

## FIFO单采模式

FIFO单模式使用车载存储器作为FIFO缓冲区进行连续数据采集，并将数据连续传输到PC内存。人们可以用获取的数据进行在线计算，将数据连续存储到磁盘以供以后使用，甚至可以使用带有在线数据显示的数据记录器功能。

### 卡模式

必须将卡模式设置为正确模式SPC\_REC\_FIFO\_SINGLE。

登记册	价值	方向	说明
SPC_CARDMODE	9500	读/写	定义使用的操作模式，读取命令将返回当前使用的模式。
SPC_REC_FIFO_SINGLE	10h		连续数据采集到PC内存...完整的车载内存用作FIFO缓冲区...

### 长度和预触发器

即使在FIFO模式下，也可以编程预触发器区域。一般来说，FIFO模式可以永远运行，直到它被显式用户命令停止，或者一个人可以通过两个计数器循环和段大小来编程传输的总长度

登记册	价值	方向	说明
SPC_PRETRIGGER	10030	读/写	程序在触发事件检测之前要获取的样本数量
SPC_Segmentize	10010	读/写	要获取的段的长度。
SPC_LOOPS	10020	读/写	总的要获取的段数。如果设置为零，FIFO模式将连续运行，直到用户停止。

每个信道获取的样本总量可由[SPC\_LOOPS\*SPC\_Segments]计算。请坚持下面提到的寄存器的限制。

### 与标准单采模式的差异

标准模式和FIFO模式与编程方面差别不大。事实上，人们甚至可以使用FIFO模式来获得与标准模式相同的行为。下一章显示的缓冲区处理对于两种模式都是相同的。

#### 预触发器

当做标准单采集存储器时，作为循环缓冲区，预触发器可以达到[已安装的内存]-[最小后触发器]。与此相比，FIFO模式下的预触发器受到特殊的预触发器FIFO的限制，因此大大缩短。

#### 购置时间

在标准模式下，采集长度是在开始之前定义的，并且仅限于安装的车载内存，而在FIFO模式下，采集长度可以被定义，也可以连续运行，直到用户停止它。

### 示例FIFO获取

下面的示例显示了一个简单的FIFO单模数据采集设置，然后读取数据。为了使此示例保持简单，没有实现错误检查。

```

spcm_dw Set Param_i32 ( hDrv , SPC_CHENABLE , Channel0 ) ; /只激活一个通道
spcm_dw Set Param_i32 ( hDrv , SPC_CARDMODE , SPC_REC_FIFO_SINGLE ) ; /设置FIFO单记录模式
spcm_dw Set Param_i64 ( hDrv , SPC_PRETRIGGER , 1024 ) ; /触发前1k数据样本

//在FIFO模式下，我们需要在开始传输之前定义缓冲区
void*pvData=pv Alloc Mem Page对齐 ( llBufsize in samples*2 ) ; /每个样本2字节
spcm_dw Def Transfer_i64 ( hDrv , SPCM_BUF_DATA , SPCM_DIR_CAR
DTOPC , 4096 ,
pvData , 0 , 2 * llBufsizeInSamples ) ;

//现在我们开始收购，等待第一个区块
dw Error=spcm_dw Set Param_i32 ( hDrv , SPC_M2CMD , M2CMD_CARD_START_START_M2CMD_CARD_ENABLE_TRIGGER ) ; dw Error=spcm_dw Set Param
_i32 ( hDrv , SPC_M2CMD , M2CMD_DATA_STAR TDMA_M2CMD_DATA_DATWA DMA ) ;

//我们在循环中获取数据.由于我们定义了4k的通知大小，我们将得到>=4kchuncks|| Total Bytes=0中的数据；
同时 ( ! dw错误 )
{
    spcm_dw Get Param_i64 ( hDrv , SPC_DATA_AVAIL_USER_LEN , &ll Avail Bytes ) ; /读出可用的字节|| Total Bytes|| Avail Bytes ;

    //这里有正确的位来处理数据 ( printf仅限于32位变量 ) printf ( “当前可用的：%lld，总数：%lld\n” , ll Avail 字节 , ll Total 字节 )
;

    //现在我们释放字节数，等待下一个缓冲区
    spcm_dw Set Param_i64 ( hDrv , SPC_DATA_AVAIL_CARD_LEN , ll Av
albytes ) ; dw Error=spcm_dw Set Param_i32 ( hDrv , SPC_M2CMD , M2CMD_DATA_WAITDMA ) ;
}

}

