3.9 DMAC Memory access controller DMAC

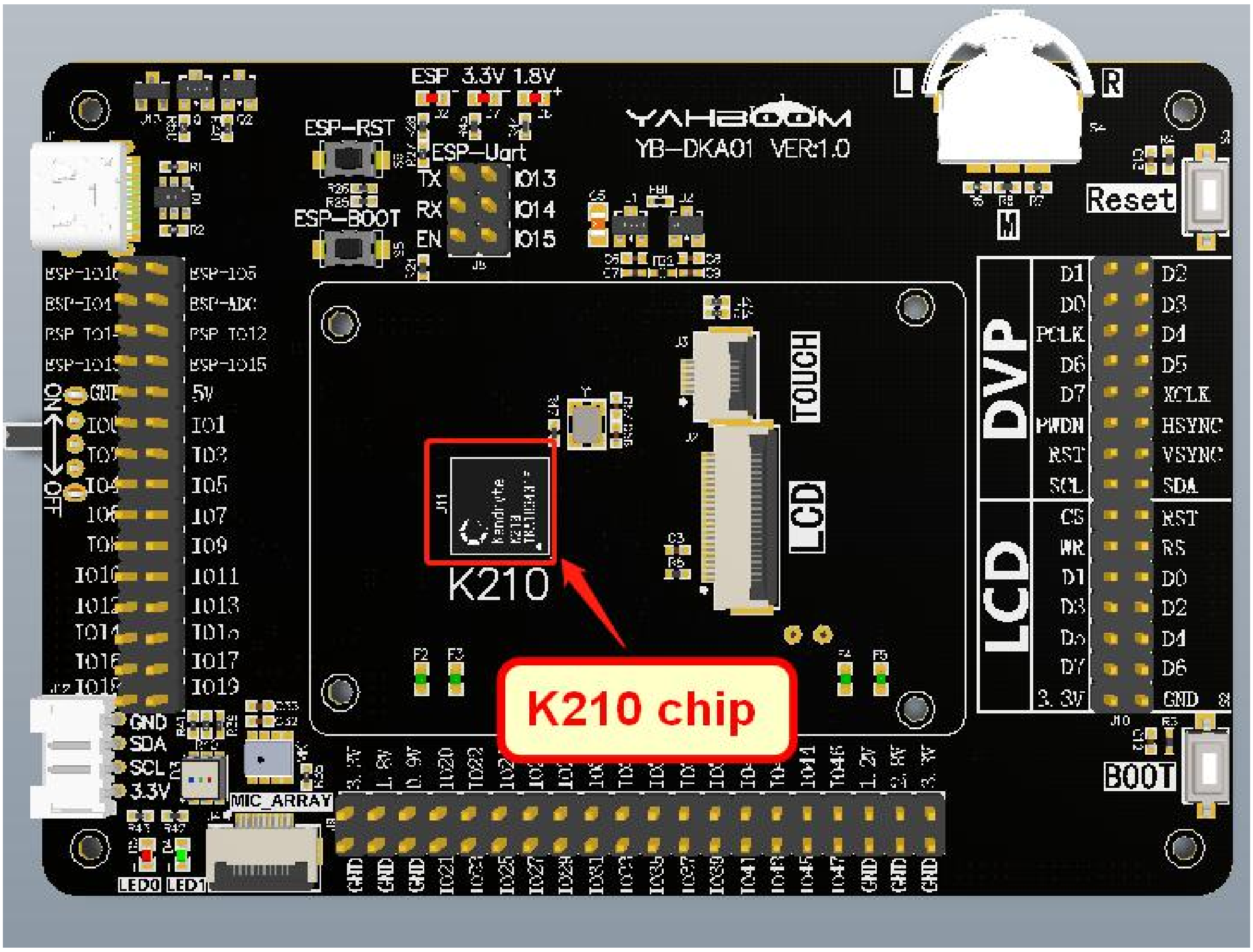
1. Experiment purpose

In this lesson, we mainly learns the K210 direct memory access controller DMAC.

2.Experiment preparation

2.1 components

Direct memory access controller DMAC function of K210 chip



2.2 Component characteristics

Direct Memory Access (DMA) is used to provide high-speed data transfer between peripherals and memory, between memory and memory.

User can be moved quickly data through DMA without any CPU operation, thereby improving the efficiency of the CPU.

The DMA module possess the following functions.

- Automatically select an idle DMA channel for transmission.
- Automatically select software or hardware handshake protocol according to source address and destination address
- Support element sizes of 1, 2, 4, and 8 bytes, the source and target sizes do not have to be the same
- Asynchronous or synchronous transmission function

- Loop transmission function, often used in scenes such as refreshing the screen or audio recording and playback

2.3 SDK API function
The header file is dmac.h
• dmac_init: Initialize DMA
• dmac_set_single_mode: Set single-channel DMA parameters
• dmac_is_done: It is used to judge whether the transfer is completed after DMAC is started.
• dmac_wait_done: Wait DMA complete work.
• dmac_set_irq: Set the callback function of DMAC interrupt
• dmac_set_src_dest_length: Set the source address, destination address and length of the DMAC. Then, start the DMAC transfer. If src is NULL, the source address is not set, dest is NULL, the destination address is not set, and len<=0, the length is not set.
This function is generally used in DMAC interrupts to make DMA continue to transmit data without having to set all DMAC parameters again to save time.
• dmac_is_idle: Determine whether the current DMAC channel is idle. This function can be used to determine the DMAC status before and after transmission.
• dmac_wait_idle: Wait for the DMAC to enter the idle state.

