eBox 标准 API 手册

打造 eBox 生态圈

版本：版本 16.4.19

发布日期：2016 年 2 月

本指南内容及产如有更新，请参考最新手

勘误记录

日期	摘要	提交者
2018.12.10	添加 Stream 类说明	好心情
2019.01.02	添加 ADC 类说明	好心情

如何使用本 API 手册

本 API 手册只提供了公共基础函数和类中 public 函数的解释。并没有对程序中所有的变量和宏作出详细解释。如果涉及到相关宏定义的内容请阅读程序代码中的注释。

本手册更新可能没有程序更新的快，部分内容可能有所书写错误，最终以头文件中的内容为准。

本手册只作为引导性学习使用，请多读程序头文件已获得更全面的信息。

完整的应用示例可参考 example 文件夹下的内容

如发现有文字性错误或者部分代码错误请联系 995207301@qq.com。

目录

目录.....1
第 1 章 公共基础接口.....1
1.1 ebox_init(void);.....1
1.2 millis(void) ;.....2
1.3 delay_ms(uint64_t ms); 2
1.4 delay_us(uint64_t us);.....3
1.5 shift_out(GPIO* data_pin, GPIO* clock_pin,.....3
1.6 shift_in(GPIO* data_pin, GPIO* clock_pin, uint8_t bit_order) 4
1.7 uint32_t chip_id[3]; 4
1.8 uint16_t flash_size; 4
第 2 章 GPIO 类 5
2.1 GPIO(GPIO_TypeDef *port, uint16_t pin); 5
2.2 mode();.....6
2.3 set(); 6
2.4 reset(); 6
2.5 write();.....7
2.6 toggle();.....7
2.7 read(uint8_t* val);.....8
2.8 read(void);.....8
第 3 章 ADC 类.....9
3.1 add_ch(Gpio *io) 9
3.2 add_temp_senser() 9
3.3 begin() 9
第 4 章 PARALLEL_GPIO 类 10
4.1 GPIO *bit[8]; 10
4.2 mode(PIN_MODE mode);.....10
4.3 write(uint8_t data);.....11

4.4 read(void);.....11
第 5 章 PWM 类.....13
5.1 PWM(GPIO* pwm_pin);.....13
5.2 begin(uint32_t frq,uint16_t duty); 14
5.3 set_frq(uint32_t frq);.....14
5.4 set_duty(uint16_t duty);.....15
5.5 set_oc_polarity(uint8_t flag);.....15
第 6 章 IN_CAPTURE 类.....16
6.1 IN_CAPTURE(GPIO *capture_pin); 16
6.2 begin(uint16_t prescaler); 17
6.3 set_count(uint16_t count); 17
6.4 set_polarity_falling(); 18
6.5 set_polarity_rising();.....18
6.6 overflow_event_process();.....19
6.7 get_capture(); 19
6.8 get_overflow_state(); 20
6.9 attch_ic_interrupt(void(*callback)(void)); 20
6.10 attch_update_interrupt(void(*callback)(void));.....21
第 7 章 EXTI 类 22
7.1 EXTIx(GPIO *EXTI_pin,EXTITrigger_TypeDef trigger);.....22
7.2 begin();.....23
7.3 attach_interrupt(void (*callbackFun)(void)); 23
7.4 interrupt(FunctionalState enable); 23
第 8 章 USART 类.....25
8.1 USART(USART_TypeDef * USARTx,.....25
8.2 begin(uint32_t baud_rate);.....26
8.3 begin(uint32_t baud_rate,uint8_t data_bit,uint8_t parity,float stop_bit); 26
8.4 attach_rx_interrupt(void (*callback_fun)(void));.....27

8.5 attach_tx_interrupt(void (*callback_fun)(void)); 27
8.6 receive(); 28
8.7 put_char(char ch); 28
8.8 put_string(const char *str); 29
8.9 printf(const char* fmt,...); 29
8.10 printf_length(const char *str,uint16_t length); 29
8.11 wait_busy() 30
第 9 章 GTIMER 类 31
9.1 TIM(TIM_TypeDef* TIMx); 31
9.2 begin(uint32_t frq); 32
9.3 attach_interrupt(void(*callback)(void)); 32
9.4 interrupt(FunctionalState enable); 32
9.5 start(void); 33
9.6 stop(void); 33
9.7 void reset_frq(uint32_t frq); 34
9.8 set_reload(uint16_t autoreload); 34
9.9 clear_count(void); 34
第 10 章 TIMERONE 类 36
第 11 章 SPI 类 37
11.1 SPI(SPI_TypeDef *SPIx,GPIO *sck,GPIO *miso,GPIO *mosi); 37
11.2 begin (SPI_CONFIG_TYPE* spi_config); 38
11.3 config(SPI_CONFIG_TYPE* spi_config); 38
11.4 read_config(void); 39
11.5 write(uint8_t data); 39
11.6 write(uint8_t *data,uint16_t data_length); 39
11.7 read(); 40
11.8 read(uint8_t* recv_data); 40
11.9 read(uint8_t *recv_data,uint16_t datalength); 41
11.10 take_spi_right(SPI_CONFIG_TYPE* spi_config); 41

11.11 release_spi_right(void); 41
第 12 章 SOFTSPI 类....42
12.1 SOFTSPI(GPIO* sck,GPIO* miso,GPIO* mosi); 43
12.2 begin(SPI_CONFIG_TYPE* spi_config); 44
12.3 config(SPI_CONFIG_TYPE* spi_config); 44
12.4 read_config(void);....44
12.5 write(uint8_t data);....44
12.6 write(uint8_t *data,uint16_t dataln); 44
12.7 read();....44
12.8 read(uint8_t* data);....44
12.9 read(uint8_t *rcvdata,uint16_t dataln);....44
12.10 take_spi_right(SPI_CONFIG_TYPE* spi_config); 44
12.11 release_spi_right(void); 44
第 13 章 I2C 类....45
13.1 I2C(I2C_TypeDef* I2Cx,GPIO* scl_pin,GPIO* sda_pin); 46
13.2 begin(uint32_t speed); 46
13.3 config(uint32_t speed); 47
13.4 read_config();....47
13.5 write_byte(uint8_t slave_address,....47
13.6 write_byte(uint8_t slave_address,....48
13.7 read_byte (uint8_t slave_address,....48
13.8 read_byte (uint8_t slave_address,....49
13.9 wait_dev_busy(uint8_t slave_address);....50
13.10 take_i2c_right(uint32_t speed); 50
13.11 release_i2c_right(void); 50
第 14 章 SOFTI2C 类....52
14.1 SOFTI2C(GPIO* scl, GPIO* sda); 53
14.2 begin(uint32_t speed); 53
14.3 config(uint32_t speed); 54

14.4 其他公有成员函数.....54
第 15 章 RTC 类.....55
15.1 begin();.....56
15.2 attach_interrupt(uint16_t event,.....56
15.3 interrupt(uint32_t bits,FunctionalState x);.....57
15.4 set_counter(uint32_t count);.....57
15.5 set_alarm(uint32_t count);.....58
15.6 get_counter();.....58
15.7 set_time_HMS(uint8_t h,uint8_t m,uint8_t s);.....58
15.8 set_alarm(uint8_t h,uint8_t m,uint8_t s);.....59
15.9 get_time_HMS(uint8_t* h,uint8_t* m,uint8_t* s);.....59
15.10 RTC 下的 HHMMSS 时钟变量.....60
第 16 章 FLASHCLASS 类.....61
16.1 read(uint32_t iAddress, uint8_t *buf, int32_t iNbrToRead) ;.....61
16.2 write(uint32_t iAddress, uint8_t *buf, uint32_t iNbrToWrite);.....62
第 17 章 EBOX_IWDG 类.....62
17.1 EBOX_IWDG();.....62
17.2 begin(uint16_t ms);.....63
17.3 feed();.....63
第 18 章 Stream 类.....64
18.1 available(void).....64
18.2 read(void).....64
18.3 peak(void).....64
18.4 findUntil(char *target, size_t targetLen, char *terminate, size_t termLen); 65
18.5 find(char *target);.....65
18.6 find(char *target, size_t length);.....65
18.7 find(char target)；.....65
18.8 findUntil(char *target, char *terminator);.....65

