Gowin Primitives

User Guide

SUG283-3.4E, 04/30/2025

Disclaimer

GOWINSEMI assumes no liability and provides no warranty (either expressed or implied) and is not responsible for any damage incurred to your hardware, software, data, or property resulting from usage of the materials or intellectual property except as outlined in the GOWINSEMI Terms and Conditions of Sale. GOWINSEMI may make changes to this document at any time without prior notice. Anyone relying on this documentation should contact GOWINSEMI for the current documentation and errata.

Revision History

Date	Version	Description
04/20/2017	1.0E	Initial version published.
09/19/2017	1.1E	● GW1NR-4, GW1N-6, GW1N-9, GW1NR-9 devices added. ● ELVDS_IOBUF, TLVDS_IOBUF, BUFG, BUFS, OSC, IEM added. ● DSP primitive updated. ● Some ports of ODDR/ODDRC, IDDR_MEM, IDES4_MEM, IDES8_MEM, RAM16S1, RAM16S2、RAM16S4, RAM16SDP1, RAM16SDP2, RAM16SDP4, ROM16 updated. ● Some Attribute of OSC, PLL and DLLDLY updated; ● Some primitive instantiation updated. ● MIPI_IBUF_HS, MIPI_IBUF_LP, MIPI_OBUF, IDES16 and OSER16 updated. ● Some Attribute of CLKDIV updated.
04/12/2018	1.2E	Vhdl primitives instantiation added.
08/08/2018	1.3E	● GW1N-2B, GW1N-4B, GW1NR-4B, GW1N-6ES, GW1N-9ES, GW1NR-9ES, GW1NS-2,GW1NS-2C devices added. ● I3C_IOBUF, DHCEN added. ● User Flash added. ● EMPU added. ● Primitive name updated.
10/26/2018	1.4E	● GW1NZ-1, GW1NSR-2C devices added. ● OSCZ, FLASH96KZ added.
11/15/2018	1.5E	● GW1NSR-2 device added. ● GW1N-6ES, GW1N-9ES, GW1NR-9ES devices removed.
01/26/2019	1.6E	● GW1NS-2 supported by 8 frequency division of CLKDIV added. ● Removed GW1N-1 from the devices supported by TLVDS_TBUF/OBUF.
02/25/2019	1.7E	Removed GW1N-1 from the devices supported by TLVDS_IOBUF.
05/20/2019	1.8E	● GW1N-1S device added. ● MIPI_IBUF added. ● OSCH added. ● SPMI added. ● I3C added. ● Devices supported by OSC updated.
10/20/2019	1.9E	IOB, BSRAM, CLOCK modules updated.
11/28/2019	2.0E	● GSR and INV modules added in Miscellaneous. ● Devices supported updated. ● FLASH64KZ added and FLASH96KZ removed.
01/16/2020	2.1E	● IODELAYA, rPLL, PLLVR, CLKDIV2 added. ● DPB/DPX9B, SDPB/SDPX9B, rSDP/rSDPX9, rROM/rROMX9, pROM/pROMX9 added. ● EMCU, BANDGAP, FLASH64K added. ● IODELAY, PLL, CLKDIV, OSC, DQCE updated. ● Placement rule of FF、LATCH added. ● GW2A-55C added. ● GW1N-6/GW1N-9/GW1NR-9 disabled DP/DPX9, DPB/DPX9B.

Date	Version	Description
		● Notes of register added in IOLOGIC. ● GW1NZ-1 disabled 1, 2, 4, 8 bit width of DP/DPB and 9 bit width of DPX9/DPX9.
03/09/2020	2.2E	● GW1NS-2, GW1NS-2C, GW1NSR-2, GW1NSR-2C, GW1NSE-2C disable DP/DPX9 and DPB/DPX9B. ● OSCEN port description added in OSCF. ● PLL/rPLL/PLLVR parameter description updated.
06/08/2020	2.3E	● GW1N-2, GW1N-2B and GW1N-6 removed. ● GW1N-9C and GW1NR-9C added. ● IODELAYC, DHCENC and DCC added. ● Functional description of MIPI_IBUF added. ● MIPI_IBUF_HS, MIPI_IBUF_LP and DLL removed. ● Port diagram of LUT5 and MUX8 added. ● VCC and GND added. ● PLLVR, FLASH64K, BUFS, EMPU and CLKDIV2 updated. ● DP/DPX9, ROM/ROMX9, SDP/SDPX9, rSDP/rSDPX9, rROM/rROMX9 and PLL removed.
09/11/2020	2.4E	The description of IP invoking of ADC, BANDGAP, SPMI and I3C modules added.
12/30/2020	2.5E	The description of Chapter 2 CFU updated.
07/22/2021	2.6E	● activeFlash added. ● Help information on IP GUI removed and figures updated. ● GW1NZ-1C, GW1N-2, GW1N-2B, GW1N-1P5, GW1N-1P5B, GW1NR-2, GW1NR-2B added.
10/28/2021	2.7E	The description of activeFlash updated.
07/22/2022	2.8E	● OTP module added. ● SAMB module added.
10/28/2022	2.9E	MCU, USB20_PHY, and ADC removed.
04/20/2023	3.0E	Arora V devices and related descriptions added.
07/14/2023	3.1E	Shutdown by VCCEN option removed from SPMI configuration.
12/29/2023	3.2E	Introduction to Arora V OTP function added.
10/25/2024	3.3E	Descriptions of 8.9 OTP updated.
04/30/2025	3.4E	● Devices supported for OTP updated. ● SAMBA module description added. ● CMSERB module description added.

Contents

List of Figures

List of Tables

1 IOB

IOB includes input/output buffer (IO Buffer) and input/output logic (IO Logic). For the IO Buffer and IO Logic primitives of Arora V decives, see *UG304, Arora V Programmable IO (GPIO) User Guide*; for those of the other devices, see *UG289, Gowin Programmable IO(GPIO) User Guide*.

2 CFU

The Configurable Function Unit (CFU) and the Configurable Logic Unit (CLU) are two basic units for FPGA core of GOWINSEMI. Each unit consists of four configurable logic sections (CLS) and its configurable routing unit (CRU). Configurable logical sections in CLU cannot be configured as SRAM, but as basic logic, ALU, and ROM. The configurable logic sections in the CFU can be configured as basic logic, ALU, SRAM, and ROM depending on the applications. For CFU primitives of Arora V decives, see *UG303, Arora V Configurable Function Unit (CFU) User Guide*, *UG288;* for those of the other devices, see *Gowin Configurable Function Unit (CFU) User Guide*.

