|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 21 | CLK\_RTC\_RTC |  | RW | 0x1 |
| 20 | CLK\_CLK\_ROSC |  | RW | 0x1 |
| 19 | CLK\_CLK\_ROM |  | RW | 0x1 |
| 18 | CLK\_SYS\_RESETS |  | RW | 0x1 |
| 17 | CLK\_PWM |  | RW | 0x1 |
| 16 | CLK\_CLK\_PSM |  | RW | 0x1 |
| 15 | CLK\_PLL\_USB |  | RW | 0x1 |
| 14 | CLK\_PLL\_PLL |  | RW | 0x1 |
| 13 | CLK\_SYS\_PIO1 |  | RW | 0x1 |
| 12 | CLK\_SYS\_PIO0 |  | RW | 0x1 |
| 11 | CLK\_PADS |  | RW | 0x1 |
| 10 | CLK\_CLK\_VREG\_AND\_CHIP\_CHIP |  | RW | 0x1 |
| 9 | CLK\_CLK\_JTAG |  | RW | 0x1 |
| 8 | CLK\_CLK\_IO |  | RW | 0x1 |
| 7 | CLK\_SYS\_I2C1 |  | RW | 0x1 |
| 6 | CLK\_SYS\_I2C0 |  | RW | 0x1 |
| 5 | CLK\_SYS\_DMA |  | RW | 0x1 |
| 4 | CLK\_BUSFABRIC |  | RW | 0x1 |
| 3 | CLK\_CLK\_BUSCTRL |  | RW | 0x1 |
| 2 | CLK\_CLK\_ADC |  | RW | 0x1 |
| 1 | CLK\_ADC\_ADC |  | RW | 0x1 |
| 0 | CLK\_SYS\_CLOCKS |  | RW | 0x1 |

表248。唤醒\_EN1

寄存器

#### 时钟:WAKE\_EN1寄存器

**偏移量**:0xa4

描述

使能唤醒模式时钟

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:15 | Reserved. | - | - | - |
| 14 | CLK\_CLK\_XOSC |  | RW | 0x1 |
| 13 | CLK\_CLK\_XIP |  | RW | 0x1 |
| 12 | CLK\_SYS\_WATCHDOG |  | RW | 0x1 |
| 11 | CLK\_USB\_USBCTRL |  | RW | 0x1 |
| 10 | CLK\_CLK\_USBCTRL |  | RW | 0x1 |
| 9 | CLK\_SYS\_UART1 |  | RW | 0x1 |
| 8 | CLK\_PERI\_UART1 |  | RW | 0x1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 7 | CLK\_SYS\_UART0 |  | RW | 0x1 |
| 6 | CLK\_PERI\_UART0 |  | RW | 0x1 |
| 5 | CLK\_CLK\_TIMER |  | RW | 0x1 |
| 4 | CLK\_STUDIO\_TBMAN |  | RW | 0x1 |
| 3 | CLK\_CLK\_SYSINFO |  | RW | 0x1 |
| 2 | CLK\_CLK\_SYSCFG |  | RW | 0x1 |
| 1 | CLK\_SYS\_SRAM 5 |  | RW | 0x1 |
| 0 | CLK\_SYS\_SRAM 4 |  | RW | 0x1 |

表249.睡眠\_EN0

寄存器

#### 时钟:SLEEP\_EN0寄存器

**偏移量**:0xa8

描述

使能睡眠模式时钟

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | CLK\_SYS\_SRAM 3 |  | RW | 0x1 |
| 30 | CLK\_SYS\_SRAM 2 |  | RW | 0x1 |
| 29 | CLK\_SYS\_SRAM 1 |  | RW | 0x1 |
| 28 | CLK\_SYS\_SRAM 0 |  | RW | 0x1 |
| 27 | CLK\_SYS\_SPI1 |  | RW | 0x1 |
| 26 | CLK\_PERI\_SPI1 |  | RW | 0x1 |
| 25 | CLK\_SYS\_SPI0 |  | RW | 0x1 |
| 24 | CLK\_PERI\_SPI0 |  | RW | 0x1 |
| 23 | CLK\_CLK\_SIO |  | RW | 0x1 |
| 22 | CLK\_CLK\_RTC |  | RW | 0x1 |
| 21 | CLK\_RTC\_RTC |  | RW | 0x1 |
| 20 | CLK\_CLK\_ROSC |  | RW | 0x1 |
| 19 | CLK\_CLK\_ROM |  | RW | 0x1 |
| 18 | CLK\_SYS\_RESETS |  | RW | 0x1 |
| 17 | CLK\_PWM |  | RW | 0x1 |
| 16 | CLK\_CLK\_PSM |  | RW | 0x1 |
| 15 | CLK\_PLL\_USB |  | RW | 0x1 |
| 14 | CLK\_PLL\_PLL |  | RW | 0x1 |
| 13 | CLK\_SYS\_PIO1 |  | RW | 0x1 |
| 12 | CLK\_SYS\_PIO0 |  | RW | 0x1 |
| 11 | CLK\_PADS |  | RW | 0x1 |
| 10 | CLK\_CLK\_VREG\_AND\_CHIP\_CHIP |  | RW | 0x1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 9 | CLK\_CLK\_JTAG |  | RW | 0x1 |
| 8 | CLK\_CLK\_IO |  | RW | 0x1 |
| 7 | CLK\_SYS\_I2C1 |  | RW | 0x1 |
| 6 | CLK\_SYS\_I2C0 |  | RW | 0x1 |
| 5 | CLK\_SYS\_DMA |  | RW | 0x1 |
| 4 | CLK\_BUSFABRIC |  | RW | 0x1 |
| 3 | CLK\_CLK\_BUSCTRL |  | RW | 0x1 |
| 2 | CLK\_CLK\_ADC |  | RW | 0x1 |
| 1 | CLK\_ADC\_ADC |  | RW | 0x1 |
| 0 | CLK\_SYS\_CLOCKS |  | RW | 0x1 |

表250。SLEEP\_EN1

寄存器

#### 时钟:SLEEP\_EN1寄存器

**偏移**:0xac

描述

使能睡眠模式时钟

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:15 | Reserved. | - | - | - |
| 14 | CLK\_CLK\_XOSC |  | RW | 0x1 |
| 13 | CLK\_CLK\_XIP |  | RW | 0x1 |
| 12 | CLK\_SYS\_WATCHDOG |  | RW | 0x1 |
| 11 | CLK\_USB\_USBCTRL |  | RW | 0x1 |
| 10 | CLK\_CLK\_USBCTRL |  | RW | 0x1 |
| 9 | CLK\_SYS\_UART1 |  | RW | 0x1 |
| 8 | CLK\_PERI\_UART1 |  | RW | 0x1 |
| 7 | CLK\_SYS\_UART0 |  | RW | 0x1 |
| 6 | CLK\_PERI\_UART0 |  | RW | 0x1 |
| 5 | CLK\_CLK\_TIMER |  | RW | 0x1 |
| 4 | CLK\_STUDIO\_TBMAN |  | RW | 0x1 |
| 3 | CLK\_SYS\_BUSCTRL |  | RW | 0x1 |
| 2 | CLK\_CLK\_SYSCFG |  | RW | 0x1 |
| 1 | CLK\_SYS\_SRAM 5 |  | RW | 0x1 |
| 0 | CLK\_SYS\_SRAM 4 |  | RW | 0x1 |

#### 时钟:ENABLED0寄存器

**偏移**:0xb0

描述

指示时钟使能状态

表251.已启用0

寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | CLK\_SYS\_SRAM 3 |  | RO | 0x0 |
| 30 | CLK\_SYS\_SRAM 2 |  | RO | 0x0 |
| 29 | CLK\_SYS\_SRAM 1 |  | RO | 0x0 |
| 28 | CLK\_SYS\_SRAM 0 |  | RO | 0x0 |
| 27 | CLK\_SYS\_SPI1 |  | RO | 0x0 |
| 26 | CLK\_PERI\_SPI1 |  | RO | 0x0 |
| 25 | CLK\_SYS\_SPI0 |  | RO | 0x0 |
| 24 | CLK\_PERI\_SPI0 |  | RO | 0x0 |
| 23 | CLK\_CLK\_SIO |  | RO | 0x0 |
| 22 | CLK\_CLK\_RTC |  | RO | 0x0 |
| 21 | CLK\_RTC\_RTC |  | RO | 0x0 |
| 20 | CLK\_CLK\_ROSC |  | RO | 0x0 |
| 19 | CLK\_CLK\_ROM |  | RO | 0x0 |
| 18 | CLK\_SYS\_RESETS |  | RO | 0x0 |
| 17 | CLK\_PWM |  | RO | 0x0 |
| 16 | CLK\_CLK\_PSM |  | RO | 0x0 |
| 15 | CLK\_PLL\_USB |  | RO | 0x0 |
| 14 | CLK\_PLL\_PLL |  | RO | 0x0 |
| 13 | CLK\_SYS\_PIO1 |  | RO | 0x0 |
| 12 | CLK\_SYS\_PIO0 |  | RO | 0x0 |
| 11 | CLK\_PADS |  | RO | 0x0 |
| 10 | CLK\_CLK\_VREG\_AND\_CHIP\_CHIP |  | RO | 0x0 |
| 9 | CLK\_CLK\_JTAG |  | RO | 0x0 |
| 8 | CLK\_CLK\_IO |  | RO | 0x0 |
| 7 | CLK\_SYS\_I2C1 |  | RO | 0x0 |
| 6 | CLK\_SYS\_I2C0 |  | RO | 0x0 |
| 5 | CLK\_SYS\_DMA |  | RO | 0x0 |
| 4 | CLK\_BUSFABRIC |  | RO | 0x0 |
| 3 | CLK\_CLK\_BUSCTRL |  | RO | 0x0 |
| 2 | CLK\_CLK\_ADC |  | RO | 0x0 |
| 1 | CLK\_ADC\_ADC |  | RO | 0x0 |
| 0 | CLK\_SYS\_CLOCKS |  | RO | 0x0 |

#### 时钟: ENABLED1寄存器

**偏移量**:0xb4

表252.启用1

寄存器

表253.INTR寄存器

表254.INTE寄存器

描述

指示时钟使能状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:15 | Reserved. | - | - | - |
| 14 | CLK\_CLK\_XOSC |  | RO | 0x0 |
| 13 | CLK\_CLK\_XIP |  | RO | 0x0 |
| 12 | CLK\_SYS\_WATCHDOG |  | RO | 0x0 |
| 11 | CLK\_USB\_USBCTRL |  | RO | 0x0 |
| 10 | CLK\_CLK\_USBCTRL |  | RO | 0x0 |
| 9 | CLK\_SYS\_UART1 |  | RO | 0x0 |
| 8 | CLK\_PERI\_UART1 |  | RO | 0x0 |
| 7 | CLK\_SYS\_UART0 |  | RO | 0x0 |
| 6 | CLK\_PERI\_UART0 |  | RO | 0x0 |
| 5 | CLK\_CLK\_TIMER |  | RO | 0x0 |
| 4 | CLK\_STUDIO\_TBMAN |  | RO | 0x0 |
| 3 | CLK\_SYS\_SYSINFO |  | RO | 0x0 |
| 2 | CLK\_CLK\_SYSCFG |  | RO | 0x0 |
| 1 | CLK\_SYS\_SRAM 5 |  | RO | 0x0 |
| 0 | CLK\_SYS\_SRAM 4 |  | RO | 0x0 |

#### 时钟:INTR寄存器

**偏移量**:0xb8

描述

原始中断

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | CLK\_RESUS |  | RO | 0x0 |

#### 时钟:INTE寄存器

**偏移量**:0xbc

描述

中断使能

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | CLK\_RESUS |  | RW | 0x0 |

#### 时钟:INTF寄存器

**偏移:0xc0**

表255.INTF寄存器

表256.INTS寄存器

描述

中断强制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | CLK\_RESUS |  | RW | 0x0 |

#### 时钟:INTS寄存器

**偏移量**:0xc4

描述

强制屏蔽后的屏蔽状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | CLK\_RESUS |  | RO | 0x0 |

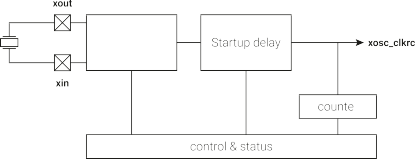
* 1. 晶体振荡器（XOSC）
     1. 概述

晶体振荡器（XOSC）使用外部晶体来产生精确的参考时钟。RP 2040支持1 MHz至15 MHz晶振，RP 2040参考设计（参见[使用RP 2040的硬件设计](https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf)，[最小设计示例](https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf#minimal-design-example)）使用12 MHz晶振。参考时钟被分配给PLL，PLL可用于乘以XOSC频率，以提供准确的高速时钟。例如，它们可以生成满足USB接口频率精度要求的48MHz时钟和133MHz最高速度系统时钟。XOSC时钟也是时钟发生器的时钟源，因此可以在需要时直接使用

如果用户已经有一个精确的时钟源，则可以直接将外部时钟驱动到XIN（又名Xi），并禁用振荡器电路。在这种模式下，XIN可以在高达50MHz的频率下驱动

如果用户希望在RP 2040外部使用XOSC时钟，则必须通过一个UART\_gpout时钟发生器将其路由到GPIO不建议直接从XIN（又名Xi）或XOUT（又名XO）中获取

*图33. XOSC概述*



* + - 1. 推荐晶体

*表257. 关键晶体规格。*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 最小 | 典型 | 最大 | 单位 | 注意到 |
| 中心频率 | 12.000 | 12.000 | 12.000 | MHz |  |
| 操作模式 | 基本面-AT | 基频-AT | 基本面-AT |  |  |

为了在典型的工作温度范围内获得最佳性能和稳定性，建议使用Abracon ABM 8 -272-T3。您可以直接从Abracon或授权经销商处采购ABM 8 -272-T3。Abracon ABM 8 -272-T3具有以下规格:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 最小 | 典型 | 最大 | 单位 | 注意到 |
| 操作温度 | -40 |  | +85 | ºC |  |
| 储存温度 | -55岁 |  | +125 | 摄氏度 |  |
| 频率容差（25ºC） | -30 |  | +30 | ppm |  |
| 频率稳定度（25ºC） | -30 |  | +30 | ppm |  |
| 等效串联电阻（R1） |  |  | 50 | Ω |  |
| 并联电容（C0） |  |  | 3.0 | PF |  |
| 负载电容（CL） | 10 | 10 | 10 | PF |  |
| 驱动电平 |  | 10 | 200 | 微瓦 |  |
| 老化 | -五个 |  | +5级 | ppm | @25±3°C，第1年 |
| 绝缘电阻 | 500 |  |  | MΩ | @100Vdc±15V |

即使您使用具有类似规格的晶体，也需要在一定温度范围内测试电路以确保稳定性。

晶体振荡器由VDDIO电压供电因此，Abracon晶体和该特定阻尼电阻器被调谐为3.3V操作。 如果使用不同的IO电压，则需要重新调整。

晶体参数的任何变化都可能导致连接到晶体电路的任何组件不稳定

如果您无法直接从Abracon或其他供应商获得推荐的晶体，请联系[applications@raspberrypi.com](mailto:applications@raspberrypi.com)。

Raspberry Pi Pico已针对Abracon ABM 8 -272-T3晶体的规格进行了专门调整。有关如何将晶振与RP 2040配合使用的示例，请参阅[Pico数据表](https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf#pico-schematic-diagram)附录B中的Raspberry Pi Pico板原理图和[Raspberry Pi Pico设计文件](https://datasheets.raspberrypi.com/pico/RPi-Pico-R3-PUBLIC-20200119.zip)。

* + 1. 使用

该XOSC是禁用芯片启动和RP2040引导使用环形振荡器（ROSC）。要启动XOSC，程序员必须设置CTRL\_ENABLE寄存器。XOSC不能立即使用，因为振荡需要时间才能达到足够的振幅。该时间将取决于所选择的晶体，但大约为几毫秒。XOSC集成了一个由STARTUP\_DELAY寄存器控制的定时器，用于在XOSC时钟可用时自动管理该寄存器并设置标志（STATUS\_STABLE）。

* + 1. 启动延迟

STARTUP\_DELAY寄存器指定在使用晶振之前必须从晶振看到多少个时钟周期。它以256的倍数表示。SDKxosc\_init函数设置此值。对于以12MHz运行XOSC的RP 2040参考设计（参见[使用RP 2040的硬件设计](https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf)，[最小设计示例](https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf#minimal-design-example)）当定时器超时时，STATUS\_STABLE标志将被设置以指示可以使用XOSC输出

在启动XOSC之前，编程器必须确保STARTUP\_DELAY寄存器配置正确。所需值可通过以下公式计算:



因此，使用12 MHz晶振和1 ms等待时间，计算结果为:



**注意**

该值将四舍五入到最接近的整数，因此等待时间将略高于1ms

* + 1. XOSC计数器

COUNT寄存器提供了一种管理短软件延迟的方法。将一个值写入寄存器会自动触发寄存器，使其以XOSC频率开始倒计时至零然后程序员简单地轮询寄存器，直到它达到零。这比在软件循环中使用NOP更可取，因为它独立于核心时钟频率、编译器和编译代码的执行时间

* + 1. 休眠模式

在休眠模式下（参见第2.11.3节），可以暂停所有片内时钟以节省功耗。这在电池供电的应用中特别有用。通过外部事件（如GPIO引脚上的边沿）或片内RTC的中断，将RP 2040从DORMANT模式唤醒。必须在进入休眠模式之前进行配置。如果RTC用于触发唤醒，则必须从外部源对其进行时钟控制要进入休眠模式，程控仪必须切换所有内部时钟，从XOSC或ROSC驱动，并停止PLL。然后，必须将特定的32位值写入所选振荡器（XOSC或ROSC）中的DORMANT寄存器，以停止其振荡。退出休眠模式时，所选振荡器将重新启动。如果选择XOSC，则频率将更精确，但由于启动延迟，重启时间更长（在RP 2040参考设计上>1ms（参见[使用RP 2040的硬件设计](https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf)，[最小设计示例](https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf#minimal-design-example)））。如果选择ROSC，则频率精度较低，但启动时间非常短（约1μs）。

**注意**

PLL必须在进入休眠模式前停止

SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_xosc/xosc.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_xosc/xosc.c#L52-L57)第52-57

1. public voidrun（）{
2. //停止:这将停止xosc，直到被irq唤醒
3. xosc\_hw->dormant=XOSC\_DORMANT\_VALUE\_DORMANT;
4. //等待它在被唤醒后变得稳定
5. while（！(xosc\_hw->status&XOSC\_STATUS\_STABLE\_BITS））;

57}

**警告**

如果在进入休眠模式之前没有配置IRQ，XOSC或ROSC将永远不会重新启动。

2.11.5.2有关使用XOSC的DORMANT模式的完整示例，请参阅www.example.com部分。

* + 1. 程序员模型

SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2040/hardware\_structs/include/hardware/structs/xosc.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2040/hardware_structs/include/hardware/structs/xosc.h#L24-L57)第24-57

24typedef struct{

25

26

27

\_REG\_（XOSC\_CTRL\_OFFSET）*//XOSC\_CTRL*

//晶振控制

//0x 00 fff 000 [23:12]:ENABLE（0）:上电时，此字段初始化为DISABLE，

28

29

30

31

32

33

34

芯片从ROSC运行

//0x 00000 fff [11:0]:FREQ\_RANGE（0）:频率范围

io\_rw\_32 ctrl;

\_REG\_（XOSC\_STATUS\_OFFSET）*//XOSC\_STATUS*

//晶振状态

//0x8000000[31]

//0x01000000[24]

:STABLE（0）:振荡器正在运行且稳定

:BADWRITE（0）:无效值已写入CTRL\_ENABLE或

CTRL\_FREQ\_RANGE或休眠

35 *//0x 00001000 [12]:ENABLED（0）:振荡器已启用，但不一定运行且稳定，重置为0*

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

//0x0000003[1:0]

io\_rw\_32 status;

:FREQ\_RANGE（0）:当前频率范围设置，始终读取0

\_REG\_（XOSC\_DORMANT\_OFFSET）*//XOSC\_DORMANT*

//晶体振荡器暂停控制

io\_rw\_32 dormant;

\_REG\_（XOSC\_STARTUP\_OFFSET）//XOSC\_STARTUP

//控制启动延迟

//0x 00100000[20] :X4（0）:将startup\_delay乘以4

//0x 00003 fff [13:0]:DELAY（0xc 4）:256\*xtal\_period的倍数

io\_rw\_32 startup;

int[3];

\_REG\_（XOSC\_OFFSET）*//XOSC\_OFFSET*

//以xosc频率运行的向下计数器，计数到零并停止

//0x00000ff[7:0]

io\_rw\_32计数;

:中文（简体）

xosc\_hw\_t;

56

57*#define xosc\_hw（（xosc\_hw\_t\*）XOSC\_BASE）*

SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_xosc/xosc.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_xosc/xosc.c#L29-L41)第29-41

publicvoid run（）{

30

31

32

33

34

35

36

37

38

39

40

41}

//假设输入为1-15 MHz，如上所述。

xosc\_hw-> cnc =XOSC\_CTRL\_FREQ\_RANGE\_VALUE\_1\_15MHZ;

//设置xosc启动延迟

xosc\_hw->startup =STARTUP\_DELAY;

//既然我们已经设置了频率范围和启动延迟，就设置启用位

hw\_set\_bits（xosc\_hw-> countries，XOSC\_CTRL\_ENABLE\_VALUE\_ENABLE XOSC\_CTRL\_ENABLE\_LSB）;

//等待XOSC稳定

同时（！(xosc\_hw->status&XOSC\_STATUS\_STABLE\_BITS））;

* + 1. 寄存器列表

XOSC寄存器从基址0x40024000（在SDK中定义为XOSC\_BASE）开始。

*表258.第258页 XOSC寄存器*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [Ctrl](#_bookmark1) | 晶振控制 |
| 0x04 | STATUS | 晶振状态 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x08 | DORMANT | 晶体振荡器暂停控制 |
| 0x0C | STARTUP | 控制启动延迟 |
| 0x1c | COUNT | 以XOSC频率运行的递减计数器，计数到零并停止。 |

表259.ctrl寄存器

表260.第260页地位

寄存器

#### [XOSC](#_bookmark0):CTRL寄存器

**偏移**:0x00

描述

晶振控制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23:12 | ENABLE | 上电时，此字段初始化为DISABLE，芯片从ROSC运行。  如果芯片随后被编程为从XOSC运行，则将此字段设置为DISABLE可能会锁定芯片。如果这是一个问题，则从ROSC运行RESSOR\_ref  12位代码旨在提供一些保护，防止意外写入。 无效设置将启用振荡器。  0xd1e→禁用0xfab→启用 | RW | - |
| 11:0 | 频率范围 | 频率范围。 这将重置为0xAA0，并且无法更改。  0xaa0→1\_15MHZ  0xaa1→保留\_1  0xaa2→保留\_2  0xaa3→保留\_3 | RW | - |

#### 状态寄存器

**偏移**:0x04

描述

晶振状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | STABLE | 振荡器运行稳定 | RO | 0x0 |
| 30:25 | Reserved. | - | - | - |
| 24 | BADWRITE | 无效值已写入CTRL\_ENABLE或CTRL\_FREQ\_RANGE或DORMANT | WC | 0x0 |
| 23:13 | Reserved. | - | - | - |
| 12 | ENABLED | 振荡器已启用，但不一定运行和稳定，重置为0 | RO | - |
| 11:2 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 1:0 | 频率范围 | 当前频率范围设置，始终读取00x0→ 1\_15MHZ  0x1→保留\_1  0x2→保留\_2  0x3→保留\_3 | RO | - |

表261.休眠

寄存器

表262.启动

寄存器

表263.计数

寄存器

#### [XOSC](#_bookmark0):DORMANT注册

**偏移量**:0x 08

描述

晶体振荡器暂停控制

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 这用于通过暂停XOSC来节省功率。上电时  无效写入也将选择WAKE  :选择休眠模式前停止PLL  0x77616b65→唤醒 | RW | - |

#### [XOSC](#_bookmark0):STARTUP注册

**偏移**:0x0c

描述

控制启动延迟

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:21 | Reserved. | - | - | - |
| 20 | X4 | 将startup\_delay乘以4。 这对用户来说没有什么价值，因为延迟可以直接编程。 | RW | 0x0 |
| 19:14 | Reserved. | - | - | - |
| 13:0 | DELAY | 256\*xtal\_period的倍数。0xc4的复位值对应于大约50000个周期。 | RW | 0x00c4 |

#### [XOSC](#_bookmark0): COUNT寄存器

**偏移**:0x1c

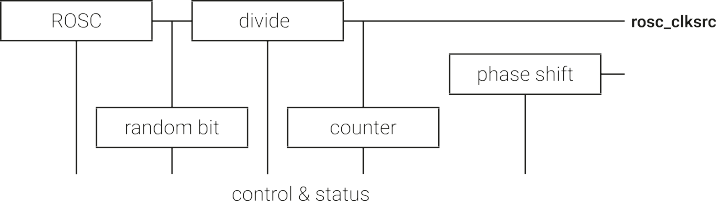
|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - |
| 7:0 | 以xosc频率运行的递减计数器，计数到零并停止。要启动计数器，请写入一个非零值。  在设置时间敏感的硬件时，可用于短暂的软件暂停。 | RW | 0x00 |

* 1. 环形振荡器（ROSC）
     1. 概述

环形振荡器（ROSC）是一种片上振荡器，由一个反相器环构成它不需要外部元件，并在RP 2040上电时自动启动它在引导期间为内核提供时钟ROSC的频率是可编程的，它可以直接为内核提供高速时钟，但频率随工艺、电压和温度（PVT）而变化，因此它不能为需要精确频率的组件提供时钟，如RTC、USB和ADC。第2.15讨论了减小频率变化的方法，但这些方法仅适用于极低功耗设计。对于大多数需要精确时钟频率的应用，建议切换到XOSC和PLL。在启动期间，ROSC以标称6.5MHz运行，并保证在1.8MHz至12MHz范围内。

一旦芯片启动，程序员可以选择从ROSC继续运行并增加其频率或启动晶体振荡器（XOSC）和PLL。在系统时钟切换到XOSC后，可以禁用ROSC每种振荡器都有各自的优势，程序员可以在它们之间切换，以实现应用的最佳解决方案。

*图34. 关于遵守安全和安保标准情况的概览。*



* + 1. ROSC/XOSC权衡

ROSC的优点是其灵活性和低功耗。此外，当使用ROSC提供时钟时，不需要内部或外部元件它的频率是可编程的，因此它可以用来提供快速的内核时钟，而无需启动PLL，并且可以由时钟发生器分频（第2.15）以产生较慢的外设时钟。ROSC立即启动并立即响应频率控制。当进入和退出休眠状态时，它将保留频率设置（参见第2.11.3）。但是，用户必须意识到，由于电源电压和芯片温度的变化，退出休眠状态时频率可能已经漂移。

ROSC的缺点是其频率随PVT（工艺、电压温度）变化，这使得它不适合生成精确的时钟或软件执行时序很重要的应用然而，可以利用PVT频率变化来提供自动频率缩放以最大化性能。这将在2.15节中讨论。

XOSC的唯一优点是其精确的频率，但这在许多应用中是最重要的要求

XOSC的缺点是它需要外部元件（晶体等），功耗较高，启动时间慢（>1ms）和固定的低频。PLL需要产生更高频率的时钟。它们消耗更多功率，并且需要大量时间来启动和改变频率。退出DORMANT模式比ROSC慢得多，因为必须重新启动XOSC，并且必须重新配置PLL。

* + 1. 修改频率

ROSC被布置为8个级，每个级具有可编程驱动器。有两种控制频率的方法。频率范围控制ROSC环路中的级数，FREQA FREQB寄存器控制各级的驱动强度。

通过写入控制ROSC环路级数的FREQ\_RANGE寄存器来改变频率范围。默认LOW范围有8个（0-7级），MEDIUM有6个（2-7级），HIGH有4个（4-7级），TOOHIGH有2个（6-7级）。建议一步一步地更改FREQ\_RANGE，直到达到所需范围。增加频率范围时，ROSC输出不会出现毛刺，因此可以继续使用输出时钟然而，当回到频率范围时，情况并非如此必须为ROSC提供时钟的模块选择备用时钟源，否则必须在转换期间保持复位行为尚未完全表征，但中等范围约为低范围的1.33倍，高范围约为低范围的2TOOHIGH系列的名称恰如其分。不应该使用它，因为ROSC的内部逻辑不会在该频率下运行

FREQA FREQB寄存器控制ROSC环路中各级的驱动强度。增加驱动强度减少了通过级的延迟并增加了振荡频率。每级有3个驱动强度控制位。 每个位打开额外的驱动器，因此每个阶段都有4个驱动强度设置，等于设置的位数，0为默认值，1为双驱动，2为三驱动，3为四驱动。打开额外的驱动器不会对频率产生线性影响，设置第二个比特的影响小于设置第一个比特的影响，依此类推。为确保平滑过渡，建议一次更改一个驱动强度比特当FREQ\_RANGE用于缩短ROSC环路时，被旁路的级仍然传播信号，因此它们的驱动强度必须被设置为至少与环路中的级中的最低驱动强度相同的水平。这不会影响振荡频率。

* + 1. ROSC分频器

ROSC频率太快，无法直接使用，因此在由DIV寄存器控制的整数分频器中分频DIV可以改变，而ROSC正在运行，输出时钟将改变频率没有毛刺。默认除数为16，可确保芯片启动时输出时钟在1.8至12 MHz范围内

分频器有2个输出，rosc\_clksrc和rosc\_clksrc\_ph，第二个是第一个的相移版本这主要用于产品开发期间，如果PHASE寄存器保持默认状态，则输出将相同

* + 1. 随机数生成器

如果系统时钟从XOSC和/或PLL运行，则ROSC可用于生成随机数。只需使能ROSC并读取RANDOMBIT寄存器以获得1位随机数，然后读取n次以获得n位值。这不符合安全系统的随机性要求，因为它可能会受到损害，但在不太关键的应用中可能有用。如果内核从ROSC运行，则该值将不是随机的，因为寄存器读取的定时将与ROSC的相位相关

* + 1. ROSC计数器

延迟寄存器提供了一种管理短软件延迟的方法。将一个值写入到寄存器会自动触发它开始以ROSC频率倒计时到零然后程序员简单地轮询寄存器，直到它达到零。这比在软件循环中使用NOP更可取，因为它独立于核心时钟频率、编译器和编译代码的执行时间

* + 1. 休眠模式

在休眠模式下（参见第2.11.3节），可以暂停所有片内时钟以节省功耗。这在电池供电的应用中特别有用。通过外部事件（如GPIO引脚上的边沿）或片内RTC的中断，将RP 2040从DORMANT模式唤醒。必须在进入休眠模式之前进行配置。如果RTC用于触发唤醒，则必须从外部源对其进行时钟控制要进入休眠模式，编程器必须切换所有内部时钟，从XOSC或ROSC驱动，并停止

PLL。然后，必须将特定的32位值写入所选振荡器（XOSC或ROSC）中的DORMANT寄存器，以停止其振荡。退出休眠模式时，所选振荡器将重新启动。如果选择XOSC，则频率将更精确，但由于启动延迟，重启时间更长（在RP 2040参考设计上>1ms（参见[使用RP 2040的硬件设计](https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf)，[最小设计示例](https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf#minimal-design-example)））。如果选择ROSC，则频率精度较低，但启动时间非常短（约1μs）。

Pico Extras:[https://github.com/raspberrypi/pico-extras/blob/master/src/rp2\_common/hardware\_rosc/rosc.c](https://github.com/raspberrypi/pico-extras/blob/master/src/rp2_common/hardware_rosc/rosc.c#L56-L61)Lines 56-61

1. public voidonDestination（ void）{
2. //返回:这将停止rosc，直到被irq唤醒
3. rosc\_write（rosc\_hw->休眠，ROSC\_DORMANT\_VALUE\_DORMANT）;
4. //等待它在被唤醒后变得稳定
5. 同时（！(rosc\_hw->status&ROSC\_STATUS\_STABLE\_BITS））;

61}

**警告**

如果在进入休眠模式之前没有配置IRQ，ROSC将永远不会重新启动。

参见www.example.com部分2.11.5.2，了解休眠模式的一些示例。

* + 1. 登记册一览表

表264.ROSC寄存器列表

表265.CTRL寄存器

ROSC寄存器从基址0x40060000（在SDK中定义为ROSC\_BASE）开始。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [Ctrl](#_bookmark7) | 环形振荡器控制 |
| 0x04 | [FREQA](#_bookmark8) | 环形振荡器频率控制A |
| 0x08 | [FREQB](#_bookmark9) | 环形振荡器频率控制B |
| 0x0C | [休眠](#_bookmark10) | 环形振荡器暂停控制 |
| 0x10 | [DIV](#_bookmark11) | 控制输出分频器 |
| 0x14 | [相](#_bookmark12) | 控制相移输出 |
| 0x18 | [地位](#_bookmark13) | 环形振荡器状态 |
| 0x1c | [随机比特](#_bookmark14) | 返回1位随机值 |
| 0x20 | [计数](#_bookmark15) | 以ROSC频率运行的递减计数器，计数到零并停止。 |

#### [ROSC](#_bookmark6):CTRL寄存器

**偏移**:0x00

描述

环形振荡器控制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 23:12 | 使 | 上电时，此字段初始化为ENABLE  在将此字段设置为DISABLE之前，必须将系统时钟切换到另一个源，否则芯片将锁定  12位代码旨在提供一些保护，防止意外写入。 无效设置将启用振荡器。  0xd1e→禁用0xfab→启用 | RW | - |
| 11:0 | 频率范围 | 控制ROSC环中的延迟级数低电平  MEDIUM使用阶段2至7  HIGH使用4到7级  TOOHIGH使用第6级至第7级，不应使用，因为其频率超过设计规格。每次向上改变范围时，时钟输出不会出现毛刺  时钟输出将毛刺时，改变范围下降  注意:这里的值是灰色编码的，这就是为什么HIGH在TOOHIGH之前  0xfa4→低0xfa5→中0xfa7→高0xfa6→太高 | RW | 0xaa0 |

表266.FREQA

寄存器

#### [ROSC](#_bookmark6):FREQA注册

**偏移**:0x04

描述

FREQA FREQB寄存器通过控制各级的驱动强度来控制频率。驱动强度有4个级别，由设置的位数决定

增加设置的比特数增加了驱动强度并增加了振荡频率

1. 位设置为默认驱动强度
2. 位设置使驱动强度加倍
3. 位设置三倍驱动强度
4. 位设置四倍驱动强度

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | PASSWD | 设置为0x9696以应用设置  此字段中的任何其他值将所有驱动强度设置为00x9696→通过 | RW | 0x0000 |
| 15 | Reserved. | - | - | - |
| 14:12 | DS3 | 第3阶段驱动强度 | RW | 0x0 |
| 11 | Reserved. | - | - | - |
| 10:8 | DS2 | 第2阶段驱动强度 | RW | 0x0 |
| 7 | Reserved. | - | - | - |
| 6:4 | DS1 | 第1阶段驱动强度 | RW | 0x0 |
| 3 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 2:0 | DS0 | 0级驱动强度 | RW | 0x0 |

表267.FREQB

寄存器

表268.休眠

寄存器

表269.DIV寄存器

#### [ROSC](#_bookmark6):FREQB寄存器

**偏移**:0x08

描述

有关详细说明，请参见freqa寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | PASSWD | 设置为0x9696以应用设置  此字段中的任何其他值将所有驱动强度设置为00x9696→通过 | RW | 0x0000 |
| 15 | Reserved. | - | - | - |
| 14:12 | DS7 | 第7阶段驱动强度 | RW | 0x0 |
| 11 | Reserved. | - | - | - |
| 10:8 | DS6 | 第6级驱动强度 | RW | 0x0 |
| 7 | Reserved. | - | - | - |
| 6:4 | DS5 | 第5阶段驱动强度 | RW | 0x0 |
| 3 | Reserved. | - | - | - |
| 2:0 | DS4 | 第4阶段驱动强度 | RW | 0x0 |

#### [ROSC](#_bookmark6):DORMANT寄存器

**偏移**:0x0c

描述

环形振荡器暂停控制

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 这用于通过暂停ROSC来节省功率。上电时  无效写入也将选择WAKE  警告:在选择休眠模式前设置irq0x636f6d61→休眠  0x77616b65→唤醒 | RW | - |

#### [ROSC](#_bookmark6):DIV寄存器

**偏移**:0x10

描述

控制输出分频器

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - |

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 11:0 | 设置为0xaa0 + div，其中div = 0除以32  div = 1-31除以div任何其他值设置div=31  此寄存器复位为div=16 0xaa0→通过 | RW | - |

表270.第270页相

寄存器

表271.地位

寄存器

#### 相位寄存器

**偏移**:0x14

描述

控制相移输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - | - |
| 11:4 | PASSWD | 设置为0xaa  任何其他值使能shift=0的输出 | RW | 0x00 |
| 3 | 使 | 使能相移输出，这可以在运行中更改 | RW | 0x1 |
| 2 | 翻转 | 反转相移输出，当div=1时忽略 | RW | 0x0 |
| 1:0 | 移位 | 相移相移输出通过时钟输入时钟进行相移，这可以在运行中进行更改  在设置div=1之前，必须将其设置为0 | RW | 0x0 |

#### 状态寄存器

**偏移**:0x18

描述

环形振荡器状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | STABLE | 振荡器运行稳定 | RO | 0x0 |
| 30:25 | Reserved. | - | - | - |
| 24 | BADWRITE | 无效值已写入CTRL\_ENABLE或CTRL\_FREQ\_RANGE或FREQA或FREQB或DIV或PHASE或DORMANT | WC | 0x0 |
| 23:17 | Reserved. | - | - | - |
| 16 | DIV\_RUNNING | 后分频器正在运行  这将复位为0，但在芯片启动期间转换为1 | RO | - |
| 15:13 | Reserved. | - | - | - |
| 12 | ENABLED | 振荡器已启用，但不一定运行且稳定  这将复位为0，但在芯片启动期间转换为1 | RO | - |
| 11:0 | Reserved. | - | - | - |

表272.RANDOMBIT注册

表273.计数

寄存器

#### [ROSC](#_bookmark6):RANDOMBIT寄存器

**偏移**:0x1c

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - |
| 0 | 这只是读取振荡器输出的状态，因此如果环形振荡器停止或以总线频率的谐波运行， | RO | 0x1 |

#### [ROSC](#_bookmark6):ROSC寄存器

**偏移**:0x20

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - |
| 7:0 | 以ROSC频率运行的递减计数器，计数到零并停止。  要启动计数器，请写入一个非零值。  在设置时间敏感的硬件时，可用于短暂的软件暂停。 | RW | 0x00 |

* 1. PLL
     1. 概述

PLL被设计为采用参考时钟，并使用具有反馈环路的VCO（压控振荡器）将其相乘。VCO必须在高频（750到1600 MHz之间）下运行，因此有两个分频器，称为后分频器，可以在VCO频率分配给芯片上的时钟发生器之前对其

RP 2040中有两个PLL它们是:

* SYS\_SYS- 用于生成高达133 MHz的系统时钟
* USB\_USB- 用于生成48 MHz USB参考时钟

*图35.在两个PLL上，FREF（参考）输入连接到晶体振荡器的Xi输入。 PLL包含一个VCO，通过反馈环路（相位频率检测器和环路滤波器）锁定到参考时钟的恒定比率。 这可以合成非常高的频率，其可以由后分频器分频。*



锁定检测

锁

FOUTVCO

FREF

FOUTPOSTDIV

P1 -63 PFD

VCO

电话:0755 - 8888888

REFDIV

6'b

3'b

3'b

旁路

FBDIV12'b

反馈划分

粤ICP备16036660号-1

POSTDIV1POSTDIV2

CLKSSCG

模拟电路

后分频器速率电路参考速率电路

* + 1. 计算PLL参数

要配置PLL，您必须知道参考时钟的频率，在RP 2040上，该频率直接从晶振路由这通常是一个12MHz的晶体，与RP 2040的USB bootrom兼容。PLL的最终输出频率FOUTPOSTDIV可计算为（FREF/REFDIV）×FBDIV/（POSTDIV 1×POSTDIV 2）。考虑到所需的输出频率，必须根据PLL设计的以下约束条件选择PLL参数

* 最小参考频率（FREF/REFDIV）为5 MHz
* 振荡器频率（FOUTVCO）必须在750 MHz→ 1600 MHz范围内
* 反馈分频器（FBDIV）必须在16→320范围内
* 立柱分隔线POSTDIV1和POSTDIV2必须在1→7的范围内
* 最大输入频率（FREF/REFDIV）是VCO频率除以16，这是由于最小反馈除数。此外，必须考虑芯片时钟发生器（连接到FOUTPOSTDIV）的最大频率。对于系统PLL，这是133 MHz，对于USB PLL，是48 MHz。

**注意**

RP2040上的晶体振荡器专为5至15MHz之间的晶体设计，因此通常REFDIV应为1。如果应用电路将更快的基准电压直接驱动到Xi输入，并且需要低VCO频率，则可以增加基准电压除数，以将PLL输入保持在适当的范围内。

**尖端**

当POSTDIV1和POSTDIV2需要两个不同的值时，最好将较高的值分配给POSTDIV1，以降低功耗。

在RP 2040参考设计（参见[使用RP 2040的硬件设计](https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf)，[最小设计示例](https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf#minimal-design-example)）中，将12 MHz晶振连接到晶振，这意味着可实现的最小合法VCO频率为12 MHz × 63 = 756 MHz，最大VCO频率为12 MHz × 133 = 1596 MHz，因此FBDIV必须保持在63→ 133范围例如，将FBDIV设置为100将合成1200 MHz VCO频率。如果POSTDIV1的值为6，POSTDIV2的值为2，则将其除以12，从而在PLL的最终输出端产生干净的100 MHz

* + - 1. 抖动与功耗

通常有几组PLL配置参数可以达到或非常接近所需的输出频率。由编程人员决定是优先考虑低PLL功耗还是降低*抖动*，抖动是PLL输出时钟周期的周期变化就系统稳定性而言，这不是一个问题，因为RP 2040的数字逻辑设计有系统时钟上最坏情况可能抖动的裕度，但音频和视频应用或正在发送和接收数据的应用通常需要高度精确的时钟

根据规范。例如，USB规范定义了允许抖动的最大量

VCO以尽可能高的频率运行，从而可以使用更高的分频后值，从而将抖动降至最低例如，1500 MHz VCO / 6 / 2 = 125 MHz。为了降低功耗，VCO频率应尽可能低。例如:750 MHz VCO / 6 / 1 = 125 MHz。

这里的另一个考虑因素是，稍微调整输出频率可能会通过使输出达到输入的更接近有理倍数来实现更低的VCO频率。实际上，利用任何允许的VCO频率或除数的组合可能无法精确地实现精确的期望频率

SDK提供了一个Python脚本，用于搜索所需输出频率的最佳VCO和后分频器选项

SDK:<https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/scripts/vcocalc.py>

1*号！/ usr/bin/envpython3*

2

3importargparse

4

1. parser = argparse.ArgumentParser（描述=“PLL参数计算器”）
2. parser.add\_argument（“--input”，“-i”，default= 12，help=“输入（参考）频率。默认

12 MHz”，类型=浮点）

1. parser.add\_argument（“--ref-min”，默认值= 5，help=“覆盖最小参考频率。默认5 MHz”，类型=浮点）
2. parser.add\_argument（“--vco-max”，default= 1600，help=“设置VCO的最大频率。默认1600 MHz”，类型=浮点）
3. parser.add\_argument（“--vco-min”，default= 750，help=“设置最小VCO频率。默认750 MHz”，类型=浮点）
4. parser.add\_argument（“--low-frequency”，“-l”，action=“store\_true”，help=“尽可能使用较低的VCO频率。这降低了功耗，但代价是抖动增加”）
5. parser.add\_argument（“output”，help=“输出频率，单位为MHz。“，type=float）
6. args = parser. parse\_args（）13

14*#固定硬件参数*

15 fbdiv\_range = range（16，320+1）

16 postdiv\_range = range（1，7+1）

17 ref\_min =5

18 refdiv\_min =1

19 refdiv\_max =63

20

21 refdiv\_range = range（refdiv\_min，max（refdiv\_min，min（refdiv\_max，int（args.input/args

.ref\_min）+1）

22

23 best =（0，0，0，0，0）

24 best\_margin =参数输出25

1. 对于refdiv\_range中的refdiv
2. 对于fbdiv in（fbdiv\_rangeifargs.low\_order elsereversed（fbdiv\_range））:
3. 输入/refdiv \*fbdiv
4. 如果参数为args. args\_min或参数>args. args\_max:
5. 继续
6. #pd1是内部循环，所以我们更喜欢pd1:pd2的比率更高
7. 对于postdiv\_range中的pd2
8. 对于postdiv\_range中的pd1
9. 输出= 1/pd 1/pd 2
10. margin = abs（out-args.output）
11. ifmarginbest\_margin:
12. best =（out，fbdiv，pd1，pd2，refdiv）
13. best\_margin = margin39
14. print（“请求:{}MHz”.format（args.output））
15. print（“已实现:{}MHz”.format（best[0]））
16. print（“REFDIV:{}”.format（best[4]））
17. print（“FBDIV:{}（VCO ={} MHz）”.格式（best[1]，args.input/best[4]\*best[1]））
18. print（“PD1:{}”.format（best[2]））

45 print（“PD2:{}”.format（best[3]））

给定输入和输出频率，此脚本将找到尽可能接近的最佳PLL参数集。当找到多个同样好的组合时，它返回产生最高VCO频率的参数- l或--low-bandwidth标志将替代地优选较低频率，以降低功耗。

这里要求48MHz输出

$./vcocalc.py48

要求:48.0MHz

实现:48.0MHz

FBDIV:120（VCO = 1440MHz）

PD1:6

PD2:5

如果可能的话，要求48 MHz输出和较低的VCO频率

$./vcocalc.py-l 48请求的:48.0 MHz

实现:48.0MHz

FBDIV:64（VCO = 768MHz）

PD1:4

PD2:4

对于125MHz系统时钟和12MHz输入，最低VCO频率相当高。

$./vcocalc.py-l 125请求:125.0 MHz

实现:125.0MHz

FBDIV:125（VCO = 1500MHz）

PD1:6

PD2:2

我们可以将搜索限制在较低的VCO频率，以便脚本考虑较宽松的频率匹配。请注意，虽然750 MHz VCO在这里是理想的，但我们无法通过将12 MHz输入乘以整数来精确实现750 MHz，这就是为什么前面的调用返回如此高的VCO频率。

$./vcocalc.py-l 125--vco-max800

要求:125.0MHz

实现:126.0MHz

FBDIV:63（VCO = 756MHz）

PD1:6

PD2:1

126 MHz系统时钟可能与期望的125 MHz存在可容忍的偏差，并且生成该时钟在PLL处消耗较少的功率。

* + 1. 配置

SDK使用以下PLL设置:

SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_clocks/include/hardware/clocks.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/include/hardware/clocks.h#L93-L110)第93-110

1. //
2. //RP2040中有两个PLL
3. //1."PLL“产生125 MHz系统时钟，频率定义如下:

`CLK\_KHZ`。

1. //2.“USB PLL”产生48MHz USB时钟，频率由“USB\_CLK\_KHZ”定义。
2. //
3. //两个PLL直接使用晶振输出作为其参考频率输入; PLL参考
4. //频率不能被时钟块中存在的分频器降低晶体频率由“XOSC\_KHZ”定义，或
5. //`XOSC\_MHZ`.
6. //
7. //系统的默认定义对于上述频率是正确的，
8. //crystal frequency.如果需要不同的频率，则必须在
9. //板配置文件以及修改后的PLL设置
10. //使用`vcocalc.py`检查和计算新的PLL设置，如果你改变任何这些频率。
11. //
12. //默认PLL配置:
13. //REF FBDIVVCO POSTDIV
14. //PLL频率:12/1 = 12 MHz\* 125 = 1500 MHz/6/2 =125 MHz
15. //PLL USB:12/1 = 12MHz\* 100 = 1200MHz/5/5 =48MHz

我们将在下面介绍SDK中的init\_init函数，它在尝试配置PLL之前断言所有这些条件都为真。

SDK将PLL控制寄存器定义为结构。 然后将它们映射到PLL的每个实例的内存中。

SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2040/hardware\_structs/include/hardware/structs/pll.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2040/hardware_structs/include/hardware/structs/pll.h#L24-L53)第24-53

24typedef struct{

25

26

27

28

\_REG\_（PLL\_CS\_OFFSET）*//PLL\_CS*

//控制和状态

//0x8000000[31]

//0x00000100[8]

分频的VCO

//0x000003f [5:0]

io\_rw\_32cs;

:PLL（0）:PLL已锁定

:BYPASS（0）:将参考时钟传递到输出，而不是

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

:REFDIV（1）:分频PLL输入参考时钟

\_REG\_（PLL\_PWR\_OFFSET）*//PLL\_PWR*

//控制PLL电源模式

//0x00000020[5]

//0x0000008[3]

//0x0000004[2]

//0x0000001[0]

io\_rw\_32r;

:VCOPD（1）:PLL VCO关断

:POSTDIVPD（1）:PLL分频器关断后

:DSMPD（1）:PLL DSM掉电

:PD（1）:PLL关断

\_REG\_（PLL\_FBDIV\_INT\_OFFSET）*//PLL\_FBDIV\_INT*

//反馈除数

//0x 00000 fff [11:0]: FBDIV\_INT（0）:有关约束，请参阅ctrl注册表描述

io\_rw\_32fbdiv\_int;

\_REG\_（PLL\_PRIM\_OFFSET）*//PLL\_PRIM*

//控制主输出的PLL后置分频器

//0x 00070000 [18:16]: POSTDIV 1（0x 7）:除以1-7

48

49

//0x 00007000 [14:12]:POSTDIV 2（0x 7）:除以1-7

io\_rw\_32;

50}hw\_t;

51

52*#define PLL\_SYS\_HW（（PLL\_HW\_t\*）PLL\_SYS\_BASE）*

53*#define PLL\_USB\_hw（（PLL\_hw\_t\*）PLL\_USB\_BASE）*

SDK定义了用于配置或重新配置PLL的init\_init 它首先清除PLL中先前的任何电源状态，然后计算适当的反馈分频器值。有一些断言可以检查这些值是否满足上面的约束

SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_pll/pll.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_pll/pll.c#L13-L21)第13-21

13void functions\_init（PLL functions\_1，uint refdiv，uint functions\_freq，uint post\_div1，uint post\_div2）{

14

15

16

17

uint32\_tref\_freq = XOSC\_KHZ \* KHZ/refdiv;

//检查freq是否在可接受的范围

assert（PICO\_PLL\_VCO\_MIN\_FREQ\_KHZ \* KHZ）PICO\_PLL\_VCO\_MAX\_FREQ\_KHZ \* KHZ）;

18

1. //我们将参考时钟乘以什么来获得时钟频率
2. *//（这些对象被称为div，因为你将对象的输出对象并将其与引用对象进行比较）*
3. uint32\_tfbdiv = freq/ref\_freq;

PLL的编程顺序如下:

* 编程参考时钟分频器（在RP 2040情况下为1分频
* 设置反馈分频器
* 打开主电源和VCO
* 等待VCO锁定（即保持其输出频率稳定）
* 设置帖子分隔器并打开它们

SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_pll/pll.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_pll/pll.c#L42-L71)第42-71

42

43

44

45

46

47

48

49

50

if（（cs->cs PLL\_CS\_PLL\_BITS）&&

（refdiv ==（->cs PLL\_CS\_REFDIV\_BITS））&&

（fbdiv ==（->fbdiv\_int PLL\_FBDIV\_INT\_BITS））&&

（pdiv

==（PLL-> PLL&（PLL\_PRIM\_POSTDIV1\_BITS |PLL\_PRIM\_POSTDIV2\_BITS）{

//不会中断已经正确配置和运行的PLL

返回;

}

51

52

53

54

55

56

57

58

59

60

61

62

63

uint32\_t\_reset =（\_usb\_hw ==）？RESETS\_PLL\_USB\_BITS:RESETS\_PLL\_PLL\_BITS;

reset\_block（）;

unreset\_block\_wait（）;

//在启动VCO之前加载VCO相关分频器

String->cs =refdiv;

fbdiv\_int =fbdiv;

//打开PLL

uint32\_t电源= PLL\_PWR\_PD\_BITS|/*/主电源*

PLL\_PWR\_VCOPD\_BITS;*//VCO电源*

hw\_clear\_bits（hw-> bits，power）;

64

65

66

67

68

69

70

71

//等待PLL锁定

同时（！(pll->cs PLL\_CS\_PLL\_BITS））tight\_loop\_contents（）;

//设置帖子分隔符

[编辑]

//打开postdivider

hw\_clear\_bits（PLL\_PWR\_POSTDIVPD\_BITS）;

请注意，VCO首先开启，然后是后置分频器，因此PLL在VCO锁定时不会输出脏时钟。

* + 1. 登记册一览表

表274.PLL寄存器列表

表275.CS寄存器

PLL\_CLK和PLL\_USB寄存器分别从基址0x40028000和0x4002c000开始（在SDK中定义为PLL\_CLK\_BASE和PLL\_USB\_BASE）。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x0 | [CS](#_bookmark17) | 控制和状态 |
| 0x4 | [PWR](#_bookmark18) | 控制PLL电源模式。 |
| 0x8 | [FBDIV\_INT](#_bookmark19) | 反馈因子 |
| 0xc | [引物](#_bookmark20) | 控制主输出的PLL后置分频器 |

#### [PLL](#_bookmark16):CS寄存器

**偏移**:0x0

描述

控制和状态一般约束:

参考时钟频率最小=5MHz，最大=800MHz反馈分频器最小=16，最大=320

VCO频率最小值= 750 MHz，最大值= 1600 MHz

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | 锁 | PLL锁定 | RO | 0x0 |
| 30:9 | Reserved. | - | - | - |
| 8 | 旁路 | 将参考时钟传递到输出，而不是分频VCO。VCO继续运行，因此用户可以在参考时钟和分频VCO之间切换，但这样做时输出会出现毛刺。 | RW | 0x0 |
| 7:6 | Reserved. | - | - | - |
| 5:0 | REFDIV | 分频PLL输入参考时钟。对于div=0，行为未定义。  PLL输出在refdiv更改期间不可预测，请在使用之前等待lock=1。 | RW | 0x01 |

#### [PLL](#_bookmark16):PWR寄存器

**偏移**:0x4

表276.PWR寄存器

表277.FBDIV\_INT

寄存器

表278.PRIM寄存器

描述

控制PLL电源模式。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:6 | Reserved. | - | - | - |
| 5 | VCOPD | PLL VCO关断  当PLL输出不需要或旁路=1时，设置为高电平以节省功率。 | RW | 0x1 |
| 4 | Reserved. | - | - | - |
| 3 | POSTDIVPD | PLL分频器关断后  当PLL输出不需要或旁路=1时，设置为高电平以节省功率。 | RW | 0x1 |
| 2 | DSMPD | PLL DSM掉电  把这个设定得低一点是没有用的。 | RW | 0x1 |
| 1 | Reserved. | - | - | - |
| 0 | PD | PLL掉电  当不需要PLL输出时，为了节省功率，将其设置为高电平。 | RW | 0x1 |

#### [PLL](#_bookmark16):FBDIV\_INT寄存器

**偏移**:0x8

描述

反馈因子

(note:此PLL不支持小数分频）

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - |
| 11:0 | 有关约束条件，请参见ckreg描述 | RW | 0x000 |

#### [PLL](#_bookmark16):PRIM寄存器

**偏移**:0xc

描述

控制主输出的PLL后置分频器（注意:此PLL没有辅助输出）

主输出由VCO除以postdiv1\*postdiv2驱动

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:19 | Reserved. | - | - | - |
| 18:16 | POSTDIV1 | 除以1-7 | RW | 0x7 |
| 15 | Reserved. | - | - | - |
| 14:12 | POSTDIV2 | 除以1-7 | RW | 0x7 |
| 11:0 | Reserved. | - | - | - |

* 1. GPIO
     1. 概述

RP 2040具有36个多功能通用输入/输出（GPIO）引脚，分为两组。在典型用例中，QSPI存储体（QSPI\_SS、QSPI\_SCLK和QSPI\_SD 0至QSPI\_SD 3）中的引脚用于执行外部闪存器件中的代码，而用户存储体（GPIO 0至GPIO 29）则供编程器使用。所有GPIO都支持数字输入和输出，但GPIO 26至GPIO 29也可用作芯片的模拟数字转换器（ADC）的输入。每个GPIO可以直接由处理器上运行的软件控制，或者由多个其他功能块控制。

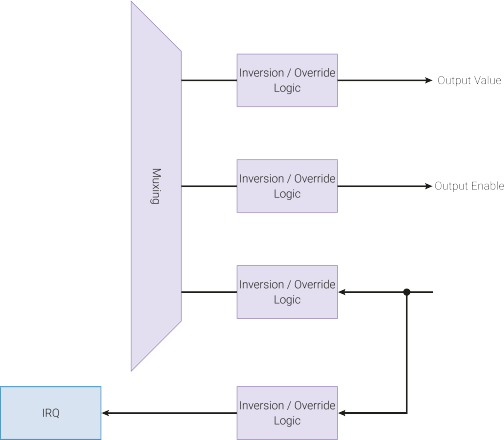
用户GPIO库支持以下功能:

