







版权页

版权所有© 2020-2024 Raspberry Pi Ltd（原Raspberry Pi（Trading）Ltd.）

RP 2040微控制器的文档采用Creative Commons[Attribution-NoDerivatives 4.0International](https://creativecommons.org/licenses/by-nd/4.0/)（CC BY-ND）许可

版权所有© 2019 Synopsys，Inc.

All rights reserved.经允许使用Synopsys DesignWare是Synopsys，Inc.的注册商标部分版权所有© 2000-2001，2005，2007，2009，2011-2012，2016 ARM有限公司。

Allrightsreserved. 经允许使用。

发布日期：2024-05-02

构建版本：576 cee 3-clean

关于SDK

全文中的“SDK”指的是我们的[Raspberry Pi Pico SDK](https://github.com/raspberrypi/pico-sdk)。有关SDK的更多详细信息，请参阅**[Raspberry Pi Pico C/C++ SDK](https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf)**手册。文档中包含的源代码版权所有© 2020-2023 Raspberry Pi Ltd（前身为Raspberry Pi（Trading）Ltd.）并在[3-Clause BSD](https://opensource.org/licenses/BSD-3-Clause)许可证下获得许可

法律免责声明

“服务”以“按现状”和“按可得到”的基础提供。中华讲师网明确声明不作出任何种类的所有明示或暗示的保证，包括但不限于关于适销性、适用于某一特定用途和无侵权行为等方面的保证。13.责任范围。您明确理解和同意，中国标准件网不对因下述任一情况而发生的任何损害赔偿承担责任，包括但不限于利润、商誉、使用、数据等方面的损失或其他无形损失的损害赔偿（无论中国标准件网是否已被告知该等损害赔偿的可能性）

RPL保留随时对资源或其中描述的任何产品进行任何增强、改进、更正或任何其他修改的权利，恕不另行通知。

这些资源面向具有适当设计知识水平的熟练用户用户对自己选择和使用资源以及其中描述的产品的任何应用程序承担全部责任用户同意赔偿并保持RPL免受因使用资源而产生的所有责任，费用，损害或其他损失。

RPL授予用户仅与Raspberry Pi产品一起使用资源的权限。禁止将资源用于任何其他用途没有授权给任何其他RPL或其他第三方知识产权。

高风险活动。Raspberry Pi产品并非设计、制造或预期用于需要故障安全性能的危险环境，例如核设施、飞机导航或通信系统、空中交通管制、武器系统或安全关键应用（包括生命支持系统和其他医疗设备）的操作，在这些环境中，产品故障可能直接导致死亡、人身伤害或严重的物理或环境损害（“高风险活动”）。RPL特别声明不对高风险活动的适用性提供任何明示或暗示的保证，并且不对在高风险活动中使用或包含Raspberry Pi产品承担任何责任

Raspberry Pi产品受RPL[标准条款的](https://www.raspberrypi.com/terms-conditions-sale/)约束。RPL提供的资源不会扩展或以其他方式修改RPL的[标准条款](https://www.raspberrypi.com/terms-conditions-sale/)，包括但不限于声明和保证

表达在其中。

目录

[版权页1](#_bookmark0)

[免责声明1](#_bookmark1)

1. [一、导言](#_bookmark2) [9](#_bookmark2)
   1. [芯片为什么叫RP2040？](#_bookmark3) [9](#_bookmark3)
   2. [摘要.](#_bookmark4) [10](#_bookmark4)
   3. [芯片](#_bookmark6) [10](#_bookmark6)
   4. [引脚排列参考](#_bookmark8) [11](#_bookmark8)
      1. [引脚位置](#_bookmark9) [11](#_bookmark9)
      2. [引脚说明](#_bookmark10) [12](#_bookmark10)
      3. [GPIO功能](#_bookmark11) [13](#_bookmark11)
2. [系统描述](#_bookmark12) [15](#_bookmark12)
   1. [总线结构](#_bookmark13)（Bus Fabric） [15](#_bookmark13)
      1. [AHB-Lite Crossbar](#_bookmark15) [16](#_bookmark15)
      2. [原子寄存器访问](#_bookmark17)(Atomic Register Access) [18](#_bookmark17)
      3. APB总线 [18](#_bookmark18)
      4. [窄IO寄存器写入](#_bookmark19)(Narrow IO Register Writes) [18](#_bookmark19)
      5. 寄存器列表(List of Registers) [19](#_bookmark20)
   2. [地址映射](#_bookmark32) [24](#_bookmark32)
      1. [摘要.](#_bookmark33) [24](#_bookmark33)
      2. [细节](#_bookmark35) [25](#_bookmark35)
   3. [处理器子系统](#_bookmark43) [27](#_bookmark43)
      1. [SIO](#_bookmark44) [27](#_bookmark44)
      2. [中断](#_bookmark111) [60](#_bookmark111)
      3. [事件信号](#_bookmark112) [61](#_bookmark112)
      4. 调试(Debug) [61](#_bookmark113)
   4. Cortex-M0+ [62](#_bookmark115)
      1. [特征](#_bookmark116) [63](#_bookmark116)
      2. [功能说明](#_bookmark117) [64](#_bookmark117)
      3. [编程模型](#_bookmark119)(Programmer’s model) [69](#_bookmark119)
      4. [系统控制](#_bookmark123)(System control) [74](#_bookmark123)
      5. [NVIC](#_bookmark126) [74](#_bookmark126)
      6. [MPU](#_bookmark128) [76](#_bookmark128)
      7. 调试(Debug) [76](#_bookmark130)
      8. 寄存器列表(List of Registers) [77](#_bookmark131)
   5. [DMA](#_bookmark163) [91](#_bookmark163)
      1. [配置通道](#_bookmark164) [92](#_bookmark164)
      2. [启动通道](#_bookmark166) [94](#_bookmark166)
      3. [数据请求（DREQ）](#_bookmark170) [95](#_bookmark170)
      4. [中断](#_bookmark171) [97](#_bookmark171)
      5. [附加功能](#_bookmark172) [97](#_bookmark172)
      6. [示例用例](#_bookmark173) [98](#_bookmark173)
      7. [寄存器列表(List of Registers) 102](#_bookmark175)
   6. [存储器](#_bookmark209) [121](#_bookmark209)
      1. [ROM](#_bookmark210) [121](#_bookmark210)
      2. [SRAM](#_bookmark211) [122](#_bookmark211)
      3. Flash [123](#_bookmark213)
   7. [启动顺序](#_bookmark223)(Boot Sequence) [130](#_bookmark223)
   8. [Bootrom](#_bookmark224) [130](#_bookmark224)
      1. 处理器控制的启动顺序(Processor Controlled Boot Sequence) [131](#_bookmark225)
      2. 处理器核心1上的启动代码(Launching Code On Processor Core 1) [133](#_bookmark228)
      3. [Bootrom目录](#_bookmark229) [134](#_bookmark229)
      4. [USB大容量存储接口](#_bookmark235) [145](#_bookmark235)
      5. [USB PICOBOOT接口](#_bookmark237) [146](#_bookmark237)
   9. [电源](#_bookmark242) [152](#_bookmark242)
      1. [数字IO电源（IOVDD）](#_bookmark243) [153](#_bookmark243)
      2. [数字核心电源（DVDD）](#_bookmark244) [153](#_bookmark244)
      3. [片内稳压器输入电源（VREG\_VIN）](#_bookmark245) [153](#_bookmark245)
      4. [USB PHY电源（USB\_VDD）](#_bookmark246) [153](#_bookmark246)
      5. [ADC电源（ADC\_AVDD）](#_bookmark247) [154](#_bookmark247)
      6. [电源时序](#_bookmark248)(Power Supply Sequencing) [154](#_bookmark248)
      7. 供电计划(Power Supply Schemes) [154](#_bookmark249)
   10. [核心电源调节器](#_bookmark256) [157](#_bookmark256)
       1. [应用电路](#_bookmark258) [157](#_bookmark258)
       2. [操作模式](#_bookmark259) [158](#_bookmark259)
       3. [输出电压选择](#_bookmark261) [159](#_bookmark261)
       4. [状态](#_bookmark262) [159](#_bookmark262)
       5. [当前限制](#_bookmark263) [159](#_bookmark263)
       6. 寄存器列表(List of Registers) [159](#_bookmark264)
       7. [详细规格](#_bookmark269) [162](#_bookmark269)
   11. [电源控制](#_bookmark270) [162](#_bookmark270)
       1. [顶级时钟门](#_bookmark271)(Top-level Clock Gates) [162](#_bookmark271)
       2. [睡眠状态](#_bookmark272)(SLEEP State) [163](#_bookmark272)
       3. [休眠状态](#_bookmark273)(DORMANT State) [163](#_bookmark273)
       4. [内存断电](#_bookmark274)(Memory Power Down) [163](#_bookmark274)
       5. 编程模式(Programmer’s Model) [164](#_bookmark275)
   12. [芯片级复位](#_bookmark276) [165](#_bookmark276)
       1. [概览](#_bookmark277) [165](#_bookmark277)
       2. [上电复位](#_bookmark278)(Power-on Reset) [166](#_bookmark278)
       3. [欠压检测](#_bookmark281)(Brown-out Detection) [167](#_bookmark281)
       4. [电源监视(](#_bookmark286) Supply Monitor) [169](#_bookmark286)
       5. [外部复位](#_bookmark287) [169](#_bookmark287)
       6. [Rescue调试端口复位](#_bookmark288)(Rescue Debug Port Reset) [169](#_bookmark288)
       7. [最后一次重置的来源](#_bookmark289)(Source of Last Reset) [169](#_bookmark289)
       8. 寄存器列表(List of Registers) [170](#_bookmark290)
   13. 上电状态机(Power-On State Machine) [170](#_bookmark291)
       1. [概览](#_bookmark292) [170](#_bookmark292)
       2. [通电顺序](#_bookmark293)(Power On Sequence) [170](#_bookmark293)
       3. [寄存器控制](#_bookmark294) [171](#_bookmark294)
       4. [与Watchdog的](#_bookmark295)交互 171
       5. 寄存器列表(List of Registers) [171](#_bookmark296)
   14. [子系统重置](#_bookmark302) [174](#_bookmark302)
       1. [概览](#_bookmark303) [174](#_bookmark303)
       2. 编程模式(Programmer’s Model) [175](#_bookmark304)
       3. 寄存器列表(List of Registers) [177](#_bookmark305)
   15. [时钟](#_bookmark310) [180](#_bookmark310)
       1. [概览](#_bookmark311) [180](#_bookmark311)
       2. [时钟源](#_bookmark312)(Clock sources) [181](#_bookmark312)
       3. [时钟发生器](#_bookmark314)(Clock Generators) [184](#_bookmark314)
       4. [频率计数器](#_bookmark315)(Frequency Counter) [188](#_bookmark315)
       5. [Resus](#_bookmark317) [188](#_bookmark317)
       6. 编程模式(Programmer’s Model) [189](#_bookmark318)
       7. 寄存器列表(List of Registers) [195](#_bookmark321)
   16. 晶体振荡器（XOSC） 216
       1. 概览 216
       2. 用法 217
       3. 启动延迟(Startup Delay) 217
       4. XOSC计数器 218
       5. 休眠模式(DORMANT mode) 218
       6. 编程模式(Programmer’s Model) 218
       7. 寄存器列表(List of Registers) 219
   17. 环形振荡器（ROSC） 222
       1. 概览 222
       2. ROSC/XOSC的权衡 222
       3. 修改频率 222
       4. ROSC分频器 223
       5. 随机数生成器 223
       6. ROSC计数器 223
       7. 休眠模式 223
       8. 寄存器列表(List of Registers) 224
   18. PLL 228
       1. 概览 228
       2. 计算PLL参数 229
       3. 配置 232
       4. 寄存器列表(List of Registers) 234
   19. GPIO 236
       1. 概览 236
       2. 功能选择 237
       3. 中断 239
       4. 便签(Pads) 240
       5. 软件实例 240
       6. 寄存器列表(List of Registers) 244
   20. 系统信息 304
       1. 概览 304
       2. 寄存器列表(List of Registers) 304
   21. Syscfg 305
       1. 概览 305
       2. 寄存器列表(List of Registers) 306
   22. TBMAN 309
       1. 寄存器列表(List of Registers) 309
3. PIO 310
   1. 概览 310
   2. 程序员模型 311
      1. PIO方案 311
      2. 控制流 312
      3. 寄存器 313
      4. 拖延 316
      5. 引脚映射 317
      6. IRQ标志 317
      7. 状态机之间的交互 317
   3. PIO汇编程序（pioasm） 318
      1. 指令(Directives) 318
      2. 数值(Values) 319
      3. 表达式（Expressions） 319
      4. 注释(Comments) 319
      5. 标签( Labels) 319
      6. 指令(Instructions) 320
      7. 伪指令(Pseudoinstructions) 320
   4. 指令集(Instruction Set) 320
      1. 概要 320
      2. JMP 321
      3. WAIT 322
      4. IN 323
      5. OUT 324
      6. PUSH 325
      7. PULL 326
      8. MOV 327
      9. IRQ 328
      10. SET 329
   5. 功能细节 330
      1. 端口设置(Side-set) 330
      2. 程序包装 331
      3. FIFO连接 333
      4. Autopush和Autopull 334
      5. 时钟分频器 338
      6. GPIO映射 339
      7. 强制和执行指令 341
   6. 示例 343
      1. 双工SPI 343
      2. WS2812LED 347
      3. TX 348
      4. RX 351
      5. 曼彻斯特串行TX和RX 354
      6. 差分曼彻斯特（BMC）TX和RX 356
      7. I2C 359
      8. PWM 363
      9. Addition 365
      10. 其他示例 366
   7. 寄存器列表(List of Registers) 368
4. 外围设备 382
   1. USB 382
      1. 概览 382
      2. 建筑 383
      3. 程序员模型(Programmer’s Model) 394
      4. 寄存器列表(List of Registers) 398

参考文献 416

* 1. 通用异步收发(UART) 416
     1. 概述 417
     2. 功能描述 417
     3. 操作 419
     4. 可编程硬件流控制 422
     5. DMA接口 423
     6. 中断 424
     7. 编程模式(Programmer’s Model) 426
     8. 寄存器列表(List of Registers) 428
  2. I2C 440
     1. 特点 441
     2. IP配置 441
     3. I2C概述 442
     4. I2C术语 444
     5. I2C行为 444
     6. I2C协议 446
     7. Tx FIFO管理和START、STOP和RESTART生成 449
     8. 多主仲裁 451
     9. 时钟同步 452
     10. 操作模式 453
     11. 尖峰抑制 458
     12. 快速模式加操作 459
     13. 总线清除功能 459
     14. IC\_CLK频率配置 460
     15. DMA控制器接口
     16. 寄存器的操作 465
     17. 寄存器列表(List of Registers) 465
  3. SPI 503
     1. 概览 504
     2. 功能描述 505
     3. 操作 507
     4. 寄存器列表(List of Registers) 517
  4. 脉宽调制 524
     1. 概览 524
     2. 编程模式(Programmer’s Model) 525
     3. 寄存器列表(List of Registers) 532
  5. 计时器 537
     1. 概览 537
     2. 计数器 538
     3. 警报 538
     4. 程序员模式(Programmer’s Model) 539
     5. 寄存器列表(List of Registers) 542
  6. 看门狗 547
     1. 概览 547
     2. Tick生成(Tick generation) 547
     3. 看门狗计数器 548
     4. 暂存寄存器(Scratch Registers) 548
     5. 编程模式(Programmer’s Model) 548
     6. 寄存器列表(List of Registers) 550
  7. RTC 551
     1. 存储格式 551
     2. 闰年 552
     3. 中断 552
     4. 参考时钟 552
     5. 编程模式(Programmer’s Model) 553
     6. 寄存器列表(List of Registers) 556
  8. ADC和温度传感器 560
     1. ADC控制器 561
     2. SARADC 562
     3. ADCENOB 564
     4. INL和DNL 565
     5. 温度传感器 566
     6. 寄存器列表(List of Registers) 567
  9. SSI 570
     1. 概览 571
     2. 特点 572
     3. IP修改 572
     4. 时钟比率(Clock Ratios) 574
     5. 发送和接收FIFO缓冲器 574
     6. 32位帧大小支持 575
     7. SSI中断 575
     8. 传输模式(Transfer Modes) 576
     9. 操作模式(Operation Modes) 577
     10. 合作伙伴连接接口( Partner Connection Interfaces) 582
     11. DMA控制器接口 601
     12. APB接口 600
     13. 寄存器列表(List of Registers) 601

1. 电气和机械 610
   1. 包装 610
      1. 热特性 611
      2. 推荐PCB封装 611
      3. 包装标记 611
   2. 储存条件 612
   3. 焊料轮廓 612
   4. 合规性 614
   5. 引脚说明 614
      1. 引脚位置 614
      2. 引脚定义 615
      3. 引脚规格 617
   6. 电源 625
   7. 耗电量 625
      1. 外围设备功耗 625
      2. 典型用户案例的功耗 626

附录A：寄存器字段类型 628

标准型 628

RW 628

RO 628

WO 628

清除类型 628

SC 628

WC 628

FIFO类型 628

RF 628

WF 629

RWF 629

附录B：勘误表 630

Bootrom 630

RP2040-E9 630

RP2040-E14 630

钟表 631

RP2040-E7 631

RP2040-E10 631

DMA 632

RP2040-E12 632

RP2040-E13 632

GPIO/ADC 633

RP2040-E6 633

RP2040-E11 633

USB 633

RP2040-E2 633

RP2040-E3 634

RP2040-E4 634

RP2040-E5 634

RP2040-E15 636

看门狗 637

RP2040-E1 637

XIPFlash 637

RP2040-E8 637

附录C：可用性 638

支撑 638

订购代码 638

附录D：文档发布历史 639

第1介绍

微控制器将软件世界连接到硬件世界。它们允许开发人员编写软件，该软件以与数字逻辑相同的确定性、周期精确的方式与物理世界进行交互。它们占据了性价比空间的左下角，其销量是功能更强大的同类产品的十倍。它们是推动我们世界数字化转型的主力军

RP2040是Raspberry Pi推出的首款微控制器。它为微控制器领域带来了高性能、低成本和易用性的标志性价值观。

它具有大容量片内存储器、对称双核处理器、确定性总线结构以及丰富的外设集，并通过我们独特的可编程I/O（PIO）子系统增强，为专业用户提供无与伦比的功能和灵活性。它有详细的文档，抛光的MicroPython端口和ROM中的UF2引导程序，对于初学者和业余爱好者用户来说，它具有最低的入门门槛

RP 2040是一款无状态设备，支持从外部QSPI存储器缓存就地执行。这一设计决策使您能够为应用选择适当密度的非易失性存储，并从低价位的商用闪存部件中获益

RP2040采用现代化40纳米工艺节点制造，具有高性能、低动态功耗和低泄漏特性，并具有多种低功耗模式，可支持电池供电的长时间运行。

主要特点：

* 双ARM Cortex-M0+@133MHz
* 264kB片内SRAM，6个独立存储体
* 通过专用QSPI总线支持高达16 MB的片外闪存
* DMA控制器
* 全连接AHB交叉开关
* 插值器和整数除法器外设
* 片内可编程LDO产生内核电压
* 2个片内PLL，用于生成USB和内核时钟
* 30个GPIO引脚，其中4个可用作模拟输入
* 外围设备
  + 2个UART
  + 2个SPI控制器
  + 2个I2C控制器
  + 16个PWM通道
  + USB 1.1控制器和PHY，支持主机和设备
  + 8个PIO状态机

无论您的微控制器应用是什么，从机器学习到电机控制，从农业到音频，RP 2040都具有让您的产品飞起来的性能，功能集和支持

* 1. 芯片为什么叫RP2040？

RP 2040上的后缀数字来自以下内容，

1. 处理器核心数（2）
2. 不确定是哪种类型的处理器（M0+）
3. 地板（log 2（ram/16 k））
4. floor（log 2（非易失性/16 k））或0（如果没有板载非易失性存储）参见[图1](#_bookmark5)。

RP 2 0 40

floor（log2（nonvolatile/16k））floor（log2（ram / 16k））

核心类型（例如M0+）核心数量

RaspberryPi

图1.RP2040芯片名称的解释。

* 1. 总结

RP 2040是一款低成本、高性能的微控制器器件，具有灵活的数字接口。主要特点：

* 双核Cortex M0+处理器内核，最高133 MHz
* 264kB嵌入式SRAM，6个存储体
* 30多功能GPIO
* 6个SPI闪存专用IO（支持XIP）
* 常用外设的专用硬件
* 可编程IO，用于扩展外设支持
* 内置温度传感器的4通道ADC， 500ksps，12位转换
* USB 1.1主机/设备
  1. 芯片

RP 2040具有双M0+处理器内核、DMA、内部存储器和通过AHB/APB总线结构连接的外围模块。

*图2.RP 2040芯片的系统概述*



代码可以通过专用SPI、DSPI或QSPI接口直接从外部存储器执行。小型缓存可提高典型应用程序的性能。

可通过SWD界面使用该工具。

内部SRAM可以包含代码或数据。它被寻址为单个264 kB区域，但在物理上被划分为6个存储体，以允许来自不同主机的同时并行访问。

DMA总线主控器可用于从处理器卸载重复的数据传输任务GPIO引脚可以直接驱动，也可以通过各种专用逻辑功能驱动。

用于固定功能的专用硬件，如SPI、I2C、UART。

灵活可配置的PIO控制器可用于提供各种IO功能。

具有嵌入式PHY的USB控制器可用于在软件控制下提供FS/LS主机或设备连接四个ADC输入，与GPIO引脚共享。

两个PLL为USB或ADC提供固定的48MHz时钟，以及高达133MHz的灵活系统时钟。

内部电压调节器提供核心电压，因此最终产品只需要提供IO电压。

* 1. 引脚排列参考

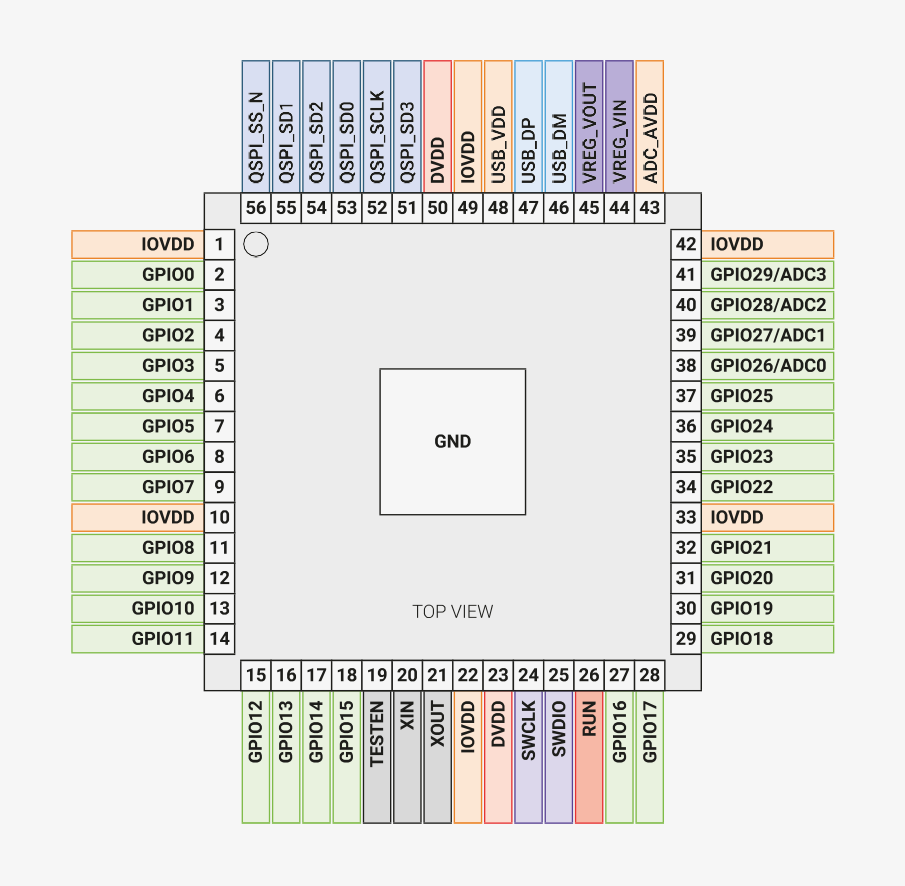
本节提供引脚排列和引脚功能的快速参考完整的细节，包括电气规格和包装图纸，可以在第5章中找到。

* + 1. 引脚位置

图3.RP 2040引脚说明QFN-567× 7 mm（减小ePad尺寸）

*表1. 这里简要介绍每个引脚的功能。 完整的电气规格见第5章。*

|  |  |
| --- | --- |
| 名称 | 描述 |
| GPIOx | 通用数字输入和输出。RP 2040可以将多个内部外设之一连接到每个GPIO，或直接从软件控制GPIO |
| GPIOx/ADCy | 通用数字输入输出，具有模数转换功能。RP 2040 ADC具有一个模拟多路复用器，可以选择这些引脚中的任何一个，并对电压进行采样 |
| QSPIx | 与SPI、双SPI或四SPI闪存器件的接口，支持就地执行。如果闪存访问不需要这些引脚，则这些引脚也可以用作软件控制的GPIO |
| USB\_DM和USB\_DP | USB控制器，支持全速设备和全速/低速主机。每个引脚上需要一个27Ω串联端接电阻，但内部提供总线上拉和下拉 |
| XIN和XOUT | 将晶体连接到RP 2040的晶体振荡器。XIN也可用作单端CMOS时钟输入，XOUT断开。USB引导程序需要12MHz晶振或12MHz时钟输入。有关推荐的晶体，请参见晶体振荡器（第2.16）。 |
| 运行 | 全局异步复位引脚。低电平驱动时复位，高电平驱动时运行。如果不需要外部复位，此引脚可直接连接到IOVDD。 |
| SWCLK和SWDIO | 访问内部Serial Wire USB多点总线。提供对两个处理器的调试访问 |
| 测试 | 出厂测试模式引脚。接地。 |
| GND | 单个外部接地连接，连接到RP 2040芯片上的多个内部接地焊盘。 |
| IOVDD | 数字GPIO电源，标称电压1.8V至3.3V |



* + 1. 引脚描述

|  |  |
| --- | --- |
| 名称 | 描述 |
| USB\_VDD | 内部USB全速PHY电源，标称电压3.3V |
| ADC\_AVDD | 模数转换器电源，标称电压3.3V |
| VREG\_VIN | 内部核心电压调节器的电源输入，标称电压1.8V至3.3V |
| VREG\_VOUT | 内部核心电压调节器的电源输出，标称电压1.1V，最大电流100mA |
| DVDD | 数字核心电源，标称电压1.1V。可以连接到VREG\_VOUT或其他板级电源。 |

* + 1. GPIO功能

*表2. 通用输入/输出（GPIO）Bank 0*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 |
| 21 | SPI0CSn | UART1RX | I2C0SCL | PWM2B | SiO | PIO0 | PIO 1 | 时钟GPOUT0 | USB OVCURDET |

功能

每个GPIO引脚都可以通过下面定义的GPIO功能连接到内部外设。 一些内部外围设备连接出现在多个位置，以提供一定的系统级灵活性。SIO、PIO 0和PIO 1可以连接到所有GPIO引脚，并由软件（或软件控制的状态机）控制，因此可以用于实现许多功能。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 功能 | | | | | | | | |
| 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3. GPIObank0功能说明*

|  |  |
| --- | --- |
| 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PL011 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bank 0 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。可以驱动多个内部时钟（包括PLL输出）到GPIO，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 |

第2章.系统描述

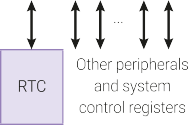
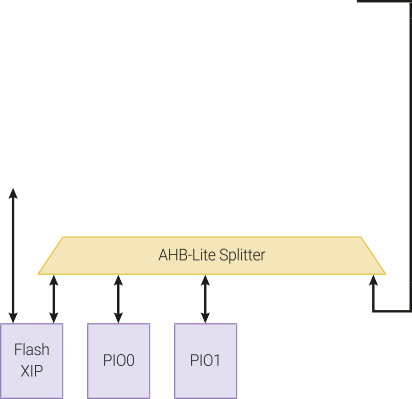
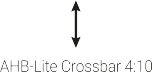
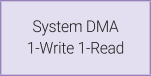
本章介绍RP 2040的关键系统功能，包括处理器、内存、模块连接方式、时钟、复位、电源和IO。有关概览图，请参阅[图2](#_bookmark7)

* 1. 总线结构

RP2040总线结构在芯片上路由地址和数据。

[图4](#_bookmark14)显示了总线结构的高级结构。主AHB-Lite crossbar在其4个上游端口和10个下游端口之间路由地址和数据：每个周期最多可以进行4次总线传输。所有数据路径均为32位宽。存储器设备在主交叉开关上具有专用端口，以满足其高带宽要求。高带宽AHB-Lite外设在交叉开关上有一个共享端口，APB桥提供对系统控制寄存器和低带宽外设的总线访问。

*图4. RP2040总线结构概述。*



总线结构连接4个AHB-Lite主机，即生成地址的设备

* 处理器核心0
* 处理器芯1
* DMA控制器读端口
* DMA控制器写端口

这些端口被路由到主交叉开关上的10个下游端口

* ROM
* FlashXIP
* SRAM 0到5（每个端口
* 快速AHB-Lite外设：PIO 0、PIO 1、USB、DMA控制寄存器、XIP aux（一个共享端口）
* 桥接至所有APB外设和系统控制寄存器

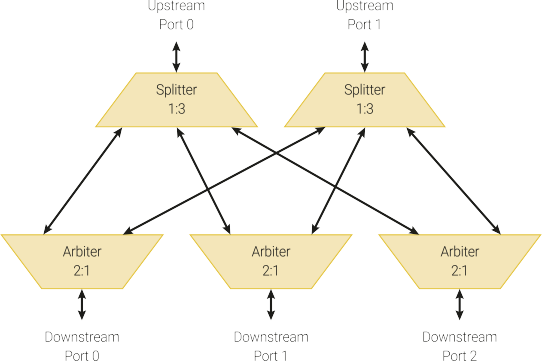
四个总线主控器可以同时访问任意四个*不同的*交叉开关端口，总线结构不增加等待

状态到任何AHB-Lite从机访问。因此，在125 MHz的系统时钟下，最大持续总线带宽为2.0 GBps。 系统地址映射已被安排为使这种并行带宽可用于尽可能多的软件用例-例如，条带化SRAM别名（[第2.6.2节](#_bookmark211)）将主存储器访问分散在四个交叉开关端口（SRAM 0.3）上，以便更多的存储器访问可以并行进行。

* + 1. AHB-Lite横杆

RP 2040总线结构的中心是一个4：10全连接的交叉开关。它的4个上行端口连接到4个系统总线主机，10个下行端口连接到最高带宽的AHB-Lite从机（即存储器接口）和结构的较低层[图5](#_bookmark16)显示了2：3 AHB-Lite交叉开关的结构，其排列方式与RP 2040上的4：10交叉开关相同，但更容易在图中显示

*图5.一个2：3的AHB-Lite横杆。 每个上游端口都连接到一个分离器，该分离器将总线请求路由到3个下游端口之一，并将响应路由回来。 每个下游端口都连接到一个仲裁器，该仲裁器安全地管理对该端口的并发访问。*



横杆由两个组件构成

* 分路器
  + 执行粗地址解码
  + 将请求（地址、写入数据）路由到初始地址解码所指示的下游端口
  + 将来自正确仲裁器的响应（读取数据、总线错误）路由回上游端口
* 仲裁器
  + 管理对下游端口的并发请求
  + 将响应（读取数据、总线错误）路由到正确的拆分器
  + 实施公交优先规则

RP 2040上的主交叉开关由4个1：10分配器和10个4：1仲裁器组成，它们之间有40个AHB-Lite总线通道。注意，由于AHB-Lite是流水线总线，因此分离器可能正在从下游端口A路由回对先前请求的响应，而对下游端口B的新请求已经在进行中。这不会导致任何循环损失。

* + - 1. 公交优先

主AHB-Lite交叉开关中的仲裁器实现两级总线优先级方案。使用BUSCTRL寄存器块中的[BUS\_PRIORITY](#_bookmark22)寄存器为每个主机配置优先级

当存在对同一仲裁器的多个同时访问时，来自高优先级主机（优先级1）的任何请求将在来自低优先级主机（优先级0）的任何请求之前被考虑如果相同优先级的多个主设备试图同时访问相同的从设备，则应用循环平局打破，即仲裁器依次授予每个主机访问权。

##### 注意

优先级仲裁仅适用于在同一周期内尝试访问**同一**从机的多个主机对不同的从设备（例如，不同的SRAM存储体）的访问可以同时进行。

当访问具有零等待状态的从机时，例如SRAM（即每个系统时钟周期可以访问一次），高优先级主机将永远不会观察到由低优先级主机访问引起的任何减速或其他时序效应这允许在硬实时用例中*保证*延迟和吞吐量;然而，这确实意味着低优先级主机可能会停止，直到有一个空闲周期。

* + - 1. 总线性能计数器

性能计数器自动计算对主AHB-Lite交叉开关仲裁器的访问。这可以帮助诊断高流量用例中的性能问题。

有四个性能计数器。每个都是一个24位饱和计数器。可以从BUSCTRL\_PERFCTRx读取计数器值，并通过向BUSCTRL\_PERFCTRx写入任何值来清除计数器值。每个计数器一次可以计数20个可用事件中的一个，由BUSCTRL\_PERFSELx选择。可用的总线事件包括：

|  |  |  |
| --- | --- | --- |
| 完美  X | 事件 | 描述 |
| 0 | 全境通告访问，有争议 | 完成对APB仲裁器（位于所有APB外设的上游）的访问，之前由于另一个主机的访问而延迟 |
| 1 | APB访问 | 完成对APB仲裁器的访问 |
| 2 | FASTPERI访问，有争议 | 完成对FASTPERI仲裁器（位于PIO、DMA配置端口、USB、XIP辅助FIFO端口的上游）的访问，之前由于另一个主机的访问而延迟 |
| 3 | FASTPERI访问 | 完成对FASTPERI仲裁器的访问 |
| 4 | SRAM5访问，有争议 | 完成对SRAM 5仲裁器的访问，之前由于另一个主机的访问而延迟 |
| 5 | SRAM 5访问 | 完成对SRAM 5仲裁器的访问 |
| 6 | SRAM4访问，有争议 | 完成对SRAM 4仲裁器的访问，之前由于另一个主机的访问而延迟 |
| 7 | SRAM 4访问 | 完成对SRAM 4仲裁器的访问 |
| 8 | SRAM 3访问，有争议 | 完成对SRAM 3仲裁器的访问，之前由于另一个主机的访问而延迟 |
| 9 | SRAM 3访问 | 完成对SRAM 3仲裁器的访问 |
| 10 | SRAM2访问，有争议 | 完成对SRAM 2仲裁器的访问，之前由于另一个主机的访问而延迟 |
| 11 | SRAM 2访问 | 完成对SRAM 2仲裁器的访问 |
| 12 | SRAM1访问，有争议 | 完成对SRAM 1仲裁器的访问，之前由于另一个主机的访问而延迟 |
| 13 | SRAM 1存取 | 完成对SRAM 1仲裁器的访问 |
| 14 | SRAM 0访问，有争议 | 完成对SRAM 0仲裁器的访问，之前由于另一个主机的访问而延迟 |
| 15 | SRAM 0访问 | 完成对SRAM 0仲裁器的访问 |

|  |  |  |
| --- | --- | --- |
| 完美  X | 事件 | 描述 |
| 16 | XIP\_MAIN访问，有争议 | 完成对XIP\_MAIN仲裁器的访问，之前由于另一个主机的访问而延迟 |
| 17 | XIP\_MAIN访问 | 完成对XIP\_MAIN仲裁器的访问 |
| 18 | ROM访问，有争议 | 完成对ROM仲裁器的访问，之前由于另一个主机的访问而延迟 |
| 19 | 只读存储器存取 | 完成对ROM仲裁器的访问 |

* + 1. 原子寄存器访问

每个外设寄存器块分配有4kB的地址空间，寄存器访问使用4种方法之一，通过地址解码选择。

* Addr+0x0000：正常读写访问
* Addr +0x1000：写入时的原子XOR
* Addr +0x2000：写入时设置原子位掩码
* Addr +0x3000：写入时清除原子位掩码

这允许修改控制寄存器的各个字段，而无需在软件中执行读取-修改-写入序列：而是将更改发布到外围设备，并在原位执行。如果没有这种能力，当中断服务例程与前台运行的代码并发时，或者当两个处理器并行运行代码时，很难安全地访问IO寄存器

四个原子访问别名总共占用16kB。RP 2040上的大多数外设都提供了这种功能，原子写入与正常的读/写访问具有相同的时序。一些外设（I2C、UART、SPI和SSI）使用总线插入器添加了此功能，该总线插入器在外设的边界处将上游原子写入转换为这将访问时间延长了两个系统时钟周期。

SIO（[第2.3.1节](#_bookmark44)）是一个直接连接到内核IO端口的单周期IO块，**不**支持总线级的原子访问，尽管一些单独的寄存器（例如，GPIO）具有set/clear/xor别名。

* + 1. APB桥

APB桥接器将高速主AHB-Lite互连连接到低带宽外设。虽然AHB-Lite结构在任何地方都提供零等待状态访问，但APB访问具有周期损失：

* APB总线访问至少需要两个周期（设置阶段和访问阶段）
* 当总线请求和响应被注册时，桥向读访问添加额外的周期
* 桥增加了**两**个额外的周期来写访问，因为APB设置阶段在AHB-Lite写数据有效之前不能开始

因此，总线结构的APB部分的吞吐量略低于AHB-Lite部分。但是，有足够的带宽使APB串行外设饱和

* + 1. 窄IO寄存器写入

RP 2040上的内存映射IO寄存器忽略总线读/写访问的宽度。它们将所有写入视为32位大小。这意味着软件不能使用字节或半字写入来修改IO寄存器的一部分：对30个地址MSB与寄存器地址匹配的地址的任何写入都会影响整个IO寄存器的内容。

注册.

要更新IO寄存器的一部分，而不需要读-修改-写序列，RP 2040上的最佳解决方案是原子设置/清除/XOR（见[第2.1.2](#_bookmark17)）。请注意，这比字节或半字写入更灵活，因为任何字段组合

在8位或16位写入时（例如Cortex-M0+上的strb指令），IO寄存器将对整个32位写入数据总线进行采样。RP 2040上的Cortex-M0+和DMA将始终在总线上复制窄数据：

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/system/narrow\_io\_write/narrow\_io\_write.c](https://github.com/raspberrypi/pico-examples/blob/master/system/narrow_io_write/narrow_io_write.c#L19-L62)第19-62

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

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62}

stdio\_init\_all（）;

//我们将使用WATCHDOG\_SCRATCH 0作为方便的32位读/写寄存器

//我们可以将任意值赋给

io\_rw\_32 \*scratch32 = watchdog\_hw->scratch[0];

//将暂存寄存器设置为两个半字，偏移量分别为+0x0和+0x2

volatile uint16\_t\*scratch16 =（ volatile uint16\_t\*）scratch32;

//将暂存寄存器设置为偏移量为+0x0、+0x1、+0x2、+0x3的四个字节

volatile uint8\_t\*scratch8 =（ volatile uint8\_t\*）scratch32;

//显示我们可以正常读/写暂存器

printf（“写入32位值\n”）;

\*scratch32 =0xdeadbeef;

printf（“应该是0xdeadbeef：0x%08x\n”，\* scratch 32）;

//我们可以很好地进行窄读-IO寄存器将其视为32位

//读取，处理器/DMA将根据

//根据传输大小和地址LSB

printf（“\n每次读回1个字节）;

//小端！

printf（“Should be ef be ad de：%02x“，scratch8[0]）; printf（“%02x“，scratch8[1]）;

printf（“%02x\n“，scratch8[2]）;printf（“%02x\n”，scratch8[3]）;

//Cortex-M0+和RP 2040 DMA通过总线复制字节写入

*//和IO寄存器将始终对整个写总线进行采样。*printf（“\n在偏移量0\n处写入8位值0xa 5”）;scratch 8 [0]=0xa 5;

//一次性读回整个暂存器

printf（“应该是0xa 5a 5a 5a 5：0x%08x\n”，\* scratch 32）;

//IO寄存器忽略地址LSB [1：0]以及传输

*//size，所以我们使用什么字节偏移量并不*重要printf（“\n Writing 8 bit value at offset 1\n”）;scratch 8 [1]=0x 3c;

printf（“应该是0x 3c 3c 3c 3c：0x%08x\n”，\* scratch 32）;

//半字写入也跨写入数据总线复制

printf（“\n在偏移量0\n处写入 16位值）; scratch 16 [0] =0xf 00 d;

printf（“应该是0xf 00 df 00 d：0x%08x\n”，\* scratch 32）;

* + 1. 登记册一览表

总线结构寄存器从基址0x40030000（在SDK中定义为[BUSCTRL\_BASE](#_bookmark39)）开始。

*表4.BUSCTRL寄存器列表*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [总线\_优先级](#_bookmark22) | 为总线仲裁设置每个主机的优先级 |
| 0x04 | [总线优先级确认](#_bookmark23) | 总线优先级确认 |
| 0x08 | [性能0](#_bookmark24) | 总线结构性能计数器0 |
| 0x0C | [PERFSEL0](#_bookmark25) | PERFCTR 0的总线结构性能事件选择 |
| 0x10 | [性能1](#_bookmark26) | 总线结构性能计数器1 |
| 0x14 | [PERFSEL1](#_bookmark27) | PERFCTR 1的总线结构性能事件选择 |
| 0x18 | [PERFCTR2](#_bookmark28) | 总线结构性能计数器2 |
| 0x1c | [PERFSEL2](#_bookmark29) | PERFCTR 2的总线结构性能事件选择 |
| 0x20 | [PERFCTR3](#_bookmark30) | 总线结构性能计数器3 |
| 0x24 | [PERFSEL3](#_bookmark31) | PERFCTR 3的总线结构性能事件选择 |

表5.总线\_优先级

寄存器

表6.总线优先级确认

寄存器

### BUS\_PRIORITY寄存器

**偏移**：0x00

描述

为总线仲裁设置每个主机的优先级

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:十三分 | Reserved. | - | - | - |
| 12 | DMA\_W | 0-低优先级，1 -高优先级 | RW | 0x0 |
| 11:九分 | Reserved. | - | - | - |
| 8 | DMA\_R | 0-低优先级，1 -高优先级 | RW | 0x0 |
| 七点五 | Reserved. | - | - | - |
| 4 | Proc1 | 0-低优先级，1 -高优先级 | RW | 0x0 |
| 3：1 | Reserved. | - | - | - |
| 0 | 100 | 0-低优先级，1 -高优先级 | RW | 0x0 |

### [BUSCTRL](#_bookmark21)：BUS\_PRIORITY\_ACK寄存器

**偏移**：0x04

描述

总线优先级确认

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:一 | Reserved. | - | - |
| 0 | 一旦所有仲裁器都注册了新的全局优先级，则变为1仲裁器在服务新的非顺序访问时更新其本地优先级。在正常情况下，这几乎会立即发生。 | RO | 0x0 |

### [BUSCTRL](#_bookmark21)：PERFCTR 0寄存器

**偏移**：0x08

表7.性能0

寄存器

表8.PERFSEL0

寄存器

描述

总线结构性能计数器0

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - |
| 23：0 | 总线结构饱和性能计数器0  对来自总线结构仲裁器的一些事件信号进行计数。  写入任何要清除的值使用PERFSEL 0选择要计数的事件 | WC | 0x000000 |

### [BUSCTRL](#_bookmark21)：PERFSEL 0寄存器

**偏移**：0x0c

描述

PERFCTR 0的总线结构性能事件选择

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:5 | Reserved. | - | - |
| 4:0 | 选择PERFCTR 0的事件对主交叉开关的下游端口上的竞争访问或所有访问进行  0x00→ apb\_contested 0x01→ apb  0x02→fastbatch\_contested0x03→ fastbatch  0x04→ sram5\_有争议0x05→ sram5  0x06→ sram4\_有争议0x07→ sram4  0x08→ sram3\_有争议0x09→ sram3  0x0a→ sram2\_有争议的0x0b→ sram2  0x0c→ sram1\_contested 0x0d→ sram1  0x0e→ sram0\_contested 0x0f→ sram0  0x10→xip\_main\_contested0x11→ xip\_main  0x12→ rom\_contested 0x13→ rom | RW | 0x1f |

### [BUSCTRL](#_bookmark21)：PERFCTR 1寄存器

**偏移**：0x10

描述

总线结构性能计数器1

*表9. 性能1*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31：24 | Reserved. | - | - |
| 23：0 | 总线结构饱和性能计数器1  对来自总线结构仲裁器的一些事件信号进行计数。  写入任何要清除的值使用PERFSEL 1选择要计数的事件 | WC | 0x000000 |

寄存器

表10. PERFSEL1

寄存器

### [BUSCTRL](#_bookmark21)：PERFSEL 1寄存器

**偏移**：0x14

描述

PERFCTR 1的总线结构性能事件选择

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:5 | Reserved. | - | - |
| 4:0 | 选择PERFCTR 1的事件对主交叉开关的下游端口上的竞争访问或所有访问进行  0x00→ apb\_contested 0x01→ apb  0x02→fastbatch\_contested0x03→ fastbatch  0x04→ sram5\_有争议0x05→ sram5  0x06→ sram4\_有争议0x07→ sram4  0x08→ sram3\_有争议0x09→ sram3  0x0a→ sram2\_有争议的0x0b→ sram2  0x0c→ sram1\_contested 0x0d→ sram1  0x0e→ sram0\_contested 0x0f→ sram0  0x10→xip\_main\_contested0x11→ xip\_main  0x12→ rom\_contested 0x13→ rom | RW | 0x1f |

### [BUSCTRL](#_bookmark21)：PERFCTR 2寄存器

**偏移**：0x18

描述

总线结构性能计数器2

*表11. PERFCTR2*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - |
| 23:0 | 总线结构饱和性能计数器2  对来自总线结构仲裁器的一些事件信号进行计数。  写入任何要清除的值使用PERFSEL 2选择要计数的事件 | WC | 0x000000 |

寄存器

表12. PERFSEL2

寄存器

### [BUSCTRL](#_bookmark21)：PERFSEL 2寄存器

**偏移**：0x1c

描述

PERFCTR 2的总线结构性能事件选择

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:5 | Reserved. | - | - |
| 4:0 | 选择PERFCTR 2的事件对主交叉开关的下游端口上的竞争访问或所有访问进行  0x 00→ apb\_contested 0x 01→ apb  0x02→fastbatch\_contested0x03→ fastbatch  0x04→ sram5\_有争议0x05→ sram5  0x06→ sram4\_有争议0x07→ sram4  0x08→ sram3\_有争议0x09→ sram3  0x0a→ sram2\_有争议的0x0b→ sram2  0x0c→ sram1\_contested 0x0d→ sram1  0x0e→ sram0\_contested 0x0f→ sram0  0x10→xip\_main\_contested0x11→ xip\_main  0x12→ rom\_contested 0x13→ rom | RW | 0x1f |

### [BUSCTRL](#_bookmark21)：PERFCTR 3寄存器

**偏移**：0x20

描述

总线结构性能计数器3

*表13. PERFCTR3*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - |
| 23:0 | 总线结构饱和性能计数器3  对来自总线结构仲裁器的一些事件信号进行计数。  写入任何要清除的值使用PERFSEL 3选择要计数的事件 | WC | 0x000000 |

寄存器

表14. PERFSEL3

寄存器

表15. 地址映射摘要

### [BUSCTRL](#_bookmark21)：PERFSEL 3寄存器

**偏移**：0x24

描述

PERFCTR3的巴士面料表演活动选择

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:5 | Reserved. | - | - |
| 4:0 | 选择PERFCTR 3的事件对主交叉开关的下游端口上的竞争访问或所有访问进行  0x 00→ apb\_contested 0x 01→ apb  0x02→fastbatch\_contested0x03→ fastbatch  0x04→ sram5\_有争议0x05→ sram5  0x06→ sram4\_有争议0x07→ sram4  0x08→ sram3\_有争议0x09→ sram3  0x0a→ sram2\_有争议的0x0b→ sram2  0x0c→ sram1\_contested 0x0d→ sram1  0x0e→ sram0\_contested 0x0f→ sram0  0x10→xip\_main\_contested0x11→ xip\_main  0x12→ rom\_contested 0x13→ rom | RW | 0x1f |

* 1. 地址映射

设备的地址映射分为几个部分，如[表15](#_bookmark34)所示。详情见以下各节。访问未映射的地址范围时会引发总线错误。

### 总结

|  |  |
| --- | --- |
| [ROM](#_bookmark36) | 0x00000000 |
| [XIP](#_bookmark37) | 0x10000000 |
| [SRAM](#_bookmark38) | 0x20000000 |
| [APB外围设备](#_bookmark39) | 0x40000000 |

|  |  |
| --- | --- |
| [AHB-Lite外设](#_bookmark40) | 0x50000000 |
| [IOPORT寄存器](#_bookmark41) | 0xd0000000 |
| [Cortex-M0+内部寄存器](#_bookmark42) | 0xe0000000 |

* + 1. 详细

罗：

|  |  |
| --- | --- |
| [ROM\_BASE](#_bookmark210) | 0x00000000 |

XIP：

|  |  |
| --- | --- |
| [XIP\_BASE](#_bookmark213) | 0x10000000 |
| XIP\_NOALLOC\_BASE | 0x11000000 |
| XIP\_NOCACHE\_BASE | 0x12000000 |
| XIP\_NOCACHE\_NOALLOC\_BASE | 0x13000000 |
| XIP\_CTRL\_BASE | 0x14000000 |
| XIP\_SRAM\_BASE | 0x15000000 |
| XIP\_SRAM\_END | 0x15004000 |
| XIP\_SSI\_BASE | 0x18000000 |

SRAM。SRAM 0 -3条纹：

|  |  |
| --- | --- |
| [SRAM\_BASE](#_bookmark211) | 0x20000000 |
| SRAM\_STRIPED\_BASE | 0x20000000 |
| SRAM\_STRIPED\_END | 0x20040000 |

SRAM 4-5总是非条带化的：

|  |  |
| --- | --- |
| SRAM 4\_BASE | 0x20040000 |
| SRAM 5\_BASE | 0x20041000 |
| SRAM\_END | 0x20042000 |

SRAM 0 -3的非条带化别名

|  |  |
| --- | --- |
| SRAM0\_BASE | 0x21000000 |
| SRAM 1\_BASE | 0x21010000 |
| SRAM 2\_BASE | 0x21020000 |
| SRAM 3\_BASE | 0x21030000 |

APB外设：

|  |  |
| --- | --- |
| SYSINFO\_Base | 0x40000000 |
| SYSCFG\_BASE | 0x40004000 |
| [时钟\_底座](#_bookmark310) | 0x40008000 |

|  |  |
| --- | --- |
| [重置\_基础](#_bookmark302) | 0x4000c000 |
| [PSM\_基础](#_bookmark291) | 0x40010000 |
| IO\_BANK0\_BASE | 0x40014000 |
| IO\_QSPI\_BASE | 0x40018000 |
| PADS\_BANK0\_BASE | 0x4001c000 |
| PADS\_QSPI\_基础 | 0x40020000 |
| XOSC\_基础 | 0x40024000 |
| PLL\_PLL\_BASE | 0x40028000 |
| PLL\_USB\_BASE | 0x4002c000 |
| [总线控制\_底座](#_bookmark13) | 0x40030000 |
| UART0\_BASE | 0x40034000 |
| UART1\_BASE | 0x40038000 |
| SPI0\_BASE | 0x4003c000 |
| SPI1\_BASE | 0x40040000 |
| I2C0\_BASE | 0x40044000 |
| I2C1\_BASE | 0x40048000 |
| ADC\_BASE | 0x4004c000 |
| PWM\_BASE | 0x40050000 |
| 定时器底座 | 0x40054000 |
| 看门狗基地 | 0x40058000 |
| RTC\_BASE | 0x4005c000 |
| ROSC\_基础 | 0x40060000 |
| [VREG\_AND\_CHIP\_BASED\_](#_bookmark276) | 0x40064000 |
| 公司简介 | 0x4006c000 |

AHB-Lite外设：

|  |  |
| --- | --- |
| [DMA\_Base](#_bookmark163) | 0x50000000 |

USB在其基础上有一个DPRAM，后面是寄存器：

|  |  |
| --- | --- |
| USBCTRL\_BASE | 0x5010000 |
| USBCTRL\_DPRAM\_BASE | 0x5010000 |
| USBCTRL\_REGS\_BASE | 0x5011000 |

剩余AHB-Lite外围设备：

|  |  |
| --- | --- |
| PIO0\_BASE | 0x50200000 |
| PIO1\_BASE | 0x50300000 |
| XIP\_AUX\_BASE | 0x50400000 |

IOPORT外设：

|  |  |
| --- | --- |
| [SIO\_BASE](#_bookmark44) | 0xd0000000 |

Cortex-M0+内部外设：

|  |  |
| --- | --- |
| [PPB\_基础](#_bookmark115) | 0xe0000000 |

* 1. 处理器子系统

RP 2040处理器子系统由两个Arm Cortex-M0+处理器组成，每个处理器都有标准的内部Arm CPU外设，以及用于GPIO访问和核间通信的外设。有关Arm Cortex-M0+处理器的详细信息，包括RP 2040上使用的特定功能配置，请参见[第2.4](#_bookmark115)。

从外围设备

从外部调试器

中断

串行线调试

事件

IOPORT

SiO

IOPORT

AHB-精简版

GPIO×36

AHB-精简版

到总线织物

至GPIO复用

到总线织物

图6. 两个Cortex-M0+处理器，每个处理器都有一个专用的32位AHB-Lite总线端口，用于代码读取、加载和存储。SIO连接到每个处理器的单周期IOPORT总线，并提供GPIO访问、双向通信和其他核心本地外设。这两个处理器都可以通过一个多点串行总线进行调试。 26个中断（加上NMI）被路由到每个处理器上的NVIC和WIC。

|  |  |
| --- | --- |
| NVIC | DAP |
| 核心0Cortex-M0+ | |
| 总线接口 | |

|  |  |
| --- | --- |
| NVIC | DAP |
| 核心1Cortex-M0+ | |
| 总线接口 | |

##### 注意

术语*core0*和*core1*、*proc0*和*proc1*在RP 2040的寄存器和文档中可互换使用，分别指处理器0和处理器1。

处理器使用许多接口与系统的其余部分通信

* 每个处理器都使用自己独立的32位AHB-Lite总线来访问内存和内存映射外设（详见[第2.1节](#_bookmark13)）
* 单周期IO块通过每个处理器的IOPORT提供对GPIO的高速、确定性访问
* 26个系统级中断路由到两个处理器
* 多点串行线串行总线提供从外部调试主机到两个处理器的调试访问
  + 1. SiO

单周期IO块（SIO）包含几个需要处理器进行低延迟、确定性访问的外设。它通过每个处理器的IOPORT访问：这是Cortex-M0+上的辅助总线端口，可以执行快速的32位读取和写入。SIO为每个处理器的IOPORT提供专用总线接口，如[图7](#_bookmark45)所示。处理器通过正常的加载和存储指令访问它们的IOPORT，这些指令指向特殊的IOPORT地址段0xd000000.0xdfffffff。SIO在IOPORT空间中显示为内存映射硬件。

##### 注意

由于时间要求严格，SIO未连接到主系统总线它只能由处理器访问

*图7.单周期IO块包含处理器必须能够快速访问的内存映射硬件。 FIFO和自旋锁支持两个内核之间的消息传递和同步。 共享GPIO寄存器提供对支持GPIO的引脚的快速且并发安全的直接访问。*



IOPORT

单周期IO

IOPORT

CPUID0

CPUID1

FIFO 0到1

FIFO 1到0

总线

接口

硬件Spinlock×32

总线

接口

分隔器

插值器0

插值器0

插值器1

插值器1

GPIO寄存器共享，原子设置/清除/异或

GPIO×36

至GPIO复用

芯1

核心0

一些核心本地算术硬件可以用于加速处理器上的常见任务。

所有IOPORT读取和写入（因此所有SIO访问）都在一个周期内发生，这与主AHB-Lite系统总线不同，Cortex-M0+需要两个周期进行加载或存储，并且可能由于其他系统总线主机的竞争而必须等待更长时间。这对于GPIO等具有严格时序要求的接口至关重要。

SIO寄存器映射到0xd000000.0xd000017c范围内的字对齐地址。IOPORT空间的剩余部分保留供将来使用。

SIO外设将在以下章节中详细介绍

* + - 1. CPUID

寄存器[CPUID](#_bookmark48)是IOPORT空间中的第一个寄存器核心0在访问该地址时读取值0，核心1读取值1。这是一种方便的方法，软件可以确定它正在哪个内核上运行。这在初始引导序列期间进行检查：两个核心同时开始运行，核心1进入深度睡眠状态，核心0继续主引导序列。

**重要**

[CPUID](#_bookmark48)不应与每个处理器内部专用外设总线上的Cortex-M0+ CPUID寄存器（[第2.4.4.1.1节](#_bookmark125)）混淆，后者列出了处理器的部件号和版本。

* + - 1. GPIO控制

处理器可以访问GPIO寄存器，以便快速直接控制具有GPIO功能的引脚有两套相同的寄存器：

* GPIO\_x用于直接控制IO bank 0（用户GPIO 0至29，从LSB开始
* GPIO\_HI\_x用于直接控制QSPI IO库（顺序为SCLK、SSn、SD 0、SD 1、SD 2、SD 3，从LSB开始

##### 注意

要使用SIO的GPIO寄存器驱动某个引脚，必须首先将该引脚的GPIO多路复用器配置为选择SIO GPIO功能。见表279。

这些GPIO寄存器在两个内核之间*共享*，两个内核可以同时访问它们。每个存储体有三个寄存器：

* 输出寄存器[GPIO\_OUT](#_bookmark51)和[GPIO\_HI\_OUT](#_bookmark59)用于设置GPIO的输出电平（1/0表示高/低）
* 输出使能寄存器[GPIO\_OE](#_bookmark55)和[GPIO\_HI\_OE](#_bookmark63)用于使能输出驱动器。0表示高阻抗，1表示基于[GPIO\_OUT](#_bookmark51)和[GPIO\_HI\_OUT的](#_bookmark59)驱动高/低电平。
* 输入寄存器[GPIO\_IN](#_bookmark49)和[GPIO\_HI\_IN](#_bookmark50)允许处理器采样GPIO的当前状态

读取[GPIO\_IN](#_bookmark49)可在单次读取中返回所有30个GPIO值（[GPIO\_HI\_IN](#_bookmark50)为6个）然后，软件可以屏蔽它感兴趣的各个引脚。

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_gpio/include/hardware/gpio.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_gpio/include/hardware/gpio.h#L674-L676)第674-676

public static inlinebool gpio\_get（uint gpio）{

675 return！！（（1ulgpio）sio\_hw->gpio\_in）;

676}

OUT和OE寄存器还具有原子SET、XOR和XOR别名，允许软件在一次操作中更新引脚子集。这不仅对于两个内核之间的安全并行GPIO访问至关重要，而且对于在一个内核上运行的中断处理程序和前台代码中的安全并发GPIO访问也至关重要

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_gpio/include/hardware/gpio.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_gpio/include/hardware/gpio.h#L696-L698)第696-698

public static inline voidgpio\_set\_mask（uint32\_tmask）{

697sio\_hw->gpio\_set =mask;

698}

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_gpio/include/hardware/gpio.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_gpio/include/hardware/gpio.h#L705-L707)第705-707

public static voidgpio\_clr\_mask（uint32\_tmask）{

706sio\_hw->gpio\_clr =mask;

707}

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_gpio/include/hardware/gpio.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_gpio/include/hardware/gpio.h#L748-L754)第748-754

1. public int findDuplicate（int findDuplicate，int findDuplicate）{
2. uint32\_tmask =1ul<<gpio;
3. if（value）
4. gpio\_set\_mask（mask）;
5. 其他
6. gpio\_clr\_mask（mask）;

754}

如果两个处理器在同一个时钟周期写入OUT或OE寄存器（或其任何SET/XOR别名），则结果就像核心0先写入，核心1紧接着写入。例如，如果核心0设置了一个位，而核心1同时对它进行了异或，则该位将被设置为0，而不管它的原始值如何。

**注意**

这是两个内核同时写入GPIO寄存器时产生的结果的概念模型寄存器实际上在任何时候都不包含这个中间值。在前面的示例中，如果引脚初始为0，并且内核0执行SET，而内核1执行XOR，则GPIO输出保持低电平，没有任何正毛刺。

* + - 1. 硬件自旋锁

SIO提供32个硬件自旋锁，可用于管理对共享软件资源的互斥访问。每个自旋锁都是一个1位标志，映射到不同的地址（从[SPINLOCK0](#_bookmark110)到[SPINLOCK31](#_bookmark110)）。软件通过以下操作之一与每个自旋锁

