AN0119
应用笔记

AT32F413/415 GPIO使用指南

前言

AT32F413/415 的通用功能 I/O (GPIO)提供了一系列与外部环境通讯的接口，可用于 MCU 与其他嵌入式设备之间通过数字或模拟方式的通讯。GPIO 的复用功能和重映射功能，能够使得多个外设可以同时工作，在特定的封装下获得最大的灵活性。

这篇应用笔记介绍 AT32F413/415 的 GPIO 功能及固件驱动程序 API 的配置和使用，并对 BSP 例程的软件设计加以说明，同时演示使用方法并展示实验效果，供用户参考。

注：本应用笔记对应的代码是基于雅特力提供的V2.x.x板级支持包（BSP）而开发，对于其他版本BSP，需要注意使用上的区别。

参考资料：

- AT32F413_Firmware_Library_V2.x.x\project\at_start_f413\examples\gpio
- AT32F415_Firmware_Library_V2.x.x\project\at_start_f415\examples\gpio
- RM_AT32F413 系列技术手册的通用功能和复用功能输入输出（GPIO 和 IOMUX）章节
- RM_AT32F415 系列技术手册的通用功能和复用功能输入输出（GPIO 和 IOMUX）章节
- Datasheet_AT32F413 系列数据手册
- Datasheet_AT32F415 系列数据手册

支持型号列表：

支持型号	AT32F413 系列
	AT32F415 系列

目录

表目录

图目录

1 GPIO 概述

AT32F413/415 支持多达 55 个双向 I/O 引脚，这些引脚分为 5 组，分别为 PA、PB、PC、PD 和 PF， 每组最多包含 16 个引脚，每个引脚都可以实现与外部的通讯、控制以及数据采集的功能。每个引脚都支持通用功能输入输出（GPIO）或复用功能输入输出（IOMUX）。

GPIO 主要特性：

- 最大封装（64pin）具有 55 个多功能双向的 I/O 口；
- 所有 GPIO 都可以配置为外部中断输入；
- 大部分 GPIO 引脚 5V 容忍；
- 所有 GPIO 都支持配置锁定功能，以避免意外的操作 GPIO；
- 所有 GPIO 引脚都可以软件配置成四种输入模式（输入浮空、输入上拉、输入下拉、模拟输入）和四种输出模式（开漏输出、推挽式输出、推挽式复用、开漏复用），见表 1；
- 可软件配置的 GPIO 引脚电流推动/吸入能力；

2 GPIO 功能描述

- 系统上电和刚复位后，复用功能未开启，除了 JATG 相关的引脚，都被配置为浮空输入模式。JTAG 相关引脚则配置为：PA15/JTDI、PA13/JTMS-SWDIO 和 PB4/NJTRST 为输入上拉模式，PA14/JTCK- SWCLK 为输入下拉模式，PB3/JTDO 为浮空输入模式。
- 当配置为输出模式时，写到输出数据寄存器（GPIOx_ODT）上的值会输出到相应的 GPIO 引脚。可以以推挽模式或开漏模式（仅低电平被驱动，高电平表现为高阻）使用输出驱动器。
- 当配置为输入模式时，输入数据寄存器（GPIOx_IDT）在每个 AHB 时钟周期捕捉 GPIO 引脚上的数据。
- 端口位设置/清除寄存器（GPIOx_SCR）和端口位清除寄存器（GPIOx_CLR）为端口位输出数据寄存器（GPIOx_ODT）提供位访问能力。
- 为了防止误操作导致 GPIO 功能混乱，提供每个对应引脚的锁定机制（通过 GPIO 写保护寄存器 GPIOx_WPR）。一旦锁定，在下次复位或者上电之前都不能进行对应引脚的 GPIO 配置。

