4.2.3.3.字符框

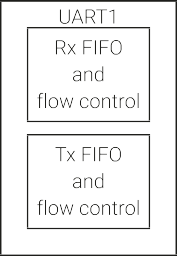
*图61. 字符帧。*



* + 1. 可编程硬件流控制

硬件流控制功能是完全可选的，使您能够通过使用nUARTRTS输出和nUARTCTS输入信号来控制串行数据流。[图62](#_bookmark0)显示了两个设备如何使用硬件流控制相互通信。

*图62. 两个类似设备之间的硬件流量控制。*



423号。用于启用和禁用硬件流控制的控制位

当RTS流控制使能时，nUARTRTS置位，直到接收FIFO填充到编程的水印电平。当CTS流控制使能时，发射机只能在nUARTCTS置位时发送数据。

可使用控制寄存器[UARTCR](#_bookmark12)中的RTSEn和CTSEn位选择硬件流控制。[表423](#_bookmark1)列出了必须如何设置这些位才能同时独立启用RTS和CTS流量控制

|  |  |  |
| --- | --- | --- |
| [UARTCR](#_bookmark12)寄存器位 | | |
| CTSEn | RTSEn | 描述 |
| 1 | 1 | RTS和CTS流量控制均已启用 |
| 1 | 0 | 仅启用CTS流控制 |
| 0 | 1 | 仅启用RTS流控制 |
| 0 | 0 | RTS和CTS流量控制均禁用 |

**注意**

当RTS流控制使能时，软件不能使用控制寄存器[UARTCR](#_bookmark12)中的RTSEn位来控制nUARTRTS的状态。

* + - 1. RTS流量控制

RTS流控制逻辑链接到可编程接收FIFO水印级别。当RTS流控制使能时，nUARTRTS置位，直到接收FIFO填满至水印级别。当达到接收FIFO水印水平时，nUARTRTS信号被解除断言，指示没有更多的空间来接收任何更多的数据。在当前字符传输完毕后，数据传输将停止

当数据已从接收FIFO中读出时，nUARTRTS信号被重新声明，以使其填充至低于水印水平。如果RTS流控制被禁用，但FIFO仍使能，则接收数据，直到接收FIFO已满，或不再向其发送数据

* + - 1. CTS流量控制

如果CTS流控制被使能，则发射器在发射下一个字节之前检查nUARTCTS信号如果nUARTCTS信号被断言，则它传输字节，否则不发生传输

nUARTCTS置位且发送FIFO不为空时，数据继续发送如果发送FIFO为空且nUARTCTS信号置位，则不发送数据

如果nUARTCTS信号被解除断言并且CTS流控制被使能，则当前字符传输在停止之前完成。如果CTS流控制被禁用，而FIFO被使能，则数据将继续传输，直到传输FIFO为空。

* + 1. DMA接口

该接口提供了一个连接到DMA控制器的接口DMA控制寄存器[UARTDMACR](#_bookmark18)控制DMA操作。DMA接口包括以下信号:

对于接收:

UARTRXDMAS

单字符DMA传输请求，由CPU断言对于接收，一个字符最多由12位组成当接收FIFO包含至少一个字符时，此信号置位

UARTRXDMABREQ

突发DMA传输请求，由CPU断言。当接收FIFO包含的字符数超过编程的水印级别时，该信号被置位。您可以使用FIFO电平选择寄存器[UARTIFLS](#_bookmark13)

UARTRXDMACLR

DMA请求清除，由DMA控制器置位以清除接收请求信号。如果请求DMA脉冲串传输，则在脉冲串中最后一个数据的传输期间，清除信号被置位

对于传输:

UARTTXDMAS

单字符DMA传输请求，由CPU断言。对于传输，一个字符最多由八位组成。当发送FIFO中至少有一个空位置时，该信号置位

UARTTXDMABREQ

突发DMA传输请求，由CPU断言。当发送FIFO包含的字符少于水印级别时，该信号被置位。您可以使用FIFO电平选择寄存器[UARTIFLS](#_bookmark13)为每个FIFO设置水印电平。

UARTTXDMACLR

DMA请求清除，由DMA控制器置位以清除发送请求信号。如果请求DMA脉冲串传输，则在脉冲串中最后一个数据的传输期间，清除信号被置位

突发传输和单个传输请求信号并不互斥，它们可以同时被断言例如，当在接收FIFO中存在比水印水平更多的数据时，断言突发传输请求当接收FIFO中剩余的数据量小于水印水平时，仅断言单个请求这对于流中剩余的待接收字符数小于突发的情况很有用

例如，如果必须接收19个字符并且水印级别被编程为4。的DMA

然后控制器传送四个字符的四个突发和三个单传送以完成流。

**注意**

对于剩余的三个字符，则无法断言猝发请求。

424号。发送和接收FIFO的 DMA触发点。

每个请求信号保持被断言，直到相关的DMACLR信号被断言。在请求清除信号被解除断言之后，请求信号可以再次变为有效，这取决于先前描述的条件。如果DMA控制寄存器UARTDMACR中的禁用了UART或清除了相关DMA使能位TXDMAE或RXDMAE，则所有请求信号都将被取消置位。

如果禁用FIFO，则它将以字符模式运行，并且只有DMA单传输模式可以运行，因为任何时候只能向FIFO传输一个字符或从FIFO传输一个字符。UARTRXDMASSTACK和UARTTXDMASSTACK是唯一可以被断言的请求信号。有关禁用FIFO的信息，请参见线路控制寄存器[UARTLCR\_H](#_bookmark11)

当EEPROM处于FIFO使能模式时，可以通过单次或突发传输进行数据传输，具体取决于编程的水印级别和FIFO中的数据量。[表424](#_bookmark2)列出了发送和接收FIFO的UARTRXDMABREQ和UARTTXDMABREQ的触发点（取决于水印级别）

|  |  |  |
| --- | --- | --- |
| 水印水平 | 突发长度 | |
| 传输（空位置数） | 接收（已填充位置的数量） |
| 1/8 | 28 | 4 |
| 1/4 | 24 | 8 |
| 1/2 | 16 | 16 |
| 3/4 | 8 | 24 |
| 7/8 | 4 | 28 |

此外，DMA控制寄存器[UARTDMACR](#_bookmark18)中的DMAONERR位支持使用接收错误中断UARTEINTR。它使能DMA接收请求输出UARTRXDMASBREQ或UARTRXDMABREQ，以便在置位错误中断UARTEINTR时屏蔽DMA接收请求输出保持非活动状态，直到UARTEINTR被清除。DMA传输请求输出不受

*图63. DMA传输波形。*

[图63](#_bookmark3)显示了单个传输请求和突发传输请求的时序图，以及相应的所有信号均与PCLK同步。为了清楚起见，假设在DMA控制器中没有请求信号的

* + 1. 中断

有11个可屏蔽中断产生的中断在RP 2040上，仅连接组合中断输出UARTINTR。

您可以通过更改[UARTISMC](#_bookmark14)掩码设置/清除寄存器中的掩码位来启用或禁用各个中断。将适当的屏蔽位设为高电平可使能中断。

提供单独的输出和组合的中断输出，使您能够使用全局中断服务例程或模块化设备驱动程序来处理中断。

发送和接收低电平中断UARTRXINTR和UARTTXINTR已从状态中分离

打断了我这使您能够使用UARTRXINTR和UARTTXINTR，以便根据FIFO触发电平读取或写入数据。

错误中断UARTEINTR可在数据接收出现错误时触发。可能出现许多错误情况

调制解调器状态中断UARTMSINTR是所有单个调制解调器状态信号的组合中断

各个中断源的状态可以从原始中断状态寄存器[UARTRIS](#_bookmark15)或屏蔽中断状态寄存器[UARTMIS](#_bookmark16)读取。

* + - 1. UARTMSINTR

如果任何调制解调器状态信号（nUARTCTS、nUARTDCD、nUARTDSR和nUARTRI）发生变化，则调制解调器状态中断被置位。根据产生中断的调制解调器状态信号，将1写入中断清除寄存器[UARTICR](#_bookmark17)

* + - 1. UARTRXINTR

当发生以下事件之一时，接收中断改变状态

* + - * + 如果FIFO使能且接收FIFO达到编程触发电平。当这种情况发生时，接收中断被置位为高电平。通过从接收FIFO读取数据来清除接收中断，直到

变得小于触发电平，或者通过清除中断。

* + - * + 如果FIFO被禁用（深度为一个位置）并且接收到数据从而填充该位置，则接收中断被置位为HIGH。通过执行接收FIFO的单次读取，或通过

清除中断。

* + - 1. UARTTXINTR

当发生以下事件之一时，传输中断改变状态

* + - * + 如果FIFO使能且发送FIFO等于或低于编程触发电平，则发送中断置位为高电平。发送中断通过向发送FIFO写入数据来清除，直到

大于触发电平，或清除中断。

* + - * + 如果FIFO被禁用（具有一个位置的深度），并且发送器单个位置中没有数据，则发送中断被置位为高电平。通过对发送FIFO执行单次写入来清除，或

通过清除中断。

要更新发送FIFO，您必须:

* + - * + 在使能FIFO和中断之前或使能FIFO和中断之后，将数据写入发送FIFO

**注意**

传输中断基于通过电平的转换，而不是电平本身。如果在将任何数据写入发送FIFO之前使能中断和FIFO，则不会设置中断只有在写入数据离开发送FIFO的单个位置且其变为空后，才设置中断

* + - 1. UARTRTINTR

当接收FIFO不为空且在32位周期内没有接收到更多数据时，接收超时中断置位当FIFO通过读取所有数据（或通过读取保持寄存器）变为空时，或当1写入到FIFO清除寄存器[UARTICR](#_bookmark17)的相应位时，接收超时中断被清除。

* + - 1. UARTEINTR

错误中断是断言当错误发生在接收数据的数据由单片机。中断可能由多种不同的错误条件引起:

* + - * + 成帧
        + 奇偶校验
        + 打破
        + 超限。

通过读取原始中断状态寄存器[UARTRIS](#_bookmark15)或屏蔽中断状态寄存器[UARTMIS，](#_bookmark16)可以确定中断的原因。可通过写入错误清除寄存器[UARTICR](#_bookmark17)的相关位（位7至10为错误清除位）来清除。

* + - 1. UARTINTR

中断也被合并为单个输出，即各个屏蔽源的OR函数您可以将此输出连接到系统中断控制器，以在单个外围设备的基础上提供另一级屏蔽

如果任何单独的中断被断言并使能，则组合中断被断言

* + 1. 程序员模型

SDK提供了一个uart\_init函数来配置具有特定波特率的UART初始化GPIO后，用户必须将GPIO引脚配置为GPIO\_TX和GPIO\_RX。有关选择GPIO功能的更多信息，请参见2.19.5.1部分

要初始化该函数，uart\_init函数执行以下步骤:

* 取消断言重置
* 启用\_
* 设置控制寄存器中的使能位
* 启用FIFO
* 设置波特率除数
* 设置格式

*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#L39-L65)第39 - 65*

39 uint uart\_init（uart\_inst\_t \*uart，uint波特率）{

40

41

42

43

44

45

46

47

48

invalid\_params\_if（int，uart！=uart0 uart！=uart1）;

if（clock\_get\_hz（）==0）{ return0;

}

uart\_reset（uart）; unreset（uart）;

1. #如果PICO\_ENABLE\_CRLF\_SUPPORT
2. uart\_set\_translate\_crlf（uart，PICO\_PROFILE\_CRLF）;
3. #结束

52

1. //任何LCR写入都需要在启用缓存之前进行
2. uint baud = uart\_set\_baudrate（uart，baudrate）;
3. uart\_set\_format（uart，8，1，parity\_NONE）;

56

57

58

59

60

61

62

63

64

//启用FIFO（必须在设置UARTEN之前，因为这是LCR访问）

hw\_set\_bits（uart\_get\_hw（uart）->lcr\_h，UARTLCR\_H\_FEN\_BITS）;

//使能TX和RX两个端口

uart\_get\_hw（uart）->cr= UARTCR\_UARTEN\_BITS |UARTCR\_TXE\_BITS |UARTCR\_RXE\_BITS;

//始终启用DREQ信号--如果DMA没有监听，则不会造成任何损害

uart\_get\_hw（uart）->dmacr= UARTDMACR\_TXDMAE\_BITS |RXDMAE\_BITS;

返回波特率;

65}

* + - 1. 波特率计算

UART的波特率是从除以UART得到的。

如果所需波特率为115200，UARTCLK = 125 MHz，则波特率除数=（125 \* 10^6）/（16 \* 115200）~= 67.817因此，BRDI = 67，BRDF = 0.817，

因此，小数部分，m =整数（（0.817 \* 64）+0.5）= 52生成波特率分频器= 67 + 52/64 = 67.8125生成波特率=（125 \* 10^6）/（16 \* 67.8125）~= 115207误差=（abs（115200 - 115207）/ 115200）\* 100 ~= 0.006%

*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#L128-L153)第128 - 153*

128 uint uart\_set\_baudrate（uart\_inst\_t \*uart，uint baudrate）{

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153}

invalid\_params\_if（int，baudrate ==0）;

uint32\_tbaud\_rate\_div =（8\* clock\_get\_hz（时钟）/波特率）; uint32\_tbaud\_ibrd = baud\_rate\_div >>7;

uint32\_tbaud\_fbrd;

if（baud\_ibrd ==0）{ baud\_ibrd =1;

return0;

}else if（baud\_ibrd >=65535）{ baud\_ibrd =65535;

return0;

}否则{

baud\_fbrd =（（baud\_rate\_div0x7f）+1）/2;

}

uart\_get\_hw（uart）->ibrd = baud\_ibrd; uart\_get\_hw（uart）->fbrd = baud\_fbrd;

//PL 011需要一个（虚拟）LCR\_H写入来锁存除数。

//我们不想在这里实际更改LCR\_H内容

uart\_write\_lcr\_bits\_masked（uart，0，0）;

//请参见

return（4\* clock\_get\_hz（时钟\_频率））/（64\* baud\_ibrd + baud\_fbrd）;

* + 1. 寄存器列表

425号列表

426号。UARTDR

寄存器

UART0和UART1寄存器分别从基址0x40034000和0x40038000开始（在SDK中定义为UART0\_BASE和UART1\_BASE）。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x000 | [UARTDR](#_bookmark5) | 数据寄存器，UARTDR |
| 0x004 | [UARTRSR](#_bookmark6) | 接收状态寄存器/错误清除寄存器 |
| 0x018 | [UARTFR](#_bookmark7) | 标志寄存器，UARTFR |
| 0x020 | [UARTILPR](#_bookmark8) | 低功耗计数器寄存器 |
| 0x024 | [UARTIBRD](#_bookmark9) | 波特率寄存器 |
| 0x028 | [UARTFBRD](#_bookmark10) | 小数波特率寄存器 |
| 0x02c | [UARTLCR\_H](#_bookmark11) | 线路控制寄存器UARTLCR\_H |
| 0x030 | [UARTCR](#_bookmark12) | 控制寄存器 |
| 0x034 | [UARTIFLS](#_bookmark13) | FIFO电平选择寄存器 |
| 0x038 | [UARTMSC](#_bookmark14) | 屏蔽设置/清除寄存器 |
| 0x03c | UARTRIS | 原始数据状态寄存器 |
| 0x040 | [UTMIS](#_bookmark16) | 屏蔽状态寄存器 |
| 0x044 | [UARTICR](#_bookmark17) | 清除寄存器 |
| 0x048 | [UARTDMACR](#_bookmark18) | DMA控制寄存器 |
| 0xfe0 | [UARTPERIPHID0](#_bookmark19) | UARTMENTID0寄存器 |
| 0xfe4 | [UARTPERIPHID1](#_bookmark20) | UARTMENTID1寄存器 |
| 0xfe8 | [UARTPERIPHID2](#_bookmark21) | UARTMENTID2寄存器 |
| 0xfec | [UARTPERIPHID3](#_bookmark22) | UARTMENTID3寄存器 |
| 0xff0 | [UARTPCELLID0](#_bookmark23) | UARTPCellID0寄存器 |
| 0xff4 | [UARTPCELLID1](#_bookmark24) | UARTPCellID1寄存器 |
| 0xff8 | [UARTPCELLID2](#_bookmark25) | UARTPCellID2寄存器 |
| 0xffc | [UARTPCELLID3](#_bookmark26) | UARTPCellID3寄存器 |

### [用户](#_bookmark4):UARTDR寄存器

**偏移**:0x000

描述

数据寄存器，UARTDR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - | - |
| 11 | OE | 溢出错误。如果接收到数据且接收FIFO已满，则该位置1一旦FIFO中有空空间并且可以向其写入新字符，该值将被清除为0 | RO | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 10 | BE | 中断错误。如果检测到中断条件，则该位设置为1，表示接收到的数据输入保持低电平的时间长于全字传输时间（定义为起始位、数据位、奇偶校验位和停止位）。在FIFO模式下，此错误与FIFO顶部的字符相关当中断发生时，只有一个0字符被加载到FIFO中。下一个字符仅在接收数据输入变为1（标记状态）并接收到下一个有效起始位后启用 | RO | - |
| 9 | PE | 奇偶校验错误。设为1时，表示接收数据字符的奇偶校验与线路控制寄存器UARTLCR\_H中EPS和SPS位的奇偶校验不匹配在FIFO模式下，此错误与FIFO顶部的字符 | RO | - |
| 8 | Fe | 框架错误。当设置为1时，表示接收到的字符没有有效的停止位（有效的停止位为1）。在FIFO模式下，此错误与FIFO顶部 | RO | - |
| 7:0 | DATA | 接收（读）数据字符。发送（写入）数据字符。 | RWF | - |

427号UARTRSR

寄存器

### [用户](#_bookmark4):UARTRSR寄存器

**偏移量**:0x004

描述

接收状态寄存器/错误清除寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:4 | Reserved. | - | - | - |
| 3 | OE | 溢出错误。如果接收到数据且FIFO已满，则该位置1写入UARTECR后，该位清0。FIFO内容保持有效，因为FIFO已满时不再写入数据，只有移位寄存器的内容被覆盖。CPU现在必须读取数据，以清空FIFO。 | WC | 0x0 |
| 2 | BE | 中断错误。如果检测到中断条件，则该位置1，表示接收到的数据输入保持低电平的时间长于全字传输时间（定义为起始位、数据位、奇偶校验位和停止位）。写入UARTECR后，该位清0在FIFO模式下，此错误与FIFO顶部的字符相关。当中断发生时，只有一个0字符被加载到FIFO中。下一个字符仅在接收数据输入变为1（标记状态）且接收到下一个有效起始位后启用 | WC | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 1 | PE | 奇偶校验错误。设为1时，表示接收数据字符的奇偶校验与线路控制寄存器UARTLCR\_H中EPS和SPS位的奇偶校验不匹配写入UARTECR后，该位清0。在FIFO模式下，此错误与FIFO顶部 | WC | 0x0 |
| 0 | Fe | 框架错误。当设置为1时，表示接收到的写入UARTECR后，该位清0在FIFO模式下，此错误与FIFO顶部的字符相关 | WC | 0x0 |

428号。UARTFR

寄存器

### [中文（](#_bookmark4)简体）

**偏移**:0x018

描述

标志寄存器，UARTFR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:9 | Reserved. | - | - | - |
| 8 | RI | 振铃指示器。该位是UART振铃指示符nUARTRI调制解调器状态输入的补充。也就是说，当nUARTRI为低电平时，该位为1 | RO | - |
| 7 | TXFE | 发送FIFO为空。此位的含义取决于线路控制寄存器UARTLCR\_H中FEN位的状态。如果FIFO禁用，则当发送保持寄存器为空时，该位置1。如果FIFO使能，则当发送FIFO为空时，TXFE位置1。此位不指示发送移位寄存器中是否有数据。 | RO | 0x1 |
| 6 | RXFF | 接收FIFO已满。该位的含义取决于UARTLCR\_H寄存器中FEN位的状态。如果FIFO禁用，则当接收保持寄存器满时，该位置1。如果FIFO使能，则RXFF位在接收FIFO满时置1。 | RO | 0x0 |
| 5 | TXFF | 发送FIFO已满。该位的含义取决于UARTLCR\_H寄存器中FEN位的状态。如果FIFO禁用，则该位在发送保持寄存器满时置1。如果FIFO使能，则当发送FIFO满时，TXFF位置1。 | RO | 0x0 |
| 4 | RXFE | 接收FIFO为空。该位的含义取决于UARTLCR\_H寄存器中FEN位的状态如果FIFO禁用，则当接收保持寄存器为空时，该位置1如果FIFO使能，则接收FIFO为空时RXFE位 | RO | 0x1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 3 | BUSY | 很忙。如果此位设置为1，则表示收发器正忙于传输数据。该位保持置位状态，直到从移位寄存器发送完包括所有停止位在内的完整字节发送FIFO变为非空时，该位即置1，而不管是否使能FIFO。 | RO | 0x0 |
| 2 | DCD | 数据载体检测。此位是调制解调器状态输入的补码，用于接收数据载波检测、nUARTDCD也就是说，当nUARTDCD为低电平时，该位为1 | RO | - |
| 1 | DSR | 数据准备就绪。该位是UART数据集就绪、nUARTDSR、调制解调器状态输入的补充也就是说，当nUARTDSR为低电平时，该位为1 | RO | - |
| 0 | CTS | 可以发送了此位是清零发送（nUARTCTS）调制解调器状态输入的补码。也就是说，当nUARTCTS为低电平时，该位为1 | RO | - |

表429。UARTILPR

寄存器

430号。UARTIBRD

寄存器

### [用户名](#_bookmark4):UARTILPR寄存器

**偏移**:0x020

描述

低功耗计数器寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | ILPDVSR | 8位低功耗除数值。复位时，这些位清 | RW | 0x00 |

### [中文](#_bookmark4)（简体）

**偏移量**:0x024

描述

波特率寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - | - |
| 15:0 | BAUD\_DIVINT | 整数波特率除数。复位时，这些位清 | RW | 0x0000 |

### [标签](#_bookmark4):UARTFBRD寄存器

**偏移量**:0x028

描述

小数波特率寄存器

*431号。UARTFBRD*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:6 | Reserved. | - | - | - |
| 5:0 | BAUD\_DIVFRAC | 小数波特率除数。复位时，这些位清 | RW | 0x00 |

寄存器

432号。UARTLCR\_H寄存器

### UARTLCR\_H寄存器

**偏移**:0x02c

描述

线路控制寄存器UARTLCR\_H

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7 | SPS | 坚持奇偶校验选择。0 =禁用坚持奇偶校验1 =任一:\* 如果EPS位为0，则传输奇偶校验位并检查为1 \* 如果EPS位为1，则传输奇偶校验位当PEN位禁用奇偶校验和生成时，此位无效。 | RW | 0x0 |
| 6:5 | WLEN | 单词长度。这些比特表示在一帧中发送或接收的数据比特数，如下:b11 = 8比特b10 = 7比特b01 = 6比特b00 = 5比特。 | RW | 0x0 |
| 4 | FEN | 启用FIFO:0 = FIFO被禁用（字符模式），即FIFO变为1字节深的保持寄存器1  =使能发送和接收FIFO缓冲器（FIFO模式）。 | RW | 0x0 |
| 3 | STP2 | 选择两个停止位如果此位设置为1，则在帧结束时发送两个停止位接收逻辑不检查是否接收到两个停止位 | RW | 0x0 |
| 2 | EPS | 偶数奇偶校验选择。控制发送和接收期间奇偶校验的类型:0 =奇奇偶校验。ADC在数据和奇偶校验位中生成或检查奇数个11 =偶数奇偶校验。ADC在数据和奇偶校验位中生成或检查偶数个1当PEN位禁用奇偶校验和生成时，此位无效。 | RW | 0x0 |
| 1 | PEN | 奇偶校验使能:0 =奇偶校验被禁用，没有奇偶校验位添加到数据帧1 =奇偶校验检查和生成被启用。 | RW | 0x0 |
| 0 | BRK | 发送中断。如果此位设置为1，则在完成当前字符的传输后，UARTTXD输出端将持续输出低电平为了正确执行break命令，软件必须为至少两个完整帧设置此位正常使用时，该位必须清0。 | RW | 0x0 |

### [用户](#_bookmark4):UARTCR寄存器

**偏移**:0x030

433号UARTCR

寄存器

描述

控制寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - | - |
| 15 | CTSEN | CTS硬件流控制使能。如果此位设置为1，则CTS硬件流控制使能。仅当nUARTCTS信号被置位时才传输数据。 | RW | 0x0 |
| 14 | RTSEN | RTS硬件流控制使能。如果此位设置为1，则RTS硬件流控制使能。仅当接收FIFO中有空间可供接收时，才请求数据。 | RW | 0x0 |
| 13 | OUT2 | 此位是调制解调器状态输出nUARTOut2（nUARTOut2）的补码也就是说，当该位编程为1时，输出为0。对于DTE，这可以用作振铃指示器（RI）。 | RW | 0x0 |
| 12 | OUT1 | 此位是调制解调器状态输出nUARTOut1（nUARTOut1）的补码也就是说，当该位编程为1时，输出为0。对于DTE，这可以用作数据载波检测（DCD）。 | RW | 0x0 |
| 11 | RTS | 请求发送。此位是调制解调器状态输出nUARTRTS的补码也就是说，当该位被编程为1时，nUARTRTS为低电平。 | RW | 0x0 |
| 10 | DTR | 数据传输准备就绪。该位是调制解调器数据传输就绪、nUARTDTR、调制解调器状态输出的补码也就是说，当该位被编程为1时，nUARTDTR为低电平。 | RW | 0x0 |
| 9 | RXE | 接收使能。如果该位设置为1，则使能ADC的接收部分根据SIREN位的设置，对SIR信号或SIR信号进行数据接收。当在接收过程中禁用字符串 | RW | 0x1 |
| 8 | TXE | 发射使能。如果该位设置为1，则使能ADC的发送根据SIREN位的设置，数据传输是针对SIR信号还是SIR信号进行的。当在传输过程中禁用字符串时，它会在停止之前完成当前 | RW | 0x1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 7 | LBE | 环回使能。如果此位设置为1，SIREN位设置为1，测试控制寄存器UARTTCR中的SIRTEST位设置为1，则nSIROUT路径反转，并馈入SIRIN路径。测试寄存器中的SIRTEST位必须设置为1，以覆盖正常的半双工SIR操作。这必须是在正常操作期间访问测试寄存器的要求，并且当完成测试时，SIRTEST必须清除为0。此功能减少了系统测试期间所需如果该位设为1，SIRTEST位设为0，则UARTTXD路径馈入UARTRXD路径。在SIR模式或SIR模式下，当该位置1时，调制解调器输出也馈通到调制解调器输入。复位时，该位清0，禁用复位。 | RW | 0x0 |
| 6:3 | Reserved. | - | - | - |
| 2 | SIRLP | SIR低功耗IrDA模式。此位选择IrDA编码模式。如果该位清0，则低电平位作为高电平有效脉冲传输，脉冲宽度为位周期的3 /16。如果该位被设置为1，则低电平位以IrLPBaud16输入信号周期的3倍的脉冲宽度发送，而与所选的位速率无关。设置此位会降低功耗，但可能会缩短传输距离。 | RW | 0x0 |
| 1 | SIREN | SIR使能:0 = IrDA SIR ENDEC被禁用。nSIROUT保持低电平（不产生光脉冲），SIRIN上的信号转换不起作用。1 = IrDA SIR ENDEC使能。数据在nSIROUT和SIRIN上传输和接收在标记状态下，UARTTXD保持高电平  UARTRXD或调制解调器状态输入上的信号转换没有影响。如果UARTEN位禁用复位，则此位无效 | RW | 0x0 |
| 0 | UARTEN | 禁用启用:0 =禁用。如果在传输或接收过程中禁用了字符串，则它会1 =使能UART。根据SIREN位的设置，UART信号或SIR信号会发生数据传输和接收。 | RW | 0x0 |

434号UARTIFLS

寄存器

### [中文（](#_bookmark4)简体）

**偏移量**:0x034

描述

FIFO电平选择寄存器

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 5:3 | RXIFLSEL | 接收中断FIFO电平选择。接收中断的触发点如下:b 000 =接收FIFO变为>= 1/8满b 001 =接收FIFO变为>= 1/ 4满b 010 =接收FIFO变为>= 1 / 2满b 011 =接收FIFO变为>= 3 / 4满b100 =接收FIFO变为>= 7/8满b101-b111 =保留。 | RW | 0x2 |
| 2::0 | TXIFLSEL | 发送中断FIFO电平选择。发送中断的触发点如下:b 000 =发送FIFO变为= 1/8满b 001 =发送FIFO变为  <= 1/4满b 010 =发送FIFO变为= 1/2满b 011 =发送FIFO变为= 3 / 4满b100 =发送FIFO变为= 7 / 8满b101-b111 =保留。 | RW | 0x2 |

435号UARTMSC

寄存器

### [UART](#_bookmark4):UARTIMSC寄存器

**偏移量**:0x038

描述

屏蔽设置/清除寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:11 | Reserved. | - | - | - |
| 10 | OEIM | 溢出错误中断屏蔽。读操作返回UARTOEINTR中断的当前掩码写入1时，UARTOEINTR中断的掩码被设置。写入0将清除掩码。 | RW | 0x0 |
| 9 | BEIM | 中断错误中断屏蔽。读操作返回UARTBEINTR中断的当前掩码。写入1时，UARTBEINTR中断的掩码被设置。写入0将清除掩码。 | RW | 0x0 |
| 8 | PEIM | 奇偶校验错误中断屏蔽。读操作返回UARTPEINTR中断的当前掩码写入1时，UARTPEINTR中断的掩码被设置。写入0将清除掩码。 | RW | 0x0 |
| 7 | FEIM | 成帧错误中断掩码。读操作返回UARTFEINTR中断的当前掩码写入1时，UARTFEINTR中断的掩码被设置。写入0将清除掩码。 | RW | 0x0 |
| 6 | RTIM | 接收超时中断掩码。读操作返回UARTRTINTR中断的当前掩码。写入1时，UARTRTINTR中断的掩码被设置。写入0将清除掩码。 | RW | 0x0 |
| 5 | TXIM | 传输中断屏蔽。读操作返回UARTTXINTR中断的当前掩码写入1时，UARTTXINTR中断的掩码写入0将清除掩码。 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 4 | RXIM | 接收中断掩码。读操作返回UARTRXINTR中断的当前掩码。写入1时，UARTRXINTR中断的掩码写入0将清除掩码。 | RW | 0x0 |
| 3 | DSRMIM | nUARTDSR调制解调器中断掩码。读操作返回UARTDSRINTR中断的当前掩码写入1时，UARTDSRINTR中断的掩码被设置。写入0将清除掩码。 | RW | 0x0 |
| 2 | DCDMIM | nUARTDCD调制解调器中断屏蔽。读操作返回UARTDCDINTR中断的当前掩码写入1时，UARTDCDINTR中断的掩码被设置。写入0将清除掩码。 | RW | 0x0 |
| 1 | CTSMIM | nUARTCTS调制解调器中断屏蔽。读操作返回UARTCTSINTR中断的当前掩码写入1时，UARTCTSINTR中断的掩码被设置。写入0将清除掩码。 | RW | 0x0 |
| 0 | RIMIM | nUARTRI调制解调器中断屏蔽。读操作返回UARTRIINTR中断的当前掩码写入1时，UARTRIINTR中断的掩码被设置。写入0将清除掩码。 | RW | 0x0 |

436号

寄存器

### [UARTRIS寄存器](#_bookmark4)

**偏移**:0x03c

描述

原始数据状态寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:11 | Reserved. | - | - | - |
| 10 | OERIS | 超限错误中断状态。返回UARTOEINTR中断的原始中断状态 | RO | 0x0 |
| 9 | BERIS | 中断错误中断状态。返回UARTBEINTR中断的原始中断 | RO | 0x0 |
| 8 | PERIS | 奇偶校验错误中断状态。返回UARTPEINTR中断的原始中断状态 | RO | 0x0 |
| 7 | FERIS | 帧错误中断状态。返回UARTFEINTR中断的原始中断状态 | RO | 0x0 |
| 6 | RTRIS | 接收超时中断状态。返回UARTRTINTR中断的原始中断状态一 | RO | 0x0 |
| 5 | TXRIS | 传输中断状态。返回UARTTXINTR中断的原始中断 | RO | 0x0 |
| 4 | RXRIS | 接收中断状态。返回UARTRXINTR中断的原始中断状态 | RO | 0x0 |
| 3 | DSRRMIS | nUARTDSR调制解调器中断状态。返回UARTDSRINTR中断的原始中断状态 | RO | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 2 | DCDRMIS | nUARTDCD调制解调器中断状态。返回UARTDCDINTR中断的原始中断状态 | RO | - |
| 1 | CTSRMIS | 调制解调器中断状态。返回UARTCTSINTR中断的原始中断状态 | RO | - |
| 0 | RIRMIS | nUARTRI调制解调器中断状态。返回UARTRIINTR中断的原始中断状态 | RO | - |

437号UTMIS

寄存器

### [用户](#_bookmark4):UARTMIS寄存器

**偏移**:0x040

描述

屏蔽状态寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:11 | Reserved. | - | - | - |
| 10 | OEMIS | 溢出错误屏蔽中断状态。返回UARTOEINTR中断的屏蔽中断状态 | RO | 0x0 |
| 9 | Bemis | 中断错误屏蔽中断状态。返回UARTBEINTR中断的屏蔽中断状态 | RO | 0x0 |
| 8 | PEMIS | 奇偶校验错误屏蔽中断状态。返回UARTPEINTR中断的屏蔽中断状态 | RO | 0x0 |
| 7 | FEMIS | 成帧错误屏蔽中断状态。返回UARTFEINTR中断的屏蔽中断状态 | RO | 0x0 |
| 6 | RTMIS | 接收超时屏蔽中断状态。返回UARTRTINTR中断的屏蔽中断状态 | RO | 0x0 |
| 5 | TXMIS | 传输屏蔽中断状态。返回UARTTXINTR中断的屏蔽中断状态 | RO | 0x0 |
| 4 | RXMIS | 接收屏蔽中断状态。返回UARTRXINTR中断的屏蔽中断状态 | RO | 0x0 |
| 3 | DSRMMIS | nUARTDSR调制解调器屏蔽中断状态。返回UARTDSRINTR中断的屏蔽中断状态 | RO | - |
| 2 | DCDMMIS | nUARTDCD调制解调器屏蔽中断状态。返回UARTDCDINTR中断的屏蔽中断状态 | RO | - |
| 1 | CTSMMIS | 调制解调器屏蔽中断状态。返回UARTCTSINTR中断的屏蔽中断状态 | RO | - |
| 0 | RIMMIS | nUARTRI调制解调器屏蔽中断状态。返回UARTRIINTR中断的屏蔽中断状态 | RO | - |

### UART: UARTICR寄存器

**偏移量**:0x044

描述

清除寄存器

438号。UARTICR

寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:11 | Reserved. | - | - | - |
| 10 | OEIC | 溢出错误中断清除。清除UARTOEINTR中断。 | WC | - |
| 9 | BEIC | 中断错误中断清除。清除UARTBEINTR中断。 | WC | - |
| 8 | PEIC | 奇偶校验错误中断清除。清除UTPEINTR中断。 | WC | - |
| 7 | FEIC | 帧错误中断清除。清除UARTFEINTR中断。 | WC | - |
| 6 | RTIC | 接收超时中断清除。清除UARTRTINTR中断。 | WC | - |
| 5 | TXIC | 传输中断清除。清除UARTTXINTR中断。 | WC | - |
| 4 | RXIC | 接收中断清除。清除UARTRXINTR中断。 | WC | - |
| 3 | DSRMIC | nUARTDSR调制解调器中断清除。清除UARTDSRINTR中断。 | WC | - |
| 2 | DCDMIC | nUARTDCD调制解调器中断清除。清除UARTDCDINTR中断。 | WC | - |
| 1 | CTSMIC | nUARTCTS调制解调器中断清除。清除UARTCTSINTR中断。 | WC | - |
| 0 | RIMIC | nUARTRI调制解调器中断清除。清除UARTRIINTR中断。 | WC | - |

439号。UARTDMACR寄存器

### UART:UARTDMACR寄存器

**偏移量**:0x048

描述

DMA控制寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:3 | Reserved. | - | - | - |
| 2 | DMAONERR | DMA错误。如果此位设置为1，则当置位错误中断时，DMA接收请求输出UARTRXDMASSEND或UARTRXDMABREQ禁用。 | RW | 0x0 |
| 1 | TXDMAE | 发送DMA使能。如果该位设置为1，则发送FIFO的DMA使能。 | RW | 0x0 |
| 0 | RXDMAE | 接收DMA使能。如果该位设置为1，则使能接收FIFO的DMA | RW | 0x0 |

### UART:UARTPERIPHID 0寄存器

**偏移量**:0xfe0

描述

UARTMENTID0寄存器

*440号。UARTPERIPHID0*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | 零件编号0 | 这些位读回为0x11 | RO | 0x11 |

寄存器

441号。UARTPERIPHID1

寄存器

442号。UARTPERIPHID2

寄存器

443号。UARTPERIPHID3

寄存器

### UART:UARTPERIPHID 1寄存器

**偏移量**:0xfe4

描述

UARTMENTID1寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 七比四 | 公司简介 | 这些位读回为0x1 | RO | 0x1 |
| 三比零 | 零件编号1 | 这些位读回为0x0 | RO | 0x0 |

### [用户名](#_bookmark4):UARTPERIPHID 2寄存器

**偏移量**:0xfe8

描述

UARTMENTID2寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 七比四 | 修订 | 此字段取决于版本号:r1p0 0x0 r1p1 0x1 r1p3 0x2 r1p4 0x2 r1p5 0x3 | RO | 0x3 |
| 三比零 | 联系我们 | 这些位读回为0x 4 | RO | 0x4 |

### [用户名](#_bookmark4):UARTPERIPHID 3寄存器

**偏移量**:0xfec

描述

UARTMENTID3寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | 配置 | 这些位回读为0x00 | RO | 0x00 |

### [用户名](#_bookmark4):UARTPCELLID0寄存器

**偏移量**:0xff0

描述

UARTPCellID0寄存器

*444号。UARTPCELLID0*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | UARTPCELLID0 | 这些位读回为0x0D | RO | 0x0D |

寄存器

445号。UARTPCELLID1

寄存器

446号。UARTPCELLID2

寄存器

447号。UARTPCELLID3

寄存器

### [用户名](#_bookmark4):UARTPCELLID1寄存器

**偏移量**:0xff4

描述

UARTPCellID1寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | UARTPCELLID1 | 这些位回读为0xF0 | RO | 0xf0 |

### [用户名](#_bookmark4):UARTPCELLID2寄存器

**偏移量**:0xff8

描述

UARTPCellID2寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | UARTPCELLID2 | 这些位读回为0x05 | RO | 0x05 |

### [用户名](#_bookmark4):UARTPCELLID3寄存器

**偏移量**:0xffc

描述

UARTPCellID3寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | UARTPCELLID3 | 这些位回读为0xB1 | RO | 0xb1 |

* 1. I2c

###### Synopsys文档

Synopsys专有。经允许使用

I2C是一种常用的2线式接口，可用于连接器件，以便使用时钟SCL进行低速数据传输

和数据SDA线。

RP 2040有两个相同的I2C控制器实例每个控制器的外部引脚都连接到GPIO引脚，如第2.19.2GPIO多路复用表中所定义。多路复用选项提供了一定的IO灵活性。

* + 1. 特征

每个I2C控制器都基于Synopsys DW\_apb\_i2c（v2.01）IP的配置。支持以下功能:

* + - * Master或Slave（默认为Master模式）
      * 标准模式、快速模式或快速模式+
      * 默认从机地址0x055
      * 在主机模式下支持10位寻址
      * 16-单元发送缓冲器
      * 16-元件接收缓冲器
      * 可从DMA驱动
      * 可以生成中断
      1. 标准

I2C控制器是根据2014年4月发布的I2C总线规范6.0版设计的

* + - 1. 时钟

I2C控制器中的所有时钟都连接到I2C\_sys，包括后面章节中提到的ic\_sysI2C时钟由该时钟分频产生，由模块内部的寄存器控制

* + - 1. iOS

每个控制器必须将其时钟SCL和数据SDA连接到一对GPIO。I2C标准要求驱动器将信号驱动为低电平，或者当不驱动时，信号将被拉高。这适用于SCL和SDA。GPIO焊盘应配置为:

* + - * + 上拉使能
        + 限摆率
        + 施密特触发器使能

**注意**

电路板上还应该有外部上拉，因为内部焊盘上拉可能不够强，无法上拉外部电路。

* + 1. IP配置

I2C配置细节（每个实例完全独立）:

* + - * 32-位APB存取
      * 支持标准模式、快速模式或快速模式+（非高速）
      * 默认从机地址为0x055
      * 主模式或从模式
      * 默认情况下为主机（复位时禁用从机模式
      * 主机模式下支持10位寻址（默认为
      * 16入口发送缓冲器
      * 16入口接收缓冲器
      * 当主机（可以禁用以支持旧设备
      * 可配置的时序来调整TsuDAT/ThDAT
      * 重置时响应的一般呼叫
      * DMA接口
      * 单中断输出
      * 可配置时序以调整时钟频率
      * 尖峰抑制（默认为7个周期）
      * 从机收到数据后是否可以NACK
      * TX FIFO为空时保持传输
      * 保持总线，直到RX FIFO中有可用空间
      * 从机模式下重启检测中断
      * 可选的阻塞主机命令（默认情况下不启用
    1. I2C概述

I2C总线是一个双线串行接口，由串行数据线SDA和串行时钟SCL组成。这些导线在连接到总线的设备每个设备都有一个唯一的地址，根据设备的功能，可以在执行数据传输时，设备也可以被视为主设备或从设备。主机是一种在总线上发起数据传输并生成时钟信号以允许该传输的设备。此时，任何被寻址的设备都被视为从设备。

**注意**

I2C模块只能编程为仅在主机或从机模式下工作不支持同时作为主设备和从设备运行

I2C模块可以在以下模式下工作

* + - * 标准模式（数据速率从0到100kbps），
      * 快速模式（数据速率小于或等于400kbps），
      * 快速模式+（数据速率小于或等于1000kbps）。不支持以下模式:
      * 高速模式（数据速率小于或等于3.4Mbps），
      * 超高速模式（数据速率小于或等于5Mbps）。

**注意**

对快速模式的引用也适用于快速模式+，除非另有特别说明

I2C模块可以与这些模式之一的设备通信，只要它们连接到总线。此外，快速模式设备向下兼容。例如，快速模式设备可以在0到100kbps的I2C总线系统中与标准模式设备通信。然而，标准模式设备不是向上兼容的，并且不应该被并入快速模式I2C总线系统中，因为它们不能遵循更高的传输速率并且会发生不可预测的状态。

高速模式器件的一个示例是LCD显示器、高位计数ADC和高容量EEPROM。这些设备通常需要传输大量数据。大多数维护和控制应用，通常使用I2C总线，通常工作在100 kHz（标准和快速模式）。任何DW\_apb\_i2c设备都可以连接到I2C总线，并且每个设备都可以与任何主机通信，来回传递信息总线上需要至少一个主机（如微控制器或DSP），但可以有多个主机，这需要它们仲裁所有权。多主机和仲裁将在本章后面介绍。I2C模块不支持SMBus和PMBus协议（用于系统管理和电源管理）。

DW\_apb\_i2c由一个AMBA APB从机接口、一个I2C接口和FIFO逻辑组成，用于保持两个接口之间的一致性。组件的模块如[图64](#_bookmark27)所示。

*图64. I2C框图*

DW\_apb\_i2c

TX FIFO

RX FIFO

中断控制器

DMA接口

同步器

切换

Rx滤波器

Tx移位

Rx移位

时钟发生器

主状态机

从状态机

寄存器文件

AMBA总线

接口单元

下面定义了[图64](#_bookmark27)中的块的功能:

* + - * **AMBA总线接口单元**- 获取APB接口信号并将其转换为通用通用接口，该通用接口允许寄存器文件与总线协议无关。
      * **寄存器文件**- 包含配置寄存器，是与软件的接口
      * **从机状态机**- 遵循从机的协议并监控总线的地址匹配。
      * **主状态机**- 为主传输生成I2C协议
      * **时钟发生器**- 计算执行以下操作所需的时序
        + 配置为主机时生成SCL时钟
        + 检查总线空闲
        + 生成START和STOP
        + 设置数据并保存数据
      * **Rx Shift**- 将数据纳入设计并以字节格式提取
      * **Tx Shift**- 显示CPU提供的数据，以便在I2C总线上传输
      * **Rx Filter**- 检测总线中的事件;例如，启动、停止和仲裁丢失。
      * **Toggle**-在两端产生脉冲并切换以跨时钟域传输信号。
      * **同步器**- 将信号从一个时钟域传输到另一个时钟域。
      * **DMA接口**- 生成到中央DMA控制器的握手信号，以便在没有CPU干预的情况下自动进行
      * **中断控制器**- 生成原始中断和中断标志，允许设置和清除它们
      * **RX FIFO**/**TX FIFO**-保存RX FIFO和TX FIFO寄存器组和控制器及其状态级别。
    1. I2C术语

使用的术语和定义如下:

* + - 1. I2C总线术语

以下术语涉及I2C器件的作用以及它如何与总线上的其他I2C器件交互

* + - * + **发送器**-将数据发送到总线的设备发送器可以是发起到总线的数据传输的设备（主发送器），也可以是响应来自主发送器的请求以将数据发送到总线的设备（主发送器）。

从发射机）。

* + - * + **接收器**-从总线接收数据的设备接收器可以是根据自己的请求（主接收器）或响应于来自主的请求（从接收器）接收数据的设备
        + **主机**-启动传输（START命令）、生成时钟SCL信号并终止传输（STOP命令）的组件主设备可以是发射机或接收机。
        + **从机**- 由主机寻址的设备从机可以是接收机或发射机。
        + **多主机**-多个主机同时在总线上共存而不会发生冲突或数据丢失的能力。
        + **仲裁**-一次只授权一个主机控制总线的预定义过程有关此行为的更多信息，请参见[第4.3.8](#_bookmark48)。
        + **同步**-预定义的过程，同步两个或多个主机提供的时钟信号有关此功能的更多信息，请参见[第4.3.9](#_bookmark51)。
        + SDA- 数据信号线（串行数据）
        + SCL-时钟信号线（串行时钟）
      1. 巴士接送条款

以下术语专用于与I2C总线之间的数据传输

* + - * + **START（RESTART）**-数据传输以START或RESTART条件开始SDA数据线的电平从高电平变为低电平，而SCL时钟线保持高电平。当这种情况发生时，总线变得繁忙。

**注意**

START和RESTART条件在功能上是相同的。

* + - * + **STOP**-数据传输由STOP条件终止当SDA数据线上的电平从低状态变为高状态，而SCL时钟线保持高电平时，就会发生这种情况当数据传输完成后，

结束时，总线再次空闲或空闲如果生成RESTART而不是STOP条件，则总线保持忙碌状态

* + 1. I2C行为

DW\_apb\_i2c可通过软件控制为:

* + - * 仅I2C主机，与其他I2C从机通信;或
      * 仅I2C从机，与一个或多个I2C主机通信

主机负责产生时钟和控制数据传输。从机负责向主机发送数据或从主机接收数据。数据确认由接收数据的设备发送，该设备可以是主设备或从设备。如前所述，I2C协议还允许多个

主机驻留在I2C总线上，并使用仲裁程序来确定总线所有权。

每个从机都有一个由系统设计人员确定的唯一地址当主设备想要与从设备通信时，主设备发送START/RESTART条件，然后发送从设备的地址和控制位（R/W），以确定主设备想要发送数据还是从设备接收数据。然后，从机在地址之后发送确认（ACK）脉冲

如果主机（主发送器）正在向从机（从接收器）写入，则接收器获得一个字节的数据。此事务将继续进行，直到主机在STOP条件下终止传输如果主设备从从设备（主接收器）读取数据此事务将继续，直到主机在收到最后一个字节后通过不确认（NACK）事务来终止传输，然后主机发出STOP条件或在发出RESTART条件后寻址另一个从机这种行为如[图65](#_bookmark28)所示。

*图65. I2C总线上的数据传输*

SDA

MSB

LSB

ACK

从接收器

P

或R

SCL

S

或R

1

2 7

8

ACK

从奴隶

9

1

2 3-8

9

R

或P

SCL在服务中断时保持低电平

START或RESTART条件

从机内的字节完全计数

停止和重新启动条件

DW\_apb\_i2c是一个同步串行接口。SDA线是双向信号，仅在SCL线为低电平时发生变化，但STOP、START和RESTART条件除外。输出驱动器是开漏或集电极开路，用于在总线上执行线与功能。总线上的最大器件数量仅受400 pF的最大电容规格数据以字节包传输

DW\_apb\_i2c中实现的I2C协议在[第4.3.6](#_bookmark30)中有更详细的描述。

* + - 1. START和STOP生成

作为I2C主机工作时，将数据放入发送FIFO会导致DW\_apb\_i2c在I2C总线上生成START条件向[IC\_DATA\_CMD](#_bookmark67).STOP写入1会导致DW\_apb\_i2c在I2C总线上生成STOP条件;如果该位未置1，即使发送FIFO为空，也不会发出STOP条件

作为从机工作时，DW\_apb\_i2c不会根据协议生成START和STOP条件。但是，如果向DW\_apb\_i2c发出读请求，则它会将SCL线保持为低电平，直到向其提供读数据。这会使I2C总线暂停，直到向从机DW\_apb\_i2c提供读数据，或者通过向[IC\_ENABLE](#_bookmark88). ENABLE写入0来禁用DW\_apb\_i2c从机

* + - 1. 组合式电风扇

DW\_apb\_i2c支持7位和10位寻址模式下的混合读写组合格式事务。DW\_apb\_i2c不支持混合地址和混合地址格式，即7位地址事务后接10位地址事务，或7位地址合并格式事务。要启动组合格式传输，[IC\_CON](#_bookmark64).IC\_RESTART\_EN应设置为1。设置此值并作为主机运行时，DW\_apb\_i2c完成I2C传输时，会检查发送FIFO并执行下一次传输。如果此转帐的方向与前一转帐不同，则使用组合格式发出转帐。如果当前I2C传输完成时发送FIFO为空

* + - * + 检查[IC\_DATA\_CMD](#_bookmark67).STOP，并且:

如果设置为1，则发出STOP位

如果设置为0，SCL保持低电平，直到下一个命令写入发送FIFO。更多详细信息，请参见[第4.3.7](#_bookmark39)。

* + 1. I2C协议

DW\_apb\_i2c具有本节中讨论的协议

* + - 1. 开始和停止状况

当总线空闲时，SCL和SDA信号都通过总线上的外部上拉电阻拉高当主机想要在总线上开始传输时，主机发出START条件。这被定义为当SCL为1时SDA信号的高到低转换当主机想要终止传输时，主机发出STOP条件。这被定义为当SCL为1时SDA线的低到高转换[图66](#_bookmark31)显示了启动和停止条件的时序当数据在总线上传输时，当SCL为1时，SDA线必须稳定

*图66.I2C开始*

SDA

SCL

S

P

开始条件 更改数据允许数据行稳定数据有效

允许更改

停止条件

和停止状况

**注意**

START/STOP条件的信号转换（如[图6和](#_bookmark31)6所示）反映了在驱动I2C总线的主机输出在从机输入信号处观察SDA/SCL信号时应格外小心，因为不相等的线路延迟可能导致SDA/SCL时序关系不正确

* + - 1. 寻址从属协议

有两种地址格式:7位地址格式和10位地址格式。

* + - * 1. 7位地址格式

在7位地址格式期间，第一个字节的前7位（位7:1）设置从机地址，LSB位（位0）为R/W位，如[图67](#_bookmark32)所示。当位0（R/W）设置为0时，主机写入从机。当位0（R/W）设置为1时，主机从从机读取

*图67. I2C 7位地址格式*

S A6 A5 A4 A3 A2 A1 A0 R/W ACK

从机发送

从机地址

S =开始条件ACK =确认R/W =读/写脉冲

* + - * 1. 10位地址格式

在10位寻址期间，传输两个字节以设置10位地址。第一个字节的传输包含以下位定义。前五位（位7:3）通知从机这是一次10位传输，随后两位（位2:1）设置从机地址位9:8，LSB位（位0）是R/W位。传输的第二个字节设置从机地址的位7:0[图68](#_bookmark33)显示了10位地址格式。

*图68. 10位地址格式*

Ave

为10位地址保留

S =开始条件ACK =确认R/W =读/写脉冲

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S | ‘1’ | ‘1’ | ‘1’ | ‘0’ | A9 | A8 | R/W | ACK | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | ACK |
|  |  |  |  |  |  |  |  | 发送 | 苏丹解放军 | ve |  |  |  |  |  | 发送 | 由sl |

表448.I2C/SMBus第一个字节

[此表](#_bookmark34)定义了专用和保留的第一字节地址。

|  |  |  |
| --- | --- | --- |
| 从机地址 | R/W位 | 描述 |
| 0000 000 | 0 | 一般呼叫地址。DW\_apb\_i2c将数据放入接收缓冲区并发出通用调用中断。 |
| 0000 000 | 1 | 开始字节。有关更多详细信息，请参阅[4.3.6.4部分](#_bookmark37)。 |
| 0000 001 | X | CBUS地址。DW\_apb\_i2c忽略这些访问。 |
| 0000 010 | X | Reserved. |
| 0000 011 | X | Reserved. |
| 0000 1XX | X | 高速主代码（更多信息请参见[第4.3.8节](#_bookmark48)）。 |
| 小行星1111 1XX | X | Reserved. |
| 1111 0XX | X | 10-位从属寻址。 |
| 0001 000 | X | SMBus主机（不支持） |
| 0001 100 | X | SMBus警报响应地址（不支持） |
| 1100 001 | X | SMBus设备默认地址（不支持） |

DW\_apb\_i2c不限制您使用这些保留地址。但是，如果您使用这些保留地址，则可能会遇到与其他I2C组件不兼容的问题

* + - 1. 发送和接收协议

主设备可以启动到/从总线的数据传输和接收，充当主发送器或主接收器。从机响应来自主机的请求，向总线发送数据或从总线接收数据，分别充当从机发送器或从机接收器

* + - * 1. 主发送器和从接收器

所有数据都以字节格式传输，对每次数据传输的字节数没有限制在主机发送地址和R/W位或主机向从机发送一个字节的数据后，从机接收器必须以确认信号（ACK）进行响应当从接收器不响应ACK脉冲时，主机通过发出STOP条件来中止传输。从机必须使SDA线保持高电平，以便主机可以中止传输。如果主发送器正在发送数据，如图6和[图9](#_bookmark35)所示，则第n个从接收器在接收到每个字节的数据之后用确认脉冲响应主发送器

*图69.I2C主机-发送器协议*

对于7位地址

S 从机地址 R/W 一 数据 一 数据 A/A P

“0”（已读）

对于10位地址

S

从机地址前7位

R/W A

从机地址第二字节

数据 A/A P

'11110xxx' “0”（写入）

从主机到从机A =确认（SDA低）S =开始条件从机到主机A =无确认（SDA高）P =停止条件

* + - * 1. 主接收器和从发送器

[如图70所示，如果](#_bookmark36)主设备正在接收数据，则第n个主设备在已经接收到数据的一个字节（除了最后一个字节）之后用确认脉冲来响应从发送这是主接收器通知从发送器这是最后一个字节的方式从发射机在检测到无应答（NACK）后放弃SDA线路，以便主机可以发出STOP条件。

*图70. I2C主机-接收器协议*



对于7位地址

S 从机地址 R/W 一 数据 一 数据 一个P

“% 1”（已读）

对于10位地址

S

从机地址前7位

R/W A

从机地址第二字节

的Sr

从机地址前7位

R/W A 数据 一个P

'11110xxx' “0”（写入）

'11110xxx' “% 1”（已读）

从主控器到从控器

从奴隶到主人

A =确认（SDA低）

A =无确认（SDA高电平）S =启动条件

R =重新启动条件

P =停止条件

当主机不想在STOP条件下放弃总线时，主机可以发出RESTART条件。这与START条件相同，除了它发生在ACK脉冲之后在主机模式下工作时，DW\_apb\_i2c可以使用不同方向的传输与同一从机通信有关DW\_apb\_i2c支持的组合格式事务的说明，请参阅[4.3.5.2部分](#_bookmark29)。

**注意**

在重新编程目标从机地址寄存器（[IC\_TAR](#_bookmark65)）之前，必须完全禁用DW\_apb\_i2c。

* + - 1. 起始字节传输协议

START BYTE传输协议适用于没有板载专用I2C硬件模块的系统当DW\_apb\_i2c作为从机寻址时，它始终以支持的最高速度对I2C总线进行采样，因此它永远不需要传输START BYTE。但是，当DW\_apb\_i2c是主机时，它支持在每次传输开始时生成START BYTE传输，以防从机设备需要。

该协议由七个0和一个1组成，如[图71](#_bookmark38)所示。这允许轮询总线的处理器对地址相位进行欠采样，直到检测到零。一旦微控制器检测到零点，它就会从欠采样率切换到主机的正确采样率

*图71. I2C开始字节传输*

SDA

虚拟送达确认

SCL

1 2

7 8

S

（HIGH）9

Ack

Sr

起始字节00000001

START BYTE程序如下:

1. 主机生成启动条件。
2. 主机发送START字节（0000 0001）。
3. 主机发送ACK时钟脉冲。（仅用于符合总线上使用的字节处理格式
4. 没有从机将ACK信号设置为零。
5. 主机生成RESTART（R）条件。

硬件接收器不响应START BYTE，因为它是一个保留地址，并在RESTART条件生成后复位。

* + 1. Tx FIFO管理和START、STOP和RESTART生成

作为主机运行时，DW\_apb\_i2c组件支持[图72](#_bookmark40)所示的Tx FIFO管理模式

4.3.7.1. Tx FIFO管理

如果Tx FIFO为空，器件不会生成STOP;在这种情况下，器件会将SCL线保持为低电平，从而使总线停止，直到Tx FIFO中有新条目可用。仅当用户通过设置写入[IC\_DATA\_CMD](#_bookmark67)寄存器的命令的第9位（停止位）明确请求时，才会生成停止条件[图72](#_bookmark40)显示[IC\_DATA\_CMD](#_bookmark67)寄存器中的位

IC\_DATA\_CMD

9 8 7

0

数据读/写字段;从从属设备检索的数据从该字段读取;要发送到从属设备的数据写入该字段

CMD只写字段;此位确定要执行的传输是读（CMD=1）还是写（CMD=0）

停止只写字段;此位确定在发送或接收数据字节后是否生成STOP

重启只写字段;此位确定在发送或接收数据之前是否生成RESTART（或在未启用重启功能的情况下，在STOP后生成START）

图72. [IC\_DATA\_CMD](#_bookmark67)

寄存器

|  |  |  |  |
| --- | --- | --- | --- |
| 重启 | 停止 | CMD | 数据 |

[图73](#_bookmark41)显示了DW\_apb\_i2c作为主发送器工作时Tx FIFO变为空时的行为，并显示了STOP条件的生成

*图73.主发送器-发送FIFO清空/停止生成*

S

P

SDA

一个6 一个5 一个4 一个3 一个2 的1 a0级 W确认D7 D6 D5 D4 D3 D2 D1 D0确认D 7 D6 D5 D4 D3 D2 D1 D0确认

D7 D6 D5 D4 D3 D2 D1 D0 Ack

SCL

FIFO\_EMPTY

STOP位使能触发总线

数据可用性触发总线上的START条件

最后一个字节从弹出因为停止位未在Tx FIFO上设置，停止位最后一个字节从Tx FIFO弹出

Tx FIFO加载新数据

Tx FIFO加载数据（本例中为写入数据

未设置

主机保持SCL低电平

主机释放SCL线路并恢复传输，因为新数据可用

从Tx FIFO弹出的最后一个字节，设置STOP位

[图74](#_bookmark42)显示了DW\_apb\_i2c作为主接收器工作时Tx FIFO变为空时的行为，并显示了STOP条件的生成

*图74.主接收器-Tx FIFO清空/停止生成*

S

S

SDA

一个6 一个5 一个4 一个3 一个2 的1 a0级 R Ack D7 D6 D5 D4 D3 D2 D1 D0确认D 7 D6 D5 D4 D3 D2 D1 D0