* 读取：尝试声明锁。如果成功声明了锁，则Read值为非零;如果前一次读取已经声明了锁，则Read值为零
* 写（任何值）：释放锁。下一次请求锁的尝试将成功。如果两个核心都试图在同一个时钟周期上请求相同的锁，核心0就会成功。

通常，软件将通过重复轮询锁位（在锁上“旋转”）来获取锁，直到它被成功声明。如果锁被长时间持有，这是低效的，所以通常自旋锁应该用来保护高级原语的短关键部分，如互斥锁，信号量和队列。

出于调试的目的，可以通过[SPINEARTH\_ST](#_bookmark70)观察所有32个自旋锁的当前状态。

* + - 1. 处理器间FIFO（邮箱）

SIO包含两个FIFO，用于在两个内核之间传递数据、消息或有序事件。每个FIFO为32位宽，8个条目深。其中一个FIFO只能由核心0写入，并由核心1读取另一个只能由核心1写入，由核心0读取

每个内核通过写入[FIFO\_WR](#_bookmark68)来写入其传出FIFO，并通过从[FIFO\_RD](#_bookmark69)读取来从其传入FIFO读取。状态寄存器[FIFO\_ST](#_bookmark67)提供以下状态信号：

* 传入FIFO包含数据（VLD）
* 传出FIFO可容纳更多数据（RDY）
* 在过去的某个时间点，当输入FIFO为空时，从输入FIFO读取（ROE）
* 传出FIFO在过去某个时间点（WOF）满时被写入

当FIFO满时写入输出FIFO，或当FIFO空时从输入FIFO读取，不会影响FIFO状态。保留FIFO的当前内容和级别但是，这确实表示访问FIFO的软件丢失了一些数据或接收到无效数据，因此会产生一个粘性错误标志（ROE或WOF）。

SIO为每个内核提供FIFO IRQ输出，映射到系统IRQ编号15和16。每个IRQ输出都是该内核[FIFO\_ST](#_bookmark67)寄存器中VLD、ROE和WOF位的逻辑或：也就是说，如果这三个位中的任何一位为高电平，则IRQ置位，当它们都为低电平时，IRQ再次清零。通过向[FIFO\_ST](#_bookmark67)写入任意值来清除ROE和WOF标志，通过从FIFO读取数据来清除VLD标志，直到为空。

如果在Cortex-M0+NVIC中使能了相应的中断线，则处理器将在每次数据出现在其FIFO中时，或者如果它执行了某些无效的FIFO操作（空时读取，满时写入）时进行中断。核心0使用IRQ 15，核心1使用IRQ 16。如果以相反的方式使用IRQ，那么被中断的核心就很难正确识别中断的原因，因为该核心无法访问其他核心的FIFO状态寄存器。

##### 注意

ROE和WOF只有在软件以某种方式行为不当时才被设置通常，当数据出现在FIFO中时（提升VLD标志），中断处理程序将触发，并且中断处理程序通过从FIFO中读取数据来清除IRQ，直到VLD再次变为低电平。

处理器间FIFO和Cortex-M0+事件信号由bootrom（[第2.8节](#_bookmark224)）wait\_for\_vector例程使用，其中核心1保持在睡眠状态，直到其被唤醒，并通过FIFO提供其初始堆栈指针、入口点和

* + - 1. 整数分频器

SIO为每个内核提供一个8周期有符号/无符号除/模模块。计算开始时，将被除数和除数写入两个参数寄存器DIVIDEND和DIVISOR。除法器在接下来的8个周期内计算该除法的商/和余数%，在第9个周期，可以从两个结果寄存器[DIV\_QUOTIENT](#_bookmark75)和[DIV\_REMAINDER](#_bookmark76)读取结果。可以轮询寄存器[DIV\_CSR](#_bookmark77)中的“就绪”位以等待计算完成，或者软件可以插入固定的8周期延迟。

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_divider/divider.S](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_divider/divider.S#L8-L12)第8-12

1. regular\_func\_with\_section hw\_divider\_divmod\_s32
2. ldr r3，=（SIO\_BASE）
3. 字符串r0，[r3，#SIO\_DIV\_SDIVIDEND\_OFFSET]
4. 字符串r1，[r3，#SIO\_DIV\_SDIVISOR\_OFFSET]
5. bhw\_divider\_divmod\_return

##### 注意

在这8个周期内，软件可自由执行其他非分频器操作

操作数寄存器有两个别名：写入有符号别名（[DIV\_SDIVIDEND](#_bookmark73)和[DIV\_SDIVISOR](#_bookmark74)）将启动有符号计算，写入另一个别名（[DIV\_UDIVIDEND](#_bookmark71)和[DIV\_UDIVISOR](#_bookmark72)）将启动无符号计算。

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_divider/divider.S](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_divider/divider.S#L16-L20)第16-20

1. regular\_func\_with\_section hw\_divider\_divmod\_u32
2. ldr r3，=（SIO\_BASE）
3. 字符串r0，[r3，#SIO\_DIV\_UDIVIDEND\_OFFSET]
4. 字符串r1，[r3，#SIO\_DIV\_UDIVISOR\_OFFSET]
5. bhw\_divider\_divmod\_return

##### 注意

*每次*写入操作数寄存器时，新的计算立即开始，而新的操作数写入会立即压缩当前正在进行的任何计算例如，当用同一个除数除多个数字时，只需写入xDIVISOR，每次计算的带符号性由是否写入SDIVIDEND或UDIVIDEND决定

为了支持在中断处理程序进入/退出时（或在RTOS上下文切换），结果寄存器也是可写的。写入结果寄存器将取消当时正在进行的任何操作[DIV\_CSR](#_bookmark77).DIRTY标志有助于提高保存/恢复效率：该标志在写入*任何*分频器寄存器（操作数或结果）时置位，在读取商时清除。

##### 注意

启用时，默认分频器AEABI支持将C电平/和%映射到硬件分频器。使用SDK构建软件并直接使用除法器时，最*后*读取商寄存器非常重要。这可确保使用分频器的任何中断代码都能正确保存和恢复部分分频器状态无论是否需要该值，都应读取商寄存器

SDK模块pico\_divider<https://github.com/raspberrypi/pico-sdk/blob/master/src/common/pico_divider/include/pico/divider.h>提供*了AEABI*实现，用于为32位和64位整数除法挂接C/和%运算符，以及一些同时返回整数和整数的所有这些函数都能正确地保存和恢复硬件分频器状态（脏时），以便它们可以在用户或IRQ处理程序代码中使用

SDK模块hardware\_divider<https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_divider/include/hardware/divider.h>提供了用于访问hardware\_divider的低级宏和帮助函数，但这些不会保存和恢复硬件分频器状态（尽管此标头确实提供了单独的函数来执行此操作）。

* + - 1. 插值器

每个内核都配备了两个*插值器*（INTERP0和INTERP1），可以通过将某些预配置操作组合到单个处理器周期中来加速任务针对预先配置的操作重复多次的情况，这会导致代码在代码的时间关键部分使用更少的CPU周期和更少的CPU寄存器

插值器用于加速SDK中的音频操作，但其灵活的配置使其能够优化许多其他任务，如量化和抖动，表查找地址生成，仿射纹理映射，解压缩和线性反馈。

图8. 一个... 两个累加器寄存器和三个基址寄存器可以从处理器进行单周期读/写访问。该符号被组织成两个通道，其在两个符号上执行掩蔽、移位和符号扩展 这通过将中间移位/掩码值添加到三个基址寄存器来产生三种可能的结果。从左到右，每个通道上的多路复用器由CTRL寄存器中的以下标志控制：CROSS\_RESULT、CROSS\_INPUT、SIGNED、ADD\_RAW。

基地0

结果0 0

0

1

1

累加器0右移掩码符号-从掩码

+

结果0

11 1 0 0

蓄能器1

+ 结果2

累加器0

结果0 1

1

0

0

累加器1右移掩码符号-从掩码

+

结果1

结果1 0 0 1 1

基座1

基座2

处理器可以在一个周期内写入或读取任何寄存器，结果在下一个周期准备就绪处理器还可以通过写入相应的ACCUMx\_ADD寄存器，对两个寄存器ACCUM0或ACCUM1之一执行加法

三个结果在只读位置PEEK 0、PEEK 1、PEEK 2中可用。从这些位置读取数据不会更改磁盘的状态。结果还在位置POP 0、POP 1、POP 2处有别名;从POPx别名读取将返回与相应PEEKx相同的结果，并同时将通道结果写回至EEPROM。这可以用于在每次读取结果时提升缓存的状态

此外，该函数还支持两个值之间的简单分数混合以及箝位值，使它们位于给定范围内

下面的示例显示了一*个弹出*通道结果以生成简单迭代反馈的简单示例

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\_interp/hello\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c#L11-L23)第11-23

publicvoid println（）{

12

13

14

15

16

17

18

19

20

21

22

23}

puts（“9次表：“）;

*//在此核心上的interp 0上初始化通道0*interp\_config cfg = interp\_default\_config（）;interp\_set\_config（interp 0，0，cfg）;

int [0]=0;

int [0]=0;

for（ inti =0; i10; ++i）printf（“%d\n”，interp0->pop[0]）;

##### 注意

完全巧合的是，插值器非常适合SNES MODE7风格的图形例程。例如，在每个核上，INTERP0可以为某个仿射变换提供瓦片查找流，INTERP1可以为同一变换提供到瓦片中的偏移。

* + - * 1. 车道运营

图9. 每个内插器的每个通道都可以配置为对其中一个累加器执行屏蔽、移位和符号扩展。 这被馈送到产生最终结果的加法器中，该最终结果可以可选地与每次读取一起被反馈到加法器中。数据路径可以使用几个32位多路复用器进行配置。 从左到右，这些由以下CTRL标志控制：CROSS\_HRESULT、CROSS\_INPUT、SIGNED、ADD\_RAW。

结果0 0

0

1

累加器0 右移 掩码

结果1 1

1

符号扩展

从面具

1

0

0

添加到BASE1

(forPEEK 0/POP 0）

蓄能器1

添加到BASE2（构成

PEEK 2/POP 2）

每个通道依次执行以下三个操作

* 右移CTRL\_LANEx\_NULL（0至31位）
* 从CTRL\_LANEx\_MASK\_LSB到CTRL\_LANEx\_MASK\_MSB的位掩码（包括0位和31位）
* 从掩模顶部延伸的符号，即取位CTRL\_LANEx\_MASK\_MSB并将其OR为所有更高有效位，如果

设置CTRL\_LANEx\_SIGNED

* ACCUM0=0x死牛肉
* CTRL\_LANE0\_CN=8
* CTRL\_LANE0\_MASK\_LSB=4
* CTRL\_LANE0\_MASK\_MSB=7
* CTRL\_SIGNED=1

然后泳道0将在每个阶段产生以下结果

* 右移8以产生0x00deadbe
* 屏蔽位7至4以产生0x00deadbe 0x000000f0=0x000000b0
* 符号-从位7向上扩展以生成0xffffffb 0

在软件方面：

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\_interp/hello\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c#L25-L46)第25-46

publicvoid println（）{

26

27

28

29

30

31

32

33

34

35

interp\_config cfg =interp\_default\_config（）;

interp0->accum[0]=0x1234abcd;

36

37

38

39

40

41

42

43

44

45

puts（“）;

printf（“ACCUM0 =%08x\n”，interp0->accum[0]）;for（inti =0; i8; ++i）{

//LSB，然后是MSB。这些是包含性的，因此0，31表示“整个32位寄存器”。

interp\_config\_set\_mask（cfg，i \*4，i \*4+3）;

interp\_set\_config（interp 0，0，cfg）;

//从ACCUMX\_ADD读取返回原始通道移位和掩码值，不带BASEX

添加

printf（“半字节%d：%08x\n”，i，interp 0->add\_raw[0]）;

}

puts（“带符号扩展的掩蔽：”）;interp\_config\_set\_signed（cfg，true）;for（inti =0; i8; ++i）{

interp\_config\_set\_mask（cfg，i \*4，i \*4+3）;

interp\_set\_config（interp 0，0，cfg）;

printf（“半字节%d：%08x\n”，i，interp 0->add\_raw[0]）;

}

46}

上面的例子应该打印出来：

ACCUM0 =1234abcd

半字节0：000000d

半字节1：000000c0

Nibble 2：00000b00

Nibble 3：0000a000

Nibble 4：00040000

Nibble 5：00300000

Nibble 6：02000000

Nibble 7：10000000

掩码与符号扩展：半字节0：fffffffd

半字节1：ffffffc0

蚕食2：fffffb00

Nibble 3：ffffa000

Nibble 4：00040000

Nibble 5：00300000

Nibble 6：02000000

Nibble 7：10000000

改变结果和输入多路复用器可以在ADC之间产生反馈这是有用的，例如。 用于音频抖动。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\_interp/hello\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c#L48-L66)第48-66

48void cross\_lanes（）{

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66}

interp\_config cfg = interp\_default\_config（）; interp\_config\_set\_cross\_result（cfg，true）;

//ACCUM 0获取通道1的结果：

interp\_set\_config（interp 0，0，cfg）;

//ACCUM 1获取通道0结果：

interp\_set\_config（interp 0，1，cfg）;

int [0]=123;

int [1]=456;

int [0]=1;

inti =0; int n=0; int n = 0;

uint32\_tpeek0 = interp0->peek[0]; uint32\_tpop1 = interp0->pop[1];

printf（“PEEK 0，POP 1：%d，%d\n”，peek 0，pop1）;

}

这应该打印：

|  |  |  |  |
| --- | --- | --- | --- |
| PEEK 0， | POP1： | 124, | 456 |
| PEEK 0， | POP1： | 457, | 124 |
| PEEK 0， | POP1： | 125, | 457 |
| PEEK 0， | POP1： | 458, | 125 |
| PEEK 0， | POP1： | 126, | 458 |
| PEEK 0， | POP1： | 459, | 126 |
| PEEK 0， | POP1： | 127, | 459 |
| PEEK 0， | POP1： | 460, | 127 |
| PEEK 0， | POP1： | 128, | 460 |
| PEEK 0， | POP1： | 461, | 128 |

* + - * 1. 混合模式

混合模式在每个内核的INTERP0上可用，并通过CTRL\_LANE0\_BLEND控制标志启用它执行线性插值，我们定义如下：



其中，是寄存器BASE0，是寄存器BASE1，是由通道1移位和掩码值的最低8位构成的分数值。

混合模式与普通模式有以下

* + PEEK0、POP0返回8位alpha值（通道1移位和掩码值的8个LSB），结果位31至24为零
  + PEEK1、POP1返回BASE0和BASE1之间的线性插值
  + PEEK2、POP2不包括泳道1导致添加（即，它是BASE2+通道0移位和掩码值）

当alpha值为0时，线性插值的结果等于BASE 0，并且等于BASE 0+255/256 \*（BASE 1-

BASE0），当alpha值为全1时。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\_interp/hello\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c#L68-L87)第68-87

publicvoid println（）{

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

八七}

puts（“简单混合1：“）;

interp\_config cfg = interp\_default\_config（）; interp\_config\_set\_blend（cfg，true）; interp\_set\_config（interp 0，0，cfg）;

cfg = interp\_default\_config（）; interp\_set\_config（interp 0，1，cfg）;

int [0]=500;

int [1]=1000;

public intfindDuplicate（ inti =0; i =6; i++）{

//将fraction设置为0到255之间的值

interp0->accum[1]=255\* i/6;

//500+（1000- 500）\* i/6;

printf（“%d\n”，（int）interp0->peek[1]）;

}

这应该打印（注意255/256导致998而不是1000）：

500

582

666

748

832

914

998

CTRL\_LANE1\_SIGNED控制BASE0和BASE1是否针对该插值进行符号扩展（需要进行符号扩展，因为插值会产生大小为40位的中间乘积值）。CTRL\_LANE0\_SIGNED继续正常控制PEEK2、POP2中通道0中间结果的符号扩展

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\_interp/hello\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c#L90-L121)第90-121

voidprint\_simple\_blend2\_results（bool is\_signed）{

91

92

93

94

95

96

97

98

99

100

101

102

103

104}

105

//lane 1 signed flag控制基数0/1被视为有符号还是无符号

interp\_config cfg = interp\_default\_config（）; interp\_config\_set\_signed（cfg，is\_signed）; interp\_set\_config（interp 0，1，cfg）;

for（ inti =0; i =6; i++）{ interp0->accum[1]=255\* i/6; if（is\_signed）{

printf（“%d\n”，（int）interp0->peek[1]）;

}否则{

printf（“0x%08x\n”，（uint）interp0->peek[1]）;

}

}

publicvoid println（）{

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121}

puts（“简单混合2：“）;

interp\_config cfg = interp\_default\_config（）; interp\_config\_set\_blend（cfg，true）; interp\_set\_config（interp 0，0，cfg）;

interp0->base[0]=（uint32\_t）-1000; interp0->base[1] =1000;

puts（“签名：”）; print\_simple\_blend2\_results（true）;

puts（“unsigned：“）; print\_simple\_blend2\_results（false）;

这应该打印：

签署：

-一千

-672

-三百三十六

-八个

328

656

992

无符号：0xfffffc18 0xd5fffd60 0xaafffeb0 0x80fffff8 0x56000148

0x2c000290

0x010003e0

最后，在混合模式下，当使用BASE\_1AND0寄存器通过单次32位写入向BASE0和BASE1发送16位值时，写入期间这些16位值到完整32位值的符号扩展由两个基的CTRL\_LANE1\_SIGNED控制，这与非混合模式操作相反，其中CTRL\_LANE0\_SIGNED影响扩展到BASE0，CTRL\_LANE1\_SIGNED影响扩展到BASE1。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\_interp/hello\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c#L124-L145)第124-145

publicvoid println（）{

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145}

puts（“简单混合3：“）;

interp\_config cfg = interp\_default\_config（）; interp\_config\_set\_blend（cfg，true）; interp\_set\_config（interp 0，0，cfg）;

cfg = interp\_default\_config（）; interp\_set\_config（interp 0，1，cfg）;

interp0->accum[1] =128; interp0->base01 =0x30005000;

printf（“0x%08x\n”，（int）interp0->peek[1]）; interp0->base01 =0xe000f000; printf（“0x%08x\n”，（int）interp0->peek[1]）;

interp\_config\_set\_signed（cfg，true）;interp\_set\_config（interp 0，1，cfg）;

int（“0x%08x\n”，（int）int [1]）;int（

这应该打印：

0x00004000

0x0000e800

0xffffe800

* + - * 1. 箝位模式

箝位模式在每个内核的INTERP1上可用，并通过CTRL\_LANE0\_CLAMP控制标志启用在箝位模式下，PEEK 0/POP 0结果是箝位在BASE 0和BASE 1之间的通道值（移位、屏蔽、符号扩展ACCUM 0）。换句话说，如果通道值大于BASE 1，则产生BASE 1的值;如果小于BASE 0，则产生BASE 0的值;否则，该值通过。不进行添加。这些比较的符号由CTRL\_LANE0\_SIGNED标志控制。

除此之外，该按钮的行为与正常模式下相同

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\_interp/hello\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c#L193-L211)第193-211

193void clamp（）{

194

195

196

197

198

199

200

201

202

203

204

205

206

207

puts（“Clamp：“）;

interp\_config cfg = interp\_default\_config（）; interp\_config\_set\_clamp（cfg，true）; interp\_config\_set\_shift（cfg，2）;

//根据符号位的新位置设置掩码。

interp\_config\_set\_mask（cfg，0，29）;

*//...因此，移位的值是正确的符号扩展*interp\_config\_set\_signed（&cfg，true）; interp\_set\_config（interp 1，0，&cfg）;

int [0]=0;

interp1->base[1]=255

public intfindDuplicate（ inti =-1024; i =1024; i +=256）{

|  |  |  |
| --- | --- | --- |
| 208 |  | int [0] i; |
| 209 |  | printf（“%d\t%d\n”，i，（int）interp1->peek[0]）; |
| 210 | } |  |
| 211} |  |  |

这应该打印：

|  |  |
| --- | --- |
| -1024 | 0 |
| -七六八 | 0 |
| -512 | 0 |
| -256 | 0 |
| 0 | 0 |
| 256 | 64 |
| 512 | 128 |
| 768 | 192 |
| 1024 | 255 |

* + - * 1. 示例用例：线性插值

线性插值是将混合模式与其他插值函数结合使用的更完整示例：

在此示例中，ACCUM0用于跟踪要插值的值列表内的固定点（整数/分数）位置通道0用于为该位置的整数部分生成值数组中的地址移动位置的小数部分以产生混合的0-255之间的值在数组中的两个连续值之间执行混合

最后，通过对ACCUM0\_ADD\_RAW的单次写入来更新小数位置。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\_interp/hello\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c#L147-L191)第147-191

publicvoid println（）{

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

puts（“线性插值：“）;

const intuv\_fractional\_bits =12;

//对于通道0

//shift和mask函数的值FFFF FFFF（accum0）

//到

0000 0000 000X无尾龙XXX0

*//使用方法 非小数部分乘以2（对于uint16\_t）*interp\_config cfg = interp\_default\_config（）; interp\_config\_set\_shift（&cfg，uv\_fractional\_bits -1）; interp\_config\_set\_mask（&cfg，1，32-uv\_fractional\_bits）; interp\_config\_set\_blend（&cfg，true）;

interp\_set\_config（interp 0，0，cfg）;

//对于通道1

// shift切换切换FFFF FFFF FFFF（accum 0 via crossinput）

//to

0000千炮齐发FFFFFFFF

cfg = interp\_default\_config（）; interp\_config\_set\_shift（cfg，uv\_fractional\_bits-8）; interp\_config\_set\_signed（cfg，true）;

interp\_config\_set\_cross\_input（cfg，true）;*//有符号混合*

interp\_set\_config（interp 0，1，cfg）;&

int 16\_tsamples[]={0，10，-20，-1000，500};

//在我们的分数表示中，步长是1/4

|  |  |  |  |
| --- | --- | --- | --- |
| 178 |  | 为 | public int findDuplicate（inti =0; i= 0; i = 0）{ |
| 179 |  |  | //结果2 =样本+（泳道0原始结果） |
| 180 |  |  | //使用方法 ptr到两个样品中的第一个，以在 |
| 181 |  |  | int 16\_t\*sample\_pair =（ int 16\_t\*）interp0->peek[2]; |
| 182 |  |  | int [0]=int [0]; |
| 183 |  |  | int [1]=int [1]; |
| 184 |  |  | uint32\_tpeek 1 = interp 0->peek[1]; |
| 185 |  |  | uint32\_tadd\_raw1 = interp0->add\_raw[1]; |
| 186 |  |  | printf（“%d\t（%d%% between %d and %d）\n”，（int）peek1， |
| 187 |  |  | 100 \*（add\_raw1 0xff）/0xff，& |
| 188 |  |  | sample\_pair[0]，sample\_pair[1]）; |
| 189 |  |  | interp0->add\_raw[0]=step; |
| 190 |  | } |  |
| 191 | } |  |  |

这应该打印：

174

175

176

177

uint step =（1uv\_fractional\_bits）/4;

interp 0->accum[0]=0;*//初始采样偏移;*

interp0->base[2]=（uintptr\_t）samples;

|  |  |
| --- | --- |
| 0 | （0-10之间为 |
| 2 | (25 0至10之间的百分比 |
| 5 | (50%介于0和10之间） |
| 7 | (75 0至10之间的百分比 |
| 10 | （10-20之间为 |
| 2 | (2510至-20%） |
| -五个 | (5010至-20%） |
| -十三岁 | (7510至-20%） |
| -二十 | （0%在-20和-1000之间） |
| -265 | (25%介于-20和-1000之间） |
| -510 | (50%介于-20和-1000之间） |
| -七五五 | (75%介于-20和-1000之间） |
| -一千 | （-1000至500之间 |
| -625 | (25%在-1000至500之间） |
| -二百五十 | (50%在-1000至500之间） |
| 125 | (75%在-1000至500之间） |

此方法用于SDK中的快速近似音频放大

* + - * 1. 示例用例：简单仿射纹理映射

简单的仿射纹理映射可以通过对纹理坐标使用定点算法，并且对扫描线中的每个像素在每个坐标上步进固定量来实现纹理坐标的整数部分用于在纹理内形成地址以查找像素颜色。

通过使用两个通道、所有三个基值和CTRL\_LANEx\_ADD\_RAW标志，可以将相当昂贵的CPU操作减少到使用CPU的单周期迭代

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/interp/hello\_interp/hello\_interp.c](https://github.com/raspberrypi/pico-examples/blob/master/interp/hello_interp/hello_interp.c#L214-L272)第214-272

1. voidtexture\_mapping\_setup（ uint 8\_t\*texture，uint texture\_width\_bits，uinttexture\_height\_bits，
2. public int findDuplicate（int findDuplicate）{
3. interp\_config cfg =interp\_default\_config（）;
4. *//设置add\_raw标志以在添加时使用原始（未移位和未屏蔽）通道累加器值*
5. //它到车道基地，使车道的结果

interp\_config\_set\_add\_raw（cfg，true）;

interp\_config\_set\_shift（cfg，uv\_fractional\_bits）;

interp\_config\_set\_mask（cfg，0，texture\_width\_bits-1）;

interp\_set\_config（interp 0，0，cfg）;

223

1. interp\_config\_set\_shift（cfg，uv\_fractional\_bits-纹理\_宽度\_bits）;
2. interp\_config\_set\_mask（cfg，texture\_width\_bits，texture\_width\_bits + texture\_height\_bits -1）
3. interp\_set\_config（interp 0，1，cfg）;

227

228interp 0->base[2]=（uintptr\_t）纹理;

229}

230

1. voidtexture\_mapped\_span（ uint8\_t\*output， uint32\_tu， uint32\_tv， uint32\_tdu， uint32\_tdv，uint count）{
2. //u，v是带有uv\_fractional\_bits小数位的定点纹理坐标
3. //du，dv是同一固定点上跨跨度的纹理坐标步长。
4. int [0]=nums;
5. int [0];
6. int [1]=v;
7. int [1]，int [2];
8. public int findDuplicate（int i =0; i++）{
9. //等价于
10. *//uint32\_t sm\_result0 =（uint0>> uv\_fractional\_bits）（1（texture\_width\_bits-1）;*
11. *//uint32\_t sm\_result1 =（uint1>> uv\_fractional\_bits）（1（texture\_height\_bits-1）;*
12. //uint8\_t \*address = texture + sm\_result0+（sm\_result1texture\_width\_bits）;
13. //return [i];
14. //d0 = du +d0;
15. //dv 1 = dv +dv 1;

246

1. //result 2是当前像素的纹理地址
2. //弹出结果前进到下一次迭代
3. int [i]=\*（int\*）int[2];

250 }

251}

252

|  |  |
| --- | --- |
| 253 | public voidrun（）{ |
| 254 | puts（“仿射纹理映射（带纹理包裹）：“）; |
| 256 | uint8\_t纹理[]={ |
| 257 | 0x00、 0x01、 0x02、0x03、 |
| 258 | 0x10、 0x11、 0x12、0x13、 |
| 259 | 0x20、 0x21、 0x22、0x23、 |
| 260 | 0x30、 0x31、 0x32、0x33、 |
| 261 | }; |
| 262 | //4x 4纹理 |
| 263 | texture\_mapping\_setup（texture，2，2，16）; |
| 264 | int [12]; |
| 265 | uint32\_tdu =65536/2;*//步长为1/2* |
| 266 | uint32\_tdv =65536/3;*//步长为1/3* |
| 267 | texture\_mapped\_span（output，0，0，du，dv，12）; |
| 268 |  |
| 269 | public int findDuplicate（int i =0; i12; i++）{ |
| 270 | printf（“0x%02x\n”，output[i]）; |
| 271 | } |
| 272 | } |

255

这应该打印：

0x00

0x00

0x01

0x01

0x12

0x12

0x13

0x23

0x20

0x20

0x31

0x31

* + - 1. 登记册一览表

SIO寄存器从基址0xd0000000（在SDK中定义为[SIO\_BASE](#_bookmark41)）开始。

*表16. SIO寄存器列表*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x000 | [CPUID](#_bookmark48) | 处理器核心标识符 |
| 0x004 | [GPIO\_IN](#_bookmark49) | GPIO引脚的输入值 |
| 0x008 | [GPIO\_HI\_IN](#_bookmark50) | QSPI引脚的输入值 |
| 0x010 | [GPIO\_OUT](#_bookmark51) | GPIO输出值 |
| 0x014 | [GPIO输出设置](#_bookmark52) | GPIO输出值设置 |
| 0x018 | [GPIO输出端口](#_bookmark53) | GPIO输出值清除 |
| 0x01c | [GPIO\_OUT\_XOR](#_bookmark54) | GPIO输出值XOR |
| 0x020 | [GPIO\_OE](#_bookmark55) | GPIO输出使能 |
| 0x024 | [GPIO\_OE\_SET](#_bookmark56) | GPIO输出使能设置 |
| 0x028 | [GPIO\_OE\_接口](#_bookmark57) | GPIO输出使能清除 |
| 0x02c | [GPIO\_OE\_XOR](#_bookmark58) | GPIO输出使能XOR |
| 0x030 | [GPIO\_HI\_OUT](#_bookmark59) | QSPI输出值 |
| 0x034 | [GPIO\_HI\_OUT\_SET](#_bookmark60) | QSPI输出值设置 |
| 0x038 | [GPIO\_HI\_OUT\_UART](#_bookmark61) | QSPI输出值清零 |
| 0x03c | [GPIO\_HI\_OUT\_XOR](#_bookmark62) | QSPI输出值XOR |
| 0x040 | [GPIO\_HI\_OE](#_bookmark63) | QSPI输出使能 |
| 0x044 | [GPIO\_HI\_OE\_SET](#_bookmark64) | QSPI输出使能设置 |
| 0x048 | [GPIO\_HI\_OE\_USB](#_bookmark65) | QSPI输出使能清零 |
| 0x04c | [GPIO\_HI\_OE\_XOR](#_bookmark66) | QSPI输出使能XOR |
| 0x050 | [FIFO\_ST](#_bookmark67) | 内核间FIFO（邮箱）的状态寄存器 |
| 0x054 | [FIFO\_WR](#_bookmark68) | 对该内核的TXFIFO进行写访问 |
| 0x058 | [FIFO\_RD](#_bookmark69) | 读取此内核的RXFIFO |
| 0x05c | [联系我们](#_bookmark70) | 自旋锁态 |
| 0x060 | [DIV\_UDIVIDEND](#_bookmark71) | 无符号被除数 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x064 | [DIV\_UDIVISOR](#_bookmark72) | 无符号除数 |
| 0x068 | [DIV\_SDIVIDEND](#_bookmark73) | 有符号被除数 |
| 0x06c | [DIV\_SDIVISOR](#_bookmark74) | 带符号除数 |
| 0x070 | [DIV\_报价](#_bookmark75) | 除数结果商 |
| 0x074 | [DIV\_剩余](#_bookmark76) | 除法器结果余数 |
| 0x078 | [公司简介](#_bookmark77) | 分频器的控制和状态寄存器 |
| 0x080 | [INTERP0\_ACCUM0](#_bookmark78) | 对累加器0的读/写 |
| 0x084 | [INTERP0\_ACCUM1](#_bookmark79) | 读/写访问累加器1 |
| 0x088 | [INTERP0\_BASE0](#_bookmark80) | 对BASE 0寄存器进行读/写访问。 |
| 0x08c | [INTERP0\_BASE1](#_bookmark81) | 对BASE 1寄存器进行读/写访问。 |
| 0x090 | [INTERP0\_BASE2](#_bookmark82) | 对BASE 2寄存器进行读/写访问。 |
| 0x094 | [INTERP0\_POP\_LANE0](#_bookmark83) | 读取LANE0结果，并同时将通道结果写入两个存储器（POP）。 |
| 0x098 | [INTERP0\_POP\_LANE1](#_bookmark84) | 读取通道1结果，并同时将通道结果写入两个存储器（POP）。 |
| 0x09c | [INTERP0\_POP\_FULL](#_bookmark85) | 读取完整结果，并同时将通道结果写入两个存储器（POP）。 |
| 0x0a0 | [INTERP0\_PEEK\_LANE0](#_bookmark86) | 读取LANE0结果，不改变任何内部状态（PEEK）。 |
| 0x0a4 | [INTERP0\_PEEK\_LANE1](#_bookmark87) | 读取LANE 1结果，不改变任何内部状态（PEEK）。 |
| 0x0a8 | [INTERP0\_PEEK\_FULL](#_bookmark88) | 读取完整结果，不改变任何内部状态（PEEK）。 |
| 0x0ac | [INTERP0\_CTRL\_LANE0](#_bookmark89) | 通道0的控制寄存器 |
| 0x0b0 | [INTERP0\_CTRL\_LANE1](#_bookmark90) | 通道1的控制寄存器 |
| 0x0b4 | [INTERP0\_ACCUM0\_ADD](#_bookmark91) | 此处写入的值以原子方式添加到ACCUM0 |
| 0x0b8 | [INTERP0\_ACCUM1\_ADD](#_bookmark92) | 此处写入的值以原子方式添加到ACCUM1 |
| 0x0bc | [INTERP0\_BASE\_1AND0](#_bookmark93) | 写入时，低16位变为BASE0，高16位同时变为BASE1。 |
| 0x0c0 | [INTERP1\_ACCUM0](#_bookmark94) | 对累加器0进行读/写 |
| 0x0c4 | [INTERP1\_ACCUM1](#_bookmark95) | 对累加器1的读/写 |
| 0x0c8 | [INTERP1\_BASE0](#_bookmark96) | 对BASE 0寄存器进行读/写访问。 |
| 0x0cc | [INTERP1\_BASE1](#_bookmark97) | 对BASE 1寄存器进行读/写访问。 |
| 0x0d0 | [INTERP1\_BASE2](#_bookmark98) | 对BASE 2寄存器进行读/写访问。 |
| 0x0d4 | [INTERP1\_POP\_LANE0](#_bookmark99) | 读取LANE0结果，并同时将通道结果写入两个存储器（POP）。 |
| 0x0d8 | [INTERP1\_POP\_LANE1](#_bookmark100) | 读取通道1结果，并同时将通道结果写入两个存储器（POP）。 |
| 0x0dc | [INTERP1\_POP\_FULL](#_bookmark101) | 读取完整结果，并同时将通道结果写入两个存储器（POP）。 |
| 0x0e0 | [INTERP1\_PEEK\_LANE0](#_bookmark102) | 读取LANE0结果，不改变任何内部状态（PEEK）。 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x0e4 | [INTERP1\_PEEK\_LANE1](#_bookmark103) | 读取LANE 1结果，不改变任何内部状态（PEEK）。 |
| 0x0e8 | [INTERP1\_PEEK\_FULL](#_bookmark104) | 读取完整结果，不改变任何内部状态（PEEK）。 |
| 0x0ec | [INTERP1\_CTRL\_LANE0](#_bookmark105) | 通道0的控制寄存器 |
| 0x0f0 | [INTERP1\_CTRL\_LANE1](#_bookmark106) | 通道1的控制寄存器 |
| 0x0f4 | [INTERP1\_ACCUM0\_ADD](#_bookmark107) | 此处写入的值以原子方式添加到ACCUM0 |
| 0x0f8 | [INTERP1\_ACCUM1\_ADD](#_bookmark108) | 此处写入的值以原子方式添加到ACCUM1 |
| 0x0fc | [INTERP1\_BASE\_1AND0](#_bookmark109) | 写入时，低16位变为BASE 0，高16位同时变为BASE 1。 |
| 0x100 | [SPINLOCK0](#_bookmark110) | 自旋锁寄存器0 |
| 0x104 | [SPINLOCK 1](#_bookmark110) | 自旋锁寄存器1 |
| 0x108 | [SPINLOCK 2](#_bookmark110) | 自旋锁寄存器2 |
| 0x10c | [SPINLOCK 3](#_bookmark110) | 自旋锁寄存器3 |
| 0x110 | [旋转锁4](#_bookmark110) | 自旋锁寄存器4 |
| 0x114 | [SPINLOCK 5](#_bookmark110) | 自旋锁寄存器5 |
| 0x118 | [SPINLOCK6](#_bookmark110) | 自旋锁寄存器6 |
| 0x11c | [SPINLOCK 7](#_bookmark110) | 自旋锁寄存器7 |
| 0x120 | [SPINLOCK 8](#_bookmark110) | 自旋锁寄存器8 |
| 0x124 | [SPINLOCK 9](#_bookmark110) | 自旋锁寄存器9 |
| 0x128 | [旋转锁10](#_bookmark110) | 自旋锁寄存器10 |
| 0x12c | [SPINLOCK11](#_bookmark110) | 自旋锁寄存器11 |
| 0x130 | [SPINLOCK 12](#_bookmark110) | 自旋锁寄存器12 |
| 0x134 | [SPINLOCK 13](#_bookmark110) | 自旋锁寄存器13 |
| 0x138 | [SPINLOCK 14](#_bookmark110) | 自旋锁寄存器14 |
| 0x13c | [SPINLOCK 15](#_bookmark110) | 自旋锁寄存器15 |
| 0x140 | [SPINLOCK 16](#_bookmark110) | 自旋锁寄存器16 |
| 0x144 | [SPINLOCK 17](#_bookmark110) | 自旋锁寄存器17 |
| 0x148 | [SPINLOCK 18](#_bookmark110) | 自旋锁寄存器18 |
| 0x14c | [SPINLOCK19](#_bookmark110) | 自旋锁寄存器19 |
| 0x150 | [SPINLOCK20](#_bookmark110) | 自旋锁寄存器20 |
| 0x154 | [SPINLOCK21](#_bookmark110) | 自旋锁寄存器21 |
| 0x158 | [SPINLOCK22](#_bookmark110) | 自旋锁寄存器22 |
| 0x15c | [SPINLOCK23](#_bookmark110) | 自旋锁寄存器23 |
| 0x160 | [SPINLOCK24](#_bookmark110) | 自旋锁寄存器24 |
| 0x164 | [SPINLOCK25](#_bookmark110) | 自旋锁寄存器25 |
| 0x168 | [SPINLOCK26](#_bookmark110) | 自旋锁寄存器26 |
| 0x16c | [SPINLOCK27](#_bookmark110) | 自旋锁寄存器27 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x170 | [SPINLOCK28](#_bookmark110) | 自旋锁寄存器28 |
| 0x174 | [SPINLOCK29](#_bookmark110) | 自旋锁寄存器29 |
| 0x178 | [SPINLOCK30](#_bookmark110) | 自旋锁寄存器30 |
| 0x17c | [SPINLOCK31](#_bookmark110) | 自旋锁寄存器31 |

表17.CPUID寄存器

表18.GPIO\_IN寄存器

表19.GPIO\_HI\_IN寄存器

表20.GPIO\_OUT寄存器

### [SIO](#_bookmark47)：CPUID寄存器

**偏移**：0x000

描述

处理器核心标识符

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 从处理器核心0读取时，值为0;从处理器核心1读取时，值为1。 | RO | - |

### [SIO](#_bookmark47)：GPIO\_IN寄存器

**偏移量**：0x004

描述

GPIO引脚的输入值

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:半 | Reserved. | - | - |
| 29:0 | GPIO的输入值 | RO | 0x00000000 |

### [SIO](#_bookmark47)：GPIO\_HI\_IN寄存器

**偏移量**：0x008

描述

QSPI引脚的输入值

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:六 | Reserved. | - | - |
| 5:0 | QSPI IO上的输入值顺序为0..5：SCLK、SSn、SD 0、SD 1、SD 2、SD 3 | RO | 0x00 |

### [SIO](#_bookmark47)：GPIO\_OUT寄存器

**偏移**：0x010

描述

GPIO输出值

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

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 29:0 | 设置GPIO 0.29的输出电平（1/0→高/低）  回读给出最后写入的值，而不是引脚的输入值。如果核心0和核心1都同时写入GPIO\_OUT（或写入SET/XOR别名），  结果就好像来自核心0的写入首先发生  然后将来自核心1的写入应用于该中间结果。 | RW | 0x00000000 |

表21.GPIO输出设置

寄存器

表22.GPIO输出端口

寄存器

表23.GPIO\_OUT\_XOR

寄存器

表24.GPIO\_OE寄存器

### [SIO](#_bookmark47)：GPIO\_OUT\_SET寄存器

**偏移量**：0x014

描述

GPIO输出值设置

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:半 | Reserved. | - | - |
| 29:0 | 对GPIO\_OUT执行原子位设置，即GPIO\_OUT|=wdata | WO | 0x00000000 |

### [SIO](#_bookmark47)：GPIO\_OUT\_CLK寄存器

**偏移**：0x018

描述

GPIO输出值清除

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:半 | Reserved. | - | - |
| 29:0 | 对GPIO\_OUT执行原子GPIO\_OUT&=~wdata | WO | 0x00000000 |

### [SIO](#_bookmark47)：GPIO\_OUT\_XOR寄存器

**偏移**：0x01c

描述

GPIO输出值XOR

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:半 | Reserved. | - | - |
| 29:0 | 对GPIO\_OUT执行原子逐位XOR，即GPIO\_OUT^=wdata | WO | 0x00000000 |

### [SIO](#_bookmark47)：GPIO\_OE寄存器

**偏移**：0x020

描述

GPIO输出使能

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

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 29:0 | 设置GPIO 0. 29的输出使能（1/0→输出/输入）。回读给出最后写入的值。  如果核心0和核心1都同时写入GPIO\_OE（或写入SET/XOR别名），  结果就好像来自核心0的写入首先发生  然后将来自核1的写入应用于该中间结果。 | RW | 0x00000000 |

表25.GPIO\_OE\_SET寄存器

表26.GPIO\_OE\_UART寄存器

表27.GPIO\_OE\_XOR

寄存器

表28.GPIO\_HI\_OUT寄存器

### [SIO](#_bookmark47)：GPIO\_OE\_SET寄存器

**偏移量**：0x024

描述

GPIO输出使能设置

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:30 | Reserved. | - | - |
| 29:0 | 对GPIO\_OE执行原子位设置，即GPIO\_OE|=wdata | WO | 0x00000000 |

### [SIO](#_bookmark47)：GPIO\_OE\_UART寄存器

**偏移量**：0x028

描述

GPIO输出使能清除

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:30 | Reserved. | - | - |
| 29:0 | 对GPIO\_OE执行原子位清除，即GPIO\_OE&=~wdata | WO | 0x00000000 |

### [SIO](#_bookmark47)：GPIO\_OE\_XOR寄存器

**偏移**：0x02c

描述

GPIO输出使能XOR

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:30 | Reserved. | - | - |
| 29:0 | 对GPIO\_OE执行原子按位XOR，即GPIO\_OE^=wdata | WO | 0x00000000 |

### [SIO](#_bookmark47)：GPIO\_HI\_OUT寄存器

**偏移**：0x030

描述

QSPI输出值

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:6 | Reserved. | - | - |

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 5:0 | 设置QSPI IO 0. 5的输出电平（1/0→高/低）  回读给出最后写入的值，而不是引脚的输入值。如果核心0和核心1都同时写入GPIO\_HI\_OUT（或写入SET/XOR别名），  结果就好像来自核心0的写入首先发生  然后将来自核1的写入应用于该中间结果。 | RW | 0x00 |

表29.GPIO\_HI\_OUT\_SET

寄存器

表30.GPIO\_HI\_OUT\_UART

寄存器

表31.GPIO\_HI\_OUT\_XOR

寄存器

表32.GPIO\_HI\_OE寄存器

### [SIO](#_bookmark47)：GPIO\_HI\_OUT\_SET寄存器

**偏移量**：0x034

描述

QSPI输出值设置

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:6 | Reserved. | - | - |
| 5:0 | 对GPIO\_HI\_OUT执行原子位设置，即GPIO\_HI\_OUT|=wdata | WO | 0x00 |

### [SIO](#_bookmark47)：GPIO\_HI\_OUT\_CLR寄存器

**偏移量**：0x038

描述

QSPI输出值清零

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:6 | Reserved. | - | - |
| 5:0 | 对GPIO\_HI\_OUT执行原子位清除，即GPIO\_HI\_OUT&=~wdata | WO | 0x00 |

### [SIO](#_bookmark47)：GPIO\_HI\_OUT\_XOR寄存器

**偏移**：0x03c

描述

QSPI输出值XOR

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:6 | Reserved. | - | - |
| 5:0 | 对GPIO\_HI\_OUT执行原子逐位XOR，即GPIO\_HI\_OUT^=wdata | WO | 0x00 |

### [SIO](#_bookmark47)：GPIO\_HI\_OE寄存器

**偏移**：0x040

描述

QSPI输出使能

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:6 | Reserved. | - | - |

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 5:0 | 为QSPI IO 0. 5设置输出使能（1/0→输出/输入）。回读给出最后写入的值。  如果内核0和内核1都同时写入GPIO\_HI\_OE（或写入SET/XOR别名），  结果就好像来自核心0的写入首先发生  然后将来自核1的写入应用于该中间结果。 | RW | 0x00 |

表33.GPIO\_HI\_OE\_SET

寄存器

表34.GPIO\_HI\_OE\_USB

寄存器

表35.GPIO\_HI\_OE\_XOR

寄存器

### [SIO](#_bookmark47)：GPIO\_HI\_OE\_SET寄存器

**偏移量**：0x044

描述

QSPI输出使能设置

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:6 | Reserved. | - | - |
| 5:0 | 对GPIO\_HI\_OE执行原子位设置，即GPIO\_HI\_OE|=wdata | WO | 0x00 |

### [SIO](#_bookmark47)：GPIO\_HI\_OE\_RESET寄存器

**偏移量**：0x048

描述

QSPI输出使能清零

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:6 | Reserved. | - | - |
| 5:0 | 对GPIO\_HI\_OE执行原子位清除，即GPIO\_HI\_OE&=~wdata | WO | 0x00 |

[SIO](#_bookmark47)：GPIO\_HI\_OE\_XOR寄存器

**偏移**：0x04c

描述

QSPI输出使能XOR

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:6 | Reserved. | - | - |
| 5:0 | 对GPIO\_HI\_OE执行原子按位XOR，即GPIO\_HI\_OE^=wdata | WO | 0x00 |

### [SIO](#_bookmark47)：FIFO\_ST寄存器

**偏移量**：0x 050

描述

核心间FIFO（邮箱）的状态寄存器

在核心0→核心1方向上有一个FIFO，在核心1→核心0方向上有一个FIFO。两者都是32位宽和8字深。

核心0可以看到1→ 0 FIFO（RX）的读端和0→ 1 FIFO（TX）的写端核心1可以看到0 → 1 FIFO（RX）的读端和1 → 0 FIFO（TX）的写端。

每个内核的SIO IRQ是其FIFO\_ST寄存器的VLD、WOF和ROE字段的逻辑或。

*表36.FIFO\_ST寄存器*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:4 | Reserved. | - | - | - |
| 3 | Roe | 粘性标志指示RX FIFO在为空时被读取FIFO忽略此读取。 | WC | 0x0 |
| 2 | WOF | 粘性标志，指示TXFIFO已满时写入。FIFO忽略此写入。 | WC | 0x0 |
| 1 | RDY | 如果此内核的TXFIFO未满（即，如果FIFO\_WR准备好接收更多数据） | RO | 0x1 |
| 0 | VLD | 如果该内核的RXFIFO不为空（即，如果FIFO\_RD有效） | RO | 0x0 |

表37.FIFO\_WR寄存器

表38.FIFO\_RD寄存器

表39.联系我们

寄存器

表40.DIV\_UDIVIDEND

寄存器

### [SIO](#_bookmark47)：FIFO\_WR寄存器

**偏移量**：0x054

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 对该内核的TXFIFO进行写访问 | WF | 0x00000000 |

### [SIO](#_bookmark47)：FIFO\_RD寄存器

**偏移量**：0x058

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 读取此内核的RXFIFO | RF | - |

### [SIO](#_bookmark47)：SPINESTO\_ST寄存器

**偏移**：0x05c

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 自旋锁态  包含所有32个自旋锁状态的位图（1=锁定）。主要用于调试。 | RO | 0x00000000 |

### [SIO](#_bookmark47)：DIV\_UDIVIDEND寄存器

**偏移**：0x060

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 无符号被除数  写入除法器的DIVIDEND操作数，即 p/q中的p。  任何操作数写入都会启动新的计算。结果显示在QUOTIENT，REMAINDER中。  UDIVIDEND/SDIVIDEND是同一内部寄存器的别名U别名开始一个  无符号计算，S别名开始有符号计算。 | RW | 0x00000000 |

### [SIO](#_bookmark47)：DIV\_UDIVISOR注册

**偏移量**：0x064

*表41.DIV\_UDIVISOR*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 无符号除数  写入除法器的DIVISOR操作数，即 p/q中的q。  任何操作数写入都会启动新的计算。结果显示在QUOTIENT，REMAINDER中。  UDIVISOR/SDIVISOR是同一个内部寄存器的别名U别名开始一个  无符号计算，S别名开始有符号计算。 | RW | 0x00000000 |

寄存器

表42.DIV\_SDIVIDEND

寄存器

表43.DIV\_SDIVISOR

寄存器

表44.DIV\_报价

寄存器

### [SIO](#_bookmark47)：DIV\_SDIVIDEND寄存器

**偏移量**：0x068

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 有符号被除数  与UDIVIDEND相同，但启动有符号计算，而不是无符号计算。 | RW | 0x00000000 |

### [SIO](#_bookmark47)：DIV\_SDIVISOR寄存器

**偏移**：0x06c

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 带符号除数  与UDIVISOR相同，但启动有符号计算，而不是无符号计算。 | RW | 0x00000000 |

### [SIO](#_bookmark47)：DIV\_QUOTIENT注册

**偏移**：0x070

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 除数结果商  DIVIDEND/ DIVISOR（除法）的结果。CSR\_READY为低电平时，内容未定义  对于有符号计算，当DIVIDEND和DIVISOR的符号不同时，QUOTIENT为负数  可直接写入此寄存器，以用于上下文保存/恢复。这阻止了任何  正在进行的计算，并设置CSR\_READY和CSR\_DIRTY标志。从QUOTIENT读取会清除CSR\_DIRTY标志，因此应按顺序读取结果  如果使用CSR\_DIRTY，则为REMAINDER，QUOTIENT。 | RW | 0x00000000 |

### [SIO](#_bookmark47)：DIV\_REMAINDER注册

**偏移量**：0x074

*表45.DIV\_剩余*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 除法器结果余数  DIVIDEND% DIVISOR（模）的结果。CSR\_READY为低电平时，内容未定义  对于有符号计算，仅当DIVIDEND为负时，REMAINDER为负。  可直接写入此寄存器，以用于上下文保存/恢复。这阻止了任何  正在进行的计算，并设置CSR\_READY和CSR\_DIRTY标志。 | RW | 0x00000000 |

寄存器

表46.DIV\_CSR注册

表47.INTERP0\_ACCUM0

寄存器

表48.INTERP0\_ACCUM1

寄存器

### [SIO](#_bookmark47)：DIV\_CSR注册

**偏移**：0x078

描述

分频器的控制和状态寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:2 | Reserved. | - | - | - |
| 1 | 脏 | 写入任何寄存器时变为1，读取QUOTIENT时返回  软件可以使用此标志使保存/恢复更有效（如果不脏则跳过）。  如果以这种方式使用该标志，建议只读取QUOTIENT，  或REMAINDER，然后QUOTIENT，以防止上下文切换时的数据 | RO | 0x0 |
| 0 | 准备 | 当计算正在进行时读取为0，否则读取为1写入操作数（xDIVIDEND，xDIVISOR）将立即开始新的计算，否  如果一个已经在进行中。  写入结果寄存器将立即终止任何正在进行的计算  并设置READY和DIRTY标志。 | RO | 0x1 |

### [SIO](#_bookmark47)：INTERP0\_ACCUM0寄存器

**偏移**：0x080

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 对累加器0进行读/写 | RW | 0x00000000 |

### [SIO](#_bookmark47)：INTERP0\_ACCUM1寄存器

**偏移量**：0x084

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 对累加器1的读/写 | RW | 0x00000000 |

### [SIO](#_bookmark47)：INTERP0\_BASE0寄存器

**偏移量**：0x088

*表49.INTERP0\_BASE0*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 对BASE 0寄存器进行读/写访问。 | RW | 0x00000000 |

寄存器

表50.INTERP0\_BASE1

寄存器

表51.INTERP0\_BASE2

寄存器

表52.INTERP0\_POP\_LANE0

寄存器

表53.INTERP0\_POP\_LANE1

寄存器

表54.INTERP0\_POP\_FULL

寄存器

表55.INTERP0\_PEEK\_LANE

0注册

### [SIO](#_bookmark47)：INTERP0\_BASE1寄存器

**偏移**：0x08c

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 对BASE 1寄存器进行读/写访问。 | RW | 0x00000000 |

### [SIO](#_bookmark47)：INTERP0\_BASE2寄存器

**偏移**：0x090

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 对BASE 2寄存器进行读/写访问。 | RW | 0x00000000 |

### [SIO](#_bookmark47)：INTERP0\_POP\_LANE0寄存器

**偏移量**：0x094

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 读取LANE0结果，并同时将通道结果写入两个存储器（POP）。 | RO | 0x00000000 |

### [SIO](#_bookmark47)：INTERP0\_POP\_LANE1寄存器

**偏移量**：0x098

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 读取通道1结果，并同时将通道结果写入两个存储器（POP）。 | RO | 0x00000000 |

### [SIO](#_bookmark47)：INTERP0\_POP\_FULL寄存器

**偏移**：0x09c

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 读取完整结果，并同时将通道结果写入两个存储器（POP）。 | RO | 0x00000000 |

### [SIO](#_bookmark47)：INTERP0\_PEEK\_LANE0寄存器

**偏移量**：0x0a0

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 读取LANE0结果，不改变任何内部状态（PEEK）。 | RO | 0x00000000 |

### [SIO](#_bookmark47)：INTERP0\_PEEK\_LANE1寄存器

**偏移量**：0x0a4

*表56.INTERP0\_PEEK\_LANE*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 读取LANE 1结果，不改变任何内部状态（PEEK）。 | RO | 0x00000000 |

1寄存器

表57.INTERP0\_PEEK\_FULL

寄存器

表58.INTERP0\_CTRL\_LANE

0注册

### [SIO](#_bookmark47)：INTERP0\_PEEK\_FULL寄存器

**偏移量**：0x0a8

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 读取完整结果，不改变任何内部状态（PEEK）。 | RO | 0x00000000 |

### [SIO](#_bookmark47)：INTERP0\_CTRL\_LANE0寄存器

**偏移**：0x0ac

描述

通道0的控制寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:26 | Reserved. | - | - | - |
| 25 | OVERF | 如果设置了OVERF0或OVERF1，则置1。 | RO | 0x0 |
| 24 | OVERF1 | 指示ACCUM1中是否设置了任何屏蔽MSB。 | RO | 0x0 |
| 23 | OVERF0 | 指示是否设置了ACCUM 0中的任何屏蔽MSB。 | RO | 0x0 |
| 22 | Reserved. | - | - | - |
| 21 | 共混 | 仅存在于每个内核的INTERP0如果启用BLEND模式   * LANE 1结果是BASE 0和BASE 1之间的线性插值，受控   通过通道1移位和掩码值的8个LSB（介于  0和255/256次）   * LANE0结果未添加BASE0（仅产生通道1移位+掩码值的8个LSB * FULL结果未添加通道1移位+掩码值（BASE2 +通道0移位+掩码）   LANE1 SIGNED标志控制插值是有符号还是无符号。 | RW | 0x0 |
| 20:19 | FORCE\_MSB | 与通道结果的位29：28进行“或”运算，并在总线上提供给  对内部32位数据路径没有影响。方便使用车道生成序列  指向闪存或SRAM的指针 | RW | 0x0 |
| 18 | ADD\_RAW | 如果为1，则LANE0结果绕过掩码+移位这不会影响完整结果。 | RW | 0x0 |
| 17 | 交叉结果 | 如果为1，则在POP上将相反通道的结果输入该通道的累加器。 | RW | 0x0 |
| 16 | 交叉输入 | 如果为1，则将相反通道的累加器馈送到该通道的移位+掩码硬件。  即使设置了ADD\_RAW也会生效（CROSS\_INPUT多路复用器在移位+掩码旁路之前） | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 15 | 签署 | 如果设置了SIGNED，则移位和屏蔽的累加器值将符号扩展为32位  在添加到BASE 0之前，LANE 0 PEEK/POP在由处理器读取时显示为扩展到32位。 | RW | 0x0 |
| 14:10 | MASK\_MSB | 掩码允许传递的最高有效位（包括）  设置MSB LSB可能导致芯片翻转 | RW | 0x00 |
| 9:5 | MASK\_LSB | 掩码允许传递的最低有效位（包括） | RW | 0x00 |
| 4:0 | 移位 | 逻辑右移在屏蔽前应用于累加器 | RW | 0x00 |

表59.INTERP0\_CTRL\_LANE

1寄存器

### [SIO](#_bookmark47)：INTERP0\_CTRL\_LANE 1寄存器

**偏移量**：0x0b0

描述

通道1的控制寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:21 | Reserved. | - | - | - |
| 20:19 | FORCE\_MSB | 与通道结果的位29：28进行“或”运算，并在总线上提供给  对内部32位数据路径没有影响。方便使用车道生成序列  闪存或SRAM中的指针 | RW | 0x0 |
| 18 | ADD\_RAW | 如果为1，则LANE 1结果绕过掩码+移位这不会影响完整结果。 | RW | 0x0 |
| 17 | 交叉结果 | 如果为1，则在POP上将相反通道的结果输入该通道的累加器。 | RW | 0x0 |
| 16 | 交叉输入 | 如果为1，则将相反通道的累加器馈送到该通道的移位+掩码硬件。  即使设置了ADD\_RAW也会生效（CROSS\_INPUT多路复用器在移位+掩码旁路之前） | RW | 0x0 |
| 15 | 签署 | 如果设置了SIGNED，则移位和屏蔽的累加器值将符号扩展为32位  在添加到BASE 1之前，LANE 1 PEEK/POP在由处理器读取时显示为扩展到32位。 | RW | 0x0 |
| 14:10 | MASK\_MSB | 掩码允许传递的最高有效位（包括）  设置MSB LSB可能导致芯片翻转 | RW | 0x00 |
| 9:5 | MASK\_LSB | 掩码允许传递的最低有效位（包括） | RW | 0x00 |
| 4:0 | 移位 | 逻辑右移在屏蔽前应用于累加器 | RW | 0x00 |

### [SIO](#_bookmark47)：INTERP0\_ACCUM0\_ADD寄存器

**偏移量**：0x0b4

*表60.INTERP0\_ACCUM0\_AD*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - |
| 23:0 | 此处写入的值以原子方式添加到ACCUM0  读取产生通道0的原始移位和掩码值（未添加BASE0）。 | RW | 0x000000 |

D寄存器

表61.INTERP0\_ACCUM1\_AD

D寄存器

表62.INTERP0\_BASE\_1AND

0注册

表63.INTERP1\_ACCUM0

寄存器

表64.INTERP1\_ACCUM1

寄存器

表65.INTERP1\_BASE0

寄存器

### [SIO](#_bookmark47)：INTERP0\_ACCUM1\_ADD寄存器

**偏移量**：0x0b8

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - |
| 23:0 | 此处写入的值以原子方式添加到ACCUM1  读取产生通道1的原始移位和掩码值（未添加BASE1）。 | RW | 0x000000 |

### [SIO](#_bookmark47)：INTERP0\_BASE\_1AND0寄存器

**偏移量**：0x0bc

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 写入时，低16位变为BASE 0，高16位同时变为BASE 1如果设置了该通道的SIGNED标志，则每一半都会被符号扩展为32位 | WO | 0x00000000 |

### [SIO](#_bookmark47)：INTERP1\_ACCUM0寄存器

**偏移**：0x0c0

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 对累加器0进行读/写 | RW | 0x00000000 |

### [SIO](#_bookmark47)：INTERP1\_ACCUM1寄存器

**偏移量**：0x0c4

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 对累加器1的读/写 | RW | 0x00000000 |

### [SIO](#_bookmark47)：INTERP1\_BASE0寄存器

**偏移量**：0x0c8

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 对BASE 0寄存器进行读/写访问。 | RW | 0x00000000 |

### [SIO](#_bookmark47)：INTERP1\_BASE1寄存器

**偏移**：0x0cc

*表66.INTERP1\_BASE1*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 对BASE 1寄存器进行读/写访问。 | RW | 0x00000000 |

寄存器

表67.INTERP1\_BASE2

寄存器

表68.INTERP1\_POP\_LANE0

寄存器

表69.关于联合国难民事务高级专员的报告INTERP1\_POP\_LANE1

寄存器

表70.INTERP1\_POP\_FULL

寄存器

表71.INTERP1\_PEEK\_LANE

0注册

表72.INTERP1\_PEEK\_LANE

1寄存器

### [SIO](#_bookmark47)：INTERP1\_BASE2寄存器

**偏移**：0x0d0

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 对BASE 2寄存器进行读/写访问。 | RW | 0x00000000 |

### [SIO](#_bookmark47)：INTERP1\_POP\_LANE0寄存器

**偏移量**：0x0d4

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 读取LANE0结果，并同时将通道结果写入两个存储器（POP）。 | RO | 0x00000000 |

