应用笔记

Application Note

AN1182

G32A14xx

外设 **CFGIO** 应用笔记

版本：**V 1.0**

1 引言

该手册主要介绍 G32A14x5 系列产品通用外设模块 CFGIO 模拟各种串行/并行的通信协议：UART、I2C 主机、SPI 主从机、I2S 主从机（支持 PCM 数字音频数据）、LIN 等。此外设的一个主要优势是能够让用户在 MCU 中直接构建自己的外设。

目录

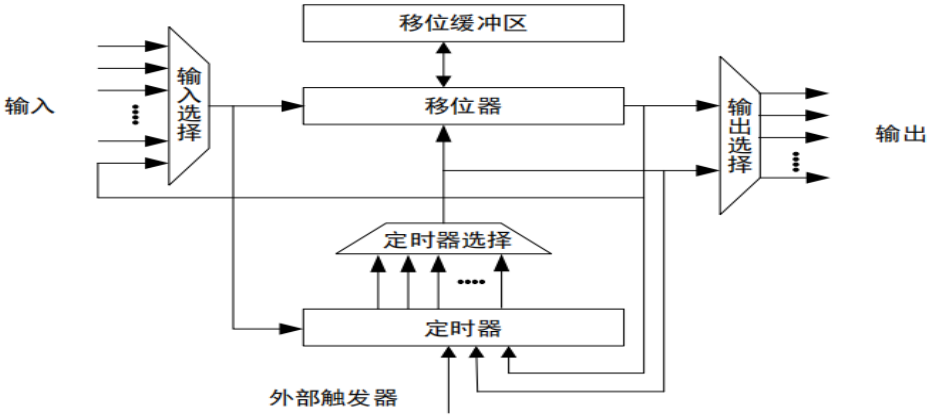
2 通用外设模块 CFGIO 介绍

提供以下主要功能：

- 移位器级联以支持较大的发送大小
- 自动启动/停止位生成
- 具有发送，接收和数据匹配模式的 32 位移位寄存器阵列
- 双缓冲移位器操作，可进行连续数据发送
- 中断，DMA 或轮询的发送/接收操作
- 可编程的波特率与总线时钟频率无关，并在停止模式下支持异步操作
- 高度可配置的 16 位定时器，支持各种内部或外部触发，复位，使能和禁止条件

CFGIO 结构框图如下：

图 1 CFGIO 结构框图



CFGIO 模拟各协议的资源使用情况：

模拟的串行通信协议	使用资源	CFGIO 资源占用
LPUART	发送和接收是两个独立的部分： 发送：1 个定时器，1 个移位器，1 个引脚 接收：1 个定时器，1 个移位器，1 个引脚	50%
LPI2C master	2 个定时器，2 个移位器，2 个引脚	50%
SPI master	2 个定时器，2 个移位器，4 个引脚	50%
SPI slave	1 个定时器，2 个移位器，4 个引脚	50%
I2S master	2 个定时器，2 个移位器，4 个引脚	50%
LIN master	2 个定时器，2 个移位器，4 个引脚	50%
LIN slave	2 个定时器，2 个移位器，4 个引脚	50%

3 使用 CFGIO 模拟 LPUART

3.1 实现原理

3.1.1 LPUART 发送：

可以使用一个定时器，一个移位器和一个引脚来支持 LPUART 发送。自动处理起始位和停止位插入，并且可以使用 DMA 控制器支持多次发送。定时器状态标志用以在发生停止位时进行指示。

间隔和空闲字符需要软件干预，在发送间隔或空闲字符之前，应更改 SSTAFSEL 和 SSTOPFSEL 以发送所需的状态，且要发送的数据必须等于 0xFF 或 0x00。支持第二个停止位需要使用软件将停止位插入数据流。当执行字节写入 CFGIO_SBUF 时，寄存器的地址标记位或其他停止位保持不变。CFGIO 不支持自动插入奇偶校验位。

3.1.2 LPUART 接收：

可以使用一个定时器，一个移位器和一个引脚来支持 LPUART 接收。自动处理起始位和停止位验证，且可以使用 DMA 控制器支持多次传输。定时器的状态标志当在接收到停止位时给出指示。CFGIO 不支持对接收到的数据进行三重表决，数据在每个位的中间仅进行一次采样。另一个定时器可用于对传入的数据进行毛刺过滤，另一个定时器也可用于检测可编程长度的空闲线。中断字符将导致错误标志设置，并且移位缓冲区寄存器将返回 0x00。CFGIO 不支持奇偶校验位的自动验证。

3.2 驱动组件配置及 API 函数说明

用户结构体：

```
typedef struct
{
    CFGIO_UART_DRIVER_DIR_T direction;      /* 驱动方向:接收（Rx）或发送（Tx） */
    CFGIO_DRIVER_TYPE_T transferType;       /* 数据传输类型：中断/DMA/轮询 */
    uint32_t baudrate;                      /* 波特率 */
    uint8_t numBits;                        /* 每字位数 */
    uint8_t cfgioPin;                       /* 用作接收（Rx）或发送（Tx）引脚的 CFGIO 引脚
*/
    uint8_t dmaChannel;                     /* DMA 通道号，仅在 DMA 模式下使用 */
    UART_CALLBACK_T callback;               /* 用户回调函数 */
    void *callbackParam;                    /* 回调函数的参数 */
```

```
} CFGIO_UART_USER_CONFIG_T;
```

初始化函数：

```
STATUS_T CFGIO_UART_Init(                                   /* 返回错误状态 */
    uint32_t instance,                                      /* CFGIO 外设实例编号 */
    const CFGIO_UART_USER_CONFIG_T *configPtr,              /* 指向 CFGIO_UART 用户配置结构的指针 */
    CFGIO_UART_STATE_T *states);                            /* 指向 CFGIO_UART 驱动状态结构的指针 */
```

