HPLC 数据手册

Specifications are subject to change without notice.

目录

目录

目录 2
1. 芯片简介 5
1.1. 产品基本特性 5
1.2. 应用范围 5
1.3. 电气特性 6
1.3.1. 电源参数 6
1.3.2. 功耗参数 6
2.引脚配置 7
2.1. V6201E1 引脚配置 7
2.2. V6211E1 引脚配置 8
2.3. 引脚说明 9
2.3.1. 电源相关接口 9
2.3.2. 工厂测试模式控制 10
2.3.3. JTAG 端口 10
2.3.4. 时钟复位控制 10
2.3.5. 通用 GPIO 11
2.3.6. 外围接口 11
2.3.7. 网络接口 12
2.3.8. 模拟接口 14
3. 电源及复位 15
3.1. 电源系统 15
3.2. 复位系统 15
3.3. 芯片上电时序 15
4. 系统结构 17
4.1. 数字功能模块 17
4.2. 模拟功能模块 18
4.3. 嵌入式 CPU 18
4.4. 中断控制系统 18
4.4.1. 中断处理 19
4.5. 芯片内部寻址空间 19
5. 系统时钟 21
6. GBC 芯片全局控制 22
6.1. 全局时钟控制 22
6.1.1.时钟配置说明 23
6.2. 全局复位控制 24
6.3. DMA 硬件连接映射 25
6.4. 全局中断控制 26
6.5. 总线控制 30
7. SDMA 系统 DMA 控制器 32
7.1. SDMA 通道传输控制 32
7.1.1. 通道地址及长度控制 35
7.1.2. 握手模式 36
7.1.3. 通道级联 36
7.2. SDMA 通用配置 36

7.2.1. SDMA 通道优先级.....36
8. 存储控制.....38
8.1. MC 内存控制器.....38
8.2. LRAM 片内存储.....38
9. GPIO 通用输入输出.....39
9.1. GPIO PAD 复用.....39
9.1.1. GPIO PAD 复用实例.....44
9.2. GPIO 输入输出功能.....44
9.2.1. GPIO 电气性能.....45
9.2.2. 特殊功能 GPIO.....45
9.3. GPIO 外部中断功能.....45
10. SPI MASTER.....47
10.1. SPI SCK 及模式控制.....48
10.2. SPI CS 片选控制.....49
10.3. SPI BOOT FLASH 器件限制.....49
11. SPI_SLAVE.....51
11.1. 软件使用注意事项.....53
12. UART 串行接口.....54
12.1. UART 波特率.....57
12.1.1. 波特率检测.....57
12.2. UART 中断.....58
12.3. 红外载波控制.....58
12.4. UART DEBUG 模式.....58
13. EFUSE 一次性写入单元.....59
13.1. 烧写功能外部硬件电路.....60
13.2. UART DEBUG 模式写入.....60
13.3. 软件烧写与读取 EFUSE.....62
14. MDIO.....63
14.1. MAC 模式编程实例.....64
15. GE&RMII.....65
15.1 方式选择.....69
15.2. GEI 地址说明.....70
15.3. RX 接收数据.....70
15.3.1. 被动 AHB 方式.....70
15.4. TX 发送数据.....70
15.4.1. TX 的 BUF 分配.....70
15.4.2. 被动 AHB 方式.....71
16. WDT 看门狗.....72
16.1 看门狗复位控制.....72
17. ACU 通用加解密引擎.....74
17.1. ACU 加解密流程.....74
17.2. CRC 计算.....74
17.2.1. CRC32 参数配置.....75
17.2.2. CRC24 参数配置.....75
17.3. AES/SMS4 计算.....76
17.3.1. ACU SDMA 使用说明.....78
17.3.2. AES/SMS4 使用注意事项.....78
17.4. 实例 img 文件解密.....78

18. TICK 硬件 timer.....80
18.1. TICK 功能.....80
18.1.1. Tick 计数器.....82
18.1.2. Tick Timers.....82
18.1.3. Tick 事件.....83
18.2. HTIMER 功能.....83
18.2.1. HTIMER 计数器.....85
18.2.2. HTIMER 匹配事件.....85
18.2.3. HTIMER 捕获.....85
18.3. IO 输出功能.....85
18.4. PWM 输出实例.....87
19. 芯片封装尺寸.....89
19.1. V6201E1 封装.....89
19.2. V6211E1 封装.....91
声明.....93

1. 芯片简介

1.1. 产品基本特性

- 嵌入式高性能低功耗 tensilica Xtensa ® 233L CPU
  - 最高工作于 400MHz
  - 支持全局可屏蔽中断
  - 支持 JTAG 调试
- 支持电网 PLC 协议
  - 支持 ROBO 模式
  - 高吞吐率 Turbo 解码引擎
  - 支持外部过零点检测
- 支持系统频率动态切换，有效降低系统功耗
- 片内 SRAM 1MB
- 可扩展内封 SDRAM（V6211E1）
- 支持系统 DMA 功能
- 内置通用加解密引擎，128bit AES/SMS4
- 支持 SPI master 功能
  - 支持 CPU 从 SPI 端口 boot
  - 支持通用 SPI 功能
  - 复位状态下，SPI 控制器所有管脚为高阻态
- UART 标准通信（5 组）
  - UART0 可支持系统 debug 功能
- 支持通用 GPIO 功能
- 支持 32bit timer 功能（4 组）
- 支持看门狗
- 支持 SPI slave 功能
- 支持 128bit 一次性写入 EFUSE 功能

1.2. 应用范围

- 支持使用 MDIO 及 RMII 端口连接外部以太网 PHY 芯片
- 远程抄表
- 智能家居
- 工业自动化控制
- 智能楼宇

1.3. 电气特性

远程监控

表格 1：电源参数

1.3.1. 电源参数

名称	描述	MIN	TYP	MAX	UNIT
DCDC_OUT	内部 DCDC 转换输出	1.0	1.1	1.2	V
VDD11	数字电源供给	1.0	1.1	1.2	V
VDD33	IO、内封 Flash 或 SDRAM 电源供给	3.1	3.3	3.5	V
VDDA	模拟电源供给	3.1	3.3	3.5	V

1.3.2. 功耗参数

VDDA25	EFUSE 电源供给	2.3	2.5	2.7	V

表格 2：典型工作电流参数

名称	描述	STATIC	DYNAMIC	UNIT
VDD33 VDDA	3.3V 电源供给			mA
VDD11	数字核心电源供给			mA

注：STATIC 为 CPU 正常工作，未启动物理层接收

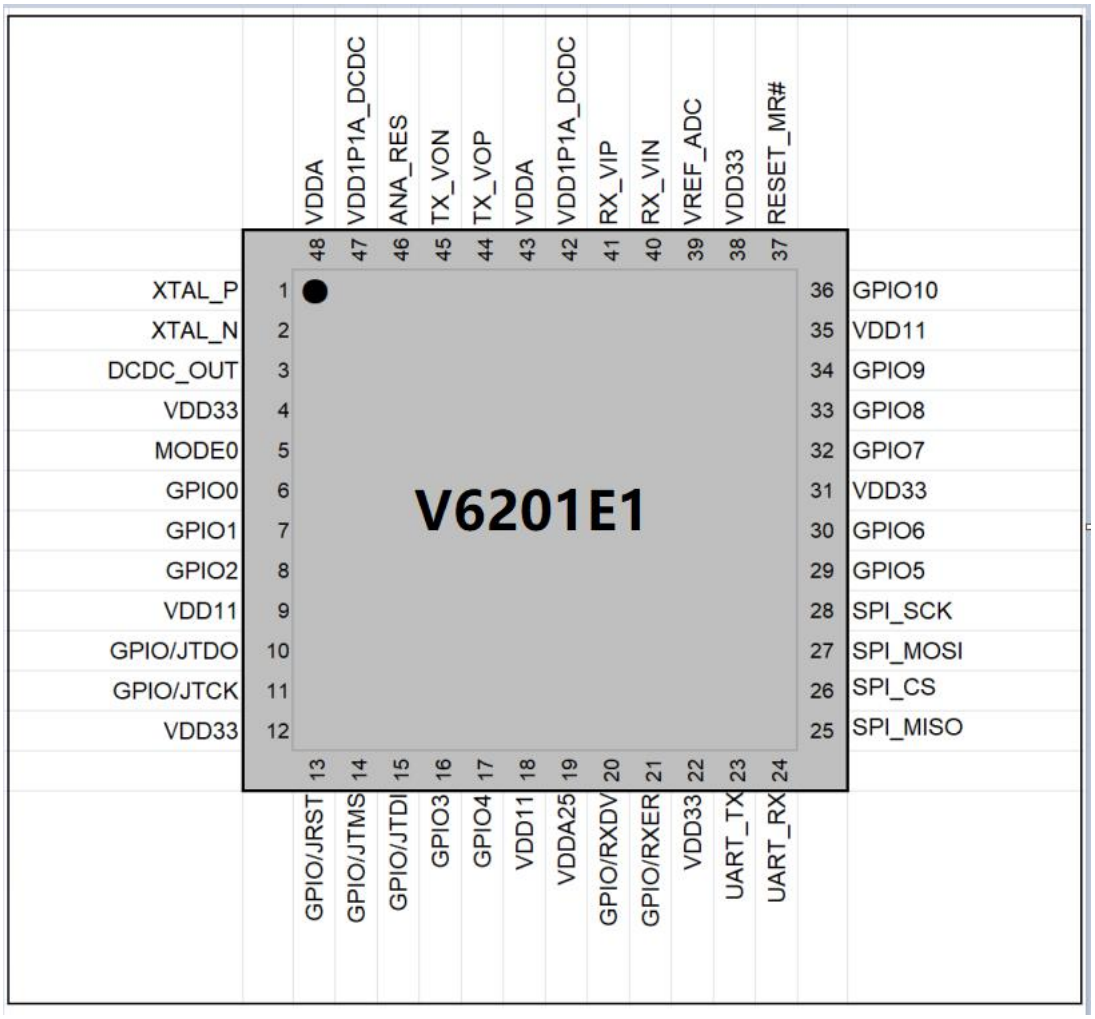
DYNAMIC 为 CPU 正常工作，同时启动物理层接收

表格 3：芯片功耗参数

名称	3.3V 单电源供电功耗	双电源供电功耗	UNIT
芯片启动			mW
物理层接收			mW

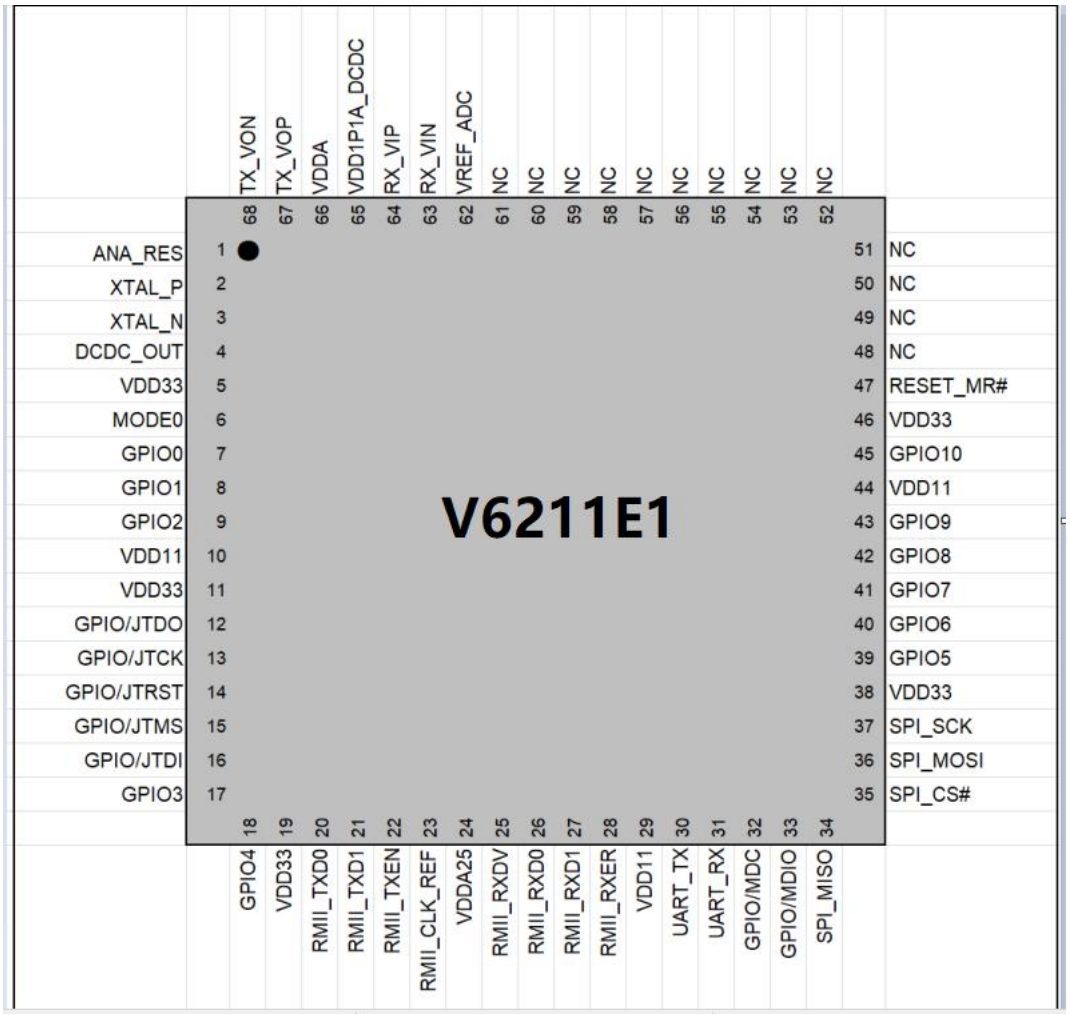
2.引脚配置

2.1. V6201E1 引脚配置



图表 1：V6201E1 引脚图

2.2. V6211E1 引脚配置



图表 2：V6211E1 引脚图

2.3. 引脚说明

引脚名后缀说明：

SIGNAL	高电平有效
SIGNAL#	低电平有效

引脚类型缩写说明：

I	输入
ICLK	输入时钟
O	输出
OCLK	输出时钟
IO	双向信号，上下拉可配置
IO5VT	双向信号，上下拉可配置，且支持 5V 输入
IOOD	双向信号，上下拉可配置，且支持开漏输出
IA	模拟输入
OA	模拟输出
IOA	模拟输入输出
PWR	电源
PWRA	模拟电源
GND	电源地
GNDA	模拟电源地
NC	无连接，统一接地

2.3.1. 电源相关接口

表格 4：电源接口引脚

信号名	V6201E1 ID	V6211E1 ID	类型	说明
DCDC_OUT	3	4	PWR	内部 DCDC 输出 1.1V±0.1V
VDD11	9,18,35	10,29,44	PWR	数字电路 1.1V±0.1V 供电
VDD33	4,12, 22,31,38	5,11, 19,38,46	PWR	芯片 3.3V±0.2V 供电
VDDA25	19	24	PWR	EFUSE 2.5V 供电

VDDA	43,48	66	PWRA	模拟电路电源 3.3V±0.2V
VDD1P1A_DCDC	42,47	65	PWRA	模拟电路 1.1V 电源
VSSA	EPAD	EPAD	GNDA	模拟地
VSS	EPAD	EPAD	GND	数字地

2.3.2. 工厂测试模式控制

表格 5：工作模式设定引脚

信号名	V6201E1 ID	V6211E1 ID	类型	说明
MODE0	5	6	I	0：芯片常规功能模式 1：芯片工厂测试模式 NOTE：此管脚默认内部下拉

2.3.3. JTAG 端口

表格 6：JTAG 引脚

信号名	V6201E1 ID	V6211E1 ID	类型	说明
JTDO	10	12	IO	JTAG 数据输出 GPIO 编号：5
JTDI	15	16	IO	JTAG 数据输入 GPIO 编号：4
JTCK	11	13	IO	JTAG 时钟端口 GPIO 编号：6
JTMS	14	15	IO5VT	JTAG 测试模式选择 GPIO 编号：3
JTRST	13	14	IO	JTAG 复位信号 GPIO 编号：2 **NOTE**：上电默认输入下拉。

2.3.4. 时钟复位控制

表格 7：时钟复位引脚

信号名	V6201E1 ID	V6211E1 ID	类型	说明
RESET_MR#	37	47	I	外部手动复位，低电平有效
XTAL_P	1	2	ICLK	25MHz 晶振输入，或者外部 25MHz 时钟输入
XTAL_N	2	3	OCLK	25MHz 晶振输出，如果连接外部 25MHz 时钟，此引脚无需连接

2.3.5. 通用 GPIO

表格 8：通用 GPIO 引脚

信号名	V6201E1 ID	V6211E1 ID	类型	说明
GPIO0	6	7	IO	通用 IO 端口 GPIO 编号：0 **NOTE**：上电默认输入下拉。
GPIO1	7	8	IO	通用 IO 端口；GPIO 编号：1
GPIO2	8	9	IO	通用 IO 端口 GPIO 编号：7
GPIO3	16	17	IO	通用 IO 端口 GPIO 编号：12 **NOTE**：上电默认输出 25MHz 时钟
GPIO4	17	18	IO	通用 IO 端口 GPIO 编号：17
GPIO5	29	39	IO5VT	通用 IO 端口 GPIO 编号：18 **NOTE**：烧写 EFUSE 时用于输出正确性检测信号。
GPIO6	30	40	IO5VT	通用 IO 端口 GPIO 编号：19
GPIO7	32	41	IO	通用 IO 端口 GPIO 编号：28
GPIO8	33	42	IO	通用 IO 端口 GPIO 编号：29 **NOTE**：烧写 EFUSE 时用于输出控制 VDDA25 的开关信号。
GPIO9	34	43	IOOD	通用 IO 端口 GPIO 编号：30
GPIO10	36	45	IOOD	通用 IO 端口 GPIO 编号：31

注：除了 GPIO3 上电后默认为输出晶振时钟，其他 GPIO 上电复位后均为输入状态。

2.3.6. 外围接口

表格 9：外围接口引脚

信号名	V6201E1 ID	V6211E1 ID	类型	说明
SPI_MOSI	27	36	IO	SPI master 数据输出 GPIO 编号：24
SPI_CS#	26	35	IO	SPI master 片选信号 GPIO 编号：25
SPI_MISO	25	34	IO	SPI master 数据输入 GPIO 编号：26
SPI_SCK	28	37	IO	SPI master 时钟输出 GPIO 编号：27
UART_RX	24	31	IO	UART 数据接收端口 GPIO 编号：21
UART_TX	23	30	IO	UART 数据发送端口 GPIO 编号：20

