DRV\_3 [31: 0]寄存器：

| 位               | 功能                                                                                                | 类型 |
|-----------------|---------------------------------------------------------------------------------------------------|----|
| GPIO_PAD_DRV_3_ | 初始值：0xffffffff，GPIO 48~63 输出强度调整，每个 pad 占用两个 bit：<br>0: 2.4mA<br>1: 4.8mA<br>2: 7.2mA<br>3: 9.6mA | RW |

### 17.2.1 GPIO 电气性能

此芯片内部所有 GPIO 为输出状态时，最大驱动能力为 9.6mA；GPIO 为输入状态时，低电平电压范围为-0.3V 到 0.8V，高电平电压范围为 2.0V 到 3.6V，芯片 GPIO 管脚最高支持的输入电压为 3.6V（特殊功能 GPIO 除外）。

### 17.2.2 特殊功能 GPIO

芯片的 JTAG 端口使用特殊功能 GPIO，此组端口均支持 5V 输入信号；另外 JTDI 和 JTMS 使用 open-drain 模式的 GPIO（这两个 GPIO 没有输出驱动能力）。

### 17.3 外部中断功能

GPIO 所有的端口均支持可屏蔽外部中断功能。支持边沿或电平触发。GPIO 外部中断相关寄存器如下表：

| 地址         | 寄存器名称                  |
|------------|------------------------|
| 0xF2000070 | GPIO_EINT_EN[31: 0]    |
| 0xF2000074 | GPIO_EINT_EN1[31: 0]   |
| 0xF2000078 | GPIO_EINT_NEG[31: 0]   |
| 0xF200007C | GPIO_EINT_NEG1[31: 0]  |
| 0xF2000080 | GPIO_EINT_EDGE[31: 0]  |
| 0xF2000084 | GPIO_EINT_EDGE1[31: 0] |
| 0xF2000088 | GPIO_EINT_STA[31: 0]   |
| 0xF200008C | GPIO_EINT_STA1[31: 0]  |

GPIO\_EINT\_EN[31: 0]寄存器：

| 位                     | 功能                      | 类型 |
|-----------------------|-------------------------|----|
| [31: 0]: GPIO_EINT_EN | 初始值：0x0，GPIO[31: 0]中断使能 | RW |

GPIO\_EINT\_EN1[31: 0]寄存器：

| 位                      | 功能                       | 类型 |
|------------------------|--------------------------|----|
| [31: 0]: GPIO_EINT_EN1 | 初始值：0x0，GPIO[63: 32]中断使能 | RW |

GPIO\_EINT\_NEG[31: 0]寄存器：

| 位                      | 功能                                            | 类型 |
|------------------------|-----------------------------------------------|----|
| [31: 0]: GPIO_EINT_NEG | 初始值：0x0，GPIO[31: 0]中断触发电平选择<br>0：高电平<br>1：低电平 | RW |

GPIO\_EINT\_NEG1[31: 0]寄存器：

| 位                       | 功能                                             | 类型 |
|-------------------------|------------------------------------------------|----|
| [31: 0]: GPIO_EINT_NEG1 | 初始值：0x0，GPIO[63: 32]中断触发电平选择<br>0：高电平<br>1：低电平 | RW |

GPIO\_EINT\_EDGE[31: 0]寄存器：

| 位                       | 功能                                                | 类型 |
|-------------------------|---------------------------------------------------|----|
| [31: 0]: GPIO_EINT_EDGE | 初始值：0x0，GPIO[31: 0]中断触发边沿触发使能<br>1：边沿触发<br>0：电平触发 | RW |

GPIO\_EINT\_EDGE1[31: 0]寄存器：

| 位                        | 功能                                                 | 类型 |
|--------------------------|----------------------------------------------------|----|
| [31: 0]: GPIO_EINT_EDGE1 | 初始值：0x0，GPIO[63: 32]中断触发边沿触发使能<br>1：边沿触发<br>0：电平触发 | RW |

GPIO\_EINT\_STA[31: 0]寄存器：

| 位                      | 功能                         | 类型 |
|------------------------|----------------------------|----|
| [31: 0]: GPIO_EINT_STA | 初始值：0x0，GPIO[31: 0]中断状态寄存器 | RC |

GPIO\_EINT\_STA1[31: 0]寄存器:

| 位                       | 功能                            | 类型 |
|-------------------------|-------------------------------|----|
| [31: 0]: GPIO_EINT_STA1 | 初始值: 0x0, GPIO[63: 32]中断状态寄存器 | RC |

实例一: GPIO0 端口上升沿触发

解除 GPIO0 复用

配置 GPIO\_EINT\_NEG[0]为 1

配置 GPIO\_EINT\_EDGE[0]为 1

配置 GPIO\_EINT\_EN[0]为 1

实例二: JTDI 端口下降沿触发

解除 JTDI 端口复用

配置 GPIO\_EINT\_NEG[4]为 0

配置 GPIO\_EINT\_EDGE[4]为 1

配置 GPIO\_EINT\_EN[4]为 1

## 18 EFUSE一次性写入单元

芯片内部使用 EFUSE 实现了一次性写入存储器模块。主要用于一次写入 CHIP ID 等不可修改的关键信息。EFUSE 主要支持如下功能：

|   |                                   |
|---|-----------------------------------|
| 1 | 按 SMIC 提供的烧写时序，实现 efuse 烧写功能      |
| 2 | 按 SMIC 提供的读取时序，实现 efuse 读取功能      |
| 3 | 提供 EFUSE VDD 使能控制功能               |
| 4 | 提供时序控制逻辑，默认配置是按 12.5M 的 PCLK 计算而来 |
| 5 | 共 128bit 数据位                      |

EFUSE 功能寄存器如下：

| 地址         | 寄存器名称               |
|------------|---------------------|
| 0xF2D00000 | EFUSE_CTRL[31: 0]   |
| 0xF2D00008 | CTRL_STATUS[31: 0]  |
| 0xF2D00100 | PGM_DATA_0_[31: 0]  |
| 0xF2D00104 | PGM_DATA_1_[31: 0]  |
| 0xF2D00108 | PGM_DATA_2_[31: 0]  |
| 0xF2D0010C | PGM_DATA_3_[31: 0]  |
| 0xF2D00200 | READ_DATA_0_[31: 0] |
| 0xF2D00204 | READ_DATA_1_[31: 0] |
| 0xF2D00208 | READ_DATA_2_[31: 0] |
| 0xF2D0020C | READ_DATA_3_[31: 0] |
| 0xF2D00010 | BURN_TIME_0[31: 0]  |
| 0xF2D00014 | BURN_TIME_1[31: 0]  |
| 0xF2D00020 | READ_TIME[31: 0]    |

EFUSE\_CTRL[31: 0]寄存器：

| 位                | 功能                                                           | 类型 |
|------------------|--------------------------------------------------------------|----|
| EFUSE_CTRL[8]    | 初始值：0x0，当读取命令结束后将读取到的数据与写入的数据进行比较功能使能                        | RW |
| [4]: EFUSE_LOOP  | 初始值：0x0，测试功能                                                 | RW |
| [1: 0]: RW_START | 初始值：0x0，读写功能完成后硬件清零<br>2: EFUSE 烧写<br>1: EFUSE 读取<br>0: 空闲状态 | RW |

## CTRL\_STATUS[31: 0]寄存器:

| 位                 | 功能               | 类型 |
|-------------------|------------------|----|
| CTRL_STATUS[6: 4] | 初始值: 0x0, 状态机状态  | RW |
| CTRL_STATUS[0]    | 初始值: 0x0, 数据比对成功 | RW |

## PGM\_DATA\_n [31: 0]寄存器:

| 位                 | 功能                       | 类型 |
|-------------------|--------------------------|----|
| PGM_DATA_n [1: 0] | 初始值: 0x0, 需要写入的数据, n=0~3 | RW |

## READ\_DATA\_n [31: 0]寄存器:

| 位                  | 功能                       | 类型 |
|--------------------|--------------------------|----|
| READ_DATA_n [1: 0] | 初始值: 0x0, 读取出来的数据, n=0~3 | RO |

## BURN\_TIME\_0[31: 0]寄存器:

| 位                   | 功能                         | 类型 |
|---------------------|----------------------------|----|
| BURN_TIME_0[31: 24] | 初始值: 0xd, 烧写信号低电平时间        | RW |
| BURN_TIME_0[23: 16] | 初始值: 0x33, 烧写信号高电平时间       | RW |
| BURN_TIME_0[15: 8]  | 初始值: 0x2, PGMEN 到 AEN 建立时间 | RW |
| BURN_TIME_0[7: 0]   | 初始值: 0x1, AEN 后 ADDR 保持时间  | RW |

## BURN\_TIME\_1[31: 0]寄存器:

| 位                         | 功能                                                        | 类型 |
|---------------------------|-----------------------------------------------------------|----|
| [31: 16]: BURN_SP_PG_AVDD | 初始值: 0x61a8, VDD 到 PGMEN 建立时间, 实际配置值较大在此时间内 2.5V 电压必须稳定下来 | RW |
| [15: 8]: BURN_HP_PG_AVDD  | 初始值: 0x0f, RDEN 信号到 AVDD 建立时间                             | RW |
| [7: 0]: BURN_SP_RD        | 初始值: 0x2, RDEN 信号到 AVDD 建立时间                              | RW |

## READ\_TIME[31: 0]寄存器:

| 位                    | 功能                        | 类型 |
|----------------------|---------------------------|----|
| [31: 24]: READ_HR_A  | 初始值: 0x1, AEN 到 ADDR 保持时间 | RW |
| [23: 16]: READ_SR_RD | 初始值: 0x2, RDEN 到 AEN 建立时间 | RW |
| [15: 8]: READ_RD_L   | 初始值: 0x3, 读取 AEN 低电平持续时间  | RW |
| [7: 0]: READ_RD_H    | 初始值: 0x4, 读取 AEN 高电平持续时间  | RW |

