HPM6200 系列

HPM6200 系列微控制器 PLA 实战---多摩川

先楫半导体HPM6200系列微控制器PLA实战----多摩川

目录

版本：

日期	版本号	说明
2023-6-24	1.0	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 HPM6200 系列微控制器中，增加了 PLA(Programmable Logic Array)可编程逻辑阵列功能。用户可以通过对 PLA 的设置，实现对输入信号的逻辑进行处理与组合，实现预期的功能。本文通过对 PLA 的配置，实现了对多摩川编码器的数据传输过程中，对传输延迟的精准的计算。

2 目的和意义

编码器的协议有很多种，多摩川，海德汉，BISS， Hiperface 等等。其数字接口普遍是由 RS485 或者 RS422 组成。经过相关的收发器，从 MCU 的串口进行数据交互。从编码器的协议角度而言，并不复杂，高性能的先楫单片机足以完成相应的解析任务。但是，有一个问题却会影响到马达的使用效果。在工业领域，之所以采用 RS485/RS422，是为了考虑在比较长的连接距离下使用。但是在几百米的连接长度下，传输的时延往往会导致 MCU 所获取的编码器的信号不能很准确的表达瞬时的位置信息。如果能够在传输的过程中获取传输的延迟信息，并对这一延迟进行补偿，那么 MCU 就会获取准确的马达位置信息，为精准的控制马达提供了前提。基于此，我们选择使用 PLA 来进行信号的相互触发，进而得到传输线上的延迟时间。

3 设计构想

3.1 多摩川介绍

多摩川（tamagawa）是常见的编码器协议。对于此类编码器的硬件连接以及协议的具体解析不在此文的讲述范围内。多摩川编码器的输出，经过 RS485 收发器得到标准的串口(TXD/RXD)。在理想的情况下(即不考虑传输线长度)，MCU 通过串口发送命令，编码器接收到命令后，根据命令发出对应信息。根据协议，MCU 端命令发出后到接受到编码器的反馈之间的间隔时间是 3us。如图 1。

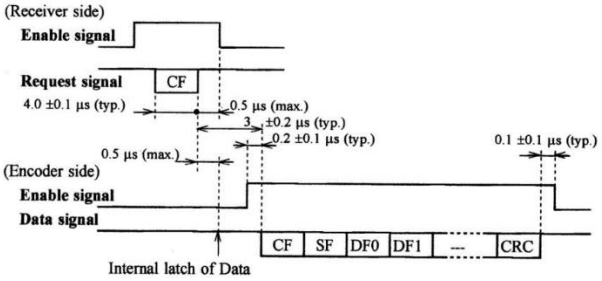

图 1

3.2 设计构想

在真实的使用情况下，如果我们可以得到命令发出与编码器反馈的时间差，那么将此间隔时间减去固定的 3us，然后再除以 2，就是整个信号再传输线上的延迟时间。那么用户就可以根据这个时间，对马达进行预测或者是补偿，进而可以进行更加精准的控制。

因为串口是异步的信号，MCU 不能控制信号的收发时间，为了获得可靠的时间间隔，我们需要利用一个主动的时钟源作为基准来控制整个传输。基于这个想法，我们可以利用 SPI 的接口来解决这一问题，此外，SPI 本身对数据传输的长度以及传输速度有较好的扩展性，能满足更多类似场景的使用需求。以下，是 PLA 部分的设计思路

- 需要 PLA 营造一个 CLK 源，这个时钟源作为第三方来控制 MCU 的 SPI 接口和编码器的输出。这个 CLK 的频率是 2.5MHz，这样设置的目的是编码器的波特率是 2.5Mbps。
- 需要 PLA 侦测编码器的输出引脚的变化，并与上述生成的 CLK 同步，使得 MCU 的 SPI 的 MOSI 获得来自编码器的数据信号。

- 需要 PLA 能够控制 CLK 发出的数量，因为只有这样，才能控制传输过程中出现的数据不会错误。
- 需要 PLA 控制一路 GPIO 输出，对 RS485 的传输方向进行控制。
- 需要给 PLA 释放一个触发信号，使得 PLA 完成整个的逻辑处理。

基于上述的设想，在实际的传输中，可以看作是基于 UART，PLA 以及 SPI 三方来实现的。从 485 收发器的角度看，MCU 的 PLA 和 SPI 实现的是一个串口功能；从 MCU 的 SPI 接口的角度看，485 收发器和 MCU 的 PLA 组成了一个 SPI 的主设备。下图 2 是基本的框架。

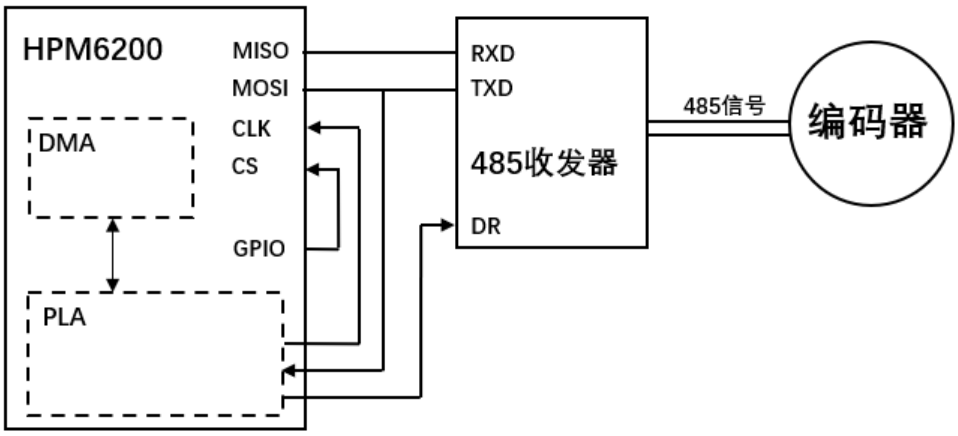

图 2

如上一节所述，本例程的目的是在实现正常的编码器传输的基础上，得到时延的信息。而 PLA 本身实现的只是逻辑组合，因此，我们还需要用 DMA 将内部的信号进行触发，得到需要的数据。

