恩智浦半导体
应用笔记

文档编号：AN12174
第0版，2018年6月

使用 FlexIO 模拟通信和定时外设

1. 简介

FlexIO 是用在 Kinetis 和 S32K 微控制器系列上的新外设模块。它具有高度可配置性，能够模拟各种通信协议，例如本文档中提到的UART、I2C、SPI、I2S和LIN，以及其它一些通讯协议，像J1850、I3C、曼彻斯特。

FlexIO 作为微控制器的一个独立外设模块，并不能替代任何通信外设。FlexIO 的主要特点是可以根据用户的需求来直接构建自己的外设。

本文的示例代码基于 S32K SDK（包含在 S32DS_v2018 中的软件开发套件中）和 Bare Metal 的配置代码，以便于更好地了解什么是 FlexIO。通过这些示例，用户可以模拟不同的通信协议和 PWM 信号。

目录

2. FlexIO 模块概述

FlexIO 模块具有以下主要硬件资源：

- 移位器
- 定时器
- 引脚

可以从 FLEXIO_PARAM 寄存器中获取特定MCU 里面这些资源数量。例如，S32K1xx 系列中有 4 个移位器、4 个定时器和 8 个引脚。

表1. 用于通信协议的资源

用例	支持使用	FlexIO 使用	注释
UART 应用	**一路发送部分和一路接收部分**： **发送：**一个定时器、一个移位器、一个引脚 **接收：**一个定时器、一个移位器、一个引脚	50%	允许轮询和中断模式 可配置的位顺序 使用 DMA 控制器可以支持多次传输 不支持自动插入奇偶校验位
SPI主设备	两个定时器、两个移位器、四个引脚	50%	支持 CPHA=0 和 CPHA=1
SPI从设备	一个定时器、两个移位器、四个引脚	33%	支持 CPHA=0 and CPHA=1
I2C主设备	一个计时器，两个移位器，两个引脚	50%	FlexIO 在每个字节后插入一个停止位以生成或验证 ACK/NACK
I2S主设备	两个定时器、两个移位器、四个引脚	50%	可以使用 DMA 支持数据传输

下图显示了 FlexIO 模块的概览。

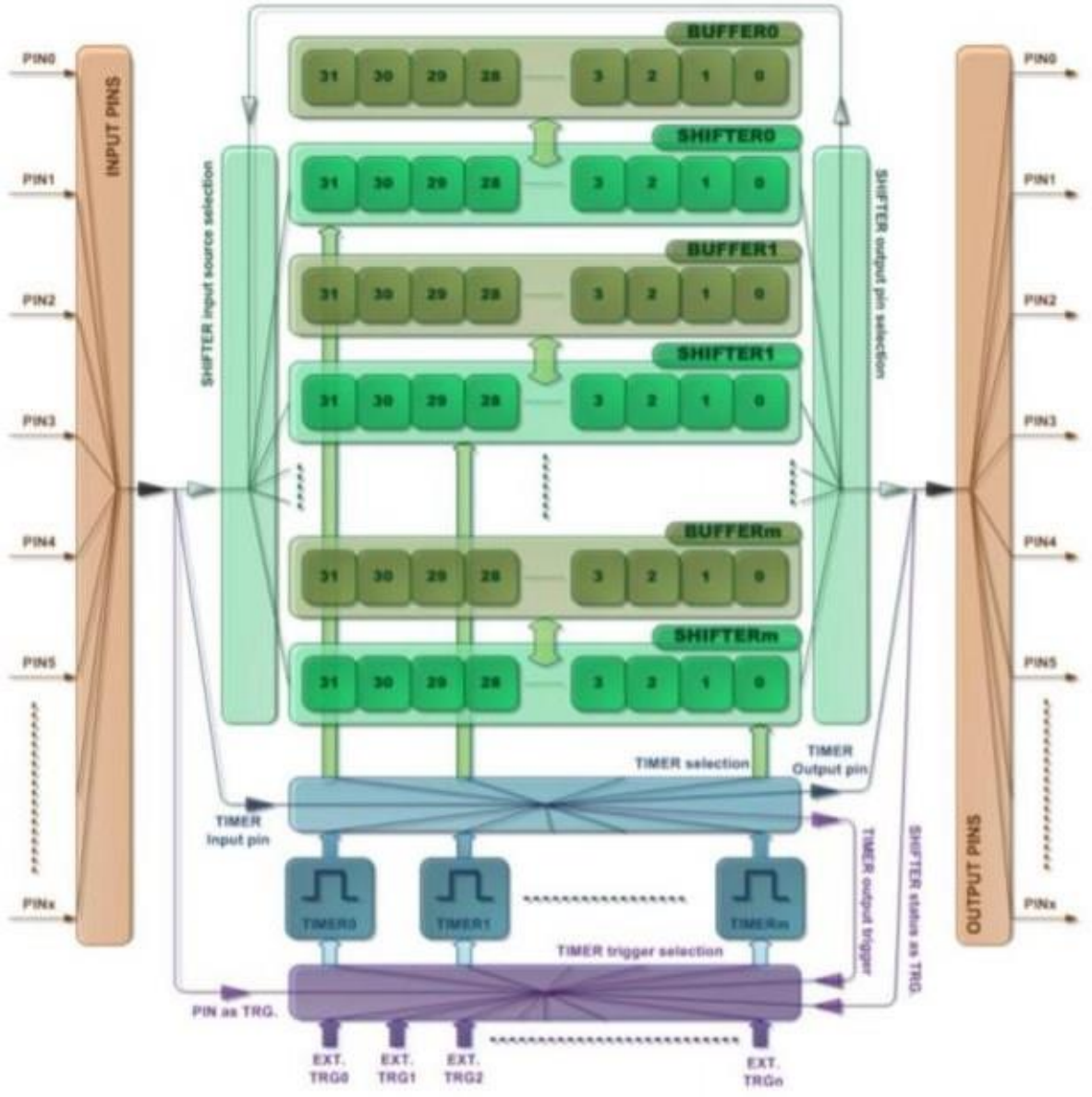


图1. FlexIO 框图

提供以下主要功能：

- 具有发送、接收和数据匹配模式的 32 位移位器
- 双缓冲移位器操作
- 具有高度灵活性的 16 位定时器，支持各种内部或外部触发，以及复位/启用/禁用/递减条件
- 自动生成或检查开始/停止位

- 中断、DMA 或轮询模式操作
- 移位器、定时器、引脚和触发器可以灵活组合操作

发送和接收是移位器的两种基本模式。如果一个移位器配置为发送模式，它会从其缓冲寄存器加载数据，并将数据逐位移出到其指定的引脚。如果一个移位器配置为接收模式，它会将数据从其分配的引脚移入并将数据存储在其缓冲寄存器中。加载、存储和移位操作都由移位器指定的计时器控制。

定时器还可以根据您的要求配置为不同的操作模式，包括双 8 位计数器波特率/位模式、双 8 位计数器 PWM 模式和单 16 位计数器模式。

3. 使用 FlexIO 模拟 UART

3.1. 指导

有关 FlexIO 的详细说明，请参阅微控制器参考手册。本示例创建了一个基于 S32K SDK 的简单软件样例，以及裸机配置示例代码驱动程序，供用户使用来模拟 UART。

本节介绍如何使用 FlexIO 模拟 UART。对于此应用，使用了图 2 中的评估开发板 S32K144EVB-Q100。

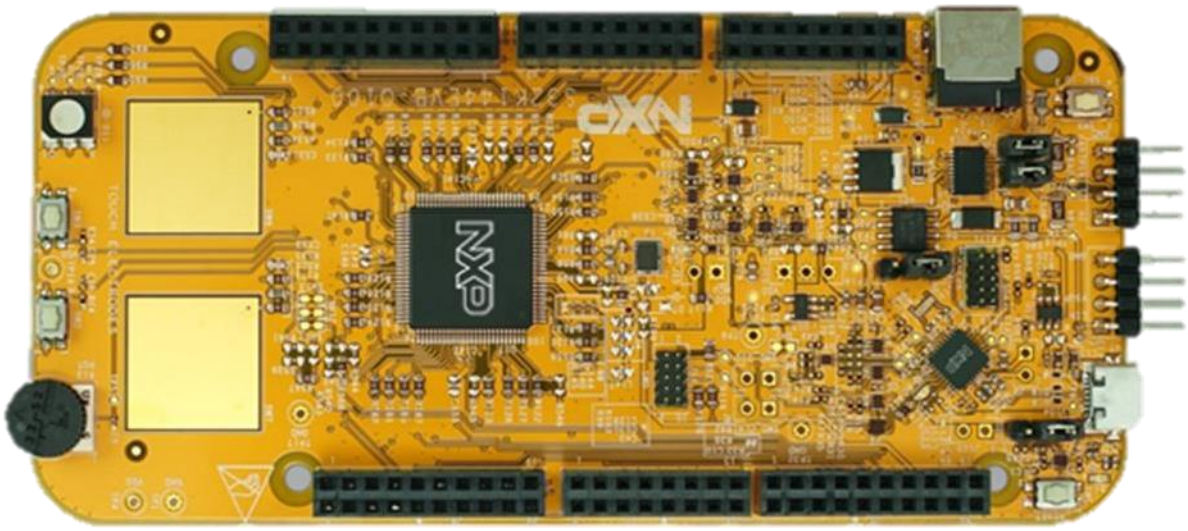


图2. 开发板 S32K144EVB-Q100

在本应用中，FlexIO D0 引脚配置为 UART TXD，FlexIO D1 引脚配置为 UART RXD。使用 1 条外部连接线连接 TXD 和 RXD。您可以使用通用串行终端/控制台来验证数据传输的结果。

下图显示了硬件平台和数据传输：

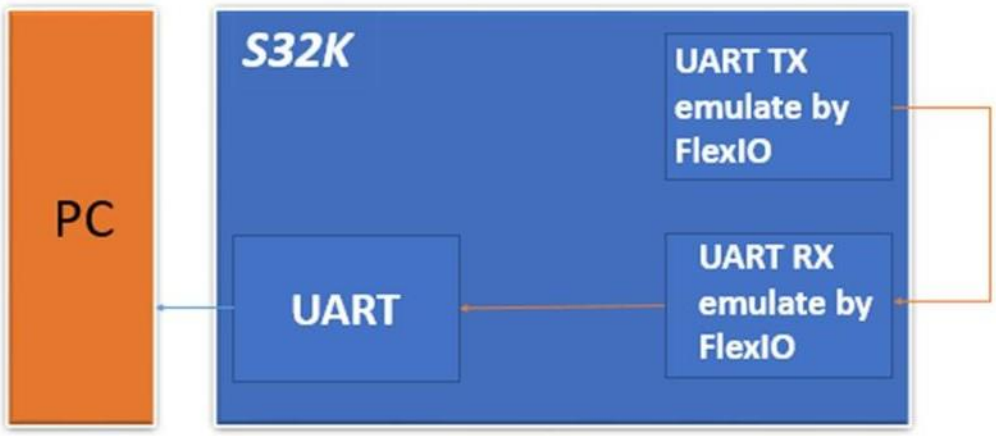


图3. 此应用程序的硬件平台和数据流

UART 发送使用以下资源：

- 1 个定时器 — 配置为 8 位计数器模式以控制数据移位。
- 1 个引脚 — 由定时器控制以从 SHIFTBUF 输出数据。
- 1 个移位器 — 由定时器控制从 SHIFTBUF 或配置的起始位和停止位移位数据。

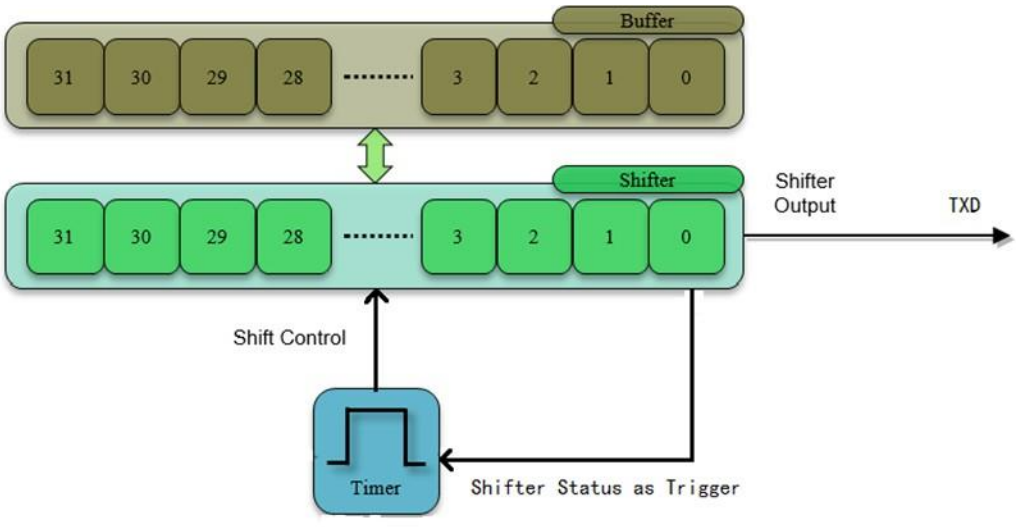


图4. FlexIO 的资源分配以模拟 UART 发送

UART 接收使用以下资源：

- 1 个定时器 — 配置为 8 位计数器模式以控制数据移位。
- 1 个引脚 — 由定时器控制以将数据输入到 SHIFTBUF。

- 1 个移位器 — 由定时器控制将数据移入 SHIFTBUF 并配置输入数据是否有起始位和停止位。

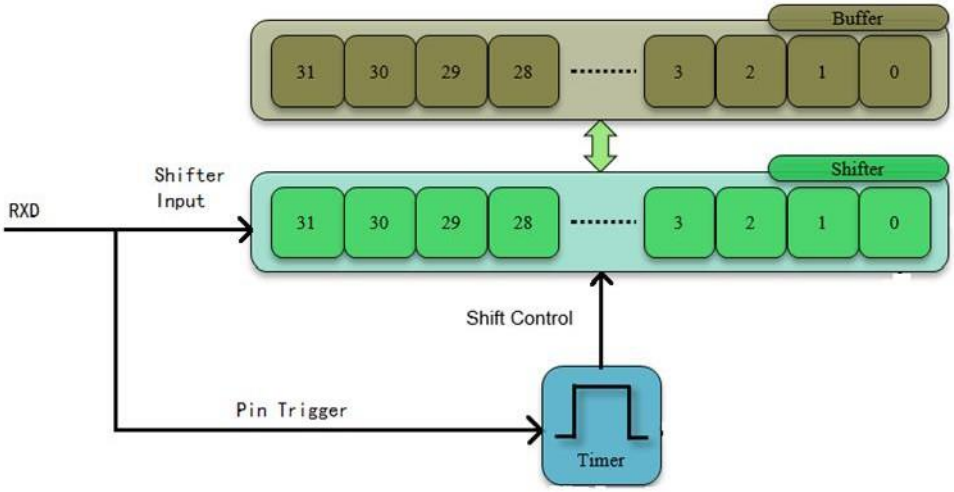


图5. FlexIO 的资源分配以模拟 UART 接收

以下部分提供了详细的配置和使用信息。

3.2. 配置移位器和定时器

本节提供移位器和定时器的详细配置。注意本节的参数设置基于：UART波特率=115200，UART 位数 = 8 位，1个起始位，1个停止位，无奇偶校验位。其中一些设置必须由软件更改以支持不同的 UART 功能。要了解这些配置，请参阅以下部分和 S32K 参考手册。

