AB32VG1 and RT-Thread tutorials

Table of Contents

Building an application and using the development environment....3
1. Introduction to the AB32VG1 development board....3
2. RT-Thread Studio installation and package management....3
2.1. RT-Thread Studio installation....4
2.2. RT-Thread Studio package management....7
3. Create a project using RT-Thread Studio....8
3.1. New project....8
3.2. Configuration items....10
3.3. Compile RT-Thread firmware....11
3.4. Download RT-Thread firmware....12
3.5. Final result....13
Create magical colours with RGB LED....15
1. Introduction to RGB LED....15
2. Simple realization of seven colours....15
2.1. Experimental analysis....15
2.2. Code implementation....15
2.3. Result....17
3. PWM control full color LED....17
3.1. Experimental analysis....17
2.3. Code implementation....18
3.3. Result....22
Use of serial port devices....23
1. Simple use of serial devices....23
1.1. Sending and receiving on the serial port....23
1.2. Result....25
2. serial device optimization....27
Watchdog....30
1. Working principle of watchdog....30
2. Configure watchdog....30
3. Watchdog code implementation....30
Feed the (watch)dog with buttons....32
1. Introduction to buttons....32
2. Key to feed the dog – Implementation....33
3. Result....35
Music player....36
2.2. Storage file system configuration....37
3. Implementation of music player....38
3.1. Brief analysis of WavPlay playing audio....38
3.2. PWM control RGB lights....38
3.3. Serial port control audio equipment....40
3.4. Buttons to control audio equipment....43
4. Function demonstration....46
5. Summary....46
I2C OLED display with SSD1306 controller....47
BLE development....52
Install a recent SDK....52
Build an RT-Thread project....52
Configure the project....53
Modify the code....54
Modify mac_addr....54

Modify BLE_name....55
Modify service and device information....55
Modify service properties....56
Compile....56
Debugging....58
Scan for BLE devices....60
List of device-related data services....60
Get BLE device data – Manufacturer information....61
Obtain BLE device data – Device model information....63
Read heart rate related data....64
Automatic data notification reporting of heart rate data....65
Remaining issues....67
Summary....67

Building an application and using the development environment

Original post: https://club.rt-thread.org/ask/article/8a607410ffec3de8.html
Copyright: BruceOu License: CC-BY-SA 4.0

1. Introduction to the AB32VG1 development board

The AB32VG1 development board is based on Bluetrum's AB5301A MCU.

Onboard resources:

- CPU: AB5301A; (LQFP48 package, 120MHz, 192KB RAM, 8 Mbit flash, ADC, PWM, USB, UART, I2C and other resources)
- Equipped with Bluetooth module, FM module, one TF Card interface, one USB interface, one I2C interface, one audio interface (CTIA American standard), six ADC input pins, six PWM output pins, an RGB LED, one power indicator, three programming indicators, one infrared remote-control receiver, one Reset button, three user buttons
- Board size: 6cm*9cm
- The 2.54mm pitch I/O connectors are compatible with the Arduino Uno expansion interface.

2. RT-Thread Studio installation and package management

RT-Thread Studio mainly includes project creation and management, code editing, SDK management, RT-Thread configuration, build configuration, debug configuration, program download and debugging and other functions, combined with graphical configuration system and software package and component resources, reducing repetitive work, improving development efficiency.

Download link: https://www.rt-thread.io/download.html?download=Studio

2.1. RT-Thread Studio installation

After the download is complete, the next step is to install the software. Double-click the .exe file of the installation package, the installation interface is as shown below:

Click [Next] to start the installation.

Check 'I agree to this agreement', and then click [Next].

Note: Do not include spaces and Chinese characters when specifying the installation path.

Specify the start menu folder name, the default is fine, and then click [Next].

Click [Install] to start the installation.

Just wait for the installation to complete. After the installation is complete, click [OK] to start RT-Thread Studio, as shown below.

Or cancel the check of Run RT-Thread Studio, click Finish, and start RT-Thread Studio from the desktop shortcut.

The first time you start RT-Thread Studio, you need to log in with an account. After logging in once, the account will be automatically remembered, and you don’t need to log in again later. Login supports third-party account login.

After successful login, a welcome interface will pop up, and you can learn how to use RT-Thread Studio through the link below.

2.2. RT-Thread Studio package management

RT-Thread Studio is installed, and then install some dependent packages, enter 'SDK' in the search bar, and click to enter the RT-Thread Studio SDK manager.

First, let's upgrade RTT first, if 'latest' is not installed, just install it directly.

This article uses Bluetrum's AB32VG1 development board, so it is enough to install the latest board support package.

It has already been installed here. If it is not installed, check a version and click [Install Resource Pack].

Then install the cross-compilation tool. The RISC-V toolchain used by AB5301A is not installed by default.

3. Create a project using RT-Thread Studio

3.1. New project

Right-click in the project [Explorer] window and select the new sub-menu item, as shown in the figure below:

Note: Of course, there are many ways to create a new project, and you can also create a new project through [File].

In the pop-up New Project Wizard dialog box, select the RT-Thread project type, and then click [Next].

Fill in the project name, select the RT-Thread source code version, select the corresponding BSP, and click the [Finish] button. That's it.

Wait for the creation to complete. You can see some basic information about the board.

After the project is successfully created, the newly created project will appear in the project explorer window.

3.2. Configuration items

Double-click the RT-Thread Settings file to open the RT-Thread project configuration interface. The configuration interface displays the software package and the architecture configuration interface of the component and service layer by default.

The gray ones in the above figure are not loaded into the project. If you need any components and drivers, right click on the interface to load.

After the configuration is complete, save the configuration and exit the interface.

The description of the main directories and files of the project framework is shown in the following table:

File / Directory	Description
Includes	Header files
applications	User application code
drivers or board	Drivers / BSP provided by RT-Thread
Libraries	Firmware library downloaded from the MCU's official website
rt-thread	RT-Thread source code
Kconfig	Configuration file for menuconfig
README.md	BSP documentation
rtconfig.h	BSP configuration header file

3.3. Compile RT-Thread firmware

The next step is to compile the project and generate the object code. Before compiling, configure the cross-compilation tool. Right-click the project, click Properties, select 'MCU', and set the cross-compilation tool.

Cross-compilation tool path:

D:\RT-ThreadStudio\repo\Extract\ToolChain_Support_Packages\RISC-V\RISC-V-GCC\10.1.0\bin

Click the Build button on the toolbar to compile the project.

The compiled process log is printed on the console, and the compilation is completed, as shown below.

Note: If the cross-compilation tool is not configured, the following errors will occur during compilation.

3.4. Download RT-Thread firmware

Flashing tool address: https://github.com/BLUETRUM/Downloader

The development board is programmed through the Type-C interface, and the programming adopts single-wire programming (1 wire program&debug); when programming, a jumper cap needs to be used to connect the VBAT and VIN of J5 and J1, where J5 is connected to the chip to be programmed. The two pins of J1 supply power to the programming terminal.

Bluetrum's development board cannot use the tools that come with RTT, you need to use the Bluetrum's flashing tool. Open the Downloader tool and select the serial port.

Next select the [.dcf] file,

Then click the [Download] option in the 'Start' menu, and then click [Start].

After the download is complete, the RT-Thread command prompt is displayed:

3.5. Final result

Reset after programming successfully. You can see the LED turn on and off.

Please note Bluetrum's development board cannot use putty as a console, and needs to use Bluetrum's tools. Just activate the 'Developer Mode' of Downloader.

Create magical colours with RGB LED

Original post: https://club.rt-thread.org/ask/article/29ca7d9323fbe519.html
Copyright: BruceOu License: CC-BY-SA 4.0

1. Introduction to RGB LED

Everyone's usual LEDs are monochromatic, and you can turn the LED on and off by controlling the high and low levels at one end. RGB LEDs are LEDs that can combine multiple colours through RGB three basic LEDs. There are also RGBW LED with four-primary colours. RGB LEDs are divided into two types: common anode and common cathode. Bluetrum's AB32VG1 has a common anode RGB LED.

There are two ways to control an RGB LED: one is to directly control through three pins, and the other is to control through driver chips such as SM16703, WS2811, and TM1829. This article adopts the direct control method, but in order to obtain more colours, it needs to be controlled by PWM.

2. Simple realization of seven colours

2.1. Experimental analysis

Schematic diagram

Jumpers

The cathodes of these LED lights are connected to the GPIO pins of AB5301A, as long as we control the level output state of the GPIO pins, we can control the LED lights on and off.

2.2. Code implementation

Regarding project creation, I won't talk about it here. If you don't understand, go to the previous chapter.

Development environment setup and use (RT-Thread Studio)

The seven colours are very simple, and the colours are combined by controlling the three pins. Look directly at the code.

```
/**Includes**********************************************************************/
#include <rtthread.h>
#include "board.h"

/**define************************************************************************/
#define LED_R "PE.1"
#define LED_G "PE.4"
#define LED_B "PA.2"

/** the macro definition to the led on or off
  * 1 - off
  * 0 - on
  */
#define ON  PIN_LOW
#define OFF PIN_HIGH

/**
  * @brief  main
  * @param  None
  * @retval None
  */
int main(void) {
    uint32_t cnt = 0;
    uint8_t pin_r = rt_pin_get(LED_R);
    uint8_t pin_g = rt_pin_get(LED_G);
    uint8_t pin_b = rt_pin_get(LED_B);
    rt_pin_mode(pin_r, PIN_MODE_OUTPUT);
    rt_pin_mode(pin_g, PIN_MODE_OUTPUT);
    rt_pin_mode(pin_b, PIN_MODE_OUTPUT);

    while (1) {
        // Red
        rt_pin_write(pin_r, ON);
        rt_pin_write(pin_g, OFF);
        rt_pin_write(pin_b, OFF);
        rt_thread_mdelay(1000);

        // Green
        rt_pin_write(pin_r, OFF);
        rt_pin_write(pin_g, ON);
        rt_pin_write(pin_b, OFF);
        rt_thread_mdelay(1000);

        // Blue
        rt_pin_write(pin_r, OFF);
        rt_pin_write(pin_g, OFF);
        rt_pin_write(pin_b, ON);
        rt_thread_mdelay(1000);

        // Yellow (red+green)
        rt_pin_write(pin_r, ON);
        rt_pin_write(pin_g, ON);
        rt_pin_write(pin_b, OFF);
        rt_thread_mdelay(1000);

        // Purple (red+blue)
        rt_pin_write(pin_r, ON);
        rt_pin_write(pin_g, OFF);
        rt_pin_write(pin_b, ON);
        rt_thread_mdelay(1000);

        // Cyan (green + blue)
        rt_pin_write(pin_r, OFF);
        rt_pin_write(pin_g, ON);
        rt_pin_write(pin_b, ON);
        rt_thread_mdelay(1000);

        // White (red+green+blue)
        rt_pin_write(pin_r, ON);
        rt_pin_write(pin_g, ON);
        rt_pin_write(pin_b, ON);
        rt_thread_mdelay(1000);

        // black (all off)
```

```
        rt_pin_write(pin_r, OFF);
        rt_pin_write(pin_g, OFF);
        rt_pin_write(pin_b, OFF);
        rt_thread_mdelay(1000);
        cnt++;
    }

    return 0;
}
```

The code is very simple, mainly through the rt_pin_write() function to control different LEDs.

