

---

title: 纯C语言写的按键驱动，将按键逻辑与按键处理事件分离~ author: 杰杰 top: false cover: false toc: true  
mathjax: false date: 2019-10-05 10:03:27 img: coverImg: password: summary: tags:

- 框架
  - ButtonDrive
  - STM32
  - 开源项目 categories: 框架
- 

## ButtonDrive

---

自己写的一个按键驱动，支持单双击、连按、长按；采用回调处理按键事件（自定义消抖时间），使用只需3步，创建按键，按键事件与回调处理函数链接映射，周期检查按键。源码地址：

<https://github.com/jiejieTop/ButtonDrive>

### 前言

前几天写了个按键驱动，参考了[MulitButton](#)的数据结构的用法，逻辑实现并不一样。在这里感谢所有的开源开发者，让我从中学到了很多，同时网络也是一个好平台，也希望所有的开发者能形成良性循环，从网络中学知识，回馈到网络中去。感谢[MulitButton](#)的作者[0x1abin](#)，感谢两位rtt的大佬：[大法师](#)、[流光](#)。

### Button\_drive简介

Button\_drive是一个小巧的按键驱动，支持单击、双击、长按、连续触发等（后续可以在按键控制块中添加触发事件），理论上可无限量扩展Button，Button\_drive采用按键触发事件回调方式处理业务逻辑，支持在RTOS中使用，我目前仅在[RT-Thread](#)上测试过。写按键驱动的目的是想要将用户按键逻辑与按键处理事件分离，用户无需处理复杂麻烦的逻辑事件。

### Button\_drive使用效果

#### 1. 单击与长按

```
[INFO] >> welcome to learn jiejie stm32 library!
```

```
[INFO] >> button struct information:
```

```
    btn->Name:Button1
    btn->Button_State:6
    btn->Button_Trigger_Event:6
    btn->Button_Trigger_Level:1
    btn->Button_Last_Level:0
```

```
[INFO] >> button node have Button1
```

```
[INFO] >> button struct information:
```

```
    btn->Name:Button2
    btn->Button_State:6
    btn->Button_Trigger_Event:6
    btn->Button_Trigger_Level:1
    btn->Button_Last_Level:0
```

单击与长按

```
[INFO] >> button node have Button2
```

```
[INFO] >> button node have Button1|
```

```
[INFO] >> Button1 单击!
```

```
[INFO] >> Button1 单击!
```

```
[INFO] >> Button1 单击!
```

```
[INFO] >> Button2 单击!
```

```
[INFO] >> Button2 单击!
```

```
[INFO] >> Button2 单击!
```

```
[INFO] >> Button2 长按!
```

```
[INFO] >> Button2 长按!
```

```
[INFO] >> Button1 长按!
```

```
[INFO] >> 删除Button1
```

```
[INFO] >> button node have Button1
```

<https://jiejie.blog.csdn.net>

## 2. 双击

```
[INFO] >> welcome to learn jiejie stm32 library!
```

```
[INFO] >> button struct information:
```

```
    btn->Name:Button1
    btn->Button_State:6
    btn->Button_Trigger_Event:6
    btn->Button_Trigger_Level:1
    btn->Button_Last_Level:0
```

```
[INFO] >> button node have Button1
```

```
[INFO] >> button struct information:
```

```
    btn->Name:Button2
    btn->Button_State:6
    btn->Button_Trigger_Event:6
    btn->Button_Trigger_Level:1
    btn->Button_Last_Level:0
```

```
[INFO] >> button node have Button2
```

```
[INFO] >> button node have Button1
```

```
[INFO] >> Button1 双击!
```

```
[INFO] >> Button2 双击!
```

```
[INFO] >> Button1 双击!
```

```
[INFO] >> Button1 双击!
```

```
[INFO] >> Button1 双击!
```

```
[INFO] >> Button1 双击!
```

```
[INFO] >> Button1 双击!
```

```
[INFO] >> Button2 双击!
```

```
[INFO] >> Button2 长按!
```

```
[INFO] >> Button1 长按!
```

```
[INFO] >> 删除Button1
```

```
[INFO] >> button node have Button1|
```

<https://jiejie.blog.csdn.net>

## 3. 连接

[illegible]

#### 4. 连按释放

[illegible]

## 使用方法

## 1. 创建按键句柄

```
Button_t Button1;  
Button_t Button2;
```

2. 创建按键，初始化按键信息，包括按键名字、按键电平检测函数接口、按键触发电平。

```

Button_Create("Button1",                // 按键名字
              &Button1,                 // 按键句柄
              Read_Button1_Level,        // 按键电平检测函数接口
              BTN_TRIGGER);              // 触发电平

.....

```

3. 按键触发事件与事件回调函数链接映射，当按键事件被触发的时候，自动跳转回调函数中处理业务逻辑。

```

Button_Attach(&Button1,BUTTON_DOWNM,Btn2_Downm_Callback);           // 按键单击
Button_Attach(&Button1,BUTTON_DOUBLE,Btn2_Double_Callback);         // 双击
Button_Attach(&Button1,BUTTON_LONG,Btn2_Long_Callback);              // 长按

.....