* 通过SIO（单周期IO）进行软件控制-第www.example.com节2.3.1.2，“GPIO控制”
* 可编程IO（PIO）-[第3章，](#_bookmark110)*[PIO](#_bookmark110)*
* 2 × SPI-第4.4节“SPI”
* 2 × UART-[第4.2节，“UART”](#_bookmark190)
* 2 × I2C（双线串行接口）-第4.3节“I2C”
* 8 ×双通道PWM-第4.5节“PWM”
* 2 ×外部时钟输入-第2.15.2.3节，“外部时钟”
* 4 ×通用时钟输出-第2.15节“时钟”
* 4 × ADC输入-第4.9节“ADC和温度传感器”
* USB VBUS管理-[第4.1.2.8节，“VBUS控制”](#_bookmark161)
* 外部中断请求，电平或边沿敏感QSPI库支持以下功能:
* 通过SIO（单周期IO）进行软件控制-第www.example.com节2.3.1.2，“GPIO控制”
* 闪存就地执行（XIP）-第2.6.3节“闪存”

示例IO的逻辑结构如[图36所示](#_bookmark22)。

图36.GPIO的逻辑结构每个GPIO可以由多个外设中的一个或SIO中的软件控制寄存器控制功能选择（FSEL）选择哪个外设输出控制GPIO的方向和输出电平，和/或哪个外设输入可以看到该GPIO的输入电平。 这三个信号（输出电平、输出使能、输入电平）也可以使用GPIO控制寄存器反相，或强制为高电平或低电平。

表279.通用输入/输出（GPIO）用户组功能



* + 1. 功能选择

通过写入GPIO的CTRL寄存器中的FUNCSEL字段来选择分配给每个GPIO的功能。 参见[GPIO0\_CTRL](#_bookmark29)作为示例。表279和表281显示了每个IO的可用功能。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 功能 | | | | | | | | |
| GPIO | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 |
| 0 | SPI0RX | UART0TX | I2C0SDA | PWM0A | SiO | PIO0 | PIO 1 |  | USB OVCURDET |
| 1 | SPI0CSn | UART0RX | I2C0SCL | PWM0B | SiO | PIO0 | PIO 1 |  | USB VBUSDET |
| 2 | SPI0SCK | UART 0CTS | I2C1SDA | PWM1A | SiO | PIO0 | PIO 1 |  | USB VBUSEN |
| 3 | SPI0TX | UART0RTS | I2C1SCL | PWM1B | SiO | PIO0 | PIO 1 |  | USB OVCURDET |
| 4 | SPI0RX | UART1TX | I2C0SDA | PWM2A | SiO | PIO0 | PIO 1 |  | USB VBUSDET |
| 5 | SPI0CSn | UART1RX | I2C0SCL | PWM2B | SiO | PIO0 | PIO 1 |  | USB VBUSEN |
| 6 | SPI0SCK | UART 1CTS | I2C1SDA | PWM3A | SiO | PIO0 | PIO 1 |  | USB OVCURDET |
| 7 | SPI0TX | UART 1RTS | I2C1SCL | PWM3B | SiO | PIO0 | PIO 1 |  | USB VBUSDET |
| 8 | SPI1RX | UART1TX | I2C0SDA | PWM4A | SiO | PIO0 | PIO 1 |  | USB VBUSEN |
| 9 | SPI1CSn | UART1RX | I2C0SCL | PWM4B | SiO | PIO0 | PIO 1 |  | USB OVCURDET |
| 10 | SPI1SCK | UART 1CTS | I2C1SDA | PWM 5A | SiO | PIO0 | PIO 1 |  | USB VBUSDET |
| 11 | SPI1TX | UART 1RTS | I2C1SCL | PWM5B | SiO | PIO0 | PIO 1 |  | USB VBUSEN |
| 12 | SPI1RX | UART0TX | I2C0SDA | PWM6A | SiO | PIO0 | PIO 1 |  | USB OVCURDET |
| 13 | SPI1CSn | UART0RX | I2C0SCL | PWM6B | SiO | PIO0 | PIO 1 |  | USB VBUSDET |
| 14 | SPI1SCK | UART 0CTS | I2C1SDA | PWM7A | SiO | PIO0 | PIO 1 |  | USB VBUSEN |
| 15 | SPI1TX | UART0RTS | I2C1SCL | PWM7B | SiO | PIO0 | PIO 1 |  | USB OVCURDET |
| 16 | SPI0RX | UART0TX | I2C0SDA | PWM0A | SiO | PIO0 | PIO 1 |  | USB VBUSDET |
| 17 | SPI0CSn | UART0RX | I2C0SCL | PWM0B | SiO | PIO0 | PIO 1 |  | USB VBUSEN |
| 18 | SPI0SCK | UART 0CTS | I2C1SDA | PWM1A | SiO | PIO0 | PIO 1 |  | USB OVCURDET |
| 19 | SPI0TX | UART0RTS | I2C1SCL | PWM1B | SiO | PIO0 | PIO 1 |  | USB VBUSDET |
| 20 | SPI0RX | UART1TX | I2C0SDA | PWM2A | SiO | PIO0 | PIO 1 | 时钟GPIN0 | USB VBUSEN |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 功能 | | | | | | | | |
| 21 | SPI0CSn | UART1RX | I2C0SCL | PWM2B | SiO | PIO0 | PIO 1 | 时钟GPOUT0 | USB OVCURDET |
| 22 | SPI0SCK | UART 1CTS | I2C1SDA | PWM3A | SiO | PIO0 | PIO 1 | 时钟GPIN1 | USB VBUSDET |
| 23 | SPI0TX | UART 1RTS | I2C1SCL | PWM3B | SiO | PIO0 | PIO 1 | 时钟GPOUT1 | USB VBUSEN |
| 24 | SPI1RX | UART1TX | I2C0SDA | PWM4A | SiO | PIO0 | PIO 1 | 时钟GPOUT2 | USB OVCURDET |
| 25 | SPI1CSn | UART1RX | I2C0SCL | PWM4B | SiO | PIO0 | PIO 1 | 时钟GPOUT3 | USB VBUSDET |
| 26 | SPI1SCK | UART 1CTS | I2C1SDA | PWM 5A | SiO | PIO0 | PIO 1 |  | USB VBUSEN |
| 27 | SPI1TX | UART 1RTS | I2C1SCL | PWM5B | SiO | PIO0 | PIO 1 |  | USB OVCURDET |
| 28 | SPI1RX | UART0TX | I2C0SDA | PWM6A | SiO | PIO0 | PIO 1 |  | USB VBUSDET |
| 29 | SPI1CSn | UART0RX | I2C0SCL | PWM6B | SiO | PIO0 | PIO 1 |  | USB VBUSEN |

表280.GPIO用户库功能说明

*表281. 通用输入/输出（GPIO）QSPI库*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 功能 | | | | | | | | | |
| IO | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 |
| QSPISCK | 公司简介 |  |  |  |  | SiO |  |  |  |  |
| QSPICSn | XIPCSn |  |  |  |  | SiO |  |  |  |  |
| QSPISD0 | XIPSD0 |  |  |  |  | SiO |  |  |  |  |

功能

每个GPIO一次可以选择一个功能。同样，一次只能在一个*GPIO*上选择每个外设输入（例如UART0 RX）。如果同一外设输入连接到多个GPIO，则外设会看到这些GPIO输入的逻辑OR。

|  |  |
| --- | --- |
| 函数名 | 描述 |
| Spix | 将一个内部PL 022 SPI外设连接到GPIO |
| UARTx | 将一个内部PL 011 UART外设连接到GPIO |
| I2Cx | 将一个内部DW I2C外设连接到GPIO |
| PWMxA/B | 将PWM片连接到GPIO。有八个PWM片，每个片有两个输出通道（A/B）。 B引脚也可用作输入，用于频率和占空比测量。 |
| SiO | GPIO的软件控制，来自单周期IO（SIO）模块。处理器必须选择SIO功能（F5）才能*驱动*GPIO，但输入端始终处于连接状态，因此软件可以随时检查GPIO的状态。 |
| PIOx | 将其中一个可编程IO模块（PIO）连接到GPIO。PIO可以实现各种接口，并具有自己的内部引脚映射硬件，允许在用户组GPIO上灵活放置数字接口PIO必须选择PIO功能（F6、F7）才能*驱动*GPIO，但输入始终处于连接状态，因此PIO始终可以看到所有引脚的状态。 |
| 时钟GPINx | 通用时钟输入。可以路由到RP 2040上的多个内部时钟域，例如为RTC提供1Hz时钟，也可以连接到内部频率计数器。 |
| 时钟GPOUTx | 通用时钟输出。可以驱动GPIO上的内部时钟数，可选整数分频。 |
| USB OVCUR DET/VBUS DET/VBUS EN | 与内部USB控制器之间的USB电源控制信号 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 功能 | | | | | | | | | |
| QSPISD1 | XIPSD1 |  |  |  |  | SiO |  |  |  |  |
| QSPISD2 | XIPSD2 |  |  |  |  | SiO |  |  |  |  |
| QSPISD3 | XIPSD3 |  |  |  |  | SiO |  |  |  |  |

表282.GPIOQSPI

银行功能描述

|  |  |
| --- | --- |
| 函数名 | 描述 |
| XIP | 连接到闪存就地执行（XIP）子系统内部的同步串行接口（SSI）这允许处理器直接从外部SPI、双SPI或四SPI闪存执行代码 |
| SiO | GPIO的软件控制，来自单周期IO（SIO）模块。处理器必须选择SIO功能（F5）才能*驱动*GPIO，但输入端始终处于连接状态，因此软件可以随时检查GPIO的状态QSPI IO通过SIO\_GPIO\_HI\_x寄存器控制，并从LSB开始按SCK、CSn、SD 0、SD 1、SD 2、SD 3的顺序映射到寄存器位 |

XIP外设通常使用六个QSPI Bank GPIO引脚与外部闪存器件通信。但是，在两种情况下，引脚可用作软件控制的GPIO:

* 如果使用SPI或双SPI闪存设备进行就地执行，则SD 2和SD 3引脚不用于闪存访问，而是可用于电路板上的其他GPIO功能
* 如果RP 2040用于无闪存配置（仅USB引导），则所有六个引脚均可用于软件控制的GPIO功能
  + 1. 中断

在四种情况下，每个GPIO引脚都可以产生

* 高电平:GPIO引脚为逻辑1
* 低电平:GPIO引脚为逻辑0
* 边沿高电平:GPIO已从逻辑0转换为逻辑1
* 边沿低电平:GPIO已从逻辑1转换为逻辑0

电平中断未锁存。这意味着，如果引脚为逻辑1且高电平中断处于活动状态，则引脚变为逻辑0后，高电平中断将立即变为非活动边沿中断存储在INTR寄存器中，可通过写入INTR寄存器来清除

有三个中断目的地的使能、状态和强制寄存器:proc0、proc1和dormant\_wake。对于proc0，寄存器为使能（[E0\_INTE0](#_bookmark34)）、状态（[E00\_INTS 0](#_bookmark42)）和强制（[E00\_INTF 0](#_bookmark38)）。休眠唤醒用于将ROSC或XOSC从休眠模式唤醒有关休眠模式的更多信息，请参见2.11.5.2部分

所有中断都按每个存储体和每个目的地进行“或”运算，总共产生六个GPIO中断:

* IO bank 0到休眠唤醒
* IO bank 0到proc0
* IO bank 0到proc1
* IO QSPI到休眠唤醒
* IO QSPI到进程0
* IO QSPI到进程1

这意味着用户可以同时观察多个GPIO事件。

* + 1. 垫

每个GPIO通过一个“焊盘”连接到片外世界焊盘是芯片内部逻辑和外部电路之间的电气接口它们转换信号电压电平，支持更高的电流，并提供一定程度的静电放电（ESD）保护。焊盘的电气性能可以调整，以满足外部电路的要求可进行以下调整

* 输出驱动强度可设置为2mA、4mA、8mA或12mA
* 输出压摆率可设置为慢或快
* 可以使能输入迟滞（施密特触发器模式）
* 可以使能上拉或下拉，以便在输出驱动器禁用时设置输出信号电平
* 当焊盘未使用、未连接或连接到模拟信号时，可以禁用输入缓冲器，以减少电流消耗

图37中示出了示例垫。

*图37.单个IO焊盘的示意图。*



转换速率

GPIO

牧兴

输出使能输出数据

驱动强度

垫

2

输入使能输入数据施密特触发器

上拉/下拉

2

焊盘的输出使能、输出数据和输入数据端口通过IO多路复用器连接到控制焊盘的功能所有其他端口由焊盘控制寄存器控制该寄存器还允许通过覆盖来自控制焊盘的功能的输出使能信号来禁用焊盘的输出驱动器有关焊盘控制寄存器的示例，请参见[GPIO 0](#_bookmark86)

焊盘上的输出信号电平和可接受的输入信号电平均由数字IO电源（IOVDD）决定。IOVDD可以是1.8V至3.3V之间的任何标称电压，但要满足1.8V供电时的规格，必须通过向焊盘VOLTAGE\_SELECT寄存器写入1来调整焊盘输入阈值。默认情况下，焊盘输入阈值对2.5V至3.3V之间的IOVDD电压有效使用1.8V电压和默认输入阈值是一种安全的工作模式，尽管这会导致输入阈值不符合规格。

**警告**

使用大于1.8V的IOVDD电压，输入阈值设置为1.8V，可能会导致芯片损坏

焊盘输入阈值按存储体进行调整，与用户IO存储体（IO存储体0）和QSPI IO存储体相关的焊盘使用单独的VOLTAGE\_SELECT寄存器但是，两个存储体共享相同的数字IO电源（IOVDD），因此两个寄存器应始终设置为相同的值。

有关Pad寄存器的详细信息，请参见[第2.19.6.3节“Pad控制-用户库”](#_bookmark83)和[第2.19.6.4节“Pad控制-QSPI库”](#_bookmark89)。

* + 1. 软件示例
       1. 选择IO功能

IO引脚可以执行许多不同的功能，必须在使用前进行配置例如，您可能希望它是一个RX\_TX引脚或PWM输出。SDK为此提供了gpio\_set\_function。许多SDK示例会在开始时调用gpio\_set\_function，以便它可以打印到打印机。

SDK首先定义一个结构来表示IO bank 0（用户IO bank）的寄存器。每个IO都有一个状态寄存器，后面是一个控制寄存器。有30个IO，所以包含状态和控制寄存器的结构被实例化为io[30]，以重复30次。

*SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2040/hardware\_structs/include/hardware/structs/iobank0.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2040/hardware_structs/include/hardware/structs/iobank0.h#L163-L211)第163-211*

202

203

204

205

206

207

208

io\_rw\_32 intr[4];io\_irq\_candidate\_hw\_t proc0\_irq\_candidate; io\_irq\_candidate\_hw\_tproc1\_irq\_candidate;

io\_irq\_cycle\_hw\_t休眠唤醒irq\_cycle;

209}iobank 0\_hw\_t;

210

211*#define iobank0\_hw（（iobank0\_hw\_t\*）IO\_BANK0\_BASE）*

|  |  |  |  |
| --- | --- | --- | --- |
| 163 | 类型定义结构{ | | |
| 164 | iobank0\_status\_ctrl\_hw\_t io[NUM\_BANK0\_GPIOS];*//30* | | |
| 165 |  | | |
| 166 | \_REG\_（IO\_BANK0\_INTR0\_OFFSET）*//IO\_BANK0\_INTR0* | | |
| 167 | //（从数组索引0寄存器IO\_BANK0\_INTR0复制的描述类似地适用于 | | |
| 168 | 其他数组索引）  // | |  |
| 169 | // | 原始中断 |  |
| 170 | // | 0x8000000[31] : | GPIO7\_EDGE\_HIGH（0） |
| 171 | // | 0x40000000[30] : | GPIO7\_EDGE\_LOW（0） |
| 172 | // | 0x20000000[29] : | GPIO7\_LEVEL\_HIGH（0） |
| 173 | // | 0x10000000[28] : | GPIO7\_LEVEL\_LOW（0） |
| 174 | // | 0x08000000[27] : | GPIO6\_EDGE\_HIGH（0） |
| 175 | // | 0x04000000[26] : | GPIO6\_EDGE\_LOW（0） |
| 176 | // | 0x02000000[25] : | GPIO6\_LEVEL\_HIGH（0） |
| 177 | // | 0x01000000[24] : | GPIO6\_LEVEL\_LOW（0） |
| 178 | // | 0x00800000[23] : | GPIO5\_EDGE\_HIGH（0） |
| 179 | // | 0x00400000[22] : | GPIO5\_EDGE\_LOW（0） |
| 180 | // | 0x00200000[21] : | GPIO5\_LEVEL\_HIGH（0） |
| 181 | // | 0x00100000[20] : | GPIO5\_LEVEL\_LOW（0） |
| 182 | // | 0x00080000[19] : | GPIO4\_EDGE\_HIGH（0） |
| 183 | // | 0x00040000[18] : | GPIO4\_EDGE\_LOW（0） |
| 184 | // | 0x00020000[17] : | GPIO4\_LEVEL\_HIGH（0） |
| 185 | // | 0x00010000[16] : | GPIO4\_LEVEL\_LOW（0） |
| 186 | // | 0x00008000[15] : | GPIO3\_EDGE\_HIGH（0） |
| 187 | // | 0x00004000[14] : | GPIO3\_EDGE\_LOW（0） |
| 188 | // | 0x00002000[13] : | GPIO3\_LEVEL\_HIGH（0） |
| 189 | // | 0x00001000[12] : | GPIO3\_LEVEL\_LOW（0） |
| 190 | // | 0x00000800[11] : | GPIO2\_EDGE\_HIGH（0） |
| 191 | // | 0x00000400[10] : | GPIO2\_EDGE\_LOW（0） |
| 192 | // | 0x00000200[9] : | GPIO2\_LEVEL\_HIGH（0） |
| 193 | // | 0x00000100[8] : | GPIO2\_LEVEL\_LOW（0） |
| 194 | // | 0x0000080[7] : | GPIO1\_EDGE\_HIGH（0） |
| 195 | // | 0x0000040[6] : | GPIO1\_EDGE\_LOW（0） |
| 196 | // | 0x0000020[5] : | GPIO1\_LEVEL\_HIGH（0） |
| 197 | // | 0x0000010[4] : | GPIO1\_LEVEL\_LOW（0） |
| 198 | // | 0x0000008[3] : | GPIO0\_EDGE\_HIGH（0） |
| 199 | // | 0x0000004[2] : | GPIO0\_EDGE\_LOW（0） |
| 200 | // | 0x0000002[1] : | GPIO0\_LEVEL\_HIGH（0） |
| 201 | // | 0x00000001[0] : | GPIO0\_LEVEL\_LOW（0） |

为IO存储体1的焊盘控制寄存器定义了类似的结构默认情况下，所有焊盘从复位状态出来即可使用，其输入使能，输出禁用设置为0。无论如何，SDK中的gpio\_set\_function都会设置这些参数，以确保pad已准备好供所选函数使用。最后，所需的功能选择被写入IO控制寄存器（有关IO控制寄存器的示例，请参阅[GPIO0\_CTRL](#_bookmark29)

SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_gpio/gpio.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_gpio/gpio.c#L30-L43)第30-43

1. //选择此GPIO的功能，并确保在焊盘处启用输入/输出。
2. //这也会清除输入/输出/irq覆盖位。
3. publicint getName（int getName int getName，int getName）{
4. param（param）;
5. invalid\_params\_if（GPIO，（（uint32\_t）fn IO\_BANK0\_GPIO0\_CTRL\_FUNCSEL\_LSB）&

~IO\_BANK0\_GPIO0\_CTRL\_FUNCSEL\_BITS）;

1. //设置输入启用，输出禁用关闭
2. hw\_write\_masked（padsbank0\_hw->io[gpio]，
3. PADS\_BANK0\_GPIO0\_IE\_BITS，
4. PADS\_BANK0\_GPIO0\_IE\_BITS|PADS\_BANK0\_GPIO0\_OD\_BITS

39）;

1. //将fsel以外的所有字段置零;我们希望这个IO按照外设告诉它的去做。
2. //这不会影响例如pullup/pulldown，就像这些在pad控件中一样。
3. iobank0\_hw->io[gpio]. cbss = fnIO\_BANK0\_GPIO0\_CTRL\_FUNCSEL\_LSB;

43}

* + - 1. 启用GPIO中断

SDK提供了GPIO引脚状态改变时中断的方法

SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_gpio/gpio.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_gpio/gpio.c#L176-L182)第176-182

1. voidgpio\_set\_irq\_enabled（uint gpio， uint32\_tevents，bool enabled）{
2. //每个核心单独的mask/force/status，所以检查哪个核心调用了，
3. //设置相关的IRQ控件。
4. io\_irq\_codeword\_hw\_t\*irq\_codeword\_base = get\_core\_num（）？
5. &iobank0\_hw-> proc1\_irq\_cycle:iobank0\_hw-

> proc0\_irq\_center;

1. \_gpio\_set\_irq\_enabled（gpio，events，enabled，irq\_callback\_base）;

182}

gpio\_set\_irq\_enabled使用较低级别的函数\_gpio\_set\_irq\_enabled:

*SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_gpio/gpio.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_gpio/gpio.c#L163-L174)第163-174*

163static void\_gpio\_set\_irq\_enabled（uint gpio，uint32\_tevents，bool enabled，io\_irq\_callback\_hw\_t

\*irq\_codeword\_base）{

164

165

166

167

168

169

170

//清除可能导致立即虚假处理程序条目的过时事件

gpio\_acknowledge\_irq（gpio，events）;

io\_rw\_32 \*en\_reg = irq\_codeword\_base->inte[gpio/8]; events =4\*（gpio %8）;

if（enabled）

|  |  |  |
| --- | --- | --- |
| 171 |  | hw\_set\_bits（en\_reg，events）; |
| 172 |  | 其他 |
| 173 |  | hw\_clear\_bits（en\_reg，events）; |
| 174 | } |  |

用户提供一个指针，指向当GPIO事件发生时调用的回调函数的示例应用

使用这个系统的是hello\_gpio\_irq:

Pico示例:<https://github.com/raspberrypi/pico-examples/blob/master/gpio/hello_gpio_irq/hello_gpio_irq.c>

1*/\*\**

* + - * 1. \*版权所有（c）2020 Raspberry Pi（Trading）Ltd.
        2. \*
        3. \* SPDX-License-Identifier:BSD-3-Clause
        4. \*/

6

7*#include<stdio.h>*

8*#包含“pico/stdlib.h”*

9*#include“hardware/gpio.h”*

10

11static char event\_str[128];12

13void gpio\_event\_string（char\*buf，uint32\_tevents）;14

1. public int findDuplicate（int findDuplicate，int findDuplicate）{
2. //将刚刚发生的GPIO事件放入event\_str
3. //这样我们就可以打印它了
4. gpio\_event\_string（event\_str，events）;
5. printf（“GPIO %d %s\n”，gpio，event\_str）;

20}

21

1. public voidrun（）{
2. stdio\_init\_all（）;

24

1. printf（“Hello GPIOIRQ”）;
2. gpio\_set\_irq\_enabled\_with\_callback（2，GPIO\_IRQ\_EDGE\_RISE |GPIO\_IRQ\_EDGE\_FALL，true，&gpio\_callback）;

27

1. //等待永远
2. while（1）;

30}

31

32

1. static const char\*gpio\_irq\_str[]={
2. “LEVEL\_LOW”，*//0x1*
3. “LEVEL\_HIGH”，*//0x2*
4. “EDGE\_FALL”，*//0x4*
5. “EDGE\_RISE” *//0x8*

38};

39

1. public intfindDuplicate（ char\*buf）{
2. public int findDuplicate（int i =0; i=0; i = 0）{
3. int count =（1）;
4. if（events mask）{
5. //将此事件字符串复制到用户字符串中
6. const char\*event\_str =gpio\_irq\_str[i];
7. while（\*event\_str！= “\0”）{
8. \*buf++=\*event\_str++;

48 }

49个事件=~掩码;50

51 *//如果有更多事件添加“，“*

52 if（events）{

53\*buf++='，';

54\*buf++=“”;

55 }

56 }

57 }

58\*buf++='\0';

59}

* + 1. 登记册一览表
       1. IO-用户银行

用户银行IO寄存器从基址0x40014000（在SDK中定义为IO\_BANK0\_BASE）开始。

*表283. IO\_BANK0寄存器列表*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x000 | [GPIO0\_STATUS](#_bookmark28) | GPIO状态 |
| 0x004 | [GPIO0\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x008 | [GPIO 1\_STATUS](#_bookmark28) | GPIO状态 |
| 0x 00c | [GPIO1\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x010 | [GPIO2\_STATUS](#_bookmark28) | GPIO状态 |
| 0x014 | [GPIO2\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x018 | [GPIO3\_STATUS](#_bookmark28) | GPIO状态 |
| 0x01c | [GPIO3\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x020 | [GPIO 4\_STATUS](#_bookmark28) | GPIO状态 |
| 0x024 | [GPIO4\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x028 | [GPIO5\_STATUS](#_bookmark28) | GPIO状态 |
| 0x02c | [GPIO5\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x030 | [GPIO6\_STATUS](#_bookmark28) | GPIO状态 |
| 0x034 | [GPIO6\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x038 | [GPIO7\_STATUS](#_bookmark28) | GPIO状态 |
| 0x03c | [GPIO7\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x040 | [GPIO8\_STATUS](#_bookmark28) | GPIO状态 |
| 0x044 | [GPIO8\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x048 | [GPIO9\_STATUS](#_bookmark28) | GPIO状态 |
| 0x04c | [GPIO9\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x 050 | [GPIO10\_STATUS](#_bookmark28) | GPIO状态 |
| 0x054 | [GPIO10\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x058 | [GPIO11\_STATUS](#_bookmark28) | GPIO状态 |
| 0x05c | [GPIO11\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x060 | [GPIO12\_STATUS](#_bookmark28) | GPIO状态 |
| 0x064 | [GPIO12\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x068 | [GPIO 13状态](#_bookmark28) | GPIO状态 |
| 0x06c | [GPIO13\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x070 | [GPIO 14状态](#_bookmark28) | GPIO状态 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x074 | [GPIO14\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x078 | [GPIO 15状态](#_bookmark28) | GPIO状态 |
| 0x07c | [GPIO15\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x080 | [GPIO16\_STATUS](#_bookmark28) | GPIO状态 |
| 0x084 | [GPIO16\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x088 | [GPIO17\_STATUS](#_bookmark28) | GPIO状态 |
| 0x08c | [GPIO17\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x090 | [GPIO 18\_STATUS](#_bookmark28) | GPIO状态 |
| 0x094 | [GPIO18\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x098 | [GPIO19\_STATUS](#_bookmark28) | GPIO状态 |
| 0x09c | [GPIO19\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x0a0 | [GPIO20\_STATUS](#_bookmark28) | GPIO状态 |
| 0x0a4 | [GPIO20\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x0a8 | [GPIO21\_STATUS](#_bookmark28) | GPIO状态 |
| 0x0ac | [GPIO21\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x0b0 | [GPIO22\_STATUS](#_bookmark28) | GPIO状态 |
| 0x0b4 | [GPIO22\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x0b8 | [GPIO23状态](#_bookmark28) | GPIO状态 |
| 0x0bc | [GPIO23\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x0c0 | [GPIO24\_STATUS](#_bookmark28) | GPIO状态 |
| 0x0c4 | [GPIO24\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x0c8 | [GPIO25\_STATUS](#_bookmark28) | GPIO状态 |
| 0x0cc | [GPIO25\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x0d0 | [GPIO26\_STATUS](#_bookmark28) | GPIO状态 |
| 0x0d4 | [GPIO26\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x0d8 | [GPIO27\_STATUS](#_bookmark28) | GPIO状态 |
| 0x0dc | [GPIO27\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x0e0 | [GPIO28\_STATUS](#_bookmark28) | GPIO状态 |
| 0x0e4 | [GPIO28\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x0e8 | [GPIO29状态](#_bookmark28) | GPIO状态 |
| 0x0ec | [GPIO29\_CTRL](#_bookmark29) | GPIO控制，包括功能选择和覆盖。 |
| 0x0f0 | [INTR0](#_bookmark30) | 原始中断 |
| 0x0f4 | [INTR1](#_bookmark31) | 原始中断 |
| 0x0f8 | [INTR2](#_bookmark32) | 原始中断 |
| 0x0fc | [INTR 3](#_bookmark33) | 原始中断 |
| 0x100 | [0\_INTE0](#_bookmark34) | 启用proc0 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x104 | [联系我们](#_bookmark35) | 启用proc0 |
| 0x108 | [联系我们](#_bookmark36) | 启用proc0 |
| 0x10c | [网站地图](#_bookmark37) | 启用proc0 |
| 0x110 | [PROC0\_INTF0](#_bookmark38) | Proc0的力 |
| 0x114 | [0\_INTF1](#_bookmark39) | Proc0的力 |
| 0x118 | [0\_INTF2](#_bookmark40) | Proc0的力 |
| 0x11c | [0\_INTF3](#_bookmark41) | Proc0的力 |
| 0x120 | [0\_INTS0](#_bookmark42) | proc0强制屏蔽后的状态 |
| 0x124 | [0\_INTS1](#_bookmark43) | proc0强制屏蔽后的状态 |
| 0x128 | [PROC0\_INTS2](#_bookmark44) | proc0强制屏蔽后的状态 |
| 0x12c | [PROC0\_INTS3](#_bookmark45) | proc0强制屏蔽后的状态 |
| 0x130 | PROC1\_INTE0 | 启用proc1 |
| 0x134 | PROC1\_INTE1 | 启用proc1 |
| 0x138 | PROC1\_INTE2 | 启用proc1 |
| 0x13c | PROC1\_INTE3 | 启用proc1 |
| 0x140 | [1\_INTF 0](#_bookmark50) | Proc1的力 |
| 0x144 | [PROC1\_INTF 1](#_bookmark51) | Proc1的力 |
| 0x148 | [PROC1\_INTF 2](#_bookmark52) | Proc1的力 |
| 0x14c | [PROC1\_INTF 3](#_bookmark53) | Proc1的力 |
| 0x150 | [INTS 1\_0](#_bookmark54) | 对proc 1强制屏蔽后中断状态 |
| 0x154 | [INTS 1](#_bookmark55) | 对proc 1强制屏蔽后中断状态 |
| 0x158 | [PROC1\_INTS2](#_bookmark56) | proc1强制屏蔽后的状态 |
| 0x15c | [PROC1\_INTS3](#_bookmark57) | proc1强制屏蔽后的状态 |
| 0x160 | [DORMANT\_WAKE\_INTE0](#_bookmark58) | DORMANT\_唤醒中断启用 |
| 0x164 | [DORMANT\_WAKE\_INTE1](#_bookmark59) | 启用 DORMANT\_唤醒 |
| 0x168 | [DORMANT\_WAKE\_INTE2](#_bookmark60) | 启用 DORMANT\_唤醒 |
| 0x16c | [DORMANT\_WAKE\_INTE3](#_bookmark61) | 启用 DORMANT\_唤醒 |
| 0x170 | [DORMANT\_WAKE\_INTF 0](#_bookmark62) | DORMANT\_唤醒的中断力 |
| 0x174 | [DORMANT\_WAKE\_INTF 1](#_bookmark63) | DORMANT\_唤醒的中断力 |
| 0x178 | [DORMANT\_WAKE\_INTF 2](#_bookmark64) | DORMANT\_唤醒的中断力 |
| 0x17c | [DORMANT\_WAKE\_INTF3](#_bookmark65) | DORMANT\_唤醒的中断力 |
| 0x180 | [DORMANT\_WAKE\_INTS0](#_bookmark66) | DORMANT\_唤醒强制屏蔽后的休眠状态 |
| 0x184 | [DORMANT\_WAKE\_INTS1](#_bookmark67) | DORMANT\_唤醒强制屏蔽后的休眠状态 |
| 0x188 | [DORMANT\_WAKE\_INTS2](#_bookmark68) | DORMANT\_唤醒强制屏蔽后的休眠状态 |
| 0x18c | [DORMANT\_WAKE\_INTS3](#_bookmark69) | DORMANT\_唤醒强制屏蔽后的休眠状态 |

表284.GPIO0\_STATUS，GPIO1\_STATUS，.，GPIO28\_STATUS，GPIO29\_STATUS

寄存器

表285.GPIO0\_CTRL，GPIO1\_CTRL，.，GPIO28\_CTRL，GPIO29\_CTRL

寄存器

#### [IO\_BANK0](#_bookmark27):GPIO0\_STATUS，GPIO1\_STATUS， ...， GPIO28\_STATUS，

GPIO 29\_STATUS寄存器

**偏移量**:0x000、0x008、...、0x0e0、0x0e8

描述

GPIO状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:27 | Reserved. | - | - | - |
| 26 | IRQTOPROC | 在应用覆盖后，中断到处理器 | RO | 0x0 |
| 25 | Reserved. | - | - | - |
| 24 | IRQFROMPAD | 在应用覆盖之前从焊盘中断 | RO | 0x0 |
| 23:20 | Reserved. | - | - | - |
| 19 | INTOPERI | 应用覆盖后，将信号输入到外围设备 | RO | 0x0 |
| 18 | Reserved. | - | - | - |
| 17 | INFROMPAD | 来自焊盘的输入信号，在应用超控 | RO | 0x0 |
| 16:14 | Reserved. | - | - | - |
| 13 | OETOPAD | 在寄存器覆盖后，输出使能到焊盘 | RO | 0x0 |
| 12 | OEFROMPERI | 在应用寄存器覆盖之前，从所选外围设备启用输出 | RO | 0x0 |
| 11:10 | Reserved. | - | - | - |
| 9 | OUTTOPAD | 寄存器覆盖后输出信号到焊盘 | RO | 0x0 |
| 8 | OUTFROMPERI | 在应用寄存器覆盖之前，来自所选外设的输出信号 | RO | 0x0 |
| 7:0 | Reserved. | - | - | - |

#### [IO\_BANK0](#_bookmark27):GPIO0\_CTRL，GPIO1\_CTRL， .， GPIO28\_CTRL，GPIO29\_CTRL

寄存器

**偏移**:0x004、0x00c、...、0x0e4、0x0ec

描述

GPIO控制，包括功能选择和覆盖。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:30 | Reserved. | - | - | - |
| 29:28 | IRQover | 0x0 →不反转中断0x1→反转中断  0x2→驱动中断低电平0x3→驱动中断高电平 | RW | 0x0 |
| 27:18 | Reserved. | - | - | - |
| 17:16 | INOVER | 0x0→不反相PWM输入0x1→反相PWM输入  0x2→驱动整流器输入低0x3→驱动整流器输入HIGH | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 15:14 | Reserved. | - | - | - |
| 13:12 | OEOVER | 0x0→从funcsel选择的外围信号驱动输出使能  0x1→通过funcsel选择的外设信号的反相使能驱动输出  0x2→禁用输出0x3→启用输出 | RW | 0x0 |
| 11:10 | Reserved. | - | - | - |
| 9:8 | Outover | 0x0→从funcsel选择的外围信号驱动输出  0x1→从funcel选择的外围信号的反相驱动输出  0x2→驱动输出低0x3→驱动输出高 | RW | 0x0 |
| 7:5 | Reserved. | - | - | - |
| 4:0 | FUNCSEL | 功能选择。31 == NULL。有关可用功能，请参见GPIO功能表 | RW | 0x1f |

表286.INTR0

寄存器

#### [IO\_BANK0](#_bookmark27):INTR0寄存器

**偏移**:0x0f0

描述

原始中断

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 7\_EDGE\_HIGH |  | WC | 0x0 |
| 30 | GPIO7\_EDGE\_LOW |  | WC | 0x0 |
| 29 | GPIO 7\_LEVEL\_HIGH |  | RO | 0x0 |
| 28 | GPIO7\_LEVEL\_LOW |  | RO | 0x0 |
| 27 | GPIO 6\_EDGE\_HIGH |  | WC | 0x0 |
| 26 | GPIO 6\_EDGE\_LOW |  | WC | 0x0 |
| 25 | GPIO 6\_LEVEL\_HIGH |  | RO | 0x0 |
| 24 | GPIO 6\_LEVEL\_LOW |  | RO | 0x0 |
| 23 | GPIO 5\_EDGE\_HIGH |  | WC | 0x0 |
| 22 | GPIO5\_EDGE\_LOW |  | WC | 0x0 |
| 21 | GPIO 5\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO5\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO 4\_EDGE\_HIGH |  | WC | 0x0 |
| 18 | GPIO 4\_EDGE\_LOW |  | WC | 0x0 |
| 17 | GPIO 4\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO 4\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 3\_EDGE\_HIGH |  | WC | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 14 | GPIO3\_EDGE\_LOW |  | WC | 0x0 |
| 13 | GPIO 3\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO3\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 2\_EDGE\_HIGH |  | WC | 0x0 |
| 10 | GPIO2\_EDGE\_LOW |  | WC | 0x0 |
| 9 | GPIO 2\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO2\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO1\_EDGE\_HIGH |  | WC | 0x0 |
| 6 | GPIO 1\_EDGE\_LOW |  | WC | 0x0 |
| 5 | GPIO 1\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO 1\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO0\_EDGE\_HIGH |  | WC | 0x0 |
| 2 | GPIO0\_EDGE\_LOW |  | WC | 0x0 |
| 1 | GPIO0\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO0\_LEVEL\_LOW |  | RO | 0x0 |

表287.INTR1

寄存器

#### [IO\_BANK0](#_bookmark27):INTR1寄存器

**偏移量**:0x0f4

描述

原始中断

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 15\_EDGE\_HIGH |  | WC | 0x0 |
| 30 | GPIO15\_EDGE\_LOW |  | WC | 0x0 |
| 29 | GPIO 15\_LEVEL\_HIGH |  | RO | 0x0 |
| 28 | GPIO 15\_LEVEL\_LOW |  | RO | 0x0 |
| 27 | GPIO14\_EDGE\_HIGH |  | WC | 0x0 |
| 26 | GPIO 14\_EDGE\_LOW |  | WC | 0x0 |
| 25 | GPIO 14\_LEVEL\_HIGH |  | RO | 0x0 |
| 24 | GPIO 14\_LEVEL\_LOW |  | RO | 0x0 |
| 23 | GPIO 13\_EDGE\_HIGH |  | WC | 0x0 |
| 22 | GPIO 13\_EDGE\_LOW |  | WC | 0x0 |
| 21 | GPIO 13\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO 13\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO 12\_EDGE\_HIGH |  | WC | 0x0 |
| 18 | GPIO 12\_EDGE\_LOW |  | WC | 0x0 |
| 17 | GPIO 12\_LEVEL\_HIGH |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 16 | GPIO 12\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 11\_EDGE\_HIGH |  | WC | 0x0 |
| 14 | GPIO 11\_EDGE\_LOW |  | WC | 0x0 |
| 13 | GPIO 11\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO 11\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 10\_EDGE\_HIGH |  | WC | 0x0 |
| 10 | GPIO 10\_EDGE\_LOW |  | WC | 0x0 |
| 9 | GPIO 10\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO 10\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO 9\_EDGE\_HIGH |  | WC | 0x0 |
| 6 | GPIO9\_EDGE\_LOW |  | WC | 0x0 |
| 5 | GPIO9\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO9\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO 8\_EDGE\_HIGH |  | WC | 0x0 |
| 2 | GPIO8\_EDGE\_LOW |  | WC | 0x0 |
| 1 | GPIO8\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO8\_LEVEL\_LOW |  | RO | 0x0 |

表288。INTR2

寄存器

#### [IO\_BANK0](#_bookmark27):INTR2寄存器

**偏移量**:0x0f8

描述

原始中断

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 23\_EDGE\_HIGH |  | WC | 0x0 |
| 30 | GPIO23\_EDGE\_LOW |  | WC | 0x0 |
| 29 | GPIO23\_LEVEL\_HIGH |  | RO | 0x0 |
| 28 | GPIO 23\_LEVEL\_LOW |  | RO | 0x0 |
| 27 | GPIO 22\_EDGE\_HIGH |  | WC | 0x0 |
| 26 | GPIO22\_EDGE\_LOW |  | WC | 0x0 |
| 25 | GPIO 22\_LEVEL\_HIGH |  | RO | 0x0 |
| 24 | GPIO22\_LEVEL\_LOW |  | RO | 0x0 |
| 23 | GPIO 21\_EDGE\_HIGH |  | WC | 0x0 |
| 22 | GPIO21\_EDGE\_LOW |  | WC | 0x0 |
| 21 | GPIO 21\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO21\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO 20\_EDGE\_HIGH |  | WC | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 18 | GPIO20\_EDGE\_LOW |  | WC | 0x0 |
| 17 | GPIO20\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO20\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 19\_EDGE\_HIGH |  | WC | 0x0 |
| 14 | GPIO 19\_EDGE\_LOW |  | WC | 0x0 |
| 13 | GPIO 19\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO 19\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 18\_EDGE\_HIGH |  | WC | 0x0 |
| 10 | GPIO 18\_EDGE\_LOW |  | WC | 0x0 |
| 9 | GPIO 18\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO 18\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO 17\_EDGE\_HIGH |  | WC | 0x0 |
| 6 | GPIO 17\_EDGE\_LOW |  | WC | 0x0 |
| 5 | GPIO 17\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO 17\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO 16\_EDGE\_HIGH |  | WC | 0x0 |
| 2 | GPIO 16\_EDGE\_LOW |  | WC | 0x0 |
| 1 | GPIO 16\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO 16\_LEVEL\_LOW |  | RO | 0x0 |

表289.INTR 3

寄存器

#### [IO\_BANK0](#_bookmark27):INTR3寄存器

**偏移量**:0x0fc

描述

原始中断

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO 29\_EDGE\_HIGH |  | WC | 0x0 |
| 22 | GPIO29\_EDGE\_LOW |  | WC | 0x0 |
| 21 | GPIO 29\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO 29\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO 28\_EDGE\_HIGH |  | WC | 0x0 |
| 18 | GPIO 28\_EDGE\_LOW |  | WC | 0x0 |
| 17 | GPIO 28\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO 28\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 27\_EDGE\_HIGH |  | WC | 0x0 |
| 14 | GPIO27\_EDGE\_LOW |  | WC | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 13 | GPIO 27\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO27\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 26\_EDGE\_HIGH |  | WC | 0x0 |
| 10 | GPIO26\_EDGE\_LOW |  | WC | 0x0 |
| 9 | GPIO 26\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO 26\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO 25\_EDGE\_HIGH |  | WC | 0x0 |
| 6 | GPIO25\_EDGE\_LOW |  | WC | 0x0 |
| 5 | GPIO 25\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO 25\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO 24\_EDGE\_HIGH |  | WC | 0x0 |
| 2 | GPIO24\_EDGE\_LOW |  | WC | 0x0 |
| 1 | GPIO 24\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO 24\_LEVEL\_LOW |  | RO | 0x0 |

表290.第290段0\_INTE0寄存器

#### [IO\_BANK0](#_bookmark27):IO0\_INTE0寄存器

**偏移**:0x100

描述

启用proc0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 7\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO7\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 7\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO7\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO 6\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO 6\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 6\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO 6\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO 5\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO5\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 5\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 5\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 4\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO 4\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 4\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 4\_LEVEL\_LOW |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 15 | GPIO 3\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO3\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 3\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO3\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 2\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO2\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 2\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO2\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO1\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO 1\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO 1\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 1\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO0\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO0\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO0\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO0\_LEVEL\_LOW |  | RW | 0x0 |

表291.0\_INTE1寄存器

#### [IO\_BANK0](#_bookmark27):IO0\_INTE1寄存器

**偏移**:0x104

描述

启用proc0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 15\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO 15\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 15\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO 15\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO14\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO 14\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 14\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO 14\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO 13\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO 13\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 13\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 13\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 12\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO 12\_EDGE\_LOW |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 17 | GPIO 12\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 12\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 11\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO 11\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 11\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO 11\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 10\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO 10\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 10\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 10\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 9\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO9\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO9\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO9\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 8\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO8\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO8\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO8\_LEVEL\_LOW |  | RW | 0x0 |

表292.0\_INTE2寄存器

#### [IO\_BANK0](#_bookmark27):IO0\_INTE2寄存器

**偏移**:0x108

描述

启用proc0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 23\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO23\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 23\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO 23\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO 22\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO22\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 22\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO22\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO21\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO21\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 21\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO21\_LEVEL\_LOW |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 19 | GPIO 20\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO20\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO20\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO20\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 19\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO 19\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 19\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO 19\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 18\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO 18\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 18\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 18\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 17\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO 17\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO 17\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 17\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 16\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO 16\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO 16\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO 16\_LEVEL\_LOW |  | RW | 0x0 |

表293.0\_INTE3寄存器

#### [IO\_BANK0](#_bookmark27):IO0\_INTE3寄存器

**偏移**:0x10c

描述

启用proc0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO 29\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO29\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 29\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 29\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 28\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO 28\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 28\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 28\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 27\_EDGE\_HIGH |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 14 | GPIO27\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 27\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO27\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 26\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO26\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 26\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 26\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 25\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO25\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO 25\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 25\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 24\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO24\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO 24\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO 24\_LEVEL\_LOW |  | RW | 0x0 |

表294.INTF 0寄存器

#### [IO\_BANK0](#_bookmark27):INTF 0寄存器

**偏移**:0x110

描述

Proc0的力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 7\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO7\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 7\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO7\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO 6\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO 6\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 6\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO 6\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO 5\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO5\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 5\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 5\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 4\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO 4\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 4\_LEVEL\_HIGH |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 16 | GPIO 4\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 3\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO3\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 3\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO3\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 2\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO2\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 2\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO2\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO1\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO 1\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO1\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 1\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO0\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO0\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO0\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO0\_LEVEL\_LOW |  | RW | 0x0 |

表295.0\_INTF 1寄存器

#### [IO\_BANK0](#_bookmark27):IO0\_INTF 1寄存器

**偏移**:0x114

描述

Proc0的力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 15\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO15\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 15\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO 15\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO 14\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO 14\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 14\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO 14\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO 13\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO 13\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 13\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 13\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 12\_EDGE\_HIGH |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 18 | GPIO 12\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 12\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 12\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 11\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO 11\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 11\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO 11\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 10\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO 10\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 10\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 10\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 9\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO9\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO9\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO9\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 8\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO8\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO8\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO8\_LEVEL\_LOW |  | RW | 0x0 |

表296.PROC0\_INTF 2寄存器

#### [IO\_BANK0](#_bookmark27):IO0\_INTF 2寄存器

**偏移**:0x118

描述

Proc0的力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 23\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO23\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 23\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO 23\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO 22\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO22\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 22\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO22\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO21\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO21\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 21\_LEVEL\_HIGH |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 20 | GPIO21\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 20\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO20\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO20\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO20\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 19\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO 19\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 19\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO 19\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 18\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO 18\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 18\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 18\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 17\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO 17\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO 17\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 17\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 16\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO 16\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO 16\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO 16\_LEVEL\_LOW |  | RW | 0x0 |

表297.0\_INTF 3寄存器

#### [IO\_BANK0](#_bookmark27):IO0\_INTF 3寄存器

**偏移**:0x11c

描述

Proc0的力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO 29\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO29\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 29\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 29\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 28\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO28\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 28\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 28\_LEVEL\_LOW |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 15 | GPIO 27\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO27\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 27\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO27\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 26\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO26\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 26\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 26\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 25\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO25\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO 25\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 25\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 24\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO24\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO 24\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO 24\_LEVEL\_LOW |  | RW | 0x0 |

表298.0\_INTS0

寄存器

#### [IO\_BANK0](#_bookmark27):INTS0寄存器

**偏移**:0x120

描述

proc0强制屏蔽后的状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 7\_EDGE\_HIGH |  | RO | 0x0 |
| 30 | GPIO7\_EDGE\_LOW |  | RO | 0x0 |
| 29 | GPIO 7\_LEVEL\_HIGH |  | RO | 0x0 |
| 28 | GPIO7\_LEVEL\_LOW |  | RO | 0x0 |
| 27 | GPIO 6\_EDGE\_HIGH |  | RO | 0x0 |
| 26 | GPIO 6\_EDGE\_LOW |  | RO | 0x0 |
| 25 | GPIO 6\_LEVEL\_HIGH |  | RO | 0x0 |
| 24 | GPIO 6\_LEVEL\_LOW |  | RO | 0x0 |
| 23 | GPIO 5\_EDGE\_HIGH |  | RO | 0x0 |
| 22 | GPIO5\_EDGE\_LOW |  | RO | 0x0 |
| 21 | GPIO 5\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO 5\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO 4\_EDGE\_HIGH |  | RO | 0x0 |
| 18 | GPIO 4\_EDGE\_LOW |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 17 | GPIO 4\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO 4\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 3\_EDGE\_HIGH |  | RO | 0x0 |
| 14 | GPIO3\_EDGE\_LOW |  | RO | 0x0 |
| 13 | GPIO 3\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO3\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 2\_EDGE\_HIGH |  | RO | 0x0 |
| 10 | GPIO2\_EDGE\_LOW |  | RO | 0x0 |
| 9 | GPIO 2\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO2\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO 1\_EDGE\_HIGH |  | RO | 0x0 |
| 6 | GPIO 1\_EDGE\_LOW |  | RO | 0x0 |
| 5 | GPIO 1\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO 1\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO0\_EDGE\_HIGH |  | RO | 0x0 |
| 2 | GPIO0\_EDGE\_LOW |  | RO | 0x0 |
| 1 | GPIO0\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO0\_LEVEL\_LOW |  | RO | 0x0 |

表299.0\_INTS1

寄存器

#### [IO\_BANK0](#_bookmark27):INTS0\_1寄存器

**偏移**:0x124

描述

proc0强制屏蔽后的状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 15\_EDGE\_HIGH |  | RO | 0x0 |
| 30 | GPIO15\_EDGE\_LOW |  | RO | 0x0 |
| 29 | GPIO 15\_LEVEL\_HIGH |  | RO | 0x0 |
| 28 | GPIO 15\_LEVEL\_LOW |  | RO | 0x0 |
| 27 | GPIO14\_EDGE\_HIGH |  | RO | 0x0 |
| 26 | GPIO 14\_EDGE\_LOW |  | RO | 0x0 |
| 25 | GPIO 14\_LEVEL\_HIGH |  | RO | 0x0 |
| 24 | GPIO 14\_LEVEL\_LOW |  | RO | 0x0 |
| 23 | GPIO 13\_EDGE\_HIGH |  | RO | 0x0 |
| 22 | GPIO 13\_EDGE\_LOW |  | RO | 0x0 |
| 21 | GPIO 13\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO 13\_LEVEL\_LOW |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 19 | GPIO 12\_EDGE\_HIGH |  | RO | 0x0 |
| 18 | GPIO 12\_EDGE\_LOW |  | RO | 0x0 |
| 17 | GPIO 12\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO 12\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 11\_EDGE\_HIGH |  | RO | 0x0 |
| 14 | GPIO 11\_EDGE\_LOW |  | RO | 0x0 |
| 13 | GPIO 11\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO 11\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 10\_EDGE\_HIGH |  | RO | 0x0 |
| 10 | GPIO 10\_EDGE\_LOW |  | RO | 0x0 |
| 9 | GPIO 10\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO 10\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO 9\_EDGE\_HIGH |  | RO | 0x0 |
| 6 | GPIO9\_EDGE\_LOW |  | RO | 0x0 |
| 5 | GPIO9\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO9\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO 8\_EDGE\_HIGH |  | RO | 0x0 |
| 2 | GPIO8\_EDGE\_LOW |  | RO | 0x0 |
| 1 | GPIO8\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO8\_LEVEL\_LOW |  | RO | 0x0 |

300号桌产品编号0\_INTS2

寄存器

#### [IO\_BANK0](#_bookmark27):INTS0\_2寄存器

**偏移**:0x128

描述

proc0强制屏蔽后的状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 23\_EDGE\_HIGH |  | RO | 0x0 |
| 30 | GPIO23\_EDGE\_LOW |  | RO | 0x0 |
| 29 | GPIO 23\_LEVEL\_HIGH |  | RO | 0x0 |
| 28 | GPIO 23\_LEVEL\_LOW |  | RO | 0x0 |
| 27 | GPIO 22\_EDGE\_HIGH |  | RO | 0x0 |
| 26 | GPIO22\_EDGE\_LOW |  | RO | 0x0 |
| 25 | GPIO 22\_LEVEL\_HIGH |  | RO | 0x0 |
| 24 | GPIO22\_LEVEL\_LOW |  | RO | 0x0 |
| 23 | GPIO21\_EDGE\_HIGH |  | RO | 0x0 |
| 22 | GPIO21\_EDGE\_LOW |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 21 | GPIO 21\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO21\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO 20\_EDGE\_HIGH |  | RO | 0x0 |
| 18 | GPIO20\_EDGE\_LOW |  | RO | 0x0 |
| 17 | GPIO20\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO20\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 19\_EDGE\_HIGH |  | RO | 0x0 |
| 14 | GPIO 19\_EDGE\_LOW |  | RO | 0x0 |
| 13 | GPIO 19\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO 19\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 18\_EDGE\_HIGH |  | RO | 0x0 |
| 10 | GPIO 18\_EDGE\_LOW |  | RO | 0x0 |
| 9 | GPIO 18\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO 18\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO 17\_EDGE\_HIGH |  | RO | 0x0 |
| 6 | GPIO 17\_EDGE\_LOW |  | RO | 0x0 |
| 5 | GPIO 17\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO 17\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO 16\_EDGE\_HIGH |  | RO | 0x0 |
| 2 | GPIO 16\_EDGE\_LOW |  | RO | 0x0 |
| 1 | GPIO 16\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO 16\_LEVEL\_LOW |  | RO | 0x0 |

301号桌产品编号0\_INTS3

寄存器

#### [IO\_BANK0](#_bookmark27):INTS0\_3寄存器

**偏移**:0x12c

描述

proc0强制屏蔽后的状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO 29\_EDGE\_HIGH |  | RO | 0x0 |
| 22 | GPIO29\_EDGE\_LOW |  | RO | 0x0 |
| 21 | GPIO 29\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO 29\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO 28\_EDGE\_HIGH |  | RO | 0x0 |
| 18 | GPIO 28\_EDGE\_LOW |  | RO | 0x0 |
| 17 | GPIO 28\_LEVEL\_HIGH |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 16 | GPIO 28\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 27\_EDGE\_HIGH |  | RO | 0x0 |
| 14 | GPIO27\_EDGE\_LOW |  | RO | 0x0 |
| 13 | GPIO 27\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO27\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 26\_EDGE\_HIGH |  | RO | 0x0 |
| 10 | GPIO26\_EDGE\_LOW |  | RO | 0x0 |
| 9 | GPIO 26\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO 26\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO 25\_EDGE\_HIGH |  | RO | 0x0 |
| 6 | GPIO25\_EDGE\_LOW |  | RO | 0x0 |
| 5 | GPIO 25\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO 25\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO 24\_EDGE\_HIGH |  | RO | 0x0 |
| 2 | GPIO24\_EDGE\_LOW |  | RO | 0x0 |
| 1 | GPIO 24\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO 24\_LEVEL\_LOW |  | RO | 0x0 |

302号桌INTE1\_INTE0寄存器

#### [IO\_BANK0](#_bookmark27):IO1\_INTE0寄存器

**偏移**:0x130

描述

启用proc1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 7\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO7\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 7\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO7\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO 6\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO 6\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 6\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO 6\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO 5\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO5\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 5\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 5\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 4\_EDGE\_HIGH |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 18 | GPIO 4\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 4\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 4\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 3\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO3\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 3\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO3\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 2\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO2\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 2\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO2\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 1\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO 1\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO1\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 1\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO0\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO0\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO0\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO0\_LEVEL\_LOW |  | RW | 0x0 |

303号桌INTE1\_INTE1寄存器

#### [IO\_BANK0](#_bookmark27):IO1\_INTE1寄存器

**偏移**:0x134

描述

启用proc1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 15\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO15\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 15\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO 15\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO 14\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO 14\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 14\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO 14\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO 13\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO 13\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 13\_LEVEL\_HIGH |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 20 | GPIO 13\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 12\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO 12\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 12\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 12\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 11\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO 11\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 11\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO 11\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 10\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO 10\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 10\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 10\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 9\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO9\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO9\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO9\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 8\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO8\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO8\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO8\_LEVEL\_LOW |  | RW | 0x0 |

304号桌1\_INTE2寄存器

#### [IO\_BANK0](#_bookmark27):IO1\_INTE2寄存器

**偏移**:0x138

描述

启用proc1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 23\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO23\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO23\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO 23\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO 22\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO22\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 22\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO22\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO21\_EDGE\_HIGH |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 22 | GPIO21\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 21\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO21\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 20\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO20\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO20\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO20\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 19\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO 19\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 19\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO 19\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 18\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO 18\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 18\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 18\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 17\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO 17\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO 17\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 17\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 16\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO16\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO 16\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO 16\_LEVEL\_LOW |  | RW | 0x0 |