2.3.7. 网络接口

表格 10：网络接口引脚

信号名	V6201E1 ID	V6211E1 ID	类型	说明
RMII_TXD0	NC	20	IO	发送数据 0 GPIO 编号：9
RMII_TXD1	NC	21	IO	发送数据 1 GPIO 编号：8
RMII_TXEN	NC	22	IO	发送使能 GPIO 编号：10
RMII_CLK_REF	NC	23	IO	RMII 随路时钟，50MHz 输出 GPIO 编号：11
RMII_RXDV	20	25	IO5VT	接收使能 GPIO 编号：13
RMII_RXD0	NC	26	IO	接收数据 0 GPIO 编号：14
RMII_RXD1	NC	27	IO	接收数据 1

				GPIO 编号：15
RMII_RXER	21	28	IO	接收错误标志 GPIO 编号：16
MDC	NC	32	IO	MDIO 时钟输出 GPIO 编号：22
MDIO	NC	33	IO	MDIO 数据信号 GPIO 编号：23

2.3.8. 模拟接口

表格 11：模拟接口引脚

信号名	V6201E1 ID	V6211E1 ID	类型	说明
RX_VIP	41	64	IA	模拟接收
RX_VIN	40	63	IA	模拟接收
TX_VOP	44	67	OA	模拟发送
TX_VON	45	68	OA	模拟发送
VREF_ADC	39	62	IOA	模拟参考电压
ANA_RES	46	1	IOA	对地连接 10KΩ电阻
NC	NC	48-61	NC	空 pin，接地处理

3. 电源及复位

3.1. 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 及内部 DCDC 电源（VDD33）。1.1V 电源（VDD11）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或者使用

3.2. 复位系统

芯片内部的 DCDC 进行供给。

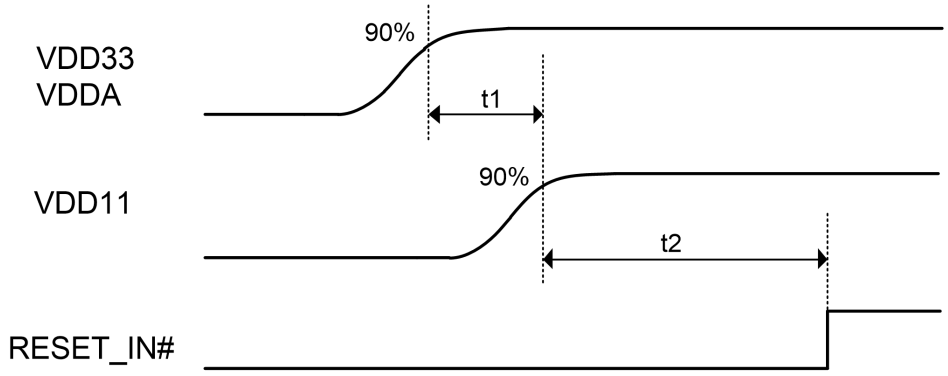
芯片支持如下 6 种复位：

- 上电复位
- 外部引脚复位（RESET_MR#）
- 软件复位
- 看门狗复位
- 欠压复位
- 模拟复位

3.3. 芯片上电时序

芯片支持记录上次系统复位的复位源，详细内容请参考表格 15：全局复位控制寄存器。

芯片的 3.3V、1.1V、RESET 信号上电过程时序如下图所示：



图表 3：芯片上电及复位时序

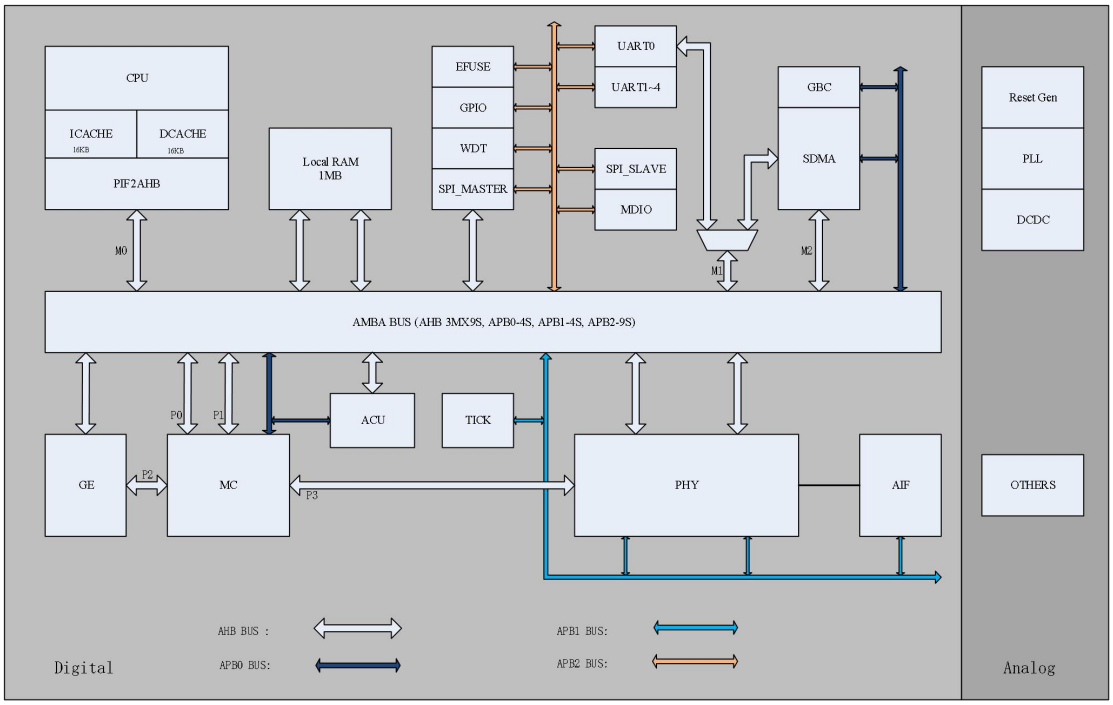
表格 12：上电及复位时间参数

名称	描述	MIN	MAX	UNIT
t1	3.3V 到 1.1V 上电延时	10	-	us

t2	电源稳定后复位信号持续时间	21	-	ms

4. 系统结构

芯片由数字和模拟两部分组成，系统结构图如下：



图表 4：SOC 系统结构

4.1. 数字功能模块

数字电路主要由如下几个模块组成：

- 嵌入式 CPU，最高频率 400MHz
- 双端口 SRAM，1MB，工作于 AHB 时钟频率
- 片上 AHB、APB 总线，最高频率 100MHz
- SDRAM 内存控制器（MC），最高频率 200MHz
- RMII MAC（GE），最高频率 100MHz
- 通用加解密引擎（ACU）
- 8 通道 DMA
- PLC 功能 MAC 及 PHY
- TICK 功能，此模块可提供 4 组 timer 功能（可用于产生 PWM）
- EFUSE 支持一次性写入唯一的 CHIP ID
- SPI master，系统从此端口读取 FLASH 并启动，系统启动后可将此端口用作通用 SPI MASTER 功能
- GPIO 功能
- 看门狗 WDT
- 多个 UART 接口，其中 UART0 支持 master debug 功能

- SPI SLAVE

4.2. 模拟功能模块

- MDIO

模拟电路主要由以下模块组成：

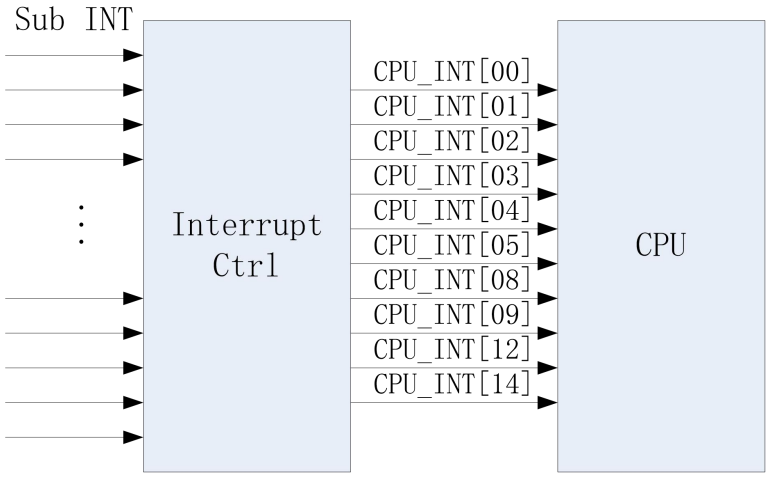
- RESET 控制电路
- 3.3V 到 1.1V DCDC 功能
- PLL 电路等

4.3. 嵌入式 CPU

本产品使用 tensilica Xtensa ® LX4 高性能低功耗的 CPU。CPU 的具体使用方法请参考对应的 CPU 使用说明文档。

4.4. 中断控制系统

芯片中断控制系统如下图所示：



图表 5：中断控制框架

SOC 子模块将各自的中断送入到中断控制模块中（中断控制模块集成在 GBC 模块中），中断控制模块将子模块的中断处理后送入到 CPU 中。软件可以通过不同的寄存器配置来使能不同子模块的中断。可以通过配置将子模块的中断映射到不同的 CPU 中断上以方便软件操作。10 个 CPU 中断拥有 5 个不同的中断优先级。详细配置请参考 GBC 模块说明。

SOC 支持如下模块的中断：

- TICK timer 中断
- GE、GEI、MDIO 模块中断
- PHY 相关中断
- ACU 中断
- UART、SPI_SLAVE、SPI_MASTER 中断
- SDMA 中断

4.4.1. 中断处理

- GPIO 外部中断（全部通用 GPIO 均可使能中断功能）

芯片内部的中断处理流程如下：

1) 子模块发起中断，拉起对应的中断信号
2) 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
3) CPU 接收到中断，并触发对应的中断处理函数

4.5. 芯片内部寻址空间

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

表格 13：芯片寻址空间

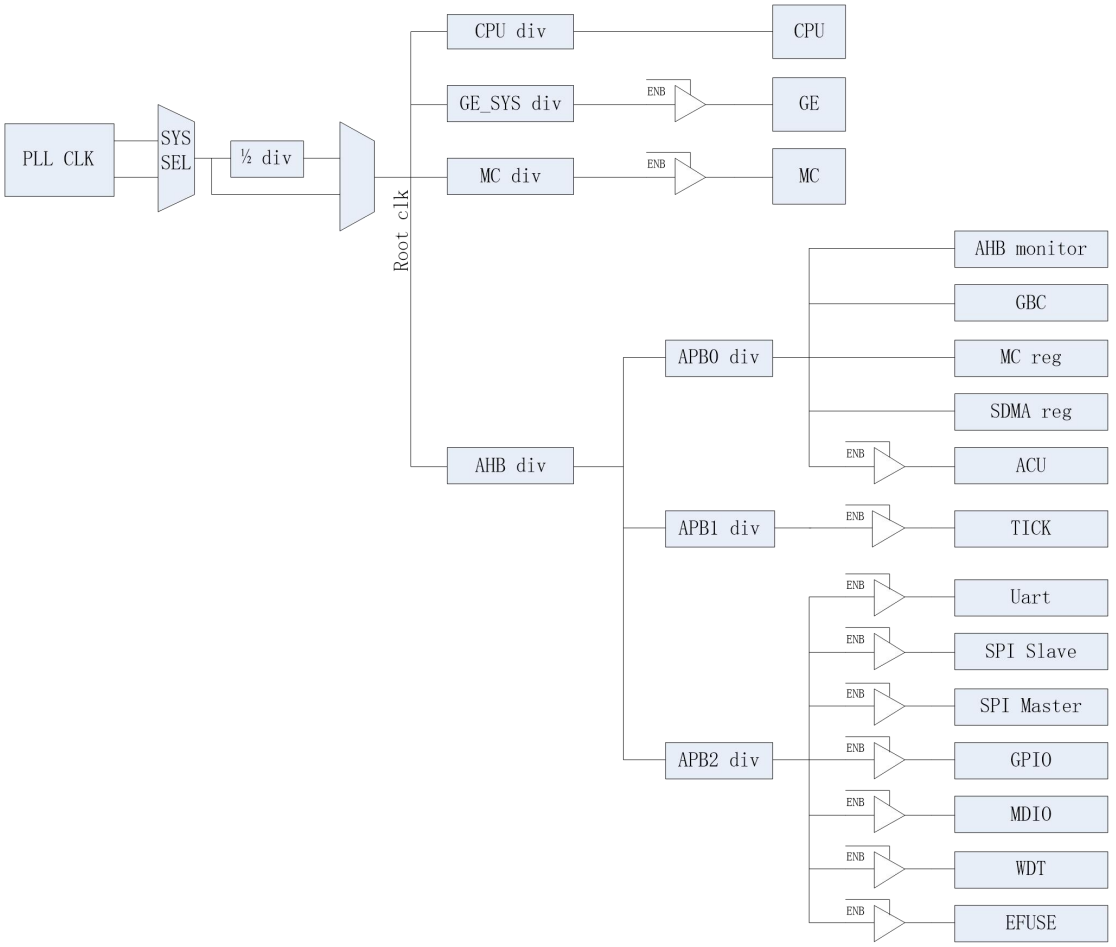
功能	起始地址	结束地址	最大空间
BOOT Address	0xFE000000	0xFEFFFFFF	16MByte
Local RAM Port0	0x10000000	0x1FFFFFFF	256MByte
Local RAM Port1	0x20000000	0x2FFFFFFF	256MByte
GE Memory and Registers	0xD0000000	0xD0FFFFFF	16MByte
ACU Memory	0xD1000000	0xD1FFFFFF	16MByte
PMA Memory	0xD2000000	0xD2FFFFFF	16MByte
PMD Memory	0xD3000000	0xD3FFFFFF	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
SPI	0xF2000000	0xF20FFFFF	1MByte
GPIO	0xF2100000	0xF21FFFFF	1MByte

SPI Slave	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
UART4	0xF2900000	0xF29FFFFF	1MByte
EFUSE	0xF2A00000	0xF2AFFFFF	1MByte
I2C	0xF2B00000	0xF2BFFFFF	1MByte
SPI1	0xF2C00000	0xF2CFFFFF	1MByte
SPI2	0xF2D00000	0xF2DFFFFF	1MByte

5. 系统时钟

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（配置寄存器请参考表格 14:全局时钟控制寄存器）。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：



图表 6：芯片时钟结构

6. GBC 芯片全局控制

6.1. 全局时钟控制

GBC 是此芯片的全局控制模块，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。

芯片内时钟结构如图表 6：芯片时钟结构图所示。相关寄存器如下表：

表格 14：全局时钟控制寄存器

地址/名称	域	初值	类型	说明
0xF0000000 SYS_CLK_CFG	[3]: SYS_PLL_SRC	0x0	RW	系统时钟选择 PLL 源 0：SYS_PLL 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 0：OSC (25MHz) 1：PLL
0xF0000004 SYS_CLK_DIV	[31:24]: DEBUG_BAUD_RATE	0x7	RW	UART0 debug 模式下的波特率配置，默认波特率为 115200，在 debug 模式下配置系统时钟分频的时候需要根据分频比重新计算波特率。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 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
0xF000000C	[18:14]: CLK_EN	0x3fff	RW	1:enable; 0:disable

CLK_GATE				[18]: UART4 [17]: UART3 [16]: SPI2 [15]: SPI1 [14]: I2C

6.1.1.时钟配置说明

0xF000000C CLK_GATE	[13:0]: CLK_EN	0x3fff	RW	[13]: EFUSE [12]: TICK [11]: ACU [10]: UART2 [9]: UART1 [8]: UART0 [7]: SPI_SLAVE [6]: MIDO [5]: GPIO [4]: WDT [3]: SPI [2]: PMA [1]: PMD [0]: AIF

- 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

6.2. 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：

表格 15：全局复位控制寄存器

地址/名称	域	初值	类型	说明
0xF0000010 SOFT_RST	[31]: SYS_RST_EN	0x0	RW	写 1 将对芯片进行全局复位
	[27:15]: RST_EN	0x0	RW	1：软复位；0：撤销复位 [27]: UART4 [26]: UART3 [25]: SPI2 [24]: SPI1 [23]: I2C [22]: efuse [21]: tick [20]: ge_cg [19]: ge_mii_rx [18]: ge_mii_tx [17]: mon [16]: acu [15]: uart2
0xF0000010 SOFT_RST	[14:0]: RST_EN	0x0	RW	[14]: uart1 [13]: uart0 [12]: gpio [11]: spi_slave [10]: wdt [9]: mdio [8]: spi [7]: ge [6]: pma_rx [5]: pma_tx [4]: pma [3]: pmd [2]: aif [1]: mc [0]: sdma

0xF0000014 RST_STA	[15]: SFT_GLB_RST_GEN	0x0	RO	软件复位记录
	[14]: WDT_GLB_RST_GEN	0x0	RO	看门口全局复位记录
	[13]: EXT_WDT_RST_GEN	0x0	RO	看门口外部复位记录
	[12]: EXT_RST_GEN	0x1	RO	芯片外部复位记录(RESET_IN 管脚)
	[11]: ANA_PAD_RST_GEN	0x1	RO	模拟复位记录(RESET_MR 管脚)
	[10]: ANA_WDT_RST_GEN	0x0	RO	看门狗模拟复位记录
	[9]: ANA_LP_RST_GEN	0x0	RO	欠压复位记录
	[8]: ANA_POR_RST_GEN	0x1	RO	上电复位记录
	[0]: RST_GEN_STA_CLR	0x0	RW	清除复位记录，完成后需要软件写 0，以便恢复复位记录

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。

6.3. 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：

表格 16：DMA 通道硬件连接寄存器

地址/名称	域	初值	类型	说明
0xF0000020 DRQ_EN	[15:12]: DRQ_MDIO_MGDI	0x8	RW	MDIO 模块 DMA 通道
	[11:8]: DRQ_UART2_RX	0x8	RW	UART2 模块 DMA 通道
	[7:4]: DRQ_UART1_RX	0x8	RW	UART1 模块 DMA 通道
	[3:0]: DRQ_UART0_RX	0x8	RW	UART0 模块 DMA 通道
0xF0000024 DFC_EN	[27:24]: DFC_GE_GEI_TX	0x8	RW	GE 发送流控 DMA 通道
	[23:20]:	0x8	RW	GE 接收流控 DMA 通道

	DFC_GE_GEI_RX			
	[19:16]: DFC_ACU_CRC_RX	0x8	RW	ACU CRC 写入流控 DMA 通道
	[15:12]: DFC_ACU_CRYPT_RX	0x8	RW	ACU 加解密写入流控 DMA 通道
	[11:8]: DFC_ACU_CRYPT_TX	0x8	RW	ACU 加解密读取流控 DMA 通道
	[7:4]: DFC_PMA_PMI_RX	0x8	RW	PMI 接收流控 DMA 通道
	[3:0]: DFC_PMA_PMI_TX	0x8	RW	PMI 发送流控 DMA 通道
0xF0000028 DFC_EN_1	[7:4]: DFC_SPI_SLAVE_RX	0x8	RW	SPI_SLAVE 接收流控 DMA 通道
	[3:0]: DFC_SPI_SLAVE_TX	0x8	RW	SPI_SLAVE 发送流控 DMA 通道

