HPM6200 系列

HPM6200 系列微控制器 PLA 使用介绍

先楫半导体HPM6200系列微控制器PLA使用介绍

目录

版本：

日期	版本号	说明
2023-4-13	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。

2 PLA 功能介绍

HPM6200 包含了 2 个 PLA，分别为 PLA0, PLA1。2 个 PLA 功能完全一致。

每一个 PLA 单元最大可以支持 8 路输入和 8 路输出组成。PLA 的输入单元和输出单元都是由 TRGM 来控制。也就是说，当用户使用 PLA 的时候，相应的输入与输出引脚必须有 trigmux 的功能。

PLA 的寄存器由 3 个部分组成：

- 输入配置部分
- 输出通道配置部分
- 输出通道使能部分

每路输出可以单独使能。对于没有使能的通道，其对应的输出通道配置部分的寄存器，用户可以将其当作通用寄存器进行使用。当整个 PLA 均未使用的时候，输出通道配置部分和输入配置部分的寄存器，总计 1KB，可以视作通用寄存器。

2.1 PLA 的输入

PLA 的输入包括 2 个部分，共 16 路。

- PLA IN[0:7]
- PLA OUT[0:7]

其中 PLA IN 为 8 路输入，由用户通过 trigmux 指定其输入源。因此，请不必将输入局限

在芯片的物理引脚上。PLA OUT 实际为 PLA 的 8 路输出，即 PLA CHN[0:7]，这 8 路输出直接反馈到输入端，与 PLA IN 一起形成 16 路输入源。如下图 1 所示。16 路的输入源并不是全部必须的，理论上只需要确定 1 路输入即可完成 PLA 的功能。

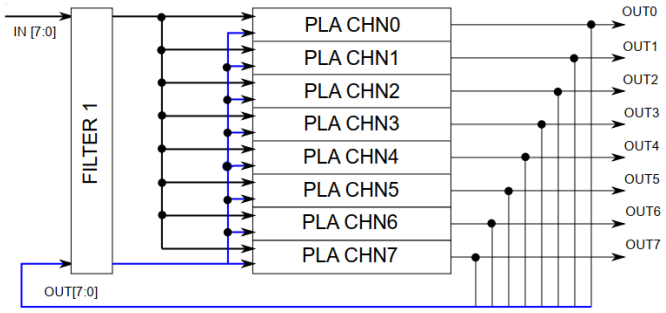

图 1

2.2 PLA 的结构

一个 PLA 包括 3 个过滤器和 2 个逻辑电路阵列以及 1 个触发器电路间隔组合而成。分别命名为

- 过滤器 1（FILTER1）
- 16 输入 8 输出与逻辑阵列（16to8 AOI）
- 过滤器 2（FILTER2）
- 8 输入 7 输出或逻辑阵列（8to7 AOI）
- 过滤器 3（FILTER3）
- 可配置触发器（CFF）

以上顺序即为 PLA 单元内部的信号顺序。在配置好 FILTER1 之后，每一路 PLA 的输出 PLA CHN[x]都需要对剩余的部分单独设置而成。

2.2.1 过滤器（FILTER）

不同过滤模块的功能是一致的，都是输入信号同步，边沿检测，软件注入，滤波后得到输出信号。如图 2

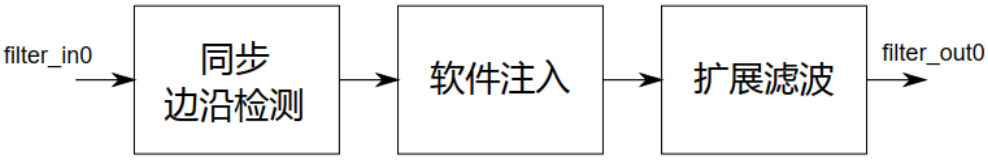

图 2

对于 FILTER1 来说，其 16 路输入分别为

- filter1in[0:7]与 PLA IN[0:7]一一对应。
- filter1in[8:15]与 PLA OUT[0:7](PLA CHN[0:7])一一对应。

