前 言

**一、TPS08U的诞生**

在远距离、复杂的工业现场应用场合常常使用4-20mA通信以减轻各种干扰源的影响，市面上有较多的传感器设备或执行器已集成了4-20mA通信功能，但如果4-20mA通信携带的是一个高精度、数据范围比较宽的数据时，接收模块精度达不到0.1%，则会引起数据传输误差，发挥不了传感器性能。TPS08U是就考虑了各种因素而研发出的一款带有隔离功能的高精度模拟量采集模块。

**二、存在的问题**

诚然，市面上已经存在与TPS08U模块相似的标准信号采集模块，但这些模块实质很难快速应用到实际项目中。这是因为，一个好的产品，不仅仅是一系列硬件的堆叠，还需要优质软件的密切配合。而这正是市场所缺少的，使用某个模块往往需要花费大量的精力阅读数据手册，了解底层指令、操作方法，再针对特定的系统（自有系统、Linux、FreeRTOS……）编程，即使模块的厂商提供了一些Demo资料，由于可移植性的问题，往往也还是需要花费大量的时间移植、测试、验证。

为了便于用户设计与开发，ZLG提供了相应的软件包，用户可以直接基于软件包开发应用程序，软件包与具体平台无关，用户可以方便的嵌入到自己的系统中，此外，ZLG已经适配了AWorks、AMetal、Linux等常用平台，若用户在这些系统中开发应用程序，则不需要关心任何底层细节，直接基于API编程即可。

实际开发中，要设计出优质的软件并非易事，还涉及到一些细节问题（如中断的处理等），因此，在软件包的基础上，还进一步提供了本编程指南，除了介绍各个API的功能和使用方法外，还详尽的介绍了一些编程中可能遇到的问题，以指导用户编程。

**三、本书目的**

本编程指南旨在为用户提供编程指导，书中列举了程序范例，使用户可以尽可能充分的理解TPS08U的各种功能以及相应API的使用方法，快速上手，设计并开发出稳定可靠的应用程序。

**四、面向对象**

本书主要为使用TPS08U的软件工程师编写，也可作为了解TPS08U模块的阅读资料。此外，书中讲解了部分与TPS08U无关的跨平台通用接口，展示了一般专用芯片（模块）的软件设计方法，因而也可作为一般的软件读物，以了解一些编程方法。

周立功

2019年8月30日

目 录

[第1章 TPS08U简介 1](#_Toc18414890)

[1.1 产品简介 1](#_Toc18414891)

[1.2 特点 1](#_Toc18414892)

[1.3 引脚说明 1](#_Toc18414893)

[1.4 典型应用电路 2](#_Toc18414894)

[第2章 TPS08U模块功能介绍 3](#_Toc18414895)

[2.1 模块特点 3](#_Toc18414896)

[2.2 基本参数配置 3](#_Toc18414897)

[2.2.1 通道使能寄存器配置 3](#_Toc18414898)

[2.2.2 通道模式寄存器配置 4](#_Toc18414899)

[第3章 TPS08U通用驱动软件包 5](#_Toc18414900)

[3.1 软件包获取 5](#_Toc18414901)

[3.2 软件包结构 5](#_Toc18414902)

[3.3 软件包适配 6](#_Toc18414903)

[3.3.1 类型适配 7](#_Toc18414904)

[3.3.2 宏定义 7](#_Toc18414905)

[3.3.3 工具函数 8](#_Toc18414906)

[3.3.4 其他平台相关函数 8](#_Toc18414907)

[3.4 TPS08U的初始化 12](#_Toc18414908)

[3.5 功能接口 14](#_Toc18414909)

[3.5.1 数据接口 14](#_Toc18414910)

[3.5.2 TPS08U的解初始化 18](#_Toc18414911)

[3.6 典型应用范例 19](#_Toc18414912)

[第4章 在AMetal中使用TPS08U 21](#_Toc18414913)

[4.1 实例初始化函数 21](#_Toc18414914)

[4.2 实例解初始化函数 21](#_Toc18414915)

[4.3 配置文件 21](#_Toc18414916)

[4.4 应用 23](#_Toc18414917)

[第5章 在AWorks中使用TPS08U 25](#_Toc18414918)

[5.1 设备使能 25](#_Toc18414919)

[5.2 设备配置 25](#_Toc18414920)

[5.3 使用TPS08U通用软件包接口 29](#_Toc18414921)

# TPS08U简介

**✍本章导读**

TPS08U模块是广州致远电子有限公司研发的信号调理模块，集ADC、信号调理电路于一体，可以实现高精度的电压和电流采集，可用于4~20mA/0~5V工业标准传感器输出信号。本章将对TPS08U模块做简单介绍。

## 产品简介

TPS08U是一款测量标准工业信号的8通道数据采集模块，可同时测量电压电流信号。电压量程0~5V，满量程精度0.1%±5mV，电流量程4~20mA，整个量程内读数精度0.1%。TPS08U模块在电压电流同时测量的情况下，通道模式最大配置比例为6:1。模块采用sigma-delta型24bit ADC，抗干扰能力较强，且拥有较好的分辨率。TPS08U数据输出速率为12.5sps，每个通道均分数据输出速率，即若只用两个通道，则每个通道的输出速率为6.25sps。通信采用SPI总线方式，与微控制器的接口仅需四根信号线。

## 特点

* 标准工业信号采集；
* 每两个通道为一组，每组通道可自由配置为电压通道或电流通道；
* 电压量程：0~5V，满量程精度0.1%±5mV；
* 电流量程：4~20mA，读数精度0.1%；
* 接触静电4KV；
* 空气静电8KV；
* 隔离耐压2500V AC；
* 工作温度：-40℃~+85℃；
* 与微控制器之间采用SPI串行总线接口进行通信，只需四根信号线，节省I/O资源。

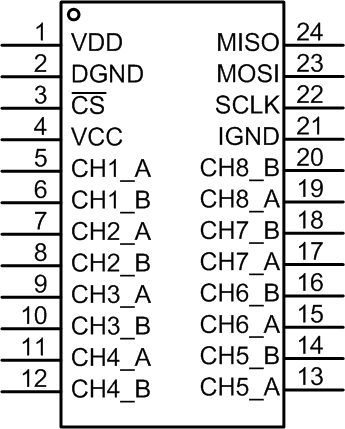


图1.1 TPS08U引脚分布图

## 引脚说明

TPS08U的引脚排列如图1.1所示。

TPS08U标准信号采集模块引脚功能说明如表1.1所示。

表1.1 TPS08U模块引脚说明

|  |  |  |
| --- | --- | --- |
| 引脚序号 | 引脚名称 | 功能描述 |
| 1 | VDD | 3.3V电源输入 |
| 2 | DGND | 数字地 |
| 3 | CS | 片选，低电平有效 |
| 4 | VCC | 5.2V电源输出 |
| 5、7、9、11、13、15、17、19 | CH1\_A ~ CH8\_A | 通道输入信号高电位引脚 |
| 6、8、10、12、14、16、18、20 | CH1\_B ~ CH8\_B | 通道输入信号低电位引脚 |
| 21 | IGND | 模拟通道的地，与DGND隔离 |
| 22 | SCLK | SPI通信时钟 |
| 23 | MOSI | SPI数据通信引脚 |
| 24 | MISO | SPI数据通信引脚 |

## 典型应用电路

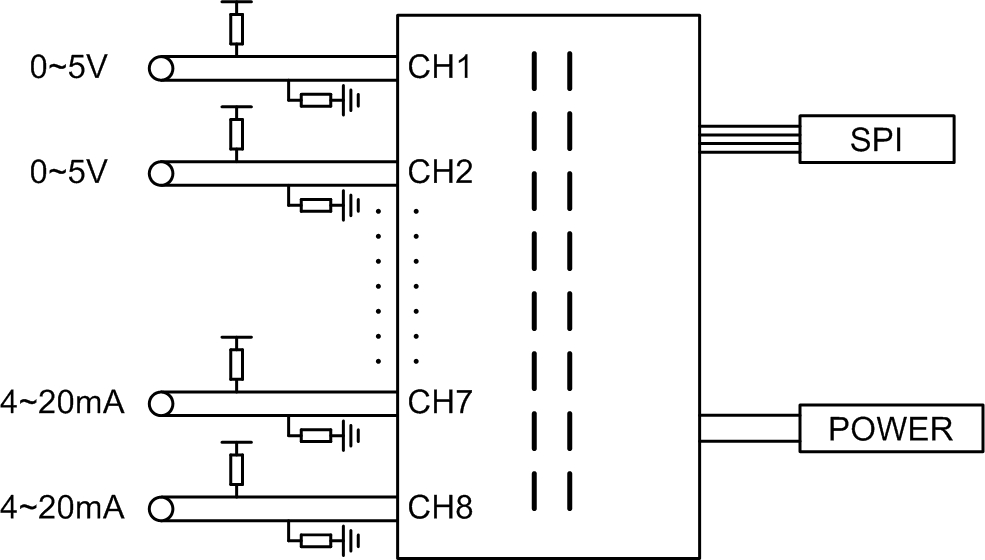


图1.2 TPS08U最小系统应用

TPS08U典型应用电路如图1.2 所示。由于标准信号的接入通道属于悬浮状态，所以通道上必须要有上下拉电阻，用来提供标准信号的参考电位。上下拉电阻均使用1MR电阻，电源和地引脚均从模块引脚引出。同时，由于共模电位抬高，提升了通道的抗干扰能力。

