Gowin RiscV_AE350_SOC 软件编程

用户手册

MUG1029-1.0, 2023-09-12

免责声明

本文档并未授予任何知识产权的许可，并未以明示或暗示，或以禁止反言或其它方式授予任何知识产权许可。除高云半导体在其产品的销售条款和条件中声明的责任之外，高云半导体概不承担任何法律或非法律责任。高云半导体对高云半导体产品的销售和／或使用不作任何明示或暗示的担保，包括对产品的特定用途适用性、适销性或对任何专利权、版权或其它知识产权的侵权责任等，均不作担保。高云半导体对文档中包含的文字、图片及其它内容的准确性和完整性不承担任何法律或非法律责任，高云半导体保留修改文档中任何内容的权利，恕不另行通知。高云半导体不承诺对这些文档进行适时的更新。

版本信息

日期	版本	描述
2023/09/12	1.0	初始版本。

目录

图目录

表目录

1 关于本手册

1.1 手册内容

本手册主要描述 Gowin® RiscV_AE350_SOC 的软件编程模式，内存映射，中断处理，系统启动方式，Cache，外部设备（UART、GPIO、I2C、SPI、RTC、PIT、WDT、DMA、PLMT、SMU）的特征、结构框图、功能描述、寄存器和驱动函数，DSP，嵌入式实时操作系统（FreeRTOS、uC/OS-III、RT-Thread Nano 和 Zephyr），基准测试程序和应用程序等。

1.2 术语、缩略语

本手册中的相关术语、缩略语及相关释义如表 1-1 所示。

表 1-1 术语、缩略语

术语、缩略语	全称	含义
AHB	Advanced High Performance Bus	高性能总线
APB	Advanced Peripheral Bus	高级外设总线
BMC	Bus Matrix Controller	总线矩阵控制器
CLIC	Core Local Interrupt Controller	内核局部中断控制器
CRC	Cyclic Redundancy Check	循环冗余校验
CSR	Control and Status Register	控制和状态寄存器
DDR	Double Data Rate Memory	双倍数据速率存储器
DLM	Data Local Memory	数据局部存储器
DMA	Direct Memory Access	直接内存访问
DSP	Digital Signal Processor	数字信号处理器
FIFO	First In First Out	先进先出
GPIO	Gowin Programmable IO	Gowin 可编程通用管脚
HSP	Hardware Stack Protection	硬件堆栈保护

术语、缩略语	全称	含义
I2C	Inter-Integrated Circuit Bus	内部集成电路总线
ILM	Instruction Local Memory	指令局部存储器
LRU	Least Recently Used	近期最少使用替换算法
LSB	Least Significant Bit	最低有效位
MCU	Micro Controller Unit	微控制器单元
MM	Memory Management	内存管理
MSB	Most Significant Bit	最高有效位
PFM	Performance Monitor	性能监视器
PIT	Programmable Interval Timer	可编程间隔定时器
PLIC	Platform Level Interrupt Controller	平台级中断控制器
PLMT	Platform Level Machine Timer	平台级机器模式定时器
PMP	Physical Memory Protection	物理内存保护
PWM	Pulse Width Modulation	脉冲宽度调制
RISC-V	Reduced Instruction Set Computer V	第五代精简指令集计算机
ROM	Read Only Memory	只读存储器
RTC	Real Time Clock	实时时钟
RTOS	Real Time Operating System	实时操作系统
SaG	Scattering and Gathering	散布和收集
SMU	System Manage Unit	系统管理单元
SPI	Serial Peripheral Interface	串行外设接口
SRAM	Static Random Access Memory	静态随机存取存储器
TPM	Timing Parameter Multiplier	时序参数乘法器
UART	Universal Synchronous and Asynchronous Receiver/Transmitter	通用同步和异步接收器/发射器
WDT	Watch Dog Timer	看门狗定时器
WFI	Wait for Interrupt	等待中断唤醒

1.3 技术支持与反馈

高云®半导体提供全方位技术支持，在使用过程中如有疑问或建议，可直接与公司联系：

网址：www.gowinsemi.com.cn

E-mail：support@gowinsemi.com

Tel：+86 755 8262 0391

2 软件编程模式

Gowin RiscV_AE350_SOC 支持以下几种软件编程模式：

- 单片机软件编程
- 嵌入式 RTOS 软件编程
- DSP 软件编程

2.1 单片机软件编程

Gowin RiscV_AE350_SOC 单片机软件编程函数库如表 2-1 所示。

表 2-1 单片机软件编程

函数库文件	描述
系统定义	
ae350.h	定义系统时钟、中断源、外设寄存器、内存地址映射、设备名称等
ae350.c	定义平台初始化、系统初始化、系统启动等
cache.c cache.h	定义一级 Cache 访问方法
core_v5.h	定义 RISC-V 内核
csr.h	定义 CSR
interrupt.c interrupt.h	定义中断处理函数
loader.c	定义 BUILD_BURN 启动引导方法
platform.h	定义外设名称
plic.h	定义 PLIC
clic.h	定义 CLIC
reset.c	定义复位处理方法

函数库文件	描述
start.S	定义系统启动方法
trap.c	定义机器异常处理方法
initfini.c	定义初始化例程和清理例程
config.h	定义系统配置
外设驱动定义	
dma_ae350.c dma_ae350.h	定义 DMA 驱动函数
gpio_ae350.c gpio_ae350.h Driver_GPIO.h	定义 GPIO 驱动函数
i2c_ae350.c i2c_ae350.h Driver_I2C.h	定义 I2C 驱动函数
pit_ae350.c pit_ae350.h Driver_PIT.h	定义 PIT Timer 驱动函数
pwm_ae350.c pwm_ae350.h Driver_PWM.h	定义 PWM 驱动函数
rtc_ae350.c rtc_ae350.h Driver_RTC.h	定义 RTC 驱动函数
spi_ae350.c spi_ae350.h Driver_SPI.h	定义 SPI 驱动函数
uart_ae350.c uart_ae350.h Driver_UART.h	定义 UART 驱动函数
wdt_ae350.c wdt_ae350.h Driver_WDT.h	定义 Watch Dog 驱动函数
Driver_Common.h	定义共用的参数和结构
SaG 定义	
ae350-ddr.sag	程序运行于 DDR/SRAM 的 SaG 文件和 LD 文件

函数库文件	描述
ae350-ddr.ld	
ae350-ilm.sag ae350-ilm.ld	程序运行于 ILM 的 SaG 文件和 LD 文件
ae350-xip.sag ae350-xip.ld	程序运行于 Flash/ROM 的 SaG 文件和 LD 文件
应用程序定义	
delay.c delay.h	定义延时和计时函数
mm.c mm.h	定义堆栈申请与释放的内存管理函数
printf.c read.c	简单地重定义标准输入输出函数，用于简单地输入输出整型、字符型、字符串数据
uart.c uart.h	定义 UART 初始化，发送和接收函数
stubs\close.c stubs\exit.c stubs\fstat.c stubs\isatty.c stubs\lseek.c stubs\read.c stubs\sbrk.c stubs\stub.h stubs\write.c stubs\write_hex.c	重定义标准输入输出函数，用于输入输出浮点型数据

2.2 嵌入式 RTOS 软件编程

Gowin RiscV_AE350_SOC 支持以下几种嵌入式 RTOS 软件编程：

- FreeRTOS
- uC/OS-III
- RT-Thread Nano
- Zephyr

2.3 DSP 软件编程

Gowin RiscV_AE350_DSP 支持 DSP 软件编程，提供完整的 DSP 软件编程 API 函数，函数代码精简，方便用户快速开发 DSP 系统。

DSP 软件编程 API 函数包括向量、矩阵和复向量等基本运算，以及滤波和变换函数等复杂运算。

3 内存映射

Gowin RiscV_AE350_SOC 内存映射定义如表 3-1 所示。内存映射定义位于 bsp\ae350\ae350.h。

表 3-1 内存映射

地址		描述
起始地址	结束地址	
0x00000000	0x7FFFFFFF	DDR/SRAM Data Memory
0x80000000	0x8FFFFFFF	ROM/Flash Instruction Memory
0xA0000000	0xA000FFFF	ILM
0xA0200000	0xA020FFFF	DLM
0xC0000000	0xC00FFFFF	BMC
0xE0000000	0xE00FFFFF	AHB Decoder
0xE4000000	0xE5FFFFFF	PLIC
0xE6000000	0xE60FFFFF	Machine Timer
0xE6400000	0xE67FFFFF	PLIC-SWINT
0xE6800000	0xE68FFFFF	Debug Module
0xF0000000	0xF00FFFFF	APB Bridge
0xF0100000	0xF01FFFFF	SMU
0xF0200000	0xF02FFFFF	UART1
0xF0300000	0xF03FFFFF	UART2
0xF0400000	0xF04FFFFF	PIT
0xF0500000	0xF05FFFFF	WDT
0xF0600000	0xF06FFFFF	RTC
0xF0700000	0xF07FFFFF	GPIO
0xF0A00000	0xF0AFFFFF	I2C

地址		描述
起始地址	结束地址	
0xF8000000	0xFBFFFFFF	Extended APB Slave Interface
0xF0C00000	0xF0CFFFFF	DMA
0xF0F00000	0xF0FFFFFF	SPI
0xE8000000	0xEFFFFFFF	Extended AHB Slave Interface

4 中断处理

4.1 中断处理

Gowin RiscV_AE350_SOC 的中断处理包括两种方式：局部中断处理和全局中断处理。局部中断处理是指直接进入 AE350 RISC-V 处理器进行处理的中断；全局中断处理是指在进入 AE350 RISC-V 处理器进行处理前，需要通过 PLIC 进行仲裁的外部中断。RiscV_AE350_SOC 的中断处理方式如表 4-1 所示。

AE350 RISC-V 处理器支持的局部中断处理包括不可屏蔽中断处理（nmi）、机器模式定时器中断处理（mtip）和机器模式软件中断处理（msip）。

AE350 RISC-V 处理器支持的全局中断处理包括所有来自 PLIC 仲裁的外部中断处理（meip 和 seip）。

表 4-1 中断处理方式

中断处理方式	描述
nmi	WDT interrupt
mtip	Machine timer interrupt
msip	Machine software interrupt
meip	Machine PLIC interrupt
seip	Supervisor PLIC interrupt

4.2 全局中断处理

Gowin RiscV_AE350_SOC 在二次实例化 PLIC 模块时，将所有绑定到位置 0 的中断源作为软件中断控制器（PLIC_SW）。PLIC 通过编程寄存器产生中断信号用于产生软件中断。

Gowin RiscV_AE350_SOC 全局中断处理方式包括向量式中断处理和非向量式中断处理。例如软件编程应用程序 ae350_demo 使用非向量式中

断处理，ae350_vectored 使用向量式中断处理。非向量式中断处理与向量式中断处理的不同之处位于 bsp\ae350\start.S、trap.c、interrupt.h 和 interrupt.c。

Gowin RiscV_AE350_SOC 全局中断处理的中断源及其所对应的中断信号和中断处理函数如表 4-2 所示。

中断源与中断信号定义位于 bsp\ae350\ae350.h。中断处理函数定义位于 bsp\ae350\interrupt.c。

表 4-2 PLIC 全局中断处理

中断源	中断信号	中断处理函数	描述
1	IRQ_RTCPERIOD_SOURCE	rtc_period_irq_handler	RTC period interrupt
2	IRQ_RTCALARM_SOURCE	rtc_alarm_irq_handler	RTC alarm interrupt
3	IRQ_PIT_SOURCE	pit_irq_handler	PIT interrupt
4	IRQ_GP0_SOURCE	gp0_irq_handler	User defined interrupt
5	IRQ_SPI2_SOURCE	spi_irq_handler	SPI2 interrupt
6	IRQ_I2C_SOURCE	i2c_irq_handler	I2C interrupt
7	IRQ_GPIO_SOURCE	gpio_irq_handler	GPIO interrupt
8	IRQ_UART1_SOURCE	uart1_irq_handler	UART1 interrupt
9	IRQ_UART2_SOURCE	uart2_irq_handler	UART2 interrupt
10	IRQ_DMA_SOURCE	dma_irq_handler	DMAC interrupt
11	IRQ_GP1_SOURCE	gp1_irq_handler	User defined interrupt
12	IRQ_GP2_SOURCE	gp2_irq_handler	User defined interrupt
13	IRQ_GP3_SOURCE	gp3_irq_handler	User defined interrupt
14	IRQ_GP4_SOURCE	gp4_irq_handler	User defined interrupt
15	IRQ_GP5_SOURCE	gp5_irq_handler	User defined interrupt
16	IRQ_GP6_SOURCE	gp6_irq_handler	User defined interrupt

中断源	中断信号	中断处理函数	描述
17	IRQ_GP7_SOURCE	gp7_irq_handler	User defined interrupt
18	IRQ_GP8_SOURCE	gp8_irq_handler	User defined interrupt
19	IRQ_GP9_SOURCE	gp9_irq_handler	User defined interrupt
20	IRQ_GP10_SOURCE	gp10_irq_handler	User defined interrupt
21	IRQ_GP11_SOURCE	gp11_irq_handler	User defined interrupt
22	IRQ_GP12_SOURCE	gp12_irq_handler	User defined interrupt
23	IRQ_GP13_SOURCE	gp13_irq_handler	User defined interrupt
24	IRQ_GP14_SOURCE	gp14_irq_handler	User defined interrupt
25	IRQ_GP15_SOURCE	gp15_irq_handler	User defined interrupt
26	IRQ_STANDBY_SOURCE	standby_irq_handler	SMU standby request interrupt
27	IRQ_WAKEUP_SOURCE	wakeup_irq_handler	SMU wakeup interrupt

5 系统启动方式

Gowin RiscV_AE350_SOC 支持以下几种系统启动方式：

- BUILD_LOAD
- BUILD_BURN
- BUILD_XIP

Gowin RiscV_AE350_SOC 系统启动方式，以及对应的 SaG 文件如表 5-1 所示。系统启动方式定义位于 bsp\config\config.h。

表 5-1 系统启动方式

启动方式	SaG 文件	描述
BUILD_LOAD	ae350-ilm.sag	The program is loaded in ILM by GDB or eBIOS.
	ae350-ddr.sag	The program is loaded in DDR/SRAM memory by GDB or eBIOS.
BUILD_BURN	ae350-ddr.sag	The Program is burned to the Flash/ROM memory, but run in DDR/SRAM memory.
BUILD_XIP	ae350-xip.sag	The program is burned to the Flash/ROM memory, and run in Flash/ROM memory.

6 Cache

6.1 简介

Gowin RiscV_AE350_SOC 支持 L1 Cache，包括指令缓存（I-Cache）和数据缓存（D-Cache）。I-Cache 定义如表 6-1 所示，D-Cache 定义如表 6-2 所示。

Cache size = Cache lines per way X Ways X Line size

表 6-1 I-Cache 定义

属性	配置
Size	32 KB
Cache lines per way	256
Ways	Direct-mapped, 4-way
Lines size (bytes)	32
Soft Error Protection	ECC, single-error and double error correction
Replacement Policy	LRU

表 6-2 D-Cache 定义

属性	配置
Size	32 KB
Cache lines per way	256
Ways	Direct-mapped, 4-way
Lines size (bytes)	32
Soft Error Protection	ECC, single-error and double error detection
Replacement Policy	LRU

6.2 控制寄存器

Cache 控制寄存器定义如表 6-3 所示。

表 6-3 Cache Control Register

名称	位置	描述	类型	初始值
IC_EN	[0]	Controls if the instruction cache is enabled or not. 0: I-Cache is disabled 1: I-Cache is enabled	RW	0
DC_EN	[1]	Controls if the data cache is enabled or not. 0: D-Cache is disabled. 1: D-Cache is enabled	RW	0
IC_ECCEN	[3:2]	Parity/ECC error checking enable control for the instruction cache. 0: Disable parity/ECC 1: Reserved 2: Generate exceptions only on uncorrectable parity/ECC errors 3: Generate exceptions on any type of parity/ECC errors	RW	0
DC_ECCEN	[5:4]	Parity/ECC error checking enable control for the data cache. 0: Disable parity/ECC 1: Reserved 2: Generate exceptions only on uncorrectable parity/ECC errors 3: Generate exceptions on any type of parity/ECC errors	RW	0
EC_RWECC	[6]	Controls diagnostic accesses of ECC codes of the instruction cache RAMs. It is set to enable CCTL operations to access the ECC codes. This bit can be set for injecting ECC errors to test	RW	0

名称	位置	描述	类型	初始值
		the ECC handler. 0: Disable diagnostic accesses of ECC codes 1: Enable diagnostic accesses of ECC codes		
DC_RWECC	[7]	Controls diagnostic accesses of ECC codes of the data cache RAMs. It is set to enable CCTL operations to access the ECC codes. This bit can be set for injecting ECC errors to test the ECC handler. 0: Disable diagnostic accesses of ECC codes 1: Enable diagnostic accesses of ECC codes	RW	0
CCTL_SUEN	[8]	Enable bit for Superuser-mode and User-mode software to access ucctlbeginaddr and ucctlcommand CSRs. 0: Disable ucctlbeginaddr and ucctlcommand accesses in S/U mode 1: Enable ucctlbeginaddr and ucctlcommand accesses in S/U mode	RW	0
IC_FIRST_WORLD	[11]	I-Cache miss allocation filling policy. 0: Cache line data is returned critical (double) word first 1: Cache line data is returned the lowest address (double) word first	RO	-

6.3 CCTL 命令

Cache CCTL 命令定义如表 6-4 所示。Cache CCTL 操作定义位于 bsp\ae350\cache.c。

表 6-4 Cache CCTL Command

索引	配置	命令	类型
0	0b00_000	L1D_VA_INVAL	VA
1	0b00_001	L1D_VA_WB	VA
2	0b00_010	L1D_VA_WBINVAL	VA
3	0b00_011	L1D_VA_LOCK	VA
4	0b00_100	L1D_VA_UNLOCK	VA
6	0b00_110	L1D_WBINVAL_ALL	-
7	0b00_111	L1D_WB_ALL	-
8	0b01_000	L1I_VA_INVAL	VA
11	0b01_011	L1I_VA_LOCK	VA
12	0b01_100	L1I_VA_UNLOCK	VA
16	0b10_000	L1D_IX_INVAL	Index
17	0b10_001	L1D_IX_WB	Index
18	0b10_010	L1D_IX_WBINVAL	Index
19	0b10_011	L1D_IX_RTAG	Index
20	0b10_100	L1D_IX_RDATA	Index
21	0b10_101	L1D_IX_WTAG	Index
22	0b10_110	L1D_IX_WDATA	Index
23	0b10_111	L1D_INVAL_ALL	-
24	0b11_000	L1I_IX_INVAL	Index
27	0b11_011	L1I_IX_RTAG	Index
28	0b11_100	L1I_IX_RDATA	Index
29	0b11_101	L1I_IX_WTAG	Index
30	0b11_110	L1I_IX_WDATA	Index

6.4 配置方法

如果开启 Cache 则可以加速系统启动和程序运行。

Cache 采用 LRU 算法，从主存预取指令和替换指令。如果 AE350 RISC-V 处理器每次都能击中 Cache 中的预取指令，则可以提高系统启动与程序运行的速度和效率。如果 AE350 RISC-V 处理器未击中 Cache 的预取指令，则 Cache 使用 LRU 算法与主存替换指令。

Cache 配置定义位于 bsp\config\config.h 和 bsp\ae350\ae350.c。

7 UART

7.1 简介

Gowin RiscV_AE350_SOC 包含两个 UART 控制器。

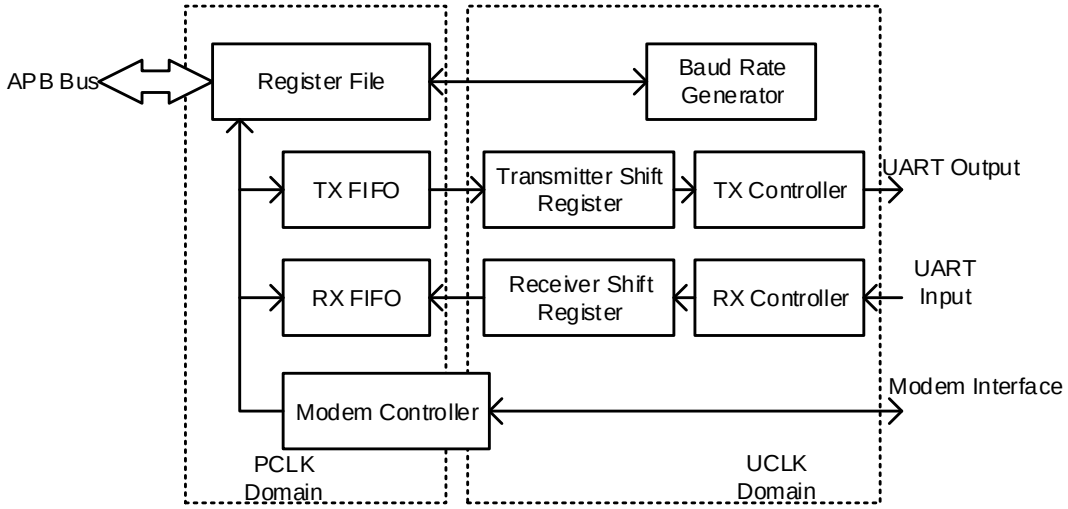
7.1.1 特征

- 通过 AMBA 2.0 总线接口访问寄存器
- 支持硬件 64 字节的发送和接收 FIFO
- 支持可编程过采样频率（8x 和 16x）
- 编程时序兼容 16C550A UART
  - 支持每字节 5~8 位宽数据
  - 支持 1、1.5 和 2 位宽停止位
  - 支持奇、偶校验和固定校验位
  - 支持硬件流控制（CTS/RTS）
  - 支持 DMA 传输功能
  - 支持可编程的传输速率
  - 支持调制解调器控制接口
  - 支持完整的状态报告功能
  - 支持线中止、奇偶校验错误、帧错误、数据溢出检测

7.1.2 结构框图

UART 结构框图如图 7-1 所示。

图 7-1 UART 结构框图



7.1.3 功能描述

UART 是一个串行通信控制器，为外部设备或调制解调器提供异步串行接口。UART 包括，发送模块、接收模块、波特率产生模块、调制解调器控制模块，以及寄存器文件组和 APB 总线接口。

7.2 寄存器定义

7.2.1 寄存器定义

UART 寄存器定义如表 7-1 所示。UART 寄存器定义位于 bsp\ae350\ae350.h。

表 7-1 寄存器定义

地址偏移	寄存器名称	描述
0x00	IDREV	ID and Revision Register
0x04~0x0C	-	Reserved
0x10	CFG	Hardware Configuration Register
0x14	OSCR	Over Sample Control Register
0x18~0x1C	-	Reserved
0x20	DLAB = 0	
	RBR	Receiver Buffer Register (Read only)
	THR	Transmitter Holding Register (Write only)
	DLAB = 1	
	DLL	Divisor Latch LSB

地址偏移	寄存器名称	描述
0x24	DLAB = 0	
	IER	Interrupt Enable Register
	DLAB = 1	
	DLM	Divisor Latch MSB
0x28	IIR	Interrupt Identification Register (Read only)
	FCR	FIFO Control Register (Write only)
0x2C	LCR	Line Control Register
0x30	MCR	Modem Control Register
0x34	LSR	Line Status Register
0x38	MSR	Modem Status Register
0x3C	SCR	Scratch Register

7.2.2 寄存器描述

以下各节详细描述 UART 寄存器定义。

寄存器类型缩略语概括如下：

- RO: Read-only
- WO: Write-only
- R/W: Readable and writable
- W1C: Write 1 to clear
- RC: Read clear

ID 和 Revision 寄存器（0x00）

ID 和 Revision 寄存器用于保存 ID 和 Revision 编号，初始值依赖于所用版本。ID 和 Revision 寄存器定义如表 7-2 所示。

表 7-2 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:16	RO	ID number for UART	0x0201
Major	15:4	RO	Major revision number	Revision dependent
Minor	3:0	RO	Minor revision number	Revision dependent

Hardware Configuration 寄存器（0x10）

Hardware Configuration 寄存器定义如表 7-3 所示。

表 7-3 Hardware Configuration Register

Name	Bit	Type	Description	Reset
-	31:2	-	Reserved	0x00
FIFO_DEPTH	1:0	RO	The depth of RXFIFO and TXFIFO 0: 16-byte FIFO 1: 32-byte FIFO 2: 64-byte FIFO 3: 128-byte FIFO	Configuration dependent

Over Sample Control 寄存器（0x14）

Over Sample Control 寄存器定义如表 7-4 所示。

表 7-4 Over Sample Control Register

Name	Bit	Type	Description	Reset
-	31:5	-	Reserved	0x0
OSC	4:0	R/W	Over-sample control The value must be an even number; any odd value writes to this field will be converted to an even value. OSC = 0: The over-sample ratio is 32 OSC ≤ 8: The over-sample ratio is 8 8 < OSC < 32: The over-sample ratio is OSC	0x10

Receiver Buffer 寄存器（当 DLAB = 0）（0x20）

Receiver Buffer 寄存器包括两种模式，FIFO 模式和 BUFFER 模式。FIFO Control 寄存器的 Bit0 位（FIFOE）控制这两种模式的选择，具体参见 FIFO Control 寄存器（0x28）。当 FIFOE 为 1 时（FIFO 模式），Receiver Buffer 寄存器为 RXFIFO；当 FIFOE 为 0 时（BUFFER 模式），Receiver Buffer 寄存器为一个字节缓存区。

Receiver Buffer 寄存器定义如表 7-5 所示。

表 7-5 Receiver Buffer Register (when DLAB = 0)

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
RBR	7:0	RO	Receive data read port	0x0

Transmitter Holding 寄存器（当 DLAB = 0）（0x20）

Transmitter Holding 寄存器包括两种模式，FIFO 模式和 BUFFER 模式。FIFO Control 寄存器的 Bit0 位（FIFOE）控制这两种模式的选择 具体参见 FIFO Control 寄存器（0x28）。当 FIFOE 为 1 时（FIFO 模式），Transmitter Holding 寄存器为 TXFIFO；当 FIFOE 为 0 时（BUFFER 模式），Transmitter Holding 寄存器为一个字节缓存区。

Transmitter Holding 寄存器定义如表 7-6 所示。

表 7-6 Transmitter Holding Register (when DLAB = 0)

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
THR	7:0	WO	Transmit data write port	0x0

Interrupt Enable 寄存器（when DLAB = 0）（0x24）

Interrupt Enable 寄存器定义如表 7-7 所示。

表 7-7 Interrupt Enable Register (when DLAB = 0)

Name	Bit	Type	Description	Reset
-	31:4	-	Reserved	0x0
EMSI	3	R/W	Enable modem status interrupt The interrupt asserts when the status of one of the following occurs: The status of modem_rin, modem_dcdn, modem_dsrn or modem_ctsn (If the auto-cts mode is disabled) has been changed. If the auto-cts mode is enable (MCR bit4 (AFE) = 1), modem_ctsn would be used to control the transmitter.	0x0
ELSI	2	R/W	Enable receiver line status interrupt	0x0
ETHEI	1	R/W	Enable transmitter holding register interrupt	0x0
ERBI	0	R/W	Enable received data available interrupt and the character timeout interrupt 0: Disable 1: Enable	0x0

Divisor Latch LSB（当 DLAB = 1）（0x20）

Divisor Latch LSB 保存除数值用于从 UART 时钟源（uclk）产生采样

时钟。Divisor Latch LSB 的大小为 16 位（2 个字节），寄存器保存 Devisor Latch 的最低有效字节。Divisor Latch LSB 的有效值在 1 和 65535（2^{16}-1）之间，包括 1 和 65535。

Divisor Latch LSB 定义如表 7-8 所示。

表 7-8 Divisor Latch LSB (when DLAB = 1)

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
DLL	7:0	R/W	Least significant byte of the Divisor Latch	0x1

Divisor Latch MSB（当 DLAB = 1）（0x24）

Divisor Latch MSB 保存除数值用于从 UART 时钟源（uclk）产生采样时钟。Divisor Latch MSB 的大小为 16 位（2 个字节），寄存器保存 Devisor Latch MSB 的最高有效字节。Divisor Latch MSB 的有效值在 1 和 65535（2^{16}-1）之间，包括 1 和 65535。

Divisor Latch MSB 定义如表 7-9 所示。

表 7-9 Divisor Latch MSB (when DLAB = 1)

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
DLM	7:0	R/W	Most significant byte of the Divisor Latch	0x0

Interrupt Identification 寄存器（0x28）

Interrupt Identification 寄存器定义如表 7-10 所示。

表 7-10 Interrupt Identification Register

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
FIFOED	7:6	RO	FIFOs enabled These two bits are 1 when bit0 of the FIFO Control Register (FIFOE) is set to 1	0x0
-	5:4	-	Reserved	0x0
INTRID	3:0	RO	Interrupt ID See Table 7-11 for encodings	0x1

表 7-11 Interrupt Control Type

Interrupt Identification Register					Interrupt Type	Interrupt Source Description	Interrupt Reset Method
Bit3	Bit2	Bit1	Bit0	Priority Level			
0	0	0	1	None	None	None	None
0	1	1	0	1	Receiver Line status	Overrun errors, parity errors, framing errors, or line breaks	Read the Line Status Register (LSR)
0	1	0	0	2	Received data available	If FIFOE is disabled, there is one received data available in the RBR. If FIFOE is enabled, the numbers of received data available reach the trigger level (RFIFOT). The interrupt signal will stay active until the number of data available becoming smaller than the trigger level.	Read the Receiver Buffer Register (RBR)
1	1	0	0	2	Character timeout	When FIFOE is enabled and no character has been removed from or input to receive FIFO and there is at least one character in receive FIFO during the last four character times.	Read the Receiver Buffer Register (RBR)
0	0	1	0	3	Transmitter Holding Register empty	If FIFOE is disabled, the 1-byte THR is empty. If FIFOE is enabled, the whole 16-byte transmit FIFO is empty.	Write the Transmitter Holding Register (THR) or Read the Interrupt Identification Register (IIR).
0	0	0	0	4	Modem status	The Modem Status Register (MSR) bit [3:0] is not 0. One of the following events occurred: Clear To Send (CTS), Data Set Ready (DSR), Ring Indicator (RI), or Data Carrier Detect (DCD)	Read the Modem Status Register (MSR)

FIFO Control 寄存器（0x28）

FIFO Control 寄存器定义如表 7-12 所示。

表 7-12 FIFO Control Register

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
RFIFOT	7:6	WO	Receiver FIFO trigger level Please refer to Table 7-13	0x0
TFIFOT	5:4	WO	Transmitter FIFO trigger level Please refer to Table 7-14	0x0
DMAE	3	WO	DMA enable 0: Disable 1: Enable	0x0
TFIFORST	2	WO	Transmitter FIFO reset Write 1 to clear all bytes in the TXFIFO and resets its counter. The Transmitter Shift Register is not cleared. This bit will automatically be cleared.	0x0
RFIFORST	1	WO	Receiver FIFO reset Write 1 to clear all bytes in the RXFIFO and resets its counter. The Receiver Shift Register is not cleared. This bit will automatically be cleared.	0x0
FIFOE	0	WO	FIFO enable Write 1 to enable both the transmitter and receiver FIFOs. The FIFOs are reset when the value of this bit toggles.	0x0

接收 FIFO 触发等级定义如表 7-13 所示。

表 7-13 Receive FIFO Trigger Level

RFIFOT Value	RXFIFO Trigger Level			
	16-byte RXFIFO	32-byte RXFIFO	64-byte RXFIFO	128-byte RXFIFO
0	Not empty	Not empty	Not empty	Not empty
1	More than 3	More than 7	More than 15	More than 31
2	More than 7	More than 15	More than 31	More than 63
3	More than 13	More than 27	More than 55	More than 111

发送 FIFO 触发等级定义如表 7-14 所示。

表 7-14 Transmit FIFO Trigger Level

TFIFOT Value	TXFIFO Trigger Level			
	16-byte TXFIFO	32-byte TXFIFO	64-byte TXFIFO	128-byte TXFIFO
0	Not full	Not full	Not full	Not full
1	Less than 12	Less than 24	Less than 48	Less than 96
2	Less than 8	Less than 16	Less than 32	Less than 64
3	Less than 4	Less than 8	Less than 16	Less than 32

Line Control 寄存器（0x2C）

Line Control 寄存器定义如表 7-15 所示。

表 7-15 Line Control Register

Name	Bit	Type	Description	Reset
-	[31:8]	-	Reserved	0x0
DLAB	7	R/W	Divisor latch access bit	0x0
BC	6	R/W	Break control	0x0
SPS	5	R/W	Stick parity 1: Parity bit is constant 0 or 1, depending on bit4 (EPS). 0: Disable the sticky bit parity. Please refer to Table 7-16	0x0
EPS	4	R/W	Even parity select 1: Even parity (an even number of logic-1 is in the data and parity bits). 0: Odd parity. Please refer to Table 7-16.	0x0
PEN	3	R/W	Parity enable When this bit is set, a parity bit is generated in transmitted data before the first STOP bit and the parity bit would be checked for the received data. Please refer to Table 7-16	0x0
STB	2	R/W	Number of STOP bits 0: 1 bits 1: The number of STOP bit is based on the WLS setting When WLS = 0, STOP bit is 1.5 bits	0x0

Name	Bit	Type	Description	Reset
			When WLS = 1, 2, 3, STOP bit is 2 bits	
WLS	1:0	R/W	Word length setting 0: 5 bits 1: 6 bits 2: 7 bits 3: 8 bits	0x0

Parity Bit Selection 定义如表 7-16 所示。

表 7-16 Parity Bit Selection

PEN (bit3)	SPS (bit5)	EPS (bit4)	Parity Bit
0	X	X	No parity bit
1	0	0	Parity is odd
1	0	1	Parity is even
1	1	01	Parity bit is always 1
1	1	-	Parity bit is always 0

Modem Control 寄存器（0x30）

Modem Control 寄存器用于控制调制解调器状态信号输出，以及回环模式和自动流控制。

Modem Control 寄存器定义如表 7-17 所示。

表 7-17 Modem Control Register

Name	Bit	Type	Description	Reset
-	31:6	-	Reserved	0x0
AFE	5	R/W	Auto flow control enable 0: Disable 1: The auto-CTS and auto-RTS setting is based on the RTS bit setting: When RTS = 0, auto-CTS only When RTS = 1, auto-CTS and auto-RTS	0x0
LOOP	4	R/W	Enable loopback mode 0: Disable 1: Enable	0x0
OUT2	3	R/W	User-defined output 2	0x0

Name	Bit	Type	Description	Reset
			This bit controls the uart_out2n output. 0: The uart_out2n output signal will be driven HIGH 1: The uart_out2n output signal will be driven LOW	
OUT1	2	R/W	User-defined output 1 This bit controls the uart_out1n output. 0: The uart_out1n output signal will be driven HIGH 1: The uart_out1n output signal will be driven LOW	0x0
RTS	1	R/W	Request to send This bit controls the modem_rstn output 0: The modem_rstn output signal will be driven HIGH 1: The modem_rstn output signal will be driven LOW	0x0
DTR	0	R/W	Data terminal ready This bit controls the modem_dtrn output. 0: The modem_dtrn output signal will be driven HIGH 1: The modem_dtrn output signal will be driven LOW	0x0

Line Status 寄存器（0x34）

Line Status 寄存器，报告发送器和接收器的状态。Line Status 寄存器定义如表 7-18 所示。

表 7-18 Line Status Register

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
ERRF	7	RO	Error in RXFIFO In the FIFO mode, this bit is set when there is at least one parity error, framing error, or line break associated with data in the RXFIFO. It is cleared when this register is read and there is no more error for the rest of data in the RXFIFO	0x0
TEMT	6	RO	Transmitter empty This bit is 1 when the THR (TXFIFO in the FIFO	0x1

Name	Bit	Type	Description	Reset
			mode) and the Transmitter Shift Register (TSR) are both empty. Otherwise, it is zero.	
THRE	5	RO	Transmitter Holding Register empty This bit is 1 when the THR (TXFIFO in the FIFO mode) is empty. Otherwise, it is zero. If the THRE interrupt is enabled, an interrupt is triggered when THRE becomes 1.	0x1
LBreak	4	RO	Line break This bit is set when the uart_sin input signal was held LOW for longer than the time for a full-word transmission. A full-word transmission is the transmission of the START, data, parity, and STOP bits. It is cleared when this register is read. In the FIFO mode, this bit indicates the line break for the received data at the top of the RXFIFO.	0x0
FE	3	RO	Framing error This bit is set when the received STOP bit is not HIGH. It is cleared when this register is read. In the FIFO mode, this bit indicates the framing error for the received data at the top of the RXFIFO.	0x0
PE	2	RO	Parity error This bit is set when the received parity does not match with the parity selected in the LCR [5:4]. It is cleared when this register is read. In the FIFO mode, this bit indicates the parity error for the received data at the top of the RXFIFO.	0x0
OE	1	RO	Overrun error This bit indicates that data in the Receiver Buffer Register (RBR) is overrun.	0x0
DR	0	RO	Data ready This bit is set when there are incoming received data in the Receiver Buffer Register (RBR). It is cleared when all of the received data are read.	0x0

Modem Status 寄存器（0x38）

Modem Status 寄存器定义如表 7-19 所示。

表 7-19 Modem Status Register

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
DCD	7	RO	Data carrier detect 0: The modem_dcdn input signal is HIGH. 1: The modem_dcdn input signal is LOW.	0x0
RI	6	RO	Ring indicator 0: The modem_rin input signal is HIGH. 1: The modem_rin input signal is LOW.	0x0
DSR	5	RO	Data set ready 0: The modem_dsrn input signal is HIGH. 1: The modem_dsrn input signal is LOW.	0x0
CTS	4	RO	Clear to send 0: The modem_ctsn signal is HIGH. 1: The modem_ctsn signal is LOW.	0x0
DDCD	3	RC	Delta data carrier detect This bit is set when the state of the modem_dcdn input signal has been changed since the last time this register is read. Otherwise, it is zero.	0x0
TERI	2	RC	Trailing edge ring indicator This bit is set when the state of the modem_rin input signal has been changed from LOW to HIGH since the last time this register is read.	0x0
DDSR	1	RC	Delta data set ready This bit is set when the state of the modem_dsrn input signal has been changed since the last time this register is read.	0x0
DCTS	0	RC	Delta clear to send This bit is set when the state of the modem_ctsn input signal has been changed since the last time this register is read.	0x0

Scratch 寄存器（0x3C）

Scratch 寄存器定义如表 7-20 所示。

表 7-20 Scratch Register

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
SCR	7:0	R/W	An one-byte storage register with no UART related function; available to software with no usage restrictions.	0x0

7.3 驱动函数定义

7.3.1 驱动函数定义

UART 驱动函数定义如表 7-21 所示。UART 驱动函数定义位于 bsp\driver\ae350\uart_ae350.c、uart_ae350.h 和 bsp\driver\include\Driver_UART.h。

表 7-21 驱动函数定义

驱动函数	描述
GetVersion	获取 UART 驱动的版本信息
GetCapabilities	获取 UART 驱动的功能信息
Initialize	初始化 UART 接口
Uninitialize	卸载 UART 接口
PowerControl	指定 UART 接口的功耗模式
Send	发送数据到 UART 发送器
Receive	从 UART 接收器接收数据
Transfer	通过 UART 接口传输数据
GetTxCount	获取 UART 接口发送数据的数量
GetRxCount	获取 UART 接口接收数据的数量
Control	配置 UART 接口的设置，执行指定的操作
GetStatus	获取 UART 接口的状态
SetModemControl	控制 UART 接口的调制解调器控制线的状态
GetModemStatus	获取 UART 接口的调制解调器控制线的状态

