应用指南

CM32M4xxR 系列芯片
驱动库总体说明

V1.1

目录

一、 概述

CM32M4xxR_Library 是针对 CM32M4xxR 系列 RISC-V 内核 MCU 开发的标准驱动库，提供了一整套涵盖芯片内核及片上外设的功能接口和定义，以及配套 CM32M4xxR-LQFP128 开发板和 CM32M433R 开发板的丰富例程。库中的接口和定义按照外设模块分别组织，覆盖外设的常用功能，并且遵循编程规范，含义明确、简洁易用，开发者无需关注其具体实现即可快速上手使用。驱动库中附带的例程以更直观的方式呈现了驱动及外设的使用方法，例如针对 TIMx 提供了包括 PWM、捕获、比较等功能在内的多达 20 个不同功能的例程，开发者可以直接在开发板上运行例程查看结果或者基于例程进行二次开发，从而有效降低学习成本。

CM32M4xxR_Library 包括以下两部分：

- Drivers：内核及片上外设的标准驱动。
- Projects：基于开发板的例程、BSP，以及工程模板。

本手册主要对 Drives 目录下的内容进行说明，Projects 请参考《CM32M4xxR-LQFP128 开发板样例工程总体说明》和《CM32M433R-START 开发板样例工程总体说明》。

二、 缩略词

表 2-1 缩略词及定义

缩略词	定义
ADC	模拟数字转换器
AES	高级加密标准
BSP	板级支持包
CAN	控制器局域网络
DES	数据加密标准
DSP	数字信号处理
ECLIC	改进型内核中断控制器
FPU	浮点运算单元
HASH	散列函数
HSE	外部高速时钟
HSI	内部高速时钟
ILM	指令本地存储器
LSE	外部低速时钟
LSI	内部低速时钟
MCU	微控制单元，又称单片机
NMSIS	芯来微控制器软件接口标准
PLL	锁相环
PMP	物理存储器保护
PWM	脉冲宽度调制