确认D7 D6 D5 D4 D3 D2 D1 D0Nak

SCL

FIFO\_EMPTY

命令可用性触发总线上的START条件

由于从Tx FIFO弹出的最后一个命令未设置STOP位，因此主机将SCL保持为低电平

STOP位使能触发总线

Tx FIFO加载命令（本例中为读操作

从Tx FIFO弹出的最后一个命令，STOP位未设置

发送FIFO加载了新命令

主机释放SCL线路并恢复传输，因为新命令可用

从Tx FIFO弹出的最后一个命令，设置STOP位

[图75](#_bookmark43)和[图76所](#_bookmark44)示配置中，用户可控制I2C总线上RESTART条件的生成。如果[IC\_DATA\_CMD](#_bookmark67)寄存器的位10（重启）置1且重启功能使能（IC\_RESTART\_EN=1），则在向从机写入或读取数据字节之前会生成一个RESTART。如果未启用重启功能，则生成STOP（停止）和START（启动），以代替RESTART（重启）。[图75](#_bookmark43)示出了在作为主发射机操作

*图75.主发送器-[IC\_DATA\_CMD](#_bookmark67)的重启位置1*

S

SR

SDA

一个6 一个5 一个4 一个3 一个2 的1 a0级 W确认D7 D6 D5 D4 D3 D2 D1 D0确认D 7 D6 D5 D4 D3 D2 D1 D0确认

一个6 一个5 一个4 一个3 一个2 的1 a0级 W Ack D7 D6

SCL

FIFO\_EMPTY

数据可用性触发总线上的START条件

Tx FIFO加载数据（本例中为写入数据

Tx FIFO中的下一个字节已设置RESTART位

由于Tx FIFO上的下一个字节已标记有RESTART位，因此主机发出RESTART并启动新传输

[图76](#_bookmark44)说明了相同的情况，但在作为主接收器操作

*图76.主接收器-[IC\_DATA\_CMD](#_bookmark67)的*

S

SR

SDA

一个6 一个5 一个4 一个3 一个2 的1 a0级 R Ack D7 D6 D5 D4 D3 D2 D1 D0确认D 7 D6 D5 D4 D3 D2 D1 D0 Nak

一个6 一个5 一个4 一个3 一个2 的1 a0级 R Ack D7 D6

SCL

FIFO\_EMPTY

命令可用性触发总线上的START条件

由于Tx FIFO上的下一个命令已标记有RESTART位，因此主机发出RESTART并启动新传输

Tx FIFO加载命令Tx FIFO中的Next命令主机发出NOT ACK，

(read在本例中的操作）作为接收器操作时，在重新启动之前需要重新启动位设置

[图77](#_bookmark45)显示了作为主发送器的操作，其中[IC\_DATA\_CMD](#_bookmark67)寄存器的停止位已设置且Tx FIFO不为空

*图77.主发送器-[IC\_DATA\_CMD](#_bookmark67)的停止位设置/发送FIFO不为空*

S

P S

SDA

一个6 一个5 一个4 一个3 一个2 的1 a0级 W确认D7 D6 D5 D4 D3 D2 D1 D0确认D 7 D6 D5 D4 D3 D2 D1 D0 Ack

一个6 一个5 一个4 一个3 一个2 的1 a0级 W确认D7 D6

SCL

FIFO\_EMPTY

数据可用性触发总线上的START条件

Tx FIFO加载数据（本例中为写入数据

一个字节（不是最后一个）因为STOP位设置在最后一个因为更多数据可用，从Tx FIFO弹出字节从Tx FIFO弹出，主Tx FIFO，新的传输与STOP位设置产生STOP条件 立即启动（提供

主设备被授权访问总线）

[图78](#_bookmark46)显示了作为主发送器的操作，其中允许加载到Tx FIFO中的第一个字节变为空，并设置重启位

*图78.主发送器-加载到Tx FIFO的第一个字节允许为空，重启位设置*

S

SR

SDA

一个6 一个5 一个4 一个3 一个2 的1 a0级 W确认D7 D6 D5 D4 D3 D2 D1 D0确认D 7 D6 D5 D4 D3 D2 D1 D0确认

一个6 一个5 一个4 一个3 一个2 的1 a0级 W确认D7 D6

SCL

FIFO\_EMPTY

数据可用性触发总线

由于从Tx FIFO弹出的最后一个字节上未设置STOP位，因此主机将SCL保持为低电平

主机发出RESTART并启动新的传输

Tx FIFO加载数据（本例中为写入数据

从Tx FIFO弹出的最后一个字节，STOP位未设置

发送FIFO加载了新命令

[图79](#_bookmark47)显示了作为主接收器的操作，其中[IC\_DATA\_CMD](#_bookmark67)寄存器的停止位已设置且Tx FIFO不为空

*图79.主接收器-[IC\_DATA\_CMD](#_bookmark67)的停止位置位/TxFIFO不为空*

S

P S

SDA

一个6 一个5 一个4 一个3 一个2 的1 a0级 R Ack D7 D6 D5 D4 D3 D2 D1 D0确认D 7 D6 D5 D4 D3 D2 D1 D0Nak

一个6 一个5 一个4 一个3 一个2 的1 a0级 R Ack D7 D6

SCL

FIFO\_EMPTY

命令可用性触发总线上的START条件

由于在从Tx FIFO弹出的最后一个命令上设置了STOP位，因此主机生成STOP条件

Tx FIFO加载命令（本例中为读操作

从Tx FIFO弹出一个命令（不是最后一个），并设置STOP位

由于Tx FIFO中有更多命令可用，因此会立即启动新的传输（前提是主机被授予访问总线的权限）

[图80](#_bookmark49)显示了作为主接收器的操作，其中允许Tx FIFO清空后加载的第一个命令，并设置重启位

*图80.主接收器-Tx FIFO允许清空/重启位设置后加载的第一个命令*

S

SR

SDA

一个6 一个5 一个4 一个3 一个2 的1 a0级 R Ack D7 D6 D5 D4 D3 D2 D1 D0确认D 7 D6 D5 D4 D3 D2 D1 D0

Nak

一个6 一个5 一个4 一个3 一个2 的1 a0级 R Ack D7 D6

SCL

FIFO\_EMPTY

命令可用性触发总线上的START条件

Tx FIFO加载命令（本例中为读操作

因为STOP位

未在上次主机问题重新启动时设置，

弹出的命令 启动来自Tx FIFO、主机的新传输

保持SCL低电平主机在重新启动前

作为接收器从Tx FIFO操作时弹出最后一个命令，

STOP位未设置加载到Tx FIFO中的下一个命令已设置RESTART位

发送FIFO加载了新命令

* + 1. 多主仲裁

DW\_apb\_i2c总线协议允许多个主机驻留在同一总线上。如果在同一I2C总线上有两个主机，如果两者都试图通过同时生成START条件来同时控制总线，则存在仲裁过程一旦主设备（例如微控制器）控制了总线，其他主设备就无法控制，直到第一个主设备发送STOP条件并将总线置于空闲状态。

仲裁发生在SDA线上，而SCL线是一条。当另一个主机发送0时发送1的主机仲裁失败的主机可以继续生成时钟，直到字节传输结束如果两个主设备都在寻址同一个从设备，则仲裁可以进入数据阶段。

一旦检测到它已失去对另一主机的仲裁，DW\_apb\_i2c将停止生成SCL（将禁用输出驱动器）。[图81](#_bookmark50)显示了两个主机在总线上进行仲裁时的时序

*图81.多主仲裁*



CLKA

MSB

‘1’

DATA1仲裁失败

匹配数据

DATA2

MSB

‘0’

SDA镜像DATA2

SDA

MSB

SCL

SDA与DATA 1 START条件

总线的控制由地址或主机代码以及竞争主机发送的数据决定，因此总线上没有中央主机，也没有任何优先级顺序

在下列情况之间不允许仲裁

* + - * RESTART条件和数据位
      * STOP条件和数据位
      * 重新启动条件和停止条件

**注意**

奴隶不参与仲裁程序。

* + 1. 时钟同步

当两个或多个主机试图同时在总线上传输信息时，它们必须仲裁并同步SCL时钟。所有主机都生成自己的时钟来传输消息。数据仅在SCL时钟的高电平期间有效时钟同步通过与SCL信号的有线AND连接来执行。当主机将SCL时钟转换为0时，主机开始计数SCL时钟的低电平时间，并在下一个时钟周期开始时将SCL时钟信号转换为1但是，如果另一个主机将SCL线保持为0，则主机进入高等待状态，直到SCL时钟线转换为1。

然后，所有主机计算其高电平时间，高电平时间最短的主机将SCL线转换为零。然后，主设备计算出它们的低时间，具有最长低时间的主设备迫使其他主设备进入高等待状态。因此，会生成同步SCL时钟，如[图82](#_bookmark52)所示。从机也可以将SCL线保持为低电平，以减慢I2C总线上的时序

*图82. 多主时钟同步*

等待状态

开始计数高周期

公司简介

SCL

低电平SCL转换重置所有CLK以开始计算其低电平周期

当所有CLK处于高电平状态

* + 1. 操作模式

本节提供有关操作模式的信息

**注意**

需要注意的是，DW\_apb\_i2c只应设置为作为I2C主机或I2C从机运行，而不能同时作为两者运行。这是通过确保[IC\_CON](#_bookmark64).IC\_SLAVE\_DISABLE和[IC\_CON](#_bookmark64).MASTER\_MODE从不分别设置为0和1来

* + - 1. 从属模式操作

本节讨论从机模式程序。

* + - * 1. 初始配置

要将DW\_apb\_i2c用作从设备，请执行以下步骤:

通过向[IC\_ENABLE](#_bookmark88). ENABLE写入“0”来禁用DW\_apb\_i2c

写入[IC\_SAR](#_bookmark66)寄存器（位9:0）以设置从机地址。这是DW\_apb\_i2c响应的地址

写入[IC\_CON](#_bookmark64)寄存器以指定支持哪种寻址类型（通过设置位3，7位或10位通过将“0”写入位6（IC\_SLAVE\_DISABLE）并将“0 "写入位0（MASTER\_MODE），使能仅从机模式下的DW\_apb\_i2c

**注意**

从机和主机不必使用相同类型的7位或10位地址进行编程。例如，从设备可以用7位寻址编程，主设备可以用10位寻址编程，反之亦然。

1. 通过向[IC\_ENABLE](#_bookmark88). ENABLE写入“1”来启用DW\_apb\_i2c

**注意**

根据所选的复位值，可能不需要第二步和第三步，因为可以配置复位值。例如，如果设备只作为主设备，则无需设置从设备地址，因为您可以配置DW\_apb\_i2c，使其在复位后禁用从设备，并在复位后启用主设备如果DW\_apb\_i2c禁用，则存储的值为静态值，无需重新编程

**警告**

建议仅当I2C总线空闲时，才使DW\_apb\_i2c从机退出复位。当总线上正在进行传输时，取消复位会导致用于同步SDA和SCL的内部同步触发器从复位值1切换到总线上的实际值这可能导致SDA从1切换到0，而SCL为1，从而导致DW\_apb\_i2c从机检测到错误的START条件。通过将DW\_apb\_i2c配置为IC\_SLAVE\_DISABLE = 1和MASTER\_MODE = 1，也可以避免这种情况 以便在复位后禁用从机接口。然后，在内部SDA和SCL与总线上的值同步后，可以通过编程[IC\_CON](#_bookmark64)[0] = 0和[IC\_CON](#_bookmark64)[6]= 0来使能它;重置解除断言后，这大约需要六个ic\_clk周期

* + - * 1. 单字节从发送器操作

当总线上的另一个I2C主器件对DW\_apb\_i2c进行寻址并请求数据时，DW\_apb\_i2c充当从发送器，并执行以下步骤

另一个I2C主器件使用与DW\_apb\_i2c的[IC\_SAR](#_bookmark66)寄存器中的从机地址匹配的地址发起I2C传输

DW\_apb\_i2c确认发送的地址并识别传输方向，以指示其作为从属发送器。

DW\_apb\_i2c置位RD\_REQ中断（[IC\_RAW\_INTR\_STAT](#_bookmark74)寄存器的第5位），并使SCL线保持它处于等待状态，直到软件响应。如果由于[IC\_INTR\_MASK](#_bookmark73).M\_RD\_REQ设置为零而屏蔽了RD\_REQ中断，则建议使用硬件和/或软件时序例程来指示CPU定期读取[IC\_RAW\_INTR\_STAT](#_bookmark74)寄存器。

指示[IC\_RAW\_INTR\_STAT](#_bookmark74).RD\_REQ被设置为1的读取必须被视为等同于RD\_REQ中断被断言。

然后，软件必须采取行动，以满足I2C传输。

所使用的时序间隔应该是DW\_apb\_i2c可以处理的最快SCL时钟周期的10倍左右例如，对于400kbps，定时间隔为25μs。

**注意**

此处建议使用值10，因为这大约是在I2C总线上传输单个字节数据所需的时间

1. 如果在接收读取请求之前，Tx FIFO中有任何剩余数据，则DW\_apb\_i2c置位TX\_ABRT中断（[IC\_RAW\_INTR\_STAT](#_bookmark74)寄存器的第6位），以从TX FIFO中清除旧数据如果由于[IC\_INTR\_MASK](#_bookmark73).M\_TX\_ABRT设置为零而屏蔽了TX\_ABRT中断，则建议重新使用时序例程（如上一步所述）或类似例程来读取[IC\_RAW\_INTR\_STAT](#_bookmark74)寄存器。

**注意**

由于每当发生TX\_ABRT事件时，DW\_apb\_i2c的Tx FIFO都会强制进入刷新/复位状态，因此软件必须在尝试写入Tx FIFO之前读取[IC\_REQ\_TX\_ABRT](#_bookmark82)寄存器，有关详细信息，请参见寄存器[IC\_RAW\_INTR\_STAT](#_bookmark74)

* 1. 指示位6（R\_TX\_ABRT）被置1的读取必须被视为等同于TX\_ABRT中断被置位。
  2. 无需软件采取进一步措施
  3. 所用的时序间隔应与上一步中[IC\_RAW\_INTR\_STAT](#_bookmark74).RD\_DATA寄存器的时序间隔相似
     1. 软件将待写入的数据写入[IC\_DATA\_CMD](#_bookmark67)寄存器（通过在位8中写入“0”
     2. 软件在继续之前必须清除[IC\_RAW\_INTR\_STAT](#_bookmark74)寄存器的RD\_ADRT和TX\_ABRT中断（分别为位5和位6）。如果RD\_RST和/或TX\_ABRT中断已被屏蔽，则当R\_RD\_RST或R\_TX\_ABRT位已被读为1时，[IC\_RAW\_INTR\_STAT](#_bookmark74)寄存器的清零已被执行
     3. DW\_apb\_i2c释放SCL并传输字节。
     4. 主机可通过发出RESTART条件保持I2C总线，或通过发出STOP条件释放总线。

**注意**

由于不支持读取传输，因此在超快模式下，单个字节的从属发送器操作不适用。

* + - * 1. 单个字节的从接收器操作

当总线上的另一个I2C主设备寻址DW\_apb\_i2c并发送数据时，DW\_apb\_i2c充当从接收器，并执行以下步骤

另一个I2C主器件使用与[IC\_SAR](#_bookmark66)寄存器中DW\_apb\_i2c从机地址匹配的地址发起I2C传输

DW\_apb\_i2c确认发送的地址并识别传输方向，以指示DW\_apb\_i2c作为从接收器。

DW\_apb\_i2c接收传输的字节并将其放入接收缓冲区。

**注意**

如果在按下一个字节时Rx FIFO已完全填满数据，则DW\_apb\_i2c从机将I2CSCL线保持为低电平，直到Rx FIFO有一些空间，然后继续下一个读取请求。

1. DW\_apb\_i2c置位RX\_FULL中断[IC\_RAW\_INTR\_STAT](#_bookmark74).RX\_FULL。如果由于IC[\_INTR\_MASK](#_bookmark73).M\_RX\_FULL寄存器设置为零或[IC\_TX\_TL](#_bookmark76)设置为大于零的值，RX\_FULL中断已被屏蔽，则建议执行定时例程（如[第4.3.10.1.2](#_bookmark53)所述），以便定期读取[IC\_STATUS](#_bookmark89)寄存器。[IC\_STATUS](#_bookmark89)寄存器的读取（位3（RFNE）置1）必须由软件视为RX\_FULL中断的等效断言。
2. 软件可从[IC\_DATA\_CMD](#_bookmark67)寄存器（位7:0）读取字节
3. 另一主设备可通过发出RESTART条件来保持I2C总线，或通过发出STOP条件来释放总线
   * + - 1. 批量传输

在标准I2C协议中，所有事务都是单字节事务，编程器通过将一个字节写入从机的TX FIFO来响应远程主机读取请求当从机（从发射机）收到来自远程主机（主接收机）的DW\_apb\_i2c旨在处理TX FIFO中的更多数据，以便后续读取请求可以获取该数据，而无需引发中断来获取更多数据。最后，这消除了在每次引发数据中断之间产生显著延迟的可能性，如果存在仅将一个条目放置在TX FIFO中的限制的此模式仅在DW\_apb\_i2c作为从变送器时出现如果远程主机确认从机发送器发送的数据，但从机的TX FIFO中没有数据，则DW\_apb\_i2c将I2CSCL线路保持为低电平，同时引发读请求中断（RD\_REQ），并等待数据写入TX FIFO，然后才能将其发送到远程主机。

如果由于[IC\_INTR\_STAT](#_bookmark72).R\_RD\_STAT设置为零而屏蔽RD\_RST中断，则建议使用时序例程激活[IC\_RAW\_INTR\_STAT](#_bookmark74)寄存器的周期性读取读取[IC\_RAW\_INTR\_STAT](#_bookmark74)并将第5位（RD\_REQ）设置为1时，必须将其视为本节中提到的RD\_REQ中断的等效该定时程序类似于[第4.3.10.1.2](#_bookmark53)中所述。

RD\_REQ中断在读请求时产生，与中断一样，在退出中断服务处理例程（ISR）时必须清除。ISR允许您将一个字节或多个字节写入Tx FIFO。在将这些字节传输到主机期间，如果主机确认最后一个字节，则从机必须再次提高如果编程器预先知道远程主机请求“n”字节的数据包，则当另一主机寻址DW\_apb\_i2c并请求数据时，Tx FIFO可以写入“n”字节，并且远程主机将其作为连续数据流比如说

只要远程主机确认发送的数据并且Tx FIFO中有可用数据，DW\_apb\_i2c从机就会继续向远程主机发送数据无需将SCL线路保持为低电平或再次发出RD\_CLK

如果远程主机要从DW\_apb\_i2c接收“n”个字节，但程序员将大于“n”的字节数写入Tx FIFO，则当从机完成发送请求的“n”个字节时，它会清除Tx FIFO并忽略任何多余的字节。

在本例中，DW\_apb\_i2c生成发送中止（TX\_ABRT）事件以指示Tx FIFO的清零在预期ACK/NACK时，如果接收到NACK，则远程主机具有其想要的所有数据此时，从机的状态机内会产生一个标志，以清除Tx FIFO中的剩余数据此标志被传送到FIFO所在的处理器总线时钟域，此时Tx FIFO的内容被清除

* + - 1. 主控模式操作

本节讨论主模式过程。

* + - * 1. 初始配置

要将DW\_apb\_i2c用作主设备，请执行以下步骤:

通过将0写入[IC\_ENABLE](#_bookmark88). ENABLE来禁用DW\_apb\_i2c

写入[IC\_CON](#_bookmark64)寄存器以设置支持的最大速度模式（位2:1）和DW\_apb\_i2c主机发起传输的所需速度确保位6（IC\_SLAVE\_DISABLE）写入“1”，位0（MASTER\_MODE）写入“1”。

注意:从机和主机不必使用相同类型的7位或10位地址进行编程例如，从设备可以用7位寻址编程，主设备可以用10位寻址编程，反之亦然。

1. 向[IC\_TAR](#_bookmark65)寄存器写入要寻址的I2C器件的地址（位9:0）。该寄存器还指示I2C是否将执行通用调用或START BYTE命令
2. 通过向[IC\_ENABLE](#_bookmark88). ENABLE写入1来启用DW\_apb\_i2c
3. 现在写入传输方向和要发送到[IC\_DATA\_CMD](#_bookmark67)寄存器的数据。如果在使能DW\_apb\_i2c之前写入[IC\_DATA\_CMD](#_bookmark67)寄存器，则数据和命令会丢失，因为DW\_apb\_i2c禁用时缓冲器保持清零状态此步骤在DW\_apb\_i2c上生成START条件和地址字节使能DW\_apb\_i2c且TX FIFO中有数据后，DW\_apb\_i2c开始读取数据。

**注意**

根据所选的复位值，可能不需要步骤2、3、4和5，因为复位值可以配置。存储的值是静态的，如果DW\_apb\_i2c被禁用，则不需要重新编程，传输方向和数据除外

* + - * 1. 主机发送和主机接收

DW\_apb\_i2c支持在读和写之间动态地来回切换。要发送数据，将待写入的数据写入I2C Rx/Tx数据缓冲器和命令寄存器（[IC\_DATA\_CMD](#_bookmark67)）的低位字节对于I2C写操作，CMD位[8]应写入为零随后，可以通过将“无关”写入[IC\_DATA\_CMD](#_bookmark67)寄存器的低位字节来发出读命令只要发送FIFO中存在命令，DW\_apb\_i2c主机就会继续发起传输如果发送FIFO变为空，则主机在完成当前传输后插入STOP条件

* 如果设置为1，则在完成当前传输后发出STOP条件
* 如果设置为零，则SCL保持低电平，直到下一个命令写入发送FIFO。更多详细信息，请参见[第4.3.7](#_bookmark39)。
  + - 1. 禁用DW\_apb\_i2c

添加寄存器[IC\_ENABLE\_STATUS](#_bookmark100)，以允许软件明确确定硬件何时完全关闭，以响应[IC\_ENABLE](#_bookmark88).ENABLE从1设置为0。

只需要监视一个寄存器，而不是监视两个寄存器（[IC\_STATUS](#_bookmark89)和[IC\_RAW\_INTR\_STAT](#_bookmark74)），这是DW\_apb\_i2c早期版本的要求

**注意**

仅当当前正在处理的命令（发生ic\_enable取消断言时）将STOP位设置为1时，才可禁用DW\_apb\_i2c主机。如果在未设置STOP位的情况下处理命令时尝试禁用DW\_apb\_i2c主机，DW\_apb\_i2c主机将继续保持活动状态，使SCL线路保持当DW\_apb\_i2c主机在未设置STOP位的情况下处理命令时，您可以发出ABORT（[IC\_ENABLE](#_bookmark88).ABORT）以放弃I2C总线，然后禁用DW\_apb\_i2c。

* + - * 1. 程序

定义一个定时器间隔（ti2c\_poll），其等于系统中使用的最高I2C传输速度的信令周期的10倍，例如，如果最高I2C传输模式为400 kbps，则此ti2c\_poll为25μs。

定义一个最大超时参数MAX\_T\_POLL\_MAX，以便在任何重复轮询操作超过此最大值时报告错误

执行阻止线程/进程/函数，阻止软件启动任何进一步的I2C主机事务，但允许完成任何挂起的传输

**注意**

如果DW\_apb\_i2c编程为仅作为I2C从机工作，则可以忽略此步骤

1. 变量POLL\_POLL初始化为零。
2. 将[IC\_ENABLE](#_bookmark88)寄存器的位0设置为0。
3. 读取[IC\_ENABLE\_STATUS](#_bookmark100)寄存器并测试IC\_EN位（位0）。将POLL\_NULL递增1。如果POLL\_T>= MAX\_T\_POLL\_T，则退出并返回相关的错误代码。
4. 如果[IC\_ENABLE\_STATUS](#_bookmark100)[0]为1，则休眠ti2c\_poll并继续进行上一步骤。否则，使用相关的成功代码退出
   * + 1. 中止I2C传输

[IC\_ENABLE](#_bookmark88)寄存器的ABORT控制位允许软件在完成Tx FIFO发出的传输命令为响应ABORT请求，控制器通过I2C总线发出STOP条件，随后进行Tx FIFO刷新。只有在主操作模式下才允许中止传输

* + - * 1. 程序

停止用新命令填充Tx FIFO（[IC\_DATA\_CMD](#_bookmark67)）

在DMA模式下工作时，通过将TDMAE设置为零来禁用发送DMA

将[IC\_ENABLE](#_bookmark88).ABORT设置为1。

等待M\_TX\_ABRT中断。

读取[IC\_TX\_ABRT\_SOURCE](#_bookmark93)寄存器，将源标识为ABRT\_USER\_ABRT。

* + 1. 突波抑制

DW\_apb\_i2c包含可编程尖峰抑制逻辑，符合SS/FS模式I2C总线规范的要求。此逻辑基于计数器，该计数器监视输入信号（SCL和SDA），检查它们在内部采样之前是否在预定数量的ic\_CLK周期内保持稳定。每个信号（SCL和SDA）都有一个单独的计数器ic\_clk周期数可由用户编程，并且应考虑ic\_clk的频率和相关尖峰长度规范来计算。每当其输入信号改变其值时，每个计数器被启动根据输入信号的行为，会出现以下情况之一

* + - * 输入信号保持不变，直到计数器达到其计数极限值。当这种情况发生时，信号的内部版本将更新为输入值，计数器将重置并停止。计数器不

重新启动，直到检测到输入信号的新变化

* + - * 在计数器达到其计数极限值之前，输入信号再次改变当这种情况发生时，计数器

复位并停止，但信号的内部版本不更新。计数器保持停止，直到检测到输入信号的新变化

[图83](#_bookmark54)中的时序图说明了上述行为

*图83.尖峰抑制示例*

恢复时钟

SCL

穗长计数器

0 1 2 3 0 1 2 3 4 5

0

内部滤波SCL

**注意**

SCL输入端有一个2级同步器，但为简单起见，[图83](#_bookmark54)中的时序图未包含此同步延迟。

I2C总线规范根据工作模式要求不同的最大尖峰长度-SS和FS为50 ns，因此需要此寄存器来存储所需的值

* + - * 寄存器[IC\_FS\_SPKLEN](#_bookmark101)保存SS和FS模式的最大尖峰长度

该寄存器为8位宽，可通过APB接口进行读写访问;但是，仅当DW\_apb\_i2c禁用时才能可以编程到这些寄存器中的最小值是1;尝试编程小于1的值将导致写入值1

这些寄存器的默认值是基于ic\_duration周期的100 ns值，因此应针对

RP 2040上使用的时间段为\_sys。

**注意**

* 由于[IC\_FS\_SPKLEN](#_bookmark101)寄存器可编程的最小值为1，因此低频ic\_CLK可能会超过尖峰长度规格。考虑10 MHz（100 ns）的简单示例

周期）IC\_A;在这种情况下，可以编程的最小尖峰长度是100 ns，这意味着抑制达到该长度的

* 标准同步逻辑（串联的两个触发器）在尖峰抑制逻辑的上游实现，并且不受尖峰长度寄存器的内容或尖峰操作的任何影响

抑制逻辑;这两个操作（同步和尖峰抑制）是完全独立的。由于SCL和SDA输入与ic\_clk异步，因此这些信号的采样中存在一个ic\_clk周期不确定性;也就是说，根据它们相对于ic\_clk上升沿的发生时间，相同原始长度的尖峰可能会显示差异采样后一个ic\_clk周期

* 尖峰抑制是对称的;也就是说，从0到1和从1到0的转换行为完全相同
  + 1. 快速模式加操作

在快速模式下，DW\_apb\_i2c允许扩展快速模式操作，以支持高达1000 kbps的速度。要使DW\_apb\_i2c在快速模式下运行，请在启动任何数据传输之前执行以下步骤

1. 设置ic\_frequency大于或等于32MHz（参见[第4.3.14.2.1](#_bookmark60)）。
2. 对于快速模式或快速模式+，将[IC\_CON](#_bookmark64)寄存器[2:1]设置为2'b10
3. 对[IC\_FS\_SCL\_LCNT](#_bookmark71)和[IC\_FS\_SCL\_HCNT](#_bookmark70)寄存器进行编程，以满足快速模式加SCL的要求（参见[第4.3.14](#_bookmark57)）。
4. 对[IC\_FS\_SPKLEN](#_bookmark101)寄存器进行编程，以抑制50 ns的最大尖峰
5. 对[IC\_SDA\_SETUP](#_bookmark98)寄存器进行编程，以满足最小数据设置时间（tSU; DAT）。
   * 1. 总线清除功能

DW\_apb\_i2c支持总线清除功能，该功能在时钟或数据线停留在低电平的不太可能事件期间提供数据SDA和时钟SCL线的优雅恢复

#### SDA线路卡在低恢复

在SDA线路卡在低电平的情况下，主机执行以下操作以恢复，如[图84](#_bookmark55)和[图85](#_bookmark56)所示:

* + - * 1. 主机最多发送9个时钟脉冲，以在这9个时钟内恢复总线低电平

时钟脉冲的数量将随着从机剩余发送的位数而变化由于最大位数为9，主机最多发送9个时钟脉冲，并允许从机恢复

主机尝试在SDA线路上置位逻辑1，并检查SDA是否恢复。如果SDA未恢复，它将继续发送最多9个SCL时钟。

* + - * 1. 如果SDA线在9个时钟脉冲内恢复，则主机将发送STOP以释放总线。
        2. 如果SDA线甚至在第九个时钟脉冲之后也没有恢复，则系统需要硬件复位。

*图84. 9*个SCL*时钟*的SDA恢复

恢复时钟

SCL SDA MST\_SDA

0 1 2 3 4 5 6 7 8 9

10

主驱动9个时钟以恢复SDA卡在低电平

*图85. 使用6*个SCL*时钟*进行SDA恢复

恢复时钟

SCL SDA MST\_SDA

0 1

2

3

4

5

6 7

主机驱动6个时钟以恢复SDA卡在低电平

#### SCL线卡在低电平

在不太可能发生的情况下（由于电路的电气故障），时钟（SCL）被卡在低电平，没有有效的方法来克服这个问题，但复位总线使用硬件复位信号。

* + 1. IC\_CLK**频率配置**

当DW\_apb\_i2c配置为标准（SS）、快速（FS）/增强型快速模式（FM+）时，必须在进行任何I2C总线事务之前设置 \*CNT寄存器，以确保正确的I/O时序。\*CNT寄存器包括:

* + - * [IC\_SS\_SCL\_HCNT](#_bookmark68)
      * [IC\_SS\_SCL\_LCNT](#_bookmark69)
      * [IC\_FS\_SCL\_HCNT](#_bookmark70)
      * [IC\_FS\_SCL\_LCNT](#_bookmark71)

**注意**

START、STOP和RESTART寄存器的tBUF时序和建立/保持时间使用对应速度模式的 \*HCNT/\*LCNT寄存器设置。

**注意**

如果使能DW\_apb\_i2c仅作为I2C从机工作，则无需对任何 \*CNT寄存器进行编程，因为这些寄存器仅用于确定作为I2C主机工作的SCL时序要求

[表449](#_bookmark58)列出了从 \*CNT编程寄存器获得I2C时序参数的方法

*449号。I2C时序参数的推导*

|  |  |  |  |
| --- | --- | --- | --- |
| 定时参数 | 符号 | 标准速度 | 快速/快速升级 |
| SCL时钟的低电平周期 | tLOW | [IC\_SS\_SCL\_LCNT](#_bookmark69) | [IC\_FS\_SCL\_LCNT](#_bookmark71) |
| SCL时钟的高电平周期 | 大腿 | [IC\_SS\_SCL\_HCNT](#_bookmark68) | [IC\_FS\_SCL\_HCNT](#_bookmark70) |
| 重复启动条件的建立时间 | tSU;STA | [IC\_SS\_SCL\_LCNT](#_bookmark69) | [IC\_FS\_SCL\_HCNT](#_bookmark70) |
| 保持时间（重复）START条件 \* | 甲状腺激素缺乏症 | [IC\_SS\_SCL\_HCNT](#_bookmark68) | [IC\_FS\_SCL\_HCNT](#_bookmark70) |
| STOP条件的设置时间 | 甲状腺素 | [IC\_SS\_SCL\_HCNT](#_bookmark68) | [IC\_FS\_SCL\_HCNT](#_bookmark70) |

\*CNT寄存器

|  |  |  |  |
| --- | --- | --- | --- |
| 定时参数 | 符号 | 标准速度 | 快速/快速升级 |
| STOP和START状态之间的总线空闲时间 | tBUF | [IC\_SS\_SCL\_LCNT](#_bookmark69) | [IC\_FS\_SCL\_LCNT](#_bookmark71) |
| 穗长 | tSP | [IC\_FS\_SPKLEN](#_bookmark101) | [IC\_FS\_SPKLEN](#_bookmark101) |
| 数据保持时间 | 去甲肾上腺素 | [IC\_SDA\_HOLD](#_bookmark92) | [IC\_SDA\_HOLD](#_bookmark92) |
| 数据建立时间 | 甲状腺素 | [IC\_SDA\_设置](#_bookmark98) | [IC\_SDA\_设置](#_bookmark98) |

* + - 1. SS、FS和FM+模式下的最小高和低计数

当DW\_apb\_i2c作为I2C主机运行时，在发送和接收传输中:

* + - * + [IC\_SS\_SCL\_LCNT](#_bookmark69)和[IC\_FS\_SCL\_LCNT](#_bookmark71)寄存器值必须大于[IC\_FS\_SPKLEN](#_bookmark101)+7。
        + [IC\_SS\_SCL\_HCNT](#_bookmark68)和[IC\_FS\_SCL\_HCNT](#_bookmark70)寄存器值必须大于[IC\_FS\_SPKLEN](#_bookmark101)+5。有关DW\_apb\_i2c高计数和低计数的详细信息如下:
        + \*\_LCNT寄存器IC\_\*\_SPKLEN +7的最小值是由于DW\_apb\_i2c在SCL的负沿之后驱动SDA所需的时间。
        + \*\_HCNT寄存器的IC\_\*\_SPKLEN +5的最小值取决于DW\_apb\_i2c在SCL高电平期间对SDA进行采样所需的时间。
        + DW\_apb\_i2c将一个周期加到编程的 \*\_LCNT值，以生成SCL的低周期

时钟;这是由于SCL低电平计数到（\*\_LCNT +1）的计数逻辑

* + - * + DW\_apb\_i2c将IC\_\*\_SPKLEN +7个周期与编程的 \*\_HCNT值相加，以生成SCL时钟的高电平周期;这是由于以下因素造成的:

SCL高电平的计数逻辑计数到（\*\_HCNT+1）。

应用于SCL线路的数字滤波引起SPKLEN +2ic\_CLK周期的延迟，其中SPKLEN为:

[IC\_FS\_SPKLEN](#_bookmark101)（如果组件在SS或FS中运行）

每当DW\_apb\_i2c将SCL从1驱动为0时（即SCL达到高电平时间），就会产生三个ic\_Vdd周期的内部逻辑延迟因此，DW\_apb\_i2c为

最短SCL高电平时间为13个ic\_CLK周期（6 + 1 + 3

+ 3）。

**注意**

DW\_apb\_i2c主机生成的SCL的总高电平时间和低电平时间也受SCL线路的上升时间和下降时间的影响，如[图86](#_bookmark59)中的插图和公式所示。应注意，SCL上升和下降时间参数会因外部因素而异，例如:

* IO驱动程序的特性
* 上拉电阻值
* SCL线路上的总电容，等等

这些特性超出DW\_apb\_i2c的控制范围

*图86.* SCL*上升时间和下降时间对生成*SCL的影响



离子交换树脂

IC\_IN\_A/SCL

HCNT + IC\_\*\_SPKLEN +7

LCNT +1

SCL SCL

上升时间下降时间上升时间

SCL\_High\_time =[（HCNT + IC\_\*\_SPKLEN +7）\* ic\_time]+ SCL\_Fall\_time SCL\_low\_time =[（LCNT +1）\* ic\_time]-SCL\_Fall\_time + SCL\_Rise\_time

* + - 1. **最小**IC\_CLK**频率**

本节介绍DW\_apb\_i2c支持的每种速度模式的最小ic\_frequency，以及相关的高计数值和低计数值。在从机模式下，需要对[IC\_SDA\_HOLD](#_bookmark92)（Thd;dat）和[IC\_SDA\_SETUP](#_bookmark98)（Tsu:dat）进行编程，以满足I2C协议时序要求。以下示例针对[IC\_FS\_SPKLEN](#_bookmark101)编程为2的情况。

* + - * 1. 标准模式（SM）、快速模式（FM）和快速模式增强版（FM+）

本节详细介绍如何推导DW\_apb\_i2c标准和快速模式的最小ic\_max值。虽然下面的方法显示了如何进行快速模式计算，但您也可以使用相同的方法进行标准模式和快速模式+的计算。

**注意**

以下计算不考虑SCL\_Rise\_time和SCL\_Fall\_time。

给定快速模式下最小DW\_apb\_i2cic\_dft值的条件和计算

快速模式的数据速率为400 kbps;意味着SCL周期为1/400 kHz = 2.5μs

最小hcnt值14作为种子值; IC\_HCNT\_FS = 14

方案最小SCL高电平和低电平时间:

MIN\_SCL\_LOWtime\_FS = 1300 ns

MIN\_SCL\_HIGH\_time\_FS = 600ns导出的公式:

SCL\_PERIOD\_FS/（IC\_HCNT\_FS + IC\_LCNT\_FS）= IC\_CLK\_PERIOD IC\_LCNT\_FS × IC\_CLK\_PERIOD = MIN\_SCL\_LOWtime\_FS

结合前面的等式，可以得到以下结果:

IC\_LCNT\_FS ×（SCL\_PERIOD\_FS/（IC\_LCNT\_FS + IC\_HCNT\_FS））= MIN\_SCL\_LOWtime\_FS

求解IC\_LCNT\_FS:

IC\_LCNT\_FS ×（2.5μs /（IC\_LCNT\_FS + 14））= 1.3μs

前面的等式给出:

IC\_LCNT\_FS =舍入（15.166）= 16

这些计算产生IC\_LCNT\_FS = 16和IC\_HCNT\_FS = 14，给出的ic\_FS值为:

2.5μs /（16 + 14）= 83.3ns = 12MHz

*450号 与高计数和低计数有关的ic\_bits*

测试这些结果表明，协议要求得到满足。

[表450](#_bookmark61)列出了具有高计数值和低计数值的所有模式的最小ic\_rst值

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 速度模式 | IC\_clkfreq（MHz） | IC\_\*\_SPKLEN的最小 | SCL**低电平时间（单位:`ic\_s）** | SCL**低编程值** | SCL**低电平时间** | SCL**高时间**  'ic\_bits's | SCL**高编程值** | SCL**高电平时间** |
| SS | 2.7 | 1 | 13 | 12 | 4.7微秒 | 14 | 6 | 5.2微秒 |
| FS | 12.0 | 1 | 16 | 15 | 1.33微秒 | 14 | 6 | 1.16微秒 |
| FM+ | 32 | 2 | 16 | 15 | 500NS | 16 | 7 | 500NS |

* IC\_\*\_SCL\_LCNT和IC\_\*\_SCL\_HCNT寄存器使用表45 0中的SCL低和高编程值进行编程，分别使用SCL低计数减1和SCL高计数减8计算。的

[表450](#_bookmark61)中的值基于IC\_SDA\_RX\_HOLD = 0。最大IC\_SDA\_RX\_HOLD值取决于

IC\_\*CNT寄存器处于主机模式。

* 为了计算考虑RC时序的HCNT和LCNT，使用以下公式:
  + IC\_HCNT\_\*=[（HCNT + IC\_\*\_SPKLEN +7）\*ic\_CLK]+ SCL\_Fall\_time
  + IC\_LCNT\_\*=[（LCNT +1）\*ic\_CLK]-SCL\_Fall\_time + SCL\_Rise\_time
    - 1. 计算高计数和低计数

以下计算显示了如何计算DW\_apb\_i2c中每个速度模式的SCL高和低计数为使计算有效，使用的ic\_frequency不得小于[表450](#_bookmark61)中规定的最小ic\_frequency。

默认ic\_clock周期值设置为100 ns，因此将根据此时钟计算每个速度模式的默认SCL高和低计数值这些值需要根据以下准则进行更新

计算设置适当SCL时钟高电平和低电平时间所需的适当ic\_CLK信号数的公式如下:

IC\_xCNT =（ROUNDUP（MIN\_SCL\_xxxtime\*OSCFREQ，0））

MIN\_SCL\_HIGH时间=最小高电平周期MIN\_SCL\_HIGH时间= 4000 ns（100 kbps），

600 ns（400 kbps），260 ns（1000 kbps），

MIN\_SCL\_LOWtime =最小低电平周期MIN\_SCL\_LOWtime = 4700 ns（100 kbps），

400kbps为1300 ns， 1000kbps为500 ns，

OSCFREQ = ic\_CLK时钟频率（Hz）。

举例来说，请注意

OSCFREQ = 100MHz

I2C模式=快速，400 kbps MIN\_SCL\_HIGH时间= 600 ns。MIN\_SCL\_LOW时间= 1300 ns。

IC\_xCNT =（ROUNDUP（MIN\_SCL\_HIGH\_LOWtime\*OSCFREQ，0））

IC\_HCNT =（RUNDUP（600ns \* 100MHz，0））IC\_HCNTSCL PERIOD = 60

IC\_LCNT =（ROUNDUP（1300ns \* 100MHz，0））IC\_LCNTSCL PERIOD = 130

实际MIN\_SCL\_HIGH时间= 60\*（1/100 MHz）= 600 ns实际MIN\_SCL\_LOW时间= 130\*（1/100 MHz）= 1300 ns

* + 1. DMA控制器接口

DW\_apb\_i2c具有内置DMA功能;它具有到DMA控制器的握手接口，用于请求和控制传输。APB总线用于执行与DMA之间的数据传输由于数据速率相对较低，DMA传输作为单次访问进行传输

* + - 1. 启用DMA控制器接口

要在DW\_apb\_i2c上启用DMA控制器接口，必须写入DMA控制寄存器（[IC\_DMA\_CR](#_bookmark95)）。将1写入[IC\_DMA\_CR](#_bookmark95)寄存器的TDMAE位字段可使能DW\_apb\_i2c发送握手接口。将1写入[IC\_DMA\_CR](#_bookmark95)寄存器的RDMAE位字段可使能DW\_apb\_i2c接收握手接口。

* + - 1. 操作概述

DMA控制器通过DW\_apb\_i2c发送或接收的数据项数量（传输计数）进行编程。

该传输在总线上被分成单个传输，每个传输由来自DW\_apb\_i2c的请求发起

例如，编程到DMA控制器中的传输计数为4。DMA传输由一系列四个单事务组成。如果DW\_apb\_i2c向该通道发出发送请求，则单个数据项被写入DW\_apb\_i2c TX FIFO。类似地，如果DW\_apb\_i2c向该通道发出接收请求，则从DW\_apb\_i2c RX FIFO读取单个在写入或读取所有四个数据项之前，必须向该DMA通道发出四个单独的

* + - 1. 水印级别

在DW\_apb\_i2c中，用于设置水印以允许DMA突发的寄存器不需要设置为除其复位值之外的任何值具体地，[IC\_DMA\_TDLR](#_bookmark96)和[IC\_DMA\_RDLR](#_bookmark97)可以保留为零的复位值这是因为，由于I2C带宽相对于系统带宽较低，而且DMA控制器也较低，因此只需要单次传输

通常在系统总线上具有最高优先级，因此通常将非常快地完成

* + 1. 寄存器的操作

451号。清零和设置寄存器

[表451](#_bookmark62)列出了DW\_apb\_i2c中断寄存器的工作原理及其设置和清除方式。某些位由硬件设置并由软件清除，而其他位由硬件设置并清除

|  |  |  |
| --- | --- | --- |
| 中断位字段 | 由硬件设置/由软件 | 由硬件设置和清除 |
| 重启\_检测 | Y | N |
| GEN\_CALL | Y | N |
| START\_DET | Y | N |
| STOP\_DET | Y | N |
| 活动 | Y | N |
| RX\_完成 | Y | N |
| TX\_ABRT | Y | N |
| 研发中心 | Y | N |
| TX\_空 | N | Y |
| 发送结束 | Y | N |
| RX\_FULL | N | Y |
| RX\_OVER | Y | N |
| RX\_UNDER | Y | N |

* + 1. 登记册一览表

I2C0和I2C1寄存器分别从基址0x40044000和0x40048000开始（在SDK中定义为I2C0\_BASE和I2C1\_BASE

**注意**

您可能会在I2C寄存器描述中看到对配置常量的引用;这些常量是**固定**值，在硬件设计时设置。其值的完整列表可在<https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2040/hardware_regs/include/hardware/regs/i2c.h>中找到

*452号 I2C寄存器列表*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [IC\_CON](#_bookmark64) | I2C控制寄存器 |
| 0x04 | [IC\_TAR](#_bookmark65) | I2C目标地址寄存器 |
| 0x08 | [IC\_SAR](#_bookmark66) | I2C从机地址寄存器 |
| 0x10 | [IC\_DATA\_CMD](#_bookmark67) | I2C Rx/Tx数据缓冲器和命令寄存器 |
| 0x14 | [IC\_SS\_SCL\_HCNT](#_bookmark68) | 标准速度I2C时钟SCL高计数寄存器 |
| 0x18 | [IC\_SS\_SCL\_LCNT](#_bookmark69) | 标准速度I2C时钟SCL低计数寄存器 |
| 0x1c | [IC\_FS\_SCL\_HCNT](#_bookmark70) | 快速模式或快速模式加I2C时钟SCL高计数寄存器 |
| 0x20 | [IC\_FS\_SCL\_LCNT](#_bookmark71) | 快速模式或快速模式加I2C时钟SCL低计数寄存器 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x2c | [IC\_INTR\_STAT](#_bookmark72) | I2C UART状态寄存器 |
| 0x30 | [IC\_INTR\_MASK](#_bookmark73) | I2C掩码寄存器 |
| 0x34 | [IC\_RAW\_INTR\_STAT](#_bookmark74) | I2C原始寄存器状态寄存器 |
| 0x38 | [IC\_RX\_TL](#_bookmark75) | I2C接收FIFO阈值寄存器 |
| 0x3c | [IC\_TX\_TL](#_bookmark76) | I2C发送FIFO阈值寄存器 |
| 0x40 | [IC\_IC\_INTR](#_bookmark77) | 清除组合和单个寄存器 |
| 0x44 | [IC\_RX\_UNDER](#_bookmark78) | 清除RX\_UNDER寄存器 |
| 0x48 | IC\_CLR\_RX\_OVER | 清除RX\_OVER寄存器 |
| 0x4c | IC\_CLR\_TX\_OVER | 清除TX\_OVER中断寄存器 |
| 0x50 | IC\_CLR\_RD\_REQ | 清除RD\_缓存寄存器 |
| 0x54 | [IC\_TX\_ABRT](#_bookmark82) | 清除TX\_ABRT寄存器 |
| 0x58 | IC\_CLR\_RX\_DONE | 清除RX\_DONE中断寄存器 |
| 0x5c | IC\_CLR\_ACTIVITY | 清除ACTIVITY寄存器 |
| 0x60 | IC\_CLR\_STOP\_DET | 清除STOP\_DET寄存器 |
| 0x64 | [IC\_START\_DET](#_bookmark86) | 清除START\_DET寄存器 |
| 0x68 | IC\_CLR\_GEN\_CALL | 清除GEN\_CALL寄存器 |
| 0x6c | [IC\_ENABLE](#_bookmark88) | I2C ENABLE寄存器 |
| 0x70 | [IC\_STATUS](#_bookmark89) | I2C状态寄存器 |
| 0x74 | [IC\_TXFLR](#_bookmark90) | I2C发送FIFO电平寄存器 |
| 0x78 | [IC\_RXFLR](#_bookmark91) | I2C接收FIFO电平寄存器 |
| 0x7c | [IC\_SDA\_HOLD](#_bookmark92) | I2C SDA保持时间长度寄存器 |
| 0x80 | [IC\_TX\_ABRT\_SOURCE](#_bookmark93) | I2C发送中止源寄存器 |
| 0x84 | [IC\_SLV\_DATA\_NACK\_ONLY](#_bookmark94) | 生成从机数据NACK寄存器 |
| 0x88 | [IC\_DMA\_CR](#_bookmark95) | DMA控制寄存器 |
| 0x8c | [IC\_DMA\_TDLR](#_bookmark96) | DMA发送数据电平寄存器 |
| 0x90 | [IC\_DMA\_RDLR](#_bookmark97) | DMA发送数据电平寄存器 |
| 0x94 | IC\_SDA\_SETUP | I2C SDA设置寄存器 |
| 0x98 | [IC\_ACK\_GENERAL\_CALL](#_bookmark99) | I2C ACK通用调用寄存器 |
| 0x9c | [IC\_ENABLE\_STATUS](#_bookmark100) | I2C使能状态寄存器 |
| 0xa0 | [IC\_FS\_SPKLEN](#_bookmark101) | I2C SS、FS或FM+尖峰抑制限值 |
| 0xa8 | IC\_CLR\_RESTART\_DET | 清除RESTART\_DET中断寄存器 |
| 0xf4 | [IC\_COMP\_PARAM\_1](#_bookmark103) | 元件参数寄存器1 |
| 0xf8 | [IC\_COMP\_](#_bookmark104)VERSION | I2C组件版本寄存器 |
| 0xfc | [IC\_COMP\_TYPE](#_bookmark105) | I2C元件类型寄存器 |

### [I2C](#_bookmark63):IC\_CON寄存器

453号IC\_CON寄存器

**偏移**:0x00

描述

I2C控制寄存器。仅当DW\_apb\_i2c禁用（对应于IC\_ENABLE[0]寄存器设为0）时，才能写入该寄存器其他时间的写入没有任何效果。

读/写访问:-位10是只读的。- 位11为只读-位16为只读-位17为只读-位18和19为只读。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:11 | Reserved. | - | - | - |
| 10 | STOP\_DET\_IF\_MASTER\_ACTIVE | 无论主机是否处于活动状态，主机都会发出STOP\_DET中断 | RO | 0x0 |
| 9 | RX\_FIFO\_FULL\_HLD\_CTRL | 该位控制当Rx FIFO物理上已满至RX\_BUFFER\_DEPTH时，DW\_apb\_i2c是否应保持总线，如IC\_RX\_FULL\_HLD\_BUS\_EN参数所述。  重置值:0x0。  0x0→ RX\_FIFO已满时溢出0x1→ RX\_FIFO已满时保持总线 | RW | 0x0 |
| 8 | TX\_EMPTY\_CTRL | 该位控制TX\_EMPTY中断的生成，如IC\_RAW\_INTR\_STAT寄存器所述  重置值:0x0。  0x0→ TX\_EMPTY中断的默认行为0x1→ TX\_EMPTY中断的受控生成 | RW | 0x0 |
| 7 | STOP\_DET\_IFADDRESSED | 在从机模式下:-1 'b1:仅在寻址时发出STOP\_DET中断- 1'b0:发出STOP\_DET，不管它是否被寻址。重置值:0x0  注:在一般调用地址期间，如果STOP\_DET\_IF\_ADDRESSED = 1'b1，即使从机通过生成ACK响应一般调用地址，该从机也仅当发送的地址与从机地址（SAR）匹配时，才会产生STOP\_DET中断。0x0→从机始终发出STOP\_DET intr  0x1→仅当寻址时，从机发出STOP\_DET intr | RW | 0x0 |
| 6 | IC\_SLAVE\_DISABLE | 此位控制I2C是否禁用其从机，这意味着一旦施加预置信号，此位即被设置，从机禁用。  如果该位设置（从机禁用），则DW\_apb\_i2c仅作为主机工作，不执行任何需要从机的操作。  注:软件应确保如果此位写入0，则位0也应写入0。  0x0→使能从机模式0x1→禁用从机模式 | RW | 0x1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 5 | IC\_RESTART\_EN | 确定在充当主设备时是否可以发送RESTART条件。某些较旧的从机不支持处理RESTART条件;但是，RESTART条件在多个DW\_apb\_i2c操作中使用  当RESTART被禁用时，主机被禁止执行以下功能:-发送START BYTE-执行任何高速模式操作-高速模式操作-在组合格式模式中执行方向改变-用10位地址执行读取操作通过替换RESTART条件，然后是STOP和随后的START条件，分割操作被分解为多个DW\_apb\_i2c传输。如果执行上述操作，将导致IC\_RAW\_INTR\_STAT寄存器的位6（TX\_ABRT）置1。  复位值:启用  0x0→禁用主机重启0x1→启用主机重启 | RW | 0x1 |
| 4 | IC\_10BITADDR\_MASTER | 控制DW\_apb\_i2c作为主机时是以7位还是10位寻址模式启动传输-0:7位寻址-1:10位寻址  0x0→主机7位寻址模式0x1→主机10位寻址模式 | RW | 0x0 |
| 3 | IC\_10BITADDR\_SLAVE | 作为从机时，该位控制DW\_apb\_i2c响应7位还是10位地址。-0:7位寻址。DW\_apb\_i2c忽略涉及10位寻址的事务;对于7位寻址，仅比较IC\_SAR寄存器的低7位-1:10位寻址。DW\_apb\_i2c仅响应与IC\_SAR寄存器的完整10位相匹配的10位寻址传输。  0x0→从机7位寻址0x1→从机10位寻址 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 2:1 | 速度 | 这些位控制DW\_apb\_i2c的运行速度;只有在主机模式下运行DW\_apb\_i2c时，其设置才相关硬件可防止软件编程的非法值。这些位也必须针对从机模式进行适当编程，因为它用于根据速度模式捕获尖峰滤波器的正确值  此寄存器只能使用1至IC\_MAX\_SPEED\_MODE范围内的值进行编程  1:标准模式（100 kbit/s）  2:快速模式（=400 kbit/s）或快速模式+（= 1000 kbit/s）  3:高速模式（3.4 Mbit/s）  注:当IC\_ULTRA\_FAST\_MODE=1时，此字段不适用  0x1→标准速度操作模式0x2→快速或快速+操作模式0x3→高速操作模式 | RW | 0x2 |
| 0 | 主模式 | 此位控制DW\_apb\_i2c主机是否使能。  注:软件应确保如果该位写入“1”，则第6位也应写入“1”。  0x0→禁用主机模式0x1→启用主机模式 | RW | 0x1 |

454号IC\_TAR寄存器

### [I2C](#_bookmark63):IC\_TAR寄存器

**偏移**:0x04

描述

I2C目标地址寄存器

该寄存器为12位宽，保留位31:12。仅当IC\_ENABLE[0]设为0时，才能写入该寄存器

注意事项:如果软件或应用程序知道DW\_apb\_i2c未将TAR地址用于Tx FIFO中的待处理命令，则即使Tx FIFO具有条目（IC\_STATUS[2]= 0），也可以更新TAR地址。- 如果DW\_apb\_i2c仅作为I2C从机使能，则无需对此寄存器执行任何写入操作。

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 11 | 特别 | 此位指示软件是否执行设备ID或通用调用或START BYTE命令。-0:忽略位10 GC\_OR\_START并正常使用IC\_TAR- 1:执行Device\_ID或GC\_OR\_START位中指定的特殊I2C命令复位值:0x 0  0x0→禁用GENERAL\_CALL或START\_BYTE传输的编程  0x 1→启用GENERAL\_CALL或START\_BYTE传输编程 | RW | 0x0 |
| 10 | GC\_OR\_START | 如果位11（SPECIAL）设置为1，位13（设备ID）设置为0，则该位指示DW\_apb\_i2c是否执行通用调用或START字节命令。- 0:通用调用地址-发出通用调用后，只能执行写入尝试发出读命令会导致IC\_RAW\_INTR\_STAT寄存器的位6（TX\_ABRT）置1。DW\_apb\_i2c保持在常规调用模式，直到SPECIAL位值（位11）被清除。-1:START BYTE重置值:0x 0  0x0→ GENERAL\_CALL字节传输0x1→ START字节传输 | RW | 0x0 |
| 9:0 | IC\_TAR | 这是任何主事务的目标地址。当传输一般呼叫时，这些位被忽略。为了生成一个START BYTE，CPU只需要向这些位写入  如果IC\_TAR和IC\_SAR相同，则存在FIFO，但FIFO在主设备和从设备之间共享，因此完全FIFO是不可行的。仅支持单向双工模式（单工），不支持双工。主设备不能向自己传输，它只能向从设备传输 | RW | 0x055 |

455号IC\_SAR寄存器

### [I2C](#_bookmark63):IC\_SAR寄存器

**偏移**:0x08

描述

I2C从机地址寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:10 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 9:0 | IC\_SAR | 当I2C作为从机工作时，IC\_SAR保存从机地址。对于7位寻址，仅使用IC\_SAR[6:0]  仅当I2C接口禁用时（对应于IC\_ENABLE[0]寄存器置0），才能写入该寄存器其他时间的写入没有任何效果。  注意:默认值不能是任何保留地址位置:即0x00到0x07，或0x78到0x7f。如果将IC\_SAR或IC\_TAR编程为保留值，则无法保证器件的正确工作有关这些保留值的完整列表，请参阅[表448](#_bookmark34) | RW | 0x055 |

456号。IC\_DATA\_CMD

寄存器

### [I2C](#_bookmark63):IC\_DATA\_CMD寄存器

**偏移**:0x10

描述

I2C Rx/Tx数据缓冲区和命令寄存器;这是CPU在填充TX FIFO时写入的寄存器，CPU在从RX FIFO检索字节时读取

寄存器的大小变化如下:

写:- IC\_EMPTYFIFO\_HOLD\_MASTER\_EN=1时为11位-IC\_EMPTYFIFO\_HOLD\_MASTER\_EN=0时为9位读取:-IC\_FIRST\_DATA\_BYTE\_STATUS = 1时为12位-IC\_FIRST\_DATA\_BYTE\_STATUS = 0时为8位注:为了使DW\_apb\_i2c继续确认读取，应为每个要接收的字节写入读取命令;否则DW\_apb\_i2c将停止确认。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - | - |
| 11 | 首个数据字节E | 指示在主机接收器或从机接收器模式下接收传输的地址阶段之后接收的第一个数据字节  重置值:0x0  注:如果APB\_DATA\_WIDTH=8，   1. 用户必须对IC\_DATA\_CMD执行两次APB读取才能获得11位状态 2. 为了读取11位，用户必须执行第一个数据字节读取[7:0]（偏移量0x10），然后执行第二个读取[15:8]（偏移量0x11），以了解11位的状态（上一次读取中接收的数据是否为第一个数据字节 3. 第11位是可选读取字段，如果对FIRST\_DATA\_BYTE状态不感兴趣，用户可以   0x0→接收到顺序数据字节  0x1→接收到非顺序数据字节 | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 10 | 重启 | 此位控制在发送或接收字节之前是否发出RESTART  1-如果IC\_RESTART\_EN为1，则在发送/接收数据之前发出RESTART（根据CMD的值），而不管传输方向是否从先前的命令发生变化;如果IC\_RESTART\_EN为 0，则改为发出STOP（停止），然后发出START（开始）。  0-如果IC\_RESTART\_EN为1，则仅当传输方向从上一个命令改变时才发出RESTART;如果IC\_RESTART\_EN为0，则改为发出STOP，然后发出START。  重置值:0x0  0x0→不要在此命令之前发出RESTART 0x1→在此命令之前发出RESTART | SC | 0x0 |
| 9 | 停止 | 此位控制在发送或接收字节后是否发出STOP  -1-STOP在此字节后发出，无论Tx FIFO是否为空。如果Tx FIFO不为空，则主机立即尝试通过发出START并仲裁总线来开始新的传输。- 0 -无论Tx FIFO是否为空，在此字节后都不会发出如果Tx FIFO不为空，则主机根据CMD位的值发送/接收数据字节，如果Tx FIFO为空，则主机保持SCL线为低电平并暂停总线，直到Tx FIFO中有新命令可用。重置值:0x0  0x0→在此命令后不发出STOP 0x1→在此命令后发出STOP | SC | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 8 | CMD | 此位控制是执行读取还是写入。当DW\_apb\_i2con作为从机时，此位不控制方向。当它作为主控时，它只控制方向  当命令输入TX FIFO时，此位区分写命令和读命令。在从属接收器模式下，此位为“无关”位，因为不需要写入此寄存器在从发送器模式下，“0”表示要发送IC\_DATA\_CMD中的数据。  对此位进行编程时，应记住以下几点:在发送General Call命令后尝试执行读取操作会如果在接收到RD\_ABRT中断后将“1”写入该位，则发生TX\_ABRT中断。  重置值:0x0  0x0→主机写命令0x1→主机读命令 | SC | 0x0 |
| 7:0 | DAT | 该寄存器包含要在I2C总线上发送或接收的数据。如果写入此寄存器并希望执行读取，DW\_apb\_i2c将忽略位7:0（DAT）但是，读取此寄存器时，这些位返回DW\_apb\_i2c接口上接收的数据值。  重置值:0x0 | RW | 0x00 |