7.3.2 驱动函数描述

以下各节详细描述 UART 的驱动函数定义。

GetVersion

GetVersion 函数定义如表 7-22 所示。

表 7-22 GetVersion 函数定义

原型	AE350_DRIVER_VERSION (*GetVersion)(void)
描述	获取 UART 驱动的版本信息
参数	无
返回值	UART 驱动实现的版本信息

GetCapabilities

GetCapabilities 函数定义表 7-23 所示。

表 7-23 GetCapabilities 函数定义

原型	AE350_UART_CAPABILITIES (*GetCapabilities) (void)
描述	获取 UART 驱动的功能信息
参数	无
返回值	UART 驱动的功能信息

Initialize

Initialize 函数定义如表 7-24 所示。

表 7-24 Initialize 函数定义

原型	int32_t (*Initialize)(AE350_UART_SignalEvent cb_event)
描述	初始化 UART 接口
参数	cb_event：指向 AE350_UART_SignalEvent 回调函数的指针
返回值	如果发生执行错误，返回一个负值

Uninitialize

Uninitialize 函数定义如表 7-25 所示。

表 7-25 Uninitialize 函数定义

原型	int32_t (*Uninitialize)(void)
描述	卸载 UART 接口
参数	无
返回值	如果发生执行错误，返回一个负值

PowerControl

PowerControl 函数定义如表 7-26 所示。

表 7-26 PowerControl 函数定义

原型	int32_t (*PowerControl)(AE350_POWER_STATE state)
描述	指定 UART 接口的功耗模式
参数	state：UART 接口功耗模式，包括： AE350_POWER_FULL: Set up peripherals for data transfers, enable interrupts and DMA AE350_POWER_LOW: Enable power-saving AE350_POWER_OFF: Terminate pending data transfers and disable peripherals, related interrupts and DMA
返回值	如果发生执行错误，返回一个负值

Send

Send 函数定义如表 7-27 所示。

表 7-27 Send 函数定义

原型	int32_t (*Send)(const void *data, uint32_t num)
描述	发送数据到 UART 发送器
参数	data：指向发送数据缓存区的指针 num：发送数据的长度
返回值	如果发生执行错误，返回一个负值

Receive

Receive 函数定义如表 7-28 所示。

表 7-28 Receive 函数定义

原型	int32_t (*Receive)(void *data, uint32_t num)
描述	从 UART 接收器接收数据
参数	data：指向接收数据缓存区的指针 num：接收数据的长度
返回值	如果发生执行错误，返回一个负值

Transfer

Transfer 函数定义如表 7-29 所示。

表 7-29 Transfer 函数定义

原型	int32_t (*Transfer)(const void *data_out, void * data_in, uint32_t num)
描述	通过 UART 接口传输数据
参数	data_out：指向发送数据缓存区的指针 data_in：指向接收数据缓存区的指针 num：传输数据的长度
返回值	如果发生执行错误，返回一个负值

GetTxCount

GetTxCount 函数定义如表 7-30 所示。

表 7-30 GetTxCount 函数定义

原型	uint32_t (*GetTxCount)(void)
描述	获取 UART 接口发送数据的数量
参数	无
返回值	UART 最后一次执行发送传输时，发送数据的数量

GetRxCount

GetRxCount 函数定义如表 7-31 所示。

表 7-31 GetRxCount 函数定义

原型	uint32_t (*GetRxCount)(void)
描述	获取 UART 接口接收数据的数量
参数	无
返回值	UART 最后一次执行接收传输时，接收数据的数量

Control

Control 函数定义如表 7-32 所示。

表 7-32 Control 函数定义

原型	int32_t (*Control)(uint32_t control, uint32_t arg)
描述	配置 UART 接口的设置，执行指定的操作
参数	control：UART 驱动接口的一种设置或执行的一种操作 arg：指定设置或操作的附加信息
返回值	如果发生执行错误，返回一个负值

“control”和“arg”设置与操作如表 7-33 所示。

表 7-33 Control Settings and Operations

Options for control	arg Specifies	Settings or Operations
Operation modes (Bits: 0~7)		
AE350_UART_MODE_ASYNCHRONOUS	baudrate	Sets to be asynchronous UART mode
AE350_UART_MODE_SYNCHRONOUS_MASTER	baudrate	Sets to the synchronous master mode with clock signal generation
AE350_UART_MODE_SYNCHRONOUS_SLAVE	-	Sets to the synchronous slave mode with external clock signal
AE350_UART_MODE_SINGLE_WIRE	baudrate	Sets to the single-wire (half-duplex) mode
AE350_UART_MODE_IRDA	baudrate	Sets to the infra-red data mode
AE350_UART_MODE_SMART_CARD	baudrate	Sets to the Smart Card mode
Data bit (Bits: 8~11)		
AE350_UART_DATA_BITS_5	-	Sets to 5 data bits
AE350_UART_DATA_BITS_6	-	Sets to 6 data bits
AE350_UART_DATA_BITS_7	-	Sets to 7 data bits
AE350_UART_DATA_BITS_8	-	Sets to 8 data bits
AE350_UART_DATA_BITS_9	-	Sets to 9 data bits
Parity bit (Bits: 12~13)		
AE350_UART_PARITY_NONE	-	Sets to no parity (default)
AE350_UART_PARITY_EVEN	-	Sets to even parity
AE350_UART_PARITY_ODD	-	Sets to odd parity
Stop bit (Bits: 14~15)		
AE350_UART_STOP_BITS_1	-	Sets to 1 stop bit (default)
AE350_UART_STOP_BITS_2	-	Sets to 2 stop bits

Options for control	arg Specifies	Settings or Operations
AE350_UART_STOP_BITS_1_5	-	Sets to 1.5 stop bits
AE350_UART_STOP_BITS_0_5	-	Sets to 0.5 stop bits
Flow control (Bits: 16~17)		
AE350_UART_FLOW_CONTROL_NONE	-	Sets to have no flow control signal (default)
AE350_UART_FLOW_CONTROL_RTS	-	Sets to use the RTS flow control signal
AE350_UART_FLOW_CONTROL_CTS	-	Sets to use the CTS flow control signal
AE350_UART_FLOW_CONTROL_RTS_CTS	-	Sets to use the RTS and CTS flow control signal
Clock parity (Bit: 18)		
AE350_UART_CPOL0	-	CPOL=0 (default): Data are captured on the rising edge
AE350_UART_CPOL1	-	CPOL=1: Data are captured on the falling edge
Clock phase (Bit: 19)		
AE350_UART_CPHA0	-	CPHA=0 (default): Data are sampled on the first edge
AE350_UART_CPHA1	-	CPHA=1: Data are sampled on the second edge
Other operations (Bits: 0~19)		
AE350_UART_SET_DEFAULT_TX_VALUE	transmit value	Sets the default transmit value
AE350_UART_SET_IRDA_PULSE	0=3/16 of bit period	Sets the IrDA pulse value in ns
AE350_UART_SET_SMART_CARD_GUARD_TIME	number of bit periods	Sets the Smart Card guard time
AE350_UART_SET_SMART_CARD_CLOCK	0=Clock not set	Sets the Smart Card clock in Hz
AE350_UART_CONTROL_SMART	0=disable;	Enables or disables

Options for control	arg Specifies	Settings or Operations
_CARD_NACK	1=enable	the Smart Card NACK generation
AE350_UART_CONTROL_TX	0=disable; 1=enable	Enables or disables the transmitter
AE350_UART_CONTROL_RX	0=disable; 1=enable	Enables or disables the receiver
AE350_UART_CONTROL_BREAK	0=disable; 1=enable	Enables or disables continuous break transmission
AE350_UART_ABORT_SEND	-	Aborts the send operations
AE350_UART_ABORT_RECEIVE	-	Aborts the receive operation
AE350_UART_ABORT_TRANSFER	-	Aborts the transfer operation

GetStatus

GetStatus 函数定义如表 7-34 所示。

表 7-34 GetStatus 函数定义

原型	AE350_UART_STATUS (*GetStatus)(void)
描述	获取 UART 接口的状态
参数	无
返回值	UART 接口的当前状态

SetModemControl

SetModemControl 函数定义如表 7-35 所示。

表 7-35 SetModemControl 函数定义

原型	int32_t (*SetModemControl)(AE350_UART_MODEM_CONTROL control)
描述	控制 UART 接口的调制解调器控制线的状态
参数	control：启动或释放调制解调器的控制线，可用选项包括： AE350_UART_RTS_CLEAR: Deactivate RTS AE350_UART_RTS_SET: Activate RTS AE350_UART_DTR_CLEAR: Deactivate DTR

	AE350_UART_DTR_SET: Activate DTR
返回值	如果发生执行错误，返回一个负值

GetModemStatus

GetModemStatus 函数定义如表 7-36 所示。

表 7-36 GetModemStatus 函数定义

原型	AE350_UART_MODEM_STATUS (*GetModemStatus)(void)
描述	获取 UART 接口的调制解调器控制线的状态
参数	无
返回值	UART 接口的调制解调器控制线的当前状态

8 GPIO

8.1 简介

Gowin RiscV_AE350_SOC 包含一个 GPIO 控制器，支持多达 32 个可独立编程的输入输出控制通道。

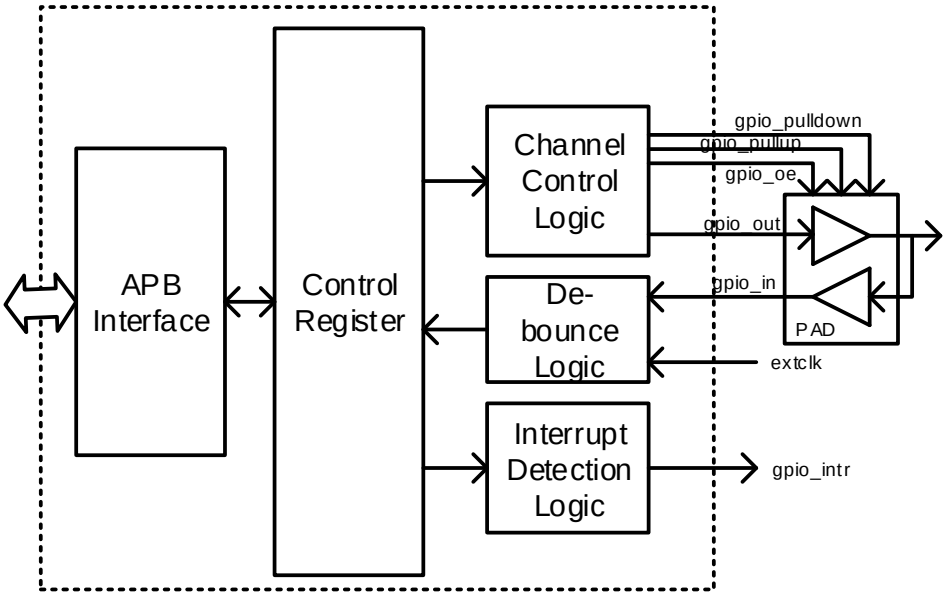
8.1.1 特征

- 支持 AMBA 2.0 APB 总线协议规范
- 多达 32 个独立 I/O 通道
- 每个通道可编程上拉和下拉
- 每个输入通道可编程防抖动功能
- 每个输入通道可配置作为中断输入源
  - 中断源可配置为电平触发或边沿触发

8.1.2 结构框图

GPIO 结构框图如图 8-1 所示。

图 8-1 GPIO 结构框图



8.1.3 功能描述

GPIO 控制器提供多达 32 个独立 I/O 通道，每个通道可以独立配置为输入端口或输出端口。如果作为输入端口，控制器可以对输入信号进行采样并产生中断。如果控制器作为输出端口，可以通过上拉或下拉来驱动输出信号。

GPIO 控制器还支持防抖动功能，可以过滤掉输入通道的毛刺，该功能可以配置为每个通道单独启用，防抖动持续时间也是可配置的。

8.2 寄存器定义

8.2.1 寄存器定义

GPIO 寄存器定义如表 8-1 所示。GPIO 寄存器定义位于 bsp\ae350\ae350.h。

表 8-1 寄存器定义

地址偏移	寄存器名称	描述
0x00	IDREV	ID and Revision Register
0x04~0x0C	-	Reserved
0x10	CFG	Configuration Register
0x14~0x1C	-	Reserved
0x20	DATAIN	Channel Data-In Register

地址偏移	寄存器名称	描述
0x24	DATAOUT	Channel Data-Out Register
0x28	CHANNELDIR	Channel Direction Register
0x2C	DOUTCLEAR	Channel Data-Out Clear Register
0x30	DOUTSET	Channel Data-Out Set Register
0x34~0x3C	-	Reserved
0x40	PULLEN	Pull Enable Register
0x44	PULLTYPE	Pull Type Register
0x48~0x4C	-	Reserved
0x50	INTREN	Interrupt Enable Register
0x54	INTRMODE0	Interrupt Mode Register (0~7)
0x58	INTRMODE1	Interrupt Mode Register (8~15)
0x5C	INTRMODE2	Interrupt Mode Register (16~23)
0x60	INTRMODE3	Interrupt Mode Register (24~31)
0x64	INTRSTATUS	Interrupt Status Register
0x68~0x6C	-	Reserved
0x70	DEBOUNCEEN	De-bounce Enable Register
0x74	DEBOUNCECTRL	De-bounce Control Register
0x78~0x7C	-	Reserved

8.2.2 寄存器描述

以下各节详细描述 GPIO 寄存器定义。

寄存器类型缩略语概括如下：

- RO: Read-only
- WO: Write-only (read as zero)
- R/W: Readable and writable
- W1C: Write 1 to clear

ID 和 Revision 寄存器（0x00）

ID 和 Revision 寄存器用于保存 ID 和 Revision 编号，初始值依赖于所用版本。ID 和 Revision 寄存器定义如表 8-2 所示。

表 8-2 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:8	RO	ID number for GPIO	0x020310
RevMajor	7:4	RO	Major revision number	Revision dependent
RevMinor	3:0	RO	Minor revision number	Revision dependent

Configuration 寄存器（0x10）

Configuration 寄存器定义如表 8-3 所示。

表 8-3 Configuration Register

Name	Bit	Type	Description	Reset
Pull	31	RO	Pull option 0: Pull option is not configured 1: Pull option is configured	Configuration dependent
Intr	30	RO	Interrupt option 0: interrupt option is not configured 1: interrupt option is configured	Configuration dependent
Debounce	29	RO	De-bounce option 0: de-bounce option is not configured 1: de-bounce option is configured	Configuration dependent
-	28:6	-	Reserved	-
ChannelNum	5:0	RO	Number of channels	Configuration dependent

Channel Data-In 寄存器（0x20）

Channel Data-In 寄存器定义如表 8-4 所示。

表 8-4 Channel Data-In Register

Name	Bit	Type	Description	Reset
DataIn	N:0	RO	Channel data input register	0x0

Channel Data-Out 寄存器（0x24）

Channel Data-Out 寄存器定义如表 8-5 所示。

表 8-5 Channel Data-Out Register

Name	Bit	Type	Description	Reset
DataOut	N:0	RO	GPIO data output	0x0

Channel Direction 寄存器（0x28）

Channel Direction 寄存器定义如表 8-6 所示。

表 8-6 Channel Direction Register

Name	Bit	Type	Description	Reset
ChannelDir	N:0	R/W	GPIO channel direction 0: input 1: output	0x0

Channel Data-Out Clear 寄存器（0x2C）

Channel Data-Out Clear 寄存器定义如表 8-7 所示。

表 8-7 Channel Data-Out Clear Register

Name	Bit	Type	Description	Reset
DoutClear	N:0	WO	GPIO data-out clear; write 1 to clear the corresponding output channels.	0x0

Channel Data-Out Set 寄存器（0x30）

Channel Data-Out Set 寄存器定义如表 8-8 所示。

表 8-8 Channel Data-Out Set Register

Name	Bit	Type	Description	Reset
DoutSet	N:0	WO	GPIO data-out set; write 1 to set the corresponding output channels.	0x0

Pull Enable 寄存器（0x40）

Pull Enable 寄存器定义如表 8-9 所示。

表 8-9 Pull Enable Register

Name	Bit	Type	Description	Reset
PullEn	N:0	R/W	GPIO pull enable; write 1 to enable pull-up/pull-down of the corresponding channels.	0x0

Pull Type 寄存器（0x44）

Pull Type 寄存器定义如表 8-10 所示。

表 8-10 Pull Type Register

Name	Bit	Type	Description	Reset
PullType	N:0	R/W	GPIO pull control 0: pull-up 1: pull-down	0x0

Interrupt Enable 寄存器（0x50）

Interrupt Enable 寄存器定义如表 8-11 所示。

表 8-11 Interrupt Enable Register

Name	Bit	Type	Description	Reset
IntEn	N:0	R/W	GPIO interrupt enable; write 1 to enable interrupts of the corresponding channels.	0x0

Interrupt Mode 寄存器（0x54, 0x58, 0x5C, 0x60）

表 8-12、表 8-13、表 8-14、表 8-15 描述了对应 GPIO 通道的中断触发方式。有关各通道的模式编码，请参考“Ch0IntrM”字段的描述，如果通道不存在则保留该字段。

表 8-12 Channel (0~7) Interrupt Mode Register

Name	Bit	Type	Description	Reset
-	31	-	Reserved	-
Ch7IntrM	30:28	R/W	Channel 7 interrupt mode	0x0
-	27	-	Reserved	-
Ch6IntrM	26:24	R/W	Channel 6 interrupt mode	0x0
-	23	-	Reserved	-
Ch5IntrM	22:20	R/W	Channel 5 interrupt mode	0x0
-	19	-	Reserved	-

Name	Bit	Type	Description	Reset
Ch4IntrM	18:16	R/W	Channel 4 interrupt mode	0x0
-	15	-	Reserved	-
Ch3IntrM	14:12	R/W	Channel 3 interrupt mode	0x0
-	11	-	Reserved	-
Ch2IntrM	10:8	R/W	Channel 2 interrupt mode	0x0
-	7	-	Reserved	-
Ch1IntrM	6:4	R/W	Channel 1 interrupt mode	0x0
-	3	-	Reserved	-
Ch0IntrM	2:0	R/W	Channel 0 interrupt mode 0: No operation 2: High-level 3: Low-level 5: Negative-edge 6: Positive-edge 7: Dual-edge 1, 4: Reserved	0x0

表 8-13 Channel (8~15) Interrupt Mode Register

Name	Bit	Type	Description	Reset
-	31	-	Reserved	-
Ch15IntrM	30:28	R/W	Channel 15 interrupt mode	0x0
-	27	-	Reserved	-
Ch14IntrM	26:24	R/W	Channel 14 interrupt mode	0x0
-	23	-	Reserved	-
Ch13IntrM	22:20	R/W	Channel 13 interrupt mode	0x0
-	19	-	Reserved	-
Ch12IntrM	18:16	R/W	Channel 12 interrupt mode	0x0
-	15	-	Reserved	-
Ch11IntrM	14:12	R/W	Channel 11 interrupt mode	0x0
-	11	-	Reserved	-
Ch10IntrM	10:8	R/W	Channel 10 interrupt mode	0x0
-	7	-	Reserved	-
Ch9IntrM	6:4	R/W	Channel 9 interrupt mode	0x0

Name	Bit	Type	Description	Reset
–	3	–	Reserved	–
Ch8IntrM	2:0	R/W	Channel 8 interrupt mode	0x0

表 8-14 Channel (16~23) Interrupt Mode Register

Name	Bit	Type	Description	Reset
–	31	–	Reserved	–
Ch23IntrM	30:28	R/W	Channel 23 interrupt mode	0x0
–	27	–	Reserved	–
Ch22IntrM	26:24	R/W	Channel 22 interrupt mode	0x0
–	23	–	Reserved	–
Ch21IntrM	22:20	R/W	Channel 21 interrupt mode	0x0
–	19	–	Reserved	–
Ch20IntrM	18:16	R/W	Channel 20 interrupt mode	0x0
–	15	–	Reserved	–
Ch19IntrM	14:12	R/W	Channel 19 interrupt mode	0x0
–	11	–	Reserved	–
Ch18IntrM	10:8	R/W	Channel 18 interrupt mode	0x0
–	7	–	Reserved	–
Ch17IntrM	6:4	R/W	Channel 17 interrupt mode	0x0
–	3	–	Reserved	–
Ch16IntrM	2:0	R/W	Channel 16 interrupt mode	0x0

表 8-15 Channel (24~31) Interrupt Mode Register

Name	Bit	Type	Description	Reset
–	31	–	Reserved	–
Ch31IntrM	30:28	R/W	Channel 31 interrupt mode	0x0
–	27	–	Reserved	–
Ch30IntrM	26:24	R/W	Channel 30 interrupt mode	0x0
–	23	–	Reserved	–
Ch29IntrM	22:20	R/W	Channel 29 interrupt mode	0x0
–	19	–	Reserved	–
Ch28IntrM	18:16	R/W	Channel 28 interrupt mode	0x0

Name	Bit	Type	Description	Reset
-	15	-	Reserved	-
Ch27IntrM	14:12	R/W	Channel 27 interrupt mode	0x0
-	11	-	Reserved	-
Ch26IntrM	10:8	R/W	Channel 26 interrupt mode	0x0
-	7	-	Reserved	-
Ch25IntrM	6:4	R/W	Channel 25 interrupt mode	0x0
-	3	-	Reserved	-
Ch24IntrM	2:0	R/W	Channel 24 interrupt mode	0x0

Channel Interrupt Status 寄存器（0x64）

Channel Interrupt Status 寄存器定义如表 8-16 所示。

表 8-16 Channel Interrupt Status Register

Name	Bit	Type	Description	Reset
IntrStatus	N:0	R/W1C	The interrupt status of the corresponding channel; write 1 to clear. 0: No interrupt 1: Interrupt	0x0

De-bounce Enable 寄存器（0x70）

De-bounce Enable 寄存器定义如表 8-17 所示。

表 8-17 De-bounce Enable Register

Name	Bit	Type	Description	Reset
DeBounceEn	N:0	R/W	Data-in de-bounce enable; write 1 to enable de-bouncing of the corresponding channels.	0x0

De-bounce Control 寄存器（0x74）

De-bounce Control 寄存器定义如表 8-18 所示。

表 8-18 De-bounce Control Register

Name	Bit	Type	Description	Reset
DBClkSel	31	R/W	GPIO de-bounce clock source selection. Select pclk (the faster clock) as the de-bounce clock source would shorten the de-	0x0

Name	Bit	Type	Description	Reset
			bounce latency. 0: extclk 1: pclk	
-	30:8	-	Reserved	-
DBPreScale	7:0	R/W	GPIO pre-scale base, to scale the de-bounce clock source before it is used as the actual de-bounce clock. The de-bounce period would be multiplied by (DBPreScale+1); e.g., setting DBPreScale to 3 would filter out pulses which are less than 4 de-bounce clock period.	0x0

8.3 驱动函数定义

8.3.1 驱动函数定义

GPIO 驱动函数定义如表 8-19 所示。GPIO 驱动函数定义位于 bsp\driver\ae350\gpio_ae350.c、gpio_ae350.h 和 bsp\driver\include\Driver_GPIO.h。

表 8-19 驱动函数定义

驱动函数	描述
GetVersion	获取 GPIO 驱动的版本信息
Initialize	初始化 GPIO 接口
Uninitialize	卸载 GPIO 接口
PinWrite	GPIO 输出引脚输出一个数据
PinRead	GPIO 输入引脚输入一个数据
Write	GPIO 输出端口输出数据
Read	GPIO 输入端口输入数据
SetDir	设置 GPIO 端口输入输出方向
Control	配置 GPIO 接口的设置，执行指定的操作
GetStatus	获取 GPIO 接口的状态，包括上拉或下拉选项、中断选项、防抖动选项和通道号

8.3.2 驱动函数描述

以下各节详细描述 GPIO 的驱动函数定义。

GetVersion

GetVersion 函数定义如表 8-20 所示。

表 8-20 GetVersion 函数定义

原型	AE350_DRIVER_VERSION (*GetVersion)(void)
描述	获取 GPIO 驱动的版本信息
参数	无
返回值	GPIO 驱动实现的版本信息

Initialize

Initialize 函数定义如表 8-21 所示。

表 8-21 Initialize 函数定义

原型	int32_t (*Initialize)(AE350_GPIO_SignalEvent cb_event)
描述	初始化 GPIO 接口
参数	cb_event：指向 AE350_GPIO_SignalEvent 回调函数的指针
返回值	如果发生执行错误，返回一个负值

Uninitialize

Uninitialize 函数定义如表 8-22 所示。

表 8-22 Uninitialize 函数定义

原型	int32_t (*Uninitialize)(void)
描述	卸载 GPIO 接口
参数	无
返回值	如果发生执行错误，返回一个负值

PinWrite

PinWrite 函数定义如表 8-23 所示。

表 8-23 PinWrite 函数定义

原型	void (*PinWrite)(uint32_t pin_num, int32_t val)
描述	GPIO 输出引脚输出一个数据

参数	pin_num：被使用的 GPIO 引脚 val：该引脚输出的高电平或低电平
返回值	无

PinRead

PinRead 函数定义如表 8-24 所示。

表 8-24 PinRead 函数定义

原型	uint8_t (*PinRead)(uint32_t pin_num)
描述	GPIO 输入引脚输入一个数据
参数	pin_num：被使用的 GPIO 引脚
返回值	GPIO 输入引脚输入的高电平或低电平

Write

Write 函数定义如表 8-25 所示。

表 8-25 Write 函数定义

原型	void (*Write)(uint32_t mask, uint32_t val)
描述	GPIO 输出端口输出数据
参数	mask：被使用的 GPIO 端口 val：该端口输出的高电平或低电平数据
返回值	无

Read

Read 函数定义如表 8-26 所示。

表 8-26 Read 函数定义

原型	uint32_t (*Read)(void)
描述	GPIO 输入端口输入数据
参数	无
返回值	GPIO 输入端口输入的高电平或低电平数据

SetDir

SetDir 函数定义如表 8-27 所示。

表 8-27 SetDir 函数定义

原型	void (*SetDir)(uint32_t mask, int32_t dir)
描述	设置 GPIO 端口输入输出方向
参数	mask：被使用的 GPIO 端口 dir：设置该 GPIO 端口方向，输入或输出
返回值	无

Control

Control 函数定义如表 8-28 所示。

表 8-28 Control 函数定义

原型	int32_t (*Control)(uint32_t mode, uint32_t mask, uint32_t clksel, uint32_t scale)
描述	配置 GPIO 接口的设置，执行指定的操作
参数	mode：GPIO 驱动接口的一种设置或执行的一种操作 mask：被使用的 GPIO 端口 clksel：选择防抖动时钟源，外部时钟或 PCLK scale：防抖动的时钟源分频系数
返回值	如果发生执行错误，返回一个负值

表 8-29 描述了“mode”选项，“mask”详细信息，以及这些选项的设置与操作。

表 8-29 Mode Settings or Operations

Options for Mode	Mask Specifies	Settings or Operations
AE350_GPIO_INTR_ENABLE	GPIO ports	Enables the interrupt of GPIO ports
AE350_GPIO_INTR_DISABLE		Disables the interrupt of GPIO ports
AE350_GPIO_SET_INTR_LOW_LEVEL		Sets the interrupt type of GPIO ports

Options for Mode	Mask Specifies	Settings or Operations
		as low level trigger
AE350_GPIO_SET_INTR_HIGH_LEVEL		Sets the interrupt type of GPIO ports as high level trigger
AE350_GPIO_SET_INTR_NEGATIVE_EDGE		Sets the interrupt type of GPIO ports as negative edge trigger
AE350_GPIO_SET_INTR_POSITIVE_EDGE		Sets the interrupt type of GPIO ports as positive edge trigger
AE350_GPIO_SET_INTR_DUAL_EDGE		Sets the interrupt type of GPIO ports as dual edge trigger
AE350_GPIO_INTR_CLEAR		Clears the interrupt of GPIO ports
AE350_GPIO_PULL_ENABLE		Enables the pull mode of GPIO ports
AE350_GPIO_PULL_DISABLE		Disables the pull mode of GPIO ports
AE350_GPIO_SET_PULL_UP		Sets the pull mode of GPIO ports as up
AE350_GPIO_SET_PULL_DOWN		Sets the pull mode of GPIO ports

Options for Mode	Mask Specifies	Settings or Operations
		as down
AE350_GPIO_DB_ENABLE		Enables the de-bounce of GPIO ports
AE350_GPIO_DB_DISABLE		Disables the de-bounce of GPIO ports
AE350_GPIO_SET_DB_CLKSRC		Sets the de-bounce clock source of GPIO ports
AE350_GPIO_SET_DB_PRESCALE		Sets the de-bounce pre-scale of GPIO ports

GetStatus

GetStatus 函数定义如表 8-30 所示。

表 8-30 GetStatus 函数定义

原型	int32_t (*GetStatus)(AE350_CFG_STATUS* cfg_info)
描述	获取 GPIO 接口的状态，包括上拉或下拉选项、中断选项、防抖动选项和通道号
参数	cfg_info：GPIO 接口的状态信息
返回值	如果发生执行错误，返回一个负值

9 I2C

9.1 简介

Gowin RiscV_AE350_SOC 包含一个 I2C 控制器，支持主机模式和从机模式。

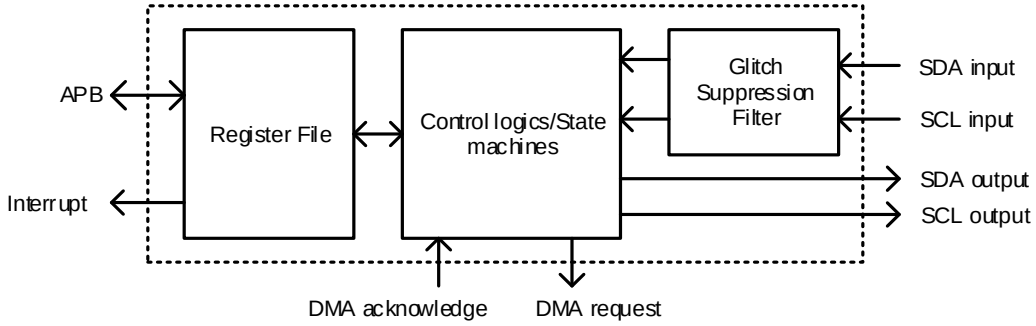
9.1.1 特征

- 支持 AMBA 2.0 APB 总线协议规范
- 支持标准模式（100 Kb/s）、快速模式（400 Kb/s）和超快速模式（1 Mb/s）
- 可编程的主机模式和从机模式
- 支持 7 位和 10 位寻址模式
- 支持通用广播地址
- 自动时钟拉伸
- 可编程的时钟和数据时序
- 支持 DMA 传输

9.1.2 结构框图

I2C 结构框图如图 9-1 所示。

图 9-1 I2C 结构框图



9.1.3 功能描述

I2C 控制器通过设置控制寄存器，可以配置为 I2C 主机模式或 I2C 从机模式。如果作为 I2C 主机模式，控制器提供一种高效的 I2C 事务发起方式；如果作为 I2C 从机模式，当 I2C 事务的地址字节与地址寄存器匹配时，控制器被寻址。

I2C 提供一个时序参数乘法器寄存器（TPM），当 I2C 运行在相对较快的 APB 时钟频率时，使 I2C 能够满足 I2C 总线的时序要求。Setup 寄存器中所有时序参数的实际值，都需要乘以（TPM+1）。

9.2 寄存器定义

9.2.1 寄存器定义

I2C 寄存器定义如表 9-1 所示。I2C 寄存器定义位于 bsp\ae350\ae350.h。

表 9-1 寄存器定义

地址偏移	寄存器名称	描述
0x00	IDREV	ID and Revision Register
0x04~0x0C	-	Reserved
0x10	CFG	Configuration Register
0x14	INTEN	Interrupt Enable Register
0x18	STATUS	Status Register
0x1C	ADDR	Address Register
0x20	DATA	Data Register
0x24	CTRL	Control Register
0x28	CMD	Command Register
0x2C	SETUP	Setup Register

地址偏移	寄存器名称	描述
0x30	TPM	I2C Timing Parameter Multiplier Register

9.2.2 寄存器描述

以下各节详细描述 I2C 寄存器定义。当 I2C 控制器作为主机模式或从机模式时，寄存器定义有所不同，下面将分别描述。

寄存器类型缩略语概括如下：

- RO: Read-only
- WO: Write-only (read as zero)
- R/W: Readable and writable
- R/W1C: Readable and Write 1 to clear

ID 和 Revision 寄存器（0x00）

ID 和 Revision 寄存器用于保存 ID 和 Revision 编号，初始值依赖于所用版本。ID 和 Revision 寄存器定义如表 9-2 所示。

表 9-2 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:8	RO	ID number for I2C	0x020210
RevMajor	7:4	RO	Major revision number	Revision Dependent
RevMinor	3:0	RO	Minor revision number	Revision Dependent

Configuration 寄存器（0x10）

Configuration 寄存器用于保存数据 FIFO 的大小。Configuration 寄存器定义如表 9-3 所示。

表 9-3 Configuration Register

Name	Bit	Type	Description	Reset
-	31:2	-	Reserved	-
FIFOSize	1:0	RO	FIFO size: 0: 2 bytes 1: 4 bytes 2: 8 bytes 3: 16 bytes	Configuration Dependent

Interrupt Enable 寄存器（0x14）

Interrupt Enable 寄存器用于开启或关闭中断。Interrupt Enable 寄存器定义如表 9-4 所示。

表 9-4 Interrupt Enable Register

Name	Bit	Type	Description	Reset
-	31:10	-	Reserved	-
Cmpl	9	R/W	Set to enable the Completion Interrupt. Master: interrupts when a transaction is issued from this master and completed without losing the bus arbitration. Slave: interrupts when a transaction addressing the controller is completed.	0x0
ByteRecv	8	R/W	Set to enable the Byte Receive Interrupt. Interrupts when a byte of data is received. Auto-ACK will be disabled if this interrupt is enabled, that is, the software needs to ACK/NACK the received byte manually.	0x0
ByteTrans	7	R/W	Set to enable the Byte Transmit Interrupt. Interrupts when a byte of data is transmitted.	0x0
Start	6	R/W	Set to enable the START Condition Interrupt. Interrupts when a START condition/repeated START condition is detected.	0x0
Stop	5	R/W	Set to enable the STOP Condition Interrupt. Interrupts when a STOP condition is detected.	0x0
ArbLose	4	R/W	Set to enable the Arbitration Lose Interrupt. Master: interrupts when the controller loses the bus arbitration. Slave: not available in this mode.	0x0
AddrHit	3	R/W	Set to enable the Address Hit Interrupt. Master: interrupts when the addressed slave returned an ACK. Slave: interrupts when the controller is addressed.	0x0
FIFOHalf	2	R/W	Set to enable the FIFO Half Interrupt. Receiver: Interrupts when the FIFO is half-full, i.e. there is ≥ 1/2 entries in the FIFO.	0x0

Name	Bit	Type	Description	Reset
			Transmitter: Interrupts when the FIFO is half-empty, i.e. there is ≤ 1/2 entries in the FIFO. This interrupt depends on the transaction direction; don't enable this interrupt unless the transfer direction is determined, otherwise unintended interrupts may be triggered.	
FIFOFull	1	R/W	Set to enable the FIFO Full Interrupt. Interrupts when the FIFO is full.	0x0
FIFOEmpty	0	R/W	Set to enable the FIFO Empty Interrupt. Interrupts when FIFO is empty.	0x0

Status 寄存器（0x18）

Status 寄存器用于保存中断状态和 I2C 总线状态。Status 寄存器定义如表 9-5 所示。

表 9-5 Status Register

Name	Bit	Type	Description	Reset
-	31:15	-	Reserved	-
LineSDA	14	RO	Indicates the current status of the SDA line on the bus. 1: High 0: Low	SDA line status
LineSCL	13	RO	Indicates the current status of the SCL line on the bus. 1: High 0: Low	SCL line status
GenCall	12	RO	Indicates that the address of the current transaction is a general call address. This status is only valid in slave mode. 1: General call 0: Not general call	0x0
BusBusy	11	RO	Indicates that the bus is busy. The bus is busy when a START condition is on bus and it ends when a STOP condition is seen on bus. 1: Busy	0x0

Name	Bit	Type	Description	Reset
			0: Not busy	
ACK	10	RO	Indicates the type of the last received/transmitted acknowledgement bit. 1: ACK 0: NACK	0x0
Cmpl	9	R/W1C	Transaction Completion Master: Indicates that a transaction has been issued from this master and completed without losing the bus arbitration. Slave: Indicates that a transaction addressing the controller has been completed. This status bit must be cleared to receive the next transaction; otherwise, the next incoming transaction will be locked.	0x0
ByteRecv	8	R/W1C	Indicates that a byte of data has been received.	0x0
ByteTrans	7	R/W1C	Indicates that a byte of data has been transmitted.	0x0
Start	6	R/W1C	Indicates that a START Condition or a repeated START condition has been transmitted/received.	0x0
Stop	5	R/W1C	Indicates that a STOP Condition has been transmitted/received.	0x0
ArbLose	4	R/W1C	Indicates that the controller has lost the bus arbitration (master mode only).	0x0
AddrHit	3	R/W1C	Master: indicates that a slave has responded to the transaction. Slave: indicates that a transaction is targeting the controller (including the General Call).	0x0
FIFOHalf	2	RO	Transmitter: Indicates that the FIFO is half-empty. Receiver: Indicates that the FIFO is half-full.	0x0
FIFOFull	1	RO	Indicates that the FIFO is full.	0x0

Name	Bit	Type	Description	Reset
FIFOEmpty	0	RO	Indicates that the FIFO is empty.	0x1

Address 寄存器（0x1C）

Address 寄存器用于保存从机地址。当 I2C 配置作为主机模式时，这是下一个事务的目标从机地址；当 I2C 配置作为从机模式时，这是总线上控制器的地址。

Address 寄存器定义如表 9-6 所示。

表 9-6 Address Register

Name	Bit	Type	Description	Reset
-	31:10	-	Reserved	-
Addr	9:0	R/W	The slave address. For 7-bit addressing mode, the most significant 3 bits are ignored and only the least-significant 7 bits of Addr are valid.	0x0

Data 寄存器（0x20）

Data 寄存器用于 FIFO 的数据访问端口。Data 寄存器定义如表 9-7 所示。

表 9-7 Data Register

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	-
Data	7:0	R/W	Write this register to put one byte of data to the FIFO. Read this register to get one byte of data from the FIFO.	0x0

Control 寄存器（0x24）

Control 寄存器用于控制事务的阶段选择，以及记录数据阶段的进程。Control 寄存器定义如表 9-8 所示。

表 9-8 Control Register

Name	Bit	Type	Description	Reset
-	31:13	-	Reserved	-

Name	Bit	Type	Description	Reset
Phase_start	12	R/W	Enable this bit to send a START condition at the beginning of transaction. Master mode only.	0x1
Phase_addr	11	R/W	Enable this bit to send the address after START condition. Master mode only.	0x1
Phase_data	10	R/W	Enable this bit to send the data after Address phase. Master mode only.	0x1
Phase_stop	9	R/W	Enable this bit to send a STOP condition at the end of a transaction. Master mode only.	0x1
Dir	8	R/W	Transaction direction Master: Set this bit to determine the direction for the next transaction. 0: Transmitter 1: Receiver Slave: The direction of the last received transaction. 0: Receiver 1: Transmitter	0x0
DataCnt	7:0	R/W	Data counts in bytes. Master: The number of bytes to transmit/receive. 0 means 256 bytes. DataCnt will be decreased by one for each byte transmitted/received. Slave: the meaning of DataCnt depends on the DMA mode: If DMA is not enabled, DataCnt is the number of bytes transmitted/received from the bus master. It is reset to 0 when the controller is addressed and then increased by one for each byte of data transmitted/received. If DMA is enabled, DataCnt is the number of bytes to transmit/receive. It will not be reset to 0 when the slave is addressed and it will be decreased by one for each byte of data	0x0

Name	Bit	Type	Description	Reset
			transmitted/received.	