### [SIO](#_bookmark47)：INTERP1\_POP\_LANE1寄存器

**偏移量**：0x0d8

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 读取通道1结果，并同时将通道结果写入两个存储器（POP）。 | RO | 0x00000000 |

### [SIO](#_bookmark47)：INTERP1\_POP\_FULL寄存器

**偏移**：0x0dc

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 读取完整结果，并同时将通道结果写入两个存储器（POP）。 | RO | 0x00000000 |

### [SIO](#_bookmark47)：INTERP1\_PEEK\_LANE0寄存器

**偏移量**：0x0e0

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 读取LANE0结果，不改变任何内部状态（PEEK）。 | RO | 0x00000000 |

### [SIO](#_bookmark47)：INTERP1\_PEEK\_LANE1寄存器

**偏移量**：0x0e4

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 读取LANE 1结果，不改变任何内部状态（PEEK）。 | RO | 0x00000000 |

[SIO](#_bookmark47)：INTERP1\_PEEK\_FULL寄存器

**偏移量**：0x0e8

*表73.INTERP1\_PEEK\_FULL*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 读取完整结果，不改变任何内部状态（PEEK）。 | RO | 0x00000000 |

寄存器

表74.INTERP1\_CTRL\_LANE

0注册

### [SIO](#_bookmark47)：INTERP1\_CTRL\_LANE0寄存器

**偏移**：0x0ec

描述

通道0的控制寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:26 | Reserved. | - | - | - |
| 25 | OVERF | 如果设置了OVERF0或OVERF1，则置1。 | RO | 0x0 |
| 24 | OVERF1 | 指示ACCUM1中是否设置了任何屏蔽MSB。 | RO | 0x0 |
| 23 | OVERF0 | 指示是否设置了ACCUM 0中的任何屏蔽MSB。 | RO | 0x0 |
| 22 | 夹具 | 仅存在于每个内核的INTERP1如果箝位模式启用：   * LANE0结果被移位和掩蔽ACCUM0，由下界箝位   BASE0和BASE1的上界。   * 这些比较的正负号由LANE0\_CTRL\_SIGNED确定 | RW | 0x0 |
| 21 | Reserved. | - | - | - |
| 20:19 | FORCE\_MSB | 与通道结果的位29：28进行“或”运算，并在总线上提供给  对内部32位数据路径没有影响。方便使用车道生成序列  闪存或SRAM中的指针 | RW | 0x0 |
| 18 | ADD\_RAW | 如果为1，则LANE0结果绕过掩码+移位这不会影响完整结果。 | RW | 0x0 |
| 17 | 交叉结果 | 如果为1，则在POP上将相反通道的结果输入该通道的累加器。 | RW | 0x0 |
| 16 | 交叉输入 | 如果为1，则将相反通道的累加器馈送到该通道的移位+掩码硬件。  即使设置了ADD\_RAW也会生效（CROSS\_INPUT多路复用器在移位+掩码旁路之前） | RW | 0x0 |
| 15 | 签署 | 如果设置了SIGNED，则移位和屏蔽的累加器值将符号扩展为32位  在添加到BASE 0之前，LANE 0 PEEK/POP在由处理器读取时显示为扩展到32位。 | RW | 0x0 |
| 14:10 | MASK\_MSB | 掩码允许传递的最高有效位（包括）  设置MSB LSB可能导致芯片翻转 | RW | 0x00 |
| 9:5 | MASK\_LSB | 掩码允许传递的最低有效位（包括） | RW | 0x00 |
| 4:0 | 移位 | 逻辑右移在屏蔽前应用于累加器 | RW | 0x00 |

### 

### SIO：INTERP\_CTRL\_LANE 1寄存器

表75.INTERP1\_CTRL\_LANE

1寄存器

表76.INTERP1\_ACCUM0\_AD

D寄存器

**偏移**：0x0f0

描述

通道1的控制寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:21 | Reserved. | - | - | - |
| 20:19 | FORCE\_MSB | 与通道结果的位29：28进行“或”运算，并在总线上提供给  对内部32位数据路径没有影响。方便使用车道生成序列  闪存或SRAM中的指针 | RW | 0x0 |
| 18 | ADD\_RAW | 如果为1，则LANE 1结果绕过掩码+移位这不会影响完整结果。 | RW | 0x0 |
| 17 | 交叉结果 | 如果为1，则在POP上将相反通道的结果输入该通道的累加器。 | RW | 0x0 |
| 16 | 交叉输入 | 如果为1，则将相反通道的累加器馈送到该通道的移位+掩码硬件。  即使设置了ADD\_RAW也会生效（CROSS\_INPUT多路复用器在移位+掩码旁路之前） | RW | 0x0 |
| 15 | 签署 | 如果设置了SIGNED，则移位和屏蔽的累加器值将符号扩展为32位  在添加到BASE 1之前，LANE 1 PEEK/POP在由处理器读取时显示为扩展到32位。 | RW | 0x0 |
| 14:10 | MASK\_MSB | 掩码允许传递的最高有效位（包括）  设置MSB LSB可能导致芯片翻转 | RW | 0x00 |
| 9:5 | MASK\_LSB | 掩码允许传递的最低有效位（包括） | RW | 0x00 |
| 4:0 | 移位 | 逻辑右移在屏蔽前应用于累加器 | RW | 0x00 |

### [SIO](#_bookmark47)：INTERP1\_ACCUM0\_ADD寄存器

### **偏移量**：0x0f4

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - |
| 23：0 | 此处写入的值以原子方式添加到ACCUM0  读取产生通道0的原始移位和掩码值（未添加BASE0）。 | RW | 0x000000 |

### [SIO](#_bookmark47)：INTERP1\_ACCUM1\_ADD寄存器

**偏移量**：0x0f8

*表77.INTERP1\_ACCUM1\_AD*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31：24 | Reserved. | - | - |
| 23:0 | 此处写入的值以原子方式添加到ACCUM1  读取产生通道1的原始移位和掩码值（未添加BASE1）。 | RW | 0x000000 |

D寄存器

表78.对妇女的歧视INTERP1\_BASE\_1AND

0注册表

表79.联合国难民事务高级专员的报告SPINLOCK 0，SPINLOCK 1，.，SPINLOCK 30，SPINLOCK 31

寄存器

表80.对妇女的歧视中断

### [SIO](#_bookmark47)：INTERP1\_BASE\_1AND0寄存器

**偏移量**：0x0fc

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 写入时，低16位变为BASE 0，高16位同时变为BASE 1如果该通道的SIGNED标志被设置，则每一半被符号扩展为32位 | WO | 0x00000000 |

### [SIO](#_bookmark47)：SPINLOCK 0，SPINLOCK 1，.，SPINLOCK 30，SPINLOCK 31寄存器

**偏移**：0x100、0x104、...、0x178、0x17c

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 从自旋锁地址读取将：   * 如果锁已锁定，则返回0 * 否则返回非零值，同时声明锁   写入（任何值）将释放锁。  如果核心0和核心1试图同时声明同一个锁，核心0获胜。  成功返回的值是0x1<<锁号。 | RW | 0x00000000 |

* + 1. 中断

每个核心都配备了一个标准的ARM嵌套矢量控制器（NVIC），具有32个中断输入。每个NVIC都有相同的中断路由到它，GPIO中断除外：每个库，每个内核都有一个GPIO中断它们是完全独立的，例如。核心0可以被存储体0中的GPIO 0中断，核心1可以被同一存储体中的GPIO 1中断。

在RP 2040上，只有较低的26个IRQ信号连接在NVIC上，IRQ 26到31被连接到零（从不触发）。通过在NVIC ISPR寄存器中写入位26至31，仍然可以强制内核进入相关中断处理程序。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| IRQ | 中断源 | IRQ | 中断源 | IRQ | 中断源 | IRQ | 中断源 | IRQ | 中断源 |
| 0 | 定时器\_IRQ\_0 | 6 | XIP\_IRQ | 12 | DMA\_IRQ\_1 | 18 | SPI0\_IRQ | 24 | I2C1\_IRQ |
| 1 | 定时器\_IRQ\_1 | 7 | PIO0\_IRQ\_0 | 13 | IO\_IRQ\_BANK0 | 19 | SPI1\_IRQ | 25 | RTC\_IRQ |
| 2 | 定时器\_IRQ\_2 | 8 | PIO0\_IRQ\_1 | 14 | IO\_IRQ\_QSPI | 20 | UART0\_IRQ |  |  |
| 3 | 定时器\_IRQ\_3 | 9 | PIO1\_IRQ\_0 | 15 | SIO\_IRQ\_SIO0 | 21 | UART 1\_IRQ |  |  |
| 4 | PWM\_IRQ\_WRAP | 10 | PIO1\_IRQ\_1 | 16 | SIO\_IRQ\_SIO1 | 22 | ADC\_IRQ\_FIFO |  |  |
| 5 | USBCTRL\_IRQ | 11 | DMA\_IRQ\_0 | 17 | 时钟\_IRQ | 23 | I2C0\_IRQ |  |  |

##### 注意

XIP\_IRQ来自构成[XIP](#_bookmark213)块的一部分的SSI它可以用于代码从SRAM而不是闪存运行的配置中。在此配置中，XIP模块可用作普通SSI外设。

硬件支持嵌套中断：低优先级中断可以被高优先级中断（或另一个异常，如HardFault）抢占，一旦高优先级异常完成，低优先级中断将恢复。优先顺序由以下因素决定

* 首先，由NVIC\_IPR0寄存器为每个中断配置的动态优先级。Cortex-M0+实现

8位优先级字段的两个最高有效位，因此有四个优先级可用，数字上最低的级别（级别0）是最高优先级。

* 其次，对于具有相同动态优先级的中断，编号较低的IRQ具有较高的优先级（使用上表中给出的

在安排RP 2040中断表以给出合理的默认优先级排序时，已经进行了一些注意，但是可以使用[NVIC\_IPR0](#_bookmark141)到[NVIC\_IPR7](#_bookmark148)来提高或降低各个中断的优先级，以适应特定的用例。

26个系统IRQ信号被屏蔽（NMI屏蔽），然后一起进行OR运算，为内核创建NMI信号。每个内核的NMI掩码可以使用Syscfg寄存器块中的NMI 0\_NMI\_MASK和NMI 1\_NMI\_MASK进行配置这些寄存器中的每一个对于每个系统中断具有一个位，并且如果断言系统中断**并且**针对该内核设置了对应的NMI掩码位，则

**谨慎**

如果看门狗已装备，并且内核1 NMI掩码上的某些位已设置，则应将RESETS块（以及Syscfg）包括在看门狗复位列表中。否则，在看门狗事件之后，当核心进入引导ROM时，核心1 NMI可以被核心0在进入bootrom时获取NMI是安全的（处理程序将清除NMI掩码）。

* + 1. 事件信号

使用WFE指令，Cortex-M0+可以进入睡眠状态，直到发生“事件”（或中断）它还可以使用SEV指令生成事件在RP2040上，事件信号在两个处理器之间交叉连接，因此一个处理器发送的事件将在另一个处理器上接收

##### 注意

事件标志是“粘性”的，因此如果两个处理器同时发送事件（SEV），然后都进入睡眠（WFE），它们都会立即唤醒，而不是陷入睡眠状态。

当处于WFE（或WFI）睡眠状态时，处理器可以关闭其内部时钟门，从而消耗更少的功率。当**两个**处理器都处于睡眠状态，并且DMA处于非活动状态时，RP 2040作为一个整体可以进入睡眠状态，禁用未使用的基础设施（如总线结构）上的时钟，并在其中一个处理器唤醒时自动唤醒 参见[第2.11.2节](#_bookmark272)。

* + 1. 调试

2线式串行总线（SWD）端口提供对硬件和软件调试功能的访问，包括：

* 将固件加载到SRAM或外部闪存
* 控制处理器执行：运行/停止、步进、设置断点、其他标准Arm调试功能
* 访问处理器体系结构状态
* 通过系统总线访问内存和内存映射IO

SWD总线暴露在两个专用引脚上，通电后立即可用。

##### 注意

我们建议最大SWD频率为24MHz。这在很大程度上取决于你的设置。您可能需要运行得更慢（1MHz），具体取决于电缆的质量和长度。

通过连接到共享多点SWD总线（SWD v2）的独立DAP（每个核心一个）进行以太网访问如果SWDTARGETSEL命令正确寻址，则每个DAP仅响应调试命令;所有其他DAP的输出均为三态此外，还提供了连接到系统控制功能的救援DP（参见[第2.3.4.2节](#_bookmark114)）。每个调试端口的默认地址如下所示：

* 核心0：0x01002927
* 核心1：0x11002927
* 救援DP：0xf1002927

实例ID（上面ID的前4位）可以通过sysconfig寄存器进行更改，这在多芯片应用中可能很有用。但是，请注意，ID=0xf是为内部Rescue DP保留的（请参见[2.3.4.2部分](#_bookmark114)）。

*图10. RP2040系列*

sys\_cfg.proc0\_dap\_instid

处理器

IO

社署

DAP\_0

DP-0AP

SWCLK

SWDIO

社署

多点仲裁器

社署

DAP\_1

DP-1AP

Core1

社署

救援DP

pam重启

sys\_cfg.proc1\_dap\_instid

核心0

* + - 1. SWD引脚的软件控制

内核0和内核1的SWD引脚可以通过syscfg中的寄存器进行位转换（参见DBGFORCE）。这意味着Core 1可以运行允许Core 0或类似调试的USB应用程序

* + - 1. 救援DP

Rescue DP（调试端口）可通过SWD总线使用，仅用于芯片锁定的特定情况，例如，如果代码已被编程到闪存中，永久停止系统时钟：在这种情况下，正常调试器无法与处理器通信以将系统返回到工作状态，因此需要采取更激烈的行动。通过设置救援DP的CTRL/STAT寄存器中的CDBGPWRUPREQ位来调用救援。

这会导致芯片的硬复位（功能上类似于上电复位），并在芯片级复位块中设置一个标志，以指示发生了救援复位。bootrom在初始引导过程中（在看门狗、闪存或USB引导之前）几乎立即检查此标志，通过清除该位进行确认，然后停止处理器。这使系统处于安全状态，系统时钟运行，以便调试器可以重新连接到内核并加载新的代码。

有关使用Rescue DP的实际示例，请参阅《**[使用RP 2040进行硬件设计](https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf)**》一书。

* 1. 皮质-M0+

ARM文档

[Cortex-M0+技术参考](https://developer.arm.com/documentation/ddi0484/latest)手册经允许使用

ARM Cortex-M0+处理器是一款极低门数、高能效的处理器，适用于需要面积优化、低功耗处理器的微控制器和深度嵌入式应用

* + 1. 特征

ARM Cortex-M0+处理器的特性和优势包括：

* 系统外围设备的紧密集成减少了面积和开发成本。
* Thumb指令集结合了高代码密度和32位性能。
* 支持单周期I/O访问。
* 系统组件的功率
* 集成休眠模式，实现低功耗。
* 快速代码执行使处理器能够以较慢的时钟运行或增加睡眠模式时间。
* 优化的代码提取，以降低闪存和ROM功耗。
* 硬件乘法器。
* 针对时间关键型应用的确定性高性能中断处理
* 确定性指令周期定时。
* 支持系统级调试身份验证。
* 串行线调试减少了调试所需的引脚数量
  + - 1. 接口

处理器中用于外部访问的接口包括：

* 总线结构的外部AHB-Lite接口
* 无线接入端口（DAP）
* 单周期I/O端口到SIO外设
  + - 1. 配置

每个处理器配置有以下功能：

* 架构时钟门控（用于节能）
* LittleEndian巴士通道
* 四个断点
* UART支持（通过2线调试引脚SWD/SWCLK）
* 32-位指令提取（匹配32位数据总线）
* IOPORT（用于低延迟访问本地外围设备（请参阅[SIO](#_bookmark44)）
* 26个中断
* 8个MPU区域
* 上电时所有寄存器复位
* 快速乘法器（MULS 32×32单周期）
* SysTick定时器
* 向量表偏移寄存器（[VTOR](#_bookmark151)）
* 34条WIC（唤醒控制器）线（32条IRQ和NMI、RXEV）
* DAP功能：停止事件支持
* DAP功能：SerialWire调试接口（协议2，支持多点
* DAP功能：未实现微跟踪缓冲区（MTB）

架构时钟门控允许处理器内核通过禁用处理器内核部分请注意，不支持电源门控

每个M0+内核都有自己的中断控制器，可以根据需要单独屏蔽中断源相同的

* + - 1. ARM架构

处理器实现ARMv 6-M架构配置文件。请参阅[ARMv 6-M架构参考手册](https://developer.arm.com/documentation/ddi0419/latest/)，更多详情请参阅[ARM Cortex M0+技术参考手册](https://developer.arm.com/documentation/ddi0484/latest/)。

* + 1. 功能描述
       1. 概述

Cortex-M0+处理器是一款可配置的多级32位RISC处理器。它有一个AMBA AHB-Lite接口，并包括一个NVIC组件。它还具有硬件调试、单周期I/O接口和内存保护功能。该处理器可以执行Thumb代码，并与其他Cortex-M配置文件处理器兼容

[图11](#_bookmark118)示出了处理器的功能块和周围的块。

*图11. Cortex M0+功能框图*

Cortex-M0+子系统

AHB-LiteMaster

时钟

PMU

HCLK FCLK DCLK

Cortex M0+核心

MPU

总线接口

单周期IO端口

复位

联系我们

NVIC

断点和观察点单元

调试器接口

中断

串行线调试

WIC

DAP

* + - 1. 特征

M0+的特点：

* ARMv6-M Thumb®指令集
* 拇指-2技术。
* ARMv 6-M兼容的24位SysTick定时器。
* 一个32位硬件乘法器。这是标准的单周期乘数
* 具有确定性、固定延迟、中断处理的能力。
* 加载/存储多条可以放弃和重新启动的指令，以促进快速中断处理。
* C应用程序二进制接口兼容异常模型。这是ARMv 6-M，C应用程序二进制接口（C-ABI）兼容的异常模型，支持使用纯C函数作为中断处理程序。
* 低功耗睡眠模式进入使用等待休眠（WFI），等待事件（WFE）指令，或从中断睡眠退出功能返回
  + - 1. NVIC特点

嵌套矢量控制器（NVIC）的特点是：

* 26个外部中断输入，每个输入具有四个优先级
* 专用不可屏蔽中断（NMI）输入（可从任何标准中断源驱动
* 支持电平敏感和脉冲敏感中断线路。
* 唤醒控制器（WIC），提供超低功耗睡眠模式支持。
* 可重定位向量表。

##### 注意

NVIC支持异常的硬件嵌套，例如，如果在处理程序运行时有更高优先级的中断请求到达，则中断处理程序本身可能会

更多详情见[第2.4.5节](#_bookmark126)。

* + - 1. 调试特征

这些功能包括：

* 四个硬件断点。
* 两个观察点。
* 程序计数器采样寄存器（PCSR）用于非侵入式代码分析。
* 单步和矢量捕获能力。
* 支持使用BKPT指令的无限软件断点。
* 通过一个紧凑的总线矩阵非侵入式访问核心外设和零等待状态系统从机调试器可以访问这些设备，包括内存，即使处理器正在运行。
* 当处理器暂停时，可以完全访问内核寄存器。
* CoreSight兼容调试访问通过支持串行线调试连接的以太网访问端口（DAP）。
  + - * 1. 调试访问端口

该处理器采用低门数的高速存取端口（DAP）实现。低门数的UART访问端口（DAP）提供串行线调试端口，并连接到处理器从机端口以提供完整的系统级调试访问。有关DAP的更多信息，请参见ADI v5.1版本的ARM UART Interface v5，架构规范

* + - 1. MPU特点

存储器保护单元（MPU）的特点是：

* 八个用户可配置的内存区域。
* 每个区域禁用八个子区域
* 永远不执行（XN）支持。
* 默认内存映射支持。更多详情见[第2.4.6节](#_bookmark128)。
  + - 1. AHB-Lite接口

AHB-Lite接口上的事务始终标记为非顺序的。处理器访问和调试访问共享外部AHB外设的外部接口。处理器访问优先于调试访问。任何特定于供应商的组件都可以填充此总线。

**注意**

指令只能使用AHB-Lite接口获取。为了优化性能，Cortex-M0+处理器会在其执行的指令之前进行读取。为了最大限度地降低功耗，预取限制为最大32位。

* + - 1. 单周期I/O端口

该处理器实现了一个单周期I/O端口，可高速访问紧密耦合的外设，如通用I/O（GPIO）。该端口可由来自处理器或调试器的加载和存储访问。无法从I/O端口执行代码

* + - 1. 功率管理单元

每个处理器都有自己的电源管理单元（PMU），可以通过关闭处理器核心部分的时钟来节省电源。RP 2040上没有单独的电源域。

PMU根据处理器时钟运行，处理器时钟由芯片级时钟块控制。PMU可以控制处理器内的以下时钟域：

* 包含处理器调试资源和DAP其余部分的调试时钟。
* 包含NVIC的系统时钟。
* 包含内核和相关接口的处理器时钟

控制仅限于时钟使能/禁用。启用时，所有域都以相同的时钟速度运行

PMU还与WIC接口，以确保掉电和唤醒行为对软件透明这包括SCR寄存器中控制的睡眠或深度睡眠支持。

* + - * 1. 电源管理

RP 2040 ARM Cortex M0+使用ARMv 6-M，支持使用等待启动（WFI）和等待事件（WFE）指令作为系统电源管理的一部分：

WFI提供了一种机制，用于硬件支持进入一个或多个睡眠状态。硬件可以暂停执行

直到唤醒事件发生。

WFE为软件提供了一种机制，用于暂停程序执行，直到出现唤醒条件，对唤醒延迟的影响最小或没有WFI和WFE都是提示指令，可能对程序执行没有影响。通常，它们用于软件空闲循环，仅在中断或感兴趣的事件发生后才恢复程序执行。

注意

使用WFE和WFI的代码必须处理由调试暂停或其他原因引起的任何虚假唤醒事件。

请参阅SDK和ARMv 6-M指南了解更多信息。

* + - * 1. 等待事件并发送事件

RP2040可以支持基于软件的系统事件同步，使用发送事件（SEV）和WFE提示指令。软件可以：

* 使用WFE指令来指示它能够挂起进程或线程的执行，直到事件发生，允许硬件进入低功率状态。
* 依赖于对软件透明并提供低延迟唤醒的机制

WFE机制依赖于硬件和软件一起工作来实现节能。例如，暂停处理器的执行，直到设备或另一处理器设置了标志：

* 硬件提供进入WFE低功率状态的机制。
* 软件进入轮询循环以确定何时设置标志
* 如果该标志被清除，则轮询处理器发出WFE指令作为轮询循环的一部分。
* 当设置标志时，产生事件（硬件中断或来自另一处理器的发送事件指令）。

WFE**唤醒事件**

以下事件是WFE唤醒事件：

* 在另一个处理器上执行SEV指令
* 如果系统控制寄存器中的SEVONPEND设置为1，则进入挂起状态的任何异常。
* 异步异常，优先于任何当前活动的异常。
* 启用调试的调试事件。

事件寄存器

事件寄存器是一个单比特寄存器。当设置时，事件寄存器指示自寄存器上次被清除以来发生的事件，这可能防止处理器在发出WFE指令时必须挂起操作以下条件适用于事件登记册：

* 复位将清除事件寄存器。
* 任何WFE唤醒事件或异常返回指令的执行都会设置事件寄存器。
* WFE指令清除事件寄存器。
* 软件无法直接读取或写入事件寄存器的值

发送事件指令

SEV（Send-Event）指令将事件发送给另一个处理器。Send-Event指令生成唤醒事件。

等待事件指令

WFE指令的操作取决于事件寄存器的状态

* 如果设置了事件寄存器，则指令将清除寄存器并立即返回
* 如果事件寄存器被清除，则处理器可以暂停执行并进入低功耗状态。它可以保持在

直到处理器检测到WFE唤醒事件或复位。当处理器检测到WFE唤醒事件时，WFE指令完成。

WFE唤醒事件可以在发出WFE指令之前发生。使用WFE机制的软件必须容忍虚假唤醒事件，包括多次唤醒。

* + - * 1. 等待中断

RP 2040通过提示指令WFI支持等待灭菌。

当处理器发出WFI指令时，它可以暂停执行并进入低功耗状态。它可以保持该状态，直到处理器检测到以下WFI唤醒事件之一

* 一个重置。
* 一个优先级异步异常，如果PRIMASK.PM设置为0，将抢占任何当前活动的异常。

注意

如果PRIMASK.PM设置为1，则组优先级高于任何活动异常的异步异常将导致WFI指令退出。如果异常的组优先级小于或等于执行组优先级，则忽略该异常

* 如果启用了调试，则为调试事件。
* WFI唤醒事件。

当硬件检测到WFI唤醒事件时，WFI指令完成处理器仅在发出WFI指令后识别WFI唤醒事件

* + - * 1. 唤醒控制器

唤醒休眠控制器（WIC）用于将处理器从[SCR](#_bookmark153)寄存器控制的深度休眠状态唤醒在深度休眠状态下，处理器内核和NVIC的时钟未运行。从深度睡眠状态唤醒可能需要几个周期

WIC从接收事件信号（来自另一个处理器）、32条中断线和NMI获取输入。

为了更省电，RP 2040支持[第2.11节](#_bookmark270)中定义的系统级省电模式，其中还包括代码示例。

* + - 1. 复位控制

Cortex M0+复位控制模块控制以下复位：

* 复位
* M0+核心复位
* PMU复位

上电后，两个处理器均从复位状态释放（详见[第2.13.2节](#_bookmark293)）。这将释放复位到UART、M0+内核和PMU。

一旦运行，复位可以从触发器、NVIC（使用[AIRCR](#_bookmark152).SYSRESETREQ）或RP 2040上电状态机控制器触发（详见[第2.13](#_bookmark291)）。NVIC仅重置Cortex-M0+处理器内核（而不是CPU或PMU），而上电状态机控制器可以重置处理器子系统，该子系统断言子系统（CPU、M0+内核、PMU）中的所有重置。

* + 1. 程序员模型
       1. 关于程序员的模型

ARMv 6-M架构参考手册提供了对编程器模型的完整描述本章概述了Cortex-M0+编程器的模型，该模型描述了实现定义的选项。它还包含它使用的ARMv 6-M Thumb指令及其处理器的周期计数。其他详细信息见以下章节

* [第2.4.4节](#_bookmark123)总结了程控仪模型的系统控制功能。
* [第2.4.5节](#_bookmark126)总结了程控仪型号的NVIC功能。
* [第2.3.4节](#_bookmark113)总结了编程器模型的可扩展特性。
  + - 1. 运作和执行方式

有关操作和执行模式的信息，请参见ARMv 6-M架构参考手册

* + - 1. 指令集摘要

表81.关于联合国难民事务高级专员的报告Cortex-M0+指令摘要

该处理器实现ARMv 6-M Thumb指令集，包括许多使用Thumb-2技术的32位指令ARMv 6-M指令集包括：

* ARMv7-M的所有16位Thumb指令，不包括CBZ、CBNZ和IT。
* 32位Thumb指令BL、DMB、DSB、ISB、MRS和MSR。

[表81](#_bookmark120)显示了Cortex-M0+指令及其周期计数。循环计数基于零等待状态的系统。

|  |  |  |  |
| --- | --- | --- | --- |
| 操作 | 描述 | 汇编程序 | 周期 |
| 移动 | 8位立即数 | MOVS Rd，#imm> | 1 |
|  | 罗，罗 | MOVS Rd，Rm | 1 |
|  | 任何到任何 | MOV Rd，Rm | 1 |
|  | 任何到PC | MOV PC，RM | 2 |
| 添加 | 3位立即数 | ADDS Rd，Rn，#imm> | 1 |
|  | 所有寄存器日志 | ADDS Rd，Rn，Rm | 1 |
|  | 任何到任何 | ADD Rd，Rd，Rm | 1 |
|  | 任何到PC | 添加PC、PC、RM | 2 |
|  | 8位立即数 | ADDS Rd，Rd，#imm> | 1 |
|  | 带进位 | ADCS Rd，Rd，Rm | 1 |
|  | 直接到SP | ADD SP，SP，#imm> | 1 |
|  | 来自SP的表格地址 | ADD Rd，SP，#imm> | 1 |
|  | 来自PC的表单地址 | ADR Rd，<label> | 1 |
| 减去 | 罗和罗 | SUBS Rd，Rn，Rm | 1 |
|  | 3位立即数 | SUBS Rd，Rn，#imm> | 1 |
|  | 8位立即数 | SUBS Rd，Rd，#imm> | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| 操作 | 描述 | 汇编程序 | 周期 |
|  | 带进位 | SBCS Rd，Rd，Rm | 1 |
|  | 直接从SP | SP，SP，#imm> | 1 |
|  | 否定 | RSBS Rd，Rn，#0 | 1 |
| 繁殖 | 繁殖 | Muls Rd，Rm，Rd | 1 |
| 比较 | 比较 | CMP Rn、Rm | 1 |
|  | 负 | CMN Rn，Rm | 1 |
|  | 立即 | CMP RN，#imm> | 1 |
| 逻辑 | 和 | ANDS Rd，Rd，Rm | 1 |
|  | 异或 | EORS Rd，Rd，Rm | 1 |
|  | 或 | ORRS Rd，Rd，Rm | 1 |
|  | 位清透 | BICS Rd，Rd，Rm | 1 |
|  | 举动没有 | MVNS Rd，Rm | 1 |
|  | 和测试 | TST Rn，Rm | 1 |
| 移位 | 立即数逻辑左移 | LSLS Rd，Rm，#shift> | 1 |
|  | 寄存器逻辑左移 | LSLS Rd，Rd，Rs | 1 |
|  | 逻辑右移立即数 | LSRS Rd，Rm，#shift> | 1 |
|  | 寄存器逻辑右移 | LSRS Rd，Rd，Rs | 1 |
|  | 算术右移 | ASRS Rd，Rm，#shift> | 1 |
|  | 按寄存器算术右移 | ASRS Rd，Rd，Rs | 1 |
| 旋转 | 通过寄存器向右旋转 | RORS Rd，Rd，Rs | 1 |
| 负载 | 字，立即偏移 | LDR Rd，[Rn，#imm>] | 2或1a |
|  | 半字，立即偏移 | LDRH Rd，[Rn，#imm>] | 2或1a |
|  | 字节，立即偏移 | LDRB Rd，[Rn，#imm>] | 2或1a |
|  | 字，寄存器偏移 | LDR Rd，[Rn，Rm] | 2或1a |
|  | 半字，寄存器偏移 | LDRH Rd，[Rn，Rm] | 2或1a |
|  | 有符号半字，寄存器偏移 | LDRSH Rd，[Rn，Rm] | 2或1a |
|  | 字节，寄存器偏移 | LDRB Rd，[Rn，Rm] | 2或1a |
|  | 带符号字节，寄存器偏移 | LDRSB Rd，[Rn，Rm] | 2或1a |
|  | PC-相对 | LDR Rd，<label> | 2或1a |
|  | SP-相对 | LDR Rd，[SP，#imm>] | 2或1a |
|  | 多个，不包括基地 | LDM Rn！，联系我们 | 1+Nb |
|  | 多个，包括基地 | LDM Rn，{loreglist>} | 1+Nb |
| 店 | 字，立即偏移 | STR Rd，[Rn，#imm>] | 2或1a |
|  | 半字，立即偏移 | StRH Rd，[Rn，#imm>] | 2或1a |
|  | 字节，立即偏移 | STRB Rd，[Rn，#imm>] | 2或1a |
|  | 字，寄存器偏移 | STR Rd，[Rn，Rm] | 2或1a |

|  |  |  |  |
| --- | --- | --- | --- |
| 操作 | 描述 | 汇编程序 | 周期 |
|  | 半字，寄存器偏移 | STRH Rd，[Rn，Rm] | 2或1a |
|  | 字节，寄存器偏移 | STRB Rd，[Rn，Rm] | 2或1a |
|  | SP-相对 | STR Rd，[SP，#imm>] | 2或1a |
|  | 多 | STM Rn！，联系我们 | 1+Nb |
| 推 | 推 | PUSH{loreglist>} | 1+Nb |
|  | 带链接寄存器的推送 | PUSH {loreglist>，LR} | 1+Nc |
| 流行 | 流行 | POP{loreglist>} | 1+Nb |
|  | 弹出并返回 | POP {loreglist>，PC} | 3+Nc |
| 分支 | 条件 | B cc><label> | 1或2天 |
|  | 无条件 | B标签> | 2 |
|  | 与链路 | BL<标签> | 3 |
|  | 与exchange | BX室 | 2 |
|  | 链接和交换 | BLX室 | 2 |
| 延伸 | 带符号半字到字 | SXTH Rd，Rm | 1 |
|  | 带符号字节到字 | SXTB Rd，Rm | 1 |
|  | 无符号半字 | UXTH Rd，Rm | 1 |
|  | 无符号字节 | UXTB Rd，Rm | 1 |
| 反向 | 用词 | REV Rd，Rm | 1 |
|  | 在两个半字中 | REV 16 Rd，RM | 1 |
|  | 带符号下半字 | REVSH Rd，Rm | 1 |
| 状态 | 更改主管呼叫 | SVC#imm> | -e |
|  | 禁用中断 | CPSIDi | 1 |
|  | 启用中断 | CPSIEi | 1 |
|  | 读专用寄存器 | MRS Rd，<specreg> | 3 |
|  | 写专用寄存器 | MSR specreg>，Rn | 3 |
|  | 断点 | BKPT#imm> | -e |
| 提示 | 发送事件 | SEV | 1 |
|  | 等待事件 | WFE | 2F |
|  | 等待中断 | WFI | 2F |
|  | 产量 | 产量 | 1f |
|  | 无操作 | NOP | 1 |
| 壁垒 | 指令同步 | ISB | 3 |
|  | 数据存储器 | DMB | 3 |
|  | 数据同步 | DSB | 3 |

表格附注

a2如果到AHB接口或SCS，1如果到单周期I/O端口。

bN是列表中元素的个数。

cN是列表中包含PC或LR的元素的数量。

d2如果服用，1如果未服用。

e周期计数取决于处理器和调试配置。

f不包括等待中断或事件的时间。

g作为国家行动方案执行。

有关ARMv 6-M Thumb指令的更多信息，请参见ARMv 6-M架构参考手册。

* + - 1. 内存模型

表82.M0+默认内存映射使用

处理器包含一个总线矩阵，用于仲裁处理器内核和外部存储器系统以及内部NVIC和调试组件的DAP存储器访问。

优先级总是给予处理器，以确保任何调试访问尽可能是非侵入性对于零等待状态系统，对系统存储器、NVIC和调试资源的所有调试访问对于典型代码执行是完全非侵入

系统内存映射符合ARMv 6-M架构，并且对于调试器和处理器访问都是通用交易按以下方式进行

* 所有低于0xd 0000000或高于0xefffffff的访问在处理器的AHB-Lite主端口上显示为AHB-Lite事务
* 范围为0xd0000000到0xdfffffff的访问由SIO处理。
* 0xe 0000000到0xefffffff范围内的访问在处理器内处理，不会出现在处理器的AHB-Lite主端口上。

处理器仅支持0xd 0000000 - 0xefffffff范围内的字长访问。

[表82](#_bookmark121)显示了默认内存映射每个区域的代码、数据和设备适用性这是MPU禁用时实现使用的除了针对Cortex-M0+ NVIC和调试组件的区域外，所有区域的属性和权限都可以使用已实现的MPU进行修改。

|  |  |  |  |
| --- | --- | --- | --- |
| 地址范围 | 代码 | 数据 | 装置 |
| 0xf0000000-0xffffffff | 没有 | 没有 | 是的 |
| 0xe0000000-0xefffffff | 没有 | 没有 | 没有a |
| 0xa0000000-0xdfffffff | 没有 | 没有 | 是的 |
| 0x60000000-0x9fffffff | 是的 | 是的 | 没有 |
| 0x40000000-0x5fffffff | 没有 | 没有 | 是的 |
| 0x20000000-0x3fffffff | 是的 | 是的 | 没有 |
| 0x0000000-0x1fffffff | 是的 | 是的 | 没有 |

a. 为Cortex-M0+ NVIC和调试组件保留的空间。

注意

未标记为适合代码的区域的行为与eXecute-Never（XN）相同，如果代码试图从该位置执行，则会生成HardFault异常

有关内存模型的更多信息，请参见

* + - 1. 处理器核心寄存器摘要

[表83](#_bookmark122)显示了处理器核心寄存器集摘要。 这些寄存器中的每一个都是32位宽。

表83.M0+处理器内核寄存器集摘要

|  |  |
| --- | --- |
| 名称 | 描述 |
| R0-R12 | R 0-R12是用于数据操作的通用寄存器 |
| MSP/PSP（R13） | 堆栈指针（SP）是寄存器R13。在线程模式下，CONTROL寄存器指示要使用的堆栈指针 |
| LR（R14） | 链路寄存器（LR）是寄存器R14。 它存储子程序、函数调用和异常的返回信息。 |
| PC（R15） | 程序计数器（PC）是寄存器R15。它包含当前程序地址。 |
| PSR | 程序状态寄存器（PSR）包括：   * 应用程序状态寄存器（APSR）。 * 程序状态寄存器（IPSR）。 * 执行程序状态寄存器（EPSR）。这些寄存器提供PSR的不同视图 |
| PRIMASK | PRIMASK寄存器可防止激活具有可配置优先级的 |
| 控制 | 当处理器处于线程模式时，CONTROL寄存器控制所使用的堆栈和代码特权级别 |

注意

有关处理器内核寄存器及其地址、访问类型和复位值的信息，请参阅ARMv 6-M架构参考手册。

* + - 1. 例外

本节描述处理器的异常模型

* + - * 1. 异常处理

该处理器实现高级异常和中断处理，如《ARMv 6-M架构参考手册》中所述。为了最小化中断延迟，处理器放弃任何加载多路或存储多路指令以执行任何挂起中断。在从中断处理程序返回时，处理器从头开始重新启动load-multiple或

这意味着当在读敏感或对重复写敏感的存储器区域中访问设备时，软件不得使用加载多个或存储多个指令。在重复读取或写入可能导致不一致的结果或不必要的副作用的任何情况

处理器实现可以确保NVIC检测中断信号并且处理器获取相关联的中断处理程序的第一指令需要固定数量的周期。 如果这样做，最高优先级的中断就没有抖动。这将取决于中断处理程序位于何处，以及是否有另一个更高优先级的主机正在访问该内存。SRAM 4和SRAM 5可以分配给每个处理器的中断处理程序

为了减少中断延迟和抖动，Cortex-M0+处理器实现了ARMv 6-M架构定义的中断延迟到达和中断尾链机制。在不使用抖动抑制的零等待状态系统中，最高优先级活动中断的最坏情况中断延迟为15个周期。

除了体系结构之外，处理器异常模型还具有以下实现定义的行为

特定行为：

* + 在NMI优先级下，从HardFault到NMI锁定的堆栈上进行重新配置。
  + 在HardFault优先级下，从NMI拆堆到HardFault锁定时打开。
    1. 系统控制
       1. 系统控制寄存器摘要

*表84.对妇女的歧视M0+系统控制寄存器*

|  |  |
| --- | --- |
| 名称 | 描述 |
| [公司简介](#_bookmark133) | SysTick控制和状态寄存器 |
| [系统\_RVR](#_bookmark134) | SysTick计数值寄存器 |
| [系统\_CVR](#_bookmark135) | SysTick当前值寄存器 |
| [SYST\_CALIB](#_bookmark136) | SysTick校准值寄存器 |
| [CPUID](#_bookmark149) | 参见[CPUID](#_bookmark149)寄存器 |
| [ICSR](#_bookmark150) | 控制状态寄存器 |
| [AIRCR](#_bookmark152) | 应用复位和复位控制寄存器 |
| [CCR](#_bookmark154) | 配置和控制寄存器 |
| [SHPR2](#_bookmark155) | 系统优先级寄存器 |
| [SHPR3](#_bookmark156) | 系统优先级寄存器 |
| [SHCSR](#_bookmark157) | 系统控制和状态寄存器 |
| [VTOR](#_bookmark151) | 向量表偏移寄存器 |
| ACTLR | 辅助控制寄存器 |

[表84](#_bookmark124)给出了系统控制寄存器。 这些寄存器中的每一个都是32位宽。

注意

* 所有系统控制寄存器只能通过字传输访问任何试图读取或写入半字或字节的操作都是不可预测的
* 有关系统控制寄存器及其地址和访问类型以及复位值的更多信息，请参见寄存器列表或ARMv 6-M架构参考手册
  + - * 1. CPUID寄存器

[CPUID](#_bookmark149)包含特定于处理器的部件号、版本和实现信息。

**重要**

此标准内部Arm寄存器包含有关处理器类型的信息它不应与[CPUID](#_bookmark48)（[第2.3.1.1节](#_bookmark46)）混淆，

* + 1. NVIC
       1. 关于NVIC

外部中断信号连接到嵌套矢量中断控制器（NVIC），NVIC对中断进行优先级排序。软件可以设置每个中断的优先级。NVIC和Cortex-M0+处理器内核紧密耦合，提供低延迟中断处理和有效处理延迟中断。

##### 注意

“嵌套”是指中断本身可以被高优先级中断。“向量化”指的是硬件将每个中断分派到由向量表指定的不同处理程序例程嵌套和矢量化行为的详细信息在ARMv 6-M架构参考手册中给出

所有NVIC寄存器只能通过字传输访问任何单独读取或写入半字或字节的尝试都是不可预测的。

NVIC寄存器总是小端的。

处理器异常处理将在第二节中描述。

* + - * 1. SysTick定时器

24位SysTick系统定时器扩展了处理器和NVIC的功能，并提供：

* + 一个24位系统定时器（SysTick）。
  + 额外的可配置优先级SysTick中断。

SysTick定时器使用1μs脉冲作为时钟使能。这在看门狗块中生成为timer\_tick。SysTick计时的准确性取决于此timer\_tick的准确性 SysTick计时器也可以从系统时钟运行（参见[SYST\_CALIB](#_bookmark136)）。

有关更多信息，请参阅ARMv 6-M架构参考手册。

* + - * 1. 低功率模式

该实现包括WIC。这使得处理器和NVIC能够进入非常低功耗的睡眠模式，让WIC识别中断并确定中断的优先

处理器完全执行等待事件（WFI）、等待事件（WFE）和发送事件（SEV）指令。此外，该处理器还支持使用SLEEPONEXIT，这会导致处理器内核在从异常处理程序返回到线程模式时进入睡眠模式。有关更多信息，请参阅ARMv 6-M架构参考手册。

* + - 1. NVIC寄存器摘要

[表85](#_bookmark127)显示了NVIC寄存器。 这些寄存器中的每一个都是32位宽。

*表85.M0+NVIC*

|  |  |
| --- | --- |
| 名称 | 描述 |
| [公司简介](#_bookmark137) | 设置启用寄存器。 |
| [公司简介](#_bookmark138) | 清除启用寄存器。 |
| [NVIC\_ISPR](#_bookmark139) | 挂起设置寄存器。 |
| [NVIC\_ICPR](#_bookmark140) | 清除挂起寄存器。 |
| [NVIC\_IPR0](#_bookmark141)-[NVIC\_IPR7](#_bookmark148) | 优先级寄存器。 |

寄存器

注意

有关NVIC寄存器及其地址、访问类型和复位值的更多信息，请参见寄存器列表或ARMv 6-M架构参考手册

* + 1. MPU
       1. 关于MPU

MPU是一个内存保护组件，允许处理器支持ARMv6受保护内存系统架构模型。MPU提供全面支持：

* 八个统一的保护区。
* 重叠保护区域，区域优先级递增
  + 7=最高优先级。
  + 0=最低优先级。
* 访问权限。
* 正在将内存属性导出到系统。

MPU不匹配和权限冲突调用HardFault处理程序。有关更多信息，请参阅ARMv 6-M架构参考手册。

您可以使用MPU：

* 执行特权规则。
* 分开的过程。
* 管理内存属性。
  + - 1. MPU寄存器摘要

表86.关于联合国难民事务高级专员的报告M0+MPU

寄存器

[表86](#_bookmark129)显示MPU寄存器。 这些寄存器中的每一个都是32位宽。

注意

|  |  |
| --- | --- |
| 名称 | 描述 |
| [MPU\_类型](#_bookmark158) | MPU类型寄存器。 |
| [MPU\_CTRL](#_bookmark159) | MPU控制寄存器。 |
| [MPU\_RNR](#_bookmark160) | MPU区域号寄存器。 |
| [MPU\_RBAR](#_bookmark161) | MPU区域基地址寄存器。 |
| [MPU\_RASR](#_bookmark162) | MPU区域属性和大小寄存器。 |

* 有关MPU寄存器及其地址、访问类型和复位值的更多信息，请参见ARMv 6-M架构参考手册
* MPU支持从256字节到4Gb的区域大小，每个区域有8个子区域。
  + 1. 调试

基本调试功能包括处理器暂停、单步调试、处理器内核寄存器访问、复位和硬故障向量捕获、无限制的软件断点和完整的系统内存访问。查看ARMv 6-M架构

参考手册。

此设备的调试功能包括：

* 断点单元支持4个硬件断点。
* 一个监视点单元支持2个监视点。
  + 1. 登记册一览表

ARM Cortex-M0+寄存器从基址0xe 000000（在SDK中定义为[PPB\_BASE](#_bookmark42)）开始。

*表87. M0PLUS寄存器列表*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0xe010 | [公司简介](#_bookmark133) | SysTick控制和状态寄存器 |
| 0xe014 | [系统\_RVR](#_bookmark134) | SysTick计数值寄存器 |
| 0xe018 | [系统\_CVR](#_bookmark135) | SysTick当前值寄存器 |
| 0xe01c | [SYST\_CALIB](#_bookmark136) | SysTick校准值寄存器 |
| 0xe100 | [公司简介](#_bookmark137) | 设置启用寄存器 |
| 0xe180 | [公司简介](#_bookmark138) | 中断清除启用寄存器 |
| 0xe200 | [NVIC\_ISPR](#_bookmark139) | 中断设置挂起寄存器 |
| 0xe280 | [NVIC\_ICPR](#_bookmark140) | 中断清除挂起寄存器 |
| 0xe400 | [NVIC\_IPR0](#_bookmark141) | 优先级寄存器0 |
| 0xe404 | [NVIC\_IPR1](#_bookmark142) | 优先级寄存器1 |
| 0xe408 | [NVIC\_IPR2](#_bookmark143) | 优先级寄存器2 |
| 0xe40c | [NVIC\_IPR3](#_bookmark144) | 优先级寄存器3 |
| 0xe410 | [NVIC\_IPR4](#_bookmark145) | 优先级寄存器4 |
| 0xe414 | [NVIC\_IPR5](#_bookmark146) | 优先级寄存器5 |
| 0xe418 | [NVIC\_IPR6](#_bookmark147) | 优先级寄存器6 |
| 0xe41c | [NVIC\_IPR7](#_bookmark148) | 优先级寄存器7 |
| 0xed00 | [CPUID](#_bookmark149) | CPUID基址寄存器 |
| 0xed 04 | [ICSR](#_bookmark150) | 控制和状态寄存器 |
| 0xed08 | [VTOR](#_bookmark151) | 向量表偏移寄存器 |
| 0xed0c | [AIRCR](#_bookmark152) | 应用复位和复位控制寄存器 |
| 0xed10 | [SCR](#_bookmark153) | 系统控制寄存器 |
| 0xed14 | [CCR](#_bookmark154) | 配置和控制寄存器 |
| 0xed1c | [SHPR2](#_bookmark155) | 系统优先级寄存器2 |
| 0xed20 | [SHPR3](#_bookmark156) | 系统优先级寄存器 |
| 24小时 | [SHCSR](#_bookmark157) | 系统控制和状态寄存器 |
| 0xed90 | [MPU\_类型](#_bookmark158) | MPU类型寄存器 |
| 0xed94 | [MPU\_CTRL](#_bookmark159) | MPU控制寄存器 |
| 0xed98 | [MPU\_RNR](#_bookmark160) | MPU区域号寄存器 |
| 0xed9c | [MPU\_RBAR](#_bookmark161) | MPU区域基地址寄存器 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0xeda0 | [MPU\_RASR](#_bookmark162) | MPU区域属性和大小寄存器 |

表88. SYST\_CSR寄存器

表89. SYST\_RVR寄存器

### [M0PLUS](#_bookmark132)：SYST\_CSR寄存器

**偏移量**：0xe010

描述

使用SysTick控制和状态寄存器启用SysTick功能。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:17 | Reserved. | - | - | - |
| 16 | COUNTFLAG | 如果自上次读取后计时器计数为0，则返回1应用程序或调试器读取时清除。 | RO | 0x0 |
| 十5:三 | Reserved. | - | - | - |
| 2 | CLKSOURCE | SysTick时钟源。如果SYST\_CALIB报告NOREF，则始终读取为1。  查看SysTick定时器时钟源：  0 =外部参考时钟。  1=处理器时钟。 | RW | 0x0 |
| 1 | 票务信息 | 启用SysTick异常请求：  0 =倒计时到零不会断言SysTick异常请求。  1 =倒计时到零以断言SysTick异常请求。 | RW | 0x0 |
| 0 | 使 | 启用SysTick计数器：  0=计数器禁用。  1=计数器使能。 | RW | 0x0 |

### [M0PLUS](#_bookmark132)：SYST\_RVR寄存器

**偏移量**：0xe014

描述

使用SysTick计数值寄存器指定计数器达到0时要加载到当前值寄存器的起始值。它可以是0到0x00FFFFFF之间的任何值起始值0是可能的，但没有效果，因为从1计数到0时会激活SysTick中断和COUNTFLAG。此寄存器的复位值为UNKNOWN。

要生成一个周期为N个处理器时钟周期的多触发定时器，请使用RELOAD值N-1。例如，如果每隔100个时钟脉冲需要SysTick中断，则将RELOAD设置为99。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31：24 | Reserved. | - | - | - |
| 23:0 | 重新加载 | 计数器达到0时加载到SysTick当前值寄存器的值 | RW | 0x000000 |

### [M0PLUS](#_bookmark132)：SYST\_CVR寄存器

**偏移量**：0xe018

描述

使用SysTick当前值寄存器查找寄存器中的当前值此寄存器的复位值为UNKNOWN。

*表90.对妇女的歧视SYST\_CVR寄存器*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31：24 | Reserved. | - | - | - |
| 23:0 | 电流 | 读取返回SysTick计数器的当前值。此寄存器为写清除。写入任何值都会将寄存器清零。 清除此寄存器也会清除SysTick控制和状态寄存器的COUNTFLAG位。 | RW | 0x000000 |

表91.关于联合国难民事务高级专员的报告SYST\_CALIB

寄存器

表92. 公司简介

寄存器

### [M0PLUS](#_bookmark132)：SYST\_CALIB寄存器

**偏移**：0xe01c

描述

使用SysTick校准值寄存器，使软件能够使用除法和乘法缩放到任何所需的速度。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | 无参考 | 如果读数为1，则不提供参考时钟-SysTick控制和状态寄存器的CLKSOURCE位将强制为1，且不能清0。 | RO | 0x0 |
| 30 | 偏斜 | 如果读数为1，则10 ms的校准值不准确（由于时钟频率）。 | RO | 0x0 |
| 29:24 | Reserved. | - | - | - |
| 23:0 | TENMS | 一个可选的时钟值，用于10 ms（100 Hz）定时，受系统时钟偏差误差的影响。如果读数为0，则校准值未知。 | RO | 0x000000 |

### [M0PLUS](#_bookmark132)：NVIC\_ISER注册

**偏移量**：0xe100

描述

使用中断设置启用寄存器启用中断并确定当前启用哪些中断。

如果使能挂起中断，NVIC将根据中断的优先级激活中断。如果中断未被使能，断言其中断信号会将中断状态更改为挂起，但NVIC永远不会激活中断，无论其优先级如何。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:0 | SETENA | 置位使能位。写：  0=无影响。  1 =使能中断。阅读：  0 =已禁用。  1 =已启用故障排除 | RW | 0x00000000 |

### [M0PLUS](#_bookmark132)：NVIC\_ICER注册

**偏移量**：0xe180

描述

使用清零使能寄存器禁用中断并确定当前使能的中断

*表93. 公司简介*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:0 | CLRENA | 清零使能位。写：  0=无影响。  1 =禁用中断。阅读：  0 =已禁用。  1 =已启用故障排除 | RW | 0x00000000 |

寄存器

表94. NVIC\_ISPR

寄存器

表95.关于联合国难民事务高级专员的报告NVIC\_ICPR

寄存器

### [M0PLUS](#_bookmark132)：NVIC\_ISPR注册

**偏移量**：0xe200

描述

NVIC\_ISPR强制中断进入挂起状态，并显示哪些中断挂起。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:0 | 设置 | 置位挂起位。写：  0=无影响。  1 =将中断状态更改为挂起。阅读：  0 =中断未挂起。  1 =挂起。  注：将1写入对应于以下情况的NVIC\_ISPR位：挂起的中断无效。  禁用的中断将该中断的状态设置为挂起。 | RW | 0x00000000 |

### [M0PLUS](#_bookmark132)：NVIC\_ICPR注册

**偏移量**：0xe280

描述

使用清零挂起寄存器清除挂起中断并确定当前挂起的中断。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:0 | CLRPEND | 清除挂起位。写：  0=无影响。  1 =中断挂起状态和中断。阅读：  0 =中断未挂起。  1 =挂起。 | RW | 0x00000000 |

### [M0PLUS](#_bookmark132)：NVIC\_IPR0注册

**偏移量**：0xe400

描述

使用中断优先级寄存器为每个可用中断分配从0到3的优先级。0是最高优先级，3是最低优先级。

注：将1写入NVIC\_ICPR位不会影响相应中断的活动状态。这些寄存器只能通过字访问

*表96. NVIC\_IPR0*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:30 | IP\_3 | 中断优先级3 | RW | 0x0 |
| 29:24 | Reserved. | - | - | - |
| 23:22 | IP\_2 | 中断优先级2 | RW | 0x0 |
| 21:16 | Reserved. | - | - | - |
| 15:14 | IP\_1 | 中断1的优先级 | RW | 0x0 |
| 13:8 | Reserved. | - | - | - |
| 7：6 | IP\_0 | 中断优先级0 | RW | 0x0 |
| 5比零 | Reserved. | - | - | - |

寄存器

表97. NVIC\_IPR1

寄存器

表98. NVIC\_IPR2

寄存器

### [M0PLUS](#_bookmark132)：NVIC\_IPR1寄存器

**偏移量**：0xe404

描述

使用中断优先级寄存器为每个可用中断分配从0到3的优先级。0是最高优先级，3是最低优先级。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:30 | IP\_7 | 中断优先级7 | RW | 0x0 |
| 29:24 | Reserved. | - | - | - |
| 23:22 | IP\_6 | 中断优先级6 | RW | 0x0 |
| 21:16 | Reserved. | - | - | - |
| 15:14 | IP\_5 | 中断优先级5 | RW | 0x0 |
| 13:8 | Reserved. | - | - | - |
| 7：6 | IP\_4 | 中断优先级4 | RW | 0x0 |
| 5:0 | Reserved. | - | - | - |

### [M0PLUS](#_bookmark132)：NVIC\_IPR2寄存器

**偏移量**：0xe408

描述

使用中断优先级寄存器为每个可用中断分配从0到3的优先级。 0是最高优先级，3是最低优先级。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:30 | IP\_11 | 中断优先级11 | RW | 0x0 |
| 29:24 | Reserved. | - | - | - |
| 23:22 | IP\_10 | 中断优先级10 | RW | 0x0 |
| 21:16 | Reserved. | - | - | - |
| 15:14 | IP\_9 | 中断优先级9 | RW | 0x0 |
| 13:8 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 7：6 | IP\_8 | 中断优先级8 | RW | 0x0 |
| 5:0 | Reserved. | - | - | - |

表99. NVIC\_IPR3

寄存器

表100。NVIC\_IPR4

寄存器

### [M0PLUS](#_bookmark132)：NVIC\_IPR3寄存器

**偏移**：0xe40c

描述

使用中断优先级寄存器为每个可用中断分配从0到3的优先级。0是最高优先级，3是最低优先级。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:30 | IP\_15 | 中断优先级15 | RW | 0x0 |
| 29:24 | Reserved. | - | - | - |
| 23:22 | IP\_14 | 中断优先级14 | RW | 0x0 |
| 21:16 | Reserved. | - | - | - |
| 15:14 | IP\_13 | 中断优先级13 | RW | 0x0 |
| 13:8 | Reserved. | - | - | - |
| 7：6 | IP\_12 | 中断优先级12 | RW | 0x0 |
| 5:0 | Reserved. | - | - | - |

### [M0PLUS](#_bookmark132)：NVIC\_IPR4寄存器

**偏移量**：0xe410

描述

使用中断优先级寄存器为每个可用中断分配从0到3的优先级。0是最高优先级，3是最低优先级。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:30 | IP\_19 | 中断优先级19 | RW | 0x0 |
| 二19:24 | Reserved. | - | - | - |
| 23:22 | IP\_18 | 中断优先级18 | RW | 0x0 |
| 21:16 | Reserved. | - | - | - |
| 15:14 | IP\_17 | 中断优先级17 | RW | 0x0 |
| 13:8 | Reserved. | - | - | - |
| 7：6 | IP\_16 | 中断优先级16 | RW | 0x0 |
| 5:0 | Reserved. | - | - | - |

### [M0PLUS](#_bookmark132)：NVIC\_IPR5寄存器

**偏移量**：0xe414

表101.NVIC\_IPR5

寄存器

表102.NVIC\_IPR6

寄存器

表103.NVIC\_IPR7

寄存器

描述

使用中断优先级寄存器为每个可用中断分配从0到3的优先级。0是最高优先级，3是最低优先级。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:30 | IP\_23 | 中断优先级23 | RW | 0x0 |
| 29:24 | Reserved. | - | - | - |
| 23:22 | IP\_22 | 中断优先级22 | RW | 0x0 |
| 21:16 | Reserved. | - | - | - |
| 15:14 | IP\_21 | 中断优先级21 | RW | 0x0 |
| 13:8 | Reserved. | - | - | - |
| 7：6 | IP\_20 | 中断优先级20 | RW | 0x0 |
| 5:0 | Reserved. | - | - | - |

### [M0PLUS](#_bookmark132)：NVIC\_IPR6寄存器

**偏移量**：0xe418

描述

使用中断优先级寄存器为每个可用中断分配从0到3的优先级。0是最高优先级，3是最低优先级。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:30 | IP\_27 | 中断优先级27 | RW | 0x0 |
| 29:24 | Reserved. | - | - | - |
| 23:22 | IP\_26 | 中断优先级26 | RW | 0x0 |
| 21:16 | Reserved. | - | - | - |
| 15:14 | IP\_25 | 中断优先级25 | RW | 0x0 |
| 13:8 | Reserved. | - | - | - |
| 7：6 | IP\_24 | 中断优先级24 | RW | 0x0 |
| 5:0 | Reserved. | - | - | - |

### [M0PLUS](#_bookmark132)：NVIC\_IPR7注册

**偏移量**：0xe41c

描述

使用中断优先级寄存器为每个可用中断分配从0到3的优先级。 0是最高优先级，3是最低优先级。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:30 | IP\_31 | 中断优先级31 | RW | 0x0 |
| 29:24 | Reserved. | - | - | - |
| 23:22 | IP\_30 | 中断优先级30 | RW | 0x0 |
| 21:16 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 15:14 | IP\_29 | 中断优先级29 | RW | 0x0 |
| 13:8 | Reserved. | - | - | - |
| 7：6 | IP\_28 | 中断优先级28 | RW | 0x0 |
| 5:0 | Reserved. | - | - | - |

表104.对妇女的歧视CPUID

寄存器

### [M0PLUS](#_bookmark132)：CPUID寄存器

**偏移量**：0xed00

描述

读取CPU ID Base Register以确定：处理器核心的ID号、处理器核心的版本号、处理器核心的实现细节

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31：24 | 实施者 | 实现器代码：0x41 =ARM | RO | 0x41 |
| 23:20 | 变体 | rnpm修订状态中的主要修订号n：0x0 =修订0。 | RO | 0x0 |
| 19:16 | 架构 | 定义处理器架构的常量：0xC = ARMv 6-M架构。 | RO | 0xc |
| 15:4 | PartNo | 处理器数量：0xC 60 = Cortex-M0+ | RO | 0xc60 |
| 3:0 | 修订 | rnpm修订状态中的次修订号 | RO | 0x1 |

### [M0PLUS](#_bookmark132)：ICSR登记

**偏移量**：0xed04

描述

使用该寄存器控制状态寄存器来设置挂起的不可屏蔽寄存器（NMI）、设置或清除挂起的PendSV、设置或清除挂起的SysTick、检查挂起的异常、检查最高优先级挂起的异常的向量编号、检查活动异常的向量编号。

