

# BabyOS设计和使用手册

---

V0.1

## 修订记录:

日期	记录	修订人
2022.03.18	编写初稿	notrynohigh

# 目录

---

## BabyOS设计和使用者手册

### 目录

#### 1. 项目介绍

#### 2. 设计思路

#### 3. 快速体验

- 3.1 准备基础工程
- 3.2 添加BabyOS代码
- 3.3 修改配置
- 3.4 调用必要的函数
- 3.5 快速体验结果

#### 4. 进阶体验

- 4.1 补充MCU资源初始化
- 4.2 添加驱动文件
- 4.3 添加硬件接口
- 4.4 记录开机次数

#### 5. 详细介绍

- 5.1 添加MCU
- 5.2 HAL层介绍
  - 5.2.1 心跳时钟
  - 5.2.2 延时函数
  - 5.2.3 通讯接口
- 5.3 驱动层介绍
  - 5.3.1 硬件接口
  - 5.3.2 注册设备
  - 5.3.3 操作设备
- 5.4 SECTION介绍
- 5.5 功能组件

#### 6. 参与开发

# 1. 项目介绍

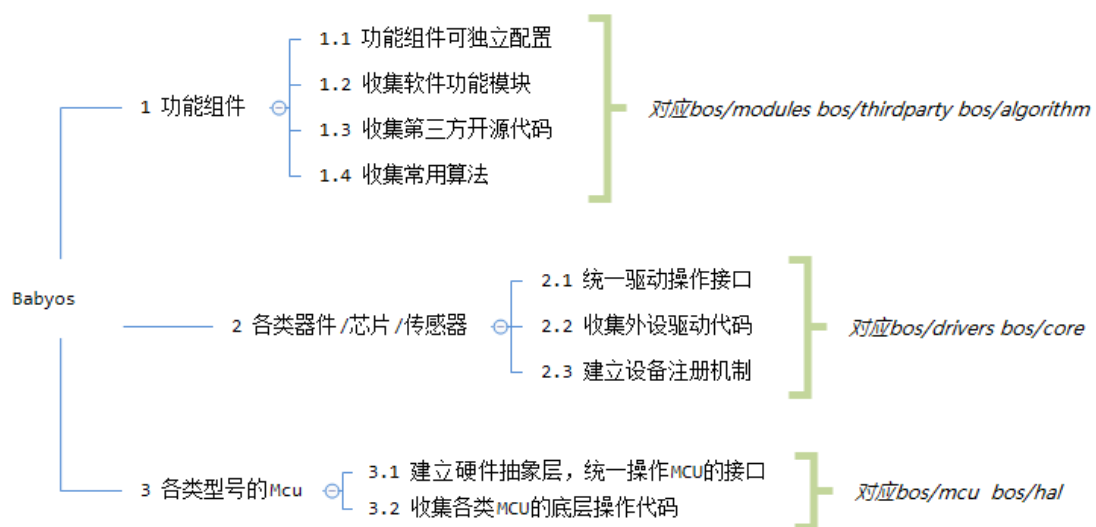
BabyOS构想是搭建一个货架存放软件CBB（CommonBuildingBlock）。

不同产品、系统之间有许多共用的模块，这些模块调试稳定后放入货架作为积累。久而久之，货架上便有许多成熟的CBB供开发人员使用。减少大量重复劳动或者研发已经存在的成果。另一方面，如果产品是基于这些成熟的CBB搭建而成，产品的质量、进度都会得到更好的控制和保证。

货架的搭建和CBB积累并不是一个人可以完成的，需要集合众人的力量。于是2019年底发起了BabyOS开源项目。开发人员以兴趣为动力，无任何薪资报酬。坚持开源互助，共同进步。

# 2. 设计思路

BabyOS是想搭建一个货架，那么货架上是怎么存放东西的呢？这便决定了代码的结构。



## 3. 快速体验

以STM32F107进行说明。相关的例子在 [https://gitee.com/notrynohigh/BabyOS\\_Example](https://gitee.com/notrynohigh/BabyOS_Example)

### 3.1 准备基础工程

基础功能需要做到以下几点：

①MCU时钟及片内外的初始化：

初始化时钟、GPIO、滴答定时器和串口1。

②实现用于心跳的定时器：

将滴答定时器作为心跳时钟。

```
static void _ClockInit()
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_USART1 |
RCC_APB2Periph_AFIO, ENABLE);
}

static void _GpioInit()
{
    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}

static void _UartInit()
{
    NVIC_InitTypeDef NVIC_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    USART_InitStructure.USART_BaudRate = 115200;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx |
USART_Mode_Tx;
    USART_Init(USART1, &USART_InitStructure);
    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_Init(&NVIC_InitStructure);
    USART_Cmd(USART1, ENABLE);
}
```

```
void BoardInit()
{
    _ClockInit();
    _GpioInit();
    _UartInit();
}

//滴答定时器
SysTick_Config(SystemCoreClock / TICK_FRQ_HZ);
NVIC_SetPriority(SysTick_IRQn, 0x0);
```