457号。IC\_SS\_SCL\_HCNT

寄存器

### [I2C](#_bookmark63):IC\_SS\_SCL\_HCNT寄存器

**偏移**:0x14

描述

标准速度I2C时钟SCL高计数寄存器

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 15:0 | IC\_SS\_SCL\_HCNT | 此寄存器必须在任何I2C总线事务发生之前设置，以确保正确的I/O时序。此寄存器设置标准速度下的SCL时钟高电平周期计数有关更多信息，请参阅“IC\_CLK频率校准”。  仅当I2C接口禁用（对应于IC\_ENABLE[0]寄存器置0）时，才能写入该寄存器其他时间的写入没有任何效果。  最小有效值为6;硬件阻止写入小于此值的值，如果尝试，则会导致设置为6。对于APB\_DATA\_WIDTH = 8的设计，编程顺序对于确保DW\_apb\_i2c的正确操作非常重要必须首先对低位字节进行编程。然后对高位字节进行编程。  注:此寄存器的值不得设置为高于65525，因为DW\_apb\_i2c使用16位计数器在计数器达到IC\_SS\_SCL\_HCNT +10时标记I2C总线空闲状态 | RW | 0x0028 |

458号。IC\_SS\_SCL\_LCNT

寄存器

### [I2C](#_bookmark63):IC\_SS\_SCL\_LCNT寄存器

**偏移**:0x18

描述

标准速度I2C时钟SCL低计数寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - | - |
| 15:0 | IC\_SS\_SCL\_LCNT | 此寄存器必须在任何I2C总线事务发生之前设置，以确保正确的I/O时序。此寄存器设置标准速度下的SCL时钟低电平周期计数有关更多信息，请参阅“IC\_CLK频率配置”  仅当I2C接口禁用（对应于IC\_ENABLE[0]寄存器置0）时，才能写入该寄存器其他时间的写入没有任何效果。  最小有效值为8;硬件阻止写入小于此值的值，如果尝试写入，则会导致设置为对于APB\_DATA\_WIDTH = 8的设计，编程顺序对于确保DW\_apb\_i2c的正确操作非常重要必须先对低位字节进行编程，然后再对高位字节进行编程。 | RW | 0x002f |

### [I2C](#_bookmark63):IC\_FS\_SCL\_HCNT寄存器

**偏移**:0x1c

459号。IC\_FS\_SCL\_HCNT

寄存器

460号。IC\_FS\_SCL\_LCNT

寄存器

描述

快速模式或快速模式加I2C时钟SCL高计数寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - | - |
| 15:0 | IC\_FS\_SCL\_HCNT | 此寄存器必须在任何I2C总线事务发生之前设置，以确保正确的I/O时序。此寄存器设置快速模式或快速模式+的SCL时钟高电平周期计数它在高速模式下用于发送主机代码和启动字节或通用呼叫。有关更多信息，请参阅“IC\_CLK频率校准”。  如果IC\_MAX\_SPEED\_MODE = standard，则该寄存器消失并变为只读，仅当I2C接口禁用时（对应于IC\_ENABLE[0]寄存器置0），才能写入其他时间的写入没有任何效果。  最小有效值为6;硬件阻止写入小于此值的值，如果尝试，则会导致设置为6。对于APB\_DATA\_WIDTH == 8的设计，编程顺序对于确保DW\_apb\_i2c的正确操作非常重要必须首先对低位字节进行编程。然后对高位字节进行编程。 | RW | 0x0006 |

### [I2C](#_bookmark63):IC\_FS\_SCL\_LCNT寄存器

**偏移**:0x20

描述

快速模式或快速模式加I2C时钟SCL低计数寄存器

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 15:0 | IC\_FS\_SCL\_LCNT | 此寄存器必须在任何I2C总线事务发生之前设置，以确保正确的I/O时序。此寄存器设置SCL时钟低电平周期计数，以实现快速。它在高速模式下用于发送主机代码和启动字节或通用呼叫。有关更多信息，请参阅“IC\_CLK频率校准”。  如果IC\_MAX\_SPEED\_MODE = standard，则该寄存器消失并变为只读，返回0。  仅当I2C接口禁用时（对应于IC\_ENABLE[0]寄存器置0），才能写入该寄存器其他时间的写入没有任何效果。  最小有效值为8;硬件阻止写入小于此值的值，如果尝试，则会导致设置为对于APB\_DATA\_WIDTH = 8的设计，编程顺序对于确保DW\_apb\_i2c的正确操作非常重要。必须首先对低位字节进行编程。然后对高位字节进行编程。如果该值小于8，则计数值变为8。 | RW | 0x000d个字符 |

表461。IC\_INTR\_STAT

寄存器

### [I2C](#_bookmark63):IC\_INTR\_STAT寄存器

**偏移**:0x2c

描述

I2C UART状态寄存器

此寄存器中的每个位在IC\_INTR\_MASK寄存器中都有相应的掩码位。通过读取匹配的中断清除寄存器来清除这些位。IC\_RAW\_INTR\_STAT寄存器中提供这些位的未屏蔽原始版本

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:13 | Reserved. | - | - | - |
| 12 | R\_RESTART\_DET | 有关R\_RESTART\_DET位的详细说明，请参见IC\_RAW\_INTR\_STAT  重置值:0x0  0x0→ R\_RESTART\_DET中断无效0x1→ R\_RESTART\_DET中断有效 | RO | 0x0 |
| 11 | R\_GEN\_CALL | 有关R\_GEN\_CALL位的详细说明，请参见IC\_RAW\_INTR\_STAT  重置值:0x0  0x0→ R\_GEN\_CALL中断无效0x1→ R\_GEN\_CALL中断有效 | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 10 | R\_START\_DET | 有关R\_START\_DET位的详细说明，请参见IC\_RAW\_INTR\_STAT  重置值:0x0  0x0→ R\_START\_DET中断无效0x1→ R\_START\_DET中断有效 | RO | 0x0 |
| 9 | R\_STOP\_DET | 有关R\_STOP\_DET位的详细说明，请参见IC\_RAW\_INTR\_STAT  重置值:0x0  0x0→ R\_STOP\_DET中断无效0x1→ R\_STOP\_DET中断有效 | RO | 0x0 |
| 8 | R\_ACTIVITY | 有关R\_ACTIVITY位的详细说明，请参见IC\_RAW\_INTR\_STAT  重置值:0x0  0x0→ R\_ACTIVITY中断无效0x1→ R\_ACTIVITY中断有效 | RO | 0x0 |
| 7 | R\_RX\_完成 | 有关R\_RX\_DONE位的详细说明，请参见IC\_RAW\_INTR\_STAT  重置值:0x0  0x0→ R\_RX\_DONE中断处于非活动状态0x1→ R\_RX\_DONE中断处于活动状态 | RO | 0x0 |
| 6 | R\_TX\_ABRT | 有关R\_TX\_ABRT位的详细说明，请参见IC\_RAW\_INTR\_STAT  重置值:0x0  0x0→ R\_TX\_ABRT中断无效0x1→ R\_TX\_ABRT中断有效 | RO | 0x0 |
| 5 | 研发中心 | 有关R\_RD\_STANDARD位的详细说明，请参见IC\_RAW\_INTR\_STAT  重置值:0x0  0x0→ R\_RD\_REQ中断处于非活动状态0x1→ R\_RD\_REQ中断处于活动状态 | RO | 0x0 |
| 4 | R\_TX\_空 | 有关R\_TX\_EMPTY位的详细说明，请参见IC\_RAW\_INTR\_STAT  重置值:0x0  0x0 → R\_TX\_EMPTY中断无效0x1 → R\_TX\_EMPTY中断有效 | RO | 0x0 |
| 3 | R\_TX\_OVER | 有关R\_TX\_OVER位的详细说明，请参见IC\_RAW\_INTR\_STAT  重置值:0x0  0x0→ R\_TX\_OVER中断无效0x1→ R\_TX\_OVER中断有效 | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 2 | R\_RX\_FULL | 有关R\_RX\_FULL位的详细说明，请参见IC\_RAW\_INTR\_STAT  重置值:0x0  0x0→ R\_RX\_FULL中断处于非活动状态0x1→ R\_RX\_FULL中断处于活动状态 | RO | 0x0 |
| 1 | 接收结束 | 有关R\_RX\_OVER位的详细说明，请参见IC\_RAW\_INTR\_STAT  重置值:0x0  0x0→ R\_RX\_OVER中断无效0x1→ R\_RX\_OVER中断有效 | RO | 0x0 |
| 0 | R\_RX\_UNDER | 有关R\_RX\_UNDER位的详细说明，请参见IC\_RAW\_INTR\_STAT  重置值:0x0  0x0→ RX\_UNDER中断无效0x1→ RX\_UNDER中断有效 | RO | 0x0 |

462号。IC\_INTR\_MASK

寄存器

### [I2C](#_bookmark63):IC\_INTR\_MASK寄存器

**偏移**:0x30

描述

I2C掩码寄存器。

这些位屏蔽其相应的中断状态位。该寄存器为低电平有效;值0屏蔽中断，值1不屏蔽中断。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:13 | Reserved. | - | - | - |
| 12 | M\_RESTART\_DET | 该位屏蔽IC\_INTR\_STAT寄存器中的R\_RESTART\_DET中断  重置值:0x0  0x0→ RESTART\_DET中断被屏蔽0x1→ RESTART\_DET中断被解除屏蔽 | RW | 0x0 |
| 11 | M\_GEN\_CALL | 该位屏蔽IC\_INTR\_STAT寄存器中的R\_GEN\_CALL中断  复位值:0x1  0x0→ GEN\_CALL中断被屏蔽0x1→ GEN\_CALL中断未被屏蔽 | RW | 0x1 |
| 10 | M\_START\_DET | 该位屏蔽IC\_INTR\_STAT寄存器中的R\_START\_DET中断  重置值:0x0  0x0→屏蔽 START\_DET中断0x1→解除屏蔽START\_DET中断 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 9 | M\_STOP\_DET | 该位屏蔽IC\_INTR\_STAT寄存器中的R\_STOP\_DET中断  重置值:0x0  0x0→ STOP\_DET中断被屏蔽0x1→ STOP\_DET中断被解除屏蔽 | RW | 0x0 |
| 8 | M\_ACTIVITY | 该位屏蔽IC\_INTR\_STAT寄存器中的R\_ACTIVITY中断  重置值:0x0  0x0→ ACTIVITY中断被屏蔽0x1→ ACTIVITY中断未被屏蔽 | RW | 0x0 |
| 7 | M\_RX\_完成 | 该位屏蔽IC\_INTR\_STAT寄存器中的R\_RX\_DONE中断  重置值:0x1  0x0→ RX\_DONE中断被屏蔽0x1→ RX\_DONE中断被解除屏蔽 | RW | 0x1 |
| 6 | M\_TX\_ABRT | 该位屏蔽IC\_INTR\_STAT寄存器中的R\_TX\_ABRT中断  复位值:0x1  0x0→屏蔽TX\_ABORT中断0x1→取消屏蔽TX\_ABORT中断 | RW | 0x1 |
| 5 | M\_RD\_请求 | 该位屏蔽IC\_INTR\_STAT寄存器中的R\_RD\_CLK中断  重置值:0x1  0x0→ RD\_REQ中断被屏蔽0x1→ RD\_REQ中断被解除屏蔽 | RW | 0x1 |
| 4 | M\_TX\_空 | 该位屏蔽IC\_INTR\_STAT寄存器中的R\_TX\_EMPTY中断  复位值:0x1  0x0→屏蔽 TX\_EMPTY中断0x1 →取消屏蔽TX\_EMPTY中断 | RW | 0x1 |
| 3 | M\_TX\_OVER | 该位屏蔽IC\_INTR\_STAT寄存器中的R\_TX\_OVER中断  重置值:0x1  0x0→ TX\_OVER中断被屏蔽0x1→ TX\_OVER中断未被屏蔽 | RW | 0x1 |
| 2 | M\_RX\_FULL | 该位屏蔽IC\_INTR\_STAT寄存器中的R\_RX\_FULL中断  重置值:0x1  0x0→ RX\_FULL中断被屏蔽0x1→ RX\_FULL中断未被屏蔽 | RW | 0x1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 1 | M\_RX\_OVER | 该位屏蔽IC\_INTR\_STAT寄存器中的R\_RX\_OVER中断  重置值:0x1  0x0→ RX\_OVER中断被屏蔽0x1→ RX\_OVER中断未被屏蔽 | RW | 0x1 |
| 0 | M\_RX\_UNDER | 该位屏蔽IC\_INTR\_STAT寄存器中的R\_RX\_UNDER中断  重置值:0x1  0x0→ RX\_UNDER中断被屏蔽0x1→ RX\_UNDER中断被解除屏蔽 | RW | 0x1 |

表463. IC\_RAW\_INTR\_STAT

寄存器

### [I2C](#_bookmark63):IC\_RAW\_INTR\_STAT寄存器

**偏移**:0x34

描述

I2C原始寄存器状态寄存器

与IC\_INTR\_STAT寄存器不同，这些位未被屏蔽，因此始终显示DW\_apb\_i2c的真实状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:13 | Reserved. | - | - | - |
| 12 | 重启\_检测 | 指示当DW\_apb\_i2c在从机模式下工作且从机正在寻址时，I2C接口上是否出现RESTART条件仅当IC\_SLV\_RESTART\_DET\_EN=1时启用。  注:但是，在高速模式下或在START BYTE传输期间，根据I2C协议，RESTART位于地址字段之前。在这种情况下，当发出RESTART时，从机不是寻址的从机，因此DW\_apb\_i2c不会生成RESTART\_DET中断。  重置值:0x0  0x0→ RESTART\_DET中断无效0x1→ RESTART\_DET中断有效 | RO | 0x0 |
| 11 | GEN\_CALL | 仅当接收到一般呼叫地址并确认时置1它保持设置状态，直到通过禁用DW\_apb\_i2c或CPU读取IC\_GEN\_CALL寄存器的位0DW\_apb\_i2c将接收到的数据存储在Rx缓冲区中  重置值:0x0  0x0→ GEN\_CALL中断无效0x1→ GEN\_CALL中断有效 | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 10 | START\_DET | 指示I2C接口上是否出现START或RESTART条件，无论DW\_apb\_i2c是在从机模式还是主机模式下工作  重置值:0x0  0x0→ START\_DET中断无效0x1→ START\_DET中断有效 | RO | 0x0 |
| 9 | STOP\_DET | 指示I2C接口上是否出现STOP（停止）条件，无论DW\_apb\_i2c是在从机模式还是主机模式下工作。  在从机模式下:-如果IC\_CON[7]= 1 'b1（STOP\_DET\_IFADDRESSED），则仅在从机寻址时才会发出STOP\_DET中断。注意事项:在一般调用地址期间，如果STOP\_DET\_IF\_ADDRESSED=1'b1，则即使从机通过生成ACK来响应一般调用地址，该从机也不会发出STOP\_DET中断。仅当发送的地址与从机地址（SAR）匹配时，才会产生STOP\_DET中断。- 如果IC\_CON[7]=1'b0（STOP\_DET\_IFADDRESSED），则无论是否寻址，都会发出STOP\_DET中断。在主机模式下:-如果IC\_CON[10]= 1 'b1（STOP\_DET\_IF\_MASTER\_ACTIVE），则仅当主机处于活动状态时才会发出- 如果IC\_CON[10]= 1 'b 0（STOP\_DET\_IFADDRESSED），则无论主机是否处于活动状态，都将发出重置值:0x0  0x0→ STOP\_DET中断无效0x1→ STOP\_DET中断有效 | RO | 0x0 |
| 8 | 活动 | 此位捕获DW\_apb\_i2c活动并保持设置，直到其被清除。有四种方法可将其清除:-禁用DW\_apb\_i2c -读取IC\_ACTIVITY\_寄存器-读取IC\_ACTIVITY\_INTR寄存器-系统复位此位设置后，将保持设置状态，除非使用四种方法之一将其清除即使DW\_apb\_i2c模块空闲，该位也会保持置位，直到清零，表示总线上有活动  重置值:0x0  0x0→ RAW\_INTR\_ACTIVITY中断无效0x1→ RAW\_INTR\_ACTIVITY中断有效 | RO | 0x0 |
| 7 | RX\_完成 | 当DW\_apb\_i2c作为从发送器时，如果主机未确认发送的字节，则该位设置为1。这发生在传输的最后一个字节，表示传输完成。  重置值:0x0  0x0→ RX\_DONE中断处于非活动状态0x1→ RX\_DONE中断处于活动状态 | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 6 | TX\_ABRT | 此位表示作为I2C发送器的DW\_apb\_i2c是否无法完成对发送FIFO内容的预期操作。这种情况既可能发生在I2C主机上，也可能发生在I2C从机上，称为“传输中止”。当该位设置为1时，IC\_TX\_ABRT\_SOURCE寄存器指示发送中止发生的原因  注:每当IC\_TX\_ABRT\_SOURCE寄存器跟踪的任何事件导致传输中止时，DW\_apb\_i2c都会刷新/重置/清空TX\_FIFO和RX\_FIFOFIFO保持这种刷新状态，直到读取寄存器IC\_TX\_ABRT一旦执行此读取，Tx FIFO就可以从APB接口接收更多数据字节  重置值:0x0  0x0→ TX\_ABRT中断无效0x1→ TX\_ABRT中断有效 | RO | 0x0 |
| 5 | 研发中心 | 当DW\_apb\_i2c作为从机，而另一个I2C主机尝试从DW\_apb\_i2c读取数据时，该位置1。DW\_apb\_i2c将I2C总线保持在等待状态（SCL=0），直到此中断得到服务，这意味着从机已被请求传输数据的远程主机寻址。处理器必须响应此中断，然后将请求的数据写入IC\_DATA\_CMD寄存器。处理器刚读取IC\_CLK\_RD\_CLK寄存器后，此位即设为0  重置值:0x0  0x0→ RD\_REQ中断处于非活动状态0x1→ RD\_REQ中断处于活动状态 | RO | 0x0 |
| 4 | TX\_空 | TX\_EMPTY中断状态的行为因IC\_CON寄存器中的TX\_EMPTY\_CTRL选择而异。- TX\_EMPTY\_CTRL = 0时:当发送缓冲器等于或低于IC\_TX\_TL寄存器中设置的阈值时，- 当TX\_EMPTY\_CTRL = 1时:当发送缓冲器等于或低于IC\_TX\_TL寄存器中设置的阈值时，该位设置为1，并且完成了当缓冲区级别超过阈值时，硬件会自动清除该值。当IC\_ENABLE[0]设置为0时，TX FIFO被刷新并保持复位状态。TX FIFO看起来似乎没有数据，因此如果主状态机或从状态机中有活动，则该位设置为1。当不再有任何活动时，ic\_en=0，此位设为0。  重置值:0x0。  0x0→ TX\_EMPTY中断无效0x1→ TX\_EMPTY中断有效 | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 3 | 发送结束 | 发送期间，如果发送缓冲区已满IC\_TX\_BUFFER\_DEPTH，且处理器尝试通过写入IC\_DATA\_CMD寄存器来发出另一个I2C命令，则置1。当模块禁用时，此位保持其电平，直到主状态机或从  重置值:0x0  0x0→ TX\_OVER中断无效0x1→ TX\_OVER中断有效 | RO | 0x0 |
| 2 | RX\_FULL | 当接收缓冲区达到或超过IC\_RX\_TL寄存器中的RX\_TL阈值时，置1。当缓冲区电平低于阈值时，硬件会自动清除如果模块被禁用（IC\_ENABLE[0]=0），RX FIFO被刷新并保持复位状态;因此RX FIFO未满。因此，一旦IC\_ENABLE位0被编程为0，该位即被清0，而不管活动是否继续。  重置值:0x0  0x0→ RX\_FULL中断处于非活动状态0x1→ RX\_FULL中断处于活动状态 | RO | 0x0 |
| 1 | RX\_OVER | 如果接收缓冲区已完全填满IC\_RX\_BUFFER\_DEPTH且从外部I2C器件接收到额外字节，则置DW\_apb\_i2c对此进行确认，但FIFO已满后接收的任何数据字节如果模块被禁用（IC\_ENABLE[0]=0），则该位保持其电平，直到主状态机或从状态机进入空闲状态，当ic\_en变为0时，该中断被清除。  注:如果IC\_CON寄存器（RX\_FIFO\_FULL\_HLD\_CTRL）的位9被编程为高电平，则RX\_OVER中断永远不会发生，因为Rx FIFO永远不会溢出。  重置值:0x0  0x0→ RX\_OVER中断无效0x1→ RX\_OVER中断有效 | RO | 0x0 |
| 0 | RX\_UNDER | 如果处理器尝试通过读取IC\_DATA\_CMD寄存器来读取空接收缓冲区，则置1。如果模块被禁用（IC\_ENABLE[0]=0），则该位保持其电平，直到主状态机或从  重置值:0x0  0x0→ RX\_UNDER中断无效0x1→ RX\_UNDER中断有效 | RO | 0x0 |

### [I2C](#_bookmark63):IC\_RX\_TL寄存器

464号IC\_RX\_TL寄存器

表465. IC\_TX\_TL寄存器

466号。IC\_INTR寄存器

**偏移**:0x38

描述

I2C接收FIFO阈值寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | RX\_TL | 接收FIFO阈值电平。  控制触发RX\_FULL中断（IC\_RAW\_INTR\_STAT寄存器中的位2）的条目电平（或更高）。有效范围为0-255，附加限制是硬件不允许将此值设置为大于缓冲区深度如果尝试这样做，则实际设置的值将是缓冲区的最大值为0设置1个条目的阈值，值为255设置256个条目的阈值 | RW | 0x00 |

[I2C](#_bookmark63):IC\_TX\_TL寄存器

**偏移**:0x3c

描述

I2C发送FIFO阈值寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | TX\_TL | 发送FIFO阈值电平。  控制触发TX\_EMPTY中断（IC\_RAW\_INTR\_STAT寄存器中的位4）的条目电平（或更低）。有效范围为0-255，但有一个附加限制，即该值不得设置为大于缓冲区深度的值如果尝试这样做，则实际设置的值将是缓冲区的最大深度值0设置0个条目的阈值，值255设置255个条目的阈值 | RW | 0x00 |

### [I2C](#_bookmark63):IC\_INTR寄存器

**偏移**:0x40

描述

清除组合和单个寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 0 | CLR\_INTR | 读取此寄存器以清除组合中断、所有单个中断和IC\_TX\_ABRT\_SOURCE寄存器。此位不清除硬件可清除中断，但清除软件可清除中断。有关清除IC\_TX\_ABRT\_SOURCE的异常，请参阅IC\_TX\_ABRT\_SOURCE寄存器的位  重置值:0x0 | RO | 0x0 |

467号。IC\_RX\_UNDER

寄存器

468号。IC\_切换\_接收\_结束

寄存器

### [I2C](#_bookmark63):IC\_RX\_UNDER寄存器

**偏移**:0x44

描述

清除RX\_UNDER寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | CLR\_RX\_UNDER | 读取此寄存器以清除IC\_RAW\_INTR\_STAT寄存器的RX\_UNDER中断（位  重置值:0x0 | RO | 0x0 |

[I2C](#_bookmark63):IC\_RX\_OVER寄存器

**偏移**:0x48

描述

清除RX\_OVER寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | CLR\_RX\_OVER | 读取此寄存器以清除IC\_RAW\_INTR\_STAT寄存器的RX\_OVER中断（位1）  重置值:0x0 | RO | 0x0 |

[I2C](#_bookmark63):IC\_TX\_OVER寄存器

**偏移**:0x4c

描述

清除TX\_OVER寄存器

*469号。IC\_发送\_结束*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | CLR\_TX\_OVER | 读取此寄存器以清除IC\_RAW\_INTR\_STAT寄存器的TX\_OVER中断（位3）  重置值:0x0 | RO | 0x0 |

寄存器

470号。IC\_芯片\_研发\_芯片

寄存器

471号表。IC\_TX\_ABRT

寄存器

### [I2C](#_bookmark63):IC\_RD\_R001寄存器

**偏移**:0x50

描述

清除RD\_缓存寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | CLR\_RD\_REQ | 读取此寄存器以清除IC\_RAW\_INTR\_STAT寄存器的RD\_UART中断（位5）  重置值:0x0 | RO | 0x0 |

[I2C](#_bookmark63):IC\_TX\_ABRT寄存器

**偏移**:0x54

描述

清除TX\_ABRT寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | CLR\_TX\_ABRT | 读取此寄存器以清除IC\_RAW\_INTR\_STAT寄存器和IC\_TX\_ABRT\_SOURCE寄存器的TX\_ABRT中断（位6）这也会将TX FIFO从刷新/复位状态释放，从而允许向TX FIFO进行更多写入有关清除IC\_TX\_ABRT\_SOURCE的异常，请参阅IC\_TX\_ABRT\_SOURCE寄存器的位9。  重置值:0x0 | RO | 0x0 |

[I2C](#_bookmark63):IC\_RX\_DONE寄存器

**偏移**:0x58

描述

清除RX\_DONE寄存器

*472号。IC\_接收\_完成*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | CLR\_RX\_DONE | 读取此寄存器以清除IC\_RAW\_INTR\_STAT寄存器的RX\_DONE中断（位7）。  重置值:0x0 | RO | 0x0 |

寄存器

473号。IC\_测试\_活动

寄存器

474号。IC\_停止\_检测

寄存器

### [I2C](#_bookmark63):IC\_ACTIVITY寄存器

**偏移**:0x5c

描述

清除ACTIVITY寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | CLR\_ACTIVITY | 如果I2C不再有效，则读取此寄存器将清除ACTIVITY中断。如果总线上的I2C模块仍处于活动状态，则ACTIVITY中断位继续置位。如果模块被禁用，并且总线上没有进一步的活动，则从该寄存器读取的值，用于获取IC\_RAW\_INTR\_STAT寄存器的ACTIVITY中断（位8）的状态  重置值:0x0 | RO | 0x0 |

[I2C](#_bookmark63):IC\_STOP\_DET寄存器

**偏移**:0x60

描述

清除STOP\_DET寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | CLR\_STOP\_DET | 读取此寄存器以清除IC\_RAW\_INTR\_STAT寄存器的STOP\_DET中断（位  重置值:0x0 | RO | 0x0 |

### [I2C](#_bookmark63):IC\_START\_DET寄存器

**偏移**:0x64

描述

清除START\_DET寄存器

*表475. IC\_START\_DET*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | CLR\_START\_DET | 读取此寄存器以清除IC\_RAW\_INTR\_STAT寄存器的START\_DET中断（位10）  重置值:0x0 | RO | 0x0 |

寄存器

476号。IC\_触发\_生成\_调用

寄存器

477号IC\_ENABLE

寄存器

### [I2C](#_bookmark63):IC\_CHARACTERISTIC\_GEN\_CALL寄存器

**偏移**:0x68

描述

清除GEN\_CALL寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | CLR\_GEN\_CALL | 读取此寄存器以清除IC\_RAW\_INTR\_STAT寄存器的GEN\_CALL中断（位  重置值:0x0 | RO | 0x0 |

### [I2C](#_bookmark63):IC\_ENABLE寄存器

**偏移**:0x6c

描述

I2C使能寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:3 | Reserved. | - | - | - |
| 2 | TX\_CMD\_BLOCK | 在主机模式下:-1 'b1:阻止I2C总线上的数据传输- 1'b0:一旦Tx FIFO中的第一个数据可用，则注:要阻止主机命令的执行，仅当Tx FIFO为空（IC\_STATUS[2]==1）且主机处于空闲状态（IC\_STATUS[5]== 0）时，才设置TX\_CMD\_BLOCK位。在TX\_CMD\_BLOCK位未置位之前，不会执行放入Tx FIFO的任何其他命令复位值:IC\_TX\_CMD\_BLOCK\_DEFAULT  0x0→ Tx命令执行未被阻止0x1→ Tx命令执行被阻止 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 1 | ABORT | 设置后，控制器启动传输中止。-0:ABORT未启动或ABORT已完成-1:ABORT操作正在进行软件可通过设置此位来中止主机模式只有在ENABLE已设置时，软件才能设置一旦设置，软件无法清除ABORT位。为响应ABORT，控制器在完成当前传输后发出STOP并刷新Tx FIFO，然后在中止操作后设置TX\_ABORT中断中止操作后，ABORT位自动清零  有关如何中止I2C传输的详细说明，请参阅“中止I2C传输”。  重置值:0x0  0x0→ ABORT操作未进行0x1→ ABORT操作正在进行 | RW | 0x0 |
| 0 | ENABLE | 控制是否启用DW\_apb\_i2c-0:禁用DW\_apb\_i2c（TX和RX FIFO保持在擦除状态）  - 1:启用DW\_apb\_i2c软件可以在DW\_apb\_i2c处于活动状态时禁用它。但是，务必注意确保正确禁用DW\_apb\_i2c。“禁用DW\_apb\_i2c”中介绍了推荐的过程。  禁用DW\_apb\_i2c时，会发生以下情况:-TX FIFO和RX FIFO被刷新。- IC\_INTR\_STAT寄存器中的状态位仍然有效，直到DW\_apb\_i2c进入IDLE状态。如果模块正在传输，则在当前传输完成后停止并删除传输缓冲区的内容如果模块正在接收，DW\_apb\_i2c将在当前字节结束时停止当前传输，并且不确认传输。  在IC\_CLK\_TYPE参数设置为asynchronous（1）时具有异步pCLK和ic\_CLK的系统中，使能或禁用DW\_apb\_i2c时会有两个ic\_CLK延迟。有关如何禁用DW\_apb\_i2c的详细说明，请参阅“禁用DW\_apb\_i2c”  重置值:0x0  0x 0→ I2C被禁用0x 1→ I2C被启用 | RW | 0x0 |

### [I2C](#_bookmark63):IC\_STATUS寄存器

**偏移**:0x70

描述

I2C状态寄存器

这是一个只读寄存器，用于指示当前传输状态和FIFO状态。可以随时读取状态寄存器此寄存器中的任何位都不请求中断。

478号IC\_STATUS

寄存器

当通过在IC\_ENABLE寄存器的位0中写入0来禁用I2C时:-位1和2被设置为1-位3和10被设置为0当主状态机或从状态机变为空闲并且ic\_en=0时:-位5和6被设置为0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:7 | Reserved. | - | - | - |
| 6 | SLV\_ACTIVITY | 从属FSM活动状态。当从机有限状态机（FSM）不处于IDLE状态时，该位置1。 -0:从FSM处于空闲状态，因此DW\_apb\_i2c的从部分不活动- 1:从FSM不处于空闲状态，因此DW\_apb\_i2c的从部分活动复位值:0x 0  0x0→从机空闲0x1→从机未空闲 | RO | 0x0 |
| 5 | MST\_ACTIVITY | 主状态机活动状态。当主有限状态机（FSM）不处于IDLE状态时，该位置1。-0:主FSM处于IDLE状态，因此DW\_apb\_i2c的主部分不活动-1:主FSM未处于IDLE状态，因此DW\_apb\_i2c的主部分活动注:IC\_STATUS[0]-即ACTIVITY位-是SLV\_ACTIVITY和MST\_ACTIVITY位的OR。  复位值:0x0 0x0→主机空闲  0x1→主机未空闲 | RO | 0x0 |
| 4 | RFF | 接收FIFO完全满。当接收FIFO完全满时，该位置1。当接收FIFO包含一个或多个空位置时，该位被清0。- 0:接收FIFO未满-1:接收FIFO已满复位值:0x 0  0x0→ Rx FIFO未满0x1→ Rx FIFO已满 | RO | 0x0 |
| 3 | RFNE | 接收FIFO不为空。当接收FIFO包含一个或多个条目时，该位置1;当接收FIFO为空时，该位清0- 0:接收FIFO为空- 1:接收FIFO不为空复位值:0x 0  0x0→ Rx FIFO为空0x1→ Rx FIFO不为空 | RO | 0x0 |
| 2 | TFE | 发送FIFO完全为空。当发送FIFO完全为空时，该位置1。当它包含一个或多个有效条目时，该位被清0。此位字段不请求中断。-0:发送FIFO不为空- 1:发送FIFO为空复位值:0x 1  0x0→ Tx FIFO不为空0x1→ Tx FIFO为空 | RO | 0x1 |
| 1 | TFNF | 传输FIFO未满。当发送FIFO包含一个或多个空位置时置1，当FIFO已满时清0。-0:发送FIFO已满-1:发送FIFO未满复位值:0x 1  0x0→ Tx FIFO已满0x1→ Tx FIFO未满 | RO | 0x1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 0 | ACTIVITY | I2C活动状态。复位值:0x0 0x0→ I2C空闲  0x1→ I2C有效 | RO | 0x0 |

479号。IC\_TXFLR

寄存器

480号。IC\_RXFLR

寄存器

### [I2C](#_bookmark63):IC\_TXFLR寄存器

**偏移**:0x74

描述

I2C发送FIFO电平寄存器此寄存器包含发送FIFO缓冲器中的有效数据条目数在以下情况下清除:- I2C禁用-传输中止-即，IC\_RAW\_INTR\_STAT寄存器中的TX\_ABRT位被置位-从机批量传输模式中止。每当将数据放入传输FIFO时，该寄存器递增;当从传输FIFO中取出数据时，该寄存器递减。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:5 | Reserved. | - | - | - |
| 4:0 | TXFLR | 发送FIFO电平。包含发送FIFO中的有效数据条目数  重置值:0x0 | RO | 0x00 |

### [I2C](#_bookmark63):IC\_RXFLR寄存器

**偏移**:0x78

描述

I2C接收FIFO电平寄存器此寄存器包含接收FIFO缓冲器中有效数据条目的数量当以下情况发生时，该寄存器被清除:-I2C被禁用-当IC\_TX\_ABRT\_SOURCE中跟踪的任何事件导致发送中止时，该寄存器在数据被放入接收FIFO时递增，在数据从接收FIFO中取出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:5 | Reserved. | - | - | - |
| 4:0 | RXFLR | 接收FIFO电平。包含接收FIFO中的有效数据条目数  重置值:0x0 | RO | 0x00 |

### [I2C](#_bookmark63):IC\_SDA\_HOLD寄存器

**偏移**:0x7c

描述

I2C SDA保持时间长度寄存器

此寄存器的位[15:0]用于控制从机和主机模式下发送期间（SCL从高电平变为低电平后）SDA的保持时间

在主机或从机模式下，接收器中的SCL为高电平时，此寄存器的位[23:16]用于延长SDA转换（如果有）

仅当IC\_ENABLE[0]=0时，才能成功写入此寄存器

此寄存器中的值以ic\_period为单位IC\_SDA\_TX\_HOLD中编程的值必须大于每个模式下的最小保持时间（主机模式下为一个周期，从机模式下为七个周期），以实现该值。

481号表。IC\_SDA\_HOLD

寄存器

482号。IC\_TX\_ABRT\_SOURCE

寄存器

发送期间的编程SDA保持时间（IC\_SDA\_TX\_HOLD）在任何时候都不能超过scl低电平部分的持续时间因此，编程值不能大于N\_SCL\_LOW-2，其中N\_SCL\_LOW是以ic\_CLK周期为单位测量的scl周期低电平部分的持续时间

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23:16 | IC\_SDA\_RX\_HOLD | 当DW\_apb\_i2c用作接收器时，以ic\_time周期为单位设置所需的SDA保持时间  复位值:IC\_DEFAULT\_SDA\_HOLD[23:16]。 | RW | 0x00 |
| 15:0 | IC\_SDA\_TX\_HOLD | 当DW\_apb\_i2c用作发射器时，以ic\_period为单位设置所需的SDA保持时间  复位值:IC\_DEFAULT\_SDA\_HOLD[15:0]。 | RW | 0x0001 |

### [I2C](#_bookmark63):IC\_TX\_ABRT\_SOURCE寄存器

**偏移**:0x80

描述

I2C发送中止源寄存器

该寄存器有32位，用于指示TX\_ABRT位的源除位9外，每当读取IC\_TX\_ABRT寄存器或IC\_TX\_INTR寄存器时，此寄存器将清0要清除位9，必须首先固定ABRT\_SBYTE\_NORSTRT的源;必须使能RESTART（IC\_CON[5]=1），必须清除SPECIAL位（IC\_TAR[11]），或必须清除GC\_OR\_START位（IC\_TAR[10]）。

一旦ABRT\_SBYTE\_NORSTRT的源固定，该位可以与该寄存器中的其他位相同的方式清0如果ABRT\_SBYTE\_NORSTRT的源在尝试清0此位之前未固定，则位9清0一个周期，然后重新置位。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:23 | TX\_FLUSH\_CNT | 此字段指示由于TX\_ABRT中断而刷新I2C禁用时，该值被清除  重置值:0x0  DW\_apb\_i2c的作用:主发送器或从发送器 | RO | 0x000 |
| 22:17 | Reserved. | - | - | - |
| 16 | ABRT\_USER\_ABRT | 这是一个仅用于主模式的位。主机已检测到传输中止（IC\_ENABLE[1]）  重置值:0x0  DW\_apb\_i2c的作用:主发送器  0x 0→主设备检测到传输中止-方案不存在  0x1→主机检测到传输中止 | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 15 | ABRT\_SLVRD\_INTX | 1:当处理器端响应从机模式请求将数据传输到远程主机时，用户在IC\_DATA\_CMD寄存器的CMD（位8）中写入1  重置值:0x0  DW\_apb\_i2c的作用:从发送器  0x 0→从机尝试在读取模式下传输到远程主机-场景不存在  0x1→从机尝试在读取模式下传输到远程主机 | RO | 0x0 |
| 14 | ABRT\_SLV\_ARBLOST | 此字段表示从机在向远程主机传输数据时丢失了总线  同时设置IC\_TX\_ABRT\_SOURCE[12]。注意:即使从设备从未“拥有”总线，总线上也可能出现问题。这是故障安全检查。例如，在SCL的低到高转换处的数据传输期间，如果数据总线上的不是应该传输的，则DW\_apb\_i2c不再拥有总线。  重置值:0x0  DW\_apb\_i2c的作用:从发送器  0x 0→从机丢失对远程主机的仲裁-场景不存在  0x1→从机失去对远程主机的仲裁 | RO | 0x0 |
| 13 | ABRT\_SLVFLUSH\_TXFIFO | 此字段指定从机已收到读命令，并且TX FIFO中存在某些数据，因此从机发出TX\_ABRT中断以刷新TX FIFO中的旧数据  重置值:0x0  DW\_apb\_i2c的作用:从发送器  0x 0→从机在获得读取命令后刷新TX-FIFO中的现有数据-场景不存在  0x 1→从机在获得读取命令后刷新TX-FIFO中的现有数据 | RO | 0x0 |
| 12 | ARB\_LOST | 此字段指定主机已失去仲裁，或者如果IC\_TX\_ABRT\_SOURCE[14]也被设置，则从机发射机已失去仲裁。  重置值:0x0  DW\_apb\_i2c的作用:主发送器或从发送器  0x 0→主机或从机-发送器丢失仲裁-场景不存在  0x 1→主机或从机-发送器丢失仲裁 | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 11 | ABRT\_MASTER\_DIS | 此字段表示用户尝试在禁用主模式的情况下启动主操作  重置值:0x0  DW\_apb\_i2c的作用:主发送器或主接收器  0x 0→当MASTER禁用时，用户启动主操作-场景不存在  0x1→当MASTER禁用时，用户启动主操作 | RO | 0x0 |
| 10 | ABRT\_10B\_RD\_NORSTRT | 此字段表示禁用重启（IC\_RESTART\_EN位（IC\_CON[5]）=0），主机在10位寻址模式下发送读取命令。  重置值:0x0  DW\_apb\_i2c的角色:主接收方  0x0→当RESTART禁用时，主机不尝试在10位寻址模式下读取  0x1→当RESTART禁用时，主机尝试在10位寻址模式下读取 | RO | 0x0 |
| 9 | ABRT\_SBYTE\_NORSTRT | 要清除位9，必须首先修复ABRT\_SBYTE\_NORSTRT的源;必须使能重启（IC\_CON[5]=1），必须清除SPECIAL位（IC\_TAR[11]），或必须清除GC\_OR\_START位（IC\_TAR[10]）。一旦ABRT\_SBYTE\_NORSTRT的源固定，该位可以与该寄存器中的其他位相同的方式清0如果ABRT\_SBYTE\_NORSTRT的源在尝试清除该位之前未固定，则位9会清除一个周期，然后重新置位。当此字段设置为1时，重启被禁用（IC\_RESTART\_EN位（IC\_CON[5]）=0），并且用户正在尝试发送START字节。  重置值:0x0  DW\_apb\_i2c的角色:主  0x 0→当RESTART禁用时，用户尝试发送START字节-场景不存在  0x1→当RESTART禁用时，用户尝试发送START字节 | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 8 | ABRT\_HS\_NORSTRT | 此字段表示重启已禁用（IC\_RESTART\_EN位（IC\_CON[5]）=0），并且用户正尝试使用主机在高速模式下传输数据  重置值:0x0  DW\_apb\_i2c的作用:主发送器或主接收器  0x 0→当RESTART禁用时，用户尝试将主机切换到HS模式-场景不存在  0x1→当RESTART禁用时，用户尝试将主机切换到HS模式 | RO | 0x0 |
| 7 | ABRT\_SBYTE\_ACKDET | 此字段表示主机已发送开始字节，且开始字节已确认（错误行为）。  重置值:0x0  DW\_apb\_i2c的角色:Master  0x 0→检测到START字节的ACK-场景不存在0x 1→检测到START字节的ACK | RO | 0x0 |
| 6 | ABRT\_HS\_ACKDET | 此字段表示主机处于高速模式，且高速主机代码已确认（错误行为）。  重置值:0x0  DW\_apb\_i2c的角色:Master  0x 0→ HS模式下确认HS主代码-场景不存在  0x1→ HS模式下确认的HS主代码 | RO | 0x0 |
| 5 | ABRT\_GCALL\_READ | 此字段表示主机模式下的DW\_apb\_i2c已发送通用调用，但用户将通用调用后的字节编程为从总线读取（IC\_DATA\_CMD[9]设置为1）。  重置值:0x0  DW\_apb\_i2c的作用:主发送器  0x 0→ GCALL之后是从总线读取-场景不存在  0x1→ GCALL之后是从总线读取 | RO | 0x0 |
| 4 | ABRT\_GCALL\_NOACK | 此字段表示主机模式下的DW\_apb\_i2c已发送通用调用，但总线上没有从机确认通用调用。  重置值:0x0  DW\_apb\_i2c的作用:主发送器  0x 0→ GCALL未被任何从机确认-方案不存在  0x1→ GCALL未被任何从机确认 | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 3 | ABRT\_TXDATA\_NOACK | 此字段表示仅主机模式位。当主机收到地址确认，但发送地址后的数据字节时，未收到来自远程从机的确认  重置值:0x0  DW\_apb\_i2c的作用:主发送器  0x 0→发送的数据未被寻址的从机确认-不存在这种  0x1→发送的数据未被寻址的从机确认 | RO | 0x0 |
| 2 | ABRT\_10ADDR2\_NOACK | 此字段表示主机处于10位地址模式，且10位地址的第二个地址字节未被任何从机确认  重置值:0x0  DW\_apb\_i2c的作用:主发送器或主接收器  0x0→未生成此中止  0x1→ 10位地址的字节2未被任何从机确认 | RO | 0x0 |
| 1 | ABRT\_10ADDR1\_NOACK | 此字段表示主机处于10位地址模式，且第一个10位地址字节未被任何从机确认。  重置值:0x0  DW\_apb\_i2c的作用:主发送器或主接收器  0x0→未生成此中止  0x1 → 10位地址的字节1未被任何从机确认 | RO | 0x0 |
| 0 | ABRT\_7B\_ADDR\_NOACK | 此字段表示主机处于7位寻址模式，并且发送的地址未得到任何从机的确认  重置值:0x0  DW\_apb\_i2c的作用:主发送器或主接收器  0x0→未生成此中止  0x1→由于7位地址的NOACK而产生此中止 | RO | 0x0 |

[I2C](#_bookmark63):IC\_SLV\_DATA\_NACK\_ONLY寄存器

**偏移**:0x84

描述

生成从机数据NACK寄存器

当DW\_apb\_i2c充当从接收器时，该寄存器用于为传输的数据部分生成NACK此寄存器仅在IC\_SLV\_DATA\_NACK\_ONLY参数设置为1时存在禁用此参数时，此寄存器不存在，写入寄存器地址无效。

483号。IC\_SLV\_DATA\_NACK\_

只登记

484号。IC\_DMA\_CR寄存器

485号。IC\_DMA\_TDLR

寄存器

如果同时满足以下两个条件，则可对此寄存器进行写操作:-DW\_apb\_i2c禁用（IC\_ENABLE[0]= 0）-从机部分处于非活动状态（IC\_STATUS[6] = 0）注:IC\_STATUS[6]是内部slv\_activity信号的寄存器回读位置;用户应在写入ic\_slv\_data\_nack\_only位之前对此进行轮询

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | Nack | 生成NACK。该NACK生成仅在DW\_apb\_i2c是从接收机时发生。如果此寄存器的值设置为1，则它只能在接收到数据字节后生成NACK;因此，数据传输中止，并且接收到的数据不会被推送到接收缓冲区。  当寄存器设置为0时，它会根据正常标准生成NACK/ACK。-0:正常生成NACK/ACK复位值:0x 0  0x0→从接收器正常生成NACK 0x1→从接收器仅在数据接收时生成NACK | RW | 0x0 |

### [I2C](#_bookmark63):IC\_DMA\_CR寄存器

**偏移量**:0x88

描述

DMA控制寄存器

该寄存器用于使能DMA控制器接口操作。有一个单独的位用于发送和接收。无论IC\_ENABLE的状态如何，都可以对其进行编程

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:2 | Reserved. | - | - | - |
| 1 | TDMAE | 发送DMA使能。该位使能/禁用发送FIFO DMA通道。重置值:0x0  0x0→发送FIFO DMA通道禁用0x1→发送FIFO DMA通道使能 | RW | 0x0 |
| 0 | RDMAE | 接收DMA启用。该位使能/禁用接收FIFO DMA通道。重置值:0x0  0x0→接收FIFO DMA通道禁用0x1→接收FIFO DMA通道使能 | RW | 0x0 |

### [I2C](#_bookmark63):IC\_DMA\_TDLR寄存器

**偏移**:0x8c

描述

DMA发送数据电平寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:4 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 3:0 | DMATDL | 传输数据电平。该位字段控制发送逻辑发出DMA请求的电平。它等于水印级别;也就是说，当发送FIFO中的有效数据条目数等于或低于此字段值且TDMAE =1时，生成dma\_tx\_req信号  重置值:0x0 | RW | 0x0 |

486号。IC\_DMA\_RDLR

寄存器

### [I2C](#_bookmark63):IC\_DMA\_RDLR寄存器

**偏移**:0x90

描述

I2C接收数据电平寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:4 | Reserved. | - | - | - |
| 3:0 | DMARDL | 接收数据级别。此位字段控制接收逻辑发出DMA请求的电平。水印级别= DMARDL+1;也就是说，当接收FIFO中的有效数据条目的数量等于或大于该字段值+1且RDMAE =1时，生成dma\_req\_req。例如，当DMARDL为0时，当接收FIFO中存在1个或多个数据条目时，dma\_req被断言  重置值:0x0 | RW | 0x0 |

### [I2C](#_bookmark63):IC\_SDA\_SETUP寄存器

**偏移**:0x94

描述

I2C SDA设置寄存器

当DW\_apb\_i2c在从发送器操作中处理读请求时，此寄存器控制SCL上升沿引入的时间延迟量（以ic\_PLL时钟周期数表示相关的I2C要求是tSU:DAT（注4），详见I2C总线规范。此寄存器必须设置为等于或大于2的值

仅当IC\_ENABLE[0]= 0时，才能成功写入此寄存器

注:设置时间的长度使用[（IC\_SDA\_SETUP-1）\*（ic\_period）]计算，因此如果用户需要10个ic\_period的设置时间，则应将值编程为11。IC\_SDA\_SETUP寄存器仅在DW\_apb\_i2c作为从发送器工作

*487号。IC\_SDA\_设置*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | SDA\_SETUP | SDA设置。建议如果所需延迟为1000 ns，则对于10 MHz的ic\_ADC频率，IC\_SDA\_SETUP应编程为值11。IC\_SDA\_SETUP必须设置为最小值2。 | RW | 0x64 |

寄存器

488号。IC\_ACK\_GENERAL\_CA

LL寄存器

489号。IC\_ENABLE\_STATUS

寄存器

### [I2C](#_bookmark63):IC\_ACK\_GENERAL\_CALL寄存器

**偏移**:0x98

描述

I2C ACK通用调用寄存器

该寄存器控制DW\_apb\_i2c在接收到I2C通用调用地址时是以ACK还是NACK进行响应此寄存器仅在DW\_apb\_i2c处于从机模式时适用

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | ACK\_GEN\_CALL | ACK通用呼叫。当设置为1时，DW\_apb\_i2c在接收到一般呼叫时以ACK响应（通过断言ic\_data\_oe）否则，DW\_apb\_i2c以NACK（通过否定ic\_data\_oe）进行响应。  0x0→为一般呼叫生成NACK 0x1→为一般呼叫生成ACK | RW | 0x1 |

### [I2C](#_bookmark63):IC\_ENABLE\_STATUS寄存器

**偏移**:0x9c

描述

I2C使能状态寄存器

当IC\_ENABLE[0]寄存器从1设置为0（即DW\_apb\_i2c禁用）时，该寄存器用于报告DW\_apb\_i2c硬件状态

如果IC\_ENABLE[0]已设置为1，则位2:1被强制为0，位0被强制为1。

如果IC\_ENABLE[0]已设置为0，则位2:1仅在位0读取为“0”时有效

注:IC\_ENABLE[0]设置为0时，位0读取为0会出现延迟，因为DW\_apb\_i2c的禁用取决于I2C总线活动。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:3 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 2 | SLV\_RX\_DATA\_LOST | 从机接收数据丢失。此位表示是否由于IC\_ENABLE的0位从1设置为0，从机接收器操作已中止，且至少有一个数据字节从I2C传输接收当读取为1时，DW\_apb\_i2c被视为已主动参与中止的I2C传输（具有匹配地址），并且已进入I2C传输的数据阶段，即使数据字节已响应NACK。  注:如果远程I2C主机在DW\_apb\_i2c有机会NACK传输之前以STOP条件终止传输，且IC\_ENABLE[0]已设为0，则此位也设为1。  当读取为0时，DW\_apb\_i2c被视为已被禁用，而没有主动参与从接收器传输的数据  注:当IC\_EN（位0）读为0时，CPU可以安全地读取此位  重置值:0x0  0x0→从RX数据未丢失0x1→从RX数据丢失 | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 1 | SLV\_DISABLED\_WHILE\_BUSY | 忙时禁用从机（发送、接收）。此位表示潜在或活动的从机操作是否因IC\_ENABLE寄存器的位0从 1设置为0而中止当CPU向IC\_ENABLE寄存器写入0时，该位置1，同时:   1. DW\_apb\_i2c正在从远程主机接收从机-发送器操作的地址字节   或者，   1. 从远程主机接收从机-接收机操作的地址和数据字节   当读取为1时，无论I2C地址是否与DW\_apb\_i2c（IC\_SAR寄存器）中设置的从机地址匹配，或者传输是否在IC\_ENABLE设置为0之前完成但尚未生效，DW\_apb\_i2c都被视为在I2C传输的任何部分期间强制NACK。  注:如果远程I2C主机在DW\_apb\_i2c有机会NACK传输之前以STOP条件终止传输，并且IC\_ENABLE[0]已设置为0，则该位也将设置为1。  当读取为0时，当主机活动或I2C总线空闲时，  注:当IC\_EN（位0）读为0时，CPU可以安全地读取此位  重置值:0x0  0x0→空闲时禁用从机0x1→活动时禁用从机 | RO | 0x0 |
| 0 | IC\_EN | ic\_en状态。此位始终反映输出端口ic\_en上驱动的值- 当读取为1时，DW\_apb\_i2c被视为处于启用状态。- 当读取为0时，DW\_apb\_i2c被视为完全不活动。注意:CPU可以随时安全地读取此位当此位读为0时，CPU可以安全地读取SLV\_RX\_DATA\_LOST（位2）和SLV\_DISABLED\_WHILE\_BUSY（位1）。  复位值:0x0 0x0→ I2C禁用0x1→ I2C使能 | RO | 0x0 |

### [I2C](#_bookmark63):IC\_FS\_SPKLEN寄存器

**偏移量**:0xa0

描述

I2C SS、FS或FM+尖峰抑制限值

表490。IC\_FS\_SPKLEN

寄存器

表491。IC\_重启\_检测

寄存器

表492。IC\_COMP\_PARAM\_1

寄存器

此寄存器用于存储器件在SS、FS或FM+模式下工作时由尖峰抑制逻辑滤除的最长尖峰的持续时间（以ic\_CLK周期为单位）相关的I2C要求是tSP（表4），详见I2C总线规范。此寄存器必须设置为最小值1。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | IC\_FS\_SPKLEN | 该寄存器必须在任何I2C总线事务发生之前设置，以确保稳定运行。此寄存器设置SCL或SDA线路中将由尖峰抑制逻辑滤除的最长尖峰的持续时间（以ic\_CLK周期为单位）仅当I2C接口禁用（对应于IC\_ENABLE[0]寄存器置0）时，才能写入该寄存器。其他时间的写入没有任何效果。最小有效值为1;硬件阻止写入小于此值的值，如果尝试，则会导致设置为1。或更多信息，请参阅“尖峰抑制”。 | RW | 0x07 |

### [I2C](#_bookmark63):IC\_RESTART\_DET寄存器

**偏移量**:0xa8

描述

清除RESTART\_DET寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | CLR\_RESTART\_DE T | 读取此寄存器以清除IC\_RAW\_INTR\_STAT寄存器的RESTART\_DET中断  重置值:0x0 | RO | 0x0 |

### [I2C](#_bookmark63):IC\_COMP\_PARAM\_1寄存器

**偏移量**:0xf4

描述

元件参数寄存器1

注:此寄存器未实现，因此读取为0。如果实现了它，它将是一个常量只读寄存器，包含有关组件参数设置的编码信息。下面显示的字段是这些参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23:16 | TX\_BUFFER\_DEPTH | TX缓冲区深度= 16 | RO | 0x00 |
| 15:8 | RX\_BUFFER\_DEPTH | RX缓冲区深度= 16 | RO | 0x00 |
| 7 | ADD\_ENCODED\_PARAMS | 编码参数不可见 | RO | 0x0 |
| 6 | HAS\_DMA | DMA握手信号使能 | RO | 0x0 |
| 5 | INTR\_IO | 组合式双输出 | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 4 | HC\_COUNT\_VALU ES | 每个模式的可编程计数值 | RO | 0x0 |
| 3:2 | MAX\_SPEED\_MODE | 最大速度模式=快速模式 | RO | 0x0 |
| 1:0 | APB\_DATA\_WIDTH | APB数据总线宽度为32位 | RO | 0x0 |

表493。IC\_COMP\_版本

寄存器

表494。IC\_COMP\_TYPE

寄存器

### [I2C](#_bookmark63):IC\_COMP\_VERSION寄存器

**偏移量**:0xf8

描述

I2C组件版本寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:0 | IC\_COMP\_VERSION |  | RO | 0x3230312a |

### [I2C](#_bookmark63):IC\_COMP\_TYPE寄存器

**偏移量**:0xfc

描述

I2C元件类型寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:0 | IC\_COMP\_TYPE | Designware组件类型编号= 0x44\_57\_01\_40。这个分配的唯一十六进制值是常量，并且是从两个ASCII字母'DW'后面跟一个16位无符号数字派生出来的。 | RO | 0x44570140 |

* 1. SPI

###### ARM文档

摘自[ARM PrimeCell同步串行端口（PL 022）技术参考手册](https://developer.arm.com/documentation/ddi0194/latest)。经允许使用。

RP 2040有两个相同的SPI控制器，都基于ARM Primecell同步串行端口（SSP）（PL 022）（版本r1p4）。请注意，这与[第4.10](#_bookmark225)中介绍的QSPI接口不同。

每个控制器都支持以下功能:

* + - 主模式或从模式
      * Motorola SPI兼容接口
      * Texas Instruments同步串行接口
      * National Semiconductor Microwire接口
    - 8个深Tx和Rx FIFO
    - 生成服务FIFO或指示错误条件的故障
    - 可从DMA驱动
    - 可编程时钟频率
    - 可编程数据大小4-16位

每个控制器都可以连接到第2.19.2GPIO多路复用表279中定义的多个GPIO引脚。GPIO多路复用连接的前缀为SPI实例名称spi0\_或spi1\_，包括以下内容

* + - clocksclk（当控制器在主机模式下工作时，连接到以下部分中的SSPCLKOUT，或当控制器在从机模式下工作时，连接到SSPCLKIN）
    - 低电平有效芯片选择或帧同步ss\_n（在以下部分中称为SSPFSSOUT
    - 传输数据tx（在以下部分中称为SSPTXD，注意nSSPOE未连接到tx

焊盘，因此输出数据不会被SPI控制器三态化

* + - 接收数据rd（在以下部分中称为SSPRXD

SPI TX引脚功能通过接线始终置位焊盘输出使能，不受nSSPOE驱动当多个SPI从机共享一条总线时，软件需要切换输出使能。这可以通过切换相关iobank0.ccumbs寄存器的oeover字段或切换GPIO功能来完成。

SPI使用clk\_peri作为SPI定时的参考时钟，在以下部分中称为SSPCLK

CLK\_sys用作总线时钟，在以下章节中称为PCLK（另见第2.15.1）。

* + 1. 概述

PrimeCell SSP是一种主或从接口，用于与具有Motorola SPI、National Semiconductor Microwire或Texas Instruments同步串行接口的外围设备进行同步串行通信。

PrimeCell SSP对从外围设备接收的数据执行串并转换CPU通过AMBA APB接口访问数据、控制和状态信息发送和接收路径通过内部FIFO存储器进行缓冲串行数据在SSPTXD上传输，在SSPRXD上接收

PrimeCell SSP包括一个可编程位速率时钟分频器和预分频器，用于从输入时钟SSPCLK生成串行输出时钟SSPCLKOUT。支持的比特率为2 MHz及更高，具体取决于SSPCLK的频率选择，最大比特率由外围设备确定

可以使用控制寄存器SSPCR 0和SSPCR 1对PrimeCell SSP工作模式、帧格式和大小进行编程。

生成以下可单独屏蔽的中断

* + - * SSPTXINTR请求为传输缓冲区提供
      * SSPRXINTR请求服务接收缓冲区
      * SSPRORINTR指示接收FIFO中的溢出条件
      * SSPRTINTR指示接收FIFO中存在数据时超时期限已到期

如果任何单独的中断被置位并解除屏蔽，则单个组合中断被置位该中断连接到RP 2040中的处理器中断控制器

除了上述中断之外，还提供一组DMA信号用于与DMA控制器接口

根据所选的工作模式，SSPFSSOUT输出的工作方式为:

* + - * 高电平有效帧同步输出，适用于Texas Instruments同步串行帧格式
      * SPI和Microwire的低电平有效从机选择
    1. 功能描述

*图87. PrimeCellSSP框图。为清楚起见，未显示测试逻辑。*

SSPTXINTR

预置n

[15:0]

SSPTXINTR

PSEL

PCLK

处罚

TX FIFO

16位宽，

8个位置深度

[15:0]

PWRITE

AMBAAPB

接口

FIFO状态和中断生成

SSPINTR

PADDR[11:2]

[15:0]

RxFRdData[15:0]

PRDATA[15:0]

PCLK

PCLK

RX FIFO

16位宽，

8个位置深度

SSPRXINTR SSPRORINTR

PCLK

DATAIN

DATAOUT

SSPRTRINTR

SSPRXRINTR

SSPCLK

PCLK

nSSP

寄存器块

SSPCLK

预缩放值

时钟预分频器

SSPCLK

Tx/Rx参数SSPCLKDIV

nSSPOE

SSPTXD SSPFSSOUT

SSPRXDMACLR 发送和

接收逻辑SSPCLKOUT

SSPTXDMACLR

Tx/Rx FIFO水印电平

nSSPCTLOE

SSPRXDMAS

SSPCLKIN

DMA

SSPRXDMABREQ接口

SSPFSSIN

SSPTXDMAS

SSPRXD

SSPTXDMABREQ

接收数据[15:0]

* + - 1. AMBA APB接口

AMBA APB接口生成读写解码，用于访问状态和控制寄存器以及发送和接收FIFO存储器。

* + - 1. 寄存器块

寄存器块存储通过AMBA APB接口写入或读取的数据

* + - 1. 时钟预分频器

当配置为主机时，内部预分频器（包括两个自由运行的可复位串行连接计数器）提供串行输出时钟SSPCLKOUT。

您可以使用SSPCPSR寄存器对时钟预分频器进行编程，使SSPCLK分频系数为2-254，分频步长为2。由于不利用SSPCPSR寄存器的最低有效位，因此无法进行奇数分频，从而确保生成对称、等占空比时钟。参见[SSPCPSR](#_bookmark129)。

通过对SSPCR 0控制寄存器进行编程，预分频器的输出再次除以1-256，得到最终的主输出时钟SSPCLKOUT。

**注意**

[图87](#_bookmark106)中的PCLK和SSPCLK时钟输入分别连接到RP 2040上的UART\_SYS和UART\_SYS系统级时钟网络默认情况下，SPI\_CLK直接连接到系统时钟，但如果系统时钟动态变化，则可以断开SPI\_CLK以保持恒定的SPI频率有关RP 2040时钟架构的概述，请参见图28

* + - 1. 发送FIFO

通用发送FIFO是一个16位宽、8个位置深的存储器缓冲区。通过AMBA APB接口写入的CPU数据存储在缓冲区中，直到被传输逻辑读出

当配置为主机或从机时，并行数据在串行转换之前写入发送FIFO，并通过SSPTXD引脚分别传输到所连接的从机或主机

* + - 1. 接收FIFO

通用接收FIFO是一个16位宽、8个位置深的存储器缓冲区。从串行接口接收的数据存储在缓冲区中，直到CPU通过AMBA APB接口读出

当配置为主机或从机时，通过SSPRXD引脚接收的串行数据将在分别并行加载到连接的从机或主机接收FIFO之前进行寄存器。

* + - 1. 发送和接收逻辑

当配置为主机时，连接的从机的时钟通过前面描述的预分频器操作从SSPCLK的分频版本获得。主发送逻辑从其发送FIFO中连续读取值，并对其执行并串转换。然后，与SSPCLKOUT同步的串行数据流和帧控制信号通过SSPTXD引脚输出到连接的从机。主接收逻辑对传入的同步SSPRXD数据流执行串并转换，提取值并将其存储到其接收FIFO中，以便随后通过APB接口读取

当配置为从机时，SSPCLKIN时钟由所连接的主机提供，并用于为其发送和接收序列计时。从机发送逻辑在主时钟的控制下，依次从其发送FIFO中读取值，执行并串转换，然后通过从机SSPTXD引脚输出串行数据流和帧控制信号。从机接收逻辑对传入的SSPRXD数据流执行串并转换，提取值并将其存储到其接收FIFO中，以便随后通过APB接口读取

* + - 1. 可编程逻辑

PrimeCell SSP生成四个单独的可屏蔽高电平有效中断。组合中断输出作为各个中断请求的OR函数

发送和接收动态数据流中断SSPTXINTR和SSPRXINTR与状态中断分离，因此可以根据FIFO触发电平读取或写入数据

* + - 1. DMA接口

PrimeCell SSP提供连接DMA控制器的接口，请参见[4.4.3.16部分](#_bookmark120)。

* + - 1. 同步寄存器和逻辑

PrimeCell SSP支持时钟PCLK和SSPCLK的异步和同步操作。同步寄存器和握手逻辑已经实现，并且始终处于活动状态在数据流的两个方向上执行控制信号的同步，即:

* + - * + 从PCLK到SSPCLK域
        + 从SSPCLK到PCLK域。
    1. 操作
       1. 接口重置

PrimeCell SSP由全局复位信号PRESETn和块特定复位信号nSSPeN复位器件复位控制器异步置位nSSP，并与SSPCLK同步取反

* + - 1. 配置SSP

复位后，PrimeCell SSP逻辑被禁用，并且在此状态下必须进行配置。必须对控制寄存器SSPCR 0和SSPCR 1进行编程，以将外设配置为在以下协议之一下运行的主设备或从设备:

* + - * + 摩托罗拉SPI
        + 德州仪器SSI
        + 国家半导体公司。

位速率来自外部SSPCLK，需要对时钟预分频寄存器SSPCPSR进行编程

* + - 1. 启用PrimeCell SSP操作

您可以在禁用PrimeCell SSP时通过写入最多8个16位值来准备发送FIFO，也可以允许发送FIFO服务请求中断CPU。使能后，数据的发送或接收从发送SSPTXD和接收SSPRXD引脚开始

* + - 1. 时钟比

对PCLK与SSPCLK的频率之比存在约束SSPCLK的频率必须小于或等于PCLK的频率。这可确保从SSPCLK域到PCLK域的控制信号在一个帧持续时间之前同步

.

