UART 驱动编写指导手册

编写：贺敏 2015 年 3 月 23 日
Review：_年_月_日
审阅:罗侍田

1. 概述

DJYOS 设计通用的串口驱动模型，在此模型的基础上，移植到不同硬件平台时，只需提供若干硬件操作函数，即可完成串口驱动开发，使开发工作变得简单而快速执行效率高。

DJYOS 源代码都有特定的存放位置，

建议文件路径：在 eclipse 工程中的目录中为：

src->OS_code->bsp->cpudrv->src->cpu_peri_uart.c。

相应的头文件目录为：

src->OS_code->bsp->cpudrv-> src->cpu_peri_uart.h。

在文件系统（硬盘）中的目录结构是：

djysrc\bsp\cpudrv\cpu_name\src\cpu_peri_uart.c。

如果 UART 为 CPU 所在板件的板级外设，则

建议文件路径：在 eclipse 工程中的目录中为：

src->OS_code->bsp->boarddrv->src->chip_name.c。

相应的头文件目录为：

src->OS_code->bsp-> boarddrv -> src-> chip_name.h。

在文件系统（硬盘）中的目录结构是：

djysrc\bsp \chipdrv\chip_name\chip_name.c。

本指导手册简明扼要的介绍了都江堰操作系统的 UART 驱动模型，着重讲解了在此驱动架构下编写具体 UART 控制芯片的驱动编写方法。

2. 驱动架构

UART 驱动架构如图 2-1所示。该模型包括通用设备驱动 driver.c 及 UART 通用层 uart.c 及底层硬件驱动部分 cpu_peri_uart.c。它具有明显分层结构，顶层是应用程序，它通过访问设备驱动接口层访问串口通用接层；串口通用层是底层驱动与设备驱动之间桥梁，它调用了若干钩子函数，通过钩子函数间接访问实现串口收发。最下面一层是由 BSP 设计者提供的硬件驱动，BSP 设计者只需要按照要求实现钩子函数。

BSP 驱动程序人员的工作是编写图 2-1中底层驱动部分代码，它是根据具体 UART 控制器（即不同的 CPU）实现与硬件相关的 UART 寄存器级别的操作。

驱动编写的主要工作，是：

1、 初始化 uart 硬件。

2、 实现 3 个回调函数。

3、 若使用中断，则还要实现中断服务函数。

4、 若 printk 使用 uart（绝大多数情况是这样），则要实现 PutChark、Putsk、GetChark、Getsk 这几个函数。

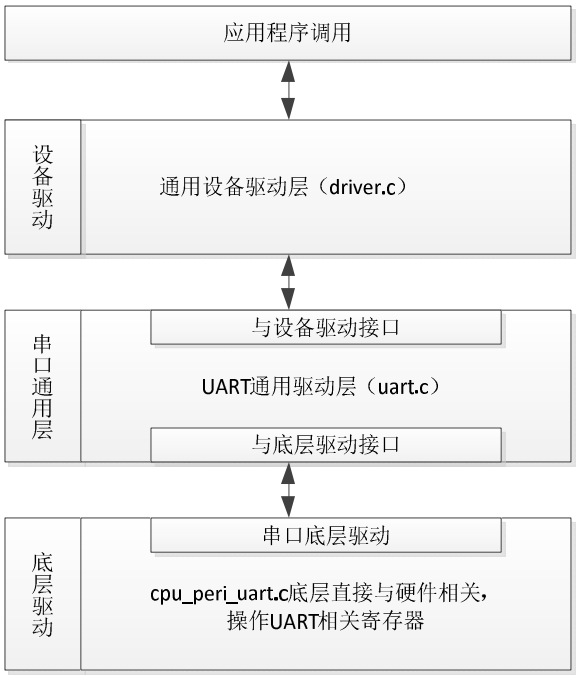

图 2-1UART 驱动架构图

3. UART 驱动接口

3.1. 初始化函数

3.1.1. Step1：初始化硬件

1、 包括默认的参数配置，如 IO、波特率、停止位、奇偶校验位等；

2、 挂载 UART 中断到中断系统，并配置 UART 的中断类型，如配置为异步信号；

3.1.2. Step2：初始化参数结构体

UART 初始化参数数据类型为 truct tagUartParam，初始化该数据类型变量的成员，并调用函数 UART_InstallPort()，注册 UART 为设备，并将它添加到资源列表。