# TPS08U模块功能介绍

**✍本章导读**

TPS08U克服了ADC采样电阻温飘、基准噪声、电源噪声和PCB布局等影响转换精度的因素，解决了ADC失调误差、增益误差、积分非线性差等问题。本章将对TPS08U模块采样通道相关配置进行介绍。

## 模块特点

* **通道多**

该模块支持8个通道，每个通道有单独的使能开关。通道采集的信号类型（电压或电流）可以按组单独配置。8个通道分为4组：CH1与CH2、CH3与CH4、CH5与CH6、CH7与CH8，每组内的两个通道采集的信号类型是相同的。

* **抗干扰能力强**

该模块采用极低温飘的电阻，号称零漂移的运放，集成了电源及通信隔离电路，并优化了layout走线布局。

* **数据精度高**

该模块以24位有符号定点数存储数据，第23位为符号位，17~22位为整数部分，0~16位为小数部分，小数精度为1/217，约为0.0000076。负数以补码形式存放。通道配置为电压模式时，单位为V；通道配置为电流模式时，单位为mA。

## 基本参数配置

### 通道使能寄存器配置

TPS08U 共有8 个通道，用户可以自由配置每个通道是失能或者使能。失能的通道不进行采样，所以关闭不使用的通道可以加快数据更新速率。通道使能寄存器的每一位代表1 个通道，0 表示失能通道，1 表示使能通道。具体定义如表2.1所示：

表2.1 通道使能配置

|  |  |
| --- | --- |
| 位 | 该位对应的通道号 |
| 0 | CH1 |
| 1 | CH2 |
| 2 | CH3 |
| 3 | CH4 |
| 4 | CH5 |
| 5 | CH6 |
| 6 | CH7 |
| 7 | CH8 |

例如，使能CH1~CH5时，可配置通道使能寄存器为0x1F。

### 通道模式寄存器配置

TPS08U 共有8 个通道，用户可以通过通道模式寄存器配置通道为电压模式或电流模式。每一位代表2 个通道，0 表示电压通道，1 表示电流通道。具体定义如表2.2所示：

表2.2 通道使能配置

|  |  |  |
| --- | --- | --- |
| 位 | 值 | 通道模式 |
| 0 | 0 | V1、V2 |
| 1 | C1、C2 |
| 1 | 0 | V3、V4 |
| 1 | C3、C4 |
| 2 | 0 | V5、V6 |
| 1 | C5、C6 |
| 3 | 0 | V7、V8 |
| 1 | C7、C8 |

例如，将CH1~CH4配置为电压通道，CH5~CH8配置为电流通道，可配置通道模式寄存器为0xC。

# TPS08U通用驱动软件包

**✍本章导读**

为了便于用户快速将TPS08U应用到实际项目中，避免浪费大量的时间和精力去阅读、学习TPS08U模块指令，特提供了TPS08U通用驱动包，用户可以通过该软件包提供的接口快速开发应用项目，只需简单的与当前开发平台适配，便可实现软件的跨平台复用。本章将对如何使用该软件包作详细的介绍。

在 3.3 和 3.4 节中，详细介绍了在不同平台中的适配方法以及 handle 的获取方法（TPS08U初始化）。这部分内容主要针对用户自有的特殊平台，若用户使用 AMetal、AWorks 等已经完成适配的平台，则可以跳过 3.3 和 3.4 节。直接阅读 3.5节中介绍的各个功能接口以及平台相应的章节（例如：AMetal 对应第 4 章）。

## 软件包获取

TPS08U通用驱动软件包（为便于描述，后文将TPS08U通用驱动软件包简称为软件包）提供了一系列接口，实现了对TPS08U模块各个接口的封装，使用户调用这些接口就可以快捷的操作TPS08U模块，同时，这些接口与具体平台或硬件无关，是“通用”的，换句话说，可以在Linux中使用，也可以在AWorks中使用，还可以在AMetal中使用。

该软件包存于TPS08U资料集中，可在立功科技官网（http://www.zlgmcu.com/）下载（或联系立功科技相关区域销售获取）。

通用驱动包中主要包含两个文件夹：driver 和 examples。driver 目录下存放了驱动相关的文件；examples下存放了一些应用示例代码（基于驱动提供的功能接口编写的）。各文件简要说明详见表3.1。

表3.1 软件包文件说明

|  |  |  |
| --- | --- | --- |
| 目录 | 文件名 | 功能描述 |
| driver | tps08u.h | 各功能接口的声明，编写应用程序时需查看 |
| tps08u.c | 各功能接口的实现，用户一般无需关心具体内容 |
| tps08u\_platform.h | 与平台相关的接口声明、类型定义等 |
| tps08u\_platform.c | 平台相关接口的实现，根据平台具体实现 |
| examples | app\_tps08u.h | 应用程序 demo 的入口函数声明 |
| demo\_tps08u.c | TPS08U简单信号采集测试demo |

driver目录下存放的文件是驱动包的核心。其中。tps08u.c和tps08u.h中包含了TPS08U功能接口的申明和实现，对于应用来说主要是基于tps08u.h文件中的接口编程；tps08u\_platform.h和tps08u\_platform.c文件中的内容与具体平台相关，不同的平台其实现可能存在差异，在3.3节中会详细介绍平台适配的方法，若用户在已适配好的的平台使用软件包，则可无需关心这两个文件。

## 软件包结构

软件包基本结构图详见图3.1。

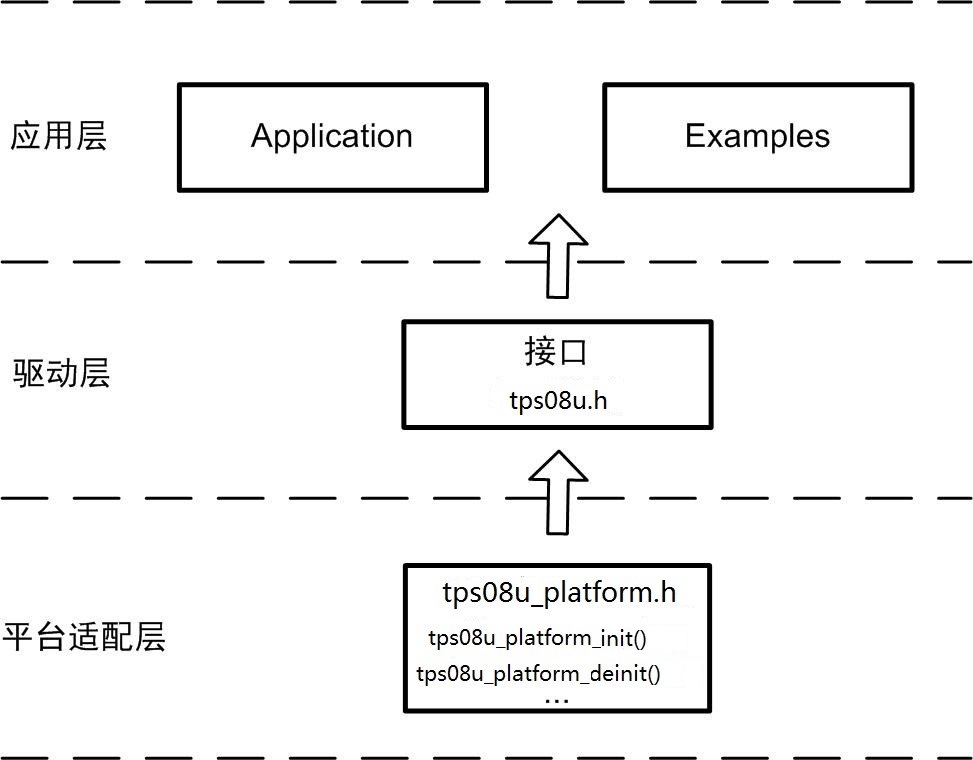


图3.1 软件包基本结构

可见，可以将软件包看做三层结构：应用层、驱动层和平台适配层。

* 应用层

应用层主要包括两部分：用户应用和 demo 程序。它们都基于tps08u.h 文件中的功能接口编程（这些接口在 3.5 节中详细介绍）。后文会提到，在使用这些接口时，需要传递一个 handle（句柄）作为参数的值。