### 3.2 添加BabyOS代码

路径	部分/全部	用于快速体验
bos/algorithm	根据需要添加	暂时不添加其中文件
bos/core	全部添加	全部添加
bos/drivers	根据需要添加	暂时不添加其中文件
bos/hal	全部添加	全部添加
bos/mcu	根据需要添加	添加bos/mcu/st/stm32f10x/ 路径代码
bos/modules	全部添加	全部添加
bos/thirdparty	根据需要添加	添加bos/thirdparty/nr_micro_shell/ 路径代码
bos/utils	全部添加	全部添加
bos/_config		b_config.h 全局 配置文件 b_device_list.h 注册设备的文件 b_hal_if.h 驱动接口文件

编译器添加两个路径即可：

bos/

\_config/ 如果配置文件拷贝到其他路径了，则添加相应路径即可。

### 3.3 修改配置

配置项	说明	用于快速体验
Version Configuration	版本配置项，硬件和固件版本	无改动
Platform Configuration	平台配置项，指定心跳频率和MCU平台	MCU平台选择 STM32F10X_CL
Hal Configuration	硬件接口配置，可配置硬件接口参数是固定还是可变的	无改动
Utils Configuration	实用软件配置，部分软件代码的配置	无改动
Modules Configuration	模块配置项，各个功能模块的配置	无改动
Thirdparty Configuration	第三方开源代码配置项	勾选 NR Micro Shell Enable/Disable

b\_hal\_if.h 中指定DEBUG接口

```
#ifndef __B_HAL_IF_H__
#define __B_HAL_IF_H__
#include "b_config.h"
// debug
#define HAL_LOG_UART B_HAL_UART_1
#endif
```

## 3.4 调用必要的函数

包含头文件 b\_os.h

①滴答定时器中断服务函数调用 bHalIncSysTick();

```
void sysTick_Handler()
{
    bHalIncSysTick();
}
```

②调用 bInit(); bExec();`

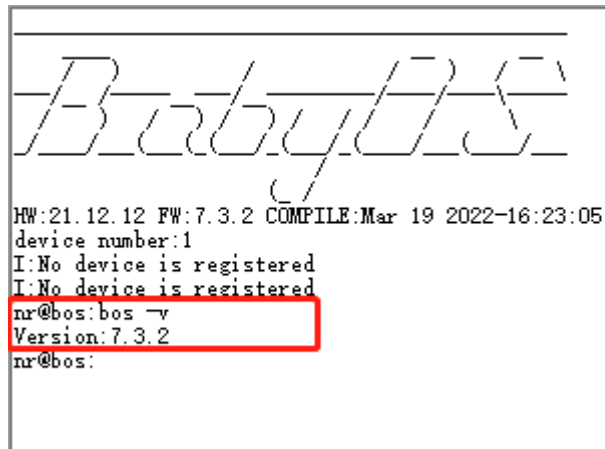
```
int main()
{
    BoardInit();
    SysTick_Config(SystemCoreClock / TICK_FRQ_HZ);
    NVIC_SetPriority(SysTick_IRQn, 0x0);

    bInit();           //bos初始化
    bShellInit();      //shell初始化
    while (1)
    {
        bExec();       //bos执行函数
    }
}
```

③由于勾选了shell功能模块，所以需要在串口接收中断服务函数里调用 `bshellParse`，将数据喂给模块。

## 3.5 快速体验结果

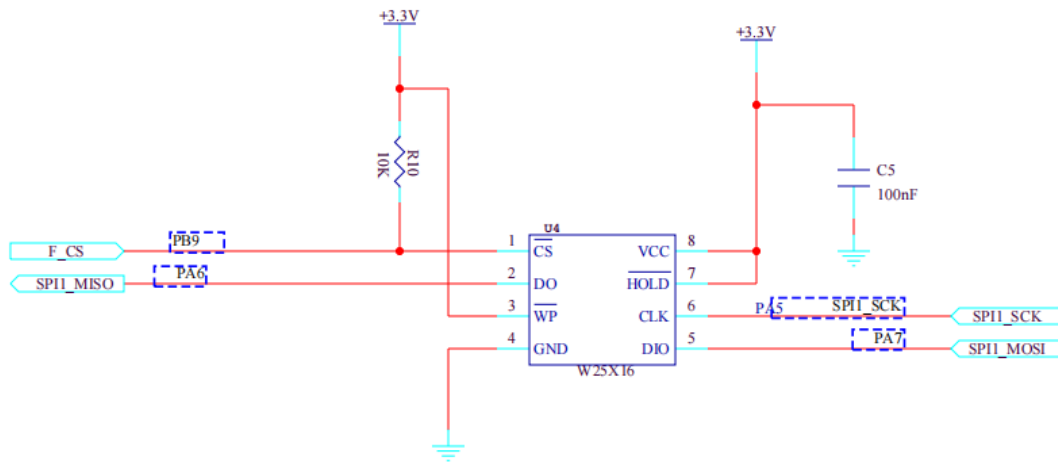
BabyOS的shell模块默认支持查询版本的指令，输入 `bos -v` 便可以查询到版本。



```
BabyOS
HW:21.12.12 FW:7.3.2 COMPILE:Mar 19 2022-16:23:05
device number:1
I:No device is registered
I:No device is registered
nr@bos:bos ~v
Version:7.3.2
nr@bos:
```

## 4. 进阶体验

完成快速体验后，再体验设备的注册和相关操作。以SPIFlash为例进行说明。



### SPI FLASH

#### 4.1 补充MCU资源初始化

在快速体验工程的基础上，增加了硬件SPI，和F\_CS引脚。增加SPI的初始化以及GPIO的初始化。

代码省略....

