



Gowin 原语 用户指南

SUG283-3.1, 2023-07-14

版权所有 © 2023 广东高云半导体科技股份有限公司

GOWIN高云、、Gowin、GowinSynthesis、小蜜蜂、晨熙、云源以及高云均为广东高云半导体科技股份有限公司注册商标，本手册中提到的其他任何商标，其所有权利属其拥有者所有。未经本公司书面许可，任何单位和个人都不得擅自摘抄、复制、翻译本文档内容的部分或全部，并不得以任何形式传播。

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，或以禁止发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	说明
2017/04/20	1.0	初始版本。
2017/09/19	1.1	● 增加支持器件系列 GW1NR-4、GW1N-6、GW1N-9、GW1NR-9; ● 增加 ELVDS_IOBUF、TLVDS_IOBUF、BUFG、BUFS、OSC、IEM; ● 更新 DSP 原语; ● 更新 ODDR/ODDR、IDDR_MEM、IDES4_MEM、IDES8_MEM、RAM16S1、RAM16S2、RAM16S4、RAM16SDP1、RAM16SDP2、RAM16SDP4、ROM16 部分 port 名称; ● 更新 OSC、PLL、DLLDL 部分 Attribute; ● 更新部分原语例化; ● 增加 MIPI_IBUF_HS、MIPI_IBUF_LP、MIPI_OBUF、IDES16、OSER16; ● 更新 CLKDIV 部分 Attribute。
2018/04/12	1.2	增加 vhdI 原语例化。
2018/08/08	1.3	● 增加支持器件系列 GW1N-2B、GW1N-4B、GW1NR-4B、GW1N-6ES、GW1N-9ES、GW1NR-9ES、GW1NS-2、GW1NS-2C; ● 增加 I3C_IOBUF、DHCEN; ● 增加 User Flash; ● 增加 EMPU; ● 更新原语名称。
2018/10/26	1.4	● 增加支持器件系列 GW1NZ-1、GW1NSR-2C; ● 增加 OSCZ、FLASH96KZ。
2018/11/15	1.5	● 增加支持器件系列 GW1NSR-2; ● 删除器件 GW1N-6ES、GW1N-9ES、GW1NR-9ES。
2019/01/26	1.6	● CLKDIV 的 8 分频新增支持 GW1NS-2 器件; ● 删除 TLVDS_TBUF/OBUF 支持器件中的 GW1N-1。
2019/02/25	1.7	删除 TLVDS_IOBUF 支持器件中的 GW1N-1。
2019/05/20	1.8	● 增加支持器件系列 GW1N-1S; ● 增加 MIPI_IBUF; ● 增加 OSCH; ● 增加 SPMI; ● 增加 I3C; ● 更新 OSC 的支持器件。
2019/10/20	1.9	更新 IOB、BSRAM、CLOCK 模块。
2019/11/28	2.0	● 增加 GSR、INV 等 Miscellaneous 模块; ● 更新支持器件信息; ● 增加 FLASH64KZ, 删除 FLASH96KZ。
2020/01/16	2.1	● 增加 IODELAYA、rPLL、PLLVR、CLKDIV2; ● 增加 DPB/DPX9B、SDPB/SDPX9B、rSDP/rSDPX9、rROM/rROMX9、pROM/pROMX9; ● 增加 EMCU、BANDGAP、FLASH64K;

日期	版本	说明
		● 更新 IODELAY、PLL、CLKDIV、OSC、DQCE; ● 增加 FF、LATCH 放置规则; ● 增加支持器件 GW2A-55C; ● GW1N-6/GW1N-9/GW1NR-9 禁掉 DP/DPX9、DPB/DPX9B; ● IOLOGIC 增加 register 说明备注; ● GW1NZ-1 禁掉 DP/DPB 的 1,2,4,8 位宽, DPX9/DPX9 的 9 位宽。
2020/03/09	2.2	● GW1NS-2、GW1NS-2C、GW1NSR-2、GW1NSR-2C、GW1NSE-2C 禁掉 DP/DPX9、DPB/DPX9B; ● OSCF 补充 OSCEN 端口说明; ● 更新 PLL/rPLL/PLLVR 参数说明。
2020/06/08	2.3	● 删除器件 GW1N-2、GW1N-2B、GW1N-6; ● 增加器件 GW1N-9C、GW1NR-9C; ● 增加 IODELAYC、DHCENC、DCC; ● 增加 MIPI_IBUF 功能描述。 ● 删除 MIPI_IBUF_HS、MIPI_IBUF_LP、DLL; ● 增加 LUT5、MUX8 端口示意图; ● 增加 VCC、GND; ● 更新 PLLVR、FLASH64K、BUFS、EMPU、CLKDIV2 原语介绍; ● 删除 DP/DPX9、ROM/ROMX9、SDP/SDPX9、rSDP/rSDPX9、rROM/rROMX9、PLL; ● 调整 Iologic 结构顺序, 统一端口示意图标题名称。
2020/09/11	2.4	添加 ADC, BANDGAP, SPMI, I3C 模块 IP 调用说明。
2020/12/30	2.5	更新第二章 CFU 描述。
2021/07/22	2.6	● 增加 activeFlash; ● 更新 IP 调用部分结构图, 去掉 help 内容; ● 增加器件 GW1NZ-1C、GW1N-2、GW1N-2B、GW1N-1P5、GW1N-1P5B、GW1NR-2、GW1NR-2B。
2021/10/28	2.7	更新 activeFlash 描述。
2022/07/22	2.8	● 新增 OTP 模块; ● 新增 SAMB 模块。
2022/10/28	2.9	删除 MCU、USB20_PHY、ADC。
2023/04/20	3.0	新增 Arora V 器件及相关描述。
2023/07/14	3.1	删除 SPMI 配置选项 Shutdown by VCCEN。

目录

目录..... i
 图目录 ii
 表目录 iii
1 IOB..... 1
2 CFU..... 2
3 Memory 3
4 DSP..... 4
5 Clock 5
6 User Flash..... 6
7 EMPU..... 7
 7.1 EMCU 7
8 其它..... 20
 8.1 GSR 20
 8.2 INV 21
 8.3 VCC 22
 8.4 GND 23
 8.5 BANDGAP 23
 8.6 SPMI 26
 8.7 I3C 31
 8.8 activeFlash..... 39
 8.9 OTP..... 40
 8.10 SAMB..... 43
 8.11 CM SER 46
 8.12 CMSERA 49

图目录

图 7-1 EMCU 端口示意图	8
图 8-1 GSR 端口示意图	20
图 8-2 INV 端口示意图	21
图 8-3 VCC 端口示意图	22
图 8-4 GND 端口示意图	23
图 8-5 BANDGAP 端口示意图	24
图 8-6 BandGap 的 IP Customization 窗口结构	25
图 8-7 SPMI 端口示意图	26
图 8-8 SPMI 的 IP Customization 窗口结构	29
图 8-9 I3C 端口示意图	31
图 8-10 I3C 的 IP Customization 窗口结构	38
图 8-11 activeFlash 端口示意图	39
图 8-12 GW2AN OTP 端口示意图	41
图 8-13 Arora V OTP 端口示意图	41
图 8-14 GW2AN SAMB 端口示意图	43
图 8-15 Arora V SAMB 端口示意图	43
图 8-16 CMSER 端口示意图	46
图 8-17 CMSERA 端口示意图	49

表目录

表 7-1 EMCU 适用器件	7
表 7-2 EMCU 端口介绍	9
表 8-1 GSR 端口介绍	20
表 8-2 INV 端口介绍	21
表 8-3 VCC 端口介绍	22
表 8-4 GND 端口介绍	23
表 8-5 BANDGAP 适用器件	24
表 8-6 BANDGAP 端口介绍	24
表 8-7 SPMI 适用器件	26
表 8-8 SPMI 端口介绍	27
表 8-9 I3C 适用器件	31
表 8-10 I3C 端口介绍	32
表 8-11 activeFlash 适用器件	39
表 8-12 activeFlash 端口介绍	39
表 8-13 OTP 适用器件	40
表 8-14 GW2AN OTP 端口介绍	41
表 8-15 Arora V OTP 端口介绍	41
表 8-16 Arora V OTP 参数介绍	41
表 8-17 SAMB 适用器件	43
表 8-18 GW2AN SAMB 端口介绍	44
表 8-19 Arora V SAMB 端口介绍	44
表 8-20 Arora V SAMB 参数介绍	44
表 8-21 CMSER 适用器件	46
表 8-22 CMSER 端口介绍	46
表 8-23 CMSERA 适用器件	49
表 8-24 CMSERA 端口介绍	49

1 IOB

IOB 主要包括输入输出缓存 (IO Buffer)、输入输出逻辑 (IO Logic)，其中 Arora V 器件的 IO Buffer 及 IO Logic 原语可参考 [UG304, Arora V 可编程通用管脚 \(GPIO\) 用户指南](#)，其他器件可参考 [UG289, Gowin 可编程通用管脚 \(GPIO\) 用户指南](#)。

2 CFU

可配置功能单元（**CFU**）和可配置逻辑单元（**CLU**）是构成高云半导体 **FPGA** 产品内核的两种基本单元，每个基本单元可由四个可配置逻辑块（**CLS**）以及相应的可配置布线单元（**CRU**）组成。**CLU** 中的可配置逻辑块不能配置为静态随机存储器，可配置为基本查找表、算术逻辑单元和只读存储器。**CFU** 中的可配置逻辑块可根据应用场景配置成基本查找表、算术逻辑单元、静态随机存储器和只读存储器四种工作模式。**Arora V** 器件的 **CFU** 原语可参考 [UG303, Arora V 可配置功能单元\(CFU\)用户指南](#)，其他器件可参考 [UG288, Gowin 可配置功能单元（CFU）用户指南](#)。

3 Memory

高云半导体 **FPGA** 产品提供了丰富的存储器资源，包括块状静态随机存储器（**BSRAM**）和分布式静态随机存储器（**SSRAM**）。Arora V器件的 **BSRAM/SSRAM** 原语可参考 [UG300, Arora V 存储器 \(BSRAM & SSRAM\) 用户指南](#)，其他器件可参考 [UG285, Gowin 存储器 \(BSRAM & SSRAM\) 用户指南](#)。

4 DSP

高云半导体 FPGA 产品具有丰富的 DSP 资源，可满足用户对高性能数字信号的处理需求。Arora V 器件的 DSP 原语可参考 [UG305, Arora V 数字信号处理 \(DSP\) 模块用户指南](#)，其他器件可参考 [UG287, Gowin 数字信号处理 \(DSP\) 模块用户指南](#)。

5 Clock

高云半导体 **FPGA** 产品提供了专用全局时钟网络（**GCLK**，包括 **PCLK** 和 **SCLK**），直接连接到器件的所有资源。除了 **GCLK** 资源，还提供了锁相环（**PLL**）、高速时钟 **HCLK** 和 **DDR** 存储器接口数据脉冲时钟 **DQS** 等时钟资源。**Arora V**器件的 **CLOCK** 原语可参考 [UG306, Arora V 时钟资源 \(Clock\) 用户指南](#)，其他器件可参考 [UG286, Gowin 时钟资源 \(Clock\) 用户指南](#)。

6 User Flash

Gowin 小蜜蜂®家族 FPGA 产品提供用户闪存资源（User Flash），不同系列器件支持不同容量大小的 Flash。FLASH 原语可参考 [UG295, Gowin 闪存资源\(User Flash\)用户指南](#)。

