

# CW32\_BLDC\_EVA 型

## 评估板

开发指南

2022.2

## 目录

第一章 Cortex-M0+内核及 CW32 微处理器概述 .....	4
1.1 Cortex-M0+概述 .....	4
1.2 CW32 产品特性及芯片选型 .....	4
1.2.1 产品特性 .....	4
1.2.2 芯片选型 .....	7
1.3 CW32 资料获取 .....	7
第二章 CW32 开发环境使用 .....	8
2.1 开发环境概述 .....	8
2.1.1 MDK .....	8
2.1.2 IAR .....	9
2.2 开发工具选择 .....	10
2.2.1 DAP .....	10
2.2.2 ST-LINK .....	10
2.3 开发环境的安装与配置 .....	10
2.3.1 MDK 的安装与配置 .....	10
2.4 CW32 固件库简介 .....	14
2.5 快速入门实例 .....	15
第三章 CW32_BLDC_EVA 评估板硬件设计 .....	24
3.1 CW32_BLDC_EVA 评估板 .....	24
3.1.1 评估板简介 .....	24
3.1.2 评估板硬件资源 .....	25
3.2 评估板原理图 .....	26
3.2.1 MCU .....	26
3.2.2 SWD 下载电路 .....	27
3.2.3 复位电路 .....	28
3.2.4 按键&LED 电路 .....	29
3.2.5 OLED 电路 .....	30
3.2.6 蓝牙接口 .....	31
3.2.7 NTC 温度传感器 .....	31
3.2.8 6 路 MOS 管与 EG3013 驱动电路 .....	32
第四章 CW32_BLAC_EVA 评估板基础实例 .....	33
4. 1 实例一 流水灯实验 .....	33
4.1.1 功能要求 .....	33
4.1.2 硬件原理 .....	33
4.1.3 软件编程 .....	33
4.1.4 软件运行 .....	37
4. 2 实例二 按键指示灯实验 .....	38
4.2.1 功能要求 .....	38
4.2.2 硬件设计 .....	38
4.2.3 软件设计 .....	39
4.2.4 下载与验证 .....	40
4. 3 实例三 蜂鸣器实验 .....	41
4.3.1 功能要求 .....	41

4.3.2 硬件设计 .....	41
4.3.3 软件设计 .....	41
4.3.4 下载与验证 .....	42
4.4 实例四 UART 串口实验 .....	43
4.4.1 功能要求 .....	43
4.4.2 硬件设计 .....	43
4.4.3 软件设计 .....	43
4.4.4 下载与验证 .....	46
4.5 实例五 定时器应用实验 .....	47
4.5.1 功能要求 .....	47
4.5.2 硬件设计 .....	47
4.5.3 软件设计 .....	47
4.4.4 下载与验证 .....	50
4.6 实例六 OLED 显示实验 .....	51
4.6.1 功能要求 .....	51
4.6.2 硬件设计 .....	51
4.6.3 软件设计 .....	52
4.6.4 下载与验证 .....	54
4.7 实例七 ADC 电位器实验 .....	55
4.7.1 功能要求 .....	55
4.7.2 硬件设计 .....	55
4.7.3 软件设计 .....	56
4.7.4 下载与验证 .....	58
4.8 实例八 霍尔开环电机实例 .....	59
4.8.1 功能要求 .....	59
4.8.2 硬件设计 .....	59
4.8.3 软件设计 .....	60
4.9 实例九 电机运转（霍尔&PID） .....	65
4.9.1 功能要求 .....	65
4.9.2 硬件设计 .....	65
4.9.3 软件设计 .....	66
4.10 实例十 电机运转（无霍尔开环） .....	70
4.10.1 功能要求 .....	70
4.10.2 硬件设计 .....	70
4.10.3 软件设计 .....	71
4.11 实例十一 电机运转（无霍尔闭环） .....	74
4.11.1 功能要求 .....	74
4.11.2 硬件设计 .....	74
4.11.3 软件设计 .....	75

# 第一章 Cortex-M0+内核及 CW32 微处理器概述

## 1.1 Cortex-M0+概述

ARM® Cortex®-M0+ 内核是 ARM® 为小型嵌入式系统开发的最新一代 32 位内核平台，用以实现方便使用的低成本解决方案。该平台在仅需有限的引脚数和功率消耗的同时，给用户提供出色的计算性能和快速的中断响应。ARM® Cortex®-M0+ 32 位精简指令集处理器提供出色的代码效率，在小储存空间的条件下给用户提供对 ARM 内核所期望的高性能。CW32F030 家族产品均采用嵌入式 ARM 内核并保持与所有 ARM 工具和软件的全面兼容。

## 1.2 CW32 产品特性及芯片选型

### 1.2.1 产品特性

- 内核：ARM® Cortex®-M0+
- 最高主频 64MHz
- 工作温度：-40°C 至 105°C；工作电压：1.65V 至 5.5V
- 存储容量
- 最大 64K 字节 FLASH，数据保持 25 年 @85°C

- 最大 8K 字节 RAM，支持奇偶校验
- 128 字节 OTP 存储器
- CRC 硬件计算单元
- 复位和电源管理
  - 低功耗模式 (Sleep, DeepSleep)
  - 上电和掉电复位 (POR/BOR)
  - 可编程低电压检测器 (LVD)
- 时钟管理
  - 4 ~ 32MHz 晶体振荡器
  - 32kHz 低速晶体振荡器
  - 内置 48MHz RC 振荡器
  - 内置 32kHz RC 振荡器
  - 内置 10kHz RC 振荡器
  - 内置 150kHz RC 振荡器
  - 时钟监测系统
  - 允许独立关断各外设时钟
- 支持最多 39 路 I/O 接口
  - 所有 I/O 口支持中断功能
  - 所有 I/O 支持中断输入滤波功能
- 五通道 DMA 控制器
- 模数转换器
  - 12 位精度, ±1 LSB

- 最高 1M SPS 转换速度
- 内置电压参考
- 模拟看门狗功能
- 内置温度传感器
- 双路电压比较器
- 实时时钟和日历
- 支持由 Sleep/DeepSleep 模式唤醒
- 定时器
  - 16 位高级控制定时器，支持 6 路捕获 / 比较通道和 3 对互补 PWM 输出，死区时间和灵活的同步功能
  - 四组 16 位通用定时器
  - 三组 16 位基本定时器
  - 窗口看门狗定时器
  - 独立看门狗定时器
- 通信接口
  - 三路低功耗 UART，支持小数波特率
  - 两路 SPI 接口 12Mbit/s
  - 两路 I2C 接口 1Mbit/s
  - IR 调制器
- 串行调试接口 (SWD)
- 80 位唯一 ID
- 所有封装兼容 ECOPACK 2

## 1.2.2 芯片选型

系列	型号	封装
CW32F030x8	CW32F030C8	LQFP48
	CW32F030K8	LQFP32
	CW32F030F8	QFN32
	CW32F030F6	QFN20
CW32F030x6	CW32F030F6	TSSOP20

图 1.2.2

## 1.3 CW32 资料获取

CW32 各系列芯片资料可登录官网下载

官网地址：<https://www.whxy.com/support/filelist/16>

首页 > 产品中心 > MCU > ARM Cortex-M0+

ARM Cortex-M0+

产品 资料

产品型号	描述	数据手册	封装	内核	主频	FLASH	RAM	定时器	12位ADC	GPIO	I2C	SPI	UART	工作电压	工作温度	购买
CW32F030C8	通用MCU产品	下载	LQFP48	ARM Cortex-M0+	64MHz	64KB	8KB	8	1 (13外 3内)	39	2	2	3	1.65V-5.5V	-40°C-105°C	
CW32F030K8	通用MCU产品	下载	LQFP32, QFN32	ARM Cortex-M0+	64MHz	64KB	8KB	8	1 (10外 3内)	25	2	2	3	1.65V-5.5V	-40°C-105°C	
CW32F030F8	通用MCU产品	下载	TSSOP20	ARM Cortex-M0+	64MHz	32KB	6KB	8	1 (9外 3内)	15	2	2	3	1.65V-5.5V	-40°C-105°C	
CW32F030F8	通用MCU产品	下载	QFN20	ARM Cortex-M0+	64MHz	64KB	8KB	8	1 (9外 3内)	15	2	2	3	1.65V-5.5V	-40°C-105°C	
CW32F003E4	通用MCU产品	下载	TSSOP24	ARM Cortex-M0+	48MHz	20KB	3KB	5	1 (13外 3内)	21	1	1	2	1.65V-5.5V	-40°C-105°C	
CW32F003F4	通用MCU产品	下载	TSSOP20, QFN20	ARM Cortex-M0+	48MHz	20KB	3KB	5	1 (13外 3内)	17	1	1	2	1.65V-5.5V	-40°C-105°C	

6 | 1/1 首页 上一页 1 下一页 末页

图 1.3



数据手册

文件名称	文件ID(大小)	版本	日期
CW32F003数据手册	PSMCU0013.pdf (1261KB)	Rev1.3	2022/02/23
CW32F030数据手册	PSMCU0003.pdf (1482KB)	Rev1.4	2022/01/10

2 | 1/1 首页 上一页 1 下一页 末页

图 1.4

## 第二章 CW32 开发环境使用

### 2.1 开发环境概述

#### 2.1.1 MDK

RealView MDK 开发工具源自德国 Keil 公司，被全球超过 10 万的嵌入式开发工程师验证和使用，是 ARM 公司目前最新推出的针对各种嵌入式处理器的软件开发工具。RealViewMDK 集成了业内最领先的技术，融合了中国多数软件开发工程师所需的特点和功能。包括 μVision3 集成开发环境与 RealView 编译器，支持 ARM7、ARM9 和

最新的 Cortex-M3 核处理器以及 Cortex-M0 核处理器，自动配置启动代码，集成 Flash 烧写模块，强大的 Simulation 设备模拟，性能分析等功能，与 ARM 之前的工具包 ADS 等相比，RealView 编译器的最新版本可将性能改善超过 20%。如图 2.1.1 所示：

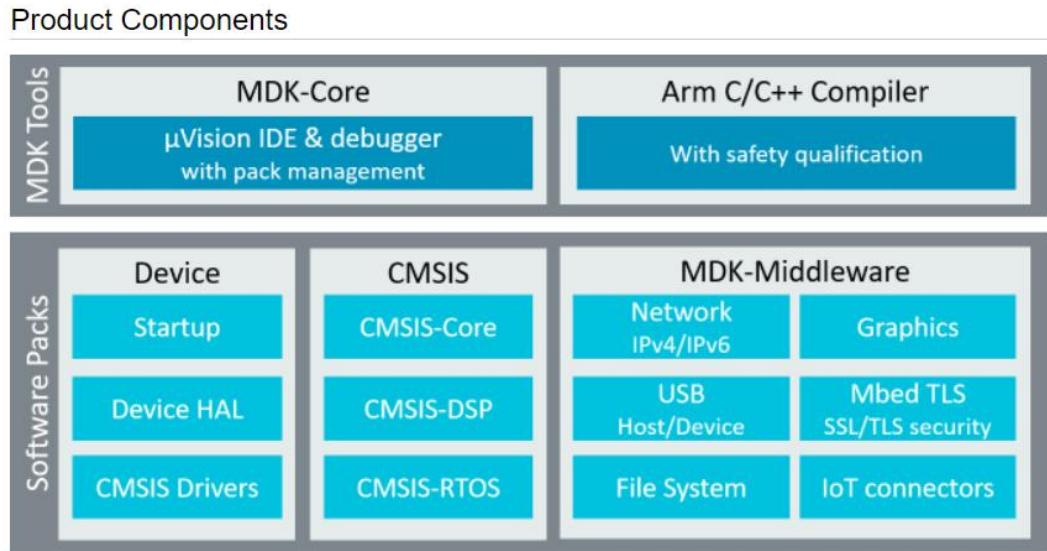


图 2.1.1 MDK5 组成

MDK5 安装包可以在：<http://www.keil.com/demo/eval/arm.htm> 下载到。而器件支持、设备驱动、CMSIS 等组件，则可以点击 MDK5 的 Build Toolbar 的最后一个图标调出 Pack Installer，来进行各种组件的安装。也可以在 <http://www.keil.com/dd2/pack> 这个地址下载，然后进行安装。要让 MDK5 支持 CW32 的开发，我们还需要将复制固件包“IdeSupport\MDK\flashloader”目录下的 \*.FLM 文件复制到 MDK-ARM 的安装目录“Keil\_v5\ARM\Flash\”文件中”固件包我们都已经在开发板光盘提供了，大家自行安装即可。