18.9 parseInt(LookaheadMode lookahead = SKIP_ALL, char ignore = NO_IGNORE_CHAR); 65

18.10 parseFloat(LookaheadMode lookahead = SKIP_ALL, char ignore = NO_IGNORE_CHAR); 66

18.11 readBytes(char *buffer, size_t length); 66

18.12 readBytesUntil(char terminator, char *buffer, size_t length); 66

第1章 公共基础接口

```
ebox_init(void);
millis( void ) ;
delay_ms(uint64_t ms);
delay_us(uint64_t us);
analog_read(GPIO* pin);
analog_read_voltage(GPIO* pin);
analog_write(GPIO* pwm_pin, uint16_t duty)
shift_out(GPIO* data_pin, GPIO* clock_pin, uint8_t bit_order, uint8_t val);
shift_in(GPIO* data_pin, GPIO* clock_pin, uint8_t bit_order);
//////////////////宏定义//////////////////////////
#define true          0x1
#define false         0x0
#define HIGH          0x1
#define LOW           0x0
#define LSB_FIRST     0
#define MSB_FIRST     1
#define PI            3.1415926535898
#define NVIC_GROUP_CONFIG    NVIC_PriorityGroup_2
#define interrupts()          __enable_irq()
#define no_interrupts()       __disable_irq()
```

MCU 信息类

```
class SYSTEM
{
      public:
            uint32_t chip_id[3];
            uint16_t flash_size;

            void    get_chip_info();
            float get_cpu_temperature();
};
```

1.1 ebox_init(void);

- **描述：**

初始化 eBox 运行需要的基础条件，包括 ADC1，systicks，NVIC 分组，PB4 默认功能为 GPIO。

- **参数：**
- **返回值：**
- **示例：**

1.2 millis(void) ;

- **描述：**

返回 milli_seconds 当前值，milli_seconds 从 eBox 程序一开始运行便开始自动的每一毫秒加 1，milli_seconds 计数到 2^64 后会溢出（变成 0）。

- **参数：**

空

- **返回值：**

返回 milli_seconds 当前值。

- **示例：**

```
uint64_t time;
time = millis();
```

1.3 delay_ms(uint64_t ms);

- **描述：**

精确的毫秒延时函数，使用滴答时钟完成的硬件计时。相当精确。

- **参数：**

ms:延时的时间。最大值 2^64。

- **返回值：**
- **返回值：**
- **示例：**

```
delay_ms(1000);
```

1.4 delay_us(uint64_t us);

- **描述：**

  精确的微秒延时函数，使用滴答时钟完成的硬件计时。相当精确。

- **参数：**

  us:延时的时间。最大值 2^64。

- **返回值：**

空

- **示例：**

  delay_us(1);

1.5 shift_out(GPIO* data_pin, GPIO* clock_pin, uint8_t bit_order, uint8_t val)

- **描述：**

将一个 8 位的数据按 bit 顺序输出到 IO 引脚上。

- **参数：**

data_pin：GPIO 的指针或地址

clock_pin：GPIO 的指针或地址

bit_order：是高位先输出还是低位先输出。

Val：输出的值

- **返回值：**

空

- **示例：**

shift_out(PA1，PA2，LSBFIRST,0X5A);

1.6 shift_in(GPIO* data_pin, GPIO* clock_pin, uint8_t bit_order)

- **描述：**

读取一个 IO 上的 8 位的数据按 bit 顺序组合成一个 8 位数据

- **参数：**

data_pin：GPIO 的指针或地址

clock_pin：GPIO 的指针或地址

bit_order：是高位先输出还是低位先输出。

- **返回值：**

data_pin 引脚读入的数据。

- **示例：**

```
uin8_t val;
val= shift_in(PA1，PA2，LSBFIRST);
```

1.7 uint32_t chip_id[3];

- **描述：**

STM32 的芯片 ID；在 eBox 的初始化函数中已经被更新完成，用户可以直接使用。

1.8 uint16_t flash_size;

- **描述：**

STM32 芯片的 flash 容量；在 eBox 的初始化函数中已经被更新完成，用户可以直接使用。

第2章 GPIO 类

```
class GPIO
{
    public:
        GPIO(GPIO_TypeDef *port, uint16_t pin);
        void mode(PIN_MODE mode);
        void set();
        void reset();
        void write(uint8_t val);
        void toggle();
        void read(uint8_t *val);
        uint8_t read(void);

        GPIO_TypeDef* port;
        uint16_t pin;
};
```

2.1 GPIO(GPIO_TypeDef *port, uint16_t pin);

- **描述：**

GPIO 的构造函数，实例化一个 GPIO 引脚（创建 GPIO 对象）。

- **参数：**

port：GPIO 所在的 port，如 GPIOA；

pin：GPIO 引脚的掩码，如 GPIO_Pin_1；

- **返回值：**

空

- **示例：**

```
GPIO  PA1(GPIOA,GPIO_Pin_1);
#define     PPA1        ((GPIO*)&GPA1)
GPIO        *PPA1   =   new   GPIO(GPIOA,GPIO_Pin_1);
```

2.2 mode();

- **描述：**

GPIO 引脚模式设置

- **参数：**

mode：PIN_MODE 枚举变量，如 INPUT,OUTPUT,AF_PP,AF_OD 等；

- **返回值：**

- **示例：**

PA1->mode(OUTPUT);

2.3 set();

- **描述：**

GPIO 引脚输出高电平。执行效率最高。和官方固件库速度一样

- **参数：**

空

- **返回值：**

空

- **示例：**

PA1.mode(OUTPUT_PP);

PA1.set();

2.4 reset();

- **描述：**

GPIO 引脚输出低电平。执行效率最高。和官方固件库速度一样

- **参数：**

空

- **返回值：**

空

- **示例：**

PA1.mode(OUTPUT_PP);

PA1.reset();

2.5 write();

- **描述：**

GPIO 引脚输出控制。执行效率略低于 set()和 reset()。5 个指令周期。

- **参数：**

空

- **返回值：**

空

- **示例：**

PA1.mode(OUTPUT_PP);

PA1.write(1);

PA1.write(0);

PA1.write(HIGH);

PA1.write(LOW);

2.6 toggle();

- **描述：**

GPIO 引脚输出翻转。

- **参数：**

空

- **返回值：**

空

- **示例：**

```
PA1.mode(OUTPUT_PP);
PA1.toggle();
```

2.7 read(uint8_t* val);

- **描述：**

读取 GPIO 引脚输入电平。

- **参数：**

val:输出 uint8_t 指针

- **返回值：空**

- **示例：**

```
uint8_t val;
PA1.mode(INPUT);
PA1.read(&val);
```

2.8 read(void);

- **描述：**

读取 GPIO 引脚输入电平。

- **参数：空**

- **空返回值：**

val:GPIO 引脚电平

- **示例：**

```
uint8_t val;
PA1.mode(INPUT);
val = PA1.read();
```

第3章 ADC 类

ADC 类设计为 DMA 模式，软件触发，连续采样,分辨率 12bit。

如何使用 ADC 类：

1 创建对象 Adc adc(ADC1);

2 添加通道 adc.add_ch(&PA0);

adc.add_ch(&PA1);

3 启动 ADC adc.begin();