7 EMPU

7.1 EMCU

原语介绍

EMCU（ARM Cortex-M3 Microcontroller Unit）是一款基于 ARM Cortex-M3 的微处理器。采用了 32 位 AHB/APB 的总线模式。其内部实现了 2 个 UART、2 个 Timer 和 Watchdog 的功能。并且对外提供 16 位 GPIO、2 个 UART、JTAG、6 个 User Interrupt 接口。以及 AHB Flash 读取接口、AHB Sram 读写接口。同时对外还提供了 2 个 AHB 总线扩展接口和 1 个 APB 总线扩展接口。EMCU 增强了中断处理能力,改善了 FLASH 接口,提高了 MCU 运行主频。

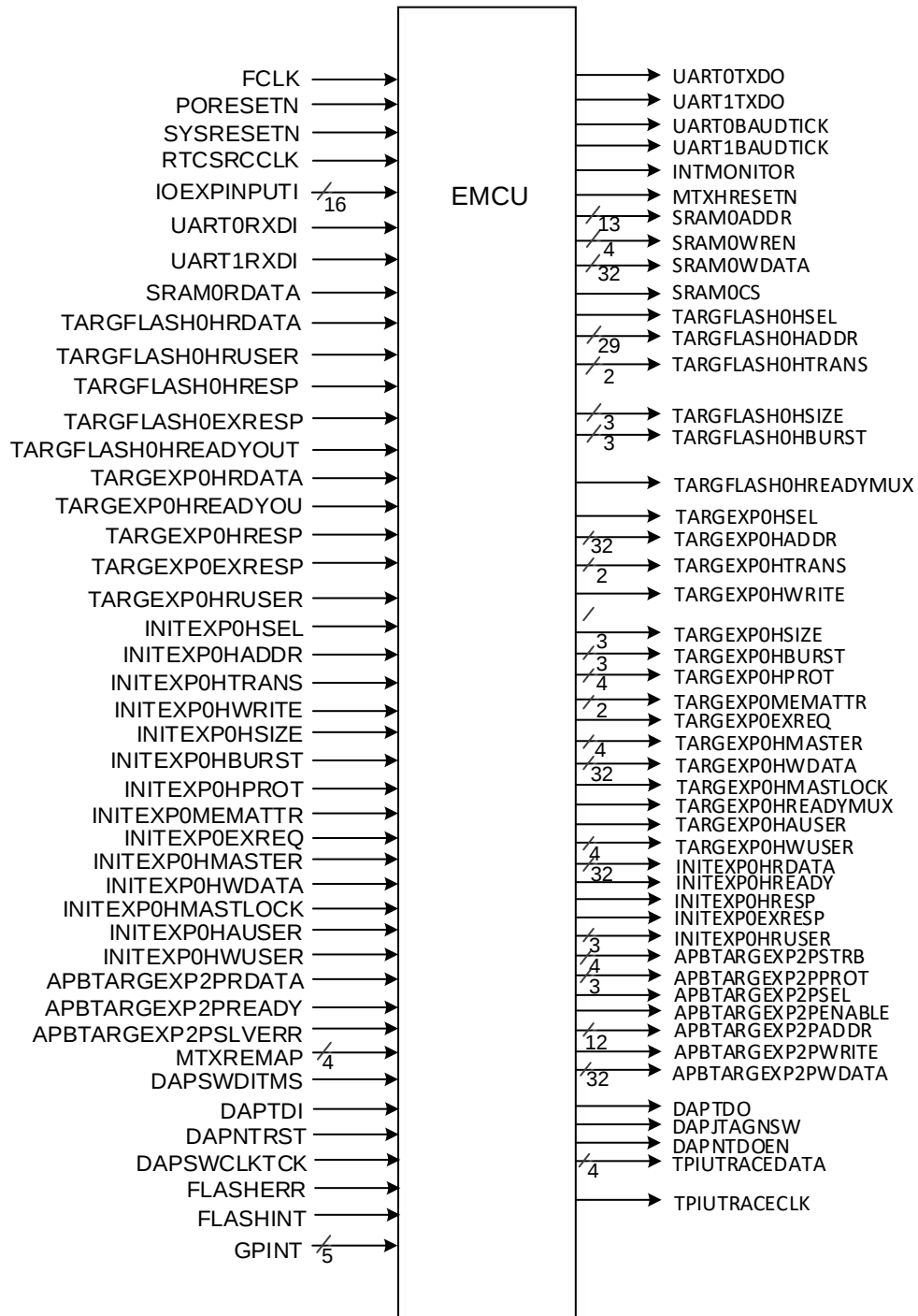
适用器件

表 7-1 EMCU 适用器件

家族	系列	器件
小蜜蜂®	GW1NS	GW1NS-4C
	GW1NSR	GW1NSR-4C
	GW1NSER	GW1NSER-4C

端口示意图

图 7-1 EMCU 端口示意图



端口介绍

表 7-2 EMCU 端口介绍

端口	I/O	描述
FCLK	input	Free running clock
PORESETN	input	Power on reset
SYSRESETN	input	System reset
RTCSRCLK	input	Used to generate RTC clock
IOEXPINPUTI[15:0]	input	IOEXPINPUTI
UART0RXDI	input	UART0RXDI
UART1RXDI	input	UART1RXDI
SRAM0RDATA[31:0]	input	SRAM Read data bus
TARGFLASH0HRDATA[31:0]	input	TARGFLASH0, HRDATA
TARGFLASH0HRUSER[2:0]	input	TARGFLASH0, HRUSER
TARGFLASH0HRESP	input	TARGFLASH0, HRESP
TARGFLASH0EXRESP	input	TARGFLASH0, EXRESP
TARGFLASH0HREADYOUT	input	TARGFLASH0, EXRESP
TARGEXP0HRDATA[31:0]	input	TARGEXP0, HRDATA
TARGEXP0HREADYOUT	input	TARGEXP0, HREADY
TARGEXP0HRESP	input	TARGEXP0, HRESP
TARGEXP0EXRESP	input	TARGEXP0, EXRESP
TARGEXP0HRUSER[2:0]	input	TARGEXP0, HRUSER
INITEXP0HSEL	input	INITEXP0, HSELx
INITEXP0HADDR[31:0]	input	INITEXP0, HADDR
INITEXP0HTRANS[1:0]	input	INITEXP0, HTRANS
INITEXP0HWRITE	input	INITEXP0, HWRITE
INITEXP0HSIZE[2:0]	input	INITEXP0, HSIZE
INITEXP0HBURST[2:0]	input	INITEXP0, HBURST
INITEXP0HPROT[3:0]	input	INITEXP0, HPROT
INITEXP0MEMATTR[1:0]	input	INITEXP0, MEMATTR
INITEXP0EXREQ	input	INITEXP0, EXREQ
INITEXP0HMASTER[3:0]	input	INITEXP0, HMASTER
INITEXP0HWDATA[31:0]	input	INITEXP0, HWDATA
INITEXP0HMASTLOCK	input	INITEXP0, HMASTLOCK
INITEXP0HAUSER	input	INITEXP0, HAUSER
INITEXP0HWUSER[3:0]	input	INITEXP0, HWUSER
APBTARGEXP2PRDATA[31:0]	input	APBTARGEXP2, PRDATA
APBTARGEXP2PREADY	input	APBTARGEXP2, PREADY
APBTARGEXP2PSLVERR	input	APBTARGEXP2, PSLVERR
MTXREMAP[3:0]	input	The MTXREMAP signals control the remapping of the boot memory range.

端口	I/O	描述
DAPSWDITMS	input	Debug TMS
DAPTDI	input	Debug TDI
DAPNTRST	input	Test reset
DAPSWCLKTCK	input	Test clock / SWCLK
FLASHERR	input	Output clock, used by the TPA to sample the other pins
FLASHINT	input	Output clock, used by the TPA to sample the other pins
GPINT	input	GPINT
IOEXPOUTPUTO[15:0]	output	IOEXPOUTPUTO
IOEXPOUTPUTENO[15:0]	output	IOEXPOUTPUTENO
UART0TXDO	output	UART0TXDO
UART1TXDO	output	UART1TXDO
UART0BAUDTICK	output	UART0BAUDTICK
UART1BAUDTICK	output	UART1BAUDTICK
INTMONITOR	output	INTMONITOR
MTXHRESETN	output	SRAM/Flash Chip reset
SRAM0ADDR[12:0]	output	SRAM address
SRAM0WREN[3:0]	output	SRAM Byte write enable
SRAM0WDATA[31:0]	output	SRAM Write data
SRAM0CS	output	SRAM Chip select
TARGFLASH0HSEL	output	TARGFLASH0, HSELx
TARGFLASH0HADDR[28:0]	output	TARGFLASH0, HADDR
TARGFLASH0HTRANS[1:0]	output	TARGFLASH0, HTRANS
TARGFLASH0HSIZE[2:0]	output	TARGFLASH0, HSIZE
TARGFLASH0HBURST[2:0]	output	TARGFLASH0, HBURST
TARGFLASH0HREADYMUX	output	TARGFLASH0, HREADYOUT
TARGEXP0HSEL	output	TARGEXP0, HSELx
TARGEXP0HADDR[31:0]	output	TARGEXP0, HADDR
TARGEXP0HTRANS[1:0]	output	TARGEXP0, HTRANS
TARGEXP0HWRITE	output	TARGEXP0, HWRITE
TARGEXP0HSIZE[2:0]	output	TARGEXP0, HSIZE
TARGEXP0HBURST[2:0]	output	TARGEXP0, HBURST
TARGEXP0HPROT[3:0]	output	TARGEXP0, HPROT
TARGEXP0MEMATTR[1:0]	output	TARGEXP0, MEMATTR
TARGEXP0EXREQ	output	TARGEXP0, EXREQ
TARGEXP0HMASTER[3:0]	output	TARGEXP0, HMASTER
TARGEXP0HWDATA[31:0]	output	TARGEXP0, HWDATA
TARGEXP0HMASTLOCK	output	TARGEXP0, HMASTLOCK
TARGEXP0HREADYMUX	output	TARGEXP0, HREADYOUT
TARGEXP0HAUSER	output	TARGEXP0, HAUSER
TARGEXP0HWUSER[3:0]	output	TARGEXP0, HWUSER

端口	I/O	描述
INITEXP0HRDATA[31:0]	output	INITEXP0, HRDATA
INITEXP0HREADY	output	INITEXP0, HREADY
INITEXP0HRESP	output	INITEXP0, HRESP
INITEXP0EXRESP	output	INITEXP0, EXRESP
INITEXP0HRUSER[2:0]	output	INITEXP0, HRUSER
APBTARGEXP2PSTRB[3:0]	output	APBTARGEXP2, PSTRB
APBTARGEXP2PPROT[2:0]	output	APBTARGEXP2, PPROT
APBTARGEXP2PSEL	output	APBTARGEXP2, PSELx
APBTARGEXP2PENABLE	output	APBTARGEXP2, PENABLE
APBTARGEXP2PADDR[11:0]	output	APBTARGEXP2, PADDR
APBTARGEXP2PWRITE	output	APBTARGEXP2, PWRITE
APBTARGEXP2PWDATA[31:0]	output	APBTARGEXP2, PWDATA
DAPTDO	output	Debug TDO
DAPJTAGNSW	output	JTAG or Serial-Wire selection JTAG mode(1) or SW mode(0)
DAPNTDOEN	output	TDO output pad control signal
TPIUTRACEDATA[3:0]	output	Output data
TPIUTRACECLK	output	Output clock, used by the TPA to sample the other pins