Command 寄存器（0x28）

Command 寄存器定义如表 9-9 所示。

表 9-9 Command Register

Name	Bit	Type	Description	Reset
-	31:3	-	Reserved	-
CMD	2:0	R/W	Write this register with the following values to perform the corresponding actions: 0: no action 1: issue a data transaction (Master only) 2: respond with an ACK to the received byte 3: respond with a NACK to the received byte 4: clear the FIFO 5: reset the I2C controller (abort current transaction, set the SDA and SCL line to the open-drain mode, reset the Status Register and the Interrupt Enable Register, and empty the FIFO) When issuing a data transaction by writing 0x1 to this register, the CMD field stays at 0x1 for the duration of the entire transaction, and it is only cleared to 0x0 after when the transaction has completed or when the controller loses the arbitration. Note: No transaction will be issued by the controller when all phases (Start, Address, Data and Stop) are disabled.	0x0

Setup 寄存器（0x2C）

Setup 寄存器用于保存可编程配置和 I2C 总线的时序参数。更多时序设置细节，请参考 9.3 时序设置方法。

Setup 寄存器定义如表 9-10 所示。

表 9-10 Setup Register

Name	Bit	Type	Description	Reset
-	31:29	-	Reserved	-

Name	Bit	Type	Description	Reset
T_SUDAT	28:24	R/W	T_SUDAT defines the data setup time before releasing the SCL. Setup time = (2 * t_{pclk}) + (2 + T_SP + T_SUDAT) * t_{pclk} * (TPM + 1) t_{pclk} = PCLK period TPM = The multiplier value in Timing Parameter Multiplier Register	0x5
T_SP	23:21	R/W	T_SP defines the pulse width of spikes that must be suppressed by the input filter. Pulse width = T_SP * t_{pclk} * (TPM + 1)	0x1
T_HDDAT	20:16	R/W	T_HDDAT defines the data hold time after SCL goes LOW Hold time = (2 * t_{pclk}) + (2 + T_SP + T_HDDAT) * t_{pclk} * (TPM + 1)	0x5
-	15:14	-	Reserved	-
T_SCLRatio	13	R/W	The LOW period of the generated SCL clock is defined by the combination of T_SCLRatio and T_SCLHi values. When T_SCLRatio = 0, the LOW period is equal to HIGH period. When T_SCLRatio = 1, the LOW period is roughly two times of HIGH period. SCL LOW period = (2 * t_{pclk}) + (2 + T_SP + T_SCLHi * ratio) * t_{pclk} * (TPM + 1) 1: ratio = 2 0: ratio = 1 This field is only valid when the controller is in the master mode.	0x1
T_SCLHi	12:4	R/W	The HIGH period of generated SCL clock is defined by T_SCLHi. SCL HIGH period = (2 * t_{pclk}) + (2 + T_SP + T_SCLHi) * t_{pclk} * (TPM + 1) The T_SCLHi value must be greater than T_SP and T_HDDAT values. This field is only valid when the controller is in the master mode.	0x10
DMAEn	3	R/W	Enable the direct memory access mode data transfer.	0x0

Name	Bit	Type	Description	Reset
			1: Enable 0: Disable	
Master	2	R/W	Configure this device as a master or a slave 1: Master mode 0: Slave mode	0x0
Addressing	1	R/W	I2C addressing mode: 1: 10-bit addressing mode 0: 7-bit addressing mode	0x0
IICEn	0	R/W	Enable the I2C controller 1: Enable 0: Disable	0x0

Timing Parameter Multiplier 寄存器（0x30）

Timing Parameter Multiplier 寄存器用于保存一个乘法器数值，放大定义于 Setup 寄存器中的 I2C 总线时序参数。当控制器运行于较高的 APB 时钟频率时，该乘法器数值有助于控制器满足 I2C 总线接口的时序要求。该数值的计算方法，请参考 9.3.5 时序参数乘法器。

Timing Parameter Multiplier 寄存器定义如表 9-11 所示。

表 9-11 Timing Parameter Multiplier Register

Name	Bit	Type	Description	Reset
-	31:5	-	Reserved	-
TPM	4:0	R/W	A multiplication value for I2C timing parameters. All the timing parameters in the Setup Register are multiplied by (TPM + 1).	0x0

9.3 时序设置方法

开启 I2C 控制器之前需要：

- 通过编程 Timing Parameter Multiplier 寄存器，建立时序参数乘法器
- 通过编程 Setup 寄存器，建立 I2C 总线时序参数

以下各节描述了如何设置 Setup 寄存器，以满足 I2C 总线的时序参数。假设 APB 时钟为 40MHz，则 APB 时钟周期为 25ns，如果用户设计的 APB 时钟频率不是 40MHz，请按照如下方法设置寄存器。

9.3.1 峰值抑制宽度

表 9-12 描述了通过输入滤波器抑制峰值脉冲宽度。

对于快速模式和超快速模式，必须抑制小于 50ns 的峰值（假设 TPM == 0）。

T_SP = 50ns / (25ns * (TPM + 1)) = 2

表 9-12 Timing Parameters for Spike Suppression

Symbol	Parameter	Standard-mode		Fast-mode		Fast-mode Plus		Unit
		Min	Max	Min	Max	Min	Max	
t_{SP}	Pulse width of spikes that must be suppressed by the input filter.	-	-	0	50	0	50	ns

9.3.2 数据建立时间

数据建立时间定义了在 SCL 上升沿之前，SDA 应该保持稳定的时间。

表 9-13 描述了数据建立时间的时序参数。

表 9-10 描述了数据建立时间等式，如下所示：

Setup time = (2 * t_{pclk}) + (2 + T_SP + T_SUDAT) * t_{pclk} * (TPM + 1)

例如标准模式（假设 TPM == 0），

250ns = 50ns + (2 + 2 + T_SUDAT) * 25ns

计算得出，

T_SUDAT = 4

其他模式，T_SUDAT 计算方法类似。

表 9-13 Timing Parameters for the Data Setup Time

Symbol	Parameter	Standard-mode		Fast-mode		Fast-mode Plus		Unit
		Min	Max	Min	Max	Min	Max	
t_{SUDAT}	Data setup time	250	-	100	-	50	-	ns

9.3.3 数据保持时间

数据保持时间，定义了在 SCL 下降沿之后，SDA 应该保持稳定的时间。

表 9-14 描述了数据保持时间的时序参数。

表 9-10 描述了数据保持时间等式，如下所示：

Hold time = (2 * t_{pclk}) + (2 + T_SP + T_HDDAT) * t_{pclk} * (TPM + 1)

例如标准模式（假设 TPM == 0），

300ns = 50ns + (2 + 2 + T_HDDAT) * 25ns

计算得出，

T_HDDAT = 6

其他模式，T_HDDAT 计算方法类似。

表 9-14 Timing Parameters for the Data Hold Time

Symbol	Parameter	Standard-mode		Fast-mode		Fast-mode Plus		Unit
		Min	Max	Min	Max	Min	Max	
t_{HDDAT}	Data hold time	300	-	300	-	0	-	ns

9.3.4 I2C 总线时钟频率

I2C 总线时钟频率，通过参数 t_{HIGH} 和 t_{LOW} 定义，表 9-15 描述 Setup 寄存器中计算 I2C 总线时钟频率的 T_SCLHi 和 T_SCLRatio。

表 9-15 Timing Parameters for the SCL Clock

Symbol	Parameter	Standard-mode		Fast-mode		Fast-mode Plus		Unit
		Min	Max	Min	Max	Min	Max	
t_{HIGH}	HIGH period of the SCL clock	4.0	-	0.6	-	0.26	-	us
t_{LOW}	LOW period of the SCL clock	4.7	-	1.3	-	0.5		us

如果是标准模式，t_{HIGH} 和 t_{LOW} 的最低需求接近，所以 T_SCLRatio 可以设置为 0 来简化计算。表 9-10 中描述的 SCL 周期等式，如下所示：

SCL HIGH period = (2 * t_{pclk}) + (2 + T_SP + T_SCLHi) * t_{pclk} * (TPM + 1) >= 4000ns

SCL LOW period = (2 * t_{pclk}) + (2 + T_SP + T_SCLHi * ratio) * t_{pclk} * (TPM + 1) >= 4700ns

替换等式 2 的 T_SP，ratio = 1，t_{pclk} = 25ns，则等式简化为（假设 TPM == 0）：

50ns + (2 + 2 + T_SCLHi) * 25ns * (0 + 1) >= 4000ns

50ns + (2 + 2 + T_SCLHi * 1) * 25ns * (0 + 1) >= 4700ns

T_SCLHi >= 182

如果是快速模式，t_{LOW} 的最低需求为 t_{HIGH} 的 2 倍，所以 T_SCLRatio 可以设置为 1。SCL 周期等式，如下所示：

SCL HIGH period = (2 * t_{pclk}) + (2 + T_SP + T_SCLHi) * t_{pclk} * (TPM + 1) >= 600ns

SCL LOW period = (2 * t_{pclk}) + (2 + T_SP + T_SCLHi * ratio) * t_{pclk} * (TPM + 1) >= 1300ns

替换等式 2 的 T_SP，ratio = 2，t_{pclk} = 25ns，则等式简化为（假设 TPM == 0）：

50ns + (4 + T_SCLHi) * 25ns * (0 + 1) >= 600ns

50ns + (4 + T_SCLHi * 2) * 25ns * (0 + 1) >= 1300ns

T_SCLHi >= 23

如果是超快速模式，T_SCLHi 可以使用快速模式的计算方法。

9.3.5 时序参数乘法器

为满足 I2C 总线的时序要求，计算 TPM 时，假设 T_SCLHi 为最大值（511），T_SP 为最小有效值（1）。

例如参考表 9-15 和表 9-10，假设 t_{SP} = 50ns，t_{HIGH} = 4.0us，t_{LOW} = 4.7us。

SCL HIGH period = 2 * t_{pclk} + (2 + T_SP + T_SCLHi) * t_{pclk} * (TPM + 1) >= 4.0us

SCL LOW period = 2 * t_{pclk} + (2 + T_SP + T_SCLHi * ratio) * t_{pclk} * (TPM + 1) >= 4.7us

已知 T_SP = 1，T_SCLHi = 511，ratio = 1，t_{pclk} = 2ns。

SCL HIGH period = 2 * 2ns + (2 + 1 + 511) * 2ns * (TPM + 1) >= 4.0us

SCL LOW period = 2 * 2ns + (2 + 1 + 511 * 1) * 2ns * (TPM + 1) >= 4.7us

因此，

2 * 2ns + (2 + 1 + 511) * 2ns * (TPM + 1) >= 4000ns

2 * 2ns + (2 + 1 + 511 * 1) * 2ns * (TPM + 1) >= 4700ns

所以：

(TPM + 1) >= 4.57

TPM >= 3.57

计算得出 TPM = 4，以此来计算表 9-10 所示的所有时序参数。

即，计算 T_SP：

T_SP >= 50ns / ((TPM+1) * tpclk) = 50ns / ((4 + 1) * 2ns) = 5

计算 T_SCLHi：

SCL HIGH period = 2 * t_{pclk} + (2 + T_SP + T_SCLHi) * t_{pclk} * (TPM + 1) >= 4000ns

SCL LOW period = 2 * t_{pclk} + (2 + T_SP + T_SCLHi * ratio) * t_{pclk} * (TPM + 1) >= 4700ns

因此，

SCL HIGH period = 2 * 2ns + (2 + 5 + T_SCLHi) * 2ns * (4 + 1) >= 4000ns

SCL LOW period = 2 * 2ns + (2 + 5 + T_SCLHi) * 2ns * (4 + 1) >= 4700ns

合并两等式得出：

2 * 2ns + (2 + 5 + T_SCLHi) * 2ns * (4 + 1) >= 4700ns

7 + T_SCLHi >= (4700 – 4) / (2 * 5)

T_SCLHi >= 462.6

因此，T_SCLHi = 463，T_SP = 5，TPM = 4。

9.4 驱动函数定义

9.4.1 驱动函数定义

I2C 驱动函数定义如表 9-16 所示。I2C 驱动函数定义，位于 bsp\driver\ae350\i2c_ae350.c、i2c_ae350.h 和 bsp\driver\include\Driver_I2C.h。

表 9-16 驱动函数定义

驱动函数	描述
GetVersion	获取 I2C 驱动的版本信息
GetCapabilities	获取 I2C 驱动的功能信息
Initialize	初始化 I2C 接口
Uninitialize	卸载 I2C 接口
PowerControl	指定 I2C 接口的功耗模式
MasterTransmit	I2C 作为主机模式发送数据
MasterReceive	I2C 作为主机模式接收数据
SlaveTransmit	I2C 作为从机模式发送数据

驱动函数	描述
SlaveReceive	I2C 作为从机模式接收数据
GetDataCount	获取 I2C 传输数据的数量
Control	配置 I2C 接口的设置，执行指定的操作
GetStatus	获取 I2C 接口的状态

9.4.2 驱动函数描述

以下各节详细描述 I2C 的驱动函数定义。

GetVersion

GetVersion 函数定义如表 9-17 所示。

表 9-17 GetVersion 函数定义

原型	AE350_DRIVER_VERSION (*GetVersion)(void)
描述	获取 I2C 驱动的版本信息
参数	无
返回值	I2C 驱动实现的版本信息

GetCapabilities

GetCapabilities 函数定义如表 9-18 所示。

表 9-18 GetCapabilities 函数定义

原型	AE350_I2C_CAPABILITIES (*GetCapabilities) (void)
描述	获取 I2C 驱动的功能信息
参数	无
返回值	I2C 驱动的功能信息

Initialize

Initialize 函数定义如表 9-19 所示。

表 9-19 Initialze 函数定义

原型	int32_t (*Initialize)(AE350_I2C_SignalEvent cb_event)
描述	初始化 I2C 接口
参数	cb_event：指向 AE350_I2C_SignalEvent 回调函数的指针
返回值	如果发生执行错误，返回一个负值

Uninitialize

Uninitialize 函数定义如表 9-20 所示。

表 9-20 Uninitialize 函数定义

原型	int32_t (*Uninitialize)(void)
描述	卸载 I2C 接口
参数	无
返回值	如果发生执行错误，返回一个负值

PowerControl

PowerControl 函数定义如表 9-21 所示。

表 9-21 PowerControl 函数定义

原型	int32_t (*PowerControl)(AE350_POWER_STATE state)
描述	指定 I2C 接口的功耗模式
参数	state：I2C 接口功耗模式，包括： AE350_POWER_FULL: Set up peripherals for data transfers, enable interrupts and DMA AE350_POWER_LOW: enable power-saving AE350_POWER_OFF: Terminate pending data transfers and disable peripherals, related interrupts and DMA
返回值	如果发生执行错误，返回一个负值

MasterTransmit

MasterTransmit 函数定义如表 9-22 所示。

表 9-22 MasterTransmit 函数定义

原型	int32_t (*MasterTransmit)(uint32_t addr, const uint8_t *data, uint32_t num, bool xfer_pending)
描述	I2C 作为主机模式发送数据
参数	Addr: 从机地址 Data: 指向发送数据缓存区的指针 Num: 发送数据的长度 xfer_pending: 传输最后阶段设置是否产生停止状态
返回值	如果发生执行错误，返回一个负值

MasterReceive

MasterReceive 函数定义如表 9-23 所示。

表 9-23 MasterReceive 函数定义

原型	int32_t (*MasterReceive)(uint32_t addr, uint8_t *data, uint32_t num, bool xfer_pending)
描述	I2C 作为主机模式接收数据
参数	Addr: 从机地址 Data: 指向接收数据缓存区的指针 Num: 接收数据的长度 xfer_pending: 传输最后阶段设置是否产生停止状态
返回值	如果发生执行错误，返回一个负值

SlaveTransmit

SlaveTransmit 函数定义如表 9-24 所示。

表 9-24 SlaveTransmit 函数定义

原型	int32_t (*SlaveTransmit)(const uint8_t *data, uint32_t num)
描述	I2C 作为从机模式发送数据
参数	data：指向发送到主机的数据缓存区的指针 num：发送数据的长度
返回值	如果发生执行错误，返回一个负值

SlaveReceive

SlaveReceive 函数定义如表 9-25 所示。

表 9-25 SlaveReceive 函数定义

原型	int32_t (*SlaveReceive)(uint8_t *data, uint32_t num)
描述	I2C 作为从机模式接收数据
参数	data：指向接收数据缓存区的指针 num：接收数据的长度
返回值	如果发生执行错误，返回一个负值

GetDataCount

GetDataCount 函数定义如表 9-26 所示。

表 9-26 GetDataCount 函数定义

原型	uint32_t (*GetDataCount)(void)
描述	获取 I2C 传输数据的数量
参数	无
返回值	I2C 接口传输的数据长度

Control

Control 函数定义如表 9-27 所示。

表 9-27 Control 函数定义

原型	int32_t (*Control)(uint32_t control, uint32_t arg)
描述	配置 I2C 接口的设置，执行指定的操作
参数	Control: I2C 驱动接口的一种设置或执行的一种操作 Arg: 指定设置或操作的附加信息
返回值	如果发生执行错误，返回一个负值

“control”和“arg”设置与操作如表 9-28 所示。

表 9-28 Control Settings or Operations

Options for Control	arg Specifies	Settings or Operations
AE350_I2C_OWN_ADDRESS	slave address	Sets the slave address
AE350_I2C_BUS_SPEED	bus speed	Sets the bus speed
AE350_I2C_BUS_CLEAR	-	Clears the bus by sending nine clock pulses
AE350_I2C_ABORT_TRANSFER	-	Aborts the data transfer between the master and slave

GetStatus

GetStatus 函数定义如表 9-29 所示。

表 9-29 GetStatus 函数定义

原型	AE350_I2C_STATUS (*GetStatus)(void)
描述	获取 I2C 接口的状态

参数	无
返回值	I2C 接口的当前状态

10 SPI

10.1 简介

Gowin RiscV_AE350_SOC 包含一个 SPI 控制器，支持主机模式和从机模式。如果作为 SPI 主机模式，控制器可以连接多个 SPI 从机设备；如果作为 SPI 从机模式，控制器可以响应主机请求进行数据交换。

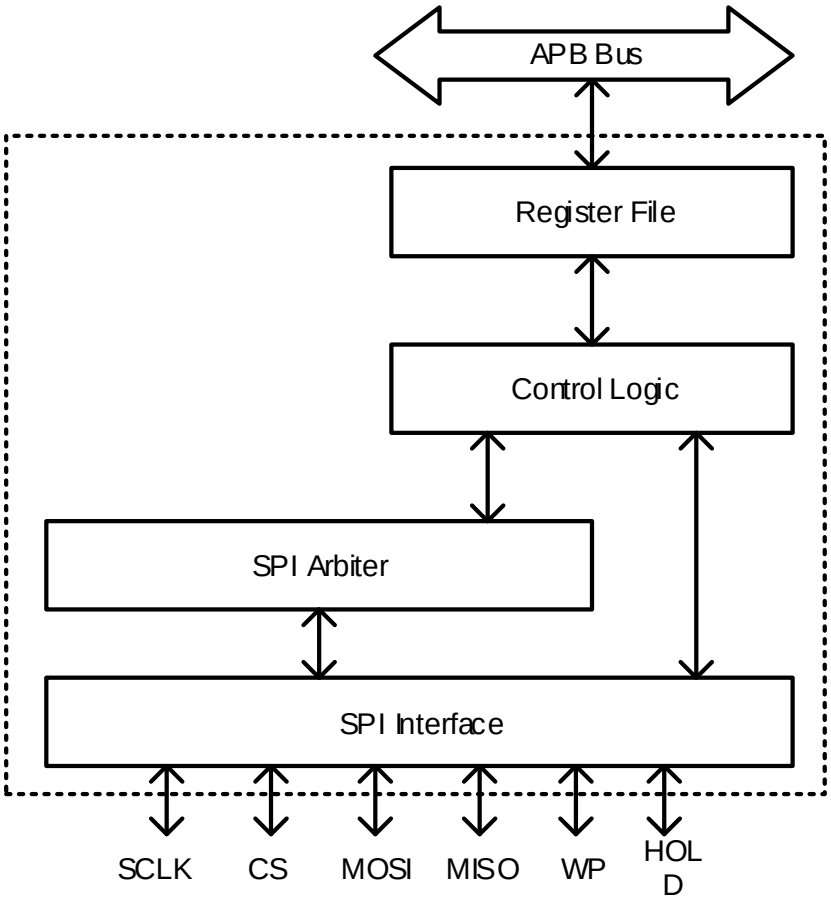
10.1.1 特征

- 支持 AMBA 3.0 APB 总线协议规范
- 支持 MSB/LSB 优先传输
- 支持 DMA 传输
- 支持可编程的 SPI SCLK
- 支持 SPI 从机模式
- 支持 Quad I/O SPI 接口
- 支持 128 位 TX/RX FIFO 深度
- 支持 32 位 TX/RX FIFO 宽度

10.1.2 结构框图

SPI 结构框图如图 10-1 所示。

图 10-1 SPI 结构框图



10.1.3 功能描述

SPI 控制器包括主机模式、从机模式和 Quad I/O 模式。

SPI 控制器可以作为 SPI 主机模式，在 SPI 总线上发起 SPI 传输。SPI 传输格式和接口时序，可以通过 APB 编程端口来编程配置。

SPI 控制器可以作为 SPI 从机模式，接收如表 10-1 所示的常用命令。此外控制器还支持用户自定义的命令，其中从机的数据字段格式由传输控制寄存器定义。

表 10-1 Supported Commands under the Slave Mode

Slave Command Name	OP Code	Slave Data
Read Status Single IO	0x05	32-bit Status
Read Status Dual IO	0x15	32-bit Status
Read Status Quad IO	0x25	32-bit Status
Read Data Single IO	0x0B	Replay data from the Data Register in the FIFO manner
Read Data Dual IO	0x0C	Replay data from the Data

Slave Command Name	OP Code	Slave Data
		Register in the FIFO manner
Read Data Quad IO	0x0E	Replay data from the Data Register in the FIFO manner
Write Data Single IO	0x51	Data saved to the Data Register in the FIFO manner
Write Data Dual IO	0x52	Data saved to the Data Register in the FIFO manner
Write Data Quad IO	0x54	Data saved to the Data Register in the FIFO manner
User-defined	Any 8-bit numbers other than the listed OP Codes	Depending on the Transfer Controller Register

Quad I/O 模式通过将主机输入/从机输出（MISO）、主机输出/从机输入（MOSI）、写保护（WP）和 HOLD 信号作为双向信号线，四倍 SPI 带宽。

10.2 寄存器定义

10.2.1 寄存器定义

SPI 寄存器定义如表 10-2 所示。SPI 寄存器定义位于 bsp\ae350\ae350.h。

表 10-2 寄存器定义

地址偏移	寄存器名称	描述
0x00	IDREV	ID and Revision Register
0x04~0x0C	-	Reserved
0x10	TRANSFMT	SPI Transfer Format Register
0x14	DIRECTIO	SPI Direct IO Control Register
0x18~0x1C	-	Reserved
0x20	TRANSCTRL	SPI Transfer Control Register
0x24	CMD	SPI Command Register
0x28	ADDR	SPI Address Register
0x2C	DATA	SPI Data Register
0x30	CTRL	SPI Control Register
0x34	STATUS	SPI Status Register

地址偏移	寄存器名称	描述
0x38	INTREN	SPI Interrupt Enable Register
0x3C	INTRST	SPI Interrupt Status Register
0x40	TIMING	SPI Interface Timing Register
0x44~0x4C	-	Reserved
0x50	MEMCTRL	SPI Memory Access Control Register
0x54~0x5C	-	Reserved
0x60	SLVST	SPI Slave Status Register
0x64	SLVDATACNT	SPI Slave Data Count Register
0x68~0x78	-	Reserved
0x7C	CONFIG	Configuration Register

10.2.2 寄存器描述

以下各节详细描述 SPI 寄存器定义。

寄存器类型缩略语概括如下：

- RO: Read-only
- R/W: Readable and writable
- W1C: Write 1 to clear

ID 和 Revision 寄存器（0x00）

ID 和 Revision 寄存器用于保存 ID 和 Revision 编号，初始值依赖于所用版本。ID 和 Revision 寄存器定义如表 10-3 所示。

表 10-3 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:8	RO	ID number for SPI	0x020020
RevMajor	7:4	RO	Major revision number	Revision Dependent
RevMinor	3:0	RO	Minor revision number	Revision Dependent

SPI Transfer Format 寄存器（0x10）

SPI Transfer Format 寄存器定义了 SPI 传输格式。SPI Transfer Format 寄存器定义如表 10-4 所示。

表 10-4 SPI Transfer Format Register

Name	Bit	Type	Description	Reset
-	31:18	-	Reserved	-
AddrLen	17:16	RW	Address length in bytes 0: 1 byte 1: 2 bytes 2: 3 bytes 3: 4 bytes	0x2
-	15:13	-	Reserved	-
DataLen	12:8	RW	The length of each data unit in bits The actual bit number of a data unit is (DataLen + 1)	0x07
DataMerge	7	RW	Enable Data Merge mode, which does automatic data split on write and data coalescing on read. This bit only takes effect when DataLen = 0x7. Under Data Merge mode, each write to the Data Register will transmit all four bytes of the write data; each read from the Data Register will retrieve four bytes of received data as a single word data. When Data Merge mode is disabled, only the least (DataLen + 1) significant bits of the Data Register are valid for read/write operations; no automatic data split/coalescing will be performed.	0x1
-	6:5	-	Reserved	-
MOSIBiDir	4	RW	Bi-directional MOSI in regular (single) mode 0: MOSI is uni-directional in regular mode. 1: MOSI is bi-directional signal in regular mode. This bi-directional signal replaces the two uni-directional data signals, MOSI and MISO	0x0
LSB	3	RW	Transfer data width the least significant bit first 0: Most significant bit first 1: Least significant bit first	0x0

Name	Bit	Type	Description	Reset
SlvMode	2	RW	SPI Master/Slave mode selection 0: Master mode 1: Slave mode	-
CPOL	1	RW	SPI Clock Polarity 0: SCLK is LOW in the idle states 1: SCLK is HIGH in the idle states	-
CPHA	0	RW	SPI Clock Phase 0: Sampling data at odd SCLK edges 1: Sampling data at even SCLK edges	-

SPI Direct IO Control 寄存器（0x14）

SPI Direct IO Control 寄存器开启 SPI 接口信号的直接控制。SPI Direct IO Control 寄存器定义如表 10-5 所示。

表 10-5 SPI Direct IO Control Register

Name	Bit	Type	Description	Reset
-	31:25	-	Reserved	-
DirectIOEn	24	RW	Enable Direct IO 0: Disable 1: Enable	0x0
-	23:22	-	Reserved	-
HOLD_OE	21	RW	Output enable for the SPI Flash hold signal	0x0
WP_OE	20	RW	Output enable for the SPI Flash write protect signal	0x0
MISO_OE	19	RW	Output enable for the SPI MISO signal	0x0
MOSI_OE	18	RW	Output enable for the SPI MOSI signal	0x0
SCLK_OE	17	RW	Output enable for the SPI SCLK signal	0x0
CS_OE	16	RW	Output enable for the CS (chip select) signal	0x0
-	15:14	-	Reserved	-
HOLD_O	13	-	Output value for the SPI Flash hold signal	0x1
WP_O	12	-	Output value for the SPI Flash write project signal	0x1
MISO_O	11	-	Output value for the SPI MISO signal	0x0
MOSI_O	10	-	Output value for the SPI MOSI signal	0x0

Name	Bit	Type	Description	Reset
SCLK_O	9	-	Output value for the SPI SCLK signal	0x0
CS_O	8	-	Output value for the SPI CS (chip select) signal	0x1
-	7:6	-	Reserved	-
HOLD_I	5	RO	Status of the SPI Flash hold signal	-
WP_I	4	RO	Status of the SPI Flash write protect signal	-
MISO_I	3	RO	Status of the SPI MISO signal	-
MOSI_I	2	RO	Status of the SPI MOSI signal	-
SCLK_I	1	RO	Status of the SPI SCLK signal	-
CS_I	0	RO	Status of the SPI CS (chip select)	-

SPI Transfer Control 寄存器（0x20）

SPI Transfer Control 寄存器控制 SPI 传输相位，具体参考 SPI Command 寄存器（0x24）。SPI Transfer Control 寄存器定义如表 10-6 所示。

表 10-6 SPI Transfer Control Register

Name	Bit	Type	Description	Reset
SlvDataOnly	31	RW	Data-only mode (slave mode only) 0: Disable the data-only mode 1: Enable the data-only mode Note: This mode only works in the uni-directional regular (single) mode so MOSIiDir, DualQuad and TransMode should be set to 0	0x0
CmdEn	30	RW	SPI command phase enable (Master mode only) 0: Disable the command phase 1: Enable the command phase	0x0
AddrEn	29	RW	SPI address phase enable (Master mode only) 0: Disable the address phase 1: Enable the address phase	0x0
AddrFmt	28	RW	SPI address phase format (Master mode only) 0: Address phase is the regular (single)	0x0

Name	Bit	Type	Description	Reset
			mode 1: The format of the address phase is the same as the data phase (DualQuad)	
TransMode	27:24	RW	Transfer mode The transfer sequence could be: 0: Write and read at the same time 1: Write only 2: Read only 3: Write, Read 4: Read, Write 5: Write, Dummy, Read 6: Read, Dummy, Write 7: None Data (must enable CmdEn or AddrEn in master mode) 8: Dummy, Write 9: Dummy, Read 0xa~0xf: Reserved	0x0
DualQuad	23:22	RW	SPI data phase format 0: Regular (Single) mode 1: Dual I/O mode 2: Quad I/O mode 3: Reserved	0x0
TokenEn	21	RW	Token transfer enable (Master mode only) Append a one-byte special token following the address phase for SPI read transfers. The value of the special token should be selected in TokenValue. 0: Disable the one-byte special token 1: Enable the one-byte special token	0x0
WrTranCnt	20:12	RW	Transfer count for write data WrTranCnt indicates the number of units of data to be transmitted to the SPI bus from the Data Register. The actual transfer count is (WrTranCnt + 1). WrTranCnt only takes effect when TransMode is 0, 1, 3, 4, 5, 6 or 8. The size (bit-width) of a data unit is defined	

Name	Bit	Type	Description	Reset
			by the DataLen field of the Transfer Format Register. For TransMode 0, WrTranCnt must be equal to RdTranCnt.	
TokenValue	11	RW	Token value (Master mode only) The value of the one-byte special token following the address phase for SPI read transfers. 0: token value = 0x00 1: token value = 0x69	0x0
DummyCnt	10:9	RW	Dummy data count. The actual dummy count is (DummyCnt + 1). The number of dummy cycles on the SPI interface will be (DummyCnt+1) * ((DataLen+1)/SPI IO width) The Data pins are put into the high impedance during the dummy data phase. DummyCnt is only used for TransMode 5, 6, 8 and 9, which has dummy data phases. The =dummy cycle settings under some common transfer formats is as shown in Table 10-7.	0x0
RdTranCnt	8:0	RW	Transfer count for read data RdTranCnt indicates the number of units of data to be received from SPI bus and stored to the Data Register. The actual received count is (RdTranCnt+1). RdTransCnt only takes effect when TransMode is 0, 2, 3, 4, 5, 6 or 9. The size (bit-width) of a data unit is defined by the DataLen field of the Transfer Format Register. For TransMode 0, WrTranCnt must equal RdTranCnt.	0x0

表 10-7 Dummy Cycle Settings under Some Common Transfer Formats

DummyCnt+1	DataLen+1	DualQuad	#Dummy Cycles on the SPI Interface
1	8	Regular/Single	8
1	8	Dual	4
1	8	Quad	2
2	8	Quad	4
3	8	Quad	6
1	32	Quad	8

SPI Command 寄存器（0x24）

SPI Command 寄存器中的写操作，触发 SPI 传输。即使命令阶段没有开启，该寄存器也必须写入一个空值来开启一个 SPI 传输。当 SPI 控制器作为从机模式时，SPI Command 寄存器保存最后一次接收到的 SPI 事务的命令字段。

SPI Command 寄存器定义如表 10-8 所示。

表 10-8 SPI Command Register

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	-
CMD	7:0	RW	SPI Command	0x0

SPI Address 寄存器（0x28）

SPI Address 寄存器定义如表 10-9 所示。

表 10-9 SPI Address Register

Name	Bit	Type	Description	Reset
ADDR	31:0	RW	SPI Address (Master mode only)	0x0

SPI Data 寄存器（0x2C）

当控制器为数据归并模式时 SPI Data 寄存器的字节存储顺序为小端模式。SPI Data 寄存器定义如表 10-10 所示。

表 10-10 SPI Data Register

Name	Bit	Type	Description	Reset
DATA	31:0	RW	Data to transmit or the received data For writes, data is enqueued to the TX FIFO. The least significant byte is always transmitted first. If the TX FIFO is full and the SPIActive bit of the status register is 1, the ready signal hready/pready will be deasserted to insert wait states to the transfer. For reads, data is read and dequeued from the RX FIFO. The least significant byte is the first received byte. If the RX FIFO is empty and the SPIActive bit of the status register is 1, the ready signal hready/pready will be deasserted to insert wait states to the transfer. The FIFOs decouple the speed of the SPI transfers and the software's generation/consumption of data. When the TX FIFO is empty, SPI transfers will hold until more data is written to the TX FIFO; when the RX FIFO is full, SPI transfers will hold until there is more room in the RX FIFO. If more data is written to the TX FIFO than the write transfer count (WrTranCnt), the remaining data will stay in the TX FIFO for the next transfer or until the TX FIFO is reset.	0x0

SPI Control 寄存器（0x30）

SPI Control 寄存器定义如表 10-11 所示。

表 10-11 SPI Control Register

Name	Bit	Type	Description	Reset
-	31:24	-	Reserved	-
TXTHRES	23:16	RW	Transmit (TX) FIFO Threshold The TXFIFOInt interrupt or DMA request would be issued to replenish the TX FIFO when the TX data count is less than or equal to the TX FIFO threshold.	0x0
RXTHRES	15:8	RW	Receive (RX) FIFO Threshold The RXFIFOInt interrupt or DMA request would be issued for consuming the RX	0x0

Name	Bit	Type	Description	Reset
			FIFO when the RX data count is more than or equal to the RX FIFO threshold.	
-	7:5	-	Reserved	-
TXDMAEN	4	RW	TX DMA enable	0x0
RXDMAEN	3	RW	RX DMA enable	0x0
TXFIFORST	2	RW	Transmit FIFO reset Write 1 to reset. It is automatically cleared to 0 after the reset operation completes.	0x0
RXFIFORST	1	RW	Receive FIFO reset Write 1 to reset. It is automatically cleared to 0 after the reset operation completes.	0x0
SPIRST	0	RW	SPI reset Write 1 to reset. It is automatically cleared to 0 after the reset operation completes.	0x0

SPI Status 寄存器（0x34）

SPI Status 寄存器定义如表 10-12 所示。

表 10-12 SPI Status Register

Name	Bit	Type	Description	Reset
-	31:30	-	Reserved	-
TXNUM[7:6]	29:28	RO	Number of valid entries in the Transmit FIFO	0x0
-	27:26	-	Reserved	-
RXNUM[7:6]	25:24	RO	Number of valid entries in the Receive FIFO	0x0
TXFULL	23	RO	Transmit FIFO Full flag	0x0
TXEMPTY	22	RO	Transmit FIFO Empty flag	0x1
TXNUM[5:0]	21:16	RO	Number of valid entries in the Transmit FIFO	0x0
RXFULL	15	RO	Receive FIFO Full flag	0x0
RXEMPTY	14	RO	Receive FIFO Empty flag	0x1
RXNUM[5:0]	13:8	RO	Number of valid entries in the Receive FIFO	0x0
-	7:1	RO	Reserved	-
SPIActive	0	RO	SPI register programming is in progress. In master mode, SPIActive becomes 1 after	0x0

Name	Bit	Type	Description	Reset
			the SPI command register is written and becomes 0 after the transfer is finished. In slave mode, SPIActive becomes 1 after the SPI CS signal is asserted and becomes 0 after the SPI CS signal is deasserted. Note that due to clock synchronization, it may take at most two spi_clock cycles for SPIActive to change when the corresponding condition happens. Note this bit stays 0 when Direct IO Control or the memory-mapped interface is used.	