4 读取 ADC adc.read(&PA0);

adc.read_voltage(&PA0)

3.1 add_ch(Gpio *io)

- **描述：**

通过 gpio 添加 1 个 adc 通道

- **参数：**

Gpio *io GPIO 对象指针或地址，如&PA7；

- **返回值：**

空

3.2 add_temp_senser()

添加 temp 通道

- **参数：**

无

- **返回值：**

空

3.3 begin()

启动 ADC

- **参数：**

无

- **返回值：**

空

第4章 PARALLEL_GPIO 类

```
class PARALLEL_GPIO
{
    public:
        GPIO *bit[8];
    public:
        void    mode(PIN_MODE mode);
        void    write(uint8_t data);
        uint8_t read();
};
```

4.1 GPIO *bit[8];

- **描述：**

并行 IO 口指针，用于传递要使用的 1-8 位 GPIO 口

- **示例**

PARALLEL_GPIO P1;

P1.bit[0] = &PA1;

P1.bit[1]=&PB2;……

4.2 mode(PIN_MODE mode);

- **描述：**

GPIO 引脚模式设置

- **参数：**

mode：PIN_MODE 枚举变量，如 INPUT,OUTPUT,AF_PP,AF_OD 等；

- **返回值：**
- **示例：**

PARALLEL_GPIO P1;

P1.mode(OUTPUT);

4.3 write(uint8_t data);

- **描述：**

并行 GPIO 引脚输出控制。

- **参数：**

data：并行 GPIO 端口的值。

- **返回值：**

空

- **示例：**

P1.write(0xff);

P1.write(0x55);

4.4 read(void);

- **描述：**

读取 GPIO 引脚输入电平。

- **参数：**

空

- **空返回值：**

val:GPIO 引脚电平

- **示例：**

```
uint8_t val;
val = P1.read();
```

第5章 PWM 类

```
class PWM
{
    public:
        PWM(GPIO *pwm_pin);
        void begin(uint32_t frq,uint16_t duty);
        void set_frq(uint32_t frq);
        void set_duty(uint16_t duty);
        void set_oc_polarity(uint8_t flag);
    private:
        GPIO *pwm_pin;
        TIM_TypeDef *TIMx;
        uint32_t    rcc;
        uint8_t     ch;
        uint16_t    period;//保存溢出值，用于计算占空比
        uint16_t    duty;//保存占空比值
        uint16_t    oc_polarity;

        void init_info(GPIO *pwm_pin);
        void base_init(uint16_t Period,uint16_t Prescaler);
};
```

5.1 PWM(GPIO* pwm_pin);

- **描述：**

PWM 的构造函数，实例化一个 PWM（创建 •• PWM 对象）。初始化某个引脚为 PWM 输出引脚。

- **参数：**

pwm_pin：GPIO 对象指针或地址，如&PA7；

- **返回值：**

空

- **示例：**

PWM pwm(&PB8);

5.2 begin(uint32_t frq,uint16_t duty);

- **描述：**

PWM 的初始化函数，自动开启相应的定时器，并设置相应的参数。

- **参数：**

frq：当前定时器所有 pwm 引脚的频率。最小值 1（1Hz）.最大值 720K（当频率小于 72000Hz 时，pwm 精度为 0.1%；频率在 72K~720K 之间时，pwm 精度为 1%；频率再高将停止输出）；

duty：当前 pwm 引脚的占空比；

- **返回值：**

空

- **示例：**

初始化为 10K 频率，占空比为 50%的 PWM 波。

```
PWM pwm(&PB8);
pwm.begin(10000,500);
```

5.3 set_frq(uint32_t frq);

- **描述：**

重新设定 PWM 波形频率，创建 PWM 对象后可调用，只改变波形频率不改变波形占空比。

- **参数：**

frq：当前定时器所有 pwm 引脚的频率。最小值 1（1Hz）.最大值 720K（当频率小于 72000Hz 时，pwm 精度为 0.1%；频率在 72K~720K 之间时，pwm 精度为 1%；频率再高将停止输出）；

- **返回值：**

空

- **示例：**

pwm.set_frq(1000);

5.4 set_duty(uint16_t duty);

- **描述：**

重新设定 PWM 波形占空比，创建 PWM 对象后可调用，只改变波形占空比不改变波形频率。

- **参数：**

duty：当前 pwm 引脚的占空比；

- **返回值：**

空

- **示例：**

pwm.set_duty(500);

5.5 set_oc_polarity(uint8_t flag);

- **描述：**

设置 PWM 输出极性。

- **参数：**

flag：1：比较后输出高电平，0：比较后输出低电平；

- **返回值：**

空

- **示例：**

pwm1.set_oc_polarity(1);//设置比较后输出高电平

第6章 IN_CAPTURE 类

```
class IN_CAPTURE
{
    public:
        IN_CAPTURE(GPIO *capture_pin);
        void            begin(uint16_t prescaler);
        void            set_count(uint16_t count);
        void            set_polarity_falling();
        void            set_polarity_rising();
        void            overflow_event_process();
        uint32_t        get_capture();
        IC_OVERFLOW_STATE_TYEP          get_overflow_state();
        void            attch_ic_interrupt(void(*callback)(void));
        void            attch_update_interrupt(void(*callback)(void));

    private:
        GPIO            *capture_pin;
        TIM_TypeDef *TIMx;
        uint8_t         ch;
        uint16_t        period;//保存溢出值，用于计算占空比
        uint16_t        prescaler;//保存时钟分频值，用于计算占空比
        uint16_t        overflow_times;//溢出次数
        IC_OVERFLOW_STATE_TYEP      overflow_state;//溢出的情况。
如果发生 16 位溢出：IC_OVERFLOW，如果发送 32 位溢出:IC_FAULT
        uint8_t         polarity;

        void            init_info(GPIO *capture_pin);
        void            base_init(uint16_t Period,uint16_t Prescaler);

        uint16_t        (*_get_capture)(TIM_TypeDef *TIMx);
        void                (*_set_polarity)(TIM_TypeDef   *TIMx,uint16_t
TIM_OCPolarity);//设置为下降沿捕获
};
```

6.1 IN_CAPTURE(GPIO *capture_pin);

- **描述：**

IN_CAPTURE 的构造函数，实例化一个 IN_CAPTURE 对象。初始化某个引脚为 IN_CAPTURE 输入捕获引脚。该引脚必须是定时器 2,3,4 的某个通道之一。

- **参数：**

capture _pin：GPIO 对象指针或地址，如&PA7；

- **返回值：**

空

- **示例：**

IN_CAPTURE capture (&PB8);

6.2 begin(uint16_t prescaler);

- **描述：**

IN_CAPTURE 的初始化函数。将定时器的分频系数填入，计数器默认值为0xffff。

- **参数：**

prescaler：分频系数，定时器频率的计算方法，72M/prescaler；

- **返回值：**

空

- **示例：**

capture .begin(1);//一分频

6.3 set_count(uint16_t count);

- **描述：**

设置定时器计数器的值，一般用于捕获中断中，清零计数器。

- **参数：**

count：计数器的值：0~0xffff;

- **返回值：**

空

- **示例：**

capture.set_count (0);

6.4 set_polarity_falling();

- **描述：**

设置捕获下降沿产生中断。

- **参数：**

空;

- **返回值：**

空

- **示例：**

IN_CAPTURE capture (&PB8);

6.5 set_polarity_rising();

- **描述：**

设置捕获上升沿产生中断。

- **参数：**

空;

