AN5380

在 KE1xF 上使用 FTM、PDB 和 ADC 驱动双 PMSM FOC 和 PFC

版本 0 — 2016 年 11 月

应用笔记

目录

1 简介

本应用笔记介绍了三合一系统的设计，该系统通过单个 KE1xF 芯片控制 PFC 和基于 FOC 的双 PMSM 电机。

三合一系统需要高性能的处理器和一组外围设备来同时驱动 PFC 和双 PMSM FOC。这三个控制器需要协调工作，并满足应用程序对系统控制的要求。

本文档介绍了 KE1xF 的 FlexTimer、PDB 和 ADC 功能，并以三合一演示系统的控制参数为例来说明系统时序设计。此外，本文还列出了 FlexTimer、PDB 和 ADC 的详细配置，以实现预期的系统时序。

注意

本文的描述基于给定示例的控制参数。但是，同样方法也适用于其他控制参数。

2 Kinetis KE1xF 子家族

与现有的 Kinetis E 系列的 MCU 相比，KE1xF 微控制器基于 ARM® Cortex®-M4 处理器架构，具有更高的性能、更丰富的外设集成和更高的存储密度。凭借这些先进的功能，KE1xF 可以同时驱动两个高端电机控制算法和传统上需要两个或多个微控制器的数字 PFC（功率因数校正）控制。

2.1 KE1xF 功能概述和框图

Kinetis KE1xF 芯片的功能摘要如下：

- 核处理器和系统：
  - 基于 ARMv7 架构和 Thumb-2 ISA 的 ARM Cortex-M4 内核
  - CPU 频率：高达 168 MHz，每兆赫兹 1.25 Dhrystone MIPS
  - 集成数字信号处理器（DSP）和单精度浮点单元（FPU）
  - 可配置的嵌套矢量中断控制器（NIVC）
  - 通过 DMAMUX 将 16 通道 DMA 控制器扩展到 64 通道
- 内存和内存接口：
  - 带有 ECC 的最大 512 KB 程序闪存
  - 带有 ECC 的最大 64 KB SRAM
  - 64 KB FlexNVM，带有 ECC，可用于数据闪存和 EEPROM 仿真
  - 4 KB FlexRAM，用于 EEPROM 仿真
  - 8 KB I/D 高速缓存，最大程度地降低内存访问延迟对性能的影响
  - 内置引导程序的 Boot ROM
- 混合信号模拟：

  - 3 个 12 位模数转换器（ADC），最高 1 M sps
  - 3 个内含 8 位数模转换器（DAC）的高速模拟比较器（CMP）
  - 1 个 12 位数模转换器（DAC）
- 计时和控制：
  - 4 个 Flex 计时器（FTM），提供最多 32 个标准通道
  - 3 个可编程延迟模块（PDB），具有灵活的触发系统，可为模块间同步提供准确的延迟和生成触发
  - 1 个低功耗定期中断计时器（LPIT）
  - 1 个低功耗计时器（LPTMR），在停止模式下工作
  - 脉冲宽度计时器（PWT）
  - 实时计时器时钟（RTC）
- 连接和通信接口：
  - TriggerMUX 用于内部模块的连接
  - LPUART、LPSPI、LPI2C、FlexIO 和 CAN（可选）
- 操作特性：
  - 具有齐全的闪存编程/擦除/读取功能，足够宽的工作电压范围，从 2.7 至 5.5 V
  - 环境工作温度范围为 -40 ℃ 至 105 ℃

图 1 展示了 MCU 超集合设备的顶层框图。FlexCAN 仅适用于部分器件。

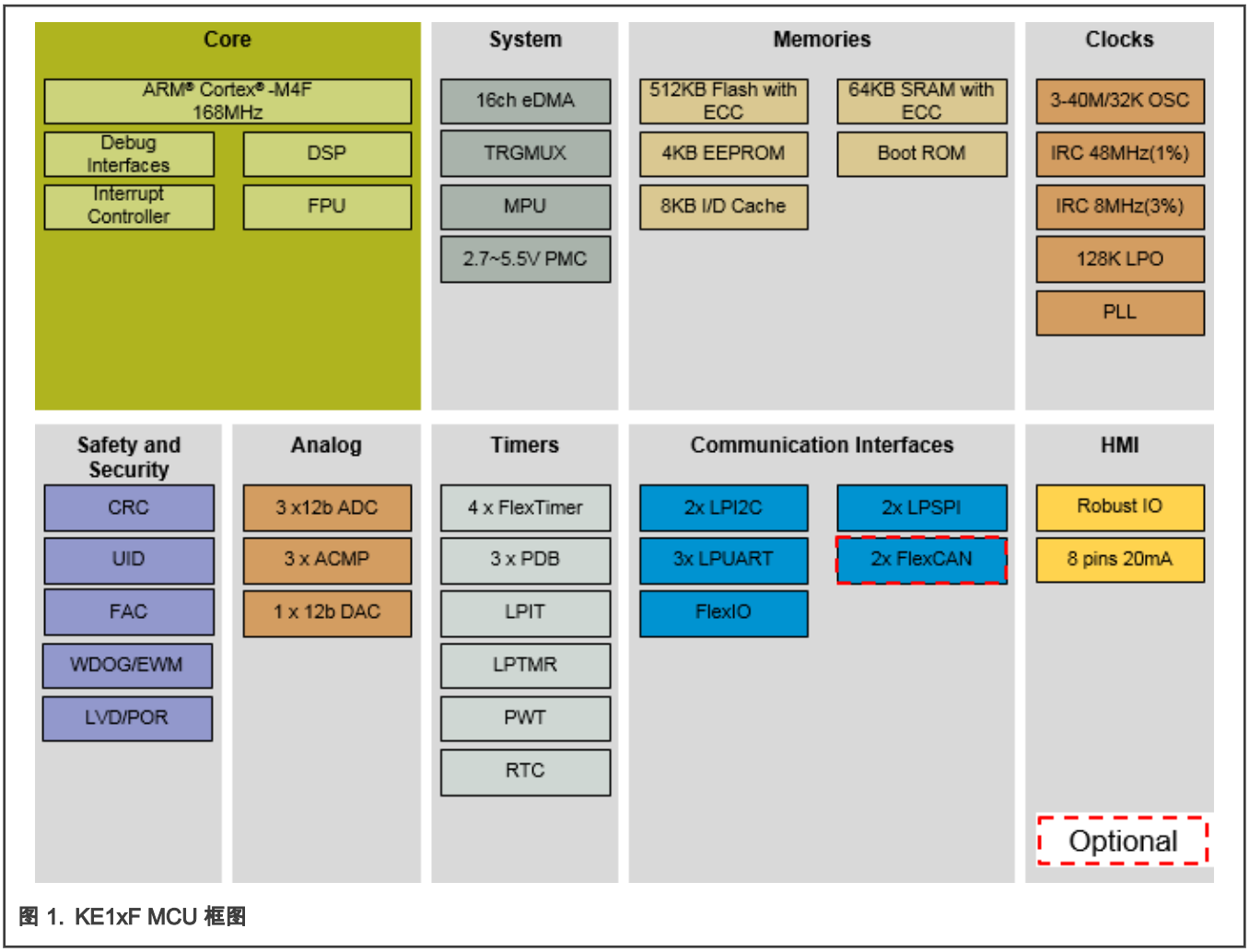

图 1. KE1xF MCU 框图

2.2 电机控制和 PFC 的接口

三合一系统在单个 KE1xF MCU 中同时运行具有高级电机磁场定向控制（FOC）算法和功率因数校正（PFC）的压缩机电机和风扇电机。同时运行的所有控件都从 FTM、PDB、ADC 和 TRGMUX 模块中受益匪浅。

FTM 提供灵活的配置，并可以实现高效的三相电动机控制或 PFC 控制。根据由写入到 C（n）V 和 C（ n + 1）V 寄存器控制的导通边缘和截断边缘，该 FTM 结合模式能够生成非对称 PWM 占空比和中心对称占空比。

FTM 模块具有以下功能：

- FTM 源时钟是可选择的
- 预分频器，可以实现 1、2、4、8、16、32、64 或 128 分频
- 16 位计数器：
  - 可以是自由运行的计数器，也可以是具有初始值和最终值的计数器
  - 计数可以递增或递减
- 可配置为输入捕获、输出比较或边沿对齐的 PWM 模式
- 在输入捕获模式下：
  - 捕获可以发生在上升沿、下降沿或两个边缘上
  - 可以为某些通道选择输入滤波器。一个预分频器可适用于所有滤波器

- 在输出比较模式下，在匹配时可以设置、清除或切换输出信号
- 所有通道均可配置为中心对齐的 PWM 模式
- 每对通道可以组合在一起，以独立控制 PWM 信号的两个边沿来生成 PWM 信号
- FTM 通道可以成对使用，使其具有相等的输出或者互补的输出；或用作独立的通道，具有独立的输出
- 死区（deadtime）插入适用于每个互补对
- 生成匹配触发器和初始化触发器
- PWM 输出的软件控制
- 多达四个故障输入，用于全局故障控制
- 每个通道的极性是可配置的
- 每个通道生成一个中断
- 计数器溢出时生成中断
- 检测到故障条件时生成中断
- 发生寄存器重载点时生成中断
- 写入缓冲的 FTM 寄存器的同步加载
- 半周期和全周期寄存器的重载容量
- 关键寄存器的写入保护
- 具备 TPM 的向后兼容。
- 测试输入捕获模式
- 直接访问输入引脚状态
- 双边沿捕获，用于脉冲和周期宽度测量
- 正交解码器，具有预缩放的输入滤波器、相对位置计数、位置计数中断或外部事件的位置计数捕获
- 对于 FTM 通道，可以选择在通道输出上生成触发脉冲，而不是在 PWM 上生成
- 抖动功能可以模拟 PWM 周期或 PWM 占空比的精确边沿控制

PDB 模块具有以下功能：

- TRGMUX 模块的一个输入源和一个软件触发源
- 在 KE1xF 上仅配置了一个 PDB 通道用于 ADC 硬件触发
  - 一个 PDB 通道与一个 ADC 相关联
  - 每个 PDB 通道为 ADC 硬件触发提供一个触发输出，为 ADC 触发选择提供多达 8 个预触发输出
  - 触发输出可以独立启用或禁用
  - 每个预触发输出配置一个 16 位延迟寄存器
  - 预触发输出的延迟寄存器具备可选旁路
  - 以单拍或连续模式操作
  - 可选择背对背模式，该操作使 ADC 转换完成以触发下一个 PDB 通道
  - 一个可编程的延迟中断
  - 一个顺序错误中断
  - 每个预触发配置一个通道标志和一个序列错误标志
  - 支持 DMA
- 在 KE1xF 上仅配置了一个 DAC 间隔触发

  - 每个 DAC 具备一个间隔触发输出
  - 每个 DAC 触发输出具备一个 16 位延迟间隔寄存器
  - 延迟间隔触发寄存器具备可选旁路
  - 可选的外部触发器
- 在 KE1xF 上仅配置了一个脉冲输出
  - 可以独立启用或禁用脉冲输出
  - 可编程的脉冲宽度

12 位 ADC 模块具有以下功能：

- 线性逐次逼近算法，分辨率最高为 12 位
- 多达 16 个单端外部模拟输入
- 输出模式：单端 12 位、10 位和 8 位模式
- 以右对齐的无符号格式单端输出
- 单次或连续转换，即单次转换后自动返回空闲状态
- 可配置的采样时间和转换速度/功耗
- 转换完成/硬件平均完成标志和中断
- 输入时钟最多可以从四个源中选择
- 在低功耗模式下运行可降低噪声
- 带有硬件通道选择的可选硬件转换触发器
- 自动比较中断，可编程为小于、大于或等于，范围内或范围外
- 温度传感器
- 硬件平均功能
- 可选参考电压：外部或备用
- 自校准模式

