3. Serial communication

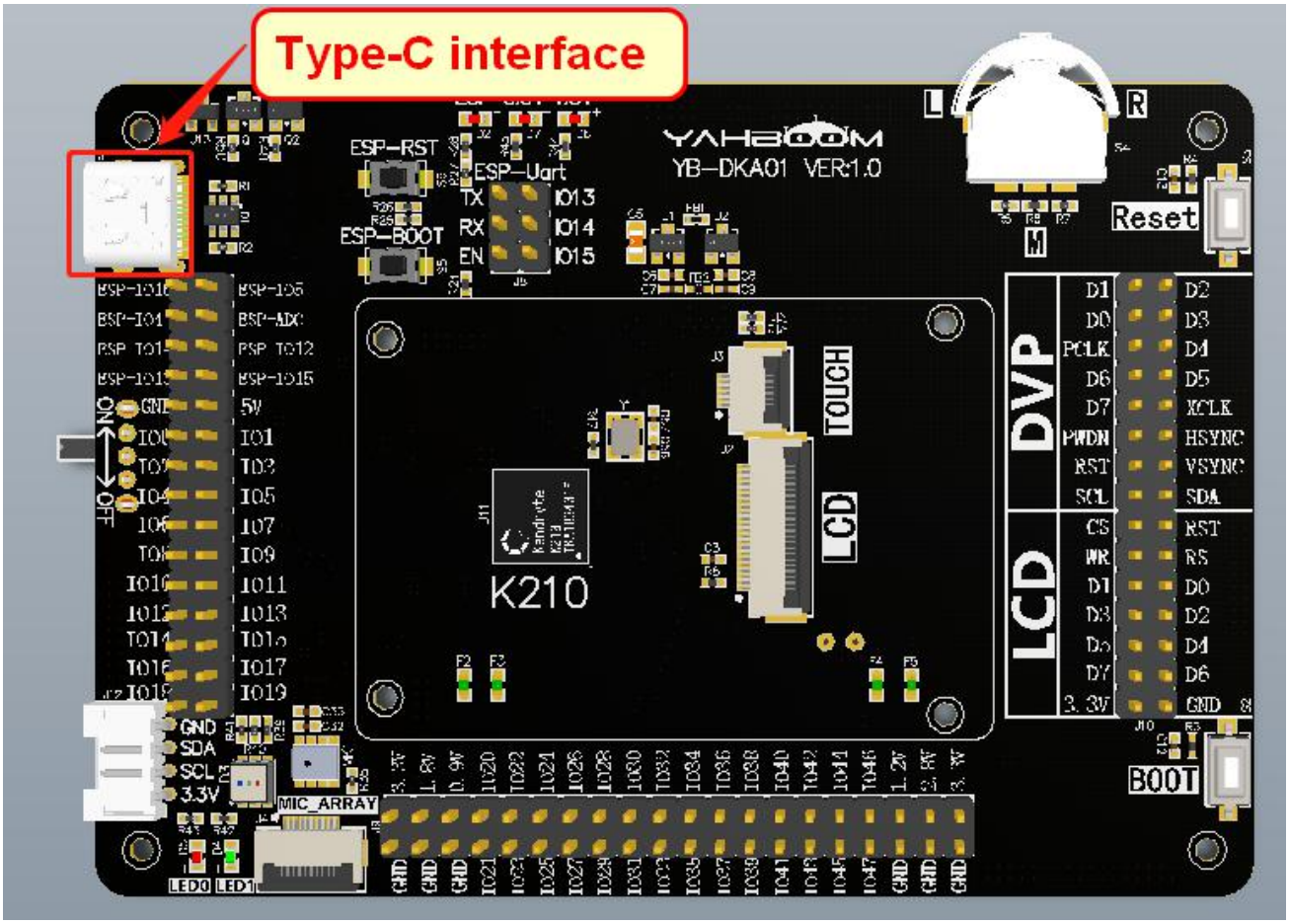
1. Experiment purpose

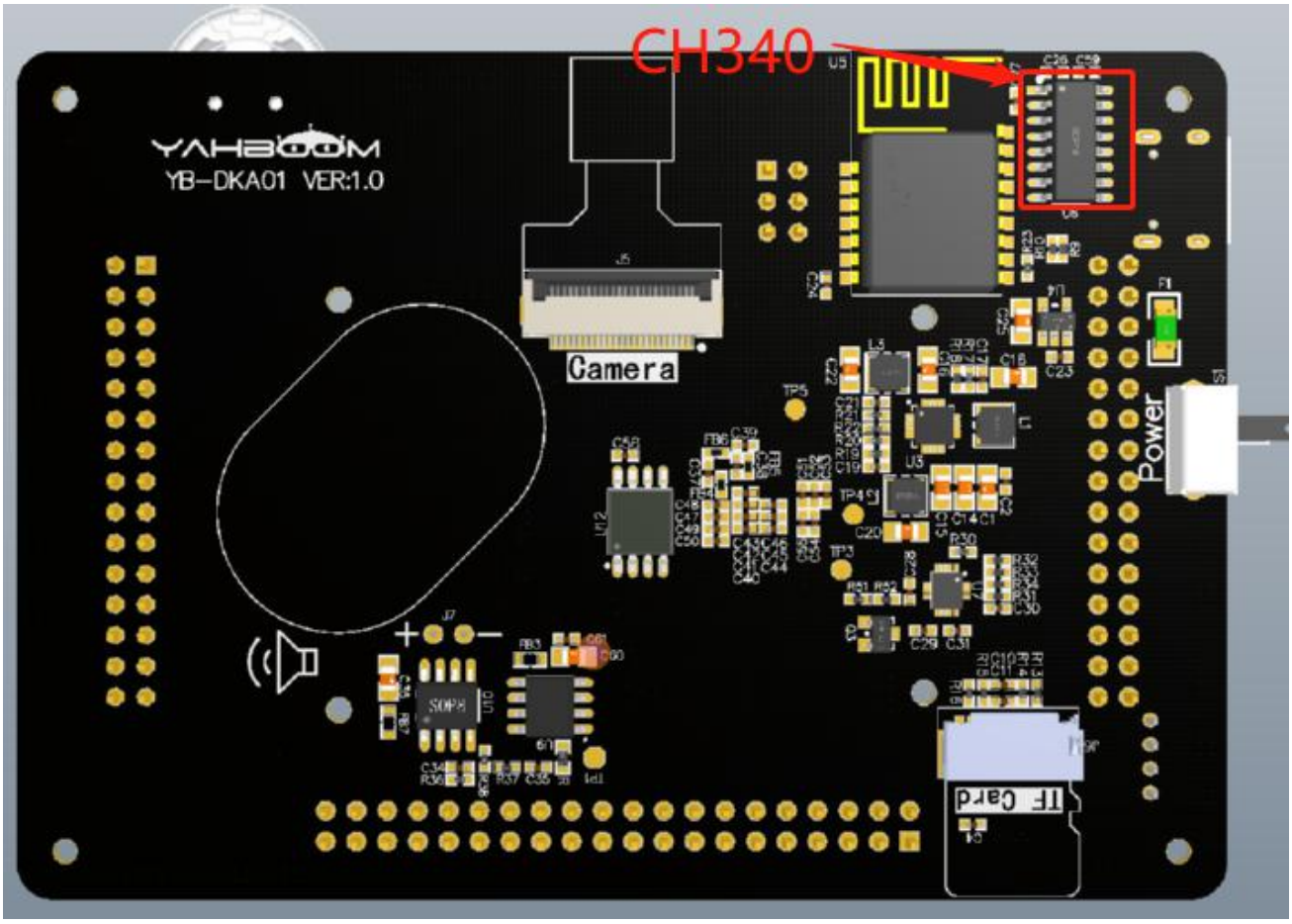
In this lesson mainly learns the serial communication.

2.Experiment preparation

2.1 components

Type-C interface and CH340 Serial port chip





2.2 Component characteristics

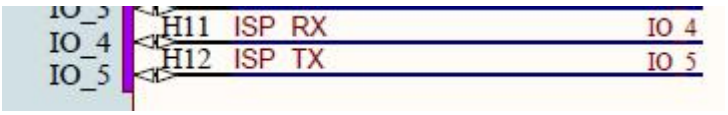
Tpye-C interface is connected to the serial chip, which can be used power supply and serial data transmission. The download program based on serial data transmission function.

Type-C interface is the current mainstream data transmission connection interface.

CH340 serial port hip is a USB bus adapter chip, which can realize USB to serial port. It possess full-speed USB device interface, hardware full-duplex serial port, built-in transceiver buffer, supports baud rate of 50bps~2Mbps, and is compatible with USBV2.0.

2.3 Hardware connection

Type-C port connect IO4 and IO5 pin of K210. IO4 is the receiving pin, IO5 is the sending pin.



2.4 SDK API function

Header file is uart.h

General UART is UART1, UART2 and UART3, support asynchronous communication (RS232, RS485 and IRDA, the communication rate can up to 5Mbps. UART supports hardware

management of CTS and RTS signals and software flow control (XON and XOFF).
All three interfaces can be accessed by DMA or can be accessed directly by CPU. Each transmission data is 8 bytes, it supports asynchronous clocks and the data clock can be configured separately to realize the full duplex mode, which can ensure data synchronization in the two clock domains.

uart the default is RS232 mode, it can be configured as software-programmable RS485 mode.

We can use THRE interrupt mode to improve serial port performance. When THRE mode and FIFO mode are selected, if the FIFO is less than the threshold, the THRE interrupt will be triggered.