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

6.4. 全局中断控制

芯片中断控制框架见图表 5：中断控制架构。中断控制寄存器如下表所示：

表格 17：全局中断控制寄存器

地址/名称	域	初值	类型	说明
0xF0000030 CINTR_EN	[9:0]: CIE	0x0	RW	CPU 中断使能标识 [9]：中断 14 使能 [8]：中断 12 使能 [7]：中断 9 使能 [6]：中断 8 使能 [5]：中断 5 使能 [4]：中断 4 使能 [3]：中断 3 使能 [2]：中断 2 使能 [1]：中断 1 使能 [0]：中断 0 使能
0xF0000034 CINTR_STA	[9:0]: CIS	0x0	RO	CPU 中断状态标识，对应于 CIE 寄存器
0xF0000040 MINTR_EN	[30]: MIE_UART4	0x0	RW	UART4 全局中断使能
	[29]: MIE_UART3	0x0	RW	UART3 全局中断使能
	[28]: MIE_SPI2	0x0	RW	SPI2 全局中断使能

	[27]: MIE_SPI1	0x0	RW	SPI1 全局中断使能
	[26]: MIE_I2C	0x0	RW	I2C 全局中断使能
0xF0000040 MINTR_EN	[25]: MIE_HTMR	0x0	RW	TICK TIMER 全局中断使能
	[24]: MIE_GE_GEI	0x0	RW	GEI 全局中断使能
	[23]: MIE_PMA_PMI	0x0	RW	PMI 全局中断使能
	[22]: MIE_ACU	0x0	RW	ACU 全局中断使能
	[21]: MIE_UART2	0x0	RW	UART2 全局中断使能
	[20]: MIE_UART1	0x0	RW	UART1 全局中断使能
	[19]: MIE_MDIO_MGDI	0x0	RW	MDIO 全局中断使能
	[18]: MIE_GE	0x0	RW	GE 全局中断使能
	[17]: MIE_SPI_SLAVE	0x0	RW	SPI_SLAVE 全局中断使能
	[16]: MIE_SDMA_CH7	0x0	RW	SDMA 通道 7 全局中断使能
	[15]: MIE_SDMA_CH6	0x0	RW	SDMA 通道 6 全局中断使能
	[14]: MIE_SDMA_CH5	0x0	RW	SDMA 通道 5 全局中断使能
	[13]: MIE_SDMA_CH4	0x0	RW	SDMA 通道 4 全局中断使能
	[12]: MIE_SDMA_CH3	0x0	RW	SDMA 通道 3 全局中断使能
	[11]: MIE_SDMA_CH2	0x0	RW	SDMA 通道 2 全局中断使能
	[10]: MIE_SDMA_CH1	0x0	RW	SDMA 通道 1 全局中断使能
	[9]: MIE_SDMA_CH0	0x0	RW	SDMA 通道 0 全局中断使能
	[8]: MIE_UART0	0x0	RW	UART0 全局中断使能
	[7]: MIE_GPIO_EXT	0x0	RW	GPIO 外部中断全局使能
	[6]: MIE_SPI	0x0	RW	SPI_MASTER 全局中断使能
	[5]: MIE_MC	0x0	RW	MC 全局中断使能
	[4]: MIE_PMA_ERR	0x0	RW	PMA 错误全局中断使能
	[3]: MIE_PMA_HDR	0x0	RW	PMA Header 全局中断使能
	[2]: MIE_PMD_ERR	0x0	RW	PMD 错误全局中断使能
	[1]: MIE_TICK	0x0	RW	TICK 全局中断使能
	[0]: MIE_PMD_PHY	0x0	RW	PMD 物理层全局中断使能
0xF0000044 MINTR_STA	[30]: MIS_UART4	0x0	RO	UART4 全局中断状态
	[29]: MIS_UART3	0x0	RO	UART3 全局中断状态
	[28]: MIS_SPI2	0x0	RO	SPI2 全局中断状态
	[27]: MIS_SPI1	0x0	RO	SPI1 全局中断状态
	[26]: MIS_I2C	0x0	RO	I2C 全局中断状态
	[25]: MIS_HTMR	0x0	RO	TICK TIMER 全局中断状态

	[24]: MIS_GE_GEI	0x0	RO	GEI 全局中断状态
	[23]: MIS_PMA_PMI	0x0	RO	PMI 全局中断状态
	[22]: MIS_ACU	0x0	RO	ACU 全局中断状态
	[21]: MIS_UART2	0x0	RO	UART2 全局中断状态
	[20]: MIS_UART1	0x0	RO	UART1 全局中断状态
	[19]: MIS_MDIO_MGDI	0x0	RO	MDIO 全局中断状态
	[18]: MIS_GE	0x0	RO	GE 全局中断状态
	[17]: MIS_SPI_SLAVE	0x0	RO	SPI_SLAVE 全局中断状态
	[16]: MIS_SDMA_CH7	0x0	RO	SDMA 通道 7 全局中断状态
	[15]: MIS_SDMA_CH6	0x0	RO	SDMA 通道 6 全局中断状态
	[14]: MIS_SDMA_CH5	0x0	RO	SDMA 通道 5 全局中断状态
0xF0000044 MINTR_STA	[13]: MIS_SDMA_CH4	0x0	RO	SDMA 通道 4 全局中断状态
	[12]: MIS_SDMA_CH3	0x0	RO	SDMA 通道 3 全局中断状态
	[11]: MIS_SDMA_CH2	0x0	RO	SDMA 通道 2 全局中断状态
	[10]: MIS_SDMA_CH1	0x0	RO	SDMA 通道 1 全局中断状态
	[9]: MIS_SDMA_CH0	0x0	RO	SDMA 通道 0 全局中断状态
	[8]: MIS_UART0	0x0	RO	UART0 全局中断状态
	[7]: MIS_GPIO_EXT	0x0	RO	GPIO 外部中断全局状态
	[6]: MIS_SPI	0x0	RO	SPI_MASTER 全局中断状态
	[5]: MIS_MC	0x0	RO	MC 全局中断状态
	[4]: MIS_PMA_ERR	0x0	RO	PMA 错误全局中断状态
	[3]: MIS_PMA_HDR	0x0	RO	PMA Header 全局中断状态
	[2]: MIS_PMD_ERR	0x0	RO	PMD 错误全局中断状态
	[1]: MIS_TICK	0x0	RO	TICK 全局中断状态
	[0]: MIS_PMD_PHY	0x0	RO	PMD 物理层全局中断状态
0xF0000048 MINTR_CLR	[30]: MIC_UART4	0x0	A0	写 1 清理对应的全局中断状态
	[29]: MIC_UART3	0x0	A0	
	[28]: MIC_SPI2	0x0	A0	
	[27]: MIC_SPI1	0x0	A0	
	[26]: MIC_I2C	0x0	A0	
	[25]: MIC_HTMR	0x0	A0	
	[24]: MIC_GE_GEI	0x0	A0	
	[23]: MIC_PMA_PMI	0x0	A0	
	[22]: MIC_ACU	0x0	A0	
	[21]: MIC_UART2	0x0	A0	

	[20]: MIC_UART1	0x0	A0	
	[19]: MIC_MDIO_MGDI	0x0	A0	
	[18]: MIC_GE	0x0	A0	
	[17]: MIC_SPI_SLAVE	0x0	A0	
	[16]: MIC_SDMA_CH7	0x0	A0	
	[15]: MIC_SDMA_CH6	0x0	A0	
	[14]: MIC_SDMA_CH5	0x0	A0	
	[13]: MIC_SDMA_CH4	0x0	A0	
	[12]: MIC_SDMA_CH3	0x0	A0	
	[11]: MIC_SDMA_CH2	0x0	A0	
	[10]: MIC_SDMA_CH1	0x0	A0	
	[9]: MIC_SDMA_CH0	0x0	A0	
	[8]: MIC_UART0	0x0	A0	
	[7]: MIC_GPIO_EXT	0x0	A0	
	[6]: MIC_SPI	0x0	A0	
	[5]: MIC_MC	0x0	A0	
	[4]: MIC_PMA_ERR	0x0	A0	
	[3]: MIC_PMA_HDR	0x0	A0	
	[2]: MIC_PMD_ERR	0x0	A0	
0xF0000048 MINTR_CLR	[1]: MIC_TICK	0x0	A0	写 1 清理对应的全局中断状态
	[0]: MIC_PMD_PHY	0x0	A0	
0xF0000050 MINRT_PRI0	[31:28]: MIP_GPIO_EXT	0x0	RW	各个子模块中断到 CPU 中断的映射配置
	[27:24]: MIP_SPI	0x0	RW	
	[23:20]: MIP_MC	0x0	RW	
	[19:16]: MIP_PMA_ERR	0x0	RW	
	[15:12]: MIP_PMA_HDR	0x0	RW	
	[11:8]: MIP_PMD_ERR	0x0	RW	
	[7:4]: MIP_TICK	0x0	RW	
	[3:0]: MIP_PMD_PHY	0x0	RW	
0xF0000054 MINRT_PRI1	[31:28]: MIP_SDMA_CH6	0x0	RW	
	[27:24]: MIP_SDMA_CH5	0x0	RW	
	[23:20]: MIP_SDMA_CH4	0x0	RW	
	[19:16]: MIP_SDMA_CH3	0x0	RW	
	[15:12]: MIP_SDMA_CH2	0x0	RW	
	[11:8]: MIP_SDMA_CH1	0x0	RW	

	[7:4]: MIP_SDMA_CH0	0x0	RW	
	[3:0]: MIP_UART0	0x0	RW	
0xF0000058 MINRT_PRI2	[31:28]: MIP_PMA_PMI	0x0	RW	
	[27:24]: MIP_ACU	0x0	RW	
	[23:20]: MIP_UART2	0x0	RW	
	[19:16]: MIP_UART1	0x0	RW	
	[15:12]: MIP_MDIO_MGDI	0x0	RW	
	[11:8]: MIP_GE	0x0	RW	
	[7:4]: MIP_SPI_SLAVE	0x0	RW	
	[3:0]: MIP_SDMA_CH7	0x0	RW	
0xF000005C MINRT_PRI3	[27:24]: MIP_UART4	0x0	RW	
	[23:20]: MIP_UART3	0x0	RW	
	[19:16]: MIP_SPI2	0x0	RW	
	[15:12]: MIP_SPI1	0x0	RW	
	[11:8]: MIP_I2C	0x0	RW	
	[7:4]: MIP_HTMR	0x0	RW	
	[3:0]: MIP_GE_GEI	0x0	RW	

不同 CPU 中断号对应于不同的中断优先级。具体关系如下：

表格 18：CPU 中断优先级关系

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
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

6.5. 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：

表格 19：总线控制寄存器

地址/名称	域	初值	类型	说明
0xF0000060 AHB_ERR_EN	[0]: AHB_ERR_EN	0x0	RW	AHB 总线错误检测使能
0xF0000064 AHB_TIMEOUT	[31:0]: AHB_TIMEOUT	0x1000000 0	RW	AHB 总线超时周期，单位 AHB 时钟
0xF0000068 LRAM_BANK_PRI	[1:0]: LRAM_BANK_PRI	0x0	RW	[1]: 1: BANK1 端口 1 优先级高于端口 0 [0]: 1: BANK0 端口 1 优先级高于端口 0

7. 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。

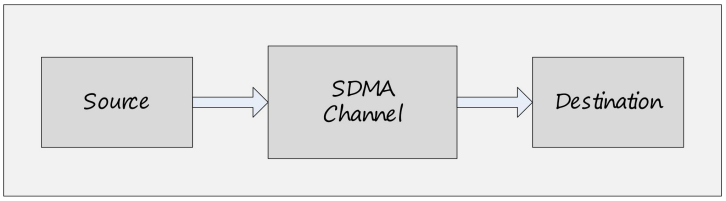
SDMA 支持如下特性：

- 双 AHB master 接口，接口连接方式请参考图表 4：SOC 系统结构。
- 8 条独立通道，可并行工作
- 每个通道可独立配置使用几个 AHB master 接口
- 每个通道包含 32 个 word 缓冲区，支持水线配置
- 支持 byte、half_word、word 传输
- 支持自动切分数据总长到多个总线 burst
- 支持通道优先级配置
- 支持地址递增、锁定、跳变、环回模式
- 支持硬件握手模式
- 支持通道级联功能
- 支持自动重载通道配置
- 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

7.1. SDMA 通道传输控制

- 支持 DMA 通道错误检测

SDMA 通道典型传输框图如下所示：



图表 7：SDMA 通道传输示意图

各通道 DMA 功能寄存器如下表所示：

表格 20：DMA 通道功能寄存器

地址/名称	域	初值	类型	说明
0xF0100X00 SA	[31:0]: SA	0x0	RW	源数据地址，读取值为当前正在搬移的源数据地址
0xF0100X04 DA	[31:0]: DA	0x0	RW	目的数据地址，读取值为当前正在搬移的目的数据地址

0xF0100X08 SOA	[31:0]: SOA	0x0	RW	特定模式源地址附加配置： 跳变模式下，高 16 位表示块大小，低 16 位表示偏移量 环回模式下，低 16 位表示环回块大小
0xF0100X08 DOA	[31:0]: DOA	0x0	RW	特定模式目的地址附加配置： 跳变模式下，高 16 位表示块大小，低 16 位表示偏移量 环回模式下，低 16 位表示环回块大小
0xF0100X20 LEN	[15:0]: LEN	0x0	RW	DMA 传输数据长度，读取值为本次传输剩余长度
0xF0100X24 CTRL	[29]: DFC_EN	0x0	RW	目的端硬件流控使能
	[28]: SFC_EN	0x0	RW	源端硬件流控使能
	[26]: RELOAD_LEN	0x0	RW	完成后自动重载 LEN 寄存器
	[25]: RELOAD_DA	0x0	RW	完成后自动重载 DA 寄存器
	[24]: RELOAD_SA	0x0	RW	完成后自动重载 SA 寄存器
	[23:20]: SW_HS	0x0	A0	软件握手模式 DMA 指令： 4'b0000：无效 4'b0001：通道启动 4'b0010：通道暂停 4'b0100：强制停止改通道
	[19]: PAUSER	0x0	RW	可暂停通道端口选择 0：源端 1：目的端
	[18]: STOPPER	0x0	RW	可停止通道端口选择 0：源端 1：目的端
	[17:16]: HS_MODE	0x0	RW	握手模式选择 0：软件模式 1：硬件模式 2：软硬件混合模式
	[15:12]: DA_MODE	0x0	RW	目的地址产生模式 0：递增 1：固定 2：跳变

				3：环回
	[11:8]: SA_MODE	0x0	RW	源地址产生模式 0：递增 1：固定 2：跳变 3：环回
	[7]: CHAN_EN	0x0	RW	通道级联使能
	[6:4]: CHAN_TO	0x0	RW	上一级通道号
	[3:2]: PT_CFG	0x0	RW	源端和目的端端口配置 0：AHB0->源端; AHB1->目的端 1：AHB1->源端; AHB0->目的端 2：AHB0->源端; AHB0->目的端 3：AHB1->源端; AHB1->目的端
0xF0100X24 CTRL	[1]: CH_RST	0x0	A0	通道复位
	[0]: CH_EN	0x0	RW	通道使能，通道不使用的时候请关闭该通道
0xF0100X28 STA	[15:8]: ERR_STA	0x0	RO	通道错误状态 8'b0000_0000：无错误 8'bxxxx_xxx1：源端口错误 8'bxxxx_xx1x：目的端口错误 8'bxxxx_x1xx：软件握手错误 8'bxxxx_1xxx：硬件握手错误 8'bxxx1_xxxx：FIFO 错误 8'bxx1x_xxxx：超时错误
	[3:0]: CH_STA	0x0	RO	通道状态 4'b0000：当前无传输操作 4'b0001：正在传输 4'b0010：传输暂停
0xF0100X40 IE	[8]: IE_TRANS_ERR	0x0	RW	通道错误中断
	[4]: IE_TRANS_DONE	0x0	RW	通道传输结束中断
	[3]: IE_BLK_DONE	0x0	RW	通道传输块传输结束中断
	[2]: IE_STOP	0x0	RW	通道停止中断
	[1]: IE_PAUSE	0x0	RW	通道暂停中断

	[0]: IE_START	0x0	RW	通道启动中断
0xF0100X44 IS	[8]: IS_TRANS_ERR	0x0	RC	通道错误中断状态
	[4]: IS_TRANS_DONE	0x0	RC	通道传输结束中断状态
	[3]: IS_BLK_DONE	0x0	RC	通道传输块传输结束中断状态
	[2]: IS_STOP	0x0	RC	通道停止中断状态
	[1]: IS_PAUSE	0x0	RC	通道暂停中断状态
	[0]: IS_START	0x0	RC	通道启动中断状态
0xF0100X64 TO	[31:0]: TIMEOUT	0x10 0000	RW	通道传输超时计数
0xF0100X68 BUF	[19:16]: WT_WM	0x4	RW	目的端写出 FIFO 水线控制 0：只要有数据就启动写出 1：1/8 深度以上启动写出 2：2/8 深度以上启动写出 3：3/8 深度以上启动写出 4：4/8 深度以上启动写出 5：5/8 深度以上启动写出 6：6/8 深度以上启动写出 7：7/8 深度以上启动写出
	[11:8]: RD_WM	0x4	RW	源端读入 FIFO 水线控制 0：只要有空间就启动读入 1：1/8 深度以下启动读入 2：2/8 深度以下启动读入 3：3/8 深度以下启动读入 4：4/8 深度以下启动读入
0xF0100X68 BUF	[11:8]: RD_WM	0x4	RW	5：5/8 深度以下启动读入 6：6/8 深度以下启动读入 7：7/8 深度以下启动读入
	[3:0]: BLK_SIZE	0x5	RW	单次总线 burst 传输最大长度 0：burst_block_size = LEN other：burst_block_size = 2^BLK_SIZE Bytes

7.1.1. 通道地址及长度控制

注：寄存器地址中的 X 为 0 到 7，分别代表 8 条通道地址

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：

- 递增：按总线位宽实际操作地址递增
- 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
- 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
- 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