3 Memory

Gowin FPGA products provide abundant memory resources, including Block Static Random Access Memory (BSRAM) and Shadow Static Random Access Memory (SSRAM). For BSRAM and SSRAM primitives of Arora V decives, see *UG300, Arora V BSRAM & SSRAM User Guide*; for those of the other devices,see *UG285, Gowin BSRAM & SSRAM User Guide*.

4 DSP

Gowin FPGA products provide abundant DSP resources, which can meet the demand of high-performance digital signal processing. For DSP primitives of Arora V devices, see *UG305, Arora V Digital Signal Processing (DSP) User Guide*; for those of the other devices, see *UG287, Gowin Digital Signal Processing (DSP) User Guide*.

5 Clock

Gowin FPGA products provide dedicated global clock (GCLK, including PCLK and SCLK) directly connected to all resources of the devices. In addition to GCLK, PLL, HCLK and bidirectional data strobe circuit (DQS) for DDR Memory are also available. For clock primitives of Arora V devices, see *UG306, Arora V Clock User Guide*; for those of the other devices, see *UG286, Gowin Clock User Guide*.

6 User Flash

Gowin LittleBee Family FPGA products provide User Flash. Different series of devices support different sizes of Flash. For the Flash primitives, see *UG295, Gowin Flash User Guide*.

7 EMPU

7.1 EMCU

Primitive

ARM Cortex-M3 Microcontroller Unit (EMCU) is a micro-processor based on the ARM Cortex-m3. The 32-bit AHB/APB bus is used. It supports the functions of two UARTs, two Timers and Watchdog. It also provides 16-bit GPIO, two UARTs and JTAGs, six User Interrupt interfaces, AHB Flash read interface, AHB Sram read/write interface, two AHB bus extension interfaces, and one APB bus extension interface.

Devices Supported

Table 7-1 EMCU Devices Supported

Family	Series	Device
LittleBee	GW1NS	GW1NS-4C
	GW1NSR	GW1NSR-4C
	GW1NSER	GW1NSER-4C

Port Diagram

Figure 7-1 Port Diagram of EMCU

Port Description

Table 7-2 Port Description

Port	I/O	Description
FCLK	input	Free running clock
PORESETN	input	Power on reset
SYSRESETN	input	System reset
RTCSRCCLK	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.
DAPSWDITMS	input	Debug TMS

Port	I/O	Description
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
INITEXP0HRDATA[31:0]	output	INITEXP0, HRDATA

Port	I/O	Description
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

Primitive Instantiation

Verilog Instantiation:

```
MCU u_sse050_top_syn (
.FCLK(fclk),
.PORESETN(poresetn),
.SYSRESETN(sysresetn),
.RTCSRCCLK(rtcsrcclk),
.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(dapjtagnsw),
 .TPIUTRACEDATA(tpiutracedata[3:0]),
 .TPIUTRACECLK(tpiutraceclk),
 .FLASHERR(flasherr),
  .GPINT(gpint),
 .FLASHINT(flashint)
);
```

Vhdl Instantiation:

```
COMPONENT EMCU
```

```
    PORT(
FCLK:IN std_logic;
PORESETN:IN std_logic;
SYSRESETN:IN std_logic;
RTCSRCCLK: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;
        DAPTDO: OUT std_logic;
        DAPJTAGNSW: OUT std_logic;
        DAPNTDOEN: OUT std_logic;
        TPIUTRACECLK: OUT std_logic;
    );
        END COMPONENT;

        uut: EMCU
          PORT MAP (
        FCLK=> fclk;
        PORESETN=> poresetn;
        SYSRESETN=> sysresetn;
        RTCSRCCLK=> rtcsrcclk;
        UART0RXDI=> uart0rxdi;
        UART1RXDI=> uart1rxdi;
        CLK=>clk,
        RESET=>reset,
        IOEXPINPUTI=>ioexpinputi,
        SRAM0RDATA=>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=>initexp0mematttr,
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=>dapjtagnsw,
DAPNTDOEN=>dapntdoen,
TPIUTRACECLK=>tpiutraceclk );
```

8 Miscellaneous

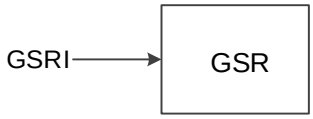
8.1 GSR

Primitive

Global Reset/Set (GSR) is the global set/reset module, which can realize the global set/reset function, active-low. The default is connected to high level, if you want to control dynamically, you can connect an external signal to pull down to achieve set/reset of registers and other modules.

Port Diagram

Figure 8-1 GSR Port Diagram



Port Description

Table 8-1 Port Description

Name	I/O	Description
GSRI	Input	GSR input, active-low

Primitive Instantiation

Verilog instantiation:

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

Vhdl instantiation:

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

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

8.2 INV

Primitive

Inverter (INV)

Port Diagram

Figure 8-2 INV Port Diagram

Port Description

Table 8-2 Port Description

Name	I/O	Description
I	Input	INV data input
O	Output	INV data output

Primitive Instantiation

Verilog instantiation:

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

Vhdl instantiation:

```
COMPONENT INV
      PORT (
            O:OUTPUT std_logic;
            I:IN std_logic

      );
END COMPONENT;
uut:INV
        PORT MAP(
```

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

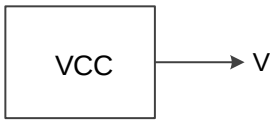
8.3 VCC

Primitive

VCC is the logic high level generator.

Port Diagram

Figure 8-3 VCC Port Diagram



Port Description

Table 8-3 Port Description

Name	I/O	Description
V	Output	VCC output

Primitive Instantiation

Verilog instantiation:

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

Vhdl instantiation:

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

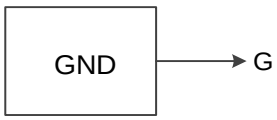
8.4 GND

Primitive

GND is the logic low level generator.

Port Diagram

Figure 8-4 GND Port Diagram



Port Description

Table 8-4 Port Description

Name	I/O	Description
G	Output	GND output

Primitive Instantiation

Verilog instantiation:

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

Vhdl instantiation:

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

8.5 BANDGAP

Primitive

BANDGAP is used to provide constant voltage and current for certain modules in the chip. If BANDGAP is turned off, the OSC, PLL, FLASH and other modules will no longer work, which can reduce the power consumption.

