HPM6000 系列

HPM 电机外接 SPI 编码器方案

本文适用于先楫半导体HPM 电机外接SPI编码器方案

目 录

版本：

日期	版本号	说明
2023-11-29	V1.0	初版

1 简介

HPM6000 系列 MCU 是来自上海先楫半导体科技有限公司的高性能实时 RISC-V 微控制器，为工业自动化及边缘计算应用提供了极大的算力、高效的控 制能力。上海先楫半导体目前已经发布了如 HPM6700/6400、HPM6300、HPM6200 等多个系列的高性能微控制器产品。

在 HPM6700/6400、6300、6200 系列微控制器上均支持 16 位 ADC 采样和串行外设总线 SPI。HPM6200 系列支持 3 个 16 位的 ADC 转换器，可以转换来自外部引脚以及芯片内部的模拟信号。ADC 支持读取转换模式、周期转换模式、序列转换模式和抢占转换模式。SPI 支持 DMA 数据传输，支持 2 线 4 线模式，并有多种数据传输模式可选，主模式下可启用命令段以及地址段传输。

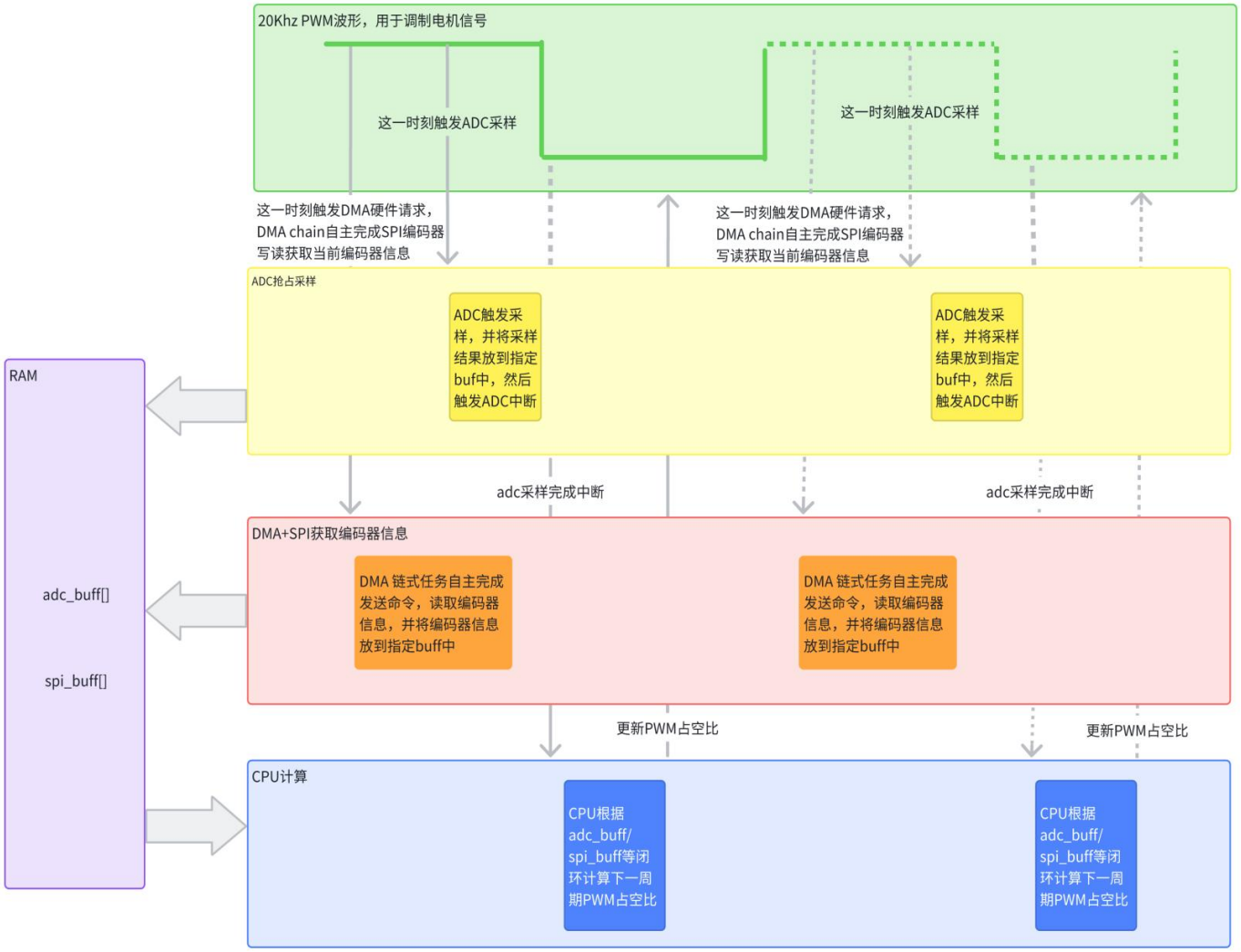
该方案通过 PWM 特定时刻触发 ADC 采样，同时在 PWM 另一特定时刻触发 HDMA 操作 SPI 控制器，并依靠 HDMA 链式模式，做成循环链表，由 HDMA 全自动完成 SPI 的读写任务，将读到的数据存放到指定内存中，无需 CPU 参与。同时 HPM ADC 控制器也自带 DMA，ADC 采样完成后自带 DMA 将采样的数据存放到指定内存中，CPU 也无需参与。用户可直接在 ADC 采样完成中断中使用 ADC 采样结果及 SPI 编码器信息，去做后续闭环算法，最终调整下一周期 PWM 占空比。

本文不涉及介绍闭环算法等相关内容。

DMA 链式传输通常能够降低数据传输的延时以及将规则的连续任务由 DMA 自主完成。此方案中 ADC 抢占采样、DMA 获取 SPI 编码器信息并行执行，从而减少了数据从源到目标的传输时间。这对于实时系统和需要快速响应的应用非常关键。

2 整体方案

整体方案的功能框图如下图 1 所示：



图表 1

电机信号调节是由 20KHz(0~reload 范围)的 PWM 波形调制，设定好 reload（重载寄存器）值，当计数器开始计时后，CNT 计数到比较器配置的 CMP 值时，就会产生匹配事件，

从而 OCx 输出逻辑 1，当计数器 CNT 值到达重载寄存器，发生重载事件，OCx 输出重置逻辑 0。

把比较器设置为输出比较，设定一个比较值，开始计数后，CNT 计数到比较器配置的 CMP 值时,可以触发 ADC 抢占采样；再设置另一个比较器，让 PWM 另一特定时刻可以触发 DMA 链式传输获取编码器信息。DMA 控制 SPI 获取编码器信息和 ADC 抢占采样并行执行，并将获取的编码器信息和采样的 ADC 结果放到指定 ram 区域 spi_buff[]和 adc_buff[]中。在 ADC 中断中，CPU 直接使用 adc_buff 和 spi_buff 等闭环计算下一周期的 PWM 占空比，并更新 PWM 的比较器寄存器，整个方案循环进行。

特性：

（1） ADC 抢占采样由 PWM 硬件触发，自主完成采样并将采样结果放到指定内存中。CPU 零负载参与。

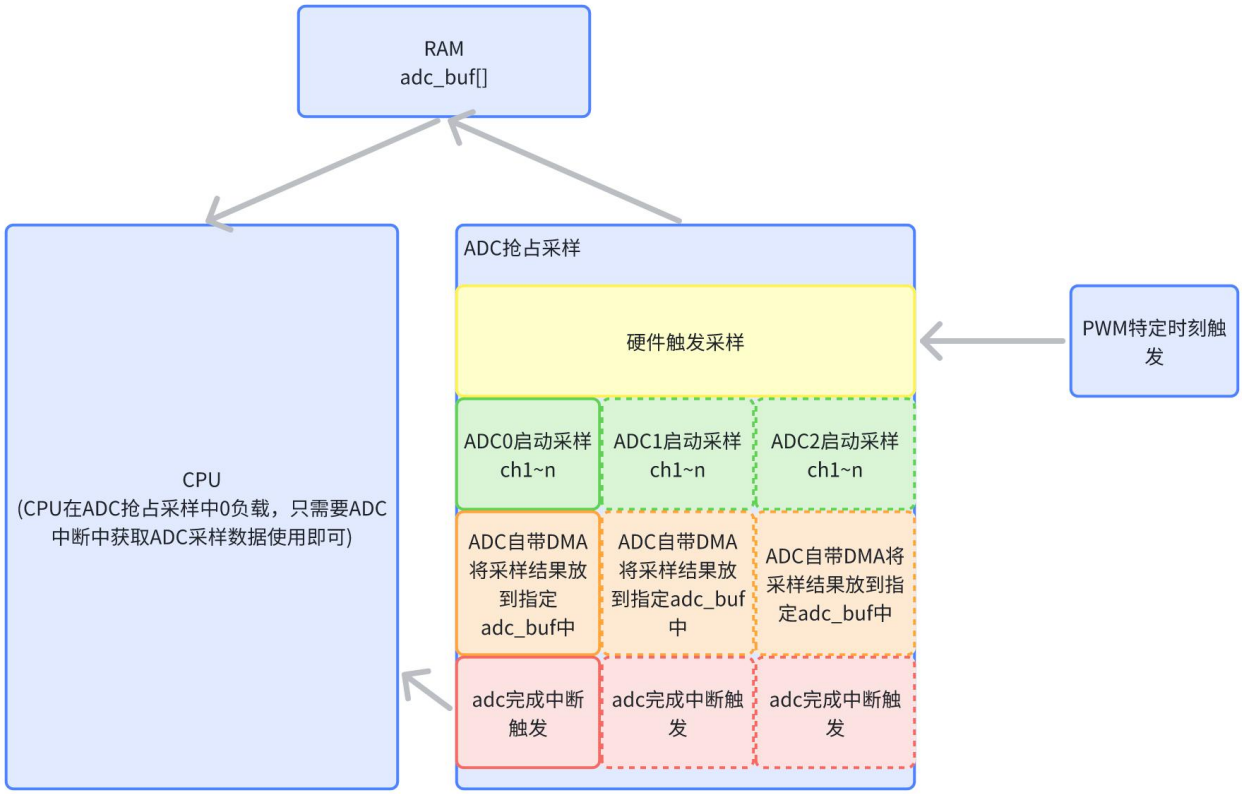
（2） SPI 获取编码器信息，由 PWM 硬件触发 DMA 链式任务，DMA 自主完成编码器信息获取并将结果放到指定内存中。CPU 零负载参与。

（3） ADC 抢占采样和 DMA 链式任务 SPI 获取编码器信息可并行执行。

注意：此方案中 ADC 触发采样时刻，以及 DMA 链式任务中 SPI 获取编码器信息的时刻均可任意调节。用户需根据 SPI 编码器特性，来调整 SPI 获取编码器信息及 ADC 采样时刻。确保在 ADC 采样完成进中断时，SPI 获取编码器信息也已完成。

3 ADC 抢占采样(PWM 硬件触发)

ADC 抢占模式下的功能框图 2 如下所示：



图表 2

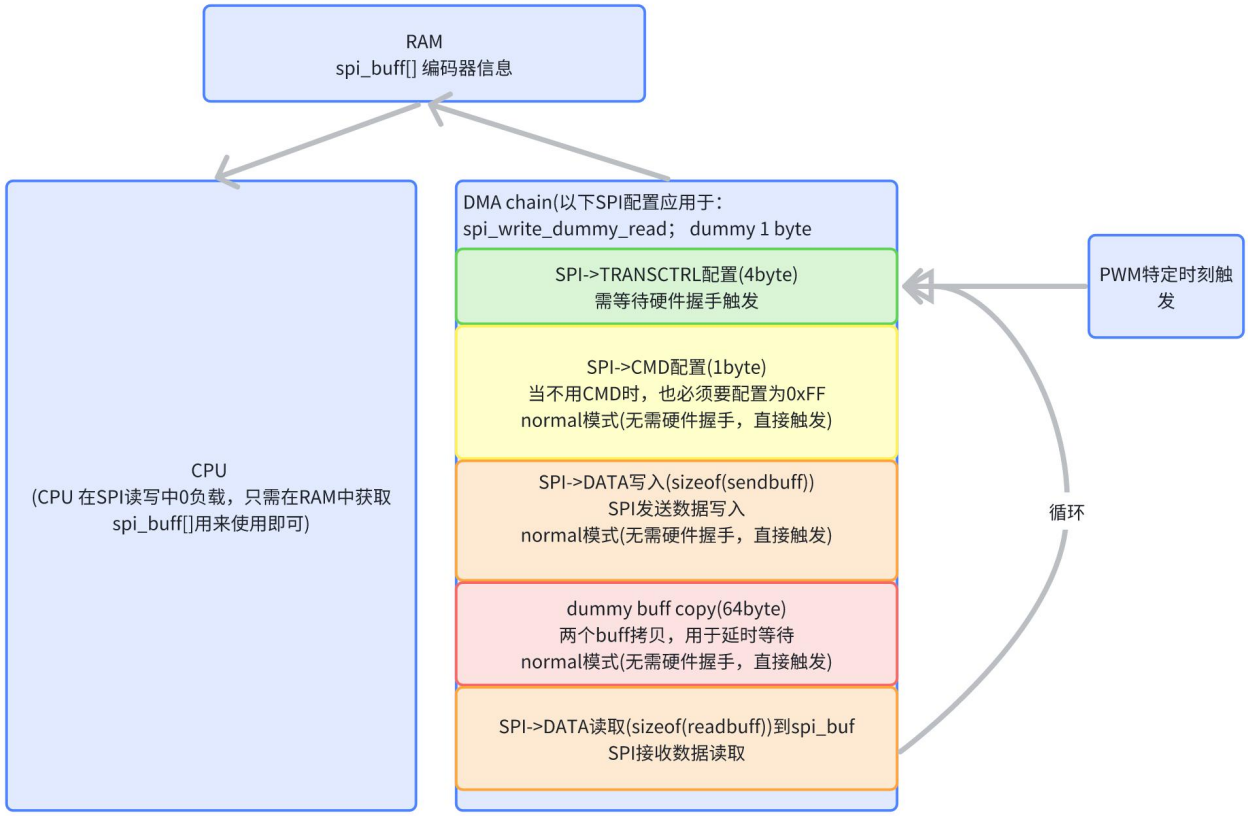
在 PWM 的特定时刻，触发 ADC 采样。本方案使用 ADC 抢占转换模式，HPM6000 全系列均支持此模式。本文使用的 HPM6200 系列中，ADC 支持 12 组抢占转换序列，触发信号分别来自于芯片的片上互联管理器模块，共 12 个抢占触发源，对于本方案使用到的硬件触发方式，只要在 PTRGIxA， PTRGIxB 或者 PTRGIxC 上捕获到上升沿，即触发抢占序列 xA，xB 或 xC。

抢占转换一旦开始，就会根据 [TRIG_LEN] 的配置连续转换完成整个抢占序列。抢占序列的第 x 次转换完成后，如果 CONFIGx 寄存器的 [INTENx] 位置 1， INT_STS[TRIG_CMPT] 标志位会置 1，如果相应的中断控制位也置 1， ADC 就会生成中断。

同时 ADC 抢占转换模式支持内置 DMA，按照触发来源以及在抢占转换序列中的序号，DMA 会把转换结果写入缓冲区对应的位置。这样 CPU 可以直接去中断中获取 ADC 采样数据即可。

4 DMA 结合 SPI 获取编码器信息 (PWM 硬件触发)

DMA 结合 SPI 获取编码器信息的功能框图 3 如下所示：



图表 3

在 PWM 的特定时刻，触发 DMA 搬运请求。互联管理器支持管理与它互联的部分外设的 DMA 请求，这些外设的 DMA 请求通过互联管理器转发到 DMAMUX，DMAMUX 将此信号转发到指定的 DMA 通道，DMA 通道获取到此信号后完成硬件握手触发 DMA 搬用。此方案还使用到了 DMA 链式模式，并设置为循环链，可以由 DMA 全自动完成 SPI 的读写等任务。这样 CPU 无需干预 SPI 的读写工作，在 ADC 采样的同时，DMA 也在搬运 SPI 的读写数据，极大的增加了效率。

4.1 SPI 传输简介

SPI 在主机模式下控制发起传输。 SPI 传输的格式和接口时序可以通过寄存器编程。SPI 传输包括命令、地址和数据字段， SPI 控制器提供专用的寄存器用来存储这些字段。

对于 SPI 的传输，可以在先楫软件 SPI 驱动中，使用 spi_transfer 赋值以下结构体：

- cmd_enable:使能命令段传输。
- addr_enable：使能地址段传输。
- addr_pahase_fmt：选择是单线模式还是四线模式传输地址。
- trans_mode：选择的传输模式，比如全双工的方式：支持同时读写。或者半双工的方式：仅写，仅读，先写后读，先读后写。
- dummy_cnt：填充 Dummy 字节数量。

本方案使用 spi_trans_write_dummy_read（先写-继填充-后读）的方式，也就是先写然后填充 Dummy 字节，最后再读，部分配置代码如下所示：

```
spi_master_get_default_control_config(&control_config);
 control_config.master_config.cmd_enable = false;
 control_config.master_config.addr_enable = false;
 control_config.master_config.addr_phase_fmt = spi_address_phase_format_single_io_mode;
 control_config.common_config.trans_mode = spi_trans_write_dummy_read;
 control_config.common_config.data_phase_fmt = spi_single_io_mode;
 control_config.common_config.dummy_cnt = spi_dummy_count_1;


 spi_transfer(BOARD_APP_SPI_BASE,
```

```
&control_config,
NULL, NULL,
sendbuff, ARRAY_SIZE(sendbuff), (uint8_t *)spi_buf, ARRAY_SIZE(spi_buf));
```

在 spi_transfer 这个 API 接口中，做了以下三步步骤：

第一步：

配置 TRANSCTRL 寄存器，此寄存器可控制是否启动地址段，命令段以及传输格式，写入与读取数据的长度功能。在先楫用户手册中也可以找到相关描述，如下图 4 部分截图所示:

46.4.2 TRANSCTRL (0x20)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SLVDATAONLY	CMDEN	ADDREN	ADDRFMT	TRANSMODE				DUALQUAD		TOKENEN	WRTRANCNT									TOKENVALUE	DUMMYCNT		RDTRANCNT								
RW	RW	RW	RW	RW				RW		RW	RW									RW	RW		RW								
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TRANSCTRL [31:0]

位域	名称	描述
31	SLVDATAONLY	纯数据模式（仅从模式）： 0x0：禁用纯数据模式， 0x1：启用纯数据模式。 注：此模式仅在常规模式下有效，因此 MOSIBiDir、DualQuad 和 TransMode 应设置为 0。
30	CMDEN	启用命令段（仅主模式）： 0x0：禁用命令段， 0x1: 启用命令段。
29	ADDREN	启用地址段（仅主模式）： 0x0：禁用地址段， 0x1: 启用地址段。
28	ADDRFMT	SPI 地址段格式（仅主模式）： 0x0：常规（1 位）模式， 0x1：数据段（2 线/4 线）相同。
27-24	TRANSMODE	传输模式，传输顺序可以是 0x0：同时读写， 0x1：仅写， 0x2：只读， 0x3：写，读， 0x4：读、写， 0x5：写、填充、读， 0x6：读、填充、写， 0x7：无数据（必须在主模式下启用 CmdEn 或 AddrEn）， 0x8：填充，写， 0x9：填充，读， 0xa~0xf: 保留。

图表 4

如图 5 所示，SPI 写入数据长度与读取数据长度的配置也在这个 API 接口里配置了:WRTRANCNT,RDTRANCNT。由写入与读取的实际 buffer 大小决定，例如写入数据长度，配置为 sizeof（sendbuf）。

位域	名称	描述
20-12	WRTRANCNT	写入数据长度。 WrTranCnt 表示 SPI 写入数据长度。实际传输长度为（WrTranCnt+1）。 WrTranCnt 仅在 TransMode 为 0、1、3、4、5、6 或 8 时生效。 数据单位的大小（位宽度）由传输格式寄存器的 DataLen 字段定义。 对于传输模式 0，WrTranCnt 必须等于 RdTranCnt。
11	TOKENVALUE	令牌值（仅主模式），SPI 读取地址段后的令牌。 0x0：令牌值 =0x00， 0x1：令牌值 =0x69。
10-9	DUMMYCNT	填充数据长度。实际填充长度为（DummyCnt+1）。 SPI 接口上的填充周期数为（DummyCnt+1）*（(DataLen+1)/SPI IO 宽度），在填充数据段，数据引脚被置于高阻抗中。 DummyCnt 仅用于 TransMode 5、6、8 和 9，其具有填充数据段。
8-0	RDTRANCNT	读取数据长度，RdTranCnt 表示从 SPI 总线接收的数据长度。实际接收长度为（RdTranCnt+1）。 RdTransCnt 仅在 TransMode 为 0、2、3、4、5、6 或 9 时生效。 数据单位由传输格式寄存器的 DataLen 字段定义。 对于传输模式 0，WrTranCnt 必须等于 RdTranCnt。

TRANSCTRL 位域

图表 5

而数据单位的大小（位宽度）是由传输格式寄存器的 DATALEN 字段定义的，这个配置在 SPI 初始化里，已经配置，如下部分代码所示：

```
spi_master_get_default_format_config(&format_config);

format_config.master_config.addr_len_in_bytes = 2U;

format_config.common_config.data_len_in_bits = 8;
```

第二步：

需要配置 CMD 寄存器，该寄存器在 SPI 主机模式下，无论是否禁用命令，都需要为之赋值，因为一旦赋值就是标志开始传输，此时会根据需要传输的长度发出相对的 SCLK 时钟数量。

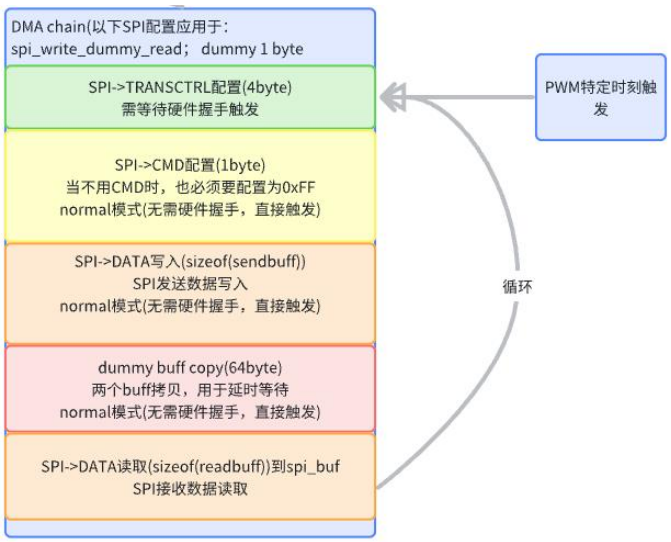
第三步：

往 DATA 寄存器读取或者写入相应的数据。

本方案中，就是将以上任务根据编码器 SPI 时序要求分成几个子任务，做成循环链表，从而达到 DMA 自动搬运的功能。

4.2 DMA 链式传输

HPM6000 系列的 DMA 支持链式连接多个任务，该方案利用这一特点，将 SPI 数据读写分为了五个子任务，并做成循环链表，DMA 自动循环完成各个任务。DMA 采用硬件握手的方式，在 PWM 特定时刻触发硬件握手。DMA 链式执行的任务功能框图如下图 6 所示:



图表 6

按照 SPI 传输的特性，使用先写后读的传输方式，分为了 5 个链表任务，DMA 描述符的写入同样也有对应的 API 接口 -> dma_config_linked_descriptor。各任务通过 adc_descriptors1 数组连接起来，利用 HDMA 顺序循环执行。DMA 任务描述符中的 ChnLLPointer 控制字就是 DMA 通道 n 下一个任务描述符的指针。如果 ChnLLPointer 值非零， DMA 控制器会在完成当前任务描述符的相应任务后，从 ChnLLPointer 指向地址取下一个任务描述符。如果 ChnLLPointer 的值为 0，表示此任务描述符为列表中最后一个，链条中止。

链表任务 1：配置 SPI->TRANSCTRL 寄存器的代码如下所示：

```
  /* SPI CTRL */
  dma_ch_config.size_in_byte = 4;
  dma_ch_config.src_addr = core_local_mem_to_sys_address(HPM_CORE0,
(uint32_t)&spi_transctrl[0]);
  dma_ch_config.dst_addr = core_local_mem_to_sys_address(HPM_CORE0,
(uint32_t)&BOARD_APP_SPI_BASE->TRANSCTRL);
  dma_ch_config.src_width = DMA_TRANSFER_WIDTH_WORD;
  dma_ch_config.dst_width = DMA_TRANSFER_WIDTH_WORD;
  dma_ch_config.src_burst_size = DMA_NUM_TRANSFER_PER_BURST_1T;
  dma_ch_config.src_mode = DMA_HANDSHAKE_MODE_NORMAL;
  dma_ch_config.dst_mode = DMA_HANDSHAKE_MODE_HANDSHAKE;
  dma_ch_config.src_addr_ctrl = DMA_ADDRESS_CONTROL_FIXED;
  dma_ch_config.dst_addr_ctrl = DMA_ADDRESS_CONTROL_FIXED;
```

```
  dma_ch_config.linked_ptr = core_local_mem_to_sys_address(HPM_CORE0,
(uint32_t)&adc_descriptors1[1]);
  dma_config_linked_descriptor(HPM_HDMA, &adc_descriptors1[0], 0, &dma_ch_config);
```

在这个链表配置中，有一些参数值得注意：

- 需要根据 TRANSCTRL 寄存器的位数，决定传输的数据位宽，如上述代码所示，TRANSCTRL 寄存器为 32 位字节，所以此处源数据以及目标传输位宽都需要配置为 32 位（DMA_TRANSFER_WIDTH_WORD）。
- TRANSCTRL 寄存器为固定地址，所以源数据和目标数据地址的访问控制都配置为固定地址。
- 而作为链式第一个任务，需要让 DMA 采用硬件握手的方式，从而在 PWM 特定时刻触发硬件握手，可以开始这个链式循环，所以需要将目标数据的 DMA 握手模式配置为 DMA_HANDSHAKE_MODE_HANDSHAKE。
- 链式传输的描述符地址，指向 adc_descriptors1[1]的地址，这样在执行完这个任务之后，可以顺利进入下一个链表任务。

链表任务 2：配置 SPI->CMD 寄存器的代码如下所示：

```
/* SPI CMD */
  dma_ch_config.size_in_byte = 1;
  dma_ch_config.src_addr = core_local_mem_to_sys_address(HPM_CORE0, (uint32_t)&0xff);
  dma_ch_config.dst_addr = core_local_mem_to_sys_address(HPM_CORE0,
(uint32_t)&BOARD_APP_SPI_BASE->CMD);
```

```
  dma_ch_config.src_width = DMA_TRANSFER_WIDTH_BYTE;
  dma_ch_config.dst_width = DMA_TRANSFER_WIDTH_BYTE;
  dma_ch_config.src_burst_size = DMA_NUM_TRANSFER_PER_BURST_1T;
  dma_ch_config.src_mode = DMA_HANDSHAKE_MODE_NORMAL;
  dma_ch_config.dst_mode = DMA_HANDSHAKE_MODE_NORMAL;
  dma_ch_config.src_addr_ctrl = DMA_ADDRESS_CONTROL_FIXED;
  dma_ch_config.dst_addr_ctrl = DMA_ADDRESS_CONTROL_FIXED;
  dma_ch_config.linked_ptr = core_local_mem_to_sys_address(HPM_CORE0,
(uint32_t)&adc_descriptors1[2]);
  dma_config_linked_descriptor(HPM_HDMA, &adc_descriptors1[1], 0, &dma_ch_config);
```

在这个链表配置中，有一些参数值得注意：

- 配置 CMD 寄存器（1byte），寄存器在 SPI 主机模式下，如果禁用命令段，都需要为之赋值，此处赋值为 0xff。因为一旦赋值就是标志开始传输，此时会根据需要传输的长度发出相对的 SCLK 时钟数量。
- CMD 寄存器为固定地址，所以源数据和目标数据地址的访问控制都配置为固定地址。
- 源数据与目标数据都配置为普通模式，无需硬件握手，直接触发。
- 链式传输的描述符地址，指向 adc_descriptors1[2]的地址，这样在执行完这个任务之后，可以顺利进入下一个链表任务。

链表任务 3：SPI->DATA 寄存器写入(SPI 发送)的代码如下所示：

```
  dma_ch_config.size_in_byte = sizeof(sendbuff);
  dma_ch_config.src_addr = core_local_mem_to_sys_address(HPM_CORE0, (uint32_t)& sendbuff);
  dma_ch_config.dst_addr = core_local_mem_to_sys_address(HPM_CORE0,
(uint32_t)&BOARD_APP_SPI_BASE->DATA);
  dma_ch_config.src_width = DMA_TRANSFER_WIDTH_BYTE;
  dma_ch_config.dst_width = DMA_TRANSFER_WIDTH_BYTE;
  dma_ch_config.src_burst_size = DMA_NUM_TRANSFER_PER_BURST_1T;
  dma_ch_config.src_mode = DMA_HANDSHAKE_MODE_NORMAL;
  dma_ch_config.dst_mode = DMA_HANDSHAKE_MODE_NORMAL;
  dma_ch_config.src_addr_ctrl = DMA_ADDRESS_CONTROL_INCREMENT;
  dma_ch_config.dst_addr_ctrl = DMA_ADDRESS_CONTROL_FIXED;
  dma_ch_config.linked_ptr = core_local_mem_to_sys_address(HPM_CORE0,
(uint32_t)&adc_descriptors1[3]);
  dma_config_linked_descriptor(HPM_HDMA, &adc_descriptors1[2], 0, &dma_ch_config);
```

在这个链表配置中，有一些参数值得注意：

- 把 SPI 需要写入的 buffer：sendbuff 写入 DATA 寄存器中，配置相应源数据地址和目标数据地址。
- DATA 寄存器是 32 位的，由于该 SPI 编码器数据位宽是 8bit(spi init 中设置 data_len_int_bits=8)，故这里将源数据位宽和目标数据位宽均设置 8 位(DMA_TRANSFER_WIDTH_BYTE)。并连续写 sizeof(sendbuff)个字节，因为在 TRANSCTRL 里面配置过 WRTRANCN 位了，对应 4.1 节中图 5 介绍。

- Burst size 配置为 1，src 和 dst 数据位宽均为单字节 byte。意思是每次传输的字节数是(srcBurstSize*SrcWidth)1*1=1byte。
- 将 src addr ctrl 设置为 increment，将 dst addr ctrl 设置为 fixed，是因为源地址 buff 是以递增的方式依次读取 4 个字节的数据放到目标 DATA 中，而目标 DATA 地址是固定，DATA 是 fifo。
- 源数据与目标数据都配置为普通模式,无需硬件握手，直接触发。
- 链式传输的描述符地址，指向 adc_descriptors1[3]的地址，这样在执行完这个任务之后，可以顺利进入下一个链表任务。

链表任务 4：延时等待 dummy buff copy 的代码如下所示：

```
//dummy
  dma_ch_config.size_in_byte = 64;
  dma_ch_config.src_addr = core_local_mem_to_sys_address(HPM_CORE0, (uint32_t)&
dummy_buff1);
  dma_ch_config.dst_addr = core_local_mem_to_sys_address(HPM_CORE0, (uint32_t)&
dummy_buff2);
  dma_ch_config.src_width = DMA_TRANSFER_WIDTH_BYTE;
  dma_ch_config.dst_width = DMA_TRANSFER_WIDTH_BYTE;
  dma_ch_config.src_burst_size = DMA_NUM_TRANSFER_PER_BURST_1T;
  dma_ch_config.src_mode = DMA_HANDSHAKE_MODE_NORMAL;
  dma_ch_config.dst_mode = DMA_HANDSHAKE_MODE_NORMAL;
  dma_ch_config.src_addr_ctrl = DMA_ADDRESS_CONTROL_FIXED;
```

```
  dma_ch_config.dst_addr_ctrl = DMA_ADDRESS_CONTROL_FIXED;
  dma_ch_config.linked_ptr = core_local_mem_to_sys_address(HPM_CORE0,
(uint32_t)&adc_descriptors1[4]);
  dma_config_linked_descriptor(HPM_HDMA, &adc_descriptors1[3], 0, &dma_ch_config);
```

在这个链表配置中，有一些参数值得注意：

- 该方案中使用的编码器 SPI 特性要求，在 SPI 发完之后，需要有一个 dummy 空指令时间，作一个延时处理。
- 让两个 dummy_buff 互相搬运数据，写入 64 个 byte，用户可以根据实际需要等待时间进行调整。
- 源数据与目标数据都配置为普通模式,无需硬件握手，直接触发。
- 链式传输的描述符地址，指向 adc_descriptors1[4]的地址，这样在执行完这个任务之后，可以顺利进入下一个链表任务。

链表任务 5：SPI->DATA 寄存器读取（SPI 接收）的代码如下所示：

```
  //read
  dma_ch_config.size_in_byte = sizeof(spi_buf);
  dma_ch_config.src_addr = core_local_mem_to_sys_address(HPM_CORE0,
(uint32_t)&BOARD_APP_SPI_BASE->DATA);
  dma_ch_config.dst_addr = core_local_mem_to_sys_address(HPM_CORE0, (uint32_t)& spi_buf);
  dma_ch_config.src_width = DMA_TRANSFER_WIDTH_BYTE;
  dma_ch_config.dst_width = DMA_TRANSFER_WIDTH_BYTE;
  dma_ch_config.src_burst_size = DMA_NUM_TRANSFER_PER_BURST_1T;
```

```
  dma_ch_config.src_mode = DMA_HANDSHAKE_MODE_NORMAL;
  dma_ch_config.dst_mode = DMA_HANDSHAKE_MODE_NORMAL;
  dma_ch_config.src_addr_ctrl = DMA_ADDRESS_CONTROL_FIXED;
  dma_ch_config.dst_addr_ctrl = DMA_ADDRESS_CONTROL_INCREMENT;
  dma_ch_config.linked_ptr = core_local_mem_to_sys_address(HPM_CORE0,
(uint32_t)&adc_descriptors1[0]);
  dma_config_linked_descriptor(HPM_HDMA, &adc_descriptors1[4], 0, &dma_ch_config);
```

在这个链表配置中，是将读到的数据放入 spi_buf 中，这样在 ADC 中断中，可以直接使用 spi_buf：

- DATA 寄存器是 32 位的，由于该 SPI 编码器数据位宽是 8bit(spi init 中设置 data_len_int_bits=8)，故这里将源数据位宽和目标数据位宽均设置 8 位(DMA_TRANSFER_WIDTH_BYTE)。并连续读 sizeof(spi_buf)个字节，因为在 TRANSCTRL 里面配置过 RDTRANCN 位了，对应 4.1 节中图 5 介绍。
- Burst size 配置为 1，src 和 dst 数据位宽均为单字节 byte。意思是每次传输的字节数是(srcBurstSize*SrcWidth)1*1=1byte。
- 将 src addr ctrl 设置为 fixed，将 dst addr ctrl 设置为 increment，是因为源地址 DATA 固定的，DATA 是 fifo，而目标 buff 是以递增的方式依次读取 4 个字节的数据放到目标数组中。
- 在硬件触发第一个链表任务之后，其余的链表的 DMA 握手模式都配置为普通模式，意味紧跟上一个链表结束后继续数据搬运。
- 链式传输的描述符地址，重新指回 adc_descriptors1[0]的地址，完成一个循环。

5 总结

本文主要介绍了在 PWM 两个特定时刻下，分别触发 ADC 采样和 HDMA 操作 SPI 控制器，并利用 DMA 分别将 ADC 采样数据和 SPI 编码信息存放到指定内存中，用户可以在 ADC 中断中直接使用，去做后续闭环算法，达成让 ADC 采样和 DMA 获取 SPI 编码器信息并行执行的方案。

该方案的优点与特性主要体现在以下几个方面：

（1） 在电机控制系统，需要在指定时间对指定通道进行转换的实时性要求比较高的情况下，HPM6000 全系列均支持 ADC 抢占转换模式，配合芯片的片上互联模块，可支持软件或者硬件来触发转换，让优先级最高。同时此模式下支持内置 DMA，可以直接把转换结果写入内存中用户指定的缓冲区，CPU 零负载参与。

（2） HPM6000 系列的 SPI 支持多种读写方式，可自由选择全双工或者半双工的传输模式来传输命令,地址，空指令和数据字段。

（3） 根据编码器 SPI 时序要求分成几个子任务，做成循环链表。在 PWM 特定时刻触发硬件握手，让链表开始全自动循环读写，CPU 只需从内存中获取 SPI 编码器信息使用即可，大大降低 CPU 负载。可以解决由于各类 SPI 编码器读写速度限制，导致 CPU 在通过 SPI 读写编码器信息时耗时严重，从而导致 CPU 算力(CPU 等待 SPI 通信耗时)不够的情况。

（4） ADC 采样和 DMA 获取 SPI 编码器信息并行执行，大大地提升了性能，系统能够更有效地利用时间，提高整体效率， 对于实时性要求较高的应用场景非常重要，CPU 可以直接使用 ADC 采样信息和 SPI 编码器信息去做相关算法操作。