SPI Interrupt Enable 寄存器（0x38）

SPI Interrupt Enable 寄存器定义如表 10-13 所示。

表 10-13 SPI Interrupt Enable Register

Name	Bit	Type	Description	Reset
-	31:6	-	Reserved	-
SlvCmdEn	5	RW	Enable the Slave Command Interrupt. Control whether interrupts are triggered whenever slave commands are received. (Slave mode only)	0x0
EndIntEn	4	RW	Enable the End of SPI Transfer interrupt. Control whether interrupts are triggered when SPI transfers end. (In slave mode, end of read status transaction doesn't trigger this interrupt.)	0x0
TXFIFOIntEn	3	RW	Enable SPI Transmit FIFO Threshold interrupt. Control whether interrupts are triggered when the valid entries are less than or equal to TX FIFO threshold.	0x0
RXFIFOIntEn	2	RW	Enable the SPI Receive FIFO Threshold interrupt. Control whether interrupts are triggered when the valid entries are greater than or equal to the RX FIFO threshold.	0x0

Name	Bit	Type	Description	Reset
TXFIFOURIntEn	1	RW	Enable the SPI Transmit FIFO Underrun interrupt. Control whether interrupts are triggered when the Transmit FIFO run out of data. (Slave mode only)	0x0
RXFIFOORIntEn	0	RW	Enable the SPI Receive FIFO Overrun interrupt. Control whether interrupts are triggered when the Receive FIFO overflows. (Slave mode only)	0x0

SPI Interrupt Status 寄存器（0x3C）

SPI Interrupt Status 寄存器定义如表 10-14 所示。

表 10-14 SPI Interrupt Status Register

Name	Bit	Type	Description	Reset
-	31:6	-	Reserved	-
SlvCmdInt	5	W1C	Slave Command Interrupt. This bit is set when Slave Command interrupts occur. (Slave mode only)	0x0
EndInt	4	W1C	End of SPI Transfer interrupt. This bit is set when End of SPI Transfer interrupts occur.	0x0
TXFIFOInt	3	W1C	TX FIFO Threshold interrupt. This bit is set when TX FIFO Threshold interrupts occur.	0x0
RXFIFOInt	2	W1C	RX FIFO Threshold interrupt. This bit is set when RX FIFO Threshold interrupts occur.	0x0
TXFIFOURInt	1	W1C	TX FIFO Underrun interrupt. This bit is set when TX FIFO Underrun interrupts occur. (Slave mode only)	0x0
RXFIFOORInt	0	W1C	RX FIFO Overrun interrupt. This bit is set when RX FIFO Overrun	0x0

Name	Bit	Type	Description	Reset
			interrupts occur. (Slave mode only)	

SPI Interface Timing 寄存器（0x40）

SPI Interface Timing 寄存器控制 SPI 接口时序以满足 SPI 从机接口时序要求。仅主机模式时需要编程该寄存器。SPI Interface Timing 寄存器定义如表 10-15 所示。

表 10-15 SPI Interface Timing Register

Name	Bit	Type	Description	Reset
-	31:14	-	Reserved	-
CS2SCLK	13:12	RW	The minimum time between the edges of SPI CS and the edges of SCLK. The actual duration is (SCLK period / 2) × (CS2SCLK + 1)	Configuration dependent
CSHT	11:8	RW	The minimum time that SPI CS should stay HIGH. The actual duration is (SCLK period / 2) × (CSHT + 1)	Configuration dependent
SCLK_DIV	7:0	RW	The clock frequency ratio between the clock source and SPI interface SCLK. SCLK period = ((SCLK_DIV + 1) × 2) × (Period of the SPI clock source) The SCLK_DIV value 0xff is a special value which indicates that the SCLK frequency should be the same as the spi_clock frequency.	Configuration dependent

SPI Memory Access Control 寄存器（0x50）

SPI Memory Access Control 寄存器定义了用于内存映射的 AHB/EILM 读访问的 SPI 命令。

当正在编程该寄存器或 SPI Interface Timing 寄存器（0x40）时，应停止内存映射的 AHB/EILM 读访问。当 MemCtrlChg 位清零后，才能继续

AHB/EILM 读访问。

SPI Memory Access Control 寄存器定义如表 10-16 所示。

表 10-16 SPI Memory Access Control Register

Name	Bit	Type	Description	Reset
-	31:9	-	Reserved	-
MemCtrlChg	8	RO	This bit is set when this register (0x50) or the SPI Interface Timing Register (0x40) is written; it is automatically cleared when the new programming takes effect.	0
-	7:4	-	Reserved	-
MemRdCmd	3:0	RW	Selects the SPI command for serving the memory-mapped reads on the AHB/EILM bus The command encoding table is listed in Table 10-16 The latency of each command is listed in Table 10-17	Configuration dependent

表 10-17 Supported SPI Read Commands for Memory-Mapped AHB/EILM Reads

MemRdCmd	Command	Address	Dummy	Data
0	0x03	3 bytes in Regular mode	N/A	Regular mode
1	0x0B	3 bytes in Regular mode	1 byte in Regular mode	Regular mode
2	0x3B	3 bytes in Regular mode	1 byte in Regular mode	Dual mode
3	0x6B	3 bytes in Regular mode	1 byte in Regular mode	Quad mode
4	0xBB	(3-byte address + 1-byte 0) in Dual mode	N/A	Dual mode
5	0xEB	(3-byte address + 1-byte 0) in Quad mode	2 bytes in Quad mode	Quad mode
6-7	Reserved	-	-	-
8	0x13	4 bytes in Regular mode	N/A	Regular mode

MemRdCmd	Command	Address	Dummy	Data
9	0x0C	4 bytes in Regular mode	1 byte in Regular mode	Regular mode
10	0x3C	4 bytes in Regular mode	1 byte in Regular mode	Dual mode
11	0x6C	4 bytes in Regular mode	1 byte in Regular mode	Quad mode
12	0xBC	(4-byte address + 1-byte 0) in Dual mode	N/A	Dual mode
13	0xEC	(4-byte address + 1-byte 0) in Quad mode	2 bytes in Quad mode	Quad mode
14-15	Reserved	-	-	-

表 10-18 Latency of a 4 Bytes Data Transfer through the AHB/EILM Memory Read Port

Command	Non-sequential	Sequential	Sequential (prefetched*)
0x03	8 BUS_CLK + 10 SPI_CLK + 64 SCLK	3 BUS_CLK + 32 SCLK	1 BUS_CLK
0x0B	8 BUS_CLK + 10 SPI_CLK + 72 SCLK	3 BUS_CLK + 32 SCLK	1 BUS_CLK
0x3B	8 BUS_CLK + 10 SPI_CLK + 56 SCLK	3 BUS_CLK + 16 SCLK	1 BUS_CLK
0x6B	8 BUS_CLK + 10 SPI_CLK + 48 SCLK	3 BUS_CLK + 8 SCLK	1 BUS_CLK
0xBB	8 BUS_CLK + 10 SPI_CLK + 40 SCLK	3 BUS_CLK + 16 SCLK	1 BUS_CLK
0xEB	8 BUS_CLK + 10 SPI_CLK + 28 SCLK	3 BUS_CLK + 8 SCLK	1 BUS_CLK
0x13	8 BUS_CLK + 10 SPI_CLK + 72 SCLK	3 BUS_CLK + 32 SCLK	1 BUS_CLK
0x0C	8 BUS_CLK + 10 SPI_CLK + 80 SCLK	3 BUS_CLK + 32 SCLK	1 BUS_CLK
0x3C	8 BUS_CLK + 10 SPI_CLK + 64 SCLK	3 BUS_CLK + 16 SCLK	1 BUS_CLK
0x6C	8 BUS_CLK + 10 SPI_CLK	3 BUS_CLK +	1 BUS_CLK

Command	Non-sequential	Sequential	Sequential (prefetched*)
	+ 56 SCLK	8 SCLK	
0xBC	8 BUS_CLK + 10 SPI_CLK + 44 SCLK	3 BUS_CLK + 16 SCLK	1 BUS_CLK
0xEC	8 BUS_CLK + 10 SPI_CLK + 30 SCLK	3 BUS_CLK + 8 SCLK	1 BUS_CLK

BUS_CLK：AHB/EILM 总线时钟周期；SCLK：SCLK 时钟周期。

SPI Slave Status 寄存器（0x60）

SPI Slave Status 寄存器保存从机的状态，SPI 主机可以通过读状态命令获取这些状态。SPI Slave Status 寄存器定义如表 10-19 所示。

表 10-19 SPI Slave Status Register

Name	Bit	Type	Description	Reset
-	31:19	-	Reserved	-
UnderRun	18	W1C	Data underrun occurs in the last transaction	0
OverRun	17	W1C	Data overrun occurs in the last transaction	0
Ready	16	RW	Set this bit to indicate that the SPI is ready for data transaction. When an SPI transaction other than slave status-reading command ends, this bit will be cleared to 0.	0
USR_Status	15:0	RW	User defined status flags	0

SPI Slave Data Count 寄存器（0x64）

SPI Slave Data Count 寄存器描述从机模式下的读/写事务的数据数量，根据数据数量信息访问 Data 寄存器。SPI Slave Data Count 寄存器定义如表 10-20 所示。

表 10-20 SPI Slave Data Count Register

Name	Bit	Type	Description	Reset
-	31:26	-	Reserved	-
WCnt	25:16	RO	Slave transmitted data count	0
-	15:10	-	Reserved	-
RCnt	9:0	RO	Slave received data count	0

Configuration 寄存器（0x7C）

Configuration 寄存器定义如表 10-21 所示。

表 10-21 Configuration Register

Name	Bit	Type	Description	Reset
-	31:15	-	Reserved	-
Slave	14	RO	Support for SPI Slave mode	Configuration dependent
EILMMem	13	RO	Support for memory-mapped access (read-only) through EILM bus	Configuration dependent
AHBMem	12	RO	Support for memory-mapped access (read-only) through AHB bus	Configuration dependent
DirectIO	11	RO	Support for Direct SPI IO	Configuration dependent
-	10	-	Reserved	-
QuadSPI	9	RO	Support for Quad I/O SPI	Configuration dependent
DualSPI	8	RO	Support for Dual I/O SPI	Configuration dependent
TxFIFOSize	7:4	RO	Depth of TX FIFO 0: 2 words 1: 4 words 2: 8 words 3: 16 words 4: 32 words 5: 64 words 6: 128 words	Configuration dependent
RxFIFOSize	3:0	RO	Depth of RX FIFO 0: 2 words 1: 4 words 2: 8 words 3: 16 words 4: 32 words 5: 64 words 6: 128 words	Configuration dependent

10.3 驱动函数定义

10.3.1 驱动函数定义

SPI 驱动函数定义如表 10-22 所示。SPI 驱动函数定义位于 bsp\driver\ae350\spi_ae350.c、spi_ae350.h 和 bsp\driver\include\Driver_SPI.h。

表 10-22 驱动函数定义

驱动函数	描述
GetVersion	获取 SPI 驱动的版本信息
GetCapabilities	获取 SPI 驱动的功能信息
Initialize	初始化 SPI 接口
Uninitialize	卸载 SPI 接口
PowerControl	指定 SPI 接口的功耗模式
Send	通过 SPI 驱动发送器发送数据
Receive	从 SPI 驱动接收器接收数据
Transfer	通过 SPI 接口传输数据
GetDataCount	获取 SPI 接口传输数据的数量
Control	配置 SPI 接口的设置，执行指定的操作
GetStatus	获取 SPI 接口的状态

10.3.2 驱动函数描述

以下各节详细描述 SPI 的驱动函数定义。

GetVersion

GetVersion 函数定义如表 10-23 所示。

表 10-23 GetVersion 函数定义

原型	AE350_DRIVER_VERSION (*GetVersion)(void)
描述	获取 SPI 驱动的版本信息
参数	无
返回值	SPI 驱动实现的版本信息

GetCapabilities

GetCapabilities 函数定义表 10-24 所示。

表 10-24 GetCapabilities 函数定义

原型	AE350_SPI_CAPABILITIES (*GetCapabilities) (void)
描述	获取 SPI 驱动的功能信息
参数	无
返回值	SPI 驱动的功能信息

Initialize

Initialize 函数定义如表 10-25 所示。

表 10-25 Initialze 函数定义

原型	int32_t (*Initialize)(AE350_SPI_SignalEvent cb_event)
描述	初始化 SPI 接口
参数	cb_event：指向 AE350_SPI_SignalEvent 回调函数的指针
返回值	如果发生执行错误，返回一个负值

Uninitialize

Uninitialize 函数定义如表 10-26 所示。

表 10-26 Uninitialize 函数定义

原型	int32_t (*Uninitialize)(void)
描述	卸载 SPI 接口
参数	无
返回值	如果发生执行错误，返回一个负值

PowerControl

PowerControl 函数定义如表 10-27 所示。

表 10-27 PowerControl 函数定义

原型	int32_t (*PowerControl)(AE350_POWER_STATE state)
描述	指定 SPI 接口的功耗模式
参数	state：SPI 接口功耗模式，包括： AE350_POWER_FULL: Set up peripherals for data transfers, enable interrupts and DMA AE350_POWER_LOW: Enable power-saving AE350_POWER_OFF: Terminate pending data transfers and disable peripherals, related interrupts and DMA

返回值	如果发生执行错误，返回一个负值

Send

Send 函数定义如表 10-28 所示。

表 10-28 Send 函数定义

原型	int32_t (*Send)(const void *data, uint32_t num)
描述	通过 SPI 驱动发送器发送数据
参数	data：指向发送数据缓存区的指针 num：发送数据的长度
返回值	如果发生执行错误，返回一个负值

Receive

Receive 函数定义如表 10-29 所示。

表 10-29 Receive 函数定义

原型	int32_t (*Receive)(void *data, uint32_t num)
描述	从 SPI 驱动接收器接收数据
参数	data：指向接收数据缓存区的指针 num：接收数据的长度
返回值	如果发生执行错误，返回一个负值

Transfer

Transfer 函数定义如表 10-30 所示。

表 10-30 Transfer 函数定义

原型	int32_t (*Transfer)(const void *data_out, void * data_in, uint32_t num)
描述	通过 SPI 接口传输数据
参数	data_out: 指向发送数据缓存区的指针 data_in: 指向接收数据缓存区的指针 num: 传输数据的长度
返回值	如果发生执行错误，返回一个负值

GetDataCount

GetDataCount 函数定义如表 10-31 所示。

表 10-31 GetDataCount 函数定义

原型	uint32_t (*GetDataCount)(void)
描述	获取 SPI 接口传输数据的数量
参数	无
返回值	SPI 最后一次执行数据传输时，传输数据的数量

Control

Control 函数定义如表 10-32 所示。

表 10-32 Control 函数定义

原型	int32_t (*Control)(uint32_t control, uint32_t arg)
描述	配置 SPI 接口的设置，执行指定的操作
参数	control：SPI 驱动接口的一种设置或执行的一种操作 arg：指定设置或操作的附加信息
返回值	如果发生执行错误，返回一个负值

“control”和“arg”设置与操作，如表 10-33 所示。

表 10-33 Control Settings or Operations

Options for Control	arg Specifies	Settings or Operations
Mode controls (Bits: 0~7)		
AE350_SPI_MODE_INACTIVE	-	Sets the SPI to inactive
AE350_SPI_MODE_MASTER	Bus speed in bps	Sets the SPI to the master (output on MOSI, and input on MISO)
AE350_SPI_MODE_MASTER_SIMPLEX	Bus speed in bps	Sets the SPI to the master (output and input on MOSI)
AE350_SPI_MODE_SLAVE	-	Sets the SPI to the slave (output on MISO, and input on MOSI)
AE350_SPI_MODE_SLAVE_SIMPLEX	-	Sets the SPI to the slave (output and input on

Options for Control	arg Specifies	Settings or Operations
		MISO)
Clock polarity (Frame format) (Bits: 8~11)		
AE350_SPI_CPOL0_CPHA0 (default)	-	Sets the clock polarity to 0 and clock phase to 0
AE350_SPI_CPOL0_CPHA1	-	Sets the clock polarity to 0 and clock phase to 1
AE350_SPI_CPOL1_CPHA0	-	Sets the clock polarity to 1 and clock phase to 0
AE350_SPI_CPOL1_CPHA1	-	Sets the clock polarity to 1 and clock phase to 1
AE350_SPI_TI_SSI	-	Uses the Texas Instruments Frame Format
AE350_SPI_MICROWIRE	-	Uses the National Microwire Frame Format
Parity bit (Bits: 12~17)		
AE350_SPI_DATA_BITS(N)	-	Sets the number of bits per SPI frame; N ranges from 1 to 32. This is the minimum required parameter.
Bit order (Bits: 18)		
AE350_SPI_MSB_LSB (default)	-	Sets the bit order from MSB to LSB
AE350_SPI_LSB_MSB	-	Sets the bit order from LSB to MSB
Slave select mode (Bits: 19~21)		
AE350_SPI_SS_MASTER_UNUSED (default)	-	Sets the Slave Select mode for the master to “Not used”. It is specified along with the option AE350_SPI_MODE_MASTER. The master does not drive or monitor the SS line.
AE350_SPI_SS_MASTER_SW	-	Sets the Slave Select mode for the master to “Software controlled”. It is

Options for Control	arg Specifies	Settings or Operations
		specified along with the option AE350_SPI_MODE_MASTER. The Slave Select line is configured as output and controlled via AE350_SPI_CONTROL_SS. By default, the Slave Select line is not active (i.e., high), and is not affected by transfer, send, or receive functions.
AE350_SPI_SS_MASTER_HW_OUTPUT	-	Sets the Slave Select mode for the master to “Hardware controlled output”. It is specified along with the option AE350_SPI_MODE_MASTER. The Slave Select line is configured as output and controlled by the hardware. The transfers via the line is activated or deactivated by the hardware and is not affected by AE350_SPI_CONTROL_SS.
AE350_SPI_SS_MASTER_HW_INPUT	-	Sets the Slave Select mode for the master to “Hardware monitored input”. It is specified along with the option AE350_SPI_MODE_MASTER and used in multi-master configuration where a master monitors the Slave Select but not drives it. The Slave Select is configured as input. When another master

Options for Control	arg Specifies	Settings or Operations
		activates this line, the previous active master backs off. This causes a Mode Fault AE350_SPI_EVENT_MODE_FAULT and makes the SPI switch to be inactive.
AE350_SPI_SS_SLAVE_HW (default)	-	Sets the Slave Select mode for the slave to “Hardware monitored”. It is specified along with the option AE350_SPI_MODE_SLAVE. The hardware monitors the Slave Select line and accepts transfers only when the line is active. Transfers are ignored while the Slave Select line is inactive.
AE350_SPI_SS_SLAVE_SW	-	Sets the Slave Select mode for the slave to “Software controlled”. It is specified along with the option AE350_SPI_MODE_SLAVE when the Slave Select line is not used. For example, when a single master and slave are connected in the system, the Slave Select line will not be needed. The software controls whether the slave respond or not (not respond by default) and enables/disables transfers by AE350_SPI_CONTROL_SS.

Options for Control	arg Specifies	Settings or Operations
Other controls (Bits: 0~21)		
AE350_SPI_SET_BUS_SPEED	Bus speed in bps	Sets the bus speed
AE350_SPI_GET_BUS_SPEED	-	Gets the bus speed
AE350_SPI_SET_DEFAULT_TX_VALUE	Transmission value	Sets the default transmission value
AE350_SPI_CONTROL_SS	available values are AE350_SPI_SS_INACTIVE, AE350_SPI_SS_ACTIVE	Controls the Slave Select (SS) signal
AE350_SPI_ABORT_TRANSFER	-	Aborts the current data transfer

GetStatus

GetStatus 函数定义如表 10-34 所示。

表 10-34 GetStatus 函数定义

原型	AE350_SPI_STATUS (*GetStatus)(void)
描述	获取 SPI 接口的状态
参数	无
返回值	SPI 接口的当前状态

11 RTC

11.1 简介

Gowin RiscV_AE350_SOC 包含一 个低功耗的 RTC（Real Time Clock），来保存当前时间信息，并提供周期性和闹钟中断。

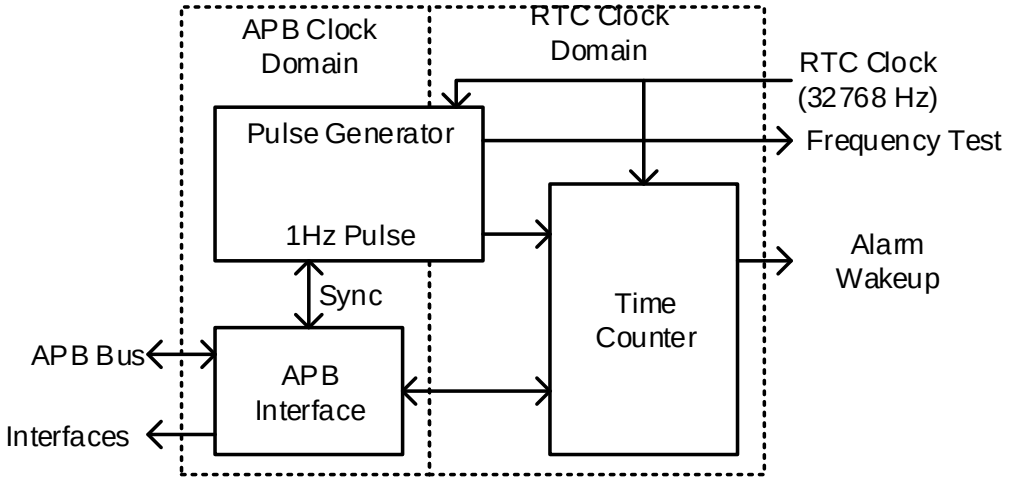
11.1.1 特征

- 通过 AMBA 2.0 APB 接口访问寄存器
- 可配置计数器大小
- 周期性中断：包括半秒、秒、分钟、小时和天
- 可编程闹钟中断
- 支持硬件数字微调，以补偿外部时钟源的不精准

11.1.2 结构框图

RTC 结构框图如图 11-1 所示。

图 11-1 RTC 结构框图



11.2 寄存器定义

11.2.1 寄存器定义

RTC 寄存器定义如表 11-1 所示。RTC 寄存器定义位于 bsp\ae350\ae350.h。

表 11-1 寄存器定义

地址偏移	寄存器名称	描述
0x00	IDREV	ID and Revision Register
0x04~0x0C	–	Reserved
0x10	CNTR	Counter Register
0x14	ALARM	Alarm Register
0x18	CTRL	Control Register
0x1C	STATUS	Status Register
0x20	TRIM	Digital Trimming Register

11.2.2 寄存器描述

以下各节详细描述 RTC 寄存器定义。

寄存器类型缩略语概括如下：

- RO: Read-only
- WO: Write-only
- R/W: Readable and writable
- W1C: Write 1 to clear

ID 和 Revision 寄存器（0x00）

ID 和 Revision 寄存器用于保存 ID 和 Revision 编号，初始值依赖于所用版本。ID 和 Revision 寄存器定义如表 11-2 所示。

表 11-2 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:8	RO	ID number for RTC	0x030110
Major	7:4	RO	Major revision number	Revision Dependent
Minor	3:0	RO	Minor revision number	Revision Dependent

Counter 寄存器（0x10）

Counter 寄存器记录当前时间。Counter 寄存器定义如表 11-3 所示。

表 11-3 Counter Register

Name	Bit	Type	Description	Reset
Day	31:17	R/W	Day passed after RTC enabled	0x0
Hour	16:12	R/W	Hour field of current time; range: 0~23	0x0
Min	11:6	R/W	Minute field of current time; range: 0~23	0x0
Sec	5:0	R/W	Second field of current time; range: 0~59	0x0

Alarm 寄存器（0x14）

当 Counter 寄存器的值与 Alarm 寄存器的值匹配时，将触发闹钟中断或闹钟唤醒信号。

Control 寄存器的“Alarm_Int”和“Alarm_Wakeup”位控制闹钟中断和闹钟唤醒信号。

Alarm 寄存器定义如表 11-4 所示。

表 11-4 Alarm Register

Name	Bit	Type	Description	Reset
Hour	16:12	R/W	Hour field of alarm setting; range: 0~23	0x0
Min	11:6	R/W	Minute of field of alarm settings; range: 0~59	0x0
Sec	5:0	R/W	Second field of alarm settings; range: 0~59	0x0

Control 寄存器（0x18）

Control 寄存器控制 RTC 的开启和中断。Control 寄存器定义如表 11-5 所示。

表 11-5 Control Register

Name	Bit	Type	Description	Reset
Freq_Test_En	8	R/W	Enable the 512 Hz frequency test output	0x0
Hsec	7	R/W	Enable half-second interrupt; half-second interrupt is generated when half a second passed	0x0
Sec	6	R/W	Enable second interrupt; second interrupt is generated when one second passed	0x0
Min	5	R/W	Enable minute interrupt; minute interrupt is generated when the seconds of RTC time changes from 59 to 0	0x0
Hour	4	R/W	Enable hour interrupt; hour interrupt is generated when the minutes of RTC time changes from 59 to 0	0x0
Day	3	R/W	Enable day interrupt; day interrupt is generated when the hours of RTC time changes from 23 to 0	0x0
Alarm_Int	2	R/W	Enable alarm interrupt	0x0
Alarm_Wakeup	1	R/W	Enable alarm wakeup signal	0x0
RTC_En	0	R/W	Enable RTC	0x0

Status 寄存器（0x1C）

Status 寄存器记录中断状态和 RTC 寄存器的更新同步状态。Status 寄存器定义如表 11-6 所示。

表 11-6 Interrupt Status Register

Name	Bit	Type	Description	Reset
WriteDone	16	RO	This bit indicates the synchronization progress of RTC register updates. This bit becomes zero when any of RTC control registers (the Counter, Alarm and Control registers) are updated. It returns to one when all prior updates to these three registers have been successfully synchronized to the RTC clock domain. While an RTC register update is being synchronized to the RTC clock domain, a second update to the same register may be dropped.	1

Name	Bit	Type	Description	Reset
			Each of the RTC registers is synchronized independently while their synchronization status is lumped into this single bit. Thus writes to different RTC registers can be done in a batch before checking this bit. Since the frequency of the RTC clock is quite slow when compared to the typical frequency of the APB clock, the synchronization period can be pretty long. The APB clock domain should not be shut down while the synchronization is still in progress.	
-	15:8	-	Reserved	-
Hsec	7	W1C	Half-second interrupt status; write 1 to clear	0x0
Sec	6	W1C	Second interrupt status	0x0
Min	5	W1C	Minute interrupt status	0x0
Hour	4	W1C	Hour interrupt status	0x0
Day	3	W1C	Day interrupt status	0x0
Alarm_Int	2	W1C	Alarm interrupt status	0x0

Digital Trimming 寄存器（0x20）

Digital Trimming 寄存器保存每个 RTC 时间周期的微调系数和方向。当关闭 RTC 后，该寄存器可以继续编程。Digital Trimming 寄存器定义如表 11-7 所示。

表 11-7 Digital Trimming Register

Name	Bit	Type	Description	Reset
Day_Sign	31	R/W	Sign bit for the trimming value for the second on the day boundary: 1: Slow down the timer. 0: Speed up the timer.	0x0
-	30:29	-	Reserved	-
Day_Trim	28:24	R/W	Digital trimming value for the second on the day boundary.	0x0
Hour_Sign	23	R/W	Sign bit for the trimming value for the second on the hour boundary: 1: Slow down the timer.	0x0

Name	Bit	Type	Description	Reset
			0: Speed up the timer.	
-	22:21	-	Reserved	-
Hour_Trim	20:16	R/W	Digital trimming value for the second on the hour boundary.	0x0
Min_Sign	15	R/W	Sign bit for the trimming value for the second on the minute boundary: 1: Slow down the timer. 0: Speed up the timer.	0x0
-	14:13	-	Reserved	-
Min_Trim	12:8	R/W	Digital trimming value for the second on the minute boundary.	0x0
Sec_Sign	7	R/W	Sign bit for the trimming value for the rest of seconds: 1: Slow down the timer. 0: Speed up the timer.	0x0
-	6:5	-	Reserved	-
Sec_Trim	4:0	R/W	Digital trimming value for the rest of seconds	0x0

11.3 驱动函数定义

11.3.1 驱动函数定义

RTC 驱动函数定义如表 11-8 所示。RTC 驱动函数定义位于 bsp\driver\ae350\rtc_ae350.c、rtc_ae350.h 和 bsp\driver\include\Driver_RTC.h。

表 11-8 驱动函数定义

驱动函数	描述
GetVersion	获取 RTC 驱动的版本信息
Initialize	初始化 RTC 接口
Uninitialize	卸载 RTC 接口
PowerControl	指定 RTC 接口的功耗模式
SetTime	调整当前的硬件时钟计数器
GetTime	从当前的硬件时钟计数器获取时间
SetAlarm	调整当前的硬件闹钟计数器
GetAlarm	从当前的硬件闹钟计数器查询闹钟时间

驱动函数	描述
Control	配置 RTC 接口的设置，执行指定的操作
GetStatus	获取 RTC 接口的状态

11.3.2 驱动函数描述

以下各节详细描述 RTC 的驱动函数定义。

GetVersion

GetVersion 函数定义如表 11-9 所示。

表 11-9 GetVersion 函数定义

原型	AE350_DRIVER_VERSION (*GetVersion)(void)
描述	获取 RTC 驱动的版本信息
参数	无
返回值	RTC 驱动实现的版本信息

Initialize

Initialize 函数定义如表 11-10 所示。

表 11-10 Initialze 函数定义

原型	int32_t (*Initialize)(AE350_RTC_SignalEvent cb_event)
描述	初始化 RTC 接口
参数	cb_event：指向 AE350_RTC_SignalEvent 回调函数的指针
返回值	如果发生执行错误，返回一个负值

Uninitialize

Uninitialize 函数定义如表 11-11 所示。

表 11-11 Uninitialize 函数定义

原型	int32_t (*Uninitialize)(void)
描述	卸载 RTC 接口
参数	无
返回值	如果发生执行错误，返回一个负值

PowerControl

PowerControl 函数定义如表 11-12 所示。

表 11-12 PowerControl 函数定义

原型	int32_t (*PowerControl)(AE350_POWER_STATE state)
描述	指定 RTC 接口的功耗模式
参数	state：RTC 接口功耗模式，包括： AE350_POWER_FULL: Set up peripherals for data transfers, enable interrupts and DMA AE350_POWER_LOW: Enable power-saving AE350_POWER_OFF: Terminate pending data transfers and disable peripherals, related interrupts and DMA
返回值	如果发生执行错误，返回一个负值

SetTime

SetTime 函数定义如表 11-13 所示。

表 11-13 SetTime 函数定义

原型	int32_t (*SetTime)(AE350_RTC_TIME* stime)
描述	调整当前的硬件时钟计数器
参数	stime：为当前硬件时钟计数器设置的时间
返回值	如果发生执行错误，返回一个负值

GetTime

GetTime 函数定义如表 11-14 所示。

表 11-14 GetTime 函数定义

原型	int32_t (*GetTime)(AE350_RTC_TIME* stime)
描述	从当前的硬件时钟计数器获取时间
参数	stime：从当前硬件时钟计数器查询的时间
返回值	如果发生执行错误，返回一个负值

SetAlarm

SetAlarm 函数定义如表 11-15 所示。

表 11-15 SetAlarm 函数定义

原型	int32_t (*SetAlarm)(AE350_RTC_ALARM* salarm)
描述	调整当前的硬件闹钟计数器
参数	alarm：为当前硬件闹钟计数器设置的闹钟时间

返回值	如果发生执行错误，返回一个负值

GetAlarm

GetAlarm 函数定义如表 11-16 所示。

表 11-16 GetAlarm 函数定义

原型	int32_t (*GetAlarm)(AE350_RTC_ALARM* salarm)
描述	从当前的硬件闹钟计数器查询闹钟时间
参数	alarm：从当前硬件闹钟计数器查询的闹钟时间
返回值	如果发生执行错误，返回一个负值

Control

Control 函数定义如表 11-17 所示。

表 11-17 Control 函数定义

原型	int32_t (*Control)(uint32_t control, uint32_t arg)
描述	配置 RTC 接口的设置，执行指定的操作
参数	Control: RTC 驱动接口的一种设置或执行的一种操作 arg: 指定设置或操作的附加信息
返回值	如果发生执行错误，返回一个负值

“control”和“arg”设置与操作如表 11-18 所示。

表 11-18 Control Settings or Operations

Options for Control	arg Specifies	Settings or Operations
(Bits: 0 ~ 7)		
AE350_RTC_CTRL_EN	On/Off	Enables or disables RTC
AE350_RTC_CTRL_ALARM_WAKEUP		Enables or disables the alarm wakeup signal
AE350_RTC_CTRL_ALARM_INT		Enables or disables the alarm interrupt
AE350_RTC_CTRL_DAY_INT		Enables or disables the day interrupt
AE350_RTC_CTRL_HOUR_INT		Enables or disables the hour interrupt

Options for Control	arg Specifies	Settings or Operations
AE350_RTC_CTRL_MIN_INT		Enables or disables the minute interrupt
AE350_RTC_CTRL_SEC_INT		Enables or disables the second interrupt
AE350_RTC_CTRL_HSEC_INT		Enables or disables the half second interrupt

GetStatus

GetStatus 函数定义如表 11-19 所示。

表 11-19 GetStatus 函数定义

原型	AE350_RTC_STATUS (*GetStatus)(void)
描述	获取 RTC 接口的状态
参数	无
返回值	RTC 接口的当前状态

12 PIT

12.1 简介

Gowin RiscV_AE350_SOC 包含一个 PIT（Programmable Interval Timer），PIT 是一个小型的多功能定时器，可以用作 PWM（Pulse Width Modulation）和简单的定时器。

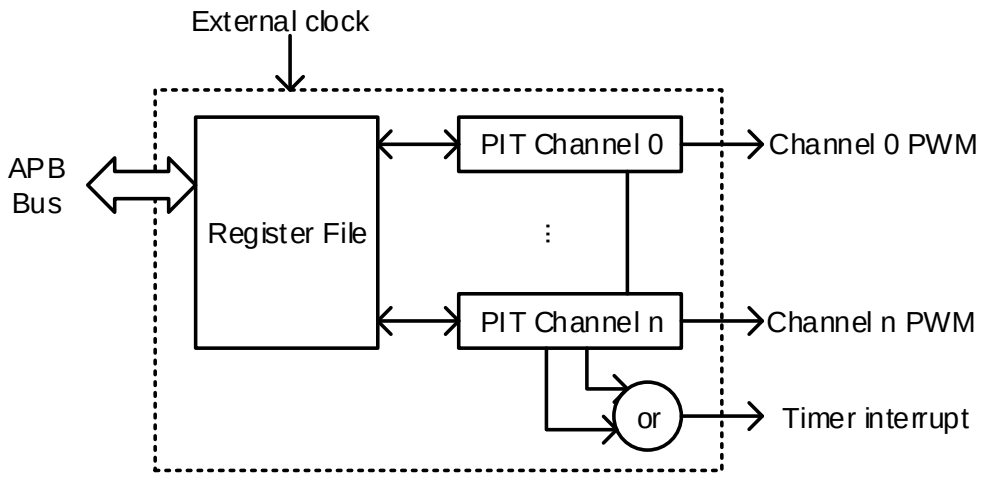
12.1.1 特征

- 支持 AMBA 2.0 APB 总线协议规范
- 支持多达 4 个多功能定时器
- 每个多功能定时器可以提供 6 种应用场景（定时器和 PWM 的组合）
- 可编程的定时器时钟源
- 定时器可以外部控制暂停

12.1.2 结构框图

PIT 结构框图如图 12-1 所示。

图 12-1 PIT 结构框图



12.1.3 功能描述

PIT 支持多达 4 个 PIT 通道，每个 PIT 通道都是一个多功能的定时器，可以提供 6 种应用场景：

- 1 个 32 位定时器
- 2 个 16 位定时器
- 4 个 8 位定时器
- 1 个 16 位 PWM
- 1 个 16 位定时器和 1 个 8 位 PWM
- 2 个 8 位定时器和 1 个 8 位 PWM

PIT 通道模式如表 12-1 所示。

表 12-1 Effective Devices of Channel Modes

Channel Mode	32-bit Timer	16-bit Timers	8-bit Timers	PWM	Mixed PWM/16-bit Timer	Mixed PWM/8-bit Timers
–	32-bit Timer 0	16-bit Timer 0	8-bit Timer 0	–	16-bit Timer 0	8-bit Timer 0
–	–	16-bit Timer 1	8-bit Timer 1	–	–	8-bit Timer 1
–	–	–	8-bit Timer 2	–	–	–
–	–	–	8-bit Timer 3	–	–	–
–	–	–		16-bit PWM	8-bit PWM	8-bit PWM

12.2 寄存器定义

12.2.1 寄存器定义

PIT 寄存器定义如表 12-2 所示。PIT 寄存器定义位于 bsp\ae350\ae350.h。

表 12-2 寄存器定义

地址偏移	寄存器名称	描述
0x00	IDREV	ID and Revision Register
0x04~0x0C	-	Reserved
0x10	CFG	Configuration Register
0x14	INTEN	Interrupt Enable Register
0x18	INTST	Interrupt Status Register
0x1C	CHNEN	Channel Enable Register
0x20	Ch0Ctrl	Channel 0 Control Register
0x24	Ch0Reload	Channel 0 Reload Register
0x28	Ch0Cntr	Channel 0 Counter Register
0x2C	-	Reserved
0x30	Ch1Ctrl	Channel 1 Control Register
0x34	Ch1Reload	Channel 1 Reload Register
0x38	Ch1Cntr	Channel 1 Counter Register
0x3C	-	Reserved
0x40	Ch2Ctrl	Channel 2 Control Register
0x44	Ch2Reload	Channel 2 Reload Register
0x48	Ch2Cntr	Channel 2 Counter Register
0x4C	-	Reserved
0x50	Ch3Ctrl	Channel 3 Control Register
0x54	Ch3Reload	Channel 3 Reload Register
0x58	Ch3Cntr	Channel 3 Counter Register
0x5C	-	Reserved

12.2.2 寄存器描述

以下各节详细描述 PIT 寄存器定义。

寄存器类型缩略语概括如下：

- RO: Read-only
- R/W: Readable and writable
- W1C: Write 1 to clear

ID 和 Revision 寄存器（0x00）

ID 和 Revision 寄存器，用于保存 ID 和 Revision 编号，初始值依赖于所用版本。ID 和 Revision 寄存器定义如表 12-3 所示。

表 12-3 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:12	RO	ID number for PIT	0x03031
RevMajor	11:4	RO	Major revision number	Revision Dependent
RevMinor	3:0	RO	Minor revision number	Revision Dependent

Configuration 寄存器（0x10）

Configuration 寄存器定义如表 12-4 所示。

表 12-4 Configuration Register

Name	Bit	Type	Description	Reset
–	31:3	–	Reserved	–
NumCh	2:0	RO	Number of PIT channels	Configuration dependent

Interrupt Enable 寄存器（0x14）

Interrupt Enable 寄存器定义如表 12-5 所示。

表 12-5 Interrupt Enable Register

Name	Bit	Type	Description	Reset
–	31:16	–	Reserved	–
Ch3Int3En	15	R/W	Channel 3 Timer 3 interrupt enable	0x0
Ch3Int2En	14	R/W	Channel 3 Timer 2 interrupt enable	0x0
Ch3Int1En	13	R/W	Channel 3 Timer 1 interrupt enable	0x0
Ch3Int0En	12	R/W	Channel 3 Timer 0 interrupt enable	0x0

Name	Bit	Type	Description	Reset
Ch2Int3En	11	R/W	Channel 2 Timer 3 interrupt enable	0x0
Ch2Int2En	10	R/W	Channel 2 Timer 2 interrupt enable	0x0
Ch2Int1En	9	R/W	Channel 2 Timer 1 interrupt enable	0x0
Ch2Int0En	8	R/W	Channel 2 Timer 0 interrupt enable	0x0
Ch1Int3En	7	R/W	Channel 1 Timer 3 interrupt enable	0x0
Ch1Int2En	6	R/W	Channel 1 Timer 2 interrupt enable	0x0
Ch1Int1En	5	R/W	Channel 1 Timer 1 interrupt enable	0x0
Ch1Int0En	4	R/W	Channel 1 Timer 0 interrupt enable	0x0
Ch0Int3En	3	R/W	Channel 0 Timer 3 interrupt enable	0x0
Ch0Int2En	2	R/W	Channel 0 Timer 2 interrupt enable	0x0
Ch0Int1En	1	R/W	Channel 0 Timer 1 interrupt enable	0x0
Ch0Int0En	0	R/W	Channel 0 Timer 0 interrupt enable 0: Disable 1: Enable	0x0

Interrupt Status 寄存器（0x18）

Interrupt Status 寄存器定义如表 12-6 所示。

表 12-6 Interrupt Status Register

Name	Bit	Type	Description	Reset
-	31:16	-	Reserved	-
Ch3Int3	15	W1C	Channel 3 Timer 3 interrupt status	0x0
Ch3Int2	14	W1C	Channel 3 Timer 2 interrupt status	0x0
Ch3Int1	13	W1C	Channel 3 Timer 1 interrupt status	0x0
Ch3Int0	12	W1C	Channel 3 Timer 0 interrupt status	0x0
Ch2Int3	11	W1C	Channel 2 Timer 3 interrupt status	0x0
Ch2Int2	10	W1C	Channel 2 Timer 2 interrupt status	0x0
Ch2Int1	9	W1C	Channel 2 Timer 1 interrupt status	0x0
Ch2Int0	8	W1C	Channel 2 Timer 0 interrupt status	0x0
Ch1Int3	7	W1C	Channel 1 Timer 3 interrupt status	0x0
Ch1Int2	6	W1C	Channel 1 Timer 2 interrupt status	0x0
Ch1Int1	5	W1C	Channel 1 Timer 1 interrupt status	0x0
Ch1Int0	4	W1C	Channel 1 Timer 0 interrupt status	0x0

Name	Bit	Type	Description	Reset
Ch0Int3	3	W1C	Channel 0 Timer 3 interrupt status 0: No effect 1: Timer 3 time up	0x0
Ch0Int2	2	W1C	Channel 0 Timer 2 interrupt status 0: No effect 1: Timer 2 time up	0x0
Ch0Int1	1	W1C	Channel 0 Timer 1 interrupt status 0: No effect 1: Timer 1 time up	0x0
Ch0Int0	0	W1C	Channel 0 Timer 0 interrupt status 0: No effect 1: Timer 0 time up	0x0

