# 嵌入式系统实验报告



|  |  |
| --- | --- |
| 实验名称： | UART 串口通信与 SHELL |
| 姓 名： | 江姝潼 |
| 学 号： | 2019211653 |
| 学 院(系)： | 计算机学院 |
| 专 业： | 网络工程 |
| 指导教师： | 刘健培、戴志涛 |

2021年 12 月 8 日

# 实验目的

* 通过 FSM4 实验板了解实验的软硬件环境，熟悉 MDK 开发环境和STM32CubeMx 开发工具的使用。
* 学会 STM32 USART 的基本使用方法。
* 掌握轮询、中断、DMA 方式传输 STM32 USART 数据的编程方式。
* 重点掌握 STM32 中断处理方式。
* 了解 I/O 驱动与系统/应用程序的关系

# 实验环境

* FS-STM32F407开发平台
* ST-Link 仿真器
* RealView MDK5.23集成开发软件
* STM32CUBEMX图形开发软件
* PC机Window7/8/10 (32/64bit)
* 串口调试工具

# 实验要求

* 使用轮询、中断、DMA 三种方式实现 USART3 与 shell 的驱动接口。
* 参考代码已经实现了轮询/DMA 实现 shell 接收/发送的功能，参考其实现方式，实现通过中断实现 shell 收发的功能（中断方式的驱动+中断服务程序）
* 测试：可以在主机的串口软件中通过串口操作 shell。

# 实验原理

1. **UART串口通信协议**

嵌入式系统中，设备间一般采用并行或者串行的方式实现通信和数据交换。并行通信同时使用多条数据传输数据，串行通信使用一条数据线将数据一位一位一次传输。在数据通信过程中，发送端和接收端之间相互协调的方式有两种：同步与异步。同步通信通过在收发端使用共同的时钟信号保持协调（即两端有一根时钟信号线相连），异步通信则不存在共同时钟，而是通过“握手”信号协调数据收发，有的协议使用单独的握手信号线，而有的则是将数据以指定的格式打包为数据帧进行传输，并在数据帧的首尾使用特殊的 bit 模式（起始位、停止位）界定数据帧的边界，从而实现收发间的协调。本章实验的 UART 即属于异步串行通信。

在STM３２中，通用同步异步收发器(Universal Synchronous Asynchronous Receiver and Transmitter)是一个串行通信设备，可以灵活地与外部设备进行全双工数据交换。有别于 USART 还有一个 UART(Universal Asynchronous Receiver and Transmitter)，它是在 USART 基础上裁剪掉了同步通信功能，只有异步通信。 简单区分同步和异步就是看通信时需不需要对外提供时钟输出，我们平时用的串口通信基本都是 UART。

串行通信一般是以帧格式传输数据，即是一帧一帧的传输，每帧包含有起始信号、数据信息、停止信息，可能还有校验信息。USART 就是对这些传输参数有具体规定，当然也不是只有唯一一个参数值，很多参数值都可以自定义设置，只是增强它的兼容性。

USART 满足外部设备对工业标准 NRZ 异步串行数据格式的要求，并且使用了小数波特率发生器， 可以提供多种波特率，使得它的应用更加广泛。USART支持同步单向通信和半双工单线通信；还支持局域互连网络 LIN、智能卡(SmartCard)协议与 lrDA(红外线数据协会) SIR ENDEC 规范。

USART 支持使用 DMA，可实现高速数据通信。USART 在 STM32 应用最多莫过于“打印”程序信息，一般在硬件设计时都会预留一个 USART 通信接口连接电脑，用于在调试程序时可以把一些调试信息“打印”在电脑端的串口调试助手工具上，从而了解程序运行是否正确、出错位置等等。

USART的主要构造如下：

1. 功能引脚

TX： 发送数据输出引脚。

RX： 接收数据输入引脚。

SW\_RX： 数据接收引脚，只用于单线和智能卡模式，属于内部引脚，没有具体外部引脚。

nRTS： 请求以发送(Request To Send)，n 表示低电平有效。如果使能 RTS流控制，当 USART 接收器准备好接收新数据时就会将 nRTS 变成低电平； 当接收寄存器已满时，nRTS 将被设置为高电平。该引脚只适用于硬件流控制。

nCTS： 清除以发送(Clear To Send)，n 表示低电平有效。如果使能 CTS 流控制，发送器在发送下一帧数据之前会检测 nCTS 引脚， 如果为低电平，表示可以发送数据，如果为高电平则在发送完当前数据帧之后停止发送。该引脚只适用于硬件流控制。

SCLK： 发送器时钟输出引脚。这个引脚仅适用于同步模式。

1. 数据寄存器

USART 数据寄存器(USART\_DR)只有低 9 位有效，并且第 9 位数据是否有效要取决于 USART 控制寄存器 1(USART\_CR1)的 M 位设置，当 M 位为 0 时表示 8 位数据字长，当 M 位为 1 表示 9 位数据字长，我们一般使用 8 位数据字长。