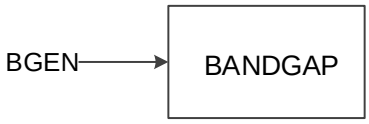
Supported Devices

Table 8-5 BANDGAP Devices Supported

Family	Series	Device
LittleBee	GW1NZ	GW1NZ-1, GW1NZ-2
	GW1N	GW1N-2,GW1N-2B,GW1N-1P5,GW1N-1P5B
	GW1NR	GW1NR-2,GW1NR-2B

Port Diagram

Figure 8-5 BANDGAP Port Diagram



Port Description

Table 8-6 Port Description

Name	I/O	Description
BGEN	Input	BANDGAP enable signal, active-high.

Primitive Instantiation

Verilog Instantiation:

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

Vhdl Instantiation:

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

Invoke IP

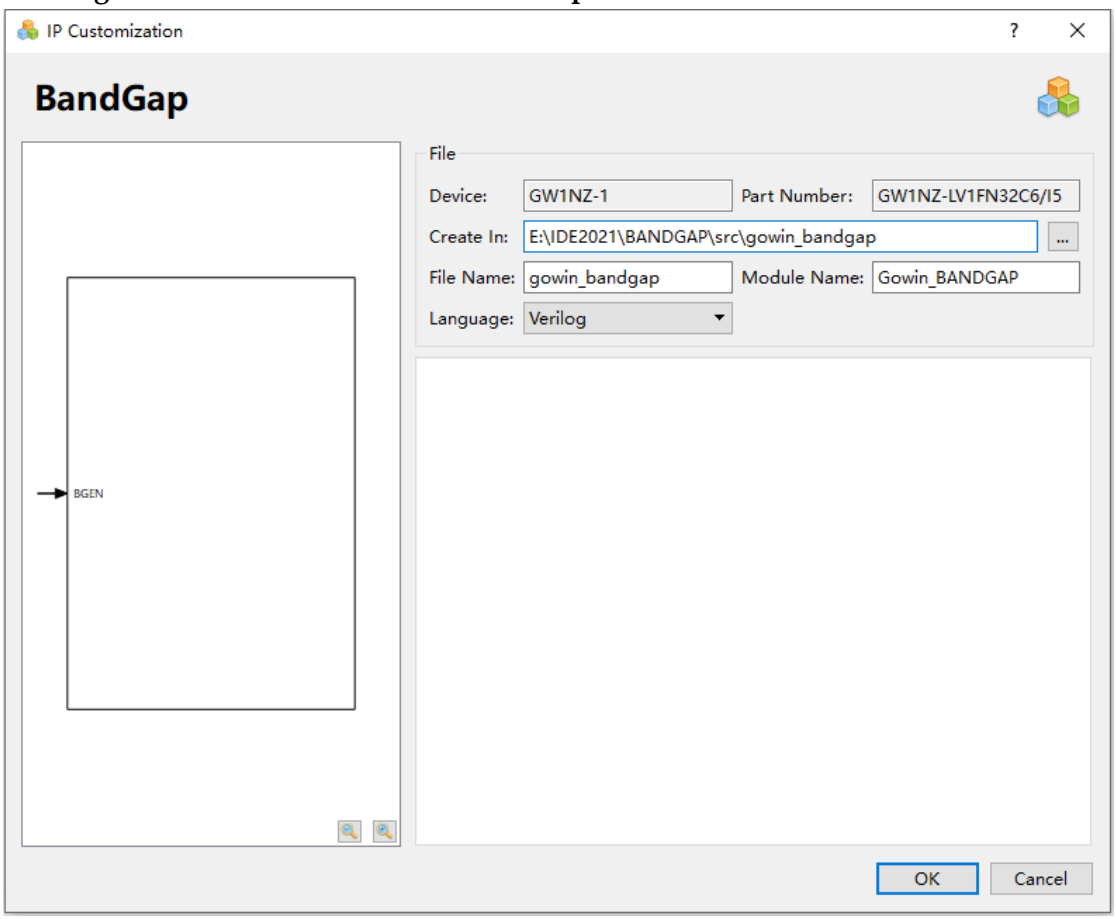
Click "BandGap" on the IP Core Generator, and a brief introduction to the BandGap will be displayed.

IP Configuration

Double-click the "BandGap" to open the "IP Customization" window.

This displays the "File", "Options", and ports diagram, as shown in Figure 8-6.

Figure 8-6 IP Customization of BandGap



1. File
   The File displays the basic information related to BandGap.
   - Device: Selected device
   - Part Number: Selected Part Number
   - Create In: The target path; you can reedit in the textbox or select a path by clicking the button.
   - File Name: The file name; you can reedit in the textbox.
   - Module Name: The module name; you can reedit in the textbox. The module name can not be the same as the primitive name. If it is the same, an error prompt will pop up.
   - Language: Verilog and VHDL
2. Ports Diagram
   The ports diagram is based on the current IP Core configuration, as shown in Figure 8-6.

Generated Files

After configuration, it will generate three files that are named after the

"File Name".

- "gowin_bandgap.v " file is a complete Verilog module to generate instance BandGap, and it is generated according to the IP configuration;
- "gowin_bandgap_tmp.v" is the instance template file;
- "gowin_bandgap.ipc" file is IP configuration file. The user can load the file to configure the IP.

Note!

If VHDL is selected as the hardware description language, the first two files will be named with .vhd suffix.

8.6 SPMI

Primitive

System Power Management Interface (SPMI) is a two-wire serial interface, which can be used to dynamically control the internal power supply of the on-chip system.

