UART中断

刘天驰

UART-interrupt是一种用于处理串行通信的机制。

在使用 UART 进行数据传输时，设备通过串行接口接收和发送数据。为了避免设备在接收或发送每个字节时都需要主动轮询，UART中断机制可以在特定事件发生时触发中断，以简化通信管理。

通常有两类常见的 UART 中断：

- 接收中断 (RX Interrupt)：当 UART 接收到数据时，UART 控制器会触发接收中断。此时，处理器会停止当前任务，进入中断处理程序，以读取缓冲区中的接收到的数据
- 发送中断 (TX Interrupt)：当 UART 控制器的发送缓冲区为空，可以发送新的数据时，它会触发发送中断，通知处理器可以向 UART 控制器发送更多数据

工作原理：

当接收缓冲区中有数据或发送缓冲区可以接受新数据时，UART 会向处理器发出中断请求 (IRQ)，处理器会执行预先定义的中断处理程序。

在中断处理程序中，开发者可以读取接收到的数据或写入数据进行发送，而不需要持续检测缓冲区的状态，从而提高了系统效率

```c
int main()
{
    io_mux_init();
    plic_init();
    sysctl_enable_irq();

    gpiohs_set_drive_mode(3, GPIO_DM_OUTPUT);
    gpio_pin_value_t value = GPIO_PV_HIGH;
    gpiohs_set_pin(3, value);

    uart_init(UART_NUM);
    uart_configure(UART_NUM, 115200, 8, UART_STOP_1, UART_PARITY_NONE);

    uart_set_receive_trigger(UART_NUM, UART_RECEIVE_FIFO_8);
    uart_irq_register(UART_NUM, UART_RECEIVE, on_uart_recv, NULL, 2);

    uart_set_send_trigger(UART_NUM, UART_SEND_FIFO_0);
    uint32_t v_uart_num = UART_NUM;
    uart_irq_register(UART_NUM, UART_SEND, on_uart_send, &v_uart_num, 2);

    char *hel = {"hello world!\n"};
    uart_send_data(UART_NUM, hel, strlen(hel));

    while(1);
}
```

```c
#define UART_NUM    UART_DEVICE_3
```

```c
void io_mux_init(void)
{
    fpioa_set_function(4, FUNC_UART1_RX + UART_NUM * 2);
    fpioa_set_function(5, FUNC_UART1_TX + UART_NUM * 2);
    fpioa_set_function(10, FUNC_GPIOHS3);
}
```

7.3.1 fpioa_set_function

7.3.1.1 描述

设置 IO0-IO47 管脚复用功能。

7.3.1.2 函数原型

```c
int fpioa_set_function(int number, fpioa_function_t function)
```

7.3.1.3 参数

参数名称	描述	输入输出
number	IO 管脚号	输入
function	FPIOA 功能号	输入

```c
FUNC_UART1_RX           = 64,       /*!< UART1 Receiver */
FUNC_UART1_TX           = 65,       /*!< UART1 Transmitter */
FUNC_UART2_RX           = 66,       /*!< UART2 Receiver */
FUNC_UART2_TX           = 67,       /*!< UART2 Transmitter */
FUNC_UART3_RX           = 68,       /*!< UART3 Receiver */
FUNC_UART3_TX           = 69,       /*!< UART3 Transmitter */
```

```c
int main()
{
    io_mux_init();
    plic_init();
    sysctl_enable_irq();

    gpiohs_set_drive_mode(3, GPIO_DM_OUTPUT);
    gpio_pin_value_t value = GPIO_PV_HIGH;
    gpiohs_set_pin(3, value);

    uart_init(UART_NUM);
    uart_configure(UART_NUM, 115200, 8, UART_STOP_1, UART_PARITY_NONE);

    uart_set_receive_trigger(UART_NUM, UART_RECEIVE_FIFO_8);
    uart_irq_register(UART_NUM, UART_RECEIVE, on_uart_recv, NULL, 2);

    uart_set_send_trigger(UART_NUM, UART_SEND_FIFO_0);
    uint32_t v_uart_num = UART_NUM;
    uart_irq_register(UART_NUM, UART_SEND, on_uart_send, &v_uart_num, 2);

    char *hel = {"hello world!\n"};
    uart_send_data(UART_NUM, hel, strlen(hel));

    while(1);
}
```