图 1. GPIO 基本结构

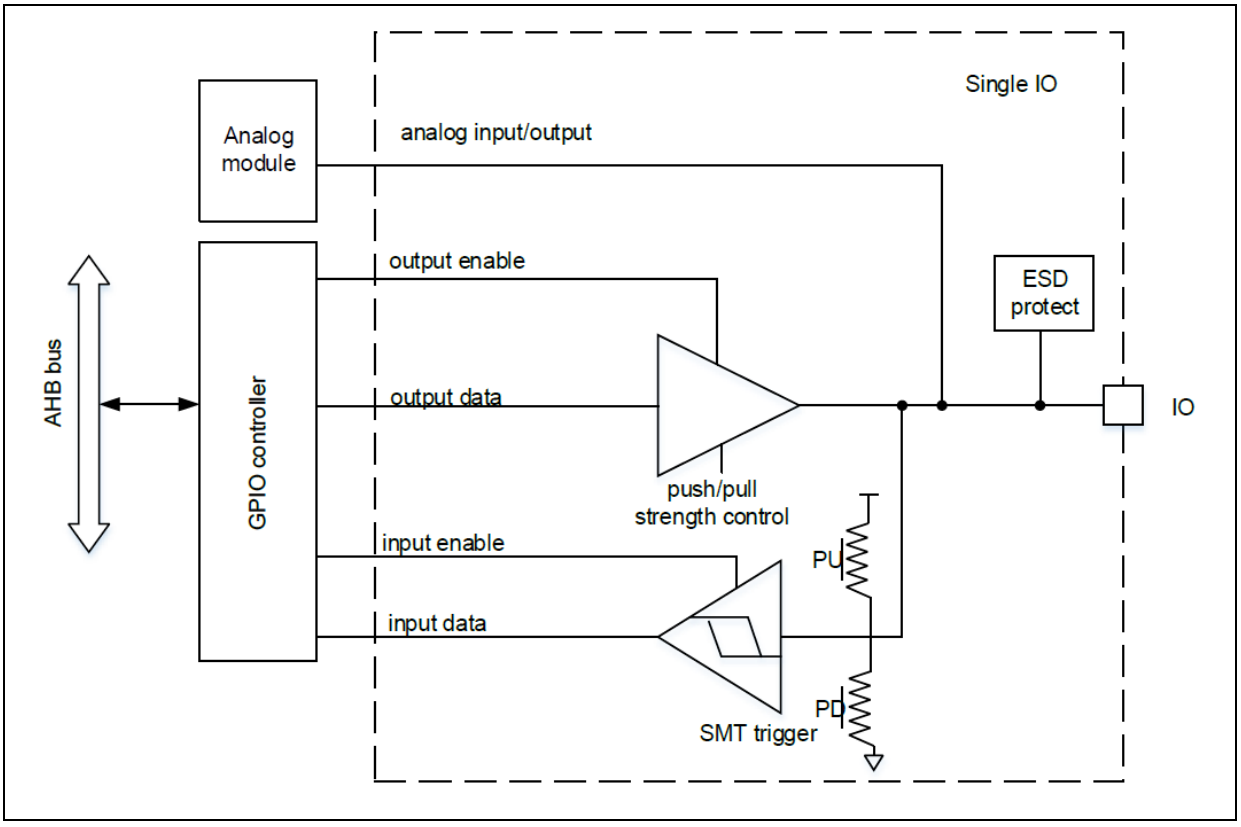


表 1. GPIO 端口配置表

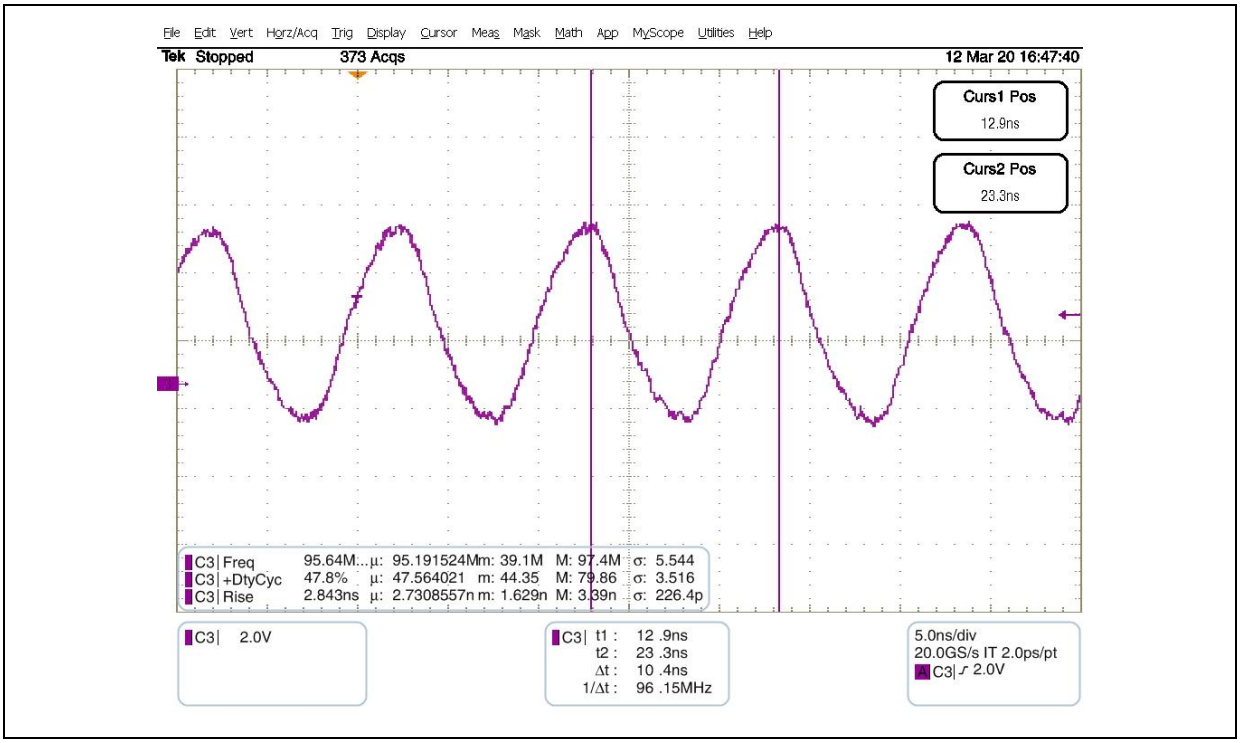
配置模式		GPIOx_CFGLR/ GPIOx_CFGHR 寄存器				GPIOx_ODT 寄存器
		IOFC1	IOFC0	IOMC1	IOMC0	ODT
通用输出	推挽(Push-Pull)	0(通用)	0(推挽)	01(较大电流推动/吸入能力输出) 1x(适中电流推动/吸入能力输出)		0(低电平)/1(高电平)
	开漏(Open-Drain)		1(开漏)			0(低电平)/1(高电平)
复用输出	推挽(Push-Pull)	1(复用)	0(推挽)			不使用
	开漏(Open-Drain)		1(开漏)			不使用

配置模式		GPIOx_CFGLR/ GPIOx_CFGHR 寄存器				GPIOx_ODT 寄存器
		IOFC1	IOFC0	IOMC1	IOMC0	ODT
输入	模拟输入	00(模拟)		00(输入)		不使用
	浮空输入	01(浮空)				不使用
	下拉输入	10（下拉或上拉）				0(下拉)
	上拉输入					1(上拉)

2.1 GPIO Toggle

AT32F413/415 提供快速 I/O，寄存器存取速度最高为 f_{AHB}，可以看到在 AT32F413 主频为 192MHz 时，GPIO 翻转频率能够轻松达到 96MHz：