```

4. 周期调用回调按键处理函数即可，建议调用周期20-50ms。

```

Button_Process();           // 需要周期调用按键处理函数

```

需要用户实现的 **2** 个函数：

- 按键电平检测接口：

```

uint8_t Read_Button1_Level(void)
{
    return GPIO_ReadInputDataBit(BTN1_GPIO_PORT,BTN1_GPIO_PIN);
}

uint8_t Read_Button2_Level(void)
{
    return GPIO_ReadInputDataBit(BTN2_GPIO_PORT,BTN2_GPIO_PIN);
}

// 这是我在stm32上简单测试的伪代码，以实际源码为准

```

- 按键逻辑处理

```

void Btn1_Downm_Callback(void *btn)
{
    PRINT_INFO("Button1 单击!");
}

```

```

void Btn1_Double_CallBack(void *btn)
{
    PRINT_INFO("Button1 双击!");
}

void Btn1_Long_CallBack(void *btn)
{
    PRINT_INFO("Button1 长按!");

    Button_Delete(&Button2);
    PRINT_INFO("删除Button1");
    Search_Button();
}

```

## 特点

Button\_drive开放源码，按键控制块采用数据结构方式，按键事件采用枚举类型，确保不会重复，也便于添加用户需要逻辑，采用宏定义方式定义消抖时间、连按触发时间、双击时间间隔、长按时间等，便于修改。同时所有被创建的按键采用单链表方式连击，用户只管创建，无需理会按键处理，只需调用[Button\\_Process\(\)](#)即可，在函数中会自动遍历所有被创建的按键。支持按键删除操作，用户无需在代码中删除对应的按键创建于映射链接代码，也无需删除关于按键的任何回调事件处理函数，只需调用[Button\\_Delete\(\)](#)函数即可，这样子，就不会处理关于被删除按键的任何状态。当然目前按键内存不会释放，如果使用os的话，建议释放按键内存。

### 按键控制块

```

/*
    每个按键对应1个全局的结构体变量。
    其成员变量是实现消抖和多种按键状态所必须的
*/
typedef struct button
{
    /* 下面是一个函数指针，指向判断按键是否按下的函数 */
    uint8_t (*Read_Button_Level)(void); /* 读取按键电平函数，需要用户实现 */

    char Name[BTN_NAME_MAX];

    uint8_t Button_State           : 4;      /* 按键当前状态（按下还是弹起） */
    uint8_t Button_Last_State     : 4;      /* 上一次的按键状态，用于判断双击 */
    /*
    uint8_t Button_Trigger_Level   : 2;      /* 按键触发电平 */
    uint8_t Button_Last_Level     : 2;      /* 按键当前电平 */

    uint8_t Button_Trigger_Event; /* 按键触发事件，单击，双击，长按等 */

    Button_Callback Callback_Function[number_of_event];
    uint8_t Button_Cycle;         /* 连续按键周期 */

    uint8_t Timer_Count;          /* 计时 */
    uint8_t Debounce_Time;        /* 消抖时间 */
    */
}

```

```
uint8_t Long_Time;          /* 按键按下持续时间 */

struct button *Next;

}Button_t;
```

## 触发事件

```
typedef enum {
    BUTTON_DOWNM = 0,
    BUTTON_UP,
    BUTTON_DOUBLE,
    BUTTON_LONG,
    BUTTON_CONTINUOUS,
    BUTTON_CONTINUOUS_FREE,
    BUTTON_ALL_TRIGGER,
    number_of_event, /* 触发回调的事件 */
    NONE_TRIGGER
}Button_Event;
```

## 宏定义选择

```
#define BTN_NAME_MAX 32      //名字最大为32字节

/* 按键消抖时间40ms，建议调用周期为20ms
   只有连续检测到40ms状态不变才认为有效，包括弹起和按下两种事件
*/

#define CONTINUOUS_TRIGGER 0 //是否支持连续触发，连发的话就不要检测单双击
与长按了

/* 是否支持单击&双击同时存在触发，如果选择开启宏定义的话，单双击都回调，只不过单击会延迟
响应，
   因为必须判断单击之后是否触发了双击否则，延迟时间是双击间隔时间 BUTTON_DOUBLE_TIME。
   而如果不开启这个宏定义，建议工程中只存在单击/双击中的一个，否则，在双击响应的时候会触
发一次单击，
   因为双击必须是有一次按下并且释放之后才产生的 */
#define SINGLE_AND_DOUBLE_TRIGGER 1

/* 是否支持长按释放才触发，如果打开这个宏定义，那么长按释放之后才触发单次长按，
   否则在长按指定时间就一直触发长按，触发周期由 BUTTON_LONG_CYCLE 决定 */
#define LONG_FREE_TRIGGER 0

#define BUTTON_DEBOUNCE_TIME 2 //消抖时间 (n-1)*调用周期
#define BUTTON_CONTINUOUS_CYCLE 1 //连按触发周期时间 (n-1)*调用周期
```

```

#define BUTTON_LONG_CYCLE      1          //长接触发周期时间 (n-1)*调用周期
#define BUTTON_DOUBLE_TIME     15         //双击间隔时间 (n-1)*调用周期 建议在200-600ms
#define BUTTON_LONG_TIME       50         /* 持续n秒((n-1)*调用周期ms), 认为长按事件 */

#define TRIGGER_CB(event) \
    if(btn->CallBack_Function[event]) \
        btn->CallBack_Function[event]((Button_t*)btn)