- **返回值：**

空

- **示例：**

```
void mesure_duty()//输入捕获中断事件
{
    ic.set_count(0);
    if(ic.polarity == TIM_ICPOLARITY_FALLING)//测量高电平时间完成
    {
        value1 = ic.get_capture() + 170;//校正值，查表可得
```

```
            ic.set_polarity_rising();//切换至测量低电平时间完成
        }
        else//测量低电平时间完成
        {
                value2 = ic.get_capture() + 170;//校正值，查表可得
            ic.set_polarity_falling();//切换至测量高电平时间完成
        }
}
```

6.6 overflow_event_process();

- **描述：**

溢出中断处理函数，为了使该模块能测量更长脉冲宽度的脉冲，增加溢出中断处理，此函数放到定时器溢出中断里面即可。

- **参数：**

空;

- **返回值：**

空

- **示例：**

```
void update_event()
{
      ic.overflow_event_process();
}
```

6.7 get_capture();

- **描述：**

获取在脉冲宽度之间的计数器的计数值，如果用户使用了溢出中断处理函数，会将溢出次数乘以 0xffff 再加上当前计数器的值返回。

- **参数：**

空;

- **返回值：**

空

- **示例：**

```
void mesure_frq()//
{
    ic.set_count(0);
    value1 = ic.get_capture() + 170;
}
```

6.8 get_overflow_state();

- **描述：**

获取定时器的溢出状态。

- **参数：**

溢出状态：0：没有溢出发生，1：代表溢出，2：代表溢出次数超过 0xff;如果状态是 2，则表示当前的分频系数过低，请调大分频次数。在测量时间不超过 60s 的情况下不用使用此函数。

- **返回值：**

空

- **示例：**

State = ic.get_overflow_state ();

If(state == 2)

Return err;

6.9 attch_ic_interrupt(void(*callback)(void));

- **描述：**

绑定捕获中断函数。

- **参数：**

callback：用户函数入口指针；

- **返回值：**

空

- **示例：**

```
ic.attch_ic_interrupt(mesure_frq);
```

6.10 attch_update_interrupt(void(*callback)(void));

- **描述：**

绑定捕获中断函数。

- **参数：**

callback：用户函数入口指针；

- **返回值：**

空

- **示例：**

```
ic.attch_update_interrupt(update_event);
```

第7章 EXTI 类

```
class EXTIx
{
    public:
        EXTIx(GPIO *exti_pin, EXTITrigger_TypeDef    trigger);
        void begin();
        void attach_interrupt(void (*callback_fun)(void));
        void interrupt(FunctionalState enable);

    private:
        GPIO                    *exti_pin;
        EXTITrigger_TypeDef trigger;
        uint8_t                 port_source;
        uint8_t                 pin_source;
        uint32_t                exti_line;
        uint8_t                 irq;

        void init_info(GPIO *exti_pin);
};
```

7.1 EXTIx(GPIO *EXTI_pin,EXTITrigger_TypeDef trigger);

- **描述：**

EXTIx 的构造函数，实例化一个 EXTIx（创建 EXTIx 对象）。

- **参数：**

EXTI_pin：GPIO 对象指针或地址，如&PA5；

trigger:EXTITrigger_TypeDef 类型枚举变量；如：EXTI_Trigger_Falling

- **返回值：**

空；

- **示例：**

EXTIx ex(&PA8,EXTI_Trigger_Falling);

7.2 begin();

- **描述：**

EXTIx 初始化。

- **参数：**

空

- **返回值：**

空

- **示例：**

EXTIx ex(&PA8,EXTI_Trigger_Falling);

ex.begin();

7.3 attach_interrupt(void (*callbackFun)(void));

- **描述：**

外部中断绑定回调函。

- **参数：**

(*callbackFun)(void)：中断回调函数指针（函数名），如 void exti_it_event(void)的函数名 exti_it_event。

- **返回值：**

空

- **示例：**

exti.attach_interrupt(exti_it_event);

7.4 interrupt(FunctionalState enable);

- **描述：**

外部中断使能控制。

- **参数：**

enable：ENABLE/DISABLE。

- **返回值：**

空

- **示例：**

ex.interrupt(ENABLE);

第8章 USART 类

```
class USART
{
    public:
        USART(USART_TypeDef *USARTx,GPIO *tx_pin,GPIO *rx_pin);

        void    begin(uint32_t baud_rate);
        void    begin(uint32_t baud_rate,uint8_t data_bit,uint8_t parity,float
stop_bit);
        void    attach_rx_interrupt(void (*callback_fun)(void));
        void    attach_tx_interrupt(void (*callback_fun)(void));

        int     put_char(uint16_t ch);
        void    put_string(const char *str);
        void    printf_length(const char *str,uint16_t length);
        void    printf(const char *fmt,...);
        void    wait_busy();

        uint16_t    receive();

    private:
        USART_TypeDef           *_USARTx;
        DMA_Channel_TypeDef *_DMA1_Channelx;
        char                    send_buf[UART_MAX_SEND_BUF];
        uint16_t                dma_send_string(const char *str,uint16_t
length);
        void                    put_string(const char *str,uint16_t length);
        void                    set_busy();
        void                    interrupt(FunctionalState enable);
};
```

8.1 USART(USART_TypeDef * USARTx, GPIO* tx_pin,GPIO* rx_pin);

- **描述：**

USART 的构造函数，实例化一个串口（创建串口对象）。

- **参数：**

USARTx：USART_TypeDef 类型值，如 USART1、2、3、4、5；

tx_pin：GPIO 对象指针或地址，如 PA9,PA2,PB10；

rx_pin：GPIO 对象指针或地址，如 PA10,PA3,PB11；

- **返回值：**

空

- **示例：**

uart1(USART1,&PA9,&PA10);

8.2 begin(uint32_t baud_rate);

- **描述：**

串口初始化函数。

- **参数：**

baud_rate：串口波特率，如 9600，115200 等；

- **返回值：**

空

- **示例：**

uart1.begin(9600);

8.3 begin(uint32_t baud_rate,uint8_t data_bit,uint8_t parity,float stop_bit);

- **描述：**

串口初始化函数。

- **参数：**

baud_rate：串口波特率，如 9600，115200 等；

data_bit：数据位，例如 8

parity：校验位，例如 0：无校验、1：奇校验、2：偶校验

stop_bit：停止位，例如 1、1.5、2

- **返回值：**

空

- **示例：**

uart1.begin(9600);

8.4 attach_rx_interrupt(void (*callback_fun)(void));

- **描述：**

串口中断绑定回调函。

- **参数：**

(*callback_fun)(void)：中断回调函数指针（名），如 void uar_it_rx(void)的函数名 uar_it_rx。

- **返回值：**

空

- **示例：**

uart1.attach_interrupt(uar_it_rx);

8.5 attach_tx_interrupt(void (*callback_fun)(void));

- **描述：**

串口中断绑定回调函。

- **参数：**

(*callback_fun)(void)：发送完成中断回调函数指针（名），如 void uar_it_tx(void)的函数名 uar_it_tx。

- **返回值：**

空

- **示例：**

uart1.attach_interrupt(uar_it_tx);

8.6 receive();

- **描述：**

读取串口接收到的数据，一般应用于串口接收中断事件中。

- **参数：**

空；

- **返回值：**

u16 类型串口接收到的数据；(串口数据格式有 9 位数据的模式，所以使用 u16 类型，一般情况下使用的是 u8 类型的数据)

- **示例：**

uint8_t char;

char = receive();

8.7 put_char(char ch);

- **描述：**

串口输出一个字节。

- **参数：**

ch：char 类型参数。

- **返回值：**

空

- **示例：**

uart1.put_ char (‘1’);

uart1.put_char(0x31);

8.8 put_string(const char *str);

- **描述：**

串口输出一个字符串。遇到'\0'将会停止输出

- **参数：**

str：char 类型指针。

- **返回值：空**

- **示例：**

uart1.put_string("hellow world");

