采用 dsPIC33AK 系列 DSC 实现交错式三相 LLC

AN6330

目的

作者：Justin O'Shea，Microchip Technology Inc.

本应用笔记旨在提供一种利用 dsPIC33AK 器件系列所具备的资源来实现双路三相交错式半桥 LLC 谐振转换器的解决方案。 本文档概述了电路拓扑、控制方案以及 PWM 初始化和控制固件的实现细节。 内容范围仅限于 PWM 的设置及其基本操作。

目录

1. 半桥（HB）LLC 拓扑概述

得益于高效率与低 EMI 噪声，LLC 谐振转换器拓扑在服务器和数据中心应用中仍然广受欢迎。随着功率需求日益增长，对 LLC 转换器进行交错设计变得至关重要。 多相系统可以进一步交错，以提供更高的功率容量并进一步降低纹波电流。 本应用笔记重点介绍一种配置：采用中心抽头变压器和“Y”型输出结构的双路三相交错式半桥。初级侧由半桥驱动器和 LLC 谐振回路组成。次级侧整流器由具备有源控制的功率开关实现。次级绕组采用中心抽头结构并作为直流输出。图 1-1 给出了半桥 LLC 的高阶原理图。图 1-2 将三相组合为双路交错式应用的其中一路。 总共有六相，每相由一对 PWM 输出驱动，共计 24 个开关。 主 PWM 外设包含 8 个高速 PWM 发生器（PWM Generator，PG），但该双路三相交错式系统需要 12 个独立的 PWM 对。某些 dsPIC33AK 器件系列包含 1 个额外的 PWM 外设，称为辅助 PWM（Auxiliary PWM，APWM）。PWM 与 APWM 共用相同的系统资源，并且两者的事件和信号可以互连。

图 1-1. 单相半桥 LLC 实现

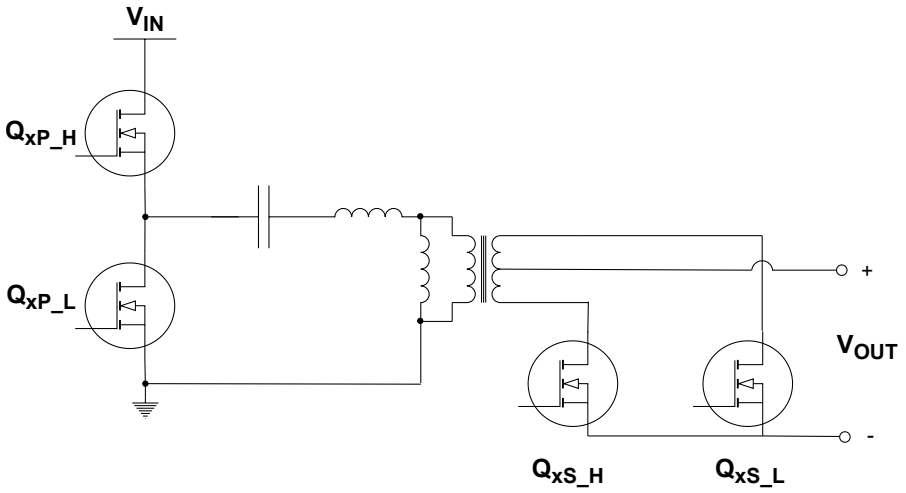
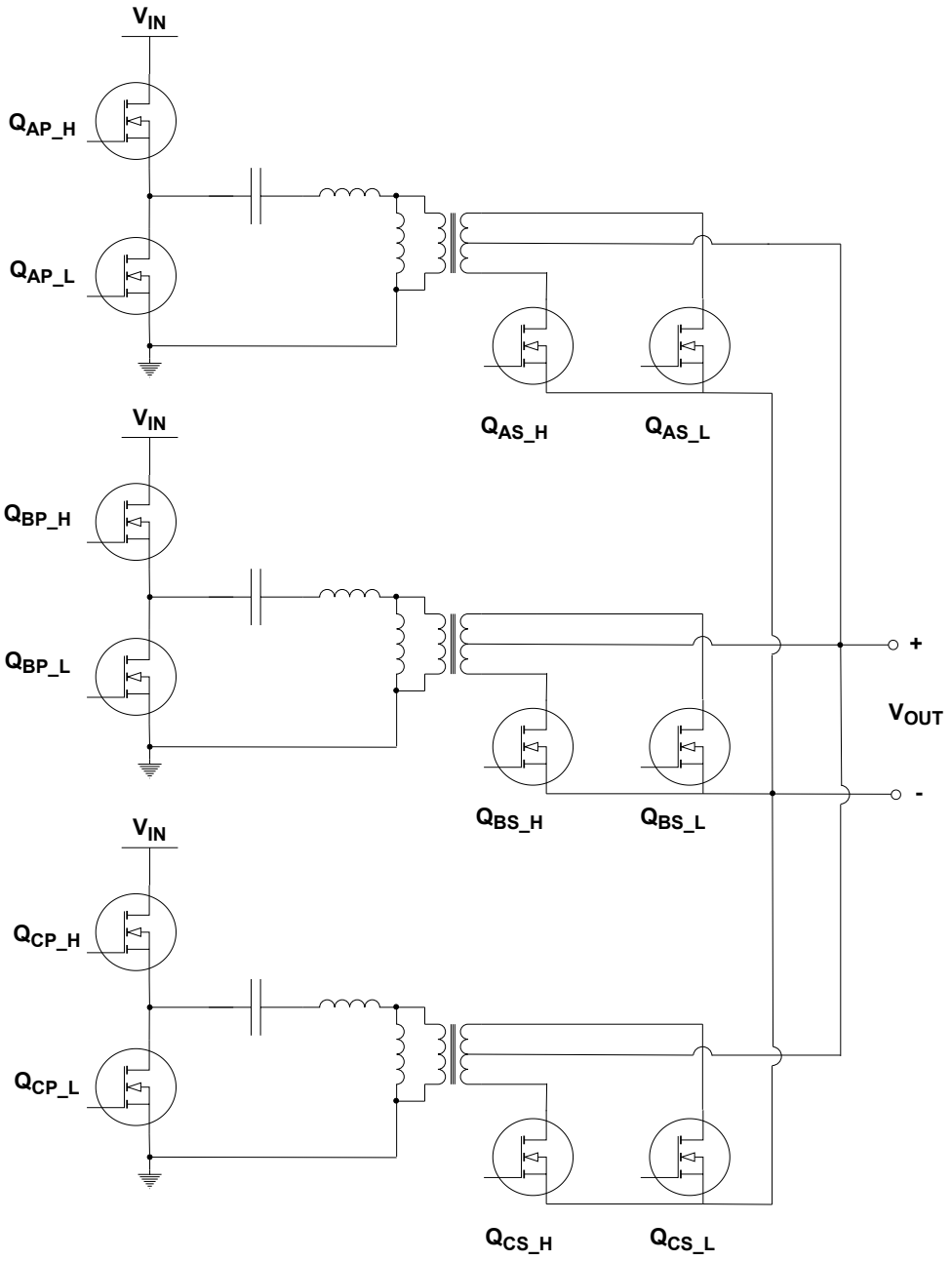


图 1-2. 三相半桥 LLC



2. PWM 到开关的映射

可以任意指定 PWM 实例与开关之间的映射关系。dsPIC33A 器件允许将 PWM 输出路由至任何可重映射的引脚（RPx），从而提高灵活性。本例选用的 PWM 实例映射旨在允许 PWM 与 APWM 共用主周期寄存器、占空比寄存器和相位寄存器（MPER、MDC 和 MPHASE）。这样可以减少工作期间需要写入的寄存器数。所有辅助 PWM 发生器（Auxiliary PWM Generator，APG）均位于次级侧。24 个 PWM 输出分别映射至其对应的开关，如表 2-1 所示。