2.3. Result

After the compilation is completed, download the program through Downloader, and you can see the different seven colour lights appearing in sequence.

3. PWM control full color LED

3.1. Experimental analysis

Before the formal explanation, let's talk about what PWM is. PWM (Pulse Width Modulation) is a method of digitally encoding the level of an analogue signal. The duty cycle of a square wave is used to encode the level of a specific analogue signal through pulses of different frequencies, so that a device that obtains a series of pulses of equal amplitude at the output and uses these pulses to replace the desired waveform.

The figure above is a diagram of the PWM principle. It is assumed that the timer’s working mode is counting up. When the count value is less than the threshold, it will output a level state, such as high level. When the count value is greater than the threshold, it will output the opposite voltage, such as low level. When the count value reaches the maximum value, the counter starts counting again from 0 and returns to the initial level state. The ratio of the high level duration (pulse width) to the cycle time is the duty cycle, ranging from 0 to 100%. The duration of the high level in the above figure is exactly half of the cycle time, so the duty cycle is 50%.

In the previous section, we talked about controlling the on and off of the three primary colours to achieve seven colours. In this section, we use PWM to control the colours of the three lights R, G, and B. Combined, you can mix into a variety of different colours.

So what to do? In fact, it is very simple, that is, by changing the duty cycle of the level, RGB can display different colour values, and then display the desired colour through combination. What is

the difference between controlling the three primary colours through PWM and directly controlling the three primary colours? In fact, there is no difference. The granularity controlled by PWM is finer, so more colours can be obtained.

It is worth noting that the GPIO of LED_R should be adjusted here, and LED_R should be connected to PA1 to facilitate subsequent control.

Name	Type	Function
PE4(LED_G)	I/O	SPI0DI-G2 SPI1DI-G6 LPWM0-G1 IISMCLK-G2 PE4
PA1(LED_R)	I/O	SPDIF1 SPI1CLK-G1 TX0-G5 HSTRX-G5 LPWM1-G3 IISDO-G1 PA1
PA2(LED_B)	I/O	SPI1DI-G1 LPWM2-G3 IISSCLK-G1 PA2

2.3. Code implementation

First of all, we need to configure the PWM driver. Double-click the RT-Thread Settings file, open the RT-Thread project configuration interface, check the PWM driver, and then save it.

Next you need to enable the PWM of the 3 LEDs.

It can also be configured through the **env** tool.

You can also directly add the following macro definitions in rtconfig.h.

```
rtconfig.h

#define BSP_USING_USB_TO_USART

/* On-chip Peripheral Drivers */

#define BSP_USING_UART0
#define BSP_USING_PWM
#define BSP_USING_LPWM0

/* G1, G2 and G3 are mutually exclusive */

#define BSP_USING_LPWM0_G1
#define BSP_USING_LPWM1

/* G1, G2 and G3 are mutually exclusive */

#define BSP_USING_LPWM1_G3
#define BSP_USING_LPWM2

/* G1, G2 and G3 are mutually exclusive */

#define BSP_USING_LPWM2_G3
```

If the above addition is successful, compile and flash it to the development board, check the settings, and you can see the PWM device that controls the 3 LEDs.

```
 \ | /
- RT -     Thread Operating System
 / | \     4.0.3 build Mar 14 2021
 2006 - 2021 Copyright by rt-thread team
msh >list_device
device           type         ref count
-------- -------------------- ----------
lpwm2    Miscellaneous Device 0
lpwm1    Miscellaneous Device 0
lpwm0    Miscellaneous Device 0
uart1    Character Device     0
uart0    Character Device     2
pin      Miscellaneous Device 0
msh >
```

The next step is to download the application code, create a new file under the applications folder, add a thread task, and then operate 3 LEDs in the task. See code below:

```
/**
  ****************************************************************************
  * @file                task.c
  * @author              BruceOu
  * @version             V1.0
  * @date                2021-03-12
  * @blog                https://blog.bruceou.cn/
  * @Official Accounts   嵌入式实验楼
  * @brief               RTT任务
  ****************************************************************************
  */

/*Includes*****************************************************************/
#include "task.h"
```

```
#define THREAD_PRIORITY       7
#define THREAD_STACK_SIZE     512
#define THREAD_TIMESLICE      3
#define PWM_DEV_NAME_R        "lpwm1" /* PWM设备名称 */
#define PWM_DEV_CHANNEL_R     3       /* PWM通道 */
#define PWM_DEV_NAME_G        "lpwm0" /* PWM设备名称 */
#define PWM_DEV_CHANNEL_G     1       /* PWM通道 */
#define PWM_DEV_NAME_B        "lpwm2" /* PWM设备名称 */
#define PWM_DEV_CHANNEL_B     3       /* PWM通道 */
struct rt_device_pwm *pwm_dev_r;      /* PWM设备句柄 */
struct rt_device_pwm *pwm_dev_g;      /* PWM设备句柄 */
struct rt_device_pwm *pwm_dev_b;      /* PWM设备句柄 */
static rt_thread_t pwm_led_tid = RT_NULL;
/* 线程 pwm_led_thread_entry 的入口函数 */

/**
  * @brief  pwm_led_thread_entry
  * @param  parameter
  * @retval None
  */
static void pwm_led_thread_entry(void *parameter) {
    rt_uint32_t period, pulse_r,pulse_g,pulse_b, dir_r,dir_g,dir_b;
    period = 655360;    /* 周期为0.5ms，单位为纳秒ns */
    dir_r = 1;          /* PWM脉冲宽度值的增减方向 */
    dir_g = 1;
    dir_b = 1;
    pulse_r = 0;        /* PWM脉冲宽度值，单位为纳秒ns */
    pulse_g = 0;
    pulse_b = 0;
    rt_uint16_t r,g,b;
    /* 查找设备 */
    pwm_dev_r = (struct rt_device_pwm *)rt_device_find(PWM_DEV_NAME_R);

    if (pwm_dev_r == RT_NULL) {
        rt_kprintf("pwm led r run failed! can't find %s device!\n", PWM_DEV_NAME_G);
    }

    pwm_dev_g = (struct rt_device_pwm *)rt_device_find(PWM_DEV_NAME_G);

    if (pwm_dev_g == RT_NULL) {
        rt_kprintf("pwm led g run failed! can't find %s device!\n", PWM_DEV_NAME_G);
    }

    pwm_dev_b = (struct rt_device_pwm *)rt_device_find(PWM_DEV_NAME_B);

    if (pwm_dev_b == RT_NULL) {
        rt_kprintf("pwm led b run failed! can't find %s device!\n", PWM_DEV_NAME_B);
    }

    /* 设置PWM周期和脉冲宽度默认值 */
    rt_pwm_set(pwm_dev_r, PWM_DEV_CHANNEL_R, period, pulse_r);
    rt_pwm_set(pwm_dev_g, PWM_DEV_CHANNEL_G, period, pulse_g);
    rt_pwm_set(pwm_dev_b, PWM_DEV_CHANNEL_B, period, pulse_b);

    /* 使能设备 */
    rt_pwm_enable(pwm_dev_r, PWM_DEV_CHANNEL_R);
    rt_pwm_enable(pwm_dev_g, PWM_DEV_CHANNEL_G);
    rt_pwm_enable(pwm_dev_b, PWM_DEV_CHANNEL_B);

    while (1) {
        for (r =0 ; r < 8; r++) {
            if (dir_r) {
                pulse_r += 81920;      /* 从0值开始每次增加81920ns */
            } else {
                pulse_r -= 81920;      /* 从最大值开始每次减少81920ns */
            }

            if ((pulse_r) >= period) {
                dir_r = 0;
            }

            if (81920 > pulse_r) {
                dir_r = 1;
            }
```

```
            /* 设置 PWM 周期和脉冲宽度 */
            rt_pwm_set(pwm_dev_r, PWM_DEV_CHANNEL_R, period, pulse_r);

            for (g = 0; g < 8; g++) {
                if (dir_g) {
                    pulse_g += 81920;      /* 从 0 值开始每次增加 81920ns */
                } else {
                    pulse_g -= 81920;      /* 从最大值开始每次减少 81920ns */
                }

                if ((pulse_g) >= period) {
                    dir_g = 0;
                }

                if (81920 > pulse_g) {
                    dir_g = 1;
                }

                rt_pwm_set(pwm_dev_g, PWM_DEV_CHANNEL_G, period, pulse_g);

                for (b = 0; b < 8; b++) {
                    rt_thread_mdelay(10);

                    if (dir_b) {
                        pulse_b += 81920;      /* 从 0 值开始每次增加 81920ns */
                    } else {
                        pulse_b -= 81920;      /* 从最大值开始每次减少 81920ns */
                    }

                    if ((pulse_b) >= period) {
                        dir_b = 0;
                    }

                    if (81920 > pulse_b) {
                        dir_b = 1;
                    }

                    rt_pwm_set(pwm_dev_b, PWM_DEV_CHANNEL_B, period, pulse_b);
                }
            }
        }
    }
}

/* 线程初始化*/
int thread_init(void) {
    /* 创建线程，名称是 pwm_led_thread，入口是 pwm_led_thread*/
    pwm_led_tid = rt_thread_create(
        "pwm_led_thread",
        pwm_led_thread_entry,
        RT_NULL,
        THREAD_STACK_SIZE,
        THREAD_PRIORITY,
        THREAD_TIMESLICE
    );

    /* 如果获得线程控制块，启动这个线程 */
    if (pwm_led_tid != RT_NULL)
        rt_thread_startup(pwm_led_tid);

    return 0;
}

/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(thread_init, thread init);
```

Of course, the above code can also be optimized, if you are interested in modifying it yourself. Control PWM mainly uses the following three functions:

```
rt_device_find()  // Find the device according to the PWM device name to get the device handle
rt_pwm_set()      // Set PWM Period and Pulse Width
rt_pwm_enable()   // Enable PWM device
```

The code is also very simple, which is to control the pulse width of the three LEDs through PWM to get different colors.