需要注意的是，AMetal、AWorks、Linux等平台已经完成了对驱动包的适配，因此，这些平台均已提供了获取 handle 的方法。 若用户在这些平台中使用 TPS08U，则可以查看对应章节的内容，了解获取 handle 的具体方法（通常，仅需调用一个函数即可），获取到 handle后，直接使用 3.5 节中介绍的接口操作 TPS08U即可。

* 驱动层

驱动层包含 tps8u.h 和 tps08u.c 文件，它们实现了TPS08U核心功能的封装，其中部分与平台相关的功能，可能需要调用平台适配层的接口。

* 平台适配层

平台适配层位于最底层，其中包含tps08u\_platform.h和tps08u\_platform.c文件，这部分代码接口固定（需要被驱动层调用），但其具体实现与平台相关。

在实际情况中，平台适配和基于TPS08U应用编程可能是相同的人，但两者的侧重点不同：平台适配工作主要完成平台相关的适配，位于最底层，代码与平台息息相关；而应用编程位于最上层，代码与平台无关。因此，为了便于区分，在后文的叙述中，将完成平台适配工作的用户称之为“平台适配者”，而基于接口编写应用程序的一般用户称之为“用户”。

## 软件包适配

通用软件包的存在，使得用户完全不需要关心TPS08U操作指令的内部细节，降低了用户使用TPS08U模块的难度，使用户可以更快的将TPS08U应用到实际项目中。由主控MCU与TPS08U之间的连接可知（详见图1.2），主控MCU与TPS08U之间主要通过SPI连接。由于在不同的软件环境（Linux、AWorks、AMetal、uc/OS、rt-thread或用户使用的其它平台）中，SPI的控制方法不尽相同，因此，在使用软件包提供的各个功能接口之前，必须根据具体平台完成对软件包的适配，实现相应的SPI数据传输控制方法。

对于大部分MCU或平台来说，都具有SPI设备，甚至很多平台都会提供SPI的操作接口，这样一来适配工作也会变得十分简洁。实际上，本节讲解的适配方法主要是针对部分用户适配自有平台而言的，ZLG已经对常用平台（AWorks、AMetal以及Linux等）进行了适配。若用户选用这些平台进行应用程序的开发，则可以跳过本小节，直接使用下节介绍的各个接口操作TPS08U模块，非常便捷。

对于软件包适配，只需关心适配文件tps08u\_platform.h，所有需要适配的接口均在该文件中申明，主要包含了4个部分：类型适配、宏定义、工具函数适配以及其他平台相关函数。

### 类型适配

在 tps08u\_platform.h 文件中，使用 typedef 定义了以下几个类型：

1. uint32\_t ：32位无符号整数类型
2. uint16\_t ：16位无符号整数类型
3. uint8\_t ： 8位无符号整数类型
4. int8\_t ： 8位有符号整数类型
5. tps08u\_platform\_devinfo\_t ：平台信息类型
6. tps08u\_platform\_dev\_t ：平台设备类型

它们的默认定义如下

typedef unsigned int uint32\_t;

typedef unsigned short uint16\_t;

typedef unsigned char uint8\_t;

typedef char int8\_t;

typedef void\* tps08u\_platform\_devinfo\_t;

typedef void\* tps08u\_platform\_dev\_t;

对于前4个数据类型，用户可以根据实际平台进行调整。若用户所处平台已经存在这

些类型的定义，则可以删除或注释掉 typedef 定义的语句，直接包含相应头文件即可。

对于平台信息类型（tps08u\_platform\_devinfo\_t）和平台设备类型（tps08u\_platform\_dev\_t）可能用户暂时还不太理解，这两个类型的具体含义和用法在后文介绍平台初始化函数时，会根据实际适配过程进行详解。

### 宏定义

在TPS08U的驱动中，使用了NULL 表示空指针，因此，平台必须确保 NULL 被有效

定义：

#ifndef NULL

#define NULL ((void \*)0)

#endif

绝大部分情况下，NULL 都采用这种方式定义，因此，一般无需对该部分内容进行修改。

### 工具函数

#### 延时函数

TPS08U驱动或 example 程序，可能需要使用“延时一段时间”的功能，不同平台可能具有不同的延时方法，因此，需要由平台根据实际情况实现一个延时函数。其函数申明为：

void tps08u\_platform\_mdelay (uint32\_t ms);

由此可见，该函数无返回值，仅有一个表示延时时间（单位：毫秒）的参数。在 tps08u\_platform.c 文件中，提供了一个简单的实现范例，详见程序清单3.1。

程序清单3.1 延时函数默认实现

1 void tps08u\_platform\_mdelay (uint32\_t ms)

2 {

3 volatile uint32\_t i;

4 while (ms--) {

5 for (i = 0; i < 5000; i++); //此处需根据具体CPU频率进行修改

6 }

该延时函数用以提供粗略的毫秒级别延时，测试环境：Cortex-M3，主频96MHz，若需要精准的延时，用户可基于定时器实现。

在特定的系统中，可能已经存在相应的延时函数，例如，在 AMetal 平台中，提供了am\_mdelay()函数，用于 ms 级别的延时，则该函数可以直接基于am\_mdelay()函数实现，详见程序清单3.2。

程序清单3.2 延时函数在AMetal中实现

1 void tps08u\_platform\_mdelay (uint32\_t ms)

2 {

3 am\_mdelay(ms);

4 }

### 其他平台相关函数

这类函数与TPS08U模块驱动密切相关，也是整个适配过程的关键。主要包括5个函数，均在tps08u\_platform.h文件中申明，主要适配接口详见表3.2。

表3.2 软件包适配相关接口

|  |  |
| --- | --- |
| 函数原型 | 功能简介 |
| uint8\_t tps08u\_platform\_init(tps08u\_platform\_dev\_t \* p\_dev,  const tps08u\_platform\_devinfo\_t \*p\_info); | 平台初始化函数 |
| uint8\_t tps08u\_platform\_deinit(tps08u\_platform\_dev\_t \* p\_dev); | 平台解初始化函数 |
| int tps08u\_platform\_write(tps08u\_platform\_dev\_t \*p\_dev,  const uint8\_t \*p\_txbuf0,  uint32\_t n\_tx0,  const void \*p\_txbuf1,  uint32\_t n\_tx1) | SPI写函数 |
| int tps08u\_platform\_read (tps08u\_platform\_dev\_t \*p\_dev,  const uint8\_t \*p\_txbuf,  uint32\_t n\_tx,  void \*p\_rxbuf,  uint32\_t n\_rx) | SPI读函数 |

这些接口在 tps08u\_platform.c 文件中实现。在默认的 tps08u\_platform.c 文件中，均为未实现的空函数，需要用户根据实际情况适配。

观察各个接口可以发现，所有函数的第一个参数均为 tps08u\_platform\_dev\_t \* 类型的 p\_dev，这里使用了面向对象编程的思想：将平台看作一个对象， 基于平台相关的所有数据均保存在了 tps08u\_platform\_dev\_t 类型的平台设备中，通过该指针即可访问平台设备中的数据，以便进一步实现平台相关的功能。接下来 ，将详细介绍这几个接口的实现方法，以及tps08u\_platform\_dev\_t 类型的定义。

#### 平台初始化函数

该函数需要申请相关硬件资源，完成对所有平台相关的硬件设备的初始化，如GPIO、SPI的初始化等等。则这些资源相关的初始化操作都需要在平台初始化函数中完成。该函数会在后文介绍的 tps08u\_init()函数中被首先调用。

平台初始化函数原型为：

uint8\_t tps08u\_platform\_init(tps08u\_platform\_dev\_t \*p\_dev,

const tps08u\_platform\_devinfo\_t \*p\_info);

下面对其参数做详细介绍。

* p\_dev：

该参数为指向平台设备实例的指针，其类型的具体定义与平台相关，典型的，其可以定义为一个结构体类型，其中包含一些成员，用以保存平台函数中需要保存的状态。例如，参数 p\_info 是一个指向平台信息的指针（该指针的具体用法在后文介绍 p\_info 参数时详细介绍），在平台相关程序运行时可能需要使用该信息，可将其定义在tps08u\_platform\_dev\_t中：

typedef struct tps08u\_platform\_dev{

//其它自行定义需要的变量

const tps08u\_platform\_devinfo\_t \*p\_devinfo;

} tps08u\_platform\_dev\_t;

该类型完全由平台适配者定义，其决定了用户在该平台下使用TPS08U时，需要传递的一些与平台相关的基础信息，而TPS08U通用驱动将不会对该类型成员做任何操作。通常情况下，可以遵循这样一个原则：当期望平台相关的函数在运行结束后，保存一些数据，以便后续再使用，则可以在 tps08u\_platform\_dev\_t 类型中定义相应的成员，需要保存数据时，为该成员赋值即可。当然，用户可选择不使用该类型，可将其定义为：

typedef void\* tps08u\_platform\_dev\_t;