```

## 例子

```

Button_Create("Button1",
              &Button1,
              Read_KEY1_Level,
              KEY_ON);
Button_Attach(&Button1, BUTTON_DOWM, Btn1_Downm_CallBack); //
单击
Button_Attach(&Button1, BUTTON_DOUBLE, Btn1_Double_CallBack); //
双击
Button_Attach(&Button1, BUTTON_CONTINUOS, Btn1_Continuos_CallBack); //
连接
Button_Attach(&Button1, BUTTON_CONTINUOS_FREE, Btn1_ContinuosFree_CallBack); //
连接释放
Button_Attach(&Button1, BUTTON_LONG, Btn1_Long_CallBack); //
长按

Button_Create("Button2",
              &Button2,
              Read_KEY2_Level,
              KEY_ON);
Button_Attach(&Button2, BUTTON_DOWM, Btn2_Downm_CallBack); //单
击
Button_Attach(&Button2, BUTTON_DOUBLE, Btn2_Double_CallBack); //双
击
Button_Attach(&Button2, BUTTON_CONTINUOS, Btn2_Continuos_CallBack); //连
按
Button_Attach(&Button2, BUTTON_CONTINUOS_FREE, Btn2_ContinuosFree_CallBack); //连
按释放
Button_Attach(&Button2, BUTTON_LONG, Btn2_Long_CallBack); //长
按

Get_Button_Event(&Button1);
Get_Button_Event(&Button2);

```

## 后续

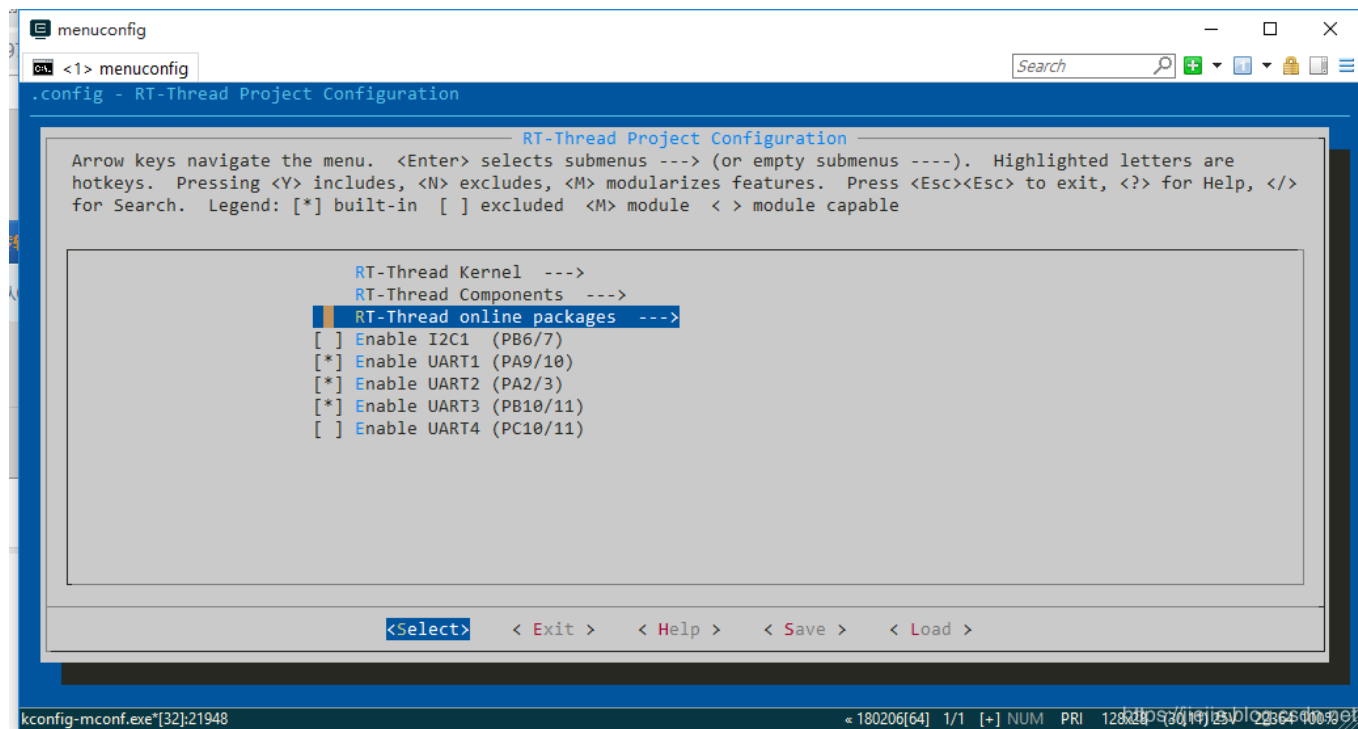
流光大佬的要求，让我玩一玩RTT的rtkpgs，打算用Button\_drive练一练手吧。

