文档编号：AN2017

上海东软载波微电子有限公司

简　介

ES32 SDK

修订历史

版本	修改日期	更改概要
V1.0	2019/9/26	初版
V1.1	2020/12/16	添加 RT-Thread、LittleVGL 中间件及其例程介绍
V1.2	2022/6/30	添加驱动库函数用户手册路径说明

地　址：上海市徐汇区古美路 1515 号凤凰园 12 号楼 3 楼
E-mail：support@essemi.com
电　话：+86-21-60910333
传　真：+86-21-60914991
网　址：http://www.essemi.com/

目　录

内容目录

第1章 简介

1.1 概述

ES32 SDK 是东软载波微电子开发的、面向 ES32 系列微控制器内核以及外设的一套高集成、易使用、面向过程与对象相结合的驱动库，包括 MD、ALD、BSP、中间层驱动。SDK 兼容 CMSIS 标准，可以高效地移植不同的操作系统、文件系统等第三方软件。SDK 集成了多方中间层软件，中间层软件对整个 ES32 系列微控制器保持兼容，能够做到不同平台之间的简单移植。MD 驱动面向过程开发，接近微控制器底层操作，提供用户微控制器寄存器操作简单接口，操作方便，同时 MD 驱动提供内联以及非内联调用函数的方式，最大化用户代码执行效率。ALD 驱动对微控制器外设做抽象化操作，是一套抽象化的面向对象的驱动，用户无需更多关注底层设备的原理，只需要调用相关外设的接口函数即可对外设进行操作，ALD 驱动同时能够为上层应用以及操作系统、文件系统等应用和系统层软件提供接口，提高移植的兼容性。BSP 驱动提供了基于 ES32 微控制器开发板 ES-PDS 的驱动，方便用户进行板级开发。此外为了更加方便用户使用 SDK 开发项目，ES-Link II、USB-Lab、ES-Bridge、SCONS 自动构建工具、ES-PDS 等软件、硬件工具都已经推出。使用 ES32 SDK 开发应用程序可以大大缩减开发时间，降低开发难度，降低项目开发的开销。

