Application Note

SPC11X8/SPD11X8 SIO_DUART 使用指南

Revision 1.1– January 2021

目录

表格列表

图片列表

1 基于 SIO 的 DUART 模块

1.1 概述

SIO_DUART 是一个在 SIO（Spintrol Smart Input/Output）模块上实现的两个 UART（Universal Asynchronous Receiver/Transmitter）接口。

SIO_DUART 提供了 2 个深度为 6 的接收/发送 FIFO 作为与 CPU 通讯的数据接口。数据帧格式为固定的 8bit，偶校验和 1bit 的停止位。双 UART 提供了接收 FIFO 满、奇偶校验错误和接收 FIFO 数据溢出中断，用户可以使用这些中断去实现自己定义的功能，该双 UART 还提供了数据溢出标志位（mark），提醒用户在数据接收过程中有数据已经丢失。

1.2 特性

- 可配置的波特率，最高支持 1Mbps
- 内建深度为 6 的接收和发送 FIFO
- 接收数据 1 字节或 6 字节中断触发机制
- 16 bits 系数锁存寄存器存储波特率系数用于产生波特率，范围是 0~65536,例如 100Mhz 系统时钟条件下，最低支持的波特率是 100M/65536 = 1525，如果使用低于 1525 的波特率，需要适当的降低系统时钟使用
- 可检测以下错误：
  - 奇偶校验位错误
  - 接收 FIFO 溢出
- 双 UART 固定特性
  - 8bits 数据位
  - 偶校验
  - 1-bit 停止位

1.3 信号描述

错误!未找到引用源。描述了 SIO_DUART 的外部信号，这些信号通过 GPIO 与片外通信。可通过 SIO 配置工具灵活选择信号所使用的 GPIO。

表 1-1: SIO_DUART 信号描述

信号名	方向	描述
TXD	输出	UART0 串行数据输出
RXD	输入	UART0 串行数据输入
TXD1	输出	UART1 串行数据输出
RXD1	输入	UART1 串行数据输入