We will provide the following ports for users.
• uart_init: Initialize uart
• uart_config (Versions after 0.6.0 are no longer supported, please use uart_configure)
• uart_configure: Configure the baud rate of serial port, etc
• uart_send_data: Send data through a serial port.
• uart_send_data_dma: Send data by DMA channels.
• uart_send_data_dma_irq: UART sends data through DMA and registers the DMA receive completion interrupt function, only a single interrupt.
• uart_receive_data: Read serial port data.
• uart_receive_data_dma: Serial port receive data by dma.
• uart_receive_data_dma_irq: UART receive data through DMA and registers the DMA receive completion interrupt function, only a single interrupt.
• uart_irq_register: Register the serial port interrupt function.
• uart_irq_deregister: Log off serial port interruption.
• uart_set_work_mode: Set uart working mode. There are four modes: ordinary uart, infrared, RS485 full duplex, RS485 half duplex.
• uart_set_rede_polarity: Set the polarity when the RS485 re de pin is valid.
• uart_set_rede_enable: Enable re de pin, mainly used in rs485 full-duplex mode, re and de must be controlled manually. Single-duplex mode does not need to call this function, the hardware will automatically set re de in single-duplex mode.
• uart_set_tat: Configure the time intervals between re de pins, which are related to the external 485 modules.
• uart_set_det: Set the time delay of the data, when de is valid and invalid conversion.
• uart_debug_init: Configure the debug serial port. The system defaults to use UART3 as the debug serial port. Users can customize which uart to use as the debug serial port by calling this function. If you use UART1 or UART2, you need to use fpioa to set the pins. Therefore, the debug pin is not limited to fpioa4, 5.
• uart_handle_data_dma: UART transfers data by DMA

High-speed universal asynchronous receiver transmitter (UARTHS) corresponding to the header file uarths.h

High-speed UART is UARTHS (UART0), the communication rate can up to 5Mbps, 8-byte transmission and reception FIFO, programmable THRE interrupt, does not support hardware flow control or other modem control signals, or synchronous serial data converters. By default, the system printf debugging function calls the UARTHS serial port to send data.

We provide following interfaces for users:
• uarths_init: Initialize UARTHS, the system default baud rate is 115200. Data bit is 8. Stop bit is bit 1. Parity bit NONE. stop bit without check bit. Because the uarths clock source is PLL0, you need to call this function again to set the baud rate after setting PLL0, otherwise it will print garbled characters.
• uarths_config: Set the parameters of UARTHS. Default 8bit data, Parity bit NONE.
• uarths_receive_data: Read data through UARTHS.
• uarths_send_data: Sending data through UART.
• uarths_set_irq: Set the UARTHS interrupt callback function.
• uarths_get_interrupt_mode: Get the interrupt type of UARTHS. Receiving, sending, or receiving and sending are interrupted at the same time.
• uarths_set_interrupt_cnt: Set the FIFO depth when UARTHS is interrupted. When the interrupt type is UARTHS_SEND_RECEIVE, the transmit and receive FIFO interrupt depths are cnt;

3. Experimental principle

Serial communication refers to a communication method that transmits data (sending and receiving) bit by bit between peripherals and computers through data signal lines, ground wires, etc.,.
Its transmission process is to transmit character by character, each character is transmitted bit by bit, and when a character is transmitted, it always starts with "start bit" and ends with "stop bit". There is no fixed time interval requirement between characters, but the data is the LOW bit in the front, the HIGH bit in the back, the last one is parity bit.

serial number	Name	Remarks
1	start bit	Indicates the beginning of the transmission character, a logic "0" signal.
2	data bits	The number of data bits can be 5, 6, 7, 8, etc. to form a character Start transmission from the lowest bit, locationing by clock
3	parity bit	After adding this bit to the data bit, the number of "1" bits should be even (even parity) or odd (odd parity) to verify the correctness of data transmission
4	Stop bit	The end mark of a character data can be 1-bit, 1.5-bit, 2-bit high level

The serial port supports full-duplex communication, that is, while using one wire to send data, use another wire to receive data. The most important parameters of serial communication are baud rate, data bit, stop bit and parity check. For two communication ports, these parameters must be set to the same.