TRGMUX 引入了一种非常灵活的方法，用于将各种触发源连接到多个引脚/外设。它允许软件为各种外设配置触发输入。

3 系统控制要求

3.1 系统概念

图 2 介绍了带有交错式 PFC 的双电机控制的概念。它提出了空调的典型应用要求，其中一台电机控制压缩机，另一台电机控制风扇。

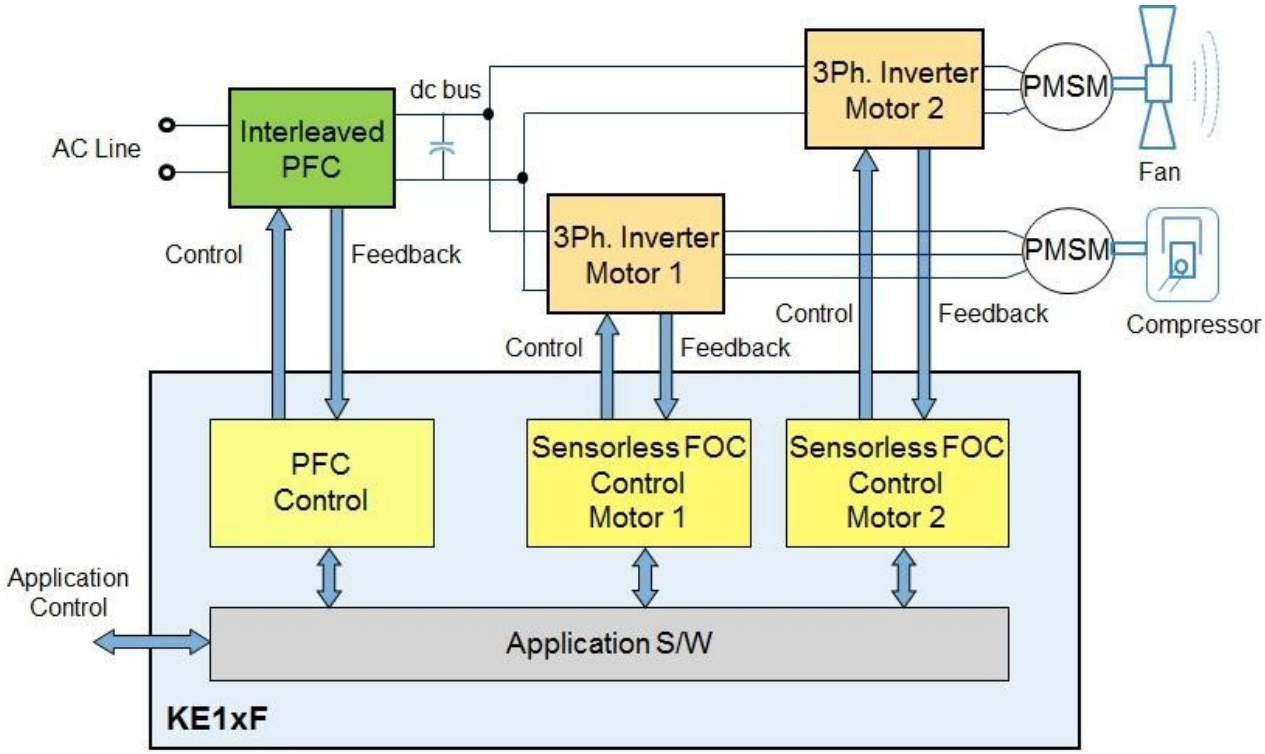

图 2. PFC 双电机控制概念

3.2 系统信号连接

该三合一应用程序包含两个无传感器永磁同步电动机的磁场定向控制以及功率因数校正的数字控制。KE1xF MCU 控制器具有许多有益于应用程序的功能。为了用单个 KE1xF MCU 处理器控制三个对象，需要使用几个外设，并准确分配对应的芯片引脚。电机分流电流的测量方法和 PFC 拓扑会影响引脚分配。电机分流电流测量值可以是单分流、两分流或三分流。

设计人员需要为不同的模拟信号分配不同的 ADC 实例引脚，以使系统时序尽可能简单。电动机的两相电流信号分配给不同的 ADC 实例（在此示例中为 ADC0 和 ADC1），以便可以同时测量两个电流。如果使用三分流测量方法，则电动机的第三分流相电流连接到两个 ADC 实例。对于 PFC，将输入电压和并联电流分配给不同的 ADC 实例，可以同时测量它们。

图 3 展示了如何将处理器信号连接到板上的电源电子设备。本应用笔记中给出的系统时序设计基于该连接示例，并且实现了两路电流测量。

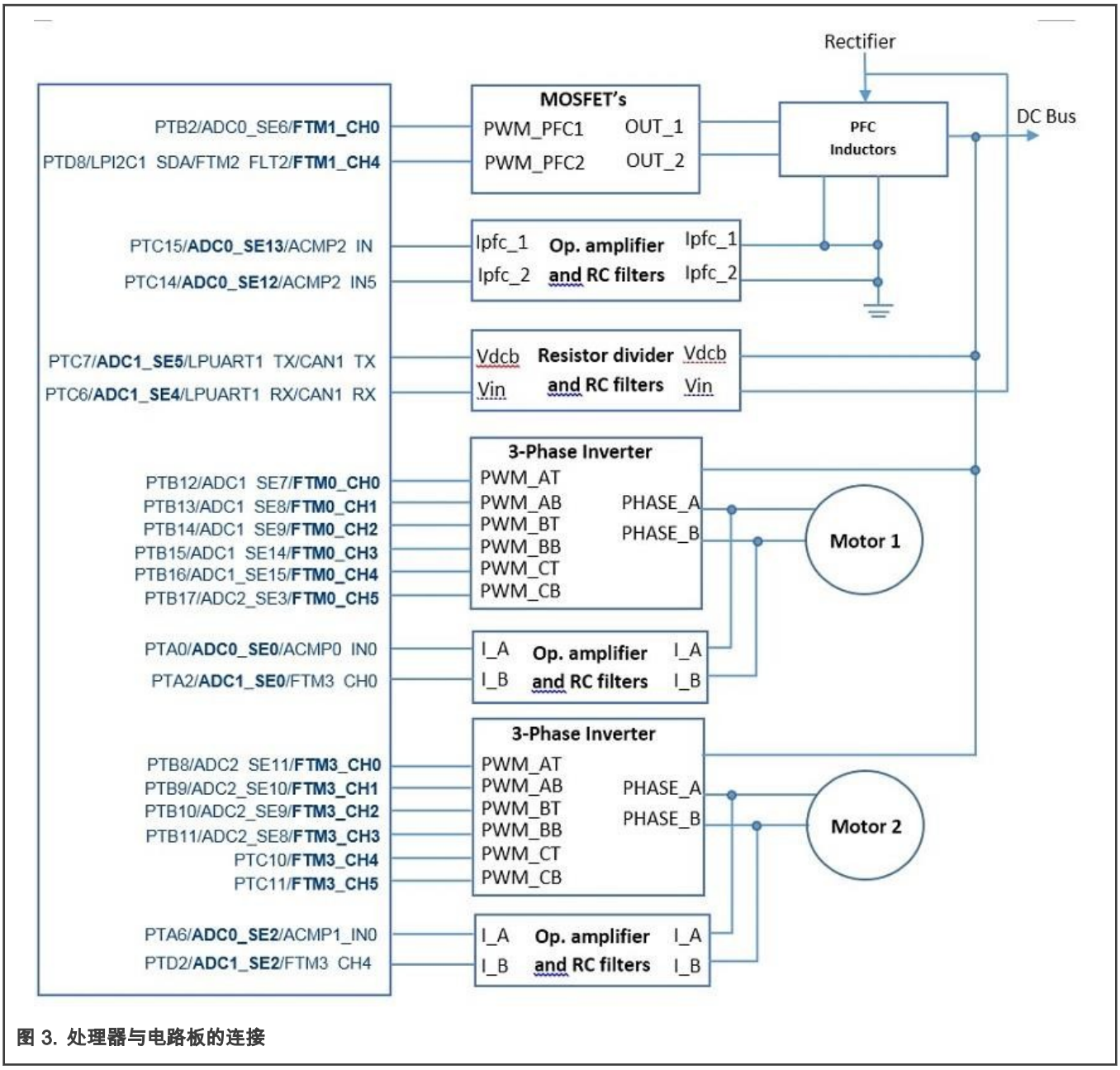

图 3. 处理器与电路板的连接

3.3 系统时间规格

三合一应用程序示例中每个组件的时间规格如下：

- 电机 1—压缩机驱动
  - — 通过两个分流电阻重构三相电动机电流
  - — PWM 频率 5 kHz
  - — 5 kHz 快速环路频率
  - — 1 kHz 慢速环路频率
- 电机 2 —风扇驱动
  - — 通过两个分流电阻重构三相电动机电流

  - PWM 频率 10 kHz
  - 10 kHz 快速环路频率
  - 1 kHz 慢速环路频率
- PFC
  - 2-MOSFET 交错模式
  - 两个分流电阻上的电流测量
  - PWM 频率 80 kHz
  - 20 kHz 快速环路频率
  - 500 Hz 慢速环频率

4 系统时序

时序对于三合一应用非常重要。这取决于如何利用外围资源以及如何分配 MCU 性能。三合一系统涉及三个控制对象：两个无传感器 FOC PMSM 电动机和一个交错式 PFC。每个控制对象都有两个或多个由特定控制算法确定的控制器。每个控制器的模拟信号测量将在特定时间点进行。从软件角度来看，在测量模拟信号并在下一个 PWM 周期的重载点前更新执行器之后，必须尽快计算每个控制器。很难为每个控制器预分配特定的时隙，并在时间范围内对这些时隙进行排序。

时序实现需要 MCU 外设的支持。有效利用外设可以简化时序设计。基于前面介绍的板上信号的连接，以下小节展示了内部模块的连接，并描述了示例应用程序的系统时序设计。

4.1 外设互联

凭借丰富的外围设备和灵活的互联功能，可以轻松实现复杂的时序，使得 KE1xF MCU 驱动具有 PFC 的双无传感器 PMSM FOC。图 4 展示了该示例应用程序的系统框图，其中 FTM、PDB、ADC 和 TRGMUX 相互连接。

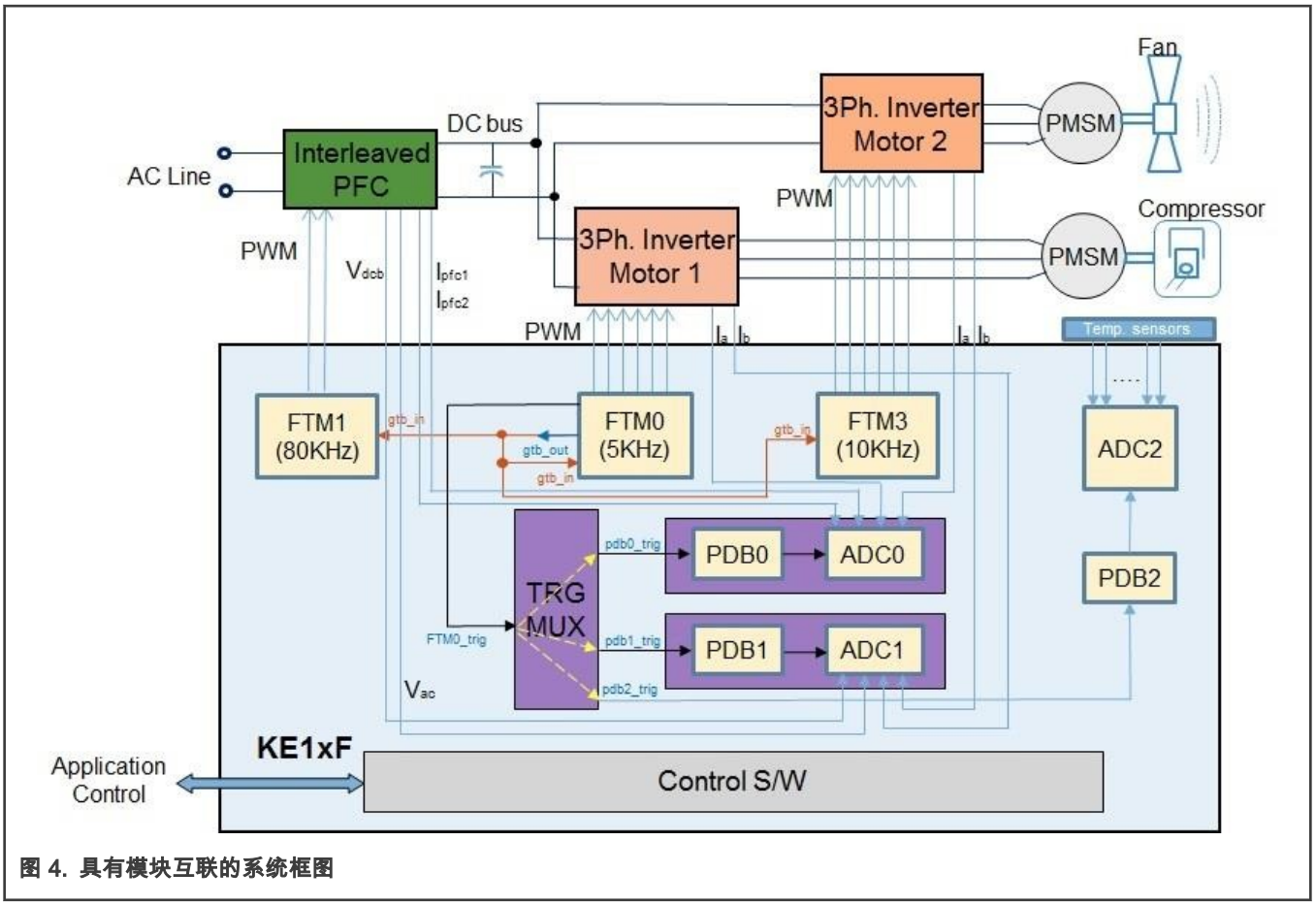

图 4. 具有模块互联的系统框图

例如，在此三合一应用程序中，FTM0、FTM3 和 FTM1 分别驱动压缩机、风扇和 PFC。FTM0 生成连接到 FTM0、FTM3 和 FTM1 的 gtb_in 信号的 gtb_out 信号，以进行 PWM 同步。FTM0 生成硬件触发信号作为 TRGMUX 的输入，并分别被选作 PDB0、PDB1 和 PDB2 的输入触发信号。PDBn 的输出预触发和触发器连接到 ADCn 以启动 ADC 转换。ADC0 和 ADC1 用于测量双电机和 PFC 的模拟信号。ADC2 用作系统温度信号和对时间要求不严格的其他模拟信号。电动机的两相并联电流由 ADC0 和 ADC1 测量，输入电压和 PFC 电流也是如此。