在从机工作模式下，来自外部主机的SSPCLKIN信号经过双重同步，然后延迟以检测边沿。需要三个SSPCLK来检测SSPCLKIN上的边沿。SSPTXD到主机对线路进行采样的SSPCLKIN

参考SSPCLKIN，SSPRXD上的设置和保持时间必须更保守，以确保在SSPMS内进行实际采样时，其值正确为确保器件正常工作，SSPCLK必须至少比SSPCLKIN的最大预期频率快12倍

为SSPCLK选择的频率必须适应所需的位时钟速率范围。在从机模式下，最小SSPCLK频率与SSPCLKOUT最大频率之比为12，而在主机模式下，最小SSPCLK频率与最大频率之比为2。

例如，在RP 2040上的最大SSPCLK（CLK\_CLK）频率为133 MHz时，主机模式下的最大峰值比特率为62.5 Mbps。这是通过将SSPCPSR寄存器的值设置为2，将SSPCR 0寄存器中的SCR[7:0]字段设置为0来实现的。

在从机模式下，相同的最大SSPCLK频率133 MHz可以实现133/12 = 1.5MHz的峰值比特率。

~11.083Mbps。SSPCPSR寄存器可以设置为值12，SSPCR 0寄存器中的SCR[7:0]字段可以设置为值0。同样，SSPCLK最大频率与SSPCLKOUT最小频率之比为254 × 256。

SSPCLK的最小频率由以下不等式决定，这两个不等式都必须满足:

，用于主模式

，用于从机模式。

SSPCLK的最大频率由以下不等式决定，这两个不等式都必须满足:

，用于主模式

，用于从机模式。

* + - 1. SSPCR 0控制寄存器编程

SSPCR 0寄存器用于:

* + - * + 设置串行时钟频率
        + 选择三个协议之一
        + 选择数据字大小（如果适用）。

串行时钟速率（SCR）值与SSPCPSR时钟预分频除数值CPSDVSR一起用于从外部SSPCLK推导出PrimeCell SSP发送和接收比特率

帧格式通过FRF位编程，数据字大小通过DSS位编程。

位相位和极性仅适用于Motorola SPI格式，通过SPH和SPO位进行编程

* + - 1. SSPCR 1控制寄存器编程

SSPCR 1寄存器用于:

* + - * + 选择主模式或从模式
        + 启用环回测试功能
        + 使能PrimeCell SSP外设。

要将PrimeCell SSP配置为主机，请将SSPCR 1寄存器主机或从机选择位MS清0。这是重置时的默认值

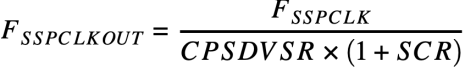
将SSPCR 1寄存器MS位设置为1可将PrimeCell SSP配置为从机。当配置为从机时，通过SSPCR 1从机模式SSPTXD输出禁用位SOD来启用或禁用PrimeCell SSP SSPTXD信号。您可以在一些主设备可能并行广播的多从设备环境中使用此方法

要使能PrimeCell SSP的操作，请将同步串行端口使能（SSE）位设置为1。

4.4.3.6.1. 比特率生成

串行比特率通过对输入时钟SSPCLK进行分频而得。时钟首先除以2-254范围内的偶数预分频值CPSDVSR，然后编程为SSPCPSR。时钟再次除以1- 256范围内的值，即1 + SCR，其中SCR是SSPCR 0中编程的值

以下公式定义了输出信号位时钟SSPCLKOUT的频率



例如，如果SSPCLK是125MHz，并且CPSDVSR = 2，则SSPCLKOUT具有从244kHz到244kHz的频率范围。 62.5MHz.

* + - 1. 帧格式

每个数据帧的长度在4-16位之间，具体取决于编程数据的大小，并从MSB开始传输您可以选择以下基本框架类型:

* + - * + Texas Instruments同步串行
        + 摩托罗拉SPI
        + 国家半导体公司

对于所有格式，当PrimeCell SSP空闲时，串行时钟SSPCLKOUT保持非活动状态，并且仅在数据的活动发送或接收期间以编程频率SSPCLKOUT的空闲状态用于提供接收超时指示，当接收FIFO在超时周期后仍包含数据时，会出现该指示

对于Motorola SPI和National Semiconductor Microwire帧格式，串行帧SSPFSSOUT引脚处于低电平有效状态，并在整个帧传输过程中置位、拉低

对于Texas Instruments同步串行帧格式，SSPFSSOUT引脚在每个帧传输之前的一个串行时钟周期内（从上升沿开始）对于这种帧格式，PrimeCell SSP和片外从器件都在SSPCLKOUT的上升沿驱动其输出数据，并在下降沿锁存来自另一个器件的数据

与其他两种帧格式的全双工传输不同，National Semiconductor Microwire格式使用一种特殊的主从消息传递技术，以半双工方式工作。在此模式下，当帧开始时，将向片外从机发送8位控制消息在此发送期间，SSS没有接收到传入数据。消息发送后，片外从机对其进行解码，并在8位控制消息的最后一位发送后等待一个串行时钟后，以请求的数据进行响应返回的数据长度可以是4-16位，使得总帧长度在13-25位的范围

* + - 1. Texas Instruments同步串行帧格式

[图88](#_bookmark107)显示了单个传输帧的Texas Instruments同步串行帧格式

*图88. Texas Instruments同步串行帧格式，单次传输*



SSPCLKOUT/SSPCLINSSPFSSOUT/SSPFSSIN

SSPTX/SSPRXD

MSB

LSB

4至16位

nSSPOE

在这种模式下，SSPCLKOUT和SSPFSSOUT被强制为低电平，并且每当PrimeCell SSP空闲时，发送数据线SSPTXD被三态化。当发送FIFO的底部条目包含数据时，SSPFSSOUT在一个SSPCLKOUT周期内被脉冲变为高要发送的值也从发送FIFO传输到发送逻辑的串行移位寄存器在SSPCLKOUT的下一个上升沿，4位至16位数据帧的MSB通过SSPTXD引脚移出同样，片外串行从器件将接收数据的MSB移位到SSPRXD引脚。

然后，PrimeCell SSP和片外串行从器件都在每个SSPCLKOUT的下降沿将每个数据位时钟输入其串行移位器LSB锁存后，在PCLK的第一个上升沿，接收数据从串行移位器传输到接收FIFO

[图89](#_bookmark108)显示了传输背靠背帧时的Texas Instruments同步串行帧格式

*图89.德州仪器同步串行帧格式，连续传输*



SSPCLKOUT/SSPCLINSSPFSSOUT/SSPFSSIN

SSPTX/SSPRXD

MSB

LSB

4至16位

nSSPOE（=0）

* + - 1. Motorola SPI帧格式

Motorola SPI接口是一个四线接口，其中SSPFSSOUT信号作为从机选择。Motorola SPI格式的主要特点是，可以使用SSPSCR0控制寄存器的SPO和SPH位对SSPCLKOUT信号的非活动状态和相位进行

* + - * 1. SPO，时钟极性

当SPO时钟极性控制位为低电平时，它在SSPCLKOUT引脚上产生稳态低电平值如果SPO时钟极性控制位为高电平，则在不传输数据时，SSPCLKOUT引脚上会设置一个稳态高电平值。

* + - * 1. SPH，时钟相位

SPH控制位选择捕获数据的时钟沿，并使其能够改变状态。在第一个数据捕获沿之前允许或不允许时钟跳变，对传输的第一位影响最大

当SPH相位控制位为低电平时，数据在第一个时钟边沿跳变时被捕获

当SPH时钟相位控制位为高电平时，数据在第二个时钟边沿跳变时捕获

* + - 1. Motorola SPI格式，SPO=0，SPH=0

[图90](#_bookmark109)和[图91](#_bookmark110)显示了SPO=0、SPH=0的Motorola SPI帧格式的连续传输信号序列[图90](#_bookmark109)显示了SPO=0、SPH=0的Motorola SPI帧格式的单个传输信号序列

*图90. MotorolaSPI帧格式，单次传输，SPO=0和SPH=0*



SSPCLKOUT/SSPCLIN

SSPFSSOUT/SSPFSSIN

SSPRXD

MSB

LSB Q

4至16位

nSSPOE

SSPRXD

MSB

LSB

[图91](#_bookmark110)显示了SPO=0、SPH=0的Motorola SPI帧格式的连续传输信号序列

*图91. MotorolaSPI帧格式，单次传输，SPO=0和SPH=0*



SSPCLKOUT/SSPCLIN

SSPFSSOUT/SSPFSSIN SSPTXD/SSPRXDLSB

MSB

LSB

MSB

4至16位

nSSPOE（=0）

在此配置中，在空闲时段期间:

* + - * + SSPCLKOUT信号被强制为低电平
        + SSPFSSOUT信号被强制为高电平
        + 发送数据线SSPTXD被任意强制为低电平
        + nSSPOE焊盘使能信号被强制为高电平（请注意，这未连接到RP 2040中的焊盘
        + 当PrimeCell SSP被配置为主机时，nSSPCTLOE线被驱动为低电平，使能SSPCLKOUT焊盘，低电平有效使能
        + 当PrimeCell SSP被配置为从设备时，nSSPCTLOE线被驱动为高电平，禁用SSPCLKOUT焊盘，低电平有效使能。

如果PrimeCell SSP使能，并且发送FIFO内有有效数据，则发送开始由SSPFSSOUT主信号驱动为低电平表示。这将使能从机数据到主机的SSPRXD输入线nSSPOE线被驱动为低电平，使能主SSPTXD输出焊盘。

半个SSPCLKOUT周期后，有效的主数据传输到SSPTXD引脚。既然主机和从机数据均已设置，SSPCLKOUT主机时钟引脚在再经过半个SSPCLKOUT周期后变为高电平

现在，数据在SSPCLKOUT信号的上升沿捕获，并在下降沿传播

在单个字传输的情况下，在数据字的所有位被传输之后，SSPFSSOUT线在最后一位被捕获之后的一个SSPCLKOUT周期返回到其空闲高电平状态

然而，在连续背靠背传输的情况下，SSPFSSOUT信号必须在每次数据字传输之间脉冲为高电平。这是因为从机选择引脚冻结其串行外设寄存器中的数据，并且如果SPH位为逻辑0，则不允许更改数据因此，主器件必须在每次数据传输之间提升从器件的SSPFSSIN引脚连续传输完成后，SSPFSSOUT引脚在捕获最后一位后的一个SSPCLKOUT周期返回空闲状态

* + - 1. Motorola SPI格式，SPO=0，SPH=1

[图92](#_bookmark111)显示了SPO=0、SPH=1的Motorola SPI格式的传输信号序列，它涵盖了单次传输和连续传输。

*图92. MotorolaSPI帧格式，SPO=0且SPH=1，*



SSPCLKOUT/SSPCLIN

SSPFSSOUT/SSPFSSIN

SSPRXD

Q MSB

LSB Q

4至16位

nSSPOE

SSPRXD

MSB

LSB

单次和连续传输

在此配置中，在空闲时段期间:

* + - * + SSPCLKOUT信号被强制为低电平
        + SSPFSSOUT信号被强制为高电平
        + 发送数据线SSPTXD被任意强制为低电平
        + nSSPOE焊盘使能信号被强制为高电平（请注意，这未连接到RP 2040中的焊盘
        + 当PrimeCell SSP被配置为主机时，nSSPCTLOE线被驱动为低电平，使能SSPCLKOUT焊盘，低电平有效使能
        + 当PrimeCell SSP被配置为从设备时，nSSPCTLOE线被驱动为高电平，禁用SSPCLKOUT焊盘，低电平有效使能。

如果PrimeCell SSP使能，并且发送FIFO内有有效数据，则发送开始由SSPFSSOUT主信号驱动为低电平表示。nSSPOE线被驱动为低电平，使能主SSPTXD输出焊盘。在额外的半个SSPCLKOUT周期之后，主设备和从设备的有效数据都被使能到它们各自的传输线上。同时，SSPCLKOUT通过上升沿跃迁使能

然后，在下降沿捕获数据，并在SSPCLKOUT信号的上升沿传播数据

在单个字传输的情况下，在所有位被传输之后，SSPFSSOUT线在最后一位被捕获之后的一个SSPCLKOUT周期返回到其空闲对于连续背靠背传输，SSPFSSOUT引脚在连续数据字之间保持低电平，终止方式与单个字传输相同。

* + - 1. Motorola SPI格式，SPO=1，SPH=0

[图93](#_bookmark112)和[图94](#_bookmark113)显示了SPO=1、SPH=0的Motorola SPI格式的单次和连续传输信号序列。

[图93](#_bookmark112)所示为Motorola SPI格式的单个传输信号序列，SPO=1，SPH=0。

*图93. MotorolaSPI帧格式，单次传输，SPO=1且SPH=0*



SSPCLKOUT/SSPCLIN

SSPFSSOUT/SSPFSSIN

SSPRXD

MSB

LSB Q

4至16位

nSSPOE

SSPRXD

MSB

LSB

[图94](#_bookmark113)显示了摩托罗拉SPI格式的连续传输信号序列，SPO=1，SPH=0。

**注意**

在[图93中](#_bookmark112)，Q是未定义的信号。

*图94. MotorolaSPI帧格式，连续传输，SPO=1和SPH=0*



SSPCLKOUT/SSPCLIN

SSPFSSOUT/SSPFSSIN SSPTXD/SSPRXDLSB

MSB

LSB

MSB

4至16位

nSSPOE（=0）

在此配置中，在空闲时段期间:

* SSPCLKOUT信号被强制为高电平
* SSPFSSOUT信号被强制为HIGH
* 发送数据线SSPTXD被任意强制为低电平
* nSSPOE焊盘使能信号被强制为高电平（请注意，这未连接到RP 2040中的焊盘
* 当PrimeCell SSP被配置为主机时，nSSPCTLOE线被驱动为低电平，使能SSPCLKOUT焊盘，低电平有效使能
* 当PrimeCell SSP被配置为从设备时，nSSPCTLOE线被驱动为高电平，禁用SSPCLKOUT焊盘，低电平有效使能。

如果PrimeCell SSP使能，且发送FIFO内有有效数据，则SSPFSSOUT主机信号被驱动为低电平表示发送开始，这会导致从机数据立即传输到主机的SSPRXD线路nSSPOE线被驱动为低电平，使能主SSPTXD输出焊盘。

半个周期后，有效的主数据传输到SSPTXD线。既然主机和从机数据均已设置，SSPCLKOUT主机时钟引脚在再经过半个SSPCLKOUT周期后变为低电平。这意味着数据在下降沿捕获，并在SSPCLKOUT信号的上升沿传播

在单个字传输的情况下，在数据字的所有位被传输之后，SSPFSSOUT线在最后一位被捕获之后的一个SSPCLKOUT周期返回到其空闲高电平状态

然而，在连续背靠背传输的情况下，SSPFSSOUT信号必须在每次数据字传输之间脉冲为高电平。这是因为从机选择引脚冻结其串行外设寄存器中的数据，并且如果SPH位为逻辑0，则不允许更改数据因此，主器件必须在每次数据传输之间提升从器件的SSPFSSIN引脚连续传输完成后，SSPFSSOUT引脚在捕获最后一位后的一个SSPCLKOUT周期返回空闲状态

* + - 1. Motorola SPI格式，SPO=1，SPH=1

[图95](#_bookmark114)显示了SPO=1、SPH=1的Motorola SPI格式的传输信号序列，其中包括单次传输和连续传输。

*图95. MotorolaSPI帧格式，SPO=1且SPH=1，*



SSPCLKOUT/SSPCLIN

SSPFSSOUT/SSPFSSIN

SSPRXD

Q MSB

LSB Q

4至16位

nSSPOE

SSPRXD

MSB

LSB

单次和连续传输

**注意**

在[图95中](#_bookmark114)，Q是未定义的信号。

在此配置中，在空闲时段期间:

* + - * + SSPCLKOUT信号被强制为高电平
        + SSPFSSOUT信号被强制为HIGH
        + 发送数据线SSPTXD被任意地强制为低
        + nSSPOE焊盘使能信号被强制为高电平（请注意，这未连接到RP 2040中的焊盘
        + 当PrimeCell SSP被配置为主机时，nSSPCTLOE线被驱动为低电平，使能SSPCLKOUT焊盘，低电平有效使能
        + 当PrimeCell SSP被配置为从设备时，nSSPCTLOE线被驱动为高电平，禁用SSPCLKOUT焊盘，低电平有效使能。

如果PrimeCell SSP使能，并且发送FIFO内有有效数据，则发送开始由SSPFSSOUT主信号驱动为低电平表示。nSSPOE线被驱动为低电平，使能主SSPTXD输出焊盘。在额外的半个SSPCLKOUT周期之后，主数据和从数据都被使能到它们各自的传输线上。同时，SSPCLKOUT通过下降沿跃迁使能然后，在上升沿捕获数据，并在SSPCLKOUT信号的下降沿传播数据

在传输完所有位后，如果是单个字传输，则在捕获最后一位后的一个SSPCLKOUT周期内，SSPFSSOUT线返回到空闲高电平状态

对于连续背靠背传输，SSPFSSOUT引脚保持低电平有效状态，直到捕获最后一个字的最后一位，然后返回到空闲状态，如前一节所述。

对于连续背靠背传输，SSPFSSOUT引脚在连续数据字之间保持低电平，终止方式与单个字传输相同

* + - 1. National Semiconductor Microwire帧格式

[图96](#_bookmark115)显示了单个帧的National Semiconductor Microwire帧格式。[图97](#_bookmark116)显示了背靠背帧传输时

*图96. Microwire帧格式，单次传输*



SSPCLKOUT/SSPCLIN

SSPFSSOUT/SSPFSSIN

SSPTXD

MSB

LSB

8-位控制

SSPRXD

0 MSB LSB

4至16位输出数据

nSSPOE

Microwire格式与SPI格式非常相似，只是传输是半双工而不是全双工，使用主从消息传递技术。每次串行传输都以从PrimeCell SSP传输到片外从器件的8位控制字开始在该传输期间，PrimeCell SSP不接收输入数据。消息发送后，片外从机对其进行解码，并在8位控制消息的最后一位发送后等待一个串行时钟后，以所需数据进行响应返回的数据长度为4到16位，使总帧长度在13-25位范围

在此配置中，在空闲时段期间:

* + - * + SSPCLKOUT被强制为低电平
        + SSPFSSOUT被强制为高电平
        + 发送数据线SSPTXD被任意强制为低电平
        + nSSPOE焊盘使能信号被强制为高电平（请注意，这未连接到RP 2040中的焊盘

通过向发送FIFO写入控制字节来触发发送。SSPFSSOUT的下降沿使发送FIFO底部条目中包含的值传输到发送逻辑的串行移位寄存器，并使SSPFSSOUT在帧传输期间保持低电平。在此传输期间，SSPRXD引脚保持三态

片外串行从机器件在每个SSPCLKOUT的上升沿将每个控制位锁存到其串行移位器中。最后一位被从机锁存后，控制字节在一个时钟等待状态下解码，从机通过将数据发送回PrimeCell SSP进行响应。每个位在SSPCLKOUT的下降沿驱动到SSPRXD线上。PrimeCell SSP又在SSPCLKOUT的上升沿锁存每个位。在帧结束时，对于单次传输，SSPFSSOUT信号在接收串行移位器中锁存最后一位后的一个时钟周期被拉高，从而将数据传输到接收FIFO。

**注意**

接收移位器锁存LSB后，片外从器件可以在SSPCLKOUT下降沿或SSPFSSOUT引脚变为高电平时

对于连续传输，数据传输的开始和结束方式与单次传输相同。然而，SSPFSSOUT线持续置位，保持低电平，数据传输发生背靠背。下一帧的控制字节紧跟在当前帧接收数据的LSB之后在帧的LSB已被锁存到PrimeCell SSP中之后，在下降沿SSPCLKOUT上从接收移位器传送每个接收值

[图97](#_bookmark116)显示了连续传输帧时的National Semiconductor Microwire帧格式

*图97. Microwire帧格式，连续传输*

SSPCLKOUT/SSPCLIN

SSPFSSOUT/SSPFSSIN

SSPTXD

LSB

MSB

LSB

8-位控制

SSPRXD

0 MSB LSB

4至16位输出数据

MSB

nSSPOE

在Microwire模式下，在SSPFSSIN变为低电平后，PrimeCell SSP从机在SSPCLKIN的上升沿对接收数据的第一位进行采样。驱动自由运行SSPCKLIN的主机必须确保SSPFSSIN信号相对于SSPCLKIN的上升沿具有足够的建立和保持裕量

[图98](#_bookmark117)显示了这些建立和保持时间要求。

对于接收数据的第一位将由PrimeCell SSP从机采样的SSPCLKIN上升沿，SSPFSSIN必须具有至少两倍于PrimeCell SSP操作的SSPCLK周期的设置

对于该沿之前的SSPCLKIN上升沿，SSPFSSIN必须保持至少一个SSPCLK周期。

*图98. Microwire帧格式，SSPFSSIN输入设置和保持要求*



tHold=t SSPCLK

tSetup=（2×t SSPCLK）

SSPCLKINSSPFSSIN

SSPRXD

SSP从机采样的第一个RX数据位

* + - 1. 主机和从机配置示例

[图99](#_bookmark118)、[图100](#_bookmark119)和[图101](#_bookmark121)显示了当PrimeCel1SSP（PL 022）外设配置为主设备或从设备时，如何将其连接到其他同步串行

**注意**

SSP（PL022）不支持系统中主机和从机之间的动态切换。每个实例被配置和连接为主设备或从设备。

[图99](#_bookmark118)显示PrimeCell SSP（PL 022）实例化两次，作为一个主机和一个从机。主机可以通过主机SSPTXD线路向从机广播。作为响应，从机将其nSSPOE信号驱动为高电平，使能其SSPTXD数据到主机的SSPRXD线上

*图99. PrimeCellSSP主机耦合到PL 022从机*

PL022配置为主机

SSPTXD

PL022配置为从机

SSPRXD

nSSPOE

nSSPOE

SSPRXD

SSPTXD

SSPFSSOUT

SSPFSSIN

SSPFSSIN

OV

SSPFSSOUT

SSPCLKOUT SSPCLKIN

nSSPCTLOE

nSSPCTLOE

SSPCLKIN

OV

SSPCLKOUT

[图100](#_bookmark119)显示了配置为主机的PrimeCell SSP（PL 022）如何与Motorola SPI从机接口。SPI从机选择（SS）信号永久连接低电平，并将其配置为从机。类似于上述操作，主设备可以通过主PrimeCell SSP SSPTXD线向从设备广播。作为响应，从机将其SPI MISO端口驱动到主机的SSPRXD线路上

*图100. PrimeCell SSP主设备耦合到SPI从设备*

PL022配置为主机

SSPTXD

SPI从机

Mosi

nSSPOE

SSPRXD

MISO

SSPFSSOUT

SSPFSSIN

OV

SSPCLKOUT

SCK

nSSPCTLOE

SS

SSPCLKIN

OV

[图101](#_bookmark121)所示为Motorola SPI配置为主机，并与配置为从机的PrimeCell SSP（PL 022）实例接口。在这种情况下，从机选择信号（SS）永久连接高电平，以将其配置为主机。主机可以通过主机SPI MOSI线向从机广播，作为响应，从机将其nSSPOE信号驱动为低电平。这将使其SSPTXD数据能够传输到主机的MISO线路

*图101. SPI主机耦合到PrimeCellSSP从机*

SPI主

Mosi

PL022配置为从机

SSPRXD

nSSPOE

MISO

SSPTXD

SSPFSSIN

OV

SSPFSSOUT

SCK

SSPCLKIN

VDD

nSSPCTLOE

SS

SSPCLKOUT

* + - 1. PrimeCell DMA接口

PrimeCell SSP提供连接到DMA控制器的接口。PrimeCell SSP DMA控制寄存器SSPDMACR控制PrimeCell SSP的DMA操作

DMA接口包括用于接收的以下信号

SSPRXDMAS

单字符DMA传输请求，由SSP置位当接收FIFO包含至少一个字符时，此信号置位

SSPRXDMABREQ

突发DMA传输请求，由SSP置位。当接收FIFO包含四个或更多字符时，此信号置位。

SSPRXDMACLR

DMA请求清除，由DMA控制器置位以清除接收请求信号。如果请求DMA脉冲串传输，则在脉冲串中最后一个数据的传输期间，清除信号被置位

DMA接口包括以下用于传输的信号

SSPTXDMAS

单字符DMA传输请求，由SSP置位。当发送FIFO中至少有一个空位置时，

SSPTXDMABREQ

突发DMA传输请求，由SSP置位。当发送FIFO包含四个或更少字符时，此信号置位。

SSPTXDMACLR

DMA请求清除，由DMA控制器断言，用于清除发送请求信号。如果请求DMA脉冲串传输，则在脉冲串中最后一个数据的传输期间，清除信号被置位

突发传送和单个传送请求信号并不相互排斥。它们可以同时被断言例如，当接收FIFO中的数据多于水印级别4时，突发传输请求和单个传输请求被断言。当接收FIFO中剩余的数据量小于水印水平时，仅断言单个请求这对于流中剩余的待接收字符数小于突发的情况很有用

例如，如果必须接收19个字符，则DMA控制器传输四个四字符的突发，以及三个单传输以完成流。

**注意**

对于剩余的三个字符，PrimeCell SSP不断言突发请求。

表495。发送和接收FIFO的 DMA触发点

每个请求信号保持有效，直到相关DMA清除信号有效。在请求清除信号被解除断言之后，请求信号可以再次变为有效，这取决于前面部分描述的条件如果PrimeCell SSP被禁用或DMA使能信号被清除，则所有请求信号都会被取消断言

[表495](#_bookmark122)显示了发送和接收FIFO的DMABREQ触发点

|  |  |  |
| --- | --- | --- |
| 突发长度 | | |
| 水印水平 | 传输，空位置 | 接收，已填充位置 |
| 1/2 | 4 | 4 |

[图102](#_bookmark123)所示为单个传输请求和突发传输请求的时序图，其中包含适当的DMA清除信号。所有信号均与PCLK同步

*图102. DMA传输波形*



PCLK DMABREQ DMACLR

* + 1. 登记册一览表

*表496。SPI寄存器列表*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x000 | [SSPCR0](#_bookmark125) | 控制寄存器0，SSPCR 0（第3-4 |
| 0x004 | [SSPCR 1](#_bookmark126) | 控制寄存器1，SSPCR 1（第3-5 |

SPI0和SPI1寄存器分别从基址0x4003c000和0x40040000开始（在SDK中定义为SPI0\_BASE和SPI1\_BASE）。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x008 | [SSPDR](#_bookmark127) | 数据寄存器，SSPDR，第3-6 |
| 0x 00c | [SSPSR](#_bookmark128) | 状态寄存器，SSPSR，第3-7 |
| 0x010 | [SSPCPSR](#_bookmark129) | 时钟预分频寄存器，SSPCPSR，第3-8 |
| 0x014 | [SSPIMSC](#_bookmark130) | 屏蔽设置或清除寄存器，SSPIMSC，第3-9 |
| 0x018 | [SSPRIS](#_bookmark131) | 原始中断状态寄存器，SSPRIS（第3-10 |
| 0x01c | [SSPMIS](#_bookmark132) | 屏蔽中断状态寄存器，SSPMIS（第3-11 |
| 0x020 | [SSPICR](#_bookmark133) | 清除寄存器，SSPICR，第3-11 |
| 0x024 | [SSPDMACR](#_bookmark134) | DMA控制寄存器，第3-12页的SSPDMACR |
| 0xfe0 | [SSPPERIPHID0](#_bookmark135) | 外围设备标识寄存器，第3-13页的SSP ID 0 -3 |
| 0xfe4 | [SSPPERIPHID1](#_bookmark136) | 外设标识寄存器，SSPPeriphID 0 -3（第3-13 |
| 0xfe8 | [SSPPERIPHID2](#_bookmark137) | 外围设备标识寄存器，第3-13页的SSP ID 0 -3 |
| 0xfec | [SSPPERIPHID3](#_bookmark138) | 外设标识寄存器，SSPPeriphID 0 -3（第3-13 |
| 0xff0 | [SSPPCELLID0](#_bookmark139) | PrimeCell识别寄存器，第3-16页的SSPPCellID 0 -3 |
| 0xff4 | [SSPPCELLID1](#_bookmark140) | PrimeCell识别寄存器，第3-16页的SSPPCellID 0 -3 |
| 0xff8 | [SSPPCELLID2](#_bookmark141) | PrimeCell识别寄存器，第3-16页的SSPPCellID 0 -3 |
| 0xffc | [SSPPCELLID3](#_bookmark142) | PrimeCell识别寄存器，第3-16页的SSPPCellID 0 -3 |

497号表。SSPCR0

寄存器

### [SPI](#_bookmark124):SSPCR 0寄存器

**偏移**:0x000

描述

控制寄存器0，SSPCR 0（第3-4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - | - |
| 15:8 | SCR | 串行时钟速率。值SCR用于生成PrimeCell SSP的发送和接收比特率位速率为:F SSPCLK CPSDVSR x（1+SCR）其中，CPSDVSR为2-254之间的偶数值，通过SSPCPSR寄存器编程，SCR为0-255之间的值 | RW | 0x00 |
| 7 | SPH | SSPCLKOUT阶段，仅适用于Motorola SPI帧格式。参见第2-10页的Motorola SPI帧格式 | RW | 0x0 |
| 6 | SPO | SSPCLKOUT极性，仅适用于Motorola SPI帧格式。参见第2-10页的Motorola SPI帧格式 | RW | 0x0 |
| 五比四 | FRF | 帧格式:00 Motorola SPI帧格式。01 TI同步串行帧格式。10 National Microwire帧格式。11保留，未定义的操作。 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三比零 | DSS | 数据大小选择:0000保留，未定义操作。0001保留，未定义操作。0010保留，未定义操作。0011 4位数据。0100 5位数据。0101 6位数据。0110 7位数据。0111 8位数据。1000个9位数据。1001 10位数据。1010 11位数据。1011 12位数据。1100 13位数据。1101 14位数据。1110 15位数据。1111 16位数据。 | RW | 0x0 |

表498。SSPCR 1

寄存器

### [SPI](#_bookmark124):SSPCR 1寄存器

**偏移量**:0x004

描述

控制寄存器1，SSPCR 1（第3-5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:4 | Reserved. | - | - | - |
| 3 | SOD | 从机模式输出禁用。此位仅在从机模式下相关，MS=1。在多从站系统中，PrimeCell SSP主站可以向系统中的所有从站广播消息，同时确保只有一个从站将数据驱动到其串行输出线上。在这样的系统中，来自多个从站的RXD线路可以被绑在一起。为了在这样的系统中操作，如果PrimeCell SSP从机不应该驱动SSPTXD线，则可以设置SOD位:0 SSP可以在从机模式下驱动SSPTXD输出。1 SSP不得在从机模式下驱动SSPTXD输出 | RW | 0x0 |
| 2 | MS | 主模式或从模式选择。此位仅在PrimeCell SSP禁用时才能修改1设备配置为从设备。 | RW | 0x0 |
| 1 | SSE | 同步串行端口使能:0 SSP操作禁用。1 SSP操作使能。 | RW | 0x0 |
| 0 | LBM | 环回模式:0正常串行端口操作使能。1发送串行移位器的输出在内部连接到接收串行移位器的输入 | RW | 0x0 |

### [SPI](#_bookmark124):SSPDR寄存器

**偏移量**:0x008

描述

数据寄存器，SSPDR，第3-6

*表499。SSPDR*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - | - |
| 15:0 | 数据 | 发送/接收FIFO:读取接收FIFO。写入发送FIFO。当PrimeCell SSP编程为小于16位的数据大小发送逻辑忽略顶部未使用的位。接收逻辑自动右对齐. | RWF | - |

寄存器

500号SSPSR

寄存器

501号。SSPCPSR

寄存器

表502. SSPIMSC

寄存器

### [SPI](#_bookmark124):SSPSR寄存器

**偏移**:0x00c

描述

状态寄存器，SSPSR（第3-7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:5 | Reserved. | - | - | - |
| 4 | BSY | PrimeCell SSP忙标志，RO:0 SSP空闲。1 SSP当前正在发送和/或接收帧，或者发送FIFO不为空。 | RO | 0x0 |
| 3 | RFF | 接收FIFO已满，RO:0接收FIFO未满。1接收FIFO已满。 | RO | 0x0 |
| 2 | RNE | 接收FIFO不为空，RO:0接收FIFO为空。1接收FIFO不为空。 | RO | 0x0 |
| 1 | TNF | 发送FIFO未满，RO:0发送FIFO已满。1发送FIFO未满。 | RO | 0x1 |
| 0 | TFE | 发送FIFO为空，RO:0发送FIFO不为空。1发送FIFO为空。 | RO | 0x1 |

[SPI](#_bookmark124):SSPCPSR寄存器

**偏移**:0x010

描述

时钟预分频寄存器，SSPCPSR，第3-8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | CPSDVSR | 时钟预分频因子。必须是2- 254之间的偶数读取时最低有效位始终返回零 | RW | 0x00 |

[SPI](#_bookmark124):SSPIMSC寄存器

**偏移量**:0x014

描述

屏蔽设置或清除寄存器，SSPIMSC，第3-9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:4 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 3 | TXIM | 发送FIFO中断屏蔽:0发送FIFO半空或更少条件中断被屏蔽。1未屏蔽发送FIFO半空或更少条件中断 | RW | 0x0 |
| 2 | RXIM | 接收FIFO中断屏蔽:0接收FIFO半满或更少条件中断被屏蔽。1接收FIFO半满或更少条件中断未屏蔽。 | RW | 0x0 |
| 1 | RTIM | 接收超时中断屏蔽:0接收FIFO不为空，并且在超时周期中断被屏蔽之前没有读取1接收FIFO不为空且在超时周期中断之前没有读取未被屏蔽。 | RW | 0x0 |
| 0 | 罗里姆 | 接收溢出中断屏蔽:0接收FIFO写入，而完全条件中断被屏蔽。1在未屏蔽满状态中断时 | RW | 0x0 |

表503. SSPRIS

寄存器

表504. SSPMIS

寄存器

### [SPI](#_bookmark124):SSPRIS寄存器

**偏移**:0x018

描述

原始中断状态寄存器，SSPRIS（第3-10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:4 | Reserved. | - | - | - |
| 3 | TXRIS | 给出屏蔽之前SSPTXINTR中断的原始中断状态 | RO | 0x1 |
| 2 | RXRIS | 给出屏蔽之前SSPRXINTR中断的原始中断状态 | RO | 0x0 |
| 1 | RTRIS | 给出SSPRTINTR中断屏蔽前的原始中断状态 | RO | 0x0 |
| 0 | 罗里斯 | 给出SSPRORINTR中断屏蔽前的原始中断状态 | RO | 0x0 |

[SPI](#_bookmark124):SSPMIS寄存器

**偏移**:0x01c

描述

屏蔽中断状态寄存器，SSPMIS（第3-11

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:4 | Reserved. | - | - | - |
| 3 | TXMIS | 给出SSPTXINTR中断屏蔽后的发送FIFO屏蔽中断状态 | RO | 0x0 |
| 2 | RXMIS | 给出SSPRXINTR中断屏蔽后的接收FIFO屏蔽中断状态 | RO | 0x0 |
| 1 | RTMIS | 给出SSPRTINTR中断屏蔽后的接收超时屏蔽中断状态 | RO | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 0 | RORMIS | 给出SSPRORINTR中断屏蔽后的接收过运行屏蔽中断状态 | RO | 0x0 |

505号。SSPICR

寄存器

表506. SSPDMACR

寄存器

507号。SSPPERIPHID0

寄存器

表508. SSPPERIPHID1

寄存器

### [SPI](#_bookmark124):SSPICR寄存器

**偏移**:0x020

描述

清除寄存器，SSPICR，第3-11

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一比二 | Reserved. | - | - | - |
| 1 | RTIC | 清除SSPRTINTR中断 | WC | 0x0 |
| 0 | 罗里克 | 清除SSPRORINTR中断 | WC | 0x0 |

[SPI](#_bookmark124):SSPDMACR寄存器

**偏移量**:0x024

描述

DMA控制寄存器，第3-12页的SSPDMACR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一比二 | Reserved. | - | - | - |
| 1 | TXDMAE | 发送DMA使能。如果该位设置为1，则发送FIFO的DMA使能。 | RW | 0x0 |
| 0 | RXDMAE | 接收DMA启用。如果该位设置为1，则使能接收FIFO的DMA | RW | 0x0 |

[SPI](#_bookmark124):SSPPERIPHID 0寄存器

**偏移量**:0xfe0

描述

外围设备标识寄存器，第3-13页的SSP ID 0 -3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | 零件编号0 | 这些位回读为0x22 | RO | 0x22 |

[SPI](#_bookmark124):SSPPERIPHID 1寄存器

**偏移量**:0xfe4

描述

外设标识寄存器，SSPPeriphID 0 -3（第3-13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 七比四 | 公司简介 | 这些位读回为0x1 | RO | 0x1 |
| 三比零 | 零件编号1 | 这些位读回为0x0 | RO | 0x0 |

509号。SSPPERIPHID2

寄存器

表510。SSPPERIPHID3

寄存器

表511。SSPPCELLID0寄存器

512号。SSPPCELLID1寄存器

513号。SSPPCELLID2寄存器

### [SPI](#_bookmark124):SSPPERIPHID 2寄存器

**偏移量**:0xfe8

描述

外围设备标识寄存器，第3-13页的SSP ID 0 -3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 七比四 | 修订 | 这些位返回外围版本 | RO | 0x3 |
| 三比零 | 联系我们 | 这些位读回为0x 4 | RO | 0x4 |

[SPI](#_bookmark124):SSPPERIPHID 3寄存器

**偏移量**:0xfec

描述

外设标识寄存器，SSPPeriphID 0 -3（第3-13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | 配置 | 这些位回读为0x00 | RO | 0x00 |

[SPI](#_bookmark124):SSPPCELLID0寄存器

**偏移量**:0xff0

描述

PrimeCell识别寄存器，第3-16页的SSPPCellID 0 -3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | SSPPCELLID0 | 这些位读回为0x0D | RO | 0x0D |

[SPI](#_bookmark124):SSPPCELLID1寄存器

**偏移量**:0xff4

描述

PrimeCell识别寄存器，第3-16页的SSPPCellID 0 -3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | SSPPCELLID1 | 这些位回读为0xF0 | RO | 0xf0 |

[SPI](#_bookmark124):SSPPCELLID2寄存器

**偏移量**:0xff8

描述

PrimeCell识别寄存器，第3-16页的SSPPCellID 0 -3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 7:0 | SSPPCELLID2 | 这些位读回为0x05 | RO | 0x05 |

514号。SSPPCELLID3寄存器

### [SPI](#_bookmark124):SSPPCELLID3寄存器

**偏移量**:0xffc

描述

PrimeCell识别寄存器，第3-16页的SSPPCellID 0 -3

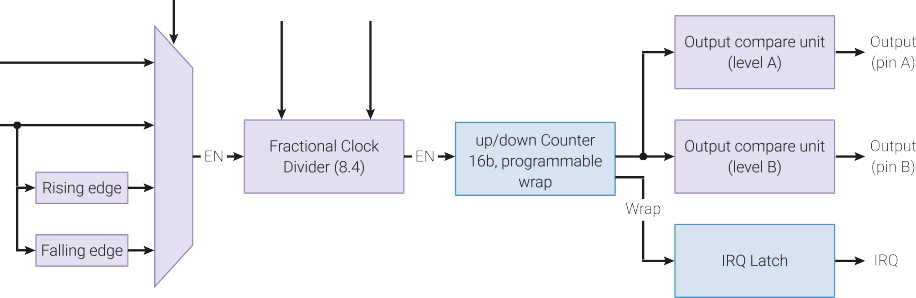
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | SSPPCELLID3 | 这些位回读为0xB1 | RO | 0xb1 |

* 1. PWM
     1. 概述

脉宽调制（PWM）是一种数字信号提供平滑变化的平均电压的方案。这是通过以规则的间隔控制宽度的正脉冲来高电平所占的时间部分称为占空比。这可用于近似模拟输出或控制开关模式功率电子器件。

RP 2040 PWM模块有8个相同的切片。每个片可以驱动两个PWM输出信号，或测量输入信号的频率或占空比这提供了总共多达16个可控PWM输出。所有30个GPIO引脚均可由PWM模块驱动。

图103.单个PWM切片。一个16位计数器从0计数到某个编程值，然后返回到零，或再次向下计数，具体取决于PWM模式。A和B输出根据当前计数值和预编程的A和B阈值转换高电平和低电平计数器的进步的基础上的一些事件:它可能是自由运行，或门控的水平或边缘的输入信号的B引脚。小数分频器可降低总体计数速率，以便更好地控制输出频率。



每个PWM片均配备以下功能:

* + - * 16-位计数器
      * 8.4分数时钟分频器
      * 两个独立的输出通道，占空比从0%到100%（**含）**
      * 双斜率和后缘调制
      * 边缘敏感输入模式，用于频率测量
      * 用于占空比测量的电平敏感输入模式
      * 可配置计数器回绕值
        + 回绕和电平寄存器是双缓冲的，可以在PWM运行时进行无竞争更改
      * 计数器回绕时的DMA请求
* 运行时相位可以精确提前或延迟（增量为一个计数）

可以通过单个全局控制寄存器同时使能或禁用切片。然后，切片完美同步运行，以便可以通过多个切片的输出切换更复杂的电源电路

* + 1. 程序员模型

515号 将PWM通道映射到RP 2040上的GPIO引脚。主GPIO功能表（表279）也显示了这一点

RP 2040上的所有30个GPIO引脚均可用于PWM:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| GPIO | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| PWM通道 | 0A | 0B | 1A | 1B | 2A | 2B | 3A | 3B | 4A | 4B | 5A | 5B | 6A | 6B | 7A | 7B |
| GPIO | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |  |  |
| PWM通道 | 0A | 0B | 1A | 1B | 2A | 2B | 3A | 3B | 4A | 4B | 5A | 5B | 6A | 6B |  |  |

* 16个PWM通道（8个双通道切片）出现在GPIO 0至GPIO 15上，顺序为PWM0 A、PWM0 B、PWM1 A.
* 这对于GPIO 16至GPIO 29重复GPIO 16为PWM 0 A，GPIO 17为PWM 0 B，因此GPIO 29上的PWM 6 B
* 可以在两个GPIO引脚上选择相同的PWM输出;每个GPIO上将显示相同的信号
* 如果PWM B引脚用作输入，并且在多个GPIO引脚上被选择，则PWM片将看到这两个GPIO输入的逻辑或
  + - 1. 脉宽调制

PWM硬件通过连续比较输入值与自由运行计数器来运行。这将产生一个切换输出，其中在高输出电平上花费的时间量与输入值成比例在高信号电平上花费的时间部分被称为信号的占空比

计数周期由TOP寄存器控制，最大可能周期为65536个周期，因为计数器和TOP的大小为16位输入值通过CC寄存器配置

*图104.计数器从0到TOP重复计数，形成一个圆形。计数器不断地与某个输入值进行比较。 当输入值高于计数器时，输出被驱动为高电平。否则，输出为低电平。输出周期T由计数器的TOP值以及计数器的计数速度来定义。***平均***输出电压（作为IO电源的一部分）是输入值除以计数器周期（TOP +1）*

输入（计数）

计数

顶部

首页/3

0

不

2T

3T

不

输出（脉冲）

V

IOVDD

0

不

2T

3T

不

GPIO脉冲输出

计数器比较级计数器

此示例显示了在RP 2040的一个PWM片上配置的计数周期以及A和B计数器比较电平。

Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pwm/hello\_pwm/hello\_pwm.c](https://github.com/raspberrypi/pico-examples/blob/master/pwm/hello_pwm/hello_pwm.c#L15-L29)第15 - 29

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

//告诉GPIO 0和1它们被分配给PWM

gpio\_set\_function（0，GPIO\_PWM）; gpio\_set\_function（1，GPIO\_PWM）;

//找出哪个PWM片连接到GPIO 0（它是片0）

int slice\_num = pwm\_gpio\_to\_slice\_num（0）;

//设置周期为4个周期（0到3，包括0和3）

pwm\_set\_wrap（slice\_num，3）;

//将通道A输出设置为高电平一个周期，然后丢弃

pwm\_set\_chan\_level（slice\_num，PWM\_CHAN\_A，1）;

//将初始B输出设置为高电平三个周期， 下降

pwm\_set\_chan\_level（slice\_num，PWM\_CHAN\_B，3）;

//设置PWM运行

pwm\_set\_enabled（slice\_num，true）;

[图105](#_bookmark144)显示了PWM硬件在以这种方式配置后的工作

图105.切片从0到3重复计数，配置为TOP值。因此，输出波的周期为4。 输出A在4个周期中有1个周期为高电平，因此平均输出电压为IO电源电压的1/4。

输出B在每4个周期中有3个周期为高电平。请注意，A和B的上升沿始终对齐。

计数

A B

0 1 2 3 0 1 2 3 0 1 2 3

PWM片的默认行为是向上计数，直到达到TOP寄存器的值，然后立即回绕为0。PWM片还提供相位校正模式，通过将CSR\_PH\_CORRECT设置为1启用，计数器在达到TOP后开始向下计数，直到再次达到0

它被称为相位校正模式，因为无论占空比如何，脉冲始终集中在同一点上。换句话说，其相位不是占空比的函数。相位校正模式使能时，输出频率减半。

*图106. 在相位校正模式下，计数器一旦到达TOP，就从TOP倒计数到0.*

输入（计数）

计数

顶部

首页/3

0

不

2T

3T

不

输出（脉冲）

V

IOVDD

0

不

2T

3T

不

GPIO脉冲输出

计数器比较级计数器

* + - 1. 0%和100%占空比

RP2040 PWM可以产生无切换的0%和100%占空比输出。

*图107.CC = 0时无毛刺0%占空比输出，CC = TOP + 1时无毛刺100%占空比输出*

输入（计数）

计数

顶部

0

不

2T

3T

不

输出（脉冲）

V

IOVDD

0

不

2T

3T

不

GPIO脉冲输出

计数器比较级计数器

CC值为0将产生0%的输出，即输出信号总是低。TOP+ 1的CC值（即等于周期，在非相位校正模式下）将产生100%的输出。例如，如果TOP被编程为254，则计数器将具有255个周期的周期，并且在0到255（包括0和255）范围内的CC值将产生在0%到100%（包括0%和100%）范围内的占空比

0%和100%时的无毛刺输出非常重要，例如，当MOSFET被控制在其最小和最大电流水平时，可以避免开关损耗。

* + - 1. 双缓冲

[图108](#_bookmark145)显示输入值的变化如何导致输出占空比的变化。这可以用来近似一些模拟波形，如正弦波。

*图108. 输入值随每个计数器周期而变化:首先是TOP/3，然后是2 × TOP*

输入（计数）

计数

TOP2×TOP/3

首页/3

0

不

2T

3T

不

输出（脉冲）

V

IOVDD

0

T/3

不

5T/3 2T

3T

不

GPIO脉冲输出

计数器比较级计数器

/3，最后TOP +1，占空比为100%。输入值的每一次增加都会导致输出占空比的相应增加。

在[图10](#_bookmark145)- 8中，输入值仅在计数器换行为0时发生变化。[图109](#_bookmark146)显示了如果允许输入值在任何其他时间改变会发生什么:输出端会产生不必要的毛刺

*图109. 当计数器处于中间斜坡时，输入值发生变化。 这会在输出端产生额外的切换。*

输入（计数）

计数

TOP2×TOP/3

首页/3

0

不

2T

3T

不

输出（脉冲）

V

IOVDD

0

T/3

不

5T/3 2T

3T

不

GPIO脉冲输出

计数器比较级计数器

如果TOP寄存器也被修改，这种行为就会变得更加令人困惑。软件很难以正确的时间写入CC或TOP为了解决这个问题，每个切片都有CC和TOP寄存器的两个副本:一个副本可以由软件修改，另一个是内部副本，在计数器回绕的瞬间从第一个寄存器更新软件可以随意修改其寄存器副本，但直到下一次循环之前，PWM输出不会捕获这些更改

[图110](#_bookmark147)显示了软件中断处理程序在每次计数器回绕时更改CC\_A

图110.每次计数器回绕都会使中断请求信号生效。处理器进入其中断处理程序，写入CC寄存器的副本，并清除中断。当计数器再次回绕时，CC寄存器的锁存版本会立即更新为软件写入的最新值，该值控制下一周期的占空比。IRQ被重新置位，以便软件可以将另一个新值写入其CC寄存器副本。

顶部柜台

IRQ

CC\_A

0

1

2

3

CC\_A锁存

0

1

2

对于可以将哪些值写入CC或TOP，或者何时写入，没有限制。在正常PWM模式下（CSR\_PH\_CORRECT为0），当计数器回卷为0时，锁存副本更新，每TOP+ 1周期发生一次。在相位校正模式下（CSR\_PH\_CORRECT为1），锁存副本在0到0计数转换时更新，即计数器停止向下计数并再次开始向上计数的点

* + - 1. 时钟分频器

每个芯片都有一个小数时钟分频器，由DIV寄存器配置。这是一个8整数位、4小数位时钟分频器，可将计数速率降低到原来的256倍时钟分频器允许实现低得多的低于此频率将需要系统定时器中断（[第4.6](#_bookmark160)）

它通过产生一个门控计数器操作的使能信号来实现这一点

DIV\_INT DIV\_FRAC

计数器启用

DIV\_INT DIV\_FRAC

计数器启用

DIV\_INT DIV\_FRAC

计数器启用

*图111.时钟分频器产生使能信号。计数器仅在此信号为高电平时计数周期。时钟除数为1时，使能在每个周期置位，因此计数器在每个系统时钟周期计数1。除数越高，计数使能的置位频率就越低。*

分数分频通过将一些使能脉冲间隔得比其他脉冲更远来实现平均分数计数率。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 1 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | .0 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | 3 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | .0 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | 2 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | .5 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

分数分频器是一阶delta-sigma类型。

当使用电平敏感或边沿敏感模式进行占空比或频率测量时，时钟分频器还允许扩展有效计数范围。

* + - 1. 电平敏感和边缘敏感触发

图112. PWM切片事件选择。计数器在其使能输入为高电平时前进，并且该使能在两个连续阶段中产生。首先，四种事件类型（常通、引脚B高电平、引脚B上升、引脚B下降）中的任何一种都可以为小数时钟分频器生成使能脉冲。分频器可以在将使能脉冲传递到计数器之前降低使能脉冲的速率。

事件选择阶段阶段

提前延迟

1

输入（引脚B）

EN 小数时钟分频器（8.4）

计数使能

上升沿

下降沿

默认情况下，每个切片的计数器都是自由运行的，只要启用切片，就会连续计数。还有三个其他选项可用:

* + - * + 当在B引脚上检测到高电平时，连续计数
        + 在B引脚上检测到每个上升沿时计数一次
        + 在B引脚上检测到每个下降沿时计数一次

这些模式由每个切片CSR中的DIVMODE字段选择。在自由运行模式下，A和B引脚都是输出。在任何其他模式下，B引脚成为输入，并控制计数器的操作CC\_B在不处于自由运行模式时被忽略

通过允许切片在电平敏感或边沿敏感模式下运行固定的时间，可以测量输入信号的占空比或频率由于所使用的边沿检测电路的类型，低时段和高时段

当进行频率测量时，测量信号的周期必须严格大于系统时钟周期。

时钟分频器在电平敏感和边沿敏感模式下仍能工作。在最大分频（将0写入DIV\_INT）时，计数器在电平敏感模式下每256个高电平输入周期仅前进一次，在边沿敏感模式下每256个边沿仅前进一次这允许进行更长时间的测量，尽管分辨率仍然只有16位。

*Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/pwm/measure\_duty\_cycle/measure\_duty\_cycle.c](https://github.com/raspberrypi/pico-examples/blob/master/pwm/measure_duty_cycle/measure_duty_cycle.c#L19-L37)第19 - 37*

19float measure\_duty\_cycle（uint pio）{

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37}

//仅PWMB引脚可用作输入。

assert（pwm\_gpio\_to\_channel（gpio）== PWM\_CHAN\_B）; uint slice\_num = pwm\_gpio\_to\_slice\_num（gpio）;

*//每100个周期计数一次PWM B输入为高*pwm\_config cfg = pwm\_get\_default\_config（）; pwm\_config\_set\_clkdiv\_mode（cfg，PWM\_DIV\_B\_HIGH）; pwm\_config\_set\_clkdiv（cfg，100）;

pwm\_init（slice\_num，cfg，false）; gpio\_set\_function（gpio，GPIO\_FLOW\_PWM）;

pwm\_set\_enabled（slice\_num，true）; sleep\_ms（10）; pwm\_set\_enabled（slice\_num，false）;

float counting\_rate = clock\_get\_hz（counting\_sys）/100; float max\_possible\_count = counting\_rate \* 0。01;

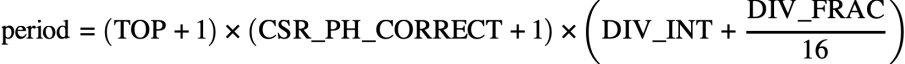
返回pwm\_get\_counter（slice\_num）/max\_possible\_count;

* + - 1. 配置PWM周期

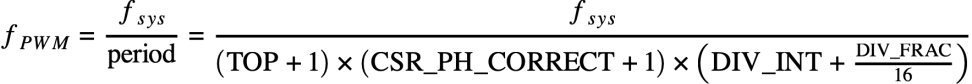
当自由运行时，PWM片的输出周期（以系统时钟周期测量）由三个参数控制:

* + - * + 顶级寄存器
        + 是否启用相位校正模式（CSR\_PH\_CORRECT）
        + DIV寄存器

切片从0计数到TOP，然后根据CSR\_PH\_CORRECT的设置进行换行或开始向后计数。计数速率由时钟分频器减慢，最大速度为每周期一次计数，最小速度为每周期一次计数时钟周期中的周期可以计算为:



然后，可以基于系统时钟频率来确定输出频率



* + - 1. 数据请求（IRQ）和DMA数据请求（DREQ）

PWM模块有一个IRQ输出。中断状态寄存器INTR、INTS和INTE允许软件控制哪些片将断言IRQ输出，检查哪些片是IRQ断言的原因，以及清除和确认中断。

每次计数器回卷时（或者，如果CSR\_PH\_CORRECT使能，则每次计数器返回0时），片都会生成中断请求这将在原始中断状态寄存器INTR中设置与该片对应的标志。如果该片的中断在INTE中使能，则该标志将导致PWM模块的IRQ被置位，并且该标志也将出现在屏蔽中断状态寄存器INTS中。

通过将掩码写回INTR来清除标志。这在“LED fade”SDK示例中演示

该方案允许多个切片并发地生成中断，并且允许系统中断处理程序确定哪些切片导致了最近的中断，并进行适当的处理。通常，这意味着重新加载这些片的TOP或CC寄存器，但PWM块也可以用作非PWM相关目的的常规中断请求源。

在INTR中设置中断标志的同一脉冲也可作为一个周期数据请求发送到RP 2040系统DMA。对于DMA看到DREQ断言的每个周期，它将尽可能及时地将一个数据传输到其编程位置。结合CC和TOP的双缓冲行为，这使得DMA能够以每个计数器周期传输一次的速率将数据高效地传输到PWM片。或者，PWM片可以用作DMA传输到其他内存映射硬件的起搏定时器

* + - 1. 实时相位调整

对于某些应用，有必要控制不同切片上的两个PWM输出之间的相位关系

全局使能寄存器EN包含每个片的CSR\_EN标志的别名，并允许同时启动和停止多个片如果具有相同输出频率的两个切片同时启动，它们将以完美的锁步运行，并具有由初始计数器值确定的固定相位关系

CSR\_PH\_ADV和CSR\_PH\_RET字段将在切片运行时将切片的输出相位提前或延迟一个计数它们通过从时钟使能（时钟分频器的输出）插入或删除脉冲来实现，如[图113](#_bookmark148)所示。

*图113.由时钟分频器输出的时钟使能信号控制计数速率。相位超前在时钟使能为低电平的周期强制时钟使能为高电平，从而使计数器向前跳一个计数。相位延迟迫使时钟使能在高电平时变为低电平，使计数器后退一个计数。*

时钟

DIV\_INT

时钟使能

计数

2

0

1

2

3

4

5

DIV\_INTCSR\_PH\_ADV

时钟使能

计数

2

0

1

2 3

4

5

6

DIV\_INT CSR\_PH\_RET

时钟使能

计数

2

0

1

2

3

4

计数器的计数速度不能超过每个周期一次，因此PH\_ADV要求DIV\_INT> 1或DIV\_FRAC> 0。同样，如果在时钟使能永久为低电平时PH\_RET置位，计数器也不会开始反向计数