## 2.1.2 IAR

IAR Systems 推出的开发工具，也是一种集成开发环境的名称，我们平时所说的 IAR 主要是指集成开发环境。IAR 这家公司的发展也是经历了一系列历史变化，从开始针对 8051 做 C 编译器，逐渐发展至今，已经是一家庞大的、技术力量雄厚的公司。而 IAR 集成开发环境也是从单一到现在针对不同处理器，拥有多种 IAR 版本的集成开发环境。

请参看官网：

<https://www.iar.com/device-search/#!tab=devices>

## 2.2 开发工具选择

### 2.2.1 DAP

WCH-LINK:带有 SWD 接口的 ARM 核单片机，并且自带虚拟串口功能。

### 2.2.2 ST-LINK

ST-LINK 是用于 STM8 和 STM 和 STM32 微控制器的在线调试器和编程器，也是大家口中的下载器，ST-Link 具有 SWIM,JTAG/SWD 等通信接口，用于与 STM8 或 STM32 微控制器进行通信（各版本有差异）。

## 2.3 开发环境的安装与配置

### 2.3.1 MDK 的安装与配置

在光盘中找到“MDK 开发环境安装软件”，安装 MDK533.EXE 软件，按以下步骤进行软件安装。（说明，以下 MDK 的安装有可能

有升级版，无论光盘中配有什么版本，安装和配置方法均跟下面描述相同。）

（1）双击文件，开始安装。（以下安装为 MDK533.EXE 为例，更高版本安装方法相同）

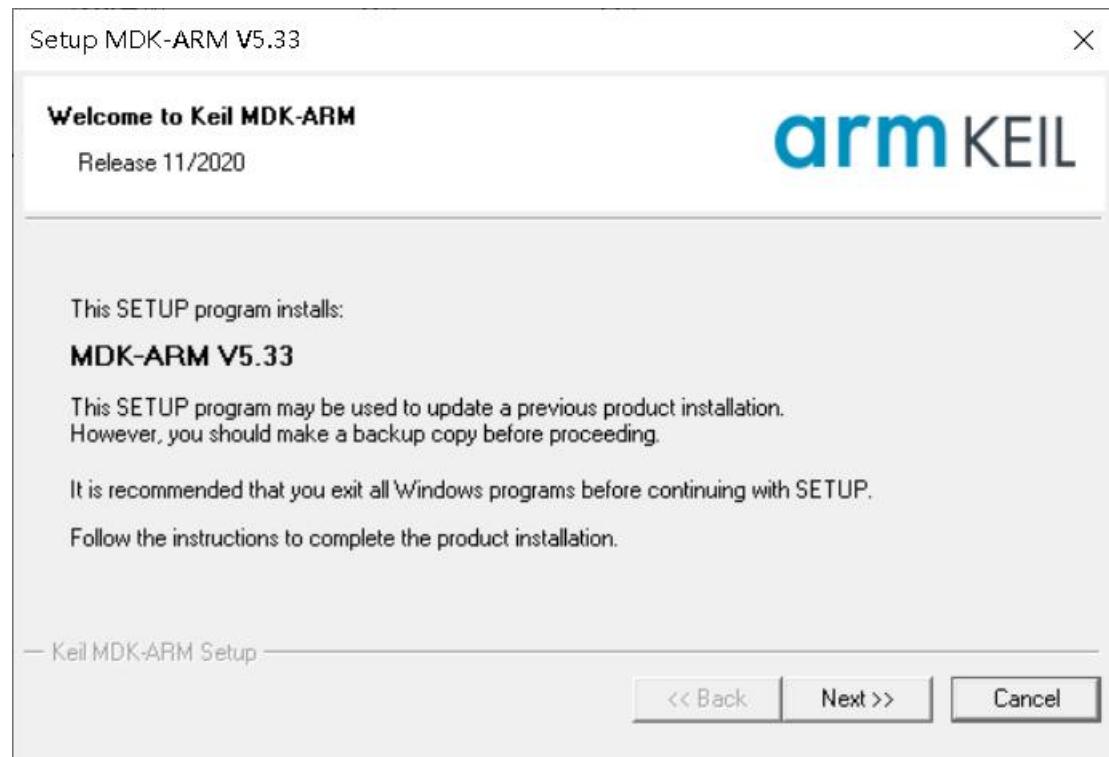


图 2.3.1 安装界面

（2）选中同意安装选项框。

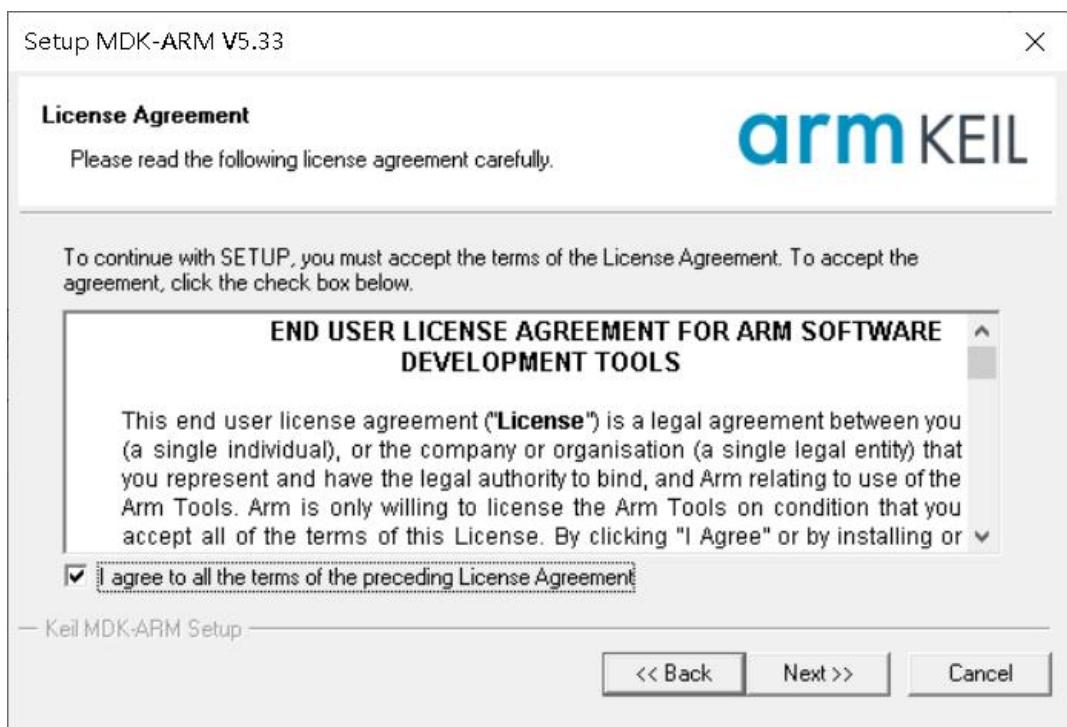


图 2.3.2 软件安装协议

(3) 根据需要修改安装路径。

(4) 填入对应信息。

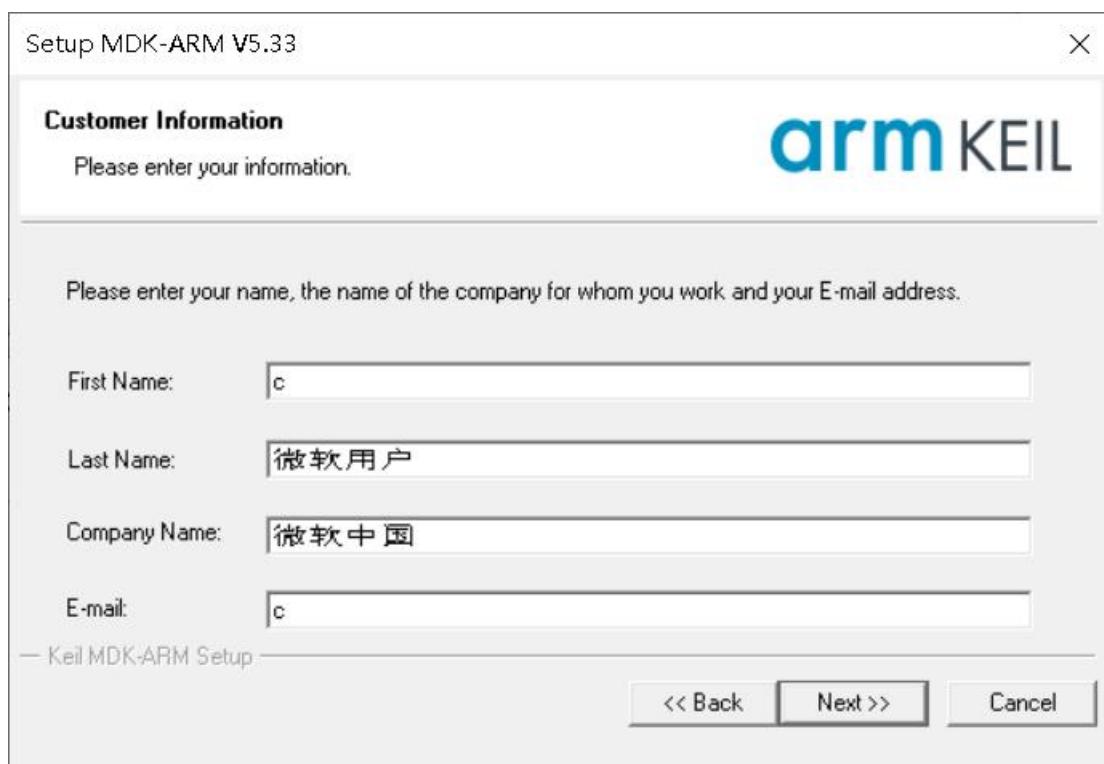


图 2.3.3 填写用户信息

(5)点击“NEXT”开始安装，直到安装完成。

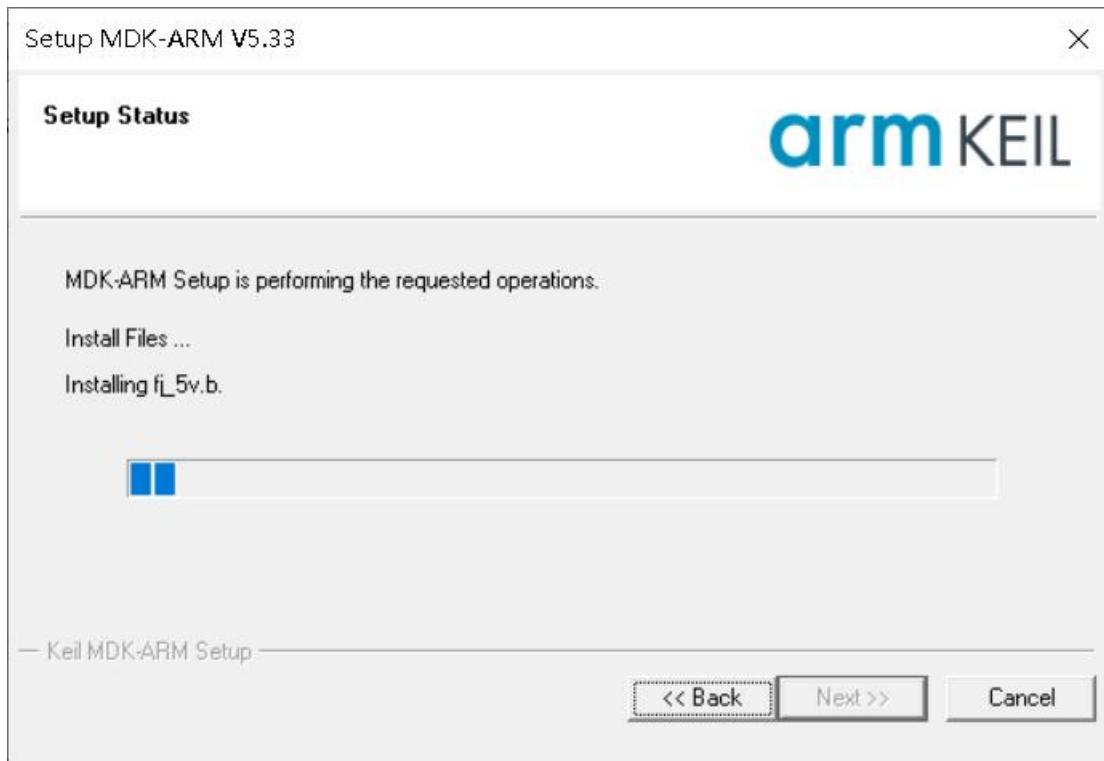


图 2.3.4 安装过程

(6) MDK 安装完成。