原语例化

Verilog 例化:

```
MCU u_sse050_top_syn (
.FCLK(fclk),
.PORESETN(poresetn),
.SYSRESETN(sysresetn),
.RTCSRCLK(rtsrcclk),
.IOEXPINPUTI(ioexpinputi[15:0]),
.IOEXPOUTPUTO(ioexpoutputo[15:0]),
.IOEXPOUTPUTENO(ioexpoutputeno[15:0]),
.UART0RXDI(uart0rxdi),
.UART0TXDO(uart0txdo),
.UART1RXDI(uart1rxdi),
.UART1TXDO(uart1txdo),
.SRAM0RDATA(sram0rdata[31:0]),
.SRAM0ADDR(sram0addr[12:0]),
.SRAM0WREN(sram0wren[3:0]),
.SRAM0WDATA(sram0wdata[31:0]),
.SRAM0CS(sram0cs),
```

```

.MTXHRESETN(mtxhreset),
.TARGFLASH0HSEL(targflash0hsel),
.TARGFLASH0HADDR(targflash0haddr[28:0]),
.TARGFLASH0HTRANS(targflash0htrans[1:0]),
.TARGFLASH0HSIZE(targflash0hsize[2:0]),
.TARGFLASH0HBURST(targflash0hburst[2:0]),
.TARGFLASH0HREADYMUX(targflash0hreadymux),
.TARGFLASH0HRDATA(targflash0hrdata[31:0]),
.TARGFLASH0HRUSER(targflash0hruser[2:0]),
.TARGFLASH0HRESP(targflash0hresp),
.TARGFLASH0EXRESP(targflash0exresp),
.TARGFLASH0HREADYOUT(targflash0hreadyout),
.TARGEXP0HSEL(targexp0hsel),
.TARGEXP0HADDR(targexp0haddr[31:0]),
.TARGEXP0HTRANS(targexp0htrans[1:0]),
.TARGEXP0HWRITE(targexp0hwrite),
.TARGEXP0HSIZE(targexp0hsize[2:0]),
.TARGEXP0HBURST(targexp0hburst[2:0]),
.TARGEXP0HPROT(targexp0hprot[3:0]),
.TARGEXP0MEMATTR(targexp0memattr[1:0]),
.TARGEXP0EXREQ(targexp0exreq),
.TARGEXP0HMASTER(targexp0hmaster[3:0]),
.TARGEXP0HWDATA(targexp0hwdata[31:0]),
.TARGEXP0HMASTLOCK(targexp0hmastlock),
.TARGEXP0HREADYMUX(targexp0hreadymux),
.TARGEXP0HAUSER(targexp0hauser),
.TARGEXP0HWUSER(targexp0hwuser[3:0]),
.TARGEXP0HRDATA(targexp0hrdata[31:0]),
.TARGEXP0HREADYOUT(targexp0hreadyout),
.TARGEXP0HRESP(targexp0hresp),
.TARGEXP0EXRESP(targexp0exresp),
.TARGEXP0HRUSER(targexp0hruser[2:0]),
.INITEXP0HSEL(initexp0hsel),
.INITEXP0HADDR(initexp0haddr[31:0]),
.INITEXP0HTRANS(initexp0htrans[1:0]),
.INITEXP0HWRITE(initexp0hwrite),
.INITEXP0HSIZE(initexp0hsize[2:0]),
.INITEXP0HBURST(initexp0hburst[2:0]),

```

```

.INITEXP0HPROT(initexp0hprot[3:0]),
.INITEXP0MEMATTR(initexp0memattr[1:0]),
.INITEXP0EXREQ(initexp0exreq),
.INITEXP0HMASTER(initexp0hmaster[3:0]),
.INITEXP0HWDATA(initexp0hwdata[31:0]),
.INITEXP0HMASTLOCK(initexp0hmastlock),
.INITEXP0HAUSER(initexp0hauser),
.INITEXP0HWUSER(initexp0hwuser[3:0]),
.INITEXP0HRDATA(initexp0hrdata[31:0]),
.INITEXP0HREADY(initexp0hready),
.INITEXP0HRESP(initexp0hresp),
.INITEXP0EXRESP(initexp0exresp),
.INITEXP0HRUSER(initexp0hruser[2:0]),
.APBTARGEXP2PSEL(apbtargexp2psel),
.APBTARGEXP2PENABLE(apbtargexp2penable),
.APBTARGEXP2PADDR(apbtargexp2paddr[11:0]),
.APBTARGEXP2PWRITE(apbtargexp2pwrite),
.APBTARGEXP2PWDATA(apbtargexp2pwdata[31:0]),
.APBTARGEXP2PRDATA(apbtargexp2prdata[31:0]),
.APBTARGEXP2PREADY(apbtargexp2pready),
.APBTARGEXP2PSLVERR(apbtargexp2pslverr),
.APBTARGEXP2PSTRB(apbtargexp2pstrb[3:0]),
.APBTARGEXP2PPROT(apbtargexp2pprot[2:0]),
.MTXREMAP(mtxremap[3:0]),
.DAPSWDITMS(dapswditms),
.DAPTDI(daptdi),
.DAPTDO(daptdo),
.DAPNTRST(dapntrst),
.DAPSWCLKTCK(dapswclk_tck),
.DAPNTDOEN(dapntdoen),
.DAPJTAGNSW(dapjtagns),
.TPIUTRACEDATA(tpiutracedata[3:0]),
.TPIUTRACECLK(tpiutracedata),
.FLASHERR(flasherr),
.GPINT(gpint),
.FLASHINT(flashint)
);

```

Vhdl 例化:

COMPONENT MCU

```
    PORT(  
    FCLK:IN std_logic;  
    PORESETN:IN std_logic;  
    SYSRESETN:IN std_logic;  
    RTCSRCLK:IN std_logic;  
    UART0RXDI:IN std_logic;  
    UART1RXDI:IN std_logic;  
    CLK:IN std_logic;  
    RESET:IN std_logic;  
    IOEXPINPUTI:IN std_logic_vector(15 downto 0);  
    SRAM0RDATA:IN std_logic_vector(31 downto 0);  
    TARGFLASH0HRDATA:IN std_logic_vector(31 downto 0);  
    TARGFLASH0HRUSER:IN std_logic_vector(2 downto 0);  
    TARGFLASH0HRESP:IN std_logic;  
    TARGFLASH0EXRESP:IN std_logic;  
    TARGFLASH0HREADYOUT:IN std_logic;  
    TARGEXP0HRDATA: IN std_logic_vector(31 downto 0);  
    TARGEXP0HREADYOUT:IN std_logic;  
    TARGEXP0HRESP:IN std_logic;  
    TARGEXP0EXRESP:IN std_logic;  
    TARGEXP0HRUSER: IN std_logic_vector(2 downto 0);  
    INITEXP0HSEL:IN std_logic;  
    INITEXP0HADDR: IN std_logic_vector(31 downto 0);  
    INITEXP0HTRANS: IN std_logic_vector(1 downto 0);  
    INITEXP0HWRITE: IN std_logic;  
    INITEXP0HSIZE: IN std_logic_vector(2 downto 0);  
    INITEXP0HBURST: IN std_logic_vector(2 downto 0);  
    INITEXP0HPROT: IN std_logic_vector(3 downto 0);  
    INITEXP0MEMATTR: IN std_logic_vector(1 downto 0);  
    INITEXP0EXREQ: IN std_logic;  
    INITEXP0HMASTER: IN std_logic_vector(3 downto 0);  
    INITEXP0HWDATA: IN std_logic_vector(31 downto 0);  
    INITEXP0HMASTLOCK: IN std_logic;  
    INITEXP0HAUSER: IN std_logic;  
    INITEXP0HWUSER: IN std_logic_vector(3 downto 0);  
    APBTARGEXP2PRDATA: IN std_logic_vector(3 downto 0);  
    APBTARGEXP2PREADY: IN std_logic;
```

APBTARGEXP2PSLVERR: IN std_logic;
MTXREMAP: IN std_logic_vector(3 downto 0);
DAPSWDITMS: IN std_logic;
DAPTDI: IN std_logic;
DAPNTRST: IN std_logic;
DAPSWCLKTCK: IN std_logic;
FLASHERR: IN std_logic;
FLASHINT: IN std_logic;
GPINT: IN std_logic;
IOEXPOUTPUTO:OUT std_logic_vector(15 downto 0);
IOEXPOUTPUTENO:OUT std_logic_vector(15 downto 0);
IOEXPINPUTI:OUT std_logic_vector(15 downto 0);
UART0TXDO: OUT std_logic;
UART1TXDO: OUT std_logic;
UART0BAUDTICK: OUT std_logic;
UART1BAUDTICK: OUT std_logic;
INTMONITOR: OUT std_logic;
MTXHRESETN: OUT std_logic;
SRAM0ADDR:OUT std_logic_vector(12 downto 0);
SRAM0WREN:OUT std_logic_vector(3 downto 0);
SRAM0WDATA:OUT std_logic_vector(31 downto 0);
SRAM0CS: OUT std_logic;
TARGFLASH0HSEL: OUT std_logic;
TARGFLASH0HREADYMUX: OUT std_logic;
SRAM0RDATA:OUT std_logic_vector(31 downto 0);
TARGFLASH0HADDR:OUT std_logic_vector(28 downto 0);
TARGFLASH0HTRANS:OUT std_logic_vector(1 downto 0);
TARGFLASH0HSIZE:OUT std_logic_vector(2 downto 0);
TARGFLASH0HBURST:OUT std_logic_vector(2 downto 0);
TARGFLASH0HRDATA:OUT std_logic_vector(31 downto 0);
TARGEXP0HADDR:OUT std_logic_vector(31 downto 0);
TARGEXP0HSEL: OUT std_logic;
TARGEXP0HWRITE: OUT std_logic;
TARGEXP0EXREQ: OUT std_logic;
TARGEXP0HMASTLOCK: OUT std_logic;
TARGEXP0HREADYMUX: OUT std_logic;
TARGEXP0HAUSER: OUT std_logic;
INITEXP0HREADY: OUT std_logic;

```

INITEXP0HRESP: OUT std_logic;
INITEXP0EXRESP: OUT std_logic;
TARGEXP0HTRANS:OUT std_logic_vector(1 downto 0);
TARGEXP0HSIZE:OUT std_logic_vector(2 downto 0);
TARGEXP0HBURST:OUT std_logic_vector(2 downto 0);
TARGEXP0HPROT:OUT std_logic_vector(3 downto 0);
TARGEXP0MEMATTR:OUT std_logic_vector(1 downto 0);
TARGEXP0HMASTER:OUT std_logic_vector(3 downto 0);
TARGEXP0HWDATA:OUT std_logic_vector(31 downto 0);
TARGEXP0HWUSER:OUT std_logic_vector(3 downto 0);
INITEXP0HRDATA:OUT std_logic_vector(31 downto 0);
INITEXP0HRUSER:OUT std_logic_vector(2 downto 0);
APBTARGEXP2PSTRB:OUT std_logic_vector(3 downto 0);
APBTARGEXP2PPROT:OUT std_logic_vector(2 downto 0);
APBTARGEXP2PADDR:OUT std_logic_vector(11 downto 0);
APBTARGEXP2PWDATA:OUT std_logic_vector(31 downto 0);
TPIUTRACEDATA:OUT std_logic_vector(3 downto 0);
APBTARGEXP2PSEL: OUT std_logic;
APBTARGEXP2PENABLE: OUT std_logic;
APBTARGEXP2PWRITE: OUT std_logic;
DAPTD0: OUT std_logic;
DAPJTAGNSW: OUT std_logic;
DAPNTDOEN: OUT std_logic;
TPIUTRACECLK: OUT std_logic;
);

END COMPONENT;

uut: MCU
    PORT MAP (
        FCLK=> fclk;
        PORESETN=> poresetn;
        SYSRESETN=> sysresetn;
        RTCSRCCLK=> rtcsrclk;
        UART0RXDI=> uart0rxdi;
        UART1RXDI=> uart1rxdi;
        CLK=>clk,
        RESET=>reset,
        IOEXPINPUTI=>ioexpinputi,