设置波特率和位宽：

```
STATUS_T CFGIO_UART_SetBaudrate(                            /* 返回错误状态 */
    CFGIO_UART_STATE_T *states,                             /* 指向 CFGIO_UART 驱动状态结构的指针 */
    uint32_t baudrate,                                      /* 所需的波特率 */
    uint8_t numBits);                                       /* 每个字的位数 */
```

阻塞式接收：

```
STATUS_T CFGIO_UART_RxDataBlocking(                         /* 返回错误状态 */
    CFGIO_UART_STATE_T *states,                             /* 指向 CFGIO_UART 驱动状态结构的指针 */
    uint8_t *rxBuffer,                                      /* 指向数据缓冲区的指针 */
    uint32_t rxLength,                                      /* 要接收的数据长度（以字节为单位） */
    uint32_t timeout);                                      /* 超时时间（以毫秒为单位） */
```

阻塞式发送：

```
STATUS_T CFGIO_UART_TxDataBlocking(                         /* 返回错误状态 */
    CFGIO_UART_STATE_T *states,                             /* 指向 CFGIO_UART 驱动状态结构的指针 */
    const uint8_t *txBuffer,                                /* 指向数据缓冲区的指针 */
```

```
    uint32_t txLength,                  /* 要传输的数据长度（以字节为单位） */
    uint32_t timeout);                  /* 超时时间（以毫秒为单位）*/
```

3.3 测试用例

3.3.1 用例介绍

本示例展示了 CFGIO_UART 驱动程序的使用方法。

它使用 CFGIO_D2 和 CFGIO_D3 作为 UART 端口（115200，8N1）。

它在启动时打印欢迎信息，并接受用户输入的命令。

打开 LED 灯。用户可以输入“红色”、“绿色”、“蓝色”等指令，然后红色、绿色或蓝色 LED 灯将会亮起。

3.3.2 硬件准备

- G32A1445 EVAL 评估板×1
- J-Link 烧录器（带 Type-B to A USB 线、FC-20P 双头线/2.54MM 间距）×1
- MINI USB 数据线×1
- 杜邦线若干
- 电脑×1

3.3.3 软件准备

- G32A1xxx_SDK_x.x.x
- Keil MDK 开发环境及 G32A 系列支持包
- SEGGER - J-Link Vx.xx
- 串口助手

3.3.4 测试过程

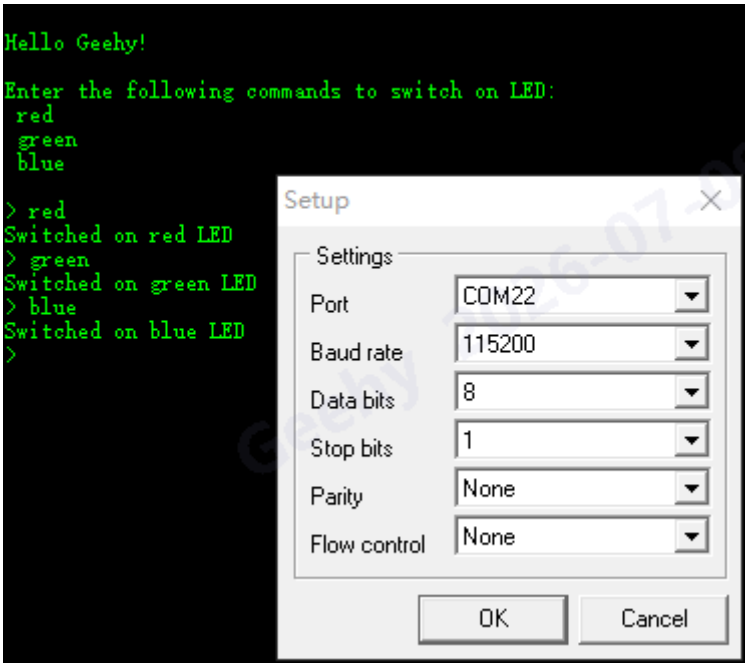
（1） 打开相应的 Keil MDK 工程，编译，下载到评估板：“G32A1xxx_SDK_x.x.x\Examples\G32A1445\CFGIO\CFGIO_UART_Echo\Project\MDK\CFGIO_UART_Echo.uvprojx”，直接编译，下载到 G32A1445 EVAL 评估板，并断开 J-Link 烧录器与评估板的连接。

（2） 设备连接：断开 G32A1445 EVAL 评估板上标号 J9、J10 的跳线帽连接，拿一根杜邦线连接评估板上标号 A0 和 J9，拿另外一根杜邦线一端连接评估板上 A1 和 J10，MINI

USB 数据线一端连接评估板另一端连接电脑，电脑打开串口助手，配置串口参数后打开相应的端口。

（3） 按 RST 复位键，串口助手会收到评估板发来的命令信息，通过串口助手单个字节发送："red\r", "green\r", "blue\r"，评估板对应的 LED 灯亮起。

图 2 CFGIO_UART_Echo 示例串口收发



4 使用 CFGIO 模拟 LPI2C 主机

4.1 实现原理

（1） 可以使用两个定时器，两个移位器和两个引脚来支持 LPI2C 主机模式。定时器分别用于控制移位器，和生成 SCL 输出。移位器用于每个字的发送和接收，当接收器发送器必须发送 0xFF 以使输出处于三态时。

（2） CFGIO 在每个字之后插入一个停止位，以生成/验证 ACK/NACK。在使能 SCL 生成之前，CFGIO 等待首次写入发送数据缓冲区。可以使用 DMA 控制器支持数据发送，且在发送欠载或接收溢出时设置移位器错误标志。

（3） 第一个定时器为整个数据包生成位时钟，因此需要用数据包中时钟沿的总数对比较寄存器进行配置。当引脚等于输出时，定时器使用复位计数器支持时钟延长。第二个定时器使用 SCL 输入引脚来控制发送/接收移位寄存器，这将使 SDA 数据保持时间额外增加 2 个 CFGIO 时钟周期。