RTC	实时钟
SM4	国密分组加密标准
USART	通用同步/异步串行接收/发送器

三、 驱动库概述

3.1 目录结构

驱动库的全部内容包含在 CMIOT.CM32M4xxR_Library 文件夹下，其整体结构及各部分之间的调用关系分别如图 3-1、3-2 所示：

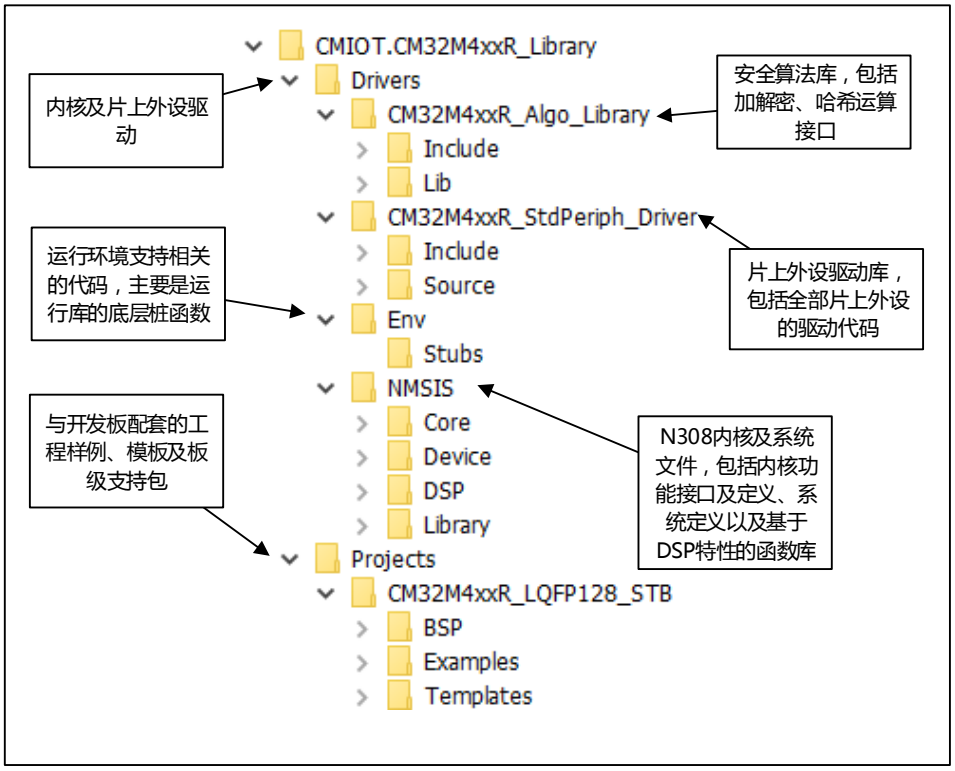

图 3-1 驱动库目录结构

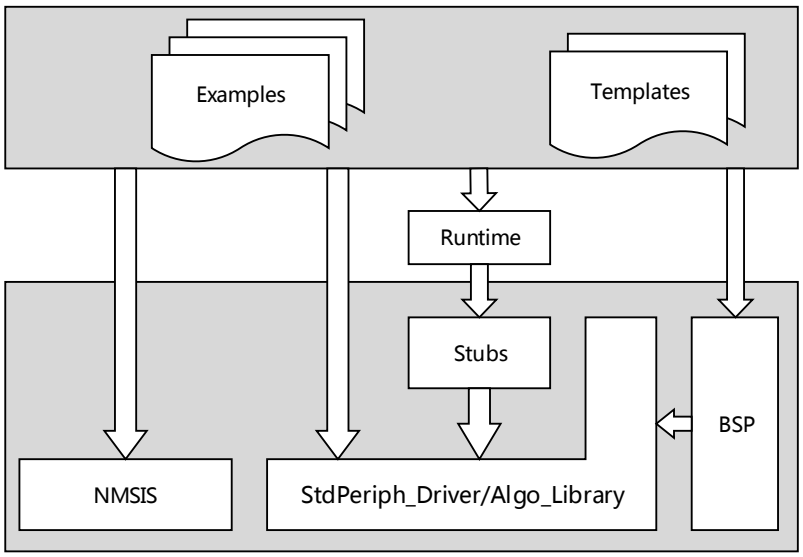

图 3-2 驱动库各部分调用关系

需要注意的是如果使用直接导入的方式创建样例工程，Projects 下的所有例程共享 Drivers 下的驱动，开发者使用过程中如需修改驱动，建议将要修改的文件拷贝到相应工程目录下单独修改，编译时排除原文件，以免影响其他工程。关于工程导入方式的详细说明请参考《CM32M4xxR 系列芯片快速上手指南》的 5.1 章节。

3.2 驱动文件

Drivers 包含以下一系列驱动文件：

表 3-1 安全算法库文件

文件	描述
cm32m4xxr_aes.h	AES 算法头文件 包含数据结构、状态、参数定义，及对外接口声明。
cm32m4xxr_des.h	DES 算法头文件 包含数据结构、状态、参数定义，及对外接口声明。
cm32m4xxr_hash.h	HASH 算法头文件 包含数据结构、状态、参数定义，及对外接口声明。
cm32m4xxr_sm4.h	SM4 算法头文件 包含数据结构、状态、参数定义，及对外接口声明。

cm32m4xxr_algo_common.h	算法公共头文件 包含数据结构、状态、参数定义，及对外接口声明。
libcm32m4xxr_aes_<isa>.a libcm32m4xxr_des_<isa>.a libcm32m4xxr_hash_<isa>.a libcm32m4xxr_sm4_<isa>.a libcm32m4xxr_common_<isa>.a	各算法实现的静态库 <isa>表示指令集，如 rv32imafc，使用时应根据内核的指令集特性及编译选项选择相应版本的库，原则是内核支持的指令集应是编译时选择的指令集的超集，且<isa>应与编译时选择的指令集一致。具体可参考 Projects/CM32M4xxR_LQFP128_STB/Examples/ALGO 中的例程的使用方法。

表 3-2 标准外设驱动文件

文件	描述
cm32m4xxr_<ppp>.h	标准外设（模块）驱动头文件 包含数据结构、状态、参数、宏定义，及对外接口声明。 如：cm32m4xxr_adc.h、cm32m4xxr_tim.h。
cm32m4xxr_<ppp>.c	标准外设驱动源文件 包含驱动接口实现。 如：cm32m4xxr_adc.c、cm32m4xxr_tim.c。
misc.c	辅助驱动源文件 包含标准驱动以外且对开发有帮助的功能接口实现，如 PMP 配置、精确延时等。