表105.对妇女的歧视ICSR注册

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | NMIPENDSET | 设置此位将激活NMI。由于NMI是优先级最高的异常，因此一旦注册，它将立即激活。  NMI置位挂起位。写：  0=无影响。  1 =将NMI异常状态更改为挂起。阅读：  0= NMI异常未挂起。  1= NMI异常挂起。  由于NMI是最高优先级的异常，因此处理器通常会  异常处理程序，一旦它检测到1写入此位。输入处理程序，然后清除  这位为0。这意味着NMI异常处理程序读取此位时  当处理器执行该处理程序时，NMI信号被重新声明 | RW | 0x0 |
| 30:29 | Reserved. | - | - | - |
| 28 | PENDSVSET | PendSV设置挂起位。写：  0=无影响。  1 =将PendSV异常状态更改为挂起。阅读：  0= PendSV异常未挂起。  1= PendSV异常挂起。  将1写入此位是将PendSV异常状态设置为挂起的唯一方法 | RW | 0x0 |
| 27 | PENDSVCLR | PendSV清除挂起位。写：  0=无影响。  1=从PendSV异常中删除挂起状态。 | RW | 0x0 |
| 26 | PENDSTSET | SysTick异常设置挂起位。写：  0=无影响。  1 =将SysTick异常状态更改为挂起。阅读：  0= SysTick异常未挂起。  1= SysTick异常挂起。 | RW | 0x0 |
| 25 | PENDSTYLE | SysTick异常清除挂起位。写：  0=无影响。  1=从SysTick异常中删除挂起状态。  这一点是WO。在寄存器读取时，其值为未知。 | RW | 0x0 |
| 24 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 23 | ISRPREEMPT | 系统只能在内核停止时访问此位。它表示在下一个运行周期中将采用挂起中断如果中断停止控制和状态寄存器中的C\_MASKINTS被清除，则中断被服务。 | RO | 0x0 |
| 22 | 正在等待 | 外部中断挂起标志 | RO | 0x0 |
| 21 | Reserved. | - | - | - |
| 20:12 | 矢量待定 | 指示最高优先级挂起异常的异常编号：0 =无挂起异常。非零=挂起状态包括内存映射使能和掩码寄存器的影响它不包括PRIMASK专用寄存器限定符。 | RO | 0x000 |
| 11:9 | Reserved. | - | - | - |
| 8:0 | 矢量 | 活动例外编号字段。复位清除VECTACTIVE字段。 | RO | 0x000 |

表106.对妇女的歧视VTOR寄存器

表107.对妇女的歧视AIRCR

寄存器

### [M0PLUS](#_bookmark132)：VTOR寄存器

**偏移量**：0xed08

描述

VTOR保存向量表偏移地址。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31：8 | TBLOFF | 的位[31：8]表示向量表偏移地址。 | RW | 0x000000 |
| 7:0 | Reserved. | - | - | - |

### [M0PLUS](#_bookmark132)：AIRCR注册

**偏移量**：0xed0c

描述

使用应用程序中断和复位控制寄存器：确定数据字节顺序，清除调试暂停模式中的所有活动

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | VECTKEY | 注册码：  读作未知  在写入时，将0x05FA写入VECTKEY，否则忽略写入 | RW | 0x0000 |
| 15 | 端序 | 实现的数据端序：0 =小端序。 | RO | 0x0 |
| 十4:三 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 2 | SYSRESETREQ | 将1写入该位会使发送至外部系统的SYSRESETREQ信号置位，请求复位。其目的是强制对除调试之外的所有主要组件进行大型系统重置由于系统重置请求，DHSR中的C\_HALT位被清除。调试器不会失去与设备的联系。 | RW | 0x0 |
| 1 | 矢量 | 清除固定和可配置异常的所有活动状态信息。此位：自清，仅在内核停止时由DAP设置设置时：清除处理器的所有活动异常状态，强制返回线程模式，强制IPSR为0。调试器必须重新初始化堆栈。 | RW | 0x0 |
| 0 | Reserved. | - | - | - |

表108.对妇女的歧视SCR寄存器

### [M0PLUS](#_bookmark132)：SCR寄存器

**偏移量**：0xed10

描述

系统控制寄存器。使用系统控制寄存器实现电源管理功能：当处理器可以进入低功耗状态时向系统发出信号，控制处理器如何进入和退出低功耗状态。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:5 | Reserved. | - | - | - |
| 4 | SEVONPEND | 发送挂起事件位：  0=只有启用的中断或事件可以唤醒处理器，禁用的中断被排除。  1=启用事件和所有中断（包括禁用中断）可以唤醒处理器。  当一个事件或中断成为挂起，事件信号唤醒处理器从WFE。如果处理器没有在等待事件，则注册该事件并影响下一个WFE。  处理器还在执行SEV指令或外部事件时唤醒 | RW | 0x0 |
| 3 | Reserved. | - | - | - |
| 2 | SLEEPDEEP | 控制处理器是使用睡眠模式还是深度睡眠模式作为其低功耗模式：  0=睡眠。  1=深度睡眠。 | RW | 0x0 |
| 1 | 睡眠出口 | 指示从线程模式返回到线程模式时退出休眠  0 =返回线程模式时不休眠  1=从ISR返回到线程模式时进入睡眠或深度睡眠  将此位设置为1，可使中断驱动应用程序避免返回空的主应用程序。 | RW | 0x0 |
| 0 | Reserved. | - | - | - |

表109.对妇女的歧视CCR寄存器

表110.对妇女的歧视SHPR2

寄存器

表111.对妇女的歧视SHPR3

寄存器

### [M0PLUS](#_bookmark132)：CCR寄存器

**偏移量**：0xed14

描述

配置和控制寄存器永久启用堆栈对齐，并导致未对齐的访问导致硬故障。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:10 | Reserved. | - | - | - |
| 9 | STKALIGN | 总是读作1，指示异常条目上的8字节堆栈对齐在异常进入时，处理器使用堆栈PSR的位[9]来指示堆栈对齐。在从异常返回时，它使用这个堆栈位来恢复正确的堆栈对齐。 | RO | 0x0 |
| 8：4 | Reserved. | - | - | - |
| 3 | UNALIGN\_TRP | 始终读取为1，表示所有未对齐的访问都会生成HardFault。 | RO | 0x0 |
| 2:0 | Reserved. | - | - | - |

### [M0PLUS](#_bookmark132)：SHPR2寄存器

**偏移**：0xed1c

描述

系统处理程序是一类特殊的异常处理程序，可以将其优先级设置为任何优先级。使用系统优先级寄存器2设置SVCall的优先级

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:30 | PRI\_11 | 系统处理程序11的优先级，SVCall | RW | 0x0 |
| 29:0 | Reserved. | - | - | - |

### [M0 PLUS](#_bookmark132)：SHPR 3注册

**偏移量**：0xed20

描述

系统处理程序是一类特殊的异常处理程序，可以将其优先级设置为任何优先级。使用系统时钟优先级寄存器3设置PendSV和SysTick的优先级

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:30 | PRI\_15 | 系统处理程序15的优先级，SysTick | RW | 0x0 |
| 29:24 | Reserved. | - | - | - |
| 23:22 | PRI\_14 | 系统处理程序14的优先级，PendSV | RW | 0x0 |
| 21:0 | Reserved. | - | - | - |

### [M0PLUS](#_bookmark132)：SHCSR注册

**偏移量**：0xed24

表112.对妇女的歧视SHCSR

寄存器

表113.对妇女的歧视MPU\_类型

寄存器

表114.MPU\_CTRL

寄存器

描述

使用系统故障控制和状态寄存器确定或清除SVCall的挂起状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - | - |
| 15 | SVCALLPENDED | 如果SVCall处于挂起状态，则读取为1写入1以设置待决SVCall，写入0以清除待决SVCall。 | RW | 0x0 |
| 14:0 | Reserved. | - | - | - |

### [M0PLUS](#_bookmark132)：MPU\_TYPE寄存器

**偏移**：0xed90

描述

读取MPU类型寄存器以确定处理器是否实现MPU，以及MPU支持多少区域。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23:16 | IREGION | 指令区。 由于ARMv 6-M仅支持统一的MPU，因此读数为零。 | RO | 0x00 |
| 15:8 | DREGION | MPU支持的区域数。 | RO | 0x08 |
| 7:1 | Reserved. | - | - | - |
| 0 | 单独 | 指示支持单独的指令和数据地址映射。读取为0，因为ARMv 6-M仅支持统一MPU。 | RO | 0x0 |

### [M0PLUS](#_bookmark132)：MPU\_CTRL寄存器

**偏移量**：0xed94

描述

使用MPU控制寄存器启用和禁用MPU，并控制是否启用默认内存映射作为特权访问的后台区域，以及是否启用MPU的硬故障和NMI。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:3 | Reserved. | - | - | - |
| 2 | PRIVDEFENA | 控制是否将默认内存映射启用为特权访问的后台区域。清除ENABLE时，忽略此位。  0 =如果启用MPU，则禁用默认内存映射。对未被任何启用区域覆盖的位置的任何内存访问  1 =如果启用MPU，则启用默认内存映射作为特权软件访问的后台区域  启用后，背景区域的作用就像区域编号-1一样。任何已定义和启用的区域都具有高于此默认贴图的优先级。 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 1 | HFNMIENA | 控制MPU对硬故障和NMI的使用当ENABLE清除时设置此位会导致不可预测的行为。  当MPU启用时：  0 =无论ENABLE位的值如何，在HardFault和NMI处理程序期间禁用MPU  1=在HardFault和NMI处理程序期间启用MPU。 | RW | 0x0 |
| 0 | 使 | 启用MPU。如果MPU被禁用，则特权和非特权访问使用默认内存映射。  0=禁用MPU。  1= MPU使能。 | RW | 0x0 |

表115.MPU\_RNR寄存器

表116.对妇女的歧视MPU\_RBAR

寄存器

### [M0PLUS](#_bookmark132)：MPU\_RNR寄存器

**偏移量**：0xed98

描述

使用MPU区域号寄存器选择MPU\_RBAR和MPU\_RASR当前访问的区域。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:4 | Reserved. | - | - | - |
| 3:0 | 区域 | 指示MPU\_RBAR和MPU\_RASR寄存器引用的MPU区域  MPU支持8个内存区域，因此该字段的允许值为0-7。 | RW | 0x0 |

### [M0PLUS](#_bookmark132)：MPU\_RBAR寄存器

**偏移**：0xed9c

描述

读取MPU区域基址寄存器，以确定MPU\_RNR标识的区域的基址。写入以更新所述区域或指定区域的基地址，其编号MPU\_RNR也将被更新。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31：8 | ADDR | 区域的基址。 | RW | 0x000000 |
| 7:5 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 4 | 有效 | 在写入时，指示写入是否必须更新REGION字段标识的区域的基址，更新MPU\_RNR以指示此新区域。  写：  0 = MPU\_RNR未更改，处理器：  更新MPU\_RNR中指定的区域的基址。  忽略REGION字段的值。1 =处理器：  将MPU\_RNR的值更新为REGION字段的值  更新REGION字段中指定的区域的基址  始终读取为零。 | RW | 0x0 |
| 3:0 | 区域 | 在写入时，指定要更新的基址（提供VALID）被设置为1的区域的编号读取时，返回MPU\_RNR的位[3：0]。 | RW | 0x0 |

表117.MPU\_RASR

寄存器

### [M0PLUS](#_bookmark132)：MPU\_RASR寄存器

**偏移量**：0xeda0

描述

使用MPU区域属性和大小寄存器定义MPU\_RNR标识的区域的大小、访问行为和内存类型，并启用该区域。

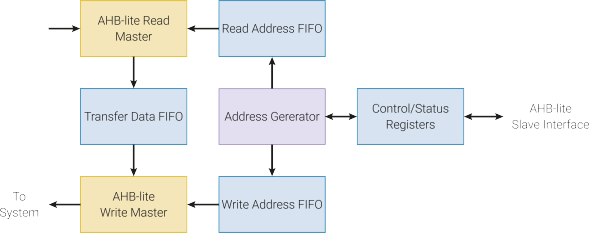
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | ATTRS | MPU区域属性字段。用于定义区域属性控件。  28 = XN：指令访问禁用位：  0 =指令提取已启用。  1 =禁用指令提取。26：24 = AP：访问权限字段18 = S：可共享位  17 = C：可缓存位16 = B：可缓冲位 | RW | 0x0000 |
| 15:8 | SRD | 子区域禁用。对于256字节或更大的区域，此字段的每个位控制是否启用八个相等子区域之一 | RW | 0x00 |
| 7：6 | Reserved. | - | - | - |
| 5:1 | 大小 | 指示区域大小。以字节为单位的区域大小= 2^（SIZE+1）。最小允许值为7（b00111）=256 Ω | RW | 0x00 |
| 0 | 使 | 启用区域。 | RW | 0x0 |

* 1. DMA

RP 2040直接内存访问（DMA）控制器具有独立的读和写主机连接到总线结构，并代表处理器执行批量数据传输这使得处理器可以自由地处理其他任务，或进入低功耗睡眠状态。DMA的数据吞吐量也明显高于RP 2040的处理器之一

图12. DMA体系结构概述。读主机可以在每个时钟周期从某个地址读取数据同样，写主机可以写入另一个地址。

地址生成器产生匹配的读写地址对，主机通过地址FIFO使用这些地址对。多达12个传输序列可以同时进行，由软件通过控制和状态寄存器进行监督。



DMA可以在每个时钟周期执行一次读访问和一次写访问，大小最多为32位。有12个独立的通道，每个通道监督一系列总线传输，通常在以下情况之一

* *内存到外设*：外设在需要传输更多数据时向DMA发出信号。DMA从RAM或闪存中的阵列读取数据，并写入外设的数据FIFO。
* *外设到存储器*：外设在收到数据时向DMA发送信号。DMA从外设的数据FIFO中读取该数据，并将其写入RAM中的数组
* *内存到内存*：DMA在RAM中的两个缓冲区之间尽可能快地传输数据

每个通道都有自己的控制和状态寄存器（CSR），软件可以使用这些寄存器对通道的进度进行编程和监控。当多个通道同时处于活动状态时，DMA在通道之间均匀共享带宽，并在当前请求数据传输的所有通道上进行循环

传输大小可以是32位、16位或8位。每个通道配置一次：源传输大小和目标传输大小相同。DMA对窄写入执行标准字节通道复制，因此字节数据在数据总线的所有4个字节中可用，半字数据在两个半字中可用。

渠道可以以各种方式组合，以实现更复杂的行为和更大的自主性。例如，一个通道可以配置另一个通道，从存储器中的一系列控制块加载配置数据，然后第二个通道可以在需要重新配置时通过CHAIN\_TO选项回调到第一个通道

使DMA更加自主意味着需要更少的处理器监督：总体而言，这允许系统一次做更多的事情，或者消耗更少的功率。

* + 1. 配置通道

每个通道有四个控制/状态寄存器：

* READ\_ADDR是指向下一个要读取的地址的指针
* WRITE\_ADDR是指向下一个要写入的地址的指针
* TRANS\_DATA显示当前传输序列中剩余的传输数，并用于编程下一个传输序列中的传输数（参见[2.5.1.2部分](#_bookmark165)）。
* CTRL用于配置通道行为的所有其他方面，以启用/禁用它，并检查完成。

这些是实时寄存器：它们随着通道的进展而不断更新

* + - 1. 读地址和写地址

READ\_ADDR和WRITE\_ADDR分别包含通道下一次读取和写入的地址这些寄存器在每次读/写访问后自动更新。它们每次递增1、2或4个字节，具体取决于CTRL中配置的传输大小。

通常，每当新的传输序列开始时，软件都应使用新的起始地址对这些寄存器进行如果READ\_ADDR和WRITE\_ADDR未重新编程，DMA将使用当前值作为下一次传输的起始地址。举例来说，请注意：

* 如果地址不递增（例如，它是外设FIFO的地址），并且下一个传输序列是去往/来自*同一*地址，则不需要再次写入寄存器
* 当向/从存储器中的一系列连续缓冲区传输时（例如， 分散和聚集），则在传送完成时，地址寄存器将已经递增到下一个缓冲器的开始。

通过不对每个传输序列的所有四个CSR进行编程，软件可以使用更短的中断处理程序，并在与通道链接一起使用时使用更紧凑的控制块格式（参见[第2.5.2.1节](#_bookmark167)中[的寄存器别名，第2.5.2.2节](#_bookmark168)中的链接）。

**谨慎**

READ\_ADDR和WRITE\_ADDR必须始终与CTRL.DATA\_SIZE中指定的当前传输大小对齐。软件负责确保初始值正确对齐。

* + - 1. 传输计数

从TRANS\_DATA读取数据会产生当前传输序列中剩余的传输数该值随着通道的进展而不断更新写入TRANS\_DATA设置*下*一个传输序列的长度在一个序列中最多可以执行*2个32 - 1\f25 transfer-1（*

每次通道启动新的传输序列时，写入TRANS\_DATA的最新值将复制到实时传输计数器，然后随着新的传输序列的进行，实时传输计数器将再次开始递减出于调试目的，最后写入的值可以从DBG\_TCR（TRANS\_RELOAD值）寄存器中读取

如果通道被触发多次而没有对TRANS\_DATA进行写入，则它每次执行相同数量的传输。例如，当链接到时，一个通道可能将固定大小的控制块加载到另一个通道的CSR中。TRANS\_DATA由软件编程一次，每次自动重新加载

或者，在开始每个传输序列之前，可以用新值写入TRANS\_DATA如果TRANS\_DATA是通道触发器（参见[第2.5.2.1](#_bookmark167)），则通道将立即启动，并且将使用刚写入的值，而*不是*重载寄存器中的当前值

**注意**

TRANS\_DATA是要执行的*传输*次数传输的总字节数为TRANS\_DATA

乘以每次传输的字节大小，由CTRL.DATA\_SIZE给出。

* + - 1. 控制/状态

CTRL寄存器比其他3个寄存器有更多更小的字段，这些字段的详细信息在CTRL寄存器列表中给出除其他外，CTRL用于：

* + - * + 通过CTRL.DATA\_SIZE配置此通道数据传输的大小。读和写的大小相同
        + 通过CTRL.INCR\_WRITE、CTRL.INCR\_READ、CTRL.RING\_SEL、CTRL.RING\_SIZE配置READ\_ADDR和WRITE\_ADDR在每次读取或写入后是否以及如何递增。环形传输是可用的，其中一个地址指针以一定的幂-

2边界。

* + - * + 通过CTRL.CHAIN\_TO选择要在此通道完成时触发的另一个通道（或无）。
        + 通过CTRL.TREQ\_SEL选择一个外设数据请求（DREQ）信号来调整此通道的传输。
        + 通过CTRL.BUSY查看通道何时空闲。
        + 查看通道是否遇到总线错误，例如：由于通过CTRL.AHB\_ERROR、CTRL.READ\_ERROR或CTRL.WRITE\_ERROR访问错误地址。
    1. 启动通道

有三种方法可以启动一个频道：

* 写入通道触发寄存器
* 来自另一个通道的链触发器，该通道刚刚完成，并配置了其CHAIN\_TO字段
* MULTI\_CHAN\_TRIGGER寄存器，可同时启动多个通道

每一个都涵盖了不同的用例。例如，在中断服务例程中配置和启动通道时，触发寄存器简单而有效

**注意**

触发一个已经在运行的通道没有任何效果。

* + - 1. 别名和触发器

表118.对妇女的歧视 控制寄存器别名。 每个通道有四个控制/状态寄存器。 每个寄存器可以在多个不同的地址访问。 在每个自然排列的四个组中，所有四个寄存器都以不同的顺序出现。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 偏移 | +0x0 | +0x4 | +0x8 | **+0xC**（触发） |
| 0x00（0） | READ\_ADDR | 写入地址 | 联系我们 | 控制触发 |
| 0x10（101） | Ctrl | READ\_ADDR | 写入地址 | 转换触发器 |
| 0x20（102） | Ctrl | 联系我们 | READ\_ADDR | 写入地址触发 |
| 0x30（103） | Ctrl | 写入地址 | 联系我们 | 读取\_添加\_触发 |

这四个CSR在内存中有多次别名。每个别名（共有四个别名）暴露相同的四个物理寄存器，但顺序不同。每个别名中的最后一个寄存器（偏移量为+0xC，突出显示）是触发寄存器。写入触发寄存器将启动通道。

通常，仅使用别名0，而别名1-3可以忽略。通过写入READ\_ADDR、WRITE\_ADDR、TRANS\_ADDR和最后CTRL来配置*和*启动通道。由于CTRL是别名0中的触发寄存器，因此这将启动通道。

当使用一个通道配置另一个通道时，其他别名允许更紧凑的控制块列表，并且在中断处理程序中更有效地重新配置和启动

* 每个CSR都是一个别名中的触发寄存器
  + 当将固定大小的缓冲区收集到外围设备中时，可以通过仅写入READ\_ADDR\_TRIG来配置和启动DMA通道。
  + 当从外围设备分散到固定大小的缓冲区时，可以通过仅写入WRITE\_ADDR\_TRIG来配置和启动通道。
* 寄存器的有用组合表现为包含触发寄存器的自然对齐元组。结合信道链接和地址包装，这些实现压缩控制块格式，例如：
  + （WRITE\_ADDR，TRANS\_ADDR\_TRIG）用于外围设备分散操作
  + （TRANS\_TRIG，READ\_ADDR\_TRIG）用于外围设备收集操作，或计算缓冲区列表上的CRC
  + （READ\_ADDR，WRITE\_ADDR\_TRIG）用于操作内存中的固定大小缓冲区触发寄存器在以下情况下不启动通道：
* 通过CTRL.EN禁用该通道。（如果触发器为CTRL，则使用刚写入的EN值，而*不是*CTRL寄存器中的当前值
* 频道已在运行
* 值0写入触发寄存器。（这对于结束控制块链很有用。参见空触发器，[第2.5.2.3](#_bookmark169)）
  + - 1. 链接

当通道完成时，它可以命名一个不同的通道以立即触发。这可以用作第二个通道的回调，以重新配置和重新启动第一个通道。

此功能通过通道CTRL寄存器中的CHAIN\_TO字段配置此4位值选择将在此通道结束时启动的通道通道不能链接到自身。将CHAIN\_TO设置为通道自己的索引意味着不会发生链接。

链式触发器的行为与来自其他源（如触发器寄存器）的触发器相同例如，它们导致

如果目标通道已经在运行，则忽略它们

CHAIN\_TO的一个应用是一个通道从存储器中的一系列控制块请求另一个通道重新配置通道A被配置为执行从存储器到通道B的控制寄存器（包括触发寄存器）的包装传输这在DMA控制块示例中更明确地显示（[第2.5.6.2](#_bookmark174)）。

使用寄存器别名（[第2.5.2.1节](#_bookmark167)）可实现DMA控制块的紧凑格式：在某些情况下只有一个字。

链接的另一种用途是“乒乓”配置，其中两个通道各自触发彼此。处理器可以响应通道完成中断，并在完成后重新配置每个通道;但是，已经配置的链接通道会立即启动。换句话说，通道配置和通道操作是流水线式的。在需要许多短传输序列的情况下，性能可以显著提高

第[2.5.6节](#_bookmark173)更详细地介绍了现实世界中链式触发器的可能性

* + - 1. 触发器和链中断

如[2.5.2.1](#_bookmark167)部分所述，将全零写入触发寄存器*不会*启动通道。这被称为空触发器，它有两个用途：

* 通过附加一个全零块，在控制块数组的末尾引起暂停
* 减少使用控制块时产生的中断数

默认情况下，通道将在每次完成传输序列时生成中断，除非该通道的IRQ在INTE0或INTE1中被屏蔽。中断率可能过高，特别是当一系列控制块正在进行时，通常不需要处理器注意;然而，在链的末端需要处理器注意。

通道CTRL寄存器有一个名为IRQ\_QUIET的字段。其默认值为0。当此设置为1时，通道在接收到空触发时生成中断，其他时间不生成中断。中断由接收触发的通道产生

* + 1. 数据请求（DREQ）

外围设备以自己的速度生产或消费数据如果DMA只是尽可能快地传输数据，那么数据的丢失或损坏将随之发生。DREQ是外设和DMA之间的通信通道，它使DMA能够根据外设的需要调整传输

CTRL.TREQ\_SEL（传输请求）字段选择外部DREQ。它还可用于选择内部起搏定时器之一，或根本不选择TREQ（传输尽可能快地进行），例如：用于存储器到存储器的传输。

* + - 1. 系统DREQ表

表119.DREQs

DREQ编号可全局分配给外围DREQ通道。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| DREQ | DREQ通道 | DREQ | DREQ通道 | DREQ | DREQ通道 | DREQ | DREQ通道 |
| 0 | DREQ\_PIO0\_TX0 | 10 | DREQ\_PIO1\_TX2 | 20 | DREQ\_UART0\_TX | 30 | DREQ\_PWM\_WRAP 6 |
| 1 | DREQ\_PIO0\_TX1 | 11 | DREQ\_PIO1\_TX3 | 21 | DREQ\_UART0\_RX | 31 | DREQ\_PWM\_WRAP 7 |
| 2 | DREQ\_PIO0\_TX2 | 12 | DREQ\_PIO1\_RX0 | 22 | DREQ\_UART1\_TX | 32 | DREQ\_I2C0\_TX |
| 3 | DREQ\_PIO0\_TX3 | 13 | DREQ\_PIO1\_RX1 | 23 | DREQ\_UART1\_RX | 33 | DREQ\_I2C0\_RX |
| 4 | DREQ\_PIO0\_RX0 | 14 | DREQ\_PIO1\_RX2 | 24 | DREQ\_PWM\_WRAP0 | 34 | DREQ\_I2C1\_TX |
| 5 | DREQ\_PIO0\_RX1 | 15 | DREQ\_PIO1\_RX3 | 25 | DREQ\_PWM\_WRAP 1 | 35 | DREQ\_I2C1\_RX |
| 6 | DREQ\_PIO0\_RX2 | 16 | DREQ\_SPI0\_TX | 26 | DREQ\_PWM\_WRAP 2 | 36 | DREQ\_ADC |
| 7 | DREQ\_PIO0\_RX3 | 17 | DREQ\_SPI0\_RX | 27 | DREQ\_PWM\_WRAP 3 | 37 | DREQ\_XIP\_STREAM |
| 8 | DREQ\_PIO1\_TX0 | 18 | DREQ\_SPI1\_TX | 28 | DREQ\_PWM\_WRAP 4 | 38 | DREQ\_XIP\_SSITX |
| 9 | DREQ\_PIO1\_TX1 | 19 | DREQ\_SPI1\_RX | 29 | DREQ\_PWM\_WRAP 5 | 39 | DREQ\_XIP\_SSIRX |

* + - 1. 基于信用的DREQ计划

RP 2040 DMA专为以下系统设计

* 大型外围数据FIFO的面积和功率成本是禁止的
* 单个外围设备的带宽需求可能很高，例如，>50%总线注入率，用于短时间
* 总线延迟较低，但多个主机可能会竞争总线访问

此外，DMA的传输FIFO和双主机结构允许对同一外围设备的多个访问同时进行，以提高总吞吐量。因此，DREQ机制的选择至关重要：

* 如果在TDF中备份多个写入，则传统的“打开抽头”方法可能会导致溢出一些系统通过过度配置外围FIFO并将DREQ阈值设置为低于满电平来解决此问题，但

这浪费了宝贵的面积和功率

* ARM风格的单次和突发握手不允许在当前请求被服务的同时注册额外的请求当FIFO非常浅时，这限制了性能

RP 2040 DMA使用基于信用的DREQ机制。对于每个外围设备，DMA会尝试保持与外围设备容量一样多的传输。这样，在没有结构延迟或争用的情况下，通过8深外设FIFO实现全总线吞吐量（每时钟1个字），不可能出现上溢或下溢

对于每个通道，DMA维护一个计数器。dreq信号上的每个1时钟脉冲将递增此计数器（饱和）。当非零时，通道向DMA的内部仲裁器请求传输，并且当传输被发送到地址FIFO时，计数器递减。此时，转移正在进行中，但尚未完成。

*图13. DREQ计数*

德雷格

更改计数

禅问题

0

1

0

1 1 2

其效果是根据外围FIFO中可用的空间或数据量来上限处理中传输的数量。 在稳定状态下，这给出了最大吞吐量，但不能下溢或下溢。

一个警告是，用户*不得*访问当前由DMA提供服务的FIFO这使得

通道和外围设备被破坏，并可能导致数据损坏或丢失另一个警告是，多个通道不应连接到同一个DREQ。

* + 1. 中断

每个通道都可以生成中断;可以使用INTE 0或INTE 1寄存器对每个通道进行屏蔽。在两种情况下，通道会发出中断请求：

* 每个传输序列完成后，如果禁用CTRL.IRQ\_QUIET
* 如果CTRL.IRQ\_QUIET使能，则在接收到空触发时

屏蔽中断状态在INTS寄存器中可见;每个通道对应一位通过向INTS写入位掩码来清除中断确认中断的一个习惯用法是读取INTS，然后写回相同的值，因此只有启用的中断被清除。

RP 2040 DMA提供两个系统IRQ，具有独立的屏蔽和状态寄存器（例如INTE0、INTE1）。任何通道中断请求的组合都可以路由到任一系统IRQ。举例来说

* 如果某些通道具有特别紧的时序要求，则在系统中断控制器中可以给予它们更高的优先级
* 在多处理器系统中，不同的通道中断可以独立地路由到不同的内核。为了调试，INTF寄存器可以强制断言任一IRQ。
  + 1. 附加特征
       1. 起搏定时器

这些允许大约每*n*个时钟周期传输一次数据，而不是使用外部外设DREQ来触发传输。使用小数（X/Y）分频器，并且将在每个SYS\_SYS周期生成最多1个请求

RP 2040中有4个定时器可用 每个DMA都可以在CTRL.TREQ\_SEL中选择其中任何一个。

* + - 1. CRC计算

DMA可以监视来自给定通道的数据通过数据FIFO，并基于此数据计算校验和。这是一个纯粹被动的事情：数据不会被这个硬件改变，只是观察到。

该功能通过SNIFF\_CTRL和SNIFF\_DATA寄存器进行控制，并可通过CTRL.SNIFF\_EN字段在每次DMA传输时启用/禁用

由于该硬件不能对FIFO施加反压，因此它必须跟上DMA每时钟32位的最大传输速率

支持的校验和包括：

* CRC-32，MSB优先和LSB优先
* CRC-16-CCITT，MSB优先和LSB优先
* 简单求和（加到32位累加器）
* 偶校验

结果寄存器既可读又可写，因此可以设置初始种子值。位/字节操作可用于结果，这可能有助于特定用例：

* 比特反转
* 位反转
* 字节交换

这些操作不会影响CRC计算，只会影响数据在结果寄存器中的显示

* + - 1. 通道中止

通道可能会进入不可恢复的状态：例如。如果被命令传输的数据比外围设备所请求的要多清除CTRL.EN位只会暂停通道，并不能解决问题。正常情况下不应发生这种情况，但重要的是，有一种机制可以恢复，而无需简单地硬重置整个DMA块。

[CHAN\_ABORT](#_bookmark205)寄存器强制通道提前完成。每个通道有一个位，写入1会终止该通道。这将清除传输计数器并强制通道进入非活动状态。

##### 谨慎

由于RP 2040-E13，中止正在进行的DMA通道（即未在非活动DREQ上停止）可能会导致完成IRQ断言。在执行中止前应清除通道中断使能，中止后应检查并清除中断

在中止被触发时，通道可能具有当前在读主机和写主机之间进行的总线传输，并且这些传输不能被撤销。CTRL.BUSY标志保持高电平，直到这些传输完成，并且通道达到安全状态，这通常只需要几个周期。通道必须在其CTRL.BUSY标志取消置位后才能重新启动在旧序列的传输仍在进行中时开始新的传输序列可能会导致不可预测的行为。

* + - 1. 调试

每个DMA通道都有寄存器可用于显示dreq计数器DBG\_CTDREQ和下一个传输计数DBG\_TCR。如果需要，这些也可用于重置DMA通道。

* + 1. 示例用例
       1. 使用中断重新配置通道

当一个通道完成一个传输块时，它就可以进行更多的传输。软件检测到通道不再繁忙，并重新配置和重启通道。一种方法是轮询CTRL\_BUSY位，直到通道完成，但这失去了DMA的一个关键优势，即它*不必*与处理器同步操作。通过设置INTE0或INTE1中的正确位，我们可以指示DMA在给定通道完成时引发其两条中断请求线而不是反复询问一个频道是否完成，我们被告知。

##### 注意

有两个系统中断线允许不同的通道完成中断被路由到不同的内核，或者如果一个通道对时间要求更高，则在同一个内核上彼此抢占

当中断被断言时，处理器可以被配置为放弃它正在做的任何事情，并调用用户指定的处理程序函数。处理程序可以重新配置和重新启动通道。当处理程序退出时，处理器返回到在前台运行的中断代码

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/dma/channel\_irq/channel\_irq.c](https://github.com/raspberrypi/pico-examples/blob/master/dma/channel_irq/channel_irq.c#L35-L52)第35-52

publicvoid println（）{

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52}

intmaximum =0;

静态uint32\_twavetable[N\_PWM\_LEVELS];静态bool first\_run = true;

//条目号`i`有`i` 1位和`（32-i）` 0位。

if（first\_run）{ first\_run = false;

for（ inti =0; i N\_PWM\_LEVELS;++i）wavetable[i] = ~（~0u<<i）;

}

//清除中断请求。

dma\_hw->ints0 =1udma\_chan;

//给通道一个新的波表条目来读取，并重新触发它

dma\_channel\_set\_read\_addr（dma\_chan，wavetable[pwm\_level]，true）;

pwm\_level =（pwm\_level+1）%N\_PWM\_LEVELS;

在许多情况下，大多数配置可以在通道第一次启动时完成，只有地址和传输长度需要在DMA处理程序中重新编程。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/dma/channel\_irq/channel\_irq.c](https://github.com/raspberrypi/pico-examples/blob/master/dma/channel_irq/channel_irq.c#L54-L94)第54-94

1. public voidrun（）{
2. #ifndefPICO\_DEFAULT\_LED\_PIN
3. #warning dma/channel\_irq example需要一个带常规LED的主板
4. 其他#

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

//设置PIO状态机来序列化我们的位

uint offset = pio\_add\_program（pio0，pio\_serialiser\_program）; pio\_serialiser\_program\_init（pio0，0，offset，PICO\_DEFAULT\_LED\_PIN，PIO\_SERIAL\_CLKDIV）;

//配置一个通道，将同一个字（32位）重复写入PIO 0

//SM 0的TX FIFO，由来自该外设的数据请求信号进行调整。

dma\_chan =dma\_claim\_unused\_channel（true）;

dma\_channel\_config c = dma\_channel\_get\_default\_config（dma\_chan）; channel\_config\_set\_transfer\_data\_size（c，DMA\_SIZE\_32）; channel\_config\_set\_read\_increment（c，false）; channel\_config\_set\_dreq（c，DREQ\_PIO0\_TX0）;

DMA\_channel\_configure（DMA\_chan，

&c、

&pio0\_hw->txf[0]，*//写地址（只需要设置一次）*

空值，

//不提供读地址

PWM\_REPEAT\_REQ，*//多次写入相同的值，然后停止并中断*

假

//不要开始

);

//当通道完成一个块时，告诉DMA提升IRQ线0

dma\_channel\_set\_irq0\_enabled（dma\_chan，true）;

*//配置处理器在DMA IRQ 0被断言时*运行dma\_handler（）irq\_set\_exclusive\_handler（DMA\_IRQ\_0，dma\_handler）; irq\_set\_enabled（DMA\_IRQ\_0，true）;

//手动调用handler一次，触发第一次转账

mysql（）;

//从这一点开始，其他所有内容都是中断驱动的。处理器具有

90

91

92

//是时候坐下来考虑一下它的提前退休了--也许开一家面包店？

while（true）

return（）;

1. #结束
2. }

这种技术的一个缺点是，直到通道进行最后一次传输之后的一段时间，我们才开始重新配置通道如果处理器上有大量的中断活动，这可能是相当长的时间，因此在传输中有相当大的间隙，如果我们需要维持高数据吞吐量，这是有问题的

这可以通过使用两个通道来解决，它们的CHAIN\_TO字段交叉，以便通道A在完成时触发通道B，反之亦然。 在任何时间点，其中一个通道正在传输数据，另一个通道已经配置为在当前通道完成时立即开始下一个传输，或者正在重新配置。 当通道A完成时，它立即开始通道B上的提示传输。与此同时，中断被触发，处理程序重新配置通道A，以便在通道B完成时做好准备

* + - 1. DMA控制块

通常，多个较小的缓冲区必须聚集在一起并发送到同一个外围设备。为了解决这一用例，RP 2040 DMA可以在没有处理器控制的情况下执行长而复杂的传输*序列*。一个通道重复重新配置第二个通道，第二个通道在每次完成传输块时重新启动第一个通道。

由于第一个DMA通道直接将数据从存储器传输到第二个通道的控制寄存器，因此存储器中控制块的格式必须与这些寄存器匹配。每次写入的最后一个寄存器将是其中一个触发寄存器（[第2.5.2.1节](#_bookmark167)），它将启动其编程传输块上的第二个通道。寄存器别名（[第2.5.2.1](#_bookmark167)）为块布局提供了一些灵活性，更重要的是允许从块中省略一些寄存器，因此它们占用的内存更少，加载速度更快。

此示例显示如何通过重新编程TRANS\_PROGRAM来收集多个缓冲区并将其传输到PROGRAM

READ\_ADDR\_TRIG：

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

1*/\*\**

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

6

1. //使用两个DMA通道进行编程的数据传输序列，
2. //数据收集操作（data一个通道负责传输
3. //实际数据，另一个重复重新编程该通道。

10

11*#include<stdio.h>*

12*#包含“pico/stdlib.h”*

13*#include“hardware/dma.h”*

14*#include“hardware/structs/uart.h”*

15

16*//这些缓冲区将一个接一个地DMA到CPU。*

17

18const char word 0 []=“正在传输“;

19const char word1[]=“one“;

20const char word2[]=“word“;

21const char word3[]=“at“;

22const char word4[]=“a“;

23const char word5[]=“时间。\ n”; 24

25*//注意这里字段的顺序重要的*是长度*在前面*

1. //读地址，因为控制通道将写入最后一个
2. //数据通道上别名3中的两个寄存器

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 28*//* |  | +0x0 | +0x4 | +0x8 | +0xC（触发） |
| 29*//* | 00： | READ\_ADDR | 写入地址 | 联系我们 | Ctrl |
| 30*//* | 第1章： | Ctrl | READ\_ADDR | 写入地址 | 联系我们 |
| 31*//* | 第二： | Ctrl | 联系我们 | READ\_ADDR | 写入地址 |
| 32*//* | 第三章： | Ctrl | 写入地址 | 联系我们 | READ\_ADDR |
| 33*//* |  |  |  |  |  |

1. //这将编程数据通道的传输计数和读取地址
2. //并触发它。 一旦数据通道完成，它将重新启动
3. //控制通道（通过CHAIN\_TO）将下两个字加载到其控制中
4. //注册表。

38

1. const struct{ uint32\_tlen; const char\*data;} control\_blocks[]={
2. {count\_of（word 0）-1，word 0}，*//跳过空终止符*
3. {count\_of（word1）-1，word1}，
4. {count\_of（word2）-1，word2}，
5. {count\_of（word3）-1，word3}，
6. {count\_of（word4）-1，word4}，
7. {count\_of（word5）-1，word5}，
8. {0，NULL} *//触发器结束链。*

47};

48

1. public voidrun（）{
2. #ifndefuart\_default
3. #warning dma/control\_blocks example requires atoken
4. 其他#
5. stdio\_init\_all（）;
6. puts（“DMA control block example：“）;55
7. //clog\_chan将控制块加载到data\_chan中，由data\_chan执行它们。
8. intcnc\_chan =dma\_claim\_unused\_channel（true）;
9. intdata\_chan = dma\_claim\_unused\_channel（true）;59
10. //控制通道将两个字传输到数据通道的控制
11. //注册，然后停止。写入地址在两个字上换行
12. //（八字节）边界，以便控制通道写入相同的两个
13. //当下一次触发时进行注册。

64

1. dma\_channel\_config c =dma\_channel\_get\_default\_config（codeword\_chan）;
2. channel\_config\_set\_transfer\_data\_size（c，DMA\_SIZE\_32）;
3. channel\_config\_set\_read\_increment（c，true）;
4. channel\_config\_set\_write\_increment（c，true）;
5. channel\_config\_set\_ring（c，true，3）;*//1 3字节边界上写ptr*

70

1. dma\_channel\_configure（
2. chan\_chan，
3. &c、
4. &dma\_hw->ch[data\_chan].al3\_transfer\_count，*//初始写地址*
5. &control\_blocks[0]， *//初始读地址*
6. 2， *//在每个控制块后停止*
7. false *//还没有开始*

78）;

79

1. //数据通道设置为写入FIFO（由
2. //发送器的TX数据请求信号），然后链接到控制信道
3. //一旦完成。控制通道编程新的读地址，
4. //数据长度，并重新触发数据通道。

84

1. dma\_channel\_get\_default\_config（data\_chan）;
2. channel\_config\_set\_transfer\_data\_size（c，DMA\_SIZE\_8）;
3. channel\_config\_set\_dreq（c，uart\_get\_dreq（uart\_default，true））;
4. //当data\_chan完成时触发clog\_chan
5. channel\_config\_set\_chain\_to（c，chain\_chan）;

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

//当0被写入触发寄存器时，提升IRQ标志（链结束）：

channel\_config\_set\_irq\_quiet（c，true）;

DMA\_channel\_configure（data\_chan，

&c、

&uart\_get\_hw（uart\_default）->dr，

NULL，0，

假

//初始读取地址和传输计数不重要;

//控制通道每次都会重新编程它们。

//现在不要开始。

);

一切都准备好了。告诉控制频道加载第一个

//控制块。从这里开始一切都是自动的。

dma\_start\_channel\_mask（1ucodeword\_chan）;

//数据通道将在获得空触发器时断言其IRQ标志

//表示控制块列表的结束我们就等着

//设置IRQ标志，而不是设置中断处理程序。

同时（！(dma\_hw->intr&1u <<data\_chan））tight\_loop\_contents（）;

dma\_hw->ints0 =1udata\_chan;

puts（“DMA完成。");

1. #结束
2. }
   * 1. 登记册一览表

DMA寄存器从基址0x50000000（在SDK中定义为[DMA\_BASE](#_bookmark40)）开始。

*表120. DMA寄存器列表*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x000 | [CH0\_READ\_ADDR](#_bookmark177) | DMA通道0读地址指针 |
| 0x004 | [CH0\_WRITE\_ADDR](#_bookmark178) | DMA通道0写地址指针 |
| 0x008 | [CH0\_TRANS\_CH2](#_bookmark179) | DMA通道0传输计数 |
| 0x00c | [CH0\_CTRL\_TRIG](#_bookmark180) | DMA通道0控制和状态 |
| 0x010 | [CH0\_AL1\_CTRL](#_bookmark181) | 通道0 CTRL寄存器的复位 |
| 0x014 | [CH0\_AL1\_READ\_ADDR](#_bookmark182) | 通道0 READ\_ADDR寄存器的寄存器 |
| 0x018 | [CH0\_AL1\_WRITE\_ADDR](#_bookmark183) | 通道0 WRITE\_ADDR寄存器的寄存器 |
| 0x01c | [CH0\_AL1\_TRANS\_COUNT\_TRIG](#_bookmark184) | 通道0 TRANS\_DATA寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x020 | [CH0\_AL2\_CTRL](#_bookmark185) | 通道0 CTRL寄存器的复位 |
| 0x024 | [CH0\_AL2\_TRANS\_CRYSTAL](#_bookmark186) | 通道0 TRANS\_DATA寄存器 |
| 0x028 | [CH0\_AL2\_READ\_ADDR](#_bookmark187) | 通道0 READ\_ADDR寄存器的寄存器 |
| 0x02c | [CH0\_AL2\_WRITE\_ADDR\_TRIG](#_bookmark188) | 通道0 WRITE\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x030 | [CH0\_AL3\_CTRL](#_bookmark189) | 通道0 CTRL寄存器的复位 |
| 0x034 | [CH0\_AL3\_WRITE\_ADDR](#_bookmark190) | 通道0 WRITE\_ADDR寄存器的寄存器 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x038 | [CH0\_AL3\_TRANS\_CRYSTAL](#_bookmark191) | 通道0 TRANS\_DATA寄存器 |
| 0x03c | [CH0\_AL3\_READ\_ADDR\_TRIG](#_bookmark192) | 通道0 READ\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x040 | [CH1\_READ\_ADDR](#_bookmark177) | DMA通道1读地址指针 |
| 0x044 | [CH1\_WRITE\_ADDR](#_bookmark178) | DMA通道1写地址指针 |
| 0x048 | [CH1\_TRANS\_CH1](#_bookmark179) | DMA通道1传输计数 |
| 0x04c | [CH1\_CTRL\_触发](#_bookmark180) | DMA通道1控制和状态 |
| 0x 050 | [CH1\_AL1\_CTRL](#_bookmark181) | 通道1 CTRL寄存器的复位 |
| 0x054 | [CH1\_AL1\_READ\_ADDR](#_bookmark182) | 通道1 READ\_ADDR寄存器的寄存器 |
| 0x058 | [CH1\_AL1\_WRITE\_ADDR](#_bookmark183) | 通道1 WRITE\_ADDR寄存器的寄存器 |
| 0x05c | [CH1\_AL1\_TRANS\_COUNT\_TRIG](#_bookmark184) | 通道1 TRANS\_DATA寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x060 | [CH1\_AL2\_CTRL](#_bookmark185) | 通道1 CTRL寄存器的复位 |
| 0x064 | [CH1\_AL2\_TRANS\_CRYSTAL](#_bookmark186) | 通道1 TRANS\_DATA寄存器 |
| 0x068 | [CH1\_AL2\_READ\_ADDR](#_bookmark187) | 通道1 READ\_ADDR寄存器的寄存器 |
| 0x06c | [CH1\_AL2\_WRITE\_ADDR\_TRIG](#_bookmark188) | 通道1 WRITE\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x070 | [CH1\_AL3\_CTRL](#_bookmark189) | 通道1 CTRL寄存器的复位 |
| 0x074 | [CH1\_AL3\_WRITE\_ADDR](#_bookmark190) | 通道1 WRITE\_ADDR寄存器的寄存器 |
| 0x078 | [CH1\_AL3\_TRANS\_CRYSTAL](#_bookmark191) | 通道1 TRANS\_DATA寄存器 |
| 0x07c | [CH1\_AL3\_READ\_ADDR\_TRIG](#_bookmark192) | 通道1 READ\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x080 | [CH2\_READ\_ADDR](#_bookmark177) | DMA通道2读地址指针 |
| 0x084 | [CH2\_WRITE\_ADDR](#_bookmark178) | DMA通道2写地址指针 |
| 0x088 | [CH2\_TRANS\_CH2](#_bookmark179) | DMA通道2传输计数 |
| 0x08c | [CH2\_CTRL\_触发](#_bookmark180) | DMA通道2控制和状态 |
| 0x090 | [CH2\_AL1\_CTRL](#_bookmark181) | 通道2 CTRL寄存器的复位 |
| 0x094 | [CH2\_AL1\_READ\_ADDR](#_bookmark182) | 通道2 READ\_ADDR寄存器的寄存器 |
| 0x098 | [CH2\_AL1\_WRITE\_ADDR](#_bookmark183) | 通道2 WRITE\_ADDR寄存器的别名 |
| 0x09c | [CH2\_AL1\_TRANS\_COUNT\_TRIG](#_bookmark184) | 通道2 TRANS\_DATA寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x0a0 | [CH2\_AL2\_CTRL](#_bookmark185) | 通道2 CTRL寄存器的复位 |
| 0x0a4 | [CH2\_AL2\_TRANS\_CH2](#_bookmark186) | 通道2 TRANS\_DATA寄存器 |
| 0x0a8 | [CH2\_AL2\_READ\_ADDR](#_bookmark187) | 通道2 READ\_ADDR寄存器的寄存器 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x0ac | [CH2\_AL2\_WRITE\_ADDR\_TRIG](#_bookmark188) | 通道2 WRITE\_ADDR寄存器的别名  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x0b0 | [CH2\_AL3\_CTRL](#_bookmark189) | 通道2 CTRL寄存器的复位 |
| 0x0b4 | [CH2\_AL3\_WRITE\_ADDR](#_bookmark190) | 通道2 WRITE\_ADDR寄存器的寄存器 |
| 0x0b8 | [CH2\_AL3\_TRANS\_CH2](#_bookmark191) | 通道2 TRANS\_DATA寄存器 |
| 0x0bc | [CH2\_AL3\_READ\_ADDR\_TRIG](#_bookmark192) | 通道2 READ\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x0c0 | [CH3\_READ\_ADDR](#_bookmark177) | DMA通道3读地址指针 |
| 0x0c4 | [CH3\_WRITE\_ADDR](#_bookmark178) | DMA通道3写地址指针 |
| 0x0c8 | [CH3\_TRANS\_CH3](#_bookmark179) | DMA通道3传输计数 |
| 0x0cc | [CH3\_CTRL\_触发](#_bookmark180) | DMA通道3控制和状态 |
| 0x0d0 | [CH3\_AL1\_CTRL](#_bookmark181) | 通道3 CTRL寄存器的复位 |
| 0x0d4 | [CH3\_AL1\_READ\_ADDR](#_bookmark182) | 通道3 READ\_ADDR寄存器的寄存器 |
| 0x0d8 | [CH3\_AL1\_WRITE\_ADDR](#_bookmark183) | 通道3 WRITE\_ADDR寄存器的寄存器 |
| 0x0dc | [CH3\_AL1\_TRANS\_COUNT\_TRIG](#_bookmark184) | 通道3 TRANS\_DATA寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x0e0 | [CH3\_AL2\_CTRL](#_bookmark185) | 通道3 CTRL寄存器的复位 |
| 0x0e4 | [CH3\_AL2\_TRANS\_CH3](#_bookmark186) | 通道3 TRANS\_DATA寄存器 |
| 0x0e8 | [CH3\_AL2\_READ\_ADDR](#_bookmark187) | 通道3 READ\_ADDR寄存器的寄存器 |
| 0x0ec | [CH3\_AL2\_WRITE\_ADDR\_TRIG](#_bookmark188) | 通道3 WRITE\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x0f0 | [CH3\_AL3\_CTRL](#_bookmark189) | 通道3 CTRL寄存器的复位 |
| 0x0f4 | [CH3\_AL3\_WRITE\_ADDR](#_bookmark190) | 通道3 WRITE\_ADDR寄存器的寄存器 |
| 0x0f8 | [CH3\_AL3\_TRANS\_CH3](#_bookmark191) | 通道3 TRANS\_DATA寄存器 |
| 0x0fc | [CH3\_AL3\_READ\_ADDR\_TRIG](#_bookmark192) | 通道3 READ\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x100 | [CH4\_READ\_ADDR](#_bookmark177) | DMA通道4读地址指针 |
| 0x104 | [CH4\_WRITE\_ADDR](#_bookmark178) | DMA通道4写地址指针 |
| 0x108 | [CH4\_TRANS\_CH4](#_bookmark179) | DMA通道4传输计数 |
| 0x10c | [CH4\_CTRL\_触发](#_bookmark180) | DMA通道4控制和状态 |
| 0x110 | [CH4\_AL1\_CTRL](#_bookmark181) | 通道4 CTRL寄存器的复位 |
| 0x114 | [CH4\_AL1\_READ\_ADDR](#_bookmark182) | 通道4 READ\_ADDR寄存器的寄存器 |
| 0x118 | [CH4\_AL1\_WRITE\_ADDR](#_bookmark183) | 通道4 WRITE\_ADDR寄存器的寄存器 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x11c | [CH4\_AL1\_TRANS\_COUNT\_TRIG](#_bookmark184) | 通道4 TRANS\_DATA寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x120 | [CH4\_AL2\_CTRL](#_bookmark185) | 通道4 CTRL寄存器的复位 |
| 0x124 | [CH4\_AL2\_TRANS\_CH2](#_bookmark186) | 通道4 TRANS\_DATA寄存器 |
| 0x128 | [CH4\_AL2\_READ\_ADDR](#_bookmark187) | 通道4 READ\_ADDR寄存器的寄存器 |
| 0x12c | [CH4\_AL2\_WRITE\_ADDR\_TRIG](#_bookmark188) | 通道4 WRITE\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x130 | [CH4\_AL3\_CTRL](#_bookmark189) | 通道4 CTRL寄存器的复位 |
| 0x134 | [CH4\_AL3\_WRITE\_ADDR](#_bookmark190) | 通道4 WRITE\_ADDR寄存器的别名 |
| 0x138 | [CH4\_AL3\_TRANS\_CRYSTAL](#_bookmark191) | 通道4 TRANS\_DATA寄存器 |
| 0x13c | [CH4\_AL3\_READ\_ADDR\_TRIG](#_bookmark192) | 通道4 READ\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x140 | [CH5\_READ\_ADDR](#_bookmark177) | DMA通道5读地址指针 |
| 0x144 | [CH5\_WRITE\_ADDR](#_bookmark178) | DMA通道5写地址指针 |
| 0x148 | [CH5\_TRANS\_CH5](#_bookmark179) | DMA通道5传输计数 |
| 0x14c | [CH5\_CTRL\_触发](#_bookmark180) | DMA通道5控制和状态 |
| 0x150 | [CH5\_AL1\_CTRL](#_bookmark181) | 通道5 CTRL寄存器的复位 |
| 0x154 | [CH5\_AL1\_READ\_ADDR](#_bookmark182) | 通道5 READ\_ADDR寄存器的寄存器 |
| 0x158 | [CH5\_AL1\_WRITE\_ADDR](#_bookmark183) | 通道5 WRITE\_ADDR寄存器的别名 |
| 0x15c | [CH5\_AL1\_TRANS\_COUNT\_TRIG](#_bookmark184) | 通道5 TRANS\_DATA寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x160 | [CH5\_AL2\_CTRL](#_bookmark185) | 通道5 CTRL寄存器的复位 |
| 0x164 | [CH5\_AL2\_TRANS\_CRYSTAL](#_bookmark186) | 通道5 TRANS\_DATA寄存器 |
| 0x168 | [CH5\_AL2\_READ\_ADDR](#_bookmark187) | 通道5 READ\_ADDR寄存器的寄存器 |
| 0x16c | [CH5\_AL2\_WRITE\_ADDR\_TRIG](#_bookmark188) | 通道5 WRITE\_ADDR寄存器的别名  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x170 | [CH5\_AL3\_CTRL](#_bookmark189) | 通道5 CTRL寄存器的复位 |
| 0x174 | [CH5\_AL3\_WRITE\_ADDR](#_bookmark190) | 通道5 WRITE\_ADDR寄存器的别名 |
| 0x178 | [CH5\_AL3\_TRANS\_CRYSTAL](#_bookmark191) | 通道5 TRANS\_DATA寄存器 |
| 0x17c | [CH5\_AL3\_READ\_ADDR\_TRIG](#_bookmark192) | 通道5 READ\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x180 | [CH6\_READ\_ADDR](#_bookmark177) | DMA通道6读地址指针 |
| 0x184 | [CH6\_WRITE\_ADDR](#_bookmark178) | DMA通道6写地址指针 |
| 0x188 | [CH6\_TRANS\_CH2](#_bookmark179) | DMA通道6传输计数 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x18c | [CH6\_CTRL\_触发](#_bookmark180) | DMA通道6控制和状态 |
| 0x190 | [CH6\_AL1\_CTRL](#_bookmark181) | 通道6 CTRL寄存器的复位 |
| 0x194 | [CH6\_AL1\_READ\_ADDR](#_bookmark182) | 通道6 READ\_ADDR寄存器的寄存器 |
| 0x198 | [CH6\_AL1\_WRITE\_ADDR](#_bookmark183) | 通道6 WRITE\_ADDR寄存器的寄存器 |
| 0x19c | [CH6\_AL1\_TRANS\_COUNT\_TRIG](#_bookmark184) | 通道6 TRANS\_DATA寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x1a0 | [CH6\_AL2\_CTRL](#_bookmark185) | 通道6 CTRL寄存器的复位 |
| 0x1a4 | [CH6\_AL2\_TRANS\_CRYSTAL](#_bookmark186) | 通道6 TRANS\_DATA寄存器 |
| 0x1a8 | [CH6\_AL2\_READ\_ADDR](#_bookmark187) | 通道6 READ\_ADDR寄存器的寄存器 |
| 0x1ac | [CH6\_AL2\_WRITE\_ADDR\_TRIG](#_bookmark188) | 通道6 WRITE\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x1b0 | [CH6\_AL3\_CTRL](#_bookmark189) | 通道6 CTRL寄存器的复位 |
| 0x1b4 | [CH6\_AL3\_WRITE\_ADDR](#_bookmark190) | 通道6 WRITE\_ADDR寄存器的寄存器 |
| 0x1b8 | [CH6\_AL3\_TRANS\_CRYSTAL](#_bookmark191) | 通道6 TRANS\_DATA寄存器 |
| 0x1bc | [CH6\_AL3\_READ\_ADDR\_TRIG](#_bookmark192) | 通道6 READ\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x1c0 | [CH7\_READ\_ADDR](#_bookmark177) | DMA通道7读地址指针 |
| 0x1c4 | [CH7\_WRITE\_ADDR](#_bookmark178) | DMA通道7写地址指针 |
| 0x1c8 | [CH7\_TRANS\_CH2](#_bookmark179) | DMA通道7传输计数 |
| 0x1cc | [CH7\_CTRL\_触发](#_bookmark180) | DMA通道7控制和状态 |
| 0x1d0 | [CH7\_AL1\_CTRL](#_bookmark181) | 通道7 CTRL寄存器的复位 |
| 0x1d4 | [CH7\_AL1\_READ\_ADDR](#_bookmark182) | 通道7 READ\_ADDR寄存器的寄存器 |
| 0x1d8 | [CH7\_AL1\_WRITE\_ADDR](#_bookmark183) | 通道7 WRITE\_ADDR寄存器的寄存器 |
| 0x1dc | [CH7\_AL1\_TRANS\_COUNT\_TRIG](#_bookmark184) | 通道7 TRANS\_DATA寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x1e0 | [CH7\_AL2\_CTRL](#_bookmark185) | 通道7 CTRL寄存器的复位 |
| 0x1e4 | [CH7\_AL2\_TRANS\_CRYSTAL](#_bookmark186) | 通道7 TRANS\_DATA寄存器 |
| 0x1e8 | [CH7\_AL2\_READ\_ADDR](#_bookmark187) | 通道7 READ\_ADDR寄存器的寄存器 |
| 0x1ec | [CH7\_AL2\_WRITE\_ADDR\_TRIG](#_bookmark188) | 通道7 WRITE\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x1f0 | [CH7\_AL3\_CTRL](#_bookmark189) | 通道7 CTRL寄存器的复位 |
| 0x1f4 | [CH7\_AL3\_WRITE\_ADDR](#_bookmark190) | 通道7 WRITE\_ADDR寄存器的寄存器 |
| 0x1f8 | [CH7\_AL3\_TRANS\_CRYSTAL](#_bookmark191) | 通道7 TRANS\_DATA寄存器 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x1fc | [CH7\_AL3\_READ\_ADDR\_TRIG](#_bookmark192) | 通道7 READ\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x200 | [CH8\_READ\_ADDR](#_bookmark177) | DMA通道8读地址指针 |
| 0x204 | [CH8\_WRITE\_ADDR](#_bookmark178) | DMA通道8写地址指针 |
| 0x208 | [CH8\_TRANS\_CH8](#_bookmark179) | DMA通道8传输计数 |
| 0x20c | [CH8\_CTRL\_触发](#_bookmark180) | DMA通道8控制和状态 |
| 0x210 | [CH8\_AL1\_CTRL](#_bookmark181) | 通道8 CTRL寄存器的复位 |
| 0x214 | [CH8\_AL1\_READ\_ADDR](#_bookmark182) | 通道8 READ\_ADDR寄存器的寄存器 |
| 0x218 | [CH8\_AL1\_WRITE\_ADDR](#_bookmark183) | 通道8 WRITE\_ADDR寄存器的寄存器 |
| 0x21c | [CH8\_AL1\_TRANS\_COUNT\_TRIG](#_bookmark184) | 通道8 TRANS\_DATA寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x220 | [CH8\_AL2\_CTRL](#_bookmark185) | 通道8 CTRL寄存器的复位 |
| 0x224 | [CH8\_AL2\_TRANS\_COUNT](#_bookmark186) | 通道8 TRANS\_DATA寄存器的寄存器 |
| 0x228 | [CH8\_AL2\_READ\_ADDR](#_bookmark187) | 通道8 READ\_ADDR寄存器的寄存器 |
| 0x22c | [CH8\_AL2\_WRITE\_ADDR\_TRIG](#_bookmark188) | 通道8 WRITE\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x230 | [CH8\_AL3\_CTRL](#_bookmark189) | 通道8 CTRL寄存器的复位 |
| 0x234 | [CH8\_AL3\_WRITE\_ADDR](#_bookmark190) | 通道8 WRITE\_ADDR寄存器的寄存器 |
| 0x238 | [CH8\_AL3\_TRANS\_CH3](#_bookmark191) | 通道8 TRANS\_DATA寄存器的寄存器 |
| 0x23c | [CH8\_AL3\_READ\_ADDR\_TRIG](#_bookmark192) | 通道8 READ\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x240 | [CH9\_READ\_ADDR](#_bookmark177) | DMA通道9读地址指针 |
| 0x244 | [CH9\_WRITE\_ADDR](#_bookmark178) | DMA通道9写地址指针 |
| 0x248 | [CH9\_TRANS\_CH2](#_bookmark179) | DMA通道9传输计数 |
| 0x24c | [CH9\_CTRL\_触发](#_bookmark180) | DMA通道9控制和状态 |
| 0x250 | [CH9\_AL1\_CTRL](#_bookmark181) | 通道9 CTRL寄存器的复位 |
| 0x254 | [CH9\_AL1\_READ\_ADDR](#_bookmark182) | 通道9 READ\_ADDR寄存器的寄存器 |
| 0x258 | [CH9\_AL1\_WRITE\_ADDR](#_bookmark183) | 通道9 WRITE\_ADDR寄存器的寄存器 |
| 0x25c | [CH9\_AL1\_TRANS\_COUNT\_TRIG](#_bookmark184) | 通道9 TRANS\_DATA寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x260 | [CH9\_AL2\_CTRL](#_bookmark185) | 通道9 CTRL寄存器的复位 |
| 0x264 | [CH9\_AL2\_TRANS\_CRYSTAL](#_bookmark186) | 通道9 TRANS\_DATA寄存器的寄存器 |
| 0x268 | [CH9\_AL2\_READ\_ADDR](#_bookmark187) | 通道9 READ\_ADDR寄存器的寄存器 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x26c | [CH9\_AL2\_WRITE\_ADDR\_TRIG](#_bookmark188) | 通道9 WRITE\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x270 | [CH9\_AL3\_CTRL](#_bookmark189) | 通道9 CTRL寄存器的复位 |
| 0x274 | [CH9\_AL3\_WRITE\_ADDR](#_bookmark190) | 通道9 WRITE\_ADDR寄存器的寄存器 |
| 0x278 | [CH9\_AL3\_TRANS\_CRYSTAL](#_bookmark191) | 通道9 TRANS\_DATA寄存器的寄存器 |
| 0x27c | [CH9\_AL3\_READ\_ADDR\_TRIG](#_bookmark192) | 通道9 READ\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x280 | [CH10\_READ\_ADDR](#_bookmark177) | DMA通道10读地址指针 |
| 0x284 | [CH10\_WRITE\_ADDR](#_bookmark178) | DMA通道10写地址指针 |
| 0x288 | [CH10\_TRANS\_CH10](#_bookmark179) | DMA通道10传输计数 |
| 0x28c | [CH10\_CTRL\_TRIG](#_bookmark180) | DMA通道10控制和状态 |
| 0x290 | [CH10\_AL1\_CTRL](#_bookmark181) | 通道10 CTRL寄存器的复位 |
| 0x294 | [CH10\_AL1\_READ\_ADDR](#_bookmark182) | 通道10 READ\_ADDR寄存器的寄存器 |
| 0x298 | [CH10\_AL1\_WRITE\_ADDR](#_bookmark183) | 通道10 WRITE\_ADDR寄存器的寄存器 |
| 0x29c | [CH10\_AL1\_TRANS\_COUNT\_TRIG](#_bookmark184) | 通道10的寄存器TRANS\_DATA寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x2a0 | [CH10\_AL2\_CTRL](#_bookmark185) | 通道10 CTRL寄存器的复位 |
| 0x2a4 | [CH10\_AL2\_TRANS\_CRYSTAL](#_bookmark186) | 通道10的寄存器TRANS\_DATA寄存器 |
| 0x2a8 | [CH10\_AL2\_READ\_ADDR](#_bookmark187) | 通道10 READ\_ADDR寄存器的寄存器 |
| 0x2ac | [CH10\_AL2\_WRITE\_ADDR\_TRIG](#_bookmark188) | 通道10 WRITE\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x2b0 | [CH10\_AL3\_CTRL](#_bookmark189) | 通道10 CTRL寄存器的复位 |
| 0x2b4 | [CH10\_AL3\_WRITE\_ADDR](#_bookmark190) | 通道10 WRITE\_ADDR寄存器的寄存器 |
| 0x2b8 | [CH10\_AL3\_TRANS\_CRYSTAL](#_bookmark191) | 通道10的寄存器TRANS\_DATA寄存器 |
| 0x2bc | [CH10\_AL3\_READ\_ADDR\_TRIG](#_bookmark192) | 通道10 READ\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x2c0 | [CH11\_READ\_ADDR](#_bookmark177) | DMA通道11读地址指针 |
| 0x2c4 | [CH11\_WRITE\_ADDR](#_bookmark178) | DMA通道11写地址指针 |
| 0x2c8 | [CH11\_TRANS\_CH11](#_bookmark179) | DMA通道11传输计数 |
| 0x2cc | [CH11\_CTRL\_触发](#_bookmark180) | DMA通道11控制和状态 |
| 0x2d0 | [CH11\_AL1\_CTRL](#_bookmark181) | 通道11 CTRL寄存器的复位 |
| 0x2d4 | [CH11\_AL1\_READ\_ADDR](#_bookmark182) | 通道11 READ\_ADDR寄存器的寄存器 |
| 0x2d8 | [CH11\_AL1\_WRITE\_ADDR](#_bookmark183) | 通道11 WRITE\_ADDR寄存器的寄存器 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x2dc | [CH11\_AL1\_TRANS\_COUNT\_TRIG](#_bookmark184) | 通道11 TRANS\_DATA寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x2e0 | [CH11\_AL2\_CTRL](#_bookmark185) | 通道11 CTRL寄存器的复位 |
| 0x2e4 | [CH11\_AL2\_TRANS\_CRYSTAL](#_bookmark186) | 通道11 TRANS\_DATA寄存器 |
| 0x2e8位 | [CH11\_AL2\_READ\_ADDR](#_bookmark187) | 通道11 READ\_ADDR寄存器的寄存器 |
| 0x2ec | [CH11\_AL2\_WRITE\_ADDR\_TRIG](#_bookmark188) | 通道11 WRITE\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x2f0 | [CH11\_AL3\_CTRL](#_bookmark189) | 通道11 CTRL寄存器的复位 |
| 0x2f4 | [CH11\_AL3\_WRITE\_ADDR](#_bookmark190) | 通道11 WRITE\_ADDR寄存器的别名 |
| 0x2f8 | [CH11\_AL3\_TRANS\_CRYSTAL](#_bookmark191) | 通道11 TRANS\_DATA寄存器 |
| 0x2fc | [CH11\_AL3\_READ\_ADDR\_TRIG](#_bookmark192) | 通道11 READ\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 |
| 0x400 | [INTR](#_bookmark193) | 数据状态（原始） |
| 0x404 | [INTE0](#_bookmark194) | IRQ0启用 |
| 0x408 | [INTF 0](#_bookmark195) | 强制中断 |
| 0x40c | [INTS 0](#_bookmark196) | IRQ0的中断状态 |
| 0x414 | [INTE1](#_bookmark197) | IRQ1启用 |
| 0x418 | [INTF 1](#_bookmark198) | IRQ1强制中断 |
| 0x41c | [INTS 1](#_bookmark199) | IRQ1的中断状态（屏蔽） |
| 0x420 | [Timer0](#_bookmark200) | 起搏（X/Y）部分计时器  起搏定时器以（（X/Y）\* sys\_bits）设置的速率产生TREQ断言。此公式在每个sys\_task周期进行评估，因此只能以每个sys\_task 1（即永久TREQ）或更少的速率生成TREQ |
| 0x424 | [Timer1](#_bookmark200) | 起搏（X/Y）部分计时器  起搏定时器以（（X/Y）\* sys\_bits）设置的速率产生TREQ断言。此公式在每个sys\_task周期进行评估，因此只能以每个sys\_task 1（即永久TREQ）或更少的速率生成TREQ |
| 0x428 | [Timer2](#_bookmark200) | 起搏（X/Y）部分计时器  起搏定时器以（（X/Y）\* sys\_bits）设置的速率产生TREQ断言。此公式在每个sys\_task周期进行评估，因此只能以每个sys\_task 1（即永久TREQ）或更少的速率生成TREQ |
| 0x42c | [定时器3](#_bookmark200) | 起搏（X/Y）部分计时器  起搏定时器以（（X/Y）\* sys\_bits）设置的速率产生TREQ断言。此公式在每个sys\_task周期进行评估，因此只能以每个sys\_task 1（即永久TREQ）或更少的速率生成TREQ |
| 0x430 | [多通道触发器](#_bookmark201) | 同时触发一个或多个通道 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x434 | [SNIFF\_CTRL](#_bookmark202) | 嗅探器控制 |
| 0x438 | [SNIFF\_DATA](#_bookmark203) | 用于嗅探硬件的数据累加器 |
| 0x440 | [FIFO\_LEVELS](#_bookmark204) | CARAF、WAF、TDF水平 |
| 0x444 | [CHAN\_中止](#_bookmark205) | 中止一个或多个通道上正在进行的传输序列 |
| 0x448 | [N\_通道](#_bookmark206) | 此DMA实例配备的通道数此DMA支持多达16个硬件通道，但可以配置为一个，以最大限度地减少硅面积。 |
| 0x800 | [CH0\_DBG\_CTDREQ](#_bookmark207) | 读取：获取通道DREQ计数器（即DMA期望它可以在外围设备上执行多少次访问而不会上溢/下溢。写入任意值：清除计数器，并使通道重新启动DREQ握手。 |
| 0x804 | [CH0\_DBG\_TCR](#_bookmark208) | 读取以获取通道TRANS\_REQ重载值，即下一次传输的长度 |
| 0x840 | [CH1\_DBG\_CTDREQ](#_bookmark207) | 读取：获取通道DREQ计数器（即DMA期望它可以在外围设备上执行多少次访问而不会上溢/下溢。写入任意值：清除计数器，并使通道重新启动DREQ握手。 |
| 0x844 | [CH1\_DBG\_TCR](#_bookmark208) | 读取以获取通道TRANS\_REQ重载值，即下一次传输的长度 |
| 0x880 | [CH2\_DBG\_CTDREQ](#_bookmark207) | 读取：获取通道DREQ计数器（即DMA期望它可以在外围设备上执行多少次访问而不会上溢/下溢。写入任意值：清除计数器，并使通道重新启动DREQ握手。 |
| 0x884 | [CH2\_DBG\_TCR](#_bookmark208) | 读取以获取通道TRANS\_REQ重载值，即下一次传输的长度 |
| 0x8c0 | [CH3\_DBG\_CTDREQ](#_bookmark207) | 读取：获取通道DREQ计数器（即DMA期望它可以在外围设备上执行多少次访问而不会上溢/下溢。写入任意值：清除计数器，并使通道重新启动DREQ握手。 |
| 0x8c4 | [CH3\_DBG\_TCR](#_bookmark208) | 读取以获取通道TRANS\_REQ重载值，即下一次传输的长度 |
| 0x900 | [CH4\_DBG\_CTDREQ](#_bookmark207) | 读取：获取通道DREQ计数器（即DMA期望它可以在外围设备上执行多少次访问而不会上溢/下溢。写入任意值：清除计数器，并使通道重新启动DREQ握手。 |
| 0x904 | [CH4\_DBG\_TCR](#_bookmark208) | 读取以获取通道TRANS\_REQ重载值，即下一次传输的长度 |
| 0x940 | [CH5\_DBG\_CTDREQ](#_bookmark207) | 读取：获取通道DREQ计数器（即DMA期望它可以在外围设备上执行多少次访问而不会上溢/下溢。写入任意值：清除计数器，并使通道重新启动DREQ握手。 |
| 0x944 | [CH5\_DBG\_TCR](#_bookmark208) | 读取以获取通道TRANS\_REQ重载值，即下一次传输的长度 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x980 | [CH6\_DBG\_CTDREQ](#_bookmark207) | 读取：获取通道DREQ计数器（即DMA期望它可以在外围设备上执行多少次访问而不会上溢/下溢。写入任意值：清除计数器，并使通道重新启动DREQ握手。 |
| 0x984 | [CH6\_DBG\_TCR](#_bookmark208) | 读取以获取通道TRANS\_REQ重载值，即下一次传输的长度 |
| 0x9c0 | [CH7\_DBG\_CTDREQ](#_bookmark207) | 读取：获取通道DREQ计数器（即DMA期望它可以在外围设备上执行多少次访问而不会上溢/下溢。写入任意值：清除计数器，并使通道重新启动DREQ握手。 |
| 0x9c4 | [CH7\_DBG\_TCR](#_bookmark208) | 读取以获取通道TRANS\_REQ重载值，即下一次传输的长度 |
| 0xa00 | [CH8\_DBG\_CTDREQ](#_bookmark207) | 读取：获取通道DREQ计数器（即DMA期望它可以在外围设备上执行多少次访问而不会上溢/下溢。写入任意值：清除计数器，并使通道重新启动DREQ握手。 |
| 0xa04 | [CH8\_DBG\_TCR](#_bookmark208) | 读取以获取通道TRANS\_REQ重载值，即下一次传输的长度 |
| 0xa40 | [CH9\_DBG\_CTDREQ](#_bookmark207) | 读取：获取通道DREQ计数器（即DMA期望它可以在外围设备上执行多少次访问而不会上溢/下溢。写入任意值：清除计数器，并使通道重新启动DREQ握手。 |
| 0xa44 | [CH9\_DBG\_TCR](#_bookmark208) | 读取以获取通道TRANS\_REQ重载值，即下一次传输的长度 |
| 0xa80 | [CH10\_DBG\_CTDREQ](#_bookmark207) | 读取：获取通道DREQ计数器（即DMA期望它可以在外围设备上执行多少次访问而不会上溢/下溢。写入任意值：清除计数器，并使通道重新启动DREQ握手。 |
| 0xa84 | [CH10\_DBG\_TCR](#_bookmark208) | 读取以获取通道TRANS\_REQ重载值，即下一次传输的长度 |
| 0xac0 | [CH11\_DBG\_CTDREQ](#_bookmark207) | 读取：获取通道DREQ计数器（即DMA期望它可以在外围设备上执行多少次访问而不会上溢/下溢。写入任意值：清除计数器，并使通道重新启动DREQ握手。 |
| 0xac4 | [CH11\_DBG\_TCR](#_bookmark208) | 读取以获取通道TRANS\_REQ重载值，即下一次传输的长度 |