其 16 路输出 filter1out[0:15]即为 16to8 AOI 的输入。

对于 FILTER2 来说，其 8 路输入即为 16to8 AOI 的输出，其 8 路输出即为 8to7 AOI 的输入。

对于 FILTER3 来说，其 7 路输入即为 8to7 AOI 的输出，其 7 路输出即为 CFF 的输入。

对于过滤器的配置，如前所说，

- FILTER1, 配置寄存器为 FILTER_1ST_PLA_IN[0:7]和 FILTER_1ST_PLA_OUT[0:7]
- FILTER2 和 FILTER3 的配置则需要根据不同的 PLA 输出通道单独配置。即 CHN[x].FILTER_2ND[0:7]，CHN[x].FILTER_3RD[0:6]。

过滤器的扩展滤波包括以下几个类型

- 输入高电平扩展

  这个指的是，当输入的信号是高时，输出将会根据扩展的计数（FILTER_EXT_COUNTER）进行扩展，当到达计数时，采样输入信号，此时如果为低，则输出为低，此时如果为高，则继续扩展相应的计数后，再采样输入信号。扩展期间的输入变化，不会影响输出。
- 输入低电平扩展

  这个指的是，当输入的信号是低时，输出将会根据扩展的计数（FILTER_EXT_COUNTER）进行扩展，当到达计数时，采样输入信号，此时如果为高，则输出为高，此时如果为低，则继续扩展相应的计数后，再采样输入信号。扩展期间的输入变化，不会影响输出。
- 输出状态扩展

这个指的是，当输入的信号的变化周期在扩展的计数（FILTER_EXT_COUNTER）之内时，输出不会发生变化。当输入的信号的长度周期超过扩展的计数时，输出将和输入的状态保持一致。

- 输入跳变扩展

  这个指的是，当输入的信号发生跳变后，输出将会随之变化，但在扩展的计数周期（FILTER_EXT_COUNTER）之内时，无论输入发生什么变化，输出都不会发生变化。当周期超过扩展的计数时，只有输入发生跳变，输出才会发生变化。

2.2.2 16 输入 8 输出与逻辑阵列（16to8 AOI）

该逻辑阵列的输入来自于 FILTER1 的输出 filter1out[0：15]，该逻辑阵列的输出为 16to8chn[0:7]。每一路输出均是由 16 路输入设置组成。下图 3 是 16to8chn0 的逻辑解释

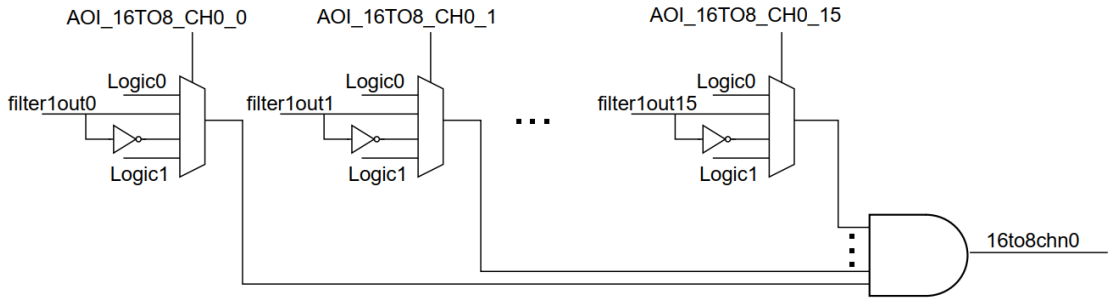

图 3

用户需要在寄存器中配置 AOI_16TO8_CH0_x，通过对 16 路 filter1out[0:15]的逻辑设置，得到相应的 AOI_16TO8_CH0_x。然后这 16 路 AOI_16TO8_CH0_[0:15]经过与操作得到 16to8cn0。

2.2.3 8 输入 7 输出或逻辑阵列（8to7 AOI）

该逻辑阵列的输入来自于 FILTER2 的输出 filter2out[0:7]，该逻辑阵列的输出为 8to7chn[0:6]。每一路输出均是由 8 路输入设置组成。下图 4 是 8to7chn0 的逻辑解释

