Application Note

TI 智能保险丝高侧开关的常见软件用例示例

Timothy Logan

摘要

德州仪器 (TI) 的智能保险丝高侧开关产品系列（HCS 器件系列）提供了一套功能强大的多功能器件，可在汽车应用中使用可配置的半导体设计无缝替换物理熔断型保险丝器件。HCS 高侧开关器件系列采用 SPI 作为通信方式，以配置各种参数（如电容充电/I2T）以及读取电流检测（通过集成 ADC）和故障检测等诊断信息。为了简化软件开发，德州仪器 (TI) 在多个处理器或微控制器平台（包括德州仪器 (TI) 或其他平台）上提供一整套软件驱动程序和代码示例。本应用手册深入探讨了德州仪器 (TI) 智能保险丝软件生态系统的各个方面，包括顶级驱动程序、配置或评估工具以及一整套展示底层智能保险丝器件常见用例的代码示例。

内容

商标

1 软件生态系统

德州仪器 (TI) 的智能保险丝生态系统包含以下软件组件：

表 1-1. 智能保险丝配套产品

配套资料名称	说明
Smart Fuse Configurator	主机 GUI 工具，用于配置 TPSxHCxx-Q1 器件并导出 C 配置文件以进行软件开发。此软件还用于控制 HSS-HCSMOTHERBRDEVM 和相应的子卡。
器件特定的 C 头文件	HCS 器件的寄存器映射的头文件表示。此文件包含器件的所有寄存器定义和枚举。
HCS 平台驱动程序	具有较低级别的 SPI 驱动程序移植层的通用驱动程序集。提供了针对多种处理器/微控制器的实现示例。
应用代码示例	一组通用代码示例，展示了使用德州仪器 (TI) 高侧开关 HCS 系列的一些常见功能和差异化特性。

可在相应的软件页面上找到包含 HCS 平台驱动程序和应用代码示例的软件包。驱动程序和代码示例均为经过 BSD 许可的开源代码，允许灵活的移植/重用。软件包可以在 HCS-SMARTFUSE-DRIVERS 中找到。

请注意，*智能保险丝评估模块* 用户指南中详细介绍了 Smart Fuse Configurator 软件与 HSS-HCSMOTHERBRDEVM 相关的功能。

所有软件配套资料均可相互配合，简化软件开发，使 HCS 系列高压侧开关的使用尽可能容易上手。标准开发流程可包括以下步骤：

1. 使用 Smart Fuse Configurator 生成初始配置。这些设置将在启动期间由微控制器通过 SPI 加载到高侧开关。这些设置包括电流限制配置、电容充电模式和基于特定线规曲线所需的任何特定 I2T 导通。
2. 配置完毕后，该软件会导出一个包含用于初始配置/编程的通用 C 结构的通用 C 文件。
3. 该配置结构的指针被传递到 HCS_initializeDevice 函数（ 节 2.4.9 ）中。然后，驱动器对器件的所有相关寄存器进行编程。该 API 通常在微控制器引导/初始化时调用一次。

上述各个步骤以及各个元件将在后续部分中进行介绍。

2 平台驱动程序

2.1 驱动程序概念

德州仪器 (TI) 为 HCS 系列智能保险丝高侧开关提供的驱动器套件被设计为通用型，并允许对底层器件进行完整配置和利用。这些驱动程序的特性包括：

- 通过 Smart Fuse Configurator 软件中已导出文件进行初始配置。
- 通用寄存器读取或写入，支持返回单个事务标头。
- 用于将高侧开关 ADC 寄存器的原始 ADC 结果转换为人类可读的浮点值的便利函数。
- 提供器件和通道级诊断状态的函数。