3. Experimental principle
DMA transfer is to copy data from one address space to another address space.
A complete DMA transmission process must go through the 4 steps.
DMA request -> DMA response -> DMA transmission -> DMA end.

4. Experiment procedure
4.1 According to the above hardware connection pin diagram, K210 hardware pins and software functions use FPIOA mapping relationship.

```
/*****************************HARDWARE-PIN*********************************/
//Hardware IO port, corresponding Schematic
#define PIN_RGB_R             (6)
#define PIN_RGB_G             (7)
#define PIN_RGB_B             (8)

#define PIN_UART_USB_RX       (4)
#define PIN_UART_USB_TX       (5)

/*****************************SOFTWARE-GPIO********************************/
//Software GPIO port, corresponding program
#define RGB_R_GPIONUM          (0)
#define RGB_G_GPIONUM          (1)
#define RGB_B_GPIONUM          (2)

#define UART_USB_NUM           UART_DEVICE_3

/*****************************FUNC-GPIO************************************/
//Function of GPIO port, bound to hardware IO port
#define FUNC_RGB_R             (FUNC_GPIOHS0 + RGB_R_GPIONUM)
#define FUNC_RGB_G             (FUNC_GPIOHS0 + RGB_G_GPIONUM)
#define FUNC_RGB_B             (FUNC_GPIOHS0 + RGB_B_GPIONUM)

#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_RGB_R, FUNC_RGB_R);
    fpioa_set_function(PIN_RGB_G, FUNC_RGB_G);
    fpioa_set_function(PIN_RGB_B, FUNC_RGB_B);

    fpioa_set_function(PIN_UART_USB_RX, FUNC_UART_USB_RX);
    fpioa_set_function(PIN_UART_USB_TX, FUNC_UART_USB_TX);
}
```

4.2 It needs to be initialized before using RGB light, that is, set the software GPIO of RGB light to output mode.

```
void init_rgb(void)
{
    /* Set the GPIO mode of the RGB light to output */
    gpiohs_set_drive_mode(RGB_R_GPIONUM, GPIO_DM_OUTPUT);
    gpiohs_set_drive_mode(RGB_G_GPIONUM, GPIO_DM_OUTPUT);
    gpiohs_set_drive_mode(RGB_B_GPIONUM, GPIO_DM_OUTPUT);

    /* Close RGB */
    rgb_all_off();
}
```

4.3 Set the GPIO of the RGB light to high to make the RGB light go out.

```
void rgb_all_off(void)
{
    gpiohs_set_pin(RGB_R_GPIONUM, GPIO_PV_HIGH);
    gpiohs_set_pin(RGB_G_GPIONUM, GPIO_PV_HIGH);
    gpiohs_set_pin(RGB_B_GPIONUM, GPIO_PV_HIGH);
}
```

4.4 Initialize serial port 3, and set the serial port baud rate to 115200.
Then, send "hello yahboom!" by dma channel 0.

```
/* Initialize the serial port and 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);

/* Send "hello yahboom!"" when booting */
char *hel = {"hello yahboom!\n"};
uart_send_data_dma(UART_USB_NUM, DMAC_CHANNEL0, (uint8_t *)hel, strlen(hel));
```

4.5 Next, dma channel 1 to read the data of the serial port, and it will through the process of protocol filtering.
By setting the value of rec_flag, Ensure to parse the data starting with hexadecimal FFAA
After parsing is complete, the real data received will be sent out through dma channel 0.

```c
while (1)
{
    /* Receive serial port data through DMA channel 1 and save it in recv*/
    uart_receive_data_dma(UART_USB_NUM, DMAC_CHANNEL1, &recv, 1);
    /* Judgment agreement, it must be the data at the beginning of FFAA */
    switch(rec_flag)
    {
    case 0:
        if(recv == 0xFF)
            rec_flag = 1;
        break;
    case 1:
        if(recv == 0xAA)
            rec_flag = 2;
        else if(recv != 0xFF)
            rec_flag = 0;
        break;
    case 2:
        /* Parse the real data */
        parse_cmd(&recv);
        /* Send serial data through DMA channel 0 */
        uart_send_data_dma(UART_USB_NUM, DMAC_CHANNEL0, &recv, 1);
        rec_flag = 0;
        break;
    }
}
```