要将相位提前或延迟一个计数，软件将1写入PH\_ADV或PH\_RET。一旦插入或删除使能脉冲，PH\_ADV或PH\_RET寄存器位将返回0，软件可以轮询CSR，直到发生这种PH\_ADV将始终在下一个可用间隙中插入脉冲，PH\_RET将始终删除下一个可用脉冲。

* + 1. 登记册一览表

PWM寄存器从基址0x40050000（在SDK中定义为PWM\_BASE

*表516. PWM寄存器列表*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [CH0\_CSR](#_bookmark150) | 控制和状态寄存器 |
| 0x04 | [CH0\_DIV](#_bookmark151) | INT和FRAC构成一个定点小数。  计数率是系统时钟频率除以这个数字。分数除法使用简单的一阶Σ-Δ。 |
| 0x08 | [CH0\_CTR](#_bookmark152) | 直接访问PWM计数器 |
| 0x0C | [CH0\_CC](#_bookmark153) | 计数器比较值 |
| 0x10 | [CH0\_TOP](#_bookmark154) | 计数器回绕值 |
| 0x14 | [CH1\_CSR](#_bookmark150) | 控制和状态寄存器 |
| 0x18 | [CH1\_DIV](#_bookmark151) | INT和FRAC构成一个定点小数。  计数率是系统时钟频率除以这个数字。分数除法使用简单的一阶Σ-Δ。 |
| 0x1c | [CH1\_CTR](#_bookmark152) | 直接访问PWM计数器 |
| 0x20 | [CH1\_CC](#_bookmark153) | 计数器比较值 |
| 0x24 | [CH1\_TOP](#_bookmark154) | 计数器回绕值 |
| 0x28 | [CH2\_CSR](#_bookmark150) | 控制和状态寄存器 |
| 0x2c | [CH2\_DIV](#_bookmark151) | INT和FRAC构成一个定点小数。  计数率是系统时钟频率除以这个数字。分数除法使用简单的一阶Σ-Δ。 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x30 | [CH2\_CTR](#_bookmark152) | 直接访问PWM计数器 |
| 0x34 | [CH2\_CC](#_bookmark153) | 计数器比较值 |
| 0x38 | [CH2\_TOP](#_bookmark154) | 计数器回绕值 |
| 0x3c | [CH3\_CSR](#_bookmark150) | 控制和状态寄存器 |
| 0x40 | [CH3\_DIV](#_bookmark151) | INT和FRAC构成一个定点小数。  计数率是系统时钟频率除以这个数字。分数除法使用简单的一阶sigma-delta。 |
| 0x44 | [CH3\_CTR](#_bookmark152) | 直接访问PWM计数器 |
| 0x48 | [CH3\_CC](#_bookmark153) | 计数器比较值 |
| 0x4c | [CH3\_TOP](#_bookmark154) | 计数器回绕值 |
| 0x50 | [CH4\_CSR](#_bookmark150) | 控制和状态寄存器 |
| 0x54 | [CH4\_DIV](#_bookmark151) | INT和FRAC构成一个定点小数。  计数率是系统时钟频率除以这个数字。分数除法使用简单的一阶sigma-delta。 |
| 0x58 | [CH4\_CTR](#_bookmark152) | 直接访问PWM计数器 |
| 0x5c | [CH4\_CC](#_bookmark153) | 计数器比较值 |
| 0x60 | [CH4\_TOP](#_bookmark154) | 计数器回绕值 |
| 0x64 | [CH5\_CSR](#_bookmark150) | 控制和状态寄存器 |
| 0x68 | [CH5\_DIV](#_bookmark151) | INT和FRAC构成一个定点小数。  计数率是系统时钟频率除以这个数字。分数除法使用简单的一阶sigma-delta。 |
| 0x6c | [CH5\_CTR](#_bookmark152) | 直接访问PWM计数器 |
| 0x70 | [CH5\_CC](#_bookmark153) | 计数器比较值 |
| 0x74 | [CH5\_TOP](#_bookmark154) | 计数器回绕值 |
| 0x78 | [CH6\_CSR](#_bookmark150) | 控制和状态寄存器 |
| 0x7c | [CH6\_DIV](#_bookmark151) | INT和FRAC构成一个定点小数。  计数率是系统时钟频率除以这个数字。分数除法使用简单的一阶sigma-delta。 |
| 0x80 | [CH6\_CTR](#_bookmark152) | 直接访问PWM计数器 |
| 0x84 | [CH6\_CC](#_bookmark153) | 计数器比较值 |
| 0x88 | [CH6\_TOP](#_bookmark154) | 计数器回绕值 |
| 0x8c | [CH7\_CSR](#_bookmark150) | 控制和状态寄存器 |
| 0x90 | [CH7\_DIV](#_bookmark151) | INT和FRAC构成一个定点小数。  计数率是系统时钟频率除以这个数字。分数除法使用简单的一阶Σ-Δ。 |
| 0x94 | [CH7\_CTR](#_bookmark152) | 直接访问PWM计数器 |
| 0x98 | [CH7\_CC](#_bookmark153) | 计数器比较值 |
| 0x9c | [CH7\_TOP](#_bookmark154) | 计数器回绕值 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0xa0 | [EN](#_bookmark155) | 此寄存器对所有通道的CSR\_EN位进行  写入此寄存器允许同时启用或禁用多个通道对于每个通道，只有一个物理EN寄存器位，可通过here或CHx\_CSR访问 |
| 0xa4 | [INTR](#_bookmark156) | 原始中断 |
| 0xa8 | [INTE](#_bookmark157) | 中断使能 |
| 0xac | [INTF](#_bookmark158) | 中断强制 |
| 0xb0 | [INTs](#_bookmark159) | 强制屏蔽后的屏蔽状态 |

表517.CH0\_CSR，CH1\_CSR，.，CH6\_CSR，CH7\_CSR

寄存器

### [PWM](#_bookmark149):CH0\_CSR、CH1\_CSR、.、CH6\_CSR、CH7\_CSR寄存器

**偏移**:0x00、0x14、...、0x78、0x8c

描述

控制和状态寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7 | PH\_ADV | 计数器运行时，将计数器的相位提前1个计数  自动清除写入1，并轮询直到低电平。计数器必须运行  低于全速（div\_int + div\_frac/16> 1） | SC | 0x0 |
| 6 | PH\_RET | 计数器运行时，将其相位延迟1个计数  自动清除写入1，并轮询直到低电平。计数器必须运行。 | SC | 0x0 |
| 五比四 | DIVMODE | 0x0→以小数分频器指定的速率自由运行计数  0x1→小数分频器工作由PWM B引脚选通  0x2→计数器随PWM B引脚的每个上升沿  0x3→计数器随PWM B引脚的每个下降沿前进 | RW | 0x0 |
| 3 | B\_INV | 反相输出B | RW | 0x0 |
| 2 | A\_INV | 反相输出A | RW | 0x0 |
| 1 | PH\_正确 | 1:启用相位校正调制。0:后缘 | RW | 0x0 |
| 0 | EN | 使能PWM通道。 | RW | 0x0 |

### [PWM](#_bookmark149):CH0\_DIV、CH1\_DIV、.、CH6\_DIV、CH7\_DIV寄存器

**偏移量**:0x04、0x18、...、0x7c、0x90

描述

INT和FRAC构成一个定点小数。

计数率是系统时钟频率除以这个数字。分数除法使用简单的一阶sigma-delta。

*518号。CH0\_DIV，CH1\_DIV，.，CH6\_DIV，*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - | - |
| 十一点四分 | INT |  | RW | 0x01 |
| 三比零 | FRAC |  | RW | 0x0 |

CH7\_DIV寄存器

表519. CH0\_CTR，CH1\_CTR，.，CH6\_CTR，CH7\_CTR

寄存器

520号CH0\_CC、CH1\_CC、...、CH6\_CC、

CH7\_CC寄存器

表521。CH0\_TOP、CH1\_TOP、...、CH6\_TOP、CH7\_TOP

寄存器

522号EN寄存器

### [PWM](#_bookmark149):CH0\_CTR、CH1\_CTR、.、CH6\_CTR、CH7\_CTR寄存器

**偏移**:0x08、0x1c、...、0x80、0x94

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - |
| 15:0 | 直接访问PWM计数器 | RW | 0x0000 |

### [PWM](#_bookmark149):CH0\_CC、CH1\_CC、.、CH6\_CC、CH7\_CC寄存器

**偏移**:0x0c、0x20、...、0x84、0x98

描述

计数器比较值

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | B |  | RW | 0x0000 |
| 15:0 | 一 |  | RW | 0x0000 |

### [PWM](#_bookmark149):CH0\_TOP、CH1\_TOP、.、CH6\_TOP、CH7\_TOP寄存器

**偏移量**:0x10、0x24、...、0x88、0x9c

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - |
| 15:0 | 计数器回绕值 | RW | 0xffff |

### [PWM](#_bookmark149):EN寄存器

**偏移量**:0xa0

描述

此寄存器对所有通道的CSR\_EN位进行

写入此寄存器允许同时启用或禁用多个通道对于每个通道，只有一个物理EN寄存器位，可通过here或CHx\_CSR访问

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7 | CH7 |  | RW | 0x0 |
| 6 | CH6 |  | RW | 0x0 |
| 5 | CH5 |  | RW | 0x0 |
| 4 | CH4 |  | RW | 0x0 |
| 3 | CH3 |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 2 | CH2 |  | RW | 0x0 |
| 1 | CH1 |  | RW | 0x0 |
| 0 | CH0 |  | RW | 0x0 |

523号INTR寄存器

524号 INTE寄存器

### [PWM](#_bookmark149):INTR寄存器

**偏移量**:0xa4

描述

原始中断

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7 | CH7 |  | WC | 0x0 |
| 6 | CH6 |  | WC | 0x0 |
| 5 | CH5 |  | WC | 0x0 |
| 4 | CH4 |  | WC | 0x0 |
| 3 | CH3 |  | WC | 0x0 |
| 2 | CH2 |  | WC | 0x0 |
| 1 | CH1 |  | WC | 0x0 |
| 0 | CH0 |  | WC | 0x0 |

### [PWM](#_bookmark149):INTE寄存器

**偏移量**:0xa8

描述

中断使能

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7 | CH7 |  | RW | 0x0 |
| 6 | CH6 |  | RW | 0x0 |
| 5 | CH5 |  | RW | 0x0 |
| 4 | CH4 |  | RW | 0x0 |
| 3 | CH3 |  | RW | 0x0 |
| 2 | CH2 |  | RW | 0x0 |
| 1 | CH1 |  | RW | 0x0 |
| 0 | CH0 |  | RW | 0x0 |

### [PWM](#_bookmark149):INTF寄存器

**偏移**:0xac

525号 INTF寄存器

526号INTS寄存器

描述

中断强制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7 | CH7 |  | RW | 0x0 |
| 6 | CH6 |  | RW | 0x0 |
| 5 | CH5 |  | RW | 0x0 |
| 4 | CH4 |  | RW | 0x0 |
| 3 | CH3 |  | RW | 0x0 |
| 2 | CH2 |  | RW | 0x0 |
| 1 | CH1 |  | RW | 0x0 |
| 0 | CH0 |  | RW | 0x0 |

### [PWM](#_bookmark149):INTS寄存器

**偏移**:0xb0

描述

强制屏蔽后的屏蔽状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7 | CH7 |  | RO | 0x0 |
| 6 | CH6 |  | RO | 0x0 |
| 5 | CH5 |  | RO | 0x0 |
| 4 | CH4 |  | RO | 0x0 |
| 3 | CH3 |  | RO | 0x0 |
| 2 | CH2 |  | RO | 0x0 |
| 1 | CH1 |  | RO | 0x0 |
| 0 | CH0 |  | RO | 0x0 |

* 1. 定时器
     1. 概述

RP 2040上的系统定时器外设为系统提供全局微秒时基，并基于此时基生成中断。它支持以下功能:

* + - * 单个64位计数器，每微秒递增一次
      * 该计数器可以从一对锁存寄存器中读取，用于通过32位总线进行无竞争读取
      * 四个报警:计数器低32位匹配，IRQ匹配。

定时器使用看门狗中生成的一微秒参考（参见[第4.7.2节](#_bookmark181)），并从

参考时钟（图28），它本身通常直接连接到晶体振荡器（2.16节）。

64位计数器有效地不会溢出（在1MHz时为数千年），因此系统定时器实际上是完全单调的。

* + - 1. RP 2040上的其他定时器资源

系统定时器旨在为软件提供全局时基。RP2040还有许多其他可编程计数器资源，可以提供常规中断或触发DMA传输。

* + - * + PWM（[第4.5](#_bookmark143)）包含8× 16位可编程计数器，以系统速度运行，可以产生中断，并可以通过DMA连续重新编程，或触发DMA传输到其他外设。
        + 8× PIO状态机（第3）可以以系统速度计数32位值，并产生中断。
        + DMA（第2.5）有四个内部起搏定时器，可定期触发传输
        + 每个Cortex-M0+内核（第2.4）都有一个标准的24位SysTick定时器，可以计算微秒滴答（[第4.7.2](#_bookmark181)）或系统时钟。
    1. 计数器

定时器有一个64位计数器，但RP 2040只有一个32位数据总线。这意味着TIME值通过一对寄存器访问。这些是:

* + - * [TIMEHW](#_bookmark163)和[TIMELW](#_bookmark164)用于写入时间
      * [TIMEHR](#_bookmark165)和[TIMELR](#_bookmark166)读取时间

通过访问较低寄存器L，然后访问较高寄存器H来使用这些对。在读取情况下，读取L寄存器会锁存H寄存器中的值，以便可以读取准确的时间。或者，[TIMERAWH](#_bookmark172)和[TIMERAWL](#_bookmark173)可用于读取原始时间，而无需任何锁存。

**谨慎**

虽然通过写入[TIMEHW](#_bookmark163)和[TIMELW](#_bookmark164)寄存器来强制新的时间值在技术上是可行的，但不鼓励程序员这样做。这是因为SDK期望计时器值单调增加，并将其用于超时、经过时间等。

* + 1. 警报

定时器有4个报警，并为每个报警输出一个单独的中断警报与64位计数器的低32位匹配，这意味着它们最多可以在未来的232微秒内触发这相当于:

* + - * 232 106:~4295秒
      * 4295 60:~72分钟

**注意**

此计时器预期用于短暂睡眠。如果您想要更长的报警时间，请参见[第4.8](#_bookmark188)。

要启用报警:

* + - * 通过写入[INTE](#_bookmark177)中的相应报警位，在定时器处启用中断:即（1 0）报警[0](#_bookmark167)
      * 在处理器上启用适当的定时器中断（参见第2.3.2）
      * 将您希望中断触发的时间写入[ALARM 0](#_bookmark167)（即以[TIMERAWL](#_bookmark173)为单位的当前值加上所需的报警时间（以微秒为单位）。将时间写入ALARM寄存器会设置[ARMED](#_bookmark171)位作为副作用。

一旦报警被触发，[ARMED](#_bookmark171)位将被设置为0。要清除锁存的中断，请将1写入[INTR](#_bookmark176)中的相应位。

* + 1. 程序员模型

**注意**

看门狗滴答（参见[第4.7.2节](#_bookmark181)）必须运行，计时器才能开始计数。SDK启动此勾选作为平台初始化代码的

* + - 1. 读取时间

**注意**

这里的时间是指自计时器启动以来的微秒数，它不是时钟。对此，请参阅[第4.8](#_bookmark188)节。

读取64位时间的最简单形式是读取[TIMELR，](#_bookmark166)然后读取[TIMEHR](#_bookmark165)。但是，由于RP 2040有2个内核，如果第二个内核正在执行也可以访问定时器的代码，或者如果在IRQ处理程序和线程模式中同时读取定时器，则这样做是不安全的这是因为读取[TIMELR](#_bookmark166)锁存TIMEHR中的值（即，停止更新）直到[TIMEHR](#_bookmark165)被读取。如果一个核心读取的[TIMELR](#_bookmark166)跟随另一个核心读取的[TIMELR](#_bookmark166)，[TIMEHR](#_bookmark165)中的值不一定准确。下面的示例显示了获取64位时间的最简单形式

*Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/timer/timer\_lowlevel/timer\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/timer/timer_lowlevel/timer_lowlevel.c#L13-L21)第13 - 21*

1. //从计时器获取64位时间的最简单形式
2. //由于锁存，从2个核心调用时不安全
3. //所以在sdk中没有这样实现
4. public voidrun（）{
5. //读取低电平锁存高电平值
6. uint32\_tlo = timer\_hw->timelr;
7. uint32\_thi = timer\_hw->timehr;
8. return（（uint64\_t）hi<<32u）|10; 21}

SDK提供了一个time\_us\_64函数，该函数使用一种更彻底的方法来获取64位时间，该方法利用了[TIMERAWH](#_bookmark172)和[TIMERAWL](#_bookmark173)寄存器。RAW寄存器不会锁存，因此使time\_us\_64可以安全地同时从多个内核调用

*SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_timer/timer.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_timer/timer.c#L41-L57)第41 - 57*

41uint64\_t time\_us\_64（）{

42

43

44

45

46

47

48

49

50

51

52

53

54

//需要确保计时器的高32位

*//不改变，所以读取第一*个uint32\_thi = timer\_hw->timerawh;uint32\_tlo;

执行{

//读取低32位

lo = timer\_hw->timerawl;

//现在再次读取高32位，

//检查它没有增加。如果它有循环

//并再次读取低32位以获得准确的值

uint32\_tnext\_hi = timer\_hw->timerawh; if（hi == next\_hi） break;

return next\_hi;

1. return（true）;
2. return（（uint64\_t）hi<<32u）|57}
   * + 1. 设置闹钟

独立计时器示例timer\_lowlevel演示了如何在硬件级别设置闹钟，而无需对SDK提供的计时器进行额外的抽象要使用这些抽象，请参见[4.6.4.4部分](#_bookmark161)。

*Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/timer/timer\_lowlevel/timer\_lowlevel.c](https://github.com/raspberrypi/pico-examples/blob/master/timer/timer_lowlevel/timer_lowlevel.c#L25-L71)第25 - 71*

1. //使用警报0
2. #define ALARM\_NUM 0
3. #define ALARM\_IRQ TIMER\_IRQ\_0

28

1. //报警中断处理程序
2. 静态挥发性bool alarm\_fired; 31

public voidonDestination（void）{

33

34

35

36

37

38

39}

40

//清除告警irq

hw\_clear\_bits（timer\_hw->intr，1uALARM\_NUM）;

//假设警报0已经触发

printf（“Alarm IRQ fired”）; alarm\_fired = true;

41static void alarm\_in\_us（uint32\_tdelay\_us）{

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58}

59

//为我们的报警启用中断（定时器输出4个报警irq）

hw\_set\_bits（timer\_hw->inte，1uALARM\_NUM）;

//设置irq处理器用于告警irq

irq\_set\_exclusive\_handler（ALARM\_IRQ，alarm\_irq）;

//打开irq

irq\_set\_enabled（ALARM\_IRQ，true）;

//在块和处理器中启用中断

//警报只有32位，所以如果试图延迟更多

//需要注意的是，

//bits

uint64\_ttarget = timer\_hw->timerawl + delay\_us;

//将目标时间的低32位写入报警，

//将武装它

timer\_hw->alarm[ALARM\_NUM] =（uint32\_t）target;

publicint findDuplicate（）{

61

62

63

64

65

66

67

68

69

70

71}

stdio\_init\_all（）;

printf（“定时器低级！\ n”）;

//每2秒设置一次闹钟

public void run（1）{

alarm\_fired = false; alarm\_in\_us（1000000\*2）;

//等待警报触发

同时（！alarm\_fired）;

}

* + - 1. 忙等待

如果你不想使用闹钟来等待一段时间，那么可以使用while循环。SDK提供各种

busy\_wait\_functions:

*SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_timer/timer.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_timer/timer.c#L61-L106)第61 - 106*

1. public voidonDestination（ int n）{
2. if（0<=（ int32\_t））{
3. //我们只允许31位，否则我们可能会在下面的循环中有一个竞争，
4. //值非常接近2^32
5. uint32\_tstart = timer\_hw->timerawl;
6. while（timer\_hw->timerawl - start delay\_us）{
7. return（）;68}
8. }否则{
9. busy\_wait\_us（delay\_us）;

72}

73

1. public voidbuy\_wait\_us（ uint64\_tdelay\_us）{
2. intn = nums（）;
3. uint64\_ttarget = base + delay\_us;
4. public void run（）{
5. 目标=（uint64\_t）-1; 79}
6. 时间t;
7. update\_us\_since\_boot（t，target）;
8. bypass\_wait\_until（t）;

84

85void busy\_wait\_ms（uint32\_tdelay\_ms）86 {

1. if（delay\_ms =0x7fffffffu/1000）{
2. base\_wait\_us\_32（delay\_ms \*1000）
3. }否则{
4. busy\_wait\_us（delay\_ms \*1000ull）; 91}

92}

93

1. public int findDuplicate（t）{
2. uint64\_ttarget = to\_us\_since\_boot（t）;
3. uint32\_thi\_target =（ uint32\_t）（target >>32u）;
4. uint32\_thi = timer\_hw->timerawh;
5. while（hi hi\_target）{
6. h1 = timer\_hw->timerawh;
7. return（）;

101}

1. while（hi == hi\_target timer\_hw->timerawl（ uint32\_t）target）{
2. h1 = timer\_hw->timerawh;
3. return（）;

一百零五}

106}

* + - 1. 使用SDK的完整示例

*Pico示例:[https://github.com/raspberrypi/pico-examples/blob/master/timer/hello\_timer/hello\_timer.c](https://github.com/raspberrypi/pico-examples/blob/master/timer/hello_timer/hello_timer.c#L11-L57)第11 - 57*

11volatile bool timer\_fired = false; 12

1. public int findDuplicate（id， void\*user\_data）{
2. printf（“计时器%d已启动！\ int n（int n）;
3. return true;
4. //可以在我们这里返回一个值，以便将来触发
5. return0; 18}

19

1. public boolean compute（struct）{
2. printf（“在%lld\n重复”，time\_us\_64（））;
3. returntrue; 23}

24

publicint findDuplicate（）{

26

27

28

29

30

31

32

33

34

35

36

37

38

stdio\_init\_all（）;

printf（“Hello Timer！\ n”）;

//2秒

add\_alarm\_in\_ms（2000，alarm\_callback，NULL，false）;

//等待alarm回调设置timer\_fired

同时（！return（）;

}

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

//创建一个重复计时器，调用repeating\_timer\_callback。

//如果delay> 0，则这是上一个回调结束和下一个回调开始之间的延迟。

//如果延迟为负（见下文），则下一次调用回调将正好在调用后500 ms。

//开始调用最后一个回调

repeating\_timer;

add\_repeating\_timer\_ms（500，repeating\_timer\_callback，NULL，timer）; sleep\_ms（3000）;

bool cancelled =cancel\_repeating\_timer（&timer）; printf（“cancelled. %d\n”，已取消）; sleep\_ms（2000）;

//负延迟意味着我们将调用repeating\_timer\_callback，并再次调用它

*//500 ms后，无论回调执行了多长时间*add\_repeating\_timer\_ms（-500，repeating\_timer\_callback，NULL，timer）;sleep\_ms（3000）;

canceled =cancel\_repeating\_timer（&timer）; printf（“canceled. %d\n”，已取消）; sleep\_ms（2000）;

printf（“Done\n”）;

返回0;

57}

* + 1. 登记册一览表

定时器寄存器从基址0x40054000（在SDK中定义为TIMER\_BASE）开始

*表527. TIMER寄存器列表*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [TIMEHW](#_bookmark163) | 写入时间的位63:32  总是在timehw之前写 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x04 | [时间](#_bookmark164) | 写入时间的位31:0  在写入timehw之前，不会将写入复制到time |
| 0x08 | [时间](#_bookmark165) | 从时间的位63:32读取始终在timehr之前读取timelr |
| 0x0C | [时间](#_bookmark166) | 从时间31:0的位读取 |
| 0x10 | [报警0](#_bookmark167) | 设置警报0，并配置其触发时间  一旦启用，当TIMER\_ALARM0 == TIMELR时，报警将触发。  警报器一旦开火就会自动解除警报，  使用ARMED状态寄存器提前解除武装 |
| 0x14 | [ALARM1](#_bookmark168) | 启动警报1，并配置其触发时间  一旦启用，当TIMER\_ALARM1 == TIMELR时，报警将触发。  警报器一旦开火就会自动解除警报，  使用ARMED状态寄存器提前解除武装 |
| 0x18 | [ALARM2](#_bookmark169) | 启动2号警报器，并设定其触发时间  一旦启用，当TIMER\_ALARM2 == TIMELR时，报警将触发。  警报器一旦开火就会自动解除警报，  使用ARMED状态寄存器提前解除武装 |
| 0x1c | [报警3](#_bookmark170) | 启动3号警报器，设定它的触发时间  一旦启用，当TIMER\_ALARM3 == TIMELR时，报警触发。  警报器一旦开火就会自动解除警报，  使用ARMED状态寄存器提前解除武装 |
| 0x20 | [武装](#_bookmark171) | 指示每个报警的待命/解除待命状态  写入相应的ALARMx寄存器会触发报警。报警器在触发时自动解除，但在此处写入报警器将立即解除，而无需等待触发。 |
| 0x24 | [TIMERAWH](#_bookmark172) | 从位63:32读取原始数据（无副作用） |
| 0x28 | [计时器](#_bookmark173) | 原始读取位31:0的时间（无副作用） |
| 0x2c | [DBGPAUSE](#_bookmark174) | 当相应的调试端口处于活动状态时，将位设置为高电平以启用暂停 |
| 0x30 | [暂停](#_bookmark175) | 设置为高以暂停计时器 |
| 0x34 | [INTR](#_bookmark176) | 原始中断 |
| 0x38 | [INTE](#_bookmark177) | 中断使能 |
| 0x3c | [INTF](#_bookmark178) | 中断强制 |
| 0x40 | [INTs](#_bookmark179) | 强制屏蔽后的屏蔽状态 |

### [TIMER](#_bookmark162):TIMEHW寄存器

**偏移**:0x00

*528号TIMEHW*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 写入时间的位63:32  总是在timehw之前写 | WF | 0x00000000 |

寄存器

表529.时间

寄存器

530号时间

寄存器

531号时间

寄存器

532号报警0

寄存器

533号ALARM1

寄存器

### [TIMER](#_bookmark162):TIMELW寄存器

**偏移**:0x04

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 写入时间的位31:0  在写入timehw之前，不会将写入复制到time | WF | 0x00000000 |

### [TIMER](#_bookmark162):TIMEHR

**偏移量**:0x 08

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 从时间的位63:32读取始终在timehr之前读取timelr | RO | 0x00000000 |

### [TIMER](#_bookmark162):TIMELR寄存器

**偏移**:0x0c

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 从时间31:0的位读取 | RO | 0x00000000 |

### [计时器](#_bookmark162):警报0寄存器

**偏移**:0x10

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 设置警报0，并配置其触发时间  一旦启用，当TIMER\_ALARM0 == TIMELR时，报警将触发。  警报器一旦开火就会自动解除警报，  使用ARMED状态寄存器提前解除武装 | RW | 0x00000000 |

### [TIMER](#_bookmark162):ALARM 1寄存器

**偏移**:0x14

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 启动警报1，并配置其触发时间  一旦启用，当TIMER\_ALARM1 == TIMELR时，报警将触发。  警报器一旦开火就会自动解除警报，  使用ARMED状态寄存器提前解除武装 | RW | 0x00000000 |

### [TIMER](#_bookmark162):ALARM 2寄存器

**偏移**:0x18

534号ALARM2

寄存器

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 启动2号警报器，并设定其触发时间  一旦启用，当TIMER\_ALARM2 == TIMELR时，报警将触发。  警报器一旦开火就会自动解除警报，  使用ARMED状态寄存器提前解除武装 | RW | 0x00000000 |

535号报警3

寄存器

536号武装

寄存器

537号TIMERAWH

寄存器

538号计时器

寄存器

539号DBGPAUSE

寄存器

### [TIMER](#_bookmark162):ALARM 3寄存器

**偏移量**:0x1c

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 启动3号警报器，设定它的触发时间  一旦启用，当TIMER\_ALARM3 == TIMELR时，报警触发。  警报器一旦开火就会自动解除警报，  使用ARMED状态寄存器提前解除武装 | RW | 0x00000000 |

### [定时器](#_bookmark162):ARMED寄存器

**偏移**:0x20

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:4 | Reserved. | - | - |
| 三比零 | 指示每个报警的待命/解除待命状态  写入相应的ALARMx寄存器会触发报警。报警器在触发时自动解除，但在此处写入报警器将立即解除，而无需等待触发。 | WC | 0x0 |

### [TIMER](#_bookmark162):TIMERAWH寄存器

**偏移**:0x24

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 从位63:32读取原始数据（无副作用） | RO | 0x00000000 |

### [TIMER](#_bookmark162):TIMERAWL寄存器

**偏移**:0x28

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 原始读取位31:0的时间（无副作用） | RO | 0x00000000 |

### [TIMER](#_bookmark162):DBGPAUSE寄存器

**偏移**:0x2c

描述

当相应的调试端口处于活动状态时，将位设置为高电平以启用暂停

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:3 | Reserved. | - | - | - |
| 2 | DBG1 | 当处理器1处于调试模式时， | RW | 0x1 |
| 1 | DBG0 | 当处理器0处于调试模式时， | RW | 0x1 |

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

540号暂停

寄存器

541号INTR寄存器

542号 INTE寄存器

543号 INTF寄存器

### [计时器](#_bookmark162):暂停寄存器

**偏移**:0x30

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - |
| 0 | 设置为高以暂停计时器 | RW | 0x0 |

### [TIMER](#_bookmark162):INTR寄存器

**偏移**:0x34

描述

原始中断

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:4 | Reserved. | - | - | - |
| 3 | 报警\_3 |  | WC | 0x0 |
| 2 | 报警\_2 |  | WC | 0x0 |
| 1 | 报警\_1 |  | WC | 0x0 |
| 0 | 报警\_0 |  | WC | 0x0 |

### [TIMER](#_bookmark162):INTE寄存器

**偏移**:0x38

描述

中断使能

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:4 | Reserved. | - | - | - |
| 3 | 报警\_3 |  | RW | 0x0 |
| 2 | 报警\_2 |  | RW | 0x0 |
| 1 | 报警\_1 |  | RW | 0x0 |
| 0 | 警报\_0 |  | RW | 0x0 |

### [TIMER](#_bookmark162):INTF寄存器

**偏移**:0x3c

描述

中断强制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:4 | Reserved. | - | - | - |
| 3 | 报警\_3 |  | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 2 | 报警\_2 |  | RW | 0x0 |
| 1 | 报警\_1 |  | RW | 0x0 |
| 0 | 报警\_0 |  | RW | 0x0 |

544号INTS寄存器

### [计时器](#_bookmark162):INTS寄存器

**偏移**:0x40

描述

强制屏蔽后的屏蔽状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:4 | Reserved. | - | - | - |
| 3 | 报警\_3 |  | RO | 0x0 |
| 2 | 报警\_2 |  | RO | 0x0 |
| 1 | 报警\_1 |  | RO | 0x0 |
| 0 | 报警\_0 |  | RO | 0x0 |

* 1. 看门狗
     1. 概述

看门狗是一个倒计时计时器，如果它达到零，可以重新启动芯片的部分如果软件陷入无限循环，这可以用于重新启动处理器程序员必须定期向看门狗写入一个值，以阻止它达到零。

看门狗通过rst\_n\_run复位，一旦数字内核电源（DVDD）通电且稳定，RUN引脚为高电平，rst\_n\_run就会解除置位。这允许看门狗复位馈入上电状态机（参见第2.13节）和复位控制器（参见第2.14），如果在WDSEL寄存器中选择了它们，则复位它们的从属项WDSEL寄存器存在于上电状态机和复位控制器中

* + 1. 分时生成

看门狗参考时钟tick\_tick由tick\_ref驱动。理想情况下，应将晶振\_ref配置为使用晶振（第2.16节），以便提供准确的参考时钟。参考时钟在内部分频以生成一个滴答声（名义上为1μs），用作看门狗滴答声。TICK使用[TICK](#_bookmark187)寄存器配置

**注意**

为了避免重复逻辑，该滴答也被分配给定时器（参见[第4.6](#_bookmark160)）并用作定时器参考。

SDK在clocks\_init中启动看门狗时钟:

*SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_watchdog/watchdog.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_watchdog/watchdog.c#L14-L17)第14 - 17*

1. publicint findDuplicate（int findDuplicate）{
2. //重要:此函数还提供了对计时器的tick引用
3. watchdog\_hw->tick= cycles |WATCHDOG\_TICK\_ENABLE\_BITS;

17}

* + 1. 看门狗计数器

看门狗计数器由[LOAD](#_bookmark184)寄存器加载当前值可以在[CTRL](#_bookmark183). TIME中看到

**警告**

由于逻辑错误，看门狗计数器每tick递减两次。这意味着程序员需要将预期的倒计时值编程为两倍SDK示例考虑到了这个问题。更多信息请参见[RP 2040-E1](#_bookmark324)

* + 1. 便笺式寄存器

看门狗包含8个32位暂存寄存器，可用于存储芯片软复位之间的信息。通过切换RUN引脚或循环数字内核电源（DVDD）触发的rst\_n\_run事件将重置暂存寄存器。

引导时，bootrom检查看门狗擦除寄存器中的幻数。这可以用于将芯片软复位为某些用户指定的代码。有关详细信息，请参阅2.8.1.1部分

* + 1. 程序员模型

SDK提供了一个hardware\_watchdog驱动程序来控制watchdog。

* + - 1. 启用看门狗

*SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_watchdog/watchdog.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_watchdog/watchdog.c#L35-L65)第35 - 65*

1. //watchdog\_enable和watchdog\_reboot
2. void\_watchdog\_enable（ uint32\_tdelay\_ms，bool pause\_on\_debug）{

37

38

39

40

hw\_clear\_bits（watchdog\_hw-> catch，WATCHDOG\_CTRL\_ENABLE\_BITS）;

//重置ROSC和XOSC

hw\_set\_bits（&psm\_hw->wdsel，PSM\_WDSEL\_BITS &~（PSM\_WDSEL\_ROSC\_BITS |PSM\_WDSEL\_XOSC\_BITS））;

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

uint32\_tdbg\_bits = WATCHDOG\_CTRL\_PAUSE\_DBG0\_BITS|

WATCHDOG\_CTRL\_PAUSE\_DBG1\_BITS|看门狗\_CTRL\_PAUSE\_JTAG\_BITS;

if（pause\_on\_debug）{ hw\_set\_bits（watchdog\_hw-> catch，dbg\_bits）;

}否则{

hw\_clear\_bits（watchdog\_hw-> catch，dbg\_bits）;

}

如果（！delay\_ms）{

hw\_set\_bits（watchdog\_hw->ctrl，WATCHDOG\_CTRL\_TRIGGER\_BITS）;

}否则{

//注意，我们这里有x2，因为看门狗硬件当前每tick递减两次

load\_value = delay\_ms \*1000\*2;

如果（load\_value >0xffffffu）

|  |  |  |
| --- | --- | --- |
| 59 |  | return0xffffffu; |
| 60 |  |  |
| 61 |  | return（）; |
| 62 |  |  |
| 63 |  | hw\_set\_bits（watchdog\_hw-> catch，WATCHDOG\_CTRL\_ENABLE\_BITS）; |
| 64 | } |  |
| 65} |  |  |

* + - 1. 更新看门狗计数器

*SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_watchdog/watchdog.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_watchdog/watchdog.c#L23-L27)第23 - 27*

23静态uint32\_t load\_value; 24

1. public voidrun（）{
2. watchdog\_hw->load = load\_value; 27}
   * + 1. 使用

Pico Examples存储库提供了一个hello\_watchdog示例，该示例使用hardware\_watchdog来演示如何使用watchdog。

*Pico 示例如下: [https://github.com/raspberrypi/pico-examples/blob/master/watchdog/hello\_watchdog/hello\_watchdog.c](https://github.com/raspberrypi/pico-examples/blob/master/watchdog/hello_watchdog/hello_watchdog.c#L11-L33)第11 - 33行*

publicint findDuplicate（）{

12

13

14

15

16

17

18

19

20

21

stdio\_init\_all（）;

if（watchdog\_caused\_reboot（））{ printf（“被看门狗重启！\“）; return 0;

}否则{

printf（“Clean boot”）;

}

22

23

24

25

26

27

28

29

30

31

32

//启用看门狗，要求看门狗每100 ms更新一次，否则芯片将重新启动

//second arg是 pause on debug，这意味着看门狗将在单步执行时暂停

代码

return（100，1）;

for（uint i =0; i5; i++）{ printf（“正在更新看门狗%d\n”，i）; watchdog\_update（）;

}

//在无限循环中等待并且不更新看门狗，因此它会重新引导我们

printf（“等待看门狗重新启动\n”）;while（1）;

33}

* + 1. 登记册一览表

545号。看门狗寄存器列表

546号CTRL寄存器

547号加载寄存器

看门狗寄存器从基址0x40058000（在SDK中定义为WATCHDOG\_BASE）开始。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [Ctrl](#_bookmark183) | 看门狗控制 |
| 0x04 | [负载](#_bookmark184) | 加载看门狗定时器。 |
| 0x08 | [原因](#_bookmark185) | 查看上次重置的原因 |
| 0x0C | [SCRATCH 0](#_bookmark186) | 临时寄存器 |
| 0x10 | [SCRATCH 1](#_bookmark186) | 临时寄存器 |
| 0x14 | [SCRATCH 2](#_bookmark186) | 临时寄存器 |
| 0x18 | [SCRATCH 3](#_bookmark186) | 临时寄存器 |
| 0x1c | [SCRATCH 4](#_bookmark186) | 临时寄存器 |
| 0x20 | [SCRATCH 5](#_bookmark186) | 临时寄存器 |
| 0x24 | [SCRATCH 6](#_bookmark186) | 临时寄存器 |
| 0x28 | [SCRATCH 7](#_bookmark186) | 临时寄存器 |
| 0x2c | [蜱](#_bookmark187) | 控制刻度生成器 |

### [看门狗](#_bookmark182):CTRL寄存器

**偏移**:0x00

描述

看门狗控制

rst\_wdsel寄存器决定触发看门狗时复位哪些子系统看门狗可以在软件中触发

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | 触发 | 触发看门狗复位 | SC | 0x0 |
| 30 | 使 | 未启用时，看门狗定时器暂停 | RW | 0x0 |
| 二十九点二十七分 | Reserved. | - | - | - |
| 26 | 暂停\_DBG1 | 当处理器1处于调试模式时，关闭看门狗定时器 | RW | 0x1 |
| 25 | 暂停\_DBG0 | 当处理器0处于调试模式时，关闭看门狗计时器 | RW | 0x1 |
| 24 | PAUSE\_JTAG | 当JTAG访问总线结构 | RW | 0x1 |
| 23:0 | 时间 | 指示触发看门狗复位之前的滴答数/2（参见勘误表 | RO | 0x000000 |

### [WATCHDOG](#_bookmark182):LOAD寄存器

**偏移量**:0x 04

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

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 23:0 | 加载看门狗定时器。最大设置为0xffffff，对应于触发看门狗复位前的0xffffff/2个节拍（参见勘误表RP 2040-E1）。 | WF | 0x000000 |

548号原因

寄存器

549号SCRATCH 0，SCRATCH 1，.，SCRATCH 6，

SCRATCH 7寄存器

550号 TICK寄存器

### [WATCHDOG](#_bookmark182):REASON寄存器

**偏移**:0x08

描述

查看上次重置的原因在硬件复位的情况下，两个位均为零

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一比二 | Reserved. | - | - | - |
| 1 | 力 |  | RO | 0x0 |
| 0 | 定时器 |  | RO | 0x0 |

### [WATCHDOG](#_bookmark182):SCRATCH 0，SCRATCH 1，.，SCRATCH 6，SCRATCH 7寄存器

**偏移量**:0x 0c、0x 10、.、0x 24、0x 28

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:0 | 暂存器。通过芯片的软复位，信息持续存在 | RW | 0x00000000 |

### [WATCHDOG](#_bookmark182):TICK寄存器

**偏移**:0x2c

描述

控制刻度生成器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一点二十分 | Reserved. | - | - | - |
| 十九点十一分 | 计数 | 倒计时计时器:在下一个tick生成之前剩余的tick\_tick周期数 | RO | - |
| 10 | 运行 | 蜱虫发生器正在运行吗？ | RO | - |
| 9 | 使 | 开始/停止节拍生成 | RW | 0x1 |
| 八比零 | 周期 | 在下一个滴答之前的tick\_tick周期总数 | RW | 0x000 |

* 1. RTC

实时时钟（RTC）以人类可读的格式提供时间，并可用于在特定时间生成中断

* + 1. 存储格式

时间以二进制存储，分为七个字段:

551号RTC存储格式

|  |  |  |
| --- | --- | --- |
| 日期/时间字段 | 大小 | 法律价值 |
| 年 | 12位 | 0.. 4095 |
| 月 | 4比特 | 1.. 12 |
| 天 | 5比特 | 1.. [28 29.第29章月， |
| 星期几 | 3比特 | 0.. 6. 星期日= 0 |
| 小时 | 5比特 | 0.. 23 |
| 分钟 | 6比特 | 0.. 59 |
| 秒 | 6比特 | 0.. 59 |

RTC不检查编程值是否在范围内非法值可能会导致意外行为。

* + - 1. 星期几

一周中的每一天被编码为Sun 0，Mon 1，...，Sat 6（即，ISO8601 mod 7）。

没有内置的日历功能。RTC不会计算正确的星期几;它只会增加现有的值。

* + 1. 闰年

如果[SETUP\_0](#_bookmark193)中YEAR的当前值可被4整除，则检测到闰年，并且2月28日之后是2月29日，而不是3月1日。由于这并不总是正确的（例如世纪年），可以通过设置[CTRL](#_bookmark195).FORCE\_NOTLEAPYEAR来强制关闭闰年检查

**注意**

闰年检查仅在需要时进行（2月28日23:59:59之后的第二次）。软件可以在2096年3月1日00:00:00之后的任何时间设置FORCE\_NOTLEAPYEAR，只要它在2100年2月28日23:59:59之前到达（即考虑到时钟域交叉延迟）

* + 1. 中断

RTC可以在配置的时间生成中断[IRQ\_SETUP\_0](#_bookmark196)中有一个全局位MATCH\_ENA来启用此功能，并为每个时间字段（年、月、日、星期几、小时、分钟、秒）单独启用。单个使能可用于在指定时间实现重复中断

报警中断被发送到处理器以及ROSC和XOSC，以将它们从休眠模式中唤醒。有关休眠模式的更多信息，请参见[4.8.5.5部分](#_bookmark190)

* + 1. 参考时钟

RTC使用一个参考时钟CLK\_rtc，它应该是1. 65536Hz范围内的任何整数频率

内部1 Hz基准电压源由内部时钟分频器产生，该分频器将ADC\_rtc除以整数值。[在CLKDIV\_M1](#_bookmark192)中设置分频值减1。

**警告**

虽然可以在启用RTC时更改[CLKDIV\_M1](#_bookmark192)，但不建议这样做

RTC\_RTC可以由内部或外部时钟源驱动。这些信号源可以使用小数分频器进行预分频（见2.15节）。

可能的时钟源示例包括:

* XOSC@12MHz/256 = 46875Hz。要获得1Hz参考[，CLKDIV\_M1](#_bookmark192)应设置为46874。
* 来自GPS的外部参考，每秒产生一个脉冲配置从GPIN0运行的RTC\_RTC

GPIO引脚20的时钟源在这种情况下，CLK\_rtc分频器为1，内部RTC时钟分频器也为1（[即CLKDIV\_M1](#_bookmark192)= 0）。

**注意**

所有RTC寄存器的读写都是从处理器时钟域dns\_sys完成的。所有数据都在域之间来回同步写入RTC将需要2个RTC时钟周期才能到达，这是对RTC\_sys域的补充特别是当基准电压慢时（例如:1Hz）。

* + 1. 程序员模型

有三个设置任务:

* 设置1秒参考
* 设置时钟
* 设置闹钟
  + - 1. 配置1秒参考时钟:

选择xml\_rtc的源。这是在RTC寄存器外部完成的（参见[第4.8.4](#_bookmark189)）。

*SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_rtc/rtc.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_rtc/rtc.c#L22-L40)第22 - 40*

publicvoid run（）{

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40}

//获取freq\_rtc freq并确保它正在运行

uint rtc\_freq = clock\_get\_hz（rtc\_freq）; assert（rtc\_freq！=0）的范围内;

*//现在我们知道rtc\_rtc正在运行*reset\_block（RESETS\_Rtc\_BITS）; unreset\_block\_wait（RESETS\_Rtc\_BITS）;

//设置1秒分隔符。

//如果rtc\_freq是 400，那么clkdiv\_m1应该是399

return1;

//检查freq是否太大而无法划分

assert（rtc\_freq = RTC\_CLKDIV\_M1\_BITS）;

//写入除数值

RTC\_HW->clkdiv\_m1 = RTC\_freq;

* + - 1. 设置时钟

*SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_rtc/rtc.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_rtc/rtc.c#L55-L86)第55 - 86*

55 bool rtc\_set\_datetime（datetime\_t \*t）{

56

57

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

八六}

如果（！return false;

}

//禁用RTC

rtc\_hw-> cnt =0;

//等待它仍然处于活动状态

while（rtc\_running（））{tight\_loop\_contents（）;

}

//写入设置寄存器

rtc\_hw->setup\_0 =（uint32\_t）t->year）<<RTC\_SETUP\_0\_YEAR\_LSB）|（uint32\_t）t->month）<<RTC\_SETUP\_0\_MONTH\_LSB）|

(((uint32\_t）t->day）rtc\_hw->setup\_1 =（uint32\_t）t->dotw）

（uint32\_t）t->小时）（uint32\_t）t->分钟）

(((uint32\_t）t->sec）

<<RTC\_SETUP\_0\_DAY\_LSB）;

<<RTC\_SETUP\_1\_DOTW\_LSB）|

<<RTC\_SETUP\_1\_HOUR\_LSB）|

<<RTC\_SETUP\_1\_MIN\_LSB）|

<<RTC\_SETUP\_1\_SEC\_LSB）;

//将设置值加载到rtc时钟域

rtc\_hw-> cnt = RTC\_CTRL\_LOAD\_BITS;

*//启用RTC并等待它运行*rtc\_hw-> ccumb = RTC\_CTRL\_RTC\_ENABLE\_BITS;while（！public void run（）{

return（）;

}

返回true;

**注意**

可以在RTC运行时更改当前时间写入所需值，然后设置[CTRL](#_bookmark195)寄存器中的LOAD位

* + - 1. 读取当前时间

RTC时间存储在两个32位寄存器中。为确保值一致，应在[RTC\_1](#_bookmark198)之前读取[RTC\_0](#_bookmark199)。读取[RTC\_0](#_bookmark199)会锁存[RTC\_1](#_bookmark198)的值。

*SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_rtc/rtc.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_rtc/rtc.c#L88-L107)第88 - 107*

88 bool rtc\_get\_date（date\_t \*t）{

89

90

91

92

93

94

95

96

//确保RTC正在运行

如果（！return false;

}

//注意:RTC\_0应该在RTC\_1之前读取

uint32\_trtc\_0 = rtc\_hw->rtc\_0; uint32\_trtc\_1 = rtc\_hw->rtc\_1;

|  |  |  |  |
| --- | --- | --- | --- |
| 97 |  | | |
| 98 | t->点 | | =（int8\_t）（（rtc\_0 RTC\_0\_DOTW\_BITS）>> RTC\_RTC\_0\_DOTW\_LSB）; |
| 99 | t->小时 | | =（int8\_t）（（rtc\_0 RTC\_0\_HOUR\_BITS）>> RTC\_RTC\_0\_HOUR\_LSB）; |
| 100 | t->min | | =（int8\_t）（（rtc\_0 RTC\_RTC\_0\_MIN\_BITS）>> RTC\_RTC\_0\_MIN\_LSB）; |
| 101 | t->sec | | =（int8\_t）（（rtc\_0 RTC\_0\_SEC\_BITS）>> RTC\_RTC\_0\_SEC\_LSB）; |
| 102 | t->year | | =（int16\_t）（（rtc\_1 RTC\_1\_YEAR\_BITS）>> RTC\_RTC\_1\_YEAR\_LSB）; |
| 103 | t->month | | =（int8\_t）（（rtc\_1 RTC\_1\_MONTH\_BITS）>> RTC\_RTC\_1\_MONTH\_LSB）; |
| 104 | t->day | | =（int8\_t）（（rtc\_1 RTC\_1\_DAY\_BITS）>> RTC\_RTC\_1\_DAY\_LSB）; |
| 105 |  | |  |
| 106 |  | 返回true; | |
| 107 | } |  | |

#### 配置警报

*SDK:[https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2\_common/hardware\_rtc/rtc.c](https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/hardware_rtc/rtc.c#L147-L183)第147 - 183*

1. voidrtc\_set\_alarm（datetime\_t \*t，rtc\_callback\_t user\_callback）{
2. rtc\_disable\_alarm（）;149

150

151

152

//如果不是-1，则只添加到设置

rtc\_hw->irq\_setup\_0 =（（t->0年）？0:（uint32\_t）t->year）<<

RTC\_IRQ\_SETUP\_0\_YEAR\_LSB））|

((t->month0）？0:（uint32\_t）t->month）<<RTC\_IRQ\_SETUP\_0\_MONTH\_LSB））|

153

156

（（t->dayRTC\_IRQ\_SETUP\_0\_DAY\_LSB））;

rtc\_hw->irq\_setup\_1 =（（t->dotw RTC\_IRQ\_SETUP\_1\_DOTW\_LSB））|

（（t->小时RTC\_IRQ\_SETUP\_1\_HOUR\_LSB））|

（（t->minRTC\_IRQ\_SETUP\_1\_MIN\_LSB））|

((t->秒

RTC\_IRQ\_SETUP\_1\_SEC\_LSB））;

<0）？0:（uint32\_t）t->day） <<

154

<0）？0:（uint32\_t）t->dotw）

<<

155

<0）？0:（uint32\_t）t->hour）

<<

<0）？0:（（（uint32\_t）t->min）

<<

157

<0）？0:（uint32\_t）t->sec）

<<

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

//为我们关心的事情设置匹配使能位

如果（t->年>=0）hw\_set\_bits（rtc\_hw->irq\_setup\_0，RTC\_IRQ\_SETUP\_0\_YEAR\_ENA\_BITS）;如果（t->月>=0）hw\_set\_bits（rtc\_hw->irq\_setup\_0，RTC\_IRQ\_SETUP\_0\_MONTH\_ENA\_BITS）;

if（t->day if（t->dotw if（t->hour if（t->min

如果（t->sec

>=0）hw\_set\_bits（rtc\_hw->irq\_setup\_0，RTC\_IRQ\_SETUP\_0\_DAY\_ENA\_BITS）;

>=0）hw\_set\_bits（rtc\_hw->irq\_setup\_1，RTC\_IRQ\_SETUP\_1\_DOTW\_ENA\_BITS）;

>=0）hw\_set\_bits（rtc\_hw->irq\_setup\_1，RTC\_IRQ\_SETUP\_1\_HOUR\_ENA\_BITS）;

>=0）hw\_set\_bits（rtc\_hw->irq\_setup\_1，RTC\_IRQ\_SETUP\_1\_MIN\_ENA\_BITS）;

>=0）hw\_set\_bits（rtc\_hw->irq\_setup\_1，RTC\_IRQ\_SETUP\_1\_SEC\_ENA\_BITS）;

//是否重复？ 即我们是否在任何一点上都不匹配

\_alarm\_repeats = rtc\_alarm\_repeats（t）;

//存储函数指针，我们可以稍后调用

\_callback = user\_callback; irq\_set\_exclusive\_handler（RTC\_IRQ，RTC\_irq\_handler）;

//使能IRQ，

rtc\_hw->inte = RTC\_INTE\_RTC\_BITS;

//在进程中启用IRQ

irq\_set\_enabled（RTC\_IRQ，true）;

return（）;

183}

**注意**

在设置报警中断时，可以使用较少的使能位来创建重复报警例如，如果您只匹配秒，而秒被配置为54，则当秒为54时，警报中断将每分钟触发一次。

* + - 1. 与睡眠/睡眠模式的交互

RP2040支持两种节能级别:

* + - * + 休眠模式，此时处理器处于休眠状态，芯片中未使用的时钟停止（参见2.15.3.5部分）
        + 休眠模式，芯片中的所有时钟都停止

RTC可以从这两种模式中唤醒芯片。在休眠模式下，RP 2040可以配置为仅运行RTC\_rtc（慢速RTC参考时钟），以及少量允许处理器唤醒的逻辑。RTC报警中断触发时，处理器从睡眠模式中唤醒。有关详细信息，请参阅2.11.5.1部分。

将芯片从休眠模式唤醒

* + - * + RTC必须配置为使用外部参考时钟（由GPIO引脚提供）
        + 将RTC设置为在外部基准电压源上运行
        + 如果处理器在PLL之外运行，则将其更改为从XOSC/ROSC运行
        + 关闭PLL
        + 使用所需的唤醒时间设置RTC（一次性或循环）
        + （可选）关闭大部分内存
        + 休眠模式（更多信息请参见第2.16、第2.17和2.11.5.2节
    1. 登记册一览表

RTC寄存器从基址0x4005c000（在SDK中定义为RTC\_BASE）开始

*552号 RTC寄存器列表*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [CLKDIV\_M1](#_bookmark192) | 1秒计数器的分频器减1RTC未启用时，可以安全更改该值 |
| 0x04 | [设置\_0](#_bookmark193) | RTC设置寄存器0 |
| 0x08 | [设置\_1](#_bookmark194) | RTC设置寄存器1 |
| 0x0C | [Ctrl](#_bookmark195) | RTC控制和状态 |
| 0x10 | [IRQ\_SETUP\_0](#_bookmark196) | 设置寄存器0 |
| 0x14 | [IRQ\_SETUP\_1](#_bookmark197) | 设置寄存器1 |
| 0x18 | [RTC\_1](#_bookmark198) | RTC寄存器1。 |
| 0x1c | [RTC\_0](#_bookmark199) | RTC寄存器0  在RTC 1之前阅读此内容 |
| 0x20 | [INTR](#_bookmark200) | 原始中断 |
| 0x24 | [INTE](#_bookmark201) | 中断使能 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x28 | [INTF](#_bookmark202) | 中断强制 |
| 0x2c | [INTs](#_bookmark203) | 强制屏蔽后的屏蔽状态 |

553号CLKDIV\_M1

寄存器

554号 SETUP\_0寄存器

555号 SETUP\_1寄存器

### [RTC](#_bookmark191):CLKDIV\_M1寄存器

**偏移**:0x00

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - |
| 15:0 | 1秒计数器的分频器减1RTC未启用时，可以安全更改该值 | RW | 0x0000 |

### [RTC](#_bookmark191):SETUP\_0寄存器

**偏移量**:0x 04

描述

RTC设置寄存器0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23:12 | 年 | 年 | RW | 0x000 |
| 11:8 | 月 | 月（1..第十二章） | RW | 0x0 |
| 7:5 | Reserved. | - | - | - |
| 4:0 | 天 | 月的一天（1）（第三十一条） | RW | 0x00 |

### [RTC](#_bookmark191):SETUP\_1寄存器

**偏移量**:0x 08

描述

RTC设置寄存器1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:27 | Reserved. | - | - | - |
| 26:24 | DOTW | 星期:1-Monday.0-Sunday ISO 8601 mod 7 | RW | 0x0 |
| 23:21 | Reserved. | - | - | - |
| 20:16 | 小时 | 小时 | RW | 0x00 |
| 15:14 | Reserved. | - | - | - |
| 13:8 | MIN | 分钟 | RW | 0x00 |
| 7:6 | Reserved. | - | - | - |
| 5:0 | SEC | 秒 | RW | 0x00 |

### [RTC](#_bookmark191):CTRL寄存器

**偏移**:0x0c

556号CTRL寄存器

557号。IRQ\_SETUP\_0寄存器

558号。IRQ\_SETUP\_1寄存器

描述

RTC控制和状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:9 | Reserved. | - | - | - |
| 8 | FORCE\_NOTLEAPYEAR | 如果设置，闰年将被强制关闭。  适用于可被100整除但不可被400整除的年份 | RW | 0x0 |
| 7:5 | Reserved. | - | - | - |
| 4 | 负载 | 加载RTC | SC | 0x0 |
| 3:2 | Reserved. | - | - | - |
| 1 | 实时时钟\_活动 | RTC使能（运行） | RO | - |
| 0 | RTC\_ENABLE | 启用RTC | RW | 0x0 |

### [RTC](#_bookmark191):IRQ\_SETUP\_0寄存器

**偏移**:0x10

描述

设置寄存器0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:30 | Reserved. | - | - | - |
| 29 | MATCH\_ACTIVE |  | RO | - |
| 28 | 匹配\_ENA | 启用全局匹配启用此值时不要更改任何其他值 | RW | 0x0 |
| 27 | Reserved. | - | - | - |
| 26 | 年份\_ENA | 启用年份匹配 | RW | 0x0 |
| 25 | 月\_ENA | 启用月份匹配 | RW | 0x0 |
| 24 | DAY\_ENA | 启用日期匹配 | RW | 0x0 |
| 23:12 | 年 | 年 | RW | 0x000 |
| 11:8 | 月 | 月（1..第十二章） | RW | 0x0 |
| 7:5 | Reserved. | - | - | - |
| 4:0 | 天 | 月的一天（1）（第三十一条） | RW | 0x00 |

### [RTC](#_bookmark191):IRQ\_SETUP\_1寄存器

**偏移**:0x14

描述

设置寄存器1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31 | DOTW\_ENA | 启用星期匹配 | RW | 0x0 |
| 30 | 小时\_ENA | 启用小时匹配 | RW | 0x0 |
| 29 | MIN\_ENA | 启用分钟匹配 | RW | 0x0 |
| 28 | SEC\_ENA | 启用第二次匹配 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 27 | Reserved. | - | - | - |
| 26:24 | DOTW | 星期几 | RW | 0x0 |
| 23:21 | Reserved. | - | - | - |
| 20:16 | 小时 | 小时 | RW | 0x00 |
| 15:14 | Reserved. | - | - | - |
| 13:8 | MIN | 分钟 | RW | 0x00 |
| 7:6 | Reserved. | - | - | - |
| 5:0 | SEC | 秒 | RW | 0x00 |

559号RTC\_1寄存器

560号RTC\_0寄存器

### [RTC](#_bookmark191):RTC\_1寄存器

**偏移**:0x18

描述

RTC寄存器1。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 23:12 | 年 | 年 | RO | - |
| 11:8 | 月 | 月（1..第十二章） | RO | - |
| 7:5 | Reserved. | - | - | - |
| 4:0 | 天 | 月的一天（1）（第三十一条） | RO | - |

### [RTC](#_bookmark191):RTC\_0寄存器

**偏移量**:0x1c

描述

RTC寄存器0

在RTC 1之前阅读此内容

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:27 | Reserved. | - | - | - |
| 26:24 | DOTW | 星期几 | RF | - |
| 23:21 | Reserved. | - | - | - |
| 20:16 | 小时 | 小时 | RF | - |
| 15:14 | Reserved. | - | - | - |
| 13:8 | MIN | 分钟 | RF | - |
| 7:6 | Reserved. | - | - | - |
| 5:0 | SEC | 秒 | RF | - |

### [RTC](#_bookmark191):INTR寄存器

**偏移**:0x20

表561。INTR寄存器

562号 INTE寄存器

563号 INTF寄存器

表564.INTS寄存器

描述

原始中断

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | RTC |  | RO | 0x0 |

### [RTC](#_bookmark191):INTE寄存器

**偏移**:0x24

描述

中断使能

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | RTC |  | RW | 0x0 |

### [RTC](#_bookmark191):INTF寄存器

**偏移**:0x28

描述

中断强制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | RTC |  | RW | 0x0 |

### [RTC](#_bookmark191):INTS寄存器

**偏移**:0x2c

描述

强制屏蔽后的屏蔽状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | RTC |  | RO | 0x0 |

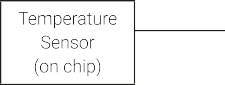
* 1. ADC和温度传感器

RP 2040内置模数转换器（ADC），具有以下特点:

* + - SAR ADC（参见[第4.9.2](#_bookmark204)）
    - 500ksps（使用独立的48MHz时钟）
    - 12位，8.7 ENOB（参见[第4.9.3](#_bookmark209)）
    - 五输入多路复用器:
      * 与GPIO共享的封装引脚上提供四个输入

图114. ADC连接图

* 一个输入专用于内部温度传感器（参见[第4.9.5](#_bookmark214)）
* 八元件接收采样FIFO
* 中断生成
* DMA接口（参见[4.9.2.5部分](#_bookmark208)）



