




Gowin プリミティブ ユーザーガイド

SUG283-3.3J, 2024-10-25

著作権について(2024)

著作権に関する全ての権利は、**Guangdong Gowin Semiconductor Corporation** に留保されています。

 **GOWIN高云**、Gowin、及びLittleBeeは、当社により、中国、米国特許商標庁、及びその他の国において登録されています。商標又はサービスマークとして特定されたその他全ての文字やロゴは、それぞれの権利者に帰属しています。何れの団体及び個人も、当社の書面による許可を得ず、本文書の内容の一部もしくは全部を、いかなる視聴覚的、電子的、機械的、複写、録音等の手段によりもしくは形式により、伝搬又は複製をしてはなりません。

免責事項

当社は、GOWINSEMI Terms and Conditions of Sale (GOWINSEMI取引条件)に規定されている内容を除き、(明示的か又は黙示的かに拘わらず)いかなる保証もせず、また、知的財産権や材料の使用によりあなたのハードウェア、ソフトウェア、データ、又は財産が被った損害についても責任を負いません。当社は、事前の通知なく、いつでも本文書の内容を変更することができます。本文書を参照する何れの団体及び個人も、最新の文書やエラッタ(不具合情報)については、当社に問い合わせる必要があります。

バージョン履歴

日付	バージョン	説明
2017/04/20	1.0J	初版。
2017/09/19	1.1J	<ul style="list-style-type: none"> ● サポートされるデバイスを追加 : GW1NR-4、GW1N-6、GW1N-9、GW1NR-9。 ● ELVDS_IOBUF、TLVDS_IOBUF、BUFG、BUFS、OSC、IEM を追加。 ● DSP プリミティブを更新。 ● ODDR/ODDR_C、IDDR_MEM、IDES4_MEM、IDES8_MEM、RAM16S1、RAM16S2、RAM16S4、RAM16SDP1、RAM16SDP2、RAM16SDP4、ROM16 一部の port 名を更新。 ● OSC、PLL、DLLDLY の一部の Attribute を更新 ● 一部のプリミティブのインスタンス化を更新。 ● MIPI_IBUF_HS、MIPI_IBUF_LP、MIPI_OBUF、IDES16、OSER16 を追加。 ● CLKDIV の一部の Attribute を追加
2018/04/12	1.2J	VHDL でのプリミティブのインスタンス化を追加
2018/08/08	1.3J	<ul style="list-style-type: none"> ● サポートされるデバイスを追加 : GW1N-2B、GW1N-4B、GW1NR-4B、GW1N-6ES、GW1N-9ES、GW1NR-9ES、GW1NS-2、GW1NS-2C。 ● I3C_IOBUF、DHCEN を追加。 ● User Flash を追加。 ● EMPU を追加。 ● プリミティブ名称を更新
2018/10/26	1.4J	<ul style="list-style-type: none"> ● サポートされるデバイスを追加 : GW1NZ-1、GW1NSR-2C。 ● OSCZ、FLASH96KZ を追加。
2018/11/15	1.5J	<ul style="list-style-type: none"> ● サポートされるデバイスを追加 : GW1NSR-2; ● GW1N-6ES、GW1N-9ES、GW1NR-9ES を削除。
2019/01/26	1.6J	<ul style="list-style-type: none"> ● CLKDIV の 8 分周が GW1NS-2 をさらにサポート。 ● TLVDS_TBUF/OBUF をサポートするデバイスから GW1N-1 を削除。
2019/02/25	1.7J	TLVDS_IOBUF をサポートするデバイスから GW1N-1 を削除。
2019/05/20	1.8J	<ul style="list-style-type: none"> ● GW1N-1S のサポートを追加。 ● MIPI_IBUF を追加 ● OSCH を追加。 ● SPMI を追加。 ● I3C を追加 ● OSC をサポートするデバイスを更新。
2019/10/20	1.9J	IOB、BSRAM、CLOCK モジュールを更新。
2019/11/28	2.0J	<ul style="list-style-type: none"> ● GSR、INV などの Miscellaneous モジュールを追加。 ● サポートされるデバイスの情報を更新。 ● FLASH64KZ を追加、FLASH96KZ を削除。
2020/01/16	2.1J	<ul style="list-style-type: none"> ● IODELAYA、rPLL、PLLVR、CLKDIV2 を追加。

日付	バージョン	説明
		<ul style="list-style-type: none"> ● DPB/DPX9B、SDPB/SDPX9B、rSDP/rSDPX9、rROM/rROMX9、pROM/pROMX9 を追加。 ● EMCU、BANDGAP、FLASH64K を追加。 ● IODELAY、PLL、CLKDIV、OSC、DQCE を更新。 ● FF、LATCH の配置ルールを追加。 ● GW2A-55C のサポートを追加。 ● GW1N-6/GW1N-9/GW1NR-9 での DP/DPX9、DPB/DPX9B の使用を無効にする。 ● IOLOGIC で register の説明を追加。 ● GW1NZ-1 での DP/DPB の 1,2,4,8 ビット幅、DPX9/DPX9 の 9 ビット幅を無効にする。
2020/03/09	2.2J	<ul style="list-style-type: none"> ● GW1NS-2、GW1NS-2C、GW1NSR-2、GW1NSR-2C、GW1NSE-2C での DP/DPX9、DPB/DPX9B の使用を無効にする。 ● OSCF の OSCEN ポートの説明を追加。 ● PLL/rPLL/PLLVR のパラメータの説明を更新。
2020/06/08	2.3J	<ul style="list-style-type: none"> ● デバイス GW1N-2、GW1N-2B、および GW1N-6 を削除。 ● GW1N-9C、GW1NR-9C を追加。 ● IODELAYC、DHCENC、DCC を追加。 ● MIPI_IBUF の機能の説明を追加。 ● MIPI_IBUF_HS、MIPI_IBUF_LP、DLL を削除。 ● LUT5、MUX8 のポート図を追加。 ● VCC、GND を追加。 ● PLLVR、FLASH64K、BUFS、EMPU、CLKDIV2 プリミティブの紹介を更新。 ● DP/DPX9、ROM/ROMX9、SDP/SDPX9、rSDP/rSDPX9、rROM/rROMX9、PLL を削除。 ● Iologic のセクションの構造を調整し、ポート説明図のタイトルを統一。
2020/09/11	2.4J	ADC、BANDGAP、SPMI、および I3C IP の呼び出しの説明を追加。
2020/12/30	2.5J	チャプター「2 CFU」の説明を更新。
2021/07/22	2.6J	<ul style="list-style-type: none"> ● activeFlash を追加。 ● IP 呼び出しの図面を更新。 ● デバイス(GW1NZ-1C、GW1N-2、GW1N-2B、GW1N-1P5、GW1N-1P5B、GW1NR-2、GW1NR-2B)のサポートを追加。
2021/10/28	2.7J	activeFlash の説明を更新。
2022/07/22	2.8J	<ul style="list-style-type: none"> ● OTP モジュールを追加。 ● SAMB モジュールを追加。
2022/10/28	2.9J	MCU、USB20_PHY、および ADC を削除。
2023/04/20	3.0J	Arora V デバイスの内容を追加。
2023/07/14	3.1J	SPMI 構成オプションの"Shutdown by VCCEN"を削除。
2023/12/29	3.2J	Arora V の OTP プリミティブの説明を追加。
2024/10/25	3.3J	「8.9 OTP」を更新

目次

目次.....	i
図一覧.....	ii
表一覧.....	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 その他	20
8.1 GSR.....	20
8.2 INV.....	21
8.3 VCC.....	22
8.4 GND.....	23
8.5 BANDGAP	24
8.6 SPMI.....	27
8.7 I3C	32
8.8 activeFlash	40
8.9 OTP	42
8.10 SAMB	46
8.11 CMSER.....	49
8.12 CMSERA	52