（4） 发送中的字都需要为发送移位器和接收移位器服务，发送移位器在接收时必须发送 0xFF，接收移位器返回 SDA 引脚上实际存在的数据。发送移位器将在 SCL 引脚的最后一个下降沿加载 1 个字，如果产生停止条件，则该字应为 0x00；如果产生重复的开始条件，则该字应为 0xFF。

（5） 在主接收器发送的最后一个字期间，应通过软件将发送 SSTOPFSEL 位置 1 以产生 NACK。如果检测到 NACK，则接收移位寄存器将产生一个错误中断，软件负责产生停止或重复的开始条件。如果在主发送期间检测到 NACK，则中断例程应立即用 0x00 或 0xFF 写入发送移位寄存器。然后，软件应等待 SCL 的下一个上升沿，同时禁用两个定时器。然后，在等待建立延迟以重复的开始或停止条件之后，应禁用发送移位器。

（6） 由于同步延迟，发送输出的数据有效时间为 2 个 CFGIO 时钟周期，因此最大波特率是 CFGIO 时钟频率的 6 分频。LPI2C 主数据有效延迟 2 个周期，因为时钟输出在为发送/接收移位器计时之前通过了同步器。由于 SCL 输出与 CFGIO 时钟同步，因此同步延迟为 1 个周期，然后 1 个周期产生输出。

4.2 驱动组件配置及 API 函数说明

用户结构体：

```
typedef struct
{
    uint16_t slaveAddr;      /* 从机地址（7 位） */
    uint8_t sdaPin;          /* 用作 I2C SDA 引脚的 CFGIO 引脚 */
    uint8_t sclPin;          /* 用作 I2C SCL 引脚的 CFGIO 引脚 */
```

```
    uint32_t baudrate;          /* 波特率（单位：赫兹） */
    CFGIO_DRIVER_TYPE_T transferType; /* 数据传输类型：中断/DMA/轮询 */
    uint8_t txDmaChannel;               /* Tx DMA 通道编号 */
    uint8_t rxDmaChannel;               /* Rx DMA 通道编号 */
    I2C_MASTER_CALLBACK_T callback; /* 用户回调函数 */
    void *callbackParam;                /* 回调函数的参数 */
} CFGIO_I2C_MASTER_USER_CONFIG_T;
```

初始化函数：

```
STATUS_T CFGIO_I2C_MasterInit(                                  /* 返回错误状态 */
    uint32_t instance,                                          /* CFGIO 实例编号 */
    const CFGIO_I2C_MASTER_USER_CONFIG_T *configPtr, /* 指向 CFGIO_I2C 主用户配置的指针 */
    CFGIO_I2C_MASTER_STATE_T *states);              /* 指向主上下文结构的指针 */
```

设置波特率函数：

```
STATUS_T CFGIO_I2C_MasterSetBaudrate(       /* 返回错误状态 */
    CFGIO_I2C_MASTER_STATE_T *states,       /* 指向主上下文结构的指针 */
    uint32_t baudrate);       /* 所需的波特率（单位为赫兹） */
```

设置从设备地址：

```
STATUS_T CFGIO_I2C_MasterSetSlaveAddr(        /* 返回错误状态 */
    CFGIO_I2C_MASTER_STATE_T *states,       /* 指向主机上下文结构的指针 */
    uint16_t address);       /* 从机地址（7 位） */
```

非阻塞式发送：

```
STATUS_T CFGIO_I2C_MasterWriteNonBlocking(        /* 返回错误状态 */
    CFGIO_I2C_MASTER_STATE_T *states,       /* 指向主上下文结构的指针 */
    const uint8_t *txBuf,       /* 指向待传输数据的指针 */
```

```
    uint32_t txLen,      /* 待传输数据的字节长度 */
    bool sendStop);      /* 传输完成后是否生成停止条件 */
```

非阻塞式接收：

```
STATUS_T CFGIO_I2C_MasterReadNonBlocking(        /* 返回错误状态 */
    CFGIO_I2C_MASTER_STATE_T *states,      /* 指向主上下文结构的指针 */
    uint8_t *rxBuf,      /* 指向用于存储接收数据的缓冲区的指针 */
    uint32_t rxLen,      /* 要传输的数据的长度（以字节为单位） */
    bool sendStop);      /* 是否在接收后生成停止条件 */
```

4.3 测试用例

4.3.1 用例介绍

本示例展示了 CFGIO_I2C 驱动的使用方法。它使用 CFGIO_I2C 作为 I2C 主设备，使用 LPI2C 作为 I2C 从设备。

4.3.2 硬件准备

- G32A1445 EVAL 评估板×1
- J-Link 烧录器（带 Type-B to A USB 线、FC-20P 双头线/2.54MM 间距）×1
- MINI USB 数据线×1
- 杜邦线若干
- 电脑×1