3.2.1. UART 发送配置

移位器 0 的配置

移位器 0 作用于 UART的TXD，即 FlexIO_D0 脚。它具有以下初始配置。

表2. 移位器 0/1 的配置

项目	配置	
移位模式	发送	
定时器选择	timer 0	
定时器极性	上升沿移位	
引脚选择	FlexIO D0	
引脚配置	电平输出	
引脚电平	高电平有效	

输入源	从引脚输入	
起始位	起始位 0	
停止位	停止位 1	
缓冲区使用	由 SHIFTBUF[7:0] 启动 8 位数据传输	

定时器 0 的配置

定时器 0 被 UART 用来对移位器 0 的控制。移位器状态标志被置位意味着 SHIFTBUF 已加载来自移位器的数据，每次读取 SHIFTBUF 寄存器数据后状态标记位自动被清零，也意味着 SHIFTBUF 中的数据已传输到移位器（SHIFTBUF 为空）。移位器状态标志 0 被配置为定时器 0 的触发器，因此只要写入SHIFTBUT，状态标志就会被清除并启用定时器 0。移位器开始在时钟的下降沿移出数据，直到定时器向下计数到 0 自动停用。定时器 0 具有以下初始配置。

表3. timer 0 的配置

项目	配置
定时器模式	双 8 位计数器波特/位模式
触发选择	移位器 0 状态标志
触发电平	低电平有效
触发源	内部触发
引脚配置	输出禁用
定时器初始输出	使能时输出逻辑 1，不受复位影响
定时器递减源	按 FlexIO 时钟递减，并移位输出
定时器使能条件	高电平触发
定时器禁用条件	定时器比较
定时器复位条件	定时器不复位
起始位	启用
停止位	定时器自动停用后启用
定时器比较值	((n*2-1)<<8) | (baudrate_divider/2-1)) [1]

1) n 是传输中的字节数。Baudrate_divider 是用于对来自 FlexIO 时钟源的波特率进行分频的值。

3.2.2. UART 接受配置

移位器 1 的配置

[1] n 是传输中的字节数。Baudrate_divider 是用于对来自 FlexIO 时钟源的波特率进行分频的值。

移位器 1 作用于UART的RXD，即引脚 FlexIO_D1。它具有以下初始配置。

表4. 移位器 1 的配置

项目	配置
移位器模式	接收模式
定时器选择	定时器 1
定时器极性	下降沿移位
引脚选择	FlexIO D1
引脚配置	无输出引脚
引脚电平	高电平有效
输入源	引脚输入
起始位	起始位 0
停止位	停止位 1
使用缓冲区	SHIFTBUF[31:24] 接收数据

定时器 1 的配置

定时器 1 被 UART 用来作为对移位器 1 的控制。pin1 上升沿时配置使能定时器 1。移位器在时钟的下降沿开始移入数据，直到定时器向下技术到 0 时自动被停用。定时器 1 具有以下初始配置。

表5. 定时器 1 的配置

项目	配置
定时器模式	双 8 位计数器波特/位模式
触发选择	从引脚 1 触发
触发电平	高电平有效
触发源	外部触发
引脚配置	无输出引脚
定时器初始输出	使能时输出逻辑 1，不受复位影响
定时器递减源	按 FlexIO 时钟递减，并移位输出
定时器使能条件	引脚处于上升沿
定时器禁用条件	定时器比较
定时器复位条件	计时器不复位
起始位	启用
停止位	定时器自动停用后启用
定时器比较值	((n*2-1)<<8) | (baudrate_divider/2-1)) [2]

2) n 是传输中的字节数。Baudrate_divider 是用于对来自 FlexIO 时钟源的波特率进行分频的值。

软件实现概述

本节描述了基于SDK（包含在 S32DS_v2018 中的软件开发套件中）的软件实现，请注意，裸机示例代码（遵循本文档表格中的相同配置）也包含在该应用笔记附带的软件包中，所有函数都可以由用户在自己的代码中直接使用，只需稍作改动。

实现的功能特性

- 中断、DMA或轮询模式
- 提供阻塞和非阻塞发送和接收功能
- 可配置的波特率和位数
- 单个停止位
- 无奇偶校验位

初始化

在使用任何 FlexIO 驱动程序之前，必须首先使用函数 **FLEXIO_DRV_InitDevice** 初始化设备。然后再使用函数 **FLEXIO_UART_DRV_Init** 初始化 FLEXIO_UART。如果有足够的资源可用，可以在同一个 FlexIO 设备上实现更多的驱动程序实例。同一 FlexIO 设备上的不同驱动程序实例可以相互独立运行。使用结束后，可以使用 **FLEXIO_UART_DRV_Deinit()** 反初始化，释放使用的硬件资源。

3.3. 函数说明

FLEXIO_DRV_InitDevice 函数打开 FlexIO IP 模块的时钟，并为 FlexIO 选择合适的外设时钟源。源文件中定义的 FLEXIO_CLK 是外设时钟源的频率。这是一个通用的 FlexIO IP 模块初始化函数，该函数会对 FLEXIO IP模块进行复位重置，在使用移位器和定时器之前需要调用。

FLEXIO_UART_DRV_Init 函数将定时器0 配置为双 8 位计数器波特率/位模式，以通过 引脚 0 移位输出数据来模拟 UART TXD，并将定时器 1 配置为双 8 位计数器波特率/位模式以将数据从FlexIO D1 脚移入输入来模拟UART RXD。‘baud’ 参数表示 **UART** 的波特率。**‘bits’ 参数表示一个 UART 帧的位数。**

3.3.1. FLEXIO_UART_DRV_SendData

该函数用于通过 FlexIO UART 发送数据。

3.3.2. FLEXIO_UART_DRV_ReceiveDataBlocking

该函数用于通过 FlexIO UART 接收数据。

可以参考在 main() 函数中调用顺序来使用FlexIO函数进行UART发送和接收数据。样例程序中，波特率配置为 115200bps，PTD0 为 TXD，PTD1 为 RXD。

3.4. 运行例程

该样例工程在 S32K144EVB-Q100 上运行。TXD 和 RXD 的 FlexIO 引脚分配如下表所示：

FlexIO UART TXD	
FlexIO UART TXD Pin:FlexIO_D0	PTD0
FlexIO UART RXD	
FlexIO UART RXD Pin:FlexIO_D1	PTD1

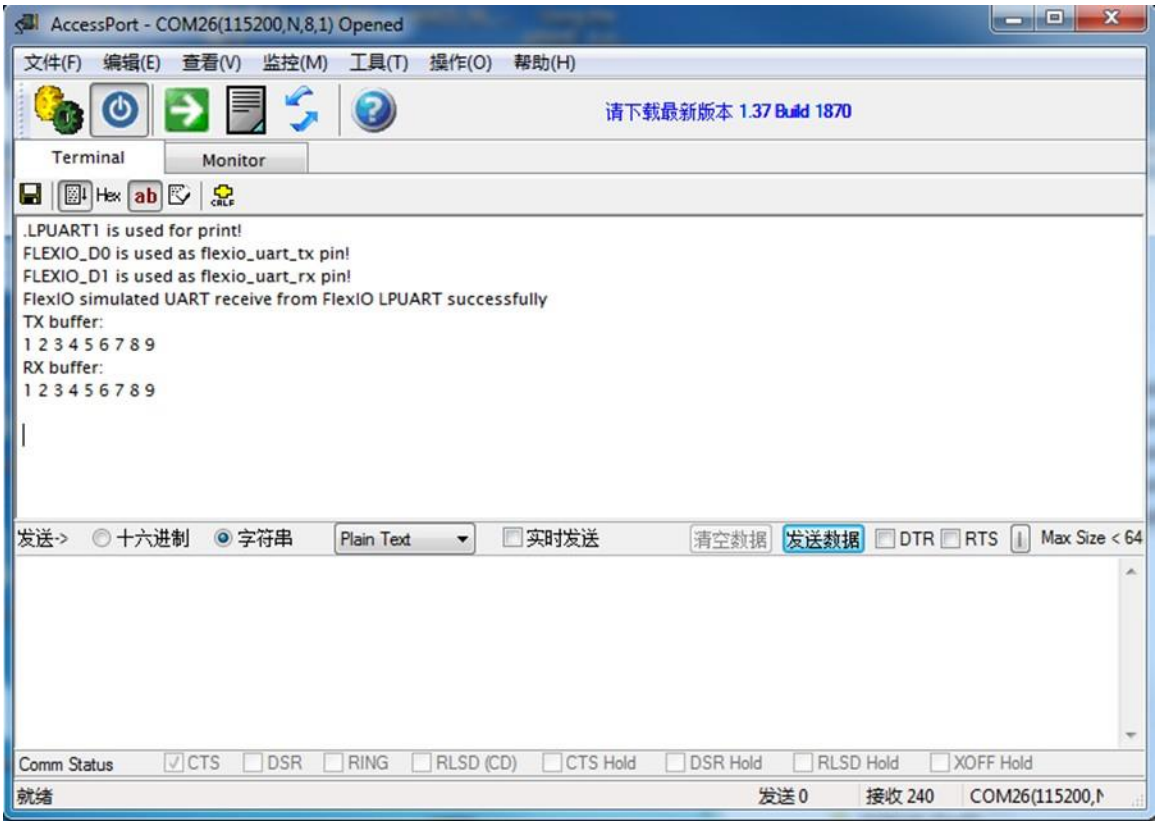
表6. UART TXD 和 RXD 的 FlexIO 引脚分配表

在通过 J-link 或 OpenSDA 将程序下载到 MCU 之前，您需要使用 1 条外部线将TXD 和 RXD 连接起来：

- UART TXD <--- > UART RXD.

然后按照以下步骤运行并检查结果：

- 插入 Micro USB 以连接 PC 和 S32K144EVB-Q100 目标板
- 在 PC 上打开 UART 调试终端，波特率设置为 115200bps
- 在 S32DS 工作区中打开项目
- 编译样例工程并下载到目标板运行
- 数据传输完成后，结果打印在UART调试终端上



4. 使用 FlexIO 模拟双 SPI

本节介绍如何使用 FlexIO 模拟双 SPI。此应用使用了图 2 中所示的S32K144EVB-Q100 硬件板。

在本应用中，FlexIO D0~D3 引脚配置为 SPI 主设备，FlexIO D4~D7 引脚配置为 SPI 从设备。使用 4 条外部线连接主设备和从设备。您可以使用UART调试终端检查数据环回传输的结果。下图显示了硬件平台和数据流：

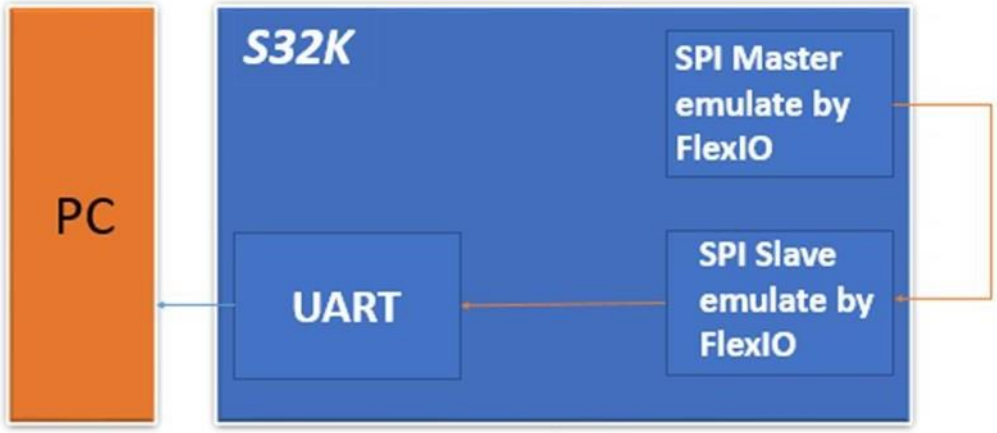


图6. 硬件平台和数据流

使用以下方法模拟 SPI 主设备：

- 2 个移位器：一个移位器用作数据发送器，另一个移位器用作数据接收器。
- 2 个定时器：一个定时器用于SPI_CS 输出控制，另一个定时器用于两个移位器的加载/存储/移位控制和SPI_SCK 生成。
- 4 个引脚：用作 SPI_CS、SPI_SCK、SPI_SOUT 和 SPI_SIN。