図一覧

図 7-1 EMCU のポート図	8
図 8-1 GSR のポート図	20
図 8-2 INV のポート図	21
図 8-3 VCC のポート図	22
図 8-4 GND のポート図	23
図 8-5 BANDGAP のポート図	24
図 8-6 BandGap IP の構成ウィンドウ	25
図 8-7 SPMI のポート図	27
図 8-8 SPMI IP の構成ウィンドウ	30
図 8-9 I3C のポート図	32
図 8-10 I3C IP の構成ウィンドウ	39
図 8-11 activeFlash のポート図	40
図 8-12 GW2AN の OTP のポート図	42
図 8-13 Arora V の OTP のポート図	42
図 8-14 Arora V OTP プリミティブのインターフェースのタイミング図	44
図 8-15 GW2AN の SAMB のポート図	46
図 8-16 Arora V の SAMB のポート図	46
図 8-17 CMSER のポート図	49
図 8-18 CMSERA のポート図	53

表一覧

表 7-1 EMCU 対応デバイス	7
表 7-2 EMCU のポートの説明.....	8
表 8-1 GSR のポートの説明	20
表 8-2 INV のポートの説明	21
表 8-3 VCC のポートの説明.....	22
表 8-4 GND のポートの説明	23
表 8-5 BANDGAP 対応デバイス	24
表 8-6 BANDGAP のポートの説明	24
表 8-7 SPMI 対応デバイス	27
表 8-8 SPMI のポートの説明	27
表 8-9 I3C 対応デバイス	32
表 8-10 I3C のポートの説明.....	32
表 8-11 activeFlash 対応デバイス.....	40
表 8-12 activeFlash のポートの説明.....	40
表 8-13 OTP 対応デバイス	42
表 8-14 GW2AN の OTP のポートの説明	42
表 8-15 Arora V の OTP の説明.....	42
表 8-16 Arora V の OTP のパラメータの説明	43
表 8-17 Arora V の OTP User Efuse 領域の説明.....	43
表 8-18 SAMB 対応デバイス.....	46
表 8-19 GW2AN の SAMB のポートの説明	47
表 8-20 Arora V の SAMB のポートの説明	47
表 8-21 Arora V の SAMB のパラメータの説明	47
表 8-22 CMSER 対応デバイス.....	49
表 8-23 CMSER のポートの説明	49
表 8-24 CMSERA 対応デバイス	52
表 8-25 CMSERA のポートの説明.....	53

1 IOB

IOB には、入出力バッファ (IO Buffer)、入出力ロジック (IO Logic) が含まれます。Arora V デバイスの IO Buffer プリミティブおよび IO Logic プリミティブについては、『Arora V プログラマブル汎用 IO (GPIO) ユーザーガイド ([UG304](#))』、その他のデバイスの場合は、『Gowin プログラマブル汎用 IO (GPIO) ユーザーガイド ([UG289](#))』を参照してください。

2_{CFU}

コンフィギュラブル機能ユニット(CFU)とコンフィギュラブル論理ユニット(CLU)は、Gowin FPGA 製品のコアを構成する 2 つの基本構成要素です。各基本構成要素は、4 つのコンフィギュラブル論理セクション(CLS)と対応するコンフィギュラブル配線ユニット(CRU)で構成されます。CLU 内の CLS は、LUT、ALU、および ROM として構成することができ、SRAM として構成することはできません。CFU 内の CLS は、アプリケーションシナリオに応じて、LUT、ALU、SRAM、および ROM として構成することができます。Arora V デバイスの CFU プリミティブについては、『Arora V FPGA 製品コンフィギュラブル機能ユニット(CFU)ユーザーガイド ([UG303](#))』、その他のデバイスの場合は、『Gowin コンフィギュラブル機能ユニット(CFU)ユーザーガイド([UG288](#))』を参照してください。

3Memory

Gowin FPGA 製品には、ブロック SRAM(BSRAM)と分散 SRAM(SSRAM)を含む豊富なメモリリソースがあります。Arora V デバイスの BSRAM/SSRAM プリミティブについては、『Arora V BSRAM & SSRAM ユーザーガイド([UG300](#))』、その他のデバイスの場合は、『Gowin BSRAM & SSRAM ユーザーガイド([UG285](#))』を参照してください。

4DSP

Gowin FPGA 製品には、豊富な DSP リソースがあります。Arora V デバイスの DSP プリミティブについては、『Arora V DSP ユーザーガイド ([UG305](#))』、その他のデバイスの場合は、『Gowin DSP ユーザーガイド ([UG287](#))』を参照してください。

5 Clock

GOWIN セミコンダクターFPGA 製品は、直接にデバイスのあらゆるリソースに接続される専用のグローバルクロック(GCLK(PCLK および SCLK を含む))を提供しています。さらに、位相同期回路(PLL)、高速クロック(HCLK)、および DQS 等のクロックリソースも提供されています。Arora V デバイスの CLOCK プリミティブについては、『Arora V Clock ユーザーガイド([UG306](#))』、その他のデバイスの場合は、『Gowin Clock ユーザーガイド([UG286](#))』を参照してください。

6 User Flash

Gowin LittleBee ファミリーFPGA 製品は、User Flash を提供します。サポートされる User Flash の容量は、FPGA デバイスによって異なります。Flash プリミティブの詳細については、『Gowin User Flash ユーザーガイド ([UG295](#))』を参照してください。

7 EMPU

7.1 EMCU

プリミティブの紹介

EMCU(ARM Cortex-M3 Microcontroller Unit)は、ARM Cortex-M3 ベースのマイクロプロセッサです。32 ビット AHB/APB のバスモードを採用しています。内部では 2 つの UART、2 つの Timer、及び Watchdog が実装されています。また、外部には 16 ビットの GPIO、2 つの UART、JTAG、6 つの User Interrupt インターフェースが提供されています。また、AHB Flash 読み出しインターフェース、AHB Sram 書き込み/読み出しインターフェースも利用可能です。さらに、外部には 2 つの AHB バス拡張インターフェースと 1 つの APB バス拡張インターフェースが提供されています。EMCU は、割り込み処理機能を強化し、FLASH インターフェースを改善したと同時に、MCU の動作周波数も高めています。