USART\_DR 包含了已发送的数据或者接收到的数据。USART\_DR 实际是包含了两个寄存器，一个专门用于发送的可写 TDR， 一个专门用于接收的可读RDR。当进行发送操作时，往 USART\_DR 写入数据会自动存储在 TDR 内；当进行读取操作时，向 USART\_DR 读取数据会自动提取 RDR 数据。TDR 和 RDR 都是介于系统总线和移位寄存器之间。串行通信是一个位一个位传输的，发送时把 TDR 内容转移到发送移位寄存器， 然后把移位寄存器数据每一位发送出去，接收时把接收到的每一位顺序保存在接收移位寄存器内然后才转移到 RDR。USART 支持 DMA 传输，可以实现高速数据传输。

1. 控制器

USART 有专门控制发送的发送器、控制接收的接收器，还有唤醒单元、中断控制等等。 使用 USART 之前需要向 USART\_CR1 寄存器的 UE 位置 1 使能USART，UE 位用来开启供给给串口的时钟。发送或者接收数据字长可选 8 位或 9 位，由 USART\_CR1 的 M 位控制。

1. 发送器

当 USART\_CR1 寄存器的发送使能位 TE 置 1 时，启动数据发送，发送移位寄存器的数据会在 TX 引脚输出， 低位在前，高位在后。如果是同步模式 SCLK也输出时钟信号。

一个字符帧发送需要三个部分：起始位+数据帧+停止位。起始位是一个位周期的低电平，位周期就是每一位占用的时间； 数据帧就是我们要发送的 8 位 或 9 位数据，数据是从最低位开始传输的；停止位是一定时间周期的高电平。停止位时间长短是可以通过USART控制寄存器2(USART\_CR2)的STOP[1:0]位控制，可选 0.5 个、1 个、1.5 个和 2 个停止位。 默认使用 1 个停止位。2 个停止位适用于正常 USART 模式、单线模式和调制解调器模式。0.5 个和 1.5 个停止位用于智能卡模式。

当发送使能位 TE 置 1 之后，发送器开始会先发送一个空闲帧(一个数据帧长度的高电平)，接下来就可以往 USART\_DR 寄存器写入要发送的数据。 在写入最后一个数据后，需要等待 USART 状态寄存器(USART\_SR)的 TC 位为 1，表示数据传输完成，如果 USART\_CR1 寄存器的 TCIE 位置 1，将产生中断。

在发送数据时，编程时有几个比较重要的标志位：



1. 接收器

如果将 USART\_CR1 寄存器的 RE 位置 1，使能 USART 接收，使得接收器在 RX 线开始搜索起始位。 在确定到起始位后就根据 RX 线电平状态把数据存放在接收移位寄存器内。接收完成后就把接收移位寄存器数据移到 RDR 内， 并 把 USART\_SR 寄存器的 RXNE 位置 1，同时如果 USART\_CR2 寄存器的 RXNEIE置 1 的话可以产生中断。