Channel Enable 寄存器（0x1C）

Channel Enable 寄存器定义如表 12-7 所示。

表 12-7 Channel Enable Register

Name	Bit	Type	Description	Reset
-	31:16	-	Reserved	-
Ch3TMR3En/Ch3PWMEn	15	R/W	ChMode = 1, 2, 3 Channel 3 Timer 3 enable ChMode = 4, 6, 7 Channel 3 PWM enable	0x0
Ch3TMR2En	14	R/W	Channel 3 Timer 2 enable	0x0
Ch3TMR1En	13	R/W	Channel 3 Timer 1 enable	0x0
Ch3TMR0En	12	R/W	Channel 3 Timer 0 enable	0x0
Ch2TMR3En/Ch2PWMEn	11	R/W	ChMode = 1, 2, 3 Channel 2 Timer 3 enable ChMode = 4, 6, 7 Channel 2 PWM enable	0x0
Ch2TMR2En	10	R/W	Channel 2 Timer 2 enable	0x0
Ch2TMR1En	9	R/W	Channel 2 Timer 1 enable	0x0
Ch2TMR0En	8	R/W	Channel 2 Timer 0 enable	0x0
Ch1TMR3En/Ch1PWMEn	7	R/W	ChMode = 1, 2, 3 Channel 1 Timer 3 enable	0x0

Name	Bit	Type	Description	Reset
			ChMode = 4, 6, 7 Channel 1 PWM enable	
Ch1TMR2En	6	R/W	Channel 1 Timer 2 enable	0x0
Ch1TMR1En	5	R/W	Channel 1 Timer 1 enable	0x0
Ch1TMR0En	4	R/W	Channel 1 Timer 0 enable	0x0
Ch0TMR3En/Ch0PWMEn	3	R/W	ChMode = 1, 2, 3 Channel 0 Timer 3 enable ChMode = 4, 6, 7 Channel 0 PWM enable	0x0
Ch0TMR2En	2	R/W	Channel 0 Timer 2 enable	0x0
Ch0TMR1En	1	R/W	Channel 0 Timer 1 enable	0x0
Ch0TMR0En	0	R/W	Channel 0 Timer 0 enable 0: Disable 1: Enable	0x0

注！

如果对应的通道不存在或不是该通道模式下的有效设备，则定时器或 PWM 不能启用。例如，当通道 0 设置为 32 位定时器模式时，通道 0 的定时器 1 不能启用。

Channel n Control 寄存器（0x20 + n * 0x10）

Channel 0~3 Control 寄存器定义如表 12-8 所示。

表 12-8 Channel 0~3 Control Register

Name	Bit	Type	Description	Reset
-	31:5	-	Reserved	-
PWMPark	4	R/W	PWM park value. When this channel is disabled, this bit reflects the output of PWM and writing to it will change the output of PWM. The value of this bit also governs how the PWM waveform is generated when this channel is enabled. 0: the PWM output is LOW when the channel is disabled; the low-period PWM counter will be counted first before toggling the output to HIGH and counting the high-period PWM counter when this channel is enabled. 1: the PWM output is HIGH when the channel is disabled; the high-period PWM counter will be counted first before toggling the output to LOW	0x0

Name	Bit	Type	Description	Reset
			and counting the low-period PWM counter when this channel is enabled.	
ChClk	3	R/W	Channel clock source: 0: External clock 1: APB clock	0x0
ChMode	2:0	R/W	Channel mode: 0: Reserved 1: 32-bit timer 2: 16-bit timers 3: 8-bit timers 4: PWM 5: Reserved 6: Mixed PWM/16-bit timer 7: Mixed PWM/8-bit timers	0x0

Channel n Reload Register (0x24 + n * 0x10)

Reload 寄存器，用于保存 PWM/定时器的计数器初始/重载的值，依赖于通道模式。PWM/定时器的周期等于重载的值加 1，例如，32 位定时器模式，每（TMR32_0 + 1）周期产生一次定时中断。PWM 模式，最高周期为（PWM16_Hi + 1）周期，最低周期为（PWM16_Lo + 1）周期。

表 12-9、表 12-10、表 12-11、表 12-12、表 12-13、表 12-14 描述了 Reload 寄存器的定义。

表 12-9 Reload Register for 32-bit Timer Mode (ChMode = 1)

Name	Bit	Type	Description	Reset
TMR32_0	31:0	R/W	Reload value for 32-bit Timer 0	0x0

表 12-10 Reload Register for 16-bit Timers Mode (ChMode = 2)

Name	Bit	Type	Description	Reset
TMR16_1	31:16	R/W	Reload value for 16-bit Timer 1	0x0
TMR16_0	15:0	R/W	Reload value for 16-bit Timer 0	0x0

表 12-11 Reload Register for 8-bit Timers Mode (ChMode = 3)

Name	Bit	Type	Description	Reset
TMR8_3	31:24	R/W	Reload value for 8-bit Timer 3	0x0
TMR8_2	23:16	R/W	Reload value for 8-bit Timer 2	0x0
TMR8_1	15:8	R/W	Reload value for 8-bit Timer 1	0x0
TMR8_0	7:0	R/W	Reload value for 8-bit Timer 0	0x0

表 12-12 Reload Register for PWM Mode (ChMode = 4)

Name	Bit	Type	Description	Reset
PWM16_Hi	31:16	R/W	Reload value for PWM high period	0x0
PWM16_Lo	15:0	R/W	Reload value for PWM low period	0x0

表 12-13 Reload Register for Mixed PWM/16-bit Timer Mode (ChMode = 6)

Name	Bit	Type	Description	Reset
PWM8_Hi	31:24	R/W	Reload value for PWM high period	0x0
PWM8_Lo	23:16	R/W	Reload value for PWM low period	0x0
TMR16_0	15:0	R/W	Reload value for 16-bit Timer 0	0x0

表 12-14 Reload Register for Mixed PWM/8-bit Timers Mode (ChMode = 7)

Name	Bit	Type	Description	Reset
PWM8_Hi	31:24	R/W	Reload value for PWM high period	0x0
PWM8_Lo	23:16	R/W	Reload value for PWM low period	0x0
TMR8_1	15:8	R/W	Reload value for 8-bit Timer 1	0x0
TMR8_0	7:0	R/W	Reload value for 8-bit Timer 0	0x0

Channel n Counter 寄存器（0x28 + n * 0x10）

Channel n Counter 寄存器，指示下一个定时器中断或 PWM 翻转的剩余周期。与 Reload 寄存器类似，Counter 寄存器的位定义也是根据通道模式而不同。

Channel n Counter 寄存器定义如表 12-15 所示。

表 12-15 Counter Register

Name	Bit	Type	Description	Reset
Counter	31:0	RO	The counter for the counting of Timer/PWM Refer to Table 9 to Table -14 for the field definitions with respect to the channel mode.	0x0

12.3 驱动函数定义（PWM）

12.3.1 驱动函数定义

PWM 驱动函数定义如表 12-16 所示。PWM 驱动函数定义位于 bsp\driver\ae350\pwm_ae350.c、pwm_ae350.h 和 bsp\driver\include\Driver_PWM.h。

表 12-16 驱动函数定义

驱动函数	描述
GetVersion	获取 PWM 驱动的版本信息
GetCapabilities	获取 PWM 驱动的功能信息
Initialize	初始化 PWM 接口
Uninitialize	卸载 PWM 接口
PowerControl	指定 PWM 接口的功耗模式
Control	配置 PWM 接口的设置，执行指定的操作
SetFreq	设置 PWM 频率
Output	根据设定的占空比开启 PWM 输出
GetStatus	获取 PWM 接口的状态

12.3.2 驱动函数描述

以下各节详细描述 PWM 的驱动函数定义。

GetVersion

GetVersion 函数定义如表 12-17 所示。

表 12-17 GetVersion 函数定义

原型	AE350_DRIVER_VERSION (*GetVersion)(void)
描述	获取 PWM 驱动的版本信息
参数	无
返回值	PWM 驱动实现的版本信息

GetCapabilities

GetCapabilities 函数定义表 12-18 所示。

表 12-18 GetCapabilities 函数定义

原型	AE350_PWM_CAPABILITIES (*GetCapabilities) (void)
描述	获取 PWM 驱动的功能信息
参数	无
返回值	PWM 驱动的功能信息

Initialize

Initialize 函数定义如表 12-19 所示。

表 12-19 Initialze 函数定义

原型	int32_t (*Initialize)(AE350_PWM_SignalEvent cb_event)
描述	初始化 PWM 接口
参数	cb_event：指向 AE350_PWM_SignalEvent 回调函数的指针
返回值	如果发生执行错误，返回一个负值

Uninitialize

Uninitialize 函数定义如表 12-20 所示。

表 12-20 Uninitialize 函数定义

原型	int32_t (*Uninitialize)(void)
描述	卸载 PWM 接口
参数	无
返回值	如果发生执行错误，返回一个负值

PowerControl

PowerControl 函数定义如表 12-21 所示。

表 12-21 PowerControl 函数定义

原型	int32_t (*PowerControl)(AE350_POWER_STATE state)
描述	指定 PWM 接口的功耗模式
参数	state：PWM 接口功耗模式，包括： AE350_POWER_FULL: Set up peripherals for data transfers, enable interrupts and DMA

	AE350_POWER_LOW: Enable power-saving AE350_POWER_OFF: Terminate pending data transfers and disable peripherals, related interrupts and DMA
返回值	如果发生执行错误，返回一个负值

Control

Control 函数定义如表 12-22 所示。

表 12-22 Control 函数定义

原型	int32_t (*Control)(uint32_t control, uint32_t arg)
描述	配置 PWM 接口的设置，执行指定的操作
参数	Control: PWM 驱动接口的一种设置或执行的一种操作 arg: PWM 输出通道
返回值	如果发生执行错误，返回一个负值

“control”和“arg”设置与操作如表 12-23 所示。

表 12-23 Control Settings or Operations

Options for Control	arg Specifies	Settings or Operations
AE350_PWM_ACTIVE_CONFIGURE	PWM output channel	Activates PWM
AE350_PWM_PARK_LOW		Sets the PWM park value to low
AE350_PWM_PARK_HIGH		Sets the PWM park value to high
AE350_PWM_CLKSRC_SYSTEM		Selects a system clock source
AE350_PWM_CLKSRC_EXTERNAL		Selects an external clock source

SetFreq

SetFreq 函数定义如表 12-24 所示。

表 12-24 SetFreq 函数定义

原型	int32_t (*SetFreq)(uint8_t pwm, uint32_t freq)
描述	设置 PWM 频率
参数	pwm：PWM 输出通道号

	freq：PWM 输出频率
返回值	如果发生执行错误，返回一个负值

Output

Output 函数定义如表 12-25 所示。

表 12-25 Output 函数定义

原型	int32_t (*Output)(uint8_t pwm, uint8_t duty)
描述	根据设定的占空比开启 PWM 输出
参数	pwm：PWM 输出通道号 duty：PWM 输出的占空比
返回值	如果发生执行错误，返回一个负值

GetStatus

GetStatus 函数定义如表 12-26 所示。

表 12-26 GetStatus 函数定义

原型	AE350_PWM_STATUS (*GetStatus)(void)
描述	获取 PWM 接口的状态
参数	无
返回值	PWM 接口的当前状态

12.4 驱动函数定义（PIT Timer）

12.4.1 驱动函数定义

PIT Timer 驱动函数定义如表 12-27 所示。PIT Timer 驱动函数定义位于 bsp\driver\ae350\pit_ae350.c、pit_ae350.h 和 bsp\driver\include\Driver_PIT.h。

表 12-27 驱动函数定义

驱动函数	描述
GetVersion	获取 PIT Timer 驱动的版本信息
Initialize	初始化 PIT Timer 接口
Read	读取 PIT Timer 时间戳
Control	配置 PIT Timer 接口的设置，执行指定的操作
SetPeriod	设置 PIT Timer 周期

驱动函数	描述
GetStatus	获取 PIT Timer 接口的状态
GetTick	获取 PIT Timer 的时间单位 tick

12.4.2 驱动函数描述

以下各节详细描述 PIT Timer 的驱动函数定义。

GetVersion

GetVersion 函数定义如表 12-28 所示。

表 12-28 GetVersion 函数定义

原型	AE350_DRIVER_VERSION (*GetVersion)(void)
描述	获取 PIT Timer 驱动的版本信息
参数	无
返回值	PIT Timer 驱动实现的版本信息

Initialize

Initialize 函数定义如表 12-29 所示。

表 12-29 Initialze 函数定义

原型	int32_t (*Initialize)(void)
描述	初始化 PIT Timer 接口
参数	无
返回值	如果发生执行错误，返回一个负值

Read

Read 函数定义如表 12-30 所示。

表 12-30 Read 函数定义

原型	uint32_t (*Read)(uint32_t tmr)
描述	读取 PIT Timer 时间戳
参数	tmr：PIT 通道号
返回值	PIT Timer 的时间戳

Control

Control 函数定义如表 12-31 所示。

表 12-31 Control 函数定义

原型	int32_t (*Control)(uint32_t mode, uint32_t tmr)
描述	配置 PIT Timer 接口的设置，执行指定的操作
参数	mode：PIT Timer 驱动接口的一种设置或执行的一种操作 tmr： PIT 通道号
返回值	如果发生执行错误，返回一个负值

“mode”和“tmr”设置与操作如表 12-32 所示。

表 12-32 Mode Settings or Operations

Options for Mode	tmr Specifies	Settings or Operations
AE350_PIT_TIMER_START	PIT channel	Start simple timer
AE350_PIT_TIMER_STOP		Stop simple timer
AE350_PIT_TIMER_INTR_ENABLE		Enable simple timer interrupt
AE350_PIT_TIMER_INTR_DISABLE		Disable simple timer interrupt
AE350_PIT_TIMER_INTER_CLEAR		Clear simple timer interrupt

SetPeriod

SetPeriod 函数定义如表 12-33 所示。

表 12-33 SetPeriod 函数定义

原型	int32_t (*SetPeriod)(uint32_t tmr, uint32_t period)
描述	设置 PIT Timer 周期
参数	period：PIT Timer 周期 tmr： PIT 通道号
返回值	如果发生执行错误，返回一个负值

GetStatus

GetStatus 函数定义如表 12-34 所示。

表 12-34 GetStatus 函数定义

原型	uint32_t (*GetStatus)(uint32_t tmr)
描述	获取 PIT Timer 接口的状态

参数	tmr：PIT 通道号
返回值	PIT Timer 接口的当前状态

GetTick

GetTick 函数定义如表 12-35 所示。

表 12-35 GetTick 函数定义

原型	uint32_t (*GetTick)(uint32_t mode, uint32_t sec)
描述	获取 PIT Timer 的时间单位 tick
参数	mode：PIT Timer 驱动接口设置或执行的一种操作 sec： 时间单位 tick
返回值	PIT Timer 的秒或微秒的时间单位 tick

“mode”设置与操作如表 12-36 所示。

表 12-36 Mode Settings or Operations

Options for mode	Settings or operations
AE350_PIT_TIMER_SEC_TICK	Get second time unit tick
AE350_PIT_TIMER_MSEC_TICK	Get microsecond time unit tick

13 WDT

13.1 简介

Gowin RiscV_AE350_SOC 包含一个 WDT（Watch Dog Timer），用于防止程序在执行错误时发生系统锁定。

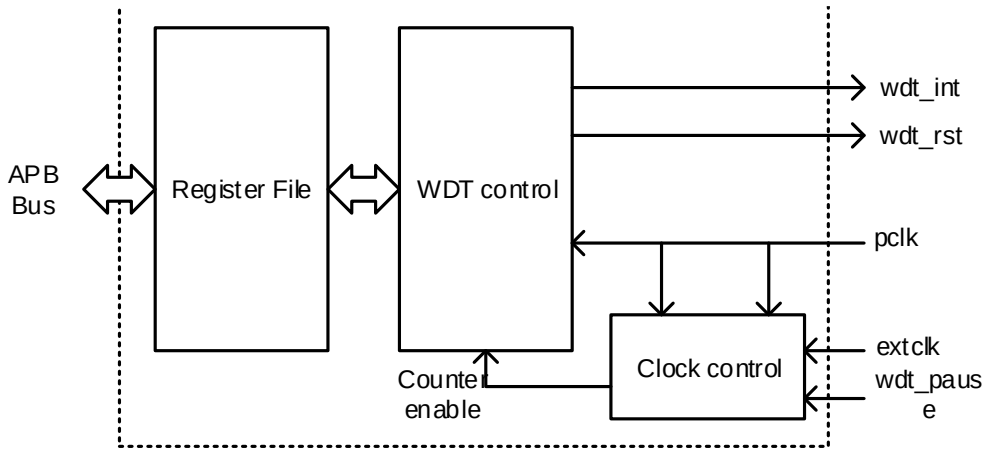
13.1.1 特征

- 支持 AMBA 2.0 APB 总线协议规范
- 当 WDT 超时，支持中断和复位机制
- 支持控制/重启寄存器写保护机制
- 可编程的定时器时钟源
- 可配置的魔术数字，用于寄存器写保护和重启定时器
- 支持外部暂停 WDT

13.1.2 结构框图

WDT 结构框图如图 13-1 所示。

图 13-1 WDT 结构框图



13.1.3 功能描述

WDT 提供一个两阶段机制，以防止系统锁定。

第一个阶段称为中断阶段。如果开启 WDT 中断，并且在中断阶段没有重启 WDT，则中断信号“wdt_int”将被断言。

第二个阶段称为复位阶段，在中断阶段之后立即开始。如果开启 WDT 复位，并且在复位阶段没有重启 WDT，则复位信号“wdt_rst”将被断言。

13.2 寄存器定义

13.2.1 寄存器定义

WDT 寄存器定义如表 13-1 所示。WDT 寄存器定义位于 bsp\ae350\ae350.h。

表 13-1 寄存器定义

地址偏移	寄存器名称	描述
0x00	IDREV	ID and Revision Register
0x04~0x0C	-	Reserved
0x10	CTRL	Control Register
0x14	RESTART	Restart Register
0x18	WREN	Write Enable Register
0x1C	ST	Status Register

13.2.2 寄存器描述

以下各节详细描述 WDT 寄存器定义。

寄存器类型缩略语概括如下：

- RO: Read-only
- WO: Write-only
- R/W1C: Readable and Write 1 to clear
- WP: Write protected
- R/WP: Readable and Write protected
- DC: Don’t care

ID 和 Revision 寄存器（0x00）

ID 和 Revision 寄存器用于保存 ID 和 Revision 编号，初始值依赖于所用版本。ID 和 Revision 寄存器定义如表 13-2 所示。

表 13-2 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:12	RO	ID number for WDT	0x03002
RevMajor	11:4	RO	Major revision number	Revision Dependent
RevMinor	3:0	RO	Minor revision number	Revision Dependent

Control 寄存器（0x10）

Control 寄存器被写保护，以防止意外重写。在写该寄存器之前，必须先写 Write Enable 寄存器。Control 寄存器定义如表 13-3 所示。

表 13-3 Control Register

Name	Bit	Type	Description	Reset
-	31:11	-	Reserved	-
RstTime	10:8	R/WP	The timer interval of the reset stage: 0: Clock period × 2^7 1: Clock period × 2^8 2: Clock period × 2^9 3: Clock period × 2^{10} 4: Clock period × 2^{11} 5: Clock period × 2^{12} 6: Clock period × 2^{13} 7: Clock period × 2^{14}	0x0

Name	Bit	Type	Description	Reset
IntTime	7:4	R/WP	The timer interval of the interrupt stage: 0: Clock period × 2^6 1: Clock period × 2^8 2: Clock period × 2^{10} 3: Clock period × 2^{11} 4: Clock period × 2^{12} 5: Clock period × 2^{13} 6: Clock period × 2^{14} 7: Clock period × 2^{15} 8: Clock period × 2^{17} 9: Clock period × 2^{19} 10: Clock period × 2^{21} 11: Clock period × 2^{23} 12: Clock period × 2^{25} 13: Clock period × 2^{27} 14: Clock period × 2^{29} 15: Clock period × 2^{31}	0x0
RstEn	3	R/WP	Enable or disable the watchdog reset 0: Disable 1: Enable	0x0
IntEn	2	R/WP	Enable or disable the watchdog interrupt 0: Disable 1: Enable	0x0
ClkSel	1	R/WP	Clock source of timer 0: EXTCLK 1: PCLK	0x0
En	0	R/WP	Enable or disable the watchdog timer 0: Disable 1: Enable	0x0

Restart 寄存器（0x14）

Restart 寄存器被写保护，以防止意外覆盖。在写该寄存器之前，必须先写 Write Enable 寄存器。

如果 Restart 寄存器被写入预配置的值“WDT_RESTART_NUM”，则重启中断定时器，以及终止系统复位定时器。如果预配置的值不是

“WDT_RESTART_NUM”，则 WDT 忽略该写操作。

Restart 寄存器定义如表 13-4 所示。

表 13-4 Restart Register

Name	Bit	Type	Description	Reset
-	31:16	-	Reserved	-
Restart	15:0	WP	Write the magic number “WDT_RESTART_NUM” to restart the watchdog timer.	DC

Write Enable 寄存器（0x18）

Control 寄存器和 Restart 寄存器都通过一个两步写的机制来进行编程，Write Enable 寄存器使用一个魔术数字“WDT_WREN_NUM”来编程，以便在更新这两个寄存器中的任何一个之前关闭写保护机制，在接收到任何 WDT 寄存器的后续写操作时，都会再次开启寄存器的写保护机制。

Write Enable 寄存器定义如表 13-5 所示。

表 13-5 Write Enable Register

Name	Bit	Type	Description	Reset
–	31:16	–	Reserved	–
WEn	15:0	WO	Write the magic number “WDT_WREN_NUM” to disable the write protection of the Control Register and the Restart Register.	DC

Status 寄存器（0x1C）

Status 寄存器定义如表 13-6 所示。

表 13-6 Status Register

Name	Bit	Type	Description	Reset
–	31:1	–	Reserved	–
IntExpired	0	R/W1C	The status of the watchdog interrupt timer 0: timer is not expired ye 1: timer is expired	0x0

13.3 驱动函数定义

13.3.1 驱动函数定义

WDT 驱动函数定义如表 13-7 所示。WDT 驱动函数定义位于 bsp\driver\ae350\wdt_ae350.c、wdt_ae350.h 和 bsp\driver\include\Driver_WDT.h。

表 13-7 驱动函数定义

驱动函数	描述
GetVersion	获取 WDT 驱动的版本信息
GetCapabilities	获取 WDT 驱动的功能信息
Initialize	初始化 WDT 接口
Uninitialize	卸载 WDT 接口
Control	配置 WDT 接口的设置
Enable	开启 WDT
Disable	关闭 WDT
RestartTimer	重启 WDT
ClearIrqStatus	清除 WDT 中断状态
GetStatus	获取 WDT 接口的状态

13.3.2 驱动函数描述

以下各节详细描述 WDT 的驱动函数定义。

GetVersion

GetVersion 函数定义如表 13-8 所示。

表 13-8 GetVersion 函数定义

原型	AE350_DRIVER_VERSION (*GetVersion)(void)
描述	获取 WDT 驱动的版本信息
参数	无
返回值	WDT 驱动实现的版本信息

GetCapabilities

GetCapabilities 函数定义表 13-9 所示。

表 13-9 GetCapabilities 函数定义

原型	AE350_WDT_CAPABILITIES (*GetCapabilities) (void)
描述	获取 WDT 驱动的功能信息
参数	无
返回值	WDT 驱动的功能信息

Initialize

Initialize 函数定义如表 13-10 所示。

表 13-10 Initialze 函数定义

原型	int32_t (*Initialize)(AE350_WDT_SignalEvent cb_event)
描述	初始化 WDT 接口
参数	cb_event：指向 AE350_WDT_SignalEvent 回调函数的指针
返回值	如果发生执行错误，返回一个负值

Uninitialize

Uninitialize 函数定义如表 13-11 所示。

表 13-11 Uninitialize 函数定义

原型	int32_t (*Uninitialize)(void)
描述	卸载 WDT 接口
参数	无
返回值	如果发生执行错误，返回一个负值

Control

Control 函数定义如表 13-12 所示。

表 13-12 Control 函数定义

原型	int32_t (*Control)(uint32_t control, uint32_t arg)
描述	配置 WDT 接口的设置
参数	control：WDT 驱动接口的一种设置 arg：WDT 的时钟周期
返回值	如果发生执行错误，返回一个负值

“control”和“arg”设置如表 13-13 所示

表 13-13 Control Settings or Operations

Options for Control	arg Specifies	Settings
AE350_WDT_CLKSRC_APB	time period (clock cycle)	The WDT timer refers to the APB clock.
AE350_WDT_CLKSRC_EXTERNAL		The WDT timer refers to the external clock.

Enable

Enable 函数定义如表 13-14 所示。

表 13-14 Enable 函数定义

原型	void (*Enable)(void)
描述	开启 WDT
参数	无
返回值	无

Disable

Disable 函数定义如表 13-15 所示。

表 13-15 Disable 函数定义

原型	void (*Disable)(void)
描述	关闭 WDT
参数	无
返回值	无

RestartTimer

RestartTimer 函数定义如表 13-16 所示。

表 13-16 RestartTimer 函数定义

原型	void (*RestartTimer)(void)
描述	重启 WDT
参数	无
返回值	无

ClearIrqStatus

ClearIrqStatus 函数定义如表 13-17 所示。

表 13-17 ClearIrqStatus 函数定义

原型	void (*ClearIrqStatus)(void)
描述	清除 WDT 中断状态
参数	无
返回值	无

GetStatus

GetStatus 函数定义如表 13-18 所示。

表 13-18 GetStatus 函数定义

原型	AE350_WDT_STATUS (*GetStatus)(void)
描述	获取 WDT 接口的状态
参数	无
返回值	WDT 接口的当前状态

14 DMA

14.1 简介

Gowin RiscV_AE350_SOC 包含一个 DMA（Direct Memory Access）控制器，用于在总线上的设备之间高效地传输数据。

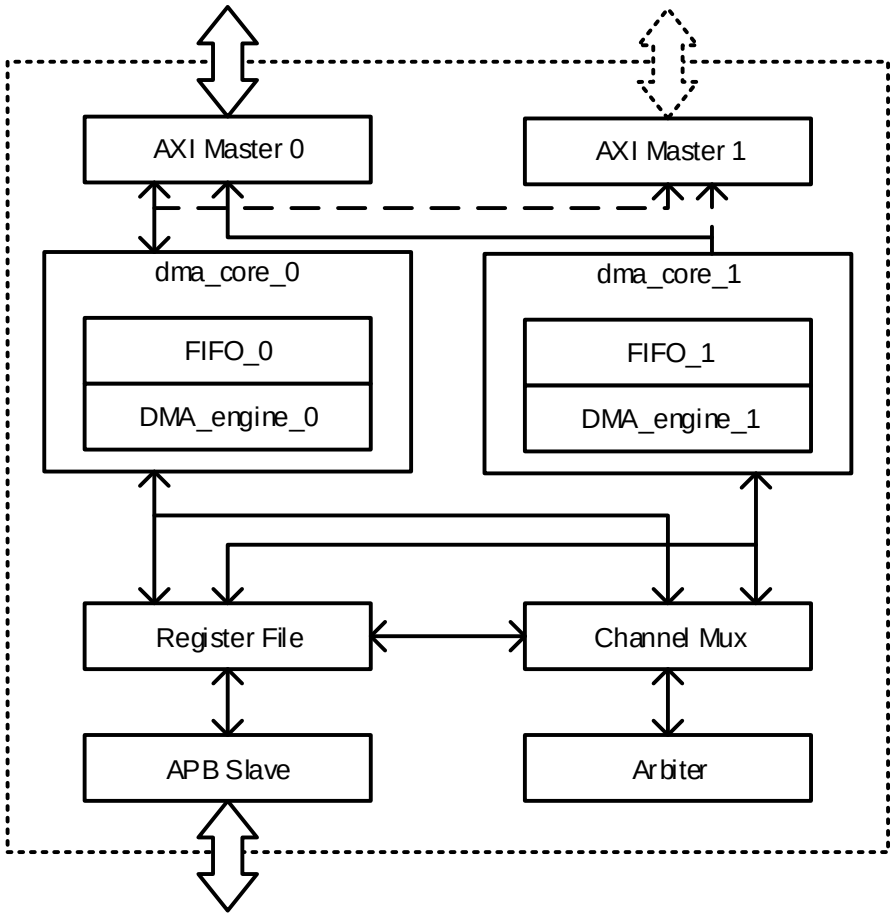
14.1.1 特征

- 支持 AMBA AXI4 和 APB4 总线协议规范
- 多达 8 个可配置的 DMA 通道
- 多达 16 组 DMA 请求/响应，用于硬件握手
- 多达 2 个 AXI 主机端口，用于数据传输
- 多达 2 个可配置的 DMA 核
- 支持一个 APB 从机端口，用于 DMA 寄存器编程
- 可配置 AXI 地址位宽：24-64 位
- 可配置 AXI 数据位宽：32、64、128 或 256 位
- 支持 2 级优先级的轮询仲裁机制
- 支持 DMA 链传输

14.1.2 结构框图

DMA 结构框图如图 14-1 所示。

图 14-1 DMA 结构框图



14.1.3 功能描述

DMA 控制器包括 2 个 AXI 主机接口用于数据传输，和一个 APB 从机接口用于寄存器编程。

DMA 控制器支持多达 8 个 DMA 通道，每个 DMA 通道提供一组寄存器来描述预期的数据传输。虽然可以同时开启多个 DMA 通道，但每个 DMA 核一次仅能控制一个 DMA 通道。

DMA 控制器支持多达 16 组硬件握手信号（dma_req/dma_ack），用于低速设备的数据传输。

DMA 控制器支持链传输功能，可以在不需要主处理器干预的情况下连续传输多个不连续的数据块。

DMA 控制器支持三种地址控制模式：递增模式、递减模式和固定模式。递增模式，DMA 控制器访问源/目的数据后，地址递增。递减模式，DMA 控制器访问源/目的数据后，地址递减。固定模式，DMA 控制器访问源/目的数据后，地址保持不变。

14.2 DMA 设备配置

14.2.1 DMA 设备

支持 DMA 功能的所有设备，硬件握手 ID 如表 14-1 所示。

表 14-1 DMA Hardware Handshake ID

Handshake ID	Devices
2	SPI TX
3	SPI RX
4	UART1 TX
5	UART1 RX
6	UART2 TX
7	UART2 RX
8	I2C

14.2.2 DMA 配置

支持 DMA 功能的所有设备，DMA 开关配置如表 14-2 所示。DMA 开关配置定义位于 bsp\config\config.h。

表 14-2 DMA 配置

Parameter	Options	Devices	Reset
DRV_I2C_DMA_TX_EN	0: Disable 1: Enable	I2C TX	0
DRV_I2C_DMA_RX_EN	0: Disable 1: Enable	I2C RX	0
DRV_UART1_DMA_TX_EN	0: Disable 1: Enable	UART1 TX	0
DRV_UART1_DMA_RX_EN	0: Disable 1: Enable	UART1 RX	0
DRV_UART2_DMA_TX_EN	0: Disable 1: Enable	UART2 TX	0
DRV_UART2_DMA_RX_EN	0: Disable 1: Enable	UART2 RX	0
DRV_SPI_DMA_TX_EN	0: Disable 1: Enable	SPI TX	0

Parameter	Options	Devices	Reset
DRV_SPI_DAM_RX_EN	0: Disable 1: Enable	SPI RX	0

14.3 寄存器定义

14.3.1 寄存器定义

DMA 寄存器定义如表 14-3 所示。DMA 寄存器定义位于 bsp\ae350\ae350.h。

表 14-3 寄存器定义

地址偏移	寄存器名称	描述
0x00	IDREV	ID and Revision Register
0x04~0x0C	-	Reserved
0x10	DMACFG	DMAC Configuration Register
0x14~0x1C	-	Reserved
0x20	DMACTRL	DMAC Control Register
0x24	CHABORT	Channel Abort Register
0x18~0x2C	-	Reserved
0x30	INTSTATUS	Interrupt Status Register
0x34	CHEN	Channel Enable Register
0x38~0x3C	-	Reserved
0x40 + N*0x20	ChnCtrl	Channel N Control Register (N: Number of channels, 0~7)
0x44 + N*0x20	ChnTranSize	Channel N Transfer Size Register (N: Number of channels, 0~7)
0x48 + N*0x20	ChnSrcAddrL	Channel N Source Address Low Part Register (N: Number of channels, 0~7)
0x4C + N*0x20	ChnSrcAddrH	Channel N Source Address High Part Register (N: Number of channels, 0~7)
0x50 + N*0x20	ChnDstAddrL	Channel N Destination Address Low Part Register (N: Number of channels, 0~7)
0x54 + N*0x20	ChnDstAddrH	Channel N Destination Address High Part Register (N: Number of channels, 0~7)
0x58 + N*0x20	ChnLLPointerL	Channel N Linked List Pointer Low Part Register (N: Number of channels, 0~7)

地址偏移	寄存器名称	描述
0x5C + N*0x20	ChnLLPointerH	Channel N Linked List Pointer High Part Register (N: Number of channels, 0~7)

14.3.2 寄存器描述

以下各节详细描述 DMA 寄存器定义。

寄存器类型缩略语概括如下：

- RO: Read-only
- WO: Write-only
- R/W: Readable and Writeable
- R/W1C: Readable and Write 1 to clear

ID 和 Revision 寄存器（0x00）

ID 和 Revision 寄存器用于保存 ID 和 Revision 编号，初始值依赖于所用版本。ID 和 Revision 寄存器定义如表 14-4 所示。

表 14-4 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:8	RO	ID number for DMAC	0x010230
RevMajor	7:4	RO	Major revision number	Revision Dependent
RevMinor	3:0	RO	Minor revision number	Revision Dependent

DMAC Configuration 寄存器（0x10）

DMAC Configuration 寄存器定义如表 14-5 所示。

表 14-5 DMAC Configuration Register

Name	Bit	Type	Description	Reset
ChainXfr	31	RO	Chain transfer 0: Chain transfer is not configured 1: Chain transfer is configured	Configuration dependent
ReqSync	30	RO	DMA request synchronization. The DMA request synchronization should be configured to avoid signal integrity problems when the request signal is not clocked by the system bus clock, which the DMA control logic operates in. If the	Configuration dependent

Name	Bit	Type	Description	Reset
			request synchronization is not configured, the request signal is sampled directly without synchronization. 0: Request synchronization is not configured 1: Request synchronization is configured	
-	29:26	-	Reserved	-
DataWidth	25:24	RO	AXI bus data width 0: 32 bits 1: 64 bits 2: 128 bits 3: 256 bits	Configuration dependent
AddrWidth	23:17	RO	AXI bus address width 24: 24 bits 25: 25 bits ... 64: 64 bits Others: Invalid	Configuration dependent
CoreNum	16	RO	DMA core number 0: 1 core 1: 2 cores	Configuration dependent
BusNum	15	RO	AXI bus interface number 0: 1 AXI bus 1: 2 AXI busses	Configuration dependent
ReqNum	14:10	RO	Request/acknowledge pair number 0: 0 pair 1: 1 pair 2: 2 pairs ... 16: 16 pairs	Configuration dependent
FIFODepth	9:4	RO	FIFO depth 4: 4 entries 8: 8 entries 16: 16 entries	Configuration dependent

Name	Bit	Type	Description	Reset
			32: 32 entries Others: Invalid	
ChannelNum	3:0	RO	Channel number 1: 1 channel 2: 2 channels ... 8: 8 channels Others: Invalid	Configuration dependent

DMAC Control 寄存器（0x20）

DMAC Control 寄存器定义如表 14-6 所示。

表 14-6 DMAC Control Register

Name	Bit	Type	Description	Reset
-	31:1	-	Reserved	-
Reset	0	WO	Software reset control. Write 1 to this bit to reset the DMA core and disable all channels.	0x0

Channel Abort 寄存器（0x24）

Channel Abort 寄存器用于控制 DMA 通道传输的终止，每个通道使用 1 位来控制。通过在位 n 写入 1，终止待处理的通道 n 的传输，当前正在运行的传输仍然会继续完成，以避免任何违反 AXI 协议的情况。通过向 Interrupt Status 寄存器对应的位写入 1 来清除终止位。

Channel Abort 寄存器定义如表 14-7 所示。

表 14-7 Channel Abort Register

Name	Bit	Type	Description	Reset
ChAbort	N-1:0	WO	Write 1 to bit n to abort channel n. The bits should only be set when the corresponding channels are enabled. Otherwise, the writes will be ignored for channels that are not enabled. (N: Number of channels)	0x0

Interrupt Status 寄存器（0x30）

Interrupt Status 寄存器包括终端计数、错误和终止状态。当通道遇到终端计数事件时，设置通道的终端计数状态。当通道遇到错误/终止事件

时，设置通道的错误/终止状态。每个通道都有一个状态位，如果相应的通道没有被配置，该状态位为 0。

Interrupt Status 寄存器定义如表 14-8 所示。

表 14-8 Interrupt Status Register

Name	Bit	Type	Description	Reset
-	31:24	-	Reserved	-
TC	23:16	R/W1C	The terminal count status, one bit per channel. The terminal count status is set when a channel transfer finishes without the abort or error event. 0: Channel n has no terminal count status 1: Channel n has terminal count status	0x0
Abort	15:8	R/W1C	The abort status of channel, one bit per channel. The abort status is set when a channel transfer is aborted. 0: Channel n has no abort status 1: Channel n has abort status	0x0
Error	7:0	R/W1C	The error status, one bit per channel. The error status is set when a channel transfer encounters the following error events: Bus error Unaligned address Unaligned transfer width Reserved configuration 0: Channel n has no error status 1: Channel n has error status	0x0

Channel Enable 寄存器（0x34）

Channel Enable 寄存器表示 DMA 通道的开启状态。只有相应的通道被配置，状态位才会存在。该寄存器是所有 Channel N Control 寄存器“Enable”位的别名。

Channel Enable 寄存器定义如表 14-9 所示。

表 14-9 Channel Enable Register

Name	Bit	Type	Description	Reset
ChEN	N-1:0	RO	Alias of the Enable field of all Channel N Control registers	0x0

Channel N Control 寄存器（0x40 + N*0x20）

Channel N Control（N：通道号 0~7）寄存器定义如表 14-10 所示。

表 14-10 Channel N Control Register

Name	Bit	Type	Description	Reset
SrcBusInfIdx	31	R/W	Bus interface index that source data is read from 0: Data is read from bus interface 0 1: Data is read from bus interface 1	0x0
DstBusInfIdx	30	R/W	Bus interface index that destination data is written to 0: Data is written to bus interface 0 1: Data is written to bus interface 1	0x0
Priority	29	R/W	Channel priority level 0: Lower priority 1: Higher priority	0x0
-	28	-	Reserved	0x0
SrcBurstSize	27:24	R/W	Source burst size. This field indicates the number of transfers before DMA channel re-arbitration. The burst transfer byte number is (SrcBurstSize * SrcWidth). 0: 1 transfer 1: 2 transfers 2: 4 transfers 3: 8 transfers 4: 16 transfers 5: 32 transfers 6: 64 transfers 7: 128 transfers 8: 256 transfers 9: 512 transfers 10: 1024 transfers 11~15: Reserved, setting this field with a reserved value triggers the error exception	0x0
SrcWidth	23:21	R/W	Source transfer width 0: Byte transfer 1: Half-word transfer	0x2

Name	Bit	Type	Description	Reset
			2: Word transfer 3: Double word transfer 4: Quad word transfer 5: Eight words transfer 6~7: Reserved, setting this field with a reserved value triggers the error exception	
DstWidth	20:18	R/W	Destination transfer width. Both the total transfer byte number and the burst transfer byte number should be aligned to the destination transfer width; otherwise the error event will be triggered. For example, destination transfer width should be set as byte transfer if total transfer byte is not aligned to half-word. See field SrcBurstSize above for the definition of burst transfer byte number and Channel N Transfer Size register（0x44 + N*0x20 for the definition of the total transfer byte number. 0: Byte transfer 1: Half-word transfer 2: Word transfer 3: Double word transfer 4: Quad word transfer 5: Eight words transfer 6~7: Reserved, setting this field with a reserved value triggers the error exception	0x2
SrcMode	17	R/W	Source DMA handshake mode 0: Normal mode 1: Handshake mode	0x0
DstMode	16	R/W	Destination DMA handshake mode 0: Normal mode 1: Handshake mode	0x0
SrcAddrCtrl	15:14	R/W	Source address control 0: Increment address 1: Decrement address 2: Fixed address 3: Reserved, setting the field with this value	0x0

Name	Bit	Type	Description	Reset
			triggers the error exception	
DstAddrCtrl	13:12	R/W	Destination address control 0: Increment address 1: Decrement address 2: Fixed address 3: Reserved, setting the field with this value triggers the error exception	0x0
SrcReqSel	11:8	R/W	Source DMA request select. Select the request/ack handshake pair that the source device is connected to.	0x0
DstReqSel	7:4	R/W	Destination DMA request select. Select the request/ack handshake pair that the destination device is connected to.	0x0
IntAbtMask	3	R/W	Channel abort interrupt mask 0: Allow the abort interrupt to be triggered 1: Disable the abort interrupt	0x0
IntErrMask	2	R/W	Channel error interrupt mask 0: Allow the error interrupt to be triggered 1: Disable the error interrupt	0x0
IntTCMask	1	R/W	Channel terminal count interrupt mask 0: Allow the terminal count interrupt to be triggered 1: Disable the terminal count interrupt	0x0
Enable	0	R/W	Channel enable bit 0: Disable 1: Enable	0x0

Channel N Transfer Size 寄存器（0x44 + N*0x20）

Channel N Transfer Size（N：通道号 0~7）寄存器定义如表 14-11 所示。

表 14-11 Channel N Transfer Size Register

Name	Bit	Type	Description	Reset
TranSize	31:0	R/W	Total transfer size from source. The total number of transferred bytes is (TranSize * SrcWidth). This register is cleared when the DMA transfer is	0x0

Name	Bit	Type	Description	Reset
			done. If a channel is enabled with zero total transfer size, the error event will be triggered and the transfer will be terminated.	

