Freescale Semiconductor
应用笔记

Document Number: AN4608
Rev 0, 10/2012

利用 MC56F84789 的 PWM 和 ADC 驱动双 PMSM 电机 FOC

作者: Jaroslav Musil

内容

1 简介

飞思卡尔数字信号控制器（DSC）具有强大的计算能力和灵活的外设，因此对应用的要求也较高。其中一个要求是用单个处理器驱动两台磁场定向控制（FOC）的永磁同步电机（PMSM）。

用单个处理器执行双 PMSM FOC 会增加应用的复杂性，主要表现在以下方面：两台电机的 PWM 模块同步；两个 PWM 模块的 ADC 同步，包括在正确的时间点触发 ADC；最后是计算两台电机快速和慢速控制环的时间。

本应用笔记旨在为以下问题提供指导：

- 如何设置和同步两个 PWM 模块
- 如何及在何处通过 PWM 模块产生 ADC 模块的触发信号
- 何时对两台电机的 FOC 算法进行快速环和慢速环计算

2 数字信号控制器（DSC）

MC56F84789 是适合双电机控制应用的 DSC。该控制器具有下列有利于应用的特性：

- 100 MHz 内核和外设时钟
- 两个 4 通道 PWM 模块，可提供多个触发信号
- 高速 12 位 ADC，可对两个信号进行同步采样
- 两个用于互连外设间信号的交叉单元

- 用于对外设之间信号进行逻辑合并的与或非模块
- 带优先级的中断控制器

该处理器还具有许多其他模块，但本应用笔记仅讨论上述模块。

图 1 显示了处理器信号如何连接到板上的电子器件。

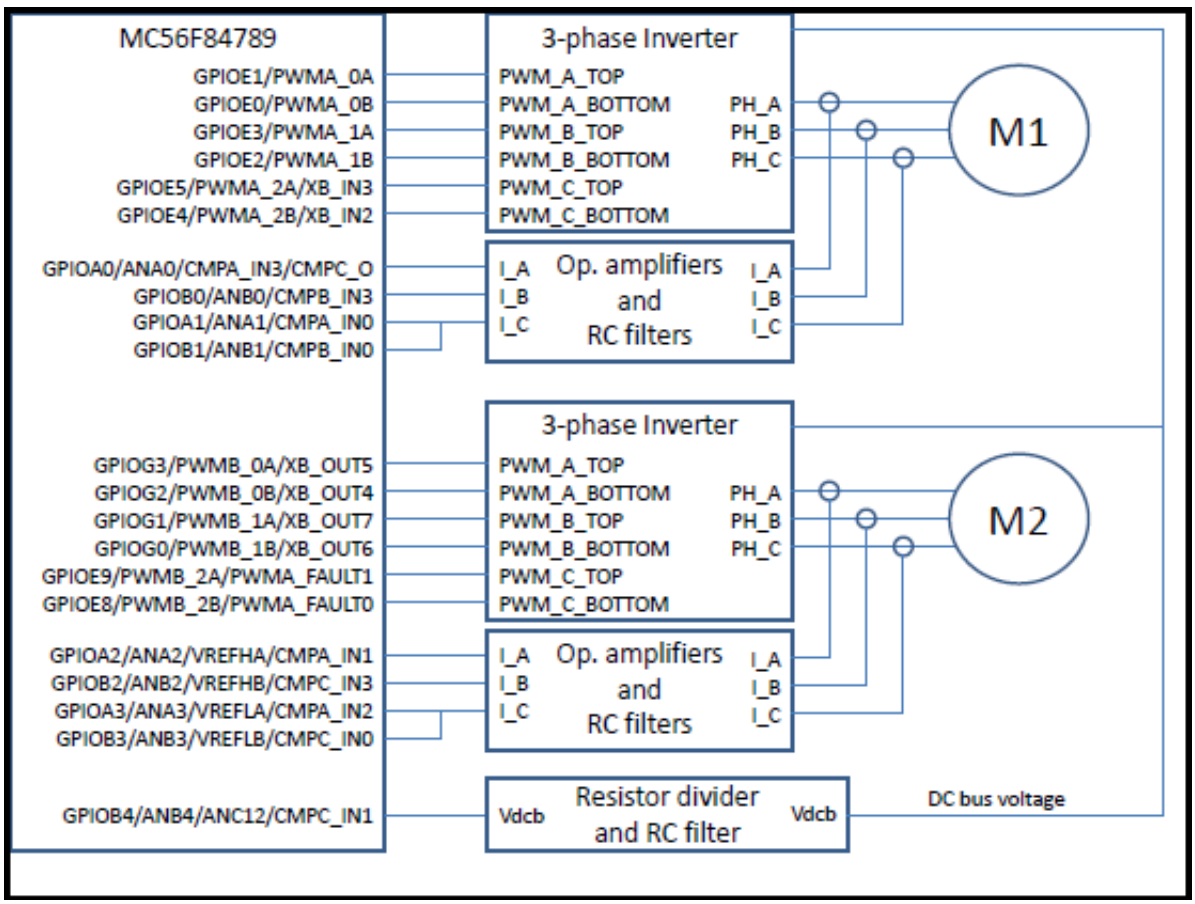

图 1. 处理器与电路板的连接

3 配置步骤

为了正确配置 DSC，请按以下步骤操作：

- PWM 配置：配置 PWM A 和 B 模块以产生驱动电机的信号。
- PWM A 和 B 同步：同步 PWM A 和 B 信号，使其在 90 度相移范围内。
- ADC 的 PWM 触发信号：设置触发 ADC 对信号采样的时间点。
- ADC 配置：配置 ADC 模块对所需信号进行采样。
- PWM 和 ADC 信号互连：配置 crossbar 和 AOI 模块，把 PWM 触发信号连接到 ADC 同步输入。
- 触发信号启动顺序：使触发信号按照与 ADC 通道顺序一致的顺序启动。
- 读取 ADC 采样：设置中断以读取采样值并调用算法。

后续章节将对上述所有步骤进行详细说明。

4 PWM 配置

在此应用的示例中，将用相同的 PWM 频率和相同的快速控制环计算频率来驱动两台电机。PWM 频率为 10 kHz，快速环计算相对于 PWM 频率的比为 1:1，因此也是 10 kHz。电机 1 将使用 PWM A 模块，即子模块 0–2；电机 2 将使用 PWM B 模块，即子模块 0–2；互补模式中采用非反相的输出逻辑。

第一步是按图 2 所示配置 PWM A 和 PWM B。

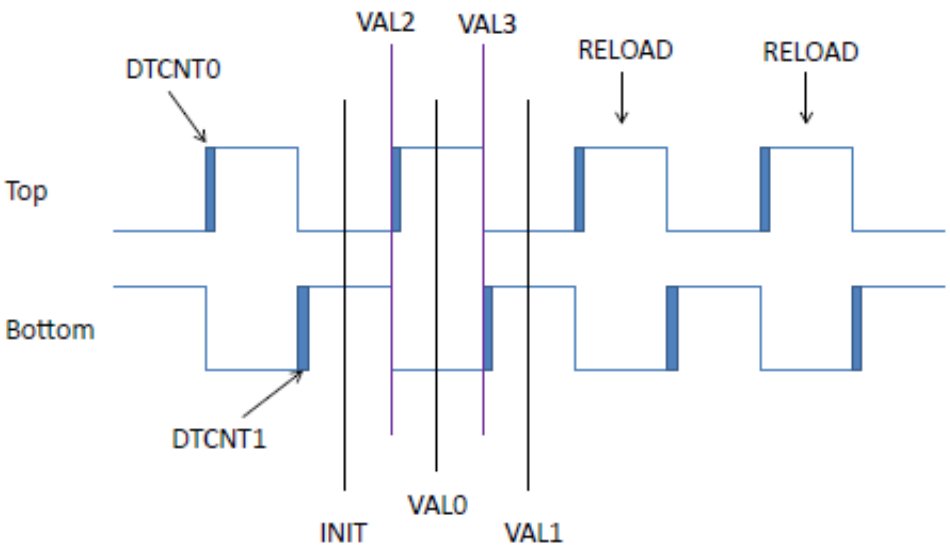

图 2. PWM 配置

PWM 时钟

为给两个 PWM 模块提供时钟，需要配置系统集成模块（SIM）中的外设时钟寄存器 3（SIM_PCE3）。

可使用以下语句来使能 PWM A 通道 0–2 的时钟：

```
SIM_PCE3 |= (SIM_PCE3_PWMACH0 | SIM_PCE3_PWMACH1 | SIM_PCE3_PWMACH2);
```

可使用以下语句来使能 PWM B 通道 0–2 的时钟：

```
SIM_PCE3 |= (SIM_PCE3_PWMBCH0 | SIM_PCE3_PWMBCH1 | SIM_PCE3_PWMBCH2);
```

PWM 控制寄存器

两台电机均使用每半个周期重裁一次。PWM 时钟频率为最大值，故预分频值为 1。

因此，PWM A 和 PWM B 模块的控制寄存器将用以下语句进行设置：

电机 1：

```
PWMA_SM0CTRL = PWMA_SM0CTRL_HALF;
PWMA_SM1CTRL = PWMA_SM1CTRL_HALF;
PWMA_SM2CTRL = PWMA_SM2CTRL_HALF;
```

电机 2：

```
PWMB_SM0CTRL = PWMB_SM0CTRL_HALF;
PWMB_SM1CTRL = PWMB_SM1CTRL_HALF;
PWMB_SM2CTRL = PWMB_SM2CTRL_HALF;
```

PWM 控制 **2** 寄存器

PWM 配置

为使子模块协同工作，需要正确设置 PWM 控制 2 寄存器。

1. 在调试和等待模式下使能 PWM，强制使通道 0 初始化
2. 对于通道 1 和 2，还要设置其他位：
   - 子模块 0 主同步的初始化位
   - 更新所用的主重载信号
   - 用于重载值寄存器的子模块 0 主重载位
   - 将用作时钟源的子模块 0 时钟