图 2-1 中展示了驱动程序以及与所提供代码示例的关系：

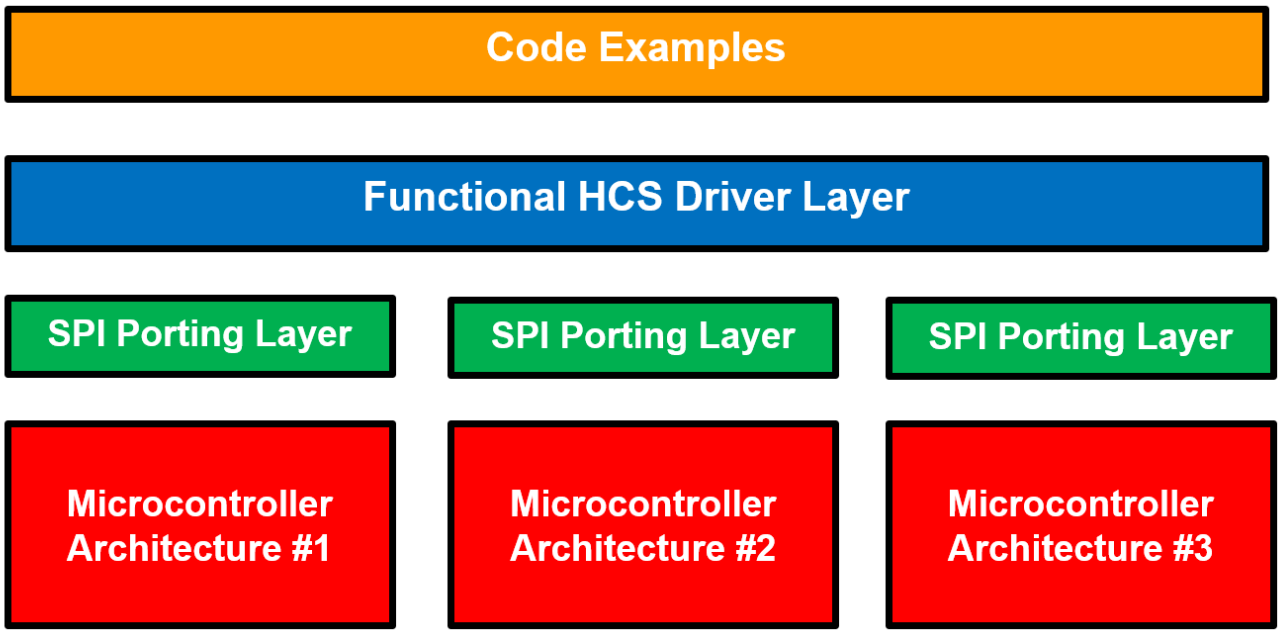

图 2-1. 驱动器架构

顶级驱动程序 API 具有 HCS_ 前缀以表示 HCS 系列智能保险丝器件，这些 API 在软件包的 ***hcs_control_driver.h*** 和 ***hcs_control_driver.c*** 中提供。顶级代码示例使用这些 API 来提供一组通用功能，从而控制和配置高侧开关。对于物理 SPI 通信，***hcs_control_driver.h*** 中声明了一组外部函数：

```
/* --------------------------- Porting Functions ---------------------------- */
/*
 * These functions need to be implemented by each individual device port. The functions
 * handle the low-level hardware specific implementation with the respective
 * architecture's specific hardware peripherals (SPI and GPIO)
 */
bool HCS_port_spiSendData(uint8_t *data, uint8_t len, uint8_t* respData);
void HCS_port_assertSPI(void);
void HCS_port_deassertSPI(void);
```

这些函数在每个单独的架构实现方案中定义，并处理每个平台的硬件 SPI 交互。有关将这些函数移植到其他架构的更多详细信息，请参阅“移植到其他平台”（节 2.3）。包含功能、参数和返回值的完整 API 列表可在 API 指南（节 2.4）中找到。

2.2 支持平台

智能保险丝驱动器和配置包包含针对多种不同微控制器或处理器的实现示例。会定期添加其他实现，但当前架构和相关开发套件的列表如表 2-1 所示：

表 2-1. 支持平台

架构	开发板	生态系统说明
德州仪器 (TI) MSPM0G3507-Q1	LP-MSPM0G3507	Code Composer Studio Theia
STMicroelectronics STM32H723ZGT6	NUCLEO-H723ZG	STM32CubeIDE

可以通过实现节 2.3 中所述的相关器件级函数来添加其他器件支持。

HSS-HCMOTHERBRDEVM 具有通用 SPI 接头，可用于将外部 SPI 信号*插入* 到所连接的高侧开关子卡中。在开发代码示例和 HCS 平台驱动程序期间，下面列出的开发板与 HSS-HCMOTHERBRDEVM 协同使用以进行验证。下面的图 2-2 中提供了此配置中使用的 LP-MSPM0G3507 评估模块的示例：