#### 4.2 添加驱动文件

添加 `bos/drivers/b_drv_spiflash.c`

BabyOS里面SPIFLASH的驱动是基于sfud代码编写。因此也要添加sfud部分的代码。

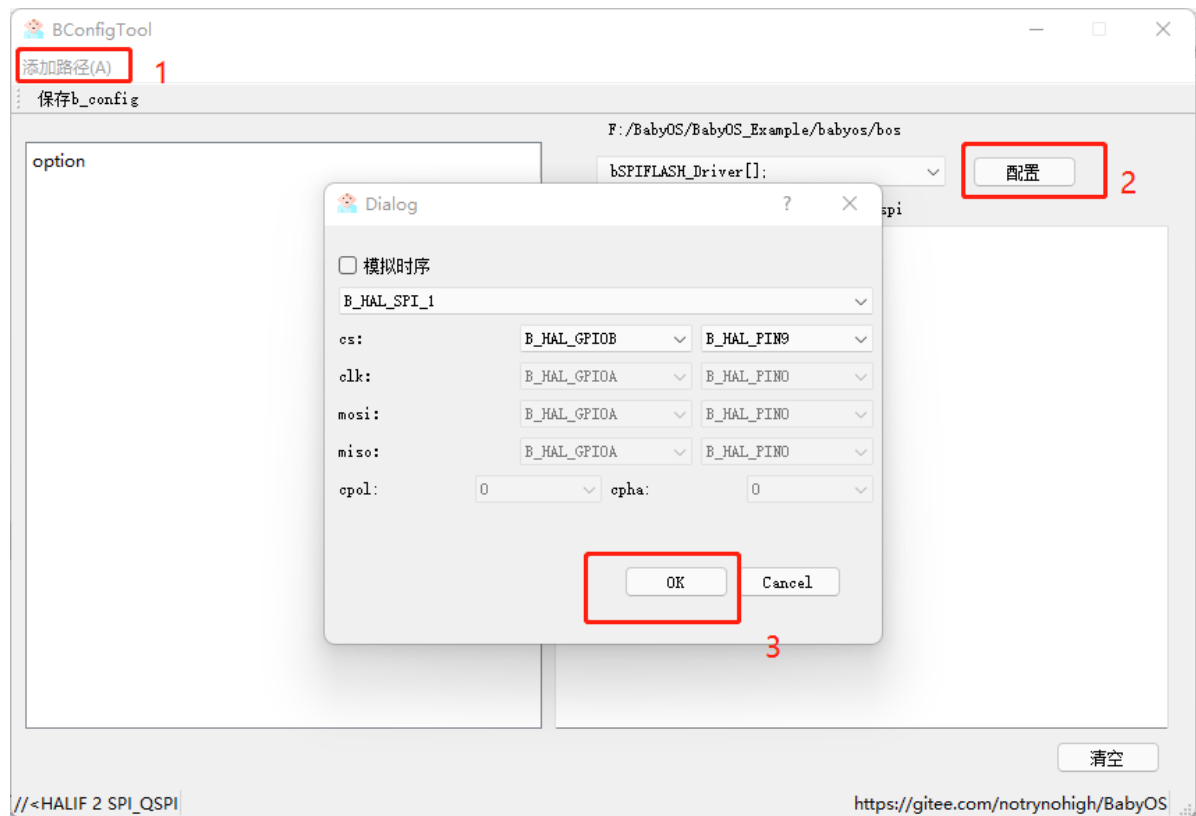
添加 `bos/drivers/sfud/` 路径的代码。

#### 4.3 添加硬件接口

在**hal\_if.h**里面添加硬件接口。

可利用BabyOS配置工具生成代码。 <https://gitee.com/notrynohigh/bconfig-tool/releases/V0.0.2>





由于sfud需要知道有多少个SPIFLASH，所以在 b\_hal\_if.h 里面增加一个宏：

```
#define HAL_SPIFLASH_TOTAL_NUMBER 1
```

## 4.4 记录开机次数

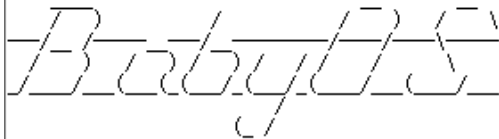
在SPIFLASH的地址0x00000000记录开机次数，增加如下代码

```
int fd = -1;
uint32_t boot_count = 0;
fd = bopen(bSPIFLASH, BCORE_FLAG_RW);
bLseek(fd, 0);
bRead(fd, (uint8_t *)&boot_count, sizeof(boot_count));
b_log("boot:%d\r\n", boot_count);
boot_count += 1;
```

```

bFlashErase_t bFlashErase;
bFlashErase.addr = 0;
bFlashErase.num = 1;
bctl(fd, BCMD_ERASE_SECTOR, &bFlashErase);
blseek(fd, 0);
bwrite(fd, (uint8_t *)&boot_count, sizeof(boot_count));
bclose(fd);

```



HW:21.12.12 FW:7.3.2 COMPILE:Mar 19 2022-19:01:14

device number:2

W:sfud\_init 130 Start initialize Serial Flash Universal Driver(SFUD) V1.1.0.