[PWM device documentation link]

3.3. Result

Reset after programming successfully, type thread_init in the terminal.

```
 \ | /
- RT -     Thread Operating System
 / | \     4.0.3 build Mar 14 2021
 2006 - 2021 Copyright by rt-thread team
msh >thread_init
msh >
```

https://blog.bruceou.cn/

You can see that the LED flashes in different colours.

Use of serial port devices

Original post: https://club.rt-thread.org/ask/article/c77650c7e1112930.html
Copyright: [BruceOu](#) License: CC-BY-SA 4.0

The peripherals in RTT exist as devices. When building a project, AB32VG1 serial port 0 is used as the system debugging serial port. If there is a serial port module that needs to communicate with the microcontroller, you can initialise another serial port. If there is no driver, then the first step is to write the driver. For the AB32VG1 development board, the written UART driver only needs to open the corresponding device.

1. Simple use of serial devices

1.1. Sending and receiving on the serial port

UART0 has been used as a serial port for downloading and debugging, so UART1 is used here. Let's look at the circuit first.

It can be seen from the circuit diagram that PA3 and PA4 are used here, so it is necessary to connect PA3 and PA4 to the PC through the USB to serial port and check the sending and receiving information through the serial port debugging assistant.

The project enables UART0 and UART1 by default, and the UART device can also be seen through the finish terminal.

Just write the application code here. Create new task.c and task.h files in applications.

task.c

```
/**
  ****************************************************************************
  * @file                 task.c
```

```
  * @author              BruceOu
  * @lib version         V3.5.0
  * @version             V1.0
  * @date                2021-01-10
  * @blog                https://blog.bruceou.cn/
  * @Official Accounts   嵌入式实验楼
  * @brief               RTT任务
  ******************************************************************************
  */
/*Includes*************************************************************************/
#include "task.h"

struct serial_configure config = RT_SERIAL_CONFIG_DEFAULT;  /* 初始化配置参数 */
/* 用于接收消息的信号量 */
static struct rt_semaphore rx_sem;
static rt_device_t serial;

/**
  * @brief  uart_input //接收数据回调函数
  * @param  dev
  *         size
  * @retval RT_EOK
  */
static rt_err_t uart_input(rt_device_t dev, rt_size_t size) {
    /* 串口接收到数据后产生中断，调用此回调函数，然后发送接收信号量 */
    rt_sem_release(&rx_sem);
    return RT_EOK;
}

/**
  * @brief  serial_thread_entry
  * @param  parameter
  * @retval None
  */
static void serial_thread_entry(void *parameter) {
    char ch;

    while (1) {
        /* 从串口读取一个字节的数据，没有读取到则等待接收信号量 */
        while (rt_device_read(serial, -1, &ch, 1) != 1)
        {
            /* 阻塞等待接收信号量，等到信号量后再次读取数据 */
            rt_sem_take(&rx_sem, RT_WAITING_FOREVER);
        }
        /* 读取到的数据输出 */
        rt_kprintf("%c",ch);
    }
}

/**
  * @brief  thread_serial
  * @param  None
  * @retval ret
  */
int thread_serial(void) {
    rt_err_t ret = RT_EOK;
    char uart_name[RT_NAME_MAX];
    char str[] = "hello RT-Thread!\r\n";
    rt_strncpy(uart_name, SAMPLE_UART_NAME, RT_NAME_MAX);
    /* 查找系统中的串口设备 */
    serial = rt_device_find(uart_name);

    if (!serial) {
        rt_kprintf("find %s failed!\n", uart_name);
        return RT_ERROR;
    }

    /* 修改串口配置参数 */
    config.baud_rate = BAUD_RATE_9600;        //修改波特率为 9600
    config.data_bits = DATA_BITS_8;           //数据位 8
    config.stop_bits = STOP_BITS_1;           //停止位 1
    config.bufsz     = 64;                   //修改缓冲区 buff size 为 128
    config.parity    = PARITY_NONE;           //无奇偶校验位
    /* 控制串口设备。通过控制接口传入命令控制字，与控制参数 */
```

```
    rt_device_control(serial, RT_DEVICE_CTRL_CONFIG, &config);
    /* 初始化信号量 */
    rt_sem_init(&rx_sem, "rx_sem", 0, RT_IPC_FLAG_FIFO);
    /* 以中断接收及轮询发送模式打开串口设备 */
    rt_device_open(serial, RT_DEVICE_FLAG_INT_RX);
    /* 设置接收回调函数 */
    rt_device_set_rx_indicate(serial, uart_input);
    /* 发送字符串 */
    rt_device_write(serial, 0, str, (sizeof(str) - 1));
    /* 创建 serial 线程 */
    rt_thread_t thread = rt_thread_create("serial", serial_thread_entry, RT_NULL, 1024, 25, 10);

    /* 创建成功则启动线程 */
    if (thread != RT_NULL) {
        rt_thread_startup(thread);
    } else {
        ret = RT_ERROR;
    }

    return ret;
}

/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(thread_serial, uart device sample);
```

task.h

```
#ifndef _TASK_H_
#define _TASK_H_

/* Standard includes. */
#include <stdio.h>

/* rtthread includes. */
#include <rtdevice.h>
#include <board.h>

#define SAMPLE_UART_NAME                "uart1"
int thread_serial(void);

#endif
```

The code is very simple, a thread is created, and interrupt reception and polling data are used in the thread. The main functions are as follows:

```
rt_device_find()     // Find device
rt_device_open()     // Open the device
rt_device_read()     // Read data
rt_device_write()    // Write data
rt_device_control()  // Control device
rt_device_set_rx_indicate()    // Set receive callback function
rt_device_set_tx_complete()    // Set transmit complete callback
rt_device_close()    // Close device
```

1.2. Result

Compile, download. Enter thread_serial in the RT-Thead console:

Use the serial port debugging assistant to view the output information of uart1.

Similarly, data can also be sent through the serial port debugging assistant, and the received data can be seen on the RT-Thead console.

For detailed use of UART applications, please go to the RTT Documentation Center.

[UART device documentation link]

2. serial device optimization

Anyone who is familiar with RTT knows that all devices in RTT are switched on and off through macro definitions, but this part of AB32VG1 is not perfect, so what can be done to switch UART devices through configuration like other BSPs? Next, I will explain them one by one.

First of all, we need to know that the UART initialization of AB32VG1 calls the rt_hw_usart_init() function in drv_uart.c.

```
drv_usart.c  rtconfig.h  drv_usart.h

int rt_hw_usart_init(void)
{
    rt_size_t obj_num = sizeof(uart_obj) / sizeof(struct ab32_uart);
    struct serial_configure config = RT_SERIAL_CONFIG_DEFAULT;
    rt_err_t result = 0;

    rt_hw_interrupt_install(IRQ_UART0_2_VECTOR, uart_isr, RT_NULL, "ut_isr");

    for (int i = 0; i < obj_num; i++)
    {
        /* init UART object */
        uart_obj[i].config          = &uart_config[i];
        uart_obj[i].serial.ops      = &ab32_uart_ops;
        uart_obj[i].serial.config   = config;
        uart_obj[i].serial.config.baud_rate = 1500000;

        /* register UART device */
        result = rt_hw_serial_register(&uart_obj[i].serial, uart_obj[i].config->name,
                                       RT_DEVICE_FLAG_RDWR
                                       | RT_DEVICE_FLAG_INT_RX
                                       | RT_DEVICE_FLAG_INT_TX
                                       | uart_obj[i].uart_dma_flag
                                       , NULL);
        RT_ASSERT(result == RT_EOK);
    }

    return result;
}
```

Here uart_config defines the base address of UART and determines the size of UART. Therefore, the definition of this variable needs to be modified as follows:

```
drv_usart.c  rtconfig.h  drv_usart.h  ab32vg1.h

static struct ab32_uart_config uart_config[] =
{
#ifdef BSP_USING_UART0
    {
        .name = "uart0",
        .instance = UART0_BASE,
    },
#endif
#ifdef BSP_USING_UART1
    {
        .name = "uart1",
        .instance = UART1_BASE,
    },
#endif
};
```

In fact, a few macros are added, and the enumeration is also modified as follows:

```
drv_usart.c  rtconfig.h  drv_usart.h  ab32vg1.h

enum
{
#ifdef BSP_USING_UART0
    UART0_INDEX,
#endif
#ifdef BSP_USING_UART1
    UART1_INDEX,
#endif
};
```

In addition, the interrupt function of UART needs to be modified:

```
drv_usart.c  rtconfig.h
static void uart_isr(int vector, void *param)
{
    rt_interrupt_enter();
#ifdef BSP_USING_UART0
    if(hal_uart_getflag(UART0_BASE, UART_FLAG_RXPND))       //RX one byte finish
    {
        rt_hw_serial_isr(&(uart_obj[UART0_INDEX].serial), RT_SERIAL_EVENT_RX_IND);
    }
#endif
#ifdef BSP_USING_UART1
    if(hal_uart_getflag(UART1_BASE, UART_FLAG_RXPND))       //RX one byte finish
    {
        rt_hw_serial_isr(&(uart_obj[UART1_INDEX].serial), RT_SERIAL_EVENT_RX_IND);
    }
#endif
    rt_interrupt_leave();
}
```

Next we can pass:

```
#define BSP_USING_UART0
#define BSP_USING_UART1
```

Controls the opening and closing of the UART device. It’s not over here, if you want to realize the configuration of the words, you need to modify the Kconfig file of the board. The modified content is as follows:

```
menu "On-chip Peripheral Drivers"
    menuconfig BSP_USING_UART
        bool "Enable UART"
        default y
        select RT_USING_SERIAL
        if BSP_USING_UART
            config BSP_USING_UART0
                bool "Enable UART0"
                default y
            config BSP_USING_UART1
                bool "Enable UART1"
                default n
        endif
```

At this point, the modification is completed, and then the configuration just now can be used. Here RT-Thread Studio configuration, open RT-Thread Setting, you can configure UART.