表 3-3 内核及系统文件

文件	描述
core_compatiable.h	内核兼容头文件 包含与 ARM CMSIS-Core 兼容的内核功能接口。 关于内核文件的详细介绍，请参考芯来官方文档（https://doc.nucleisys.com/nmsis/）。
core_feature_base.h	内核基础特性头文件 包含内核的 CSR 定义与基础的内核功能接口。
core_feature_cache.h	内核 CACHE 特性头文件 包含内核的 CACHE 相关的数据结构、命令、状态及功能接口定义。 注意：CACHE 特性与内核配置有关，且代码受__ICACHE_PRESENT

	和__DCACHE_PRESENT 宏的控制。
core_feature_dsp.h	内核 DSP 特性头文件 包含 DSP 指令接口定义。 注意：DSP 特性与内核配置有关，且代码受__DSP_PRESENT 宏的控制。
core_feature_eclic.h	内核 ECLIC 特性头文件 包含 ECLIC 相关的寄存器、参数及功能接口定义。 注意：ECLIC 特性与内核配置有关，且代码受__ECLIC_PRESENT 宏的控制。
core_feature_fpu.h	内核 FPU 特性头文件 包含 FPU 功能接口定义。 注意：FPU 特性与内核配置有关，且代码受__FPU_PRESENT 宏的控制。
core_feature_pmp/spmp.h	内核 PMP/sPMP 特性头文件 包含 PMP/sPMP 功能接口定义。 注意：PMP/sPMP 特性与内核配置有关，且代码受__PMP_PRESENT 和__sPMP_PRESENT 宏的控制。另外，使用 sPMP 功能的前提是内核支持 TEE 特性。
core_feature_timer.h	包含 TIMER 相关的寄存器、参数及功能接口定义。 注意：TIMER 特性与内核配置有关，且代码受__SYSTIMER_PRESENT 的控制。
cm32m4xxr.h	系统定义头文件 包含所有片上外设的寄存器数据结构定义，基地址、寄存器位域声明，以及中断号、异常号和内核特性的声明。此外，还包含部分预定义的外设配置参数。
cm32m4xxr_conf.h	系统配置头文件 包含驱动（模块）代码的裁剪控制、时钟定义及 ASSERT 使能控制。
cm32m4xxr_def.h	包含基本整数类型的重定义，以及通用状态、标志和寄存器操作宏的定义。
nuclei_sdk_soc.h	系统头文件 包含 cm32m4xxr.h、cm32m4xxr_conf.h 和 cm32m4xxr_def.h 三个头文件，开发者通常只需在应用文件中包含该文件即可访问到芯片所

	有外设资源。
system_cm32m4xxr.c	系统启动头文件 包含系统初始化接口、中断和异常初始化接口的定义。这些接口，如 SystemInit()会在系统启动的过程中被启动代码调用使系统准备好执行用户程序（main）。
startup_cm32m4xxr.S	启动文件 包含中断向量表及启动代码，执行异常/NMI 入口地址和中断向量表地址初始化、代码加载、系统初始化、数据初始化及其他一些需要在 main 之前完成的操作，并引导程序至 main。
intexc_cm32m4xxr.S	中断及异常文件 包含异常/NMI 入口函数、（非向量）中断公共入口函数的定义，以及用于保存上下文的宏定义。
gcc_cm32m4xxr_flash.ld gcc_cm32m4xxr_flashxip.ld gcc_cm32m4xxr_ilm.ld	链接脚本 三个链接脚本对应三种不同的程序下载及运行方式，开发者可根据需要灵活选用： _flashxip：表示程序下载至 flash 中，并在 flash 中执行，这是最常用的方式。 _flash 表示程序下载至 flash 中，在 ram 中执行，启动代码负责将用户代码从 flash 搬运至 ram。 _ilm 表示程序下载至 ilm 中，在 ilm 中执行，这种情况下代码掉电不保存，通常用于调试。 注意：cm32m4xxr 中并未配置专门的 ilm 存储器，这里说的 ilm 仍指 ram，在上述第三种方式中，由于代码和数据均位于 ram 中，因此需要按需合理划分二者空间，即调整脚本中__ILM_RAM_BASE、__ILM_RAM_SIZE、__RAM_BASE 和__RAM_SIZE 的值，保证__RAM_BASE + __RAM_SIZE 不大于总 ram 大小，两块空间地址连续且满足必要的对齐条件即可。