W:sfud\_init 131 You can get the latest version on <https://github.com/armink/SFUD>.

W:read\_jedec\_id 1026 The flash device manufacturer ID is 0xEF, memory type ID is 0x30, capacity ID is 0x15.

W:read\_sfdp\_header 134 Error: Check SFDP signature error. It's must be 50444653h('S' 'F' 'D' 'P').

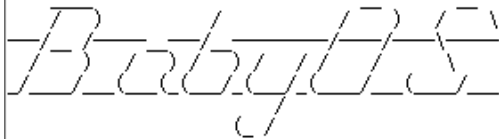
Warning: Read SFDP parameter header information failed. The 000 is not support JEDEC SFDP.

Find a Winbond W25X16BV flash chip. Size is 2097152 bytes.

W:reset 1001 Flash device reset success.

000 flash device is initialize success.

nr@bos:boot:0



HW:21.12.12 FW:7.3.2 COMPILE:Mar 19 2022-19:01:14

device number:2

W:sfud\_init 130 Start initialize Serial Flash Universal Driver(SFUD) V1.1.0.

W:sfud\_init 131 You can get the latest version on <https://github.com/armink/SFUD>.

W:read\_jedec\_id 1026 The flash device manufacturer ID is 0xEF, memory type ID is 0x30, capacity ID is 0x15.

W:read\_sfdp\_header 134 Error: Check SFDP signature error. It's must be 50444653h('S' 'F' 'D' 'P').

Warning: Read SFDP parameter header information failed. The 000 is not support JEDEC SFDP.

Find a Winbond W25X16BV flash chip. Size is 2097152 bytes.

W:reset 1001 Flash device reset success.

000 flash device is initialize success.

nr@bos:boot:1

## 5.详细介绍

### 5.1 添加MCU

`bos/mcu/` 路径是存放已调试过的MCU型号，命名规则是：*bos/mcu/厂商/型号/*。

`bos/hal/` 目录的文件及文件内定义的接口目前并不是很全，这部分的策略是：*一点点添加，上层代码有需要时再添加。*

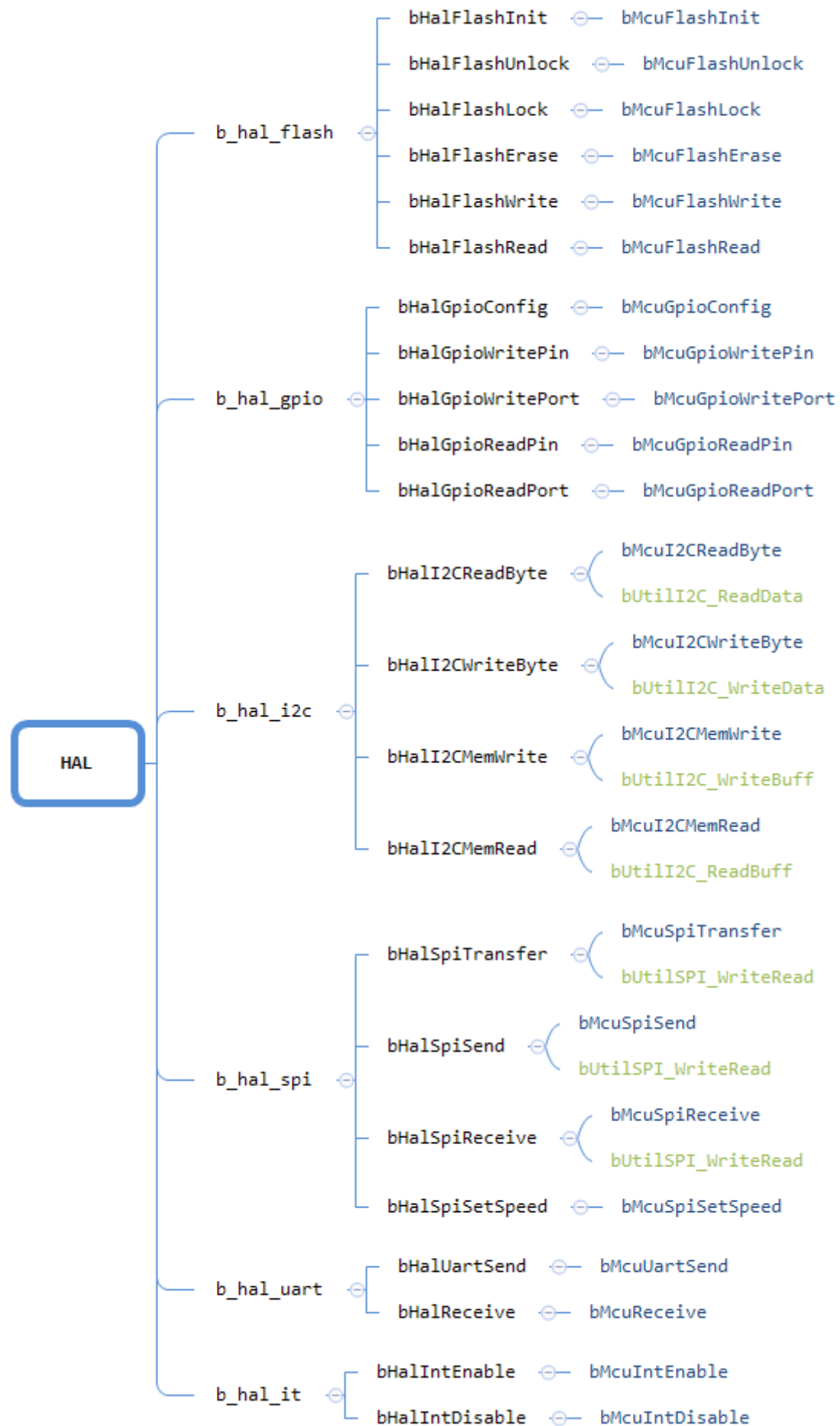
下图中黑色部分是HAL部分的内容，蓝色部分是MCU部分需要实现的，绿色部分是UTILS提供的模拟时序。