注: 软件不能随意修改此组寄存器的数值。



## 18.1 烧写功能外部硬件电路

外部电路连接图如下所示：

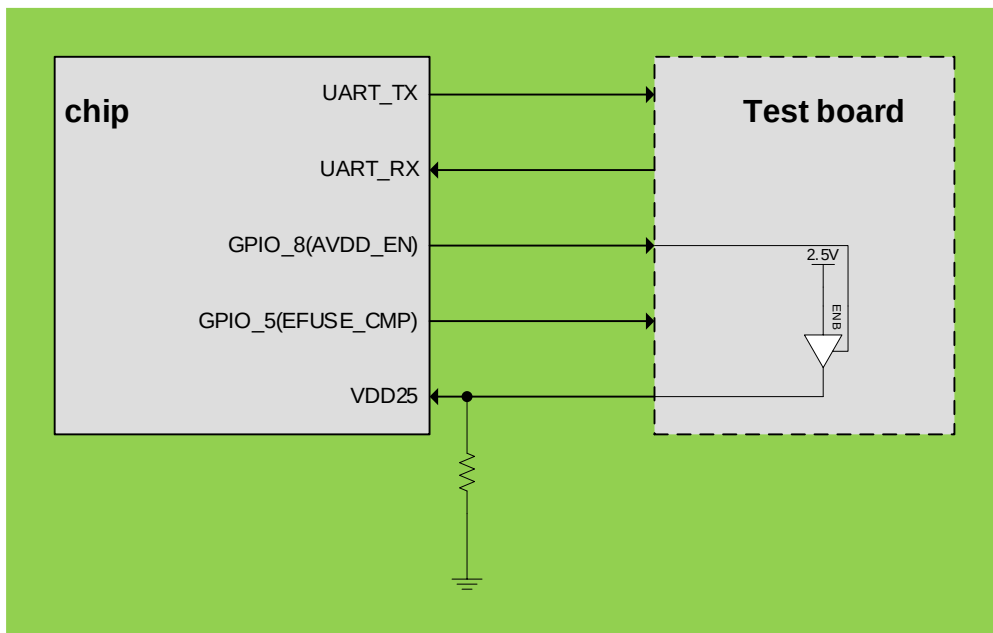


图 18-1 EFUSE 烧写控制电路

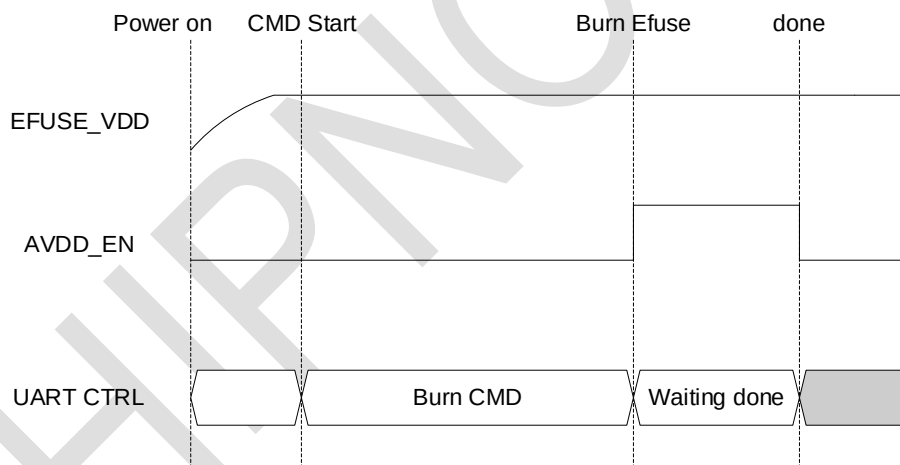


图 18-2 EFUSE 烧写控制时序示意

为了烧写 EFUSE，我们需要使用通信接口（UART），芯片会通过 GPIO8 给出 EFUSE\_AVDD(2.5V)上电使能信号，烧写完成后芯片内部会进行 EFUSE 的正确性校验，如果烧写正确的话会将 GPIO5 拉高用以指示芯片 EFUSE 烧写成功。接收到烧写命令后，一般会在 5ms 之内完成 EFUSE 烧写，烧写完成之后即可撤销 2.5V 的 EFUSE\_VDD。EFUSE\_VDD 仅用于 EFUSE 的烧写，读取 EFUSE 的时候无需提供 2.5V 的 EFUSE 供电。

## 18.2 UART DEBUG 模式写入

考虑到 FT 或模块成品测试的便利性，建议使用 UART DEBUG 功能直接烧写和读取 EFUSE。UART DEBUG 模式烧写与读取命令如下：

```
write 0xF2100010 0x1A660402
write 0xF2100014 0xC3501E04
write 0xF2100020 0x02040408
write 0xF2100054 0x400//open GPIO8 and GPIO5 for EFUSE ctrl
// write EFUSE burn data
write 0xF2A00100 0xFFFFFFFF
write 0xF2A00104 0xFFFFFFFF
write 0xF2A00108 0xFFFFFFFF
write 0xF2A0010C 0xFFFFFFFF
// Start burn Efuse'
write 0xF2A00004 0x2
// wait 3ms for Efuse burn done
// Start read Efuse
write 0xF2A00000 0x1110
write 0xF2A00004 0x1 //read efuse and enable auto compare
// if compare success GPIO5 will output high
read 0xF2A00200
read 0xF2A00204
read 0xF2A00208
read 0xF2A0020C
```

### 18.3 软件烧写与读取 EFUSE

软件操作流程基本与 UART DEBUG 模式相同，但是软件在启动 EFUSE 写入或读取之后需要通过查询 RW\_START 寄存器状态来确定烧写或读取结束。

烧写流程如下：

配置 0xF2100054 为 0x100，打开 GPIO\_EFUSE\_EN，使能 GPIO8 和 GPIO5 的 EFUSE 复用选项，GPIO8 用于控制 EFUSE\_VDD（2.5V）的上电时间，GPIO5 用于输出 EFUSE 烧写是否正确的指示（可连接 LED）

写入需要烧写的数据到 0xF2A00100(PGM\_DATA\_0\_)到 PGM\_DATA\_3\_

写入 0x2 到 0xF2A00000(EFUSE\_EFUSE\_CTRL)寄存器中，开始烧写流程

等待烧写结束，一般在 2 到 5 个 ms 会完成烧写工作，如果是软件烧写的话，可以直接读取 EFUSE\_EFUSE\_CTRL 寄存器中的 RW\_START，烧写结束后硬件会将此域写 0

读取流程如下：

写入 0x1 到 0xF2A00000(EFUSE\_EFUSE\_CTRL)寄存器中，开始读取 EFUSE 中的数据

等待读取完成，可以直接读取 EFUSE\_EFUSE\_CTRL 寄存器中的 RW\_START，读取结束后硬件会将此域写 0

读取 0xF2A00200(EFUSE\_RDATA\_0\_)到 0xF2A0020C 寄存器值，就是 EFUSE 的数据

注：寄存器中的一些 burn time 硬件默认值是按 12.5M 的时钟计算的，软件操作的时候我们使用的是 50M 的 PCLK2，所以在运行 EFUSE 的读写的时候可以直接将 PCLK2 降低到 12.5M，EFUSE 完成后即可恢复 PCLK2 的频率，或者将 BURN\_TIME\_0 配置为 0x34cf0804，BURN\_TIME\_1 配置为 0xc3503c08，READ\_TIME 配置为 0x04080810，软件只使用读取的情况下，只需要配置 READ\_TIME 寄存器即可。

GPIO5 是自动比对功能的使能，一般用于批量烧录的时候自动检测，烧写完成后软件直接启动读取过程，同时使能 RD\_CMP\_EN 域，即使用 0x101 配置 EFUSE\_EFUSE\_CTRL 来进行读取，硬件会在读取完成后将读取到的数据和写到 GM\_DATA\_N\_中的数据进行比对。如果是相同的，那么将会拉高 GPIO5，不同的时候会拉低 GPIO5。

## 19 GE&RMII

芯片支持 MII、RGMII、RMII 三种接口，用于连接外部以太网 PHY 芯片，出于端口数目限制，建议使用 RMII 接口。GE 模块的寄存器控制分为两部分，GE 及 GEI、前者主要用于 GE 的细化控制，用户使用 SDK 的默认配置即可，GEI 为用户结构，具体寄存器如下表：

| 地址         | 寄存器名称                     |
|------------|---------------------------|
| 0xD0000800 | BASIC_CFG[31: 0]          |
| 0xD0000804 | RX_BUF_THD[31: 0]         |
| 0xD0000808 | BRDY_THD[31: 0]           |
| 0xD000080C | RX_PDMA_SADDR[31: 0]      |
| 0xD0000810 | RX_PDMA_LEN[31: 0]        |
| 0xD0000814 | RX_PDMA_RD_CNT[31: 0]     |
| 0xD0000818 | TX_PDMA_SADDR[31: 0]      |
| 0xD000081C | TX_AHB_BUF[31: 0]         |
| 0xD0000820 | TX_PDMA_BUF[31: 0]        |
| 0xD0000824 | TX_PDMA_INFO_HLEN[31: 0]  |
| 0xD0000828 | PDMA_INTR_NUM[31: 0]      |
| 0xD000082C | RX_PDMA_INTR_TIMER[31: 0] |
| 0xD0000830 | TX_PKT_INFO_CHK[31: 0]    |
| 0xD0000834 | INTR_EN[31: 0]            |
| 0xD0000838 | PDMA[31: 0]               |
| 0xD0000840 | GE_CFG[31: 0]             |
| 0xD0000844 | GE_ORDER[31: 0]           |
| 0xD0000850 | RX_PKT_INFO_VLD[31: 0]    |
| 0xD0000854 | RX_AHB_PKT_INFO[31: 0]    |
| 0xD0000858 | TX_PKT_INFO_FULL[31: 0]   |
| 0xD000085C | TX_AHB_PKT_INFO[31: 0]    |

**BASIC\_CFG[31: 0]寄存器：**

| 位                 | 功能                                                                   | 类型 |
|-------------------|----------------------------------------------------------------------|----|
| BASIC_CFG[31: 24] | 初始值：0x4，接收使用 PDMA 的时候，在包头预留的地址空间长度                                   | RW |
| BASIC_CFG[16]     | 初始值：0x0，LPBK 环回                                                      | RW |
| BASIC_CFG[9: 8]   | 初始值：0x0，发送侧优先级控制<br>0：AHB 总线通道高优先级<br>1：PDMA 通道高优先级<br>2：Round Robin | RW |
| BASIC_CFG[7]      | 初始值：0x0，对需要发送的包加上 CRC，默认情况下 GE 模块已经加上了 CRC，此处不需要再次添加                 | RW |
| BASIC_CFG[6]      | 初始值：0x0，对小于 64 的包进行 padding 操作，默认情况下 GE 模块中已经进行了此操作                  | RW |
| BASIC_CFG[5]      | 初始值：0x1，错误包丢弃使能                                                      | RW |
| BASIC_CFG[4]      | 删除接收包中的 CRC，默认情况下 GE 模块已经做过此处理                                       | RW |
| BASIC_CFG[3]      | 接收包 CRC 校验，默认情况下 GE 模块已经做过此处理                                        | RW |
| BASIC_CFG[2]      | 初始值：0x0，接收包传输模式选择：<br>0：AHB SLAVE by CPU or SDMA<br>1：PDMA           | RW |
| BASIC_CFG[1]      | 初始值：0x0，发送使能                                                         | RW |
| BASIC_CFG[0]      | 初始值：0x0，接收使能                                                         | RW |

**RX\_BUF\_THD[31: 0]寄存器：**

| 位                  | 功能                                                    | 类型 |
|--------------------|-------------------------------------------------------|----|
| RX_BUF_THD[27: 16] | 初始值：0x800，接收数据超过此水位，GE 会发起 PAUSE 帧，接收 BUF 最大为 2K word | RW |
| RX_BUF_THD[11: 0]  | 初始值：0x800，接收数据 BUF 深度，最大 2K word，超过此值后将丢包             | RW |

**BRDY\_THD[31: 0]寄存器：**

| 位               | 功能                       | 类型 |
|-----------------|--------------------------|----|
| BRDY_THD[14: 8] | 初始值：0x20，接收侧 SDMA 硬件流控水位 | RW |
| BRDY_THD[6: 0]  | 初始值：0x20，发送侧 SDMA 硬件流控水位 | RW |

**RX\_PDMA\_SADDR[31: 0]寄存器：**

| 位                    | 功能                             | 类型 |
|----------------------|--------------------------------|----|
| RX_PDMA_SADDR[31: 0] | 初始值：0x0，RX PDMA 起始地址，需 word 对齐 | RW |

## RX\_PDMA\_LEN[31: 0]寄存器：

| 位                  | 功能                                                | 类型 |
|--------------------|---------------------------------------------------|----|
| RX_PDMA_LEN[19: 0] | 初始值：0x0，RX PDMA 搬移最大环回空间，为 0 的时候代表 4MB，以 word 为单位 | RW |

## RX\_PDMA\_RD\_CNT[31: 0]寄存器：

| 位                     | 功能                                                             | 类型 |
|-----------------------|----------------------------------------------------------------|----|
| RX_PDMA_RD_CNT[19: 0] | 初始值：0x0，软件在使用完 PDMA 接收到的数据后需要写入软件已经处理了多长的数据，以通知硬件电路释放接收 BUF 空间 | RW |

## TX\_PDMA\_SADDR[31: 0]寄存器：

| 位                    | 功能                               | 类型 |
|----------------------|----------------------------------|----|
| TX_PDMA_SADDR[31: 0] | 初始值：0x0，发送 PDMA 模式起始地址，需 word 对齐 | RW |

## TX\_AHB\_BUF[31: 0]寄存器：

| 位                  | 功能                                 | 类型 |
|--------------------|------------------------------------|----|
| TX_AHB_BUF[26: 16] | 初始值：0x0，发送 AHB 模式 BUF 深度，单位 word   | RW |
| TX_AHB_BUF[9: 0]   | 初始值：0x0，发送 AHB 模式 BUF 起始地址，单位 word | RW |