注意，MDK 注册完成后，为使 WCH-LINK 能正确仿真 CW32，需对 WCH-LINK 安装驱动。找到驱动软件包双击进行安装首先点击“DapLinkSer”文件夹安装驱动，在 KEIL5 环境下 WCH-LINK 要在“ARM”模式下，将 WCH-Link 切换到 ARM 模式，在 Keil 中选择 CMSIS-DAP Debugger 即可仿真调试..。

(7) 搭建 CW32 运行环境安装固件包 PS.CW32F030\_DFP.1.0.1.pack 安装到 MDK-ARM 的安装目录“Keil\_v5\”文件中”。

本地磁盘 (E:) > cw32f030-stdperiph-lib > IdeSupport > MDK			
名称	修改日期	类型	大小
PS.CW32F030_DFP.1.0.1.pack	2022/5/6 16:42	uVision Software...	127 KB
startup_cw32f030.s	2022/5/6 16:42	S 文件	10 KB

图 2.3.5

## 2.4 CW32 固件库简介

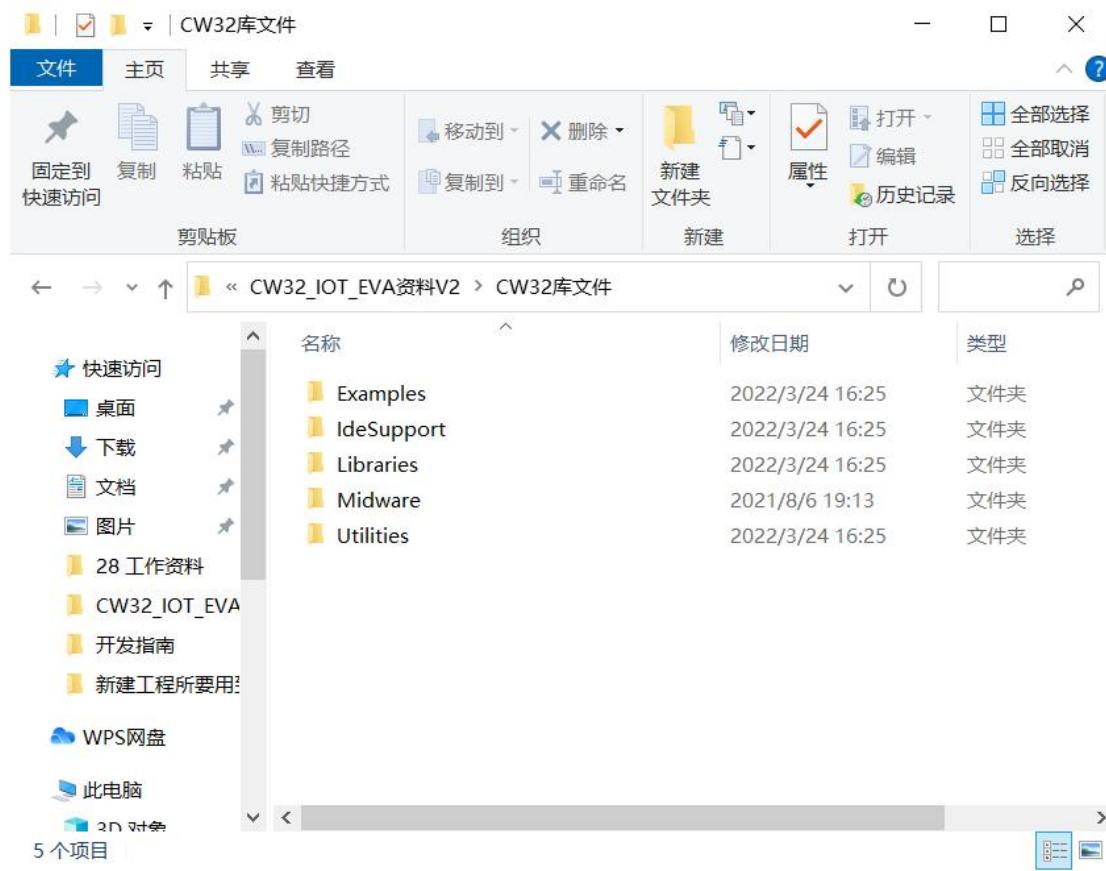


图 2.4

Examples 文件夹下面存放的芯源半导体官方提供的固件实例源码，在以后的开发过程中，可以参考修改这个官方提供的实例来快速驱动自己的外设。

IdeSupport 文件夹里面存放的是 startup\_cw32f030.s 启动文件以及用于 MDK 工具的单片机 Flash 烧写配置文件(FlashCW32F030.FLM)。

Libraries 文件夹里面的文件在我们建立工程的时候都会使用到。

Utilities 文件夹下就是官方评估板的一些对应源码，这个可以忽略不看。

## 2.5 快速入门实例

- (1)新建“GPIO”文件，在 GPIO 文件夹内建立 user 文件夹。
- (2)打开 CW32\_BLDC\_EVA 资料包里面的“新建工程所用到的库文件”将里面的 core\_cm0plus.h,interrupts\_cw32f030.c,interrupts\_cw32f030.h,Startup\_cw32f030.s 文件复制到 user 文件夹内。
- (3) 将 Libraries 文件复制到新建的 GPIO 工程文件里面。如图 2.5.1：

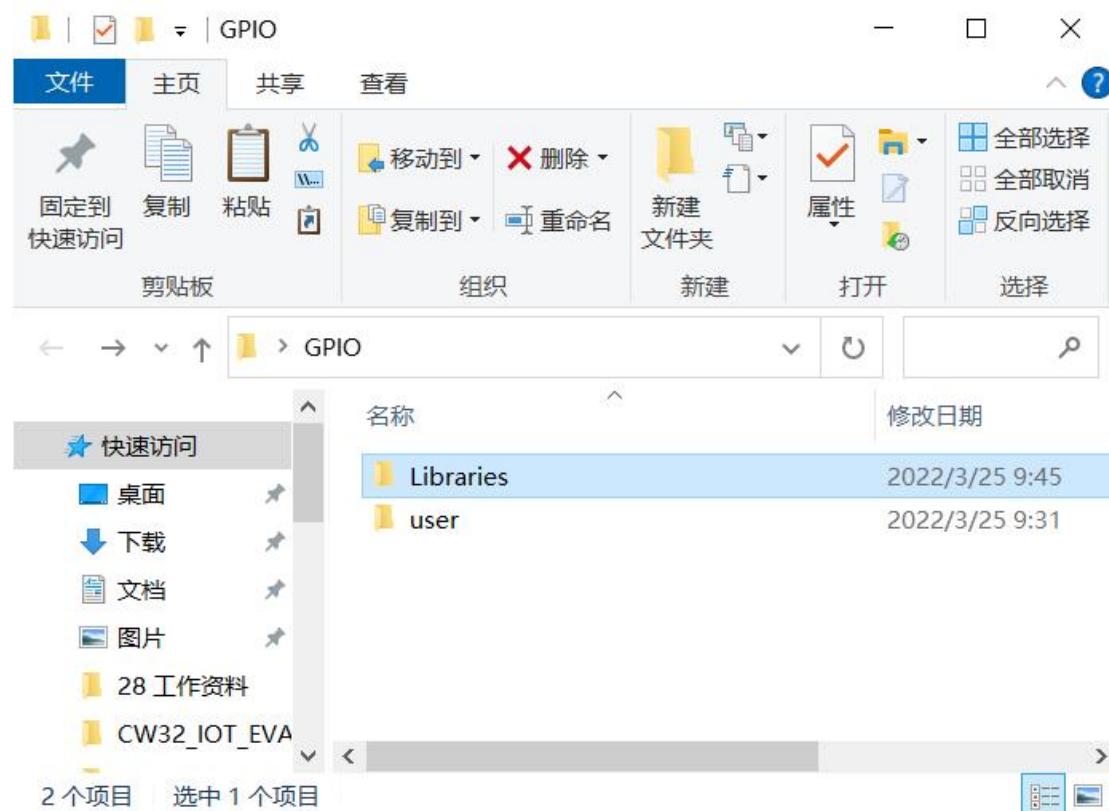
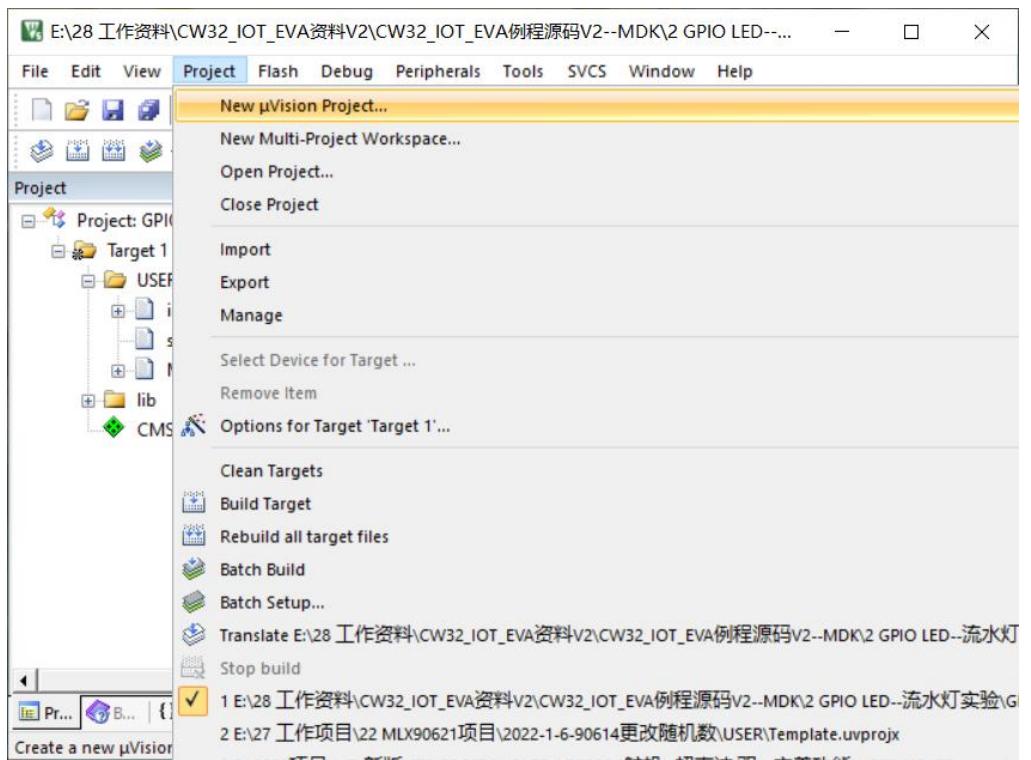