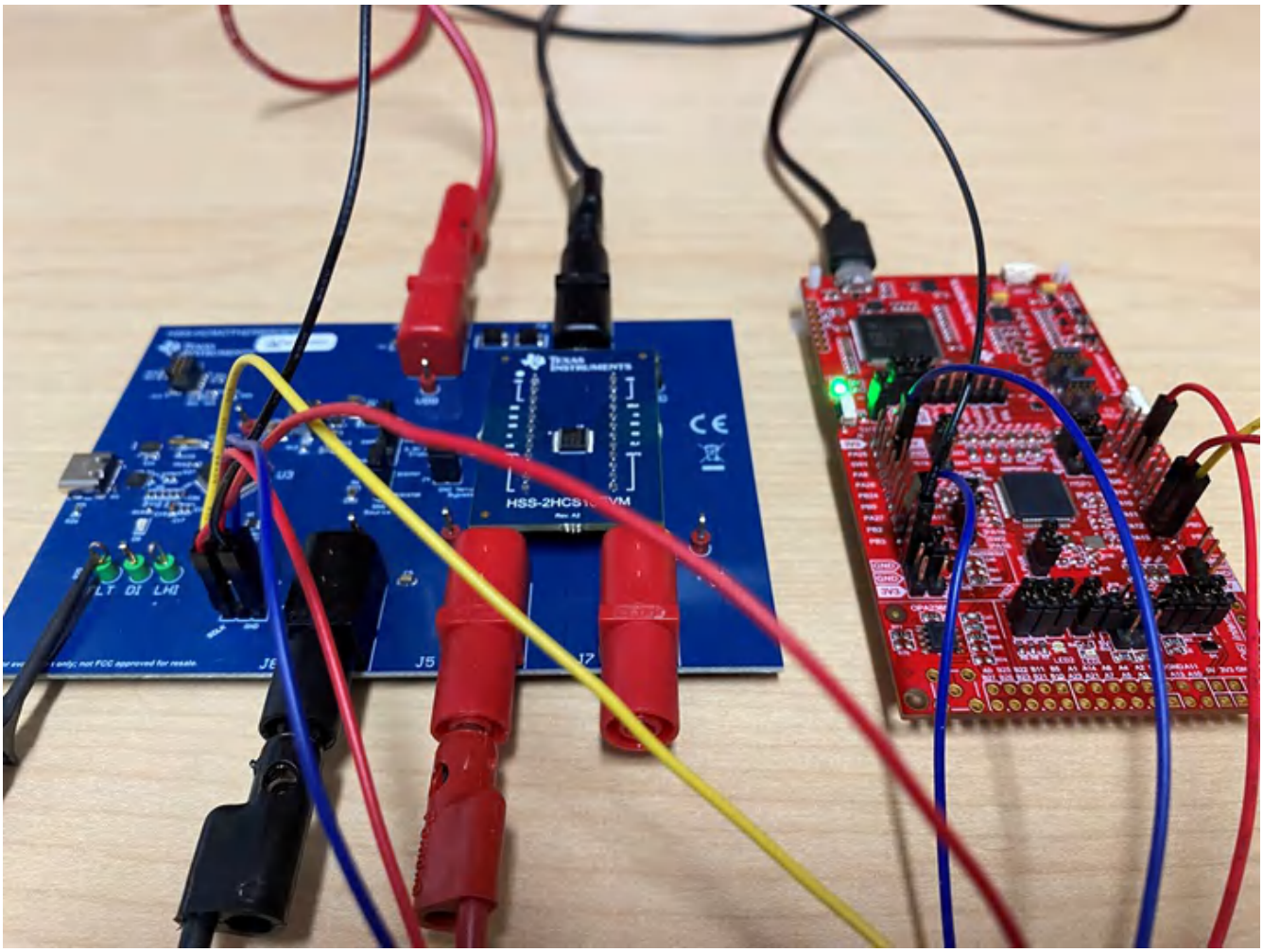

图 2-2. 连接的 MSPM0+

下图展示了使用 NUCLEO-H723ZG 板的连接：

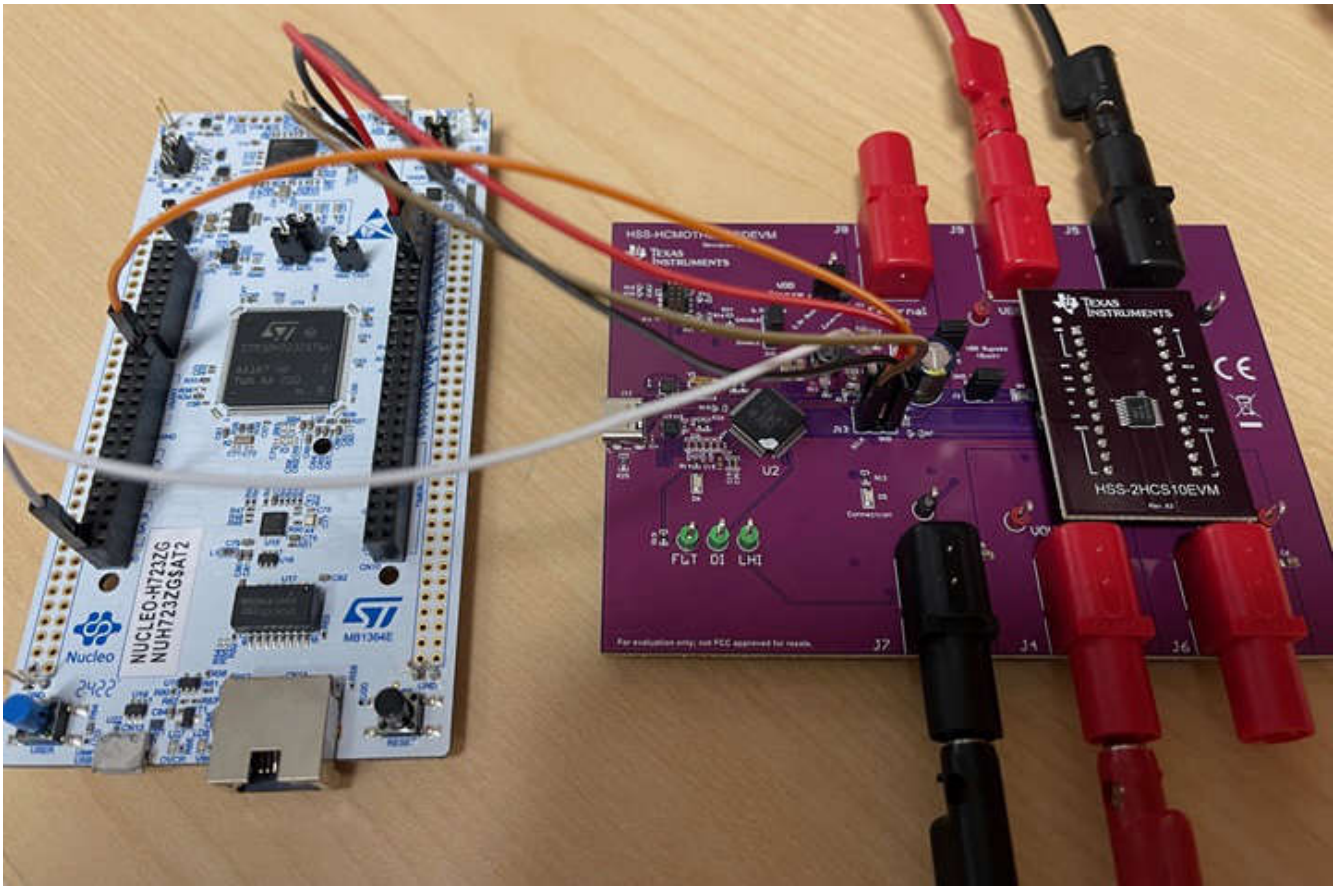

图 2-3. NUCLEO-H723ZG 连接

2.3 移植到其他平台

虽然提供了各种示例架构实现方案，但 HCS 平台驱动程序经过专门设计，可以轻松移植到支持 SPI 和 C 编程语言的任何微处理器或架构。在驱动程序的头文件 (***hcs_control_driver.h***) 中，以下两个函数通过外部引用声明：

```
/* --------------------------- Porting Functions ---------------------------- */
/*
 * These functions need to be implemented by each individual device port. The functions
 * handle the low-level hardware specific implementation with the respective
 * architecture's specific hardware peripherals (SPI and GPIO)
 */
bool HCS_port_spiSendData(uint8_t *data, uint8_t len, uint8_t* respData);
void HCS_port_assertSPI(void);
void HCS_port_deassertSPI(void);
```

移植意味着简单直接，表 2-2 展示了每个函数的说明和移植指导。

表 2-2. 架构移植函数

功能	说明	返回值
HCS_port_spiSendData	执行全双工 SPI 事务。***data*** 表示待发送的数据，***respData*** 是接收到的数据。***len*** 是事务的长度。在整个 SPI 事务完成之前，该函数需要处于阻塞状态（通过睡眠或轮询）。	事务结果的 stdbool 表示。如果事务已完成而未出现问题，则为 ***true***，否则为 ***false***。
HCS_port_assertSPI	将 SPI 总线的片选 (CS) 线路设置为低电平。这用于单链和菊花链事务处理，也可供上面的 HCS_port_spiSendData 函数使用。	无
HCS_port_deassertSPI	将 SPI 总线的片选 (CS) 线路设置为高电平。这用于单链和菊花链事务处理，也可供上面的 HCS_port_spiSendData 函数使用。	无

2.4 API 指南

参考资料

- tHCSResponseCode 联合体（节 2.4.1）
- HCS_convertCurrent（节 2.4.2）
- HCS_convertTemperature（节 2.4.3）
- HCS_convertVoltage（节 2.4.4）
- HCS_getChannelFaultStatus（节 2.4.5）
- HCS_getDeviceFaultSatus（节 2.4.6）
- HCS_gotoLPM（节 2.4.7）
- HCS_gotoSleep（节 2.4.8）
- HCS_initializeDevice（节 2.4.9）
- HCS_readRegister（节 2.4.10）