8.9 printf(const char* fmt,...);

- **描述：**

串口格式化输出字符串。使用方法和标准 c 中的一样。

- **参数：**

fmt：字符串

...：变量参数

- **返回值：空**

- **示例：**

uart1.printf("hellow world");

uart1.printf("intVal = %03d",x);

uart1.printf("floatVal = %f",f);

8.10 printf_length(const char *str,uint16_t length);

- **描述：**

串口输出指定长度字符串。这个主要针对发送特定长度的缓冲区，中间有'/0'也可以继续发送。

- **参数：**

str：char 类型指针。

length：输出字符串长度。

- **返回值：**

空

- **示例：**

uart1.putString(“hellow world”,10);

8.11 wait_busy()

- **描述：**

等待串口输出完成。这个函数的使用概率是非常低的，除非是你必须需要等待串口发送完成才能执行下面的函数才需要使用这个函数。

- **参数：**

空

- **返回值：**

空

- **示例：**

uart1.put_string(“hellow world”,10);

uart1.wait_busy();

第9章 GTIMER 类

```
class TIM
{
    public:
        TIM(TIM_TypeDef *TIMx);
        void begin(uint32_t frq);
        void attach_interrupt(void(*callback)(void));
        void interrupt(FunctionalState enable);
        void start(void);
        void stop(void);
        void reset_frq(uint32_t frq);
    private:
        void base_init(uint16_t period,uint16_t prescaler);
        void set_reload(uint16_t auto_reload);
        void clear_count(void);
        TIM_TypeDef *_TIMx;

};
```

9.1 TIM(TIM_TypeDef* TIMx);

- **描述：**

TIM 的构造函数，实例化一个 TIM（创建 TIM 对象）。

- **参数：**

TIMx：TIM_TypeDef 对象指针或地址，如 TIM2、3、4、5、6、7,；

注意：TIMx 目前只支持通用定时器，不支持高级定时器。如果使用了 PWM 请注意 pwm 引脚是否基于要使用的定时器，如果是的话最好避让该定时器，否则会改变 PWM 引脚的频率和占空比。

- **返回值：**

空

- **示例：**

TIM(TIM2);

9.2 begin(uint32_t frq);

- **描述：**

初始化定时器参数，创建通用定时器对象后可调用；

- **参数：**

frq：定时器的中断频率，取值范围 1-1000000。

返回值：空

- **示例：**

tim2.begin(1000);

9.3 attach_interrupt(void(*callback)(void));

- **描述：**

定时器溢出中断绑定回调函。

- **参数：**

(*callbackFun)(void)：中断回调函数指针（名），如 void timer2It(void)的函数名 timer2It。

- **返回值：**

空

- **示例：**

tim2.attach_interrupt(timer2It);

9.4 interrupt(FunctionalState enable);

- **描述：**

通用定时器溢出中断使能。

- **参数：**

enable：FunctionalState 类型值，如 ENABLE,DISABLE。

- **返回值：**

空

- **示例：**

tim2.interrupt(ENABLE);

9.5 start(void);

- **描述：**

启动通用定时器。初始化完成定时器，调用此函数才能启动定时器。

- **参数：**

空；

- **返回值：**

空;

- **示例：**

tim2.start();

9.6 stop(void);

- **描述：**

停止通用定时器。停止定时器计时，可调用 start 再次启动。

- **参数：**

空；

- **返回值：**

空;

- **示例：**

tim2.stop();

9.7 void reset_frq(uint32_t frq);

- **描述：**

重新初始化定时器参数。

- **参数：**

frq：定时器的中断频率，取值范围 1-1000000。

- **返回值：**

空；

- **示例：**

begin(1000);

9.8 set_reload(uint16_t autoreload);

- **描述：**

修改溢出值。

- **参数：**

autoreload：定时器溢出值；

- **返回值：**

空；

- **示例：**

tim2.set_reload(1024);

9.9 clear_count(void);

- **描述：**

定时器计数器清理。

- **参数：**

空；

- **返回值：**

空；

- **示例：**

```
tim2.clear_count();
```

第10章 TIMERONE 类

```
class TIMERONE
{
    public:
        TIMERONE();
        void begin(uint32_t frq);
        void attach_interrupt(void(*callback)(void));
        void interrupt(FunctionalState x);
        void start();
        void stop();
        void reset_frq(uint32_t frq);
    private:
        void base_init(uint16_t period,uint16_t prescaler);
        void set_reload(uint16_t autoreload);
        void clear_count(void);
};
```

注解:请参考通用定时器；

第11章 SPI 类

```
class      SPIClass
{
      public:
            SPI(SPI_TypeDef *SPIx,GPIO *sck,GPIO *miso,GPIO *mosi);

            void       begin (SPI_CONFIG_TYPE *spi_config);
            void       config(SPI_CONFIG_TYPE *spi_config);
            uint8_t    read_config(void);

            int8_t    write(uint8_t data);
            int8_t    write(uint8_t *data,uint16_t data_length);

            uint8_t    read();
            int8_t    read(uint8_t    *recv_data);
            int8_t    read(uint8_t *recv_data,uint16_t data_length);
      public:
            int8_t take_spi_right(SPI_CONFIG_TYPE *spi_config);
            int8_t release_spi_right(void);
      private:
            uint8_t         current_dev_num;
            SPI_TypeDef *spi;
            uint8_t         busy;
};
```

11.1 SPI(SPI_TypeDef *SPIx,GPIO *sck,GPIO *miso,GPIO *mosi);

- **描述：**
- **参数：**

SPIx：SPI_TypeDef 类型指针或地址，如 SPI1，SPI2，SPI3；

p_sck_pin：GPIO 对象指针或地址，如 PA5；

p_miso_pin：GPIO 对象指针或地址，如 PA6；

p_mosi_pin:GPIO 对象指针或地址，如 PA7；

注意：如果使用 SPI 的冲映射引脚需要自己添加冲映射代码。硬件资源有限，只测试了 SPI1。

- **返回值：**

空

- **示例：**

SPI spi1(SPI1,&PA5,&PA6,&PA7);

SPI spi2(SPI2,&PB13,&PB14,&PB15);

11.2 begin (SPI_CONFIG_TYPE* spi_config);

- **描述：**

SPI 初始化。

- **参数：**

spi_config：SPICONFIG 对象指针或地址；

- **返回值：**

空

- **示例：**

spi->begin(&spiDevW5500);

11.3 config(SPI_CONFIG_TYPE* spi_config);

- **描述：**

SPI 更新配置。

- **参数：**

spi_config：SPI_CONFIG_TYPE 对象指针或地址；

- **返回值：**

空

- **示例：**

spi1->config(SPI_CONFIG);

11.4 read_config(void);

- **描述：**

读取 SPI 配置号。

- **参数：**

空；

- **返回值：**

currentDevNum：spi 处于某个芯片 spi 配置的编号；

- **示例：**

spi->config(&SPIDevSDCard);

11.5 write(uint8_t data);

- **描述：**

写一个字节的数据，发送并读取；

- **参数：**

data:发送的数据；

- **返回值：**

spi->dr：接收到的数据；

- **示例：**

spi1.write(data);

11.6 write(uint8_t *data,uint16_t data_length);

- **描述：**

写一个指定长度字节的数据；

- **参数：**

***data**:传输的数据的指针；

data_length：传输数据长度；

- **返回值：**

spi->dr：接收到的数据；

- **示例：**

```
spi1.write(data);
```

11.7 read();

- **描述：**

读取一个指定长度字节的数据；

- **参数：**

空

- **返回值：**

spi->dr：接收到的数据；

- **示例：**

```
spi1.read();
```

11.8 read(uint8_t* recv_data);