As for the fact that UART0 cannot be unconfigured, it is because UART0 has been configured as a debug serial port and has been configured forcibly.

Of course, it can also be configured with the ENV tool, enter the project directory, open the env console, and enter the menuconfig command to open its interface. Go through the following options in order:

→ Hardware Drivers Config → On-chip Peripheral → Enable UART

Enable UART1, save and exit.

The same as the result of adding a serial device in RT-Thread Studio, the following macro definitions will be added in rtconfig.h in the end.

```
rtconfig.h
152
153
154 /* miscellaneous packages */
155
156
157 /* samples: kernel and components samples */
158
159
160 /* games: games run on RT-Thread console */
161
162
163 /* Hardware Drivers Config */
164
165 /* Onboard Peripheral Drivers */
166
167 #define BSP_USING_USB_TO_USART
168
169 /* On-chip Peripheral Drivers */
170
171 #define BSP_USING_UART
172 #define BSP_USING_UART0
173 #define BSP_USING_UART1
174
175 /* Board extended module Drivers */
176
177 #define BOARD_BLUETRUM_EVB
178
179 #endif
```

Well, the optimization of the serial device is here. Now the UART device can be turned on and off.

Watchdog

Original post: https://club.rt-thread.org/ask/article/c371d9f3da836866.html
Copyright: BruceOu License: CC-BY-SA 4.0

1. Working principle of watchdog

The watchdog is a counter in layman's terms. When the value of the counter decreases from a certain value to 0, the system will generate a reset signal. If the value of the counter is refreshed before the count is reduced to 0, then no reset signal will be generated. This action is what we often call feeding the dog.

2. Configure watchdog

The first step is to open the WDT device first, open RT-Thread Setting, and check the WDT option.

At the same time, turn on the timer of WDT.

This completes the configuration, and then save it.

3. Watchdog code implementation

The watchdog is generally used to detect and solve the faults caused by the program. For example, the normal running time of a program is 50ms. After running this section of the program, the dog is fed immediately. We set the timing overflow time of the independent watchdog to 60ms , a little more than 50ms of the program we need to monitor. If the dog has not been fed after 60ms, it means that the program we monitor has broken down and ran away, then a system reset will be generated to allow the program to run again.

This procedure is very simple, just need to feed the dog.

```
/**
  ******************************************************************************
  * @file                task.c
  * @author              BruceOu
  * @lib version         V3.5.0
  * @version             V1.0
  * @date                2021-01-10
  * @blog                https://blog.bruceou.cn/
  * @Official Accounts   嵌入式实验楼
  * @brief               RTT任务
  ******************************************************************************
  */
/*Includes*************************************************************************/
#include "task.h"

#define WDT_DEVICE_NAME    "wdt"    /* 看门狗设备名称 */
static rt_device_t wdg_dev;         /* 看门狗设备句柄 */

static void idle_hook(void) {
    /* 在空闲线程的回调函数里喂狗 */
    rt_device_control(wdg_dev, RT_DEVICE_CTRL_WDT_KEEPALIVE, NULL);
    //rt_kprintf("feed the dog!\n ");
}

/**
  * @brief  thread_wdt
  * @param  None
  * @retval ret
  */
int thread_wdt(void) {
    rt_err_t ret = RT_EOK;
    rt_uint32_t timeout = 3;        /* 溢出时间，单位：秒 */
    char device_name[RT_NAME_MAX];
    rt_strncpy(device_name, WDT_DEVICE_NAME, RT_NAME_MAX);

    /* 根据设备名称查找看门狗设备，获取设备句柄 */
    wdg_dev = rt_device_find(device_name);
    if (!wdg_dev) {
        rt_kprintf("find %s failed!\n", device_name);
        return RT_ERROR;
    }

    /* 设置看门狗溢出时间 */
    ret = rt_device_control(wdg_dev, RT_DEVICE_CTRL_WDT_SET_TIMEOUT, &timeout);

    if (ret != RT_EOK) {
        rt_kprintf("set %s timeout failed!\n", device_name);
        return RT_ERROR;
    }

    /* 启动看门狗 */
    ret = rt_device_control(wdg_dev, RT_DEVICE_CTRL_WDT_START, RT_NULL);

    if (ret != RT_EOK) {
        rt_kprintf("start %s failed!\n", device_name);
        return -RT_ERROR;
    }

    /* 设置空闲线程回调函数 */
    rt_thread_idle_sethook(idle_hook);
    return ret;
}

/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(thread_wdt, wdt sample);
```

The program is very simple, after initializing the watchdog, feed the dog continuously in the main loop. It is worth noting that you should not add a delay function to the idle_hook function. In addition, when the watchdog is officially used, just cancel the printing.

Please note that the timeout period here can also be replaced by the following macro definition:

```
#ifndef DRV_WDT_H__
#define DRV_WDT_H__

#define AB32_WDT_TIMEOUT_1MS        0
#define AB32_WDT_TIMEOUT_256MS      1
#define AB32_WDT_TIMEOUT_512MS      2
#define AB32_WDT_TIMEOUT_1024MS     3
#define AB32_WDT_TIMEOUT_2048MS     4
#define AB32_WDT_TIMEOUT_4096MS     5
#define AB32_WDT_TIMEOUT_8192MS     6
#define AB32_WDT_TIMEOUT_16384MS    7

#endif
```

4. Result

The compilation is correct, download the program, enter thread_wdt, the console shows:

You will open the terminal to continuously output and print information, indicating that the watchdog is working normally. If it is reset, it means that the timer overflows and the feeding of the dog fails.

This experiment is very simple, and later we will use the watchdog and other devices to achieve more complex functions.

[Watchdog documentation link]

Feed the (watch)dog with buttons

Original post: https://club.rt-thread.org/ask/article/aa53f278298da213.html
Copyright: BruceOu License: CC-BY-SA 4.0

1. Introduction to buttons

I talked about the use of the watchdog in the previous section, and this article will use buttons to feed the dog. The button circuit is as follows:

This article uses S2. It can be seen from the figure that the button is common ground, so we need to detect the falling edge to judge whether the button is pressed.

2. Key to feed the dog – Implementation

Buttons are generally implemented by cyclic scanning and interrupts. The registers of AB32VG1 are incomplete, and the official does not provide the implementation of interrupts. This article directly uses the scan method to achieve. code show as below:

task.c

```
/**
  ******************************************************************************
  * @file                task.c
  * @author              BruceOu
  * @lib version         V3.5.0
  * @version             V1.0
  * @date                2021-03-20
  * @blog                https://blog.bruceou.cn/
  * @Official Accounts   嵌入式实验楼
  * @brief               RTT任务
  ******************************************************************************
  */
/*Includes*****************************************************************/
#include "task.h"

static rt_device_t wdg_dev;         /* 看门狗设备句柄 */

/**
  * @brief  wdt_init
  * @param  None
  * @retval ret
  */
int wdt_init(void) {
    rt_err_t ret = RT_EOK;
    rt_uint32_t timeout = 5;        /* 溢出时间，4096ms 单位：秒 */
    char device_name[RT_NAME_MAX];
    rt_strncpy(device_name, WDT_DEVICE_NAME, RT_NAME_MAX);

    /* 根据设备名称查找看门狗设备，获取设备句柄 */
    wdg_dev = rt_device_find(device_name);
    if (!wdg_dev) {
        rt_kprintf("find %s failed!\n", device_name);
        return RT_ERROR;
    }

    /* 设置看门狗溢出时间 */
    ret = rt_device_control(wdg_dev, RT_DEVICE_CTRL_WDT_SET_TIMEOUT, &timeout);
    if (ret != RT_EOK) {
        rt_kprintf("set %s timeout failed!\n", device_name);
        return RT_ERROR;
    }
```

```
    /* 启动看门狗 */
    ret = rt_device_control(wdg_dev, RT_DEVICE_CTRL_WDT_START, RT_NULL);
    if (ret != RT_EOK) {
        rt_kprintf("start %s failed!\n", device_name);
        return -RT_ERROR;
    }

    return ret;
}

/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(wdt_init, wdt sample);
static rt_thread_t tid_key = RT_NULL;

/**
  * @brief  read_key
  * @param  key
  * @retval int
  */
int read_key(uint8_t key) {
    return rt_pin_read(key);
}

/**
  * @brief  thread_key_entry
  * @param  parameter
  * @retval None
  */
static void thread_key_entry(void *parameter) {
    uint8_t key = *(uint8_t *)parameter;

    while (1) {
        if (read_key(key) == PIN_LOW) {
            rt_thread_mdelay(50);

            if (read_key(key) == PIN_LOW) {
                /* 在空闲线程的回调函数里喂狗 */
                rt_device_control(wdg_dev, RT_DEVICE_CTRL_WDT_KEEPALIVE, NULL);
                rt_kprintf("feed the dog");
            }
        }

        rt_thread_mdelay(100);
    }
}

/**
  * @brief  thread_key
  * @param  None
  * @retval None
  */
int thread_key(void) {
    uint8_t key = rt_pin_get(KEY_PIN_NUM);

    // 设置按键为上拉输入模式
    rt_pin_mode(key, PIN_MODE_INPUT_PULLUP );

    /* 创建线程  */
    tid_key = rt_thread_create
    (
        "thread key",
        thread_key_entry,
        &key,
        512,
        8,
        5
    );

    if (tid_key != RT_NULL) {
        rt_thread_startup(tid_key);
    }

    rt_thread_mdelay(300);
    return 0;
}

/* 导出到 msh 命令列表中 */
```

```
MSH_CMD_EXPORT(thread_key, thread key sample);
```

task.h

```
#ifndef _TASK_H_
#define _TASK_H_

/* Standard includes. */
#include <stdio.h>

/* rtthread includes. */
#include <rtthread.h>
#include <rtdevice.h>
#include <board.h>

#define WDT_DEVICE_NAME    "wdt"    /* 看门狗设备名称 */
#define KEY_PIN_NUM  "PF.1"

int thread_key(void);
int wdt_init(void);

#endif
```

The code is very simple. In the button thread, the dog is fed by pressing the button. If the dog is fed overtime, the system is reset.