4.6 The following is the source code of the serial port to send and receive data through DMA. The SDK already contains these two functions. We can call them directly.

```c
void uart_receive_data_dma(uart_device_number_t uart_channel, dmac_channel_number_t dmac_channel, uint8_t *buffer, size_t buf_len)
{
#if FIX_CACHE
    uint32_t *v_recv_buf = (uint32_t *)iomem_malloc(buf_len * sizeof(uint32_t));
#else
    uint32_t *v_recv_buf = (uint32_t *)malloc(buf_len * sizeof(uint32_t));
#endif
    configASSERT(v_recv_buf != NULL);

    sysctl_dma_select((sysctl_dma_channel_t)dmac_channel, SYSCTL_DMA_SELECT_UART1_RX_REQ + uart_channel * 2);

    dmac_set_single_mode(dmac_channel, (void *)(&uart[uart_channel]->RBR), v_recv_buf, DMAC_ADDR_NOCHANGE, DMAC_ADDR_INCREMENT,
                         DMAC_MSIZE_1, DMAC_TRANS_WIDTH_32, buf_len);

    dmac_wait_done(dmac_channel);
    for(uint32_t i = 0; i < buf_len; i++)
    {
        buffer[i] = (uint8_t)(v_recv_buf[i] & 0xff);
    }
#if FIX_CACHE
    iomem_free(v_recv_buf);
#else
    free(v_recv_buf);
#endif
}
```

```c
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)
{
#if FIX_CACHE
    uint32_t *v_send_buf = iomem_malloc(buf_len * sizeof(uint32_t));
#else
    uint32_t *v_send_buf = malloc(buf_len * sizeof(uint32_t));
#endif
    configASSERT(v_send_buf != NULL);

    for(uint32_t i = 0; i < buf_len; i++)
        v_send_buf[i] = buffer[i];

    sysctl_dma_select((sysctl_dma_channel_t)dmac_channel, SYSCTL_DMA_SELECT_UART1_TX_REQ + uart_channel * 2);
    dmac_set_single_mode(dmac_channel, v_send_buf, (void *)(&uart[uart_channel]->THR), DMAC_ADDR_INCREMENT, DMAC_ADDR_NOCHANGE,
                         DMAC_MSIZE_1, DMAC_TRANS_WIDTH_32, buf_len);

    dmac_wait_done(dmac_channel);
#if FIX_CACHE
    iomem_free((void *)v_send_buf);
#else
    free((void *)v_send_buf);
#endif
}
```

4.7 Analyzing the real data.

We can modify the color and state of the RGB lights according to the different values of the data.

If it is 0x11, the red light is on, 0x22 the red light is off;

If it is 0x33, the green light is on, 0x44 the green light is off;

If it is 0x55, the blue light is on, 0x66 the blue light is off;

```c
#define CMD_RGB_R_ON            0x11
#define CMD_RGB_R_OFF           0x22
#define CMD_RGB_G_ON            0x33
#define CMD_RGB_G_OFF           0x44
#define CMD_RGB_B_ON            0x55
#define CMD_RGB_B_OFF           0x66
```

```c
int parse_cmd(uint8_t *cmd)
{
    switch(*cmd)
    {
    case CMD_RGB_R_ON:
        /* red light on*/
        gpiohs_set_pin(RGB_R_GPIONUM, GPIO_PV_LOW);
        break;
    case CMD_RGB_R_OFF:
        /* red light off*/
        gpiohs_set_pin(RGB_R_GPIONUM, GPIO_PV_HIGH);
        break;
    case CMD_RGB_G_ON:
        /* green light on*/
        gpiohs_set_pin(RGB_G_GPIONUM, GPIO_PV_LOW);
        break;
    case CMD_RGB_G_OFF:
        /* green light off*/
        gpiohs_set_pin(RGB_G_GPIONUM, GPIO_PV_HIGH);
        break;
    case CMD_RGB_B_ON:
        /* blue light on*/
        gpiohs_set_pin(RGB_B_GPIONUM, GPIO_PV_LOW);
        break;
    case CMD_RGB_B_OFF:
        /* blue light off*/
        gpiohs_set_pin(RGB_B_GPIONUM, GPIO_PV_HIGH);
        break;
    }
    return 0;
}
```

4.8 Compile and debug, burn and run

Copy the dmac to the src directory in the SDK.