- **描述：**

读取一个指定长度字节的数据；

- **参数：**

*recv_data：接收数据的指针。

- **返回值：**

0：正常

- **示例：**

```
spi1.read();
```

11.9 read(uint8_t *recv_data,uint16_t datalength);

- **描述：**

接收指定长度字节的数据。

- **参数：**

***recv_data**：传输的数据的指针；

datalength:传输数据长度；

- **返回值：**

空；

- **示例：**

spi1.read(&data,10);

11.10take_spi_right(SPI_CONFIG_TYPE* spi_config);

- **描述：**

获取 SPI 使用权，并设置速度；**获取不到一直等待**。

- **参数：**

spi_config：spi 的配置；

- **返回值：**

0：正常；

- **示例：看下文**

11.11release_spi_right(void);

- **描述：**

释放 SPI 使用权。

- **参数：空**

- **返回值：**

0：正常；

- **示例：**

```
spi->take_spi_right(&spiDevW5500);
    cs->reset();                                          // CS=0, SPI start
    spi->write( (addrbsb & 0x00FF0000)>>16);// Address byte 1
    spi->write( (addrbsb & 0x0000FF00)>> 8);// Address byte 2
   spi->write( (addrbsb & 0x000000F8) + 4);    //
    spi->write(data);                                  // Data write (write 1byte data)
    cs->set();                                            // CS=1,    SPI end
spi->release_spi_right();
```

第12章 SOFTSPI 类

```
class SOFTSPI
{
    public:
        SOFTSPI(GPIO *sck,GPIO *miso,GPIO *mosi);

        void       begin(SPI_CONFIG_TYPE *spi_config);
          void       config(SPI_CONFIG_TYPE *spi_config);
        uint8_t read_config(void);

        int8_t    write(uint8_t data);
        int8_t    write(uint8_t *data,uint16_t data_length);

        uint8_t read();
        int8_t    read(uint8_t *data);
        int8_t    read(uint8_t *rcvdata,uint16_t data_length);
    public:
        int8_t take_spi_right(SPI_CONFIG_TYPE *spi_config);
        int8_t release_spi_right(void);

    private:
        GPIO       *sck_pin;
        GPIO       *mosi_pin;
        GPIO       *miso_pin;

        uint8_t mode;
```

```
        uint8_t bit_order;
        uint8_t spi_delay;

        uint8_t current_dev_num;
        uint8_t busy;

        uint8_t transfer0(uint8_t data);
        uint8_t transfer1(uint8_t data);
        uint8_t transfer2(uint8_t data);
        uint8_t transfer3(uint8_t data);
        uint8_t transfer(uint8_t data);
};
```

12.1 SOFTSPI(GPIO* sck,GPIO* miso,GPIO* mosi);

- **描述：**

SOFTSPI 的构造函数，实例化一个 SOFTSPI（创建 SOFTSPI）对象。此对象应该在 object.cpp 中创建，被驱动层调用。

- **参数：**

sck：GPIO 对象指针或地址，如&PA5；

miso：GPIO 对象指针或地址，如&PA6；

mosi:GPIO 对象指针或地址，如&PA7；

- **返回值：**

空

- **示例：**

SOFTSPI sspi(&PA5,&PA6,&PA7);

12.2 begin(SPI_CONFIG_TYPE* spi_config);

12.3 config(SPI_CONFIG_TYPE* spi_config);

12.4 read_config(void);

12.5 write(uint8_t data);

12.6 write(uint8_t *data,uint16_t dataln);

12.7 read();

12.8 read(uint8_t* data);

12.9 read(uint8_t *rcvdata,uint16_t dataln);

12.10 take_spi_right(SPI_CONFIG_TYPE* spi_config);

12.11 release_spi_right(void);

以上函数的方法和硬件 SPI 兼容

第13章 I2C 类

```
class I2C
{

    public:
        I2C(I2C_TypeDef *I2Cx,GPIO *scl_pin,GPIO *sda_pin);
        void            begin(uint32_t speed);
        void            config(uint32_t speed);
        uint32_t        read_config();

        int8_t    write_byte(uint8_t    slave_address,uint8_t    reg_address,uint8_t
data);
        int8_t    write_byte(uint8_t    slave_address,uint8_t    reg_address,uint8_t
*data,uint16_t num_to_write);
        int8_t    read_byte   (uint8_t   slave_address,uint8_t   reg_address,uint8_t
*data);
        int8_t    read_byte   (uint8_t   slave_address,uint8_t   reg_address,uint8_t
*data,uint16_t num_to_read);
          int8_t  wait_dev_busy(uint8_t slave_address);
    public:
        int8_t take_i2c_right(uint32_t speed);
        int8_t release_i2c_right(void);

    private:
        int8_t start();
        int8_t stop();
        int8_t send_no_ack();
        int8_t send_ack();

        int8_t send_byte(uint8_t regData);
        int8_t send_7bits_address(uint8_t slave_address);
        int8_t receive_byte(uint8_t *data);

    private:
        I2C_TypeDef *I2Cx;
        GPIO            *sda_pin;
        GPIO            *scl_pin;
        uint32_t        speed;
        uint8_t         busy;
};
```

13.1 I2C(I2C_TypeDef* I2Cx,GPIO* scl_pin,GPIO* sda_pin);

- **描述：**

I2C 的构造函数，实例化一个 I2C（创建 I2C 对象）。此对象一般 eBox 默认提供两个对象 i2c1 和 i2c2，在 object.cpp 中定义。

- **参数：**

I2Cx：I2C_TypeDef 类型值，如 I2C1，I2C2；

sda_pin：GPIO 对象指针或地址，如 PB7；

scl_pin：GPIO 对象指针或地址，如 PB6；

- **返回值：空**

- **示例：**

I2C i2c1(I2C1,&PB6,&PB7);

I2C i2c2(I2C2,&PB10,&PB11);

13.2 begin(uint32_t speed);

- **描述：**

I2C 初始化函数；

- **参数：**

speed：I2C 速度设置，如；100000,200000,300000,400000 等；

- **返回值：**

空

- **示例：**

i2c.begin(100000);

13.3 config(uint32_t speed);

- **描述：**

I2C 速度设置函数；

- **参数：**

speed：I2C 速度设置，如；100000,200000,300000,400000；

- **返回值：**

空

- **示例：**

I2c1.config(100000);

13.4 read_config();

- **描述：**

返回当前 I2C 的速度配置；此函数

- **参数：**

空；

- **返回值：**

i2c 的速度配置：speed；

- **示例：**

I2c1.read_config();

13.5 write_byte(uint8_t slave_address, uint8_t reg_address,uint8_t data);

- **描述：**

I2C 器件写入一个字节；

- **参数：**

slave_address：I2C 器件地址；

reg_address：器件寄存器地址；

data：要写入的值；

- **返回值：**

err：0 表示正常；

- **示例：**

i2c->write_byte(SLAVEADDRESS,GYRO_CONFIG,0x18);

13.6 write_byte(uint8_t slave_address, uint8_t reg_address,uint8_t* data,uint16_t num_to_write);

- **描述：**

I2C 器件写入指定长度字节；

- **参数：**

slave_address：I2C 器件地址；

reg_address：器件寄存器地址；

data：uint8_t 数据指针；

num_to_write：写入的数据长度；

- **返回值：**

err：0 表示正常；

- **示例：**

uint8_t tmp[10];

i2c->write_byte(SLAVEADDRESS,SMPLRT_DIV,tmp，10);

13.7 read_byte (uint8_t slave_address, uint8_t reg_address,uint8_t* data);

- **描述：**

I2C 器件读取一个字节；

- **参数：**

slave_address：I2C 器件地址；

reg_address：器件寄存器地址；

data：输出数据的指针；

- **返回值：**

err：0 表示正常；

- **示例：**

uint8_t tmp;

i2c->read_byte(SLAVEADDRESS,SMPLRT_DIV,&tmp);