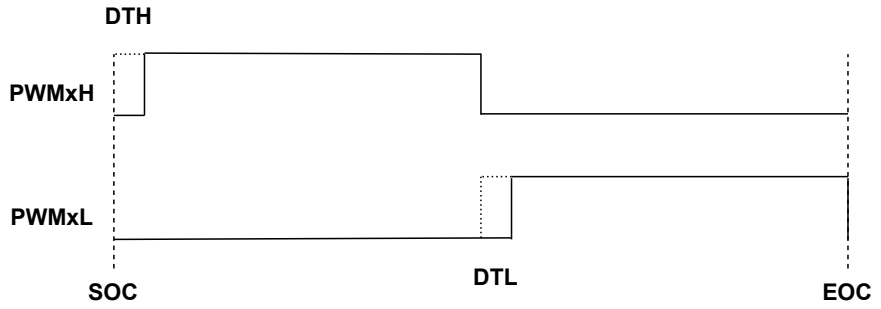
表 2-1. PWM 映射

组	相	开关位置	开关标识	PWM 输出
1	A	初级侧上桥臂	QAP_H	PWM1H
		初级侧下桥臂	QAP_L	PWM1L
		次级侧上桥臂	QAS_H	PWM2H
		次级侧下桥臂	QAS_L	PWM2L
	B	初级侧上桥臂	QBP_H	PWM3H
		初级侧下桥臂	QBP_L	PWM3L
		次级侧上桥臂	QBS_H	APWM1H
		次级侧下桥臂	QBS_L	APWM1L
	C	初级侧上桥臂	QCP_H	PWM4H
		初级侧下桥臂	QCP_L	PWM4L
		次级侧上桥臂	QCS_H	APWM2H
		次级侧下桥臂	QCS_L	APWM2L
2	A	初级侧上桥臂	QAP_H	PWM5H
		初级侧下桥臂	QAP_L	PWM5L
		次级侧上桥臂	QAS_H	PWM6H
		次级侧下桥臂	QAS_L	PWM6L
	B	初级侧上桥臂	QBP_H	PWM7H
		初级侧下桥臂	QBP_L	PWM7L
		次级侧上桥臂	QBS_H	APWM3H
		次级侧下桥臂	QBS_L	APWM3L
	C	初级侧上桥臂	QCP_H	PWM8H
		初级侧下桥臂	QCP_L	PWM8L
		次级侧上桥臂	QCS_H	APWM4H
		次级侧下桥臂	QCS_L	APWM4L

3. PWM 时序和控制概述

用于驱动电路的 PWM 控制方案基于变压器应用中典型的可变频率 50%占空比推挽式配置。 一个 PWM 输出对的单周期如图 3-1 所示。

图 3-1. 带死区的推挽 PWM 输出波形



两个输出的上升沿和下降沿均可独立控制。 在跳变之间插入死区可避免出现直通情况。扩展到三相交错式系统时，B 相和 C 相分别相对于 A 相相移 120°和 240°，如图 3-2 所示。

图 3-2. 组 1，三相 PWM 时序，初级侧

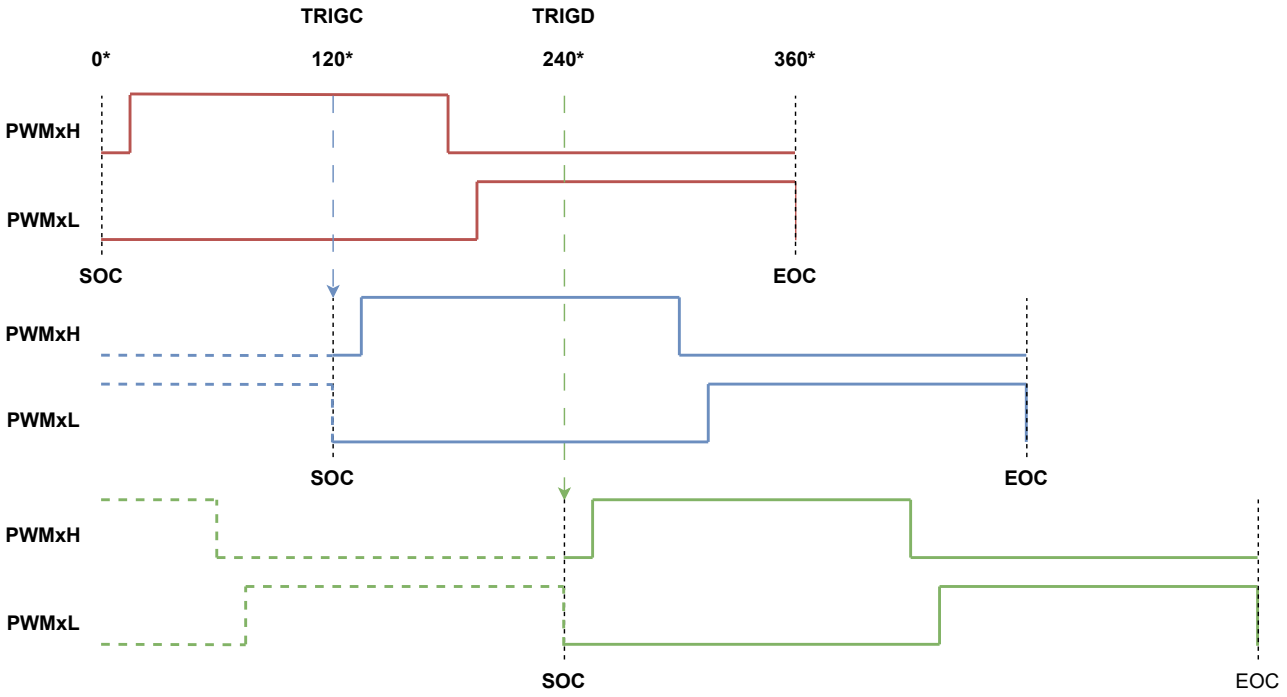


图 3-3 在上图的基础上进一步增加了用于次级侧整流（Secondary Rectification，SR）控制的 PWM 及其各自的周期开始（Start-Of-Cycle，SOC）触发信号。初级侧 PG 与次级侧 PG 之间实现了额外的相位偏移和占空比持续时间，但为简单起见未在图中显示。系统的两个组随后以 90°的相位偏移进行交错，从而构成包含 24 个开关的完整系统，如图 3-4 所示。为突出组 2 的时序依赖关系，省略了组 1 的 B 相和 C 相。