图 2. GPIO 翻转速度



2.2 GPIO 引脚的 5V or 3.3V 容忍

2.2.1 5V 容忍引脚（FT）

数据手册引脚定义表中，IO电平栏标注“FT”（FiveVoltage Tolerance）为5V容忍引脚。

表 2. FT 引脚示例

引脚名称	种类	IO电平	主功能	复用功能	
				默认功能	重映射
PA8	I/O	FT	PA8	CLKOUT / USART1_CK / SPIM_CS / USBFS1_SOF / TMR1_CH1	I2C2_SCL
PB8	I/O	FT	PB8	TMR10_CH1 / SDIO1_D4 / TMR4_CH3	I2C1_SCL / CAN1_RX

2.2.2 标准 3.3V 容忍引脚

数据手册引脚定义表中，IO电平栏标注“ - ”为标准3.3V容忍引脚。

表 3. 标准 3.3V 容忍引脚示例

引脚名称	种类	IO电平	主功能	复用功能	
				默认功能	重映射
PA11	I/O	-	PA11	USBFS1_D- / SPIM_IO0/ USART1_CTS / CAN1_RX / TMR1_CH4	-

3 IOMUX

3.1 GPIO 复用功能输入/输出

- 对某个具体的GPIO引脚，可能有多个外设复用功能，但在任意时刻应当只配置一个外设与之相连。（比如表2的PA8，可作为CLKOUT / USART1_CK / SPIM_CS / USBFS1_SOF / TMR1_CH1 / I2C2_SCL）。
- 当多个外设复用映射到同一个GPIO引脚时，外设遵循以下优先级规则：
  - 硬件抢占引脚（见3.2.1）优先
  - JTAG调试端口（见3.2.2）优先
  - 非 TMR 外设复用映射优先于 TMR 外设
  - 多个非 TMR 外设之间复用映射无优先级关系，复用功能叠加到同一个引脚
- 选择特定外设映射至哪个GPIO引脚是由IO复用重映射寄存器x（IOMUX_REMAPx）来决定的。可根据应用的需求，配置这些寄存器连接复用功能模块到对应引脚，并开启外设时钟。
  - 如果需要使用复用功能的默认功能，比如表2的PA8作为TMR1_CH1（见4.4.2章节），直接使能TMR1及GPIOA时钟并配置PA8为推挽复用输出即可。
  - 如果需要使用复用功能的重映射功能，比如表2的PB8作为I2C1_SCL（见4.4.3章节），除了使能I2C1及GPIOB时钟并配置PB8为带上拉的开漏复用输出外，还需开启IOMUX时钟，并配置IO复用重映射寄存器x（IOMUX_REMAPx）。

注意：IO复用重映射寄存器x（IOMUX_REMAPx）的具体含义请参考 ***表4. 外设复用映射表*** *及对应型号技术手册RM的IOMUX章节。*

当需要使用不同外设复用功能时的引脚配置（见4.4章节）：

- 如果外设引脚需要作为复用输出则对应的引脚配置成推挽复用/开漏复用输出
- 如果外设引脚需要作为复用输入则对应的引脚配置成浮空输入/上拉输入/下拉输入
- ADC 外设需要将模拟通道对应的引脚配置为模拟输入
- I2C 外设需要对应引脚作为双向复用功能时，需把对应的引脚配置开漏复用模式

外设复用映射表如下：

表 4. 外设复用映射表

外设	复用定义	复用引脚
SPI1	SPI1_MUX_01/ SPI1_GMUX_0001	spi1_cs/i2s1_ws(pa15),spi1_sck/i2s1_ck(pb3), spi1_miso(pb4),spi1_mosi/i2s1_sd(pb5),i2s1_mck(pb6)
SPI2	SPI2_GMUX_0001	spi2_cs/i2s2_ws(pa15),spi2_sck/i2s2_ck(pb3), spi2_miso(pb4),spi2_mosi/i2s2_sd(pb5),i2s2_mck(pc7)
I2C1	I2C1_MUX/ I2C1_GMUX_0001	i2c1_scl(pb8),i2c1_sda(pb9), i2c1_smba(pb5)
	I2C1_GMUX_0011 (仅413)	i2c1_scl(pf6), i2c1_sda(pf7) , i2c1_smba(pb5)
	I2C1_GMUX_0010 (仅415)	