TPS08U驱动包会负责为tps08u\_platform\_dev\_t类型的变量分配内存空间，在调用平台相关函数时，将其地址作为各个函数的第一个参数传递，因此，适配者无需担心 p\_dev是否会是一个野指针（指向非法的地址空间），可以放心的使用该指针，访问其中的成员。

p\_dev中所有成员的初始化都应该在平台初始化函数中完成，范例程序详见程序清单3.3。

程序清单3.3 平台初始化函数实现范例

1 uint8\_t tps08u\_platform\_init(tps08u\_platform\_dev\_t \*p\_dev,

2 const tps08u\_platform\_devinfo\_t \*p\_info)

3 {

4 if(p\_dev == NULL || p\_info == NULL){

5 return 1;

6 }

7 p\_dev ->p\_devinfo = p\_info;

8

9 // p\_dev其他成员初始化

10 //硬件资源初始化

11 }

* p\_info：

该参数为指向平台设备信息的指针，与参数p\_dev相似，其类型也是根据用户的需要自行定义，典型的，可以定义为一个结构体类型，其中包含一些固定信息，可以将其定义的实例使用关键字const修饰。例如引脚信息、串口等，在不同平台中，引脚、串口等信息的表示方法可能不同，因而对应的类型可能不同。假定用int8\_t型的引脚编号表示一个引脚，用uint32\_t型的数值表示SPI时钟频率，可定义如下：

typedef struct tps08u\_platform\_devinfo{

int8\_t cs\_pin;

uint32\_t sclk\_freq; /\*\*< \brief SPI 时钟频率 \*/

//其他需要定义的信息

} tps08u\_platform\_devinfo\_t;

同样，该类型完全由平台适配者定义，其决定了用户在该平台下使用TPS08U时，需要传递的一些与平台相关的基础信息，而TPS08U通用驱动将不会对该类型成员做任何操作。

基于此，在平台相关函数中可通过传递的第一个参数p\_dev来访问p\_info，从而找到硬件引脚信息、SPI频率等。当然，用户可以选择不通过这两个参数传递，直接将这些信息定义为全局变量，如此一来，也可不使用tps08u\_platform\_devinfo\_t类型，可以将其定义为：

typedef void\* tps08u\_platform\_devinfo\_t;

#### 平台解初始化函数

在上一小节中介绍了平台初始化函数的作用及其适配方法， 在平台初始化函数中申请了一些硬件资源，如SPI、GPIO以及中断等等，不再需要使用驱动的时候便要对申请的硬件资源进行释放。当TPS08U设备不再被使用时，这些资源应该被相应的释放（或关闭），例如，关闭引脚中断等。 与平台初始化函数相对应，这些操作在平台解初始化函数中完成。平台解初始化函数与平台初始化函数是相对应的，在平台初始化函数中打开、获得了什么资源，就应该在平台解初始化函数中关闭和释放。其函数原型为：

uint8\_t tps08u\_platform\_deinit(tps08u\_platform\_dev\_t \*p\_dev);

其中，参数p\_dev为指向平台设备的指针。

#### SPI写函数

TPS08U模块与主控 MCU 之间采用的是SPI通信。不同平台在实现SPI数据传输时，无需关心通信数据的具体含义。对于用户来讲，通用软件包会提供一系列有意义的接口供用户使用， 当用户调用这些接口时，会将用户对应的操作转换为与TPS08U的通信命令，然后调用平台实现的SPI数据传输函数，将指令传输给TPS08U模块。

SPI写函数的功能为向指定地址写入数据，其原型如下：

int tps08u\_platform\_write(tps08u\_platform\_dev\_t \*p\_dev,

const uint8\_t \*p\_txbuf0,

uint32\_t n\_tx0,

const void \*p\_txbuf1,

uint32\_t n\_tx1);

* p\_dev：

该参数为指向平台设备结构体实例的指针。

* p\_txbuf0：

该参数为指向SPI需要发送的地址信息缓存的指针，SPI需要发送的n\_tx0个字节的数据需要在该指针指向的缓存中获取。

* n\_tx0：

该参数为需要发送的字节数。

* p\_txbuf1：

该参数为指向SPI需要发送数据信息缓存的指针，SPI需要发送的n\_tx1个字节的数据需要在该指针指向的缓存中获取。

* n\_tx1：

该参数为需要发送的字节数。

* 返回值：

其返回值为0表示成功，非0表示发送失败。

#### SPI读函数

SPI读函数的功能为从指定地址读取数据，其原型如下：：

int tps08u\_platform\_read (tps08u\_platform\_dev\_t \*p\_dev,

const uint8\_t \*p\_txbuf,

uint32\_t n\_tx,

void \*p\_rxbuf,

uint32\_t n\_rx);

* p\_dev：

该参数为指向平台设备结构体实例的指针。

* p\_txbuf：

该参数为指向SPI需要发送的地址信息缓存的指针，SPI需要发送的n\_tx个字节的数据需要在该指针指向的缓存中获取。

* n\_tx：

该参数为需要发送的字节数。

* p\_rxbuf：

该参数为指向SPI接收数据信息缓存的指针，SPI需要接收n\_rx1个字节的数据并写入该指针指向的缓存。

* n\_rx：

该参数为需要读取的字节数。

* 返回值：

其返回值为0表示成功，非0表示发送失败。

## TPS08U的初始化

在完成3.3节中的接口适配工作后，还需对TPS08U设备进行初始化，此时需要使用TPS08U通用驱动包tps08u.h中的初始化函数，该函数的作用是完成TPS08U模块的初始化，实际上，该函数中首先便调用了平台初始化函数。所以在使用TPS08U前，其应首先被调用。该函数有一个非常重要的作用是：获取TPS08U设备操作句柄。基于此，该函数通常由驱动适配者直接调用。下面，从接口原型开始，详细介绍该函数的调用方法。

初始化函数的原型为：

tps08u\_handle\_t tps08u\_init (tps08u\_dev\_t \*p\_dev,

const tps08u\_devinfo\_t \*p\_devinfo);

为便于描述，将该函数的介绍分为两大部分：参数和返回值。

#### 参数详解

由初始化函数的原型可知，其有两个参数：p\_dev和p\_devinfo。其中，p\_dev用于为设备运行分配必要的内存空间；p\_devinfo 为保存固定参数的内存空间。下面，分别对各个参数作详细介绍。

* p\_dev

p\_dev的主要作用是为设备运行分配必要的内存空间（用于软件包内部保存一些与TPS08U相关的状态数据等），p\_dev指针指向的即是一段内存空间的首地址。内存大小为tps08u\_dev\_t类型数据占用的大小，基于此，可以先使用tps08u\_dev\_t类型定义一个变量，即：

tps08u\_dev\_t dev;

则dev的地址（&dev）即可作为p\_dev的实参传递。

由于这里仅需要分配一段内存空间，因此，tps08u\_dev\_t类型的具体定义并不需要关心，例如，该类型具体包含哪些数据成员等。

上面仅仅是从p\_dev参数的设置形式上对其作了直观的描述，实际上，可以从“面向对象编程”的角度对其作更深入的理解。在这里，tps08u\_dev\_t类型为TPS08U的设备类型，每个具体的TPS08U硬件设备都可以看作是一个对象，即该类型的具体实例。显然，在使用一个对象前，必须完成对象的定义，或者说实例化，对象需要占用一定的内存空间，在C中，具体的表现形式为使用某个类型定义相应的变量，即：

tps08u\_dev\_t dev;

若有多个TPS08U对象，例如，系统中使用了两个TPS08U模块，则可以使用该类型定义多个对象，即：

tps08u\_dev\_t dev0;

tps08u\_dev\_t dev1;

当存在多个TPS08U对象时，每个TPS08U对象都需要使用初始化函数进行初始化。初始化哪个对象，就将其地址作为初始化函数的p\_dev实参传递。

* p\_devinfo

参数p\_ devinfo的主要作用是保存一些固定信息，这些信息可由用户在编译前进行合理的修改，程序一旦下载完成后便不再修改这些信息，故而可将该参数使用关键字const修饰。p\_devinfo 的类型为：

typedef struct tps08u\_devinfo {

uint8\_t chan\_mode; /\*\*< \brief 通道模式，每位表示一组通道，0 为电压，1 为电流 \*/

tps08u\_platform\_devinfo\_t spi\_devinfo; /\*\*< \brief SPI 从设备信息描述结构 \*/

} tps08u\_devinfo\_t;;