- HCS_setSwitchState（节 2.4.11）
- HCS_updateConfig（节 2.4.12）
- HCS_wakeupDevice（节 2.4.13）
- HCS_writeRegister（节 2.4.14）

2.4.1 tHCSResponseCode 联合体参考

数据字段

```
typedef union
{
    uint8_t byte;
    struct
    {
        unsigned I2T_FLT : 1;
        unsigned LPM_FLT : 1;
        unsigned CHAN_TSD : 1;
        unsigned ILIMIT_FLT : 1;
        unsigned SHRT_VBB_FLT : 1;
        unsigned OL_FLT : 1;
        unsigned SUPPLY_FLT : 1;
        unsigned GLOBAL_ERR_WRN : 1;
    } bits;
} tHCSResponseCode;
```

2.4.2 float_t HCS_convertCurrent (uint16_t rawValue, uint16_t ksnsVal, uint16_t snsRes)

将原始 ADC 电流值转换为可读的浮点值。

这是一个便利的函数，可以获取从其中一个 ADC 结果寄存器读取的原始值并将该寄存器转换为人类可读的浮点值。

表 2-3. 参数

in	*rawValue*	要转换的原始值
in	*ksnsVal*	数据表中的 KSNS 常数
in	*snsRes*	SNS 引脚外接的电阻值

表 2-4. 返回值

returnCode	电流的浮点表示

2.4.3 float_t HCS_convertTemperature (uint16_t rawValue)

将原始 ADC 温度值转换为可读的浮点值。

这是一个便利的函数，可以获取从其中一个 ADC 结果寄存器读取的原始值并将其转换为人类可读的浮点值。

表 2-5. 参数

in	*rawValue*	要转换的原始值

表 2-6. 返回值

returnCode	温度的浮点表示

2.4.4 float_t HCS_convertVoltage (uint16_t rawValue)

将原始 ADC 电压值转换为可读的浮点值。

这是一个便利的函数，可以获取从其中一个 ADC 结果寄存器读取的原始值并将该值转换为人类可读的浮点值。

表 2-7. 参数

in	*rawValue*	要转换的原始值

表 2-8. 返回值

returnCode	电压的浮点表示

表 2-9. 返回值

returnCode	**tHCSResponseCode** 的实例

2.4.5 tHCSResponseCode HCS_getChannelFaultStatus (uint8_t chanNum, uint16_t * fltStatus)

读取单个通道的故障状态。

读取各个通道的故障状态并将状态存储到 fltStatus 中

表 2-10. 参数

in	*chanNum*	要读取的通道编号
out	*fltStatus*	此参数可存储故障状态

2.4.6 tHCSResponseCode HCS_getDeviceFaultSatus (uint16_t * fltStatus)