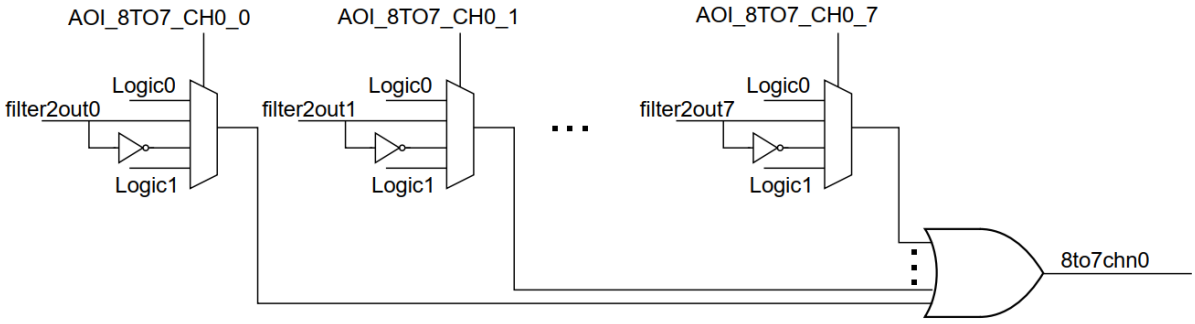

图 4

用户需要在寄存器中配置 AOI_8TO7_CHN0_x，通过对 8 路的 filter2out[0:7]的逻辑设置，得到相应的 AOI_8TO7_CHN0_x。然后这 8 路 AOI_8TO7_CHN0_[0:7]经过或操作得到 8to7chn0。

2.2.4 可配置触发器（CFF）

该逻辑阵列的输入来自于 FILTER3 的输出 filter3out[0:6]，该逻辑阵列的输出即为 PLA 所指定的输出，PLA CHNx。

可配置触发器 CFF 可以配置为

- D 触发器
- 双边沿 D 触发器
- T 触发器
- JK 触发器
- 锁存器
- 运算器
- 不配置，旁路输出 filer3out0

用户通过对 CHN[x].CFG_FF 的配置，来实现相应的选择。而 7 路输入 filter3out[0:6]的分配如下图 5 所示。

输入信号	D 触发器	双边沿 D 触发器	T 触发器	JK 触发器	锁存器	运算器
filter3out0	输入 D	输入 D	输入 T	N/A	输入 D	加数/减数
filter3out1	时钟	时钟	时钟	时钟	时钟	时钟
filter3out2	异步复位	异步复位	异步复位	异步复位	异步复位	异步复位
filter3out3	异步置位	异步置位	异步置位	异步置位	N/A	异步置位
filter3out4	使能	使能	使能	使能	N/A	使能
filter3out5	同步复位	同步复位	同步复位	输入 K	同步复位	同步复位
filter3out6	同步置位	同步置位	同步置位	输入 J	同步置位	同步置位

图 5

2.3 PLA 的配置

由于 PLA 的输入和输出和 trigmux 相关，因此用户可以参看 SDK 中的例程了解 trigmux 的工作原理。相应部分不在本文的讲述范围之内。对于 PLA 本身的配置，用户在设置了章节 2.2 中的相应的寄存器后，通过配置 CHN_CFG_ACTIVE[x]来使能或者关闭相应的 PLA CHN[x]的输出。

3 PLA 例程分析

由于 PLA 是 HPM6200 所具备的功能，所以用户需要下载 SDKv1.1.0 或以上版本。
本文中所提到的例程基于 SDK v1.1.0， 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。

3.1 例程介绍

在 SDKv1.1.0 中，PLA 的例程位于
..\..\sdk_env_v1.1.0\hpm_sdk\samples\drivers\pla
例程中，将 PB25 配置成 TRGM0_P5 作为 PLA 的输出，
init_pla_pins()。