图 2.5.1

- (4) 打开 MDK5 点击 project 新建工程如图 2.5.2:



图

### 2.5.2

(5) 保存在我们刚才所建立的 GPIO 文件夹内，之后会弹出选择芯片型号，选择 ARMCMOP 如图 2.5.3：

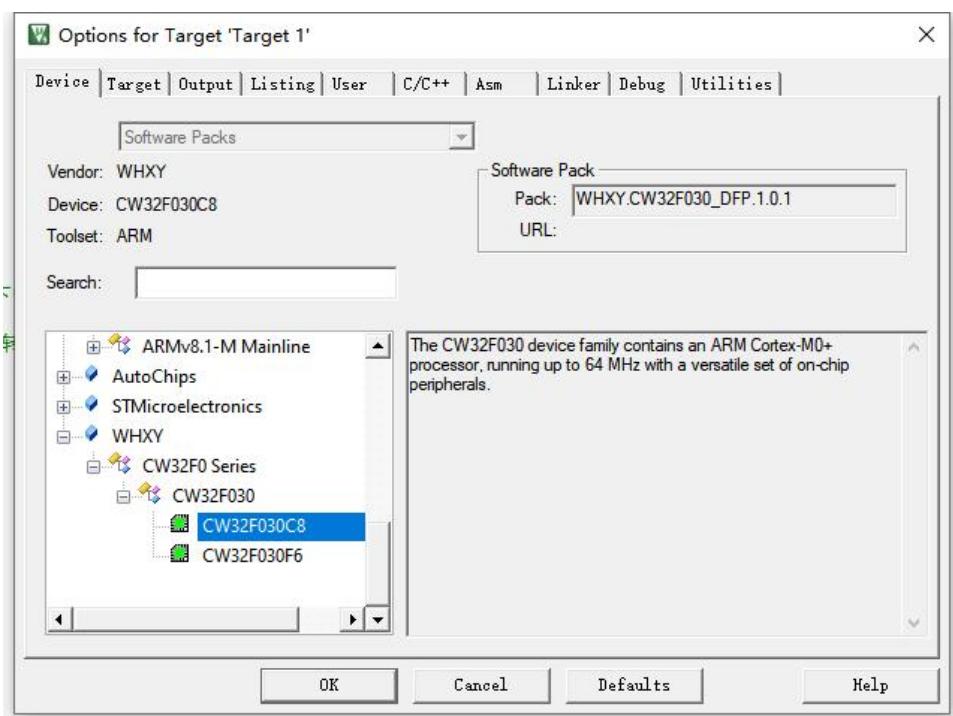


图 2.5.3

(6) 新建 main.c 与 main.h 文件并保存。

(7) 右键工程目录添加.c 文件如图 2.5.4:

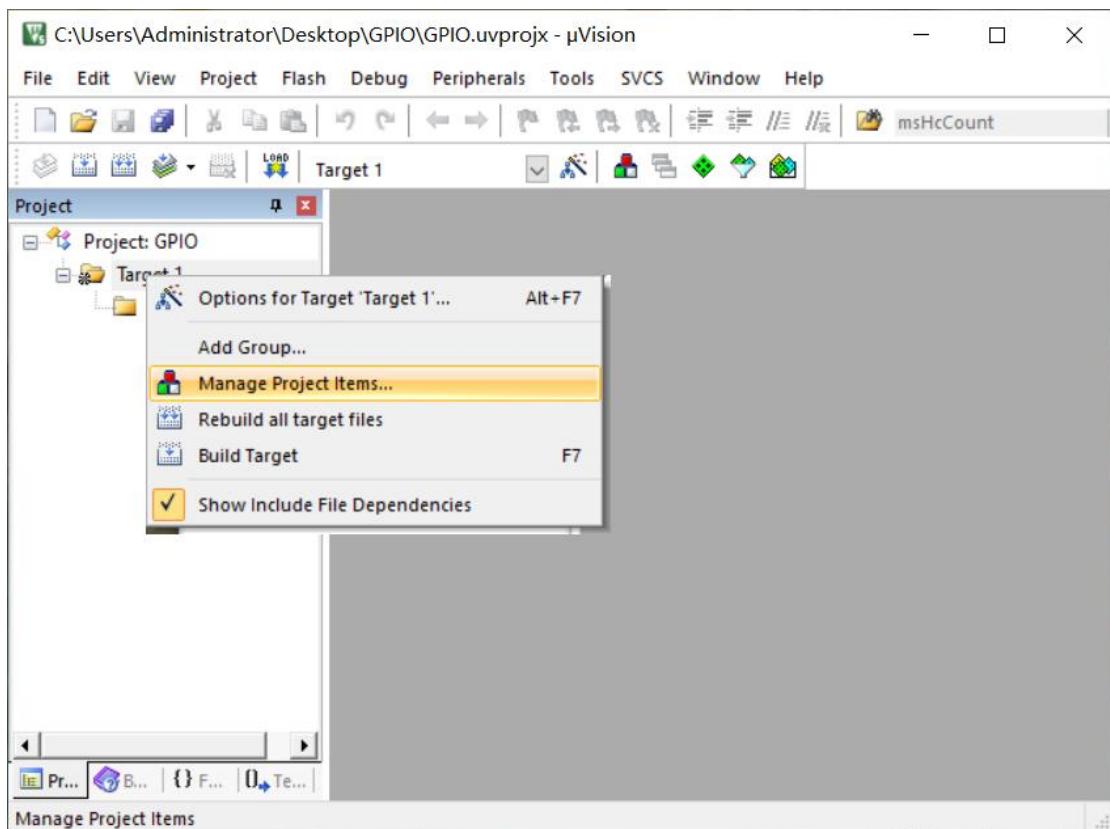


图 2.5.4

(8) 点击魔术棒进入 (C/C++) 点击 Include paths 如图 2.5.5 :

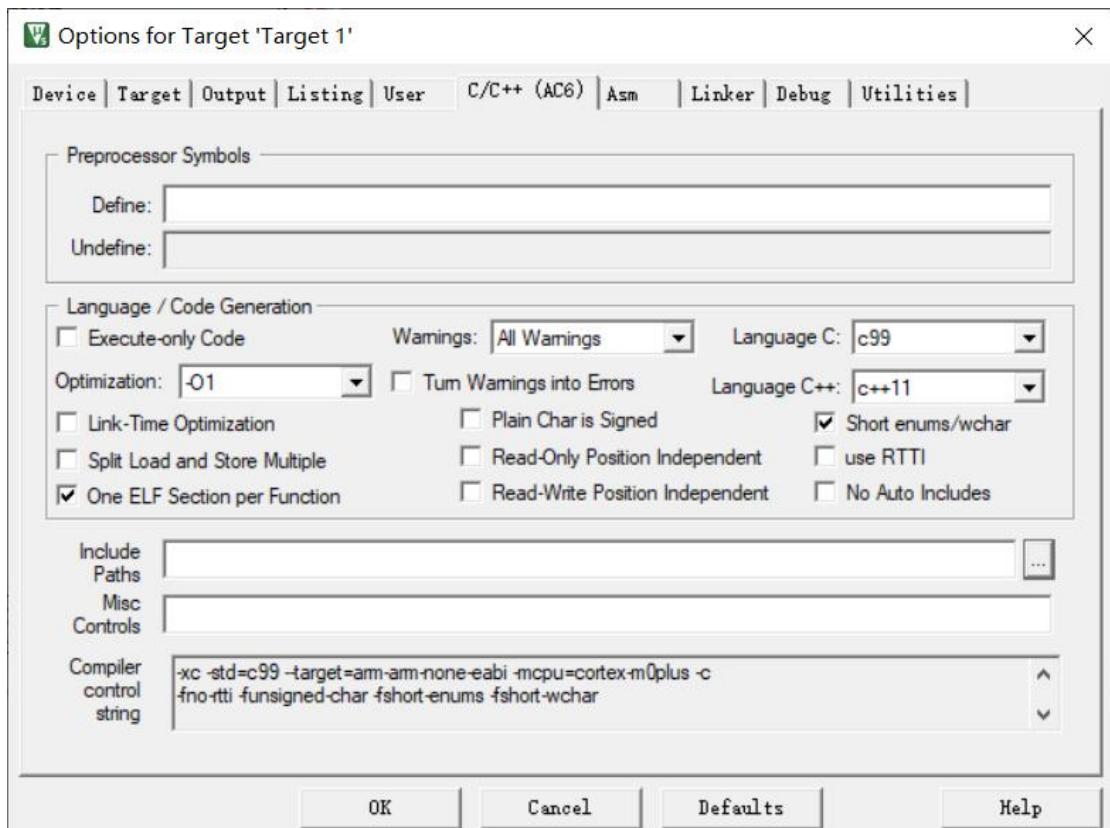


图 2.5.6

(9) 头文件与.c 文件添加完成之后打开魔术棒选择选择 Use default compiler version 5 如图 2.5.7:

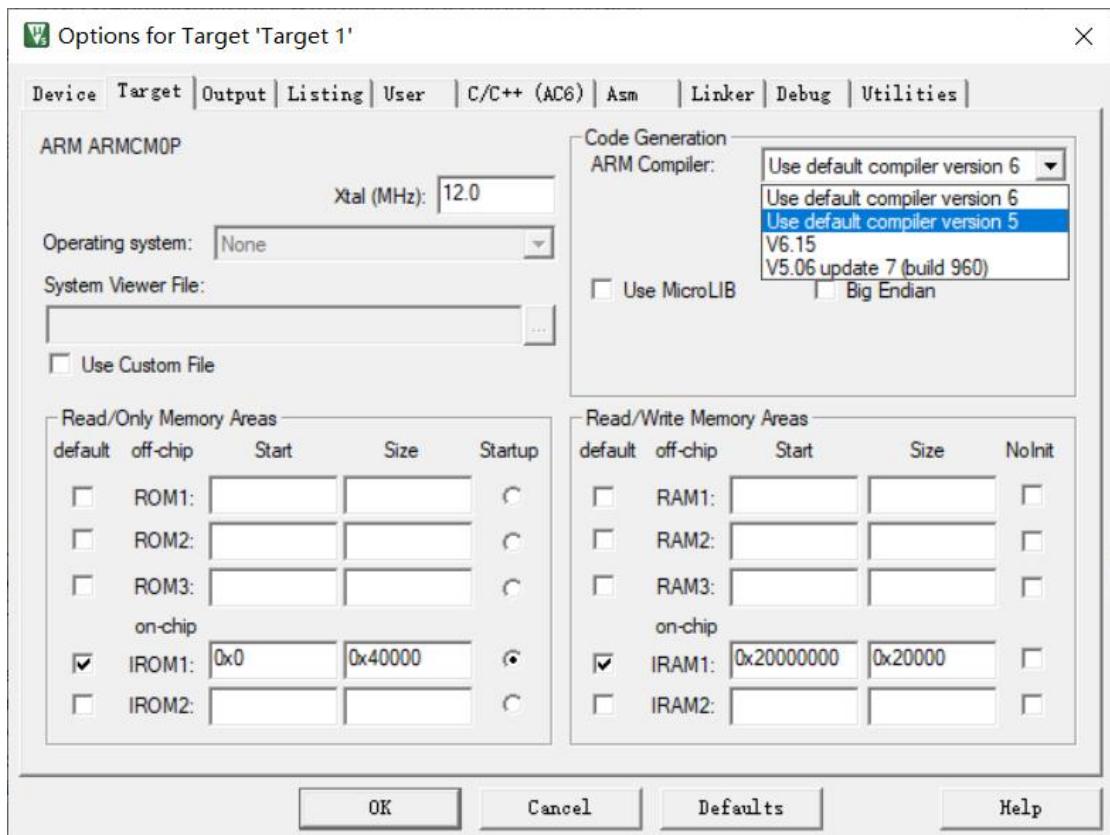
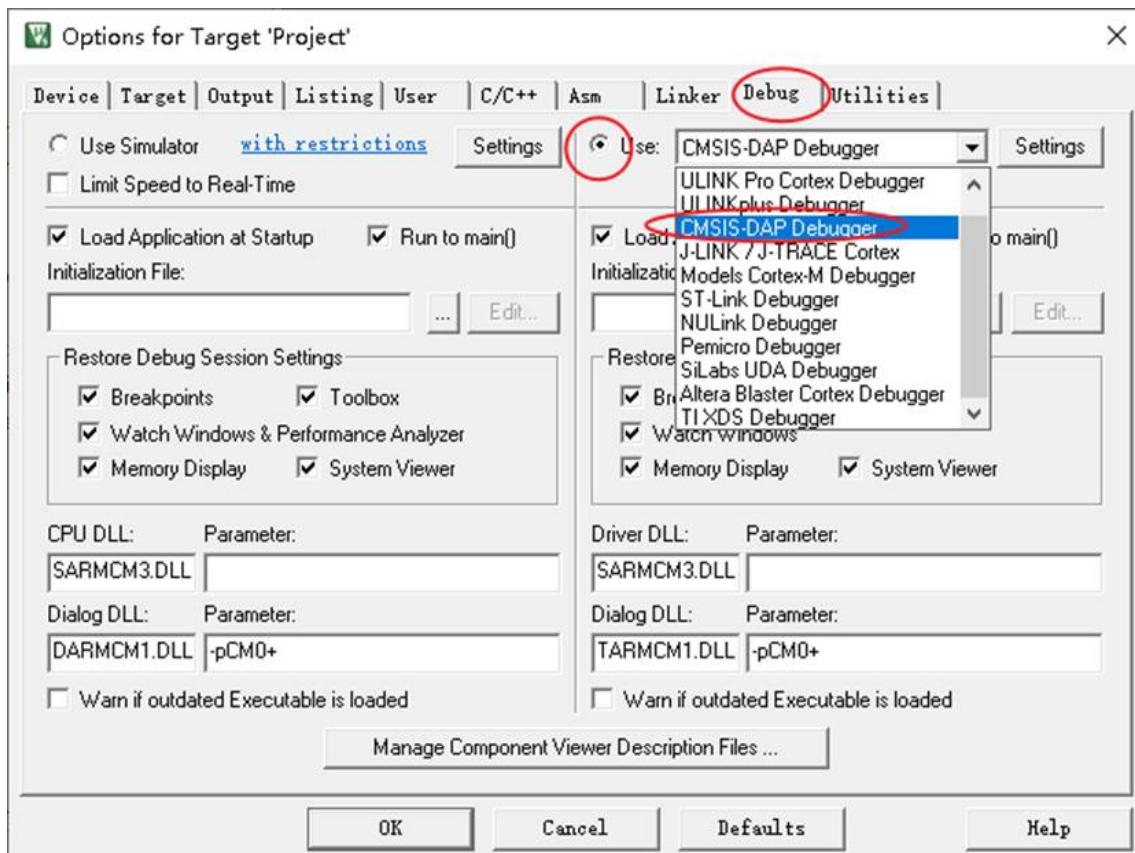


图 2.5.7

(10)选择“Debug”项，修改调试器为“CMSIS-DAP Debugger”，如

图 2.5.8：



如图 2.5.8

(11) 选择“Utilities”项，点击“Settings”按钮，在弹出的对话框中选择“Flash Download”项，如图 2.5.9：

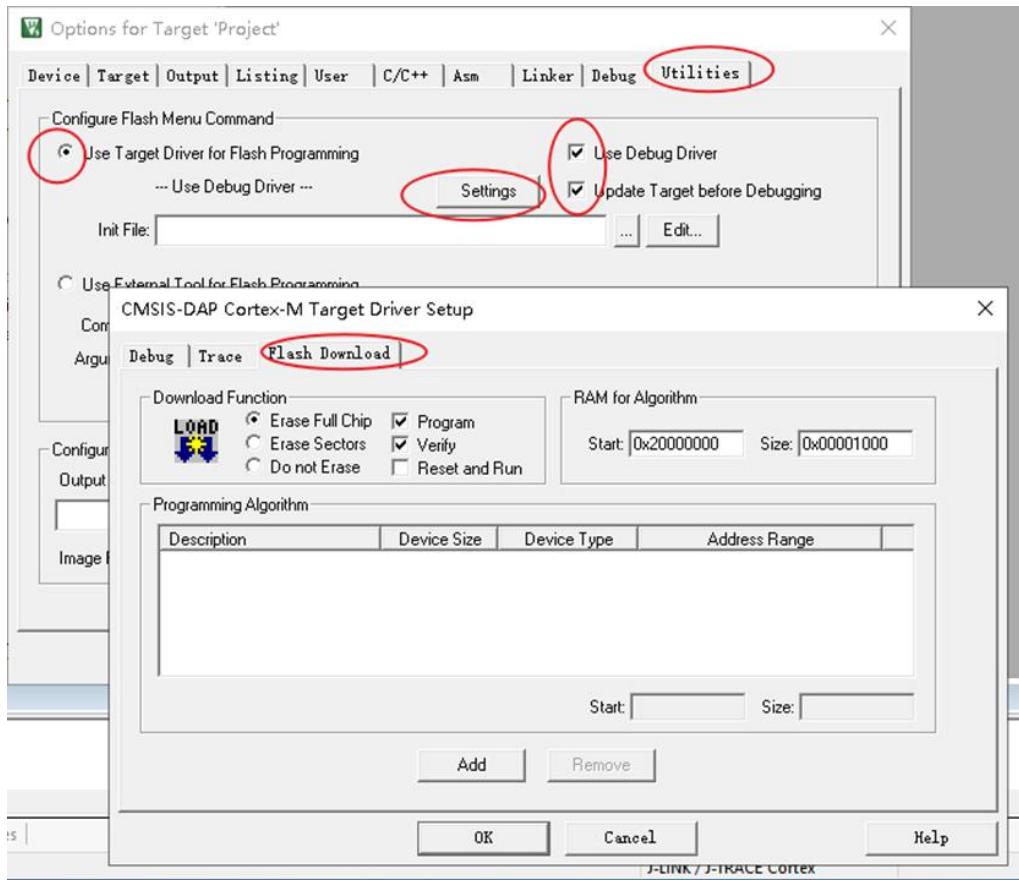


图 2.5.9

(12) 点击“Add”按钮，找到“CW32F030”的 Flash 烧写算法，点击“Add”按钮添加，如图 2.5.10：

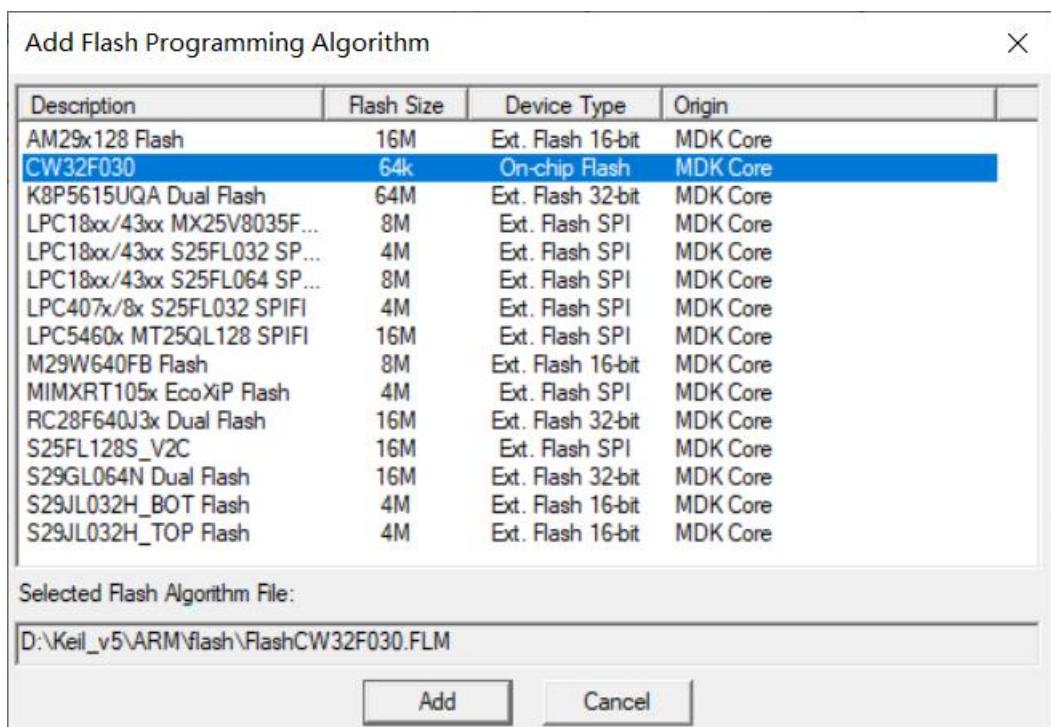


图 2.5.10

(13) 在“Project”菜单中选择“Rebuild all target files”以编译项目。

如果项目编译成功，将显示以下窗口，如图 2.5.11：

```

Build Output

compiling cw32f030_gpio.c...
compiling cw32f030_rcc.c...
compiling cw32f030_flash.c...
linking...
Program Size: Code=412 RO-data=224 RW-data=4 ZI-data=516
FromELF: creating hex file...
".\output\exe\Project.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:03