对该函数的参数 struct tagUartParam 作如**错误！未找到引用源。**说明。

代码 3-1 UART 参数初始化结构体

```
// 串口模块初始化结构体
struct tagUartParam
{
    const char *Name;                   //UART名称，如UART0
    u32 TxRingBufLen;                   //发送缓冲区配置字节数
    u32 RxRingBufLen;                   //接收缓冲区配置字节数
    u32 Baud;                           //默认的波特率
    ptu32_t UartPortTag;                //UART私有标签，如寄存器基址
    UartStartSend StartSend;            //启动发送回调函数指针
    UartDirectSend DirectlySend;        //直接轮询发送回调函数指针
    UartControl UartCtrl;               //控制函数回调函数指针
};
```

DJYOS 通用驱动架构是先将用户发送数据保存到发送环形缓冲区，然后通过操作系统的通信方式（如信号量通信）通知驱动层将发送环形缓冲区中的数据写入 UART 控制器的发送寄存器；同理，驱动接收到的数据缓存到接收环形缓冲区，然后通知应用层读取接收环形缓冲区数据。

发送和接收缓冲区的初始化由 UART_InstallPort()完成，BSP 驱动编程人员只需配置 struct tagUartParam 中的参数 TxRingBufLen 和 RxRingBufLen，它们分别是配置发送和接收缓冲区的字节长度。此外，UartPortTag 为私有标签，对于 UART，需赋值为 UART 控制器的寄存器基址。

UART 三个回调函数参数的原型如**错误！未找到引用源。**所示，其中 PrivateTag 就是结构体中 UART 的私有标签，寄存器基址。

代码 3-2 UART 回调函数类型定义

```
typedef u32 (* UartStartSend)(ptu32_t PrivateTag,u32 timeout);
typedef u32 (* UartDirectSend)(ptu32_t PrivateTag,u8 *send_buf,u32 len,u32
timeout);
typedef ptu32_t (*UartControl)(ptu32_t PrivateTag,u32 cmd, u32 data1,u32
data2);
```

3.1.3. Step3：挂载设备

挂载设备 API 函数 UART_InstallPort()，该函数完成了如下功能：

1、 初始化串口发送或接收所需的信号量或互斥量；

2、 动态分配发送和接收数据的环形缓冲区空间，并初始化环形缓冲区；

3、 动态分配 UART 的控制块 UCB，并将 UART 添加为通用设备（即 driver）；

4、 操作成功，则返回 UCB 指针，否则，返回 NULL。

值得注意的是，UART 驱动编程人员对添加串口设备的函数返回值的判断是极其重要的步骤，并将有效的返回值保存为静态变量。如**错误！未找到引用源。**所示，UART_Param 是已经初始化好的参数结构类型，pUartCB 是定义在 C 文件中类型为 static struct tagUartCB *的静态变量数组，UART_InstallPort()返回值保存到该静态变量。

代码 3-3 添加串口设备

```
    pUartCB[serial_no] = UART_InstallPort(&UART_Param);
    if( pUartCB[serial_no] == NULL)
        return 0;
    else
        return 1;
```

3.2. 回调函数

DJYOS 的 UART 驱动架构采用注册回调函数的方式，能有效的将串口通用层和硬件驱动层分开，实现模块化编程。本章将对回调函数的编程要求进行详细的说明，它们是 BSP 驱动开发人员应该着重关心的部分。

3.2.1. 启动发送

static u32 __UART_SendStart (tagUartReg *Reg,u32 Timeout)