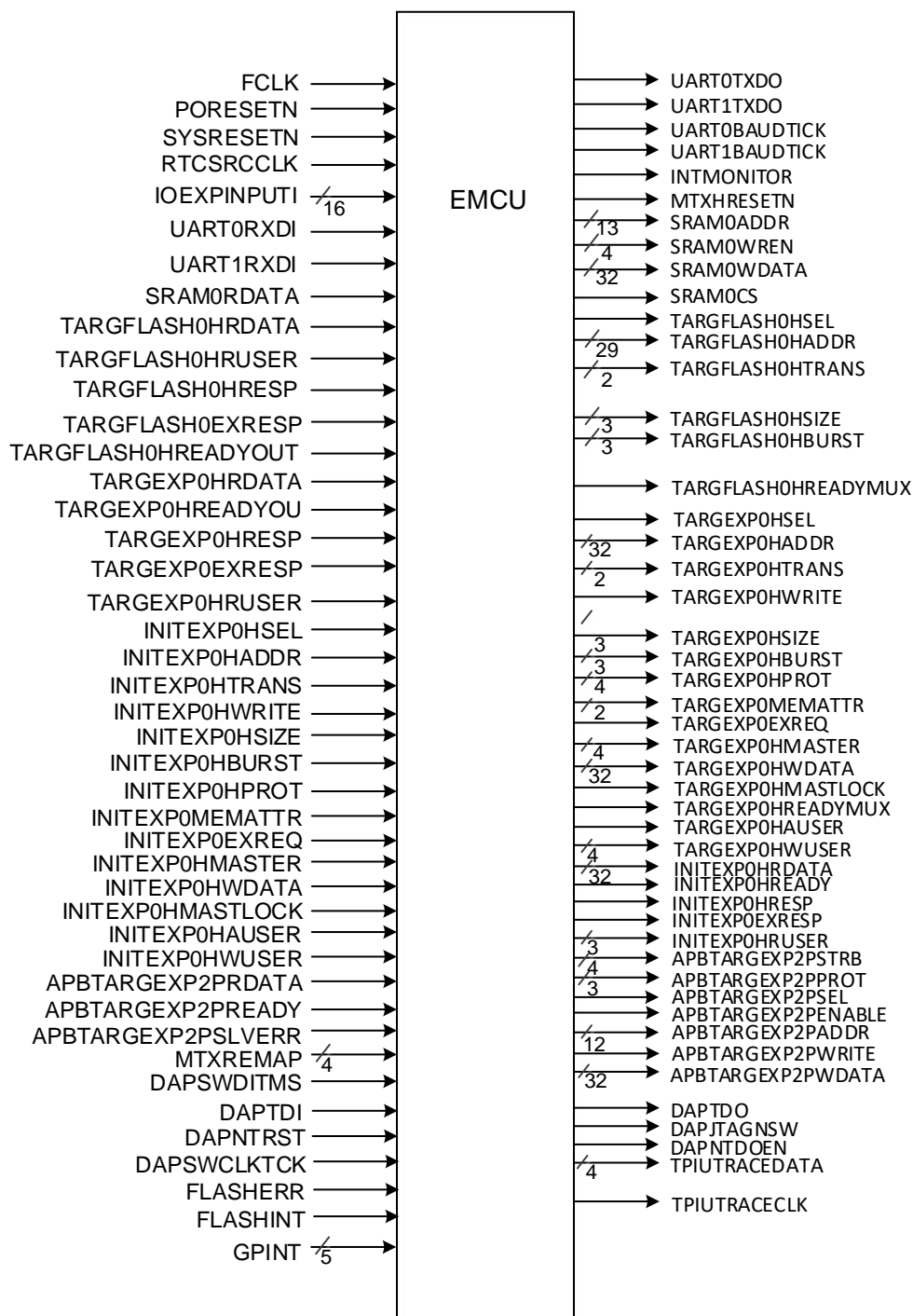
サポートされるデバイス

表 7-1 EMCU 対応デバイス

ファミリー	シリーズ	デバイス
LittleBee	GW1NS	GW1NS-4C
	GW1NSR	GW1NSR-4C
	GW1NSER	GW1NSER-4C

ポート図

図 7-1 EMCU のポート図



ポートの説明

表 7-2 EMCU のポートの説明

ポート	I/O	説明
FCLK	入力	Free running clock
PORESETN	入力	Power on reset

ポート	I/O	説明
SYSRESETN	入力	システムリセット
RTCSRCLK	入力	Used to generate RTC clock
IOEXPINPUTI[15:0]	入力	IOEXPINPUTI
UART0RXDI	入力	UART0RXDI
UART1RXDI	入力	UART1RXDI
SRAM0RDATA[31:0]	入力	SRAM Read data bus
TARGFLASH0HRDATA[31:0]	入力	TARGFLASH0, HRDATA
TARGFLASH0HRUSER[2:0]	入力	TARGFLASH0, HRUSER
TARGFLASH0HRESP	入力	TARGFLASH0, HRESP
TARGFLASH0EXRESP	入力	TARGFLASH0, EXRESP
TARGFLASH0HREADYOUT	入力	TARGFLASH0, EXRESP
TARGEXP0HRDATA[31:0]	入力	TARGEXP0, HRDATA
TARGEXP0HREADYOUT	入力	TARGEXP0, HREADY
TARGEXP0HRESP	入力	TARGEXP0, HRESP
TARGEXP0EXRESP	入力	TARGEXP0, EXRESP
TARGEXP0HRUSER[2:0]	入力	TARGEXP0, HRUSER
INITEXP0HSEL	入力	INITEXP0, HSELx
INITEXP0HADDR[31:0]	入力	INITEXP0, HADDR
INITEXP0HTRANS[1:0]	入力	INITEXP0, HTRANS
INITEXP0HWRITE	入力	INITEXP0, HWRITE
INITEXP0HSIZE[2:0]	入力	INITEXP0, HSIZE
INITEXP0HBURST[2:0]	入力	INITEXP0, HBURST
INITEXP0HPROT[3:0]	入力	INITEXP0, HPROT
INITEXP0MEMATTR[1:0]	入力	INITEXP0, MEMATTR
INITEXP0EXREQ	入力	INITEXP0, EXREQ
INITEXP0HMASTER[3:0]	入力	INITEXP0, HMASTER
INITEXP0HWDATA[31:0]	入力	INITEXP0, HWDATA
INITEXP0HMASTLOCK	入力	INITEXP0, HMASTLOCK
INITEXP0HAUSER	入力	INITEXP0, HAUSER
INITEXP0HWUSER[3:0]	入力	INITEXP0, HWUSER
APBTARGEXP2PRDATA[31:0]	入力	APBTARGEXP2, PRDATA
APBTARGEXP2PREADY	入力	APBTARGEXP2, PREADY
APBTARGEXP2PSLVERR	入力	APBTARGEXP2, PSLVERR
MTXREMAP[3:0]	入力	The MTXREMAP signals control the remapping of the boot memory range.
DAPSWDITMS	入力	Debug TMS
DAPTDI	入力	Debug TDI

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

ポート	I/O	説明
TARGEXP0HAUSER	出力	TARGEXP0, HAUSER
TARGEXP0HWUSER[3:0]	出力	TARGEXP0, HWUSER
INITEXP0HRDATA[31:0]	出力	INITEXP0, HRDATA
INITEXP0HREADY	出力	INITEXP0, HREADY
INITEXP0HRESP	出力	INITEXP0, HRESP
INITEXP0EXRESP	出力	INITEXP0, EXRESP
INITEXP0HRUSER[2:0]	出力	INITEXP0, HRUSER
APBTARGEXP2PSTRB[3:0]	出力	APBTARGEXP2, PSTRB
APBTARGEXP2PPROT[2:0]	出力	APBTARGEXP2, PPROT
APBTARGEXP2PSEL	出力	APBTARGEXP2, PSELx
APBTARGEXP2PENABLE	出力	APBTARGEXP2, PENABLE
APBTARGEXP2PADDR[11:0]	出力	APBTARGEXP2, PADDR
APBTARGEXP2PWRITE	出力	APBTARGEXP2, PWRITE
APBTARGEXP2PWDATA[31:0]	出力	APBTARGEXP2, PWDATA
DAPTDO	出力	Debug TDO
DAPJTAGNSW	出力	JTAG or Serial-Wire selection JTAG mode(1) or SW mode(0)
DAPNTDOEN	出力	TDO output pad control signal
TPIUTRACEDATA[3:0]	出力	Output data
TPIUTRACECLK	出力	Output clock, used by the TPA to sample the other pins

プリミティブのインスタンス化

Verilog でのインスタンス化 :

```

MCU u_sse050_top_syn (
.FCLK(fclk),
.PORESETN(poresetn),
.SYSRESETN(sysresetn),
.RTCSRCLK(rtsrcclk),
.IOEXPINPUTI(ioexpinputi[15:0]),
.IOEXPOUTPUTPUTO(ioexpoutputputo[15:0]),
.IOEXPOUTPUTPUTENO(ioexpoutputputeno[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(apbtargexp2pwwdata[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(tpiutracedclk),
.FLASHERR(flasherr),

```

        .GPINT(gpint),
        .FLASHINT(flashint)
    );

```

VHDL でのインスタンス化 :

```

COMPONENT MCU
    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;
    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=> rtcsrcclk;
        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=>daptidi,
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=>targflash0hssel,
TARGFLASH0HREADYMUX=>targflash0hreadymux,
SRAM0RDATA=>sram0rdata,
TARGFLASH0HADDR=>targflash0haddr,
TARGFLASH0HTRANS=>targflash0htrans,
TARGFLASH0HSIZE=>targflash0hsize,
TARGFLASH0HBURST=>targflash0hburst,
TARGFLASH0HRDATA=>targflash0hrdata,
TARGEXP0HADDR=>targexp0haddr,
TARGEXP0HSEL=>targexp0hssel,
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,
DAPTD0=>daptdo,
DAPJTAGNSW=>dapjtagnsu,
DAPNTDOEN=>dapntdoen,
TPIUTRACECLK=>tpiutracedclk);

8その他

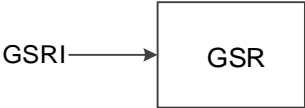
8.1 GSR

プリミティブの紹介

GSR(Global Set/Reset)は、グローバルセット/リセット機能を実装できるグローバルセット/リセットモジュールであり、アクティブ **Low** です。通常、**High** レベルに接続されていますが、動的に制御したい場合は、外部信号を接続してそれを **Low** にプルダウンすることで、レジスタなどのモジュールのセット/リセットを実現できます。