Channel N Source Address Low Part 寄存器（0x48 + N*0x20）

Channel N Source Address Low Part（N：通道号 0~7）寄存器定义如表 14-12 所示。

表 14-12 Channel N Source Address Low Part Register

Name	Bit	Type	Description	Reset
SrcAddrL	31:0	R/W	Low part of the source starting address. When the transfer completes, the value of {SrcAddrH, SrcAddrL} is updated to the ending address. This address must be aligned to the source transfer size; otherwise, an error event will be triggered.	0x0

Channel N Source Address High Part 寄存器（0x4C + N*0x20）

Channel N Source Address High Part（N：通道号 0~7）寄存器定义如表 14-13 所示。

表 14-13 Channel N Source Address High Part Register

Name	Bit	Type	Description	Reset
SrcAddrH	31:0	R/W	High part of the source starting address. When the transfer completes, the value of {SrcAddrH, SrcAddrL} is updated to the ending address. This register exists only when the address bus width is wider than 32 bits.	0x0

Channel N Destination Address Low Part 寄存器（0x50 + N*0x20）

Channel N Destination Address Low Part（N：通道号 0~7）寄存器定义如表 14-14 所示。

表 14-14 Channel N Destination Address Low Part Register

Name	Bit	Type	Description	Reset
DstAddrL	31:0	R/W	Low part of the destination starting address.	0x0

Name	Bit	Type	Description	Reset
			When the transfer completes, the value of {DstAddrH, DstAddrL} is updated to the ending address. This address must be aligned to the destination transfer size; otherwise the error event will be triggered.	

Channel N Destination Address High Part 寄存器（0x54 + N*0x20）

Channel N Destination Address High Part（N：通道号 0~7）寄存器定义如表 14-15 所示。

表 14-15 Channel N Destination Address High Part Register

Name	Bit	Type	Description	Reset
DstAddrH	31:0	R/W	High part of the destination starting address. When the transfer completes, the value of {DstAddrH, DstAddrL} is updated to the ending address. This address must be aligned to the destination transfer size; otherwise the error event will be triggered. This register exists only when the address bus width is wider than 32 bits.	0x0

Channel N Linked List Pointer Low Part 寄存器（0x58 + N*0x20）

Channel N Linked List Pointer Low Part（N：通道号 0~7）寄存器定义如表 14-16 所示。

表 14-16 Channel N Linked List Pointer Low Part Register

Name	Bit	Type	Description	Reset
LLPointerL	31:3	R/W	Low part of the pointer to the next descriptor. The pointer must be double word aligned.	0x0
-	2:1	-	Reserved	-
LLDBusInfIdx	0	R/W	Bus interface index that the next descriptor is read from 0: The next descriptor is read from bus interface 0 1: The next descriptor is read from bus	0x0

Name	Bit	Type	Description	Reset
			interface 1 This filed exists only when both dual master interfaces and chain transfer are supported	

Channel N Linked List Pointer High Part 寄存器（0x5C + N* 0x20）

Channel N Linked List Pointer High Part（N：通道号 0~7）寄存器定义如表 14-17 所示。

表 14-17 Channel N Linked List Pointer High Part Register

Name	Bit	Type	Description	Reset
LLPointerH	31:0	R/W	High part of the pointer to the next descriptor. This register exists only when the address bus width is wider than 32 bits.	0x0

14.4 驱动函数定义

14.4.1 驱动函数定义

DMA 驱动函数定义如表 14-18 所示。DMA 驱动函数定义位于 bsp\driver\ae350\dma_ae350.c、dma_ae350.h。

表 14-18 驱动函数定义

驱动函数	描述
dma_initialize	初始化 DMA 接口
dma_uninitialize	卸载 DMA 接口
dma_channel_configure	为下一次传输配置 DMA 通道
dma_channel_enable	开启 DMA 通道
dma_channel_disable	关闭 DMA 通道
dma_channel_get_status	检查 DMA 通道开启或关闭
dma_channel_get_count	获取 DMA 通道传输数据的数量
dma_channel_abort	退出 DMA 通道

14.4.2 驱动函数描述

以下各节详细描述 DMA 的驱动函数定义。

dma_initialize

dma_initialize 函数定义如表 14-19 所示。

表 14-19 dma_initialize 函数定义

原型	int32_t dma_initialize(void)
描述	初始化 DMA 接口
参数	无
返回值	如果发生执行错误，返回一个负值

dma_uninitialize

dma_uninitialize 函数定义如表 14-20 所示。

表 14-20 dma_uninitialize 函数定义

原型	int32_t dma_uninitialize(void)
描述	卸载 DMA 接口
参数	无
返回值	如果发生执行错误，返回一个负值

dma_channel_configure

dma_channel_configure 函数定义如表 14-21 所示。

表 14-21 dma_channel_configure 函数定义

原型	int32_t dma_channel_configure(uint8_t ch, uint32_t src_addr, uint32_t dst_addr, uint32_t size, uint32_t control, DMA_SignalEvent_t cb_event)
描述	为下一次传输配置 DMA 通道
参数	ch：通道号 src_addr：源地址 dst_addr：目的地址 size：传输数据的数量 control：通道控制 cb_event：指向 DMA_SignalEvent_t 回调函数的指针
返回值	如果发生执行错误，返回一个负值

dma_channel_enable

dma_channel_enable 函数定义如表 14-22 所示。

表 14-22 dma_channel_enable 函数定义

原型	int32_t dma_channel_enable(uint8_t ch)
描述	开启 DMA 通道
参数	ch：通道号
返回值	如果发生执行错误，返回一个负值

dma_channel_disable

dma_channel_disable 函数定义如表 14-23 所示。

表 14-23 dma_channel_disable 函数定义

原型	int32_t dma_channel_disable(uint8_t ch)
描述	关闭 DMA 通道
参数	ch：通道号
返回值	如果发生执行错误，返回一个负值

dma_channel_get_status

dma_channel_get_status 函数定义如表 14-24 所示。

表 14-24 dma_channel_get_status 函数定义

原型	uint32_t dma_channel_get_status(uint8_t ch)
描述	检查 DMA 通道开启或关闭
参数	ch：通道号
返回值	DMA 通道的状态

dma_channel_get_count

dma_channel_get_count 函数定义如表 14-25 所示。

表 14-25 dma_channel_get_cout 函数定义

原型	uint32_t dma_channel_get_count(uint8_t ch)
描述	获取 DMA 通道传输数据的数量
参数	ch：通道号
返回值	DMA 通道传输数据的数量

dma_channel_abort

dma_channel_abort 函数定义如表 14-26 所示。

表 14-26 dma_channel_abort 函数定义

原型	int32_t dma_channel_abort(uint8_t ch)
描述	退出 DMA 通道
参数	ch：通道号
返回值	如果运行过程中发生错误则返回负值

15 PLMT

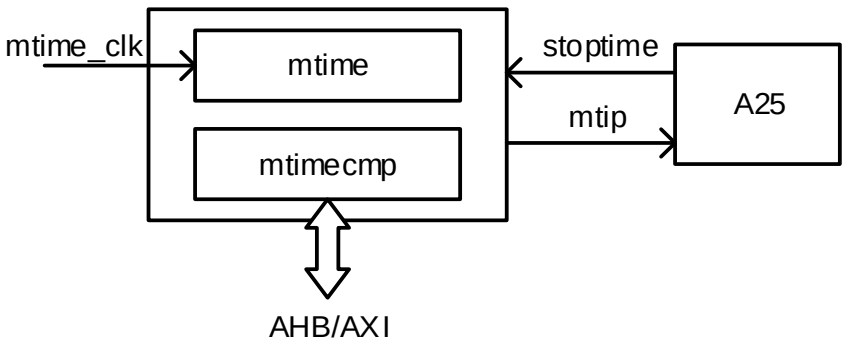
15.1 简介

RISC-V 处理器架构定义了一个 PLMT（Platform Level Machine Timer），提供一个实时的计数器，以及产生定时器中断。

15.1.1 结构框图

PLMT 结构框图如图 15-1 所示。

图 15-1 PLMT 结构框图



15.1.2 功能描述

PLMT 不同于传统计算平台的 RTC，根据 RISC-V 特权规范，定时器时钟（mtime_clk）可以工作在任意频率，只要是一个固定频率的时钟即可，而且不受所在平台其他时钟的时钟门控或频率缩放的影响，而 RTC 通常使用 32768Hz 的时钟。一般情况下，定时器时钟可以与 RTC 时钟共享相同的时钟源。RISC-V 特权规范要求软件可以通过平台特定机制获取定时器时钟的频率。

PLMT 主要由以下这些内存映射寄存器组成：mtime 和 mtimecmpn。mtime 寄存器是一个 64 位的实时计数器时钟，由 mtime_clk 计时。

mtimecmpn 寄存器保存一个 64 位的值用于与 mtime 做比较。 当 mtime 的值大于或等于 mtimecmpn 的值时，mtip[n]信号被断言，产生一个定时器中断。当 mtimecmpn 被写入后，中断被清除，mtip[n]信号被断言。

15.2 寄存器定义

15.2.1 寄存器定义

PLMT 寄存器定义如表 15-1 所示。PLMT 寄存器定义位于 bsp\ae350\ae350.h。

表 15-1 寄存器定义

地址偏移	寄存器名称	描述
0x00~0x03	MTIME[0]	mtime[31:0]
0x04~0x07	MTIME[1]	mtime[63:32]
0x08~0x0B	MTIMECMP0[0]	mtimecmp0[31:0] for hart 0
0x0C~0x0F	MTIMECMP0[1]	mtimecmp0[63:32] for hart 0
0x10~0x13	MTIMECMP1[0]	mtimecmp1[31:0] for hart 1
0x14~0x17	MTIMECMP1[1]	mtimecmp1[63:32] for hart 1
0x18~0x1B	MTIMECMP2[0]	mtimecmp2[31:0] for hart 2
0x1C~0x1F	MTIMECMP2[1]	mtimecmp2[63:32] for hart 2
0x20~0x23	MTIMECMP3[0]	mtimecmp3[31:0] for hart 3
0x24~0x27	MTIMECMP3[1]	mtimecmp3[63:32] for hart 3

15.2.2 寄存器描述

以下各节详细描述 PLMT 寄存器定义。

寄存器类型缩略语概括如下：

- R/W: Readable and Writeable

Machine Time 寄存器（0x00~0x04）

Machine Time 寄存器定义如表 15-2 所示。

表 15-2 Machine Time Register

Name	Bit	Type	Description	Reset
mtime	63:0	RW	A constant frequency real-time counter. A timer interrupt is generated when mtime >= mtimecmp	0x0

Machine Time Compare 寄存器（0x08~0x0C）

Machine Time Compare 寄存器定义如表 15-3 所示。

表 15-3 Machine Time Compare Register

Name	Bit	Type	Description	Reset
mtimecmp	63:0	RW	A timer interrupt is generated when mtime >= mtimecmp The timer interrupt input is de-asserted when this register is written.	0xFFFFFFFFFFFFFFFF

16 SMU

16.1 简介

Gowin RiscV_AE350_SOC 包含一个 SMU（System Manage Unit），提供多功能的系统管理能力，包括时钟、复位和电源控制。

16.2 寄存器定义

16.2.1 寄存器定义

SMU 寄存器定义如表 16-1 所示。SMU 寄存器定义位于 bsp\ae350\ae350.h。

表 16-1 寄存器定义

地址偏移	寄存器名称	描述
0x00	SYSTEMVER	System ID and Revision Register
0x04	BOARDID	Board ID Register
0x08	SYSTEMCFG	System Configuration Register
0x0C	-	Reserved
0x10	WRSR	Wake up and Reset Status Register
0x14	SMUCR	SMU Command Register
0x18	-	Reserved
0x1C	WRMASK	Wake up and Reset Mask Register
0x20	CER	Clock Enable Register
0x24	CRR	Clock Ratio Register
0x28~0x2C	-	Reserved
0x30	URREG0	User defined Register

地址偏移	寄存器名称	描述
0x34	URREG1	User defined Register
0x38~0x3C	-	Reserved
0x40	SCRATCH	Scratch Register
0x44	HARTS_RESET_REG	Harts Reset Register
0x48~0x4C	-	Reserved
0x50	HART0_RESET_VECTOR	Hart 0 Reset Vector Register
0x54	HART1_RESET_VECTOR	Hart 1 Reset Vector Register
0x58	HART2_RESET_VECTOR	Hart 2 Reset Vector Register
0x5C	HART3_RESET_VECTOR	Hart 3 Reset Vector Register
0x60~0xFC	-	Reserved
0x100	PMCR	Power Mode Cycle Register
0x104	PMDIR	Power Mode Disable Initialization Register
0x108	PDSR	Power Domain Select Register
0x10C	PSR	Power Status Register
0x110	PWRR	Power Wake up Record Register
0x114	PWER	Power Wake up Enable Register
0x118	PCR	Power Control Register
0x11C	PIR	Power Interrupt Register

16.2.2 寄存器描述

以下各节详细描述 SMU 寄存器定义。

寄存器类型缩略语概括如下：

- RO: Read-only
- R/W: Readable and writable
- W1C: Write 1 to clear

System ID 和 Revision 寄存器（0x00）

ID 和 Revision 寄存器用于保存 ID 和 Revision 编号，初始值依赖于所用版本。ID 和 Revision 寄存器定义如表 16-2 所示。

表 16-2 System ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:8	RO	ID number for SMU	0x414535

Name	Bit	Type	Description	Reset
RevMajor	7:4	RO	Major revision number	0x0
RevMinor	3:0	RO	Minor revision number	0x0

Board ID 寄存器（0x04）

Board ID 寄存器定义如表 16-3 所示。

表 16-3 Board ID Register

Name	Bit	Type	Description	Reset
ID	31:0	RO	ID number for Board	0x0174b010

System Configuration 寄存器（0x08）

System Configuration 寄存器定义如表 16-4 所示。

表 16-4 System Configuration Register

Name	Bit	Type	Description	Reset
CORENUM	7:0	RO	MCU core number	0x1

Wake up and Reset Status 寄存器（0x10）

Wake up and Reset Status 寄存器定义如表 16-5 所示。

表 16-5 Wake up and Reset Status Register

Name	Bit	Type	Description	Reset
DBG	10	W1C	Wake up by debug requests 0: Wake up event didn’t occur 1: Wake up event has occurred	0x0
ALM	9	W1C	Wake up by RTC alarm events 0: Wake up event didn’t occur 1: Wake up event has occurred	0x0
EXT	8	W1C	Wake up by external events 0: Wake up event didn’t occur 1: Wake up event has occurred	0x0
SW	4	W1C	Software Reset 0: Reset didn’t occur 1: Reset has occurred	HW, WDT, and SW are reset to 0 during the AOPD power on reset
WDT	3	W1C	Watchdog Reset	HW, WDT, and SW are

Name	Bit	Type	Description	Reset
			0: Reset didn't occur 1: Reset has occurred	reset to 0 during the AOPD power on reset
HW	2	W1C	Hardware Reset 0: Reset didn't occur 1: Reset has occurred	HW, WDT, and SW are reset to 0 during the AOPD power on reset
MPOR	1	W1C	MPD Power on Reset 0: No action 1: Reset has occurred	MPOR is reset to 1 during the MPD power on reset
APOR	0	W1C	AOPD Power on Reset 0: No action 1: Reset has occurred	APOR is reset to 1 during the AOPD power on reset

SMU Command 寄存器（0x14）

SMU Command 寄存器定义如表 16-6 所示。

表 16-6 SMU Command Register

Name	Bit	Type	Description	Reset
SMUCMD	7:0	WO	SMU command 0x3C: Software reset to reset the whole system. 0x5A: Power off the main power domain. 0x55: Standby command that triggers the standby request to the processor. If the Clock Ratio Register (CRR) has been modified, SMU waits for the clock ratio change to take effect and then directly wakes up the processor. Otherwise, SMU waits for a wake-up event before waking up the processor.	0x0

Wake up and Reset Mask 寄存器（0x1C）

Wake up and Reset Mask 寄存器定义如表 16-7 所示。

表 16-7 Wake up and Reset Mask Register

Name	Bit	Type	Description	Reset
DBGMASK	10	RW	Indicates whether debug requests will trigger wake up. 0: Debug requests will trigger wake up events	0x0

Name	Bit	Type	Description	Reset
			1: Debug requests will not trigger wake up events	
ALMMASK	9	RW	Indicates whether RTC events will trigger wake up. 0: RTC events will trigger wake up events 1: RTC events will not trigger wake up events	0x0
WIMASK	8	RW	Indicates whether external events will trigger wake up. 0: External events will trigger wake up events 1: External events will not trigger wake up events	0x0

Clock Enable 寄存器（0x20）

Clock Enable 寄存器用于控制系统平台中的所有时钟。Clock Enable 寄存器定义如表 16-8 所示。

表 16-8 Clock Enable Register

Name	Bit	Type	Description	Reset
PIT_CLK_EN	10	RW	PIT clock enable 0: Disable clock 1: Enable clock	0x1
WDT_CLK_EN	9	RW	WDT clock enable. 0: Disable clock 1: Enable clock	0x1
I2C_CCLK_EN	8	RW	I2C clock enable. 0: Disable clock 1: Enable clock	0x1
GPIO_CCLK_EN	7	RW	GPIO clock enable. 0: Disable clock 1: Enable clock	0x1
SPI2_CLK_EN	6	RW	SPI2 clock enable. 0: Disable clock 1: Enable clock	0x1
UART2_CLK_EN	4	RW	UART2 clock enable. 0: Disable clock 1: Enable clock	0x1

Name	Bit	Type	Description	Reset
UART1_CLK_EN	3	RW	UART1 clock enable. 0: Disable clock 1: Enable clock	0x1
PCLK_EN	2	RW	Main APB bus clock enable. 0: Disable clock 1: Enable clock	0x1
HCLK_EN	1	RW	AHB bus clock enable. 0: Disable clock 1: Enable clock	0x1
CCLK_EN	0	RW	Processor clock enable. 0: Disable clock 1: Enable clock	0x1

Clock Ratio 寄存器（0x24）

Clock Ratio 寄存器定义如表 16-9 所示。

表 16-9 Clock Ratio Register

Name	Bit	Type	Description	Reset
HPCLKSEL	3:1	RW	HCLK and PCLK clock ratio select 0: 1:1:1:1 1: 1:1:1:1/2 2: 1:1:1:1/4 3: 1:1:1/2:1/2 4: 1:1:1/2:1/4 5~7: Reserved	0x0
CCLKSEL	0	RW	Processor clock select 0: OSCH (Default) 1: Divide OSCH by 2	0x0

Scratch 寄存器（0x40）

Scratch 寄存器在系统其他部分关闭时保留一些值，可以用于在断电期间保存一些参数。Scratch 寄存器定义如表 16-10 所示。

表 16-10 Scratch Register

Name	Bit	Type	Description	Reset
SCRATCH	31:0	RW	Scratch register	0x0

Harts Reset 寄存器（0x44）

Harts Reset 寄存器用于与多核 AE350 平台兼容。Harts Reset 寄存器定义如表 16-11 所示。

表 16-11 Harts Reset Register

Name	Bit	Type	Description	Reset
HART0_RESET	0	RO	Hardwired to 1	0x1

Hart0 Reset Vector 寄存器（0x50）

Hart0 Reset 寄存器控制驱动到 AE350 RISC-V 处理器 reset_vector[31:0]输入信号的值，即引导启动地址。Hart0 Reset 寄存器定义如表 16-12 所示。

表 16-12 Hart0 Reset Vector Register

Name	Bit	Type	Description	Reset
RESET_VECTOR	31:0	RW	Entry address processor reset	0x80000000

Power Wake Up Enable 寄存器（0x114）

Power Wake Up Enable 寄存器用于控制唤醒事件。Power Wake Up Enable 寄存器定义如表 16-13 所示。

表 16-13 Power Wakeup Enable Register

Name	Bit	Type	Description	Reset
wakeup_en	31:0	RW	Each bit indicates one wakeup event. 0: Disable corresponding wakeup event 1: Enable corresponding wakeup event	0xFFFFFFFF

连接到 SMU 作为唤醒事件的外设中断如表 16-14 所示。

表 16-14 Peripheral Interrupt Sources for SMU Wakeup Events

Bits	Descriptions
10	DMA
9	UART2
8	UART1
7	GPIO
6	I2C
5	SPI
3	PIT
2	RTC alarm interrupt
1	RTC period interrupt

对于调试系统、系统总线和 AHB/APB 外设，表 16-15 所示的信号作为系统唤醒事件。

表 16-15 The SMU Wakeup Event for System

Bits	Descriptions
31	Hart0: meip/ueip/seip
30	Hart0: mtip
29	Hart0: msip
28	Hart0: debugint
27:23	Reserved
21	dbg_wakeup_req
20:1	Peripheral interrupt source, see Table 16-14
0	Reserved

对于处理器内核，表 16-16 所示的信号作为内核唤醒事件。

表 16-16 The SMU Wakeup Event for Core

Bits	Descriptions
31	Hart0: meip/ueip/seip
30	Hart0: mtip
29	Hart0: msip
28	Hart0: debugint

Bits	Descriptions
27	Watchdog timer interrupt
26:23	Reserved
21	dbg_wakeup_req
20:1	Peripheral interrupt source, see Table 16-14
0	Reserved

NMI 仅包含在处理器内核唤醒事件的电源域中，当系统出现意外挂起时，处理器内核的电源域可以通过 NMI 恢复并复位整个系统。

Power Control 寄存器（0x118）

Power Control 寄存器定义如表 16-17 所示。

表 16-17 Power Control Register

Name	Bit	Type	Description	Reset
Capability	2:0	RW	Power control capability. 0: Reserved 1: Dynamic voltage frequency scaling 2: Light sleep 3: Deep sleep 4~7: Reserved	0x0

Power Interrupt 寄存器（0x11C）

Power Interrupt 寄存器定义如表 16-18 所示。

表 16-18 Power Interrupt Register

Name	Bit	Type	Description	Reset
PIR_EN	0	RW	Enable power interrupt 0: Disable power interrupt 1: Enable power interrupt	0x0
PIR_PENDING	1	RW	Pending power interrupt 0: Not Pending power interrupt 1: Pending power interrupt	0x0

17 DSP

DSP 软件编程，参考 *MUG1032，Gowin RiscV_AE350_SOC DSP 软件编程用户手册*。

18 RTOS

Gowin RiscV_AE350_SOC 支持以下几种 RTOS：

- FreeRTOS
- uC/OS-III
- RT-Thread Nano
- Zephyr

18.1 FreeRTOS

18.1.1 特征

- FreeRTOS 是一个轻量级的实时操作系统
- FreeRTOS 作为一个轻量级的操作系统，功能包括：任务管理、时间管理、信号量、消息队列、内存管理、记录功能、软件定时器等，可基本满足较小系统的需求
- FreeRTOS 操作系统是完全免费的操作系统，具有源码公开、可移植、可裁剪、调度策略灵活的特点
- FreeRTOS 源码，请在 FreeRTOS 网站 http://www.FreeRTOS.org 下载
- RiscV_AE350_SOC 支持 FreeRTOS

18.1.2 版本

RiscV_AE350_SOC 支持的 FreeRTOS 版本：10.3.1。

18.1.3 配置

可以通过修改源码文件“FreeRTOSConfig.h”配置 FreeRTOS。

18.2 uC/OS-III

18.2.1 特征

- uC/OS-III 是一个可扩展的、可固化的、抢占式实时内核，管理的任务个数不受限制
- uC/OS-III 是第三代内核，提供了现代实时内核所期望的功能，包括资源管理、同步、任务间通信等
- uC/OS-III 提供了很多其它实时内核所没有的特性，比如能在运行时测量运行性能，直接发送信号或消息给任务，任务能同时等待多个信号量和消息队列
- uC/OS-III 遵循开源 Apache 许可证 2.0，开发人员可以免费下载使用
- uC/OS-III 源码，请在 Micrium 网站 http://www.micrium.com 下载
- RiscV_AE350_SOC 支持 uC/OS-III

18.2.2 版本

RiscV_AE350_SOC 支持的 uC/OS-III 版本：3.03.00。

18.2.3 配置

可以通过修改源码文件“uCOS_CONFIG\os_cfg.h”和“oscfg_app.h”配置 uC/OS-III。

可以通过修改源码文件“UCOS_BSP\bsp_os.c”和“bsp_os.h”获得板级支持。

18.3 RT-Thread Nano

18.3.1 特征

- RT-Thread Nano 是一个极简版的硬实时内核
- 由 C 语言开发，采用面向对象的编程思维，具有良好的代码风格，是一款可裁剪的、抢占式实时多任务的 RTOS
- 内存资源占用极小，功能包括任务处理、软件定时器、信号量、邮箱和实时调度等相对完整的实时操作系统特性
- 开源免费，遵循 Apache 许可证 2.0，实时操作系统内核及所有开源组件可以免费在商业产品中使用，不需要公布应用程序源码，没有潜在商业风险

- RT-Thread Nano 源码，请在 RT-Thread 网站 https://www.rt-thread.org 下载
- RiscV_AE350_SOC 支持 RT-Thread Nano

18.3.2 版本

RiscV_AE350_SOC 支持的 RT-Thread Nano 版本：3.1.5。

18.3.3 配置

可以通过修改源码文件“bsp\config.h”配置 RT-Thead Nano。

可以通过修改源码文件“bsp\board.c”获得板级支持。

18.4 Zephyr

18.4.1 特征

- Zephyr 是一个采用 Apache 许可证 2.0 的开源免费的 RTOS
- 为所有资源受限设备，构建了针对低功耗、小型内存微控制器设备而进行优化的物联网嵌入式小型、可扩展的 RTOS，支持多种硬件架构及多种开发板，可以在小至 8KB 内存的系统上运行
- 采用深入的安全开发生命周期：安全验证，模糊和渗透测试，频繁的代码审查，静态代码分析，威胁建模和审查，以防止代码中的后门
- 高度可配置，允许应用程序只包含所需的功能，并指定数量和大小
- 编译时资源定义，允许在编译时定义系统资源，从而减少代码大小并提高性能
- Zephyr 源码，请在 Zephyr 网站 https://zephyrproject.org/下载
- RiscV_AE350_SOC 支持 Zephyr

18.4.2 版本

RiscV_AE350_SOC 支持的 Zephyr 版本：2.4.0。

18.4.3 文件结构

Zephyr 的文件结构如下所示。

```
+- kernel/ # Zephyr kernel files
+- include/ # Zephyr kernel header files
```

```
+- arch/ # Files for architecture-specific kernel and SoC
|   +- riscv/ # Files for supporting RISC-V architecture
+- soc/ # SoC related code and configuration files
|   +- riscv/
|       +- riscv-privilege/
|           +- common/ # Files for RISC-V SoCs based on RISC-V privileged spec
|           +- andes_v5/ # Files for supporting RISC-V Andes V5 SoC platform (SoC)
|               +- ae350/ # Files for supporting RISC-V Andes V5 AE350 Platform (SoC)
+- boards/ # Boards related code and configuration files
|   +- riscv/
|       +- andes_v5/ # Files for supporting RISC-V Andes V5 SoC platforms (Board)
|           +- adp_xc7k_ae350_defconfig # Default kernel configuration of AE350 boards
|           +- adp_xc7k_ae350.dts # Device tree source file of AE350 boards
+- drivers/ # Device driver code
+- dts/ # Device tree source files
|   +- riscv/ # Device tree source files for RISC-V architecture
|       +- andes_v5_ae350.dtsi # Device tree source file for AE350 platform (SoC)
+- lib/ # Library code, including the minimal standard C library
+- doc/ # Zephyr technical documentation source files and tools used
+- misc/ # Miscellaneous code that doesn't belong to any of the other top-level directories
+- scripts/ # Various programs and other files used
```

```
to build and test Zephyr applications
    +- cmake/ # Additional build scripts need to build Zephyr
    +- subsys/ # Subystem of Zephyr, including USB device stack/network/file system codes
    +- share/ # Additional architecture independent data
    +- samples/ # Zephyr samples
    |    +- hello_world/ # Zephyr hello_world sample
    +- tests/ # Zephyr testsuite
    +- CMakeLists.txt # The top-level file for CMake build system
    +- Kconfig # The top-level Kconfig file
```

18.4.4 参考手册

Zephyr 的详细描述参考：https://docs.zephyrproject.org/2.4.0/

18.4.5 开发环境

使用 RDS 软件的终端工具“Cygwin”，假设 RDS 软件的安装路径为<RDS_ROOT>，则“Cygwin”位于“<RDS_ROOT>\cygwin\Cygwin.bat”。

18.4.6 构建方法

步骤 1

双击“<RDS_ROOT>\cygwin\Cygwin.bat”，打开终端工具“Cygwin”。

步骤 2

假设 Zephyr 根目录为<ZEPHYR_ROOT>，在终端工具“Cygwin”中，进入<ZEPHYR_ROOT>，设置环境变量，指定构建 Zephyr 的软件工具链，所用命令如下所示：

```
$ cd <ZEPHYR_ROOT>
$ source zephyr-env.sh
$ export ZEPHYR_TOOLCHAIN_VARIANT='cross-compile'
```

$ export CROSS_COMPILE=<RDS_ROOT>/toolchains/nds32le-elf-mculib-v5/bin/riscv32-elf-

注！

“Cygwin”与本地 PC 文件系统的接口目录为“/cygdrive”，例如，如果在“Cygwin”中进入 D 盘系统，则所用命令如下所示：

$ cd /cygdrive/d/

步骤 3

选定一个 Zephyr 应用程序，假设目录为<ZEPHYR_APP>，例如<ZEPHYR_ROOT>\samples\hello_world，进入<ZEPHYR_APP>，所用命令如下所示：

$ cd <ZEPHYR_APP>

步骤 4

建立文件夹“build”，用于构建 Zephyr 应用程序，进入“build”文件夹，所用命令如下所示：

$ mkdir build

$ cd build

步骤 5

执行“cmake”命令，指定 AE350 板级设备树，用于产生构建文件，所用命令如下所示：

$ cmake -DBOARD=adp_xc7k_ae350 ../

其中，可以参照应用场景修改 AE350 板级设备树文件，此文件位于<ZEPHYR_ROOT>\boards\riscv\andes_v5\adp_xc7k_ae350.dts。

例如，所用设备 UART2 的频率为 100MHz，波特率为 115200，则在 adp_xc7k_ae350.dts 文件中，修改 UART2 设备树定义如下所示。

```
&uart1 {
    status = "okay";
    clock-frequency = <100000000>;      // UCLK frequency is 100MHz
    current-speed = <115200>;           // Baud rate is 115200
};
```

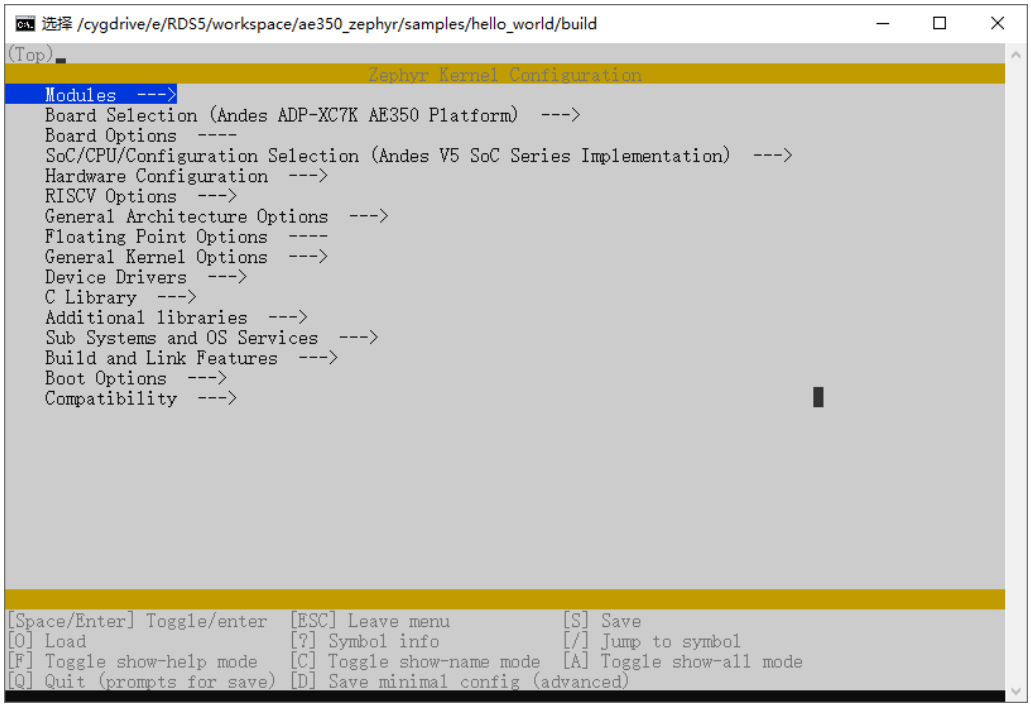
步骤 6

配置选定应用程序的 Zephyr 内核，指定内核功能，所用命令如下所示：

```
$ make menuconfig
```

Zephyr 的内核配置图形化接口如图 18-1 所示。

图 18-1 Zephyr 内核配置图形化接口



Zephyr 常用的内核配置选项如表 18-1 所示。

表 18-1 Zephyr 内核配置选项

选项	配置方法
Platform selection	(Top) > Board Selection > Andes ADP-XC7K AE350 Platform
SoC selection	(Top) > SoC/CPU/Configuration Selection > Andes V5 SoC Series Implementation (Top) > Hardware Configuration > Andes V5 SoC Selection > Andes AE350 SoC Implementation
CPU architecture selection	(Top) > Hardware Configration > CPU Architecture of SoC > RISCV32 CPU Architecture
FPU supporting	(Top) > Hardware Configration > Andes V5 FPU options > Double presion FPU
Other hardware configuration	(Top) > Hardware Configration > Enable cache (Top) > Hardware Configuration > Enable Andes V5 Hardware DSP (Top) > Hardware Configuration > Enable Andes V5

选项	配置方法
	performance throttling
XIP mode	(Top) > General Kernel Options > Execute in place
Device drivers	(Top) > Device Drivers > (uart_1) Device Name of UART Device for UART Console (Top) > Device Drivers > Console drivers > Use UART for console (Top) > Device Drivers > Serial Drivers > NS16550 serial driver (Top) > Device Drivers > Interrupt Controllers > Platform Level Interrupt Controller (PLIC) (Top) > Device Drivers > Timer Drivers > RISCV Machine Timer (Top) > Device Drivers > GPIO Drivers > Andes ATCGPIO100 GPIO driver (Top) > Device Drivers > I2C Drivers > Andes ATCIIC100 I2C driver (Top) > Device Drivers > Counter Drivers > Andes ATCPIT100 PIT driver & Do backup domain reset
Binary file	(Top) > Build and Link Features > Build Options > Build a binary in BIN format
Optimize	(Top) > Build and Link Features > Compiler Options > Optimization Level > Optimize for size

Zephyr 内核配置的详细描述，请参照：

https://docs.zephyrproject.org/2.4.0/guides/kconfig/index.html

步骤 7

执行“make”命令，构建应用程序，所用命令如下所示：

```
$ make
```

成功构建后，Binary 文件产生于 <ZEPHYR_APP>\build\zephyr\zephyr.bin。

19 基准测试程序

Gowin RiscV_AE350_SOC 支持以下几种基准测试程序，来测量处理器的硬件最高实际运行性能，以及软件优化的性能提升效果。

- Dhrystone
- CoreMark
- Whetstone

19.1 Dhrystone

19.1.1 简介

Dhrystone 主要目的是测试处理器的整数运算和逻辑运算的性能，最新版本是 1988 年更新的 Version 2.1。Dhrystone 标准的测试方法是单位时间内跑了多少次 Dhrystone 程序，其指标单位为 DMIPS/MHz。MIPS 是 Million Instructions Per Second 的缩写，每秒处理的百万级的机器语言指令数。DMIPS 中的 D 是 Dhrystone 的缩写，它表示了在 Dhrystone 标准的测试方法下的 MIPS。

19.1.2 应用程序

RiscV_AE350_SOC 提供 Dhrystone 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_dhrystone。

19.1.3 程序运行

Dhrystone 应用程序运行结果如下所示。

```
Dhrystone Benchmark, Version 2.1 (Language: C)

Program compiled without 'register' attribute
Please give the number of runs through the benchmark:
```

```
Execution starts, 2000000 runs through Dhrystone
Execution ends
Final values of the variables used in the benchmark:
Int_Glob:            5
        should be:   5
Bool_Glob:           1
        should be:   1
Ch_1_Glob:           A
        should be:   A
Ch_2_Glob:           B
        should be:   B
Arr_1_Glob[8]:       7
        should be:   7
Arr_2_Glob[8][7]:    2000010
        should be:   Number_Of_Runs + 10
Ptr_Glob->
  Ptr_Comp:          32896
        should be:   (implementation-dependent)
  Discr:             0
        should be:   0
  Enum_Comp:         2
        should be:   2
  Int_Comp:          17
        should be:   17
  Str_Comp:          DHRYSTONE PROGRAM,
SOME STRING
        should be:   DHRYSTONE PROGRAM,
SOME STRING
Next_Ptr_Glob->
  Ptr_Comp:          32896
        should be:   (implementation-dependent),
same as above
  Discr:             0
        should be:   0
  Enum_Comp:         1
        should be:   1
  Int_Comp:          18
        should be:   18
  Str_Comp:          DHRYSTONE PROGRAM,
SOME STRING
        should be:   DHRYSTONE PROGRAM,
SOME STRING
Int_1_Loc:           5
```

```
                should be:    5
Int_2_Loc:                    13
                should be:    13
Int_3_Loc:                    7
                should be:    7
Enum_Loc:                       1
                should be:    1
Str_1_Loc:                    DHRYSTONE PROGRAM, 1'ST
STRING
                should be:    DHRYSTONE PROGRAM, 1'ST
STRING
Str_2_Loc:                    DHRYSTONE PROGRAM, 2'ND
STRING
                should be:    DHRYSTONE PROGRAM, 2'ND
STRING
Microseconds for one run through Dhrystone :      0.4
Dhrystones per Second : 2416676.0

     DMIPS per MHz : 1.72
```