SPI和I2C接口支持模拟时序，HAL层判断是否使用模拟时序，然后调用对应接口。

因此新增MCU型号：

- ①新建目录，添加文件
- ②实现蓝色部分的接口
- ③修改 `_config/b_config.h`，为MCU Platform增加一个选项

```
//<o> MCU Platform
//<1001=> STM32F10X_LD
//<1002=> STM32F10X_MD
//<1003=> STM32F10X_HD
//<1004=> STM32F10X_CL
//<1101=> STM32G0X0
//<2001=> NATION_F40X
//<3001=> MM32SPIN2X
//<3002=> MM32SPIN0X
//<4001=> HC32L13X
//<7001=> CH32F103
#define MCU_PLATFORM 1004
```



## 5.2 HAL层介绍

Hal层一方面是给MCU层提供统一的接口。还有如下几点作用：

- ①提供心跳时间的查询
- ②提供微秒级和毫秒级延时函数
- ③提供通讯接口的数据结构

### 5.2.1 心跳时钟

使用BabyOS,需要给予一个心跳时钟。心跳时钟的频率在 `_config/b_config.h` 里定义 `TICK_FRQ_HZ` , 使用者自行实现一个定时器, 并定时调用 `bHalIncSysTick` 。应用代码中可根据 `bHalGetSysTick` 获取心跳时钟计数值。

### 5.2.2 延时函数

提供 `bHalDelayMs` 和 `bHalDelayUs` 两个阻塞型延时函数。毫秒级延时是通过心跳计算的。微妙级函数是通过for循环阻塞。 `bHalInit` 中会计算微妙级延时所用到的参数, 以此尽量保证微妙级函数的精准性。

### 5.2.3 通讯接口

HAL层提供通讯接口的数据结构：

```
//GPIO
typedef struct
{
    bHalGPIOPort_t port;
    bHalGPIOPin_t pin;
} bHalGPIOInstance_t;

//I2C
typedef struct
{
    uint8_t dev_addr;
    uint8_t is_simulation;
    union
    {
        bHalI2CNumber_t i2c;
        struct
        {
            bHalGPIOInstance_t clk;
            bHalGPIOInstance_t sda;
        } simulating_i2c;
    } _if;
} bHalI2CIf_t;

//SPI
typedef struct
{
    uint8_t is_simulation;
    union
    {
        bHalSPINumber_t spi;
```

```

        struct
        {
            bHalGPIOInstance_t miso;
            bHalGPIOInstance_t mosi;
            bHalGPIOInstance_t clk;
            uint8_t          CPOL;
            uint8_t          CPHA;
        } simulating_spi;
    } _if;
    bHalGPIOInstance_t cs;
} bHalSPIIf_t;

//UART
typedef enum
{
    B_HAL_UART_1,
    B_HAL_UART_2,
    ....
    B_HAL_UART_NUMBER
} bHalUartNumber_t;

//LCD
typedef struct
{
    union
    {
        uint32_t rw_addr;
        struct
        {
            bHalGPIOInstance_t data;
            bHalGPIOInstance_t rs;
            bHalGPIOInstance_t rd;
            bHalGPIOInstance_t wr;
            bHalGPIOInstance_t cs;
        } _io;
        struct
        {
            bHalGPIOInstance_t rs;
            bHalSPIIf_t        _spi;
        } _spi;
    } _if;
    uint8_t if_type; // 0: _io 1: rw_addr 2: _spi
} bLCD_HalIf_t;