2 SIO_DUART 使用方法

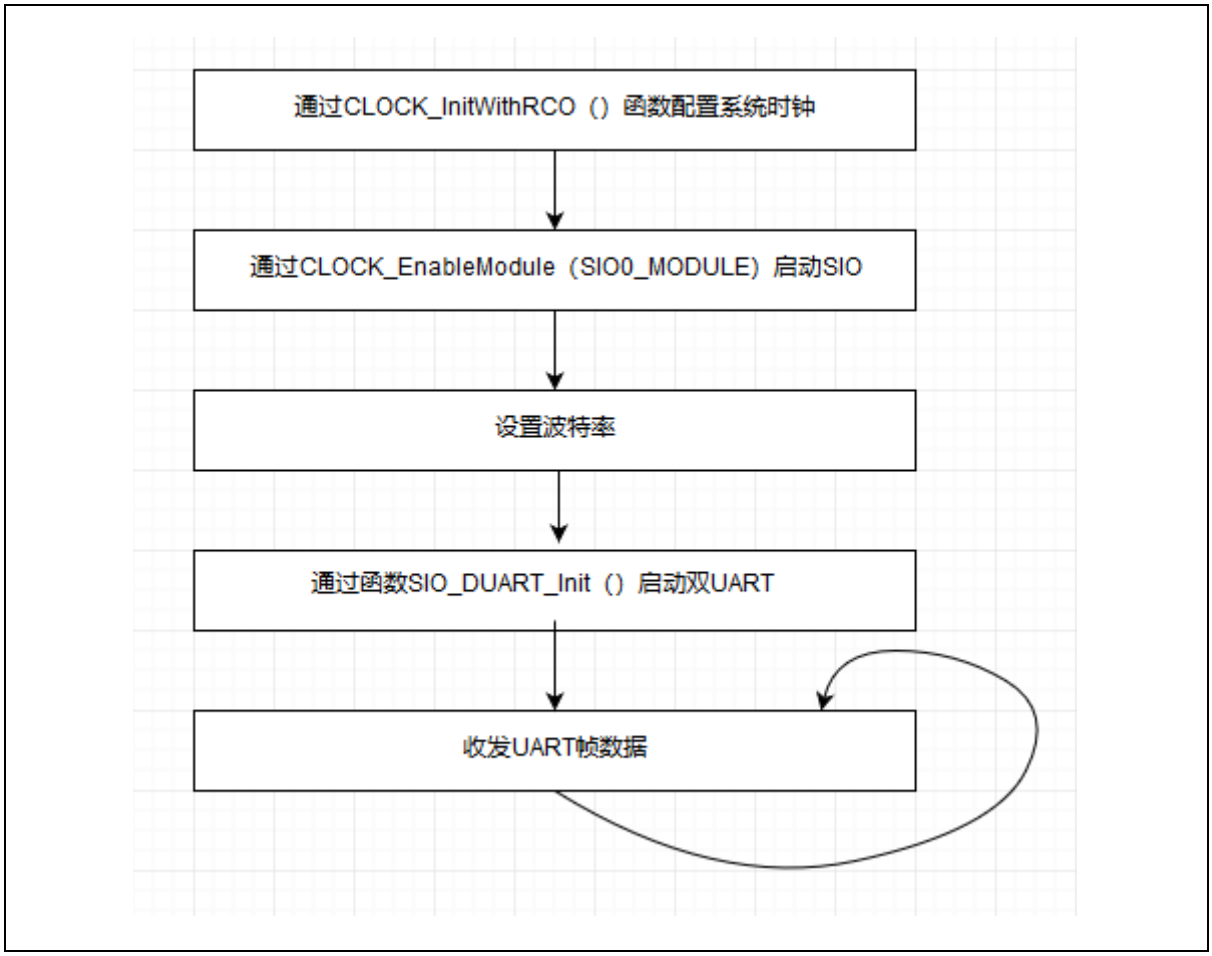
SIO_DUART 的帧格式如图 2-1 所示。图中阴影部分表示可选择配置的帧格式内容，可以在 SIO 配置工具中选择配置方案。

图 2-1: SIO_UART 帧格式

Start Bitt	Data<0>	Data<1>	Data<2>	Data<3>	Data<4>	Data<5>	Data<6>	Data<7>	Data<8>	Parity Bit	Stop Bit
TXD or RXD											
0	LSB								MSB	Even	1

图 2-2 给出了基于 SIO 的 UART 系统在使用时的具体操作流程。Spintrol 提供了相应的软件库来简化该系统的使用。

图 2-2: 基于 SIO 的 UART 系统操作流程



2.1 配置 SIO 时钟

用户可以通过 SIOCLKCTL 寄存器来配置 SIO 时钟，包括时钟的使能和分频比。具体可参见《SPC11X8/SPD11X8 Technical Reference Manual》的第 3 章。当 SIO 被配置用作 DUART 协议引擎时，所允许的 SIO 时钟最高频率为 100MHz，上图中 SIO 时钟默认为系统时钟 100MHz。

2.2 配置 PINMUX 和初始化系统内部 UART

用户需要根据所用的管脚，配置相应的 PINMUX，将管脚切换至 SIO 的输入输出通道。具体可参见《SPC11X8/SPD11X8 Technical Reference Manual》的第 4.3 章节和第 4.4 章节。

2.3 SIO_DUART 的初始化、使能

在 SIO 配置完成后，可通过函数 SIO_DUART_Init 对 SIO_DUART 进行初始化，具体如表 2-1 所示。SIO_DUART_Init 将会自动使能 SIO_DUART。

表 2-1: SIO_DUART_Init 函数介绍

void SIO_DUART_Init (uint32_t u32BaudRate)		
参数名称	**数据类型**	**说明**
u32BaudRate	32 位无符号数	波特率，单位为 bps