7.1.2. 握手模式

的配置，然后结束通道传输

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。

7.1.3. 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。

7.2. SDMA 通用配置

7.2.1. SDMA 通道优先级

SDMA 8 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：

表格 21：通道优先级控制寄存器

地址/名称	域	初值	类型	说明
0xF0100X60 ARB	[15]: CH_ARB_FIX	0x0	RW	通道优先级方案调整
	[10:8]: PRI	0x0	RW	通道优先级，值越大优先级越高
	[1]: CH_ARB	0x0	RW	2'b00：同等优先级下，不同通道按总线 burst block 切换 2'b01：锁定总线仲裁器，一次
	[0]: BLK_ARB	0x0	RW	

				完成整个通道传输 2'b1x：允许 word 级别，总线优先级切换

注：寄存器地址中的 X 为 0 到 7，分别代表 8 条通道地址

8. 存储控制

芯片存储主要由两部分组成，MC 控制 SDRAM 存储颗粒，提供大容量的存储。LRAM 片内 RAM 提供 1MB 的高速存储空间。

8.1. MC 内存控制器

MC 是一个通用的 SDRAM 内存控制器，V6201E1 无内封 SDRAM，V6211E1 内封 8MB SDRAM。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图表 4：SOC 系统结构。

MC 基础特征：

- 最高工作于 200MHz
- 支持 1MB 到 16MB SDRAM
- 支持 SDRAM 低功耗功能
- 支持调整 SDRAM 刷新周期，最小 0.976us 最大 125us
- 支持 2Bank 或 4Bank SDRAM
- CAS 支持 1 到 3
- 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。

8.2. LRAM 片内存储

LRAM 片内存储，直接连接到 AMBA 总线上，特性如下：

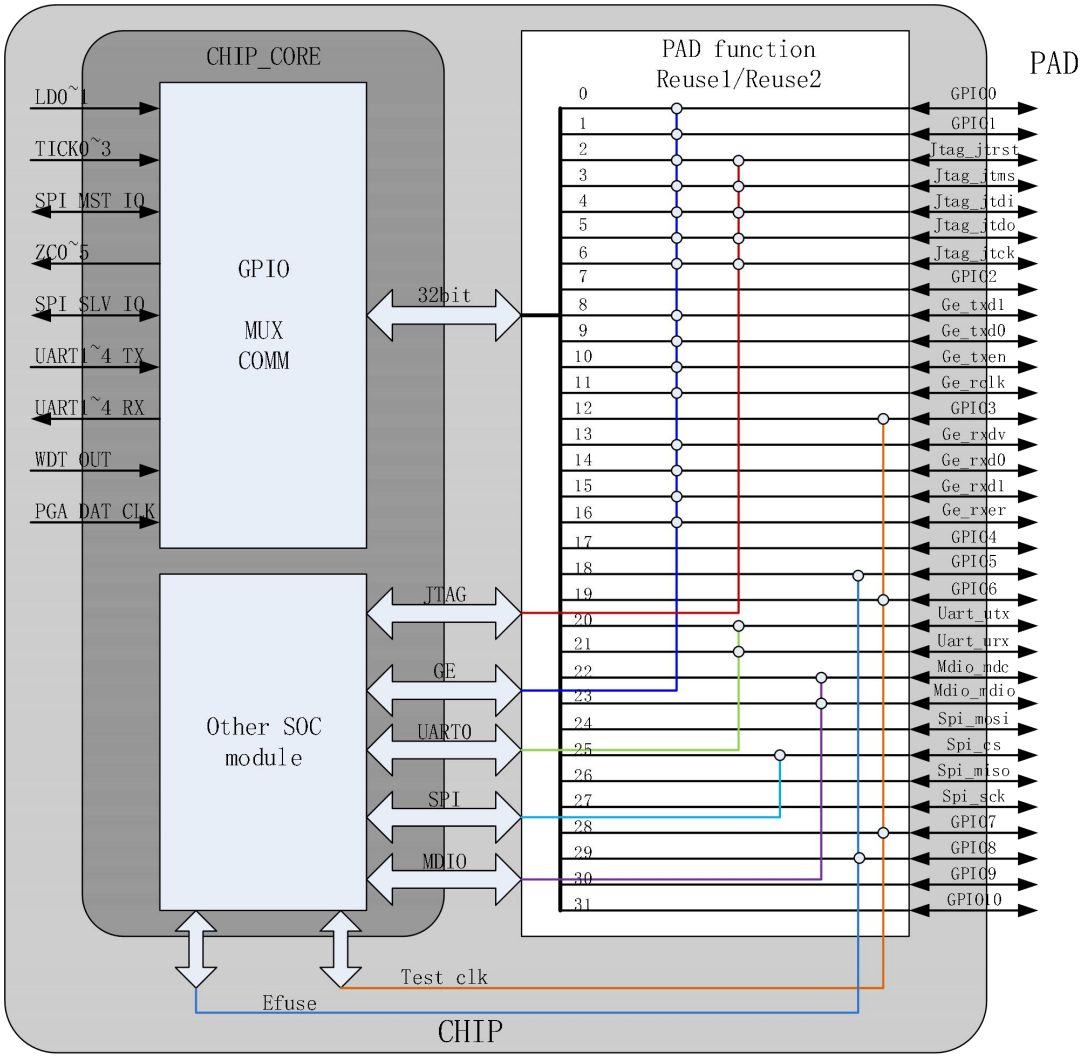
- 由 32 个 32KB 大小的 BANK 组成
- 提供两个通用的访问接口
- 支持同时访问两个 BANK
- 支持 BANK 对访问端口的优先级控制，控制寄存器请参考表格 19：总线控制寄存器。

9. GPIO 通用输入输出

9.1. GPIO PAD 复用

芯片内部供 32 个 GPIO，全部的 GPIO 支持功能复用，支持外部中断。

芯片 GPIO PAD 采用两级复用设计，具体复用图如下：



图表 8：GPIO PAD 复用示意图

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 32 个 GPIO PAD 均为普通的 GPIO 端口。

GPIO PAD 复用配置寄存器如下：

表格 22：GPIO PAD 复用相关寄存器

地址/名称	域	初值	类型	说明
0xF2100050 GPIO_SHARE_EN0	[29:28]: GPIO_SPI_3W	0x0	RW	SPI 3 线模式控制： 2：标准 SPI 4 线模式 3：SPI 3 线模式 other：NA
	[27]: GPIO_GE_MII_EN	0x0	RW	复用 GPIO0、GPIO1、JTAG、GE 端口为 MII、RGMII 接口
	[26]: GPIO_GE_RGMII_EN	0x0	RW	
	[25]: GPIO_ZC5_EN	0x0	RW	过零信号复用使能
	[24]: GPIO_ZC4_EN	0x0	RW	
	[23]: GPIO_ZC3_EN	0x0	RW	
	[22]: GPIO_ZC2_EN	0x0	RW	
	[21]: GPIO_ZC1_EN	0x0	RW	
	[20]: GPIO_ZC0_EN	0x0	RW	
	[19]: GPIO_WDT_EN	0x0	RW	WDT GPIO 输出 IO 复用使能
	[18]: GPIO_MDIO_EN	0x1	RW	MDIO GPIO 复用使能
	[17]: GPIO_JTAG_EN	0x1	RW	JTAG GPIO 复用使能
	[16]: GPIO_GE_EN	0x1	RW	GE 端口复用使能，默认支持 RMII 模式
	[15]: GPIO_TICK3_EN	0x0	RW	TICK IO output 信号复用使能
	[14]: GPIO_TICK2_EN	0x0	RW	
	[13]: GPIO_TICK1_EN	0x0	RW	
	[12]: GPIO_TICK0_EN	0x0	RW	
	[11]: GPIO_LD3_EN	0x0	RW	模拟接口 LD 信号复用使能
	[10]: GPIO_LD2_EN	0x0	RW	
	[9]: GPIO_LD1_EN	0x0	RW	
	[8]: GPIO_LD0_EN	0x0	RW	
	[7]: GPIO_SPI_SLV_EN	0x0	RW	SPI_SLAVE 接口复用使能
	[6]: GPIO_UART2_EN	0x0	RW	UART 端口复用使能
	[5]: GPIO_UART1_EN	0x0	RW	

	[4]: GPIO_UART0_EN	0x1	RW	
	[3]: GPIO_SS3_EN	0x0	RW	SPI_MASTER CS 信号复用使能
	[2]: GPIO_SS2_EN	0x0	RW	
	[1]: GPIO_SS1_EN	0x0	RW	
	[0]: GPIO_SS0_EN	0x1	RW	
0xF2100054 GPIO_SHARE_EN1	[11]: GPIO_2ND_BOOT	0x1	RW	第二片 Flash 启动支持，当 GPIO0 为高时从第二片 Flash 启动芯片
	[10]: GPIO_EFUSE_EN	0x0	RW	EFUSE 信号复用使能
	[9]: GPIO_TCK_W_DAC_EN	0x0	RW	GPIO3 输出 DAC 时钟使能
	[8]: GPIO_TCK_W_ADC_EN	0x0	RW	GPIO3 输出 ADC 时钟使能
	[5]: GPIO_TCK_DAC_EN	0x0	RW	GPIO3 输出 DAC 时钟使能
	[4]: GPIO_TCK_ADC_EN	0x0	RW	GPIO3 输出 ADC 时钟使能
0xF2100054 GPIO_SHARE_EN1	[3]: GPIO_TCK_OSC_EN	0x1	RW	GPIO3 输出 OSC 时钟使能
	[2]: GPIO_TCK_GE_EN	0x0	RW	GPIO7 输出 GE_PLL 时钟使能
	[1]: GPIO_TCK_SYS_EN	0x0	RW	GPIO6 输出 SYS_PLL 时钟使能
	[0]:GPIO_TCK_MC_SYN_EN	0x0	RW	GPIO3 输出 MC SYN 时钟使能
0xF2100058 GPIO_SHARE_EN2	[6]:GPIO_ANT_CTR_EN	0x0	RW	ANT_CTR 信号复用使能
	[5]:GPIO_DFE_PAD_EN	0x0	RW	DFE_PAD 信号复用使能
	[4]:GPIO_SPI1_CS1_EN	0x0		SPI1 CS 信号复用使能
	[3]:GPIO_SPI1_CS0_EN	0x0		
	[2]:GPIO_UART4_EN	0x0		UART4 端口复用使能
	[1]:GPIO_UART3_EN	0x0		UART3 端口复用使能
	[0]:GPIO_I2C_EN	0x0		I2C 端口复用使能
0xF2100060 GPIO_SAMP_0	[28:24]: GPIO_SS3_MAP	0x19	RW	SPI master CS1-CS3 信号复用管脚配置
	[20:16]: GPIO_SS2_MAP	0x19	RW	
	[12:8]:	0x7	RW	

	GPIO_SS1_MAP			
	[4:0]: GPIO_WDT_MAP	0xe	RW	WDT GPIO 输出复用管脚配置
0xF2100064 GPIO_SAMP_1	[28:24]: GPIO_URX2_MAP	0x15	RW	UART1、UART2 复用管脚配置
	[20:16]: GPIO_UTX2_MAP	0x14	RW	
	[12:8]: GPIO_URX1_MAP	0x15	RW	
	[4:0]: GPIO_UTX1_MAP	0x14	RW	
0xF2100068 GPIO_SAMP_2	[29:25]: GPIO_ZC5_MAP	0xc	RW	过零信号复用管脚配置
	[24:20]: GPIO_ZC4_MAP	0xc	RW	
	[19:15]: GPIO_ZC3_MAP	0xc	RW	
	[14:10]: GPIO_ZC2_MAP	0xc	RW	
	[9:5]: GPIO_ZC1_MAP	0xc	RW	
	[4:0]: GPIO_ZC0_MAP	0xc	RW	
0xF210006C GPIO_SAMP_3	[28:24]: GPIO_LD3_MAP	0x1f	RW	LD 控制信号复用管脚配置
	[20:16]: GPIO_LD2_MAP	0x1f	RW	
	[12:8]: GPIO_LD1_MAP	0x1f	RW	
	[4:0]: GPIO_LD0_MAP	0x1f	RW	
0xF2100070 GPIO_SAMP_4	[28:24]: GPIO_TICK3_MAP	0x1f	RW	TICK IO output 信号复用管脚配置
	[20:16]: GPIO_TICK2_MAP	0x1f	RW	
	[12:8]: GPIO_TICK1_MAP	0x1f	RW	
	[4:0]: GPIO_TICK0_MAP	0x1f	RW	
0xF2100074 GPIO_SAMP_5	[28:24]: GPIO_SPI_SLV_MISO_MAP	0xd	RW	SPI SLAVE 相关信号复用管脚配置
	[20:16]: GPIO_SPI_SLV _MOSI_MAP	0xd	RW	
	[12:8]: GPIO_SPI_SLV_CS_MAP	0xd	RW	

	[4:0]: GPIO_ SPI_SLV_SCK_MAP	0xd	RW	
0xF2100078 GPIO_SAMP _6	[20:16]: GPIO_SPI_MISO_MAP	0x1a	RW	SPI MASTER 相关信号复用管脚配置
0xF2100078 GPIO_SAMP _6	[12:8]: GPIO_SPI_MOSI_MAP	0x18	RW	SPI MASTER 相关信号复用管脚配置
	[4:0]: GPIO_SPI_SCK_MAP	0x1b	RW	
0xF210007C GPIO_SAMP _7	[12:8]:GPIO_I2C_SDA _MAP	0x1f	RW	I2C 相关信号复用管脚配置
	[4:0]:GPIO_I2C_SCL_ MAP	0x1f	RW	
0xF2100080 GPIO_SAMP _8	[12:8]:GPIO_URX3_M AP	0x1f	RW	UART3 相关信号复用管脚配置
	[4:0]:GPIO_UTX3_MA P	0x1f	RW	
0xF2100084 GPIO_SAMP _9	[12:8]:GPIO_URX4_M AP	0x1f	RW	UART4 相关信号复用管脚配置
	[4:0]:GPIO_UTX4_MA P	0x1f	RW	
0xF2100088 GPIO_SAMP _10	[12:8]:GPIO_SPI1_CS 1_MAP	0x1f	RW	SPI1 片选信号复用管脚配置
	[4:0]:GPIO_SPI1_CS0 _MAP	0x1f	RW	
0xF210008C GPIO_SAMP _11	[20:16]:GPIO_SPI1_M ISO_MAP	0x1f	RW	SPI1 相关信号复用管脚配置
	[12:8]:GPIO_SPI1_MO SI_MAP	0x1f	RW	
	[4:0]:GPIO_SPI1_SCK _MAP	0x1f	RW	
0xF2100090 GPIO_SAMP _12	[28:24]:GPIO_DFE_PA D_RXDQ_MAP	0x1f	RW	DFE PAD 信号复用管脚配置
	[20:16]:GPIO_DFE_PA D_RXDI_MAP	0x1f	RW	
	[12:8]:GPIO_DFE_PAD _TXDQ_MAP	0x1f	RW	
	[4:0]:GPIO_DFE_PAD_ TXDI_MAP	0x1f	RW	
0xF2100094 GPIO_SAMP _13	[12:8]:GPIO_DFE_PAD _AGC_MAP	0x1f	RW	DFE PAD 相关信号复用管脚配置
	[4:0]:GPIO_DFE_PAD_ TXG_MAP	0x1f	RW	

9.1.1. GPIO PAD 复用实例

0xF2100098 GPIO_SAMP_14	[4:0]:GPIO_ANT_CTR_MAP	0x1f	RW	ANT CTR 相关信号复用管脚配置

实例一：将 JTAG 端口用作通用 GPIO 端口

1) 关闭 GPIO_GE_MII_EN
2) 关闭 GPIO_GE_RGMII_EN
3) 关闭 GPIO_JTAG_EN
4) 解除所有一级复用对 JTAG 端口的复用
5) 按 GPIO 规则使用 JTAG 端口

实例二：将 JTAG 端口复用为 SPI_SLAVE 端口

1) 关闭 GPIO_GE_MII_EN
2) 关闭 GPIO_GE_RGMII_EN
3) 关闭 GPIO_JTAG_EN
4) 解除所有一级复用对 JTAG（GPIO MAP 3、4、5、6）端口的复用
5) 配置 GPIO_SSI_SCK_MAP 为 3，使用 JTMS 作为 SCK 信号
6) 配置 GPIO_SSI_CS_MAP 为 4，使用 JTDI 作为 CS 信号
7) 配置 GPIO_SSI_MISO_MAP 为 5，使用 JTDO 作为 MISO 信号
8) 配置 GPIO_SSI_MOSI_MAP 为 6，使用 JTCK 作为 MOSI 信号
9) 配置 GPIO_SSI_EN 为 1，使能 SPI_SLAVE 端口信号复用
10) 使用 SPI_SLAVE 器件

实例三：使用 GPIO 输出 SYS_PLL 的 50MHz 分频信号

1) 配置 GPIO_TCK_SYS_EN 信号为 1，打开 SYS_PLL 分频输出
2) 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。

9.2. 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：

表格 23：GPIO 输入输出功能寄存器

地址/名称	域	初值	类型	说明
0xF2100000	[31:0]: GPIO_DIN	N/A	RO	GPIO 输入值

GPIO_DIN				
0xF2100004 GPIO_DOUT	[31:0]: GPIO_DOUT	0x0	RW	GPIO 输出值配置
0xF2100008 GPIO_CTRL	[31:0]: GPIO_CTRL	0xffff ffff	RW	GPIO 方向配置 1：输入 0：输出
0xF210000C GPIO_IN_EN	[31:0]: GPIO_IN_EN	0x0	RW	GPIO 强制输入使能，默认情况下不需要配置
0xF21000010 GPIO_PD_EN	[31:0]: GPIO_PD_EN	0x5	RW	GPIO 下拉功能使能，下拉电阻 44KΩ
0xF21000014 GPIO_PU_EN	[31:0]: GPIO_PU_EN	0x0f0 0000 0	RW	GPIO 上拉功能使能，上拉电阻 44KΩ
0xF21000018 GPIO_SMIT_EN	[31:0]: GPIO_SMIT_EN	0x0	RW	GPIO 输入施密特触发器使能
0xF2100001C GPIO_PAD_DRV_0_	[31:0]: GPIO_DRV_0_	0xffffff ff	RW	GPIO 0~15 输出强度调整，每个 pad 占用两个 bit： 0: 2.4mA 1: 4.8mA 2: 7.2mA 3: 9.6mA
0xF21000020 GPIO_PAD_DRV_1_	[31:0]: GPIO_DRV_1_	0xffffff ff	RW	GPIO 16~31 输出强度调整，每个 pad 占用两个 bit： 0: 2.4mA 1: 4.8mA 2: 7.2mA 3: 9.6mA

9.2.1. 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 为输如状态时，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）。