```

SRAM0ORDATA=>sram0rdata,
TARGFLASH0HRDATA=>targflash0hrdata,
TARGFLASH0HRUSER=>targflash0hruser,
TARGFLASH0HRESP=>targflash0hresp,
TARGFLASH0EXRESP=>targflash0exresp,
TARGFLASH0HREADYOUT=>targflash0hreadyout,
TARGEXP0HRDATA=>targexp0hrdata,
TARGEXP0HREADYOUT=>targexp0hreadyout,
TARGEXP0HRESP=>targexp0hresp,
TARGEXP0EXRESP=>targexp0exresp,
TARGEXP0HRUSER=>targexp0hruser,
INITEXP0HSEL=>initexp0hsel,
INITEXP0HADDR=>initexp0haddr,
INITEXP0HTRANS=>initexp0htrans,
INITEXP0HWRITE=>initexp0hwrite,
INITEXP0HSIZE=>initexp0hsize,
INITEXP0HBURST=>initexp0hburst,
INITEXP0HPROT=>initexp0hprot,
INITEXP0MEMATTR=>initexp0memattr,
INITEXP0EXREQ=>initexp0exreq,
INITEXP0HMASTER=>initexp0hmaster,
INITEXP0HWDATA=>initexp0hwdata,
INITEXP0HMASTLOCK=>initexp0hmastlock,
INITEXP0HAUSER=>initexp0hauser,
INITEXP0HWUSER=>initexp0hwuser,
APBTARGEXP2PRDATA=>apbtargexp2prdata,
APBTARGEXP2PREADY=>apbtargexp2pready,
APBTARGEXP2PSLVERR=>apbtargexp2pslverr,
MTXREMAP=>mtxremap,
DAPSWDITMS=>dapswditms,
DAPTDI=>daptdi,
DAPNTRST=>dapntrst,
DAPSWCLKTCK=>dapswclktck,
FLASHERR=>flasherr,
FLASHINT=>flashint,
GPINT=>gpint,
IOEXPOUTPUTO=>ioexpoutputo,
IOEXPOUTPUTENO=>ioexpoutputeno,

IOEXPINPUTI=>ioexpinputi,
UART0TXDO=>uart0txdo,
UART1TXDO=>uart1txdo,
UART0BAUDTICK=>uart0baudtick,
UART1BAUDTICK=>uart1baudtick,
INTMONITOR=>intmonitor,
MTXHRESETN=>mtxhresetn,
SRAM0ADDR=>sram0addr,
SRAM0WREN=>sram0wren,
SRAM0WDATA=>sram0wdata,
SRAM0CS=>sram0cs,
TARGFLASH0HSEL=>targflash0hsel,
TARGFLASH0HREADYMUX=>targflash0hreadymux,
SRAM0RDATA=>sram0rdata,
TARGFLASH0HADDR=>targflash0haddr,
TARGFLASH0HTRANS=>targflash0htrans,
TARGFLASH0HSIZE=>targflash0hsize,
TARGFLASH0HBURST=>targflash0hburst,
TARGFLASH0HRDATA=>targflash0hrdata,
TARGEXP0HADDR=>targexp0haddr,
TARGEXP0HSEL=>targexp0hsel,
TARGEXP0HWRITE=>targexp0hwrite,
TARGEXP0EXREQ=>targexp0exreq,
TARGEXP0HMASTLOCK=>targexp0hmastlock,
TARGEXP0HREADYMUX=>targexp0hreadymux,
TARGEXP0HAUSER=>targexp0hauser,
INITEXP0HREADY=>initexp0hready,
INITEXP0HRESP=>initexp0hresp,
INITEXP0EXRESP=>initexp0exresp,
TARGEXP0HTRANS=>targexp0htrans,
TARGEXP0HSIZE=>targexp0hsize,
TARGEXP0HBURST=>targexp0hburst,
TARGEXP0HPROT=>targexp0hprot,
TARGEXP0MEMATTR=>targexp0memattr,
TARGEXP0HMASTER=>targexp0hmaster,
TARGEXP0HWDATA=>targexp0hwdata,
TARGEXP0HWUSER=>targexp0hwuser,
INITEXP0HRDATA=>initexp0hrdata,

```
INITEXP0HRUSER=>initexp0hruser,  
APBTARGEXP2PSTRB=>apbtargexp2pstrb,  
APBTARGEXP2PPROT=>apbtargexp2pprot,  
APBTARGEXP2PADDR=>apbtargexp2paddr,  
APBTARGEXP2PWDATA=>apbtargexp2pwdata,  
TPIUTRACEDATA=>tpiutracedata,  
APBTARGEXP2PSEL=>apbtargexp2psel,  
APBTARGEXP2PENABLE=>apbtargexp2penable,  
APBTARGEXP2PWRITE=>apbtargexp2pwrite,  
DAPTDO=>daptdo,  
DAPJTAGNSW=>dapjtagnew,  
DAPNTDOEN=>dapntdoen,  
TPIUTRACECLK=>tpiutracedclk );
```

8 其它

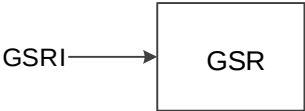
8.1 GSR

原语介绍

GSR（Global Set/Reset），全局置位/复位模块，可以实现全局置位/复位功能，低电平有效。默认一般连接高电平，若想动态控制，可连接外部信号，拉低实现寄存器等模块的置位/复位。

端口示意图

图 8-1 GSR 端口示意图



端口介绍

表 8-1 GSR 端口介绍

端口名	I/O	描述
GSRI	Input	GSR 输入，低电平有效

原语例化

Verilog 例化:

```
GSR gsr_inst(  
    .GSRI(GSRI)  
);
```

Vhdl 例化:

```
COMPONENT GSR  
    PORT (  
        GSRI:IN std_logic  
    );
```

```
END COMPONENT;  
gsr_inst:GSR  
    PORT MAP(  
        GSRI => GSRI  
    );
```

8.2 INV

原语介绍

INV (Inverter)，取反模块。

端口示意图

图 8-2 INV 端口示意图



端口介绍

表 8-2 INV 端口介绍

端口名	I/O	描述
I	Input	INV 数据输入
O	Output	INV 数据输出

原语例化

Verilog 例化:

```
INV uut (  
    .O(O),  
    .I(I)  
);
```

Vhdl 例化:

```
COMPONENT INV  
    PORT (  
        O:OUT std_logic;  
        I:IN std_logic  
    );  
END COMPONENT;  
uut:INV  
    PORT MAP(  

```

```
        O => O,  
        I => I  
    );
```

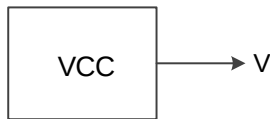
8.3 VCC

原语介绍

逻辑高电平发生器。

端口示意图

图 8-3 VCC 端口示意图



端口介绍

表 8-3 VCC 端口介绍

端口名	I/O	描述
V	Output	VCC 输出

原语例化

Verilog 例化:

```
VCC uut (  
    .V(V)  
);
```

Vhdl 例化:

```
COMPONENT VCC  
    PORT (  
        V:OUT std_logic  
    );  
END COMPONENT;  
uut:VCC  
    PORT MAP(  
        V => V  
    );
```

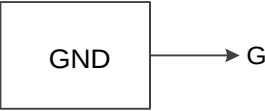
8.4 GND

原语介绍

逻辑低电平发生器。

端口示意图

图 8-4 GND 端口示意图



端口介绍

表 8-4 GND 端口介绍

端口名	I/O	描述
G	Output	GND 输出

原语例化

Verilog 例化:

```
GND uut (  
    .G(G)  
);
```

Vhdl 例化:

```
COMPONENT GND  
    PORT (  
        G:OUT std_logic  
    );  
END COMPONENT;  
uut:GND  
    PORT MAP(  
        G => G  
    );
```

8.5 BANDGAP

原语介绍

BANDGAP 的功能是为芯片中的某些模块提供恒定的电压和电流，若关掉 BANDGAP，则 OSC、PLL、FLASH 等模块将不再工作，可以起到降低器件功耗的作用。

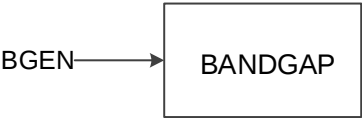
适用器件

表 8-5 BANDGAP 适用器件

家族	系列	器件
小蜜蜂®	GW1NZ	GW1NZ-1
	GW1N	GW1N-2, GW1N-1P5
	GW1NR	GW1NR-2

端口示意图

图 8-5 BANDGAP 端口示意图



端口介绍

表 8-6 BANDGAP 端口介绍

端口名	I/O	描述
BGEN	Input	BANDGAP 使能信号，高电平有效。

原语例化

Verilog 例化:

```
BANDGAP uut (  
    .BGEN(bgen)  
);
```

Vhdl 例化:

```
COMPONENT BANDGAP  
    PORT (  
        BGEN:IN std_logic  
    );  
END COMPONENT;  
uut:BANDGAP  
    PORT MAP(  
        BGEN=> I  
    );
```

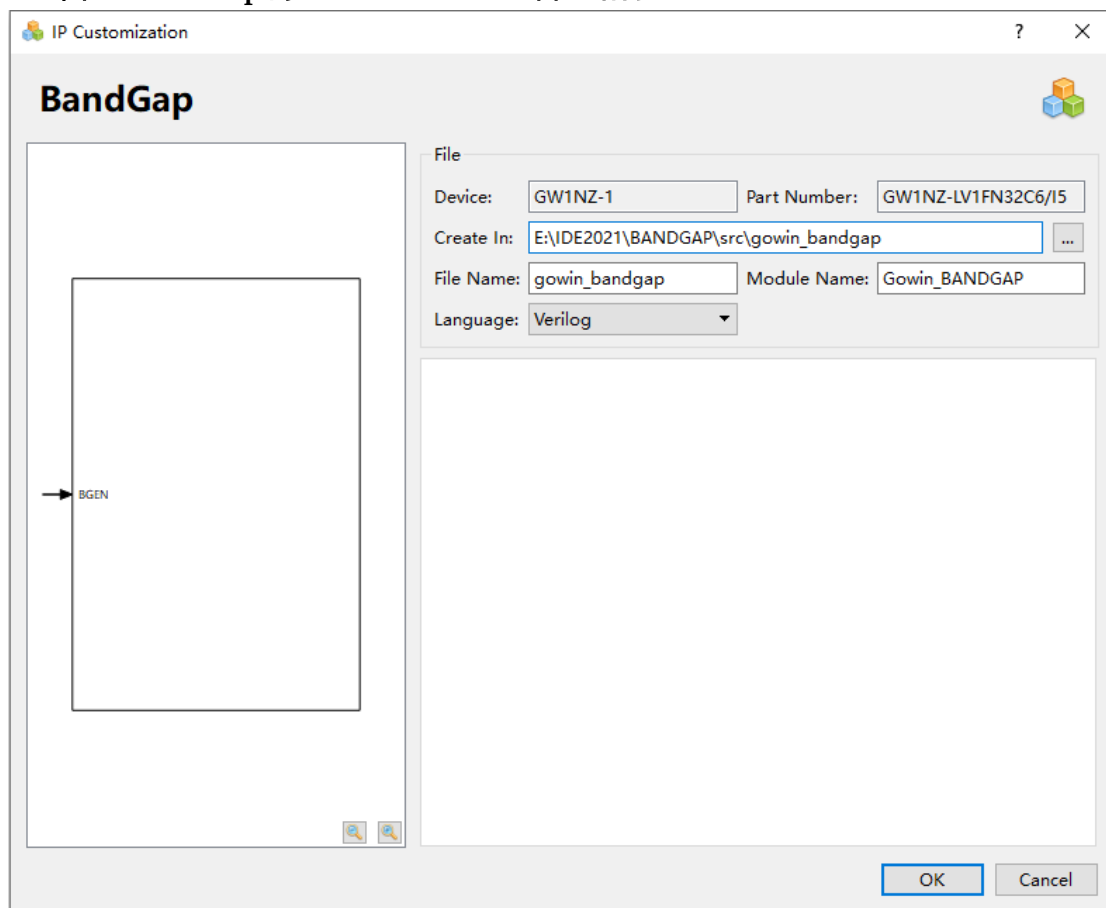
IP 调用

在 IP Core Generator 界面中，单击“BandGap”，界面右侧会显示 BandGap 的相关信息概要。

IP 配置

在 IP Core Generator 界面中，双击 BandGap，弹出 BandGap 的“IP Customization”窗口。该窗口包括“File”配置框、“Options”配置框以及端口显示框图，如图 8-6 所示。