## TX\_PDMA\_BUF[31: 0]寄存器：

| 位                   | 功能                                  | 类型 |
|---------------------|-------------------------------------|----|
| TX_PDMA_BUF[26: 16] | 初始值：0x0，发送 PDMA 模式 BUF 深度，单位 word   | RW |
| TX_PDMA_BUF[9: 0]   | 初始值：0x0，发送 PDMA 模式 BUF 起始地址，单位 word | RW |

## TX\_PDMA\_INFO\_HLEN[31: 0]寄存器：

| 位                       | 功能                                            | 类型 |
|-------------------------|-----------------------------------------------|----|
| TX_PDMA_INFO_HLEN[6: 0] | 初始值：0x0，PDMA 发送高优先级 INFO 深度设置，低优先级深度为 64-当前配置 | RW |

## PDMA\_INTR\_NUM[31: 0]寄存器：

| 位                     | 功能                                     | 类型 |
|-----------------------|----------------------------------------|----|
| PDMA_INTR_NUM[30: 24] | 初始值：0x1，PDMA 发送低优先级，INFO 数目中断水线，最小值为 1 | RW |
| PDMA_INTR_NUM[22: 16] | 初始值：0x1，PDMA 发送高优先级，INFO 数目中断水线，最小值为 1 | RW |
| PDMA_INTR_NUM[7: 0]   | 初始值：0x5，PDMA 接收 INFO 中断数目水线            | RW |

## RX\_PDMA\_INTR\_TIMER[31: 0]寄存器：

| 位                         | 功能                              | 类型 |
|---------------------------|---------------------------------|----|
| RX_PDMA_INTR_TIMER[31: 0] | 初始值：0x10000，PDMA 接收 INFO 非空超时配置 | RW |

TX\_PKT\_INFO\_CHK[31: 0]寄存器：

| 位                       | 功能                                                   | 类型 |
|-------------------------|------------------------------------------------------|----|
| TX_PKT_INFO_CHK[31]     | 初始值：0x1，发送大包丢弃使能，如果包长超过 2047Byte 那么不管此寄存器是否配置，都将直接丢弃 | RW |
| TX_PKT_INFO_CHK[27: 16] | 初始值：0x0600，大包丢弃功能包长配置                                | RW |
| TX_PKT_INFO_CHK[15]     | 初始值：0x0，发送小包丢弃使能                                     | RW |
| TX_PKT_INFO_CHK[11: 0]  | 初始值：0x30，小包丢弃功能包长配置                                  | RW |

INTR\_EN[31: 0]寄存器：

| 位           | 功能                             |             | 类型 |
|-------------|--------------------------------|-------------|----|
| INTR_EN[10] | 初始值：0x0，PDMA_BBUF_RERR_INTR_EN | PDMA 错误中断使能 | RW |
| INTR_EN[9]  | 初始值：0x0，PDMA_BBUF_WERR_INTR_EN |             | RW |
| INTR_EN[8]  | 初始值：0x0，PDMA_ERR_INTR_EN       |             | RW |

INTR\_EN[31: 0]寄存器：

| 位          | 功能                                   | 类型 |
|------------|--------------------------------------|----|
| INTR_EN[6] | 初始值：0x0，PDMA 模式发送出一个完整包中断使能          | RW |
| INTR_EN[5] | 初始值：0x0，AHB 模式发送出一个完整包中断使能           | RW |
| INTR_EN[4] | 初始值：0x0，GEI 模块发送端接收到 AHB 总线一个完整包中断使能 | RW |
| INTR_EN[3] | 初始值：0x0，GEI 模块发送端接收到 PDMA 一个完整包中断使能  | RW |
| INTR_EN[2] | 初始值：0x0，PDMA 接收包非空超时中断使能             | RW |
| INTR_EN[1] | 初始值：0x0，PDMA 接收包数目到达流水线中断使能          | RW |
| INTR_EN[0] | 初始值：0x0，AHB 接收包中断使能                  | RW |

PDMA[31: 0]寄存器：

| 位            | 功能        |           | 类型 |
|--------------|-----------|-----------|----|
| PDMA[30: 24] | 初始值：0x2，  | PDMA 模块控制 | RW |
| PDMA[22: 16] | 初始值：0x20， |           | RW |
| PDMA[7: 0]   | 初始值：0x20， |           | RW |

GE\_CFG[31: 0]寄存器：

| 位              | 功能                                                                                                                |                                           | 类型 |
|----------------|-------------------------------------------------------------------------------------------------------------------|-------------------------------------------|----|
| GE_CFG[31]     | 初始值：0x0，清空 GE 模块内部计数器                                                                                             |                                           | RW |
| GE_CFG[27: 24] | 初始值：0x4，RMII 10M 模式采样点                                                                                            |                                           | RW |
| GE_CFG[16]     | 初始值：0x1，gtxc 输出使能                                                                                                 |                                           |    |
| GE_CFG[15: 12] | 初始值：0x4，RX 时钟相位配置：<br>0: 3*36; 1: 1*36; 2: 2*36; 3: 3*36; 4: 4*36; 5: 5*36;<br>6: 6*36; 7: 7*36; 8: 8*36; 9: 9*36 |                                           | RW |
| GE_CFG[11: 8]  | 初始值：0x5，TX 时钟相位配置：<br>0: 3*36; 1: 1*36; 2: 2*36; 3: 3*36; 4: 4*36; 5: 5*36;<br>6: 6*36; 7: 7*36; 8: 8*36; 9: 9*36 |                                           | RW |
| GE_CFG[3]      | 初始值：0x1，                                                                                                          | 2'b00: MII_MODE                           | RW |
| GE_CFG[2]      | 初始值：0x0，                                                                                                          | 2'b01: RGMII_MODE                         | RW |
|                |                                                                                                                   | 2'b10: RMII_MODE                          |    |
| GE_CFG[1: 0]   | 初始值：0x1，                                                                                                          | 2'b00: 10M<br>2'b01: 100M<br>2'b1x: 1000M | RW |

GE\_ORDER[31: 0]寄存器：

| 位                | 功能                                                                            | 类型 |
|------------------|-------------------------------------------------------------------------------|----|
| GE_ORDER[13]     | 初始值：0x0，交换 TXEN 和 GTXC 信号                                                     | RW |
| GE_ORDER[12]     | 初始值：0x0，交换 TXD[3: 0]的顺序                                                       | RW |
| GE_ORDER[11: 10] | 初始值：0x0，<br>0: GTXC, TXEN, TXD;<br>1: TXEN, TXD, GTXC;<br>2: TXD, GTXC, TXEN; | RW |
| GE_ORDER[9]      | 初始值：0x0，交换 RXD[3: 0]的顺序                                                       | RW |
| GE_ORDER[8]      | 初始值：0x0，<br>0: RXDV, RXD;<br>1: RXD, RXDV;                                    | RW |

RX\_PKT\_INFO\_VLD[31: 0]寄存器：

| 位                       | 功能                                | 类型 |
|-------------------------|-----------------------------------|----|
| RX_PKT_INFO_VLD[23: 16] | 初始值：0x0，PDMA 接收队列中有效的 PKT_INFO 数目 | RO |
| RX_PKT_INFO_VLD[0]      | 初始值：0x0，AHB 接收 PKT_INFO 有效标志      | RO |

TX\_AHB\_PKT\_INFO[31: 0]寄存器：

| 位                      | 功能                         | 类型 |
|------------------------|----------------------------|----|
| TX_AHB_PKT_INFO[31: 0] | 初始值：0x0，AHB 接收 PKT_INFO 信息 | RO |



TX\_PKT\_INFO\_FULL[31: 0]寄存器:

| 位                        | 功能                                      | 类型 |
|--------------------------|-----------------------------------------|----|
| TX_PKT_INFO_FULL[31]     | 初始值: 0x0, AHB TX 模式, PKT_INFO FIFO 满    | RO |
| TX_PKT_INFO_FULL[30]     | 初始值: 0x0, PDMA TX 高速通道, PKT_INFOFIFO 满  | RO |
| TX_PKT_INFO_FULL[29]     | 初始值: 0x0, PDMA TX 低速通道, PKT_INFO FIFO 满 | RO |
| TX_PKT_INFO_FULL[20: 16] | 初始值: 0x0, AHB TX 模式, PKT_INFO 数目        | RO |
| TX_PKT_INFO_FULL[14: 8]  | 初始值: 0x0, PDMA TX 高速通道                  | RO |
| TX_PKT_INFO_FULL[6: 0]   | 初始值: 0x0, PKT_INFO 数目                   | RO |

TX\_AHB\_PKT\_INFO[31: 0]寄存器:

| 位                      | 功能                            | 类型 |
|------------------------|-------------------------------|----|
| TX_AHB_PKT_INFO[31: 0] | 初始值: 0x0, AHB TX 模式, PKT_INFO | RO |

从 GE 口接收的数据目前可以通过两种方式提供给软件处理。一种是将接收到的以太网包通过 PDMA 主动写入到 SDRAM 中, 称为主动 PDMA 方式; 另一个种是将接收的以太网包暂时缓存起来, 等待软件以直接或者通过 SDMA 方式, 通过 AHB 的一组 Slave 接口从接口模块读取得到以太网包, 称为被动 AHB 方式。GE 接口示意图如下:

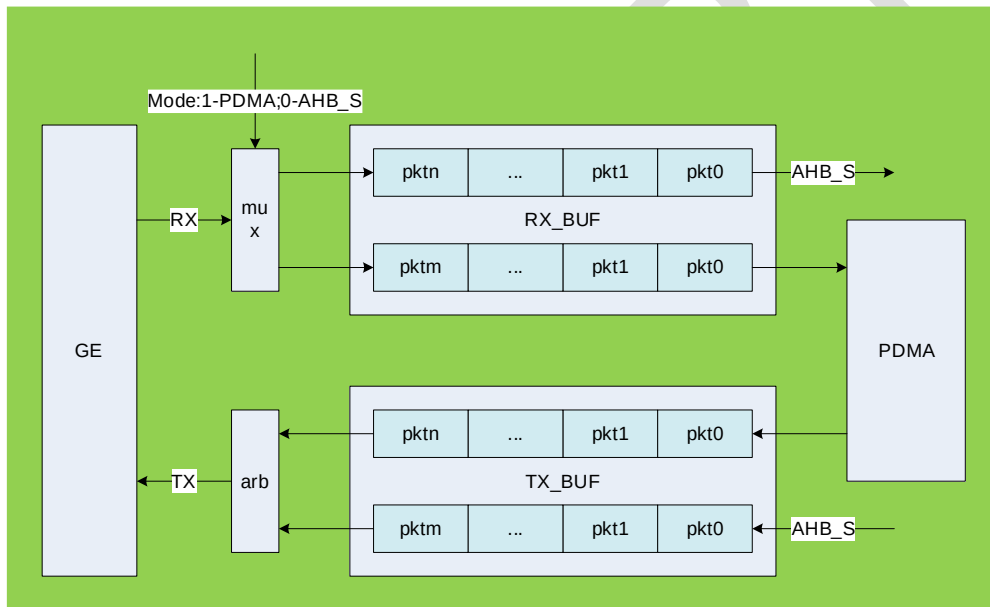


图 19-1 GEI 接口结构

## 19.1 以太网包获取方式

软件通过配置寄存器来选择采用哪种方式来获取以太网包。

SET\_ETH\_PKT\_MODE();

1: 主动 PDMA;

0: 被动 AHB。

在工作过程中软件可以根据需要, 随时修改这个寄存器的配置。硬件上, 在每个以太网包的 SOP 时采样这个寄存器, 确定当前这个以太网包应该以什么方式提供给软件。

## 19.2 GEI 地址说明

0xD00008xx 地址段为 GEI 寄存器地址。

0xD00010xx 地址段为 PDMA\_PKT\_INFO 地址段，读取 0xD0001000 到 0xD0001200 地址为 rx\_pdma\_pkt\_info\_fifo 信息，写入 0xD0001000 到 0xD0001100 即写入 tx\_pdma\_hpkt\_info\_fifo 和 tx\_pdma\_lpkt\_info\_fifo，0xD0001000 到 (0xD0001000+TX\_PDMA\_INFO\_HLEN\*4) 为 tx\_pdma\_hpkt\_info\_fifo 地址，其他剩余部分为 tx\_pdma\_lpkt\_info\_fifo 地址。