```

## 5.3 驱动层介绍

BabyOS操作设备的方式是，打开、读/写/控制，关闭。这么设计的想法是，打开（唤醒设备）然后对设备进行操作（读/写/配置），最后关闭（休眠设备）。

bos/driver/inc/b\_driver.h 里有一个数据结构：

```
typedef struct bDriverIf
{
    int status;
    int (*init)(void);
    int (*open)(struct bDriverIf *pdrv);
    int (*close)(struct bDriverIf *pdrv);
    int (*ctl)(struct bDriverIf *pdrv, uint8_t cmd, void *param);
    int (*write)(struct bDriverIf *pdrv, uint32_t offset, uint8_t *pbuf,
uint16_t len);
    int (*read)(struct bDriverIf *pdrv, uint32_t offset, uint8_t *pbuf, uint16_t
len);
    void *_hal_if;
    union
    {
        uint32_t v;
        void * _p;
    } _private;
} bDriverInterface_t;
```

每个驱动文件的目标便是实现 bDriverInterface\_t 里面的各个元素。

**status** 驱动初始化异常则将 status 设为-1 反之设 为 0。操作设备时检测此项，如果是-1 则不执行。

**init** 负责执行初始化，用于运行过程中再次初始化。

**open** 负责唤醒操作，此处可执行设备唤醒。如果设备没有休眠状态，可以赋值为 NULL。

**close** 负责休眠的操作，此处可执行设备休眠。如果设备没有休眠状态，可以赋值为 NULL。

**ctl** 负责对设备进行配置或者执行特定的操作，例如擦除，切换状态等。ctl 的调用需要传入指令 cmd 和对应的参数。执行成功返回 0，失败或者不支持指令则返回-1。

**write** 负责传输数据至设备，执行成功返回实际发送的数据长度，执行失败则返回-1。

**read** 负责从设备获取数据，获取数据的最小单元依据设备功能而定，例如，存储设备，最小可以获取 1 个字节；3 轴加速度设备，最小单元为 3 个加速度值；温湿度传感器最小单元是一组温度湿度值。读取的最小单元需 要在驱动的 h 文件进行说明让使用者能明白。

**\_hal\_if** 指向当前驱动对应的硬件接口。

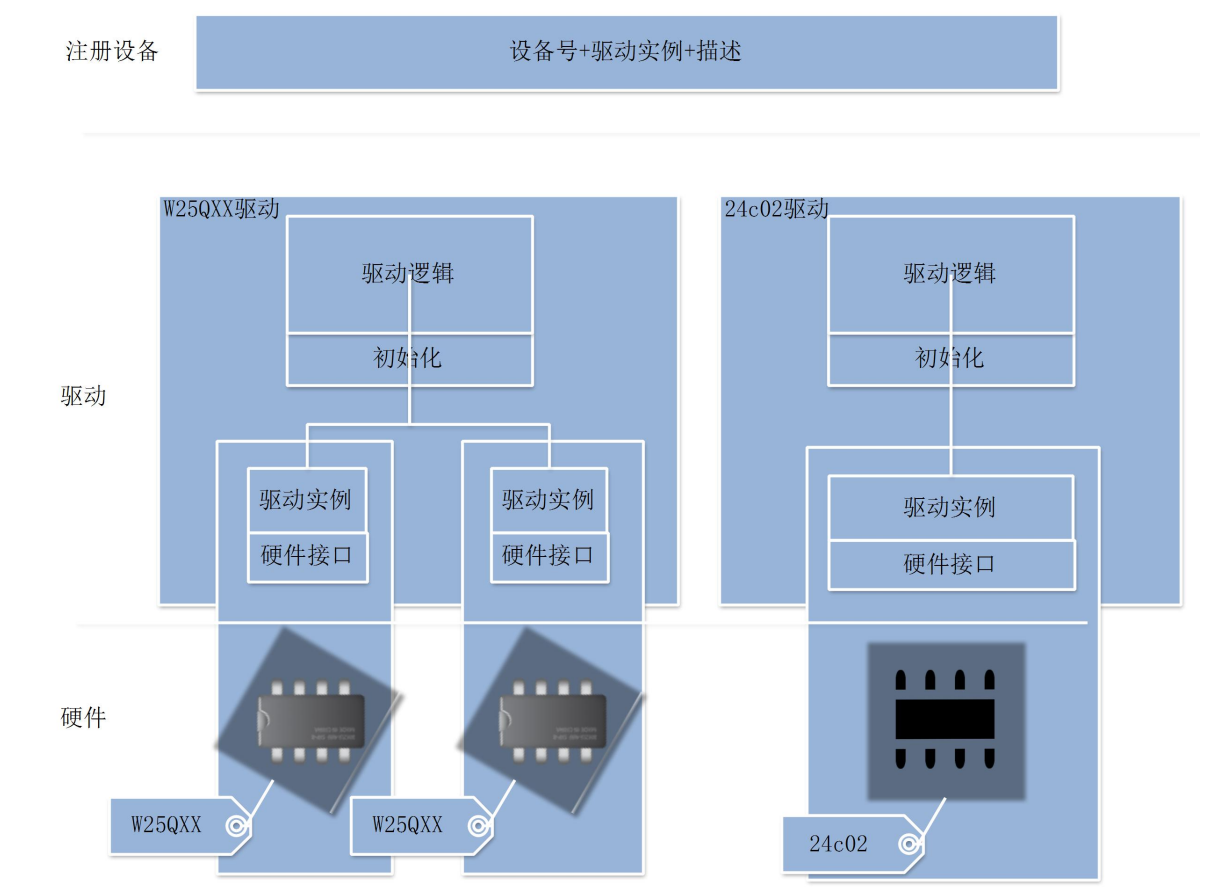
**private** 当驱动需要携带私有参数时，则利用这个字段。例如 flash 的 id，可以放在 private.v。如果需 要存放更多的信息，那么就利用private.p 指向一片数据区域。