### [DMA](#_bookmark176)： CH0\_READ\_ADDR，CH1\_READ\_ADDR，CH10\_READ\_ADDR，

CH11\_READ\_ADDR寄存器

**偏移量**：0x000、0x040、...、0x280、0x2c0

描述

DMA通道*N*读地址指针

表121.CH0\_READ\_ADDR，CH1\_READ\_ADDR，.，CH10\_READ\_ADDR，CH11\_READ\_ADDR

寄存器

表122.CH0\_WRITE\_ADDR，CH1\_WRITE\_ADDR，.，CH10\_WRITE\_ADDR，CH11\_WRITE\_ADDR

寄存器

表123.对妇女的歧视CH0\_TRANS\_CH2、CH1\_TRANS\_CH3、

.，CH10\_TRANS\_CH2，CH11\_TRANS\_CH3

寄存器

表124.CH0\_CTRL\_TRIG，CH1\_CTRL\_TRIG，.，CH10\_CTRL\_TRIG，CH11\_CTRL\_TRIG

寄存器

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 每次读取完成时，该寄存器自动更新。当前值是该通道要读取的下一个地址 | RW | 0x00000000 |

### [DMA](#_bookmark176)： CH0\_WRITE\_ADDR，CH1\_WRITE\_ADDR， .， CH10\_WRITE\_ADDR，

CH11\_WRITE\_ADDR寄存器

**偏移**：0x004、0x044、...、0x284、0x2c4

描述

DMA通道*N*写地址指针

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 每次写入完成时，该寄存器都会自动更新。当前值是该通道要写入的下一个地址 | RW | 0x00000000 |

### [DMA](#_bookmark176)：CH0\_TRANS\_CH2，CH1\_TRANS\_CH2，...，CH10\_TRANS\_CH2，

CH11\_TRANS\_RST寄存器

**偏移量**：0x 008、0x 048、.、0x 288、0x 2c8

描述

DMA通道*N*传输计数

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 设置一个通道在停止前将执行的总线传输次数请注意，如果传输的大小大于一个字节，则这不等于传输的字节数（请参见CTRL\_DATA\_SIZE）。  当通道处于活动状态时，读取此寄存器将显示剩余的传输次数，并在每次完成写入传输时自动更新。  写入此寄存器将设置传输计数器的RELOAD值。 每次触发此通道时，RELOAD值都会复制到实时传输计数器中。该通道可以启动多次，每次执行的传输次数与最近一次写入所编程的传输次数相同  可以在CHx\_DBG\_TCR处观察到RELOAD值如果TRANS\_DATA用作触发器，则写入的值将立即用作新传输序列的长度 | RW | 0x00000000 |

### [DMA](#_bookmark176)： CH0\_CTRL\_TRIG，CH1\_CTRL\_TRIG， .， CH10\_CTRL\_TRIG，

CH11\_CTRL\_TRIG寄存器

**偏移量**：0x00c、0x04c、...、0x28c、0x2cc

描述

DMA通道*N*控制和状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | AHB\_ERROR | READ\_ERROR和WRITE\_ERROR标志的逻辑OR通道在遇到任何总线错误时暂停，并始终提高其通道IRQ标志。 | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 30 | 读取错误 | 如果为1，则通道收到读总线错误。 写一个清除。  READ\_ADDR显示遇到总线错误的近似地址（不会更早，也不会超过3次传输） | WC | 0x0 |
| 29 | 写入错误 | 如果为1，则通道收到写总线错误。 写一个清除。  WRITE\_ADDR显示遇到总线错误的近似地址（不会更早，也不会超过5次传输） | WC | 0x0 |
| 二十八点二十五分 | Reserved. | - | - | - |
| 24 | 忙 | 此标志在通道开始新的传输序列时变为高电平，在该序列的最后一次传输完成时变为低BUSY为高电平时清除EN将暂停通道，BUSY将在暂停时保持高电平。  要提前终止序列（并清除BUSY标志），请参见CHAN\_ABORT。 | RO | 0x0 |
| 23 | SNIFF\_EN | 如果为1，则此通道的数据传输对嗅探硬件可见，并且每次传输都将提升校验和的状态这仅适用于启用嗅探硬件并选择此通道的情况。  这允许在每个控制块的基础上启用或禁用校验和 | RW | 0x0 |
| 22 | BSWAP | 对DMA数据应用字节交换转换。  对于字节数据，这没有影响。对于半字数据，交换每个半字的两个字节 对于字数据，每个字的四个字节按相反顺序交换。 | RW | 0x0 |
| 21 | IRQ\_QUIET | 在QUIET模式下，通道不会在每个传输块结束时生成IRQ相反，当将空值写入触发寄存器时，会引发IRQ，指示控制区块链的结束  这减少了在传输由许多小控制块组成的DMA链时CPU要处理的中断数量 | RW | 0x0 |
| 20:十五分 | TREQ\_SEL | 选择传输请求信号。  信道使用传输请求信号来调整其数据传输速率。 TREQ信号的来源是内部（TIMERS）或外部（DREQ，来自系统的数据请求）。  0x0至0x3a→选择DREQ n作为TREQ0x3b→选择定时器0作为TREQ  0x3c→选择定时器1作为TREQ  0x3d→选择定时器2作为TREQ（可选）0x3e→选择定时器3作为TREQ（可选）  0x3f→永久请求，用于不同步传输。 | RW | 0x00 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 14:十一分 | 链接到 | 当此通道完成时，它将触发CHAIN\_TO指示的通道通过设置CHAIN\_TO =*（此通道）*禁用。 | RW | 0x0 |
| 10 | 环选择 | 选择RING\_SIZE是否应用于读地址或写地址。  如果为0，则读地址在（1 RING\_SIZE）边界上绕包如果为1，则写入地址被换行。 | RW | 0x0 |
| 9:六分 | 环\_尺寸 | 地址回绕区域的大小如果为0，则不换行。对于值n  > 0时，只有地址的低n位会改变。这将地址包装在（1n）字节边界上，便于访问自然对齐的环形缓冲区。  环大小可以在2到32768字节根据RING\_SEL的值，这可以应用于读或写地址。  0x0→RING\_NONE | RW | 0x0 |
| 5 | 增量写入 | 如果为1，则写入地址随着每次传输而递增。如果为0，则每次写入都定向到相同的初始地址。  一般来说，对于内存到外围设备的传输，应禁用 | RW | 0x0 |
| 4 | INCR\_READ | 如果为1，则读取地址随着每次传输而递增。如果为0，则每次读取都指向相同的初始地址。  通常，对于外围设备到内存的传输，应该禁用 | RW | 0x0 |
| 三比二 | DATA\_SIZE | 设置每个总线传输的大小（字节/半字/字）。READ\_ADDR和WRITE\_ADDR在每次传输时都前进此量（1/2/4字节）。  0x0→大小\_字节  0x1→尺寸\_半字  0x2→SIZE\_WORD | RW | 0x0 |
| 1 | 高优先级 | HIGH\_PRIORITY在问题调度中给予通道优先处理：在每个调度轮中，首先考虑所有高优先级通道，然后仅考虑单个低优先级通道，然后返回到高优先级通道。  这只会影响DMA调度通道的顺序DMA的总线优先级不变。如果DMA未饱和，则低优先级通道不会出现吞吐量损失。 | RW | 0x0 |
| 0 | EN | DMA通道使能。  当为1时，通道将响应触发事件，这将使其变为BUSY并开始传输数据。当为0时，通道将忽略触发器，停止发出传输，并暂停当前传输序列（即，  如果已经很高，BUSY将保持高 | RW | 0x0 |

### 

### [DMA](#_bookmark176)：CH0\_AL1\_CTRL、CH1\_AL1\_CTRL、...、CH10\_AL1\_CTRL、CH11\_AL1\_CTRL

表125.CH0\_AL1\_CTRL，CH1\_AL1\_CTRL，.，CH10\_AL1\_CTRL，CH11\_AL1\_CTRL

寄存器

表126.CH0\_AL1\_READ\_ADDR

、CH1\_AL1\_READ\_ADDR

，.，CH10\_AL1\_READ\_ADDR，CH11\_AL1\_READ\_ADD

R寄存器

表127.对妇女的歧视CH0\_AL1\_WRITE\_ADDR，CH1\_AL1\_WRITE\_ADDR，.，CH10\_AL1\_WRITE\_ADDR，CH11\_AL1\_WRITE\_AD

DR寄存器

表128.对妇女的歧视CH0\_AL1\_TRANS\_COUNT\_TRIG，CH1\_AL1\_TRANS\_COUNT\_TRIG，...，CH10\_AL1\_TRANS\_COUNT\_TRIG，CH11\_AL1\_TRANS\_CO

UNT\_TRIG寄存器

表129.CH0\_AL2\_CTRL，CH1\_AL2\_CTRL，.，CH10\_AL2\_CTRL，CH11\_AL2\_CTRL

寄存器

表130.CH0\_AL2\_TRANS\_COUNT，CH1\_AL2\_TRANS\_COUNT，.，CH10\_AL2\_TRANS\_CONT，CH11\_AL2\_TRANS\_CO

UNT寄存器

### 寄存器

**偏移**：0x010、0x050、...、0x290、0x2d0

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 通道*N*CTRL寄存器的寄存器 | RW | - |

### [DMA](#_bookmark176)： CH0\_AL1\_READ\_ADDR，CH1\_AL1\_READ\_ADDR， ...，

CH10\_AL1\_READ\_ADDR、CH11\_AL1\_READ\_ADDR寄存器

**偏移量**：0x014、0x054、...、0x294、0x2d4

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 通道*N*READ\_ADDR寄存器的寄存器 | RW | - |

### [DMA](#_bookmark176)： CH0\_AL1\_WRITE\_ADDR，CH1\_AL1\_WRITE\_ADDR， .，CH10\_AL1\_WRITE\_ADDR，CH11\_AL1\_WRITE\_ADDR寄存器

**偏移量**：0x018、0x058、...、0x298、0x2d8

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 通道*N*WRITE\_ADDR寄存器的寄存器 | RW | - |

### [DMA](#_bookmark176)： CH0\_AL1\_TRANS\_COUNT\_TRIG、CH1\_AL1\_TRANS\_COUNT\_TRIG、 .、CH10\_AL1\_TRANS\_COUNT\_TRIG、CH11\_AL1\_TRANS\_COUNT\_TRIG寄存器

**偏移**：0x01c、0x05c、...、0x29c、0x2dc

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 通道*N*TRANS\_DATA寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 | RW | - |

### [DMA](#_bookmark176)：CH0\_AL2\_CTRL、CH1\_AL2\_CTRL、...、CH10\_AL2\_CTRL、CH11\_AL2\_CTRL

寄存器

**偏移量**：0x020、0x060、...、0x2a0、0x2e0

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 通道*N*CTRL寄存器的寄存器 | RW | - |

### [DMA](#_bookmark176)： CH0\_AL2\_TRANS\_CH2、CH1\_AL2\_TRANS\_CH2、 .、CH10\_AL2\_TRANS\_CH2、CH11\_AL2\_TRANS\_CH2寄存器

**偏移量**：0x024、0x064、...、0x2a4、0x2e4

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 通道*N*TRANS\_DATA寄存器 | RW | - |

### [DMA](#_bookmark176)： CH0\_AL2\_READ\_ADDR，CH1\_AL2\_READ\_ADDR， ...，

CH10\_AL2\_READ\_ADDR、CH11\_AL2\_READ\_ADDR寄存器

**偏移量**：0x028、0x068、...、0x2a8、0x2e8

表131.对妇女的歧视CH0\_AL2\_READ\_ADDR

、CH1\_AL2\_READ\_ADDR

，...，CH10\_AL2\_READ\_ADDR，CH11\_AL2\_READ\_ADD

R寄存器

表132.CH0\_AL2\_WRITE\_ADDR\_TRIG，CH1\_AL2\_WRITE\_ADDR\_TRIG，.，CH10\_AL2\_WRITE\_ADDR\_TRIG，CH11\_AL2\_WRITE\_AD

DR\_TRIG寄存器

表133.CH0\_AL3\_CTRL，CH1\_AL3\_CTRL，.，CH10\_AL3\_CTRL，CH11\_AL3\_CTRL

寄存器

表134.CH0\_AL3\_WRITE\_ADDR、CH1\_AL3\_WRITE\_ADDR、...、CH10\_AL3\_WRITE\_ADDR、CH11\_AL3\_WRITE\_AD

DR寄存器

表135.对妇女的歧视CH0\_AL3\_TRANS\_COUNT、CH1\_AL3\_TRANS\_COUNT、.、CH10\_AL3\_TRANS\_CONT、CH11\_AL3\_TRANS\_CO

UNT寄存器

表136.对妇女的歧视CH0\_AL3\_READ\_ADDR

\_TRIG，CH1\_AL3\_READ\_ADDR

\_TRIG，...，CH10\_AL3\_READ\_ADDR\_TRIG，CH11\_AL3\_READ\_ADD

R\_TRIG寄存器

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 通道*N*READ\_ADDR寄存器的寄存器 | RW | - |

### [DMA](#_bookmark176)： CH0\_AL2\_WRITE\_ADDR\_TRIG，CH1\_AL2\_WRITE\_ADDR\_TRIG， .，CH10\_AL2\_WRITE\_ADDR\_TRIG，CH11\_AL2\_WRITE\_ADDR\_TRIG寄存器

**偏移**：0x02c、0x06c、...、0x2ac、0x2ec

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 通道*N*WRITE\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 | RW | - |

### [DMA](#_bookmark176)：CH0\_AL3\_CTRL、CH1\_AL3\_CTRL、...、CH10\_AL3\_CTRL、CH11\_AL3\_CTRL

寄存器

**偏移**：0x030、0x070、...、0x2b0、0x2f0

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 通道*N*CTRL寄存器的寄存器 | RW | - |

### [DMA](#_bookmark176)： CH0\_AL3\_WRITE\_ADDR、CH1\_AL3\_WRITE\_ADDR、 .、CH10\_AL3\_WRITE\_ADDR、CH11\_AL3\_WRITE\_ADDR寄存器

**偏移量**：0x034、0x074、...、0x2b4、0x2f4

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 通道*N*WRITE\_ADDR寄存器的寄存器 | RW | - |

### [DMA](#_bookmark176)： CH0\_AL3\_TRANS\_CH2、CH1\_AL3\_TRANS\_CH2、 .、CH10\_AL3\_TRANS\_CH2、CH11\_AL3\_TRANS\_CH2寄存器

**偏移量**：0x038、0x078、...、0x2b8、0x2f8

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 通道*N*TRANS\_DATA寄存器 | RW | - |

### [DMA](#_bookmark176)： CH0\_AL3\_READ\_ADDR\_TRIG、CH1\_AL3\_READ\_ADDR\_TRIG、 .、CH10\_AL3\_READ\_ADDR\_TRIG、CH11\_AL3\_READ\_ADDR\_TRIG寄存器

**偏移**：0x03c、0x07c、...、0x2bc、0x2fc

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 通道*N*READ\_ADDR寄存器的寄存器  这是一个触发寄存器（0xc）。写入非零值将重新加载通道计数器并启动通道。 | RW | - |

### [DMA](#_bookmark176)：INTR寄存器

**偏移**：0x400

描述

数据状态（原始）

*表137.对妇女的歧视INTR寄存器*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - |
| 十5比零 | DMA通道0的原始中断状态15. 比特n对应于信道n。忽略任何掩盖或强迫。通过向INTR、INTS 0或INTS 1写入位掩码，可以清除通道中断  通道中断可以路由到基于INTE0和INTE1的两个系统级IRQ之一  这可以用于将不同的通道中断引导到不同的ISR：这样做可以允许NVICIRQ抢占时间更关键的通道，或者将IRQ负载分散在不同的内核上。  忽略此行为并仅使用INTE 0/INTS 0/IRQ 0也是有效的。 | WC | 0x0000 |

表138.对妇女的歧视INTE0

寄存器

表139.对妇女的歧视INTF 0

寄存器

### [DMA](#_bookmark176)：INTE0寄存器

**偏移量**：0x404

描述

IRQ0启用

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - |
| 十5:0 | 设置位n以将中断从通道n传递到DMA IRQ 0。 | RW | 0x0000 |

### [DMA](#_bookmark176)：INTF 0寄存器

**偏移量**：0x408

描述

强制中断

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - |
| 十5:0 | 写入1以强制INTE0中的相应位。中断保持置位，直到INTF 0被清除。 | RW | 0x0000 |

### [DMA](#_bookmark176)：INTS 0寄存器

**偏移**：0x40c

描述

IRQ0的中断状态

*表140.第140页INTS 0*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - |
| 十5比零 | 指示当前导致IRQ0置位的活动通道中断请求  通道中断可以通过在此处写入位掩码来清除 | WC | 0x0000 |

寄存器

表141.对妇女的歧视INTE1

寄存器

表142.对妇女的歧视INTF 1

寄存器

表143.对妇女的歧视INTS 1

寄存器

### [DMA](#_bookmark176)：INTE1寄存器

**偏移量**：0x414

描述

IRQ1启用

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - |
| 十5:0 | 设置位n以将中断从通道n传递到DMA IRQ 1。 | RW | 0x0000 |

### [DMA](#_bookmark176)：INTF 1寄存器

**偏移**：0x418

描述

IRQ1强制中断

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - |
| 十5:0 | 写入1以强制INTE0中的相应位。中断保持置位，直到INTF 0被清除。 | RW | 0x0000 |

### [DMA](#_bookmark176)：INTS 1寄存器

**偏移**：0x41c

描述

IRQ1的中断状态（屏蔽）

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - |
| 十5:0 | 指示当前导致IRQ1置位的活动通道中断请求  通道中断可以通过在此处写入位掩码来清除 | WC | 0x0000 |

### [DMA](#_bookmark176)：TIMER0、TIMER1、TIMER2、TIMER3寄存器

**偏移**：0x420、0x424、0x428、0x42c

描述

起搏（X/Y）部分计时器

起搏定时器以（（X/Y）\* sys\_bits）设置的速率产生TREQ断言。该等式在每个sys\_tem周期进行评估，因此只能以每个sys\_tem 1的速率生成TREQ（即，永久TREQ）或更少。

表144.对妇女的歧视定时器0、定时器1、定时器2、

TIMER3寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | X | 起搏定时器红利。指定（X/Y）小数计时器的X值 | RW | 0x0000 |
| 十5:0 | Y | 起搏计时器除数。指定（X/Y）小数计时器的Y值 | RW | 0x0000 |

表145.对妇女的歧视多通道触发

R寄存器

表146.对妇女的歧视SNIFF\_CTRL寄存器

### [DMA](#_bookmark176)：MULTI\_CHAN\_TRIGGER寄存器

**偏移**：0x430

描述

同时触发一个或多个通道

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - |
| 十5:0 | 此寄存器中的每个位对应于一个DMA通道。将1写入相关位与写入该通道的触发寄存器相同;如果通道当前启用且尚未繁忙，则通道将启动 | SC | 0x0000 |

### [DMA](#_bookmark176)：SNIFF\_CTRL寄存器

**偏移量**：0x434

描述

嗅探器控制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - | - |
| 11 | OUT\_INV | 如果设置，则读取时结果显示为反转（按位这不会影响校验和的计算方式;结果在结果寄存器和总线之间动态转换 | RW | 0x0 |
| 10 | OUT\_REV | 如果设置，则读取结果时显示为位反转。这不会影响校验和的计算方式;结果在结果寄存器和总线之间动态转换 | RW | 0x0 |
| 9 | BSWAP | 在输入校验和之前，在本地对嗅探数据执行字节反转  请注意，嗅探器硬件位于读主机中执行的DMA通道字节交换的下游：如果通道CTRL\_BSWAP和SNIFF\_CTRL\_BSWAP都启用，则从嗅探器的角度来看，它们的效果会取消。 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 八点五 | CALC | 0x 0→计算CRC-32（IEEE802.3多项式）  0x 1→使用位反转数据计算CRC-32（IEEE 802.3多项式）  0x 2→计算CRC-16-CCITT  0x 3→计算CRC-16-CCITT与位反转数据0xe→ XOR减少所有数据。== 1，如果总的1个种群计数是奇数。  0xf→计算简单的32位校验和（与32位累加器相加） | RW | 0x0 |
| 4:一 | DMACH | 用于嗅探器观察的DMA通道 | RW | 0x0 |
| 0 | EN | 启用嗅探器 | RW | 0x0 |

表147.对妇女的歧视SNIFF\_DATA寄存器

表148.对妇女的歧视FIFO\_LEVELS寄存器

表149.对妇女的歧视CHAN\_中止

寄存器

### [DMA](#_bookmark176)：SNIFF\_DATA寄存器

**偏移量**：0x438

描述

用于嗅探硬件的数据累加器

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 在SNIFF\_CTRL\_DACH指示的通道上开始DMA传输之前，在此处写入初始种子值硬件将在每次观察到从指定通道读取数据时更新此寄存器通道完成后，可从此寄存器读取最终结果 | RW | 0x00000000 |

### [DMA](#_bookmark176)：FIFO\_LEVELS寄存器

**偏移**：0x440

描述

CARAF、WAF、TDF水平

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31：24 | Reserved. | - | - | - |
| 23:16 | RAF\_LVL | 当前读地址FIFO填充水平 | RO | 0x00 |
| 15:8 | WAF\_LVL | 当前写地址FIFO填充水平 | RO | 0x00 |
| 7:0 | TDF\_LVL | 当前传输数据FIFO填充水平 | RO | 0x00 |

### [DMA](#_bookmark176)：CHAN\_ABORT寄存器

**偏移量**：0x444

描述

中止一个或多个通道上正在进行的传输序列

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - |

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 十5:0 | 每个比特对应一个信道。 写入1将中止该通道上正在进行的任何传输序列。该位将保持高电平，直到通过地址和数据FIFO刷新了任何正在  写入后，必须轮询该寄存器，直到它返回全零。在此之前，重新启动通道是不安全的 | SC | 0x0000 |

表150.第150页N\_CHANNELS寄存器

表151.对妇女的歧视CH0\_DBG\_CTDREQ、CH1\_DBG\_CTDREQ、.、CH10\_DBG\_CTDREQ、CH11\_DBG\_CTDREQ

寄存器

表152.对妇女的歧视CH0\_DBG\_TCR，CH1\_DBG\_TCR，.，CH10\_DBG\_TCR，CH11\_DBG\_TCR

寄存器

### [DMA](#_bookmark176)：N\_CHANNELS寄存器

**偏移量**：0x448

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:5 | Reserved. | - | - |
| 4:0 | 此DMA实例配备的通道数。此DMA支持多达16个硬件通道，但可以配置为一个，以最大限度地减少硅面积。 | RO | - |

### [DMA](#_bookmark176)： CH0\_DBG\_CTDREQ，CH1\_DBG\_CTDREQ， .， CH10\_DBG\_CTDREQ，

CH11\_DBG\_CTDREQ寄存器

**偏移量**：0x800、0x840、...、0xa80、0xac0

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:六 | Reserved. | - | - |
| 5:0 | 读取：获取通道DREQ计数器（即 DMA期望它可以在外围设备上执行多少次访问而不会上溢/下溢。写入任意值：清除计数器，并使通道重新启动DREQ握手。 | WC | 0x00 |

### [DMA](#_bookmark176)：CH0\_DBG\_TCR，CH1\_DBG\_TCR，.，CH10\_DBG\_TCR，CH11\_DBG\_TCR

寄存器

**偏移量**：0x804、0x844、...、0xa84、0xac4

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 读取以获取通道TRANS\_REQ重载值，即下一次传输的长度 | RO | 0x00000000 |

* 1. 存储器

RP2040具有嵌入式ROM和SRAM，并通过QSPI接口访问外部Flash。内部存储器的详细信息如下所示。

* + 1. ROM

16kB只读存储器（ROM）的地址为0x0000000。ROM内容在硅制造时是固定的它包括：

* 初始启动程序
* 闪存引导序列
* Flash编程例程
* 支持UF2的USB大容量存储设备
* 实用程序库，如快速浮点

[第2.8.1](#_bookmark225)定义了芯片的引导顺序，第2.8.2[节](#_bookmark224)详细描述了ROM内容

2.8 RP2040 bootrom的完整源代码可从以下网址获得

[微微自举只读存储器](https://github.com/raspberrypi/pico-bootrom)

ROM提供单周期只读总线访问，并且位于专用AHB-Lite仲裁器上，因此可以与其他存储器设备同时访问。尝试写入ROM无效（不会产生总线故障

* + 1. SRAM

片内SRAM的总容量为264kB物理上，这被划分为六个存储体，因为这极大地提高了多个主机的内存带宽，但软件可能会将其视为单个264kB内存区域。每个存储体中存储的内容没有限制：处理器代码、数据缓冲区或混合。有四个16k x 32位存储体（每个64kB）和两个1k x 32位存储体（每个4kB）。

##### 重要

银行是SRAM的*物理*分区，通过允许多个同时访问来提高性能*逻辑上*有一个264kB的连续内存。

表153.对妇女的歧视 SRAM bank 0/1/2/3条带映射。

每个SRAM存储体都通过专用的AHB-Lite仲裁器进行访问。这意味着不同的总线主机可以并行访问不同的SRAM存储体，因此每个系统时钟周期最多可以进行四次32位SRAM访问（每个主机一次）。

SRAM映射到从0x20000000开始的系统地址。第一个256kB地址区域跨四个较大的存储体进行字条带化，这为大多数用例提供了显著的存储器并行性优势

系统地址空间中的连续字被路由到不同的RAM存储体，如[表153所示](#_bookmark212)。

|  |  |  |
| --- | --- | --- |
| 系统地址 | SRAM存储体 | SRAM字地址 |
| 0x20000000 | 银行0 | 0 |
| 0x20000004 | 组1 | 0 |
| 0x20000008 | 缸组2 | 0 |
| 0x2000000c | 存储体3 | 0 |
| 0x20000010 | 第0 | 1 |
| 0x20000014 | 组1 | 1 |
| 0x20000018 | 缸组2 | 1 |
| 0x2000001c | 存储体3 | 1 |
| 0x20000020 | 银行0 | 2 |
| 0x20000024 | 组1 | 2 |
| 0x20000028 | 缸组2 | 2 |
| 0x2000002c | 存储体3 | 2 |
| 等 |  |  |

接下来的两个4kB区域（从0x20040000和0x20041000开始）直接映射到较小的4kB存储体。软件*可以*选择将这些用于每核目的，例如：堆栈和频繁执行的代码，保证

处理器在这些访问上从不停止。然而，与RP 2040上的所有SRAM一样，这些存储体具有来自*所有*主机的单周期访问，前提是没有其他主机在同一周期内访问该存储体，因此将存储器视为单个264kB设备是合理的。

四个64kB存储体也可在非条纹镜像中使用从0x21000000、0x21010000、0x21020000、0x21030000开始的四个64kB区域中的每一个直接映射到四个64kB SRAM存储体中的一个。软件可以显式地在物理内存中分配数据和代码，以便在异常苛刻的情况下提高内存性能。这通常是不必要的，因为存储器条带化通常提供足够的并行性和更低的软件复杂性。

非分条镜像从SRAM基础上+16MB的偏移开始，因为这是允许在较小的存储体和非分条的较大存储体之间调用ARMv6M子例程的最大偏移

* + - 1. 其他片上存储器

除了264kB主存储器之外，还有两个其他专用RAM块可以在某些情况下使用

* 如果闪存XIP缓存被禁用，则缓存将变为从0x15000000开始的16 kB内存
* 如果不使用USB，则USB数据DPRAM可用作从0x50100000开始的4kB存储器

这样，片内SRAM的总容量为284kB。这些存储器的使用方式没有限制，例如，如果您选择，可以从USB数据RAM执行代码。

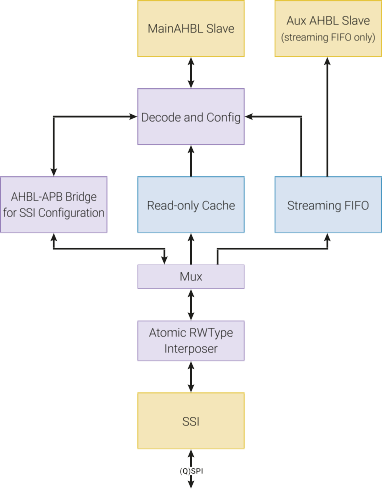
* + 1. Flash

外部闪存通过QSPI接口使用现场执行（XIP）硬件进行访问。这允许外部闪存被系统寻址和访问，就好像它是内部存储器一样。从0x10000000开始的16MB内存窗口的总线读取转换为串行闪存传输，结果返回到发起读取的主机。这个过程对主设备是透明的，因此处理器可以执行来自外部闪存的代码，而无需首先将代码复制到内部存储器，因此“就地执行”。内部缓存会记住最近访问的闪存位置的内容，从而加快接口的平均带宽和延迟

一旦RP2040的bootrom和闪存第二阶段正确配置，XIP硬件在很大程度上是透明的，软件可以将闪存视为大型只读存储器。但是，它确实提供了许多额外的功能，以满足更苛刻的软件用例。

图14.闪存就地执行（XIP）子系统。通过主AHB-Lite从站的系统访问被解码，以确定它们是否是XIP访问，直接访问SSI

例如 用于配置，或访问XIP子系统中的各种其他硬件和控制寄存器。 XIP访问首先在缓存中查找，以加速对最近使用的数据的访问。 如果在高速缓存中未找到数据，则通过SSI生成外部串行访问，并将结果数据存储在高速缓存中并转发到系统总线上。



**注意**

当使用SDK以系统时钟的整数分频器运行时，串行闪存接口由闪存第二级配置所有包含的第二阶段引导实现都支持PICO\_FLASH\_SPI\_CLKDIV设置（例如，在<https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/boot_stage2/boot2_w25q080.S>中默认为4，以使默认接口速度为125/4 = 31.25MHz）。可以通过在SDK使用的特定板配置头中指定PICO\_FLASH\_SPI\_CLKDIV来

* + - 1. XIP缓存

缓存为16kB，双向组关联，1周期命中。它位于XIP子系统内部，仅影响对XIP闪存的访问，因此软件不必考虑缓存一致性，除非执行闪存编程操作。它缓存从24位闪存地址空间读取的数据，该地址空间在RP 2040地址空间中镜像多次，每个别名具有不同的缓存行为。系统地址的8个MSB用于段解码，剩下24位用于闪存寻址，因此支持的最大闪存大小（用于XIP操作）为16 MB。可用的镜像包括：

* 0x 10...XIP访问，可缓存，分配-正常缓存操作
* 0x 11...XIP访问，可缓存，非分配-检查命中，未命中时不更新缓存
* 0x 12...XIP访问，不可缓存，正在分配-不检查命中，始终更新缓存
* 0x 13...XIP访问，不可缓存，不可分配-完全绕过缓存
* 0x15.使用XIP缓存作为SRAM组，在整个段上镜像

如果通过[CTRL](#_bookmark215).EN寄存器位禁用缓存，则所有四个XIP别名（0x10至0x13）将绕过缓存，直接访问闪存这对XIP代码执行性能有很大影响

访问0x15.段会产生总线错误，除非通过清除[CTRL](#_bookmark215). EN禁用高速缓存一旦缓存被禁用，该区域将作为额外的16kB SRAM存储体。读和写是一个周期，但是在连续的写-读序列上存在等待状态，即，没有写转发缓冲器。

* + - 1. 缓存刷新和维护

[FLUSH](#_bookmark216)寄存器允许刷新整个缓存内容如果软件对闪存内容进行了重新编程，并且需要在不执行重新启动的情况下清除陈旧的数据和代码，则这是必要的缓存刷新可以通过向[FLUSH](#_bookmark216)写入1手动触发，也可以在XIP块退出重置时自动触发。刷新是通过使用内部计数器将缓存标记存储器归零来实现的，这需要超过1024个时钟周期（16kB

总大小/每行8字节/每组2路

在访问闪存数据的同时刷新缓存（可能在一个内核上启动刷新，而另一个内核可能正在执行来自闪存的代码）是一种安全操作，但在刷新正在进行时访问闪存数据的任何主机将被暂停，直到完成。

##### 谨慎

在缓存刷新过程中，不得写入缓存即SRAM别名（0x15...）在首次写入之前，如果最近启动了缓存刷新（例如，通过看门狗重置），则建议从[FLUSH](#_bookmark216)在刷新过程中写入缓存即SRAM可能会损坏数据存储器内容。

一个完整的缓存刷新会大大减缓后续的代码执行，直到缓存再次“预热”。还有一种替代方案，它允许仅对应于某个地址范围的缓存内容无效。对0x10.镜像的写操作将在缓存中查找寻址位置，并删除找到的任何匹配条目因此，写入地址范围中的所有字对齐位置（例如，刚刚被擦除和重新编程的闪存扇区）消除了该范围中陈旧缓存数据的可能性，而不会受到完全缓存刷新的影响

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/flash/cache\_perfctr/flash\_cache\_perfctr.c](https://github.com/raspberrypi/pico-examples/blob/master/flash/cache_perfctr/flash_cache_perfctr.c#L30-L55)第30-55

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

//刷新缓存以确保我们第一次访问test\_data时错过

xip\_ctrl\_hw->flush =1;

同时（！(xip\_css\_hw->stat XIP\_STAT\_FLUSH\_READY\_BITS））tight\_loop\_contents（）;

//清除计数器（写入任何值以清除）

xip\_center\_hw->ctr\_acc =1;

xip\_center\_hw->ctr\_hit =1;

（void）\*test\_data\_ptr;

check（xip\_clog\_hw->ctr\_hit ==0 xip\_clog\_hw->ctr\_acc ==1，“首次访问数据应失败”）;

（void）\*test\_data\_ptr;

check（xip\_clog\_hw->ctr\_hit ==1 xip\_clog\_hw->ctr\_acc ==2，“第二次访问数据应该命中”）;

//写入以使单个缓存行无效（64位）

//写入必须定向到可缓存、可分配的别名（地址0x 10..\_）。 )

\*test\_data\_ptr =0;（void）\*test\_data\_ptr;

check（xip\_ctr\_hw->ctr\_hit ==1 xip\_ctr\_hw->ctr\_acc ==3，“失效后应丢失”）;

（void）\*test\_data\_ptr;

check（xip\_clog\_hw->ctr\_hit ==2 xip\_clog\_hw->ctr\_acc ==4，“无效后的第二次访问应该再次命中”）;

* + - 1. SSI

就地执行功能由SSI接口提供，见第4.10。它支持1、2或4位SPI闪存接口（SPI、DSPI和QSPI），并可在每次XIP访问时插入指令前缀或模式延续位这包括为每次访问发出标准03h串行闪存读取命令的可能性，几乎允许使用任何串行闪存设备最大SPI时钟频率为系统时钟频率的一半

SSI也可以用作标准的基于FIFO的SPI主机，支持DMA。bootrom使用此模式从外部闪存中提取第二阶段引导加载程序（参见[第2.8.1](#_bookmark225)）。总线内插器允许原子设置、清除或XOR操作以与SSI上的其他存储器映射IO相同的方式被发布到SSI控制寄存器。

RP 2040。 这在第2.1.2节中有更详细[的描述](#_bookmark17)。

* + - 1. Flash流媒体和辅助总线从属设备

由于闪存通常比SRAM大得多，因此将数据块从闪存流式传输到内存中通常很有用当软件在前台做其他事情时，让DMA在后台传输这些数据是很方便的，如果代码可以在这发生的同时继续从闪存执行，那就更方便了

这与标准的XIP操作不能很好地相互作用，因为当SSI执行串行传输时，DMA上强制的长时间总线停顿这些停顿对于处理器来说是可以容忍的，因为有序处理器在等待指令提取引退时往往没有更好的事情可做，并且因为典型的代码执行往往具有比不经常访问的数据的批量流式传输高得多的缓存命中率。相比之下，停止DMA会阻止任何*其他*活动的DMA通道在此期间取得进展，从而降低整体DMA吞吐量。

STREAM\_ADDR和STREAM\_CTR寄存器用于对闪存读取的线性序列进行编程，XIP子系统将以尽力而为的方式在后台执行为了最大限度地减少在流正在进行的同时对从闪存执行的代码的影响，流硬件对SSI的访问优先级低于常规XIP访问，并且在最后一次XIP缓存未命中和恢复流之间存在短暂的冷却（七个周期）。这有助于避免在XIP缓存未命中时增加初始访问延迟。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/flash/xip\_stream/flash\_xip\_stream.c](https://github.com/raspberrypi/pico-examples/blob/master/flash/xip_stream/flash_xip_stream.c#L45-L48)第45-48

45

46

47

48

同时（！(xip\_clog\_hw->stat&XIP\_STAT\_FIFO\_EMPTY））（void）xip\_clog\_hw->stream\_fifo;

xip\_candidate\_hw->stream\_addr =（uint32\_t）random\_test\_data[0];

xip\_candidate\_hw->stream\_ctr = count\_of（random\_test\_data）;

流数据被推送到一个小FIFO，它生成DREQ信号，告诉DMA收集流数据。由于DMA在从闪存读取数据之前不会启动读取，因此DMA在访问数据时不会停止。

尽管此方案可确保DREQ置位后数据在流式FIFO中就绪，但如果另一个主机当前在XIP从机上暂停，DMA仍可能暂停，例如：由于缓存未命中。辅助总线从控器解决了这个问题，它是一个简单的总线接口，只提供对流FIFO的此从机暴露在FASTPERI仲裁器上，该仲裁器仅为不生成等待状态的本机AHB-Lite外设提供服务，因此DMA在访问此地址的FIFO时将永远不会出现停顿（假设它具有高总线优先级）。

Pico示例：[https://github.com/raspberrypi/pico-examples/blob/master/flash/xip\_stream/flash\_xip\_stream.c](https://github.com/raspberrypi/pico-examples/blob/master/flash/xip_stream/flash_xip_stream.c#L58-L70)第58-70

58

59

60

61

62

63

64

65

66

67

68

69

70

int dma\_chan =0;

dma\_channel\_config cfg = dma\_channel\_get\_default\_config（dma\_chan）; channel\_config\_set\_read\_increment（cfg，false）; channel\_config\_set\_write\_increment（cfg，true）; channel\_config\_set\_dreq（cfg，DREQ\_XIP\_STREAM）;dma\_channel\_configure（

DMA\_chan，

&cfg，

（void\*）buf，

//写地址

（const void\*）XIP\_AUX\_BASE，*//读取地址*

count\_of（random\_test\_data），*//传输计数*

真

//立即开始！

);

* + - 1. 性能计数器

XIP子系统提供两个性能计数器。它们的大小为32位，在达到0xffffffff时饱和，并通过写入任何值来清除他们计算：

1. 对任何别名的XIP访问总数
2. 导致缓存命中的XIP访问数

对于常见用例，这允许分析缓存命中率。

* + - 1. XIP寄存器列表

表154.对妇女的歧视XIP寄存器

表155.对妇女的歧视CTRL寄存器

XIP寄存器从基址0x14000000（在SDK中定义为[XIP\_CTRL\_BASE](#_bookmark37)）开始。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [Ctrl](#_bookmark215) | 高速缓存控制 |
| 0x04 | [冲洗](#_bookmark216) | 缓存刷新控制 |
| 0x08 | [STAT](#_bookmark217) | 缓存状态 |
| 0x0C | [CTR\_HIT](#_bookmark218) | 缓存命中计数器 |
| 0x10 | [CTR\_ACC](#_bookmark219) | 高速缓存存取计数器 |
| 0x14 | [STREAM\_ADDR](#_bookmark220) | 先进先出流地址 |
| 0x18 | [STREAM\_CTR](#_bookmark221) | 先进先出流控制 |
| 0x1c | [STREAM\_FIFO](#_bookmark222) | 先进先出流数据 |

### [XIP](#_bookmark214)：CTRL寄存器

**偏移**：0x00

描述

高速缓存控制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:四 | Reserved. | - | - | - |
| 3 | 关闭电源 | 当为1时，高速缓存关闭。他们保留国家，  但是不能被访问。 这降低了静态功耗。  将1写入此位将强制CTRL\_EN为0，即高速缓存不能  在断电时启用  当出现以下情况时，缓存即SRAM访问将产生总线错误响应  高速缓存被断电。 | RW | 0x0 |
| 2 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 1 | 错误\_BADWRITE | 当为1时，写入0x0以外的任何别名（缓存、分配）  将产生总线故障。当为0时，这些写入将被忽略。  在任何一种情况下，对0x0别名的写入都将在标记匹配时解除分配  和往常一样。 | RW | 0x1 |
| 0 | EN | 当为1时，启用缓存。禁用缓存时，所有XIP访问  将直接进入闪存，而不查询缓存。启用后  可缓存XIP访问将查询缓存，闪存将  如果标签匹配并且设置了有效位，则不访问。  如果启用缓存，则缓存即SRAM访问对  缓存数据RAM，并将产生总线错误响应。 | RW | 0x1 |

表156.冲洗

寄存器

表157.对妇女的歧视STAT寄存器

### [XIP](#_bookmark214)：FLUSH寄存器

**偏移**：0x04

描述

缓存刷新控制

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:一 | Reserved. | - | - |
| 0 | 写入1以刷新缓存。这将清除标记内存，但  数据存储器保留其内容。（这意味着缓存即SRAM的内容不受刷新或重置的影响。）  读取将保持总线（使处理器停止），直到刷新完成。 或者，可以轮询STAT直到完成。 | SC | 0x0 |

### STAT寄存器

**偏移**：0x08

描述

缓存状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:三 | Reserved. | - | - | - |
| 2 | FIFO\_FULL | 为1时，表示XIP流FIFO已满。  流FIFO是2个条目深，因此满和空  标志允许确定其级别。 | RO | 0x0 |
| 1 | FIFO\_EMPTY | 为1时，表示XIP流FIFO完全为空。 | RO | 0x1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 0 | 冲洗\_就绪 | 在缓存刷新过程中读取为0，否则读取为1。  每当重置XIP块时都会刷新缓存，  当通过FLUSH寄存器请求时。 | RO | 0x0 |

表158.对妇女的歧视CTR\_HIT注册

表159.对妇女的歧视CTR\_ACC寄存器

表160.第160页STREAM\_ADDR

寄存器

表161.对妇女的歧视STREAM\_CTR寄存器

### [XIP](#_bookmark214)：CTR\_HIT注册

**偏移**：0x0c

描述

缓存命中计数器

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 一个32位饱和计数器，在每次缓存命中时递增  即当XIP访问直接从缓存数据提供服务时写入任何要清除的值。 | WC | 0x00000000 |

### [XIP](#_bookmark214)：CTR\_ACC寄存器

**偏移**：0x10

描述

高速缓存存取计数器

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 一个32位饱和计数器，在每次XIP访问时递增，无论缓存是否命中。这包括不可缓存的访问。写入任何要清除的值。 | WC | 0x00000000 |

### [XIP](#_bookmark214)：STREAM\_ADDR寄存器

**偏移**：0x14

描述

先进先出流地址

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:二 | 下一个要从闪存流到流FIFO的字的地址。  每次闪存访问后自动递增  在开始流式读取之前，在此处写入初始访问地址 | RW | 0x00000000 |
| 一比零 | Reserved. | - | - |

### [XIP](#_bookmark214)：STREAM\_CTR寄存器

**偏移**：0x18

描述

先进先出流控制

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:22 | Reserved. | - | - |

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 21:0 | 写入非零值以启动流式读取。然后，这将在后台进行，使用闪存空闲周期将线性数据块从闪存传输到流式FIFO。随着流的进展自动递减（一次1），并在达到0时停止。  写入0以停止正在进行的流，并丢弃任何正在进行的读取，以便可以立即启动新的流（在排空FIFO并重新初始化STREAM\_ADDR之后） | RW | 0x000000 |

表162.对妇女的歧视STREAM\_FIFO

寄存器

### [XIP](#_bookmark214)：STREAM\_FIFO寄存器

**偏移量**：0x1c

描述

先进先出流数据

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 流数据在此处缓冲，供系统DMA检索。  此FIFO也可通过XIP\_AUX从机访问，以避免DMA暴露于其他XIP流量导致的总线停顿。 | RF | 0x00000000 |

* 1. 引导序列

RP 2040的几个组件一起工作，以达到处理器脱离复位并能够运行bootrom的点（[第2.8节](#_bookmark224)）。bootrom是内置在芯片中的软件，执行引导序列的“处理器控制”部分我们将把处理器运行之前的步骤称为“硬件控制”引导序列。

硬件控制的引导顺序如下：

* 芯片上电，RUN引脚为高电平。（如果RUN为低电平，则芯片将保持复位状态。）
* 片内电压调节器（[第2.10](#_bookmark256)）等待数字内核电源（DVDD）稳定
* 启动通电状态机（[第2.13](#_bookmark291)）总结一下顺序：
  + 环形振荡器（第2.17节）启动，为时钟发生器提供时钟源\_sys和

目前，WRF\_ref以相对较低的频率（通常为6.5MHz）运行。

* + 复位控制器（[第2.14](#_bookmark302)）、就地执行硬件（[第2.6.3](#_bookmark213)）、存储器（[第2.6.2](#_bookmark211)和[第2.6.1](#_bookmark210)）、总线结构（[第2.1](#_bookmark13)）和处理器子系统（[第2.3](#_bookmark43)）从复位中取出
  + 处理器核心0和核心1开始执行bootrom（[第2.8节](#_bookmark224)）。
  1. Bootrom

Bootrom大小限制为16kB。它包括：

* 处理器核心0初始引导序列。
* 处理器内核1低功耗等待和启动协议。
* USBMSC类兼容的引导加载程序，支持[UF 2](https://github.com/Microsoft/uf2)，用于将代码/数据下载到闪存或RAM。
* USBPICOBOOT引导加载程序接口，用于高级管理。
* 编程和操作外部闪存的例程
* 快速浮点库。
* 快速位计数/操作功能。
* 快速存储器填充/复制功能。

Bootrom源代码

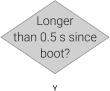
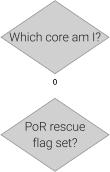
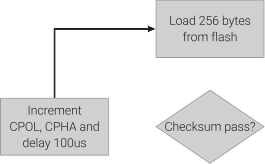
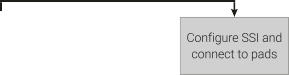
RP 2040 bootrom的完整源代码可以在<https://github.com/raspberrypi/pico-bootrom>上找到。