305号桌1\_INTE3寄存器

#### [IO\_BANK0](#_bookmark27):IO1\_INTE3寄存器

**偏移**:0x13c

描述

启用proc1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO 29\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO29\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 29\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 29\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 28\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO28\_EDGE\_LOW |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 17 | GPIO 28\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 28\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 27\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO27\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 27\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO27\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 26\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO26\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 26\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 26\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 25\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO25\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO 25\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 25\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO24\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO24\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO 24\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO 24\_LEVEL\_LOW |  | RW | 0x0 |

表306。INTF 1\_0寄存器

#### [IO\_BANK0](#_bookmark27):IO1\_INTF 0寄存器

**偏移**:0x140

描述

Proc1的力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 7\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO7\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 7\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO7\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO 6\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO 6\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 6\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO 6\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO 5\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO5\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 5\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 5\_LEVEL\_LOW |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 19 | GPIO 4\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO 4\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 4\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 4\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 3\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO3\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 3\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO3\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 2\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO2\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 2\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO2\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 1\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO 1\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO1\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 1\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO0\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO0\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO0\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO0\_LEVEL\_LOW |  | RW | 0x0 |

307号桌INTF 1寄存器

#### [IO\_BANK0](#_bookmark27):INTF1寄存器

**偏移**:0x144

描述

Proc1的力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 15\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO 15\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 15\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO 15\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO14\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO 14\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 14\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO 14\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO 13\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO 13\_EDGE\_LOW |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 21 | GPIO 13\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 13\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 12\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO 12\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 12\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 12\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 11\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO 11\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 11\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO 11\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 10\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO 10\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 10\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 10\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 9\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO9\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO9\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO9\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 8\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO8\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO8\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO8\_LEVEL\_LOW |  | RW | 0x0 |

表308。INTF 1\_2寄存器

#### [IO\_BANK0](#_bookmark27):IO1\_INTF 2寄存器

**偏移**:0x148

描述

Proc1的力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 23\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO23\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 23\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO 23\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO 22\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO22\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 22\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO22\_LEVEL\_LOW |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 23 | GPIO21\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO21\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 21\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO21\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 20\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO20\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO20\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO20\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 19\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO 19\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 19\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO 19\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 18\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO 18\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 18\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 18\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 17\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO 17\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO 17\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 17\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 16\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO 16\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO 16\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO 16\_LEVEL\_LOW |  | RW | 0x0 |

表309。INTF3寄存器

#### [IO\_BANK0](#_bookmark27):IO1\_INTF 3寄存器

**偏移**:0x14c

描述

Proc1的力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO 29\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO29\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 29\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 29\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 28\_EDGE\_HIGH |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 18 | GPIO28\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 28\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 28\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 27\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO27\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 27\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO27\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 26\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO26\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 26\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 26\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 25\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO25\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO 25\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 25\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 24\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO24\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO 24\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO 24\_LEVEL\_LOW |  | RW | 0x0 |

表310。INTS 1\_0

寄存器

#### [IO\_BANK0](#_bookmark27):INTS0寄存器

**偏移**:0x150

描述

proc1强制屏蔽后的状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 7\_EDGE\_HIGH |  | RO | 0x0 |
| 30 | GPIO7\_EDGE\_LOW |  | RO | 0x0 |
| 29 | GPIO 7\_LEVEL\_HIGH |  | RO | 0x0 |
| 28 | GPIO7\_LEVEL\_LOW |  | RO | 0x0 |
| 27 | GPIO 6\_EDGE\_HIGH |  | RO | 0x0 |
| 26 | GPIO 6\_EDGE\_LOW |  | RO | 0x0 |
| 25 | GPIO 6\_LEVEL\_HIGH |  | RO | 0x0 |
| 24 | GPIO 6\_LEVEL\_LOW |  | RO | 0x0 |
| 23 | GPIO 5\_EDGE\_HIGH |  | RO | 0x0 |
| 22 | GPIO5\_EDGE\_LOW |  | RO | 0x0 |
| 21 | GPIO 5\_LEVEL\_HIGH |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 20 | GPIO 5\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO 4\_EDGE\_HIGH |  | RO | 0x0 |
| 18 | GPIO 4\_EDGE\_LOW |  | RO | 0x0 |
| 17 | GPIO 4\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO 4\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 3\_EDGE\_HIGH |  | RO | 0x0 |
| 14 | GPIO3\_EDGE\_LOW |  | RO | 0x0 |
| 13 | GPIO 3\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO3\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 2\_EDGE\_HIGH |  | RO | 0x0 |
| 10 | GPIO2\_EDGE\_LOW |  | RO | 0x0 |
| 9 | GPIO 2\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO2\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO 1\_EDGE\_HIGH |  | RO | 0x0 |
| 6 | GPIO 1\_EDGE\_LOW |  | RO | 0x0 |
| 5 | GPIO 1\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO 1\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO0\_EDGE\_HIGH |  | RO | 0x0 |
| 2 | GPIO0\_EDGE\_LOW |  | RO | 0x0 |
| 1 | GPIO0\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO0\_LEVEL\_LOW |  | RO | 0x0 |

表311。INTS 1

寄存器

#### [IO\_BANK0](#_bookmark27):INTS1寄存器

**偏移**:0x154

描述

proc1强制屏蔽后的状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 15\_EDGE\_HIGH |  | RO | 0x0 |
| 30 | GPIO 15\_EDGE\_LOW |  | RO | 0x0 |
| 29 | GPIO 15\_LEVEL\_HIGH |  | RO | 0x0 |
| 28 | GPIO 15\_LEVEL\_LOW |  | RO | 0x0 |
| 27 | GPIO 14\_EDGE\_HIGH |  | RO | 0x0 |
| 26 | GPIO 14\_EDGE\_LOW |  | RO | 0x0 |
| 25 | GPIO 14\_LEVEL\_HIGH |  | RO | 0x0 |
| 24 | GPIO 14\_LEVEL\_LOW |  | RO | 0x0 |
| 23 | GPIO 13\_EDGE\_HIGH |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 22 | GPIO 13\_EDGE\_LOW |  | RO | 0x0 |
| 21 | GPIO 13\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO 13\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO 12\_EDGE\_HIGH |  | RO | 0x0 |
| 18 | GPIO 12\_EDGE\_LOW |  | RO | 0x0 |
| 17 | GPIO 12\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO 12\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 11\_EDGE\_HIGH |  | RO | 0x0 |
| 14 | GPIO 11\_EDGE\_LOW |  | RO | 0x0 |
| 13 | GPIO 11\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO 11\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 10\_EDGE\_HIGH |  | RO | 0x0 |
| 10 | GPIO 10\_EDGE\_LOW |  | RO | 0x0 |
| 9 | GPIO 10\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO 10\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO 9\_EDGE\_HIGH |  | RO | 0x0 |
| 6 | GPIO9\_EDGE\_LOW |  | RO | 0x0 |
| 5 | GPIO9\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO9\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO 8\_EDGE\_HIGH |  | RO | 0x0 |
| 2 | GPIO8\_EDGE\_LOW |  | RO | 0x0 |
| 1 | GPIO8\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO8\_LEVEL\_LOW |  | RO | 0x0 |

表312。PROC1\_INTS2

寄存器

#### [IO\_BANK0](#_bookmark27):INTS2寄存器

**偏移**:0x158

描述

proc1强制屏蔽后的状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 23\_EDGE\_HIGH |  | RO | 0x0 |
| 30 | GPIO23\_EDGE\_LOW |  | RO | 0x0 |
| 29 | GPIO23\_LEVEL\_HIGH |  | RO | 0x0 |
| 28 | GPIO23\_LEVEL\_LOW |  | RO | 0x0 |
| 27 | GPIO 22\_EDGE\_HIGH |  | RO | 0x0 |
| 26 | GPIO22\_EDGE\_LOW |  | RO | 0x0 |
| 25 | GPIO 22\_LEVEL\_HIGH |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 24 | GPIO22\_LEVEL\_LOW |  | RO | 0x0 |
| 23 | GPIO 21\_EDGE\_HIGH |  | RO | 0x0 |
| 22 | GPIO21\_EDGE\_LOW |  | RO | 0x0 |
| 21 | GPIO 21\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO21\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO 20\_EDGE\_HIGH |  | RO | 0x0 |
| 18 | GPIO20\_EDGE\_LOW |  | RO | 0x0 |
| 17 | GPIO20\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO20\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 19\_EDGE\_HIGH |  | RO | 0x0 |
| 14 | GPIO 19\_EDGE\_LOW |  | RO | 0x0 |
| 13 | GPIO 19\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO 19\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 18\_EDGE\_HIGH |  | RO | 0x0 |
| 10 | GPIO 18\_EDGE\_LOW |  | RO | 0x0 |
| 9 | GPIO 18\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO 18\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO 17\_EDGE\_HIGH |  | RO | 0x0 |
| 6 | GPIO 17\_EDGE\_LOW |  | RO | 0x0 |
| 5 | GPIO 17\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO 17\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO 16\_EDGE\_HIGH |  | RO | 0x0 |
| 2 | GPIO16\_EDGE\_LOW |  | RO | 0x0 |
| 1 | GPIO 16\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO 16\_LEVEL\_LOW |  | RO | 0x0 |

表313。PROC1\_INTS3

寄存器

#### [IO\_BANK0](#_bookmark27):INTS3寄存器

**偏移**:0x15c

描述

proc1强制屏蔽后的状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO 29\_EDGE\_HIGH |  | RO | 0x0 |
| 22 | GPIO29\_EDGE\_LOW |  | RO | 0x0 |
| 21 | GPIO 29\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO 29\_LEVEL\_LOW |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 19 | GPIO 28\_EDGE\_HIGH |  | RO | 0x0 |
| 18 | GPIO 28\_EDGE\_LOW |  | RO | 0x0 |
| 17 | GPIO 28\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO 28\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 27\_EDGE\_HIGH |  | RO | 0x0 |
| 14 | GPIO27\_EDGE\_LOW |  | RO | 0x0 |
| 13 | GPIO 27\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO27\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 26\_EDGE\_HIGH |  | RO | 0x0 |
| 10 | GPIO26\_EDGE\_LOW |  | RO | 0x0 |
| 9 | GPIO 26\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO 26\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO 25\_EDGE\_HIGH |  | RO | 0x0 |
| 6 | GPIO25\_EDGE\_LOW |  | RO | 0x0 |
| 5 | GPIO 25\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO 25\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO 24\_EDGE\_HIGH |  | RO | 0x0 |
| 2 | GPIO24\_EDGE\_LOW |  | RO | 0x0 |
| 1 | GPIO 24\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO 24\_LEVEL\_LOW |  | RO | 0x0 |

表314。 DORMANT\_唤醒\_内部

E0寄存器

#### [IO\_BANK0](#_bookmark27):DORMANT\_WAKE\_INTE0寄存器

**偏移**:0x160

描述

启用 DORMANT\_唤醒

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 7\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO7\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 7\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO7\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO 6\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO 6\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 6\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO 6\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO 5\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO5\_EDGE\_LOW |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 21 | GPIO 5\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 5\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 4\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO 4\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 4\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 4\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 3\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO3\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 3\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO3\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 2\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO2\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 2\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO2\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 1\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO 1\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO 1\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 1\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO0\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO0\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO0\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO0\_LEVEL\_LOW |  | RW | 0x0 |

表315。 DORMANT\_唤醒\_内部

E1寄存器

#### [IO\_BANK0](#_bookmark27):DORMANT\_WAKE\_INTE1寄存器

**偏移**:0x164

描述

启用 DORMANT\_唤醒

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 15\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO 15\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 15\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO 15\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO14\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO 14\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 14\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO 14\_LEVEL\_LOW |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 23 | GPIO 13\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO 13\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 13\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 13\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 12\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO 12\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 12\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 12\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 11\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO 11\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 11\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO 11\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 10\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO 10\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 10\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 10\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 9\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO9\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO9\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO9\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 8\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO8\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO8\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO8\_LEVEL\_LOW |  | RW | 0x0 |

表316。 DORMANT\_唤醒\_内部

E2寄存器

#### [IO\_BANK0](#_bookmark27):DORMANT\_WAKE\_INTE2寄存器

**偏移**:0x168

描述

DORMANT\_唤醒中断启用

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 23\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO23\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO23\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO 23\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO 22\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO22\_EDGE\_LOW |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 25 | GPIO 22\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO22\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO21\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO21\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 21\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO21\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 20\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO20\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO20\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO20\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 19\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO 19\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 19\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO 19\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 18\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO 18\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 18\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 18\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 17\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO 17\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO 17\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 17\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 16\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO16\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO 16\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO 16\_LEVEL\_LOW |  | RW | 0x0 |

317号表 DORMANT\_唤醒\_内部

E3寄存器

#### [IO\_BANK0](#_bookmark27):DORMANT\_WAKE\_INTE3寄存器

**偏移**:0x16c

描述

DORMANT\_唤醒中断启用

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO 29\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO29\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 29\_LEVEL\_HIGH |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 20 | GPIO 29\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 28\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO 28\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 28\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 28\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 27\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO27\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 27\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO27\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 26\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO26\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 26\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 26\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 25\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO25\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO 25\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 25\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 24\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO24\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO 24\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO 24\_LEVEL\_LOW |  | RW | 0x0 |

表318。 DORMANT\_唤醒\_内部

F0寄存器

#### [IO\_BANK0](#_bookmark27):DORMANT\_WAKE\_INTF 0寄存器

**偏移**:0x170

描述

DORMANT\_唤醒的中断力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 7\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO7\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 7\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO7\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO 6\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO 6\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 6\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO 6\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO 5\_EDGE\_HIGH |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 22 | GPIO5\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 5\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 5\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 4\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO 4\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 4\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 4\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 3\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO3\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 3\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO3\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 2\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO2\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 2\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO2\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 1\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO 1\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO 1\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 1\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO0\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO0\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO0\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO0\_LEVEL\_LOW |  | RW | 0x0 |

表319。 DORMANT\_唤醒\_内部

F1寄存器

#### [IO\_BANK0](#_bookmark27):DORMANT\_WAKE\_INTF 1寄存器

**偏移**:0x174

描述

DORMANT\_唤醒的力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 15\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO15\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 15\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO 15\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO14\_EDGE\_HIGH |  | RW | 0x0 |
| 26 | GPIO 14\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 14\_LEVEL\_HIGH |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 24 | GPIO 14\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO 13\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO 13\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 13\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 13\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 12\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO 12\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 12\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 12\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 11\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO 11\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 11\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO 11\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 10\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO 10\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 10\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 10\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 9\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO9\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO9\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO9\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 8\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO8\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO8\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO8\_LEVEL\_LOW |  | RW | 0x0 |

表320. DORMANT\_唤醒\_内部

F2寄存器

#### [IO\_BANK 0](#_bookmark27):DORMANT\_WAKE\_INTF 2寄存器

**偏移**:0x178

描述

DORMANT\_唤醒的力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 23\_EDGE\_HIGH |  | RW | 0x0 |
| 30 | GPIO23\_EDGE\_LOW |  | RW | 0x0 |
| 29 | GPIO 23\_LEVEL\_HIGH |  | RW | 0x0 |
| 28 | GPIO 23\_LEVEL\_LOW |  | RW | 0x0 |
| 27 | GPIO 22\_EDGE\_HIGH |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 26 | GPIO22\_EDGE\_LOW |  | RW | 0x0 |
| 25 | GPIO 22\_LEVEL\_HIGH |  | RW | 0x0 |
| 24 | GPIO22\_LEVEL\_LOW |  | RW | 0x0 |
| 23 | GPIO 21\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO21\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO 21\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO21\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 20\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO20\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO20\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO20\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 19\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO 19\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 19\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO 19\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 18\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO 18\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 18\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 18\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 17\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO 17\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO 17\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 17\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 16\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO16\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO 16\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO 16\_LEVEL\_LOW |  | RW | 0x0 |

表321。 DORMANT\_唤醒\_内部

F3寄存器

#### [IO\_BANK0](#_bookmark27):DORMANT\_WAKE\_INTF 3寄存器

**偏移**:0x17c

描述

DORMANT\_唤醒的力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO 29\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO29\_EDGE\_LOW |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 21 | GPIO 29\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO 29\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO 28\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO 28\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO 28\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO 28\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO 27\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO27\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO 27\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO27\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO 26\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO26\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO 26\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO 26\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO 25\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO25\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO 25\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO 25\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO 24\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO24\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO 24\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO 24\_LEVEL\_LOW |  | RW | 0x0 |

表322。 DORMANT\_唤醒\_内部

S0寄存器

#### [IO\_BANK0](#_bookmark27):DORMANT\_WAKE\_INTS 0寄存器

**偏移**:0x180

描述

DORMANT\_唤醒强制屏蔽后的休眠状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 7\_EDGE\_HIGH |  | RO | 0x0 |
| 30 | GPIO7\_EDGE\_LOW |  | RO | 0x0 |
| 29 | GPIO 7\_LEVEL\_HIGH |  | RO | 0x0 |
| 28 | GPIO7\_LEVEL\_LOW |  | RO | 0x0 |
| 27 | GPIO 6\_EDGE\_HIGH |  | RO | 0x0 |
| 26 | GPIO 6\_EDGE\_LOW |  | RO | 0x0 |
| 25 | GPIO 6\_LEVEL\_HIGH |  | RO | 0x0 |
| 24 | GPIO 6\_LEVEL\_LOW |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 23 | GPIO 5\_EDGE\_HIGH |  | RO | 0x0 |
| 22 | GPIO5\_EDGE\_LOW |  | RO | 0x0 |
| 21 | GPIO 5\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO 5\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO 4\_EDGE\_HIGH |  | RO | 0x0 |
| 18 | GPIO 4\_EDGE\_LOW |  | RO | 0x0 |
| 17 | GPIO 4\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO 4\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 3\_EDGE\_HIGH |  | RO | 0x0 |
| 14 | GPIO3\_EDGE\_LOW |  | RO | 0x0 |
| 13 | GPIO 3\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO3\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 2\_EDGE\_HIGH |  | RO | 0x0 |
| 10 | GPIO2\_EDGE\_LOW |  | RO | 0x0 |
| 9 | GPIO 2\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO2\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO 1\_EDGE\_HIGH |  | RO | 0x0 |
| 6 | GPIO 1\_EDGE\_LOW |  | RO | 0x0 |
| 5 | GPIO 1\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO 1\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO0\_EDGE\_HIGH |  | RO | 0x0 |
| 2 | GPIO0\_EDGE\_LOW |  | RO | 0x0 |
| 1 | GPIO0\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO0\_LEVEL\_LOW |  | RO | 0x0 |

表323。 DORMANT\_唤醒\_内部

S1寄存器

#### [IO\_BANK0](#_bookmark27):DORMANT\_WAKE\_INTS 1寄存器

**偏移**:0x184

描述

DORMANT\_唤醒强制屏蔽后的休眠状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 15\_EDGE\_HIGH |  | RO | 0x0 |
| 30 | GPIO 15\_EDGE\_LOW |  | RO | 0x0 |
| 29 | GPIO 15\_LEVEL\_HIGH |  | RO | 0x0 |
| 28 | GPIO 15\_LEVEL\_LOW |  | RO | 0x0 |
| 27 | GPIO14\_EDGE\_HIGH |  | RO | 0x0 |
| 26 | GPIO 14\_EDGE\_LOW |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 25 | GPIO 14\_LEVEL\_HIGH |  | RO | 0x0 |
| 24 | GPIO 14\_LEVEL\_LOW |  | RO | 0x0 |
| 23 | GPIO 13\_EDGE\_HIGH |  | RO | 0x0 |
| 22 | GPIO 13\_EDGE\_LOW |  | RO | 0x0 |
| 21 | GPIO 13\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO 13\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO 12\_EDGE\_HIGH |  | RO | 0x0 |
| 18 | GPIO 12\_EDGE\_LOW |  | RO | 0x0 |
| 17 | GPIO 12\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO 12\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 11\_EDGE\_HIGH |  | RO | 0x0 |
| 14 | GPIO 11\_EDGE\_LOW |  | RO | 0x0 |
| 13 | GPIO 11\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO 11\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 10\_EDGE\_HIGH |  | RO | 0x0 |
| 10 | GPIO 10\_EDGE\_LOW |  | RO | 0x0 |
| 9 | GPIO 10\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO 10\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO 9\_EDGE\_HIGH |  | RO | 0x0 |
| 6 | GPIO9\_EDGE\_LOW |  | RO | 0x0 |
| 5 | GPIO9\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO9\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO 8\_EDGE\_HIGH |  | RO | 0x0 |
| 2 | GPIO8\_EDGE\_LOW |  | RO | 0x0 |
| 1 | GPIO8\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO8\_LEVEL\_LOW |  | RO | 0x0 |

表324. DORMANT\_唤醒\_内部

S2寄存器

#### [IO\_BANK0](#_bookmark27):DORMANT\_WAKE\_INTS 2寄存器

**偏移**:0x188

描述

DORMANT\_唤醒强制屏蔽后的休眠状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | GPIO 23\_EDGE\_HIGH |  | RO | 0x0 |
| 30 | GPIO23\_EDGE\_LOW |  | RO | 0x0 |
| 29 | GPIO23\_LEVEL\_HIGH |  | RO | 0x0 |
| 28 | GPIO 23\_LEVEL\_LOW |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 27 | GPIO 22\_EDGE\_HIGH |  | RO | 0x0 |
| 26 | GPIO22\_EDGE\_LOW |  | RO | 0x0 |
| 25 | GPIO 22\_LEVEL\_HIGH |  | RO | 0x0 |
| 24 | GPIO22\_LEVEL\_LOW |  | RO | 0x0 |
| 23 | GPIO 21\_EDGE\_HIGH |  | RO | 0x0 |
| 22 | GPIO21\_EDGE\_LOW |  | RO | 0x0 |
| 21 | GPIO 21\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO21\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO 20\_EDGE\_HIGH |  | RO | 0x0 |
| 18 | GPIO20\_EDGE\_LOW |  | RO | 0x0 |
| 17 | GPIO20\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO20\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 19\_EDGE\_HIGH |  | RO | 0x0 |
| 14 | GPIO 19\_EDGE\_LOW |  | RO | 0x0 |
| 13 | GPIO 19\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO 19\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 18\_EDGE\_HIGH |  | RO | 0x0 |
| 10 | GPIO 18\_EDGE\_LOW |  | RO | 0x0 |
| 9 | GPIO 18\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO 18\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO 17\_EDGE\_HIGH |  | RO | 0x0 |
| 6 | GPIO 17\_EDGE\_LOW |  | RO | 0x0 |
| 5 | GPIO 17\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO 17\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO 16\_EDGE\_HIGH |  | RO | 0x0 |
| 2 | GPIO 16\_EDGE\_LOW |  | RO | 0x0 |
| 1 | GPIO 16\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO 16\_LEVEL\_LOW |  | RO | 0x0 |

表325。 DORMANT\_唤醒\_内部

S3寄存器

#### [IO\_BANK 0](#_bookmark27):DORMANT\_WAKE\_INTS3寄存器

**偏移**:0x18c

描述

DORMANT\_唤醒强制屏蔽后的休眠状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO 29\_EDGE\_HIGH |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 22 | GPIO29\_EDGE\_LOW |  | RO | 0x0 |
| 21 | GPIO 29\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO 29\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO 28\_EDGE\_HIGH |  | RO | 0x0 |
| 18 | GPIO28\_EDGE\_LOW |  | RO | 0x0 |
| 17 | GPIO 28\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO 28\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO 27\_EDGE\_HIGH |  | RO | 0x0 |
| 14 | GPIO27\_EDGE\_LOW |  | RO | 0x0 |
| 13 | GPIO 27\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO27\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO 26\_EDGE\_HIGH |  | RO | 0x0 |
| 10 | GPIO26\_EDGE\_LOW |  | RO | 0x0 |
| 9 | GPIO 26\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO 26\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO 25\_EDGE\_HIGH |  | RO | 0x0 |
| 6 | GPIO25\_EDGE\_LOW |  | RO | 0x0 |
| 5 | GPIO 25\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO 25\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO 24\_EDGE\_HIGH |  | RO | 0x0 |
| 2 | GPIO24\_EDGE\_LOW |  | RO | 0x0 |
| 1 | GPIO 24\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO 24\_LEVEL\_LOW |  | RO | 0x0 |

* + - 1. IO-QSPI银行

QSPI存储体IO寄存器起始于基址0x40018000（在SDK中定义为IO\_QSPI\_BASE）。

*表326。IO\_QSPI寄存器列表*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [GPIO\_QSPI\_SCLK\_STATUS](#_bookmark71) | GPIO状态 |
| 0x04 | [GPIO\_QSPI\_SCLK\_CTRL](#_bookmark72) | GPIO控制，包括功能选择和覆盖。 |
| 0x08 | [GPIO\_QSPI\_SS\_STATUS](#_bookmark71) | GPIO状态 |
| 0x0C | [GPIO\_QSPI\_SS\_CTRL](#_bookmark72) | GPIO控制，包括功能选择和覆盖。 |
| 0x10 | [GPIO\_QSPI\_SD0\_STATUS](#_bookmark71) | GPIO状态 |
| 0x14 | [GPIO\_QSPI\_SD0\_CTRL](#_bookmark72) | GPIO控制，包括功能选择和覆盖。 |
| 0x18 | [GPIO\_QSPI\_SD1\_STATUS](#_bookmark71) | GPIO状态 |
| 0x1c | [GPIO\_QSPI\_SD1\_CTRL](#_bookmark72) | GPIO控制，包括功能选择和覆盖。 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x20 | [GPIO\_QSPI\_SD2\_STATUS](#_bookmark71) | GPIO状态 |
| 0x24 | [GPIO\_QSPI\_SD2\_CTRL](#_bookmark72) | GPIO控制，包括功能选择和覆盖。 |
| 0x28 | [GPIO\_QSPI\_SD3\_STATUS](#_bookmark71) | GPIO状态 |
| 0x2c | [GPIO\_QSPI\_SD3\_CTRL](#_bookmark72) | GPIO控制，包括功能选择和覆盖。 |
| 0x30 | [INTR](#_bookmark73) | 原始中断 |
| 0x34 | [联系我们](#_bookmark74) | 启用proc0 |
| 0x38 | [联系我们](#_bookmark75) | Proc0的力 |
| 0x3c | [联系我们](#_bookmark76) | proc0强制屏蔽后的状态 |
| 0x40 | [简体中文](#_bookmark77) | 启用proc1 |
| 0x44 | [简体中文](#_bookmark78) | Proc1的力 |
| 0x48 | [简体中文](#_bookmark79) | proc1强制屏蔽后的状态 |
| 0x4c | [DORMANT\_唤醒\_INTE](#_bookmark80) | DORMANT\_唤醒中断启用 |
| 0x50 | [DORMANT\_唤醒\_INTF](#_bookmark81) | DORMANT\_唤醒的力 |
| 0x54 | [DORMANT\_唤醒\_INTS](#_bookmark82) | DORMANT\_唤醒强制屏蔽后的休眠状态 |

表327。GPIO\_QSPI\_SCLK\_STATUS，GPIO\_QSPI\_SS\_STATUS，.，GPIO\_QSPI\_SD2\_STATUS，GPIO\_QSPI\_SD3\_STAT

美国注册

#### [IO\_QSPI](#_bookmark70):GPIO\_QSPI\_SCLK\_STATUS，GPIO\_QSPI\_SS\_STATUS， .，GPIO\_QSPI\_SD2\_STATUS，GPIO\_QSPI\_SD3\_STATUS寄存器

**偏移**:0x00、0x08、...、0x20、0x28

描述

GPIO状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:27 | Reserved. | - | - | - |
| 26 | IRQTOPROC | 在应用覆盖后，中断到处理器 | RO | 0x0 |
| 25 | Reserved. | - | - | - |
| 24 | IRQFROMPAD | 在应用覆盖之前从焊盘中断 | RO | 0x0 |
| 23:20 | Reserved. | - | - | - |
| 19 | 国际歌剧院 | 应用覆盖后，将信号输入到外围设备 | RO | 0x0 |
| 18 | Reserved. | - | - | - |
| 17 | 信息 | 来自焊盘的输入信号，在应用超控 | RO | 0x0 |
| 16:14 | Reserved. | - | - | - |
| 13 | OETOPAD | 在寄存器覆盖后，输出使能到焊盘 | RO | 0x0 |
| 12 | 奥弗龙佩利 | 在应用寄存器覆盖之前，从所选外围设备启用输出 | RO | 0x0 |
| 11:10 | Reserved. | - | - | - |
| 9 | OUTTOPAD | 寄存器覆盖后输出信号到焊盘 | RO | 0x0 |
| 8 | 外龙贝里 | 在应用寄存器覆盖之前，来自所选外设的输出信号 | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 7:0 | Reserved. | - | - | - |

表328。GPIO\_QSPI\_SCLK\_CTRL、GPIO\_QSPI\_SS\_CTRL、

.，GPIO\_QSPI\_SD2\_CTRL，GPIO\_QSPI\_SD3\_CTRL

寄存器

表329。INTR寄存器

#### [IO\_QSPI](#_bookmark70):GPIO\_QSPI\_SCLK\_CTRL，GPIO\_QSPI\_SS\_CTRL， .，

GPIO\_QSPI\_SD2\_CTRL、GPIO\_QSPI\_SD3\_CTRL寄存器

**偏移**:0x04、0x0c、...、0x24、0x2c

描述

GPIO控制，包括功能选择和覆盖。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:30 | Reserved. | - | - | - |
| 29:28 | IRQover | 0x0 →不反转中断0x1→反转中断  0x2→驱动中断低电平0x3→驱动中断高电平 | RW | 0x0 |
| 27:18 | Reserved. | - | - | - |
| 17:16 | INOVER | 0x0→不反相PWM输入0x1→反相PWM输入  0x2→驱动整流器输入低0x3→驱动整流器输入高 | RW | 0x0 |
| 15:14 | Reserved. | - | - | - |
| 13:12 | OEOVER | 0x0→从funcsel选择的外围信号驱动输出使能  0x1→通过funcsel选择的外设信号的反相使能驱动输出  0x2→禁用输出0x3→启用输出 | RW | 0x0 |
| 11:10 | Reserved. | - | - | - |
| 9:8 | Outover | 0x0→从funcsel选择的外围信号驱动输出  0x1→从funcel选择的外围信号的反相驱动输出  0x2→驱动输出低0x3→驱动输出高 | RW | 0x0 |
| 七点五 | Reserved. | - | - | - |
| 4:0 | 功能 | 功能选择。31 == NULL。有关可用功能，请参见GPIO功能表 | RW | 0x1f |

#### [IO\_QSPI](#_bookmark70):INTR寄存器

**偏移**:0x30

描述

原始中断

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 23 | GPIO\_QSPI\_SD3\_EDGE\_HIGH |  | WC | 0x0 |
| 22 | GPIO\_QSPI\_SD3\_EDGE\_LOW |  | WC | 0x0 |
| 21 | GPIO\_QSPI\_SD3\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO\_QSPI\_SD3\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO\_QSPI\_SD2\_EDGE\_HIGH |  | WC | 0x0 |
| 18 | GPIO\_QSPI\_SD2\_EDGE\_LOW |  | WC | 0x0 |
| 17 | GPIO\_QSPI\_SD2\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO\_QSPI\_SD2\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO\_QSPI\_SD1\_EDGE\_HIGH |  | WC | 0x0 |
| 14 | GPIO\_QSPI\_SD1\_EDGE\_LOW |  | WC | 0x0 |
| 13 | GPIO\_QSPI\_SD1\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO\_QSPI\_SD1\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO\_QSPI\_SD0\_EDGE\_HIGH |  | WC | 0x0 |
| 10 | GPIO\_QSPI\_SD0\_EDGE\_LOW |  | WC | 0x0 |
| 9 | GPIO\_QSPI\_SD0\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO\_QSPI\_SD0\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO\_QSPI\_SS\_EDGE\_HIGH |  | WC | 0x0 |
| 6 | GPIO\_QSPI\_SS\_EDGE\_LOW |  | WC | 0x0 |
| 5 | GPIO\_QSPI\_SS\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO\_QSPI\_SS\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO\_QSPI\_SCLK\_EDGE\_HIGH |  | WC | 0x0 |
| 2 | GPIO\_QSPI\_SCLK\_EDGE\_LOW |  | WC | 0x0 |
| 1 | GPIO\_QSPI\_SCLK\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO\_QSPI\_SCLK\_LEVEL\_LOW |  | RO | 0x0 |

表330。0\_INTE寄存器

#### [IO\_QSPI](#_bookmark70):IO 0\_INTE寄存器

**偏移**:0x34

描述

启用proc0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO\_QSPI\_SD3\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO\_QSPI\_SD3\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO\_QSPI\_SD3\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO\_QSPI\_SD3\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO\_QSPI\_SD2\_EDGE\_HIGH |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 18 | GPIO\_QSPI\_SD2\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO\_QSPI\_SD2\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO\_QSPI\_SD2\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO\_QSPI\_SD1\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO\_QSPI\_SD1\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO\_QSPI\_SD1\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO\_QSPI\_SD1\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO\_QSPI\_SD0\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO\_QSPI\_SD0\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO\_QSPI\_SD0\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO\_QSPI\_SD0\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO\_QSPI\_SS\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO\_QSPI\_SS\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO\_QSPI\_SS\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO\_QSPI\_SS\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO\_QSPI\_SCLK\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO\_QSPI\_SCLK\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO\_QSPI\_SCLK\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO\_QSPI\_SCLK\_LEVEL\_LOW |  | RW | 0x0 |

表331。0\_INTF寄存器

#### [IO\_QSPI](#_bookmark70):IO 0\_INTF寄存器

**偏移**:0x38

描述

Proc0的力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO\_QSPI\_SD3\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO\_QSPI\_SD3\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO\_QSPI\_SD3\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO\_QSPI\_SD3\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO\_QSPI\_SD2\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO\_QSPI\_SD2\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO\_QSPI\_SD2\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO\_QSPI\_SD2\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO\_QSPI\_SD1\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO\_QSPI\_SD1\_EDGE\_LOW |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 13 | GPIO\_QSPI\_SD1\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO\_QSPI\_SD1\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO\_QSPI\_SD0\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO\_QSPI\_SD0\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO\_QSPI\_SD0\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO\_QSPI\_SD0\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO\_QSPI\_SS\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO\_QSPI\_SS\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO\_QSPI\_SS\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO\_QSPI\_SS\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO\_QSPI\_SCLK\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO\_QSPI\_SCLK\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO\_QSPI\_SCLK\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO\_QSPI\_SCLK\_LEVEL\_LOW |  | RW | 0x0 |

表332。IN0\_INTS寄存器

#### [IO\_QSPI](#_bookmark70):IO 0\_INTS寄存器

**偏移**:0x3c

描述

proc0强制屏蔽后的状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO\_QSPI\_SD3\_EDGE\_HIGH |  | RO | 0x0 |
| 22 | GPIO\_QSPI\_SD3\_EDGE\_LOW |  | RO | 0x0 |
| 21 | GPIO\_QSPI\_SD3\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO\_QSPI\_SD3\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO\_QSPI\_SD2\_EDGE\_HIGH |  | RO | 0x0 |
| 18 | GPIO\_QSPI\_SD2\_EDGE\_LOW |  | RO | 0x0 |
| 17 | GPIO\_QSPI\_SD2\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO\_QSPI\_SD2\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO\_QSPI\_SD1\_EDGE\_HIGH |  | RO | 0x0 |
| 14 | GPIO\_QSPI\_SD1\_EDGE\_LOW |  | RO | 0x0 |
| 13 | GPIO\_QSPI\_SD1\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO\_QSPI\_SD1\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO\_QSPI\_SD0\_EDGE\_HIGH |  | RO | 0x0 |
| 10 | GPIO\_QSPI\_SD0\_EDGE\_LOW |  | RO | 0x0 |
| 9 | GPIO\_QSPI\_SD0\_LEVEL\_HIGH |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 8 | GPIO\_QSPI\_SD0\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO\_QSPI\_SS\_EDGE\_HIGH |  | RO | 0x0 |
| 6 | GPIO\_QSPI\_SS\_EDGE\_LOW |  | RO | 0x0 |
| 5 | GPIO\_QSPI\_SS\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO\_QSPI\_SS\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO\_QSPI\_SCLK\_EDGE\_HIGH |  | RO | 0x0 |
| 2 | GPIO\_QSPI\_SCLK\_EDGE\_LOW |  | RO | 0x0 |
| 1 | GPIO\_QSPI\_SCLK\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO\_QSPI\_SCLK\_LEVEL\_LOW |  | RO | 0x0 |

表333。1\_INTE寄存器

#### [IO\_QSPI](#_bookmark70):INTE 1\_INTE寄存器

**偏移**:0x40

描述

启用proc1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO\_QSPI\_SD3\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO\_QSPI\_SD3\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO\_QSPI\_SD3\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO\_QSPI\_SD3\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO\_QSPI\_SD2\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO\_QSPI\_SD2\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO\_QSPI\_SD2\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO\_QSPI\_SD2\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO\_QSPI\_SD1\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO\_QSPI\_SD1\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO\_QSPI\_SD1\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO\_QSPI\_SD1\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO\_QSPI\_SD0\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO\_QSPI\_SD0\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO\_QSPI\_SD0\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO\_QSPI\_SD0\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO\_QSPI\_SS\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO\_QSPI\_SS\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO\_QSPI\_SS\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO\_QSPI\_SS\_LEVEL\_LOW |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 3 | GPIO\_QSPI\_SCLK\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO\_QSPI\_SCLK\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO\_QSPI\_SCLK\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO\_QSPI\_SCLK\_LEVEL\_LOW |  | RW | 0x0 |

334号桌INTF寄存器

#### [IO\_QSPI](#_bookmark70):IO 1\_INTF寄存器

**偏移**:0x44

描述

Proc1的力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO\_QSPI\_SD3\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO\_QSPI\_SD3\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO\_QSPI\_SD3\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO\_QSPI\_SD3\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO\_QSPI\_SD2\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO\_QSPI\_SD2\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO\_QSPI\_SD2\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO\_QSPI\_SD2\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO\_QSPI\_SD1\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO\_QSPI\_SD1\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO\_QSPI\_SD1\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO\_QSPI\_SD1\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO\_QSPI\_SD0\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO\_QSPI\_SD0\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO\_QSPI\_SD0\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO\_QSPI\_SD0\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO\_QSPI\_SS\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO\_QSPI\_SS\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO\_QSPI\_SS\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO\_QSPI\_SS\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO\_QSPI\_SCLK\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO\_QSPI\_SCLK\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO\_QSPI\_SCLK\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO\_QSPI\_SCLK\_LEVEL\_LOW |  | RW | 0x0 |

#### [IO\_QSPI](#_bookmark70):IN1\_INTS寄存器

335号桌INTS寄存器

表336。 DORMANT\_唤醒\_内部

E登记册

**偏移**:0x48

描述

proc1强制屏蔽后的状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO\_QSPI\_SD3\_EDGE\_HIGH |  | RO | 0x0 |
| 22 | GPIO\_QSPI\_SD3\_EDGE\_LOW |  | RO | 0x0 |
| 21 | GPIO\_QSPI\_SD3\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO\_QSPI\_SD3\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO\_QSPI\_SD2\_EDGE\_HIGH |  | RO | 0x0 |
| 18 | GPIO\_QSPI\_SD2\_EDGE\_LOW |  | RO | 0x0 |
| 17 | GPIO\_QSPI\_SD2\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO\_QSPI\_SD2\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO\_QSPI\_SD1\_EDGE\_HIGH |  | RO | 0x0 |
| 14 | GPIO\_QSPI\_SD1\_EDGE\_LOW |  | RO | 0x0 |
| 13 | GPIO\_QSPI\_SD1\_LEVEL\_HIGH |  | RO | 0x0 |
| 12 | GPIO\_QSPI\_SD1\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO\_QSPI\_SD0\_EDGE\_HIGH |  | RO | 0x0 |
| 10 | GPIO\_QSPI\_SD0\_EDGE\_LOW |  | RO | 0x0 |
| 9 | GPIO\_QSPI\_SD0\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO\_QSPI\_SD0\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO\_QSPI\_SS\_EDGE\_HIGH |  | RO | 0x0 |
| 6 | GPIO\_QSPI\_SS\_EDGE\_LOW |  | RO | 0x0 |
| 5 | GPIO\_QSPI\_SS\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO\_QSPI\_SS\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO\_QSPI\_SCLK\_EDGE\_HIGH |  | RO | 0x0 |
| 2 | GPIO\_QSPI\_SCLK\_EDGE\_LOW |  | RO | 0x0 |
| 1 | GPIO\_QSPI\_SCLK\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO\_QSPI\_SCLK\_LEVEL\_LOW |  | RO | 0x0 |

#### [IO\_QSPI](#_bookmark70):DORMANT\_WAKE\_INTE寄存器

**偏移**:0x4c

描述

DORMANT\_唤醒中断启用

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO\_QSPI\_SD3\_EDGE\_HIGH |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 22 | GPIO\_QSPI\_SD3\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO\_QSPI\_SD3\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO\_QSPI\_SD3\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO\_QSPI\_SD2\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO\_QSPI\_SD2\_EDGE\_LOW |  | RW | 0x0 |
| 17 | GPIO\_QSPI\_SD2\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO\_QSPI\_SD2\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO\_QSPI\_SD1\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO\_QSPI\_SD1\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO\_QSPI\_SD1\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO\_QSPI\_SD1\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO\_QSPI\_SD0\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO\_QSPI\_SD0\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO\_QSPI\_SD0\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO\_QSPI\_SD0\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO\_QSPI\_SS\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO\_QSPI\_SS\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO\_QSPI\_SS\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO\_QSPI\_SS\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO\_QSPI\_SCLK\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO\_QSPI\_SCLK\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO\_QSPI\_SCLK\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO\_QSPI\_SCLK\_LEVEL\_LOW |  | RW | 0x0 |

337号桌 DORMANT\_唤醒\_内部

F寄存器

#### [IO\_QSPI](#_bookmark70):DORMANT\_WAKE\_INTF寄存器

**偏移**:0x50

描述

DORMANT\_唤醒的力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO\_QSPI\_SD3\_EDGE\_HIGH |  | RW | 0x0 |
| 22 | GPIO\_QSPI\_SD3\_EDGE\_LOW |  | RW | 0x0 |
| 21 | GPIO\_QSPI\_SD3\_LEVEL\_HIGH |  | RW | 0x0 |
| 20 | GPIO\_QSPI\_SD3\_LEVEL\_LOW |  | RW | 0x0 |
| 19 | GPIO\_QSPI\_SD2\_EDGE\_HIGH |  | RW | 0x0 |
| 18 | GPIO\_QSPI\_SD2\_EDGE\_LOW |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 17 | GPIO\_QSPI\_SD2\_LEVEL\_HIGH |  | RW | 0x0 |
| 16 | GPIO\_QSPI\_SD2\_LEVEL\_LOW |  | RW | 0x0 |
| 15 | GPIO\_QSPI\_SD1\_EDGE\_HIGH |  | RW | 0x0 |
| 14 | GPIO\_QSPI\_SD1\_EDGE\_LOW |  | RW | 0x0 |
| 13 | GPIO\_QSPI\_SD1\_LEVEL\_HIGH |  | RW | 0x0 |
| 12 | GPIO\_QSPI\_SD1\_LEVEL\_LOW |  | RW | 0x0 |
| 11 | GPIO\_QSPI\_SD0\_EDGE\_HIGH |  | RW | 0x0 |
| 10 | GPIO\_QSPI\_SD0\_EDGE\_LOW |  | RW | 0x0 |
| 9 | GPIO\_QSPI\_SD0\_LEVEL\_HIGH |  | RW | 0x0 |
| 8 | GPIO\_QSPI\_SD0\_LEVEL\_LOW |  | RW | 0x0 |
| 7 | GPIO\_QSPI\_SS\_EDGE\_HIGH |  | RW | 0x0 |
| 6 | GPIO\_QSPI\_SS\_EDGE\_LOW |  | RW | 0x0 |
| 5 | GPIO\_QSPI\_SS\_LEVEL\_HIGH |  | RW | 0x0 |
| 4 | GPIO\_QSPI\_SS\_LEVEL\_LOW |  | RW | 0x0 |
| 3 | GPIO\_QSPI\_SCLK\_EDGE\_HIGH |  | RW | 0x0 |
| 2 | GPIO\_QSPI\_SCLK\_EDGE\_LOW |  | RW | 0x0 |
| 1 | GPIO\_QSPI\_SCLK\_LEVEL\_HIGH |  | RW | 0x0 |
| 0 | GPIO\_QSPI\_SCLK\_LEVEL\_LOW |  | RW | 0x0 |

338号桌 DORMANT\_唤醒\_内部

名册

#### [IO\_QSPI](#_bookmark70):DORMANT\_WAKE\_INTS寄存器

**偏移**:0x54

描述

DORMANT\_唤醒强制屏蔽后的休眠状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23 | GPIO\_QSPI\_SD3\_EDGE\_HIGH |  | RO | 0x0 |
| 22 | GPIO\_QSPI\_SD3\_EDGE\_LOW |  | RO | 0x0 |
| 21 | GPIO\_QSPI\_SD3\_LEVEL\_HIGH |  | RO | 0x0 |
| 20 | GPIO\_QSPI\_SD3\_LEVEL\_LOW |  | RO | 0x0 |
| 19 | GPIO\_QSPI\_SD2\_EDGE\_HIGH |  | RO | 0x0 |
| 18 | GPIO\_QSPI\_SD2\_EDGE\_LOW |  | RO | 0x0 |
| 17 | GPIO\_QSPI\_SD2\_LEVEL\_HIGH |  | RO | 0x0 |
| 16 | GPIO\_QSPI\_SD2\_LEVEL\_LOW |  | RO | 0x0 |
| 15 | GPIO\_QSPI\_SD1\_EDGE\_HIGH |  | RO | 0x0 |
| 14 | GPIO\_QSPI\_SD1\_EDGE\_LOW |  | RO | 0x0 |
| 13 | GPIO\_QSPI\_SD1\_LEVEL\_HIGH |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 12 | GPIO\_QSPI\_SD1\_LEVEL\_LOW |  | RO | 0x0 |
| 11 | GPIO\_QSPI\_SD0\_EDGE\_HIGH |  | RO | 0x0 |
| 10 | GPIO\_QSPI\_SD0\_EDGE\_LOW |  | RO | 0x0 |
| 9 | GPIO\_QSPI\_SD0\_LEVEL\_HIGH |  | RO | 0x0 |
| 8 | GPIO\_QSPI\_SD0\_LEVEL\_LOW |  | RO | 0x0 |
| 7 | GPIO\_QSPI\_SS\_EDGE\_HIGH |  | RO | 0x0 |
| 6 | GPIO\_QSPI\_SS\_EDGE\_LOW |  | RO | 0x0 |
| 5 | GPIO\_QSPI\_SS\_LEVEL\_HIGH |  | RO | 0x0 |
| 4 | GPIO\_QSPI\_SS\_LEVEL\_LOW |  | RO | 0x0 |
| 3 | GPIO\_QSPI\_SCLK\_EDGE\_HIGH |  | RO | 0x0 |
| 2 | GPIO\_QSPI\_SCLK\_EDGE\_LOW |  | RO | 0x0 |
| 1 | GPIO\_QSPI\_SCLK\_LEVEL\_HIGH |  | RO | 0x0 |
| 0 | GPIO\_QSPI\_SCLK\_LEVEL\_LOW |  | RO | 0x0 |

* + - 1. Pad Control-用户银行

用户存储区控制寄存器从基址0x4001c000（在SDK中定义为PADS\_BANK0\_BASE）开始。

*表339。PADS\_BANK0列表*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [电压选择](#_bookmark85) | 电压选择。每库控制 |
| 0x04 | [GPIO0](#_bookmark86) | 填充控制寄存器 |
| 0x08 | [GPIO1](#_bookmark86) | 填充控制寄存器 |
| 0x0C | [GPIO2](#_bookmark86) | 填充控制寄存器 |
| 0x10 | [GPIO3](#_bookmark86) | 填充控制寄存器 |
| 0x14 | [GPIO4](#_bookmark86) | 填充控制寄存器 |
| 0x18 | [GPIO5](#_bookmark86) | 填充控制寄存器 |
| 0x1c | [GPIO6](#_bookmark86) | 填充控制寄存器 |
| 0x20 | [GPIO7](#_bookmark86) | 填充控制寄存器 |
| 0x24 | [GPIO8](#_bookmark86) | 填充控制寄存器 |
| 0x28 | [GPIO9](#_bookmark86) | 填充控制寄存器 |
| 0x2c | [公司简介](#_bookmark86) | 填充控制寄存器 |
| 0x30 | [GPIO11](#_bookmark86) | 填充控制寄存器 |
| 0x34 | [GPIO12](#_bookmark86) | 填充控制寄存器 |
| 0x38 | [GPIO13](#_bookmark86) | 填充控制寄存器 |
| 0x3c | [GPIO14](#_bookmark86) | 填充控制寄存器 |
| 0x40 | [GPIO15](#_bookmark86) | 填充控制寄存器 |
| 0x44 | [公司简介](#_bookmark86) | 填充控制寄存器 |

寄存器

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x48 | [GPIO 17](#_bookmark86) | 填充控制寄存器 |
| 0x4c | [GPIO18](#_bookmark86) | 填充控制寄存器 |
| 0x50 | [GPIO19](#_bookmark86) | 填充控制寄存器 |
| 0x54 | [公司简介](#_bookmark86) | 填充控制寄存器 |
| 0x58 | [GPIO21](#_bookmark86) | 填充控制寄存器 |
| 0x5c | [GPIO22](#_bookmark86) | 填充控制寄存器 |
| 0x60 | [GPIO23](#_bookmark86) | 填充控制寄存器 |
| 0x64 | [GPIO24](#_bookmark86) | 填充控制寄存器 |
| 0x68 | [公司简介](#_bookmark86) | 填充控制寄存器 |
| 0x6c | [GPIO26](#_bookmark86) | 填充控制寄存器 |
| 0x70 | [GPIO27](#_bookmark86) | 填充控制寄存器 |
| 0x74 | [GPIO28](#_bookmark86) | 填充控制寄存器 |
| 0x78 | [GPIO29](#_bookmark86) | 填充控制寄存器 |
| 0x7c | [SWCLK](#_bookmark87) | 填充控制寄存器 |
| 0x80 | [社署](#_bookmark88) | 填充控制寄存器 |

340号桌电压选择

寄存器

341号表GPIO 0，GPIO 1，.，GPIO 28，

GPIO 29寄存器

#### [PADS\_BANK0](#_bookmark84):VOLTAGE\_SELECT寄存器

**偏移**:0x00

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - |
| 0 | 电压选择。每库控制  0x0→将电压设置为3.3V（DVDD>= 2V5）0x1→将电压设置为1.8V（DVDD =1V8） | RW | 0x0 |

#### [PADS\_BANK0](#_bookmark84):GPIO 0，GPIO 1，...，GPIO 28，GPIO 29寄存器

**偏移**:0x04、0x08、...、0x74、0x78

描述

填充控制寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7 | OD | 输出禁用。优先于外围设备的输出使能 | RW | 0x0 |
| 6 | IE | 输入使能 | RW | 0x1 |
| 5:4 | 驱动 | 驱动力。  0x0→2mA  0x1→4mA  0x2→8mA  0x3→12mA | RW | 0x1 |
| 3 | PUE | 上拉使能 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 2 | PDE | 下拉使能 | RW | 0x1 |
| 1 | 施密特 | 使能施密特触发器 | RW | 0x1 |
| 0 | 灵活 | 压摆率控制。1 =快，0 =慢 | RW | 0x0 |

342号桌SWCLK

寄存器

343号桌。社会福利署登记册

#### [PADS\_BANK0](#_bookmark84):SWCLK寄存器

**偏移**:0x7c

描述

填充控制寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7 | OD | 输出禁用。优先于外围设备的输出使能 | RW | 0x1 |
| 6 | IE | 输入使能 | RW | 0x1 |
| 5:4 | 驱动 | 驱动力。  0x0→2mA  0x1→4mA  0x2→8mA  0x3→12mA | RW | 0x1 |
| 3 | PUE | 上拉使能 | RW | 0x1 |
| 2 | PDE | 下拉使能 | RW | 0x0 |
| 1 | 施密特 | 使能施密特触发器 | RW | 0x1 |
| 0 | 灵活 | 压摆率控制。1 =快，0 =慢 | RW | 0x0 |

#### [PADS\_BANK0](#_bookmark84):SWD寄存器

**偏移**:0x80

描述

填充控制寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7 | OD | 输出禁用。优先于外围设备的输出使能 | RW | 0x0 |
| 6 | IE | 输入使能 | RW | 0x1 |
| 5:4 | 驱动 | 驱动力。  0x0→2mA  0x1→4mA  0x2→8mA  0x3→12mA | RW | 0x1 |
| 3 | PUE | 上拉使能 | RW | 0x1 |
| 2 | PDE | 下拉使能 | RW | 0x0 |
| 1 | 施密特 | 使能施密特触发器 | RW | 0x1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 0 | 灵活 | 压摆率控制。1 =快，0 =慢 | RW | 0x0 |

* + - 1. 垫控制-QSPI银行

344号桌PADS\_QSPI寄存器列表

345号桌电压选择

寄存器

346号桌GPIO\_QSPI\_SCLK

寄存器

QSPI存储体焊盘控制寄存器从基址0x40020000（在SDK中定义为PADS\_QSPI\_BASE）开始。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [电压选择](#_bookmark91) | 电压选择。每库控制 |
| 0x04 | [GPIO\_QSPI\_SCLK](#_bookmark92) | 填充控制寄存器 |
| 0x08 | [GPIO\_QSPI\_SD0](#_bookmark93) | 填充控制寄存器 |
| 0x0C | [GPIO\_QSPI\_SD1](#_bookmark93) | 填充控制寄存器 |
| 0x10 | [GPIO\_QSPI\_SD2](#_bookmark93) | 填充控制寄存器 |
| 0x14 | [GPIO\_QSPI\_SD3](#_bookmark93) | 填充控制寄存器 |
| 0x18 | [GPIO\_QSPI\_SS](#_bookmark94) | 填充控制寄存器 |

#### [PADS\_QSPI](#_bookmark90):VOLTAGE\_SELECT寄存器

**偏移**:0x00

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - |
| 0 | 电压选择。每库控制  0x0→将电压设置为3.3V（DVDD>= 2V5）0x1→将电压设置为1.8V（DVDD =1V8） | RW | 0x0 |

#### [PADS\_QSPI](#_bookmark90):GPIO\_QSPI\_SCLK寄存器

**偏移**:0x04

描述

填充控制寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7 | OD | 输出禁用。优先于外围设备的输出使能 | RW | 0x0 |
| 6 | IE | 输入使能 | RW | 0x1 |
| 5:4 | 驱动 | 驱动力。  0x0→2mA  0x1→4mA  0x2→8mA  0x3→12mA | RW | 0x1 |
| 3 | PUE | 上拉使能 | RW | 0x0 |
| 2 | PDE | 下拉使能 | RW | 0x1 |
| 1 | 施密特 | 使能施密特触发器 | RW | 0x1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 0 | 灵活 | 压摆率控制。1 =快，0 =慢 | RW | 0x0 |

347号桌GPIO\_QSPI\_SD0、GPIO\_QSPI\_SD1、GPIO\_QSPI\_SD2、GPIO\_QSPI\_SD3

寄存器

表348.GPIO\_QSPI\_SS

寄存器

#### [PADS\_QSPI](#_bookmark90):GPIO\_QSPI\_SD0、GPIO\_QSPI\_SD1、GPIO\_QSPI\_SD2，

GPIO\_QSPI\_SD3寄存器

**偏移**:0x08、0x0c、0x10、0x14

描述

填充控制寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7 | OD | 输出禁用。优先于外围设备的输出使能 | RW | 0x0 |
| 6 | IE | 输入使能 | RW | 0x1 |
| 5:4 | 驱动 | 驱动力。  0x0→2mA  0x1→4mA  0x2→8mA  0x3→12mA | RW | 0x1 |
| 3 | PUE | 上拉使能 | RW | 0x0 |
| 2 | PDE | 下拉使能 | RW | 0x0 |
| 1 | 施密特 | 使能施密特触发器 | RW | 0x1 |
| 0 | 灵活 | 压摆率控制。1 =快，0 =慢 | RW | 0x0 |

#### [PADS\_QSPI](#_bookmark90):GPIO\_QSPI\_SS寄存器

**偏移**:0x18

描述

填充控制寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7 | OD | 输出禁用。优先于外围设备的输出使能 | RW | 0x0 |
| 6 | IE | 输入使能 | RW | 0x1 |
| 5:4 | 驱动 | 驱动力。  0x0→2mA  0x1→4mA  0x2→8mA  0x3→12mA | RW | 0x1 |
| 3 | PUE | 上拉使能 | RW | 0x1 |
| 2 | PDE | 下拉使能 | RW | 0x0 |
| 1 | 施密特 | 使能施密特触发器 | RW | 0x1 |
| 0 | 灵活 | 压摆率控制。1 =快，0 =慢 | RW | 0x0 |