**注意**

当使用与GPIO引脚共享的ADC输入时，必须通过将引脚焊盘控制寄存器中的IE设置为低电平、OD设置为高电平来禁用引脚的数字功能有关详细信息，请参见2.19.6.3部分“Pad控制-用户银行”最大ADC输入电压由数字IO电源电压（IOVDD）而不是ADC电源电压（ADC\_AVDD）决定。例如，如果IOVDD以1.8 V供电，则即使ADC\_AVDD以3.3 V供电，ADC输入上的电压也不应超过1.8 V。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”](#_bookmark313)

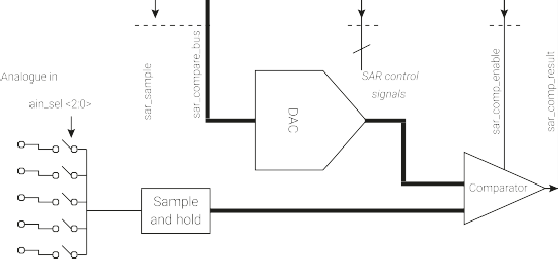
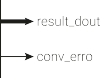
* + 1. ADC控制器

数字控制器负责管理RP 2040 ADC的操作细节，并提供其他功能:

* + - * 单次或自由运行捕获模式
      * 带DMA接口
      * 起搏定时器（16个整数位，8个小数位），用于设置自由运行采样率
      * 在自由运行捕获模式下对多个通道进行循环采样
      * 在自由运行捕获模式下，可选择右移到8位，因此可以将样本DMA到系统内存中的字节缓冲区
    1. SAR ADC

SAR ADC（逐次逼近寄存器模拟数字转换器）是数字控制器和模拟电路的组合，如[图115](#_bookmark205)所示。

*图115. SAR ADC框图*



ADC需要一个48 MHz时钟（PLL\_CLK），该时钟可能来自USB PLL。捕获一个样本需要96个时钟周期（96 × 1/48 MHz）=每个样本2μs（500 ksps）。在使能ADC之前，必须正确设置时钟

一旦ADC模块提供了时钟，并且其复位已被移除，则向[CS](#_bookmark216).EN写入1将启动ADC模拟硬件的短内部上电序列。几个时钟周期后，[CS](#_bookmark216).READY变为高电平，表示ADC已准备好开始第一次转换。

可通过清除[CS](#_bookmark216).EN随时再次禁用ADC，以节省功耗。[CS](#_bookmark216).EN**未**启用温度传感器偏置源（参见[第4.9.5](#_bookmark214)）。这是单独控制的

ADC输入为容性输入，采样时，输入两端的电容约为1pF（ADC外部会有额外的电容，例如封装和PCB布线）。即使在500 ksps采样时，有效阻抗也超过100 k Ω，对于直流测量，不需要缓冲。

* + - 1. 一次性样品

向[CS](#_bookmark216).START\_ONCE写入1将立即开始新的转换。[CS.READY](#_bookmark216)将变为低电平，表示当前正在进行转换。经过96个周期的复位后，[CS](#_bookmark216).READY将变为高电平。12位转换结果在[RESULT](#_bookmark217)中提供。

在转换开始前的任何时间，通过写入[CS](#_bookmark216).AINSEL选择要采样的ADC输入。AINSEL值为0.3时，选择GPIO 26.29上的ADC输入AINSEL为4选择内部温度传感器。

**注意**

切换AINSEL时不需要建立时间

* + - 1. 自由运行采样

设置[CS](#_bookmark216).START\_MANY时，ADC将定期自动开始新的转换。最新的转换结果始终在[结果](#_bookmark217)中可用，但对于IRQ或DMA驱动的采样流，必须使能ADC FIFO（[第4.9.2.4](#_bookmark207)）。

默认情况下（[DIV](#_bookmark220)= 0），新的转换在前一个转换完成后立即开始，因此每96个周期产生一个新的样本在48MHz的时钟频率下，这产生500ksps。

将[DIV](#_bookmark220).INT设置为某个正值*n*将每*n*+ 1个周期触发ADC一次，但如果当前正在进行转换，ADC将忽略此值，因此通常n>= 96。例如，将[DIV](#_bookmark220).INT设置为47999将使ADC以1ksps运行（如果以48 MHz时钟运行）

起搏定时器支持分数速率分频（一阶Δ Σ）。当将[DIV](#_bookmark220).FRAC设置为非零值时，

通过改变和之间的采样间隔，ADC将平均每周期启动一次新的转换。

* + - 1. 多输入采样

[CS](#_bookmark216).RROBIN允许ADC以交错方式对多个输入进行采样，同时执行自由运行采样。RROBIN中的每个位对应于[CS](#_bookmark216). AINSEL的五个可能值之一。当ADC完成转换时，[CS](#_bookmark216).AINSEL将自动循环至RROBIN中相应位置位的下一个输入

将全零写入[CS](#_bookmark216). RROBIN可禁用循环采样功能

例如，如果AINSEL初始为0，RROBIN设为0x06（位1和2置1），则ADC将按以下顺序对通道进行

* + - * 1. 频道0
        2. 通道1
        3. 通道2
        4. 通道1
        5. 通道2
        6. 一频道

**注意**

AINSEL的初始值不需要与RROBIN中的设置位对应

* + - 1. 样本FIFO

ADC样本可以直接从[RESULT](#_bookmark217)寄存器读取，也可以存储在本地8条目FIFO中，然后从[FIFO](#_bookmark219)中读取。FIFO操作由[FCS](#_bookmark218)寄存器控制

如果设置了[FCS](#_bookmark218).EN，则将每次ADC转换的结果写入FIFO。当收到ADC的IRQ或DREQ信号通知时，软件中断处理程序或RP 2040 DMA可以从FIFO读取该样本。或者，软件可以轮询[FCS](#_bookmark218)中的状态位，以等待每个样本可用。

如果转换完成时FIFO已满，则设置粘性错误标志FCS.OVER当前FIFO内容不会因此事件而改变，但在FIFO已满时完成的任何转换都将丢失。

有两个标志用于控制ADC写入FIFO的数据

* FCS将FIFO数据右移到8位大小（即FIFO位7:0为转换结果位11:4）。这适用于8位DMA传输到内存中的字节缓冲区，允许更深的捕获缓冲区，代价是一些

精度

* [FCS](#_bookmark218).ERR将设置每个FIFO值的[FIFO](#_bookmark219).ERR标志，表明发生了转换错误，即SAR未能收敛（见下文）

**谨慎**

转换错误会产生未定义的结果，应丢弃相应的样本它们表示一个或多个位的比较未能在允许的时间内完成。通常这是由比较器亚稳态引起的，即输入信号越接近比较器阈值，则作出决定所花费的时间越比较器的高增益降低了不做出决定的概率

* + - 1. DMA

RP 2040 DMA（第2.5节）可以通过对[FIFO](#_bookmark219)寄存器执行正常的存储器映射读取，并由ADC\_DREQ系统数据请求信号进行调整，必须考虑以下方面

* 采样FIFO必须使能（[FCS](#_bookmark218).EN），以便将采样写入其中; FIFO默认禁用，以便在ADC用于单触发转换时不会意外填充
* ADC的数据请求握手（DREQ）必须通过[FCS](#_bookmark218).DREQ\_EN使能
* 用于传输的DMA通道必须选择DREQ\_ADC数据请求信号（第2.5.3.1）。
* DREQ断言的阈值（[FCS](#_bookmark218).THRESH）应设置为1，以便DMA在单个

样本存在于FIFO中请注意，这也是IRQ断言所使用的阈值，因此非DMA用例可能更倾向于使用较高的值来处理频率较低的中断。

* 如果DMA传输大小设置为8位，以便DMA传输到内存中的字节数组，则还必须设置[FCS](#_bookmark218). CLK，以便将FIFO样本预移位到8位有效位。
* 如果要对多个输入通道进行采样，[CS](#_bookmark216).RROBIN包含这些通道的5位掩码（4个外部输入加温度传感器）。此外，[CS](#_bookmark216).AINSEL必须为第一个样品选择通道
* 应在启动ADC之前配置ADC采样速率（[参见4.9.2.2](#_bookmark206)）

ADC配置适当后，应首先启动DMA通道，然后通过[CS](#_bookmark216).START\_MANY启动 ADC转换一旦DMA完成，ADC可以停止，或立即开始新的DMA传输。清除[CS](#_bookmark216).START\_MANY以停止ADC后，软件还应轮询[CS](#_bookmark216).READY以确保最后一次转换已完成，然后从FIFO中排出任何杂散样本

* + - 1. 中断

当FIFO电平达到可配置阈值[FCS](#_bookmark218). THRESH时，可生成中断中断输出必须通过[INTE](#_bookmark222)使能。

可以从[INTS](#_bookmark224)读取状态。通过将FIFO排空至低于[FCS](#_bookmark218). THRESH的电平来清除中断

* + - 1. 供应

ADC电源在其自己的引脚上分离，以实现噪声滤波。

* + 1. ADC有效位

*565号。测试期间使用的参数。*

|  |  |
| --- | --- |
| 参数 | 值 |
| 采样率 | 250ksps |

表征ADC并测量ADC的ENOB。在室温下对硅批次进行测试，测试在3个典型（tt）以及3个快速（ff）和3个慢速（ss）拐角RP 2040器件上进行。[表566](#_bookmark210)中的典型值、最小值和最大值反映了测试中使用的硅

|  |  |
| --- | --- |
| 参数 | 值 |
| FFT窗口 | 5项Blackman-Harris |
| FFT仓 | 4,096 |
| FFT平均 | 没有一 |
| 最小输入电平 | 1 |
| 最大输入电平 | 4,094 |
| 输入频率 | 997赫兹 |

566号测试的各个部件的结果（快速、缓慢和典型）。

应该注意的是，THD通常使用前5或6个谐波计算。然而，由于INL/DNL误差（见[第4.9.4节](#_bookmark211)）产生的误差大于此值，因此使用前30个峰。这使得THD值略差，但更能代表现实。

|  |  |  |  |
| --- | --- | --- | --- |
|  | Min | 典型 | Max |
| THD*1* | -55.6dB | 55dB | -54.4dB |
| SNR | 60.9dB | 61.5dB | 62.0dB |
| SFDR | 59.2dB | 59.9dB | 60.5dB |
| SINAD | 53.6dB | 54.0dB | 54.6dB |
| ENOB | 8.6 | 8.7 | 8.8 |

1由于INL会产生大量谐波，因此使用了最高的30个峰值这与传统的THD计算不同

**重要**

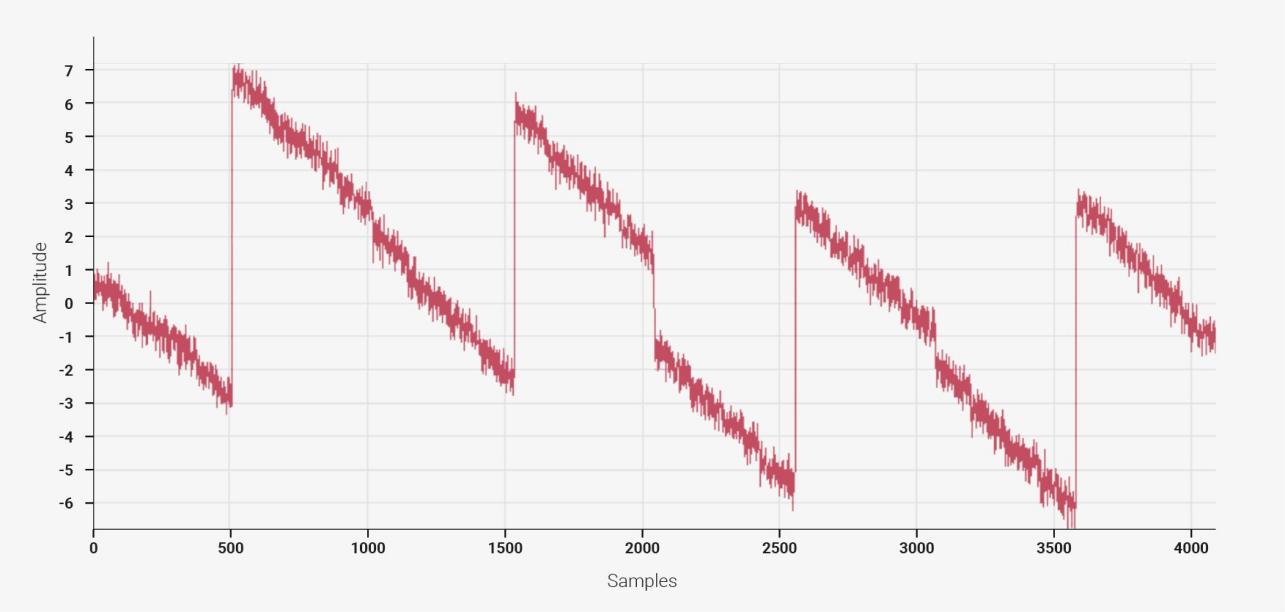
测试使用了一个具有低噪声片上基准电压源的电路板，因为在表征ADC时，没有其他噪声源影响测量非常重要

* + 1. INL和DNL

积分非线性（INL）和微分非线性（DNL）用于测量ADC生成的输入信号的量化误差在理想的ADC中，输入到输出传递函数应该在模拟输入信号和数字化输出信号之间具有线性量化传递。每个二进制结果的RP 2040 ADC INL值[如图11 -6](#_bookmark212)所示，表明误差是一个锯齿形曲线，而不是预期曲线。

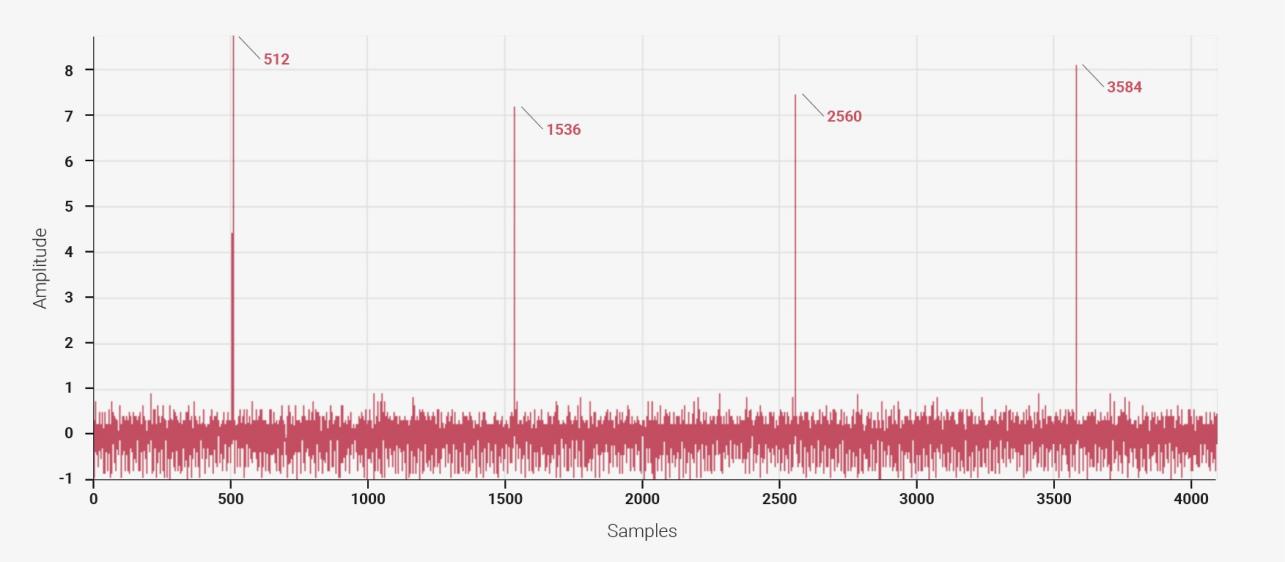
图116. INL（RP 2040）的ATE

图117. DNL（RP 2040）的ATE



名义上，ADC从一个数字值移动到下一个数字值，通俗地表示为“无失码”。但是，如果ADC跳过一个值仓，则会导致差分非线性（DNL）误差出现尖峰由于ADC的设计原因，这些类型的误差通常只发生在特定代码

RP 2040 ADC的DNL基本平坦，低于1 LSB。然而，在四个值（512、1，536、2，560和3，584）处，ADC的DNL误差峰值如[图117](#_bookmark213)所示



INL和DNL误差来自ADC某些内部电容的缩放误差这些电容器的值很小（仅为几十毫微微法拉），并且在这些非常小的值下，这些电容器的芯片模拟可能与实际略有如果这些电容正确匹配，ADC性能可能会更好。

这些INL和DNL误差会在一定程度上限制ADC的性能，具体取决于使用情况（参见勘误表[RP 2040-E11](#_bookmark323)）。

* + 1. 温度传感器

温度传感器测量连接到第五个ADC通道（AINSEL=4）的偏置双极二极管的Vbe电压。通常，在27摄氏度时，Vbe = 0.706 V，斜率为-1.721 mV/度。因此，温度可以近似如下:

T = 27-（ADC\_电压-0.706）/0.001721

由于Vbe和Vbe斜率可能在温度范围内变化，并且因器件而异，因此如果需要精确测量，则可能需要进行

使用前必须通过[CS](#_bookmark216).TS\_EN启用温度传感器的偏置源。这会使ADC\_AVDD的功耗增加约40μA。

**注意**

板载温度传感器对基准电压误差非常敏感。如果ADC返回值891，则对应的温度为20.1°C。然而，如果参考电压比3.3V低1%，则相同的读数891将对应于24.3°C。基准电压变化1%，温度变化就会超过4°C因此，如果您想提高内部温度传感器的精度，值得考虑添加外部基准电压。

**注意**

INL误差（见[第4.9.4](#_bookmark211)）不在ADC的可用温度范围内

* + 1. 登记册一览表

567号ADC寄存器列表

568号CS寄存器

ADC寄存器从基址0x4004c000（在SDK中定义为ADC\_BASE）开始。

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [CS](#_bookmark216) | ADC控制和状态 |
| 0x04 | [结果](#_bookmark217) | 最近ADC转换 |
| 0x08 | [FCS](#_bookmark218) | FIFO控制和状态 |
| 0x0C | [FIFO](#_bookmark219) | 转换结果FIFO |
| 0x10 | [DIV](#_bookmark220) | 时钟分频器。如果非零，则CS\_START\_MANY将启动转换  而不是背靠背。  写入这些字段中的任何一个时，分频器复位总周期为1 + INT + FRAC/256 |
| 0x14 | [INTR](#_bookmark221) | 原始中断 |
| 0x18 | [INTE](#_bookmark222) | 中断使能 |
| 0x1c | [INTF](#_bookmark223) | 中断强制 |
| 0x20 | [INTs](#_bookmark224) | 强制屏蔽后的屏蔽状态 |

### [ADC](#_bookmark215):CS寄存器

**偏移**:0x00

描述

ADC控制和状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一点二十一分 | Reserved. | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 20:16 | RROBIN | 循环取样。每通道1位将所有位设为0，禁用。  否则，ADC将以循环方式循环通过每个使能通道  要采样的第一个通道将是当前由AINSEL指示的通道  AINSEL将在每次转换后使用新选择的通道进行更新 | RW | 0x00 |
| 15 | Reserved. | - | - | - |
| 十四点十二分 | 安塞尔 | 选择模拟复用器输入。在循环模式下自动更新 | RW | 0x0 |
| 11 | Reserved. | - | - | - |
| 10 | 错误\_粘滞 | 以前的ADC转换遇到错误。写1为清除。 | WC | 0x0 |
| 9 | ERR | 最近的ADC转换遇到错误;结果未定义或有噪声。 | RO | 0x0 |
| 8 | 准备 | 如果ADC准备好开始新的转换，则为1表示以前的任何转换都已完成。  0，正在进行转换。 | RO | 0x0 |
| 七比四 | Reserved. | - | - | - |
| 3 | 开始\_许多 | 当此位为1时，连续执行转换。新的转换将在上一次完成后立即开始 | RW | 0x0 |
| 2 | 立即开始 | 开始单个转换。自动清除如果声明start\_many，则忽略 | SC | 0x0 |
| 1 | 简体中文 | 打开温度传感器电源1-启用。0-禁用。 | RW | 0x0 |
| 0 | EN | ADC上电并使能其时钟。1-启用。0-禁用。 | RW | 0x0 |

569号结果

寄存器

570号FCS寄存器

### [ADC](#_bookmark215):结果寄存器

**偏移**:0x04

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:12 | Reserved. | - | - |
| 十一比零 | 最近ADC转换 | RO | 0x000 |

### [ADC](#_bookmark215):FCS寄存器

**偏移量**:0x 08

描述

FIFO控制和状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一点二十八分 | Reserved. | - | - | - |
| 二十七点二十四分 | THRESH | 当电平>=阈值时，DREQ/IRQ置位 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 二十三点二十分 | Reserved. | - | - | - |
| 十九点十六分 | 水平 | 当前在FIFO中等待的转换结果数 | RO | 0x0 |
| 十五点十二分 | Reserved. | - | - | - |
| 11 | 超过 | 如果FIFO已溢出，则为1写1为清除。 | WC | 0x0 |
| 10 | 下 | 如果FIFO已下溢，则为1写1为清除。 | WC | 0x0 |
| 9 | 充分 |  | RO | 0x0 |
| 8 | 空 |  | RO | 0x0 |
| 七比四 | Reserved. | - | - | - |
| 3 | DREQ\_EN | 如果为1:FIFO包含数据 | RW | 0x0 |
| 2 | ERR | 如果1:转换错误位与结果一起出现在FIFO中 | RW | 0x0 |
| 1 | 移位 | 如果为1:FIFO结果右移为一个字节。启用DMA到字节缓冲区。 | RW | 0x0 |
| 0 | EN | 如果为1:每次转换后将结果写入FIFO | RW | 0x0 |

表571。FIFO寄存器

572号 DIV寄存器

### [ADC](#_bookmark215):FIFO寄存器

**偏移**:0x0c

描述

转换结果FIFO

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - | - |
| 15 | ERR | 如果此特定样本发生转换错误，则为1如果样本移动，则保持在同一位置 | RF | - |
| 十四点十二分 | Reserved. | - | - | - |
| 十一比零 | Val |  | RF | - |

### [ADC](#_bookmark215):DIV寄存器

**偏移**:0x10

描述

时钟分频器。如果非零，CS\_START\_MANY将定期启动转换，而不是背靠背。

写入这些字段中的任何一个时，分频器复位总周期为1 + INT + FRAC/256

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | Reserved. | - | - | - |
| 二13:8 | INT | 时钟除数的一部分 | RW | 0x0000 |
| 7:0 | FRAC | 时钟除数的小数部分一阶Δ-Σ。 | RW | 0x00 |

### [ADC](#_bookmark215):INTR寄存器

573号INTR寄存器

表574. INTE寄存器

575号 INTF寄存器

576号INTS寄存器

**偏移**:0x14

描述

原始中断

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | FIFO | 当样本FIFO达到一定水平时触发该电平可通过FCS\_THRESH字段进行编程 | RO | 0x0 |

### [ADC](#_bookmark215):INTE寄存器

**偏移**:0x18

描述

中断使能

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | FIFO | 当样本FIFO达到一定水平时触发该电平可通过FCS\_THRESH字段进行编程 | RW | 0x0 |

### [ADC](#_bookmark215):INTF寄存器

**偏移量**:0x1c

描述

中断强制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | FIFO | 当样本FIFO达到一定水平时触发该电平可通过FCS\_THRESH字段进行编程 | RW | 0x0 |

### [ADC](#_bookmark215):INTS寄存器

**偏移**:0x20

描述

强制屏蔽后的屏蔽状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | FIFO | 当样本FIFO达到一定水平时触发该电平可通过FCS\_THRESH字段进行编程 | RO | 0x0 |

* 1. SSI

###### Synopsys文档

Synopsys专有。经允许使用

RP 2040有一个同步串行接口（SSI）控制器，出现在QSPI引脚上，用于与外部闪存设备通信。SSI形成XIP块的一部分

SSI控制器基于Synopsys DW\_apb\_ssi IP（v4.01a）的配置

* + 1. 概述

为了使DW\_apb\_ssi连接到串行主机或串行从机外围设备，外围设备必须至少具有以下接口之一

摩托罗拉串行外设接口（SPI）

摩托罗拉的四线全双工串行协议。串行时钟相位和极性有四种可能的组合。时钟相位（SCPH）决定串行传输是从从机选择信号的下降沿开始，还是从串行时钟的第一个沿开始当DW\_apb\_ssi空闲或禁用时，从机选择线保持高

德州仪器串行协议（SSP）

四线、全双工串行协议。用于SPI和Microwire协议的从机选择线兼作SSP协议的帧指示器

美国国家半导体公司

一种半双工串行协议，使用从串行主机传输到目标串行从机的控制字

您可以对控制寄存器0（CTRLR0）中的FRF（帧格式）位字段进行编程，以选择使用的协议

DW\_apb\_ssi支持的串行协议允许使用硬件或软件选择或寻址串行从机。当在硬件中实现时，在专用硬件选择线的控制下选择串行从机。从串行主机生成的选择线的数量等于总线上存在串行主机设备在数据传输开始前置位目标串行从机的选择线该架构如[图118](#_bookmark226)所示。

在软件中实现时，所有串行从机的输入选择线应源自串行主机上的单个从机选择输出。在此模式下，假设串行主机只有一个从机选择输出。如果系统中有多个串行主机，则可以对所有主机的从机选择输出进行逻辑与运算，为所有串行从机器件生成单个从机选择输入。软件域中的主程序控制目标从器件的选择;该架构如[图118](#_bookmark226)所示。软件将使用所有从机中的SSIENR寄存器，以控制哪个从机响应来自主设备的串行传输请求

DW\_apb\_ssi不强制对串行从设备选择进行硬件或软件控制您可以为任一实现配置DW\_apb\_ssi，如[图118](#_bookmark226)所示。

*图118. 硬件/软件从机选择。*

硕士

数据总线

奴隶

ss\_0

SS

ss\_x

奴隶

SS

一

硕士

数据总线

奴隶

SS

SS

奴隶

SS

B

ss =从机选择线

* + 1. 特征

DW\_apb\_ssi是一个可配置和可编程的组件，是一个全双工主串行接口。主机处理器通过APB接口访问DW\_apb\_ssi上的数据、控制和状态信息。DW\_apb\_ssi还与DMA控制器接口，用于批量数据传输。

DW\_apb\_ssi配置为串行主机。DW\_apb\_ssi可以使用以下接口之一

* + - * 摩托罗拉串行外设接口（SPI）
      * 德州仪器串行协议（SSP）
      * 美国国家半导体公司

在RP 2040上，DW\_apb\_ssi是闪存就地执行子系统的组件（参见第2.6.3节），并提供与外部SPI、双SPI或四SPI闪存设备的通信。

* + - 1. IO连接

SSI控制器连接到以下引脚:

* + - * + QSPI\_SCLK连接到输出时钟*SCLK\_out*
        + QSPI\_SS\_N连接到片选*ss\_o\_n*
        + QSPI\_SD[3:0]连接到数据总线*txd*和*rxd*

IP上的一些引脚由于未使用而被绑住

* + - * + *ss\_in\_n*处于高位

时钟连接如下:

* + - * + *python*和*sclk*是从python\_sys驱动的
    1. IP修改

对Synopsys DW\_apb\_ssi硬件进行了以下修改

1. XIP访问是字节交换的，因此地址最小的字节位于最低有效位置
2. 当SPI\_CTRLR0\_RMB\_L为0时，XIP指令字段被附加到用于XIP访问的地址的末尾，而不是被前置到开头
3. DMARDLR的复位值从0增加到4。RP 2040上的SSI到DMA握手仅请求单次传输或四次突发传输，具体取决于RX FIFO电平是否达到DMARDLR，因此DMARDLR不应从该值更改。

其中第一项更改允许小端总线主控器（例如RP 2040 DMA或RP 2040上使用的Cortex-M0+配置）进行混合大小访问。请注意，这仅适用于XIP访问（范围为0x10000000到0x13ffffff的RP 2040系统地址），而不适用于直接访问DW\_apb\_ssi FIFO。当直接访问SSI时，软件可能需要手动交换字节，或者使用RP 2040 DMA的字节交换功能。

第二个支持XIP地址后的连续位，因此可以支持无命令前缀的XIP模式（例如在Winbond设备上的EBhQuad I/O Fast Read），以获得更高的性能。例如，以下配置将用于在每次访问XIP地址窗口时发出标准03h串行读取命令:

* + - * SPI\_CTRLR0\_RIG\_L = 8位
      * SPI\_CTRLR0\_ADDR\_L = 24位
      * SPI\_CTRLR0\_XIP\_CMD =0x03

这将首先发出8个命令位（0x03），然后发出24个地址位，然后输入数据位。闪存进入XIP状态后，用于EBh四通道读取的配置为:

* + - * SPI\_CTRLR0\_R0\_L = 0
      * SPI\_CTRLR0\_ADDR\_L = 32位
      * SPI\_CTRLR0\_XIP\_CMD =0xa0（W25Qx设备上的延续代码

对于每个XIP访问，DW\_apb\_ssi将发出32个“地址”位，由RP 2040系统总线地址的24个LSB组成，后跟8位延续码0xa0。未发出命令前缀

4.10.3.1. 使用软件的目标从机选择示例

以下示例是说明如何使用软件来选择目标从机的伪代码

1. public void run（）{
2. disable\_all\_serial\_devices（）;①
3. initialize\_mst（ssi\_mst\_1）;
4. initialize\_slv（ssi\_slv\_1）;
5. start\_serial\_xfer（ssi\_mst\_1）;

（六）

①此函数将每个SSIENR寄存器中的SSI\_EN位设置为逻辑“0”

串行总线上的设备

此功能用于串行传输的主设备;

1. 写入CTRLR0以匹配所需的
2. 如果接收到传输，则仅将帧数写入CTRLR1
3. 写入BAUDR以设置传输波特率。
4. 写入TXFTLR和RXFTLR以设置FIFO阈值水平
5. 写入IMR寄存器以设置中断掩码
6. 将SER寄存器位[0]写入逻辑“1”
7. 将SSIENR寄存器位[0]写入逻辑“1”以使能主机。

该函数初始化目标从设备（本示例中的从设备1）

串行传送

1. 写入CTRLR0以匹配所需的
2. 写入TXFTLR和RXFTLR以设置FIFO阈值水平
3. 写入IMR寄存器以设置中断掩码
4. 将SSIENR寄存器位[0]写入逻辑“1”以使能从机。
5. 如果从机要传输数据，则将数据写入TX FIFO。现在，从机已使能，并等待其ss\_in\_n输入端口上的请注意，所有其他串行从机均已禁用（SSI\_EN=0），因此不会响应其ss\_in\_n端口上的活动电平。

该函数通过将传输数据写入

主机的TX FIFO。用户可以使用函数轮询忙状态或使用ISR来确定串行传输何时完成。

* + 1. 时钟比

比特率时钟（sclk\_out）的最大频率是ssi\_out频率的一半这允许移位控制逻辑在sclk\_out的一个时钟沿上捕获数据并且在相对沿上传播数据

[图119](#_bookmark227)说明了sclk\_out和ssi\_out之间的最大比率

*图119.最大sclk\_out/ssi\_out比率。*



捕获 驱动器1 捕获1 驱动器2 捕获2 驱动3捕获3

ssi\_clk

sclk\_out发送/接收

MSB

sclk\_out行仅在正在进行活动传输时切换在所有其他时间，它都保持在非活动状态，如其运行的串行协议所定义的

sclk\_out的频率可从以下等式导出



SCKDV是可编程寄存器BAUDR中的位字段，保存0至65，534范围内的任何偶数值如果SCKDV为0，则sclk\_out被禁用。

4.10.4.1.频率比汇总

比特率时钟（sclk\_out）和DW\_apb\_ssi外设时钟（ssi\_out）之间的频率比限制总结如下:

·

* + 1. 发送和接收FIFO缓冲器

DW\_apb\_ssi使用的FIFO缓冲器是16个条目深的内部D型触发器。发送和接收FIFO缓冲器的宽度都固定在32位，这是由于串行规范规定串行传输（数据帧）的长度可以是4到16/32位小于32位的数据帧在写入发送FIFO缓冲区时必须右对齐移位控制逻辑自动右对齐接收FIFO缓冲器中的接收数据

FIFO缓冲区中的每个数据条目包含单个数据帧。不可能在单个FIFO位置中存储多个数据帧;例如，您可能无法在单个FIFO位置中存储两个8位数据帧。如果需要8位数据帧，则串行移位器传输数据时将忽略或不使用FIFO条目的高位

**注意**

当DW\_apb\_ssi禁用（SSI\_EN = 0）或复位（presetn）时，发送和接收FIFO缓冲器被清除

发送FIFO通过APB写命令加载到DW\_apb\_ssi数据寄存器（DR）。数据通过移位控制逻辑从发送FIFO中弹出当FIFO中的条目数小于或等于FIFO阈值时，发送FIFO生成FIFO空中断请求（ssi\_txe\_intr）通过可编程寄存器TXFTLR设置的阈值决定产生中断的FIFO条目的级别阈值允许您向处理器提供发送FIFO几乎为空的早期指示如果尝试将数据写入已满的发送FIFO，则会生成发送FIFO溢出中断（ssi\_txo\_intr）

通过APB读取命令将数据从接收FIFO弹出到DW\_apb\_ssi数据寄存器（DR）。接收FIFO由移位控制逻辑从接收移位寄存器加载。当FIFO中的条目数大于或等于FIFO阈值加上

577号。发送FIFO阈值（TFT）解码值

578号。接收FIFO阈值（TFT）解码值

一个. 通过可编程寄存器RXFTLR设置的阈值决定产生中断的FIFO条目的级别

阈值允许您向处理器提供接收FIFO即将满的早期指示当接收移位逻辑试图将数据加载到完全满的接收FIFO中时，会生成接收FIFO溢出中断（ssi\_rxo\_intr）然而，这些新接收到的数据会丢失。如果尝试从空的接收FIFO中读取，则会生成接收FIFO下溢中断（ssi\_rxu\_intr）这会提醒处理器读取的数据无效。

[表577](#_bookmark228)提供了不同发送FIFO阈值的说明

|  |  |
| --- | --- |
| TFT值 | 描述 |
| 0000\_0000 | ssi\_txe\_intr在发送FIFO中存在零数据条目时置位 |
| 0000\_0001 | ssi\_txe\_intr在发送FIFO中存在一个或更少数据条目时置位 |
| 0000\_0010 | ssi\_txe\_intr在发送FIFO中存在两个或更少数据条目时置位 |
| … | … |
| 0000\_1101 | 当发送FIFO中存在13个或更少数据条目时，ssi\_txe\_intr置位 |
| 0000\_1110 | 当发送FIFO中存在14个或更少数据条目时，ssi\_txe\_intr置位 |
| 0000\_1111 | 当发送FIFO中存在15个或更少数据条目时，ssi\_txe\_intr置位 |

[表578](#_bookmark229)提供了不同接收FIFO阈值的说明

|  |  |
| --- | --- |
| RFT值 | 描述 |
| 0000\_0000 | 当接收FIFO中存在一个或多个数据条目时，ssi\_rxf\_intr被置位 |
| 0000\_0001 | 当接收FIFO中存在两个或更多数据条目时，ssi\_rxf\_intr置位 |
| 0000\_0010 | 当接收FIFO中存在三个或更多数据条目时，ssi\_rxf\_intr置位 |
| … | … |
| 0000\_1101 | 当接收FIFO中存在14个或更多数据条目时，ssi\_rxf\_intr置位 |
| 0000\_1110 | 当接收FIFO中存在15个或更多数据条目时，ssi\_rxf\_intr置位 |
| 0000\_1111 | 当接收FIFO中存在16个数据条目时，ssi\_rxf\_intr置位 |

* + 1. 32位帧大小支持

IP被配置为将数据帧大小的最大可编程值设置为32位。因此，存在以下特征

* + - dfs\_32（CTRLR0[20:16]）有效，其中包含数据帧大小的值新寄存器字段保存值0到31。dfs（CTRLR0[3:0]）无效，写入此寄存器无效。
    - 接收和发送FIFO宽度为32位。
    - 数据寄存器的所有32位均有效。
    1. SSI中断

DW\_apb\_ssi支持组合和单独的中断请求，每个中断请求都可以被屏蔽。组合中断请求是屏蔽后所有其他DW\_apb\_ssi中断的OR结果。只有组合的中断请求被路由到中断控制器。所有DW\_apb\_ssi中断都是电平中断，高电平有效。

DW\_apb\_ssi中断描述如下:

发送FIFO空缓存（ssi\_txe\_intr）

当发送FIFO等于或低于其阈值，需要维修以防止欠载运行时，置1通过软件可编程寄存器设置的阈值确定产生中断的当数据写入发送FIFO缓冲器时，硬件会清除此中断，使其超过阈值电平。

发送FIFO溢出缓冲器（ssi\_txo\_intr）

当发送FIFO完全填满后，APB访问尝试写入发送FIFO时，置1设置后，从APB写入的数据将被丢弃。在读取发送FIFO溢出中断清除寄存器（TXOICR）之前，此中断保持置位状态。

接收FIFO全中断（ssi\_rxf\_intr）

当接收FIFO等于或大于其阈值加1，并且需要服务以防止溢出时，置1。通过软件可编程寄存器设置的阈值确定产生中断的接收FIFO条目的电平当从接收FIFO缓冲器读取数据时，硬件会清除此中断，使其低于阈值水平。

接收FIFO溢出缓存（ssi\_rxo\_intr）

当接收逻辑在接收FIFO完全填满后尝试将数据放入接收FIFO时，置1设置后，丢弃新接收的数据该中断保持设置状态，直到您读取接收FIFO溢出中断清除寄存器（RXOICR）。

接收FIFO下溢缓冲器（ssi\_rxu\_intr）

当接收FIFO为空时，APB访问尝试从接收FIFO读取时置1置1时，从接收FIFO回读0在读取接收FIFO下溢中断清除寄存器（RXUICR）之前，此中断保持设置状态。

多主争用路由器（ssi\_mst\_intr）

仅当DW\_apb\_ssi组件配置为串行主设备时才存在。当串行总线上的另一个串行主机选择DW\_apb\_ssi主机作为串行从机设备并正在主动传输数据时，将设置中断。这会通知处理器串行总线上可能存在争用。在读取多主机中断清除寄存器（MSTICR）之前，此中断保持设置状态

合并的HTTP请求（ssi\_intr）

屏蔽后上述所有中断请求的或运算结果要屏蔽此中断信号，必须屏蔽所有其他DW\_apb\_ssi中断请求。

* + 1. 传输模式

在串行总线上传输数据时，DW\_apb\_ssi以本节中讨论的模式工作通过写入控制寄存器0（CTRLR0）设置传输模式（TMOD）

**注意**

传输模式设置不影响串行传输的双工对于由MWCR寄存器控制的Microwire传输，TMOD被忽略

* + - 1. 发送和接收

当TMOD =00b时，发送和接收逻辑均有效。数据传输根据所选帧格式（串行协议）正常进行。发送数据从发送FIFO弹出，并通过txd线路发送到目标设备，目标设备在rxd线路上回复数据来自目标器件的接收数据在每个数据帧结束时从接收移位寄存器移入接收FIFO

* + - 1. 仅发送

当TMOD =01b时，接收数据无效，不应存储在接收FIFO中。根据所选的帧格式（串行协议），数据传输正常进行发送数据从发送FIFO弹出，并通过txd线路发送到目标设备，目标设备在rxd线路上回复数据在数据帧结束时，接收移位寄存器不会将其新接收的数据加载到接收FIFO中。接收移位寄存器中的数据被下一次传输覆盖。进入此模式时，应屏蔽来自接收逻辑的中断。

* + - 1. 仅接收

当TMOD =10b时，传输数据无效。当配置为从机时，发送FIFO在仅接收模式下永远不会弹出。在传输过程中，txd输出保持恒定逻辑电平数据传输根据所选帧格式（串行协议）正常进行。来自目标器件的接收数据在每个数据帧结束时从接收移位寄存器移入接收FIFO。进入此模式时，应屏蔽来自传输逻辑

* + - 1. EEPROM读取

**注意**

此传输模式仅对主机配置有效

当TMOD =11b时，传输数据用于将操作码和/或地址传输到EEPROM器件。通常，这需要三个数据帧（8位操作码，后面是8位高位地址和8位低位地址）。在操作码和地址的传输过程中，接收逻辑不会捕获任何数据（只要DW\_apb\_ssi主机在其txd线上传输数据，rxd线上的数据就会被忽略）。DW\_apb\_ssi主机继续发送数据，直到发送FIFO为空。因此，发送FIFO中应该只有足够的数据帧，以便向EEPROM提供操作码和地址。如果发送FIFO中的数据帧多于所需数据帧，则读取数据会丢失。

当发送FIFO变为空（所有控制信息已发送）时，接收线（rxd）上的数据有效并存储在接收FIFO中; txd输出保持在恒定逻辑电平。串行传输继续，直到DW\_apb\_ssi主机接收的数据帧数与CTRLR1寄存器中的NDF字段值+ 1匹配。

**注意**

将DW\_apb\_ssi配置为SSP模式时，不支持EEPROM读取模式

* + 1. 操作模式

DW\_apb\_ssi可以配置为本节中讨论的基本操作模式

4.10.9.1. 串行主机模式

此模式允许与串行从外设进行串行通信。当配置为串行主机设备时，DW\_apb\_ssi启动并控制所有串行传输。[图120](#_bookmark230)显示了将DW\_apb\_ssi配置为串行主机，而将串行总线上的所有其他设备配置为串行从机的示例。

*图120. DW\_apb\_ssi配置为主设备*



DW\_apb\_ssi

母盘1

奴隶

外设1

奴隶

外围的，外围的

应在单个主设备

系统;可能不需要粘合逻辑

胶连逻辑

txdssi\_oe\_n

rxd sclk\_outss\_n[0] ss\_n[1]

ss\_in\_n

|  |  |
| --- | --- |
|  | DI DO SCLK  SS |
|  |
|  |
|  |
|  |

|  |  |
| --- | --- |
|  | DI DO SCLK  SS |
|  |
|  |
|  |
|  |

串行比特率时钟由DW\_apb\_ssi生成和控制，通过sclk\_out线路驱动输出。当DW\_apb\_ssi被禁用（SSI\_EN = 0）时，不能发生串行传输，并且sclk\_out被保持在“非活动”状态，如其操作所依据的串行协议所定义的

不支持多主机配置

* + - * 1. RXD采样延迟

当DW\_apb\_ssi被配置为主机时，设计中可以包括额外的逻辑，以便延迟rxd信号的默认采样时间这种附加逻辑可以帮助增加串行总线上的最大可实现频率

来自主设备的sclk\_out信号和来自从设备的rxd信号的往返路由延迟可能意味着rxd信号的定时（从主设备看）已经偏离了正常采样时间。[图121](#_bookmark231)说明了这种情况。

*图121.往返路由延迟对sclk\_out信号的影响*

dly=0

dly=5

dly=6

dly=7

波特率=4

ssi\_sl

sclk\_out txd\_mst

MSB

LSB

rxd\_mst

MSB

LSB

sclk\_in

rxd\_slv

MSB

LSB

txd\_slv

MSB

LSB

从机使用来自主机的sclk\_out信号作为选通，以便将rxd信号数据驱动到串行总线上。从设备对sclk\_out信号的路由和采样延迟可能意味着在主设备对rxd信号进行采样之前，rxd位尚未稳定到正确的值[图121](#_bookmark231)显示了一个示例，说明在主机对端口进行采样的默认时间，rxd信号上的路由延迟如何导致rxd值不正确

如果没有RXD采样延迟逻辑，用户将不得不增加传输的波特率，以确保rxd信号的建立时间在范围内;这会导致串行接口的频率降低

当包含RXD采样延迟逻辑时，用户可以动态编程延迟值，以便将rxd信号的采样时间从默认值移动等于ssi\_bits周期数

采样延迟逻辑的分辨率为一个ssi\_cycle。软件可以通过编码一个循环来“训练”串行总线，该循环持续从从机读取数据并递增主机的RXD采样延迟值，直到主机接收到正确的数据

* + - * 1. 数据传输

数据传输由串行主设备启动。当DW\_apb\_ssi使能（SSI\_EN=1）时，发送FIFO中至少存在一个有效数据条目，并选择串行从机器件。当主动传输数据时，状态寄存器（SR）中的忙标志（BUSY）置位。在尝试新的串行传输之前，必须等待忙标志被清除

**注意**

当数据写入发送FIFO时，未设置BUSY状态仅当已选择目标从机且传输正在进行时，此位才置1将数据写入发送FIFO后，移位逻辑直到出现sclk\_out信号的正沿才开始串行传输。等待该上升沿的延迟取决于串行传输的波特率在轮询BUSY状态之前，您应该首先轮询TFE状态（等待1）或等待BAUDR \* ssi\_BLOCK时钟周期。

* + - * 1. 主机SPI和SSP串行传输

当传输模式为“发送和接收”或“仅发送”（分别为TMOD =00 b或TMOD =01 b）时，传输在发送FIFO为空时由移位控制逻辑终止。对于连续数据传输，必须确保在所有数据传输完成之前，传输FIFO缓冲区不会变空发送FIFO阈值水平（TXFTLR）可用于提前中断（ssi\_txe\_intr）处理器，指示发送FIFO缓冲区几乎为空。当DMA用于APB访问时，发送数据电平（DMATDLR）可用于提前请求（dma\_tx\_req）DMA控制器，指示发送FIFO几乎为空。然后，FIFO可以重新填充数据以继续串行传输。用户还可以在使能串行从机之前将数据块（至少两个FIFO条目）这可确保在构成连续传输的数据帧数量出现在传输FIFO中之前，不会开始串行传输。

当传输模式为“仅接收”（TMOD =10 b）时，选择串行从机时，通过将一个“虚拟”数据字写入发送FIFO来启动串行传输DW\_apb\_ssi的txd输出在串行传输期间保持恒定逻辑电平。发送FIFO仅在开始时弹出一次，并在串行传输期间保持为串行传输的结束由控制寄存器1（CTRLR1）中的“数据帧数”（NDF）字段控制

例如，如果您希望从串行从外设接收24个数据帧，则应将NDF字段设置为值23;当接收到的帧数等于NDF值+1时，接收逻辑将终止串行传输这种传输模式增加了APB总线的带宽，因为传输期间不需要为发送FIFO提供服务。每次接收FIFO产生FIFO满中断请求时，应读取接收FIFO缓冲区，以防止溢出。

当传输模式为“EEPROM\_read”（TMOD =11 b）时，选择串行从机（EEPROM）时，通过将操作码和/或地址写入发送FIFO来启动串行传输。操作码和地址被发送到EEPROM器件，之后从EEPROM器件接收读数据并存储在接收FIFO中。串行传输的结束由控制寄存器1（CTRLR 1）中的NDF字段控制

**注意**

将DW\_apb\_ssi配置为SSP模式时，不支持EEPROM读取模式

接收FIFO阈值水平（RXFTLR）可用于提前指示接收FIFO即将满。当DMA用于APB访问时，接收数据电平（DMARDLR）可用于提前请求（dma\_req\_req）DMA控制器，指示接收FIFO几乎已满。

从DW\_apb\_ssi串行主机完成SPI或SSP串行传输的典型软件流程概述如下:

如果使能DW\_apb\_ssi，则通过向SSI使能寄存器（SSIENR）写入0来禁用它

为传输设置DW\_apb\_ssi控制寄存器;这些寄存器可以按任意顺序设置

写控制寄存器0（CTRLR0）。对于SPI传输，串行时钟极性和串行时钟相位参数必须设置为与目标从设备相同

如果传输模式为仅接收，则将传输中的帧数减1写入CTRLR 1（控制寄存器1）;例如，如果要接收四个数据帧，如果要接收四个数据帧，

将“3”写入CTRLR1。

写入波特率选择寄存器（BAUDR）以设置传输的波特率

写入发送和接收FIFO阈值电平寄存器（分别为TXFTLR和RXFTLR）以设置FIFO阈值电平。

写入IMR寄存器以设置中断屏蔽。

从机使能寄存器（SER）寄存器可写入此处，以使能目标从机供选择。如果从机在此使能，则传输FIFO中一出现有效数据条目就开始传输如果没有

如果在写入数据寄存器（DR）之前使能从机，则在使能从机之前不会开始传输

向SSIENR寄存器写入1，使能DW\_apb\_ssi

将传输到目标从机的数据写入发送FIFO（写入DR）。如果此时SER寄存器中没有使能从机，则现在使能它以开始传输。

轮询BUSY状态以等待传输完成不能立即轮询BUSY状态

如果发出发送FIFO空中断请求，则写入发送FIFO（写入DR）。如果发出接收FIFO满中断请求，则读取接收FIFO（读取DR）。

当发送FIFO为空时，移位控制逻辑停止传输如果传输模式是只接收（TMOD =10b），则当接收到规定数目的帧时，由移位控制逻辑停止传输传输完成后，BUSY状态重置为0。

如果传输模式不是仅传输（TMOD！=01b），读取接收FIFO直到其为空。

通过向SSIENR写入0来禁用DW\_apb\_ssi

[图122](#_bookmark232)显示了启动DW\_apb\_ssi主机SPI/SSP串行传输的典型软件流程。该图还显示了serial-master组件内部的硬件流

*图122. DW\_apb\_ssi主SPI/SSP传输流*

软件流程

空闲

禁用DW\_apb\_ssi

DW\_apb\_ssi

通过写入CTRLR0配置主机。CTRLR1、BAUDR、TXFTLR、RXFTLR、IMR、SER、SPI\_CTRLR0

(if双通道/四通道SPI）

空闲

将数据从Tx FIFO弹出至移位器

启用DW\_apb\_ssi

传送比特

将数据写入Tx FIFO

您可以在此处填充FIFO:当发送FIFO中出现第一个数据字且从机使能时，

没有

帧中的所有位都已

正在转移

TMOD=01

是的

加载Rx FIFO

是的

你说呢？

中断服务例程

TMOD=00 TMOD=01

TMOD=10

没有

如果发送FIFO正在请求并且所有数据尚未发送，则将数据写入发送FIFO。如果接收FIFO正在请求，则从接收FIFO读取数据。

没有

没有

传输FIFO是否为空？

所有帧已传输

是的

忙吗？

TMOD=01

是的

是的

没有

读取Rx FIFO

端

* + - * 1. 主Microwire串行传输

来自DW\_apb\_ssi串行主机的Microwire串行传输由Microwire控制寄存器（MWCR）控制MWHS位字段启用和禁用Microwire握手接口。MDD位字段控制数据帧的方向（控制帧总是由主机发送，由从机接收）。MWMOD位字段定义传输是顺序的还是非顺序的。

当发送FIFO中至少有一个控制字且从机使能时，所有Microwire传输均由DW\_apb\_ssi串行主机启动。当DW\_apb\_ssi主机发送数据帧（MDD = 1）时，当发送FIFO为空时，移位逻辑终止传输当DW\_apb\_ssi主机接收到数据帧（MDD = 1）时，传输的终止取决于MWMOD位字段的设置。如果传输是非顺序传输（MWMOD = 0），则在从机数据帧移位后，当发送FIFO为空时，传输终止当传输为顺序传输（MWMOD = 1）时，当接收到的数据帧数等于CTRLR1寄存器中的值+1时，移位逻辑终止传输

当DW\_apb\_ssi主设备上的握手接口被使能（MWHS =1）时，目标从设备的状态在传输后被轮询。仅当从机报告就绪状态时，DW\_apb\_ssi主机才完成传输并清除其BUSY状态。如果传输是连续的，则直到从设备返回就绪状态才发送下一个控制/数据帧

从DW\_apb\_ssi串行主机完成Microwire串行传输的典型软件流程概述如下:

如果启用了DW\_apb\_ssi，则通过将0写入SSIENR来禁用它

为传输设置DW\_apb\_ssi控制寄存器这些寄存器可以按任何顺序设置写入CTRLR0以设置传输参数。

如果传输是顺序的，并且DW\_apb\_ssi主机接收数据，则将传输中的帧数减1写入CTRLR 1;例如，如果要接收四个数据帧，则将“3”写入CTRLR 1。

写入BAUDR以设置传输的波特率

写入TXFTLR和RXFTLR以设置FIFO阈值水平。

写入IMR寄存器以设置中断屏蔽。

您可以写入SER寄存器以使能目标从机进行选择。如果从机在此使能，则传输FIFO中一出现有效数据条目就开始传输如果在写入DR寄存器之前未使能从机，则在使能从机之前不会开始传输

向SSIENR寄存器写入1，使能DW\_apb\_ssi

如果DW\_apb\_ssi主机发送数据，则将控制和数据字写入发送FIFO（写入DR）。如果DW\_apb\_ssi主机接收到数据，则将控制字写入发送FIFO。

如果此时SER寄存器中没有使能从机，则立即使能以开始传输。

轮询BUSY状态以等待传输完成无法立即轮询BUSY状态

当发送FIFO为空时，移位控制逻辑停止传输。如果传输模式为顺序传输，且DW\_apb\_ssi主机接收数据，则当接收到指定数量的数据帧时，移位控制逻辑将停止传输传输完成后，BUSY状态重置为0。

如果DW\_apb\_ssi主机接收到数据，则读取接收FIFO，直到其为空。

通过向SSIENR写入0来禁用DW\_apb\_ssi

[图123](#_bookmark233)显示了启动DW\_apb\_ssi主机Microwire串行传输的典型软件流程。该图还显示了serial-master组件内部的硬件流

*图123. DW\_apb\_ssi主微导丝转移流程*

软件流程

DW\_apb\_ssi

空闲

空闲

禁用DW\_apb\_ssi

将控制帧从Tx FIFO弹出至移位器

配置Master

写CTRLR0。

CTRLR1，BAUDR，传输位

TXFTLR、RXFTLR、MWCR、IMR、SER

没有

启用DW\_apb\_ssi

控制帧中的所有位都已传输？

将控制数据写入Tx FIFO

如果主机接收到数据，用户只需将控制帧写入Tx FIFO。当发送FIFO中出现第一个控制字且从机使能时，传输开始。

MWCR[1]=1

MWCR[1]=0

将数据帧从Tx FIFO弹出至移位器

接收比特

正在转移

传送比特

是否接收到数据帧中的所有位？

你说呢？

是的

中断服务例程

没有

传输数据帧中的所有位？

是的

加载Rx FIFO

没有

忙吗？

MWCR[1]=1

如果发送FIFO正在请求并且所有数据尚未发送，则将数据写入发送FIFO。如果接收FIFO正在请求，则从接收FIFO读取数据。

是的

是的

MWCR[0]=0

MWCR[0]=1

没有

读取Rx FIFO

没有

传输FIFO是否为空？

没有

所有帧都传输了吗？

是的

是的

端

* + 1. 合作伙伴连接接口

DW\_apb\_ssi可以使用以下部分中讨论的接口之一连接到任何串行从机外围设备

* + - 1. 摩托罗拉串行外设接口（SPI）

对于SPI，时钟极性（SCPOL）配置参数决定串行时钟的非活动状态是高电平还是低电平。要传输数据，两个SPI外设必须具有相同的串行时钟相位（SCPH）和时钟极性（SCPOL）值。数据帧的长度可以是4到16/32位（取决于SSI\_MAX\_XFER\_SIZE）

当配置参数SCPH = 0时，数据传输在从机选择信号的下降沿开始。主外设和从外设在串行时钟的第一个边沿捕获第一个数据位;因此，在第一个串行时钟边沿之前，txd和rxd线路上必须存在有效数据

[图124](#_bookmark234)显示SCPH = 0时单次SPI数据传输的时序图。图中显示了配置参数SCPOL = 0和SCPOL = 1时的串行时钟。

*图124. SPI串行格式（SCPH = 0）*



sclk\_out/in 0

sclk\_out/in 1

TXD

MSB

LSB

4 - 32位

RXD

MSB

LSB

ss\_0\_n/ss\_in\_n

ssi\_oe\_n

本节的时序图说明了以下信号

sclk\_out

来自DW\_apb\_ssi主机的

ss\_0\_n

来自DW\_apb\_ssi主机的

ss\_in\_n

DW\_apb\_ssi从机的从机选择输入

ss\_oe\_n

DW\_apb\_ssi主机的输出使能

TXD

RXD

DW\_apb\_ssi主机的传输数据线

DW\_apb\_ssi主机的接收数据线

当SCPH = 0时，支持连续数据传输

* 当CTRLR0.SSTE设置为1，DW\_apb\_ssi在帧之间切换从机选择信号，

当从机选择信号有效时，时钟保持默认值;这种工作模式如[图125](#_bookmark235)所示。

*图125.串行格式连续传输（SCPH = 0）*



sclk\_out/in 0

sclk\_out/in 1

txd/rxd

LSB

MSB

LSB

MSB

ss\_0\_n/ss\_in\_n

ssi\_oe\_n

当配置参数SCPH = 1时，主外设在第一个串行时钟沿开始传输数据

在激活从选择线第一个数据位在第二个（后）串行时钟沿捕获主外设在串行时钟的前沿传播数据在连续数据帧传输期间，从选择线可以保持低电平有效，直到捕获了最后一帧的最后一位

[图126](#_bookmark236)显示配置参数SCPH = 1时SPI格式的时序图

*图126. SPI串行格式（SCPH = 1）*



sclk\_out/in 0

sclk\_out/in 1

TXD

MSB

LSB

4 - 32位

RXD

MSB

LSB

ss\_0\_n/ss\_in\_n

ssi\_oe\_n

连续数据帧的传输方式与单帧相同，下一帧的MSB紧跟在当前帧的LSB从机选择信号在传输期间保持有效

[图127](#_bookmark237)显示配置参数SCPH = 1时连续SPI传输的时序图

*图127. SPI串行格式连续传输（SCPH = 1）*



sclk\_out/in 0

sclk\_out/in 1

TXD

MSB

LSB MSB

LSB

RXD

MSB

LSB MSB

LSB

ss\_0\_n/ss\_in\_n

ssi\_oe\_n

DW\_apb\_ssi上有四种可能的传输模式用于执行SPI串行事务。对于发送和接收传输（控制寄存器0的传输模式字段（9:8）=00b），从DW\_apb\_ssi发送到外部串行器件的数据写入发送FIFO。从外部串行设备接收到DW\_apb\_ssi的数据被推入接收FIFO。

[图128](#_bookmark238)显示串行传输开始前的FIFO电平和传输完成时的FIFO电平。在本例中，两个数据字从DW\_apb\_ssi连续传输到外部串行器件。外部串行设备还使用DW\_apb\_ssi的两个数据字进行响应

*图128. 发送接收SPI和SSP传输的FIFO*

RXD

位置n

写入DR

Tx FIFO缓冲器

NULL

位置n

NULL

位置2

NULL

移位逻辑

位置1发送数据（1）

位置0发送数据（0）

位置2

位置1

地点0

传输前的FIFO状态

TXD

NULL

Rx\_Data（1）Rx\_Data（0）

Rx FIFO缓冲器

读取DR

传输完成时的FIFO状态

Rx FIFO为空

Tx FIFO为空

对于仅发送传输（控制寄存器0的传输模式字段（9:8）=01b），从DW\_apb\_ssi发送到外部串行器件的数据写入发送FIFO。由于从外部串行设备接收的数据被视为无效，因此不会存储在DW\_apb\_ssi接收FIFO中。

[图129](#_bookmark239)显示串行传输开始前的FIFO电平和传输完成时的FIFO电平。在本例中，两个数据字从DW\_apb\_ssi连续传输到外部串行器件。

*图129.仅发送SPI和SSP传输的FIFO状态*

RXD

位置n

写入DR

Tx FIFO缓冲器

NULL

位置n

NULL

位置2

NULL

移位逻辑

位置1发送数据（1）

位置0发送数据（0）

位置2

位置1

地点0

NULL

NULL NULL

Rx FIFO缓冲器

传输前的FIFO状态

TXD

读取DR

传输完成时的FIFO状态

Rx FIFO为空

Tx FIFO为空

对于仅接收传输（控制寄存器0的传输模式字段（9:8）=10b），从DW\_apb\_ssi传输到外部串行器件的数据无效，因此将单个虚拟字写入发送FIFO以开始串行传输。DW\_apb\_ssi的txd输出在串行传输期间保持恒定逻辑电平。从外部串行设备接收到DW\_apb\_ssi的数据被推入接收FIFO。

[图130](#_bookmark240)显示串行传输开始前的FIFO电平和传输完成时的FIFO电平。在本例中，DW\_apb\_ssi从外部串行设备接收两个连续传输的数据字