```

Build Output Find In Files

图 2.5.11

(14) 在 MDK-ARM IDE 中，从调试菜单中选择开始 / 停止调试会话或点击工具栏中的开始 / 停止调试会话按钮，以对 Flash 存储器

进行编程并开始调试，如图 2.5.12：



图 2.5.12

(15) MDK-ARM 中的调试器可用于用 C 语言和汇编语言调试源代码，设置断点，以及监控各个变量和代码执行过程中发生的事件，如图 2.5.13：

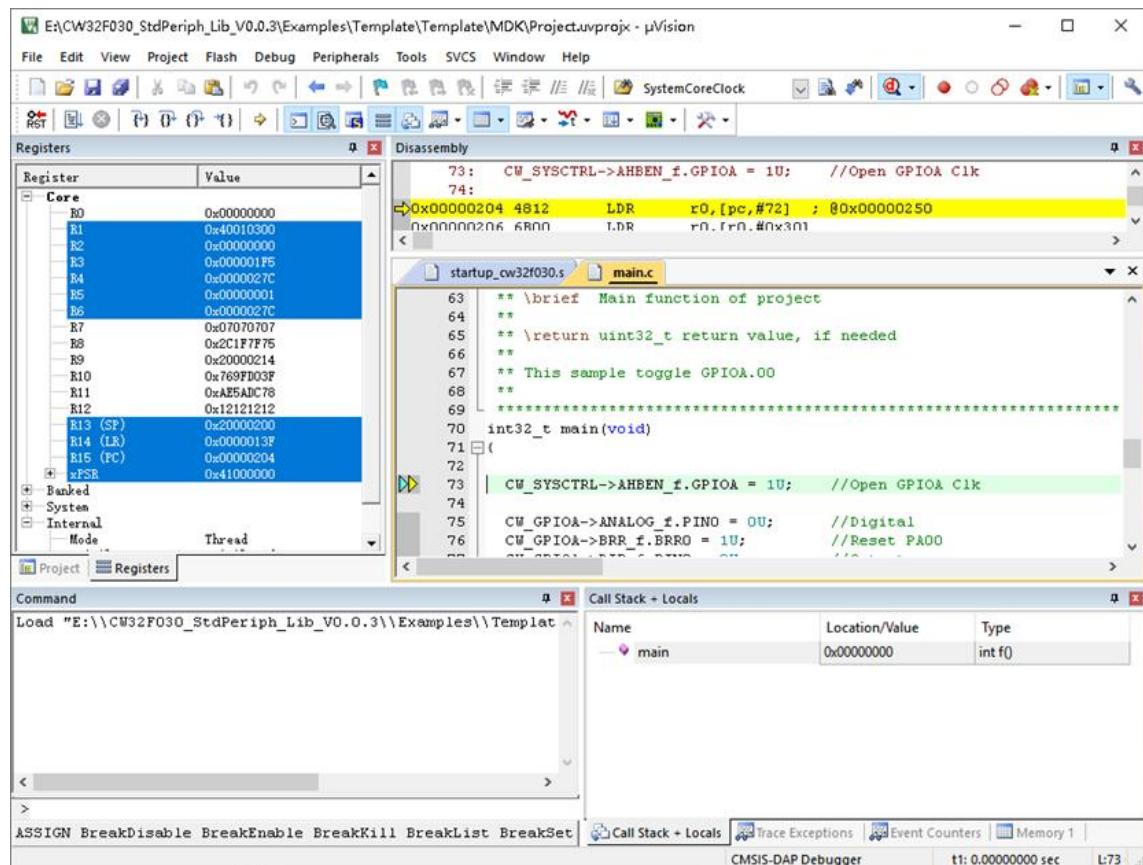


图 2.5.13

### 第三章 CW32\_BLDC\_EVA 评估板硬件设计

### 3.1 CW32\_BLDC\_EVA 评估板

### 3.1.1 评估板简介

基于 CW32\_BLDC\_EVA 评估板采用核心板与电机驱动板共用的方式，能为教师科研项目，电子设计竞赛、单片机竞赛等大学生科技活动，提供一个灵活可靠的硬件及软件调试的平台，它能满足自动化及计算机嵌入式系统等相关专业方向的实验、实习及毕业设计需求。

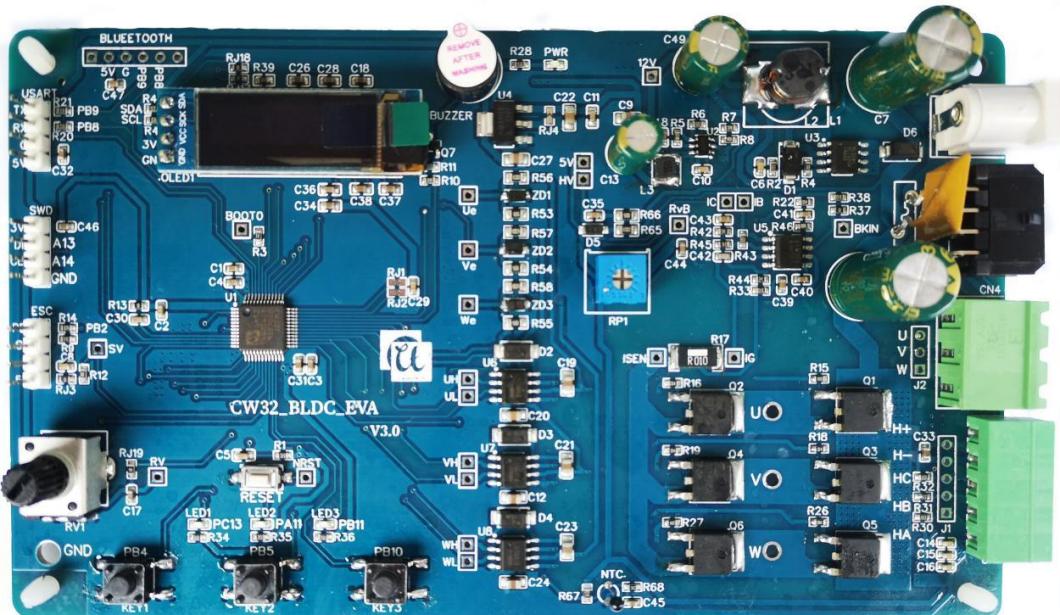


图 3.1.1 评估板实物图

### 3.1.2 评估板硬件资源

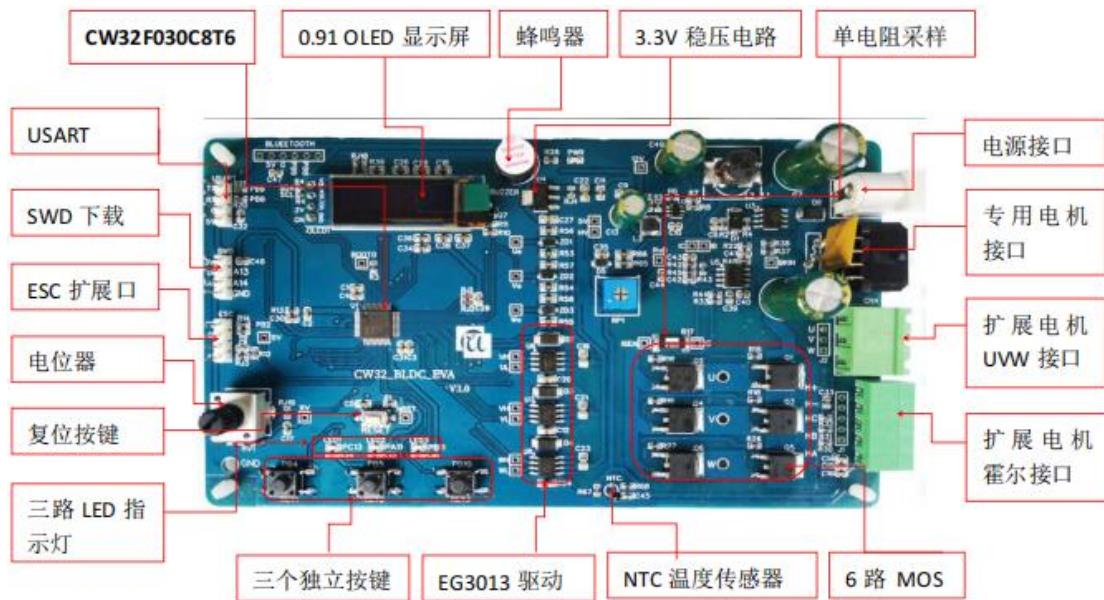


图 3.1.2 评估板资源图

CW32\_BLDC\_EVA 评估板板载资源如下：

- ◆ CPU: CW32F030, FLASH: 64K, SRAM: 8K
- ◆ 1 个 SWD 调试下载口
- ◆ 1 个电源指示灯
- ◆ 1 个 0.91OLED 模块接口 (IIC1 PB6/PB7 已留出接口)
- ◆ 1 个 USART 串口预留接口 (PB8/PB9)
- ◆ 1 个 ESC 串口预留接口 (PB1/PB2)
- ◆ 1 个 AD 电位器 (ADC PB0)
- ◆ 3 个功能按钮
- ◆ 3 个功能指示灯
- ◆ 1 个专用电机接口
- ◆ 1 个扩展电机 UVW 接口
- ◆ 1 个扩展电机霍尔接口
- ◆ 1 个 NTC 温度传感器
- ◆ 1 个复位按钮，可用于复位 MCU

### 3.2 评估板原理图

### 3.2.1 MCU

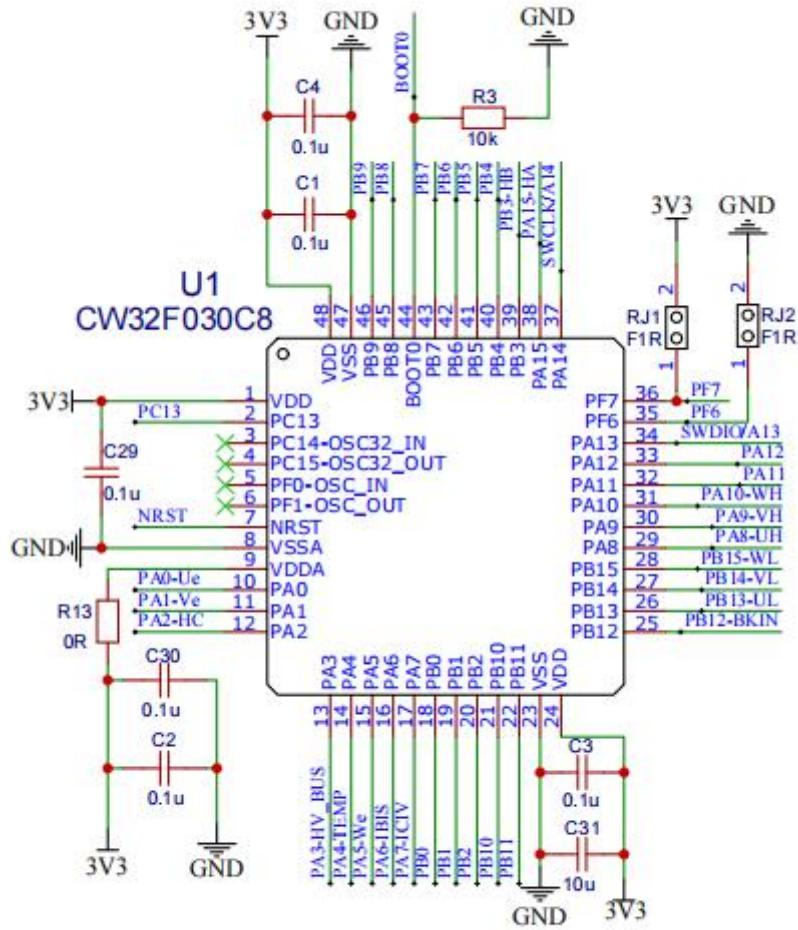


图 3.2.1 原理图

上图为我们的主芯片 CW32F030C8T6。CW32F030C8T6 是基于 eFlash 的单芯片微控制器，集成了主频高达 64MHz 的 ARM® Cortex®-M0+ 内核、高速嵌入式存储器（多至 64K 字节 FLASH 和多至 8K 字节 SRAM）以及一系列全面的增强型外设和 I/O 口。