其中，成员spi\_devinfo为SPI平台信息结构体，该成员的作用在3.3小节中已经详细介绍，其类型可由用户自行定义。

成员chan\_mode为通道的工作模式。4位各自分别代表2个通道的工作模式，共8个通道，0为电压模式，1为电流模式。

同p\_dev一样，p\_devinfo实际上也是指向tps08u\_devinfo\_t类型的一段内存地址的指针，因此需要为其分配一段内存空间，这里用tps08u\_devinfo\_t定义相应全局变量：

tps08u\_devinfo\_t devinfo;

与dev不同的是，devinfo是对固定信息的存储，需要用户手动对其成员进行一一赋值，因此，可将devinfo定义并赋初值：

const tps08u\_devinfo\_t devinfo = {

/\* 通道初始化模式 \*/

TPS08U\_MODE\_CH1\_2\_V |

TPS08U\_MODE\_CH3\_4\_V |

TPS08U\_MODE\_CH5\_6\_V |

TPS08U\_MODE\_CH7\_8\_C,

{

28, /\*CS引脚编号\*/

3000000, /\* SPI时钟频率 \*/

//其它参数

}

};

然后可将devinfo的地址（&devinfo）作为初始化函数的第二个参数传递。

#### 返回值

初始化函数的返回值为TPS08U设备句柄，其类型为：tps08u\_handle\_t，该类型的具体定义无需关心，用户唯一需要知道的是，软件包提供的各个功能接口中（详见3.5节中介绍的各个接口），均使用了一个handle参数，用以指定操作的TPS08U设备。用户在调用各个功能接口时，handle参数设置的值即为初始化函数返回的TPS08U实例句柄。

获取TPS08U实例句柄的语句为：

tps08u\_handle\_t handle = tps08u\_init(&dev, &devinfo);

其中，参数dev以及devinfo的值的定义实际是在驱动适配时完成的，对于上层应用来讲，其仅需获取到一个实例句柄，并不需要关心驱动适配的具体细节。基于此，为了避免用户对驱动适配作过多的了解，在适配软件包时，可以直接对外提供一个实例初始化函数，用以完成一个TPS08U硬件设备的初始化，进而获取到相应的handle，范例程序详见程序清单3.4。

程序清单3.4 TPS08U实例初始化函数实现范例

0 /\*\*< \brief TPS08U 设备实例 \*/

1 static tps08u\_dev\_t \_\_g\_tps08u\_dev;

2

3 /\*\*< \brief TPS08U设备信息 \*/

4 static const tps08u\_devinfo\_t \_\_g\_tps08u\_info = {

5 /\* 通道初始化模式 \*/

6 TPS08U\_MODE\_CH1\_2\_V |

7 TPS08U\_MODE\_CH3\_4\_V |

8 TPS08U\_MODE\_CH5\_6\_V |

9 TPS08U\_MODE\_CH7\_8\_C,

10 {

11 28 /\*CS引脚编号\*/

12 3000000,

13 //其它参数

14 }

15 };

16

17 /\*\* \brief TPS08U 实例初始化，获得标准服务句柄 \*/

18 tps08u\_handle\_t tps08u\_inst\_init (void)

19 {

20 return tps08u\_init(&\_\_g\_tps08u\_dev, &\_\_g\_tps08u\_info);

21 }

基于此，用户可以直接调用无参数的实例初始化函数，完成handle的获取，即：

tps08u\_handle\_t handle = tps08u\_inst\_init();

注意，上述代码均为示意性代码，并没有实现任何特定的功能，所有平台的适配均可以此为例，使用户快速获取到TPS08U设备句柄，进而使用驱动包提供的各个接口。

## 功能接口

### 数据接口

TPS08U模块的数据传输接口支持数据透传和非透传模式。为方便使用，这两种数据模式已经统一接口，即可使用一个接口来完成对数据的透明传输或非透明传输。并且考虑复杂情况下软件环境对实时性要求的不同，驱动包分别提供了同步数据接口和异步数据接口。其同步数据传输接口如表3.3所示。

表3.3 同步数据接口

|  |  |
| --- | --- |
| 函数原型 | 功能简介 |
| int8\_t tps08u\_detect (tps08u\_handle\_t handle); | 检测与 TPS08U 通信是否正常 |
| int8\_t tps08u\_chan\_enable\_get (tps08u\_handle\_t handle,  uint8\_t chan,  uint8\_t \*p\_chan\_enable); | 获取 TPS08U 使能的通道 |
| int8\_t tps08u\_chan\_enable\_set (tps08u\_handle\_t handle,  uint8\_t chan,  uint8\_t chan\_enable); | 配置 TPS08U 使能的通道 |
| int8\_t tps08u\_chan\_mode\_get (tps08u\_handle\_t handle,  uint8\_t chan,  uint8\_t \*p\_chan\_mode); | 获取 TPS08U 通道模式 |
| int8\_t tps08u\_chan\_mode\_set (tps08u\_handle\_t handle,  uint8\_t chan,  uint8\_t chan\_mode); | 配置 TPS08U 通道模式 |
| int8\_t tps08u\_chan\_data\_get (tps08u\_handle\_t handle,  uint8\_t chan,  uint32\_t \*p\_chan\_data); | 获取 TPS08U 通道数据 |
| int8\_t tps08u\_state\_get (tps08u\_handle\_t handle, uint8\_t \*p\_state); | 获取 TPS08U 状态 |
| int8\_t tps08u\_reset (tps08u\_handle\_t handle); | 复位 TPS08U |

#### 检测与 TPS08U 通信是否正常

TPS08U通信状况检测的函数原型为：

int8\_t tps08u\_detect (tps08u\_handle\_t handle);

其中，参数handle为TPS08U设备操作句柄。

该函数返回值为0表示通信连接正常，返回值小于0时表明连接异常。

#### 获取 TPS08U 使能的通道

该函数用来获取TPS08U的通道使能状况。该函数原型如下：

int8\_t tps08u\_chan\_enable\_get (tps08u\_handle\_t handle,

uint8\_t chan,

uint8\_t \*p\_chan\_enable);

其中，参数handle为TPS08U设备操作句柄；chan为需要获取的通道号，1~8为具体的单个通道，0为所有通道；p\_chan\_enable为指向保存通道使能状态的缓存的指针。其返回值为0表示获取成功，返回值小于0时表明获取失败。

该接口的使用方法详见程序清单3.5。

程序清单3.5 获取 TPS08U 使能的通道

1 int8\_t err;

2 uint8\_t temp;

3

4 tps08u\_handle\_t handle = tps08u\_inst\_init();

5

6 err = tps08u\_chan\_enable\_get (handle, 0, &temp);

7 if (err < 0) {

8 // 获取失败

9 }

#### 配置 TPS08U 使能的通道

该函数用来配置TPS08U的通道使能状况。该函数原型如下：

int8\_t tps08u\_chan\_enable\_set (tps08u\_handle\_t handle,

uint8\_t chan,

uint8\_t chan\_enable);

其中，参数handle为TPS08U设备操作句柄；chan为需要配置的通道号，1~8为具体的单个通道，0为所有通道；chan\_enable为使能状态。其返回值为0表示配置成功，返回值小于0时表明配置失败。

该接口的使用方法详见程序清单3.6。

程序清单3.6 配置 TPS08U 使能的通道

1 int8\_t err;

2 uint8\_t temp = 0x03; /\* 使能通道1、2 \*/

3

4 tps08u\_handle\_t handle = tps08u\_inst\_init();

5

6 err = tps08u\_chan\_enable\_set (handle, 0, temp);

7 if (err < 0) {

8 // 配置失败

9 }

#### 获取 TPS08U 通道模式

该函数用来获取TPS08U的通道模式。该函数原型如下：

int8\_t tps08u\_chan\_mode\_get (tps08u\_handle\_t handle,

uint8\_t chan,

uint8\_t \*p\_chan\_mode);

其中，参数handle为TPS08U设备操作句柄；chan为需要获取的通道号，1~8为具体的单个通道，0为所有通道；p\_chan\_mode为指向保存通道模式的缓存的指针。其返回值为0表示获取成功，返回值小于0时表明获取失败。

该接口的使用方法详见程序清单3.7。

程序清单3.7 获取 TPS08U 通道模式

1 int8\_t err;

2 uint8\_t mode;

3

4 tps08u\_handle\_t handle = tps08u\_inst\_init();

5

6 err = tps08u\_chan\_mode\_get (handle, 0, &mode);

7 if (err < 0) {

8 // 获取失败

9 }

#### 配置 TPS08U 通道模式

该函数用来获取TPS08U的通道使能状况。该函数原型如下：

int8\_t tps08u\_chan\_mode\_set (tps08u\_handle\_t handle,

uint8\_t chan,

uint8\_t p\_chan\_mode);

其中，参数handle为TPS08U设备操作句柄；chan为需要获取的通道号，1~8为具体的单个通道，0为所有通道；p\_chan\_mode为通道的模式值。其返回值为0表示配置成功，返回值小于0时表明配置失败。