使用 SIO_DUART_Enable()与 SIO_DUART_Disable()使能与停用 SIO_DUART。

2.4 波特率配置

SIO_DUAT 中 UART0 和 UART1 的波特率只能是完全相同的，因此只需设置一次波特率就可以使用两个 UART 中的任意一个 UART，SIO_DUART_Init 函数会根据 SIO 的时钟和输入的波特率计算系数，其系数与波特率和 SIO 时钟的关系如式 3-1 所示。

$$\text{divisor} = \frac{\text{F}_{\text{sioclk(Hz)}}}{\text{BaudRate(bps)}} - 1 \qquad (3\text{-}1)$$

SIO_DUART 最高支持波特率为 1.5Mbps。

因为锁存系数的寄存器是一个 16 位的寄存器，所以用户在使用双 UART 时，在 100M 条件下支持的最低波特率为 100Mhz/65536=1525，如果使用比波特率 1525 更低的波特率需要适当的降低 SIO 的时钟。

2.5 数据收发与 FIFO 相关操作

在 SIO_DUART 配置完成并使能后，SIO_DUART 就可以进行数据的收发工作了

在接收方面，SIO_DUART 检测 DUART_RXD 或 DUART_RXD1 信号的下降沿以识别 start bit，并按照 8-bit,偶校验停止位的数据帧格式接收数据。在完整接收到帧中数据字符后，将数据字符存入接收 FIFO。如果接收 FIFO 已满后再接收到一个字符时，最后接收到的字符会丢失，发送中断 SIO0A 或 SIO0B 通知 CPU。在收到帧校验位后，和根据接收到的帧数据计算出的校验值对比，如果错误则置 ULS.RX0_PE 或 ULS.RX1_PE 为 1，发送中断 SIO0A 或 SIOB 通知 CPU。用户可通过 NVIC_EnableIRQ(SIO0A_IRQn 或 SIOB_IRQn)函数启用 UART0 和 UART1 的中断，通过中断可以检测接收发送数据时的出现错误，也可以使用提供的中断函数做一些想实现操作。

表 2-2: SIO_DUART0_ReadBytes 函数介绍

void SIO_DUART_ReadBytes(uint32_t u32UARTx, uint8_t * pu8DataBuf, uint32_t u32Size)		
参数名称	**数据类型**	**说明**

u32UARTx	枚举类型	选择 UART0 或者是 UART1
pu8DataBuf	8 位无符号数	接收数据的数组
u32Size	32 位无符号数	接收数据字符的个数

如表 2-2 所示，用户可以使用 API SIO_UART_ReadByte() 读取接收 FIFO 中的数据字符。通过枚举变量选择使用 UART0 还是 UART1 接收数据，给定一个数组容量大于等于接收字符数量的数组和接收字符的数量就可以进行接收数据了。用户也可以通过函数 SIO_DUALUART_IsRxFIFO1Empty() 和 SIO_DUALUART_ReadFIFO0() 或 SIO_DUALUART_ReadFIFO1()自己创建函数去实现接收数据的操作。

发送数据时，用户可以使用 API SIO_UART_WriteByte()函数向发送 FIFO 中写数据。同接收数据函数一样，通过选择使用 UART0 还是 UART1 发送数据，输入将要发送的数据字节数组和发送字符的数量就可以进行发送数据的操作。用户也可以通过 SIO_DUALUART_IsTxFIFO0Empty() 或 SIO_DUALUART_IsTxFIFO1Empty()去自己创建函数实现数据的发送操作。

表 2-3: SIO_DUART_ WriteBytes 函数介绍

void SIO_DUART_Write(uint32_t u32UARTx, uint8_t *pu8DataBuf, uint32_t u32Size)		
参数名称	**数据类型**	**说明**
u32UARTx	枚举类型	选择 UART0 或者是 UART1
pu8DataBuf	8 位无符号数	发送数据的数组
u32Size	32 位无符号数	发送数据字符的个数

2.6 错误检测

目前 SIO_DUART 支持以下的错误检测：

- 奇偶校验位错误
- 接收 FIFO 溢出

校验位错误即所接收帧的校验位发生错误，通过书写提供的中断函数 SIO_DUART0_UART0ParityerrorCallback 检测是否发生了校验位错误。如果发生了奇偶校验错误会发送中断 SIO0A。用户可以通过对中断函数的操作，处理或释放中断。