9.2.2. 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）。

9.3. GPIO 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：

表格 24：GPIO 中断寄存器

地址/名称	域	初值	类型	说明
0xF2100020 GPIO_EINT_EN	[31:0]: GPIO_EINT_EN	0x0	RW	GPIO 中断使能
0xF2100024 GPIO_EINT_NEG	[31:0]: GPIO_EINT_NEG	0x0	RW	GPIO 中断触发电平选择 0：高电平 1：低电平
0xF2100028 GPIO_EINT_EDGE	[31:0]: GPIO_EINT_EDGE	0x0	RW	GPIO 中断触发边沿触发使能 1：边沿触发 0：电平触发
0xF210002C GPIO_EINT_STA	[31:0]: GPIO_EINT_STA	0x0	RC	GPIO 中断状态寄存器

实例一：GPIO0 端口上升沿触发

1) 解除 GPIO0 复用
2) 配置 GPIO_EINT_NEG[0]为 1
3) 配置 GPIO_EINT_EDGE[0]为 1
4) 配置 GPIO_EINT_EN[0]为 1

实例二：JTDI 端口下降沿触发

1) 解除 JTDI 端口复用
2) 配置 GPIO_EINT_NEG[4]为 0
3) 配置 GPIO_EINT_EDGE[4]为 1
4) 配置 GPIO_EINT_EN[4]为 1

10. SPI MASTER

芯片内置 SPI MASTER 模块，支持如下特性：

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

SPI master 相关寄存器如下表：

表格 25：SPI master 相关寄存器

地址/名称	域	初值	类型	说明
0xF2000000 INTR_EN	[0]: SPI_INTR_EN	0x0	RW	SPI 中断使能，使能之后，当 SPI 操作结束后会产生中断
0xF2000004 INTR_STA	[0]: SPI_BUSY	0x0	RO	SPI 忙状态
0xF2000010 MODE	[8]: SPI_BOOT_EN	0x1	RW	CPU boot 访问使能
	[4]: SPI_SPC0	0x0	RW	0：标准 SPI 4 线模式 1：SPI 3 线模式
	[2]: SPI_CPHA	0x0	RW	MOSI/MISO 数据极性选择
	[1]: SPI_CPOL	0x0	RW	SCK 时钟极性选择
	[0]: SPI_ENABLE	0x1	RW	SPI master 功能使能
0xF2000014 SCK_CFG	[15:8]: SPI_SCK_LOW_PERIOD	0x1f	RW	SCK 低电平持续 PCLK 数目
	[7:0]: SPI_SCK_HIGH_PERIOD	0x1f	RW	SCK 高电平持续 PCLK 数目 sck = PCLK2 / ((spi_sck_low_period+1) +(spi_sck_high_period+1))
0xF2000018 CSS	[31:24]: SPI_CSS_HCYC	0x3f	RW	SPI CS 低电平开始持续时间
	[16]: SPI_CSS_STOP	0x1	RW	下一个数据传输完成后撤销 CS 信号
	[13]: SPI_CFG_MIMO	0x0	RW	SPI 3 线模式控制 1：手动控制 MOSI 信号方向使能，SPI_MIMO 控制

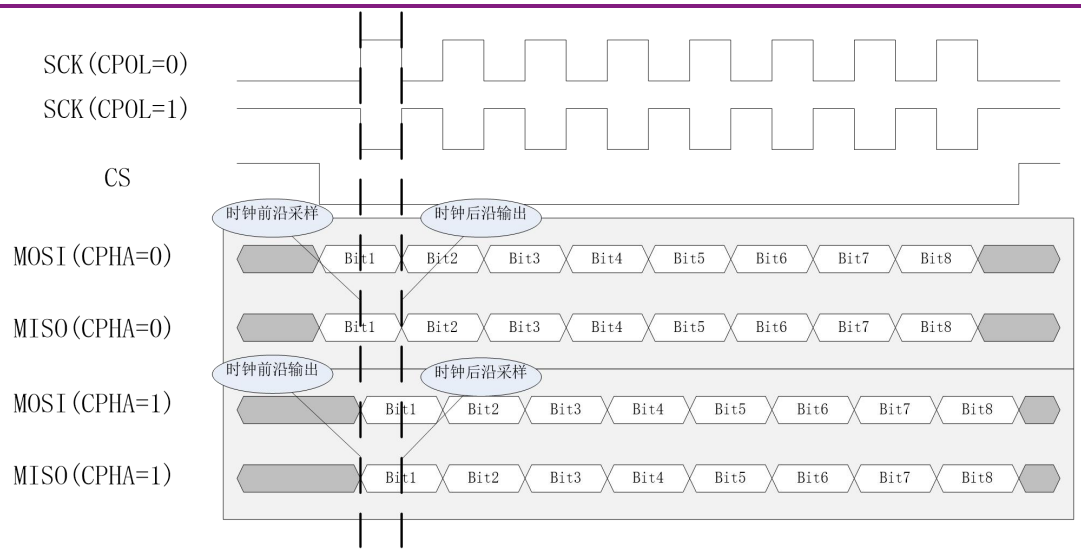
				0：硬件逻辑控制
	[12]: SPI_MIMO	0x0	RW	1：MOSI 输入 0：MOSI 输出
	[9]: SPI_CFG_CSS	0x0	RW	SPI CS 片选控制
	[8]: SPI_CFG_CSS_EN	0x0	RW	SPI CS 手动控制使能，配 1 后 SPI CS 信号受 SPI_CFG_CSS 控制
	[3:0]: SPI_CSS_EN	0x1	RW	SPI CS0-3 选择
0xF2000020 WDATA_BNUM	[8]: SPI_WDATA_MSB	0x0	RW	0：wdata = {wdata[7:0],wdata[15:8], wdata[23:16],wdata[31:24]} 1：wdata = wdata.
	[1:0]: SPI_WDATA_BNUM	0x0	RW	SPI 传输 Byte 数目 0：4Byte 1：1Byte 2：2Byte 3：3Byte
0xF2000024 SPI_WDATA	[31:0]: SPI_WDATA	0x0	RW	SPI 移位数据寄存器
0xF2000028 SPI_DFT_WDATA	[31:0]: SPI_DFT_WDATA	0x0	RW	SPI 默认传输数据，用于 CPU BOOT 功能

10.1. SPI SCK 及模式控制

0xF200002C SPI_RDATA	[31:0]: SPI_RDATA	0x0	RO	SPI 读取数据

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：



10.2. SPI CS 片选控制

图表 9：SPI 模式选择时序图

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）。

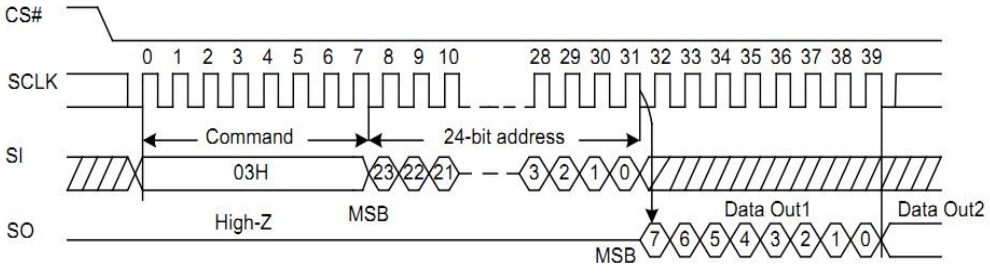
SPI master 支持 4 个 CS 信号，由 SPI_CSS_EN 寄存器控制，每次 SPI 操作可以选择一个或者多个，选择

10.3. SPI BOOT FLASH 器件限制

多个的时候需要注意 MISO 的冲突情况。

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：



图表 10：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

11. SPI_SLAVE

芯片集成一个通用的 SPI_SLAVE 模块，可以与 SPI_MASTER 直接通信。此 SPI SLAVE 模块的特性如下：

- 支持 SDMA 硬件流控模式，可通过 SDMA 来进行数据发送
- 数据位宽 4 到 16bit 可配置
- 接收方向 16bit X 32 深度的 FIFO
- 发送方向 16bit X 32 深度的 FIFO
- 支持单方向发送或接收
- 支持多个中断

SPI SLAVE 功能相关寄存器如下：

表格 26：SPI SLAVE 寄存器

地址/名称	域	初值	类型	说明
0xF2200000 CTRLR	[11]: SRL	0x0	RW	1：Loop back 测试模式 0：常规模式
	[10]: SLV_OE	0x0	RW	0：TXD 使能 1：TXD 信号关闭
	[9:8]: TMOD	0x0	RW	2'b00：发送接收同时使能 2'b01：仅发送 2'b10：仅接收 2'b11：NA
	[7]: SCPOL	0x0	RW	CPOL 模式
	[6]: SCPH	0x0	RW	CPHA 模式
	[3:0]: DFS	0x7	RW	数据帧位宽，有效值为 0x3 到 0xF，分别代表 4 到 16Bit
0xF2200008 SSIENR	[0]: SSI_EN	0x0	RW	SPI SLAVE 功能使能
0xF2200018 TXFTLR	[4:0]: TFT	0x0	RW	发送 FIFO 水线
0xF220001C RXFTLR	[4:0]: RFT	0x0	RW	接收 FIFO 水线
0xF2200020 TXFLR	[5:0]: TXTFL	0x0	RO	发送 FIFO 中的数据量

0xF2200024 RXFLR	[5:0]: RXTFL	0x0	RO	接收 FIFO 中的数据量
0xF2200028 SR	[5]: TXE	0x0	RC	发送错误
	[4]: RFF	0x0	RO	接收 FIFO 满
	[3]: RFNE	0x0	RO	接收 FIFO 非空
	[2]: TFE	0x0	RO	发送 FIFO 空
	[1]: TFNF	0x0	RO	发送 FIFO 非满
	[0]: BUSY	0x0	RO	SPI SLAVE 忙状态
0xF220002C IMR	[4]: RXFIM	0x1	RW	接收 FIFO 超过水线中断使能
	[3]: RXOIM	0x1	RW	接收 FIFO 溢出中断使能
	[2]: RXUIM	0x1	RW	接收 FIFO 向下溢出中断使能
	[1]: TXOIM	0x1	RW	发送 FIFO 写入溢出中断使能
	[0]: TXEIM	0x1	RW	发送 FIFO 低于水线中断使能
0xF2200034 RISR	[4]: RXFIR	0x0	RO	对应于上一个寄存器的中断状态
	[3]: RXOIR	0x0	RO	
	[2]: RXUIR	0x0	RO	
	[1]: TXOIR	0x0	RO	
	[0]: TXEIR	0x0	RO	
0xF2200038 TXOICR	[0]: TXOICR	0x0	RO	读此寄存器清零发送 FIFO 溢出中断
0xF220003C RXOICR	[0]: RXOICR	0x0	RO	读此寄存器清零接收 FIFO 溢出中断
0xF2200040 RXUICR	[0]: RXUICR	0x0	RO	读此寄存器清零接收 FIFO 向下溢出中断
0xF2200048 ICR	[0]: ICR	0x0	RO	清零所有 SPI SLAVE 中断
0xF2200060-0xF2200EC DR	[15:0]:DR	0x0	RW	读：读取接受 FIFO 中接受到的数据； 写：填充 SPI SLAVE 发送 FIFO
0xF22000F4	[28]: DMA_TIME_OUT_EN	0x0	RW	DMA RX 超时使能
DMA_TIME_OUT_CTR	[23:0]: DMA_TIME_OUT_INIT	0x1000	RW	DMA RX 超时计数器配置

L				

注：DR 寄存器地址为 0x60 到 0xEC 整个地址范围，内部映射到读写 FIFO 的入口地址，软件选择此范围内的任意地址操作即可；DR 寄存器的有效位宽由 DFS 寄存器决定，默认有效值为 8bit。

11.1. 软件使用注意事项

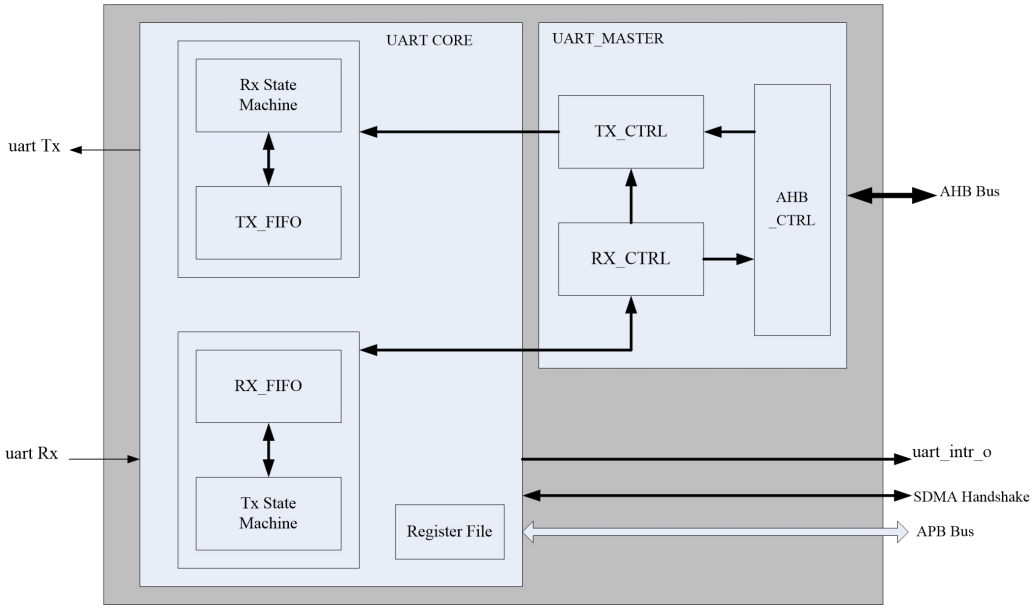
- 接受 FIFO 和发送 FIFO 的深度均为 32，实际 FIFO 空间的使用和 DFS 寄存器相关，寄存器配置为 0xF 的时候最大使用 16bitX32，DFS 配置为 0x7 的时候最大使用 8bitX32；
- 软件可以通过中断或者监控 RFNE 和 TFNF 来决定读取数据或者补充发送数据；
- 传输数据速度较快时可以通过 SDMA 来搬移数据，使用 SDMA 的时候需要打开硬件数据流控功能，否则可能会占用大量的片内总线资源，导致软件效率低下；
- 使能 SPI SLAVE 功能之前，数据无法写入都发送 FIFO 中，使能之前写入的数据将全部丢弃；
- CTRLR、TXFTLR、RXFTLR 三个寄存器需要在使能 SPI SLAVE 功能之前配置，SPI SLAVE 一旦使能后软件不可以配置这几个寄存器。

12. UART 串行接口

芯片内置 5 组通用 UART 接口，此接口支持如下特性：

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

UART 模块系统结构图如下：



图表 11：UART 系统结构图

UART 模块相关寄存器如下表：

表格 27：UART 模块相关寄存器

地址/名称	域	初值	类型	说明
0xF2X00000	[7:0]:FIFO	0x0	RW	写操作：填充发送 FIFO

BUFR				读操作：读取接收 FIFO
0xF2X00004 IER	[6]: IE_TWM	0x0	RW	发送 FIFO 低于水线中断使能
	[4]: IE_RTO	0x0	RW	接收 FIFO 非空超时中断使能
	[2]: IE_RLS	0x0	RW	接收帧错误中断使能
	[1]: IE_THRE	0x0	RW	发送 FIFO 空中断使能
	[0]: IE_RDA	0x0	RW	接收 FIFO 超过水线中断使能
0xF2X00008 IIFCR	[31:16]: IIFT	0x3ff	RW	接收 FIFO 非空中断超时时间，PCLK2 在 50MHz，默认值为 3072us
	[15:8]: ITWM	0x10	RW	发送 FIFO 中断水线
	[7:0]: IIFC	0xe0	RW	此寄存器读取时： [0]：为 0 时中断状态有效，为 1 时中断状态无效 [3:1]=3'b011：接收帧错误中断状态 [3:1]=3'b010：接收数据超过水线中断状态 [3:1]=3'b001：发送 FIFO 为空中断状态 [7:4]：接收 FIFO 中断触发水线配置 此寄存器写入时： [1]：写 1 时清空接收 FIFO [2]：写 1 时清空发送 FIFO [7:4]：接收 FIFO 中断触发水线，中断数目为[7:4]<<4；下面两个特殊值例外： 4'b0000：1Byte 4'b1111：256Byte
0xF2X0000C LCR	[7]: LC_DL	0x0	RW	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/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=50MHz 时一个时间单元是 3us。接收超时默认时间为 3us*IIFT(0x3ff+1)=3072 us
	[6]: LC_BC	0x0	RW	1：强制 TX 输出为 0 0：TX 正常输出
	[5]: LC_SP	0x0	RW	0：使用数据 0 进行校验 1：使用数据 1 进行校验
	[4]: LC_EP	0x0	RW	0：奇(odd)校验 1：偶(even)校验
0xF2X0000C LCR	[3]: LC_PE	0x0	RW	校验使能
	[2]: LC_SB	0x0	RW	0：使用 1bit 停止位 1：使用 2bit 停止位
	[1:0]: LC_BITS	0x3	RW	2'b00：使用 5bit 数据 2'b01：使用 6bit 数据 2'b10：使用 7bit 数据 2'b11：使用 8bit 数据
0xF2X00014 LSR	[31:16]: LS_RFTO	0x0	RO	接收 FIFO 非空计时器数值
	[9]: LS_TWM	0x0	RC	发送 FIFO 低于水线状态
	[8]: LS_RTO	0x0	RC	接收 FIFO 非空超时状态
	[7]: LS_EI	0x0	RC	接收 FIFO 数据中有校验错误
	[6]: LS_TE	0x0	RO	发送完成
	[5]: LS_TFE	0x0	RO	发送 FIFO 空
	[4]: LS_BI	0x0	RC	UART break 中断状态
	[3]: LS_FE	0x0	RC	帧错误中断状态
	[2]: LS_PE	0x0	RC	校验错误中断状态
	[1]: LS_OE	0x0	RC	接收 FIFO 溢出中断状态
	[0]: LS_DR	0x0	RO	接收 FIFO 中有数据
0xF2X003C0 TF_CNT	[8:0]: T_CNT	0x0	RW	发送 FIFO 中的数据长度
0xF2X003C4 RF_CNT	[8:0]: R_CNT	0x0	RW	接收 FIFO 中的数据长度
波特率检测功能				
0xF2X003D0 BAUD_DET_CTRL	[1]: BAUD_DET_CLR	0x0	A0	清空当前波特率检测值
	[0]: BAUD_DET_EN	0x0	RW	使能波特率检测功能