该接口的使用方法详见程序清单3.8。

程序清单3.8 配置 TPS08U 通道模式

1 int8\_t err;

2 uint8\_t mode;

3

4 tps08u\_handle\_t handle = tps08u\_inst\_init();

5

6 err = tps08u\_chan\_mode\_set (handle, 0, mode);

7 if (err < 0) {

8 // 配置失败

9 }

#### 获取 TPS08U 通道数据

该函数用来获取TPS08U的通道使能状况。该函数原型如下：

int8\_t tps08u\_chan\_data\_get (tps08u\_handle\_t handle,

uint8\_t chan,

uint32\_t \*p\_chan\_data);

其中，参数handle为TPS08U设备操作句柄；chan为需要获取的通道号，1~8为具体的单个通道，0为所有通道；p\_chan\_data为指向保存通道使能状态的缓存的指针。其返回值为0表示获取成功，返回值小于0时表明获取失败。

该接口的使用方法详见程序清单3.9。

程序清单3.9 获取 TPS08U 通道数据

1 int8\_t err;

2 uint32\_t data;

3

4 tps08u\_handle\_t handle = tps08u\_inst\_init();

5

6 err = tps08u\_chan\_enable\_get (handle, 0, &data);

7 if (err < 0) {

8 // 获取失败

9 }

#### 获取 TPS08U 状态

该函数用来获取TPS08U的状态，判断当前产生的异常。该函数原型如下：

int8\_t tps08u\_state\_get (tps08u\_handle\_t handle, uint8\_t \*p\_state);

其中，参数handle为TPS08U设备操作句柄；p\_state为指向保存TPS08U状态的缓存的指针。其返回值为0表示获取成功，返回值小于0时表明获取失败。

该接口的使用方法详见程序清单3.10。

程序清单3.10 获取 TPS08U 使能的通道

1 int8\_t err;

2 uint8\_t state;

3

4 tps08u\_handle\_t handle = tps08u\_inst\_init();

5

6 err = tps08u\_state\_get (handle, state);

7 if (err < 0) {

8 // 获取失败

9 }

#### 复位 TPS08U

该函数用来复位TPS08U。该函数原型如下：

int8\_t tps08u\_reset (tps08u\_handle\_t handle);

其中，参数handle为TPS08U设备操作句柄。其返回值为0表示复位成功，返回值小于0时表明复位失败。

该接口的使用方法详见程序清单3.11。

程序清单3.11 获取 TPS08U 使能的通道

1 int8\_t err;

2

3 tps08u\_handle\_t handle = tps08u\_inst\_init();

4

5 err = tps08u\_reset (handle);

6 if (err < 0) {

7 // 复位失败

8 }

在发送指令使TPS08U复位后，应等待一段时间再进行下一步的操作，以让TPS08U完成复位操作。

### TPS08U的解初始化

与初始化函数相对应，解初始化函数的作用是在不需要使用TPS08U设备的时候释放初始化函数所申请的资源。实际上，该函数首先便调用了平台解初始化函数。其函数原型为：

void tps08u\_deinit (tps08u\_handle\_t handle);

可见该函数的形式十分简单，只需将初始化函数获得的handle句柄作为其参数即可。对应的，若在使用多个设备的时候，需要将对应设备的句柄作为参数传入解初始化函数，即：

tps08u\_handle\_t handle1 = tps08u \_1\_inst\_init(); // TPS08U模块1初始化

tps08u \_handle\_t handle2 = tps08u \_2\_inst\_init(); // TPS08U模块2初始化

tps08u \_deinit (handle1); // TPS08U模块1解初始化

tps08u \_deinit (handle2); // TPS08U模块2解初始化

## 典型应用范例

为了使用户加深对各个接口的理解，下面针对一些典型应用编写了相应的应用程序范例，应用程序的入口函数原型为：

void demo\_tps08u\_entry (tps08u\_handle\_t handle);

入口函数有一个tps08u\_handle\_t类型的handle参数，该参数通过TPS08U的初始化函数得到，在不同平台中，获取的方式可能不同，在后续章节中，会分别介绍在AMetal、AWorks中使用TPS08U通用驱动包的方法，其中会提到handle的获取方法。只要获取到handle，即可将其传入到应用入口函数中，以运行相应的应用程序。

这里实现TPS08U通道使能状态的读取和配置、通道模式的读取和配置以及通道数据的读取和转换，范例程序详见程序清单3.12。

程序清单3.12 简单数据收发范例程序

1 #include "tps08u.h"

2

3 /\*\*

4 \* \brief 例程入口

5 \*/

6 void demo\_tps08u\_entry (tps08u\_handle\_t handle)

7 {

8 uint8\_t chan = 5;

9 uint8\_t chan\_enable;

10 uint8\_t mode;

11 uint32\_t data;

12 float temp;

13

14 tps08u\_chan\_enable\_get(handle, 0, &chan\_enable); /\* 获取所有通道使能状态 \*/

15 debug\_printf("chan\_enable = %x\r\n", chan\_enable);

16 debug\_printf("change chan\_enable\r\n");

17 tps08u\_chan\_enable\_set(handle, 0, (0x1 << (chan - 1))); /\* 使能通道5 \*/

18 tps08u\_chan\_enable\_get(handle, 0, &chan\_enable);

19 debug\_printf("chan\_enable = %x\r\n", chan\_enable);

20

21 tps08u\_chan\_mode\_get(handle, 0, &mode); /\* 获取所有通道模式 \*/

22 debug\_printf("chan mode = %x\r\n", mode);

23 debug\_printf("change chan mode\r\n");

24 tps08u\_chan\_mode\_set(handle,

25 0,

26 TPS08U\_MODE\_CH1\_2\_V | /\* 设置1、2通道电压模式 \*/

27 TPS08U\_MODE\_CH3\_4\_C | /\* 设置3、4通道电流模式 \*/

28 TPS08U\_MODE\_CH5\_6\_V | /\* 设置5、6通道电压模式 \*/

29 TPS08U\_MODE\_CH7\_8\_C); /\* 设置7、8通道电流模式 \*/

30 tps08u\_chan\_mode\_get(handle, 0, &mode);

31 debug\_printf("chan mode = %x\r\n", mode);

32

33 while(1) {

34 tps08u\_chan\_data\_get(handle, chan, &data); /\* 获取通道数据 \*/

35 temp = tps08u\_uint32\_to\_float(data); /\* 将数据转换为浮点型 \*/

36

37 debug\_printf("chan%d uint\_value = %d float\_value = %d.03%d\r\n",

38 chan,

39 data,

40 (int)temp,

41 (int)(temp \* 1000)% 1000);

42

43 tps08u\_platform\_mdelay(10);

44 }

45 }

实际上，在不同平台下的debug\_printf ()串口打印函数的实现可能不同，因此，运行该范例程序时，需要根据实际情况，完成对debug\_printf ()的定义，以便应用程序正常运行。

# 在AMetal中使用TPS08U

**✍本章导读**

AMetal平台已经完成了对TPS08U的适配，可以在初始化之后，使用TPS08U通用驱动软件包中提供的各个接口操作TPS08U模块，完成信号采集的相关操作，无需关心任何指令细节，使用户可以更加快捷的将TPS08U应用到实际项目中，快速实现应用程序的“跨平台复用”。本章主要讲述如何在AMetal下使用已是配好的驱动包。

## 实例初始化函数

在第3章中提到，AMetal已经完成了对TPS08U通用软件包的适配，若用户选用AMetal平台，则无需自行适配，直接使用相应的功能接口操作TPS08U模块即可。

通过3.5节中的内容可知，所有的功能接口均需传入一个handle参数，用以指定操作的TPS08U对象，当使用AMetal时，可以直接通过TPS08U的实例初始化函数获得相应的handle，其函数原型为：

tps08u\_handle\_t tps08u\_inst\_init (void);

调用形式如下：

tps08u\_handle\_t tps08u\_handle = tps08u\_inst\_init();

此处获取的tps08u\_handle即作为于TPS08U通用软件包提供的功能接口的句柄传入。

## 实例解初始化函数

与实例初始化函数对应，实例解初始化函数则是释放初始化所申请的系统资源，一般不在需要使用TPS08U设备时就该将设备解初始化。在Ametal中，实例解初始化函数原型为：

void tps08u\_inst\_deinit (tps08u\_handle\_t tps08u\_handle);

调用形式如下：

am\_tps08u\_inst\_deinit(tps08u\_handle);

## 配置文件

在AMetal中，实例初始化函数在工程中的TPS08U配置文件中实现，配置文件的默认名称为： am\_hwconf\_tps08u.c，这里以ST芯片stm32g071来驱动TPS08U模块为例，配置文件的具体内容详见程序清单4.1。