下图显示了资源分配。

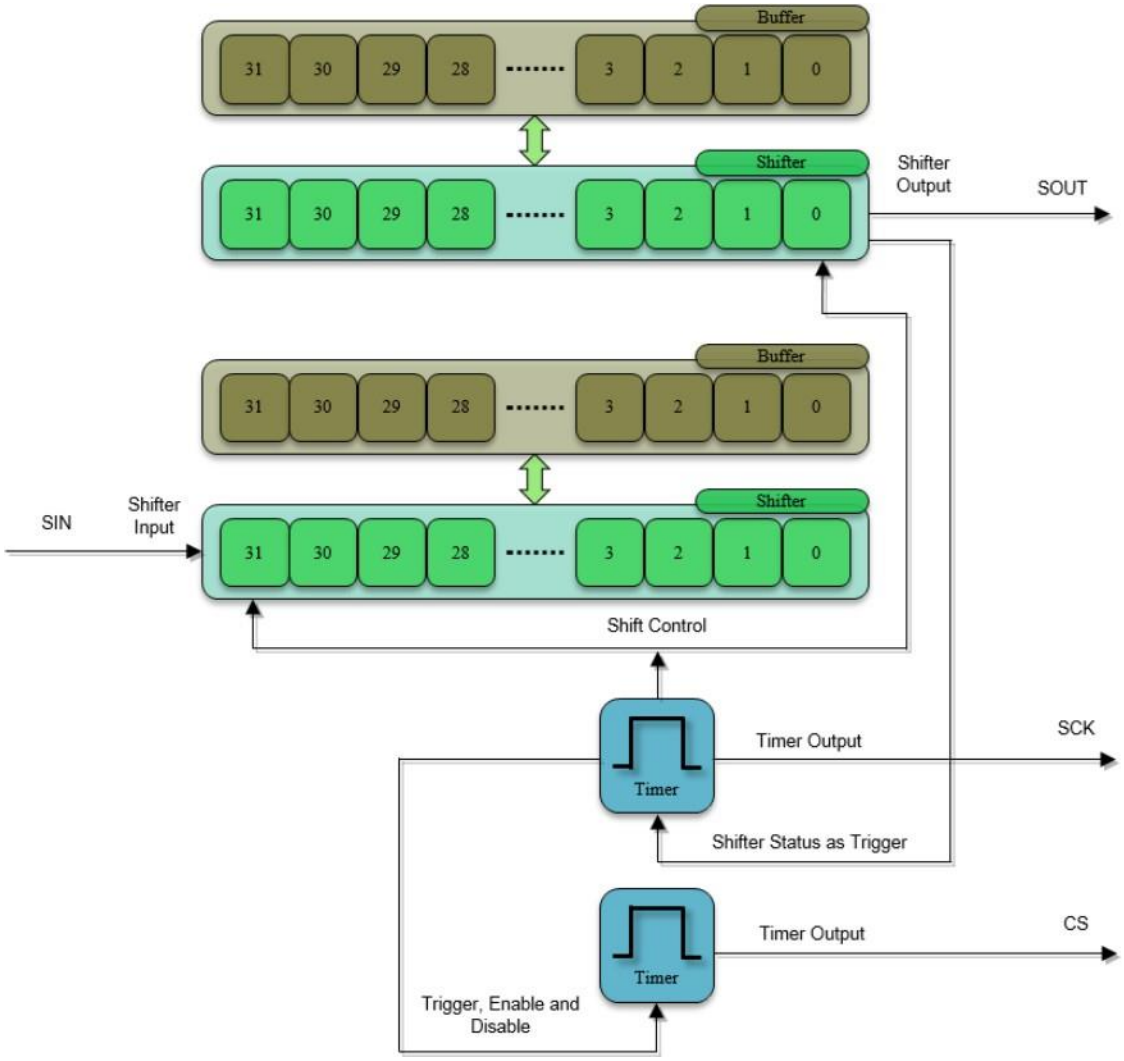


图7. FlexIO 的资源分配以模拟 SPI Master

使用以下方法模拟 SPI从设备：

- 2 个移位器：一个移位器用作数据发送器，另一个移位器用作数据接收器。
- 1 个定时器：用于两个移位器的加载/存储/移位控制。

- 4 引脚：用作 SPI_CS、SPI_SCK、SPI_SOUT 和 SPI_SIN。

下图显示了从设备资源分配。

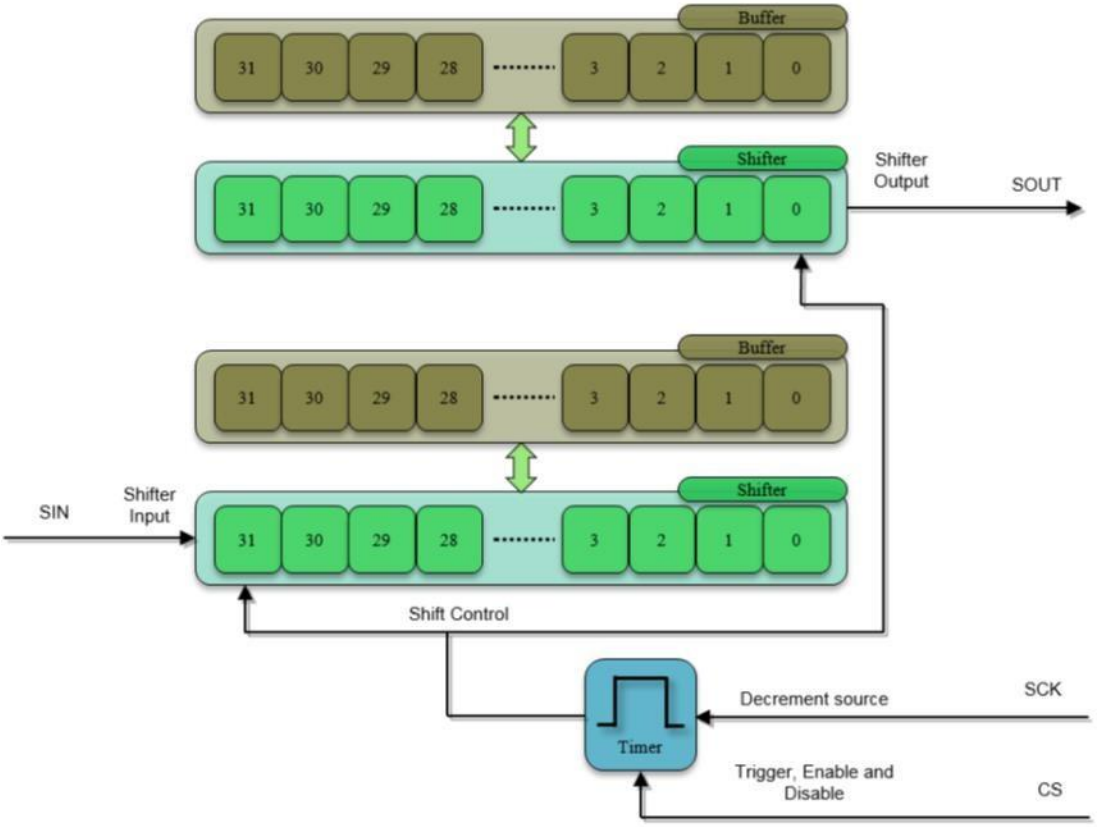


图8. FlexIO 的资源分配以模拟 SPI 从设备

以下部分提供了详细的配置和使用信息。

4.1. 移位器和定时器的配置

本节提供移位器和定时器的详细配置。请注意，本节列出的项目是初始设置，CPHA= 0，SPI 波特率= 2MHz，SPI 位计数= 8 位。其中一些设置必须由软件更改以支持不同的 SPI 功能。要了解这些配置，请参阅以下部分和 S32K 参考手册。

4.1.1. SPI 主设备配置

移位器 0 的配置

移位器 0 作用于 SPI 主设备的 SPI_SOUT，也即 FlexIO_D0 脚。它具有以下初始配置。

表7. 移位器 0 配置

项目	配置
移位器模式	发送模式
定时器选择	定时器 0
定时器极性	下降沿移位
引脚选择	引脚 0
引脚配置	输出
引脚电平	高电平有效
输入源	从引脚输入
起始位	禁用，使能时加载数据
停止位	禁用
使用缓冲区	位字节交换寄存器

移位器 1 的配置

移位器 1 作用于 SPI 主设备的 SPI_SIN，也即 FlexIO_D1 脚。它具有以下初始配置。

表8. 移位器 1 配置

项目	配置
移位器模式	接收模式
定时器选择	定时器 0
定时器极性	上升沿移位
引脚选择	FlexIO D1
引脚配置	禁用输出
引脚电平	高电平有效
输入源	从引脚输入
起始位	禁用，使能时加载数据
停止位	禁用
使用缓冲区	位字节交换寄存器

定时器 0 的配置

SPI 主设备使用定时器 0 在引脚 FlexIO_D2 上输出 SPI_SCK 以及控制两个移位器的加载、存储和移位。每次对 SHIFTBUF 寄存器进行写入和读取操作时，移位器状态标志都会置位和清零，意味着 SHIFTBUF 中的数据已经传输到移位器中（SHIFTBUF为空）。移位器状态标志 0 配置为定时器 0 的触发器，因此只要写入 SHIFTBUT，状态标志就会被清除并启用定时器 0。移位器开始在时钟的下降沿移出数据，直到定时器向下计数到 0 被停用。定时器 0 具有以下初始配置。

表9. 定时器 0 配置

项目	配置
定时器模式	双 8 位计数器波特/位模式
触发选择	移位器状态标志 0
触发电平	低电平有效
触发源	内部触发

项目	配置
引脚选择	FlexIO D2
引脚配置	输出使能
引脚电平	高电平有效
定时器初始输出	使能时输出逻辑 0，不受复位影响
定时器递减源	按 FlexIO 时钟递减，并移位输出
定时器使能条件	高电平触发
定时器禁用条件	定时器比较
定时器复位条件	计时器不复位
起始位	启用
停止位	定时器停用时启用
定时器比较值	((n*2-1)<<8) | (baudrate_divider/2-1)) [3]

定时器 1 的配置

SPI 主设备使用定时器 1 在引脚 FlexIO_D3 上输出 SPI_CS。定时器 1 配置为在定时器 0 启用时启用。比较寄存器配置为 16 位计数器并设置为 0xFFFF。使用该值，定时器不进行比较，并且在启用后始终处于活动状态。定时器 1 具有以下初始配置。

表10. 定时器 1 配置

项目	配置
定时器模式	单 16 位计数器模式
触发选择	从定时器 0 触发
触发电平	高电平有效
触发源	内部触发
引脚选择	FlexIO D3
引脚配置	输出使能
引脚电平	低电平使能
定时器初始输出	使能时输出逻辑 1，不受复位影响
定时器递减源	按 FlexIO 时钟递减计数器，并移位输出
定时器使能条件	定时器 0 启用时
定时器禁用条件	定时器 0 禁用时
定时器复位条件	定时器不重置
起始位	禁用
停止位	禁用
定时器比较值	0xFFFF

4.1.2. SPI 从机配置

移位器 2 的配置

[3] n 是传输中的字节数。Baudrate_divider 是用于对来自 FlexIO 时钟源的波特率进行分频的值。

移位器 2 作用于SPI从设备的 SPI_SOUT，也即引脚 FlexIO_D4。它具有以下初始配置。

表11. 移位器 2 配置

项目	配置
移位器模式	发送模式
定时器选择	定时器 2
定时器极性	下降沿移位
引脚选择	FlexIO D4
引脚配置	输出使能
引脚电平	高电平有效
输入源	从引脚输入
起始位	禁用，发射器在启用时加载数据
停止位	禁用
使用缓冲区	位字节交换寄存器

移位器 3 的配置

移位器 3 由引脚 FlexIO_D5 上的 SPI 从接收器用作 SPI_SIN。它具有以下初始配置。

表12. 移位器 3 配置

项目	配置
移位器模式	接收模式
定时器选择	定时器 2
定时器极性	换挡时钟正极
引脚选择	FlexIO D5
引脚配置	输出禁止
引脚电平	高电平有效
输入源	从引脚输入
起始位	禁用，使能时加载数据
停止位	禁用
使用缓冲区	位字节交换寄存器

定时器 2 的配置

SPI 从设备使用定时器 2 从主设备获取引脚 FlexIO_D6 上的 SPI_SCK 以加载/存储/移位控制两个移位器。在从机模式下，SPI_SCK 和 SPI_CS 信号被配置为输入并由 SPI 主设备驱动。当 SPI_CS 信号被置位时，发送数据在每个帧的每个 SPI_SCK 时钟沿传输到移位寄存器。因此，选择 SPI_CS 的引脚 FlexIO_D7 作为定时器 2 的触发输入。它具有以下初始配置。