0xF2X003D4 BAUD_MAX_CNT	[31:0]: BAUD_MAX_CNT	0xffff ffff	RW	波特率检测的最大值
0xF2X003D8 BAUD_MIN_CNT	[31:0]: BAUD_MIN_CNT	0x0	RW	波特率检测的最小值
0xF2X003DC BAUD_DET_CNT	[31:0]: BAUD_DET_CNT	0xffff ffff	RO	检测到的波特率值，按 PCLK2 数目计算
红外载波控制				
0xF2X003E0 IR_CTRL	[5]: UART_RX_INVERT	0x0	RW	IR 模式下反转 RX 信号
	[4]: UART_TX_INVERT	0x0	RW	IR 模式下反转 TX 信号
	[1]: IR_CARRIER_MOD	0x0	RW	1：载波信号与 TX 信号相与 0：载波信号与 TX 信号相或
	[0]: IR_CARRIER_EN	0x0	RW	红外载波使能
0xF2X003E4 CARRIER_PERIOD_CFG	[31:16]: CARRIER_CLK_PERIOD	0x522	RW	红外载波周期，PCLK2 时钟数目
	[15:0]: CARRIER_HIGH_PERIOD	0x1b5	RW	红外载波高电平持续 PCLK2 周期数目

注：X=5-7，分别代表 UART0、UART1、UART2

12.1. 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。baud_div = $50x10^6/16/115200 = 27.12$。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

12.1.1. 波特率检测

$50x10^6/16/27 = 115740$

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：

1) 打开 BAUD_DET_EN

2) 配置 BAUD_MAX_CNT 及 BAUD_MIN_CNT，确认检测波特率的范围

3) 配置 BAUD_DET_CLR 为 1，清空波特率统计值

4) 等待一段时间，确保这段时间中，UART 能够接收到数据

5) 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。

12.2. 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：

表格 28：UART 中断状态查询表

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

12.3. 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。

12.4. 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 0xXXXXXXXX // read reg
- write 0xXXXXXXXX 0xYYYYYYYY // write reg

13. 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：

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

EFUSE 功能寄存器如下：

表格 29：EFUSE 功能寄存器

地址/名称	域	初值	类型	说明
0xF2A00000 EFUSE_CTRL	[8]: RD_CMP_EN	0x0	RW	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
	[0]: EFUSE_LOOP	0x0	RW	测试功能
0xF2A00004 EFUSE_CMD	[1:0]: RW_START	0x0	RW	读写功能完成后硬件清零 2：EFUSE 烧写 1：EFUSE 读取 0：空闲状态
0xF2A0000C CTRL_STATUS	[6:4]: CUR_STATE	0x0	RO	状态机状态
	[0]: RDCMP_SUCCESS	0x0	RO	数据比对成功
0xF2A00100-0xF2A0010C PGM_DATA	[31:0]: PGM_DATA_n_n=0-3	0x0	RW	需要写入的数据
0xF2A00200-0xF2A0020C READ_DATA	[31:0]: READ_DATA_n_n=0-3	0x0	RO	读取出来的数据

表格 30：EFUSE 读写时序寄存器

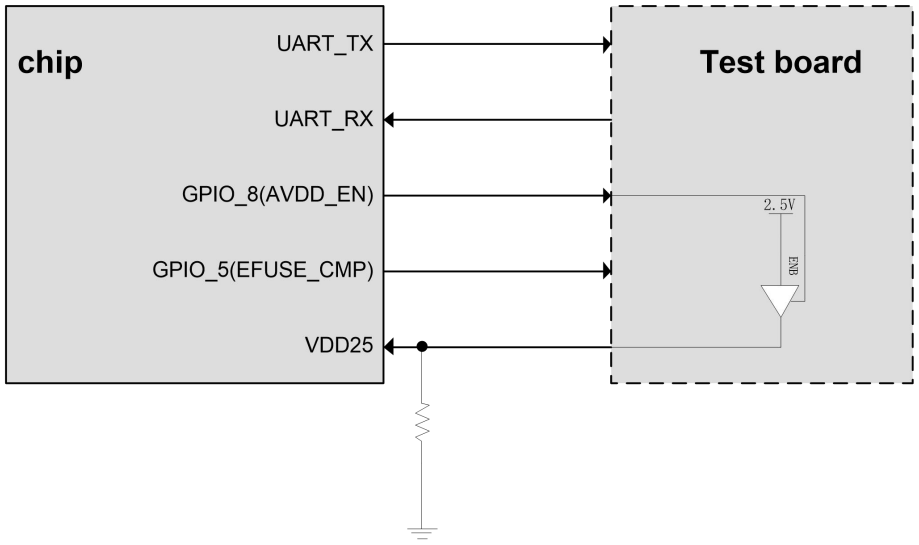
地址/名称	域	初值	类型	说明
0xF2A00010 BURN_TIM	[31:24]: BURN_TIME_L	0xd	RW	烧写信号低电平时间
	[23:16]: BURN_TIME_H	0x33	RW	烧写信号高电平时间
	[15:8]: BURN_SP_PGM	0x2	RW	PGMEN 到 AEN 建立时间

E_0	[7:0]: BURN_HP_A	0x1	RW	AEN 后 ADDR 保持时间
0xF2A00014 BURN_TIME_1	[31:16]: BURN_SP_PG_AVDD	0x61a8	RW	AVDD 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
	[15:8]: BURN_HP_PG_AVDD	0x0f	RW	PGMEN 到 AVDD 保持时间
	[7:0]: BURN_SP_RD	0x02	RW	RDEN 信号到 AVDD 建立时间
0xF2A00020 READ_TIME	[31:24]: READ_HR_A	0x1	RW	AEN 到 ADDR 保持时间
	[23:16]: READ_SR_RD	0x2	RW	RDEN 到 AEN 建立时间
	[15:8]: READ_RD_L	0x2	RW	读取 AEN 低电平持续时间
	[7:0]: READ_RD_H	0x4	RW	读取 AEN 高电平持续时间

注：软件不能随意修改此组寄存器的数值

外部电路连接图如下所示：

13.1. 烧写功能外部硬件电路



图表 12：EFUSE 烧写控制电路

图表 13：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。接受到烧写命令后，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

13.2. UART DEBUG 模式写入

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。

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 0xF2100000 0x1100

write 0xF2100054 0x100 // open GPIO8 and GPIO5 for EFUSE ctrl

// write EFUSE burn data

write 0xF2A00100 0xXXXXXXXX

write 0xF2A00104 0xXXXXXXXX

write 0xF2A00108 0xXXXXXXXX

write 0xF2A0010C 0xXXXXXXXX

// Start burn Efuse'

write 0xF2A00004 0x2

// wait 3ms for Efuse burn done

// Start read Efuse

write 0xF2A00000 0x1110

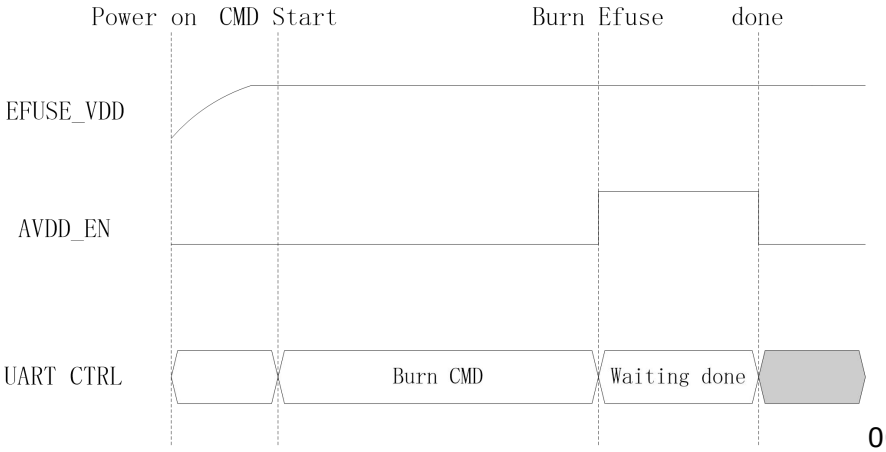
write 0xF2A00004 0x1 //read efuse and enable auto compare

// if compare success GPIO5 will output high

read 0xF2A00200

read 0xF2A00204

read 0xF2A00208



read 0xF2A 0020C

13.3. 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：

1) 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）
2) 写入需要烧写的数据到 0xF2A00100(PGM_DATA_0_)到 PGM_DATA_3_
3) 写入 0x2 到 0xF2A00000(EFUSE_EFUSE_CTRL)寄存器中，开始烧写流程
4) 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：

1) 写入 0x1 到 0xF2A00000(EFUSE_EFUSE_CTRL)寄存器中，开始读取 EFUSE 中的数据
2) 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
3) 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

14. 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

表格 31：MDIO 功能相关寄存器

地址/名称	域	初值	类型	说明
0xF2400020 MCFG	[15]: RST	0x0	RW	1：复位 MDIO 模块
	[14:10]: IFG	0x0	RW	MAC 模式读写结束之后持续等待 MDC 时钟周期
	[9]: MDIO_LS	0x1	RW	MDIO 低速模式，CLKS 寄存器大于 0 时，可以配置此寄存器
	[8]: MDIO_MAC	0x0	RW	0：MDIO PHY 模式 1：MDIO MAC 模式
	[4:2]: CLKS	0x5	RW	MDC 时钟分频设置，MDC = PCLK2/(2^(CLKS+1))
	[1]: SPRE	0x0	RW	1：关闭 32bit 的 preamble 域
	[0]: SCINC	0x0	RW	1：轮询读取模式下，FIAD 寄存器自动递增
0xF2400024 MCMD	[1]: SCAN_READ	0x0	RW	1：轮询读取模式
	[0]: RSTAT	0x0	RW	1：单次读取
0xF2400028 MADR	[12:8]: FIAD	0x0	RW	5bit PHY address 参数
	[4:0]: RGAD	0x0	RW	5bit REG address 参数
0xF240002C MWTD	[15:0]: CTLD	0x0	RW	写入到 PHY 的 16bit 数据
0xF2400030 MRDD	[15:0]: PRSD	0x0	RO	从 PHY 读取到的 16bit 数据
0xF2400034 MIND	[3]: MIILF	0x0	W1C	1：MDIO MAC 模式连接失败
	[2]: NVALID	0x0	RO	1：MDIO MAC 读取未完成，读取数据暂时无效
	[1]: SCAN	0x0	RO	1：SCAN 模式
	[0]: BUSY	0x0	RO	1：MDIO MAC 状态机忙状态
MDIO PHY 模式相关寄存器				
0xF240004C PHYID	[5]: PHYID_TEST	0x0	RW	测试模式
	[4:0]: PHYID	0x0	RW	5bit 的 PHY address

0xF2400050 PHYSTA	[15:0]: PHYSTA	0xfe00	RW	PHY 状态
0xF2400078 MGDI	[15:0]: MGDI	0x0	RC	MAC 写入到 PHY REG address 0x10 的数据，读清