0xD00020xx 地址段为被动 AHB 模式的发送和写入地址。

## 19.3 RX 接收数据

### 19.3.1 被动 AHB 方式

根据 ETH\_PKT\_MODE，硬件每次接收完成一个被动 AHB 方式的以太网帧，产生这个帧的信息如下，PKT\_LEN 表示当前这个以太网包的字节数量。

|                        |              |
|------------------------|--------------|
| Packet Length - 12bits | RSV - 20bits |
|------------------------|--------------|

接收完成一个 AHB 通道的帧后，硬件给出一个中断，AHB\_RX\_PKT\_RDY\_INTR，通知软件有一个以太网帧在 AHB SLAVE 上准备好了，软件可以通过 AHB 总线来读取这个帧了。或者由软件不断的查询中断状态寄存器 IS\_AHB\_RX\_RDY\_DONE，来获取这个信息。

软件通过以上任意一个方式得知后，要先查询 GEI\_RX\_AHB\_PKT\_INFO\_VLD 这个寄存器，读返回为 1.则可以读取 GEI\_RX\_AHB\_PKT\_INFO 这个寄存器获取帧信息。

然后开始通过 AHB SLAVE 来读取这个帧。硬件上给出中断后，会等待软件来读取这个帧的状态，直到这个帧被读走，才进行接下来的操作。

注：PKT\_INFO\_VLD 寄存器可以查询多次。只能在 PKT\_INFO\_VLD 为 1 时，去读取 PKT\_INFO 的返回才有效，且在每次获得 PKT\_INFO\_VLD 为 1 后只能被读一次 PKT\_INFO，再次读之前一定要先读 PKT\_INFO\_VLD，确定返回为 1，才能再读 PKT\_INFO。

## 19.4 TX 发送数据

### 19.4.1 TX 的 BUF 分配

当前 TX 侧总的缓存空间大小为 4KB（1k word）。TX 为 AHB 和 PDMA 并行接收数据。将 TX 侧的 BUFFER 分配给 AHB 和 PDMA 两个通道。

AHB\_TX\_BUF\_ADDR (10bit);     //AHB 通道缓存空间的起始地址  
 AHB\_TX\_BUF\_LEN (10bit);     //AHB 通道缓存空间的 word 数量，全零表示 1024  
 PDMA\_TX\_BUF\_ADDR (10bit);     //PDMA 通道缓存空间的起始地址  
 PDMA\_TX\_BUF\_LEN (10bit);     //PDMA 通道缓存空间的 word 数量，全零表示 1024

TX 侧输出仲裁

SET GEI\_TX\_ART (2bit) // 0: 优先发送 AHB 通道的帧。  
                           // 1: 优先发送 PDMA 通道的帧  
                           // 2: 两个通道轮询发送

### 19.4.2 被动 AHB 方式

软件有需要通过 AHB 总线写给 GE 发送的帧时，将帧信息按照如下格式写入硬件，PKT\_LEN 表示当前这个以太网包的字节数量。

| Sop<br>(1bit) | Eop<br>(1bit) | Part Packet Length -<br>11bits | Rsv(19bit) |
|---------------|---------------|--------------------------------|------------|
|---------------|---------------|--------------------------------|------------|

软件首先要读取 GEI\_TX\_AHB\_PKT\_INFO\_FULL，在这个值为 0 时，才能将要发送的帧信息通过 GEI\_TX\_AHB\_PKT\_INFO 寄存器写给硬件。

软件要保证先写入帧信息，再往 AHB 总线上发送数据，否则硬件不接收 AHB 总线上的数据，总线会被占用。

硬件会首先判断当前 TX\_AHB 通道剩余的缓存空间是否足够接收下将要处理的 PKT\_INFO 中的数据长度。若足够，则给出一个 AHT\_TX\_PKT\_RDY\_INTR 中断，然后到 AHB 总线上去等待或者接收数据。

硬件每次每次处理完一个 PKT\_INFO 后，即从 AHB 总线接收到一个 PKT\_INFO 的数据后，会给出一个 AHB\_TX\_PKT\_DONE\_INTR。

## 20 ACU通用加解密引擎

芯片内置一个通用的硬件加解密引擎，其特性如下：

- AES ECB/CBC；SMS4 ECB 模式加解密
- 支持 SDMA 硬件流控模式

### 20.1 ACU 加解密流程

ACU 加解密工作流程如下：

- 模块初始化，配置 ACU 的工作模式，配置密钥等
- 传输数据，一般需调用 SDMA
- ACU 开始工作

### 20.2 AES/SMS4 计算

ACU AES/SMS4 相关寄存器如下：

| 地址         | 寄存器名称               |
|------------|---------------------|
| 0xF0300000 | GLB_CFG[31: 0]      |
| 0xF0300004 | CTRL[31: 0]         |
| 0xF0300008 | DATA_ORDER[31: 0]   |
| 0xF030000C | INTR_THD[31: 0]     |
| 0xF0300010 | READY_THD[31: 0]    |
| 0xF0300014 | KEY_INTR_THD[31: 0] |
| 0xF0300018 | INTR_STA[31: 0]     |
| 0xF030001C | INTR_EN[31: 0]      |
| 0xF0300040 | KEY_0_[31: 0]       |
| 0xF0300044 | KEY_1_[31: 0]       |
| 0xF0300048 | KEY_2_[31: 0]       |
| 0xF030004C | KEY_3_[31: 0]       |
| 0xF0300050 | KEY_RD_0_[31: 0]    |
| 0xF0300054 | KEY_RD_1_[31: 0]    |
| 0xF0300058 | KEY_RD_2_[31: 0]    |
| 0xF030005C | KEY_RD_3_[31: 0]    |
| 0xF0300060 | IV_0_[31: 0]        |
| 0xF0300064 | IV_1_[31: 0]        |
| 0xF0300068 | IV_2_[31: 0]        |
| 0xF030006C | IV_3_[31: 0]        |
| 0xF0300070 | KEY_COUNT[31: 0]    |
| 0xD1000xxx | CRC_DATA_IN[31: 0]  |
| 0xD1008xxx | DATA_IN_OUT[31: 0]  |

GLB\_CFG[31: 0]寄存器:

| 位             | 功能                                               | 类型 |
|---------------|--------------------------------------------------|----|
| GLB_CFG[28]   | 初始值: 0x1, 加解密 key 计数                             | RW |
| GLB_CFG[24]   | 初始值: 0x1, CRC 计算使能                               | RW |
| GLB_CFG[20]   | 初始值: 0x1, 解密功能使能                                 | RW |
| GLB_CFG[16]   | 初始值: 0x1, 加密功能使能                                 | RW |
| GLB_CFG[8]    | 初始值: 0x1, 加解密功能选择<br>0: 解密<br>1: 加密              | RW |
| GLB_CFG[7: 4] | 初始值: 0x0, 加解密模式 0: ECB<br>1: CBC<br>other: NA    | RW |
| GLB_CFG[3: 0] | 初始值: 0x0, 加解密算法选择 0: AES<br>1: SMS4<br>other: NA | RW |

CTRL[31: 0]寄存器:

| 位            | 功能                                                    | 类型 |
|--------------|-------------------------------------------------------|----|
| CTRL[31: 16] | 初始值: 0x0,<br>AES/SMS4 模式: BLOCK 个数<br>CRC 模式: 输入数据字节数 | RW |
| CTRL[0]      | 初始值: 0x0, 指示 CBC 第一个数据块                               | AO |

DATA\_ORDER[31: 0]寄存器:

| 位                | 功能                                                                                         | 类型 |
|------------------|--------------------------------------------------------------------------------------------|----|
| DATA_ORDER[6: 4] | 初始值: 0x0, 输出结果反序<br>3'bx1: Byte 内比特反序<br>3'bx1x: Word 内 Byte 反序<br>3'b1xx: Block 中 Word 反序 | RW |
| DATA_ORDER[2: 0] | 初始值: 0x0, 输入数据反序<br>3'bx1: Byte 内比特反序<br>3'bx1x: Word 内 Byte 反序<br>3'b1xx: Block 中 Word 反序 | RW |

INTR\_THD[31: 0]寄存器:

| 位                | 功能                               | 类型 |
|------------------|----------------------------------|----|
| INTR_THD[31: 16] | 初始值: 0x20, 输出数据 BUF 中断水线, 最大值 64 | RW |

READY\_THD[31: 0]寄存器:

| 位                 | 功能                             | 类型 |
|-------------------|--------------------------------|----|
| READY_THD[31: 16] | 初始值: 0x20, 输出 SDMA 硬件流控 BUF 大小 | RW |
| READY_THD[15: 0]  | 初始值: 0x20, 输入 SDMA 硬件流控 BUF 大小 | RW |

KEY\_INTR\_THD[31: 0]寄存器:

| 位                   | 功能                      | 类型 |
|---------------------|-------------------------|----|
| KEY_INTR_THD[31: 0] | 初始值: 0x0, 加解密 key 计数值中断 | RW |

INTR\_STA[31: 0]寄存器:

| 位           | 功能                     | 类型 |
|-------------|------------------------|----|
| INTR_STA[8] | 初始值: 0x0, key 使用次数中断状态 | LH |
| INTR_STA[4] | 初始值: 0x0, CRC 结束中断状态   | LH |
| INTR_STA[1] | 初始值: 0x0, 加解密完成中断状态    | LH |
| INTR_STA[0] | 初始值: 0x0, 可接收数据中断状态    | LH |

INTR\_EN[31: 0]寄存器:

| 位          | 功能                     | 类型 |
|------------|------------------------|----|
| INTR_EN[8] | 初始值: 0x0, key 使用次数中断使能 | RW |
| INTR_EN[4] | 初始值: 0x0, CRC 结束中断使能   | RW |
| INTR_EN[1] | 初始值: 0x0, 加解密完成中断使能    | RW |
| INTR_EN[0] | 初始值: 0x0, 可接收数据中断使能    | RW |

KEY\_0\_[31: 0], KEY\_1\_[31: 0], KEY\_2\_[31: 0], KEY\_3\_[31: 0]寄存器:

| 位             | 功能                     | 类型 |
|---------------|------------------------|----|
| KEY_0_[31: 0] | 初始值: 0x0, key[31: 0]   | RW |
| KEY_1_[31: 0] | 初始值: 0x0, key[63: 32]  | RW |
| KEY_2_[31: 0] | 初始值: 0x0, key[95: 64]  | RW |
| KEY_3_[31: 0] | 初始值: 0x0, key[127: 96] | RW |

KEY\_RD\_0\_[31: 0], KEY\_RD\_1\_[31: 0], KEY\_RD\_2\_[31: 0], KEY\_RD\_3\_[31: 0]寄存器:

| 位                | 功能                         | 类型 |
|------------------|----------------------------|----|
| KEY_RD_0_[31: 0] | 初始值: 0x0, out key[31: 0]   | RO |
| KEY_RD_1_[31: 0] | 初始值: 0x0, out key[63: 32]  | RO |
| KEY_RD_2_[31: 0] | 初始值: 0x0, out key[95: 64]  | RO |
| KEY_RD_3_[31: 0] | 初始值: 0x0, out key[127: 96] | RO |

IV\_0\_[31: 0], IV\_1\_[31: 0], IV\_2\_[31: 0], IV\_3\_[31: 0]寄存器:

| 位            | 功能       | 类型 |
|--------------|----------|----|
| IV_0_[31: 0] | 初始值: 0x0 | RO |
| IV_1_[31: 0] | 初始值: 0x0 | RO |
| IV_2_[31: 0] | 初始值: 0x0 | RO |
| IV_3_[31: 0] | 初始值: 0x0 | RO |

KEY\_COUNT[31: 0]寄存器:

| 位                | 功能                                             | 类型 |
|------------------|------------------------------------------------|----|
| KEY_COUNT[31: 0] | 初始值: 0x0, key 使用计数器高 32 位输出, 内部使用 40 位 key 计数器 | RW |