外设	复用定义	复用引脚
I2C2 (仅413)	I2C2_GMUX_0001	i2c2_scl(pf6), i2c2_sda(pf7) , i2c2_smba(pa9)
	I2C2_GMUX_0010	i2c2_scl(pa8), i2c2_sda(pc9) , i2c2_smba(pa9)
	I2C2_GMUX_0011	i2c2_scl(pa8), i2c2_sda(pb4) , i2c2_smba(pa9)
I2C2 (仅415)	I2C2_GMUX_0001	i2c2_scl(pa8), i2c2_sda(pc9), i2c2_smba(pa9)
	I2C2_GMUX_0010	i2c2_scl(pa8), i2c2_sda(pb4), i2c2_smba(pa9)
	I2C2_GMUX_0011	i2c2_scl(pf6), i2c2_sda(pf7), i2c2_smba(pa9)
USART1	USART1_MUX/ USART1_GMUX_0001	usart1_tx(pb6),usart1_rx(pb7)
USART3	USART3_MUX_01/ USART3_GMUX_0001	usart3_tx(pc10), usart3_rx(pc11), usart3_ck(pc12), usart3_cts(pb13), usart3_rts(pb14)
	USART3_GMUX_0010/ USART3_GMUX_0010 (仅415)	usart3_tx(pa7), usart3_rx(pa6), usart3_ck(pa5), usart3_cts(pb1), usart3_rts(pb0)
UART4	UART4_GMUX_0001	uart4_tx(pf4), uart4_rx(pf5)
TMR1	TMR1_MUX_01/ TMR1_GMUX_0001	tmr1_ext(pa12),tmr1_ch1(pa8),tmr1_ch2(pa9), tmr1_ch3(pa10),tmr1_ch4(pa11),tmr1_brkin(pa6), tmr1_ch1c(pa7),tmr1_ch2c(pb0),tmr1_ch3c(pb1)
	TMR1_GMUX_0010 (仅415)	tmr1_ext(pa0), tmr1_ch1(pc6), tmr1_ch2(pc7), tmr1_ch3(pc8), tmr1_ch4(pc9), tmr1_brkin(pa6), tmr1_ch1c(pa7), tmr1_ch2c(pb0), tmr1_ch3c(pb1)
TMR2	TMR2_MUX_01/ TMR2_GMUX_001	tmr2_ch1_ext(pa15),tmr2_ch2(pb3), tmr2_ch3(pa2),tmr2_ch4(pa3)
	TMR2_MUX_10/ TMR2_GMUX_010	tmr2_ch1_ext(pa0),tmr2_ch2(pa1), tmr2_ch3(pb10),tmr2_ch4(pb11)
	TMR2_MUX_11/ TMR2_GMUX_011	tmr2_ch1_ext(pa15),tmr2_ch2(pb3), tmr2_ch3(pb10),tmr2_ch4(pb11)
	TMR2ITR1_GMUX (仅413)	usbdev sof as input to tmr2_int.1
TMR3	TMR3_MUX_10/ TMR3_GMUX_0010	tmr3_ch1(pb4),tmr3_ch2(pb5), tmr3_ch3(pb0),tmr3_ch4(pb1)
	TMR3_MUX_11/ TMR3_GMUX_0011	tmr3_ch1(pc6),tmr3_ch2(pc7), tmr3_ch3(pc8),tmr3_ch4(pc9)
TMR5	TMR5_GMUX_001	tmr5_ch1(pf4),tmr5_ch2(pf5)
		tmr5_ch3(pa2), tmr5_ch4(pa3) (仅415,413无ch3/ch4)
	TMR5CH4_MUX/ TMR5CH4_GMUX	lick connected to tmr5_ch4 input capture for calibration
TMR9	TMR9_GMUX	tmr9_ch1(pb14), tmr9_ch2(pb15)
TMR10	TMR10_GMUX	tmr10_ch1(pa6)
TMR11	TMR11_GMUX	tmr11_ch1(pa7)
CAN1	CAN_MUX_10/ CAN1_GMUX_0010	can_rx(pb8),can_tx(pb9)

外设	复用定义	复用引脚
CAN2 (仅413)	CAN2_GMUX_0001	can2_rx(pb5),can2_tx(pb6)
PD01	PD01_MUX/ PD01_GMUX	pd0/pd1 mapping on osc_in/osc_out
ADC1	ADC1_ETP_MUX/ ADC1_ETP_GMUX	adc1 external trigger preempted conversion muxing
	ADC1_ETO_MUX/ ADC1_ETO_GMUX	adc1 external trigger ordinary conversion muxing
ADC2 (仅413)	ADC2_ETP_MUX / ADC2_ETP_GMUX	adc2 external trigger preempted conversion muxing
	ADC2_ETO_MUX/ ADC2_ETO_GMUX	adc2 external trigger ordinary conversion muxing
SWJTAG	SWJTAG_CONF_001/ SWJTAG_GMUX_001	full swj enabled (jtag-dp + sw-dp) but without jtrst
	SWJTAG_CONF_010/ SWJTAG_GMUX_010	jtag-dp disabled and sw-dp enabled
	SWJTAG_CONF_100/ SWJTAG_GMUX_100	full swj disabled (jtag-dp + sw-dp)
SDIO1	SDIO1_GMUX_0100	sdio1_ck(pc4), sdio1_cmd(pc5), sdio1_d0(pc0), sdio1_d1(pc1), sdio1_d2(pc2), sdio1_d3(pc3), sdio1_d4(pa4), sdio1_d5(pa5), sdio1_d6(pa6), sdio1_d7(pa7)
	SDIO1_GMUX_0101	sdio1_ck(pc4), sdio1_cmd(pc5), sdio1_d0(pa4), sdio1_d1(pa5), sdio1_d2(pa6), sdio1_d3(pa7)
	SDIO1_GMUX_0110	sdio1_ck(pa2), sdio1_cmd(pa3), sdio1_d0(pc0), sdio1_d1(pc1), sdio1_d2(pc2), sdio1_d3(pc3), sdio1_d4(pa4), sdio1_d5(pa5), sdio1_d6(pa6), sdio1_d7(pa7)
	SDIO1_GMUX_0111	sdio1_ck(pa2), sdio1_cmd(pa3), sdio1_d0(pa4), sdio1_d1(pa5), sdio1_d2(pa6), sdio1_d3(pa7)
EXT_SPIM (仅413)	EXT_SPIM_EN_MUX/ EXT_SPIM_GMUX_1000	enable external spi-flash interface
	EXT_SPIM_GMUX_1001	spim_sck(pb1), spim_cs(pa8), spim_io0(pb10), spim_io1(pb11), spim_io2(pb7), spim_sio3(pb6)
CMP (仅415)	CMP_MUX_01	cmp1_out connect pa6, cmp2_out connect pa7
	CMP_MUX_10	cmp1_out connect pa11, cmp2_out connect pa12
IOMUX REMAP8 (仅415)	TMR1_BK1_CMP_GMUX_10	cmp tmr1_bpr1 connect tmr1 bk1
	TMR1_BK1_CMP_GMUX_11	cmp tmr1_bpr1 and io connect tmr1 bk1
	TMR1_CH1_CMP_GMUX_10	cmp connect tmr1 ch1
	TMR1_CH1_CMP_GMUX_11	cmp and io connect tmr1 ch1
	TMR2_CH4_CMP_GMUX_10	cmp connect tmr2 ch4
	TMR2_CH4_CMP_GMUX_11	cmp and io connect tmr2 ch4
	TMR3_CH1_CMP_GMUX_10	cmp connect tmr3 ch1
	TMR3_CH1_CMP_GMUX_11	cmp and io connect tmr3 ch1