图 8-6 BandGap 的 IP Customization 窗口结构



1. File 配置框

File 配置框用于配置产生的 IP 设计文件的相关信息。

- **Device:** 显示已配置的 Device 信息；
- **Part Number:** 显示已配置的 Part Number 信息；
- **Create In:** 配置产生的 IP 设计文件的目标路径。可在右侧文本框中重新编辑目标路径，也可通过文本框右侧选择按钮选择目标路径。
- **File Name:** 配置产生的 IP 设计文件的文件名。在右侧文本框可重新编辑文件名称；
- **Module Name:** 配置产生的 IP 设计文件的 module name。在右侧文本框可重新编辑模块名称。Module Name 不能与原语名称相同，若相同，则报出 Error 提示；
- **Language:** 配置产生的 IP 设计文件的硬件描述语言。选择右侧下拉

列表框，选择目标语言，支持 Verilog 和 VHDL。

2. 端口显示框图

端口显示框图显示当前 IP Core 的配置结果示例框图，如图 8-6 所示。

IP 生成文件

IP 窗口配置完成后，产生以配置文件“File Name”命名的三个文件，以默认配置为例进行介绍：

- IP 设计文件“gowin_bandgap.v”为完整的 verilog 模块，根据用户的 IP 配置，产生实例化的 BandGap；
- IP 设计使用模板文件 gowin_bandgap_tmp.v，为用户提供 IP 设计使用模板文件；
- IP 配置文件：“gowin_bandgap.ipc”，用户可加载该文件对 IP 进行配置。

注！

如配置中选择的语言是 VHDL，则产生的前两个文件名后缀为.vhd。

8.6 SPMI

原语介绍

SPMI（System Power Management Interface）是一种双线串行接口，可用于动态控制片上系统内部电源的关断与开启。

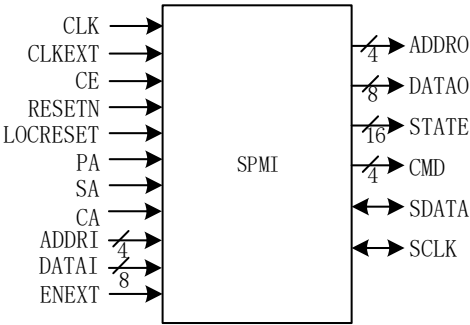
适用器件

表 8-7 SPMI 适用器件

家族	系列	器件
小蜜蜂®	GW1NZ	GW1NZ-1

端口示意图

图 8-7 SPMI 端口示意图



端口介绍

表 8-8 SPMI 端口介绍

端口	I/O	描述
CLK	input	Clock input
CLKEXT	input	External clock input
CE	input	Clock Enable
RESETN	input	Reset input
ENEXT	input	Enext input
LOCRESET	input	Local reset input
PA	input	Priority arbitration input
SA	input	Secondary arbitration input
CA	input	Connection arbitration input
ADDRI	input	Addr input
DATAI	input	Data input
ADDRO	output	Addr output
DATAO	output	data output
STATE	output	state output
CMD	output	command output
SDATA	inout	SPMI Serial data
SCLK	inout	SPMI Serial Clock

原语例化

Verilog 例化:

```

SPMI uut (
    .ADDRO(addr),
    .DATAO(datao),
    .STATE(state),
    .CMD(cmd),
    .SDATA(sdata),
    .SCLK(sclk),
    .CLK(clk),
    .CE(ce),
    .RESETN(resetn),
    .LOCRESET(locreset),
    .PA(pa),
    .SA(sa),
    .CA(ca),
    .ADDRI(addr),

```

```

        .DATAI(datai),
        .CLKEXT(clkext),
        .ENEXT(enext)
    );

```

Vhdl 例化:

```

COMPONENT SPMI
    PORT(
        CLK:IN std_logic;
        CLKEXT:IN std_logic;
        CE:IN std_logic;
        RESETN:IN std_logic;
        ENEXT:IN std_logic;
        LOCRESET:IN std_logic;
        PA:IN std_logic;
        SA:IN std_logic;
        CA:IN std_logic;
        ADDRI:IN std_logic_vector(3 downto 0);
        DATAI:IN std_logic_vector(7 downto 0);
        ADDRO:OUT std_logic_vector(3 downto 0);
        DATAO:OUT std_logic_vector(7 downto 0);
        STATE:OUT std_logic_vector(15 downto 0);
        CMD:OUT std_logic_vector(3 downto 0);
        SDATA:INOUT std_logic;
        SCLK:INOUT std_logic
    );
END COMPONENT;

 uut: SPMI
    PORT MAP (
        CLK=>clk,
        CLKEXT=>clkext,
        CE=>ce,
        RESETN=>resetn,
        ENEXT=>enext,
        LOCRESET=>locreset,
        PA=>pa,
        SA=>sa,
        CA=>ca,
        ADDRI=>addri,

```

```

DATAI=>datai,
ADDRO=>addro,
DATAO=>datao,
STATE=>state,
CMD=>cmd,
SDATA=>sdata,
SCLK=>sclk

```

```
);
```

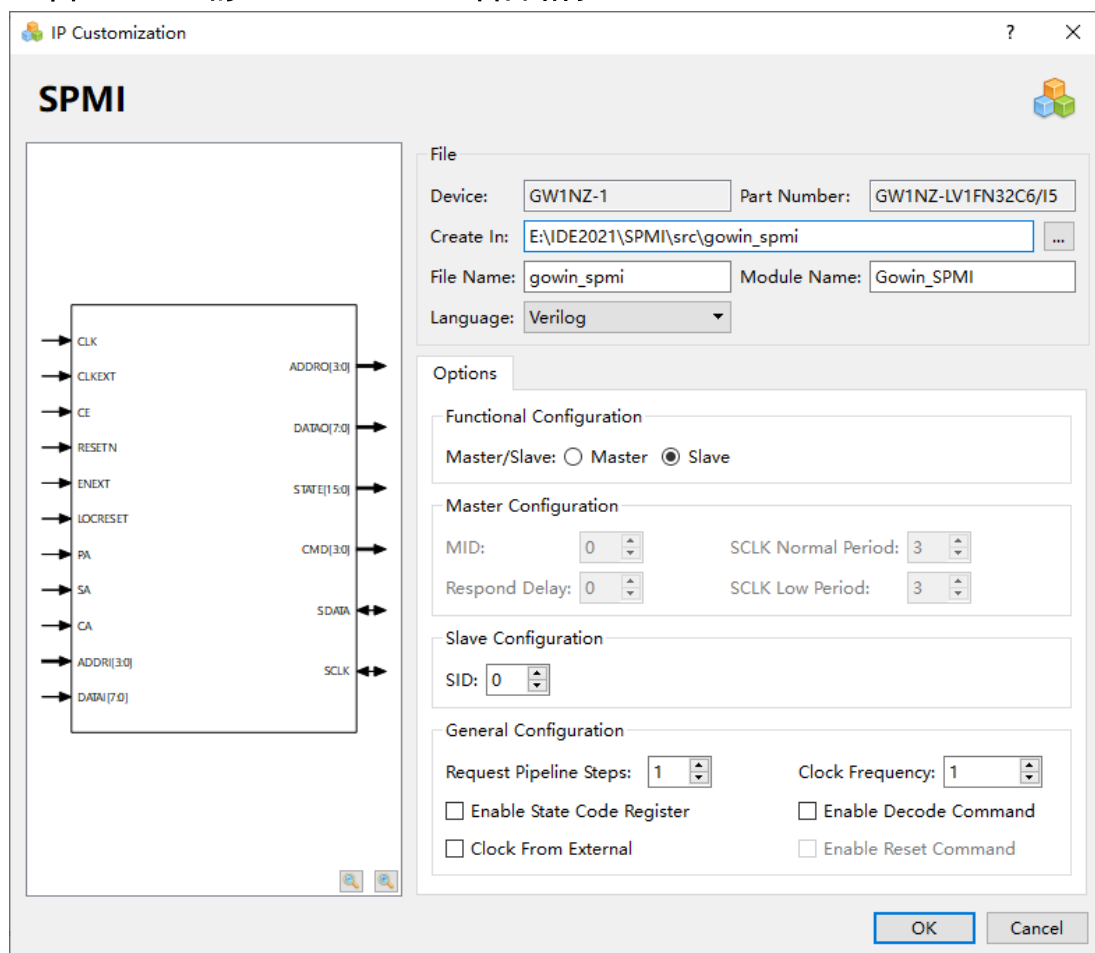
IP 调用

在 IP Core Generator 界面中，单击“SPMI”，界面右侧会显示 SPMI 的相关信息概要。

IP 配置

在 IP Core Generator 界面中，双击 SPMI，弹出 SPMI 的“IP Customization”窗口。该窗口包括“File”配置框、“Options”配置框以及端口显示框图，如图 8-8 所示。

图 8-8 SPMI 的 IP Customization 窗口结构



1. File 配置框

- File 配置框用于配置产生的 IP 设计文件的相关信息。
- SPMI 的 File 配置框的使用和 BANDGAP 模块类似，具体请参考 [8.5 BANDGAP](#) 部分的 File 配置框。

2. Options 配置框

- Options 配置框用于用户自定义配置 IP，Options 配置框如图 8-8 所示。
- Functional Configuration:
 - Master/Slave: 将 SPMI 设置为主机或从机。
- Master Configuration:
 - MID: 主机的 ID，设置范围为 0-3，默认值为 0。
 - Respond Delay: 设置响应延迟时间。
 - SCLK Normal Period: Normal 模式下 sclk 的周期长度。
 - SCLK Low Period: 睡眠模式下 sclk 的周期长度。
- Slave Configuration:
 - SID: 设置 SPMI 从机的 ID。
- General configuration:
 - Enable State Code Register: 启用或禁用寄存器。例如，如果选择“启用状态代码寄存器”选项，则输出 STATE 数据将通过一个寄存器。
 - Request Pipeline Steps: 设置请求信号采样时间的延迟步长。
 - Enable Decode Command: 启用或禁用解码。如果选择启用解码命令，SPMI 将解码复位，睡眠，关闭和唤醒命令。
 - Enable Reset Command: 启用或禁用重置命令。
 - Clock From External: 启用或禁用外部时钟。
 - Clock Frequency: 系统时钟频率。

3. 端口显示框

端口显示框图显示当前 IP Core 的配置结果示例框图，如图 8-8 所示。

IP 生成文件

IP 窗口配置完成后，产生以配置文件“File Name”命名的三个文件，以默认配置为例进行介绍：

- IP 设计文件“gowin_spmi.v”为完整的 verilog 模块，根据用户的 IP 配置，产生实例化的 SPMI；

- IP 设计使用模板文件 gowin_spmi_tmp.v，为用户提供 IP 设计使用模板文件；
- IP 配置文件：“gowin_spmi.ipc”，用户可加载该文件对 IP 进行配置。

注！
如配置中选择的语言是 VHDL，则产生的前两个文件名后缀为.vhd。

8.7 I3C

原语介绍

I3C（Improved Inter Integrated Circuit）是一种两线式总线，兼具了 I²C 和 SPI 的关键特性，能有效的减少集成电路芯片系统的物理端口、支持低功耗、高数据速率和其他已有端口协议的优点。