ポート図

図 8-1 GSR のポート図



ポートの説明

表 8-1 GSR のポートの説明

ポート名	I/O	説明
GSRI	入力	GSR 入力、アクティブ Low

プリミティブのインスタンス化

Verilog でのインスタンス化 :

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

VHDL でのインスタンス化 :

```
COMPONENT GSR  
    PORT (  

```

```

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

```

8.2 INV

プリミティブの紹介

INV (Inverter) は、インバーターです。

ポート図

図 8-2 INV のポート図



ポートの説明

表 8-2 INV のポートの説明

ポート名	I/O	説明
I	入力	INV データ入力
O	出力	INV データ出力

プリミティブのインスタンス化

Verilog でのインスタンス化 :

```

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

```

VHDL でのインスタンス化 :

```

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

```

```

uut:INV
    PORT MAP(
        O => O,
        I => I
    );

```

8.3 VCC

プリミティブの紹介

ロジック・ハイレベル・ジェネレーターです。

ポート図

図 8-3 VCC のポート図



ポートの説明

表 8-3 VCC のポートの説明

ポート名	I/O	説明
V	出力	VCC 出力

プリミティブのインスタンス化

Verilog でのインスタンス化 :

```

VCC uut (
    .V(V)
);

```

VHDL でのインスタンス化 :

```

COMPONENT VCC
    PORT (
        V:OUT std_logic
    );
END COMPONENT;

uut:VCC
    PORT MAP(
        V => V
    );

```

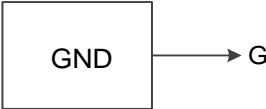
8.4 GND

プリミティブの紹介

ロジック・ローレベル・ジェネレーターです。

ポート図

図 8-4 GND のポート図



ポートの説明

表 8-4 GND のポートの説明

ポート名	I/O	説明
G	出力	GND 出力

プリミティブのインスタンス化

Verilog でのインスタンス化 :

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

VHDL でのインスタンス化 :

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

8.5 BANDGAP

プリミティブの紹介

BANDGAP はチップ内の一部のモジュールに一定の電圧と電流を供給します。**BANDGAP** をオフにすると、**OSC**、**PLL**、**FLASH** などのモジュールが動作しなくなるため、デバイスの消費電力が削減されます。

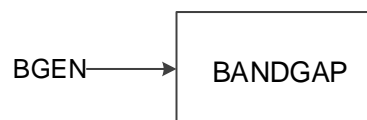
サポートされるデバイス

表 8-5 **BANDGAP** 対応デバイス

ファミリー	シリーズ	デバイス
LittleBee	GW1NZ	GW1NZ-1
	GW1N	GW1N-2, GW1N-1P5
	GW1NR	GW1NR-2

ポート図

図 8-5 **BANDGAP** のポート図



ポートの説明

表 8-6 **BANDGAP** のポートの説明

ポート名	I/O	説明
BGEN	入力	BANDGAP イネーブル信号、アクティブ High

プリミティブのインスタンス化

Verilog でのインスタンス化：

```

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

VHDL でのインスタンス化：

```

COMPONENT BANDGAP
  PORT (
      BGEN:IN std_logic
  );
END COMPONENT;
  
```

```

uut:BANDGAP
  PORT MAP(
    BGEN=> I
  );

```

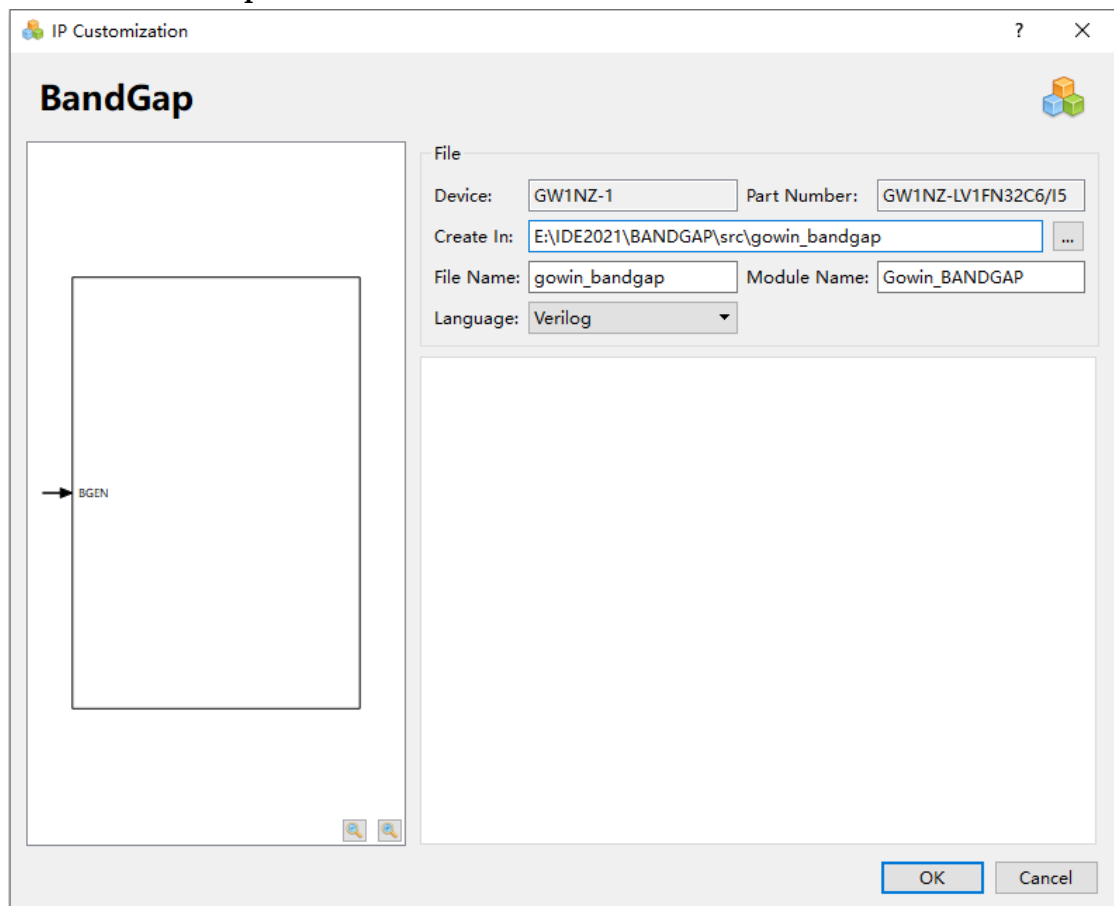
IP の呼び出し

IP Core Generator のインターフェースで **BandGap** をクリックすると、右側に **BandGap** の概要が表示されます。

IP の構成

IP Core Generator インターフェースで **BandGap** をダブルクリックすると、**BandGap** の"IP Customization"ウィンドウがポップアップします。このウィンドウには"File"構成タブ、"Options"構成タブ、およびポート図があります(図 8-6)。

図 8-6 BandGap IP の構成ウィンドウ



1. File 構成タブ

File 構成タブは、生成される IP ファイルの構成に使用されます。

- Device : 対象デバイス。
- Part Number : 部品番号。

- **Create In** : 生成される IP ファイルのパス。右側のテキストボックスでパスを直接編集するか、テキストボックスの右側にある選択ボタンを使用してパスを選択できます。
- **File Name** : 生成される IP ファイルのファイル名。右側のテキストボックスで再編集できます。
- **Module Name** : 生成される IP ファイルのモジュール名。右側のテキストボックスで編集できます。**Module Name** をプリミティブ名と同じにすることはできません。同じである場合、エラーが報告されます。
- **Language** : IP を実現するハードウェア記述言語。右側のドロップダウン・リストからターゲット言語(**Verilog** または **VHDL**)を選択します。

2. ポート図

ポート図に、IP Core の構成結果を示します(図 8-6)。

生成されるファイル

IP の構成が完了したら、"File Name"によって命名された 3 つのファイルが生成されます：

- "gowin_bandgap.v"は完全な verilog モジュールです。
- "gowin_bandgap_tmp.v"は IP のテンプレートファイルです。
- "gowin_bandgap.ipc"は IP の構成ファイルです。

注記：

VHDL が設計の言語として選択されている場合、生成される最初の 2 つのファイル名のサフィックスは.vhd になります。

8.6 SPMI

プリミティブの紹介

SPMI(System Power Management Interface)は、オンチップシステム内部の電源のオン/オフを動的に制御できる 2 線式シリアルインターフェースです。

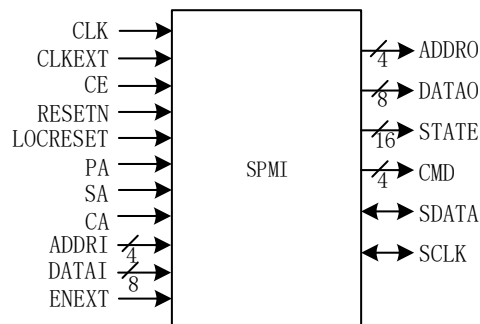
サポートされるデバイス

表 8-7 SPMI 対応デバイス

ファミリー	シリーズ	デバイス
LittleBee	GW1NZ	GW1NZ-1

ポート図

図 8-7 SPMI のポート図



ポートの説明

表 8-8 SPMI のポートの説明

ポート	I/O	説明
CLK	入力	Clock input
CLKEXT	入力	External clock input
CE	入力	Clock Enable
RESETN	入力	Reset input
ENEXT	入力	Enext input
LOCRESET	入力	Local reset input
PA	入力	Priority arbitration input
SA	入力	Secondary arbitration input
CA	入力	Connection arbitration input
ADDRi	入力	Addr input
DATAi	入力	入力データ
ADDR0	出力	Addr output
DATA0	出力	datat output

ポート	I/O	説明
STATE	出力	state output
CMD	出力	command output
SDATA	入出力	SPMI Serial data
SCLK	入出力	SPMI Serial Clock

プリミティブのインスタンス化

Verilog でのインスタンス化 :