### 5.3.1 硬件接口

每个驱动的硬件接口通过 HAL\_XXXX\_IF 指定，在驱动文件代码中会有如下一行代码：

```
HALIF_KEYWORD bXXXX_HalIf_t bXXXX_HalIfTable[] = HAL_XXXX_IF;
//或
HALIF_KEYWORD bXXXX_HalIf_t bXXXX_HalIf = HAL_XXXX_IF;
```

这两种分别对应哪种情况呢，通过下图可以看出。



硬件接口的两种情况：

- 1) 存在有相同设备的情况，例如接入 MCU 的有两个 Flash 芯片
- 2) 当前驱动对应的设备不会存在多个，例如屏，一般只会接 1 块屏

第一种情况时，可通过如下宏获取当前的硬件接口：

```
#define bDRV_GET_HALIF(name, type, pdrv) type *name = (type *)((pdrv)->_hal_if)
//例如: bDRV_GET_HALIF(_if, bSPIFLASH_HalIf_t, pdrv);
//_if便是指向硬件接口的指针
```

### 5.3.2 注册设备

操作设备是通过设备号进行，那么注册设备便是将设备号与驱动实例进行绑定。

bos/driver/inc/b\_driver.h 列出了已有的驱动实例。

b\_device\_list.h 中通过宏进行注册：

```
B_DEVICE_REG(bSPIFLASH, bSPIFLASH_Driver[0], "spiflash")
B_DEVICE_REG(bILI9341, bILI9341_Driver, "ili9341")
```

设备管理涉及到以下几个数据结构：

```
#define B_DEVICE_REG(dev, driver, desc)
#include "b_device_list.h"

typedef enum
{
```



```

#define B_DEVICE_REG(dev, driver, desc) dev,
#include "b_device_list.h"
    bDEV_NULL,
    bDEV_MAX_NUM
} bDeviceName_t;

static bDriverInterface_t bNullDriver;
static bDriverInterface_t *bDriverTable[bDEV_MAX_NUM] = {
#define B_DEVICE_REG(dev, driver, desc) &driver,
#include "b_device_list.h"
    &bNullDriver,
};

static const char *bDeviceDescTable[bDEV_MAX_NUM] = {
#define B_DEVICE_REG(dev, driver, desc) desc,
#include "b_device_list.h"
    "null",
};

```

设备注册便是填充了 `bDeviceName_t` `bDriverTable` `bDeviceDescTable`，以设备号为索引，可以从 `bDriverTable` 找到对应的驱动实例，从 `bDeviceDescTable` 中找到设备的描述。

### 5.3.3 操作设备

```

int breinit(uint8_t dev_no);
int bOpen(uint8_t dev_no, uint8_t flag);
int bRead(int fd, uint8_t *pdata, uint16_t len);
int bwrite(int fd, uint8_t *pdata, uint16_t len);
int bCtl(int fd, uint8_t cmd, void *param);
int bLseek(int fd, uint32_t off);
int bClose(int fd);
int bModifyHalIf(uint8_t dev_no, uint32_t type_size, uint32_t off, const uint8_t
*pval,
                uint8_t len);

```

**dev\_no** 注册设备时指定的设备号

**fd** 打开设备后返回的句柄，最多同时打开10（`BCORE_FD_MAX`）个设备。

配置项 `_HALIF_VARIABLE_ENABLE` 用于配置是否允许硬件接口可以改动。

```

#if _HALIF_VARIABLE_ENABLE
#define HALIF_KEYWORD static
#else
#define HALIF_KEYWORD const static
#endif

```

`bModifyHalIf` 使用例子：

```
//oled硬件接口数据结构是 boLED_HalIf_t
typedef struct
{
    union
    {
        bHalI2CIf_t _i2c;
        bHalSPIIf_t _spi;
    } _if;
    uint8_t is_spi;
} boLED_HalIf_t;
// 修改IIC的设备地址 OLED是注册的设备号，dev_addr变量存放着新的指。
bModifyHalIf(OLED, sizeof(boLED_HalIf_t), (uint8_t)(&(((boLED_HalIf_t *)0)-
>_if._i2c.dev_addr)), &dev_addr, 1)
```