### 3.2.2 SWD 下载电路

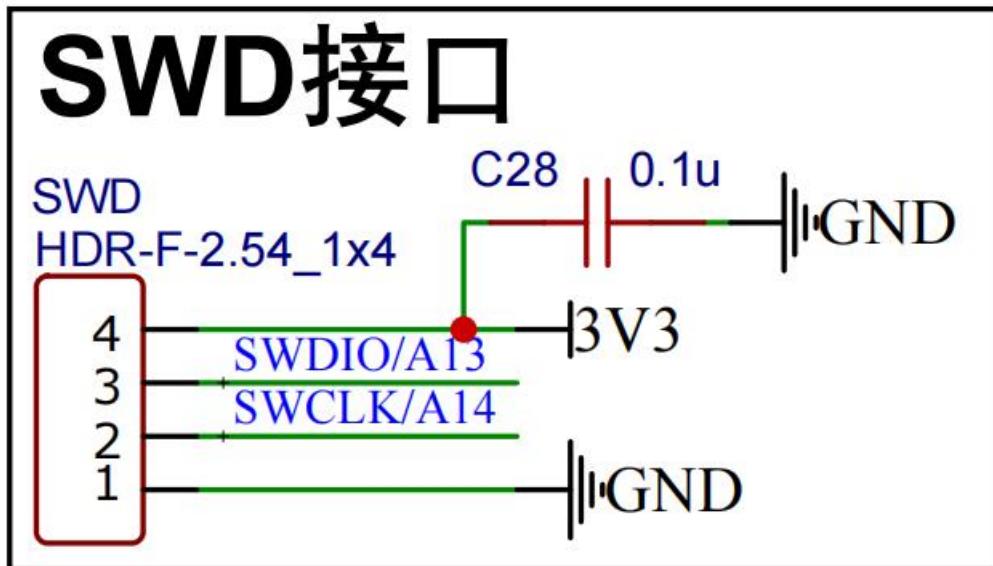


图 3.2.2 SWD

这里，我们采用的是标准的 SWD 接法，SWD 只需要根（SWCLK 和 SWDIO）就可以下载并调试代码了，这同我们使用串口下载代码差不多，而且速度非常快，能调试。你的调试器必须支持 SWD 模式，JLINK V7/V8、ULINK2 和 ST LINK 等都支持 SWD 调试。我们提供的是调试下载器是 WCH-LINK，支持 SWD 模式同时带有虚拟串口功能。

### 3.2.3 复位电路

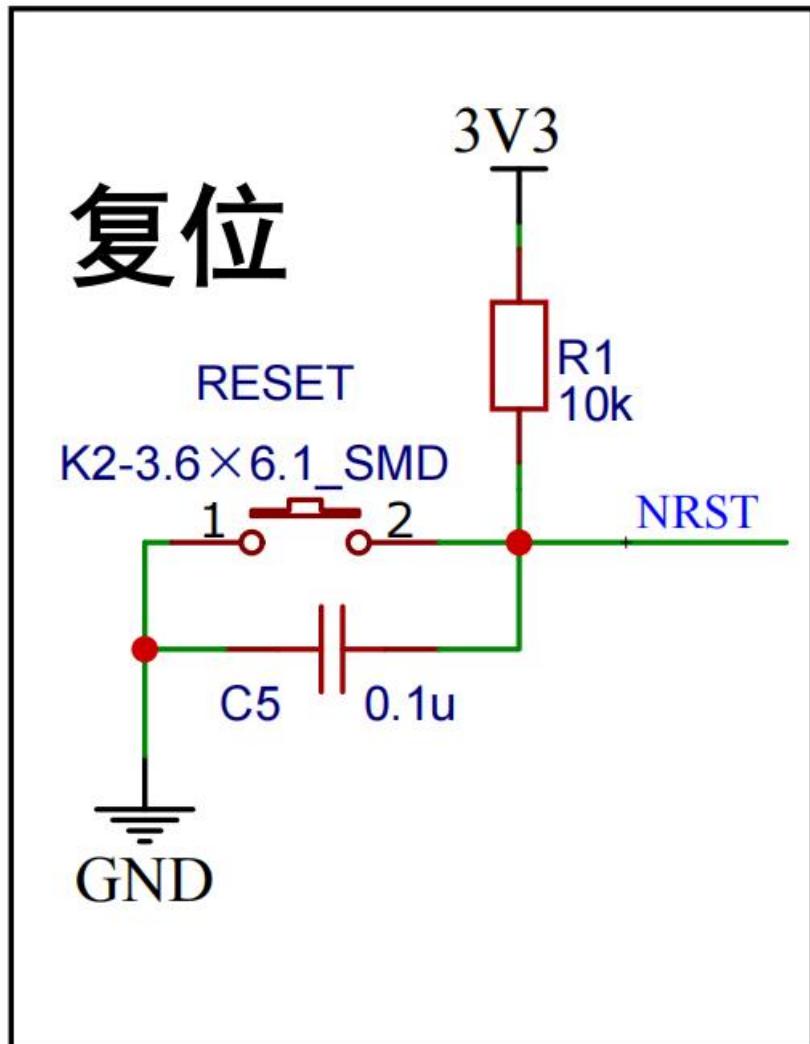


图 3.2.3 复位电路

因为 CW32 是低电平复位的，所以我们设计的电路也是低电平复位的，这里的 R1 和 C5 构成了上电复位电路。

### 3.2.4 按键&LED 电路

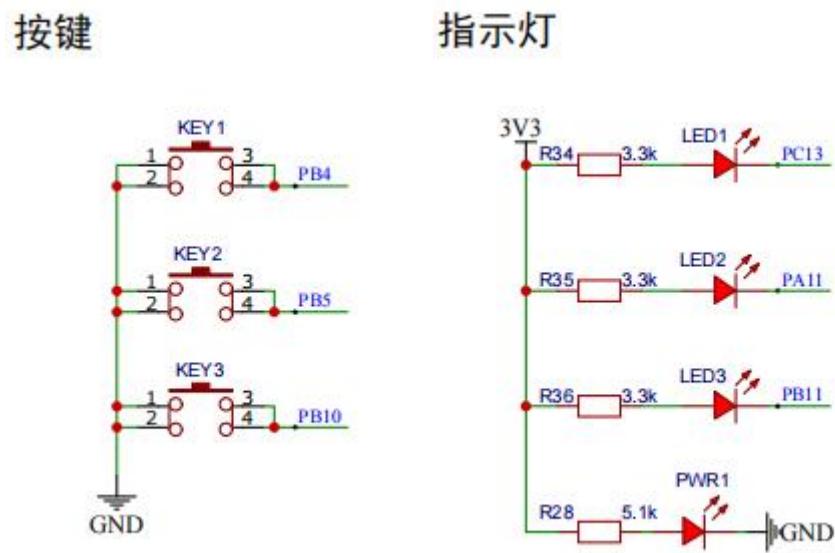


图 3.2.4 按键&LED

CW32 评估板板载总共有 3 个输入按键。KEY1, KEY2, KEY3 用作普通按键输入，分别连接在 PB4,PB8,PB10 上，这里并没有使用外部上拉电阻，但是 CW32 的 IO 作为输入的时候，可以设置上下拉电阻，所以我们使用 CW32 的内部上拉电阻来为按键提供上拉。

### 3.2.5 OLED 电路

OLED屏/接口

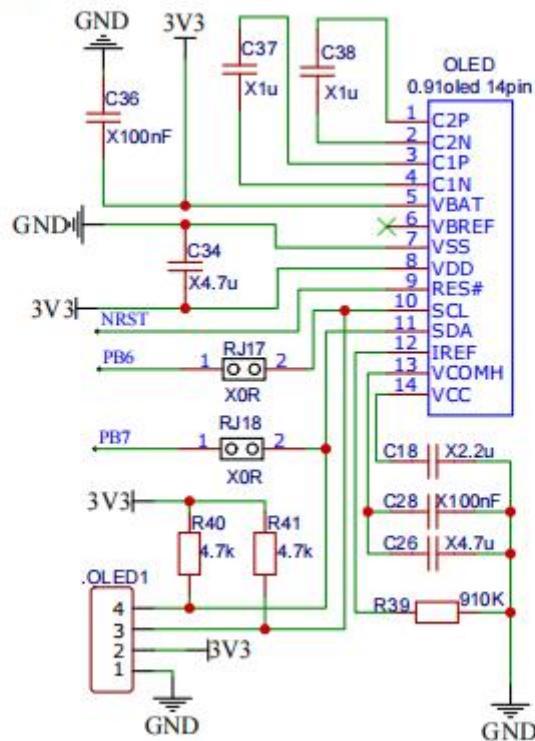


图 3.2.5 OLED

OLED 显示屏我们采用的是 0.91 寸的,考虑到该模块的尺寸以及节约宝贵的 IO 口资源这里我们采用了 12C 通信,将 SCL 连接到了 MCU 的 PB6;SDA 连接到 PB7 上。OLED 的复位 NRST 我们接到了 MCU 的复位引脚上。

### 3.2.6 蓝牙接口

蓝牙模块接口

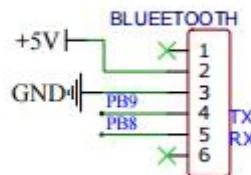


图 3.2.6 蓝牙接口

我们预留了蓝牙接口，蓝牙的 TXD 与 RXD 与 MCU 的 PB9/PB8 连接。

### 3.2.7 NTC 温度传感器

NTC

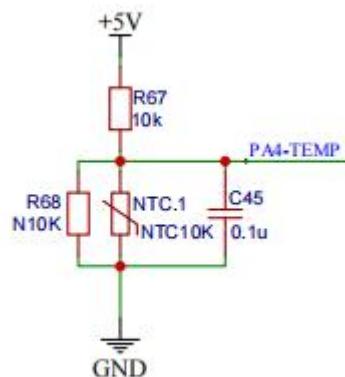
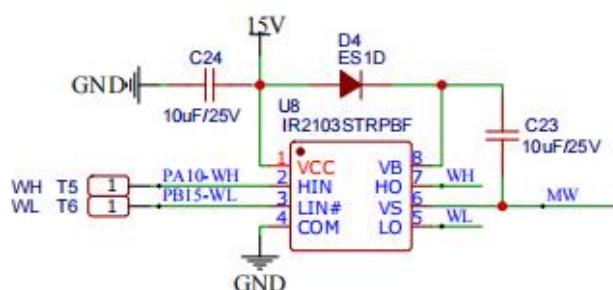
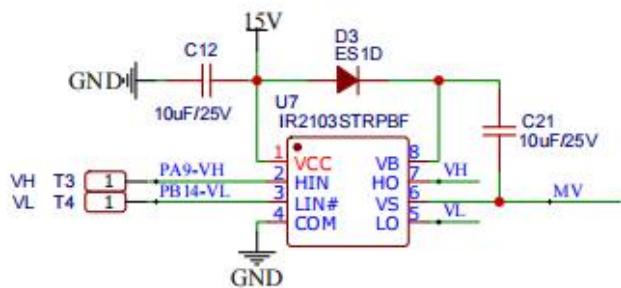
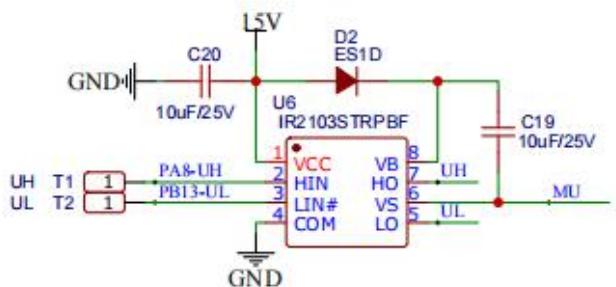