读取器件的故障状态。

读取器件的故障状态并将状态存储到 fltStatus 中

表 2-11. 参数

out	*fltStatus*	此参数可存储故障状态

表 2-12. 返回值

returnCode	**tHCSResponseCode** 的实例

2.4.7 tHCSResponseCode HCS_gotoLPM (lpm_exit_curr_ch1_t ch1ExitCurrent, lpm_exit_curr_ch2_t ch2ExitCurrent)

将器件置于 LPM 模式。

此函数可以发送命令以进入 LPM 模式。如果器件的退出电流超过退出电流电平，MCU 负责监控 FLT 引脚的下降沿。或者，可以使用 HCS_wakeupDevice 函数来唤醒器件。

表 2-13. 参数

in	*ch1ExitCurrent*	通道 1 的退出电流
in	*ch2ExitCurrent*	通道 2 的退出电流

2.4.8 tHCSResponseCode HCS_gotoSleep (void)

使器件进入 SLEEP 模式。

此函数可以发送命令以进入 SLEEP 模式。请注意，器件寄存器中的所有值都可以在退出睡眠模式后复位。该器件需要由 HCS_wakeupDevice 函数唤醒。

表 2-14. 返回值

returnCode	**tHCSResponseCode** 的实例

2.4.9 tHCSResponseCode HCS_initializeDevice (TPS2HCS10Q1_CONFIG * config)

使用器件配置结构对器件进行初始化。

此函数可以采用由 Smart Fuse Configurator GUI 软件生成的配置结构，并将所有值发送到连接的高侧开关。此处的常见做法是在 MCU 首次启动时或在所有寄存器配置丢失的情况下从睡眠模式唤醒时调用该函数。

表 2-15. 参数

in	*config*	指向配置结构的指针

表 2-16. 返回值

returnCode	**tHCSResponseCode** 的实例

2.4.10 tHCSResponseCode HCS_readRegister (uint8_t addr, uint16_t * readValue)

对指定寄存器地址执行原始寄存器读取。

该函数从给定地址执行简单的寄存器读取，并使用寄存器内容填充提供的指针。

表 2-17. 参数

in	*addr*	要读取的寄存器地址
out	*payload*	填充到此参数的寄存器的值

表 2-18. 返回值

returnCode	**tHCSResponseCode** 的实例

2.4.11 tHCSResponseCode HCS_setSwitchState (uint8_t swState)

设置每个通道的开/关状态。

这可打开或关闭高侧开关的通道。参数是要启用的通道的按位表示。

表 2-19. 参数

in	*swState*	指向配置结构的指针

表 2-20. 返回值

returnCode	**tHCSResponseCode** 的实例

2.4.12 tHCSResponseCode HCS_updateConfig (TPS2HCS10Q1_CONFIG * config)

使用智能保险丝设置更新所提供的配置。

此 API 可使用来自高侧开关的值填充提供的配置结构。当主机软件更改寄存器中的设置并想要更新初始配置结构时，可以使用此 API。

表 2-21. 参数

out	*config*	指向配置结构的指针

表 2-22. 返回值

returnCode	**tHCSResponseCode** 的实例

2.4.13 tHCSResponseCode HCS_wakeupDevice (void)

将器件从睡眠模式唤醒。

此 API 只向寄存器地址 0xFF (不存在) 发出虚拟写入，以便器件从睡眠模式唤醒。由于器件通过 CS 引脚转换从睡眠模式唤醒，因此虚拟写入会唤醒器件。

表 2-23. 返回值

returnCode	**tHCSResponseCode** 的实例

2.4.14 tHCSResponseCode HCS_writeRegister (uint8_t addr, uint16_t payload)

对指定寄存器地址执行原始寄存器写入。

该函数使用提供的有效载荷对给定地址执行简单的寄存器写入操作

表 2-24. 参数

in	*addr*	要写入的寄存器地址
in	*payload*	要写入 addr 的值

表 2-25. 返回值

returnCode	**tHCSResponseCode** 的实例

3 配置或评估工具

Smart Fuse Configurator 工具是一款软件主机工具，可与 HSS-HCMOTHERBRDEVM 一起使用，以实时配置 HCS 高侧开关以及读取电流检测和故障状况等诊断信息。此外，从 1.9.4 版开始，该软件无需物理 EVM 板即可进入*配置模式*。在配置模式下，用户能够更改器件设置的所有不同方面，例如电流限制、电容充电模式、诊断报告等等。该用途还可以使用 I2T 调谐器来配置器件，以匹配要更换的熔断型保险丝的导线配置文件和功能。要进入配置模式，请选择 ***Help->Demo/Config Mode***，如图 3-1 所示：