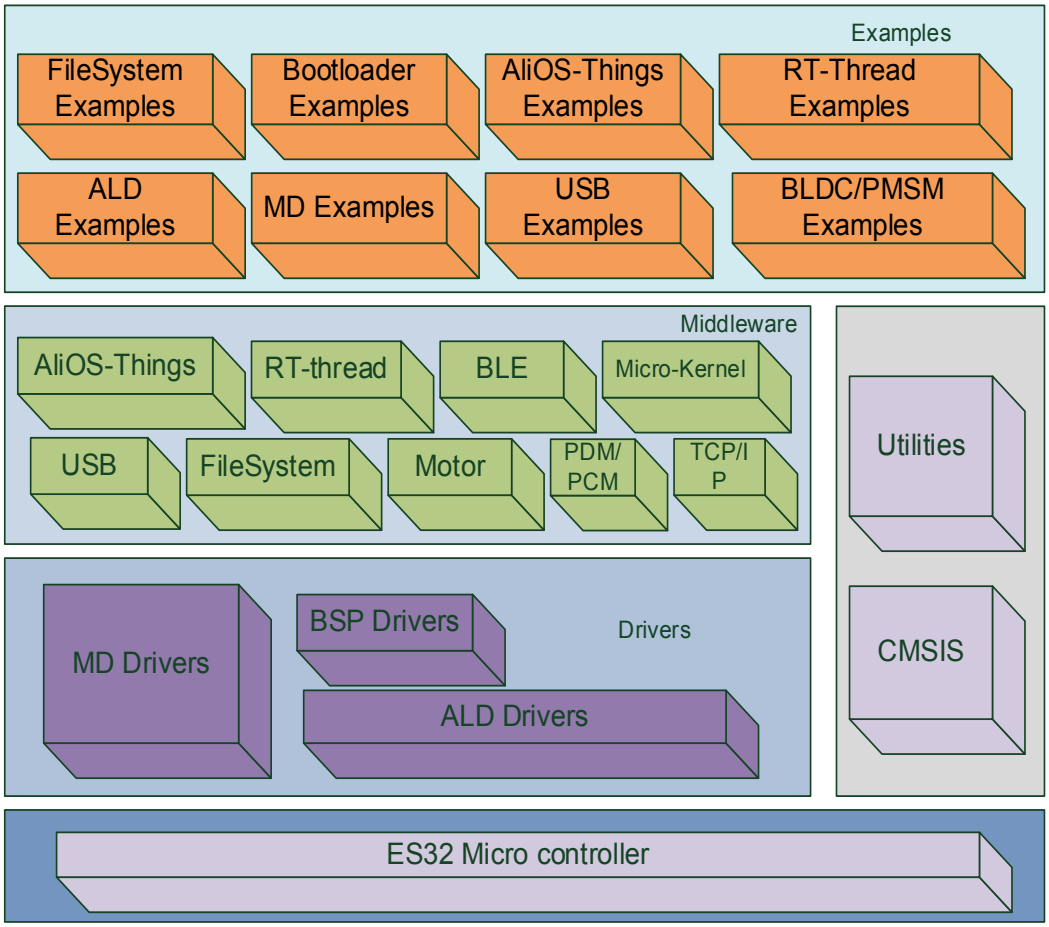

图 1-1 ES32 SDK 框架

1.1.1 特点

ES32 SDK 作为一款针对 ES32 系列微控制器的软件开发套件能够充分发挥 ES32 微控制器的功能与性能，这样一套 SDK 也具有许多优点：

- 高集成

  ES32 SDK 将不同层的软件封装成软件库，集成了包括 MD、ALD、BSP、中间层等驱动，中间层中又集成不同的操作系统、文件系统、USB 协议栈等系统、应用级软件，从微控制器底层驱动到基于微控制器的应用软件都在 SDK 中得到支持。

- 易使用

  使用 ES32 SDK 开发项目简单方便，SDK 中每款芯片的例程丰富、应用领域广泛，并且 ES-Link II、USB-Lab、ES-Bridge、SCONS 自动构建工具等软件、硬件工具同时帮助用户简化开发环境的准备。

- 面向对象与面向过程结合

  ES32 SDK 开发框架覆盖底层硬件驱动到应用程序的全面软件开发套件，在 SDK 中同样给出了面向过程以及面向对象的驱动库，通过不同层次的驱动可以使软件开发最大程度地满足应用需求。例如当用户程序需要进行 I/O 设备管理时，只需简单接口调用，便可以轻松做到上层只需要 read、write 等通用操作而不关注设备硬件的实现原理。

1.1.2 目录结构

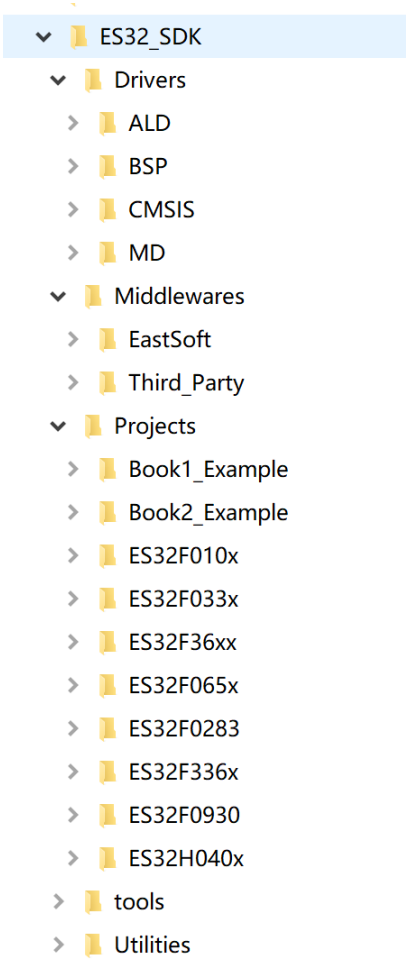

图 1-2 ES32 SDK 目录结构

SDK 目录下有 Drivers、Middlewares、Projects、tools、Utilities 五个文件目录，Drivers 目录下放置的是 MD、ALD、BSP、CMSIS 驱动代码文件以及说明文档，Middlewares 目录下放置的是中间层代码文件以及说明文档，例如 USB 驱动库文件，Projects 目录下放置的是各微控制器芯片的例程，Utilities 目录下放置的是包括 USB 例程配套的 USB PC 端驱动等其它软件工具，tools 文件下放置的是自动构建工具使用的文件，用户无需关心。