4.3.3 软件准备

- G32A1xxx_SDK_x.x.x
- Keil MDK 开发环境及 G32A 系列支持包
- SEGGER - J-Link Vx.xx
- 串口助手

4.3.4 测试过程

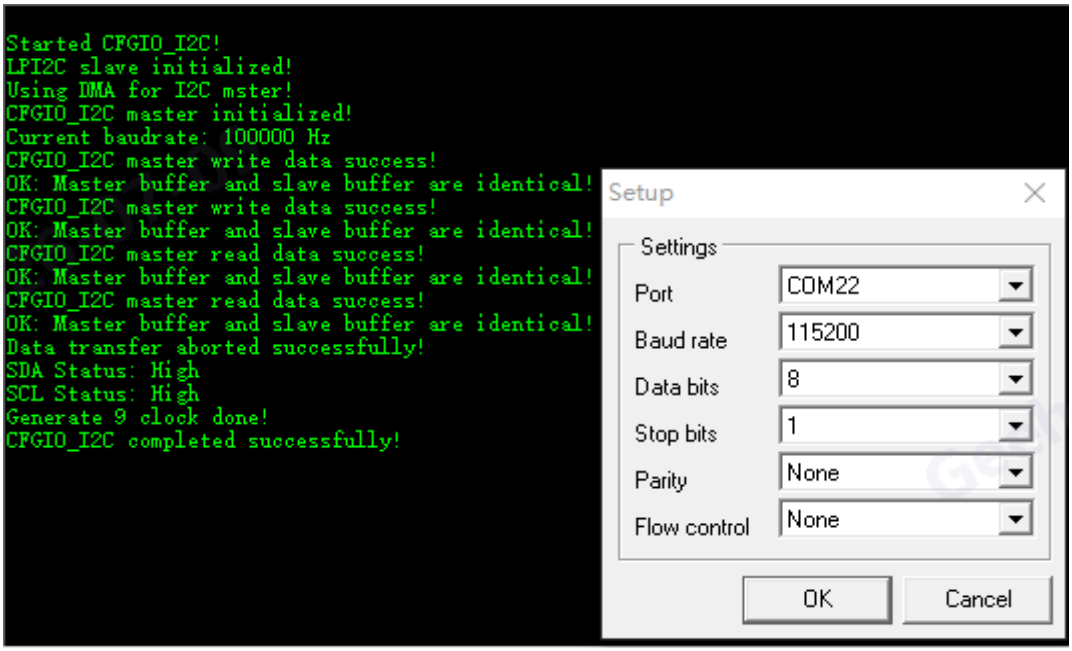
（1） 打开相应的 Keil MDK 工程，编译，下载到评估板：
G32A1xxx_SDK_x.x.x\Examples\G32A1445\CFGIO\CFGIO_I2C_Master\Project\MDK\

CFGIO_I2C_Master.uvprojx”，直接编译，下载到 G32A1445 EVAL 评估板，并断开 J-Link 烧录器与评估板的连接。

（2） 设备连接：拿一根杜邦线连接评估板上标号 A0 和 A3，拿另外一根杜邦线一端连接评估板上 A1 和 A2，MINI USB 数据线一端连接评估板另一端连接电脑，电脑打开串口助手，配置串口参数后打开相应的端口。

（3） 按 RST 复位键，串口助手会收到评估板发来的打印信息，若 I2C 通讯成功则评估板的绿色 LED 灯亮起，否则红色 LED 亮起。

图 3 CFGIO_I2C_Master 示例 I2C 收发



5 使用 CFGIO 模拟 SPI 主/从机

5.1 实现原理

5.1.1 SPI 总线主设备模拟所需的资源配置

可以使用两个定时器，两个移位器和四个引脚来支持 SPI 主机模式。可以支持 CPHA=0 或 CPHA=1，且可以使用 DMA 控制器支持发送。对于 CPHA=1，选择可保持多次发送，并且定时器状态标志可用于指示发送结束。通过内核或 DMA 写入发送缓冲区用于启动每次发送。停止位用于确保从机选择取反到下一次发送之前最少有 1 个时钟周期。由于同步延迟，串行输入数据的建立时间为 1.5 个 CFGIO 时钟周期，因此最大波特率是 CFGIO 时钟频率的 4 分频。

5.1.2 SPI 总线从设备模拟所需的资源配置

可以使用一个定时器，两个移位器和四个引脚来支持 SPI 从机模式。可以支持 CPHA=0 或 CPHA=1，且可以使用 DMA 控制器支持发送。对于 CPHA=1，选择可保持多次发送，且定时器状态标志可用于指示发送结束。在外部从机选择有效之前，必须将发送数据写入发送缓冲寄存器，否则将设置移位器错误标志。由于同步延迟，串行输出数据的输出有效时间为 2.5 个 CFGIO 时钟周期，因此最大波特率是 CFGIO 时钟频率的 6 分频。

5.2 驱动组件配置及 API 函数说明

主设备用户结构体：

```
typedef struct
{
    uint8_t txDMAChannel;                   /* Tx DMA 通道号 */
    uint8_t rxDMAChannel;                   /* Rx DMA 通道号 */
    void *callbackParam;                    /* 回调函数参数 */
    SPI_CALLBACK_T callback;                /* 用户回调，非中断回调 */
    uint8_t ssPin;                          /* 配置到 SS 的 CFGIO 引脚 */
    uint8_t sckPin;                         /* 配置到 SCK 的 CFGIO 引脚 */
    uint8_t misoPin;                        /* 配置到 MISO 的 CFGIO 引脚 */
    uint8_t mosiPin;                        /* 配置到 MOSI 的 CFGIO 引脚 */
    uint8_t clockPha;                   /* 时钟相位：0 = 在时钟前沿采样；1 = 在时钟后沿采样
*/
```

```
    uint8_t clockPol;                                /* 时钟极性：0 = 高电平时钟；1 = 低电平时钟
*/
    CFGIO_SPI_DATA_SIZE_T transferSize;          /* 传输大小（以字节为单位）：1/2/4 */
    CFGIO_SPI_SEND_BIT_FIRST_T firstBitOrder; /* 首位顺序：LSB/MSB */
    CFGIO_DRIVER_TYPE_T cfgspiDirType;          /* cfgio_SPI 驱动类型：中断/轮询/DMA */
    uint32_t baudrate;                               /* 波特率 */
} CFGIO_SPI_MASTER_CFG_T;
```