除上面所列举的常用文件之外，Drivers 还包括运行库的桩函数文件以及 DSP 函数库，对于前者中常用的 write()、read()函数，驱动库中专门提供了_put_char()和_get_char()虚函数接口和实现示例，参照示例开发者无需更改桩函数文件即可方便地对 printf()和 scanf()进行重映射；对于后者驱动库中也提供了一系列例程供开发者参

考，示例和例程均位于 Projects 下。

3.3 驱动接口

驱动的实现遵循一致的编程规范，对开发者尽量做到见名知意、拿来即用。另外，为方便有阅读或更改源码需求的开发者阅读或修改代码，驱动从文件、函数到具体语句及定义均配有详尽的注释，且避免使用不易理解的实现方式。

接口命名上，标准外设驱动和安全算法库的接口命名整体遵循“<模块>_<操作><对象>”的命名方式，其中<模块>全大写表示所属的外设或算法模块，<操作>首字母大写表示要执行的操作，<对象>首字母大写或全大写表示操作的对象或内容，如 I2C_SendData()、I2C_RecvData()；若操作无特定对象或操作本身意义明确则省略<对象>字段，如 I2C_Init()、I2C_Enable()、SM4_Crypto()、HASH_Start()。

接口分类上，标准外设驱动接口可分为如下几类：

- 初始化及去初始化接口：对外设进行初始配置或将其置于复位状态，如 USART_Init()、USART_DeInit()。
- 功能配置及操作接口：配置外设的特定功能及执行特定操作，如 USART_ConfigInt()、USART_EnableDMA()、GPIO_ResetBits()。
- 状态/标志查询及清除接口：查询或清除外设的状态或中断标志，如 ADC_GetFlagStatus()、ADC_ClearFlag()、ADC_GetIntStatus()。
- 其他辅助接口：进行一些辅助性的操作，如 XFMC_InitNandStruct()。

视外设功能及使用需要，某驱动模块的接口可能包含或不包含上述全部分类。安全算法模块的功能相对专一，接口较少，不再赘述。

内核相关接口由芯来定义和提供，遵循自有规范，开发者可以参考其官方文档（https://doc.nucleisys.com/nmsis/）。

3.4 数据结构

合理的数据结构设计能够将复杂多样的外设配置进行抽象、归类，方便开发者理解和使用驱动，与接口设计具有同等重要的意义。标准外设驱动中最重要的两个数据结构是外设寄存器和初始化数据结构。

寄存器数据结构是外设实际寄存器及其布局的抽象，位于 cm32m4xxr.h 中，以“<模块>_Module”的方式命名，如 RCC_Module、RTC_Module。结合外设基地址，开发者可以方便地通过该数据结构访问外设寄存器，如：

```
typedef struct
{                                   /*!< RCC Module,                                          Base Address: 0x40021000 */
    __IO uint32_t CTRL;             /*!< RCC clock control register,                             Address offset: 0x00 */
    __IO uint32_t CFG;              /*!< RCC clock configuration register,                       Address offset: 0x04 */
    __IO uint32_t CLKINT;           /*!< RCC clock interrupt register,                           Address offset: 0x08 */
    __IO uint32_t APB2PRST;         /*!< RCC APB2 peripheral reset register,                     Address offset: 0x0C */
    __IO uint32_t APB1PRST;         /*!< RCC APB1 peripheral reset register,                     Address offset: 0x10 */
    __IO uint32_t AHBPCLKEN;        /*!< RCC AHB peripheral clock enable register,               Address offset: 0x14 */
    __IO uint32_t APB2PCLKEN;       /*!< RCC APB2 peripheral clock enable register,              Address offset: 0x18 */
    __IO uint32_t APB1PCLKEN;       /*!< RCC APB1 peripheral clock enable register,              Address offset: 0x1C */
    __IO uint32_t BDCTRL;           /*!< RCC backup domain control register,                     Address offset: 0x20 */
    __IO uint32_t CTRLSTS;          /*!< RCC clock control and status register,                  Address offset: 0x24 */
    __IO uint32_t AHBPRST;          /*!< RCC AHB peripheral reset register,                      Address offset: 0x28 */
    __IO uint32_t CFG2;             /*!< RCC clock configuration register 2,                     Address offset: 0x2C */
    __IO uint32_t CFG3;             /*!< RCC clock configuration register 3,                     Address offset: 0x30 */
} RCC_Module;
```