19.2 CoreMark

19.2.1 简介

CoreMark 主要目标是测试处理器核心性能，这个标准被认为比陈旧的 Dhrystone 标准更有实际价值。CoreMark 程序使用 C 语言写成，包含如下的运算法则：列举（寻找并排序），数学矩阵操作（普通矩阵运算）和状态机（用来确定输入流中是否包含有效数字），还包括 CRC。CoreMark 程序的最新版本是 Version 1.0。

CoreMark 标准的测试方法是在某配置参数组合下单位时间内跑了多少次 CoreMark 程序，其指标单位为 CoreMark/MHz。CoreMark 数字越高，意味着性能越高。

19.2.2 应用程序

Gowin RiscV_AE350_SOC 提供 CoreMark 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_coremark。

19.2.3 程序运行

CoreMark 应用程序运行结果如下所示。

```
The time is from mcycle

2K performance run parameters for coremark.
CoreMark Size       : 666
Total ticks         : 2263314978
Total time (secs): 13.566562
Iterations/Sec      : 2579.872491
CoreMark/MHz        : 3.224841
Iterations          : 35000
Compiler version : GCC10.3.0
Compiler flags      : -O3 -mcmodel=medium -funroll-all-loops -finline-limit=600 -ftree-dominator-opts -fno-if-conversion2 -fselective-scheduling -fno-code-hoisting -g3 -mcpu=a25 -fmessage-length=0 -fno-builtin -fomit-frame-pointer -fno-strict-aliasing
Memory location    : STACK
seedcrc             : 0xe9f5
[0]crclist          : 0xe714
[0]crcmatrix        : 0x1fd7
[0]crcstate         : 0x8e3a

[0]crcfinal         : 0x4983

Correct operation validated. See README.md for run and reporting rules.

CoreMark 1.0 : 2579.872491 / GCC10.3.0 -O3 -mcmodel=medium -funroll-all-loops -finline-limit=600 -ftree-dominator-opts -fno-if-conversion2 -fselective-scheduling -fno-code-hoisting -g3 -mcpu=a25 -fmessage-length=0 -fno-builtin -fomit-frame-pointer -fno-strict-aliasing / STACK
```

19.3 Whetstone

19.3.1 简介

Whestone 是一个综合性的基准测试程序，主要用于评估计算机的浮点运算性能，单位为每秒钟执行千条 Whetsone 指令（KWIPS）或每秒钟执行百万条 Whetstone 指令（MWIPS）。

Whetstone 是一个合成型的程序，包含了若干个代码片段。这些片段代表在科学计算应用中常用操作的混合，包含了科学计算常用的 C 函数（如 sin、cos、sqrt、exp 和 log 函数）整数和浮点数的算术运算、数组访问、条件分支和函数调用等。

当前 Whetstone 主要用于嵌入式系统的测试。

19.3.2 应用程序

RiscV_AE350_SOC 提供 Whetstone 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_whetstone。

19.3.3 程序运行

Whetstone 应用程序运行结果如下所示。

```
    ##############################################
#
    Single Precision C Whetstone Benchmark
    Calibrate
            0.02 Seconds          1    Passes (x
100)
            0.08 Seconds          5    Passes (x
100)
            0.40 Seconds         25    Passes (x
100)
            2.01 Seconds        125    Passes (x
100)
    Use 623    passes (x 100)
                Single Precision C/C++ Whetstone
Benchmark
    Loop content                Result
MFLOPS        MOPS     Seconds
    N1 floating point     -1.12475013732910156
177.776                 0.067
    N2 floating point     -1.12274742126464844
238.506                 0.351
    N3 if then else        1.00000000000000000
135225408.000            0.000
    N4 fixed point        12.00000000000000000
799.989                 0.245
    N5 sin,cos etc.        0.49909299612045288
```

```
12.822                  4.043
      N6 floating point         0.99999982118606567
192.000                 1.750
      N7 assignments            3.00000000000000000
479.987                 0.240
      N8 exp,sqrt etc.          0.75110614299774170
7.128                   3.251
      MWIPS
626.264                 9.948
      MWIPS/MHz
0.783
```

19.4 性能指标

RiscV_AE350_SOC 基于基准测试程序测量的性能指标如表 19-1 所示。

表 19-1 性能指标

Benchmark	Performance
Dhrystone	1.72 DMIPS/MHz
CoreMark	3.224841 CoreMark/MHz
Whetstone	0.783 MWIPS/MHz

20 应用程序

以下各节详细描述 Gowin RiscV_AE350_SOC 的应用程序旨在方便用户快速使用 Gowin RiscV_AE350_SOC 软件编程。

20.1 UART

20.1.1 程序描述

UART 应用程序描述了如何使用 UART 接口实现异步通信，演示了 UART 与 PC 之间通过终端仿真程序（例如 HyperTerm，TeraTerm 等）互相发送/接收数据。

程序初始后，UART 发送字符串“Press Enter to receive a message”到终端，然后等待终端发送数据。如果 UART 接收到终端发送的字符“G”，则发送字符串“Hello World!”，否则继续等待下一次终端发送数据。

20.1.2 应用程序

RiscV_AE350_SOC 提供 UART 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\uart。

20.1.3 程序运行

UART 应用程序运行结果如下所示。

```
Press Enter to receive a message
Hello World!
Hello World!
Hello World!
```

```
Hello World!
Hello World!
Hello World!
Hello World!
```

20.2 GPIO

20.2.1 程序描述

GPIO 应用程序描述了如何使用 GPIO 接口，演示了 GPIO 输入输出控制 7 段数码管。

程序初始化后，设置 GPIO 端口输入方向作为拨码开关输入，设置 GPIO 端口输出方向作为 7 段数码管输出，然后设置拨码开关中断触发方式为下降沿触发，开启相应的中断。完成所有的设置后，程序进入无限循环，等待用户控制 1~10 个拨码开关触发中断，当中断发生后，2 个 7 段数码管将演示对应的数字 0~9。

20.2.2 应用程序

RiscV_AE350_SOC 提供 GPIO 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\gpio。

20.2.3 程序运行

GPIO 应用程序运行结果如下所示。

```
It's a GPIO 7-Segments and Switches demo.
Please press SW1-SW10 and seven-segments will show the corresponding value…
7-segment number is 0
7-segment number is 1
7-segment number is 2
7-segment number is 3
7-segment number is 4
7-segment number is 5
7-segment number is 6
```

```
7-segment number is 7
7-segment number is 8
7-segment number is 9
```

20.3 I2C

20.3.1 程序描述

I2C 应用程序描述了如何使用 I2C 接口实现主机与从机通信，演示了 I2C 主机发送 10 个字节的数据到从机，然后从从机读回这 10 个字节的数据，检查结果。该程序需要连接两块开发板，一块作为主机，一块作为从机。

首先，初始化 I2C 从机开发板，等待主机发送数据。然后，初始化 I2C 主机开发板，主机发送 10 个字节数据到从机，然后从从机读回这 10 个字节的数据，检查结果，如果数据一致，则主机 UART 发送测试通过的信息。

I2C 主机开发板与从机开发板须共地。

20.3.2 应用程序

RiscV_AE350_SOC 提供 I2C 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\i2c。

20.3.3 程序运行

I2C 主机应用程序运行结果如下所示。

```
It's a I2C Master Mode demo.
====Please hardwire 2 test board=====
I2C master mode init …..
I2C FIFO master tx …..
I2C FIFO master rx …..
I2C FIFO master test pass …..
```

I2C 从机应用程序运行结果如下所示

```
It’s a I2C Slave mode demo.
```

```
====Please hardwire 2 test board=====
I2C slave mode init …..
```

20.4 SPI

20.4.1 程序描述

SPI 应用程序描述了如何使用 SPI 接口实现主机与从机通信，演示了 SPI 主机与从机发送/接收数据。该程序需要连接两块开发板，一块作为主机，一块作为从机。

程序初始化后，SPI 主机设置 8 位数据长度、33MHz 时钟频率，完成设置后，发送 8 个字节数据到从机。然后，从从机读回这 8 个字节数据，检查结果是否一致。

20.4.2 应用程序

RiscV_AE350_SOC 提供 SPI 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\spi。

20.4.3 程序运行

SPI 主机应用程序运行结果如下所示。

```
It’s a SPI Master Mode demo.
====Please hardwire 2 test board=====
Master write/read test…
data_in[0]=1
data_in[1]=2
data_in[2]=3
data_in[3]=4
data_in[4]=5
data_in[5]=6
data_in[6]=7
data_in[7]=8
```

SPI 从机应用程序运行结果如下所示。

```
It's a SPI Slave Mode demo.
====Please hardwire 2 test board=====
Slave write/read test…
data[0]=0
data[1]=1
data[2]=2
data[3]=3
data[4]=4
data[5]=5
data[6]=6
data[7]=7
```

20.5 RTC

20.5.1 程序描述

RTC 应用程序描述了如何使用 RTC 接口实现定时器中断，演示了半秒中断、秒中断、分中断、天中断和闹钟中断。

程序初始化后，设置“rtc_time”为 0，“alarm_time”为 1 分钟。当“rtc_time”计数 1 秒钟时，触发秒中断，UART 发送“Second interrupt done”信息。同样，“rtc_timer”计数等于“alarm_time”时，触发闹钟中断，UART 发送“Alarm interrupt done”。

20.5.2 应用程序

RiscV_AE350_SOC 提供 RTC 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\rtc。

20.5.3 程序运行

RTC 应用程序运行结果如下所示。

```
It's a Real Time Clock demo.
Half second interrupt operation.....
Half Second interrupt done.....
Second interrupt operation.....
```

```
Second interrupt done.....
After 3 seconds elapsed, you will see alarm interrupt done message.....
Alarm interrupt operation.....
Alarm interrupt done.....
```

20.6 PWM

20.6.1 程序描述

PWM 应用程序描述了如何使用 PWM 接口驱动扬声器或蜂鸣器，演示了 PWM 通道 1 给扬声器或蜂鸣器输出不同频率和占空比信号来演奏音符。

程序初始化后，UART 发送指令信息到终端，等待数据输入，如果终端发送数字 1~7，PWM 相对应的输出“Do”~“Si”频率到扬声器演奏音符。如果发送空格，PWM 停止输出。如果发送回车，程序退出。

20.6.2 应用程序

RiscV_AE350_SOC 提供 PWM 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\pwm。

20.6.3 程序运行

PWM 应用程序运行结果如下所示。

```
It's a Pulse Width Modulator demo.
Let's play piano...
[1 2 3 4 5 6 7]      ==> Play [Do Re Mi Fa So La Si]
[Space]              ==> Mute
[Enter]              ==> Quit
Input a [49]
Input a [50]
Input a [51]
Input a [52]
Input a [53]
Input a [54]
```

```
Input a [55]
Input a [Space]
Input a [Space]
Input a [Enter]
QUIT!
```

20.7 WDT 和 PIT Timer

20.7.1 程序描述

WDT 和 PIT Timer 应用程序描述了如何使用 WDT 和 PIT Timer 接口实现系统复位和锁定，演示了当 WDT 暂停后，WDT 复位系统。

WDT 初始化后，PIT Timer 将周期性的重启 WDT，PIT Timer 重启 WDT 10 次后，停止 PIT Timer，然后 WDT 超时，重启系统。如果 WDT 具有中断阶段，程序在超时后接收 NMI 中断，发送错误信息，然后等待系统复位。

20.7.2 应用程序

RiscV_AE350_SOC 提供 WDT 和 PIT Timer 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\wdt_pit。

20.7.3 程序运行

WDT 和 PIT Timer 应用程序运行结果如下所示。

```
It's a Watch Dog Timer and Programmable Interval Timer (as simple timer) demo.
WDT will reset whole system if WDT doesn't restarted in 1 seconds.
PIT (as simple timer) restart WDT every 0.5 seconds, so WDT doesn't reset system.
PIT (as simple timer) restart WDT (1 times), system still alive.
PIT (as simple timer) restart WDT (2 times), system still alive.
PIT (as simple timer) restart WDT (3 times), system still alive.
```

PIT (as simple timer) restart WDT (4 times), system still alive.

PIT (as simple timer) restart WDT (5 times), system still alive.

PIT (as simple timer) restart WDT (6 times), system still alive.

PIT (as simple timer) restart WDT (7 times), system still alive.

PIT (as simple timer) restart WDT (8 times), system still alive.

PIT (as simple timer) restart WDT (9 times), system still alive.

PIT (as simple timer) restart WDT (10 times), system still alive.

Then, We disable PIT (as simple timer), so the whole system will be reset by WDT.

20.8 PLMT

20.8.1 程序描述

PLMT 应用程序描述了如何激活机器模式相关的中断，开启机器模式定时器和机器模式软件中断。

机器模式定时器中断，周期设置为 2 秒，如果发生中断则设置机器模式软件中断的触发标志。

机器模式软件中断，当机器模式软件中断触发标志被机器模式定时器中断设置后，激活机器模式软件中断。

激活机器模式软件中断后，每 2 秒输出机器模式定时器活跃状态的信息。同时，按键 1、2、3 分别控制 led 1、2、3。

20.8.2 应用程序

Gowin RiscV_AE350_SOC 提供 PLMT 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\plmt。

20.8.3 程序运行

PLMT 应用程序运行结果如下所示。

```
It's a Platform Level Machine Timer demo.
Setup machine timer.
Setup machine software interrupt handler.
Setup GPIO key and led.
Message triggered from Machine Timer interrupt handler.
mtimer count:  0
Press key1 and led1 is on.
Message triggered from Machine Timer interrupt handler.
mtimer count:  1
Press key2 and led2 is on.
Message triggered from Machine Timer interrupt handler.
mtimer count:  2
Press key3 and led3 is on.
Message triggered from Machine Timer interrupt handler.
mtimer count:  3
Message triggered from Machine Timer interrupt handler.
mtimer count:  4
```

20.9 Printf

20.9.1 程序描述

Printf 应用程序描述了如何使用 UART 发送数据实现重定义标准库 printf()/sprint()函数。

20.9.2 应用程序

Gowin RiscV_AE350_SOC 提供 Printf 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\printf。

20.9.3 程序运行

Printf 应用程序运行结果如下所示。

```
It's a printf() demo.
Hello world!
5 = 5
-2147483647 = - max integer
char a = 'a'
hex ff = ff
hex 00 = 00
signed -3 = unsigned 4294967293 = hex fffffffd
0 message(s)
0 message(s) with %
justif: "left          "
justif: "         right"
  3: 0003 zero padded
  3: 3       left justif.
  3:       3 right justif.
-3: -003 zero padded
-3: -3     left justif.
-3:     -3 right justif.
```

20.10 Scanf

20.10.1 程序描述

Scanf 应用程序描述了如何使用 UART 接收数据实现重定义标准库 scanf()函数。

20.10.2 应用程序

Gowin RiscV_AE350_SOC 提供 Scanf 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\scanf。

20.10.3 程序运行

Scanf 应用程序运行结果如下所示。

```
It's a scanf() demo.
Enter a string : hello
Enter a hex : 0xaf
Enter an integer : 111
Demo scanf() PASS.
```

20.11 HSP

20.11.1 程序描述

HSP 应用程序描述了硬件堆栈保护和记录机制。

使用汉诺塔游戏描述硬件堆栈保护特征。首先开启硬件堆栈记录机制的运行方案，获取栈顶指针，用于递归循环移动汉诺塔。然后，开启硬件堆栈保护机制的运行方案，通过降低栈边界值继续移动汉诺塔，以此来产生栈溢出异常。

20.11.2 应用程序

Gowin RiscV_AE350_SOC 提供 HSP 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\hsp。

20.11.3 程序运行

HSP 应用程序运行结果如下所示。

```
It's a Hardware Stack Protection and Recording Mechanism demo.
The initial stack pointer : 0x8000000
[Hardware Stack Recording]
HANOI Tower with 4 disks :
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
```

```
Move disk 3 from A to B
Move disk 1 from C to A
Move disk 2 from C to B
Move disk 1 from A to B
Move disk 4 from A to C
Move disk 1 from B to C
Move disk 2 from B to A
Move disk 1 from C to A
Move disk 3 from B to C
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
HANOI(4) = 15 moves
Top of Stack : 0x7fffa90
[Hardware Stack Overflow Detection]
Set stack top bound : 0x7fffc00
Retest...
HANOI Tower with 4 disks :
[Exception] : Stack Overflow (sp = 0x7fffc00)
```

20.12 PFM

20.12.1 程序描述

PFM 应用程序描述了如何使用基本的 RISC-V 分析计数器“mcycle”和“minstret”。

通过“mcycle”和“minstret”硬件性能监视器，使用斐波那契程序测量 MCU 性能。演示两种测量方法，一种是使用计数器差异，另一种是清除计数器直接使用值。

20.12.2 应用程序

Gowin RiscV_AE350_SOC 提供 PFM 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\pfm。

20.12.3 程序运行

PFM 应用程序运行结果如下所示。

```
It's a Hardware Performance Monitor demo.
Run factorial(100) mathematics ...
Demo 1: Using Counter Differences.
Loop 0: Retired 317 instructions in 490 cycles
Loop 1: Retired 317 instructions in 336 cycles
Loop 2: Retired 317 instructions in 333 cycles
Demo 2: Clearing Counters, Using Values Directly.
Loop 0: Retired 315 instructions in 329 cycles
Loop 1: Retired 315 instructions in 321 cycles
Loop 2: Retired 315 instructions in 321 cycles
Hardware Performance Monitor Demo Completed.
```

20.13 PLIC

20.13.1 程序描述

PLIC 应用程序，非向量式中断处理，描述了如何激活机器模式和 PLIC 相关的中断，开启机器模式定时器、机器模式软件、PLIC PIT Timer 中断。

机器模式定时器中断，周期设置为 2 秒，如果发生中断则设置机器模式软件中断的触发标志。

PLIC PIT Timer 中断，周期设置为 200 毫秒，如果发生中断则设置机器模式软件中断的触发标志。

当机器模式定时器中断和 PLIC PIT Timer 中断中设置了机器模式软件中断的触发标志，机器模式软件中断被激活。

机器模式软件中断，每 2 秒发送机器模式定时器活跃状态的信息，每 200 毫秒发送 PLIC PIT Timer 活跃状态的信息。

20.13.2 应用程序

Gowin RiscV_AE350_SOC 提供 PLIC 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\plic。

20.13.3 程序运行

PLIC 应用程序运行结果如下所示。

```
It's a No Vectored Platform Level Interrupt Controller demo.
Setup machine timer.
Setup machine software interrupt handler.
Setup pit timer.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from Machine Timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer
```

```
interrupt handler.
    Message triggered from PLIC PIT as simple timer
interrupt handler.
    Message triggered from PLIC PIT as simple timer
interrupt handler.
    Message triggered from PLIC PIT as simple timer
interrupt handler.
    Message triggered from PLIC PIT as simple timer
interrupt handler.
    Message triggered from PLIC PIT as simple timer
interrupt handler.
    Message triggered from PLIC PIT as simple timer
interrupt handler.
    Message triggered from Machine Timer interrupt
handler.
```

20.14 Powerbrake

20.14.1 程序描述

Powerbrake 应用程序描述了硬件性能调节机制。

MCU 分别配置为最低性能或最高性能的条件下，运行数字游戏（0 => 9）。MCU 配置为最低性能时的循环计数大于 MCU 配置为最高性能时的循环计数。

20.14.2 应用程序

RiscV_AE350_SOC 提供 Powerbrake 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\powerbrake。

20.14.3 程序运行

Powerbrake 应用程序运行结果如下所示。

```
    It's a Hardware Performance Throttling Mechanism
demo.
    ========================================
============================
```

```
    [Part 1: Run the number game with
LOWEST/HIGHEST performance]
    MCU is configured to LOWEST performance
(T_LEVEL: 15)
    Go number game: 0 --> 9.
    0...1...2...3...4...5...6...7...8...9...
    consumed cycles: 16192768.
    MCU is configured to HIGHEST performance
(T_LEVEL: 0)
    Go number game: 0 --> 9.
    0...1...2...3...4...5...6...7...8...9...
    consumed cycles: 8916257.
    Consumed cycles (LOWEST performance) >
Consumed cycles (HIGHEST performance) => OK
    ========================================
============================
    [Part 2: Run the number game in ISR while
ENABLE/DISABLE fast interrupt mode (FAST_INT)]
    Setup machine software interrupt handler.
    MCU is configured to LOWEST performance
(T_LEVEL: 15)
    <ENABLE fast interrupt mode>
    Trigger machine software interrupt handler.
    It's machine software interrupt handler.
    Go number game in ISR: 0 --> 9.
    0...1...2...3...4...5...6...7...8...9...
    consumed cycles: 8916433.
    <DISABLE fast interrupt mode>
    Trigger machine software interrupt handler.
    It's machine software interrupt handler.
    Go number game in ISR: 0 --> 9.
    0...1...2...3...4...5...6...7...8...9...
    consumed cycles: 16193024.
```

```
Consumed cycles(DISABLE FAST_INT) >
Consumed cycles(ENABLE FAST_INT) => OK

Demo Hardware Performance Throttling
Mechanism PASS!
```

20.15 WFI

20.15.1 程序描述

WFI 应用程序描述了如何让 MCU 内核进入待机和唤醒。

通过执行“wfi”指令，MCU 内核进入待机模式。通过按键 1、2、3，唤醒 MCU 内核，UART 发送“Wake up from standby mode”信息。

20.15.2 应用程序

Gowin RiscV_AE350_SOC 提供 WFI 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\wfi。

20.15.3 程序运行

WFI 应用程序运行结果如下所示。

```
It's a Wait-for-Interrupt demo.

Entering StandBy mode.....

And press SW1 ~ SW3 trying to wake it up
zzzzzzzzzzzzzzzzzzzzzzzzzzZZZZZZZZZzzzzzzz...

Wake up from standby mode.....
```

20.16 Cache

20.16.1 程序描述

Cache 应用程序描述了硬件高速缓存机制。

在可以高速缓存的内存中进行运行代码修改，以达到数据一致性。首先，检查高速缓存特征，如果特征存在则开启指令和数据高速缓存。然后，在数据高速缓存上修改一个名为“g_selfmodify”的全局变量，并且使用指令“fence.i”做内存一致性。最后，检查“g_selfmodify”的正确性。

20.16.2 应用程序

Gowin RiscV_AE350_SOC 提供 Cache 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\cache。

20.16.3 程序运行

Cache 应用程序运行结果如下所示。

```
It's a L1 Cache demo.
MCU supports CCTL operation
MCU supports CCTL auto-increment
The L1C ICache sets = 256
The L1C ICache ways = 4
The L1C ICache line size = 32
The L1C DCache sets = 256
The L1C DCache ways = 4
The L1C DCache line size = 32
Enable L1C I cache
Enable L1C D cache
Execute original self modify code.
I/D cache flush
Run selfModifyCode Pass.
L1 Cache Completed.
```

20.17 Cache Lock

20.17.1 程序描述

Cache Lock 应用程序描述了硬件高速缓存锁定机制。

在可以高速缓存的内存中进行运行代码修改，以达到数据一致性，并且说明如何锁存修改的代码到指令高速缓存中。首先，检查高速缓存特征，如果特征存在则开启指令和数据高速缓存。然后，运行“A”补丁代码，修改一个名为“g_selfmodify”的全局变量，分别使用 CCTL 指令去做没有锁定补丁代码和锁定补丁代码时的内存一致性。接下来，在指令高速缓存中运行大小为 32KB 的 nop 指令来冲洗“A”补丁代码。最后，运行另一个“B”

补丁代码，修改“g_selfmodify”，检查“g_selfmodify”的正确性，完成高速缓存锁定。

20.17.2 应用程序

Gowin RiscV_AE350_SOC 提供 Cache Lock 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\cache_lock。

20.17.3 程序运行

Cache Lock 应用程序运行结果如下所示。

```
It's a L1 Cache Lock Mechanism demo.
MCU supports cache lock feature.
==============demo cache unlock===============
Run unlock self modify code, expect g_selfmodify=8888888.....
Run self modify code to set g_selfmodify = 7777777.....
Run 32k NOP buf to full-fill Icache to flush the unlocked Icache line of self modify code.....
Run self modify code to set g_selfmodify = 8888888.....
The g_selfmodify = 8888888
Run cache unlock scenario PASS.
==============demo cache lock=================
Run lock self modify code, expect g_selfmodify=7777777.....
Run self modify code to set g_selfmodify = 7777777.....
Run 32k NOP buf to full-fill Icache to flush the unlocked Icache line of self modify code.....
Run self modify code to set g_selfmodify = 8888888.....
The g_selfmodify = 7777777
```

```
Run cache lock scenario PASS.
Demo L1 Cache Lock Completed.
```

20.18 ILM/DLM

20.18.1 程序描述

IDLM 应用程序描述了如何通过从机端口访问 ILM/DLM，演示了通过不同总线地址与 ILM/DLM 的数据传输。

首先，检查 ILM/DLM 的配置，启动从 ROM/DDR 到 ILM/DLM 的数据传输，以及从 ILM/DLM 到 DDR 的数据传输，然后分别读回数据，验证数据传输的正确性。

20.18.2 应用程序

RiscV_AE350_SOC 提供 IDLM 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\idlm。

20.18.3 程序运行

IDLM 应用程序运行结果如下所示。

```
It's a ILM/DLM Access by Slave Port demo.
Check ILM: MICM_CFG = 0x439ada
The system has ILM...
Check DLM: MDCM_CFG = 0x439ada
The system has DLM...
ILM Size = 0x10000 (64KB), DLM Size = 0x10000 (64KB)
Move 65536 data from ROM to ILM
Checking data... OK.
Move 32768 data from DDR to DLM
Checking data... OK.
Move 1024 data from ILM to DDR
Checking data... OK.
Move 1024 data from DLM to DDR
```

```
Checking data... OK.
Access ILM/DLM by Slave Port Completed.
```

20.19 MM

20.19.1 程序描述

MM 应用程序描述了如何使用堆内存管理。

内存初始化后，首先为字符数组动态申请一段堆内存，然后进行数据操作，最后释放内存。

20.19.2 应用程序

Gowin RiscV_AE350_SOC 提供 MM 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\mm。

20.19.3 程序运行

MM 应用程序运行结果如下所示。

```
It's a Memory Management demo.
allocate a...
allocate b...
set a...
copy a to b...
a :
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa
b :
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
```

```
aa aa aa aa aa aa aa aa aa aa aa aa
compare a and b PASS.
free a PASS.
free b PASS.
```

20.20 Interrupt Priority

20.20.1 程序描述

Interrupt Priority 应用程序描述了如何使用多个外设中断。

开启 GPIO 按键和 PIT Timer，设置 GPIO 中断优先级高于 PIT Timer 中断优先级。PIT Timer 中断每 4 秒触发一次中断，当 GPIO 按键触发中断后，打断 PIT Timer 的中断，执行 GPIO 中断。

20.20.2 应用程序

Gowin RiscV_AE350_SOC 提供 Interrupt Priority 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\intr。

20.20.3 程序运行

Interrupt Priority 应用程序运行结果如下所示。

```
It's a Multiple Peripherals Interrupts demo.
Setup PIT timer.
Setup GPIO.
It's in PIT timer interrupt.
It's in PIT timer interrupt.
It's in GPIO interrupt.
It's in PIT timer interrupt.
It's in PIT timer interrupt.
It's in PIT timer interrupt.
It's in PIT timer interrupt.
It's in GPIO interrupt.
It's in PIT timer interrupt.
```

```
It's in PIT timer interrupt.
It's in GPIO interrupt.
It's in GPIO interrupt.
It's in PIT timer interrupt.
It's in PIT timer interrupt.
```

20.21 LED

20.21.1 程序描述

LED 应用程序描述了如何使用 GPIO 接口，演示了流水灯。

程序初始化后，设置 GPIO 端口输出方向用于流水灯，当执行每个流水灯时，UART 发送点灯信息。

20.21.2 应用程序

Gowin RiscV_AE350_SOC 提供 LED 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\led。

20.21.3 程序运行

LED 应用程序运行结果如下所示。

```
It’s a Waterfall Led demo.
led[1] is on 381422.
led[2] is on 381376.
led[3] is on 381376.
```

20.22 Vectored PLIC

20.22.1 程序描述

Vectored PLIC 应用程序，向量式中断处理，描述了如何激活机器模式和 PLIC 相关的中断，开启机器模式定时器、机器模式软件、PLIC PIT Timer 中断。

机器模式定时器中断，周期设置为 2 秒，如果发生中断则设置机器模式软件中断的触发标志。

PLIC PIT Timer 中断，周期设置为 200 毫秒，如果发生中断则设置机

器模式软件中断的触发标志。

当机器模式定时器中断和 PLIC PIT Timer 中断中设置了机器模式软件中断的触发标志，机器模式软件中断被激活。

机器模式软件中断，每 2 秒发送机器模式定时器活跃状态的信息，每 200 毫秒发送 PLIC PIT Timer 活跃状态的信息。

20.22.2 应用程序

Gowin RiscV_AE350_SOC 提供 Vectored PLIC 应用程序设计：

…\ref_design\MCU_RefDesign\ae350_vectored。

20.22.3 程序运行

Vectored PLIC 应用程序运行结果如下所示。

```
It's a Vectored PLIC demo.
Setup machine timer.
Setup machine software interrupt handler.
Setup pit timer.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
```

```
        Message triggered from Machine Timer interrupt
handler.
        Message triggered from PLIC PIT as simple timer
interrupt handler.
        Message triggered from PLIC PIT as simple timer
interrupt handler.
        Message triggered from PLIC PIT as simple timer
interrupt handler.
        Message triggered from PLIC PIT as simple timer
interrupt handler.
        Message triggered from PLIC PIT as simple timer
interrupt handler.
        Message triggered from PLIC PIT as simple timer
interrupt handler.
        Message triggered from PLIC PIT as simple timer
interrupt handler.
        Message triggered from PLIC PIT as simple timer
interrupt handler.
        Message triggered from PLIC PIT as simple timer
interrupt handler.
        Message triggered from PLIC PIT as simple timer
interrupt handler.
        Message triggered from Machine Timer interrupt
handler.
```

20.23 PMP 和 Privilege

20.23.1 程序描述

PMP 和 Privilege 应用程序描述了如何使用正确的特权模式来保护物理内存。

首先，在机器模式下，使用 PMP 配置初始化一个“PA”内存区域。然后，切换 Machine 模式到 Supervisor 模式，在切换 Supervisor 模式到 User 模式。当程序运行在 User 模式，在没有 Write/Execute 特权的情况下，试图从“PA”内存区域中 Store/Fetch 数据，则会产生指令访问异常。

20.23.2 应用程序

Gowin RiscV_AE350_SOC 提供 PMP 和 Privilege 应用程序设计：…\ref_design\MCU_RefDesign\ae350_privilege。

20.23.3 程序运行

PMP 和 Priviledge 应用程序运行结果如下所示。

```
MCU supports N-Extension feature.
PMP uses NAPOT scheme!.
Machine mode switches to supervisor mode.
========================================
========================
It's running in supervisor mode.
Hello World [SV Mode]!
Supervisor mode PASS.
========================================
========================
Supervisor mode switches to user mode.
========================================
========================
It's running in user mode.
It's a Physical Memory Protection demo.
Try to store a word to a region allowing in user or
machine mode.
You should see the [Store Pass] message ...
[Store Pass]
Next, try to store a word to a region requiring in
machine mode.
You should see the [Data Store Access Fault]
message ...
[N-Extention U-Mode: Data Store Access Fault]
Finally, try to execute machine mode code.
You should see the [Instruction Fetch Fault]
```

```
message ...
    [N-Extention U-Mode: Instruction Fetch Fault]
    PMP in user mode Pass!
    ========================================
========================
```

20.24 FreeRTOS

20.24.1 程序描述

FreeRTOS 应用程序描述了如何使用 RTOS FreeRTOS 管理多线程。

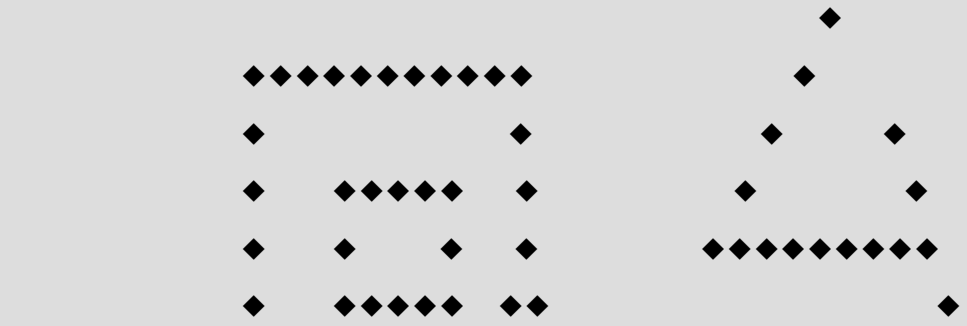
20.24.2 应用程序

Gowin RiscV_AE350_SOC 提供 FreeRTOS 应用程序设计：…\ref_design\MCU_RefDesign\ae350_freertos。

20.24.3 程序运行

FreeRTOS 应用程序运行结果如下所示。

```
    ************************************************************
***************
    Name:        FreeRTOS-AE350_SOC
    Edition:     V10.3.1
    Compiled:    Apr 24 2023, 09:20:30
    Author:      GowinSemiconductor
    ************************************************************
***************
    ************************************************************
***************
                        ◆
                   ◆◆◆◆◆◆◆◆◆◆◆          ◆◆◆◆◆◆◆
                     ◆◆◆◆◆◆◆
                     ◆         ◆
                     ◆◆◆◆◆◆◆           ◆◆◆◆◆◆◆◆◆◆◆
```



```
**********************************************************
***************
1.task1
0.task0
0.task0
0.task0
0.task0
1.task1
0.task0
0.task0
0.task0
0.task0
```

20.25 uC/OS-III

20.25.1 程序描述

uC/OS-III 应用程序描述了如何使用 RTOS uC/OS-III 管理多线程。

20.25.2 应用程序

Gowin RiscV_AE350_SOC 提供 uC/OS-III 应用程序设计：

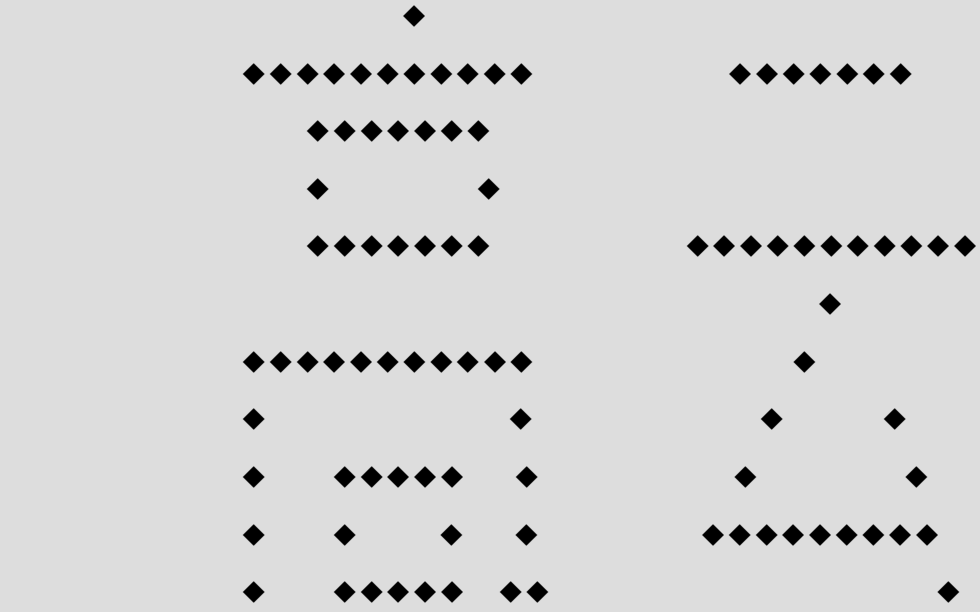
…\ref_design\MCU_RefDesign\ae350_ucosiii。

20.25.3 程序运行

uC/OS-III 应用程序运行结果如下所示。

```
**********************************************************
```

```
***************
    Name:       uCOS_III-AE350_SOC
    Edition:    V1.0.0
    Compiled:   Apr 25 2023, 08:53:15
    Author:     GowinSemiconductor
    ************************************************************
***************
    ************************************************************
***************
```



```
    ************************************************************
***************
    0.startup_task
    1.app_task
    0.startup_task
    0.startup_task
    0.startup_task
    0.startup_task
    1.app_task
    0.startup_task
    0.startup_task
    0.startup_task
```

```
0.startup_task
1.app_task
```

20.26 RT-Thread Nano

20.26.1 程序描述

RT-Thread Nano 应用程序描述了 RTOS RT-Thread Nano，演示了流水灯。

20.26.2 应用程序

Gowin RiscV_AE350_SOC 提供 RT-Thread Nano 应用程序设计：…\ref_design\MCU_RefDesign\ae350_rtthread_nano。

20.26.3 程序运行

RT-Thread Nano 应用程运行结果如下所示。

```
It's a RT-Thread Nano demo.
Initializes RT hw board...
 \ | /
- RT -         Thread Operating System
 / | \         3.1.5 build Apr 24 2023
 2006 - 2020 Copyright by rt-thread team
RT demo...
```

20.27 Zephyr

20.27.1 程序描述

Zephyr 应用程序描述了如何构建 Zephyr 及其应用程序。

20.27.2 应用程序

Gowin RiscV_AE350_SOC 提供 Zephyr 应用程序设计：…\ref_design\MCU_RefDesign\ae350_zephyr。

20.27.3 程序运行

Zephyr 应用程序之 hello_world 运行结果如下所示。

```
*** Booting Zephyr OS version 2.4.0  ***
Hello World! adp_xc7k_ae350
Hello Gowin! RiscV_AE350_SOC
```

20.28 C++

20.28.1 程序描述

C++应用程序描述了如何构建 C++应用程序。

创建 C++软件工程使用标准输入输出流“cout”输出信息，包括构造函数和析构函数的信息。

20.28.2 应用程序

Gowin RiscV_AE350_SOC 提供 C++应用程序设计：

…\ref_design\MCU_RefDesign\ae350_cpp。

20.28.3 程序运行

C++应用程运行结果如下所示。

```
It is a C++ demo program
I am Object constructor
Length of line : 6
I am Object destructor
```

20.29 Extended AHB Slave

20.29.1 程序描述

Extended AHB Slave 应用程序描述了如何使用扩展的 AHB Slave 接口。Extended AHB Slave 接口，连接外部的 AHB 接口的乘法器设备，实现乘数与被乘数的相乘运算。

20.29.2 应用程序

Gowin RiscV_AE350_SOC 提供 Extended AHB Slave 应用程序设计：

…\solution\Extended_AHB_Slave\ref_design\MCU_RefDesign\ae350_ext_ahb_slave。

20.29.3 程序运行

Extended AHB Slave 应用程序运行结果如下所示。

```
It's an Extended AHB Slave demo.
Start to multiply...
Multiply.....
Multiplication completed.
Show:
Multiplier: 50
Multiplicand: 60
Command: 0
Result: 3000
```

20.30 Extended APB Slave

20.30.1 程序描述

Extended APB Slave 应用程序描述了如何使用扩展的 APB Slave 接口。Extended APB Slave 接口，连接外部的 APB 接口的 GPIO 设备，实现 3 个按键控制 3 个 LED，开启 Extended Interrupts 作为 GPIO 设备的中断。