13.8 read_byte (uint8_t slave_address,

uint8_t reg_address,uint8_t* data,uint16_t num_to_read);

- **描述：**

I2C 器件读取指定长度数据；

- **参数：**

slave_address：I2C 器件地址；

reg_address：器件寄存器地址；

data：输出数据的指针；

num_to_read：读取的数据长度；

- **返回值：**

err：0 表示正常；

- **示例：**

uint8_t tmp[10];

i2c1->read_byte(SlaveAddress,SMPLRT_DIV,tmp,10);

13.9 wait_dev_busy(uint8_t slave_address);

- **描述：**

I2C 等待器件响应；

- **参数：**

slave_address：I2C 器件地址；

- **返回值：**

0：正常；

-1：超时，器件无响应。

- **示例：**

```
for(uint16_t i = 0; i < num_to_write; i++)
{
        write_byte(byte_addr++,buf[i]);
        ret = i2c->wait_ dev _busy(SLAVE_ADDR);
}
```

13.10 take_i2c_right(uint32_t speed);

- **描述：**

获取 I2C 使用权，并设置速度；**获取不到一直等待**。

- **参数：**

speed：I2C 速度；

- **返回值：**

0：正常；

- **示例：看下文**

13.11 release_i2c_right(void);

- **描述：**

释放 I2C 使用权。

- **参数：**

空

- **返回值：**

0：正常；

- **示例：**

```
i2c->take_i2c_right(speed);
i2c->write_byte(SLAVE_ADDR,byte_addr,byte);
i2c->release_i2c_right();
```

第14章 SOFTI2C 类

```
class SOFTI2C
{
  public:
        SOFTI2C(GPIO *scl, GPIO *sda);
        void          begin(uint32_t speed);
        int8_t        config(uint32_t speed);
        uint32_t read_config();
        int8_t        write_byte(uint8_t slave_address,
                       uint8_t reg_address,uint8_t data);
        int8_t        write_byte(uint8_t slave_address,
                       uint8_t reg_address,uint8_t *data,uint16_t num_to_write);
        int8_t        read_byte (uint8_t slave_address,
                       uint8_t reg_address,uint8_t *data);
        int8_t        read_byte (uint8_t slave_address,
                      uint8_t reg_address,uint8_t *data,uint16_t num_to_read);
       int8_t         wait_dev_busy (uint8_t slave_address);
   public:
        int8_t        take_i2c_right(uint32_t speed);
        int8_t        release_i2c_right(void);

   private:
        void     start();
        void     stop();
        int8_t   send_ack();
        int8_t   send_no_ack();

        int8_t   send_byte(uint8_t Byte);
        int8_t   send_7bits_address(uint8_t slave_address);
        uint8_t receive_byte();

        int8_t   wait_ack();

  private:
        GPIO            *sda_pin;
        GPIO            *scl_pin;
        uint32_t speed;
        uint16_t delay_times;
        uint8_t  busy;
};
```

14.1 SOFTI2C(GPIO* scl, GPIO* sda);

- **描述：**

SOFTI2C 的构造函数，实例化一个 SOFTI2C（创建 SOFTI2C）对象。此对象一般在 object.cpp 中定义。

- **参数：**

scl_pin：GPIO 对象指针或地址，如&PA4；

sda_pin：GPIO 对象指针或地址，如&PA5；

- **返回值：空**

- **示例：**

SOFTI2C si2c(&PA4,&PA5);

SOFTI2C si2c1(&PB6,&PB7);

14.2 begin(uint32_t speed);

- **描述：**

SOFTI2C 初始化函数；初始化 I2C，并设置速度为_speed。

- **参数：**

speed：I2C 速度设置，如；100000,200000,300000,400000；注意：如果不是以上四个参数，_speed 的值将会直接用于 delay_us()函数。delay_us(_speed)。

- **返回值：**

空

- **示例：**

si2c1->begin(100000);

14.3 config(uint32_t speed);

- **描述：**

SOFTI2C 速度设置函数；

- **参数：**

speed：I2C 速度设置，如；100000,200000,300000,400000；注意：如果不是以上四个参数，_speed 的值将会直接用于 delay_us()函数。delay_us(_speed)。

- **返回值：**

空

- **示例：**

si2c1->config(100000);

14.4 其他公有成员函数

和硬件 I2C 的完全兼容，请参考上一章节。

第15章 RTC 类

```
class RTCCLASS
{
    public:
        uint8_t sec;
        uint8_t min;
        uint8_t hour;
    public:
        void begin();
        void attach_interrupt(uint16_t event, void (*callbackFun)(void));
        void interrupt(uint32_t bits,FunctionalState x);
        void set_counter(uint32_t count);
        void set_alarm(uint32_t count);
        uint32_t get_counter();
        void set_time_HMS(uint8_t h,uint8_t m,uint8_t s);
        void set_alarm   (uint8_t h,uint8_t m,uint8_t s);
        void get_time_HMS(uint8_t *h,uint8_t *m,uint8_t *s);

    private:
        void      config();
        uint8_t is_config(uint16_t configFlag);
        void      set_config_flag(uint16_t configFlag);
};
```

注意：该类有库本身默认实例化对象 rtc，不需要用户自己创建对象，

```
#define RTC_CLOCK_LSI 0
#define RTC_CLOCK_LSE 1
#define RTC_CLOCK_HSI 2//² »Ö §³ Ö£¬ÔÝÊ±² »¿ÉÓÃ

//中断源
#define RTC_EVENT_OW        ((uint16_t)0x0004)   /*!< Overflow interrupt */
#define RTC_EVENT_ALR  ((uint16_t)0x0002)   /*!< Alarm interrupt */
#define RTC_EVENT_SEC       ((uint16_t)0x0001)   /*!< Second interrupt */

//用户配置
```

```
#define RTC_CFG_FLAG 0XA5A5
#define RTC_CLOCK_SOURCE RTC_CLOCK_LSI
//#define RTC_CLOCK_SOURCE RTC_CLOCK_LSE
```

15.1 begin();

- **描述：**

初始化 rtc；

- **参数：**

空；

- **返回值：**

空

- **示例：**

```
rtc.begin();
```

15.2 attach_interrupt(uint16_t event, void (*callbackFun)(void));

- **描述：**

串口中断绑定回调函。

- **参数：**

event: 中断事件类型，如 RTC_EVENT_OW，RTC_EVENT_ALR，RTC_EVENT_SEC

(*callbackFun)(void)：中断回调函数指针（名），如 void rtcSECIt(void)的函数名 rtcSECIt。

- **返回值：**

空

- **示例：**

rtc.attach_interrupt(RTC_EVENT_SEC,rtcSECIt);

15.3 interrupt(uint32_t bits,FunctionalState x);

- **描述：**

rtc 中断使能。

- **参数：**

bits: 中断事件类型，如 RTC_EVENT_OW，RTC_EVENT_ALR，RTC_EVENT_SEC

x：FunctionalState 类型值，如 ENABLE,DISABLE。

- **返回值：**

空

- **示例：**

rtc.interrupt(RTC_EVENT_SEC,ENABLE);

15.4 set_counter(uint32_t count);

- **描述：**

设定 rtc 计数器的值；

- **参数：**

count：计数器设定值

注解：最大 2^32；开始运行 136 年后溢出；

- **返回值：**

空

- **示例：**

rtc.set_counter(100);

15.5 set_alarm(uint32_t count);

- **描述：**

设定 rtc 闹钟值，如果 rtc 的 counter 中的值和 alarm 中的值一样，将置位 alarm 中断标志位；

- **参数：**

count：闹钟设定值

注解：最大 2^32；

- **返回值：**

空

- **示例：**

rtc.set_alarm(110);