从设备用户结构体：

```
typedef struct
{
    uint8_t txDMAChannel;                        /* Tx DMA 通道号 */
    uint8_t rxDMAChannel;                        /* Rx DMA 通道号 */
    void *callbackParam;                         /* 回调函数参数 */
    SPI_CALLBACK_T callback;                     /* 用户回调，非中断回调 */
    uint8_t ssPin;                               /* 配置到 SS 的 CFGIO 引脚 */
    uint8_t sckPin;                              /* 配置到 SCK 的 CFGIO 引脚 */
    uint8_t misoPin;                             /* 配置到 MISO 的 CFGIO 引脚 */
    uint8_t mosiPin;                             /* 配置到 MOSI 的 CFGIO 引脚 */
    uint8_t clockPha;                    /* 时钟相位：0 = 在时钟前沿采样；1 = 在时钟后沿采样
*/
    uint8_t clockPol;                            /* 时钟极性：0 = 高电平时钟；1 = 低电平时钟
*/
    CFGIO_SPI_DATA_SIZE_T transferSize;          /* 传输大小（以字节为单位）：1/2/4 */
    CFGIO_SPI_SEND_BIT_FIRST_T firstBitOrder; /* 首位顺序：LSB/MSB */
    CFGIO_DRIVER_TYPE_T cfgspiDirType;          /* cfgio_SPI 驱动类型：中断/轮询/DMA */
} CFGIO_SPI_SLAVE_CFG_T;
```

主设备初始化函数：

```
STATUS_T CFGIO_SPI_MasterInit(                  /* 返回错误状态 */
uint32_t ins,                                   /* CFGIO 的外设实例编号 */
const CFGIO_SPI_MASTER_CFG_T *configPtr,        /* 指向 CFGIO_SPI 主用户配置结构的指针 */
CFGIO_SPI_MASTER_STATE_T *master);              /* 用户定义的 CFGIO_SPI 主设备状态 */
```

从设备初始化函数：

```
STATUS_T CFGIO_SPI_SlaveInit(                   /* 返回错误状态 */
uint32_t ins,                                   /* CFGIO 的外设实例编号 */
const CFGIO_SPI_SLAVE_CFG_T *configPtr,         /* 指向 CFGIO_SPI 从用户配置结构的指针 */
CFGIO_SPI_SLAVE_STATE_T *slave);                /* 用户定义的 CFGIO_SPI 从设备状态 */
```

主设备设置波特率函数：

```
STATUS_T CFGIO_SPI_MasterSetBaudrate(           /* 返回错误状态 */
CFGIO_SPI_MASTER_STATE_T *master,               /* 用户定义的 CFGIO_SPI 主设备状态 */
uint32_t baudrate);                             /* 当前波特率 */
```

主设备非阻塞发送/接收：

```
STATUS_T CFGIO_SPI_MasterTransferNonBlocking( /* 返回错误状态 */
    CFGIO_SPI_MASTER_STATE_T *master,          /* 用户定义的 CFGIO_SPI 主状态 */
    const uint8_t *txData,                     /* 用户发送的数据缓存 */
    uint8_t *rxData,                           /* 用户接收到的数据缓存 */
    uint32_t dataSize);                        /* 待传输数据的字节长度 */
```

从设备非阻塞发送/接收：

```
STATUS_T CFGIO_SPI_SlaveTransferNonBlocking(    /* 返回错误状态 */
    CFGIO_SPI_SLAVE_STATE_T *slave,             /* 用户定义的 CFGIO_SPI 从状态 */
    const uint8_t *txData,                      /* 用户发送的数据缓存 */
    uint8_t *rxData,                            /* 用户接收到的数据缓存 */
```

```
uint32_t dataSize);                                    /* 待传输数据的字节长度 */
```

5.3 测试用例

5.3.1 用例介绍

本用例使用 CFGIO 模块模拟 SPI 通信，需要准备两块开发板。代码中存在 SPI_MASTER 宏定义，当其被写为 1 时，SPI 为主机；当其被写为 0 时，SPI 为从机。

5.3.2 硬件准备

- G32A1445 EVAL 评估板×2
- J-Link 烧录器（带 Type-B to A USB 线、FC-20P 双头线/2.54MM 间距）×1
- MINI USB 数据线×2
- 杜邦线若干
- 电脑×1

5.3.3 软件准备

- G32A1xxx_SDK_x.x.x
- Keil MDK 开发环境及 G32A 系列支持包
- SEGGER - J-Link Vx.xx
- 串口助手

5.3.4 测试过程

（1） 打开相应的 Keil MDK 工程，修改，编译，下载到对应的评估板：“G32A1xxx_SDK_x.x.x\Examples\G32A1445\CFGIO\CFGIO_SPI_Dual\Project\MDK\CFGIO_SPI_Dual.uvprojx”，修改 main.c 文件的 CFGIOSPI_MASTER 宏为 1，编译，下载到一块 G32A1445 EVAL 评估板（作为主设备）；修改 main.c 文件的 CFGIOSPI_MASTER 宏为 0，编译，下载到另一块 G32A1445 EVAL 评估板（作为从设备），并断开 J-Link 烧录器与评估板的连接。

（2） 设备连接：两块评估板的连接如下：

master		slave	
PD9	<==>	PD9	mosi
PA11	<==>	PA11	miso
PA0	<==>	PA0	sck

PA1 <==> PA1 ss

（3） 两根 MINI USB 数据线分别连接到两块评估板，另一端连接电脑，电脑打开串口助手，配置串口参数后打开相应的端口。

（4） 从设备先按 RST 复位键，主设备紧跟着按 RST 复位键，串口助手会收到评估板发来的打印信息，若 SPI 通讯成功则评估板的绿色 LED 灯亮起，否则红色 LED 亮起。