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

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

make

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

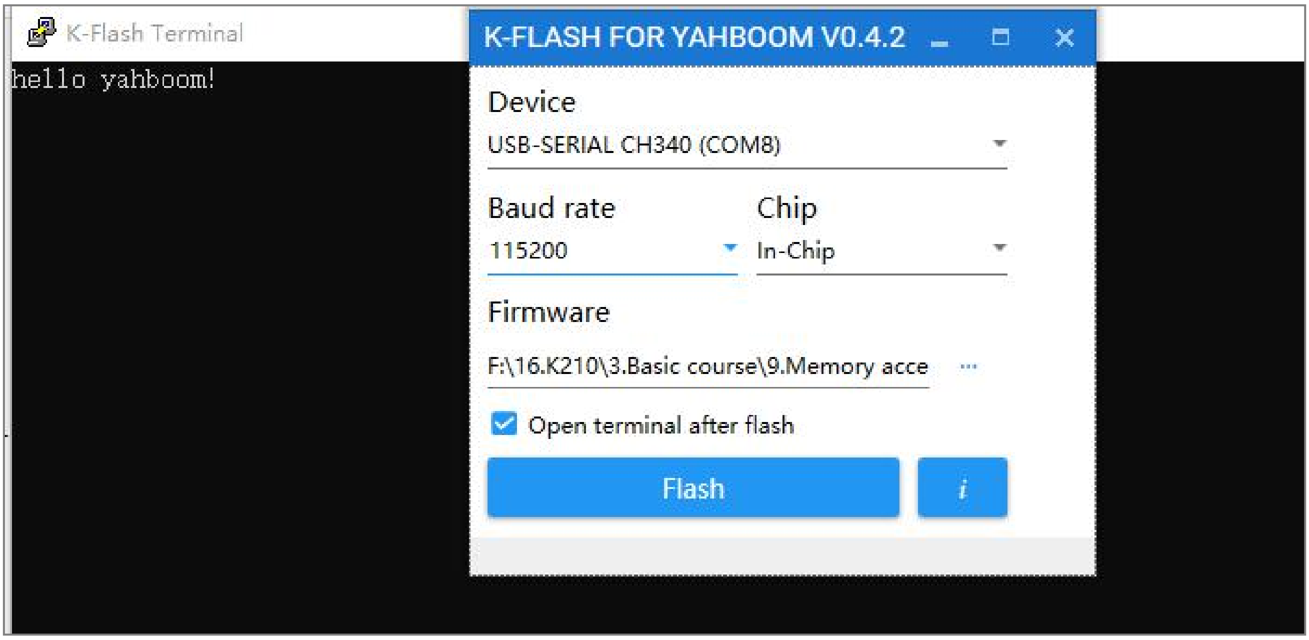
After the compilation is complete, the dmac.bin file will be generated in the build folder.

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 button.bin file to the K210 development board.

5. Experimental phenomenon

After the firmware is write, a terminal interface will pop up. As shown below.



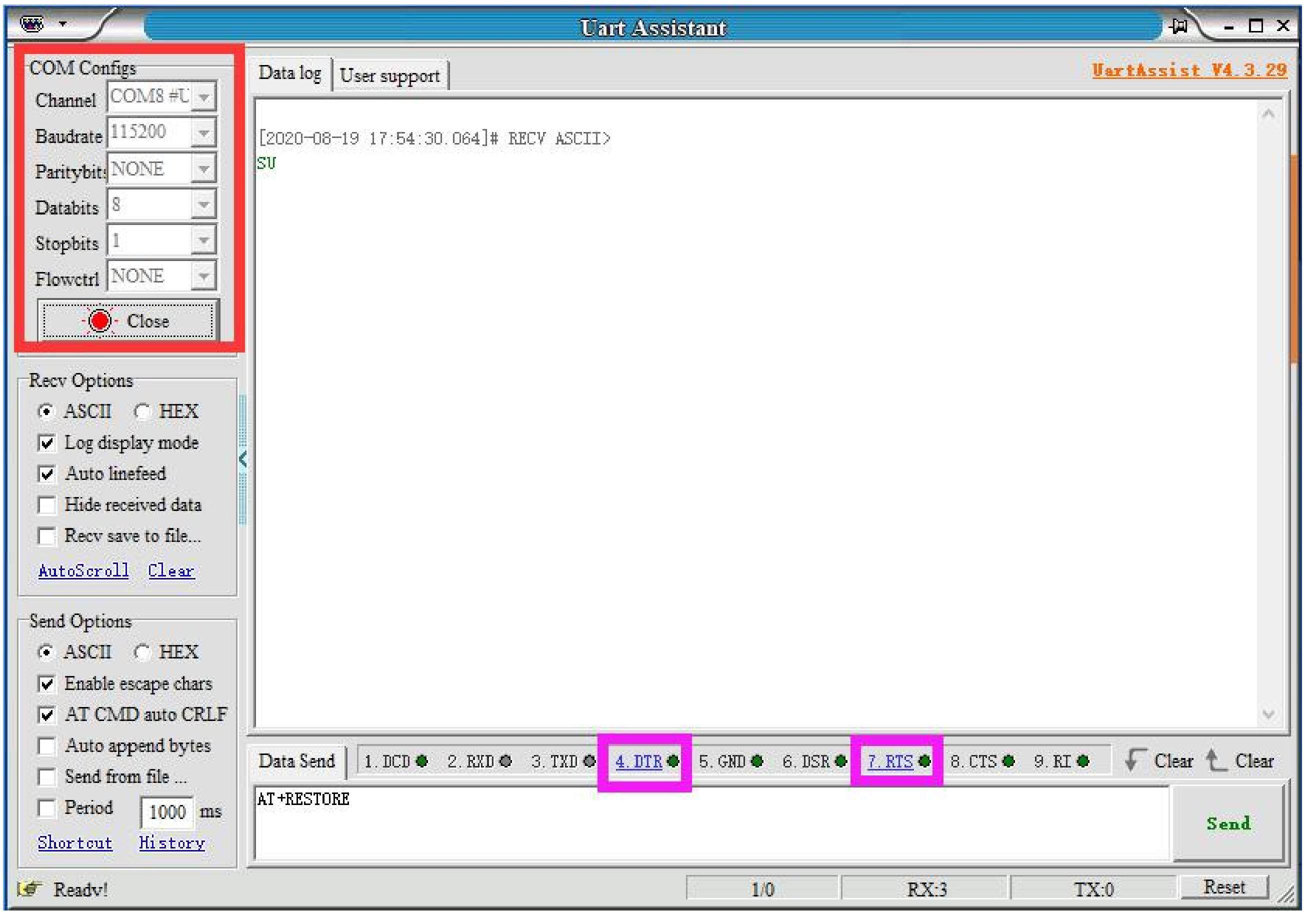
Close this terminal interface. Open the serial port assistant of the computer, select the corresponding serial port number of the corresponding K210 development board, set the baud rate to 115200.