图 3-3. 组 1，三相 PWM 时序图

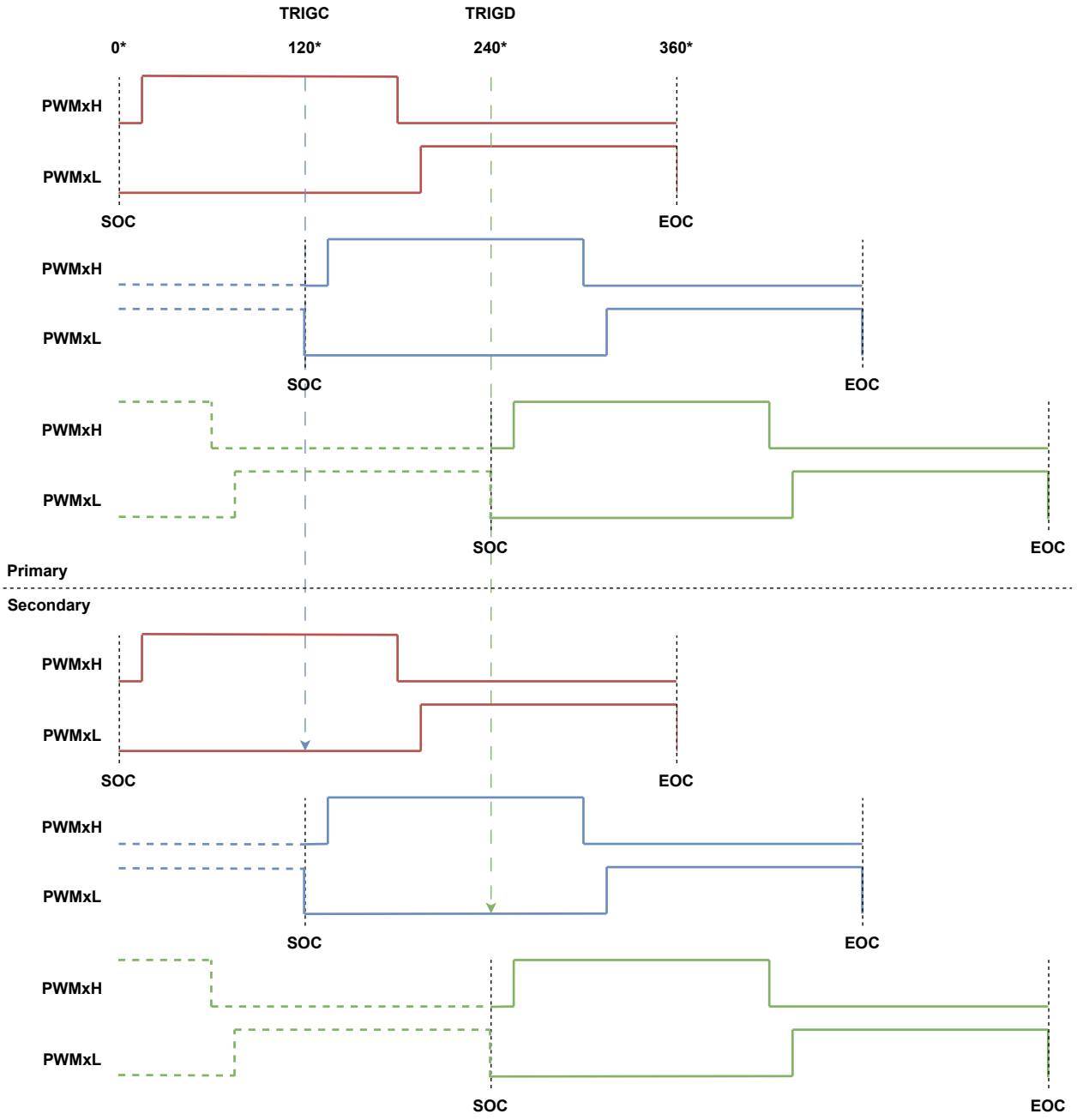
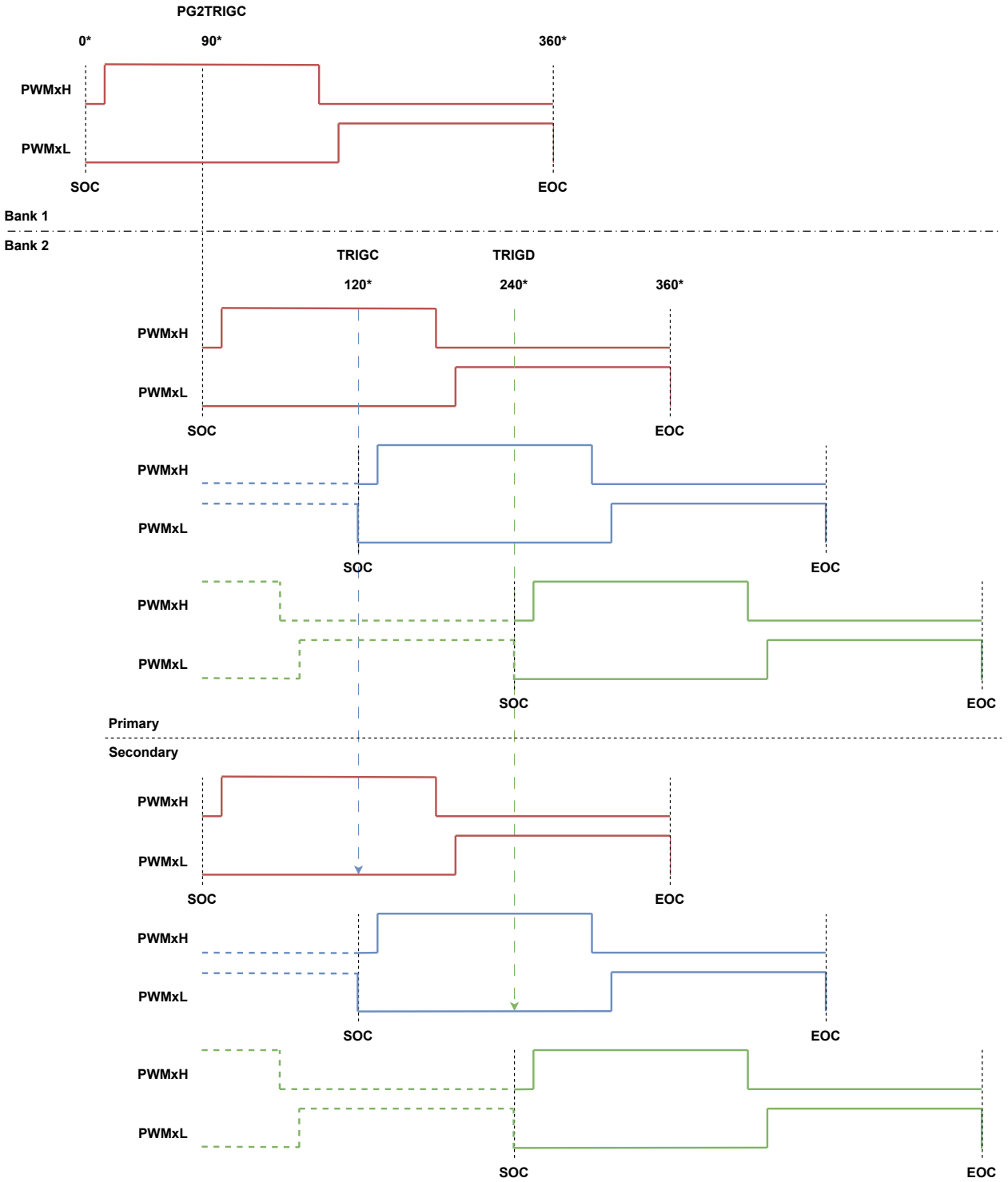


图 3-4. 交错式三相 PWM 时序图



4. 基本 PWM 设置和时序

所有 PWM 通道均配置为独立边沿 PWM 模式、双输出，并将输出模式设为独立。该配置能够对所有 PWM 边沿时序实现真正的独立控制。这是强制性配置，因为初级侧半桥 FET 和 SR FET 通常需要不同的脉冲宽度，以补偿栅极驱动器延时并优化次级侧的零电流开关。各 PWM 发生器（PGx）之间共用主数据寄存器（MPER、MDC 和 MPHASE），以简化数据更新。 对于辅助 PWM 外设，使用主寄存器 AMPER、AMDC 和 AMPHASE 来简化数据更新。

图 4-1. PWM 配置

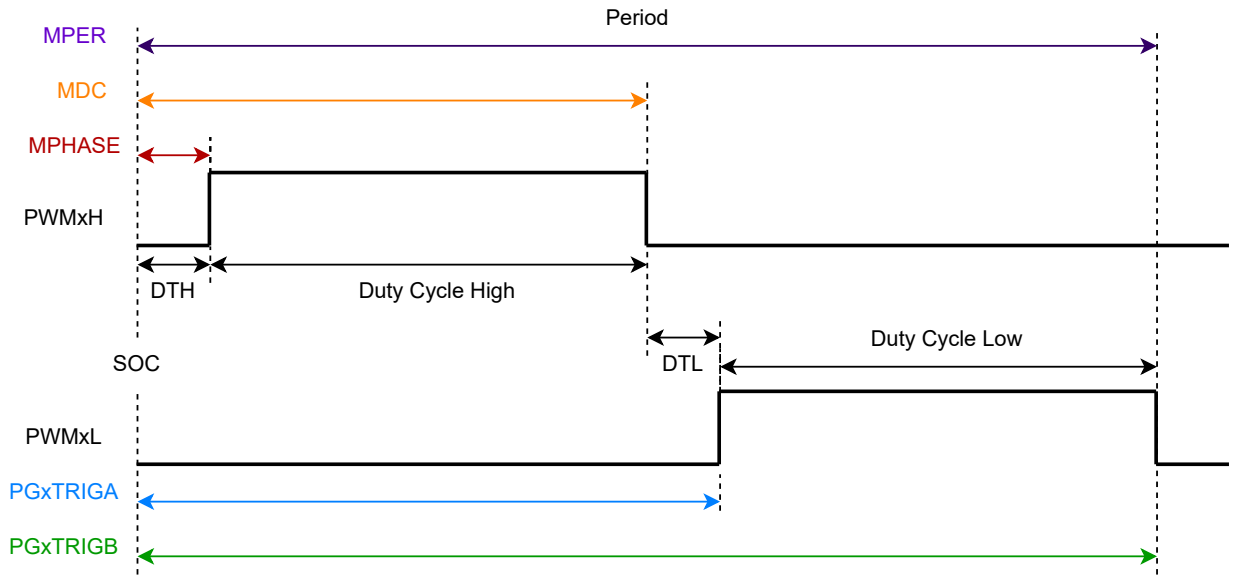


图 4-1 中所示的边沿时序由以下公式定义：

DTH = MPHASE

上桥臂占空比 = MDC - MPHASE

DTL = PGxTRIGA – MDC

下桥臂占空比 = PGxTRIGB – PGxTRIGA

代入固件写入的数据值后：

MPER = 周期

MPHASE = DTH

MDC = 上桥臂占空比 = MPER/2（50% 占空比）

PGxTRIGA = MPER/2 + DTL

PGxTRIGB = MPER

PG1 配置为系统中的“主”PG。所有边沿时序都源自 PG1，其中包括相位偏移事件触发信号以及主数据寄存器 MPER、MDC 和 MPHASE。PG1 同时也是数据更新主控器，用于将更新同步请求广播到其所在组内的其他 7 个 PG。同理，APG1 也是数据更新主控器，用于将更新同步请求广播到其所在组内的其他 3 个 APG。

4.1. PWM 时序数据

表 4-1 列出了所有 PG 的 PWM 时序数据分配。PG2 和 PG6 使用各自的相位和占空比数据寄存器，因为两者位于次级侧，其时序与位于初级侧的其他 6 个 PG 有所不同。如前所述，PGxTRIGA 和 PGxTRIGB 寄存