程序清单4.1 am\_hwconf\_tps08u.c

1 #include "ametal.h"

2 #include "am\_spi.h"

3 #include "tps08u.h"

4 #include "tps08u\_platform.h"

5 #include "stm32g071\_pin.h"

6

7 /\*\*< \brief TPS08U 设备实例 \*/

8 static tps08u\_dev\_t \_\_g\_tps08u\_dev;

9

10 /\*\*< \brief TPS08U设备信息 \*/

11 static const tps08u\_devinfo\_t \_\_g\_tps08u\_info = {

12 /\* 通道初始化模式 \*/

13 TPS08U\_MODE\_CH1\_2\_V |

14 TPS08U\_MODE\_CH3\_4\_V |

15 TPS08U\_MODE\_CH5\_6\_V |

16 TPS08U\_MODE\_CH7\_8\_C,

17 {

18 PIOB\_0,

19 3000000,

20 am\_stm32g071\_spi1\_int\_inst\_init,

21 am\_stm32g071\_spi1\_int\_inst\_deinit

22 }

23 };

24

25 /\*\* \brief TPS08U 实例初始化，获得标准服务句柄 \*/

26 tps08u\_handle\_t tps08u\_inst\_init (void)

27 {

28 return tps08u\_init(&\_\_g\_tps08u\_dev, &\_\_g\_tps08u\_info);

29 }

30

31 /\*\* \brief TPS08U 实例解初始化 \*/

32 void tps08u\_inst\_deinit (tps08u\_handle\_t tps08u\_handle)

33 {

34 tps08u\_deinit(tps08u\_handle);

35 }

注：若在用户所使用的工程中不存在该文件，则可以将上述范例程序中的内容原封不动的复制到一个新文件中（新文件可命名为am\_hwconf\_tps08u.c），完成配置文件的添加。

由程序清单4.1可知，配置文件的主要用户是对设备信息结构体以及设备结构体分配必要的内存，不仅如此， 用户可以需要根据实际情况修改配置文件中的一些信息，例如，TPS08U的SPI控制引脚与主控MCU连接的引脚号、SPI时钟频率等。在AMetal的TPS08U的配置文件中，用户可以修改的信息大总共5个配置项，各配置项的功能简介详见表4.1。

表4.1 AMetal下TPS08U设备的配置项

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 对应程序清单4.1的行号 | 配置项 | 默认值 |
| 1 | 13 | TPS08U通道模式 | TPS08U\_MODE\_CH1\_2\_V | TPS08U\_MODE\_CH3\_4\_V | TPS08U\_MODE\_CH5\_6\_V | TPS08U\_MODE\_CH7\_8\_C |
| 2 | 18 | SPI使能引脚 | PIOB\_0 |
| 3 | 19 | SPI时钟频率 | 3000000 |
| 4 | 20 | SPI句柄获取函数 | am\_stm32g071\_spi1\_int\_inst\_init |
| 5 | 21 | SPI解初始化函数 | am\_stm32g071\_spi1\_int\_inst\_deinit |

下面，分别对这5个配置项进行详细介绍。

1. TPS08U通道模式

此配置项，实际上是设置一个默认的通道模式，如果用户后续没有对通道模式进行配置，则TPS08U按照此处设置的默认通道模式运行：

TPS08U\_MODE\_CH1\_2\_V | /\* 通道1、2电压通道模式 \*/

TPS08U\_MODE\_CH3\_4\_V | /\* 通道3、4电压通道模式 \*/

TPS08U\_MODE\_CH5\_6\_V | /\* 通道5、6电压通道模式 \*/

TPS08U\_MODE\_CH7\_8\_C /\* 通道7、8电流通道模式 \*/

1. SPI使能引脚

TPS08U在SPI通信协议中是作为从机，因此需要一个引脚对其进行使能选择，当引脚为低电平时TPS08U被使能，当引脚为高电平时TPS08U被禁能。该配置项位于程序清单4.1的第18行：

18 PIOB\_0,

1. SPI时钟频率

TPS08U的SPI时钟频率最高可达6M(部分平台SPI时钟频率太高可能无法正常工作)。该配置项位于程序清单4.1的第19行：

19 3000000,

1. SPI句柄获取函数

该配置项位于程序清单4.1的第20行：

20 am\_stm32g071\_spi1\_int\_inst\_init,

此处默认使用中断模式的SPI1，若无特殊需要，一般不对其作修改。若用户需要在操作时修改SPI模式，则程序清单4.1的第20行可修改为轮询模式的SPI1：

20 am\_stm32g071\_spi1\_poll\_inst\_init,

1. SPI解初始化函数

该配置项位于程序清单4.1的第21行：

21 am\_stm32g071\_spi1\_int\_inst\_deinit

其作用是释放配置的SPI资源。

## 应用

当调用实例初始化函数获得TPS08U句柄时，即可使用该句柄操作TPS08U，范例程序详见程序清单4.2。

程序清单4.2 使用TPS08U通用测试程序进行测试

1 #include "ametal.h"

2 #include "app\_tps08u.h"

3 #include "tps08u.h"

4 #include "am\_hwconf\_tps08u.h"

5

6 int am\_main (void)

7 {

8 tps08u\_handle\_t handle = tps08u\_inst\_init();

11 demo\_tps08u\_entry (handle);

12

13 while(1) {

14 }

15 }

由此可见，通过实例初始化函数可以快捷的获取到TPS08U操作句柄，该句柄可用于通用软件包提供的各个功能接口。

在TPS08U应用程序中（详见程序清单3.12），使用了debug\_printf()串口打印函数，不同平台需要根据实际情况对这些函数进行实现，在AMetal平台中，提供了这个函数的操作接口，详见程序清单4.3 。

程序清单4.3 debug\_printf()函数在AMetal中的实现

1 #include "am\_vdebug.h"

2 #define debug\_printf(...) (void)am\_kprintf(\_\_VA\_ARGS\_\_)

其中，am\_kprintf ()在am\_vdebug.h文件中声明，用于串口打印调试信息。

# 在AWorks中使用TPS08U

**✍本章导读**

AWorks平台已经完成了对TPS08U通用驱动包的适配，可以在使能设备之后，使用TPS08U通用驱动软件包中提供的各个接口操作TPS08U模块，完成数据收发，无需关心任何底层细节，使用户可以更加快捷的将TPS08U应用到实际项目中。本章主要讲述如何在AWorks下使用已适配好的驱动包。

## 设备使能

在使用TPS08U之前，必须使能TPS08U相关的硬件设备，并完成相关的配置。

设备使能的方法为：确保在aw\_prj\_params.h文件中定义的AW\_DEV\_IMX1050\_LPSPI3和AW\_DEV\_TPS08U宏处于有效状态，即未被注释。

#define AW\_DEV\_IMX1050\_LPSPI3

#define AW\_DEV\_TPS08U

若aw\_prj\_params.h文件中没有定义AW\_DEV\_TPS08U，则可以自行添加该宏的定义。通常情况下，若未定义该宏，表明用户所使用的模版工程没有添加TPS08U设备的默认配置，此时，用户还需添加相应的配置文件（在下节介绍配置时会详细介绍）。

## 设备配置

这里以RT1050开发板为例，设备相关的配置集中在用户配置文件目录（user\_config\awbl\_hwconf\_usrcfg\）下的awbl\_hwconf\_tps08u.h文件中，文件的示意内容详见程序清单5.1。

程序清单5.1 awbl\_hwconf\_tps08u.h文件内容

1 #ifndef \_\_AMBL\_HWCONF\_TPS08U\_H

2 #define \_\_AMBL\_HWCONF\_TPS08U\_H

3

4 #ifdef AW\_DEV\_TPS08U

5

6 #include "aworks.h"

7 #include "tps08u.h"

8 #include "awbl\_tps08u.h"

9

10 /\*\*< \brief TPS08U 设备实例 \*/

11 aw\_local awbl\_tps08u\_dev\_t \_\_g\_tps08u\_dev;

12

13 /\*\*< \brief TPS08U设备信息 \*/

14 aw\_local const tps08u\_devinfo\_t \_\_g\_tps08u\_info = {

15 /\* 通道初始化模式 \*/

16 TPS08U\_MODE\_CH1\_2\_V |

17 TPS08U\_MODE\_CH3\_4\_V |

18 TPS08U\_MODE\_CH5\_6\_V |

19 TPS08U\_MODE\_CH7\_8\_C,

20 {

21 28, /\*CS引脚编号\*/

22 3000000,

23 2 /\* SPI3总线 \*/

24 }

25 };