这包括bootrom的版本1、2和3，分别对应于B0、B1和B2硅版本

* + 1. 处理器控制的引导序列

引导序列的流程图如[图15所示](#_bookmark226)。

*图15. RP2040启动顺序*



在[第2.7](#_bookmark223)中描述的硬件控制的引导序列之后，处理器控制的引导序列开始：

* 重置为两个处理器释放：都进入ROM在同一位置
* 处理器检查SIO。CPUID
  + 处理器1进入睡眠状态（WFE，启用SCR.SLEEPDEEP），并保持睡眠状态，直到通过邮箱被用户代码唤醒
  + 处理器0继续从ROM执行
* 如果上电事件来自救援DP，清除此标志并**立即停止**
  + 调试主机（启动救援的主机）将提供进一步的指示。
* 如果看门狗划痕寄存器设置为指示SRAM中存在预加载代码，则跳转到该代码
* 检查SPI CS引脚是否为低电平（“bootrom按钮”），如果是，则跳过闪存引导。
* 在QSPI引脚上设置IO多路复用、焊盘控制，并初始化Synopsys SSI以用于标准SPI模式
* 在闪存仍处于XIP模式且未重新通电的情况下，发出XIP退出序列
* 将256字节从SPI复制到内部SRAM（SRAM 5）并检查有效的CRC 32校验和
* 如果校验和通过，则假设我们加载的是有效的闪存第二级
* 开始执行从SRAM（SRAM 5）加载的代码
* 如果在尝试引导0.5秒后未在SPI中找到有效映像，则转至USB设备引导
* USB设备启动：显示为USB大容量存储设备
  + 可以编程SPI闪存，或直接加载到SRAM中并运行，通过拖放UF2格式的图像。
  + 还支持扩展的PICOBOOT接口
    - 1. 看门狗启动

看门狗引导允许用户安装自己的引导处理程序，并在非POR/BOR重置时将控制从主引导序列转移它还简化了JTAG测试接口上的运行代码。它可识别写入看门狗高位暂存寄存器的以下值：

* 划痕4：魔术数字0xb007c0d3
* 划痕5：入口点与magic-0xb 007 c 0 d3（0x 4ff 83 f2 d）进行XOR
* Scratch 6：堆栈指针
* 第7章：切入点

如果任何一个幻数不匹配，则不会进行看门狗引导。如果数字匹配，则Bootrom在转移控制之前将零擦除4，以便该行为在随后的重新引导中不会持续

* + - 1. 闪存启动顺序

热闪存启动的主要挑战之一是强制外部闪存从XIP模式转换为接受标准SPI命令的模式。在未知闪存上中断XIP没有标准方法。Bootrom提供了具有广泛兼容性的尽力而为序列，如下所示

* CSn=1，IO[3：0]=4'b0000（通过下拉避免争用），发出×32个时钟
* CSn=0，IO[3：0]=4'b1111（通过上拉避免争用），发出×32个时钟
* CSn=1
* CSn=0，MOSI=1'b1（驱动低Z，所有其他IO高Z），发出×16个时钟

这是为了错过赛普拉斯，美光和华邦零件上的XIP连续代码。如果设备已处于SPI模式，则它将此序列解释为两个FFh NOP指令，应忽略该指令

由于这只是尽力而为，可能会有一些设备顽固地保持在XIP模式。那么有两种选择：

* 使用效率较低的XIP模式，其中每次传输都有一个SPI指令前缀，因此闪存器件在SPI模式下保持通信。
* 引导代码在SRAM中安装兼容的XIP退出序列，并配置看门狗，以便热引导将直接跳转到该序列，而不是我们的固定序列。

发出XIP退出序列后，Bootrom尝试使用标准03h在第二阶段从闪存读取

串行读取命令，几乎普遍支持。由于Bootrom是不可变的，因此它的目标是兼容性

而不是业绩。

* + - 1. Flash第二阶段

闪存第二阶段必须配置SSI和外部闪存，以获得最佳的就地执行性能。这包括接口宽度、SCK频率、SPI指令前缀和用于仅地址数据模式的XIP连续码。一般来说，可以在外部闪存上执行某些操作，因此每次访问时不需要指令前缀，只需使用数据响应地址

在为连接的闪存设备正确配置SSI之前，无法通过XIP地址窗口访问闪存此外，Synopsys SSI在没有禁用之前根本无法重新配置。因此，第二阶段必须通过bootrom从闪存复制到SRAM，并在SRAM中执行。

或者，第二阶段可以简单地将来自外部闪存的映像映射到SRAM中，而不配置就地执行。

这是第二阶段的**唯一**工作。所有其他芯片设置（例如PLL，电压调节器）可以通过XIP接口执行的平台初始化代码执行，一旦第二阶段已经运行。

* + - * 1. 校验和

从闪存加载的图像的最后四个字节（我们希望这是有效的闪存第二阶段）是前252字节的CRC 32校验和校验和的参数为：

* 多项式：0x04c11db7
* 输入反射：无
* 输出反射：无
* 初始值：0xffffffff
* 最终XOR：0x0000000
* 校验和值在图像末尾显示为小端整数

Bootrom进行128次尝试，每次大约4ms，总共大约0.5秒，然后放弃并进入USB代码，以加载和校验具有不同SPI参数的第二阶段如果它看到校验和通过，它将立即跳转到包含闪存第二级的252字节有效载荷

* + 1. 在处理器核心1上启动代码

如[第2.8.1节](#_bookmark225)的介绍中所述，复位后，处理器内核1“休眠（WFE启用SCR.SLEEPDEEP）并保持休眠状态，直到通过邮箱被用户代码唤醒

如果您使用的是SDK，那么您可以简单地使用multicore\_launch\_core1函数在处理器内核1上启动代码但是，本节将介绍您自己在处理器内核1上启动代码的过程。

在处理器内核1上开始运行的过程涉及两个内核通过状态机以锁步移动，该状态机通过在处理器间FIFO上传递消息来协调。该状态机被设计为足够鲁棒以应对最近复位的处理器核心1，该处理器核心1可以在其引导代码中的任何地方，直到并且包括进入睡眠。结果，该过程可以在处理器核心1已经被重置（通过系统重置，或者明确地仅重置处理器核心1）之后的任何点处执行

下面的C代码是描述该过程的最简单方法

//要通过FIFO从核心0到核心1顺序发送的值

//

//vector\_table是 VTOR寄存器的值

//sp是初始堆栈指针（SP）

//entry是初始程序计数器（PC）（不要忘记设置拇指位！）

const uint32\_tcmd\_sequence[]=

{0，0，1，（uintptr\_t）向量表，（uintptr\_t）sp，（uintptr\_t）条目};

int n =0;do{

int sum =sum\_sum [sum];

//总是在发送0之前清空读FIFO（从核心1）

如果（！（int）{

//从读FIFO中丢弃数据直到清空

multicore\_fifo\_drain（）;

//执行SEV，因为核心1可能正在等待FIFO空间

sev（）;

}

//将32位值写入FIFO

multicore\_fifo\_push\_blocking（cmd）;

//一旦可用，从读FIFO读取32位值

uint32\_tresponse =multicore\_fifo\_pop\_blocking（）;

//如果响应正确（echo-d值），则移动到下一个状态，否则重新开始

seq = cmd ==响应？seq+1：0;

}while（seqcount\_of（cmd\_sequence））;

* + 1. Bootrom内容

表163.对妇女的歧视固定（已知）地址上的Bootrom内容

一些bootrom专用于执行引导序列和USB引导接口。bootrom中也有对用户程序有用 [表163](#_bookmark230)显示了Bootrom中前几个字的固定内存布局，这些字有助于定位Bootrom中的其他内容。

|  |  |  |
| --- | --- | --- |
| 地址 | 内容 | 描述 |
| 0x00000000 | 32位指针 | 初始引导堆栈指针 |
| 0x00000004 | 32位指针 | 引导复位处理程序函数指针 |
| 0x0000008 | 32位指针 | 引导NMI处理程序函数的指针 |
| 0x000000c | 32位指针 | 引导硬故障处理程序函数的指针 |
| 0x00000010 | 'M'，'u'，0x01 | 魔法 |
| 0x00000013 | 字节 | Bootrom版本 |
| 0x00000014个字符 | 16位指针 | 指向公共函数查找表的指针（rom\_func\_table） |
| 0x0000016 | 16位指针 | 指向公共数据查找表（rom\_data\_table）的指针 |
| 0x0000018 | 16位指针 | 指向辅助函数的指针（rom\_table\_lookup（）） |

* + - 1. Bootrom函数

Bootrom包含许多公共函数，这些函数提供了在设备上没有任何其他代码的情况下可能需要的有用的RP 2040功能，以及某些关键功能的高度优化版本，否则这些功能将占用大多数用户二进制文件的空间。

这些函数通常由SDK提供给用户，但是提供了一个较低级别的方法来定位它们（它们的位置可能会随着每个Bootrom版本而变化）并直接调用它们

假设从地址0x00000010开始的三个字节是（“M”，“u”，0x01），则从偏移开始的三个半字

0x00000014有效。

这三个值可用于动态定位Bootrom中的其他函数或数据偏移量0x00000013处的版本字节是信息性的，不应用于推断任何函数的确切位置

以下来自SDK的代码显示了如何使用三个16位指针来查找其他函数或数据。

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/pico\_bootrom/bootrom.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/pico_bootrom/bootrom.c#L11-L19)第11-19

1. void\*rom\_func\_lookup（ uint32\_tcode）{
2. returnrom\_func\_lookup\_inline（code）;

13}

14

1. void\*rom\_data\_lookup（ uint32\_tcode）{
2. rom\_table\_lookup\_fn rom\_table\_lookup =（rom\_table\_lookup\_fn）rom\_hword\_as\_ptr（0x18）;
3. uint16\_t\*data\_table =（ uint16\_t\*）rom\_hword\_as\_ptr（0x16）;
4. 返回rom\_table\_lookup（data\_table，code）;

19}

code参数对应于下表中的*CODE*值，计算如下：

uint32\_trom\_table\_code（char c1，char c2）{return（c2<<8）|c1;

}

* + - * 1. 快速位计数/操作功能

表164.对妇女的歧视 快速位计数/操作功能。

这些是常见位计数/操作函数的优化版本。

通常，您不需要直接调用这些方法，因为SDK*pico\_bit\_ops*库默认会替换相应的标准编译器库函数，以便builtin\_popcount或clzdi2等标准函数自动使用相应的Bootrom实现（有关更多详细信息，请参阅[pico\_bit\_ops](https://www.raspberrypi.com/documentation/pico-sdk/runtime.html#pico_bit_ops)

这些函数在bootrom版本1（V1）和版本2（V2）之间的速度略有变化。

|  |  |  |  |
| --- | --- | --- | --- |
| 代码 | 周期平均值V1 | 周期平均值V2/V3 | 描述 |
| “P”，“3” | 18 | 20 | uint32\_t\_popcount32（uint32\_t值） |
| 返回值中1位数的计数。 |
| “R”、“3” | 21 | 22 | uint32\_t\_reverse32（uint32\_t值） |
| 以相反的顺序返回值 |
| “L”、“3” | 13 | 9.6 | uint32\_t\_clz32（uint32\_t值） |
| 返回值的连续高位0位数。 如果value为0，则返回32。 |
| “T”、“3” | 12 | 11 | uint32\_t\_ctz32（uint32\_t值） |
| 返回值的连续低位0位数。 如果value为0，则返回32。 |

* + - * 1. 快速批量存储器填充/复制功能

这些是大多数语言运行时通常提供的高度优化的批量内存填充和复制函数

通常，您不需要直接调用这些方法，因为SDK*pico\_pico\_ops*库默认会替换相应的标准ARM EABI函数，因此标准C库函数（例如memcpy或memset）使用Bootrom

表165.对妇女的歧视优化的批量内存填充/复制功能

表166.对妇女的歧视闪存访问功能

自动实现（有关更多详细信息，请参见[pico\_pico\_ops](https://www.raspberrypi.com/documentation/pico-sdk/runtime.html#pico_mem_ops)）。

|  |  |
| --- | --- |
| 代码 | 描述 |
| “M”、“S” | uint8\_t \*\_memset（uint8\_t \*ptr，uint8\_t c，uint32\_tn） |
| 将ptr处开始的n个字节设置为值c并返回ptr。 |
| “S”、“4” | uint32\_t \*\_memset4（uint32\_t \*ptr，uint8\_t c，uint32\_tn） |
| 将ptr处开始的n个字节设置为值c并返回ptr。注意，这是一个稍微更有效的  \_memset，只能在ptr是字对齐的情况下使用 |
| “M”、“C” | uint8\_t \*\_memcpy（uint8\_t \*dest，uint8\_t \*src，uint32\_tn） |
| 将从src开始的n个字节复制到dest并返回dest。如果区域重叠，则结果未定义 |
| “C”、“4” | uint8\_t \*\_memcpy44（uint32\_t \*dest，uint32\_t \*src，uint32\_tn） |
| 将从src开始的n个字节复制到dest并返回dest。如果区域重叠，则结果未定义。请注意，这是\_memcpy的一个稍微更有效的变体，仅在dest和src是字对齐的情况下使用。 |

* + - * 1. 闪存访问功能

这些都是低级别的Flash助手函数。

|  |  |
| --- | --- |
| 代码 | 描述 |
| “I”、“F” | void\_connect\_internal\_flash（void） |
| 将所有QSPI焊盘控制恢复到默认状态，并将SSI连接到QSPI焊盘 |
| “E”、“X” | void\_flash\_exit\_xip（void） |
| 首先为串行模式操作设置SSI，然后发出固定的XIP退出序列，如[第2.8.1.2](#_bookmark227)所述。请注意，bootrom代码使用IO强制逻辑来驱动CS引脚，在将SSI返回到XIP模式之前必须清除CS引脚（例如，通过调用\_flash\_flush\_cache）。此函数将SSI配置为固定的SCK时钟除数/6。 |
| “R”、“E” | void\_flash\_range\_erase（uint32\_t addr，size\_t count，uint32\_t block\_size，uint8\_tblock\_cmd） |
| 从addr开始计数字节（从闪存开始的或者，传递块擦除命令  例如D8 h块擦除，以及此命令擦除的块的大小-此功能将在可能的情况下使用较大的块擦除，以获得更高的擦除速度。addr必须与4096字节的扇区对齐，并且计数必须是4096字节的倍数。 |
| “R”、“P” | void flash\_range\_program（uint32\_t addr，const uint8\_t \*data，size\_tcount） |
| 将数据编程到从addr（从闪存开始的偏移量）开始的一系列闪存地址，并计算字节大小。addr必须与256字节边界对齐，count必须是256的倍数 |
| “F”、“C” | void\_flash\_flush\_cache（void） |
| 刷新并启用XIP缓存。同时清除QSPI CSn上的IO强制，以便SSI可以正常驱动 |
| “C”、“X” | void\_flash\_enter\_cmd\_xip（void） |
| 配置SSI以在每次**XIP**访问时生成标准03h串行读取命令（具有24个地址位）这是一个非常慢的XIP配置，但得到了非常广泛的支持。调试器在执行闪存擦除/编程操作后调用此函数，以便调试主机可以看到新编程的代码和数据，而无需确切知道连接的闪存设备类型。 |

从用户代码中擦除闪存扇区的典型调用序列为：

* \_connect\_internal\_flash
* \_flash\_exit\_xip
* \_flash\_range\_erase（addr，1 12，1 16，0xd8）
* \_flash\_flush\_cache
* 对\_flash\_enter\_cmd\_xip的调用或对先前复制到SRAM中的闪存第二阶段的调用请注意，在此序列中的第一次和最后一次调用之间，SSI**不**处于可以处理XIP访问的状态SDK hardware\_flash库隐藏了这些细节。
  + - * 1. 支持功能

表167.对妇女的歧视支持功能

表168.对妇女的歧视辅助功能

这两个方法简化了在设备上调用代码然后将控制返回给调试器的任务。

|  |  |
| --- | --- |
| 代码 | 描述 |
| D、T | 调试蹦床 |
| 返回时中断的简单调试器蹦床  此方法帮助调试器调用ROM例程，而无需设置硬件断点。根据ABI，函数地址在r7中传递，参数通过r0...r3  此方法不返回，而是在最后执行BKPT#0。 |
| D、E | \_debug\_trampoline\_end |
| 这是debug\_trampoline的最后一条BKPT #0指令的地址。这可以与程序计数器进行比较，以检测debug\_trampoline调用的完成 |

* + - * 1. 辅助功能

这些剩余的函数不适合其他类别，并通过pico\_bootrom库在SDK中公开（请参阅[pico\_bootrom](https://www.raspberrypi.com/documentation/pico-sdk/runtime.html#pico_bootrom)）。

|  |  |
| --- | --- |
| 代码 | 描述 |
| “U”、“B” | void\_reset\_to\_usb\_boot（uint32\_t gpio\_activity\_pin\_mask，uint32\_tdisable\_interface\_mask） |
| 重置RP 2040并使用看门狗工具在BOOTSEL模式下重新启动   * gpio\_activity\_pin\_mask用于通过USB大容量存储设备的GPIO连接LED启用“活动指示灯”   + 0冷启动时不使用引脚。   + 否则，设置一个位，指示哪个GPIO引脚应设置为输出，并在主机有大容量存储活动时升高。 * disable\_interface\_mask可用于控制暴露的USB接口：   + 0启用两个接口（根据冷启动）   + 1禁用USB大容量存储接口（参见[第2.8.4](#_bookmark235)）   + 2禁用USB PICOBOOT接口（参见[第2.8.5](#_bookmark237)） |

|  |  |
| --- | --- |
| “W”，“V” | 等待向量 |
| 这是核心1在复位时输入的方法，等待核心0启动在很少的情况下，您应该调用此方法（重置核心1要好得多此方法不会返回，并且应该只在核心1上调用。 |
| “E”、“C” | 弃用 |
| 请勿使用可能不存在的此功能。 |

* + - 1. 快速浮点库

Bootrom包含优化的单精度浮点实现。此外，V2还包含一个优化的双精度浮点实现。每个精度的函数指针保存在通过rom\_data\_lookup表找到的表结构中（请参见[第2.8.3.3](#_bookmark234)）。

* + - * 1. 实现细节

在速度和尺寸之间总是有一个权衡虽然浮点例程的总体目标是在较小的占用空间内实现良好的性能，但重点更多地放在提高基本运算（加、减、乘、除和平方根）的性能上，而更多地放在减少科学函数（三角函数、多项式和指数）的占用空间上

IEEE单精度和双精度数据格式在整个过程中使用，但为了减少代码大小，输入反规范被视为零，输入NaN被视为无穷大，输出反规范被刷新为零，输出NaN被渲染为无穷大。仅支持舍入到最近值、连系偶数舍入模式。不支持陷阱。

这五个基本操作返回的结果总是正确舍入的。

科学函数总是返回精确结果的1 ULP（最后一个单位）范围内的结果在许多情况下，结果更好。

科学函数是使用内部定点表示计算的，因此在将结果转换回浮点需要大的归一化偏移的情况下，准确性（以ULP误差而不是绝对值测量）较差例如，当计算接近pi的倍数的值的正弦、接近pi/2的奇数倍的值的余弦或接近1的值的对数时，会发生这种情况当结果非常大时，正切函数的精度也虽然覆盖这些情况是可能的，但它会大大增加代码占用空间，并且很少有类型的程序在这些情况下的准确性是必不可少的。

正弦、余弦和正切函数也只能在有限的范围内正确运行：单精度参数x为-128x+128，双精度参数x为-1024 x +1024这是为了避免（至少实际上）在代码中以高精度存储pi值的需要如果需要的话，可以在库外部对更大范围的参数进行精确的范围缩减，但是也有一些情况需要这样做

##### 注意

SDK的cos/sin函数执行这种范围缩减，因此接受整个范围的参数，尽管对于这些范围之外的输入速度较慢。

* + - * 1. 功能

这些函数遵循标准ARM EABI传递浮点值。

您不需要直接调用这些方法，因为默认使用的SDK[pico\_float](https://www.raspberrypi.com/documentation/pico-sdk/runtime.html#pico_float)和[pico\_double](https://www.raspberrypi.com/documentation/pico-sdk/runtime.html#pico_double)库会替换*ARM EABI Float函数*，这样C/C++级别的代码（或以C实现的*MicroPython*等语言间接代码）会自动使用这些Bootrom函数进行相应的浮点运算。

其中一些函数的行为与相应的C库函数不完全相同。因此，如果您使用SDK，强烈建议您仅使用常规math.h函数或pico/float. h或pico/double. h中的函数，而不要尝试直接调用bootrom

请注意，bootrom的版本1（V1）中不支持双精度浮点数，但上面提到的SDK中的[pico\_double](https://www.raspberrypi.com/documentation/pico-sdk/runtime.html#pico_double)库将负责拉入V1所需的任何额外代码

##### 注意

有关在SDK中使用浮点的更多信息，以及真实世界的计时（还请注意，某些转换函数在SDK中重新实现以更快），请参阅[浮点支持](https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf#section_floating_point)。

*表169.对妇女的歧视单精度浮点函数表。 计时是随机（最坏情况）输入的平均时间。该ROM版本中不存在计时为N/A的函数，并且函数指针应被视为无效。从偏移量0x54开始的函数（和表条目）仅存在于V2ROM中。*

|  |  |  |  |
| --- | --- | --- | --- |
| 偏移 | V1周期（平均值） | V2/V3  循环次数（平均值） | 描述 |
| 所有版本的bootrom通用的功能 | | | |
| 0x00 | 71 | 71 | float\_fadd（float a，floatb） |
| Return a +B |
| 0x04 | 74 | 74 | float\_fsub（float a，floatb） |
| 返回a-b |
| 0x08 | 69 | 58 | float\_fmul（float a，floatb） |
| 返回a \*b |
| 0x0C | 71 | 71 | float\_fdiv（float a，floatb） |
| 返回a /b |
| 0x10 | N/A | N/A | 弃用 |
| 不要使用此功能 |
| 0x14 | N/A | N/A | 弃用 |
| 不要使用此功能 |
| 0x18 | 63 | 63 | float\_fsqrt（floatv） |
| 如果v为负，则返回或*-无限*（注意V1在这种情况下返回*+Infinity* |
| 0x1c | 37 | 40 | int\_float2int（floatv） |
| 将浮点数转换为有符号整数，向*-Infinity*舍入，并将结果限制在-0x80000000到0x 7 FFFFFFF范围 |
| 0x20 | 36 | 39 | int n（int i，intn） |
| 将浮点数转换为有符号定点整数表示，其中*n*指定二进制点在所得定点表示中的位置-例如。  \_float2fix（0.5f，16）== 0x8000. 此方法向*-Infinity*舍入，并将结果整数限制在-0x80000000到0x 7 FFFFFFF范围 |
| 0x24 | 38 | 39 | uint\_float2uint（floatv） |
| 将浮点数转换为无符号整数，四舍5入到*-Infinity*，并将结果限制在0x 00000000到0xFFFFFFFF范围 |

|  |  |  |  |
| --- | --- | --- | --- |
| 0x28 | 38 | 38 | int n（int i，intn） |
| 将浮点数转换为无符号定点整数表示，其中*n*指定二进制点在所得定点表示中的位置，例如。  \_float2ufix（0.5f，16）== 0x8000. 此方法向*-Infinity*舍入，并将结果整数限制在0x 0000000到0xFFFFFFFF的范围 |
| 0x2c | 55 | 55 | public int findDuplicate（intv） |
| 将有符号整数转换为最接近的浮点值，并舍入为偶数 |
| 0x30 | 53 | 53 | int n（int v，intn） |
| 将有符号定点整数表示形式转换为最接近的浮点值，并舍入为偶数。*n*指定定点中二进制点的位置 |
| 0x34 | 54 | 54 | float\_uint2float（uint32\_tv） |
| 将无符号整数转换为最接近的浮点值，并在平局时舍入为偶数 |
| 0x38 | 52 | 52 | int n（int v，intn） |
| 将无符号定点整数表示形式转换为最接近的浮点值，并舍入为偶数。*n*指定定点中二进制点的位置 |
| 0x3c | 603 | 587 | float\_fcos（浮动角度） |
| 返回*角度*的余弦。*角度*以弧度为单位，并且必须在-128到128的范围内 |
| 0x40 | 593 | 577 | float\_fsin（浮动角度） |
| 返回*角度*的正弦值。*角度*以弧度为单位，并且必须在-128到128的范围内 |
| 0x44 | 669 | 653 | float\_ftan（浮动角度） |
| 返回*角度*的切线。*角度*以弧度为单位，并且必须在-128到128的范围内 |
| 0x48 | N/A | N/A | 弃用 |
| 不要使用此功能 |
| 0x4c | 542 | 524 | float\_fexp（floatv） |
| 返回v的指数值，即所以 |
| 0x50 | 810 | 789 | float\_fln（floatv） |
| 返回v的自然对数。如果返回*-无限* |
| 仅V2/V3 bootrom中存在的函数（和表项） | | | |
| 0x54 | N/A | 25 | int\_fcmp（float a，floatb） |
| 比较两个浮点数，返回：   * 0 if a ==b * -1如果ab * 1，如果a>b |
| 0x58 | N/A | 667 | float\_fatan2（float y，floatx） |
| 使用参数的符号计算y/x的反正切以确定正确的象限 |

|  |  |  |  |
| --- | --- | --- | --- |
| 0x5c | N/A | 62 | float\_int642float（int64\_tv） |
| 将有符号的64位整数转换为最接近的浮点值，并四舍五入为偶数 |
| 0x60 | N/A | 60 | float\_fix642float（int v，intn） |
| 将有符号定点64位整数表示形式转换为最接近的浮点值，并舍入为偶数。*n*指定定点中二进制点的位置 |
| 0x64 | N/A | 58 | float\_uint642float（uint64\_tv） |
| 将无符号64位整数转换为最接近的浮点值，并在平局时舍入为偶数 |
| 0x68 | N/A | 57 | float\_ufix642float（uint64\_t v，intn） |
| 将无符号定点64位整数表示形式转换为最接近的浮点值，并舍入为偶数。*n*指定定点中二进制点的位置 |
| 0x6c | N/A | 54 | \_float2int64 |
| 将浮点数转换为有符号的64位整数，向*-Infinity*舍入，并将结果固定在-0x80000000000000到0x 7 FFFFFFFFFFFF范围 |
| 0x70 | N/A | 53 | \_float2fix64 |
| 将浮点数转换为有符号定点64位整数表示形式，其中*n*  指定二进制点在生成的定点表示中的位置  - 例如\_float2fix（0.5f，16）== 0x8000。此方法向*-Infinity*舍入，并将结果整数限制在-0x8000000000000到0x 7 FFFFFFFFFFFF范围内 |
| 0x74 | N/A | 42 | \_float2uint64 |
| 将浮点数转换为无符号64位整数，向*-Infinity*舍入，并将结果箝位在0x 0000000000000到0xFFFFFFFFFFFF范围 |
| 0x78 | N/A | 41 | \_float2ufix64 |
| 将浮点数转换为无符号定点64位整数表示形式，其中*n*  指定二进制点在结果定点表示中的位置  例如\_float2ufix（0.5f，16）== 0x8000。此方法向*-Infinity*舍入，并将结果整数限制在0x 000000000000到0xFFFFFFFFFFFF范围内 |
| 0x7c | N/A | 15 | double\_float2double（floatv） |
| 将float转换为double |
| 仅在V3 bootrom中提供的功能 | | | |
| 0x48  (uses以前不推荐使用的插槽） |  | 577（V3）  仅限） | float*（，float）*\_fsincos（float angle） |
| 计算*角度*的正弦和余弦。*角度*以弧度为单位，并且必须在-128到128之间。正弦值返回寄存器*r0*（因此是正式函数返回值），余弦值返回寄存器*r1*。这个方法比单独调用\_fsin和\_fcos |

请注意，V2/V3 Bootrom包含用于双精度浮点运算的等效函数表偏移量是相同的，但是在float的地方现在有double（对于float>double转换，反之亦然）

*表170.第170段双精度浮点函数表。时间是平均的*

|  |  |  |
| --- | --- | --- |
| 偏移 | 循环次数（平均值）\* | 描述 |
| 0x00 | 91 | double\_dadd（double a，doubleb） |
| Return a +B |
| 0x04 | 95 | double\_dsub（double a，doubleb） |
| Return a-b |
| 0x08 | 155 | double\_dmul（double a，doubleb） |
| Return a \*b |
| 0x0C | 183 | double\_ddiv（double a，doubleb） |
| Return a/b |
| 0x10 | N/A | 弃用 |
| 不要使用此功能 |
| 0x14 | N/A | 弃用 |
| 不要使用此功能 |
| 0x18 | 169 | double\_dsqrt（doublev） |
| 如果v为负数，则返回或*-无穷大* |
| 0x1c | 75 | int\_double2int（doublev） |
| 将double转换为有符号整数，向*-Infinity*舍入，并将结果限制在-0x80000000到0x 7 FFFFFFF |
| 0x20 | 74 | int n（int i，intn） |
| 将double转换为带符号定点整数表示，其中*n*指定二进制点在所得定点表示中的位置-例如。\_double2fix（0.5f，16）== 0x8000. 此方法向*-Infinity*舍入，并将结果整数限制在-0x80000000到0x 7 FFFFFFF |
| 0x24 | 63 | uint\_double2uint（双v） |
| 将double转换为无符号整数，向*-Infinity*舍入，并将结果固定在0x 00000000到0xFFFFFFFF |
| 0x28 | 62 | int n（int i，intn） |
| 将double转换为无符号定点整数表示，其中*n*指定所得定点表示中二进制点的位置，例如。\_double2ufix（0.5f，16）== 0x8000. 此方法向*-Infinity*舍入，并将结果整数限制在0x 0000000到0xFFFFFFFF |
| 0x2c | 69 | public int findDuplicate（int v） |
| 将有符号整数转换为最接近的double值，并在平局时舍入为偶数 |
| 0x30 | 68 | double\_fix2double（int32\_t v，int n） |
| 将有符号定点整数表示形式转换为最接近的双精度值，并舍入为偶数。*n*指定定点中二进制点的位置，因此 |
| 0x34 | 64 | double\_uint2double（uint32\_t v） |
| 将无符号整数转换为最接近的double值，并在平局时舍入为偶数 |

时间在我们结束

随机（最坏情况）输入。该ROM版本中不存在计时为N/A的函数，并且函数指针应被视为无效。从偏移量0x54开始的函数（和表条目）仅存在于V2和V3 ROM中。

|  |  |  |
| --- | --- | --- |
| 偏移 | 循环次数（平均值）\* | 描述 |
| 0x38 | 62 | int n（int v，int n） |
| 将无符号定点整数表示形式转换为最接近的双精度值，并舍入为偶数。*n*指定定点中二进制点的位置，因此 |
| 0x3c | 1617 | double\_dcos（双角度） |
| 返回*角度*的余弦。*角度*以弧度为单位，并且必须在-1024到1024范围内 |
| 0x40 | 1618 | double\_dsin（双角度） |
| 返回*角度*的正弦值。*角度*以弧度为单位，并且必须在-1024到1024范围内 |
| 0x44 | 1891 | double\_dtan（双角度） |
| 返回*角度*的切线。*角度*以弧度为单位，并且必须在-1024到1024范围内 |
| 0x48 | N/A | 弃用 |
| 不要使用此功能 |
| 0x4c | 804 | double\_dexp（double v） |
| 返回v的指数值，即所以 |
| 0x50 | 428 | double\_dln（double v） |
| 返回v的自然对数。如果返回*-无限* |
| 0x54 | 39 | int\_dcmp（double a，double b） |
| 比较两个浮点数，返回：   * 0 if a ==b * -1如果ab * 1，如果a>b |
| 0x58 | 2168 | double\_datan2（double y，double x） |
| 使用参数的符号计算y/x的反正切以确定正确的象限 |
| 0x5c | 55 | double\_int642double（int64\_t v） |
| 将有符号的64位整数转换为最接近的双精度值，并舍入为偶数 |
| 0x60 | 56 | double\_dix642double（int64\_t v，int n） |
| 将有符号定点64位整数表示形式转换为最接近的双精度值，并舍入为偶数。*n*指定定点中二进制点的位置，因此 |
| 0x64 | 50 | double\_uint642double（uint64\_t v） |
| 将无符号的64位整数转换为最接近的双精度值，并舍入为偶数 |
| 0x68 | 49 | double\_ufix642double（uint64\_t v，int n） |
| 将无符号定点64位整数表示形式转换为最接近的双精度值，并舍入为偶数。*n*指定定点中二进制点的位置，因此 |
| 0x6c | 64 | \_double2int64 |
| 将双精度转换为有符号64位整数，四舍五入至*-Infinity*，并将结果限制在-0x800000000000000至0x 7 FFFFFFFFFFFF范围 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 循环次数（平均值）\* | 描述 |
| 0x70 | 63 | \_double2fix64 |
| 将double转换为有符号定点64位整数表示，其中*n*指定二进制点在所得定点表示中的位置-例如\_double2fix（0.5f，16）== 0x8000. 此方法向*-Infinity*舍入，并将结果整数限制在-0x8000000000000到0x 7 FFFFFFFFFFFF |
| 0x74 | 53 | \_double2uint64 |
| 将double转换为无符号64位整数，向*-Infinity*舍入，并将结果箝位为0x 0000000000000到0xFFFFFFFFFFFF范围内 |
| 0x78 | 52 | \_double2ufix64 |
| 将double转换为无符号定点64位整数表示，其中*n*指定二进制点在所得定点表示中的位置，例如  \_double2ufix（0.5f，16）== 0x8000. 此方法向*-Infinity*舍入，并将结果整数限制在0x 000000000000到0xFFFFFFFFFFFF范围 |
| 0x7c | 23 | float\_double2float（double v） |
| 将double转换为float |
| 仅在V3 bootrom中提供的功能 | | |
| 0x48  (uses以前不推荐使用的插槽） | 1718  (V3仅限） | double（，double）\_sincos（double angle） |
| 计算*角度*的正弦和余弦。*角度*以弧度为单位，并且必须在-1024至1024范围内正弦值返回寄存器*r 0*/*r1*（因此是正式返回值），余弦值返回寄存器*r2*/*r3*。该方法比单独调用\_sin和\_cos快得多。 |

* + - 1. Bootrom数据

Bootrom数据表（rom\_data\_table）包含以下指针。

*表171.对妇女的歧视 Bootrom数据指针*

|  |  |
| --- | --- |
| 代码 | 值（16位指针）说明 |
| C、R | const char\*copyright\_string |
| Raspberry Pi Trading Ltd版权字符串。 |
| “G”、“R” | const uint32\_t\*git\_revision |
| Bootrom git版本的8个最重要的十六进制数字。 |
| “F”、“S” | fplib\_start |
| 浮点库代码和数据的起始地址. 如果需要，可以使用This和fplib\_end以及soft\_float\_table中的各个函数指针将浮点实现复制到RAM中。 |
| “S”、“F” | 软浮台 |
| 本表内容见[表169](#_bookmark232)。 |
| “F”、“E” | fplib\_end |
| 浮点库代码和数据的结束地址 |

|  |  |
| --- | --- |
| “S”、“D” | 软双层桌 |
| 此条目仅存在于V2 bootrom中。本表内容见[表170](#_bookmark233) |
| 'P'、'8' | *被弃*用V2 bootrom中不存在此条目;请勿使用它。 |
| “R”、“8” | *被弃*用V2 bootrom中不存在此条目;请勿使用它。 |
| “L”、“8” | *被弃*用V2 bootrom中不存在此条目;请勿使用它。 |
| “T”、“8” | *被弃*用V2 bootrom中不存在此条目;请勿使用它。 |

* + 1. USB大容量存储接口

Bootrom提供了一个标准的USB引导加载程序，该引导程序提供了一个可写驱动器，用于使用*UF 2*文件将代码复制到RP 2040（参见[第2.8.4.2节](#_bookmark236)）。

复制到驱动器的*UF2*文件被下载并写入闪存或RAM，设备自动重启，仅使用USB连接即可在RP 2040上下载和运行代码

* + - 1. RPI-RP 2驱动器

RP 2040显示为一个名为*RPI-RP 2*的标准128 MB闪存驱动器，格式为FAT 16的单个分区在指定的驱动器上只有两个实际文件可见

* INFO\_UF2.TXT- 包含UF 2引导加载程序和版本的字符串描述
* INDEX.HTM- 重定向到有关RP 2040设备的信息

任何类型的文件都可以从主机写入USB驱动器，但一般来说，这些文件不会存储，只会*出现*

因为在主机端有缓存。

但是，当*UF2*文件写入设备时，会识别特殊内容，并将数据写入RAM或闪存中的指定位置在完成整个有效*UF2*文件的下载后，RP 2040自动重启以运行新下载的代码。

**注意**

INDEX.HTM文件当前重定向到<https://www.raspberrypi.com/documentation/microcontrollers/>

* + - 1. UF2格式详细信息

*UF2*文件有下载到RP 2040的有效要求请务必始终使用有效的*UF2*文件（例如由<https://github.com/raspberrypi/pico-sdk/blob/master/tools/elf2uf2/main.cpp>生成的文件），因为无效文件可能会部分写入，然后默默失败。请注意，在某些操作系统上，您可能会在失败时收到磁盘写入错误，但不能保证这一点

* 所有发往设备的数据都必须位于UF 2块中，其中存在familyID并设置为0xe48bff56，并且payload\_size

在256。

* 所有数据必须指定（并完全适合）以下内存范围（取决于下载的二进制类型，由遇到的第一个*UF2*块的地址确定
  1. 一个普通的闪光二进制
     + 0x 10000000-0x 11000000*闪存：*所有块必须以256字节对齐为目标。超出物理闪存末尾的写入
  2. 一个*RAM只有*二进制
     + 0x 20000000-0x 20042000*主RAM：* 块可以通过字节对齐进行
     + 0x 15000000-0x 15004000*闪存缓存：*（由于闪存不是目标，闪存缓存可用作RAM，其属性与*主RAM*相同）。

##### 注意

传统上，*UF2*仅用于写入闪存，但这更多地是使用无元数据的

.BIN文件作为源来生成*UF2*文件。RP2040充分利用了*UF2*固有的灵活性，以支持由构建生成的更丰富的.ELF格式的全部二进制文件，以用作*UF2*文件的源

* numBlocks必须指定适合上述指定区域的二进制文件的总大小
* numBlocks或二进制类型（由*UF2*块目标地址确定）的更改将丢弃当前正在进行的传输
* 所有数据必须在没有UF2\_FLAG\_NOT\_MAIN\_FLASH标记的块中，该标记与要忽略的内容有关，而不是与闪存和RAM有关。

闪存总是一次擦除4kB扇区，因此在闪存二进制UF 2中仅包括扇区内256字节页的子集的数据将使扇区的其余256字节页被擦除但未定义。RP 2040 bootrom将接受具有这种部分填充扇区的UF 2二进制文件，但是由于错误（RP 2040-E14），如果有任何部分填充扇区而不是在末尾，则可能无法正确写入这种二进制文件。大多数闪存二进制文件都是4kB对齐且连续的，因此通常只有最后一个扇区被部分填充。如果您需要将非对齐或非连续的UF2写入闪存，则应确保为闪存中将写入的每个扇区（最后一个扇区除外）包含完整的4kB数据这是由SDK 1.3.1版及以后版本中的elf2uf2工具自动处理的，它会显式地将零填充页面添加到相应的*部分填充*扇区。

当每个numBlocks块在单个有效传输过程中至少被看到一次时，二进制文件被认为是“下载”的。在主机重新发送重复数据块的情况下，仅在第一次写入数据块的数据

在下载常规闪存二进制文件之后，执行复位，之后闪存二进制文件第二阶段（在地址

0x 10000000- 闪存的开始）将通过bootrom输入（如果有效）

通过看门狗复位将下载的*仅RAM*二进制文件输入到二进制文件的开头，计算为下载块的最低地址（如果主RAM和闪存都存在，则认为主RAM低于闪存

最后，主机软件可以通过PICOBOOT接口暂时禁用*UF2*写入，以防止干扰通过该接口执行的操作（见下文），在这种情况下*，*任何正在进行的UF2文件写入都将被中止。

* + 1. USB PICOBOOT接口

PICOBOOT接口是一个低级USB协议，用于在RP 2040处于BOOTSEL模式时与RP 2040进行交互此接口可与USB大容量存储接口同时使用。

它提供了灵活的RAM或闪存读写、重启、在设备上执行代码以及其他一些管理功能。

与接口相关的常量和结构可以在SDK标头<https://github.com/raspberrypi/pico-sdk/blob/master/src/common/boot_picoboot/include/boot/picoboot.h>中找到

* + - 1. 识别所述装置

RP 2040设备通过其设备描述符中的*供应商ID*和*产品ID*进行识别（如[表172](#_bookmark238)所示）。

表172.对妇女的歧视RP2040

引导设备描述符

表173.对妇女的歧视PICOBOOT

接口描述符

|  |  |
| --- | --- |
| 领域 | 值 |
| b长度 | 18 |
| b描述符类型 | 1 |
| bcdUSB | 1.10 |
| bDeviceClass | 0 |
| bDeviceSubClass | 0 |
| b设备协议 | 0 |
| bMaxPacketSize0 | 64 |
| idVendor | 0x2e8a |
| id产品 | 0x0003 |
| bcdDevice | 1.00 |
| iManufacturer | 1 |
| iProduct | 2 |
| iSerial | 3 |
| b数字 | 1 |

* + - 1. 识别接口

PICOBOOT接口由“供应商特定”*接口类*和零*接口子类*和*接口协议*识别（如[表173](#_bookmark239)所示）。请注意，您不应依赖于接口编号，因为这取决于设备是否也暴露大容量存储接口。还请注意，设备同样可能根本没有暴露PICOBOOT接口，因此您不应假设它存在。

|  |  |
| --- | --- |
| 领域 | 值 |
| b长度 | 9 |
| b描述符类型 | 4 |
| b接口编号 | 变化 |
| bAlternateSettings | 0 |
| bNum端点 | 2 |
| bInterfaceClass | 0xff（特定于供应商） |
| b接口子类 | 0 |
| b接口协议 | 0 |
| iInterface | 0 |

* + - 1. 确定终点

PICOBOOT接口提供单个*BULK OUT*和单个*BULK IN*端点。这些可以通过它们的方向和类型来识别。您不应该依赖端点编号。

* + - 1. PICOBOOT命令

表174.对妇女的歧视PICOBOOT

命令定义

表175.对妇女的歧视PICOBOOT

独占存取命令结构

表176.对妇女的歧视PICOBOOT

重新引导访问命令结构

这两个批量终结点用于发送命令和检索成功的命令结果。所有命令均为32字节（见[表174](#_bookmark240)），并发送到*BULK OUT*端点。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 描述 |
| 0x00 | dMagic | 值0x431fd10b |
| 0x04 | dToken | 用户提供的用于标识此请求的令牌， |
| 0x08 | bCmdId | 命令的ID。注意，最高位指示数据传输方向（0x80 = IN） |
| 0x09 | bCmdSize | *args*字段中有效数据的字节数 |
| 0x0A | 保留 | 0x0000 |
| 0x0C | dTransferLength | 主机期望通过批量通道发送或接收的字节数 |
| 0x10 | args | 16个字节的命令特定数据，用零填充 |

如果发送的命令无效或无法识别，则批量终结点将停止。更多信息将通过*GET\_COMMAND\_STATUS*请求提供（参见[第2.8.5.5.2节](#_bookmark241)）。

在初始的32字节数据包之后，如果*dTransferLength*非零，则通过批量管道传输该数量的字节如果*dTransferLength*为零，则命令成功由空IN分组指示。

支持以下命令（注意，为了清楚起见，省略了公共字段*dMagic*、*dToken*、*reserved*

* + - * 1. EXCLUSIVE\_ACCESS（0x01）

声明或释放通过USB写入RP 2040的独占访问权限（相对于大容量存储接口）

|  |  |  |  |
| --- | --- | --- | --- |
| 偏移 | 名称 | 数值/描述 | |
| 0x08 | bCmdId | 0x01（EXCLUSIVE\_ACCESS） | |
| 0x09 | bCmdSize | 0x01 | |
| 0x0C | dTransferLength | 0x00000000 | |
| 0x10 | b专属 | 非排他性（0） | 对USB大容量存储操作没有限制 |
| 独家（1） | 禁用USB大容量存储写入（主机应将其视为写保护失败，但在任何情况下，任何活动*的UF2*下载都将中止） |
| EXCLUSIVE\_AND\_EQUIPMENT（2） | 通过将驱动器介质标记为不存在（弹出驱动器） |

* + - * 1. 重新启动（0x02）

从BOOTSEL模式重新启动RP 2040请注意，如果重新引导到闪存，并且没有找到有效的第二阶段引导加载程序，则可能会重新进入BOOTSEL模式

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 数值/描述 |
| 0x08 | bCmdId | 0x02（重新启动） |
| 0x09 | bCmdSize | 0x0C |

|  |  |  |  |
| --- | --- | --- | --- |
| 0x0C | dTransferLength | 0x00000000 | |
| 0x10 | DPC | 开始执行的地址。有效值为： | |
| 0x00000000 | 通过标准闪存启动机制重新启动 |
| RAM地址 | 通过看门狗重新引导并在RAM中的指定地址开始执行 |
| 0x14 | DSP | 重新启动后的初始堆栈指针（仅在引导到RAM时使用 | |
| 0x18 | dDelayMS | 重新启动前延迟的毫秒数 | |

* + - * 1. FLASH\_ERASE（0x03）

表177.对妇女的歧视PICOBOOT

闪存擦除命令结构

表178.对妇女的歧视PICOBOOT

读取存储器命令（闪存、RAM、ROM）结构

表179.对妇女的歧视PICOBOOT

写存储器命令（闪存、RAM）结构

擦除连续范围的闪存扇区。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 数值/描述 |
| 0x08 | bCmdId | 0x03（FLASH\_ERASE） |
| 0x09 | bCmdSize | 0x08 |
| 0x0C | dTransferLength | 0x00000000 |
| 0x10 | 地址 | 闪存中要擦除的地址，从该位置开始这必须与扇区（4kB）对齐 |
| 0x14 | dSize | 要擦除的字节数这必须是扇区的确切倍数（4kB） |

* + - * 1. 读取（0x84）

从RP 2040读取连续内存（闪存或RAM或ROM）范围

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 数值/描述 |
| 0x08 | bCmdId | 0x84（读取） |
| 0x09 | bCmdSize | 0x08 |
| 0x0C | dTransferLength | 必须与dSize |
| 0x10 | 地址 | 要从中读取的地址可能在闪存或RAM或ROM中 |
| 0x14 | dSize | 要读取的字节数 |

* + - * 1. 写入（0x05）

在RP 2040上写入连续的内存范围（闪存或RAM）。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 数值/描述 |
| 0x08 | bCmdId | 0x05（写入） |
| 0x09 | bCmdSize | 0x08 |
| 0x0C | dTransferLength | 必须与dSize |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 数值/描述 |
| 0x10 | 地址 | 写入的地址。可以在闪存或RAM中，但如果在闪存中，则必须页（256字节）对齐注意，必须先擦除闪存，否则结果不确定。 |
| 0x14 | dSize | 要写入的字节数。如果写入闪存且大小不是页面的精确倍数（256字节），则最后一页将被零填充到末尾。 |

* + - * 1. EXIT\_XIP（0x06）

表180.第180页PICOBOOT

退出就地执行（XIP）命令结构

表181.对妇女的歧视PICOBOOT

输入就地执行（XIP）命令

表182.对妇女的歧视PICOBOOT

设备命令结构上的执行功能

退出Flash XIP模式。这首先初始化串行传输的SSI，然后发出[第2.8.1.2](#_bookmark227)中给出的XIP退出序列，以尝试使闪存响应标准串行SPI命令。SSI配置有固定时钟除数/6，因此USB引导加载程序将以8 MHz驱动SCLK

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 数值/描述 |
| 0x08 | bCmdId | 0x06（EXIT\_XIP） |
| 0x09 | bCmdSize | 0x00 |
| 0x0C | dTransferLength | 0x00000000 |

* + - * 1. ENTER\_XIP（0x07）

进入Flash XIP模式。这将SSI配置为针对每次XIP访问发出标准03h串行读取命令，具有24个地址时钟这是一种缓慢但得到广泛支持的读取闪存的方法。此函数的目的是使闪存易于访问（即仅访问0x10......段中的地址），而不必确切知道连接的是哪种闪存的详细信息这种模式适合于从flash执行代码，但

慢于，例如 QSPI XIP访问。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 数值/描述 |
| 0x08 | bCmdId | 0x07（ENTER\_XIP） |
| 0x09 | bCmdSize | 0x00 |
| 0x0C | dTransferLength | 0x00000000 |

* + - * 1. 执行（0x08）

在设备上执行一个功能这个函数不接受参数，也不返回结果，所以它必须通过RAM进行通信此方法的执行将阻止任何其他命令以及大容量存储接口*UF2*写入，因此应仅在独占模式下使用，并且应非常小心（并且应根据ARM EABI保存和恢复寄存器）。这个方法是从常规（非IRQ）上下文调用的，并且有一个非常有限的堆栈，所以函数应该使用自己的堆栈。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 数值/描述 |
| 0x08 | bCmdId | 0x08（EXEC） |
| 0x09 | bCmdSize | 0x04 |
| 0x0C | dTransferLength | 0x00000000 |
| 0x10 | 地址 | 函数执行地址（由于您忘记了，因此将为您添加一个拇指位 |

* + - * 1. VECTORIZE\_FLASH（0x09）

表183.对妇女的歧视PICOBOOT

矢量化闪存命令结构

请求将Mass Storage和PICOBOOT接口内部使用的闪存访问函数的矢量表复制到RAM中，以便可以用自定义版本替换方法实现（例如，如果板使用不支持标准命令的闪存）

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 数值/描述 |
| 0x08 | bCmdId | 0x09（VECTORIZE\_FLASH） |
| 0x09 | bCmdSize | 0x04 |
| 0x0C | dTransferLength | 0x00000000 |
| 0x10 | 地址 | 指向RAM中向量表放置位置的指针 |

Flash函数向量表

构造函数

uint32\_t size;//28

uint32\_t（\*do\_flash\_enter\_cmd\_xip）（）;uint32\_t（\*do\_flash\_exit\_xip）（）; uint32\_t（\*do\_flash\_erase\_sector）（）;

uint32\_t（\*do\_flash\_erase\_range）（uint32\_t addr，uint32\_t size）; uint32\_t（\*do\_flash\_page\_program）（uint32\_t addr，uint8\_t \*data）; uint32\_t（\*do\_flash\_page\_read）（uint32\_t addr，uint8\_t \*data）;

};

这些方法与bootrom中相应的闪存访问函数具有相同的签名和参数（参见[第2.8.3.1.3节](#_bookmark231)）。

请注意，主机随后必须通过在RP 2040上运行的EXEC命令更新此表的RAM副本，因为从主机通过PICOBOOTWRITE对RAM的任何写入（与此（现在在RAM中处于活动状态）矢量表重叠）将导致重置为使用默认ROM闪存功能矢量表。

* + - 1. 控制请求

以下请求通过默认控制管道发送到接口

* + - * 1. 接口\_重置（0x 41）

表184.对妇女的歧视PICOBOOT

重置PICOBOOT接口控制

主机发送此控制请求以重置PICOBOOT接口。此命令：

* 清除每个批量终结点上的HALT条件（如果已设置
* 中止任何进程中的PICOBOOT或大容量存储传输以及任何闪存写入（此方法是终止卡住的闪存传输的唯一方法
* 清除上一个命令结果
* 如果大容量存储驱动器由于独占性而被弹出，则重新安装EXCLUSIVE\_ACCESS。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| bmRequestType | b请求 | wValue | Windex | wLength | 数据 |
| 01000001b | 01000001b | 0000H | 接口 | 0000H | 没有一 |

此命令在成功时以空数据包响应。

* + - * 1. GET\_COMMAND\_STATUS（0x42）

表185.对妇女的歧视PICOBOOT

获取上次命令状态控制

表186.PICOBOOT

获取上次命令状态控制响应

检查最后一个命令的状态（可能是仍在执行中的命令）。PICOBOOT协议命令的成功完成是通过批量管道确认的，但是如果操作仍在进行中或已经失败（停止批量管道），则此方法可用于确定操作的状态。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| bmRequestType | b请求 | wValue | Windex | wLength | 数据 |
| 11000001 b | 01000010 b | 0000H | 接口 | 0000H | 没有一 |

该命令以以下16字节响应进行响应

|  |  |  |  |
| --- | --- | --- | --- |
| 偏移 | 名称 | 描述 | |
| 0x00 | dToken | 使用命令指定的用户令牌 | |
| 0x04 | dStatusCode | 无（0） | 命令已成功完成（或仍在进行中） |
| Unknown\_CMD（1） | 无法识别命令的ID |
| INVALID\_CMD\_LENGTH（2） | 命令请求的长度不正确 |
| INVALID\_TRANSFER\_LENGTH（3） | 数据传输长度不正确，给出命令 |
| INVALID\_ADDRESS（4） | 指定的地址对于命令类型无效  即与命令预期的类型Flash/RAM不 |
| （5）第一章 | 指定的地址未根据命令的要求正确对齐 |
| Interleaved\_WRITE（6） | 大容量存储接口*UF2*写入操作干扰了当前操作。该命令被放弃，状态不明。请注意，如果您拥有*独占*访问权限，则不会发生这种情况。 |
| 重新启动（7） | 设备正在重新启动，因此忽略了该 |
| Unknown\_ERROR（8） | 发生其他错误。 |
| 0x08 | bCmdId | 命令的ID | |
| 0x09 | bInProgress | 如果命令仍在进行中，则为1 | 否则为0 |
| 0x0A | 保留 | （6个零字节） | |

* 1. 电源

RP2040需要五个独立的电源。然而，在大多数应用中，其中的几个可以组合并连接到单个电源。在典型应用中，仅需要3.3V单电源。参见[2.9.7.1部分“单3.3V电源”](#_bookmark250)。

以下章节描述了电源和多种潜在的电源方案详细的电源参数见第5.6节“电源”。

* + 1. 数字IO电源（IOVDD）

IOVDD为芯片的数字IO供电，应在1.8V和3.3V之间的标称电压下供电。电源电压设置数字IO的外部信号电平，应根据所需的信号电平进行选择。有关详细信息，请参见第5.5.3节“引脚规格”。所有数字IO共享相同的电源，并在相同的信号电平下工作。

IOVDD应通过靠近每个芯片IOVDD引脚的100 nF电容去耦。

**谨慎**

如果数字IO以标称1.8 V供电，则IO输入阈值应通过VOLTAGE\_SELECT寄存器进行调整默认情况下，当数字IO以以下标称电压供电时，IO输入阈值有效：

2.5V和3.3V。有关详细信息，请参见第2.19节“GPIO”以1.8V为IO供电，输入阈值设置为2.5V，

3.3V电源是安全工作模式，但会导致输入阈值不符合规格。 在输入阈值设置为1.8V电源的情况下，以大于标称1.8V的电压为IO供电可能会导致芯片损坏。

* + 1. 数字核心电源（DVDD）

DVDD提供芯片的核心数字逻辑，并应在标称1.1V供电。提供专用片内电压调节器，允许从数字IO电源（IOVDD）或另一个标称1.8V电源产生 DVDD

3.3V电源片内稳压器的输出引脚（VREG\_VOUT）与DVDD电源引脚之间的连接在片外进行，因此，如果需要，DVDD可以从片外电源供电

DVDD应通过靠近每个芯片DVDD引脚的100 nF电容去耦。

* + 1. 片内稳压器输入电源（VREG\_VIN）

VREG\_VIN是片内电压调节器的输入电源它应该在1.8V和3.3V之间的标称电压供电为了减少外部电源的数量，VREG\_VIN可以使用与数字IO电源（IOVDD）相同的电源

应在VREG\_VIN和靠近芯片VREG\_VIN引脚的地之间连接一个1μF电容。

##### 谨慎

VREG\_VIN还为芯片的上电复位和欠压检测模块供电，因此即使不使用片内电压调节器，也必须对其供电

有关片内电压调节器的更多详细信息，请参见[第2.10节“内核电源调节器”](#_bookmark256)。

* + 1. USB PHY电源（USB\_VDD）

USB\_VDD为芯片的USB PHY供电，并且应该以标称3.3V供电。为了减少外部电源的数量，USB\_VDD可以使用与数字IO电源（IOVDD）相同的电源，假设IOVDD也以3.3V供电。如果IOVDD不是以3.3V供电，则USB PHY将需要单独的3.3V电源，请参见[第2.9.7.3节“带功能USB和ADC的1.8V数字IO](#_bookmark253)。在从不使用USB PHY的应用中，USB\_VDD可以连接到标称电压在1.8V和3.3V之间的任何电源有关示例，请参见[第2.9.7.4节“单1.8V电源”](#_bookmark254)USB\_VDD不应处于未连接状态。

USB\_VDD应通过靠近芯片USB\_VDD引脚的100 nF电容去耦。

* + 1. ADC电源（ADC\_AVDD）

ADC\_AVDD提供芯片的模数转换器（ADC）。它可以在标称电压下供电，

1.8V和3.3V，但ADC的性能将在低于2.97V的电压下受到影响为了减少外部电源的数量，ADC\_AVDD可以使用与数字IO电源（IOVDD）相同的电源

##### 注意

以高于或低于IOVDD的电压为ADC\_AVDD供电是安全的，例如为ADC提供3.3V电源，以实现最佳性能，同时在数字IO上支持1.8V信号电平但ADC模拟输入端的电压不得超过IOVDD，例如，如果IOVDD以1.8V供电，则ADC输入端的电压应限制为1.8V。大于IOVDD的电压将导致通过ESD保护二极管的漏电流。有关详细信息，请参见第5.5.3节“引脚规格”。

ADC\_AVDD应通过靠近芯片ADC\_AVDD引脚的100 nF电容去耦。

* + 1. 电源时序控制

RP 2040的电源可以以任何顺序通电或断电但是，如果ADC电源（ADC\_AVDD）在数字内核电源（DVDD）之前上电或之后关断，则ADC电源（ADC\_AVDD）中可能会流过较小的瞬态电流这不会损坏芯片，但可以通过在ADC\_AVDD之前或同时上电DVDD，并在ADC\_AVDD之后或同时关断DVDD来避免。在最常见的电源方案中，芯片由3.3V单电源供电，由于片内电压调节器的启动时间，DVDD将在ADC\_AVDD之后不久上电这是可以接受的行为。参见[2.9.7.1部分“单3.3V电源”](#_bookmark250)。

* + 1. 供电方案
       1. 单3.3V电源

在大多数应用中，RP 2040将由3.3V单电源供电，如[图16](#_bookmark251)所示。 数字IO（IOVDD）、USB PHY（USB\_VDD）和ADC（ADC\_AVDD）将直接由3.3V电源供电，1.1V数字内核电源（DVDD）将由片内电压调节器从3.3V电源进行调节。请注意，稳压器输出引脚（VREG\_VOUT）必须连接到芯片的片外DVDD引脚。

有关片内电压调节器的更多详细信息，请参见[第2.10节“内核电源调节器”](#_bookmark256)。

*图16. 为芯片供电*



3.3V电源（简化图，省略去耦元件）

* + - 1. 外部核心供应

数字内核（DVDD）可直接由外部1.1 V电源供电，而不是由片内稳压器供电，如[图17](#_bookmark252)所示。如果系统中的其他地方有合适的外部稳压器，或者对于低功耗应用（其中可以使用高效的开关模式稳压器而不是效率较低的线性片上稳压器），这种方法可能是有意义的

如果使用外部内核电源，则片内电压调节器（VREG\_VOUT）的输出应保持未连接状态。然而，电源仍然必须提供给稳压器输入（VREG\_VIN），以提供芯片的上电复位和掉电检测模块。一旦VREG\_VIN可用，片内电压调节器就会上电，但一旦芯片脱离复位，则可以在软件控制有关详细信息，请参见[第2.10节“堆芯电源调节器”](#_bookmark256)

*图17. 使用外部磁芯电源*



* + - 1. 1.8V数字IO，带功能性USB和ADC

数字IO信号电平低于3.3V的应用将需要为USB PHY和ADC提供单独的3.3V电源，因为USB PHY在低于3.135V的电压下不符合规格，而ADC性能在低于2.97V的电压下会受到影响。[图18](#_bookmark255)显示了一个示例应用，其中数字IO（IOVDD）采用1.8 V供电，USB PHY（USB\_VDD）和ADC（ADC\_AVDD）采用单独在本例中，电压调节器输入（VREG\_VIN）连接到1.8V电源，但它也可以连接到3.3V电源。如果1.8V电源由高效的开关模式稳压器产生，则将其连接到1.8V电源将降低整体功耗

*图18. 支持*



使用USB和ADC时为

* + - 1. 单一的1.8V电源

如果不需要功能正常的USB PHY和最佳ADC性能，则RP 2040可以由小于3.3V的单电源供电[图19](#_bookmark257)显示了采用1.8 V单电源的示例。在本例中，内核电源（DVDD）由片内电压调节器从1.8V电源进行

*图19. 为芯片供电*

的1.8V电源

* 1. 核心电源调节器

RP 2040包括一个片内电压调节器，允许从外部（标称1.8V至3.3V）电源产生数字核心电源（DVDD）在大多数情况下，稳压器的输入电源将与芯片的数字IO电源IOVDD共享外部电源，从而简化了整体电源要求。