接收 FIFO 数据溢出时将会丢失溢出的数据，数据溢出时通过中断函数 SIO_DUART0_RF0OverflowCallback()或 SIO_DUART0_RF1OverflowCallback()告知，产生中断后不会自动清除标志位，需要用户需要手动通过函数 SIO_DUALUART_ClearRx0FIFOOverflow() 或 SIO_DUALUART_ClearRx1FIFOOverflow()进行释放。

3 API 列表

表 3-1 列举了在 SIO_UART 的驱动库中所提供的驱动函数。

表 3-1: API 列表

API 名	描述
void SIO_Init(void)	初始化 SIO 模块，并获得时钟
void SIO_PinInit(void)	按照 SIO 配置工具所选择方案配置 SIO 所使用的 GPIO
void SIO_DUART0_Disable(void)	通过 UFS.SIOINIT 置 0，停用 SIO_DUART
void SIO_DUART0_Enable(void)	通过 UFS.SIOINIT 置 1，启用 SIO_DUART
void SIO_DUART0_WriteBytes(uint32_t u32UARTx, uint8_t *pu8DataBuf , uint32_t u32Size)	UART0/UART1 发送数据，需给出发送的字符和发送字符的数量
void SIO_DUART0_ReadBytes(uint32_t u32UARTx, uint8_t *pu8DataBuf , uint32_t u32Size)	UART0/UART1 接收数据，需给出接收字符的容器（数组）和接收字符的数量
void SIO_DUART0_IRQHandler(void)	中断触发函数，包含双 UART 的全部中断触发条件，包括 UART0/UART1 溢出中断，奇偶校验位错误中断，接收 FIFO 满中断和接收 FIFO 丢失数据标志
uint32_t SIO_DUALUART_IsTxFIFO0Full(void)	判断 UART0 发送 FIFO 是否为满，当其值为 1 时表示 UART0 发送 FIFO 已满
uint32_t SIO_DUALUART_IsTxFIFO1Full(void)	判断 UART1 发送 FIFO 是否为满，当其值为 1 时表示 UART1 发送 FIFO 已满
uint32_t SIO_DUALUART_IsRxFIFO0Full(void)	判断 UART0 接收 FIFO 是否为满，当其值为 1 时表示 UART0 接收 FIFO 已满
uint32_t SIO_DUALUART_IsRxFIFO1Full(void)	判断 UART1 接收 FIFO 是否为满，当其值为 1 时表示 UART1 接收 FIFO 已满
uint32_t SIO_DUALUART_IsRxFIFO0Empty(void)	判断 UART0 接收 FIFO 是否为空，当其值为 1 时表示 UART0 接收 FIFO 为空
uint32_t SIO_DUALUART_IsRxFIFO1Empty(void)	判断 UART1 接收 FIFO 是否为空，当其值为 1 时表示 UART1 接收 FIFO 为空
uint32_t SIO_DUALUART_IsTxFIFO0Empty(void)	判断 UART0 发送 FIFO 是否为空，当其值为 1 时表示 UART0 发送 FIFO 为空
uint32_t SIO_DUALUART_IsTxFIFO1Empty(void)	判断 UART1 发送 FIFO 是否为空，当其值为 1 时表示 UART1 发送 FIFO 为空
uint32_t SIO_DUALUART_IsRx0ParityError(void)	判断 UART0 接收数据是否有奇偶校验位错误，当其值为 1 时，代表 UART0 接收数据发生奇偶校验位错误
uint32_t SIO_DUALUART_IsRx1ParityError(void)	判断 UART1 接收数据是否有奇偶校验位错误，当其值为 1 时，代表发生 UART1 发生奇偶校验位错误