14.1. MAC 模式编程实例

0xF240007C MGDO	[15:0]: MGDO	0x0	RW	PHY REG address 0x14 输出到 MAC 的数据

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_MOIO_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();
}
```

15. 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，处于端口数目限制，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：

表格 32：GE 相关配置寄存器

地址/名称	域	初值	类型	说明
0xD0000800 BASIC_CFG	[31:24]: RX_PDMA_PKT_WRSV	0x4	RW	接收使用 PDMA 的时候，在包头预留的地址空间长度
	[16]: LPBK_EN	0x0	RW	LPBK 环回
	[9:8]: TX_ARB	0x0	RW	发送侧优先级控制 0：AHB 总线通道高优先级 1：PDMA 通道高优先级 2：Round Robin
	[7]: TX_CRC_EN	0x0	RW	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
	[6]: TX_PAD_EN	0x0	RW	对小于 64 的包进行 pading 操作，默认情况下 GE 模块中已经进行了此操作
	[5]: RX_ERR_DROP	0x1	RW	错误包丢弃使能
	[4]: RX_CRC_RM	0x0	RW	删除接收包中的 CRC，默认情况下 GE 模块已经做过此处理
	[3]: RX_CRC_CHK	0x0	RW	接收包 CRC 校验，默认情况下 GE 模块已经做过此处理
	[2]: PKT_MODE	0x0	RW	接收包传输模式选择： 0：AHB SLAVE by CPU or SDMA 1：PDMA
	[1]: TX_EN	0x0	RW	发送使能
	[0]: RX_EN	0x0	RW	接收使能
0xD0000804 RX_BUF_THD	[27:16]: PAUSE_THD	0x800	RW	接收数据超过此水线，GE 会发起 PAUSE 帧，接收 BUF 最大为 2K word
	[11:0]: RBUF_MAX_THD	0x800	RW	接收数据 BUF 深度，最大 2K word，超过此值后将丢包
0xD0000808	[14:8]: TX_BRDY_THD	0x20	RW	接收侧 SDMA 硬件流控水线

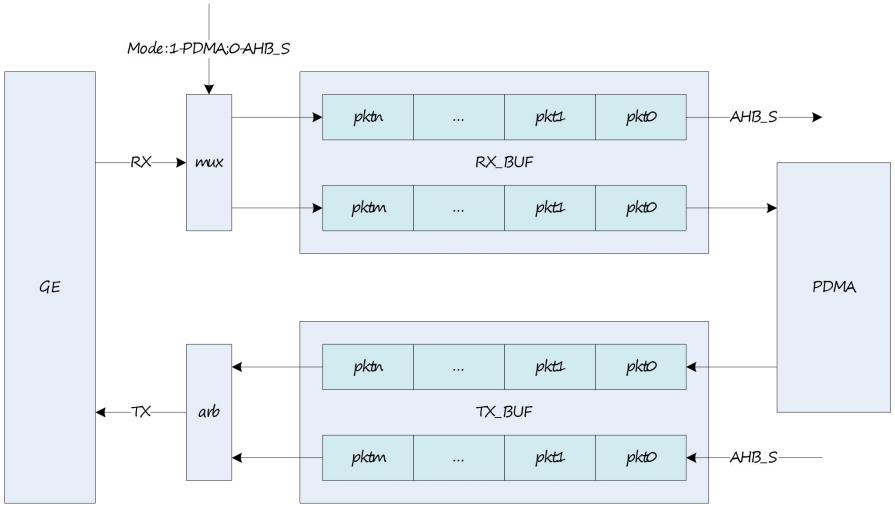
BRDY_THD	[6:0]: RX_BRDY_THD	0x20	RW	发送侧 SDMA 硬件流控水线
0xD000080C RX_PDMA_SADDR	[31:0]: RX_PDMA_SADDR	0x0	RW	RX PDMA 起始地址，需 word 对齐
0xD0000810 RX_PDMA_LEN	[19:0]: RX_PDMA_LEN	0x0	RW	RX PDMA 搬移最大环回空间，为 0 的时候代表 4MB，以 word 为单位
0xD0000814 RX_PDMA_RD_CNT	[19:0]: RX_PDMA_RD_CNT	0x0	RW	软件在使用完 PDMA 接收到的数据后需要写入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
0xD0000818 TX_PDMA_SADDR	[31:0]: TX_PDMA_SADDR	0x0	RW	发送 PDMA 模式起始地址，需 word 对齐
0xD000081C TX_AHB_BUF	[26:16]: AHB_TX_BUF_LEN	0x0	RW	发送 AHB 模式 BUF 深度，单位 word
	[9:0]: AHB_TX_BUF_SADDR	0x0	RW	发送 AHB 模式 BUF 起始地址，单位 word
0xD0000820 TX_PDMA_BUF	[26:16]: PDMA_TX_BUF_LEN	0x0	RW	发送 PDMA 模式 BUF 深度，单位 word
	[9:0]: PDMA_TX_BUF_SADDR	0x0	RW	发送 PDMA 模式 BUF 起始地址，单位 word
0xD0000824 TX_PDMA_INFO_HLEN	[6:0]: TX_PDMA_INFO_HLEN	0x20	RW	PDMA 发送高优先级 INFO 深度设置，低优先级深度为 64-当前配置
0xD0000828 PDMA_INTR_NUM	[30:24]: TX_PDMA_INTR_LNUM	0x1	RW	PDMA 发送低优先级，INFO 数目中断水线，最小值为 1
	[22:16]: TX_PDMA_INTR_HNUM	0x1	RW	PDMA 发送高优先级，INFO 数目中断水线，最小值为 1
	[7:0]: RX_PDMA_INTR_NUM	0x5	RW	PDMA 接收 INFO 中断数目水线
0xD000082C RX_PDMA_INTR_TIMER	[31:0]: RX_PDMA_INTR_TIMER	0x10000	RW	PDMA 接收 INFO 非空超时配置
0xD0000830 TX_PKT_INFO_CHK	[31]: TX_PKT_MAX_DROP	0x0	RW	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。

	[27:16]: TX_PKT_MAX_BNUM	0x600	RW	大包丢弃功能包长配置
	[15]: TX_PKT_MIN_DROP	0x0	RW	发送小包丢弃使能
	[11:0]: TX_PKT_MIN_BNUM	0x30	RW	小包丢弃功能包长配置
0xD0000834 INTR_EN	[10]: PDMA_BBUF_RERR_INTR_EN	0x0	RW	PDMA 错误中断使能
	[9]: PDMA_BBUF_WERR_INTR_EN	0x0	RW	
	[8]: PDMA_ERR_INTR_EN	0x0	RW	
0xD0000834 INTR_EN	[6]: PDMA_TX_EOP_INTR_EN	0x0	RW	PDMA 模式发送出一个完整包中断使能
	[5]: AHB_TX_EOP_INTR_EN	0x0	RW	AHB 模式发送出一个完整包中断使能
	[4]: AHB_TX_PKT_DONE_INTR_EN	0x0	RW	GEI 模块发送端接收到 AHB 总线一个完整包中断使能
	[3]: PDMA_TX_PKT_DONE_INTR_EN	0x0	RW	GEI 模块发送端接收到 PDMA 一个完整包中断使能
	[2]: PDMA_RX_PKT_TO_INTR_EN	0x0	RW	PDMA 接收包非空超时中断使能
	[1]: PDMA_RX_PKT_DONE_INTR_EN	0x0	RW	PDMA 接收包数目到达水线中断使能
	[0]: AHB_RX_PKT_RDY_INTR_EN	0x0	RW	AHB 接收包中断使能
0xD0000838 PDMA	[30:24]: PDMA_RBURST_THD	0x2	RW	PDMA 模块控制
	[22:16]: PDMA_WBURST_THD	0x20	RW	
	[6:0]: PDMA_BL	0x20	RW	
GE 功能寄存器				
0xD0000840	[31]: GE_CNT_CLR	0x0	RW	清空 GE 模块内部计数器

GE_CFG	[27:24]: GE_RX_DATA_SMP	0x4	RW	RMII 10M 模式采样点
	[16]: GE_TXC_EN	0x1	RW	gtxc 输出使能
	[15:12]: GE_RXC_PHASE	0x4	RW	RX 时钟相位配置： 0:3*36; 1:1*36; 2:2*36; :3*36; 4:4*36; 5:5*36; 6:6*36; :7*36; 8:8*36; 9:9*36
	[11:8]: GE_TXC_PHASE	0x5	RW	TX 时钟相位配置： 0:3*36; 1:1*36; 2:2*36; :3*36; 4:4*36; 5:5*36; 6:6*36; :7*36; 8:8*36; 9:9*36
	[3]: GE_RMII_MODE	0x1	RW	2'b00: MII_MODE 2'b01: RGMII_MODE 2'b10: RMII_MODE
	[2]: GE_RGMII_MODE	0x0	RW	
	[1:0]: GE_SPEED_SEL	0x1	RW	2'b00: 10M 2'b01: 100M 2'b1x: 1000M
0xD0000844 GE_ORDER	[13]: GE_PIN_TXE_SWAP	0x0	RW	交换 TXEN 和 GTXC 信号
	[12]: GE_PIN_TXD_SWAP	0x0	RW	交换 TXD[3:0]的顺序
	[11:10]: GE_PIN_TX_ORDER	0x0	RW	0 : GTXC, TXEN, TXD; 1 : TXEN, TXD, GTXC; 2 : TXD, GTXC, TXEN;
	[9]: GE_PIN_RXD_SWAP	0x0	RW	交换 RXD[3:0]的顺序
	[8]: GE_PIN_RX_ORDER	0x0	RW	0 : RXDV, RXD; 1 : RXD, RXDV;
GEI PKT INFO				
0xD0000850 RX_PKT_INFO_VLD	[23:16]: RX_PDMA_PKT_INFO_CNT	0x0	RO	PDMA 接收队列中有效的 PKT_INFO 数目
	[0]: RX_AHB_PKT_INFO_VLD	0x0	RO	AHB 接收 PKT_INFO 有效标志
0xD0000854 RX_AHB_PKT_INFO	[31:0]: RX_AHB_PKT_INFO	0x0	RO	AHB 接收 PKT_INFO 信息
0xD0000858	[31]:	0x0	RO	AHB TX 模式，PKT_INFO

TX_PKT_INFO_FULL	TX_AHB_PKT_INFO_FULL			FIFO 满
	[30]: TX_PDMA_LPKT_INFO_FULL	0x0	RO	PDMA TX 高速通道，PKT_INFOFIFO 满
	[29]: TX_PDMA_HPKT_INFO_FULL	0x0	RO	PDMA TX 低速通道，PKT_INFO FIFO 满
	[20:16]: TX_AHB_PKT_INFO_CNT	0x0	RO	AHB TX 模式，PKT_INFO 数目
	[14:8]: TX_PDMA_LPKT_INFO_CNT	0x0	RO	PDMA TX 高速通道，PKT_INFO 数目
	[6:0]: TX_PDMA_HPKT_INFO_CNT	0x0	RO	PDMA TX 低速通道，PKT_INFO 数目
0xD000085C TX_AHB_PKT_INFO	[31:0]:TX_AHB_PKT_INFO	0x0	RW	AHB TX 模式，PKT_INFO

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：



图表 14：GEI 接口结构

15.1 方式选择

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

15.2. 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。

15.3. RX 接收数据

- 0xD00020xx 地址段为被动 AHB 模式的发送和写入地址。

15.3.1. 被动 AHB 方式

1) 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：

Packet Length – 12bits	RSV – 20bits

PKT_LEN 表示当前这个以太网包的字节数量。

2) 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。

3) 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。

4) 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。

15.4. TX 发送数据

15.4.1. 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

PMDA_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

15.4.2. 被动 AHB 方式

1) 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

Sop (1bit)	Eop (1bit)	Part Packet Length – 11bits	Rsv(19bit)

PKT_LEN 表示当前这个以太网包的字节数量。

2) 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。

3) 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，总线被会被占用。

4) 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。

5) 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。

16. 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：

表格 33：看门狗复位寄存器

地址/名称	域	初值	类型	说明
0xF2300000 WDOG_CFG	[31]: WDOG_EN	0x0	RW	看门狗使能
	[27:0]: WDOG_CFG	0xfffffff	RW	看门狗计数器，看门狗溢出时间为：WDOG_CFG*PCLK2*256
0xF2300004 WDOG_CLR	[0]: WDOG_CLR	0x0	A0	写 1 清零看门狗计数器，即喂狗操作
0xF2300008 WDOG_IND	[15:0]: WDOG_IND	N/A	RO	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
0xF230000C WDOG_RST_CNT_CFG	[24:0]: WDOG_RST_CNT_CFG	0x1ffff	RW	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
0xF2300010 WDOG_RST_CFG	[2]: WDOG_POR_RST_EN	0x0	RW	模拟复位输出
	[1]: WDOG_GPIO_RST_EN	0x0	RW	GPIO 复位输出使能，需要 GPIO 配置 GPIO_WDT_EN
	[0]: WDOG_GLB_RST_EN	0x0	RW	芯片全局复位使能

16.1 看门狗复位控制

典型的电路将 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。

17. 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 模式加解密
- SMS4 ECB 模式加解密
- HPAV 32/24bit CRC 计算
- 支持 SDMA 硬件流控模式

17.1. ACU 加解密流程

ACU 加解密工作流程如下：

1) 模块初始化，配置 ACU 的工作模式，配置秘钥等
2) 传输数据，一般需调用 SDMA
3) ACU 开始工作

17.2. CRC 计算

ACU CRC 功能相关寄存器如下：

表格 34：ACU CRC 功能寄存器

地址/名称	域	初值	类型	说明
0xF0300090 CRC_OP	[29:28]: CRC_FSM	0x0	RO	CRC 计算状态 0：IDLE 1/2：CRC 计算中 3：CRC 计算结束
	[21:16]: CRC_STATUS	0x0	RO	[4:0]：当前 FO 计数 [5]：CRC 引擎工作标志
	[8]: CRC_CLR	0x0	A0	1：清除 CRC 结果
	[0]: CRC_START	0x0	A0	1：启动 CRC 计算
0xF0300094 CRC_BASE_CFG	[8]: CRC_XOROUT	0x1	RW	异或输出使能
	[4]: CRC_INIT_VALUE	0x1	RW	CRC 初始值 0：0x0 1：0xffffffff
	[0]: CRC_MODE	0x1	RW	CRC 模式 0：HPAV24 1：HPAV32/ETH_FCS
0xF0300098	[31]: CRC_FIFO_ERR	0x0	LH	CRC FIFO 错误

CRC_MISC_CFG	[4]: CRC_REFIN	0x0	RW	交换输入数据大小端
	[0]: CRC_REFOUT	0x1	RW	交换输出数据大小端
0xF030009C CRC_DATA_CFG	[15:0]: CRC_DATA_OFFSET	0x0	RW	CRC 计算数据偏移地址

17.2.1. CRC32 参数配置

0xF03000A0 CRC	[31:0]: CRC	0x0	RO	CRC 计算结果

- 知道计算 crc 数据长度

  配 crc_mode 1

  crc_init_value 1

  crc_xorout 1

  crc_data_offset 计算 crc 数据的起始位置

  data_len 计算 crc 数据的长度

- 不知道计算 crc 数据长度

  配 crc_mode 1

  crc_init_value 1

  crc_xorout 1

  crc_data_offset 计算 crc 数据的起始位置

17.2.2. CRC24 参数配置

  data_len 0

- 知道计算 crc 数据长度

  配 crc_mode 0

  crc_init_value 1

  crc_xorout 1

  crc_data_offset 计算 crc 数据的起始位置

  data_len 计算 crc 数据的长度

- 不知道计算 crc 数据长度

  配 crc_mode 0

  crc_init_value 1

  crc_xorout 1

  crc_data_offset 计算 crc 数据的起始位置

  data_len 0

17.3. AES/SMS4 计算

ACU AES/SMS4 相关寄存器如下：

表格 35：ACU AES/SMS4 寄存器

地址/名称	域	初值	类型	说明
0xF0300000 GLB_CFG	[28]: KEY_COUNT_EN	0x1	RW	加解密 key 计数
	[24]: CRC_EN	0x1	RW	CRC 计算使能
	[20]: DEC_EN	0x1	RW	解密功能使能
	[16]: ENC_EN	0x1	RW	加密功能使能
	[8]: ENC	0x1	RW	加解密功能选择 0：解密 1：加密
0xF0300000 GLB_CFG	[7:4]: CRYPT_MODE	0x0	RW	加解密模式 0：ECB 1：CBC other：NA
	[3:0]: CRYPT_ALG	0x0	RW	加解密算法选择 0：AES 1：SMS4 other：NA
0xF0300004 CTRL	[31:16]: DATA_LEN	0x0	RW	AES/SMS4 模式：BLOCK 个数 CRC 模式：输入数据字节数
	[0]: READY	0x0	A0	指示 CBC 第一个数据块
0xF0300008 DATA_ORDER	[6:4]: OUT_ORDER	0x0	RW	输出结果反序 3'bxx1：Byte 内比特反序 3'bx1x：Word 内 Byte 反序 3'b1xx：Block 中 Word 反序
	[2:0]: IN_ORDER	0x0	RW	输入数据反序 3'bxx1：Byte 内比特反序 3'bx1x：Word 内 Byte 反序 3'b1xx：Block 中 Word 反序
0xF030000C INTR_THD	[31:16]: RX_INTR_THD	0x20	RW	输出数据 BUF 中断水线，最大值 64
0xF0300010	[31:16]:	0x20	RW	输出 SDMA 硬件流控 BUF 大

READY_THD	RX_BRDY_THD			小
	[15:0]: TX_BRDY_THD	0x20	RW	输入 SDMA 硬件流控 BUF 大小
0xF0300014 KEY_INTR_THD	[31:0]: KEY_INTR_THD	0x0	RW	加解密 key 计数值中断
0xF0300018 INTR_STA	[8]: KEY_INTR	0x0	LH	key 使用次数中断状态
	[4]: CRC_INTR	0x0	LH	CRC 结束中断状态
	[1]: TX_INTR	0x0	LH	加解密完成中断状态
	[0]: RX_INTR	0x0	LH	可接收数据中断状态
0xF030001C INTR_EN	[8]: KEY_INTR_EN	0x0	RW	key 使用次数中断使能
	[4]: CRC_INTR_EN	0x0	RW	CRC 结束中断使能
	[1]: TX_INTR_EN	0x0	RW	加解密完成中断使能
	[0]: RX_INTR_EN	0x0	RW	可接收数据中断使能
0xF0300040 KEY_0_	[31:0]: KEY_0_	0x0	RW	key[31:0]
0xF0300044 KEY_1_	[31:0]: KEY_1_	0x0	RW	key[63:32]
0xF0300048 KEY_2_	[31:0]: KEY_2_	0x0	RW	key[95:64]
0xF030004C KEY_3_	[31:0]: KEY_3_	0x0	RW	key[127:96]
0xF0300050 KEY_RD_0_	[31:0]: KEY_RD_0_	0x0	RO	out key[31:0]
0xF0300054 KEY_RD_1_	[31:0]: KEY_RD_1_	0x0	RO	out key[63:32]
0xF0300058 KEY_RD_2_	[31:0]: KEY_RD_2_	0x0	RO	out key[95:64]
0xF030005C KEY_RD_3_	[31:0]: KEY_RD_3_	0x0	RO	out key[127:96]
0xF0300060 IV_0_	[31:0]: IV_0_	0x0	RW	CBC 模式初始向量
0xF0300064 IV_1_	[31:0]: IV_1_	0x0	RW	
0xF0300068 IV_2_	[31:0]: IV_2_	0x0	RW	
0xF030006C IV_3_	[31:0]: IV_3_	0x0	RW	

0xF0300070 KEY_COUNT	[31:0]: KEY_COUNT	0x0	WC	key 使用计数器高 32 位输出，内部使用 40 位 key 计数器
0xD1000xxx CRC_DATA_IN	[31:0]:DATA_IN	0x0	RW	CRC 计算数据输入地址

17.3.1. ACU SDMA 使用说明

0xD1008xxx ACU_DATA_IO	[31:0]:DATA_IN_OUT	0x0	RW	AES/SMS4 加解密数据输入地址

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：

1) 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

2) 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
```

17.3.2. 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。

17.4. 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。

18. 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。

18.1. TICK 功能

TICK 主要实现以下功能。

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

TICK 功能相关寄存器如下：

表格 36：TICK 功能相关寄存器

地址/名称	域	初值	类型	说明
0xF1300000 INTR_MASK	[27:16]: IMSK_EVENT	0x0	RW	TICK 事件中断使能 每个 bit 对应于一个 TICK 事件
	[15:12]: IMSK_CTIMER	0x0	RW	Cyclic-Timers 事件中断使能 每个 bit 对应于一个 timer
	[11:0]: IMSK_TIMER	0x0	RW	TICK timer 硬件中断使能 每个 bit 对应于一个 timer
0xF1300004 INTR_STAT	[27:16]: INTR_EVENT	0x0	LH	对应于 INTR_MASK 中断使能的中断状态
	[15:12]: INTR_CTIMER	0x0	LH	
	[11:0]: INTR_TIMER	0x0	LH	
0xF1300008 FREQ	[11:8]: FREQ_RMD	0x1	RW	TICK 频率计算，计算公式： Tick_freqency = (freq_int + (freq_rmd/(freq_div+1))) * Clock_freqency
	[7:4]: FREQ_DIV	0x2	RW	
	[3:0]: FREQ_INT	0x1	RW	
0xF130000C	[31:0]: FO	0x0	RW	SFO

FO				
0xF1300010 SET_IMD	[31:0]: SET_IMD	0x0	RW	TICK 值配置，立即生效
0xF1300014 ADJ	[15:0]: ADJ	0x0	RW	TICK 调整，此寄存器为带符号数
0xF1300018 CURR	[31:0]: CURR	0x0	RO	实时 TICK 值
0xF130001C INTV	[31:28]: INTV_END_SRC	0x6	RW	结束事件
	[27:24]: INTV_START_SRC	0xb	RW	起始事件
0xF130001C INTV	[21:20]: INTV_MODE	0x1	RW	计算起始结束事件时间差模式 0：每次更新时间差 1：锁存最大时间差 2：锁存最小时间差 3：NA
	[19:0]: INTV	0x0	RO	起始结束事件时间差输出
0xF1300020 MISC	[3:2]: EXT_SRC1_EDGE	0x0	RW	Htimer 外部事件捕获方式 [0]：上升沿 [1]：下降沿
	[1:0]: EXT_SRC0_EDGE	0x1	RW	同 EXT_SRC1_EDGE；此外部源支持用作过零点检测
0xF1300030-0xF130005C TIMER_0_-TIMER_11_	[31:0]: TIMER_n_	0xffff ffff	RW	TICK 事件值设置
0xF1300070-0xF130009C EVENT_0_-EVENT_11_	[31:0]: EVENT_n_	0x0	RO	TICK 事件锁存值
0xF13000A0 EVENT_SRC	[31:28]: EVENT_SRC_7_	0xF	RW	事件源选择 0~13：系统内部源 14：外部事件 15：NA
	[27:24]: EVENT_SRC_6_	0xF	RW	
	[23:20]: EVENT_SRC_5_	0xF	RW	
	[19:16]: EVENT_SRC_4_	0xF	RW	
	[15:12]: EVENT_SRC_3_	0xF	RW	

	[11:8]: EVENT_SRC_2_	0xF	RW	
	[7:4]: EVENT_SRC_1_	0xF	RW	
	[3:0]: EVENT_SRC_0_	0xF	RW	
0xF13000A4 EVENT_SRC1	[15:12]: EVENT_SRC_11_	0xF	RW	
	[11:8]: EVENT_SRC_10_	0xF	RW	
	[7:4]: EVENT_SRC_9_	0xF	RW	
	[3:0]: EVENT_SRC_8_	0xF	RW	

18.1.1. 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。

18.1.1.1. Tick 频率设置

Tick 频率可配。

举例：系统时钟为 75MHz，Tick 频率为 25MHz。

25 MHz = (0+(1/3))*75 MHz

FREQ_INT = 0

FREQ_RMD = 1

18.1.1.2. Tick 计数器调整

FREQ_DIV = 3-1=2

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

18.1.2. 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

18.1.3. 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，

18.2. HTIMER 功能

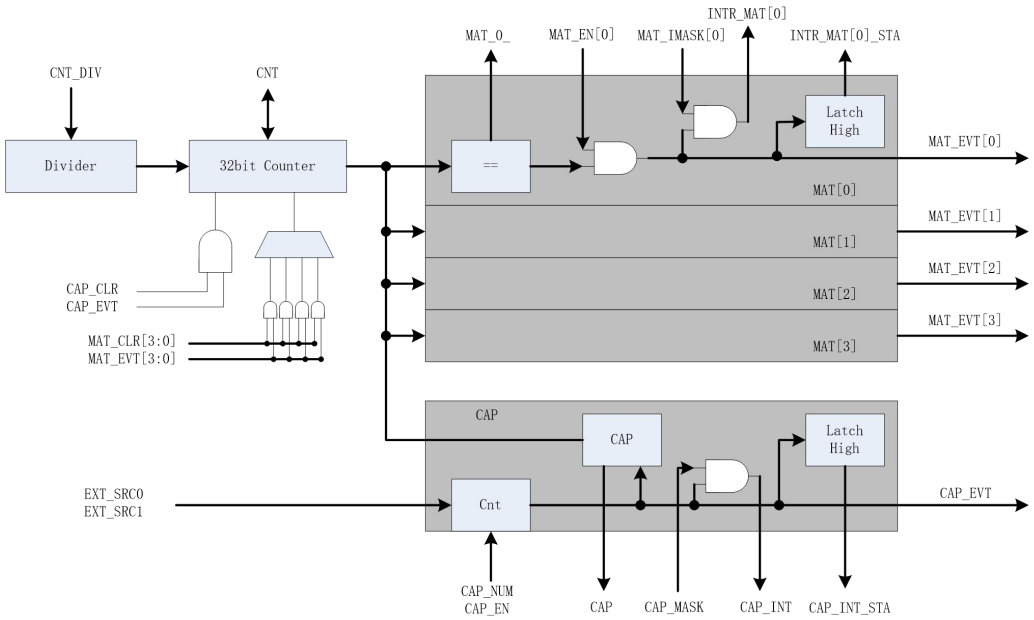
过零事件的时间可通过 EVENT_0_读取。

提供 4 个 32bit 与 Tick 无关的独立 timer。

每个 timer 具备如下功能：

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

HTIMER 硬件结构图如下：



图表 15：HTIMER 硬件结构图

HTIMER 相关寄存器如下：

表格 37：HTIMER 相关寄存器

地址/名称	域	初值	类型	说明
0xF1300300 HTMR0_CNT	[31:0]: HTMR0_CNT	0x0	RW	硬件 timer0 计数器，写入的时候直接修改当前的计数值
0xF1300304 HTMR0_CFG	[31:28]: HTMR0_MAT_IMSK	0x0	RW	HTIMER0 中断使能，每个 bit 对应于一个匹配事件
	[27:24]: HTMR0_MAT_CLR	0x0	RW	事件匹配时清零 timer 计数，每个 bit 对应于一个匹配事件
	[23:20]: HTMR0_MAT_EN	0x0	RW	计数匹配使能，每个 bit 对应于一个匹配事件
	[19:16]: HTMR0_CAP_NUM	0x0	RW	捕获次数配置，当捕获次数到达此配置值时触发捕获事件
	[15]: HTMR0_CAP_IMSK	0x0	RW	捕获事件中断使能
	[14]: HTMR0_CAP_CLR	0x0	RW	捕获事件清零 timer 使能
	[13:12]: HTMR0_CAP_EN	0x0	RW	捕获功能使能
	[11:4]: HTMR0_CNT_DIV	0x0	RW	TIMER 计数器时钟分频
	[0]: HTMR0_CNT_EN	0x0	RW	TIMER 功能使能
0xF1300308 HTMR0_CAP	[31:0]: HTMR0_CAP	0x0	RO	捕获事件发生时，锁存的 TIMER 计数值
0xF1300310-0xF13003 1C HTMR0_MAT	[31:0]: HTMR0_MAT_n_n=0:3	0xffff ffff	RW	HTIMER 匹配计数值配置
0xF1300320-0xF130037C 寄存器为 HTIMER1-3 的配置寄存器				
0xF1300380 HTIMR_INTR	[19]: HTMR3_CAP_INTR	0x0	LH	TIMER3 捕获中断状态
	[18]: HTMR2_CAP_INTR	0x0	LH	TIMER2 捕获中断状态
	[17]: HTMR1_CAP_INTR	0x0	LH	TIMER1 捕获中断状态
	[16]: HTMR0_CAP_INTR	0x0	LH	TIMER0 捕获中断状态
	[15:12]: HTMR3_MAT_INTR	0x0	LH	HTIMER3 匹配中断状态
	[11:8]: HTMR2_MAT_INTR	0x0	LH	HTIMER2 匹配中断状态