Please note that the timeout here is defined in drv_wdt.h.

```
#ifndef DRV_WDT_H__
#define DRV_WDT_H__

#define AB32_WDT_TIMEOUT_1MS        0
#define AB32_WDT_TIMEOUT_256MS      1
#define AB32_WDT_TIMEOUT_512MS      2
#define AB32_WDT_TIMEOUT_1024MS     3
#define AB32_WDT_TIMEOUT_2048MS     4
#define AB32_WDT_TIMEOUT_4096MS     5
#define AB32_WDT_TIMEOUT_8192MS     6
#define AB32_WDT_TIMEOUT_16384MS    7
```

https://blog.bruceou.cn/

```
#endif
```

3. Result

The compilation is correct, download the program, first enter thread_key, then enter wdt_init, the phenomenon is as follows:

```
 \ | /
- RT -     Thread Operating System
 / | \     4.0.3 build Mar 21 2021
 2006 - 2021 Copyright by rt-thread team
msh />thread_key
msh />wdt_init
[I/drv.wdt] The watchdog timeout is set to 4096ms
msh />feed the dog
msh />feed the dog
msh />feed the dog
msh />feed the dog
msh />feed the dog
msh />feed the dog
msh />
 \ | /
- RT -     Thread Operating System
 / | \     4.0.3 build Mar 21 2021
 2006 - 2021 Copyright by rt-thread team
msh />
```

系统复位

https://blog.bruceou.cn/

Press the button S2, there will be a printed message, indicating that the watchdog is working normally, if you do not feed the dog, the system will reset.

Music player

Original post: https://club.rt-thread.org/ask/article/aa53f278298da213.html
Copyright: BruceOu License: CC-BY-SA 4.0

1. Introduction

In the previous chapters, we used AB32VG1 to do a few small experiments. This chapter combines the previous contents to make a music player. The main functions are as follows:

1. It can store multiple complete music pieces;
2. Realize song switching;
3. Realize volume adjustment.

Of course, the above are the most basic functions, and the selection of the song playback mode can also be realized. The content of this chapter mainly realizes the above three basic functions. In addition, it is very cool to change the flashing frequency of the RGB light according to the volume.

Well, let's take a look at how to implement a music player.

2. Music player configuration

The whole project configuration is divided into three parts: audio part, storage and file system part, LED, serial port and other three parts.

For the creation of the project, please refer to the author's previous articles.

2.1. Audio configuration

First look at the audio section. Enabling hardware is essential.

After enabling the hardware, we can use the audio device, but the audio device is more complicated than other peripherals. RT-Thread provides the software package WavPlay for operating the audio device. Just enable it.

Please note that the WavPlay package depends on optparse, so the optparse package is automatically selected after wavplayer is checked. The optparse module is mainly used to pass command parameters for scripts, and uses pre-defined options to parse command line parameters.

2.2. Storage file system configuration

The internal storage of the AB32VG1 development board is very small, but the music files are large, so an external storage device is required to store music, so here an SD card is used to store music. First, you need to enable the SD card device.

After the SD card device is enabled, the file system is checked by default.

2.3. Other configurations

The first two parts are the important configuration of this project and are indispensable, and the next step is configuration, PWM, UART, and KEY devices.

For the configuration of PWM and UART, refer to the author's previous articles.

Add a button function package below.

MultiButton is a small and easy-to-use event-driven button driver module, which can expand buttons indefinitely. The callback asynchronous processing method of button events can simplify your program structure and remove redundant hard-coded button processing, which is very easy to use.

Well, that's all for the configuration of the music player.

3. Implementation of music player

This article will play music, switch songs, control volume and other operations by pressing buttons or sending commands through the serial port. In addition, adjust the frequency of PWM according to the volume, so as to change the blinking frequency of RGB lights.

3.1. Brief analysis of WavPlay playing audio

For audio devices, the operation process is as follows:

1. First find the Audio device to obtain the device handle.
2. Open the Audio device as write-only.
3. Set audio parameter information (sampling rate, channel, etc.).
4. Decode the data of the audio file.
5. Write audio file data.
6. When the playback is complete, turn off the device.

If it is quite complicated to implement these operations by yourself, the wavplayer software package encapsulates the operation of the audio device. The master needs to simply call several function interfaces to play music. The main interfaces are as follows:

```
int wavplayer_play(char *uri);  // Play music
int wavplayer_stop(void);       // End playing
int wavplayer_pause(void);      // Pause playback
int wavplayer_resume(void);     // Continue playing
int wavplayer_volume_set(int volume);  // Volume setting
```

Of course, only the application implementation is explained here. For the audio driver, please refer to the official manual.

[wavplayer documentation link]

3.2. PWM control RGB lights

This part of the content has already been discussed in the previous chapters, so I won't talk about it here. The code is as follows:

```
#include "led_app.h"

#define THREAD_PRIORITY         7
#define THREAD_STACK_SIZE       512
#define THREAD_TIMESLICE        3
uint32_t pulse_pulse = 90000;
#define PWM_DEV_NAME_R        "t5pwm"  /* PWM设备名称 */
#define PWM_DEV_CHANNEL_R     1       /* PWM通道 */
#define PWM_DEV_NAME_G        "lpwm0"  /* PWM设备名称 */
#define PWM_DEV_CHANNEL_G     1       /* PWM通道 */
#define PWM_DEV_NAME_B        "lpwm2"  /* PWM设备名称 */
#define PWM_DEV_CHANNEL_B     3       /* PWM通道 */
struct rt_device_pwm *pwm_dev_r;      /* PWM设备句柄 */
struct rt_device_pwm *pwm_dev_g;      /* PWM设备句柄 */
struct rt_device_pwm *pwm_dev_b;      /* PWM设备句柄 */
static rt_thread_t pwm_led_tid = RT_NULL;

/* 线程 pwm_led_thread_entry 的入口函数 */
/**
  * @brief  pwm_led_thread_entry
  * @param  parameter
  * @retval None
  */
```

```
static void pwm_led_thread_entry(void *parameter) {
    rt_uint32_t period, pulse_r,pulse_g,pulse_b, dir_r,dir_g,dir_b;
    period = 655360;    /* 周期为 0.5ms，单位为纳秒 ns */
    dir_r = 1;           /* PWM 脉冲宽度值的增减方向 */
    dir_g = 1;
    dir_b = 1;
    pulse_r = 0;          /* PWM 脉冲宽度值，单位为纳秒 ns */
    pulse_g = 0;
    pulse_b = 0;
    rt_uint16_t r,g,b;

    /* 查找设备 */
    pwm_dev_r = (struct rt_device_pwm *)rt_device_find(PWM_DEV_NAME_R);

    if (pwm_dev_r == RT_NULL) {
        rt_kprintf("pwm led r run failed! can't find %s device!\n", PWM_DEV_NAME_G);
    }

    pwm_dev_g = (struct rt_device_pwm *)rt_device_find(PWM_DEV_NAME_G);

    if (pwm_dev_g == RT_NULL) {
        rt_kprintf("pwm led g run failed! can't find %s device!\n", PWM_DEV_NAME_G);
    }

    pwm_dev_b = (struct rt_device_pwm *)rt_device_find(PWM_DEV_NAME_B);

    if (pwm_dev_b == RT_NULL) {
        rt_kprintf("pwm led b run failed! can't find %s device!\n", PWM_DEV_NAME_B);
    }

    /* 设置 PWM 周期和脉冲宽度默认值 */
    rt_pwm_set(pwm_dev_r, PWM_DEV_CHANNEL_R, period, pulse_r);
    rt_pwm_set(pwm_dev_g, PWM_DEV_CHANNEL_G, period, pulse_g);
    rt_pwm_set(pwm_dev_b, PWM_DEV_CHANNEL_B, period, pulse_b);

    /* 使能设备 */
    rt_pwm_enable(pwm_dev_r, PWM_DEV_CHANNEL_R);
    rt_pwm_enable(pwm_dev_g, PWM_DEV_CHANNEL_G);
    rt_pwm_enable(pwm_dev_b, PWM_DEV_CHANNEL_B);

    while (1) {
        for (r =0 ; r < 8; r++) {
            if (dir_r) {
                pulse_r += pulse_pulse;      /* 从 0 值开始每次增加 5000ns */
            } else {
                pulse_r -= pulse_pulse;      /* 从最大值开始每次减少 5000ns */
            }
            if ((pulse_r) >= period) {
                dir_r = 0;
            }
            if (81920 > pulse_r) {
                dir_r = 1;
            }

            /* 设置 PWM 周期和脉冲宽度 */
            rt_pwm_set(pwm_dev_r, PWM_DEV_CHANNEL_R, period, pulse_r);

            for (g = 0; g < 8; g++) {
                if (dir_g) {
                    pulse_g += pulse_pulse;      /* 从 0 值开始每次增加 5000ns */
                } else {
                    pulse_g -= pulse_pulse;      /* 从最大值开始每次减少 5000ns */
                }

                if ((pulse_g) >= period) {
                    dir_g = 0;
                }

                if (81920 > pulse_g) {
                    dir_g = 1;
                }

                rt_pwm_set(pwm_dev_g, PWM_DEV_CHANNEL_G, period, pulse_g);

                for (b = 0; b < 8; b++) {
```

```
                    rt_thread_mdelay(10);

                    if (dir_b) {
                        pulse_b += pulse_pulse;      /* 从 0 值开始每次增加 5000ns */
                    } else {
                        pulse_b -= pulse_pulse;      /* 从最大值开始每次减少 5000ns */
                    }

                    if ((pulse_b) >= period) {
                        dir_b = 0;
                    }

                    if (81920 > pulse_b) {
                        dir_b = 1;
                    }

                    rt_pwm_set(pwm_dev_b, PWM_DEV_CHANNEL_B, period, pulse_b);
                }
            }
        }
    }
}

/* 线程初始化*/
int pwm_led(void) {
    /* 创建线程，名称是 pwm_led_thread，入口是 pwm_led_thread*/
    pwm_led_tid = rt_thread_create(
        "pwm_led_thread",
        pwm_led_thread_entry,
        RT_NULL,
        THREAD_STACK_SIZE,
        THREAD_PRIORITY,
        THREAD_TIMESLICE
    );

    /* 如果获得线程控制块，启动这个线程 */
    if (pwm_led_tid != RT_NULL) {
        rt_thread_startup(pwm_led_tid);
    }

    return 0;
}

/* 导出到 msh 命令列表中 */
//MSH_CMD_EXPORT(pwm_led, pwm led);
INIT_APP_EXPORT(pwm_led);
```