3.2 特殊 I/O

3.2.1 硬件抢占引脚

某些引脚不管 GPIO 配置为何种模式，都会被特定的硬件功能占用，称为硬件抢占。

表 5. 硬件抢占引脚

引脚名称	抢占使能位	说明
PA0	PWC_CTRLSTS[8] = 1	抢占使能位有效之后，PA0 引脚直接作为 PWC 的 WKUP 功能使用
PA11	CRM_APB1EN[23]=1	抢占使能位有效之后，PA11 作为 USB_DM 通道使用
PA12	CRM_APB1EN[23]=1	抢占使能位有效之后，PA12 作为 USB_DP 通道使用
PC13	CRM_APB1EN[27]=1& (BPR_CTRL[0]=1 | BPR_RTCCAL[8]＝1 | BPR_RTCCAL[7]＝1)	抢占使能位有效之后，PC13 作为 RTC 通道使用
PC14	CMR_BPDC[0]=1	抢占使能位有效之后，PC14 作为 LEXT 通道使用
PC15	CMR_BPDC[0]=1	抢占使能位有效之后，PC15 作为 LEXT 通道使用

上表中，PC13、PC14以及PC15属于电池供电域引脚。电池供电域由VDD或VBAT引脚供电，当VDD主电源被切断时，电池供电域自动切换至VBAT引脚供电，以保障RTC/ERTC正常工作。

- 当电池供电域由VDD供电时，由于这三个引脚只能够推动有限的电流（3 mA），因此作为输出引脚时不能作为电流源（如驱动LED）。
- 当电池供电域由VBAT供电时，PC13可以作为TAMPER引脚、RTC/ERTC闹钟或秒输出，PC14和PC15只能用于LEXT引脚。

3.2.2 SWJTAG 调试端口

在进行芯片调试时，为了防止其他外设对调试端口的干扰，导致不能被调试，配置好后的调试端口引脚，不管对应引脚的 GPIO 寄存器被配置为任何模式，都会一直保持为调试端口。

为了在调试期间可以使用更多引脚，通过设置复用重映射和调试 I/O 配置寄存器（IOMUX_REMAP）的 SWJTAG_MUX [2：0]位或复用重映射和调试 I/O 配置寄存器 7（IOMUX_REMAP7）的 SWJTAG_GMUX [2：0]位，可以改变上述重映射配置。具体配置方法见 5.2 案例 2--SWJTAG 接口复用。

表 6. 调试端口映射

SWJTAG_MUX [2：0]或 SWJTAG_GMUX [2：0]	SWJI/O 引脚分配				
	PA13/JTMS/ SWDIO	PA14/JTCK/ SWCLK	PA15/JTDI	PB3/JTDO/ TRACESWO	PB4/NJTRST
000	√	√	√	√	√
001	√	√	√	√	x
010	√	√	x	x	x
100	x	x	x	x	x
其它	-	-	-	-	-

注意：√ 表示该引脚被强制分配给调试端口，x 表示该引脚可以释放给其他外设使用。

3.2.3 晶体振荡器复用引脚

对于高速外部晶振（HEXT）和低速外部晶振（LEXT）的复用引脚：

- 振荡器关闭的状态下，相关引脚可用作GPIO
- 振荡器使能状态下，相应引脚的GPIO配置无效
- 振荡器处于bypass模式（使用外部时钟源）时，HEXT_IN/LEXT_IN为振荡器时钟输入引脚，HEXT_OUT/LEXT_OUT可做GPIO使用

4 GPIO 固件驱动程序 API

Artery 提供的固件驱动程序包含了一系列固件函数来管理 GPIO 的下列功能：

- GPIO寄存器复位
- 初始化配置
- 读取输入端口或某个输入引脚
- 读取输出端口或某个输出引脚
- 设置或清除某个引脚的输出
- 锁定引脚
- 引脚的复用功能配置

4.1 输出模式

GPIO提供了两种不同类型的输出模式分别是，推挽输出和开漏输出，下面是输出模式的配置示例：

```
gpio_init_struct.gpio_pins = GPIO_PINS_X;          /* 引脚选择 */
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;    /* 可选无、上拉或下拉 */
gpio_init_struct.gpio_mode = GPIO_MODE_OUTPUT; /* 选择输出模式 */
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL; /* 推挽或开漏 */
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER; /* 驱动力 */
gpio_init(GPIOX, &gpio_init_struct);
```

4.2 输入模式

GPIO提供了三种不同类型的输入模式分别是，浮空输入、上拉输入以及下拉输入，下面是输入模式的配置示例：

```
gpio_init_struct.gpio_pins = GPIO_PINS_X;          /* 引脚选择 */
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;    /* 可选无、上拉或下拉 */
gpio_init_struct.gpio_mode = GPIO_MODE_INPUT; /* 选择输入模式 */
gpio_init_struct.gpio_out_type    = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init(GPIOX, &gpio_init_struct);
```

4.3 模拟模式

当需要使用ADC或COMP通道作为输入时，需要将相应的引脚配置为模拟模式，下面是模拟模式的配置示例：

```
gpio_init_struct.gpio_pins = GPIO_PINS_X;          /* 引脚选择 */
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;    /* 可选无、上拉或下拉 */
```

```
gpio_init_struct.gpio_mode = GPIO_MODE_ANALOG; /* 选择模拟模式 */
gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init(GPIOX, &gpio_init_struct);
```