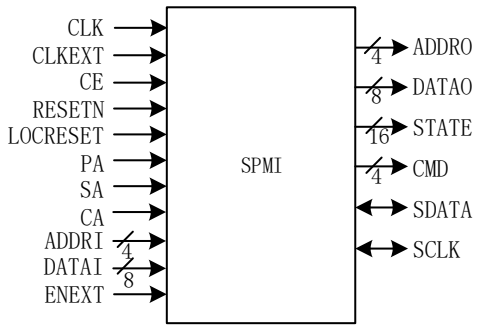
Devices Supported

Table 8-7 SPMI Devices Supported

Family	Series	Device
LittleBee	GW1NZ	GW1NZ-1, GW1NZ-1C

Port Diagram

Figure 8-7 SPMI Port Diagram



Port Description

Table 8-8 Port Description

Port	I/O	Description
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

Primitive Instantiation

Verilog Instantiation:

```
SPMI uut (
        .ADDRO(addro),
        .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(addri),
        .DATAI(datai),
```

```
        .CLKEXT(clkext),
        .ENEXT(enext)
    );
```

Vhdl Instantiation:

```
    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
    );
```

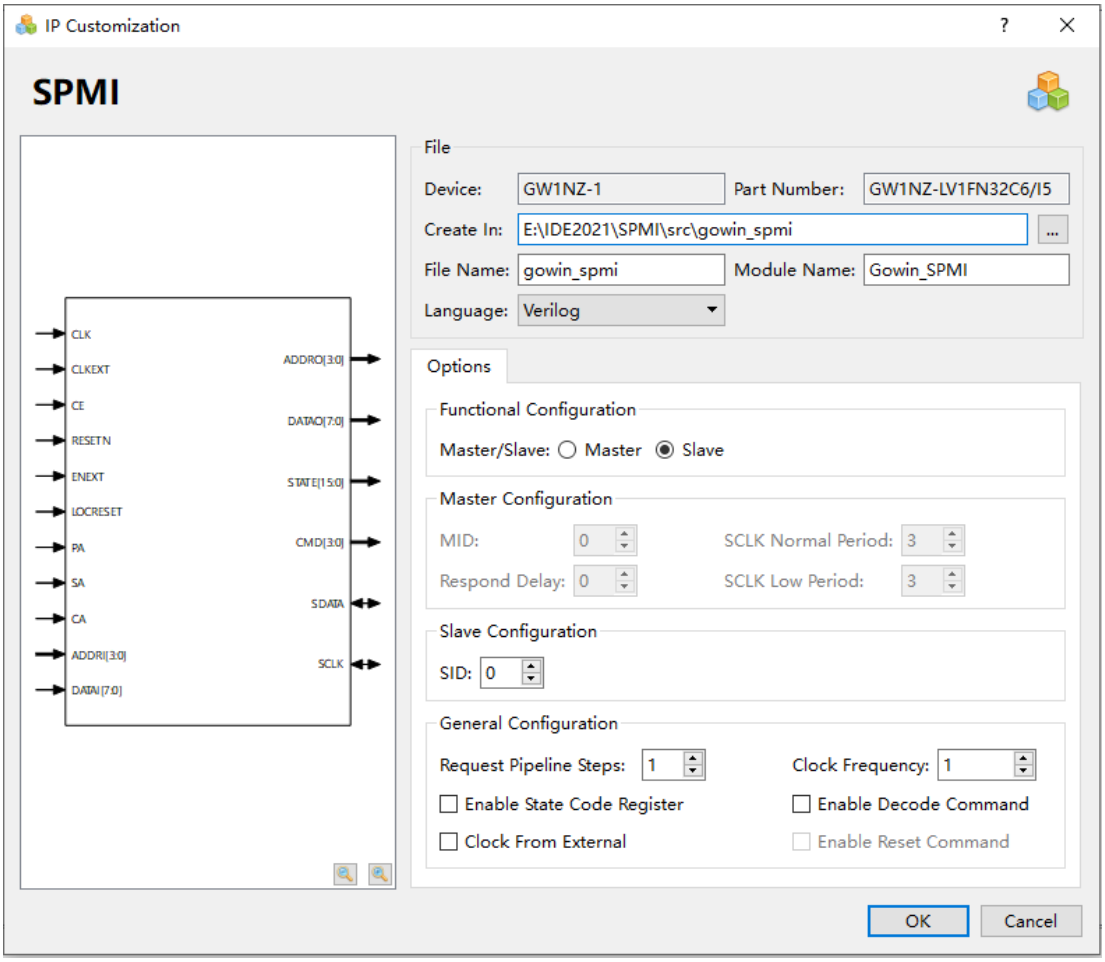
Invoke IP

Click "SPMI" on the IP Core Generator, and a brief introduction to the SPMI will be displayed.

IP Configuration

Double-click the "SPMI" to open the "IP Customization" window. This displays the "File", "Options", and ports diagram, as shown in Figure 8-8.

Figure 8-8 IP Customization of SPMI



1. File
   The File displays the basic information related to SPMI.The SPMI file configuration box is similar to that of BANDGAP. For the details, please see 8.5 BANDGAP.

2. Options
   - The Options is used to configure SPMI by users, as shown in Figure 8-8.
   - Functional Configuration：
     - Master/Slave: Set SPMI as Master or Slave.
   - Master Configuration:
     - MID: Master ID. The range is 0-3, and default value is 0.
     - Respond Delay: Set the response delay.
     - SCLK Normal Period: Set SCLK period in normal mode.
   - SCLK Normal Period: Set SCLK period in low mode.
   - Slave Configuration: SID: Slave ID.
   - General configuration:
     - Enable State Code Register: Enable or disable the state code register. If "Enable State Code Register" is checked, the output state code will pass a register.
     - Request Pipeline Steps: Set the sampling delay step of the request signal.
     - Enable Decode Command: Enable or disable decode. If "Enable Decode Command" is checked, SPMI will decode the reset, sleep, shutdown, and wakeup.
     - Enable Decode Command: Enable or disable reset.
     - Clock From External: Enable or disable the external clock.
     - Clock Frequency: System clock frequency.
3. Ports Diagram
   The ports diagram is based on the current IP Core configuration, as shown in Figure 8-8.

Generated Files

After configuration, it will generate three files that are named after the "File Name".

- "gowin_spmi.v " file is a complete Verilog module to generate instance SPMI, and it is generated according to the IP configuration;
- "gowin_spmi_tmp.v" is the instance template file;
- "gowin_spmi.ipc " file is IP configuration file. The user can load the file to configure the IP.

Note!

If VHDL is selected as the hardware description language, the first two files will be named with .vhd suffix.

8.7 I3C

Primitive

Improved Inter Integrated Circuit (I3C) is a two-wire bus with the key features of I^2C and SPI, which can effectively reduce the physical ports of integrated circuit, support the advantages of low power, high data rate and other existing port protocols.