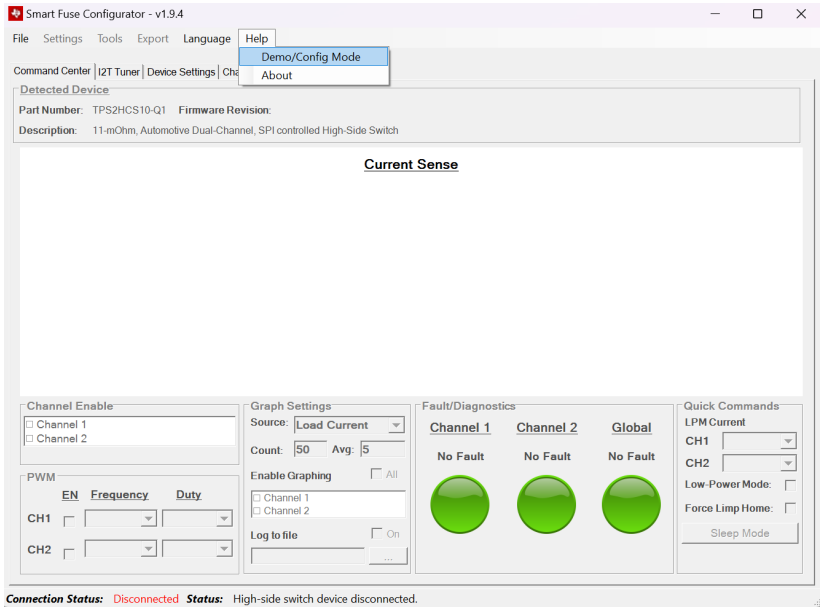

图 3-1. 配置模式

进入演示模式后，用户可以使用智能保险丝评估模块用户指南中所述的软件。请注意，与 EVM 的实际通信不在此模式下执行，并且 GUI 上报告的任何诊断也不会反映出来。通过选择 ***Help->Demo/Config Mode*** 或通过将 EVM 插入器件可以退出演示。

一旦器件配置为满足应用需求，即可通过选择 ***Export->Configuration Files*** 导出配置：

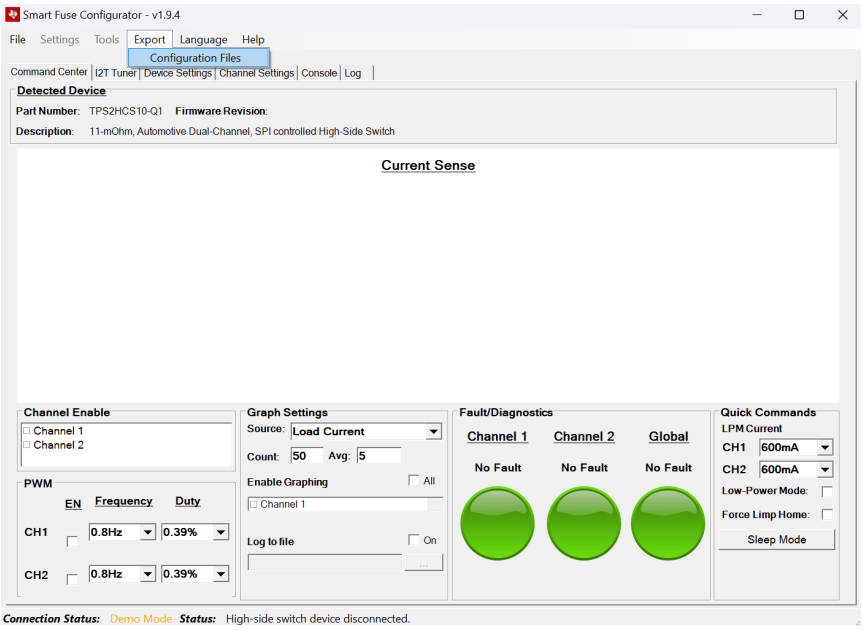

图 3-2. 导出配置

从此对话框导出的文件与所提供的代码示例随附的配置文件（默认名称为 ***tps2hcs10_config.h*** 和 ***tps2hcs10_config.c***）相同。作为一个简单的切入点，可以将配置文件导出到空示例（节 4.1）并覆盖用于开始智能保险丝开发的空白程序的默认文件。

请注意，导出的配置文件取决于项目页面上提供的特定于器件的头文件。此头文件包含与特定高侧开关器件型号相关的所有寄存器定义和枚举。

导出的文件包含一个与器件的每个相关配置寄存器相对应的寄存器定义：

```
typedef struct TPS2HCS10Q1_CONFIG
{
    TPS2HC10S_CRC_CONFIG_OBJ        crcConfig;
    TPS2HC10S_LPM_OBJ               lpmConfig;
    TPS2HC10S_FAULT_MASK_OBJ        faultMaskConfig;
    TPS2HC10S_SW_STATE_OBJ          swState;
    TPS2HC10S_DEVICE_SAF_OBJ        devSAFConfig;
    TPS2HC10S_DEV_CONFIG_OBJ        devConfig;
    TPS2HC10S_ADC_CONFIG_OBJ        adcConfig;
    TPS2HC10S_PWM_CH1_OBJ           pwmCh1Config;
    TPS2HC10S_ILIM_CONFIG_CH1_OBJ   ilimCh1Config;
    TPS2HC10S_DIAG_CONFIG_CH1_OBJ   diagConfigCh1;
    TPS2HC10S_I2T_CONFIG_CH1_OBJ    i2tConfigCh1;
    TPS2HC10S_PWM_CH2_OBJ           pwmCh2Config;
    TPS2HC10S_ILIM_CONFIG_CH2_OBJ   ilimCh2Config;
    TPS2HC10S_DIAG_CONFIG_CH2_OBJ   diagConfigCh2;
    TPS2HC10S_I2T_CONFIG_CH2_OBJ    i2tConfigCh2;
 } TPS2HCS10Q1_CONFIG;
```

该函数的指针被传递到平台驱动程序的 HCS_initializeDevice 函数中（通常在微控制器启动时），以初始配置高侧开关。在结构定义之后，使用表示 Smart Fuse Configurator 工具的所有配置值的值来声明结构的实例化。用户可以使用此实例化作为起点，并在需要更改时通过代码手动更新，或从 Smart Fuse Configurator 程序重新生成配置文件。