通过 TRGM，将 PWM 信号配置成 PLA IN0，将 PLA CHN0 配置到 TRGM0_P5(PB25)，

init_pwm_pla_trgm(BOARD_PLA_TRGM)。

通过配置 PWM， 将一路 PWM 使能，输出频率是 1KHz，占空比为 50%。

init_pwm()。

通过上述配置，完成 PLA 输入输出的配置。准确的讲，是 PLA 基于 TRGM 的输入输出配置。用户可以在 TRGM 所关联的引脚上，得到需要的输出。

3.2 PLA 设置分析

例程通过 **pla_7bit_counter_init()**对 PLA 进行设置。在这个配置中，主要包括了 2 个部分。一个是对 PLA CHN0 的配置，另一个是对 PLA CHN[1:7]的配置。

- FILTER1 的配置

  例程将 filter1in0(PLA IN0)配置成边沿同步，软件注入无效，将 filter1in[1:7](PLA IN[1:7])配置软件注入为高，将 filter1in[8:15](PLA OUT[0:7])配置成边沿同步，软件注入无效。基于上述的配置，实际上 filter1out[0:15]就是 filter1in[0]和 filter1in[8:15]这 9 路的输入，其余为高。

```
filter_1st_chn_cfg.val = 0;
filter_1st_chn_cfg.sync_edge_filter_disable = true;
filter_1st_chn_cfg.software_inject = pla_filter_sw_inject_disable;
pla_set_filter1_in(BOARD_PLA_COUNTER, BOARD_PLA_PWM_IN_CHN, &filter_1st_chn_cfg);
filter_1st_chn_cfg.val = 0;
filter_1st_chn_cfg.software_inject = pla_filter_sw_inject_height;
for (uint16_t i = 1; i <= BOARD_PLA_1ST_FILTER_IN_END; i++) {
  pla_set_filter1_in(BOARD_PLA_COUNTER, i, &filter_1st_chn_cfg);
}
filter_1st_chn_cfg.val = 0;
filter_1st_chn_cfg.sync_edge_filter_disable = true;
filter_1st_chn_cfg.software_inject = pla_filter_sw_inject_disable;
for (uint16_t i = 0; i <= BOARD_PLA_1ST_FILTER_OUT_END; i++) {
  pla_set_filter1_out(BOARD_PLA_COUNTER, i, &filter_1st_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[9:15]的输入配置成反相输出后，配置到 16to8_chn0

实际是将 filter1out[9:15]每一路取反后再进行与操作，作为 16to8_chn0 的输出。

```
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_1st_filter_out_0; i <= pla_1st_filter_out_15; i++) {
  aoi_16to8_chn_cfg.input[i].op = pla_aoi_operation_or_1;
  aoi_16to8_chn_cfg.input[i].signal = i;
}

for (uint16_t i = pla_1st_filter_out_9; i <= pla_1st_filter_out_15; i++) {
  aoi_16to8_chn_cfg.input[i].op = pla_aoi_operation_xor_1;
  aoi_16to8_chn_cfg.input[i].signal = i;
}
pla_set_aoi_16to8_one_channel(BOARD_PLA_COUNTER, &aoi_16to8_chn_cfg);
```

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

实际是将 filter1out[0]的反相作为 16to8_chn1 的输出。

```
aoi_16to8_chn_cfg.chn = pla_chn_0;
aoi_16to8_chn_cfg.aoi_16to8_chn = pla_aoi_16to8_chn_1;
for (uint16_t i = pla_1st_filter_out_0; i <= pla_1st_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_1st_filter_out_0].op = pla_aoi_operation_xor_1;
aoi_16to8_chn_cfg.input[pla_1st_filter_out_0].signal = pla_1st_filter_out_0;
pla_set_aoi_16to8_one_channel(BOARD_PLA_COUNTER, &aoi_16to8_chn_cfg);
```

16to8_chn2:先将 16 路的 filter1out[0:15]的输入全部配置成高输出，再将 filter1out[8]的输入配置成反相输出后，配置到 16to8_chn2

实际是将 filter1out[8]的反相作为 16to8_chn2 的输出。

