

收购模式

您的卡可以在不同的模式下运行。根据所选模式，有不同的寄存器，每个寄存器都定义了此模式的一个方面。在本章中解释了单一模式。只有在卡上安装了一个选项时才能使用的任何其他模式都将在后面的章节中记录下来。

概述

本章概述了不同模式的相关寄存器。这些寄存器在不同模式下的使用将在下面的章节中描述。

模式的设置

模式寄存器被组织为位图。每个模式对应于此位图的一位。在定义要使用的模式时，请确保只设置其中一位。所有其他设置将返回错误代码。

所有标准模式和所有FIFO模式的主要区别在于，标准模式仅限于车载内存，因此可以完全采样率运行。FIFO模式旨在通过总线将数据连续传输到PC内存或硬盘，因此可以运行更长的时间。FIFO模式受到PC机可以使用的最大总线传输速度的限制。FIFO模式使用完整安装的车载内存作为FIFO缓冲区。

然而，正如您将在整个模式的详细文档中看到的那样，标准和FIFO模式在编程和行为上是相似的，它们之间只有很少的区别。

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_AVAILCARDMOTES	9501	念	返回一个位图与所有可用的模式在您的卡。下面列出了这些模式。

收购模式

Mode	价值	可於	说明
SPC_REC_STD_SINGLE	1h	所有卡片	一个触发事件的数据采集到车载内存。
SPC_REC_STD_MULTI	2h	所有卡片	多个触发事件的数据采集到车载内存。每个记录的段都有相同的大小。这种模式在一个关于多个记录选项的特殊章节中有更详细的描述。
SPC_REC_STD_GATE	4h	所有卡片	使用外部门信号对车载存储器进行数据采集..只有当门信号具有编程电平时，才能进行采集。在一个关于门控采样选项的特殊章节中，对该模式进行了更详细的描述。
SPC_REC_STD_ABA	8h	只有数字化器	多个触发事件的数据采集到车载内存。当多个触发事件以编程采样率存储时，输入以较慢的采样速度连续采样。该模式在关于ABA模式选项的特殊章节中进行了描述。
SPC_REC_STD_SEGSTATS	10000h	M4i/M4x.22xx M4i/M4x.44xx DN2/DN6.22x DN2/D N6.44x数字化仪	数据采集到车载内存中的多个触发事件，使用块/分类统计模块（FPGA固件选项）。
SPC_REC_STD_AVERAGE	20000h	M4i/M4x.22xx M4i/M4x.44xx DN2/DN6.22x DN2/D N6.44x数字化仪	多个触发事件的数据采集到车载内存，使用块平均模块（FPGA固件选项）。
SPC_REC_STD_BOXCAR	800000h	M4i/M4x.44xx DN2/DN6.44x数字化仪	启用BoxcarAveraging进行标准获取。要求数字化器模块与固件版本V29或更新。
SPC_REC_FIFO_SINGLE	10h	所有卡片	单个触发事件的连续数据采集。车载内存完全用作FIFO缓冲区..
SPC_REC_FIFO_MULTI	20h	所有卡片	多个触发事件的连续数据采集..
SPC_REC_FIFO_GATE	40h	所有卡片	使用外部门信号进行连续数据采集。
SPC_REC_FIFO_ABA	80h	只有数字化器	多个触发事件的连续数据采集，以及一个较慢的采样时钟的连续数据采集。
SPC_REC_FIFO_SEGSTATS	100000h	M4i/M4x.22xx M4i/M4x.44xx DN2/DN6.22x DN2/D N6.44x数字化仪	启用用于FIFO获取的块/Segment Statistics（FPGA固件选项）。
SPC_REC_FIFO_AVERAGE	200000h	M4i/M4x.22xx M4i/M4x.44xx DN2/DN6.22x DN2/D N6.44x数字化仪	为FIFO获取（FPGA固件选项）启用块验证。
SPC_REC_FIFO_BOXCAR	1000000h	M4i/M4x.44xx DN2/DN6.44x数字化仪	启用BoxcarAveraging进行FIFO获取。需要数字化器模块固件版本V29或更新。
SPC_REC_FIFO_SINGLE_MONITOR	2000000h	只有数字化器	为监测目的，将SPC_REC_FIFO_SINGLE模式与其他较慢的采样时钟数据流相结合（与SPC_REC_FIFO_ABA模式的数据相同）。