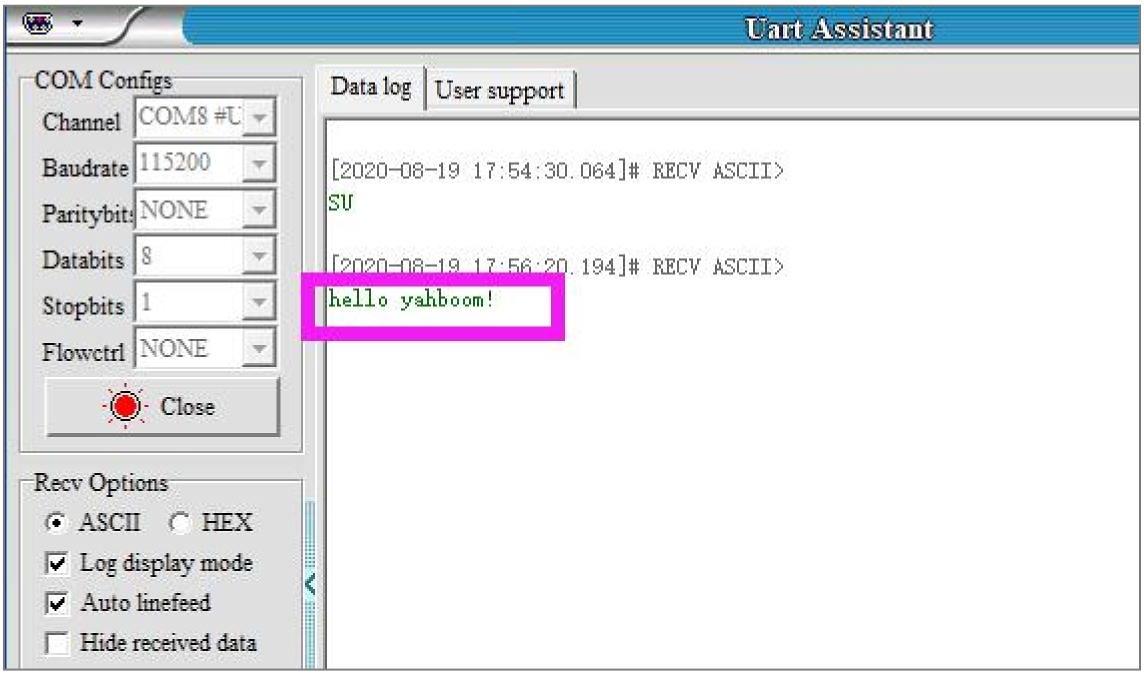
Then, click to open the serial port assistant.

!Note: you also need to set the DTR and RTS of the serial port assistant.