```
aoi_16to8_chn_cfg.chn = pla_chn_0;
aoi_16to8_chn_cfg.aoi_16to8_chn = pla_aoi_16to8_chn_2;
for (uint16_t i = pla_1st_filter_out_0; i <= pla_1st_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_1st_filter_out_8].op = pla_aoi_operation_xor_1;
aoi_16to8_chn_cfg.input[pla_1st_filter_out_8].signal = pla_1st_filter_out_8;
pla_set_aoi_16to8_one_channel(BOARD_PLA_COUNTER, &aoi_16to8_chn_cfg);
```

16to8_chn[3:7]:先将 16 路的 filter1out[0:15]的输入全部配置成高输出，再配置到 16to8_chn[3:7]，即 16to8_chn[3:7]输出均为高。

```
for (uint16_t j = pla_aoi_16to8_chn_3; 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_1st_filter_out_0; i < pla_1st_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(BOARD_PLA_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_2st_chn_cfg.val = 0;
filter_2st_chn_cfg.sync_edge_filter_disable = true;
filter_2st_chn_cfg.software_inject = pla_filter_sw_inject_disable;
for (uint16_t i = pla_filter2_chn0; i <= pla_filter2_chn7; i++) {
    pla_set_filter2(BOARD_PLA_COUNTER, pla_chn_0, i, &filter_2st_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]以及 filter2out[1]的输入配置成同相输出，然后配置到 8to7_chn0，实际将 filter2out[0]与 filter2out[1]进行或操作形成 8to7_chn0。

```
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_2st_filter_out_0; i <= pla_2st_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_2st_filter_out_0].op = pla_aoi_operation_and_1;
aoi_8to7_chn_cfg.input[pla_2st_filter_out_0].signal = pla_2st_filter_out_0;
aoi_8to7_chn_cfg.input[pla_2st_filter_out_1].op = pla_aoi_operation_and_1;
aoi_8to7_chn_cfg.input[pla_2st_filter_out_1].signal = pla_2st_filter_out_1;
pla_set_aoi_8to7_one_channel(BOARD_PLA_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_3st_chn_cfg.val = 0;
filter_3st_chn_cfg.sync_edge_filter_disable = true;
filter_3st_chn_cfg.software_inject = pla_filter_sw_inject_disable;
for (uint16_t i = pla_filter3_chn0; i < pla_filter3_chn1; i++) {
  pla_set_filter3(BOARD_PLA_COUNTER, pla_chn_0, i, &filter_3st_chn_cfg);
}
filter_3st_chn_cfg.val = 0;
filter_3st_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(BOARD_PLA_COUNTER, pla_chn_0, pla_filter3_chn2, &filter_3st_chn_cfg);
filter_3st_chn_cfg.val = 0;
filter_3st_chn_cfg.software_inject = pla_filter_sw_inject_low;
pla_set_filter3(BOARD_PLA_COUNTER, pla_chn_0, pla_filter3_chn3, &filter_3st_chn_cfg);
filter_3st_chn_cfg.val = 0;
filter_3st_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(BOARD_PLA_COUNTER, pla_chn_0, pla_filter3_chn4, &filter_3st_chn_cfg);
filter_3st_chn_cfg.val = 0;
filter_3st_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(BOARD_PLA_COUNTER, pla_chn_0, pla_filter3_chn5, &filter_3st_chn_cfg);
filter_3st_chn_cfg.val = 0;
filter_3st_chn_cfg.software_inject = pla_filter_sw_inject_low;
pla_set_filter3(BOARD_PLA_COUNTER, pla_chn_0, pla_filter3_chn6, &filter_3st_chn_cfg);
```

- CFF 的配置

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

```
pla_ff_cfg.val = 0;
pla_ff_cfg.sel_cfg_ff_type = pla_ff_type_3th_filter0;
pla_ff_cfg.sel_clk_source = 0;
pla_set_ff(BOARD_PLA_COUNTER, pla_chn_0, &pla_ff_cfg);
```

