ANALOG DEVICES 超越一切可能™ | ADI AnalogDialogue

如何优化MCU SPI驱动程序以实现高ADC吞吐速率

Denny Wang，应用工程师
Sally Tseng，应用工程师

摘要

随着技术的进步，低功耗物联网(IoT)和边缘/云计算需要更精确的数据传输。图1展示的无线监测系统是一个带有24位模数转换器(ADC)的高精度数据采集系统。在此我们通常会遇到这样一个问题，即微控制单元(MCU)能否为数据转换器提供高速的串行接口。

本文描述了设计MCU和ADC之间的高速串行外设接口(SPI)关于数据事务处理驱动程序的流程，并简要介绍了优化SPI驱动程序的不同方法及其ADC与MCU配置。本文还详细介绍了SPI和直接存储器访问(DMA)关于数据事务处理的示例代码。最后，本文演示了在不同MCU（ADuCM4050、MAX32660）中使用相同驱动程序时ADC的吞吐率。

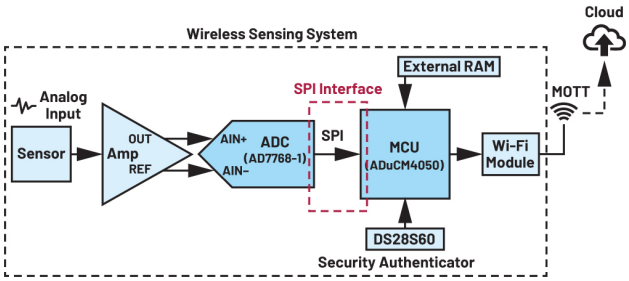

图1. 状态监控。

引言

通用SPI驱动程序简介

通常，MCU厂商会在例程代码中提供通用的SPI驱动程序/API。通用SPI驱动程序/API通常可以涵盖大多数用户的应用，这些代码可能包含许多配置或判断语句。但在某些特定情况下，比如ADC数据采集，通用的SPI驱动程序可能无法满足ADC数据的全速的吞吐速率需求，因为通用的驱动程序中有过多的配置，而未使用的配置会产生额外的开销并导致时间延迟。

```
void adi_spi_MasterSubmitBuffer (struct hDevice, struct *buffer)
{
    if(…)
        *p = xxx;
    else if(…)
        *p = xxx << xxx;
        for (…)
    :
    start_spi_transaction();
}
```

Config.

图2. 通用API的配置。

设计思路与实践框架

我们通常会选择低功耗高性能的MCU作为主机通过SPI提取ADC的输出数据。但是，由于ADI的SPI驱动程序的数据事务处理命令存在冗余，因此数据输出速率可能被显著降低。为了充分释放ADC的潜在速率，本文使用ADuCM4050和AD7768-1进行实验并尝试可能的解决方案。尽管在使用默认滤波器的情况下，ADuCM4050的最大数据输出速率可达256 kHz，但在当前情况下，其速率被限制在8 kHz。提高输出速率的潜在解决方案包括删除不必要的命令以及激活DMA控制器。本文将在以下小节中介绍这些思路。

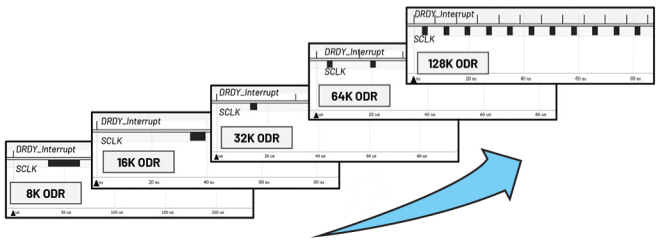

图3. 不同ODR以及DRDY与SCLK之间的关系。

以MCU作为主机

ADuCM4050 MCU是一款主时钟速率为26 MHz的超低功耗微控制器，内核为ARM® Cortex®-M4F处理器。ADuCM4050配有三个SPI，每个SPI都有两个DMA通道（接收和发射通道）可与DMA控制器连接。DMA控制器和DMA通道可实现存储器与外设之间的数据传输。这是一种高效的数据分配方法，可将内核释放以处理其他任务。