表13. 定时器 2 配置

项目	配置
定时器模式	单 16 位计数器模式
触发选择	从 SPI_CS 的 FlexIO ID7 触发

项目	配置
触发电平	低电平有效
触发源	内部触发
引脚选择	FlexIO D6
引脚配置	输出使能
引脚电平	高电平有效
定时器初始输出	使能时输出逻辑 0，不受复位影响
定时器递减源	引脚输入递减，移位时钟等于引脚输入
定时器使能条件	上升沿触发
定时器禁用条件	定时器不被禁用
定时器复位条件	定时器不复位
起始位	禁用
停止位	禁用
定时器比较值	(n*2-1) [4]

4.2. 软件概述

通过使用FLEXIO_SPI驱动程序，可以在S32K1系列MCU上通过FlexIO模块来实现SPI总线通讯。

特征：

- 主或从操作
- 中断、DMA或轮询模式
- 提供阻塞和非阻塞传输功能
- 可配置的波特率
- 可配置的时钟极性和相位
- 可配置的位顺序和数据大小
- 通讯功能

初始化

在使用任何 FlexIO 驱动程序之前，必须首先使用函数 FLEXIO_DRV_InitDevice 初始化设备。然后再使用函数 FLEXIO_SPI_DRV_MasterInit() 或 FLEXIO_SPI_DRV_SlaveInit() 初始化 FLEXIO_SPI 驱动程序。如果有足够的资源可用，可以在同一个 FlexIO 设备上使用更多的驱动程序实例。同一 FlexIO 设备上的不同驱动程序实例可以相互独立运行。当不再需要时，可以使用 FLEXIO_SPI_DRV_MasterDeinit() 或 FLEXIO_SPI_DRV_SlaveDeinit() 释放硬件资源。

[4] n 是传输中的字节数。

主模式

主模式提供向 SPI 从设备发送数据或从 SPI 从设备接收数据的功能。波特率在初始化时设置，也可以在运行时使用 FLEXIO_SPI_DRV_MasterSetBaudRate() 函数更改。请注意，由于模块限制，无法实现所有波特率。驱动程序会将波特率设置为尽可能接近所请求的波特率，但可能存在较大差异，特别是在使用低频 FlexIO 时钟而需要高波特率。应用程序应在调用 FLEXIO_SPI_DRV_MasterSetBaudRate() 之后调用FLEXIO_SPI_DRV_MasterGetBaudRate() 以检查实际的波特率。

使用函数 FLEXIO_SPI_DRV_MasterTransfer() 来发送或接收数据。发送和接收缓冲区以及其它传输参数通过 flexio_spi_transfer_t 结构体提供。如果只需要发送或只需要接收，可以把接收或发送缓冲区设置为 NULL。此驱动程序不支持使用用户在回调函数中实现连续发送或接收。回调函数仅用于表示通知传输结束。

阻塞操作只会在传输完成后返回，无论是成功还是失败。非阻塞操作会发起传输并立即返回 STATUS_SUCCESS，但模块仍然在传输中，直到当前传输完成才能发起另一次传输。当传输完成时，应用程序将通过回调函数得到通知。或者也可以通过调用 FLEXIO_SPI_DRV_MasterGetStatus() 来检查当前传输的状态。如果传输仍在进行，此函数将返回 STATUS_BUSY。如果传输完成，该函数将返回 STATUS_SUCCESS 或错误代码，具体取决于传输的结果。

驱动程序支持中断、DMA 和轮询模式。在轮询模式下，通过函数 FLEXIO_SPI_DRV_MasterGetStatus() 检查和处理 FlexIO 模块报告的事件来确保传输的进度。应用程序需要频繁的调用该函数（每个传输字节至少一次）以避免 Tx 下溢或 Rx 上溢。在DMA 模式下，驱动程序将使用的 DMA 通道来传输数据。需要在 FLEXIO_SPI 驱动程序初始化之前初始化 DMA 通道。FLEXIO_SPI 驱动程序会设置相应的DMA 请求源。

从模式

从模式与主模式类似，主要区别在于通过 FLEXIO_SPI_DRV_SlaveInit() 函数初始化 FlexIO 模块以使用从主机接收的时钟信号，而不是产生时钟。因此，在从模式下没有 SetBaudRate 功能。除此之外，从模式提供与主模式类似的接口。

FLEXIO_SPI_DRV_SlaveTransfer() 可用于发起传输，FLEXIO_SPI_DRV_SlaveGetStatus() 用于在轮询模式下检查传输状态。主模式描述中的所有其他操作也适用于从模式。

4.3. 操作实现

此应用例程基于 S32K SDK（包含在 S32DS_v2018 中的软件开发套件版本）实现，例程包括两个源文件：main.c 和 retarget.c，提供配置FlexIO模拟双SPI、DMA配置和传输后的数据验证等功能。用户只需稍作改动即可直接在自己的代码中使用。该例程采用 DMA 模式，与轮询模式和中断模式相比，它具有更好的性能。

FlexIO SPI 初始化函数

在使用任何 FlexIO 驱动程序之前，必须首先使用函数 FLEXIO_DRV_InitDevice 初始化设备。

然后再使用函数 FLEXIO_SPI_DRV_MasterInit() 或 FLEXIO_SPI_DRV_SlaveInit() 初始化 FLEXIO_SPI 驱动程序。如果有足够的资源可用，可以在同一个 FlexIO 设备上使用多个的驱动程序实例。同一 FlexIO 设备上的不同驱动程序实例可以相互独立运行。使用结束后，通过 FLEXIO_SPI_DRV_MasterDeinit() 或 FLEXIO_SPI_DRV_SlaveDeinit() 释放硬件资源。

DMA 配置函数

该例程通过 DMA 方式实现 FlexIO 模拟双 SPI 环回传输。以下原型用于配置 DMA 和 DMAMUX。

```
DMA_Init();
ConfigDMAfor_SPI_MASTER_TX();
ConfigDMAfor_SPI_MASTER_RX();
ConfigDMAfor_SPI_SLAVE_TX();
ConfigDMAfor_SPI_SLAVE_RX();
```

发送/接收函数

- **FLEXIO_SPI_DRV_MasterTransfer**

4.4. 运行例程

该例程在 S32K144EVB-Q100 板上运行。SPI 主设备和 从设备的 FlexIO 引脚分配如下表所示：

表14. SPI 主机和从机的 FlexIO 引脚分配表

FlexIO SPI 主机	
FlexIO SPI master TX Pin:FlexIO_D0	PTD0
FlexIO SPI master RX Pin:FlexIO_D1	PTD1
FlexIO SPI master SCK Pin:FlexIO_D2	PTE15
FlexIO SPI master CS Pin:FlexIO_D3	PTE16
FlexIO SPI 从机	
FlexIO SPI slave TX Pin:FlexIO_D4	PTE10
FlexIO SPI slave RX Pin:FlexIO_D5	PTE11
FlexIO SPI slave SCK Pin:FlexIO_D6	PTA8
FlexIO SPI slave CS Pin:FlexIO_D7	PTA9

在通过 J-link 或 OpenSDA 将程序下载到 MCU 之前，需要使用 4 条外部线连接主从设备：

- SPI master TX < --- > SPI slave RX
- SPI master RX < --- > SPI Slave TX
- SPI master SCK < ---> SPI slave SCK
- SPI master CS < --- > SPI slave CS

然后按照以下步骤运行并检查结果：

- 插入 Micro USB 连接 PC 和 S32K144EVB-Q100 目标板
- 在 PC 上打开 UART 调试终端，波特率设置为 115200bps
- 在 S32DS 工作台打开项目
- 编译样例工程并下载到目标板运行
- 按任意键运行例程
- 数据传输完成后，结果将显示在 UART 终端上。

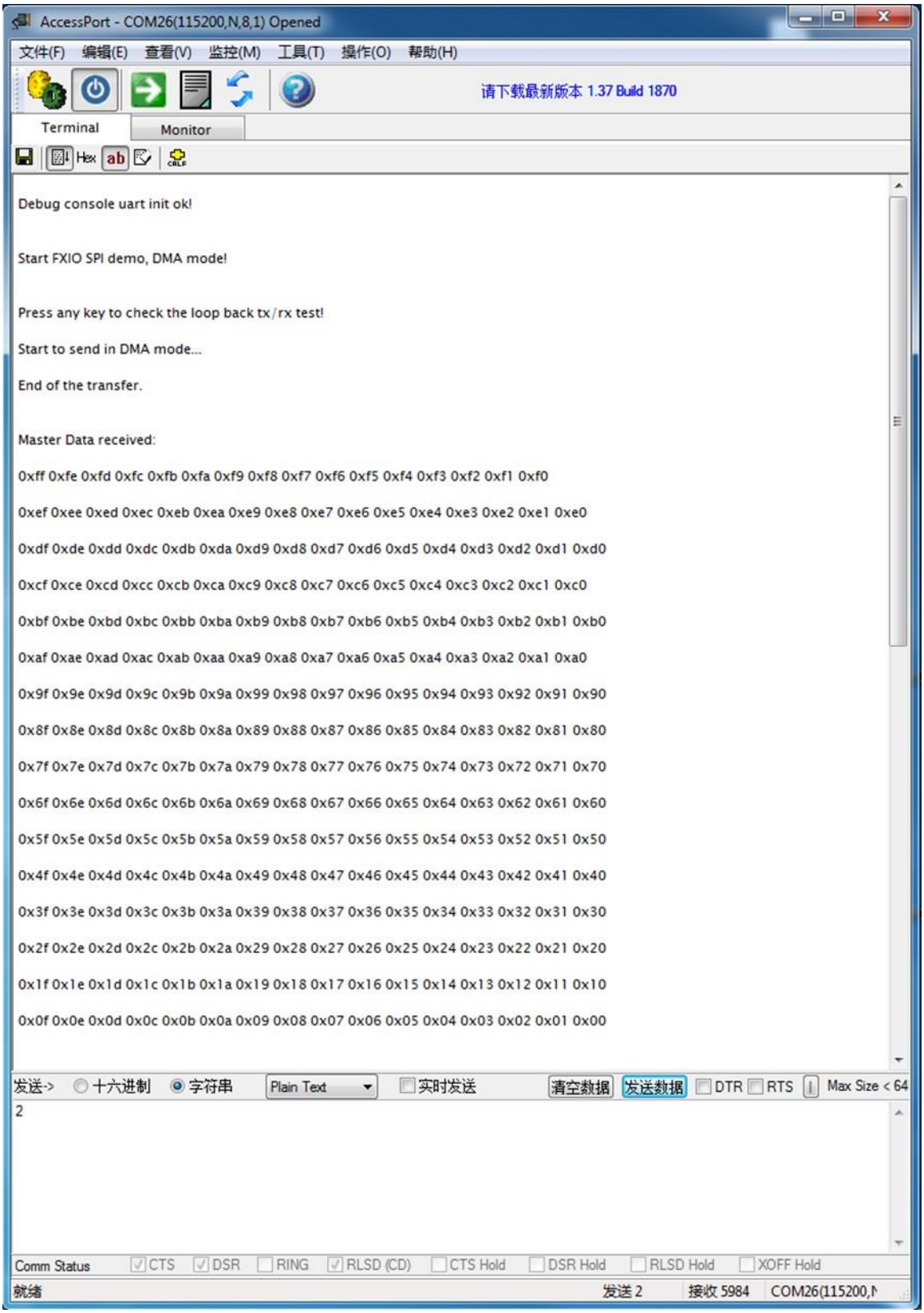

AccessPort - COM26(115200,N,8,1) Opened
文件(F) 编辑(E) 查看(V) 监控(M) 工具(T) 操作(O) 帮助(H)
请下载最新版本 1.37 Build 1870
Terminal
Monitor
Hex
ab
Debug console uart init ok!
Start FXIO SPI demo, DMA mode!
Press any key to check the loop back tx/rx test!
Start to send in DMA mode...
End of the transfer.
Master Data received:
0xff 0xfe 0xfd 0xfc 0xfb 0xfa 0xf9 0xf8 0xf7 0xf6 0xf5 0xf4 0xf3 0xf2 0xf1 0xf0
0xef 0xee 0xed 0xec 0xeb 0xea 0xe9 0xe8 0xe7 0xe6 0xe5 0xe4 0xe3 0xe2 0xe1 0xe0
0xdf 0xde 0xdd 0xdc 0xdb 0xda 0xd9 0xd8 0xd7 0xd6 0xd5 0xd4 0xd3 0xd2 0xd1 0xd0
0xcf 0xce 0xcd 0xcc 0xcb 0xca 0xc9 0xc8 0xc7 0xc6 0xc5 0xc4 0xc3 0xc2 0xc1 0xc0
0xbf 0xbe 0xbd 0xbc 0xbb 0xba 0xb9 0xb8 0xb7 0xb6 0xb5 0xb4 0xb3 0xb2 0xb1 0xb0
0xaf 0xae 0xad 0xac 0xab 0xaa 0xa9 0xa8 0xa7 0xa6 0xa5 0xa4 0xa3 0xa2 0xa1 0xa0
0x9f 0x9e 0x9d 0x9c 0x9b 0x9a 0x99 0x98 0x97 0x96 0x95 0x94 0x93 0x92 0x91 0x90
0x8f 0x8e 0x8d 0x8c 0x8b 0x8a 0x89 0x88 0x87 0x86 0x85 0x84 0x83 0x82 0x81 0x80
0x7f 0x7e 0x7d 0x7c 0x7b 0x7a 0x79 0x78 0x77 0x76 0x75 0x74 0x73 0x72 0x71 0x70
0x6f 0x6e 0x6d 0x6c 0x6b 0x6a 0x69 0x68 0x67 0x66 0x65 0x64 0x63 0x62 0x61 0x60
0x5f 0x5e 0x5d 0x5c 0x5b 0x5a 0x59 0x58 0x57 0x56 0x55 0x54 0x53 0x52 0x51 0x50
0x4f 0x4e 0x4d 0x4c 0x4b 0x4a 0x49 0x48 0x47 0x46 0x45 0x44 0x43 0x42 0x41 0x40
0x3f 0x3e 0x3d 0x3c 0x3b 0x3a 0x39 0x38 0x37 0x36 0x35 0x34 0x33 0x32 0x31 0x30
0x2f 0x2e 0x2d 0x2c 0x2b 0x2a 0x29 0x28 0x27 0x26 0x25 0x24 0x23 0x22 0x21 0x20
0x1f 0x1e 0x1d 0x1c 0x1b 0x1a 0x19 0x18 0x17 0x16 0x15 0x14 0x13 0x12 0x11 0x10
0x0f 0x0e 0x0d 0x0c 0x0b 0x0a 0x09 0x08 0x07 0x06 0x05 0x04 0x03 0x02 0x01 0x00
发送-> 十六进制 字符串 Plain Text 实时发送 清空数据 发送数据 DTR RTS Max Size < 64
2
Comm Status CTS DSR RING RLSD (CD) CTS Hold DSR Hold RLSD Hold XOFF Hold
就绪 发送 2 接收 5984 COM26(115200,N