实际通过上述的配置，PLA CHN[0]所形成的是 6 路 PLA CHN[1:7]的反相后相与的结果再和 PLA IN[0]的反相进行或操作。

- PLA CHN[1:7]的设置

  PLA CHN[1:7]的设置是比较雷同的，因此在例程中用了循环来进行设置。

  - 16to8 AOI 的配置

    16to8_chn0 都是上一路 PLA CHN 的输出，比如说 PLA CHN1 的 16to8_chn0 是 PLA CHN0 的输出; PLA CHN2 的 16to8_chn0 是 PLA CHN1 的输出，以此类推。

    16to8_chn1 都是自身 PLA CHN 的取反后的输出，比如说 PLA CHN1 的 16to8_chn1 是 PLA CHN1 的取反输出; PLA CHN2 的 16to8_chn1 是 PLA CHN2 的取反输出，以此类推。

    其余的 16to8_chn[x]均由软件设置为高。

```
aoi_16to8_chn_cfg.chn = m;
aoi_16to8_chn_cfg.aoi_16to8_chn = pla_aoi_16to8_chn_0;
for (uint16_t i = pla_1st_filter_out_0; i <= pla_1st_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_1st_filter_out_num].op = pla_aoi_operation_and_1;
aoi_16to8_chn_cfg.input[pla_1st_filter_out_num].signal = pla_1st_filter_out_num;
pla_set_aoi_16to8_one_channel(BOARD_PLA_COUNTER, &aoi_16to8_chn_cfg);

aoi_16to8_chn_cfg.chn = m;
aoi_16to8_chn_cfg.aoi_16to8_chn = pla_aoi_16to8_chn_1;
for (uint16_t i = pla_1st_filter_out_0; i <= pla_1st_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_1st_filter_out_num++;
aoi_16to8_chn_cfg.input[pla_1st_filter_out_num].op = pla_aoi_operation_xor_1;
aoi_16to8_chn_cfg.input[pla_1st_filter_out_num].signal = pla_1st_filter_out_num;
pla_set_aoi_16to8_one_channel(BOARD_PLA_COUNTER, &aoi_16to8_chn_cfg);

for (uint16_t j = pla_aoi_16to8_chn_2; j <= pla_aoi_16to8_chn_7; j++) {
  aoi_16to8_chn_cfg.chn = m;
```

```
    aoi_16to8_chn_cfg.aoi_16to8_chn = j;
    for (uint16_t i = pla_1st_filter_out_0; i < pla_1st_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(BOARD_PLA_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_2st_chn_cfg.val = 0;
  filter_2st_chn_cfg.sync_edge_filter_disable = true;
  filter_2st_chn_cfg.software_inject = pla_filter_sw_inject_disable;
  for (uint16_t i = pla_filter2_chn0; i <= pla_filter2_chn7; i++) {
    pla_set_filter2(BOARD_PLA_COUNTER, m, i, &filter_2st_chn_cfg);
  }
```

- 8to7 AOI 的配置

  通过对 filter2 的配置，实现 8to7_chn0 即为 filter2out[1], 而 8to7_chn1 即为 filter2out[0]

  其余的 8to7_chn[x]均由软件设置为低。

```
  aoi_8to7_chn_cfg.chn = m;
  aoi_8to7_chn_cfg.aoi_8to7_chn = pla_aoi_8to7_chn_0;
  for (uint16_t i = pla_2st_filter_out_0; i <= pla_2st_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_2st_filter_out_1].op = pla_aoi_operation_and_1;
  aoi_8to7_chn_cfg.input[pla_2st_filter_out_1].signal = pla_2st_filter_out_1;
  pla_set_aoi_8to7_one_channel(BOARD_PLA_COUNTER, &aoi_8to7_chn_cfg);

  aoi_8to7_chn_cfg.chn = m;
  aoi_8to7_chn_cfg.aoi_8to7_chn = pla_aoi_8to7_chn_1;
  for (uint16_t i = pla_2st_filter_out_0; i <= pla_2st_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_2st_filter_out_0].op = pla_aoi_operation_and_1;
  aoi_8to7_chn_cfg.input[pla_2st_filter_out_0].signal = pla_2st_filter_out_0;
```

```
pla_set_aoi_8to7_one_channel(BOARD_PLA_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]为软件设置高，

  filter3out[6]为软件设置低