*图130.仅接收SPI和SSP传输的FIFO状态*

RXD

写入DR

Tx FIFO缓冲器

位置n

NULL

位置n

NULL

位置2

NULL

NULL

移位逻辑

位置1

位置2

位置0 虚拟词 位置1

地点0

传输前的FIFO状态

TXD

NULL

Rx\_Data（1）Rx\_Data（0）

Rx FIFO缓冲器

读取DR

传输完成时的FIFO状态

Rx FIFO为空

Tx FIFO为空

对于EEPROM\_read传输（控制寄存器0的传输模式字段[9:8]=11 b），操作码和/或EEPROM地址写入发送FIFO。在传输这些控制帧期间，DW\_apb\_ssi主机不会捕获接收到的数据。控制帧发送后，来自EEPROM的接收数据存储在接收FIFO中。

[图131](#_bookmark241)显示串行传输开始前的FIFO电平和传输完成时的FIFO电平。在本例中，一个操作码和一个上下地址被发送到EEPROM，八个数据帧从EEPROM读取并存储在DW\_apb\_ssi主机的接收FIFO

*图131. EEPROM读取传输模式的FIFO状态*

RXD

位置n

写入DR

Tx FIFO缓冲器

NULL

位置n

NULL

位置3

NULL

移位逻辑

位置2地址[7:0]

[15:00:00]第一次

位置0操作码

位置7

位置6

Rx\_Data（7）

Rx\_Data（6）

位置1

地点0

传输前的FIFO状态

TXD

Rx\_Data（1）Rx\_Data（0）

Rx FIFO缓冲器

读取DR

传输完成时的FIFO状态

Rx FIFO为空

Tx FIFO为空

* + - 1. 德州仪器同步串行协议（SSP）

数据传输开始于在一个串行时钟周期内置位帧指示线（ss\_0\_n/ss\_in\_n）。要传输的数据在一个串行时钟周期后被驱动到txd线上;类似地，来自从机的数据被驱动到rxd线上。数据在串行时钟的上升沿（sclk\_out/sclk\_in）传播，并在下降沿捕获数据帧的长度范围从4位到32位。

[图132](#_bookmark242)显示了单个SSP串行传输的时序图

*图132. SSP串行格式*



sclk\_out/in

txd/rxd

MSB

LSB

ss\_0\_n/ss\_in\_n

ssi\_oe\_n

连续数据帧的传输方式与单个数据帧相同帧指示符在与当前传输的LSB相同的周期内被置位一个时钟周期，指示另一个数据帧随后到来。

[图133](#_bookmark243)显示了连续SSP传输的时序

*图133. SSP串行格式连续传输*



sclk\_out/in

txd/rxd

MSB

LSB MSB

ss\_0\_n/ss\_in\_n

ssi\_oe\_n

* + - 1. 美国国家半导体公司

数据传输从从选择信号（ss\_0\_n）的下降沿半个串行时钟（sclk\_out）周期后，控制的第一位在txd线上发出控制字的长度可以在1至16位的范围内，并通过在CTRLR0中写入位字段CFS（位15:12）来设置控制字的其余部分由DW\_apb\_ssi串行主机传输在此传输期间，串行主机的rxd线路上

数据字的方向由Microwire控制寄存器（MWCR）中的MDD位字段（位1）控制MDD=0时，表示DW\_apb\_ssi串行主机从外部串行从机接收数据。在传输控制字的LSB后的一个时钟周期，从机外设以伪0位响应，随后是数据帧，长度可以为4至32位。数据在串行时钟的下降沿传播，并在上升沿捕获

从机选择信号在传输期间保持低电平有效，在数据传输后半个时钟周期后解除断言[图134](#_bookmark244)显示从外部串行从机读取单个DW\_apb\_ssi串行主机的时序图

*图134. 单个DW\_apb\_ssi主微导丝串行传输（MDD=0）*



sclk\_out

控制字

TXD

MSB

LSB

4 - 32位

RXD

0 MSB

LSB

ss\_0\_n

ssi\_oe\_n

[图135](#_bookmark245)显示了传输前数据帧和控制帧在发送FIFO中的结构，还显示了编程到MWCR寄存器

*图135.单根微丝传输的FIFO状态（接收数据帧）*

MWHS MDD MWMOD

MWCR

RXD

写入DR

Tx FIFO缓冲器

位置n

NULL

位置n

NULL

位置3

位置2

NULL

NULL NULL

移位逻辑

位置3

位置1

位置2

位置0Ctrl Word（0） 位置1

地点0

传输前的FIFO状态

TXD

NULL NULL NULL

Rx\_Data（0）

Rx FIFO缓冲器

读取DR

传输完成时的FIFO状态

Rx FIFO为空

Tx FIFO为空

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |

Microwire协议的连续传输可以是顺序的，也可以是非顺序的，由MWCR寄存器中的MWMOD位字段（位0）控制

[图13 -6中示出](#_bookmark246)了非顺序连续传输的发生，其中对紧接在当前数据字的LSB之后的下一次传输的控制

*图136.连续非顺序微丝传输（接收数据帧）*



sclk\_out

控制字0

控制字1

TXD

MSB

LSB

MSB

LSB

数据字0数据字1

RXD

0 MSB

LSB

0 MSB

LSB

ss\_0\_n

ssi\_oe\_n

执行连续非顺序传输所需的唯一修改是将更多控制字写入发送FIFO缓冲区，如[图137](#_bookmark247)所示。在本例中，从外部串行从机设备读取两个数据字

*图137.非顺序微丝传输的FIFO状态（接收数据帧）*

MWHS MDD MWMOD

MWCR

RXD

位置n

写入DR

Tx FIFO缓冲器

NULL

位置n

位置3

位置2

NULL

NULL

移位逻辑

位置3

位置1控制字（1）

位置0Ctrl Word（0）

位置2

位置1

地点0

Rx FIFO缓冲器

传输前的FIFO状态

TXD

读取DR

传输完成时的FIFO状态

Rx FIFO为空

Tx FIFO为空

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |

|  |  |
| --- | --- |
| NULL | |
|  |  |
| NULL | |
| NULL | |
| Rx\_Data（1） | |
| Rx\_Data（0） | |
|  |  |

在顺序连续传输期间，DW\_apb\_ssi主机仅传输一个控制字传输以与非顺序读取操作相同的方式开始，但该周期继续读取更多数据。从设备自动将其地址指针递增到下一个位置，并继续从该位置提供数据以这种方式可以读取任意数量的位置;当接收到的字数等于CTRLR 1寄存器中的值加1时，DW\_apb\_ssi主机终止传输

[图138](#_bookmark248)中的时序图和[图139](#_bookmark249)中的示例显示从外部从器件连续顺序读取两个数据帧。

*图138. 连续顺序微丝传输（接收数据帧）*



sclk\_out

控制字

TXD

MSB

LSB

数据字0

数据字1

RXD

0 MSB

LSB MSB

LSB

ss\_0\_n

ssi\_oe\_n

*图139.顺序微丝传输的FIFO状态（接收数据帧）*

MWHS MDD MWMOD

MWCR

RXD

写入DR

Tx FIFO缓冲器

位置n

NULL

位置n

位置3

位置2

NULL NULL NULL

Ctrl Word（0）

移位逻辑

位置3

位置1

位置2

地点0

位置1

地点0

Rx FIFO缓冲器

传输前的FIFO状态

TXD

读取DR

传输完成时的FIFO状态

Rx FIFO为空

Tx FIFO为空

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 1 |

|  |  |
| --- | --- |
| NULL | |
|  |  |
| NULL | |
| NULL | |
| Rx\_Data（1） | |
| Rx\_Data（0） | |
|  |  |

MDD = 1时，表示DW\_apb\_ssi串行主机向外部串行从机传输数据在传输控制字的LSB后， DW\_apb\_ssi主机立即开始向从机外设传输数据帧

[图140](#_bookmark250)显示了单个DW\_apb\_ssi串行主机写入外部串行从机的时序图

*图140.单线传输（传输数据帧）*



sclk\_out

控制字

数据字0

TXD

MSB

LSB MSB

LSB

rxdss\_0\_n

ssi\_oe\_n

**注意**

DW\_apb\_ssi不支持连续顺序Microwire写入，其中MDD = 1且MWMOD = 1。

[图141](#_bookmark251)显示了传输前数据和控制帧在发送FIFO中的结构，还显示了编程到MWCR寄存器中的值。

MWHS MDD MWMOD

MWCR

RXD

位置n

写入DR

Tx FIFO缓冲器

NULL

位置n

位置3

NULL NULL

Tx数据（0）

Ctrl Word（0）

移位逻辑

位置2

位置3

位置1

位置2

地点0

位置1

地点0

Rx FIFO缓冲器

传输前的FIFO状态

TXD

传输完成时的FIFO状态

Rx FIFO为空

Tx FIFO为空

图141.单根微丝传输的FIFO状态（传输数据帧）

|  |  |  |
| --- | --- | --- |
| 0 | 1 | 0 |

|  |  |
| --- | --- |
| NULL | |
|  |  |
| NULL | |
| NULL | |
| NULL | |
| NULL | |

[图14-2](#_bookmark252)中显示了连续传输的发生，其中对紧接在当前数据字的LSB之后的下一个传输的控制字

*图142. 连续微丝传输（传输数据帧）*



sclk\_out

控制字0数据字0控制字1数据字1

TXD

MSB

LSB MSB

LSB MSB

LSB MSB

LSB

RXD

ss\_0\_nssi\_oe\_n

执行连续传输所需的唯一修改是将更多控制和数据字写入发送FIFO缓冲区，如[图143](#_bookmark253)所示。此示例显示将两个数据字写入外部串行从设备。

*图143. 连续微丝传输的FIFO状态（传输数据帧）*

MWHS MDD MWMOD

MWCR

RXD

写入DR

Tx FIFO缓冲器

位置n

位置n

位置3

移位逻辑

位置2 位置3

位置1 位置2

位置0 位置1

地点0

Rx FIFO缓冲器

传输前的FIFO状态

TXD

传输完成时的FIFO状态

NULL

NULL

Ctrl Word（0）

NULL

Tx数据（0）

NULL

Ctrl Word（1）

数据字（1）

NULL

NULL

Rx FIFO为空

Tx FIFO为空

|  |  |  |
| --- | --- | --- |
| 0 | 1 | 0 |

还可以启用Microwire握手接口，以便对外部串行从设备进行DW\_apb\_ssi主写操作。要使能握手接口，必须将1写入MWCR寄存器的MHS位字段（位2）当MHS设置为1时，DW\_apb\_ssi串行主机在完成传输或传输下一个控制字以进行连续传输之前，会检查从机是否处于就绪状态

[图144](#_bookmark254)显示了启用握手接口的连续Microwire传输示例

*图144. 带握手的连续微丝*传输（传输数据帧）



sclk\_out

控制字0

数据字0

控制字1

数据字1

TXD

MSB

LSB MSB

LSB

MSB

LSB MSB

LSB

起始位

RXD

忙 准备

忙 准备

ss\_0\_n

ssi\_oe\_n

第一个数据字发送至串行从机后，DW\_apb\_ssi主机轮询rxd输入，等待从机的就绪状态。收到就绪状态后，DW\_apb\_ssi主机开始传输下一个控制字。完成最后一个数据帧的传输后，DW\_apb\_ssi主机在完成传输前发送一个起始位，以清除从机的就绪状态此传输的FIFO状态[与图14 -3中的相同](#_bookmark253)，但MWHS比特字段为（1）。

要从DW\_apb\_ssi主机向串行从机器件发送控制字（后面不跟数据），发送FIFO缓冲区中必须只有一个条目在连续传输中不可能传输两个控制字，因为DW\_apb\_ssi中的移位逻辑将第二个控制字视为数据字。当DW\_apb\_ssi主机仅传输控制字时，MDD位字段（MWCR寄存器的位1）必须置

在[图14 -5](#_bookmark255)[所示的示例和图14 -6](#_bookmark256)所示的时序图中，启用了握手接口。如果握手接口被禁用（MHS=0），则在从设备捕获控制字的LSB

*图145. 微丝控制字传输的FIFO状态*

MWHS MDD MWMOD

MWCR

RXD

写入DR

Tx FIFO缓冲器

位置n

位置n

位置3

移位逻辑

位置2 位置3

位置1 位置2

位置0 位置1

地点0

Rx FIFO缓冲器

传输前的FIFO状态

TXD

传输完成时的FIFO状态

NULL

NULL

Ctrl Word（0）

NULL

NULL

NULL

NULL

NULL

NULL

NULL

Rx FIFO为空

Tx FIFO为空

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 0 |

*图146.微丝控制字*



sclk\_out

控制字0

TXD

MSB

LSB

起始位

RXD

忙 准备

ss\_0\_n

ssi\_oe\_n

* + - 1. 增强型SPI模式

DW\_apb\_ssi支持RP 2040中SPI的双模式和四模式;不支持八进制模式。TXD、RXD和SSI\_OE\_N信号是四位宽。

数据在多条线路上移出/移入，从而提高了整体吞吐量。串行时钟的极性和相位的所有四种组合在此模式下均有效，并且与正常SPI模式下的双通道SPI或四通道SPI模式的功能类似，但txd、rxd和ssi\_oe\_n信号的宽度不同。可以使用CTRLR0.TMOD字段选择操作模式（写入/读取）。

* + - * 1. 增强型SPI模式下的写入操作

双通道或四通道SPI写操作可分为三部分:

指令阶段

地址阶段

数据阶段

以下寄存器字段用于写操作:

CTRLR0.SPI\_FRF-指定帧的传输格式

SPI\_CTRLR0（控制寄存器0寄存器）-指定指令、地址和数据的长度

SPI\_CTRLR0.SPI\_L-指定指令的长度（指令的可能值为0、4、8或16位）。

SPI\_CTRLR0.ADDR\_L-指定地址长度（解码值见[表579](#_bookmark266)

CTRLR0.DFS或CTRLR0.DFS\_32-指定数据长度。

一条指令占用一个FIFO位置。一个地址可以占用多个FIFO位置。

指令和地址都必须在数据寄存器（DR）中编程。DW\_apb\_ssi将等待，直到两者都被编程以开始写入操作。

指令、地址和数据可编程为在双/四模式下发送，可从SPI\_CTRLR0.TRANS\_TYPE和CTRLR0.SPI\_FRF字段中选择。

**注意**

* 如果CTRLR0.SPI\_FRF被选择为“标准SPI格式”，则所有内容都以标准SPI模式发送，并且忽略SPI\_CTRLR0.TRANS\_TYPE字段。
* 仅当CTRLR0.FRF程控为00b时，CTRLR0.SPI\_FRF才适用。

[图147](#_bookmark257)显示双通道或四通道SPI模式下的典型写操作如果SSI\_SPI\_MODE设置为3，则N的值为7;如果SSI\_SPI\_MODE设置为2，则N的值为3;如果SSI\_SPI\_MODE设置为1，则N的值为1对于1-写操作，指令和地址仅发送一次，随后在DR中编程数据帧，直到发送FIFO变空。

*图147.典型写入操作双通道/四通道SPI模式*



sclk\_out

txd[N:0]

指令

地址

数据

ssi\_oe\_n[N:0]

ss\_oe\_n

要启动双通道/四通道写入操作，CTRLR0.SPI\_FRF必须分别设置为01/10/11这将设置传输类型，对于每个写入命令，数据将以CTLR0.SPI\_FRF字段中指定的格式传输

情况A:指令和地址均以标准SPI格式传输

为此，SPI\_CTRLR0.TRANS\_TYPE字段必须设置为00b。[图148](#_bookmark258)显示了以标准SPI格式传输指令和地址N的值将为:如果CTRLR0.SPI\_FRF设置为11b，则为7;如果CTRLR0.SPI\_FRF设置为10 b，则为3;如果CTRLR0.SPI\_FRF设置为01 b，则为1。

*图148.以标准SPI格式传输的指令和地址*



sclk\_out

联系我们[0]

指令

地址

数据

txd[N-1:0]

数据

ss\_oe\_n[0]ss\_oe\_n[N-1:0]

ss\_oe\_n

情况B:以标准格式传输指令，以增强型SPI格式传输地址

为此，SPI\_CTRLR0.TRANS\_TYPE字段必须设置为1。[图149](#_bookmark259)显示了以标准格式传输指令、以CTRLR0.SPI\_FRF字段中指定的双SPI格式传输地址时的时序图。N的值将为:如果CTRLR0.SPI\_FRF设置为11b，则为7;如果CTRLR0.SPI\_FRF设置为10 b，则为3;如果CTRLR0.SPI\_FRF设置为01 b，则为1。

*图149.以标准格式传输指令和以增强型SPI格式传输地址*



sclk\_out

联系我们[0]

指令

地址

数据

txd[N-1:0]

地址

数据

ss\_oe\_n[0]ss\_oe\_n[N-1:0]

ss\_oe\_n

情况C:指令和地址均以增强型SPI格式传输

为此，SPI\_CTRLR0.TRANS\_TYPE字段必须设置为10b。[图150](#_bookmark260)显示了以CTRLR0.SPI\_FRF字段中指定的SPI格式传输指令和地址N的值将为:如果CTRLR0.SPI\_FRF设置为11b，则为7;如果CTRLR0.SPI\_FRF设置为10 b，则为3;如果CTRLR0.SPI\_FRF设置为01 b，则为1。

*图150.指令和地址均以增强型SPI格式传输*



sclk\_out

txd[N:0]

指令

地址

数据

ssi\_oe\_n[N:0]

ss\_0\_n

情况D:增强型SPI格式的仅指令传输

为此，SPI\_CTRLR0.TRANS\_TYPE字段必须设置为10b。[图151](#_bookmark261)显示了这种传输的时序图。N的值将为:如果CTRLR0.SPI\_FRF设置为11b，则为7;如果CTRLR0.SPI\_FRF设置为10 b，则为3;如果CTRLR0.SPI\_FRF设置为10 b，则为1。

CTRLR0.SPI\_FRF设置为01b。

*图151.增强型SPI格式的仅指令传输*



sclk\_out

txd[N:0]

指令

ssi\_oe\_n[N:0]

ss\_0\_n

* + - * 1. 增强型SPI模式下的读取操作

双通道或四通道SPI读取操作可分为四个阶段:

指令阶段

地址阶段

等待周期

数据阶段

可使用SPI\_CTRLR0.WAIT\_CYCLES字段对等待周期进行编程。编程到SPI\_CTRLR0.WAIT\_CYCLES中的值直接映射到sclk\_out时间。例如，WAIT\_CYCLES=0表示没有等待，WAIT\_CYCLES=1表示一个等待周期，依此类推。引入等待周期以使目标从设备将其模式从输入切换到输出，并且等待周期可以针对不同的设备而变化

对于READ操作，DW\_apb\_ssi发送一次指令和控制数据，然后等待，直到接收到NDF（CTRLR1寄存器）个数据帧，然后解除从机选择信号。

[图152](#_bookmark262)显示双通道四通道SPI模式下的典型读取操作。如果SSI\_SPI\_MODE设置为四模式，则N值为3;如果SSI\_SPI\_MODE设置为双模式，则N值为1

*图152.增强型SPI模式下的典型读取操作*

sclk\_out

txd[N:0]

指令

地址

等待周期

rxd[N:0]

数据

ss\_oe\_n[N:0]

ss\_oe\_n

要启动双通道/四通道读取操作，CTRLR0.SPI\_FRF必须分别设置为01/10/11这将设置传输类型，现在对于每个读取命令，数据将以CTLR0.SPI\_FRF字段中指定的格式传输

以下是增强型SPI模式下的写入操作可能情况

情况A:指令和地址均以标准SPI格式传输

为此，SPI\_CTRLR0.TRANS\_TYPE字段应设置为00b。[图153](#_bookmark263)显示了以标准SPI格式传输指令和地址时的时序图。图中还显示了寻址后的等待周期，可在SPI\_CTRLR0.WAIT\_CYCLES字段中进行编程如果CTRLR0.SPI\_FRF设置为11 b，则N的值为7;如果CTRLR0.SPI\_FRF设置为10 b，则N的值为3;如果CTRLR0.SPI\_FRF设置为01 b，则N的值为1。

*图153.以标准SPI格式传输的指令和地址*

sclk\_out

联系我们[0]

指令

地址

等待周期

txd[N-1:0]

rxd[N:0]

数据

ssi\_oe\_n[0] ssi\_oe\_n[N-1:0]

ss\_0\_n

案例B:以标准传输指令，以双SPI格式传输地址

为此，SPI\_CTRLR0.TRANS\_TYPE字段应设置为01b。[图154](#_bookmark264)显示了以标准格式传输指令、以双SPI格式传输地址如果CTRLR0.SPI\_FRF设置为11 b，则N的值为7;如果CTRLR0.SPI\_FRF设置为10 b，则N的值为3;如果CTRLR0.SPI\_FRF设置为01 b，则N的值为1。

*图154.以标准格式传输指令和以增强型SPI格式传输地址*



sclk\_out

联系我们[0]

指令

地址

txd[N-1:0]

地址

rxd[N:0]

数据

ssi\_oe\_n[0]ssi\_oe\_n[N-1:0]

ss\_0\_n

情况C:指令和地址均以双SPI格式传输

为此，SPI\_CTRLR0.TRANS\_TYPE字段必须设置为10b。[图155](#_bookmark265)所示为以双通道SPI格式传输指令和地址N的值将为:如果CTRLR0.SPI\_FRF设置为11b，则为7;如果CTRLR0.SPI\_FRF设置为10 b，则为3;如果CTRLR0.SPI\_FRF设置为01 b，则为1。

*图155.以增强型SPI格式传输的指令和地址*



sclk\_out

txd[N:0]

指令

地址

rxd[N:0]

数据

ssi\_oe\_n[N:0]

ss\_0\_n

表579.增强SPI模式下的ADDR\_L解码

###### 情况D:无指令，无地址读取传输

为此，SPI\_CTRLR0.ADDR\_L和SPI\_CTRLR0.ADDR\_L必须设置为0，SPI\_CTRLR0.WAIT\_CYCLES必须设置为非零值。[表579](#_bookmark266)列出了增强型（双通道/四通道）SPI模式的ADDR\_L解码值和相应说明

|  |  |
| --- | --- |
| ADDR\_L解码值 | 描述 |
| 0000 | 0-位地址宽度 |
| 0001 | 4-位地址宽度 |
| 0010 | 8-位地址宽度 |
| 0011 | 12-位地址宽度 |
| 0100 | 16-位地址宽度 |
| 0101 | 20-位地址宽度 |
| 0110 | 24-位地址宽度 |
| 0111 | 28-位地址宽度 |
| 1000 | 32-位地址宽度 |
| 1001 | 36-位地址宽度 |
| 1010 | 40-位地址宽度 |
| 1011 | 44-位地址宽度 |
| 1100 | 48-位地址宽度 |
| 1101 | 52-位地址宽度 |
| 1110 | 56-位地址宽度 |
| 1111 | 60-位地址宽度 |

[图156](#_bookmark267)显示了此类传输的时序图N的值将为:如果CTRLR0.SPI\_FRF设置为11b，则为7;如果CTRLR0.SPI\_FRF设置为10 b，则为3;如果CTRLR0.SPI\_FRF设置为01 b，则为1。要启动此传输，软件必须在数据寄存器（DR）中执行虚拟写入，DW\_apb\_ssi将等待编程的等待周期，然后获取NDF字段中指定的数据量

*图156.无指令且无地址读取传输*



sclk\_out

txd[N:0]

等待周期

rxd[N:0]

数据

ssi\_oe\_n[N:0]

ss\_0\_n

* + - * 1. 用于增强型SPI模式的高级I/O映射

增强型SPI模式（双通道和四通道）的输入/输出映射在DW\_apb\_ssi内部进行硬编码。rxd[1]信号将用于在标准SPI工作模式下对输入数据进行

对于其他协议（如SSP和Microwire），I/O映射保持不变。因此，其他协议很容易与任何支持双通道/四通道SPI操作的器件连接，因为其他协议不需要在设计之外存在MUX逻辑

[图157](#_bookmark268)显示了DW\_apb\_ssi在四通道模式下与另一个支持四通道模式的SPI器件的I/O映射。[如图15-7](#_bookmark268)所示，IO[1]引脚在标准SPI模式下使用数据，并连接到rxd[1]引脚，该引脚将在标准操作模式下对输入进行采样

[3]

rxd[3]

[2]

RXD[2]

txd[1]

Rxd[1]

联系我们[0]

联系我们[0]

DW\_apb\_ssi

SPI从设备

IO缓冲器

IO缓冲器

IO缓冲器

IO缓冲器

图157.四通道SPI模式下的高级I/O映射

|  |  |
| --- | --- |
|  | |
|  | [3] |
|  | |
|  | [2] |
|  | |
|  | 执行干事[1]/执行干事 |
|  | |
|  | IO[0]/DI |
|  | |

* + - 1. SPI操作中的双数据速率（DDR）支持

在标准操作中，SPI模式下的数据传输发生在时钟的正沿或负沿。为了提高吞吐量，可以使用双数据速率传输来读取或写入存储器。

DDR模式支持以下SPI协议模式

* SCPH=0 SCPOL=0（模式0）
* SCPH=1 SCPOL=1（模式3）

DDR命令使数据能够在时钟的两个边沿上传输。以下是不同类型的DDR命令:

* 地址和数据以DDR格式传输（或在数据的情况下接收），而指令以标准格式传输
* 指令、地址和数据都以DDR格式传输或接收

DDR\_EN（SPI\_CTRLR0[16]）位用于确定地址和数据是否必须以DDR模式传输，INST\_DDR\_EN（SPI\_CTRLR0[17]）位用于确定指令是否必须以DDR格式传输。这些位

仅当CTRLR0.SPI\_FRF位设置为双通道或四通道模式时有效

[图158](#_bookmark269)描述了DDR写传输，其中指令继续以标准格式传输。在[图158](#_bookmark269)中，如果CTRLR0.SPI\_FRF设置为11b，则N的值将为7，如果CTRLR0.SPI\_FRF设置为10b，则N的值将为3，如果CTRLR0.SPI\_FRF设置为01b，则N的值将为1。

*图158. DDR传输，SCPH=0且SCPOL=0*

sclk\_out ss\_oe\_n

txd[N:0]

INST

A3 A2 A1 A0 D3 D2 D1 D0

rxd[N:0]

ss\_oe\_n[N:0]

INST =指令阶段A3、A2、A1、A0 =地址寄存器

D3、D2、D1、D0 =数据流

[图159](#_bookmark270)描述了DDR写传输，其中指令、地址和数据都以DDR格式传输

*图159.以DDR格式传输指令、地址和数据的DDR传输*

sclk\_outss\_0\_n

txd[N:0]

INST-1 INST-2 A3 A2 A1 A0 D3 D2 D1 D0

rxd[N:0]

ssi\_oe\_n[N:0]

INST-1、INST-2 =指令集A3、A2、A1、A0 =地址集

D3、D2、D1、D0 =数据流

**注意**

在DDR传输中，地址和指令不能被编程为值0。

* + - * 1. 以DDR模式传输数据

在DDR模式下，数据在两个边缘上传输，因此很难正确采样数据。DW\_apb\_ssi使用一个内部寄存器来确定应在哪个边沿上传输数据这将确保接收器能够在采样时获得稳定的数据内部寄存器（DDR\_DRIVE\_EDGE）确定数据传输的边沿DW\_apb\_ssi发送与波特率时钟相关的数据，波特率时钟是内部时钟（ssi\_BAUDR \* BAUDR）的整数倍。数据需要在半个时钟周期（BAUDR/2）内传输，因此DDR\_DRIVE\_EDGE的最大值等于[（BAUDR/2）-1]。如果DDR\_DRIVE\_EDGE的编程值为0，则传输数据时会相对于sclk\_out（波特时钟）进行边沿对齐。如果DDR\_DRIVE\_EDGE的编程值为1，则数据在sclk\_out边沿之前一个ssi\_th发送

**注意**

如果波特率编程为2，则数据将始终沿对齐。

[图160](#_bookmark271)、[图161](#_bookmark272)和图[162显示](#_bookmark273)了如何使用DDR\_DRIVE\_EDGE寄存器的默认值传输数据的示例这些示例中的绿色箭头表示驱动数据的点。所有这些示例中的波特率使用都是12。在[图160](#_bookmark271)中，数据的传输边缘和驱动边缘相同。这是DDR模式下的默认

*图160.DDR\_DRIVE\_EDGE = 0时传输数据*

ssi\_ssi\_sclk\_outss\_0\_n

txd[N:0]

仪器 A3 A2 A1 A0 D3 D2 D1 D0

rxd[N:0]

ssi\_oe\_n[N:0]

INST =指令阶段A3、A2、A1、A0 =地址寄存器

D3、D2、D1、D0 =数据流

[图160](#_bookmark271)显示了数据的发射沿和驱动沿相同时的默认行为

*图161. 使用DDR\_DRIVE\_EDGE = 1传输数据*

ssi\_ssi\_sclk\_outss\_0\_n

txd[N:0]

仪器 A3 A2 A1 A0 D3 D2 D1 D0

rxd[N:0]

ssi\_oe\_n[N:0]

INST =指令阶段A3、A2、A1、A0 =地址寄存器

D3、D2、D1、D0 =数据流

*图162.DDR\_DRIVE\_EDGE = 2时传输数据*

ssi\_ssi\_sclk\_outss\_0\_n

txd[N:0]

仪器 A3 A2 A1 A0 D3 D2 D1 D0

rxd[N:0]

ssi\_oe\_n[N:0]

INST =指令阶段A3、A2、A1、A0 =地址寄存器

D3、D2、D1、D0 =数据流

* + - 1. SPI模式下的XIP模式支持

就地执行（XIP）模式可直接通过APB接口传输SPI数据，而无需写入DW\_apb\_ssi的数据寄存器。当启用XIP缓存时，在DW\_apb\_ssi中启用XIP模式。该控制信号指示APB传输是寄存器读写还是XIP读取。在XIP模式下，DW\_apb\_ssi只期望APB接口上的读取请求。该请求被转换为串行接口上的SPI读取，接收数据后不久，数据就会在同一事务中返回到APB接口

**注意**

* XIP操作期间仅支持APB读取

地址长度来自SPI\_CTRLR0.ADDR\_L字段，来自paddr（[SPI\_CTRLR0.ADDR\_L- 1:0]）的相关位作为地址传输到SPI接口。XIP地址由XIP缓存控制器管理

* + - * 1. XIP模式下的读取操作

XIP操作仅在增强型SPI操作模式（双通道、四通道）下受支持因此，不应将CTRLR0.SPI\_FRF位编程为0。XIP读取操作分为两个阶段:

地址阶段

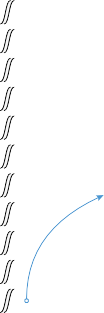
数据阶段

对于XIP读取操作

1. 在CTRLR0寄存器中设置SPI帧格式和数据帧大小值注意，最大数据帧大小的值是32。
2. 在SPI\_CTRLR0寄存器中设置地址长度、等待周期和事务类型请注意，最大地址长度为32。

完成这些设置后，用户可以通过APB接口启动读事务，该事务将使用编程值传输到SPI外设。[图163](#_bookmark274)显示了典型的XIP传输。SPI模式双通道和四通道模式的N值分别为1、3和7。

*图163. XIP模式下的典型读取操作*



* + 1. DMA控制器接口

DW\_apb\_ssi具有内置DMA功能;它有一个到DMA控制器的握手接口，用于请求和控制传输。APB总线用于执行与DMA之间的数据传输

**注意**

当DW\_apb\_ssi连接到DMA控制器时，DMA控制器始终是流控制器;也就是说，它控制块大小。这必须通过DMA控制器中的软件进行编程

580号。DMA发送数据电平（DMATDL）解码值

DW\_apb\_ssi使用两个DMA通道，一个用于发送数据，一个用于接收数据。DW\_apb\_ssi具有以下DMA寄存器:

DMACR

使能DMA操作的控制寄存器

DMATDLR

寄存器，用于设置发送DMA请求时的FIFO电平

DMARDLR

寄存器，用于设置发出DMA请求时的接收FIFO电平

DW\_apb\_ssi使用以下握手信号与DMA控制器连接

* DMA\_TX\_REQ
* DMA\_TX\_单
* DMA\_TX\_ACK
* dma请求
* DMA\_TX\_REQ
* DMA\_TX\_单
* DMA\_TX\_ACK
* dma请求

要在DW\_apb\_ssi上启用DMA控制器接口，必须写入DMA控制寄存器（DMACR）。将1写入DMACR寄存器的TDMAE位字段可使能DW\_apb\_ssi发送握手接口。将1写入DMACR寄存器的RDMAE位字段可使能DW\_apb\_ssi接收握手接口。

[表580](#_bookmark275)提供了不同DMA发送数据电平值的说明。

|  |  |
| --- | --- |
| DMATDL值 | 描述 |
| 0000\_0000 | 当发送FIFO中存在零数据条目时，dma\_tx\_req被置位 |
| 0000\_0001 | 当发送FIFO中存在一个或更少的数据条目时，dma\_tx\_req被置位 |

|  |  |
| --- | --- |
| 0000\_0010 | 当发送FIFO中存在两个或更少数据条目时，dma\_tx\_req置位 |
| … | … |
| 0000\_1101 | 当发送FIFO中存在13个或更少数据条目时，dma\_tx\_req置位 |
| 0000\_1110 | 当发送FIFO中存在14个或更少数据条目时，dma\_tx\_req置位 |
| 0000\_1111 | 当发送FIFO中存在15个或更少数据条目时，dma\_tx\_req置位 |

表581。DMA接收数据电平（DMARDL）解码值

[表581](#_bookmark276)提供了不同DMA接收数据电平值的说明。

|  |  |
| --- | --- |
| DMARDL值 | 描述 |
| 0000\_0000 | 当接收FIFO中存在一个或多个数据条目时，dma\_REQ\_req被断言 |
| 0000\_0001 | 当接收FIFO中存在两个或多个数据条目时，dma\_REQ\_req被置位 |
| 0000\_0010 | 当接收FIFO中存在三个或更多数据条目时，dma\_REQ\_req被置位 |
| … | … |
| 0000\_1101 | 当接收FIFO中存在14个或更多数据条目时，dma\_REQ\_req被置位 |
| 0000\_1110 | 当接收FIFO中存在15个或更多数据条目时，dma\_REQ\_req被置位 |
| 0000\_1111 | 当接收FIFO中存在16个数据条目时，dma\_REQ\_req被置位 |

4.10.11.1. 操作概述

作为一个块流控制设备，DMA控制器由处理器编程，具有DW\_apb\_ssi要发送或接收的数据项的数量（块大小）

该块被分解为多个事务，每个事务都由来自DW\_apb\_ssi的请求发起DMA控制器还必须使用每个DMA请求要传输的数据项（在本例中为DW\_apb\_ssi FIFO条目）的数量进行编程。这也称为突发事务长度。

[图164](#_bookmark277)显示了单个数据块传输，其中DMA控制器中编程的数据块大小为12，突发事务长度设置为4。在这种情况下，数据块大小是突发事务长度的倍数;因此，DMA数据块传输由一系列突发事务组成

**谨慎**

在RP 2040上，SSI的DMA接口的突发事务长度固定为四次传输。SSI.DMARDLR必须始终等于4，这是复位时的值当SSI的FIFO中有一到三个项目时，SSI将请求单个传输，当有四个或更多项目时，SSI将请求4个突发

*图164.将DMA传输分解为突发事务。数据块大小，*DMA.CTLx.BLOCKS\_TS *=*

12数据项

12数据项

4数据项

4数据项

4数据项

DMA

多块传输级

*12. 每个源突发事务的数据项数量，*DMA.CTLx.SRC\_MSIZE*=*

*4. SSI接收FIFO水印级别，*SSI.DMARDLR*+ 1 =*DMA.CTLx.SRC\_MSIZE*= 4*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | |  | | | |
|  | | | DMA  块级 | |  | | |
|  | | | |  | | | |
|  |  | | |  | | |  |
| DMA突发事务1 | |  | DMA突发事务2 | |  | DMA突发事务3 | |

如果DW\_apb\_ssi向该通道发出发送请求，则将四个数据项写入DW\_apb\_ssi发送FIFO。类似地，如果DW\_apb\_ssi向该通道发出接收请求，则从DW\_apb\_ssi接收FIFO读取四个数据项。在写入或读取所有12个数据项之前，必须向该DMA通道发出三个单独的请求。

当编程到DMA控制器中的块大小不是突发事务长度的倍数时，如[图16 -5](#_bookmark278)所示，需要一系列突发事务跟随单个事务以完成块传输。

*图165. 将DMA传输分解为单个事务和突发事务。数据块大小，*DMA.CTLx.BLOCK\_TS *=*

15个数据项

DMA

多块传输级

15个数据项

DMA

块级

DMA突发DMA突发DMA突发DMA单DMA单DMA单事务1事务2事务3事务1事务2 事务3

4数据项4数据项4数据项1数据项1数据项1数据项

*15. 每个突发事务的数据项数量，*DMA.CTLx.DEST\_MSIZE

*= 4。 SSI发送FIFO水印级别，*SSI.DMATDLR *=* DMA.CTLx.DEST\_MSIZE

= 4

* + 1. APB接口

主机处理器通过APB接口访问DW\_apb\_ssi上的数据、控制和状态信息。对DW\_apb\_ssi外设的APB访问将在以下小节中介绍

* + - 1. 控制和状态寄存器APB访问

DW\_apb\_ssi中的控制和状态寄存器是字节可寻址的。DW\_apb\_ssi中控制或状态寄存器的最大宽度因此，对DW\_apb\_ssi控制和状态寄存器的所有读写操作仅需一次APB访问。

* + - 1. 数据寄存器APB访问

DW\_apb\_ssi内的数据寄存器（DR）为32位宽，以保持与最大串行传输大小（数据帧）一致。对DR的APB写操作将数据从FIFO数据移动到发送FIFO缓冲区。来自DR的APB读操作将数据从接收FIFO缓冲区移动到prdata。

DW\_apb\_ssi DR可以在一次APB访问中写入/读取

**注意**

DW\_apb\_ssi中的DR寄存器占用内存映射的64个32位位置，以便于AHB突发传输。APB总线本身没有突发事务，但DW\_apb\_ssi支持在AHB/APB桥的AHB侧发生的AHB突发。写入任何这些地址位置的效果与将数据从FIFO数据总线推入发送FIFO的效果相同。从这些位置读取数据的效果与从接收FIFO将数据弹出到prdata总线上的效果相同。DW\_apb\_ssi上的FIFO缓冲区不可寻址。

* + 1. 登记册一览表

SSI寄存器从基址0x18000000开始（在SDK中定义为XIP\_SSI\_BASE

*582号SSI寄存器列表*

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x00 | [CTRLR0](#_bookmark280) | 控制寄存器0 |
| 0x04 | [CTRLR1](#_bookmark281) | 主控制寄存器1 |
| 0x08 | [SSIENR](#_bookmark282) | SSI启用 |
| 0x0C | [MWCR](#_bookmark283) | Microwire控制 |
| 0x10 | [SER](#_bookmark284) | 从属装置启用 |
| 0x14 | [BAUDR](#_bookmark285) | 波特率 |
| 0x18 | [TXFTLR](#_bookmark286) | TX FIFO阈值电平 |
| 0x1c | [RXFTLR](#_bookmark287) | RX FIFO阈值电平 |
| 0x20 | [TXFLR](#_bookmark288) | TX FIFO电平 |
| 0x24 | [RXFLR](#_bookmark289) | RX FIFO水平 |
| 0x28 | [Sr](#_bookmark290) | 状态寄存器 |
| 0x2c | [IMR](#_bookmark291) | 中断屏蔽 |
| 0x30 | [ISR](#_bookmark292) | 中断状态 |
| 0x34 | [RISR](#_bookmark293) | 原始中断状态 |
| 0x38 | [TXOICR](#_bookmark294) | 发送FIFO溢出中断清除 |
| 0x3c | [RXOICR](#_bookmark295) | RX FIFO溢出中断清除 |
| 0x40 | [RXUICR](#_bookmark296) | RX FIFO下溢中断清除 |
| 0x44 | [MSTICR](#_bookmark297) | 多主中断清除 |
| 0x48 | [ICR](#_bookmark298) | 中断清除 |
| 0x4c | [DMACR](#_bookmark299) | DMA控制 |
| 0x50 | [DMATDLR](#_bookmark300) | DMA TX数据电平 |
| 0x54 | [DMARDLR](#_bookmark301) | DMA RX数据级别 |

|  |  |  |
| --- | --- | --- |
| 偏移 | 名称 | 信息 |
| 0x58 | [IDR](#_bookmark302) | 标识寄存器 |
| 0x5c | [SSI版本ID](#_bookmark303) | 版本ID |
| 0x60 | [Dr0](#_bookmark304) | 数据寄存器0（共36个） |
| 0xf0 | [RX\_SAMPLE\_DLY](#_bookmark305) | RX采样延迟 |
| 0xf4 | [SPI\_CTRLR0](#_bookmark306) | SPI控制 |
| 0xf8 | [TXD\_DRIVE\_EDGE](#_bookmark307) | TX驱动沿 |

583号CTRLR0

寄存器

### [SSI](#_bookmark279):CTRLR0寄存器

**偏移**:0x00

描述

控制寄存器0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一点二十五分 | Reserved. | - | - | - |
| 24 | SSTE | 从机选择切换使能 | RW | 0x0 |
| 23 | Reserved. | - | - | - |
| 二十二点二十一分 | SPI\_FRF | SPI帧格式  0x 0→标准1位SPI帧格式;每个SCK 1位，全双工  0x 1→双SPI帧格式;每个SCK两位，半双工  0x 2→四路SPI帧格式;每SCK四位，半双工 | RW | 0x0 |
| 20:16 | DFS\_32 | 32b传输模式下的数据帧大小每帧n→ n+1个时钟的 | RW | 0x00 |
| 十五点十二分 | CFS | 控制帧大小  每帧n→ n+1个时钟的值 | RW | 0x0 |
| 11 | SRL | 移位寄存器循环（测试模式） | RW | 0x0 |
| 10 | SLV\_OE | 从机输出使能 | RW | 0x0 |
| 九点八分 | TMOD | 传输模式  0x0→发送和接收  0x1→仅发送（不适用于FRF == 0，标准SPI模式）  0x 2→仅接收（不适用于FRF == 0，标准SPI模式）0x 3→ EEPROM读取模式（TX，然后RX; RX在控制数据TX'd后开始） | RW | 0x0 |
| 7 | SCPOL | 串行时钟极性 | RW | 0x0 |
| 6 | SCPH | 串行时钟相位 | RW | 0x0 |
| 五比四 | FRF | 帧格式 | RW | 0x0 |
| 三比零 | DFS | 数据帧大小 | RW | 0x0 |

### [SSI](#_bookmark279):CTRLR1寄存器

**偏移**:0x04

584号CTRLR1

寄存器

585号。SSIENR

寄存器

586号 MWCR寄存器

587号SER寄存器

588号BAUDR

寄存器

描述

主控制寄存器1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:16 | Reserved. | - | - | - |
| 15:0 | NDF | 数量的数据帧 | RW | 0x0000 |

[SSI](#_bookmark279):SSIENR寄存器

**偏移量**:0x 08

描述

SSI启用

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - | - |
| 0 | 简体中文 | SSI使能 | RW | 0x0 |

### [SSI](#_bookmark279):MWCR寄存器

**偏移**:0x0c

描述

Microwire控制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:3 | Reserved. | - | - | - |
| 2 | MHS | 微丝握手 | RW | 0x0 |
| 1 | MDD | 微丝控制 | RW | 0x0 |
| 0 | MWMOD | 微丝传输模式 | RW | 0x0 |

### [SSI](#_bookmark279):SER寄存器

**偏移**:0x10

描述

从属装置启用

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - |
| 0 | 对于每个比特:  0→未选择从机1→已选择从机 | RW | 0x0 |

### [SSI](#_bookmark279):BAUDR寄存器

**偏移**:0x14

描述

波特率

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 15:0 | SCKDV | SSI时钟分频器 | RW | 0x0000 |

589号TXFTLR

寄存器

表590。RXFTLR

寄存器

表591。TXFLR

寄存器

表592。RXFLR

寄存器

### [SSI](#_bookmark279):TXFTLR寄存器

**偏移**:0x18

描述

TX FIFO阈值电平

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | TFT | 发送FIFO阈值 | RW | 0x00 |

### [SSI](#_bookmark279):RXFTLR寄存器

**偏移**:0x1c

描述

RX FIFO阈值电平

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | RFT | 接收FIFO阈值 | RW | 0x00 |

### [SSI](#_bookmark279):TXFLR寄存器

**偏移**:0x20

描述

TX FIFO电平

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | TFTFL | 发送FIFO电平 | RO | 0x00 |

### [SSI](#_bookmark279):RXFLR寄存器

**偏移**:0x24

描述

RX FIFO水平

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | RXTFL | 接收FIFO电平 | RO | 0x00 |

[SSI](#_bookmark279):SR寄存器

**偏移**:0x28

描述

状态寄存器

*表593。SR寄存器*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:7 | Reserved. | - | - | - |
| 6 | DCOL | 数据冲突错误 | RO | 0x0 |
| 5 | TXE | 传输错误 | RO | 0x0 |
| 4 | RFF | 接收FIFO已满 | RO | 0x0 |
| 3 | RFNE | 接收FIFO不为空 | RO | 0x0 |
| 2 | TFE | 发送FIFO为空 | RO | 0x0 |
| 1 | TFNF | 发送FIFO未满 | RO | 0x0 |
| 0 | 忙 | SSI忙标记 | RO | 0x0 |

表594。IMR寄存器

表595。ISR寄存器

### [SSI](#_bookmark279):IMR寄存器

**偏移**:0x2c

描述

中断屏蔽

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:6 | Reserved. | - | - | - |
| 5 | MSTIM | 多主机争用中断屏蔽 | RW | 0x0 |
| 4 | 简体中文 | 接收FIFO全中断掩码 | RW | 0x0 |
| 3 | 公司简介 | 接收FIFO溢出中断掩码 | RW | 0x0 |
| 2 | RXUIM | 接收FIFO下溢中断掩码 | RW | 0x0 |
| 1 | TXOIM | 发送FIFO溢出中断掩码 | RW | 0x0 |
| 0 | TXEIM | 发送FIFO空中断屏蔽 | RW | 0x0 |

[SSI](#_bookmark279):ISR寄存器

**偏移**:0x30

描述

中断状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:6 | Reserved. | - | - | - |
| 5 | MSTIS | 多主争用中断状态 | RO | 0x0 |
| 4 | RXFIS | 接收FIFO满中断状态 | RO | 0x0 |
| 3 | RXOIS | 接收FIFO溢出中断状态 | RO | 0x0 |
| 2 | RXUIS | 接收FIFO下溢中断状态 | RO | 0x0 |
| 1 | TXOIS | 发送FIFO溢出中断状态 | RO | 0x0 |
| 0 | TXEIS | 发送FIFO空中断状态 | RO | 0x0 |

[SSI](#_bookmark279):RISR寄存器

**偏移**:0x34

表596。RISR寄存器

597号TXOICR

寄存器

表598。RXOICR

寄存器

表599。RXUICR

寄存器

描述

原始中断状态

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:6 | Reserved. | - | - | - |
| 5 | MSTIR | 多主机争用原始中断状态 | RO | 0x0 |
| 4 | RXFIR | 接收FIFO完整原始中断状态 | RO | 0x0 |
| 3 | RXOIR | 接收FIFO溢出原始中断状态 | RO | 0x0 |
| 2 | RXUIR | 接收FIFO下溢原始中断状态 | RO | 0x0 |
| 1 | TXOIR | 发送FIFO溢出原始中断状态 | RO | 0x0 |
| 0 | TXEIR | 发送FIFO空原始中断状态 | RO | 0x0 |

### [SSI](#_bookmark279):TXOICR寄存器

**偏移**:0x38

描述

发送FIFO溢出中断清除

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - |
| 0 | 读取时清除发送FIFO溢出中断 | RO | 0x0 |

[SSI](#_bookmark279):RXOICR寄存器

**偏移**:0x3c

描述

RX FIFO溢出中断清除

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - |
| 0 | 读取时清除接收FIFO溢出中断 | RO | 0x0 |

[SSI](#_bookmark279):RXUICR寄存器

**偏移**:0x40

描述

RX FIFO下溢中断清除

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - |
| 0 | 读取时清除接收FIFO下溢中断 | RO | 0x0 |

[SSI](#_bookmark279):MSTICR寄存器

**偏移**:0x44

描述

多主中断清除

*600号MSTICR*

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - |
| 0 | 读时清除多主机争用中断 | RO | 0x0 |

寄存器

表601。 ICR寄存器

表602。DMACR

寄存器

表603。DMATDLR

寄存器

表604。DMARDLR

寄存器

### [SSI](#_bookmark279):ICR寄存器

**偏移**:0x48

描述

中断清除

|  |  |  |  |
| --- | --- | --- | --- |
| 比特 | 描述 | 类型 | 复位 |
| 31:1 | Reserved. | - | - |
| 0 | 读取时清除所有活动中断 | RO | 0x0 |

### [SSI](#_bookmark279):DMACR寄存器

**偏移**:0x4c

描述

DMA控制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 三十一比二 | Reserved. | - | - | - |
| 1 | TDMAE | 发送DMA使能 | RW | 0x0 |
| 0 | RDMAE | 接收DMA使能 | RW | 0x0 |

[SSI](#_bookmark279):DMATDLR寄存器

**偏移**:0x50

描述

DMA TX数据电平

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | DMATDL | 传输数据水印级别 | RW | 0x00 |

[SSI](#_bookmark279):DMARDLR寄存器

**偏移**:0x54

描述

DMA RX数据电平

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | DMARDL | 接收数据水印级别（DMARDLR+1） | RW | 0x00 |

### [SSI](#_bookmark279):IDR寄存器

**偏移**:0x58

605号IDR寄存器

表606。SSI版本ID

寄存器

607号 DR0寄存器

表608。RX\_SAMPLE\_DLY

寄存器

表609。SPI\_CTRLR0寄存器

描述

标识寄存器

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:0 | IDCODE | 外围设备识别码 | RO | 0x51535049 |

### [SSI](#_bookmark279):SSI\_VERSION\_ID寄存器

**偏移**:0x5c

描述

版本ID

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:0 | SSI\_COMP\_VERSION | SNPS组件版本（格式X.YY） | RO | 0x3430312a |

### [SSI](#_bookmark279):DR0寄存器

**偏移**:0x60

描述

数据寄存器0（共36个）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:0 | 博士 | 36的第一个数据寄存器 | RW | 0x00000000 |

### [SSI](#_bookmark279):RX\_SAMPLE\_DLY寄存器

**偏移**:0xf0

描述

RX采样延迟

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | RSD | RXD采样延迟（SCLK周期） | RW | 0x00 |

### [SSI](#_bookmark279):SPI\_CTRLR0寄存器

**偏移量**:0xf4

描述

SPI控制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:24 | XIP\_CMD | 在XIP模式下发送的SPI命令（INST\_L = 8位）或附加到地址的SPI命令（INST\_L = 0位） | RW | 0x03 |
| 二十三点十九分 | Reserved. | - | - | - |
| 18 | 中文（简体） | 读数据选通使能 | RW | 0x0 |
| 17 | INST\_DDR\_EN | 指令DDR传输使能 | RW | 0x0 |
| 16 | SPI\_DDR\_EN | SPI DDR传输使能 | RW | 0x0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 十五点十一分 | 等待循环 | 控制帧发送和数据接收之间的等待周期（SCLK周期） | RW | 0x00 |
| 10 | Reserved. | - | - | - |
| 九点八分 | INST\_L | 指令长度（0/4/8/16 b）0x 0→无指令  0x1→ 4位指令0x2→ 8位指令0x3→ 16位指令 | RW | 0x0 |
| 7:6 | Reserved. | - | - | - |
| 五比二 | 地址\_L | 地址长度（0 b-60 b，以4 b为增量） | RW | 0x0 |
| 一比零 | 变速器类型 | 地址和指令传送格式  0x0→命令和地址均采用标准SPI帧格式  0x1→标准SPI格式的命令，FRF指定格式的地址  0x2→命令和地址均采用FRF规定的格式双SPI） | RW | 0x0 |

表610。TXD\_DRIVE\_EDGE

寄存器

### [SSI](#_bookmark279):TXD\_DRIVE\_EDGE寄存器

**偏移量**:0xf8

描述

TX驱动沿

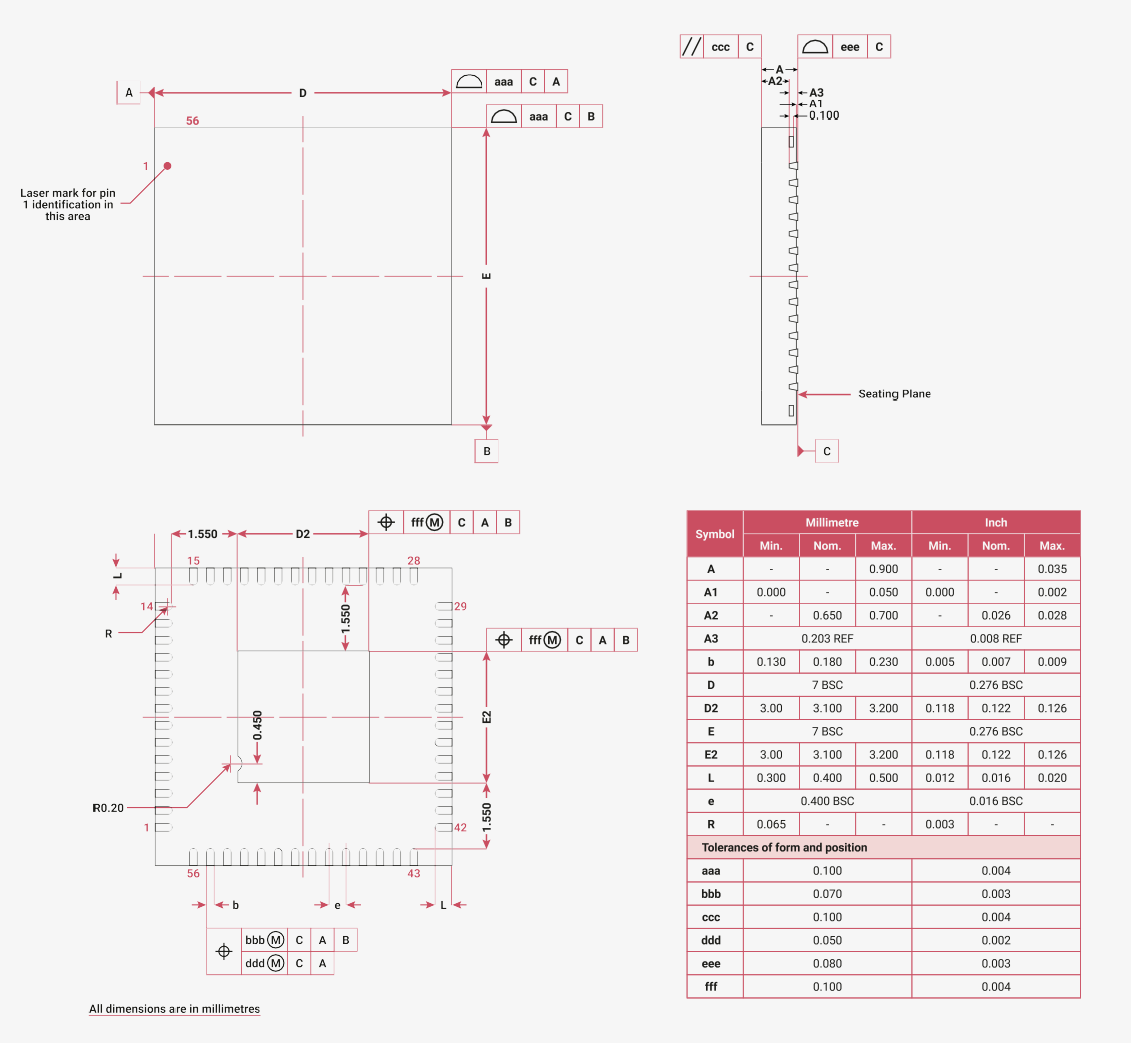
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 比特 | 名称 | 描述 | 类型 | 复位 |
| 31:8 | Reserved. | - | - | - |
| 7:0 | TDE | TXD驱动边缘 | RW | 0x00 |

第5章. 机电

RP2040芯片的物理和电气细节

* 1. 包

*图166.RP 2040 QFN-56的俯视图（左，顶）和侧视图（右，顶）以及仰视图（左，底）*



引脚1

包

**注意**

QFN的中央GND焊盘（或ePad）没有标准尺寸然而，RP 2040上的一个比大多数都小这意味着CAD工具提供的标准0.4mm QFN-56封装可能需要调整。这使得有机会在中央焊盘和外围焊盘之间布线，这有助于在更便宜的PCB上保持电源和接地完整性有关示例，请参见[最小设计示例](https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf#minimal-design-example)

**注意**

导线具有哑光锡（Sn）表面。退火在镀覆后进行，在150°C下烘烤1小时。铅镀层的最小厚度为8微米，中间层材料为CuFe2P（粗糙铜（Cu））。

* + 1. 热特性

表611。RP2040 QFN 56封装的散热数据。

封装的热特性如[表611](#_bookmark308)所示。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| θJA（°C/W） | 温度JT（°C/W） | 温度JB（°C/W） | TJ（°C） | TT（°C） | θJC（°C/W） | θJB（°C/W） |
| 48.00 | 0.80 | 29.20 | 42.00 | 41.8 | 19.01 | 29.03 |

* + 1. 推荐PCB封装

*图167. 推荐用于RP 2040 QFN-56的PCB基底面*

7.75

6.00

3.20

0.20

7.75 6.00

3.20

5.40

0.20

0.875

1.175

0.40

5.40

尺寸单位为mm

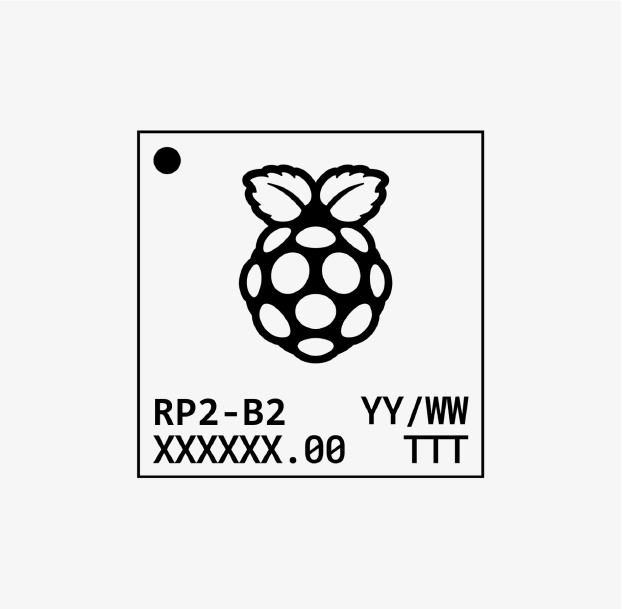
包

* + 1. 包装标志

RP20407×7mmQFN-56包装[的标记见图168](#_bookmark309)，规格见[表612](#_bookmark310)。坐标原点位于包的左下角

图168.包装标记格式

表612。标记要求和尺寸



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 线 | 步骤 | 项目 | 坐标。X | 坐标系Y | Char. 高度 | Char. 宽度 | Char. 空间 |
| 1 | 1 | 引脚1点 | 0.5 | 6 | 0.5 | 0.5 |  |
| 2 | 1 | Logo | 3.5 | 2.395 | 3.83 | 3.05 |  |
| 3 | 1 | RP2-B2 | 0.555 | 1.585 | 0.61 | 0.37 | 0.09 |
| 3 | 2 | YY/WW | 4.235 | 1.585 | 0.61 | 0.37 | 0.09 |
| 4 | 1 | XXXXXX.00 | 0.555 | 0.775 | 0.61 | 0.37 | 0.09 |
| 4 | 2 | TTT  （可选） | 5.155 | 0.775 | 0.61 | 0.37 | 0.09 |

**注意**

在第3行第1步，“RP 2-B2”标记表示器械名称“RP 2”和硅版本“B2”。"

* 1. 储存条件

为了保持裸露RP 2040器械的有效期和地板寿命，根据J- STD（020 E 033 D），RP 2040（归类为MSL 1）的建议储存条件应保持在30°C和85%相对湿度下。

* 1. 焊料轮廓

RP 2040是无铅部件，Tp值为260°C。

所有温度均指封装中心，在组装回流期间（实时错误方向），在封装体表面朝上测量。如果部件回流方向不是正常的活虫组件回流方向（即，死虫），Tp应在活虫Tp的±2°C范围内，并仍满足Tc要求;否则，应调整曲线以达到后者。