## ButtonDrive在env使用

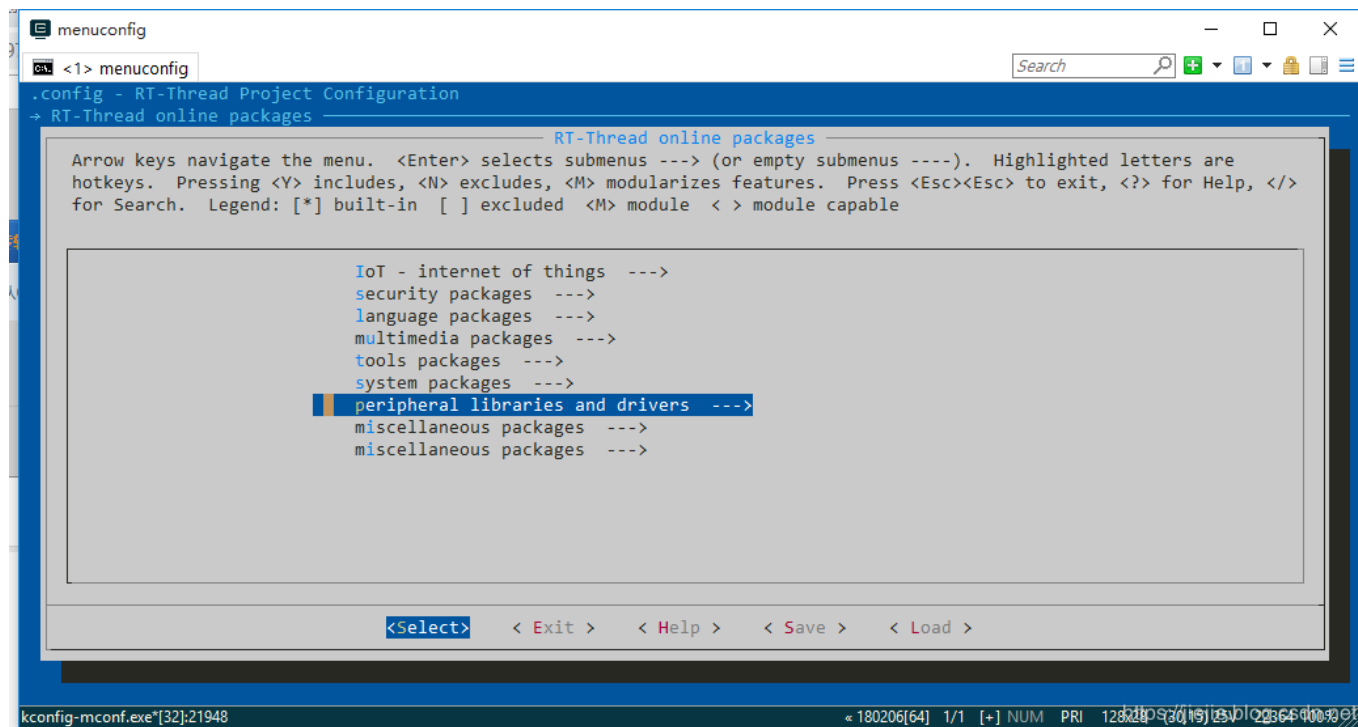
目前我已将按键驱动做成软件包（packages），如果使用RT-Thread操作系统的话，可以在env中直接配置使用！