图 4 CFGIO_SPI_Dual.uvprojx 示例 SPI 主设备收发

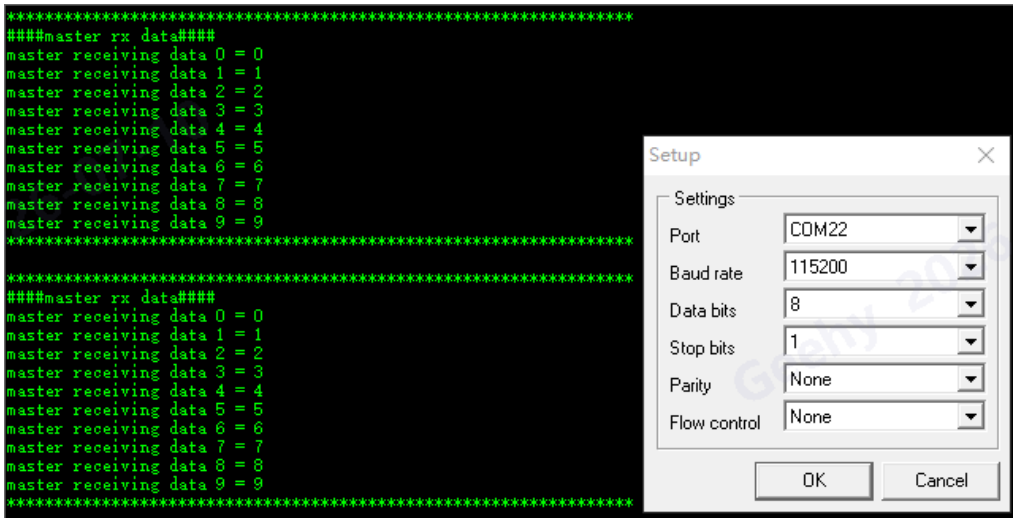
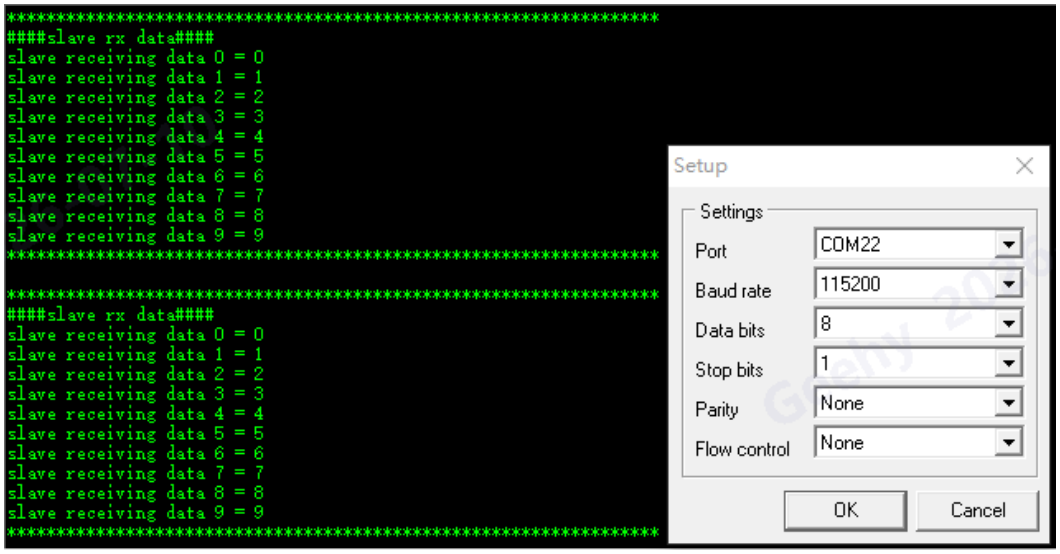


图 5 CFGIO_SPI_Dual.uvprojx 示例 SPI 从设备收发



6 使用 CFGIO 模拟 I2S 主/从机

6.1 实现原理

6.1.1 I2S 总线主设备模拟所需的资源配置

可以使用两个定时器，两个移位器和四个引脚来支持 I2S 主机模式。定时器分别用于生成帧同步和生成位时钟并控制移位器。在使能位时钟和帧同步生成之前，CFGIO 等待首次写入发送数据缓冲区。可以使用 DMA 控制器支持数据发送，且在发送欠载或接收溢出时将设置移位器错误标志。定时器使用起始位来确保在第一个输出数据之前的一个时钟周期生成帧同步。位时钟频率是 CFGIO 时钟频率的偶数整数倍，并且初始帧同步与第一个位时钟沿同时发生。由于同步延迟，接收器输入的建立时间为 1.5 个 CFGIO 时钟周期，因此最大波特率是 CFGIO 时钟频率的 4 分频。

6.1.2 I2S 总线从设备模拟所需的资源配置

可以使用三个定时器，两个移位器和四个引脚来支持 I2S 从机模式。必须在外部帧同步有效之前将发送数据写入发送缓冲寄存器，否则将设置移位器错误标志。因为时钟同步的最大延迟为 1.5 个周期，再加上 1 个周期即可输出数据，所以 I2S 从机的输出有效时间最大为 2.5 个周期。由于同步延迟，串行输出数据的输出有效时间为 2.5 个 CFGIO 时钟周期，因此最大波特率是 CFGIO 时钟频率的 6 分频。定时器 1 输出有效时，定时器 1 检测到位时钟的上升沿，并控制移位寄存器进行 32 位发送。定时器 2 检测到帧同步的下降沿，并产生输出直到位时钟的下降沿。定时器 0 在定时器 2 的输出有效的情况下检测位时钟的下降沿，并在帧长的时间内确认输出。

6.2 驱动组件配置及 API 函数说明

主设备用户结构体：

```
typedef struct
{
    CFGIO_DRIVER_TYPE_T driverType;     /* 驱动类型：中断/轮询/DMA */
    uint32_t baudRate;                  /* 波特率（以赫兹为单位） */
    uint8_t bitsWidth;                  /* 一个字中的位数 - 8 的倍数 */
    uint8_t txPin;                      /* 用于传输的 Cfgio 引脚 */
    uint8_t rxPin;                      /* 用于接收的配置 IO 引脚 */
    uint8_t sckPin;                     /* 用于串行时钟的 Cfgio 引脚 */
    uint8_t wsPin;                      /* 用于字选择的 Cfgio 引脚 */
    I2S_CALLBACK_T callback;            /* 用户回调函数 */
```

```
    void *callbackParam;                    /* 回调函数的参数 */
    uint8_t rxDMAChannel;                   /* Rx DMA 通道号。仅在 DMA 模式下使用 */
    uint8_t txDMAChannel;                   /* Tx DMA 通道号。仅在 DMA 模式下使用 */
} CFGIO_I2S_MASTER_CFG_T;
```