```
SPMI uut (
    .ADDRO(addr0),
    .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(addr1),
    .DATAI(datai),
    .CLKEXT(clkext),
    .ENEXT(enext)
);
```

VHDL でのインスタンス化 :

```
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;
        ADDR:IN std_logic_vector(3 downto 0);
        DATA: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,
        ADDR=>addri,
        DATA=>datai,
        ADDRO=>addro,
        DATAO=>datao,
        STATE=>state,
        CMD=>cmd,
        SDATA=>sdata,
        SCLK=>sclk
    );

```

IP の呼び出し

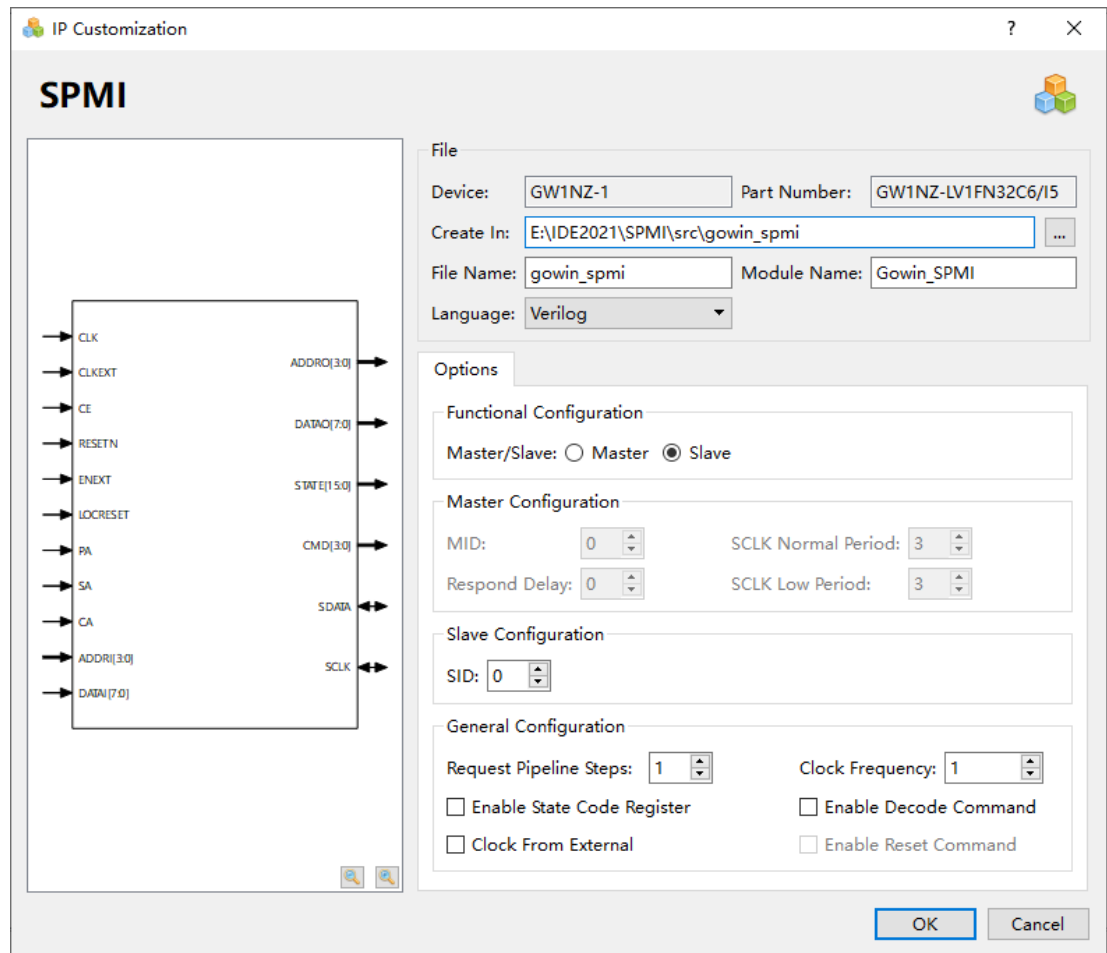
IP Core Generator のインターフェースで"SPMI"をクリックすると、右側に SPMI の概要が表示されます。

IP の構成

IP Core Generator インターフェースで SPMI をダブルクリックすると、

SPMI の"IP Customization"ウィンドウがポップアップします。このウィンドウには"File"構成タブ、"Options"構成タブ、およびポート図があります(図 8-8)。

図 8-8 SPMI IP の構成ウィンドウ



1. File 構成タブ

- File 構成タブは、生成される IP ファイルの構成に使用されます。
- SPMI の File 構成タブの使用は BANDGAP モジュールと同様であるので、[8.5 BANDGAP](#) の File 構成タブを参照してください。

2. Options 構成タブ

- Options 構成タブは、IP のカスタマイズに使用されます(図 8-8)。
- Functional Configuration :
 - Master/Slave : SPMI をマスターまたはスレーブとして設定します。
- Master Configuration :
 - MID : マスターの ID です。範囲は 0~3 で、デフォルト値は 0 です。

- Respond Delay : 応答の遅延時間を設定します。
- SCLK Normal Period : Normal モードでの SCLK の周期です。
- SCLK Low Period : スリープモードでの SCLK の周期です。
- Slave Configuration :
SID : SPMI スレーブの ID を設定します。
- General configuration :
 - Enable State Code Register : レジスタを有効または無効にします。例えば、「状態コードレジスタを有効にする(Enable State Code Register)」がチェックされている場合、出力 STATE データは 1 つのレジスタを通過します。
 - Request Pipeline Steps : リクエスト信号のサンプリング時間の遅延ステップを設定します。
 - Enable Decode Command : デコードを有効または無効にします。「デコードコマンドを有効にする(Enable Decode Command)」がチェックされている場合、SPMI はリセット、スリープ、シャットダウン、およびウェイクアップコマンドをデコードします。
 - Enable Reset Command : リセットコマンドを有効または無効にします。
 - Clock From External : 外部クロックを有効または無効にします。
 - Clock Frequency : システムクロック周波数。

3. ポート図

ポート図に、IP Core の構成結果を示します(図 8-8)。

生成されるファイル

IP の構成が完了したら、「File Name」によって命名された 3 つのファイルが生成されます :

- "gowin_spmi.v"は完全な verilog モジュールです。
- "gowin_spmi_tmp.v"は IP のテンプレートファイルです。
- "gowin_spmi.ipc"は IP の構成ファイルです。

注記 :

VHDL が設計の言語として選択されている場合、生成される最初の 2 つのファイル名のサフィックスは.vhd になります。

8.7 I3C

プリミティブの紹介

I²C と SPI の重要な特性を有する I3C (Improved Inter Integrated Circuit) は、IC チップシステムの物理ポート数の減少を可能にし、低消費電力、高いデータレート、およびその他既存のポートプロトコルとの互換性などの特性を備える 2 線式バスです。

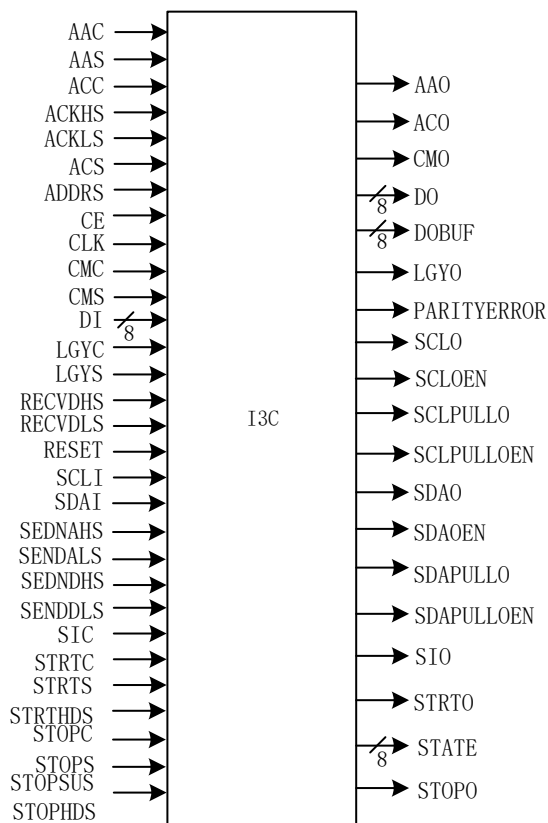
サポートされるデバイス

表 8-9 I3C 対応デバイス

ファミリー	シリーズ	デバイス
LittleBee	GW1NZ	GW1NZ-1

ポート図

図 8-9 I3C のポート図



ポートの説明

表 8-10 I3C のポートの説明

ポート	I/O	説明
CE	入力	Clock Enable
RESET	入力	Reset input

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

ポート	I/O	説明
STRTO	出力	Output the START command
PARITYERROR	出力	Output check when receiving data
DOBUF	出力	Data output after caching
DO	出力	Data output directly
STATE	出力	Output the internal state
SDAO	出力	I3C serial data output
SCLO	出力	I3C serial clock output
SDAOEN	出力	I3C serial data oen output
SCLOEN	出力	I3C serial clock oen output
SDAPULLO	出力	Controllable pull-up of the I3C serial data
SCLPULLO	出力	Controllable pull-up of the I3C serial clock
SDAPULLOEN	出力	Controllable pull-up of the I3C serial data oen
SCLPULLOEN	出力	Controllable pull-up of the I3C serial clock oen

プリミティブのインスタンス化

Verilog でのインスタンス化 :