## 5.4 SECTION介绍

b\_section.h 定义段的操作。现有的段有如下几个：

```
bSECTION_DEF_FLASH(bos_polling, pbPoling_t);
#define BOS_REG_POLLING_FUNC(func) //将func放入bos_polling段

bSECTION_DEF_FLASH(driver_init_0, pbDriverInit_t);
bSECTION_DEF_FLASH(driver_init, pbDriverInit_t);
#define bDRIVER_REG_INIT_0(func) //将func放入driver_init_0段
#define bDRIVER_REG_INIT(func) //将func放入driver_init段

bSECTION_DEF_FLASH(b_mod_shell, static_cmd_st);
#define bSHELL_REG_INSTANCE(cmd_name, cmd_handler) //将cmd信息放入b_mod_shell段

bSECTION_DEF_FLASH(b_mod_param, bParamInstance_t);
#define bPARAM_REG_INSTANCE(param, param_size) //将参数信息放入b_mod_param段
```

驱动文件最后会有一行这样的代码：`bDRIVER_REG_INIT(bXXXX_Init);`将初始化函数放入 `driver_init` 段。

```
//设备初始化时，将遍历driver_init_0和driver_init内的函数，并执行。
int bDeviceInit()
{
    memset(&bNullDriver, 0, sizeof(bNullDriver));
    bSECTION_FOR_EACH(driver_init_0, pbDriverInit_t, pdriver_init_0)
    {
        (*pdriver_init_0)();
    }
    bSECTION_FOR_EACH(driver_init, pbDriverInit_t, pdriver_init)
    {
        (*pdriver_init)();
    }
    return 0;
}
```

```
int bExec()
{
    //BabyOS的执行函数遍历需要轮询的函数即在bos_polling段的函数。
    bSECTION_FOR_EACH(bos_polling, pbPoling_t, polling)
    {
        (*polling)();
    }
    return 0;
}
```

当使用gcc编译时，需要编辑链接文件，在链接文件中补充这几个段，例如：

```
/* Define output sections */
SECTIONS
{
    .....
    /* BabyOS Section -----*/
    .driver_init :
```

```
{
    . = ALIGN(4);
    PROVIDE(__start_driver_init = .);
    KEEP(*(SORT(.driver_init*)))
    PROVIDE(__stop_driver_init = .);
    . = ALIGN(4);
} > FLASH

.driver_init_0 :
{
    . = ALIGN(4);
    PROVIDE(__start_driver_init_0 = .);
    KEEP(*(SORT(.driver_init_0*)))
    PROVIDE(__stop_driver_init_0 = .);
    . = ALIGN(4);
} > FLASH

.bos_polling :
{
    . = ALIGN(4);
    PROVIDE(__start_bos_polling = .);
    KEEP(*(SORT(.bos_polling*)))
    PROVIDE(__stop_bos_polling = .);
    . = ALIGN(4);
} > FLASH

.b_mod_shell :
{
    . = ALIGN(4);
    PROVIDE(__start_b_mod_shell = .);
    KEEP(*(SORT(.b_mod_shell*)))
    PROVIDE(__stop_b_mod_shell = .);
    . = ALIGN(4);
} > FLASH

/* BabyOS Section -----end-----*/
.....
}
```

## 5.5 功能组件

功能组件包括: 功能模块、第三方开源代码，算法模块和工具模块。

组件	描述	代码
功能模块	收集BabyOS开发者编写的通用软件模块	b_mod_adchub b_mod_button b_mod_error b_mod_event b_mod_fs b_mod_gui b_mod_kv b_mod_menu b_mod_modbus b_mod_param b_mod_protocol b_mod_pwm b_mod_sda b_mod_sdb b_mod_shell b_mod_timer b_mod_trace b_mod_xm128 b_mod_ymodem
第三方开源	收集第三方实用的开源代码	cjson cm_backtrace fatfs flexiblebutton littlefs nr_micro_shell ugui sfud
算法模块	收集常用的算法。目前这部分处于空白状态	
工具模块	支持其他各模块的通用代码	b_util_at b_util_fifo b_util_i2c b_util_log b_util_lunar b_util_memp b_util_spi b_util_uart b_util_utc

组件的每个部分都可以通过全局配置文件使能以及配置参数。组件中的代码，操作MCU资源只能调用HAL层接口，操作设备只能基于设备号进行操作。

组件中每个c文件功能单一,提供的功能接口放在对应的h文件。尽量做到，根据h文件的函数名便知道如何使用。

## 6. 参与开发

---

目前还需要广大开源爱好者的加入，将货架做稳固，再填充高质量的货物。

<https://gitee.com/notrynohigh/BabyOS>（主仓库）

<https://github.com/notrynohigh/BabyOS>（自动同步）

管理员邮箱：[notrynohigh@outlook.com](mailto:notrynohigh@outlook.com)

开发者基于<https://gitee.com/notrynohigh/BabyOS>仓库dev分支进行。

有贡献的开发者（不局限于提交代码），记录到<http://babyos.cn/>网站Team页面。

有意者随时私信联系！