4.2 设计系统时序

系统时间规范定义了应用程序的基本时序。板上的系统信号连接和模块的互联是特定系统时序的基础结构，它考虑了何时测量模拟信号，何时计算控制器以及如何同步从三个 FTM 实例生成的 PWM。

无法为每个控制器计算预先分配时隙，但是， MCU 处理器为所有控制器的计算在运行时仍是周期性的。ADC 对模拟信号的测量也是周期性的。对于特定的系统时序，分别定义处理器计算周期的计算时间片段和 ADC 测量周期的测量时间片段。高达 168 MHz 的 KE1xF 处理器功能强大，足以在计算时间段内完成所有强制性计算。KE1xF 12 位 ADC 的转换速度足够快，可以在测量时间段内转换所有模拟信号。

示例应用程序的设计系统时序如 图 5 所示。

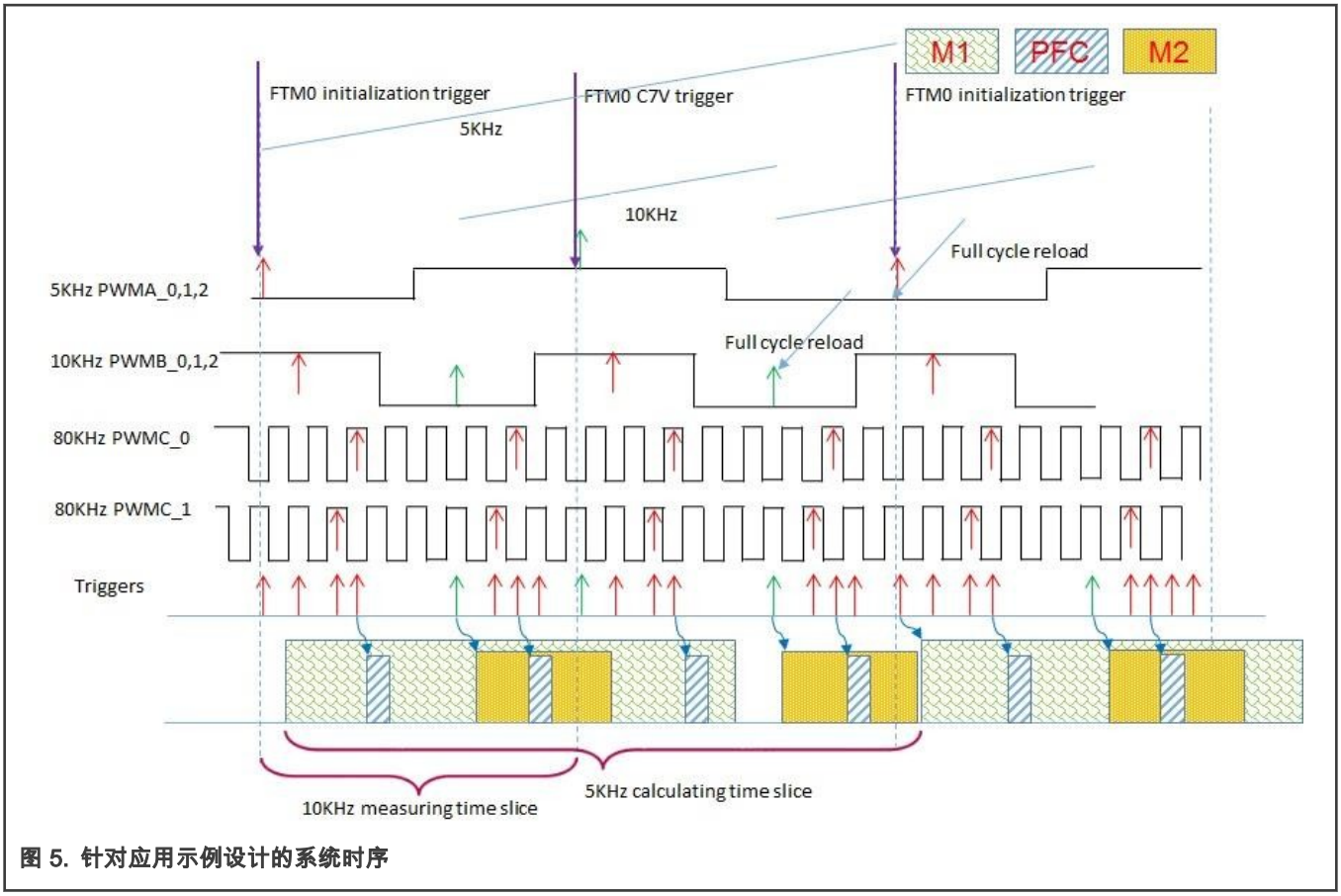

图 5. 针对应用示例设计的系统时序

M1、M2 和 PFC 的所有 PWM 都是中心对齐的 PWM。驱动 M1（压缩机电动机）的 FTM0 PWM 的开关频率为 5 KHz。用于 M2（风扇电机）的 FTM3 PWM 的开关频率为 10 kHz。用于 PFC 的 FTM1 PWM 的开关频率为 80 kHz。三个转换频率之比为 1：2：16，但所有 PWM 并非同时从零相位开始。 精心设计了 FTM1 和 FTM3 的溢出情况。FTM1 将以与 FTM0 完全相同的相位开始，并且它的八次溢出恰好发生在 FTM0 的第二次溢出时。FTM3 的第一次溢出恰好发生在 FTM0 计数器经过 12.5 us（PFC PWM 的 1 个周期）的时间，即 FTM1 的 PFC 的第二次溢出。

FTM0 通过 TRGMUX 产生两个触发信号作为 PDB0 和 PDB1 的输入。在示例应用系统时序中，在 FTM0 中设置初始化触发信号和与 C7V 匹配的外部触发信号。PDB0 和 PDB1 均通过八个延迟计数器生成八个预触发，以选择八个不同的 ADC 配置寄存器。PDB0 输出触发信号连接到 ADC0 输入触发信号，而 PDB1 输出触发信号连接到 ADC1 输入触发信号。这两个连接被配置为并行工作，并且它们同时测量模拟信号。与两个电动机和 PFC 相关的所有模拟信号均由 ADC0 和 ADC1 测量。

为了减少由 PWM 开关噪声引起的测量误差，尽可能将 ADC 测量点安排在靠近 PWM 中心点的位置。对于 PDB2 和 ADC2 连接，延迟配置非常灵活，因为所有温度测量都是缓慢的，并且对时间要求不严格。有关这些模拟信号的引脚分配，请参见系统信号连接。

示例系统时序中的计算时间片段为 200 us（5 KHz），与压缩机电机的 PWM 周期相同。模拟测量时间片段为 100 us（10 KHz），与风扇电机的 PWM 周期相同。每一个 FTM0 PWM 周期为 PDB 生成两个触发信号，以触发两个测量时间片段。根据系统时间规格，处理器的一个计算时间片段中将包含一个压缩机控制计算、两个风扇控制计算和四个 PFC 控制计算。在 ADC 的一个测量时间段内，两组 16 个模拟信号进行转换。

如何为 16 个模拟信号分配 ADC 结果寄存器也取决于系统时序。在本示例中，为压缩机 A 相和 B 相电流分配了 ADCx_R0（x 为 0、1），并且在每隔一个测量时间片段中轮流测量其偏移量或数值。ADCx_R1 用于风扇电机两相电流偏移：ADCx_R2 和 ADC_R3 用于交错 PFC 电流和输入电压； ADCx_R4 用于风扇电动机的两相电流。ADCx_R5 和 ADC_R6 再次用于交错 PFC 电流和输入电压。 ADCx_R7 用于直流母线电压。

为了在一个计算时间段内完成所有控制任务，分别为这些计算精心安排了特定的中断。我们可以将 ADC 转换完成中断的 ISR 用于所有计算，但这会增加中断延迟。启动转换完成中断的 ADC 结果寄存器需要进行检查，以确定是否分别针对 PFC、压缩机或风扇控制进行计算。较好的方法是对不同的控制对象使用不同的硬件中断 ISR。

PFC 控制计算将在 ADC 转换完成中断的 ISR 中完成。在每个 ADC 测量时间段中，仅完成了第 4 和第 7 转换的中断。风扇控制计算为 10 KHz，可以通过 PDB0、PDB1 或 PDB3 延迟中断 ISR 完成，因为它们的输入触发是以 10 KHz 频率生成的。在此示例应用程序中，PDB3 延迟中断被安排有特定的延迟值，该延迟值在每个 ADC 测量时间片段中完成第 5 次 ADC 转换之后，即在测量风扇电动机相电流之后。

压缩机控制计算为 5 KHz，与 FTM0 PWM 开关频率相同。但是，必须在测量压缩机电动机的相电流之后进行计算。计划中断的时间点必须位于 ADC 测量时间片段中第一个 ADC 转换完成之后的时间，其触发条件为 FTM0 初始化触发条件。在 ADC 测量时间段内完成的第一个 ADC 转换，由 C7V 匹配外部触发器触发，且仅用于测量压缩机相电流偏移。在该示例中，通过将 C6V 设置为特定值来满足上述时序要求，启用并调度 FTM0 通道 6 中断。FTM0 通道中断的 ISR 将计算压缩机电机控制。

中断的 ISR 具有不同的优先级值，方法是将 ADC0 设置为 1，将 PDB2 设置为 2，将 FTM0 设置为 3。这意味着 PFC 计算的优先级最高，压缩机计算的优先级最低，风扇计算的优先级为中。如 图 5 所示。

4.3 系统时序启动

如何启动系统时序是系统时序正确工作的关键。它包含三个因素：

- 如何同步 PWM
- 何时触发 ADC
- 何时启用 ISR 进行计算

FTM0、FTM3 和 FTM1 分别配置为压缩机电机、风扇电机和 PFC。为了实现上一小节中的设计系统时序，必须同步三个 FTM 实例，并且启动时，在 FTM 计数器寄存器中它们必须具有特定的值。

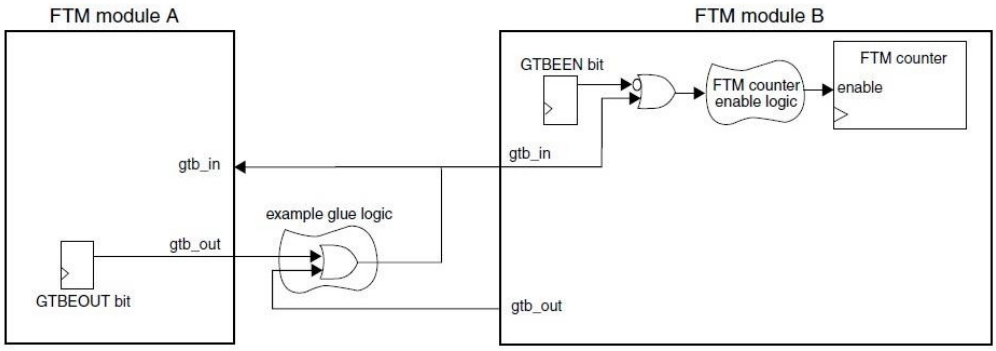

图 6. 全局时基框图

FTM 模块只有全局时基（GTB）方案可用。在示例应用程序中，所有 FTM 实例都与 FTM0 同步。仅使 FTM0 启用 GTBOUT 位，以使 FTM0 输出 gtb_out 信号。仅当 gtb_in 为 1 时，FTM0、FTM1 和 FTM3 的 GTBEEN 位才能启动计数器更新。GTB 功能不提供 FTM 计数器的连续同步。但是，如果为所有 FTM 计数器配置相同的时钟源和相同的时钟源分频因数，则 FTM 计数器在 FTM 操作期间不会失去同步。