4 代码示例

智能保险丝软件包中提供了一组代码示例，其中重点介绍了智能保险丝器件系列的功能或差异，以及 HCS 平台驱动程序的易用性。表 4-1 中显示了可用代码示例的摘要。

表 4-1. 智能保险丝代码示例

代码示例名称	说明
空	简单代码示例，用于从 Smart Fuse Configurator 导出结构初始化高侧开关。
I2T 跳变	展示了 I2T 功能以及如何检测 I2T 故障/从该故障中恢复。
低功耗模式	将器件置于低功耗模式并等待唤醒事件
电流检测	演示了如何使用 HCS 系列的高精度电流检测以及如何优化较低和较高电流的读数

4.1 空示例

空代码示例是一个简单的代码示例，可用作智能保险丝应用的起点。这些代码示例用于启动、配置底层 SPI 外设，然后将控制权移交给用户。初始配置的主要部分如下所示：

```
    /* Configuring the device initially */
    HCS_wakeupDevice();
    HCS_initializeDevice(&exportConfig);
```

HCS_wakeupDevice 函数只是向器件发出虚拟写入。复位结束后，HCS 系列处于睡眠模式。唤醒函数会发出对寄存器 0xFF（不存在）的写入操作，以确保器件退出睡眠状态。

HCS_initializeDevice 函数会获取从 Smart Fuse Configurator 应用导出的配置文件并将其加载到高侧开关中。通常每次 MCU 启动时都会调用该函数，以初始化高侧开关。

4.2 I2T 跳变示例

I2T 跳变代码示例展示了当系统中发生 I2T 事件时如何处理高侧开关上的事件。此代码示例假设器件设置为针对 I2T 故障的锁存模式，并演示了在 I2T 事件发生后*复位*器件的正确事件序列。在自动重试模式（I2T_CONFIG_CHx 寄存器的 TCLDN_CHx 设置为超时）下，器件会在重新启用通道之前自动等待适当的持续时间。

请注意，在该代码示例中，FAULT 引脚设置为用作下降沿中断。当 I2T 跳变发生时，FAULT 引脚（开漏）被拉低，微控制器被中断。然后，微控制器上的软件可以唤醒器件执行，并采取必要的缓解措施来重新启用器件。在锁存模式下，I2T 跳变事件发生后需要执行的适当步骤为：

1. 禁用受影响的通道
2. 将 I2T_CONFIG_CHx 寄存器的 TCLDN_CHx 设置为适当的冷却时间
3. 在微控制器上休眠，等待指定的持续时间
4. 将 I2T_CONFIG_CHx 寄存器的 TCLDN_CHx 设置回锁存模式
5. 重新启用通道

以下是相关的代码片段：

```
        if(currentValue & TPS2HC10S_FLT_STAT_CH1_I2T_FLT_CH1_MASK)
        {
            /* Disabling the channel */
            HCS_setSwitchState(0);

            /* Setting the device to 2s retry state */
            exportConfig.i2tConfigCh1.value.bits.TCLDN_CH1 =
                (tcldn_ch1_en_3_0x2 >> TPS2HC10S_I2T_CONFIG_CH1_TCLDN_CH1_OFS);
           HCS_writeRegister(TPS2HC10S_I2T_CONFIG_CH1_REG,
                        exportConfig.i2tConfigCh1.value.word);

            /* Waiting for two seconds. At this point we can normally
                yield the tasks if we were in an RTOS, but just waiting for
                an interrupt here.  */

            DL_TimerA_startCounter(TIMER_0_INST);
            while(timerTriggered == false)
            {
                __WFI();
            }
            timerTriggered = false;

            /* Setting back to latch mode */
            exportConfig.i2tConfigCh1.value.bits.TCLDN_CH1 = 0;
           HCS_writeRegister(TPS2HC10S_I2T_CONFIG_CH1_REG,
                        exportConfig.i2tConfigCh1.value.word);

            /* Re-enabling the channel */
            HCS_setSwitchState(1);
        }
```

4.3 低功耗模式示例

低功耗模式示例展示了如何将 HCS 高侧开关设置为低功耗模式，然后在发生故障事件时唤醒。要使用此示例，在将代码下载到微控制器之前，在通道 1 上设置一个负载，该负载消耗的电流低于 800mA（我们使用 HCS_gotoSleep 将 LPM 退出电流设置为此值）。下载代码示例之后，将负载电流增加到大于 800mA。这可能导致 LPM 退出，触发 FAULT 引脚，并使微控制器中断/处理此事件。相关的应用程序代码如下所示：

```
    while(1)
    {
        /* Putting the device into LPM. 800mA exit current on CH1 */
        HCS_gotoLPM(lpm_exit_curr_ch1_en_2_0x1, lpm_exit_curr_ch2_en_1_0x0);

        /* Wait for the fault line to trigger low on PB3 */
        __WFI();

        /* If we woke up from the interrupt, check to make sure it was a signal
            for an LPM wakeup. The idea here is that the user increases the
            load current somehow to "force the device" from LPM. */
```

```
        resCode = HCS_readRegister(TPS2HC10S_FLT_STAT_CH1_REG,
                                   &currentValue);
        resCode.byte |=  HCS_readRegister(TPS2HC10S_FLT_STAT_CH1_REG,
                                   &currentValue).byte;

        if(currentValue & TPS2HC10S_FLT_STAT_CH1_LPM_WAKE_CH1_MASK)
        {
            /* Set a breakpoint here for demonstration */
            asm ("nop");
        }

        if(resCode.byte != 0)
        {
            handleError(resCode);
        }
    }
```