CRC\_DATA\_IN[31: 0]寄存器:

| 位                  | 功能                     | 类型 |
|--------------------|------------------------|----|
| CRC_DATA_IN[31: 0] | 初始值: 0x0, CRC 计算数据输入地址 | RW |

ACU\_DATA\_IO[31: 0]寄存器:

| 位                  | 功能                           | 类型 |
|--------------------|------------------------------|----|
| ACU_DATA_IO[31: 0] | 初始值: 0x0, AES/SMS4 加解密数据输入地址 | RW |

### 20.2.1 ACU SDMA 使用说明

ACU 模块需要 SDMA 配合工作，需要对 SDMA 做相关配置。配置过程如下：

配置 GBC SDMA 硬件流控寄存器(GBC\_DFC\_EN)

DFC\_ACU\_CRC\_RX 用于配置数据源端到 ACU 端的 SDMA 通道

DFC\_ACU\_CRC\_TX 用于配置 ACU 端到目的地址段的 SDMA 通道

配置 SDMA 相关寄存器

```
SET_SDMA_CH_EN(1'b1, ch_idx); // 通道使能
```

```
SET_SDMA_BLK_SIZE(blk_size, ch_idx); // block size, 必须小于 5(32 Byte)
```

```
SET_SDMA_SFC_EN(hs_en[0], ch_idx); // 源端流控对应于 DFC_ACU_CRC_TX
```

```
SET_SDMA_DFC_EN(hs_en[1], ch_idx); // 目的端流控对应于 DFC_ACU_CRC_RX
```

```
SET_SDMA_SA_MODE(sa_mode, ch_idx); // RAM 端配置为递增模式, ACU 端配置为固定模
```

式

```
SET_SDMA_DA_MODE(da_mode, ch_idx); // 同 SA 模式
```

```
SET_SDMA_SA(sa, ch_idx); // ACU 端地址为 0xD1008xxx
```

```
SET_SDMA_DA(da, ch_idx);
```

```
SET_SDMA_LEN(len, ch_idx); // 加解密长度
```

```
SET_SDMA_SW_HS(1'b1, ch_idx); // 启动 SDMA
```

### 20.2.2 AES/SMS4 使用注意事项

使用 sms4 算法加密时，软件需要将输入的 key 异或 FK 之后再填入 key 寄存器。FK 为：FK=128'hA3B1BAC656AA3350677D9197B27022DC。

AES/SMS4 加解密如果使用 SDMA 搬移数据的话，需要同时使用两个 SDMA 通道，并且需要打开硬件流控功能。

### 20.3 实例 img 文件解密

此模块支持对 SPI FLASH 中的加密文件进行解密。这里以对 AES 128bit ECB 模式为例，软件知道 img 大小（也支持对 img 长度进行加密）。需要如下步骤：

- 初始 ACU 模块

```
SET_ACU_ENC(0); // use decrypt mode
```

```
SET_ACU_DATA_LEN(img_len); // config img length
```

```
SET_ACU_KEY(aes_key); // set decrypt key
```

- 解密数据，使用两条 SDMA 通道并打开硬件流控，一条通道将数据从 FLASH 中搬移至 ACU 中，另外一条用到将 ACU 解密数据从 ACU 搬移至 SDRAM 中。

- 软件使用解密后的 img

注：如果解密之前不知道 img 的长度，那么需要在解密全部完成之前能够解出 img 的长度信息，并在最后一个 block 块解密之前将正确的长度信息配置到寄存器中（软件需要用 img 的长度减掉已经解密的长度），否则最后一个 block 可能解密错误。

## 21 WDT看门狗

芯片内置硬件看门狗 WDT。看门狗支持输出如下复位信号：

- 模拟电路复位（硬件上如果将 RESET\_OUT#和 RESET\_IN#连接，那么复位模拟电路的时候也会导致系统复位）
- GPIO 独立输出复位信号
- 全局复位

看门狗复位寄存器如下：

| 地址         | 寄存器名称                   |
|------------|-------------------------|
| 0xF2300000 | WDOG_CFG[31: 0]         |
| 0xF2300004 | WDOG_CLR[31: 0]         |
| 0xF2300008 | WDOG_IND[31: 0]         |
| 0xF230000C | WDOG_RST_CNT_CFG[31: 0] |
| 0xF2300010 | WDOG_RST_CFG[31: 0]     |

WDOG\_CFG[31: 0]寄存器：

| 位               | 功能                                                | 类型 |
|-----------------|---------------------------------------------------|----|
| WDOG_CFG[31]    | 初始值：0x0，看门狗使能                                     | RW |
| WDOG_CFG[27: 0] | 初始值：0xffffffff，看门狗计数器，看门狗溢出时间为：WDOG_CFG*PCLK2*256 | RW |

WDOG\_CLR[31: 0]寄存器：

| 位           | 功能                         | 类型 |
|-------------|----------------------------|----|
| WDOG_CLR[0] | 初始值：0x0，写 1 清零看门狗计数器，即喂狗操作 | AO |

WDOG\_IND[31: 0]寄存器：

| 位               | 功能                                                                           | 类型 |
|-----------------|------------------------------------------------------------------------------|----|
| WDOG_IND[15: 0] | 初始值：N/A，看门狗复位标志，如果读取值为 0xaa55，表示产生过看门狗复位。此芯片可以直接读取 GBC 中 WDT_GLB_RST_GEN 来代替 | RO |

WDOG\_RST\_CNT\_CFG[31: 0]寄存器：

| 位                       | 功能                                                              | 类型 |
|-------------------------|-----------------------------------------------------------------|----|
| WDOG_RST_CNT_CFG[24: 0] | 初始值：0x1ffff，看门狗复位信号持续时钟周期数目，主要用于 GPIO 复位，用于 GLB 复位的时候，此寄存器也会被清零 | RW |

WDOG\_RST\_CFG[31: 0]寄存器：

| 位               | 功能                                         | 类型 |
|-----------------|--------------------------------------------|----|
| WDOG_RST_CFG[2] | 初始值：0x0，模拟复位输出                             | RW |
| WDOG_RST_CFG[1] | 初始值：0x0，GPIO 复位输出使能，需要 GPIO 配置 GPIO_WDT_EN | RW |
| WDOG_RST_CFG[0] | 初始值：0x0，芯片全局复位使能                           | RW |

### 21.1 看门狗复位控制

RESET\_OUT#连接到 RESET\_IN#端口上，对模拟电路复位的时候同时会对数字电路进行全局复位。全局复位的复位效果等同于在 GBC 中使用 SYS\_RST\_EN 寄存器对芯片进行全局复位。但是 GBC 中 WDT\_GLB\_RST\_GEN 寄存器会记录看门狗复位。

GPIO 复位可将 WDT 用于复位 PLC 的外围芯片。支持对外围芯片进行独立看门狗复位。复位信号持续时长受 WDOG\_RST\_CNT\_CFG 控制。注意如果同时使能 WDOG 全局复位，那么 GPIO 的复位时长就不受寄存器控制，而是会产生一个复位脉冲。



## 22 TICK硬件timer

芯片内置 TICK 模块，用于维护系统 tick 级别时钟同步和硬件 timer 功能。TICK-TIMER 模块功能主要分为系统 tick 时钟维护和硬件 timer（HTIMER）单元两个大块。前者主要服务于 PLC 物理层，应用软件层尽量不要使用此部分内容；后者可以用作普通的硬件 timer 功能，支持输出 PWM。

### 22.1 TICK 功能

TICK 功能如下：

|   |                         |
|---|-------------------------|
| 1 | Tick 粒度可配，Tick 时间可调     |
| 2 | 维护 32bit 的 Tick 粒度的时钟计数 |
| 3 | 产生 Tick 粒度的 Timer，供系统调用 |
| 4 | 可记录事件（Event）发生的 Tick 时间 |
| 5 | 基于 Timer 控制 IO 输出       |
| 6 | 支持过零点事件记录               |

TICK 相关寄存器如下：

| 地址                    | 寄存器名称                   |
|-----------------------|-------------------------|
| 0xF1300000            | INTR_MASK[31: 0]        |
| 0xF1300004            | INTR_STAT[31: 0]        |
| 0xF1300008            | FREQ[31: 0]             |
| 0xF130000C            | FO[31: 0]               |
| 0xF1300010            | SET_IMD[31: 0]          |
| 0xF1300014            | ADJ[31: 0]              |
| 0xF1300018            | CURR[31: 0]             |
| 0xF130001C            | INTV[31: 0]             |
| 0xF1300020            | MISC[31: 0]             |
| 0xF1300030~0xF130005C | TIMER_n_[31: 0], n=0~11 |
| 0xF1300070~0xF130009C | EVENT_n_[31: 0], n=0~11 |
| 0xF13000A0            | EVENT_SRC[31: 0]        |
| 0xF13000A4            | EVENT_SRC1[31: 0]       |

INTR\_MASK[31: 0]寄存器：

| 位                 | 功能                                              | 类型 |
|-------------------|-------------------------------------------------|----|
| INTR_MASK[27: 16] | 初始值：0x0，TICK 事件中断使能，每个 bit 对应于一个 TICK 事件        | RW |
| INTR_MASK[15: 12] | 初始值：0x0，Cyclic-Timers 事件中断使能，每个 bit 对应于一个 timer | RW |
| INTR_MASK[11: 0]  | 初始值：0x0，TICK timer 硬件中断使能，每个 bit 对应于一个 timer    | RW |

INTR\_STAT[31: 0]寄存器：

| 位                 | 功能                              | 类型 |
|-------------------|---------------------------------|----|
| INTR_STAT[27: 16] | 初始值：0x0，对应于 INTR_MASK 中断使能的中断状态 | LH |
| INTR_STAT[15: 12] | 初始值：0x0，对应于 INTR_MASK 中断使能的中断状态 | LH |
| INTR_STAT[11: 0]  | 初始值：0x0，对应于 INTR_MASK 中断使能的中断状态 | LH |

**FREQ[31: 0]寄存器:**

| 位                                                                                                                                  | 功能                 | 类型 |
|------------------------------------------------------------------------------------------------------------------------------------|--------------------|----|
| FREQ[11: 8]                                                                                                                        | 初始值: 0x1, freq_rmd | RW |
| FREQ[7: 4]                                                                                                                         | 初始值: 0x2, freq_div | RW |
| FREQ[3: 0]                                                                                                                         | 初始值: 0x1, freq_int | RW |
| TICK 频率计算, 计算公式: $\text{Tick\_frequency} = (\text{freq\_int} + (\text{freq\_rmd}/(\text{freq\_div}+1))) * \text{Clock\_frequency}$ |                    |    |

**FO[31: 0]寄存器:**

| 位         | 功能            | 类型 |
|-----------|---------------|----|
| FO[31: 0] | 初始值: 0x0, SFO | RW |

**SET\_IMD[31: 0]寄存器:**

| 位              | 功能                       | 类型 |
|----------------|--------------------------|----|
| SET_IMD[31: 0] | 初始值: 0x0, TICK 值配置, 立即生效 | RW |

**ADJ[31: 0]寄存器:**

| 位          | 功能                           | 类型 |
|------------|------------------------------|----|
| ADJ[15: 0] | 初始值: 0x0, TICK 调整, 此寄存器为带符号数 | RW |

**CURR[31: 0]寄存器:**

| 位           | 功能                  | 类型 |
|-------------|---------------------|----|
| CURR[31: 0] | 初始值: 0x0, 实时 TICK 值 | RW |

**INTV[31: 0]寄存器:**

| 位            | 功能                                                                         | 类型 |
|--------------|----------------------------------------------------------------------------|----|
| INTV[31: 28] | 初始值: 0x6, 结束事件                                                             | RW |
| INTV[27: 24] | 初始值: 0xB, 起始事件                                                             | RW |
| INTV[21: 20] | 初始值: 0x1, 计算起始结束事件时间差模式<br>0: 每次更新时间差<br>1: 锁存最大时间差<br>2: 锁存最小时间差<br>3: NA | RW |
| INTV[19: 0]  | 初始值: 0x0, 起始结束事件时间差输出                                                      | RO |