4. Experiment procedure

4.1 According to the above hardware connection pin diagram, K210 hardware pins and software functions use FPIOA mapping relationship.
!Note: All the operations in the program are software pins, so you need to map the hardware pins to software GPIO functions. Then, you can directly operate the software GPIO.

```
/*****************************HARDWARE-PIN*********************************/
//Hardware IO port, corresponding Schematic
#define PIN_UART_USB_RX       (4)
#define PIN_UART_USB_TX       (5)

/*****************************SOFTWARE-GPIO********************************/
//Software GPIO port, corresponding program
#define UART_USB_NUM          UART_DEVICE_3

/*****************************FUNC-GPIO************************************/
//Function of GPIO port, bound to hardware IO port
#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 mapping
    fpioa_set_function(PIN_UART_USB_RX, FUNC_UART_USB_RX);
    fpioa_set_function(PIN_UART_USB_TX, FUNC_UART_USB_TX);
}
```

4.2 Then, we need to initialize the serial port. We choose the serial port 3, set the baud rate to 115200, the serial port data width to 8 bits, the stop bit to 1 bit, and no parity bit is none.

```
//Initialize serial port 3, set the baud rate to 115200
uart_init(UART_USB_NUM);
uart_configure(UART_USB_NUM, 115200, UART_BITWIDTH_8BIT, UART_STOP_1, UART_PARITY_NONE);
```

4.3 Send "hello yahboom!" when booting, prompting that booting is complete.

```
/* Turn on send hello yahboom! */
char *hello = {"hello yahboom!\n"};
uart_send_data(UART_USB_NUM, hello, strlen(hello));
```

4.4 Waiting for the serial port data in a loop. If the serial port data is received, the received data is sent out through the serial port.

```
char recv = 0;

while (1)
{
    /* wait to receive data and send the received data by Serial port*/
    while(uart_receive_data(UART_USB_NUM, &recv, 1))
    {
        uart_send_data(UART_USB_NUM, &recv, 1);
    }
}
return 0;
```

4.5 Compile and debug, burn and run

Copy the uart folder to the src directory in the SDK.

Then, enter the build directory and run the following command to compile.