图9. 终端实用程序输出

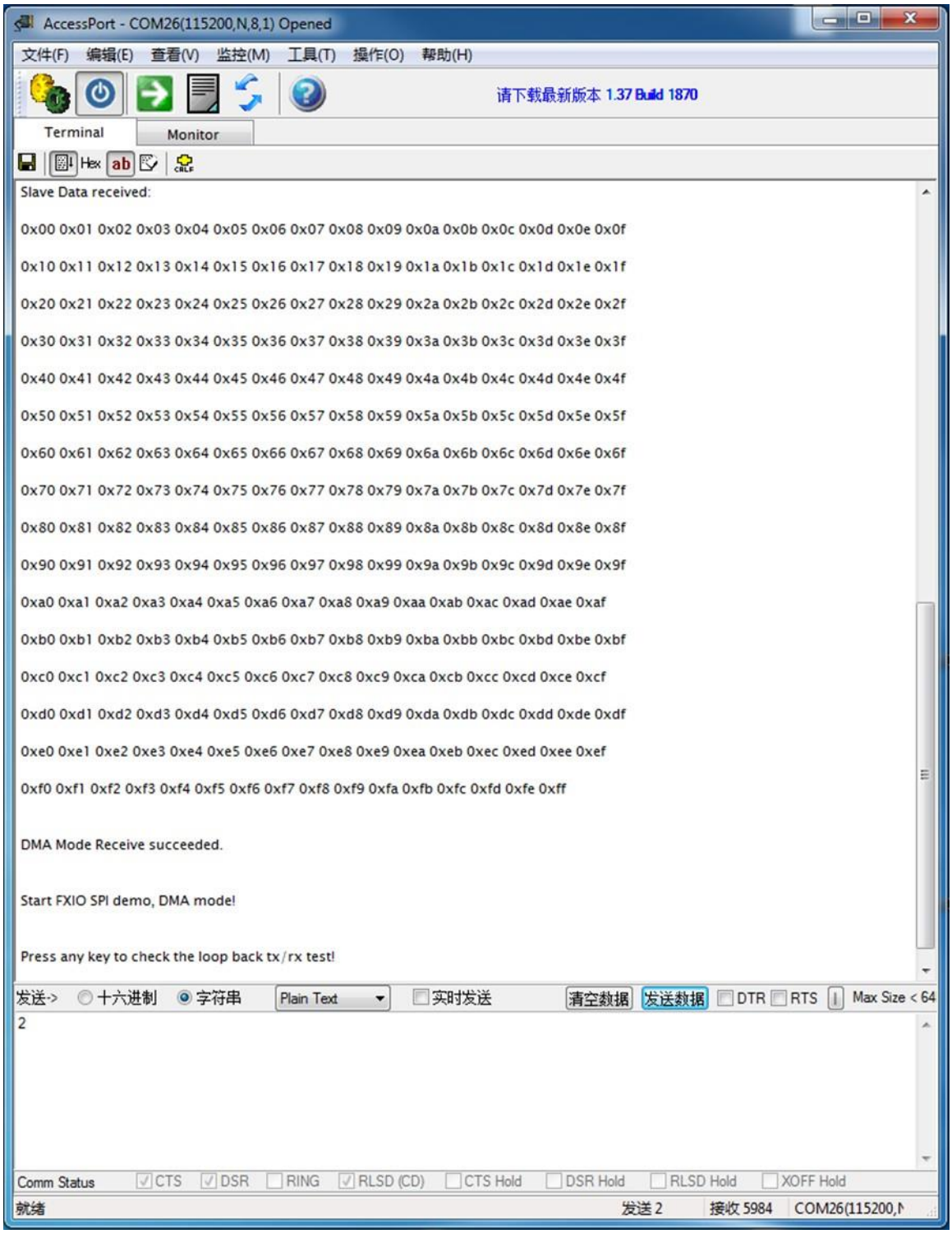


图10. 终端实用程序输出（续）

该软件旨在确保 SPI 主设备向从机传输从 0x0 到 0xFF的 256 个字节的。同时从设备向主设备传输从 0xFF 到 0x0 的 256 个反转字节。使用调试终端检查数据传输结果是否正确。

5. 使用 FlexIO 模拟 I2C 总线主设备

5.1. 简介

本节介绍如何使用 FlexIO 模拟 I2C 主设备。该例程使用开发板 S32K144EVB-Q100 通过 I2C 总线与六轴传感器 FXOS8700CQ 通信。从 FXOS8700CQ 读取的数据通过 UART 口发送到 PC 终端。

下图显示了硬件平台和数据流。有关图表中Open- SDA的更多信息，请参见开发板的相关材料。

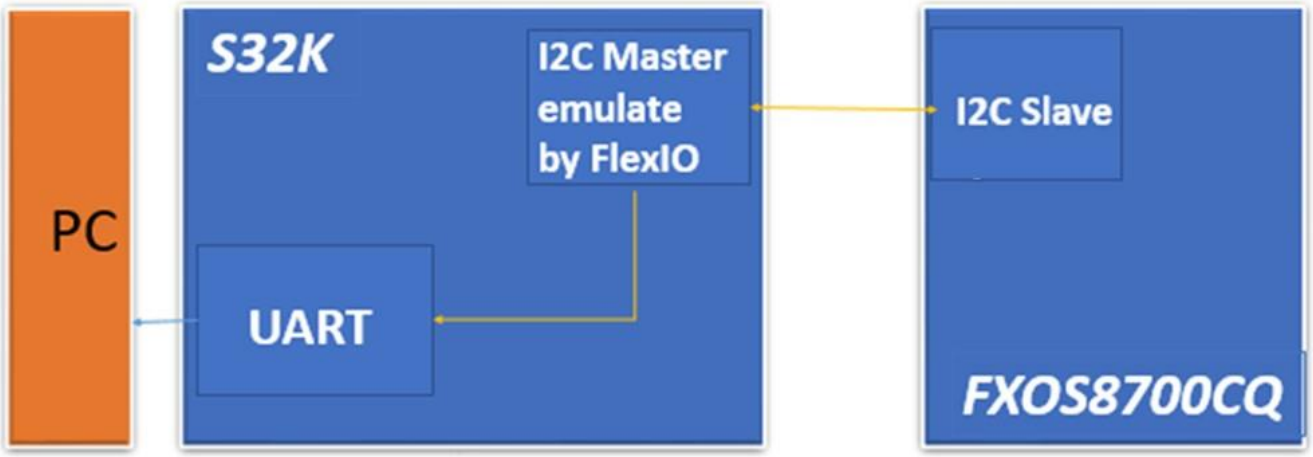


图1. S32K144EVB-Q100 与 OS8700CQ 互联

5.2. 总体说明

I2C主机使用以下资源进行模拟：

- 两个移位器 — 分别用作发射器和接收器。
- 两个定时器 — 一个用于SCL输出生成，另一个用于两个移位器的加载/存储/移位控制。
- 两个引脚 — 分别用作SDA和SCL。

下图显示了资源分配：

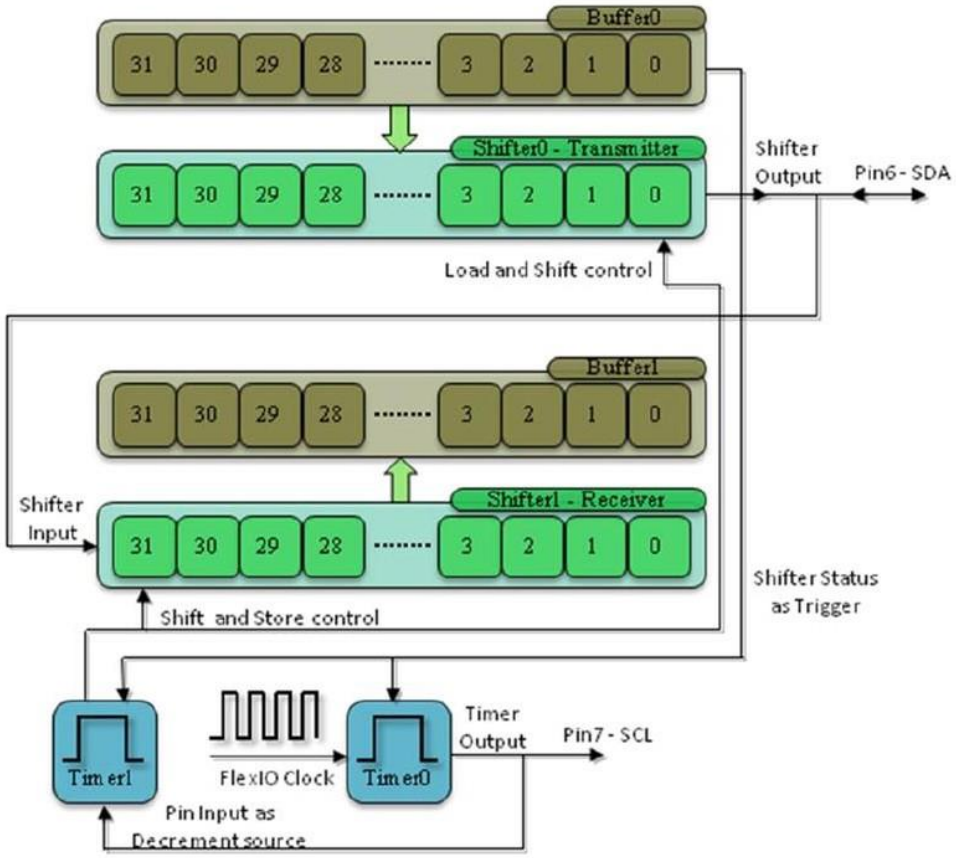


图2. FlexIO 的资源分配以模拟 I2C 主控

5.3. 移位器和定时器的配置

本节提供移位器和定时器的详细配置。

要了解这些配置的详细信息，请参阅 S32K 参考手册。

5.3.1. 移位器 0 的配置

移位器 0 用作发送器。它具有以下初始配置。

表15. 移位器 0 的初始配置

项目	配置
移位器模式	发送模式
定时器选择	定时器 1
定时器极性	上升沿移位
引脚选择	FlexIO D0
引脚配置	开漏或双向输出使能
引脚电平	低电平有效

项目	配置
输入源	从引脚输入
起始位	逻辑值 0
停止位	逻辑值 1
使用缓冲区	位字节交换寄存器

5.3.2. 移位器 1的配置

移位器 1 用作接收器。它具有以下初始配置。

表16. 移位器 1 初始配置

项目	配置
移位器模式	接收模式
定时器选择	定时器 1
定时器极性	下降沿移位
引脚选择	FlexIO D0
引脚配置	输出禁用
引脚电平	高电平有效
输入源	从引脚输入
起始位	禁用
停止位	逻辑值 0
使用缓冲区	位字节交换寄存器

5.3.3. 定时器 0 的配置

定时器 0 用于产生 SCL 输出和触发定时器 1。它具有以下初始配置。

表17. 定时器 0 初始配置

项目	配置
定时器模式	双 8 位计数器波特/位模式
触发选择	移位器 0 状态标志
触发电平	低电平有效
触发源	内部触发
引脚选择	FlexIO D1
引脚配置	开漏或双向输出使能
引脚电平	高电平有效
定时器初始输出	使能时输出逻辑 0，不受复位影响
定时器递减源	按 FlexIO 时钟递减，并移位输出
定时器使能条件	高电平触发
定时器禁用条件	定时器比较时禁用
定时器复位条件	在定时器引脚等于定时器输出时复位

项目	配置
起始位	启用
停止位	定时器禁用时启用
定时器比较值	((n*9+1)*2-1<<8) | (baudrate_divider)) [5]

5.3.4. 定时器 1 的配置

定时器 1 用于控制移位器 0 和移位器 1。它有以下初始配置。

表18. 定时器 1 的配置

项目	配置
定时器模式	单 16 位计数器模式
触发选择	移位器 0 状态标志
触发电平	低电平有效
触发源	内部触发
引脚选择	FlexIO D1
引脚配置	输出禁用
引脚电平	低电平有效
定时器初始输出	使能时输出逻辑 1，不受复位影响
定时器递减源	按引脚输入递减，移位时钟等于引脚输入
定时器使能条件	定时器 0 启用时
定时器禁用条件	定时器 0 禁用时
定时器复位条件	不复位
起始位	启用
停止位	在定时器比较时启用
定时器比较值	0x0F