* 1. Sysinfo
     1. 概述

sysinfo块包含系统信息。第一个寄存器包含芯片ID，它允许程序员知道哪个版本的芯片软件正在运行。第二个寄存器在器件上的读数始终为1

* + 1. 登记册一览表

349号桌SYSINFO寄存器列表

350。CHIP\_ID寄存器

351号表平台

寄存器

sysinfo寄存器从基址0x40000000（在SDK中定义为SYSINFO\_BASE）开始。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [芯片ID](#_bookmark97) | JEDECJEP-106兼容芯片标识符。 |
| 0x04 | [平台](#_bookmark98) | 站台登记。允许软件知道它在什么环境 |
| 0x40 | [GITREF\_RP2040](#_bookmark99) | Git hash的chip源码。用于识别芯片版本。 |

#### [SYSINFO](#_bookmark96):CHIP\_ID寄存器

**偏移**:0x00

描述

JEDECJEP-106兼容芯片标识符。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:28 | 修订 |  | RO | - |
| 27:12 | 部分 |  | RO | - |
| 11:0 | 制造商 |  | RO | - |

#### [SYSINFO](#_bookmark96):PLATFORM注册

**偏移量**:0x 04

描述

站台登记。 允许软件知道它在什么环境中运行。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:2 | Reserved. | - | - | - |
| 1 | ASIC |  | RO | 0x0 |
| 0 | FPGA |  | RO | 0x0 |

#### [SYSINFO](#_bookmark96):GITREF\_RP2040寄存器

**偏移**:0x40

352号桌GITREF\_RP2040

寄存器

表353. SYSCFG寄存器列表

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | Git hash的chip源码。用于识别芯片版本。 | RO | - |

* 1. Syscfg
     1. 概述

系统配置块控制各种芯片设置，包括:

* NMI（不可屏蔽）掩码，用于拾取生成NMI的源
* 处理器配置
  + DAP实例ID（用于更改SWD在调试中与核心通信时使用的地址
  + 处理器状态（如果处理器暂停，这在调试中可能很有用
* 处理器IO配置
  + 输入同步器控制（允许旁路输入同步器，以减少时钟同步时的延迟
* 调试控制
  + 提供从芯片内部控制SWD接口的能力这意味着核心0可以调试核心1，这可能使调试连接更容易。
* 内存断电（如果不使用，可以关闭每个内存，以节省少量额外电量）。
  + 1. 登记册一览表

系统配置寄存器从基址0x40004000（在SDK中定义为SYSCFG\_BASE）开始。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [00\_NMI\_MASK](#_bookmark101) | 处理器内核0 NMI源掩码 |
| 0x04 | [NMI\_MASK](#_bookmark102) | 处理器内核1 NMI源掩码 |
| 0x08 | [程序配置](#_bookmark103) | 处理器配置 |
| 0x0C | [PROC\_IN\_SYNC\_BYPASS](#_bookmark104) | 对于每个位，如果为1，则旁路GPIO之间的输入同步器  和GPIO输入寄存器。输入同步器应  通常不被绕过，以避免将可重构性注入处理器。  如果你觉得勇敢，你可以绕过，以节省两个周期的输入  延迟。此寄存器适用于GPIO 0. 29。 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x10 | [PROC\_IN\_SYNC\_BYPASS\_HI](#_bookmark105) | 对于每个位，如果为1，则旁路GPIO之间的输入同步器  和GPIO输入寄存器。输入同步器应  通常不可绕过，以避免将亚稳态注入处理器。  如果你觉得勇敢，你可以绕过，以节省两个周期的输入  延迟。此寄存器适用于GPIO 30.35（QSPI IO）。 |
| 0x14 | [DBGFORCE](#_bookmark106) | 直接控制任一处理器的SWD调试端口 |
| 0x18 | [MEMPOWERDOWN](#_bookmark107) | 控制内存断电 设为高电平，关闭存储器。  使用时要格外小心 |

354号桌00\_NMI\_MASK

寄存器

355号桌NMI\_MASK

寄存器

356号桌程序配置

寄存器

#### [SYSCFG](#_bookmark100):PROC0\_NMI\_MASK寄存器

**偏移**:0x00

描述

处理器内核0 NMI源掩码

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 将一个位设为高电平以从该IRQ启用NMI | RW | 0x00000000 |

#### [SYSCFG](#_bookmark100):MASK 1\_NMI\_MASK寄存器

**偏移量**:0x 04

描述

处理器内核1 NMI源掩码

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 将一个位设为高电平以从该IRQ启用NMI | RW | 0x00000000 |

#### [SYSCFG](#_bookmark100):PROC\_CONFIG寄存器

**偏移**:0x08

描述

处理器配置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:28 | 产品1\_DAP\_INSTID | 配置proc1 DAP实例ID。  建议在多芯片环境中需要调试访问权限之前不要更改此设置  警告:不要设置为15，因为这是为RescueDP保留的 | RW | 0x1 |
| 27:24 | PROC0\_DAP\_INSTID | 配置proc0 DAP实例ID。  建议在多芯片环境中需要调试访问权限之前不要更改此设置  警告:不要设置为15，因为这是为RescueDP保留的 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 23:2 | Reserved. | - | - | - |
| 1 | 第一章\_HALTED | 指示proc1已停止 | RO | 0x0 |
| 0 | HALTED | 指示proc0已停止 | RO | 0x0 |

357号桌PROC\_IN\_SYNC\_BYPA

SS寄存器

358号桌PROC\_IN\_SYNC\_BYPA

SS\_HI寄存器

359号桌DBGFORCE

寄存器

#### [SYSCFG](#_bookmark100):PROC\_IN\_SYNC\_BYPASS寄存器

**偏移**:0x0c

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:30 | Reserved. | - | - |
| 29:0 | 对于每个位，如果为1，则旁路该GPIO与SIO中GPIO输入寄存器之间的输入同步器输入同步器应  通常不可绕过，以避免将亚稳态注入处理器。  如果你觉得勇敢，你可以绕过，以节省两个周期的输入延迟。此寄存器适用于GPIO 0. 29。 | RW | 0x00000000 |

#### [SYSCFG](#_bookmark100):PROC\_IN\_SYNC\_BYPASS\_HI寄存器

**偏移**:0x10

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:6 | Reserved. | - | - |
| 5:0 | 对于每个位，如果为1，则旁路该GPIO与SIO中GPIO输入寄存器之间的输入同步器输入同步器应  通常不可绕过，以避免将亚稳态注入处理器。  如果你觉得勇敢，你可以绕过，以节省两个周期的输入延迟。此寄存器适用于GPIO 30.35（QSPI IO）。 | RW | 0x00 |

#### [SYSCFG](#_bookmark100):DBGFORCE注册

**偏移**:0x14

描述

直接控制任一处理器的SWD调试端口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7 | 联系我们 | 将处理器1调试端口连接到syscfg控件，并将其从外部SWD焊盘断开。 | RW | 0x0 |
| 6 | 时钟1\_SWCLK | 直接驱动处理器1 SWCLK（如果设置了ATT1\_ATTACH） | RW | 0x1 |
| 5 | 100\_SWDI | 直接驱动处理器1 SWDIO输入，如果设置了AD1\_ATTACH | RW | 0x1 |
| 4 | 公司简介 | 观察处理器1 SWDIO输出的值。 | RO | - |
| 3 | 联系我们 | 将处理器0调试端口连接到syscfg控件，并将其从外部SWD焊盘断开。 | RW | 0x0 |
| 2 | 时钟0\_SWCLK | 直接驱动处理器0 SWCLK（如果设置了ATT0\_ATTACH） | RW | 0x1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 1 | 公司简介 | 直接驱动处理器0 SWDIO输入，如果设置了AD0\_ATTACH | RW | 0x1 |
| 0 | 公司简介 | 观察处理器0 SWDIO输出的值。 | RO | - |

表360.MEMPOWERDOWN

寄存器

表361。TBMAN寄存器列表

表362。平台

寄存器

#### [SYSCFG](#_bookmark100):MEMPOWERDOWN寄存器

**偏移**:0x18

描述

控制内存断电设为高电平，关闭存储器。使用时要格外小心

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7 | ROM |  | RW | 0x0 |
| 6 | USB |  | RW | 0x0 |
| 5 | SRAM 5 |  | RW | 0x0 |
| 4 | SRAM 4 |  | RW | 0x0 |
| 3 | SRAM 3 |  | RW | 0x0 |
| 2 | SRAM 2存储器 |  | RW | 0x0 |
| 1 | 静态随机存取存储器1 |  | RW | 0x0 |
| 0 | SRAM0 |  | RW | 0x0 |

* 1. TBMAN

TBMAN是指测试平台管理器，用于芯片开发模拟过程中验证设计。在这些模拟过程中，TBMAN允许在RP 2040上运行的软件控制测试台和模拟环境。 在真实芯片上，它除了提供单个PLATFORM寄存器以指示这是真实芯片之外没有任何效果。 该PLATFORM功能在sysinfo（[第2.20节](#_bookmark95)）寄存器中重复。

* + 1. 寄存器列表

TBMAN寄存器从基址0x4006c000（在SDK中定义为TBMAN\_BASE）开始。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x0 | PLATFORM | 表示使用的平台 |

#### [TBMAN](#_bookmark108):PLATFORM注册

**偏移**:0x0

描述

表示使用的平台

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:2 | Reserved. | - | - | - |

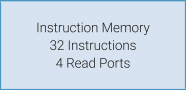
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 1 | FPGA | 表示平台为FPGA | RO | 0x0 |
| 0 | ASIC | 表示平台是ASIC | RO | 0x1 |

第3章 PIO

* 1. 概述

RP 2040中有2个相同的PIO模块。每个PIO块都有专用的连接到总线结构、GPIO和中断控制器。单个PIO块的示意图如[图38](#_bookmark111)所示。

*图38. PIO块级图。 有两个PIO块，每个块有四个状态机。 四个状态机同时执行来自共享指令存储器的程序。 FIFO数据队列缓冲在PIO和系统之间传输的数据。 GPIO映射逻辑允许每个状态机观察和操作多达30个GPIO。*



可编程输入/输出块（PIO）是一种通用的硬件接口。它可以支持多种IO标准，包括:

* 8080和6800并行总线
* I2c
* 3-pinI2S
* SDIO
* SPI、DSPI、QSPI
* UART
* DVI或VGA（通过电阻DAC）

PIO是可编程的，就像处理器一样。有两个PIO块，每个块有四个状态机，可以独立地执行顺序程序来操作GPIO和传输数据。与通用处理器不同，PIO状态机是高度专用于IO的，重点是确定性、精确定时以及与固定功能硬件的紧密集成。每个状态机配备有:

* 两个32位移位寄存器-任意方向，任意移位计数
* 两个32位暂存寄存器
* 每个方向（TX/RX）4×32位总线FIFO，可重新配置为单方向8×32位
* 小数时钟分频器（16整数，8小数位）
* 灵活的GPIO映射
* DMA接口，来自系统DMA的持续吞吐量高达每时钟1个字
* IRQ标志设置/清除/状态

每个状态机及其支持硬件占用的硅面积与标准串行接口块（如SPI或I2C控制器）大致相同。然而，PIO状态机可以被动态地配置和重新配置以实现许多不同的接口。

使状态机以类似软件的方式可编程，而不是像CPLD那样完全可配置的逻辑结构，允许在相同的成本和功率包络中提供更多的硬件接口这也为那些希望通过直接编程而不是使用PIO库中的预制接口来充分利用PIO灵活性的人提供了更熟悉的编程模型和更简单的工具流程

PIO具有高性能和灵活性，这要归功于每个状态机内精心选择的一组固定功能硬件。当输出DPI时，PIO在48MHz系统时钟下运行时，在活动扫描线期间可以维持360Mbps。在本例中，一个状态机处理帧/扫描线定时并生成像素时钟，而另一个状态机处理像素数据并解包游程长度编码的扫描线。

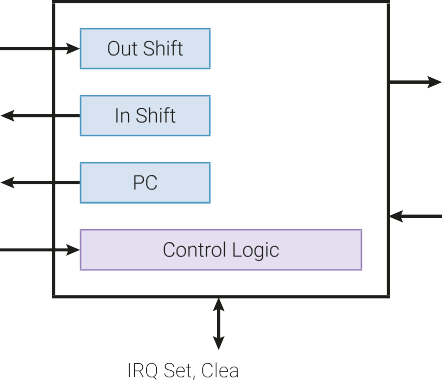
状态机的输入和输出映射到多达32个GPIO（对于RP 2040，限制为30个GPIO），并且所有状态机都可以独立地同时访问任何GPIO。例如，标准UART代码允许TX、RX、CTS和RTS成为任意四个GPIO，I2C允许SDA和SCL也是如此。可用的自由度取决于给定的PIO程序选择如何使用PIO的引脚映射资源，但至少可以通过一定数量的GPIO自由地向上或向下移动接口

* 1. 程序员模型

四个状态机从共享指令存储器执行系统软件将程序加载到此内存中PIO程序有多种来源:用户直接汇编、从PIO库中提取或由用户软件以编程方式生成。

从这一点上，状态机通常是自主的，系统软件通过DMA，中断和控制寄存器进行交互，就像RP 2040上的其他外设一样。对于更复杂的接口，PIO提供了一个小而灵活的原语集，允许系统软件更多地使用状态机控制流。

*图39. 状态机概述。 数据通过一对FIFO流入和流出。 状态机执行在这些FIFO、一组内部寄存器和引脚之间传输数据的程序。 时钟分频器可以将状态机的执行速度降低一个常数。*



* + 1. PIO项目

PIO状态机执行简短的二进制程序。

PIO库中提供了用于通用接口（如UART、SPI或I2C）的程序，因此在许多情况下，无需编写PIO程序。然而，PIO在直接编程时要灵活得多

设计者可能没有预见到的各种接口

PIO总共有9条指令:JMP、WAIT、IN、OUT、PUSH、PINK、MOV、IRQ和SET。 有关这些说明的详细信息，请参见第3.4节。

虽然PIO总共只有9条指令，但手工编辑PIO程序二进制文件是很困难的。PIO汇编是一种描述PIO程序的文本格式，其中每个命令对应于输出二进制中的一条指令。下面是一个PIO程序集的示例程序:

*Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.pio#L7-L12)第7-12*

7 .program squarewave

8 set pindirs, 1 ; Set pin to output

9 again:

10 set pins, 1 [1] ; Drive pin high and then delay for one cycle

11 set pins, 0 ; Drive pin low

12 jmp again ; Set PC to label `again`

PIO汇编器包含在SDK中，称为pioasm。此程序处理PIO汇编输入文本文件（可能包含多个程序），并写出准备使用的汇编程序。对于SDK，这些组装的程序以C头的形式发出，包含常量数组:有关更多信息，请参见[第3.3](#_bookmark113)节

* + 1. 控制流

在每个系统时钟周期，每个状态机获取、解码和执行一条指令。每个指令只占用一个周期，除非它显式地暂停（例如WAIT指令）。指令也可以在执行下一条指令之前插入最多31个周期的延迟，以帮助编写周期精确的程序。

程序计数器，或PC，指向指令存储器中在此周期执行的位置。一般来说，PC在每个周期递增1，在指令存储器的结尾处回绕跳转指令是一个例外，它显式地提供PC将采用的下一个值

我们的示例汇编程序（上面列出的.program squarewave）在实践中展示了这两个概念它驱动一个50/50占空比的方波到GPIO上，周期为四个周期。使用一些其他功能（例如，侧置），这可以低至两个循环。

**注意**

侧集是状态机*除了*其执行的指令的主要副作用之外还驱动少量GPIO的地方在[第3.5.1](#_bookmark122)中进行了详细描述。

系统对指令存储器具有只写访问权限，指令存储器用于加载程序:

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.c#L34-L38)第34-38

//将汇编后的程序直接加载到PIO的指令存储器中。

//每个PIO实例都有一个32插槽的指令存储器，所有4个状态

机器可以看到。 系统具有只写访问权限。

for（ int i =0; i count\_of（squarewave\_program\_instructions）;++i）

pio->instr\_order [i] = squarewave\_program\_instructions[i];

35

36

37

38

时钟分频器通过一个常数因子（表示为16.8定点小数）减慢状态机的执行速度使用上面的示例，如果编程为2.5的时钟分频，则方波将具有一个周期。这对于设置串行接口（如UART）的精确波特率很有用

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.c#L42-L47)第42-47

42

43

44

45

46

47

//配置状态机0在sysclk/2.5运行状态机可以

//每个时钟周期运行一条指令，但我们可以扩展它们的

//均匀减速以满足某个精确的频率目标，例如用于

//波特率。该寄存器具有16个整数除数位和8个

// fractional divisor bits.

pio->sm[0].clkdiv =（uint32\_t）（2.5f \*（116））;

上述代码片段是一个完整代码示例的一部分，该示例从GPIO 0（或我们可能选择映射的任何其他引脚）驱动一个12.5MHz方波 我们还可以使用引脚WAIT PIN指令将状态机的执行延迟一段时间，或者使用JMP PIN指令根据引脚状态进行分支，因此控制流可以根据引脚状态而变化。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.c#L51-L59)第51-59

51

52

53

54

55

56

57

58

59

//有五个引脚映射组（out、in、set、side-set、jmp引脚）

//它们被不同的指令或在不同的情况下使用。

//这里我们只使用SET指令。配置状态机0SET

//只影响GPIO 0;然后配置GPIO 0由PIO 0控制

//例如，处理器。

pio->sm[0]. pinch=

（1 <<PIO\_SM0\_PINCTRL\_SET\_COUNT\_LSB）|（0<< PIO\_SM0\_PINCTRL\_SET\_BASE\_LSB）;

gpio\_set\_function（0，GPIO\_FUNC\_PIO0）;

系统可以通过CTRL寄存器随时启动和停止每个状态机。多个状态机可以同时启动，PIO的确定性意味着它们可以保持完美的同步。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.c#L63-L67)第63-67

63

64

65

66

67

//设置状态机运行。PIO CTRL寄存器在

//PIO实例，因此可以启动/停止多个状态机

//同时我们使用寄存器的硬件原子集别名来

//在寄存器上不做读-修改-写的情况下使一位为HIGH。

hw\_set\_bits（pio-> ccumble，1（PIO\_CTRL\_SM\_ENABLE\_LSB +0））;

大多数指令从指令存储器执行，但也有其他来源，可以自由混合:

* 写入特殊配置寄存器（SMx\_R）的指令会立即执行，暂时中断其他执行。例如，写入SMxXOR的JMP指令将导致状态机启动

从不同的位置执行

* 可以使用MOVEXEC指令从寄存器执行指令
* 可以使用OUTEXEC指令从输出移位器执行指令

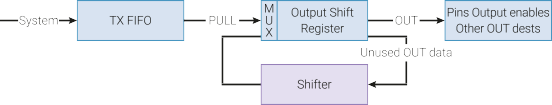
最后一种方法特别通用:指令可以嵌入到通过FIFO的数据流I2C示例使用它来嵌入，例如STOP和RESTART线路状态以及正常数据。 在MOV和OUTEXEC的情况下，MOV/OUT本身在一个周期中执行，而被执行者在下一个周期中执行。

* + 1. 寄存器

每个状态机拥有少量的内部寄存器。它们保存输入或输出数据，以及临时值，如循环计数器变量。

* + - 1. 输出移位寄存器（OSR）

图40. 输出移位寄存器（OSR）。 数据被一次打包出1.32位，未使用的数据由双向移位器回收。 一旦清空，OSR将从TX FIFO重新加载。



输出移位寄存器（OSR）在TX FIFO和引脚（或其他目的地，如暂存寄存器）之间保存和移位输出数据

* PULL指令:从TX FIFO中删除一个32位字，并放入OSR。
* OUT指令将数据从OSR转移到其他目的地，每次1.32位
* 随着数据移出，OSR填充为零
* 一旦达到某个总移位计数阈值（如果启用自动清零），状态机将根据OUT指令自动从FIFO重新填充OSR
* 移位方向可以是左/右，可由处理器通过配置寄存器进行配置

例如，要使数据流经FIFO并以每两个时钟一个字节的速率输出到引脚，请执行以下操作

1 .program pull\_example1

2 loop:

3 out pins, 8

4 public entry\_point:

5 pull

6 out pins, 8 [1]

7 out pins, 8 [1]

8 out pins, 8

9 jmp loop

Autopull（参见[第3.5.4](#_bookmark124)）允许硬件在大多数情况下自动重新填充OSR，如果状态机试图从空OSR中退出，则状态机会停止这有两个好处:

* + 在正确的时间从FIFO中显式拉取时没有指令花费
  + 更高的吞吐量:可以输出高达32位的每一个时钟周期，如果FIFO保持加满配置后，上述程序可以简化为以下，其行为相同:

1 .program pull\_example2

2

3 loop:

4 out pins, 8

5 public entry\_point:

6 jmp loop

程序包装（[第3.5.2节](#_bookmark123)）允许进一步简化，如果需要，每个系统时钟周期输出1个字节。

1 .program pull\_example3

2

3 public entry\_point:

4 .wrap\_target

5 out pins, 8 [1]

6 .wrap

* + - 1. 输入移位寄存器（ISR）

图41. 输入移位寄存器（ISR）。 数据一次输入1.32位，当前内容向左或向右移动以腾出空间。 一旦满了，内容就被写入RX FIFO。



* IN指令一次将1...32位移入寄存器。
* PUSH指令将ISR内容写入RX FIFO。
* 按下时，ISR被清除为全零
* 如果启用了自动推送，一旦达到某个移位阈值，状态机将自动推送ISR到IN
* 移位方向可由处理器通过配置寄存器进行配置

某些外设（如UART）必须从左移入以获得正确的位顺序，因为线序是LSB优先;但是，处理器可能期望得到的字节是右对齐的。这个问题可以通过特殊的null输入源来解决，它允许程序员在ISR中跟随数据移动一定数量的零

* + - 1. 移位型计数器

状态机记住总共有多少位通过OUT指令移出OSR，通过IN指令移入ISR该信息由一对硬件计数器（输出移位计数器和输入移位计数器）始终跟踪，每个硬件计数器能够保持从0到32（包括0和 32）的值每次移位操作时，相关计数器都会增加移位计数，直到最大值32（等于移位寄存器的宽度状态机可以配置为在计数器达到可配置阈值时执行某些操作

* + - * + OSR可以自动重新填充，一旦一些数量的位已被移出。参见[第3.5.4](#_bookmark124)
        + 一旦移入一定数量的位，ISR就可以自动清空参见[第3.5.4](#_bookmark124)
        + PUSH或PINTH指令可以分别以输入或输出移位计数器为条件

在PIO复位或CTRL\_SM\_RESTART置位时，输入移位计数器被清除为0（没有移入），输出移位计数器被初始化为32（没有剩余要移出;完全耗尽）。其他一些指令会影响移位计数器:

* + - * + 一个成功的PINGER会将输出移位计数器清零
        + 成功的PUSH会将输入移位计数器清零
        + MOVOSR，.（即任何写入OSR的MOV指令）将输出移位计数器清零
        + MOVISR，.（即任何写入ISR的MOV指令）将输入移位计数器清零
        + OUTISR，count将输入移位计数器设置为计数
      1. 便笺式寄存器

每个状态机都有两个32位内部暂存器，称为X和Y。它们被用作:

* + - * + IN/OUT/SET/MOV的源/目的地
        + 分支条件源

例如，假设我们想要为“1”数据位产生一个长脉冲，为“0”数据位产生一个短脉冲

 1 .program ws2812\_led

 2

 3 public entry\_point:

 4 pull

 5 set x, 23 ; Loop over 24 bits

 6 bitloop:

 7 set pins, 1 ; Drive pin high

 8 out y, 1 [5] ; Shift 1 bit out, and write it to y

 9 jmp !y skip ; Skip the extra delay if the bit was 0

10 nop [5]

11 skip:

12 set pins, 0 [5]

13 jmp x-- bitloop ; Jump if x nonzero, and decrement x

14 jmp entry\_point

这里，X用作循环计数器，Y用作临时变量，用于从OSR进行单个比特的分支此程序可用于驱动WS2812 LED接口，但也可以实现更紧凑的实现（只有3条指令）。

MOV允许使用暂存寄存器来保存/恢复移位寄存器，例如，如果你想重复移出相同的序列。

**注意**

一个更紧凑的WS2812示例（共4条指令）在[第3.6.2](#_bookmark129)中显示

* + - 1. 个fifo

每个状态机有一对4字深FIFO，一个用于从系统到状态机（TX）的数据传输，另一个用于状态机到系统（RX）的数据传输。TX FIFO由系统总线主控器（如处理器或DMA控制器）写入，RX FIFO由状态机写入FIFO将PIO状态机和系统总线的时序解耦，允许状态机在没有处理器干预的情况下运行更长的时间

FIFO还生成数据请求（DREQ）信号，其允许系统DMA控制器基于RX FIFO中的数据的存在或TX FIFO中的新数据的空间来对其读取/写入进行定步。这允许处理器建立一个长事务，可能涉及许多字节的数据，这将在没有进一步处理器干预的情况下继续进行。

通常，状态机只在一个方向上传输数据在这种情况下，SHFORTCTRL\_FJOIN选项可以将两个FIFO合并为一个仅沿一个方向的8条目FIFO这对于DPI等高带宽接口很有用

* + 1. 拖延

状态机可能会由于以下几个原因暂时暂停执行

* WAIT指令的条件尚未满足
* 当TX FIFO为空时，阻塞PUSH，或当RX FIFO为满时，阻塞PUSH
* 一种IRQWAIT指令，它已设置了IRQ标志，并等待该标志被清除
* 当自动清零被使能且OSR已经达到其移位阈值时的OUT指令
* 自动推送使能、ISR达到其移位阈值且RX FIFO已满时的IN指令

在这种情况下，程序计数器不会前进，状态机将在下一个周期继续执行此指令如果指令在下一条指令开始之前指定了一定数量的延迟周期，则这些延迟周期直到停止清除**之后**才开始

**注意**

侧置（[第3.5.1](#_bookmark122)）不受停顿的影响，并且始终发生在所附指令的第一个周期

* + 1. 引脚映射

PIO控制多达32个GPIO的输出电平和方向，并可以观察它们的输入电平。在每个系统时钟周期，每个状态机可以不执行以下操作，也可以执行以下操作之一或两者:

* 通过OUT或SET指令更改某些GPIO的级别或方向，或通过IN指令读取某些GPIO
* 通过侧置操作更改某些GPIO的级别或方向

这些操作中的每一个都在GPIO的四个连续范围之一上，每个范围的基数和计数都通过每个状态机的PINCTRL寄存器配置。每个OUT、SET、IN和侧置操作都有一个范围每个范围可以覆盖给定PIO块可访问的任何GPIO（在RP 2040上，这是30个用户GPIO），并且范围可以重叠。

对于每个GPIO输出（分别为电平和方向），PIO会考虑在该周期内可能发生的所有8次写入，并从编号最高的状态机应用写入如果同一状态机执行SET

/OUT和侧集同时位于同一GPIO上，则使用侧集。如果没有状态机写入此GPIO输出，则其值与前一周期相比不会发生变化

通常，每个状态机的输出都映射到一组不同的GPIO，从而实现某些外围接口。

* + 1. IRQ标志

IRQ标志是状态位，可以由状态机或系统设置或清除总共有8个:所有8个对所有状态机都是可见的，低4个也可以通过IRQ 0\_INTE和IRQ 1\_INTE控制寄存器屏蔽到PIO的中断请求线中

它们有两个主要用途:

* 断言来自状态机程序的系统级中断，并且可选地等待中断被确认
* 在两个状态机之间同步执行

状态机通过IRQ和WAIT指令与标志交互

* + 1. 状态机之间的交互

指令存储器被实现为1-写4-读寄存器文件，因此所有四个状态机都可以在同一周期读取指令，而不会停止。

有三种方法可以应用多状态机:

* 将多个状态机指向同一个程序
* 将多个状态机指向不同的程序
* 使用多个状态机来运行同一接口的不同部分，例如:DPI显示器的TX和RX端，或

状态机不能进行数据通信，但它们可以通过使用IRQ标志来彼此同步总共有8个标志（其中较低的四个可以被屏蔽用作系统IRQ），每个状态机可以使用IRQ指令设置或清除任何标志，并可以使用WAIT IRQ指令等待标志变为高电平或低电平。这允许状态机之间的周期精确同步。

* 1. PIO汇编程序（pioasm）

PIO汇编器解析PIO源文件，并输出准备包含在RP 2040应用程序中的汇编版本。这包括基于SDK构建的C和C++应用程序，以及在RP 2040 MicroPython端口上运行的Python程序。

本节简要介绍可在pioasm输入中使用的指令和说明关于如何使用pioasm，如何将其集成到SDK构建系统中，代码传递等扩展功能以及它可以产生的各种输出格式的更深入讨论，请参见**[Raspberry Pi Pico C/C++ SDK](https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf)**书籍。

* + 1. 指令

表363。外交指令

以下指令控制PIO程序的汇编

*.define*（*PUBLIC*）*<symbol> value>* 定义一个名为*<symbol>*的整数符号，其值为*<value>*（参见[第](#_bookmark116)

[3.3.2](#_bookmark116)）。如果这个*.define*出现在输入文件中的第一个程序之前，那么define对所有程序都是全局的，否则它对它所在的程序是局部如果指定*了PUBLIC*，则该符号将被发送到组装输出中以供用户代码使用对于SDK，这采取以下形式:

#define<program\_name>\_symbol>value用于程序符号或#define<symbol>value用于全局符号

*.program<name>* 启动一个新程序，名称为*<name>*。请注意，该名称用于

所以代码应该是字母数字/下划线，而不是以数字开头 程序将持续到另一个*.program*指令或源文件结束。PIO指令只允许在程序中使用

*.origin offset>* 可选指令，用于指定程序*必须*加载的PIO指令内存偏移量这通常用于必须在偏移量0处加载的程序，因为它们使用基于数据的JMP，并且（绝对）JMP目标仅存储在几位中。此指令在程序外无效

*.side\_set<count>（opt）（pindirs）* 如果存在*<*此指令*，则count>*指示要被

采用此外*，*可以指定opt来指示侧值>对于指令是可选的（请注意，除了*<计数>*位之外，这需要从可用于指令延迟的位中窃取额外的位最后，可以指定*pindir*以指示侧设置值应该应用于PINDIR而不是PIN。此指令仅在第一条指令之前的程序内有效

*.wrap\_target* 放置在指令之前，此指令指定由于程序包装而继续执行的指令。此指令在程序外无效，只能在程序内使用一次，如果未指定，则默认为程序的开始

*.wrap* 放置在指令之后，此指令指定在正常控制流（即，jmpwithfalsecondition，ornojmp），则程序会进行包装（to*.wrap\_target*指令）。此指令在程序外无效，只能在程序内使用一次，如果未指定，则默认为在最后一条程序指令之后。

*.lang\_opt<lang><name><option>* 指定与特定语言生成器相关的程序的选项。

(See[语言生成器](https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf#pioasm_language_generators)）。此指令在程序外无效

*.word value>* 将原始16位值存储为程序中的指令此指令在程序外无效。

* + 1. 值

364号表 在pioasm值，即<值>

表365。pioasm中的表达式

即<expression>

以下类型的值可用于定义整数或分支目标

|  |  |
| --- | --- |
| integer | 一个整数值，例如3或-7 |
| hex | 十六进制值，例如0xf |
| binary | 二进制值，例如0b1001 |
| symbol | 由.define定义的值（参见[[pioasm\_define]](#_bookmark114)） |
| <label> | 程序内标签的指令偏移量这在与JMP指令一起使用时最有意义（参见[第3.4.2节](#_bookmark119)） |
| ( <expression> ) | 要计算的表达式;参见[表达式](#_bookmark117)。注意括号是必要的。 |

* + 1. 表达式

表达式可以在pioasm值中自由使用。

|  |  |
| --- | --- |
| <expression>+expression> | 两个表达式的和 |
| <表达式>-表达式> | 两个表达式的区别 |
| <表达式>\* 表达式> | 两个表达式的乘法 |
| <表达式> /表达式> | 两个表达式的整数除法 |
| -<表达式> | 另一个表达式的否定 |
| ::<expression> | 另一个表达式的位逆 |
| <值> | 任何值（见[第3.3.2](#_bookmark116)） |

* + 1. 注释

行注释支持//或;

通过/\*和\*/支持C风格的块注释

* + 1. 标签

标签的形式为

<symbol>:

或

PUBLIC symbol>:

在一行的开头

**尖端**

标签实际上只是一个自动的.define，其值设置为当前程序指令偏移量。*PUBLIC*标签以与PUBLIC.define相同的方式向用户代码公开。

* + 1. 指令

所有pioasm指令都遵循一个共同的模式:

*<instruction>*（*side side\_set\_value>*）（*[delay\_value>]*）其中:

*<instruction>* 是一个汇编指令，将在下面的章节中详细介绍。(See第[3.4节](#_bookmark118)）

*<side\_set\_value>* 是在指令开始时应用于side\_set引脚的值（参见[第3.3.2节](#_bookmark116)）。请注意，通过side side\_set\_value>的side-set值的规则取决于程序的.side\_set（参见[[pioasm\_side\_set]](#_bookmark115)如果未指定.side\_set，则

<side\_set\_value>无效，如果指定了可选的边集管脚数，则side

<side\_set\_value>可以存在，并且如果指定了非可选的侧集管脚数量，则需要side side\_set\_value>*<side\_set\_value>*必须符合.side\_set指令中指定的边集位数

*<delay\_value>* 指定指令完成后要延迟的周期数delay\_value被指定为一个值（参见[第3.3.2节](#_bookmark116)），通常介于0和31之间（含0和31）（5位值），但是，当通过.side\_set启用sideset时，位数会减少（参见[[pioasm\_side\_set]](#_bookmark115)）指令。如果*<delay\_value>*不存在，则指令没有延迟

**注意**

pioasm指令名称、关键字和指令不区分大小写;在程序*集语法*中使用小写

因为这是SDK中使用的样式。

**注意**

逗号出现在下面的一些*程序集配置文件*部分，但完全是可选的，例如。 out pins，3可以写成out pins 3，jmp x--label可以写成jmp x--，label。下面的程序*集样式表*部分在每种情况下都使用第一种样式，因为这是SDK中使用的样式

* + 1. 伪指令

目前，pioasm提供了一个伪指令，作为方便:

nop Assembles tomovey， y.“无操作”，没有特别的副作用，但对于侧置操作或额外延迟是有用的车辆。

* 1. 指令集
     1. 总结

PIO指令是16位长，并且具有以下编码:

表366。PIO指令编码

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 比特: | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| JMP | 0 | 0 | 0 | 延迟/侧集 | | | | | 条件 | | | 地址 | | | | |
| WAIT | 0 | 0 | 1 | 延迟/侧集 | | | | | PoI | 源 | | 索引号 | | | | |
| IN | 0 | 1 | 0 | 延迟/侧集 | | | | | 源 | | | 位计数 | | | | |
| OUT | 0 | 1 | 1 | 延迟/侧集 | | | | | 目标 | | | 位计数 | | | | |
| PUSH | 1 | 0 | 0 | 延迟/侧集 | | | | | 0 | IfF | Blk | 0 | 0 | 0 | 0 | 0 |
| PULL | 1 | 0 | 0 | 延迟/侧集 | | | | | 1 | IfE | Blk | 0 | 0 | 0 | 0 | 0 |
| MOV | 1 | 0 | 1 | 延迟/侧集 | | | | | 目标 | | | Op | | 源 | | |
| IRQ | 1 | 1 | 0 | 延迟/侧集 | | | | | 0 | CLR | Wait | 索引号 | | | | |
| SET | 1 | 1 | 1 | 延迟/侧集 | | | | | 目标 | | | 数据 | | | | |

所有PIO指令在一个时钟周期内执行

5位延迟/侧集字段的功能取决于状态机的SIDESET\_SET配置:

* 最多5个LSB（5减去SIDESET\_CLK）对插入在此指令和下一个指令之间的空闲周期数进行
* 最多5个MSB，由SIDESET\_RESET设置，编码一个侧集（[第3.5.1](#_bookmark122)），它可以在主指令执行的同时在一些GPIO上断言一个常量
  + 1. JMP
       1. 编码

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 比特: | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| JMP | 0 | 0 | 0 | 延迟/侧集 | | | | | 条件 | | | 地址 | | | | |

* + - 1. 操作

如果条件为真，则将程序计数器设置为地址，否则不操作。

JMP上的延迟周期始终生效，无论Condition为真还是假，并且*在*Condition求值和程序计数器更新后发生

* + - * + 条件:

000:*（无条件）*:始终

001:X:划痕X零

010:X--:在递减之前，擦除X非零

011:Y:划痕Y零

100:Y--:在递减之前，擦除Y非零

101:X！= Y:划痕X不等于划痕Y

110:PIN:输入引脚上的分支

111:！OSRE:输出移位寄存器不为空

* + - * + 地址:跳转到的指令地址在指令编码中，这是PIO指令内存中的绝对地址

JMP PIN在由EXECCTRL\_JMP\_PIN选择的GPIO上分支，EXECCTRL\_JMP\_PIN是一个配置字段，可从状态机可见的最多32个GPIO输入中选择一个，与状态机的其他输入映射无关如果GPIO为高，则采取该分支

! OSRE将自上一次PINK以来移出的位与SHIFT CTRL\_PINK\_THRESH配置的移位计数阈值进行比较。这与autopull使用的阈值相同（[第3.5.4节](#_bookmark124)）。

JMP X和JMP Y始终分别递减暂存寄存器X或Y。递减不取决于暂存寄存器的当前值。转移以寄存器的*初始*值为条件，即在递减发生之前:如果寄存器初始为非零，则转移。

* + - 1. 汇编程序语法

*jmp*（*<cond>*）*<target>*

其中:

*<cond>* 是上面列出的可选条件（例如， ！x代表刮擦X零）。如果未指定条件代码，则

*<target>* 是一个程序标签或值（见[3.3.2节](#_bookmark116)），表示程序内的指令偏移量（第一条指令偏移量为0）。请注意，由于PIO JMP指令使用PIO指令内存中的绝对地址，因此需要在运行时根据程序加载偏移量调整JMP在使用SDK加载程序时，系统会为您处理此问题，但在对JMP指令进行编码以供OUT EXEC使用时，应特别小心

* + 1. WAIT
       1. 编码

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 比特: | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| WAIT | 0 | 0 | 1 | 延迟/侧集 | | | | | Pol | 源 | | 指数 | | | | |

* + - 1. 操作

拖延，直到满足某个条件。

像所有的暂停指令（[3.2.4节](#_bookmark112)），延迟周期开始后，指令*完成*。也就是说，如果存在任何延迟周期，则它们直到满足等待条件*之后*才开始计数

* + - * + 极性:

1:等待1。

0:等待0。

* + - * + 来源:等待什么值为:

00:GPIO:由索引选择的系统GPIO输入。这是一个绝对GPIO索引，不受状态机输入IO映射的影响

01:PIN:步进选择的输入引脚。首先应用此状态机的输入IO映射，然后应用Index

选择等待哪个映射位换句话说，通过将Index添加到

PINCTRL\_IN\_BASE配置，模32。

10:IRQ:索引选择PIO IRQ标志

11:保留

* + - * + 索引:要检查的引脚或位

WAITxIRQ的行为与其他WAIT源略有不同

* + - * + 如果Polarity为1，则在满足等待条件时，状态机将清除所选IRQ标志。
        + 标志索引以与IRQ索引字段相同的方式解码:如果MSB被设置，则通过对两个LSB进行模4加法，将状态机ID（0.3）添加到IRQ索引例如，具有标志的状态机2

值“0x11”将在标志3上等待，并且标志值“0x13”将在标志1上等待这允许多个运行相同程序的状态

**谨慎**

WAIT 1 IRQ x不应该与提供给中断控制器的IRQ标志一起使用，以避免与系统中断处理程序的竞争条件

**3.4.3.3. 汇编程序**

*wait <polarity> gpio <gpio\_num>*

*wait <polarity> pin <pin\_num>*

*wait <polarity> irq <irq\_num> ( rel )*

其中:

*<polarity>* 是指定极性（0或1）的值（参见[第3.3.2](#_bookmark116)）

*<pin\_num>* 是指定输入管脚号（由SM输入管脚映射映射）的值（参见[第3.3.2](#_bookmark116)）

*<gpio\_num>* 是指定实际GPIO引脚号的值（参见[第3.3.2](#_bookmark116)节

*<irq\_num>*（*rel*） 是一个值（参见[第3.3.2节](#_bookmark116)），指定要等待的irq号（0-7）。如果*rel*存在，则使用的实际irq数是通过将irq数（*irq\_num10*）的低两位替换为和（*irq\_num10*+*sm\_num10*）的低两位来计算的，其中*sm\_num10*是状态机号

* + 1. IN
       1. 编码

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 比特: | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IN | 0 | 1 | 0 | 延迟/侧集 | | | | | 源 | | | 位计数 | | | | |

* + - 1. 操作

将位计数位从源移位到输入移位寄存器（ISR）。通过以下方式为每个状态机配置移位方向:

SHIFT CTRL\_IN\_SHIFT键。此外，将输入移位计数增加Bitcount，在32处饱和。

* + - * + 资料来源:

000:引脚

001:X（暂存寄存器X）

010:Y（暂存器Y）

011:NULL（全零）

第100章:一个人

第101章:你是谁

第110章:ISR

第111章:一夜情

* + - * + Bit count:要移入ISR的位数1.32位，32被编码为00000。

如果启用自动推送，则IN也会在达到推送阈值（SHIFTCTRL\_PUSH\_THRESH）时将ISR内容推送到RX FIFO。无论是否发生自动推送，IN仍在一个周期内执行。如果自动推送发生时RX FIFO已满，则状态机将停止自动推送将ISR内容清除为全零，并清除输入移位计数。参见[第3.5.4节](#_bookmark124)。

IN始终使用源数据的最低有效位计数位。例如，如果PINCTRL\_IN\_BASE设置为5，则指令IN PINS，3将取引脚5、6和7的值，并将这些值移入ISR。首先，ISR向左或向右移动，为新的输入数据腾出空间，然后将输入数据复制到留下的间隙输入数据的位顺序与移位方向无关

NULL可用于转移ISR的内容。例如，UART首先接收LSB，因此必须向右移位。在8IN PINS，1指令之后，输入串行数据将占用ISR的位31... 24INNULL，24指令将移入24个零位，将输入数据对齐ISR位7. 0。或者，处理器或DMA可以从FIFO地址+3执行字节

* + - 1. 汇编程序

在<source>中，<bit\_count>

其中:

*<source>* 是上面指定的源之一。

*<bit\_count>* 是一个指定要移位的位数*的*值（参见[第3.3.2](#_bookmark116)）（有效范围1-32）

* + 1. OUT
       1. 编码

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 比特: | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| OUT | 0 | 1 | 1 | 延迟/侧集 | | | | | 目的地 | | | 位计数 | | | | |

* + - 1. 操作

从输出移位寄存器（OSR）中移出的移位位计数位，并将这些位写入目标。此外，将输出移位计数增加Bit count，在32处饱和。

* + - * + 目的地:

000:引脚

001:X（暂存寄存器X）

010:Y（暂存器Y）

011:NULL（丢弃数据）

第100章:一夜情

第101章:一夜情

110:ISR（也将ISR移位计数器设置为位计数）

111:EXEC（执行OSR移位数据作为指令）

* + - * + 位数:从OSR中移出的位数1.32位，32被编码为00000。

一个32位的值被写入目的地:低位计数位来自OSR，其余为零。如果SHIFTCTRL\_OUT\_SHIFTCTRL位于右侧，则此值为OSR的最低有效位计数位，否则为最高有效位。

PINS和PINDIRS使用OUT引脚映射，如[第3.5.6](#_bookmark128)所述。

如果启用了自动拉取，则在达到拉取阈值SHIFTCTRL\_PACK\_THRESH时，OSR将自动从TX FIFO重新填充。输出移位计数同时清零。在这种情况下，如果TX FIFO为空，则OUT将停止，否则仍在一个周期内执行具体信息见[第3.5.4](#_bookmark124)。

OUT EXEC允许将指令内联包含在FIFO数据流中。OUT本身在一个周期执行，而来自OSR的指令在下一个周期执行对于可以由该机制执行的指令的类型没有限制初始OUT上的延迟周期被忽略，但被执行者可以正常插入延迟周期。

OUTPC表现为无条件跳转到从OSR移出的地址。

* + - 1. **汇编程序***输出目的地>，位计数>*，其中:

*<目的地>* 是上面指定的目的地之一。

*<bit\_count>* 是一个指定要移位的位数*的*值（参见[第3.3.2](#_bookmark116)）（有效范围1-32）

* + 1. PUSH
       1. 编码

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 比特: | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PUSH | 1 | 0 | 0 | 延迟/侧集 | | | | | 0 | 森林论坛 | Blk | 0 | 0 | 0 | 0 | 0 |

* + - 1. 操作

将ISR的内容作为单个32位字推入RX FIFO把侦察监视系统清除为零

* + - * + IfFull:如果为1，则不执行任何操作，除非总输入移位计数已达到其阈值，SHIFTCTRL\_PUSH\_THRESH（与自动推送相同;参见[第3.5.4节](#_bookmark124)）。
        + 块:如果为1，则在RX FIFO已满时停止执行

PUSH IFFULL有助于使程序更紧凑，如autopush。如果启用自动推送，IN将在不适当的时间停止，则此功能非常有用，例如: 如果状态机在这一点上断言某个外部控制信号。

PIO汇编程序默认设置Block位如果未设置Block位，则PUSH不会在满RX FIFO时停止，而是立即继续执行下一条指令。发生这种情况时，FIFO状态和内容不变ISR仍然被清除为全零，并且设置FDEBUG\_RXSTALL标志（与阻塞PUSH或自动推送到全RXFIFO相同）以指示数据丢失。

* + - 1. 汇编程序语法

*推*（*满*）

*push*（*iffull*）block*push*（*iffull*）noblock其中:

*iffull* 相当于上面的IfFull==1。 即如果未指定，默认值为IfFull==0

*block* 等同于上面的Block==1。如果既未指定*block*也*未指定noblock*，则这是默认值

*noblock* 相当于上面的Block==0。

* + 1. PULL
       1. 编码

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 比特: | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PULL | 1 | 0 | 0 | 延迟/侧集 | | | | | 1 | Ife | Blk | 0 | 0 | 0 | 0 | 0 |

* + - 1. 操作

将TX FIFO中的32位字加载到OSR。

* + - * + IfEmpty:如果为1，则不执行任何操作，除非总输出移位计数已达到其阈值SHIFTCTRL\_PALTH\_THRESH（与自动清零相同;参见[第3.5.4节](#_bookmark124)）。
        + 阻塞:如果为1，则在TX FIFO为空时停止如果为0，则从空FIFO中拉出将临时X复制到OSR。

某些外设（UART、SPI等）在没有数据可用时应暂停，并在数据进入时拾取数据;其他外设（I2 S）应连续时钟，输出占位符或重复数据比停止时钟更好这可以通过Block参数来

在空FIFO上的非阻塞PSNR与MOV OSR，X具有相同的效果。程序可以使用适当的默认值预加载临时寄存器X，或者在每个PINDBNOBLOCK之后执行MOVX，OSR，以便回收最后一个有效的FIFO字，直到有新数据可用。

当TX FIFO为空时，如果带有自动清零功能的OUT在不适当的位置停止，则PADEIFEMPTY非常有用IfEmpty允许一些与autopull相同的程序简化-例如，消除外部循环计数器-但停顿发生在程序中的受控点

**注意**

当启用自动清零时，当OSR已满时，任何PADER指令都是空操作，因此PADER指令表现为屏障。OUT NULL，32可用于显式丢弃OSR内容。更多详情请参见[3.5.4.2部分](#_bookmark127)

* + - 1. 汇编程序语法

*pull*（*ifempty*）

*pull*（*ifempty*）block*pull*（*ifempty*）noblock其中:

*ifempty* 等同于上面的IfEmpty == 1即如果未指定，则默认值为IfEmpty == 0 *block* 等同于上面的Block==1。如果既没有*指定block*也没有指定 *noblock*，这是默认值noblock相当于上面的Block == 0

* + 1. MOV
       1. 编码

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 比特: | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| MOV | 1 | 0 | 1 | 延迟/侧集 | | | | | 目的地 | | | Op | | 源 | | |

* + - 1. 操作

将数据从源复制到目标。

* + - * + 目的地:

000:PINS（使用与OUT相同的引脚映射）

001:X（暂存寄存器X）

010:Y（暂存寄存器Y）

第011章:你是谁

100:EXEC（执行数据作为指令）

第101章:一夜情

110:ISR（输入移位计数器通过此操作复位为0，即为空）

111:OSR（输出移位计数器通过此操作复位为0，即（全部）

* + - * + 操作方式:

00:无

01:反相（按位补码）

第10章:反转

11:保留

* + - * + 资料来源:

000:PINS（使用与IN相同的引脚映射）

001:X

010:是

011:空

第100章:一个人

第101章:一夜情

第110章:ISR

第111章:一夜情

MOV PC导致无条件跳转。MOV EXEC具有与OUT EXEC相同的行为（[第3.4.5](#_bookmark120)），并允许寄存器内容作为指令执行 MOV本身在1个周期内执行，而Source中的指令在下一个周期执行。MOV EXEC上的延迟周期被忽略，但被执行者可以正常插入延迟周期。

STATUS源的值为全1或全0，具体取决于某些状态机状态，如FIFO满/空，由EXECCTRL\_STATUS\_SEL配置。

MOV可以通过Operation参数指定的有限方式操作传输的数据Invert将Destination中的每个位设置为Source中相应位的逻辑NOT，即1位变为0位，反之亦然。位反转将目标中的每个位*n*设置为源中的位31-*n*，假设位编号为0到31。

MOV dst，PINS使用IN引脚映射读取引脚，并将完整的32位值写入目标，而无需屏蔽。读取值的LSB是PINCTRL\_IN\_BASE所指示的引脚，每个连续位来自编号较高的引脚，在31之后绕回。

* + - 1. **汇编程序***将<目标>，*（*操作*）*<源>移动*到其中:

*<destination>* 是上面指定的目的地之一。

*<op>* 如果存在，则为:

! 或~表示NOT（注意:这始终是按位NOT）

\*用于位反转

*<source>* 是上面指定的源之一。

* + 1. IRQ
       1. 编码

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 比特: | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IRQ | 1 | 1 | 0 | 延迟/侧集 | | | | | 0 | CLR | 等 | 指数 | | | | |

* + - 1. 操作

设置或清除索引参数选择的IRQ标志

* + - * + 清除:如果为1，则清除Index选择的标志，而不是升起它。如果清除被设置，等待位不起作用。
        + 等待:如果为1，则暂停，直到升起的标志再次降低，例如，如果系统中断处理程序已确认该标志。
        + 联系我们

3个LSB指定IRQ索引（0-7）。 该IRQ标志将根据清除位进行设置/清除。

如果MSB被设置，则通过对两个LSB进行模4加法，将状态机ID（0.3）添加到IRQ索引例如，标志值为0x11的状态机2将引发标志3，标志值为0x13的状态机2将引发标志3。

升国旗1.

IRQ标志4-7只对状态机可见;IRQ标志0-3可以路由到系统级中断，在PIO的两条外部中断请求线中的任何一条上，由IRQ 0\_INTE和IRQ 1\_INTE配置。

模加法位允许“IRQ”和“WAIT”指令的相对寻址，用于同步运行相同程序的状态机。位2（第三LSB）不受此加法的影响。

如果设置为等待，则延迟周期直到等待时间结束后才开始

* + - 1. 汇编程序语法

*irq irq\_num>*（*rel*）

*irq set irq\_num>*（*rel*）

*irq nowait irq\_num>*（*rel*）*irq wait irq\_num>*（*rel*）*irq clear irq\_num>*（*rel*）其中:

*<irq\_num>*（*rel*） 是一个值（参见[第3.3.2节](#_bookmark116)），指定要等待的irq号（0-7）。如果*rel*存在，则使用的实际irq数是通过将irq数（*irq\_num10*）的低两位替换为和（*irq\_num10*+*sm\_num10*）的低两位来计算的，其中*sm\_num10*是状态机号

*irq* 意味着设置IRQ而不等待

*irqset* 也意味着设置IRQ而不等待

*irqnowait* 再次，意味着设置IRQ而不等待

*irqwait* 意味着设置IRQ并等待它被清除后再继续

*irqclear* 表示清除IRQ

* + 1. SET
       1. 编码

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 比特: | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SET | 1 | 1 | 1 | 延迟/侧集 | | | | | 目的地 | | | 数据 | | | | |

* + - 1. 操作

将立即值数据写入目标。

* + - * + 目得地:

000:引脚

001:X（暂存寄存器X）5个LSB设置为数据，所有其他LSB清零。

010:Y（暂存寄存器Y）5个LSB设置为数据，所有其他LSB清零。

第011章:你是谁

第100章:一夜情

第101章:你是谁

第110章:一个人

第111章:一夜情

* + - * + 数据:5位立即值，用于驱动到引脚或寄存器。

这可以用来断言控制信号，如时钟或芯片选择，或初始化循环计数器。由于数据的大小为5位，因此临时寄存器可以设置为0-31之间的值，这对于32次迭代循环来说是足够

SET和OUT到引脚的映射是独立配置的。它们可以被映射到不同的位置，例如，如果一个引脚将被用作时钟信号，而另一个引脚用于数据。它们也可能是重叠的引脚范围:UART发送器可能使用SET来断言开始和停止位，并使用OUT指令将FIFO数据移出到相同的引脚。

* + - 1. 汇编程序语法

设置目的地>，<值>

其中:

*<destination>* 是上面指定的目的地之一。

*<value>* 要设置的值（参见[第3.3.2](#_bookmark116)）（有效范围0-31）

* 1. 功能细节
     1. Side-set

侧集是一项功能，允许状态机在指令的主执行同时更改最多5个引脚的级别或方向

一个需要这样做的例子是快速SPI接口:在这里，时钟转换（切换1→0或0→1）必须与数据转换同时进行，其中新数据位从OSR移位到GPIO。在这种情况下，具有侧集的OUT

这使得接口的定时更加精确，减少了总体程序大小（因为不需要单独的SET指令来切换时钟引脚），并且还增加了SPI可以运行的最大频率

侧集也使得GPIO映射更加灵活，因为其映射独立于SET。 示例I2C代码允许SDA和SCL映射到任意两个引脚（如果时钟扩展禁用）。通常，SCL切换以同步数据传输，SDA包含被移出的数据位。但是，某些特定的I2C序列（如开始和停止线路条件）需要在SDA和SCL上驱动固定模式I2C用来实现这一点的映射

* 侧集→SCL
* OUT→SDA
* SET→SDA

这使得状态机可以服务于SDA上的数据和SCL上的时钟这两个用例，或者SDA和SCL上的固定转换，同时仍然允许SDA和SCL映射到任何两个选择的GPIO。

侧集数据编码在每个指令的延迟/侧集字段中。任何指令都可以与side set结合使用，包括写入引脚的指令，如OUT PINS或SET PINS。Side-set的引脚映射独立于OUT和SET映射，尽管它可能重叠。如果边置位和OUT或SET同时写入同一引脚，则使用边置位数据。

**注意**

如果指令暂停，侧集仍然立即生效

1.编程spi\_tx\_fast

2.side\_set1

3

4个循环:

5

6

输出引脚，1面0

jmp循环

侧1

spi\_tx\_fast示例显示了这样做的两个好处:数据和时钟转换可以更精确地对齐，并且程序可以整体更快，在这种情况下，每两个系统时钟周期输出一位程序也可以做得更小。

在使用侧集时，有四件事需要配置

1. 延迟/侧集字段的MSB数，用于侧集而不是延迟。这是通过以下方式配置的:

PINCTRL\_SIDESET\_COUNT。 如果设置为5，则延迟周期不可用。 如果设置为0，则不会发生侧集。

1. 是否使用这些位的最高有效位作为使能。侧置发生在使能为高电平的指令上。如果没有使能位，则该状态机上的**每条**指令都将执行边集（如果SIDESET\_NULL为非零）。这由EXECCTRL\_SIDE\_EN配置。
2. 将最低有效侧置位映射到的GPIO编号由PINCTRL\_SIDESET\_BASE配置。
3. 侧置写入是到GPIO电平还是GPIO方向。由EXECCTRL\_SIDE\_PINDIR配置

在上面的例子中，我们只有一个侧集数据位，并且每条指令都执行侧集，因此不需要使能位。SIDESET\_EN为1，SIDESET\_EN为false。SIDE\_PINDIR也将为假，因为我们希望驱动时钟的高电平和低电平，而不是高阻抗和低阻抗。SIDESET\_BASE将选择驱动时钟的GPIO。

* + 1. 程序包装

PIO程序通常有一个“外循环”:当它们在FIFO和外部世界之间传输数据流时，引言中的方波程序是一个最小的例子:

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave.pio#L7-L12)第7-12

 7 .program squarewave

 8 set pindirs, 1 ; Set pin to output

 9 again:

10 set pins, 1 [1] ; Drive pin high and then delay for one cycle

11 set pins, 0 ; Drive pin low

12 jmp again ; Set PC to label `again`

程序的主体驱动一个引脚高电平，然后低电平，产生一个周期的方波。然后整个程序循环，驱动周期性输出。跳转本身需要一个周期，每个设置指令也是如此，因此为了保持高电平和低电平周期的持续时间相同，设置引脚1增加了一个延迟周期，这使得状态机在执行设置引脚0指令之前空闲一个周期总的来说，每个循环需要四个周期。这里有两个挫折:

* JMP占用指令存储器中可用于其他程序的空间
* 执行JMP所需的额外周期最终会使最大输出速率*减半*

由于程序计数器（PC）在递增超过31时自然会回绕到0，因此我们可以通过用设置引脚1和设置引脚0的重复模式填充整个指令存储器来解决第二个问题，但这是浪费的。状态机有一个硬件特性，通过其EXECCTRL控制寄存器进行配置，解决了这种常见情况。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave\_wrap.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave_wrap.pio#L11-L19)第11-19

11 .program squarewave\_wrap

12 ; Like squarewave, but use the state machine's .wrap hardware instead of an

13 ; explicit jmp. This is a free (0-cycle) unconditional jump.