从设备用户结构体：

```
typedef struct
{
    CFGIO_DRIVER_TYPE_T driverType;     /* 驱动类型：中断/轮询/DMA */
    uint8_t bitsWidth;                  /* 一个字中的位数 - 8 的倍数 */
    uint8_t txPin;                      /* 用于传输的配置 IO 引脚 */
    uint8_t rxPin;                      /* 用于接收的配置 IO 引脚 */
    uint8_t sckPin;                     /* 用于串行时钟的 Cfgio 引脚 */
    uint8_t wsPin;                      /* 用于字选择的 Cfgio 引脚 */
    I2S_CALLBACK_T callback;            /* 用户回调函数 */
    void *callbackParam;                /* 回调函数的参数 */
    uint8_t rxDMAChannel;               /* Rx DMA 通道号 */
    uint8_t txDMAChannel;               /* Tx DMA 通道号。仅在 DMA 模式下使用 */
} CFGIO_I2S_SLAVE_CFG_T;
```

主设备初始化函数：

```
STATUS_T CFGIO_I2S_DRV_MasterInit(              /* 返回错误状态 */
uint32_t instance,                              /* CFGIO 的外设实例编号 */
const CFGIO_I2S_MASTER_CFG_T * userConfigPtr, /* 指向 CFGIO_I2S 主用户配置结构的指针 */
CFGIO_I2S_MASTER_STATE_T * master);             /* 用户定义的 CFGIO_I2S 主状态 */
```

从设备初始化函数：

```
STATUS_T CFGIO_I2S_DRV_SlaveInit(                    /* 返回错误状态 */
uint32_t instance,                                   /* CFGIO 的外设实例编号 */
const CFGIO_I2S_SLAVE_CFG_T * userConfigPtr, /* 指向 CFGIO_I2S 从用户配置结构的指针 */
cfgio_i2s_slave_state_t * slave);                    /* 用户定义的 CFGIO_I2S 从状态 */
```

主设备阻塞式发送：

```
STATUS_T CFGIO_I2S_DRV_MasterSendDataBlocking( /* 返回错误状态 */
CFGIO_I2S_MASTER_STATE_T * master,                /* 用户定义的 CFGIO_I2S 主状态 */
const uint8_t * txBuff,                          /* 用户发送的数据缓存 */
uint32_t txSize,                                 /* 待发送数据的字节长度 */
uint32_t timeout);                               /* 超时时间（以毫秒为单位） */
```

从设备阻塞式发送：

```
STATUS_T CFGIO_I2S_DRV_SlaveSendDataBlocking( /* 返回错误状态 */
cfgio_i2s_slave_state_t * slave,                 /* 用户定义的 CFGIO_I2S 从状态 */
const uint8_t * txBuff,                          /* 用户发送的数据缓存 */
uint32_t txSize,                                 /* 待发送数据的字节长度 */
uint32_t timeout);                               /* 超时时间（以毫秒为单位） */
```

主设备阻塞式接收：

```
STATUS_T CFGIO_I2S_DRV_MasterReceiveDataBlocking( /* 返回错误状态 */
CFGIO_I2S_MASTER_STATE_T * master,                /* 用户定义的 CFGIO_I2S 主状态 */
uint8_t * rxBuff,                                /* 用户接收的数据缓存 */
uint32_t rxSize,                                 /* 待接收数据的字节长度 */
uint32_t timeout);                               /* 超时时间（以毫秒为单位） */
```

从设备阻塞式接收：

```
STATUS_T CFGIO_I2S_DRV_SlaveReceiveDataBlocking( /* 返回错误状态 */
cfgio_i2s_slave_state_t * slave,                 /* 用户定义的 CFGIO_I2S 从状态 */
uint8_t * rxBuff,                                /* 用户接收的数据缓存 */
uint32_t rxSize,                                 /* 待发送数据的字节长度 */
uint32_t timeout);                               /* 超时时间（以毫秒为单位） */
```

6.3 测试用例

6.3.1 用例介绍

本用例使用 CFGIO 模块模拟 I2S 通信，在从配置中使用一个 CFGIO 实例，在主配置中使用另一个 CFGIO 实例，通过中断利用 I2S 总线进行数据通信。

6.3.2 硬件准备

- G32A1445 EVAL 评估板×1
- J-Link 烧录器（带 Type-B to A USB 线、FC-20P 双头线/2.54MM 间距）×1
- MINI USB 数据线×1
- 杜邦线若干
- 电脑×1

6.3.3 软件准备

- G32A1xxx_SDK_x.x.x
- Keil MDK 开发环境及 G32A 系列支持包
- SEGGER - J-Link Vx.xx
- 串口助手

6.3.4 测试过程

（1） 打开相应的 Keil MDK 工程，编译，下载到评估板：“G32A1xxx_SDK_x.x.x\Examples\G32A1445\CFGIO\CFGIO_I2S\Project\MDK\CFGIO_I2S.uvprojx”，直接编译，下载到 G32A1445 EVAL 评估板，并断开 J-Link 烧录器与评估板的连接。