**MISC[31: 0]寄存器:**

| 位          | 功能                                                | 类型 |
|------------|---------------------------------------------------|----|
| MISC[3: 2] | 初始值: 0x0, Htimer 外部事件捕获方式<br>[0]: 上升沿<br>[1]: 下降沿 | RW |
| MISC[1: 0] | 初始值: 0x1, 同 EXT_SRC1_EDGE; 此外部源支持用作过零点检测          | RW |

**TIMER\_n\_[31: 0]寄存器:**

| 位               | 功能                                             | 类型 |
|-----------------|------------------------------------------------|----|
| TIMER_n_[31: 0] | 初始值: 0xffffffff, TICK 事件值设置 n=0: 11, 共 12 个寄存器 | RW |

EVENT\_n\_[31: 0]寄存器:

| 位               | 功能                                      | 类型 |
|-----------------|-----------------------------------------|----|
| EVENT_n_[31: 0] | 初始值: 0x0, TICK 事件锁存值 n=0: 11, 共 12 个寄存器 | RW |

EVENT\_SRC[31: 0]寄存器:

| 位                 | 功能                     |                                            | 类型 |
|-------------------|------------------------|--------------------------------------------|----|
| EVENT_SRC[31: 28] | 初始值: 0x0, EVENT_SRC_7_ | 事件源选择<br>0~13: 系统内部源<br>14: 外部事件<br>15: NA | RW |
| EVENT_SRC[27: 24] | 初始值: 0x0, EVENT_SRC_6_ |                                            | RW |
| EVENT_SRC[23: 20] | 初始值: 0x0, EVENT_SRC_5_ |                                            | RW |
| EVENT_SRC[19: 16] | 初始值: 0x0, EVENT_SRC_4_ |                                            | RW |
| EVENT_SRC[15: 12] | 初始值: 0x0, EVENT_SRC_3_ |                                            | RW |
| EVENT_SRC[11: 8]  | 初始值: 0x0, EVENT_SRC_2_ |                                            | RW |
| EVENT_SRC[7: 4]   | 初始值: 0x0, EVENT_SRC_1_ |                                            | RW |
| EVENT_SRC[3: 0]   | 初始值: 0x0, EVENT_SRC_0_ |                                            | RW |

EVENT\_SRC1[31: 0]寄存器:

| 位                  | 功能                      |             | 类型 |
|--------------------|-------------------------|-------------|----|
| EVENT_SRC1[15: 12] | 初始值: 0x0, EVENT_SRC_11_ | 事件源选择       | RW |
| EVENT_SRC1[11: 8]  | 初始值: 0x0, EVENT_SRC_10_ | 0~13: 系统内部源 | RW |
| EVENT_SRC1[7: 4]   | 初始值: 0x0, EVENT_SRC_9_  | 14: 外部事件    | RW |
| EVENT_SRC1[3: 0]   | 初始值: 0x0, EVENT_SRC_8_  | 15: NA      | RW |

### 22.1.1 Tick 计数器

Tick counter 是一个 32bit 计数器, 计数到最大值  $2^{32}-1$  后, 折返为 0, 重新计数, 当前的 Tick 值可通过 CURR 寄存器读出。

#### Tick 频率设置

Tick 频率可配。

举例: 系统时钟为 75MHz, Tick 频率为 25MHz。

$$25 \text{ MHz} = (0 + (1/3)) * 75 \text{ MHz}$$

$$\text{FREQ\_INT} = 0$$

$$\text{FREQ\_RMD} = 1$$

$$\text{FREQ\_DIV} = 3 - 1 = 2$$

#### Tick 计数器调整

可以通过 SET\_IMD 直接配置 32bit 的 TICK 值, 立即生效。

可以通过配置 ADJ, 调整相应的 Tick 值。正值为往前调整; 负值为向后调整。

### 22.1.2 Tick Timers

可以基于 Tick counter 产生 32bit Timer, 当 Tick counter 达到 Tick Timer 时, 可以触发相应 Timer, Timer 可以触发以下操作:

- 产生可屏蔽中断 (见 Tick interrupt)
  - 控制某些 GPIO (见 Tick IO output)
  - 触发某些特定硬件动作, 如 PHY frame 的发送等
- 一共有 12 个 Timer, 其中前 8 个保留用以触发硬件动作。

### 22.1.3 Tick 事件

硬件会产生一些事件源（Event source），这些事件发生的 Tick 时间可以被锁存到 12 个可配的事件（Event0~11），同时可以触发以下操作：

- 产生可屏蔽中断（见 Tick interrupt）
- 控制某些 GPIO（见 Tick IO output）

举例：Zero-crossing 通过 External Source event 的 IO 送到 Tick 中。

把 EVENT\_SRC\_0 设置为 15，并将 EXT\_SRC\_EDGE 设为 0（上升沿）。那么当 External Source IO 的上升沿发生，此时的 32bit Tick 值就会被记录在 EVENT\_0\_中。并可以触发中断通知软件，发生过一次过零事件，过零事件的时间可通过 EVENT\_0\_读取。

### 22.2 HTIMER 功能

提供 4 个 32bit 与 Tick 无关的独立 timer。每个 timer 具备如下功能：

|   |                               |
|---|-------------------------------|
| 1 | 可设置初值                         |
| 2 | 可配 1~256 分频                   |
| 3 | 可在计数过程中修改计数值                  |
| 4 | 支持 4 个匹配时刻（HTMRx_MAT[0~3]）    |
| 5 | 支持匹配时对 counter 清零             |
| 6 | 支持匹配触发可屏蔽中断                   |
| 7 | 匹配时刻可以调制输出到 GPIO（见 IO Output） |

HTIMER 模块结构图如下：

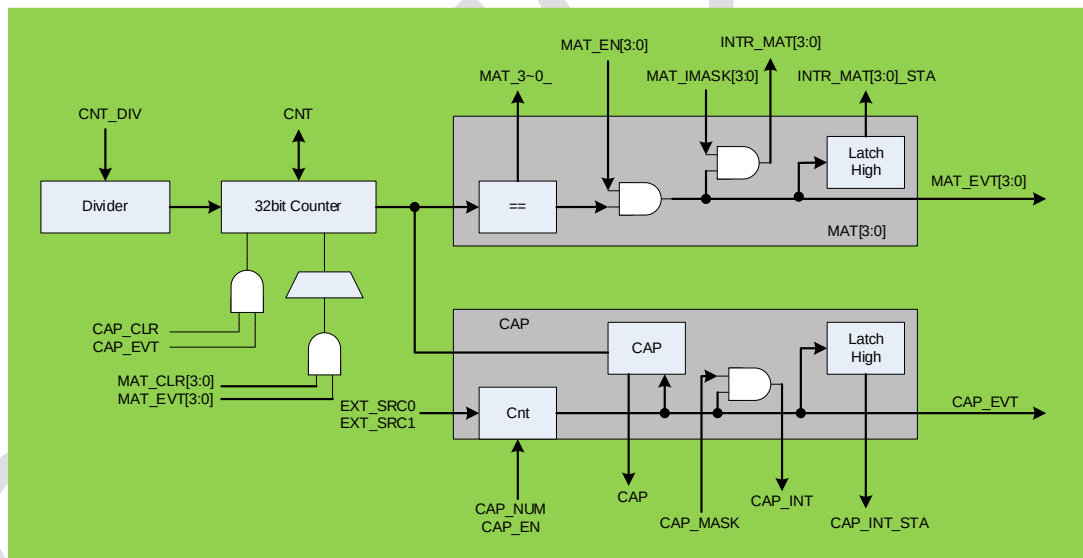


图 22-1 HTIMER 结构图

HTIMER 相关寄存器如下：

| 地址         | 寄存器名称             |
|------------|-------------------|
| 0xF1300300 | HTMR0_CNT[31: 0]  |
| 0xF1300304 | HTMR0_CFG[31: 0]  |
| 0xF1300308 | HTMR0_CAP[31: 0]  |
| 0xF1300310 | HTMR0_MAT0[31: 0] |
| 0xF1300314 | HTMR0_MAT1[31: 0] |
| 0xF1300318 | HTMR0_MAT2[31: 0] |
| 0xF130031C | HTMR0_MAT3[31: 0] |
| 0xF1300380 | HTMR_INTR[31: 0]  |

注：0xF1300300~0xF130031C 为 HTIMER0 配置寄存器，0xF1300320~0xF130037C 为 HTIMER1~3 的配置寄存器。

HTMR0\_CNT[31: 0]寄存器：

| 位                | 功能                                    | 类型 |
|------------------|---------------------------------------|----|
| HTMR0_CNT[31: 0] | 初始值：0x0，硬件 timer0 计数器，写入的时候直接修改当前的计数值 | RW |

HTMR0\_CFG[31: 0]寄存器：

| 位                 | 功能                                        | 类型 |
|-------------------|-------------------------------------------|----|
| HTMR0_CFG[31: 28] | 初始值：0x0，硬件 timer0 计数器，写入的时候直接修改当前的计数值     | RW |
| HTMR0_CFG[27: 24] | 初始值：0x0，事件匹配时清零 timer 计数，每个 bit 对应于一个匹配事件 | RW |
| HTMR0_CFG[23: 20] | 初始值：0x0，计数匹配使能，每个 bit 对应于一个匹配事件           | RW |
| HTMR0_CFG[19: 16] | 初始值：0x0，捕获次数配置，当捕获次数到达此配置值时触发捕获事件         | RW |
| HTMR0_CFG[15]     | 初始值：0x0，捕获事件中断使能                          | RW |
| HTMR0_CFG[14]     | 初始值：0x0，捕获事件清零 timer 使能                   | RW |
| HTMR0_CFG[13: 12] | 初始值：0x0，捕获功能使能                            | RW |
| HTMR0_CFG[11: 4]  | 初始值：0x0，TIMER 计数器时钟分频                     | RW |
| HTMR0_CFG[0]      | 初始值：0x0，TIMER 功能使能                        | RW |

HTMR0\_CAP[31: 0]寄存器：

| 位                | 功能                            | 类型 |
|------------------|-------------------------------|----|
| HTMR0_CAP[31: 0] | 初始值：0x0，捕获事件发生时，锁存的 TIMER 计数值 | RO |

HTMR0\_MAT0[31: 0]~HTMR0\_MAT3[31: 0]寄存器：

| 位                 | 功能                                   | 类型 |
|-------------------|--------------------------------------|----|
| HTMR0_MATn[31: 0] | 初始值：0xffffffff，HTIMER 匹配计数值配置，n=0: 3 | RW |

HTIMR\_INTR[31: 0]寄存器:

| 位                  | 功能                       | 类型 |
|--------------------|--------------------------|----|
| HTIMR_INTR[19]     | 初始值: 0x0, TIMER3 捕获中断状态  | RW |
| HTIMR_INTR[18]     | 初始值: 0x0, TIMER2 捕获中断状态  | RW |
| HTIMR_INTR[17]     | 初始值: 0x0, TIMER1 捕获中断状态  | RW |
| HTIMR_INTR[16]     | 初始值: 0x0, TIMER0 捕获中断状态  | RW |
| HTIMR_INTR[15: 12] | 初始值: 0x0, HTIMER3 匹配中断状态 | RW |
| HTIMR_INTR[11: 8]  | 初始值: 0x0, HTIMER2 匹配中断状态 | RW |
| HTIMR_INTR[7: 4]   | 初始值: 0x0, HTIMER1 匹配中断状态 | RW |
| HTIMR_INTR[3: 0]   | 初始值: 0x0, HTIMER0 匹配中断状态 | RW |

### 22.2.1 HTIMER 计数器

以 HTIMER0 为例: HTMR0\_CNT\_DIV 可将 timer 的工作时钟进行 1~256 倍的分频。

HTMR0\_CNT\_EN 为计数使能, 为 1 时, counter 开始计数; counter 记到 32bit 全 1 时, 返回 0 继续计数。读取 HTMR0\_CNT 可以得到当前的 counter 值。无论 HTMR0\_CNT\_EN 为何种状态, 写 HTMR0\_CNT 可以立即对 counter 赋值。