14

15 set pindirs, 1 ; Set pin to output

16 .wrap\_target

17 set pins, 1 [1] ; Drive pin high and then delay for one cycle

18 set pins, 0 [1] ; Drive pin low and then delay for one cycle

19 .wrap

在执行来自程序存储器的指令之后，状态机使用以下逻辑来更新PC:

1. 如果当前指令为JMP，且条件为真，则将PC设置为目标
2. 否则，如果PC匹配EXECCTRL\_WRAP\_TOP，则将PC设置为EXECCTRL\_WRAP\_BOTTOM
3. 否则，递增PC，或者如果当前值为31，则设置为0

pioasm中的.wrap\_target和.wrapassembly指令本质上是标签。它们导出可分别写入WRAP\_BOTTOM和WRAP\_TOP控制字段的常量

Pico示例:<https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/generated/squarewave_wrap.pio.h>

  1 // -------------------------------------------------- //

 2 // This file is autogenerated by pioasm; do not edit! //

 3 // -------------------------------------------------- //

 4

 5 #pragma once

 6

 7 #if !PICO\_NO\_HARDWARE

 8 #include "hardware/pio.h"

 9 #endif

10

11 // --------------- //

12 // squarewave\_wrap //

13 // --------------- //

14

15 #define squarewave\_wrap\_wrap\_target 1

16 #define squarewave\_wrap\_wrap 2

17

18 static const uint16\_t squarewave\_wrap\_program\_instructions[] = {

19 0xe081, // 0: set pindirs, 1

20 // .wrap\_target

21 0xe101, // 1: set pins, 1 [1]

22 0xe 100，*//2:设置 引脚，0*

[1]第一章

23

24};

25

//

.wrap

1. #如果！PICO\_NO\_硬件
2. static const structpio\_program squarewave\_wrap\_program ={
3. .instructions =squarewave\_wrap\_program\_instructions，
4. .length =3，
5. .origin =-1，

31};

32

1. static inlinepio\_sm\_configsquarewave\_wrap\_program\_get\_default\_config（uintoffset）{
2. pio\_sm\_config c =pio\_get\_default\_sm\_config（）;
3. sm\_config\_set\_wrap（c，offset + squarewave\_wrap\_wrap\_target，offset +squarewave\_wrap\_wrap）;
4. returnc;

37}

38#*endif*

这是来自PIO汇编程序pioasm的原始输出，它创建了一个包含WRAP的默认pio\_sm\_config对象

从程序列表中注册值控制寄存器字段也可以直接初始化

**注意**

WRAP\_BOTTOM和WRAP\_TOP是PIO指令存储器中的绝对地址。如果一个程序是在一个偏移量上加载的，则必须相应地调整回绕地址。

squarewave\_wrap示例插入了延迟周期，因此它的行为与原始方波程序相同由于程序包装，这些现在可以被删除，所以输出切换速度是原来的两倍，同时保持高和低周期的平衡。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave\_fast.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/squarewave/squarewave_fast.pio#L12-L18)第12-18

1. .programsquarewave\_fast
2. 类似于squarewave\_wrap，但删除了延迟周期，这样我们就可以运行两倍的速度。
3. set pindirs，1;将引脚设置为输出
4. .wrap\_target
5. 设置引脚，1;驱动引脚为高电平
6. 设置引脚，0;驱动引脚为低电平
7. .wrap
   * 1. FIFO Joining

默认情况下，每个状态机在每个方向上都有一个4条目FIFO:一个用于从系统到状态机（TX）的数据传输，另一个用于反向（RX）。然而，许多应用程序不需要系统和单个状态机之间的双向数据传输，但可以从更深的FIFO中受益:特别是DPI等高带宽接口 对于这些情况，SHIFTCTRL\_FJOIN可以将两个4条目FIFO合并为一个8条目FIFO。



*图42.可连接双FIFO。 一对四项FIFO，由四个数据寄存器、一个1:4解码器和一个4:1多路复用器实现。 额外的多路复用允许写数据和读数据在TX和RX通道之间交叉，以便从两个端口都可以访问所有8个条目*

另一个例子是一个UART:因为UART的TX/CTS和RX/RTS部分是异步的，所以它们在两个独立的状态机上实现。让每个状态机的FIFO资源的一半空闲是浪费的。对于TX/CTS状态机，将两个半部分结合成TX FIFO，或者对于RX/RTS状态机，将两个半部分结合成RX FIFO，这允许充分利用。一个配备8深FIFO的FIFO可以在两个中断之间保持两倍的时间，而只有一个4深FIFO的FIFO

当一个FIFO的大小增加（从4增加到8）时，该状态机上的另一个FIFO将减少到零。例如，如果加入TX，则RX FIFO不可用，并且任何PUSH指令将停止。RX FIFO将在FSTAT寄存器中同时显示RXFULL和RXEMPTY。如果加入RX，则反之亦然:TXFIFO不可用，并且此状态机的TXFULL和TXEMPTY位都将在FSTAT中设置。同时设置FJOIN\_RX和FJOIN\_TX会使两个FIFO都不可用。

8个FIFO条目足以通过RP 2040系统DMA实现每个时钟1个字，前提是DMA不会因与其他主机争用而变慢。

**谨慎**

更改FJOIN会丢弃状态机的FIFO中存在的任何数据。如果这些数据是不可替代的，则必须事先清除

* + 1. Autopush和Autopull

对于每个OUT指令，随着数据移出，OSR逐渐清空。一旦为空，就必须重新填充:例如，一个PSTO将一个字的数据从TX FIFO传输到OSR。同样，ISR一旦装满，就必须清空。其中一种方法是在适当数量的数据被移位后执行PADER的循环

1. .programmanual\_pull
2. .side\_set 1选项

3

1. .wrap\_target
2. 设置x，2; X =位计数-2
3. 拉侧1 [1];如果没有TX数据，则在此暂停
4. bitloop:

8

9

10

输出引脚，1

第0面[1]

jmp x--bitloop side 1[1]

输出引脚，1

0边

;移出数据位并切换时钟低电平

;循环运行3次

;在重新加载X之前移出最后一位

11.包装

该程序以每4个周期1位的恒定速率从每个FIFO字中移出4位，并附带位时钟。当TXFIFO为空时，它会在时钟为高时停止（注意，侧置仍然发生在

指令暂停）。[图43](#_bookmark125)显示了状态机如何执行这个程序。

图43. 执行manual\_pull程序。 X用作循环计数器。 每次迭代时，移出一个数据位，时钟先置位为低电平，然后置位为高电平。每个指令上的延迟周期使每次迭代的总周期达到四个。 在第三次循环之后，第四位被移出，并且状态机立即返回到程序的开始以重新加载循环计数器并拉取新的数据，同时保持4个周期/位的节奏。

系统时钟指令擦除X数据引脚（OUT）

时钟引脚（侧置）

设置 PSTO OUT

JMP

出来

JMP

出来

JMP

外集Pingle

2

1

0

-1

2

第0

位1

位2

位3

OSR移位计数

32

0

1

2

3

4

这个程序有一些限制:

* 它占用5个指令槽，但其中只有2个是立即有用的（输出引脚，1组0和..
* 由于需要额外的周期来拉入新数据和重新加载循环计数器，其吞吐量仅限于系统时钟超过4

这是PIO的常见问题，因此每个状态机都有一些额外的硬件来处理它。状态机跟踪OSR的总移位计数OUT和ISR的总移位计数IN，并在这些计数器达到可编程阈值时触发某些操作

* 在OUT指令达到或超过拉取阈值时，如果数据可用，状态机可以同时从TX FIFO重新填充
* 当IN指令达到或超过推送阈值时，状态机可以将移位结果直接写入RX FIFO，并清除ISR。

可以重写manual\_pull示例以利用自动pull（autopull）:

1 .program autopull

2 .side\_set 1

3

4 .wrap\_target

5 out pins, 1 side 0 [1]

6 nop side 1 [1]

7 .wrap

这比原来的更短，更简单，如果删除延迟周期，运行速度可以提高*一倍*，因为硬件“免费”重新填充OSR。请注意，该程序并不确定下一次拉取之前要移位的总位数;一旦达到可编程阈值SHIFCTRL\_PALTH\_THRESH，硬件就会自动拉取，因此同一程序也可以从每个FIFO字中移出例如16或32位。

最后，请注意，上述程序与原始程序并不*完全*相同，因为它在时钟输出为低而不是高时停止。我们可以使用PALTENIFEMPTY指令来改变停止的位置，它使用与自动清除相同的可配置阈值:

1 .program somewhat\_manual\_pull

2 .side\_set 1

3

4 .wrap\_target

5 out pins, 1 side 0 [1]

6 pull ifempty side 1 [1]

7 .wrap

下面是一个完整的例子（PIO程序，加上一个C程序来加载和运行它），它说明了在同一个状态机上启用自动填充和自动它对状态机0进行编程，将数据从TX FIFO环回至RX FIFO，吞吐量为每两个时钟一个字。它还演示了当OSR和TX FIFO都为空时，如果状态机尝试OUT，它将如何停止

1 .program auto\_push\_pull

2

3 .wrap\_target

4 out x, 32

5 in x, 32

6 .wrap

12 tb\_init();

13

14 // Load program and configure state machine 0 for autopush/pull with

15 // threshold of 32, and wrapping on program boundary. A threshold of 32 is

16 // encoded by a register value of 00000.

17 for (int i = 0; i < count\_of(auto\_push\_pull\_program); ++i)

18 mm\_pio->instr\_mem[i] = auto\_push\_pull\_program[i];

19 mm\_pio->sm[0].shiftctrl =

20 (1u << PIO\_SM0\_SHIFTCTRL\_AUTOPUSH\_LSB) |

21 (1u << PIO\_SM0\_SHIFTCTRL\_AUTOPULL\_LSB) |

22 (0u << PIO\_SM0\_SHIFTCTRL\_PUSH\_THRESH\_LSB) |

23 (0u << PIO\_SM0\_SHIFTCTRL\_PULL\_THRESH\_LSB);

24 mm\_pio->sm[0].execctrl =

25 (auto\_push\_pull\_wrap\_target << PIO\_SM0\_EXECCTRL\_WRAP\_BOTTOM\_LSB) |

26 (auto\_push\_pull\_wrap << PIO\_SM0\_EXECCTRL\_WRAP\_TOP\_LSB);

27

28 // Start state machine 0

29 hw\_set\_bits(&mm\_pio->ctrl, 1u << (PIO\_CTRL\_SM\_ENABLE\_LSB + 0));

30

31 // Push data into TX FIFO, and pop from RX FIFO

32 for (int i = 0; i < 5; ++i)

33 mm\_pio->txf[0] = i;

34 for (int i = 0; i < 5; ++i)

35 printf("%d\n", mm\_pio->rxf[0]);

36

37 return 0;

38 }

[图44](#_bookmark126)显示了状态机如何执行示例程序。 最初，OSR为空，因此状态机在第一个OUT指令时停止。 一旦数据在TXFIFO中可用，状态机将其传输到OSR中。在下一个周期，OUT可以使用OSR中的数据执行（在本例中，将此数据传输到X暂存寄存器），状态机同时使用FIFO中的新数据重新填充OSR由于每个IN指令都会立即填充ISR，因此ISR保持为空，IN将数据直接从擦除X传输到RX FIFO。

图44.执行auto\_push\_pull程序。 状态机在OUT上暂停，直到数据通过TX FIFO进入OSR。随后，OSR在每次OUT操作的同时被重新填充（由于位数为32），IN数据绕过ISR，直接进入RXFIFO。当FIFO耗尽时，状态机再次停止，并且OSR再次为空。

时钟电流指令

失速

出来

在外面在外面

出来

TX FIFO空TX FIFO弹出

OSR计数（0=满）

32

0

0

0

0

0

32

RX FIFO推送ISR计数（0=空）

RXFIFO推送

0

0

0

0

0

0

1

2

3

4

5

为了在正确的时间触发自动推或拉，状态机使用一对饱和的6位计数器跟踪ISR和OSR的总移位计数

* 复位时或CTRL\_SM\_RESTART置位时，ISR移位计数器设置为0（没有移入），OSR设置为32（没有移出）
* OUT指令使OSR移位计数器增加位数
* IN指令使ISR移位计数器增加位数
* 一条PSTO指令或自动清零会将OSR计数器清零
* PUSH指令或自动推送将ISR计数器清除为0
* MOVOSR，x或MOVISR，x分别将OSR或ISR移位计数器清零
* OUT ISR，n指令将ISR移位计数器设置为n

在任何OUT或IN指令上，状态机将移位计数器与SHIFTCTRL\_PUSH\_THRESH和SHIFTCTRL\_PUSH\_THRESH的值进行比较，以决定是否需要操作。自动填充和自动推送分别由SHIFT CTRL\_AUTOPUSH和SHIFT CTRL\_AUTOPUSH字段启用。

* + - 1. 自动推送详情

启用自动推送的“IN”的伪代码

 1 isr = shift\_in(isr, input())

 2 isr count = saturate(isr count + in count)

 3

 4 if rx count >= threshold:

 5 if rx fifo is full:

 6 stall

 7 else:

 8 push(isr)

 9 isr = 0

10 isr count = 0

请注意，硬件在单个机器时钟周期内执行上述步骤（除非出现停顿）。阈值可从1到32进行配置。

* + - 1. Autopull详细信息

在非“OUT”周期，硬件执行以下伪代码的等效操作

1 if MOV or PULL:

2 osr count = 0

3

4 if osr count >= threshold:

5 if tx fifo not empty:

6 osr = pull()

7 osr count = 0

因此，根据数据何时到达FIFO，自动清零可能发生在两个“OUT”之间的任何点。在“OUT”周期中，顺序略有不同:

1 if osr count >= threshold:

 2 if tx fifo not empty:

 3 osr = pull()

 4 osr count = 0

 5 stall

 6 else:

 7 output(osr)

 8 osr = shift(osr, out count)

 9 osr count = saturate(osr count + out count)

10

11 if osr count >= threshold:

12 if tx fifo not empty:

13 osr = pull()

14 osr count = 0

硬件能够在移出最后一个移位数据的同时重新填充OSR，因为这两个操作可以并行进行然而，它不能在同一个周期内填充空的OSR并将其“OUT”，因为这将创建较长的逻辑路径。

重新填充与您的程序有点异步，但“OUT”的行为就像数据围栏，状态机永远不会“OUT”您没有写入FIFO的数据

请注意，当启用自动清零时，来自OSR的“MOV”未定义;您将读取任何未移出的剩余数据，或来自FIFO的新字，具体取决于与系统DMA的竞争同样，OSR的“MOV”然而，数据，你'MOV'到OSR将永远不会被覆盖，因为'MOV'更新移位计数器。

如果您**确实**需要读取OSR的内容，则应该执行某种显式的“PSTIM”。上面描述的不确定性是硬件自动管理拉取的成本当启用自动清零时，“PALK”的行为这是为了避免对系统DMA的竞争条件。它的行为就像一个栅栏:要么是自动清空已经发生，在这种情况下，“PADER”没有效果，要么是程序将在“PADER”上停止，直到数据在FIFO中可用

“PUSH”不需要类似的行为，因为自动推送不具有相同的非确定性。

* + 1. 时钟分频器

PIO以系统时钟运行，但这对于许多接口来说太快了，并且可以插入的延迟周期数有限。一些设备，如UART，需要精确控制和改变信号速率，理想情况下，多个状态机可以在运行相同程序时独立改变。为此，每个状态机都配备了一个时钟分频器

时钟分频器不是减慢系统时钟本身，而是重新定义多少系统时钟周期被认为是“一个周期”，以用于执行目的。它通过生成时钟使能信号来实现这一点，该信号可以在每个系统时钟周期的基础上暂停和时钟分频器以规则的间隔产生时钟使能脉冲，

状态机以某种稳定的速度运行，可能比系统时钟慢得多

以这种方式实现时钟分频器允许状态机和系统之间的接口更简单、延迟更低并且占用空间更小状态机在时钟使能为低电平的周期内完全空闲，但系统仍可以访问状态机的FIFO并更改其配置。

时钟分频器为16位整数分频器、8位小数分频器，小数分频器为一阶Δ-Σ分频器时钟除数可以在1和65536之间变化，增量为。

如果时钟除数设置为1，则状态机在每个周期上运行，即全速:

图45.状态机操作，时钟除数为

1. 一旦状态机通过CTRL寄存器使能，其时钟使能将在每个周期置位。

系统时钟CLKDIV\_INT CLKDIV\_FRAC CTRL\_SM\_ENABLE

时钟使能

1

.0

一般来说，整数时钟除数*n*将导致状态机在每*n个周期中运行1个周期，*从而给出有效时钟速度 。

*图46. 时钟分频器产生周期性时钟使能。时钟分频器从n开始重复倒数*，并*发出使能脉冲*

系统时钟CLKDIV\_INT CLKDIV\_FRAC CTRL\_SM\_ENABLE

时钟使能

2

.0

*当它达到1。* 小数分频将保持稳态分频速率，其中*n*和*f*是此状态机CLKDIV寄存器的整数和小数字段。它通过选择性地将一些分裂期从周期延长到

.

图47. 分数时钟分频，平均除数为2.5。 时钟分频器保持来自每个分频周期的分数值的运行总和，并且每当该值绕过1时，整数除数在下一个分频周期增加1。

系统时钟CLKDIV\_INT CLKDIV\_FRAC CTRL\_SM\_ENABLE

时钟使能

2

.5

对于较小*的n*，小数分频器引入的抖动可能是不可接受的。但是，对于较大的值，这种影响就不那么明显了。

**注意**

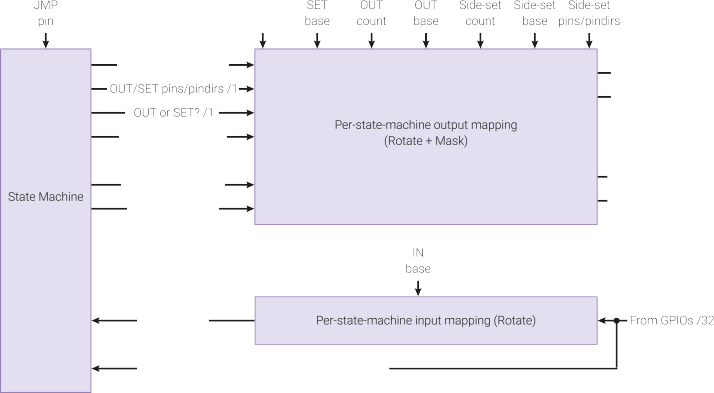
对于快速异步串行，建议尽可能使用偶数分频或1 Mbaud的倍数，而不是传统的300的倍数，以避免不必要的抖动。

* + 1. GPIO映射

在内部，PIO有一个32位寄存器用于它可以驱动的每个GPIO的输出电平，另一个寄存器用于输出使能（Hi/Lo-Z）。在每个系统时钟周期，每个状态机都可以写入这些寄存器中的部分或全部GPIO。

图48.状态机有两个独立的输出通道，一个由OUT/SET共享，另一个由side­set使用（可以随时发生）。三个独立的映射（第一个GPIO，GPIO数量）控制哪些GPIO OUT、SET和

侧集是针对。 输入数据根据哪个GPIO映射到IN数据的LSB进行旋转。



输出电平和输出使能寄存器的写数据和写掩码来自以下来源:

* OUT指令最多可写入32位。 取决于指令的目的地字段，这适用于引脚或引脚指示器。OUT数据的最低有效位被映射到PINCTRL\_OUT\_BASE，并且该映射持续

PINCTRL\_OUT\_OUT位，GPIO 31后换行。

* SET指令最多可写入5位。取决于指令的目的地字段，这适用于引脚或引脚指示器。SET数据的最低有效位被映射到PINCTRL\_SET\_BASE，并且该映射持续

PINCTRL\_SET\_Bit，GPIO 31后换行。

* 边集操作最多写入5位。取决于寄存器字段EXECCTRL\_SIDE\_PINDIR，这适用于引脚或引脚指示器。边集数据的最低有效位映射到PINCTRL\_SIDESET\_BASE，继续

如果设置了EXECCTRL\_SIDE\_EN，则PINCTRL\_SIDESET\_CLK引脚减1。

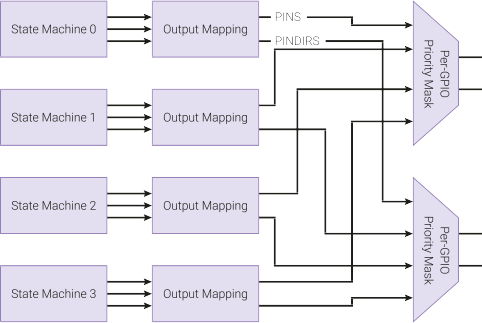
每个OUT/SET/侧置位操作写入一个连续的引脚范围，但每个范围的大小和位置都这对于许多应用来说是足够灵活的。例如，如果一个状态机正在一组引脚上实现某个接口（例如SPI），则另一个状态机可以运行映射到不同引脚组的相同程序，并提供第二个SPI接口。

在任何给定的时钟周期上，状态机可以执行OUT或SET，并且可以同时执行侧置位。引脚映射逻辑根据该请求和引脚映射配置，为输出电平和输出使能寄存器生成32位写掩码和写数据总线

如果侧集与该状态机在同一周期执行的OUT/SET重叠，则侧集在重叠区域中优先

* + - 1. 输出优先级

*图49.从每个状态机按GPIO优先级选择写掩码。 每个GPIO考虑来自四个状态机中的每个状态机的级别和方向写入，并应用来自编号最高的状态机的值。*



每个状态机可以在每个周期上通过其引脚映射硬件断言OUT/SET和侧置位。这将从每个状态机生成GPIO输出电平和输出使能寄存器的32位写数据和写掩码。

对于每个GPIO，PIO整理来自所有四个状态机的写入，并应用来自编号最高的状态机的写入

状态机这分别针对输出电平和输出方向发生-状态机可以在同一周期内改变同一引脚的电平和方向（例如，经由同时的SET和侧置位），或者一个状态机改变GPIO的方向而另一个状态机改变GPIO的电平。如果没有状态机断言对GPIO的电平或方向的写入，则值不会更改。

在每个状态机和引脚映射逻辑之间存在寄存器级，并且在输入映射逻辑和每个状态机之间存在寄存器级假设零传播延迟，观察其自身输出的状态机受到以下延迟:

* + - * + 当旁路同步器时，
        + 当同步器接合时，
      1. 输入映射

通过IN指令观察到的数据被映射，使得LSB是通过PINCTRL\_IN\_BASE选择的GPIO，并且连续更高有效位来自连续更高编号的GPIO，在31之后环绕。

换句话说，IN总线是GPIO输入值的右旋转，通过PINCTRL\_IN\_BASE。如果存在少于32个GPIO，则PIO输入用最多32位的零填充

某些指令（如WAIT GPIO）使用绝对GPIO编号，而不是IN数据总线的索引在这种情况下，不应用右旋转

* + - 1. 输入同步器

为了保护PIO不受干扰，每个GPIO输入都配备了标准的2触发器同步器。这为输入采样增加了两个周期的延迟，但好处是状态机可以在任何点执行IN PINS，并且只会看到干净的高或低电平，而不是可能干扰状态机电路的中间值这对于异步接口（如WRX）是绝对必要的。

可以在每个GPIO的基础上旁路这些同步器这减少了输入延迟，但随后由用户来保证状态机不会在不适当的时间对其输入进行采样通常，这仅适用于SPI等同步接口通过设置INPUT\_SYNC\_BYPASS中的相应位，可旁路同步器。

**警告**

对亚稳态输入进行采样可能导致不可预测的状态机行为。这是应该避免的。请勿禁用同步器，除非应用于引脚的数据满足相对于CLK\_CLK的建立和保持时间。

* + 1. 强制和执行指令

除了指令存储器，状态机还可以执行来自3个其他来源的指令

* MOVEXEC执行来自某个寄存器的指令
* OUTEXEC，执行从OSR移出的数据
* SMx\_SERR控制寄存器，系统可以向其写入指令以立即执行

  1 .program exec\_example

 2

 3 hang:

 4 jmp hang

 5 execute:

 6 out exec, 32

 7 jmp execute

 9 .program instructions\_to\_push

10

11 out x, 32

12 in x, 32

13 push

 1 #include "tb.h" // TODO this is built against existing sw tree, so that we get printf etc

 2

 3 #include "platform.h"

 4 #include "pio\_regs.h"

 5 #include "system.h"

 6 #include "hardware.h"

 7

 8 #include "exec\_example.pio.h"

 9

10 int main()

11 {

12 tb\_init();

13

14 for (int i = 0; i < count\_of(exec\_example\_program); ++i)

15 mm\_pio->instr\_mem[i] = exec\_example\_program[i];

16

17 // Enable autopull, threshold of 32

18 mm\_pio->sm[0].shiftctrl = (1u << PIO\_SM0\_SHIFTCTRL\_AUTOPULL\_LSB);

19

20 // Start state machine 0 -- will sit in "hang" loop

21 hw\_set\_bits(&mm\_pio->ctrl, 1u << (PIO\_CTRL\_SM\_ENABLE\_LSB + 0));

22

23 // Force a jump to program location 1

24 mm\_pio->sm[0].instr = 0x0000 | 0x1; // jmp execute

25

26 // Feed a mixture of instructions and data into FIFO

27 mm\_pio->txf[0] = instructions\_to\_push\_program[0]; // out x, 32

28 mm\_pio->txf[0] = 12345678; // data to be OUTed

29 mm\_pio->txf[0] = instructions\_to\_push\_program[1]; // in x, 32

30 mm\_pio->txf[0] = instructions\_to\_push\_program[2]; // push

31

32 // The program pushed into TX FIFO will return some data in RX FIFO

33 while (mm\_pio->fstat & (1u << PIO\_FSTAT\_RXEMPTY\_LSB))

34 ;

35

36 printf("%d\n", mm\_pio->rxf[0]);

37

38 return 0;

39 }

在这里，我们将一个示例程序加载到状态机中，它做两件事:

* 进入一个无限循环
* 进入一个循环，重复从TX FIFO中提取32位数据，并将低16位作为指令执行

C程序将状态机设置为运行，此时它进入挂起循环。当状态机仍在运行时，C程序强制执行一条jmp指令，这将导致状态机跳出循环。

当一条指令被写入寄存器时，状态机立即解码并执行该指令，而不是从PIO的指令存储器中提取的指令程序计数器不会前进，因此在下一个周期（假设强制进入到WTR接口的指令没有停止），状态机继续从它停止的点执行其当前程序，除非写入的指令本身

操作PC

延迟周期在写入到寄存器的指令上被忽略，并立即执行，忽略状态机时钟分频器。提供此接口用于执行初始设置和影响控制流更改，因此无论状态机如何配置，它都会及时执行指令

写入到寄存器的指令被允许暂停，在这种情况下，状态机将在内部锁存该指令，直到完成。 这由EXECCTRL\_EXEC\_STALLED标志表示。这可以通过重新启动状态机或将一个NOP写入到PSNR来清除。

在示例状态机程序的第二阶段中，使用OUT EXEC指令OUT本身占用一个执行周期，OUT执行的指令在下一个执行周期。请注意，我们执行的指令之一也是OUT-状态机在任何给定的周期中只能执行一条OUT指令。

OUT EXEC的工作原理是将OUT移位数据写入内部指令锁存器。在下一个周期，状态机记住它必须从这个锁存器而不是指令存储器执行，并且也知道在第二个周期不推进PC

这个程序运行时会打印“12345678”。

**谨慎**

如果写入INTR的指令停止，则它会存储在OUT EXEC和MOV EXEC使用的同一指令锁存器中，并将覆盖其中正在进行的指令。 如果使用EXEC指令，则写入到ODR的指令不能停止。

* 1. 示例

这些示例通过实现公共I/O接口来说明PIO的一些硬件特性

想要开始吗？

**[Raspberry Pi Pico C/C++ SDK](https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf)**书有一个全面的PIO章节，介绍了编写和构建第一个PIO应用程序，并逐行介绍了一些程序它还涵盖了更广泛的主题，如使用PIO与DMA，并深入到如何PIO可以集成到您的软件。

* + 1. 双工SPI

图50.在SPI中，主机和设备通过一对双向串行数据线交换数据，与时钟（SCK）同步两个标志CPOL和CPHA指定时钟的行为。

CPOL是时钟的空闲状态:0表示低电平，1表示高电平。 时钟脉冲多次，每个脉冲在每个方向上传输一位，但总是返回到其空闲状态。 CPHA确定在时钟的哪个边沿捕获数据:0表示前沿，1表示后沿。图中的箭头显示主机和设备同时捕获数据的时钟沿。



SPI是一种常见的串行接口，有着曲折的历史。下面的程序实现了全双工（即同时在两个方向上传输数据）SPI，其中CPHA参数为0。

*Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/spi.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/spi.pio#L14-L32)第14-32*

1. .程序spi\_cpa0
2. .side\_set1

16

1. ;引脚分配:
2. ;-SCK是侧置引脚0
3. ;-MOSI为OUT引脚0
4. ;-MISO为IN引脚0
5. ;
6. ;必须启用自动推送和自动填充，串行帧大小由
7. 配置推/拉阈值。向左/向右移动是可以的，但您必须
8. 自己证明数据的正确性。这是最方便的帧大小为
9. 通过使用窄存储复制和窄加载字节，8或16位
10. RP 2040的IO结构的拾取行为27
11. 时钟相位= 0:在每个SCK脉冲的前沿捕获数据，以及
12. 在后沿上或在第一前沿之前的某个时间过渡30
13. out pins，1 side 0 [1]; Stall here on empty（sideet proceeds evenif
14. 在引脚中，1侧1 [1];指令暂停，因此我们在SCK为低时暂停

这段代码使用自动推送和自动填充来连续地从FIFO传输数据。整个程序每传输一位就运行一次，然后循环。状态机跟踪移入/移出了多少位，并在正确的点自动推/拉FIFO一个类似的程序处理CPHA=1的情况:

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/spi.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/spi.pio#L34-L42)第34-42

1. .程序spi\_cpa1
2. .side\_set1

36

1. 时钟相位= 1:每个SCK脉冲的前沿上的数据转换，以及
2. 在后缘捕获39

40

41

42

输出x，1

0边

;在空时暂停（保持SCK无效）

mov引脚，x侧1 [1];输出数据，置位SCK（mov引脚使用OUT映射）

在引脚中，1侧0

;输入数据，取消断言SCK

**注意**

这些程序不控制片选线;由于不同SPI硬件之间的行为存在很大差异，片选通常被实现为软件控制的上面链接的完整spi.pio源代码包含一些PIO如何实现硬件芯片选择线的示例。

C helper函数配置状态机，连接GPIO，并设置状态机运行。请注意，SPI帧大小（即每个FIFO记录传输的位数）可以编程为1到32之间的任何值，而无需修改程序。一旦配置完毕，状态机就被设置为运行状态。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/spi.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/spi.pio#L46-L72)第46-72

1. static inline voidpio\_spi\_init（PIO pio，uint sm，uint prog\_offs，uintn\_bits，
2. booleanclkdiv，bool cpa，bool cpol，uint pin\_sck，uint pin\_mosi，uint pin\_miso）{
3. pio\_sm\_config c = cpha？spi\_cpa1\_program\_get\_default\_config（prog\_offs）:spi\_cpa0\_program\_get\_default\_config（prog\_offs）;
4. sm\_config\_set\_out\_pins（c，pin\_mosi，1）;
5. sm\_config\_set\_in\_pins（c，pin\_miso）;
6. sm\_config\_set\_sideset\_pins（c，pin\_sck）;
7. *//本例代码仅支持MSB优先（左移，自动推拉，阈值=nbits）*
8. sm\_config\_set\_out\_shift（c，false，true，n\_bits）;
9. sm\_config\_set\_in\_shift（c，false，true，n\_bits）;
10. sm\_config\_set\_clkdiv（c，clkdiv）;

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

//MOSI、SCK输出为低电平，MISO为输入

pio\_sm\_set\_pins\_with\_mask（pio，sm，0，（1u<< pin\_sck））|（1u<< pin\_mosi））; pio\_sm\_set\_pindirs\_with\_mask（pio，sm，（1u <<pin\_sck））|（1u <<pin\_mosi）、（1u <<pin\_sck）

| （1u <<pin\_mosi）|（1u <<pin\_miso））; pio\_gpio\_init（pio，pin\_mosi）; pio\_gpio\_init（pio，pin\_miso）; pio\_gpio\_init（pio，pin\_sck）;

//引脚多路复用器可以配置为反转输出（除其他外

//这是一个让CPOL=1的简单方法

gpio\_set\_outover（pin\_sck，cpol？GPIO\_OVERRIDE\_INVERT:GPIO\_OVERRIDE\_NORMAL）;

//SPI是同步的，所以旁路输入同步器以减少输入延迟。

hw\_set\_bits（pio->input\_sync\_bypass，1upin\_miso）;

pio\_sm\_init（pio，sm，prog\_offs，c）; pio\_sm\_set\_enabled（pio，sm，true）;

72}

状态机现在将立即开始移出TX FIFO中出现的任何数据，并将接收到的数据推入RX FIFO。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/pio\_spi.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/pio_spi.c#L18-L34)第18-34

1. 无效time\_critical\_func（pio\_spi\_write8\_blocking）（constpio\_spi\_inst\_t \*spi， constuint8\_t

\*src，size\_tlen）{

1. size\_ttx\_remain = len，int\_remain =len;
2. //在FIFO上进行 8位访问，因此写入数据是字节复制的。这
3. //获取free的左对齐（对于MSB优先移出）
4. io\_rw\_8 \*txfifo =（io\_rw\_8 \*）spi->pio->txf[spi->sm];
5. io\_rw\_8 \*rxfifo =（io\_rw\_8 \*）spi->pio->rxf[spi->sm];
6. while（tx\_remain||\_remain）{
7. if（tx\_remain&&！pio\_sm\_is\_tx\_fifo\_full（spi->pio，spi->sm））{
8. \* fifo =\*src++;
9. --tx\_remain;

28 }

|  |  |  |  |
| --- | --- | --- | --- |
| 30 |  |  | （void）; |
| 31 |  |  | --保留; |
| 32 |  | } |  |
| 33 | } |  |  |
| 34} |  |  |  |

将所有这些放在一起，这个完整的C程序将通过1 MHz的PIO SPI循环回一些数据，所有四种CPOL/CPHA组合:

29

if（0）&&; pio\_sm\_is\_fifo\_empty（spi->pio，spi->sm））{

Pico示例:<https://github.com/raspberrypi/pico-examples/blob/master/pio/spi/spi_loopback.c>

1*/\*\**

1. \*版权所有（c）2020 Raspberry Pi（Trading）Ltd.
2. \*
3. \* SPDX-License-Identifier:BSD-3-Clause
4. \*/

6

1. #include<stdlib. h>
2. #include<stdio.h>

9

1. #include“pico/stdlib. h”
2. #include“pio\_spi. h”

12

1. //此程序实例化一个PIO SPI，其中包含四种可能的
2. //CPOL/CPHA组合，串行输入和输出引脚映射到
3. //相同的GPIO。然后，写入状态机的TX FIFO的任何数据都应
4. //串行化、串行化并重新出现在状态机的RX FIFO中。

17

18*#define PIN\_SCK18*

19*#define PIN\_MOSI16*

20*#定义PIN\_MISO 16//与MOSI相同，因此我们得到PIN\_MISO 16。*

21

22*#define BUF\_SIZE20*

23

voidtest（constpio\_spi\_inst\_t \*spi）{

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47}

48

static uint8\_ttxbuf[BUF\_SIZE]; static uint8\_trxbuf[BUF\_SIZE];printf（“TX:“）;

for（ inti =0; i BUF\_SIZE;++i）{ txbuf[i] = rand（）>>16; rxbuf[i] =0;

printf（“%02x”，（int）txbuf[i]）;

}

printf（“"）;

pio\_spi\_write8\_read8\_blocking（spi，txbuf，rxbuf，BUF\_SIZE）;

printf（“x:“）;

returnfalse;

for（ inti =0; i BUF\_SIZE;++i）{ printf（“%02x”，（ int）rxbuf[i]）;

mismatch=不匹配||rxbuf [i]！= int [i];

}

if（unmatch）

printf（“\nNope\n”）;else

printf（“\n”）;

1. public voidrun（）{
2. stdio\_init\_all（）;

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77}

pio\_spi\_inst\_t spi ={

.pio =pio 0，

.sm =0

};

floatclkdiv = 31。25f;*//1 MHz@125clk\_sys*

uint cpha0\_offs = pio\_add\_program（spi.pio，spi\_cpha0\_program）; uint cpha1\_offs = pio\_add\_program（spi.pio，spi\_cpha1\_program）;

public intfindDuplicate（ intfindDuplicate =0; int findDuplicate =1）{

for（ intcpol =0; cpol =1; ++cpol）{ printf（“CPHA = %d，CPOL = %d\n”，cpha，cpol）; pio\_spi\_init（spi.pio，spi.sm，

cpa？cpha1\_offs\_offs:cpha0\_offs\_offs，

八、

clkdiv、cpha、cpol、PIN\_SCK、

//每SPI帧 8位

PIN\_MOSI，

PIN\_MISO

);

sleep\_ms（10）;

}

}

* + 1. WS2812LED

WS2812 LED由专有的脉冲宽度串行格式驱动，宽的正脉冲表示“1”位，窄的正脉冲表示“0”。每个LED都有一个串行输入和一个串行输出; LED连接成一个链，每个串行输入连接到前一个LED的串行输出。

图51.WS2812行格式。宽正脉冲为1，窄正脉冲为0，非常长的负脉冲为锁存使能



符号

输出

1

0

0

1

闩锁

LED消耗24位像素数据，然后将任何额外的输入数据传递到其输出。通过这种方式，单个串行突发可以单独编程链中每个LED的颜色 长负脉冲将像素数据锁存到LED中。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/ws2812/ws2812.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/ws2812/ws2812.pio#L7-L26)第7-26

1. .程序ws2812
2. .side\_set1

9

1. .define public T12
2. .define public T25
3. .定义公共T33

13

1. .lang\_opt python sideet\_init =pico.PIO.OUT\_HIGH
2. .lang\_opt pythonout\_init =pico.PIO.OUT\_HIGH
3. .lang\_opt python out\_shiftdir =1

17

1. .wrap\_target
2. bitloop:

20

21

输出x，1

side 0 [T3 - 1];当指令停止时，仍然会发生side set

jmp！x do\_zero side 1 [T1 - 1];在我们移出的位上分支正脉冲

22do\_one:

23

jmpbitloop侧1 [T2 - 1];继续驱动高电平，形成长脉冲

1. do\_zero:
2. nop侧0 [T2 - 1];或驱动低电平，用于短脉冲
3. .wrap

该程序将位从OSR移位到X，并根据每个数据位的值在侧置位引脚0上产生宽脉冲或窄脉冲。必须配置自动填充，阈值为24。然后，软件可以将24位像素值写入FIFO，这些值将被串行化到WS2812 LED链中。.pio文件包含一个用于设置的C helper函数

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/ws2812/ws2812.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/ws2812/ws2812.pio#L31-L47)第31-47

31static inline void ws2812\_program\_init（PIO pio，uint sm，uint offset，uint pin，floatfreq，bool rgbw）{

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47}

pio\_gpio\_init（pio，pin）;

pio\_sm\_set\_consequential\_pindirs（pio，sm，pin，1，true）;

pio\_sm\_config c = ws2812\_program\_get\_default\_config（offset）; sm\_config\_set\_sideet\_pins（c，pin）; sm\_config\_set\_out\_shift（c，false，true，rgbw？32:24）; sm\_config\_set\_fifo\_join（c，PIO\_FIFO\_JOIN\_TX）;

intcycles\_per\_bit = ws2812\_T1 + ws2812\_T2 + ws2812\_T3; floatdiv = clock\_get\_hz（时钟\_sys）/（频率 \* cycles\_per\_bit）; sm\_config\_set\_clkdiv（c，div）;

pio\_sm\_init（pio，sm，offset，c）; pio\_sm\_set\_enabled（pio，sm，true）;

因为移位是MSB优先的，并且我们的像素不是2的幂（因此我们不能依赖RP 2040上的窄写复制行为来为我们扇出位），所以我们需要对写入TX FIFO的值进行

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/ws2812/ws2812.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/ws2812/ws2812.c#L25-L27)第25-27

1. public static int findDuplicate（intfindDuplicate）{
2. pio\_sm\_put\_blocking（pio 0，0，pixel\_grb8u）;

27}

要DMA像素，我们可以将自动清空阈值设置为8位，将DMA传输大小设置为8位，并每次将每个像素将是3个单字节传输。由于RP 2040上的总线结构和DMA的工作方式，DMA传输的每个字节在写入32位IO寄存器时将出现四次复制，因此实际上您的数据位于移位寄存器的两端，您可以在任何方向上移动而无需担心。

再详细点？

WS 2812示例是**[Raspberry Pi Pico C/C++ SDK](https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf)**文档中PIO章节中教程的主题本教程逐行剖析ws2812程序，跟踪程序如何执行，并显示程序中每个点的GPIO输出波形图

* + 1. UARTTX

图52. 串行格式。空闲时线路高。发送器将线路拉低一个比特周期，以表示串行帧的开始（“起始比特”），随后是少量固定数量的数据比特。在下一个串行帧可以开始之前，线路返回到空闲状态至少一个比特周期（“停止比特”）。



位时钟

TX

状态

0

1

2 3

4 5

6

7

空闲

开始

数据（LSB优先）

停止

此程序实现通用异步接收/发送（UART）串行外设的发送组件。也许将其称为UAT更正确

*Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\_tx/uart\_tx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_tx/uart_tx.pio#L7-L17)第7-17*

1. .programuart\_tx
2. .side\_set 1选项

9

1. 一个8n1单片机传输程序。
2. ;OUT引脚0和侧置引脚0均映射到UART TX引脚。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12 |  | | | |
| 13 | 拉 | 第一面[7] | | ;断言停止位，或在线路处于空闲状态时停止 |
| 14 | 集合x，7 | 0面[7] | | ;预加载位计数器，置位起始位8个时钟 |
| 15bitloop:;此循环将运行8次（8 n1次） | | | | |
| 16 | 输出引脚，1 | | 从OSR向第一个OUT引脚移位1位 | |
| 17 | jmp x--bitloop | | [6] 每个循环迭代为8个循环。 | |

正如所写的那样，它将:

* 使引脚处于高电平状态，直到出现数据为止（注意，即使状态机处于停顿状态，侧置也会生效
* 置位开始位，持续8个SM执行周期
* 移出8个数据位，每个持续8个周期
* 在断言下一个起始位之前，返回到空闲线状态至少8个周期

如果状态机的时钟分频器配置为以8倍于所需波特率的速率运行，则无论何时通过软件或系统DMA将数据推送到TX FIFO，该程序都将传输格式良好的串行帧为了扩展程序以覆盖不同的帧大小（不同数量的数据位），可以用movx， y替换集合x，7，使得y暂存寄存器成为针对每个帧大小的每个SM配置寄存器。

SDK中的.pio文件还包含此函数，用于在程序加载到PIO指令存储器后配置引脚和状态机:

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\_tx/uart\_tx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_tx/uart_tx.pio#L23-L50)第23-50

23static inline void uart\_tx\_program\_init（PIO pio，uint sm，uint offset，uint pin\_tx，uint

baud）{

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

//告诉PIO最初在所选引脚上驱动输出高电平，然后映射PIO

//通过IO多路复用器连接到该引脚上。

pio\_sm\_set\_pins\_with\_mask（pio，sm，1upin\_tx，1upin\_tx）; pio\_sm\_set\_pindirs\_with\_mask（pio，sm，1upin\_tx，1upin\_tx）; pio\_gpio\_init（pio，pin\_tx）;

pio\_sm\_config c =uart\_tx\_program\_get\_default\_config（offset）;

//OUT向右移动，没有自动清零

sm\_config\_set\_out\_shift（c，true，false，32）;

//我们将OUT和side-set映射到同一个引脚，因为有时

//我们需要将用户数据断言到引脚上（使用OUT），有时

*//断言常量值（开始/停止位）*sm\_config\_set\_out\_pins（c，pin\_tx，1）; sm\_config\_set\_sideet\_pins（c，pin\_tx）;

//我们只需要TX，所以使用8深FIFO！

42

43

44

45

46

47

48

49

50}

sm\_config\_set\_fifo\_join（c，PIO\_FIFO\_JOIN\_TX）;

//SM每8个执行周期发送1位。

floatdiv =（ float）clock\_get\_hz（css\_sys）/（8\* 波特率）; sm\_config\_set\_clkdiv（c，div）;

pio\_sm\_init（pio，sm，offset，c）; pio\_sm\_set\_enabled（pio，sm，true）;

状态机被配置为右移输入输出指令，因为UART通常先发送数据LSB。配置完成后，状态机将打印推送到TX FIFO的任何字符

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\_tx/uart\_tx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_tx/uart_tx.pio#L52-L54)第52-54

1. public static int findDuplicate（int findDuplicate，int findDuplicate， intfindDuplicate）{
2. pio\_sm\_put\_blocking（pio，sm，（uint32\_t）c）;

54}

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\_tx/uart\_tx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_tx/uart_tx.pio#L56-L59)第56-59

1. public static int findDuplicate（intsize，int strategy， intstrategy）{
2. while（\*s）
3. uart\_tx\_program\_putc（pio，sm，\*s++）;

59}

SDK中的示例程序将配置一个PIO状态机作为一个PIO TX外设，并使用它以每秒115200波特的速度在GPIO 0上打印一条消息。

Pico示例:<https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_tx/uart_tx.c>

1*/\*\**

1. \*版权所有（c）2020 Raspberry Pi（Trading）Ltd.
2. \*
3. \* SPDX-License-Identifier:BSD-3-Clause
4. \*/

6

7*#包含“pico/stdlib.h”*

8*#包含“hardware/pio.h”*

9*#包含“uart\_tx.pio.h”*

10

publicint findDuplicate（）{

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

//我们将使用PIO打印“Hello，world！“在我们使用的同一个GPIO上

//通常将UART 0附加到。

constuint PIN\_TX =0;

//这与Pico上的默认波特率相同

constuint SERIAL\_BAUD =115200;

PIO pio = pio0; uint sm =0;

uint offset = pio\_add\_program（pio，uart\_tx\_program）; uart\_tx\_program\_init（pio，sm，offset，PIN\_TX，SERIAL\_BAUD）;

while（true）{

uart\_tx\_program\_puts（pio，sm，“Hello，world！（来自PIO！））; sleep\_ms（1000）;

}

27}

通过RP 2040上的两个PIO实例，这可以扩展到8个额外的UART TX接口，8个不同的引脚，8个不同的波特率。

* + 1. UARTRX

回顾[图52](#_bookmark130)，其示出了8n1中继器的格式



位时钟

TX

状态

0

1

2 3

4 5

6

7

空闲

开始

数据（LSB优先）

停止

我们可以通过等待起始位，以正确的时序采样8次，并将结果推送到RX FIFO来恢复数据下面是可以做到这一点的最短程序:

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\_rx/uart\_rx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_rx/uart_rx.pio#L7-L18)第7-18

7.程序uart\_mini.ini

8

9;最小可行8 n1接收器。等待起始位，然后采样8位

10、正确的时间。

11; IN引脚0映射到用作GPIO RX的GPIO。

12;必须启用自动推送，阈值为8。13

1. 等待0引脚0
2. set x，7[10]
3. bitloop:

17

18

在引脚中，1

;等待开始位

;预加载位计数器，延迟直到第一个数据位的眼

;循环8次

样本数据

jmp x--bitloop [6];每次迭代为8个循环

这是可行的，但它有一些恼人的特性，如重复输出NUL字符，如果行被卡住低。理想情况下，我们希望丢弃未被开始和停止位正确帧化的数据（并设置一些粘性标志来指示这种情况已经发生），并在线路长时间处于低电平时暂停接收我们可以将这些添加到程序中，代价是多了一些指令。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\_rx/uart\_rx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_rx/uart_rx.pio#L43-L62)第43-62

43.程序uart\_blog

44

45;稍微充实的8 n1接收器，可处理帧错误，

第46章更优雅的突破

47; IN引脚0和JMP引脚都映射到用作GPIO RX的GPIO48

49开始:

50

51

等待0引脚0

集合x，7

[10个国家]

52位循环:

53

54

55

56

57

58

59

60

在引脚中，1

;暂停，直到起始位被置位

;预加载位计数器，然后延迟到中途

第一个数据位（12个周期，包括等待、置位）。

;将数据位移位到ISR

jmp x--bitloop [6];循环8次，每次循环迭代8个周期

jmp pingood\_stop

;检查停止位（应为高）

irq 4rel

wait 1 pin 0 jmp start

;要么是框架错误，要么是中断。插上一面旗子，

并等待线路返回空闲状态。

如果我们没有看到好的框架，就不要推送数据。

61good\_stop:;返回开始前无延迟;稍微松弛

62 push ;在TX时钟稍微过快时很重要。

第二个例子不使用自动推送（[第3.5.4](#_bookmark124)），而是使用显式推送指令，以便它可以根据是否看到正确的停止位来调整推送.pio文件包含一个helper函数，用于配置状态机并将其连接到启用上拉的GPIO

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\_rx/uart\_rx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_rx/uart_rx.pio#L66-L84)第66-84

public static voiduart\_programme\_init（PIO，uint sm，uint offset，uint pin，uint baud）{

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

八四}

pio\_sm\_set\_consequential\_pindirs（pio，sm，pin，1，false）; pio\_gpio\_init（pio，pin）;

gpio\_pull\_up（pin）;

pio\_sm\_config c = uart\_rx\_program\_get\_default\_config（offset）;sm\_config\_set\_in\_pins（c，pin）;*//对于WAIT，IN*sm\_config\_set\_jmp\_pin（c，pin）;*//对于JMP*

//右移，自动推送禁用

sm\_config\_in\_shift（c，true，false，32）;

//更深的FIFO，因为我们不做任何TX

sm\_config\_set\_fifo\_join（c，PIO\_FIFO\_JOIN\_RX）;

//SM每8个执行周期发送1位。

floatdiv =（ float）clock\_get\_hz（css\_sys）/（8\* 波特率）; sm\_config\_set\_clkdiv（c，div）;

pio\_sm\_init（pio，sm，offset，c）; pio\_sm\_set\_enabled（pio，sm，true）;

为了正确接收LSB优先发送的数据，ISR配置为向右移动在移位8位后，这不幸地将8个数据位留在ISR的位31:24中，LSB中有24个零这里的一个选项是使用in null，24指令将ISR内容混洗到7:0。另一种方法是从FIFO中读取3个字节的偏移量，读取8位，以便处理器的总线硬件（或DMA）免费挑选相关字节

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/uart\_rx/uart\_rx.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_rx/uart_rx.pio#L86-L92)第86-92

1. public static inline charger（int n，int n）{
2. //从FIFO的最高字节读取8位，因为数据左对齐
3. io\_rw\_8 \*rxfifo\_shift =（io\_rw\_8\*）pio->rxf[sm]+3;
4. while（pio\_sm\_is\_fifo\_empty（pio，sm））
5. return（）;
6. return（char）;

92}

一个示例程序显示了如何使用该RX程序来接收RP 2040上的一个硬件UART发送的字符必须将GPIO 4连接到GPIO 3，才能使此程序正常工作。为了使3个不同的串行端口的争论更容易一些，这个程序使用核心1在测试控制台（CPU 1）上打印出一个字符串，在核心0上运行的代码将从PIO状态机中提取字符，并将它们传递给用于调试控制台（CPU 0）的CPU这里的另一种方法是基于中断的IO，使用PIO的FIFO IRQ。如果IRQ0\_INTE寄存器中的SM0\_RXNEMPTY位置1，则只要状态机0的RX FIFO中有字符，PIO就会引发其第一个中断请求行

Pico示例:<https://github.com/raspberrypi/pico-examples/blob/master/pio/uart_rx/uart_rx.c>

1*/\*\**

1. \*版权所有（c）2020 Raspberry Pi（Trading）Ltd.
2. \*
3. \* SPDX-License-Identifier:BSD-3-Clause

5*\*/*

6

7*#include<stdio.h>*

8

1. #include“pico/stdlib. h”
2. #include“pico/multicore.h”
3. #包含“hardware/pio.h”
4. #include“hardware/uart.h”
5. #include“uart\_rx.pio.h”

14

1. //这个程序
2. //-使用UART 1（默认情况下为备用UART）传输一些文本
3. //-使用PIO状态机接收该文本
4. //-将接收到的文本打印到默认控制台（UART 0）
5. //这可能需要在UART 1是
6. //默认值

21

1. #define SERIAL\_BAUDPICO\_DEFAULT\_BAUD\_RATE
2. #define HARD\_INSTuart1

24

1. //你需要从GPIO 4->GPIO 3连接一条线
2. #define HARD\_TX\_PIN4
3. #define PIO\_RX\_PIN3

28

1. //让core 1打印一个字符串，这样在core0上更容易
2. publicvoid run（）{
3. const char\*s =（ const char\*）multicore\_fifo\_pop\_blocking（）;
4. INST findDuplicate（s）;

33}

34

1. public voidrun（）{
2. //控制台输出（也是一个命令行，是的，它令人困惑）
3. setup. add（）;
4. printf（“正在启动PIO RX示例\n”）;39
5. //设置我们要用来打印字符的硬对象
6. uart\_init（HARD\_BAUD\_INST，SERIAL\_BAUD）;
7. gpio\_set\_function（HARD\_SET\_TX\_PIN，GPIO\_FLOW\_PIN）;

43

1. //设置我们要用来接收它们的状态机。
2. pio =pio 0;
3. int sum =0;
4. uint offset = pio\_add\_program（pio，&uart\_add\_program）;
5. uart\_programme\_init（pio，sm，offset，PIO\_RX\_PIN，SERIAL\_BAUD）;49
6. //告诉core 1尽可能快地打印一些文本到uart 1
7. multicore\_launch\_core1（core1\_main）;
8. constchar \*text =“Hello，world from PIO！（加上2个UART和2个核心，原因复杂）\n”;
9. multicore\_fifo\_push\_blocking（（uint32\_t）text）;

54

1. //将从PIO接收到的字符回显到控制台
2. while（true）{
3. charc = uart\_programme\_getc（pio，sm）;
4. return（c）;

59 }

60岁

* + 1. 曼彻斯特串行TX和RX

图53. 曼彻斯特串行线代码。 每个数据位由高脉冲后接低脉冲（表示“0”位）或低脉冲后接高脉冲（表示“1”位）表示。



*Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester\_encoding/manchester\_encoding.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester_encoding/manchester_encoding.pio#L7-L29)第7-29*

1. .programmanchester\_tx
2. .side\_set 1选项

9

1. ;每12个周期发送一位"0“被编码为高低序列
2. ;（每个部分持续半个比特周期，或6个周期），并且“1”被编码为
3. 低-高序列。
4. ;
5. 侧置位0必须映射到用于TX的GPIO。
6. 必须启用自动填充--此程序不关心阈值。
7. ;程序从公共标签“start”开始17
8. .wrap\_target
9. do\_1:

20

21

NOP

0侧[5];低电平持续6个周期（5个延迟，+1代表nop）

jmp get\_bit side 1 [3];高电平持续4个周期。'get\_bit'需要另外2个周期

22do\_0:

1. nop侧1 [5];输出高电平持续6个周期
2. nop侧0 [3];输出低电平持续4个周期
3. publicstart:
4. get\_bit:
5. out x，1;总是从OSR移出一个位到X，所以我们可以
6. jmp！xdo\_0;在其上分支 当OSR为空时，自动填充。
7. .wrap

从名为start的标签开始，该程序每次将一个数据位移入X寄存器，以便它可以在值上分支根据结果，它使用侧集将1-0或0-1序列驱动到所选GPIO上。此程序使用自动填充（[第3.5.4.2](#_bookmark127)），在移出一定量的数据后，自动从TX FIFO补充OSR，而此功能由.pio文件中的helper函数启用，该函数配置并启动状态机:

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester\_encoding/manchester\_encoding.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester_encoding/manchester_encoding.pio#L32-L45)第32-45