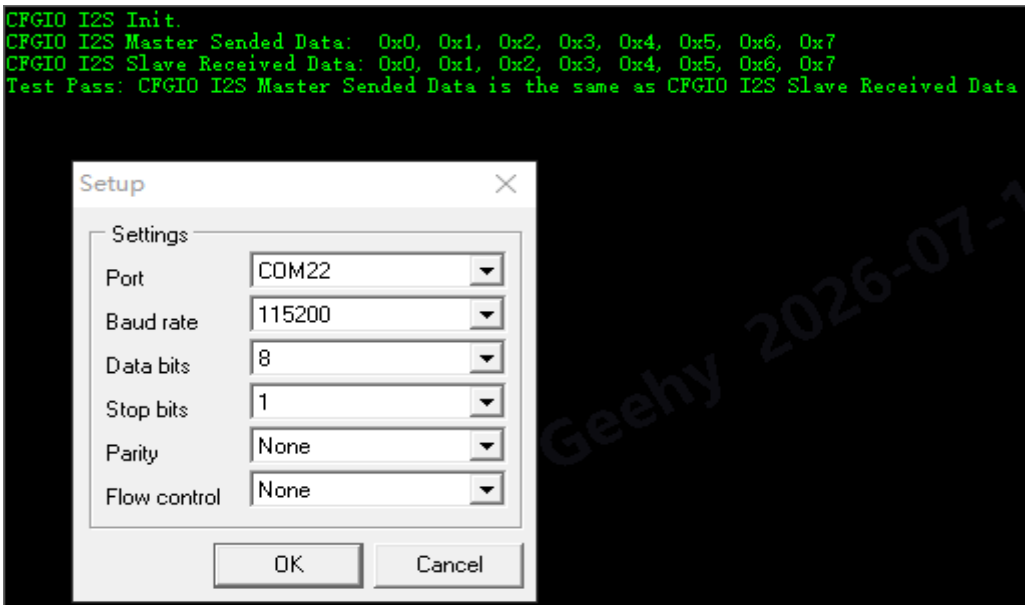
（2） 设备连接：评估板上引脚连接：

	master		slave	
sck	PD9	<==>	PA2	sck
ws	PA11	<==>	PA3	ws
tx	PA0	<==>	PA9	rx
rx	PA1	<==>	PA8	tx

（3） MINI USB 数据线一端连接评估板另一端连接电脑，电脑打开串口助手，配置串口参数后打开相应的端口。

（4） 按 RST 复位键，串口助手会收到评估板发来的打印信息，若 I2S 通讯成功则评估板的绿色 LED 灯亮起，否则红色 LED 亮起。

图 6 CFGIO_I2S.uvprojx 示例 I2S 主从设备收发



7 版本历史

表格 1 文件版本历史

日期	版本	变更历史
2026.07	1.0	新建

声明

本手册由珠海极海半导体有限公司（以下简称“极海”）制订并发布，所列内容均受商标、著作权、软件著作权相关法律法规保护，极海保留随时更正、修改本手册的权利。使用极海产品前请仔细阅读本手册，一旦使用产品则表明您（以下称“用户”）已知悉并接受本手册的所有内容。用户必须按照相关法律法规和本手册的要求使用极海产品。

1、权利所有

本手册仅应当被用于与极海所提供的对应型号的芯片产品、软件产品搭配使用，未经极海许可，任何单位或个人均不得以任何理由或方式对本手册的全部或部分内容进行复制、抄录、修改、编辑或传播。

本手册中所列带有“®”或“™”的“极海”或“Geehy”字样或图形均为极海的商标，其他在极海产品上显示的产品或服务名称均为其各自所有者的财产。

2、无知识产权许可

极海拥有本手册所涉及的全部权利、所有权及知识产权。

极海不应因销售、分发极海产品及本手册而被视为将任何知识产权的许可或权利明示或默示地授予用户。

如果本手册中涉及任何第三方的产品、服务或知识产权，不应被视为极海授权用户使用前述第三方产品、服务或知识产权，也不应被视为极海对第三方产品、服务或知识产权提供任何形式的保证，包括但不限于任何第三方知识产权的非侵权保证，除非极海在销售订单或销售合同中另有约定。

3、版本更新

用户在下单购买极海产品时可获取相应产品的最新版的手册。

如果本手册中所述的内容与极海产品不一致的，应以极海销售订单或销售合同中的约定为准。

4、信息可靠性

本手册相关数据经极海实验室或合作的第三方测试机构批量测试获得，但本手册相关数据难免会出现校正笔误或因测试环境差异所导致的误差，因此用户应当理解，极海对本手册中可能出现的该等错误无需承担任何责任。本手册相关数据仅用于指导用户作为性能参数参照，不构成极海对任何产品性能方面的保证。

用户应根据自身需求选择合适的极海产品，并对极海产品的应用适用性进行有效验证和测试，以确认极海产品满足用户自身的需求、相应标准、安全或其它可靠性要求；若因用户未充分对极海产品进行有效验证和测试而致使用户损失的，极海不承担任何责任。

5、合规要求

用户在使用本手册及所搭配的极海产品时，应遵守当地所适用的所有法律法规。用户应了解产品可能受到产品供应商、极海、极海经销商及用户所在地等各国有关出口、再出口或其它法律的限制，用户（代表其本身、子公司及关联企业）应同意并保证遵守所有关于取得极海产品及/或技术与直接产品的出口和再出口适用法律与法规。

6、免责声明

本手册由极海“按原样”（as is）提供，在适用法律所允许的范围内，极海不提供任何形式的明示或暗示担保，包括但不限于对产品适销性和特定用途适用性的担保。

极海产品并非设计、授权或担保适合用于军事、生命保障系统、污染控制或有害物质管理系统中的关键部件，亦非设计、授权或担保适合用于在产品失效或故障时可导致人员受伤、死亡、财产或环境损害的应用。

如果产品未标明“汽车级”，则表示不适用于汽车应用。如果用户对产品的应用超出极海提供的规格、应用领域、规范，极海不承担任何责任。

用户应该确保对产品的应用符合相应标准以及功能安全、信息安全、环境标准等要求。用户对极海产品的选择和使用负全部的责任。对于用户后续在针对极海产品进行设计、使用的过程中所引起的任何纠纷，极海概不承担责任。

7、责任限制

在任何情况下，除非适用法律要求或书面同意，否则极海和/或以“按原样”形式提供本手册

及产品的任何第三方均不承担损害赔偿责任，包括任何一般、特殊因使用或无法使用本手册及产品而产生的直接、间接或附带损害（包括但不限于数据丢失或数据不准确，或用户或第三方遭受的损失），这涵盖了可能导致的人身安全、财产或环境损害等情况，对于这些损害极海概不承担责任。

8、适用范围

本手册的信息用以取代本手册所有早期版本所提供的信息。