器用于 PWMxL 时序，其中触发信号 B 寄存器保存最新计算的周期（MPER/AMPER），触发信号 A 寄存器保存周期/2 + 下桥臂死区时间。

表 4-1. PWM 时序数据

PG 实例	周期	上桥臂死区时间	上桥臂占空比	下桥臂死区时间	下桥臂占空比	相位偏移	分配
PG1	PG1PER	MPHASE	MDC	PG1TRIGA – MDC	PG1TRIGB – PG1TRIGA	----------	组 1，初级侧
PG3	MPER	MPHASE	MDC	PG3TRIGA – MDC	PG3TRIGB – PG3TRIGA	PG1TRIGC	组 1，初级侧
PG4	MPER	MPHASE	MDC	PG4TRIGA – MDC	PG4TRIGB – PG4TRIGA	PG1TRIGD	组 1，初级侧
PG2	MPER	PG2PHASE	PG2DC	PG2TRIGA – MDC	PG2TRIGB – PG2TRIGA	----------	组 1，次级侧
APG1	AMPER	AMPHASE	AMDC	APG1TRIGA – AMDC	APG1TRIGB – APG1TRIGA	PG1TRIGC	组 1，次级侧
APG2	AMPER	AMPHASE	AMDC	APG2TRIGA – AMDC	APG2TRIGB – APG2TRIGA	PG1TRIGD	组 1，次级侧
PG5	MPER	MPHASE	MDC	PG5TRIGA – MDC	PG5TRIGB – PG5TRIGA	PG2TRIGC	组 2，初级侧
PG7	MPER	MPHASE	MDC	PG7TRIGA – MDC	PG7TRIGB – PG7TRIGA	PG5TRIGC	组 2，初级侧
PG8	MPER	MPHASE	MDC	PG8TRIGA – MDC	PG8TRIGB – PG8TRIGA	PG5TRIGD	组 2，初级侧
PG6	MPER	PG6PHASE	PG6DC	PG6TRIGA – MDC	PG6TRIGB – PG6TRIGA	PG2TRIGC	组 2，次级侧
APG3	AMPER	AMPHASE	AMDC	APG3TRIGA – AMDC	APG3TRIGB – APG3TRIGA	PG5TRIGC	组 2，次级侧
APG4	AMPER	AMPHASE	AMDC	APG4TRIGA – AMDC	APG4TRIGB – APG4TRIGA	PG5TRIGD	组 2，次级侧

由于 LLC 控制系统直接调制开关频率，因此许多寄存器都会实时更新。两个 PG 之间的相位偏移也会随着周期的变化而更新。

4.2. 初级侧与次级侧的相位偏移时序

次级侧整流器边沿时序可以相对于初级侧设置一个偏移量，以优化转换器的性能。 在附录提供的代码示例中，宏 SYNC_DIFF_RISE 和 SYNC_DIFF_FALL 可用于调整上升沿相位偏移并缩短 SR 的占空比（下降沿）。在最终应用中，可以使用变量替换这些宏，以便实时调整 SR 的导通时间。对于给定应用，上升沿延时通常是固定的。来自主 PWM 外设的 PG2/PG6 与 APWM 一起用于产生 6 个 SR 的驱动信号。

AMPHASE/PGxPHASE = DTH + SYNC_DIFF_RISE

AMDC/PGxDC = (A)MPER/2 - SYNC_DIFF_FALL

(A)PGxTRIGA = (A)MPER/2 + DTL+ SYNC_DIFF_RISE

(A)PGxTRIGB = (A)MPER - SYNC_DIFF_FALL

其中，*AMPER = MPER = ctrlOutput*

5. PWM 触发

PWM 模块基于触发信号工作。每个周期都需要一个触发信号才能产生 PWM 周期。PG1 可自触发，因此会连续运行。所有其他 PG 的 SOC 触发信号均直接来自 PG1 或经由另一个 PG 获得。 触发 SOC 的方法有两种： 周期开始选择（Start Of Cycle Select，SOCS）以及 PG 控制输入（PG Control Input，PCI）的同步实例。

用于 0°、90°、120°和 240° SOC 触发信号的相位偏移使用 PG 内的 TRIGy 定时器来实现。这些定时器事件随后供触发输出使用。 触发输出有多种类型，分别具有不同的用途，其中包括 SOC 触发信号。这些触发输出分别称为 PGTRIGOUT、DACx TRIG 和 ADCx TRIG。这些触发信号随后会映射至顶层 PWM 事件输出（PWMEVTA 至 PWMEVTD）。最后，这些触发信号会分配至其他 PG 的 PCI 输入，用于各自的 SOC 事件触发信号。 图 5-1 说明了本例中使用的 SOC 触发方案。

以下公式定义了系统使用的相位偏移。 每次周期发生变化时，都会计算并更新这些相位偏移。

PG1TRIGC = 1/3 × 周期（B 相 120°相移，组 1）

PG1TRIGD = 2/3 × 周期（C 相 240°相移，组 1）

PG2TRIGC = 1/4 × 周期（从组 1 到组 2 的 90°相移）

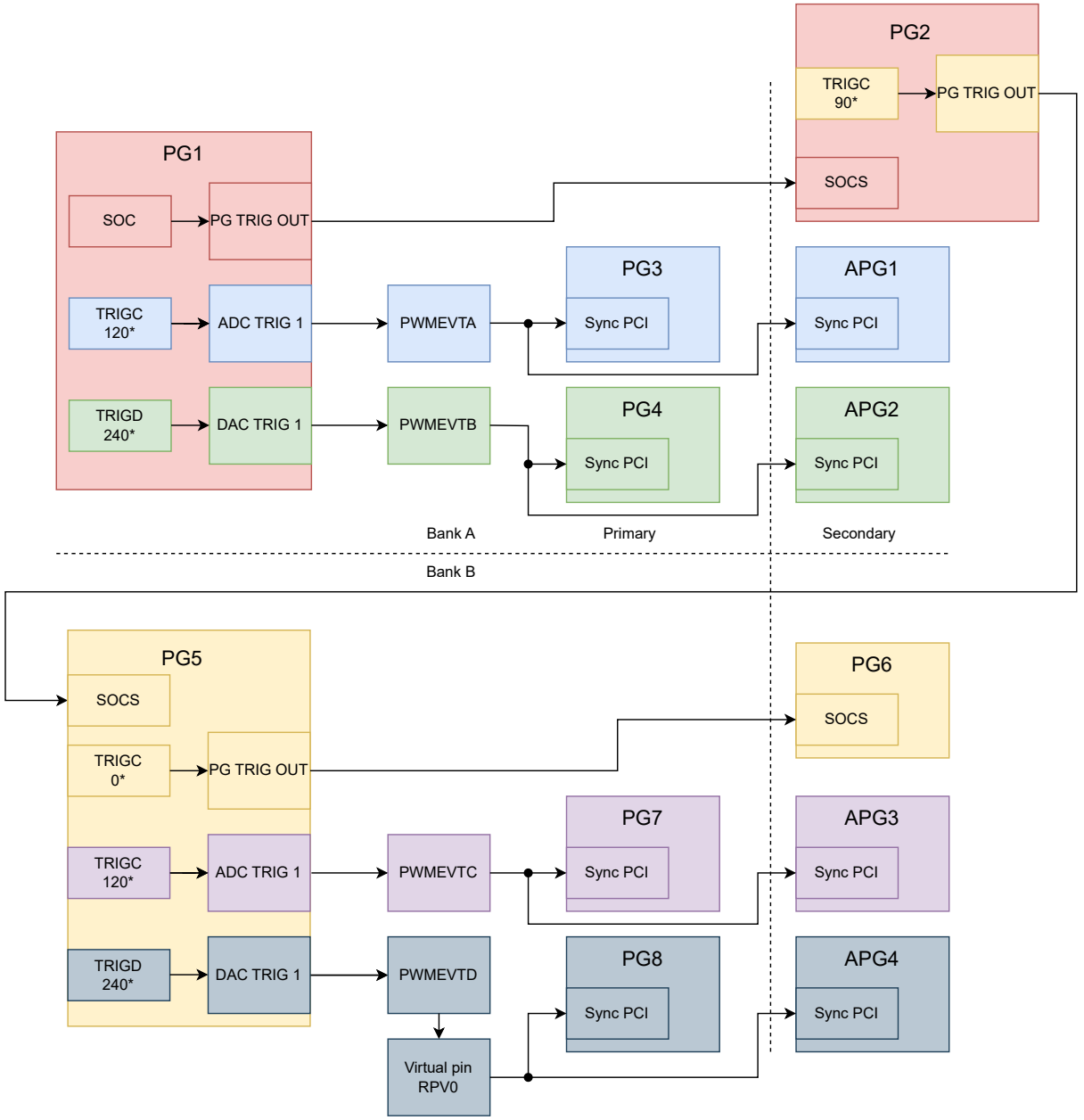
PG5TRIGC = 1/3 × 周期（B 相 120°相移，组 2）

PG5TRIGD = 2/3 × 周期（C 相 240°相移，组 2）

注：PG5 的周期开始已相对于组 1 进行 90°相移，因此对其同样采用上述 120°/240°相移的计算公式。

除 PG1 外，所有其他 PG 均工作在可重新触发模式下。因此，即便当前周期尚未结束，后续的 SOC 触发信号也能启动新周期，从而缩短周期长度，实现频率调制。除 PG1 外，所有其他 PG 使用的周期均为允许的最大值，正常工作中 PG 不会达到该值。

图 5-1. PWM 触发图



5.1. 同步 PCI 触发配置

PG 的主要实例间触发机制 PGTRGOUT 仅在 PWM 或 APWM 外设内部可用，无法跨外设使用。在本应用中，PWM 与 APWM 需要协同工作并共用触发信号。 为此，使用了同步 PCI 模块。 PCI 模块可将外部事件和异步事件同步到 PWM 域中。