3.3. Serial port control audio equipment

Let's take a look at the code for UART to play music.

```
#include "uart_app.h"
#include "key_app.h"
#include "led_app.h"

#define SAMPLE_UART_NAME                "uart1"
uint8_t ch;
uint8_t r_index = 0;
uint8_t flag = 0;
extern uint32_t cnt_music;
extern uint32_t cnt_channels;
extern uint32_t cnt_volume;
extern uint32_t  start_flag;
extern char *table[NUM_OF_SONGS];
extern uint32_t pulse_pulse;
struct serial_configure config = RT_SERIAL_CONFIG_DEFAULT;  /* 初始化配置参数 */
/* 用于接收消息的信号量 */
static struct rt_semaphore rx_sem;
static rt_device_t serial;

void analyticald_data(void)
{
    uint8_t sum;

    if (ch == 0x01) {
```

```
        wavplayer_play(table[(cnt_music++) % NUM_OF_SONGS]);
    } else if (ch == 0x02) {
        if (cnt_volume < 11) {
            if(start_flag) {
                start_flag = 0;
                cnt_volume = (int)saia_volume_get()/10;
                pulse_pulse = 9000;
            } else {
                saia_volume_set(cnt_volume * 10);
                pulse_pulse = cnt_volume*9000;
            }
        } else {
            saia_volume_set(10);
            cnt_volume = 1;
            rt_kprintf("The volume has been adjusted to maximum\n");
        }

        cnt_volume ++;
        rt_kprintf("vol=%d\n", saia_volume_get());
    } else if (ch == 0x03) {
        if (cnt_channels < 3) {
            saia_channels_set(cnt_channels);
        } else {
            saia_channels_set(cnt_channels);
            cnt_channels = 1;
        }

        cnt_channels++;
    }
}

/* 接收数据回调函数 */
static rt_err_t uart_rx_ind(rt_device_t dev, rt_size_t size) {
    /* 串口接收到数据后产生中断，调用此回调函数，然后发送接收信号量 */
    if (size > 0) {
        rt_sem_release(&rx_sem);
    }

    return RT_EOK;
}

static char uart_sample_get_char(void) {
    uint8_t ch;

    while (rt_device_read(serial, 0, &ch, 1) == 0) {
        rt_sem_control(&rx_sem, RT_IPC_CMD_RESET, RT_NULL);
        rt_sem_take(&rx_sem, RT_WAITING_FOREVER);
    }

    return ch;
}

/* 数据解析线程 */
static void data_parsing(void) {
    while (1) {
        ch = uart_sample_get_char();
        flag = 1;
    }
}

int uart_init(void) {
    rt_err_t ret = RT_EOK;
    char uart_name[RT_NAME_MAX];
    //char str[] = "hello RT-Thread!\r\n";
    rt_strncpy(uart_name, SAMPLE_UART_NAME, RT_NAME_MAX);

    /* 查找系统中的串口设备 */
    serial = rt_device_find(uart_name);
    if (!serial) {
        rt_kprintf("find %s failed!\n", uart_name);
        return RT_ERROR;
    }

    /* step2: 修改串口配置参数 */
    config.baud_rate = BAUD_RATE_9600;        //修改波特率为 9600
    config.data_bits = DATA_BITS_8;           //数据位 8
```

```
    config.stop_bits = STOP_BITS_1;           //停止位 1
    config.bufsz     = 128;                   //修改缓冲区 buff size 为 128
    config.parity    = PARITY_NONE;           //无奇偶校验位
    /* step3: 控制串口设备。通过控制接口传入命令控制字，与控制参数 */
    rt_device_control(serial, RT_DEVICE_CTRL_CONFIG, &config);
    /* 初始化信号量 */
    rt_sem_init(&rx_sem, "rx_sem", 0, RT_IPC_FLAG_FIFO);
    /* 以中断接收及轮询发送模式打开串口设备 */
    rt_device_open(serial, RT_DEVICE_FLAG_INT_RX);
    /* 设置接收回调函数 */
    rt_device_set_rx_indicate(serial, uart_rx_ind);
    /* 发送字符串 */
    //rt_device_write(serial, 0, str, (sizeof(str) - 1));
    /* 创建 serial 线程 */
    rt_thread_t thread = rt_thread_create("serial", (void (*)(void *parameter))data_parsing,
RT_NULL, 2048, 5, 5);

    /* 创建成功则启动线程 */
    if (thread != RT_NULL) {
        rt_thread_startup(thread);
    } else {
        ret = RT_ERROR;
    }

    return ret;
}

/* 导出到 msh 命令列表中 */
MSH_CMD_EXPORT(uart_init, uart device sample);

#define THREAD_PRIORITY      9
#define THREAD_TIMESLICE     5
#define EVENT_FLAG (1 << 3)
/* 事件控制块 */
static struct rt_event event;

ALIGN(RT_ALIGN_SIZE)
/* 线程 1 入口函数 */
static void thread1_recv_event(void *param) {
    rt_uint32_t e;

    while(1) {
        /* 第一次接收事件，事件 3 或事件 5 任意一个可以触发线程 1，接收完后清除事件标志 */
        if (rt_event_recv(&event, (EVENT_FLAG ),
                          RT_EVENT_FLAG_OR | RT_EVENT_FLAG_CLEAR,
                          RT_WAITING_FOREVER, &e) == RT_EOK) {
            rt_kprintf("thread1: recv event 0x%x\n", e);
            analyticald_data();
            rt_kprintf("thread1: delay 1s to prepare the second event\n");
        }

        rt_thread_mdelay(100);
    }
}

ALIGN(RT_ALIGN_SIZE)
/* 线程 2 入口 */
static void thread2_send_event(void *param) {
    while(1) {
        if(flag==1) {
            flag = 0;
            rt_kprintf("thread2: send event\n");
            rt_event_send(&event, EVENT_FLAG);
        }

        rt_thread_mdelay(200);
    }
}

int event_wavplayer(void) {
    rt_err_t result;

    /* 初始化事件对象 */
    result = rt_event_init(&event, "event", RT_IPC_FLAG_FIFO);
```

```
    if (result != RT_EOK) {
        rt_kprintf("init event failed.\n");
        return -1;
    }

    rt_thread_t thread1 = rt_thread_create("serial", thread1_recv_event, RT_NULL, 512, 10, 5);
    rt_thread_startup(thread1);
    rt_thread_t thread2 = rt_thread_create("serial", thread2_send_event, RT_NULL, 512, 9, 5);
    rt_thread_startup(thread2);

    return 0;
}

/* 导出到 msh 命令列表中 */
//MSH_CMD_EXPORT(event_wavplayer, event sample);
INIT_APP_EXPORT(event_wavplayer);
```

The code is relatively simple. There are two parts, one is the operation of the serial port, and the other is the analysis event of the serial port data. When the serial port receives the command, event_wavplayer parses the serial port command and operates the audio device according to the corresponding command. The serial port instructions are as follows:

Command	Description
0x01	Music switching
0x02	Volume adjustment

3.4. Buttons to control audio equipment

The MultiButton package is used here, and the code is as follows:

```
#include "key_app.h"
#include "led_app.h"

extern uint32_t pulse_pulse;
#define BUTTON_PIN_0 rt_pin_get("PF.0")
#define BUTTON_PIN_1 rt_pin_get("PF.1")
static struct button btn_0;
static struct button btn_1;
uint32_t cnt_channels = 1;
uint32_t cnt_volume = 1;
uint32_t cnt_music = 0;
uint32_t  start_flag = 1;

char *table[NUM_OF_SONGS] =
{
    "/Try.wav",
    "/Bad.wav",
};

static uint8_t button_read_pin_0(void) {
    return rt_pin_read(BUTTON_PIN_0);
}

static uint8_t button_read_pin_1(void) {
    return rt_pin_read(BUTTON_PIN_1);
}

static void button_0_callback(void *btn) {
    uint32_t btn_event_val;
    btn_event_val = get_button_event((struct button *) btn);

    switch(btn_event_val) {
    case SINGLE_CLICK:
        if (cnt_volume < 11) {
            if (start_flag) {
                start_flag = 0;
                cnt_volume = (int)saia_volume_get()/10;
                pulse_pulse = 9000;
            } else {
                saia_volume_set(cnt_volume * 10);
                pulse_pulse = cnt_volume*9000;
```

```
            }
        } else {
            saia_volume_set(10);
            cnt_volume = 1;
            rt_kprintf("The volume has been adjusted to maximum\n");
        }

        cnt_volume ++;
        rt_kprintf("vol=%d\n", saia_volume_get());
        rt_kprintf("button 0 single click\n");
        break;
    case DOUBLE_CLICK:
        if (cnt_channels < 3) {
            saia_channels_set(cnt_channels);
        } else {
            saia_channels_set(cnt_channels);
            cnt_channels = 1;
        }

        cnt_channels++;
        rt_kprintf("button 0 double click\n");
        break;
    case LONG_PRESS_START:
        rt_kprintf("button 0 long press start\n");
        break;
    case LONG_PRESS_HOLD:
        rt_kprintf("button 0 long press hold\n");
        break;
    }
}

static void button_1_callback(void *btn) {
    uint32_t btn_event_val;
    btn_event_val = get_button_event((struct button *)btn);

    switch (btn_event_val) {
    case SINGLE_CLICK:
        wavplayer_play(table[(cnt_music++) % NUM_OF_SONGS]);
        rt_kprintf("button 1 single click\n");
        break;
    case DOUBLE_CLICK:
        rt_kprintf("button 1 double click\n");
        break;
    case LONG_PRESS_START:
        rt_kprintf("button 1 long press start\n");
        break;
    case LONG_PRESS_HOLD:
        rt_kprintf("button 1 long press hold\n");
        break;
    }
}

static void btn_thread_entry(void* p) {
    while (1) {
        /* 5ms */
        rt_thread_delay(RT_TICK_PER_SECOND/200);
        button_ticks();
    }
}

static int multi_button_wavplayer(void) {
    rt_thread_t thread = RT_NULL;
    /* Create background ticks thread */
    thread = rt_thread_create("btn", btn_thread_entry, RT_NULL, 512, 10, 10);

    if (thread == RT_NULL) {
        return RT_ERROR;
    }

    rt_thread_startup(thread);
    /* low level drive */
    rt_pin_mode  (BUTTON_PIN_0, PIN_MODE_INPUT_PULLUP);
    button_init  (&btn_0, button_read_pin_0, PIN_LOW);
    button_attach(&btn_0, SINGLE_CLICK,     button_0_callback);
    button_attach(&btn_0, DOUBLE_CLICK,     button_0_callback);
    button_attach(&btn_0, LONG_PRESS_START, button_0_callback);
    button_attach(&btn_0, LONG_PRESS_HOLD,  button_0_callback);
    button_start (&btn_0);
```

```
    rt_pin_mode  (BUTTON_PIN_1, PIN_MODE_INPUT_PULLUP);
    button_init  (&btn_1, button_read_pin_1, PIN_LOW);
    button_attach(&btn_1, SINGLE_CLICK,     button_1_callback);
    button_attach(&btn_1, DOUBLE_CLICK,     button_1_callback);
    button_attach(&btn_1, LONG_PRESS_START, button_1_callback);
    button_attach(&btn_1, LONG_PRESS_HOLD,  button_1_callback);
    button_start (&btn_1);

    return RT_EOK;
}

MSH_CMD_EXPORT(multi_button_wavplayer, button wavplayer)
```