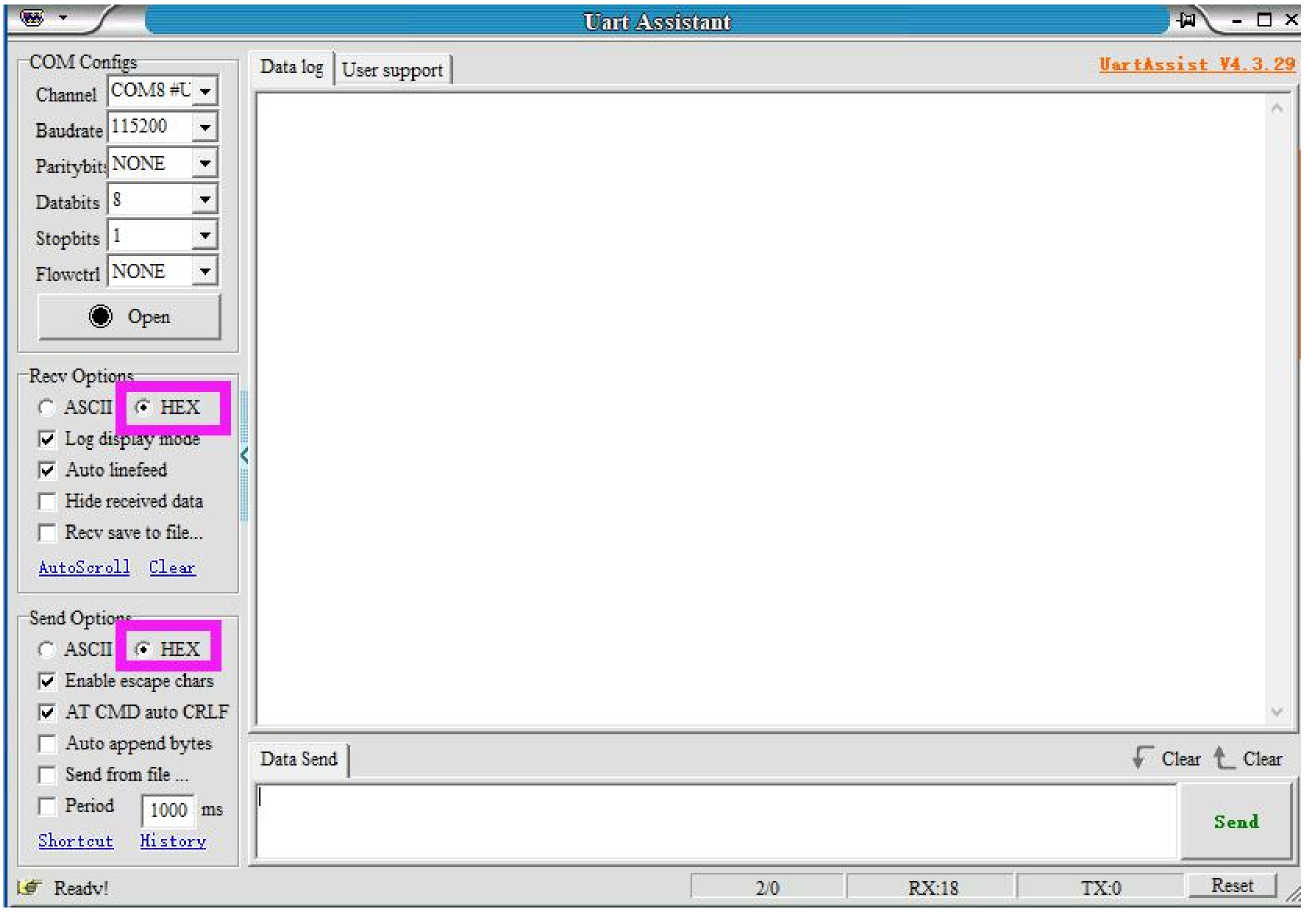
Click 4.DTR and 7.RTS to set them to green.

Press the reset button of the K210 development board, “hello yahboom!” will be printed.