图 7 展示了系统时序启动的时间。要使 FTM 计数器从特定值开始计数，需要将该特定值写入计数器的初始值寄存器（FTM_CNTIN）。在写入非零值至 CLKS [1：0]之前，将任何值写入 FTMs_CNT 寄存器都会将 FTM_CNTIN 寄存器中的初始值更新到计数器中，以启动 FTM 计数器。一旦将非零值写入 CLKS [1：0]以启动 FTM 计数器，就必须使用真实的初始值再次写入计数器的初始值寄存器（FTM_CNTIN），该初始值会在 FTM 计数器溢出时重新加载到 FTM 计数器中。

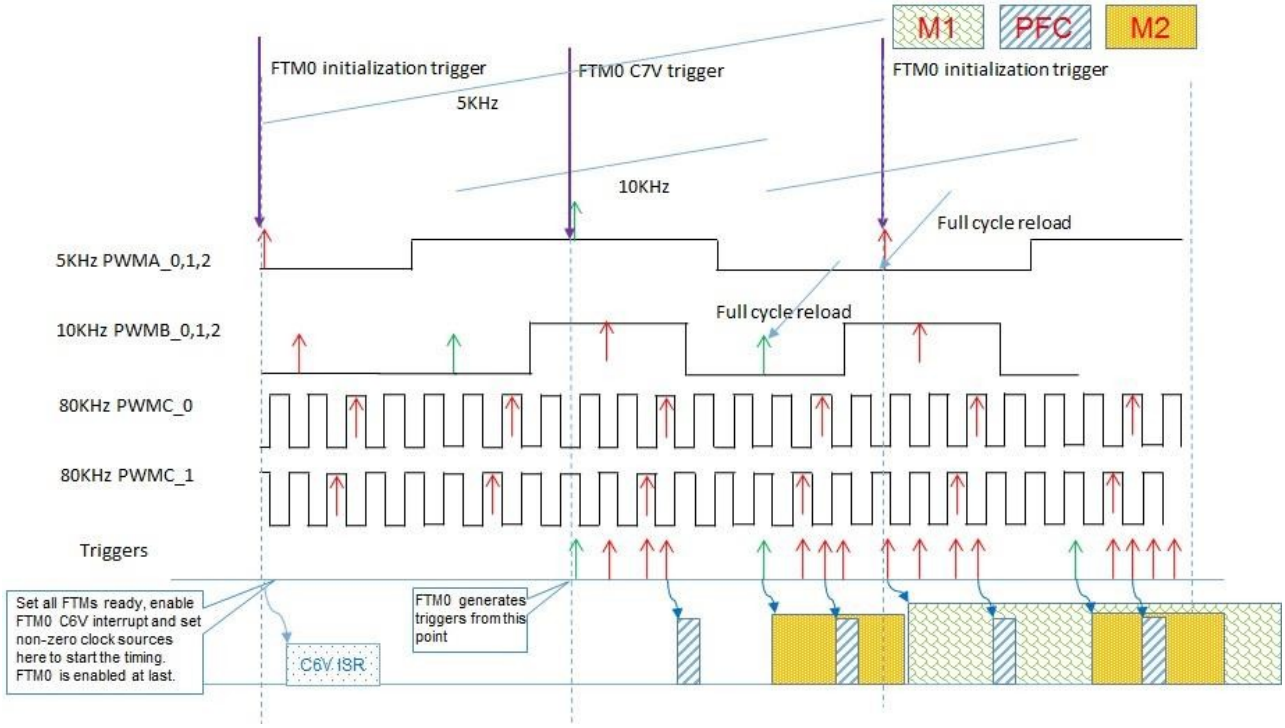

图 7. 系统时序启动方法

FTM0 初始化触发信号和 C7V 匹配的外部触发信号均在初始化例程中禁用。除了 FTM0 通道 6 的匹配中断，所有中断均被禁止。通过设置 FTM0 CLKS [1：0] ！= 0：0 来启动系统时序，使能 PWM，但禁止其输出，FTM 触发和中断也是如此。在 C6V 匹配的ISR 中，启用了触发信号和中断。将根据第二个 ADC 测量时间片段为 ADC 转换生成触发信号。在 图 7 中，它展示了启用 FTM0 之后的第一个 100us 周期，该时段用于系统启动处理。有效的系统时序是在 100 us 之后才开始。

5 模块配置

5.1 M1 的 PWM 配置

为 FTM0 配置定义了一些宏。示例应用程序的时钟配置设置为“高速运行”模式。PCC 模块将系统时钟 168 MHz 路由为 FTM 模块的外围接口时钟。最后，通过设置 CLKS [1：0] = 01 和 PS [2：0] = 0，将其选择为 FTM 输入时钟；通过 168 MHz FTM 输入时钟，PWM 模块生成 5 KHz PWM。因此，模数为 168 MHz / 5 kHz = 33600 个 tick。插入的死区时间约为 16 * 20/168 MHz≈2 us。开启时间和关闭时间由板上安装的 IPM 组件确定，可以离线测量。

```
/* Constant definition for 5KHz Motor 1 */
#define M1_MODULO_HALF 16800
#define M1_DEADTIME_VAL 20
#define M1_FTM_CNT_START (-M1_MODULO_HALF)
#define M1_DEADTIME_PS 3
#define M1_DEADTIME (16*20)
#define M1_IMP_UP 189
#define M1_IMP_DOWN 445
#define M1_ADC_DELAY ((M1_IMP_DOWN+M1_IMP_UP+M1_DEADTIME)/2)
#define M1_DISABLE_PWM_OUTPUT() FTM0->SC &= ~(FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK |
FTM_SC_PWMEN2_MASK\
 | FTM_SC_PWMEN3_MASK | FTM_SC_PWMEN4_MASK |
FTM_SC_PWMEN5_MASK)
#define M1_ENABLE_PWM_OUTPUT() FTM0->SC |= (FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK |
```

```
FTM_SC_PWMEN2_MASK\ | FTM_SC_PWMEN3_MASK |
FTM_SC_PWMEN4_MASK | FTM_SC_PWMEN5_MASK)
```

FTM0 配置为驱动压缩机电动机。必须通过配置外设时钟控制器模块（PCC）的 PCC FLEXTMR0 寄存器（PCC_FLEXTMR0）来启用 FTM0 时钟。

以下语句用于启用 FTM0 的时钟：

```
BITBAND_ACCESS32(&PCC_FLEXTMR0, PCC_CLKCFG_CGC_SHIFT) = 1;
```

禁用写入保护，并启用与兼容 TPM 功能不同的 FTM。

```
/* Disable the write-protection */
BITBAND_ACCESS32(&FTM0->MODE, FTM_MODE_WPDIS_SHIFT) = 1;
/* Enable FTM */
BITBAND_ACCESS32(&FTM0->MODE, FTM_MODE_FTMEN_SHIFT) = 1;
```

在功能模式下运行计数器；初始化触发信号是在计数器回滚（counter wrap）事件时生成的。每次重载都要设置 PWM 重载频率。

```
/* Counter running in the functional mode, ITRIGR=0; LDFQ=0 */
FTM0->CONF = FTM_CONF_BDMMODE(3);
```

过电流信号根据硬件连接到低电平有效的 FTM0 故障信号。必须启用故障控制并设置自动故障清除。故障中断被禁用。以下代码对故障进行了配置：

```
/* Enable fault control and automatic fault clearing */
FTM0->MODE |= FTM_MODE_FAULTM_MASK;
/* Disable fault interrupt */
BITBAND_ACCESS32(&FTM0->MODE, FTM_MODE_FAULTIE_SHIFT) = 0;
/* Active low polarity for fault 3 */
FTM0->FLTPOL = FTM_FLTPOL_FLT3POL_MASK;
/* Enable fault3, fault filter 5 system clocks, output safe state for fault */
FTM0->FLTCTRL = FTM_FLTCTRL_FFVAL(5) | FTM_FLTCTRL_FFLTR3EN_MASK | FTM_FLTCTRL_FAULT3EN_MASK;
```

在组合模式和互补模式下每对通道均配置为对称 PWM。在每对通道中启用了 PWM 更新同步、故障控制和死区插入。

即使没有 PWM 输出，通道（6）和通道（7）也被配置为组合模式，以避免进入输入捕捉模式。使用以下语句配置 FTM0 中的每对通道：

```
FTM0->COMBINE = FTM_COMBINE_FAULTEN0_MASK | FTM_COMBINE_SYNCEN0_MASK | FTM_COMBINE_DTEN0_MASK
              | FTM_COMBINE_COMP0_MASK | FTM_COMBINE_COMBINE0_MASK
              | FTM_COMBINE_FAULTEN1_MASK | FTM_COMBINE_SYNCEN1_MASK | FTM_COMBINE_DTEN1_MASK
              | FTM_COMBINE_COMP1_MASK | FTM_COMBINE_COMBINE1_MASK
              | FTM_COMBINE_FAULTEN2_MASK | FTM_COMBINE_SYNCEN2_MASK | FTM_COMBINE_DTEN2_MASK
              | FTM_COMBINE_COMP2_MASK | FTM_COMBINE_COMBINE2_MASK |
FTM_COMBINE_COMBINE3_MASK;
```

FTM 计数器初始值寄存器 CNTIN 设置为半模的负值。FTM 模寄存器 MOD 设置为半模-1 的正值。

```
/* Set PWM frequency */
FTM0->MOD = M1_MODULO_HALF-1;
FTM0->CNTIN = M1_FTM_CNT_START;
```

设置了死区时间插入。在周期开始时以及在通道（n + 1）匹配时（CNT = C（n + 1）V），通道（n）的输出被强制定为低电平；在通道（n）匹配时（CNT = C（n）V）被强制定为高电平。 PWM 占空比初始化为与 50%，占空比中心对齐。

```
/* Deadtime = PS*VAL*/
FTM0->DEADTIME = FTM_DEADTIME_DTPS(M1_DEADTIME_PS) |FTM_DEADTIME_DTVAL(M1_DEADTIME_VAL);
```

```
/* ELSnB:ELSnA = 1:0, the channle(n) ouput is forced low at the beginning of
* the period and on the channle(n+1) match (CNT = C(n+1)V). It's forced high
* on the channel(n) match (CNT = C(n)V).
*/
BITBAND_ACCESS32(&FTM0->CONTROLS[0].CnSC, FTM_CnSC_ELSB_SHIFT) = 1;
BITBAND_ACCESS32(&FTM0->CONTROLS[1].CnSC, FTM_CnSC_ELSB_SHIFT) = 1;
BITBAND_ACCESS32(&FTM0->CONTROLS[2].CnSC, FTM_CnSC_ELSB_SHIFT) = 1;
BITBAND_ACCESS32(&FTM0->CONTROLS[3].CnSC, FTM_CnSC_ELSB_SHIFT) = 1;
BITBAND_ACCESS32(&FTM0->CONTROLS[4].CnSC, FTM_CnSC_ELSB_SHIFT) = 1;
BITBAND_ACCESS32(&FTM0->CONTROLS[5].CnSC, FTM_CnSC_ELSB_SHIFT) = 1;

/* Initialize PWM with centeral aligned, 50% duty */
FTM0->CONTROLS[0].CnV = -M1_MODULO_HALF/2;
FTM0->CONTROLS[1].CnV = M1_MODULO_HALF/2 - 1;
FTM0->CONTROLS[2].CnV = -M1_MODULO_HALF/2;
FTM0->CONTROLS[3].CnV = M1_MODULO_HALF/2 - 1;
FTM0->CONTROLS[4].CnV = -M1_MODULO_HALF/2;
FTM0->CONTROLS[5].CnV = M1_MODULO_HALF/2 - 1;
```

使能通道 7 匹配以触发 PDB，将其设置为 0 以便在 PWM 周期的中间生成触发信号。该触发信号与初始化触发信号通过 TRGMUX 模块“或”在一起作为 PDB 的输入触发信号。 通道 6 寄存器设置为压缩机相电流测量完成后的时间以及尽可能早地计算压缩机控制。

```
/* Set Channel 7 to tigger the PDB0 and PDB1 */
FTM0->CONTROLS[7].CnV = 0; // Middle of PWM peroid
/* Set Channel 6 to schedule an interrupt to start triggers or motor 1 calculation */
FTM0->CONTROLS[6].CnV = -M1_MODULO_HALF + M1_CALCULATE_INT_DELAY;
```