1.1.3 支持平台

ES32 SDK 目前支持 MDK、IAR、iDesigner 等平台。

第2章 底层驱动

2.1 概述

ES32 SDK 底层驱动包括 MD、ALD、BSP、CMSIS 驱动，底层驱动从不同的角度给出微控制器操作的接口，用户可以自由选择相应的库，以满足开发的需求。

各芯片的的驱动库函数用户手册以 chm 文件的形式存放在 ES32_SDK\Drivers\ALD\ES32Fxxxx 和 ES32_SDK\Drivers\MD\ES32Fxxxx 文件夹下。

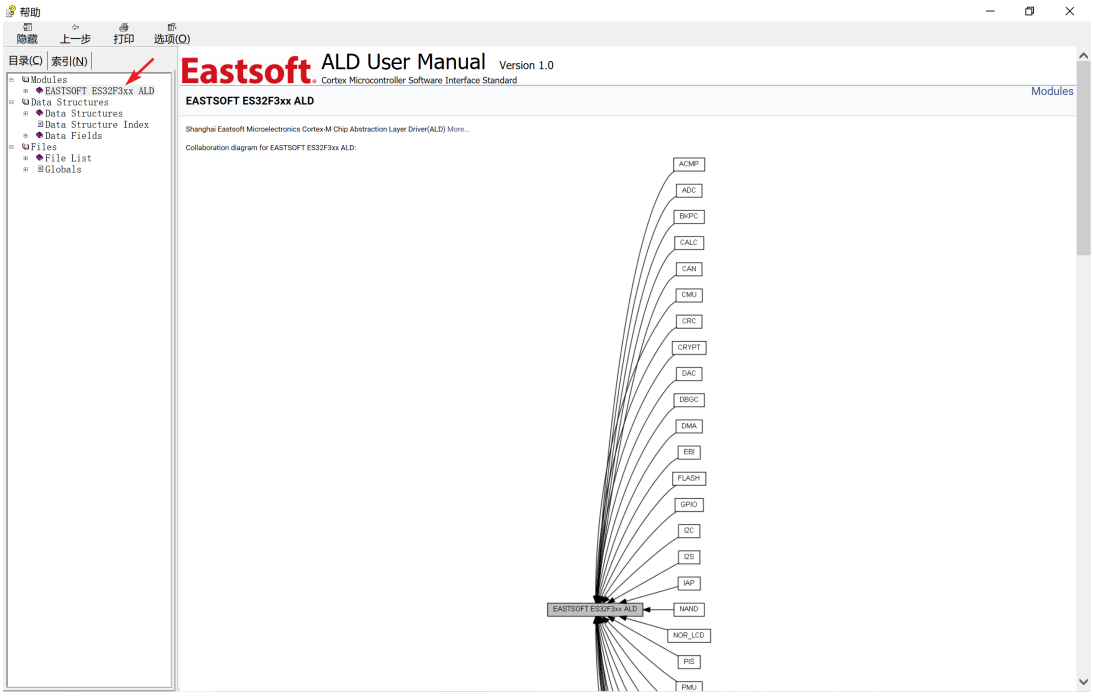

图 2-1 驱动库函数用户手册

2.2 MD驱动

MD（Micro Driver：微驱动）驱动接口主要在头文件中定义，使用内联的方式直接对寄存器做操作，MD 库编译出的程序往往执行效率高、微控制器处理速度快，同时容易控制最终编译出的程序空间。MD 驱动可裁剪，用户可以通过 MD 驱动配置文件来选择需要使用哪些模块的驱动。

使用 MD 驱动操作外设可以分为以下几步：

- 调用 MD 驱动初始化函数，做好 MD 驱动心跳时钟移植；
- 使用 MD 驱动配置文件，选择需要用到的外设模块；
- 调用 MD 驱动相关外设初始化接口，进行外设初始化操作；
- 调用 MD 驱动外设功能性接口，对外设进行操作。