以ADC作为从机

AD7768-1是一款24位低功耗、高性能的Σ-Δ ADC。其数据输出速率(ODR)和功耗模式均可根据用户的要求进行配置。ODR由抽取系数和功耗模式共同决定，如表1中所示。

表1. 数据输出速率的功耗模式配置

功耗模式	抽取系数	ODR
快速功耗(MCLK/2)	×32	256 kHz
快速功耗(MCLK/2)	×64	128 kHz
中速功耗(MCLK/4)	×32	128 kHz
中速功耗(MCLK/4)	×64	64 kHz
低速功耗(MCLK/16)	×32	32 kHz
低速功耗(MCLK/16)	×64	16 kHz

AD7768-1的连续读取模式也是该产品的一个重要特性。ADC的输出数据存储在寄存器0x6C中。一般而言，每次读/写操作之前，ADC寄存器中的数据都需要地址才可以访问，但是连续读取模式则支持在收到每个数据就绪信号后直接从0x6C寄存器提取数据。ADC的输出数据为24位的数字信号，对应的电压如表2所示。

表2. 数字输出码和模拟输入电压

说明	模拟输入电压	数字输出码
+全摆幅-1 LSB	+4.095999512 V	0x7FFFFF
中间电平+1 LSB	+488 nV	0x000001
中间电平	0 V	0x000000
中间电平-1 LSB	−488 nV	0xFFFFFF
-全摆幅-1 LSB	-4.095999512 V	0x800001
-全摆幅	+4.096 V	0x800000

引脚连接示意图

ADuCM4050和AD7768-1组成的数据事务处理示例模型的引脚连接如图4所示。

ADC的复位信号引脚RST_1连接至MCU的GPIO28，而数据就绪信号引脚DRDY_1则连接至MCU的GPIO27。其余引脚则根据通用的SPI配置标准进行连接，其中MCU为主机，而ADC为从机。SDI_1接收MCU发送的ADC寄存器读/写命令，而DOUT_1则将ADC的输出数据发送至MCU。

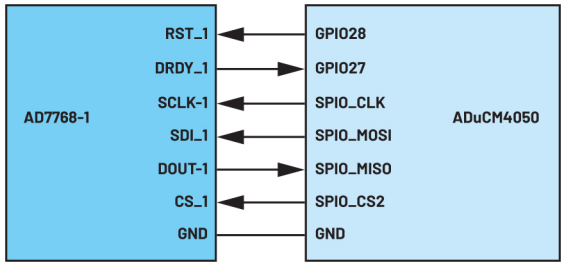

图4. AD7768-1和ADuCM4050的接口引脚连接。

数据事务处理的实现

中断数据事务处理

为实现连续数据事务处理，本文将MCU的GPIO27引脚（连接至ADC的DRDY_1引脚）用作中断触发引脚。ADC将数据就绪信号发送至GPIO27时会触发MCU包含数据事务处理命令的中断回调函数。如图5所示，数据采集必须在中断A和中断B之间的时间间隔内进行。

利用ADI的SPI驱动程序可以在ADC和MCU之间轻松实现数据事务处理。但是，由于驱动程序内存在冗余命令，ADC的ODR会被限制在8 kHz。本文尽可能地精简了代码以加快ODR，将介绍实现DMA数据事务处理的两种方法：基本模式的DMA事务处理和乒乓模式的DMA事务处理。

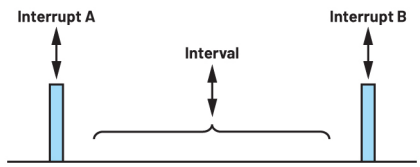

图5. 两次中断的时间间隔。

基本模式的DMA事务处理

在实现每个DMA事务处理之前需要对SPI和DMA进行配置（参见图6中的示例代码）。SPI_CTL为SPI配置，其值为0x280f，源于ADI的SPI驱动程序的设定值。SPI_CNT为传输字节数。由于每个DMA事务处理只能发送固定的16位数据，因此SPI_CNT必须是2的倍数。本例设置SPI_CNT为4，以满足ADC的24位的输出数据要求。SPI_DMA寄存器为SPI的DMA使能寄存器，设定其值为0x5以使能DMA接收请求。命令pADI_DMA0->EN_SET=(1<<5)使能第五个通道的DMA，即SPI0 RX。