*图169. 分类概况（不按比例）*

供应商Tp ≥ Tc

UserTp ≤Tc

Tc

Tc-5 ° C

供应商tp

Tp使用者

Tp

最大升温速率= 3°C/s最大值斜降速率= 6°C/s

不

Tc-5 ° C

p

TL

不

Tsmax

预热区

Tsmin

ts

25

25°C至峰值的时间

时间

**注意**

本文档中的回流焊配置文件用于分类/预处理，并不意味着指定电路板组装配置文件。实际的电路板组装轮廓应根据具体的工艺需求和电路板设计制定，并且不应超过[表613](#_bookmark311)中的参数。

温度

*表613。 焊料轮廓值*

|  |  |
| --- | --- |
| 轮廓特征 | 值 |
| Temperature min（Tsmin） | 摄氏150度 |
| Temperature max（T） | 200°C |
| 从Tsmin到Tsmax的时间（ts） | 60 - 120秒 |
| 升温速率（TL至Tp） | 最大3°C/秒 |
| 液态温度（TL） | 217°C |
| 保持在TL以上的时间（tL） | 60至150秒 |
| 包装体峰值温度（Tp） | 260°C（摄氏度） |
| 分级温度（Tc） | 260°C（摄氏度） |
| 指定分类温度（Tc）5°C以内的时间（tp） | 30秒 |
| 斜降速率（Tp至TL） | 最大6°C/秒 |
| 25°C至峰值温度的时间 | 最多8分钟 |

* 1. 合规

RP 2040符合湿度敏感度1级。

RP 2040符合ECHA于2020年6月25日宣布的REACH高度关注物质（SVHC）要求。

RP 2040符合RoHS指令（EU）2011/65/EU和指令（EU）2015/863的受控环境相关物质的要求和标准。

在RP 2040上进行的包装级可靠性鉴定

* 根据JESD 22-A104的
* HAST符合JESD 22-A110
* HTSL符合JESD 22-A103

**注意**

未进行锡须测试，因为RP 2040是不适用于JEDEC标准（JESD 201A）的底部端接器件（QFN封装）。

图170.RP 2040 QFN-56封装引脚排列

* 1. 引脚说明

### 引脚位置

* + 1. 引脚定义
       1. 引脚类型

表614。引脚类型

*表615。GPIO引脚*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | Number | 类型 | 功率域 | 复位状态 | 描述 |
| GPIO0 | 2 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO1 | 3 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO2 | 4 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO3 | 5 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO4 | 6 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO5 | 7 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO6 | 8 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO7 | 9 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO8 | 11 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO9 | 12 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| 公司简介 | 13 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO11 | 14 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO12 | 15 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO13 | 16 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO14 | 17 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO15 | 18 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |

在下面的GPIO引脚表（[表615](#_bookmark312)）中，引脚类型定义如下。

|  |  |  |
| --- | --- | --- |
| 销式 | 方向 | 描述 |
| 数字化 | 仅输入 | **标准数字**。可编程上拉，下拉，压摆率，施密特触发器和驱动强度。默认驱动强度为4mA。 |
| 数字IO | 双向 |
| 数字输入（FT） | 仅输入 | **容错数字**。这些引脚被描述为容错，在这种情况下，这意味着当引脚低于3.63V且IOVDD为0V时，流入引脚的电流非常小。这些引脚还具有增强的ESD保护功能可编程上拉，下拉，压摆率，施密特触发器和驱动强度。默认驱动强度为4mA。 |
| 数字IO（FT） | 双向 |
| 数字IO/Analog | 双向（数字），输入（Analog） | **标准数字和ADC输入**。可编程上拉，下拉，压摆率，施密特触发器和驱动强度。默认驱动强度为4mA。 |
| USB IO | 双向 | 这些引脚用于USB，并包含内部上拉和下拉电阻，根据USB规范。请**注意**，USB操作需要外部27Ω串联电阻 |
| Anglomerate（XOSC） |  | 振荡器输入引脚用于连接12MHz晶体。或者，XIN可以由方波驱动 |

* + - 1. 端号列表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | Number | 类型 | 功率域 | 复位状态 | 描述 |
| 公司简介 | 27 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO 17 | 28 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO18 | 29 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO19 | 30 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| 公司简介 | 31 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO21 | 32 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO22 | 34 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO23 | 35 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO24 | 36 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| 公司简介 | 37 | 数字IO（FT） | IOVDD | 下拉 | 用户IO |
| GPIO 26/ADC 0 | 38 | 数字IO/Analog | IOVDD /ADC\_AVDD | 下拉 | 用户IO或ADC输入 |
| GPIO27 /ADC 1 | 39 | 数字IO/Analog | IOVDD /ADC\_AVDD | 下拉 | 用户IO或ADC输入 |
| GPIO 28/ADC 2 | 40 | 数字IO/Analog | IOVDD /ADC\_AVDD | 下拉 | 用户IO或ADC输入 |
| GPIO29 /ADC 3 | 41 | 数字IO/Analog | IOVDD /ADC\_AVDD | 下拉 | 用户IO或ADC输入 |

*表616.QSPI引脚*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | Number | 类型 | 功率域 | 复位状态 | 描述 |
| QSPI\_SD3 | 51 | 数字IO | IOVDD |  | QSPI数据 |
| QSPI\_SCLK | 52 | 数字IO | IOVDD | 下拉 | QSPI时钟 |
| QSPI\_SD0 | 53 | 数字IO | IOVDD |  | QSPI数据 |
| QSPI\_SD2 | 54 | 数字IO | IOVDD |  | QSPI数据 |
| QSPI\_SD1 | 55 | 数字IO | IOVDD |  | QSPI数据 |
| QSPI\_CSN | 56 | 数字IO | IOVDD | 上拉 | QSPI芯片选择 |

*617号。晶体振荡器引脚*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | Number | 类型 | 功率域 | 描述 |
| 鑫 | 20 | Anglomerate（XOSC） | IOVDD | 晶体振荡器XIN也可以由方波驱动 |
| XOUT | 21 | Anglomerate（XOSC） | IOVDD | 晶体振荡器 |

*表618。 串行线调试引脚*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | Number | 类型 | 功率域 | 复位状态 | 描述 |
| SWCLK | 24 | 数字输入（FT） | IOVDD | 上拉 | 调试时钟 |
| 社署 | 25 | 数字IO（FT） | IOVDD | 上拉 | 调试数据 |

*表619。其他引脚*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | Number | 类型 | 功率域 | 复位状态 | 描述 |
| 运行 | 26 | 数字输入（FT） | IOVDD | 上拉 | 芯片使能/复位 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | Number | 类型 | 功率域 | 复位状态 | 描述 |
| 测试 | 19 | 数字化 | IOVDD | 下拉 | 测试使能（连接到Gnd） |

620号USB引脚

表621。 电源引脚

表622.数字IO的绝对最大额定值（标准和容错）

*623号。所有引脚的ESD性能，除非另有说明*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数 | 符号 | 最大 | 单位 | 评论 |
| 人体模型 | HBM | 2 | kV | 符合JEDEC规范JS-001- 2012（2012年4月） |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | Number | 类型 | 功率域 | 描述 |
| USB\_DP端口 | 47 | USB IO | USB\_VDD | USB数据+ve. USB操作需要27Ω串联电阻 |
| USB\_DM | 46 | USB IO | USB\_VDD | USB数据-ve. USB操作需要27Ω串联电阻 |

|  |  |  |
| --- | --- | --- |
| 名称 | 数量 | 描述 |
| IOVDD | 第一、十、二十二、三十三、四十二、四十九条 | IO电源 |
| DVDD | 二十三，五十 | 核心供应 |
| VREG\_VIN | 44 | 稳压器输入电源 |
| VREG\_VOUT | 45 | 电压调节器输出 |
| USB\_VDD | 48 | USB供电 |
| ADC\_AVDD | 43 | ADC电源 |
| GND | 57 | 通过中央焊盘 |

* + 1. 引脚规格

以下电气规格是通过在指定温度和电压范围内以及工艺变化的表征获得的，除非规格标记为“模拟”。在这种情况下，数据仅供参考，不作保证。

* + - 1. 绝对最大额定值

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 符号 | 最小 | 最大 | 单位 | 评论 |
| I/O电源电压 | IOVDD | -0.5 | 3.63 | V |  |
| IO时的电压 | V引脚 | -0.5 | IOVDD +0.5 | V |  |

* + - 1. ESD性能

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数 | 符号 | 最大 | 单位 | 评论 |
| 人体模型  仅数字（FT）引脚 | HBM | 4 | kV | 符合JEDEC规范JS-001- 2012（2012年4月） |
| 充电器件模型 | CDM | 500 | V | 符合JESD 22-C101 E  （2009年12月） |

表624. 热性能

*625号 数字IO特性-标准和FT，除非另有说明*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 符号 | 最小 | 最大 | 单位 | 评论 |
| 引脚输入漏电流 | 我在 |  | 1 | μA |  |
| 输入电压高@ IOVDD=1.8V | VIH | 0.65\* IOVDD | IOVDD +0.3 | V |  |
| 输入电压高@ IOVDD=2.5V | VIH | 1.7 | IOVDD +0.3 | V |  |
| 输入电压高@ IOVDD=3.3V | VIH | 2 | IOVDD +0.3 | V |  |
| Input Voltage Low @ IOVDD=1.8V | VIL | -0.3 | 0.35\* IOVDD | V |  |
| Input Voltage Low @ IOVDD=2.5V | VIL | -0.3 | 0.7 | V |  |
| 输入电压低@ IOVDD=3.3V | VIL | -0.3 | 0.8 | V |  |
| IOVDD=1.8V时的输入 | VHYS | 0.1\* IOVDD |  | V | 施密特触发器使能 |
| IOVDD=2.5V时的输入 | VHYS | 0.2 |  | V | 施密特触发器使能 |
| IOVDD=3.3V时的输入 | VHYS | 0.2 |  | V | 施密特触发器使能 |
| 输出电压高@ IOVDD=1.8V | VOH | 1.24 | IOVDD | V | IOH = 2、4、8或  12mA取决于设置 |

* + - 1. 热性能

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 参数 | 符号 | 最小 | 典型 | 最大 | 单位 | 评论 |
| 外壳温度 | TC | -40 |  | 85 | 摄氏度 |  |

* + - 1. IO Electrical Characteristics

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 符号 | 最小 | 最大 | 单位 | 评论 |
| 输出电压高@ IOVDD=2.5V | VOH | 1.78 | IOVDD | V | IOH = 2、4、8或  12mA取决于设置 |
| 输出电压高@ IOVDD=3.3V | VOH | 2.62 | IOVDD | V | IOH = 2、4、8或  12mA取决于设置 |
| 输出电压低@ IOVDD=1.8V | VOL | 0 | 0.3 | V | IOL = 2、4、8或  12mA取决于设置 |
| 输出电压低@ IOVDD=2.5V | VOL | 0 | 0.4 | V | IOL = 2、4、8或  12mA取决于设置 |
| 输出电压低@ IOVDD=3.3V | VOL | 0 | 0.5 | V | IOL = 2、4、8或  12mA取决于设置 |
| 上拉电阻 | RPU | 50 | 80 | kΩ |  |
| 下拉电阻 | RPD | 50 | 80 | kΩ |  |
| 最大总IOVDD电流 | IIOVDD\_MAX |  | 50 | 马 | GPIO和QSPI提供的所有电流之和  销 |
| IO引起的最大总VSS电流（IOVSS） | IIOVSS\_MAX |  | 50 | 马 | 流入GPIO和QSPI的所有电流之和  销 |

*626号USB IO*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 符号 | 最小 | 最大 | 单位 | 评论 |
| 引脚输入漏电流 | 我在 |  | 1 | μA |  |
| 单端输入电压高 | 五、IHSE | 2 |  | V |  |
| 单端输入电压低 | VILSE |  | 0.8 | V |  |
| 差分输入电压高 | VIHDIFF | 0.2 |  | V |  |
| 差分输入电压低 | VILDIFF |  | -0.2 | V |  |
| 输出电压高 | VOH | 2.8 | USB\_VDD | V |  |
| 输出电压低 | VOL | 0 | 0.3 | V |  |
| 上拉电阻  - RPU 2 | RPU2 | 0.873 | 1.548 | kΩ |  |

特性

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 符号 | 最小 | 最大 | 单位 | 评论 |
| 上拉电阻  - RPU 1 2 | RPU1 2 | 1.398 | 3.063 | kΩ |  |
| 下拉电阻 | RPD | 14.25 | 15.75 | kΩ |  |

627号ADC特性

628号。使用方波输入时的振荡器引脚特性

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 符号 | 最小 | 最大 | 单位 | 评论 |
| ADC输入电压范围 | VPIN\_ADC | 0 | ADC\_AVDD | V |  |
| 有效位 | ENOB | 8.7 |  | 比特 | 参见[第4.9.3](#_bookmark209) |
| 已解决的位 |  |  | 12 | 比特 |  |
| ADC输入阻抗 | RIN\_ADC | 100 |  | kΩ |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 符号 | 最小 | 最大 | 单位 | 评论 |
| 输入电压高 | VIH | 0.65\*IOVDD | IOVDD +0.3 | V | 只有XIN。XOUT浮动 |
| 输入电压低 | VIL | 0 | 0.35 \* IOVDD | V | 只有XIN。XOUT浮动 |

有关振荡器的更多详细信息，请参见第2.16，有关晶体使用的信息，请参见[最小设计示例](https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf#minimal-design-example)

* + - 1. 解释GPIO输出电压规格

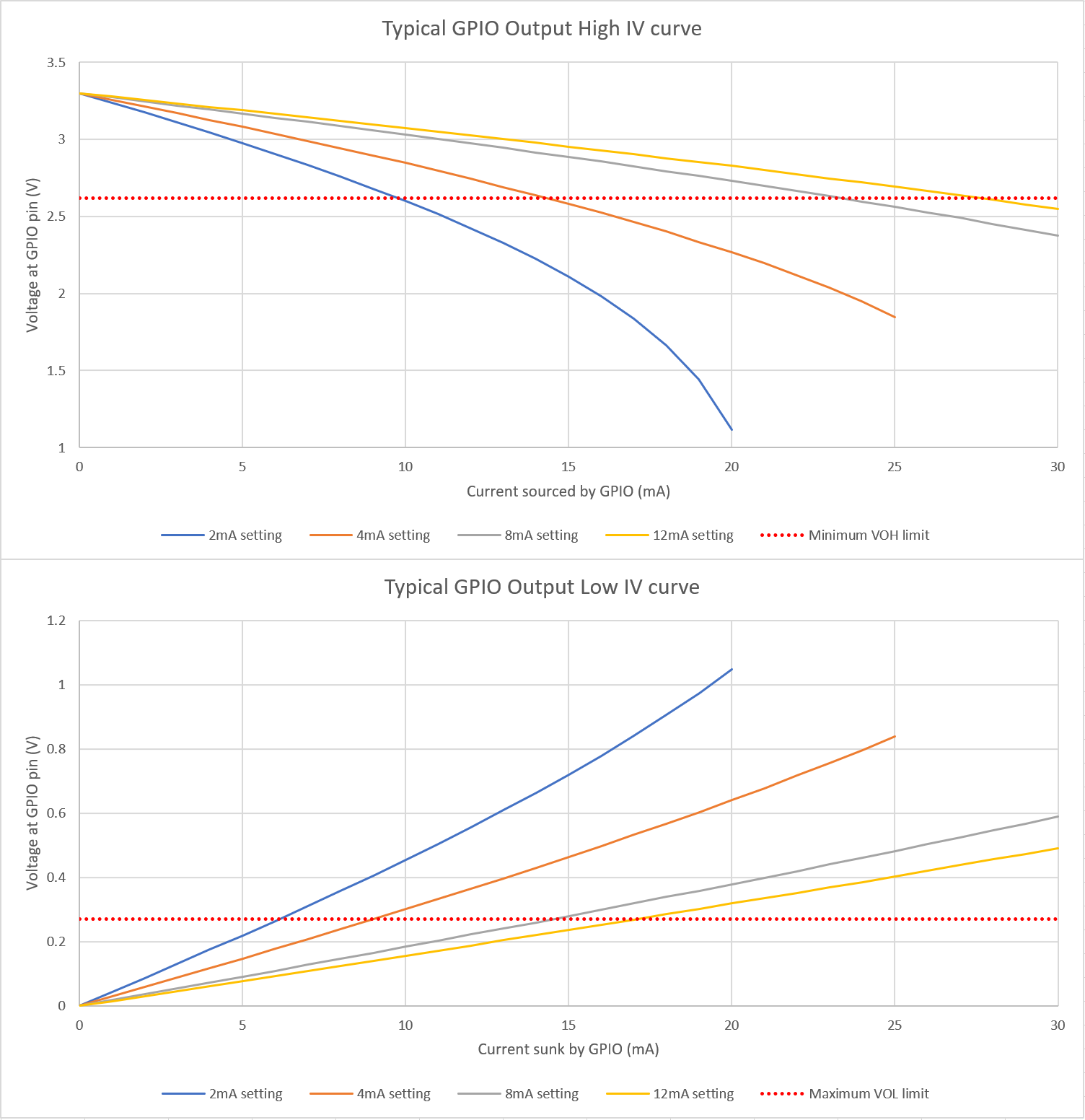
RP2040上的GPIO有四种不同的输出驱动强度，名义上称为2、4、8和12 mA模式。这些不是硬性限制，也不意味着它们将始终提供（或吸收）选定的毫安量。GPIO提供或吸收的电流量取决于其所连接的负载它将尝试将输出驱动到IOVDD电平（或逻辑0情况下的0V），但它能够提供的电流量是有限的，这将取决于所选的驱动强度。因此，电流负载越高，引脚处的电压就越低在某些时候，GPIO将提供如此多的电流，以至于电压如此之低，它不会被连接设备的输入识别为逻辑1[表625](#_bookmark314)中的输出规格旨在尝试并量化从引脚汲取指定电流量时，预期电压可以降低多少

输出高电压（VOH）被定义为当以特定的选定驱动强度驱动到逻辑1时输出引脚可以具有的最低电压;例如，在4mA驱动强度模式下，4mA由引脚提供电源。输出低电压类似，但驱动逻辑0。

除此之外，所有输入输出电流的总和（即当输出被驱动为高电平时），必须不超过IIOVDD\_MAX。类似地，所有被吸收的IO电流的总和当输出被驱动为低电平时）不得超过IIOVSS\_MAX。

图171. GPIO输出的典型电流与电压曲线。

629号。引脚从触发器到焊盘的最小和最大延迟，单位为纳秒。



[图171](#_bookmark315)显示引脚电流负载增加时对输出电压的影响。您可以清楚地看到不同驱动强度的影响;对于给定电流，驱动强度越高，输出电压越接近IOVDD（或0 V）最小VOH和最大VOL限值显示为红色。您可以看到，在每个驱动强度的指定电流下，电压都在允许的限值范围内，这意味着该特定器件可以驱动更多的电流，但仍在VOH/VOL规格范围内这是室温下的典型器件，将有一系列其他器件的电压更接近此极限。当然，如果您的应用不需要如此严格控制的电压，那么您可以从GPIO提供或吸收比所选驱动强度设置更多的电流，但需要进行实验以确定在您的应用中这样做是否确实安全，因为这将超出本规范的范围。

* + - 1. 引脚IO延迟

这些延迟包括PIO的输入/输出映射逻辑、IO多路复用以及实际焊盘延迟到5 pF标称负载。最小值/最大值超过工艺变化、电压（1.1 V +-10%）和温度（-40 C至125 C）的极限值

这些延迟假定IOVDD为1.8 V，PADS\_VSEL置1。IOVDD = 3.3 V时，延迟显著降低，范围也更小。

触发器本身的典型建立时间为10.6 ps，保持时间为2.2 ps。必须考虑触发器和焊盘之间的IO延迟

有关从CLK\_CLK到达PIO中任何触发器到数据在特定GPIO焊盘有效的最小和最大输出延迟，请参见[表629](#_bookmark316)。

|  |  |  |
| --- | --- | --- |
| 垫输出 | 最小延迟（ns） | 最大延迟（ns） |
| GPIO0 | 2.27 | 7.10 |
| GPIO1 | 2.31 | 7.07 |

|  |  |  |
| --- | --- | --- |
| 垫输出 | 最小延迟（ns） | 最大延迟（ns） |
| GPIO2 | 2.33 | 7.08 |
| GPIO3 | 2.24 | 7.00 |
| GPIO4 | 2.30 | 7.07 |
| GPIO5 | 2.34 | 7.10 |
| GPIO6 | 2.32 | 7.10 |
| GPIO7 | 2.39 | 7.09 |
| GPIO8 | 2.34 | 7.09 |
| GPIO9 | 2.38 | 7.08 |
| 公司简介 | 2.33 | 7.07 |
| GPIO11 | 2.36 | 7.08 |
| GPIO12 | 2.35 | 7.04 |
| GPIO13 | 2.31 | 7.08 |
| GPIO14 | 2.38 | 7.06 |
| GPIO15 | 2.33 | 7.05 |
| 公司简介 | 2.34 | 7.09 |
| GPIO17 | 2.37 | 7.09 |
| GPIO18 | 2.37 | 7.04 |
| GPIO19 | 2.27 | 7.10 |
| 公司简介 | 2.38 | 7.09 |
| GPIO21 | 2.05 | 7.10 |
| GPIO22 | 2.34 | 7.07 |
| GPIO23 | 2.16 | 7.05 |
| GPIO24 | 2.12 | 7.06 |
| 公司简介 | 2.26 | 7.10 |
| GPIO26 | 2.32 | 7.09 |
| GPIO27 | 2.26 | 7.08 |
| GPIO28 | 2.34 | 7.09 |
| GPIO29 | 2.30 | 7.07 |

有关从焊盘输入到输入同步器的最小和最大输入延迟，请参见[表630](#_bookmark317)。

*630号。引脚从焊盘输入到输入同步器的最小和最大延迟，单位为纳秒。*

|  |  |  |
| --- | --- | --- |
| 垫输出 | 最小延迟（ns） | 最大延迟（ns） |
| GPIO1 | 1.89 | 5.22 |
| GPIO2 | 1.84 | 5.25 |
| GPIO3 | 1.83 | 5.24 |
| GPIO4 | 1.90 | 5.17 |
| GPIO5 | 1.90 | 5.14 |

|  |  |  |
| --- | --- | --- |
| 垫输出 | 最小延迟（ns） | 最大延迟（ns） |
| GPIO6 | 1.91 | 5.19 |
| GPIO7 | 1.91 | 5.14 |
| GPIO8 | 1.95 | 5.14 |
| GPIO9 | 1.96 | 5.12 |
| 公司简介 | 1.95 | 5.11 |
| GPIO11 | 1.92 | 5.16 |
| GPIO12 | 1.92 | 5.15 |
| GPIO13 | 1.94 | 5.16 |
| GPIO14 | 1.90 | 5.18 |
| GPIO15 | 1.92 | 5.15 |
| 公司简介 | 1.95 | 5.13 |
| GPIO17 | 1.95 | 5.12 |
| GPIO18 | 1.95 | 5.10 |
| GPIO19 | 1.95 | 5.12 |
| GPIO21 | 2.07 | 4.98 |
| GPIO23 | 1.98 | 5.06 |
| GPIO24 | 1.97 | 5.07 |
| 公司简介 | 1.97 | 5.08 |
| GPIO26 | 1.96 | 5.12 |
| GPIO27 | 1.94 | 5.13 |
| GPIO28 | 1.95 | 5.13 |
| GPIO29 | 1.99 | 5.10 |

*631号。引脚从焊盘输入到状态机IN数据触发器（同步器旁路）的最小和最大延迟，*

|  |  |  |
| --- | --- | --- |
| 板输入 | 最小延迟（ns） | 最大延迟（ns） |
| GPIO1 | 2.22 | 5.45 |
| GPIO2 | 2.25 | 5.49 |
| GPIO3 | 2.23 | 5.18 |
| GPIO4 | 2.24 | 5.41 |
| GPIO5 | 2.30 | 5.65 |
| GPIO6 | 2.25 | 5.48 |
| GPIO7 | 2.26 | 5.50 |
| GPIO8 | 2.30 | 5.51 |
| GPIO9 | 2.25 | 5.68 |
| 公司简介 | 2.34 | 5.71 |
| GPIO11 | 2.28 | 5.47 |

有关从焊盘输入到状态机IN数据触发器（同步器旁路）的所有转折点的最小和最大输入延迟，请参见[表631](#_bookmark318)。

|  |  |  |
| --- | --- | --- |
| 板输入 | 最小延迟（ns） | 最大延迟（ns） |
| GPIO12 | 2.29 | 5.40 |
| GPIO13 | 2.25 | 5.47 |
| GPIO14 | 2.24 | 5.41 |
| GPIO15 | 2.23 | 5.47 |
| 公司简介 | 2.30 | 5.42 |
| GPIO17 | 2.28 | 5.44 |
| GPIO18 | 2.28 | 5.34 |
| GPIO19 | 2.30 | 5.50 |
| GPIO21 | 2.16 | 5.79 |
| GPIO23 | 2.33 | 5.53 |
| GPIO24 | 2.28 | 5.60 |
| 公司简介 | 2.29 | 5.53 |
| GPIO26 | 2.28 | 5.38 |
| GPIO27 | 2.27 | 5.39 |
| GPIO28 | 2.24 | 5.28 |
| GPIO29 | 2.33 | 5.47 |

* + - * 1. IOVDD的影响

632号1.8V和3.3V时的最佳和最差焊盘延迟。

633号IOVDD = 3.3V时的IO延迟范围。

上面给出的所有IO延迟都假设IOVDD = 1.8V。将IOVDD增加到3.3V可显著降低焊盘延迟，并且焊盘延迟是上述延迟的很大一部分。有关1.8V和3.3V下最佳和最差焊盘延迟的总结，请参阅[表632](#_bookmark319)

|  |  |  |  |
| --- | --- | --- | --- |
| 路径类型 | IOVDD | 最小延迟（ns） | 最大延迟（ns） |
| 输出 | 1.8V | 1.54 | 3.65 |
| 输出 | 3.3V | 1.11 | 2.14 |
| 输入 | 1.8V | 0.63 | 1.06 |
| 输入 | 3.3V | 0.47 | 0.76 |

更改IOVDD不会影响内核域中的任何逻辑，因此可以将这些差异添加到上述IO延迟表中，以估计IOVDD = 3.3 V时的IO延迟范围（见[表633](#_bookmark320)）。

|  |  |  |  |
| --- | --- | --- | --- |
| 路径群 | IOVDD | 最小延迟（ns） | 最大延迟（ns） |
| 输出 | 1.8V | 2.12 | 7.10 |
| 输出 | 3.3V | 1.69 | 5.59 |
| 同步输入 | 1.8V | 1.83 | 5.25 |
| 同步输入 | 3.3V | 1.67 | 4.95 |
| SM输入 | 1.8V | 2.16 | 5.79 |
| SM输入 | 3.3V | 2.00 | 5.49 |

634号。电源规格

635号基准功率消耗

* 1. 电源

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 电源 | 用品 | Min | Typ | Max | 单位 |
| IOVDDa | 数字IO | 1.62 | 1.8/3.3 | 3.63 | V |
| DVDD b | 数字核心 | 1.05 | 1.1 | 1.16 | V |
| VREG\_VIN | 电压调节器 | 1.62 | 1.8/3.3 | 3.63 | V |
| USB\_VDD | USB PHY | 3.135 | 3.3 | 3.63 | V |
| ADC\_AVDDc | ADC | 1.62 | 3.3 | 3.63 | V |

a如果IOVDD 2.5V，GPIO VOLTAGE\_SELECT寄存器应相应调整详见第2.9。

b短期瞬变应在+/-100 mV范围内

c电压低于2.97 V时，ADC性能将受到影响

* 1. 功耗
     1. 外围设备功耗

基线读数仅在WAKE\_EN0 / WAKE\_EN1寄存器中的时钟源和基本外设（BUSCTRL、BUSFAB、VREG、复位、ROM、SRAM）处于活动状态时获取。时钟设置为默认时钟设置。通过使能WAKE\_EN0 / WAKE\_EN1寄存器中外设的所有时钟源，依次激活每个外设。电流消耗是指使能外设时钟时的电流

|  |  |
| --- | --- |
| 外围 | 典型DVDD电流消耗（μA/MHz） |
| DMA | 2.6 |
| I2c0 | 3.9 |
| I2C1 | 3.8 |
| IO +垫 | 23.6 |
| PIO0 | 12.3 |
| PIO 1 | 12.5 |
| PWM | 5.0 |
| RTC | 1.1 |
| SiO | 1.9 |
| SPI0 | 1.7 |
| SPI1 | 1.8 |
| 定时器 | 1.2 |
| UART0 | 3.5 |
| UART1 | 3.7 |
| 看门狗 | 1.0 |
| XIP | 37.6 |

636号。ADC和USBCTRL的基准功耗

由于48 MHz的固定外部基准时钟以及可变系统时钟输入，ADC和USBCTRL功耗不会随系统时钟呈线性变化（与仅具有系统和/或外设时钟输入的其他外设不同ADC和USBCTRL模块在标准时钟（系统时钟为125 MHz）下的绝对DVDD功耗如下所示

|  |  |
| --- | --- |
| 外围 | 典型DVDD电流消耗（μA/MHz） |
| ADC | 0.1 |
| USBCTRL | 1.3 |

* + 1. 典型用户案例的功耗

以下数据显示了在四种不同的软件用例下，典型（tt）、快速（ff）和慢速（ss）拐角RP 2040设备上的各种电源的电流消耗

**注意**

有关Raspberry Pi Pico的功耗，请参阅**[Raspberry Pi Pico数据表](https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf)**。

首先，'爆米花'（媒体播放器演示）使用VGA，SD卡和音频板。本演示使用VGA视频、I2S音频和4位SD卡访问，系统时钟频率为48 MHz。

**注意**

有关VGA板的更多详细信息，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。

第二，RP2040的BOOTSEL模式这些测量是在总线上有和没有USB活动的情况下进行的，使用Raspberry Pi 4作为主机。

第三个用例使用hello\_dormant二进制文件，将RP 2040置于低功耗状态，即DORMANT模式。最后一个用例使用hello\_sleep二进制代码，将RP 2040置于低功耗状态，即SLEEP模式。

[表637中](#_bookmark321)每个电源都有两列:“典型平均电流”和“最大平均电流”。前者是几秒钟内的平均电流，您可能期望典型的RP 2040在室温和标称电压下消耗（例如，DVDD=1.1V，IOVDD=3.3V等）。“最大平均电流”是在最差情况RP 2040器件上，在极端温度和最大电压（例如，DVDD=1.21V等）。

**注意**

“Popcorn”消耗测量取决于当时正在显示的视频“典型”值是在几秒钟的视频中获得的，具有不同的颜色和强度。“最大”值是在白色视频期间测量的，此时所需电流最高。

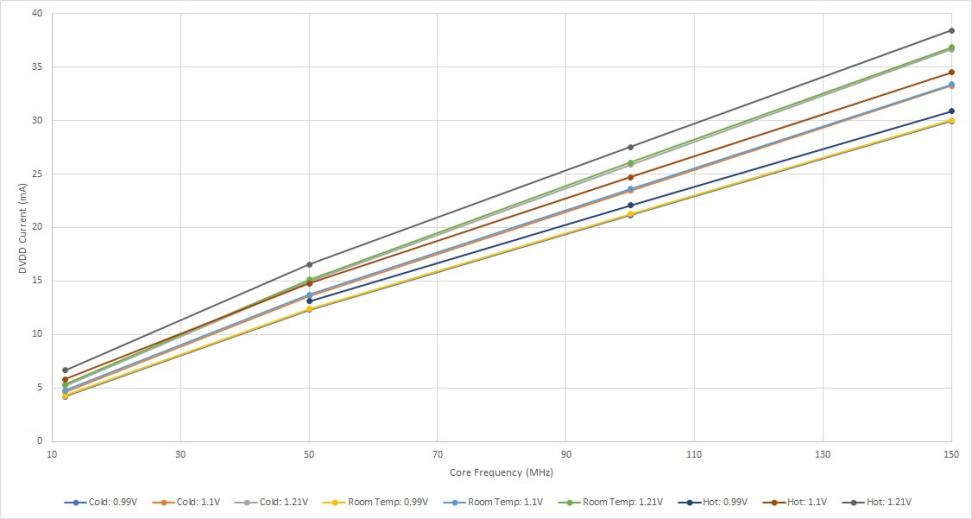
*637号 功耗*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 软件用例 | 典型平均DVDD电流 | 最大平均DVDD电流 | 典型平均IOVDD电流 | 最大平均IOVDD电流 | 典型平均USB\_VDD  电流 | 最大平均USB\_VDD  电流 | 单位 |
| 爆米花 | 10.9 | 16.6 | 24.8 | 35.5 | - | - | 马 |
| BOOTSEL  模式-活动 | 9.4 | 14.7 | 1.2 | 4.3 | 1.4 | 2.0 | 马 |
| BOOTSEL  模式-空闲 | 9.0 | 14.3 | 1.2 | 4.3 | 0.2 | 0.6 | 马 |
| 休眠 | 0.18 | 4.2 | - | - | - | - | 马 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 软件用例 | 典型平均DVDD电流 | 最大平均DVDD电流 | 典型平均IOVDD电流 | 最大平均IOVDD电流 | 典型平均USB\_VDD  电流 | 最大平均USB\_VDD  电流 | 单位 |
| 睡眠 | 0.39 | 4.5 | - | - | - | - | 马 |

图172.运行FFT计算时，典型RP 2040器件的DVDD电流与内核频率

* + - 1. 功耗与频率

为了说明RP 2040工作的内核频率与DVDD电源消耗的电流之间的关系，[图172](#_bookmark322)显示了典型RP 2040器件在不同内核时钟频率下在两个内核上连续运行FFT计算时的测量结果。[图172](#_bookmark322)还显示了外壳温度和DVDD电压对功耗的影响。

附录A:寄存器字段类型

标准类型

RW

处理器可以写入此字段并读回值。

RO

处理器只能读取此字段。

WO

处理器只能写入此字段。

清除类型

SC

这是由处理器写入的单个位，然后在下一个时钟周期清除一个例子是触发一个事件的起始位，然后再次清零，这样事件就不会一直触发。

WC

这是一个位，通常由硬件设置，然后由处理器写入以清除该位。通过使用正常写入或清零别名写入1来清零该位有关明确别名的更多信息，请参阅第2.1.2

FIFO类型

这些字段是特定于实现的

RF

实现定义从硬件读取

WF

实现定义的对硬件的写入。

RWF

实现定义的从硬件读取和写入

附录B:勘误表

硬件块按顺序列出勘误表以数字形式列在相关方框下。

Bootrom

RP2040-E9

|  |  |
| --- | --- |
| 参考 | RP2040-E9 |
| 总结 | ROM引导加载程序无法直接引导到XIP缓存作为SRAM |
| 描述 | 当禁用XIP缓存时，XIP缓存可用作额外的16kB SRAM存储体（参见2.6.3.1）。UF2引导加载程序支持仅RAM的UF2二进制文件，它直接加载到内存中，并通过看门狗重新启动进入。一个UF2二进制文件可以初始化XIP缓存内容和主系统内存，引导加载程序禁用缓存，以便写入缓存内容  但是，看门狗复位会重新启用缓存，因此直接引导到缓存即SRAM别名会立即导致总线故障。缓存内容将被保留，但无法在启动后立即访问 |
| 解决方法 | 在主SRAM中添加代码，以便在访问缓存作为SRAM别名之前重新禁用XIP缓存。当进入一个仅RAM的UF2二进制文件时，引导程序选择主SRAM或缓存即SRAM中最低的加载地址作为入口点，如果两者都被加载，则优先选择主SRAM  此外，如果在启动后立即*写入*0x15.段，则需要对FLUSH寄存器进行虚拟读取，以便在看门狗触发的标记内存刷新期间不会发生缓存即SRAM写入（请参见第2.6.3.2）。 |
| 影响 | RP2040B0，RP2040B1，RP2040B2 |
| 固定 | 文件 |

RP2040-E14

|  |  |
| --- | --- |
| 参考 | RP2040-E14 |
| 总结 | 闪存稀疏或未对齐-UF 2引导加载程序可能无法将二进制UF 2正确写入闪存 |

|  |  |
| --- | --- |
| 描述 | RP2040 UF2文件由256字节的数据页组成，每个数据页都标记为由UF2引导加载程序写入特定地址闪存二进制UF2是其中之一，其每个256字节页被标记为在闪存中的256字节对齐地址处  写入闪存时，必须一次擦除整个4kB闪存扇区，然后才能（重新）写入该扇区内的任何页面。UF2引导加载程序不要求闪存二进制UF2包含扇区内所有页的数据在这种情况下，整个扇区将首先被擦除，任何当前页将被写入，而其余的4kB扇区将不被定义。  当*部分填充*的扇区位于二进制文件的末尾时，这种机制会按照预期工作，这当然是常见的，因为二进制文件不需要是4kB的倍数。  然而，如果*部分填充的*扇区出现在二进制的开始（即，二进制文件在4kB页面上未对齐），或者如果*部分填充的*扇区出现在二进制文件的中间（即二进制文件是稀疏/不连续的），则UF 2文件可能被错误写入  请注意，SDK生成的绝大多数UF 2确实在4kB边界上对齐并且是连续的，但是SDK可能会通过修改链接器脚本或对静态数据提出极端对齐要求来其他语言或工具也可能产生不4KB对齐或不连续的二进制文件 |
| 解决方法 | 解决方法是在包含*某些*页面数据的任何4kB扇区（最后一个扇区除外）中包含*所有*页面的数据  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。 |
| 影响 | RP2040B0，RP2040B1，RP2040B2 |
| 固定 | 文档/软件 |

时钟

RP2040-E7

|  |  |
| --- | --- |
| 参考 | RP2040-E7 |
| 总结 | ROSC和XOSCCOUNT寄存器不可靠 |
| 描述 | ROSC和XOSCUART寄存器用于PHY和PLL等器件的配置，这些器件需要微秒级延迟，而NOP环路不适用，因为UART\_sys频率是可变的。然而，由于同步问题，ROSC:R2和XOSC:R3寄存器不可靠。 |
| 解决方法 | 不要使用ROSC:XOS或XOSC:XOS |
| 影响 | RP2040B0，RP2040B1，RP2040B2 |
| 固定 | 未固定，请勿使用。C SDK不使用这些寄存器 |

RP2040-E10

|  |  |
| --- | --- |
| 参考 | RP2040-E10 |

|  |  |
| --- | --- |
| 总结 | ROSCSTATUS寄存器中的BADWRITE字段不可靠 |
| 描述 | ROSCSTATUS寄存器中的BADWRITE字段用于报告何时将无效值写入其他ROSC寄存器。但是由于内部错误，ROSC:STATUS.BADWRITE字段是不可靠的。 |
| 解决方法 | 不要使用ROSC:STATUS.BADWRITE字段 |
| 影响 | RP2040B0，RP2040B1，RP2040B2 |
| 固定 | 未固定，请勿使用。C SDK不使用此字段 |

DMA

RP 2040-E12

|  |  |
| --- | --- |
| 参考 | RP 2040-E12 |
| 总结 | 当地址环绕或非递增传输序列正在进行时，读取DMAWRITE\_ADDR和READ\_ADDR寄存器会给出错误值 |
| 描述 | DMA的内部WRITE\_ADDR和READ\_ADDR寄存器在每次DMA向其总线管道发出新地址时递增如果处理器在传输序列进行时读取这些寄存器，则DMA报告的值将向下调整，调整量为传输中（即发送到总线管道且尚未完成）的传输次数乘以单个传输大小（以字节为单位）。  添加此逻辑是为了确保读取READ\_ADDR和WRITE\_ADDR反映的是读/写已*完成*的地址，而不仅仅是发出地址的地址 该逻辑没有考虑READ\_ADDR和WRITE\_ADDR对于某些传输模式不是线性递增的，具体地，当CTRL.INCR\_WRITE == 0、CTRL.INCR\_READ == 0或CTRL.RING\_SIZE！ 0。 |
| 解决方法 | 不要检查READ\_ADDR或WRITE\_ADDR来监视传输序列的进度，而是检查  TRANS\_EXTRANS.  TRANS\_STANDARD具有类似的飞行中调整逻辑，但不受此错误的影响，因为它总是线性递减。READ\_ADDR和WRITE\_ADDR的正确值可以根据它们的初始值和TRANS\_DATA计算。 |
| 影响 | RP2040B0，RP2040B1，RP2040B2 |
| 固定 | 文件 |

RP2040-E13

|  |  |
| --- | --- |
| 参考 | RP2040-E13 |
| 总结 | 中止通道后，ABORT状态过早清除，中断可能被断言 |
| 描述 | DMAABORT寄存器用于取消正在进行的传输序列，例如当通道卡在非活动外设DREQ上时。如果在触发中止时，通道当前有任何传输正在进行中（即，传输的读周期已经发生，但写周期还没有发生），ABORT位在清除之前不会等待这些正在进行的传输完成  当传输完成时，由于ABORT位被过早清除，DMA将其视为正常完成。这将设置通道的中断状态标志，假设CTRL.IRQ\_QUIET未被设置。 |

|  |  |
| --- | --- |
| 解决方法 | 在中止通道之前，清除其中断使能。中止通道后，轮询CTRL.BUSY位以等待完成（而不是ABORT位），清除杂散IRQ，并恢复中断使能。 |
| 影响 | RP2040B0，RP2040B1，RP2040B2 |
| 固定 | 软件 |

GPIO /ADC

RP2040-E6

|  |  |
| --- | --- |
| 参考 | RP2040-E6 |
| 总结 | 默认情况下，ADC引脚未禁用GPIO数字输入 |
| 描述 | GPIO 26 -29与ADC输入AIN 0 -3共享释放RUN后，GPIO数字输入使能如果引脚连接到模拟信号进行测量，则这些焊盘上可能会出现意外的信号电平。这不太可能导致问题，因为数字输入默认使能迟滞 |
| 解决方法 | 如果使用模拟输入，则应在启动后尽早禁用数字输入这是在RP2040B2 bootrom中完成的，并在RP2040B0和RP2040B1上的SDK平台设置代码中完成。如果用户希望使用数字输入，则必须启用数字输入。 |
| 影响 | RP2040B0，RP2040B1 |
| 固定 | RP2040B2 bootrom。修复SDK中的RP2040B0和RP2040B1。自定义用户代码应该尽早禁用这些输入。 |

RP2040-E11

|  |  |
| --- | --- |
| 参考 | RP2040-E11 |
| 总结 | ADC中的DNL误差峰值 |
| 描述 | RP 2040 ADC的DNL基本平坦，低于1 LSB。然而，在四个值（512、1,536、2,560和3,584）处，ADC的DNL误差峰值高于此值。ADC的有效位已从9位（模拟）降至8.7位（实测），参见[第4.9.3](#_bookmark209)。DNL误差会在一定程度上限制ADC的性能，具体取决于使用情况。 |
| 解决方法 | 没有一 |
| 影响 | RP2040B0，RP2040B1，RP2040B2 |
| 固定 | 不固定。 |

USB

RP2040-E2

|  |  |
| --- | --- |
| 参考 | RP2040-E2 |
| 总结 | 未清除USB设备终结点中止 |

|  |  |
| --- | --- |
| 描述 | USB设备控制器（第4.1）能够通过设置EP\_ABORT寄存器中端点的位来中止端点上的任何挂起事务由于逻辑错误，如果为EP\_ABORT位设置的任何端点发起事务，则USB设备控制器将在所有端点上永远回复NAK |
| 解决方法 | 不要使用EP\_ABORT位。 |
| 影响 | RP2040B0，RP2040B1 |
| 固定 | RP2040B2 |

RP2040-E3

|  |  |
| --- | --- |
| 参考 | RP2040-E3 |
| 总结 | USB主机:中断端点缓冲区已完成标志可通过不正确的缓冲区选择设置 |
| 描述 | USB主机有两种类型的事务:正常软件启动的传输和中断传输，其中主机在特定时间量后轮询中断端点例如，每1ms轮询一次鼠标以检查移动。 数据传输是单缓冲的，但控制器不会将缓冲选择器重置为零。 这意味着，如果发生软件发起的传输，则中断传输可能会在选择BUF1而不是BUF0的情况下提升缓冲区完成标志。 修复方法是忽略中断端点的BUFF\_CPU\_SHOULD\_HANDLE寄存器。 |
| 解决方法 |  |
| 影响 | RP2040B0，RP2040B1，RP2040B2 |
| 固定 | 软件 |

RP2040-E4

|  |  |
| --- | --- |
| 参考 | RP2040-E4 |
| 总结 | USB主机在单缓冲模式下写入缓冲状态的上半部分 |
| 描述 | USB主机维护一个缓冲区选择器，可在BUF0和BUF1之间切换。这应该只在双缓冲模式下切换，但在单缓冲模式下也可以切换。对于持续多个分组的事务（即，在低速模式下长度超过8字节，在全速模式下长度超过64字节），当缓冲区选择错误地设置为BUF1时，缓冲区状态可以写回到状态寄存器BUF1的一半。注意，这并不影响从缓冲区控制寄存器读取新的缓冲区信息，因为在单缓冲模式下，控制器在读取缓冲区控制寄存器时会忽略缓冲区选择器 |
| 解决方法 | 如果缓冲器选择器为BUF1，则将端点控制寄存器右移16位。当缓冲区被标记为完成时，可以使用BUFF\_CPU\_SHOULD\_HANDLE查找缓冲区选择器的值 |
| 影响 | RP2040B0，RP2040B1，RP2040B2 |
| 固定 | 软件 |

RP2040-E5

|  |  |
| --- | --- |
| 参考 | RP2040-E5 |
| 总结 | USB设备无法在繁忙的USB总线上退出中断状态 |

|  |  |
| --- | --- |
| 描述 | USB总线中断状态由主机向设备发送SE010 ms触发。USB设备控制器在总线复位后需要800μs的空闲（J状态），然后才能进入CONNECTED状态。如果没有此空闲时间，USB设备将不会连接，也不会从主机接收任何数据包，因此不会枚举。  设备插入后立即发生设备重置虽然主机在与新重置的设备通话之前会等待，但连接到同一USB集线器的其他设备也可能正在与主机通信  USB 2.0和USB 3.0集线器具有一个或多个事务转换器，其促进在更高速度总线上的低速和全速事务这取决于集线器的设计，但事务转换器通常在几个端口之间共享  由于RP 2040 USB设备是全速的，当连接到集线器时，其流量将通过事务转换器来。这意味着，如果您在RP 2040旁边插入了另一个设备，则RP 2040可能会看到一些从主机发送到其他设备的消息。如果设备不是很活跃，例如，鼠标每8毫秒轮询一次，这不是问题。然而，某些设备（例如USB串行端口）每30-50μs就会被轮询一次在这种情况下，总线是非常活跃的，将导致RP 2040永远不会退出中断状态，不连接。  RP2040B2中有一个硬件修复，可以避免在休眠状态后需要800μs的空闲时间  此问题有软件解决方法（参见解决方法部分）。用户也可以解决这个问题，关闭USB串行端口或任何其他违规设备，而连接他们的RP2040，然后重新打开他们的USB串行端口。  在较大的集线器上，可以通过将RP 2040远离违规设备（移动到不同的事务转换器上）来解决问题例如，将RP 2040连接到7端口集线器的端口1，并将USB串行控制台连接到端口7，可以解决此问题。将RP 2040连接到单独的USB集线器到任何繁忙的设备也将解决这个问题。 |
| 解决方法 | 使用软件强制USB设备控制器查看800μs的空闲USB总线，以将设备从断开状态移至连接状态。此修复使用连接到GPIO 15的内部调试逻辑，时间较这迫使控制器将DP视为逻辑1（并且将DM视为逻辑0），以使USB设备控制器相信USB总线上存在J状态GPIO15不需要以任何特定的方式绑定此修复工作。相反，我们可以使用第2.19节的输入覆盖功能在软件中强制输入路径。参见<https://github.com/raspberrypi/pico-sdk/blob/master/src/rp2_common/pico_fix/rp2040_usb_device_enumeration/rp2040_usb_device_enumeration.c>。  **注**:解决方法在设备复位期间控制GPIO 15，因此在使用解决方法之前，您需要确保设备重置发生在第一次连接之后，但也可能在主机控制下的其他时间发生  使用TinyUSB和SDK的解决方法很容易，因为上面的源文件包含在库pico\_fix\_rp2040\_usb\_device\_enumeration中（在设备模式下自动添加为TinyUSB的依赖项）。不过，修复本身在默认情况下仍然是关闭的，因为修复对GPIO 15的使用可能与应用程序自己对GPIO 15的使用冲突您可以通过在CMakeLists.txt中将PICO\_RP2040\_USB\_DEVICE\_ENUMERATION\_FIX=1设置为编译器定义的一部分来启用它，或者在tusb\_ENUMERATION.h中将TUD\_OPT\_RP2040\_USB\_DEVICE\_ENUMERATION\_FIX=1设置为编译器定义的一部分来启用它。  即使使用包含硬件修复的RP 2040版本，启用软件解决方案也是安全的（且成本低廉） |
| 影响 | RP2040B0，RP2040B1 |

|  |  |
| --- | --- |
| 固定 | RP 2040 B2。RP 2040 B0、RP 2040 B1的软件解决方案。解决方法不存在于引导程序中的USB大容量存储代码软件解决方法要求在USB总线复位期间使用GPIO 15 |

RP2040-E15

|  |  |
| --- | --- |
| 参考 | RP2040-E15 |
| 总结 | 如果在IN传输期间发生某些总线错误，USB设备控制器将挂起 |
| 描述 | 如果发生以下关键事件序列，USB设备控制器将进入不可恢复状态:   * RP 2040连接到VL805 xHCI控制器，并以全速模式运行 * 集成集线器检测下游端口事务转换器流量和广播上游流量（帧开始令牌）之间即将发生的线路冲突 * 集成集线器在下游进行中的数据包或令牌的PID或CRC部分期间强制执行位填充错误   已知此序列发生在Raspberry Pi 4或Raspberry Pi 400上的面向下游的端口以及数据缓冲区大小超过50字节的Bulk IN端点上。在这种情况下，集成的USB2.0集线器在预期来自主机的SOF数据包时错误地确定了剩余的全速帧时间，并错误地传输了IN令牌，导致后来的ACK应答被损坏并被传播的SOF数据包取代  设备状态机无法正确处理此类数据损坏，必须重置设备控制器。  在商用USB2.0集线器上，以及在VL805 xHCI控制器未提供的根端口上，均未出现此序列。 |
| 变通 | 1. VL805固件版本0138c1   更新的固件已被推送到Pi 4产品上的raspberrypi-bootloaderApt包中的DEFAULT通道这纠正了错误的集线器时间计算。此固件更新不会自动应用，用户必须在Pi 4上运行sudo rpi-update-a并按照屏幕上的说明进行操作。   1. Linux内核xHCI驱动程序补丁   Raspberry Pi 4系列产品的内核更新可用于早于0138c1的 VL805固件版本，避免在全速帧的最后一个微帧此更新在raspberrypi- kernelApt软件包中提供  2）SDK v1.5.0/TinyUSB 0.15.0  TinyUSB从0.15.0版开始为这个错误添加了一个解决方案，这个版本在SDK的v1.5.0版本中被采用dcd\_rp2040驱动器将避免在全速帧的最后200μs期间使能批量IN缓冲器这将使可用的批量输入带宽减少约20%，并有选择地使能帧开始中断。  TinyUSB解决方案对于永远不会连接到易受攻击的VL805端口的实现是不必要的，例如在RP 2040直接连接到板载集线器的电路设计可通过在tusb\_USB. h中定义TUD\_OPT\_RP2040\_USB\_DEVICE\_UFRAME\_FIX=0来禁用该解决方法。 |

|  |  |
| --- | --- |
| 影响 | RP2040B0，RP2040B1，RP2040B2 |
| 固定 | 文档、软件 |

看门狗

RP2040-E1

|  |  |
| --- | --- |
| 参考 | RP2040-E1 |
| 总结 | 看门狗计数每滴答递减两 |
| 描述 | 看门狗（[第4.7](#_bookmark180)）有一个24位计数器，从[LOAD](#_bookmark184)寄存器中设置的用户定义值开始，每一次滴答都递减有一个逻辑错误，这意味着计数器每滴答递减两次，而不是每滴答递减一次在建议的设置中，滴答每隔1μs发生一次，这会使看门狗计数器复位之间的最大时间减半，从约16.7秒降至约8.3秒。 |
| 解决方法 | 在[LOAD](#_bookmark184)中使用所需值的两倍。 |
| 影响 | RP2040B0，RP2040B1，RP2040B2 |
| 固定 | 文档、软件 |

XIP Flash

RP2040-E8

|  |  |
| --- | --- |
| 参考 | RP2040-E8 |
| 总结 | 中止XIP DMA流并立即启动新流时的竞争条件 |
| 描述 | XIP DMA流硬件允许闪存读取的线性序列在后台进行，并由DMA读取，而不会使DMA受到由正常XIP窗口访问引起的总线停顿。数据流开始写入STREAM\_ADDR寄存器，然后写入STREAM\_CTR，可以通过将0写入STREAM\_CTR中途*中止*。  当以这种方式中止流时，软件有足够的时间加载新地址并开始新流，同时中止流的最终SPI/QSPI访问仍在进行中。这导致新加载的流地址在新流序列的第一次数据传输之前递增一次，因此整个流以4字节偏移量发生 |
| 解决方法 | 在清除STREAM\_CTR之后，立即从未缓存的XiP窗口执行虚拟读取，例如（void）\*（io\_ro\_32\*）XIP\_NOCACHE\_NOALLOC\_BASE;。如果XIP流传输仍在进行中，则此虚拟读取将暂停，直到传输完成。然后，通过写入STREAM\_ADDR，然后写入STREAM\_CTR来开始新的流是安全的。 |
| 影响 | RP2040B0，RP2040B1，RP2040B2 |
| 固定 | 文档、软件 |

附录C:可用性

Raspberry Pi了解产品长期可用性对客户的价值，因此旨在尽可能长时间地我们预计RP 2040将至少持续生产到2041年1月

支持

有关支持，请参阅[Raspberry Pi网站的Pico部分](https://www.raspberrypi.com/documentation/microcontrollers/)，并在[Raspberry Pi论坛](https://forums.raspberrypi.com/)上发布问题。

订购代号

638号部件编号

RP2040可以从[Raspberry Pi Direct](https://direct.raspberrypi.com/)批量订购。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 模型 | 订单代码 | 最小订货量 | RRP | 每个芯片的等效价格 |
| 7”卷500 ×  RP 2040芯片 | 粤ICP备09014777号-1 | 1+件/散装 | 400.00美元 | US$0.80 |
| 13”卷轴3，400 ×  RP2040芯片 | 粤ICP备09014443号-1 | 1+件/散装 | 2，380美元 | 0.70美元 |

**注意**

RRP在发布时是正确的，不包括税款。

*639号。文档发布历史*

|  |  |  |
| --- | --- | --- |
| 释放 | 日期 | 描述 |
| 1.0 | 2021年1月21日 | * 初始版本 |
| 1.1 | 2021年1月26日 | * 小更正 * 有关将DMA与ADC配合使用的更多信息 * 阐明了M0+和SIO CPUID寄存器 * 添加了更多关于计时器的讨论 * 更新Windows和macOS构建说明 * 重命名书籍并优化输出PDF |
| 1.2 | 2021年2月1日 | * 小更正 * PIO文档的小改进 * 将缺失的TIMER2和TIMER3寄存器添加到DMA * 解释了如何在Linux * 随C SDK的V1.0.1发布 |
| 1.3 | 2021年2月23日 | * 小更正 * 更改字体 * 关于RP 2040汇/源限值的补充文件 * 社署文件 * 更新MicroPython构建说明 * MicroPython UART示例代码 * 更新了Thonny说明 * 更新项目生成器说明 * 添加FAQ文档 * 添加了勘误表[E7](https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf#errata-e7)、[E8](https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf#errata-e8)和[E9](https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf#errata-e9) |
| 1.3.1 | 2021年3月5日 | * 小更正 * 随着C SDK的V1.1.0版本 * 改进的MicroPython示例 * 改进的引脚排列图 |
| 1.4 | 2021年4月7日 | * 小更正 * 添加了勘误表[E10](https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf#errata-e10) * 关于如何从Github更新C SDK的说明 * 随着C SDK的V1.1.2版本 |

# 附录D:文档发布历史

|  |  |  |
| --- | --- | --- |
| 释放 | 日期 | 描述 |
| 1.4.1 | 2021年4月13日 | * 小更正 * 澄清了文档中的所有源代码都在[3-Clause BSD](https://opensource.org/licenses/BSD-3-Clause)许可证下。 |
| 1.5 | 2021年6月7日 | * 次要更新和更正 * 更新FAQ * 添加SDK发布历史 * 随着C SDK的V1.2.0发布， |
| 1.6 | 2021年6月23日 | * 次要更新和更正 * 艺发局资料更新 * 添加了勘误表[E11](https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf#errata-e11) |
| 1.6.1 | 2021年9月30日 | * 次要更新和更正 * 有关B2发布的信息 * 更新B2发布版本的勘误表 |
| 1.7 | 2021年11月03日 | * 次要更新和更正 * 修正了一些寄存器访问类型和描述 * 增加核心1发射序列信息 * 描述SDK“panic”处理 * 更新picotool文档 * 添加到**附录A**的其他示例:[Raspberry Pi Pico C/C++ SDK](https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf)手册中的App Notes附录 * 随着C SDK的V1.3.0发布， |
| 1.7.1 | 2021年11月04日 | * 次要更新和更正 * USB双缓冲更好的文档 * Picoprobe分支变化 * 更新了文档链接 |
| 1.8 | 2022年6月17日 | * 次要更新和更正 * 更新[了Raspberry Pi Pico](https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf)入门中的Windows安装说明 * SDK配置的附加说明 * RP 2040现在可达到-40 ° C，最低工作温度从-20 ° C更改为-40 ° C * PLL最小VCO从400 MHz增加到750 MHz，提高了工作条件 * 增加回流焊温度曲线 * 添加了勘误表[E12](https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf#errata-e12)、[E13](https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf#errata-e13)和[E14](https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf#errata-e14) * 随着C SDK的V1.3.1版本 |

|  |  |  |
| --- | --- | --- |
| 释放 | 日期 | 描述 |
| 1.9 | 2022年6月30日 | * 次要更新和更正 * 更新Raspberry Pi Pico W发布的VGA板硬件描述 * 随着C SDK的V1.4.0发布， |
| Pico和Pico W数据手册合并为统一的发布历史 | | |
| 2.0 | 2022年12月1日 | * 次要更新和更正 * 添加了RP 2040可用性信息 * 增加了RP 2040储存条件和热特性 * 将SDK库文档替换为指向在线版本的链接 * 更新Picoprobe构建和使用说明 |
| 2.1 | 2023年3月3日 | * 大量的小更新和更正 * 修正Pico W的 SMT封装 * 针对Raspberry Pi Pico C SDK的1.5.0版本进行了更新 * 添加了勘误表[E15](https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf#errata-e15) * 添加了有关新[Pico Windows XP](https://github.com/raspberrypi/pico-setup-windows)的文档 * 添加了有关用于Python开发的[Pico-W-Go](https://marketplace.visualstudio.com/items?itemName=paulober.pico-w-go)扩展的文档 * 在Python文档中添加了无线网络示例 * 增加了包装标记规范 * 添加了RP 2040基线功耗数据 * 在Pico W系列中添加了天线禁止图 |
| 2.2 | 2023年6月14日 | * 次要更新和更正 * 针对Raspberry Pi Pico C SDK的1.5.1版本进行了更新 * 有关Pico W蓝牙支持的文档 |
| 2.3 | 2024年2月2日 | * 许多小的更新和更正 * 更新ROSC登记信息 * 更新了MS Windows和Apple macOS的入门文档 * Raspberry Pi 5发布后的更新 * 重新引入更新的SDK库文档（由于XML冲突，在2.0 * 更新以包括与RP 2040一起使用的晶体的新推荐部件号 * 为Pico和Pico W添加了新的粘贴模板信息 * 对辅助文件的其他更新 |

|  |  |  |
| --- | --- | --- |
| 释放 | 日期 | 描述 |
| 2.4 | 2024年5月2日 | * 许多小的更新和更正 * Pico C SDK API级别文档的修复 * 将picoprobe固件重命名为debugprobe * 澄清了CMake构建配置使用缓存变量，而不是配置变量 * 修复了@asm\_pio装饰器和StateMachine初始化器示例中使用的错误参数名称 * 扩展了MicroPythonrshell示例，包括在设备上加载和运行程序的完整指南 * 添加了演示如何使用OpenOCD从命令行重置Pico的示例 * 增强的VS Code MicroPico插件文档，   插件的新名称，内置FTP服务器的删除，以及一些额外的使用说明   * 添加了关于官方Raspberry Pi Pico VS Code扩展的文档 |

最新版本可以在<https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf>上找到。