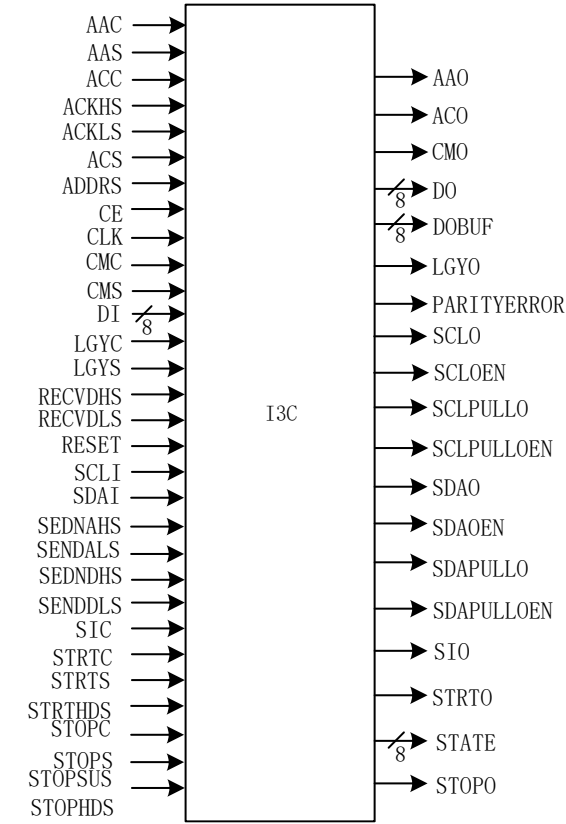
适用器件

表 8-9 I3C 适用器件

家族	系列	器件
小蜜蜂®	GW1NZ	GW1NZ-1

端口示意图

图 8-9 I3C 端口示意图



端口介绍

表 8-10 I3C 端口介绍

端口	I/O	描述
CE	input	Clock Enable
RESET	input	Reset input
CLK	input	Clock input
LGYS	input	The current communication object is the I2C setting signal
CMS	input	The device enters the Master's set signal
ACS	input	Select the setting signal when determining whether to continue.
AAS	input	Reply the ACK setting signal when a reply is required from the ACK/NACK
STOPS	input	Input the STOP command
STRTS	input	Input the START command.
LGYC	input	The current communication object is the I2C
CMC	input	The reset signal that the device is in master.
ACC	input	The reset signal that selects continue when selecting whether to continue
AAC	input	Reply the ACK reset signal when a reply is required from the ACK/NACK
SIC	input	Interrupt to identify the reset signal
STOPC	input	The reset signal is in STOP state
STRTC	input	The reset signal is in START state
STRTHDS	input	Adjust the setting signal when generating START
SENDAHS	input	Adjust the setting signal of SCL at a high level when the address is sent.
SENDALS	input	Adjust the setting signal of SCL at a low level when the address is sent
ACKHS	input	Adjust the setting signal of SCL at a high level in ACK.
SENDCLS	input	Adjust the setting signal of SCL at a low level in ACK.
RECVDHS	input	Adjust the setting signal of SCL at a high level when the data are received
RECVDLS	input	Adjust the setting signal of SCL at a low level when the data are received
ADDRS	input	The slave address setting interface
DI	input	Data Input.
SDAI	input	I3C serial data input
SCLI	input	I3C serial clock input
LGYO	output	Output the current communication object as the I2C command.
CMO	output	Output the command of the device is in the Master mode.
ACO	output	Continue to output when selecting whether to continue
AAO	output	Reply ACK when you need to reply ACK/NACK

端口	I/O	描述
SIO	output	Interrupt to output the identity bit
STOPO	output	Output the STOP command
STRTO	output	Output the START command
PARITYERROR	output	Output check when receiving data
DOBUF	output	Data output after caching
DO	output	Data output directly
STATE	output	Output the internal state
SDAO	output	I3C serial data output
SCLO	output	I3C serial clock output
SDAOEN	output	I3C serial data oen output
SCLOEN	output	I3C serial clock oen output
SDAPULLO	output	Controllable pull-up of the I3C serial data
SCLPULLO	output	Controllable pull-up of the I3C serial clock
SDAPULLOEN	output	Controllable pull-up of the I3C serial data oen
SCLPULLOEN	output	Controllable pull-up of the I3C serial clock oen

原语例化

Verilog 例化:

```

I3C i3c_inst (
    .LGYO(lgyo),
    .CMO(cmo),
    .ACO(aco),
    .AAO(aao),
    .SIO(sio),
    .STOPO(stopo),
    .STRTO(strto),
    .PARITYERROR(parityerror),
    .DOBUF(dobuf),
    .DO(dout),
    .STATE(state),
    .SDAO(sdao),
    .SCLO(sclo),
    .SDAOEN(sdaoen),
    .SCLOEN(scloen),
    .SDAPULLO(sdapullo),
    .SCLPULLO(sclpullo),
    .SDAPULLOEN(sdapulloen),

```

```

.SCLPULLOEN(sclpulloen),
.LGYS(lgys),
.CMS(cms),
.ACS(acs),
.AAS(aas),
.STOPS(stops),
.STRTS(strts),
.LGYC(lgyc),
.CMC(cmc),
.ACC(acc),
.AAC(aac),
.SIC(sic),
.STOPC(stopc),
.STRTC(strtc),
.STRTHDS(strthds),
.SENDAHS(sendahs),
.SENDALS(sendals),
.ACKHS(ackhs),
.ACKLS(ackls),
.STOPSUS(stopsus),
.STOPHDS(stophds),
.SENDDHS(senddhs),
.SENDDLs(senddls),
.RECVDHS(recvdhs),
.RECVDLS(recvdls),
.ADDRS(addr),
.DI(di),
.SDAI(sdai),
.SCLI(scli),
.CE(ce),
.RESET(reset),
.CLK(clk)

```

```
);
```

Vhdl 例化:

```
COMPONENT I3C
```

```
PORT (
```

```
    LGYO: OUT STD_LOGIC;
```

```
    CMO: OUT STD_LOGIC;
```

```
ACO: OUT STD_LOGIC;
AAO: OUT STD_LOGIC;
SIO: OUT STD_LOGIC;
STOPO: OUT STD_LOGIC;
STRTO: OUT STD_LOGIC;
PARITYERROR: OUT STD_LOGIC;
DOBUF: OUT STD_LOGIC_VECTOR(7 DOWNT0 0);
DOUT: OUT STD_LOGIC_VECTOR(7 DOWNT0 0);
STATE: OUT STD_LOGIC_VECTOR(7 DOWNT0 0);
SDAO: OUT STD_LOGIC;
SCLO: OUT STD_LOGIC;
SDAOEN: OUT STD_LOGIC;
SCLOEN: OUT STD_LOGIC;
SDAPULLO: OUT STD_LOGIC;
SCLPULLO: OUT STD_LOGIC;
SDAPULLOEN: OUT STD_LOGIC;
SCLPULLOEN: OUT STD_LOGIC;
LGYS: IN STD_LOGIC;
CMS: IN STD_LOGIC;
ACS: IN STD_LOGIC;
AAS: IN STD_LOGIC;
STOPS: IN STD_LOGIC;
STRTS: IN STD_LOGIC;
LGYC: IN STD_LOGIC;
CMC: IN STD_LOGIC;
ACC: IN STD_LOGIC;
AAC: IN STD_LOGIC;
SIC: IN STD_LOGIC;
STOPC: IN STD_LOGIC;
STRTC: IN STD_LOGIC;
STRTHDS: IN STD_LOGIC;
SENDAHS: IN STD_LOGIC;
SENDALS: IN STD_LOGIC;
ACKHS: IN STD_LOGIC;
ACKLS: IN STD_LOGIC;
STOPSUS: IN STD_LOGIC;
STOPHDS: IN STD_LOGIC;
SENDHDS: IN STD_LOGIC;
```

```

        SENDDLS: IN STD_LOGIC;
        RECV DHS: IN STD_LOGIC;
        RECV DLS: IN STD_LOGIC;
        ADDR S: IN STD_LOGIC;
        DI: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        SDAI: IN STD_LOGIC;
        SCLI: IN STD_LOGIC;
        CE: IN STD_LOGIC;
        RESET: IN STD_LOGIC;
        CLK: IN STD_LOGIC
    );
END COMPONENT;

```

uut: I3C

```

PORT MAP (
    LGYO => lgyo,
    CMO => cmo,
    ACO => aco,
    AAO => aao,
    SIO => sio,
    STOPO => stopo,
    STRTO => strto,
    PARITYERROR => parityerror,
    DOBUF => dobuf,
    DOUT => dout,
    STATE => state,
    SDAO => sdao,
    SCLO => sclo,
    SDAOEN => sdaoen,
    SCLOEN => scloen,
    SDAPULLO => sdapullo,
    SCLPULLO => sclpullo,
    SDAPULLOEN => sdapulloen,
    SCLPULLOEN => sclpulloen,
    LGYS => lgys,
    CMS => cms,
    ACS => acs,
    AAS => aas,

```

```
    STOPS => stops,  
    STRTS => strts,  
    LGYC => lgyc,  
    CMC => cmc,  
    ACC => acc,  
    AAC => aac,  
    SIC => sic,  
    STOPC => stopc,  
    STRTC => strtc,  
    STRTHDS => strthds,  
    SENDAHS => sendahs,  
    SENDALS => sendals,  
    ACKHS => ackhs,  
    ACKLS => ackls,  
    STOPSUS => stopsus,  
    STOPHDS => stophds,  
    SENDDHS => senddhs,  
    SENDDLS => senddls,  
    RECVDHS => recvdhs,  
    RECVDLS => recvdls,  
    ADDRS => addrs,  
    DI => di,  
    SDAI => sdai,  
    SCLI => scli,  
    CE => ce,  
    RESET => reset,  
    CLK => clk  
);
```

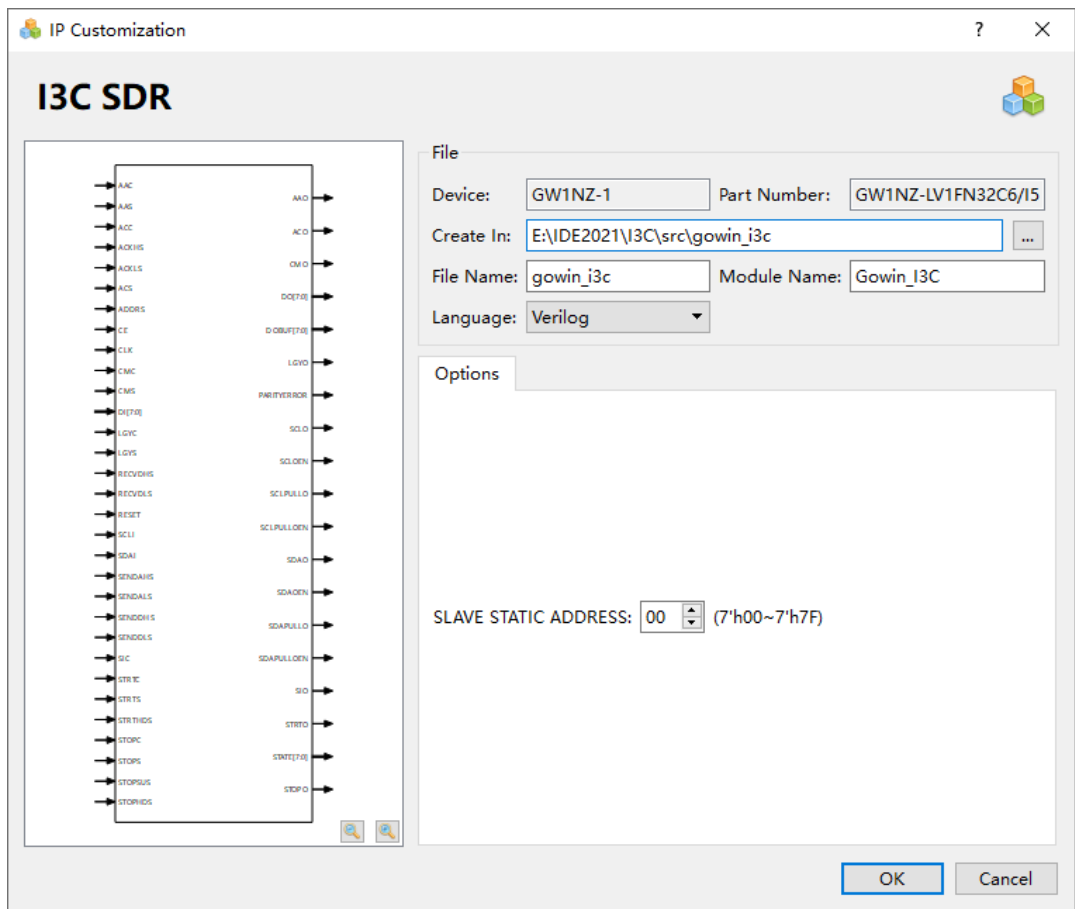
IP 调用

在 IP Core Generator 界面中单击 I3C 下的 I3C SDR，界面右侧会显示 I3C SDR 的相关信息概要。

IP 配置

在 IP Core Generator 界面中，双击“I3C SDR”，弹出 I3C 的“IP Customization”窗口，该窗口包括“File”配置框、“Options”配置框以及端口显示框图，如图 8-10 所示。