4.4 电流检测示例

电流检测代码示例展示了 HCS 系列智能保险丝高侧开关如何实现可扩展且极其灵活的电流检测。此代码示例会设置 1ms 的时间，以便定期唤醒高侧开关通道 1 的负载电流并对其进行采样。如果负载电流低于 500mA，则器件启用高侧开关的以下设置：

- 启用输入电压调节（DIAG_CONFIG_CHx 寄存器的 ISNS_SCALE_CHx），允许将 ADC 输入电压按 8 倍调节
- 启用开路负载检测（DIAG_CONFIG_CHx 的 OL_ON_EN_CHx），这会将 KSNS 比率更改为较低的值（请参阅数据表中的电气规格）

在每个事件上都会设置一个断点行，以允许用户中断并理解电流检测的行为。当在软件中进入低电流检测模式时，会设置状态变量 (inLowCurrent) 以指示当前状态，器件会继续定期对电流采样。如果电流电平使 ADC 读数饱和，则会退出低电流模式并使用正常调节/KSNS 模式。以下代码中演示了相关代码的一个片段：

```
    while(1)
    {
        __WFI();

        /* Reading the load current value. We need to read the register twice
            as the */
        resCode = HCS_readRegister(TPS2HC10S_ADC_RESULT_CH1_I_REG,
                                   &currentValue);
        resCode.byte |= HCS_readRegister(TPS2HC10S_ADC_RESULT_CH1_I_REG,
                                   &currentValue).byte;

        /* For each transaction, the high-side switch can return an error
            status code that can report common faults of the device. */
        if(resCode.byte != 0)
        {
            handleError(resCode);
        }

        /* Masking out the relevant bits */
        currentValue &= TPS2HC10S_ADC_RESULT_CH1_I_ADC_RESULT_CH1_I_MASK;

        /* Check to see if we are below the threshold where we want to turn
            on current sense scaling and change the KSNS ratio and enable
            scaling by 8x. This can allow for  */
        if((currentValue < LOW_CURRENT_SNS_THRESHOLD) &&
                (inLowCurrent == false))
        {
            exportConfig.diagConfigCh1.value.bits.OL_ON_EN_CH1 = 1;
            exportConfig.diagConfigCh1.value.bits.ISNS_SCALE_CH1 = 1;
            inLowCurrent = true;
            HCS_writeRegister(TPS2HC10S_DIAG_CONFIG_CH1_REG,
                              exportConfig.diagConfigCh1.value.word);

            /* Adding place to set a breakpoint for the sake of demonstration */
            asm ("nop");
        }
        else if((currentValue == LOW_CURRENT_SNS_SATURATION) &&
                    (inLowCurrent == true))
        {
            exportConfig.diagConfigCh1.value.bits.OL_ON_EN_CH1 = 0;
            exportConfig.diagConfigCh1.value.bits.ISNS_SCALE_CH1 = 0;
            HCS_writeRegister(TPS2HC10S_DIAG_CONFIG_CH1_REG,
```

```
                                exportConfig.diagConfigCh1.value.word);

            /* Adding place to set a breakpoint for the sake of demonstration */
            asm ("nop");
        }
    }
```

由于能够检测低电流并启用调节/KSNS 模式，高侧检测能够提高电流检测分辨率，并满足需要高电流检测精度的应用的要求。

5 总结

HCS 系列智能保险丝器件的软件驱动程序和代码示例为开发适用于汽车智能保险丝应用的软件提供了多种函数。可以轻松移植这些驱动程序以支持各种不同的主机架构，并且驱动程序的简化设计使开发人员能够在几乎无后端开销的情况下加快 HCS 器件的软件开发。此外，通过使用 Smart Fuse Configurator 软件工具，用户能够顺畅地配置智能保险丝器件，并根据最终应用的确切要求调整驱动器。这些软件工具消除了将 HCS 器件系列改造成智能保险丝或区域应用的障碍，并提供了结合软件或硬件开发的灵活方法。

6 参考资料

- 德州仪器 (TI)，*TPS2HCS10-Q1 具有 I²T 线保护、低 IQ 模式和 SPI 的汽车级双通道 10mΩ 智能高侧开关*
- 德州仪器 (TI)，*HCS-SMARTFUSE-DRIVERS 适用于 HCS 智能保险丝器件的简单 C 驱动程序和代码示例*
- 德州仪器 (TI)，*HSS-HCMOTHERBRDEVM 智能保险丝评估模块*
- 德州仪器 (TI)，*HSS-2HCS10EVM TPS2HCS10-Q1 智能保险丝高侧开关子卡*
- 德州仪器 (TI)，*HSS-SMART-CONFIGURATOR 用于 HSS-HCMOTHERBRDEVM 和 TI 智能保险丝高侧开关的配置工具*
- 德州仪器 (TI)，*HCS-HEADER-FILES 带寄存器定义的智能保险丝高侧开关的 C 头文件*

7 修订历史记录

Changes from Revision * (May 2024) to Revision A (July 2024)	Page
• 通篇更新了表格、图和交叉参考的编号格式	1
• 通篇更新或删除了几个 API 引用	1

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265