图 3-3 寄存器数据结构示例

初始化数据结构是外设寄存器中配置参数的抽象，位于 cm32m4xxr_<ppp>.h 中，以“<模块>_<子模块>InitType”的方式命名，如 USART_InitType、TIM_TimeBaseInitType。结合数据结构中各字段的注释以及预定义的参数，开发者无需查阅外设寄存器即可准确初始化及配置外设，如：

```
typedef struct
{
    uint32_t BaudRate;              /*!< This member configures the USART communication baud rate.
                                         The baud rate is computed using the following formula:
                                         - IntegerDivider = ((PCLKx) / (16 * (USART_InitStruct->BaudRate)))
                                         - FractionalDivider = ((IntegerDivider - ((u32) IntegerDivider)) * 16) + 0.5 */

    uint16_t WordLength;            /*!< Specifies the number of data bits transmitted or received in a frame.
                                         This parameter can be a value of @ref USART_Word_Length */

    uint16_t StopBits;              /*!< Specifies the number of stop bits transmitted.
                                         This parameter can be a value of @ref USART_Stop_Bits */

    uint16_t Parity;                /*!< Specifies the parity mode.
                                         This parameter can be a value of @ref Parity
                                         @note When parity is enabled, the computed parity is inserted
                                         at the MSB position of the transmitted data (9th bit when
                                         the word length is set to 9 data bits; 8th bit when the
                                         word length is set to 8 data bits). */

    uint16_t Mode;                  /*!< Specifies wether the Receive or Transmit mode is enabled or disabled.
                                         This parameter can be any combination of @ref Mode */

    uint16_t HardwareFlowControl;   /*!< Specifies whether the hardware flow control mode is enabled
                                         or disabled. This parameter can be a value of @ref USART_Hardware_Flow_Control */
} USART_InitType;
```

图 3-4 初始化数据结构示例

此外，针对特定外设在其驱动头文件中定义符合其使用特性的数据结构以及一些表示操作状态、配置参数的枚举类型，开发者可根据注释说明及示例理解使用。安全算法库中的数据结构内容较少，开发者结合示例即可轻松掌握，不再赘述。

内核相关数据结构由芯来定义和提供，遵循自有规范，开发者可以参考其官方文档（https://doc.nucleisys.com/nmsis/）。

3.5 文件包含关系

系统头文件（nuclei_sdk_soc.h）位于驱动库的上层，并通过包含关系将整个驱动库的通用配置、外设资源及接口声明等纳入其中，开发者在访问外设、调用驱动接口时仅需在应用文件中包含该文件即可。具体包含关系如下：