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

make

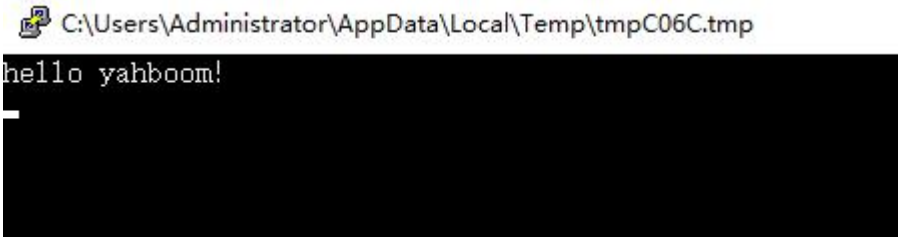
```
[ 97%] Building C object CMakeFiles/uart.dir/src/uart/main.c.obj
[100%] Linking C executable uart
Generating .bin file ...
[100%] Built target uart
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

After the compilation is complete, the uart.bin file will be generated in the build folder.

We need to use the type-C data cable to connect the computer and the K210 development board. Open kflash, select the corresponding device, and then burn the gpio_rgb.bin file to the K210 development board.

5. Experimental phenomenon

After the firmware is burned, a terminal interface will pop up. If the terminal interface does not pop up, we can open the serial port assistant to display the debugging content.



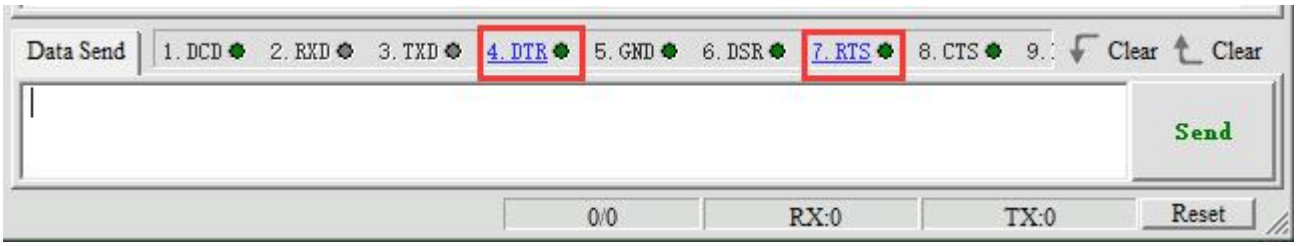
Open the serial port assistant of the computer, select the corresponding serial port number of the corresponding K210 development board, set the baud rate to 115200, and then click to open the serial port assistant.

! Note:

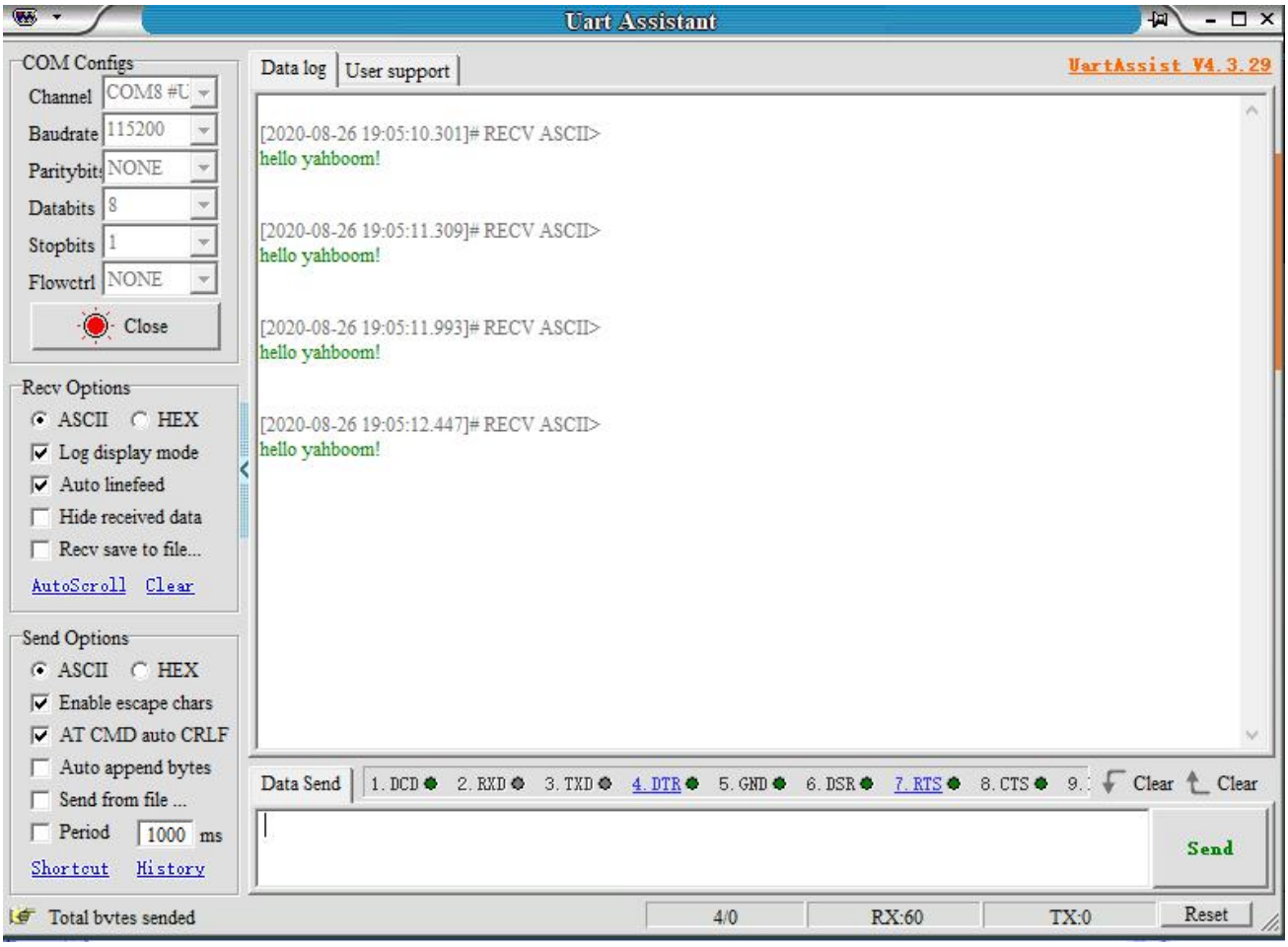
We also need to set the DTR and RTS of the serial port assistant.

At the bottom of the serial port assistant, we can see that 4.DTR and 7.RTS are red by default. Click 4.DTR and 7.RTS to set both to green, and then press the reset button of the K210 development

board.



From Serial port assistant, we can receive the “hello yahboom!”. Then, we can input the characters to be sent at the bottom, and then click send, the K210 will send the received data back and display it on the serial port assistant.



6. Experiment summary

6.1 There are three UART in total, namely UART1,UART2 and UART3

6.2 uart uses RS232 mode by default, and it can be additionally configured into programmable RS485 mode.

6.3 If uart pins are mapped to other hardware pins, we need to connect other serial chip such as CH340 to display data.

Appendix -- API

Header file is uart.h

1)uart_init

Function: initialize the uart

Function prototype: **void uart_init(uart_device_number_t channel)**

Parameter:

Parameter name	Description	Input/Output
channel	UART serial number	Input

Return value: No

2)uart_config

Function: Set UART related parameters. This function is obsolete, and the replacement function is uart_configure.

uart_configure

Function: Set UART related parameters.

Function prototype: **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)**

Parameter:

Parameter name	Description	Input/Output
channel	UART serial number	Input
baud_rate	baud rate	Input
data_width	data bits (5-8)	Input
stopbit	stop bits	Input
parity	parity bit	Input

Return value: No

3)uart_send_data

Function: Sending data by UART.

Function prototype: **int uart_send_data(uart_device_number_t channel, const char *buffer, size_t buf_len)**

Parameter:

Parameter name	Description	Input/Output
channel	UART serial number	Input
buffer	Data to be sent	Input
buf_len	The length of the data to be sent	Input

Return value: The length of the sent data.

4)uart_send_data_dma

Function: UART sends data via DMA. The data is returned after all sending.

Function prototype: **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)**
Parameter:

Parameter name	Description	Input/Output
uart_channel	UART serial number	Input
dmac_channel	DMA channel	Input
buffer	Data waiting to be sent	Input
buf_len	Length Data waiting to be sent	Input

Return value: No

5)uart_send_data_dma_irq
Function: UART sends data through DMA, and sets the DMA transmission completion interrupt function, only a single interrupt.
Function prototype: **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)**
Parameter:

Parameter name	Description	Input/Output
uart_channel	UART serial number	Input
dmac_channel	DMA channel	Input
buffer	Data waiting to be sent	Input
buf_len	Length Data waiting to be sent	Input
uart_callback	DMA interrupt callback	Input
ctx	interrupt function parameter	Input
priority	interrupt priority level	Input

Return value: No

6)uart_receive_data
Function: Read data through UART.
Function prototype: **int uart_receive_data(uart_device_number_t channel, char *buffer, size_t buf_len);**

Parameter:

Parameter name	Description	Input/Output
channel	UART serial number	Input
buffer	Receive data	Output
buf_len	The length of the received data	Input

Return value: The length of the received data

7)uart_receive_data_dma

Function: UART receives data via DMA
Function prototype: **void uart_receive_data_dma(uart_device_number_t uart_channel, dmac_channel_number_t dmac_channel, uint8_t *buffer, size_t buf_len)**

Parameter:

Parameter name	Description	Input/Output
uart_channel	UART serial number	Input
dmac_channel	DMA channel	Input
buffer	Receive data	Output
buf_len	The length of the received data	Input

Return value: No

8)uart_receive_data_dma_irq
Function: UART receives data through DMA, and registers the DMA reception completion interrupt function, only a single interrupt.
Function prototype: **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)**
Parameter:

Parameter name	Description	Input/Output
uart_channel	UART serial number	Input
dmac_channel	DMA channel	Input
buffer	Receive data	Output
buf_len	The length of the received data	Input
uart_callback	DMA interrupt callback	Input
ctx	interrupt function parameter	Input
priority	interrupt priority level	Input

Return value: No

9)uart_irq_register
Function: Register the UART interrupt function
Function prototype: **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)**
Parameter:

Parameter name	Description	Input/Output
channel	UART serial number	Input
interrupt_mode	interrupt type	Input
uart_callback	interrupt callback	Input
ctx	interrupt function parameter	Input
priority	interrupt priority level	Input

Return value: No

10)uart_irq_deregister

Function: Log off the UART interrupt function.

Function prototype: **void uart_irq_deregister(uart_device_number_t channel, uart_interrupt_mode_t interrupt_mode)**

Parameter:

Parameter name	Description	Input/Output
channel	UART serial number	Input
interrupt_mode	interrupt type	Input

Return value: No

11)uart_set_work_mode

Function: set uart work mode. There are four modes: ordinary UART, IR, RS485 full-duplex, RS485 half-duplex.

Function prototype: **void uart_set_work_mode(uart_device_number_t uart_channel, uart_work_mode_t work_mode)**

Parameter:

Parameter name	Description	Input/Output
channel	UART serial number	Input
work_mode	Work mode, see uart_work_mode_t structure description for details	Input

Return value: No

12) uart_set_rede_polarity

Function: Set the polarity when the RS485 re de pin is valid.

Function prototype: **void uart_set_rede_polarity(uart_device_number_t uart_channel, uart_rs485_rede_t rede, uart_polarity_t polarity)**

Parameter:

Parameter name	Description	Input/Output
uart_channel	UART serial number	Input
rede	re or de pins	Input
polarity	Effective time polarity	Input

Return value: No

13) uart_set_rede_enable

Function: Enable re de pin, mainly used in rs485 full duplex mode, re and de must be controlled manually. Single-duplex mode does not need to call this function, the hardware will automatically set re de in single-duplex mode.

Function prototype: **void uart_set_rede_enable(uart_device_number_t uart_channel, uart_rs485_rede_t rede, bool enable)**

Parameter:

Parameter name	Description	Input/Output
uart_channel	UART serial number	Input
rede	re or de pins	Input
enable	whether it is effective or not	Input

Return value: No

14) uart_set_tat

Function: Configure the time interval between rede and de, this is related to the external 485 module.

Function prototype: **void uart_set_tat(uart_device_number_t uart_channel, uart_tat_mode_t tat_mode, size_t time)**

Parameter:

Parameter name	Description	Input/Output
uart_channel	UART serial number	Input
tat_mode	Convert mode re to de or de to re	Input
time	Time (ns)	Input

Return value: No

15) uart_set_det

Function: Set de, When valid and invalid conversion, the time delay of data

Function prototype: **void uart_set_det(uart_device_number_t uart_channel, uart_det_mode_t det_mode, size_t time)**

Parameter:

Parameter name	Description	Input/Output
uart_channel	UART serial number	Input
det_mode	Convert mode, de invalid to valid or valid to invalid	Input
time	ns	Input

Return value: No

16) uart_debug_init

Function: Configure the debug serial port. The system defaults to use UART3 as the debug serial port. Users can customize which uart to use as the debug serial port by calling this function. If you use UART1 or UART2, you need to use fpioa to set the pins. Therefore, the debug pins are not limited to fpioa4 and 5.
Function prototype: **void uart_debug_init(uart_device_number_t uart_channel)**
Parameter:

Parameter name	Description	Input/Output
uart_channel	UART serial number	Input
el	-1 will use the last set UART by default	Input

Return value: No

17) uart_handle_data_dma
Function: UART transfers data through DMA.
Function prototype: **void uart_handle_data_dma(uart_device_number_t uart_channel ,uart_data_t data, plic_interrupt_t *cb)**
Parameter:

Parameter name	Description	Input/Output
uart_channel	UART serial number	Input
data	UART data related parameters, see i2c_data_t description for details	Input
cb	DMA interrupt callback function, if it is set to NULL, it is in blocking mode, and the function exits after the transmission is completed	Input

Return value: No

Type of data
The related data types and data structures are defined as follows:uart_device_number_t：UART serial number
uart_bitwidth_t：UART Data bit width.
uart_stopbits_t：UART stop bit
uart_parity_t：UART parity bit
uart_interrupt_mode_t：UART interrupt type, receive or send.
uart_send_trigger_t：send interrupt or DMA trigger FIFO depth.
uart_receive_trigger_t：receive interrupt or DMA trigger FIFO depth.
uart_data_t：Data related parameters when using dma transmission.
uart_interrupt_mode_t：transmission mode, receive or send.
uart_work_mode_t：UART work mode.
uart_rs485_rede_t：Select re de pin.
uart_polarity_t：de re polarity.
uart_tat_mode_t：de re convert selection
uart_det_mode_t：de Valid or invalid selection.

uart_device_number_t

Description: UART serial number

Define

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

Member

Member name	Description
UART_DEVICE_1	UART 1
UART_DEVICE_2	UART 2
UART_DEVICE_3	UART 3

uart_bitwidth_t

Description: UART Data bit width.

Define

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

Member

Member name	Description
UART_BITWIDTH_5BIT	5 bit
UART_BITWIDTH_6BIT	6 bit
UART_BITWIDTH_7BIT	7 bit
UART_BITWIDTH_8BIT	8 bit

uart_stopbits_t

Description: UART stop bit.

Define

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

Member

Member name	Description
UART_STOP_1	1 stop bit
UART_STOP_1_5	1.5 stop bit
UART_STOP_2	2 stop bit

uart_parity_t

Description: UART parity bit.

Define

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

Member

Member name	Description
UART_PARITY_NONE	No parity bit
UART_PARITY_ODD	Odd Parity bit
UART_PARITY_EVEN	Even Parity bit

uart_interrupt_mode_t

Description: UART interrupt type, receiving or sending.

Define

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

Member

Member name	Description
UART_SEND	UART send
UART_RECEIVE	UART receive

uart_send_trigger_t

Description: Send interrupt or DMA trigger FIFO depth. When the data in the FIFO is less than or equal to this value, an interrupt or DMA transfer is triggered. The depth of the FIFO is 16 bytes.

Define