2.3 ALD驱动

ALD（Abstraction Layer Driver：抽象层驱动）驱动屏蔽硬件底层操作，用户无需关心相关外设的硬件原理，更不用自己代码实现外设的工作过程，只需做好相关外设的使用准备即可调用外设功能接口来对外设进行操作。ALD 驱动接口可以分为三方面，初始化接口、中断处理接口、功能接口。初始化接口用来初始化相关外设，在外设使用之前调用；中断处理函数需在外设使用前在相关中断服务函数中被调用，中断处理函数主要用来处理微控制器相关中断，在 ALD 库中给出了每个外设以及内核的中断处理函数，用户只需要进行调用即可；功能接口即每个外设核心功能的接口，比如串行、并行外设总线发送数据。功能接口分为阻塞以及非阻塞类型，在调用非阻塞接口时，操作完成的回调函数由应用程序给出，需要作为参数传递到 ALD 模块驱动中（当然回调函数可以为 NULL），当操作完成时回调函数会被调用。

ALD 驱动对于外设的操作有多种方式，如 DMA、中断、查询等，不同的方式有不同的优点。查询方式直接通过程序控制外设，外设的数据交流完全通过 CPU 来控制，查询的方式操作简单方便，容易实现。中断方式操作减少了 CPU 的等待时间，提高了 CPU 的使用率，外设操作响应快，效率高。中断方式的外设操作接口为非阻塞类型的接口，当外设操作完成后会以回调的方式来通知应用程序。DMA 的方式主要用于串行以及并行外设接口数据的传输，DMA 传输方式可以只在传输前以及传输完成后对外设做相关的配置，传输时无需 CPU 干预，在高速传输、低功耗场景下都能够应用。DMA 传输方式的功能接口也为非阻塞类型的接口，当外设操作完成后会以回调的方式通知应用程序。

ALD 驱动可裁剪，用户可以通过 ALD 驱动的配置文件来选择需要使用到哪些驱动。

使用 ALD 驱动操作外设可以分为以下几步：

- 调用 ALD 驱动初始化函数，做好 ALD 驱动心跳时钟移植；
- 使用 ALD 驱动配置文件，选择需要用到的外设模块；
- 在微控制器中断服务函数中调用 ALD 驱动中已经给出的相关模块的中断处理函数，如果模块使用 DMA 方式进行操作，那么还需要在 DMA 中断服务函数中调用 DMA 的中断处理函数。
- 调用 ALD 驱动相关外设初始化接口，进行外设初始化操作；
- 调用 ALD 驱动外设功能性接口，对外设进行操作。
- 当调用非阻塞类型的功能接口后，回调函数会被调用，进行用户操作完成处理。

2.4 CMSIS

CMSIS 即微控制器软件接口标准，是 Cortex-M 处理器系列的与供应商无关的硬件抽象层，CMSIS 提供了内核与外设、实时操作系统和中间设备之间的通用接口。使用 CMSIS，可以为处理器和外设实现一致且简单的软件接口，从而简化软件的重用、缩短微控制器新开发人员的学习过程，并缩短新设备的上市时间。

CMSIS 可以分为多个软件层次，分别由 ARM 公司、芯片供应商提供。

- 内核设备访问层

  包含了用来访问内核的寄存器设备的名称定义，地址定义和配置函数。同时也为

RTOS(实时操作系统)定义了独立于微控制器的接口，该接口包括调试通道定义。

- 微控制器外设访问层

  提供片上所有外设的定义，包括所有 ES32 微控制器外设寄存器头文件、ES32 微控制器启动文件、ES32 微控制器系统初始化模板文件。

- DSP 库

  优化的信号处理算法，并为 SIMD 指令提供 Cortex-M4 支持。

- RTOS 接口定义

  独立于微控制器的 RTOS 接口，带调试通道。

第3章 软件中间层

3.1 概述

ES32 SDK 软件中间层是在底层驱动的基础上开发出来的一套兼容所有 ES32 微控制器的驱动，软件中间层在不同 ES32 微控制器上使用方法相同。软件中间层包括 USB 协议栈、文件系统、电机库、TCP/IP 协议栈、BLE 协议栈、各种 RTOS(如 RT-Thread)、Micro-Kernel、嵌入式图形界面编程(littleVGL)等驱动。