### 22.2.2 HTIMER 匹配事件

对每个 Timer 都可以设置 4 个 32bit match 值 (HTMR0\_MAT0~3)。这样, 当相应的 match 使能打开时 (HTMR0\_MAT\_EN[3: 0]相应位为高), counter 计数等于设定的 match 值时会触发一个脉冲事件。

该脉冲事件可以用来:

- 如果 HTMR0\_MAT\_CLR 相应位为高, 脉冲会将 counter 清零
- 如果 HTMR0\_MAT\_IMSK 相应位为高, 脉冲会触发中断
- 调制 IO 输出

该脉冲会以高锁存的方式, 记录在寄存器 HTMR0\_MAT\_INTR 相应位中。一共有 4 个 timer, 每个 timer 有 4 个 match 值, 故一共有 16 种 match 事件。

### 22.2.3 HTIMER 捕获

支持外部事件通过 GPIO 捕捉:

可选两个 GPIO 当做事件触发源 EXT\_SOURCE0/1 (其中 EXT\_SOURCE0 一般作为过零测使用)

可选事件触发源 EXT\_SOURCE0/1 上升沿, 下降沿触发, 或双沿触发

可配事件触发次数 HTMR0\_CAP\_NUM, 范围 1~16 次。每当触发次数被满足时, 会触发一次捕捉脉冲

捕捉脉冲会将当时的 Timer counter 计数记录下来, 可以通过只读寄存器 HTMR0\_CAP 读取。下一次捕捉脉冲会覆盖该值

捕捉脉冲会以高锁存的方式, 记录在寄存器 HTMR0\_CAP\_INTR 中

如果 HTMR0\_CAP\_CLR 相应位为高, 捕捉脉冲会将 Timer counter 清零

如果 HTMR0\_CAP\_IMSK 相应位为高, 捕捉脉冲会触发中断

## 22.3 IO 输出功能

IO output 内部结构图如下：

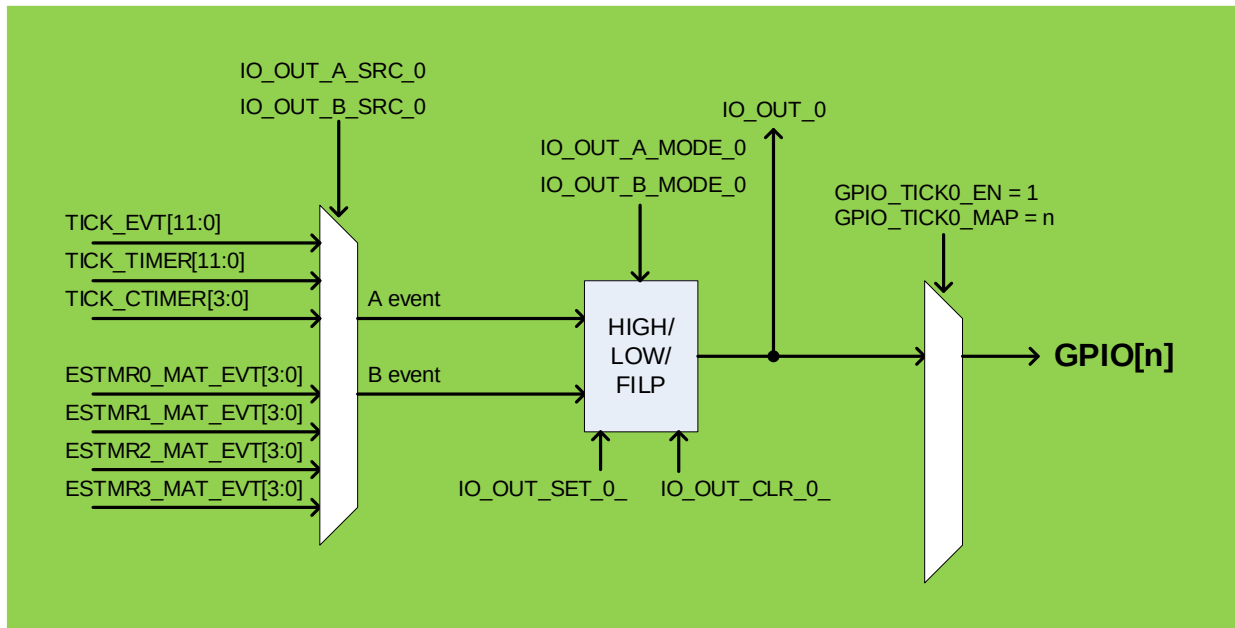


图 22-2 IO output 结构

IO output 相关寄存器地址：

| 地址         | 寄存器名称                 |
|------------|-----------------------|
| 0xF1300100 | IO_OUT[31: 0]         |
| 0xF1300104 | IO_OUT_SET_CLR[31: 0] |
| 0xF1300110 | IO_OUT_0_CFG[31: 0]   |
| 0xF1300114 | IO_OUT_1_CFG[31: 0]   |
| 0xF1300118 | IO_OUT_2_CFG[31: 0]   |
| 0xF130011C | IO_OUT_3_CFG[31: 0]   |

IO\_OUT[31: 0]寄存器：

| 位         | 功能                    | 类型 |
|-----------|-----------------------|----|
| IO_OUT[0] | 初始值：0x0，TICK IO 3 输出值 | RO |
| IO_OUT[1] | 初始值：0x0，TICK IO 2 输出值 | RO |
| IO_OUT[2] | 初始值：0x0，TICK IO 1 输出值 | RO |
| IO_OUT[3] | 初始值：0x0，TICK IO 0 输出值 | RO |

### IO\_OUT\_SET\_CLR[31: 0]:

| 位                 | 功能                        | 类型 |
|-------------------|---------------------------|----|
| IO_OUT_SET_CLR[7] | 初始值: 0x0, TICK IO 3 设置为 0 | AO |
| IO_OUT_SET_CLR[6] | 初始值: 0x0, TICK IO 2 设置为 0 | AO |
| IO_OUT_SET_CLR[5] | 初始值: 0x0, TICK IO 1 设置为 0 | AO |
| IO_OUT_SET_CLR[4] | 初始值: 0x0, TICK IO 0 设置为 0 | AO |
| IO_OUT_SET_CLR[3] | 初始值: 0x0, TICK IO 3 设置为 1 | AO |
| IO_OUT_SET_CLR[2] | 初始值: 0x0, TICK IO 2 设置为 1 | AO |
| IO_OUT_SET_CLR[1] | 初始值: 0x0, TICK IO 1 设置为 1 | AO |
| IO_OUT_SET_CLR[0] | 初始值: 0x0, TICK IO 0 设置为 1 | AO |

### IO\_OUT\_0\_CFG[31: 0]寄存器:

| 位                    | 功能                                                                                                                                | 类型 |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------|----|
| IO_OUT_0_CFG[25: 24] | 初始值: 0x0, TICK IO 0 B 输出模式                                                                                                        | RW |
| IO_OUT_0_CFG[21: 16] | 初始值: 0x3f, TICK IO 0 B 事件源选择                                                                                                      | RW |
| IO_OUT_0_CFG[9: 8]   | 初始值: 0x0, TICK IO 0 A 输出模式<br>0: 输出低电平<br>1: 输出高电平<br>2: 输出翻转信号<br>3: 输出单周期脉冲                                                     | RW |
| IO_OUT_0_CFG[5: 0]   | 初始值: 0x3f, TICK IO 0 A 事件源选择 0~11: TICK timer 12~15: Cyclic timer<br>16~27: 事件 0 到事件 11<br>32~35: HTIMER0 事件<br>36~39: HTIMER1 事件 | RW |

Tick Timer, Tick Cyclic Timer 和 Tick Event 以及 HTIMER 的 16 个 match 事件, 均可调制 IO 输出。一共有 4 个 IO 可被控制, 可以通过 GPIO MUX 模块 (见 GPIO 用户手册) 映射到任意 GPIO 上。

以 IO\_OUT0 为例:

可以通过 IO\_OUT\_A\_SRC\_0\_ 和 IO\_OUT\_B\_SRC\_0\_, 为 IO\_OUT0 选择两个的触发事件源(Aevent 和 B event)。

然后通过 IO\_OUT\_A\_MODE\_0\_ 和 IO\_OUT\_B\_MODE\_0\_ 分别配置当该 Aevent 和 B event 发生时, IO\_OUT0 的行为 (包括输出低/高电平, 输出脉冲, 输出翻转。)注意, 当 A event 和 B event 同时发生时, 以 A event 优先。

可以读取 IO\_OUT\_0\_ 寄存器, 查看当前 IO\_OUT0 的高低状态。

可以通过 IO\_OUT\_SET\_0\_, 将 IO\_OUT0 置高; 设置 IO\_OUT\_CLR\_0\_, 将 IO\_OUT0 置低。注意 IO\_OUT\_SET\_0\_ 和 IO\_OUT\_CLR\_0\_ 为'A0'属性, 即自动清除。



## 22.4 PWM 输出实例

这里给出一个使用 HTIMER 来输出三分之一占空比的 PWM 配置实例：

```
`SET_GPIO_GPIO_TICK0_EN(1);
`SET_GPIO_GPIO_TICK0_MAP(31); // pad_gpio_10
`SET_TICK_IO_OUT_A_SRC_0_(32) // 对应第一路 match 事件
`SET_TICK_IO_OUT_A_MODE_0 OUT_HIGH_
`SET_TICK_IO_OUT_B_SRC_0_(33)
`SET_TICK_IO_OUT_B_MODE_1 OUT_LOW_
`SET_TICK_HTMRO_CNT_DIV (8'h0)
`SET_TICK_HTMRO_MAT_EN (4'b0011) // 使能两路 match
`SET_TICK_HTMRO_MAT_CLR (4'b0010) // 第二路 match 的时候 clear 计数器
`SET_TICK_HTMRO_MAT_0_ (10-1) // 第一次 match 事件的计数器值
`SET_TICK_HTMRO_MAT_1_ (15-1)
`SET_TICK_HTMRO_CNT_EN (1)
```

需要产生占空比可以调整的 PWM 需要使用 HTIMER 中两路 match 事件，当第一路 match 的时候产生事件 32，并将 IO 驱动为高电平。当第二路 match 的时候产生事件 33 并将 IO 驱动为低电平。具体事件触发过程如下图所示：