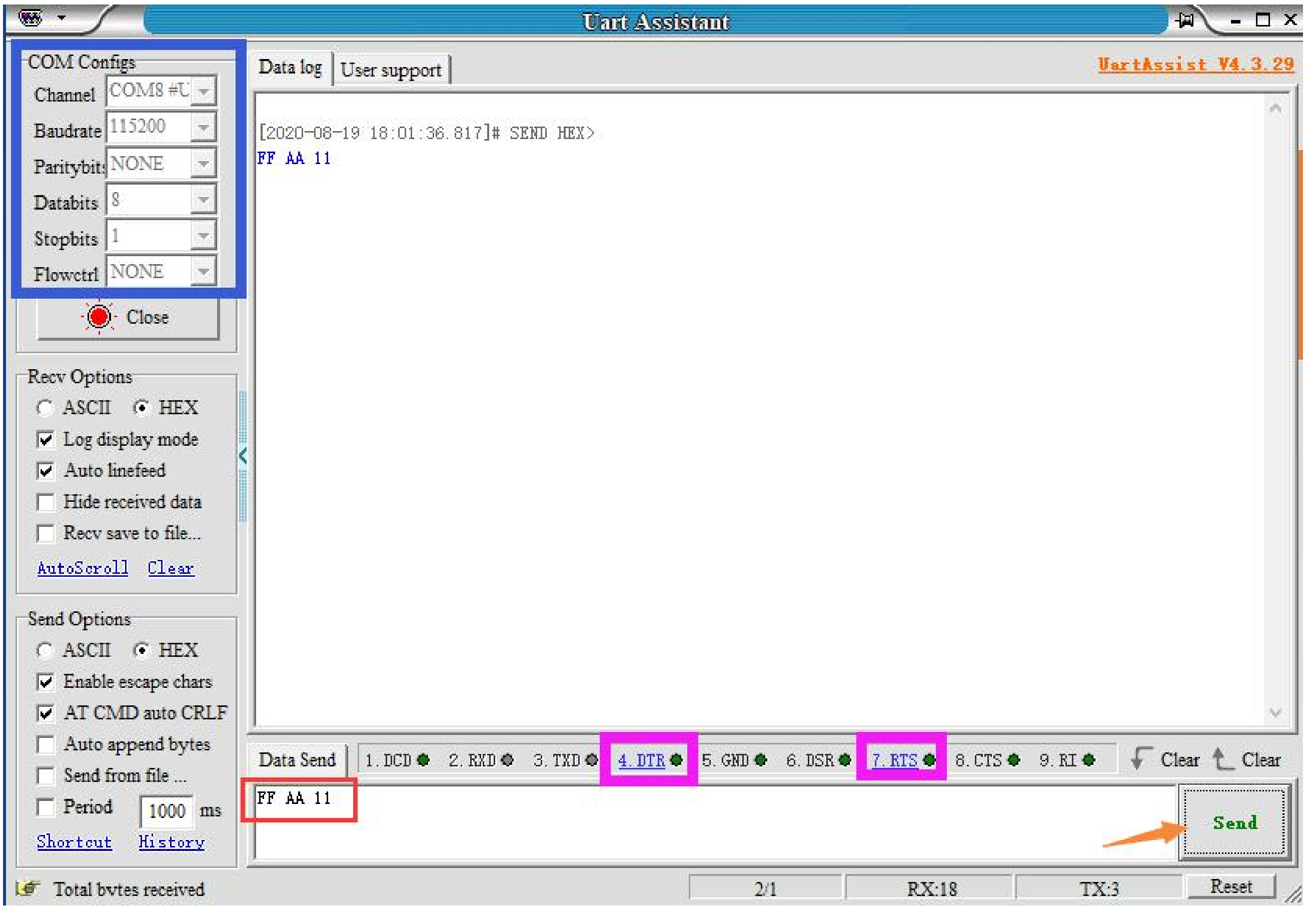




Due to the uint8_t type data is used in the communication, we need to change the sending and receiving settings of the serial port assistant to HEX.
As shown below.



Send FF AA 11 in the serial assistant, red light is on; FF AA 22, red light is off.
Send FF AA 33 in the serial assistant, green light is on; FF AA 44, green light is off;
Send FF AA 55 in the serial assistant, blue light is on; FF AA 66, blue light is off .
When click “send” , we can see the effect.



6. Experiment summary

6.1 Direct memory access controller DMAC needs to be used with other equipment, such as serial port, I2C or I2S communication.

6.2 DMAC can improve the efficiency of the CPU, and directly transfer data between the device and the memory through DMA. CPU only needs to start the dma to transfer.

Appendix -- API

Header file is dmac.h

dmac_init

Description: Initialize DMA.

Function prototype: **void dmac_init(void)**
Parameter: No
Return value: No

dmac_set_single_mode
Description: Set the single-channel DMA parameters.
Function prototype:
void dmac_set_single_mode(dmac_channel_number_t channel_num, const void *src, void *dest, dmac_address_increment_t src_inc, dmac_address_increment_t dest_inc, dmac_burst_trans_length_t dmac_burst_size, dmac_transfer_width_t dmac_trans_width, size_t block_size)
Parameter:

Parameter name	Description	Input/Output
channel_num	DMA Channel number	Input
src	Source address	Input
dest	Target address	Output
src_inc	Whether the source address is incremented	Input
dest_inc	Whether the target address is incremented	Input
dmac_burst_size	Number of bursts	Input
dmac_trans_width	Single transmission data bit width	Input
block_size	Number of transmitted data	Input

Return value: No

dmac_is_done
Description: It is used to judge whether the transfer is completed after the DMAC is started.
Function prototype: **int dmac_is_done(dmac_channel_number_t channel_num)**
Parameter:

Parameter name	Description	Input/Output
channel_num	DMA Channel number	Input

Return value:

Return value	Description
0	Undone
1	Done

dmac_wait_done
Description: Wait for the DMA to finish its work.
Function prototype: **void dmac_wait_done(dmac_channel_number_t channel_num)**
Parameter:

Parameter name	Description	Input/Output
channel_num	DMA Channel number	Input

Return value: No

dmac_set_irq

Description: Set the callback function of DMAC interrupt

Function prototype: **void dmac_set_irq(dmac_channel_number_t channel_num , plic_irq_callback_t dmac_callback, void *ctx, uint32_t priority)**

Parameter:

Parameter name	Description	Input/Output
channel_num	DMA Channel number	Input
dmac_callback	Interrupt callback function	Input
ctx	Callback function parameter	Input
priority	Interrupt priority	Input

Return value: No

dmac_set_src_dest_length

Description: Set the source address, destination address and length of the DMAC. Then start the DMAC transfer.

If src is NULL, the source address is not set. If dest is NULL, the destination address is not set. If the length is not set if len<=0.

This function is generally used in the DMAC interrupt to make the DMA continue to transmit data.

Function prototype: **void dmac_set_src_dest_length(dmac_channel_number_t channel_num, const void *src, void *dest, size_t len)**

Parameter:

Parameter name	Description	Input/Output
channel_num	DMA Channel number	Input
src	Interrupt callback function	Input
dest	Callback function parameter	Input
len	Transmission length	Input

Return value: No

dmac_is_idle

Description: Determine whether the current channel of the DMAC is idle. This function can be used to determine the DMAC status before and after the transfer.