在接收数据时，编程时有几个比较重要的标志位：

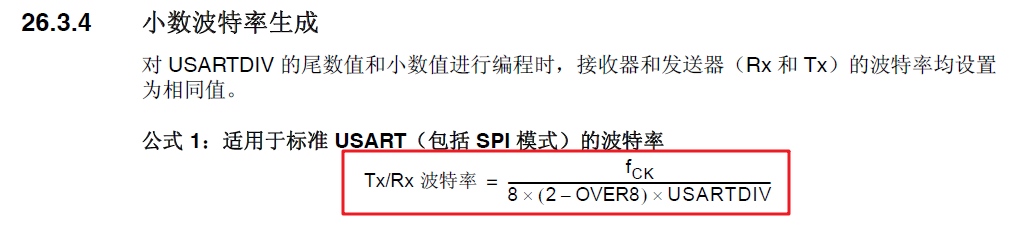


1. 中断控制

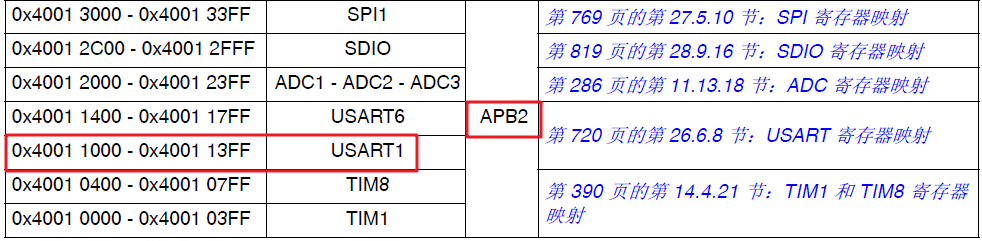
USART 有多个中断请求事件，具体事件见下表：

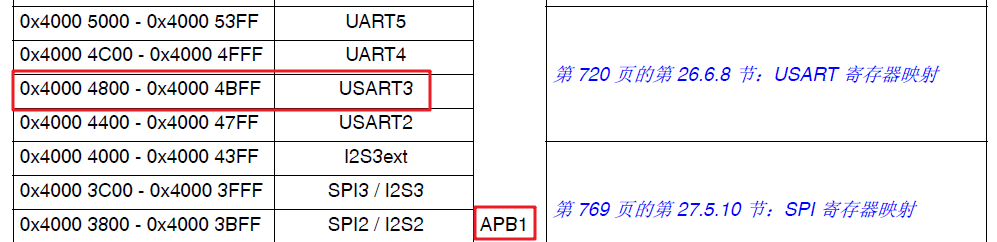


USART波特率的计算：









**2. 3 种基本的 I/O 数据传输方式：轮询、中断、DMA**

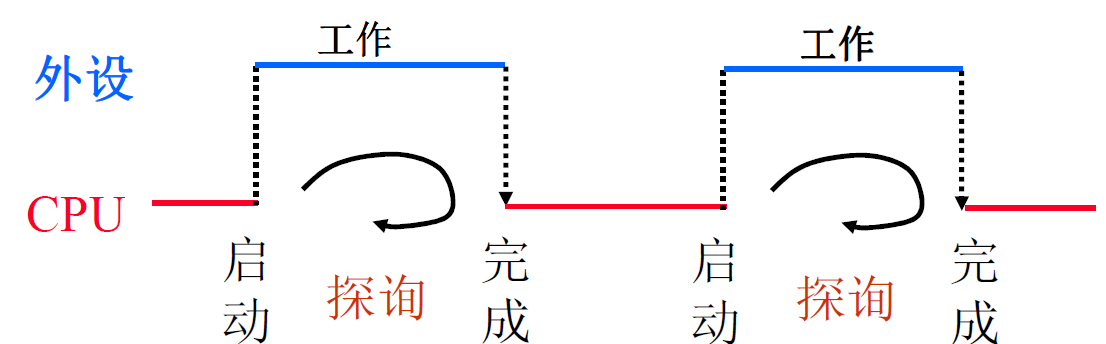
嵌入式系统中，I/O 设备与主机进行数据交换的三种基本方式是：

1. 程序直接控制方式

程序直接控制方式是最简单的 I/O 方式。又分为 2 种：

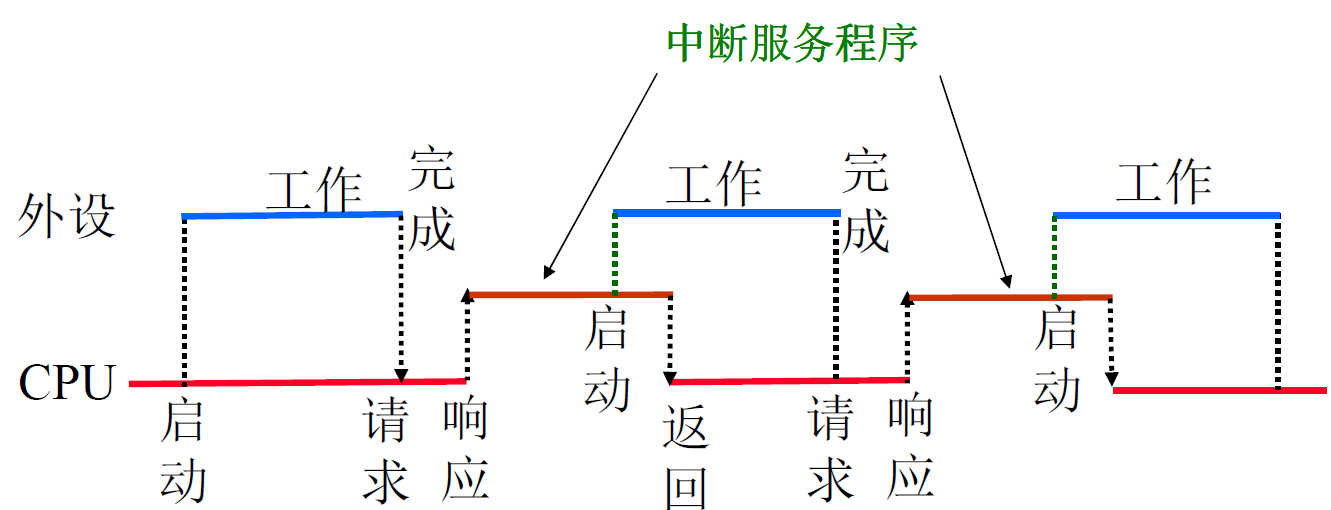
无条件传送：对简单外设定时（同步）进行数据传送。

条件传送：轮询（Polling）方式，即程序主动查询设备状态确定是否可以传送，所以也称为程序查询方式。工作方式是：I/O 设备（I/O 接口）将自己的状态放到一个状态寄存器中，主机程序定期查询状态寄存器中的特定状态，以决定下一步动作。所以，轮询时，CPU 与外设将处于串行工作方式，优点是简单易于控制，缺点是如果外设速度慢于 CPU，CPU 将在完全“空转”以等待外设完成，导致效率低下。



(2) 中断方式（I/O Interrupt）。

工作方式是：若一个 I/O 设备需要 CPU 干预，它就通过中断请求通知 CPU，CPU 中止当前程序的执行，转而调用中断服务程序（ISR）来执行。处理结束后，再返回到被中止的程序继续执行，因为程序被动调出，也称为中断驱动I/O方式。



(3) DMA 方式（Direct Memory Access）。

DMA方式常用于高速外设。

工作方式是：外设成批地直接和主存进行数据交换。需要专门的 DMA 控制器控制总线，完成数据传送。当外设准备好数据后，向 DMA 控制器发 DMA 请求信号，DMA 控制器再向 CPU 发总线请求，CPU 让出总线后，由 DMA 控制器控制总线进行传输，无需 CPU 干涉。

STM32 的 USART 支持轮询、中断、DMA3 种方式，本实验将使用此 3 种方式实现 USART3 收发，以支持目标板 shell 与主机的通信。

**3. 轮询模式和中断模式的选择**

当编写一个外设驱动时，其编程模式到底采用中断模式触发还是轮询模式触发往往是驱动开发人员首先要考虑的问题，并且这个问题在实时操作系统与分时操作系统中差异还非常大。因为轮询模式本身采用顺序执行的方式：查询到相应的事件然后进行对应的处理。所以轮询模式从实现上来说，相对简单清晰。

但是在实时系统中轮询模式可能会出现非常大问题，因为在实时操作系统中，当一个程序持续地执行时（轮询时），它所在的线程会一直运行，比它优先级低的线程都不会得到运行。而分时系统中，这点恰恰相反，几乎没有优先级之分，可以在一个时间片运行这个程序，然后在另外一段时间片上运行另外一段程序。所以通常情况下，实时系统中更多采用的是中断模式来驱动外设。当数据达到时，由中断唤醒相关的处理线程，再继续进行后续的动作。

故当一个实时系统想要提升数据吞吐量时，可以考虑的几种方式：

* 增加每次数据量发送的长度，每次尽量让外设尽量多地发送数据；
* 必要情况下更改中断模式为轮询模式。同时为了解决轮询方式一直抢占处理机，其他低优先级线程得不到运行的情况，可以把轮询线程的优先级适当降低。

**4. 中断服务程序与FIFO缓冲区**

由于外设与 CPU 速率不一致，二者并行运行时，收发数据时需要同步。轮询是最简单的同步方式，由 CPU 等待外设完成，将 CPU 与外设 2 个本质上并行的设备串行化了，简化了编程，但也降低了效率所以嵌入式系统常使用中断方式（DMA 也是用中断通知 CPU 处理）处理与外设的数据传输。此时，CPU 与外设是并行的，当外设完成时，触发中断服务程序 ISR，中断服务程序抢占 CPU 上的“正常”应用程序执行。所以，CPU 上应用程序与中断服务程序是并发执行的，在同一个 CPU 中的这 2 个程序也需要同步。

解决速率不匹配的同步方式主要是使用缓冲区。缓冲区是在两个设备 (应用) 间传输数据时，用来临时存放数据的内存区。使用缓冲的原因主要有：

1. 缓和 CPU 和 I/O 设备之间数据速率不匹配的矛盾

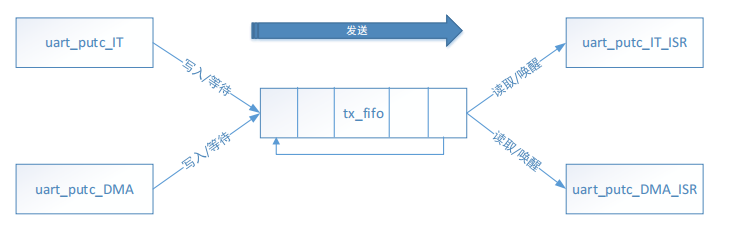
2. 提高 CPU 和 I/O 设备之间的并行性

3. 减少对 CPU 的中断频率

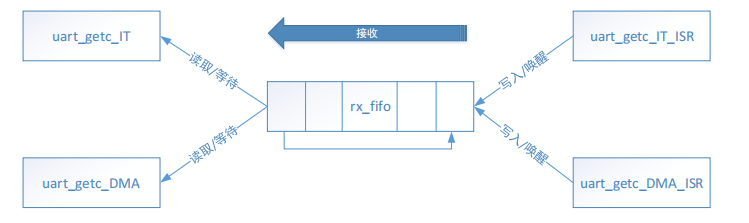
常用的缓冲机制有单缓冲、双缓冲、循环缓冲、缓冲池等方式，本实验使用的是用一个环形 FIFO 数组作为 USART 中断服务程序与 UART 驱动之间的缓冲区。本实验的 fifo 用的是 Linux 内核中的 kfifo。如果只有一个读线程、一个写线程，kfifo 不需要并发控制，因为二者没有共享的被修改的控制变量。这个条件在 shell 和 USART 中断服务程序之间是成立的。

中断与 DMA 方式都是需要与中断服务程序交互。中断与 DMA 的收发程序与相应收发的中断服务程序通过缓冲区通信，构成生产者-消费者模型。

发送时 kshell 写入 tx\_fifo，启动中断或者 DMA 从 tx\_ fifo 发送字符。

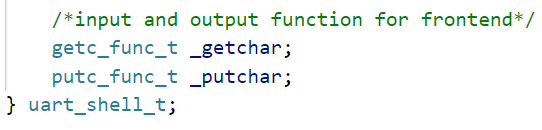


接收时启动中断或者 DMA 将字符接收到 rx\_fifo，然后唤醒 kshell 从 rx\_fifo读出字符。

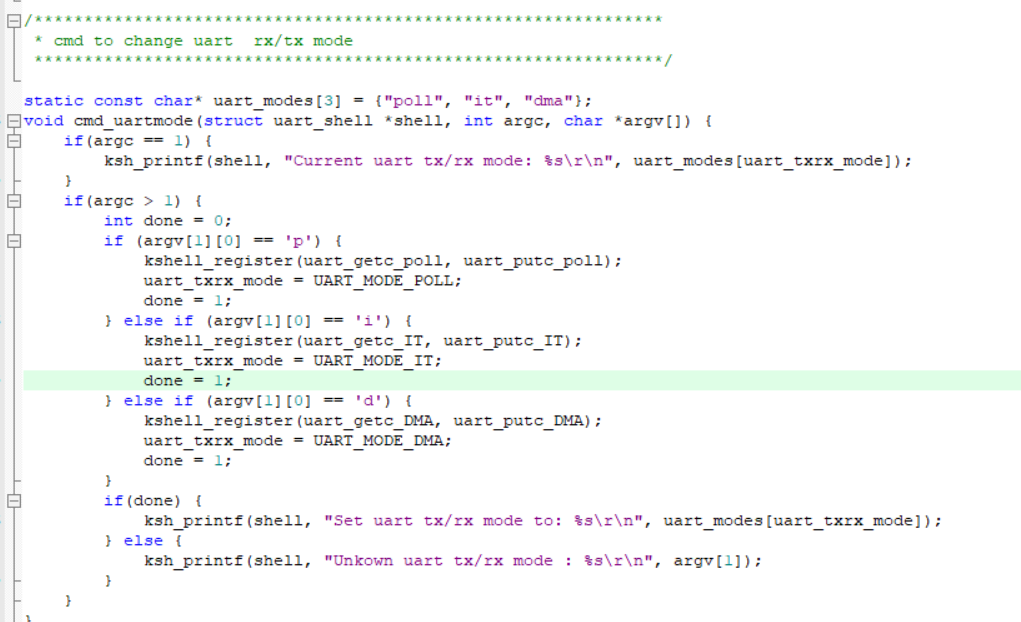


# 实验步骤

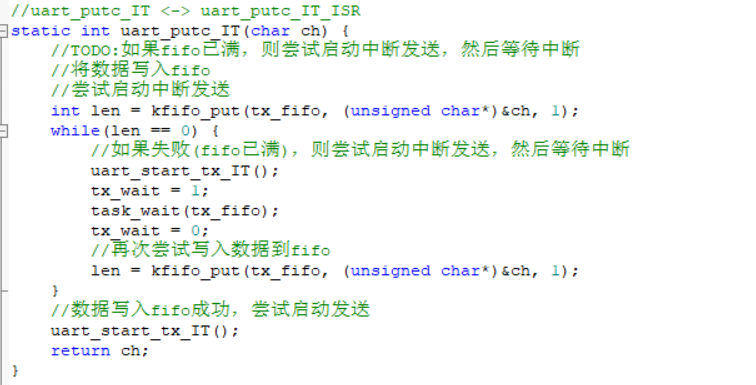
1. 进入程序入口project3\_main/project3\_main.c/project3\_main()，理解其结构和实现方式，可以看出，Kshell使用了两个函数指针，分别是\_getchar和\_putchar，通过kshell\_register设置：



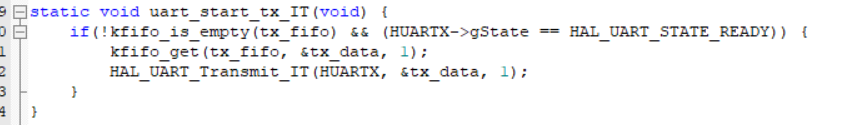
1. 在cmd\_uartmode中判断采用什么样的方式发送，如果读入的首字母为”i”，则采用中断的方式，调用uart\_getc\_IT和uart\_putc\_IT两个函数实现中断收发。



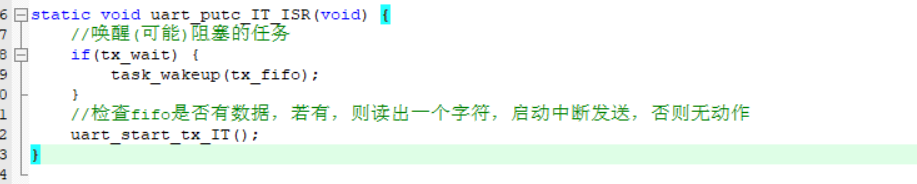
1. 下面进行中断发送的编写，基本思路为，如果fifo已满，则尝试启动中断发送，然后等待中断后再将数据写入fifo，写完后尝试启动中断发送。这里用变量len存储缓冲区剩余的大小，如果len的值等于0的话，说明缓冲区已满，此时调用uart\_start\_tx\_IT函数等待中断发生。注意此时访问缓冲区时需要互斥访问，用信号量tx\_wait来实现互斥。



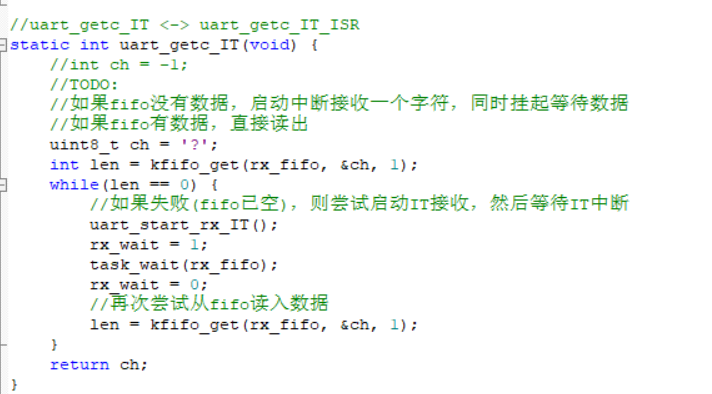
用uart\_start\_tx\_IT函数检查中断是否在发送，是则返回，否则读出一个字符，启动中断发送：



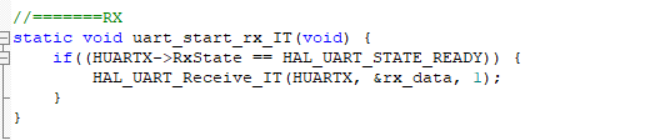
在uart\_putc\_IT\_ISR函数中，尝试唤醒处于阻塞态的任务，同时再次调用uart\_start\_tx\_IT()函数，检查fifo是否有数据，若有，则读出一个字符，启动中断发送，否则无动作。



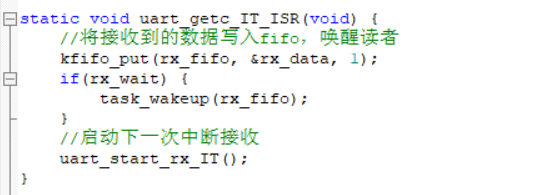
1. 下面进行中断接受的编写，基本思路为，如果fifo中没有数据，则启动中断接收一个字符，同时挂起等待数据，如果有数据则直接读出。这里用变量len存储缓冲区剩余的大小，如果len的值等于0的话，说明缓冲区已空，此时调用uart\_start\_rx\_IT函数等待中断。注意此时访问缓冲区时也需要互斥访问，用信号量tx\_wait来实现互斥。



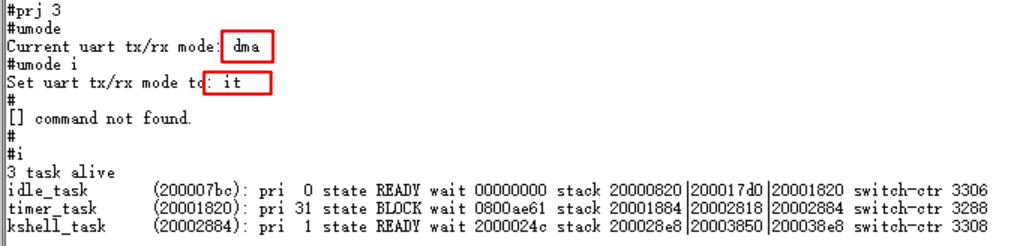
用uart\_start\_rx\_IT函数检查中断是否在接收数据，是则返回，否则读出一个字符，启动中断接收：



在uart\_getc\_IT\_ISR函数中，将接收到的数据写入fifo，唤醒读者，同时再次调用uart\_start\_rx\_IT()函数，启动下一次中断接收。



1. 运行程序，测试输出结果，在串口工具sscom中输入prj 3 命令，调用本实验入口。 在 sscom 中输入 umode i 命令，将 kshell 切换到中断收发模式。再输入其他任意内容，可以看到串口输出上是可以正常显示的，说明成功地用中断的方式实现了串口通信。



# 实验方案与实现

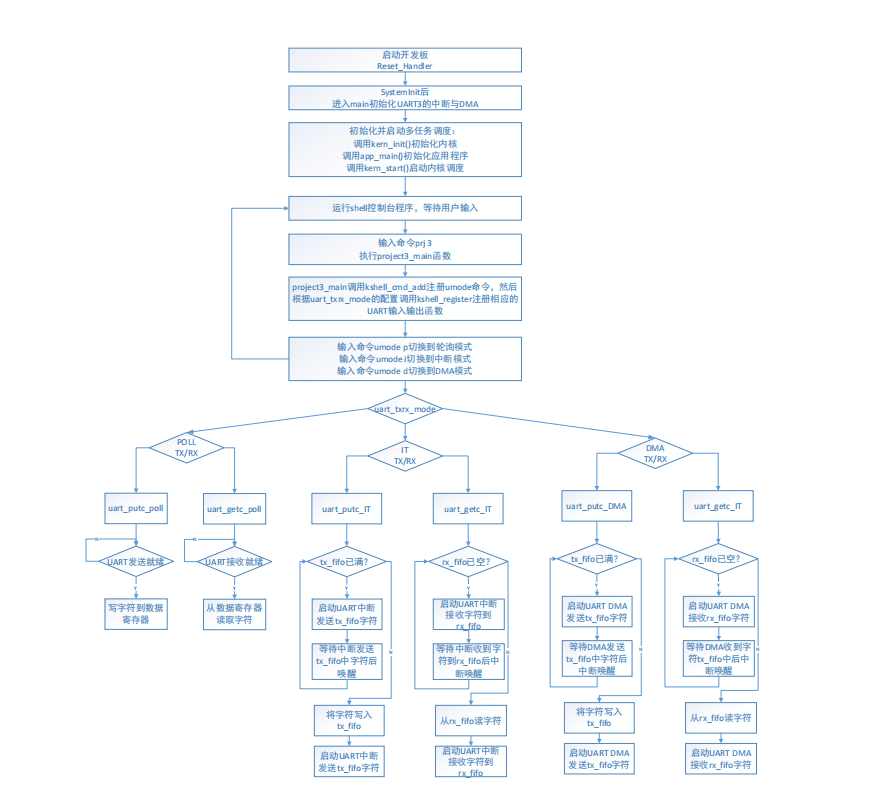
## 软件结构

在MDK环境中，点击启动调试并打开串口监视器查看串口输出。首先执行reset\_handler来启动开发板，并调用systemInit进入main初始化各外设，接下来初始化并启动多任务调度，在使用控制台程序中等待成用户输入命令。

在本实验中我们输入命令prj 3,来执行project3\_main函数，在project3\_main函数中调用kshell\_cmd\_add注册umode命令，然后根据uart\_txrx\_mode的配置调用kshell\_register注册相应的UART输入输出函数。然后根据不同的输入命令切换成不同的串口通信模式。本次实验主要关注采用中断的方式实现。

在中断模式下，分别调用uart\_putc\_IT和uart\_getc\_IT函数来实现对数据的发送和接收。在发送的过程中，首先检查缓冲区是否已满，如果已满，则启动uart中断并发送字符，等待发送后再唤醒。上述过程一直持续到fifo不再满为止，这时候再将字符写入缓冲区，并启动中断来发送该字符。而接收字符的过程与之类似，首先检查缓冲区是否为空，如果为空，则启动中断，一直接收字符，直到缓冲区不再为空为止。若不再为空，则从缓冲区中读取字符，并启动中断接收字符到缓冲区中。

使用DMA通信的过程与中断相类似，同样也使用了一个环形的fifo缓冲区。在发送数据时检查是否为满，直到不为满的时候才发送，而接收时则检查缓冲区是否为空，等待为空的时候才会读取字符。与中断不同的是，DMA则需要专门的DMA控制总线来完成数据传送，无需CPU的干涉。而轮询的方法实现则更为简单，它是属于一直处于询问状态，等待是否发送或接收就绪，再将对应的字符写或者从寄存器中取出。



## 源代码

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* Interrupt Mode

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DEFINE\_KFIFO(tx\_fifo, 7);//128 bytes

static uint8\_t tx\_data;

static int tx\_wait;

DEFINE\_KFIFO(rx\_fifo, 7);//128 bytes

static uint8\_t rx\_data;

static int rx\_wait;

//=======TX

//检查中断是否在发送，是则返回，否则读出一个字符，启动中断发送

static void uart\_start\_tx\_IT(void) {

    if(!kfifo\_is\_empty(tx\_fifo) && (HUARTX->gState == HAL\_UART\_STATE\_READY)) {

        kfifo\_get(tx\_fifo, &tx\_data, 1);

        HAL\_UART\_Transmit\_IT(HUARTX, &tx\_data, 1);