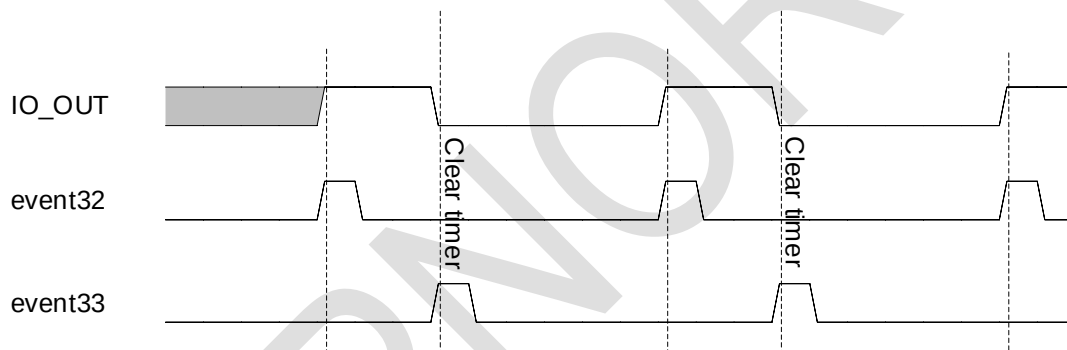


图 22-3 HTIMER 调制 PWM 示意图

## 23 封装信息

### 19.1CCO 封装

CCO 采用 10mmX10mm，88-pin MQFN 封装，封装尺寸图如下：

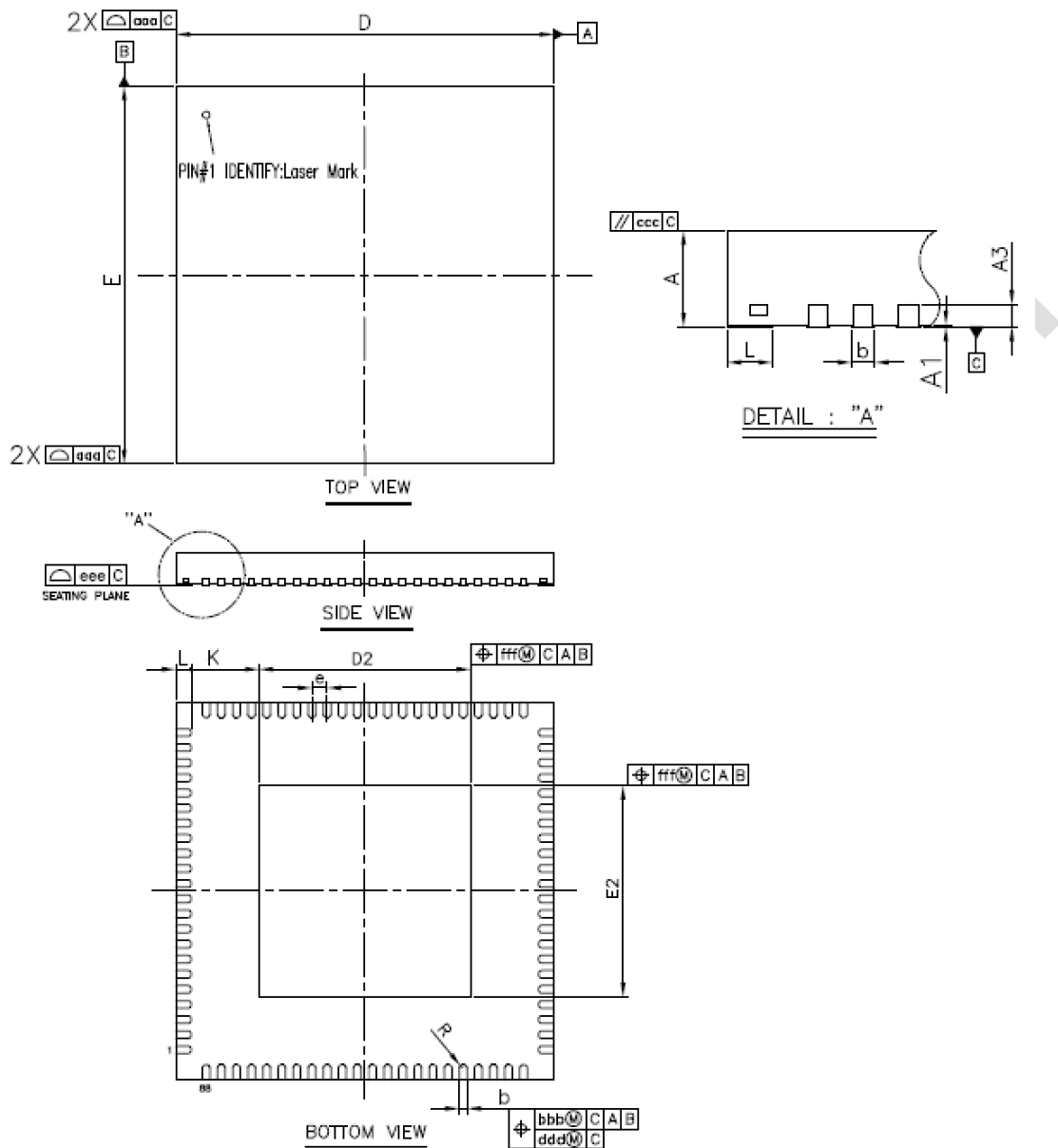


图 23-1 CCO 封装尺寸

CCO 封装参数:

| Symbol | Dimension in mm |       |       | Dimension in inch |       |       |
|--------|-----------------|-------|-------|-------------------|-------|-------|
|        | Min             | Nom   | Max   | Min               | Nom   | Max   |
| A      | 0.80            | 0.85  | 0.90  | 0.031             | 0.033 | 0.035 |
| A1     | 0.00            | 0.02  | 0.05  | 0.000             | 0.001 | 0.002 |
| A3     | 0.20 REF        |       |       | 0.008 REF         |       |       |
| b      | 0.15            | 0.20  | 0.25  | 0.006             | 0.008 | 0.010 |
| D      | 9.90            | 10.00 | 10.10 | 0.390             | 0.394 | 0.398 |
| E      | 9.90            | 10.00 | 10.10 | 0.390             | 0.394 | 0.398 |
| D2     | 5.50            | 5.60  | 5.70  | 0.217             | 0.220 | 0.224 |
| E2     | 5.50            | 5.60  | 5.70  | 0.217             | 0.220 | 0.224 |
| e      | 0.40 BSC        |       |       | 0.016 BSC         |       |       |
| L      | 0.30            | 0.40  | 0.50  | 0.012             | 0.016 | 0.020 |
| K      | 0.20            | ---   | ---   | 0.008             | ---   | ---   |
| R      | 0.075           | ---   | 0.125 | 0.003             | ---   | 0.005 |
| aaa    | 0.10            |       |       | 0.004             |       |       |
| bbb    | 0.07            |       |       | 0.003             |       |       |
| ccc    | 0.10            |       |       | 0.004             |       |       |
| ddd    | 0.05            |       |       | 0.002             |       |       |
| eee    | 0.08            |       |       | 0.003             |       |       |
| fff    | 0.10            |       |       | 0.004             |       |       |

## 19.2 STA 封装

STA 采用 8mmX8mm，68-pin QFN 封装，封装尺寸图如下：

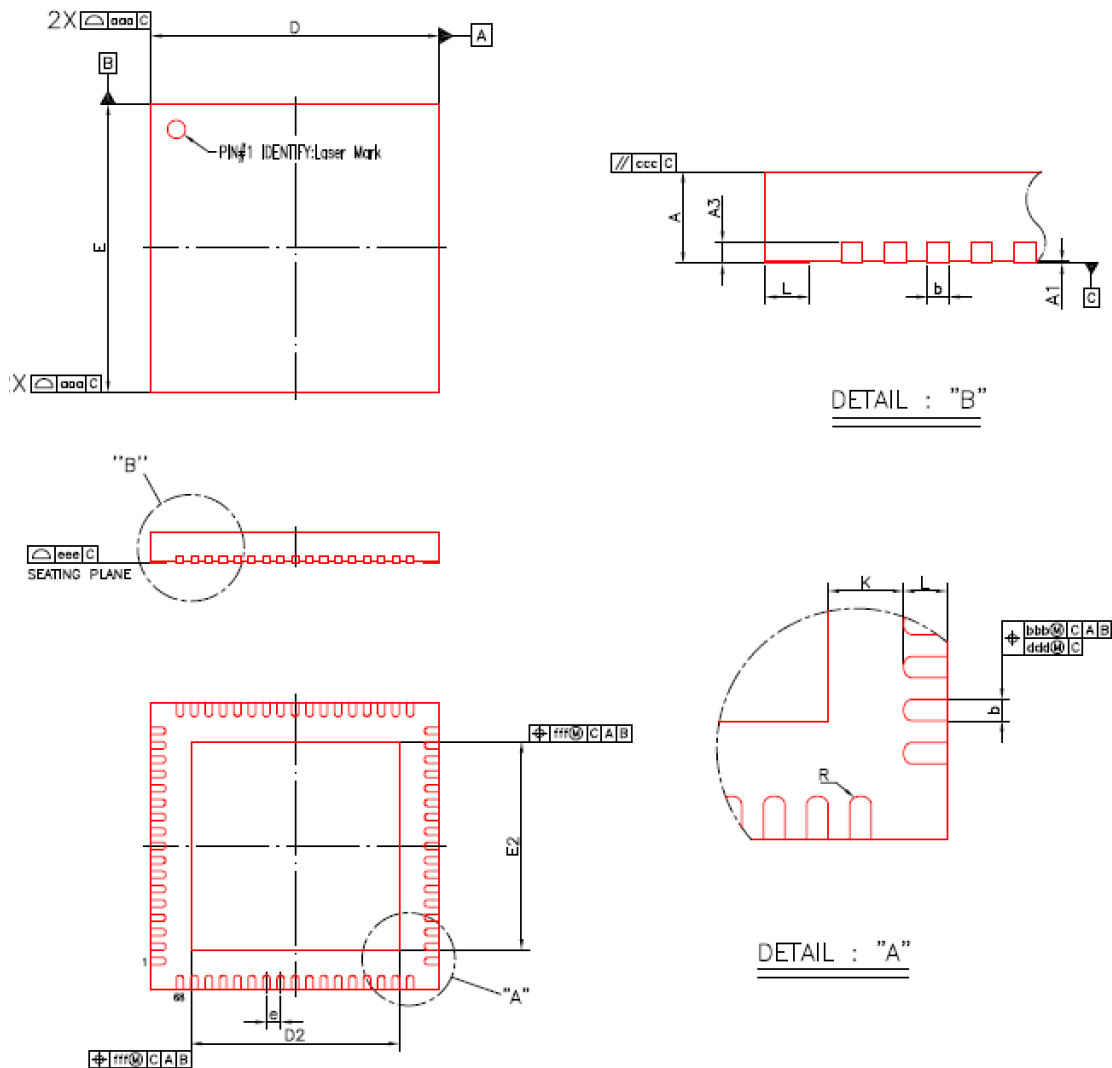


图 23-2 STA 封装尺寸

STA 封装参数:

| Symbol | Dimension in mm |      |      | Dimension in inch |       |       |
|--------|-----------------|------|------|-------------------|-------|-------|
|        | Min             | Nom  | Max  | Min               | Nom   | Max   |
| A      | 0.80            | 0.85 | 0.90 | 0.031             | 0.033 | 0.035 |
| A1     | 0.00            | 0.02 | 0.05 | 0.000             | 0.001 | 0.002 |
| A3     | 0.20 REF        |      |      | 0.008 REF         |       |       |
| b      | 0.15            | 0.20 | 0.25 | 0.006             | 0.008 | 0.010 |
| D/E    | 7.90            | 8.00 | 8.10 | 0.311             | 0.315 | 0.319 |
| D2     | 5.50            | 5.80 | 5.95 | 0.222             | 0.228 | 0.234 |
| E2     | 5.65            | 5.80 | 5.95 | 0.222             | 0.228 | 0.234 |
| e      | 0.40 BSC        |      |      | 0.016 BSC         |       |       |
| L      | 0.30            | 0.40 | 0.50 | 0.012             | 0.016 | 0.020 |
| R      | 0.075           | ---  | ---  | 0.003             | ---   | ---   |
| K      | 0.20            | ---  | ---  | 0.008             | ---   | ---   |
| aaa    | 0.10            |      |      | 0.004             |       |       |
| bbb    | 0.07            |      |      | 0.003             |       |       |
| ccc    | 0.10            |      |      | 0.004             |       |       |
| ddd    | 0.05            |      |      | 0.002             |       |       |
| eee    | 0.08            |      |      | 0.003             |       |       |
| fff    | 0.10            |      |      | 0.004             |       |       |

## 24 重要声明

芯北电子科技（南京）有限公司及其子公司保留对本文件及本文所述任何产品进行修改、改进、更正或其他变更的权利，恕不另行通知。芯北电子科技（南京）有限公司不承担因使用本文件或本文所述任何产品而产生的任何责任；芯北电子科技（南京）有限公司也不转让其专利权或商标权及其他权利的任何许可。在使用本文件或本文所述产品的任何客户或用户应承担所有风险，并同意芯北电子科技（南京）有限公司和其产品在芯北电子科技（南京）有限公司网站上展示的所有公司免受任何损害。

对于通过未经授权的销售渠道购买的任何产品，芯北电子科技（南京）有限公司不作任何保证，也不承担任何责任。如果客户购买或使用芯北电子科技（南京）有限公司的产品用于任何非预期或未经授权的用途，客户应赔偿芯北电子科技（南京）有限公司及其代表，使其免受因直接或间接引起的任何人身伤害或死亡造成的所有索赔、损害赔偿和律师费。