步骤如下：

### 1. 选择在线软件包

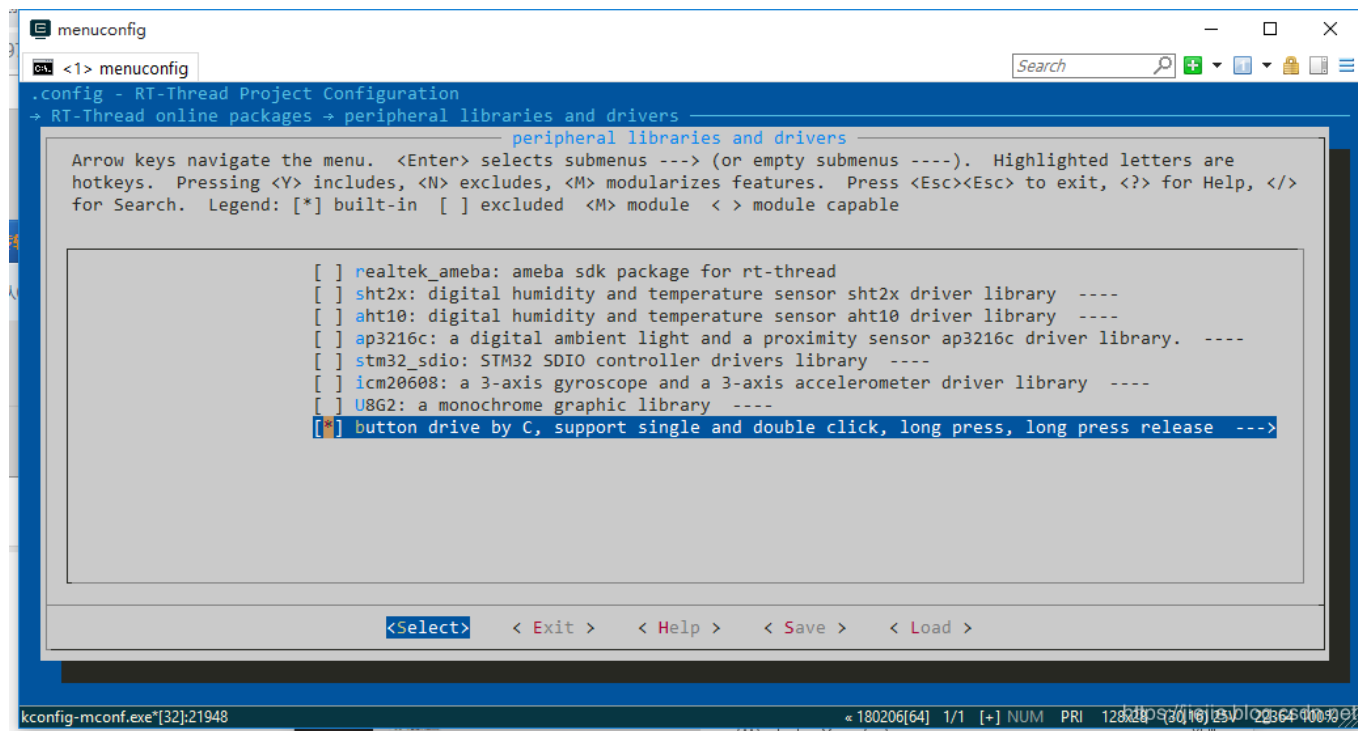


### 2. 选择软件包属性为外设相关

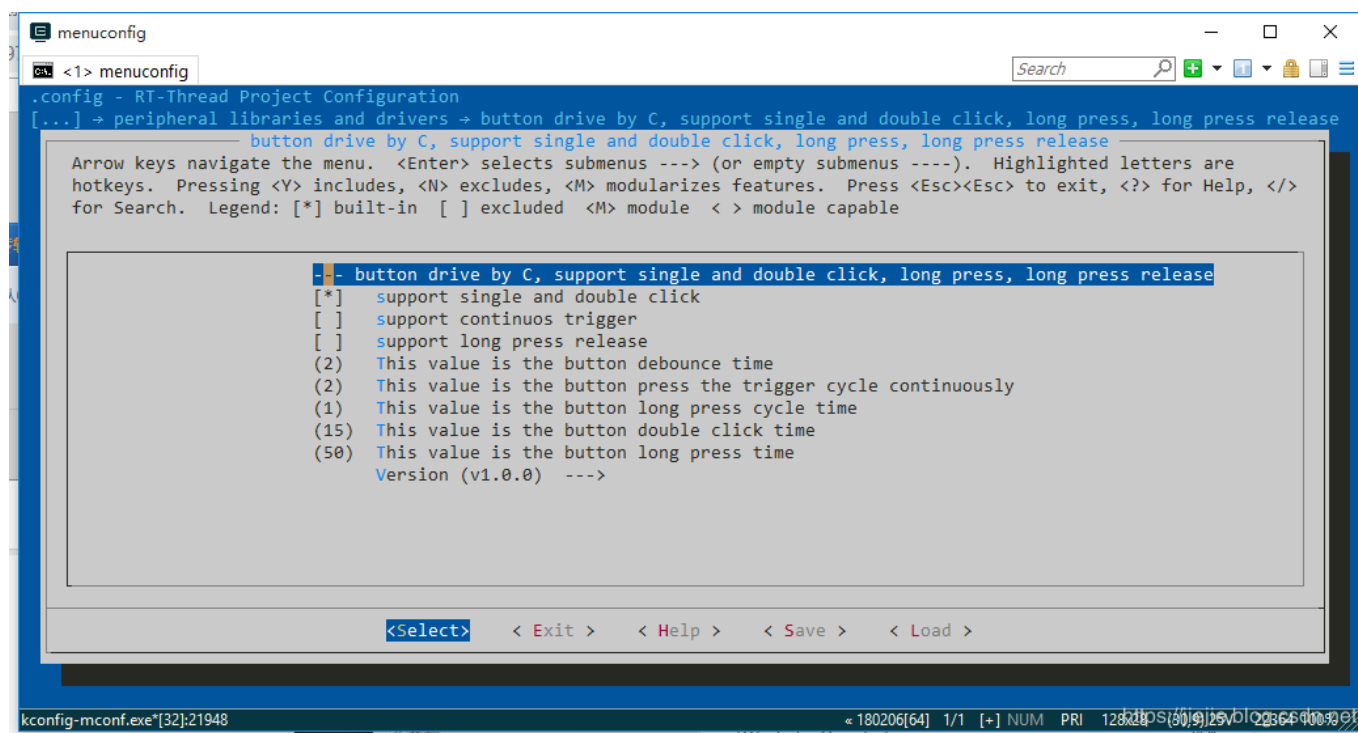


### 3. 选择button\_drive

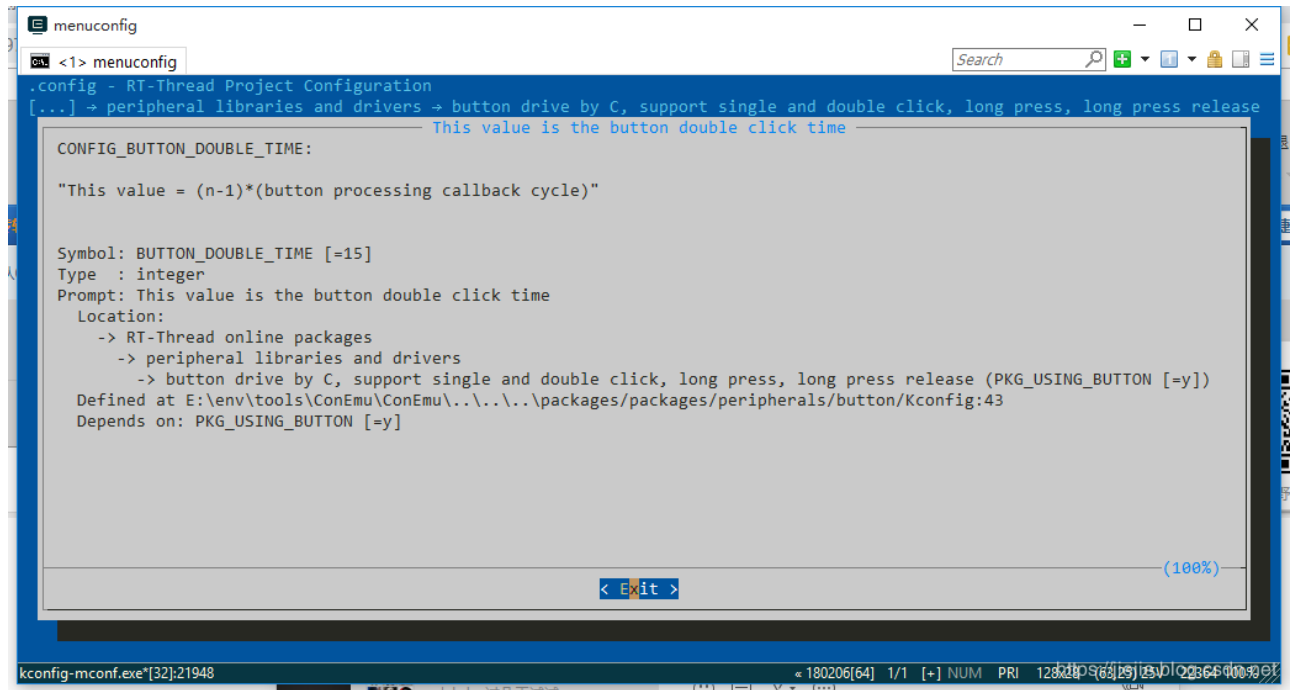




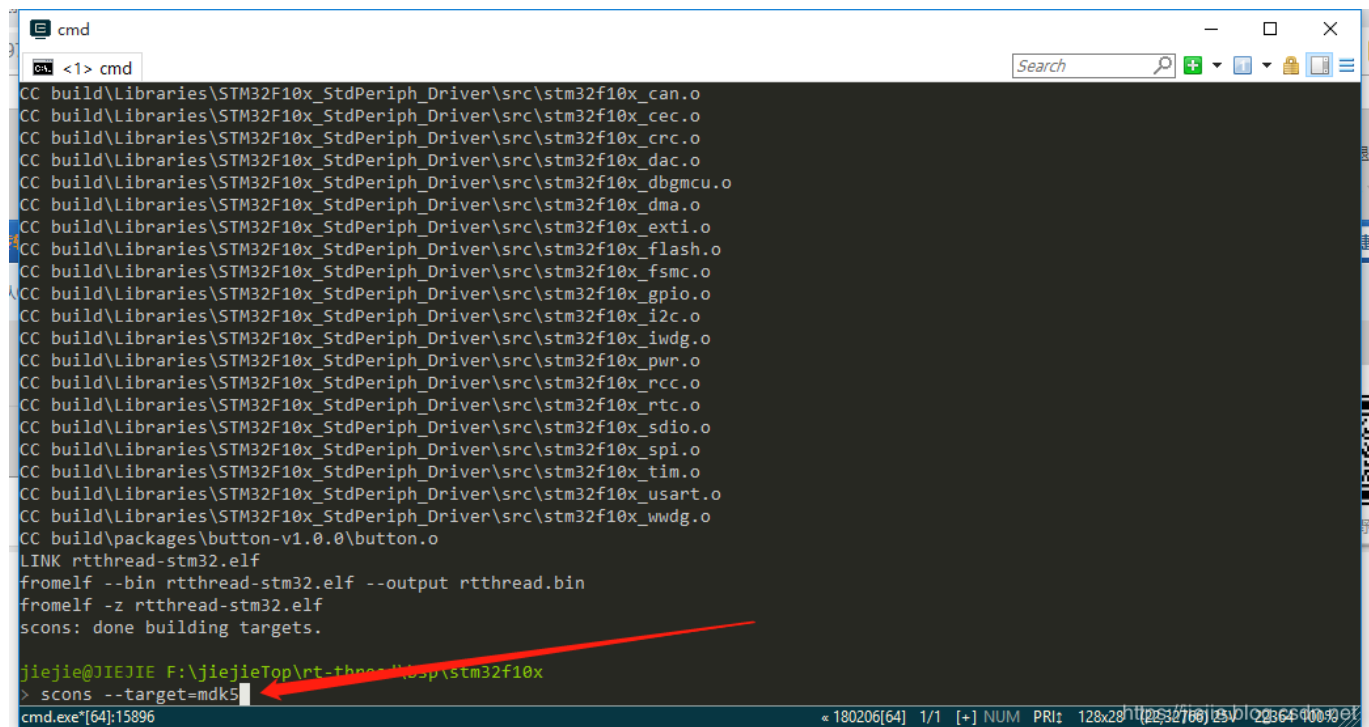
#### 4. 进入驱动选项配置（自带默认属性）



5. 如果不懂按键的配置是什么意思，按下“**shift+?**”，即可有解释

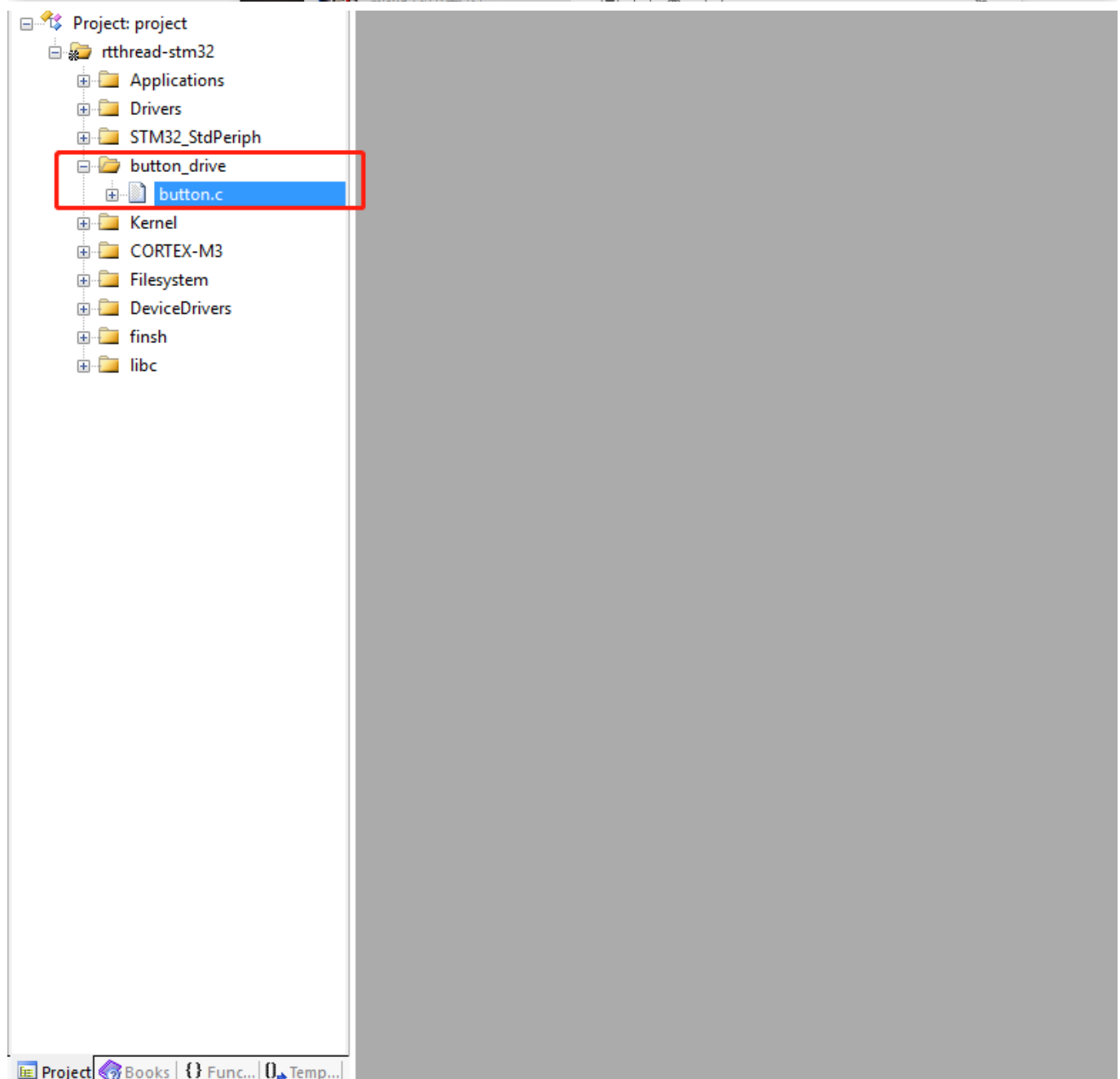


6. 编译生成mdk/iar工程



```
cmd
C:\> cmd
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_can.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_cec.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_crc.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_dac.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_dbgmcu.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_dma.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_exti.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_flash.