5.4. 软件概述

通过使用 FLEXIO_I2C 驱动程序，可以在 S32K1 系列 MCU 上通过使用 FlexIO 模块来实现 I2C 总线通信。

特征

- 仅限主机操作
- 中断、DMA或轮询模式
- 提供阻塞和非阻塞发送和接收功能
- 7 位寻址

[5] n 是传输中的字节数。Baudrate_divider 是用于对来自 FlexIO 时钟源的波特率进行分频的值。

- 时钟延展
- 可配置的波特率
- 功能

初始化

在使用任何 FlexIO 驱动程序之前，必须首先使用函数 FLEXIO_DRV_InitDevice 初始化设备。然后在使用函数 FLEXIO_I2C_DRV_MasterInit() 初始化 FLEXIO_I2C 驱动程序。如果有足够的资源可用，可以在同一个 FlexIO 设备上使用更多的驱动程序实例。同一 FlexIO 设备上的不同驱动程序实例可以相互独立运行。当不再需要时，可以使用 FLEXIO_I2C_DRV_MasterDeinit() 释放硬件资源。

主机模式

主机模式提供向/从任何 I2C 从设备发送或接收数据的功能。在初始化时配置从地址和波特率，也可以在运行过程中使用 FLEXIO_I2C_DRV_MasterSetBaudRate() 或 FLEXIO_I2C_DRV_MasterSetSlaveAddr() 更改它们。请注意，由于模块限制，无法实现任意波特率。驱动程序会将波特率设置为尽可能接近所请求的波特率，但可能存在很大差异，特别是在使用低频 FlexIO 时钟而需要高波特率。应用程序应在 FLEXIO_I2C_DRV_MasterSetBaudRate() 之后调用 FLEXIO_I2C_DRV_MasterGetBaudRate() 以检查实际的波特率。

要向/从当前配置的从地址发送或接收数据，请使用函数 FLEXIO_I2C_DRV_MasterSendData() 或 FLEXIO_I2C_DRV_MasterReceiveData()（或相应的阻塞函数）。参数 sendStop 可用于链接多个传输，它们之间有 RE-START 信号。最后一次传输需要将 sendStop 设置为 true。此驱动程序不支持使用用户通过回调函数进行连续发送/接收。回调函数仅用于表示传输结束。

阻塞操作只会在传输完成时返回，无论是成功还是失败。非阻塞操作会发起传输并返回 STATUS_SUCCESS，但模块仍然在传输中，直到当前传输完成才能发起另一次传输。当传输完成时，应用程序可以通过回调得到通知，也可以通过调用 FLEXIO_I2C_DRV_MasterGetStatus() 来检查当前传输的状态。如果传输仍在进行，此函数将返回 STATUS_BUSY。如果传输完成，该函数将返回 STATUS_SUCCESS 或错误代码，具体取决于传输的结果。

驱动程序支持中断、DMA和轮询模式。在轮询模式下，通过函数 FLEXIO_I2C_DRV_MasterGetStatus() 检查和处理 FlexIO 模块报告的事件来确保传输的进度。此函数需要被频繁地调用（每个传输字节至少一次）以避免 Tx 下溢或 Rx 上溢。在 DMA 模式下，驱动程序将使用 DMA 通道来传输数据。需要在 flexio_i2c 驱动程序初始化之前初始化 DMA 通道。flexio_i2c 驱动程序会设置 DMA 请求源。

在使用 FLEXIO_I2C 驱动程序之前，必须先配置 FlexIO 时钟。有关时钟配置，请参阅 SCG HAL 和 PCC HAL。

以下提示是使用 FlexIO 模拟 I2C 总线主控的主要配置点。

处理发送器和接收器：两个移位器共用一个定时器（Timer 1）和SDA引脚（Pin 0），分别用作发送器和接收器。因此，对于传输中的每个字节，同时使用两个移位器。在每个字节传输后读取接收器以清除接收缓冲区和状态标志。发送器在接收时必须发送 0xFF 以将输出设为三态。

定时器触发设置：定时器的触发源设置为发射器的状态标志。将一个字节填充到发送器的缓冲区会清除使状态标志无效，从而使定时器 0 开始递减计数。定时器 1 的递减源设置为 SCL 引脚输入。每个 SCL 边沿使定时器 1 减 1。SCL的两个 边沿使定时器 1 经过一个周期，移位器中的数据相应的移一位。

TIMCMP（定时器比较寄存器）设置：定时器的 TIMCMP 存储定时器的 Module 值。TIMCMP 0 的高 8 位需要设置为传输中 SCL 边沿（两个边沿）数量的值。计算该值时需要考虑数据字节数、ACK/NACK 位和停止条件位。TIMCMP 0 的低 8 位用于配置波特率。TIMCMP 1 的值是根据单个帧中的位数计算得出的。

双缓冲移位器的操作：移位器设计为双缓冲结构。为避免在高波特率下发送或接收多个字节时发生下溢，发送移位器及其缓冲区应提前由软件填充数据。该过程可仪描述为以下步骤：（1）将第一个字节填充到发送缓冲区中；（2）通过轮询状态标志等待第一个字节装入发送移位器。将第二个字节填充到发送缓冲区中；（3）等待第一个字节移出，第二个字节装入发送移位器。在此期间，第一个接收到的字节被移入接收移位器。等待第一个接收到的字节被存储到接收缓冲区中后读取接收缓冲区。然后将第三个字节填充到发送缓冲区中。从第三步到最后的读取操作，可以使用轮询/中断/DMA模式。在延时上，DMA小于中断，中断小于轮询。

ACK/NACK 位生成和检查：在 I2C 总线协议中，每次传输后都需要一个 ACK 或 NACK 位。发送器的 SSTOP 位可用于生成 ACK/NACK。将 SSTOP 位设置为 0 以生成 ACK，将其设置为 1 以生成 NACK。接收器的 SSTOP 位可用于检查ACK/NACK。要移出/移入 ACK/NACK 位，定时器 0 需要输出两个额外的边沿。当定时器 1 减少到 0 时，接下来的 SCL 会将 ACK/NACK 位移出/移入。

重复的 START 和 STOP 信号的产生：（重复的）START 信号是 SCL 处于高电平期间的 SDA下降沿，STOP 信号是SCL处于高电平器件的 SDA 的上升沿。因此，需要提前将SDA 置高（SCL 处于低电平期间）以产生重复的 START 信号，置低以产生 STOP 信号。这些可以通过在每个字节传输的 SCL 的最后一个下降沿将一个额外的字节加载到发送器中来实现。但是，只会移出最高位。这也由定时器控制。因此，如果需要生成重复的 START 信号，则加载 0xFF，如果生成 STOP 信号，则加载 0x00。（重复的）START 和 STOP 信号的电平分别由发送器移位器配置寄存器中的 SSTART 和 SSTOP 决定。在第一个数据位移出之前，配置的起始位被加载到发送移位器中，然后被移出。当定时器 0 递减计数到 0 时，配置的停止位将被加载到发送移位器中，然后移出。还有一点，如果停止位被使能，当移位器初始配置为发送模式时，移位器将立即加载停止位。

5.5. 执行

本节介绍软件实现。本应用中实现的 I2C 功能基于 S32K SDK（S32DS_v2018 中包含的软件开发套件版本）。

该演示以轮询模式运行。本节介绍几个主要的驱动功能。

5.5.1. 初始化函数

Initialize 函数用于在应用程序初始化阶段配置移位器和定时器。原型是：

FlexIO_I2C_Init()

5.5.2. 传输函数

该函数用于向指定地址的从设备传输一个或多个字节。原型是：

*status_t FLEXIO_I2C_DRV_MasterSendData (flexio_i2c_master_state_t * master, const uint8_t * txBuff, uint32_t txSize, bool sendStop)*

函数参数：

master - 指向 FLEXIO_I2C 主驱动程序上下文结构的指针。

txBuff - 指向要传输的数据的指针

txSize - 要传输的数据的字节长度

sendStop - 指定传输结束后是否产生停止条件

Returns - 错误或成功状态由 API 返回

5.5.3. 接收函数

该函数用于从指定地址的从设备接收一个或多个字节。原型是：

*status_t FLEXIO_I2C_DRV_MasterReceiveData (flexio_i2c_master_state_t * master,*
*uint8_t * rxBuff,*
uint32_t rxSize,
bool sendStop
)

函数参数：

master	指向 FLEXIO_I2C 主驱动程序上下文结构的指针。
rxBuff	指向存储接收数据的缓冲区的指针
rxSize	要传输的数据的字节长度
sendStop	指定在接收结束后是否生成停止条件
Returns	错误或成功状态由 API 返回。

5.6. 运行例程

此例程在 S32K144EVB-Q100 上运行。I2C 主机的 FlexIO 引脚分配如下表所示：

表19. I2C 主设备的 FlexIO 引脚分配表

FlexIO I2C 主机	
FlexIO I2C master SDA Pin:FlexIO_D0	PTD0
FlexIO I2C master SCL Pin:FlexIO_D1	PTD1

按照以下步骤运行例程并检查结果：

- 插入 Micro USB连接 PC 和 S32K144EVB-Q100 目标板
- 在 PC 上打开 UART 调试终端，波特率设置为 115200bps
- 在 S32DS 工作区中打开项目
- 编译例程并将固件下载到目标板运行
- 结果在主机终端显示

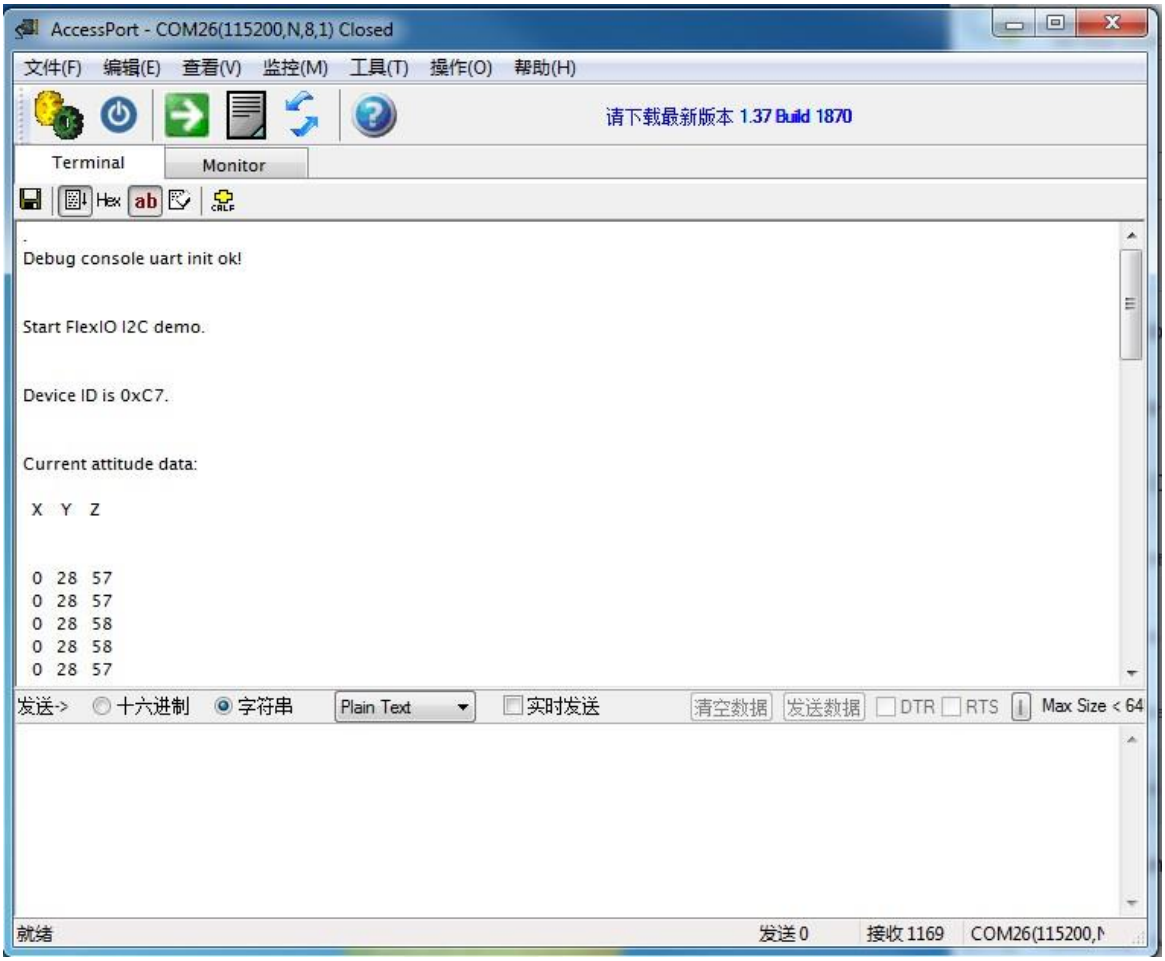


图3. 终端实用程序输出

6. 使用 FlexIO 生成 PWM

6.1. 简介

此例程基于 SDK（包含在 S32DS_v2018 中的软件开发套件中）和基本裸机驱动程序创建了一个简单的演示，使用 FlexIO 模块轻松实现指定频率和占空比配置的 PWM。

本节介绍如何通过 FlexIO 生成 PWM。此应用使用了图 2 中的开发板 S32K144EVB-Q100。

在应用中，FlexIO 在 FXIO0_D0 (PTD0) 引脚上生成 PWM 波形。PTD0 也连接到蓝色LED灯。使用 PWM 点亮不同占空比的蓝色 LED，显示不同占空比配置下的亮度变化。