图 8-10 I3C 的 IP Customization 窗口结构



1. File 配置框

- File 配置框用于配置产生的 IP 设计文件的相关信息。
- I3C 的 File 配置框的使用 BANDGAP 模块类似，具体请参考 [8.5 BANDGAP](#) 部分的 File 配置框。。

2. Options 配置框

- Options 配置框用于用户自定义配置 IP，Options 配置框如图 8-10 所示。
- SLAVE STATIC ADDRESS - 指定从机的静态地址。

3. 端口显示框图

端口显示框图显示 IP Core 的配置结果示例框图，如图 8-10 所示。

IP 生成文件

IP 窗口配置完成后，产生以配置文件“File Name”命名的三个文件，以默认配置为例进行介绍：

- IP 设计文件“gowin_i3c.v”为完整的 verilog 模块，根据用户的 IP 配置，产生实例化的 I3C；

- IP 设计使用模板文件 `gowin_i3c_tmp.v`，为用户提供 IP 设计使用模板文件；
- IP 配置文件：“`gowin_i3c.ipc`”，用户可加载该文件对 IP 进行配置。

注！

如配置中选择的语言是 VHDL，则产生的前两个文件名后缀为.vhd。

8.8 activeFlash

原语介绍

激活内嵌 SPI Nor Flash 的模块。实例化 `activeFlash` 时，`I_active_flash_sclk` 需为时钟信号，`I_active_flash_holdn` 需要拉高。

注！

实例化 `activeFlash`，用户设计资源会增加 14 个 LUT 和 10 个 REG 资源。

适用器件

表 8-11 activeFlash 适用器件

家族	系列	器件
晨熙®	GW2AN	GW2AN-18X, GW2AN-9X

端口示意图

图 8-11 activeFlash 端口示意图



端口介绍

表 8-12 activeFlash 端口介绍

端口	I/O	描述
<code>I_active_flash_holdn</code>	input	控制时钟保持信号，高电平输入激活 Flash
<code>I_active_flash_sclk</code>	input	时钟输入信号
<code>O_active_flash_ready</code>	output	Ready 输出标志，高电平表示 Flash 已被激活

原语例化

Verilog 例化：

```

activeFlash activeFlash_inst (
    . I_active_flash_holdn (I_active_flash_holdn),
    . I_active_flash_sclk (I_active_flash_sclk),
    . O_active_flash_ready (O_active_flash_ready)
  )
  
```

);

Vhdl 例化:

```
COMPONENT activeFlash
    PORT(
        I_active_flash_holdn:IN std_logic;
        I_active_flash_sclk:IN std_logic;
        O_active_flash_ready:OUT std_logic
    );
END COMPONENT;
 uut: activeFlash
    PORT MAP (
        I_active_flash_holdn => I_active_flash_holdn,
        I_active_flash_sclk => I_active_flash_sclk,
        O_active_flash_ready => O_active_flash_ready
    );
```

调用条件

满足以下条件之一时，需要在用户设计中实例化 activeFlash 模块。

- 1. 高云半导体云源®软件 Configuration>Bitstream 配置页面 background programming 配置为 I2C/JTAG/SSPI/QSSPI, 需要实例化 activeFlash;
- 2. 高云半导体云源®软件 Configuration>Bitstream 配置页面 background programming 配置为 OFF，且使用 SPI Nor Flash Interface IP 时，需要实例化 activeFlash。

8.9 OTP

原语介绍

OTP(One Time Programming), 芯片产品信息存储在工厂区，使用 OTP 可以读取芯片产品信息。

适用器件

表 8-13 OTP 适用器件

家族	系列	器件
晨熙®	GW2AN	GW2AN-18X, GW2AN-9X
	GW5AT	GW5AT-138
	GW5A	GW5A-25, GW5A-138 B 版本
	GW5AST	GW5AST-138 B 版本

端口示意图

图 8-12 GW2AN OTP 端口示意图

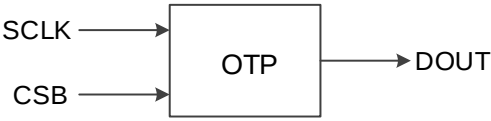
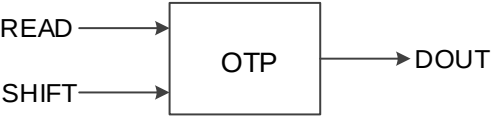


图 8-13 Arora V OTP 端口示意图



端口介绍

表 8-14 GW2AN OTP 端口介绍

端口	I/O	描述
SCLK	input	串行输入时钟，下降沿将数据移出到 DOUT，读数据时建议使用 2.5MHz 时钟。
CSB	input	片选信号，低电平有效。
DOUT	output	串行数据输出

表 8-15 Arora V OTP 端口介绍

端口	I/O	描述
READ	input	将相应的 128 位 efuse 寄存器数据加载到移位寄存器，脉冲信号，高有效。
SHIFT	input	将移位寄存器数据移位输出到输出接口，电平信号，高有效。
DOUT	output	读芯片 DNA 信息/用户信息数据输出

参数介绍

表 8-16 Arora V OTP 参数介绍

参数名	取值范围	默认值	描述
MODE	1'b0, 1'b1	1'b0	OTP 模式选择 1'b0:读取芯片 DNA 信息; 1'b1:读取用户信息

原语例化

GW2AN OTP 例化

Verilog 例化:

OTP uut (

```

        . SCLK (SCLK),
        .CSB (CSB),
        . DOUT (DOUT)
    );

```

Vhdl 例化:

```

COMPONENT OTP
    PORT(
        SCLK:IN std_logic;
        CSB:IN std_logic;
        DOUT:OUT std_logic
    );
END COMPONENT;
 uut: OTP
    PORT MAP (
        SCLK => SCLK,
        CSB => CSB,
        DOUT => DOUT
    );

```

Arora V OTP 例化**Verilog 例化:**

```

OTP uut (
    . READ(READ),
    .SHIFT(SHIFT),
    . DOUT (DOUT)
);
defparam uut.MODE=1'b0;

```

Vhdl 例化:

```

COMPONENT OTP
    GENERIC (
        MODE : bit := '0'
    );
    PORT(
        READ:IN std_logic;
        SHIFT:IN std_logic;
        DOUT:OUT std_logic
    );

```

```
END COMPONENT;  
uut: OTP  
    GENERIC MAP (  
        MODE =>'0'  
    )  
    PORT MAP (  
        READ => READ,  
        SHIFT => SHIFT,  
        DOUT => DOUT  
    );
```

8.10 SAMB

原语介绍

SAMB(SPI Address for Multi Boot)，用于 Multi Boot 模式的地址选择，可以选择静态和动态地址。

适用器件

表 8-17 SAMB 适用器件

家族	系列	器件
晨熙®	GW2AN	GW2AN-18X, GW2AN-9X
	GW5AT	GW5AT-138
	GW5A	GW5A-25, GW5A-138 B 版本
	GW5AST	GW5AST-138 B 版本

端口示意图

图 8-14 GW2AN SAMB 端口示意图

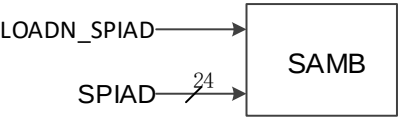
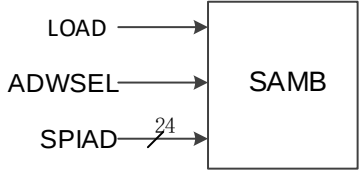


图 8-15 Arora V SAMB 端口示意图



端口介绍

表 8-18 GW2AN SAMB 端口介绍

端口	I/O	描述
LOADN_SPIAD	input	用来选择静态 SPI 地址或者动态 SPI 地址信号指定的位置。高电平时选择静态 SPI 地址，低电平时选择动态 SPI 地址信号 SPIAD。
SPIAD[23:0]	input	SPI 地址信号

表 8-19 Arora V SAMB 端口介绍

端口	I/O	描述
LOAD	input	用来选择静态 SPI 地址或者动态 SPI 地址信号指定的位置。低电平时选择静态 SPI 地址，高电平时选择动态 SPI 地址信号 SPIAD。
SPIAD[23:0]	input	SPI 地址信号
ADWSEL	Input	地址位宽选择信号 0: 地址为 24-bit; 1: 地址是 32-bit (低 8 位地址补零)。

参数介绍

表 8-20 Arora V SAMB 参数介绍

参数名	取值范围	默认值	描述
MODE	2'b00, 2'b01, 2'b10, 2'b11	2'b00	SAMB 模式选择 ● 2'b00: Normal mode ● 2'b01: Fast mode ● 2'b10: Dual mode ● 2'b11: Quad mode

原语例化

GW2AN SAMB 例化

Verilog 例化:

```
SAMB uut (  
    . LOADN_SPIAD (LOADN_SPIAD),  
    . SPIAD (SPIAD)  
);
```

Vhdl 例化:

```
COMPONENT SAMB  
PORT(  
    LOADN_SPIAD:IN std_logic;
```

```

        SPIAD:IN std_logic_vector (23 downto 0)
    );
END COMPONENT;
 uut: SAMB
    PORT MAP (
        LOADN_SPIAD => LOADN_SPIAD,
        SPIAD => SPIAD
    );

```

Arora V SAMB 例化

Verilog 例化:

```

SAMB uut (
    . LOAD (LOAD),
    . SPIAD (SPIAD),
    .ADWSEL(ADWSEL)
);
defparam uut.MODE=2'b00;

```

Vhdl 例化:

```

COMPONENT SAMB
    GENERIC (
        MODE : bit_vector := "00"
    );
    PORT(
        LOAD:IN std_logic;
        ADWSEL:IN std_logic;
        SPIAD:IN std_logic_vector (23 downto 0)
    );
END COMPONENT;
 uut: SAMB
    GENERIC (
        MODE => "00"
    );
    PORT MAP (
        LOAD => LOAD,
        ADWSEL => ADWSEL,
        SPIAD => SPIAD
    );