	[7:4]: HTMR1_MAT_INTR	0x0	LH	HTIMER1 匹配中断状态

18.2.1. HTIMER 计数器

	[3:0]: HTMR0_MAT_INTR	0x0	LH	HTIMER0 匹配中断状态

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

18.2.2. HTIMER 匹配事件

即对 counter 赋值。

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

18.2.3. HTIMER 捕获

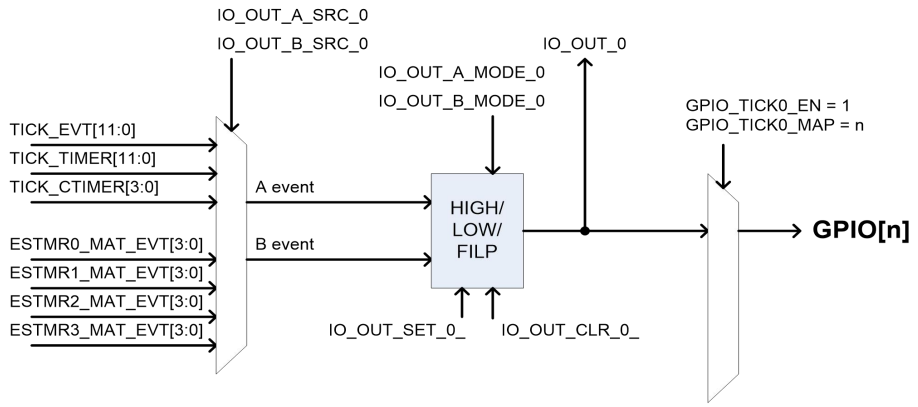
4 个 match 值，故一共有 16 种 match 事件。

支持外部事件通过 GPIO 捕捉：

- 可选择两个 GPIO 当做事件触发源 EXT SOURCE0/1（其中 EXT SOURCE0 一般作为过零检测使用）
- 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

18.3. IO 输出功能

IO output 内部结构图如下：



图表 16：IO output 结构

IO output 相关寄存器如下：

表格 38：TICK IO 相关寄存器

地址/名称	域	初值	类型	说明
0xF1300100 IO_OUT	[3]: IO_OUT_3_	0x0	RO	TICK IO 3 输出值
	[2]: IO_OUT_2_	0x0	RO	TICK IO 2 输出值
	[1]: IO_OUT_1_	0x0	RO	TICK IO 1 输出值
	[0]: IO_OUT_0_	0x0	RO	TICK IO 0 输出值
0xF1300104 IO_OUT_SET_CLR	[7]: IO_OUT_CLR_3_	0x0	A0	TICK IO 3 设置为 0
	[6]: IO_OUT_CLR_2_	0x0	A0	TICK IO 2 设置为 0
	[5]: IO_OUT_CLR_1_	0x0	A0	TICK IO 1 设置为 0
	[4]: IO_OUT_CLR_0_	0x0	A0	TICK IO 0 设置为 0
	[3]: IO_OUT_SET_3_	0x0	A0	TICK IO 3 设置为 1
	[2]: IO_OUT_SET_2_	0x0	A0	TICK IO 2 设置为 1
	[1]: IO_OUT_SET_1_	0x0	A0	TICK IO 1 设置为 1
	[0]: IO_OUT_SET_0_	0x0	A0	TICK IO 0 设置为 1
0xF1300110-0xF130011C IO_OUT_0_CFG-IO_OUT_3_CFG	[25:24]: IO_OUT_B_MODE_0_	0x0	RW	TICK IO 0 B 输出模式
	[21:16]: IO_OUT_B_SRC_0_	0x3f	RW	TICK IO 0 B 事件源选择
	[9:8]: IO_OUT_A_MODE_0	0x0	RW	TICK IO 0 A 输出模式 0：输出低电平

	–			1：输出高电平 2：输出翻转信号 3：输出单周期脉冲
0xF1300110-0xF130011C IO_OUT_0_CFG-IO_OUT_3_CFG	[5:0]: IO_OUT_A_SRC_0_	0x3f	RW	TICK IO 0 A 事件源选择 0~11：TICK timer 12:~15：Cyclic timer 16~27：事件 0 到事件 11 32~35：HTIMER0 事件 36~39：HTIMER1 事件 40~43：HTIMER2 事件 44~47：HTIMER3 事件

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（A event 和 B event）
- 然后通过 IO_OUT_A_MODE_0_和 IO_OUT_B_MODE_0_分别配置当该 A event 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

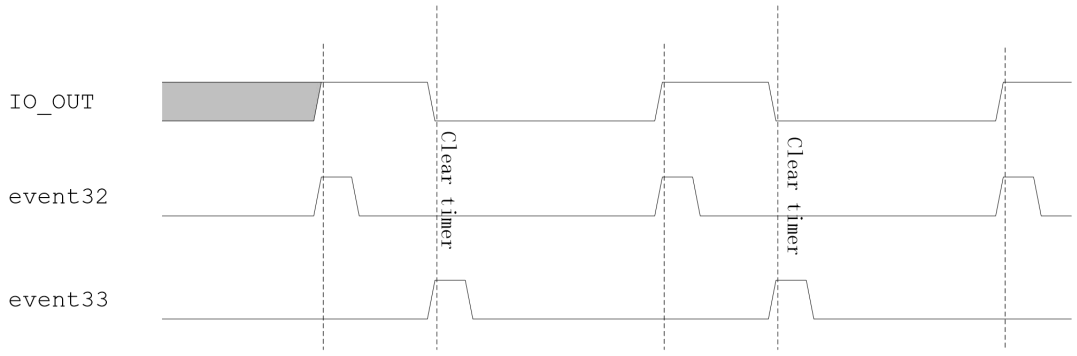
18.4. PWM 输出实例

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_HTMR0_CNT_DIV (8'h0)
```

```
`SET_TICK_HTMR0_MAT_EN   (4'b0011) // 使能两路 match
`SET_TICK_HTMR0_MAT_CLR (4'b0010)      // 第二路 match 的时候 clear 计数器
`SET_TICK_HTMR0_MAT_0_   (10-1)        // 第一次 match 事件的计数器值
`SET_TICK_HTMR0_MAT_1_   (15-1)
`SET_TICK_HTMR0_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：

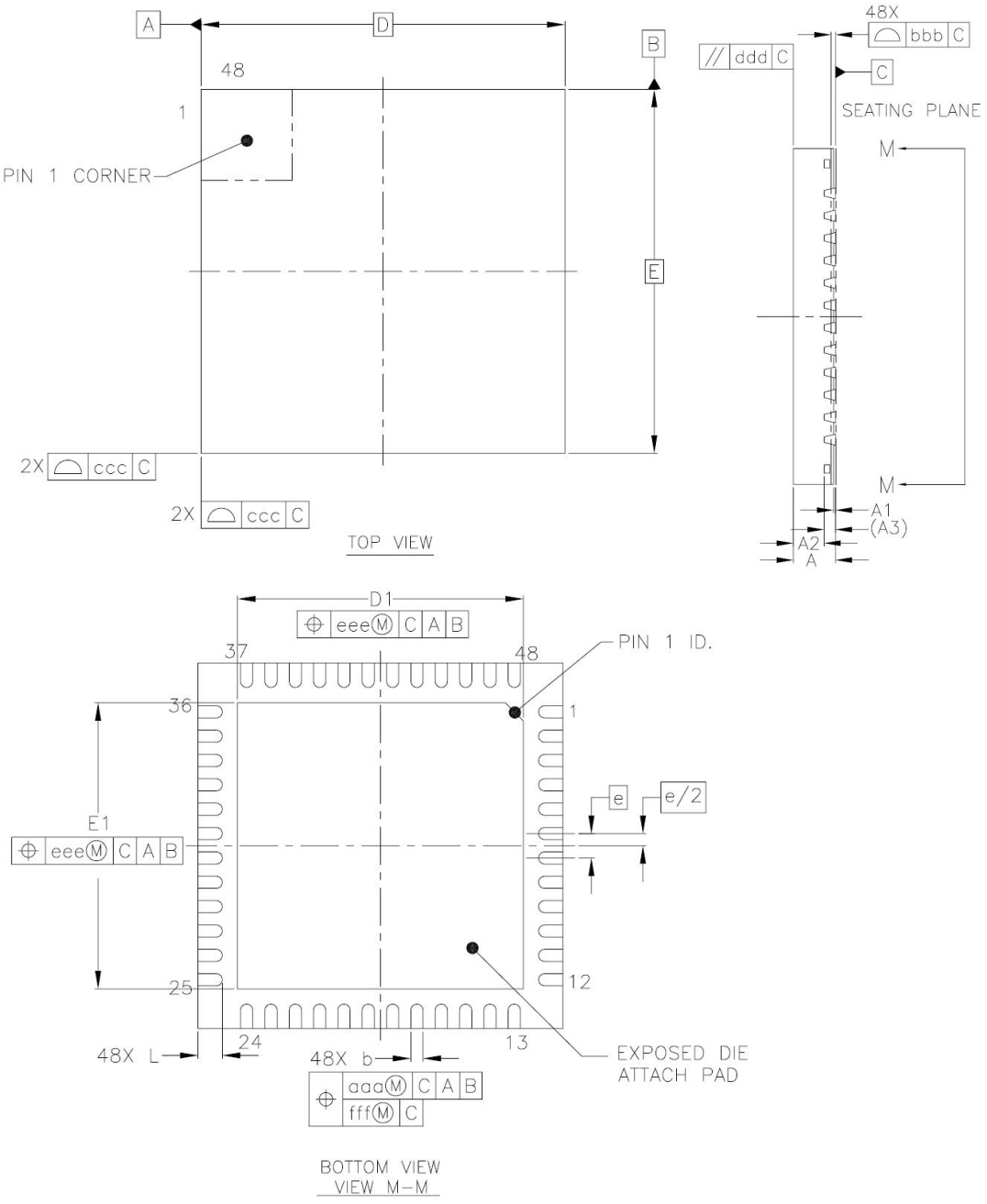


图表 17：HTIMER 调制 PWM 示意图

19. 芯片封装尺寸

19.1. V6201E1 封装

V6201E1 采用 6mmX6mm，48-pin QFN 封装，封装尺寸图如下：



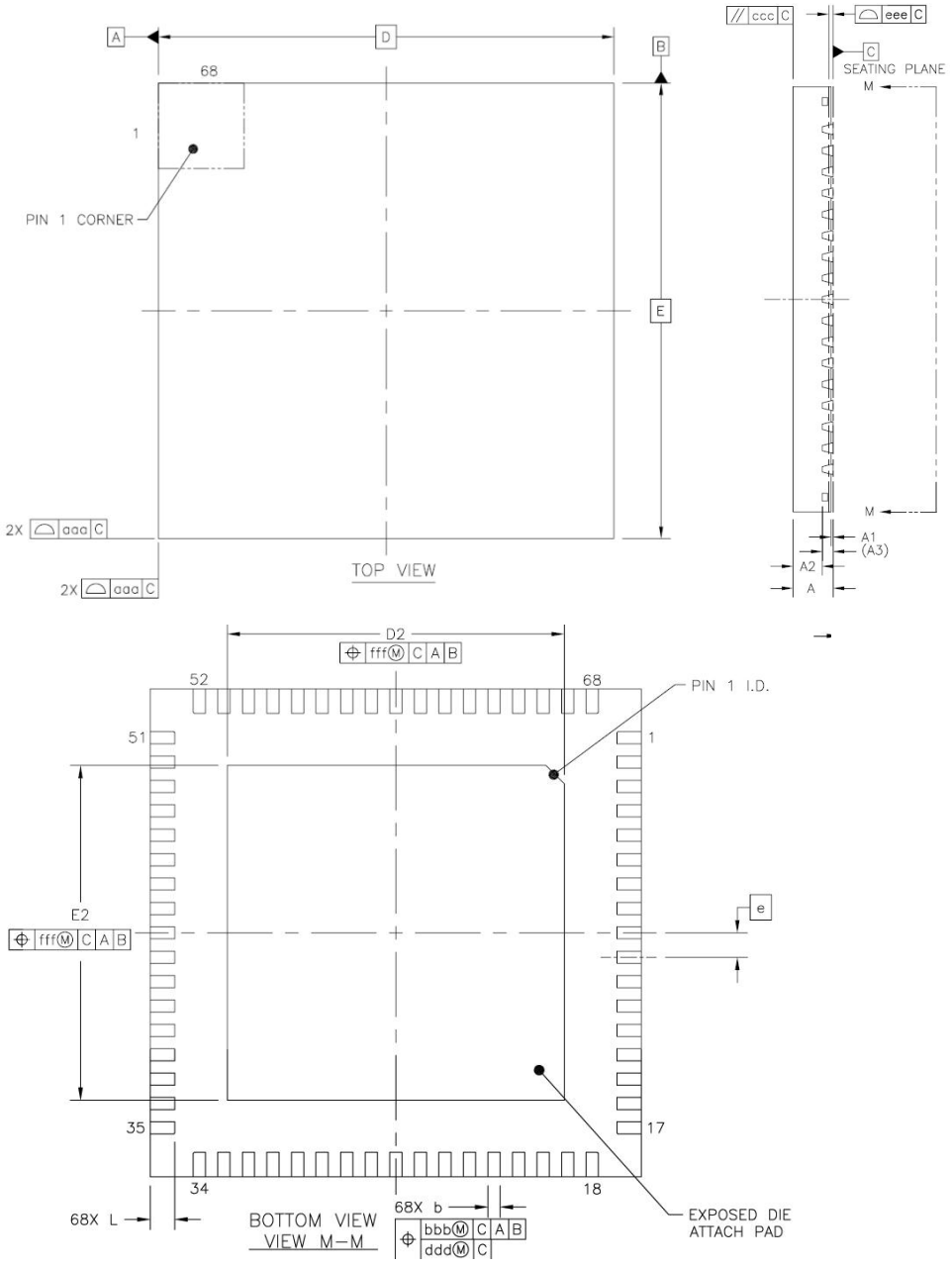
图表 18：V6201E1 封装尺寸

表格 39：V6201E1 封装参数

PACKAGE TYPE		QFN		
PIN COUNT		48		
DESCRIPTION	SYMBOL	MILLIMETER		
		MIN	NOM	MAX
TOTAL THICKNESS	A	0.70	0.75	0.80
STAND OFF	A1	0	0.035	0.050
MOLD THICKNESS	A2	–	0.55	0.57
MATERIAL THICKNESS	A3	–	0.203 REF	–
PACKAGE SIZE	D	–	6 BSC	–
	E	–	6 BSC	–
EP SIZE	D1	4.6	4.7	4.8
	E1	4.6	4.7	4.8
LEAD LENGTH	L	0.30	0.4	0.50
LEAD PITCH	e	0.4 BSC		
LEAD WIDTH	b	0.15	0.20	0.25
LEAD POSITION OFFSET	aaa	0.07		
LEAD COPLANARITY	bbb	0.08		
PACKAGE EDGE PROFILE	ccc	0.10		
MOLD FLATNESS	ddd	0.10		
EP POSITION OFFSET	eee	0.10		
	fff	0.05		

19.2. V6211E1 封装

V6211E1 采用 8mmX8mm，68-pin QFN 封装，封装尺寸图如下：



图表 19：V6211E1 封装尺寸

表格 40：V6211E1 封装参数

DESCRIPTION		SYMBOL	MILLIMETER		
			MIN	NOM	MAX
TOTAL THICKNESS		A	0.7	0.75	0.8
STAND OFF		A1	0	0.035	0.05
MOLD THICKNESS		A2	---	0.55	0.57
L/F THICKNESS		A3	0.203 REF		
LEAD WIDTH		b	0.15	0.20	0.25
BODY SIZE	X	D	8 BSC		
	Y	E	8 BSC		
LEAD PITCH		e	0.4 BSC		
EP SIZE	X	D2	5.39	5.49	5.59
	Y	E2	5.39	5.49	5.59
LEAD LENGTH		L	0.30	0.40	0.50
PACKAGE EDGE TOLERANCE		aaa	0.10		
MOLD FLATNESS		ccc	0.10		
LEAD OFFSET		bbb	0.07		
		ddd	0.05		
COPLANARITY		eee	0.08		
EXPOSED PAD OFFSET		fff	0.10		

声明

杭州万高科技股份有限公司保留对本手册所涉及的产品及相关的技术信息进行补正或更新的权利。使用本手册时，请您从我们的销售渠道或登录公司网站 http://www.vangotech.com 获取最新信息。