4.4 复用模式

使用复用模式时，针对不同外设的不同引脚，GPIO配置有所差异，请参考下表。

表 7. 不同外设复用时的 GPIO 配置

外设引脚		功能	GPIO配置
TMR	TMRx_CHx	输入捕获通道x	浮空输入
		输出比较通道x	推挽复用输出
	TMRx_EXT	外部触发时钟输入	浮空输入
	TMRx_CHxC（仅高级定时器）	互补输出通道x	推挽复用输出
	TMR1_BRK（仅高级定时器）	刹车输入	浮空输入
USART	USARTx_TX	全双工模式	推挽复用输出
		半双工同步模式	推挽复用输出
	USARTx_RX	全双工模式	浮空/上拉输入
		半双工同步模式	未使用
	USARTx_CK	同步模式	推挽复用输出
	USARTx_RTS	硬件流控制	推挽复用输出
	USARTx_CTS	硬件流控制	浮空/上拉输入
SPI	SPIx_SCK	主模式	推挽复用输出
		从模式	浮空输入
	SPIx_MOSI	全双工模式/主模式	推挽复用输出
		全双工模式/从模式	浮空/上拉输入
		简单的双向数据线/主模式	推挽复用输出
		简单的双向数据线/从模式	未使用
	SPIx_MISO	全双工模式/主模式	浮空/上拉输入
		全双工模式/从模式	推挽复用输出
		简单的双向数据线/主模式	未使用
		简单的双向数据线/从模式	推挽复用输出
	SPIx_CS	硬件主/从模式	浮空/上拉/下拉输入
		硬件主模式/ CS输出使能	推挽复用输出
		软件模式	未使用
	I2Sx_WS	主模式	推挽复用输出

外设引脚		功能	GPIO配置
I2S		从模式	浮空输入
	I2Sx_CK	主模式	推挽复用输出
		从模式	浮空输入
	I2Sx_SD	发送器	推挽复用输出
		接收器	浮空/上拉/下拉输入
	I2Sx_MCK	主模式	推挽复用输出
		从模式	未使用
I2C	I2Cx_SCL	I2C时钟	（带上拉）开漏复用输出
	I2Cx_SDA	I2C数据	（带上拉）开漏复用输出
	I2Cx_SMBA	SMBUS Alert	（带上拉）开漏复用输出
CAN	CAN_TX	CAN发送引脚	推挽复用输出
	CAN_RX	CAN接收引脚	浮空/上拉输入
USB	USBFS1_D- (仅413) / OTGFS1_D- (仅415)	一旦使能了 USB 模块，这些引脚会自动连接到内部 USB 收发器，见3.2.1硬件抢占引脚章节	
	USBFS1_D+ (仅413) / OTGFS1_D+ (仅415)		
	USBFS1_SOF (仅413) / OTGFS1_SOF (仅415)	推挽复用输出	
	OTGFS1_VBUS (仅415)	浮空输入	
	OTGFS1_ID (仅415)	上拉输入	
SDIO	SDIO1_CK	SDIO时钟	推挽复用输出
	SDIO1_CMD	SDIO命令	推挽复用输出
	SDIO1_Dx (x=0…7)	SDIO数据	推挽复用输出
TAMPER-RTC		RTC / ERTC输出	由硬件强制设置，见3.2.1 硬件抢占引脚章节
		侵入事件输入	
CLKOUT		时钟输出	推挽复用输出
EXTINT		外部中断输入	浮空/上拉/下拉输入

下面介绍几种常用外设的配置示例，更多外设配置例程请参考上表以及BSP的example目录。

4.4.1 USART I/O 复用模式配置

```
/* 使能 GPIOA / USART1 的时钟 */
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_USART1_PERIPH_CLOCK, TRUE);
```

```
/* 将 USART1_Tx 配置为推挽复用输出 */
gpio_init_struct.gpio_pins = GPIO_PINS_9;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init(GPIOA, &gpio_init_struct);
/* 将 USART1_Tx 配置为上拉输入 */
gpio_init_struct.gpio_pins = GPIO_PINS_10;
gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
gpio_init_struct.gpio_pull = GPIO_PULL_UP;
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init(GPIOA, &gpio_init_struct);
```

4.4.2 TMR I/O 复用模式配置

```
/* 使能 GPIOA/ GPIOB/ TMR1 的时钟 */
crm_periph_clock_enable(CRM_TMR1_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);

/* 将TMR1的4路输出比较通道引脚(PA8/PA9/PA10/PA11)配置为推挽复用输出 */
gpio_init_struct.gpio_pins = GPIO_PINS_8 | GPIO_PINS_9 | GPIO_PINS_10 | GPIO_PINS_11;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init(GPIOA, &gpio_init_struct);
/* 将TMR1的3路互补输出通道引脚(PB13/B14/PB15)配置为推挽复用输出 */
gpio_init_struct.gpio_pins = GPIO_PINS_13 | GPIO_PINS_14 | GPIO_PINS_15;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init(GPIOB, &gpio_init_struct);
```

```
/* 将TMR1的刹车输入TMR1_BRK引脚(PB12)配置为浮空输入 */
gpio_init_struct.gpio_pins = GPIO_PINS_12;
gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init(GPIOB, &gpio_init_struct);
/* 将TMR1的外部触发时钟输入TMR1_EXT引脚(PA12)配置为浮空输入 */
gpio_init_struct.gpio_pins = GPIO_PINS_12;
gpio_init_struct.gpio_mode = GPIO_MODE_ INPUT;
gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init(GPIOA, &gpio_init_struct);
```