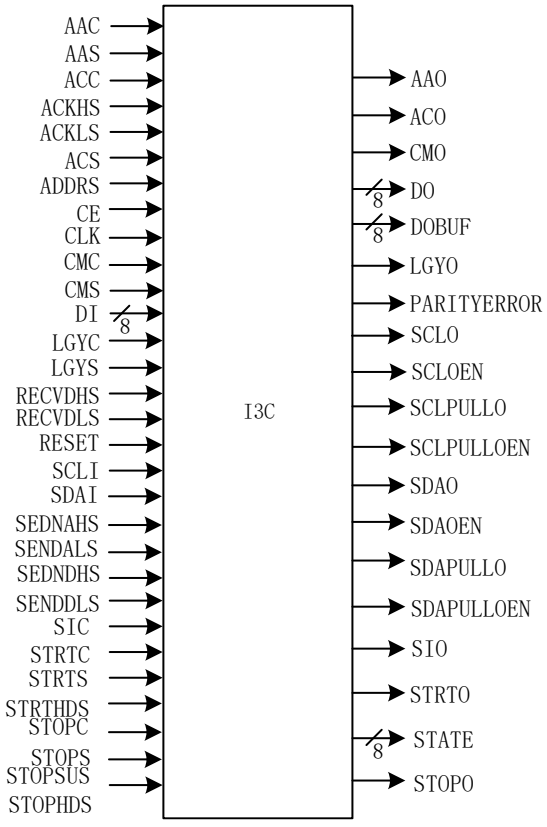
Devices Supported

Table 8-9 I3C Devices Supported

Family	Series	Device
LittleBee	GW1NZ	GW1NZ-1, GW1NZ-1C

Port Diagram

Figure 8-9 I3C Port Diagram



Port Description

Table 8-10 Port Description

Port	I/O	Description
CE	input	Clock Enable
RESET	input	Reset input
CLK	input	Clock input

Port	I/O	Description
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.
SENDDLS	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
SIO	output	Interrupt to output the identity bit

Port	I/O	Description
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

Primitive Instantiation

Verilog Instantiation:

```
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(addrs),
        .DI(di),
        .SDAI(sdai),
        .SCLI(scli),
        .CE(ce),
        .RESET(reset),
        .CLK(clk)
    );
```

Vhdl Instantiation：

```
    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 DOWNTO 0);
DOUT: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
STATE: OUT STD_LOGIC_VECTOR(7 DOWNTO 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;
SENDDHS: IN STD_LOGIC;
SENDDLS: IN STD_LOGIC;
RECVDHS: IN STD_LOGIC;
```

```
            RECVDLS: IN STD_LOGIC;
            ADDRS: 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
    );
```

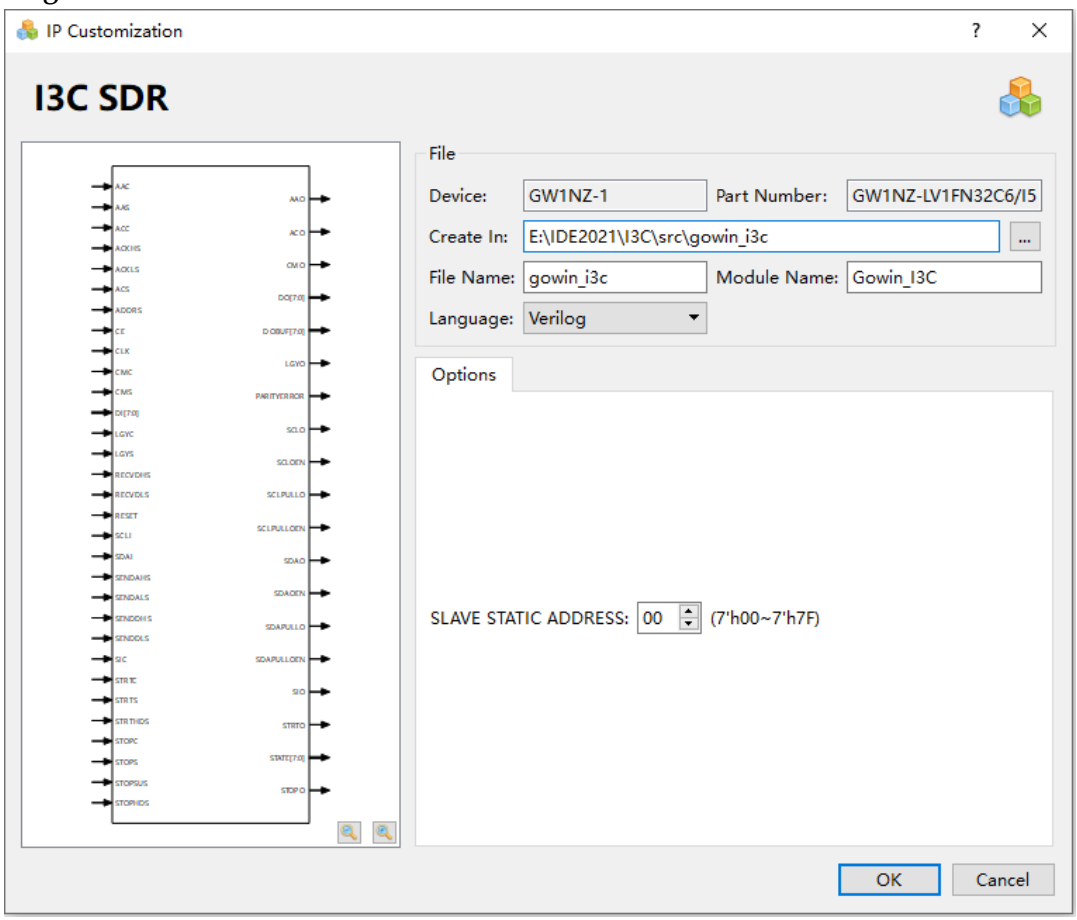
Invoke IP

Click "I3C > I3C SDR" on the "IP Core Generator" page. A brief introduction to the I3C SDR will be displayed.

Configure IP

Double-click "I3C SDR", and the "IP Customization" window pops up. This displays the "File", "Options", and port diagram, as shown in Figure 8-10.

Figure 8-10 IP Customization of I3C



1. File
   The File displays the basic information related to the I3C.The I3C file configuration is similar to BANDGAP. For the details, please see 8.5 BANDGAP.
2. Options
   - The Options is used to configure I3C by users, as shown in Figure 8-10.
   - SLAVE STATIC ADDRESS: Specify the static address of the Slave.
3. Ports Diagram
   The ports diagram is based on the IP Core configuration, as shown in Figure 8-10.