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_fsmc.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_gpio.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_i2c.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_iwdg.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_pwr.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_rcc.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_rtc.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_sdio.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_spi.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_tim.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_usart.o
CC build\Libraries\STM32F10x_StdPeriph_Driver\src\stm32f10x_wwdg.o
CC build\packages\button-v1.0.0\button.o
LINK rtthread-stm32.elf
fromelf --bin rtthread-stm32.elf --output rtthread.bin
fromelf -z rtthread-stm32.elf
scons: done building targets.

jiejie@JIEJIE F:\jiejieTop\rt-thread\bsp\stm32f10x
> scons --target=mdk5
cmd.exe*[64]:15896
```



## Build Output

```

Program Size: Code=121920 RO-data=11128 RW-data=1528 ZI-data=6376
After Build - User command #1: fromelf --bin .\build\rtthread-stm32.axf --output rtthread.bin
".\build\rtthread-stm32.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:02

```

<https://jiejie.blog.csdn.net>

## 关于rtkpgs

### 简介 (English)

buildpkg 是用于生成 RT-Thread package 的快速构建工具。

一个优秀的 package 应该是这样的：

1. 代码优雅, 规范化。
2. examples 例程, 提供通俗易懂的使用例程。
3. SConscript 文件, 用于和 RT-Thread 环境一起进行编译。
4. README.md 文档, 向用户提供必要的功能说明。
5. docs 文件夹, 放置除了 README 之外的其他细节文档。
6. license 许可文件, 版权说明。

为了方便快速的生成 RT-Thread package 规范化模板 以及 减轻开源仓库迁移 RT-Thread 的前期准备工作的负担, 基于此目的的 buildpkg 应运而生, 为开发 Rt-Thread 的 package 的开发者提供辅助开发工具。

序号	支持功能	描述