20.30.2 应用程序

Gowin RiscV_AE350_SOC 提供 Extended APB Slave 应用程序设计：

…\solution\Extended_APB_Slave\ref_design\MCU_RefDesign\ae350_ext_apb_slave。

20.30.3 程序运行

Extended APB Slave 应用程序运行结果如下所示。

```
It's an Extended APB Slave demo.
Please press key1, key2 or key3 to control led1, led2 or led3...
Press key1 and led1 is on.
Press key2 and led2 is on.
Press key3 and led3 is on.
Press key2 and led2 is on.
Press key1 and led1 is on.
```

20.31 Extended Interrupts

20.31.1 程序描述

Extended Interrupts 应用程序描述了如何使用扩展的 16 个中断信号。通过按键分别触发扩展的 16 个中断信号，进入对应的中断处理程序，打印信息。

20.31.2 应用程序

Gowin RiscV_AE350_SOC 提供 Extended Interrupts 应用程序设计：

…\solution\Extended_Interrupts\ref_design\MCU_RefDesign\ae350_ext_intr。

20.31.3 程序运行

Extended Interrupts 应用程序运行结果如下所示。

```
It's an Extended Interrupts demo.
Entry extended interrupt 14...
Entry extended interrupt 15...
Entry extended interrupt 16...
Entry extended interrupt 15...
Entry extended interrupt 14...
```

20.32 Flash Memory R/W

20.32.1 程序描述

Flash Memory R/W 应用程序描述了如何使用 Flash 作为内存外设来读、写和擦除操作。

Flash 初始化后，切换为读、写和擦除的内存访问模式。Flash 擦除一段指定起始地址和大小的地址空间，然后指定地址写入数据。最后，读取指定地址的数据，与写入时的数据对比，验证读与写。

20.32.2 应用程序

Gowin RiscV_AE350_SOC 提供 Flash Memory R/W 应用程序设计：

…\solution\Extended_APB_Flash\ref_design\MCU_RefDesign\ae350_ext_flash。

20.32.3 程序运行

Flash Memory R/W 应用程序运行结果如下所示。

```
It's a Flash demo.
Write data to Flash...
Write data done.
Read data from Flash...

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73
74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91
92 93 94 95 96 97 98 99 100 101 102 103 104 105 106
107 108 109 110 111 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129 130 131
132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155
156 157 158 159 160 161 162 163 164 165 166 167
168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191
192 193 194 195 196 197 198 199 200 201 202 203
204 205 206 207 208 209 210 211 212 213 214 215
216 217 218 219 220 221 222 223 224 225 226 227
228 229 230 231 232 233 234 235 236 237 238 239
```

240 241 242 243 244 245 246 247 248 249 250 251
252 253 254 255
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93
94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120
121 122 123 124 125 126 127 128 129 130 131 132
133 134 135 136 137 138 139 140 141 142 143 144
145 146 147 148 149 150 151 152 153 154 155 156
157 158 159 160 161 162 163 164 165 166 167 168
169 170 171 172 173 174 175 176 177 178 179 180
181 182 183 184 185 186 187 188 189 190 191 192
193 194 195 196 197 198 199 200 201 202 203 204
205 206 207 208 209 210 211 212 213 214 215 216
217 218 219 220 221 222 223 224 225 226 227 228
229 230 231 232 233 234 235 236 237 238 239 240
241 242 243 244 245 246 247 248 249 250 251 252
253 254 255
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93
94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120
121 122 123 124 125 126 127 128 129 130 131 132
133 134 135 136 137 138 139 140 141 142 143 144
145 146 147 148 149 150 151 152 153 154 155 156
157 158 159 160 161 162 163 164 165 166 167 168
169 170 171 172 173 174 175 176 177 178 179 180
181 182 183 184 185 186 187 188 189 190 191 192
193 194 195 196 197 198 199 200 201 202 203 204
205 206 207 208 209 210 211 212 213 214 215 216
217 218 219 220 221 222 223 224 225 226 227 228
229 230 231 232 233 234 235 236 237 238 239 240
241 242 243 244 245 246 247 248 249 250 251 252
253 254 255
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57

```
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93
94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120
121 122 123 124 125 126 127 128 129 130 131 132
133 134 135 136 137 138 139 140 141 142 143 144
145 146 147 148 149 150 151 152 153 154 155 156
157 158 159 160 161 162 163 164 165 166 167 168
169 170 171 172 173 174 175 176 177 178 179 180
181 182 183 184 185 186 187 188 189 190 191 192
193 194 195 196 197 198 199 200 201 202 203 204
205 206 207 208 209 210 211 212 213 214 215 216
217 218 219 220 221 222 223 224 225 226 227 228
229 230 231 232 233 234 235 236 237 238 239 240
241 242 243 244 245 246 247 248 249 250 251 252
253 254 255

    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29 30

    Flash R/W PASS.
```

20.33 Atomic Instructions

20.33.1 程序描述

Atomic Instructions 应用程序描述了如何调用 RV32“A”内建函数实现原子指令访问。

20.33.2 应用程序

Gowin RiscV_AE350_SOC 提供 Atomic Instructions 应用程序设计：…\ref_design\MCU_RefDesign\ae350_atomic。

20.33.3 程序运行

Atomic Instructions 应用程序运行结果如下所示。

```
It's an Atomic Instruction demo.
amoswap.w : oldv = 0x0
amoadd.w : oldv = 0xa
amoxor.w : oldv = 0x14
```

```
amoand.w : oldv = 0x22334441
amoor.w : oldv = 0x22334441
amomin.w : oldv = 0x22334455
amomax.w : oldv = 0xa
amominu.w : oldv = 0xa
amomaxu.w : oldv = 0xa
Atomic instruction PASS!
```

21 编译器内建函数

在编译器中，内建函数是在给定编程语言中可用的函数，其实现由编译器特定处理。如果一个函数是内建函数，则此函数的代码通常会被内联插入，避免了函数调用的开销，并且可以为此函数发起高效的机器指令。

RDS 软件的编译器内建函数是提供给不想使用汇编编程的用户（包括操作系统工程师）提供，包含了编译器无法直接产生的所有操作。

21.1 内建函数定义

21.1.1 RV32“I”内建函数

RV32 基本整数“I”指令的内建函数，其函数原型、别名、所映射的指令和可调度性（编译器是否可调度该函数）如表 21-1 所示。

表 21-1 RV32“I”指令的内建函数

函数原型	别名	映射指令	可调度性
void __nds__fence(const unsigned int PIORW, const unsigned int SIORW)	-	fence	No
viod __nds__fencei(void)	-	fence.i	No
long __nds__ecall(long ID)	-	ecall	No
long __nds__ecall1(long ID, long ARG1)	-	ecall	-
long __nds__ecall2(long ID, long ARG1, long ARG2)	-	ecall	-
long __nds__ecall3(long ID, long ARG1, long ARG2, long ARG3)	-	ecall	-
long __nds__ecall4(long ID, long ARG1, long ARG2, long ARG3, long ARG4)	-	ecall	-
long __nds__ecall5(long ID, long ARG1, long ARG2, long ARG3, long ARG4,	-	ecall	-

函数原型	别名	映射指令	可调度性
long ARG5)			
long __nds__ecall6(long ID, long ARG1, long ARG2, long ARG3, long ARG4, long ARG5, long ARG6)	-	ecall	-
void __nds__ebreak(unsigned long ARG)	-	ebreak	No
unsigned long __nds__csrrw(unsigned long SRC, const unsigned int CSRN)	-	csrrw csrrwi	No
unsigned long __nds__swap_csr(unsigned long VAL, const unsigned int CSRN)	__nds__csrrw(VAL, CSRN)	csrrw csrrwi	No
unsigned long __nds__csrrs(unsigned long SRC, const unsigned int CSRN)	-	csrrs csrrsi	No
unsigned long __nds__read_and_set_csr(unsigned long SETMASK, const unsigned int CSRN)	__nds__csrrs(SETMASK, CSRN)	csrrs csrrsi	No
unsigned long __nds__csrrc(unsigned long SRC, const unsigned int CSRN)	-	csrrc csrrci	No
unsigned long __nds__read_and_clear_csr(unsigned long CLRMASK, const unsigned int CSRN)	__nds__csrrc(CLRMASK, CSRN)	csrrc csrrci	No
unsigned long __nds__csrr(const unsigned int CSRN)	-	csrr	No
unsigned long __builtin_riscv_csrr(unsigned long CSRN)	__nds__csrr(CSRN)	csrr	No
unsigned long __nds__mfsr(unsigned long CSRN)	__nds__csrr(CSRN)	csrr	No
unsigned long __nds__read_csr(const unsigned int CSRN)	__nds__csrr(CSRN)	csrr	No
void __nds__csrw(unsigned long SRC, const unsigned int CSRN)	-	csrw csrwi	No
void __builtin_riscv_csrw(unsigned long SRC, unsigned long CSRN)	__nds__csrw(SRC, CSRN)	csrw csrwi	No

函数原型	别名	映射指令	可调度性
void __nds__mtsr(unsigned long SRC, unsigned long CSRN)	__nds__csrw(SRC, CSRN)	csrw csrwi	No
void __nds__write_csr(unsigned long VAL, const unsigned int CSRN)	__nds__csrw(CSRN)	csrw csrwi	No
void __nds__csrs(unsigned long SRC, const unsigned int CSRN)	-	csrs csrsi	No
void __nds__set_csr_bits(unsigned long SETMASK, const unsigned int CSRN)	__nds__csrs(SETMASK, CSRN)	csrs csrsi	No
void __nds__csrc(unsigned long SRC, const unsigned int CSRN)	-	csrc csrci	No
void __nds__clear_csr_bits(unsigned long CLRMASK, const unsigned int CSRN)	__nds__csrc(CLRMASK, CSRN)	csrc csrci	No
unsigned long __nds__get_current_sp()	-	-	No
unsigned long __builtin_riscv_get_current_sp()	__nds__get_current_sp()	-	No
void __nds__set_current_sp(unsigned long VALUE)	-	-	No
unsigned long __builtin_riscv_set_current_sp(unsigned long VALUE)	__nds__set_current_sp(VALUE)	-	No

21.1.2 RV32“F”和“D”内建函数

RV32 浮点型扩展“F”和“D”指令的内建函数，其函数原型、别名、所映射的指令和可调度性（编译器是否可调度该函数）如表 21-2 所示。

表 21-2 RV32“F”和“D”指令的内建函数

函数原型	别名	映射指令	可调度性
unsigned long __nds__frcsr()	-	frcsr	No
unsigned long __nds__fscsr(unsigned long SRC)	-	fscsr	No
void __nds__fwcsr(unsigned long SRC)	-	fscsr	No
unsigned long __nds__frrm()	-	frrm	No
unsigned long	-	fsrm	No

函数原型	别名	映射指令	可调度性
__nds__fsrm(unsigned long SRC)		fsrmi	
void __nds__fwrm(unsigned long SRC)	-	fsrm fsrmi	No
unsigned long __nds__frflags()	-	frflags	No
unsigned long __builtin_riscv_frflags()	__nds__frflags()	frflags	No
unsigned long __nds__fsflags(unsigned long SRC)	-	fsflags fsflagsi	No
unsigned long __builtin_riscv_fsflags(unsigned long SRC)	__nds__fsflags(SRC)	fsflags fsflagsi	No
void __nds__fwflags(unsigned long SRC)	-	fsflags fsflagsi	No
float __nds_fcvt_s_bf16(__bf16 BF16_SRC)	-	fcvt.s.bf16	Yes
__bf16 __nds_fcvt_bf16_s(float FP32_SRC)	-	fcvt.bf16.s	Yes

21.1.3 RV32“A”内建函数

RV32 原子扩展“A”指令的内建函数，其函数原型、所映射的指令和可调度性（编译器是否可调度该函数）如表 21-3 所示。

表 21-3 RV32“A”指令的内建函数

函数原型	映射指令	可调度性
signed int __nds__amoswapw(signed int SRC, signed int* BASE, const unsigned int ORDERING)	amoswap.w	No
signed int __nds__amoaddw(signed int SRC, signed int* BASE, const unsigned int ORDERING)	amoadd.w	No
unsigned int __nds__amoxorw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)	amoxor.w	No
unsigned int __nds__amoandw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)	amoand.w	No

函数原型	映射指令	可调度性
unsigned int __nds__amoorw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)	amoor.w	No
signed int __nds__amominw(signed int SRC, signed int* BASE, const unsigned int ORDERING)	amomin.w	No
signed int __nds__amomaxw(signed int SRC, signed int* BASE, const unsigned int ORDERING)	amomax.w	No
unsigned int __nds__amominuw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)	amominu.w	No
unsigned int __nds__amomaxuw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)	amomaxu.w	No

21.2 内建函数描述

以下各节详细描述编译器内建函数的定义。

21.2.1 RV32“I”内建函数

__nds__fence

__nds__fence 函数的详细描述如表 21-4 所示。

表 21-4 __nds_fence

原型	void __nds__fence(const unsigned int PIORW, const unsigned int SIORW)
描述	This intrinsic inserts a fence instruction into the instruction stream. The constant operands of PIORW and SIORW are defined in the following Table 21-5.
返回值	None
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//Make sure “valid” is set after “data” is written.
	unsigned int valid = 0;
	unsigned int data;
	data = 1000;

	__nds__fence(FENCE_W, FENCE_W);
	valid = 1;
	}

表 21-5 PIORW and SIORW

unsigned int IORW	Value
FENCE_W	1
FENCE_R	2
FENCE_RW	3
FENCE_O	4
FENCE_OW	5
FENCE_OR	6
FENCE_ORW	7
FENCE_I	8
FENCE_IW	9
FENCE_IR	10
FENCE_IRW	11
FENCE_IO	12
FENCE_IOW	13
FENCE_IOR	14
FENCE_IORW	15

__nds__fencei

__nds__fencei 函数的详细描述如表 21-6 所示。

表 21-6 __nds_fencei

原型	viod __nds__fencei(void)
描述	This intrinsic inserts a fence.i instruction into the instruction stream.
返回值	None
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//Self-modify program code.

	unsigned int *code;
	code = 0x20000;
	*code = 0x12345678;
	__nds__fencei();
	}

__nds__ecall, __nds__ecall[1..6]

__nds__ecall, __nds__ecall[1..6]函数的详细描述如表 21-7 所示。

表 21-7__nds_ecall, __nds_ecall[1..6]

原型	long __nds__ecall(long ID) long __nds__ecall1(long ID, long ARG1) long __nds__ecall2(long ID, long ARG1, long ARG2) long __nds__ecall3(long ID, long ARG1, long ARG2, long ARG3) long __nds__ecall4(long ID, long ARG1, long ARG2, long ARG3, long ARG4) long __nds__ecall5(long ID, long ARG1, long ARG2, long ARG3, long ARG4, long ARG5) long __nds__ecall6(long iD, long ARG1, long ARG2, long ARG3, long ARG4, long ARG5, long ARG6)
描述	This intrinsic inserts an ecall instruction into the instruction stream.
返回值	Return value for ecall ID.
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//Insert an “ecall” instruction.
	long status;
	status = __nds__ecall(0x12341234);
	}

__nds__ebreak

__nds__ebreak 函数的详细描述如表 21-8 所示。

表 21-8 __nds_ebreak

原型	void __nds__ebreak(unsigned long ARG)
描述	This intrinsic inserts an ebreak instruction into the instruction stream.

返回值	None
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	  //Insert an “ebreak” instruction.
	  __nds__ebreak(0x98761234);
	}

__nds__csrrw

__nds__csrrw 函数的详细描述如表 21-9 所示。

表 21-9 __nds_csrrw

原型	unsigned long __nds__csrrw(unsigned long SRC, const unsigned int CSRN)
别名	unsigned long __nds__swap_csr(unsigned long VAL, const unsigned int CSRN)
描述	This intrinsic inserts a csrrw or csrrwi instruction into the instruction stream. The content of the source operand SRC will be written into the CSR csrn. The original content of the CSR csrn will be returned.
返回值	The original content of the CSR csrn
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	  //Read and write “mie” CSR.
	  unsigned int src, mie_data;
	  src = 0x888;
	  mie_data = __nds__csrrw(src, 0x304);
	}

__nds__csrrs

__nds__csrrs 函数的详细描述如表 21-10 所示。

表 21-10 __nds_csrrs

原型	unsigned long __nds__csrrs(unsigned long SRC, const unsigned int CSRN)

别名	unsigned long __nds__read_and_set_csr(unsigned long SETMASK, const unsigned int CSRN)
描述	This intrinsic inserts a csrrs or csrrsi instruction into the instruction stream. The content of the source operand SRC will be used to set bits in the CSR csrn. The original content of the CSR csrn will be returned.
返回值	The original content of the CSR csrn
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//Read and set bits in “mie” CSR.
	unsigned int src, mie_data;
	src = 0x888;
	mie_data = __nds__csrrs(src, 0x304);
	}

__nds__csrrc

__nds__csrrc 函数的详细描述如表 21-11 所示。

表 21-11 __nds_csrrc

原型	unsigned long __nds__csrrc(unsigned long SRC, const unsigned int CSRN)
别名	unsigned long __nds__read_and_clear_csr(unsigned long CLRMASK, const unsigned int CSRN)
描述	This intrinsic inserts a csrrc or csrrci instruction into the instruction stream. The content of the source operand SRC will be used to clear bits in the CSR csrn. The original content of the CSR csrn will be returned.
返回值	The original content of the CSR
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//Read and clear bits in “mie” CSR.
	unsigned int src, mie_data;
	src = 0x888;

	mie_data = __nds__csrrs(src, 0x304); }

__nds__csrr

__nds__csrr 函数的详细描述如表 21-12 所示。

表 21-12 __nds_csrr

原型	unsigned long __nds__csrr(const unsigned int CSRN)
别名	unsigned long __builtin_riscv_csrr(unsigned long CSRN) unsigned long __nds__mfsr(unsigned long CSRN) unsigned long __nds__read_csr(const unsigned int CSRN)
描述	This intrinsic inserts a csrr or csrri pseudo-instruction into the instruction stream. The original content of the CSR csrn will be returned.
返回值	The original content of the CSR
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//Read “mie” CSR.
	unsigned int mie_data;
	mie_data = __nds__csrr(0x304);
	}

__nds__csrw

__nds__csrw 函数的详细描述如表 21-13 所示。

表 21-13 __nds_csrw

原型	void __nds__csrw(unsigned long SRC, const unsigned int CSRN)
别名	void __builtin_riscv_csrw(unsigned long SRC, unsigned long CSRN) void __nds__mtsr(unsigned long SRC, unsigned long CSRN) void __nds__write_csr(unsigned long VAL, const unsigned int CSRN)
描述	This intrinsic inserts a csrw or csrwi pseudo-instruction into the instruction stream. The content of the source operand SRC will be written into the CSR csrn.
返回值	None
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	  //Write “mie” CSR.
	  unsigned int src;
	  src = 0x888;
	  __nds__csrw(src, 0x304);
	}

__nds__csrs

__nds__csrs 函数的详细描述如表 21-14 所示。

表 21-14 __nds_csrs

原型	void __nds__csrs(unsigned long SRC, const unsigned int CSRN)
别名	void __nds__set_csr_bits(unsigned long SETMASK, const unsigned int CSRN)
描述	This intrinsic inserts a csrs or csrsi pseudo-instruction into the instruction stream. The content of the source operand SRC will be used to set bits in the CSR csrn.
返回值	None
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	  //Set bits in “mie” CSR.
	  unsigned int src;
	  src = 0x888;
	  __nds__csrs(src, 0x304);
	}

__nds__csrc

__nds__csrc 函数的详细描述如表 21-15 所示。

表 21-15 __nds_csrc

原型	void __nds__csrc(unsigned long SRC, const unsigned int CSRN)
别名	void __nds__clear_csr_bits(unsigned long CLRMASK, const unsigned int CSRN)

描述	This intrinsic inserts a csrc or csrci pseudo-instruction into the instruction stream. The content of the source operand SRC will be used to clear bits in the CSR csrn.
返回值	None
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//Clear bits in “mie” CSR.
	unsigned int src;
	src = 0x888;
	__nds__csrc(src, 0x304);
	}

__nds__get_current_sp

__nds__get_current_sp 函数的详细描述如表 21-16 所示。

表 21-16 __nds_get_current_sp

原型	unsigned long __nds__get_current_sp()
别名	unsigned long __builtin_riscv_get_current_sp()
描述	This intrinsic returns the content of the stack pointer register.
返回值	The current stack pointer register value.
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	unsigned long spval;
	spval = __nds__get_current_sp();
	}

__nds__set_current_sp

__nds__set_current_sp 函数的详细描述如表 21-17 所示。

表 21-17 __nds_set_current_sp

原型	void __nds__set_current_sp(unsigned long VALUE)
别名	unsigned long __builtin_riscv_set_current_sp(unsigned long VALUE)

描述	This intrinsic sets the content of the stack pointer register.
返回值	None
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	  unsigned long spval;
	  spval = 0x80000
	  __nds__set_current_sp(spval);
	}

21.2.2 RV32“F”和“D”内建函数

__nds__frcsr

__nds__frcsr 函数的详细描述如表 21-18 所示。

表 21-18__nds_frcsr

原型	unsigned long __nds__frcsr()
描述	This intrinsic inserts a frcsr pseudo-instruction into the instruction stream. The content of the CSR fcsr will be returned.
返回值	The content of the CSR fcsr
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	  //Read “fcsr” CSR.
	  unsigned int fcsr_data;
	  fcsr_data = __nds__frcsr();
	}

__nds__fscsr

__nds__fscsr 函数的详细描述如表 21-19 所示。

表 21-19 __nds_fscsr

原型	unsigned long __nds__fscsr(unsigned long SRC)
描述	This intrinsic inserts a fscsr pseudo-instruction into the instruction stream. The content of the source operand SRC will be written into the

	CSR fcsr. The original content of the CSR fcsr will be returned.
返回值	The original content of the CSR fcsr
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//Read and write “fcsr” CSR.
	unsigned int src, fcsr_data;
	src = 0x60;
	fcsr_data = __nds__fscsr(src);
	}

__nds__fwcsr

__nds__fwcsr 函数的详细描述如表 21-20 所示。

表 21-20 __nds_fwcsr

原型	void __nds__fwcsr(unsigned long SRC)
描述	This intrinsic inserts a “fscsr rs” pseudo-instruction into the instruction stream. The content of the source operand SRC will be written into the CSR fcsr.
返回值	None
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//Write “fcsr” CSR.
	unsigned int src;
	src = 0x60;
	__nds__fwcsr(src);
	}

__nds__frrm

__nds__frrm 函数的详细描述如表 21-21 所示。

表 21-21 __nds_frrm

原型	unsigned long __nds__frrm()
描述	This intrinsic inserts a frrm pseudo-instruction into the instruction

	stream. The content of the CSR field fcsr.FRM will be returned.
返回值	The content of the CSR field fcsr.FRM
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	  //Read “fcsr.FRM” CSR field.
	  unsigned int frm_data;
	  frm_data = __nds__frrm();
	}

__nds__fsrm

__nds__fsrm 函数的详细描述如表 21-22 所示。

表 21-22 __nds_fsrm

原型	unsigned long __nds__fsrm(unsigned long SRC)
描述	This intrinsic inserts a fsrm or fsrmi pseudo-instruction into the instruction stream. The content of the source operand SRC will be written into the CSR field fcsr.FRM. The original content of the CSR field fcsr.FRM will be returned.
返回值	The original content of the CSR field fcsr.FRM
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	  //Read and write “fcsr.FRM” CSR field.
	  unsigned int src, frm_data;
	  src = 0x3;
	  frm_data = __nds__fsrm(src);
	}

__nds__fwrm

__nds__fwrm 函数的详细描述如表 21-23 所示。

表 21-23 __nds_fwrm

原型	void __nds__fwrm(unsigned long SRC)
描述	This intrinsic inserts a “fsrm rs” or “fsrmi imm” pseudo-instruction into

	the instruction stream. The content of the source operand SRC will be written into the CSR field fcsr.FRM.
返回值	None
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//Write “fcsr.FRM” CSR field.
	unsigned int src;
	src = 0x3;
	__nds__fwrm(src);
	}

__nds__frflags

__nds__frflags 函数的详细描述如表 21-24 所示。

表 21-24 __nds_frflags

原型	unsigned long __nds__frflags()
别名	unsigned long __builtin_riscv_frflags()
描述	This intrinsic inserts a frflags pseudo-instruction into the instruction stream. The content of the CSR field fcsr.FFLAGS will be returned.
返回值	The content of the CSR field fcsr.FFLAGS.
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//Read “fcsr.FFLAGS” CSR field.
	unsigned int flags_data;
	flags_data = __nds__frflags();
	}

__nds__fsflags

__nds__fsflags 函数的详细描述如表 21-25 所示。

表 21-25 __nds_fsflags

原型	unsigned long __nds__fsflags(unsigned long SRC)

别名	unsigned long __builtin_riscv_fsflags(unsigned long SRC)
描述	This intrinsic inserts a fsflags or fsflagsi pseudo-instruction into the instruction stream. The content of the source operand SRC will be written into the CSR field fcsr.FFLAGS. The original content of the CSR field fcsr.FFLAGS will be returned.
返回值	The original content of the CSR field fcsr.FFLAGS.
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//Read and write “fcsr.FFLAGS” CSR field.
	unsigned int src, flags_data;
	src = 0x0;
	flags_data = __nds__fsflags(src);
	}

__nds__fwflags

__nds__fwflags 函数的详细描述如表 21-26 所示。

表 21-26 __nds_fwflags

原型	void __nds__fwflags(unsigned long SRC)
描述	This intrinsic inserts a “fsflags rs” or “fsflagsi imm” pseudo-instruction into the instruction stream. The content of the source operand SRC will be written into the CSR field fcsr.FFLAGS.
返回值	None
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//Write “fcsr.FFLAGS” CSR field.
	unsigned int src;
	src = 0x0;
	__nds__fwflags(src);
	}

__nds_fcvt_s_bf16

__nds_fcvt_s_bf16 函数的详细描述如表 21-27 所示。

表 21-27 __nds_fcvt_s_bf16

原型	float __nds_fcvt_s_bf16(__bf16 BF16_SRC)
描述	This intrinsic converts BFLOAT16 data to single-precision floating-point (SP) data.
返回值	float
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	__bf16 bf_input;
	float sf_result;
	bf_input = __nds_fcvt_bf16_s(sf_result);
	sf_result = __nds_fcvt_s_bf16(bf_input);
	}

__nds_fcvt_bf16_s

__nds_fcvt_bf16_s 函数的详细描述如表 21-28 所示。

表 21-28 __nds_fcvt_bf16_s

原型	__bf16 __nds_fcvt_bf16_s(float FP32_SRC)
描述	This intrinsic converts single-precision floating-point (SP) data to BFLOAT16 data.
返回值	BFLOAT16
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	__bf16 bf_input;
	float sf_result;
	bf_input = __nds_fcvt_bf16_s(sf_result);
	sf_result = __nds_fcvt_s_bf16(bf_input);
	}

21.2.3 RV32“A”内建函数

__nds__amoswapw

__nds__amoswapw 函数的详细描述如表 21-29 所示。

表 21-29 __nds_amoswapw

原型	signed int __nds__amoswapw(signed int SRC, signed int* BASE, const unsigned int ORDERING)
描述	This intrinsic inserts an AMOSWAP.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
返回值	The content of the memory address BASE
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic swap operation.
	signed int data, newv, oldv;
	newv = 10;
	// new value: 10
	oldv = __nds__amoswapw(newv, &data, UNORDER);
	}

表 21-30 ORDERING

unsigned int ORDERING	Value
UNORDER	0
RELEASE	1
ACQUIRE	2
SEQUENTIAL	3

__nds__amoaddw

__nds__amoaddw 函数的详细描述如表 21-31 所示。

表 21-31 __nds_amoaddw

原型	signed int __nds__amoaddw(signed int SRC, signed int* BASE, const unsigned int ORDERING)

描述	This intrinsic inserts an AMOADD.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
返回值	The content of the memory address BASE
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic add operation.
	signed int data, addv, oldv;
	addv = 10;
	// new value: data + 10
	oldv = __nds__amoaddw(addv, &data, UNORDER);
	}

__nds__amoxorw

__nds__amoxorw 函数的详细描述如表 21-32 所示。

表 21-32 __nds_amoxorw

原型	unsigned int __nds__amoxorw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)
描述	This intrinsic inserts an AMOXOR.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
返回值	The content of the memory address BASE
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic xor operation.
	unsigned int data, xorv, oldv;
	xorv = 0x22334455;
	// new value: data xor 0x22334455
	oldv = __nds__amoxorw(xorv, &data, UNORDER);
	}

__nds__amoandw

__nds__amoandw 函数的详细描述如表 21-33 所示。

表 21-33 __nds_amoandw

原型	unsigned int __nds__amoandw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)
描述	This intrinsic inserts an AMOAND.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
返回值	The content of the memory address BASE
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic AND operation.
	unsigned int data, andv, oldv;
	andv = 0x22334455;
	// new value: data AND 0x22334455
	oldv = __nds__amoandw(andv, &data, UNORDER);
	}

__nds__amoorw

__nds__amoorw 函数的详细描述如表 21-34 所示。

表 21-34 __nds_amoorw

原型	unsigned int __nds__amoorw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)
描述	This intrinsic inserts an AMOOR.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
返回值	The content of the memory address BASE
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic or operation.

	unsigned int data, orv, oldv;
	orv = 0x22334455;
	// new value: data OR 0x22334455
	oldv = __nds__amoorw(orv, &data, UNORDER);
	}

__nds__amominw

__nds__amominw 函数的详细描述如表 21-35 所示。

表 21-35 __nds_amominw

原型	signed int __nds__amominw(signed int SRC, signed int* BASE, const unsigned int ORDERING)
描述	This intrinsic inserts an AMOMIN.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
返回值	The content of the memory address BASE
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic min operation.
	signed int data, cmpv, oldv;
	cmpv = 10;
	// new value: minimum(data, cmpv)
	oldv = __nds__amominw(cmpv, &data, UNORDER);
	}

__nds__amomaxw

__nds__amomaxw 函数的详细描述如表 21-36 所示。

表 21-36 __nds_amomaxw

原型	signed int __nds__amomaxw(signed int SRC, signed int* BASE, const unsigned int ORDERING)
描述	This intrinsic inserts an AMOMAX.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30

返回值	The content of the memory address BASE
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic max operation.
	signed int data, cmpv, oldv;
	cmpv = 10;
	// new value: maximum(data, cmpv)
	oldv = __nds__amomaxw(cmpv, &data, UNORDER);
	}

__nds__amominuw

__nds__amominuw 函数的详细描述如表 21-37 所示。

表 21-37 __nds_amominuw

原型	unsigned int __nds__amominuw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)
描述	This intrinsic inserts an AMOMINU.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
返回值	The content of the memory address BASE
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic unsigned min operation.
	unsigned int data, cmpv, oldv;
	cmpv = 0x22334455;
	// new value: unsigned minimum(data, cmpv)
	oldv = __nds__amominuw(cmpv, &data, UNORDER);
	}

__nds__amomaxuw

__nds__amomaxuw 函数的详细描述如表 21-38 所示。

表 21-38 __nds_amomaxuw

原型	unsigned int __nds__amomaxuw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)
描述	This intrinsic inserts an AMOMAXU.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
返回值	The content of the memory address BASE
特权等级	ALL

示例	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic unsigned min operation.
	unsigned int data, cmpv, oldv;
	cmpv = 0x22334455;
	// new value: unsigned maximum(data, cmpv)
	oldv = __nds__amomaxuw(cmpv, &data, UNORDER);
	}

22 PLIC/PLIC_SW 内建函数

RDS 软件的编译器提供访问 PLIC/PLIC_SW 的内建函数，宏定义符号“NDS_PLIC_BASE”和“NDS_PLIC_SW_BASE”用于开启 PLIC/PLIC_SW 内建函数。为激活 PLIC/PLIC_SW 内建函数，须在包含平台头文件前定义和分配 PLIC/PLIC_SW 基地址。

例如，

```
#define NDS_PLIC_BASE 0xE4000000
#define NDS_PLIC_SW_BASE 0xE6400000
#include <nds_v5_platform.h>
```

22.1 内建函数定义

PLIC/PLIC_SW 内建函数的定义如下所示。

函数原型

void __nds__plic_set_feature (unsigned int FEATURE)

void __nds__plic_set_threshold (unsigned int THRESHOLD)

void __nds__plic_set_priority (unsigned int SOURCE, unsigned int PRIORITY)

void __nds__plic_set_pending (unsigned int SOURCE)

void __nds__plic_enable_interrupt (unsigned int SOURCE)

void __nds__plic_disable_interrupt (unsigned int SOURCE)

unsigned int __nds__plic_claim_interrupt(void)

void __nds__plic_complete_interrupt(unsigned int SOURCE)

```
void __nds__plic_sw_set_threshold (unsigned int THRESHOLD)

void __nds__plic_sw_set_priority (unsigned int SOURCE, unsigned int PRIORITY)

void __nds__plic_sw_set_pending (unsigned int SOURCE)

void __nds__plic_sw_enable_interrupt (unsigned int SOURCE)

void __nds__plic_sw_disable_interrupt (unsigned int SOURCE)

unsigned int __nds__plic_sw_claim_interrupt(void)

void __nds__plic_sw_complete_interrupt(unsigned int SOURCE)
```

22.2 内建函数描述

22.2.1 PLIC 内建函数

以下各节详细描述 PLIC 内建函数的定义。

__nds__plic_set_feature

__nds__plic_set_feature 函数的详细描述如表 22-1 所示。

表 22-1 __nds_plic_set_feature

原型	void __nds__plic_set_feature (unsigned int FEATURE)
描述	This function sets the feature for PLIC.
参数	FEATURE is the feature that will be enabled for PLIC. It includes the following options: NDS_PLIC_FEATURE_PREEMPT to enable priority-based preemption NDS_PLIC_FEATURE_VECTORED to enable vectored mode for PLIC
返回值	None

__nds__plic_set_threshold

__nds__plic_set_threshold 函数的详细描述如表 22-2 所示。

表 22-2 __nds_plic_set_threshold

原型	void __nds__plic_set_threshold (unsigned int THRESHOLD)
描述	This function sets the interrupt priority threshold for PLIC.
参数	THRESHOLD is the interrupt priority threshold for PLIC. The greater the value is, the higher the threshold is.

返回值	None

__nds__plic_set_priority

__nds__plic_set_priority 函数的详细描述如表 22-3 所示。

表 22-3 __nds_plic_set_priority

原型	void __nds__plic_set_priority (unsigned int SOURCE, unsigned int PRIORITY)
描述	This function sets the priority level for the specified PLIC interrupt.
参数	SOURCE is the ID number of the specified interrupt. PRIORITY is the priority level.
返回值	None

__nds__plic_set_pending

__nds__plic_set_pending 函数的详细描述如表 22-4 所示。

表 22-4 __nds_plic_set_pending

原型	void __nds__plic_set_pending (unsigned int SOURCE)
描述	This function sets the pending status for the specified PLIC interrupt.
参数	SOURCE is the IDnumber of the specified interrupt.
返回值	None

__nds__plic_enable_interrupt

__nds__plic_enable_interrupt 函数的详细描述如表 22-5 所示。

表 22-6 __nds_plic_enable_interrupt

原型	void __nds__plic_enable_interrupt (unsigned int SOURCE)
描述	This function enables the specified PLIC interrupt.
参数	SOURCE is the ID number of the specified interrupt.
返回值	None

__nds__plic_disable_interrupt

__nds__plic_disable_interrupt 函数的详细描述如表 22-6 所示。

表 22-5 __nds_plic_disable_interrupt

原型	void __nds__plic_disable_interrupt (unsigned int SOURCE)

描述	This function disables the specified PLIC interrupt.
参数	SOURCE is the ID number of the specified interrupt.
返回值	None

__nds__plic_claim_interrupt

__nds__plic_claim_interrupt 函数的详细描述如表 22-6 所示。

表 22-6 __nds_plic_claim_interrupt

原型	unsigned int __nds__plic_claim_interrupt(void)
描述	This function gets the ID number of the claimed interrupt.
返回值	The ID number of the claimed interrupt, which is read from PLIC

__nds__plic_complete_interrupt

__nds__plic_complete_interrupt 函数的详细描述如表 22-7 所示。

表 22-7 __nds_plic_complete_interrupt

原型	void __nds__plic_complete_interrupt(unsigned int SOURCE)
描述	SOURCE is the ID number of the specified interrupt.
参数	This function sets the specified interrupt as completed so that the next such interrupt can be forwarded to the PLIC and processed.
返回值	None

22.2.2 PLIC_SW 内建函数

以下各节详细描述 PLIC_SW 内建函数的定义。

__nds__plic_sw_set_threshold

__nds__plic_sw_set_threshold 内建函数的详细描述如表 22-8 所示。

表 22-8 __nds_plic_sw_set_threshold

原型	void __nds__plic_sw_set_threshold (unsigned int THRESHOLD)
描述	This function set the interrupt priority threshold for PLIC_SW.
参数	THRESHOLD is the interrupt priority threshold for PLIC_SW. The larger the value is, the higher the threshold is.
返回值	None

__nds__plic_sw_set_priority

__nds__plic_sw_set_priority 函数的详细描述如表 22-9 所示。

表 22-9 __nds_plic_sw_set_priority

原型	void __nds__plic_sw_set_priority (unsigned int SOURCE, unsigned int PRIORITY)
描述	This function sets the priority level for the specified PLIC_SW interrupt.
参数	SOURCE is the ID number of the specified interrupt. PRIORITY is the priority level.
返回值	None

__nds__plic_sw_set_pending

__nds__plic_sw_set_pending 函数的详细描述如表 22-10 所示。

表 22-10 __nds_plic_sw_set_pending

原型	void __nds__plic_sw_set_pending (unsigned int SOURCE)
描述	This function sets the pending status for the specified PLIC_SWinterrupt.
参数	SOURCE is the ID number of the specified interrupt.
返回值	None

__nds__plic_sw_enable_interrupt

__nds__plic_sw_enable_interrupt 函数的详细描述如表 22-11 所示。

表 22-11 __nds_plic_sw_enable_interrupt

原型	void __nds__plic_sw_enable_interrupt (unsigned int SOURCE)
描述	This function enables the specified PLIC_SW interrupt.
参数	SOURCE is the ID number of the specified interrupt.
返回值	None

__nds__plic_sw_disable_interrupt

__nds__plic_sw_disable_interrupt 函数的详细描述如表 22-12 所示。

表 22-12 __nds_plic_sw_disable_interrupt

原型	void __nds__plic_sw_disable_interrupt (unsigned int SOURCE)
描述	This function disables the specified PLIC_SW interrupt.
参数	SOURCE is the ID number of the specified interrupt.

返回值	None

__nds__plic_sw_claim_interrupt

__nds__plic_sw_claim_interrupt 函数的详细描述如表 22-13 所示。

表 22-13 __nds_plic_sw_claim_interrupt

原型	unsigned int __nds__plic_sw_claim_interrupt(void)
描述	This function gets the ID number of the claimed interrupt.
返回值	The ID number of the claimed interrupt, which is read from PLIC_SW

__nds__plic_sw_complete_interrupt

__nds__plic_sw_complete_interrupt 函数的详细描述，如表 22-14 所示。

表 22-14 __nds_plic_sw_complete_interrupt

原型	void __nds__plic_sw_complete_interrupt(unsigned int SOURCE)
描述	This function sets the specified interrupt as completed so that the next such interrupt can be forwarded to the PLIC_SW and processed.
参数	SOURCE is the ID number of the specified interrupt.
返回值	None


GOWIN高云
智慧逻辑 定制未来