```

8.11 CMSER

原语介绍

配置内存软错误恢复（Configuration Memory Soft Error Recovery，CMSER）可以持续监控配置内存以检测任何软错误，然后尝试在其能力范围内纠正它们。通过在用户设计后台逐帧读取配置内存，并进行 ECC 解码和 CRC 校验比对来实现。通过将校正后的帧数据编程回 SRAM 来校正有限数量的错误位。

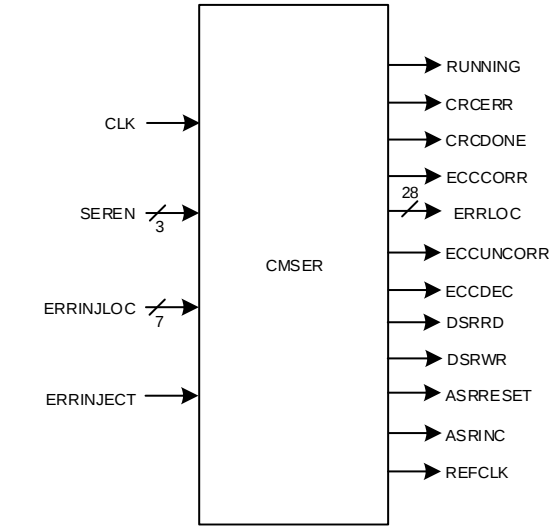
适用器件

表 8-21 CMSER 适用器件

家族	系列	器件
晨熙®	GW5AT	GW5AT-138
	GW5A	GW5A-138 B 版
	GW5AST	GW5AST-138 B 版

端口示意图

图 8-16 CMSER 端口示意图



端口介绍

表 8-22 CMSER 端口介绍

端口	I/O	描述
CLK	input	Clock input
SEREN	input	(the critical signal using TMR scheme for error reduction) rising edge (at least two of three bits are detected to transit from "0" to "1") to start CMSER; falling edge (at least two bites are detected to transit from "1" to "0") to stop CMSER
ERRINJECT	input	one cycle high pulse to indicate that an error is required to

端口	I/O	描述
		be injected into a ECC block; the pulse must arise at the same clock cycle as the target ECC block is under decoding
ERRINJLOC	input	the location of error within a 72-bit ECC block 0_nnnnnn: the location of 64-bit data, for example: 0_000000: the ECC data bit[0] 0_111111: the ECC data bit[63] 1_xxxnnn: the location of 8-bit parity (x means "don't care), for example: 1_xxx000: the ECC parity bit[0] 1_xxx111: the ECC parity bit[7]
RUNNING	output	The high level of this signal indicates CMSER is running
CRCERR	output	one cycle high pulse to indicate the CRC error event
CRCDONE	output	one cycle high pulse to indicate the completion of CRC calculation and comparison
ECCCORR	output	one cycle high pulse to indicate the correctable ECC error event
ECCUNCORR	output	one cycle high pulse to indicate the uncorrectable ECC error event
ERRLOC	output	the location of ECC error
ECCDEC	output	The indication of ECC block decoding is running. 1: one ECC block is decoding at that clock cycle; 0: no ECC decoding at that clock cycle
DSRRD	output	one cycle high pulse to indicate the reading operation of DSR
DSRWR	output	one cycle high pulse to indicate the writing operation of DSR
ASRRESET	output	one cycle high pulse to indicate the reset of ASR
ASRINC	output	one cycle high pulse to indicate the increase of ASR address
REFCLK	output	the output reference clock for the generation of user CMSER interface design

原语例化

Verilog 例化:

```

CMSER uut (
    . RUNNING (RUNNING),
    . CRCERR (CRCERR),
    . CRCDONE (CRCDONE),
    . ECCCORR (ECCCORR),
    . ECCUNCORR (ECCUNCORR),
    . ERRLOC (ERRLOC),

```

```

        . ECCDEC (ECCDEC),
        . DSRRD (DSRRD),
        . DSRWR (DSRWR),
        . ASRRESET (ASRRESET),
        . ASRINC (ASRINC),
        . REFCLK (REFCLK),
        . CLK (CLK),
        . SEREN (SEREN),
        . ERRINJECT (ERRINJECT),
        . ERRINJLOC (ERRINJLOC)
    );

```

Vhdl 例化:

```

COMPONENT CMSER
    PORT (
        RUNNING,CRCERR,CRCDONE : OUT std_logic;
        ECCCORR,ECCUNCORR : OUT std_logic;
        ERRLOC : OUT std_logic_vector(27 downto 0);
        ECCDEC,DSRRD,DSRWR : OUT std_logic;
        ASRRESET,ASRINC,REFCLK : OUT std_logic;
        CLK,ERRINJECT : IN std_logic;
        SEREN : IN std_logic_vector(2 downto 0);
        ERRINJLOC : IN std_logic_vector(6 downto 0)
    );
END COMPONENT;

```

uut: CMSER

```

    PORT MAP (
        RUNNING => RUNNING,
        CRCERR => CRCERR,
        CRCDONE => CRCDONE,
        ECCCORR => ECCCORR,
        ECCUNCORR => ECCUNCORR,
        ERRLOC => ERRLOC,
        ECCDEC => ECCDEC,
        DSRRD => DSRRD,
        DSRWR => DSRWR,
        ASRRESET => ASRRESET,
        ASRINC => ASRINC,

```

```
REFCLK => REFCLK,  
CLK => CLK,  
ERRINJECT => ERRINJECT,  
SEREN => SEREN,  
ERRINJLOC => ERRINJLOC  
);
```

8.12 CMSERA

原语介绍

配置内存软错误恢复（CMSERA）可以持续监控配置内存以检测任何软错误，然后尝试在其能力范围内纠正它们。通过在用户设计后台逐帧读取配置内存，并进行 ECC 解码和 CRC 校验比对来实现。 通过将校正后的帧数据编程回 SRAM 来校正有限数量的错误位。

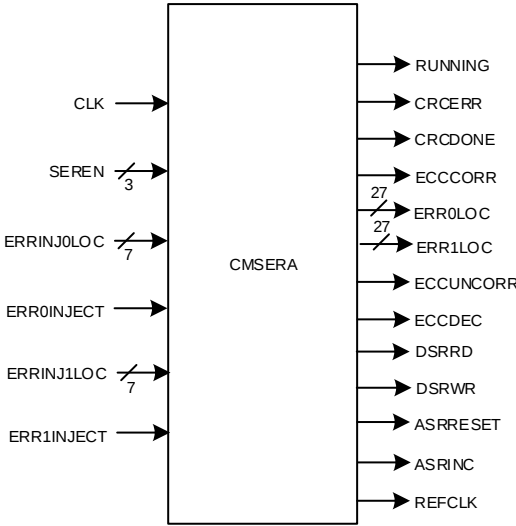
适用器件

表 8-23 CMSERA 适用器件

家族	系列	器件
晨熙®	GW5A	GW5A-25

端口示意图

图 8-17 CMSERA 端口示意图



端口介绍

表 8-24 CMSERA 端口介绍

端口	I/O	描述
CLK	input	Clock input
SEREN	input	(the critical signal using TMR scheme for error reduction)

端口	I/O	描述
		rising edge (at least two of three bits are detected to transit from "0" to "1") to start CMSER; falling edge (at least two bites are detected to transit from "1" to "0") to stop CMSER
ERR0INJECT	input	one cycle high pulse to indicate that an error is required to be injected into a ECC block; the pulse must arise at the same clock cycle as the target ECC block is under decoding
ERRINJ0LOC	input	the location of error 0 within a 72-bit ECC block 0_nnnnnn: the location of 64-bit data, for example: 0_000000: the ECC data bit[0] 0_111111: the ECC data bit[63] 1_xxxnnn: the location of 8-bit parity (x means "don't care"), for example: 1_xxx000: the ECC parity bit[0] 1_xxx111: the ECC parity bit[7]
ERR1INJECT	input	one cycle high pulse to indicate that an error is required to be injected into a ECC block; the pulse must arise at the same clock cycle as the target ECC block is under decoding
ERRINJ1LOC	input	the location of error 1 within a 72-bit ECC block 0_nnnnnn: the location of 64-bit data, for example: 0_000000: the ECC data bit[0] 0_111111: the ECC data bit[63] 1_xxxnnn: the location of 8-bit parity (x means "don't care"), for example: 1_xxx000: the ECC parity bit[0] 1_xxx111: the ECC parity bit[7]
RUNNING	output	The high level of this signal indicates CMSER is running
CRCERR	output	one cycle high pulse to indicate the CRC error event
CRCDONE	output	one cycle high pulse to indicate the completion of CRC calculation and comparison
ECCCORR	output	one cycle high pulse to indicate the correctable ECC error event
ECCUNCORR	output	one cycle high pulse to indicate the uncorrectable ECC error event
ERR0LOC	output	the location of ECC error 0
ERR1LOC	output	the location of ECC error 1
ECCDEC	output	The indication of ECC block decoding is running. 1: one ECC block is decoding at that clock cycle; 0: no ECC decoding at that clock cycle
DSRRD	output	one cycle high pulse to indicate the reading operation of DSR
DSRWR	output	one cycle high pulse to indicate the writing operation of DSR

端口	I/O	描述
ASRRESET	output	one cycle high pulse to indicate the reset of ASR
ASRINC	output	one cycle high pulse to indicate the increase of ASR address
REFCLK	output	the output reference clock for the generation of user CMSER interface design

原语例化

Verilog 例化:

```

CMSERA uut (
    . RUNNING (RUNNING),
    . CRCERR (CRCERR),
    . CRCDONE (CRCDONE),
    . ECCCORR (ECCCORR),
    . ECCUNCORR (ECCUNCORR),
    . ERR0LOC (ERR0LOC),
    . ERR1LOC (ERR1LOC),
    . ECCDEC (ECCDEC),
    . DSRRD (DSRRD),
    . DSRWR (DSRWR),
    . ASRRESET (ASRRESET),
    . ASRINC (ASRINC),
    . REFCLK (REFCLK),
    . CLK (CLK),
    . SEREN (SEREN),
    . ERR0INJECT (ERR0INJECT),
    . ERR1INJECT (ERR1INJECT),
    . ERRINJ0LOC (ERRINJ0LOC),
    . ERRINJ1LOC (ERRINJ1LOC)
);

```

Vhdl 例化:

```

COMPONENT CMSERA
PORT (
    RUNNING,CRCERR,CRCDONE : OUT std_logic;
    ECCCORR,ECCUNCORR : OUT std_logic;
    ERR0LOC,ERR1LOC : OUT std_logic_vector(26 downto
0);
    ECCDEC,DSRRD,DSRWR : OUT std_logic;

```

```

        ASRRESET,ASRINC,REFCLK : OUT std_logic;
        CLK,ERR0INJECT,ERR1INJECT : IN std_logic;
        SEREN : IN std_logic_vector(2 downto 0);
        ERRINJ0LOC,ERRINJ1LOC : IN std_logic_vector(6 downto
0)
    );
END COMPONENT;

uut: CMSERA
    PORT MAP (
        RUNNING => RUNNING,
        CRCERR => CRCERR,
        CRCDONE => CRCDONE,
        ECCCORR => ECCCORR,
        ECCUNCORR => ECCUNCORR,
        ERR0LOC => ERR0LOC,
        ERR1LOC => ERR1LOC,
        ECCDEC => ECCDEC,
        DSRRD => DSRRD,
        DSRWR => DSRWR,
        ASRRESET => ASRRESET,
        ASRINC => ASRINC,
        REFCLK => REFCLK,
        CLK => CLK,
        ERR0INJECT => ERR0INJECT,
        ERR1INJECT => ERR1INJECT,
        SEREN => SEREN,
        ERRINJ0LOC => ERRINJ0LOC,
        ERRINJ1LOC => ERRINJ1LOC
    );

```