3.2 USB协议栈

ES32 USB 协议栈提供了一套供上层应用使用的接口，包括 USB 主机以及设备功能接口。协议栈涵盖了目前市场上大部分常用的 USB 主机与设备的驱动，并同时提供给用户用于拓展协议栈驱动的方法，以及复合类设备开发的方法。ES32 USB 协议栈设备类支持 audio、bulk、cdc、dfu、hid、hid 鼠标、hid 键盘、hid 手柄、msc 等，设备驱动支持多种设备的复合，即复合设备类；主机类支持 audio、hid、hid 键盘、hid 鼠标、hub、msc 等。完善的设备类驱动为用户开发 USB 设备提供了极大的便利。本套协议栈属于 ES32_SDK 的一部分，作为软件中间层代码，相关驱动代码可以在 ES32_SDK\firmware\ES32_SDK\Middlewares\EastSoft\usblib 目录下找到，相关应用例程可以在 ES32_SDK\firmware\ES32_SDK\Projects\[相应芯片型号]\Applications\USB 目录下找到。USB 协议栈与 MD 以及 ALD 库对接，在 MD/ALD 的基础上拓展出主机/设备核心驱动层、主机/设备类层驱动、主机/设备层驱动。

USB 协议栈核心层驱动提供了包括枚举、中断管理、上层驱动管理、传输管理、描述符管理、标准请求管理等功能接口，是整个 USB 协议栈工作的核心。各主机/设备类层的驱动提供了 USB 各设备类的描述符解析与创建、设备类请求管理、设备类层数据传输管理、事件管理等功能，可以有效地处理 USB 核心层传来的数据。主机/设备层驱动提供了 USB 具体某个设备的驱动。图 3-1、3-2 展示了 USB 协议栈主机以及从机部分的结构：

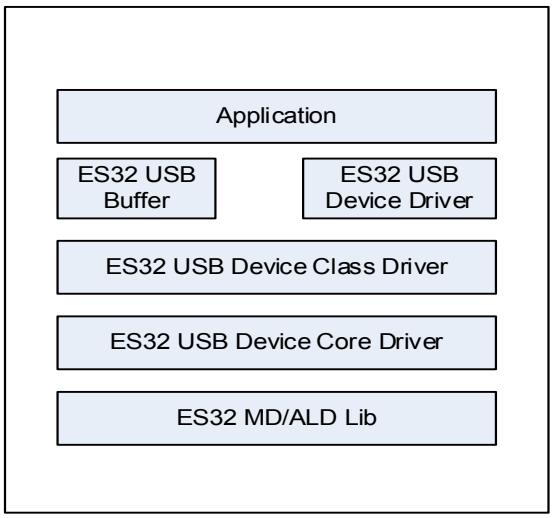

图 3-1 USB 协议栈设备驱动结构

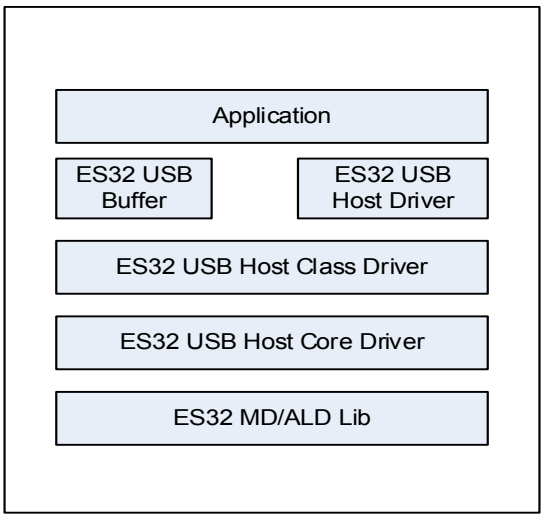

图 3-2 USB 协议栈主机驱动结构

3.2.1 设备部分驱动结构

USB 协议栈设备部分的驱动是建立在 MD/ALD 库基础上的一套三层结构的函数库，

用户可以在此三层结构上开发设备。协议栈分为核心层、设备类层、设备层的驱动。其中核心层负责处理 USB 的中断以及将相应的事件传输给设备类层驱动、USB 数据的初步处理，当有需要传输到设备类层驱动的数据，核心层会以事件回调的方式将数据传达设备类层驱动；设备类层驱动负责 USB 各设备类协议的处理，核心层将相关的事件数据传到设备类层时，设备类会将数据进一步处理，如果设备类下有多个子类设备的话，需要子类用到的数据也会以事件回调的方式传输到设备层；USB 某些设备类下有多个子类，例如 HID 设备类下有 HID 键盘、HID 鼠标等设备，那么这些子类设备的驱动放在设备层中，设备层的功能和相关的子设备相关，如 HID 鼠标设备驱动中就提供了 HID 鼠标的描述符以及数据发送的 API。