```
filter_3st_chn_cfg.val = 0;
filter_3st_chn_cfg.sync_edge_filter_disable = true;
filter_3st_chn_cfg.software_inject = pla_filter_sw_inject_disable;
for (uint16_t i = pla_filter3_chn0; i < pla_filter3_chn2; i++) {
  pla_set_filter3(BOARD_PLA_COUNTER, m, i, &filter_3st_chn_cfg);
}
filter_3st_chn_cfg.val = 0;
filter_3st_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(BOARD_PLA_COUNTER, m, pla_filter3_chn2, &filter_3st_chn_cfg);
filter_3st_chn_cfg.val = 0;
filter_3st_chn_cfg.software_inject = pla_filter_sw_inject_low;
pla_set_filter3(BOARD_PLA_COUNTER, m, pla_filter3_chn3, &filter_3st_chn_cfg);
filter_3st_chn_cfg.val = 0;
filter_3st_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(BOARD_PLA_COUNTER, m, pla_filter3_chn4, &filter_3st_chn_cfg);
filter_3st_chn_cfg.val = 0;
filter_3st_chn_cfg.software_inject = pla_filter_sw_inject_height;
pla_set_filter3(BOARD_PLA_COUNTER, m, pla_filter3_chn5, &filter_3st_chn_cfg);
filter_3st_chn_cfg.val = 0;
filter_3st_chn_cfg.software_inject = pla_filter_sw_inject_low;
pla_set_filter3(BOARD_PLA_COUNTER, m, pla_filter3_chn6, &filter_3st_chn_cfg);
```

- CFF 的设置

  CFF 设置为 D 触发器，并且 clk 源为 filter3out[1]

```
pla_ff_cfg.val = 0;

pla_ff_cfg.sel_cfg_ff_type = pla_ff_type_dff;
pla_ff_cfg.sel_clk_source = 1;
pla_set_ff(BOARD_PLA_COUNTER, m, &pla_ff_cfg);
```

实际 PLA CHN[1:7]实现的是 7 个 D 触发器串联的效果，每个触发的输出接到了下一

个触发器的 CLK。

- 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);
```

  - 强制设置 PLA CHN[1:7]的 filter3out[3]为高，使得 PLA CHN[1:7]强制输出高

```
for (uint8_t m = pla_chn_1; m <= pla_chn_7; m++) {
  filter_3st_chn_cfg.val = 0;
  filter_3st_chn_cfg.software_inject = pla_filter_sw_inject_height;
  pla_set_filter3(BOARD_PLA_COUNTER, m, pla_filter3_chn3, &filter_3st_chn_cfg);
}
```

  - 强制设置 PLA CHN[1:7]的 filter3out[3]为低，使得 PLA CHN[1:7]的输出均由前一路的输出作为输入并在上升沿时改变输出的状态。

```
for (uint8_t m = pla_chn_1; m <= pla_chn_7; m++) {
  filter_3st_chn_cfg.val = 0;
  filter_3st_chn_cfg.software_inject = pla_filter_sw_inject_low;
  pla_set_filter3(BOARD_PLA_COUNTER, m, pla_filter3_chn3, &filter_3st_chn_cfg);
}
```