中断 PLIC

4.1 概述

可以将任一外部中断源单独分配到每个 CPU 的外部中断上。这提供了强大的灵活性，能适应不同的应用需求。

4.2 功能描述

PLIC 模块具有以下功能：

- 启用或禁用中断
- 设置中断处理程序
- 配置中断优先级

4.3.1 plic_init

4.3.1.1 描述

PLIC 初始化外部中断。

4.3.1.2 函数原型

```c
void plic_init(void)
```

```c
int main()
{
    io_mux_init();
    plic_init();
    sysctl_enable_irq();

    gpiohs_set_drive_mode(3, GPIO_DM_OUTPUT);
    gpio_pin_value_t value = GPIO_PV_HIGH;
    gpiohs_set_pin(3, value);

    uart_init(UART_NUM);
    uart_configure(UART_NUM, 115200, 8, UART_STOP_1, UART_PARITY_NONE);

    uart_set_receive_trigger(UART_NUM, UART_RECEIVE_FIFO_8);
    uart_irq_register(UART_NUM, UART_RECEIVE, on_uart_recv, NULL, 2);

    uart_set_send_trigger(UART_NUM, UART_SEND_FIFO_0);
    uint32_t v_uart_num = UART_NUM;
    uart_irq_register(UART_NUM, UART_SEND, on_uart_send, &v_uart_num, 2);

    char *hel = {"hello world!\n"};
    uart_send_data(UART_NUM, hel, strlen(hel));

    while(1);
}
```

21.3.15 sysctl_enable_irq

21.3.15.1 描述

使能系统中断，如果使用中断一定要开启系统中断。

21.3.15.2 函数原型

```c
void sysctl_enable_irq(void)
```

```c
int main()
{
    io_mux_init();
    plic_init();
    sysctl_enable_irq();

    gpiohs_set_drive_mode(3, GPIO_DM_OUTPUT);
    gpio_pin_value_t value = GPIO_PV_HIGH;
    gpiohs_set_pin(3, value);

    uart_init(UART_NUM);
    uart_configure(UART_NUM, 115200, 8, UART_STOP_1, UART_PARITY_NONE);

    uart_set_receive_trigger(UART_NUM, UART_RECEIVE_FIFO_8);
    uart_irq_register(UART_NUM, UART_RECEIVE, on_uart_recv, NULL, 2);

    uart_set_send_trigger(UART_NUM, UART_SEND_FIFO_0);
    uint32_t v_uart_num = UART_NUM;
    uart_irq_register(UART_NUM, UART_SEND, on_uart_send, &v_uart_num, 2);

    char *hel = {"hello world!\n"};
    uart_send_data(UART_NUM, hel, strlen(hel));

    while(1);
}
```

6.3.1 gpiohs_set_drive_mode

6.3.1.1 描述

设置 GPIO 驱动模式。

6.3.1.2 函数原型

```c
void gpiohs_set_drive_mode(uint8_t pin, gpio_drive_mode_t mode)
```

6.3.1.3 参数

参数名称	描述	输入输出
pin	GPIO 管脚	输入
mode	GPIO 驱动模式	输入

5.4.1 gpio_drive_mode_t

5.4.1.1 描述

GPIO 驱动模式。

5.4.1.2 定义

```c
typedef enum _gpio_drive_mode
{
    GPIO_DM_INPUT,
    GPIO_DM_INPUT_PULL_DOWN,
    GPIO_DM_INPUT_PULL_UP,
    GPIO_DM_OUTPUT,
} gpio_drive_mode_t;
```

5.4.1.3 成员

成员名称	描述
GPIO_DM_INPUT	输入
GPIO_DM_INPUT_PULL_DOWN	输入下拉
GPIO_DM_INPUT_PULL_UP	输入上拉
GPIO_DM_OUTPUT	输出

```c
int main()
{
    io_mux_init();
    plic_init();
    sysctl_enable_irq();

    gpiohs_set_drive_mode(3, GPIO_DM_OUTPUT);
    gpio_pin_value_t value = GPIO_PV_HIGH;
    gpiohs_set_pin(3, value);

    uart_init(UART_NUM);
    uart_configure(UART_NUM, 115200, 8, UART_STOP_1, UART_PARITY_NONE);

    uart_set_receive_trigger(UART_NUM, UART_RECEIVE_FIFO_8);
    uart_irq_register(UART_NUM, UART_RECEIVE, on_uart_recv, NULL, 2);

    uart_set_send_trigger(UART_NUM, UART_SEND_FIFO_0);
    uint32_t v_uart_num = UART_NUM;
    uart_irq_register(UART_NUM, UART_SEND, on_uart_send, &v_uart_num, 2);

    char *hel = {"hello world!\n"};
    uart_send_data(UART_NUM, hel, strlen(hel));

    while(1);
}
```