API 名	描述
uint32_t SIO_DUALUART_IsFIFO0Error(void)	判断 UART0 接收 FIFO 是否发生溢出中断，也就是 FIFO 内有 6 个数据，当第 7 个数据收到时会产生中断
uint32_t SIO_DUALUART_IsFIFO1Error(void)	判断 UART1 接收 FIFO 是否发生溢出中断，也就是 FIFO 内有 6 个数据，当第 7 个数据收到时会产生中断
uint32_t SIO_DUALUART_IsRx0FIFOOverflow(void)	判断 UART0 接收 FIFO 是否发生溢出标志，当 CPU 正在通过中断处理 FIFO 中数据时，在这期间收到的字节将会丢失，此时生成此标志位，当此值为 1 时，表示已经发生数据丢失，此标志位需要用户使用 SIO_DUALUART_ClearRx0FIFOOverflow() 手动清除
uint32_t SIO_DUALUART_IsRx1FIFOOverflow(void)	判断 UART1 接收 FIFO 是否发生溢出标志，当 CPU 正在通过中断处理 FIFO 中数据时，在这期间收到的字节将会丢失，此时生成此标志位，当此值为 1 时，表示已经发生数据丢失，此标志位需要用户使用 SIO_DUALUART_ClearRx1FIFOOverflow() 手动清除
void SIO_DUALUART_WriteFIFO0(u8Data)	向 UART0 的 FIFO 中写数据
void SIO_DUALUART_WriteFIFO1(u8Data)	向 UART1 的 FIFO 中写数据
void SIO_DUALUART_ReadFIFO0()	从 UART0 的 FIFO 中读取数据
void SIO_DUALUART_ReadFIFO1()	从 UART1 的 FIFO 中读取数据
void SIO_DUART0_Init()	初始化双 UART，其中调用了 SIO 初始化和 GPIO 引脚初始化，然后配置双 UART 波特率，初始化后，双 UART 拥有相同的波特率

4 代码示例

例 4-1：初始化 SIO_UART

```
void main()
{
  double f64Clock;
  uint32_t u32Baudrate;

  /*** Step 1: Clock initial ***/
  CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);
  /*** Step 2: Delay initial ***/
  Delay_Init();
  /*** Step 3: Configure PINMUX ***/
  PINMUX->GPIO34.bit.MUXSEL = GPIO34_BIT_MUXSEL_UART_TXD;
  PINMUX->GPIO35.bit.MUXSEL = GPIO35_BIT_MUXSEL_UART_RXD;
  /*** Step 4: Initial UART with Baud-Rate configuration for program
   download***/
  UART_Init(UART,256000);
  /*** Step 5: Enable Bus Fault ***/
  SCB->SHCSR |= SCB_SHCSR_BUSFAULTENA_Msk |\
                SCB_SHCSR_USGFAULTENA_Msk |\
                SCB_SHCSR_MEMFAULTENA_Msk ;
  /*** Step 6: Enable SIO module and get SIO Clock ***/
  CLOCK_EnableModule(SIO0_MODULE);
  f64Clock = CLOCK_GetModuleClock(SIO0_MODULE);
  /*** Step 7: set baudrate and Enable SIO_DUART ***/
  u32Baudrate = 256000;
  SIO_DUART0_Init( u32Baudrate );
}
```

例 4-2：收发数据

```
#define BUF_SIZE(6*10)
uint8_t txbyte[BUF_SIZE]
uint8_t rxbyte[BUF_SIZE];
uint32_t i;
/*** Send data example ***/

    for(i = 0; i < BUF_SIZE ; i++)
    {
        txbyte[i] = rand();
    }
```

```
//Send data:
//***note: UARTx is an option, you can choose either UART0 or UART1***//
SIO_DUART0_WriteBytes(UARTx,txbyte,BUF_SIZE);
/*** Receive data example ***/
//Receive data without interrupt:
SIO_UART_ReadBytes(UARTx,rxbyte,BUF_SIZE);

//Receive data with interrupt:
//***note: must set datacnt0 or datacnt1 as global variables***//
While （datacnt0 < BUF_SIZE）
{
}

//interrupt function callback:
void SIO_DUART0_RF0FullCallback(void)
{
        rxbyte[datacnt0++] = SIO_DUALUART_ReadFIFO0();
        SIO_DUART0_ClearNotEmptyInt();
}

void SIO_DUART0_RF1FullCallback(void)
{
        rxbyte[datacnt1++] = SIO_DUALUART_ReadFIFO1();
        SIO_DUART1_ClearNotEmptyInt();
}
```