Function prototype: **int dmac_is_idle(dmac_channel_number_t channel_num)**

Parameter:

Parameter name	Description	Input/Output
channel_num	DMA Channel number	Input

Return value:

Return value	Description
0	Busy
1	Idle

dmac_wait_idle

Description: Wait for the DMAC to enter the idle state.

Parameter:

Parameter name	Description	Input/Output
channel_num	DMA Channel number	Input

Return value: No

Eg:

```
/* I2C sends 128 int data through DMA */
uint32_t buf[128];
dmac_wait_idle(SYSCTL_DMA_CHANNEL_0);
sysctl_dma_select(SYSCTL_DMA_CHANNEL_0, SYSCTL_DMA_SELECT_I2C0_TX_REQ);
dmac_set_single_mode(SYSCTL_DMA_CHANNEL_0, buf, (void*)(&i2c_adapter->data_cmd),
DMAC_ADDR_INCREMENT, DMAC_ADDR_NOCHANGE, DMAC_MSIZE_4,
DMAC_TRANS_WIDTH_32, 128);
dmac_wait_done(SYSCTL_DMA_CHANNEL_0);
```

Data type

- dmac_channel_number_t: DMA Channel number
- dmac_address_increment_t: Address growth method
- dmac_burst_trans_length_t: Number of burst transfers
- dmac_transfer_width_t: Number of data bits in a single transmission

dmac_channel_number_t

Description: DMA Channel number

Define

```
typedef enum _dmac_channel_number
{
    DMAC_CHANNEL0 = 0,
    DMAC_CHANNEL1 = 1,
    DMAC_CHANNEL2 = 2,
    DMAC_CHANNEL3 = 3,
    DMAC_CHANNEL4 = 4,
```

```
    DMAC_CHANNEL5 = 5,
    DMAC_CHANNEL_MAX
} dmac_channel_number_t;
```

member

Member name	Description
DMAC_CHANNEL0	DMA channel 0
DMAC_CHANNEL1	DMA channel 1
DMAC_CHANNEL2	DMA channel 2
DMAC_CHANNEL3	DMA channel 3
DMAC_CHANNEL4	DMA channel 4
DMAC_CHANNEL5	DMA channel 5

dmac_address_increment_t

Description: Address growth method

Define

```
typedef enum _dmac_address_increment
{
    DMAC_ADDR_INCREMENT = 0x0,
    DMAC_ADDR_NOCHANGE   = 0x1
} dmac_address_increment_t;
```

member

Member name	Description
DMAC_ADDR_INCREMENT	Automatic address growth
DMAC_ADDR_NOCHANGE	Address unchanged

dmac_burst_trans_length_t

Description: Number of burst transfers

Define

```
typedef enum _dmac_burst_trans_length
{
    DMAC_MSIZE_1     = 0x0,
    DMAC_MSIZE_4     = 0x1,
    DMAC_MSIZE_8     = 0x2,
    DMAC_MSIZE_16    = 0x3,
    DMAC_MSIZE_32    = 0x4,
    DMAC_MSIZE_64    = 0x5,
    DMAC_MSIZE_128 = 0x6,
    DMAC_MSIZE_256 = 0x7
} dmac_burst_trans_length_t;
```

member

Member name	**Description**
DMAC_MSIZE_1	Multiply the number of single transfers by 1
DMAC_MSIZE_4	Multiply the number of single transfers by 4
DMAC_MSIZE_8	Multiply the number of single transfers by 8
DMAC_MSIZE_16	Multiply the number of single transfers by 16
DMAC_MSIZE_32	Multiply the number of single transfers by 32
DMAC_MSIZE_64	Multiply the number of single transfers by 64
DMAC_MSIZE_128	Multiply the number of single transfers by 128
DMAC_MSIZE_256	Multiply the number of single transfers by 256

dmac_transfer_width_t

Description: Number of data bits in a single transmission

Define

```
typedef enum _dmac_transfer_width
{
    DMAC_TRANS_WIDTH_8    = 0x0,
    DMAC_TRANS_WIDTH_16   = 0x1,
    DMAC_TRANS_WIDTH_32   = 0x2,
    DMAC_TRANS_WIDTH_64   = 0x3,
    DMAC_TRANS_WIDTH_128 = 0x4,
    DMAC_TRANS_WIDTH_256 = 0x5
} dmac_transfer_width_t;
```

member

Member name	**Description**
DMAC_TRANS_WIDTH_8	Multiply the number of single transfers by 8
DMAC_TRANS_WIDTH_16	Multiply the number of single transfers by 16
DMAC_TRANS_WIDTH_32	Multiply the number of single transfers by 32
DMAC_TRANS_WIDTH_64	Multiply the number of single transfers by 64
DMAC_TRANS_WIDTH_128	Multiply the number of single transfers by 128
DMAC_TRANS_WIDTH_256	Multiply the number of single transfers by 256