6.4.2 gpio_pin_value_t

6.4.2.1 描述

GPIO 值。

6.4.2.2 定义

```c
typedef enum _gpio_pin_value
{
    GPIO_PV_LOW,
    GPIO_PV_HIGH
} gpio_pin_value_t;
```

6.4.2.3 成员

成员名称	描述
GPIO_PV_LOW	低
GPIO_PV_HIGH	高

```c
int main()
{
    io_mux_init();
    plic_init();
    sysctl_enable_irq();

    gpiohs_set_drive_mode(3, GPIO_DM_OUTPUT);
    gpio_pin_value_t value = GPIO_PV_HIGH;
    gpiohs_set_pin(3, value);

    uart_init(UART_NUM);
    uart_configure(UART_NUM, 115200, 8, UART_STOP_1, UART_PARITY_NONE);

    uart_set_receive_trigger(UART_NUM, UART_RECEIVE_FIFO_8);
    uart_irq_register(UART_NUM, UART_RECEIVE, on_uart_recv, NULL, 2);

    uart_set_send_trigger(UART_NUM, UART_SEND_FIFO_0);
    uint32_t v_uart_num = UART_NUM;
    uart_irq_register(UART_NUM, UART_SEND, on_uart_send, &v_uart_num, 2);

    char *hel = {"hello world!\n"};
    uart_send_data(UART_NUM, hel, strlen(hel));

    while(1);
}
```

11.3.3 uart_configure

11.3.3.1 描述

设置 UART 相关参数。

11.3.3.2 函数原型

```c
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)
```

11.3.3.3 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

```c
int main()
{
    io_mux_init();
    plic_init();
    sysctl_enable_irq();

    gpiohs_set_drive_mode(3, GPIO_DM_OUTPUT);
    gpio_pin_value_t value = GPIO_PV_HIGH;
    gpiohs_set_pin(3, value);

    uart_init(UART_NUM);
    uart_configure(UART_NUM, 115200, 8, UART_STOP_1, UART_PARITY_NONE);

接  uart_set_receive_trigger(UART_NUM, UART_RECEIVE_FIFO_8);
收  uart_irq_register(UART_NUM, UART_RECEIVE, on_uart_recv, NULL, 2);

发  uart_set_send_trigger(UART_NUM, UART_SEND_FIFO_0);
出  uint32_t v_uart_num = UART_NUM;
    uart_irq_register(UART_NUM, UART_SEND, on_uart_send, &v_uart_num, 2);

    char *hel = {"hello world!\n"};
    uart_send_data(UART_NUM, hel, strlen(hel));

    while(1);
}
```

UART触发中断

11.4.7 uart_receive_trigger_t

11.4.7.1 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。

11.4.7.2 定义

```c
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

```c
int main()
{
    io_mux_init();
    plic_init();
    sysctl_enable_irq();

    gpiohs_set_drive_mode(3, GPIO_DM_OUTPUT);
    gpio_pin_value_t value = GPIO_PV_HIGH;
    gpiohs_set_pin(3, value);

    uart_init(UART_NUM);
    uart_configure(UART_NUM, 115200, 8, UART_STOP_1, UART_PARITY_NONE);

接  uart_set_receive_trigger(UART_NUM, UART_RECEIVE_FIFO_8);
收  uart_irq_register(UART_NUM, UART_RECEIVE, on_uart_recv, NULL, 2);

发  uart_set_send_trigger(UART_NUM, UART_SEND_FIFO_0);
出  uint32_t v_uart_num = UART_NUM;
    uart_irq_register(UART_NUM, UART_SEND, on_uart_send, &v_uart_num, 2);

    char *hel = {"hello world!\n"};
    uart_send_data(UART_NUM, hel, strlen(hel));

    while(1);
}
```

UART触发中断

11.4.6 uart_send_trigger_t

11.4.6.1 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。

11.4.6.2 定义

```c
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;
```

11.4.6.3 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