26

27 #define AWBL\_HWCONF\_TPS08U \

28 { \

29 AWBL\_TPS08U\_NAME, \

30 0, \

31 AWBL\_BUSID\_PLB, \

32 0, \

33 &\_\_g\_tps08u\_dev.dev, \

34 &\_\_g\_tps08u\_info \

35 },

36

37 #else

38 #define AWBL\_HWCONF\_TPS08U

39

40 #endif /\* AW\_DEV\_TPS08U \*/

41

42 #endif /\* \_\_AMBL\_HWCONF\_TPS08U\_H \*/

特别地，若在用户获取的SDK中，不包含此文件，则可以自行新建一个名为awbl\_hwconf\_tps08u.h的文件，并将程序清单5.1中的内容作为新建文件的内容。

接下来，首先对这个文件的作用进行一个整体的分析，然后再对其中用户可能需要修改的内容（配置项）作详细介绍。

#### 文件作用整体分析

该文件的主要作用是完成TPS08U的配置，并对外提供一个设备宏定义，即：AWBL\_HWCONF\_TPS08U，仅关注与定义该宏相关的语句，忽略其它代码，即：

#ifdef AW\_DEV\_TPS08U

#define AWBL\_HWCONF\_TPS08U \

{ \

AWBL\_TPS08U\_NAME, \

0, \

AWBL\_BUSID\_PLB, \

0, \

&\_\_g\_tps08u\_dev.dev, \

&\_\_g\_tps08u\_info \

},

#else

#define AWBL\_HWCONF\_TPS08U

#endif /\* AW\_DEV\_TPS08U \*/

由此可见，该宏可能有两种定义，具体定义被AW\_DEV\_TPS08U宏控制，仅当AW\_DEV\_TPS08U被有效定义时，AWBL\_HWCONF\_TPS08U的定义才包含实际内容，否则，AWBL\_HWCONF\_TPS08U是一个内容为空的宏。这实际上也进一步展示了为什么AW\_DEV\_TPS08U可以作为设备使能宏。

一个硬件设备要正常工作，必须将其对应的设备宏加入到AWorks指定的硬件设备列表中，硬件设备列表在awbus\_lite\_hwconf\_usrcfg.c文件中定义，即一个名为：g\_awbl\_devhcf\_list[]的数组，该数组的每一个成员都描述了系统中的一个硬件设备。一个简单的示例片段详见程序清单5.2。

程序清单5.2 硬件设备列表（awbus\_lite\_hwconf\_usrcfg.c）

1 aw\_const struct awbl\_devhcf g\_awbl\_devhcf\_list[] = { // 硬件设备列表

2 AWBL\_HWCONF\_IMX1050\_NVIC // 中断控制器

3 AWBL\_HWCONF\_IMX1050\_GPIO // GPIO

4 AWBL\_HWCONF\_GPIO\_LED // LED设备

5 // ...... 其它硬件设备

6 };

在g\_awbl\_devhcf\_list[]数组中，每个元素都是以“AWBL\_HWCONF\_”作为前缀的一个宏，该宏本质上完成了一个设备描述的定义。例如，要使用TPS08U，则应该将TPS08U对应的设备宏加入到硬件设备列表中，详见程序清单5.3。

程序清单5.3 硬件设备列表中加入TPS08U

1 aw\_const struct awbl\_devhcf g\_awbl\_devhcf\_list[] = { // 硬件设备列表

2 AWBL\_HWCONF\_IMX1050\_NVIC // 中断控制器

3 AWBL\_HWCONF\_IMX1050\_GPIO // GPIO

4 AWBL\_HWCONF\_GPIO\_LED // LED设备

5 **AWBL\_HWCONF\_TPS08U** // TPS08U设备

6 // ...... 其它硬件设备

7 };

通常情况下，若在系统工程中存在TPS08U的配置文件，则该宏默认已经加入到了硬件设备列表中，用户只需要用过使能宏AW\_DEV\_TPS08U控制设备是否使能即可。

此外，还需将TPS08U设备注册到AWBus子系统中，在已适配好AWorks的TPS08U驱动中，awbl\_tps08u.h文件 以提供了驱动注册接口，其函数原型如下：

void awbl\_tps08u\_drv\_register (void);

在工程目录user\_config文件夹下的aw\_prj\_config.c中，函数awbl\_group\_init()完成了所有注册在AWBus总线上的驱动，如：

0 static void awbl\_group\_init (void)

1 {

2 //其他注册信息

3

4 #ifdef AW\_DRV\_TPS08U

5 awbl\_tps08u\_drv\_register();

6 #endif

7

8 //其他注册信息

9 }

其中，宏AW\_DRV\_TPS08U在文件aw\_prj\_param\_auto\_cfg.h文件中定义，其定义详情如下：

#ifdef AW\_DEV\_TPS08U

#define AW\_DRV\_TPS08U

#endif;

由此可见，宏AW\_DRV\_TPS08U也是由宏AW\_DEV\_TPS08U来控制系统是否需要将其注册到AWBus中。

#### 配置项详解

用户可能需要根据实际情况修改配置文件中的一些信息，例如，TPS08U的SPI使能引脚号、SPI时钟频率等。在TPS08U的配置文件中，用户可以修改的信息大致分为5个配置项，各配置项的功能简介详见表5.1。

表5.1 AWorks下TPS08U设备的配置项

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 对应程序清单5.1的行号 | 配置项 | 默认值 |
| 1 | 16 | 通道模式 | TPS08U\_MODE\_CH1\_2\_V | TPS08U\_MODE\_CH3\_4\_V | TPS08U\_MODE\_CH5\_6\_V | TPS08U\_MODE\_CH7\_8\_C |
| 2 | 21 | SPI使能引脚编号 | 28 |
| 3 | 22 | SPI时钟频率 | 3000000 |
| 4 | 23 | SPI总线编号 | 2 |

下面，分别对这5个配置项作详细介绍。

1. 通道模式

该配置项对应程序清单5.1的第（11-14）行：

16 TPS08U\_MODE\_CH1\_2\_V | /\* 通道1、2电压通道模式 \*/

17 TPS08U\_MODE\_CH3\_4\_V | /\* 通道3、4电压通道模式 \*/

18 TPS08U\_MODE\_CH5\_6\_V | /\* 通道5、6电压通道模式 \*/

19 TPS08U\_MODE\_CH7\_8\_C /\* 通道7、8电流通道模式 \*/

1. SPI总线编号

该配置项对应程序清单5.1的第（23）行：

23 2, /\* SPI3总线 \*/

1. SPI时钟频率

该配置项对应程序清单5.1的第（22）行：

22 3000000,

TPS08U的SPI时钟频率最高可达6M(部分平台SPI时钟频率太高可能无法正常工作)。

1. SPI使能引脚编号

TPS08U在SPI通信协议中是作为从机，因此需要一个引脚对其进行使能选择，当引脚为低电平时TPS08U被使能，当引脚为高电平时TPS08U被禁能。该配置项对应程序清单5.1的第（21）行：

16 28 /\*CS引脚编号\*/

## 使用TPS08U通用软件包接口

在第三章中提到，AWorks已经完成了对TPS08U通用软件包的适配，若用户选用AWorks平台，则无需自行适配，直接使用相应的功能接口操作TPS08U即可。

通过3.5节中的内容可知，所有的功能接口均需传入一个handle参数，用以指定操作的TPS08U对象，在AWorks平台下的awbl\_hwconf\_tps08u.h文件提供了获取操作句柄handle的接口，可以直接通过该接口获得handle，其函数原型为：

tps08u\_handle\_t tps08u\_inst\_init (void);

当调用实例初始化函数获得TPS08U设备句柄时，即可使用该句柄操作TPS08U设备，范例程序详见程序清单5.4。

程序清单5.4 使用TPS08U通用测试程序进行测试

1 #include "aworks.h"

2 #include "aw\_vdebug.h"

3 #include "aw\_delay.h"

4 #include "aw\_led.h" /\* LED服务 \*/

5

6 #include "tps08u.h"

7 #include "app\_tps08u.h"

8 #include "awbl\_tps08u.h"

9

10 #define LED 0

11

12 int aw\_main()

13 {

14 tps08u\_handle\_t handle = awbl\_tps08u\_handle\_get(0);

15

16 aw\_kprintf("\r\nApplication Start. \r\n");

17

18 demo\_tps08u\_entry(handle);

19

20 while (1) {

21 aw\_led\_toggle(LED);

22 aw\_mdelay(500);

23 }

24

25 return 0;

26 }

在TPS08U应用程序中（详见程序清单3.12），使用了debug\_printf()串口打印函数，不同平台需要根据实际情况对这个函数进行实现，在AWorks平台中，提供了这个函数的操作接口，详见程序清单5.5。

程序清单5.5 debug\_printf()函数在AWorks中的实现

1 #include "aw\_vdebug.h"

2 #define debug\_printf(...) (void)aw\_kprintf(\_\_VA\_ARGS\_\_)

其中，aw\_kprintf ()在aw\_vdebug.h文件中声明，用于串口打印调试信息。