6.2. 总体概述

生成 PWM 使用以下资源：

一个定时器 — 配置为 8 位 PWM 模式以控制相关引脚输出。

一个引脚 — 由定时器控制引脚输出并生成 PWM。

下图显示了资源分配：

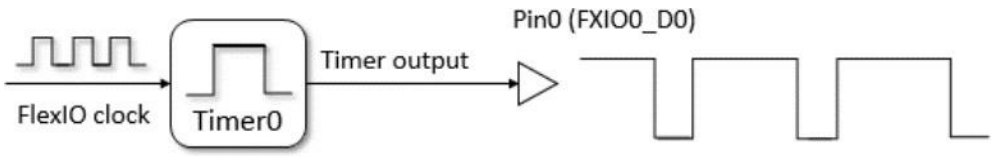


图4. FlexIO 的资源分配以生成 PWM

以下部分提供了详细的配置和使用信息。

6.3. 定时器的配置

本节提供定时器的详细配置。

要了解这些配置，请参阅以下说明和参考手册。

定时器 0 的配置

定时器 0 产生 PWM 输出到引脚 0。它具有以下初始配置。

表1. 定时器 0 的初始配置

项目	配置
定时器模式	双 8 位 PWM 模式
触发模式	N/A
触发极性	N/A
触发源	内部
引脚选择	FlexIO D0
引脚配置	输出使能
引脚电平	高电平有效
定时器初始输出	使能时输出逻辑 1，不受复位影响

定时器递减源	按 FlexIO 时钟递减，并移位输出
定时器使能条件	始终启用
定时器禁用条件	不禁用
定时器复位条件	不重置
起始位	禁用
停止位	禁用
定时器比较值	(((FLEXIO_CLK / freq) * (100 - duty) / 100 - 1) << 8) | ((FLEXIO_CLK / freq) * duty / 100 - 1) (((FLEXIO_CLK*Low_Period)/2-1)<<8)|((FLEXIO_CLK*High_Period)/2-1) or ((((FLEXIO_CLK/freq)*(100-duty)/100)/2-1)<<8)|(((FLEXIO_CLK/freq)*duty/100)/2-1)

以下提示是使用 FlexIO 生成 PWM 的关键点。

- 定时器模式配置（TIMCTL[TIMOD] 和 TIMCTL[TIMDEC]）
  - 选择双 8 位 PWM 模式。在此模式下，计数器的低 8 位仅在定时器输出引脚为高电平时减少，而高 8 位仅在定时器输出引脚为低电平时减少。当计数器的低 8 位减少到 0 时，定时器翻转并导致高 8 位减少。低 8 位控制 PWM 高电平脉宽，高 8 位控制低电平脉宽。
  - 将 FlexIO 时钟（$FLEXIO\_CLK_{[1]}$）配置为定时器计数器递减源。因此，如果满足条件，计数器会在每个 FlexIO 时钟上递减。
- 定时器比较值（TIMCMP[CMP]）：
  - 定时器第一次使能时或者当定时器复位或减少到 0 时，定时器比较值被加载到定时器计数器中。将双 8 位值输入到这个比较寄存器中以确保 PWM 的每个周期可以从该寄存器中重新加载定时计数器。该比较值控制 PWM 频率和占空比。
  - 定时器比较值 = (((FLEXIO_CLK / freq) * (100 - duty) / 100 – 1) << 8) | ((FLEXIO_CLK / $freq_{[2]}$) * 占空比$_{[3]}$ / 100 - 1)
- 定时器输出配置（TIM CFG[TIMEOUT]）
  - 定时器输出设置为逻辑 1（引脚上的高电平），定时器不受复位影响。在 8 位 PWM 模式下，必须在定时器启用时将定时器输出设置为 1，否则计数器的低 8 位值不会如上所述减少。

[1] FLEXIO_CLK 是来自 SCG 或 MCG 等时钟模块的时钟，用于控制 FlexIO 时序。

[2] 生成的 PWM 频率。

[3] PWM 波形的占空比。

下图举例解释了 TIMCP=0x58 时的时序和信号。

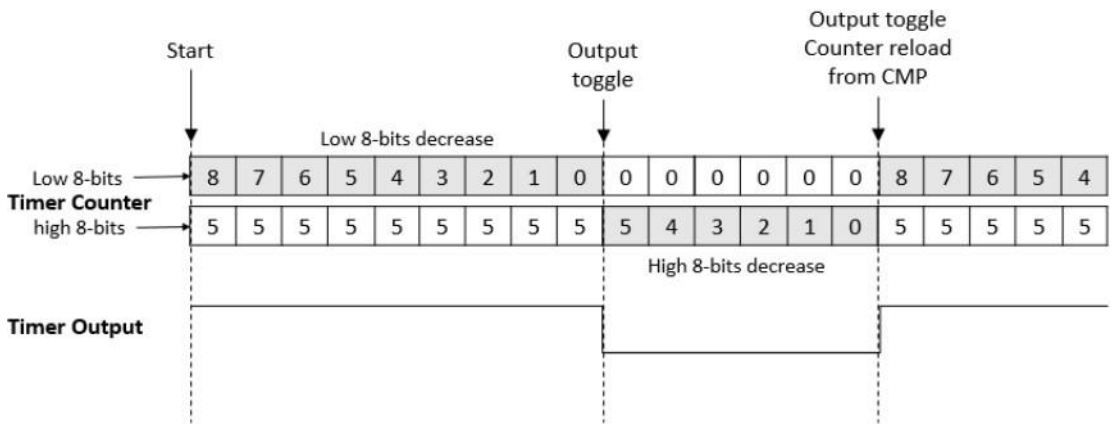


图5. 时间配置

6.4. 软件实现概述

本节介绍软件实现。此应用程序中使用的多个驱动函数都是基于S32K SDK（包含在S32DS_v2018中的软件开发套件中）实现的。

本应用笔记提供了一个软件包。样例工程包含一个源文件 main.c。该源文件提供了以下函数来将 FlexIO 配置为 PWM 发生器，所有这些函数都可以由用户在自己的代码中直接使用，只需稍作改动：

- FLEXIO_DRV_InitDevice(uint32_t instance, flexio_device_state_t *deviceState)
- flexio_pwm_init(uint32_t freq, uint8_t duty)
- flexio_pwm_start(void)
- flexio_pwm_stop(void)

6.4.1. flexio_init()

flexio_init() 函数启用 FlexIO IP 模块的时钟门控，并为 FlexIO 选择合适的外设时钟源。源文件中定义的 FLEXIO_CLK 正是外设时钟源的频率。该函数会重置 FlexIO IP 模块到初始状态。这是一个通用的 FlexIO IP 模块初始化函数，需要在使用其移位器和定时器之前调用。

6.4.2. flexio_pwm_init()

flexio_pwm_init() 函数将 定时器 0 配置为 8 位 PWM 模式，并在 pin0 输出以生成 PWM 波形。定时器的详细配置可以在 6.3 节中找到。此函数中的 “freq” 参数是要生成的 PWM 频率。该频率值必须在源文件中定义的 [MIN_FREQ, MAX_FREQ] 宏的范围内。“duty” 参数是以%为单位的指定占空比，范围为[1, 99]。

6.4.3. flexio_pwm_start()

flexio_pwm_start() 函数通过将 TIMOD 设置为 8 位 PWM 并开始生成 PWM 来启用定时器 0。

6.4.4. flexio_pwm_stop()

flexio_pwm_stop() 函数通过设置 TIMOD 来禁用 timer0，并禁用生成 PWM。

要将 FlexIO 用作 PWM 发生器，可以参考 main() 函数中一样依次调用这三个函数。它将 PWM 频率配置为 8KHz，并将占空比从 99 依次变化到 1，通过 PTD0 改变蓝色 LED 的亮度。

6.5. 运行例程

使用 Open-SDA 将编译出的程序下载到微控制器。PC 主机与 S32K144EVB-Q100 上的 Open-SDA 之间通过 USB 线连接后，PC 主机可以连接串口检查程序的运行。

运行过程中用户可以看到红色 LED 的亮度从最小变化到最大。也可以用示波器在 PTD0 引脚上捕获 PWM 信号，查看频率和占空比。

7. 使用 FlexIO 模拟 I2S 总线 Master

本节介绍如何使用 FlexIO 模拟 I2S 总线 Master。该应用使用了下图中的评估板 S32K144EVB-Q100。

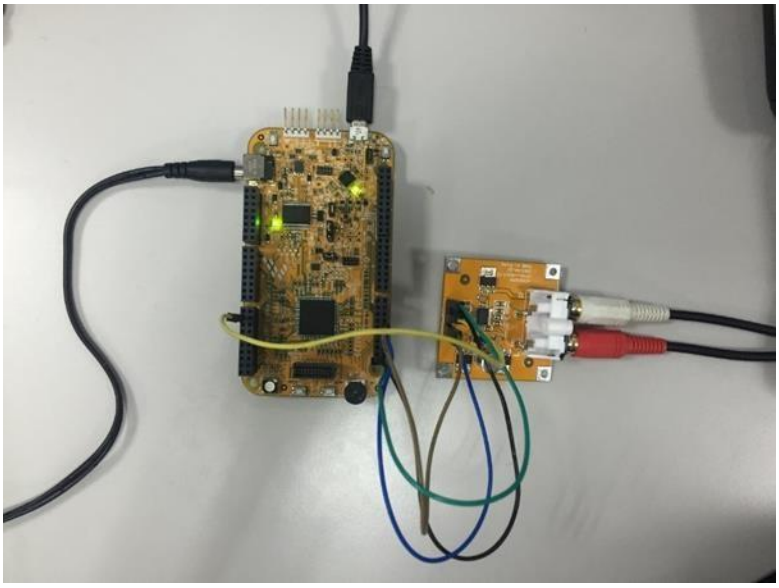


图6. S32K144EVB-Q100 接 PCM5102A

在应用中，FlexIO 模拟 I2S 接口与 PCM5102A（数模转换器）进行通信。音频数据从 S32K144 发送到 PCM5101A 进行播放。

下图显示了硬件平台和数据流。关于图中Open-SDA的更多信息，请参见NXP开发板的相关资料。

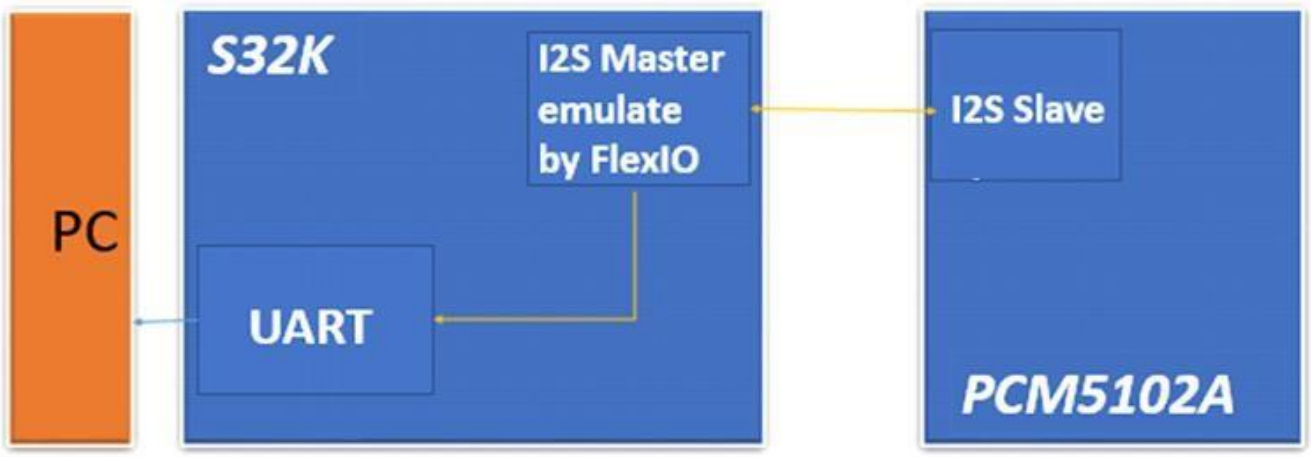


图7. 此应用程序的硬件平台和数据流

7.1. 总体概述

I2S 总线主机使用以下资源进行模拟：

一个移位器 — 用作数据发送器。

两个定时器 — 一个用于移位器的负载控制和 BCLK 输出生成，另一个用于 LRCLK 输出生成。

三个引脚 — 分别用作 DATA、LRCLK 和 BCLK。

下图显示了资源分配：

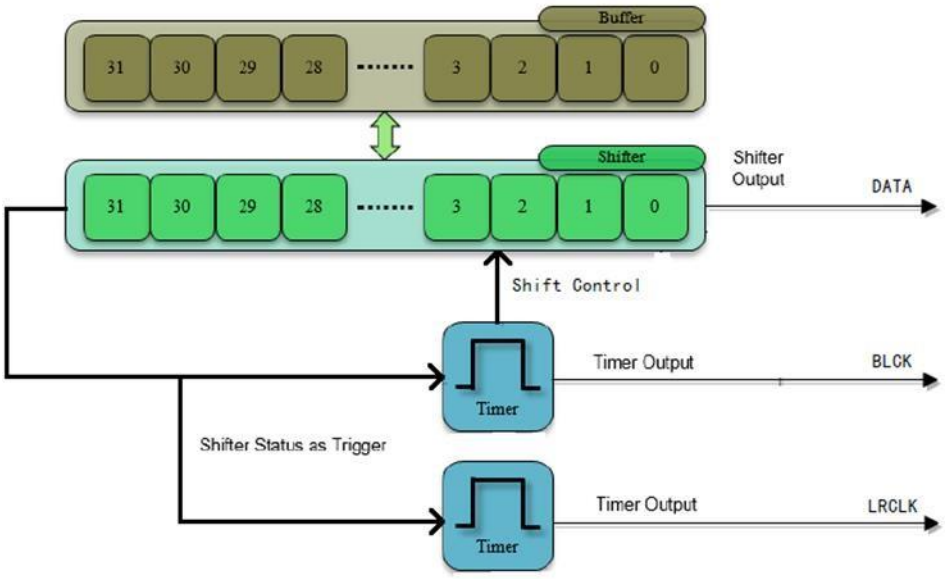


图8. FlexIO 的资源分配以模拟 I2S

以下部分提供了详细的配置和使用信息。

7.2. 移位器和定时器的配置

本节提供移位器和定时器的详细配置。

要了解这些配置，请参阅以下说明和参考手册。

移位器 0 的配置

移位器 0 用作数据发送器。它具有以下初始配置。

表2. 移位器 0 的初始配置

项目	配置
移位器模式	发送模式
定时器选择	定时器 0
定时器极性	上升沿移位数据
引脚选择	FlexIO D0
引脚配置	输出使能
引脚电平	高电平有效
输入源	从引脚输入
起始位	起始位禁用，发送器在第一次移位时加载数据
停止位	禁用
使用缓冲区	位字节交换寄存器