```c
int main()
{
    io_mux_init();
    plic_init();
    sysctl_enable_irq();

    gpiohs_set_drive_mode(3, GPIO_DM_OUTPUT);
    gpio_pin_value_t value = GPIO_PV_HIGH;
    gpiohs_set_pin(3, value);

    uart_init(UART_NUM);
    uart_configure(UART_NUM, 115200, 8, UART_STOP_1, UART_PARITY_NONE);

    uart_set_receive_trigger(UART_NUM, UART_RECEIVE_FIFO_8);
    uart_irq_register(UART_NUM, UART_RECEIVE, on_uart_recv, NULL, 2);

    uart_set_send_trigger(UART_NUM, UART_SEND_FIFO_0);
    uint32_t v_uart_num = UART_NUM;
    uart_irq_register(UART_NUM, UART_SEND, on_uart_send, &v_uart_num, 2);

    char *hel = {"hello world!\n"};
    uart_send_data(UART_NUM, hel, strlen(hel));

    while(1);
}
```

UART触发中断，开始传输数据

11.3.10 uart_irq_register

11.3.10.1 描述

注册 UART 中断函数。

11.3.10.2 函数原型

```c
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)
```

11.3.10.3 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

```c
int main()
{
    io_mux_init();
    plic_init();
    sysctl_enable_irq();

    gpiohs_set_drive_mode(3, GPIO_DM_OUTPUT);
    gpio_pin_value_t value = GPIO_PV_HIGH;
    gpiohs_set_pin(3, value);

    uart_init(UART_NUM);
    uart_configure(UART_NUM, 115200, 8, UART_STOP_1, UART_PARITY_NONE);

    uart_set_receive_trigger(UART_NUM, UART_RECEIVE_FIFO_8);
    uart_irq_register(UART_NUM, UART_RECEIVE, on_uart_recv, NULL, 2);

    uart_set_send_trigger(UART_NUM, UART_SEND_FIFO_0);
    uint32_t v_uart_num = UART_NUM;
    uart_irq_register(UART_NUM, UART_SEND, on_uart_send, &v_uart_num, 2);

    char *hel = {"hello world!\n"};
    uart_send_data(UART_NUM, hel, strlen(hel));

    while(1);
}
```

中断回调是指在发生中断事件时，系统或硬件自动调用的一个预先定义好的函数（即回调函数）

```c
volatile uint32_t recv_flag = 0;
char g_cmd[4];
volatile uint8_t g_cmd_cnt = 0;
int on_uart_recv(void *ctx)
{
    char v_buf[8];
    int ret =  uart_receive_data(UART_NUM, v_buf, 8);
    for(uint32_t i = 0; i < ret; i++)
    {
        if(v_buf[i] == 0x55 && (recv_flag == 0 || recv_flag == 1))
        {
            recv_flag = 1;
            continue;
        }
        else if(v_buf[i] == 0xAA && recv_flag == 1)
        {
            recv_flag = 2;
            g_cmd_cnt = 0;
            continue;
        }
        else if(recv_flag == 2 && g_cmd_cnt < CMD_LENTH)
        {
            g_cmd[g_cmd_cnt++] = v_buf[i];
            if(g_cmd_cnt >= CMD_LENTH)
            {
                release_cmd(g_cmd);
                recv_flag = 0;
            }
            continue;
        }
        else
        {
            recv_flag = 0;
        }
    }
    return 0;
}
```

北京大学 PEKING UNIVERSITY

中断回调