为了允许芯片启动，电压调节器默认使能，并将在其输入电源可用时通电一旦芯片脱离复位，调节器可以被禁用，置于高阻抗状态，或在软件控制下调整其输出电压。输出电压可以在0.80 V至1.30 V范围内以50 mV步进设置，但在初始上电或复位事件后设置为标称1.1 V。电压调节器可提供高达100 mA的电流。

虽然旨在提供芯片的数字核心电源（DVDD），电压调节器可以用于其他目的，如果DVDD是直接从外部电源供电

* + 1. 应用电路

*图20. 电压调节器应用电路*



稳压器必须在输入（VREG\_VIN）和输出（VREG\_VOUT）引脚附近放置1μF电容。

* + 1. 操作模式

表187.对妇女的歧视电压调节器模式选择

电压调节器以三种模式之一工作 通过写入VREG寄存器的EN和HIZ字段来选择要使用的模式，如[表187所示](#_bookmark260)。在初始上电时，或在复位事件之后，电压调节器将处于*正常操作*模式。

|  |  |  |
| --- | --- | --- |
| 模式 | EN | HIZ |
| 正常操作a | 1 | 0 |
| 高阻抗 | 1 | 1 |
| 关闭 | 0 | X |

a电压调节器将在初始上电或复位事件之后处于正常模式

* + - 1. 正常操作模式

在正常操作模式下，电压调节器的输出在选定电压下处于调节状态，并且调节器能够供电。

* + - 1. 高阻抗模式

在高阻抗模式下，稳压器禁用，其输出引脚（VREG\_VOUT）设置为高阻抗状态。在这种模式下，稳压器的功耗最小。此模式允许连接到VREG\_VOUT的负载从片内稳压器以外的电源供电。例如，这可以允许负载最初从片上电压调节器供电，然后在软件控制下切换到外部调节器。外部稳压器还需要支持高阻抗模式，一次只有一个稳压器为负载供电。当两个稳压器都处于高阻抗模式时，电源电压由稳压器的输出电容维持

* + - 1. 关断模式

在关断模式下，电压调节器被禁用，功耗被最小化，并且调节器的输出引脚（VREG\_VOUT）被拉至0 V。

只有在稳压器不提供RP 2040的数字核心电源（DVDD）时，才使用PWM模式。如果稳压器正在为DVDD供电，并且欠压检测已使能，则进入关断模式将导致复位事件，稳压器将返回正常模式。如果欠压检测未使能，电压调节器将关闭并保持关断模式，直到其输入电源（VREG\_VIN）重新上电。

* + 1. 输出电压选择

通过写入[VREG](#_bookmark266)寄存器中的VSEL字段，可以选择所需的电压调节器的输出电压可以在0.80V至1.30V的范围内以50mV的间隔设置稳压器输出电压设置为1.1V在初始上电或复位事件后有关详细信息，请参见[VREG](#_bookmark266)寄存器说明。

请注意，RP 2040在其数字核心电源（DVDD）电压为1.1V以外的情况下可能无法可靠工作

* + 1. 地位

[VREG](#_bookmark266)寄存器包含单个状态字段ROK，用于指示电压调节器的输出是否得到正确调节。

上电时，ROK保持低电平，直到稳压器启动且输出电压达到ROK置位阈值（ROKTH.ASSERT）。然后，它保持高电平，直到电压下降到低于ROK解除断言阈值（ROKTH.DEASSERT），保持低电平，直到输出电压再次高于断言阈值。 ROKTH.ASSERT标称值为选定输出电压的90%，如果选定输出电压为1.1V，则为0.99V;ROKTH.DEASSERT标称值为选定输出电压的87%，如果选定输出电压为1.1V，则为0.957V。

注意，将输出电压调整到更高的电压将导致ROK变低，直到达到更高电压的断言阈值如果稳压器处于高阻抗模式，ROK也会降低

* + 1. 电流限制

电压调节器包括电流限制，以防止负载电流超过最大额定值。当电流限制激活时，输出电压将不被调节，并将下降到选定值以下。

* + 1. 登记册一览表

表188.对妇女的歧视VREG\_AND\_CHIP\_RES列表

ET寄存器

电压调节器与芯片级复位子系统共享寄存器地址空间。两个子系统的寄存器均在此处列出 仅[VREG](#_bookmark266)寄存器是电压寄存器子系统的一部分。 [BOD](#_bookmark267)和[CHIP\_RESET](#_bookmark268)寄存器是芯片级复位子系统的一部分。共享地址空间在本文档的其他地方称为vreg\_and\_chip\_reset。

VREG\_AND\_CHIP\_CHIP寄存器从基址0x40064000（在SDK中定义为[VREG\_AND\_CHIP\_CHIP\_BASE](#_bookmark39)）开始。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x0 | [VREG](#_bookmark266) | 电压调节器控制和状态 |
| 0x4 | [BOD](#_bookmark267) | 欠压检测控制 |
| 0x8 | [芯片\_芯片](#_bookmark268) | 芯片复位控制和状态 |

表189.对妇女的歧视VREG寄存器

表190.对妇女的歧视BOD登记

### [VREG\_AND\_CHIP\_DATA](#_bookmark265)：VREG寄存器

**偏移**：0x0

描述

电压调节器控制和状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:十三分 | Reserved. | - | - | - |
| 12 | 韩国 | 监管现状  0=不符合规定，1=符合规定 | RO | 0x0 |
| 11:8 | Reserved. | - | - | - |
| 7:四 | VSEL | 输出电压选择0000至0101-0.80V  0110-0.85V  0111-0.90V  1000-0.95V  1001-1.00V  1010-1.05V  1011-1.10V（默认）1100 - 1.15V  1101-1.20V  1110-1.25V  1111-1.30V | RW | 0xb |
| 三比二 | Reserved. | - | - | - |
| 1 | HIZ | 高阻抗模式选择器  0=未处于高阻抗模式，1=处于高阻抗模式 | RW | 0x0 |
| 0 | EN | 使  0=未启用，1=已启用 | RW | 0x1 |

### [VREG\_AND\_CHIP\_DATA](#_bookmark265)：BOD寄存器

**偏移**：0x4

描述

欠压检测控制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31：8 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 7:四 | VSEL | 阈值选择0000 - 0.473V  0001-0.516V  0010-0.559V  0011-0.602V  0100-0.645V  0101-0.688V  0110-0.731V  0111-0.774V  1000-0.817V  1001-0.860V（默认）1010 - 0.903V  1011-0.946V  1100-0.989V  1101-1.032V  1110-1.075V  1111-1.118V | RW | 0x9 |
| 三比一 | Reserved. | - | - | - |
| 0 | EN | 使  0=未启用，1=已启用 | RW | 0x1 |

表191.对妇女的歧视CHIP\_CHIP注册

### [VREG\_AND\_CHIP\_DATA](#_bookmark265)：CHIP\_DATA寄存器

**偏移**：0x8

描述

芯片复位控制和状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:二十5分 | Reserved. | - | - | - |
| 24 | PSM\_RESTART\_FLAG | 这由调试器中的psm\_restart设置  它的目的是在调试器发出psm\_restart后将引导代码分支到安全模式，以便从引导锁定中恢复。  在安全模式下，调试器可以修复引导代码，清除此标志，然后重新启动处理器。 | WC | 0x0 |
| 23:21 | Reserved. | - | - | - |
| 20 | HAD\_PSM\_RESTART | 上次重置是从调试端口进行的 | RO | 0x0 |
| 19:17 | Reserved. | - | - | - |
| 16 | HAD\_RUN | 上次复位来自RUN引脚 | RO | 0x0 |
| 15:九分 | Reserved. | - | - | - |
| 8 | HAD\_POR | 上次复位来自上电复位或欠压检测块 | RO | 0x0 |
| 七比零 | Reserved. | - | - | - |

表192.对妇女的歧视电压调节器详细规格

### 详细规格

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 描述 | Min | Typ | Max | 单位 |
| VVREG\_VIN | 输入电源电压 | 1.63 | 1.8-3.3 | 3.63 | V |
| ΔVVREG\_VOUT | 输出电压变化 | -三个 |  | +3级 | 所选输出电压的百分比 |
| IMAX | 输出电流 |  |  | 100 | mA |
| I限制 | 电流限制 | 150 | 350 | 450 | mA |
| ROKTH.ASSERT | ROK断言阈值 | 87 | 90 | 93 | 所选输出电压的百分比 |
| ROKTH.DEASSERT | ROK无效阈值 | 84 | 87 | 90 | 所选输出电压的百分比 |
| tPOWER-ONa | 上电时间 |  | 275 | 350 | 微秒 |

a值将随VREG\_VOUT上的负载电流和电容而变化条件：EN = 1，负载电流= 0 mA，VREG\_VIN在100μs内斜升

* 1. 功率控制

RP 2040提供了一系列降低动态功耗的选项

* 单个外设和功能块的顶级时钟门控
* 基于处理器休眠状态的顶层时钟门自动控制
* 系统时钟频率或系统时钟源的动态更改（例如，切换到内部环形振荡器，并禁用PLL和晶体振荡器）
* 零动态功耗休眠状态，在GPIO事件或RTCIRQ时唤醒

RP2040上的所有数字逻辑都在单核电源域中。以下选项可用于降低静态功耗：

* 将内存置于状态保留断电状态
* 支持此功能的外围设备上的电源门控，例如ADC，温度传感器
  + 1. 顶级时钟门

每个时钟域（例如，系统时钟）可以驱动大量不同的硬件块，而不是所有这些硬件块都是一次需要的。为了避免不必要的功耗，每个时钟的每个端点（例如，UART系统时钟输入）可以随时禁用

启用和禁用时钟门是无毛刺的。如果外设时钟被暂时禁用，随后重新使能，外设将处于与禁用时钟之前相同的状态不需要重置或重新初始化

时钟门由两组寄存器控制：WAKE\_ENx寄存器（从[WAKE\_EN0](#_bookmark362)开始）和SLEEP\_ENx寄存器（从SLEEP\_EN0开始）。这两组寄存器在位级别上是相同的，每组寄存器都拥有一个标志来控制每个时钟端点。WAKE\_EN寄存器指定系统唤醒时使能的时钟，SLEEP\_ENx寄存器选择处理器处于SLEEP状态时使能的时钟（[第2.11.2](#_bookmark272)）。

两个Cortex-M0+处理器没有外部可控制的时钟门。相反，处理器根据WFI/WFE指令以及外部事件和IRQ信号的执行，自主门控其子系统的时钟

* + 1. 睡眠状态

当以下所有条件均为真时，RP 2040进入睡眠状态

* 两个处理器都处于休眠状态（例如，在WFE或WFI说明中）
* 系统DMA在任何通道上都没有未完成的传输

当任一处理器被中断唤醒时，RP2040退出SLEEP状态。

处于SLEEP状态时，顶级时钟门由SLEEP\_ENx寄存器（从SLEEP\_EN0开始）而不是WAKE\_ENx寄存器屏蔽。这允许在处理器休眠时更积极地修剪时钟树。

##### 注意

虽然可以在睡眠期间启用时钟，在睡眠之外禁用时钟，但这通常没有用

例如，如果系统一直处于休眠状态，直到一个字符中断，则整个系统（除了该字符中断）都可以被时钟门控（SLEEP\_ENx =全零，除了CLK\_SYS\_UART0和CLK\_PERI\_UART0）。这包括系统基础设施，如总线结构。

当CPU断言其中断并唤醒处理器时，RP 2040离开SLEEP模式，并切换回WAKE\_ENx时钟掩码。这至少应该包括总线结构，以及包含处理器堆栈和中断向量的存储器设备。

系统级时钟请求握手使处理器离开总线，直到时钟重新启用。

* + 1. 休眠状态

休眠状态是真正的零动态功耗休眠状态，其中所有时钟（和所有振荡器）都被禁用。系统可以在发生GPIO事件（高/低电平或上升/下降沿）或RTC中断时从DORMANT状态唤醒：这会重新启动其中一个振荡器（环形振荡器或晶体振荡器），并在振荡器输出稳定后取消门控系统状态被保留，因此代码执行在离开DORMANT状态时立即恢复

请注意，如果依赖RTC（第4.8）从休眠状态唤醒，RTC必须有一些外部时钟源。RTC接受低至1Hz的时钟频率。

还请注意，DORMANT不会停止PLL。为避免不必要的功耗，软件应在进入休眠状态前关闭PLL电源，并在退出后再次上电和重新配置PLL

通过向激活的振荡器（环形振荡器（第2.17）或晶体振荡器（第2.16））中的DORMANT寄存器写入关键字，进入DORMANT状态。如果两者都是活动的，那么提供处理器时钟的一个

* + 1. 内存断电

主系统存储器（SRAM 0.5，映射到总线地址0x20000000至0x20041fff）以及USB DPRAM可通过Syscfg寄存器中的MEMPOWERDOWN寄存器关断（参见第2.21节）。当断电时，存储器保留其当前内容，但不能被访问。静态功率降低。

##### 谨慎

断电时不得访问内存这样做可能会损坏内存内容。

当存储器备份时，在再次访问存储器之前需要20 ns的延迟。

XIP缓存（参见第2.6.3节）也可以通过CTRL. POWER\_DOWN关闭。当缓存断电时，XIP硬件将不会生成缓存访问请注意，如果代码继续从XIP执行，由于外部QSPI总线的电压和开关电容相对较高，这不太可能产生净功耗节省

* + 1. 程序员模型
       1. 睡眠

hello\_sleep示例<https://github.com/raspberrypi/pico-playground/blob/master/sleep/hello_sleep/hello_sleep.c>演示了睡眠模式。hello\_sleep应用程序（和底层函数）执行以下步骤：

* 从XOSC运行系统中的所有时钟
* 在RTC中配置未来10秒的报警
* 使用SLEEP\_ENx寄存器，将CLK\_rtc设置为在睡眠模式下运行的唯一时钟（参见SLEEP\_EN0）
* 在处理器中启用深度睡眠
* 在处理器上调用wfi，这将使处理器进入深度睡眠，直到被RTC中断唤醒
* RTC中断清除报警，然后调用用户提供的回调函数
* 回调函数结束示例应用程序

##### 注意

必须在proc0和proc1上启用深度睡眠，并调用wfi，同时确保DMA停止以进入睡眠模式。

hello\_sleep使用[Pico Extras](https://github.com/raspberrypi/pico-extras)的pico\_sleep中的函数。特别是，sleep\_goto\_sleep\_until使处理器进入睡眠状态，直到被假定为未来的RTC时间唤醒

Pico Extras：[https://github.com/raspberrypi/pico-extras/blob/master/src/rp2\_common/pico\_sleep/sleep.c](https://github.com/raspberrypi/pico-extras/blob/master/src/rp2_common/pico_sleep/sleep.c#L106-L122)Lines 106-122

106void sleep\_goto\_sleep\_until（datetime\_t \*t，rtc\_callback\_t callback）{

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

//我们应该已经调用了sleep\_run\_from\_dormant\_source函数

assert（dermal\_source\_valid（\_dermal\_source））;

*//在睡眠模式下*关闭*所有时钟，除了RTC*clocks\_hw->sleep\_en0 = CLOCKS\_SLEEP\_EN0\_CLK\_RTC\_BITS; clocks\_hw->sleep\_en1 =0x 0;

return（t）;

uint save = scb\_hw->scr;

//在进程中启用深度睡眠

scb\_hw->scr=保存|M0PLUS\_SCR\_SLEEPDEEP\_BITS;

//继续睡觉

getString（）;

122}

* + - 1. 休眠

hello\_dormant示例<https://github.com/raspberrypi/pico-playground/blob/master/sleep/hello_dormant/hello_dormant.c>演示了休眠模式。该示例执行以下步骤：

* 从XOSC运行系统中的所有时钟
* 为“休眠\_唤醒”硬件配置GPIO中断，可将ROSC和XOSC从休眠模式唤醒
* 将XOSC置于休眠模式，立即停止所有处理器执行（以及芯片上的所有其他时钟逻辑
* 当GPIO 10变为高电平时，XOSC再次启动，程序继续执行hello\_dormant在后台使用sleep\_goto\_dormant\_until\_pin

*PicoExtras：[https://github.com/raspberrypi/pico-extras/blob/master/src/rp2\_common/pico\_sleep/sleep.c](https://github.com/raspberrypi/pico-extras/blob/master/src/rp2_common/pico_sleep/sleep.c#L134-L155)Lines134-155*

134void sleep\_goto\_dormant\_until\_pin（uint gpio\_pin，bool edge，bool high）{

|  |  |  |
| --- | --- | --- |
| 135 |  | boollow =！较高; |
| 136 |  | boollevel =！边缘; |
| 137 |  |  |
| 138 |  | //在IO bank0配置适当的IRQ |
| 139 |  | assert（gpio\_pinNUM\_BANK0\_GPIOS）; |
| 140 |  |  |
| 141 |  | return0; |
| 142 |  |  |
| 143 |  | 如果（低电平）事件=IO\_BANK0\_DORMANT\_WAKE\_INTE0\_GPIO0\_LEVEL\_LOW\_BITS; |
| 144 |  | 如果（高电平）事件=IO\_BANK0\_DORMANT\_WAKE\_INTE0\_GPIO0\_LEVEL\_HIGH\_BITS; |
| 145 |  | if（edge high）event =IO\_BANK0\_DORMANT\_WAKE\_INTE0\_GPIO0\_EDGE\_HIGH\_BITS; |
| 146 |  | 如果（边缘低）事件=IO\_BANK0\_DORMANT\_WAKE\_INTE0\_GPIO0\_EDGE\_LOW\_BITS; |
| 147 |  |  |
| 148 |  | gpio\_set\_dormant\_irq\_enabled（gpio\_pin，event，true）; |
| 149 |  |  |
| 150 |  | mysql（）; |
| 151 |  | //执行在这里停止，直到被唤醒 |
| 152 |  |  |
| 153 |  | //清除irq，这样我们就可以再次回到休眠模式，如果我们想要的话 |
| 154 |  | gpio\_acknowledge\_irq（gpio\_pin，event）; |
| 155 | } |  |

* 1. 芯片级复位
     1. 概述

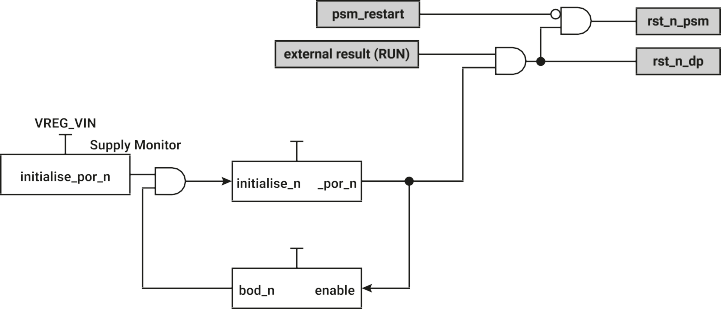
芯片级复位子系统复位整个芯片，将其置于默认状态。这种情况发生在初始上电、电源欠压事件期间或芯片的RUN引脚变为低电平时。该芯片也可以通过Rescue USB端口复位。有关详细信息，请参见[2.3.4.2“Rescue DP”部分](#_bookmark114)

该子系统有两个复位输出。rst\_n\_psm，用于重置除调试端口之外的整个芯片;rst\_n\_dp，用于仅重置救援DP。在初始上电时、欠压事件期间或RUN为低电平时，两种复位均保持低电平rst\_n\_psm还可由救援DP通过子系统的psm\_restart输入保持低电平这允许芯片通过救援DP复位，而无需复位救援DP本身。子系统通过以下方式释放芯片级复位：

rst\_n\_psm为高电平，将控制权交给上电状态机，上电状态机继续启动芯片。有关详细信息，请参见[第2.13“加电状态机”](#_bookmark291)。

芯片级复位子系统如[图21](#_bookmark279)所示，更多信息请参阅以下部分。

*图21. 芯片级复位子系统*



* + 1. 上电复位

上电复位模块通过保持复位状态，确保芯片在第一次上电时完全启动，直到数字内核电源（DVDD）能够可靠地为芯片的内核逻辑供电。该模块保持por\_n输出为低电平，直到DVDD高于*上电复位阈值*（DVDDTH.POR）的时间大于*上电复位断言延迟*（tPOR.ASSERT）。一旦为高，即使DVDD随后下降到低于DVDDTH.POR，por\_n也保持为高，除非使能欠压检测[图22](#_bookmark280)中示出了当施加功率时por\_n的行为。

DVDDTH.POR固定在标称0.957 V，这应导致阈值在0.924 V和0.99 V之间。阈值假设初始上电时的标称DVDD为1.1V，如果使用较低的电压，则por\_n一旦芯片脱离复位状态，只要失压检测被禁用或设置了合适的阈值电压，DVDD就可以降低，而por\_n不会

*图22. 上电复位周期*

DVDDTH.POR

DVDD

tPOR.ASSERT

端口号

* + - 1. 详细规格

表193.对妇女的歧视上电复位参数

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 描述 | Min | Typ | Max | 单位 |
| DVDDTH.POR | 上电复位阈值 | 0.924 | 0.957 | 0.99 | V |
| tPOR.ASSERT | 上电复位断言延迟 |  | 3 | 10 | 微秒 |

* + 1. 欠压检测

欠压检测模块可在数字内核电源（DVDD）降至安全工作电平以下时启动上电复位周期，从而防止不可靠的工作如果DVDD低于欠*压检测阈值*（DVDDTH.BOD）的时间长于欠*压检测断言延迟*（tBOD.ASSERT），则模块的bod\_n这重新初始化上电复位块，复位芯片，通过采取其por\_n输出低，并保持它在复位，直到DVDD返回到一个安全的工作水平。[图23](#_bookmark282)显示了欠压事件和随后的上电复位周期。

*图23. 欠压检测周期*

DVDDTH.POR

DVDD

DVDDTH.BOD

tPOR.ASSERT

端口号

tBOD.ASSERT

身体素质

* + - 1. 检测启用

在初始上电时或在欠压启动复位后自动启用欠压检测。然而，在por\_n变高和检测变为有效之间存在短延迟*，即欠压检测激活延迟*（t BOD.ACTIVE）。这如[图24](#_bookmark283)所示。

*图24. 在初始通电时和欠压事件之后激活欠压检测。*

DVDDTH.POR

DVDDTH.POR

DVDDTH.BOD

DVDD

tPOR.ASSERT

tPOR.ASSERT

端口号

tBOD.ASSERT

身体素质

tBOD活性

tBOD活性

检测无效

检测活动

检测无效

检测活动

一旦芯片脱离复位，检测可以在软件控制下被禁用。这也节省了少量的电力。如果检测随后重新启用，则在再次激活之前，将存在另一个短延迟，即*欠压检测启用延迟*（t BOD.ENABLE）。这在[图25](#_bookmark284)中示出。

通过向[BOD](#_bookmark267)寄存器的EN字段写入0来禁用检测，并通过向同一字段写入1来重新启用检测检测禁用时，模块的bod\_n输出为高电平

*图25. 禁用和启用掉电检测*

EN

1

0

1

tBOD.ENABLE

检测无效

检测无效

检测活动

如果[BOD](#_bookmark267)寄存器复位，则重新启用检测，因为这会将寄存器的EN字段设置为1。同样，检测将在等于*欠压检测使能延迟*（tBOD.ENABLE）的延迟后变为活动状态

**注意**

如果[BOD](#_bookmark267)寄存器通过上电或欠压启动复位而复位，则寄存器复位与欠压检测变为有效之间的延迟将等于欠*压检测激活延迟*（tBOD.ACTIVE）。延迟将等于所有其他复位源的欠*压检测使能延迟*（tBOD.ENABLE）

* + - 1. 调整检测阈值

欠*压检测阈值*（DVDDTH.BOD）在初始上电或复位事件后的标称值为0.86 V这应导致检测阈值在0.83 V和0.89 V之间。一旦复位，阈值可以在软件控制下调整新的检测阈值将在*欠压检测编程延迟*（（tBOD.PROG））后生效[图26](#_bookmark285)中示出了这样的示例。

通过写入[BOD](#_bookmark267)寄存器中的VSEL字段来调整阈值有关详细信息，请参阅[BOD](#_bookmark267)寄存器描述

*图26. 调整欠压检测阈值*

VSEL

1001

0111

tBOD.PROG

阈值0.86V

阈值0.774V

* + - 1. 详细规格

*表194.对妇女的歧视欠压检测参数*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 描述 | Min | Typ | Max | 单位 |
| DVDDTH.BOD | 欠压检测阈值 | 96.5 | 100 | 103.5 | 所选阈值电压 |
| tBOD活性 | 欠压检测激活延迟 |  | 55 | 80 | 微秒 |
| tBOD.ASSERT | 欠压检测断言延迟 |  | 3 | 10 | 微秒 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 描述 | Min | Typ | Max | 单位 |
| tBOD.ENABLE | 欠压检测使能延迟 |  | 35 | 55 | 微秒 |
| tBOD.PROG | 欠压检测编程延迟 |  | 20 | 30 | 微秒 |

表195.对妇女的歧视电压调节器输入电源监控器参数

### 供应监测器

上电和掉电重置模块由片内电压调节器的输入电源（VREG\_VIN）供电这些模块在第一次加电时初始化，但如果在VREG\_VIN下降到足够低的水平之前断电然后重新加电，则可能无法可靠地重新初始化。为防止这种情况发生，将监控VREG\_VIN，如果VREG\_VIN低于*VREG\_VIN激活阈值*（VREG\_VINTH.ACTIVE），则重新初始化上电复位模块。VREG\_VINTH.ACTIVE固定为标称1.1V，这将导致阈值在0.87V和1.26V之间。此阈值并不代表安全工作电压。VREG\_VIN必须降至该电压以下，才能可靠地重新初始化上电复位模块。为了安全工作，VREG\_VIN必须处于1.8V和3.3V之间的标称电压

* + - 1. 详细规格

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 描述 | Min | Typ | Max | 单位 |
| VREG\_VINTH.ACTIVE | VREG\_VIN  激活阈值 | 0.87 | 1.1 | 1.26 | V |

* + 1. 外部复位

芯片也可以通过将其RUN引脚拉低来复位RUN变为低电平将使芯片保持复位状态，而与内核电源（DVDD）和上电复位/欠压检测模块的状态如果所有其他复位源都已释放，则RUN变为高电平后，芯片将立即退出RUN可用于延长初始上电复位时间，也可由外部电源驱动，根据需要启动和停止芯片如果不使用RUN，则应将其绑高。

* + 1. 救援端口重置

该芯片也可以通过Rescue USB端口复位。这允许芯片从锁定状态恢复。除了复位芯片外，Rescue UART Port复位还会设置[CHIP\_UART](#_bookmark268)寄存器中的PSM\_RESTART\_FLAG这在启动时由引导代码检查，如果该位被设置，则使其进入安全状态更多信息请参见[2.3.4.2“Rescue DP”部分](#_bookmark114)

* + 1. 上次复位的来源

通过读取[CHIP\_RESET](#_bookmark268)寄存器中HAD\_POR、HAD\_RUN和HAD\_PSM\_RESTART字段的状态，可以确定最近一次HAD\_POR字段中的1表示与电源相关的复位，即上电或欠压启动的复位，HAD\_RUN字段中的1表示芯片上次由RUN引脚复位，HAD\_PSM\_RESTART字段中的1永远不应该有多个字段设置为1。

2.12.8. 登记册一览表

芯片级复位子系统与片上电压调节器共享寄存器地址空间。[第2.10.6节](#_bookmark264)列出了两个子系统的寄存器。共享地址空间在本文档的其他地方称为vreg\_and\_chip\_reset。

* 1. 加电状态机
     1. 概述

加电状态机以特定顺序从各种硬件块中移除复位上电状态机中的每个外设均由内部rst\_n低电平有效复位信号控制，并生成内部rst\_done高电平有效复位完成信号。上电状态机解除对每个外设的复位，等待该外设置位其rst\_done，然后解除对下一个外设的复位这种方法的一个重要用途是等待时钟源在芯片中干净地运行，然后才解除对时钟发生器的复位这避免了潜在的毛刺时钟被分配到芯片。

当芯片级复位子系统确认数字内核电源（DVDD）已通电且稳定，且RUN引脚为高电平时，上电状态机本身将退出复位此时，上电状态机通过其rst\_n\_run输出将许多其他块从复位中这是用来重置那些在启动时需要重置的东西，但如果重启了加电状态机，就不能重置该清单包括：

* 环形振荡器和晶体振荡器中的上电逻辑
* 在上电状态机重新启动期间必须保持运行的时钟分频器（CLK\_ref和CLK\_sys）
* 看门狗（包含临时寄存器，需要在上电状态机的软重启过程中保持
  + 1. 通电序列

*图27. 通电状态机序列。*

芯片级复位释放

环形振荡器

晶体振荡器

时钟发生器

复位控制器

芯片级复位和稳压器寄存器

XIP

（就地执行）

ROM/SRAM

总线结构

处理器复合体

上电状态机序列如下：

* 一旦数字内核电源（DVDD）通电且稳定，且RUN引脚为高电平（此时rst\_n\_run也被取消置位），芯片级复位子系统将取消上电状态机复位
* 环形振荡器启动。一旦纹波计数器已经看到足够数量的时钟边沿以指示环形振荡器稳定，
* 晶体振荡器复位无效。 此时晶体振荡器未启动，因此rst\_done立即置位。
* CLK\_ref和CLK\_sys时钟发生器退出复位。 在初始配置中，Vref\_ref从没有分频器的环形振荡器运行。 系统正在从系统参考运行。这些时钟是需要为其余的序列

进步

序列的其余部分相当简单，下面是按顺序逐个从reset中出来的：

* 重置控制器-用于重置所有非引导外设
* 芯片级复位和电压调节寄存器-由bootrom用于检查芯片的引导状态在

具体地，可以通过SWD设置CHIP\_RESTART\_FLAG寄存器中的PSM\_RESTART\_FLAG标志，以向引导代码指示闪存中存在坏代码，并且它应该停止执行。CHIP\_RESET寄存器的复位状态由芯片级复位子系统确定，不受上电状态机复位的影响

* XIP（就地执行）-由bootrom用于执行来自外部SPI闪存的代码
* ROM和SRAM-从ROM执行引导代码 SRAM用于处理器和总线结构。
* 总线结构-允许处理器与外围设备通信
* 处理器复合体-处理器终于可以开始运行了

复位后的最后一件事是处理器复合体。这包括核心0和核心1。两个内核都将从ROM开始执行引导代码bootrom做的第一件事就是读取核心id。此时，core1将进入睡眠状态，留下core0继续执行bootrom。处理器复合体有自己的复位控制和各种低功耗模式，这就是为什么内核0和内核1复位都被取消断言，尽管bootrom只需要内核0

* + 1. 寄存器控制

加电状态机是一个完全自动化的硬件。它不需要用户输入就可以工作。有一些寄存器控件可用于覆盖和查看加电状态机的状态。这允许上电状态机中的硬件块在必要时由软件复位还有一个WDSEL寄存器，用于控制看门狗复位所复位的内容。

* + 1. 与Watchdog的交互

如果看门狗触发，则可以从软件可编程位置重新启动通电状态机。例如，在处理器陷入无限循环的情况下，或者程序员以某种方式错误配置了芯片。需要注意的是，如果上电状态机中的外设设置了WDSEL位，则上电序列中该外设之后的所有外设也将复位，因为所选外设的rst\_done将被取消置位，其余外设的rst\_n将置位

* + 1. 登记册一览表

表196.对妇女的歧视PSM寄存器列表

PSM寄存器从基址0x40010000（在SDK中定义为[PSM\_BASE](#_bookmark39)）开始。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x0 | [FRCE\_ON](#_bookmark298) | 强制块退出复位（即电源启动） |
| 0x4 | [FRCE\_OFF](#_bookmark299) | 强制复位（即关闭电源） |
| 0x8 | [WDSEL](#_bookmark300) | 如果看门狗触发时应重置此外设，则设置为1 |
| 0xc | [做](#_bookmark301) | 指示外设的寄存器已准备好访问。 |

### [PSM](#_bookmark297)：FRCE\_ON寄存器

表197.对妇女的歧视FRCE\_ON寄存器

表198.对妇女的歧视FRCE\_OFF

寄存器

**偏移**：0x0

描述

强制块退出复位（即电源启动）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:17 | Reserved. | - | - | - |
| 16 | Proc1 |  | RW | 0x0 |
| 15 | 100 |  | RW | 0x0 |
| 14 | SiO |  | RW | 0x0 |
| 13 | VREG\_AND\_CHIP\_CHIP |  | RW | 0x0 |
| 12 | XIP |  | RW | 0x0 |
| 11 | SRAM 5 |  | RW | 0x0 |
| 10 | SRAM 4 |  | RW | 0x0 |
| 9 | SRAM 3 |  | RW | 0x0 |
| 8 | SRAM 2存储器 |  | RW | 0x0 |
| 7 | 静态随机存取存储器1 |  | RW | 0x0 |
| 6 | SRAM0 |  | RW | 0x0 |
| 5 | ROM |  | RW | 0x0 |
| 4 | 纺织面料 |  | RW | 0x0 |
| 3 | 重置 |  | RW | 0x0 |
| 2 | 时钟 |  | RW | 0x0 |
| 1 | XOSC |  | RW | 0x0 |
| 0 | ROSC |  | RW | 0x0 |

### [PSM](#_bookmark297)：FRCE\_OFF寄存器

**偏移**：0x4

描述

强制复位（即关闭电源）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:17 | Reserved. | - | - | - |
| 16 | Proc1 |  | RW | 0x0 |
| 15 | 100 |  | RW | 0x0 |
| 14 | SiO |  | RW | 0x0 |
| 13 | VREG\_AND\_CHIP\_CHIP |  | RW | 0x0 |
| 12 | XIP |  | RW | 0x0 |
| 11 | SRAM 5 |  | RW | 0x0 |
| 10 | SRAM 4 |  | RW | 0x0 |
| 9 | SRAM 3 |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 8 | SRAM 2存储器 |  | RW | 0x0 |
| 7 | 静态随机存取存储器1 |  | RW | 0x0 |
| 6 | SRAM0 |  | RW | 0x0 |
| 5 | ROM |  | RW | 0x0 |
| 4 | 纺织面料 |  | RW | 0x0 |
| 3 | 重置 |  | RW | 0x0 |
| 2 | 时钟 |  | RW | 0x0 |
| 1 | XOSC |  | RW | 0x0 |
| 0 | ROSC |  | RW | 0x0 |

表199.对妇女的歧视WDSEL

寄存器

### [PSM](#_bookmark297)：WDSEL寄存器

**偏移**：0x8

描述

如果看门狗触发时应重置此外设，则设置为1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:17 | Reserved. | - | - | - |
| 16 | Proc1 |  | RW | 0x0 |
| 15 | 100 |  | RW | 0x0 |
| 14 | SiO |  | RW | 0x0 |
| 13 | VREG\_AND\_CHIP\_CHIP |  | RW | 0x0 |
| 12 | XIP |  | RW | 0x0 |
| 11 | SRAM 5 |  | RW | 0x0 |
| 10 | SRAM 4 |  | RW | 0x0 |
| 9 | SRAM 3 |  | RW | 0x0 |
| 8 | SRAM 2存储器 |  | RW | 0x0 |
| 7 | 静态随机存取存储器1 |  | RW | 0x0 |
| 6 | SRAM0 |  | RW | 0x0 |
| 5 | ROM |  | RW | 0x0 |
| 4 | 纺织面料 |  | RW | 0x0 |
| 3 | 重置 |  | RW | 0x0 |
| 2 | 时钟 |  | RW | 0x0 |
| 1 | XOSC |  | RW | 0x0 |
| 0 | ROSC |  | RW | 0x0 |

### [PSM](#_bookmark297)：DONE注册

**偏移**：0xc

表200。完成注册

描述

指示外设的寄存器已准备好访问。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:17 | Reserved. | - | - | - |
| 16 | Proc1 |  | RO | 0x0 |
| 15 | 100 |  | RO | 0x0 |
| 14 | SiO |  | RO | 0x0 |
| 13 | VREG\_AND\_CHIP\_CHIP |  | RO | 0x0 |
| 12 | XIP |  | RO | 0x0 |
| 11 | SRAM 5 |  | RO | 0x0 |
| 10 | SRAM 4 |  | RO | 0x0 |
| 9 | SRAM 3 |  | RO | 0x0 |
| 8 | SRAM 2存储器 |  | RO | 0x0 |
| 7 | 静态随机存取存储器1 |  | RO | 0x0 |
| 6 | SRAM0 |  | RO | 0x0 |
| 5 | ROM |  | RO | 0x0 |
| 4 | 纺织面料 |  | RO | 0x0 |
| 3 | 重置 |  | RO | 0x0 |
| 2 | 时钟 |  | RO | 0x0 |
| 1 | XOSC |  | RO | 0x0 |
| 0 | ROSC |  | RO | 0x0 |

* 1. 子系统重置
     1. 概述

复位控制器允许软件控制所有外围设备的复位，这些外围设备对于启动RP 2040中的处理器并不重要。这包括：

* USB控制器
* Pio
* 外围设备，如UART、I2C、SPI、PWM、定时器、ADC
* 个pll
* IO和Pad寄存器

完整的列表可以在寄存器描述中看到

复位控制器的每个外设复位在上电时保持复位状态由软件来解除其打算使用的外设的复位请注意，如果您正在使用SDK，某些外围设备可能已经无法重置。

* + 1. 程序员模型

SDK定义了一个结构体来表示重置寄存器。

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2040/hardware\_structs/include/hardware/structs/resets.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2040/hardware_structs/include/hardware/structs/resets.h#L24-L113)第24-113

1. 类型定义结构{
2. \_REG\_（RESETS\_OFFSET）*//RESETS\_OFFSET*
3. //重置控件

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 27 | // | 0x01000000 | [二十四] ： | 中国（1） |
| 28 | // | 0x00800000 | [23日] ： | uart1（1） |
| 29 | // | 0x00400000 | [22日] ： | 中文（简体） |
| 30 | // | 0x00200000 | [21日] ： | 定时器（1） |
| 31 | // | 0x00100000 | [20日] ： | tbman（1） |
| 32 | // | 0x00080000 | [十九] ： | sysinfo（1） |
| 33 | // | 0x00040000 | 〔一八〕 ： | （1） |
| 34 | // | 0x00020000 | [17] ： | 中文（简体） |
| 35 | // | 0x00010000 | [十六] ： | spi0（1） |
| 36 | // | 0x00008000 | [十五] ： | rtc（1） |
| 37 | // | 0x00004000 | [十四] ： | PWM（1） |
| 38 | // | 0x00002000 | [十三] ： | USB接口（1） |
| 39 | // | 0x00001000 | [12] ： | 中文（简体） |
| 40 | // | 0x00000800 | [十一] ： | 中文（简体） |
| 41 | // | 0x00000400 | 【10】 ： | 中文（简体） |
| 42 | // | 0x00000200 | 〔九〕 ： | pads\_qspi（1） |
| 43 | // | 0x00000100 | 〔八〕 ： | pads\_bank（1） |
| 44 | // | 0x0000080 | 【7】 ： | （1） |
| 45 | // | 0x0000040 | 【6】 ： | i\_qspi（1） |
| 46 | // | 0x00000020 | 【5】 ： | io\_bank0（1） |
| 47 | // | 0x00000010 | 〔四〕 ： | 中文（简体） |
| 48 | // | 0x0000008 | 〔三〕 ： | 中文（简体） |
| 49 | // | 0x00000004 | 【2】 ： | dma（1） |
| 50 | // | 0x00000002 | 【一】 ： | 中国（1） |
| 51 | // | 0x00000001 | [0] ： | 中国（1） |

52io\_rw\_32复位;

53

1. \_REG\_（RESETS\_WDSEL\_OFFSET）*//RESETS\_WDSEL*
2. //Watchdogselect
3. //0x01000000[24] ：usbcket（0）
4. //0x00800000[23] ：uart1（0）
5. //0x00400000[22] ：uart0（0）
6. //0x 00200000[21] ：计时器（0）
7. //0x00100000[20] ：tbman（0）
8. //0x00080000[19] ：sysinfo（0）
9. //0x00040000[18] ：syscfg（0）
10. //0x00020000[17] ：spi1（0）
11. //0x00010000[16] ：spi0（0）
12. //0x00008000[15] ：rtc（0）
13. //0x00004000[14] ：pwm（0）
14. //0x 00002000[13] ：USB接口（0）
15. //0x 00001000[12] ：系统管理器（0）
16. //0x00000800[11] ：pio1（0）
17. //0x00000400[10] ：pio0（0）
18. //0x00000200[9] ：pads\_qspi（0）
19. //0x00000100[8] ：pads\_bank0（0）
20. //0x0000080[7] ：jtag（0）
21. //0x0000040[6] ：io\_qspi（0）
22. //0x0000020[5] ：io\_bank0（0）
23. //0x0000010[4] ：i2c1（0）
24. //0x0000008[3] ：i2c0（0）
25. //0x0000004[2] ：dma（0）
26. //0x00000002[1] ：busctrl（0）
27. //0x0000001[0] ：（0）

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

io\_rw\_32wdsel;

\_REG\_（RESETS\_RESETS\_DONE\_OFFSET）*//RESETS\_RESETS\_DONE*

//重置完成

//0x01000000[24]

//0x00800000[23]

//0x00400000[22]

//0x00200000[21]

//0x00100000[20]

//0x00080000[19]

//0x00040000[18]

//0x00020000[17]

//0x00010000[16]

//0x00008000[15]

//0x00004000[14]

//0x00002000[13]

//0x00001000[12]

//0x00000800[11]

//0x00000400[10]

//0x00000200[9]

//0x00000100[8]

//0x0000080[7]

//0x0000040[6]

//0x0000020[5]

//0x0000010[4]

//0x0000008[3]

//0x0000004[2]

//0x0000002[1]

//0x0000001[0]

io\_ro\_32reset\_done;

：usbcket（0）

：uart1（0）

：uart0（0）

：timer（0）

：tbman（0）

：sysinfo（0）

：syscfg（0）

：spi1（0）

：spi0（0）

：rtc（0）

：PWM（0）

：USB接口（0）

中文（简体）

：pio 1（0）

：pio0（0）

：pads\_qspi（0）

：pads\_bank0（0）

：jtag（0）

：io\_qspi（0）

：io\_bank0（0）

：i2c1（0）

：i2c0（0）

：dma（0）

：buscycle（0）

：中文（简体）

111}resets\_hw\_t;

112

113*#define resets\_hw（（resets\_hw\_t\*）RESETS\_BASE）*

定义了三个寄存器

* reset：该寄存器包含每个外设的一个可以复位的位。 如果该位设置为1，则复位有效。如果该位被清除，则复位无效。
* wdsel：如果该位被设置，那么如果看门狗触发，该外围设备将被重置（请注意，通电状态机可能会重置整个重置控制器，这将重置所有内容）
* reset\_done：每个外设的一个位，一旦外设脱离复位，该位就被设置。这允许软件等待此状态位，以防外设在使用前需要进行一些初始化

SDK中的重置函数定义如下：

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_resets/include/hardware/resets.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_resets/include/hardware/resets.h#L70-L72)第70-72

1. public static int findDuplicate（intfindDuplicate）{
2. hw\_set\_bits（resets\_hw->reset，bits）;

72}

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_resets/include/hardware/resets.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_resets/include/hardware/resets.h#L79-L81)第79-81

1. public static int findDuplicate（int[] nums）{
2. hw\_clear\_bits（resets\_hw->reset，bits）;

81}

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_resets/include/hardware/resets.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_resets/include/hardware/resets.h#L88-L92)第88-92

1. public static voidunreset\_block\_wait（ uint32\_tbits）{
2. hw\_clear\_bits（resets\_hw->reset，bits）;
3. while（~resets\_hw->reset\_done位）
4. return（）;

92}

在UART驱动程序中使用这些功能的一个示例是，驱动程序定义了一个uart\_reset函数，根据指定的uart选择复位寄存器的不同位

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_uart/uart.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_uart/uart.c#L27-L35)第27-35

1. publicstatic int findDuplicate
2. invalid\_params\_if（int，uart！=uart0 uart！=uart1）;
3. reset\_block（uart\_get\_index（uart）？RESETS\_RESET\_UART1\_BITS：RESETS\_RESET\_UART0\_BITS）;

30}

31

1. public static voidunreset（unreset）{
2. invalid\_params\_if（int，uart！=uart0 uart！=uart1）;
3. unreset\_block\_wait（uart\_get\_index（uart）？RESETS\_RESET\_UART1\_BITS：RESETS\_RESET\_UART0\_BITS）;

35}

* + 1. 登记册一览表

表201.对妇女的歧视复位寄存器列表

表202.对妇女的歧视复位

寄存器

复位控制器寄存器从基址0x4000c000（在SDK中定义为[RESETS\_BASE](#_bookmark39)）开始。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x0 | [复位](#_bookmark307) | 重置控制。 |
| 0x4 | [WDSEL](#_bookmark308) | 看门狗选择。 |
| 0x8 | [已完成](#_bookmark309) | 重置完成。 |

### [RESETS](#_bookmark306)：重置注册

**偏移**：0x0

描述

重置控制。 如果设置了一个位，则表示外设处于复位状态。 0表示外设的重置已取消断言。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:二十5分 | Reserved. | - | - | - |
| 24 | USBCTRL |  | RW | 0x1 |
| 23 | UART1 |  | RW | 0x1 |
| 22 | UART0 |  | RW | 0x1 |
| 21 | 定时器 |  | RW | 0x1 |
| 20 | TBMAN |  | RW | 0x1 |
| 19 | SYSINFO |  | RW | 0x1 |
| 18 | SYSCFG |  | RW | 0x1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 17 | SPI1 |  | RW | 0x1 |
| 16 | SPI0 |  | RW | 0x1 |
| 15 | RTC |  | RW | 0x1 |
| 14 | PWM |  | RW | 0x1 |
| 13 | PLL\_USB |  | RW | 0x1 |
| 12 | PLL\_PLL |  | RW | 0x1 |
| 11 | PIO 1 |  | RW | 0x1 |
| 10 | PIO0 |  | RW | 0x1 |
| 9 | PADS\_QSPI |  | RW | 0x1 |
| 8 | PADS\_BANK0 |  | RW | 0x1 |
| 7 | JTAG |  | RW | 0x1 |
| 6 | IO\_QSPI |  | RW | 0x1 |
| 5 | IO\_BANK0 |  | RW | 0x1 |
| 4 | I2C1 |  | RW | 0x1 |
| 3 | I2c0 |  | RW | 0x1 |
| 2 | DMA |  | RW | 0x1 |
| 1 | 总线控制 |  | RW | 0x1 |
| 0 | ADC |  | RW | 0x1 |

表203.对妇女的歧视WDSEL

寄存器

### [复位](#_bookmark306)：WDSEL寄存器

**偏移**：0x4

描述

看门狗选择。 如果设置了某个位，则看门狗将在触发时重置此外设。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:二十5分 | Reserved. | - | - | - |
| 24 | USBCTRL |  | RW | 0x0 |
| 23 | UART1 |  | RW | 0x0 |
| 22 | UART0 |  | RW | 0x0 |
| 21 | 定时器 |  | RW | 0x0 |
| 20 | TBMAN |  | RW | 0x0 |
| 19 | SYSINFO |  | RW | 0x0 |
| 18 | SYSCFG |  | RW | 0x0 |
| 17 | SPI1 |  | RW | 0x0 |
| 16 | SPI0 |  | RW | 0x0 |
| 15 | RTC |  | RW | 0x0 |
| 14 | PWM |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 13 | PLL\_USB |  | RW | 0x0 |
| 12 | PLL\_PLL |  | RW | 0x0 |
| 11 | PIO 1 |  | RW | 0x0 |
| 10 | PIO0 |  | RW | 0x0 |
| 9 | PADS\_QSPI |  | RW | 0x0 |
| 8 | PADS\_BANK0 |  | RW | 0x0 |
| 7 | JTAG |  | RW | 0x0 |
| 6 | IO\_QSPI |  | RW | 0x0 |
| 5 | IO\_BANK0 |  | RW | 0x0 |
| 4 | I2C1 |  | RW | 0x0 |
| 3 | I2c0 |  | RW | 0x0 |
| 2 | DMA |  | RW | 0x0 |
| 1 | 总线控制 |  | RW | 0x0 |
| 0 | ADC |  | RW | 0x0 |

表204.对妇女的歧视注册\_完成

### [RESETS](#_bookmark306)：重置\_完成注册

**偏移**：0x8

描述

重置完成。如果设置了一个位，则外设已返回重置完成信号。这表明外设的寄存器已准备好被访问。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:二十5分 | Reserved. | - | - | - |
| 24 | USBCTRL |  | RO | 0x0 |
| 23 | UART1 |  | RO | 0x0 |
| 22 | UART0 |  | RO | 0x0 |
| 21 | 定时器 |  | RO | 0x0 |
| 20 | TBMAN |  | RO | 0x0 |
| 19 | SYSINFO |  | RO | 0x0 |
| 18 | SYSCFG |  | RO | 0x0 |
| 17 | SPI1 |  | RO | 0x0 |
| 16 | SPI0 |  | RO | 0x0 |
| 15 | RTC |  | RO | 0x0 |
| 14 | PWM |  | RO | 0x0 |
| 13 | PLL\_USB |  | RO | 0x0 |
| 12 | PLL\_PLL |  | RO | 0x0 |
| 11 | PIO 1 |  | RO | 0x0 |
| 10 | PIO0 |  | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 9 | PADS\_QSPI |  | RO | 0x0 |
| 8 | PADS\_BANK0 |  | RO | 0x0 |
| 7 | JTAG |  | RO | 0x0 |
| 6 | IO\_QSPI |  | RO | 0x0 |
| 5 | IO\_BANK0 |  | RO | 0x0 |
| 4 | I2C1 |  | RO | 0x0 |
| 3 | I2c0 |  | RO | 0x0 |
| 2 | DMA |  | RO | 0x0 |
| 1 | 总线控制 |  | RO | 0x0 |
| 0 | ADC |  | RO | 0x0 |

* 1. 时钟
     1. 概述

时钟模块为片内和外部器件提供独立的时钟。它采用来自各种时钟源的输入，使用户能够在性能与成本、电路板面积和功耗之间进行权衡从这些来源，它使用多个时钟发生器来提供所需的时钟。 这种架构允许用户灵活地独立启动和停止时钟，并改变某些时钟频率，同时将其他时钟保持在其最佳频率。

*图28. 时钟概述*



对于不需要精确定时的低成本或低功耗应用，该芯片可以从内部环形振荡器（ROSC）运行或者，用户可以提供外部时钟或构造简单的弛张振荡器

使用GPIO和适当的外部无源元件。在时序更为关键的情况下，晶体振荡器（XOSC）可以为2个片内PLL提供精确的基准电压，从而以精确的频率提供快速时钟

时钟发生器从时钟源中进行选择，并在通过使能逻辑输出之前对选定的时钟进行分频，使能逻辑在睡眠模式下提供自动时钟禁用（参见[第2.11.2节](#_bookmark272)）。

片内频率计数器有助于时钟设置的调试，还允许测量外部时钟的频率如果系统时钟意外停止，片内resus组件会从已知正常的时钟重新启动系统时钟这允许软件调试器访问寄存器并调试问题。

该芯片具有一种称为DORMANT的超低功耗模式（参见[第2.11.3节](#_bookmark273)），在该模式下，所有片内时钟源都将停止以节省功耗。外部源不会停止，可用于向片内RTC提供时钟，片内RTC可提供警报，将芯片从休眠模式唤醒。或者，GPIO中断可以配置为响应外部事件将芯片从休眠模式唤醒。

最多可将4个生成的时钟以高达50 MHz的频率输出到GPIO。这使得用户可以为外部设备提供时钟，从而减少功耗、空间和成本敏感型应用中的元件数量。

* + 1. 时钟源

RP2040可以从各种时钟源运行。这种灵活性使用户能够针对性能、成本、电路板面积和功耗优化时钟设置。源包括片内环形振荡器（第2.17）、晶振（第2.16）、GPIO外部时钟（[第2.15.6.4](#_bookmark320)）和PLL（第2.18）。

每个时钟发生器的时钟源列表各不相同，可以在CTRL寄存器中找到枚举值。请参见[CLK\_CLK\_CTRL](#_bookmark338)示例。

* + - 1. 环形振荡器

片内环形振荡器（第2.17）不需要外部元件。它在上电时自动运行，用于在初始引导阶段为芯片提供时钟。启动频率通常为6MHz，但随PVT（工艺、电压和温度）而变化。频率可能在4- 8 MHz范围内，并保证在1.8- 12 MHz范围内。

对于频率精度不重要的低成本应用，芯片可以继续从ROSC运行。如果需要更高的性能，可以通过对寄存器进行编程来增加频率，如第

两点17频率会随着PVT（工艺、电压和温度）而变化，因此用户必须注意避免超过时钟发生器部分中描述的最大频率。如果用户希望以接近最大值的频率继续从ROSC运行，则可以通过各种方式减轻这种变化（参见[第2.15.2.1.1](#_bookmark313)）或者，用户可以使用外部时钟或XOSC提供稳定的参考时钟，并使用PLL产生更高的频率。这将需要外部元件，这将占用电路板面积并增加功耗。

如果使用外部时钟或XOSC，则可以停止ROSC以节省功率。但是，在这样做之前，参考时钟发生器和系统时钟发生器必须切换到备用源

ROSC不受睡眠模式的影响。如果需要，可以在进入睡眠模式之前降低频率以节省功率。在进入休眠模式时，ROSC自动停止，并在退出休眠模式时以相同的配置重新启动如果ROSC以接近其最大频率驱动时钟，则建议在进入睡眠或休眠模式之前降低频率，以允许由于睡眠或休眠模式期间环境条件的变化而引起的频率变化

如果用户希望在外部使用ROSC时钟，则可以使用其中一个clk\_gpclk0-3发生器将其输出到GPIO引脚。

以下部分描述用于减轻ROSC频率的PVT变化的技术 他们还提供了一些有趣的设计挑战，用于教学PVT的影响和编写软件来控制实时功能。

##### 注意

ROSC频率随PVT变化，因此用户可以将其输出发送到频率计数器，并使用它来测量这3个变量中的任何1个，如果其他2个已知。

* + - * 1. 减轻工艺引起的ROSC频率变化

过程因两个原因而不同。首先，芯片离开工厂时具有工艺参数的扩展，这导致跨芯片的ROSC频率的其次，工艺参数随着芯片老化而略有变化，尽管这只能在数千小时的操作中观察到。为了减轻工艺变化，用户可以对单个芯片进行重新配置，并相应地对ROSC频率进行编程。这对于少量芯片来说是一种足够的解决方案，但不适合批量生产。在此类应用中，用户应考虑使用下文所述的自动缓解技术。

* + - * 1. 减轻由于电压引起的ROSC频率变化

电源电压变化有两个原因。首先，电源本身可能会变化，其次，随着芯片活动的变化，将存在变化的如果应用具有最低性能目标，则用户需要针对该应用进行校准并调整ROSC频率，以确保其始终超过所需的最低值

* + - * 1. 减轻由于温度引起的ROSC频率变化

温度变化有两个原因。首先，环境温度可能变化，其次，芯片温度将随着芯片活动由于自加热而变化而变化。 这可以通过使用温度控制环境和被动或主动冷却来稳定温度来减轻。或者，用户可以使用片内温度传感器跟踪温度，并调整ROSC频率，使其保持在所需的范围内。

* + - * 1. 自动缓解由PVT引起的ROSC频率变化

用于自动ROSC频率控制的技术避免了校准各个芯片的需要，但是需要周期性地访问时钟基准或时间基准。如果时钟参考可用，则其可用于周期性地测量ROSC频率并相应地对其进行调整基准电压源可以是芯片上的XOSC，它可以周期性地开启。这在非常低功率的应用中可能是有用的，其中连续运行XOSC成本太高，并且使用PLL来实现高频成本太高如果时间基准可用，则用户可以从ROSC为片内RTC计时，并定期将其与时间基准进行比较，然后根据需要调整使用这些技术，ROSC频率将由于VT变化而漂移，因此用户必须注意这些变化不允许ROSC频率漂移出可接受的范围。

* + - * 1. 使用ROSC自动超频

任何数字设备的最大频率都是针对最坏情况的PVT而引用的。大多数芯片在大多数正常环境中的运行速度都比引用的最大频率快得多，因此可以进行超频。如果RP2040从ROSC运行，那么ROSC和数字组件都受到PVT的类似影响，因此，随着ROSC变得更快，处理器也可以运行得更快。这意味着用户可以从ROSC超频，然后依靠ROSC频率跟踪PVT变化。 ROSC频率和处理器能力的跟踪并不完美，目前没有足够的数据来指定这种操作模式的安全ROSC设置，因此需要进行一些实验。

这种操作模式将使处理器性能最大化，但会导致完成任务所需时间的变化此外，如果用户希望使用频率敏感接口（如USB或PLL），则必须使用XOSC和PLL为这些组件提供精确的时钟。

* + - 1. 晶体振荡器

晶体振荡器（第2.16节）提供精确、稳定的时钟基准，应用于需要精确定时且没有合适外部时钟的场合。频率由外部晶体决定，振荡器支持1MHz至15MHz范围内的频率。如果需要，片内PLL可用于合成更高的频率。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。使用XOSC和PLL，片内组件可以以其最高频率运行设计中内置了适当的裕度，可容忍XOSC频率高达1000 ppm的变化

XOSC在上电时处于非活动如果需要，必须在软件中启用XOSC启动需要几毫秒时间，软件必须等待XOSC\_STABLE标志置位，然后才能启动PLL和更改任何时钟发生器以使用它。在此之前，XOSC的输出可能不存在，或者可能具有非常短的脉冲宽度，如果使用，则会损坏逻辑一旦它运行，参考时钟（REF\_REF）和系统时钟（SYS\_SYS）可以被切换为从XOSC运行，并且ROSC可以被停止以节省功率。

XOSC不受睡眠模式的影响。当进入和退出休眠模式时，它会在相同的配置下自动停止和重新启动。

如果用户想要在外部使用XOSC时钟，则可以使用其中一个clk\_gpclk0-3发生器将其输出到GPIO引脚。它不能直接从XIN或XOUT引脚获取

* + - 1. 外部时钟

如果您的硬件设计中存在外部时钟，那么它们可以单独或与XOSC或ROSC一起用于为RP 2040提供时钟。这将潜在地节省功率，并允许RP 2040上的组件与外部组件同步运行，以简化芯片之间的数据传输。外部时钟可以通过GPIN0 GPIN1 GPIO输入和XOSC的XIN输入。如果以这种方式使用XIN输入，则必须将XOSC配置为通过XIN信号。所有3个输入都限制在50 MHz，但如果需要，片内PLL可以用于从XIN输入合成更高的频率如果外部时钟的频率精度低于1000 ppm，则生成的时钟不应以其最大频率运行，因为它们可能超过其设计裕度。

一旦外部时钟正在运行，参考时钟（REF\_REF）和系统时钟（SYS\_SYS）可以切换为从外部时钟运行

外部时钟源不受睡眠模式或休眠模式的影响。

* + - 1. 张弛振荡器

如果用户想要使用外部时钟来替换或补充其他时钟源，但没有合适的时钟可用，则可以使用外部无源元件构建1或2个弛张振荡器。只需将时钟源（GPIN 0或GPIN 1）发送到其中一个gpclk 0 -3发生器，通过GPIO反相器OUTOVER将其反相，然后通过RC电路连接回时钟源输入

*图29. 简单的弛张振荡器示例*



弛张振荡器产生的时钟频率取决于通过芯片的延迟和来自GPIO输出的驱动电流，这两者都随PVT而变化。它们还取决于外部元件的质量和精度。使用更精细的外部元件（如陶瓷谐振器）可以提高频率精度，出于这个原因，这里不讨论它们。鉴于这些振荡器无法达到1000 ppm，因此它们无法用于以其最大频率驱动内部时钟。

张驰振荡器不受睡眠模式或休眠模式的影响。

* + - 1. 个pll

PLL（第2.18）用于在从XOSC（或驱动到XIN引脚的外部时钟源）运行时提供快速时钟在功能齐全的应用中，USB PLL为ADC和USB提供固定的48 MHz时钟，而ADC\_rtc和ADC\_ref由XOSC或外部源驱动这样，用户就可以从系统PLL驱动PLL\_sys可从固定频率USB PLL或变频系统PLL驱动PLL\_PLL。如果PLL\_sys永远不需要超过48 MHz，则可以使用一个PLL，PLL\_sys时钟发生器中的分频器可用于根据需求调整PLL\_sys频率。

当PLL启动时，其输出不能使用，直到PLL锁定（如STATUS寄存器中的复位位所示）此后，PLL输出在参考时钟分频器、输出分频器或旁路模式变化期间不能使用。输出可以在反馈除数变化期间使用，条件是输出频率可能在反馈除数的大变化上过冲或下冲有关更多信息，请参见第2.18。

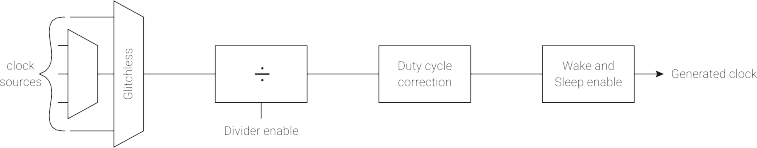
如果PLL参考时钟精确到1000 ppm，则PLL可用于以其最大频率驱动时钟，因为生成的时钟频率将在设计允许的裕度内