```

## 前置触发器、后置触发器、内存大小的限制

最大内存大小参数仅受激活通道数量和已安装内存数量的限制。请记住，每个样本需要2个字节的内存来存储。最小内存大小以及最小和最大后触发限制与激活的通道或安装的内存无关。

由于卡存储器的内部组织，在设置这些必须考虑的值时有一定的步骤。下表概述了有关预触发器、POST触发器、内存大小、段大小和循环的所有限制。该表显示了与示例中已安装的内存大小有关的所有值。如果安装了更多的内存，则最大内存大小将根据已安装的完整内存而增加。

激活通道	使用模式	内存大小SPC_M_EMSIZE			预触发器SPC_PRETRIGGER			后置触发器SPC_POSTTRIGGER			标段大小SPC_Segmentize			环路SPC_LOOPS		
		Min	Max	Step	Min	Max	Step	Min	Max	Step	Min	Max	Step	Min	Max	Step
1 Ch	标准单人	32	Mem	16	16	Mem - 16 (由mem和post定义)	16	16	8G - 16	16	未使用			未使用		
	标准多种/ABA	32	Mem	16	16	8k (按部门和员额界定)	16	16	Mem/2-16	16	32	Mem/2	16	未使用		
	标准门标准平均数	32	Mem	16	16	8k	16	16	Mem-16	16	未使用			未使用		
	FIFO Single	未使用			16	8k	16	未使用			32	8G - 16	16	0 (∞)	4G - 1	1
	FIFO Multi/ABA	未使用			16	8k (按部门和员额界定)	16	16	8G - 16	16	32	职前+员额	16	0 (∞)	4G - 1	1
	FIFO门	未使用			16	8k	16	16	8G - 16	16	未使用			0 (∞)	4G - 1	1
	FIFO平均值	有关此模式的限制，请参阅本手册中的专用章节。														
	标准单人	32	Mem/2	16	16	Mem/2 - 16 (由mem和post定义)	16	16	8G - 16	16	未使用			未使用		
	标准多种/ABA	32	Mem/2	16	16	8k (按部门和员额界定)	16	16	Mem/4-16	16	32	Mem/4	16	未使用		
2 Ch	标准门标准平均数	32	Mem/2	16	16	8k	16	16	Mem/2-16	16	未使用			未使用		
	FIFO Single	未使用			16	8k	16	未使用			32	8G - 16	16	0 (∞)	4G - 1	1
	FIFO Multi/ABA	未使用			16	8k (按部门和员额界定)	16	16	8G - 16	16	32	职前+员额	16	0 (∞)	4G - 1	1
	FIFO门	未使用			16	8k	16	16	8G - 16	16	未使用			0 (∞)	4G - 1	1
	FIFO平均值	有关此模式的限制，请参阅本手册中的专用章节。														
	标准单人	32	Mem/4	16	16	Mem/4 - 16 (由mem和post定义)	16	16	8G - 16	16	未使用			未使用		
	标准多种/ABA	32	Mem/4	16	16	8k (按部门和员额界定)	16	16	Mem/8-16	16	32	Mem/8	16	未使用		
	标准门标准平均数	32	Mem/4	16	16	8k	16	16	Mem/4-16	16	未使用			未使用		
	FIFO Single	未使用			16	8k	16	未使用			32	8G - 16	16	0 (∞)	4G - 1	1
	FIFO Multi/ABA	未使用			16	8k (按部门和员额界定)	16	16	8G - 16	16	32	职前+员额	16	0 (∞)	4G - 1	1
	FIFO门	未使用			16	8k	16	16	8G - 16	16	未使用			0 (∞)	4G - 1	1
	FIFO平均值	有关此模式的限制，请参阅本手册中的专用章节。														

