硬件抽象层

© 2018 Microchip Technology Inc.

1 硬件抽象层

1.1 简介

硬件抽象层（Hardware Abstraction Layer，HAL）为更高层（例如：应用程序框架和客户应用程序等等）提供基于API函数的服务，允许更高层独立于实际的硬件细节执行面向硬件的操作。本文档提供了硬件抽象层及其架构、元素和使用模型的详细说明。

1.1.1 架构原理和假设

1.1.1.1 开发硬件抽象层的动机

传统的电机控制应用笔记源代码项目（2015年之前发布）很少使用硬件抽象。其代码段中有多个部分使用不同的接口访问硬件/外设。因此存在以下挑战：

1. 将代码从一个硬件移植到另一个硬件需要进行大量的搜索和替换操作。
2. 每一种PIM和硬件的组合都需要专门版本的代码项目——维护起来十分耗费资源。
3. 与算法相关的代码和硬件访问混合在一起——很难在不依赖硬件的前提下改进算法。

开发硬件抽象层旨在应对上述挑战。

1.1.1.2 设计目标

硬件抽象层的设计有以下几个顶级目标：

1. 硬件抽象层应允许客户以最少的工作量生成其电路板特定的硬件抽象层文件。
2. 要求最少的执行时间开销。
3. 使用模块化架构。
4. 利用MCC生成外设驱动程序（可用时）。

1.1.1.3 与8位和32位器件系列的兼容性

硬件抽象层目前的*实现方式*设计为使用16位器件系列，特别是dsPIC33E系列。ADC、DMA、PWM、QEI和系统时钟等器件外设的架构通常因16位器件系列的不同而异。因此，其他16位器件系列与硬件抽象层的这种实现方式的兼容程度会有所不同，具体取决于所使用特定器件的特性和外设架构。

硬件抽象层的这种*实现方式*并非面向8位和32位器件系列而设计。此外，尽管硬件抽象层的这种实现中使用的静态函数驱动程序方法与8位器件系列配合完美，但是对于动态驱动程序方法更加合适的32位器件系列而言，效率却很低。不过，在理论上，硬件抽象层的模块化*架构*将允许其在使用32位器件系列时，用plib来替换当前的外设驱动程序集。

© 2018 Microchip Technology Inc.

总而言之，就8位和32位器件系列而言：

- 硬件抽象层的*实际实现*——不兼容。
- 硬件抽象层的*接口*——目前不兼容。不过，通过包装函数添加窄“适配”层可在很大程度上解决此问题。
- 硬件抽象层的*架构*——不兼容性无法感知。

1.1.1.4 电机控制算法支持

硬件抽象层目前的*实现方式*设计为在常规情况下支持磁场定向控制算法，在特殊情况下支持无传感器双电流检测电阻算法（例如：AN1078和AN1292等）。

尽管如此，初步分析表明，硬件抽象层及其架构的这种实现方式中包含的接口通常支持实现以下各项：

1. 单电流检测电阻电流重构（AN1299）
2. 带有反电动势滤波的无传感器BLDC控制（六步换相；AN1160）
3. PMSM的正弦控制，即非FOC（AN1017）
4. 带传感器的BLDC控制（AN957）

除此之外，硬件抽象层目前的实现方式还支持任何其他较新的控制算法（例如：直接转矩控制等），这些算法使用PWM和ADC模块的方式与上面列出的算法类似。

1.1.1.5 通用应用程序支持

硬件抽象层的核心元素外设驱动程序的开发从不同16位器件系列的通用外设驱动程序集入手。因此，硬件抽象层内的外设驱动程序与大多数典型的通用应用程序完全兼容。

除此之外，硬件抽象层的另外两个元素板级支持包和硬件访问函数是为满足典型电机控制应用的需求而专门设计的。不过，这两个元素的基本架构并不妨碍添加通用接口。根据所述应用程序的具体情况，可能需要在硬件抽象层中添加附加接口才能支持通用应用程序。

1.1.1.6 电机控制应用特定的优化和假设

硬件抽象层的实现使用以下专门为电机控制用例设计的函数：

1. 为了在减少指令周期开销（设计目标2）的同时保持模块化（设计目标3），硬件抽象层广泛使用静态内联函数，而不是常规的函数调用。常规的函数调用会导致函数调用开销，通常只能用于不常调用的外设驱动程序初始化函数，特别是在时序敏感的控制环中。

© 2018 Microchip Technology Inc.

2. 硬件抽象层不使用中断服务程序（Interrupt Service Routine，ISR）的回调函数，而是将预处理器宏定义用于可在应用程序中使用的ISR函数头。这仍是为了减少由于函数调用导致的指令周期开销。
3. 考虑到设计目标2，一些基本硬件抽象层操作（如SFR位置1、SFR位清零、SFR读取和SFR写入操作）可以使用静态内联函数或使用预处理器宏实现。在这一决策点上，我们假定使用静态内联函数，而不是预处理器宏，因为编译器对静态内联函数更具“可见性”，而典型预处理器宏往往需要先经过处理才有机会被编译器发现。与预处理器宏相比，选择使用静态内联函数可使编译器在应用程序中更好地进行优化。

© 2018 Microchip Technology Inc.

1.2 硬件抽象层的元素

图1所示为与电机控制（Motor Control，MC）应用程序框架和客户应用程序相关的硬件抽象层的典型框图。

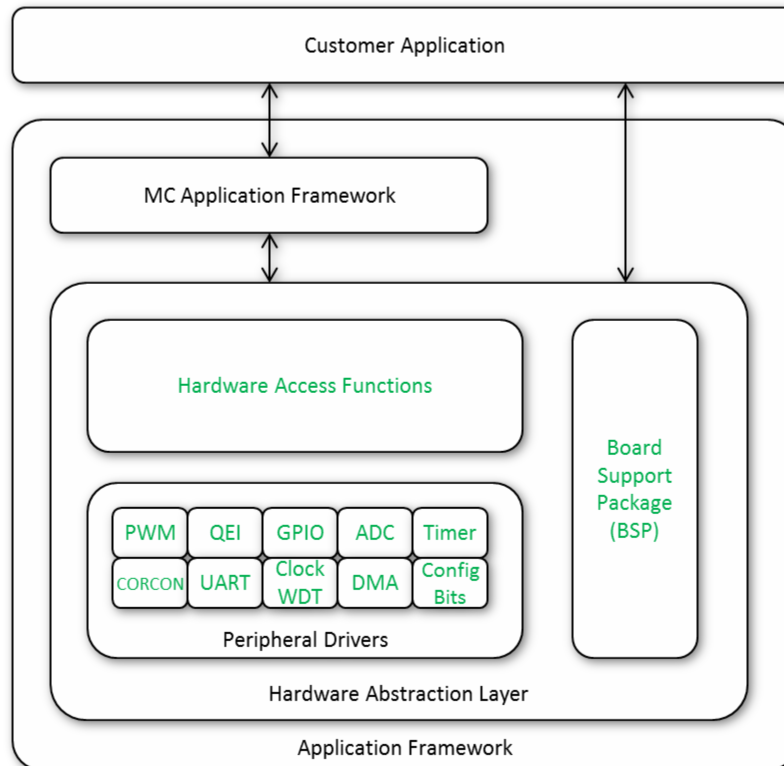

图1：硬件抽象层框图

1.2.1 外设驱动程序

该模块提供了直接访问器件外设的简单函数。外设驱动程序函数的名称中包含一个前缀，具体前缀与函数所处理的特定器件外设有关。例如，ADC1模块的外设驱动程序函数的名称带有前缀ADC1_。下面列出了一些其他示例：

```
ADC1_Initialize()
DMA_SoftwareTriggerEnable(DMA_CHANNEL channel)
OSCILLATOR_Initialize(void)
PWM2_DutycycleSet(uint16_t dutyCycle)
QEI1_PositionCount16bitRead(void)
TMR1_Start(void)
```

目前，硬件抽象层支持以下系统功能和器件外设：

1. 系统功能
    a. 振荡器配置
    b. 器件配置位
    c. CORCON处理

© 2018 Microchip Technology Inc.

   d. GPIO配置与PPS处理
   e. 所有支持HAL的外设的中断管理
   f. 看门狗定时器
2. ADC1模块
3. PWM模块
4. Timer1模块
5. QEI1模块
6. DMA模块
7. UART1模块

注：

- 外设驱动程序函数目前是手写的。这些手写外设驱动程序函数的外观保持与MCC支持的器件通过MCC生成的函数类似。
- 将来，硬件抽象层将包括对其他外设的支持以及对当前所支持外设的附加API的支持。
- 将来，对当前外设模块集的支持将适时扩展到这些模块的更多实例（例如：Timer1 -> Timer2和Timer3等）。

1.2.2 硬件访问函数

此模块提供将硬件抽象层中的低层外设驱动程序与更高层应用程序或应用程序框架接口的函数。此模块为外观模式；它包含可转换成外设驱动程序操作的简单包装函数以及利用外设驱动程序来完成器件/硬件特定操作的更复杂函数。硬件访问函数的名称中带有前缀HAL_。下面列出了几个示例：

```
HAL_PwmUpperTransistorsOverrideDisable_Motor1(void)
HAL_PwmUpperTransistorsOverrideLow_Motor1(void)
HAL_PwmSetDutyCyclesIdentical_Motor1(uint16_t dc)
HAL_PwmSetDutyCyclesMotor1(const uint16_t *pdc)
```

该模块还提供了一组宏#define的名称，旨在通过实际器件特定的中断服务程序（ISR）和陷阱函数名称实现抽象化。下面列出了几个示例：

```
#define HAL_MATHERROR_TRAP_FUNCTION     _MathError
#define HAL_DMAC_TRAP_FUNCTION          _DMACError
#define HAL_ADC1_ISR                    _AD1Interrupt
#define HAL_DMA0_ISR                    _DMA0Interrupt
```

这些ISR和陷阱函数的名称适合在具有适当编译器属性的最终应用程序中使用。

1.2.3 板级支持包

板级支持包（Board Support Package，BSP）模块的主要目的是为用户提供一个访问点，以便修改硬件映射的详细信息，而无需进行大量的搜索和替换操作。为了实现这个目标，需将BSP分成两个相互依赖的接口：

© 2018 Microchip Technology Inc.

应用程序接口和外设接口。这种方法将定位*到应用程序接口的行为/功能映射*，同时保持*外设接口内的实际硬件映射*。

1.2.3.1 应用程序接口

BSP提供了一个通用的应用程序接口，允许从应用程序抽象访问特定的硬件功能，而不依赖于实际的硬件名称或硬件连接。此处定义的通用应用程序宏的*实际名称不应更改*并且可以在应用程序中使用，而不依赖于所使用的实际硬件。在将应用程序移植到不同的硬件时，可以更新*映射定义*以获得所需的硬件行为，同时使应用程序代码保持不变。

1.2.3.1.1 GPIO接口

通用IO（如LED、开关和测试点）在应用程序中使用此处定义的通用宏名称进行接口。这些宏名称定义为基于所需的应用程序硬件行为映射到适当的外设接口宏。以下代码片段显示了dsPICDEM™ MCLV-2开发板的应用程序接口映射。

```
#define BSP_LED_GP1                               BSP_LATCH_MCLV2_LED_D2
#define BSP_LED_GP2                               BSP_LATCH_MCLV2_LED_D17
#define BSP_TESTPOINT_GP1                         BSP_LATCH_PIM_TESTPOINT_RD8
#define BSP_TESTPOINT_GP2                         BSP_LATCH_MCLV2_TESTPOINT_CANTX
#define BSP_TESTPOINT_GP3                         BSP_LATCH_MCLV2_TESTPOINT_CANRX
#define BSP_TESTPOINT_GP4                         BSP_PORT_PIM_TESTPOINT_HOME
#define BSP_TESTPOINT_GP5                         BSP_LATCH_PIM_TESTPOINT_PGC
#define BSP_TESTPOINT_GP6                         BSP_LATCH_PIM_TESTPOINT_PGD
#define BSP_BUTTON_GP1                            BSP_PORT_MCLV2_BUTTON_S2
#define BSP_BUTTON_GP2                            BSP_PORT_MCLV2_BUTTON_S3

#define BSP_MOTOR1_ADCCHANNEL_POT                 ANALOG_CHANNEL_AN13
#define BSP_MOTOR1_ADCCHANNEL_VDC                 ANALOG_CHANNEL_AN10
#define BSP_MOTOR1_ADCCHANNEL_IM1                 ANALOG_CHANNEL_AN1
#define BSP_MOTOR1_ADCCHANNEL_IM2                 ANALOG_CHANNEL_AN0
#define BSP_MOTOR1_ADCCHANNEL_ISUM                ANALOG_CHANNEL_AN2

#define BSP_MOTOR1_PHASEA_CURRENTSENSE()          ADC_SH_CHANNEL2()
#define BSP_MOTOR1_PHASEB_CURRENTSENSE()          ADC_SH_CHANNEL1()
#define BSP_MOTOR1_PHASEC_CURRENTSENSE()          0
#define BSP_MOTOR1_SUMPHASE_CURRENTSENSE()        ADC_SH_CHANNEL3()
#define BSP_MOTOR1_ADC_OUTPUT_POT()               ADC_SH_CHANNEL0()
#define BSP_MOTOR1_ADC_OUTPUT_VBUS()              ADC_SH_CHANNEL0()
```

在以上代码片段中，应用程序接口宏`BSP_LED_GP1`可以在应用程序中直接使用，而外设接口宏`BSP_LATCH_MCLV2_LED_D2`在BSP的外设接口部分中定义以提供所需的外设访问方法。

除此之外，BSP还提供静态内联函数来实现简单的IO操作（例如，在LED或测试点上设置高电平/低电平状态以及读取按钮状态）。下面的代码片段列出了其中的一些函数。

```
inline static void BSP_LedGp1Activate() { BSP_LED_GP1 = 1; }
inline static void BSP_LedGp1Deactivate() { BSP_LED_GP1 = 0; }
inline static bool BSP_ButtonIsPressedGp1() { return BSP_BUTTON_GP1; }
```


© 2018 Microchip Technology Inc.

1.2.3.1.2 模拟接口

应用程序接口支持两种不同类型的模拟接口：

模拟多路开关接口——这实际上是从电机控制板上的模拟信号到器件上相应模拟通道的映射。这种类型的接口对于具有带模拟输入多路开关的ADC外设的器件非常有用，它允许单个采样保持通道扫描多个器件模拟输入通道（一次扫描一个）。模拟多路开关接口适合与_ChannelSelectSet()等ADC外设驱动程序函数一起使用，以便在ADC输入多路开关上选择所需的通道。接下来，在采样和转换完成后，可使用ADC1_ConversionResultChannel0Get()等适当的ADC外设驱动程序函数获取所选模拟信号的模拟值。

对于图2给出的示例，我们使用这种类型的模拟接口来连续采样电位器（V_{POT}）和直流母线电压检测（V_{DC}）输入，如图2所示。

```
#define BSP_MOTOR1_ADCCHANNEL_POT          ANALOG_CHANNEL_AN13
#define BSP_MOTOR1_ADCCHANNEL_VDC          ANALOG_CHANNEL_AN10
```

此处的宏BSP_MOTOR1_ADCCHANNEL_POT和BSP_MOTOR1_ADCCHANNEL_VDC可以直接在应用程序框架中使用，而ANALOG_CHANNEL_AN13和ANALOG_CHANNEL_AN10是通过BSP助手分别定义为编号10和13的别名。

有关此用例的更多详细信息，请参见第1.3.3.3.3节“ADC通道切换”。

静态模拟映射接口——该接口提供从电机控制板上的模拟信号到其对应的ADC通道缓冲区的直接映射。这种类型的模拟接口在以下情况下十分有用：电机控制板上的模拟信号连接到始终通过专用采样保持通道进行采样和转换并且转换结果存储到ADC缓冲区之一的器件模拟引脚。

图2给出了这种类型的模拟接口适用的典型情况。在这种情况下，使用采样保持通道CH1、CH2和CH3同时对电机相电流检测输入进行采样。应用程序接口对宏进行定义，以提供电路板信号名称和适当外设访问方法之间的直接映射，如下面的代码片段所示。

```
#define BSP_MOTOR1_PHASEA_CURRENTSENSE()     ADC_SH_CHANNEL2()
#define BSP_MOTOR1_PHASEB_CURRENTSENSE()     ADC_SH_CHANNEL1()
#define BSP_MOTOR1_PHASEC_CURRENTSENSE()     0
```

此处的外设访问方法ADC_SH_CHANNEL2()和ADC_SH_CHANNEL1()通过BSP分别定义为外设驱动程序函数ADC1_ConversionResultChannel2Get()和ADC1_ConversionResultChannel1Get()的别名。

有关此用例的更多详细信息，请参见第1.3.3.3.5节“相电流”。

© 2018 Microchip Technology Inc.