PLL不受SLEEP模式影响。如果用户希望在睡眠模式下省电，则必须将所有时钟发生器从PLL切换开，并且在进入睡眠模式之前必须在软件中停止它们。当进入和退出休眠模式时，PLL不会自动停止和重启。如果它们在进入DORMANT模式时继续运行发生这种情况是因为它们来自XOSC的参考时钟将停止。因此，在进入DORMANT模式之前，必须

* + 1. 时钟发生器

时钟发生器采用标准设计，集成了时钟源多路复用、分频、占空比校正和睡眠模式启用功能。为了节省芯片面积和功耗，单个时钟发生器不支持所有功能。

*图30. 通用时钟发生器*



* + - 1. 实例

RP2040有几个时钟发生器，如下所示。

*表205.对妇女的歧视RP2040*

|  |  |  |
| --- | --- | --- |
| 时钟 | 描述 | 标称频率 |
| clk\_gpout0 | 时钟输出至GPIO。 可用于外部设备时钟或与逻辑分析仪或示波器调试片上时钟。 | N/A |
| clk\_gpout1 |
| clk\_gpout2 |
| clk\_gpout3 |

时钟发生器

|  |  |  |
| --- | --- | --- |
| 时钟 | 描述 | 标称频率 |
| 联系我们 | 除非处于休眠模式，否则始终运行的参考时钟在上电时从环形振荡器（ROSC）切换到晶体振荡器（XOSC），但可以切换到更高的精度。 | 6 - 12MHz |
| 联系我们 | 除非处于休眠模式，否则始终运行的系统时钟。上电时从PLL\_ref复位，但通常切换到PLL。 | 125MHz |
| 联系我们 | 外设时钟。通常从XNUS\_SYS运行，但如果XNUS\_SYS被软件更改，则允许外围设备以一致的速度运行 | 12-125MHz |
| USB接口 | USB参考时钟。 必须是48MHz。 | 48MHz |
| 联系我们 | ADC参考时钟。 必须是48MHz。 | 48MHz |
| 实时时钟 | RTC参考时钟。 RTC将此时钟分频以生成1秒基准电压。 | 46875Hz |

有关每个时钟发生器的时钟源的完整列表，请参见相应的CTRL寄存器。例如，[CLK\_CLK\_CTRL](#_bookmark338)。

* + - 1. 复用器

所有时钟发生器都有一个多路复用器，称为辅助（aux）mux。该多路复用器采用传统设计，其输出在改变选择控制时会出现毛刺。两个时钟发生器（CLK\_SYS和CLK\_REF）都有一个额外的多路复用器，称为无毛刺多路复用器。无毛刺多路复用器可以在时钟源之间切换，而不会在输出端产生毛刺。

应不惜一切代价避免时钟故障，因为它们可能会破坏该时钟上运行的逻辑这意味着，在切换时钟源时，必须禁用任何仅具有辅助多路复用器的时钟发生器如果时钟发生器有一个无毛刺多路复用器（PLL\_SYS和PLL\_REF），则在更改辅助多路复用器源时，无毛刺多路复用器应切换离开辅助多路复用器时钟发生器需要2个源时钟周期来停止输出，需要2个新源时钟周期来重新启动输出。用户在更改辅助多路复用器之前必须等待发生器停止，因此必须了解源时钟频率。

无毛刺多路复用器仅适用于常开时钟。在RP 2040上，始终开启的时钟是参考时钟（REF\_REF）和系统时钟（SYS\_SYS）。这些时钟必须连续运行，除非芯片处于休眠模式。无毛刺多路复用器有一个状态输出（SELECTED），指示选择了哪个源，并可从软件中读取，以确认时钟源的更改已完成。

推荐的控制顺序如下。

切换无毛刺多路复用器：

* 将无毛刺多路复用器切换到备用源
* 轮询SELECTED寄存器，直到切换完成

若要在发生器具有无毛刺多路复用器时切换辅助多路复用器

* 将无毛刺多路复用器切换到不是辅助多路复用器的源
* 轮询SELECTED寄存器，直到切换完成
* 改变辅助多路选择控制
* 将无毛刺多路复用器切换回辅助多路复用器
* 如果需要，轮询SELECTED寄存器，直到切换完成

若要在发生器没有无毛刺多路复用器时切换辅助多路复用器

* 禁用时钟分频器
* 等待生成的时钟停止（时钟源的2个周期
* 改变辅助多路选择控制
* 使能时钟分频器
* 如果需要，等待时钟发生器重新启动（时钟源的2个周期）。有关此代码示例，请参见[第2.15.6.1节](#_bookmark319)
  + - 1. 分频器

一个功能齐全的除法器可以除以1或2.0到2^24-0.01范围内的小数 分数分频是通过在2个整数除数之间切换来实现的，因此它会产生抖动时钟，这可能不适合某些应用。例如，当除以2.4时，分频器将除以2为3个周期，除以3为2个周期。对于具有大整数分量的除数，抖动将小得多并且不那么关键。

*图31. 分数除法的一个例子。*



所有分频器都支持动态除数变化，这意味着输出时钟将从一个除数干净地切换到另一个除数。时钟发生器在时钟除数变化期间不需要停止它通过将除数变化同步到时钟周期的末尾来实现这一点类似地，使能与时钟周期的结束同步，因此在使能或禁用时钟发生器时不会产生毛刺。常开时钟的时钟发生器永久使能，因此没有使能控制。

如果时钟发生器锁定并且从未完成当前时钟周期，则可以强制停止使用KILL控制。这可能导致输出毛刺，从而损坏时钟驱动的逻辑。因此，建议在此操作之前重置目标逻辑。值得一提的是，这种时钟发生器设计已经在众多芯片中使用，从未被锁定过。KILL控件既不优雅又不必要，不应用作enable的替代选项。常通时钟的时钟发生器是永久激活的，因此没有KILL控制。

* + - 1. 占空比校正

分频器在输入时钟的上升沿工作，因此在除以奇数时不会产生偶数占空比时钟。

除以3将得到33.3%的占空比，除以5将得到40%等。如果使能，占空比校正逻辑将把输出时钟的下降沿移到输入时钟的下降沿，并恢复50%的占空比。时钟运行时，可以启用和禁用占空比校正当除以偶数时，它将不起作用

*图32.duty\_cycle\_correction的例子。*

时钟源

生成的时钟（无DCC）

生成的时钟

与DCC

* + - 1. 时钟使能

每个时钟指向多个目的地，除了少数例外，每个目的地有2个使能。WAKE\_EN寄存器用于在系统唤醒时使能时钟，SLEEP\_EN寄存器用于在系统处于休眠模式时使能时钟这些使能的目的是降低时钟分配网络中未使用组件的功耗值得注意的是，未计时的组件将保留其配置，因此可以快速重新启动。

##### 注意

WAKE\_EN和SLEEP\_EN寄存器复位为0x1，这意味着默认情况下所有时钟均使能。如果程序员需要低功耗设计，他们只需要使用此功能

* + - * 1. 时钟使能

处理器内核没有时钟使能，因为它们始终需要时钟来管理自己的节能功能。

在唤醒模式下不能禁用EXP\_sys\_busfabric，因为这将阻止内核访问任何芯片寄存器，包括那些控制时钟使能的

由于禁用唤醒模式会阻止内核访问时钟控制寄存器，因此，SYS\_SYS\_Clocks没有唤醒模式使能。

gpclk没有时钟使能。

* + - * 1. 系统休眠模式

当两个内核都处于睡眠状态并且DMA没有未完成的事务时，系统会自动进入睡眠模式。在系统休眠模式下，前面段落中描述的时钟使能从WAKE\_EN寄存器切换到SLEEP\_EN寄存器。其目的是降低芯片处于非活动状态时时钟分配网络中的功耗如果用户未配置WAKE\_EN和SLEEP\_EN寄存器，则系统休眠将不执行任何操作。

在内核进入睡眠状态之前，如果不采取其他措施来降低功耗，那么使用系统睡眠的价值就很小。需要考虑的事项包括：

* 停止未使用的时钟源，如PLL和晶振
* 通过增加时钟除数来降低生成时钟的频率
* 停止外部时钟

为了在芯片处于非活动状态时实现最大的省电，用户应考虑DORMANT（参见[第2.11.3](#_bookmark273)）模式，在该模式下，时钟来自晶体振荡器和/或环形振荡器，并且这些时钟源被停止。

* + 1. 频率计数器

表206.对妇女的歧视频率计数器测试间隔与准确度

频率计数器通过对测试间隔内的时钟边沿进行计数来测量内部和外部时钟的频率该间隔通过计数必须从XOSC或从已知频率的稳定外部源驱动的Vref\_ref的周期来定义。

用户可以使用[FC0\_INTERVAL](#_bookmark358)寄存器在精度和测试时间之间进行选择 [表206](#_bookmark316)显示了权衡。

|  |  |  |
| --- | --- | --- |
| 间隔寄存器 | 测试间隔 | 精度 |
| 0 | 1μs | 2048kHz |
| 1 | 2μs | 1024kHz |
| 2 | 4μs | 512kHz |
| 3 | 8μs | 256kHz |
| 4 | 16μs | 128kHz |
| 5 | 32μs | 64kHz |
| 6 | 64μs | 32kHz |
| 7 | 125μs | 16kHz |
| 8 | 250μs | 8kHz |
| 9 | 500μs | 4kHz |
| 10 | 1ms | 2kHz |
| 11 | 2ms | 1kHz |
| 12 | 4ms | 500Hz |
| 13 | 8ms | 250Hz |
| 14 | 16ms | 125Hz |
| 15 | 32Ms | 62.5Hz |

* + 1. Resus

编写的软件可能会无意中停止MySQL\_sys。这通常会导致内核和片上调试器不可恢复的锁定，使用户无法跟踪问题。为了减轻这种情况，提供了一个自动复苏电路，如果在用户定义的时间间隔内没有检测到边沿，该电路将把clk\_sys切换到已知良好的时钟源已知良好的源是可从XOSC、ROSC或外部源驱动的Vref\_ref。

resus块在由residue\_ref控制的超时间隔内对residue\_sys上的边进行计数，如果没有检测到residue\_sys边，则强制从residue\_ref驱动residue\_sys该间隔可通过[CLK\_CLK\_RESUS\_CTRL](#_bookmark352)编程。

##### 警告

如果resus\_ref也被停止，resus就没有办法恢复芯片

要使能RESUS，编程器必须设置超时间隔，然后设置[CLK\_RESUS\_CTRL](#_bookmark352)中的ENABLE位。要检测一个resus事件，必须通过设置INTE中的中断使能位来使能CLK\_RESUS\_RESUS中断。CLOCK\_DEFAULT\_IRQ（参见[第2.3.2节](#_bookmark111)）也必须在处理器上启用。

Resus旨在作为调试辅助工具。目的是让用户跟踪触发resus的软件错误通过重新配置RESUS\_SYS，然后通过在[CLK\_RESUS\_CTRL](#_bookmark352)中写入CLEAR位来清除RESUS，可以在resus事件后继续运行。然而，应该注意的是，

由运行速度比预期慢的resus\_sys触发，并且在触发resus时可能导致resus\_sys故障那个小故障会损坏芯片这是一种罕见的事件，但在调试场景中是可以容忍的。但是，在正常操作中这是不可接受的，因此建议仅使用resus进行调试。

**警告**

Resus是一种调试辅助工具，不应用作正常操作中切换时钟的方法。

* + 1. 程序员模型
       1. 配置时钟发生器

SDK定义了一个时钟枚举

*SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2040/hardware\_structs/include/hardware/structs/clocks.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2040/hardware_structs/include/hardware/structs/clocks.h#L27-L39)第27-39*

27enum clock\_index{

|  |  |  |  |
| --- | --- | --- | --- |
| 28 | clk\_gpout0 =0， | ///< | GPIO复用0 |
| 29 | clk\_gpout1， | ///< | GPIO复用1 |
| 30 | clk\_gpout2， | ///< | GPIO Muxing2 |
| 31 | clk\_gpout3， | ///< | GPIO Muxing3 |
| 32 | \_ref， | ///< | 看门狗和定时器参考时钟 |
| 33 | \_sys， | ///< | 处理器、总线结构、内存、内存映射寄存器 |
| 34 | \_ | ///< | 用于UART和SPI的外设时钟 |
| 35 | USB接口， | ///< | USB时钟 |
| 36 | \_ | ///< | ADC时钟 |
| 37 | \_rtc， | ///< | 实时时钟 |
| 38 | 时钟计数 |  |  |
| 39}; |  |  |  |

也是一个描述时钟发生器寄存器的结构

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2040/hardware\_structs/include/hardware/structs/clocks.h](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2040/hardware_structs/include/hardware/structs/clocks.h#L43-L63)第43-63

43typedef struct{

44

45

46

\_REG\_（CLOCKS\_CLK\_GPOUT0\_CTRL\_OFFSET）*//CLOCKS\_CLK\_GPOUT0\_CTRL*

//时钟控制，可以动态更改（auxsrc除外）

//0x00100000[20]

输入时钟的1个周期

：NUDGE（0）：此信号上的边沿将输出的相位偏移

47 *//0x 00030000 [17：16]：PHASE（0）：这会将使能信号延迟最多3个周期。*

输入时钟

48

49

50

51

//0x00001000[12]

//0x00000800[11]

//0x00000400[10]

//0x000001e0[8：5]

切换

io\_rw\_32密码;

：DC50（0）：使能奇数除数的占空比校正

：ENABLE（0）：干净地启动和停止时钟发生器

：KILL（0）：异步终止时钟发生器

：AUXSRC（0）：关闭辅助时钟源，当

52

53

54

55

56

57

58

59

60

61

62

\_REG\_（CLOCKS\_CLK\_GPOUT0\_DIV\_OFFSET）*//CLOCKS\_CLK\_GPOUT0\_DIV*

//时钟除数，可以动态更改

//0xffffff 00 [31：8]： INT（1）：除数的整数分量，0->除以2^16

//0x00000ff[7：0]

io\_rw\_32div;

：FRAC（0）：除数的分数分量

\_REG\_（时钟\_CLK\_GPOUT0\_SELECTED\_OFFSET）*//时钟\_CLK\_GPOUT0\_SELECTED*

//指示无毛刺多路复用器当前选择哪个SRC（one-hot）

选择io\_ro\_32

63}clock\_hw\_t;

要配置时钟，我们需要了解以下信息：

* + - * + 述时钟源的频率
        + 时钟源的mux/aux mux位置
        + 期望的输出频率

SDK提供clock\_configure来配置时钟：

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_clocks/clocks.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/clocks.c#L48-L118)第48-118

48 bool clock\_configure（enumclock\_index，uint32\_tsrc，uint32\_tauxsrc，uint32\_t

src\_freq，uint32\_tfreq）{

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

uint32\_tdiv;

assert（src\_freq）;

if（freq> src\_freq） returnfalse;

//Div寄存器是24.8 int.frac分频器，因此乘以2^8（左移8）

div =（uint32\_t）（uint64\_t）src\_freq CLOCKS\_CLK\_GPOUT0\_DIV\_INT\_LSB）/freq）; clock\_hw\_t \*clock = clocks\_hw-> clock [clock\_index];

//如果是递增的除数，则在source之前设置除数否则设置源

//在除数之前。这避免了瞬间的混乱，例如，切换

//到更快的源并增加除数来补偿。

if（div> clock->div）clock->div =div;

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

//如果将无毛刺切片（ref或sys）切换到辅助源，则切换

//远离aux \*first\*，以避免在更改aux mux时传递毛刺。

//假设（！）无毛刺源0不比AUX源快。

如果（has\_glitchless\_mux（bandwidth\_index）src ==时钟\_时钟\_CTRL\_SRC\_VALUE\_CLKSRC\_CLK\_CLK\_CLK\_AUX）{

hw\_clear\_bits（&clock-> clock，CLOCKS\_CLK\_REF\_CTRL\_SRC\_BITS）; while（！（clock->selected&1u））

return（）;

}

//如果没有无毛刺多路复用器，则完全停止时钟以避免毛刺

//当改变辅助多路复用器时传播注意，如果你不想让我

//在其中一个无毛刺时钟（clock\_sys，clock\_ref）上执行此操作。

否则{

//禁用时钟。在blog\_ref和blog\_sys上，这不做

*//所有其他时钟都在同一位置具有ENABLE位。*hw\_clear\_bits（clock-> clock，CLOCKS\_CLK\_GPOUT0\_CTRL\_ENABLE\_BITS）;if（configured\_freq[配置\_index] >0）{

//对于ENABLE传播，延迟目标时钟的3个周期。

//注意，XOSC\_REQ在这里没有帮助，因为XOSC不是

//不一定要运行，也不是定时器...：

uint delay\_cyc = configured\_freq[配置\_sys]/configured\_freq[配置\_index]+1; busy\_wait\_at\_least\_cycles（delay\_cyc \*3）;

}

}

//首先设置辅助多路复用器，然后设置无毛刺多路复用器（如果此时钟有）

hw\_write\_masked（时钟->时钟，

（auxsrc CLOCKS\_CLK\_CLK\_CTRL\_AUXSRC\_LSB），CLOCKS\_CLK\_CLK\_CTRL\_AUXSRC\_BITS

|  |  |  |
| --- | --- | --- |
| 95 |  | ); |
| 96 |  |  |
| 97 |  | if（has\_glitchless\_mux（has\_index））{ |
| 98 |  | hw\_write\_masked（时钟->时钟， |
| 99 |  | srcCLOCKS\_CLK\_REF\_CTRL\_SRC\_LSB， |
| 100 |  | 时钟\_CLK\_REF\_CTRL\_SRC\_BITS |
| 101 |  | ); |
| 102 |  | 同时（！（clock->selected&（1u <<src）） |
| 103 |  | return（）; |
| 104 |  | } |
| 105 |  |  |
| 106 |  | //使能时钟。在blog\_ref和blog\_sys上，这不做 |
| 107 |  | //所有其他时钟都在同一位置具有ENABLE位。 |
| 108 |  | hw\_set\_bits（clock-> clock，CLOCKS\_CLK\_GPOUT0\_CTRL\_ENABLE\_BITS）; |
| 109 |  |  |
| 110 |  | //既然源代码已经配置好了，我们可以相信用户提供的 |
| 111 |  | //divisor是一个安全值。 |
| 112 |  | clock->div =div; |
| 113 |  |  |
| 114 |  | //保存配置的频率 |
| 115 |  | configured\_freq[参数索引]=（uint32\_t）（uint64\_t）src\_freq8）/div）; |
| 116 |  |  |
| 117 |  | 返回true; |
| 118 | } |  |

它在clocks\_init中为每个时钟调用下面的示例显示了SQL\_SYS配置：

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_clocks/clocks.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/clocks.c#L162-L167)第162-167

162

163

164

165

166

167

// CLK时钟= PLL时钟（通常）125 MHz/1 =125 MHz

clock\_configure（clock\_sys，

CLOCKS\_CLK\_CLK\_CLK\_PLL\_AUX、CLOCKS\_CLK\_CLK\_CLK\_CTRL\_SRC\_VALUE\_CLKSRC\_PLL\_AUX、CLK\_CLK\_KHZ \* KHZ、

CLK\_CLK\_KHZ \*KHZ）;

一旦配置了时钟，就可以调用clock\_get\_hz来获取以Hz为单位的输出频率。

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_clocks/clocks.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/clocks.c#L201-L203)第201-203

1. uint32\_tclock\_get\_hz（ enumclock\_index）{
2. returnconfigured\_freq[int\_index];

203}

**警告**

假设程控仪提供的源频率是正确的。如果不是，则返回的频率

clock\_get\_hz将不准确。

* + - 1. 使用频率计数器

要使用频率计数器，程序员必须：

* + - * + 设置参考频率：\_ref
        + 设置他们想要测量的源的多路复用器位置参见[FC0\_SRC](#_bookmark359)
        + 等待设置[FC0\_STATUS](#_bookmark360)中的DONE状态位
        + 读取结果

SDK定义了一个frequency\_count函数，该函数以源作为参数，并返回以kHz为单位的频率

SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_clocks/clocks.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/clocks.c#L211-L238)第211-238

|  |  |  |
| --- | --- | --- |
| 211uint32\_t frequency\_count\_count（uint src）{ | | |
| 212 |  | fc\_hw\_t \*fc = clocks\_hw-> fc 0; |
| 213 |  |  |
| 214 |  | //如果频率计数器正在运行，则需要等待它。 即使源为NULL，它也会运行 |
| 215 |  | while（fc->statusCLOCKS\_FC0\_STATUS\_RUNNING\_BITS）{ |
| 216 |  | return（）; |
| 217 |  | } |
| 218 |  |  |
| 219 |  | //设置参考频率 |
| 220 |  | fc->ref\_ref = clock\_get\_hz（时钟\_ref）/1000; |
| 221 |  |  |
| 222 |  | //FIXME：不要选择随机间隔。使用最佳间隔 |
| 223 |  | int f1 =10; |
| 224 |  |  |
| 225 |  | //没有最小值或最大值 |
| 226 |  | int max =0; |
| 227 |  | fC->max\_fC =0xffffffff; |
| 228 |  |  |
| 229 |  | //设置SRC，自动开始测量 |
| 230 |  | fc->src =src; |
| 231 |  |  |
| 232 |  | 同时（！(fc->status&CLOCKS\_FC0\_STATUS\_DONE\_BITS））{ |
| 233 |  | return（）; |
| 234 |  | } |
| 235 |  |  |
| 236 |  | //返回结果 |
| 237 |  | returnfc->result>>CLOCKS\_FC0\_KHZ\_LSB; |
| 238 | } |  |

还有一个包装器函数可以将单位改为MHz`：

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#L212-L214)第212-214

1. public int findDuplicate（int findDuplicate）{
2. return（（ float）（frequency\_count\_count（src）/KHZ;

214}

##### 注意

频率计数器也可用于测试模式。这允许硬件检查频率是否在[FC0\_MIN\_KHZ](#_bookmark355)和[FC0\_MAX\_KHZ](#_bookmark356)中设置的最小频率和最大频率范围内。 在此模式下，如果频率在指定范围内，则当设置DONE时，将设置FC0\_STATUS中的PASS位。否则，将设置FAST或SLOW

如果编程器尝试计数停止的时钟，或时钟停止运行，则DIED位将被设置。如果任何

设置DIED、FAST或SLOW，然后设置FAIL

* + - 1. 配置GPIO输出时钟

*SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_clocks/clocks.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/clocks.c#L310-L330)第310-330*

310void clock\_gpio\_init\_int\_frac（uint gpio，uint src，uint32\_tdiv\_int，uint8\_tdiv\_frac）{

|  |  |  |
| --- | --- | --- |
| 311 |  | //有点乱，但它的代码一样多，通过一个查找循环 |
| 312 |  | //tables.每个gpout生成器的源都是相同的 |
| 313 |  | //所以只需要调用来自GP 0的源 |
| 314 |  | int maximum =0; |
| 315 |  | 如果 （gpio ==21）gpint =clk\_gpout0; |
| 316 |  | else if（gpio ==23）gpint =clk\_gpout1; |
| 317 |  | else if（gpio ==24）gpint =clk\_gpout2; |
| 318 |  | else if（gpio ==25）gpint =clk\_gpout3; |
| 319 |  | 否则{ |
| 320 |  | invalid\_params\_if（CLOCKS，true）; |
| 321 |  | } |
| 322 |  |  |
| 323 |  | //设置gpython生成器 |
| 324 |  | clocks\_hw-> clock [gpclock]. clock=（src <<CLOCKS\_CLK\_GPOUT0\_CTRL\_AUXSRC\_LSB）| |
| 325 |  | CLOCKS\_CLK\_GPOUT0\_CTRL\_ENABLE\_BITS; |
| 326 |  | clocks\_hw-> clocks [gpclock].div=（div\_int <<CLOCKS\_CLK\_GPOUT0\_DIV\_INT\_LSB）|div\_frac; |
| 327 |  |  |
| 328 |  | //将gpio引脚设置为gpclock函数 |
| 329 |  | gpio\_set\_function（gpio，GPIO\_FLOW\_GPCK）; |
| 330 | } |  |

* + - 1. 配置GPIO输入时钟

*SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_clocks/clocks.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/clocks.c#L357-L383)第357-383*

357 bool clock\_configure\_gpin（enumclock\_index，uint gpio，uint32\_tsrc\_freq，uint32\_tfreq）{

|  |  |  |
| --- | --- | --- |
| 358 |  | //配置时钟从GPIO输入运行 |
| 359 |  | int gpin =0; |
| 360 |  | 如果 （gpio ==20）gpin =0; |
| 361 |  | if（gpio ==22）gpin =1; |
| 362 |  | 否则{ |
| 363 |  | invalid\_params\_if（CLOCKS，true）; |
| 364 |  | } |
| 365 |  |  |
| 366 |  | //找出源代码。GPIN始终为auxsrc |
| 367 |  | return0; |
| 368 |  |  |
| 369 |  | //GPIN1 == GPIN0 +1 |
| 370 |  | uint auxsrc = gpin0\_src[index]+gpin; |
| 371 |  |  |
| 372 |  | if（has\_glitchless\_mux（has\_index））{ |
| 373 |  | //AUXsrc始终为1 |
| 374 |  | return1; |
| 375 |  | } |
| 376 |  |  |
| 377 |  | //设置GPIO函数 |
| 378 |  | gpio\_set\_function（gpio，GPIO\_FLOW\_GPCK）; |
| 379 |  |  |
| 380 |  | //现在我们有了src、auxsrc，并配置了gpio输入 |
| 381 |  | //调用clock configure从gpio运行时钟 |
| 382 |  | 返回clock\_configure（src，src，auxsrc，src\_freq，freq）; |
| 383 | } |  |

* + - 1. 使能结果

*SDK：[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_clocks/clocks.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_clocks/clocks.c#L286-L308)第286-308*

286void clocks\_enable\_resus（resus\_callback\_t resus\_callback）{

|  |  |  |
| --- | --- | --- |
| 287 |  | //如果通过强制停止，则重新启动 |
| 288 |  | //返回到默认的source\_ref。如果req\_ref停止运行， |
| 289 |  | //不工作 |
| 290 |  |  |
| 291 |  | //保存用户的resus回调 |
| 292 |  | \_resus\_callback =resus\_callback; |
| 293 |  |  |
| 294 |  | irq\_set\_exclusive\_handler（CLOCKS\_IRQ，clocks\_irq\_handler）; |
| 295 |  |  |
| 296 |  | //启用时钟中的resus中断 |
| 297 |  | clocks\_hw->inte =CLOCKS\_INTE\_CLK\_CLK\_RESUS\_BITS; |
| 298 |  |  |
| 299 |  | //使能时钟irq |
| 300 |  | irq\_set\_enabled（CLOCKS\_IRQ，true）; |
| 301 |  |  |
| 302 |  | //2\*系统参考频率/系统最小频率; |
| 303 |  | //假设\_ref是3 MHz，我们希望\_sys不低于1 MHz |
| 304 |  | int count =2\*1; |
| 305 |  |  |
| 306 |  | //使用最大超时启用resus |
| 307 |  | clocks\_hw->resus. cycle= CLOCKS\_CLK\_CLK\_RESUS\_CTRL\_ENABLE\_BITS|超时; |
| 308 | } |  |

* + - 1. 配置睡眠模式

当处理器内核或DMA都不请求时钟时，休眠模式处于活动状态例如，DMA未激活，*核心0*和*核心1*都在等待中断。SLEEP\_EN寄存器设置休眠模式下运行的时钟hello\_sleep示例（<https://github.com/raspberrypi/pico-playground/blob/master/sleep/hello_sleep/hello_sleep.c>）说明了如何使芯片进入睡眠状态，直到RTC触发。这种情况下，SLEEP\_EN0寄存器中仅使能RTC时钟

##### 注意

在休眠模式下，由于某些逻辑需要计时，处理器才能再次唤醒，因此总是将logic\_sys发送到proc0和proc1。

Pico Extras：[https://github.com/raspberrypi/pico-extras/blob/master/src/rp2\_common/pico\_sleep/sleep.c](https://github.com/raspberrypi/pico-extras/blob/master/src/rp2_common/pico_sleep/sleep.c#L106-L122)Lines 106-122

106void sleep\_goto\_sleep\_until（datetime\_t \*t，rtc\_callback\_t callback）{

107

108

109

110

111

112

113

114

115

116

117

118

119

120

//我们应该已经调用了sleep\_run\_from\_dormant\_source函数

assert（dermal\_source\_valid（\_dermal\_source））;

*//在睡眠模式下*关闭*所有时钟，除了RTC*

clocks\_hw->sleep\_en0 = CLOCKS\_SLEEP\_EN0\_CLK\_RTC\_BITS; clocks\_hw->sleep\_en1 =0x 0;

return（t）;

uint save = scb\_hw->scr;

//在进程中启用深度睡眠

scb\_hw->scr=保存|M0PLUS\_SCR\_SLEEPDEEP\_BITS;

//继续睡觉

121 wfi（）;

122}

* + 1. 登记册一览表

时钟寄存器从基址0x40008000（在SDK中定义为[CLOCKS\_BASE](#_bookmark39)）开始。

*表207.对妇女的歧视 时钟寄存器列表*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [CLK\_GPOUT0\_CTRL](#_bookmark323) | 时钟控制，可即时更改（auxsrc除外） |
| 0x04 | [CLK\_GPOUT0\_DIV](#_bookmark324) | 时钟除数，可即时更改 |
| 0x08 | [CLK\_GPOUT0\_SELECTED](#_bookmark325) | 指示无毛刺多路复用器当前选择的SRC（独热）。 |
| 0x0C | [CLK\_GPOUT1\_CTRL](#_bookmark326) | 时钟控制，可即时更改（auxsrc除外） |
| 0x10 | [CLK\_GPOUT1\_DIV](#_bookmark327) | 时钟除数，可即时更改 |
| 0x14 | [CLK\_GPOUT1\_SELECTED](#_bookmark328) | 指示无毛刺多路复用器当前选择的SRC（独热）。 |
| 0x18 | [CLK\_GPOUT2\_CTRL](#_bookmark329) | 时钟控制，可即时更改（auxsrc除外） |
| 0x1c | [CLK\_GPOUT2\_DIV](#_bookmark330) | 时钟除数，可即时更改 |
| 0x20 | [CLK\_GPOUT2\_SELECTED](#_bookmark331) | 指示无毛刺多路复用器当前选择的SRC（独热）。 |
| 0x24 | [CLK\_GPOUT3\_CTRL](#_bookmark332) | 时钟控制，可即时更改（auxsrc除外） |
| 0x28 | [CLK\_GPOUT3\_DIV](#_bookmark333) | 时钟除数，可即时更改 |
| 0x2c | [CLK\_GPOUT3\_SELECTED](#_bookmark334) | 指示无毛刺多路复用器当前选择的SRC（独热）。 |
| 0x30 | [CLK\_REF\_CTRL](#_bookmark335) | 时钟控制，可即时更改（auxsrc除外） |
| 0x34 | [CLK\_REF\_DIV](#_bookmark336) | 时钟除数，可即时更改 |
| 0x38 | [CLK\_REF\_SELECT](#_bookmark337) | 指示无毛刺多路复用器当前选择的SRC（独热）。 |
| 0x3c | [CLK\_CLK\_CTRL](#_bookmark338) | 时钟控制，可即时更改（auxsrc除外） |
| 0x40 | [CLK\_CLK\_DIV](#_bookmark339) | 时钟除数，可即时更改 |
| 0x44 | [CLK\_CLK\_SELECTED](#_bookmark340) | 指示无毛刺多路复用器当前选择的SRC（独热）。 |
| 0x48 | [CLK\_PERI\_CTRL](#_bookmark341) | 时钟控制，可即时更改（auxsrc除外） |
| 0x50 | [CLK\_PERI\_SELECTED](#_bookmark342) | 指示无毛刺多路复用器当前选择的SRC（独热）。 |
| 0x54 | [CLK\_USB\_CTRL](#_bookmark343) | 时钟控制，可即时更改（auxsrc除外） |
| 0x58 | [CLK\_USB\_DIV](#_bookmark344) | 时钟除数，可即时更改 |
| 0x5c | [CLK\_USB\_SELECTED](#_bookmark345) | 指示无毛刺多路复用器当前选择的SRC（独热）。 |
| 0x60 | [CLK\_ADC\_CTRL](#_bookmark346) | 时钟控制，可即时更改（auxsrc除外） |
| 0x64 | [CLK\_ADC\_DIV](#_bookmark347) | 时钟除数，可即时更改 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x68 | [CLK\_ADC\_SELECT](#_bookmark348) | 指示无毛刺多路复用器当前选择的SRC（独热）。 |
| 0x6c | [CLK\_RTC\_CTRL](#_bookmark349) | 时钟控制，可即时更改（auxsrc除外） |
| 0x70 | [CLK\_RTC\_DIV](#_bookmark350) | 时钟除数，可即时更改 |
| 0x74 | [CLK\_RTC\_SELECTED](#_bookmark351) | 指示无毛刺多路复用器当前选择的SRC（独热）。 |
| 0x78 | [CLK\_CLK\_RESUS\_CTRL](#_bookmark352) |  |
| 0x7c | [CLK\_RESUS\_STATUS](#_bookmark353) |  |
| 0x80 | [FC0\_REF\_KHZ](#_bookmark354) | 参考时钟频率（kHz） |
| 0x84 | [FC0\_MIN\_KHZ](#_bookmark355) | 最小通过频率（kHz）。这是可选的。如果未使用通过/失败标志，则设置为0 |
| 0x88 | [FC0\_MAX\_KHZ](#_bookmark356) | 最大通过频率（单位：kHz）。这是可选的。如果未使用通过/失败标志，则设置为 |
| 0x8c | [FC0\_DELAY](#_bookmark357) | 延迟频率计数的开始，以允许多路复用器建立延迟以参考时钟周期的倍数测量 |
| 0x90 | [FC0\_INTERVAL](#_bookmark358) | 测试间隔为0.98us \* 2\*\*interval，但我们称之为1us \* 2\*\*interval  默认情况下，测试间隔为250 us |
| 0x94 | [FC0\_SRC](#_bookmark359) | 时钟发送到频率计数器，不需要时设置为0写入此寄存器启动频率计数 |
| 0x98 | [FC0\_STATUS](#_bookmark360) | 频率计数器状态 |
| 0x9c | [FC0\_结果](#_bookmark361) | 频率测量结果，仅在status\_done=1时有效 |
| 0xa0 | [唤醒\_EN0](#_bookmark362) | 使能唤醒模式时钟 |
| 0xa4 | 唤醒\_EN1 | 使能唤醒模式时钟 |
| 0xa8 | 睡眠\_EN0 | 使能睡眠模式时钟 |
| 0xac | 睡眠\_EN1 | 使能睡眠模式时钟 |
| 0xb0 | 已启用0 | 指示时钟使能状态 |
| 0xb4 | 启用1 | 指示时钟使能状态 |
| 0xb8 | INTR | 原始中断 |
| 0xbc | INTE | 中断使能 |
| 0xc0 | INTF | 中断强制 |
| 0xc4 | INTs | 强制屏蔽后的屏蔽状态 |

### [时钟](#_bookmark322)：CLK\_GPOUT0\_CTRL寄存器

**偏移**：0x00

描述

时钟控制，可即时更改（auxsrc除外）

*表208.对妇女的歧视CLK\_GPOUT0\_CTRL*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:21 | Reserved. | - | - | - |
| 20 | 轻推 | 此信号的边沿使输出的相位偏移输入时钟的1个周期  这可以在任何时候进行 | RW | 0x0 |
| 19:18 | Reserved. | - | - | - |
| 17:16 | 相 | 这将使使能信号延迟最多3个输入时钟  这必须在时钟启用之前设置才能产生任何效果 | RW | 0x0 |
| 15:30 | Reserved. | - | - | - |
| 12 | DC50 | 使能奇数除数的占空比校正 | RW | 0x0 |
| 11 | 使 | 干净地启动和停止时钟发生器 | RW | 0x0 |
| 10 | 杀 | 异步杀死时钟发生器 | RW | 0x0 |
| 9 | Reserved. | - | - | - |
| 8:5 | AUXSRC | 切换辅助时钟源，切换时会毛刺  0x0→ clksrc\_clksrc\_sys 0x1→ clksrc\_gpin0 0x2→ clksrc\_gpin1 0x3→ clksrc\_usb 0x4→ rosc\_clksrc 0x5→ xosc\_clksrc 0x6→ clksrc\_sys  0x7→ clk\_usb 0x8→ clk\_adc 0x9→ clk\_rtc 0xa→ clk\_ref | RW | 0x0 |
| 4:0 | Reserved. | - | - | - |

寄存器

表209.对妇女的歧视CLK\_GPOUT0\_DIV

寄存器

### [时钟](#_bookmark322)：CLK\_GPOUT0\_DIV寄存器

**偏移**：0x04

描述

时钟除数，可即时更改

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31：8 | INT | 除数的0分量，0→除以2^16 | RW | 0x000001 |
| 7:0 | FRAC | 除数的分数分量 | RW | 0x00 |

### [时钟](#_bookmark322)：CLK\_GPOUT0\_SELECTED寄存器

**偏移**：0x08

描述

指示无毛刺多路复用器当前选择的SRC（独热）。

*表210.对妇女的歧视CLK\_GPOUT0\_SELECT*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 该片没有无毛刺多路复用器（只有AUX\_SRC字段，没有SRC），因此该寄存器硬连线至0x1。 | RO | 0x00000001 |

ED寄存器

表211.对妇女的歧视CLK\_GPOUT1\_CTRL

寄存器

### [时钟](#_bookmark322)：CLK\_GPOUT1\_CTRL寄存器

**偏移**：0x0c

描述

时钟控制，可即时更改（auxsrc除外）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:21 | Reserved. | - | - | - |
| 20 | 轻推 | 此信号的边沿使输出的相位偏移输入时钟的1个周期  这可以在任何时候进行 | RW | 0x0 |
| 19:18 | Reserved. | - | - | - |
| 17:16 | 相 | 这将使使能信号延迟最多3个输入时钟  这必须在时钟启用之前设置才能产生任何效果 | RW | 0x0 |
| 15:13 | Reserved. | - | - | - |
| 12 | DC50 | 使能奇数除数的占空比校正 | RW | 0x0 |
| 11 | 使 | 干净地启动和停止时钟发生器 | RW | 0x0 |
| 10 | 杀 | 异步杀死时钟发生器 | RW | 0x0 |
| 9 | Reserved. | - | - | - |
| 8:5 | AUXSRC | 切换辅助时钟源，切换时会毛刺  0x0→ clksrc\_clksrc\_sys 0x1→ clksrc\_gpin0 0x2→ clksrc\_gpin1 0x3→ clksrc\_usb 0x4→ rosc\_clksrc 0x5→ xosc\_clksrc 0x6→ clksrc\_sys  0x7→ USB\_USB 0x8→ USB\_USB 0x9→ USB\_RTC 0xa→ USB\_ref | RW | 0x0 |
| 4:0 | Reserved. | - | - | - |

### [时钟](#_bookmark322)：CLK\_GPOUT1\_DIV寄存器

**偏移**：0x10

描述

时钟除数，可即时更改

*表212.对妇女的歧视CLK\_GPOUT1\_DIV*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | INT | 除数的0分量，0→除以2^16 | RW | 0x000001 |
| 7:0 | FRAC | 除数的分数分量 | RW | 0x00 |

寄存器

表213.对妇女的歧视CLK\_GPOUT1\_SELECT

ED寄存器

表214.对妇女的歧视CLK\_GPOUT2\_CTRL

寄存器

### [时钟](#_bookmark322)：CLK\_GPOUT1\_SELECTED寄存器

**偏移**：0x14

描述

指示无毛刺多路复用器当前选择的SRC（独热）。

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 该片没有无毛刺多路复用器（只有AUX\_SRC字段，没有SRC），因此该寄存器硬连线至0x1。 | RO | 0x00000001 |

### [时钟](#_bookmark322)：CLK\_GPOUT2\_CTRL寄存器

**偏移**：0x18

描述

时钟控制，可即时更改（auxsrc除外）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:21 | Reserved. | - | - | - |
| 20 | 轻推 | 此信号的边沿使输出的相位偏移输入时钟的1个周期  这可以在任何时候进行 | RW | 0x0 |
| 19:18 | Reserved. | - | - | - |
| 17:16 | 相 | 这会将使能信号延迟最多3个输入时钟  这必须在时钟启用之前设置才能产生任何效果 | RW | 0x0 |
| 15:13 | Reserved. | - | - | - |
| 12 | DC50 | 使能奇数除数的占空比校正 | RW | 0x0 |
| 11 | 使 | 干净地启动和停止时钟发生器 | RW | 0x0 |
| 10 | 杀 | 异步杀死时钟发生器 | RW | 0x0 |
| 9 | Reserved. | - | - | - |
| 8:5 | AUXSRC | 切换辅助时钟源，切换时会毛刺  0x0→ clksrc\_php\_sys 0x1→ clksrc\_gpin0 0x2→ clksrc\_gpin1 0x3→ clksrc\_php\_usb 0x4→ rosc\_clksrc\_ph 0x5→ xosc\_clksrc 0x6→ php\_sys  0x7→ clk\_usb 0x8→ clk\_adc 0x9→ clk\_rtc 0xa→ clk\_ref | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 4:0 | Reserved. | - | - | - |

表215.对妇女的歧视CLK\_GPOUT2\_DIV

寄存器

表216.对妇女的歧视CLK\_GPOUT2\_SELECT

ED寄存器

表217.对妇女的歧视CLK\_GPOUT3\_CTRL

寄存器

### [时钟](#_bookmark322)：CLK\_GPOUT2\_DIV寄存器

**偏移量**：0x1c

描述

时钟除数，可即时更改

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31：8 | INT | 除数的0分量，0→除以2^16 | RW | 0x000001 |
| 7:0 | FRAC | 除数的分数分量 | RW | 0x00 |

### [时钟](#_bookmark322)：CLK\_GPOUT2\_SELECTED寄存器

**偏移**：0x20

描述

指示无毛刺多路复用器当前选择的SRC（独热）。

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 该片没有无毛刺多路复用器（只有AUX\_SRC字段，没有SRC），因此该寄存器硬连线至0x1。 | RO | 0x00000001 |

### [时钟](#_bookmark322)：CLK\_GPOUT3\_CTRL寄存器

**偏移**：0x24

描述

时钟控制，可即时更改（auxsrc除外）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:21 | Reserved. | - | - | - |
| 20 | 轻推 | 此信号的边沿使输出的相位偏移输入时钟的1个周期  这可以在任何时候进行 | RW | 0x0 |
| 19:18 | Reserved. | - | - | - |
| 17:16 | 相 | 这将使使能信号延迟最多3个输入时钟  这必须在时钟启用之前设置才能产生任何效果 | RW | 0x0 |
| 15:13 | Reserved. | - | - | - |
| 12 | DC50 | 使能奇数除数的占空比校正 | RW | 0x0 |
| 11 | 使 | 干净地启动和停止时钟发生器 | RW | 0x0 |
| 10 | 杀 | 异步杀死时钟发生器 | RW | 0x0 |
| 9 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 八点五 | AUXSRC | 切换辅助时钟源，切换时会毛刺  0x0→ clksrc\_php\_sys 0x1→ clksrc\_gpin0 0x2→ clksrc\_gpin1 0x3→ clksrc\_php\_usb 0x4→ rosc\_clksrc\_ph 0x5→ xosc\_clksrc 0x6→ php\_sys  0x7→ clk\_usb 0x8→ clk\_adc 0x9→ clk\_rtc 0xa→ clk\_ref | RW | 0x0 |
| 4:0 | Reserved. | - | - | - |

表218.CLK\_GPOUT3\_DIV

寄存器

表219.对妇女的歧视CLK\_GPOUT3\_SELECT

ED寄存器

表220.对妇女的歧视CLK\_REF\_CTRL

寄存器

### [时钟](#_bookmark322)：CLK\_GPOUT3\_DIV寄存器

**偏移**：0x28

描述

时钟除数，可即时更改

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31：8 | INT | 除数的0分量，0→除以2^16 | RW | 0x000001 |
| 7:0 | FRAC | 除数的分数分量 | RW | 0x00 |

### [时钟](#_bookmark322)：CLK\_GPOUT3\_SELECTED寄存器

**偏移**：0x2c

描述

指示无毛刺多路复用器当前选择的SRC（独热）。

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 该片没有无毛刺多路复用器（只有AUX\_SRC字段，没有SRC），因此该寄存器硬连线至0x1。 | RO | 0x00000001 |

### [时钟](#_bookmark322)：CLK\_REF\_CTRL寄存器

**偏移**：0x30

描述

时钟控制，可即时更改（auxsrc除外）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:七 | Reserved. | - | - | - |
| 六分之5 | AUXSRC | 切换辅助时钟源，切换时会毛刺  0x0→ clksrc\_gpin\_usb 0x1→ clksrc\_gpin0 0x2→ clksrc\_gpin1 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 4:二 | Reserved. | - | - | - |
| 1：0 | SRC | 时钟源毛刺，可以在运行中更改  0x0→ rosc\_clksrc\_ph0x1→clksrc\_ref\_aux 0x2→ xosc\_clksrc | RW | - |

表221.对妇女的歧视CLK\_REF\_DIV寄存器

表222.对妇女的歧视CLK\_REF\_SELECT

寄存器

表223.对妇女的歧视CLK\_CLK\_CTRL

寄存器

### [时钟](#_bookmark322)：CLK\_REF\_DIV寄存器

**偏移**：0x34

描述

时钟除数，可即时更改

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:10 | Reserved. | - | - | - |
| 9:8 | INT | 除数的0分量，0→除以2^16 | RW | 0x1 |
| 7:0 | Reserved. | - | - | - |

### [时钟](#_bookmark322)：CLK\_REF\_SELECTED寄存器

**偏移**：0x38

描述

指示无毛刺多路复用器当前选择的SRC（独热）。

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 无毛刺多路复用器不会瞬时切换（以避免毛刺），因此软件应轮询此寄存器以等待切换完成。该寄存器包含CTRL SRC字段中列举的每个时钟源的一个解码位。任何时候，这些位中最多有一位被设置，表示时钟当前存在于无毛刺多路复用器的输出端在切换过程中，该寄存器可能会短暂显示全0。 | RO | 0x00000001 |

### [时钟](#_bookmark322)：CLK\_CLK\_CTRL寄存器

**偏移**：0x3c

描述

时钟控制，可即时更改（auxsrc除外）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31：8 | Reserved. | - | - | - |
| 七点5 | AUXSRC | 切换辅助时钟源，切换时会毛刺  0x0→ clksrc\_plug\_sys 0x1→ clksrc\_plug\_usb 0x2→ rosc\_clksrc 0x3→ xosc\_clksrc 0x4→ clksrc\_gpin0 0x5→ clksrc\_gpin1 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 4:一 | Reserved. | - | - | - |
| 0 | SRC | 时钟源毛刺，可以在运行中更改  0x0→参考  0x1→clksrc\_blog\_sys\_aux | RW | 0x0 |

表224.对妇女的歧视CLK\_CLK\_DIV寄存器

表225.对妇女的歧视CLK\_CLK\_SELECTED

寄存器

表226.对妇女的歧视CLK\_PERI\_CTRL

寄存器

### [时钟](#_bookmark322)：CLK\_CLK\_DIV寄存器

**偏移**：0x40

描述

时钟除数，可即时更改

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31：8 | INT | 除数的0分量，0→除以2^16 | RW | 0x000001 |
| 7:0 | FRAC | 除数的分数分量 | RW | 0x00 |

### [时钟](#_bookmark322)：CLK\_CLK\_SELECT寄存器

**偏移**：0x44

描述

指示无毛刺多路复用器当前选择的SRC（独热）。

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 无毛刺多路复用器不会瞬时切换（以避免毛刺），因此软件应轮询此寄存器以等待切换完成。该寄存器包含CTRL SRC字段中列举的每个时钟源的一个解码位。任何时候，这些位中最多有一位被设置，表示时钟当前存在于无毛刺多路复用器的输出端在切换过程中，该寄存器可能会短暂显示全0。 | RO | 0x00000001 |

### [时钟](#_bookmark322)：CLK\_PERI\_CTRL寄存器

**偏移**：0x48

描述

时钟控制，可即时更改（auxsrc除外）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - | - |
| 11 | 使 | 干净地启动和停止时钟发生器 | RW | 0x0 |
| 10 | 杀 | 异步杀死时钟发生器 | RW | 0x0 |
| 9:8 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 七点五 | AUXSRC | 切换辅助时钟源，切换时会毛刺  0x 0→clk\_sys  0x1→ clksrc\_gpin\_sys 0x2→ clksrc\_gpin\_usb 0x3→ rosc\_clksrc\_ph 0x4→ xosc\_clksrc 0x5→ clksrc\_gpin 0 0x6→ clksrc\_gpin 1 | RW | 0x0 |
| 4:0 | Reserved. | - | - | - |

表227.对妇女的歧视时钟\_周边\_选定

寄存器

表228.对妇女的歧视CLK\_USB\_CTRL

寄存器

### [时钟](#_bookmark322)：CLK\_PERI\_SELECT寄存器

**偏移**：0x50

描述

指示无毛刺多路复用器当前选择的SRC（独热）。

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 该片没有无毛刺多路复用器（只有AUX\_SRC字段，没有SRC），因此该寄存器硬连线至0x1。 | RO | 0x00000001 |

### [时钟](#_bookmark322)：CLK\_USB\_CTRL寄存器

**偏移**：0x54

描述

时钟控制，可即时更改（auxsrc除外）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:21 | Reserved. | - | - | - |
| 20 | 轻推 | 此信号的边沿使输出的相位偏移输入时钟的1个周期  这可以在任何时候进行 | RW | 0x0 |
| 19:18 | Reserved. | - | - | - |
| 17:16 | 相 | 这会将使能信号延迟最多3个输入时钟  这必须在时钟启用之前设置才能产生任何效果 | RW | 0x0 |
| 15:12 | Reserved. | - | - | - |
| 11 | 使 | 干净地启动和停止时钟发生器 | RW | 0x0 |
| 10 | 杀 | 异步杀死时钟发生器 | RW | 0x0 |
| 9:8 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 七点五 | AUXSRC | 切换辅助时钟源，切换时会毛刺  0x0→ clksrc\_usb\_usb 0x1→ clksrc\_usb\_sys 0x2→ rosc\_clksrc\_ph 0x3→ xosc\_clksrc 0x4→ clksrc\_gpin0 0x5→ clksrc\_gpin1 | RW | 0x0 |
| 4:0 | Reserved. | - | - | - |

表229.对妇女的歧视CLK\_USB\_DIV寄存器

表230.第230页CLK\_USB\_SELECTED

寄存器

表231.对妇女的歧视CLK\_ADC\_CTRL

寄存器

### [时钟](#_bookmark322)：CLK\_USB\_DIV寄存器

**偏移**：0x58

描述

时钟除数，可即时更改

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:10 | Reserved. | - | - | - |
| 9:8 | INT | 除数的0分量，0→除以2^16 | RW | 0x1 |
| 7:0 | Reserved. | - | - | - |

### [时钟](#_bookmark322)：CLK\_USB\_SELECTED寄存器

**偏移**：0x5c

描述

指示无毛刺多路复用器当前选择的SRC（独热）。

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 该片没有无毛刺多路复用器（只有AUX\_SRC字段，没有SRC），因此该寄存器硬连线至0x1。 | RO | 0x00000001 |

### [时钟](#_bookmark322)：CLK\_ADC\_CTRL寄存器

**偏移**：0x60

描述

时钟控制，可即时更改（auxsrc除外）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:21 | Reserved. | - | - | - |
| 20 | 轻推 | 此信号的边沿使输出的相位偏移输入时钟的1个周期  这可以在任何时候进行 | RW | 0x0 |
| 19:18 | Reserved. | - | - | - |
| 17:16 | 相 | 这会将使能信号延迟最多3个输入时钟  这必须在时钟启用之前设置才能产生任何效果 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 15:12 | Reserved. | - | - | - |
| 11 | 使 | 干净地启动和停止时钟发生器 | RW | 0x0 |
| 10 | 杀 | 异步杀死时钟发生器 | RW | 0x0 |
| 9:8 | Reserved. | - | - | - |
| 七点五 | AUXSRC | 切换辅助时钟源，切换时会毛刺  0x0→ clksrc\_usb\_usb 0x1→ clksrc\_usb\_sys 0x2→ rosc\_clksrc\_ph 0x3→ xosc\_clksrc 0x4→ clksrc\_gpin0 0x5→ clksrc\_gpin1 | RW | 0x0 |
| 4:0 | Reserved. | - | - | - |

表232.对妇女的歧视CLK\_ADC\_DIV寄存器

表233.对妇女的歧视CLK\_ADC\_SELECT

寄存器

表234.对妇女的歧视CLK\_RTC\_CTRL

寄存器

### [时钟](#_bookmark322)：CLK\_ADC\_DIV寄存器

**偏移**：0x64

描述

时钟除数，可即时更改

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:10 | Reserved. | - | - | - |
| 9:8 | INT | 除数的0分量，0→除以2^16 | RW | 0x1 |
| 7:0 | Reserved. | - | - | - |

### [时钟](#_bookmark322)：CLK\_ADC\_SELECT寄存器

**偏移**：0x68

描述

指示无毛刺多路复用器当前选择的SRC（独热）。

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 该片没有无毛刺多路复用器（只有AUX\_SRC字段，没有SRC），因此该寄存器硬连线至0x1。 | RO | 0x00000001 |

### [时钟](#_bookmark322)：CLK\_RTC\_CTRL寄存器

**偏移**：0x6c

描述

时钟控制，可即时更改（auxsrc除外）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:21 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 20 | 轻推 | 此信号的边沿使输出的相位偏移输入时钟的1个周期  这可以在任何时候进行 | RW | 0x0 |
| 19:十8 | Reserved. | - | - | - |
| 十七点16 | 相 | 这将使使能信号延迟最多3个输入时钟  这必须在时钟启用之前设置才能产生任何效果 | RW | 0x0 |
| 15:12 | Reserved. | - | - | - |
| 11 | 使 | 干净地启动和停止时钟发生器 | RW | 0x0 |
| 10 | 杀 | 异步杀死时钟发生器 | RW | 0x0 |
| 9:8 | Reserved. | - | - | - |
| 七点五 | AUXSRC | 切换辅助时钟源，切换时会毛刺  0x0→ clksrc\_usb\_usb 0x1→ clksrc\_usb\_sys 0x2→ rosc\_clksrc\_ph 0x3→ xosc\_clksrc 0x4→ clksrc\_gpin0 0x5→ clksrc\_gpin1 | RW | 0x0 |
| 4:0 | Reserved. | - | - | - |

表235.对妇女的歧视CLK\_RTC\_DIV寄存器

表236.对妇女的歧视CLK\_RTC\_SELECTED

寄存器

### [时钟](#_bookmark322)：CLK\_RTC\_DIV寄存器

**偏移**：0x70

描述

时钟除数，可即时更改

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31：8 | INT | 除数的0分量，0→除以2^16 | RW | 0x000001 |
| 7:0 | FRAC | 除数的分数分量 | RW | 0x00 |

### [时钟](#_bookmark322)：CLK\_RTC\_SELECTED寄存器

**偏移量**：0x 74

描述

指示无毛刺多路复用器当前选择的SRC（独热）。

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 该片没有无毛刺多路复用器（只有AUX\_SRC字段，没有SRC），因此该寄存器硬连线至0x1。 | RO | 0x00000001 |

### [时钟](#_bookmark322)：CLK\_RESUS\_CTRL寄存器

**偏移**：0x78

*表237.对妇女的歧视CLK\_CLK\_RESUS\_CTR*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:17 | Reserved. | - | - | - |
| 16 | 明确 | 用于在纠正触发故障的故障后清除结果 | RW | 0x0 |
| 15:13 | Reserved. | - | - | - |
| 12 | FRCE | 强制执行结果，仅用于测试目的 | RW | 0x0 |
| 11:九分 | Reserved. | - | - | - |
| 8 | 使 | 启用结果 | RW | 0x0 |
| 七比零 | 超时 | 这表示为循环次数，并且必须>= 2x循环次数/min循环 | RW | 0xff |

L寄存器

表238.对妇女的歧视CLK\_CLK\_RESUS\_STA

TUS注册

表239.对妇女的歧视FC0\_REF\_KHZ寄存器

表240.第240页FC0\_MIN\_KHZ

寄存器

表241.对妇女的歧视FC0\_MAX\_KHZ

寄存器

### [时钟](#_bookmark322)：CLK\_RESUS\_STATUS寄存器

**偏移**：0x7c

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:一 | Reserved. | - | - | - |
| 0 | 恢复 | 时钟已恢复，纠正错误，然后发送cclock\_clear=1 | RO | 0x0 |

### [时钟](#_bookmark322)：FC0\_REF\_KHZ寄存器

**偏移**：0x80

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:20 | Reserved. | - | - |
| 十九比零 | 参考时钟频率（单位：kHz） | RW | 0x00000 |

### [时钟](#_bookmark322)：FC0\_MIN\_KHZ寄存器

**偏移**：0x84

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:二十五分 | Reserved. | - | - |
| 二十4:0 | 最小通过频率（kHz）。这是可选的。如果未使用通过/失败标志，则设置为0 | RW | 0x0000000 |

### [时钟](#_bookmark322)：FC0\_MAX\_KHZ寄存器

**偏移量**：0x88

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:二十五分 | Reserved. | - | - |
| 二十4:0 | 最大通过频率（单位：kHz）。这是可选的。如果未使用通过/失败标志，则设置为0x 1ffffff | RW | 0x1ffffff |

### [时钟](#_bookmark322)：FC0\_DELAY寄存器

**偏移**：0x8c

*表242.对妇女的歧视FC0\_DELAY*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:三 | Reserved. | - | - |
| 二比零 | 延迟频率计数的开始，以允许多路复用器建立延迟以参考时钟周期的倍数测量 | RW | 0x1 |

寄存器

表243.对妇女的歧视FC0\_INTERVAL

寄存器

表244.对妇女的歧视FC0\_SRC寄存器

表245.对妇女的歧视FC0\_STATUS寄存器

### [时钟](#_bookmark322)：FC0\_INTERVAL寄存器

**偏移**：0x90

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:四 | Reserved. | - | - |
| 三比零 | 测试间隔为0.98us\*2\*\*interval，但我们称之为1us\*2\*\*interval。默认情况下，测试间隔为250us | RW | 0x8 |

### [时钟](#_bookmark322)：FC0\_SRC寄存器

**偏移量**：0x 94

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31：8 | Reserved. | - | - |
| 7:0 | 时钟发送到频率计数器，不需要时设置为0写入此寄存器启动频率计数  0x00→空  0x01→ USB\_SYS\_CLKSRC\_Primary0x02→USB\_CLKSRC\_Primary 0x03→ rosc\_CLKSRC  0x04→ rosc\_clksrc\_ph 0x05→ xosc\_clksrc 0x06→ clksrc\_gpin0 0x07→ clksrc\_gpin1 0x08→ clksrc\_ref  0x09→ USB\_SYS0x0a→USB\_USB 0x0b→ USB\_USB 0x0c→ USB\_USB 0x0d→ USB\_RTC | RW | 0x00 |

### [时钟](#_bookmark322)：FC0\_STATUS寄存器

**偏移**：0x98

描述

频率计数器状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:29 | Reserved. | - | - | - |
| 28 | 死 | 测试期间测试时钟停止 | RO | 0x0 |
| 二17:二十5分 | Reserved. | - | - | - |
| 24 | 快速 | 测试时钟比预期快，仅在status\_done=1时有效 | RO | 0x0 |
| 23:21 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 20 | 慢 | 测试时钟比预期慢，仅在status\_done=1时有效 | RO | 0x0 |
| 19：17 | Reserved. | - | - | - |
| 16 | 失败 | 测试失败 | RO | 0x0 |
| 15:十三分 | Reserved. | - | - | - |
| 12 | 等待 | 等待测试时钟启动 | RO | 0x0 |
| 11:九分 | Reserved. | - | - | - |
| 8 | 运行 | 试运行 | RO | 0x0 |
| 七点五 | Reserved. | - | - | - |
| 4 | 做 | 测试完成 | RO | 0x0 |
| 3：1 | Reserved. | - | - | - |
| 0 | 通过 | 测试通过 | RO | 0x0 |

表246.对妇女的歧视FC0\_RESULT寄存器

表247.对妇女的歧视唤醒\_EN0

寄存器

### [时钟](#_bookmark322)：FC0\_RESULT寄存器

**偏移量**：0x9c

描述

频率测量结果，仅在status\_done=1时有效

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:半 | Reserved. | - | - | - |
| 二十九：五 | kHz |  | RO | 0x0000000 |
| 4:0 | FRAC |  | RO | 0x00 |

### [时钟](#_bookmark322)：WAKE\_EN0寄存器

**偏移量**：0xa0

描述

使能唤醒模式时钟

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |