3.3 串口通讯

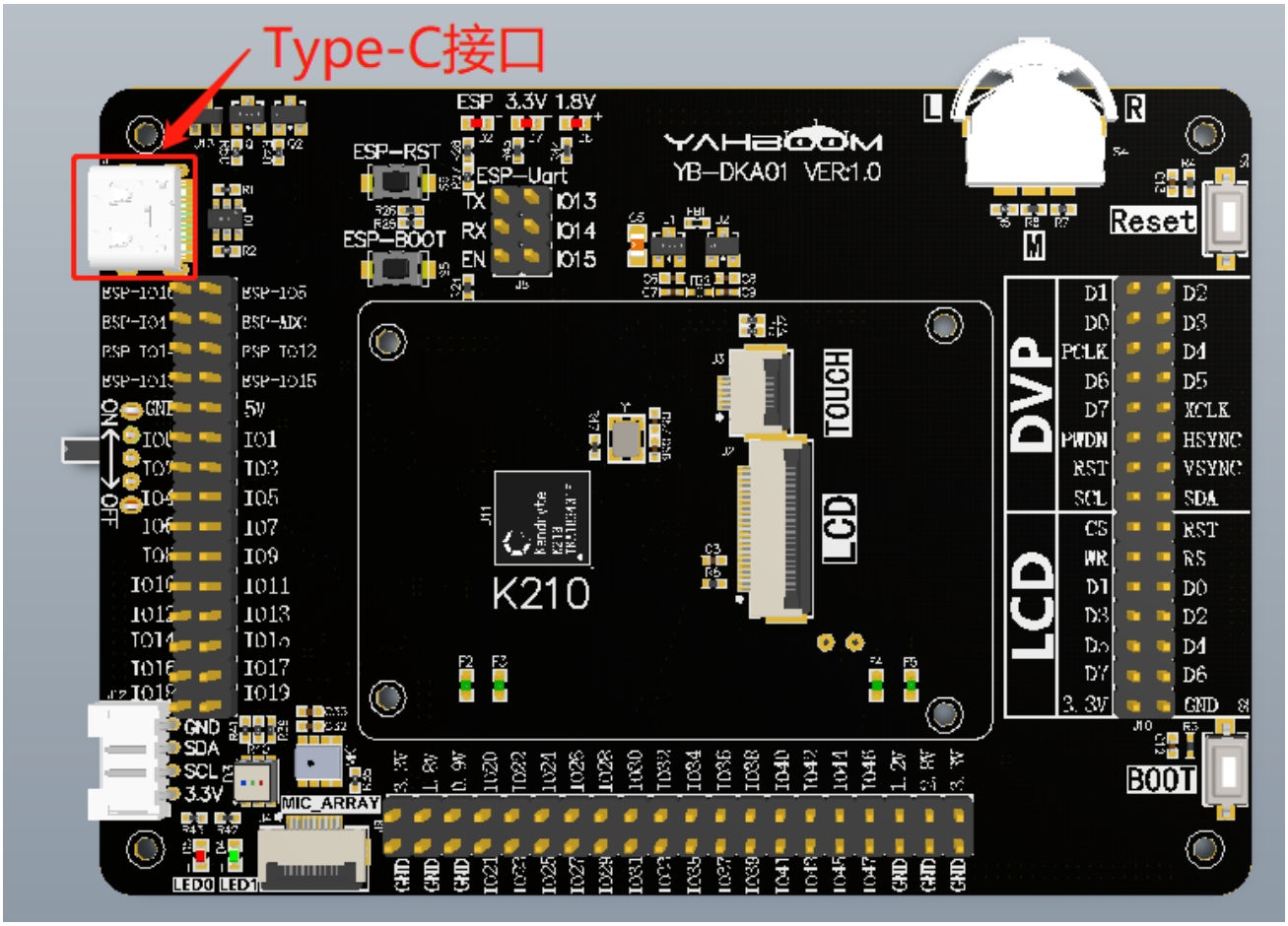
一、实验目的

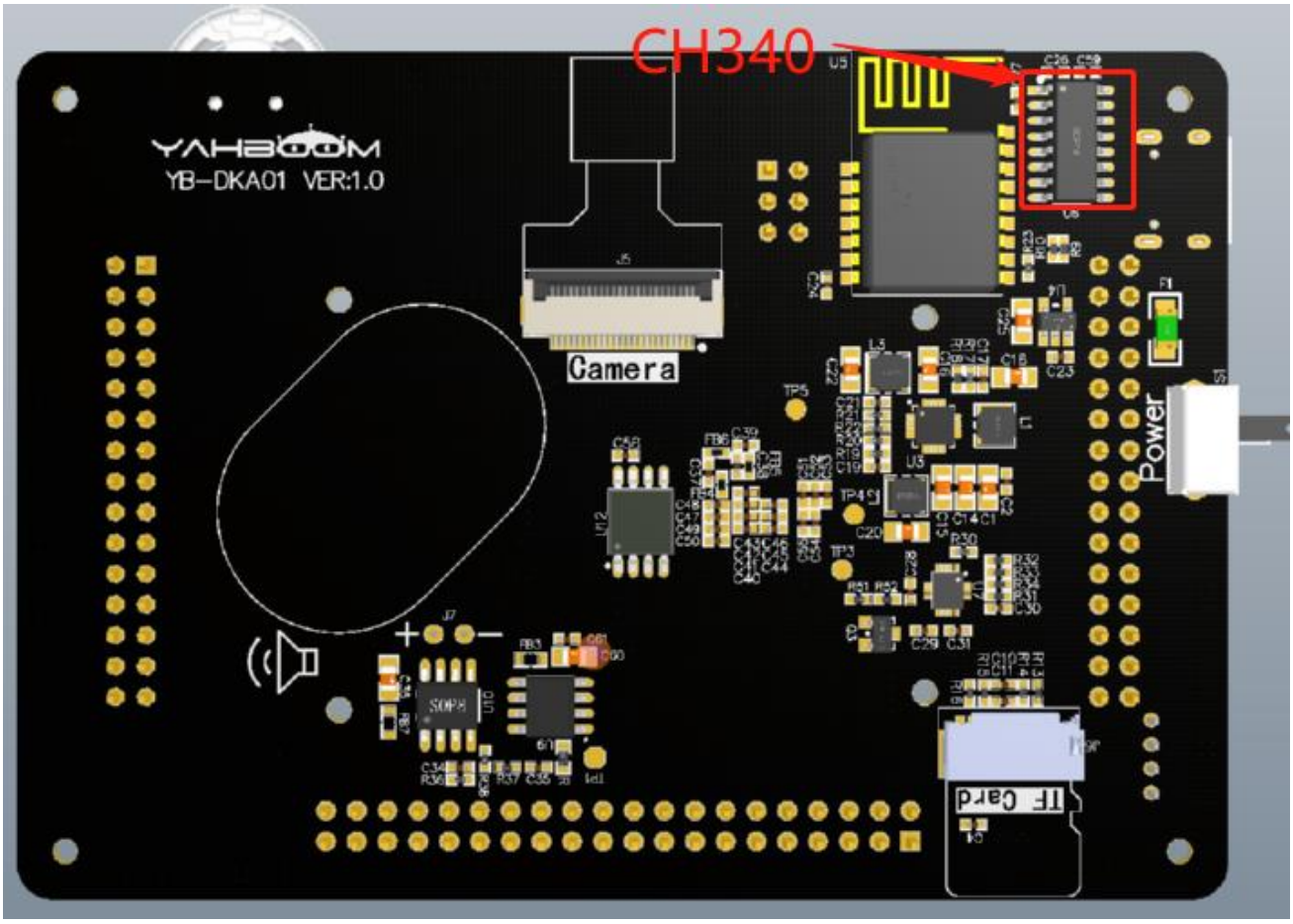
本节课主要学习 K210 的串口通讯。

二、实验准备

1.实验元件

Type-C 接口和 CH340 串口芯片





2.元件特性

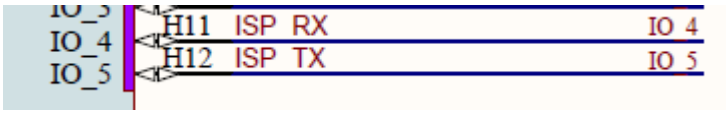
Tpye-C 接口连接到串口芯片，具有供电和串口传输数据的功能，下载程序也是通过串口传输数据的功能把固件传输到 K210 芯片上的。

Type-C 接口是目前主流的数据传输连接接口，市面上很多智能手机也是使用 Type-C 接口给手机充电和传输数据。Type-C 接口是可以正反插的，不用担心反向拿反而插不进的问题。

CH340 串口芯片是一个 USB 总线的转接芯片，可以实现 USB 转串口。具有全速 USB 设备接口，硬件全双工串口，内置收发缓冲区，支持通讯波特率为 50bps~2Mbps，兼容 USBV2.0 等特点。

3.硬件连接

Tpye-C 接口连接 K210 的 IO4 和 IO5 接口，其中 IO4 为 K210 芯片的接收引脚，IO5 为 K210 芯片的发送引脚。



4.SDK 中对应 API 功能

对应的头文件 uart.h

通用 UART 为 UART1、UART2 和 UART3，支持异步通信（RS232 和 RS485 和 IRDA，通信速率可达到 5Mbps。UART 支持 CTS 和 RTS 信号的硬件管理以及软件流控(XON 和 XOFF)。3 个接口均可被 DMA 访问或者 CPU 直接访问。每次传输数据为 8 字节，支持异步时钟，可单独配置数据时钟，实现全双工模式，保证两个时钟域中数据同步。

uart 默认为 RS232 模式，也可以配置为软件可编程式 RS485 模式。

用 THRE 中断模式来提升串口性能。当 THRE 模式和 FIFO 模式被选择之后，如果 FIFO 中少于阈值便触发 THRE 中断。

为用户提供以下接口：

- uart_init：初始化 uart
- uart_config（0.6.0 后不再支持，请使用 uart_configure）
- uart_configure：配置串口的波特率等
- uart_send_data：通过串口发送数据
- uart_send_data_dma：通过 dma 通道发送数据
- uart_send_data_dma_irq：UART 通过 DMA 发送数据，并注册 DMA 接收完成中断函数，仅单次中断
- uart_receive_data：读取串口数据
- uart_receive_data_dma：串口通过 dma 接收数据
- uart_receive_data_dma_irq：UART 通过 DMA 接收数据，并注册 DMA 接收完成中断函数，仅单次中断
- uart_irq_register：注册串口中断函数

- uart_irq_deregister：注销串口中断
- uart_set_work_mode：设置 uart 工作模式。有四种模式：普通 uart、红外、RS485 全双工、RS485 半双工。
- uart_set_rede_polarity：设置 RS485 re de 管脚有效时的极性。
- uart_set_rede_enable：使能 re de 管脚，主要用在 rs485 全双工模式，re 和 de 必须手动控制。单双工模式不用调用该函数，单双工时硬件会自动设置 re de。
- uart_set_tat：配置 re de 互相的时间间隔，这个与外部的 485 模块有关。
- uart_set_det：设置 de 有效无效转换时数据的时间延时。
- uart_debug_init：配置调试串口。系统默认使用 UART3 做为调试串口，调用该函数用户可以自定义使用哪个 uart 做为调试串口，如果使用 UART1 或 UART2 需要使用 fpioa 设置管脚。因此调试脚不局限于 fpioa4、5
- uart_handle_data_dma：UART 通过 DMA 传输数据

高速通用异步收发传输器(UARTHS)对应的头文件 uarths.h

高速 UART 为 UARTHS（UART0）,通讯速率可达到 5Mbps，8 字节发送和接收 FIFO，可编程式 THRE 中断，不支持硬件流控制或者其他调制解调器控制信号，或同步串行数据转换器。系统的 printf 调试函数默认就是调用 UARTHS 串口来实现发送数据的。

为用户提供以下接口

- uarths_init：初始化 UARTHS，系统默认波特率为 115200 8bit 1 位停止位无检验位。因为 uarths 时钟源为 PLL0，在设置 PLL0 后需要重新调用该函数设置波特率，否则会打印乱码。
- uarths_config：设置 UARTHS 的参数。默认 8bit 数据，无校验位。
- uarths_receive_data：通过 UARTHS 读取数据。
- uarths_send_data：通过 UART 发送数据。
- uarths_set_irq：设置 UARTHS 中断回调函数。
- uarths_get_interrupt_mode：获取 UARTHS 的中断类型。接收、发送或接收发送同时中断。
- uarths_set_interrupt_cnt：设置 UARTHS 中断时的 FIFO 深度。当中断类型为 UARTHS_SEND_RECEIVE，发送接收 FIFO 中断深度均为 cnt;

三、实验原理

串口通讯是指外设和计算机间，通过数据信号线、地线等，按位（bit）进行传输数据（发送和接收）的一种通讯方式，一个字符一个字符地传输，每个字符一位一位地传输，并且传输一个字符时，总是以“起始位”开始，以“停止位”

结束，字符之间没有固定的时间间隔要求，但是数据是低位在前，高位在后，然后接上奇偶检验位。

序号	名称	备注
1	起始位	表示传输字符的开始，逻辑"0"的信号。
2	数据位	数据位的个数可以是 5、6、7、8 等，构成一个字符 从最低位开始传送，靠时钟定位
3	奇偶校验位	数据位加上这一位后，使得"1"的位数应为偶数（偶校验）或奇数（奇校验），以此来校验数据传送的正确性
4	停止位	一个字符数据的结束标志，可以是 1 位、1.5 位、2 位的高电平

串口支持全双工通讯，也就是使用一根线发送数据的同时，用另一根线接收数据。串口通讯最重要的参数是波特率、数据位、停止位和奇偶校验，对于两个通讯的端口，这些参数是必须设置为相同的。

四、实验过程

1.首先根据上面的硬件连接引脚图，K210 的硬件引脚和软件功能使用的是 FPIOA 映射关系。

这里要注意的是程序里操作的都是软件引脚，所以需要先把硬件引脚映射成软件 GPIO 功能，操作的时候直接操作软件 GPIO 即可。

```
/*****************************HARDWARE-PIN*******************************
// 硬件IO口，与原理图对应
#define PIN_UART_USB_RX         (4)
#define PIN_UART_USB_TX         (5)

/*****************************SOFTWARE-GPIO******************************
// 软件GPIO口，与程序对应
#define UART_USB_NUM            UART_DEVICE_3

/*****************************FUNC-GPIO**********************************
// GPIO口的功能，绑定到硬件IO口
#define FUNC_UART_USB_RX        (FUNC_UART1_RX + UART_USB_NUM * 2)
#define FUNC_UART_USB_TX        (FUNC_UART1_TX + UART_USB_NUM * 2)
```

```
void hardware_init(void)
{
    // fpioa映射
    fpioa_set_function(PIN_UART_USB_RX, FUNC_UART_USB_RX);
    fpioa_set_function(PIN_UART_USB_TX, FUNC_UART_USB_TX);
}
```

2. 然后是初始化串口，这里我们选择的是串口 3，设置波特率为 115200，串口数据宽度为 8 位，停止位为 1 位，不使用奇偶校验。

```
// 初始化串口3，设置波特率为115200
uart_init(UART_USB_NUM);
uart_configure(UART_USB_NUM, 115200, UART_BITWIDTH_8BIT, UART_STOP_1, UART_PARITY_NONE);
```

3. 开机的时候发送“hello yahboom!”，提示已经开机完成。

```
/* 开机发送hello yahboom! */
char *hello = {"hello yahboom!\n"};
uart_send_data(UART_USB_NUM, hello, strlen(hello));
```

4. 然后循环等待串口数据，如果接收到串口数据，再通过串口把接收到的数据发送出去。

```c
char recv = 0;

while (1)
{
    /* 等待串口信息，并通过串口发送出去 */
    while(uart_receive_data(UART_USB_NUM, &recv, 1))
    {
        uart_send_data(UART_USB_NUM, &recv, 1);
    }
}
```

5.编译调试，烧录运行

把本课程资料中的 gpiohs_rgb 复制到 SDK 中的 src 目录下，然后进入 build 目录，运行以下命令编译。

cmake .. -DPROJ=uart -G "MinGW Makefiles"

make

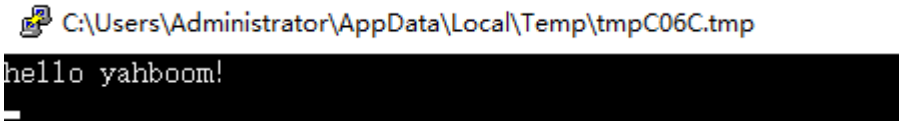
```
[ 97%] Building C object CMakeFiles/uart.dir/src/uart/main.c.obj
[100%] Linking C executable uart
Generating .bin file ...
[100%] Built target uart
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

编译完成后，在 build 文件夹下会生成 uart.bin 文件。

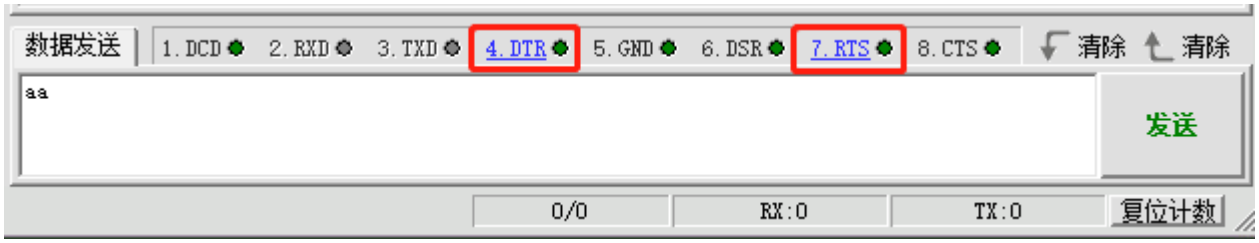
使用 type-C 数据线连接电脑与 K210 开发板，打开 kflash，选择对应的设备，再将程序固件烧录到 K210 开发板上。

五、实验现象

烧录完成固件后，系统会弹出一个终端界面，如果没有弹出终端界面的可以打开串口助手显示调试内容。



打开电脑的串口助手，选择对应的 K210 开发板对应的串口号，波特率设置为 115200,然后点击打开串口助手。注意还需要设置一下串口助手的 DTR 和 RTS。在串口助手底部此时的 4.DTR 和 7.RTS 默认是红色的，点击 4.DTR 和 7.RTS，都设置为绿色，然后按一下 K210 开发板的复位键。



从串口助手，可以接收到 hello yahboom!的欢迎语，然后在底部输入要发送的字符，然后点击发送，K210 会把接收到的数据发送回来在串口助手上显示。

六、实验总结

1.uart 总共有三个，分别是 UART1,UART2 和 UART3。

2.uart 默认使用 RS232 模式，可以另外配置成可编程式 RS485 模式。

3.uart 的引脚如果映射到其他硬件引脚上，需要连接其他串口芯片如 CH340 上才可以显示数据。

附：API

对应的头文件 uart.h

uart_init

描述

初始化 uart。

函数原型

```
void uart_init(uart_device_number_t channel)
```

参数

参数名称	描述	输入输出
channel	UART 号	输入

返回值

无。

uart_configure

描述

设置 UART 相关参数。该函数已废弃，替代函数为 uart_configure。

uart_configure

描述

设置 UART 相关参数。

函数原型

```
void uart_configure(uart_device_number_t channel, uint32_t baud_rate,
uart_bitwidth_t data_width, uart_stopbit_t stopbit, uart_parity_t parity)
```

参数

参数名称	描述	输入输出
channel	UART 编号	输入
baud_rate	波特率	输入
data_width	数据位 (5-8)	输入
stopbit	停止位	输入
parity	校验位	输入

返回值

无。

uart_send_data

描述

通过 UART 发送数据。

函数原型

```
int uart_send_data(uart_device_number_t channel, const char *buffer, size_t
buf_len)
```

参数

参数名称	描述	输入输出
channel	UART 编号	输入
buffer	待发送数据	输入
buf_len	待发送数据的长度	输入

返回值

已发送数据的长度。

uart_send_data_dma

描述

UART 通过 DMA 发送数据。数据全部发送完毕后返回。

函数原型

```
void uart_send_data_dma(uart_device_number_t uart_channel,
dmac_channel_number_t dmac_channel, const uint8_t *buffer, size_t buf_len)
```

参数

参数名称	描述	输入输出
uart_channel	UART 编号	输入
dmac_channel	DMA 通道	输入
buffer	待发送数据	输入
buf_len	待发送数据的长度	输入

返回值

无。

uart_send_data_dma_irq

描述

UART 通过 DMA 发送数据，并设置 DMA 发送完成中断函数，仅单次中断。

函数原型

```
void uart_send_data_dma_irq(uart_device_number_t uart_channel,
dmac_channel_number_t dmac_channel,
                            const uint8_t *buffer, size_t buf_len,
plic_irq_callback_t uart_callback,
                            void *ctx, uint32_t priority)
```

参数

参数名称	描述	输入输出

参数名称	描述	输入输出
uart_channel	UART 编号	输入
dmac_channel	DMA 通道	输入
buffer	待发送数据	输入
buf_len	待发送数据的长度	输入
uart_callback	DMA 中断回调	输入
ctx	中断函数参数	输入
priority	中断优先级	输入

返回值

无。

uart_receive_data

描述

通过 UART 读取数据。

函数原型

```
int uart_receive_data(uart_device_number_t channel, char *buffer, size_t buf_len);
```

参数

参数名称	描述	输入输出
channel	UART 编号	输入
buffer	接收数据	输出
buf_len	接收数据的长度	输入

返回值

已接收到的数据长度。

uart_receive_data_dma

描述

UART 通过 DMA 接收数据。

函数原型

```
void uart_receive_data_dma(uart_device_number_t uart_channel,
dmac_channel_number_t dmac_channel, uint8_t *buffer, size_t buf_len)
```

参数

参数名称	描述	输入输出
uart_channel	UART 编号	输入
dmac_channel	DMA 通道	输入
buffer	接收数据	输出
buf_len	接收数据的长度	输入

返回值

无。

uart_receive_data_dma_irq

描述

UART 通过 DMA 接收数据，并注册 DMA 接收完成中断函数，仅单次中断。

函数原型

```
void uart_receive_data_dma_irq(uart_device_number_t uart_channel,
dmac_channel_number_t dmac_channel,
                               uint8_t *buffer, size_t buf_len,
plic_irq_callback_t uart_callback,
                               void *ctx, uint32_t priority)
```

参数

参数名称	描述	输入输出
uart_channel	UART 编号	输入
dmac_channel	DMA 通道	输入
buffer	接收数据	输出
buf_len	接收数据的长度	输入
uart_callback	DMA 中断回调	输入
ctx	中断函数参数	输入
priority	中断优先级	输入

返回值

无。

uart_irq_register

描述

注册 UART 中断函数。

函数原型

```
void uart_irq_register(uart_device_number_t channel, uart_interrupt_mode_t interrupt_mode, plic_irq_callback_t uart_callback, void *ctx, uint32_t priority)
```

参数

参数名称	描述	输入输出
channel	UART 编号	输入
interrupt_mode	中断类型	输入
uart_callback	中断回调	输入
ctx	中断函数参数	输入
priority	中断优先级	输入

返回值

无。

uart_irq_deregister

描述

注销 UART 中断函数。

函数原型

```
void uart_irq_deregister(uart_device_number_t channel, uart_interrupt_mode_t interrupt_mode)
```

参数

参数名称	描述	输入输出
channel	UART 编号	输入
interrupt_mode	中断类型	输入

返回值

无。

uart_set_work_mode

描述

设置 uart 工作模式。有四种模式：普通 uart、红外、RS485 全双工、RS485 半双工。

函数原型

```
void uart_set_work_mode(uart_device_number_t uart_channel, uart_work_mode_t work_mode)
```

参数

参数名称	描述	输入输出
channel	UART 编号	输入
work_mode	工作模式，详细见 uart_work_mode_t 结构体说明	输入

返回值

无。

uart_set_rede_polarity

描述

设置 RS485 re de 管脚有效时的极性。

函数原型

```
void uart_set_rede_polarity(uart_device_number_t uart_channel, uart_rs485_rede_t rede, uart_polarity_t polarity)
```

参数

参数名称	描述	输入输出
uart_channel	UART 编号	输入
rede	re 或 de 管脚	输入
polarity	有效时极性	输入

返回值

无。

uart_set_rede_enable

描述

使能 re de 管脚，主要用在 rs485 全双工模式，re 和 de 必须手动控制。单双工模式不用调用该函数，单双工时硬件会自动设置 re de。

函数原型

```
void uart_set_rede_enable(uart_device_number_t uart_channel,
uart_rs485_rede_t rede, bool enable)
```

参数

参数名称	描述	输入输出
uart_channel	UART 编号	输入
rede	re 或 de 管脚	输入
enable	是否有效	输入

返回值

无。

uart_set_tat

描述

配置 re de 互相的时间间隔，这个与外部的 485 模块有关。

函数原型

```
void uart_set_tat(uart_device_number_t uart_channel, uart_tat_mode_t tat_mode,
size_t time)
```

参数

参数名称	描述	输入输出
uart_channel	UART 编号	输入
tat_mode	转换模式 re 到 de 或 de 到 re	输入
time	时间（纳秒）	输入

返回值

无。

uart_set_det

描述

设置 de 有效无效转换时数据的时间延时。

函数原型

```
void uart_set_det(uart_device_number_t uart_channel, uart_det_mode_t det_mode,
size_t time)
```

参数

参数名称	描述	输入输出
uart_channel	UART 编号	输入
det_mode	转换模式，de 无效转有效或有效转无效	输入
time	时间（纳秒）	输入

返回值

无。

uart_debug_init

描述

配置调试串口。系统默认使用 UART3 做为调试串口，调用该函数用户可以自定义使用哪个 uart 做为调试串口，如果使用 UART1 或 UART2 需要使用 fpioa 设置管脚。因此调试脚不局限于 fpioa4、5。

函数原型

```
void uart_debug_init(uart_device_number_t uart_channel)
```

参数

参数名称	描述	输入输出
uart_channel	UART 编号，-1 则默认使用上次设置的 UART	输入

返回值

无。

uart_handle_data_dma

描述

UART 通过 DMA 传输数据。

函数原型

```
void uart_handle_data_dma(uart_device_number_t uart_channel ,uart_data_t data,
plic_interrupt_t *cb)
```

参数

参数名称	描述	输入输出
uart_channel	UART 编号	输入
data	UART 数据相关的参数，详见 i2c_data_t 说明	输入
cb	dma 中断回调函数，如果设置为 NULL 则为阻塞模式，直至传输完毕后退出函数	输入

返回值

无

数据类型

相关数据类型、数据结构定义如下：

- uart_device_number_t：UART 编号。
- uart_bitwidth_t：UART 数据位宽。
- uart_stopbits_t：UART 停止位。
- uart_parity_t：UART 校验位。
- uart_interrupt_mode_t：UART 中断类型，接收或发送。
- uart_send_trigger_t：发送中断或 DMA 触发 FIFO 深度。
- uart_receive_trigger_t：接收中断或 DMA 触发 FIFO 深度。
- uart_data_t：使用 dma 传输时数据相关的参数。
- uart_interrupt_mode_t：传输模式，发送或接收。
- uart_work_mode_t：UART 工作模式。
- uart_rs485_rede_t：选择 re de 管脚。
- uart_polarity_t：de re 极性。
- uart_tat_mode_t：de re 转换选择。
- uart_det_mode_t：de 有效无效选择。

uart_device_number_t

描述

UART 编号。

定义

```
typedef enum _uart_device_number
{
    UART_DEVICE_1,
    UART_DEVICE_2,
    UART_DEVICE_3,
    UART_DEVICE_MAX,
} uart_device_number_t;
```

成员

成员名称	描述
UART_DEVICE_1	UART 1
UART_DEVICE_2	UART 2
UART_DEVICE_3	UART 3

uart_bitwidth_t

描述

UART 数据位宽。

定义

```
typedef enum _uart_bitwidth
{
    UART_BITWIDTH_5BIT = 0,
    UART_BITWIDTH_6BIT,
    UART_BITWIDTH_7BIT,
    UART_BITWIDTH_8BIT,
} uart_bitwidth_t;
```

成员

成员名称	描述
UART_BITWIDTH_5BIT	5 比特
UART_BITWIDTH_6BIT	6 比特
UART_BITWIDTH_7BIT	7 比特
UART_BITWIDTH_8BIT	8 比特

uart_stopbits_t

描述

UART 停止位。

定义

```
typedef enum _uart_stopbits
{
    UART_STOP_1,
    UART_STOP_1_5,
    UART_STOP_2
} uart_stopbits_t;
```

成员

成员名称	描述
UART_STOP_1	1 个停止位
UART_STOP_1_5	1.5 个停止位
UART_STOP_2	2 个停止位

uart_parity_t

描述

UART 校验位。

定义

```
typedef enum _uart_parity
{
    UART_PARITY_NONE,
    UART_PARITY_ODD,
    UART_PARITY_EVEN
} uart_parity_t;
```

成员

成员名称	描述
UART_PARITY_NONE	无校验位
UART_PARITY_ODD	奇校验
UART_PARITY_EVEN	偶校验

uart_interrupt_mode_t

描述

UART 中断类型，接收或发送。

定义

```
typedef enum _uart_interrupt_mode
{
    UART_SEND = 1,
    UART_RECEIVE = 2,
} uart_interrupt_mode_t;
```

成员

成员名称	描述
UART_SEND	UART 发送
UART_RECEIVE	UART 接收

uart_send_trigger_t

描述

发送中断或 DMA 触发 FIFO 深度。当 FIFO 中的数据小于等于该值时触发中断或 DMA 传输。FIFO 的深度为 16 字节。

定义

```
typedef enum _uart_send_trigger
{
    UART_SEND_FIFO_0,
    UART_SEND_FIFO_2,
    UART_SEND_FIFO_4,
    UART_SEND_FIFO_8,
} uart_send_trigger_t;
```

成员

成员名称	描述
UART_SEND_FIFO_0	FIFO 为空
UART_SEND_FIFO_2	FIFO 剩余 2 字节
UART_SEND_FIFO_4	FIFO 剩余 4 字节
UART_SEND_FIFO_8	FIFO 剩余 8 字节

uart_receive_trigger_t

描述

接收中断或 DMA 触发 FIFO 深度。当 FIFO 中的数据大于等于该值时触发中断或 DMA 传输。FIFO 的深度为 16 字节。

定义

```
typedef enum _uart_receive_trigger
{
    UART_RECEIVE_FIFO_1,
    UART_RECEIVE_FIFO_4,
    UART_RECEIVE_FIFO_8,
    UART_RECEIVE_FIFO_14,
} uart_receive_trigger_t;
```

成员

成员名称	描述
UART_RECEIVE_FIFO_1	FIFO 剩余 1 字节
UART_RECEIVE_FIFO_4	FIFO 剩余 2 字节
UART_RECEIVE_FIFO_8	FIFO 剩余 4 字节
UART_RECEIVE_FIFO_14	FIFO 剩余 8 字节

uart_data_t

描述

使用 dma 传输时数据相关的参数。

定义

```
typedef struct _uart_data_t
{
    dmac_channel_number_t tx_channel;
    dmac_channel_number_t rx_channel;
    uint32_t *tx_buf;
    size_t tx_len;
    uint32_t *rx_buf;
    size_t rx_len;
    uart_interrupt_mode_t transfer_mode;
} uart_data_t;
```

成员

成员名称	描述
tx_channel	发送时使用的 DMA 通道号
rx_channel	接收时使用的 DMA 通道号
tx_buf	发送的数据
tx_len	发送数据的长度
rx_buf	接收的数据
rx_len	接收数据长度
transfer_mode	传输模式，发送或接收

uart_interrupt_mode_t

描述

UART 数据传输模式。

定义

```
typedef enum _uart_interrupt_mode
{
    UART_SEND = 1,
    UART_RECEIVE = 2,
} uart_interrupt_mode_t;
```

成员

成员名称	描述
UART_SEND	发送
UART_RECEIVE	接收

uart_work_mode_t

描述

UART 工作模式。

定义

```
typedef enum _uart_work_mode
{
    UART_NORMAL,
    UART_IRDA,
    UART_RS485_FULL_DUPLEX,
    UART_RS485_HALF_DUPLEX,
```

```
} uart_work_mode_t;
```

成员

成员名称	描述
UART_NORMAL	普通 UART
UART_IRDA	红外
UART_RS485_FULL_DUPLEX	全双工 RS485
UART_RS485_HALF_DUPLEX	半双工 RS485

uart_rs485_rede_t

描述

RS485 re de 管脚。

定义

```
typedef enum _uart_rs485_rede
{
    UART_RS485_DE,
    UART_RS485_RE,
    UART_RS485_REDE,
} uart_rs485_rede_t;
```

成员

成员名称	描述
UART_RS485_DE	选择 de 管脚
UART_RS485_RE	选择 re 管脚
UART_RS485_REDE	同时选择 de re 管脚

uart_polarity_t

描述

re de 有效时的极性。

定义

```
typedef enum _uart_polarity
{
    UART_LOW,
    UART_HIGH,
```

```
} uart_polarity_t;
```

成员

成员名称	描述
UART_LOW	低电平
UART_HIGH	高电平

uart_tat_mode_t

描述

re de 转换的模式。

定义

```
typedef enum _uart_tat_mode
{
    UART_DE_TO_RE,
    UART_RE_TO_DE,
    UART_TAT_ALL,
} uart_tat_mode_t;
```

成员

成员名称	描述
UART_DE_TO_RE	de 有效转 re 有效
UART_RE_TO_DE	re 有效转 de 有效
UART_TAT_ALL	de 转 re re 转 de

uart_det_mode_t

描述

de 有效无效选择。

定义

```
typedef enum _uart_det_mode
{
    UART_DE_ASSERTION,
    UART_DE_DE_ASSERTION,
    UART_DE_ALL,
} uart_det_mode_t;
```

成员

成员名称	描述
UART_DE_ASSERTION	de 有效
UART_DE_DE_ASSERTION	de 无效
UART_DE_ALL	de 有效 无效

对应的头文件 uarths.h

uarths_init

描述

初始化 UARTHS，系统默认波特率为 115200 8bit 1 位停止位 无检验位。因为 uarths 时钟源为 PLL0，在设置 PLL0 后需要重新调用该函数设置波特率，否则会打印乱码。

函数原型

```
void uarths_init(void)
```

参数

无。

返回值

无。

uarths_config

描述

设置 UARTHS 的参数。默认 8bit 数据，无校验位。

函数原型

```
void uarths_config(uint32_t baud_rate, uarths_stopbit_t stopbit)
```

参数

参数名称	描述	输入输出
baud_rate	波特率	输入
stopbit	停止位	输入

返回值

无。

uarths_receive_data

描述

通过 UARTHS 读取数据。

函数原型

```
size_t uarths_receive_data(uint8_t *buf, size_t buf_len)
```

参数

参数名称	描述	输入输出
buf	接收数据	输出
buf_len	接收数据的长度	输入

返回值

已接收到的数据长度。

uarths_send_data

描述

通过 UART 发送数据。

函数原型

```
size_t uarths_send_data(const uint8_t *buf, size_t buf_len)
```

参数

参数名称	描述	输入输出
buf	待发送数据	输入
buf_len	待发送数据的长度	输入

返回值

已发送数据的长度。

uarths_set_irq

描述

设置 UARTHS 中断回调函数。

函数原型

```
void uarths_set_irq(uarths_interrupt_mode_t interrupt_mode,
plic_irq_callback_t uarths_callback, void *ctx, uint32_t priority)
```

参数

参数名称	描述	输入输出
interrupt_mode	中断类型	输入
uarths_callback	中断回调函数	输入
ctx	回调函数的参数	输入
priority	中断优先级	输入

返回值

无。

uarths_get_interrupt_mode

描述

获取 UARTHS 的中断类型。接收、发送或接收发送同时中断。

函数原型

```
uarths_interrupt_mode_t uarths_get_interrupt_mode(void)
```

参数

无

返回值

当前中断的类型。

uarths_set_interrupt_cnt

描述

设置 UARTHS 中断时的 FIFO 深度。 当中断类型为 UARTHS_SEND_RECEIVE，发送接收 FIFO 中断深度均为 cnt;

函数原型

```
void uarths_set_interrupt_cnt(uarths_interrupt_mode_t interrupt_mode, uint8_t cnt)
```

参数

参数名称	描述	输入输出
interrupt_mode	中断类型	输入
cnt	FIFO 深度	输入

返回值

无。

举例

```
/* 设置接收中断 中断 FIFO 深度为 0，即接收到数据立即中断并读取接收到的数据。*/
    int uarths_irq(void *ctx)
    {
        if(!uarths_receive_data((uint8_t *)&receive_char, 1))
            printf("Uarths receive ERR!\n");
        return 0;
    }

    plic_init();
    uarths_set_interrupt_cnt(UARTHS_RECEIVE , 0);
    uarths_set_irq(UARTHS_RECEIVE ,uarths_irq, NULL, 4);
    sysctl_enable_irq();
```

数据类型

相关数据类型、数据结构定义如下：

- uarths_interrupt_mode_t：中断类型。

- uarths_stopbit_t：停止位。

uarths_interrupt_mode_t

描述

UARTHS 中断类型。

定义

```
typedef enum _uarths_interrupt_mode
{
    UARTHS_SEND = 1,
    UARTHS_RECEIVE = 2,
    UARTHS_SEND_RECEIVE = 3,
} uarths_interrupt_mode_t;
```

成员

成员名称	描述
UARTHS_SEND	发送中断
UARTHS_RECEIVE	接收中断
UARTHS_SEND_RECEIVE	发送接收中断

uarths_stopbit_t

描述：

UARTHS 停止位。

定义

```
typedef enum _uarths_stopbit
{
    UART_STOP_1,
    UART_STOP_2
} uarths_stopbit_t;
```

成员

成员名称	描述
UART_STOP_1	1 位停止位
UART_STOP_2	2 位停止位