Generated Files

After configuration, it will generate three files that are named after the "File Name".

- "gowin_i3c.v " file is a complete Verilog module to generate instance I3C, and it is generated according to the IP configuration;
- "gowin_i3c_tmp.v" is the instance template file;
- "gowin_i3c.ipc " file is IP configuration file. You can load the file to configure the IP.

Note!

If VHDL is selected as the hardware description language, the first two files will be named with .vhd suffix.

8.8 activeFlash

Primitive

Activate embedded SPI Nor Flash module. When activeFlash instantiated, I_active_flash_sclk must be a clock signal and I_active_flash_holdn needs to be pulled up.

Note!

When activeFlash instantiated, the user design will add 14 LUTs and 10 REGs.

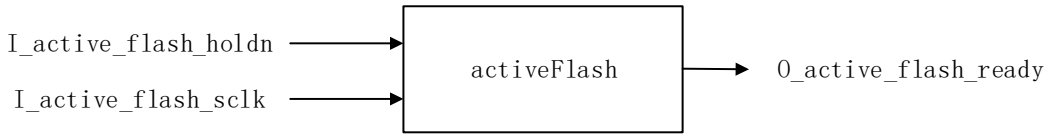
Devices Supported

Table 8-11 activeFlash Devices Supported

Family	Series	Device
Arora	GW2AN	GW2AN-18X,GW2AN-9X

Port Diagram

Figure 8-11 activeFlash Port Diagram



Port Description

Table 8-12 Port Description

Port	I/O	Description
I_active_flash_holdn	input	Clock hold signal; high input activates Flash.
I_active_flash_sclk	input	Clock input clock
O_active_flash_ready	output	Ready signal; high indicates Flash has been activated.

Primitive Intantiation

Verilog Instantiation:

```
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 Instantiation:

```
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
    );
```

Conditions

When one of the following conditions is met, the activeFlash module needs to be instantiated in the user design.

1. When Configuration - Bitstream background programming in Gowin Software is set to I2C/JTAG/SSPI/QSSPI, activeFlash needs to be instantiated.
2. When Configuration - Bitstream background programming is set to OFF and the SPI Nor Flash Interface IP is used, activeFlash needs to be instantiated.

8.9 OTP

Primitive

One Time Programming (OTP). Chip product information is stored in the factory area, and you can read the chip product information using OTP.

Devices Supported

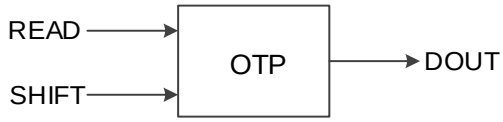
Table 8-13 OTP Devices Supported

Family	Series	Device
Arora	GW2AN	GW2AN-18X, GW2AN-9X
	GW5AT	GW5AT-138, GW5AT-75, GW5AT-60, GW5AT-15
	GW5A	GW5A-25, GW5A-138, GW5A-60
	GW5AS	GW5AS-25, GW5AS-138
	GW5AST	GW5AST-138
	GW5AR	GW5AR-25
	GW5ART	GW5ART-15

Port Diagram

Figure 8-12 GW2AN OTP Port Diagram

Figure 8-13 Arora V OTP Port Diagram



Port Description

Table 8-14 GW2AN OTP Port Description

Port	I/O	Description
SCLK	input	Serial input clock; move data out to DOUT at falling edge; 2.5MHz clock recommended for reading data.
CSB	input	Chip selected signal, active-low.
DOUT	output	Serial data output

Table 8-15 Arora V OTP Port Description

Port	I/O	Description
CLK	input	Clock signal
READ	input	pulse signal; load the corresponding 128-bit efuse register data into the shift register, active-high.
SHIFT	input	Level signal; shift the shift register data to the output interface, active-high.
DOUT	output	Data output for reading chip DNA information/user information

Parameter Description

Table 8-16 Arora V OTP Parameter Description

Parameter	Value Range	Default	Description
MODE	2'b00, 2'b01, 2’b10	2'b01	OTP mode selection ● 2'b01: Read chip DNA information ● 2'b00: Read user information ● 2'b10: Read user control information

Arora V OTP Function Introduction

Arora V OTP can read chip DNA information, user information, and

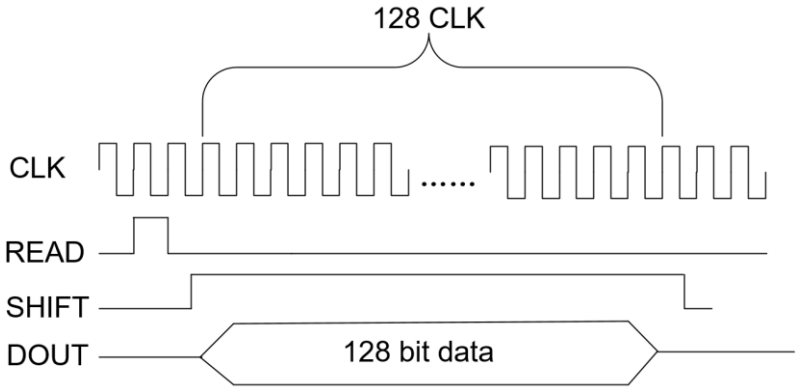
user control information from the User efuse region. The descriptions of the 128-bit User efuse region are shown in Table 8-17.

Table 8-17 Descriptions of Arora V OTP User Efuse Region

Region Position	Region Category	Description	
127~112bit (16bits)	user_misc	reserved	
111~96bit (16bits)	User Control Information	reserved	-
		107bit prgm_user_control_lock	1: can't use jtag program user control region
		106bit rd_user_misc_lock	1: can't read user_misc domain
		105bit prgm_user_misc_lock	1: lock user_misc domain
		104bit lock_sel_key_r	0: decryption module select key; 1: decryption module select key2
		103bit prgm_rd_dna_lock	1: can't use jtag program and readback 64bits DNA region
		102bit rd_fuse_user_lock	1: can't use jtag readback 32bits fuse user region
		101bit prgm_fuse_user_lock	1: can't use jtag program 32bits fuse user region
		100bit rd_key2_lock	1: can't use jtag readback key2
		99bit prgm_key2_lock	1: can't use jtag program key2 again
		98bit rd_key_lock	1: can't use jtag readback key
		97bit prgm_key_lock	1: can't use jtag program key again
		96bit cfg_aes_only	1: only can use encryption bitstream
95~32bit (64bits)	DNA Information	64 bits DNA Info	store DNA Info
31~0bit (32bits)	User Information	32 bits User defined	-

Interface Timing

Figure 8-14 Arora V OTP Interface Timing Diagram



As shown in Figure 8-14, data is loaded using READ (greater than or equal to one clock cycle); then READ needs to be pulled down, SHIFT pulled up, and data is shifted out from DOUT one by one in the form of 128 bits (with the lowest bit of the 128 bits first); one bit of data is output in one clock.

The output valid data bits in different MODEs are as follows:

- 00: Output 128 bits, with the lower 32 bits being valid data.
- 01: Output 128 bits, with bits 95 to 32 (64 bits in total) being valid data.
- 10: Output 128 bits, with bits 111 to 96 (16 bits in total) being valid data.

Primitive Instantiation

GW2AN OTP Instantiation

Verilog Instantiation:

```
OTP uut (
      . SCLK (SCLK),
      .CSB (CSB),
      . DOUT (DOUT)
);
```

Vhdl Instantiation：

```
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 Instantiation

Verilog Instantiation：

```
OTP uut (
      . READ(READ),
      .SHIFT(SHIFT),
      .CLK(CLK),
      . DOUT (DOUT)
);
defparam uut.MODE=2'b01;
```

Vhdl Instantiation：

```
COMPONENT OTP
  GENERIC (
    MODE : bit_vector := “01”
    );
    PORT(
    READ:IN std_logic;
          SHIFT:IN std_logic;
          CLK: IN std_logic;
          DOUT:OUT std_logic
    );
END COMPONENT;
uut: OTP
  GENERIC MAP (
          MODE =>”01”
    )
     PORT MAP (
    READ => READ,
          CLK => CLK,
          SHIFT => SHIFT,
          DOUT => DOUT
    );
```

8.10 SAMB

Primitive

SPI Address for Multi Boot (SAMB) is for address selection in Multi Boot mode, and you can select static and dynamic addresses.

Devices Supported

Table 8-18 SAMB Devices Supported

Family	Series	Device
Arora	GW2AN	GW2AN-18X, GW2AN-9X
	GW5AT	GW5AT-138, GW5AT-75
	GW5A	GW5A-25, GW5A-138
	GW5AS	GW5AS-138, GW5AS-25
	GW5AST	GW5AST-138
	GW5AR	GW5AR-25

Port Diagram

Figure 8-15 GW2AN SAMB Port Diagram

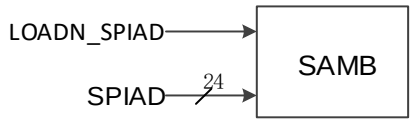
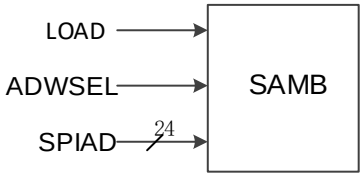


Figure 8-16 Arora V SAMB Port Diagram



Port Description

Table 8-19 GW2AN SAMB Port Description

Port	I/O	Description
LOADN_SPIAD	input	Used to select the static SPI address or the location specified by the dynamic SPI address signal. The static SPI address is selected when it is high, and the dynamic SPI address signal SPIAD is selected when it is low.
SPIAD[23:0]	input	SPI address signal

Table 8-20 Arora V SAMB Port Description

Port	I/O	Description
LOAD	input	Used to select the static SPI address or the location specified by the dynamic SPI address signal. The static SPI address is

Port	I/O	Description
		selected when it is low, and the dynamic SPI address signal SPIAD is selected when it is high.
SPIAD[23:0]	input	SPI address signal
ADWSEL	Input	Address width selection 0: 24-bit address 1: 32-bit address (the lower 8 bits are zeroed)

Parameter Description

Table 8-21 Arora V SAMB Parameter Description

Parameter	Value Range	Default	Description
MODE	2'b00, 2'b01, 2'b10, 2'b11	2'b00	SAMB mode selection ● 2'b00: Normal mode ● 2'b01: Fast mode ● 2'b10: Dual mode ● 2'b11: Quad mode

Primitive Instantiation

GW2AN SAMB Instantiation

Verilog Instantiation:

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

Vhdl Instantiation：

```
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 Instantiation

Verilog Instantiation：

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

Vhdl Instantiation：

```
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 SAMBA

Primitive

SPI Address for Multi Boot A (SAMBA) is used for address selection in Multi Boot mode, allowing the choice between static and dynamic addresses.

Devices Supported

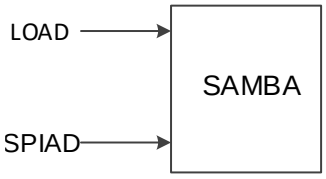
Table 8-22 SAMBA Devices Supported

Family	Series	Device
Arora	GW5AT	GW5AT-60, GW5AT-15
	GW5A	GW5A-60
	GW5ART	GW5ART-15

Family	Series	Device
	GW5ANT	GW5ANT-15
	GW5ANRT	GW5ANRT-15

Port Diagram

Figure 8-17 Arora V SAMBA Port Diagram



Port Description

Table 8-23 Arora V SAMBA Port Description

Port	I/O	Description
LOAD	input	Used to select a static SPI address or the location specified by the dynamic SPI address signal. When the signal is low, the static SPI address is selected. When the signal is high, the dynamic SPI address signal (SPIAD) is selected.
SPIAD	input	SPI address signal

Parameter Description

Table 8-24 Arora V SAMBA Parameter Description

Parameter	Value Range	Default	Description
MODE	2'b00, 2'b01, 2'b10, 2'b11	2'b00	SAMBA mode selection ● 2'b00: Normal mode ● 2'b01: Fast mode ● 2'b10: Dual mode ● 2'b11: Quad mode

Primitive Instantiation

Arora V SAMBA Instantiation

Verilog Instantiation:

```
SAMBA uut (
        . LOAD (LOAD),
        . SPIAD (SPIAD)
);
defparam uut.MODE=2'b00;
```

Vhdl Instantiation:

```
COMPONENT SAMBA
```

```
  GENERIC (
    MODE : bit_vector := "00"
      );
       PORT(
    LOAD:IN std_logic;
            SPIAD:IN std_logic
    );
END COMPONENT;
uut: SAMBA
  GENERIC (
    MODE => "00"
      );
  PORT MAP (
    LOAD => LOAD,
            SPIAD => SPIAD
     );
```

8.12 CMSER

Primitive

Configuration Memory Soft Error Recovery (CMSER) can continuously monitors configuration memory to detect any soft error and then attempts to correct them within its capabilities. This is accomplished by reading the configuration memory frame by frame in the user design backend and performing ECC decoding and CRC. A limited number of error bits are corrected by programming the corrected frame data back into SRAM.