3.3 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

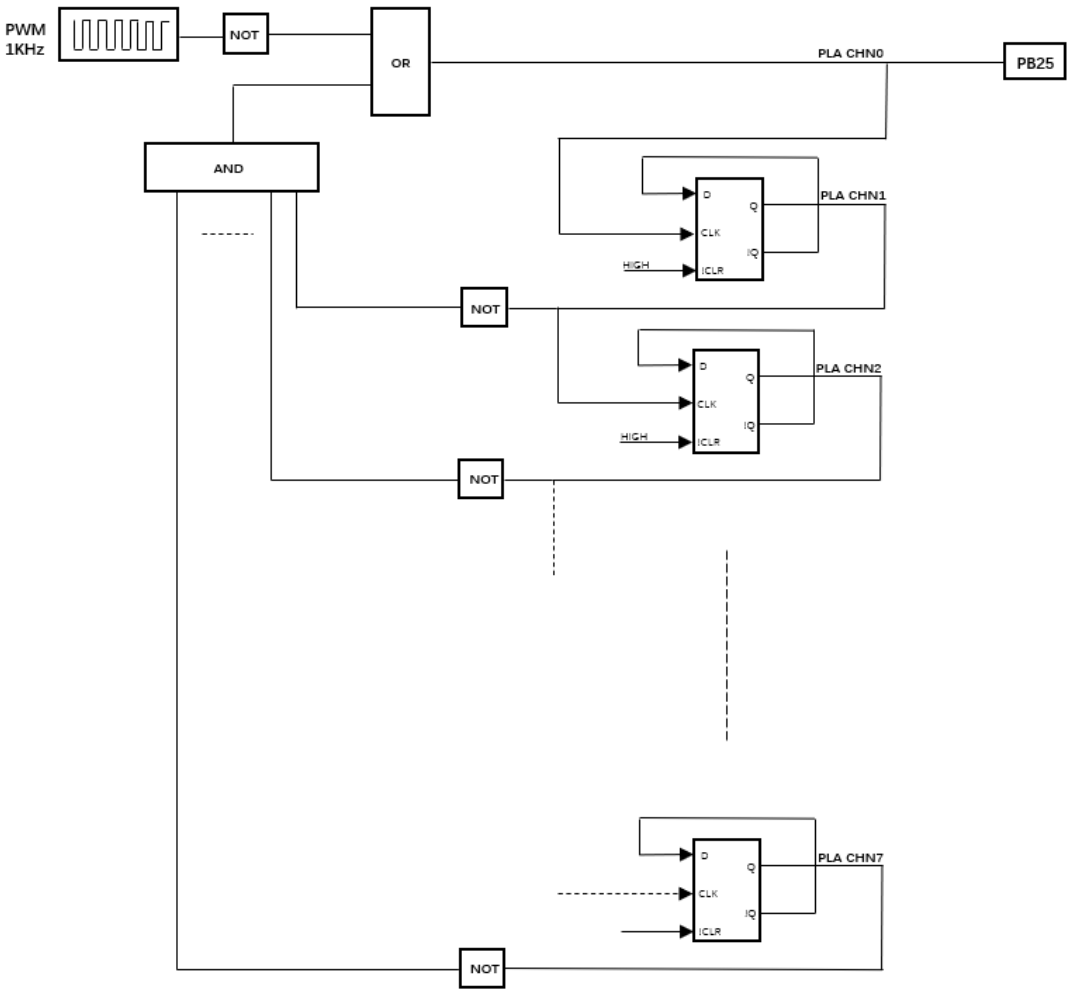

图 6

- PLA CHN[0:7]中 PLA CHN[1:7]所实现的是 7 个 D 触发器串联，每一路的 CLK 信号都是由上一路的输出实现。
- PLA CHN0 的输出由两部分相或组成
  - PLA CHN[1:7]的信号先反相再相与
  - PLA IN0 的信号反相

基于上述的设计，在初始的时候，由于软件的置高，PLA CHN[1:7]全部输出为高，等待 10us 后，再由软件的置低，PLA CHN[1:7]的输出由 CLK 触发发生变化。当经过 127 个 CLK 时，PLA_CHN[1:7]会同时出现低，此时经过逻辑组合，PLA_CHN0 的输出会一直置

高，并不会在发生变化。

在 HPM6200EVK 上，测量 PB25 上的信号，就会出现 127 个 CLK。

4 总结

PLA 本身的使用是非常灵活的，本文希望通过对 SDK 里面的相关例程的介绍，使用户能够加快用户对 PLA 的理解。通过对 PLA 的使用，可以实现最大频率为 APB 总线频率的应用，并且不占用 MCU 核的资源。