两个模块的配置寄存器均可利用以下语句进行配置。

电机 1:

```
PWMA_SM0CTRL2 = 0xC080;
PWMA_SM1CTRL2 = 0xC20E;
PWMA_SM2CTRL2 = 0xC20E;
```

电机 2:

```
PWMB_SM0CTRL2 = 0xC080;
PWMB_SM1CTRL2 = 0xC20E;
PWMB_SM2CTRL2 = 0xC20E;
```

PWM 模数寄存器设置

产生 10 kHz 的 PWM 模数源自 100 MHz 时钟。因此，模数寄存器为 100 MHz / 10 kHz = 10,000。PWM 模块的 INIT 寄存器值是计数器起始值，VAL1 寄存器值是计数器重新初始化值。因此，INIT 值设置为半模数寄存器值的负值，VAL1 值设置为半模数寄存器值–1 的正值。重载在半周期时发生，即在这两个值的中间值处发生。简而言之：

INIT =–5000 (0xEC78)，VAL1 = 4999 (0x1387)，VAL0 = 0。可使用以下语句进行配置。

电机 1:

```
PWMA_SM0INIT = 0xEC78;
PWMA_SM1INIT = 0xEC78;
PWMA_SM2INIT = 0xEC78;

PWMA_SM0VAL1 = 0x1387;
PWMA_SM1VAL1 = 0x1387;
PWMA_SM2VAL1 = 0x1387;

PWMA_SM0VAL0 = 0x0000;
PWMA_SM1VAL0 = 0x0000;
PWMA_SM2VAL0 = 0x0000;
```

电机 2:

```
PWMB_SM0INIT = 0xEC78;
PWMB_SM1INIT = 0xEC78;
PWMB_SM2INIT = 0xEC78;

PWMB_SM0VAL1 = 0x1387;
PWMB_SM1VAL1 = 0x1387;
PWMB_SM2VAL1 = 0x1387;

PWMB_SM0VAL0 = 0x0000;
PWMB_SM1VAL0 = 0x0000;
PWMB_SM2VAL0 = 0x0000;
```

50%占空比初始化

为使模块在 50%占空比处初始化，需要设置 VAL2 和 VAL3 寄存器。由于采用互补模式，PWM 边沿的产生不使用 VAL4 和 VAL5 寄存器。VAL2 等于半模数寄存器值除以 2 所得值的负值，VAL3 等于半模数寄存器值除以 2 所得值的正值。

因此，VAL2 =–2500 (0xF63CU)，VAL3 = 2500 (0x09C3)。可使用以下语句进行配置。

电机 1:

```
PWMA_SM0VAL2 = 0xF63C;
PWMA_SM1VAL2 = 0xF63C;
PWMA_SM2VAL2 = 0xF63C;

PWMA_SM0VAL3 = 0x09C3;
PWMA_SM1VAL3 = 0x09C3;
PWMA_SM2VAL3 = 0x09C3;
```

电机 2:

```
PWMB_SM0VAL2 = 0xF63C;
PWMB_SM1VAL2 = 0xF63C;
PWMB_SM2VAL2 = 0xF63C;

PWMB_SM0VAL3 = 0x09C3;
PWMB_SM1VAL3 = 0x09C3;
PWMB_SM2VAL3 = 0x09C3;
```

2 µs 死区时间

要设置死区时间，需要配置 DTCNT0（顶部开关）和 DTCNT1（底部开关）寄存器值。该值从模块时钟获得，如果时间为 2 µs，该值为 100 MHz x 2 µs = 200 (0x00C8)。以下代码可配置死区时间:

电机 1:

```
PWMA_SM0DTCNT0 = 0x00C8;
PWMA_SM1DTCNT0 = 0x00C8;
PWMA_SM2DTCNT0 = 0x00C8;

PWMA_SM0DTCNT1 = 0x00C8;
PWMA_SM1DTCNT1 = 0x00C8;
PWMA_SM2DTCNT1 = 0x00C8;
```

电机 2:

```
PWMB_SM0DTCNT0 = 0x00C8;
PWMB_SM1DTCNT0 = 0x00C8;
PWMB_SM2DTCNT0 = 0x00C8;

PWMB_SM0DTCNT1 = 0x00C8;
PWMB_SM1DTCNT1 = 0x00C8;
PWMB_SM2DTCNT1 = 0x00C8;
```

禁用故障

本例不使用故障逻辑，因此需要利用以下语句禁用故障映射寄存器。

电机 1:

```
PWMA_SM0DISMAP0 = 0;
PWMA_SM1DISMAP0 = 0;
PWMA_SM2DISMAP0 = 0;

PWMA_SM0DISMAP1 = 0;
PWMA_SM1DISMAP1 = 0;
PWMA_SM2DISMAP1 = 0;
```

电机 2:

```
PWMB_SM0DISMAP0 = 0;
PWMB_SM1DISMAP0 = 0;
PWMB_SM2DISMAP0 = 0;

PWMB_SM0DISMAP1 = 0;
PWMB_SM1DISMAP1 = 0;
PWMB_SM2DISMAP1 = 0;
```

LDOK 位

运行 PWM 之前的最后一步是利用以下代码清除和设置 MCTRL[LDOK]。

电机 1:

```
PWMA_MCTRL |= PWMA_MCTRL_CLDOK_0 | PWMA_MCTRL_CLDOK_1 | PWMA_MCTRL_CLDOK_2;
PWMA_MCTRL |= PWMA_MCTRL_LDOK_0 | PWMA_MCTRL_LDOK_1 | PWMA_MCTRL_LDOK_2;
```

电机 2:

```
PWMB_MCTRL |= PWMB_MCTRL_CLDOK_0 | PWMB_MCTRL_CLDOK_1 | PWMB_MCTRL_CLDOK_2;
PWMB_MCTRL |= PWMB_MCTRL_LDOK_0 | PWMB_MCTRL_LDOK_1 | PWMB_MCTRL_LDOK_2;
```

将 PWM A 和 PWM B 配置为可产生图 1 所示的信号。但为了从引脚输出信号，还必须正确配置 GPIO 引脚。某些 GPIO E 和 GPIO G 引脚必须设置为外设。若有多个外设选项，必须选择 PWM 选项。还必须使能 GPIO E 和 GPIO G 时钟。代码如下:

```
/* Enable GPIOE clock */
SIM_PCE0 |= SIM_PCE0_GPIOE;

/* PWMA PWMB */
GPIOE_PER |= (GPIOE_PER_PE_0 | GPIOE_PER_PE_1 | GPIOE_PER_PE_2 | GPIOE_PER_PE_3 |
GPIOE_PER_PE_4 | GPIOE_PER_PE_5 | GPIOE_PER_PE_8 | GPIOE_PER_PE_9);
SIM_GPSEL &= ~(SIM_GPSEL_E4 | SIM_GPSEL_E5);
SIM_GPSEH &= ~(SIM_GPSEH_E8 | SIM_GPSEH_E9);

/* Enable GPIOG clock */
SIM_PCE0 |= SIM_PCE0_GPIOG;

/* PWM B */
GPIOG_PER |= (GPIOG_PER_PE_0 | GPIOG_PER_PE_1 | GPIOG_PER_PE_2 | GPIOG_PER_PE_3);
SIM_GPSGL &= ~(SIM_GPSGL_G0 | SIM_GPSGL_G1 | SIM_GPSGL_G2 | SIM_GPSGL_G3);
```

最后只需将运行命令发送至模块以开始产生信号，并在引脚上输出 PWM 信号。此时不得启动 PWM 模块，因为需要同步两个 PWM。

5 PWM A 和 PWM B 同步

PWM A 和 PWM B 已完成配置，但尚未启动。因此，下一步便是以某种方式同步 PWM 模块。

要在每个 PWM 周期中执行快速环算法，必须让 PWM 信号相互移位，因为不可能同时执行两台电机的算法。因此，必须使 PWM 信号移位。为了有效分配来自直流总线电容的能量，建议避免同时切换，也就是要移位 180 度。MC56F84789 处理器具有强大的计算能力，能够快速高效地执行 FOC 算法，信号将移位 90 度以获得最优结果。

两个 PWM 模块所需的信号如图 3 所示。

PWM 90 度移位原理

如图 3 所示，PWM B INIT 值相对于 PWM A INIT 值滞后 90 度。此移位等于模数除以 4。当施加 PWM RUN 信号时，PWM 模块从 INIT 值递增计数，这对该应用是有利的。

同步 PWM 模块的过程包括:

- 启动 PWM A 模块
- 在模数/4 时刻从 PWM A 模块产生一个中断
- 依据所产生的 PWM A 模块中断，启动 PWM B 模块
- 禁用此 PWM A 模块事件的中断

模数**/4** 事件配置

启动 PWM A 模块之前，必须对模数/4 事件中断进行编程。为产生此类事件，将使用未使用的 VAL5 值。无论哪个子模块产生此事件都是可行的。本例使用子模块 1 VAL5 值。

在代码中，子模块 1 VAL5 使用模数/4 进行编程，为–2500 (0xF63C)；此子模块的比较中断须使能，并且必须对中断控制器进行编程以使能此中断。

代码如下：

```
/* Sync for the other motor's PWM */
PWMA_SM1VAL5 = 0xF63C;
/* Compare interrupt of Value 5, used at init to sync PWM A and PWM B */
PWMA_SM1INTEN = PWMA_SM1INTEN_CMPIE_5;
/* Interrupt for the SM1 CMP Level 2 */
INTC_IPR9 |= INTC_IPR9_PWMA_CMP1;
```