32static inline void manchester\_tx\_program\_init（PIO pio，uint sm，uint offset，uint pin，float

public void（）{

33

34

35

36

37

38

39

40

41

42

43

44

45}

pio\_sm\_set\_pins\_with\_mask（pio，sm，0，1upin）; pio\_sm\_set\_consecutive\_pindirs（pio，sm，pin，1，true）;

pio\_gpio\_init（pio，pin）;

pio\_sm\_config c = manchester\_tx\_program\_get\_default\_config（offset）; sm\_config\_set\_sideet\_pins（c，pin）;

sm\_config\_set\_out\_shift（c，true，true，32）;sm\_config\_set\_fifo\_join（c，PIO\_FIFO\_JOIN\_TX）; sm\_config\_set\_clkdiv（c，div）;

pio\_sm\_init（pio，sm，offset + manchester\_tx\_offset\_start，&c）;

pio\_sm\_set\_enabled（pio，sm，true）;

可以对另一个状态机进行编程，以从传输的信号中恢复原始数据

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester\_encoding/manchester\_encoding.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester_encoding/manchester_encoding.pio#L48-L70)第48-70

48.方案manchester\_casino

49

1. ;假设线路为空闲低电平，第一位为0
2. ; 1位为12个周期
3. ;“0”被编码为10
4. ;“1”被编码为01
5. ;
6. ; IN基座和JMP引脚映射都必须指向用于RX的GPIO。
7. ;必须启用自动推送。
8. 在使能SM之前，SM应处于“wait 1，pin”状态，以便
9. 在初始线路空闲状态结束之前，它不会开始采样59

60start\_of\_0:

61

62

63

64

等待0引脚0

在y，1[8]

;我们是0.25位成0-信号为高

;等待1->0转换-此时我们进入位的0.5

;发出0，休眠3/4位

jmp引脚start\_of\_0;如果信号再次为1，则为另一个0位，否则为1

1. .wrap\_target
2. start\_of\_1:

67

68

69

等待1引脚0

在x中，1[8]

;我们将0.25位转换为1-信号为1

;等待0->1转换-此时我们进入位的0.5

;发出1，休眠3/4位

jmp引脚start\_of\_0;如果信号再次为0，则为另一个1位，否则为0

70.wrap

这里的主要复杂性是保持与输入转换对齐，因为发射器和接收器的时钟可能相对于彼此漂移。在曼彻斯特码中，符号的中心总是有一个转换，并且基于初始行状态（高或低），我们知道这个转换的方向，因此我们可以使用等待指令来跟踪每个数据位上的行转换

这个程序希望X和Y寄存器分别初始化为1和0，这样就可以为in指令提供一个常量1配置状态机的代码通过在设置程序运行之前执行一些设置指令来初始化这些寄存器。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester\_encoding/manchester\_encoding.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester_encoding/manchester_encoding.pio#L73-L93)第73-93

73static inline void manchester\_program\_init（PIO pio，uint sm，uint offset，uint pin，float

public void（）{

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93}

pio\_sm\_set\_consequential\_pindirs（pio，sm，pin，1，false）;

pio\_gpio\_init（pio，pin）;

pio\_sm\_config c = manchester\_config\_program\_get\_default\_config（offset）; sm\_config\_set\_in\_pins（c，pin）;*//对于WAIT* sm\_config\_set\_jmp\_pin（c，pin）;*//对于JMP* sm\_config\_set\_in\_shift（c，true，true，32）; sm\_config\_set\_fifo\_join（c，PIO\_FIFO\_JOIN\_RX）; sm\_config\_set\_clkdiv（c，div）;

pio\_sm\_init（pio，sm，offset，&c）;

//将X和Y设置为0和1，以方便将它们发送到ISR/FIFO。

pio\_sm\_exec（pio，sm，pio\_encode\_set（pio\_x，1））;pio\_sm\_exec（pio，sm，pio\_encode\_set（pio\_y，0））;

//假设线路为空闲低电平，并且第一个传输位为0。将SM放入

//在启用之前等待状态RX将在第一个0符号

//检测到。

pio\_sm\_exec（pio，sm，pio\_encode\_wait\_pin（1，0））|pio\_encode\_delay（2））; pio\_sm\_set\_enabled（pio，sm，true）;

SDK中的示例C程序将以大约10Mbps的速率（假设系统时钟为125MHz）从GPIO 2向GPIO 3传输曼彻斯特串行数据

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester\_encoding/manchester\_encoding.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/manchester_encoding/manchester_encoding.c#L20-L43)第20-43

publicint findDuplicate（）{

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43}

stdio\_init\_all（）;

PIO pio = pio0; uint sm\_tx =0; uint sm\_tx =1;

uint offset\_tx = pio\_add\_program（pio，manchester\_tx\_program）; uint offset\_tx = pio\_add\_program（pio，manchester\_tx\_program）; printf（“在% d\n加载的发送程序”，offset\_tx）; printf（“在%d\n加载的接收程序”，offset\_tx）;

manchester\_tx\_program\_init（pio，sm\_tx，offset\_tx，pin\_tx，1.f）;manchester\_tx\_program\_init（pio，sm\_tx，offset\_tx，pin\_tx，1.f）;

pio\_sm\_set\_enabled（pio，sm\_tx，false）; pio\_sm\_put\_blocking（pio，sm\_tx，0）; pio\_sm\_put\_blocking（pio，sm\_tx，0x0ff0a55a）; pio\_sm\_put\_blocking（pio，sm\_tx，0x12345678）; pio\_sm\_set\_enabled（pio，sm\_tx，true）;

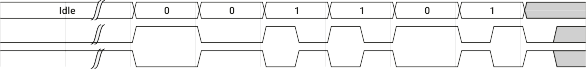
inti =0; i= 0; i= 0

printf（“%08x\n”，pio\_sm\_get\_blocking（pio，sm\_block））;

* + 1. 差分曼彻斯特（BMC）TX和RX

图54.差分曼彻斯特串行线码，也称为双相标志码（BMC）。 线在每个位周期的开始处转变。

*在位周期的中心存在转变表示*1*数据位，而不存在转变表示*0*位。 这些编码规则是相同的，无论线路具有初始高或低状态。*



传输程序类似于曼彻斯特示例:它重复地将一个位从OSR移位到X（依赖于后台自动填充OSR），分支，并根据此位的值上下驱动GPIO增加的复杂性是，我们驱动到引脚上的模式不仅取决于数据位的值，就像vanilla Manchester编码一样，还取决于最后一个位周期结束时线路所处的状态这在[图54](#_bookmark131)中示出，其中如果线初始为高，则图案反转 为了解决这个问题，有两个测试和驱动代码的副本，每个初始线路状态一个，并且这些通过一系列跳转以正确的顺序链接在一起。

*Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/differential\_manchester/differential\_manchester.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/differential_manchester/differential_manchester.pio#L7-L34)第7-34*

1. .programdifferential\_manchester\_tx
2. .side\_set 1选项

9

1. ;每16个周期发送一位在每个比特周期中:
2. ;-A“0”被编码为比特周期开始时的转换
3. ;-A“1”被编码为开头 \* 和中间\*的转换
4. ;
5. ;侧置位0必须映射到数据输出引脚。
6. ;必须启用自动填充16
7. publicstart:
8. initial\_high:
9. out x，1;位周期开始:始终断言转换
10. jmp！xhigh\_0 side 1 [6];测试刚移出OSR的数据位

21高\_1:

22

23

NOP

jmp initial\_high side 0 [6];对于`1`位，也在中间过渡

24高\_0:

25

26

jmp初始低

[7];否则，线在中间稳定

1. initial\_low:
2. out x，1;始终将1位从OSR移到X，以便我们可以
3. jmp！xlow\_0 side 0 [6];在其上分支 Autopull为我们重新填充OSR。
4. 低1:
5. NOP
6. jmpinitial\_low side 1 [6];如果有两个转换，则返回到
7. 低\_0:
8. jmpinitial\_high [7];初始行状态被翻转！

.pio文件还包括一个帮助函数，用于初始化差分曼彻斯特TX的状态机，并将其连接到选定的GPIO。我们任意选择32位帧大小和LSB优先串行化（在sm\_config\_set\_out\_shift中，shift\_to\_right为true），但由于程序每次只运行一位，因此我们可以通过重新配置状态机来更改此设置。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/differential\_manchester/differential\_manchester.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/differential_manchester/differential_manchester.pio#L37-L52)第37-52

37static inline void differential\_manchester\_tx\_program\_init（PIO pio，uint sm，uintoffset，

public int findDuplicate（intfindDuplicate）{

38

39

40

41

42

43

44

45

46

47

48

49

pio\_sm\_set\_pins\_with\_mask（pio，sm，0，1upin）; pio\_sm\_set\_consecutive\_pindirs（pio，sm，pin，1，true）;

pio\_gpio\_init（pio，pin）;

pio\_sm\_config c = differential\_manchester\_tx\_program\_get\_default\_config（offset）; sm\_config\_set\_sideet\_pins（c，pin）;

sm\_config\_set\_out\_shift（c，true，true，32）;sm\_config\_set\_fifo\_join（c，PIO\_FIFO\_JOIN\_TX）; sm\_config\_set\_clkdiv（c，div）;

pio\_sm\_init（pio，sm，offset + differential\_manchester\_tx\_offset\_start，&c）;

50

51

//执行阻塞拉取，这样我们就可以保持初始行状态，直到数据可用

pio\_sm\_exec（pio，sm，pio\_encode\_pull（false，true））;

pio\_sm\_set\_enabled（pio，sm，true）;

52}

RX程序使用以下策略:

* 等待直到位周期开始时的初始转换，因此我们保持与发送时钟对齐
* 然后等待配置位周期的3/4，以便我们以第二个半位周期为中心（见[图54](#_bookmark131)）
* 在这一点上对行进行采样，以确定在这一位周期中是有一个还是两个转换
* 重复

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/differential\_manchester/differential\_manchester.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/differential_manchester/differential_manchester.pio#L54-L84)第54-84

54.programdifferential\_manchester\_campaign

55

56;假设线路为低电平空闲

57;一位是16个周期。在每个比特周期中:

58;-A '0'被编码为时间0处的转变

- A“1”被编码为时间0处的转变和时间T/2处的转变

60岁;

61; IN映射和JMP引脚选择必须同时映射到用于

62; RX数据。必须启用自动推送63

64公共启动:

65initial\_high:;查找位周期开始时的上升沿

66

67

等待1引脚，0 [11];延迟到第二个半周期的眼（即3/4路

jmp引脚高\_0

;通过位），并在RX引脚高/低上分支。

68高\_1:

69

70

在x中，1

jmp初始高

;检测到第二次转换（“1”数据符号）

71高\_0:

72

73

74

在y中，1[1]

;Fall-through

;线仍然高，无中心转换（数据为“0”）

1. .wrap\_target
2. initial\_low:

77

78

等待0引脚，0[1]

jmp引脚低\_1

;在位周期开始时查找下降沿

;后半周期眼延迟

79低\_0:

80

81

在y中，1

jmp初始高

;线路仍为低电平，无中心转换（数据为“0”）

82low\_1:;检测到第二次转换（数据为“1”）

83in x，1[1]

84.wrap

此代码假定X和Y的值分别为1和0。这是由包含的C helper函数安排的:

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/differential\_manchester/differential\_manchester.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/differential_manchester/differential_manchester.pio#L87-L103)第87-103

87静态内联无效差示\_manchester\_rx\_program\_init（PIO pio，uint sm，uintoffset，

public int findDuplicate（intfindDuplicate）{

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103}

pio\_sm\_set\_consequential\_pindirs（pio，sm，pin，1，false）;

pio\_gpio\_init（pio，pin）;

pio\_sm\_config c = differential\_manchester\_programme\_get\_default\_config（offset）;sm\_config\_set\_in\_pins（c，pin）;*//用于WAIT*

sm\_config\_set\_jmp\_pin（c，pin）;*//适用于JMP* sm\_config\_set\_in\_shift（c，true，true，32）;sm\_config\_set\_fifo\_join（c，PIO\_FIFO\_JOIN\_RX）; sm\_config\_set\_clkdiv（c，div）; pio\_sm\_init（pio，sm，offset，c）;

//将X和Y设置为0和1，以方便将它们发送到ISR/FIFO。

pio\_sm\_exec（pio，sm，pio\_encode\_set（pio\_x，1））; pio\_sm\_exec（pio，sm，pio\_encode\_set（pio\_y，0））; pio\_sm\_set\_enabled（pio，sm，true）;

现在，所有这些部件都可以通过两个GPIO之间的线路传输一些串行数据

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/differential\_manchester/differential\_manchester. c](https://github.com/raspberrypi/pico-examples/blob/master/pio/differential_manchester/differential_manchester.c)

1*/\*\**

1. \*版权所有（c）2020 Raspberry Pi（Trading）Ltd.
2. \*
3. \* SPDX-License-Identifier:BSD-3-Clause
4. \*/

6

7*#include<stdio.h>*

8

1. #include“pico/stdlib. h”
2. #包含“hardware/pio.h”
3. #include“differential\_manchester. pio. h”

12

1. //差分串行发送/接收示例
2. //需要从GPIO 2->GPIO 3连接一条线

15

16const uint pin\_tx =2;

17const uint pin\_pin =3;

18

publicint findDuplicate（）{

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43}

stdio\_init\_all（）;

PIO pio = pio0; uint sm\_tx =0; uint sm\_tx =1;

uint offset\_tx = pio\_add\_program（pio，differential\_manchester\_tx\_program）; uint offset\_tx = pio\_add\_program（pio，differential\_manchester\_tx\_program）; printf（“在%d\n加载传输程序”，offset\_tx）;

printf（“接收程序在%d\n加载，offset\_n）;

*//配置状态机，将比特率设置为5 Mbps*differential\_manchester\_tx\_program\_init（pio，sm\_tx，offset\_tx，pin\_tx，125.f/（16\*5））; differential\_manchester\_program\_init（pio，sm\_tx，offset\_tx，pin\_tx，125.f/（16\*5））;

pio\_sm\_set\_enabled（pio，sm\_tx，false）; pio\_sm\_put\_blocking（pio，sm\_tx，0）; pio\_sm\_put\_blocking（pio，sm\_tx，0x0ff0a55a）; pio\_sm\_put\_blocking（pio，sm\_tx，0x12345678）; pio\_sm\_set\_enabled（pio，sm\_tx，true）;

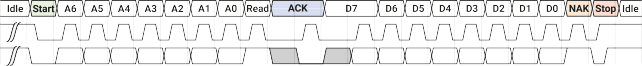
inti =0; i= 0; i= 0

printf（“%08x\n”，pio\_sm\_get\_blocking（pio，sm\_block））;

* + 1. I2c

*图55.1字节I2C读传输。 在空闲状态下，两条线路都浮高。启动器驱动SDA为低电平（启动条件），后面是7个地址位A6-A0和一个方向位（读/写）。 目标驱动SDA为低电平以确认地址（ACK）。 数据字节紧随其后。 目标在SDA上串行化数据，由SCL时钟输出。 每隔第9个时钟，***启动器***将SDA拉低以确认数据，但最后一个字节除外，在该字节，启动器将线路保持高电平（NAK）。*

当SCL为高电平时释放SDA是停止条件，使总线返回空闲状态。



I2C是一种普遍存在的串行总线，最早在死海古卷中描述，后来由飞利浦半导体使用。两条带有上拉电阻器的导线形成开漏总线，多个代理通过将总线驱动为低或释放它们以将其拉高来在该总线上彼此寻址和发信号它有一些不寻常的属性:

* SCL可以由总线的任何成员（不一定是传输的目标或发起者）在任何时间、任何持续时间内保持低这被称为时钟拉伸。直到所有司机都松开时钟，公共汽车才会前进
* 总线的成员可以是一个传输的目标，并启动其他传输（主/从角色不固定）。然而，大多数I2C硬件对这一点的支持都很差。
* SCL不是边沿敏感时钟，而是SDA必须在SCL为高电平的整个时间内有效
* 尽管SDA相对于SCL是透明的，但是当SCL为高时SDA的转换用于标记传输的开始和结束（开始/停止），或者一个内的新地址阶段（重新启动）。

下面列出的PIO程序处理序列化、时钟拉伸和发起者角色中ACK的检查。它提供了一种在FIFO数据流中转义PIO指令的机制，以便在适当的时间发出启动/停止/重新启动序列。如果未收到意外NAK，则可以从DMA缓冲区执行长序列I2C传输

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/i2c.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/i2c.pio#L7-L72)第7-72

1. .程序i2c
2. .side\_set 1 optpindirs

9

1. ; TX编码:
2. ；|十五点十分|9 |八比一|0 |
3. ；|Instr|最终|数据|NAK|
4. ;
5. ;如果Instr的值n> 0，则该FIFO字没有
6. 数据有效载荷，并且接下来的n +1个字将作为指令执行。
7. 否则，移出8个数据位，然后是ACK位。
8. ;
9. Instr机制允许对停止/启动/重新启动序列进行编程
10. ;由处理器执行，然后由状态机在定义的点执行
11. ;在数据流中。
12. ;
13. ;“Final”字段应设置为传输中的最后一个字节。
14. ;这告诉状态机忽略NAK:如果此字段不是
15. 设置，则任何NAK都将导致状态机停止和中断。
16. ;
17. ;应启用自动填充，阈值为16。
18. ;应启用自动推送，阈值为8。
19. ;应使用半字写入访问TX FIFO，以确保
20. ; OSR中的数据立即可用。
21. ;
22. ;引脚映射:
23. ;-输入引脚0为SDA，1为SCL（如果使用时钟扩展
24. ;-跳线为SDA
25. ;-侧置引脚0为SCL
26. ;-设置引脚0为SDA
27. ;-OUT引脚0为SDA
28. ;-SCL必须为SDA +1（用于等待映射）
29. ;
30. OE输出应在系统IO控制中反转！
31. ;（在这个程序中可以进行反演
32. ;但需要2个指令:1个用于反转，另一个用于处理
33. ;具有MOV对TX移位计数器的副作用43

|  |  |  |  |
| --- | --- | --- | --- |
| 51 | 输出pindirs，1 [7] | ; | 串行化写入数据（如果读取，则为全1） |
| 52 | nopside 1[2] | ; | SCL上升沿 |
| 53 | 等待1引脚，1 [4] | ; | 允许时钟被拉伸 |
| 54 | 在引脚中，1 [7] | ; | 在SCL脉冲中间采样读取数据 |
| 55 | jmp x--bitloop side 0[7] | ; | SCL下降沿 |
| 56 |  |  |  |
| 57 | ;处理ACK脉冲 |  |  |
| 58 | 输出pindirs，1 [7] | ; | 在读取时，我们提供ACK。 |
| 59 | nopside 1[7] | ; | SCL上升沿 |
| 60 | 等待1引脚，1 [7] | ; | 允许时钟被拉伸 |
| 61 | jmp pin do\_nack side 0[2] | ; | 测试SDA的ACK/NAK，如果ACK，则失败 |
| 62 |  |  |  |
| 63公共入口点:  64.wrap\_target | | | |
| 65 | 输出x，6 | ; | 拆包指令计数 |
| 66 | 输出y，1 | ; | 解包NAK忽略位 |
| 67 | jmp！xdo\_byte | ; | Instr== 0，这是一个数据记录。 |
| 68 | out null，32 | ; | Instr> 0，此OSR的其余部分无效 |
| 69do\_exec: | | | |
| 70 | out exec，16 | ; | 每个FIFO字执行一条指令 |
| 71 | jmp x--do\_exec | ; | 重复n +1次 |
| 72.wrap | | | |

I2C程序所需的IO映射相当复杂，因为必须以不同的方式驱动和采样两条串行线。一个有趣的特性是，当输出为低时，状态机必须驱动输出使能为高，因为总线是开漏的，所以数据的意义被反转。这可以在PIO程序中处理（例如， mov osr，~osr），但我们可以使用RP 2040上的IO控制在GPIO多路复用器中执行此反转，从而节省一条指令。

44do\_nack:

45

46

47

jmp y--入口点

irq等待0rel

;如果预期为NAK，则继续

;否则停止，寻求帮助

1. do\_byte:
2. 设置x，7;循环8次
3. bitloop:

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/i2c.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/i2c.pio#L80-L120)第80-120

80static inline void i2c\_program\_init（PIO pio，uint sm，uint offset，uint pin\_sda，uint

pinscl）{

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

assert（pin\_scl == pin\_sda +1）;

pio\_sm\_config c =i2c\_program\_get\_default\_config（offset）;

//IOmapping

sm\_config\_set\_out\_pins（c，pin\_sda，1）;

sm\_config\_set\_set\_pins（c，pin\_sda，1）; sm\_config\_set\_in\_pins（c，pin\_sda）;sm\_config\_set\_sideet\_pins（c，pin\_scl）; sm\_config\_set\_jmp\_pin（c，pin\_sda）;

sm\_config\_set\_out\_shift（c，false，true，16）; sm\_config\_set\_in\_shift（c，false，true，8）;

floatdiv =（ float）clock\_get\_hz（clock\_sys）/（32\*100000）; sm\_config\_set\_clkdiv（c，div）;

//在连接IO时，尽量避免总线出现毛刺把事情安排好

//向上，以便当PIO置位OE低电平时，引脚被拉低，

*//否则。*gpio\_pull\_up（pin\_scl）; gpio\_pull\_up（pin\_sda）;

102

103

104

105

106

107

108

109

110

111

112

113

uint32\_t both\_pins =（1u<< pin\_sda）|（1u<< pin\_scl）; pio\_sm\_set\_pins\_with\_mask（pio，sm，both\_pins，both\_pins）; pio\_sm\_set\_pindirs\_with\_mask（pio，sm，both\_pins，both\_pins）; pio\_gpio\_init（pio，pin\_sda）;

gpio\_set\_oeover（pin\_sda，GPIO\_OVERRIDE\_INVERT）;pio\_gpio\_init（pio，pin\_scl）;gpio\_set\_oeover（pin\_scl，GPIO\_OVERRIDE\_INVERT）;

pio\_sm\_set\_pins\_with\_mask（pio，sm，0，both\_pins）;

114

115

116

117

118

119

//在启动前清除IRQ标志，并确保该标志实际上不会

//断言系统级中断（我们将其用作状态标志）

pio\_set\_irq0\_source\_enabled（pio，（enumpio\_interrupt\_source）（（uint）pis\_interrupt0 + sm），false）;

pio\_set\_irq1\_source\_enabled（pio，（enumpio\_interrupt\_source）（（uint）pis\_interrupt0 + sm），false）;

pio\_interrupt\_clear（pio，sm）;

//配置并启动SM

pio\_sm\_init（pio，sm，offset + i2c\_offset\_entry\_point，c）; pio\_sm\_set\_enabled（pio，sm，true）;

120}

我们还可以使用PIO汇编程序生成一个通过FIFO的指令表，用于启动/停止/重新启动条件。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/i2c.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/i2c.pio#L125-L135)第125-135

1. .programset\_scl\_sda
2. .side\_set 1选项

127

1. ;汇编一个指令表，软件可以从中选择，并传递
2. 输入FIFO，以发出START/STOP/RSTART。这并不意味着
3. 一个完整的程序。131
4. 设置pindirs，0 side 0 [7]; SCL = 0，SDA =0
5. 设置引脚，1侧0 [7]; SCL = 0，SDA =1
6. 设置引脚，0侧1 [7]; SCL = 1，SDA =0
7. 设置引脚，1面1 [7]; SCL = 1，SDA =1

示例代码在状态机的FIFO上阻塞软件IO，以避免设置系统DMA的额外复杂性例如，I2C启动条件是这样排队的

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/pio\_i2c.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/pio_i2c.c#L69-L73)第69-73

1. public int findDuplicate（int findDuplicate）{
2. pio\_i2c\_put\_or\_err（pio，sm，1uPIO\_I2C\_ICOUNT\_LSB）;*//2指令序列*的*转义码*

71

72

pio\_i2c\_put\_or\_err（pio，sm，set\_scl\_sda\_program\_instructions[I2C\_SC1\_SD0]）;

已处于空闲状态，只需将SDA拉低

pio\_i2c\_put\_or\_err（pio，sm，set\_scl\_sda\_program\_instructions[I2C\_SC0\_SD0]）;

拉低时钟，以便我们可以呈现数据

//我们是

//也可以

73}

由于I2C可能在很多点上出错，因此我们需要能够检查状态机断言的错误标志，在断言停止条件并释放总线之前清除停止并重新启动它

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/pio\_i2c.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/pio_i2c.c#L15-L17)第15-17

1. public bol bol（int p，int n）{
2. returnpio\_interrupt\_get（pio，sm）;

17}

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/pio\_i2c.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/pio_i2c.c#L19-L23)第19-23

1. public int findDuplicate（int findDuplicate）{
2. pio\_sm\_drain\_tx\_fifo（pio，sm）;
3. pio\_sm\_exec（pio，sm，（pio->sm[sm]. execcodePIO\_SM0\_EXECCTRL\_WRAP\_BOTTOM\_BITS）>>PIO\_SM0\_EXECCTRL\_WRAP\_BOTTOM\_LSB）;
4. pio\_interrupt\_clear（pio，sm）;

23}

我们需要一些更高级的函数来通过FIFO传递正确格式的数据，并在正确的位置插入Starts、Stops、NAK等这足以为RP 2040上的其他硬件I2C提供类似的接口

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/i2c\_bus\_scan.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/i2c/i2c_bus_scan.c#L13-L42)第13-42

publicint findDuplicate（）{

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42}

stdio\_init\_all（）;

PIO pio = pio0; uint sm =0;

uint offset = pio\_add\_program（pio，&i2c\_program）;

i2c\_program\_init（pio，sm，offset，PIN\_SDA，PIN\_SCL）;

printf（“\nPIO I2C总线扫描\n”）;

printf（“

0 1 2 3 4 5 6 7 8 9 A B C D EF\n”）;

publicint findDuplicate（intfindDuplicate）{if（intfindDuplicate）{<<<

printf（“%02x“，addr）;

}

//从探针地址执行0字节读取读取函数

//返回一个否定的结果NAK'd任何时间以外的最后一个数据

//bytes. 跳过保留地址。

intresult;

if（reserved\_addr（addr））result = -1;

其他

return pio\_i2c\_read\_blocking（pio，sm，addr，NULL，0）;

printf（result0？". ":"@");

printf（addr %16==15？“\n”:“）;

}

printf（“完成。\“）; return 0;

* + 1. PWM

图56. 脉宽调制（PWM）。状态机以规则的间隔输出正这些脉冲的宽度是受控的，使得线在某个受控的时间部分（占空比）内为高。其用途之一是通过比人类视觉持久性更快地脉动LED来平滑地改变LED的亮度。



该程序通过Y寄存器重复计数到0，同时将Y计数与X寄存器中的脉冲宽度进行比较计数开始前，输出置位为低电平，当Y中的值达到X时，输出置位为高电平一旦Y达到0，则重复该过程，并且输出再次被驱动为低。因此，输出为高电平的时间比例与存储在X中的脉冲宽度成比例

*Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/pwm/pwm.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/pwm/pwm.pio#L9-L21)第9-21*

1. .程序PWM
2. .side\_set 1选项

11

1. pullnoblock side 0;如果可用，则从FIFO拉取到OSR，否则将X复制到OSR。
2. mov x，osr;将最近提取的值复制回临时X
3. mov y，isr; ISR包含PWM周期。 Y用作计数器。
4. countloop:

16

17

jmp x！=Y形鼻端

jmpskip

;如果X == Y，则将引脚设置为高电平，保持两个路径长度匹配

侧1

1. 鼻端:
2. nop;单个哑循环，以保持两条路径的长度相同
3. 跳过:
4. jmp y--countloop;循环直到Y达到0，然后从FIFO中提取新的PWM值

通常，PWM可以在数千个脉冲中保持特定的脉冲宽度，而不是每次都提供新的脉冲这个例子强调了一个非阻塞的PINK（[第3.4.7](#_bookmark121)）是如何实现这一点的:如果TX FIFO为空，一个非阻塞的PINK将把X复制到OSR。拉取后，程序将OSR复制到X中，以便与Y中的计数值进行比较净效应是，如果在该周期开始时没有通过TX FIFO提供新的占空比值，则将重新*使用*前一周期的占空比（已通过发生故障的PSNR从X复制到OSR，然后通过MOV复制回X），持续所需的周期数

这里展示的另一个有用的技术是，如果不需要IN指令，则将ISR用作配置寄存器。系统软件可以将任意32位值加载到ISR中（通过直接在状态机上执行指令），并且程序将在每次开始计数时将此值复制到YISR可用于配置PWM计数范围，状态机的时钟分频器控制计数速率。

要开始调制一些脉冲，我们首先需要将状态机的侧置引脚映射到我们想要输出PWM的GPIO，并告诉状态机程序在PIO指令存储器中的加载

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/pwm/pwm.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/pwm/pwm.pio#L24-L30)第24-30

1. public static int findDuplicate（init findDuplicate，initfindDuplicate，int findDuplicate）{
2. pio\_gpio\_init（pio，pin）;
3. pio\_sm\_set\_consequential\_pindirs（pio，sm，pin，1，true）;
4. pio\_sm\_config c =pwm\_program\_get\_default\_config（offset）;
5. sm\_config\_set\_sideset\_pins（c，pin）;
6. pio\_sm\_init（pio，sm，offset，&c）;

30}

需要一点技巧来为ISR加载所需的计数范围:

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/pwm/pwm.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/pwm/pwm.c#L14-L20)第14-20

1. voidpio\_pwm\_set\_period（PIO pio，uint sm， uint32\_tperiod）{
2. pio\_sm\_set\_enabled（pio，sm，false）;
3. pio\_sm\_put\_blocking（pio，sm，period）;
4. pio\_sm\_exec（pio，sm，pio\_encode\_pull（false，false））;
5. pio\_sm\_exec（pio，sm，pio\_encode\_out（pio\_isr，32））;
6. pio\_sm\_set\_enabled（pio，sm，true）;

20}

完成此操作后，可以使能状态机，并将PWM值直接写入其TX FIFO。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/pwm/pwm.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/pwm/pwm.c#L23-L25)第23-25

1. voidpio\_pwm\_set\_level（PIO pio，uint sm， uint32\_tlevel）{
2. pio\_sm\_put\_blocking（pio，sm，level）;

25}

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/pwm/pwm.c](https://github.com/raspberrypi/pico-examples/blob/master/pio/pwm/pwm.c#L27-L51)第27-51

1. public voidrun（）{
2. stdio\_init\_all（）;
3. #ifndefPICO\_DEFAULT\_LED\_PIN
4. #warning pio/pwm example需要一个带有常规LED的电路板
5. puts（“未定义默认LED引脚）;
6. 其他#

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

*//todo get free sm*pio = pio0;intsm =0;

uint offset = pio\_add\_program（pio，&pwm\_program）;

printf（“%d\n处加载的程序，offset）;

pwm\_program\_init（pio，sm，offset，PICO\_DEFAULT\_LED\_PIN）; pio\_pwm\_set\_period（pio，sm，（1u16）-1）;

public intfindDuplicate（）{

printf（“Level = %d\n”，level）; pio\_pwm\_set\_level（pio，sm，level \* level）; level =（level +1）%256;

sleep\_ms（10）;

}

1. #结束
2. }

如果TX FIFO保持充满新的脉冲宽度值，则此程序将为每个脉冲消耗新的脉冲宽度。一旦FIFO耗尽，程序将再次开始重用最近提供的值。

* + 1. 此外

虽然PIO不是为计算而设计的，但它很可能是图灵完备的，只要能找到足够长的磁带据证实，它可以运行毁灭战士，给予足够高的时钟速度。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pio/addition/addition.pio](https://github.com/raspberrypi/pico-examples/blob/master/pio/addition/addition.pio#L7-L25)第7-25

7.程序添加

8

9;从TX FIFO中弹出两个32位整数，将它们加在一起，并将

10;结果发送到TX FIFO。自动推送/拉取应该被禁用，因为我们正在使用

11;明确的push和pull指令。

12人;

13;此程序使用二进制补码恒等式x + y ==~（~x-y）14

1. 拉
2. mov x，~osr
3. 拉
4. 莫韦山
5. jmp测试

20incr:

;这个循环等价于下面的C代码:

; while（i）

jmp x--test ; x--;

22测试:

这最终会产生从x中减去y的效果。

1. jmp y--incr
2. mov isr，~x
3. 推

一个完整的32位加法在125MHz下只需要大约一分钟。该程序从TX FIFO中取出两个数字，并将它们的总和推送到RX FIFO，这非常适合与系统DMA一起使用，或直接由处理器使用:

Pico示例:<https://github.com/raspberrypi/pico-examples/blob/master/pio/addition/addition.c>

1*/\*\**

1. \*版权所有（c）2020 Raspberry Pi（Trading）Ltd.
2. \*
3. \* SPDX-License-Identifier:BSD-3-Clause
4. \*/

6

1. #include<stdlib. h>
2. #include<stdio.h>

9

1. #include“pico/stdlib. h”
2. #include“hardware/pio.h”
3. #include“addition.pio.h”

13

1. //突击测试:处理器在调用此函数时做了多少加法
2. uint32\_tdo\_addition（PIO pio，uint sm， uint32\_ta， uint32\_tb）{
3. pio\_sm\_put\_blocking（pio，sm，a）;
4. pio\_sm\_put\_blocking（pio，sm，b）;
5. returnpio\_sm\_get\_blocking（pio，sm）;

19}

20

publicint findDuplicate（）{

22

23

24

25

26

27

28

29

30

31

32

33

34

35}

stdio\_init\_all（）;

PIO pio = pio0; uint sm =0;

uint offset = pio\_add\_program（pio，addition\_program）; addition\_program\_init（pio，sm，offset）;

printf（“做一些随机加法:\n”）;for（inti =0; i10; ++i）{

int a = rand（）%100; int b = rand（）%100;

printf（“%u + %u = %u\n”，a，b，do\_addition（pio，sm，a，b））;

}

* + 1. 另外的示例

《**[Raspberry Pi Pico C/C++ SDK](https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf)》**一书中有一个PIO章节，深入介绍了一些这里没有介绍的以软件为中心的主题。它包括一个PIO+DMA逻辑分析仪示例，可以在每个周期（带宽）上对每个GPIO进行

在125MHz时接近4Gbps，尽管这确实会很快填满RP 2040的RAM）。PicoExamples存储库的pio/目录中还有更多示例。

一些更具实验性的示例代码，如DPI和SD卡支持，目前位于[Pico Extras](https://github.com/raspberrypi/pico-extras)和[Pico Playground](https://github.com/raspberrypi/pico-playground)存储库中其中的PIO部分是功能性的，但周围的软件栈仍处于实验状态。

* 1. 登记册一览表

*367号桌 PIO寄存器*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x000 | [Ctrl](#_bookmark133) | PIO控制寄存器 |
| 0x004 | [FSTAT](#_bookmark134) | FIFO状态寄存器 |
| 0x008 | [FDEBUG](#_bookmark135) | FIFO调试寄存器 |
| 0x00c | [FLEVEL](#_bookmark136) | FIFO电平 |
| 0x010 | [TXF0](#_bookmark137) | 直接写访问此状态机的TX FIFO 每次写入将一个字推入FIFO。尝试写入已满FIFO不会影响FIFO状态或内容，并会为此FIFO设置 |
| 0x014 | [TXF1](#_bookmark137) | 直接写访问此状态机的TX FIFO 每次写入将一个字推入FIFO。尝试写入已满FIFO不会影响FIFO状态或内容，并会为此FIFO设置 |
| 0x018 | [TXF2](#_bookmark137) | 直接写访问此状态机的TX FIFO 每次写入将一个字推入FIFO。尝试写入已满FIFO不会影响FIFO状态或内容，并会为此FIFO设置 |
| 0x01c | [TXF3](#_bookmark137) | 直接写访问此状态机的TX FIFO 每次写入将一个字推入FIFO。尝试写入已满FIFO不会影响FIFO状态或内容，并会为此FIFO设置 |
| 0x020 | [RXF0](#_bookmark138) | 直接读取此状态机的RX FIFO每次读取都会从FIFO中弹出一个字尝试从空FIFO中读取不会影响FIFO状态，并会为此FIFO设置粘性FDEBUG\_RXUNDER错误标志从空FIFO读取时返回系统的数据 |
| 0x024 | [RXF1](#_bookmark138) | 直接读取此状态机的RX FIFO每次读取都会从FIFO中弹出一个字尝试从空FIFO中读取不会影响FIFO状态，并会为此FIFO设置粘性FDEBUG\_RXUNDER错误标志从空FIFO读取时返回系统的数据 |
| 0x028 | [RXF2](#_bookmark138) | 直接读取此状态机的RX FIFO每次读取都会从FIFO中弹出一个字尝试从空FIFO读取不会影响FIFO状态，并会设置此FIFO的粘性FDEBUG\_RXUNDER错误标志从空FIFO读取时返回系统的数据 |

PIO0和PIO1寄存器分别从基址0x50200000和0x50300000开始（在SDK中定义为PIO0\_BASE和PIO1\_BASE）。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x02c | [RXF3](#_bookmark138) | 直接读取此状态机的RX FIFO每次读取都会从FIFO中弹出一个字尝试从空FIFO读取不会影响FIFO状态，并会设置此FIFO的粘性FDEBUG\_RXUNDER错误标志从空FIFO读取时返回系统的数据 |
| 0x030 | [IRQ](#_bookmark139) | 状态机IRQ标志寄存器。写1为清除。有8个状态机IRQ标志，可以由状态机设置、清除和等待在标志和状态机之间没有固定的关联-任何状态机都可以使用任何标志。  8个标志中的任何一个都可以用于状态机之间的定时同步，使用IRQ和WAIT指令。这些标志中较低的四个也会路由到系统级中断请求，以及FIFO状态中断-参见例如IRQ 0\_INTE。 |
| 0x034 | [IRQ\_FORCE](#_bookmark140) | 将1写入这些位将强制置位相应的IRQ。注意，这与INTF寄存器不同:在这里写入会影响PIO内部状态。INTF只是断言面向处理器的IRQ信号，用于测试ISR，对状态机不可见 |
| 0x038 | [输入\_同步\_旁路](#_bookmark141) | 在每个GPIO输入上有一个2触发器同步器，可保护PIO逻辑不受干扰。这增加了输入延迟，并且对于快速同步IO（例如，SPI）这些同步器可能需要被旁路。此寄存器中的每个位对应于一个GPIO。   1. →输入同步（默认） 2. →同步器被旁路   如果有疑问，请将此寄存器保留为全零。 |
| 0x03c | [DBG\_PADOUT](#_bookmark142) | 读取以采样PIO当前驱动至GPIO的焊盘输出值在RP2040上有30个GPIO，因此两个最高有效位被硬连线为0。 |
| 0x040 | [DBG\_PADOE](#_bookmark143) | 读取以采样焊盘输出使能（方向）PIO当前正驱动至GPIO。在RP2040上有30个GPIO，因此两个最高有效位被硬连线为0。 |
| 0x044 | [DBG\_CFGINFO](#_bookmark144) | PIO硬件有一些自由参数，这些参数可能因芯片产品而异。  这些应该在芯片封装中提供，但也暴露在这里。 |
| 0x048 | [INSTR\_MEM0](#_bookmark145) | 只写访问指令存储器位置0 |
| 0x04c | [INSTR\_MEM1](#_bookmark145) | 只写访问指令存储器位置1 |
| 0x050 | [INSTR\_MEM2](#_bookmark145) | 只写访问指令存储器位置2 |
| 0x054 | [INSTR\_MEM3](#_bookmark145) | 只写访问指令存储器位置3 |
| 0x058 | [INSTR\_MEM4](#_bookmark145) | 只写访问指令存储器位置4 |
| 0x05c | [INSTR\_MEM5](#_bookmark145) | 只写访问指令存储器位置5 |
| 0x060 | [INSTR\_MEM6](#_bookmark145) | 只写访问指令存储器位置6 |
| 0x064 | [INSTR\_MEM7](#_bookmark145) | 只写访问指令存储器位置7 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x068 | [INSTR\_MEM8](#_bookmark145) | 只写访问指令存储器位置8 |
| 0x06c | [INSTR\_MEM9](#_bookmark145) | 只写访问指令存储器位置9 |
| 0x070 | [INSTR\_MEM 10](#_bookmark145) | 对指令存储器位置10的只写访问 |
| 0x074 | [INSTR\_MEM 11](#_bookmark145) | 只写访问指令存储器位置11 |
| 0x078 | [INSTR\_MEM 12](#_bookmark145) | 对指令存储器位置12的只写访问 |
| 0x07c | [INSTR\_MEM13](#_bookmark145) | 对指令存储器位置13的只写访问 |
| 0x080 | [INSTR\_MEM 14](#_bookmark145) | 对指令存储器位置14的只写访问 |
| 0x084 | [INSTR\_MEM 15](#_bookmark145) | 只写访问指令存储器位置15 |
| 0x088 | [INSTR\_MEM 16](#_bookmark145) | 对指令存储器位置16的只写访问 |
| 0x08c | [INSTR\_MEM 17](#_bookmark145) | 对指令存储器位置17的只写访问 |
| 0x090 | [INSTR\_MEM 18](#_bookmark145) | 对指令存储器位置18的只写访问 |
| 0x094 | [INSTR\_MEM 19](#_bookmark145) | 只写访问指令存储器位置19 |
| 0x098 | [INSTR\_MEM20](#_bookmark145) | 对指令存储器位置20的只写访问 |
| 0x09c | [INSTR\_MEM 21](#_bookmark145) | 只写访问指令存储器位置21 |
| 0x0a0 | [INSTR\_MEM 22](#_bookmark145) | 对指令存储器位置22的只写访问 |
| 0x0a4 | [INSTR\_MEM 23](#_bookmark145) | 对指令存储器位置23的只写访问 |
| 0x0a8 | [INSTR\_MEM 24](#_bookmark145) | 对指令存储器位置24的只写访问 |
| 0x0ac | [INSTR\_MEM 25](#_bookmark145) | 只写访问指令存储器位置25 |
| 0x0b0 | [INSTR\_MEM 26](#_bookmark145) | 对指令存储器位置26的只写访问 |
| 0x0b4 | [INSTR\_MEM 27](#_bookmark145) | 对指令存储器位置27的只写访问 |
| 0x0b8 | [INSTR\_MEM 28](#_bookmark145) | 对指令存储器位置28的只写访问 |
| 0x0bc | [INSTR\_MEM 29](#_bookmark145) | 只写访问指令存储器位置29 |
| 0x0c0 | [INSTR\_MEM30](#_bookmark145) | 对指令存储器位置30的只写访问 |
| 0x0c4 | [INSTR\_MEM31](#_bookmark145) | 对指令存储器位置31的只写访问 |
| 0x0c8 | [SM0\_CLKDIV](#_bookmark146) | 状态机0的时钟除数寄存器  频率=时钟频率/（CLKDIV\_INT + CLKDIV\_FRAC/256） |
| 0x0cc | [SM0\_EXECCTRL](#_bookmark147) | 状态机0的执行/行为设置 |
| 0x0d0 | [SM0\_SHIFTCTRL](#_bookmark148) | 状态机0的输入/输出移位寄存器的控制行为 |
| 0x0d4 | [SM0\_ADDR](#_bookmark149) | 状态机0的当前指令地址 |
| 0x0d8 | [SM0\_BURR](#_bookmark150) | 读取以查看状态机0的程序计数器当前寻址的指令  写入以立即执行指令（包括跳转），然后继续执行。 |
| 0x0dc | [SM0\_PINCTRL](#_bookmark151) | 状态机引脚控制 |
| 0x0e0 | [SM1\_CLKDIV](#_bookmark146) | 状态机1的时钟除数寄存器  频率=时钟频率/（CLKDIV\_INT + CLKDIV\_FRAC/256） |
| 0x0e4 | [SM1\_EXECCTRL](#_bookmark147) | 状态机1的执行/行为设置 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x0e8 | [SM1\_SHIFTCTRL](#_bookmark148) | 状态机1的输入/输出移位寄存器的控制行为 |
| 0x0ec | [SM1\_ADDR](#_bookmark149) | 状态机1的当前指令地址 |
| 0x0f0 | [SM1\_BURR](#_bookmark150) | 读取状态机1的程序计数器当前寻址的指令  写入以立即执行指令（包括跳转），然后继续执行。 |
| 0x0f4 | [SM1\_PINCTRL](#_bookmark151) | 状态机引脚控制 |
| 0x0f8 | [SM2\_CLKDIV](#_bookmark146) | 状态机2的时钟除数寄存器  频率=时钟频率/（CLKDIV\_INT + CLKDIV\_FRAC/256） |
| 0x0fc | [SM2\_EXECCTRL](#_bookmark147) | 状态机2的执行/行为设置 |
| 0x100 | [SM2\_SHIFTCTRL](#_bookmark148) | 状态机2的输入/输出移位寄存器的控制行为 |
| 0x104 | [SM2\_ADDR](#_bookmark149) | 状态机2的当前指令地址 |
| 0x108 | [SM2\_BURR](#_bookmark150) | 读取状态机2的程序计数器当前寻址的指令  写入以立即执行指令（包括跳转），然后继续执行。 |
| 0x10c | [SM2\_PINCTRL](#_bookmark151) | 状态机引脚控制 |
| 0x110 | [SM3\_CLKDIV](#_bookmark146) | 状态机3的时钟除数寄存器  频率=时钟频率/（CLKDIV\_INT + CLKDIV\_FRAC/256） |
| 0x114 | [SM3\_EXECCTRL](#_bookmark147) | 状态机3的执行/行为设置 |
| 0x118 | [SM3\_SHIFTCTRL](#_bookmark148) | 状态机3的输入/输出移位寄存器的控制行为 |
| 0x11c | [SM3\_ADDR](#_bookmark149) | 状态机3的当前指令地址 |
| 0x120 | [SM3\_CQLR](#_bookmark150) | 读取状态机3的程序计数器当前寻址的指令  写入以立即执行指令（包括跳转），然后继续执行。 |
| 0x124 | [SM3\_PINCTRL](#_bookmark151) | 状态机引脚控制 |
| 0x128 | [INTR](#_bookmark152) | 原始中断 |
| 0x12c | [IRQ0\_INTE](#_bookmark153) | 启用irq 0 |
| 0x130 | [IRQ0\_INTF](#_bookmark154) | irq0的反作用力 |
| 0x134 | [IRQ0\_INTS](#_bookmark155) | irq0强制屏蔽后的屏蔽状态 |
| 0x138 | [IRQ1\_INTE](#_bookmark156) | 启用irq1 |
| 0x13c | [IRQ1\_INTF](#_bookmark157) | irq1的反作用力 |
| 0x140 | [IRQ1\_INTS](#_bookmark158) | irq1强制屏蔽后的屏蔽状态 |

#### [PIO](#_bookmark132):CTRL寄存器

**偏移**:0x000

368号桌CTRL寄存器

表369。FSTAT

寄存器

描述

PIO控制寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - | - |
| 十一点八分 | CLKDIV\_重新启动 | 从初始相位0重新启动状态机的时钟分频器时钟分频器是自由运行的，因此一旦启动，其输出（包括小数抖动）完全由SMx\_CLKDIV中配置的整数/小数分频器决定这意味着，如果同时重启具有相同除数的多个时钟分频器，则通过向该字段写入多个1位，这些状态机的执行时钟将以精确的锁步运行。  请注意，设置/清除SM\_ENABLE不会停止时钟分频器的运行，因此一旦多个状态机的时钟同步，可以安全地禁用/重新启用状态机，同时保持时钟分频器同步。  还请注意，CLKDIV\_RESTART可以在状态机运行时写入 | SC | 0x0 |
| 七比四 | SM\_重新启动 | 写入1以立即清除内部SM状态，否则可能难以访问该状态并将影响未来的执行。  具体而言，以下内容将被清除:输入和输出移位计数器;输入移位寄存器的内容;延迟计数器;等待IRQ状态;任何写入SMx\_STICKR或由OUT/MOV EXEC运行的延迟指令;因OUT\_STICKY而置位的任何引脚写入。  程序计数器、输出移位寄存器和X/Y暂存寄存器的内容不受影响。 | SC | 0x0 |
| 三比零 | SM\_ENABLE | 通过将1/0写入这四位中的每一位来启用/禁用四个状态机中的每一个禁用时，状态机将停止执行指令，但由系统直接写入SMx\_SERR的指令除外。可以一次设置/清除多个位，以同时运行/停止多个状态机 | RW | 0x0 |

#### [PIO](#_bookmark132):FSTAT寄存器

**偏移量**:0x004

描述

FIFO状态寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:28 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 27:24 | TXEMPTY | 状态机TX FIFO为空 | RO | 0xf |
| 23:20 | Reserved. | - | - | - |
| 十九点十六分 | TXFULL | 状态机TX FIFO已满 | RO | 0x0 |
| 十五点十二分 | Reserved. | - | - | - |
| 十一点八分 | RXEMPTY | 状态机RX FIFO为空 | RO | 0xf |
| 七比四 | Reserved. | - | - | - |
| 三比零 | RXFULL | 状态机RX FIFO已满 | RO | 0x0 |

表370。FDEBUG

寄存器

#### [PIO](#_bookmark132):FDEBUG注册

**偏移量**:0x008

描述

FIFO调试寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:28 | Reserved. | - | - | - |
| 27:24 | TXSTALL | 状态机在阻塞PIND期间在空TX FIFO上停止，或在启用自动清零的OUT期间停止。 写1为清除。 | WC | 0x0 |
| 23:20 | Reserved. | - | - | - |
| 十九点十六分 | TXOVER | TX FIFO溢出（即系统已写满）。 写1为清除。请注意，写满不会以任何方式改变FIFO的状态或内容，但系统尝试写入的数据会被丢弃，因此如果设置了此标志，则您的软件很可能会丢弃一些数据。 | WC | 0x0 |
| 十五点十二分 | Reserved. | - | - | - |
| 十一点八分 | RXUNDER | RXFIFO下溢（即系统空读）。写1为清除。请注意，空时读取不会以任何方式干扰FIFO的状态，但从空FIFO读取返回的数据是未定义的，因此该标志通常只会由于某种软件错误而被设置 | WC | 0x0 |
| 七比四 | Reserved. | - | - | - |
| 三比零 | RXSTALL | 在阻塞推送或启用自动推送的IN期间，当非阻塞PUSH到满FIFO发生时，也会设置此标志，在这种情况下，状态机已丢弃数据。写1为清除。 | WC | 0x0 |

#### [PIO](#_bookmark132):FLEVEL寄存器

**偏移**:0x00c

描述

FIFO电平

*表371。FLEVEL*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:28 | RX3 |  | RO | 0x0 |
| 27:24 | TX3 |  | RO | 0x0 |
| 23:20 | RX2 |  | RO | 0x0 |
| 十九点十六分 | TX2 |  | RO | 0x0 |
| 十五点十二分 | RX1 |  | RO | 0x0 |
| 十一点八分 | TX1 |  | RO | 0x0 |
| 七比四 | RX0 |  | RO | 0x0 |
| 三比零 | TX0 |  | RO | 0x0 |

寄存器

表372。TXF0、TXF1、TXF2、TXF3

寄存器

表373。RXF0、RXF1、RXF2、RXF3

寄存器

374号表IRQ寄存器

#### [PIO](#_bookmark132):TXF0、TXF1、TXF2、TXF3寄存器

**偏移**:0x010、0x014、0x018、0x01c

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 三11:0 | 直接写访问此状态机的TX FIFO每次写入将一个字推入FIFO。 尝试写入已满FIFO不会影响FIFO状态或内容，并会为此FIFO设置粘性FDEBUG\_TXOVER错误标志。 | WF | 0x00000000 |

#### [PIO](#_bookmark132):RXF0、RXF1、RXF2、RXF3寄存器

**偏移**:0x020、0x024、0x028、0x02c

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 三11:0 | 直接读取访问此状态机的RX FIFO每次读取都会从FIFO中弹出一个字尝试从空FIFO读取不会影响FIFO状态，并会设置此FIFO的粘性FDEBUG\_RXUNDER错误标志从空FIFO读取时返回系统的数据是未定义的。 | RF | - |

#### IRQ寄存器

**偏移**:0x030

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - |
| 7:0 | 状态机IRQ标志寄存器。 写1为清除。有8个状态机IRQ标志，可以由状态机设置、清除和等待标志和状态机之间没有固定的关联-任何状态机都可以使用任何标志。  8个标志中的任何一个都可以用于状态机之间的定时同步，使用IRQ和WAIT指令。这些标志中较低的四个也会路由到系统级中断请求，以及FIFO状态中断-参见例如IRQ 0\_INTE。 | WC | 0x00 |

#### [PIO](#_bookmark132):IRQ\_FORCE寄存器

**偏移量**:0x034

*375号表IRQ\_FORCE*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - |
| 7:0 | 将1写入这些位将强制置位相应的IRQ。注意，这与INTF寄存器不同:在这里写入会影响PIO内部状态。INTF只是断言面向处理器的IRQ信号，用于测试ISR，对状态机不可见。 | WF | 0x00 |

寄存器

376号表输入\_同步\_旁路

寄存器

377号桌DBG\_PADOUT寄存器

378号桌DBG\_PADOE寄存器

表379。DBG\_CFGINFO

寄存器

#### [PIO](#_bookmark132):INPUT\_SYNC\_BYPASS寄存器

**偏移量**:0x038

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 三11:0 | 在每个GPIO输入上有一个2触发器同步器，可保护PIO逻辑不受干扰。这增加了输入延迟，并且对于快速同步IO（例如SPI），这些同步器可能需要被旁路。此寄存器中的每个位对应于一个GPIO。   1. →输入同步（默认） 2. →同步器被旁路   如果有疑问，请将此寄存器保留为全零。 | RW | 0x00000000 |

#### [PIO](#_bookmark132):DBG\_PADOUT寄存器

**偏移**:0x03c

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 三11:0 | 读取以采样PIO当前驱动至GPIO的焊盘输出值在RP2040上有30个GPIO，因此两个最高有效位被硬连线为0。 | RO | 0x00000000 |

#### [PIO](#_bookmark132):DBG\_PADOE寄存器

**偏移**:0x040

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 三11:0 | 读取以采样焊盘输出使能（方向）PIO当前正驱动至GPIO。在RP2040上有30个GPIO，因此两个最高有效位被硬连线为0。 | RO | 0x00000000 |

#### [PIO](#_bookmark132):DBG\_CFGINFO寄存器

**偏移量**:0x044

描述

PIO硬件有一些自由参数，这些参数可能因芯片产品而异这些应该在芯片封装中提供，但也暴露在这里。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一时二十二分 | Reserved. | - | - | - |
| 二十一点十六分 | IMEM\_尺寸 | 指令存储器的大小，以一条指令为单位来度量 | RO | - |
| 十五点十二分 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 十一点八分 | 联系我们 | 此PIO实例配备的状态机数量 | RO | - |
| 7:6 | Reserved. | - | - | - |
| 5:0 | 先进先出深度 | 状态机TX/RX FIFO的深度，以字为单位。  通过SHIFTCTRL\_FJOIN连接FIFO，  这个深度。 | RO | - |

380号桌INSTR\_MEM0、INSTR\_MEM1、.、INSTR\_MEM30、INSTR\_MEM31

寄存器

表381。SM0\_CLKDIV、SM1\_CLKDIV、SM2\_CLKDIV、SM3\_CLKDIV

寄存器

表382。SM0\_EXECCTRL、SM1\_EXECCTRL、SM2\_EXECCTRL、SM3\_EXECCTRL

寄存器

#### [PIO](#_bookmark132):INSTR\_MEM 0、INSTR\_MEM 1、.、INSTR\_MEM 30、INSTR\_MEM 31寄存器

**偏移量**:0x048、0x04c、...、0x0c0、0x0c4

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - |
| 十5:0 | 只写访问指令存储器位置*N* | WO | 0x0000 |

#### [PIO](#_bookmark132):SM0\_CLKDIV、SM1\_CLKDIV、SM2\_CLKDIV、SM3\_CLKDIV寄存器

**偏移**:0x0c8、0x0e0、0x0f8、0x110

描述

状态机*N*的时钟除数寄存器

频率=时钟频率/（CLKDIV\_INT + CLKDIV\_FRAC/256）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | INT | 有效频率为sysclk/（int + frac/256）。  0的值被解释为65536。如果INT为0，则FRAC也必须为0。 | RW | 0x0001 |
| 十五点八分 | FRAC | 时钟除数的小数部分 | RW | 0x00 |
| 7:0 | Reserved. | - | - | - |