15.6 get_counter();

- **描述：**

获取 rtc 计数器的值；

- **参数：**

空；

- **返回值：**

counter：counter 当前值；

- **示例：**

val = rtc.get_counter();

15.7 set_time_HMS(uint8_t h,uint8_t m,uint8_t s);

- **描述：**

以 HHMMSS 格式设定 rtc 闹钟值；

- **参数：**

h：闹钟时；

m：闹钟分；

s：闹钟秒；

- **返回值：**

空

- **示例：**

rtc.set_time_HMS(11,59,50);

15.8 set_alarm(uint8_t h,uint8_t m,uint8_t s);

- **描述：**

设定 rtc 闹钟值，如果 rtc 的 counter 中的值和 alarm 中的值一样，将置位 alarm 中断标志位；

- **参数：**

h：闹钟时；

m：闹钟分；

s：闹钟秒；

注意：如果想使用内部简易的的 hhmmss 闹钟功能，因为 counter 默认不清零。请在秒中断中判断 counter 是不是 0x15179,如果是请将 counter 设置为 0。或者在闹钟中断中重新设定新的闹钟值。否则闹钟将失去作用。

- **返回值：**

空

- **示例：**

rtc.set_alarm(11,59,59);

15.9 get_time_HMS(uint8_t* h,uint8_t* m,uint8_t* s);

- **描述：**

获取 HHMMSS 格式的时间。如果不想使用 rtc 中断，可通过此函数获得当

前时间。

- **参数：**

h：闹钟时的指针；

m：闹钟分的指针；

s：闹钟秒的指针；

- **返回值：**

空

- **示例：**

uint8_t h,m,s;

rtc.get_time_HMS(&h,&m,&s);

15.10RTC 下的 HHMMSS 时钟变量

uint8_t sec;

uint8_t min;

uint8_t hour;

这三个值在允许秒中断后系统会自动的更新。如果想获取当前的系统的时间，可直接读取这三个值即可。

第16章 FLASHCLASS 类

```
class FLASHCLASS
{
    public:
        int read(uint32_t iAddress, uint8_t *buf, int32_t iNbrToRead) ;
        int write(uint32_t iAddress, uint8_t *buf, uint32_t iNbrToWrite);
    private:
        uint16_t write_without_check(uint32_t iAddress, uint8_t *buf,
                    uint16_t iNumByteToWrite);
};

#if defined (STM32F10X_HD) || defined (STM32F10X_HD_VL) || defined
(STM32F10X_CL) || defined (STM32F10X_XL)
  #define FLASH_PAGE_SIZE       ((uint16_t)0x800)
#else
  #define FLASH_PAGE_SIZE       ((uint16_t)0x400)
#endif
```

16.1 read(uint32_t iAddress, uint8_t *buf, int32_t iNbrToRead) ;

- **描述：**

读取内部 flash。

- **参数：**

iAddress： flash 绝对地址；

buf： 缓冲区；

iNbrToRead： 读取字节数量；

- **返回值：**

i:返回读取数据长度

- **示例：**

```
flash.read(0x08000000,buf,100);
```

16.2 write(uint32_t iAddress, uint8_t *buf, uint32_t iNbrToWrite);

- **描述：**

写入内部 flash。

- **参数：**

iAddress： flash 绝对地址；

buf： 缓冲区；

iNbrToWrite： 读取字节数量；

- **返回值：**

iNbrToWrite：返回写入数据长度

- **示例：**

Flash.write(0x08000000,buf,100);

第17章 EBOX_IWDG 类

```
class EBOX_IWDG
{
    public:
        EBOX_IWDG(){};
        void begin(uint16_t ms);
        void feed();

};#endif
```

17.1 EBOX_IWDG();

- **描述：**

创建一个独立看门狗对象。

- **参数：**

空；

- **返回值：**

空

- **示例：**

EBOX_IWDG wdg;

17.2 begin(uint16_t ms);

- **描述：**

初始化看门狗。

- **参数：**

ms：看门狗的最长喂狗时间；

- **返回值：**

空

- **示例：**

wdg. begin (100);

17.3 feed();

- **描述：**

喂狗函数，在定时器中调用，或者在主函数，或者在操作系统中的某个任务中。必须在规定的时间内调用一次此函数，否则 MCU 将复位

- **参数：**

空;

- **返回值：**

空

- **示例：**

wdg.feed();

第18章 Stream 类

- **描述：**

Stream 类继承了 Print，添加了数据的读取和转换功能，由于不知道底层实现，跟 Print 类似，将底层的实现申明为虚函数。

Stream 申明了 3 个虚函数：

virtual int available() = 0; //判断当前是否有可读取的数据

virtual int read() = 0; //读取一个字节并丢弃

virtual int peek() = 0; //读取这个字节但不丢弃，此时如果在读取的话还是同样的数据

读取数据时，可能当前没有数据可读取但数据很快就传送过来，也可能根本就没有数据，为了适应这种不确定性，设定了超时读取时间，超过这个时间还没有读取到数据则读取失败，单位 ms，默认 1000ms

void setTimeout(unsigned long timeout);

unsigned long getTimeout(void)

18.1 available(void)

描述： 该函数 available() 获取数据流中接收到的字节数

返回值： int 类型

18.2 read(void)

描述： 获取数据流中第一个字节数据，获取数据后会清除当前字节数据，与 peek()函数有区别

返回值： 取数据字符的第一个字节（8bit）,无数据时返回 -1

18.3 peak(void)

描述： 从数据流中读取当前的一个字节，不会清除数据流中当前字节数据，与 read() 函数有区别

返回值： 取数据字符的第一个字节（8bit）,无数据时返回 -1

18.4 findUntil(char *target, size_t targetLen, char *terminate, size_t termLen);

描述： 在接受到的数据中查找字符串 target，遇到 terminator 结束：

参数：

char *target 指向查找的目标

size_t targetLen 目标字符串长度

char *terminate 指向结束字符串

size_t termLen 结束字符串长度

返回值：找到目标字符串返回 True，超时返回 false

18.5 find(char *target);

18.6 find(char *target, size_t length);

18.7 find(char target)；

18.8 findUntil(char *target, char *terminator);

17.2->17.5 都是通过调用 17.1 实现的，相关说明参考 17.1

18.9 parseInt(LookaheadMode lookahead = SKIP_ALL, char ignore = NO_IGNORE_CHAR);

描述： 读取数据并转换成整型，转换从第一个遇到的数字或者负号开始，到第一个非数字结束

参数：

LookaheadMode lookahead 忽略

SKIP_ALL, 忽略所有无效字符

SKIP_NONE 不跳过任何字符

SKIP_WHITESPACE 跳过空格，tab，换行，回车

char ignore = NO_IGNORE_CHAR

返回值：Long

18.10 parseFloat(LookaheadMode lookahead = SKIP_ALL, char ignore = NO_IGNORE_CHAR);

描述： 读取数据并转换成浮点型，转换从第一个遇到的数字或者负号开始，到第一个非数字结束

参数： 参考 parseInt（）

返回值： Float (**注意，超时会返回 0**，建议在调用此函数前先调用 available() 判断已经收到数据)

18.11 readBytes(char *buffer, size_t length);

描述：读取指定长度的数据到 buffer

参数：

char *buffer 缓存

size_t length 要读取的长度

返回值：返回放置在缓存中的字符数

18.12 readBytesUntil(char terminator, char *buffer, size_t length);

描述： 读取指定长度的字符到数组 buffer，遇到终止字符 terminator 后停止。

参数：

char terminator, 终止字符

char *buffer 缓存

size_t length 要读取的长度

返回值： 返回存入缓存的字符数，0 表示没有有效数据