此外，USB 某些设备类的驱动在使用时需要传输较大的数据量，为了能够更好地管理这些数据的传输，USB 协议栈提供了一个 Buffer 模块用来管理大的数据的传输，例如 CDC 类的驱动就可以将相关的数据传输 API 对接上 Buffer 模块，从而实现仅对 Buffer 的操作就能够实现数据的传输。

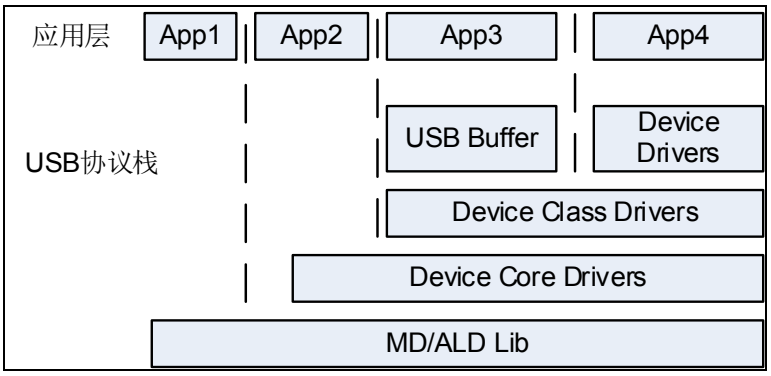

图 3-3 USB 协议栈设备驱动结构

3.2.2 主机部分驱动结构

USB 协议栈主机部分的驱动是建立在 MD/ALD 库基础上的一套三层结构的函数库，用户可以在此三层结构上开发设备。协议栈分为核心层、主机类层、主机层的驱动。其中核心层负责处理 USB 的中断以及将相应的事件传输给主机类层驱动、USB 数据的初步处理，当有需要传输到主机类层驱动的数据，核心层会以事件回调的方式将数据传达主机类层驱动；主机类层驱动负责 USB 各主机类协议的处理，核心层将相关的事件数据传到主机类层时，主机类会将数据进一步处理，如果主机类下有多个子类主机的话，需要子类用到的数据也会以事件回调的方式传输到主机层；USB 某些主机类下有多个子类，例如 HID 主机类下有 HID 键盘、HID 鼠标等主机，那么这些子类主机的驱动放在主机层中，主机层的功能和相关的子主机相关，如 HID 鼠标主机驱动中就提供了 HID 鼠标的描述符以及数据发送的 API。

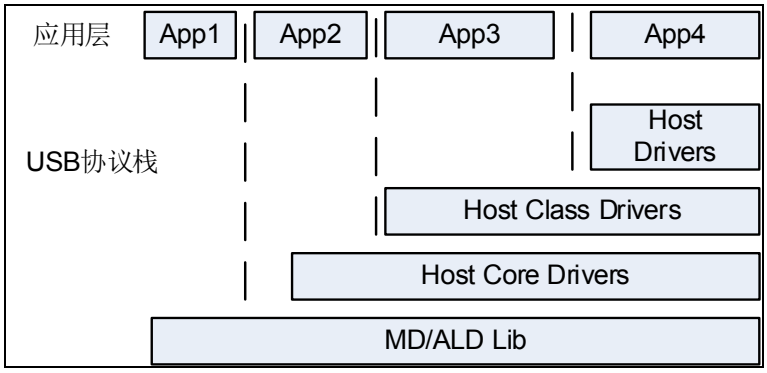

图 3-4 USB 协议栈主机驱动结构

3. 3 Micro-Kernel

Micro-kernel 是一款非抢占微内核框架，为应用层提供多任务环境。微内核主要面向无操作系统的应用开发，提供了一套在无操作系统的环境下应用程序多任务开发的方法，有效地提高应用程序代码效率。与 RTOS 相比，微内核代码量并不大，微内核只是提供了一套应用程序的开发框架，适用于对内存要求较高的应用系统。微内核使用说明请见文档“AN2016_应用笔记_微型操作系统架构及 API”。