```c
volatile uint32_t recv_flag = 0;
char g_cmd[4];
volatile uint8_t g_cmd_cnt = 0;
int on_uart_recv(void *ctx)
{
    char v_buf[8];
    int ret =  uart_receive_data(UART_NUM, v_buf, 8);
    for(uint32_t i = 0; i < ret; i++)
    {
        if(v_buf[i] == 0x55 && (recv_flag == 0 || recv_flag == 1))
        {
            recv_flag = 1;
            continue;
        }
        else if(v_buf[i] == 0xAA && recv_flag == 1)
        {
            recv_flag = 2;
            g_cmd_cnt = 0;
            continue;
        }
        else if(recv_flag == 2 && g_cmd_cnt < CMD_LENTH)
        {
            g_cmd[g_cmd_cnt++] = v_buf[i];
            if(g_cmd_cnt >= CMD_LENTH)
            {
                release_cmd(g_cmd);
                recv_flag = 0;
            }
            continue;
        }
        else
        {
            recv_flag = 0;
        }
    }
    return 0;
}
```

4

11.3.7 uart_receive_data

11.3.7.1 描述

通过 UART 读取数据。

11.3.7.2 函数原型

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

11.3.7.3 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

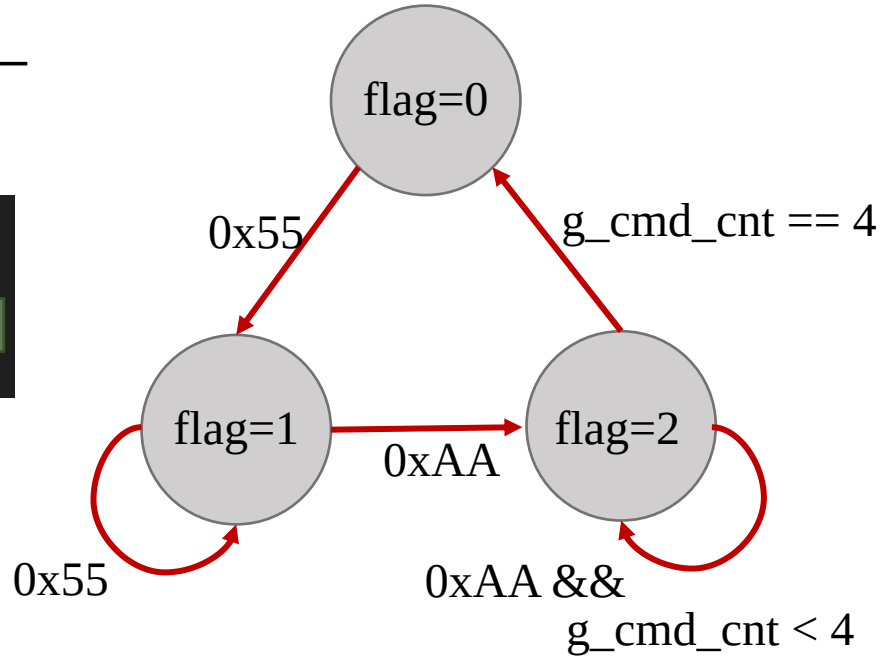
11.3.7.4 返回值

已接收到的数据长度。

```
- LED on when receive 55AA55555555

- LED off when receive 55AAAAAAAAAA
```

FSM



北京大学 PEKING UNIVERSITY

中断回调

```c
volatile uint32_t recv_flag = 0;
char g_cmd[4];
volatile uint8_t g_cmd_cnt = 0;
int on_uart_recv(void *ctx)
{
    char v_buf[8];
    int ret =  uart_receive_data(UART_NUM, v_buf, 8);
    for(uint32_t i = 0; i < ret; i++)
    {
        if(v_buf[i] == 0x55 && (recv_flag == 0 || recv_flag == 1))
        {
            recv_flag = 1;
            continue;
        }
        else if(v_buf[i] == 0xAA && recv_flag == 1)
        {
            recv_flag = 2;
            g_cmd_cnt = 0;
            continue;
        }
        else if(recv_flag == 2 && g_cmd_cnt < CMD_LENTH)
        {
            g_cmd[g_cmd_cnt++] = v_buf[i];
            if(g_cmd_cnt >= CMD_LENTH)
            {
                release_cmd(g_cmd);
                recv_flag = 0;
            }
            continue;
        }
        else
        {
            recv_flag = 0;
        }
    }
    return 0;
}
```

```c
#define CLOSLIGHT   0x55555555
#define OPENLIGHT   0xAAAAAAAA
```

```c
int release_cmd(char *cmd)
{
    switch(*((int *)cmd))
    {
        case CLOSLIGHT:
        gpiohs_set_pin(3, GPIO_PV_LOW);
        break;
        case OPENLIGHT:
        gpiohs_set_pin(3, GPIO_PV_HIGH);
        break;
    }
    return 0;
}
```

```c
int main()
{
    io_mux_init();
    plic_init();
    sysctl_enable_irq();

    gpiohs_set_drive_mode(3, GPIO_DM_OUTPUT);
    gpio_pin_value_t value = GPIO_PV_HIGH;
    gpiohs_set_pin(3, value);

    uart_init(UART_NUM);
    uart_configure(UART_NUM, 115200, 8, UART_STOP_1, UART_PARITY_NONE);

    uart_set_receive_trigger(UART_NUM, UART_RECEIVE_FIFO_8);
    uart_irq_register(UART_NUM, UART_RECEIVE, on_uart_recv, NULL, 2);

    uart_set_send_trigger(UART_NUM, UART_SEND_FIFO_0);
    uint32_t v_uart_num = UART_NUM;
    uart_irq_register(UART_NUM, UART_SEND, on_uart_send, &v_uart_num, 2);

    char *hel = {"hello world!\n"};
    uart_send_data(UART_NUM, hel, strlen(hel));