PG 的 PCI 的“同步”实例是惟一可发起 SOC 触发信号的实例类型。PG 的 SOCS 配置为使用其自身的 PCI 模块，而非自触发或从另一个 PG（PGTRGOUT）触发。对于相 2 和相 3 中的 PG，其同步 PCI 配置完全相同。每个 PCI 模块有 3 个可用的 PWMEVT，但系统需要使用 4 个，如图 5-1 所示。为了映射第 4 个，PWMEVTD 信号经虚拟器件引脚路由后再送入 PCI。PCI 配置不使用接收限定符，其接收逻辑为上升沿触发且自动终止。

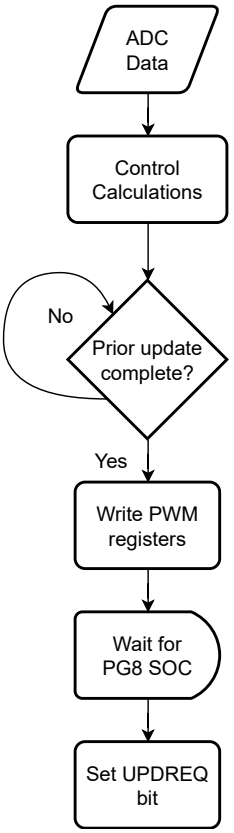
6. PWM 数据更新方案

系统控制基于可变频率操作。修改周期时，需要特别处理数据写操作的时序以及这些数据何时应用到系统中。PWM1 和 APWM1 是更新系统的“主控器”，它们将数据更新请求广播到各自外设组内的其他 PG。数据更新发生在下一个 PWM 周期开始时。

数据寄存器采用缓冲方式，以防止写操作导致数据损坏。位于 RAM SFR 空间的数据寄存器（如表 4-1 中所定义）不会直接供 PWM 使用。PWM 拥有自己的一套数据寄存器副本，专门用于 PWM 操作。因此，在 PWM 使用自身副本工作的同时，仍然可以写入 SFR。当所有数据 SFR 写操作完成后，可向 PWM 发出请求以将 SFR 中的新数据传输到 PWM 中。

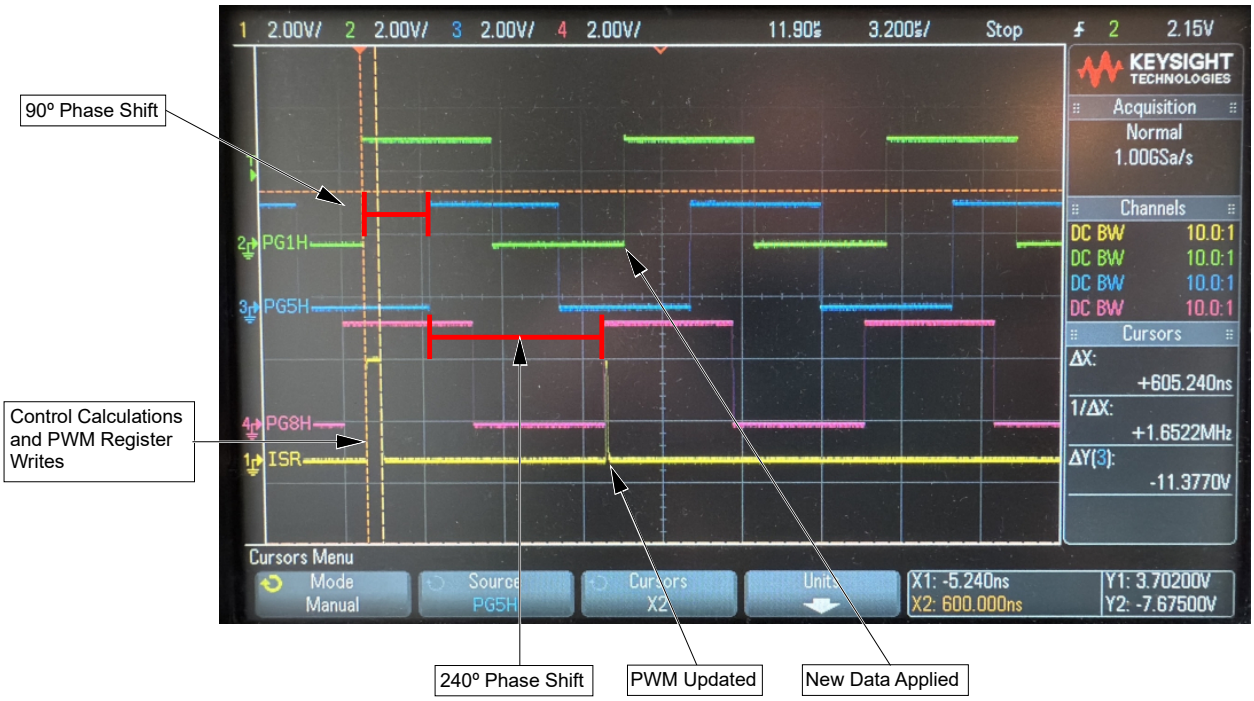
来自 ADC 的数据用于计算系统的控制参数。 这通常在 ADC 的中断服务程序中完成，并且与 PWM 异步。根据控制参数，可以计算并写入 PWM 寄存器数据。 但是，软件需确保在上一个周期完成之后再写入新值。该检查通过读取 PG 的 UPDATE 状态位来完成，旨在确保没有更新仍在等待处理。PWM 寄存器写操作的时序不依赖于 PWM 周期，因此可以随时进行。更新事件的时序与 PWM 周期相关，需要使用 PWM 的中断进行调度。时序链中的最后一个 PG 专门用于实现这一目的。本例中使用的是 PG8，其中断配置为在其 SOC 处触发。这样可以为 PWM 争取到最长的时间，使其能够在时序链中第一个 PG（即 PG1）的 SOC 到来之前完成数据传输。软件通过写入 PWM1 和 APWM1 的 UPDREQ 位来启动传输，并使之在下一个 SOC 处生效。 图 6-1 总结了更新过程。 或者，也可以使用 DMA，以 PG8 SOC 作为触发信号自动设置更新位。

图 6-1. 数据更新序列



控制系统计算出新的 PWM 开关频率后，需要根据 PWM 分辨率/节拍数来计算下一个 PWM 周期时序（相位偏移和占空比等）。在独立 PWM 工作模式下，固件必须为该双路三相交错式解决方案更新将近 40 个不同的 PWM 寄存器。当单片机以 200 MHz 运行时，这些计算和 PWM 缓冲器更新大约需要 600 ns。 图 6-2 给出了更新时序与 PWM 信号之间的关系。

图 6-2. PWM 周期时序与 PWM 信号之间的关系



7. 启动和关断同步

当关断或重新启动多相系统时，各相之间需要保持同步。PWM 输出的门控通过使用 PWM 的改写功能来实现。在改写模式下，可配置 PWM 输出的状态。在本例中，PWMxH 和 PWMxL 均驱动为逻辑“0”。鉴于系统共有 12 个 PG，并且每个 PG 都有自己的控制寄存器，软件无法同时写入所有改写使能位以实现对全部 24 个输出的并发控制。为了实现同步操作，系统使用了改写同步功能。与 PWM 数据寄存器一样，改写使能位 OVRENH 和 OVRENL 也采用缓冲方式。改写状态通过发出请求来生效，并且该请求可以一次性施加到所有从 PG。因此，可在单个 PWM 周期内（在该周期结束时）同时停止或启动所有 PWM 输出。PG1 和 APG1 作为主 PG，负责向其余 PG（即改写从 PG）广播改写更新信号。

8. 过流保护

尽管代码示例中未涉及过流保护，但 PWM 模块仍然支持该功能。此外，PWM PCI 模块还具备限流关断功能，通过输出引脚改写（CLDAT[1:0]）来实现。与用户改写不同，PWM 信号的关断会立即发生。限流 PCI 既可以配置为逐周期过流截断，也可以配置为锁存式关断（需手动终止/复位）。

将电流反馈引入单片机的方式有多种（输出电流以及用于初级侧谐振电流的 CT），但无论采用哪种方式都必须利用 DAC+CMP 模块与限流 PCI 进行互连。所有 HB LLC 可以选择共用同一个比较器输出，或者每组可以利用各自经 CT 进行逻辑或运算后的反馈电流，又或者每个 LLC 可以利用自己的 CT+比较器来实现逐周期控制。具体实现取决于应用需求，但得益于 PWM 的灵活性以及最多 8 个 DAC+CMP 模块，可以轻松地在该双路多相转换器中支持过流保护。

9. 结论

要正确驱动多相 LLC 转换器，PWM 外设需要具备相当丰富的特性。该外设不仅需要支持可变频率、相位偏移和自适应边沿控制，还需能够同步更新 12 个不同的 PWM 发生器。配置不当可能导致脉冲丢失或产生双脉冲、时序更新出错以及 PWM 信号不同步等问题，进而造成控制不当甚至损坏电源。本文档提供了实现单路和双路三相交错式 LLC 控制方案所需的详细信息。文中介绍了所有与 PWM 相关的时序和触发信息，以及如何更新和停止/重新启动系统。此外还附带一个简化的代码示例，以进一步说明具体实现细节。