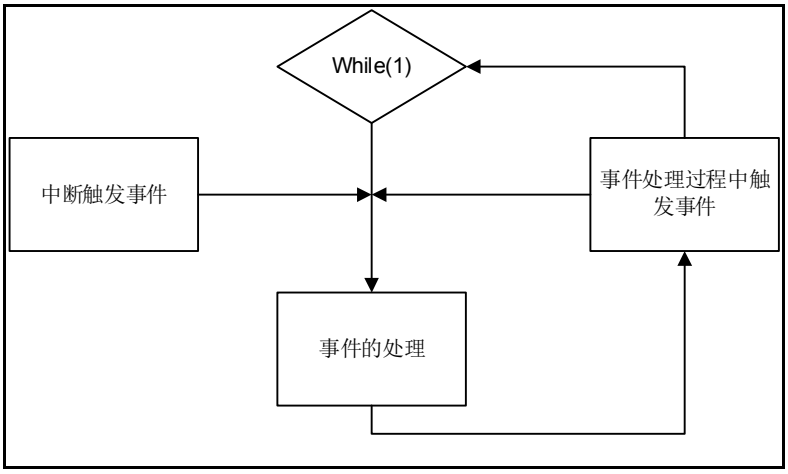

图 3-5 微内核原理

微内核主程序是一个 while(1)循环，循环体中不停的检查任务标志是否被置起，如果置起，执行任务的处理函数，否则进行下一任务标志检查。其中任务标志的置起有两个途径，一个是中断，另外一个是任务处理函数。

如果在该任务被处理之前，多次置起标志位，系统只认为是一次置起，任务的处理过程并不会被打断（中断之后，仍然会回到中断前的任务处理），因此，任务的处理过程不能持续太长时间，如果一个任务并非时间紧急、而处理过程太长，那么建议将该任务优先级调低、整个处理过程分多次执行。比如，任务执行一部分之后，保存执行结果，再将本标志位置起，结束执行。（系统下次会选择优先级高的任务处理）。

定时器也是一个特定的事件，其与任务也有关系，实际上是定时触发指定任务。

内存管理，是将系统中未使用的内存作为内存池，内存的申请与释放即是在内存池中进行操作。

消息队列是指缓存消息的队列。使用方式为在任务执行函数中读取消息队列内的消息，在中断处理程序或其他任务处理函数中将消息放入消息队列。

命令行模块，提供命令行支持，方便 Debug 问题，加快开发进度。

红黑树模块，提供红黑树的使用接口。

第4章 例程方案

4.1 概述

ES32 SDK 给出了基于不同微控制器的例程方案，包括 MD 驱动例程、ALD 驱动例程、应用例程等，应用例程中一部分为软件中间层的例程，例如 USB 例程，一部分为集成方案。

- MD 驱动例程

  MD 驱动例程基于 MD 驱动开发，用于展示 MD 驱动相关接口的使用方法，包括 MD 驱动的心跳时钟的配置方法、MD 驱动模块的选择方法、MD 驱动功能函数的使用方法；

- ALD 驱动例程

  ALD 驱动例程基于 ALD 驱动开发，用于展示 ALD 驱动接口的使用方法，包括 ALD 驱动的心跳时钟的配置方法、ALD 驱动模块的选择方法、ALD 驱动功能函数的使用方法、ALD 中断处理函数的使用方法；

- 应用例程

  应用例程基于 MD、ALD、软件中间层开发，展示了软件中间层的使用方法以及部分微控制器应用方案，例如 USB 协议栈的多个例程包括鼠标设备、键盘设备、声卡设备、鼠标主机等向用户展示了如何使用 USB 协议栈进行开发。RT-Thread 例程包括线程的创建、信号量的使用、消息队列等，向用户展示如何快速使用嵌入式操作系统。LittleVGL 例程包括按钮、对话框、多选框、进度条等控件，向用户展示如何快速使用 LittleVGL 进行嵌入式图形界面开发。