The button control is similar to the serial port control, but the buttons are limited, and the control content is relatively less than the serial port.

There is nothing to talk about SD devices. Please note that if there is no automatic mounting device, you need to manually mount the SD card device. The code for automatic mounting is as follows:

```
#include <rtthread.h>

#ifdef BSP_USING_SDIO

#include <dfs_elm.h>
#include <dfs_fs.h>
#include <dfs_posix.h>
#include "drv_gpio.h"
// #define DRV_DEBUG
#define DBG_TAG "app.card"
#include <rtdbg.h>

void sd_mount(void *parameter) {
    while (1) {
        rt_thread_mdelay(500);

        if (rt_device_find("sd0") != RT_NULL) {
            if (dfs_mount("sd0", "/", "elm", 0, 0) == RT_EOK) {
                LOG_I("sd card mount to '/'");
                break;
            } else {
                LOG_W("sd card mount to '/' failed!");
            }
        }
    }
}

int ab32_sdcard_mount(void) {
    rt_thread_t tid;
    tid = rt_thread_create(
        "sd_mount",
        sd_mount,
        RT_NULL,
        1024,
        RT_THREAD_PRIORITY_MAX - 2,
        20
    );

    if (tid != RT_NULL) {
        rt_thread_startup(tid);
    } else {
        LOG_E("create sd_mount thread err!");
    }

    return RT_EOK;
}

INIT_APP_EXPORT(ab32_sdcard_mount);

#endif
```

Well, that's all for the implementation code of the music player, please follow the instructions below to get the complete code.

4. Function demonstration

First of all, ordinary music files need to be converted into wav format. The recommended software is GoldWave. After the conversion is complete, put the music into the SD card, insert the SD card into the board, and then demonstrate the music playback.

Please note that the SD card can only be used by connecting J6, J7, J9, and J11 with jumper caps.

5. Summary

The music player in this article is just an introduction, and there are still many areas for improvement. The areas that need to be improved are as follows:

1. Improved playback mode, such as sequential playback, random playback, and single loop;
2. The control method is improved, and wireless devices can be connected externally. Of course, the onboard Bluetooth can also be used to control music playback;
3. To obtain music files, if there is a network, the music on the network can also be played, so that the music played can be diversified.

Generally speaking, the application of the music player is relatively simple. At that time, the audio driver, SD card driver and the underlying logic of the file system were still relatively complicated. Those who are interested can study in depth.

I2C OLED display with SSD1306 controller

Original post: https://mp.weixin.qq.com/s/bSFhKHAZ0VJ8C8yppcgLVA

Create a new RT-Thread Studio project and add components:

Configure the details again:

The following configuration should be the default and has no effect:

The sample code needs to be modified a bit:

ssd1306_tests.c

```
#include <rtthread.h>
#include <rtdevice.h>
#include <board.h>

#include <string.h>
#include <stdio.h>
#include "ssd1306.h"
#include "ssd1306_tests.h"

void ssd1306_TestBorder() {
    ssd1306_Fill(Black);

    //uint32_t start = HAL_GetTick();   //需替换下面一句
    uint32_t start = rt_tick_get();
    uint32_t end = start;
    uint8_t x = 0;
    uint8_t y = 0;
    do {
        ssd1306_DrawPixel(x, y, Black);

        if((y == 0) && (x < 127))
            x++;
        else if((x == 127) && (y < 63))
            y++;
        else if((y == 63) && (x > 0))
            x--;
        else
            y--;

        ssd1306_DrawPixel(x, y, White);
        ssd1306_UpdateScreen();

        //HAL_Delay(5);
        rt_thread_mdelay(5);
        //end = HAL_GetTick();
        end = rt_tick_get();
    } while((end - start) < 8000);

    //HAL_Delay(1000);   //需替换下面一句
    rt_thread_mdelay(5000);
}

void ssd1306_TestFonts() {
    ssd1306_Fill(Black);
    ssd1306_SetCursor(2, 0);
    ssd1306_WriteString("Font 16x26", Font_16x26, White);
    ssd1306_SetCursor(2, 26);
    ssd1306_WriteString("Font 11x18", Font_11x18, White);
    ssd1306_SetCursor(2, 26+18);
    ssd1306_WriteString("Font 7x10", Font_7x10, White);
    ssd1306_SetCursor(2, 26+18+10);
    ssd1306_WriteString("Font 6x8", Font_6x8, White);
    ssd1306_UpdateScreen();
}
```

```
void ssd1306_TestFPS() {
    ssd1306_Fill(White);

    //uint32_t start = HAL_GetTick();   //需替换下面一句
    uint32_t start = rt_tick_get();

    uint32_t end = start;
    int fps = 0;
    char message[] = "ABCDEFGHIJK";

    ssd1306_SetCursor(2,0);
    ssd1306_WriteString("Testing...", Font_11x18, Black);

    do {
        ssd1306_SetCursor(2, 18);
        ssd1306_WriteString(message, Font_11x18, Black);
        ssd1306_UpdateScreen();

        char ch = message[0];
        memmove(message, message+1, sizeof(message)-2);
        message[sizeof(message)-2] = ch;

        fps++;
        //end = HAL_GetTick();   //需替换下面一句
        end = rt_tick_get();
    } while((end - start) < 5000);

    //HAL_Delay(1000);
    rt_thread_mdelay(1000);

    char buff[64];
    fps = (float)fps / ((end - start) / 1000.0);
    snprintf(buff, sizeof(buff), "~%d FPS", fps);

    ssd1306_Fill(White);
    ssd1306_SetCursor(2, 18);
    ssd1306_WriteString(buff, Font_11x18, Black);
    ssd1306_UpdateScreen();
}

void ssd1306_TestLine() {
    ssd1306_Line(1,1,SSD1306_WIDTH - 1,SSD1306_HEIGHT - 1,White);
    ssd1306_Line(SSD1306_WIDTH - 1,1,1,SSD1306_HEIGHT - 1,White);
    ssd1306_UpdateScreen();
    return;
}

void ssd1306_TestRectangle() {
    uint32_t delta;

    for(delta = 0; delta < 5; delta ++) {
        ssd1306_DrawRectangle(1 + (5*delta),1 + (5*delta) ,SSD1306_WIDTH-1 -
(5*delta),SSD1306_HEIGHT-1 - (5*delta),White);
    }
    ssd1306_UpdateScreen();
    return;
}

void ssd1306_TestCircle() {
    uint32_t delta;

    for(delta = 0; delta < 5; delta ++) {
        ssd1306_DrawCircle(20* delta+30, 30, 10, White);
    }
    ssd1306_UpdateScreen();
    return;
}

void ssd1306_TestArc() {
    ssd1306_DrawArc(30, 30, 30, 20, 270, White);
    ssd1306_UpdateScreen();
    return;
}

void ssd1306_TestPolyline() {
    SSD1306_VERTEX loc_vertex[] = {
        {35,40},
```

```
        {40,20},
        {45,28},
        {50,10},
        {45,16},
        {50,10},
        {53,16}
    };

    ssd1306_Polyline(loc_vertex,sizeof(loc_vertex)/sizeof(loc_vertex[0]),White);
    ssd1306_UpdateScreen();
    return;
}

void ssd1306_TestAll() {
    ssd1306_Init();

    ssd1306_TestFPS();
    //HAL_Delay(3000);   //需替换下面一句
    rt_thread_mdelay(3000);

    ssd1306_TestBorder();

    ssd1306_TestFonts();
    //HAL_Delay(3000);   //需替换下面一句
    rt_thread_mdelay(3000);

    ssd1306_Fill(Black);
    ssd1306_TestRectangle();
    ssd1306_TestLine();
    //HAL_Delay(3000);   //需替换下面一句
    rt_thread_mdelay(3000);

    ssd1306_Fill(Black);
    ssd1306_TestPolyline();
    //HAL_Delay(3000);   //需替换下面一句
    rt_thread_mdelay(3000);

    ssd1306_Fill(Black);
    ssd1306_TestArc();
    //HAL_Delay(3000);   //需替换下面一句
    rt_thread_mdelay(3000);

    ssd1306_Fill(Black);
    ssd1306_TestCircle();
    //HAL_Delay(3000);   //需替换下面一句
    rt_thread_mdelay(3000);
}

#ifdef FINSH_USING_MSH
MSH_CMD_EXPORT(ssd1306_TestAll, test ssd1306 oled driver);
#endif
```

To compile and flash, please refer to **Building an application and using the development environment**.

Command line execution:

See video.