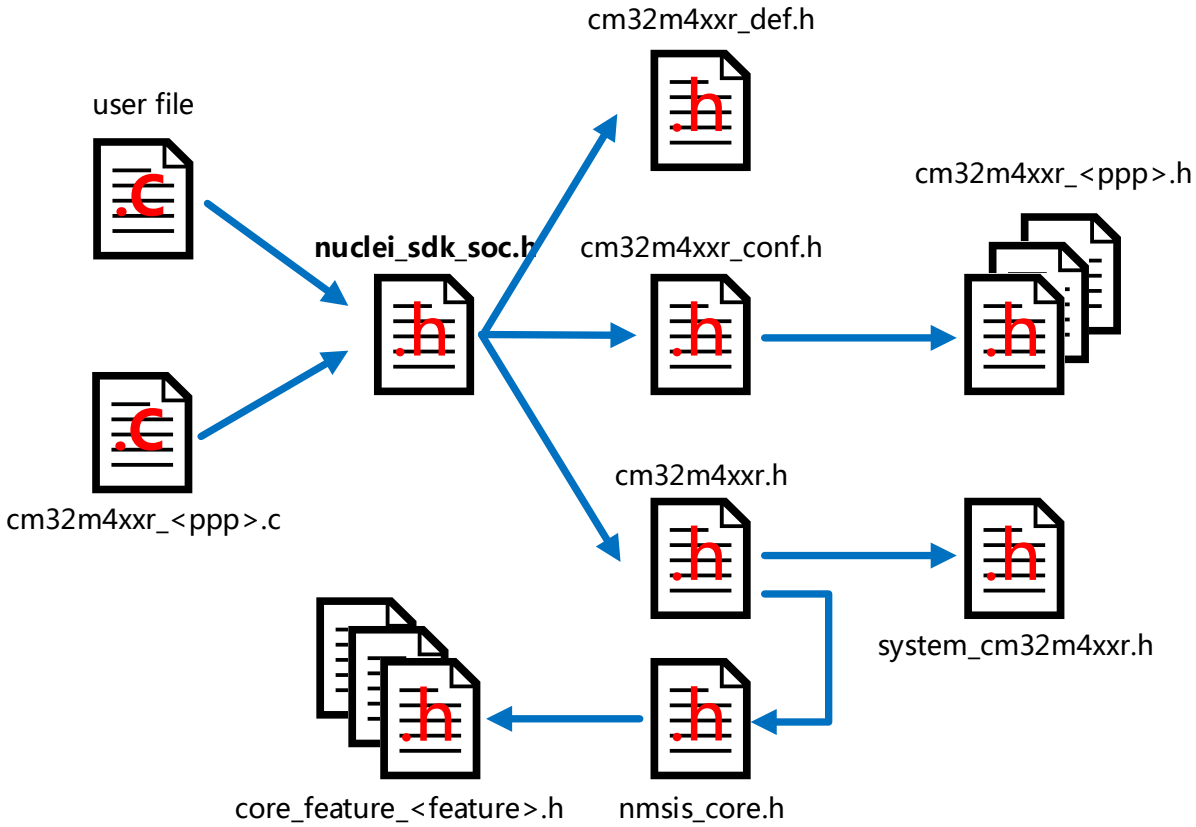

图 3-5 驱动库文件包含关系

3.6 可配置项

cm32m4xxr_conf 和 system_cm32m4xxr.c 文件中定义了一些配置项并设置了默认值，一般情况下无需修改，有需要时开发者可以在满足特定条件的前提下对其进行定制化配置以符合实际应用。可配置项如下表所示：

表 3-4 配置项

配置项	描述	默认值

DRIVER_MODULE_ENABLED _<Module>	驱动模块编译控制
	定义该宏启用相应模块驱动，可视需要裁剪，如要启用 ADC 驱动并裁掉 CAN 驱动则采用如下配置：
	`#define DRIVER_MODULE_ENABLED_ADC`
	`//#define DRIVER_MODULE_ENABLED_CAN`
		ENABLE

HSE_VALUE	外部高速时钟频率（Hz） 如果使用不同频率的外部高速时钟，需要将其修改为相应值。	8000000
HSE_STARTUP_TIMEOUT	外部高速时钟启动至稳定超时值 视实际情况调整该值。	0x500
HSI_VALUE	内部高速时钟频率（Hz）	8000000
LSI_VALUE	内部低速时钟频率（Hz）	40000
LSE_VALUE	外部低速时钟频率（Hz） 如使用不同频率的外部低速时钟，需将其修改为相应值。	32768
LSE_STARTUP_TIMEOUT	外部低速时钟启动至稳定超时值、 视实际情况调整该值。	0x8000
USE_FULL_ASSERT	ASSERT 功能控制 定义该宏则启用 ASSERT 功能，同时应定义 assert_failed()函数。	ENABLE
SYSTEM_CORE_CLOCK	系统主频（Hz） 视需要更改该频率，主频受时钟源及 PLL 资源限制，代码中自动检查其值合法性。	144000000
SYSCLK_SRC	系统时钟源 SYSCLK_SRC 可配置值及其含义为： SYSCLK_USE_HSI：使用 HSI 作为系统时钟源，不启用 PLL。 SYSCLK_USE_HSE：使用 HSE 作为系统时钟源，不启用 PLL。 SYSCLK_USE_HSI_PLL：使用经 PLL 倍频的 HSI 作为系统时钟源。 SYSCLK_USE_HSE_PLL：使用经 PLL 倍频的 HSE	SYSCLK_USE_ HSE_PLL

	作为系统时钟源。
	如要使用经 PLL 倍频的 HSE 作为系统时钟源，则采用以下配置：
	`//#define SYSCLK_SRC  SYSCLK_USE_HSI`
	`//#define SYSCLK_SRC  SYSCLK_USE_HSE`
	`//#define SYSCLK_SRC  SYSCLK_USE_HSI_PLL`
	`#define SYSCLK_SRC  SYSCLK_USE_HSE_PLL`	