启用通道 6 中断，其 ISR 已安装，用于启动触发信号和中断来启动系统时序。以下是在运行 PWM 之前如何配置 C6V 中断并设置 LDOK 位的方法。

```
/* Disable PWM channel output ports */
M1_DISABLE_PWM_OUTPUT();

/* Clear CH6 interrupt flag */
FTM0->CONTROLS[6].CnSC &= ~FTM_CnSC_CHF_MASK;

/* Enable CH6 interrupt to start input triggers of PDB */
BITBAND_ACCESS32(&FTM0->CONTROLS[6].CnSC, FTM_CnSC_CHIE_SHIFT) = 1;

/* Configure NVIC for FTM0 interrupt */
NVIC_EnableIRQ(FTM0_IRQn);
NVIC_SetPriority(FTM0_IRQn, ISR_PRIORITY_FTM0);

/* Set LDOK bit */
BITBAND_ACCESS32(&FTM0->PWMLOAD, FTM_PWMLOAD_LDOK_SHIFT) = 1;
```

在初始化期间，PWM 输出被禁用。由于 FTM 的输入时钟现在仍被禁用，因此现在无法生成 PWM。

5.2 M2 的 PWM 配置

FTM3 用于风扇电机，其配置与压缩机 FTM0 的配置几乎相同。风扇电机的 PWM 开关频率为 10 kHz。因此，模数为 168 MHz / 10 kHz = 16800 个 tick。所以，计数器初始寄存器 CNTIN 为-8400，而 MOD 寄存器为 8400-1。第二个区别是不使用通道 6 和通道 7 的数值寄存器，并且 FTM3 没有启用中断。最后一个区别是故障输入号。

下面列出了有关于 FTM3 产生用于风扇电机的 PWM 的宏：

```
/* Constant definition for 10KHz Motor 2 */ #define M2_MODULO_HALF    8400
#define M2_DEADTIME_VAL    16
#define M2_FTM_CNT_START      (-M2_MODULO_HALF/4)
```

```
#define M2_DEADTIME_PS     3
#define M2_DEADTIME       (16*16)
#define M2_IMP_UP         168
#define M2_IMP_DOWN       448
#define M2_ADC_DELAY      ((M2_IMP_DOWN+M2_IMP_UP+M2_DEADTIME)/2) // H/w propagation delay

#define M2_DISABLE_PWM_OUTPUT() FTM3->SC &= ~(FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK |
FTM_SC_PWMEN2_MASK\
| FTM_SC_PWMEN3_MASK | FTM_SC_PWMEN4_MASK | FTM_SC_PWMEN5_MASK)
#define M2_ENABLE_PWM_OUTPUT() FTM3->SC |= (FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN1_MASK |
FTM_SC_PWMEN2_MASK\
| FTM_SC_PWMEN3_MASK | FTM_SC_PWMEN4_MASK | FTM_SC_PWMEN5_MASK)
```

5.3 交错式 PFC 的 PWM 配置

应用中有两个 MOSFET 用于驱动交错式 PFC。两个 MOSFET 的 PWM 信号将中心对齐，并彼此偏移 180 度。一对通道不能输出互相偏移 180 度的两个 PWM。在应用程序中，使用两对通道来生成两个相互偏移 180 度的中心对齐 PWM。

PFC 的 PWM 频率为 80 KHz。为了产生 80 KHz，PWM 的模为 168 MHz / 80 KHz = 2100 tick。为 FTM1 计数器的初始值寄存器和模值寄存器定义了以下宏：

```
#define PFC_MODULO_HALF    1050 // 80KHz
#define PFC_FTM_CNT_START (-PFC_MODULO_HALF)

#define PFC_DISABLE_PWM_OUTPUT() FTM1->SC &= ~(FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN4_MASK)
#define PFC_ENABLE_PWM_OUTPUT() FTM1->SC |= (FTM_SC_PWMEN0_MASK | FTM_SC_PWMEN4_MASK)
```

第一步是通过配置外设时钟控制器模块（PCC）的 PCC FLEXTMR1 寄存器（PCC_FLEXTMR1）来启用 FTM1 时钟。以下语句用于启用 FTM1 的时钟：

```
/* Enable the FTM1 clock */
BITBAND_ACCESS32(&PCC_FLEXTMR1, PCC_CLKCFG_CGC_SHIFT) = 1;
```

启用 FTM 功能，并禁用写入保护模式。计数器在功能模式下运行；每次重载都会设置 PWM 重载频率。FTM1 模块有 3 个故障输入，分别是 DC-Bus 过压故障和 2 个过流故障。它们都是低电平有效的故障，并连接到输入 fault0、fault2 和 fault3。

以下是如何配置这些功能的代码：

```
/* Disable the write-protection */
BITBAND_ACCESS32(&FTM1->MODE, FTM_MODE_WPDIS_SHIFT) = 1;
/* Enable FTM */
BITBAND_ACCESS32(&FTM1->MODE, FTM_MODE_FTMEN_SHIFT) = 1;

/* Counter running in the debug mode */
FTM1->CONF |= FTM_CONF_BDMMODE(3);

/* Enable fault control and automatic fault clearing */
FTM1->MODE |= FTM_MODE_FAULTM_MASK;

/* Disable fault interrupt */
BITBAND_ACCESS32(&FTM1->MODE, FTM_MODE_FAULTIE_SHIFT) = 0;

/* Active high polarity for fault 0, 2, 3 */
FTM1->FLTPOL = 0;

/* Enable fault0, 2, 3, fault fliter 5 system clocks, output safe state for fault */
FTM1->FLTCTRL = FTM_FLTCTRL_FFVAL(5) | FTM_FLTCTRL_FFLTR0EN_MASK | FTM_FLTCTRL_FAULT0EN_MASK
              | FTM_FLTCTRL_FFLTR2EN_MASK | FTM_FLTCTRL_FAULT2EN_MASK
              | FTM_FLTCTRL_FFLTR3EN_MASK | FTM_FLTCTRL_FAULT3EN_MASK;
```

通道 0 和通道 1 这对被配置为组合和互补模式。PWM 更新同步和故障控制也被启用。PFC 的 PWM 中没有插入任何死区时间。通道 4 和通道 5 这对的配置相同。尽管配置了两对通道，但只有通道 0 和通道 4 的输出被用作两个 PWM 来驱动 PFC。为了实现两个互为 180 度的 PWM，在 PWM 周期开始时以及在 FTM1 计数器值与通道 1 值寄存器（CNT = C1V）匹配的情况下，通道 0 输出被强制设置为低电平。如果 FTM1 计数器值与通道 0 值寄存器（CNT = C0V）匹配，则输出将设置为高电平。可以通过设置 ELSB：ELSA = 1：0 来完成。另一方面，在该周期开始时以及在 FTM1 计数器值与通道 5 值寄存器（CNT = C5V）匹配的情况下，通道 4 的输出被强制为高电平。如果 FTM1 计数器值与通道 4 值寄存器（CNT = C4V）匹配，则输出被清除为低电平。可以通过设置 ELSB：ELSA = x：1 来完成。

以下语法用于配置 PWM 边沿或电平功能：

```
/* Center aligned PWM is used in combine mode
* COMBINE = 1 - combine mode set
* COMP = 1 - complementary PWM set
* SYNCEN = 1 - PWM update synchronization enabled
* FAULTEN = 1 - fault control enabled
*/

FTM1->COMBINE = FTM_COMBINE_FAULTEN0_MASK | FTM_COMBINE_SYNCEN0_MASK
       | FTM_COMBINE_COMP0_MASK | FTM_COMBINE_COMBINE0_MASK
       | FTM_COMBINE_FAULTEN2_MASK | FTM_COMBINE_SYNCEN2_MASK
       | FTM_COMBINE_COMP2_MASK | FTM_COMBINE_COMBINE2_MASK;
/* Set PWM frequency */
FTM1->MOD = PFC_MODULO_HALF-1;
FTM1->CNTIN = PFC_FTM_CNT_START;

/* ELSnB:ELSnA = 1:0, the channle(n) ouput is forced low at the beginning of
* the period and on the channle(n+1) match (CNT = C(n+1)V). It's forced high
* on the channel(n) match (CNT = C(n)V).
* ELSnB:ELSnA = x:1, the channle(n) ouput is forced high at the beginning of
* the period and on the channle(n+1) match (CNT = C(n+1)V). It's forced low
* on the channel(n) match (CNT = C(n)V).
* Set ELSnB:ELSnA = 1:0 for one PWM and ELSnB:ELSnA = x:1 for another PWM to
* to implement 180 degree phase shift between two PWMs for interleave PFC
*/
BITBAND_ACCESS32(&FTM1->CONTROLS[0].CnSC, FTM_CnSC_ELSB_SHIFT) = 1;
BITBAND_ACCESS32(&FTM1->CONTROLS[1].CnSC, FTM_CnSC_ELSB_SHIFT) = 1;
BITBAND_ACCESS32(&FTM1->CONTROLS[4].CnSC, FTM_CnSC_ELSA_SHIFT) = 1;
BITBAND_ACCESS32(&FTM1->CONTROLS[5].CnSC, FTM_CnSC_ELSA_SHIFT) = 1;
```

PWM 初始化为 0%占空比。以下是如何为两个彼此之间有 180 度偏移的 PWM 设置 0%占空比的方法：

```
/* Initialize PWM with centeral aligned, 0% duty */
FTM1->CONTROLS[0].CnV = -PFC_MODULO_HALF/2;
FTM1->CONTROLS[1].CnV = -PFC_MODULO_HALF/2;
FTM1->CONTROLS[4].CnV = -PFC_MODULO_HALF;
FTM1->CONTROLS[5].CnV = PFC_MODULO_HALF;
/* Initialize the channel output */
BITBAND_ACCESS32(&FTM1->MODE, FTM_MODE_INIT_SHIFT) = 1;
```

初始化后输出被禁用，最后一步是在运行 FTM1 之前设置 LDOK 位为 1。

```
/* Disable PWM channel output ports */
PFC_DISABLE_PWM_OUTPUT();

/* Set LDOK bit */
BITBAND_ACCESS32(&FTM1->PWMLOAD, FTM_PWMLOAD_LDOK_SHIFT) = 1;
```

为了生成彼此具有 180 度偏移的两个 PWM，需要一种特殊方法来计算从控制计算中得出的给定占空比的通道值。计算的值在下一个重载点之前写入 FTM1 通道值寄存器。 内联函数如下所示，该函数负责在下一次重载发生之前进行通道值计算和 PWM 更新。

```
inline void PFC_PWM_UPDATE(GMCLIB_2COOR_T_F16 *psDuty)
{
    int16_t w16ModuloHalf = PFC_MODULO_HALF;
    int16_t w16Result;

    /* Duty cycle limit [0 to 0.9] */
    if (!psDuty->f16A) /* 0% duty cycle */
    {
        FTM1->CONTROLS[0].CnV = -PFC_MODULO_HALF/2;
        FTM1->CONTROLS[1].CnV = -PFC_MODULO_HALF/2;
        FTM1->CONTROLS[4].CnV = -PFC_MODULO_HALF;
        FTM1->CONTROLS[5].CnV = PFC_MODULO_HALF;
 }
 else
 {
        /* PWM channel 0 */
        w16Result = MLIB_Mul_F16(psDuty->f16A, w16ModuloHalf);

        FTM1->CONTROLS[0].CnV = -w16Result;
        FTM1->CONTROLS[1].CnV = w16Result;

        /* PWM channel 4 */
        w16Result = MLIB_Mul_F16(32767 - psDuty->f16B, w16ModuloHalf);

        FTM1->CONTROLS[4].CnV = -w16Result;
        FTM1->CONTROLS[5].CnV = w16Result;
 }

 /* Set LDOK bit */
 BITBAND_ACCESS32(&FTM1->PWMLOAD, FTM_PWMLOAD_LDOK_SHIFT) = 1;
}
```

5.4 PWM 同步

FTM GTB 方案用于在系统启动时同步 PWM。第一步是将 FTM0、FTM1 和 FTM3 的 GTBEEN 位置 1，以便仅在其 gtb_in 信号为 1 时更新 FTM 计数器。

设置语法如下：

```
/* Enable GTBEEN for FTM0, FTM1 and FTM3 */
FTM0->CONF |= FTM_CONF_GTBEEN_MASK;
FTM1->CONF |= FTM_CONF_GTBEEN_MASK;
FTM3->CONF |= FTM_CONF_GTBEEN_MASK;
```