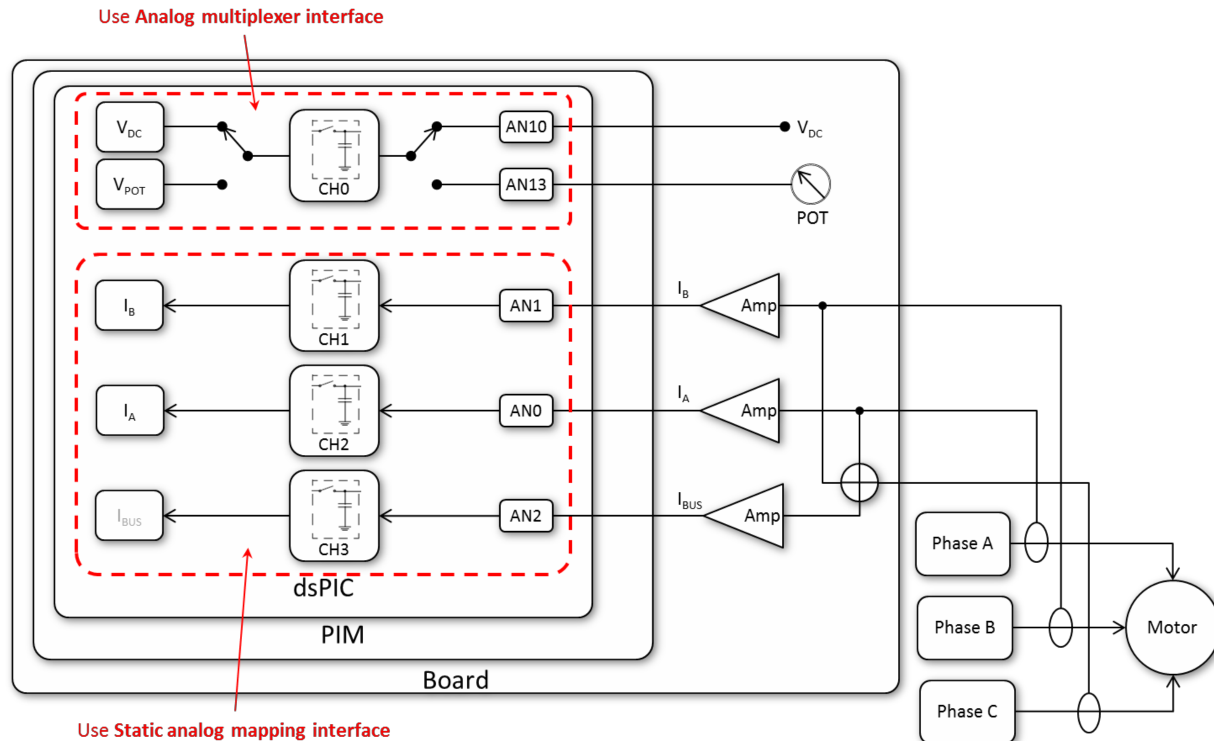

图2：典型应用中的模拟通道接口

1.2.3.2 外设接口

BSP使用外设接口将硬件特定宏名称映射到其相应的外设驱动程序/访问方法。在此接口中定义的宏仅用作一种定义映射的方法，不得直接用于应用程序/框架。在将应用程序移植到不同的硬件时，应更新此接口中定义的宏名称以匹配新硬件上的信号名称。

1.2.3.2.1 GPIO接口

外设接口提供硬件特定GPIO宏名称与其外设访问方法之间的映射，这种情况下即为PORT和LATCH寄存器的特定位。以下代码片段显示了dsPICDEM MCLV-2开发板的外设接口映射。

```
#define BSP_PORT_MCLV2_BUTTON_S2            !PORTGbits.RG7
#define BSP_PORT_MCLV2_BUTTON_S3            !PORTGbits.RG6
#define BSP_LATCH_MCLV2_LED_D2              LATDbits.LATD6
#define BSP_LATCH_MCLV2_LED_D17             LATDbits.LATD5
#define BSP_LATCH_PIM_TESTPOINT_RD8         LATDbits.LATD8
#define BSP_PORT_PIM_TESTPOINT_RD8          PORTDbits.RD8
#define BSP_LATCH_MCLV2_TESTPOINT_CANTX     LATCbits.LATC8
#define BSP_LATCH_MCLV2_TESTPOINT_CANRX     LATCbits.LATC9
#define BSP_LATCH_PIM_TESTPOINT_HOME        LATCbits.LATC10
#define BSP_PORT_PIM_TESTPOINT_HOME         PORTCbits.RC10
#define BSP_PORT_PIM_TESTPOINT_RD8          PORTDbits.RD8
#define BSP_LATCH_PIM_TESTPOINT_PGC         LATBbits.LATB6
#define BSP_LATCH_PIM_TESTPOINT_PGD         LATBbits.LATB5
```


© 2018 Microchip Technology Inc.

这些外设接口宏名称（例如：`BSP_LATCH_MCLV2_LED_D2`）仅用作一种定义映射的方法，不得直接用于应用程序/框架。

1.2.3.2.2 PWM通道到电机相的映射

外设接口还提供相应方法来定义PWM通道和电机相之间的映射。该映射不供应用程序接口使用；而是用于一些硬件访问函数以针对交叉连接的相自动调整，同时使用相应宏作为访问1x3占空比值数组的索引。

Microchip开发板通常将PWM通道线性映射到其相应的电机相，即PWM1 -> A相， PWM2 -> B相，PWM3 -> C相。以下代码片段显示了MCLV-2开发板的线性映射。

```
#define BSP_MOTOR1_PHASEA_PWMCHANNEL    PWM_CHANNEL1
#define BSP_MOTOR1_PHASEB_PWMCHANNEL    PWM_CHANNEL2
#define BSP_MOTOR1_PHASEC_PWMCHANNEL    PWM_CHANNEL3
```

对于PWM通道映射到电机相的定制电路板（如图3所示），应用程序接口映射如下所示。

```
#define BSP_MOTOR1_PHASEA_PWMCHANNEL    PWM_CHANNEL2
#define BSP_MOTOR1_PHASEB_PWMCHANNEL    PWM_CHANNEL1
#define BSP_MOTOR1_PHASEC_PWMCHANNEL    PWM_CHANNEL3
```

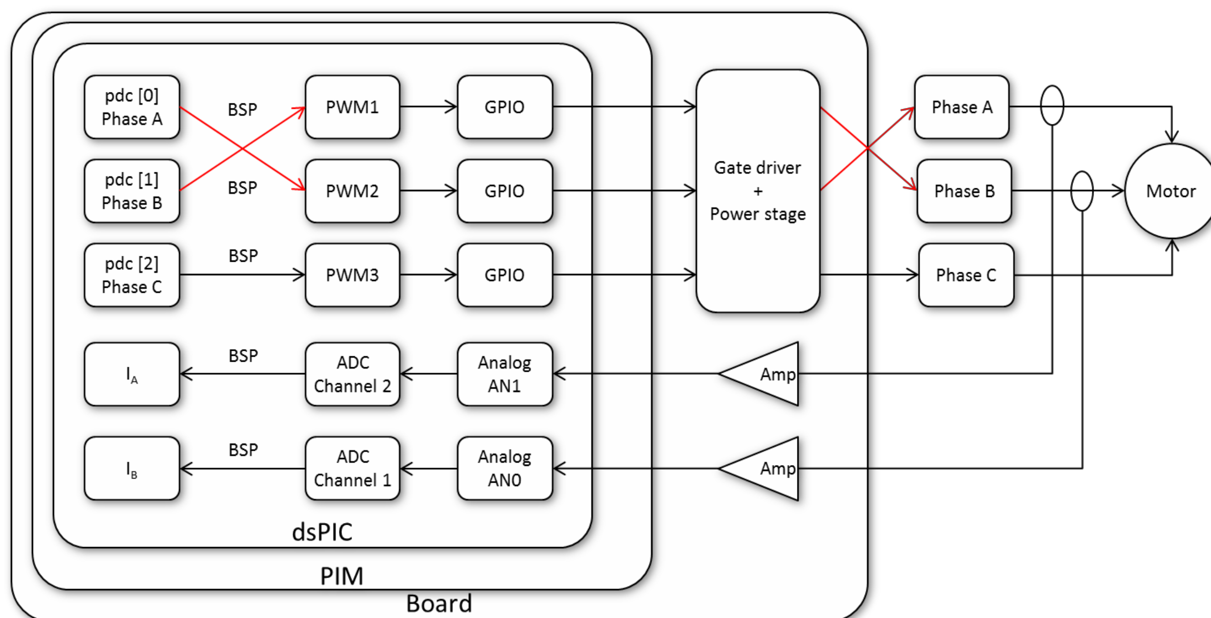

图3：PWM通道交叉映射到电机相的定制板

1.2.3.3 BSP接口摘要

图4显示了BSP的顶层接口图。

© 2018 Microchip Technology Inc.

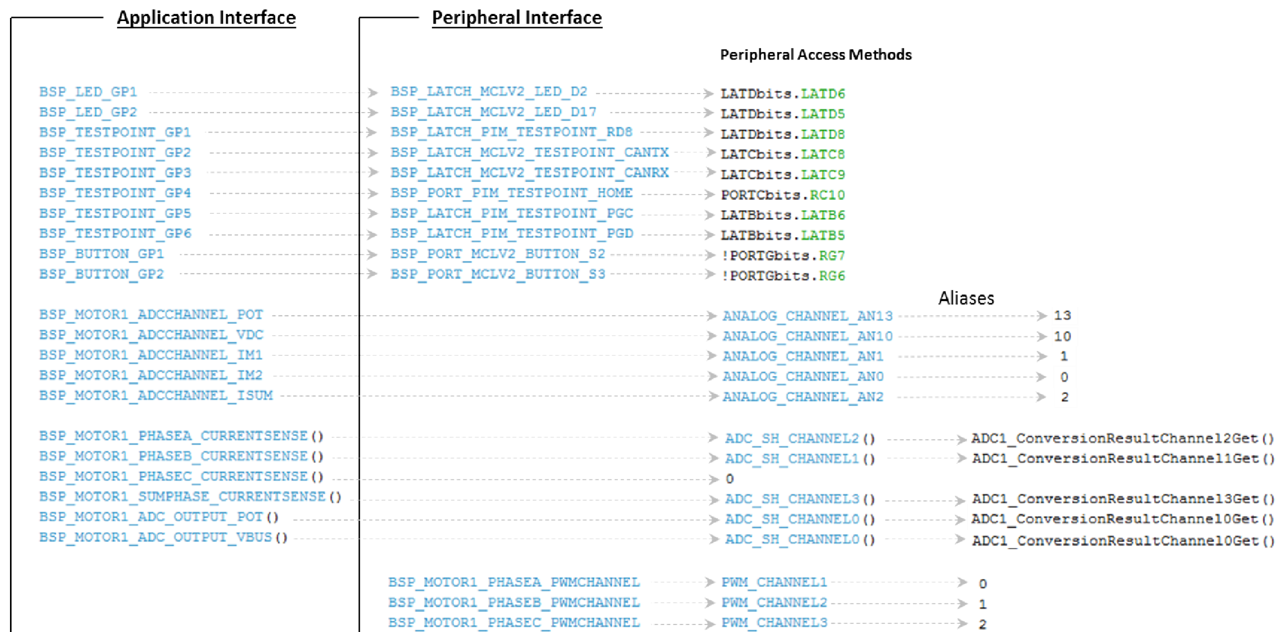

图4：BSP接口摘要

1.3 使用模型

本节介绍了硬件抽象层的使用模型。

- Microchip电机控制开发板及其兼容的PIM将通过预定义的硬件抽象层文件获得开箱即用支持
- 每个Microchip电路板和PIM组合都会有*一组*硬件抽象层文件
- 最终用户将必须为其最终应用程序/开发板创建硬件抽象层文件。下一节介绍了为任何给定的非Microchip电路板创建硬件抽象层文件所需的步骤。

1.3.1 为新电路板创建硬件抽象层文件

以下是为新电路板创建硬件抽象层文件的推荐步骤：

1. 板级支持包
    a. 从Microchip开发的BSP文件***bsp.h***开始
    b. 更新BSP的外设接口部分，以重新定义硬件映射。重命名现有的宏以匹配硬件上的信号标签。如果需要，定义新的宏以访问尚未涉及的硬件IO
    c. 更新BSP的应用程序接口部分，以重新定义硬件的行为映射。如果需要，定义新的宏以访问尚未涉及的硬件IO；但*不要重命名已经定义的宏*
2. 引脚管理器
    a. 从Microchip开发的引脚管理器文件***pin_manager.c***开始

© 2018 Microchip Technology Inc.

   b. 更新`PIN_MANAGER_Initialize()`，确保应用程序/框架所需的所有GPIO都已正确设置，并且其重映射配置已正确分配
3. 外设驱动程序
   a. 从一组Microchip开发的外设驱动程序文件开始
   b. 更新驱动程序初始化函数`XYZ_Initialize()`，按照应用程序/框架的需要初始化外设
4. 系统驱动程序
   a. 从Microchip开发的系统驱动程序文件***mcc.c***和***mcc.h***开始
   b. 根据应用程序/框架的需要更新器件配置位
   c. 根据需要更新`OSCILLATOR_Initialize()`

1.3.2 在应用程序中使用硬件抽象层

要在应用程序中使用硬件抽象层函数，请按照下列步骤操作：

1. 将硬件抽象层文件添加到MPLAB® X项目中。
2. 将***mcc.h***文件包含在引用硬件抽象层函数的所有应用程序源/头文件中。
3. 在main()中的器件初始化阶段，调用`SYSTEM_Initialize()`函数来初始化振荡器、ADC和PWM等器件外设。除此之外，还可以在稍后的时间从应用程序调用各个外设驱动程序的初始化函数。
4. 当应用程序需要访问GPIO时，请使用通过***bsp.h***文件中的BSP应用程序接口定义的宏名称。
5. 当应用程序需要访问器件外设特性时，请使用***hardware_access_functions.h***文件中定义的硬件访问函数和/或相应器件驱动程序文件中定义的外设驱动程序函数以及***bsp.h***文件中定义的BSP。
6. 使用***hardware_access_functions.h***文件中包含的ISR和陷阱助手宏来定义ISR和陷阱函数。使用`void _attribute_((interrupt))`实现此目的，并根据应用程序的需要将`auto_psv`或`no_auto_psv`属性包含在函数定义中。

有关外设驱动程序函数和硬件访问函数的更多详细信息记录在API文档（单独的文档）中。

1.3.3 MC应用程序框架中的硬件抽象层应用

本节举例概述了硬件抽象层在MC应用程序框架中的应用。

注：本节中引用的MC应用程序框架源代码可能已过时。请使用motorBench™开发工具来获取最新版本的MC应用程序框架。

1.3.3.1 系统函数

在下面显示的应用程序代码片段中，`MCAPP_SystemInit()`正在调用硬件抽象层函数`SYSTEM_Initialize()`以初始化器件外设。

© 2018 Microchip Technology Inc.

```
void MCAPP_SystemInit(MCAPP_SYSTEM_DATA *psys)
{
    psys->debugCounters.reset = ++MCAPP_resetCounter;

    SYSTEM_Initialize();
    MCAPP_ConfigurationPwmUpdate();
    MCAPP_InterruptPriorityConfigure();
    ADC1_ModuleEnable();

    MCAPP_DiagnosticsInit();
}
```

1.3.3.2 引脚管理器

硬件抽象层的引脚管理器函数`PIN_MANAGER_Initialize()`由应用程序框架使用`SYSTEM_Initialize()`函数间接访问。硬件抽象层引脚管理器执行以下任务：

1. 根据应用程序的需要初始化LATx和TRISx寄存器
2. 配置PPS设置以将重映射IO分配给相应的器件外设

1.3.3.3 ADC模块

1.3.3.3.1 电位器

以下应用程序代码片段从ADC缓冲区0读取电位器输入，并根据换算系数计算给定速度。

```
uint16_t unipolarADCResult = ADC1_ConversionResultChannel0Get() + 0x8000;
pmotor->velocityControl.velocityCmd = MCAPP_DetermineVelocityCommand(pmotor,
unipolarADCResult);
```

1.3.3.3.2 V_{BUS}检测

以下应用程序代码片段从ADC缓冲区0读取V_{BUS}检测输入并计算V_{BUS}的换算值。

```
uint16_t unipolarADCResult = ADC1_ConversionResultChannel0Get() + 0x8000;
pmotor->psys->vDC = unipolarADCResult >> 1;
```