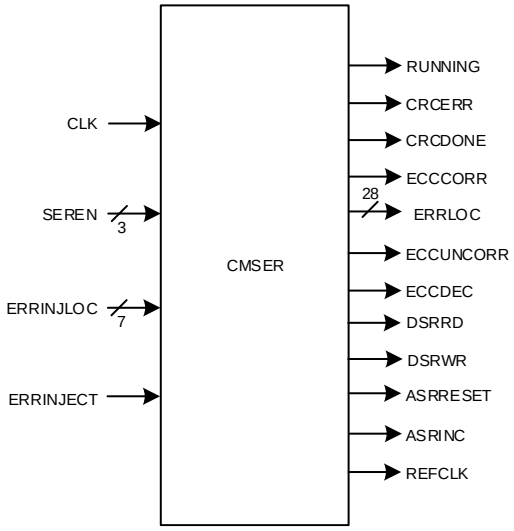
Devices Supported

Table 8-25 CMSER Devices Supported

Family	Series	Device
Arora	GW5AT	GW5AT-138, GW5AT-75
	GW5A	GW5A-138
	GW5AS	GW5AS-138
	GW5AST	GW5AST-138

Port Diagram

Figure 8-18 CMSER Port Diagram



Port Description

Table 8-26 CMSER Port Description

Port	I/O	Description
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 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

Port	I/O	Description
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

Primitive Intantiation

Verilog Intantiation：

```
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 Instantiation：

```
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
    );
```

8.13 CMSERA

Primitive

Configuration Memory Soft Error Recovery A (CMSERA) can continuously monitors configuration memory to detect any soft error and then attempts to correct them within its capabilities. This is accomplished by reading the configuration memory frame by frame in the user design backend and performing ECC decoding and CRC. A limited number of error bits are corrected by programming the corrected frame data back into SRAM.

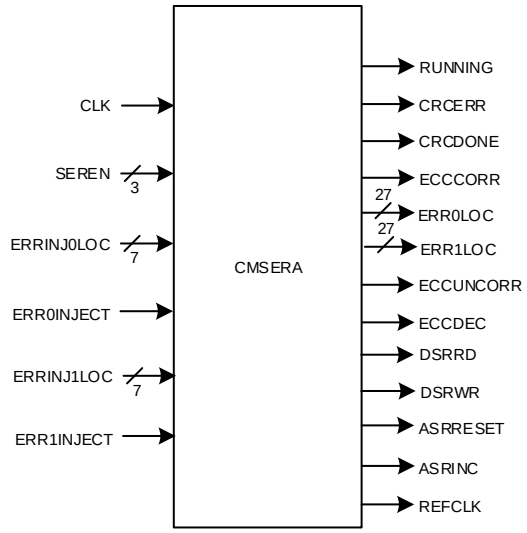
Devices Supported

Table 8-27 CMSERA Devices Supported

Family	Series	Device
Arora	GW5A	GW5A-25
	GW5AR	GW5AR-25
	GW5AS	GW5AS-25

Port Diagram

Figure 8-19 CMSERA Port Diagram



Port Description

Table 8-28 CMSERA Port Description

Port	I/O	Description
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
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]

Port	I/O	Description
		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
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

Primitive Intantiation

Verilog Intantiation：

```
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 Instantiation：

```
    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
    );
```

8.14 CMSERB

Primitive

Configuration Memory Soft Error Recovery B (CMSERB) can continuously monitors configuration memory to detect any soft error and then attempts to correct them within its capabilities. This is accomplished by reading the configuration memory frame by frame in the user design backend and performing ECC decoding and CRC. A limited number of error bits are corrected by programming the corrected frame data back into SRAM.

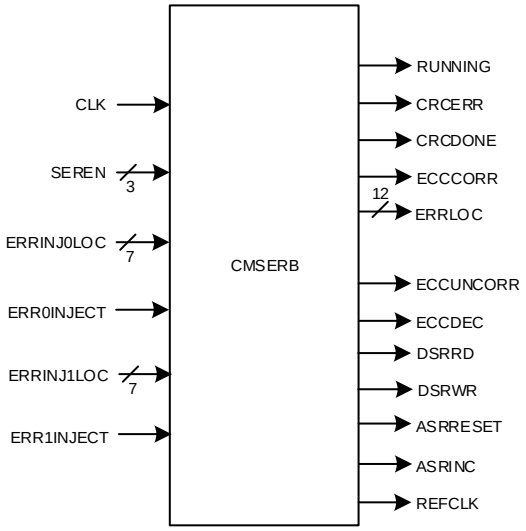
Devices Supported

Table 8-29 CMSERB Devices Supported

Family	Series	Device
Arora	GW5AT	GW5AT-60, GW5AT-15
	GW5A	GW5A-60
	GW5ART	GW5ART-15
	GW5ANT	GW5ANT-15
	GW5ANRT	GW5ANRT-15

Port Diagram

Figure 8-20 CMSERB Port Diagram



Port Description

Table 8-30 CMSERB Port Description

端口	I/O	Description
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
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]

端口	I/O	Description
		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

Primitive Instantiation

Verilog Instantiation:

```
CMSERB 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),
        . ERR0INJECT (ERR0INJECT),
      . ERR1INJECT (ERR1INJECT),
        . ERRINJ0LOC (ERRINJ0LOC),
        . ERRINJ1LOC (ERRINJ1LOC)
    );
```

Vhdl Instantiation:

```
    COMPONENT CMSERB
        PORT (
            RUNNING,CRCERR,CRCDONE : OUT std_logic;
            ECCCORR,ECCUNCORR : OUT std_logic;
            ERRLOC : OUT std_logic_vector(12 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: CMSERB
        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,
            ERR0INJECT => ERR0INJECT,
```

```
        ERR1INJECT => ERR1INJECT,
        SEREN => SEREN,
        ERRINJ0LOC => ERRINJ0LOC,
        ERRINJ1LOC => ERRINJ1LOC
);
```


GOWIN
PROGRAMMING FOR THE FUTURE