#### [PIO](#_bookmark132):SM0\_EXECCTRL、SM1\_EXECCTRL、SM2\_EXECCTRL、SM3\_EXECCTRL

寄存器

**偏移**:0x0cc、0x0e4、0x0fc、0x114

描述

状态机*N*的执行/行为设置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | 执行\_停止 | 如果为1，则写入SMx\_ADRR的指令被暂停，并被状态机锁存将在此指令完成后清除为0 | RO | 0x0 |
| 30 | 侧面\_EN | 如果为1，则延迟/侧置指令字段的MSB用作侧置使能，而不是侧置数据位。这允许指令可选地执行边集，而不是对每个指令执行边集，但最大可能的边­集宽度从5减少到4。请注意，PINCTRL\_SIDESET\_RESET的值 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 29 | 侧面\_PINDIR | 如果为1，则侧置位数据被置位到引脚方向，而不是引脚值 | RW | 0x0 |
| 二十八点二十四分 | JMP\_PIN | 用作JMP PIN条件的GPIO编号不受输入映射的影响。 | RW | 0x00 |
| 二十三点十九分 | 输出\_EN\_选择 | 哪个数据位用于内联OUT使能 | RW | 0x00 |
| 18 | INLINE\_OUT\_EN | 如果为1，则使用OUT数据位作为辅助写入使能当与OUT\_STICKY结合使用时，使能为0的写入  取消断言最新的引脚写入。这可以创建有用的掩蔽/覆盖行为  由于状态机引脚写入的优先级排序（SM0 SM1.） | RW | 0x0 |
| 17 | 出粘 | 持续将最新OUT/SET置位到引脚 | RW | 0x0 |
| 十六点十二分 | 包装\_顶部 | 到达这个地址后，执行被包装到wrap\_bottom。  如果指令是跳转，并且跳转条件为真，则跳转优先。 | RW | 0x1f |
| 十一点七分 | 包装\_底部 | 在到达wrap\_top之后，执行被包装到这个地址。 | RW | 0x00 |
| 六分之五 | Reserved. | - | - | - |
| 4 | 状态\_选择 | 用于MOV x，STATUS指令的比较。0x0→如果TXFIFO电平为N，则全1，否则全0 0x1→如果RXFIFO电平为N，则全1，否则全0 | RW | 0x0 |
| 三比零 | 状态\_N | MOV x，STATUS指令的比较电平 | RW | 0x0 |

383号表SM0\_SHIFTCTRL、SM1\_SHIFTCTRL、SM2\_SHIFTCTRL、SM3\_SHIFTCTRL

寄存器

#### [PIO](#_bookmark132):SM0\_SHIFTCTRL、SM1\_SHIFTCTRL、SM2\_SHIFTCTRL、SM3\_SHIFTCTRL

寄存器

**偏移**:0x0d0、0x0e8、0x100、0x118

描述

状态机*N*的输入/输出移位寄存器的控制行为

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | FJOIN\_RX | 当为1时，RX FIFO窃取TX FIFO的存储，并且变为两倍深。  因此，TX FIFO被禁用（始终读取为满和空）。  当该位改变时，FIFO被刷新。 | RW | 0x0 |
| 30 | FJOIN\_TX | 当为1时，TX FIFO窃取RX FIFO的存储，并且变得两倍深。  RX FIFO因此被禁用（始终读取为满和空）。  当该位改变时，FIFO被刷新。 | RW | 0x0 |
| 二十九点二十五分 | Pingle\_Thresh | 在发生自动清除或条件拉取（PINEIFEMPTY）之前，从OSR移出的位数  将值32写为0。 | RW | 0x00 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 二十四点二十分 | PUSH\_THRESH | 在自动推送或条件推送（PUSH IFFULL）发生之前，移入ISR的位数  将值32写为0。 | RW | 0x00 |
| 19 | 输出移位 | 1 =从输出移位寄存器移出到右。0 =向左。 | RW | 0x1 |
| 18 | IN\_SHIFTDIR | 1=将输入移位寄存器向右移位（数据从左进入）。0 =向左。 | RW | 0x1 |
| 17 | 自动化 | 输出移位寄存器为空时自动拉，即在OUT指令上或之后，该指令导致输出移位计数器达到或超过PINK\_THRESH。 | RW | 0x0 |
| 16 | 自动推送 | 当输入移位寄存器满时自动按下  即在导致输入移位计数器达到或超过PUSH\_THRESH的IN指令 | RW | 0x0 |
| 十5:0 | Reserved. | - | - | - |

384号桌SM0\_ADDR、SM1\_ADDR、SM2\_ADDR、

SM3\_ADDR寄存器

385号桌SM0\_ADIR、SM1\_ADIR、SM2\_ADIR、

SM3\_ADIR寄存器

386号桌SM0\_PINCTRL、SM1\_PINCTRL、SM2\_PINCTRL、SM3\_PINCTRL

寄存器

#### [PIO](#_bookmark132):SM0\_ADDR、SM1\_ADDR、SM2\_ADDR、SM3\_ADDR寄存器

**偏移**:0x0d4、0x0ec、0x104、0x11c

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 三十一比五 | Reserved. | - | - |
| 4:0 | 状态机*N*的当前指令地址 | RO | 0x00 |

#### [PIO](#_bookmark132):SM0\_ADIR、SM1\_ADIR、SM2\_ADIR、SM3\_ADIR寄存器

**偏移**:0x0d8、0x0f0、0x108、0x120

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - |
| 十5:0 | 读取以查看状态机*N*的程序计数器当前寻址的指令。  写入以立即执行指令（包括跳转），然后继续执行。 | RW | - |

#### [PIO](#_bookmark132):SM0\_PINCTRL、SM1\_PINCTRL、SM2\_PINCTRL、SM3\_PINCTRL寄存器

**偏移**:0x0dc、0x0f4、0x10c、0x124

描述

状态机引脚控制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一点二十九分 | 侧集\_计数 | 延迟/侧置指令字段中用于侧置的MSB数。 包括使能位（如果存在）。最小值为0（所有延迟位，无边集），最大值为5（所有边集，无延迟）。 | RW | 0x0 |
| 二十八点二十六分 | 联系我们 | SET置位的引脚数 在0到5的范围内，包括0和5。 | RW | 0x5 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 二十五点二十分 | OUT\_CHARACTER | OUT PINS、OUT PINDIRS或MOV PINS指令置位的引脚数。 在0到32的范围内，包括0和32。 | RW | 0x00 |
| 十九点十五分 | IN\_BASE | 映射到状态机IN数据总线最低有效位的引脚较高编号的引脚映射到连续的更高有效数据位，并对引脚编号应用模32。 | RW | 0x00 |
| 十四点十分 | 侧置\_底座 | 将受边设置操作影响的编号最小的引脚指令的边集/延迟字段的MSB（最多5个，由SIDESET\_REQ确定）用于边集数据，其余LSB用于延迟。侧集部分的最低有效位是写入此引脚的位，较高有效位写入编号较高的引脚。 | RW | 0x00 |
| 九点五 | 设置\_基础 | 受SET PINS或SET PINDIRS指令影响的编号最小的引脚写入此引脚的数据是SET数据的最低有效位 | RW | 0x00 |
| 4:0 | OUT\_BASE | 受OUT PINS、OUT PINDIRS或MOV PINS指令影响的编号最小的引脚写入此引脚的数据始终是OUT或MOV数据的最低有效位 | RW | 0x00 |

387号桌INTR寄存器

#### [PIO](#_bookmark132):INTR寄存器

**偏移**:0x128

描述

原始中断

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - | - |
| 11 | SM3 |  | RO | 0x0 |
| 10 | SM2 |  | RO | 0x0 |
| 9 | SM1 |  | RO | 0x0 |
| 8 | SM0 |  | RO | 0x0 |
| 7 | SM3\_TXNFULL |  | RO | 0x0 |
| 6 | SM2\_TXNFULL |  | RO | 0x0 |
| 5 | SM1\_TXNFULL |  | RO | 0x0 |
| 4 | SM0\_TXNFULL |  | RO | 0x0 |
| 3 | SM3\_RXNEMPTY |  | RO | 0x0 |
| 2 | SM2\_RXNEMPTY |  | RO | 0x0 |
| 1 | SM1\_RXNEMPTY |  | RO | 0x0 |
| 0 | SM0\_RXNEMPTY |  | RO | 0x0 |

#### [PIO](#_bookmark132):IRQ0\_INTE寄存器

388号桌IRQ0\_INTE

寄存器

389号表IRQ0\_INTF

寄存器

**偏移**:0x12c

描述

启用irq 0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - | - |
| 11 | SM3 |  | RW | 0x0 |
| 10 | SM2 |  | RW | 0x0 |
| 9 | SM1 |  | RW | 0x0 |
| 8 | SM0 |  | RW | 0x0 |
| 7 | SM3\_TXNFULL |  | RW | 0x0 |
| 6 | SM2\_TXNFULL |  | RW | 0x0 |
| 5 | SM1\_TXNFULL |  | RW | 0x0 |
| 4 | SM0\_TXNFULL |  | RW | 0x0 |
| 3 | SM3\_RXNEMPTY |  | RW | 0x0 |
| 2 | SM2\_RXNEMPTY |  | RW | 0x0 |
| 1 | SM1\_RXNEMPTY |  | RW | 0x0 |
| 0 | SM0\_RXNEMPTY |  | RW | 0x0 |

#### [PIO](#_bookmark132):IRQ 0\_INTF寄存器

**偏移**:0x130

描述

irq0的反作用力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - | - |
| 11 | SM3 |  | RW | 0x0 |
| 10 | SM2 |  | RW | 0x0 |
| 9 | SM1 |  | RW | 0x0 |
| 8 | SM0 |  | RW | 0x0 |
| 7 | SM3\_TXNFULL |  | RW | 0x0 |
| 6 | SM2\_TXNFULL |  | RW | 0x0 |
| 5 | SM1\_TXNFULL |  | RW | 0x0 |
| 4 | SM0\_TXNFULL |  | RW | 0x0 |
| 3 | SM3\_RXNEMPTY |  | RW | 0x0 |
| 2 | SM2\_RXNEMPTY |  | RW | 0x0 |
| 1 | SM1\_RXNEMPTY |  | RW | 0x0 |
| 0 | SM0\_RXNEMPTY |  | RW | 0x0 |

#### [PIO](#_bookmark132):IRQ 0\_INTS寄存器

表390。IRQ0\_INTS

寄存器

表391.IRQ1\_INTE

寄存器

**偏移**:0x134

描述

irq0强制屏蔽后的屏蔽状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - | - |
| 11 | SM3 |  | RO | 0x0 |
| 10 | SM2 |  | RO | 0x0 |
| 9 | SM1 |  | RO | 0x0 |
| 8 | SM0 |  | RO | 0x0 |
| 7 | SM3\_TXNFULL |  | RO | 0x0 |
| 6 | SM2\_TXNFULL |  | RO | 0x0 |
| 5 | SM1\_TXNFULL |  | RO | 0x0 |
| 4 | SM0\_TXNFULL |  | RO | 0x0 |
| 3 | SM3\_RXNEMPTY |  | RO | 0x0 |
| 2 | SM2\_RXNEMPTY |  | RO | 0x0 |
| 1 | SM1\_RXNEMPTY |  | RO | 0x0 |
| 0 | SM0\_RXNEMPTY |  | RO | 0x0 |

#### [PIO](#_bookmark132):IRQ1\_INTE寄存器

**偏移**:0x138

描述

启用irq1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - | - |
| 11 | SM3 |  | RW | 0x0 |
| 10 | SM2 |  | RW | 0x0 |
| 9 | SM1 |  | RW | 0x0 |
| 8 | SM0 |  | RW | 0x0 |
| 7 | SM3\_TXNFULL |  | RW | 0x0 |
| 6 | SM2\_TXNFULL |  | RW | 0x0 |
| 5 | SM1\_TXNFULL |  | RW | 0x0 |
| 4 | SM0\_TXNFULL |  | RW | 0x0 |
| 3 | SM3\_RXNEMPTY |  | RW | 0x0 |
| 2 | SM2\_RXNEMPTY |  | RW | 0x0 |
| 1 | SM1\_RXNEMPTY |  | RW | 0x0 |
| 0 | SM0\_RXNEMPTY |  | RW | 0x0 |

#### [PIO](#_bookmark132):IRQ1\_INTF寄存器

表392。IRQ1\_INTF

寄存器

表393。IRQ1\_INTS

寄存器

**偏移**:0x13c

描述

irq1的反作用力

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - | - |
| 11 | SM3 |  | RW | 0x0 |
| 10 | SM2 |  | RW | 0x0 |
| 9 | SM1 |  | RW | 0x0 |
| 8 | SM0 |  | RW | 0x0 |
| 7 | SM3\_TXNFULL |  | RW | 0x0 |
| 6 | SM2\_TXNFULL |  | RW | 0x0 |
| 5 | SM1\_TXNFULL |  | RW | 0x0 |
| 4 | SM0\_TXNFULL |  | RW | 0x0 |
| 3 | SM3\_RXNEMPTY |  | RW | 0x0 |
| 2 | SM2\_RXNEMPTY |  | RW | 0x0 |
| 1 | SM1\_RXNEMPTY |  | RW | 0x0 |
| 0 | SM0\_RXNEMPTY |  | RW | 0x0 |

#### [PIO](#_bookmark132):IRQ1\_INTS寄存器

**偏移**:0x140

描述

irq1强制屏蔽后的屏蔽状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - | - |
| 11 | SM3 |  | RO | 0x0 |
| 10 | SM2 |  | RO | 0x0 |
| 9 | SM1 |  | RO | 0x0 |
| 8 | SM0 |  | RO | 0x0 |
| 7 | SM3\_TXNFULL |  | RO | 0x0 |
| 6 | SM2\_TXNFULL |  | RO | 0x0 |
| 5 | SM1\_TXNFULL |  | RO | 0x0 |
| 4 | SM0\_TXNFULL |  | RO | 0x0 |
| 3 | SM3\_RXNEMPTY |  | RO | 0x0 |
| 2 | SM2\_RXNEMPTY |  | RO | 0x0 |
| 1 | SM1\_RXNEMPTY |  | RO | 0x0 |
| 0 | SM0\_RXNEMPTY |  | RO | 0x0 |

第4章.外围设备

* 1. USB
     1. 概述

所需的先决知识

本节要求了解USB协议。如果您不清楚本节中使用的术语，我们建议您使用[[usbmadesomple]](#_bookmark188)（参见[参考文献](#_bookmark187)）。

RP2040包含一个USB 2.0控制器，可以作为以下两种方式之一运行

* 全速设备（12Mbps）
* 可以与低速（1.5Mbps）和全速设备通信的主机这包括连接到USB集线器的多个下游设备。

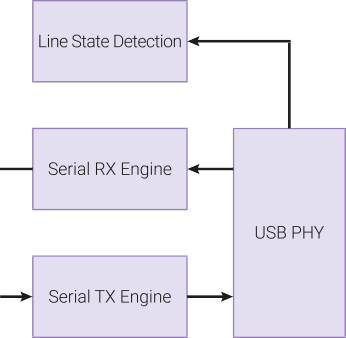
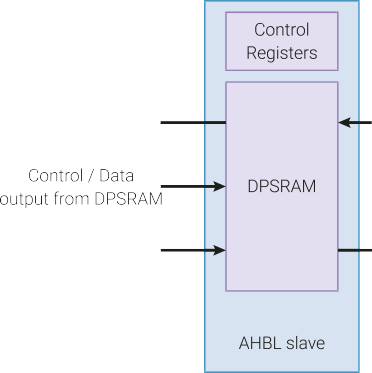
有一个集成的USB 1.1 PHY，它将USB控制器与芯片的DP和DM引脚

* + - 1. 特征

USB控制器硬件处理低级USB协议，这意味着编程器的主要工作是配置控制器，然后响应总线上的事件提供/消耗数据缓冲区。当处理器需要注意时，控制器中断处理器。USB控制器具有4kB的DPSRAM，用于配置和数据缓冲区。

* + - * 1. 设备模式
* USB 2.0兼容全速设备（12Mbps）
* 最多支持32个端点（端点0→ 15，在输入和输出方向）
* 支持Control、Isoclave、Bulk和Cable端点类型
* 支持双缓冲
* DPSRAM中有3840字节的可用缓冲空间 这相当于60 × 64字节缓冲区。
  + - * 1. 主机模式
* 可以与全速（12Mbps）设备和低速设备（1.5Mbps）通信
* 可以通过USB集线器与多个设备通信，包括连接到全速集线器的低速设备
* 在硬件中可以轮询多达15个中断端点。（集线器使用USB端点通知主机连接/断开事件，鼠标通知主机移动等）
  + 1. 架构

*图57. USB控制器架构的简要概述。*



USB控制器是一种面积有效的设计，它将设备控制器或主机控制器多路复用到一组公共组件上。每个组成部分详述如下。

* + - 1. USBPHY

USBPHY提供USBDP和DM引脚与控制器数字逻辑之间的电气接口。 DP和DM引脚是一对差分引脚，这意味着除了对特定线路状态（SE0等）进行编码外，其值始终彼此相反。USBPHY驱动DP和DM引脚传输数据，并对任何传入数据执行差分USB PHY向线路状态检测模块提供单端和差分接收数据。

USB PHY内置上拉和下拉电阻。如果控制器用作全速器件，则DP引脚上拉，以向主机指示已连接全速器件。在主机模式下，弱下拉应用于DP和DM，以便将线路拉至逻辑零，直到器件将DP拉至全速或将DM拉至低速。

* + - 1. 线路状态检测

[[usbspec]](#_bookmark189)定义了需要检测的几种线路状态（总线复位、连接、挂起、恢复、数据1、数据0等）线路状态检测模块具有若干状态机以检测这些状态并将事件用信号通知给其它硬件组件。USB中没有共享时钟信号，因此RX数据必须由内部时钟采样。USB全速的最大数据速率为12Mbps。 RX数据以48 MHz采样，提供4个时钟周期来捕获和过滤总线状态。 线路状态检测模块将经滤波的RX数据分发到串行RX引擎。

* + - 1. 串行RX引擎

串行接收引擎解码由线路状态检测模块捕获的接收数据它产生以下信息:

* 传入数据包的PID
* 传入数据的设备地址
* 传入数据的设备端点
* 数据字节

串行接收引擎还通过对传入数据执行CRC校验来检测RX数据中的错误任何错误都会被发送到其他硬件模块，并可能引发中断。

**注意**

如果您在主机或设备模式下的数据包期间断开USB电缆，您将看到硬件引发的错误。如果您启用错误中断，您的软件将需要考虑这种情况

* + - 1. 串行TX引擎

串行发送引擎是串行接收引擎的镜像它连接到当前活动的控制器（设备或主机）。它创建TOKEN和DATA数据包，包括计算CRC，并在总线上传输它们

* + - 1. DPSRAM

USB控制器具有4kB（4096字节）的DPSRAM（双端口SRAM）。DPSRAM用于存储控制寄存器和数据缓冲器。DPSRAM可作为32位宽的存储器在USB控制器的地址0（0x50100000）访问

DPSRAM具有以下特性，与RP 2040上的大多数寄存器不同

* 支持8/16/32位访问。寄存器通常仅支持32位访问
* DPSRAM不支持设置/清除别名。RP2040寄存器通常支持这些

数据缓冲区通常为64字节长，因为这是大多数FS数据包的最大正常数据包大小 对于Isocom端点，支持最大缓冲区大小为1023字节。 对于其他数据包类型，每个缓冲区的最大大小为64字节。

* + - * 1. 并发访问

USB控制器中的DPSRAM应被视为异步而非原子。它是一个双端口SRAM，这意味着处理器有一个端口来读/写内存，USB控制器也有一个端口来读/写内存。 这意味着处理器和USB控制器可以同时访问相同的内存地址。一个可以写，一个可以读。如果控制器在读取存储器的同时处理器在写入存储器，则可能得到不一致的数据必须小心避免这种情况。

缓冲区控制寄存器中的AVAILABLE位用于指示谁拥有缓冲区的所有权。处理器应将此位设置为1，以赋予控制器缓冲区的所有权当控制器使用完缓冲器后，它会将该位置回0AVAILABLE位应与缓冲区控制寄存器中的其余数据分开设置，以便在AVAILABLE位设置时，缓冲区控制寄存器中的其余数据是准确的

这是必要的，因为处理器时钟XNUMX\_SYS的运行速度可能比XNUMX\_USB时钟快几倍。因此，USB\_sys可以**在**USB控制器以较慢的时钟读取期间更新数据正确的过程是:

* 写入缓冲区信息（长度等）缓冲控制寄存器
* nop，以确保至少经过了一个USB\_SYS周期例如，如果Round\_sys以125 MHz运行，Round\_usb以48 MHz运行，则125/48四舍五入将是3个nop指令
* 设置可用位

如果USB\_SYS和USB\_USB以相同的频率运行，则无需分别设置AVAILABLE位。

**注意**

当控制器将状态写回DPSRAM时，它会对缓冲区0的低2个字节和缓冲区1的高2个字节进行16位写入。因此，如果使用双缓冲模式，在软件中更新缓冲控制寄存器时，最安全的做法是将其视为

* + - * 1. 布局

表394。DPSRAM

布局

0x0 -0xff用于包含配置数据的控制寄存器。剩余的空间，地址0x 100 -0xfff（3840字节）可用于数据缓冲区。控制器具有从地址0x10000开始的控制寄存器。

内存布局因控制器处于设备模式还是主机模式而异。在设备模式下，主机可以访问多个端点，因此必须为每个端点提供端点控制和缓冲区控制寄存器在主机模式下，处理器上运行的主机软件决定访问哪些端点和哪些设备，因此只需要一组端点控制和缓冲区控制寄存器。除了软件驱动的传输，主机控制器还可以轮询多达15个中断端点，并为每个中断端点提供一个寄存器

|  |  |  |
| --- | --- | --- |
| 偏移 | 设备功能 | 主机功能 |
| 0x0 | 设置数据包（8字节） | |
| 0x8 | 对照组中的EP 1 | 网络端点控制1 |
| 0xc | EP1输出控制 | 备用 |
| 0x10 | EP2控制 | 端点控制2 |
| 0x14 | EP2输出控制 | 备用 |
| 0x18 | EP3控制 | 端点控制3 |
| 0x1c | EP3输出控制 | 备用 |
| 0x20 | EP4控制 | 中断端点控制4 |
| 0x24 | EP4输出控制 | 备用 |
| 0x28 | EP5对照 | 端点控制5 |
| 0x2c | EP5输出控制 | 备用 |
| 0x30 | EP6控制 | 端点控制6 |
| 0x34 | EP6输出控制 | 备用 |
| 0x38 | EP7控制 | 端点控制7 |
| 0x3c | EP7输出控制 | 备用 |
| 0x40 | EP8控制 | 网络端点控制8 |
| 0x44 | EP8输出控制 | 备用 |
| 0x48 | EP9对照 | 端点控制9 |
| 0x4c | EP9输出控制 | 备用 |
| 0x50 | 对照组中的EP10 | 端点控制10 |
| 0x54 | EP10输出控制 | 备用 |
| 0x58 | EP11对照 | 端点控制11 |
| 0x5c | EP11输出控制 | 备用 |
| 0x60 | EP12控制 | 中断端点控制12 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 设备功能 | 主机功能 |
| 0x64 | EP12输出控制 | 备用 |
| 0x68 | EP13控制 | 端点控制13 |
| 0x6c | EP13输出控制 | 备用 |
| 0x70 | EP14控制 | 端点控制14 |
| 0x74 | EP14输出控制 | 备用 |
| 0x78 | EP15对照 | 端点控制15 |
| 0x7c | EP15输出控制 | 备用 |
| 0x80 | 缓冲液对照中的EP 0 | EPx缓冲液对照 |
| 0x84 | EP0输出缓冲控制 | 备用 |
| 0x88 | 缓冲液对照中的EP 1 | 端点缓冲区控制1 |
| 0x8c | EP1输出缓冲控制 | 备用 |
| 0x90 | 缓冲液对照中的EP2 | 端点缓冲区控制2 |
| 0x94 | EP2输出缓冲控制 | 备用 |
| 0x98 | 缓冲液对照中的EP3 | 端点缓冲区控制3 |
| 0x9c | EP3输出缓冲控制 | 备用 |
| 0xa0 | 缓冲液对照中的EP4 | 端点缓冲区控制4 |
| 0xa4 | EP4输出缓冲控制 | 备用 |
| 0xa8 | 缓冲液对照中的EP5 | 端点缓冲区控制5 |
| 0xac | EP5输出缓冲控制 | 备用 |
| 0xb0 | 缓冲液对照中的EP6 | 中断端点缓冲区控制6 |
| 0xb4 | EP6输出缓冲控制 | 备用 |
| 0xb8 | 缓冲液对照中的EP7 | 中断端点缓冲区控制7 |
| 0xbc | EP7输出缓冲控制 | 备用 |
| 0xc0 | 缓冲液对照中的EP8 | 端点缓冲区控制8 |
| 0xc4 | EP8输出缓冲控制 | 备用 |
| 0xc8 | 缓冲区控制中的EP 9 | 端点缓冲区控制9 |
| 0xcc | EP9输出缓冲控制 | 备用 |
| 0xd0 | 缓冲液对照中的EP10 | 端点缓冲区控制10 |
| 0xd4 | EP10输出缓冲控制 | 备用 |
| 0xd8 | 缓冲液对照中的EP11 | 端点缓冲区控制11 |
| 0xdc | EP11输出缓冲控制 | 备用 |
| 0xe0 | 缓冲液对照中的EP12 | 端点缓冲区控制12 |
| 0xe4 | EP12输出缓冲控制 | 备用 |
| 0xe8 | 缓冲液对照中的EP13 | 端点缓冲区控制13 |
| 0xec | EP13输出缓冲控制 | 备用 |
| 0xf0 | 缓冲液对照中的EP14 | 端点缓冲区控制14 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 设备功能 | 主机功能 |
| 0xf4 | EP14输出缓冲控制 | 备用 |
| 0xf8 | 缓冲液对照中的EP15 | 端点缓冲区控制15 |
| 0xfc | EP15输出缓冲控制 | 备用 |
| 0x100 | EP0缓冲器0（输入和输出之间 | EPx控制 |
| 0x140 | 可选EP 0缓冲液1 | 备用 |
| 0x180 | 数据缓冲器 | |

* + - * 1. 端点控制寄存器

表395。端点控制寄存器布局

端点控制寄存器用于配置端点。它包括:

* 端点类型
* 数据缓冲区的基地址，如果是双缓冲区，则为数据缓冲区
* 端点上的中断事件应触发

设备必须支持端点0，这样它才能回复SETUP数据包并被枚举。因此，EP 0没有端点控制寄存器它的缓冲区从0x100开始。所有其他端点可以具有单缓冲区或双缓冲区，并映射到编程的基址。由于EP 0没有端点控制寄存器，因此EP 0的中断使能控制来自[SIE\_CTRL](#_bookmark168)。

|  |  |  |
| --- | --- | --- |
| 位数 | 设备功能 | 主机功能 |
| 31 | 端点启用 | |
| 30 | 单缓冲（64字节）= 0，双缓冲（64字节x 2）=1 | |
| 29 | 为每个传输的缓冲区启用缓存 | |
| 28 | 每传输2个缓冲区启用一个缓冲区（仅对双缓冲区有效） | |
| 二十七点二十六分 | 终点类型:质控品= 0，ISO = 1，原液= 2，质控品=3 | |
| 二十五点十八分 | N/A | 主机控制器应轮询此端点的间隔仅适用于中断端点。以ms-1表示例如:值9将每10 ms轮询一次端点。 |
| 17 | 卡隆摊位 |
| 16 | 安东·纳克 |
| 十五点六分 | 数据缓冲区DPSRAM中的地址基偏移 | |

**注意**

数据缓冲区基址必须是64字节对齐的，因为0-5位将被忽略

* + - * 1. 缓冲控制寄存器

缓冲区控制寄存器包含有关该端点的数据缓冲区状态的信息。它在处理器和控制器之间共享。如果端点配置为单缓冲，则仅使用缓冲区的前半部分（位

如果是双缓冲，则缓冲区选择从缓冲区0开始。从那时起，缓冲区选择在缓冲区0和1之间翻转，除非“重置缓冲区选择”位被设置（将缓冲区选择重置为缓冲区0）。缓冲区选择的值在控制器内部，处理器无法访问

表396。缓冲控制寄存器布局

对于EPx上的主机中断和等重数据包，即使传输不成功，也会在完成时设置缓冲区满位可以读取[SIE\_STATUS](#_bookmark169)寄存器中的错误位来确定错误。

|  |  |
| --- | --- |
| 位数 | 功能 |
| 31 | 缓冲区1已满。对于IN事务处理器应将其设置为1，对于OUT事务处理器应将其设置为0。对于OUT事务，控制器将此设置为1，因为它已填满缓冲区。对于IN事务，控制器将其设置为0，因为它已清空缓冲区。仅对双缓冲有效 |
| 30 | 缓冲区1传输的最后一个缓冲区-仅对双缓冲区 |
| 29 | 缓冲区1的数据PID-DATA 0 = 0，DATA 1 = 1-仅对双缓冲有效 |
| 二十七点二十八分 | 等容模式的双缓冲器偏移（0 = 128，1 = 256，2 = 512，3 =1024） |
| 26 | 缓冲区1可用。控制器是否可以使用缓冲区进行传输。配置缓冲区时，处理器将其设置为1控制器在使用缓冲区后设置为0。 即已将数据发送到主机以进行IN事务，或者已将来自主机的数据填充到缓冲区以进行OUT事务。仅对双缓冲有效 |
| 二十五点十六分 | 缓冲区1传输长度-仅对双缓冲区 |
| 15 | 缓冲区0已满。对于IN事务处理器应将其设置为1，对于OUT事务处理器应将其设置为0。对于OUT事务，控制器将此设置为1，因为它已填满缓冲区。对于IN事务，控制器将其设置为0，因为它已清空缓冲区。 |
| 14 | 缓冲区0传输的最后一个缓冲区 |
| 13 | 缓冲区0的数据PID-数据0 = 0，数据1 =1 |
| 12 | 将缓冲区选择重置为缓冲区0-传输结束时清除。仅适用于器械 |
| 11 | 为设备发送STALL，为主机接收STALL |
| 10 | 缓冲区0可用。控制器是否可以使用缓冲区进行传输。配置缓冲区时，处理器将其设置为1控制器在使用缓冲区后设置为0。 即已将数据发送到主机以进行IN事务，或者已将来自主机的数据填充到缓冲区以进行OUT事务。 |
| 九点零分 | 缓冲区0传输长度 |

**警告**

如果以不同的速度运行USB\_SYS和USB\_USB，则应在缓冲区控制寄存器中的其他数据之后设置可用位和暂停位否则，控制器可以用来自先前分组的数据发起事务也就是说，控制器可以看到可用位集，但从前一个数据包中获得数据pid或长度

* + - 1. 设备控制器

本节详细介绍设备控制器从主机接收各种数据包类型时的操作

* + - * 1. 设置

设备控制器**必须**始终接受来自主机的设置数据包。这就是为什么DPSRAM的前8个字节具有用于设置数据包的专用空间。

[[usbspec]](#_bookmark189)指出，接收设置数据包也会清除EP 0上的任何停滞位因此，EP 0的暂停位通过[EP\_STALL\_ARM](#_bookmark175)寄存器中的两位进行门控。当接收到设置数据包时，这些位被清除。这意味着要在EP 0上发送暂停，必须同时设置缓冲区控制寄存器中的暂停位和[EP\_STALL\_ARM](#_bookmark175)中的相应位。

除非有任何错误，否则设置数据包将被放入DPSRAM偏移0x0处的设置数据包缓冲区。然后，设备

最后，设置[SIE\_STATUS](#_bookmark169).SETUP\_REC以指示已经接收到建立分组如果程控仪启用SETUP\_REC中断，则会触发中断（参见[INTE](#_bookmark184)）。

* + - * 1. 在

从设备的角度来看，IN传输意味着将数据传输**到**主机。当从主机接收到IN令牌时，将按以下方式

代币阶段:

* + 如果缓冲控制寄存器中的STALL置位（如果是EP 0，则相应的[EP\_STALL\_ARM](#_bookmark175)位置位），则发送STALL

响应并返回空闲状态。

* + 如果在缓冲区控制中设置了AVAILABLE和FULL位，则移至该阶段
  + 否则，发送NAK，除非这是一个Isocycle端点，在这种情况下，进入空闲状态。数据阶段:
  + 发送数据。如果伊索兰闲置。否则进入ACK阶段。ACK阶段:
  + 等待来自主机的ACK数据包。 如果有超时，则引发超时错误。如果收到ACK，则数据包完成，因此进入状态阶段。

状态阶段:

* + 如果这是传输中的最后一个缓冲区（即， 如果缓冲器控制寄存器中的LAST\_BUFFER位已设置），则设置[SIE\_STATUS](#_bookmark169).TRANS\_COMPUTER。
  + 如果端点是双缓冲的，则将缓冲区选择翻转到另一个缓冲区。
  + 设置[BUFF\_STATUS](#_bookmark171)中的一个位以指示缓冲已完成。处理此事件时，程序员应读取[BUFF\_CPU\_SHOULD\_HANDLE](#_bookmark172)，以查看是缓冲区0还是缓冲区1已完成。如果端点是双缓冲的，

可以同时完成两个缓冲区已清除的[BUFF\_STATUS](#_bookmark171)位将再次置位，此时[BUFF\_CPU\_SHOULD\_HANDLE](#_bookmark172)将发生变化。

* + 在缓冲区控制寄存器的相应一半中更新状态:设置length、pid和last\_buff。其他一切都写为零。

如果NAK被发送到主机，主机将稍后重试

* + - * 1. 出来

当从主机接收到OUT令牌时，将按以下方式处理请求

代币阶段:

* + DATA pid是缓冲区控制寄存器中指定的值如果不是，则引发[SIE\_STATUS](#_bookmark169).DATA\_SEQ\_ERROR。（不检查Isoclave终结点的数据pid，因为Isoclave数据始终使用DATA 0pid发送
  + 是否设置了AVAILABLE位，未设置FULL位。如果是，则进入数据阶段，除非设置了STALL位，在这种情况下，设备控制器将回复STALL。

数据阶段:

* + 将接收到的数据存储在缓冲区中。如果Isoflurane进入状态阶段。否则进入ACK阶段。ACK阶段:
  + 发送ACK。 进入状态阶段。

状态阶段:

参见[第4.1.2.6.2](#_bookmark159)的状态阶段。唯一的区别是，在缓冲区控制寄存器中设置FULL位以指示已接收到数据，而在IN情况下，清除FULL位以指示已发送数据

* + - * 1. 挂起和恢复

USB设备控制器支持挂起和恢复，以及远程恢复（由[SIE\_CTRL](#_bookmark168).RESUME触发），其中设备启动恢复。[SIE\_STATUS](#_bookmark169)中有一个中断/状态位。没有必要启用挂起和恢复中断，因为大多数设备不需要关心挂起和恢复。

当设备没有看到来自主机的任何帧数据包开始（每1ms传输一次）时，设备将进入挂起状态。

**注意**

如果启用挂起中断，则很可能在设备首次连接但总线空闲时看到挂起中断在主机开始发送帧开始数据包之前，总线可以空闲几ms如果没有连接VBUS检测电路，当设备断开时，您也会看到挂起中断这是因为如果没有VBUS检测，就不可能区分断开连接和挂起。

* + - * 1. 勘误

设备控制器有两个硬件问题，这两个问题在RP 2040 B0和RP 2040 B1上都有软件解决方法，并在RP 2040 B2上的硬件中得到修复。更多信息请参见RP 2040-E2和RP 2040-E5。

* + - 1. 主机控制器

主机控制器的设计与设备控制器相似。所有事务都是由主机启动的，因此主机总是处理它启动的事务。因此，只有一组端点控制/端点缓冲控制寄存器。当没有软件控制的事务发生时，还有额外的硬件在后台轮询中断端点

主机需要每隔1ms向设备发送一次保持活动数据包，以防止设备挂起。在全速模式下，这是通过发送SOF（帧开始）数据包来完成的。在低速模式下，发送EOP（数据包结束）。设置控制器时，应设置[SIE\_CTRL](#_bookmark168).KEEP\_ALIVE\_EN和[SIE\_CTRL](#_bookmark168).SOF\_EN以启用这些数据包。

[SIE\_CTRL](#_bookmark168)中的几个位用于开始主机事务:

* SEND\_SETUP-发送设置数据包。这通常与RECEIVE\_TRANS结合使用，因此设置数据包将在设备预期的附加数据事务之后发送。
* SEND\_TRANS- 此传输从主机输出
* RECEIVE\_TRANS- 此传输是在主机
* START\_TRANS- 开始传输-非闭锁
* STOP\_TRANS- 停止电流传输-非锁存
* PREAMBLE\_ENABLE-使用此选项向全速集线器上的低速设备发送数据包。 这将在主机发送的每个数据包之前发送一个PRE令牌数据包（即pre、token、pre、data、pre、ack）。
* SOF\_SYNC-SOF同步位用于将事务延迟到下一个SOF之后。这对于中断和等容端点很有用主机控制器防止64字节的事务与SOF数据包冲突

对于较长的Isocom数据包，软件负责通过使用SOF Sync位和限制一帧中发送的数据包数量来防止冲突如果一个事务是用多个数据包建立的，则SOF Sync位仅适用于第一个数据包。

**警告**

START\_TRANS位单独与SIE\_CTRL寄存器中的其他控制位同步。START\_TRANS位应与SIE\_CTRL寄存器中的其余数据分别置1，以便在提示控制器开始传输时，寄存器内容保持稳定。这是必要的，因为处理器时钟CLK\_SYS可以与CLK\_USB时钟异步。

* 在SIE\_CTRL中写入START\_TRANS以外的字段
* nop，以确保至少经过了两个USB\_SYS周期例如，如果xns\_sys以125 MHz运行，xns\_usb以48 MHz运行，则125/48四舍五入将是6个nop指令
* 设置START\_TRANS位。
  + - * 1. 设置

从主机发送的SETUP数据包始终来自DPSRAM偏移0x0处的专用8字节空间。与设备控制器一样，没有与设置数据包相关联的控制寄存器。当您写入START\_TRANS并设置SEND\_SETUP位时，这些参数将被硬编码并加载到硬件中。一旦发送了设置数据包，主机状态机将等待来自设备的ACK。 如果超时，则会引发RX\_TIMEOUT错误。 如果SEND\_TRANS位被设置，则主机状态机将进入输出阶段。 最常见的是，SEND\_SETUP数据包与RECEIVE\_TRANS位一起使用，因此在发送设置数据包后将进入IN阶段。

* + - * 1. 在

当START\_TRANS位置1时，RECEIVE\_TRANS位置1触发IN传输这之前可能会有一个设置

如果设置了SEND\_SETUP位，则发送数据包

对照阶段:

* 读取位于0x80的*EPx控制*寄存器以获取端点信息:
  + 我们是双缓冲吗？
  + 要启用哪些中断
  + 数据缓冲区的基地址，或数据缓冲区（如果处于双缓冲模式）的基
  + 端点类型
* 在0x100处读取*EPx缓冲区控制*寄存器，以获取端点缓冲区信息，如传输长度和数据pid。主机状态机仍然会检查是否存在AVAILABLE位，因此需要设置此位，并且需要

被解除交易将不会发生，直到这种情况。

代币阶段:

* 将IN令牌数据包发送到设备。 目标设备地址和端点来自[ADDR\_ENDP](#_bookmark163)寄存器。

数据阶段:

* 从所述设备接收所述第一数据分组。 如果设备未回复，则引发RX超时错误。如果数据包具有错误的DATA PID，则引发DATASEQ

ACK阶段:

* 向设备状态阶段发送ACK
* 设置[BUFF\_STATUS](#_bookmark171)位并更新缓冲器控制寄存器。将设置FULL、LAST\_BUFF（如果适用）、DATA\_PID、WR\_LEN。如果这是传输中的最后一个缓冲区，则将设置TRANS\_COMPLEX。

对照阶段（第2点）:

* 主机状态机将继续执行IN事务，直到在buffer\_control寄存器中看到LAST\_BUFF。如果

主机处于双缓冲模式，则主机控制器将在缓冲器控制寄存器的BUF 0和BUF 1部分之间切换。 否则，它将继续读取缓冲区0的缓冲区控制寄存器，并等待FULL被取消设置，并在开始下一个IN事务（即， 在控制阶段等待）。设备可以向主机发送零长度分组以指示它没有更多的数据。在这种情况下，无论是否设置了LAST\_BUFF标志，主机状态机都将停止侦听更多数据 主机软件可以判断出发生了这种情况，因为缓冲区控制寄存器中的BUFF\_DONE将设置为数据长度0。

**警告**

USB主机控制器有一个错误（RP 2040-E4），这意味着写回缓冲区控制寄存器的状态可能出现在寄存器的错误一半位0-15用于缓冲器0，位16-31用于缓冲器1。主机控制器有一个缓冲区选择器，在每次传输完成后翻转。即使在单缓冲模式下，也会在将状态信息写回缓冲控制寄存器时错误地使用此缓冲选择器。 读取缓冲控制寄存器时，不使用缓冲选择器。这意味着主机软件需要跟踪缓冲区选择器，如果缓冲区选择器为1，则将缓冲区控制寄存器向右移位16位

有关更多信息，请参见RP 2040-E4。

* + - * 1. 出来

当START\_TRANS位置1时，SEND\_TRANS位置1触发OUT传输这之前可能会有一个设置

如果设置了SEND\_SETUP位，则发送数据包

对照阶段:

* 读取*EPx控件*以获取端点信息（与[第4.1.2.7.2](#_bookmark160)相同）
* 读取*EPx缓冲区控件*以获取传输长度、数据pid。必须设置AVAILABLE和FULL才能开始传输。代币阶段
* 向设备发送OUT数据包。 目标设备地址和端点来自[ADDR\_ENDP](#_bookmark163)寄存器。数据阶段:
* 将第一个数据包发送到设备。如果端点类型为Isoclave，则没有ACK阶段，因此主机控制器将直接进入状态阶段。如果收到ACK，则进入状态阶段。否则:
  + 如果未收到回复，则引发[SIE\_STATUS](#_bookmark169).RX\_TIMEOUT。
  + 如果接收到NAK，则提升[SIE\_STATUS](#_bookmark169).NAK\_REC并再次发送数据包
  + 如果收到STALL，则提升[SIE\_STATUS](#_bookmark169).STALL\_REC并进入空闲状态。状态阶段:
* 设置[BUFF\_STATUS](#_bookmark171)位并更新缓冲器控制寄存器。 FULL将设置为0。如果这是传输中的最后一个缓冲区，则将设置TRANS\_COMPLEX

**警告**

上述IN部分（RP 2040-E4）中的错误也适用于OUT部分。

对照阶段（第2点）:

如果这不是传输中的最后一个缓冲区，则等待在*EPx缓冲区控制*寄存器中再次设置FULL和AVAILABLE

* + - * 1. 研究终点

主机控制器可以轮询许多设备上的中断端点（最多15个端点）。要启用这些功能，程序员必须:

* 选择主机控制器上的下一个空闲中断端点插槽（从1开始，最多15个）
* 像使用普通IN或OUT一样，对适当的端点控制寄存器和缓冲区控制寄存器进行编程

转移请注意，中断端点仅被单缓冲，因此缓冲控制寄存器的BUF1部分无效。

* 在相应的ADDR\_ENDP寄存器（[ADDR\_ENDP 1](#_bookmark164)至[ADDR\_ENDP 15](#_bookmark164)）中设置器件的地址和端点。

如果设备为低速但连接到全速集线器，则应设置前导码位。端点方向位也应设置。

* 在[INT\_EP\_CTRL](#_bookmark170)中设置中断端点活动位（即设置该寄存器的位1至15

通常，中断端点将是IN传输。例如，将轮询USB集线器以查看其任何端口的状态是否已更改。如果没有改变，集线器将用NAK回复控制器，并且什么也不会发生。类似地，鼠标将以NAK应答，除非自上次轮询中断端点以来鼠标已被移动。

一旦SOF分组已经由主机控制器发送，控制器就轮询SIP端点

控制器从1到15循环，并将尝试轮询INT\_EP\_CTRL中EP\_ACTIVE位设置为1[的任何中断端点](#_bookmark170)。然后，控制器将读取端点控制寄存器和缓冲器控制寄存器，以查看是否存在可用的缓冲器（即， FULL+AVAILABLE（如果是OUT传输）和NOTFULL+ AVAILABLE（如果是IN传输）。如果没有，控制器将移动到下一个中断端点槽。

如果有可用的缓冲区，那么传输的处理方式与正常的IN或OUT传输相同，并且当中断端点有有效的缓冲区时，将设置[BUFF\_STATUS](#_bookmark171)中的BUFF\_DONE标志[BUFF\_CPU\_SHOULD\_HANDLE](#_bookmark172)对于中断端点无效，因为只有一个缓冲区可以完成（RP 2040-E3）。

* + - 1. VBUS控制

USB控制器可以连接到GPIO引脚（参见[第2.19](#_bookmark21)），用于VBUS控制:

* VBUS使能，用于在主机模式下使能VBUSVBUS使能在[SIE\_CTRL](#_bookmark168)中设置
* VBUS检测，用于检测设备模式中是否存在VBUSVBUS检测是[SIE\_STATUS](#_bookmark169)中的一个位，也可以引发VBUS\_DETECT中断（在[INTE](#_bookmark184)中启用）
* VBUS过流，用于检测过流事件。 适用于设备和主机。VBUS过流是[SIE\_STATUS](#_bookmark169)中的一个位。

没有必要将这些引脚中的任何一个连接到GPIO。当DP或DM引脚拉高时，主机可以永久地为VBUS供电，并检测到正在连接的[USB\_PWR](#_bookmark179)中可以强制VBUS检测。

* + 1. 程序员模型
       1. TinyUSB

RP2040 TinyUSB端口应被视为此USB控制器的参考实现该端口可以在以下位置找到:

<https://github.com/hathach/tinyusb/blob/master/src/portable/raspberrypi/rp2040/dcd_rp2040.c><https://github.com/hathach/tinyusb/blob/master/src/portable/raspberrypi/rp2040/hcd_rp2040.c><https://github.com/hathach/tinyusb/blob/master/src/portable/raspberrypi/rp2040/rp2040_usb.h>

图58.dev\_lowlevel USB设备示例的USB分析器跟踪。 控制传输是设备枚举。第一个批量OUT（从主机输出）传输（以蓝色突出显示）是主机向设备发送“Hello World”。第二个批量传输IN（到主机）是设备向主机返回“HelloWorld”

##### 独立设备示例

一个独立的USB设备示例dev\_lowlevel，可以更容易地理解如何与USB控制器交互，而无需理解TinyUSB抽象。除了端点0之外，独立设备还有两个批量端点:EP1 OUT和EP2 IN。该设备被设计为将其在EP 1上接收到的任何数据发送到EP 2。这个例子附带了一个小的Python脚本，它将“Hello World”写入EP1，并检查EP2是否正确接收

本节中包含的代码将引导您设置USB设备控制器以接收设置数据包，然后响应设置数据包。



* + - * 1. 设备控制器初始化

下面的代码初始化USB设备。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev\_lowlevel/dev\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c#L183-L218)第183-218

publicvoid onDestination（）{

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

//重置USB控制器

reset\_block（RESETS\_USB\_CTRL\_BITS）; unreset\_block\_wait（RESETS\_USB\_CTRL\_BITS）;

//清除dpram中的任何先前状态以防万一

memset（usb\_dpram，0，sizeof（\*usb\_dpram））;①

//在处理器上启用USB中断

irq\_set\_enabled（USBCTRL\_IRQ，true）;

//将控制器复用到板载usbphy

usb\_hw->muxing= USB\_USB\_MUXING\_TO\_PHY\_BITS |USB\_USB\_MUXING\_UART\_BITS;

//强制VBUS检测，使设备认为它已插入主机

usb\_hw->USB\_PWR\_VBUS\_DETECT\_BITS = USB\_USB\_PWR\_VBUS\_DETECT\_BITS |USB\_USB\_PWR\_VBUS\_DETECT\_OVERRIDE\_EN\_BITS;

//在设备模式下启用USB控制器。

USB\_hw->main\_cable =USB\_MAIN\_CTRL\_CONTROLLER\_EN\_BITS;

//每个EP 0事务启用一个中断

usb\_hw->sie\_cable = USB\_SIE\_CTRL\_EP0\_INT\_1BUF\_BITS;USB

|  |  |  |
| --- | --- | --- |
| 206 |  | //当缓冲区完成时，当总线复位时， |
| 207 |  | //当接收到一个setup包时， |
| 208 |  | USB\_hw->inte= USB\_INTS\_BUFF\_STATUS\_BITS| |
| 209 |  | USB\_INTS\_BUS\_BUS\_BITS| |
| 210 |  | USB\_INTS\_SETUP\_SETUP\_BITS; |
| 211 |  |  |
| 212 |  | //设置端点（端点控制寄存器） |
| 213 |  | //由设备配置描述 |
| 214 |  | public void run（）; |
| 215 |  |  |
| 216 |  | //通过在DP上启用上拉来呈现全速设备 |
| 217 |  | USB\_hw\_set->sie\_countries =USB\_SIE\_CTRL\_PULLUP\_EN\_BITS; |
| 218 | } |  |

* + - * 1. 为EP 1和EP 2配置端点控制寄存器

函数usb\_configure\_endpoints循环通过设备配置中定义的每个端点（包括EP0 in和EP0 out，它们没有定义端点控制寄存器），并调用usb\_configure\_endpoints函数。这将为该端点设置端点控制寄存器

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev\_lowlevel/dev\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c#L149-L164)第149-164

1. public voidusb\_setup\_endpoint（ const structusb\_endpoint\_configuration\*ep）{
2. printf（“使用缓冲区地址0x%p\n设置端点，ep->descriptor-

>bEndpointAddress，ep->data\_buffer）;

151

152

153

154

155

156

157

158

159

160

161

162

163

164}

//EP 0没有，所以如果是这种情况，则

如果（！ep->endpoint\_control）{return;

}

*//获取数据缓冲区作为USB控制器的DPRAM的*偏移uint32\_tdpram\_offset = usb\_buffer\_offset（ep->data\_buffer）; uint32\_treg = EP\_CTRL\_ENABLE\_BITS

|EP\_CTRL\_INTERRUPT\_PER\_BUFFER

|（ep->descriptor->bmAttributes <<EP\_CTRL\_BUFFER\_TYPE\_LSB）

|dpram\_offset;

\*ep->endpoint\_control =reg;

* + - * 1. 接收设置包

当接收到设置数据包时会引发中断，因此中断处理程序必须处理此事件:

*Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev\_lowlevel/dev\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c#L492-L502)第492-502*

|  |  |
| --- | --- |
| 492 | voidisr\_usbcnums（ void）{ |
| 493 | //USB中断处理程序 |
| 494 | uint32\_tstatus = usb\_hw-> int; |
| 495 | return0; |
| 496 |  |
| 497 | //接收到安装包 |
| 498 | if（status USB\_INTS\_SETUP\_BITS）{ |
| 499 | 处理|= USB\_INTS\_SETUP\_SETUP\_BITS; |
| 500 | USB\_hw\_clear->sie\_status=USB\_SIE\_STATUS\_SETUP\_REC\_BITS; |
| 501 | usb\_handle\_setup\_packet（）; |
| 502 | } |

设置数据包被写入USB RAM的前8个字节，因此设置数据包处理程序将该内存区域转换为struct usb\_setup\_packet \*。

*Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev\_lowlevel/dev\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c#L384-L428)第384-428*

1. voidusb\_handle\_setup\_packet（ void）{
2. volatile structusb\_setup\_packet \*pkt =（ volatile structusb\_setup\_packet \*）&usb\_dpram

->setup\_packet;

|  |  |  |
| --- | --- | --- |
| 386 |  | uint8\_treq\_direction = pkt->bmRequestType; |
| 387 |  | uint8\_treq = pkt->bRequest; |
| 388 |  |  |
| 389 |  | //对于EP 0IN，将PID重置为1 |
| 390 |  | usb\_get\_endpoint\_configuration（EP0\_IN\_ADDR）->next\_pid =1u; |
| 391 |  |  |
| 392 |  | if（req\_direction == USB\_OUT）{ |
| 393 |  | if（request == USB\_REQUEST\_SET\_ADDRESS）{ |
| 394 |  | usb\_set\_device\_address（pkt）; |
| 395 |  | }else if（req == USB\_REQUEST\_SET\_CONFIGURATION）{ |
| 396 |  | usb\_set\_device\_configuration（pkt）; |
| 397 |  | }否则{ |
| 398 |  | usb\_acknowledge\_out\_request（）; |
| 399 |  | printf（“Other OUT request（0x%x）\r\n”，pkt->bRequest）; |
| 400 |  | } |
| 401 |  | }else if（req\_direction == USB\_IN）{ |
| 402 |  | if（request == USB\_REQUEST\_GET\_PROTECTOR）{ |
| 403 |  | uint16\_tdescriptor\_type = pkt->wValue>>8; |
| 404 |  |  |
| 405 |  | public void run（）{ |
| 406 |  | USB\_DT\_DEVICE案例: |
| 407 |  | usb\_handle\_device\_descriptor（pkt）; |
| 408 |  | printf（“GET DEVICE\_DESIGNER\r\n”）; |
| 409 |  | 破碎; |
| 410 |  |  |
| 411 |  | 案例USB\_DT\_CONFIG: |
| 412 |  | usb\_handle\_config\_descriptor（pkt）; |
| 413 |  | printf（“获取配置文件\r\n”）; |
| 414 |  | 破碎; |
| 415 |  |  |
| 416 |  | USB\_DT\_STRING用例: |
| 417 |  | usb\_handle\_string\_descriptor（pkt）; |
| 418 |  | printf（“GET STRINGPROTECTOR\r\n”）; |
| 419 |  | 破碎; |
| 420 |  |  |
| 421 |  | 默认值: |
| 422 |  | printf（“未处理的GET\_PROTECTOR类型0x%x\r\n”，描述符类型）; |
| 423 |  | } |
| 424 |  | }否则{ |
| 425 |  | printf（“Other IN request（0x%x）\r\n”，pkt->bRequest）; |
| 426 |  | } |
| 427 |  | } |
| 428 | } |  |

* + - * 1. 回复EP0IN上的设置数据包

主机请求的第一件事是设备描述符，下面的代码处理该设置请求。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev\_lowlevel/dev\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c#L267-L274)第267-274

1. voidusb\_handle\_device\_descriptor（ volatile structusb\_setup\_packet\*pkt）{
2. const结构usb\_device\_descriptor \*d =dev\_config.device\_descriptor;
3. //EP0in
4. 结构usb\_endpoint\_configuration \*ep =usb\_get\_endpoint\_configuration（EP0\_IN\_ADDR）;
5. //总是响应pid1
6. int count =1;
7. usb\_start\_transfer（ep，（uint8\_t\*）d，MIN（sizeof（structusb\_device\_descriptor），pkt-

>wLength（）;

274}

usb\_start\_transfer函数将发送的数据复制到相应的硬件缓冲区，并配置缓冲区控制寄存器。 一旦缓冲控制寄存器被写入，器件控制器将用数据响应主机。 在此之前，设备将使用NAK进行回复。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev\_lowlevel/dev\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c#L239-L261)第239-261

voidusb\_start\_transfer（structusb\_endpoint\_configuration \*ep，uint8\_t\*buf，uint16\_tlen）{

240

241

242

243

244

//为了简单起见，我们断言长度<= 64字节。

//对于多包传输，请参阅tinyusb端口。

return（len =64）;

printf（“Start transfer of len %d on ep addr 0x%x\n”，len，ep->descriptor-

>bEndpointAddress）;

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261}

//准备缓冲区控制寄存器值

uint32\_tval = len |USB\_BUF\_CTRL\_AVAIL;

if（int n，int n）{

//需要将数据从用户缓冲区复制到usb内存

memcpy（（void\*）ep->data\_buffer，（void\*）buf，len）;

//标记为已满

Val|= USB\_BUF\_CTRL\_FULL;

}

//设置pid和flip用于下一次传输

Val|= ep->next\_pid？USB\_BUF\_CTRL\_DATA1\_PID:USB\_BUF\_CTRL\_DATA0\_PID; ep->next\_pid ^=1u;

\*ep->buffer\_control =val;