参数：

Reg：UART 控制寄存器基址。

Timeout：超时时间，单位 us。

返回：发送字节数。

说明：该函数的关键在于启动发送功能，如触发发送中断，在中断实现发送数据的功能。比较通用的方法是填写 FIFO 深度的数据，触发发送中断，部分 UART 控制器能够实现不发送数据仍然能触发发送中断的功能，该函数将得到进一步的简化。

部分核心示例代码如代码 3-4所示，调用 UART_PortRead 读取发送环形缓冲区数据，数据大小为 FIFO 深度的字节数，并填写到 FIFO，然后使能中断。注意，若启动发送时 UART 硬件缓冲区非空，则无须发送触发发送中断。

代码 3-4 启动发送

```
__UART_TxIntDisable(Reg);
if(__UART_TxTranEmpty(Reg))
{
    fifodep = UART_PortRead(pUartCB[port],ch,fifodep,0);//读FIFO大小数据
    for(num = 0; num < fifodep; num++)
    {
        Reg->D = ch[num];
    }
}
__UART_TxIntEnable(Reg);
```

3.2.2. 直接发送

static u32 __UART_SendDirectly(tagUartReg *Reg,u8 *SendBuf,u32 Len,u32 Timeout)

参数：

Reg：UART 控制寄存器基址。

SendBuf： 发送数据缓冲地址。

Len：发送字节数。

Timeout：超时时间，单位 us。

返回：发送字节数。

说明：串口轮询发送的功能，主要实现操作系统未启动时的串口发送功能。由于轮询发送是比较耗时的发送方式，因此，有必要对发送过程进行超时处理，同时，发送过程必须关闭中断。

部分核心示例代码如代码 3-5所示，判断发送缓冲区为空后，往硬件发送寄存器中写数据，直到发送完成，若等待超时，则退出循环。发送过程必须关闭中断。

代码 3-5 轮询发送

```
__UART_TxIntDisable(Reg);
for(result=0; result < len; result ++)
{
    // 超时或者发送缓冲为空时退出
    while((false == __UART_TxTranEmpty(Reg))&& (timeout > 0))
    {
        timeout--;
        Djy_DelayUs(1);
    }
    if(timeout == 0)
        break;
    Reg->D = send_buf[result];
}
__UART_TxIntEnable(Reg);
```

3.2.3. 控制函数

static ptu32_t __UART_Ctrl(tagUartReg *Reg,u32 Cmd, u32 Data1,u32 Data2)

参数：

Reg：UART 控制寄存器基址。

Cmd：串口命令字，参见源码 uart.h 头文件定义。

Data1、Data2：含义依 Cmd 而定。

返回：0，执行成功，其他，自定义。

说明：底层驱动的控制函数主要完成与具体硬件相关的控制功能，如设置波特率、配置串口参数、暂停启动串口等。UART 控制命令及参数说明如表 3-1所示，需要注意，并非所有的命令都需要用到，驱动编程人员须根据实际情况选择使用的命令。

表 3-1 UART 控制命令

命令	参数 1	参数 2	说明
CN_UART_START	无意义	无意义	使能串口
CN_UART_STOP	无意义	无意义	禁止串口
CN_UART_SET_BAUD	波特率	无意义	设置波特率
CN_UART_COM_SET	struct tagCOMParam	无意义	设置串口参数
CN_UART_SEND_DATA	无意义	无意义	发送使能
CN_UART_RECV_DATA	无意义	无意义	接收使能
CN_UART_COMPLETED_SEND	无意义	无意义	完成发送
CN_UART_RX_PAUSE	无意义	无意义	停止接收
CN_UART_RX_RESUME	无意义	无意义	恢复接收
CN_UART_RECV_HARD_LEVEL	字节数	无意义	设置接收 FIFO 大小
CN_UART_SEND_HARD_LEVEL	字节数	无意义	设置发送 FIFO 大小
CN_UART_EN_RTS	无意义	无意义	使能 RTS
CN_UART_DIS_RTS	无意义	无意义	禁止 RTS
CN_UART_EN_CTS	无意义	无意义	使能 CTS
CN_UART_DIS_CTS	无意义	无意义	禁止 CTS
CN_UART_DMA_USED	无意义	无意义	使能 DMA
CN_UART_DMA_UNUSED	无意义	无意义	失能 DMA
CN_UART_HALF_DUPLEX_SEND	无意义	无意义	半双工发送
CN_UART_HALF_DUPLEX_RECV	无意义	无意义	半双工接收
CN_UART_SETTO_ALL_DUPLEX	无意义	无意义	全双工