表3. DMA结构寄存器

名称	说明
SRC_END_PTR	来源端指针
DST_END_PTR	目标端指针
CHNL_CFG	控制数据配置

每个DMA通道都有一个DMA结构寄存器，如表3中所示。需要指出的是，这里的数据来源地址的结尾（即SPI0 Rx，亦即来源端指针SRC_END_PTR）在整个操作期间无需增加，因为Rx FIFO会自动将寄存器中的数据推送出去。另一方面，数据目标地址的结尾（即目标端指针DST_END_PTR）根据ADI的SPI驱动程序的使用函数计算得出，即目标地址+ SPI_CNT -2。

当前地址为内部数组缓冲区的地址。DMA控制数据配置CHNL_CFG包括来源数据大小、来源地址增量、目标地址增量、剩余传输次数和DMA控制模式等设置，其值**0x4D000011**按照表4中所述的设置配置。

表4. 控制数据配置0x4D00011的DMA配置

寄存器	说明	值
DST_INC	目标地址增量	2字节
SRC_INC	来源地址增量	0
SRC_SIZE	来源地址增量	2字节
N_minus_1	当前DMS周期中的总传输次数-1	1 (N = 2)
Cycle_ctrl	DMA周期的工作模式	基本模式

SCLK时钟通过伪读取命令**SPI_SPI0 -> RX**启动，输出数据通过MISO从ADC传至MCU。MOSI上其它的数据传输可以忽略不计。一旦完成Rx的FIFO填充，DMA请求就会生成从而激活DMA控制器，以将数据从DMA来源地址（即SPI0 Rx FIFO）传输至DMA目标地址（即内部数组的缓冲区）。值得注意的是，**SPI_DMA=0x3**时会生成Tc请求。

最后，通过将当前目标地址加4的方式将目标地址用于下一个4字节的传输。

请注意，SPI0 DMA通道的**pADI_DMA0->DSTADDR_CLR**和**pADI_DMA0->RMSK_CLR**必须在首次中断触发之前在主函数中设置。前一个为DMA通道目标地址减量使能清零寄存器，用于在增量模式下设置每次DMA传输后的目标地址移位（目标地址计算函数仅在增量模式下有效）。后一个为DMA通道请求屏蔽清零寄存器，用于将通道的DMA请求状态清零。

基本模式的DMA事务处理时间图如图7a所示。图中三个时隙分别代表DRDY信号、SPI/DMA设置和DMA数据事务处理。在该模式中，CPU的空闲时间较多，因此希望DMA控制器在处理数据传输时能将任务分配给CPU。

```
//SPI settings
pADI_SPI0->CTL = 0x280f;
pADI_SPI0->CNT = (uint16_t)4;
pADI_SPI0->DMA = 0x5;

//DMA settings
pADI_DMA0->EN_SET = (1<<5);
pPrimaryCCD[5].DMASRCEND = (uint32_t)&pADI_SPI0->RX;
pPrimaryCCD[5].DMADSTEND(uint32_t)Current_address;
pPrimaryCCD[5].DMACDC = 0x4D000011;

//Dummy read
pADI_SPI0->RX;

//Data destination address maintenance
Current address = Current_address + 4;
```