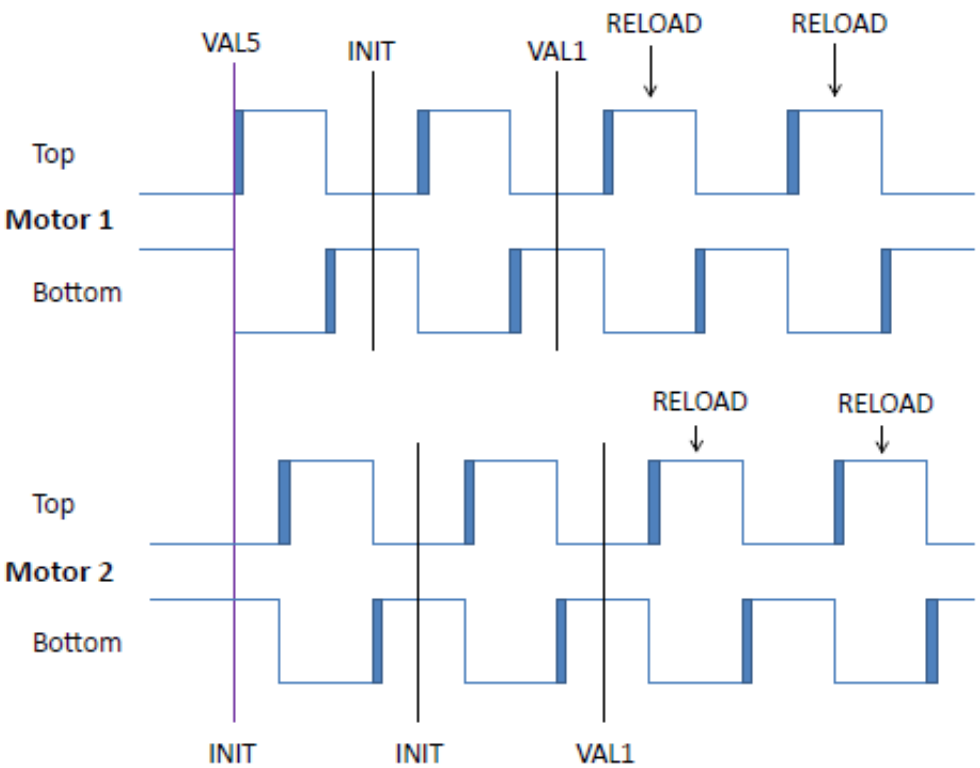

图 3. PWM A 和 PWM B 同步

VAL5 比较中断服务例程

PWM 子模块 1 VAL5 比较中断已配置完毕。现在必须创建中断服务例程（ISR）以启动 PWM B 模块。

ISR 的名称为 IsrPWMSync。此例程的原型必须在代码的原型部分创建：

```
void IsrPWMSync(void);
```

此函数的名称必须复制到向量表中；如果使用默认 CodeWarrior 10.2 项目模板，则它位于文件 MC56F847xx_vector.asm 中（Project_Settings\Startup_Code 目录下）。因此，在地址 0xAA，PWM A 子模块 1 的 85 号中断将包含以下语句：

```
JSR  >FIsrPWMSync
```

函数体本身包含以下操作：

- if 条件，如果源是 VAL5 比较
- 对 PWM B 模块应用 RUN 命令
- 禁用 VAL5 比较中断
- 清除 VAL5 比较标志

包含所有这些操作的代码如下：

```
#pragma interrupt alignsp
void IsrPWMSync(void)
{
       if ((PWMA_SM1STS & PWMA_SM1STS_CMPF_5) > 0 )
       {
             /* Starts PWM B */
             PWMB_MCTRL |= PWMB_MCTRL_RUN_0 | PWMB_MCTRL_RUN_1 | PWMB_MCTRL_RUN_2;

             /* Disable PWM SM1 CMP interrupt from 5 */
             PWMA_SM1INTEN &= ~PWMA_SM1INTEN_CMPIE_5;

             /* Clears compare flag */
             PWMA_SM1STS |= PWMA_SM1STS_CMPF_5;
       }
}
```

PWM A 启动

PWM A 和 B 通道已配置完毕，同步事件已设置，中断服务例程已写好。现在只需对 PWM A 应用 RUN 命令并使能引脚输出 PWM A 和 PWM B 信号。启动 PWM A 后，它将产生已编程的子模块 1 VAL5 比较中断，PWM B 随即启动，此中断随后被禁止。

启动 PWM A 的代码如下（PWM A 和 PWM B 模块配置完毕且设置比较中断之后，需要调用此命令）：

```
PWMA_MCTRL |= PWMA_MCTRL_RUN_0 | PWMA_MCTRL_RUN_1 | PWMA_MCTRL_RUN_2;
```

使能引脚输出 PWM A 和 PWM B 信号的命令如下：

```
PWMA_OUTEN |= (PWMA_OUTEN_PWMA_EN_0 | PWMA_OUTEN_PWMB_EN_0 | PWMA_OUTEN_PWMA_EN_1 |
PWMA_OUTEN_PWMB_EN_1 | PWMA_OUTEN_PWMA_EN_2 | PWMA_OUTEN_PWMB_EN_2);

PWMB_OUTEN |= (PWMB_OUTEN_PWMA_EN_3 | PWMB_OUTEN_PWMB_EN_3 | PWMB_OUTEN_PWMA_EN_1 |
PWMB_OUTEN_PWMB_EN_1 | PWMB_OUTEN_PWMA_EN_2 | PWMB_OUTEN_PWMB_EN_2);
```

图 3 显示了引脚上的信号，示波器上也可观察到这些信号。

6 ADC 的 PWM 触发信号

PMSM 的 FOC 要求测量控制算法所用的模拟信号。此类控制需要的模拟量为电机相位电流和直流总线电压。

通常，电流在底部 MOSFET（或 IGBT）下方连接的分流电阻处测量。请参见图 4 。在这种结构中，电机电流只能在特定底部 MOSFET 接通时得到。因此，ADC 必须与 PWM 信号同步（参见图 5 ）。可在底部 MOSFET 接通时立即测量电流；若在底部 MOSFET 断开时立即测量，只能得到 ADC 通道的偏移量。此时间点将用于校准 ADC 通道偏移量，因为预期值是已知值。

直流总线电压对测量时间点没有这种要求。

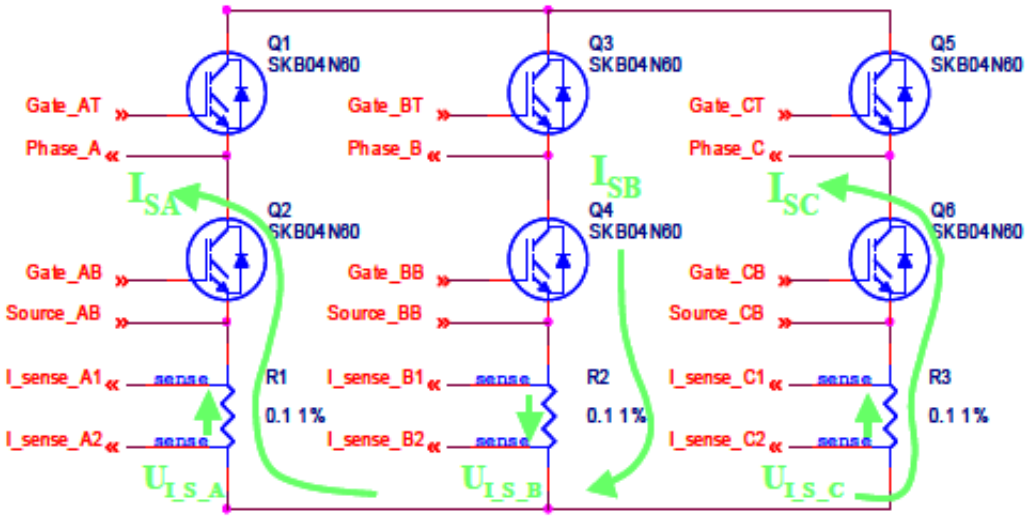

图 4. 三相逆变器原理图

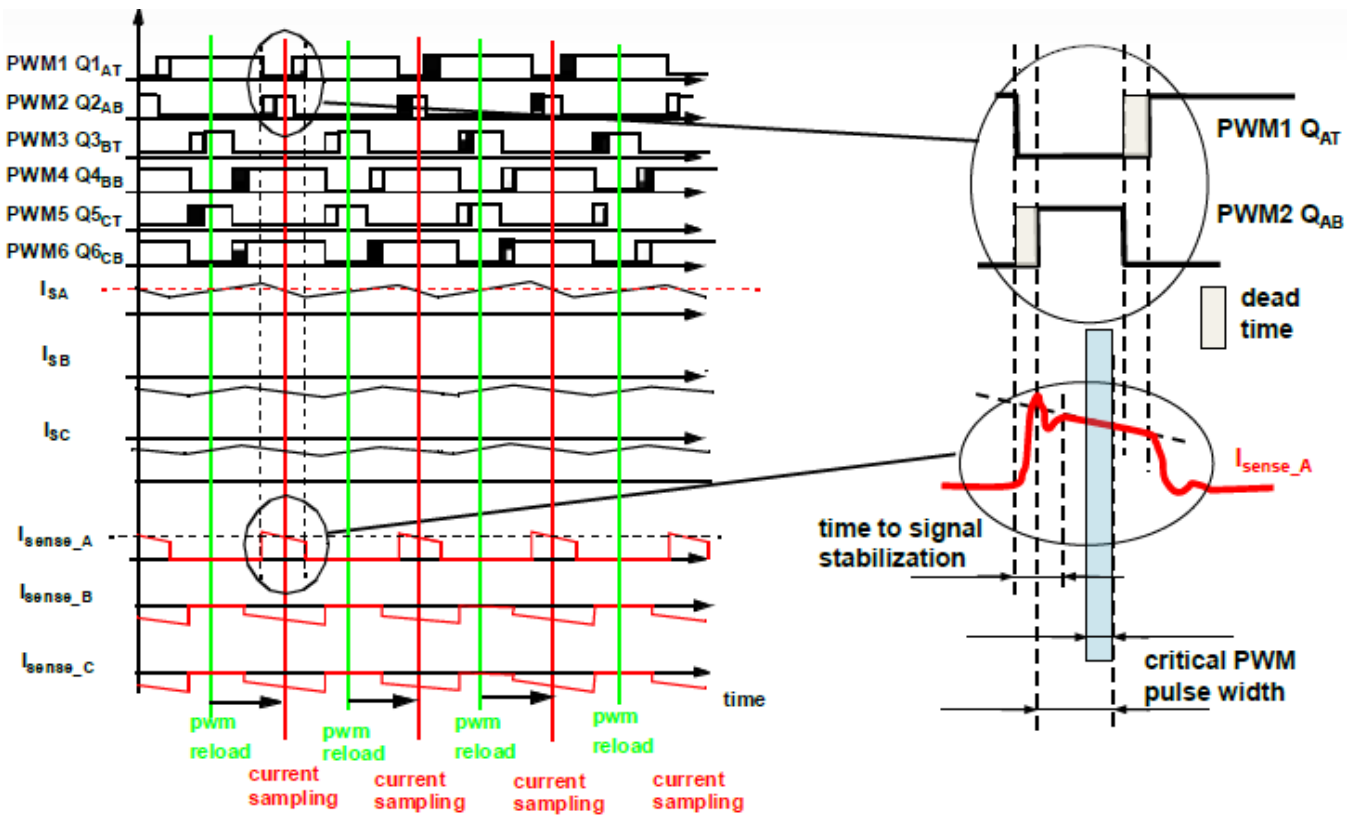

图 5. 测量电机电流的最佳时间

总而言之，需要测量以下量：

- 电机 1 的 2 个电流通道的偏移量（第 3 个电流通过计算获得）
- 电机 1 的 2 个电流
- 电机 2 的 2 个电流通道的偏移量（第 3 个电流通过计算获得）
- 电机 2 的 2 个电流
- 直流总线电压

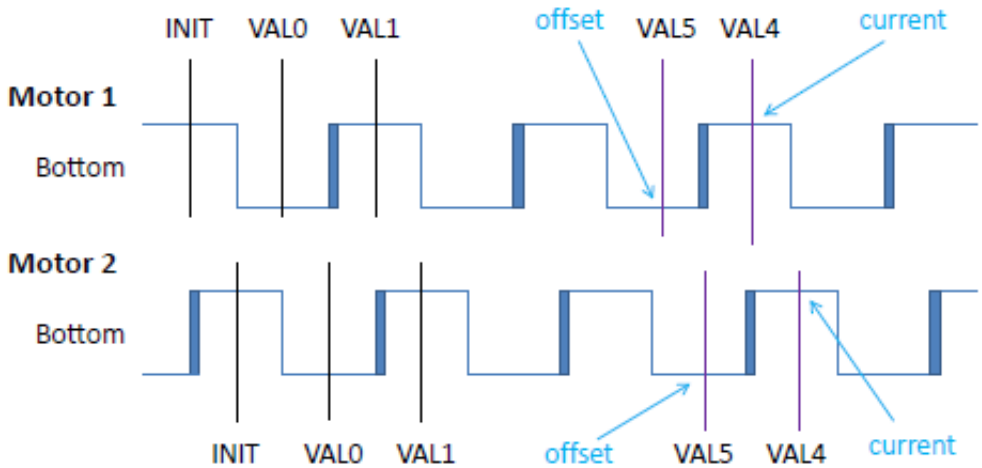

图 6. 电流和偏移量测量点

PWM 触发信号

图 6 指出了电流及其偏移量的测量点。可以看到，电流测量点稍微落后于 INIT 值，偏移量测量点稍微落后于 VAL0 值。

PWM 模块采用互补切换模式，因此可将 VAL4 和 VAL5 值用作 ADC 的触发信号。本例使用的死区时间为 2 µs，假设还有一定的硬件延迟，最终需要应用的延迟时间将是 3.25 µs。

因此，VAL4 值将为:

INIT + 3.25 µs / 100 MHz = –4675 (0xEDBD)。

VAL5 值将为:

VAL0 + 3.25 µs / 100 MHz = 325 (0x0145)。

使用子模块 0 来产生这些触发信号。这些 PWM 值的编程代码语句如下:

电机 1:

```
PWMA_SM0VAL4 = 0xEDBD;  /* Current measurement value */
PWMA_SM0VAL5 = 0x0145;  /* Offset measurement value */
```

电机 2:

```
PWMB_SM0VAL4 = 0xEDBD;  /* Current measurement value */
PWMB_SM0VAL5 = 0x0145;  /* Offset measurement value */
```

此时不得使能触发信号。触发信号必须按照触发信号启动顺序 所述的特定顺序加以使能。

7 ADC 配置

为了对模拟量进行采样，将在同步模式下使用快速 12 位 ADC A 和 ADC B。同步模式是 ADC 的一个出色特性，正是由于这个特性，我们才能同时获取两个电流值。以下步骤说明如何正确设置 ADC。

ADC 时钟

为给 ADC 提供时钟，需要配置系统集成模块（SIM）中的外设时钟寄存器 2（SIM_PCE2）。

使能 ADC 时钟的语句如下:

```
SIM_PCE2 |= SIM_PCE2_CYCADC;
```

控制寄存器 **1**

此寄存器用于配置 ADC 行为。要配置此寄存器，请执行以下操作:

- 设置硬件同步（这对用 PWM 模块信号启动 ADC 必不可少)。
- 使能扫描结束中断。
- 设置 ADC 并行扫描模式。
- 停止模式未激活。
- 所有通道均为单端。
- 已关闭其他中断和 DMA 通道。

语句如下:

```
ADC12_CTRL1 = 0x1805;
```

控制寄存器 **2**

此寄存器用于配置 ADC A 和 B 转换器同步工作，时钟将设为 20 MHz 以使 ADC 工作。语句如下:

```
ADC12_CTRL2 = ADC12_CTRL2_DIV0_2 | ADC12_CTRL2_SIMULT;
```

电源控制寄存器

此寄存器用于控制 ADC 的电源选项。要配置此寄存器，请执行以下操作:

- 必须给 ADC 转换器上电。
- “上电延迟”选项保持默认值，即 26 个时钟周期。

代码如下:

```
ADC12_PWR = 0x01A0;
```

电源控制寄存器 **2**

此寄存器用于配置 ADC B 转换器的时钟和两个转换器的速度。B 转换器的时钟为 20 MHz，两个转换器的速度设为 20 MHz。语句如下:

```
ADC12_PWR2= ADC12_PWR2_SPEEDA | ADC12_PWR2_SPEEDB | ADC12_PWR2_DIV1_2;
```

通道列表寄存器

现在需要配置通道列表寄存器。两个 ADC 转换器均有 8 个通道，也就是一次能够对 8 个通道进行采样，并且可保存 ADC A 和 ADC B 的结果。如果这两个转换器以同步模式运行，则 A 转换器仅对 ANAx 通道进行采样，B 转换器仅对 ANBx 通道进行采样。

本例在 ANA0 和 ANB0 上对电机 1 电流进行采样，在 ANA2 和 ANB2 上对电机 2 电流进行采样。直流总线电压在 ANB4 通道上采样。

图 7 显示了通道配置。为将通道与特定采样通道相关联，使用了四个通道列表寄存器。每个寄存器配置四个样本。设置通道的代码如下:

```
ADC12_CLIST1 = 0x0200;
ADC12_CLIST2 = 0x0002;
ADC12_CLIST3 = 0x8AC8;
ADC12_CLIST4 = 0x888A;
```

采样禁用寄存器

本例仅使用各转换器的 5 个采样通道。其余采样通道必须禁用。完成最后一个使能的采样通道后，ADC 会产生扫描结束中断信号。要根据图 7 禁用未使用的通道，可使用如下代码设置寄存器:

```
ADC12_SDIS = 0xE0E0;
```

扫描控制寄存器

另一个非常有用的特性是扫描控制寄存器。借助扫描控制寄存器，能够对多个事件同步采样，完成最后一个使能的采样后，会产生扫描结束中断信号。此流程可参见图 7 的 Sync 行。

1. 出现第一个同步信号（来自 PWM 的触发信号）时，ADC 对电机 1 两个相位的电流偏移量进行采样；由于下一个同步信号在样本 2 处设置，样本 1 和 9 也会被转换，ADC 随后将停止。因此，直流总线电压在通道 9 上。
2. 下一个同步信号到达时，ADC 对电机 2 的偏移量进行采样。
3. 对于下一个同步信号，ADC 对电机 1 的电流进行采样，然后停止。
4. 对于再下一个同步信号，ADC 对电机 2 的电流进行采样，然后停止并产生扫描结束中断信号。对于新的同步信号，上述过程重新开始。

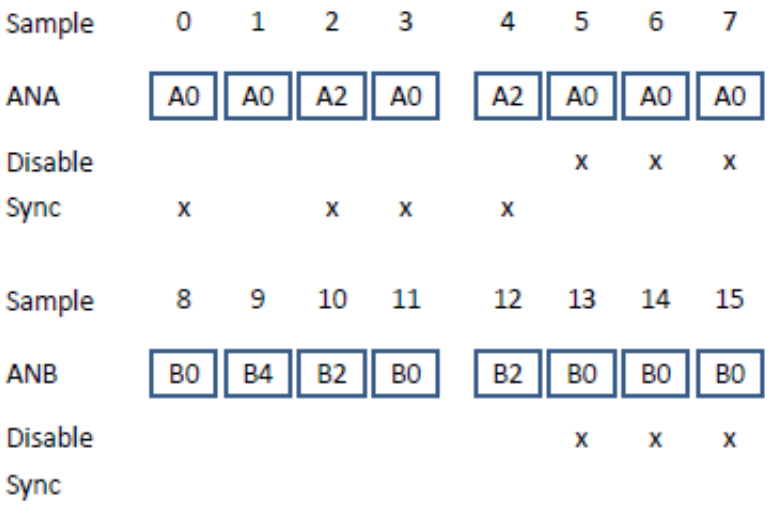

图 7. ADC 样本和通道

要配置此同步特性，应使用扫描控制寄存器。同步信号与通道 0、2、3（位 0、2、3）和通道 4（位 8）相关联。示例代码如下：

```
ADC12_SCTRL = 0x010D;
```

模拟（**AN**）引脚配置

要从引脚读出模拟值，必须利用 ADC 选项将 GPIO A 和 GPIO B 引脚 0–7 配置为外设。GPIO 模块的时钟必须使能。代码如下：

```
/* Enable GPIOA clock */
SIM_PCE0 |= SIM_PCE0_GPIOA;

/* ADCA */
GPIOA_PER |= (GPIOA_PER_PE_0 | GPIOA_PER_PE_1 | GPIOA_PER_PE_2 | GPIOA_PER_PE_3 |
GPIOA_PER_PE_4 | GPIOA_PER_PE_5 | GPIOA_PER_PE_6 | GPIOA_PER_PE_7);
SIM_GPSAL &= ~(SIM_GPSAL_A0);

/* Enable GPIOB clock */
SIM_PCE0 |= SIM_PCE0_GPIOB;

/* ADCB */
GPIOB_PER |= (GPIOB_PER_PE_0 | GPIOB_PER_PE_1 | GPIOB_PER_PE_2 | GPIOB_PER_PE_3 |
GPIOB_PER_PE_4 | GPIOB_PER_PE_5 | GPIOB_PER_PE_6 | GPIOB_PER_PE_7);
```

8 PWM 和 ADC 信号互连

到目前为止，已完成了下述步骤：

- 已配置并同步 PWM 模块。
- 已设置产生触发信号的 PWM 位置。
- 已对 ADC 模块和通道进行配置。
- 已正确设置 ADC 采样的同步位。

下一步是要将 PWM A 和 PWM B 模块的触发信号正确连接到 ADC 同步信号输入。为此，应使用外设 crossbar A（XBAR A）。

XBAR A 能够将一个外设的一路输出连接到一路输入，但此系统有来自两个外设的四个触发信号和一路 ADC 输入。在这种情况下，必须对这些触发信号进行逻辑“或”运算，并将执行“或”运算后的信号送至 ADC 输入。能够对信号进行逻辑“或”运算的模块是与/或/非（AOI）模块。为了在 AOI 输入端获取 PWM 触发信号，还要使用一个模块，即外设 crossbar B（XBAR B）。请参见图 8 。

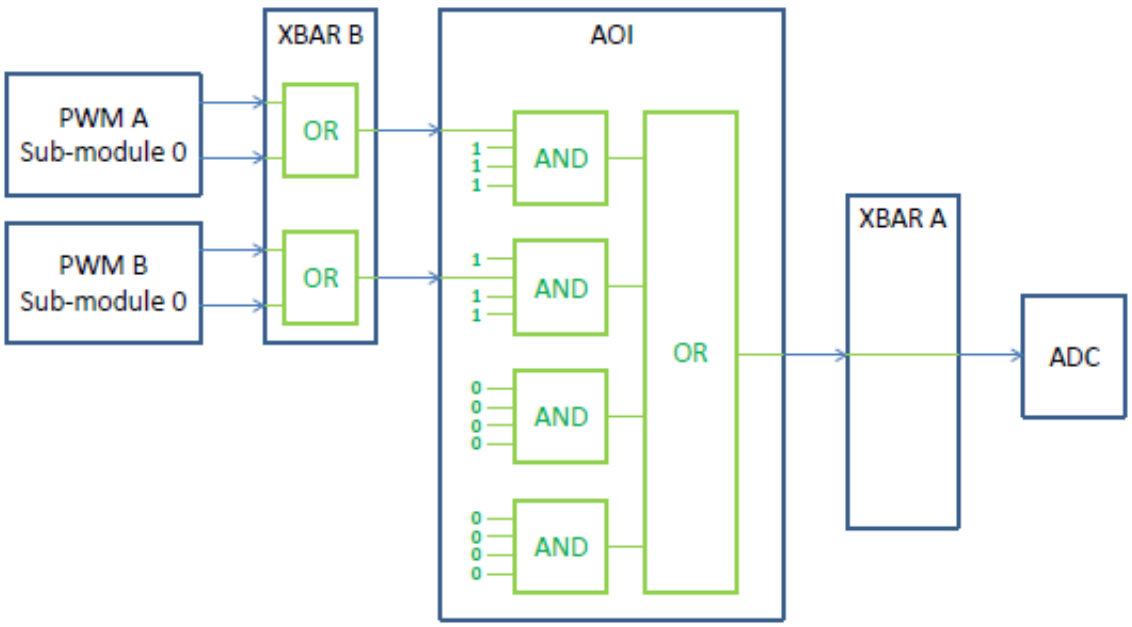

图 8. PWM 和 ADC 互连

XBAR B 配置

XBAR B 用于将外设输出连接到 AOI 模块。两个 PWM 模块均使用 VAL4 和 VAL5 寄存器来产生触发信号。这些触发信号分为两组：PWM_OUT_TRIG0 和 PWM_OUT_TRIG1。第一组包括 VAL0、VAL2 和 VAL4 触发信号，第二组包括 VAL1、VAL3 和 VAL5 触发信号。这意味着两个 PWM 模块均会发出两个输出信号。

XBAR B 的前四路输入连接到 AOI 的乘积项 0，因此，PWM 信号按照如下方式连接到 XBAR B：

- PWMA 子模块 0 触发信号（PWMA0_TRG0 | PWMA0_TRG1）连接到 AOI 乘积项 0 输入 0
- PWMB 子模块 0 触发信号（PWMB0_TRG0 | PWMB0_TRG1）连接到 AOI 乘积项 0 输入 1

因此，XBAR_IN8 和 XBAR_IN22 输入将被分配给前两路 XBAR B 输出。代码如下：

```
XBARB_SEL0 = 8 | (22 << 8);
```

AOI 配置

顾名思义，此模块能够执行与/或/非逻辑运算。此应用受益于“或”和“与”运算，未使用“非”运算。与/或运算符的工作方式请参见图 8 。

仅使用两个输入信号，这些信号连接到“与”门的一路输入。两个“与”门的其余三路输入强制设为逻辑 1，两个“与”门的未使用输入强制设为逻辑 0。这种设置只会将两个 PWM 信号传输至“或”门输入。“或”门输出将从 AOI 发出。

配置代码如下：

```
AOI_BFCRT010 = AOI_BFCRT010_PT0_AC_0 | AOI_BFCRT010_PT0_BC | AOI_BFCRT010_PT0_CC |
AOI_BFCRT010_PT0_DC | AOI_BFCRT010_PT1_AC | AOI_BFCRT010_PT1_BC_0 | AOI_BFCRT010_PT1_CC |
AOI_BFCRT010_PT1_DC;

AOI_BFCRT230 = 0;
```

XBAR A 配置

最后一个信号路径是将 AOI 输出连接到 ADC。此配置非常简单，只需将 XBAR A AND_OR_INVERT_0 的输入（AOI 输出 0）连接到 XBAR A 输出 XBAR_OUT12（ADC A 触发信号）。示例代码如下：

```
XBARA_SEL6 = 46;
```

9 触发信号启动顺序

所有外设都已正确配置，能够在要求的时间点上产生触发信号。信号经过逻辑处理后，送至 ADC 同步脉冲输入端。最后一步是启动触发信号。

ADC 配置为按如下顺序执行采样操作：

- 电机 1 的偏移量
- 电机 2 的偏移量
- 电机 1 的电流
- 电机 2 的电流

从图 6 可推断出，PWM A VAL5 触发信号必须第一个出现，PWM B VAL4 必须最后一个出现。然后，重复该过程。

为确保到达 ADC 的第一个触发信号是来自 PWM A VAL5，应对 PWM B VAL4 比较事件进行编程，以产生一个支持所有触发信号的中断。然后禁用此比较事件中断，PWM 和 ADC 同步保持不变。

在代码中，必须使能 PWM B 子模块 0 VAL4 比较中断，并且必须对中断控制器进行编程，以使能此中断。代码如下：

```
/* Compare interrupt of Value 4 */
PWMB_SM0INTEN = PWMB_SM0INTEN_CMPIE_4;

/* Interrupt for the SM0 CMP Level 2 */
INTC_IPR8 |= INTC_IPR8_PWMB_CMP0;
```