3.3 PLA 连接方式

本文所实现的例程为 SDKv1.2.0 中的 PLA 的 tamawaga 例程，

..\hpm_sdk\samples\drivers\pla\tamagawa

所选用硬件平台是基于 HPM6200EVK,用户完全可以根据自身的需要变更连接的方式。

- 外部信号连接

  具体如下表 1。

	HPM6200		
	SPI	PLA	GPIO
485_TXD	M0SI(PC23)	TRGM0P3(PB23)	
485_RXD	MISO(PC24)		
485_DR		TRGM0P1(PB21)	
	CLK(PC25)	TRGM0P5(PB25)	
	CS(PC22)		GPIO(PB30)
TAMAWAGA_PWR（注）			GPIO(PB31)

表 1

注：此需求不是必须，可以忽略。

- 内部信号连接

首先，利用 PWM 生成一个 2.5MHz 的 CLK，这个 CLK 的频率就是多摩川编码器的传输波特率。

这个 CLK 和 CLK_EN 相与后生成的信号输出给 SPI_CLK。其中 CLK_EN 是由两路控制信号(WR_EN 和 RD_EN)相或生成。MCU 通过触发信号置高 WR_EN，并通过 QEI_IN 来捕获 SPI_CLK 的个数，当达到指定的计数时（需要发出的 bit 数），则 QEI 输出一个脉冲，进而置低 WR_EN。这样，MCU 就可以控制整个 SPI 对编码器的数据的输出。而 RD_EN 则是 PLA 在监控 SPI_MOSI 信号，当 SPI_MOSI 出现下降沿，则代表有数据从编码器侧发出。此时置高 RD_EN，同时 QEI_IN 捕获 SPI_CLK 的个数，当达到指定的计数时（需要接收的 bit 数）再次发出脉冲，进而置低 RD_EN。通过 CLK_EN 的信号，触发 HALL 计数，进而得到传输的延迟时间。

4 PLA 例程分析

由于 PLA 是 HPM6200 所具备的功能，所以用户需要下载 SDKv1.2.0 或以上版本。

本文中所提到的例程基于 SDK v1.2.0， 所使用的硬件平台为 HPM6200EVK。

如何生成例程不在本文讲述范围内，用户可以在 HPM6200EVK 的用户指南中找到。

4.1 例程介绍

在 SDKv1.2.0 中，PLA 的例程位于

..\hpm_sdk\samples\drivers\pla\tamagawa

用户需要外接多摩川编码器，才能完成相应的测试。同时，用户应该按照表 1 的方式连接相应的引脚。

4.2 PLA 设置分析

例程通过 **pla_tmgw_init()**对 PLA 进行设置。用户可以参考《HPM6200 系列之 PLA 使用介绍》了解基本的 PLA 设置过程。

- FILTER1 的配置

  例程分别将 filter1in[0:3]这四路配置成了外部输入，分别为：

  - filter1in[0] ---- PWM CLK
  - filter1in[1] ---- QEI IN
  - filter1in[2] ---- MOTOR TRIG IN
  - filter1in[3] ---- SPI MOSI，本信号为下降沿触发，即下降沿生成一个脉冲。

  剩余的四路输入信号（filter1in[4..7]）被设置成软件注入为高。八路输入信号 filter1in[8:15](PLA CHN[0:7])配置成边沿同步，软件注入无效。最终形成 16 路 filter1out[0:15]。

```
filter_level1_chn_cfg.val = 0;
filter_level1_chn_cfg.sync_edge_filter_disable = true;
filter_level1_chn_cfg.software_inject = pla_filter_sw_inject_disable;
pla_set_filter1_in(PLA_TMGW_COUNTER, PLA_TMGW_PWM_IN_CHN, &filter_level1_chn_cfg);

filter_level1_chn_cfg.val = 0;
filter_level1_chn_cfg.sync_edge_filter_disable = false;
filter_level1_chn_cfg.edge_dect_en = true;
filter_level1_chn_cfg.pose_edge_dect_en = true;
filter_level1_chn_cfg.filter_reverse = false;
filter_level1_chn_cfg.software_inject = pla_filter_sw_inject_disable;
pla_set_filter1_in(PLA_TMGW_COUNTER, PLA_TMGW_QEI_IN_CHN, &filter_level1_chn_cfg);

filter_level1_chn_cfg.val = 0;
filter_level1_chn_cfg.sync_edge_filter_disable = false;
filter_level1_chn_cfg.edge_dect_en = true;
filter_level1_chn_cfg.pose_edge_dect_en = true;
filter_level1_chn_cfg.filter_reverse = false;
filter_level1_chn_cfg.software_inject = pla_filter_sw_inject_disable;
pla_set_filter1_in(PLA_TMGW_COUNTER, PLA_TMGW_MOTOR_TRG_IN_CHN, &filter_level1_chn_cfg);

filter_level1_chn_cfg.val = 0;
filter_level1_chn_cfg.sync_edge_filter_disable = false;
filter_level1_chn_cfg.edge_dect_en = true;
filter_level1_chn_cfg.nege_edge_dect_en = true;
filter_level1_chn_cfg.filter_reverse = false;
filter_level1_chn_cfg.software_inject = pla_filter_sw_inject_disable;
pla_set_filter1_in(PLA_TMGW_COUNTER, PLA_TMGW_SPI_MOSI_DATA_TRG_IN_CHN, &filter_level1_chn_cfg);

filter_level1_chn_cfg.val = 0;
filter_level1_chn_cfg.software_inject = pla_filter_sw_inject_height;
for (uint16_t i = 4; i <= PLA_TMGW_LEVEL1_FILTER_IN_END; i++) {
  pla_set_filter1_in(PLA_TMGW_COUNTER, i, &filter_level1_chn_cfg);
}
filter_level1_chn_cfg.val = 0;
filter_level1_chn_cfg.sync_edge_filter_disable = true;
filter_level1_chn_cfg.software_inject = pla_filter_sw_inject_disable;
for (uint16_t i = 0; i <= PLA_TMGW_LEVEL1_FILTER_OUT_END; i++) {
  pla_set_filter1_out(PLA_TMGW_COUNTER, i, &filter_level1_chn_cfg);
}
```

- PLA CHNO 的设置
  - 16to8 AOI 的配置

    16to8_chn0:先将 16 路的 filter1out[0:15]的输入全部配置成高输出，再将 filter1out[0]和 filter1out[10]的输入配置成同相输出后，配置到 16to8_chn0 实际是将 filter1out[0]和 filter1out[10]这两路进行与操作后作为 16to8_chn0 的输出，其余 16to8chn[2:7]则全部为高。得到 filter2in[0:7]。

```
aoi_16to8_chn_cfg.chn = pla_chn_0;
aoi_16to8_chn_cfg.aoi_16to8_chn = pla_aoi_16to8_chn_0;
for (uint16_t i = pla_level1_filter_out_0; i <= pla_level1_filter_out_15; i++) {
  aoi_16to8_chn_cfg.input[i].op = pla_aoi_operation_or_1;
  aoi_16to8_chn_cfg.input[i].signal = i;
}
aoi_16to8_chn_cfg.input[pla_level1_filter_out_0].op = pla_aoi_operation_and_1;
aoi_16to8_chn_cfg.input[pla_level1_filter_out_0].signal = pla_level1_filter_out_0;
aoi_16to8_chn_cfg.input[pla_level1_filter_out_10].op = pla_aoi_operation_and_1;
aoi_16to8_chn_cfg.input[pla_level1_filter_out_10].signal = pla_level1_filter_out_10;
pla_set_aoi_16to8_one_channel(PLA_TMGW_COUNTER, &aoi_16to8_chn_cfg);

/**
 * @brief pla 16to8 channel 2-7
 *
 */
for (uint16_t j = pla_aoi_16to8_chn_2; j <= pla_aoi_16to8_chn_7; j++) {
  aoi_16to8_chn_cfg.chn = pla_chn_0;
  aoi_16to8_chn_cfg.aoi_16to8_chn = j;
  for (uint16_t i = pla_level1_filter_out_0; i < pla_level1_filter_out_15; i++) {
    aoi_16to8_chn_cfg.input[i].op = pla_aoi_operation_or_1;
    aoi_16to8_chn_cfg.input[i].signal = i;
  }
  pla_set_aoi_16to8_one_channel(PLA_TMGW_COUNTER, &aoi_16to8_chn_cfg);
}
```

  - FILTER2 的配置

    例程将 filter2in[0:7]配置成边沿同步，软件注入无效，并配置成 filter2out[0:7]。

```
filter_level2_chn_cfg.val = 0;
filter_level2_chn_cfg.sync_edge_filter_disable = true;
filter_level2_chn_cfg.software_inject = pla_filter_sw_inject_disable;
for (uint16_t i = pla_filter2_chn0; i <= pla_filter2_chn7; i++) {
```

```
    pla_set_filter2(PLA_TMGW_COUNTER, pla_chn_0, i, &filter_level2_chn_cfg);
}
```

- 8to7 AOI 的配置

  8to7_chn[0:6]:先将 8 路的 filter2out[0:7]的输入全部配置成 0 输出，再将 filter2out[0]配置成同相输出，然后配置到 8to7_chn0。经过本次配置，实际就是将 filter2out[0]配置成 8to7_chn0，其余 8to7_chn[2:6]均为 0 输出。

```
aoi_8to7_chn_cfg.chn = pla_chn_0;
aoi_8to7_chn_cfg.aoi_8to7_chn = pla_aoi_8to7_chn_0;
for (uint16_t i = pla_level2_filter_out_0; i <= pla_level2_filter_out_7; i++) {
    aoi_8to7_chn_cfg.input[i].op = pla_aoi_operation_and_0;
    aoi_8to7_chn_cfg.input[i].signal = i;
}
aoi_8to7_chn_cfg.input[pla_level2_filter_out_0].op = pla_aoi_operation_and_1;
aoi_8to7_chn_cfg.input[pla_level2_filter_out_0].signal = pla_level2_filter_out_0;
pla_set_aoi_8to7_one_channel(PLA_TMGW_COUNTER, &aoi_8to7_chn_cfg);
```

- FILTER3 的配置

  filter3out[0]配置成边沿同步，软件注入无效，

  filter3out[2]配置为软件注入为高，filter3out[3]配置为软件注入为低，

  filter3out[4]配置为软件注入为高，filter3out[5]配置为软件注入为高，

  filter3out[6]配置为软件注入为低。

```
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.sync_edge_filter_disable = true;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_disable;
for (uint16_t i = pla_filter3_chn0; i < pla_filter3_chn1; i++) {
    pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_0, i, &filter_level3_chn_cfg);
}
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_0, pla_filter3_chn2, &filter_level3_chn_cfg);
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_low;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_0, pla_filter3_chn3, &filter_level3_chn_cfg);
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_0, pla_filter3_chn4, &filter_level3_chn_cfg);
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_height;
```

```
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_0, pla_filter3_chn5, &filter_level3_chn_cfg);
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_low;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_0, pla_filter3_chn6, &filter_level3_chn_cfg);
```

- CFF 的配置

  PLA CHN0 配置成了直接输出，即 filter3out[0]，并且时钟为 PLA 系统时钟(200MHz)

```
pla_ff_cfg.val = 0;
pla_ff_cfg.sel_cfg_ff_type = pla_ff_type_level3_filter0;
pla_ff_cfg.sel_clk_source = 0;
pla_set_ff(PLA_TMGW_COUNTER, pla_chn_0, &pla_ff_cfg);
```

实际通过上述的配置，PLA CHN[0]的输出是 PLA CHN[2]和 PLA IN[0]的进行相与操作后的结果。

- PLA CHN[1]的设置
  - 16to8 AOI 的配置

    16to8_chn0:先将 16 路的 filter1out[0:15]的输入全部配置成高输出，再将 filter1out[1]的输入配置成同相输出后，配置到 16to8_chn0

    16to8_chn1:先将 16 路的 filter1out[0:15]的输入全部配置成高输出，再将 filter1out[2]的输入配置成同相输出后，配置到 16to8_chn1

    其余的 16to8_chn[2:7]均由软件设置为高。

    得到 filter2in[0:7]。

```
aoi_16to8_chn_cfg.chn = pla_chn_1;
aoi_16to8_chn_cfg.aoi_16to8_chn = pla_aoi_16to8_chn_0;
for (uint16_t i = pla_level1_filter_out_0; i <= pla_level1_filter_out_15; i++) {
  aoi_16to8_chn_cfg.input[i].op = pla_aoi_operation_or_1;
  aoi_16to8_chn_cfg.input[i].signal = i;
}
aoi_16to8_chn_cfg.input[pla_level1_filter_out_1].op = pla_aoi_operation_and_1;
aoi_16to8_chn_cfg.input[pla_level1_filter_out_1].signal = pla_level1_filter_out_1;
pla_set_aoi_16to8_one_channel(PLA_TMGW_COUNTER, &aoi_16to8_chn_cfg);

aoi_16to8_chn_cfg.aoi_16to8_chn = pla_aoi_16to8_chn_1;
```

```
for (uint16_t i = pla_level1_filter_out_0; i <= pla_level1_filter_out_15; i++) {
  aoi_16to8_chn_cfg.input[i].op = pla_aoi_operation_or_1;
  aoi_16to8_chn_cfg.input[i].signal = i;
}
aoi_16to8_chn_cfg.input[pla_level1_filter_out_2].op = pla_aoi_operation_and_1;
aoi_16to8_chn_cfg.input[pla_level1_filter_out_2].signal = pla_level1_filter_out_2;
pla_set_aoi_16to8_one_channel(PLA_TMGW_COUNTER, &aoi_16to8_chn_cfg);

/**
 * @brief pla 16to8 channel 2-7
 *
 */
for (uint16_t j = pla_aoi_16to8_chn_2; j <= pla_aoi_16to8_chn_7; j++) {
  aoi_16to8_chn_cfg.chn = pla_chn_1;
  aoi_16to8_chn_cfg.aoi_16to8_chn = j;
  for (uint16_t i = pla_level1_filter_out_0; i < pla_level1_filter_out_15; i++) {
    aoi_16to8_chn_cfg.input[i].op = pla_aoi_operation_or_1;
    aoi_16to8_chn_cfg.input[i].signal = i;
  }
  pla_set_aoi_16to8_one_channel(PLA_TMGW_COUNTER, &aoi_16to8_chn_cfg);
}
```

- FILTER2 的配置

  对于每一路的 filter2in[0:7]，均配置成边沿同步，软件注入无效，并配置成 filter2out[0:7]。

```
filter_level2_chn_cfg.val = 0;
filter_level2_chn_cfg.sync_edge_filter_disable = true;
filter_level2_chn_cfg.software_inject = pla_filter_sw_inject_disable;
for (uint16_t i = pla_filter2_chn0; i <= pla_filter2_chn7; i++) {
  pla_set_filter2(PLA_TMGW_COUNTER, pla_chn_1, i, &filter_level2_chn_cfg);
}
```

- 8to7 AOI 的配置

  通过对 filter2out 的配置，实现 8to7_chn5 即为 filter2out[0], 而 8to7_chn6 即为 filter2out[1]

  其余的 8to7_chn[x]均由软件设置为低。

```
aoi_8to7_chn_cfg.chn = pla_chn_1;
for (uint16_t i = pla_level2_filter_out_0; i <= pla_level2_filter_out_7; i++) {
  aoi_8to7_chn_cfg.input[i].op = pla_aoi_operation_and_0;
  aoi_8to7_chn_cfg.input[i].signal = i;
```

```
}
aoi_8to7_chn_cfg.aoi_8to7_chn = pla_aoi_8to7_chn_5;
aoi_8to7_chn_cfg.input[pla_level2_filter_out_0].op = pla_aoi_operation_and_1;
aoi_8to7_chn_cfg.input[pla_level2_filter_out_0].signal = pla_level2_filter_out_0;
pla_set_aoi_8to7_one_channel(PLA_TMGW_COUNTER, &aoi_8to7_chn_cfg);
for (uint16_t i = pla_level2_filter_out_0; i <= pla_level2_filter_out_7; i++) {
  aoi_8to7_chn_cfg.input[i].op = pla_aoi_operation_and_0;
  aoi_8to7_chn_cfg.input[i].signal = i;
}
aoi_8to7_chn_cfg.input[pla_level2_filter_out_1].op = pla_aoi_operation_and_1;
aoi_8to7_chn_cfg.input[pla_level2_filter_out_1].signal = pla_level2_filter_out_1;
aoi_8to7_chn_cfg.aoi_8to7_chn = pla_aoi_8to7_chn_6;
pla_set_aoi_8to7_one_channel(PLA_TMGW_COUNTER, &aoi_8to7_chn_cfg);
```

- FILTER3 的配置

  filter3out[0]为 8to7_chn0，filter3out[1]为 8to7_chn1，
  filter3out[2]为软件设置高，filter3out[3]为软件设置低，
  filter3out[4]为软件设置高，filter3out[5]为 8to7_chn5，
  filter3out[6]为 8to7_chn6

```
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.sync_edge_filter_disable = true;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_disable;
for (uint16_t i = pla_filter3_chn0; i <= pla_filter3_chn6; i++) {
  pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_1, i, &filter_level3_chn_cfg);
}
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_1, pla_filter3_chn2, &filter_level3_chn_cfg);
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_low;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_1, pla_filter3_chn3, &filter_level3_chn_cfg);
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_1, pla_filter3_chn4, &filter_level3_chn_cfg);
```

- CFF 的设置

  CFF 设置为 JK 触发器，时钟为 PLA 的系统时钟（200MHz）。

```
pla_ff_cfg.val = 0;
pla_ff_cfg.sel_cfg_ff_type = pla_ff_type_jk_ff;
pla_ff_cfg.sel_clk_source = 0;
```

```
pla_set_ff(PLA_TMGW_COUNTER, pla_chn_1, &pla_ff_cfg);
```

实际 PLA CHN[1]实现的是通过 JK 触发器, J 为 MOTOR_TRIG_IN，K 为 QEI_IN。当 J=1，K=0 时，PLA_CHN[1]在时钟的上升沿变为 1，当 J=0，K=1 时，PLA_CHN[1]在时钟的上升沿变为 0。这个信号通过 trigmux 连接到了 PB21，进而控制 485 收发器的 DR。

- PLA CHN[2]的设置
  - 16to8 AOI 的配置

    16to8_chn0:先将 16 路的 filter1out[0:15]的输入全部配置成高输出，再将 filter1out[9](即 PLA_CHN[1])的输入配置成同相输出后，配置到 16to8_chn0

    16to8_chn1:先将 16 路的 filter1out[0:15]的输入全部配置成高输出，再将 filter1out[12](即 PLA_CHN[4])的输入配置成同相输出后，配置到 16to8_chn1

    其余的 16to8_chn[2:7]均由软件设置为高。得到 filter2in[0:7]。

```
aoi_16to8_chn_cfg.chn = pla_chn_2;
aoi_16to8_chn_cfg.aoi_16to8_chn = pla_aoi_16to8_chn_0;
for (uint16_t i = pla_level1_filter_out_0; i <= pla_level1_filter_out_15; i++) {
  aoi_16to8_chn_cfg.input[i].op = pla_aoi_operation_or_1;
  aoi_16to8_chn_cfg.input[i].signal = i;
}
aoi_16to8_chn_cfg.input[pla_level1_filter_out_9].op = pla_aoi_operation_and_1;
aoi_16to8_chn_cfg.input[pla_level1_filter_out_9].signal = pla_level1_filter_out_9;
pla_set_aoi_16to8_one_channel(PLA_TMGW_COUNTER, &aoi_16to8_chn_cfg);

aoi_16to8_chn_cfg.aoi_16to8_chn = pla_aoi_16to8_chn_1;
for (uint16_t i = pla_level1_filter_out_0; i <= pla_level1_filter_out_15; i++) {
  aoi_16to8_chn_cfg.input[i].op = pla_aoi_operation_or_1;
  aoi_16to8_chn_cfg.input[i].signal = i;
}
aoi_16to8_chn_cfg.input[pla_level1_filter_out_12].op = pla_aoi_operation_and_1;
aoi_16to8_chn_cfg.input[pla_level1_filter_out_12].signal = pla_level1_filter_out_12;
pla_set_aoi_16to8_one_channel(PLA_TMGW_COUNTER, &aoi_16to8_chn_cfg);


/**
 * @brief pla 16to8 channel 2-7
 *
 */
for (uint16_t j = pla_aoi_16to8_chn_2; j <= pla_aoi_16to8_chn_7; j++) {
  aoi_16to8_chn_cfg.chn = pla_chn_2;
```

```
    aoi_16to8_chn_cfg.aoi_16to8_chn = j;
    for (uint16_t i = pla_level1_filter_out_0; i < pla_level1_filter_out_15; i++) {
        aoi_16to8_chn_cfg.input[i].op = pla_aoi_operation_or_1;
        aoi_16to8_chn_cfg.input[i].signal = i;
    }
    pla_set_aoi_16to8_one_channel(PLA_TMGW_COUNTER, &aoi_16to8_chn_cfg);
}
```

- FILTER2 的配置

  对于每一路的 filter2in[0:7]，均配置成边沿同步，软件注入无效，并配置成 filter2out[0:7]。

```
filter_level2_chn_cfg.val = 0;
filter_level2_chn_cfg.sync_edge_filter_disable = true;
filter_level2_chn_cfg.software_inject = pla_filter_sw_inject_disable;
for (uint16_t i = pla_filter2_chn0; i <= pla_filter2_chn7; i++) {
    pla_set_filter2(PLA_TMGW_COUNTER, pla_chn_2, i, &filter_level2_chn_cfg);
}
```

- 8to7 AOI 的配置

  通过对 filter2out 的配置，实现 8to7_chn0 即为 filter2out[0]与 filter2out[1]进行或操作后输出。

  其余的 8to7_chn[x]均由软件设置为低。

```
aoi_8to7_chn_cfg.chn = pla_chn_2;
for (uint16_t i = pla_level2_filter_out_0; i <= pla_level2_filter_out_7; i++) {
    aoi_8to7_chn_cfg.input[i].op = pla_aoi_operation_and_0;
    aoi_8to7_chn_cfg.input[i].signal = i;
}
aoi_8to7_chn_cfg.aoi_8to7_chn = pla_aoi_8to7_chn_0;
aoi_8to7_chn_cfg.input[pla_level2_filter_out_0].op = pla_aoi_operation_and_1;
aoi_8to7_chn_cfg.input[pla_level2_filter_out_0].signal = pla_level2_filter_out_0;
aoi_8to7_chn_cfg.input[pla_level2_filter_out_1].op = pla_aoi_operation_and_1;
aoi_8to7_chn_cfg.input[pla_level2_filter_out_1].signal = pla_level2_filter_out_1;
pla_set_aoi_8to7_one_channel(PLA_TMGW_COUNTER, &aoi_8to7_chn_cfg);
```

- FILTER3 的配置

  filter3out[0]为 8to7_chn0，filter3out[1]为 8to7_chn1

  filter3out[2]为软件设置高，filter3out[3]为软件设置低

filter3out[4]为软件设置高，filter3out[5]为 8to7_chn5，

filter3out[6]为 8to7_chn6

```
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.sync_edge_filter_disable = true;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_disable;
for (uint16_t i = pla_filter3_chn0; i <= pla_filter3_chn6; i++) {
  pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_2, i, &filter_level3_chn_cfg);
}
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_2, pla_filter3_chn2, &filter_level3_chn_cfg);
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_low;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_2, pla_filter3_chn3, &filter_level3_chn_cfg);
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_2, pla_filter3_chn4, &filter_level3_chn_cfg);
```

- CFF 的设置

  配置成了直接输出，即 filter3out[0]，时钟为 PLA 系统时钟。

```
pla_ff_cfg.val = 0;
pla_ff_cfg.sel_cfg_ff_type = pla_ff_type_level3_filter0;
pla_ff_cfg.sel_clk_source = 0;
pla_set_ff(PLA_TMGW_COUNTER, pla_chn_2, &pla_ff_cfg);
```

实际 PLA CHN[2]实现的是或门。两个输入分别为 PLA_CHN[1]和 PLA_CHN[4]。

- PLA CHN[4]的设置
  - 16to8 AOI 的配置

    16to8_chn0:先将 16 路的 filter1out[0:15]的输入全部配置成高输出，再将 filter1out[3]的输入配置成同相输出后，配置到 16to8_chn0

    16to8_chn1:先将 16 路的 filter1out[0:15]的输入全部配置成高输出，再将 filter1out[14](即 PLA_CHN[6])的输入配置成同相输出后，配置到 16to8_chn1

    其余的 16to8_chn[2:7]均由软件设置为高。得到 filter2in[0:7]。

```
aoi_16to8_chn_cfg.chn = pla_chn_4;
aoi_16to8_chn_cfg.aoi_16to8_chn = pla_aoi_16to8_chn_0;
for (uint16_t i = pla_level1_filter_out_0; i <= pla_level1_filter_out_15; i++) {
```

```
  aoi_16to8_chn_cfg.input[i].op = pla_aoi_operation_or_1;
  aoi_16to8_chn_cfg.input[i].signal = i;
}
aoi_16to8_chn_cfg.input[pla_level1_filter_out_3].op = pla_aoi_operation_and_1;
aoi_16to8_chn_cfg.input[pla_level1_filter_out_3].signal = pla_level1_filter_out_3;
pla_set_aoi_16to8_one_channel(PLA_TMGW_COUNTER, &aoi_16to8_chn_cfg);

aoi_16to8_chn_cfg.aoi_16to8_chn = pla_aoi_16to8_chn_1;
for (uint16_t i = pla_level1_filter_out_0; i <= pla_level1_filter_out_15; i++) {
  aoi_16to8_chn_cfg.input[i].op = pla_aoi_operation_or_1;
  aoi_16to8_chn_cfg.input[i].signal = i;
}
aoi_16to8_chn_cfg.input[pla_level1_filter_out_14].op = pla_aoi_operation_and_1;
aoi_16to8_chn_cfg.input[pla_level1_filter_out_14].signal = pla_level1_filter_out_14;
pla_set_aoi_16to8_one_channel(PLA_TMGW_COUNTER, &aoi_16to8_chn_cfg);

/**
 * @brief pla 16to8 channel 2-7
 *
 */
for (uint16_t j = pla_aoi_16to8_chn_2; j <= pla_aoi_16to8_chn_7; j++) {
  aoi_16to8_chn_cfg.chn = pla_chn_4;
  aoi_16to8_chn_cfg.aoi_16to8_chn = j;
  for (uint16_t i = pla_level1_filter_out_0; i < pla_level1_filter_out_15; i++) {
    aoi_16to8_chn_cfg.input[i].op = pla_aoi_operation_or_1;
    aoi_16to8_chn_cfg.input[i].signal = i;
  }
  pla_set_aoi_16to8_one_channel(PLA_TMGW_COUNTER, &aoi_16to8_chn_cfg);
}
```

- FILTER2 的配置

  对于每一路的 filter2in[0:7]，均配置成边沿同步，软件注入无效，并配置成 filter2out[0:7]。

```
filter_level2_chn_cfg.val = 0;
filter_level2_chn_cfg.sync_edge_filter_disable = true;
filter_level2_chn_cfg.software_inject = pla_filter_sw_inject_disable;
for (uint16_t i = pla_filter2_chn0; i <= pla_filter2_chn7; i++) {
  pla_set_filter2(PLA_TMGW_COUNTER, pla_chn_4, i, &filter_level2_chn_cfg);
}
```

- 8to7 AOI 的配置

通过对 filter2out 的配置，实现 8to7_chn1 即为 filter2out[0]输出。实现 8to7_chn2 即为 filter2out[1]输出。

其余的 8to7_chn[x]均由软件设置为低。

```
aoi_8to7_chn_cfg.chn = pla_chn_4;
for (uint16_t i = pla_level2_filter_out_0; i <= pla_level2_filter_out_7; i++) {
  aoi_8to7_chn_cfg.input[i].op = pla_aoi_operation_and_0;
  aoi_8to7_chn_cfg.input[i].signal = i;
}
aoi_8to7_chn_cfg.aoi_8to7_chn = pla_aoi_8to7_chn_1;
aoi_8to7_chn_cfg.input[pla_level2_filter_out_0].op = pla_aoi_operation_and_1;
aoi_8to7_chn_cfg.input[pla_level2_filter_out_0].signal = pla_level2_filter_out_0;
pla_set_aoi_8to7_one_channel(PLA_TMGW_COUNTER, &aoi_8to7_chn_cfg);

aoi_8to7_chn_cfg.chn = pla_chn_4;
for (uint16_t i = pla_level2_filter_out_0; i <= pla_level2_filter_out_7; i++) {
  aoi_8to7_chn_cfg.input[i].op = pla_aoi_operation_and_0;
  aoi_8to7_chn_cfg.input[i].signal = i;
}
aoi_8to7_chn_cfg.aoi_8to7_chn = pla_aoi_8to7_chn_2;
aoi_8to7_chn_cfg.input[pla_level2_filter_out_1].op = pla_aoi_operation_and_1;
aoi_8to7_chn_cfg.input[pla_level2_filter_out_1].signal = pla_level2_filter_out_1;
pla_set_aoi_8to7_one_channel(PLA_TMGW_COUNTER, &aoi_8to7_chn_cfg);
```

- FILTER3 的配置

  filter3out[0]为软件设置高，filter3out[1]为 8to7_chn1，

  filter3out[2]为 8to7_chn2，filter3out[3]为 8to7_chn3，

  filter3out[4]为 8to7_chn4，filter3out[5]为软件设置高，

  filter3out[6]为软件设置低

```
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.sync_edge_filter_disable = true;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_disable;
for (uint16_t i = pla_filter3_chn0; i <= pla_filter3_chn6; i++) {
  pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_4, i, &filter_level3_chn_cfg);
}
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_4, pla_filter3_chn0, &filter_level3_chn_cfg);
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_4, pla_filter3_chn5, &filter_level3_chn_cfg);
```

```
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_low;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_4, pla_filter3_chn6, &filter_level3_chn_cfg);
```

- CFF 的设置

  配置成了锁存器,时钟为 filter3out[1]

```
pla_ff_cfg.val = 0;
pla_ff_cfg.sel_cfg_ff_type = pla_ff_type_latch;
pla_ff_cfg.sel_clk_source = 1;
pla_set_ff(PLA_TMGW_COUNTER, pla_chn_4, &pla_ff_cfg);
```

实际 PLA_CHN[4]实现的锁存器功能。两个输入分别为 PLA_IN[3](SPI_MOSI)和 PLA_CHN[6]，当 PLA_CHN[6]为低时，PLA_CHN[4]输出为低；当 PLA_CHN[6]为高时，PLA_IN[3]的上升沿触发（注意不是实际的 SPI_MOSI 上的信号），使得 PLA_CHN[4]变高。

- PLA CHN[6]的设置
  - 16to8 AOI 的配置

    16to8_chn0:先将 16 路的 filter1out[0:15]的输入全部配置成高输出，再将 filter1out[1]的输入配置成同相输出后，配置到 16to8_chn0

    其余的 16to8_chn[1:7]均由软件设置为高。得到 filter2in[0:7]。

```
aoi_16to8_chn_cfg.chn = pla_chn_6;
aoi_16to8_chn_cfg.aoi_16to8_chn = pla_aoi_16to8_chn_0;
for (uint16_t i = pla_level1_filter_out_0; i <= pla_level1_filter_out_15; i++) {
  aoi_16to8_chn_cfg.input[i].op = pla_aoi_operation_or_1;
  aoi_16to8_chn_cfg.input[i].signal = i;
}
aoi_16to8_chn_cfg.input[pla_level1_filter_out_1].op = pla_aoi_operation_and_1;
aoi_16to8_chn_cfg.input[pla_level1_filter_out_1].signal = pla_level1_filter_out_1;
pla_set_aoi_16to8_one_channel(PLA_TMGW_COUNTER, &aoi_16to8_chn_cfg);

/**
 * @brief pla 16to8 channel 1-7
 *
 */
for (uint16_t j = pla_aoi_16to8_chn_1; j <= pla_aoi_16to8_chn_7; j++) {
  aoi_16to8_chn_cfg.chn = pla_chn_6;
```

```
    aoi_16to8_chn_cfg.aoi_16to8_chn = j;
    for (uint16_t i = pla_level1_filter_out_0; i < pla_level1_filter_out_15; i++) {
        aoi_16to8_chn_cfg.input[i].op = pla_aoi_operation_or_1;
        aoi_16to8_chn_cfg.input[i].signal = i;
    }
    pla_set_aoi_16to8_one_channel(PLA_TMGW_COUNTER, &aoi_16to8_chn_cfg);
}
```

- FILTER2 的配置

  先将每一路的 filter2in[0:7]，均配置成边沿同步，软件注入无效，并配置成 filter2out[0:7]。然后再将 filter2in[0]配置成上升沿有效。

```
filter_level2_chn_cfg.val = 0;
filter_level2_chn_cfg.sync_edge_filter_disable = true;
filter_level2_chn_cfg.software_inject = pla_filter_sw_inject_disable;
for (uint16_t i = pla_filter2_chn0; i <= pla_filter2_chn7; i++) {
    pla_set_filter2(PLA_TMGW_COUNTER, pla_chn_6, i, &filter_level2_chn_cfg);
}
filter_level2_chn_cfg.val = 0;
filter_level2_chn_cfg.sync_edge_filter_disable = false;
filter_level2_chn_cfg.edge_dect_en = true;
filter_level2_chn_cfg.pose_edge_dect_en = true;
filter_level2_chn_cfg.filter_reverse = false;
filter_level2_chn_cfg.software_inject = pla_filter_sw_inject_disable;
pla_set_filter2(PLA_TMGW_COUNTER, pla_chn_6, pla_filter2_chn0, &filter_level2_chn_cfg);
```

- 8to7 AOI 的配置

  通过对 filter2 的配置，实现 8to7_chn0 即为 filter2out[0]输出。

  其余的 8to7_chn[x]均由软件设置为低。

```
aoi_8to7_chn_cfg.chn = pla_chn_6;
for (uint16_t i = pla_level2_filter_out_0; i <= pla_level2_filter_out_7; i++) {
    aoi_8to7_chn_cfg.input[i].op = pla_aoi_operation_and_0;
    aoi_8to7_chn_cfg.input[i].signal = i;
}
aoi_8to7_chn_cfg.aoi_8to7_chn = pla_aoi_8to7_chn_0;
aoi_8to7_chn_cfg.input[pla_level2_filter_out_0].op = pla_aoi_operation_and_1;
aoi_8to7_chn_cfg.input[pla_level2_filter_out_0].signal = pla_level2_filter_out_0;
pla_set_aoi_8to7_one_channel(PLA_TMGW_COUNTER, &aoi_8to7_chn_cfg);
```

- FILTER3 的配置

  filter3out[0]为 8to7_chn0，filter3out[1]为 8to7_chn1(实际为低)

  filter3out[2]为软件设置高，filter3out[3]为软件设置低，

  filter3out[4]为软件设置高，filter3out[5]为软件设置高，

  filter3out[6]为软件设置低

```
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.sync_edge_filter_disable = true;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_disable;
for (uint16_t i = pla_filter3_chn0; i <= pla_filter3_chn6; i++) {
  pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_6, i, &filter_level3_chn_cfg);
}
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_6, pla_filter3_chn2, &filter_level3_chn_cfg);
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_low;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_6, pla_filter3_chn3, &filter_level3_chn_cfg);
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_6, pla_filter3_chn4, &filter_level3_chn_cfg);
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_6, pla_filter3_chn5, &filter_level3_chn_cfg);
filter_level3_chn_cfg.val = 0;
filter_level3_chn_cfg.software_inject = pla_filter_sw_inject_low;
pla_set_filter3(PLA_TMGW_COUNTER, pla_chn_6, pla_filter3_chn6, &filter_level3_chn_cfg);
```

- CFF 的设置

  配置成了加法器,时钟为 PLA 系统时钟。

```
pla_ff_cfg.val = 0;
pla_ff_cfg.sel_cfg_ff_type = pla_ff_type_adder_minus;
pla_ff_cfg.sel_adder_minus = 0;
pla_ff_cfg.sel_clk_source = 0;
pla_set_ff(PLA_TMGW_COUNTER, pla_chn_6, &pla_ff_cfg);
```

实际 PLA CHN[6]实现的是加法器。其输入为 PLA_IN[1](QEI_IN),当 QEI_IN 出现一个上升沿的时候，信号翻转。

PLA 的使能与触发

- 使能每一路 PLA CHN

```
pla_channel_enable(BOARD_PLA_COUNTER, pla_chn_0);

pla_channel_enable(BOARD_PLA_COUNTER, pla_chn_1);
pla_channel_enable(BOARD_PLA_COUNTER, pla_chn_2);
pla_channel_enable(BOARD_PLA_COUNTER, pla_chn_3);
pla_channel_enable(BOARD_PLA_COUNTER, pla_chn_4);
pla_channel_enable(BOARD_PLA_COUNTER, pla_chn_5);
pla_channel_enable(BOARD_PLA_COUNTER, pla_chn_6);
pla_channel_enable(BOARD_PLA_COUNTER, pla_chn_7);
```

4.3 例程逻辑

经过上述的设置，例程实际上要实现的逻辑如下图

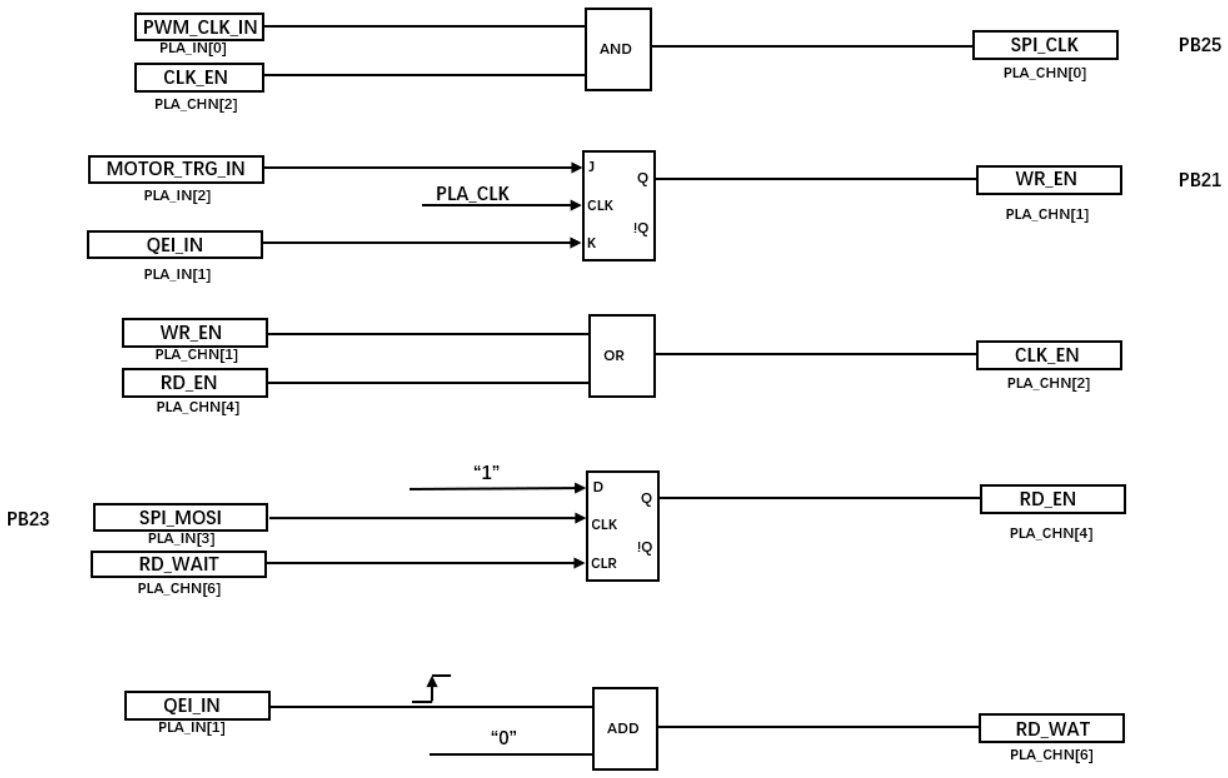

图 3

基于上述的设计，

- 在初始的时候，PWM_CLK_IN 是一个 2.5MHz 的时钟源。CLK_EN 为低，此时

SPI_CLK 被置低。

- 当 MOTOR_TRIG_IN 出现高电平后，经过 JK 触发器，使得 WR_EN 置高。此时 CLK_EN 变为高，SPI_CLK 输出，数据从 MCU 的 MISO 引脚发送到 485 收发器的接收端。
- 同时 QEI_IN 捕捉 SPI_CLK 信号并进行计数。当达到计数值时（需要发出的数据的 bit 数），QEI_IN 发出一个高脉冲信号，经过 JK 触发器，使得 WR_EN 变低。进而导致 SPI_CLK 不再输出。同时 QEI_IN 的上升沿会触发加法器，使得 RD_WAIT 为高。
- SPI_MOSI 的信号线经由 filter1 的处理，在出现下降沿时，会生成一个脉冲，形成上升沿。锁存器经过上升沿的触发，使得 RD_EN 置高。进而使得 SPI_CLK 输出。使得 MCU 的 MISO 可以接收到编码器发出的数据。
- 同时 QEI_IN 开始捕捉 SPI_CLK 信号并进行计数。当达到计数值时（需要收到的数据的 bit 数），QEI_IN 发出一个高脉冲信号，并触发加法器，使得 RD_WAIT 为低。
- RD_WAIT 为低经过锁存器，使得 RD_EN 为低，进而使得 SPI_CLK 不在输出。
- CLK_EN 会触发 HALL 的计数器，用以计算数据的收发之间的时间。
- RD_WAIT 信号会触发 DMA CHAIN 刷新，完成相应的配置。

5 总结

在实际的编码中，通过对 PLA 本身的灵活使用，可以完成不同的需求。本文希望通过对 SDK 里面的相关例程的介绍，用户能够拓展对不同编码器的使用。