* + 1. 登记册一览表

USB寄存器从基址0x50110000开始（在SDK中定义为USBCTRL\_REGS\_BASE）。

*表397。 USB寄存器列表*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [地址\_ENDP](#_bookmark163) | 设备地址和端点控制 |
| 0x04 | [ADDR\_ENDP1](#_bookmark164) | 浏览端点1。仅对主机模式有效 |
| 0x08 | [ADDR\_ENDP2](#_bookmark164) | 浏览端点2。仅对主机模式有效 |
| 0x0C | [ADDR\_ENDP3](#_bookmark164) | 3. 仅对主机模式有效 |
| 0x10 | [ADDR\_ENDP4](#_bookmark164) | 4. 仅对主机模式有效 |
| 0x14 | [ADDR\_ENDP5](#_bookmark164) | 5. 仅对主机模式有效 |
| 0x18 | [ADDR\_ENDP6](#_bookmark164) | 第六章.仅对主机模式有效 |
| 0x1c | [ADDR\_ENDP7](#_bookmark164) | 第七章.仅对主机模式有效 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x20 | [ADDR\_ENDP8](#_bookmark164) | 8. 仅对主机模式有效 |
| 0x24 | [ADDR\_ENDP9](#_bookmark164) | 9. 仅对主机模式有效 |
| 0x28 | [ADDR\_ENDP10](#_bookmark164) | 第10章仅对主机模式有效 |
| 0x2c | [ADDR\_ENDP11](#_bookmark164) | 第11章.仅对主机模式有效 |
| 0x30 | [ADDR\_ENDP12](#_bookmark164) | 第12章.仅对主机模式有效 |
| 0x34 | [ADDR\_ENDP13](#_bookmark164) | 第13章.仅对主机模式有效 |
| 0x38 | [ADDR\_ENDP14](#_bookmark164) | 14. history仅对主机模式有效 |
| 0x3c | [ADDR\_ENDP15](#_bookmark164) | 第15章.仅对主机模式有效 |
| 0x40 | [主控制](#_bookmark165) | 主控制寄存器 |
| 0x44 | [SOF\_WR](#_bookmark166) | 在主机控制器中设置SOF（帧起始）帧编号。SOF数据包每1ms发送一次，主机每次将帧号加1。 |
| 0x48 | [SOF\_RD](#_bookmark167) | 读取看到的最后一个SOF（帧开始）帧编号在设备模式下，从主机接收的最后一个SOF在主机模式下，主机发送的最后一个SOF。 |
| 0x4c | [SIE\_CTRL](#_bookmark168) | SIE控制寄存器 |
| 0x50 | [SIE\_STATUS](#_bookmark169) | SIE状态寄存器 |
| 0x54 | [INT\_EP\_CTRL](#_bookmark170) | 中断端点控制暂存器 |
| 0x58 | [BUFF\_STATUS](#_bookmark171) | 缓冲区状态寄存器。此处设置的位表示缓冲区已在端点上完成（如果启用了缓冲区中断）。有可能完成2个缓冲区，因此清除缓冲区状态位可能会在下一个时钟周期立即将其重置 |
| 0x5c | [BUFF\_CPU\_SHOULD\_HANDLE](#_bookmark172) | 应该处理双缓冲区中的哪一个 仅当每个缓冲区使用中断时有效（即 不是每2个缓冲器）。 对于主机中断端点轮询无效，因为它们仅为单缓冲。 |
| 0x60 | [EP\_中止](#_bookmark173) | 仅限设备:可以设置为忽略此端点的缓冲区控制寄存器，以防您想要撤销缓冲区。 每次访问端点时都将发送NAK，直到该位被清除。当修改缓冲区控制寄存器安全时，EP\_ABORT\_DONE中的相应位置1 |
| 0x64 | [EP\_ABORT\_DONE](#_bookmark174) | 仅器械:与EP\_ABORT配合使用。一旦端点空闲就设置，以便程序员知道修改缓冲区控制寄存器是安全的 |
| 0x68 | [EP\_STALL\_ARM](#_bookmark175) | 设备:此位必须与缓冲区控制寄存器中的STALL位一起设置当接收到SETUP数据包时，设备控制器清除这些位，因为USB规范要求当接收到SETUP数据包时清除STALL条件 |
| 0x6c | [NAK\_POLL](#_bookmark176) | 由主机控制器使用设置设备以NAK应答时重试前的等待时间（以 |
| 0x70 | [EP\_STATUS\_STALL\_NAK](#_bookmark177) | 设备:位在IRQ\_ON\_NAK或IRQ\_ON\_STALL位置1时置1对于EP 0，这来自SIE\_CTRL。对于所有其他端点，它来自端点控制寄存器。 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x74 | [USB\_MUXING](#_bookmark178) | 在哪里连接USB控制器。默认情况下应该是to\_phy |
| 0x78 | [USB\_PWR](#_bookmark179) | 在VBUS信号未连接到GPIO的情况下，覆盖电源信号 设置覆盖值，然后设置覆盖启用以切换到覆盖值。 |
| 0x7c | [USBPHY\_DIRECT](#_bookmark180) | 此寄存器允许直接控制USB物理层。与usbphy\_direct\_override寄存器配合使用，使能每个覆盖位。 |
| 0x80 | [USBPHY\_DIRECT\_OVERRIDE](#_bookmark181) | 在usbphy\_direct中为每个控件启用 |
| 0x84 | [USBPHY\_TRIM](#_bookmark182) | 用于调整USB phy下拉电阻的微调值 |
| 0x8c | [INTR](#_bookmark183) | 原始中断 |
| 0x90 | [INTE](#_bookmark184) | 中断使能 |
| 0x94 | [INTF](#_bookmark185) | 中断强制 |
| 0x98 | [INTs](#_bookmark186) | 强制屏蔽后的屏蔽状态 |

表398。ADDR\_ENDP寄存器

表399.ADDR\_ENDP1、ADDR\_ENDP2、.、ADDR\_ENDP14、ADDR\_ENDP15

寄存器

#### [USB](#_bookmark162):ADDR\_ENDP寄存器

**偏移**:0x00

描述

设备地址和端点控制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一点二十分 | Reserved. | - | - | - |
| 十九点十六分 | 端点 | 要向其发送数据的设备终结点。仅对主机模式有效 | RW | 0x0 |
| 十五点七分 | Reserved. | - | - | - |
| 六比零 | 地址 | 在设备方式下，设备应响应的地址.响应来自主机的SET\_ADDR设置数据包时置1在主机模式下，设置为要通信的设备的地址。 | RW | 0x00 |

#### [USB](#_bookmark162): ADDR\_ENDP1、ADDR\_ENDP2、 .、 ADDR\_ENDP14、ADDR\_ENDP15

寄存器

**偏移**:0x04、0x08、...、0x38、0x3c

描述

中断端点*N*。 仅对主机模式有效。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:27 | Reserved. | - | - | - |
| 26 | INTEP\_序言E | CNOEP需要前导码（是全速集线器上的低速设备 | RW | 0x0 |
| 25 | INTEP\_ | 中断端点的方向。输入=0，输出=1 | RW | 0x0 |
| 二十四点二十分 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 十九点十六分 | 端点 | 中断端点的端点号 | RW | 0x0 |
| 十五点七分 | Reserved. | - | - | - |
| 六比零 | 地址 | 设备地址 | RW | 0x00 |

400号桌MAIN\_CTRL寄存器

401号桌SOF\_WR寄存器

402号桌SOF\_RD寄存器

403号桌SIE\_CTRL

寄存器

#### [USB](#_bookmark162):MAIN\_CTRL寄存器

**偏移**:0x40

描述

主控制寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | SIM\_计时 | 减少模拟时间 | RW | 0x0 |
| 三十比二 | Reserved. | - | - | - |
| 1 | 主机\_N设备 | 设备模式= 0，主机模式=1 | RW | 0x0 |
| 0 | 控制器\_EN | 使能控制器 | RW | 0x0 |

#### [USB](#_bookmark162):SOF\_WR寄存器

**偏移**:0x44

描述

在主机控制器中设置SOF（帧起始）帧编号。SOF数据包每1ms发送一次，主机每次将帧号加1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一点十一分 | Reserved. | - | - | - |
| 十比零 | 计数 |  | WF | 0x000 |

#### [USB](#_bookmark162):SOF\_RD寄存器

**偏移**:0x48

描述

读取看到的最后一个SOF（帧开始）帧编号在设备模式下，从主机接收的最后一个SOF在主机模式下，主机发送的最后一个SOF。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一点十一分 | Reserved. | - | - | - |
| 十比零 | 计数 |  | RO | 0x000 |

#### [USB](#_bookmark162):SIE\_CTRL寄存器

**偏移**:0x4c

描述

SIE控制寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | EP0\_INT\_STALL | 器械:当EP 0发送STALL时，在EP\_STATUS\_STALL\_NAK中设置位 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 30 | EP0\_DOUBLE\_BUF | 器件:EP0单缓冲= 0，双缓冲=1 | RW | 0x0 |
| 29 | EP0\_INT\_1BUF | 器械:在EP0上完成的每个缓冲区的BUFF\_STATUS中设置位 | RW | 0x0 |
| 28 | EP0\_INT\_2BUF | 器械:在EP0上每完成2个缓冲区，在BUFF\_STATUS中设置位 | RW | 0x0 |
| 27 | EP0\_INT\_NAK | 器械:当EP 0发送NAK时，在EP\_STATUS\_STALL\_NAK中设置位 | RW | 0x0 |
| 26 | 直接\_EN | 直接总线驱动使能 | RW | 0x0 |
| 25 | DIRECT\_DP | DP的直接控制 | RW | 0x0 |
| 24 | DIRECT\_DM | 直接控制DM | RW | 0x0 |
| 二十三点十九分 | Reserved. | - | - | - |
| 18 | 收发器\_PD | 掉电总线收发器 | RW | 0x0 |
| 17 | RPU\_OPT | 器械:上拉强度（0=1K2，1=2k3） | RW | 0x0 |
| 16 | 联系我们 | 器件:使能上拉电阻 | RW | 0x0 |
| 15 | 拉低\_EN | 主机:使能下拉电阻 | RW | 0x0 |
| 14 | Reserved. | - | - | - |
| 13 | 公交车 | 主机:复位总线 | SC | 0x0 |
| 12 | 恢复 | 设备:远程唤醒。设备可以在挂起后启动自己的 | SC | 0x0 |
| 11 | VBUS\_EN | 主机:启用VBUS | RW | 0x0 |
| 10 | 保持活力 | 主机:启用保持活动数据包（用于低速总线） | RW | 0x0 |
| 9 | SOF\_EN | 主机:启用SOF生成（用于全速总线） | RW | 0x0 |
| 8 | SOF\_SYNC | 主机:延迟数据包，直到SOF之后 | RW | 0x0 |
| 7 | Reserved. | - | - | - |
| 6 | 序言\_EN | 主机:在FS集线器上为LS设备启用预置 | RW | 0x0 |
| 5 | Reserved. | - | - | - |
| 4 | STOP\_TRANS | 主机:停止事务 | SC | 0x0 |
| 3 | 接收数据 | 主机:接收事务（IN到主机） | RW | 0x0 |
| 2 | 发送数据 | 主机:发送事务（从主机OUT） | RW | 0x0 |
| 1 | 发送设置 | 主机:发送设置数据包 | RW | 0x0 |
| 0 | 开始\_TRANS | 主机:开始交易 | SC | 0x0 |

#### [USB](#_bookmark162):SIE\_STATUS寄存器

**偏移**:0x50

描述

SIE状态寄存器

404号桌SIE\_STATUS寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | 数据序列错误 | 数据序列错误。  在以下情况下，设备可能会引发序列错误   * 接收到SETUP数据包，然后是DATA 1数据包（数据相位应始终为DATA 0）\* 从主机接收到OUT数据包，但与从DPSRAM读取的缓冲区控制寄存器中的数据pid不   在以下情况下，主机可能会引发数据序列错误   * 来自设备的IN数据包具有错误的数据PID | WC | 0x0 |
| 30 | ACK\_REC | 收到ACK 由主机和设备引发。 | WC | 0x0 |
| 29 | STALL\_REC | 主持人:收到失速 | WC | 0x0 |
| 28 | NAK\_REC | 主机:收到NAK | WC | 0x0 |
| 27 | RX\_超时 | 如果在USB规范指定的最长时间内未收到ACK，主机和设备都会引发RX超时。 | WC | 0x0 |
| 26 | RX\_OVERFLOW | 如果传入数据过快，则串行RX引擎会引发RX溢出 | WC | 0x0 |
| 25 | BIT\_STUFF\_ERROR | 位填充错误。由Serial RX引擎引发 | WC | 0x0 |
| 24 | CRC错误 | CRC错误。由Serial RX引擎引发 | WC | 0x0 |
| 23:20 | Reserved. | - | - | - |
| 19 | 公交车 | 设备:收到总线复位 | WC | 0x0 |
| 18 | 变速器\_完成E | 交易完成由器械引发，如果:   * 发送IN或OUT数据包时，缓冲区控制寄存器中的LAST\_BUFF位已置位   由主机引发，如果:   * 当没有数据输入或数据输出事务跟随时，发送设置数据包\* 接收到IN数据包，并且在缓冲区控制寄存器中设置LAST\_BUFF位\*接收到长度为零的IN数据包 \* 发送OUT数据包，并且设置LAST\_BUFF | WC | 0x0 |
| 17 | 设置\_记录 | 设备:收到设置数据包 | WC | 0x0 |
| 16 | 连接 | 设备:已连接 | WC | 0x0 |
| 十五点十二分 | Reserved. | - | - | - |
| 11 | 恢复 | 主机:设备已启动远程恢复。设备:主机已启动恢复。 | WC | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 10 | VBUS\_OVER\_CUR | 检测到VBUS过电流 | RO | 0x0 |
| 9:8 | 速度 | 主机:设备速度。断开连接= 00，LS = 01，FS =10 | WC | 0x0 |
| 七点五 | Reserved. | - | - | - |
| 4 | 悬浮 | 总线处于暂停状态。对设备和主机有效。如果未启用保持活动/SOF帧，主机和设备将进入挂起状态 | WC | 0x0 |
| 三比二 | 线路状态 | USB总线状态 | RO | 0x0 |
| 1 | Reserved. | - | - | - |
| 0 | VBUS\_检测到 | 设备:检测到VBUS | RO | 0x0 |

405号桌INT\_EP\_CTRL寄存器

表406。BUFF\_STATUS

寄存器

#### [USB](#_bookmark162):INT\_EP\_CTRL寄存器

**偏移**:0x54

描述

中断端点控制暂存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - | - |
| 十五比一 | INT\_EP\_ACTIVE | 主机:启用中断端点1→15 | RW | 0x0000 |
| 0 | Reserved. | - | - | - |

#### [USB](#_bookmark162):BUFF\_STATUS寄存器

**偏移**:0x58

描述

缓冲区状态寄存器。此处设置的位表示缓冲区已在端点上完成（如果启用了缓冲区中断有可能完成2个缓冲区，因此清除缓冲区状态位可能会在下一个时钟周期立即将其重置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | EP15\_OUT |  | WC | 0x0 |
| 30 | EP15\_IN |  | WC | 0x0 |
| 29 | EP14\_OUT |  | WC | 0x0 |
| 28 | EP14\_IN |  | WC | 0x0 |
| 27 | EP13\_OUT |  | WC | 0x0 |
| 26 | EP13\_IN |  | WC | 0x0 |
| 25 | EP12\_OUT |  | WC | 0x0 |
| 24 | EP12\_IN |  | WC | 0x0 |
| 23 | EP11\_OUT |  | WC | 0x0 |
| 22 | EP11\_IN |  | WC | 0x0 |
| 21 | EP10\_OUT |  | WC | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 20 | EP10\_IN |  | WC | 0x0 |
| 19 | EP9\_OUT |  | WC | 0x0 |
| 18 | EP9\_IN |  | WC | 0x0 |
| 17 | EP8\_OUT |  | WC | 0x0 |
| 16 | EP8\_IN |  | WC | 0x0 |
| 15 | EP7\_OUT |  | WC | 0x0 |
| 14 | EP7\_IN |  | WC | 0x0 |
| 13 | EP6\_OUT |  | WC | 0x0 |
| 12 | EP6\_IN |  | WC | 0x0 |
| 11 | EP5\_OUT |  | WC | 0x0 |
| 10 | EP5\_IN |  | WC | 0x0 |
| 9 | EP4\_OUT |  | WC | 0x0 |
| 8 | EP4\_IN |  | WC | 0x0 |
| 7 | EP3\_OUT |  | WC | 0x0 |
| 6 | EP3\_IN |  | WC | 0x0 |
| 5 | EP2\_OUT |  | WC | 0x0 |
| 4 | EP2\_IN |  | WC | 0x0 |
| 3 | EP1\_OUT |  | WC | 0x0 |
| 2 | EP1\_IN |  | WC | 0x0 |
| 1 | EP0\_OUT |  | WC | 0x0 |
| 0 | EP0\_IN |  | WC | 0x0 |

407号桌BUFF\_CPU\_SHOULD\_H

ANDLE寄存器

#### [USB](#_bookmark162):BUFF\_CPU\_SHOULD\_HANDLE寄存器

**偏移**:0x5c

描述

应该处理双缓冲区中的哪一个。 仅当每个缓冲区使用中断时有效（即 不是每2个缓冲器）。 对于主机中断端点轮询无效，因为它们仅为单缓冲。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | EP15\_OUT |  | RO | 0x0 |
| 30 | EP15\_IN |  | RO | 0x0 |
| 29 | EP14\_OUT |  | RO | 0x0 |
| 28 | EP14\_IN |  | RO | 0x0 |
| 27 | EP13\_OUT |  | RO | 0x0 |
| 26 | EP13\_IN |  | RO | 0x0 |
| 25 | EP12\_OUT |  | RO | 0x0 |
| 24 | EP12\_IN |  | RO | 0x0 |
| 23 | EP11\_OUT |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 22 | EP11\_IN |  | RO | 0x0 |
| 21 | EP10\_OUT |  | RO | 0x0 |
| 20 | EP10\_IN |  | RO | 0x0 |
| 19 | EP9\_OUT |  | RO | 0x0 |
| 18 | EP9\_IN |  | RO | 0x0 |
| 17 | EP8\_OUT |  | RO | 0x0 |
| 16 | EP8\_IN |  | RO | 0x0 |
| 15 | EP7\_OUT |  | RO | 0x0 |
| 14 | EP7\_IN |  | RO | 0x0 |
| 13 | EP6\_OUT |  | RO | 0x0 |
| 12 | EP6\_IN |  | RO | 0x0 |
| 11 | EP5\_OUT |  | RO | 0x0 |
| 10 | EP5\_IN |  | RO | 0x0 |
| 9 | EP4\_OUT |  | RO | 0x0 |
| 8 | EP4\_IN |  | RO | 0x0 |
| 7 | EP3\_OUT |  | RO | 0x0 |
| 6 | EP3\_IN |  | RO | 0x0 |
| 5 | EP2\_OUT |  | RO | 0x0 |
| 4 | EP2\_IN |  | RO | 0x0 |
| 3 | EP1\_OUT |  | RO | 0x0 |
| 2 | EP1\_IN |  | RO | 0x0 |
| 1 | EP0\_OUT |  | RO | 0x0 |
| 0 | EP0\_IN |  | RO | 0x0 |

408号桌EP\_中止

寄存器

#### [USB](#_bookmark162):EP\_ABORT寄存器

**偏移**:0x60

描述

仅限设备:可以设置为忽略此端点的缓冲区控制寄存器，以防您想要撤销缓冲区。每次访问端点时都将发送NAK，直到该位被清除。当修改缓冲区控制寄存器安全时，EP\_ABORT\_DONE中的相应位置1。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | EP15\_OUT |  | RW | 0x0 |
| 30 | EP15\_IN |  | RW | 0x0 |
| 29 | EP14\_OUT |  | RW | 0x0 |
| 28 | EP14\_IN |  | RW | 0x0 |
| 27 | EP13\_OUT |  | RW | 0x0 |
| 26 | EP13\_IN |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 25 | EP12\_OUT |  | RW | 0x0 |
| 24 | EP12\_IN |  | RW | 0x0 |
| 23 | EP11\_OUT |  | RW | 0x0 |
| 22 | EP11\_IN |  | RW | 0x0 |
| 21 | EP10\_OUT |  | RW | 0x0 |
| 20 | EP10\_IN |  | RW | 0x0 |
| 19 | EP9\_OUT |  | RW | 0x0 |
| 18 | EP9\_IN |  | RW | 0x0 |
| 17 | EP8\_OUT |  | RW | 0x0 |
| 16 | EP8\_IN |  | RW | 0x0 |
| 15 | EP7\_OUT |  | RW | 0x0 |
| 14 | EP7\_IN |  | RW | 0x0 |
| 13 | EP6\_OUT |  | RW | 0x0 |
| 12 | EP6\_IN |  | RW | 0x0 |
| 11 | EP5\_OUT |  | RW | 0x0 |
| 10 | EP5\_IN |  | RW | 0x0 |
| 9 | EP4\_OUT |  | RW | 0x0 |
| 8 | EP4\_IN |  | RW | 0x0 |
| 7 | EP3\_OUT |  | RW | 0x0 |
| 6 | EP3\_IN |  | RW | 0x0 |
| 5 | EP2\_OUT |  | RW | 0x0 |
| 4 | EP2\_IN |  | RW | 0x0 |
| 3 | EP1\_OUT |  | RW | 0x0 |
| 2 | EP1\_IN |  | RW | 0x0 |
| 1 | EP0\_OUT |  | RW | 0x0 |
| 0 | EP0\_IN |  | RW | 0x0 |

409号表EP\_ABORT\_DONE

寄存器

#### [USB](#_bookmark162):EP\_ABORT\_DONE寄存器

**偏移**:0x64

描述

仅限设备:与EP\_ABORT配合使用。一旦一个端点空闲时置1，程序员就知道修改缓冲区控制寄存器是安全

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | EP15\_OUT |  | WC | 0x0 |
| 30 | EP15\_IN |  | WC | 0x0 |
| 29 | EP14\_OUT |  | WC | 0x0 |
| 28 | EP14\_IN |  | WC | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 27 | EP13\_OUT |  | WC | 0x0 |
| 26 | EP13\_IN |  | WC | 0x0 |
| 25 | EP12\_OUT |  | WC | 0x0 |
| 24 | EP12\_IN |  | WC | 0x0 |
| 23 | EP11\_OUT |  | WC | 0x0 |
| 22 | EP11\_IN |  | WC | 0x0 |
| 21 | EP10\_OUT |  | WC | 0x0 |
| 20 | EP10\_IN |  | WC | 0x0 |
| 19 | EP9\_OUT |  | WC | 0x0 |
| 18 | EP9\_IN |  | WC | 0x0 |
| 17 | EP8\_OUT |  | WC | 0x0 |
| 16 | EP8\_IN |  | WC | 0x0 |
| 15 | EP7\_OUT |  | WC | 0x0 |
| 14 | EP7\_IN |  | WC | 0x0 |
| 13 | EP6\_OUT |  | WC | 0x0 |
| 12 | EP6\_IN |  | WC | 0x0 |
| 11 | EP5\_OUT |  | WC | 0x0 |
| 10 | EP5\_IN |  | WC | 0x0 |
| 9 | EP4\_OUT |  | WC | 0x0 |
| 8 | EP4\_IN |  | WC | 0x0 |
| 7 | EP3\_OUT |  | WC | 0x0 |
| 6 | EP3\_IN |  | WC | 0x0 |
| 5 | EP2\_OUT |  | WC | 0x0 |
| 4 | EP2\_IN |  | WC | 0x0 |
| 3 | EP1\_OUT |  | WC | 0x0 |
| 2 | EP1\_IN |  | WC | 0x0 |
| 1 | EP0\_OUT |  | WC | 0x0 |
| 0 | EP0\_IN |  | WC | 0x0 |

#### [USB](#_bookmark162):EP\_STALL\_ARM寄存器

**偏移**:0x68

描述

设备:此位必须与缓冲区控制寄存器中的STALL位一起设置，以在EP 0上发送STALL当接收到SETUP数据包时，设备控制器清除这些位，因为USB规范要求当接收到SETUP数据包时清除STALL条件。

*表410。EP\_STALL\_ARM*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三11:2 | Reserved. | - | - | - |
| 1 | EP0\_OUT |  | RW | 0x0 |
| 0 | EP0\_IN |  | RW | 0x0 |

寄存器

表411。NAK\_POLL

寄存器

表412。EP\_STATUS\_STALL\_N

AK寄存器

#### [USB](#_bookmark162):NAK\_POLL寄存器

**偏移**:0x6c

描述

由主机控制器使用设置设备以NAK应答时重试前的等待时间（以微秒为单位）。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一点二十六分 | Reserved. | - | - | - |
| 二十五点十六分 | DELAY\_FS | 全速设备的NAK轮询间隔 | RW | 0x010 |
| 十五点十分 | Reserved. | - | - | - |
| 9:0 | 延迟\_LS | 低速设备的NAK轮询间隔 | RW | 0x010 |

#### [USB](#_bookmark162):EP\_STATUS\_STALL\_NAK寄存器

**偏移**:0x70

描述

设备:位在IRQ\_ON\_NAK或IRQ\_ON\_STALL位置1时置1。 对于EP 0，这来自SIE\_CTRL。 对于所有其他端点，它来自端点控制寄存器。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | EP15\_OUT |  | WC | 0x0 |
| 30 | EP15\_IN |  | WC | 0x0 |
| 29 | EP14\_OUT |  | WC | 0x0 |
| 28 | EP14\_IN |  | WC | 0x0 |
| 27 | EP13\_OUT |  | WC | 0x0 |
| 26 | EP13\_IN |  | WC | 0x0 |
| 25 | EP12\_OUT |  | WC | 0x0 |
| 24 | EP12\_IN |  | WC | 0x0 |
| 23 | EP11\_OUT |  | WC | 0x0 |
| 22 | EP11\_IN |  | WC | 0x0 |
| 21 | EP10\_OUT |  | WC | 0x0 |
| 20 | EP10\_IN |  | WC | 0x0 |
| 19 | EP9\_OUT |  | WC | 0x0 |
| 18 | EP9\_IN |  | WC | 0x0 |
| 17 | EP8\_OUT |  | WC | 0x0 |
| 16 | EP8\_IN |  | WC | 0x0 |
| 15 | EP7\_OUT |  | WC | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 14 | EP7\_IN |  | WC | 0x0 |
| 13 | EP6\_OUT |  | WC | 0x0 |
| 12 | EP6\_IN |  | WC | 0x0 |
| 11 | EP5\_OUT |  | WC | 0x0 |
| 10 | EP5\_IN |  | WC | 0x0 |
| 9 | EP4\_OUT |  | WC | 0x0 |
| 8 | EP4\_IN |  | WC | 0x0 |
| 7 | EP3\_OUT |  | WC | 0x0 |
| 6 | EP3\_IN |  | WC | 0x0 |
| 5 | EP2\_OUT |  | WC | 0x0 |
| 4 | EP2\_IN |  | WC | 0x0 |
| 3 | EP1\_OUT |  | WC | 0x0 |
| 2 | EP1\_IN |  | WC | 0x0 |
| 1 | EP0\_OUT |  | WC | 0x0 |
| 0 | EP0\_IN |  | WC | 0x0 |

413号表USB\_MUX寄存器

414号表。USB\_PWR寄存器

#### [USB](#_bookmark162):USB\_MUXING寄存器

**偏移**:0x74

描述

在哪里连接USB控制器。默认情况下应该是to\_phy

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一比四 | Reserved. | - | - | - |
| 3 | 电子邮件 |  | RW | 0x0 |
| 2 | TO\_DIGITAL\_PAD |  | RW | 0x0 |
| 1 | 到扩展 |  | RW | 0x0 |
| 0 | TO\_PHY |  | RW | 0x0 |

#### [USB](#_bookmark162):USB\_PWR寄存器

**偏移**:0x78

描述

在VBUS信号未连接到GPIO的情况下，覆盖电源信号设置覆盖值，然后设置覆盖启用以切换到覆盖值。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:6 | Reserved. | - | - | - |
| 5 | OVERCURR\_DETECT\_EN |  | RW | 0x0 |
| 4 | 过流检测 |  | RW | 0x0 |
| 3 | VBUS\_DETECT\_OVERRIDE\_EN |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 2 | VBUS\_检测 |  | RW | 0x0 |
| 1 | VBUS\_EN\_OVERRIDE\_EN |  | RW | 0x0 |
| 0 | VBUS\_EN |  | RW | 0x0 |

415号桌USBPHY\_DIRECT

寄存器

#### [USB](#_bookmark162):USBPHY\_DIRECT寄存器

**偏移**:0x7c

描述

此寄存器允许直接控制USB物理层。与usbphy\_direct\_override寄存器配合使用，使能每个覆盖位。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一点二十三分 | Reserved. | - | - | - |
| 22 | DM\_OVV | DM过电压 | RO | 0x0 |
| 21 | DP\_OVV | DP过电压 | RO | 0x0 |
| 20 | 公司简介 | DM过流 | RO | 0x0 |
| 19 | DP\_OVCN | DP过电流 | RO | 0x0 |
| 18 | RX\_DM | 双稳态 | RO | 0x0 |
| 17 | RX\_DP | DPP引脚状态 | RO | 0x0 |
| 16 | RX\_DD | 差分RX | RO | 0x0 |
| 15 | TX\_切换模式 | TX\_DMMODE =0:单端模式TX\_DMMODE =1:差分驱动模式（忽略TX\_DM、TX\_DM\_OE） | RW | 0x0 |
| 14 | TX\_FSSLEW | TX\_FSSLEW=0:低速压摆率TX\_FSSLEW=1:全速压摆率 | RW | 0x0 |
| 13 | TX\_PD | TX掉电覆盖（如果设置了覆盖使能）。1=断电。 | RW | 0x0 |
| 12 | RX\_PD | RX掉电覆盖（如果设置了覆盖使能）。1=断电。 | RW | 0x0 |
| 11 | TX\_DM | 输出数据。TX\_MODEMODE =1，忽略TX\_MODEMODE =0，仅驱动USB。TX\_DM\_OE=1以启用驱动器。TX =TX\_DM | RW | 0x0 |
| 10 | TX\_DP | 输出数据。如果TX\_MODE =1，则驱动DPP/ADDIFF对。TX\_DP\_OE=1以启用驱动器。DPP=TX\_DP，Δ =~TX\_DP  如果TX\_MODE =0，则仅驱动DPPTX\_DP\_OE=1以启用驱动器。DPP=TX\_DP | RW | 0x0 |
| 9 | TX\_DM\_OE | 输出使能。如果TX\_MODE =1，则忽略。  如果TX\_MODE =0，则OE仅用于USB0-Hi-Z状态下的电流 | RW | 0x0 |
| 8 | TX\_DP\_OE | 输出使能。如果TX\_MODE =1，则DPP/ADDIFF对的OE0 -Hi-Z状态下的DPP/PWM; 1 - DPP/PWM驱动  如果TX\_MODE =0，则OE仅适用于DPP0-DPP处于Hi-Z状态; 1 - DPP驱动 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 7 | Reserved. | - | - | - |
| 6 | DM\_PULLDN\_EN | DM下拉使能 | RW | 0x0 |
| 5 | DM\_PULLUP\_EN | DM上拉使能 | RW | 0x0 |
| 4 | DM\_PULLUP\_HISEL | 使能第二个DM上拉电阻。0-拉力= Rpu 2; 1-拉力= Rpu 1 + Rpu 2 | RW | 0x0 |
| 3 | Reserved. | - | - | - |
| 2 | DP\_PULLDN\_EN | DP下拉使能 | RW | 0x0 |
| 1 | DP\_PULLUP\_EN | DP上拉使能 | RW | 0x0 |
| 0 | DP\_PULLUP\_HISEL | 使能第二个DP上拉电阻。0-拉力= Rpu 2; 1-拉力= Rpu 1 + Rpu 2 | RW | 0x0 |

416号表USBPHY\_DIRECT\_OVE

RRIDE寄存器

#### [USB](#_bookmark162):USBPHY\_DIRECT\_OVERRIDE寄存器

**偏移**:0x80

描述

在usbphy\_direct中为每个控件启用

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - | - |
| 15 | TX\_切换模式\_覆盖\_EN |  | RW | 0x0 |
| 十四点十三分 | Reserved. | - | - | - |
| 12 | DM\_PULLUP\_OVERRIDE\_EN |  | RW | 0x0 |
| 11 | TX\_FSSLEW\_OVERRIDE\_EN |  | RW | 0x0 |
| 10 | TX\_PD\_OVERRIDE\_EN |  | RW | 0x0 |
| 9 | RX\_PD\_OVERRIDE\_EN |  | RW | 0x0 |
| 8 | TX\_DM\_OVERRIDE\_EN |  | RW | 0x0 |
| 7 | TX\_DP\_OVERRIDE\_EN |  | RW | 0x0 |
| 6 | TX\_DM\_OE\_OVERRIDE\_EN |  | RW | 0x0 |
| 5 | TX\_DP\_OE\_OVERRIDE\_EN |  | RW | 0x0 |
| 4 | DM\_PULLDN\_EN\_OVERRIDE\_EN |  | RW | 0x0 |
| 3 | DP\_PULLDN\_EN\_OVERRIDE\_EN |  | RW | 0x0 |
| 2 | DP\_PULLUP\_EN\_OVERRIDE\_EN |  | RW | 0x0 |
| 1 | DM\_PULLUP\_HISEL\_OVERRIDE\_EN |  | RW | 0x0 |
| 0 | DP\_PULLUP\_HISEL\_OVERRIDE\_EN |  | RW | 0x0 |

#### [USB](#_bookmark162):USBPHY\_TRIM寄存器

**偏移**:0x84

描述

用于调整USB phy下拉电阻的微调值

*417号桌USBPHY\_TRIM*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一点十三分 | Reserved. | - | - | - |
| 十二点八分 | DM\_PULLDN\_TRIM | 驱动到USBPHY的价值  DM下拉电阻微调控制  实验数据表明，复位值将工作，但如果需要，该寄存器允许调整 | RW | 0x1f |
| 七点五 | Reserved. | - | - | - |
| 4:0 | DP\_PULLDN\_TRIM | 驱动到USBPHY的价值  DP下拉电阻微调控制  实验数据表明，复位值将工作，但如果需要，该寄存器允许调整 | RW | 0x1f |

寄存器

418号表INTR寄存器

#### [USB](#_bookmark162):INTR寄存器

**偏移**:0x8c

描述

原始中断

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一点二十分 | Reserved. | - | - | - |
| 19 | EP\_STALL\_NAK | 当EP\_STATUS\_STALL\_NAK中的任何位置1时引发。通过清除EP\_STATUS\_STALL\_NAK中的所有位来清除 | RO | 0x0 |
| 18 | 中止\_完成 | 当ABORT\_DONE中的任何位被设置时引发通过清除ABORT\_DONE中的所有位来清除 | RO | 0x0 |
| 17 | 开发\_特种部队 | 设备每次接收SOF（帧起始）数据包时置1通过读取SOF\_RD清除 | RO | 0x0 |
| 16 | 设置\_目录 | 设备. 来源:SIE\_STATUS.SETUP\_REC | RO | 0x0 |
| 15 | DEV\_RESUME\_FR OM\_HOST | 当设备从主机接收到恢复时设置通过写入SIE\_STATUS.RESUME清除 | RO | 0x0 |
| 14 | DEV\_SUSPEND | 设备挂起状态更改时设置通过写入SIE\_STATUS清除 | RO | 0x0 |
| 13 | DEV\_CONN\_DIS | 设备连接状态更改时设置通过写入SIE\_STATUS.CONNECTED清除 | RO | 0x0 |
| 12 | 总线\_重置 | 来源:SIE\_STATUS.BUS\_DATA | RO | 0x0 |
| 11 | VBUS\_检测 | 来源:SIE\_STATUS.VBUS\_DETECTED | RO | 0x0 |
| 10 | 失速 | 来源:SIE\_STATUS.STALL\_REC | RO | 0x0 |
| 9 | 错误\_CRC | 来源:SIE\_STATUS.CRC\_ERROR | RO | 0x0 |
| 8 | 错误\_位\_填充F | 来源:SIE\_STATUS.BIT\_STUFF\_ERROR | RO | 0x0 |
| 7 | 错误\_接收\_溢出 | 来源:SIE\_STATUS.RX\_OVERFLOW | RO | 0x0 |
| 6 | ERROR\_RX\_TIMEOUT | 来源:SIE\_STATUS.RX\_TIMEOUT | RO | 0x0 |
| 5 | ERROR\_DATA\_SEQ | 来源:SIE\_STATUS.DATA\_SEQ\_ERROR | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 4 | BUFF\_STATUS | 当BUFF\_STATUS中的任何位被设置时引发 通过清除BUFF\_STATUS中的所有位来清除。 | RO | 0x0 |
| 3 | TRANS\_COMPLETE | 每次设置SIE\_STATUS.TRANS\_COMPUTER时引发。通过写入此位清除。 | RO | 0x0 |
| 2 | 主机\_SOF | 主机:每次主机发送SOF（帧开始）时引发通过读取SOF\_RD清除 | RO | 0x0 |
| 1 | 主页\_简历 | Host:当设备唤醒主机时引发通过写入SIE\_STATUS.RESUME清除 | RO | 0x0 |
| 0 | 主机连接器 | Host:当设备连接或断开连接时（即当SIE\_STATUS.SPEED更改时）引发通过写入SIE\_STATUS.SPEED清除 | RO | 0x0 |

表419。INTE寄存器

#### [USB](#_bookmark162):INTE寄存器

**偏移**:0x90

描述

中断使能

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一点二十分 | Reserved. | - | - | - |
| 19 | EP\_STALL\_NAK | 当EP\_STATUS\_STALL\_NAK中的任何位置1时引发。通过清除EP\_STATUS\_STALL\_NAK中的所有位来清除 | RW | 0x0 |
| 18 | 中止\_完成 | 当ABORT\_DONE中的任何位被设置时引发通过清除ABORT\_DONE中的所有位来清除 | RW | 0x0 |
| 17 | DEV\_SOF | 设备每次接收SOF（帧起始）数据包时置1通过读取SOF\_RD清除 | RW | 0x0 |
| 16 | 设置\_目录 | 设备. 来源:SIE\_STATUS.SETUP\_REC | RW | 0x0 |
| 15 | DEV\_RESUME\_FR OM\_HOST | 当设备从主机接收到恢复时设置通过写入SIE\_STATUS.RESUME清除 | RW | 0x0 |
| 14 | DEV\_SUSPEND | 设备挂起状态更改时设置通过写入SIE\_STATUS清除 | RW | 0x0 |
| 13 | DEV\_CONN\_DIS | 设备连接状态更改时设置通过写入SIE\_STATUS.CONNECTED清除 | RW | 0x0 |
| 12 | 总线\_重置 | 来源:SIE\_STATUS.BUS\_DATA | RW | 0x0 |
| 11 | VBUS\_检测 | 来源:SIE\_STATUS.VBUS\_DETECTED | RW | 0x0 |
| 10 | 失速 | 来源:SIE\_STATUS.STALL\_REC | RW | 0x0 |
| 9 | 错误\_CRC | 来源:SIE\_STATUS.CRC\_ERROR | RW | 0x0 |
| 8 | 错误\_位\_填充F | 来源:SIE\_STATUS.BIT\_STUFF\_ERROR | RW | 0x0 |
| 7 | 错误\_接收\_溢出 | 来源:SIE\_STATUS.RX\_OVERFLOW | RW | 0x0 |
| 6 | ERROR\_RX\_TIMEOUT | 来源:SIE\_STATUS.RX\_TIMEOUT | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 5 | 错误\_数据\_SEQ | 来源:SIE\_STATUS.DATA\_SEQ\_ERROR | RW | 0x0 |
| 4 | BUFF\_STATUS | 当BUFF\_STATUS中的任何位被设置时引发 通过清除BUFF\_STATUS中的所有位来清除。 | RW | 0x0 |
| 3 | TRANS\_COMPLETE | 每次设置SIE\_STATUS.TRANS\_COMPUTER时引发。通过写入此位清除。 | RW | 0x0 |
| 2 | 主机\_SOF | 主机:每次主机发送SOF（帧开始）时引发通过读取SOF\_RD清除 | RW | 0x0 |
| 1 | 主页\_简历 | Host:当设备唤醒主机时引发通过写入SIE\_STATUS.RESUME清除 | RW | 0x0 |
| 0 | 主机连接器 | Host:当设备连接或断开连接时（即当SIE\_STATUS.SPEED更改时）引发通过写入SIE\_STATUS.SPEED清除 | RW | 0x0 |

420号桌INTF寄存器

#### [USB](#_bookmark162):INTF寄存器

**偏移量**:0x 94

描述

中断强制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一点二十分 | Reserved. | - | - | - |
| 19 | EP\_STALL\_NAK | 当EP\_STATUS\_STALL\_NAK中的任何位置1时引发。通过清除EP\_STATUS\_STALL\_NAK中的所有位来清除 | RW | 0x0 |
| 18 | 中止\_完成 | 当ABORT\_DONE中的任何位被设置时引发通过清除ABORT\_DONE中的所有位来清除 | RW | 0x0 |
| 17 | DEV\_SOF | 设备每次接收SOF（帧起始）数据包时置1通过读取SOF\_RD清除 | RW | 0x0 |
| 16 | 设置\_目录 | 设备. 来源:SIE\_STATUS.SETUP\_REC | RW | 0x0 |
| 15 | DEV\_RESUME\_FR OM\_HOST | 当设备从主机接收到恢复时设置通过写入SIE\_STATUS.RESUME清除 | RW | 0x0 |
| 14 | DEV\_SUSPEND | 设备挂起状态更改时设置通过写入SIE\_STATUS清除 | RW | 0x0 |
| 13 | DEV\_CONN\_DIS | 设备连接状态更改时设置通过写入SIE\_STATUS.CONNECTED清除 | RW | 0x0 |
| 12 | 公交车 | 来源:SIE\_STATUS.BUS\_DATA | RW | 0x0 |
| 11 | VBUS\_检测 | 来源:SIE\_STATUS.VBUS\_DETECTED | RW | 0x0 |
| 10 | 失速 | 来源:SIE\_STATUS.STALL\_REC | RW | 0x0 |
| 9 | 错误\_CRC | 来源:SIE\_STATUS.CRC\_ERROR | RW | 0x0 |
| 8 | 错误\_位\_填充F | 来源:SIE\_STATUS.BIT\_STUFF\_ERROR | RW | 0x0 |
| 7 | 错误\_接收\_溢出 | 来源:SIE\_STATUS.RX\_OVERFLOW | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 6 | ERROR\_RX\_TIMEOUT | 来源:SIE\_STATUS.RX\_TIMEOUT | RW | 0x0 |
| 5 | ERROR\_DATA\_SEQ | 来源:SIE\_STATUS.DATA\_SEQ\_ERROR | RW | 0x0 |
| 4 | BUFF\_STATUS | 当BUFF\_STATUS中的任何位被设置时引发 通过清除BUFF\_STATUS中的所有位来清除。 | RW | 0x0 |
| 3 | TRANS\_COMPLETE | 每次设置SIE\_STATUS.TRANS\_COMPUTER时引发。通过写入此位清除。 | RW | 0x0 |
| 2 | 主机\_SOF | 主机:每次主机发送SOF（帧开始）时引发通过读取SOF\_RD清除 | RW | 0x0 |
| 1 | 主页\_简历 | Host:当设备唤醒主机时引发通过写入SIE\_STATUS.RESUME清除 | RW | 0x0 |
| 0 | 主机连接器 | Host:当设备连接或断开连接时（即当SIE\_STATUS.SPEED更改时）引发通过写入SIE\_STATUS.SPEED清除 | RW | 0x0 |

表421。INTS寄存器

#### [USB](#_bookmark162):INTS寄存器

**偏移**:0x98

描述

强制屏蔽后的屏蔽状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一点二十分 | Reserved. | - | - | - |
| 19 | EP\_STALL\_NAK | 当EP\_STATUS\_STALL\_NAK中的任何位置1时引发。通过清除EP\_STATUS\_STALL\_NAK中的所有位来清除 | RO | 0x0 |
| 18 | 中止\_完成 | 当ABORT\_DONE中的任何位被设置时引发通过清除ABORT\_DONE中的所有位来清除 | RO | 0x0 |
| 17 | 开发\_特种部队 | 设备每次接收SOF（帧起始）数据包时置1通过读取SOF\_RD清除 | RO | 0x0 |
| 16 | 设置\_目录 | 设备. 来源:SIE\_STATUS.SETUP\_REC | RO | 0x0 |
| 15 | DEV\_RESUME\_FR OM\_HOST | 当设备从主机接收到恢复时设置通过写入SIE\_STATUS.RESUME清除 | RO | 0x0 |
| 14 | DEV\_SUSPEND | 设备挂起状态更改时设置通过写入SIE\_STATUS清除 | RO | 0x0 |
| 13 | DEV\_CONN\_DIS | 设备连接状态更改时设置通过写入SIE\_STATUS.CONNECTED清除 | RO | 0x0 |
| 12 | 公交车 | 来源:SIE\_STATUS.BUS\_DATA | RO | 0x0 |
| 11 | VBUS\_检测 | 来源:SIE\_STATUS.VBUS\_DETECTED | RO | 0x0 |
| 10 | 失速 | 来源:SIE\_STATUS.STALL\_REC | RO | 0x0 |
| 9 | 错误\_CRC | 来源:SIE\_STATUS.CRC\_ERROR | RO | 0x0 |
| 8 | 错误\_位\_填充F | 来源:SIE\_STATUS.BIT\_STUFF\_ERROR | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 7 | 错误\_接收\_溢出 | 来源:SIE\_STATUS.RX\_OVERFLOW | RO | 0x0 |
| 6 | ERROR\_RX\_TIMEOUT | 来源:SIE\_STATUS.RX\_TIMEOUT | RO | 0x0 |
| 5 | 错误\_数据\_SEQ | 来源:SIE\_STATUS.DATA\_SEQ\_ERROR | RO | 0x0 |
| 4 | BUFF\_STATUS | 当BUFF\_STATUS中的任何位被设置时引发 通过清除BUFF\_STATUS中的所有位来清除。 | RO | 0x0 |
| 3 | TRANS\_COMPLETE | 每次设置SIE\_STATUS.TRANS\_COMPUTER时引发。通过写入此位清除。 | RO | 0x0 |
| 2 | 主机\_SOF | 主机:每次主机发送SOF（帧开始）时引发通过读取SOF\_RD清除 | RO | 0x0 |
| 1 | 主页\_简历 | Host:当设备唤醒主机时引发通过写入SIE\_STATUS.RESUME清除 | RO | 0x0 |
| 0 | 主机连接器 | Host:当设备连接或断开连接时（即当SIE\_STATUS.SPEED更改时）引发通过写入SIE\_STATUS.SPEED清除 | RO | 0x0 |

引用

* <http://www.usbmadesimple.co.uk/>
* <https://www.usb.org/document-library/usb-20-specification>
  1. UART

ARM文档

摘自《[PrimeCell电池（PL011）技术参考手册](https://developer.arm.com/documentation/ddi0183/latest/)》。经允许使用

RP 2040有2个相同的基于ARM Primecell SoC（PL 011）（版本r1p5）的SoC外设实例每个实例支持以下功能:

* 独立的32×8 Tx和32×12 RxFIFO
* 可编程波特率发生器，时钟由UART\_UART提供（参见第2.15.1）
* 标准异步通信位（开始、停止、奇偶校验）在发送时添加，在接收时删除
* 断线检测
* 可编程串行接口（5、6、7或8位）
* 1或2个停止位
* 可编程硬件流控制

每个GPIO都可以连接到[第2.19.2](#_bookmark23)GPIO多路复用[表](#_bookmark24)中定义的多个GPIO引脚。到GPIO多路复用的连接以UART实例名称uart0\_或uart1\_为前缀，包括以下内容

* 传输数据tx（以下章节中称为UARTTXD
* 接收到的数据流（在以下章节中称为UARTRXD
* 输出流量控制rts（在以下章节中称为nUARTRTS
* 输入流量控制cts（在以下部分中称为nUARTCTS

UARTCLK由PLL\_CLK驱动，PCLK由系统时钟PLL\_SYS驱动（参见第2.15.1节）。

* + 1. 概述

该公司执行:

* 对从外围设备接收的数据的串行到并行转换
* 传输到外围设备的数据的串行到串行转换

CPU通过AMBA APB接口读取和写入数据和控制/状态信息。发送和接收路径通过内部FIFO存储器进行缓冲，在发送和接收模式下最多可以独立存储32个字节。

第二名:

* 包括一个可编程波特率发生器，可从UART内部参考时钟输入UARTCLK生成公共发送和接收内部时钟
* 提供与行业标准16C650 MEMS器件类似的功能
* 在UART模式下支持最大波特率UARTCLK/16（125 MHz时为7.8Mbaud

UART操作和波特率值由线路控制寄存器UARTLCR\_H和波特率除数寄存器（UART波特率寄存器UARTIBRD和小数波特率寄存器UARTFBRD）控制。

可生成:

* 接收（包括超时）、发送、调制解调器状态和错误条件的可单独屏蔽中断
* 一个单一的组合中断，这样，如果任何单独的中断被断言，输出也被断言，并且不被屏蔽
* 用于与直接存储器存取（DMA）控制器接口的DMA请求信号。

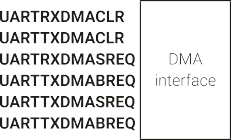
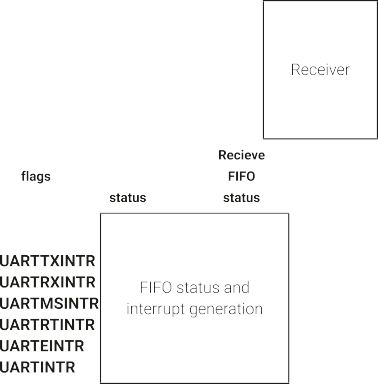
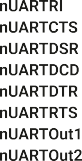
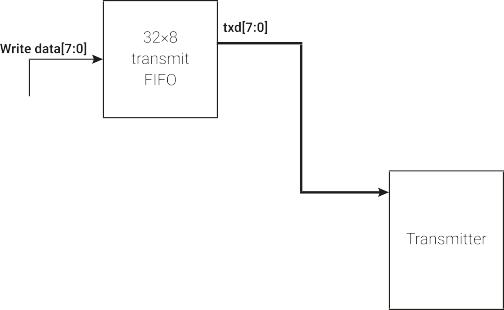
如果在接收过程中出现成帧、奇偶校验或中断错误，则会设置相应的错误位，并将其存储在FIFO中。如果发生溢出情况，溢出寄存器位立即置1，FIFO数据不会被覆盖。

您可以将FIFO编程为1字节深，提供传统的双缓冲FIFO接口。

有一个可编程硬件流控制功能，它使用nUARTCTS输入和nUARTRTS输出来自动控制串行数据流。

* + 1. 功能描述

*图59.图1框图。为清楚起见，未显示测试逻辑。*



* + - 1. AMBA APB接口

AMBA APB接口生成读写解码，用于访问状态/控制寄存器以及发送和接收FIFO。

* + - 1. 寄存器块

寄存器块存储通过AMBA APB接口写入或读取的数据。

* + - 1. 波特率发生器

波特率发生器包含产生内部时钟的自激计数器:Baud16和IrLPBaud16信号。Baud16提供定时信息，以进行无线电发射和接收控制。波特率16是一个脉冲流，宽度为一个UARTCLK时钟周期，频率为波特率的16倍

* + - 1. 发送FIFO

发送FIFO是一个8位宽、32位深的FIFO存储器缓冲区。通过APB接口写入的CPU数据存储在FIFO中，直到由发送逻辑读出 您可以禁用发送FIFO，使其充当单字节保持寄存器。

* + - 1. 接收FIFO

接收FIFO是一个12位宽、32位深的FIFO存储器缓冲区。接收逻辑将接收到的数据和相应的错误位存储在接收FIFO中，直到CPU通过APB接口读出接收FIFO可以

被禁用以充当单字节保持寄存器。

* + - 1. 发送逻辑

发送逻辑对从发送FIFO读取的数据执行并串转换根据控制寄存器中的编程配置，控制逻辑输出以起始位开始的串行位流，首先是最低有效位（LSB）的数据位，然后是奇偶校验位，然后是停止位

* + - 1. 接收逻辑

接收逻辑在检测到有效起始脉冲后对接收到的比特流执行串并转换。还执行溢出、奇偶校验、帧错误检查和断线检测，它们的状态伴随着写入接收FIFO的数据

* + - 1. 可编程逻辑

单个可屏蔽高电平有效中断由ADC产生组合中断输出作为各个中断请求的OR函数生成，并连接到处理器中断控制器。

更多信息见第4.2.6

* + - 1. DMA接口

如第4.2.5节中所述，DMA接口提供了一个连接到DMA控制器的接口

* + - 1. 同步寄存器和逻辑

该芯片支持时钟PCLK和UARTCLK的异步和同步操作。同步寄存器和握手逻辑已经实现，并且始终处于活动状态。这对性能或面积的影响最小。控制信号的同步在数据流的两个方向上执行，即从PCLK到UARTCLK域，以及从UARTCLK到PCLK域。

* + 1. 操作
       1. 时钟信号

为UARTCLK选择的频率必须适应所需的波特率范围:

* FUARTCLK（min）≥ 16 ×baud\_rate（max）
* FUARTCLK（max）≤ 16 × 65535 ×baud\_rate（min）

例如，对于110波特到460800波特的波特率范围，UARTCLK频率必须在7.3728 MHz到115.34 MHz之间。

UARTCLK的频率也必须在所使用的所有波特率所需的误差范围

PCLK与UARTCLK的时钟频率比也有限制。UARTCLK的频率不得超过PCLK频率的5/3倍

* FUARTCLK ≤ 5/3 ×FPCLK

例如，在UART模式下，若要在UARTCLK为14.7456 MHz时生成921600波特，则PCLK必须大于或等于8.85276 MHz。这确保了FIFO有足够的时间将接收到的数据写入接收FIFO。

* + - 1. 手术

控制数据写入到UART线路控制寄存器UARTLCR。该寄存器内部为30位宽，但可通过APB接口对以下寄存器进行外部访问

UARTLCR\_H寄存器定义:

* 传输参数
* 字长
* 缓冲区模式
* 传输停止位数
* 奇偶校验模式
* 打破世代。

UARTIBRD寄存器定义整数波特率分频器，UARTFBRD寄存器定义小数波特率分频器。

* + - * 1. 小数波特率分频器

波特率除数是一个22位数，由16位整数和6位小数部分组成。波特率发生器使用该值来确定位周期。小数波特率分频器允许使用任何频率> 3.6864 MHz的时钟作为UARTCLK，同时仍然可以生成所有标准波特率。

该16位整数被写入UART波特率寄存器UARTIBRD。6位小数部分写入小数波特率寄存器UARTFBRD。波特率除数与UARTCLK的关系如下

波特率除数= UARTCLK/（16×波特率）=其中是整数部分，小数部分用小数点分隔，如[图60](#_bookmark191)所示。

*图60. 波特率除数。*



计算6位数字（）的方法是，取所需波特率除数的小数部分，乘以64（即，其中是UARTFBRD寄存器的宽度），再加上0.5，以考虑舍入误差:



产生内部时钟使能信号Baud16，它是一个UARTCLK宽脉冲流，平均频率为所需波特率的16倍然后将该信号除以16以给出发送时钟。波特率除数中的低数字给出短位周期，波特率除数中的高数字给出长位周期。

* + - * 1. 数据发送或接收

接收或发送的数据存储在两个32字节FIFO中，尽管接收FIFO每个字符有额外的4位用于状态信息。对于传输，数据被写入传输FIFO。 如果使能UART，则会使数据帧开始传输，参数在线路控制寄存器UARTLCR\_H中指定。数据继续传输，直到传输FIFO中没有剩余数据一旦数据写入发送FIFO（即FIFO非空），BUSY信号即变为高BUSY仅在发送FIFO为空且已从移位寄存器发送完最后一个字符（包括停止位）时取反。BUSY可以置位为HIGH，即使不再使能UART

对于每个数据样本，取三个读数，并保留多数值在以下段落中，定义了中间采样点，并在其两侧采集一个样本

当接收器空闲（UARTRXD连续为1，处于标记状态）且在数据输入端检测到低电平（已接收到起始位）时，接收计数器（由Baud16使能时钟）开始运行，并在计数器的第八个周期（在SIR模式下）或计数器的第四个周期（在SIR模式下）对数据进行采样，以允许较短的逻辑0脉冲（位周期的一半）。

如果UARTRXD在波特率16的第八个周期仍为低电平，则起始位有效，否则检测到错误起始位，并将其忽略。

如果起始位有效，则根据数据字符的编程长度，在波特率16的每第16个周期（即一个位周期后）对连续数据位进行如果启用了奇偶校验模式，则检查奇偶校验位

最后，如果UARTRXD为高电平，则确认有效的停止位，否则发生帧错误。当接收到完整字时，数据存储在接收FIFO中，任何错误位都与该字相关联

* + - * 1. 错误位

三个错误位存储在接收FIFO的位[10:8]中，并与特定字符相关联还有一个指示溢出错误的附加错误，存储在接收FIFO的位11

* + - * 1. 超限钻头

422号桌接收FIFO位功能

溢出位与接收FIFO中的字符无关当FIFO已满且移位寄存器中完全接收到下一个字符时，设置溢出错误 移位寄存器中的数据被覆盖，但不会写入FIFO。当接收FIFO中有一个空位置可用，并且接收到另一个字符时，溢出位的状态将与接收到的字符一起复制到接收FIFO中。然后清除溢出状态。 [表422](#_bookmark192)列出了接收FIFO的位功能。

|  |  |
| --- | --- |
| 先进先出位 | 功能 |
| 11 | 超限指示器 |
| 10 | 中断错误 |
| 9 | 奇偶校验错误 |
| 8 | 成帧错误 |
| 7:0 | 接收数据 |

* + - * 1. 禁用FIFO

此外，您可以禁用FIFO。在这种情况下，FIFO的发送端和接收端具有1字节保持寄存器（FIFO的底部条目 当接收到一个字而前一个字尚未读取时，溢出位被设置。在此实现中，FIFO未被物理禁用，但标记被操纵以产生1字节寄存器的假象。禁用FIFO时，除非发送移位寄存器已在使用，否则对数据寄存器的写入将绕过保持寄存器。

* + - * 1. 系统和诊断测试

通过将控制寄存器UARTCR中的回送使能（LBE）位设置为1，可以对UART数据执行UART测试。

在UARTTXD上传输的数据在UARTRXD输入端接收。