3.7 使用方法

图 3-6、3-7 分别展示了驱动库的典型使用方式及示例代码，其中用到了串口驱动及中断：

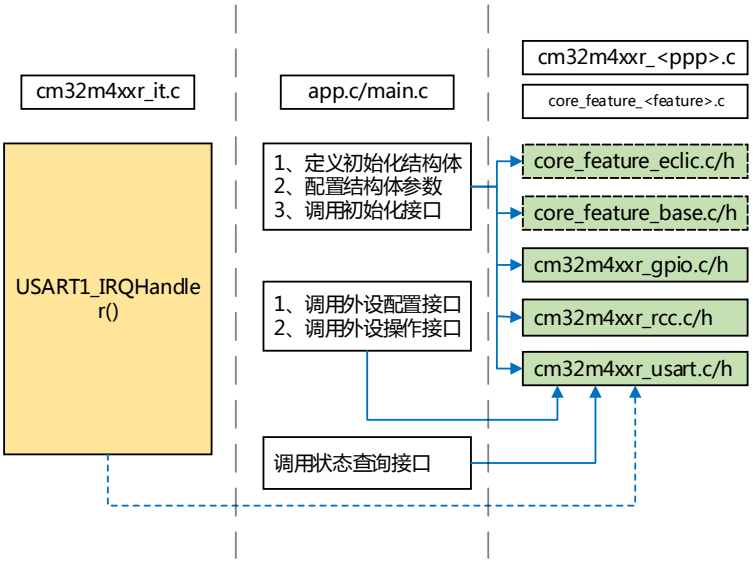

图 3-6 驱动库使用示意

```
int main(void)
{
    /* USART initialization structure */
    USART_InitType USART_InitStructure;
    uint8_t TxBuffer[] = "USART Interrupt Example.";
    uint8_t RxBuffer[sizeof(TxBuffer) - 1];
    uint8_t NbrOfDataToTransfer = NbrOfDataToRead = sizeof(TxBuffer) - 1;
    __IO uint8_t TxCounter = 0x00;
    __IO uint8_t RxCounter = 0x00;

    /* Disable global interrupt */
    __disable_irq();
    /* System Clocks Configuration */
    RCC_Configuration();
    /* ECLIC configuration */
    ECLIC_Configuration();
    /* Configure the GPIO ports */
    GPIO_Configuration();
    /* USART1 configuration ------------------------------------------------------*/
    USART_StructInit(&USART_InitStructure);
    USART_InitStructure.BaudRate            = 115200;
    USART_InitStructure.WordLength          = USART_WL_8B;
    USART_InitStructure.StopBits            = USART_STPB_1;
    USART_InitStructure.Parity              = USART_PE_NO;
    USART_InitStructure.HardwareFlowControl = USART_HFCTRL_NONE;
    USART_InitStructure.Mode                = USART_MODE_RX | USART_MODE_TX;
    /* Configure USART1 */
    USART_Init(USART1, &USART_InitStructure);
    /* Enable USART1 Receive and Transmit interrupts */
    USART_ConfigInt(USART1, USART_INT_RXDNE, ENABLE);
    USART_ConfigInt(USART1, USART_INT_TXDE, ENABLE);
    /* Enable the USART1 */
    USART_Enable(USART1, ENABLE);
    /* Enable global interrupt */
    __enable_irq();
    /* Wait until end of transmission */
    while (RxCounter < NbrOfDataToRead);
    while (TxCounter < NbrOfDataToTransfer);
}
```

```
void USART1_IRQHandler(void)
{
    if (USART_GetIntStatus(USART1, USART_INT_RXDNE) != RESET){
        /* Read one byte from the receive data register */
        RxBuffer1[RxCounter++] = (USART_ReceiveData(USART1) & 0x7F);
        if (RxCounter == NbrOfDataToRead){
            /* Disable the USART1 Receive interrupt */
            USART_ConfigInt(USART1, USART_INT_RXDNE, DISABLE);
        }
    }
    if (USART_GetIntStatus(USART1, USART_INT_TXDE) != RESET){
        /* Write one byte to the transmit data register */
        USART_SendData(USART1, TxBuffer1[TxCounter++]);
        if (TxCounter == NbrOfDataToTransfer){
            /* Disable the USART1 Transmit interrupt */
            USART_ConfigInt(USART1, USART_INT_TXDE, DISABLE);
        }
    }
}
```

图 3-7 驱动库使用代码示例（上图为 main.c、下图为 cm32m4xxr_it.c）

四、 历史版本

版本	日期	修改内容
V1.0	20211013	新建
V2.0	20220221	根据 SDK 2.0.0 版本内容调整部分说明细节