BLE development

Original post: https://club.rt-thread.org/ask/article/cbbf259f0fa863dc.html

Copyright: fannifu License: CC-BY-SA 4.0

Install a recent SDK

Make sure to install the latest SDK for the AB32VG1 development board, that is bluetrum_sdk version 1.1.0 or higher.

Build an RT-Thread project

Configure the project

Turn on:

Macro: [SOC_AB32VG1]

Macro: [PKG_USING_BLUETRUM_NIMBLE]

Macro: [BSP_USING_TIM]

I added the latter afterwards because there is an error when reporting data regularly: rt_object_get_type(&timer->parent) == RT_Object_Class_Timer) assertion failed at function:rt_timer_stop, line number:502. I tried adding the TIMER module, but no luck.

After configuration, there will be an error when saving: local variable 'get_git_root' referenced before assignment Operation failed. We'll ignore it for now, we'll modify it later.

Modify the code

First familiarize yourself with the BLE_SDK code directory:

Modify mac_addr

Under the project directory, open packages\bluetrum_sdk-latest\bluetrum_nimble\porting\npl\rtthread\src\ble_platform.c

I modified it as follows:

```
void bt_get_local_bd_addr(uint8_t *addr) {
    // Note: mac_addr is 6 bytes, but only 5 data are written in the SDK code.
    // Is this an omission or intentional?
    uint8_t addr_table[6] = {0x41, 0x42, 0x18, 0x58, 0x68, 0x88};
    memcpy(addr, addr_table, 6);
}
```

Modify BLE_name

Under the project directory, open packages\bluetrum_sdk-latest\bluetrum_nimble\apps\blehr\src\blehr.c

```
// Note: the name cannot contain Chinese characters, or the device will not be found.
static const char *device_name = "Silicon_blehr_sensor";
```

Modify service and device information

Under the project directory, open packages\bluetrum_sdk-latest\bluetrum_nimble\apps\blehr\src

I modified it as follows:

```
// manuf_name can contain Chinese characters, but its length is limited.
// Unfortunately, the limit is unspecified.
static const char *manuf_name = "谷动谷力";
static const char *model_num = "SunSilicon HR Sensor";
```

and:

```
static const struct ble_gatt_svc_def gatt_svr_svcs[] = {
    {
```

```
        /* Service: Heart-rate */
        .type = BLE_GATT_SVC_TYPE_PRIMARY,
        .uuid = BLE_UUID16_DECLARE(GATT_HRS_UUID),
        .characteristics = (struct ble_gatt_chr_def[]) {
            {
                /* Characteristic: Heart-rate measurement */
                .uuid = BLE_UUID16_DECLARE(GATT_HRS_MEASUREMENT_UUID),
                .access_cb = gatt_svr_chr_access_heart_rate,
                .val_handle = &hrs_hrm_handle,
                .flags = BLE_GATT_CHR_F_NOTIFY,
            },
            {
                /* Characteristic: Body sensor location */
                .uuid = BLE_UUID16_DECLARE(GATT_HRS_BODY_SENSOR_LOC_UUID),
                .access_cb = gatt_svr_chr_access_heart_rate,
                .flags = BLE_GATT_CHR_F_READ,
            },
            {
                0, /* No more characteristics in this service */
            },
        }
    },
    {
        /* Service: Device Information */
        .type = BLE_GATT_SVC_TYPE_PRIMARY,
        .uuid = BLE_UUID16_DECLARE(GATT_DEVICE_INFO_UUID),
        .characteristics = (struct ble_gatt_chr_def[]) {
            {
                /* Characteristic: * Manufacturer name */
                .uuid = BLE_UUID16_DECLARE(GATT_MANUFACTURER_NAME_UUID),
                .access_cb = gatt_svr_chr_access_device_info,
                .flags = BLE_GATT_CHR_F_READ,
            },
            {
                /* Characteristic: Model number string */
                .uuid = BLE_UUID16_DECLARE(GATT_MODEL_NUMBER_UUID),
                .access_cb = gatt_svr_chr_access_device_info,
                .flags = BLE_GATT_CHR_F_READ,
            },
            {
                0, /* No more characteristics in this service */
            },
        }
    },
    {
        0, /* No more services */
    },
};
```

Modify service properties

For the service here, we have kept the default.

Compile

Compile the project: Ctrl+B, there will be an error, as follows:

```
In file included from :
E:\RT-ThreadWorkspace\AB32_BLE_Demo\rtconfig_preinc.h89: error: no macro name given in #define
directive
```

```
8 | #define
| ^
In file included from :
E:\RT-ThreadWorkspace\AB32_BLE_Demo\rtconfig_preinc.h89: error: no macro name given in #define
directive
8 | #define
| ^
make: [rt-thread/src/subdir.mk57 rt-thread/src/device.o] Error 1
make: Waiting for unfinished jobs….
In file included from :
E:\RT-ThreadWorkspace\AB32_BLE_Demo\rtconfig_preinc.h89: error: no macro name given in #define
directive
8 | #define
| ^
In file included from :
E:\RT-ThreadWorkspace\AB32_BLE_Demo\rtconfig_preinc.h89: error: no macro name given in #define
directive
8 | #define
| ^
make: [rt-thread/src/subdir.mk57 rt-thread/src/clock.o] Error 1
make: [rt-thread/src/subdir.mk57 rt-thread/src/components.o] Error 1
make: * [rt-thread/src/subdir.mk57 rt-thread/src/idle.o] Error 1
“make -j4 all2” terminated with exit code 2. Build might be incomplete.
Build Failed. 9 errors, 0 warnings. (took 1s.464ms)
```

Double-click the error prompt to locate the error, the #define is empty, just delete it. Save the modified content: ctrl+s, recompile: ctrl+b.

Build is now successful.

Debugging

Burn the compiled target file: rtthread.elf to the AB32VG1 development board.

Turn on Bluetooth on the mobile phone:

Open the Bluetooth BLE debugging assistant on the mobile phone:

Scan for BLE devices

You should see the device name you modified in the code:

List of device-related data services

Get BLE device data – Manufacturer information

Just click "Read", the dialogue box show "谷动谷力", as we entered in our code:

Obtain BLE device data – Device model information

Same as above, click "Read" to display the model name:

Read heart rate related data

Automatic data notification reporting of heart rate data

Heart rate data is simulated, not on the development board hardware.

10:34
84%
Silicon_blehr_sensor
连接
Services
Log
Generic Access
UUID: 00001800-0000-1000-8000-00805f9b34fb
PRIMARY SERVICE
接收数据
实时显示
保存至文件
接收:92 Byte
HEX
实时速度:0 B/s
接收通知数据
00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00
关闭
清空
Properties:READ
Device Information
UUID: 0000180a-0000-1000-8000-00805f9b34fb
PRIMARY SERVICE

Downloader v2.2.0
工具(T)
帮助(H)
Language
置顶
串口
USB
配置
开始
开发
DownFile
E:\RT-ThreadWorkspace\AB32_BLE\Debug\rtthread.dcf
暂停
滚动
全选
复制
保存
格式
信息
擦除
9940
清空
ACL => 00 00 09 00 05 00 04 00 1b 08 00
EVT <= 13 05 01 00 00 01 00
ACL => 00 00 09 00 05 00 04 00 1b 08 00
EVT <= 13 05 01 00 00 01 00
ACL => 00 00 09 00 05 00 04 00 1b 08 00
EVT <= 13 05 01 00 00 01 00
ACL => 00 00 09 00 05 00 04 00 1b 08 00
EVT <= 13 05 01 00 00 01 00
ACL => 00 00 09 00 05 00 04 00 1b 08 00
EVT <= 13 05 01 00 00 01 00
ACL => 00 00 09 00 05 00 04 00 1b 08 00
EVT <= 13 05 01 00 00 01 00
ACL => 00 00 09 00 05 00 04 00 1b 08 00
EVT <= 13 05 01 00 00 01 00
ACL => 00 00 09 00 05 00 04 00 1b 08 00
EVT <= 13 05 01 00 00 01 00
ACL => 00 00 09 00 05 00 04 00 1b 08 00
EVT <= 13 05 01 00 00 01 00
(rt_object_get_type(&timer->parent) == RT_Object_Class_Timer) assertion failed at
function:rt_timer_stop, line number:502
完成
COM 已关闭
下载
自动

```
EVT <= 13 05 01 00 00 01 00
ACL <= 00 20 0b 00 07 00 04 00 08 10 00 ff ff 03 28
ACL => 00 00 09 00 05 00 04 00 01 08 10 00 0a
EVT <= 13 05 01 00 00 01 00
ACL <= 00 20 09 00 05 00 04 00 04 11 00 ff ff
ACL => 00 00 09 00 05 00 04 00 01 04 11 00 0a
EVT <= 13 05 01 00 00 01 00
EVT <= 3e 0a 03 00 00 00 0c 00 00 00 d0 07
ACL <= 00 20 09 00 05 00 04 00 12 09 00 01 00
ACL => 00 00 05 00 01 00 04 00 13
EVT <= 13 05 01 00 00 01 00
ACL => 00 00 09 00 05 00 04 00 1b 08 00
EVT <= 13 05 01 00 00 01 00
ACL => 00 00 09 00 05 00 04 00 1b 08 00
EVT <= 13 05 01 00 00 01 00
ACL => 00 00 09 00 05 00 04 00 1b 08 00
EVT <= 13 05 01 00 00 01 00
ACL => 00 00 09 00 05 00 04 00 1b 08 00
EVT <= 13 05 01 00 00 01 00
thread:絹x02 stack overflow
```

This data point code has a bug. It is not certain when it will report an error, and the error is different every time.

Remaining issues

1. There is a bug in the code of the data point reported regularly. The error is reported at an uncertain time, and is different every time. This has to be analysed officially.
2. For now, only the Read (BLE report data) function is available, not the Write (APP->BLE data release) function, this will be ported later.
3. Connect existing common platform functions, such as WeChat\Tuya\Tmall Genie...etc.
4. mac_addr is 6 bytes and only 5 data are written in the SDK's code. Is this an omission or intentional?
5. The length of the manufacturer name is limited. I can’t find where the limit is. Is there a limit to the length of the data in the report? Where is the code?

Summary

RT-Thread makes it is fast and convenient to develop basic BLE code for the Bluetrum AB32VG1. It is more complicated to make an official driver and software that is convenient and fast.