5 寄存器

5.1 SIO_DUART 寄存器表

表 5-1: SIO_DUART 模块基地址

外设模块	基地址
SIO_DUART	0x4000 B000

表 5-2: SIO_UART 模块寄存器表

寄存器	偏移地址	说明	默认值
UF0	0x0	UART0 FIFO	0x00000000
UF1	0x4	UART1 FIFO	0x00000000
SPCYC	0x8	DUART sample cycle (DIV value / 8)	0x00000000
DIV	0xC	DUART baud rate divisor	0x00000000
ULS	0x10	DUART line status	0x00000000
UFS	0x7C	DUART FIFO status	0x00000000

5.2 SIO_DUART 寄存器

表 5-3: UART0 FIFO (FIFODATA) Layout

UART0 FIFO(FIFO Data Read/Write entry) Offset:0x0 Default:0x00000000							
Access:SIO_DUART -> UF0.all							
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
DATA							
7	6	5	4	3	2	1	0
DATA							

表 5-4: UART0 FIFO (FIFODATA) Field Description

Bits	Field Name	Type	Reset	Description
31:16	RESERVED	RO	0x0	保留
15:0	DATA	RW	0x0	UART0 FIFO 读写数据 写这个寄存器发送数据 读这个寄存器接收数据

表 5-5: UART1 FIFO (FIFODATA) Layout

UART1 FIFO(FIFO Data Read/Write entry) Offset:0x0 Default:0x00000000							
Access:SIO_DUART -> UF1.all							
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
DATA							
7	6	5	4	3	2	1	0
DATA							

表 5-6: UART1 FIFO (FIFODATA) Field Description

Bits	Field Name	Type	Reset	Description
31:16	RESERVED	RO	0x0	保留
15:0	DATA	RW	0x0	UART1 FIFO 读写数据 写这个寄存器发送数据 读这个寄存器接收数据

表 5-7: SPCYC (sample cycle) Layout

Sample cycle(sample 8 times each bit) Offset:0x0 Default:0x00000000							
Access:SIO_DUART -> SPCYC.all							
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
TICK							
7	6	5	4	3	2	1	0
TICK							

表 5-8: SPCYC (sample cycle) Field Description

Bits	Field Name	Type	Reset	Description
31:16	RESERVED	RO	0x0	保留
15:0	TICK	RW	0x0	每个 UART 的每一个 bit 采样 8 次，即每个采样的小周期的数值，是 DIV 数值的八分之一（tick = divisor/8）

表 5-9: DIV(Frequency coefficient) Layout

divisor(frequency coefficient) Offset:0x0 Default:0x00000000							
Access:SIO_DUART -> DIV.all							
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
DIVISOR							
7	6	5	4	3	2	1	0
DIVISOR							

表 5-10: SPCYC (sample cycle) Field Description

Bits	Field Name	Type	Reset	Description
31:16	RESERVED	RO	0x0	保留
15:0	DIV	RW	0x0	系统时钟分频系数，公式$\text{divisor} = \frac{F_{sioclk}(Hz)}{BaudRate(bps)} - 1$ ，写入寄存器产生波特率

表 5-11: ULS(line status) Layout

DUART status(line status) Offset:0x0 Default:0x00000000							
Access:SIO_DUART -> ULS.all							
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
ENABLE	RESERVED						
7	6	5	4	3	2	1	0
RESERVED				UART1Overflow	UART0Overflow	UART1PartyError	UART0PartyError