VAL4 比较中断服务例程

PWM B 子模块 0 VAL4 比较中断已配置完毕。现在必须创建中断服务例程（ISR）以使能触发信号。

ISR 的名称为 IsrPWM。此例程的原型必须在代码的原型部分创建。

```
void IsrPWM(void);
```

此函数的名称必须复制到向量表中；如果使用默认 CodeWarrior 10.2 项目模板，则它位于文件 MC56F847xx_vector.asm 中（Project_Settings\Startup_Code 目录下）。因此，在地址 0x98，PWM B 子模块 0 的 76 号中断将包含以下语句：

```
JSR  >FIsrPWM
```

函数体本身包含以下操作：

- 使能 PWM A 和 B 子模块 0 VAL4 和 VAL5 触发信号
- 对 PWM B 模块应用 RUN 命令
- 禁用 PWM B 子模块 0 VAL4 比较中断
- 在中断控制器中禁用 PWM B 子模块 0 比较中断
- 清除 PWM B 子模块 0 VAL4 比较标志

包含所有这些操作的代码如下：

```
#pragma interrupt alignsp
void IsrPWM(void)
{
      /* Disable PWM B SM0 CMP VAL4 interrupt */
      PWMB_SM0INTEN &= ~PWMB_SM0INTEN_CMPIE_4;

      /* Enable triggers on VAL4 and VAL5*/
      PWMA_SM0TCTRL |= PWMA_SM0TCTRL_OUT_TRIG_EN_4 | PWMA_SM0TCTRL_OUT_TRIG_EN_5;
```

```
        PWMB_SM0TCTRL |= PWMB_SM0TCTRL_OUT_TRIG_EN_4 | PWMB_SM0TCTRL_OUT_TRIG_EN_5;

        /* Disables the interrupt in the INTC */
        INTC_IPR8 &= ~INTC_IPR8_PWMB_CMP0;

        /* Clears compare flag */
        PWMB_SM0STS |= PWMB_SM0STS_CMPF_4;
}
```

现在，所有模块均已配置完毕并同步，可以开始正常工作。

10 读取 ADC 采样

现在，所有外设都在工作，并且 ADC 在规定的时间点对所需的反馈进行采样。最后一项必须完成的任务是读取 ADC 的采样值，并在控制算法中加以使用。如果仅控制一台电机，可以利用 ADC 扫描结束中断来读取采样值并执行控制算法。但在本例中，有两台电机需要控制，最好把算法计算分散在不同时间执行。此外，第一台电机的计算无需等到第二台电机的 ADC 转换完成，因为前一台电机的反馈比后一台电机的反馈早 90 度就绪。最后，PWM 更新必须在重载之前进行，否则系统会延迟一个周期，稳定性会变差。

因此，控制算法将在三个时间点执行（参见图 9 ）：

- 电机 1 快速环控制算法在其电流反馈采样后立即执行
- 电机 2 快速环控制算法在 ADC 扫描结束中断时执行
- 电机 1 和电机 2 慢速环计算及 ADC 电流偏移量更新在 PWM B 半周期处执行

电机 **1** 快速环

从图 9 可看到，电机 1 电流反馈早在扫描结束中断之前就已就绪。因此，反馈就绪时即可从 ADC 读取反馈。ADC 转换时间是已知的，故在 ADC 触发后 1 µs 设置中断，确认电流采样已经完成。在此时间点产生中断的简单办法是使用 PWM A 中的一个比较中断，本例使用子模块 2 VAL4 比较中断。VAL4 值在子模块 0 VAL4 触发 1 µs 后设置为–4575 (0xEE21)。还必须使能比较中断，在 IsrPWM 中断例程中与触发信号一起使能即可。同时还要配置中断控制器。

示例代码如下：

```
/* Fast loop calculation */
PWMA_SM2VAL4 = 0xEE21;

/* Interrupt for the SM2 CMP Level 1 */
INTC_IPR9 |= INTC_IPR9_PWMA_CMP2_1;

/* M1 fast loop calculation 1us after the ADC trigger 4 on SM2 */
PWMA_SM2INTEN |= PWMA_SM2INTEN_CMPIE_4;
```

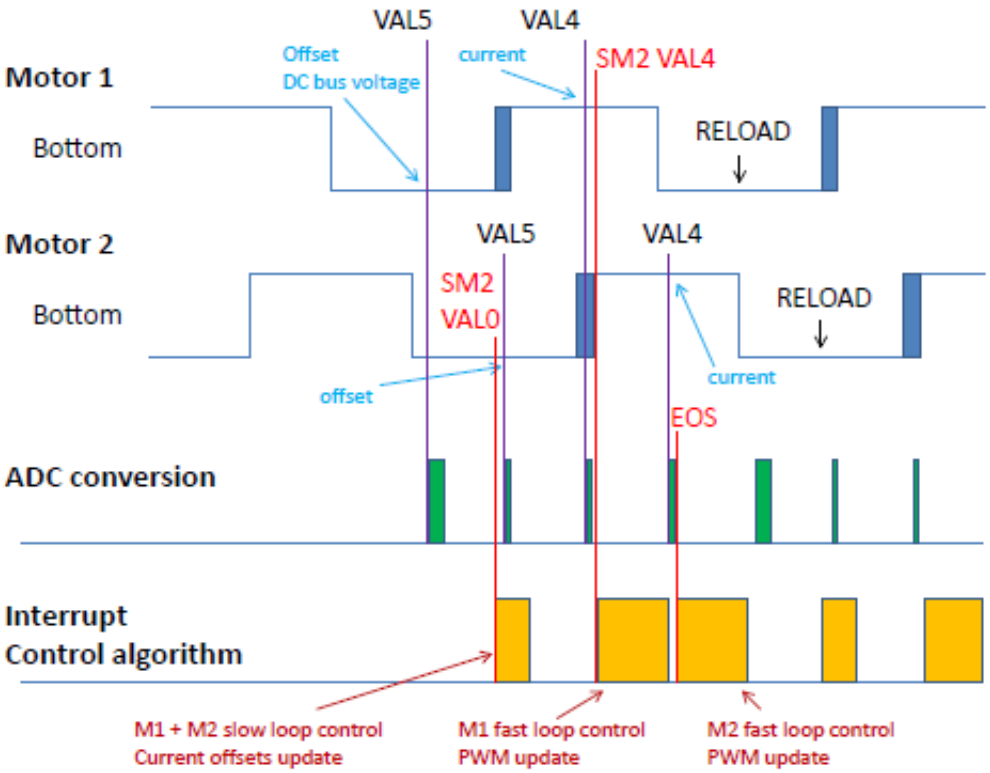

图 9. 读取 ADC 和执行控制算法

电机 **2** 快速环

最后采样的量是电机 2 电流。因此，利用 ADC 扫描结束中断来执行电机 2 快速环算法，而且已经配置 ADC 来产生扫描结束中断。现在只需配置中断控制器。示例代码如下：

```
/* Interrupt ADC end of scan Level 1 */
INTC_IPR2 |= INTC_IPR2_ADC_CC0_1;
```

电机 **1** 和电机 **2** 慢速环

只有一台电机时，慢速环可以与快速环算法在同一中断中执行。但在本例中，这会延长时间，另一台电机的快速环计算会被推迟。为了避免冲突，慢速环在另一个中断中计算。速度环控制所需的时间不像快速环算法那样长，因此两台电机可以在同一中断中计算。

此计算选择 PWM B 半周期（VAL0）时间。在此时间点上，电机 1 电流偏移量已采样完毕，可以进行滤波和更新，为测量下一电流做准备。并且可以计算电机 1 慢速环。其他应用逻辑，如状态机、温度检查、直流总线电压滤波等，也可以在此时间点应用。

包括中断进入的第一个区块至少应持续到读取电机 2 电流的时刻。如果短于此时间，则会读取前一周期的电流偏移量，这将导致中断无法使用 VAL0 比较事件源，因此必须将中断推迟到稍后的时间点。

在第二个区块中，会读取、滤波并更新电机 2 电流偏移量以进行电流测量。然后会计算电机 2 慢速环。可以在此时间点应用其他逻辑，例如电机 2 或整个应用的状态机。

因此，代码中必须使能比较中断，在 IsrPWM 中断例程中与触发信号一起使能即可。同时还要配置中断控制器。示例代码如下：

```
/* M1 and M2 slow loop calculation on half cycle of PWM B */
PWMB_SM2INTEN |= PWMB_SM2INTEN_CMPIE_0;

/* Interrupt for the SM2 CMP Level 1*/
INTC_IPR7 |= INTC_IPR7_PWMB_CMP2_1;
```

中断服务例程

中断已配置就绪，现在必须写出中断服务例程。

ISR 的名称为 IsrPWMAFastLoopCalc、IsrADC12Result 和 IsrPWMBSlowLoopCalc。这些例程的原型必须在代码的原型部分创建。

```
void IsrPWMAFastLoopCalc(void);
void IsrADC12Result(void);
void IsrPWMBSlowLoopCalc(void);
```

这些函数的名称必须复制到向量表中；如果使用默认 CodeWarrior 10.2 项目模板，则它位于文件 MC56F847xx_vector.asm 中（Project_Settings\Startup_Code 目录下）。因此，地址 0xA6 是 83 号中断，对应 PWM A 子模块 2 比较；地址 0x3C 是 30 号中断，对应 ADC 结果；地址 0x8C 是 70 号中断，对应 PWM B 子模块 2 比较。代码包含以下语句：

```
JSR  >FIsrPWMAFastLoopCalc
JSR  >FIsrADC12Result
JSR  >FIsrPWMBSlowLoopCalc
```