10. 附录 A：代码示例，PWM 初始化

例 10-1. PWM 初始化

```
#define MAX_PERIOD  (255984 & 0xFFFF0)                  // 50 kHz
#define MIN_PERIOD  (36555 & 0xFFFF0)                   // 350 kHz
#define DEADTIME    0                                   // 用户定义
#define SYNC_DIFF_RISE   0                              // SR 延时上升沿——用户定义
#define SYNC_DIFF_FALL  0                               // SR 延时下降沿——用户定义
#define INIT_DUTY       (MIN_PERIOD/2)                  // 50%占空比
#define INIT_120_DEG    ((MIN_PERIOD/3) & 0xFFFF0)      // 初始 120° SOC 触发信号
#define INIT_240_DEG    ((MIN_PERIOD*2/3) & 0xFFFF0)    // 初始 240° SOC 触发信号
#define INIT_90_DEG    ((MIN_PERIOD/4) & 0xFFFF0)       // 初始 90° SOC 触发信号

#define PERIOD_CHANGE    10             // 更改周期扫描的步长

uint32_t llcPeriod;                     // 系统周期
uint32_t ctrlOutput = MIN_PERIOD;       // 用于周期更新的仿真 ADC 数据
uint32_t priSyncDiffRising;             // 次级侧同步延时，上升沿
uint32_t priSyncDiffFalling;            // 次级侧同步延时，下降沿
uint32_t priPWMHighPhase;               // 初级侧 PWMxH 上升沿
uint32_t priPWMHighDuty;                // 初级侧 PWMxH 下降沿
uint32_t priPWMLowTrigA;                // 初级侧 PWMxL 上升沿
uint32_t priPWMLowTrigB;                // 初级侧 PWMxL 下降沿
uint32_t syncPWMHighPhase;              // 次级侧 PWMxH 上升沿
uint32_t syncPWMHighDuty;               // 次级侧 PWMxH 下降沿
uint32_t syncPWMLowTrigA;               // 次级侧 PWMxL 上升沿
uint32_t syncPWMLowTrigB;               // 次级侧 PWMxL 下降沿
uint32_t phase120OffSetDaisyChain;      // 120° SOC 触发信号偏移量
uint32_t phase240OffSetDaisyChain;      // 240° SOC 触发信号偏移量
uint32_t phase90OffSetDaisyChain;       // 组 2 90° SOC 触发信号偏移量
uint16_t updateFlag = 0;                // 更新未完成时阻止写操作的标志

void PWM_Initialize (void)
{

    PCLKCONbits.MCLKSEL = 1;
    APCLKCONbits.MCLKSEL = 1;

    /* PWM 事件 ************************************************************/

    // 组 1 SOC 触发事件
    // PWM 事件 A 用于 PG4/APG2 的 SOC 触发信号（PG1 的 120°）
    PWMEVTAbits.EVTAPGS = 0b000;          // PG1
    PWMEVTAbits.EVTASEL = 0b01001;        // ADC 触发信号 1
    // PWM 事件 B 用于 PG3/APG1 的 SOC 触发信号（PG1 的 240°）
    PWMEVTBbits.EVTBPGS = 0b000;          // PG1
    PWMEVTBbits.EVTBSEL = 0b01011;        // DAC 触发信号

    // 组 2 SOC 触发事件
    //  PWM 事件 C 用于 PG7/APG3 的 SOC 触发信号（PG5 的 120°）
    PWMEVTCbits.EVTCPGS = 0b100;          // PG5
    PWMEVTCbits.EVTCSEL = 0b01001;        // ADC 触发信号 1

    // PWM 事件 D 用于 PG8/APG4 的 SOC 触发信号（PG5 的 240°）
    PWMEVTDbits.EVTDPGS = 0b100;          // PG5
    PWMEVTDbits.EVTDSEL = 0b01011;        // DAC 触发信号
    PWMEVTDbits.EVTDOEN = 1;              // 使能输出以路由至虚拟引脚

   // 对于该伪代码，所有 PG 和 APG 需按如下方式配置：
   // PGxCONbits.MODSEL = 0b010;         // 独立边沿 PWM 模式，双输出
   // PGxCONbits.CLKSEL = 0b01;          // PG 使用 MCLK
   // PGxIOCON1bits.PENL = 1;            // 使能 L 输出
   // PGxIOCON1bits.PENH = 1;            // 使能 H 输出
   // PGxIOCON1bits.PMOD = 0b01;         // 独立输出模式
   // PGxIOCON1bits.PPSEN = 1;           // 通过 PPS 重映射 PWM 输出（如果需要）
   // 以下为每个 PG 的具体设置

    /* PG1 ********************************************************************/
    /* 组 1，A 相，初级侧 ***************************************************/
    PG1CONbits.SOCS = 0b0000;             // 自触发
    PG1CONbits.UPDMOD = 0b000;            // 自身 SOC
    PG1CONbits.MSTEN = 1;                 // PG1 主控更新广播
    PG1CONbits.MPHSEL = 1;                // 使用主相位
    PG1CONbits.MDCSEL = 1;                // 使用主占空比
```

```
    PG1EVT1bits.ADTR1EN3 = 1;           // ADC 触发 1 事件为 PG1TRIGC（120°）
    PG1EVT1bits.DACTREN1 = 1;           // DAC 触发事件为 PG1TRIGD（240°）
    PG1EVT1bits.PGTRGSEL = 0b000;       // PGTRIGOUT 为 EOC

    /* PG2 ********************************************************************/
    /* 组 1，A 相，次级侧 *****************************************/
    PG2CONbits.SOCS = 0b0001;           // 使用 PG1 的 PGTRGOUT
    PG2CONbits.TRGMOD = 0b01;           // 可重新触发
    PG2CONbits.UPDMOD = 0b010;          // 从 SOC
    PG2CONbits.MPERSEL = 1;             // 使用 MPER
    PG2IOCON2bits.OSYNC = 0b10;         // 改写由 UPDMOD 指定
    PG2EVT1bits.PGTRGSEL = 0b011;       // PGTRIGOUT 事件为 PG2TRIGC，相对于 PG5 为 90°

    /* PG3 ********************************************************************/
    /* 组 1，B 相，初级侧 *****************************************/
    PG3CONbits.SOCS = 0b1111;           // 同步 PCI 作为周期开始
    PG3CONbits.TRGMOD = 0b01;           // 可重新触发
    PG3CONbits.UPDMOD = 0b010;          // 从 SOC
    PG3CONbits.MPHSEL = 1;              // 使用 MPHASE
    PG3CONbits.MPERSEL = 1;             // 使用 MPER
    PG3CONbits.MDCSEL = 1;              // 使用 MDC
    PG3IOCON2bits.OSYNC = 0b10;         // 改写由 UPDMOD 指定

    PG3SPCI1bits.TERM = 0b001;          // 自动终止
    PG3SPCI1bits.TSYNCDIS = 1;          // 锁存 PCI 的终止立即发生
    PG3SPCI1bits.ACP = 0b001;           // 上升沿
    PG3SPCI2 = 1<<23;                   // PCI 源为 PWMEVTA（120°）

    /* APG1 *******************************************************************/
    /* 组 1，B 相，次级侧 *****************************************/
    APG1CONbits.SOCS = 0b1111;          // 同步 PCI 作为周期开始
    APG1CONbits.TRGMOD = 0b01;          // 可重新触发
    APG1CONbits.UPDMOD = 0b000;         // 自身 SOC
    APG1CONbits.MSTEN = 1;              // APG1 主控更新广播
    APG1CONbits.MPHSEL = 1;             // 使用 MPHASE
    APG1CONbits.MPERSEL = 1;            // 使用 MPER
    APG1CONbits.MDCSEL = 1;             // 使用 MDC
    APG1IOCON2bits.OSYNC = 0b10;        // 改写由 UPDMOD 指定

    APG1SPCI1bits.TERM = 0b001;         // 自动终止
    APG1SPCI1bits.TSYNCDIS = 1;         // 锁存 PCI 的终止立即发生
    APG1SPCI1bits.ACP = 0b001;          // 上升沿
    APG1SPCI2 = 1<<23;                  // PCI 源为 PWMEVTA（120°）

    /* PG4 ********************************************************************/
    /* 组 1，C 相，初级侧 *****************************************/
    PG4CONbits.SOCS = 0b1111;           // 同步 PCI 作为周期开始
    PG4CONbits.TRGMOD = 0b01;           // 可重新触发
    PG4CONbits.UPDMOD = 0b010;          // 从 SOC
    PG4CONbits.MPHSEL = 1;              // 使用 MPHASE
    PG4CONbits.MPERSEL = 1;               // 使用 MPER
    PG4CONbits.MDCSEL = 1;              // 使用 MDC
    PG4IOCON2bits.OSYNC = 0b10;         // 改写由 UPDMOD 指定

    PG4SPCI1bits.TERM = 0b001;          // 自动终止
    PG4SPCI1bits.TSYNCDIS = 1;          // 锁存 PCI 的终止立即发生
    PG4SPCI1bits.ACP = 0b001;           // 上升沿