```
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;
```

Member

Member name	Description
UART_SEND_FIFO_0	FIFO be null
UART_SEND_FIFO_2	FIFO remaining 2 bytes
UART_SEND_FIFO_4	FIFO remaining 4 bytes
UART_SEND_FIFO_8	FIFO remaining 8 bytes

uart_receive_trigger_t

Description: Receive interrupt or DMA trigger FIFO depth. When the data in the FIFO is more than or equal to this value, an interrupt or DMA transfer is triggered. The depth of the FIFO is 16 bytes.

Define

```
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;
```

Member

Member name	Description
UART_RECEIVE_FIFO_1	FIFO remaining 1 byte
UART_RECEIVE_FIFO_4	FIFO remaining 2 byte
UART_RECEIVE_FIFO_8	FIFO remaining 4 byte
UART_RECEIVE_FIFO_14	FIFO remaining 8 byte

uart_data_t

Description: Data related parameters when using dma transmission.

Define

```
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;
```

Member

Member name	Description
tx_channel	The DMA channel number used for sending
rx_channel	The DMA channel number used for receiving

Member name	Description
tx_buf	Sent data
tx_len	The length of the sent data
rx_buf	Receive data
rx_len	Length of receiving data
transfer_mode	Transmission mode, send or receive

uart_interrupt_mode_t

Description: UART data transmission mode.

Define

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

Member

Member name	Description
UART_SEND	send
UART_RECEIVE	Receive

uart_work_mode_t

Description: UART work mode.

Define