    while(1);
}
```

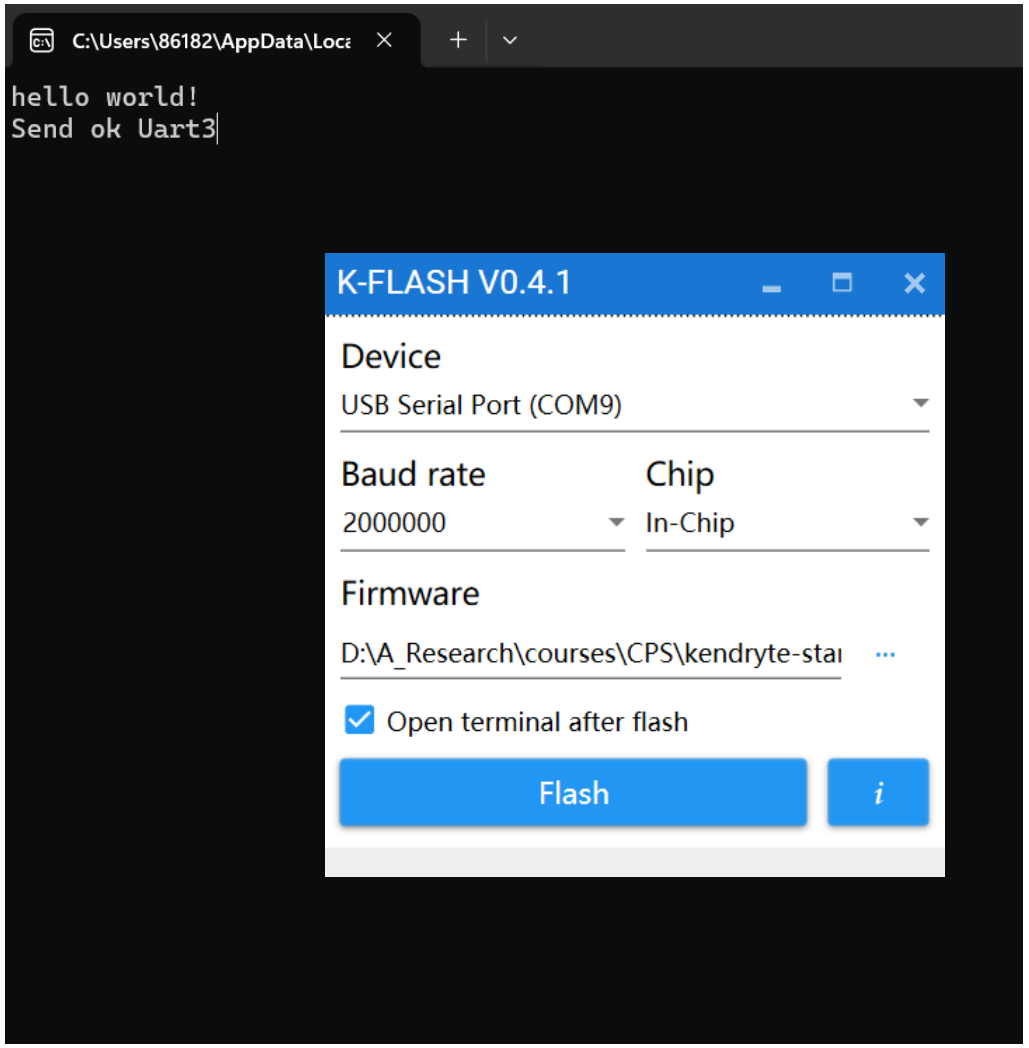
中断回调是指在发生中断事件时，系统或硬件自动调用的一个预先定义好的函数（即回调函数）

```c
int on_uart_send(void *ctx)
{
    uint8_t v_uart = *((uint32_t *)ctx) + 1 + 0x30;
    uart_irq_unregister(UART_NUM, UART_SEND);
    char *v_send_ok = "Send ok Uart";
    uart_send_data(UART_NUM, v_send_ok,strlen(v_send_ok));
    uart_send_data(UART_NUM, (char *)&v_uart,1);
    return 0;
}
```

- 从地址ctx中读取一个 uint32_t 类型的值后转换为字符
- 取消 UART 的发送中断注册
- 发送一条固定的字符串 “Send ok Uart”
- 发送一个字符
- 最后返回 0，表示成功处理。

北京大学 PEKING UNIVERSITY

发送中断——Send ok Uart



接收中断——灯亮灯关

Thanks for listening!