    PG4SPCI2 = 1<<24;                   // PCI 源为 PWMEVTB（240°）

    /* APG2 *******************************************************************/
    /* 组 1，C 相，次级侧 *****************************************/
    APG2CONbits.SOCS = 0b1111;          // 同步 PCI 作为周期开始
    APG2CONbits.TRGMOD = 0b01;          // 可重新触发
    APG2CONbits.UPDMOD = 0b010;         // 从 SOC
    APG2CONbits.MPHSEL = 1;             // 使用 MPHASE
    APG2CONbits.MPERSEL = 1;            // 使用 MPER
    APG2CONbits.MDCSEL = 1;             // 使用 MDC
    APG2IOCON2bits.OSYNC = 0b10;        // 改写由 UPDMOD 指定

    APG2SPCI1bits.TERM = 0b001;         // 自动终止
    APG2SPCI1bits.TSYNCDIS = 1;         // 锁存 PCI 的终止立即发生
    APG2SPCI1bits.ACP = 0b001;          // 上升沿
    APG2SPCI2 = 1<<24;                  // PCI 源为 PWMEVTB（240°）

    /******************第二个三相交错式 LLC**********************/

    /* PG5 ********************************************************************/
    /* 组 2，A 相，初级侧 *****************************************/
    PG5CONbits.SOCS = 0b0010;           // SOC 为 PG2 的 PGTRIGOUT（相对于 PG2 和 PG1 为 90°）
    PG5CONbits.TRGMOD = 0b01;           // 可重新触发
```

```
PG5CONbits.UPDMOD = 0b010;          // 从 SOC
PG5CONbits.MPHSEL = 1;              // 使用 MPHASE
PG5CONbits.MPERSEL = 1;             // 使用 MPER
PG5CONbits.MDCSEL = 1;              // 使用 MDC
PG5IOCON2bits.OSYNC = 0b10;         // 改写由 UPDMOD 指定

PG5EVT1bits.ADTR1EN3 = 1;           // ADC 触发 1 事件为 PG5TRIGC（120°）
PG5EVT1bits.DACTREN1 = 1;           // DAC 触发事件为 PG5TRIGD（240°）
PG5EVT1bits.PGTRGSEL = 0b101;       // PGTRIGOUT 事件为 PG5TRIGE，PG6 的 SOC

/* PG6 ************************************************************************/
/* 组 2，A 相，次级侧 ***********************************************************/
PG6CONbits.SOCS = 0b0101;           // SOC 为 PG5 的 TRIGE，PG5  SOC
PG6CONbits.TRGMOD = 0b01;           // 可重新触发
PG6CONbits.UPDMOD = 0b010;          // 从 SOC
PG6CONbits.MPERSEL = 1;             // 使用 MPER
PG6IOCON2bits.OSYNC = 0b10;         // 改写由 UPDMOD 指定

/* PG7 ************************************************************************/
/* 组 2，B 相，初级侧 ***********************************************************/
PG7CONbits.SOCS = 0b1111;          // 同步 PCI 作为周期开始
PG7CONbits.TRGMOD = 0b01;          // 可重新触发
PG7CONbits.UPDMOD = 0b010;         // 从 SOC
PG7CONbits.MPHSEL = 1;             // 使用 MPHASE
PG7CONbits.MPERSEL = 1;            // 使用 MPER
PG7CONbits.MDCSEL = 1;             // 使用 MDC
PG7IOCON2bits.OSYNC = 0b10;        // 改写由 UPDMOD 指定

PG7SPCI1bits.TERM = 0b001;          // 自动终止
PG7SPCI1bits.TSYNCDIS = 1;          // 锁存 PCI 的终止立即发生
PG7SPCI1bits.ACP = 0b001;           // 上升沿
PG7SPCI2 = 1<<25;                   // PCI 源为 PWMEVTC（120°）

/* APG3 ***********************************************************************/
/* 组 2，B 相，次级侧 ***********************************************************/
APG3CONbits.SOCS = 0b1111;         // 同步 PCI 作为周期开始
APG3CONbits.TRGMOD = 0b01;         // 可重新触发
APG3CONbits.UPDMOD = 0b010;        // 从 SOC
APG3CONbits.MPHSEL = 1;            // 使用 MPHASE
APG3CONbits.MPERSEL = 1;           // 使用 MPER
APG3CONbits.MDCSEL = 1;            // 使用 MDC
APG3IOCON2bits.OSYNC = 0b10;       // 改写由 UPDMOD 指定

APG3SPCI1bits.TERM = 0b001;       // 自动终止
APG3SPCI1bits.TSYNCDIS = 1;       // 锁存 PCI 的终止立即发生
APG3SPCI1bits.ACP = 0b001;        // 上升沿
APG3SPCI2 = 1<<25;                // PCI 源为 PWMEVTC（120°）

/* PG8 ************************************************************************/
/* 组 2，C 相，初级侧 ***********************************************************/
PG8CONbits.SOCS = 0b1111;         // 同步 PCI 作为周期开始
PG8CONbits.TRGMOD = 0b01;         // 可重新触发
PG8CONbits.UPDMOD = 0b010;        // 从 SOC
PG8CONbits.MPHSEL = 1;            // 使用 MPHASE
PG8CONbits.MPERSEL = 1;           // 使用 MPER
PG8CONbits.MDCSEL = 1;            // 使用 MDC
PG8IOCON2bits.OSYNC = 0b10;       // 改写由 UPDMOD 指定

PG8EVT1bits.IEVTSEL = 0b11;       // 禁止时基中断
PG8EVT1bits.SIEN = 1;             // PG8 中断为同步 PCI

PG8SPCI1bits.TERM = 0b001;        // 自动终止
PG8SPCI1bits.TSYNCDIS = 1;        // 锁存 PCI 的终止立即发生
PG8SPCI1bits.ACP = 0b001;         // 上升沿
PG8SPCI2 = 1<<22;                 // PCI 源为 PCI 输入 22（240°）

/* APG4 ***********************************************************************/
/* 组 2，C 相，次级侧 ***********************************************************/
APG4CONbits.SOCS = 0b1111;         // 同步 PCI 作为周期开始
APG4CONbits.TRGMOD = 0b01;         // 可重新触发
APG4CONbits.UPDMOD = 0b010;        // 从 SOC
APG4CONbits.MPHSEL = 1;            // 使用 MPHASE
APG4CONbits.MPERSEL = 1;           // 使用 MPER
APG4CONbits.MDCSEL = 1;            // 使用 MDC
APG4IOCON2bits.OSYNC = 0b10;       // 改写由 UPDMOD 指定

APG4SPCI1bits.TERM = 0b001;        // 自动终止
APG4SPCI1bits.TSYNCDIS = 1;        // 锁存 PCI 的终止立即发生
APG4SPCI1bits.ACP = 0b001;         // 上升沿
APG4SPCI2 = 1<<22;                 // PCI 源为 PCI 输入 22（240°）
```

```
    /* 数据初始化 ******************************************************/
    MPER = MAX_PERIOD;              // 设置为最大值并通过 PG 触发信号控制
    MPHASE = DEADTIME;
    MDC = INIT_DUTY;

    AMPER = MAX_PERIOD;
    AMPHASE = DEADTIME + SYNC_DIFF_RISE;
    AMDC = INIT_DUTY - SYNC_DIFF_FALL;

    PG1PER = MIN_PERIOD;
    PG1PHASE = DEADTIME;
    PG1TRIGA = INIT_DUTY + DEADTIME;
    PG1TRIGB = MIN_PERIOD;
    PG1TRIGC = INIT_120_DEG;
    PG1TRIGD = INIT_240_DEG;

    PG2PHASE = DEADTIME + SYNC_DIFF_RISE;
    PG2DC = INIT_DUTY - SYNC_DIFF_FALL;
    PG2TRIGA = INIT_DUTY + DEADTIME + SYNC_DIFF_RISE;
    PG2TRIGB = MIN_PERIOD - SYNC_DIFF_FALL;
    PG2TRIGC = INIT_90_DEG;

    PG3TRIGA = PG4TRIGA = PG5TRIGA = (INIT_DUTY + DEADTIME);
    PG3TRIGB = PG4TRIGB = PG5TRIGB = MIN_PERIOD;
    PG5TRIGC = INIT_120_DEG;
    PG5TRIGD = INIT_240_DEG;

    PG6PHASE = DEADTIME + SYNC_DIFF_RISE;
    PG6DC = INIT_DUTY - SYNC_DIFF_FALL;
    PG6TRIGA = INIT_DUTY + DEADTIME + SYNC_DIFF_RISE;
    PG6TRIGB = MIN_PERIOD - SYNC_DIFF_FALL;

    PG7TRIGA = PG8TRIGA = (INIT_DUTY + DEADTIME);
    PG7TRIGB = PG8TRIGB = MIN_PERIOD;

    APG1TRIGA = APG2TRIGA = APG3TRIGA = APG4TRIGA = (INIT_DUTY+DEADTIME+SYNC_DIFF_RISE);
    APG1TRIGB = APG2TRIGB = APG3TRIGB = APG4TRIGB = (MIN_PERIOD - SYNC_DIFF_FALL);

    // 清零所有 PG 的改写数据，默认已清零
    // 即 PG1IOCON2bits.OVRDAT = 0b00;

    // 使能所有 PWM 模块 APGx 和 PGx，最后使能 PG1
    // 即 .., ..., ..; PG1CONbits.ON = 1U;
}
```