```

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 でのインスタンス化 :
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;  
SENDDHS: IN STD_LOGIC;  
SENDDL: 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 => recvdl,  
ADDRS => addrs,  
DI => di,  
SDAI => sdai,  
SCLI => scli,  
CE => ce,  
RESET => reset,  
CLK => clk
```

```
);
```

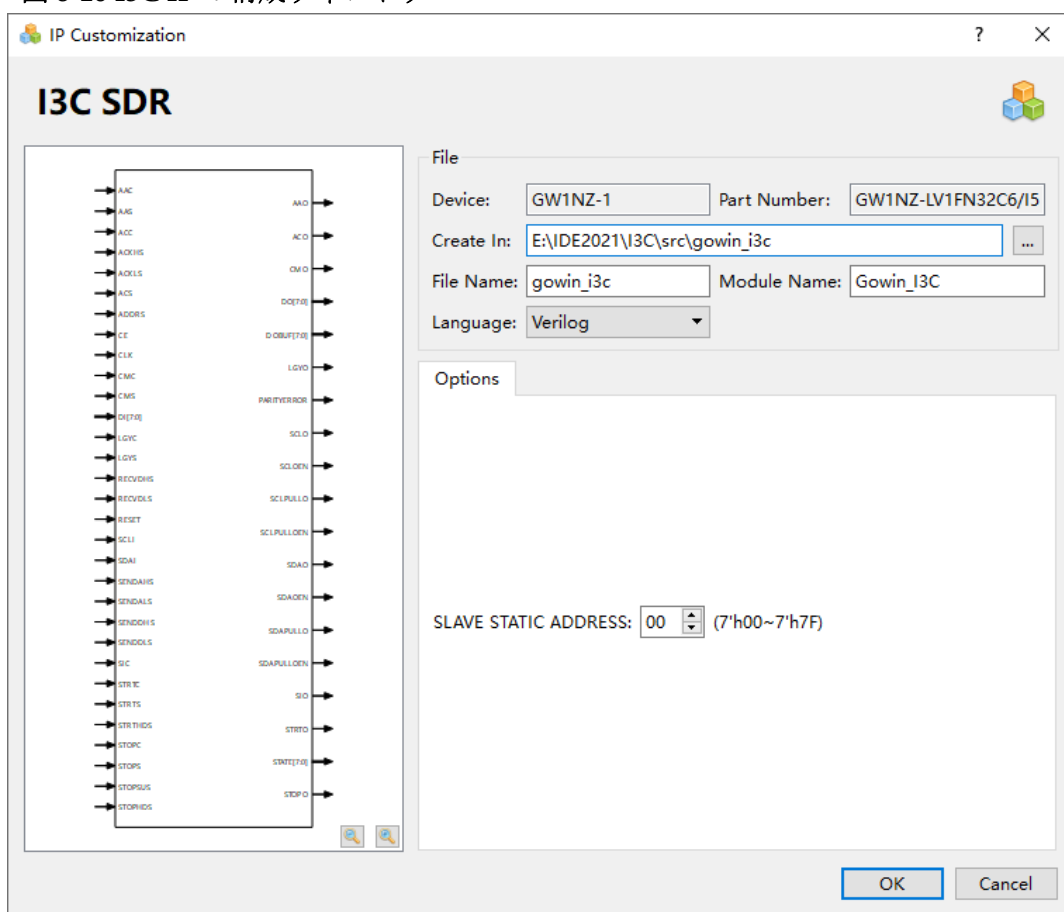
IP の呼び出し

IP Core Generator インターフェースで[I3C]> [I3C SDR]をクリックすると、右側に I3C SDR の情報の概要が表示されます。

IP の構成

IP Core Generator インターフェースで"I3C SDR"をダブルクリックすると、"IP Customization"ウィンドウがポップアップします。このウィンドウには Options 構成タブ、File 構成タブ、およびポート図があります(図 8-10)。

図 8-10 I3C IP の構成ウィンドウ



1. File 構成タブ

- File 構成タブは、生成される IP ファイルの構成に使用されます。
- I3C の File 構成タブの使用は BANDGAP モジュール同様であるので、[8.5 BANDGAP](#) の File 構成タブを参照してください。

2. Options 構成タブ

- Options 構成タブは、IP のカスタマイズに使用されます(図 8-10)。
- SLAVE STATIC ADDRESS - スレーブの静的アドレスを指定します。

3. ポート図

ポート図は、IP Core の構成結果を表示します(図 8-10)。

生成されるファイル

IP の構成が完了したら、"File Name"によって命名された 3 つのファイルが生成されます：

- "gowin_i3c.v"は完全な verilog モジュールです。
- "gw_i3c_tmp.v"は IP のテンプレートファイルです。

- "gowin_i3c.ipc"は IP の構成ファイルです。

注記：

VHDL が設計の言語として選択されている場合、生成される最初の 2 つのファイル名のサフィックスは.vhd になります。

8.8 activeFlash

プリミティブの紹介

組み込み SPI Nor Flash をアクティブにします。**activeFlash** をインスタンス化するとき、**I_active_flash_sclk** はクロック信号である必要があり、**I_active_flash_holdn** を High にプルアップする必要があります。

注記：

activeFlash をインスタンス化する場合、14 個の LUT と 10 個の REG リソースが追加で使用されます。

サポートされるデバイス

表 8-11 activeFlash 対応デバイス

ファミリー	シリーズ	デバイス
Arora	GW2AN	GW2AN-18X, GW2AN-9X

ポート図

図 8-11 activeFlash のポート図



ポートの説明

表 8-12 activeFlash のポートの説明

ポート	I/O	説明
I_active_flash_holdn	入力	High レベルの場合、Flash をアクティブにします。
I_active_flash_sclk	入力	クロック入力信号
O_active_flash_ready	出力	High レベルは Flash がアクティブ化されたことを示します。

プリミティブのインスタンス化

Verilog でのインスタンス化：