4.4.3 I2C I/O 复用模式配置

```
/* 使能 GPIOB/ I2C1 的时钟 */
crm_periph_clock_enable(CRM_I2C1_PERIPH_CLOCK, TRUE);
crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);
/* 使能 IOMUX的时钟 */
crm_periph_clock_enable(CRM_IOMUX_PERIPH_CLOCK, TRUE);
/* 配置 IOMUX_REMAP5选择将I2C1的SCL/SDA复用至PB8/PB9引脚 */
gpio_pin_remap_config(I2C1_GMUX_0001,TRUE);

/* I2C的GPIO配置为带上拉开漏复用输出 */
gpio_initstructure.gpio_out_type = GPIO_OUTPUT_OPEN_DRAIN;
gpio_initstructure.gpio_pull = GPIO_PULL_UP;
gpio_initstructure.gpio_mode = GPIO_MODE_MUX;
gpio_initstructure.gpio_drive_strength = GPIO_DRIVE_STRENGTH_MODERATE;
/* 配置I2C1_SCL（PB8） */
gpio_initstructure.gpio_pins = GPIO_PINS_8;
gpio_init(GPIOB, &gpio_initstructure);
/* 配置I2C1_SDA（PB9） */
gpio_initstructure.gpio_pins = GPIO_PINS_9;
gpio_init(GPIOB, &gpio_initstructure);
```

5 GPIO 案例

注：所有 project 都是基于 keil 5 而建立，若用户需要在其他编译环境上使用，请参考 AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates 中各种编译环境（例如 IAR6/7/8,keil 4/5）进行简单修改即可。

5.1 案例 1--LED 翻转

5.1.1 功能简介

通过 GPIO 置位/复位和系统时钟（Systick）延时来对 LED 进行翻转。

5.1.2 资源准备

1) 硬件环境：

AT32F413/ AT32F415 的 AT-START BOARD

2) 软件环境：

\project\at_start_f413\examples\gpio\led_toggle\mdk_v5

5.1.3 软件设计

1) 配置流程

- 配置系统时钟；
- 调用 AT32 板级支持包初始化延时函数和 LED；
- 翻转 LED。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  system_clock_config();                    /* 系统时钟配置，默认 192 MHz */
  at32_board_init();                        /* 初始化延时函数和 LED */
  while(1)
  {
    at32_led_toggle(LED2);                  /* 翻转 LED2 */
    delay_ms(200);                          /* 延时 200 ms */
    at32_led_toggle(LED3);                  /* 翻转 LED3 */
    delay_ms(200);                          /* 延时 200 ms */
    at32_led_toggle(LED4);                  /* 翻转 LED4 */
    delay_ms(200);                          /* 延时 200 ms */
  }
}
```

- at32_board_init();板级支持包初始化代码描述

```
void at32_board_init()
{
  delay_init();                             /* 初始化延时函数 */
```

```
  at32_led_init(LED2);                                  /* 初始化 AT-START 板的 LED */
  at32_led_init(LED3);
  at32_led_init(LED4);
  at32_led_off(LED2);                                   /* 关闭 AT-START 板的 LED */
  at32_led_off(LED3);
  at32_led_off(LED4);
  at32_button_init();                                   /* 初始化 AT-START 板的 USER 按键 */
}
```

■ at32_led_init();LED 初始化代码描述

```
void at32_led_init(led_type led)
{
  gpio_init_type gpio_init_struct;
  crm_periph_clock_enable(led_gpio_crm_clk[led], TRUE);  /* 使能 LED 使用的 GPIOx 时钟 */
  gpio_default_para_init(&gpio_init_struct);  /* 设置 GPIO 默认参数 */
  /* 配置 LED 的 GPIO 脚为推挽输出 */
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_OUTPUT;
  gpio_init_struct.gpio_pins = led_gpio_pin[led];
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init(led_gpio_port[led], &gpio_init_struct);
}
/******************** define led @file at32f413_board.h********************/
typedef enum
{
  LED2 = 0,
  LED3 = 1,
  LED4 = 2
} led_type;
#define LED_NUM 3
#if defined (AT_START_F413_V1)
#define LED2_PIN GPIO_PINS_2
#define LED2_GPIO GPIOC
#define LED2_GPIO_CRM_CLK CRM_GPIOC_PERIPH_CLOCK
#define LED3_PIN GPIO_PINS_3
#define LED3_GPIO GPIOC
#define LED3_GPIO_CRM_CLK CRM_GPIOC_PERIPH_CLOCK
```

```
#define LED4_PIN GPIO_PINS_5
#define LED4_GPIO GPIOC
#define LED4_GPIO_CRM_CLK CRM_GPIOC_PERIPH_CLOCK
#endif
/* at-start led resouce array */
gpio_type *led_gpio_port[LED_NUM] = {LED2_GPIO, LED3_GPIO, LED4_GPIO};
uint16_t led_gpio_pin[LED_NUM] = {LED2_PIN, LED3_PIN, LED4_PIN};
crm_periph_clock_type led_gpio_crm_clk[LED_NUM] = {LED2_GPIO_CRM_CLK, LED3_GPIO_CRM_CLK, LED4_GPIO_CRM_CLK};
```

- at32_led_toggle();LED 翻转代码描述

```
void at32_led_toggle(led_type led)
{
  if(led > (LED_NUM - 1))                                  /* 判断 LED 是否有效 */
    return;
  if(led_gpio_pin[led])
    led_gpio_port[led]->odt ^= led_gpio_pin[led];   /* 通过异或配置 GPIOx_ODT 寄存器翻转 GPIO */
}
```

5.1.4 实验效果

上电运行，观察到 LED2、LED3 和 LED4 以间隔 200ms 时间交替的进行翻转，符合程序设计预期。

5.2 案例 2--SWJTAG 接口复用

5.2.1 功能简介

如 3.2.2 章节描述，SWJTAG 接口使用的引脚可复用作 GPIO，本案例对此功能进行展示。

5.2.2 资源准备

1) 硬件环境：

AT32F413/ AT32F415 的 AT-START BOARD

2) 软件环境：

\project\at_start_f413\examples\gpio\swjtag_remap\mdk_v5

5.2.3 软件设计

1) 配置流程

- 配置系统时钟；
- 调用 AT32 板级支持包初始化延时函数及 USER 按键；
- 配置 GPIOx 及 IOMUX 时钟，并初始化 SWJTAG 接口使用的引脚为推挽输出；
- 考虑到禁用 SWJTAG 后，用户再次使用 AT-START 板 SW 接口下载程序可能会失败，将

程序设计为，当检测到 USER 按键摁下，则开启 SWJTAG 重映射并点亮 LED3 用以指示，然后不断翻转 SWJTAG 接口使用的引脚。

2) 代码介绍

■ main 函数代码描述

