



## 1 前言

这篇应用笔记描述了怎么使用STM32F101xx 和 STM32F103xx的直接存储器访问(DMA)控制器。STM32F101xx和STM32F103xx的DMA控制器、Cortex™-M3内核、高级微控制器总线架构(AMBA)总线和存储器系统，使得STM32具有高的数据带宽，并能使用户开发出低延迟、快响应的软件。

这篇文档也描述了怎样充分利用这些特性，以及对于不同的外设和子系统怎样保证正确的响应时间。

在下文中STM32F101xx和STM32F103xx都记作STM32F10xxx，DMA控制器都记作DMA。

译注：

本应用笔记配套例程下载地址：

<http://www.st.com/stonline/products/support/micro/files/an2548.zip>

本译文的英文版下载地址为：

<http://www.st.com/stonline/products/literature/an/13529.pdf>

# 目录

1	前言	1
2	DMA控制器	3
2.1	DMA的主要特性	3
3	性能分析	5
3.1	轮询优先级方案	5
3.2	多层结构和总线挪用	5
3.3	DMA延迟	6
3.4	数据总线带宽限制	6
3.5	通道优先级选择	7
3.5.1	应用需求	7
3.5.2	内部数据带宽	8
4	DMA编程示例	9
4.1	使用SPI传输获得ADC连续采样的数据	9
4.2	SPI直接传输实现ADC连续数据的获取	9
4.3	使用DMA实现GPIO快速数据传输	9

## 2 DMA控制器

DMA是AMBA的先进高性能总线(AHB)上的设备，它有2个AHB端口：一个是从端口，用于配置DMA，另一个是主端口，使得DMA可以在不同的从设备之间传输数据。

DMA的作用是在没有Cortex-M3核心的干预下，在后台完成数据传输。在传输数据的过程中，主处理器可以执行其它任务，只有在整个数据块传输结束后，需要处理这些数据时才会中断主处理器的操作。它可以在对系统性能产生较小影响的情况下，实现大量数据的传输。

DMA主要用来为不同的外设模块实现集中的数据缓冲存储区(通常在系统的SRAM中)。与分布式的解决方法(每个外设需要实现自己的数据存储)相比，这种解决方法无论在芯片使用面积还是功耗方面都要更胜一筹。

STM32F10XXX的DMA控制器充分利用了Cortex-M3哈佛架构和多层总线系统的优势，达到非常低的DMA数据传输延时和CPU响应中断延迟。

### 2.1 DMA的主要特性

DMA具有以下特性：

- 7个DMA通道(通道1至7)支持单向的从源端到目标端的数据传输
- 硬件DMA通道优先级和可编程的软件DMA通道优先级
- 支持存储器到存储器、存储器到外设、外设到存储器、外设到外设的数据传输(存储器可以是SRAM或者闪存)
- 能够对硬件/软件传输进行控制
- 传输时自动增加存储器和外设指针
- 可编程传输数据字长度
- 自动的总线错误管理
- 循环模式/非循环模式
- 可传输高达65536个数据字

DMA旨在为所有外设提供相对较大的数据缓冲区，这些缓冲区一般位于系统的SRAM中。

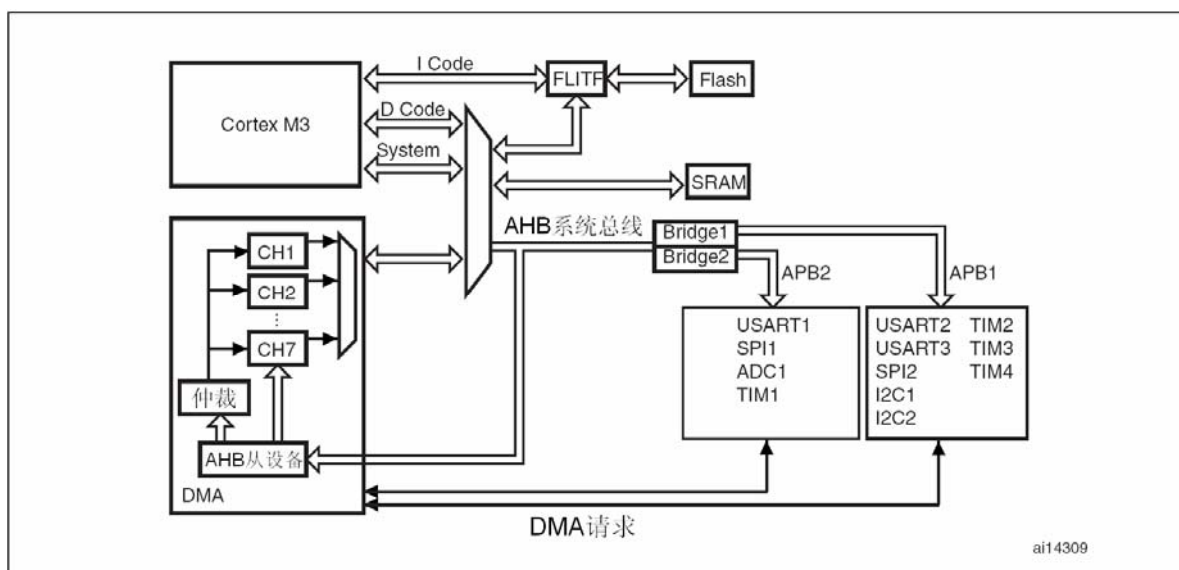
每一个通道在特定的时间里分配给唯一的外设，连接到同一个DMA通道的外设(表1中的通道1到通道7)不能够同时使用DMA功能。

支持DMA的外设如表1所示，DMA服务的外设和总线系统结构也在图1中所示。

表1 DMA服务的外设和DMA通道分配

外设模块		通道1	通道2	通道3	通道4	通道5	通道6	通道7
ADC	ADC1	ADC1						
SPI	SPI1		SPI1_RX	SPI1_TX				
	SPI2				SPI2_RX	SPI2_TX		
USART	USART1				USART1_TX	USART1_RX		
	USART2						USART2_TX	USART2_RX
	USART3		USART3_TX	USART3_RX				
I <sup>2</sup> C	I <sup>2</sup> C1						I2C1_TX	I2C1_RX
	I <sup>2</sup> C2				I2C2_TX	I2C2_RX		
TIM	TIM1		TIM1_CH1	TIM1_CH2	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
	TIM2	TIM2_CH3	TIM2_UP			TIM2_CH1		TIM2_CH2 TIM2_CH4
	TIM3		TIM3_CH3	TIM3_CH4 TIM3_UP		TIM3_CH1 TIM3_TRIG		
	TIM4	TIM4_CH1			TIM4_CH2	TIM4_CH3		TIM4_UP

图1 总线结构和支持DMA的外设



## 3 性能分析

STM32F10xxx有两个主模块——Cortex-M3处理器和DMA。他们通过总线矩阵连接到从总线、闪存总线、SRAM总线和AHB系统总线。从总线的另一端连接到两个APB总线，以服务所有的嵌入的外设(参见图1)。

总线矩阵有两个主要的特性，实现系统性能的最大化和减少延时：

- 轮询优先级方案
- 多层结构和总线挪用

### 3.1 轮询优先级方案

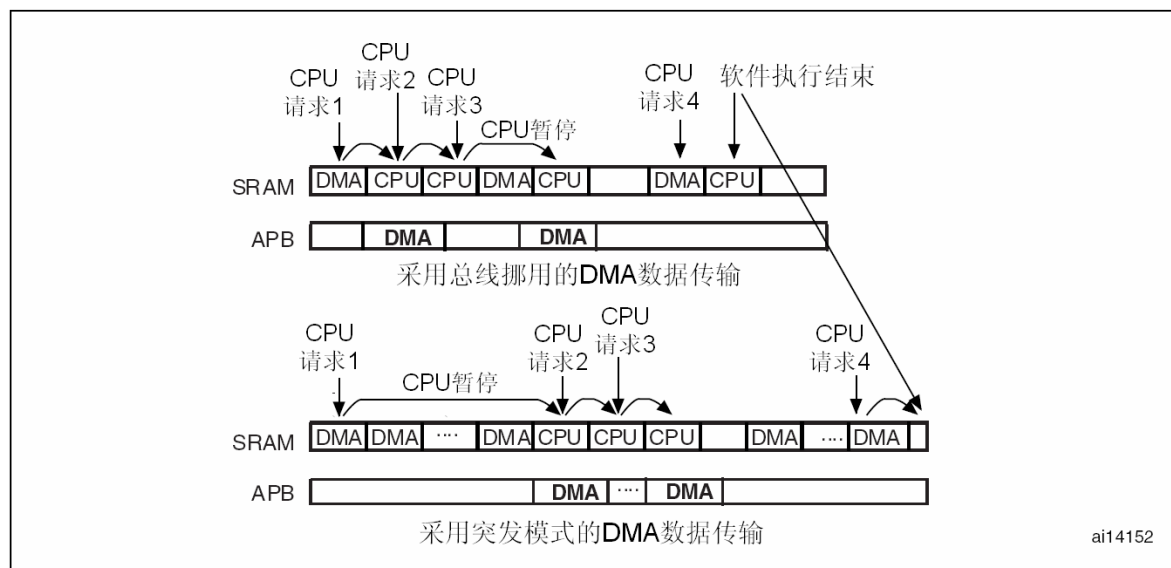
NVIC和Cortex-M3处理器实现了高性能低延时中断方案。所有的Cortex-M3指令都或者是单周期执行指令，或者可以在总线周期级上被中断。为了在系统层面上保持这个优点，DMA和总线矩阵必须确保DMA不能够长时间占用总线。轮询优先级方案能够确保，如有必要，CPU能够每两个总线周期就去访问其它从总线。因此，在CPU看来第一个数据的最大总线系统延时，就是一个总线周期(最大两个APB时钟周期)。

### 3.2 多层结构和总线挪用

多层结构允许两个主设备同时执行数据传输，只要他们寻址到不同的从模块。在Cortex-M3哈佛架构基础上，这种多层结构提高了数据的并行性，因此减少了代码执行时间并且优化了DMA效率。由于从Flash存储器取指是通过完全独立的总线，所以DMA和CPU只是在需要通过同一个从总线进行数据访问时才会产生竞争。

另外，在其它DMA控制器工作于突发模式时，STM32F10xxx的DMA数据传输只使用单个总线周期(总线挪用)。使用总线挪用存取机制时，CPU进行数据访问所等待的最大时间是很短的(一个总线周期)。通常，CPU对SRAM的访问是与DMA操作交替地进行，CPU访问SRAM的同时DMA就在通过APB总线访问外设。尽管使用DMA的突发模式可以提高(DMA访问外设)数据传输速度，但不可避免地是CPU的执行速度被拖慢。下图显示了总线挪用和突发机制的区别。

图2 DMA传输的总线挪用机制和突发机制



极端的情况发生在CPU从内存的一个地方复制一块数据到内存中的另一个地方。这种情况下，软件的执行须等到整个DMA传输完毕才能进行。实际上，CPU大部分时间是在做数据处理(寄存器的读/写)，比较少地进行数据访问，因此CPU和DMA对数据的存取还算是交替进行着。

STM32F10xxx总线结构固有的并行性，加上DMA总线挪用机制，保证了CPU不会长时间地等待从SRAM中读取数据。采用总线挪用机制的DMA因此能够更高效地使用总线，从而显著地减少了软件执行的时间。

### 3.3 DMA延迟

DMA完成从外设到SRAM存储器的数据传输有三个步骤：

1. DMA请求仲裁
2. 从外设中读取数据(DMA源)
3. 将读取的数据写入到SRAM中(DMA目标)

当DMA把数据从内存中传输到外设(例如SPI传送)，操作步骤如下：

1. DMA请求仲裁
2. 从SRAM中读取数据(DMA源)
3. 将读取到的数据通过APB总线写入到外设中(DMA目标)

服务每个DMA通道的总时间，

$$t_S = t_A + t_{ACC} + t_{SRAM}$$

这里，

$t_A$ 是仲裁时间

$$t_A = 1 \text{ 个AHB时钟周期}$$

$t_{ACC}$ 是访问外设时间

$$\begin{aligned} t_{ACC} = & 1 \text{ 个AHB时钟周期(总线矩阵仲裁)} \\ & + 2 \text{ 个APB时钟周期(实际的数据传输)} \\ & + 1 \text{ 个AHB时钟周期(总线同步)} \end{aligned}$$

$t_{SRAM}$ 是读写SRAM的时间

$$\begin{aligned} t_{SRAM} = & 1 \text{ 个AHB时钟周期(总线矩阵仲裁)} \\ & + 1 \text{ 个AHB时钟周期(单一的读/写操作)} \\ & \text{或者} + 2 \text{ 个AHB时钟周期(先读SRAM再写SRAM的情况)} \end{aligned}$$

当DMA通道空闲或者是前一个DMA通道的第3步操作完成后，DMA控制器比较所有挂起的DMA请求的优先级(先比较软件优先级；软件优先级相同时，再比较硬件优先级)，高优先级的通道将会被服务，DMA开始执行第2步操作。当一个通道正在服务时(第2、3步操作正在进行)，没有其他的通道能够被服务，不管它的优先级如何。

当至少同时使能了两个DMA通道时，最高优先级通道的DMA延迟时间为正在传输的时间(不包括仲裁阶段)，加上下个将被服务的DMA通道(挂起优先级最高的通道)数据传输的时间。

### 3.4 数据总线带宽限制

数据总线带宽限制主要是因为APB总线比系统SRAM和AHB总线速度慢。对于最高优先级的DMA通道，必须考虑以下两种情况：(参见图3)

1. 当不止一个DMA通道被使能时，最高优先级的通道在APB总线上占用的数据带宽必须低于APB最高传输率的25%。APB总线传输的所有时间必须考虑在内，即2个APB时钟周期加上用来仲裁/同步的2个AHB时钟周期。
2. 尽管高速/高优先级DMA传输通常发生在APB2上(更快的APB总线)，但是CPU和其他DMA通道可以访问APB1上的外设。大约3/4的APB传输是在APB1上完成的，最小的APB2频率依赖于最快的DMA通道数据带宽。

最大的APB时钟分频因子由下列的等式给出：

$$f_{\text{AHB}} > (2 \times N_2 + 6 \times N_1 + 6) \times B_{\text{max}}$$

如果

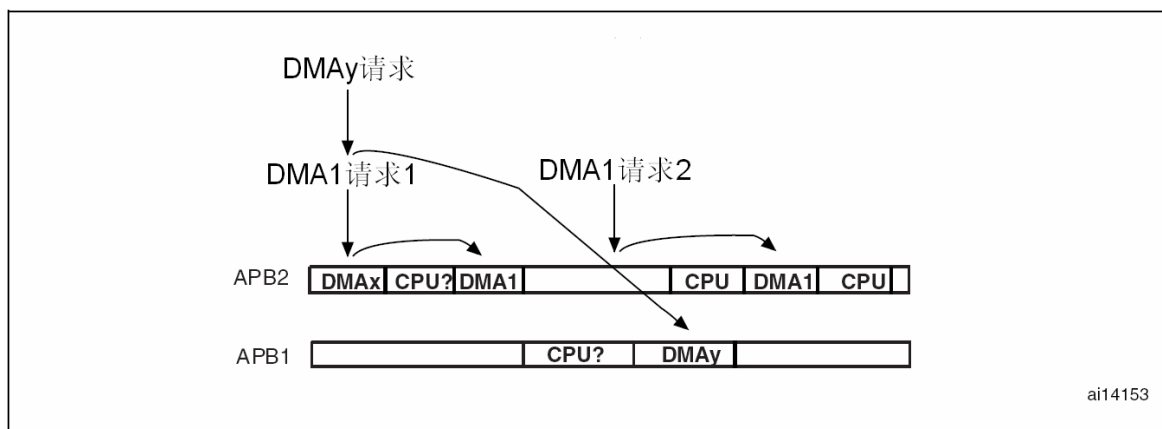
$$N_2 < N_1 \text{ 则 } N_1 < (f_{\text{AHB}} / B_{\text{max}}) / 8$$

其中 $f_{\text{AHB}}$ 是AHB时钟频率

$N_1$ 和 $N_2$ 分别是APB1和APB2的时钟分频因子

$B_{\text{max}}$ 是APB2上的最大数据带宽，单位为传输次数/秒。

图3 DMA传输过程中APB总线的占用情况



注：DMA1是最高优先级通道

## 3.5 通道优先级选择

为了实现外设数据的连续传输，相关的DMA通道必须能够维持外设数据传输率，确保DMA服务的延迟时间少于连续两个外设数据的时间间隔。

高速/高带宽外设必须拥有最高的DMA优先级，这确保了最大的数据延迟对于这些外设都是可以忍受的，而且可以避免溢出和下溢的情况。

在相同带宽需求的情况下，推荐给工作在从模式下(不能对数据传输速度进行控制)的外设分配较高的优先级，工作在主模式(能够控制数据流)下的外设分配相对低的优先级。

默认情况下，通道和硬件优先级(从1到7)的分配，是按照最快的外设分配最高优先级的顺序来分配的。当然，在某些运用场合下也许这种分配并不适用；此时，用户可以为每一个通道配置软件优先级(分4种，从非常高到低)，软件优先级优先于硬件优先级。

当同时使用几个外设(不管有没有使用DMA)时，用户必须确保内部系统能够维持应用所要求的总数据带宽，必须权衡以下两个因素，找到一个折中方案：

- 每个外设的应用需求
- 内部数据带宽

### 3.5.1 应用需求

以SPI接口为例，SPI接口数据带宽是通过波特率除以SPI的数据字长度而得到的(因为数据是一个紧接着下一个传输的)。假设SPI的波特率是18Mbps，数据是以8位传输的，操作配置在单工模式下，因此，内部数据带宽需求是2.25M传输/秒；如果SPI配置为16位模式，则数据带宽将是1.125M传输/秒。

**注意：** 当使用SPI的16位模式时，同样波特率下，数据带宽除以2，即只需要1.125M/秒的传输。  
强烈推荐，尽可能地使用16位模式，以减少总线占用和功耗。

### 3.5.2 内部数据带宽

内部数据带宽依赖于以下两个条件：

- 总线频率
  - 可获得的数据带宽与总线时钟频率是成正比的
- 总线类型
  - **AHB** 数据传输需要 2 个时钟周期(**SRAM** 先写后读访问需要 3 个周期)。数据通过 **APB** 总线传输给外设需要花费 2 个 **APB** 时钟周期加上两个 **AHB** 时钟周期用来做总线仲裁和数据同步。

推荐**DMA**对总线的占用保持在2/3以下，这样才能保证一个合理的系统和**CPU**的性能水平。



## 4 DMA编程示例

所有的示例都使用STM32F10xxx固件库，可以参考与该应用手册相关的固件。这些都可以在ST网站上下载，网址为[www.st.com](http://www.st.com)。

### 4.1 使用SPI传输获得ADC连续采样的数据

ADC配置为连续转换模式，该模式下，它以最大速度对一个输入通道进行连续转换。AHB总线频率设置为56MHz，ADC预分频为4，采样时间为13.5个ADC时钟周期。通过DMA通道1传输采样值到位于系统RAM中的缓冲区，则通道1的数据带宽为每秒0.54M次传输。

当ADC转换后的数据填充了缓冲区的一半后，软件计算出最大值并且对其数字归一化(最大值设置为0xFF)，ADC转换的结果通过SPI接口传输到外部。

ADC转换结果是通过SPI1接口来传输的，使用DMA通道3把数据从SRAM缓冲区中传输到SPI1的数据寄存器。要达到最大的DMA传输速度0.875M/秒，SPI1接口配置为16位 主传输模式和14Mbps的传输速度。

可是，当SPI1工作在主模式，SPI1有效的数据传输速度受数据的可用速率1M/S的限制时，优先级的配置如下：

- 通道1(ADC)：非常高(VeryHigh)
- 通道3(SPI1\_TX)：高(High)

### 4.2 SPI直接传输实现ADC连续数据的获取

这个例子完成的功能和前面一个几乎相同，只是没有数据的标准化。CPU内部并没有处理这些数据，直接把ADC转换结果传输到SPI数据寄存器可以减少一半的总线占用。

因此，仅使用DMA 通道1，该通道的目标存储器地址设置为SPI数据寄存器，而不需要SRAM缓冲。

### 4.3 使用DMA实现GPIO快速数据传输

这个例子示范了如何将不同的外设用于DMA请求和数据传输，这个机制允许在没有使用CPU的情况下实现简单的快速并行同步接口。

定时器3和连接到TIM3\_TRIG 的DMA通道6，用来实现获取数据的接口。在GPIO的端口上可以获取16位并行数据。一个外部时钟信号作用在定时器3的外部触发器输入端，在外部触发器上升沿，定时器产生一个DMA请求。由于GPIO数据寄存器地址已设置到DMA通道6的外设地址，DMA控制器在每一次DMA请求时从GPIO端口读取数据，并把它存储到SRAM的缓冲器中。