图6. 基本DMA事务处理模式的代码。

乒乓模式的DMA事务处理

在执行伪读取命令后，DMA控制器会开始数据事务处理，从而使得MCU的CPU处于空闲状态而不处理任何任务。如果能够让CPU和DMA控制器同时工作，那么任务处理就从串行模式转变为并行模式。这样，就可以同时进行DMA配置（通过CPU）以及DMA数据事务处理（通过DMA控制器）。为实现这一思路，需要设置DMA控制器处于乒乓模式。乒乓模式将两组DMA结构进行了整合：主结构和备用结构。每次DMA请求时，DMA控制器会在两组结构之间自动切换。变量p的初始设置为0，其值表示是主DMA结构(p = 0)还是备用DMA结构(p = 1)负责数据事务处理。如果p = 0，则在收到伪读取命令时启动主DMA结构进行数据事务处理，同时会为备用DMA结构分配值，使其在下一个中断周期内负责数据事务处理。如果p = 1，则主结构和备用结构的作用互换。当仅有主结构处于基本DMA模式时，在DMA事务处理期间对DMA结构的修改会失败。乒乓模式使得CPU能够访问和写入备用DMA结构，而DMA控制器可以读取主结构，反之亦然。如图7b所示，由于DMA的结构配置是在最后一个周期内完成的，因此在DRDY信号从ADC传送至MCU后DMA数据事务处理可以被立即执行，使得CPU和DMA同时工作而无需等待。现在，ADC的ODR得到了提升空间，因为总的工作时间已大大缩短。

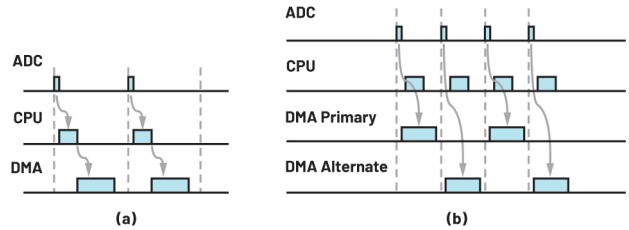

图7. (a)基本模式DMA和(b)乒乓模式的时间图。

中断处理程序的优化

两次DRDY信号之间的时间间隔不仅包括了中断回调函数的命令执行时间，还包括了ADI的GPIO中断处理函数的命令执行时间。

当MCU启动时，CPU会运行启动文件（即startup.s）。所有事件的处理函数均在该文件中定义，包括GPIO中断处理函数。一旦触发GPIO中断，CPU就会执行中断处理函数（即ADI的GPIO驱动程序中的GPIO_A_INT_HANDLER和GPIO_B_INT_HANDLER）。通用的中断处理函数会在所有的GPIO引脚中搜索触发中断的引脚并清零其中断状态、运行回调函数。由于DRDY是本文应用的唯一中断信号，因此可以对函数进行简化以加快进程。可选的解决方案包括(1)在启动文件中重新定位目标，以及(2)修改原始的中断处理函数。重新定位目标意味着自定义中断处理函数，并替换启动文件中的原始的中断处理函数。

而修改原始的中断处理函数只需要添加一个自定义的GPIO驱动程序。本文采用第二种方案修改原始的中断处理函数，如图8所示。该方案只将连接至DRDY的GPIO的引脚中断状态清零，并直接转到回调函数。请注意，这里需要通过取消选择build target中关于原始GPIO驱动函数的勾选框内容来隔离原始的GPIO驱动程序。

```
DCD   GPIO_A_Int_Handler_Denny   ; GPIO Int A          /* 9  */
DCD   GPIO_B_Int_Handler         ; GPIO Int B          /* 10 */
DCD   GPIO_Tmr0_Int_Handler      ; GP Timer 0          /* 11 */
DCD   GPIO_Tmr1_Int_Handler      ; GP Timer 1          /* 12 */
```

图8. 嵌套矢量中断控制器(NVIC)。

结果

速率性能

假定现在需要读取200个24位的ADC输出数据，并且SPI位速率设置为13 MHz。将DRDY信号和SCLK信号的引脚连接至示波器，可以通过观察DRDY信号与SPI数据事务处理（亦即DMA事务处理）启动之间的时间间隔的方法可以量化本文所述的每种方法对速率的改善程度。这里将DRDY信号至SCLK信号开始的时间间隔记为Δt，那么对于13 MHz的SPI速率，测量得出的Δt为：

- (a)基本模式DMA Δt = 3.754 µs
- (b)乒乓模式DMA Δt = 2.8433 µs
- (c)乒乓模式DMA（使用优化的中断处理函数）Δt = 1.694 µs

方法(a)和(b)可支持64 kHz的ODR，而方法(c)可支持128 kHz的ODR。这是因为方法(c)的Δt最短，从而使得SCLK信号能够更早结束。如果SCLK信号（即数据事务处理）能在T/2之前完成（T为当前ADC的数据输出周期），则ODR可实现翻倍。这较之于原始的ADISPI驱动程序可以达到的8 kHz的ODR性能是一次巨大的进步。