    }

}

//uart\_putc\_IT <-> uart\_putc\_IT\_ISR

static int uart\_putc\_IT(char ch) {

    //TODO:如果fifo已满，则尝试启动中断发送，然后等待中断

    //将数据写入fifo

    //尝试启动中断发送

        int len = kfifo\_put(tx\_fifo, (unsigned char\*)&ch, 1);

    while(len == 0) {

        //如果失败(fifo已满)，则尝试启动中断发送，然后等待中断

        uart\_start\_tx\_IT();

        tx\_wait = 1;

        task\_wait(tx\_fifo);

        tx\_wait = 0;

        //再次尝试写入数据到fifo

        len = kfifo\_put(tx\_fifo, (unsigned char\*)&ch, 1);

    }

    //数据写入fifo成功，尝试启动发送

    uart\_start\_tx\_IT();

        return ch;

}

static void uart\_putc\_IT\_ISR(void) {

    //唤醒(可能)阻塞的任务

    if(tx\_wait) {

        task\_wakeup(tx\_fifo);

    }

    //检查fifo是否有数据，若有，则读出一个字符，启动中断发送，否则无动作

    uart\_start\_tx\_IT();

}

//=======RX

static void uart\_start\_rx\_IT(void) {

    if((HUARTX->RxState == HAL\_UART\_STATE\_READY)) {

        HAL\_UART\_Receive\_IT(HUARTX, &rx\_data, 1);

    }

}

//uart\_getc\_IT <-> uart\_getc\_IT\_ISR

static int uart\_getc\_IT(void) {

        //int ch = -1;

    //TODO:

    //如果fifo没有数据，启动中断接收一个字符，同时挂起等待数据

    //如果fifo有数据，直接读出

        uint8\_t ch = '?';

    int len = kfifo\_get(rx\_fifo, &ch, 1);

    while(len == 0) {

        //如果失败(fifo已空)，则尝试启动中断接收，然后等待IT中断

        uart\_start\_rx\_IT();

        rx\_wait = 1;

        task\_wait(rx\_fifo);

        rx\_wait = 0;

        //再次尝试从fifo读入数据

        len = kfifo\_get(rx\_fifo, &ch, 1);

    }

    return ch;

}

static void uart\_getc\_IT\_ISR(void) {

    //将接收到的数据写入fifo，唤醒读者

    kfifo\_put(rx\_fifo, &rx\_data, 1);

    if(rx\_wait) {

        task\_wakeup(rx\_fifo);

    }

    //启动下一次中断接收

    uart\_start\_rx\_IT();

}

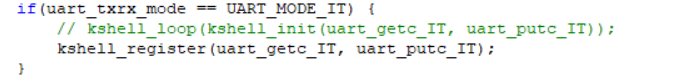
# 实验结果与分析

1. 请描述从主机端输入字符到开发板端收到字符并在 shell 上执行命令的过程。

当用户从主机端输入字符时，输入的字符会存在缓冲区内，然后通过不同的方式去处理这个缓冲区。如在本实验中采用的是中断的方式处理，则在cpu运行时启动中断，从缓冲区中读取对应的数据，并将其发送给cpu，由cpu收到数据后进行处理，分析指令内容并执行。

1. 本实验通过中断方式接收字符，但是发送还是使用轮询。如果要将发送也改为中断方式，该如何处理？

上述代码中，已经实现了用中断的方式来发送字符。运行程序时，输入命令prj 3，再输入命令umode i切换到中断模式。根据代码可以知道，切换到中断模式的时候，收发分别是对应uart\_getc\_IT和uart\_putc\_IT这两个函数，我们通过改写uart\_putc\_IT函数，实现了用中断的方式来发送数据。



3.本实验的 DMA 方式使用缓冲区接收字符，并在缓冲区满时通知程序将缓冲区字符复制处理。如果在复制的同时，又有字符需要接收，是否会丢失字符？如果是，该如何处理？

我认为在复制的同时，如果还有字符需要接收，则会出现丢失字符的情况。原因接收了新输入的字符后，再将缓冲区复制的内容复制回去后，会将新写入的字符给覆盖掉，那么新写入的字符信息就会出现丢失。

我认为有两种处理方法。一种是采用更大的缓冲区，之所以会出现缓冲区满，且还有别的字符需要读入，原因是缓冲区的大小设置不合理，如果采用更大的缓冲区，则可以解决这方面的问题。另一种解决方案是，如果缓冲区满，则拒绝接受其他的字符，要读入的字符一直挂起，等待缓冲区空了之后有空余的位置，再将该新字符给写入，就不会出现数据丢失的情况。

# 实验总结

本次实验通过使用轮询、中断 DMA 3种方式实现了USART3与shell驱动接口之间的通信。本次实验通过模拟这三种通信的方式，帮助我深入了解了这三种通信实现的具体细节，如轮询是持续询问。而中断和DMA则是采用了缓冲区帮助存储并处理，实现 shell 收发的功能并能够在在主机的串口软件中通过串口操作 shell。