```
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 でのインスタンス化 :

```

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
    );

```

呼び出し条件

次の条件のいずれかが満たされた場合、**activeFlash** モジュールをユーザーデザインでインスタンス化する必要があります。

1. Gowin ソフトウェアの **Configuration > Bitstream** 構成ページの **background programming** が **I2C/JTAG/SSPI/QSSPI** に構成されている場合、**activeFlash** をインスタンス化する必要があります。
2. Gowin ソフトウェアの **Configuration > Bitstream** 構成ページの **background programming** が **OFF** に構成され、**SPI Nor Flash Interface IP** が使用される場合、**activeFlash** をインスタンス化する必要があります。

8.9 OTP

プリミティブの紹介

OTP (One Time Programming)を使用してチップの製品情報を読み出すことができます。

サポートされるデバイス

表 8-13 OTP 対応デバイス

ファミリー	シリーズ	デバイス
Arora	GW2AN	GW2AN-18X, GW2AN-9X
	GW5AT	GW5AT-138
	GW5A	GW5A-25, B バージョンの GW5A-138
	GW5AST	B バージョンの GW5AST-138

ポート図

図 8-12 GW2AN の OTP のポート図

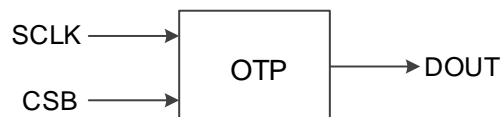
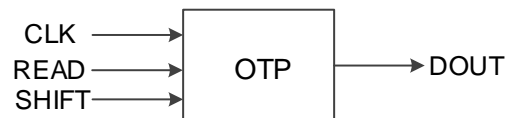


図 8-13 Arora V の OTP のポート図



ポートの説明

表 8-14 GW2AN の OTP のポートの説明

ポート	I/O	説明
SCLK	入力	シリアル入力クロック。立ち下がりエッジでデータが DOUT にシフトアウトされます。データを読み出すときは、2.5MHz のクロックを使用することをお勧めします。
CSB	入力	チップセレクト信号、アクティブ Low
DOUT	出力	シリアルデータ出力

表 8-15 Arora V の OTP の説明

ポート	I/O	説明
CLK	入力	クロック信号
READ	入力	対応する 128 ビットの efuse レジスタのデータをシフトレジスタにロードします。パルス信号。アクティブ High
SHIFT	入力	シフトレジスタのデータを出力インターフェースにシフトアウトします。レベル信号、アクティブ High

ポート	I/O	説明
DOUT	出力	チップの DNA 情報の読み出し/ユーザー情報の出力

パラメータの説明

表 8-16 Arora V の OTP のパラメータの説明

パラメータ名	値の範囲	デフォルト値	説明
MODE	2'b00, 2'b01, 2'b10	2'b01	OTP モードの選択 <ul style="list-style-type: none"> ● 2'b01 : チップの DNA を読み出します。 ● 2'b00 : ユーザー情報を読み出します。 ● 2'b10 : ユーザー制御情報を読み出します。

Arora V の OTP プリミティブの紹介

Arora V の OTP プリミティブを使用することにより、チップの DNA 情報、ユーザー情報、ユーザー制御情報を User efuse 領域から読み出すことができます。128 ビットの User efuse 領域の説明を表 8-17 に示します。

表 8-17 Arora V の OTP User Efuse 領域の説明

領域内の位置	タイプ	説明	
127~112bit (16bits)	user_misc	reserved	
111~96bit (16bits)	ユーザー制御情報	reserved	-
		107bit prgm_user_control_lock	1: JTAG でユーザー制御領域をプログラムすることはできません
		106bit rd_user_misc_lock	1: user_misc 領域を読み出すことはできません
		105bit prgm_user_misc_lock	1: user_misc 領域をロックします
		104bit lock_sel_key_r	0: 復号化モジュールが key を選択します。 1: 復号化モジュールが key2 を選択します。
		103bit prgm_rd_dna_lock	1: JTAG で 64 ビットの DNA 領域をプログラムおよびリードバックすることはできません
		102bit rd_fuse_user_lock	1: JTAG で 32 ビットのユーザー情報領域をリードバックすることはできません
		101bit prgm_fuse_user_lock	1: JTAG で 32 ビットのユーザー情報領域をプログラムすることはできません
		100bit rd_key2_lock	1: JTAG で key2 をリードバックすることはできません

領域内の位置	タイプ	説明	
		99bit prgm_key2_lock	1: JTAG で key2 を再度プログラムすることはできません
		98bit rd_key_lock	1: JTAG で key をリードバックすることはできません
		97bit prgm_key_lock	1: JTAG で key を再度プログラムすることはできません
		96bit cfg_aes_only	1: 暗号化ビットストリームのみを使用できます
95~32bit (64bits)	DNA 情報	64 bits DNA Info	DNA 情報を保存します
31~0bit (32bits)	ユーザー情報	32 bits User defined	-

インターフェースのタイミング

図 8-14 Arora V OTP プリミティブのインターフェースのタイミング図

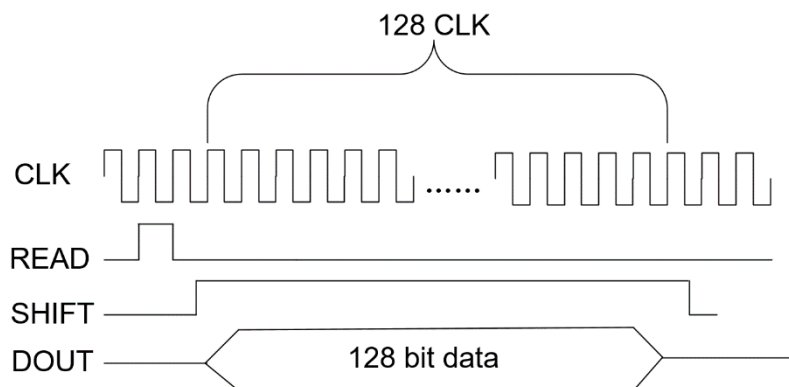


図 8-14 に示すように、まず **READ**(1 クロックサイクル以上)を使用してデータをロードします。次に、**READ** を **Low** にプルダウンし、**SHIFT** を **High** にプルアップして、128 ビットのデータ (LSB ファースト) を **DOUT** から 1 つずつシフトアウトします(クロック サイクルごとに 1 ビットのデータを出力します)。

出力されるデータの有効ビット数は、**MODE** 値によって異なります：

- 00 : 出力される 128 ビットのうち下位 32 ビットが有効です。
- 01 : 出力される 128 ビットのうちビット 95~ビット 32(64 ビット)が有効です。
- 10 : 出力される 128 ビットのうちビット 111~ビット 96(16 ビット)が有効です。

プリミティブのインスタンス化

GW2AN の OTP のインスタンス化

Verilog でのインスタンス化：

```

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

```

VHDL でのインスタンス化 :

```

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 の **OTP** のインスタンス化

Verilog でのインスタンス化 :

```

OTP uut (
    . READ(READ),
    .SHIFT(SHIFT),
    .CLK(CLK),
    . DOUT (DOUT)
);

defparam uut.MODE=2'b01;

```

VHDL でのインスタンス化 :

```

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

プリミティブの紹介

SAMB(SPI Address for Multi Boot)は、Multi Boot モード時のアドレス選択に使用され、静的アドレスと動的アドレスを選択することができます。