定时器 0 的配置

定时器 0 用于移位器的负载控制和 BCLK 输出生成。它具有以下初始配置。

表3. 定时器 0 的初始配置

项目	配置
定时器模式	双 8 位计数器波特/位模式。
触发选择	移位器 0 状态标志
触发电平	低电平有效
触发源	内部触发
引脚选择	FlexIO D1
引脚配置	输出使能
引脚电平	低电平有效
定时器初始输出	使能时输出逻辑 0，受复位影响
定时器递减源	按 FlexIO 时钟递减，并移位输出
定时器使能条件	高电平触发
定时器禁用条件	不禁用
定时器复位条件	定时器引脚等于定时器输出时复位
起始位	使能
停止位	禁止
定时器比较值	((n*2-1)<<8) | (baudrate_divider/2-1)) [6]

定时器 1 的配置

定时器 1 用于产生 LRCLK 输出。它具有以下初始配置。

表4. 定时器 1 的初始配置

项目	配置
定时器模式	单 16 位计数器模式
触发选择	移位器 0 状态标志
触发电平	低电平有效
触发源	内部触发
引脚选择	FlexIO D2
引脚配置	输出使能
引脚电平	高电平触发
定时器初始输出	使能时输出逻辑 0，受复位影响
定时器递减源	按 FlexIO 时钟递减，并移位输出
定时器使能条件	在定时器 0 启用时使能
定时器禁用条件	不禁用
定时器复位条件	不重置
起始位	禁用
停止位	禁用
定时器比较值	(baudrate_divider/2-1) [7]

[6] n 是传输中的字节数。Baudrate_divider 是用于对 FlexIO 时钟源的波特率进行分频的值。

7.3. 软件实现概述

本节介绍软件实现。该软件基于 S32K SDK（S32DS_v2018中包含的软件开发套件中），实现 FlexIO 模拟 I2S 的驱动函数。

该演示以轮询模式运行。本节主要介绍几个主要的驱动功能。

初始化函数

Initialize 函数用于在应用程序的初始化阶段配置移位器和定时器，原型是：

void FlexIO_I2S_Init(void)

传输函数

该函数用于向从设备传输一个或多个字节。原型是：

```
status_t FLEXIO_I2S_DRV_MasterSendData (        flexio_i2s_master_state_t *     master,
const uint8_t * txBuff,
uint32_t        txSize
)
```

此处，

master - 指向 FLEXIO_I2S 主驱动程序上下文结构的指针。

txBuff - 指向要传输的数据的指针

txSize - 要传输数据的字节长度

Returns - 错误或成功状态由API返回

7.4. 运行例程

此样例在 S32K144EVB-Q100 上运行。I2S 主机的 FlexIO 引脚分配如下表所示：

表5. I2S 主设备的 FlexIO 引脚分配表

FlexIO I2S 主机	
FlexIO I2S master DATA Pin:FlexIO_D0	PTD0
FlexIO I2S master BCLK Pin:FlexIO_D1	PTD1
FlexIO I2S master LRCLK Pin:FlexIO_D2	PTE15

完成后，按照以下步骤运行并检查结果：

[7] Baudrate_divider 是用于对 FlexIO 时钟源的波特率进行分频的值。

插入 Micro USB 以连接 PC 和 S32K144EVB-Q100 目标板

在 PC 上打开 UART 调试终端，波特率设置为 115200bps

打开 S32DS 工作台中的项目

编译示例工程并将程序下载到目标板

按任意键运行例程

数据开始传输后，有音乐输出。

8. 使用 FlexIO 模拟 LIN 主/从

8.1. 简介

独立外设模块 FlexIO 用作微控制器的附加外设模块，不能替代 LIN/UART 外设。本示例创建了一个基于 S32K SDK 的简单软件例程，也包括裸机配置示例驱动程序，可以用来通过使用FlexIO 模拟 LIN。

8.2. 使用 FlexIO 模拟 LIN

LIN 实现使用以下 FlexIO 模块资源，

发送器配置

一个 16 位定时器 - 双 8 位波特率模式。

一个 32 位移位器 - 传输模式。

两个输出引脚

由定时器控制以产生波特率。

输出移位器缓冲区数据。

接收机配置：

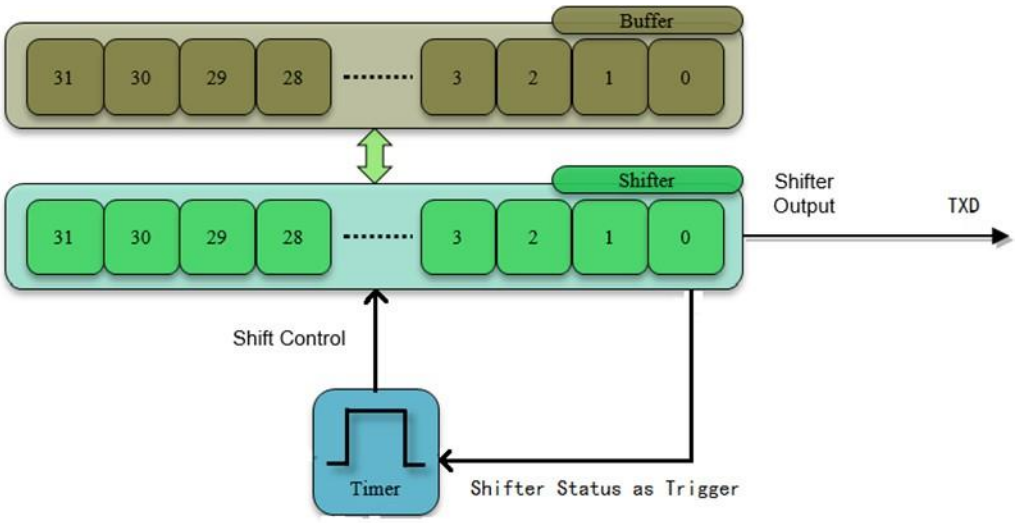
一个 16 位定时器 - 双 8 位波特率模式。

一个 32 位移位器 - 接收模式。

两个输入/输出引脚

生成波特率。

移位器输入数据。



注意：定时器和移位器配置基于第三章描述的UART配置。

8.3. 配置移位器和定时器

发射机：

在定时器复位中将 CMP 值修改为：

- 主机发送的 32 位 ID 或
- 主机发送一个 8 位字节数据。

• 加载下面的数据到移位器缓冲区

- PID << 24u | 0x555000
- 要发送的数据。

• 将数据移至引脚输出。

• 起始位和停止位在数据之前或之后自动加载。

接收机

当数据被完整移入后，会有事件通知。

• 状态标志指示何时可以读取数据（产生中断）。

• 存储在移位器缓冲区中。

• 读取移位器缓冲区字节交换寄存器。

LIN 主机和从机使用几乎相同的 FlexIO 配置，有稍许的不一样。

使用主机模式时：起始位和停止位禁用。

当使用 Slave 模式时：第一次的时候将定时器比较配置为等待 13 位长的 break 字段。

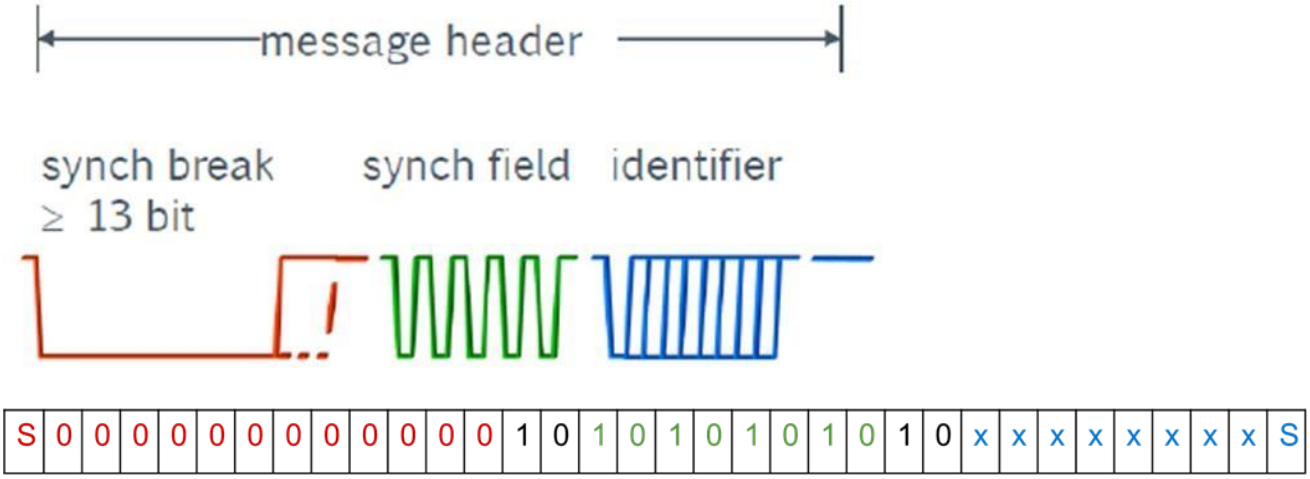


表6. 移位器 0 的初始配置

项目	配置
移位器模式	发送模式
定时器选择	定时器 0
定时器极性	上升沿移位
引脚选择	pin 0
引脚配置	输出使能
引脚电平	高电平有效
输入源	从引脚输入
起始位	发送器在第一次移位加载数据之前输出起始位值 “0”
停止位	发送器在存储时输出停止位值 “1”
使用缓冲区	移位缓冲器

表7. 定时器 0 的初始配置

项目	配置
定时器模式	双 8 位计数器波特率模式
触发选择	移位器 0 状态标志
触发电平	低电平有效
触发源	内部触发
引脚选择	FlexIO D1
引脚配置	输出禁止
引脚电平	高电平有效
定时器初始输出	使能时输出逻辑 0，受复位影响

定时器递减源	按 FlexIO 时钟递减，并移位输出
定时器使能条件	高电平触发
定时器禁用条件	定时器比较
定时器复位条件	上升沿触发
起始位	启用
停止位	在定时器比较和禁用时启用
定时器比较值	$((transfer_{size} \ll 1) - 1) \ll 8 \mid \left(\frac{FlexIO_{freq}}{BaudRate \ll 1}\right) - 1$

表8. 移位器 1 的初始配置

项目	配置
移位器模式	接受模式
定时器选择	定时器 1
定时器极性	换挡时钟负极
引脚选择	FlexIO D2
引脚配置	输出禁止
引脚电平	高电平有效
输入源	从引脚输入
起始位	发送器在第一次移位加载数据之前输出起始位值 “0”
停止位	发送器在存储时输出停止位值 “1”
使用缓冲区	移位器字节交换缓冲区

表9. 定时器 1 的初始配置

项目	配置
定时器模式	双 8 位计数器波特率模式
触发选择	pin 2 输入
触发电平	低电平有效
触发源	内部触发
引脚选择	FlexIO D3
引脚配置	输出禁止
引脚电平	高电平有效
定时器初始输出	使能时输出逻辑 0，受复位影响
定时器递减源	按 FlexIO 时钟递减，并移位输出
定时器使能条件	上升沿触发
定时器禁用条件	定时器比较
定时器复位条件	不重置
起始位	启用
停止位	在定时器比较和禁用时启用
定时器比较值	$((transfer_{size} \ll 1) - 1) \ll 8$

	$\left\lfloor \left(\frac{FlexIO_{freq}}{BaudRate \ll 1} \right) - 1 \right\rfloor$

FlexIO 输入频率是 SYSTEM OSC 除以 8 = 1Mhz

表10. 配置定时器 0 以发送LIN头文件

项目	配置
起始位	禁用
停止位	禁用

如此实施后，可用的 FlexIO 资源还有 2 个定时器、2 个移位器和 4 个引脚，它们可以用来生成 PWM 信号，模拟 UART、I2C、i2S 甚至另外一个 Master LIN 实例。

9. 结论

FlexIO 是 S32K 系列的新外设。由于移位器和定时器的高度灵活性，FlexIO 能够模拟各种协议，如 SPI、I2C、UART、I2S、LIN 以及 PWM，当然也可以用来模拟许多其他通信接口。

10. 修订历史

表11. 修订历史

修订号	日期	实质性变化
0	2018年6月	初版发布

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN12174
Rev. 0
06/2018