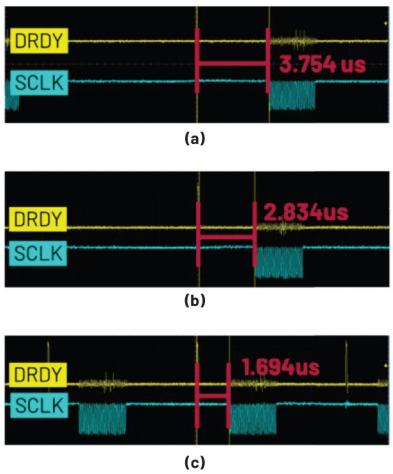

图9. (a)基本模式DMA、(b)乒乓模式以及(c)乒乓模式（使用优化的中断处理函数）的Δt。

使用MAX32660控制AD7768-1

使用主时钟速率为96 MHz的MCU MAX32660控制AD7768-1）时的结果如何？在该情况下，使用优化的中断处理函数的中断设置，可在不使用DMA函数的情况下实现256 kHz的数据输出速率。参见图10。

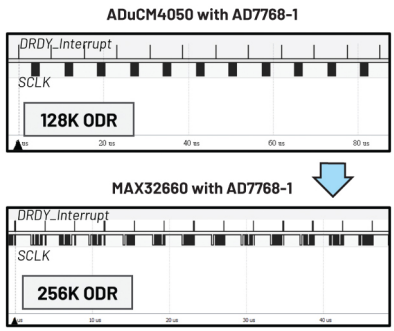

图10. 不使用DMA时MAX32660的ODR。

结论

本文利用选定的ADC(AD7768-1)和MCU（ADuCM4050或MAX32660）通过SPI实现了高速的数据事务处理。为实现速率优化的目标，本文简化了ADI的SPI驱动程序执行数据事务处理。此外本文提出，激活DMA控制器释放内核也可以加快连续数据事务处理的流程。在DMA的乒乓模式下，DMA的配置时间可通过适当的调度来节省。在此基础上，还可以通过直接指定中断引脚的方式优化中断处理函数。在13 MHz的SPI位速率下，本文提出的方案的最佳性能可达到128 kSPS的ADC ODR。

表5. 使用ADuCM405和MAX32660实现的高速SPI连接

	ADuCM4050 (MCU)				MAX32660 (MCU)
数据事务处理	未经优化的中断	基本模式DMA	乒乓模式DMA	经优化的中断	经优化的中断
总线类型	SPI	SPI	SPI	SPI	SPI
主时钟速率	26 MHz	26 MHz	26 MHz	26 MHz	96 MHz
SPI时钟速率	13 MHz	13 MHz	13 MHz	13 MHz	20 MHz
DRDY与SCLK之间的时间间隔	6.34 µs	3.754 µs	2.834 µs	1.694 µs	1.464 µs
数据输出速率	8 kSPS	32 kSPS	64 kSPS	128 kSPS	256 kSPS

致谢

在编写本文的过程中，我们获得了许多支持和帮助。

首先我们要感谢Charles Lee为我们提供了宝贵的硬件设计经验、软件支持以及调试技巧等专业知识。

我们还要感谢我们的导师William Chen给予我们的技术支持。

最后，我们要感谢Frank Chang与我们分享了自身职业生涯中的许多技术经验。

作者简介

Denny Wang是一名应用工程师，加入ADI台湾公司的NCG项目已一年半。他拥有北京大学集成电路工程硕士学位。就读期间，他从事过半导体加工、BLDC/PMSM电机控制以及模拟设计工作。他于2021年加入ADI，目前负责为系统解决方案、数据安全和认证器以及MQTT无线传输提供支持。

作者简介

Sally Tseng是ADI台湾公司的一名实习应用工程师。她是台湾大学电气工程专业的大四学生，项目主要围绕模拟电路进行研究。在ADI实习期间，她参与了嵌入式系统项目的设计。