函数体本身包含用户控制算法（包括 ADC 读取和更新以及 PWM 更新），并且最后必须清除特定中断标志，以防止再次进入该中断。

函数体可以是如下形式：

```
#pragma interrupt saveall
void IsrPWMAFastLoopCalc(void)
{
     ...

     /* Clears compare flag */
     PWMA_SM2STS |= PWMA_SM2STS_CMPF_4;
}

#pragma interrupt saveall
void IsrADC12Result(void)
{
     ...

     /* Clears the interrupt flag */
     ADC12_STAT |= ADC12_STAT_EOSI0 | ADC12_STAT_EOSI1;
}

#pragma interrupt saveall
void IsrPWMBSlowLoopCalc(void)
{
     ...

     /* Clears compare flag */
     PWMB_SM2STS |= PWMB_SM2STS_CMPF_0;
}
```

现在，同步双电机 PMSM FOC 的 PWM 和 ADC 的所有必要步骤均已完成。

11 完整代码

已配置并同步 PWM 模块，并且已对触发和中断进行编程。ADC 及其中断均已配置完毕。模块已通过 crossbar 互连。本应用中用到的所有代码行如下所示。

中断向量表

文件 MC56F847xx_vector.asm（位于 Project_Settings\Startup_Code）

完整代码

```
JSR >FIsrADC12Result          ;/* 0x3c Interrupt no. 30 */
JSR >FIsrPWMBSlowLoopCalc     ;/* 0x8c Interrupt no. 70 */
JSR >FIsrPWM                  ;/* 0x98 Interrupt no. 76 */
JSR >FIsrPWMAFastLoopCalc     ;/* 0xa6 Interrupt no. 83 */
JSR >FIsrPWMSync              ;/* 0xaa Interrupt no. 85 */
```

原型

```
static void GPIOA_Init(void);
static void GPIOB_Init(void);
static void GPIOE_Init(void);
static void GPIOG_Init(void);
static void XBAR_Init(void);
static void PWMA_SM012_Init(void);
static void PWMB_SM012_Init(void);
static void PWMA_SM012_Run(void);
static void ADC12_Init(void);

void IsrPWMSync(void);
void IsrPWM(void);
void IsrPWMAFastLoopCalc(void);
void IsrADC12Result(void);
void IsrPWMBSlowLoopCalc(void);
```