1.3.3.3.3 ADC通道切换

以下应用程序代码片段将切换ADC采样保持通道0以读取连接到电位器的模拟输入（如BSP中所定义）。

```
ADC1_ChannelSelect(BSP_MOTOR1_ADCCHANNEL_POT);
```

以上代码将ADC1输入通道0选择寄存器（AD1CHS0）设置为由宏`BSP_MOTOR1_ADCCHANNEL_POT`定义的数字。

1.3.3.3.4 ADC中断服务程序

以下应用程序代码片段使用ADC外设驱动程序来实现ADC中断服务程序。

```
ADC1_ISR_FUNCTION_HEADER(void)
```


© 2018 Microchip Technology Inc.

```
{
    BSP_TestpointGp1Activate();
#ifdef MCAPP_TEST_PROFILING
    motor.testing.timestampReference = TMR1_Counter16BitGet();
#endif
    ADC1_FlagInterruptClear();
    MCAPP_SystemStateMachine_StepIsr(&motor);
    MCAPP_UiStepIsr(&motor.ui);
    MCAPP_MonitorStepIsr(&motor);
    MCAPP_WatchdogManageIsr(&watchdog);

    /* Test and diagnostics code are always the lowest-priority routine
within
     * this ISR; diagnostics code should always be last.
     */
    MCAPP_TestHarnessStepIsr(&systemData.testing
    ); capture_timestamp(&motor.testing, 6);
    MCAPP_DiagnosticsStepIsr();
    capture_timestamp(&motor.testing, 7);
    BSP_TestpointGp1Deactivate();
}
```

其中，ADC1中断服务程序的函数头和ADC1中断标志访问宏由***hardware_access_functions.h***文件中的硬件抽象层定义。

1.3.3.3.5 相电流

以下应用程序代码片段使用模拟接口来读取电机相电流。

```
void MCAPP_FocReadADC(MCAPP_MOTOR_DATA *pmotor)
{
    pmotor->iabc.a = BSP_MOTOR1_PHASEA_CURRENTSENSE();
    pmotor->iabc.b = BSP_MOTOR1_PHASEB_CURRENTSENSE();
```

1.3.3.4 PWM模块

1.3.3.4.1 PWM故障处理

以下应用程序代码片段使用PWM外设驱动程序函数来实现PWM故障处理程序。

```
inline static bool MCAPP_OvercurrentHWDetect(void)
{
    return PWM1_FaultStatusGet();
}
```

1.3.3.5 QEI模块

1.3.3.5.1 模式初始化

以下应用程序代码片段将QEI1初始化为以模计数模式运行。

```
void MCAPP_InitQEI(MCAPP_QEI_ESTIMATOR_T *pqei)
{
    QEI1_Initialize();
    QEI1_ModuloMode16bitSet(ENCODER_COUNTS_PER_REV);
    QEI1_ModuleEnable();
```


© 2018 Microchip Technology Inc.

```
        MCAPP_TrackingLoopInit(&pqei->trackingLoop);
```

1.3.3.5.2 写位置

以下应用程序代码片段预设了QEI位置值，以校正QEI角度偏移。

```
void MCAPP_AlignQEI(MCAPP_QEI_ESTIMATOR_T *pqei)
{
    /* Reset QEI position count to the motor zero angle */
    QEI1_PositionCountWrite(MCAPP_QEI_ALIGN_COUNT);
}
```

1.3.3.6 硬件访问函数

以下应用程序代码片段将三相占空比值限制为MIN_DUTY并使用硬件抽象层将其写入到电机1的相应PWM通道。

```
inline void MCAPP_MotorControllerOnActiveStates(MCAPP_MOTOR_DATA *pmotor)
{
    MCAPP_FocStepIsrForwardPath(pmotor);

    {
        uint16_t pwmDutyCycle[3];

        pwmDutyCycle[0] = UTIL_LimitMinimumU16(pmotor->pwmDutycycle.dutycycle1,
MIN_DUTY);
        pwmDutyCycle[1] = UTIL_LimitMinimumU16(pmotor->pwmDutycycle.dutycycle2,
MIN_DUTY);
        pwmDutyCycle[2] = UTIL_LimitMinimumU16(pmotor->pwmDutycycle.dutycycle3,
MIN_DUTY);
        HAL_PwmSetDutyCycles_Motor1(pwmDutyCycle);
    }
}
```

硬件访问函数HAL_PwmSetDutyCyclesMotor1()获取pwm_dutycycle中的三相占空比值并将其写入BSP中定义的PWM通道。

© 2018 Microchip Technology Inc.

1.4 版本历史

版本	日期	作者	说明
1	2017年2月24日	Srikar Deshmukh - C14317	本文档的初始版本

© 2018 Microchip Technology Inc.