3.3. 中断函数

因为采用中断的方式收发数据效率更高，对 CPU 的消耗低，所以建议驱动编写人员采用中断的方式实现 UART 的收发。

首先，必须将 UART 的中断号添加到 critical 文件夹下面的 C 文件 critical.c 中 tg_IntUsed 数组，使中断系统初始化时，将对应的 UART 中断号添加到中断系统中；其次，UART 初始化时，配置 UART 中断类型和触发条件等。最后，完成中断服务函数 UART_ISR。

3.3.1. 中断实现过程

比较典型的 UART 中断函数实现过程如图 3-1所示。

在中断服务函数内部，通过读中断标志位判断中断类型，并进入不同的处理过程。接收中断时，将接收数据写入接收缓冲区；发送中断表示当前硬件 FIFO 中数据已经发送完成，则继续读发送缓冲区数据，直至发送结束；帧错误中断是指发生了校验、帧格式或起始结束位错误时的中断。无论是发送还是接收中断，都需要注意清中断标志位。

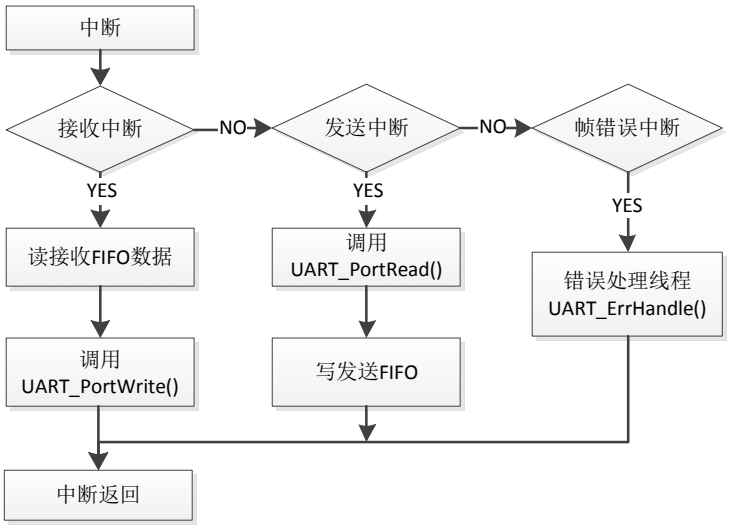

图 3-1 UART 中断流程图

3.3.2. 注意事项

中断服务函数的实现过程因 UART 控制器的不同，处理情况也有所差异，列出如下注意事项，以供参考。

1、 中断服务函数中要注意通过判断中断标志来区分中断类型，并及时清中断标志；
2、 中断类型为发送中断时，UART_PortRead()返回为 0，则表示发送环形缓冲区中无数据，建议关闭发送中断；
3、 帧错误处理方式由应用程序通过 Ctrl 函数注册，登记发生错误时弹出的事件。

4. 移植建议

由于大部分的 UART 控制器的设计基本相似，因此，BSP 程序人员可采取下面的步骤快速的完成 DJYOS 驱动架构下 UART 底层驱动的开发。

1、 拷贝其他工程已测试通过的 UART 驱动文件 cpu_peri_uart.c/cpu_peri_uart.h；
2、 添加 UART 的中断号到 critical.c 文件下面 tg_IntUsed 数组；
3、 修改 `cpu_peri_uart.c/cpu_peri_uart.h` 中与具体 `UART` 寄存器相关的部分；
4、 实现中断收发数据，并测试通过。