函数

```
static void GPIOA_Init(void)
{
   /* Enable GPIOA clock */
   SIM_PCE0 |= SIM_PCE0_GPIOA;

   /* ADCA */
   GPIOA_PER |= (GPIOA_PER_PE_0 | GPIOA_PER_PE_1 | GPIOA_PER_PE_2 | GPIOA_PER_PE_3 |
GPIOA_PER_PE_4 | GPIOA_PER_PE_5 | GPIOA_PER_PE_6 |
   GPIOA_PER_PE_7);

   SIM_GPSAL &= ~(SIM_GPSAL_A0);
}



static void GPIOB_Init(void)
{
   /* Enable GPIOB clock */
   SIM_PCE0 |= SIM_PCE0_GPIOB;

   /* ADCB */
   GPIOB_PER |= (GPIOB_PER_PE_0 | GPIOB_PER_PE_1 | GPIOB_PER_PE_2 | GPIOB_PER_PE_3 |
GPIOB_PER_PE_4 | GPIOB_PER_PE_5 | GPIOB_PER_PE_6 |
   GPIOB_PER_PE_7);
}

static void GPIOE_Init(void)
{
   /* Enable GPIOE clock */
   SIM_PCE0 |= SIM_PCE0_GPIOE;

   /* PWMA PWMB */
   GPIOE_PER |= (GPIOE_PER_PE_0 | GPIOE_PER_PE_1 | GPIOE_PER_PE_2 | GPIOE_PER_PE_3 |
GPIOE_PER_PE_4 | GPIOE_PER_PE_5 | GPIOE_PER_PE_8 |
   GPIOE_PER_PE_9);
   SIM_GPSEL &= ~(SIM_GPSEL_E4 | SIM_GPSEL_E5);
   SIM_GPSEH &= ~(SIM_GPSEH_E8 | SIM_GPSEH_E9);

}

static void GPIOG_Init(void)
{
   /* Enable GPIOG clock */
```

```
   SIM_PCE0 |= SIM_PCE0_GPIOG;

   /* PWM B */
   GPIOG_PER |= (GPIOG_PER_PE_0 | GPIOG_PER_PE_1 | GPIOG_PER_PE_2 | GPIOG_PER_PE_3);
   SIM_GPSGL &= ~(SIM_GPSGL_G0 | SIM_GPSGL_G1 | SIM_GPSGL_G2 |
   SIM_GPSGL_G3);

}

static void XBAR_Init(void)
{
   /* PWM A & B trigger to XBAR B */
   XBARB_SEL0 = 8 | (22 << 8);

   /* AOI out = A | B, i.e. PWM A trigger or PWM B trigger */
   AOI_BFCRT010 = AOI_BFCRT010_PT0_AC_0 | AOI_BFCRT010_PT0_BC | AOI_BFCRT010_PT0_CC |
AOI_BFCRT010_PT0_DC | AOI_BFCRT010_PT1_AC | AOI_BFCRT010_PT1_BC_0 |
AOI_BFCRT010_PT1_CC | AOI_BFCRT010_PT1_DC;

   AOI_BFCRT230 = 0;

   /* ADC A trigger from AOI */
   XBARA_SEL6 = 46;

}

static void PWMA_SM012_Init(void)
{
   /* Enable clock for PWM SM 0 - 2 */
   SIM_PCE3 |= (SIM_PCE3_PWMACH0 | SIM_PCE3_PWMACH1 | SIM_PCE3_PWMACH2);
   /* Half cycle reload */
   PWMA_SM0CTRL = PWMA_SM0CTRL_HALF;
   PWMA_SM1CTRL = PWMA_SM1CTRL_HALF;
   PWMA_SM2CTRL = PWMA_SM2CTRL_HALF;

   /* PWM setup for 3 phases */
   PWMA_SM0CTRL2 = 0xC080;
   PWMA_SM1CTRL2 = 0xC20E;
   PWMA_SM2CTRL2 = 0xC20E;

   /* setup for pwm frequency of 10KHz */
   PWMA_SM0INIT = 0xEC78;
   PWMA_SM1INIT = 0xEC78;
   PWMA_SM2INIT = 0xEC78;

   /* setup for pwm frequency of 10KHz */
   PWMA_SM0VAL1 = 0x1387;
   PWMA_SM1VAL1 = 0x1387;
   PWMA_SM2VAL1 = 0x1387;

   PWMA_SM0VAL0 = 0x0000;
   PWMA_SM1VAL0 = 0x0000;
   PWMA_SM2VAL0 = 0x0000;

   PWMA_SM0VAL2 = 0xF63C;
   PWMA_SM1VAL2 = 0xF63C;
   PWMA_SM2VAL2 = 0xF63C;

   PWMA_SM0VAL3 = 0x09C3;
   PWMA_SM1VAL3 = 0x09C3;
   PWMA_SM2VAL3 = 0x09C3;

   PWMA_SM0VAL4 = 0xEDBD;  /* Current measurement value */
   PWMA_SM1VAL4 = 0x0000;
   PWMA_SM2VAL4 = 0xEE21;  /* Fast loop calculation */
   PWMA_SM0VAL5 = 0x0145;  /* Offset measurement value */
   PWMA_SM1VAL5 = 0xF63C;  /* Sync for the other motor's PWM */
   PWMA_SM2VAL5 = 0x0000;

   /* deadtime count register 0 and 1 = 2.0 us */
```

```
    PWMA_SM0DTCNT0 = 0x00C8;
    PWMA_SM1DTCNT0 = 0x00C8;
    PWMA_SM2DTCNT0 = 0x00C8;

    PWMA_SM0DTCNT1 = 0x00C8;
    PWMA_SM1DTCNT1 = 0x00C8;
    PWMA_SM2DTCNT1 = 0x00C8;

    /* Fault A 0 - 3 inactive */
    PWMA_SM0DISMAP0 = 0;
    PWMA_SM1DISMAP0 = 0;
    PWMA_SM2DISMAP0 = 0;

    /* Fault A 4 - 7 inactive */
    PWMA_SM0DISMAP1 = 0;
    PWMA_SM1DISMAP1 = 0;
    PWMA_SM2DISMAP1 = 0;

    /* Compare interrupt of Value 5, used to sync PWM A and PWM B */
    PWMA_SM1INTEN = PWMA_SM1INTEN_CMPIE_5;

    /* Interrupt for the SM1 CMP L2 and SM2 CMP L1 */
    INTC_IPR9 |= INTC_IPR9_PWMA_CMP1 | INTC_IPR9_PWMA_CMP2_1;

    /* Enables PWM output */
    PWMA_OUTEN |= (PWMA_OUTEN_PWMA_EN_0 | PWMA_OUTEN_PWMB_EN_0 | PWMA_OUTEN_PWMA_EN_1 |
PWMA_OUTEN_PWMB_EN_1 | PWMA_OUTEN_PWMA_EN_2 | PWMA_OUTEN_PWMB_EN_2);

    /* Clear LDOK bit */
    PWMA_MCTRL |= PWMA_MCTRL_CLDOK_0 | PWMA_MCTRL_CLDOK_1 | PWMA_MCTRL_CLDOK_2;
    /* LDOK */
    PWMA_MCTRL |= PWMA_MCTRL_LDOK_0 | PWMA_MCTRL_LDOK_1 | PWMA_MCTRL_LDOK_2;
}

static void PWMB_SM012_Init(void)
{
    /* Enable clock for PWM SM 0 - 3 */
    SIM_PCE3 |= (SIM_PCE3_PWMBCH0 | SIM_PCE3_PWMBCH1 | SIM_PCE3_PWMBCH2);

    /* Half cycle reload */
    PWMB_SM0CTRL = PWMB_SM0CTRL_HALF;
    PWMB_SM1CTRL = PWMB_SM1CTRL_HALF;
    PWMB_SM2CTRL = PWMB_SM2CTRL_HALF;

    /* PWM setup for 3 phases, uses only SM 1 - 2 */
    PWMB_SM0CTRL2 = 0xC080;
    PWMB_SM1CTRL2 = 0xC20E;
    PWMB_SM2CTRL2 = 0xC20E;

    /* setup for pwm frequency of 10KHz */
    PWMB_SM0INIT = 0xEC78;
    PWMB_SM1INIT = 0xEC78;
    PWMB_SM2INIT = 0xEC78;

    /* setup for pwm frequency of 10KHz */
    PWMB_SM0VAL1 = 0x1387;
    PWMB_SM1VAL1 = 0x1387;
    PWMB_SM2VAL1 = 0x1387;

    PWMB_SM0VAL0 = 0x0000;
    PWMB_SM1VAL0 = 0x0000;
    PWMB_SM2VAL0 = 0x0000;

    PWMB_SM0VAL2 = 0xF63C;
    PWMB_SM1VAL2 = 0xF63C;
    PWMB_SM2VAL2 = 0xF63C;

    PWMB_SM0VAL3 = 0x09C3;
    PWMB_SM1VAL3 = 0x09C3;
    PWMB_SM2VAL3 = 0x09C3;
```

```
    PWMB_SM0VAL4 = 0xEDBD;  /* Current measurement value */
    PWMB_SM1VAL4 = 0x0000;
    PWMB_SM2VAL4 = 0x0000;

    PWMB_SM0VAL5 = 0x0145;  /* Offset measurement value */
    PWMB_SM1VAL5 = 0x0000;
    PWMB_SM2VAL5 = 0x0000;

    /* deadtime count register 0 and 1 = 2.0 us */
    PWMB_SM0DTCNT0 = 0x00C8;
    PWMB_SM1DTCNT0 = 0x00C8;
    PWMB_SM2DTCNT0 = 0x00C8;

    PWMB_SM0DTCNT1 = 0x00C8;
    PWMB_SM1DTCNT1 = 0x00C8;
    PWMB_SM2DTCNT1 = 0x00C8;

    /* Fault B 0 - 3 inactive */
    PWMB_SM0DISMAP0 = 0;
    PWMB_SM1DISMAP0 = 0;
    PWMB_SM2DISMAP0 = 0;

    /* Fault B 4 - 7 inactive */
    PWMB_SM0DISMAP1 = 0;
    PWMB_SM1DISMAP1 = 0;
    PWMB_SM2DISMAP1 = 0;

    /* Compare interrupt of Value 4, enable triggers from PWM A and B to ADC */
    PWMB_SM0INTEN = PWMB_SM0INTEN_CMPIE_4;

    /* Interrupt for the SM0 CMP L2 */
    INTC_IPR8 |= INTC_IPR8_PWMB_CMP0;

    /* Interrupt for the SM2 CMP L1*/
    INTC_IPR7 |= INTC_IPR7_PWMB_CMP2_1;

    /* Clear LDOK bit */
    PWMB_MCTRL |= PWMB_MCTRL_CLDOK_0 | PWMB_MCTRL_CLDOK_1 | PWMB_MCTRL_CLDOK_2;

    /* LDOK */
    PWMB_MCTRL |= PWMB_MCTRL_LDOK_0 | PWMB_MCTRL_LDOK_1 | PWMB_MCTRL_LDOK_2;

    /* Enables PWM output */
    PWMB_OUTEN |= (PWMB_OUTEN_PWMA_EN_3 | PWMB_OUTEN_PWMB_EN_3 | PWMB_OUTEN_PWMA_EN_1 |
PWMB_OUTEN_PWMB_EN_1 | PWMB_OUTEN_PWMA_EN_2 | PWMB_OUTEN_PWMB_EN_2);
}

static void PWMA_SM012_Run(void)
{
    /* Enable clock */
    PWMA_MCTRL |= PWMA_MCTRL_RUN_0 | PWMA_MCTRL_RUN_1 | PWMA_MCTRL_RUN_2;
}

static void ADC12_Init(void)
{
    /* enable clock to ADC modules */
    SIM_PCE2 |= SIM_PCE2_CYCADC;


    /* SMODE - triggered parallel, SYNC0 - enabled, End of scan interrupt */
    ADC12_CTRL1 = 0x1805;

    /* Simultaneous parallel mode; DIV0 = 5, 20MHz */
    ADC12_CTRL2 = ADC12_CTRL2_DIV0_2 | ADC12_CTRL2_SIMULT;

    /* Channel assignment */
    /* S0   S1   S2   S3    S4   S5   S6   S7 */
    /* A0   A0   A2   A0    A2   A0   A0   A0 */
    ADC12_CLIST1 = 0x0200;
```

```
   ADC12_CLIST2 = 0x0002;

   /* S8  S9  S10 S11  S12 S13 S14 S15 */
   /* B0  B4  B2  B0   B2  B0  B0  B0  */
   ADC12_CLIST3 = 0x8AC8;
   ADC12_CLIST4 = 0x888A;

   /* Sample count 10 channels */
   ADC12_SDIS = 0xE0E0;

   /* Sync at */
   /* S0  S1  S2  S3   S4  S5  S6  S7 */
   /* x       x   x    x              */
   ADC12_SCTRL = 0x010D;

   /* power-up delay set to 26 clocks */
   ADC12_PWR = 0x01A0;

   /* ADCA Speed <=20MHz; ADCB Speed<=20MHz; DIV1 = 5, 20 MHz */
   ADC12_PWR2= ADC12_PWR2_SPEEDA | ADC12_PWR2_SPEEDB | ADC12_PWR2_DIV1_2;

   /* Interrupt end of scan L1 */
   INTC_IPR2 |= INTC_IPR2_ADC_CC0_1;
}

#pragma interrupt alignsp
void IsrPWMSync(void)
{
   if ((PWMA_SM1STS & PWMA_SM1STS_CMPF_5) > 0 )
   {
   /* Starts PWM B */
   PWMB_MCTRL |= PWMB_MCTRL_RUN_0 | PWMB_MCTRL_RUN_1 | PWMB_MCTRL_RUN_2 |
PWMB_MCTRL_RUN_3; /* Enable clock */

   /* Disable PWM SM1 CMP interrupt from 5 */
   PWMA_SM1INTEN &= ~PWMA_SM1INTEN_CMPIE_5;

   /* Enables the PWM SM1 CMP interrupt from 4 */
   PWMA_SM1INTEN |= PWMA_SM1INTEN_CMPIE_4;

   /* Clears compare flag */
   PWMA_SM1STS |= PWMA_SM1STS_CMPF_5;
   }
}

#pragma interrupt alignsp
void IsrPWM(void)
{
   /* Disable PWM SM0 CMP interrupt */
   PWMA_SM0INTEN &= ~PWMA_SM0INTEN_CMPIE_4;
   PWMB_SM0INTEN &= ~PWMB_SM0INTEN_CMPIE_4;

   /* Enable triggers on VAL4 and VAL5 */
   PWMA_SM0TCTRL |= PWMA_SM0TCTRL_OUT_TRIG_EN_4 | PWMA_SM0TCTRL_OUT_TRIG_EN_5;
   PWMB_SM0TCTRL |= PWMB_SM0TCTRL_OUT_TRIG_EN_4 | PWMB_SM0TCTRL_OUT_TRIG_EN_5;

   /* M1 fast loop calculation 1us after the ADC trigger 4 on SM2 */
   PWMA_SM2INTEN |= PWMA_SM2INTEN_CMPIE_4;

   /* M1 and M2 slow loop calculation on half cycle of PWM B */
   PWMB_SM2INTEN |= PWMB_SM2INTEN_CMPIE_0;

   /* Disables the interrupt in the INTC */
   INTC_IPR8 &= ~INTC_IPR8_PWMB_CMP0;

   /* Clears compare flag */
   PWMB_SM0STS |= PWMB_SM0STS_CMPF_4;
}

#pragma interrupt saveall
```

```
void IsrPWMAFastLoopCalc(void)
{
   ...

   /* Clears compare flag */
   PWMA_SM2STS |= PWMA_SM2STS_CMPF_4;
}


#pragma interrupt saveall
void IsrADC12Result(void)
{
   ...

   /* Clears the interrupt flag */
   ADC12_STAT |= ADC12_STAT_EOSI0 | ADC12_STAT_EOSI1;
}

#pragma interrupt saveall
void IsrPWMBSlowLoopCalc(void)
{
   ...

   /* Clears compare flag */
   PWMB_SM2STS |= PWMB_SM2STS_CMPF_0;
}
```

函数调用顺序

应用初始化要求按如下顺序调用函数:

```
GPIOA_Init();
GPIOB_Init();
GPIOE_Init();
GPIOG_Init();
XBAR_Init();
PWMA_SM012_Init();
PWMB_SM012_Init();
PWMA_SM012_Run();
```

12 定义和首字母缩略词

GPIO	通用端口输入输出
ADC	模数转换器
XBAR	crossbar
PWM	脉宽调制
ISR	中断服务例程
AOI	与/或/非模块
SIM	系统集成模块
CW	CodeWarrior
FSLESL	飞思卡尔嵌入式软件库，该软件工具可从 **freescale.com** 下载
GFLIB	通用函数库
MCLIB	电机控制库
GDFLIB	通用数字滤波器库
ACLIB	高级控制库
DSC	数字信号控制器

下一页继续介绍此表...

FOC	磁场定向控制
PMSM	永磁同步电机
电机控制	在本应用笔记中，电机控制是指控制 BLDC PMSM、交流感应电机或其他电机的过程。

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4608
Revision 0, 10/2012