FTM0、FTM1 和 FTM3 计数器的初始值在配置期间会写入其初始值寄存器。现在将任何值写入计数器寄存器都会用其初始值寄存器 CNTIN 中的值更新计数器值。

配置如下：

```
FTM0->CNT = M1_MODULO_HALF;
FTM1->CNT = PFC_MODULO_HALF;
FTM3->CNT = M2_MODULO_HALF;
```

现在，将 FTM 输入时钟选择为带有 1 分频的 FTM 计数器时钟源，以用于所有 FTM。 可以通过以下方式完成：

```
/* Select the system clock with 1 divider for all FTMs */
FTM0->SC |= FTM_SC_CLKS(1);
FTM1->SC |= FTM_SC_CLKS(1);
FTM3->SC |= FTM_SC_CLKS(1);
```

FTM 时钟已启用，但由于启用了 GTBEEN 位，因此计数器仍被禁用。现在，所有具有已配置功能的 FTM 实例都可以启动了。将 FTM0 GTBEOUT 设置为 1 会生成全局时基信号，并使 FTM0、FTM1 和 FTM3 计数器从其初始值开始同步计数。以下为设置代码：

```
/* Set FTM0 GTBEOUT to generate the global time base signal */
FTM0->CONF |= FTM_CONF_GTBEOUT_MASK;
```

PWM 周期由写入 CNTIN 寄存器和 MOD 寄存器的值决定。一旦将 FTM0 GTBEOUT 位置 1，使能了三个 FTM 计数器，就必须用半模的负值更新三个初始值寄存器。初始化函数中写入的初始值仅用于在开始时用特定值更新计数器值。在最后一步，必须将半模的负值写入 CNTIN 寄存器以确定 PWM 周期。下面是写入三个 FTM CNTIN 寄存器的方法：

```
/* Set the FTM1 and FTM3 Counter initial value */
FTM1->CNTIN = - PFC_MODULO_HALF;
FTM3->CNTIN = - M2_MODULO_HALF;
FTM0->CNTIN = - M1_MODULO_HALF;
```

5.5 TRGMUX 配置

KE1xF 具有提供模块互连方案的 TRGMUX。仅有 FTM0 设置为产生初始化触发信号，以及由通道（7）匹配产生的外部触发信号。在芯片设计中，这两个 FTM 触发信号“或”在一起作为“TRGMUX”的一个输入触发信号。通过配置 TRGMUX，可以将该输入触发信号选择为 PDB 的输入触发信号。PDB 输入触发信号将生成 8 个预触发和触发信号。8 个预触发信号连接到 ADC 模块，以选择相应的 ADC 配置寄存器和结果寄存器。8 个触发信号在 PDB 中“或”在一起与 ADC 输入触发相连，以触发 ADC 转换。仅选择 PDB 的输入触发需要通过软件进行配置。其他连接由芯片中的硬件设计完成。

以下代码显示了如何在应用程序中分别为 PDB0、PDB1 和 PBD2 选择输入触发信号：

```
/* Enable the TRGMUX clock */
BITBAND_ACCESS32(&PCC_TRGMUX, PCC_CLKCFG_CGC_SHIFT) = 1;

/* Mux ACMP0_OUT (TRGMUX_IN17) to TRGMUX_EXTOUT7(TRGMUX_OUT11) */
TRGMUX0->TRGCFG[TRGMUX_EXTOUT1_INDEX] |= TRGMUX_TRGCFG_SEL3(17);

/* Mux FTM0_TRIG(TRGMUX0_IN11) to PDB0_TRG_IN (TRGMUX_OUT56) */
TRGMUX0->TRGCFG[TRGMUX_PDB0_INDEX] |= TRGMUX_TRGCFG_SEL0(11);

/* Mux FTM0_TRIG(TRGMUX0_IN11) to PDB1_TRG_IN (TRGMUX_OUT60) */
TRGMUX0->TRGCFG[TRGMUX_PDB1_INDEX] |= TRGMUX_TRGCFG_SEL0(11);

/* Mux FTM0_TRIG(TRGMUX0_IN11) to PDB2_TRG_IN (TRGMUX_OUT60) */
TRGMUX0->TRGCFG[TRGMUX_PDB2_INDEX] |= TRGMUX_TRGCFG_SEL0(11);
```

5.6 PDB 配置

PDB0 和 PDB1 配置为生成预触发和触发信号，以分别触发 ADC0 和 ADC1 转换。它们并行工作。因此，除了 PDB0 启用了错误中断但 PDB1 没有启用中断之外，它们的配置相同。PDB2 的配置与 PDB0 相似，但是仅从输入触发信号产生具有预定义延迟的中断。详细配置将在以下内容中给出。

第一步是通过配置外设时钟控制器模块（PCC）的 PCC_PDB0 寄存器来启用 PDB 模块时钟。以下是启用 PDB0 时钟的方法：

```
/* Enable the PDB0 clock */
BITBAND_ACCESS32(&PCC_PDB0, PCC_CLKCFG_CGC_SHIFT) = 1;
```

PDB 的输入来自 FTM0，预触发、触发和延迟中断的所有延迟均基于输入触发。PDB 的模寄存器未使用，设置为最大值，即 0xFFFF。

```
/* Set the modulus register with max value */
PDB0->MOD = 0xFFFF;
```

在将 1 写入 LDOK 位后，如果检测到触发输入事件，则 PDB 缓冲器的值会加载到内部寄存器中。输入时钟设置为与系统时钟完全相同，即 168 MHz。由于硬件设计只有 TRGMUX 提供一个触发输入，因此选择了触发输入源 0。下面是配置 PDB 功能的方法。

对于 PDB0，错误中断启用。

```
/* LDMOD=b10, PDBEIE=1, TRGSEL=0(TRGMUX_PDB_TRIG), PRESCALER=0, MULT=0, CONT=0, PDBEN=1*/
PDB0->SC = PDB_SC_LDMOD(2) | PDB_SC_PDBEIE(1) | PDB_SC_TRGSEL(0) | PDB_SC_PDBEN_MASK;
```

PDB1：

```
/* LDMOD=b10, TRGSEL=0, PRESCALER=0, MULT=0, CONT=0, PDBEN=1*/
PDB1->SC = PDB_SC_LDMOD(2) | PDB_SC_TRGSEL(0) | PDB_SC_PDBEN_MASK;
```

PDB2：

```
/* LDMOD=b10, TRGSEL=0, PRESCALER=0, MULT=0, CONT=0, PDBEN=1*/
PDB2->SC = PDB_SC_LDMOD(2) | PDB_SC_TRGSEL(0) | PDB_SC_PDBEN_MASK;
```

由于所有 8 个延迟寄存器都用于生成 8 个预触发，因此禁用了预触发旁路模式。延迟值是根据系统时序设计时如何安排完成模拟信号测量来预先定义的。有关详细信息，请参见上面的系统时序图。

以下是 PDB 配置中使用的宏。

```
/* Constant for PDB settings */
#define PDB01_CH0_DELAY_USED           0xFF // Total 8 channels are used
#define M2_CALCULATE_INT_DELAY    (PDB01_CH0DLY4_VALUE+315)
#define M1_CALCULATE_INT_DELAY    (M1_ADC_DELAY+315)
#define PDB01_CH0DLY0_VALUE            M1_ADC_DELAY
#define PDB01_CH0DLY1_VALUE           (PFC_MODULO_HALF/2*4+M2_ADC_DELAY)
#define PDB01_CH0DLY2_VALUE           (PFC_MODULO_HALF/2*8)
#define PDB01_CH0DLY3_VALUE           (PFC_MODULO_HALF/2*10)
#define PDB01_CH0DLY4_VALUE           (PFC_MODULO_HALF/2*20+M2_ADC_DELAY)
#define PDB01_CH0DLY5_VALUE           (PFC_MODULO_HALF/2*24)
#define PDB01_CH0DLY6_VALUE           (PFC_MODULO_HALF/2*26)
#define PDB01_CH0DLY7_VALUE           (PFC_MODULO_HALF/2*26+480)
```

下面给出了配置 PDB0 和 PDB1 的 PDB 延迟的语法。

PDB0：

```
/* Set the pre-trigger delay register effective (DLY0~DLY7)*/
PDB0->CH[0].C1 = PDB_C1_TOS(PDB01_CH0_DELAY_USED) | PDB_C1_EN(PDB01_CH0_DELAY_USED);
/* Set channel delay for the pre-trigger */
PDB0->CH[0].DLY[0] = PDB01_CH0DLY0_VALUE;
PDB0->CH[0].DLY[1] = PDB01_CH0DLY1_VALUE;
PDB0->CH[0].DLY[2] = PDB01_CH0DLY2_VALUE;
PDB0->CH[0].DLY[3] = PDB01_CH0DLY3_VALUE;
PDB0->CH[0].DLY[4] = PDB01_CH0DLY4_VALUE;
PDB0->CH[0].DLY[5] = PDB01_CH0DLY5_VALUE;
PDB0->CH[0].DLY[6] = PDB01_CH0DLY6_VALUE;
PDB0->CH[0].DLY[7] = PDB01_CH0DLY7_VALUE;
```

PDB1：

```
/* Set the pre-trigger delay register effective (DLY0~DLY7)*/
PDB1->CH[0].C1 = PDB_C1_TOS(PDB01_CH0_DELAY_USED) | PDB_C1_EN(PDB01_CH0_DELAY_USED);

/* LDMOD=b10, TRGSEL=0, PRESCALER=0, MULT=0, CONT=0, PDBEN=1*/
PDB1->SC = PDB_SC_LDMOD(2) | PDB_SC_TRGSEL(0) | PDB_SC_PDBEN_MASK;

/* Set channel delay for the pre-trigger */
PDB1->CH[0].DLY[0] = PDB01_CH0DLY0_VALUE;
PDB1->CH[0].DLY[1] = PDB01_CH0DLY1_VALUE;
PDB1->CH[0].DLY[2] = PDB01_CH0DLY2_VALUE;
PDB1->CH[0].DLY[3] = PDB01_CH0DLY3_VALUE;
PDB1->CH[0].DLY[4] = PDB01_CH0DLY4_VALUE;
PDB1->CH[0].DLY[5] = PDB01_CH0DLY5_VALUE;
PDB1->CH[0].DLY[6] = PDB01_CH0DLY6_VALUE;
PDB1->CH[0].DLY[7] = PDB01_CH0DLY7_VALUE;
```

PDB0 使能错误中断以处理 PDB 序列错误的情况。通过在结果寄存器中读取转换后的值来完成转换后，ADC coco 信号被发送到 PDB，并禁用 PDB 锁定状态。如果在未禁用 PDB 的情况下检测到另一个触发信号，则会产生 PDB 序列错误中断。处理此类错误中断很重要。因为 PFC 控制计算是在由 PDB 触发的 ADC 转换中断中完成的。如果发生错误，PDB-ADC 触发方案将停止并且不再有 ADC 中断。作为结果，驱动 PFC MOSFET 的 PWM 将停止更新，从而损坏 MOSFET。

以下代码是为 PDB0 配置中断。

```
/* Clear PDB0 interrupt flag */
BITBAND_ACCESS32(&PDB0->SC, PDB_SC_PDBIF_SHIFT) = 0;

/* Configure NVIC for PDB0 interrupt */
NVIC_EnableIRQ(PDB0_IRQn);
NVIC_SetPriority(PDB0_IRQn, ISR_PRIORITY_PDB0);
```

PDB2 使能延迟中断来计算风扇电动机控制计算。延迟值是根据电动机相电流测量时间点预定义的。在系统定时开始阶段，在 FTM0 C6V 匹配 ISR 中启用该中断。下面是配置它的代码：

```
/* Schedule the PDB interrupt delay to calculate M2 controllers */
PDB2->IDLY = M2_CALCULATE_INT_DELAY;

/* Clear PDB2 interrupt flag */
BITBAND_ACCESS32(&PDB2->SC, PDB_SC_PDBIF_SHIFT) = 0;

/* Configure NVIC for PDB2 interrupt */
NVIC_EnableIRQ(PDB2_IRQn);
NVIC_SetPriority(PDB2_IRQn, ISR_PRIORITY_PDB2);
```

最后一步是设置 LDOK 位为 1 以从先前写入的内部缓冲器更新 PDB 寄存器。

以下是设置 PDB0 的 LDOK 位的代码。

```
/* Set LDOK for PDB0 */
BITBAND_ACCESS32(&PDB0->SC, PDB_SC_LDOK_SHIFT) = 1;
```