サポートされるデバイス

表 8-18 SAMB 対応デバイス

ファミリー	シリーズ	デバイス
Arora	GW2AN	GW2AN-18X, GW2AN-9X
	GW5AT	GW5AT-138
	GW5A	GW5A-25, B バージョンの GW5A-138
	GW5AST	B バージョンの GW5AST-138

ポート図

図 8-15 GW2AN の SAMB のポート図

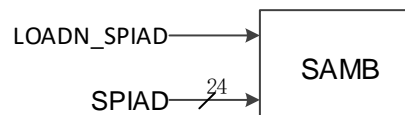
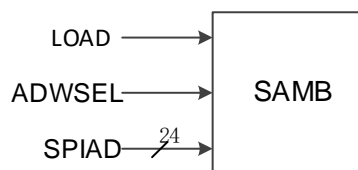


図 8-16 Arora V の SAMB のポート図



ポートの説明

表 8-19 GW2AN の SAMB のポートの説明

ポート	I/O	説明
LOADN_SPIAD	入力	静的 SPI アドレスまたは動的 SPI アドレス信号で指定された場所を選択するために使用されます。High の場合は静的 SPI アドレスを選択し、Low の場合は動的 SPI アドレス信号 SPIAD を選択します
SPIAD[23:0]	入力	SPI アドレス信号

表 8-20 Arora V の SAMB のポートの説明

ポート	I/O	説明
LOAD	入力	静的 SPI アドレスまたは動的 SPI アドレス信号で指定された場所を選択するために使用されます。Low の場合は静的 SPI アドレスを選択し、High の場合は動的 SPI アドレス信号 SPIAD を選択します
SPIAD[23:0]	入力	SPI アドレス信号
ADWSEL	入力	アドレスのビット幅選択信号 0 : アドレスは 24 ビット 1: アドレスは 32 ビット(下位 8 ビットはゼロで埋められます)

パラメータの説明

表 8-21 Arora V の SAMB のパラメータの説明

パラメータ名	値の範囲	デフォルト値	説明
MODE	2'b00, 2'b01, 2'b10, 2'b11	2'b00	SAMB モードの選択 <ul style="list-style-type: none"> ● 2'b00: Normal mode ● 2'b01: Fast mode ● 2'b10: Dual mode ● 2'b11: Quad mode

プリミティブのインスタンス化

GW2AN の SAMB のインスタンス化

Verilog でのインスタンス化 :

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

VHDL でのインスタンス化 :

```
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 の SAMB のインスタンス化

Verilog でのインスタンス化 :

```

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

```

VHDL でのインスタンス化 :

```

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

プリミティブの紹介

CMSER (Configuration Memory Soft Error Recovery) は、コンフィギュレーションメモリを継続的に監視してソフトエラーを検出し、その能力の範囲内でソフトエラーの修正を試みます。これは、ユーザーデザインのバックグラウンドでコンフィギュレーションメモリをフレームごとに読み取り、ECC デコードと CRC で実現されます。修正されたフレームデータを SRAM にプログラミングし直すことによって、限られた数のエラービットは修正されます。

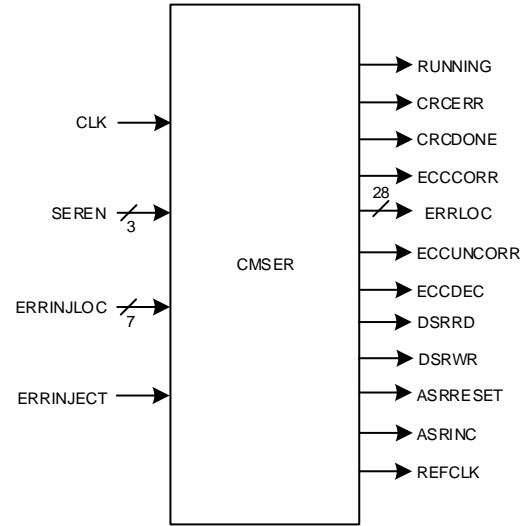
サポートされるデバイス

表 8-22 CMSER 対応デバイス

ファミリー	シリーズ	デバイス
Arora	GW5AT	GW5AT-138
	GW5A	B バージョンの GW5A-138
	GW5AST	B バージョンの GW5AST-138

ポート図

図 8-17 CMSER のポート図



ポートの説明

表 8-23 CMSER のポートの説明

ポート	I/O	説明
CLK	入力	Clock input

ポート	I/O	説明
SEREN	入力	(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	入力	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	入力	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	出力	The high level of this signal indicates CMSER is running
CRCERR	出力	one cycle high pulse to indicate the CRC error event
CRCDONE	出力	one cycle high pulse to indicate the completion of CRC calculation and comparison
ECCCORR	出力	one cycle high pulse to indicate the correctable ECC error event
ECCUNCORR	出力	one cycle high pulse to indicate the uncorrectable ECC error event
ERRLOC	出力	the location of ECC error
ECCDEC	出力	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	出力	one cycle high pulse to indicate the reading operation of DSR
DSRWR	出力	one cycle high pulse to indicate the writing operation of DSR
ASRRESET	出力	one cycle high pulse to indicate the reset of ASR
ASRINC	出力	one cycle high pulse to indicate the increase of ASR address
REFCLK	出力	the output reference clock for the generation of user CMSER interface design

プリミティブのインスタンス化

Verilog でのインスタンス化 :

```

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 でのインスタンス化 :
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

プリミティブの紹介

CMSERA (Configuration Memory Soft Error Recovery) は、コンフィギュレーションメモリを継続的に監視してソフトエラーを検出し、その能力の範囲内でソフトエラーの修正を試みます。これは、ユーザーデザインのバックグラウンドでコンフィギュレーションメモリをフレームごとに読み取り、ECC デコードと CRC で実現されます。修正されたフレームデータを SRAM にプログラミングし直すことによって、限られた数のエラービットは修正されます。

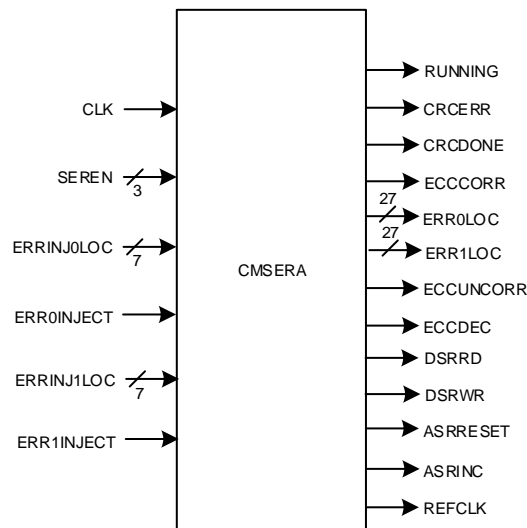
サポートされるデバイス

表 8-24 CMSERA 対応デバイス

ファミリー	シリーズ	デバイス
Arora	GW5A	GW5A-25

ポート図

図 8-18 CMSERA のポート図



ポートの説明

表 8-25 CMSERA のポートの説明

ポート	I/O	説明
CLK	入力	Clock input
SEREN	入力	(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	入力	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	入力	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	入力	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	入力	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]

ポート	I/O	説明
		1_XXXX: 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	出力	The high level of this signal indicates CMSER is running
CRCERR	出力	one cycle high pulse to indicate the CRC error event
CRCDONE	出力	one cycle high pulse to indicate the completion of CRC calculation and comparison
ECCCORR	出力	one cycle high pulse to indicate the correctable ECC error event
ECCUNCORR	出力	one cycle high pulse to indicate the uncorrectable ECC error event
ERR0LOC	出力	the location of ECC error 0
ERR1LOC	出力	the location of ECC error 1
ECCDEC	出力	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	出力	one cycle high pulse to indicate the reading operation of DSR
DSRWR	出力	one cycle high pulse to indicate the writing operation of DSR
ASRRESET	出力	one cycle high pulse to indicate the reset of ASR
ASRINC	出力	one cycle high pulse to indicate the increase of ASR address
REFCLK	出力	the output reference clock for the generation of user CMSER interface design

プリミティブのインスタンス化

Verilog でのインスタンス化 :

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 でのインスタンス化 :

```
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  
);
```

