STM32 BSP 库使用说明

目录

概述

本 BSP 库为 STM32F4 平台封装了常用外设（UART、CAN、SPI、PWM、定时器）的初始化、启停控制、DMA 传输、回调管理与信息查询接口，并统一使用 `BspResult<T>` 对结果进行包装，方便在业务层处理错误码与上下文信息。

通用 API

所有外设类均依赖以下通用组件：

- **错误码枚举**：`BspError`，包含 `OK`, `InvalidDevice`, `NullHandle`, `HalError`, `Timeout` 等。
- **结果类型**：`BspResult<T>`，支持 `ok()` 判断状态并携带错误上下文。
- **设备枚举**：`BspDevice_t`，例如 `DEVICE_CAN_1`, `DEVICE_USART_1`, `DEVICE_SPI_1` 等。

通用设备管理接口位于 `BspDevice.h`：

```
BspResult<bool> Bsp_SetDeviceHandle(BspDevice_t devID, void* handle);
BspResult<void*> Bsp_GetDeviceHandle(BspDevice_t devID);
BspResult<BspDevice_t> Bsp_FindDeviceByHandle(void* handle);
BspResult<bool> Bsp_StartDevice(BspDevice_t devID);
BspResult<bool> Bsp_StopDevice(BspDevice_t devID);
```

UART

头文件：`BspUart.h`

核心功能

- `Init(uint32_t baud)`：初始化 UART 并设置波特率。
- `EnableRxDMA()`：开启 DMA 接收。

- `SendData(const uint8_t* data, size_t size)` / `ReceiveData(uint8_t* data, size_t size)`：发送与接收。
- `Printf(const char *format, ...)`：格式化输出（VOFA/调试日志）。

回调与状态

- `SetTxCallback(Callback_t cb)` / `SetRxCallback(Callback_t cb)`：注册发送/接收完成回调。
- `InvokeTxCallback()` / `InvokeRxCallback(uint16_t size)`：内部触发接口。
- `ShowInfo()`：输出当前 UART 配置信息。

CAN

头文件：`BspCan.h`

核心功能

- `Init(uint32_t baud, CanMode mode)`：设置波特率与工作模式。
- `Start()` / `Stop()`：启停 CAN 外设。
- `SendStdData(uint32_t id, const uint8_t* data, uint8_t len)`：发送标准帧。
- `SendExtData(uint32_t id, const uint8_t* data, uint8_t len)`：发送扩展帧。
- `SendMessage(const CanMessage& msg)`：发送通用消息结构体。
- `SendRemoteFrame(uint32_t id, bool isExtended)`：发送远程帧。

滤波器配置

- `ConfigFilter(const CanFilterConfig& config)`：完整自定义滤波。
- `ConfigFilterAcceptAll(CanFIFO fifo)`：接收所有报文。
- `ConfigFilterStdId(...)` / `ConfigFilterExtId(...)`：快速配置标准/扩展 ID 滤波。

回调与状态

- `SetRxFifo0Callback(CanRxCallback_t cb)` / `SetRxFifo1Callback(CanRxCallback_t cb)`：注册 FIFO 接收回调。
- `SetTxCallback(Callback_t cb)`：注册发送邮箱完成回调。
- `ShowInfo()`：打印 CAN 速率、滤波配置、回调状态等信息。

SPI

头文件：`BspSpi.h`

核心功能

- `Init(GPIO_TypeDef* csPort, uint16_t csPin)`：初始化 SPI 并配置片选。
- `ConfigCsPin(GPIO_TypeDef* port, uint16_t pin)`：动态调整片选引脚。
- `CsLow()` / `CsHigh()`：片选控制。
- `TransmitDMA(const uint8_t* data, uint16_t size)`：DMA 方式发送。
- `ReceiveDMA(uint8_t* data, uint16_t size)`：DMA 方式接收。
- `TransmitReceiveDMA(const uint8_t* txData, uint8_t* rxData, uint16_t size)`：DMA 全双工。

回调与状态

- `SetTxCallback(SpiTxRxCallback_t cb)` / `SetRxCallback(SpiTxRxCallback_t cb)` / `SetTxRxCallback(SpiTxRxCallback_t cb)`：注册 DMA 完成回调。
- `IsBusy()` / `GetState()`：查询 SPI 硬件状态。
- `ShowInfo()`：输出实例、工作模式、数据宽度、预分频、DMA 链接等信息。

PWM

头文件：`BspPwm.h`

核心功能

- `Init(uint32_t freqHz)`：初始化目标频率。
- `Start(PwmChannel_t channel = CHANNEL_ALL)` / `Stop(PwmChannel_t channel = CHANNEL_ALL)`：控制指定通道。
- `SetDutyCycle(PwmChannel_t channel, float dutyCycle)`：按百分比设置占空比。
- `SetDutyTicks(PwmChannel_t channel, uint32_t ticks)`：按计数值设置占空比。
- `SetFrequency(uint32_t freqHz)`：动态调整频率。

状态查询

- `GetFrequency()` / `GetARR()` / `GetPSC()`：读取当前参数。
- `ShowInfo()`：输出实例、频率、PSC、ARR、CH1 占空比、回调注册情况。

Timer

头文件：`BspTimer.h`

核心功能

- `Init(uint32_t freqHz)`：初始化定时频率。
- `Start()` / `Stop()`：启停定时器。
- `SetCallback(Callback_t cb)`：注册定时回调。
- `InvokeCallback()`：内部触发接口。
- `ShowInfo()`：输出时钟参数、回调状态。

使用示例

```
#include "BspUart.h"
#include "BspCan.h"
#include "BspSpi.h"
#include "BspPwm.h"
#include "BspTimer.h"

// UART 示例
Uart uart(DEVICE_USART_1);
uart.Init(115200);
```

```
uart.Printf("Hello BSP!\n");

// CAN 示例
Can can(DEVICE_CAN_1);
can.Init(Can::BAUD_500K, Can::MODE_NORMAL);
can.Start();
uint8_t data[8] = {1,2,3,4,5,6,7,8};
can.SendStdData(0x123, data, sizeof(data));

// SPI 示例
Spi spi(DEVICE_SPI_1);
spi.Init(GPIOA, GPIO_PIN_4);
spi.CsLow();
spi.TransmitDMA(data, sizeof(data));
spi.CsHigh();

// PWM 示例
Pwm pwm(DEVICE_PWM_12);
pwm.Init(1000); // 1 kHz
pwm.SetDutyCycle(Pwm::CHANNEL_1, 0.5f);
pwm.Start(Pwm::CHANNEL_1);

// Timer 示例
Timer timer(DEVICE_TIMER_1);
timer.Init(100); // 100 Hz
timer.SetCallback([](){ /* 定时器回调 */ });
timer.Start();
```