5.7 ADC 配置

ADC0 和 ADC1 用于转换所有时序敏感的模拟信号。两个 ADC 实例几乎具有相同的配置，因为它们始终以相同的方式并行工作。但是 ADC0 在某些结果上允许转换完成中断，但没有为 ADC1 启用中断。另一个区别是输入通道选择。为 ADC0 配置的 ADC 时钟配置，采样时间和参考电压选择，自校准等也适用于 ADC1。在此仅给出 ADC1 的差异部分。

ADC 的前四个预触发和触发信号可以配置为不同的触发源。默认情况下来自 PDB。

如以下代码段所示，在此处对其进行了显式配置。

```
/* The first 4 pre-triggers and triggers are all from PDB (default) */
SIM->ADCOPT = SIM_ADCOPT_ADC0TRGSEL(0) | SIM_ADCOPT_ADC0PRETRGSEL(0)
            | SIM_ADCOPT_ADC1TRGSEL(0) | SIM_ADCOPT_ADC1PRETRGSEL(0);
```

通过配置外设时钟控制器模块（PCC）的 PCC_ADC0 寄存器，可以从 FIRCDIV2_CLK 时钟中选择 ADC 时钟源。必须在更改时钟源之前禁用 ADC 时钟，并在更改之后再次使能 ADC 时钟。在应用程序中，FIRCDIV2_CLK 被配置为 48 MHz。ADC 需要最少 275 ns 的采样时间，大约是 48 MHz 时钟的 13 + 1 个周期。ADC 参考电压设置为外部引脚。

以下代码针对这些配置进行了设置。

```
/* Disable the clock for ADC0 first*/
BITBAND_ACCESS32(&PCC_ADC0, PCC_CLKCFG_CGC_SHIFT) = 0;

/* Select the FIRCDIV2 (48MHz) as the alternate clock1 of ADC1 */
PCC_ADC0 |= PCC_CLKCFG_PCS(3);

/* Enable the clock for ADC0*/
BITBAND_ACCESS32(&PCC_ADC0, PCC_CLKCFG_CGC_SHIFT) = 1;

/* Clock source: ADC_ALTCLK1; Divide ratio: 1, MODE: 12bit */
ADC0->CFG1 = ADC_CFG1_ADIV(0) | ADC_CFG1_MODE(1) | ADC_CFG1_ADICLK(0);

/* Sample time is 13 for 48MHz clock, 275ns is minimum value */
ADC0->CFG2 = ADC_CFG2_SMPLTS(13);

/* VREFH/VREFL as ADC ADC voltage reference; Software trigger enabled */
ADC0->SC2 = 0;
```

ADC 具有校准机制，可提高采样精度。上电或复位后必须校准 ADC，以获得数据手册中指定的高精度。参阅应用笔记 *ADC Calibration on Kinetis E+ Microcontrollers*（文档 AN5314）以获得有关如何校准 ADC 的详细信息。以下代码显示了如何使用 AN5314 中介绍的方法校准 ADC：

```
/* OFS, CLP9, CLPX are signed number. The highest position bit is the signal bit.
Other calibration value registers are unsigned number. */
int32_t OFS, CLP9, CLPX;
uint32_t CLPS, CLP0, CLP1, CLP2, CLP3, Typ1, Typ2, Typ3;

/* H/W average enabled, 32 samples averaged, Continuous conversion disabled */
ADC0->SC3 = ADC_SC3_AVGE_MASK | ADC_SC3_AVGS(3);

/* Clock should be smaller than 25MHz for calibration */
ADC0->CFG1 |= (ADC0->CFG1 & ~ADC_CFG1_ADIV_MASK) | ADC_CFG1_ADIV(1);

/* starting the calibration of ADC0 */
BITBAND_ACCESS32(&ADC0->SC3, ADC_SC3_CAL_SHIFT) = 1;
/* Wait until the calibration complets */
while(!BITBAND_ACCESS32(&ADC0->SC1[0], ADC_SC1_COCO_SHIFT));

/* Check the calibration status */
/* Get raw calibration result */
OFS = (int32_t)((ADC0->OFS & ADC_OFS_OFS_MASK) >> ADC_OFS_OFS_SHIFT);
CLP9 = (int32_t)((ADC0->CLP9 & ADC_CLP9_CLP9_MASK) >> ADC_CLP9_CLP9_SHIFT);
CLPX = (int32_t)((ADC0->CLPX & ADC_CLPX_CLPX_MASK) >> ADC_CLPX_CLPX_SHIFT);
CLPS = ((ADC0->CLPS & ADC_CLPS_CLPS_MASK) >> ADC_CLPS_CLPS_SHIFT);
CLP0 = ((ADC0->CLP0 & ADC_CLP0_CLP0_MASK) >> ADC_CLP0_CLP0_SHIFT);
CLP1 = ((ADC0->CLP1 & ADC_CLP1_CLP1_MASK) >> ADC_CLP1_CLP1_SHIFT);
CLP2 = ((ADC0->CLP2 & ADC_CLP2_CLP2_MASK) >> ADC_CLP2_CLP2_SHIFT);
```

```
CLP3 = ((ADC0->CLP3 & ADC_CLP3_CLP3_MASK) >> ADC_CLP3_CLP3_SHIFT);
Typ1 = (CLP0 + CLP0);
Typ2 = (CLP1 + CLP1 - 26U);
Typ3 = (CLP2 + CLP2);

/* Transform raw calibration result to unified type int32_t when the conversion result value
is signed number. */
OFS = ADC12_TRANSFORM_CALIBRATION_RESULT(OFS, 16);
CLP9 = ADC12_TRANSFORM_CALIBRATION_RESULT(CLP9, 7);
CLPX = ADC12_TRANSFORM_CALIBRATION_RESULT(CLPX, 7);

/* Check the calibration result value with its limit range. */

if ((OFS < -48) || (OFS > 22) || (CLP9 < -12) || (CLP9 > 20) || (CLPX < -16) || (CLPX > 16)
||
(CLPS < 30U) || (CLPS > 120U) || (CLP0 < (CLPS - 14U)) || (CLP0 > (CLPS + 14U)) ||
(CLP1 < (Typ1 - 16U)) || (CLP1 > (Typ1 + 16U)) || (CLP2 < (Typ2 - 20U)) || (CLP2 > (Typ2 +
20U)) ||
(CLP3 < (Typ3 - 36U)) || (CLP3 > (Typ3 + 36U)))
{
 return 1; // Calibration failure
}

/* Recover the clock divide ratio */
ADC0->CFG1 |= (ADC0->CFG1 & ~ADC_CFG1_ADIV_MASK) | ADC_CFG1_ADIV(0);
```

ADC 校准完成后，将禁用硬件平均和连续转换，并为应用程序启用硬件触发。设置方法如下所示：

```
/* H/W average and Continuous conversion disabled */
ADC0->SC3 = 0;

/* H/W trigger enabled */
BITBAND_ACCESS32(&ADC0->SC2, ADC_SC2_ADTRG_SHIFT) = 1;
```

现在，分别为 8 个配置寄存器设置了 ADC 输入通道选项。启用第 4 和第 7 个结果的转换完成中断，其 ISR 将计算 PFC 控制器。以下代码给出了 ADC0 的详细输入通道选项及其中断设置。

```
/* Set ADC0 channels, enable SC1C and SC1E interrupt for PFC calculation
* S0 = SE0: I_A_Comp offset/Current
* S1 = SE2: I_A_Fan offset
* S2 = SE12: I_pfc2
* S3 = SE13: I_pfc1
* S4 = SE2: I_A_Fan current
* S5 = SE12: I_pfc1
* S6 = SE13: I_pfc2
* S7 = x
*/
ADC0->SC1[0] = 0;
ADC0->SC1[1] = 2;
ADC0->SC1[2] = 12;
ADC0->SC1[3] = 13 | ADC_SC1_AIEN_MASK; // Enable interrupt
ADC0->SC1[4] = 2;
ADC0->SC1[5] = 12;
ADC0->SC1[6] = 13 | ADC_SC1_AIEN_MASK; // Enable interrupt
ADC0->SC1[7] = 13;

/* Configure NVIC for ADC0 interrupt */
NVIC_EnableIRQ(ADC0_IRQn);
NVIC_SetPriority(ADC0_IRQn, ISR_PRIORITY_ADC0);
```

对于 ADC1，仅配置输入通道选项。

```
/* Set ADC1 channels
     * S0 = SE0: I_B_Comp offset/Current
     * S1 = SE2: I_B_Fan offset
     * S2 = SE4: Vin
     * S3 = SE4: Vin
     * S4 = SE2: I_B_Fan current
     * S5 = SE4: Vin
     * S6 = SE4: Vin
     * S7 = SE5: VDcbus
     */
   ADC1->SC1[0] = 0;
   ADC1->SC1[1] = 2;
   ADC1->SC1[2] = 4;
   ADC1->SC1[3] = 4;
   ADC1->SC1[4] = 2;
   ADC1->SC1[5] = 4;
   ADC1->SC1[6] = 4;
   ADC1->SC1[7] = 5;
```

5.8 模块配置顺序

除了前几节介绍的 FTM，PDB 和 ADC 配置外，模块配置还包括系统时钟初始化，GPIO 和 PORT 初始化。

下面列出了相关的函数原型：

```
/* Peripheral initializing methods */
void Apps_Init(void);
void Clock_Init(void);
void PORT_Init(void);
void FTM0_Init(void);
void FTM1_Init(void);
void FTM3_Init(void);
void FTMS_Sync(void);
void TRGMUX_Init(void);
void PDB012_Init(void);
int32_t ADC01_Init(void);
```

应用程序的初始化要求按特定顺序调用函数，以使其相互连接并按设计的系统时序运行。

函数 Apps_Init()按如下所示的适当顺序调用初始化函数。

```
void Apps_Init(void)
{
 /* Initialize system clock */
 Clock_Init();

 /* Initialize all Port/GPIO */
 PORT_Init();

 /* Initialize internal signal connection */
 TRGMUX_Init();

 /* Initailize the ADC for analog measurements */
 ADC01_Init();

 /* Initialize the PDBs */
 PDB012_Init();
```

```
  /* Initialize FTM0 for compressor */
  FTM0_Init();

  /* Initialize FTM1 for PFC */
  FTM1_Init();

  /* Initialize FTM3 for FAN */
  FTM3_Init();

  /* Initialize FreeMASTER */
  FMSTR_Init();

  /* Synchronize the FTMs */
  FTMS_Sync();
}
```

6 示波器上的系统时序

以下示波器屏幕截图显示了已实现的系统时序。在压缩机、风扇和 PFC 控制计算中断中，来自 GPIO 的信号在控制中断开始时设置，并在该中断结束时清除。可以观察到不同控制计算的持续时间及其频率。

图 8 显示了 5 KHz（黄色）的压缩机 PWM，10 KHz（青色）的风扇 PWM，以及 80 KHz 的两个 PFC PWM（红色和绿色），彼此之间有 180 度的相移。