命令

数据采集/数据重播由命令寄存器控制。 命令寄存器控制卡的一般状态和不同数据传输的状态。 数据传输将在后面的一章中加以解释。

这些命令分为两种类型的命令：执行完成作业的命令和等待中断发生的命令。同样，命令寄存器被组织为位图，允许您将多个命令与一个调用一起设置。由于并不是所有的命令组合都有意义（例如重置和同时启动的组合），如果在当前状态下不允许给定的命令之一，驱动程序将检查给定的命令并返回错误代码ERR_SEQUENCE。

登记册	价值	方向	说明
SPC_M2CMD	100	只写	执行卡或数据传输的命令。

卡执行命令

M2CMD_CARD_RESET	1h	执行硬和软件重置的卡，如上文进一步解释。
M2CMD_CARD_WRITESETUP	2h	将当前设置写入卡，而不启动硬件。如果更改一些内部设置，如时钟频率和启用输出，此命令可能是有用的。
M2CMD_CARD_START	4h	用所有选定的设置启动卡。如果自上一个设置被写入以来任何设置都已更改，则此命令将自动将所有设置写入卡。卡启动后，在卡运行时，不能更改任何设置。
M2CMD_CARD_ENABLETRIGGER	8h	触发检测已启用。此命令可以与start命令一起发送，以便立即启用触发器，也可以在某些外部硬件启动后进行第二次调用。
M2CMD_CARD_FORCE_TRIGGER	10h	此命令强制触发，即使到目前为止还没有检测到。将此命令与开始命令一起发送类似于使用软件触发器。
M2CMD_CARD_Disableabletrigger	20h	触发检测被禁用。所有进一步的触发事件都被忽略，直到再次启用触发检测。当启动卡时，触发检测将被禁用。
M2CMD_CARD_STOP	40h	停止卡的当前运行。如果卡没有运行，则此命令没有效果。

卡等待命令

这些命令在到达由卡的中断或超时计数器过期发出信号的定义状态之前不会返回。如果已达到状态，则返回带有ERR_OK的命令。如果出现超时，则命令返回ERR_TIMEOUT。如果该卡已从带有停止或重置命令的第二个线程中停止，则等待函数将使用ERR_ABORT返回。

M2CMD_CARD_WAITPREFULL	1000h	只采集模式：命令等待，直到预触发器区域一次充满数据。在预先触发区域被填充之后，内部触发引擎开始寻找触发事件，如果触发检测是启用了。
M2CMD_CARD_WAIT_TRIGGER	2000h	等待直到第一个触发事件被卡检测到。如果使用具有多个触发事件的模式，如多重记录或门控采样，则只有第一次触发检测将为此等待命令生成中断。
M2CMD_CARD_WAITREADY	4000h	等待直到卡完成当前运行。在获取模式下，接收此命令意味着所有数据都已被获取。在接收此命令的生成模式中，意味着输出已经停止。

等待命令超时

如果其中一个等待命令正在等待的状态没有达到任何等待命令，则如果没有定义超时，则将永远等待，或者如果指定的超时已过期，则它将以ERR_TIMEOUT自动返回。

登记册	价值	方向	说明
SPC_TIMEOUT	295130	读/写	以毫秒级分辨率定义任何后续等待命令的超时。将零写入此寄存器将禁用超时。

默认情况下，超时将被禁用。在定义超时之后，这对于以下所有等待命令都是有效的，直到通过向此寄存器写入零再次禁用超时为止。

发生的超时不应视为错误。它并没有改变董事会的地位。板仍在运行，将正常完成。如果没有发生触发器，您可以在某个时间之后使用超时中止运行。在这种情况下，在接收到超时后，需要一个停止命令。还可以使用超时频繁更新用户界面，然后再次调用等待函数。

卡片控制示例：

```
启动//卡，立即启用触发检测
spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_CARD_START | M2CMD_CARD_ENABLETRIGGER);

//我们最多等待1秒钟进行触发检测. 在超时的情况下，我们强制触发 spcm_dw Set Param_i32 ( hDrv , SPC_TIMEOUT , 1000 ) ;

if ( spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_CARD_WAIT_TRIGGER) == ERR_TIMEOUT)
{
    printf (" 到目前为止没有检测到触发器，我们现在强制一个触发器！\n " ) ; spcm_dw Set Param ( hDrv , SPC_M2CMD , M2CMD_CARD_FORCE_TRIGGER ) ;
}

//我们禁用超时，等待运行 spcm_dw Set Param_i32 ( hDrv , SPC_TIMEOUT , 0 ) 结束；

spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_CARD_WAITREADY);
printf ("Card has stopped now!\n");
```

信用卡状况

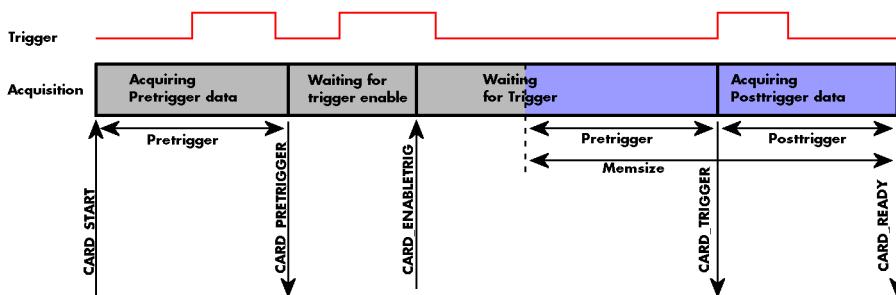
除了等待中断机制或完全代替中断机制外，还可以通过读取 SPC_M2STATUS 寄存器读出当前的卡状态。状态寄存器被组织为位图，这样就可以设置多个位，显示卡的状态以及不同的数据传输。

登记册	价值	方向	说明
SPC_M2STATUS	110	只读	读取当前状态信息
M2STAT_CARD_PREtrigger	1h		采集模式仅：预触发器区域已填充。
M2STAT_CARD_TRIGGER	2h		第一个触发器已经被检测到。
M2STAT_CARD_READY	4h		这张卡已经跑完了，准备好了。
M2STAT_CARD_SEGMENT_PRETRG	8h		只有M4i/M4x/M2p的多/ABA/Gated获取：一个段的预触发器区域已经填充。

购置卡状况概览

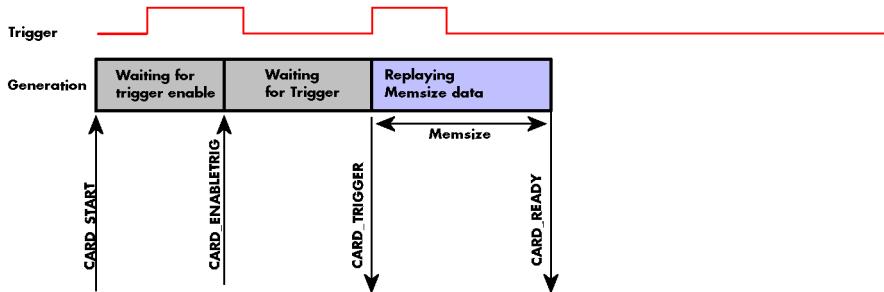
下面的绘图给出了卡命令和卡状态信息的概述。卡开始后

M2CMD_CARD_START 卡在获得完整的预触发器数据之前一直在获取预触发器数据。然后设置状态位 M2STAT_CARD_PREtrigger。要么已与开始命令一起启用触发器，要么卡现在等待触发器启用命令 M2CMD_CARD_ENABLETRIGGER。在接收到此命令后，将启用触发器引擎，并对触发事件进行卡片检查。一旦接收到触发事件，就设置状态位 M2STAT_CARD_TRIGGER，卡获取编程后触发数据。在获取所有后触发数据之后，将设置状态位 M2STAT_CARD_READY，并可读出数据：



生成卡状态概述

此图概述了简单生成模式的卡命令和状态信息。使用 M2CMD_CARD_START 启动卡后，卡被武装起来等待。要么已与开始命令一起启用触发器，要么卡现在等待触发器启用命令 M2CMD_CARD_ENABLETRIGGER。在接收到此命令后，将启用触发器引擎，并对触发事件进行卡片检查。一旦接收到触发事件，就设置状态位 M2STAT_CARD_TRIGGER，并以数据重播开始卡。重播完成后-取决于编程模式-状态位 M2STAT_CARD_READY 被设置，卡停止。



数据传输

数据传输由两部分组成：缓冲区定义和控制传输本身的命令/状态信息。数据传输与卡控命令和状态信息共享命令和状态寄存器。一般而言，以下有关数据传送的详情适用于任何方向的任何数据传送：

内存大小寄存器 (SPC_MEMSIZE) 必须在开始数据传输之前进行编程。
· 在开始数据传输之前，缓冲区必须使用 spcm_dwDef 传输函数定义。
· 每个定义的缓冲区只使用一次。在转移结束后，缓冲区自动失效。

如果数据传输正在进行或缓冲区至少已经定义，则必须删除缓冲区，则有必要调用
无效的 Buf 函数。

转移缓冲区的定义

在开始任何数据传输之前，必须定义包含所有细节的传输缓冲区。 缓冲区的定义是用 spcm_dw Def 传输函数完成的，如前面一章所解释的那样。

```
使用64位无符号整数值drv_handleh Device , /句柄到已经打开的设备定义传输缓冲区
uint32d wBuf Type , /类型的缓冲区定义如下：在SPCM_BUF_XX XX uint32dw Direction , /下面定义的传输方向
uint32d wnotifySize , /之后发送事件的字节数（0=传输结束）void*pv Data Buffer , /指向数据缓冲区的指针
uint64qw BrdOffs , /偏移量，用于板存储器uint64qw传输Len ) ; /缓冲区长度
```

此函数用于定义标准样本数据传输的缓冲区以及额外的ABA或时间戳信息的额外数据传输。 因此，dwBufType参数可以是下列参数之一：

SPCM_BUF_DATA	1000	缓冲区用于传输标准样本数据
SPCM_BUF_ABA	2000	缓冲区用于读出缓慢的ABA数据。 有关此模式的详细信息，请参阅有关ABA模式选项的章节。
SPCM_BUF_TIMESTAMP	3000	缓冲区用于读出时间戳信息。 有关此模式的详细信息将在有关时间戳选项的章节中描述。

dw方向参数定义了以下数据传输的方向：

SPCM_DIR_PCTOCARD	0	从PC内存转移到卡的车载内存
SPCM_DIR_CARDTOPC	1	从板内存到PC内存的传输。
SPCM_DIR_CARDTOGPU	2	RDMA从卡内存转移到GPU内存，只需要Linux SCAPP选项
SPCM_DIR_GPUTOCARD	3	RDMA从GPU内存传输到卡内存，只需要Linux SCAPP选项

 这里使用的方向信息必须与当前使用的模式相匹配。 虽然使用了一种采集模式，但没有从PC到允许的卡的传输，反之亦然。 可以使用特殊的内存测试模式来超过这个限制。 将 SPC_MEMTEST 寄存器设置如下。

dwNotificationSize参数定义了在此之后应该生成中断的字节数。 如果将此参数为零，则传输将运行，直到所有数据被传输，然后生成中断。 填写通知大小>零将允许您使用到目前为止已经传输的数据量。 在FIFO模式下使用通知大小来实现与驱动程序的缓冲区握手，或者在传输大量数据时，在数据传输仍在运行的情况下，它可能对启动数据处理感兴趣。 详情请参阅下面关于处理职位的章节。

 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。

光伏数据缓冲区必须指向为传输分配的数据缓冲区。 请确保至少分配了您的程序要传输的内存量。 如果传输是从卡到PC，这些数据将被覆盖与当前内容的卡的车载内存。

当不执行FIFO模式时，也可以使用QW BrdOffs参数。 此参数将数据传输的起始位置定义为相对于卡内存开始的字节值。 如果一个人已经获得了一个非常大的车载内存，使用这个参数可以让它以更小的块分割数据传输。

Qw传输Len参数定义了必须使用此缓冲区传输的字节数。 请确保分配的内存至少具有此参数中定义的大小。 在标准模式下，此参数不能大于用内存大小定义的数据量。

内存测试模式

在某些情况下，向相反方向传输数据可能会引起兴趣。 因此，一种特殊的内存测试模式是可用的，它允许随机读取和写入完整的车载内存。 当内存测试模式被激活时，没有处理正常的卡命令：

登记册	价值	方向	说明
SPC_MEMTEST	200700	读/写	写一个1激活内存测试模式，然后没有命令被处理。 写入0将再次停用内存测试模式。

转让缓冲区失效

如果缓冲区即将被用户删除，则该命令可用于使已经定义的缓冲区失效。如果定义了新的缓冲区，或者缓冲区的传输已经完成，则自动调用此函数

```
uint32 __stdcall spcm_dwInvalidateBuf ( // invalidate the transfer buffer
    drv_handle hDevice,           // handle to an already opened device
    uint32 dw Buf Type ) ; /缓冲区的类型，以使上面所列的SPCM_BUF_XX XX失效
```

dw Buf类型参数需要是定义了缓冲区的相同参数：

SPCM_BUF_DATA	1000	缓冲区用于传输标准样本数据
SPCM_BUF_ABA	2000	缓冲区用于读出缓慢的ABA数据。有关此模式的详细信息将在关于ABA模式选项的一章中描述。ABA模式只适用于模拟采集卡。
SPCM_BUF_TIMESTAMP	3000	缓冲区用于读出时间戳信息。有关此模式的详细信息将在有关时间戳选项的章节中描述。时间戳模式仅在模拟或数字采集卡上可用。

数据传输缓冲区的命令和状态信息。

如上所述，数据传输是在相同的命令和状态寄存器中执行的，就像卡控件一样。如下面的示例所示，可以同时发送用于卡控和数据传输的命令。

登记册	价值	方向	说明
SPC_M2CMD	100	只写	执行对卡或数据传输的命令
M2CMD_DATA_STARTDMA	10000h		启动已定义缓冲区的DMA传输。在获取模式下，可能是卡还没有收到触发器，在这种情况下，转移启动被延迟到卡接收到触发器事件为止
M2CMD_DATA_WAITDMA	20000h		等待直到数据传输结束，或者至少直到可用通知大小定义的字节数为止。这个等待函数还考虑了上面描述的超时参数。
M2CMD_DATA_STOPDMA	40000h		停止正在运行的DMA传输。数据事后无效..

数据传输可生成下列状态信息之一：

登记册	价值	方向	说明
SPC_M2STATUS	110	只读	读取当前状态信息
M2STAT_DATA_BLOCKREADY	100h		在通知大小中定义的下一个数据块可用。它至少是可用的数据量，但也可以是更多的数据。
M2STAT_DATA_END	200h		资料移交已完成..只有当通知大小设置为零时，才会出现此状态信息。
M2STAT_DATA_OVERRUN	400h		在进行FIFO传输时，数据传输具有溢出（采集）或欠运行（重放）。
M2STAT_DATA_ERROR	800h		在进行数据传输时发生了内部错误。

数据传输示例

```
void* pvData = pvAllocMemPage(1024);

//将数据从PC内存传输到卡内存（在重播卡上）...
spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_DATA, SPCM_DIR_PCTOCARD, 0, pvData, 0, 1024);
spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_DATA_STARTDMA | M2CMD_DATA_WAITDMA);

//...或将数据从卡内存转移到PC内存（采集卡）
spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_DATA, SPCM_DIR_CARDTOPC, 0, pvData, 0, 1024);
spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_DATA_STARTDMA | M2CMD_DATA_WAITDMA);

在失效缓冲区 spcm_dwSetParam_i32 (hDrv, SPC_M2CMD, M2CMD_DATA_STOPDMA) 之前
, //显式停止DMA转移； spcm_dw无效Buf (hDrv, SPCM_BUF_DATA) ;

vFreeMemPage(pvData, 1024);
```

为了使示例保持简单，它不进行错误检查。请务必检查错误，如果使用这些命令在现实世界的程序！

用户应注意在失效缓冲区之前显式发送M2CMD_DATA_STOPDMA命令，以避免在使用更高延迟的数据传输层（例如远程以太网设备）时由于竞赛条件而崩溃。