表 5-12: ULS (line) Field Description

Bits	Field Name	Type	Reset	Description
31:16	RESERVED	RO	0x0	保留
15	UARTENABLE	RW	0x0	双 UART 启动开关，为 1 时开启，为 0 时关闭
14:6	RESERVED	RO	0x0	保留
5	R1_NOTEMPTY	RW	0x0	UART1 的 FIFO 非空状态中断，CPU 必须清除此位才可使用
4	R0_NOTEMPTY	RW	0x0	UART0 的 FIFO 非空状态中断，CPU 必须清除此位才可使用
3	RX1_OF	RW	0x0	UART1 接收 FIFO 数据丢失(标志位) 0：数据未丢失 1：丢失

Bits	Field Name	Type	Reset	Description
2	RX0_OF	RW	0x0	UART0 接收 FIFO 数据丢失（标志位） 0：数据未丢失 1：丢失
1	RX1_PE	RW	0x0	UART1 接收 FIFO 奇偶校验位错误 0：没有奇偶校验位错误 1：发生奇偶校验位错误
0	RX0_PE	RW	0x0	UART0 接收 FIFO 奇偶校验位错误 0：没有奇偶校验位错误 1：发生奇偶校验位错误

表 5-13: UFS (FIFO status) Layout

UFS(FIFO status) Offset:0x0 Default:0x00000000							
Access:SIO_DUART -> UFS.all							
31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
TF1FULL	TF1EMPTY	RESERVED		RF1FULL	RF1EMPTY	RF1OVRUN	SIOINIT
7	6	5	4	3	2	1	0
TF0FULL	TF0EMPTY	RESERVED		RF0FULL	RF0EMPTY	RF0OVRUN	SIOENABLE

表 5-14: UFS (FIFO status) Field Description

Bits	Field Name	Type	Reset	Description
31:16	RESERVED	RO	0x0	Reserved
15	TF1FULL	RO	0x0	UART1 发送 FIFO 满状态 0：发送 FIFO 未满（FIFO 中小于 6 个数据） 1：发送 FIFO 已满（FIFO 中有 6 个数据）
14	TX1EMPTY	RO	0x0	UART1 发送 FIFO 空状态 0：发送 FIFO 未空（FIFO 中有数据） 1：发送 FIFO 空（FIFO 中没有数据）
13：12	RESERVED	RO	0x0	Reserved
11	RF1FULL	RO	0x0	UART1 接收 FIFO 满状态 0：接收 FIFO 未满 1：接收 FIFO 满
10	RF1EMPTY	RO	0x0	UART1 接收 FIFO 空状态 0：接收 FIFO 未空（FIFO 中有数据） 1：接收 FIFO 空（FIFO 中没有数据）
9	RF1OVRUN	RO	0x0	UART1 接收 FIFO 溢出状态（中断） 0：接收 FIFO 未溢出（小于等于 6） 1：接收 FIFO 溢出（收到第 7 个数据）

Bits	Field Name	Type	Reset	Description
8	SIOINIT	RW	0x0	SIO 初始化 0: 关闭 1: 开启
7	TF0FULL	RO	0x0	UART0 发送 FIFO 满状态 0：发送 FIFO 未满（FIFO 中小于 6 个数据） 1：发送 FIFO 已满（FIFO 中有 6 个数据）
6	TF0EMPTY	RO	0x0	UART0 发送 FIFO 空状态 0：发送 FIFO 未空（FIFO 中有数据） 1：发送 FIFO 空（FIFO 中没有数据）
5：4	RESERVED	RO	0x0	保留
3	RF0FULL	RO	0x0	UART0 接收 FIFO 满状态 0：接收 FIFO 未满 1：接收 FIFO 满
2	RF0EMPTY	RO	0x0	UART0 接收 FIFO 空状态 0：接收 FIFO 未空（FIFO 中有数据） 1：接收 FIFO 空（FIFO 中没有数据）
1	RF0OVRUN	RO	0x0	UART0 接收 FIFO 溢出状态（中断） 0：接收 FIFO 未溢出（小于等于 6） 1：接收 FIFO 溢出（收到第 7 个数据）
0	SIOENABLE	RO	0x0	开启 SIO 0：关闭 1：开启

6 修订记录

表 6-1: 文档修订记录

日期	版本	修改内容
2021-1-5	2	修正了 SIO 时钟，添加了单字节触发中断功能