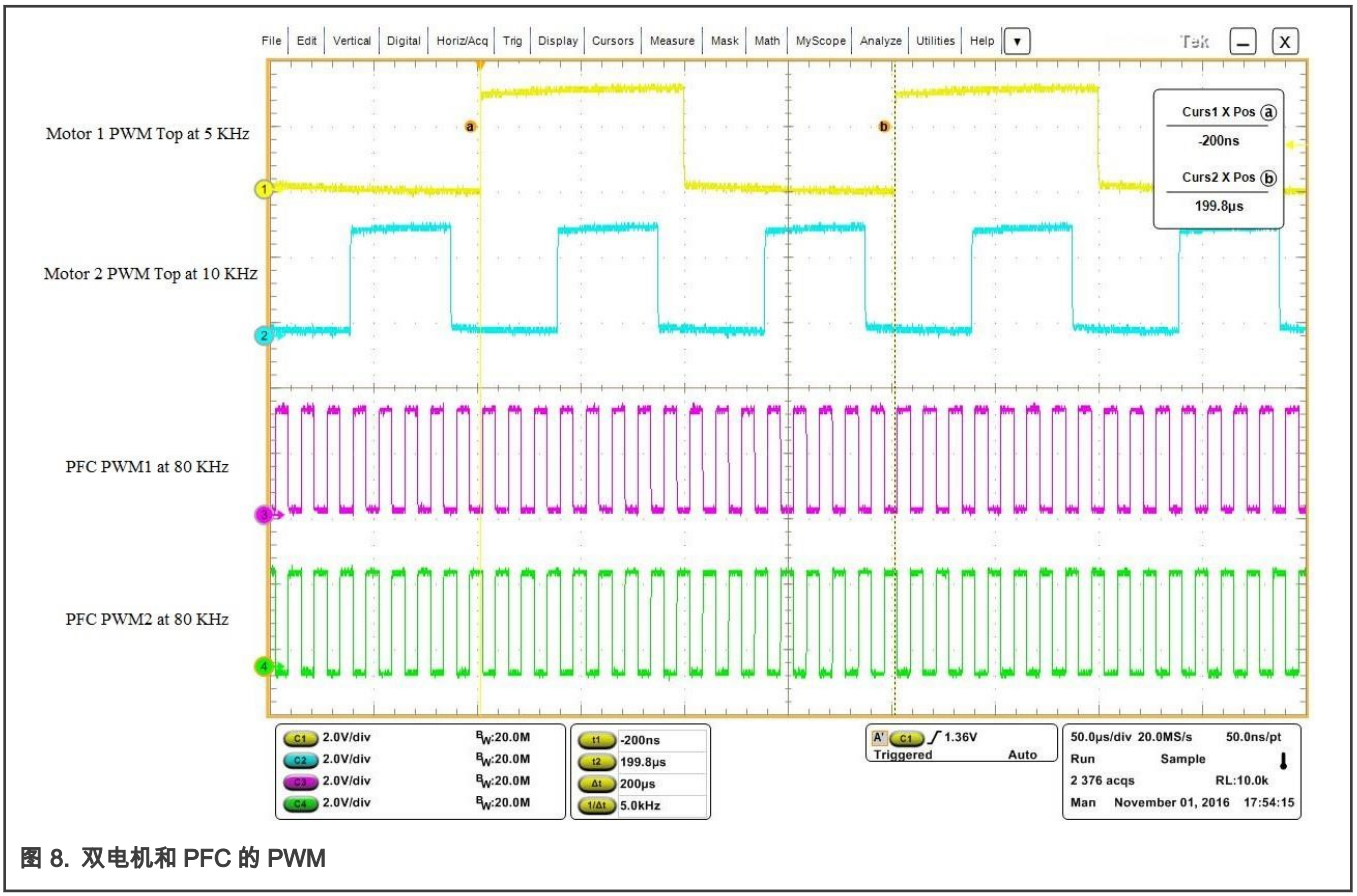

图 8. 双电机和 PFC 的 PWM

图 9 显示了系统时序的启动与预期相同（见系统时序启动）。

图 9. 系统启动时双电机和 PFC 的 PWM

图 10，图 11 ，和 图 12 显示了 PMW 以及压缩机、风扇和 PFC 的控制计算。

注意

PFC 控制计算中断具有最高优先级，因此它会中断其他电机计算中断。压缩机（电动机 1）控制计算中断的优先级最低，因此图中所示的持续时间必须减去风扇和 PFC 计算的时间。

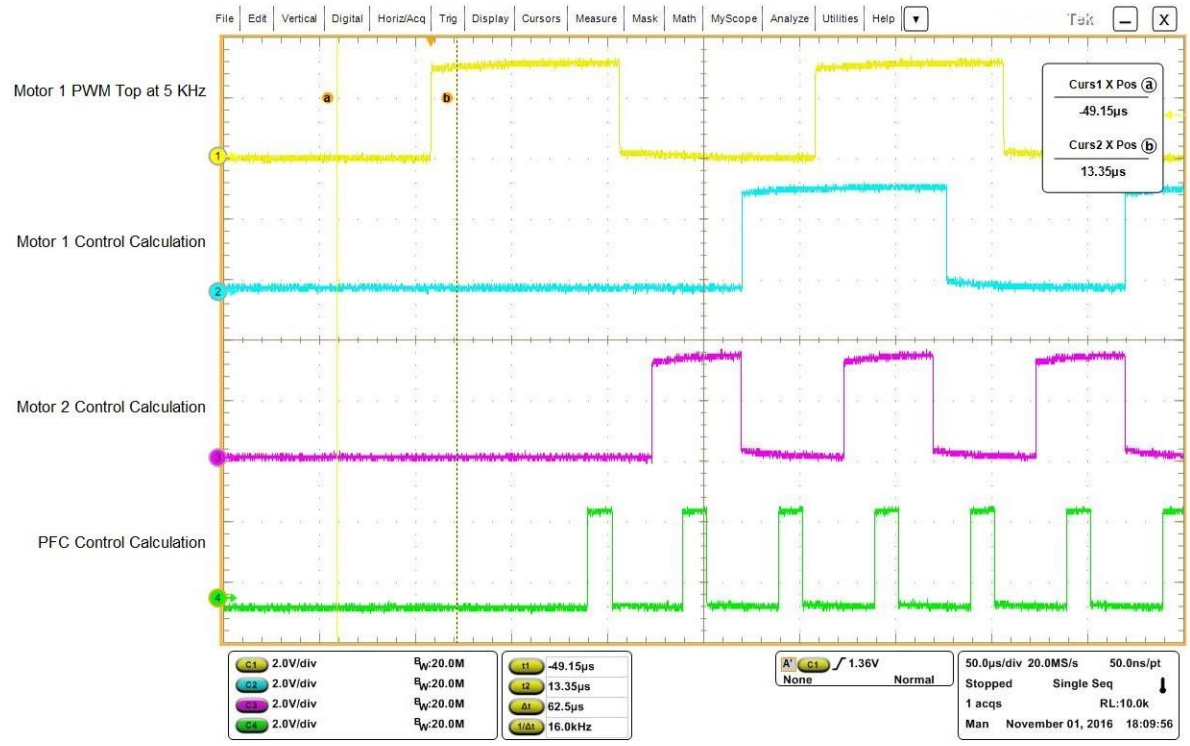

图 10. 压缩机 PWM 以及压缩机、风扇、PFC 的 ISR

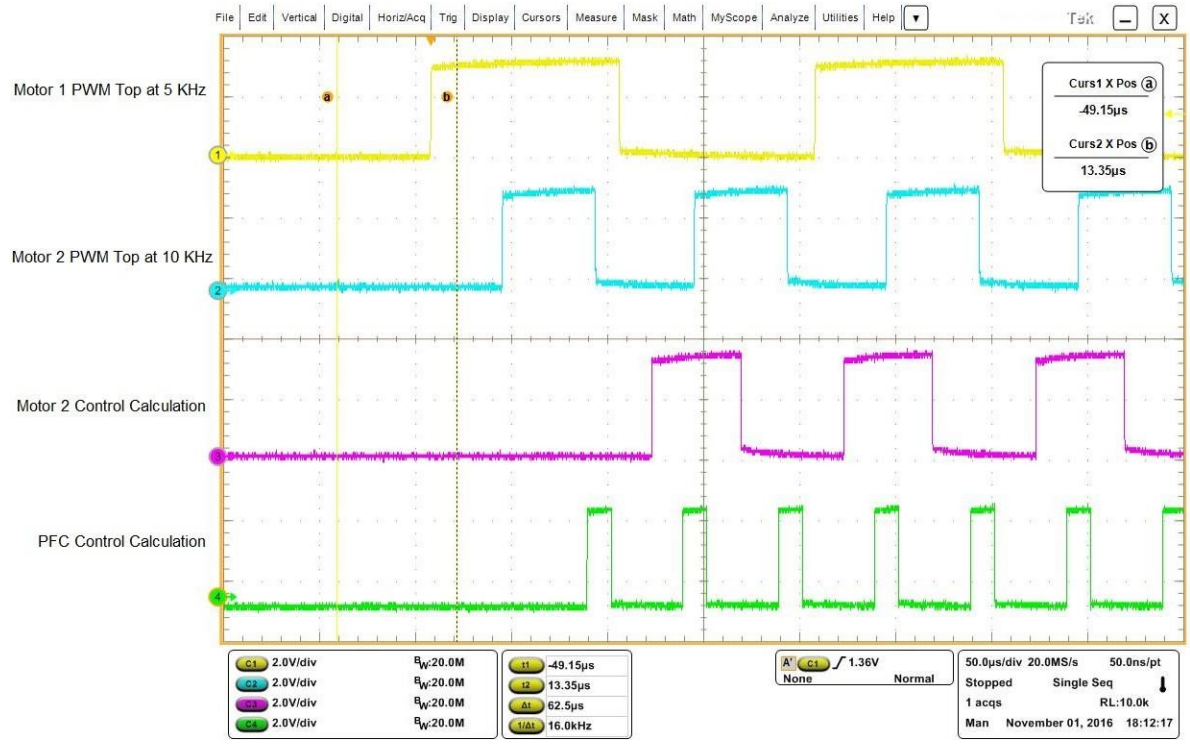

图 11. 压缩机 PWM、风扇 PWM 以及风扇和 PFC 的 ISR

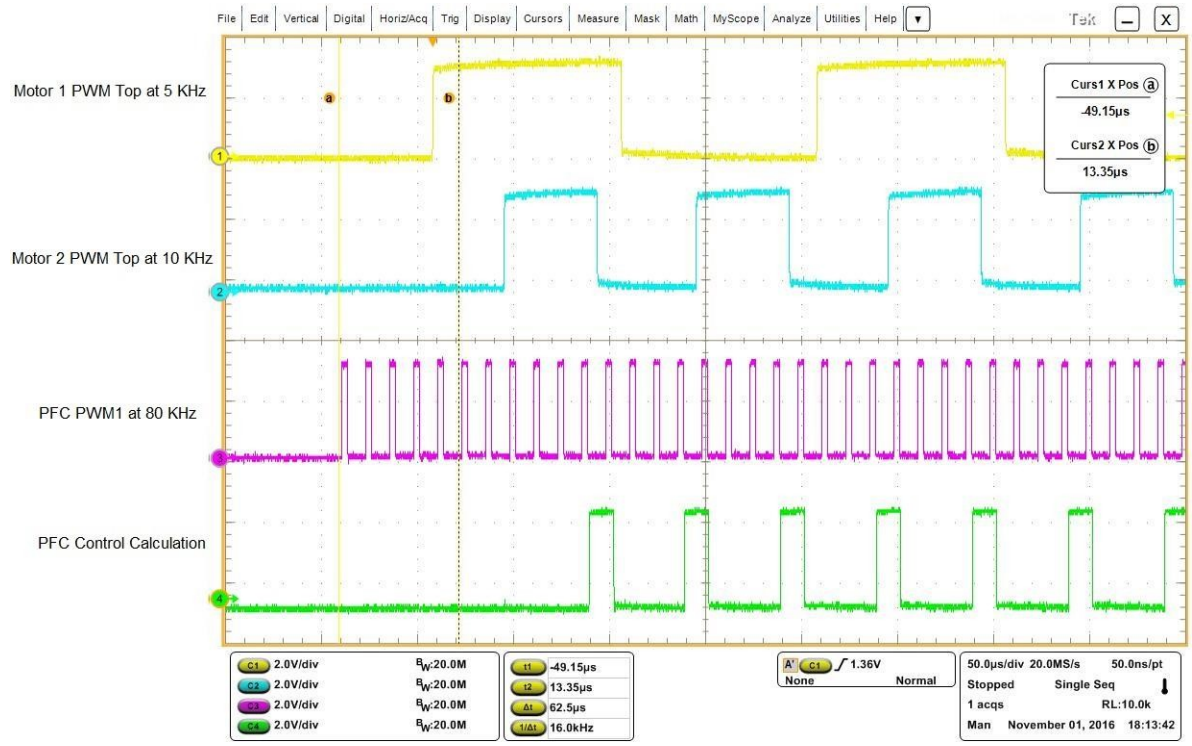

图 12. 压缩机风扇和 PFC PWM、PFC ISR

7 缩略语和缩写

表 1. 缩略语

术语	含义
ADC	Analogue-to-Digital converter（模数转换器）
DC	Direct Current（直流电）
FOC	Field-Oriented Control（磁场定向控制）
FTM	Flex Timer Module（灵活计时器模块）
GPIO	General Port Input Output（通用端口输入输出）
GTB	Global Time Base（全局时基）
ISR	Interrupt Service Routine（中断服务程序）
PCC	Peripheral Clock Controller（外围时钟控制器）
PDB	Programmable Delay Block（可编程延迟模块）
PFC	Power Factor Correction（功率因数校正）
PSMS	Permanent Magnet Synchronous Motor（永磁同步电动机）

下页继续...

表 1. 缩略语（续上页）

术语	含义
PWM	Pulse Width Modulation（脉冲宽度调制）
TRGMUX	Trigger MUX Control（触发 MUX 控制）

8 参考

以下参考文档可在 www.nxp.com 上获得:

1. *ADC Calibration on Kinetis E+ Microcontrollers*（文档 AN5314）
2. *Dual Sensorless PMSM Field-Oriented Control With Power Factor Correction on MC56F84789 DSC*（文档 DRM139）

9 修订记录

表 2. 修订记录

版本号	日期	说明
0	2016 年 11 月	初始版本

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 2016 年 11 月

Document identifier: AN5380

arm