1	构建 package 模板	创建指定名称 package, 自动添加 readme /版本号/ github ci脚本/demo/开源协议文件
2	迁移开源仓库	从指定 git 仓库构建 package, 自动添加readme/版本号/ github ci脚本/demo/开源协议文件, 但是迁移的仓库需要用户自己按照实际情况修改
3	更新 package	生成package后可以再次更新之前设定的版本号, 开源协议或者scons脚本等

## 使用说明

### 1. 构建package

```
buildpkg.exe make pkgdemo
```

### 2. 迁移开源仓库

```
buildpkg.exe make cstring https://github.com/liu2guang/cstring.git
```

### 3. 更新package

```
buildpkg.exe update pkgname
```

### 4. 可选配置

长参数	短参数	描述
--version=v1.0.0	-v v1.0.0	设置 package 的版本
--license=MIT	-l MIT	设置 package 所遵循的版权协议
--submodule	-s	删除 git 子模块

测试平台

序号	测试平台	测试结果
1	win10	exe测试通过, py测试通过
2	win7	exe待测试, py待测试
3	mac	py脚本不知道是否兼容, 没有测试条件, 后面维护下
4	linux	py脚本不知道是否兼容, 没有测试条件, 后面维护下

联系人

- 邮箱: [1004383796@qq.com](mailto:1004383796@qq.com)
- 主页: [liu2guang](#)
- 仓库: [Github](#), [Gitee](#)

喜欢就关注我吧！

设为星标 ★



物联网IoT

 全栈开发

关注物联网那些事，专注分享物联网知识

相关代码可以在公众号后台获取。