```
int main(void)
{
  system_clock_config();                                  /* 系统时钟配置，默认 192 MHz */
  at32_board_init();                                      /* 初始化延时函数和 LED，代码描述见 5.1.3 */
                                                          /* 开启 IOMUX / GPIOA / GPIOB 时钟 */
  crm_periph_clock_enable(CRM_IOMUX_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOB_PERIPH_CLOCK, TRUE);
  swj_dp_config();                                        /* 配置 SWJTAG 接口使用的引脚为推挽输出 */
  while(1)
  {
    if(USER_BUTTON == at32_button_press())                /* 等待 USER 按键摁下 */
    {
      gpio_pin_remap_config(SWJTAG_CONF_100, TRUE);       /* 禁用 SWD 和 JTAG */
      at32_led_on(LED3);                                  /* 点亮 LED3 指示 SWJTAG 复用开启 */
    }
    gpio_pins_toggle(GPIOA, GPIO_PINS_13);                /* 翻转 PA13 jtms/swdat */
    delay_us(200);                                        /* 延时 200 us */
    gpio_pins_toggle(GPIOA, GPIO_PINS_14);                /* 翻转 PA14 jtck/swclk */
    delay_us(200);                                        /* 延时 200 us */
    gpio_pins_toggle(GPIOA, GPIO_PINS_15);                /* 翻转 PA15 jtdi */
    delay_us(200);                                        /* 延时 200 us */
    gpio_pins_toggle(GPIOB, GPIO_PINS_3);                 /* 翻转 PB3 jtdo */
    delay_us(200);                                        /* 延时 200 us */
    gpio_pins_toggle(GPIOB, GPIO_PINS_4);                 /* 翻转 PB4 jtrst */
    delay_us(200);                                        /* 延时 200 us */
  }
}
```

■ swj_dp_config();函数代码描述

```
void swj_dp_config(void)
{
  gpio_init_type  gpio_init_struct;
  /* 配置 SWJTAG 接口使用的引脚 PA13/PA14/PA15/PB3/PB4 为推挽输出 */
  gpio_init_struct.gpio_pins = GPIO_PINS_13 | GPIO_PINS_14 | GPIO_PINS_15;
  gpio_init_struct.gpio_mode = GPIO_MODE_OUTPUT;
  gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
  gpio_init_struct.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
```

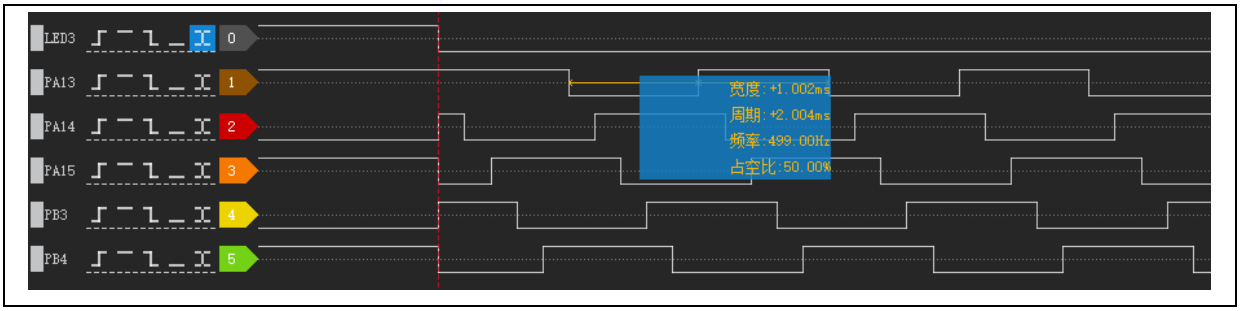
```
  gpio_init(GPIOA, &gpio_init_struct);
  gpio_init_struct.gpio_pins = GPIO_PINS_3 | GPIO_PINS_4;
  gpio_init(GPIOB, &gpio_init_struct);
}
```

■ gpio_pins_toggle();函数代码描述

```
void gpio_pins_toggle(gpio_type* gpio_x, uint16_t gpio_pin)
{
  gpio_x->odt ^= gpio_pin;              /* 通过异或配置 GPIOx_ODT 寄存器翻转 GPIO */
}
```

5.2.4 实验效果

图 3. LA 抓取 SWJTAG 接口复用实验效果



USER 按键摁下后，LED3 点亮。同时使用逻辑分析仪抓取 PA13/PA14/PA15/PB3/PB4 波形，程序设计每 200us 渐次翻转，由上图可知，翻转周期约 1ms，基本符合程序设计预期。

6 文档版本历史

表 8. 文档版本历史

日期	版本	变更
2022.04.02	2.0.0	最初版本

重要通知-请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途（及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：（A）对安全性有特别要求的应用，例如：生命支持、主动植入设备或对产品功能安全有要求的系统；（B）航空应用；（C）航天应用或航天环境；（D）武器，且/或（E）其他可能导致人身伤害、死亡及财产损害的应用。如果采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险及法律责任仍将由采购商单独承担，且采购商应独力负责在前述应用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。