11. 附录 B：代码示例，数据更新和改写

例 11-1. 数据更新和改写

```
Void __inline__ PWM_AllRegistersWrite (void)
{
// 主写操作——注意 MPER 已在初始化程序中设为 MAX_PERIOD
    MPHASE = priPWMHighPhase;
    MDC = priPWMHighDuty;

    AMPHASE = syncPWMHighPhase;
    AMDC = syncPWMHighDuty;

    PG1PER = llcPeriod;
    PG1TRIGA = priPWMLowTrigA;
    PG1TRIGB = priPWMLowTrigB;
    PG1TRIGC = phase120OffSetDaisyChain;
    PG1TRIGD = phase240OffSetDaisyChain;

    PG2PHASE = syncPWMHighPhase;
    PG2DC = syncPWMHighDuty;
    PG2TRIGA = syncPWMLowTrigA;
    PG2TRIGB = syncPWMLowTrigB;
    PG2TRIGC = phase90OffSetDaisyChain;

    PG3TRIGA = PG4TRIGA = PG5TRIGA = priPWMLowTrigA;
    PG3TRIGB = PG4TRIGB = PG5TRIGB = priPWMLowTrigB;

    PG5TRIGC = phase120OffSetDaisyChain;
    PG5TRIGD = phase240OffSetDaisyChain;

    PG6PHASE = syncPWMHighPhase;
    PG6DC = syncPWMHighDuty;
    PG6TRIGA = syncPWMLowTrigA;
    PG6TRIGB = syncPWMLowTrigB;

    PG7TRIGA = PG8TRIGA = priPWMLowTrigA;
    PG7TRIGB = PG8TRIGB = priPWMLowTrigB;

    APG1TRIGA = APG2TRIGA = APG3TRIGA = APG4TRIGA = syncPWMLowTrigA;
    APG1TRIGB = APG2TRIGB = APG3TRIGB = APG4TRIGB = syncPWMLowTrigB;
}

void __attribute__ ((interrupt, no_auto_psv, context)) _T1Interrupt(void)
{
    // 仿真 100 kHz 的 ADC ISR
    _T1IF = 0;

    // 周期上下扫描
    if (up_down == 0) {
        ctrlOutput -= PERIOD_CHANGE;}
    else {
        ctrlOutput += PERIOD_CHANGE;}

    if (ctrlOutput >= MAX_PERIOD + 64){
        up_down = 0;
        ctrlOutput = MAX_PERIOD;}
    else if (ctrlOutput <= MIN_PERIOD){
        up_down = 1;
        ctrlOutput = MIN_PERIOD;}

    if ((updateFlag == 0) && (PG8STATbits.UPDATE == 0))
    {
        priSyncDiffRising = SYNC_DIFF_RISE;  // 可在运行时配置
        priSyncDiffFalling = SYNC_DIFF_FALL;
        llcPeriod = ctrlOutput & 0xFFFF0;
        phase120OffSetDaisyChain = (__builtin_muluu_32(llcPeriod, 1365)>>12);
        phase120OffSetDaisyChain & = 0xFFFF0;
        phase90OffSetDaisyChain = (llcPeriod>>2) & 0xFFFF0;
        phase240OffSetDaisyChain = phase120OffSetDaisyChain<<1;
        priPWMHighPhase = DEADTIME;        // 可在运行时配置
        priPWMHighDuty = (llcPeriod >> 1);
        priPWMLowTrigA = priPWMHighDuty + deadTime;
        priPWMLowTrigB = llcPeriod;
        syncPWMHighPhase = DEADTIME + priSyncDiffRising;
        syncPWMHighDuty = (llcPeriod >> 1) - priSyncDiffFalling;
        syncPWMLowTrigA = priPWMLowTrigA + priSyncDiffRising;
        syncPWMLowTrigB = priPWMLowTrigB - priSyncDiffFalling;

        PWM_AllRegistersWrite();
        updateFlag = 1; // 所有数据已写入，准备 PWM 更新——PWM8 ISR

        _PWM8IF = 0;
```

```
        _PWM8IE = 1;    // 使能 PWM8 ISR 用于调度
    }
}

void __attribute__ ((interrupt, no_auto_psv, context)) _PWM8Interrupt(void) {
    if (updateFlag  == 1) {
        PG1STATbits.UPDREQ = 1;
        APG1STATbits.UPDREQ = 1;
        updateFlag = 0;
    }

    _PWM8IE = 0;
    _PWM8IF = 0;    // 禁止 PWM8 ISR，直到下一个写周期完成
}


void PrimaryOverride(void) {
    PG1IOCON2 |= 0x00300000;
    PG3IOCON2 |= 0x00300000;
    PG4IOCON2 |= 0x00300000;
    PG5IOCON2 |= 0x00300000;
    PG7IOCON2 |= 0x00300000;
    PG8IOCON2 |= 0x00300000;
}

void SecondaryOverride(void) {
    PG2IOCON2 |= 0x00300000;
    PG6IOCON2 |= 0x00300000;
    APG1IOCON2 |= 0x00300000;
    APG2IOCON2 |= 0x00300000;
    APG3IOCON2 |= 0x00300000;
    APG4IOCON2 |= 0x00300000;
}

void PrimaryOverrideDisable(void) {
    PG1IOCON2 &= 0xFFCFFFFF;
    PG3IOCON2 &= 0xFFCFFFFF;
    PG4IOCON2 &= 0xFFCFFFFF;
    PG5IOCON2 &= 0xFFCFFFFF;
    PG7IOCON2 &= 0xFFCFFFFF;
    PG8IOCON2 &= 0xFFCFFFFF;
}

void SecondaryOverrideDisable(void) {
    PG2IOCON2 &= 0xFFCFFFFF;
    PG6IOCON2 &= 0xFFCFFFFF;
    APG1IOCON2 &= 0xFFCFFFFF;
    APG2IOCON2 &= 0xFFCFFFFF;
    APG3IOCON2 &= 0xFFCFFFFF;
}
```

12. 版本历史

版本 A（2025 年 12 月）

本文档的初始版本。

Microchip 信息

商标

“Microchip”的名称和徽标组合、“M”徽标及其他名称、徽标和品牌均为 Microchip Technology Incorporated 或其关联公司和/或子公司在美国和/或其他国家或地区的注册商标或商标（“Microchip 商标”）。有关 Microchip 商标的信息，可访问 https://www.microchip.com/en-us/about/legal-information/microchip-trademarks。

ISBN： 979-8-3371-3250-1

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc.及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc.的英文原版文档。

本出版物及其提供的信息仅适用于 Microchip 产品，包括设计、测试以及将 Microchip 产品集成到您的应用中。以其他任何方式使用这些信息都将被视为违反条款。本出版物中的器件应用信息仅为您提供便利，将来可能会发生更新。您须自行确保应用符合您的规范。如需额外的支持，请联系当地的 Microchip 销售办事处，或访问 www.microchip.com/en-us/support/design-help/client-support-services。

Microchip“按原样”提供这些信息。Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对非侵权性、适销性和特定用途的适用性的暗示担保，或针对其使用情况、质量或性能的担保。

在任何情况下，对于因这些信息或使用这些信息而产生的任何间接的、特殊的、惩罚性的、偶然的或附带的损失、损害或任何类型的开销，Microchip 概不承担任何责任，即使 Microchip 已被告知可能发生损害或损害可以预见。在法律允许的最大范围内，对于因这些信息或使用这些信息而产生的所有索赔，Microchip 在任何情况下所承担的全部责任均不超出您为获得这些信息向 Microchip 直接支付的金额（如有）。如果将 Microchip 器件用于生命维持和/或生命安全应用，一切风险由买方自负。买方同意在由此引发任何一切损害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任。除非另外声明，在 Microchip 知识产权保护下，不得暗中或以其他方式转让任何许可证。

Microchip 器件代码保护功能

请注意以下有关 Microchip 产品代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术规范。
- Microchip 确信：在正常使用且符合工作规范的情况下，Microchip 系列产品非常安全。
- Microchip 注重并积极保护其知识产权。严禁任何试图破坏 Microchip 产品代码保护功能的行为，这种行为可能会违反《数字千年版权法案》（Digital Millennium Copyright Act）。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。