```
typedef enum _uart_work_mode
{
    UART_NORMAL,
    UART_IRDA,
    UART_RS485_FULL_DUPLEX,
    UART_RS485_HALF_DUPLEX,
} uart_work_mode_t;
```

Member

Member name	Description
UART_NORMAL	Ordinary UART
UART_IRDA	IR
UART_RS485_FULL_DUPLEX	full duplex RS485
UART_RS485_HALF_DUPLEX	half-duplex RS485

uart_rs485_rede_t

Description: RS485 re de pins.

Define

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

Member

Member name	Description
UART_RS485_DE	Select de pin
UART_RS485_RE	Select re pin
UART_RS485_REDE	Select de and re pin

uart_polarity_t

Description: Polarity when re de is valid.

Define

```
typedef enum _uart_polarity
{
    UART_LOW,
    UART_HIGH,
} uart_polarity_t;
```

Member

Member name	Description
UART_LOW	Low level
UART_HIGH	High level

uart_tat_mode_t

Description: The mode of re de conversion.

Define

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

Member

Member name	Description
UART_DE_TO_RE	de valid to re valid
UART_RE_TO_DE	re valid to de valid
UART_TAT_ALL	de to re re to de

uart_det_mode_t

Description: de valid or invalid selection.

Define

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

Member

Member name	Description
UART_DE_ASSERTION	de valid
UART_DE_DE_ASSERTION	de invalid
UART_DE_ALL	de valid or invalid

The corresponding header file uarths.h

uarths_init

Initialize UARTHS, the system default baud rate is 115200, data bit is 8bit, stop bit is 1 bit, parity bits is none.

Because the uarths clock source is PLL0, we need to call this function to set the baud rate again after setting PLL0, otherwise garbled characters will be printed.

Function prototype: **void uarths_init(void)**

Parameter: No

Return value: No

uarths_config

Description: Set the parameters of UARTHS. data bit is 8bit, parity bits is none.

Function prototype: **void uarths_config(uint32_t baud_rate, uarths_stopbit_t stopbit)**

Parameter:

Parameter Name	Description	Input/output
baud_rate	Baud rate	Input
stopbit	Stop bit	Input

Return value: No

uarths_receive_data

Description: Read data by UARTHS

Function prototype: **size_t uarths_receive_data(uint8_t *buf, size_t buf_len)**

Parameter:

Parameter Name	Description	Input/output
buf	Receive data	Output
buf_len	The length of the received data	Input

Return value: The length of data received.

uarths_send_data

Description: Sending data through UART.

Function prototype: **size_t uarths_send_data(const uint8_t *buf, size_t buf_len)**

Parameter:

Parameter Name	Description	Input/output
buf	Waiting to send data	Input
buf_len	Length of waiting to send data	Input

Return value: The length of the sent data.

uarths_set_irq

Description: Set the UARTHS interrupt callback function.

Function prototype: **void uarths_set_irq(uarths_interrupt_mode_t interrupt_mode, plic_irq_callback_t uarths_callback, void *ctx, uint32_t priority)**

Parameter:

Parameter Name	Description	Input/output
interrupt_mode	interrupt type	Input
uarths_callback	interrupt callback function	Input
ctx	Parameter of interrupt callback function	Input
priority	interrupt priority level	Input

Return value: No

uarths_get_interrupt_mode

Description: Get the interrupt type of UARTHS. Receiving, sending, or receiving and sending are interrupted at the same time.

Function prototype: **uarths_interrupt_mode_t uarths_get_interrupt_mode(void)**

Parameter: No

Return value: The current type of interrupt.

uarths_set_interrupt_cnt

Description: Set the FIFO depth when UARTHS is interrupted. When the interrupt type is UARTHS_SEND_RECEIVE, the transmit and receive FIFO interrupt depths are both cnt;

Function prototype: **void uarths_set_interrupt_cnt(uarths_interrupt_mode_t interrupt_mode, uint8_t cnt)**

Parameter:

Parameter Name	Description	Input/output
interrupt_mode	interrupt type	Input
cnt	FIF depth	Input

Return value: No

Eg:

```
/*Set the receive interrupt, interrupt FIFO depth to 0, that is, immediately interrupt the received data and read the received data.*/
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();
```

Data type

Relevant data types and data structures are defined as follows:

- uarths_interrupt_mode_t: interrupt type
- uarths_stopbit_t: stop bit.

uarths_interrupt_mode_t

Description: UARTHS interrupt type

Define

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

Member

Member name	Description
UARTHS_SEND	Send interrupt
UARTHS_RECEIVE	Receive interrupt
UARTHS_SEND_RECEIVE	Send and receive interrupt

uarths_stopbit_t

Description: UARTHS stop bit.

Define

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

Member

Member name	Description
UART_STOP_1	1bit stop bit
UART_STOP_2	2bit stop bit