这里列出的所有数字都是在样本中给出的。输入[8G-16]表示[8GSamples-16]=8,589, 934, 576个样本。

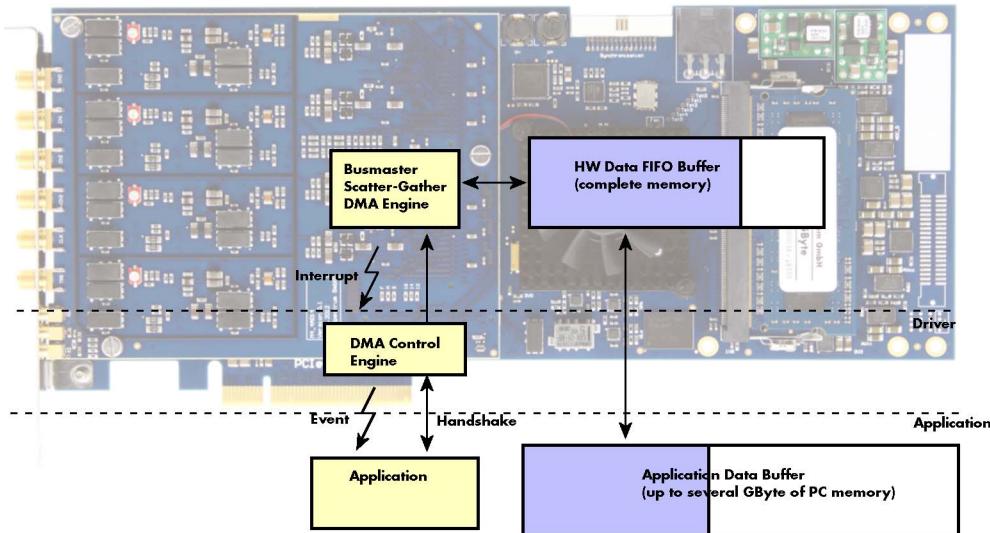
给定的内存和内存/分配器数字取决于安装的车载内存，如下所示：

	已安装的内存
2	
Mem	2丛书1丛书5
Mem / 2	1丛书256丛书
Mem / 4	
Mem / 8	

请记住，这张表同时显示了所有的值。只显示绝对最大值和最小值。可能会有额外的限制。这些值中的哪一个是编程的，取决于使用的模式。请阅读模式的详细文档。

## 缓冲区处理

为了处理M4i/M4x/M2p系列卡可能获得的大量数据，有一个非常可靠的两步缓冲策略。卡的车载存储器完全可以作为一个真正的FIFO缓冲区。此外，PC内存的一部分可以用作附加的软件缓冲区。硬件FIFO和软件缓冲器之间的传输由驱动程序驱动并自动执行中断，以获得最佳性能。下图将为您概述数据传输处理的结构：



虽然这里显示了M4i，但这也适用于M4x和M2p卡。在驱动程序中实现了数据缓冲握手，允许在不同的数据传输模式下运行卡。软件传输缓冲区作为一个大的缓冲区处理，该缓冲区由驱动程序控制，并由总线主机DMA从/到硬件FIFO缓冲区自动填充，另一方面由设置此软件缓冲区的部分以供驱动程序进一步传输的用户处理。握手用以下3个软件寄存器完成：

登记册	价值	方向	说明
SPC_DATA_AVAIL_USER_LEN	200	念	返回样本数据传输中用户可用字节数。
SPC_DATA_AVAIL_USER_POS	201	念	返回当前可用数据示例开始的位置为字节索引。
SPC_DATA_AVAIL_CARD_LEN	202	write	写入卡现在可以再次用于样本数据传输的字节数

在内部，卡片处理两个计数器，一个用户计数器和一个卡片计数器。根据传输方向，软件寄存器的含义略有不同：

转移方向	登记册	方向	说明
写到卡片上	SPC_DATA_AVAIL_USER_LEN	念	此寄存器包含当前可用的字节数，这些字节数可以自由地将新数据写入卡。用户现在可以用要传输的新数据填充此字节数。
	SPC_DATA_AVAIL_CARD_LEN	write	在将缓冲区的一个量填入要传输到卡的新数据之后，用户用这个寄存器告诉驱动程序，数据量现在已经准备好传输了。
从卡片上读	SPC_DATA_AVAIL_USER_LEN	念	此寄存器包含当前可用的字节数，这些字节数由新传输的数据填充。用户现在可以为自己的目的使用这些数据，复制它，将其写入磁盘或用这些数据开始计算。
	SPC_DATA_AVAIL_CARD_LEN	write	在使用新的可用数据完成工作之后，用户需要告诉驱动程序，此字节数对于要传输的新数据再次是免费的。
任何方向	SPC_DATA_AVAIL_USER_POS	念	寄存器保持可用字节开始的当前字节索引位置。登记册只是为了帮助你，避免自己的位置计算
任何方向	SPC_FLSIZE PROMI LLE	念	寄存器将车载存储器（FIFO缓冲区）的当前填充大小保存在完整的车载存储器的Promile（千分之一）中。请注意，硬件只在完整内存的1/16部分报告填充大小。因此，报告的填充大小仅以1000/16=63个步骤显示。

在开始传输之后，由于用户和  
SPC\_DATA\_AVAIL\_CARD\_LEN每次都与所定义的缓冲区的长度相同，因为完整的缓冲区可用于卡的传输。

保持用户缓冲区可用字节（SPC\_DATA\_AVAIL\_USER\_LEN）的计数器与Def传输调用时的通知大小有关。即使已经传输了较少的字节，你也不会注意到它，以防通知大小被编程到更高的值。



### 备注

硬件FIFO缓冲区和应用程序缓冲区之间的传输是使用总线型DMA控制器用散射集DMA完成的。位于卡片上。即使PC忙于其他作业数据的传输，直到应用程序数据缓冲区完全使用。即使应用程序数据缓冲区完全使用，仍然有硬件FIFO缓冲区，可以保存数据，直到完整的车载。

内存被使用了。因此，一个更大的车载内存将使传输更可靠的任何PC死机时间。

如您在上面的图片中所看到的，数据是直接在应用程序数据缓冲区和车载内存之间传输的。因此，在不停止实际运行的任何DMA传输的情况下，删除应用程序数据缓冲区是绝对关键的。它也是绝对的criti-

将应用程序数据缓冲区定义为DMA可以不匹配的长度，而不是尝试访问应用程序数据区域之外的内存。

如上图所示，DMA控件将通过发送事件向应用程序发布新数据。等待事件的是

如果应用程序调用其中一个等待函数，则在驱动程序内部完成。等待事件不会消耗任何CPU时间，因此，如果其他线程做了大量的计算工作，这是非常可取的。然而，没有必要使用等待函数，只要程序有时间，就可以简单地请求当前状态。当使用此轮询模式时，宣布的可用字节仍然保持在定义的通知大小上！

如果车载FIFO缓冲区有溢出（卡到PC）或欠运行（PC到卡）数据传输被停止。但是，如果发生转移

从卡到PC机，车载内存中仍然有很多数据..因此，数据传输将继续进行，直到所有数据都已传输，尽管状态信息已经显示超支。

获得最佳的总线传输性能是使用“连续缓冲区”完成的。附录中对这一模式作了更详细的解释。

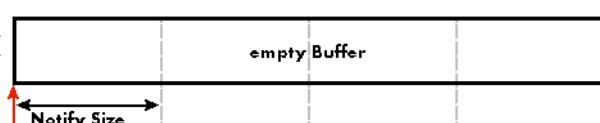
通知大小与由PC硬件和操作系统定义的页面大小一致。因此，通知大小必须是4k字节的倍数。对于数据传输，在16、32、64、128、256、512、1k或2k范围内也可能是4k的一小部分。不允许有其他值..对于ABA和时间戳通知

大小可以是2k作为最小。如果您需要在较小的块中使用ABA或时间戳数据，请使用后面描述的轮询模式。

下面的图表将显示在传输的不同状态下的当前缓冲区位置。图纸已制作完成，用于从卡到PC的转移..然而，对于相反的方向，所有块处理都是相似的，只是缓冲区的空部和填充部分是倒过来的。

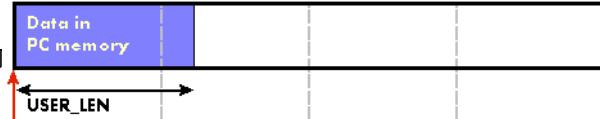
#### 步骤1：缓冲区定义

在缓冲区定义之后，完整的缓冲区是空的（卡到PC）或完全填充的（PC到卡）。在我们的示例中，我们有一个通知大小，它是完整缓冲存储器的1/4，以保持示例的简单性。在现实世界中，建议将通知大小设置为较小的步骤。



#### 第二步：提供起始数据和第一个数据

在此之间，我们已经开始传输，并等待第一个数据可供用户使用。当内存中至少有一个通知大小的块时，我们会得到一个中断，并且可以继续处理数据。任何已经被转移的数据都会被宣布。USER\_POS仍然是零，因为我们在完全转移的开始是正确的。



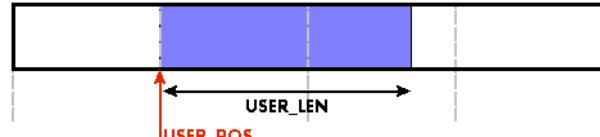
#### 第三步：设置卡可用的第一个数据

现在数据可以处理了。如果传输是从卡到PC，可能存储到硬盘或计算任何数字。如果传输是从PC到卡，这意味着我们必须再次填充可用的缓冲区的数据。在用户应用程序处理的数据量之后，我们设置它可用于卡和下一步。



#### 第四步：下一个可用数据

在到达通知大小的下一个边框后，我们得到数据缓冲区的下一部分可用。在我们的例子中，在读取USER\_LEN时，甚至更多的数据已经可用。用户位置现在将处于前一组CARD\_LEN的位置。



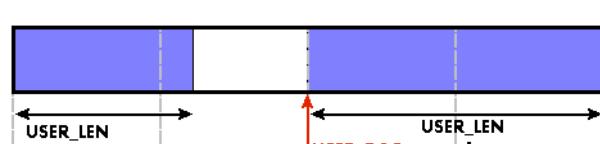
#### 第五步：再次设置可用的数据

再次，在处理数据后，我们将其免费设置为卡使用。在我们的例子中，我们现在做了一些其他的事情，并且在更长的时间内不对中断做出反应。在后台，缓冲区中充满了更多的数据。



#### 第六步：滚动缓冲区结束

现在几乎填满了完整的缓冲区。请记住，我们当前的用户位置仍然在我们在步骤4和步骤5中处理和标记的数据部分的末尾。因此，现在要处理的数据分为两部分。第1部分在缓冲区的末尾，而第2部分从地址0开始。



#### 步骤7：设置可用缓冲区的其余部分

可以随意处理完整的数据，或者只处理第1部分，直到缓冲区结束，就像我们在这个例子中所做的那样。如果您决定处理完整的缓冲区，请记住在缓冲区末尾的滚动。



这种缓冲区处理现在可以继续下去，只要我们设法

将卡可用的数据设置得足够快。步骤8的USER\_POS和USER\_LEN现在将完全像步骤2中显示的缓冲区一样。

缓冲区处理示例，用于从卡到PC的传输

```

int8*pc数据 = ( int8* ) pv Alloc mem Page对齐 ( || Buffersize in Bytes ) ;

//我们现在用页=4K字节的最小通知大小来定义传输缓冲区
spcm_dwDefTransfer_i64 ( hDrv, SPCM_BUF_DATA, SPCM_DIR_CARDTOPC , 4096, ( void* ) pcData, 0, llBufferSizeInBytes);

//我们启动DMA传输
dwError = spcm_dwSetParam_i32 ( hDrv, SPC_M2CMD, M2CMD_DATA_STARTDMA );

do
{
    如果 ( ! dw错误 )
    {
        //我们等待下一个数据可用. 通过这个调用，我们得到至少 4k的数据来进行 dwError=spcm_dw Set Param_i32 ( h Drv , SPC_M2CMD , M2CMD _DATA_WAITDMA ) ;

        //如果没有错误，我们可以继续读出可用的字节，这些字节又是 spcm_dwGet Param_i64 ( h Drv , SPC_DATA_AVAIL_USER_LEN , &ll AvailBytes ) ;

        spcm_dwGetParam_i64 ( hDrv, SPC_DATA_AVAIL_USER_POS, &llBytePos);

        打印f ( “我们现在有 %lld 新字节可用\n ” , llAvailBytes ) ; 打印f ( “ 可用数据从位置 %lld\n ” , llBytes Pos开始 ) ;

        //如果 ( llByte Pos+llAvailBytes>=ll缓冲区大小以字节为单位 ) llAvailBytes=ll缓冲区大
        小以字节为单位，我们注意不要穿过缓冲区的末尾；

        //我们的 do函数得到一个指针，指向可用数据部分的开始和长度 v Dosomething ( &pc Data[ll Bytes Pos] , ll AvailBytes ) ；

        //现在可以立即为卡 spcm_dw SetParam_i64 ( h Drv , SPC_DATA_AVAIL_CARD_LEN , ll Avalbytes ) 设置缓冲区部分 ；

    }
}
如果没有错误发生，我们将永远循环

```

缓冲区处理示例，用于从PC传输到卡

```

int8*pc数据 = ( int8* ) pv Alloc mem Page对齐 ( || Buffersize in Bytes ) ;

//在开始传输之前，我们先用数据vDo生成数据 ( &pc数据 [0] , ll缓冲区大小以字节为单位 ) 填充完整的
缓冲区内存 ；

//我们现在用页=4K字节的最小通知大小来定义传输缓冲区
spcm_dwDefTransfer_i64 ( hDrv, SPCM_BUF_DATA, SPCM_DIR_PCTOCARD , 4096, ( void* ) pcData, 0, llBufferSizeInBytes);

//在开始之前，我们必须填写一些数据，以便开始输出 spcm_dw Set Param_i32 ( h Drv , SPC_DATA_AVAIL_CARD_LEN , ll
缓冲区大小，以字节为单位 ) ；
dwError = spcm_dwSetParam_i32 ( hDrv, SPC_M2CMD, M2CMD_DATA_STARTDMA | M2CMD_DATA_WAITDMA);

do
{
    如果 ( ! dw错误 )
    {
        //如果没有错误，我们可以继续读出当前可用数据的数量 spcm_dwGet Param_i64 ( h Drv , SPC_DATA_AVAIL_USER_LEN , &ll Availbytes ) ；

        spcm_dwGetParam_i64 ( hDrv, SPC_DATA_AVAIL_USER_POS, &llBytePos);

        打印f ( “我们现在有 %lld 空闲字节可用\n ” , llAvailBytes ) ; 打印f ( “ 可用数据从位置 %lld\n ” ,
        llBytes Pos开始 ) ；

        //如果 ( llByte Pos+llAvailBytes>=ll缓冲区大小以字节为单位 ) llAvailBytes=ll缓冲区大
        小以字节为单位，我们注意不要穿过缓冲区的末尾；

        //我们的 do函数得到一个指针，指向可用数据部分的开始和长度 v do生成数据 ( &pc数据 [ll Bytes Pos] , ll AvailBytes ) ；

        //现在我们标记我们刚刚为重播生成的字节数，并等待下一个空闲缓冲区

        spcm_dw Set Param_i64 ( h Drv , SPC_DATA_AVAIL_CARD_LEN , ll Avalbytes ) ; dw Error=spcm_dw Set Param_i32 ( h Drv , SP
        C_M2CMD , M2CMD_DATA_WAITDMA ) ；

    }
}
如果没有错误发生，我们将永远循环

```

请记住，您正在使用连续的缓冲区写入/读取，如果达到缓冲区长度，将在零位置重新开始。然而，DATA\_AVAIL\_US\_ER\_LEN 寄存器将给您提供完整的可用字节，即使空闲区域的一部分在缓冲区的末尾，而下半部分在缓冲区的开头。



## 数据组织

数据在传输缓冲区中以多路复用的方式组织。如果首先使用第一激活信道的2个信道数据，然后使用第二信道的数据。

激活通道	Ch0	Ch1	Ch2	Ch3	从数据偏移零开始在缓冲存储器中排序的示例																
1频道1频 道1频道	X	X	X	X	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
					B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16
					C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
					D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15	D16
2频道2频 道2频道2 频道	X	X	X	X	A0	B0	A1	B1	A2	B2	A3	B3	A4	B4	A5	B5	A6	B6	A7	B7	A8
	X				A0	C0	A1	C1	A2	C2	A3	C3	A4	C4	A5	C5	A6	C6	A7	C7	A8
	X	X	X	X	A0	D0	A1	D1	A2	D2	A3	D3	A4	D4	A5	D5	A6	D6	A7	D7	A8
	X	X	X	X	B0	C0	B1	C1	B2	C2	B3	C3	B4	C4	B5	C5	B6	C6	B7	C7	B8
					B0	D0	B1	D1	B2	D2	B3	D3	B4	D4	B5	D5	B6	D6	B7	D7	B8
					C0	D0	C1	D1	C2	D2	C3	D3	C4	D4	C5	D5	C6	D6	C7	D7	C8
4个频道	X	X	X	X	A0	B0	C0	D0	A1	B1	C1	D1	A2	B2	C2	D2	A3	B3	C3	D3	A4

为了更好的可读性，这些样本被重新命名。其中A0为通道0的样本0，B4为通道1的样本4，以此类推..

## 示例格式

如果卡使用14位A/D样本，则将它们存储在16位数据字的较低14位中的两个补码中。14位分辨率意味着数据范围从-8192...到...8191。在标准模式下，上两位包含允许直接使用读取数据作为16位整数值的符号扩展。如果卡使用16位A/D样本，则将它们存储在16位数据字中的两个补码中。16位分辨率意味着数据从-32768...到...32767：

数据位	标准模式		启用数字输入				启用数字输入			
	M4i.445x, M4i.448x	M4i.441x, M4i.442x, M4i.447x	M4i.445x, M4i.448x	M4i.441x, M4i.442x, M4i.447x	M4i.445x, M4i.448x	M4i.441x, M4i.442x, M4i.447x				
14位 ADC分辨率	14位 ADC分辨率	16位 ADC分辨率	14位 ADC分辨率	15位 ADC分辨率	14位 ADC分辨率	14位 ADC分辨率	14位 ADC分辨率	14位 ADC分辨率	14位 ADC分辨率	
D15	ADx Bit 13 (展期标志)	ADx Bit 15 (MSB)	44x1 (4CH) 型号：C H3：数字位0 (X0) CH2：数字位2 (X2) CH1：数字位1 (X1) CH0：数字位0 (X0)	44x1 (4CH) 型号： CH3：数字位0 (X0) CH2：数字位2 (X2) CH1：数字位1 (X1) CH0：数字位0 (X0)	44x1 (4CH) 型号： CH3：数字位0 (X0) CH2：数字位2 (X2) CH1：数字位1 (X1) CH0：数字位0 (X0)	44x1 (4CH) 型号： CH3：数字位0 (X0) CH2：数字位2 (X2) CH1：数字位1 (X1) CH0：数字位0 (X0)	44x0 (2CH) 型号： CH1：数字位1 (X2) CH0： 数字位0 (X0)	44x0 (2CH) 型号： CH1：数字位1 (X2) CH0： 数字位0 (X0)	44x0 (2CH) 型号： CH1：数字位1 (X2) CH0： 数字位0 (X0)	
D14	ADx Bit 13 (展期标志)	ADx Bit 14	ADx Bit 13 (展期标志)	ADx Bit 15 (MSB)	ADx Bit 13 (MSB)	ADx Bit 14	ADx Bit 13 (MSB)	ADx Bit 15 (MSB)	ADx Bit 14	
D13	ADx Bit 13 (MSB)	ADx Bit 13	ADx Bit 13 (MSB)	ADx Bit 14	ADx Bit 13 (MSB)	ADx Bit 14	ADx Bit 13 (MSB)	ADx Bit 15 (MSB)	ADx Bit 14	
D12	ADx位12	ADx位12	ADx位12	ADx位13	ADx位12	ADx位13	ADx位12	ADx位13	ADx位14	
D11	ADx Bit 11	ADx Bit 11	ADx Bit 11	ADx位12	ADx Bit 11	ADx位12	ADx Bit 11	ADx位13	ADx位13	
D10	ADx位10	ADx位10	ADx位10	ADx位11	ADx位10	ADx位11	ADx位10	ADx位12	ADx位12	
D9	ADx Bit 9	ADx Bit 9	ADx Bit 9	ADx位10	ADx Bit 9	ADx位10	ADx Bit 9	ADx位11	ADx位11	
D8	ADx Bit 8	ADx Bit 8	ADx Bit 8	ADx Bit 9	ADx Bit 8	ADx Bit 9	ADx Bit 8	ADx位10	ADx位10	
D7	ADx Bit 7	ADx Bit 7	ADx Bit 7	ADx Bit 8	ADx Bit 7	ADx Bit 8	ADx Bit 7	ADx位9	ADx位9	
D6	ADx位6	ADx位6	ADx位6	ADx Bit 7	ADx位6	ADx Bit 7	ADx位6	ADx Bit 8	ADx位8	
D5	ADx位5	ADx位5	ADx位5	ADx位6	ADx位5	ADx位6	ADx位5	ADx Bit 7	ADx位7	
D4	ADx Bit 4	ADx Bit 4	ADx Bit 4	ADx位5	ADx Bit 4	ADx位5	ADx Bit 4	ADx位6	ADx位6	
D3	ADx Bit 3	ADx Bit 3	ADx Bit 3	ADx Bit 4	ADx Bit 3	ADx Bit 4	ADx Bit 3	ADx位5	ADx位5	
D2	ADx位2	ADx位2	ADx位2	ADx Bit 3	ADx位2	ADx Bit 3	ADx位2	ADx Bit 4	ADx位4	
D1	ADx位1	ADx位1	ADx位1	ADx位2	ADx位1	ADx位2	ADx位1	ADx Bit 3	ADx位3	
D0	ADx位0 (LSB)	ADx位0 (LSB)	ADx位0 (LSB)	ADx位1 (LSB)	ADx位0 (LSB)	ADx位1 (LSB)	ADx位0 (LSB)	ADx位2 (LSB)	ADx位2 (LSB)	

## 将ADC样本转换为电压值

频谱驱动程序还包含一个寄存器，该寄存器保存已安装的ADC的全比例表示的十进制值。当将ADC值（在LSB中）转换为实际电压值时，应该使用此值，因为此寄存器还会自动考虑任何特殊性，例如稍微降低ADC分辨率并保留增益/偏移补偿代码。

登记册	价值	方向	说明
SPC_MINST_MAXADCVALUE	1126	念	包含ADC全比例值的十进制代码（以LSB为单位）。

如果板使用8位ADC，提供完整的ADC代码（不保留任何位），则返回值为128。 $\pm 1.0\text{V}$ 输入范围的峰值为1.0V（或1000mV）。

然后，返回的样本值（例如+49（十进制，两个补码，签名表示））将转换为：

$$V_{in} = 49 \times \frac{1000\text{毫V}}{128} = 382.81\text{ mV}$$

然后，返回的样本值（例如-55（十进制））将转换为：

$$V_{in} = -55 \times \frac{1000\text{毫V}}{128} = -429.69\text{ mV}$$

当转换包含任何附加数据的样本时，例如附加的数字信道或超差位，必须首先屏蔽这些额外的信息，并且必须执行适当的符号扩展，然后这些值才能用作上述公式的有符号的两个补足值。



## 使用特殊时钟模式时应用校正因子

当使用所谓的特殊时钟模式（SPC\_SPECIA LCLOC K）时，ADC输入范围随所选采样率的变化而变化。这可以通过将上述计算的电压值与适当的校正因子相乘来补偿。如何从驱动程序获得这些因素的过程在本手册后面的专用时钟章节中描述。