图 3.2.7 蓝牙接口

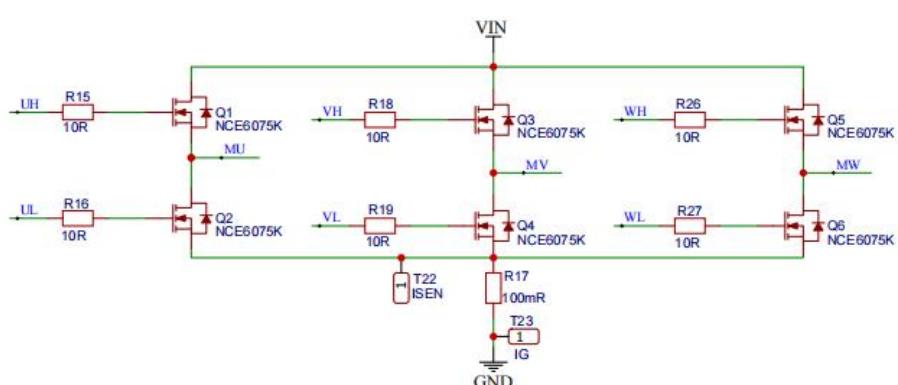
我们增加了一个温度传感器检测实时的一个温度数据。

注意：R68 与 NTC 任意焊接一个 默认焊接 NTC

### 3.2.8 6 路 MOS 管与 EG3013 驱动电路



电桥



运用 EG3013 来驱动我们的 MOS 管上下桥，使电机运转。

# 第四章 CW32\_BLAC\_EVA 评估板基础实例

## 4.1 实例一 流水灯实验

视频连接: [https://www.bilibili.com/video/BV1294y1U78a?spm\\_id\\_from=333.999.0.0](https://www.bilibili.com/video/BV1294y1U78a?spm_id_from=333.999.0.0)

文件夹路径: CW32\_BLDC\_EVA 电机板资料 V4\LCD 电机板例程-MDK\1 GPIO LED--流水灯实验

### 4.1.1 功能要求

- ① 在灵活使用 MDK 环境的条件下，熟悉实现平台的仿真与下载方法。
- ② 掌握 CW32 单片机 CW\_GPIO 口的输出控制。
- ③ 实现实验平台主板区 LED1-LED3 指示灯的流水功能。

### 4.1.2 硬件原理

LED 电路图如图所示:

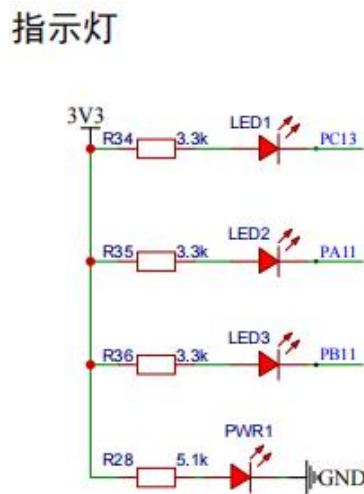


图 4.1.2 LED 电路图

由图中可知，当端口电平置为低电平时，对应的 LED 亮。因此，可以将这些端口连至 MCU 普通 GPIO 口，通过程序对每个 LED 控制，实现流水功能。由图可知 LED1 连接到的是 PC13；LED2 连接到的是 PA11；LED3 连接到的是 PB11。我们直接在程序中对这几个 IO 口进行操作。

### 4.1.3 软件编程

打开我们的源码 CW32\_BLDC\_EVA 电机板资料 V4\LCD 电机板例程-MDK\1 GPIO

## LED--流水灯实验实例。

The screenshot shows the MDK-ARM IDE interface with the following details:

- Project Explorer:** Shows the project structure under "Project: GPIO". It includes a "Target 1" folder containing "USER" (with "interrupts\_cw32f030.c" and "startup\_cw32f030.s"), "lib", and "CMSIS".
- Main File:** The "MAIN.c" file is open in the editor.
- Code Content:** The code implements a flowing light sequence. It starts with system clock configuration (64M), initializes GPIO and RCC, and then enters a loop where it toggles four pins (PB11, PA11, PA13, PC13) sequentially in a loop. The RCC configuration section includes enabling the HSI oscillator and setting the HCLK and PCLK division factors. It also enables the PLL and sets the FLASH latency.

```
1 //系统时钟配置为64M,上电, LED灭。后循环点亮
2
3 #include "main.h"
4
5 void GPIO_Configuration(void);
6 void RCC_Configuration(void);
7
8 int main()
9 {
10     uint32_t i;
11     RCC_Configuration(); //64M时钟配置
12     GPIO_Configuration(); //LED初始化
13
14     while(1)
15     {
16         PB11_TOG();
17         for(i=0;i<1000000;i++);
18         PA11_TOG();
19         for(i=0;i<1000000;i++);
20         PA13_TOG();
21         for(i=0;i<1000000;i++);
22     }
23 }
24
25
26 void RCC_Configuration(void)
27 {
28
29     /* 0. HSI使能并校准 */
30     RCC_HSI_Enable(RCC_HSIOSC_DIVE);
31
32     /* 1. 设置HCLK和PCLK的分频系数 */
33     RCC_HCLKERS_Config(RCC_HCLK_DIV1);
34     RCC_PCLKERS_Config(RCC_PCLK_DIV1);
35
36     /* 2. 使能PLL, 通过PLL倍频到72MHz */
37     RCC_PLL_Enable(RCC_PLLSOURCE_HSI, 8000000, 8); // HSI 默认输出频率8MHz
38 // RCC_PLL_OUT(); //PC13脚输出PLL时钟
39
40     //当使用的时钟源HCLK大于24M, 小于等于48MHz, 设置FLASH 读等待周期为2 cycle
41     //当使用的时钟源HCLK大于48MHz, 设置FLASH 读等待周期为3 cycle
42     RCC_FLASH_CLK_ENABLE();
43     FLASH_SetLatency(FLASH_Latency_3);
44
45 }
```

- Build Output:** A panel at the bottom showing the build status.

图 4.1.3.1

系统时钟配置为 64M, PB11\_TOG 是官方已经定义好的宏定义 (CW\_GPIOB->TOG = bv11), 翻转 LED 灯。

系统内部时钟树如下图所示：

图 4-1 系统内部时钟树

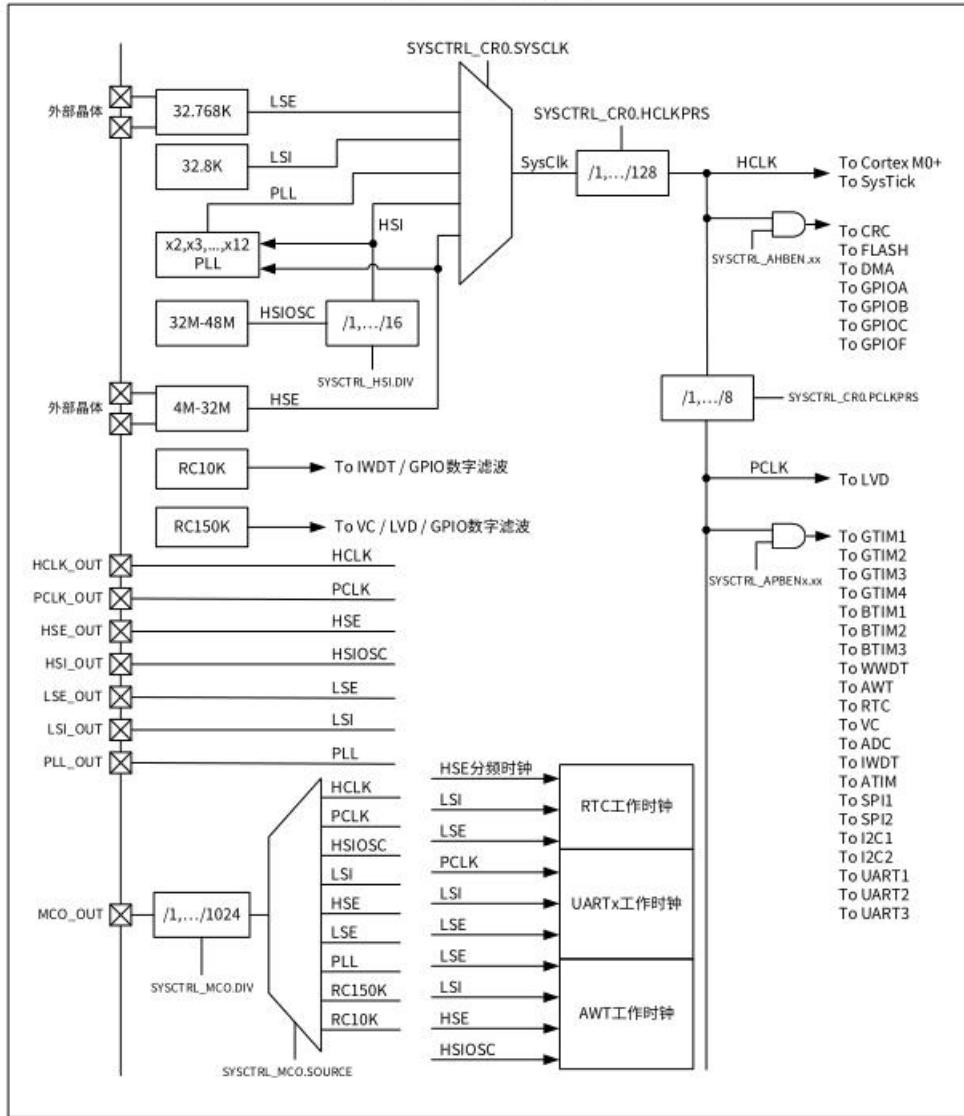


图 4.1.3.2

这里我们使用的是内部高速 RC 振荡器时钟 HSIOSC，经过分频器分频后产生 HSI 时钟，分频系数通过内置高频时钟控制寄存器 SYSCTRL\_HSI 的 DIV 位域进行设置，有效分频系数为 1、2、4、6、8、10、12、14、16。HSIOSC 时钟频率固定为 48MHz。

程序中我们使用的 64M，这里我们将 HSIOSC 时钟进行 6 分频，然后使能 PLL，通过 PLL 倍频到 64MHz 如图 4.1.3.3。

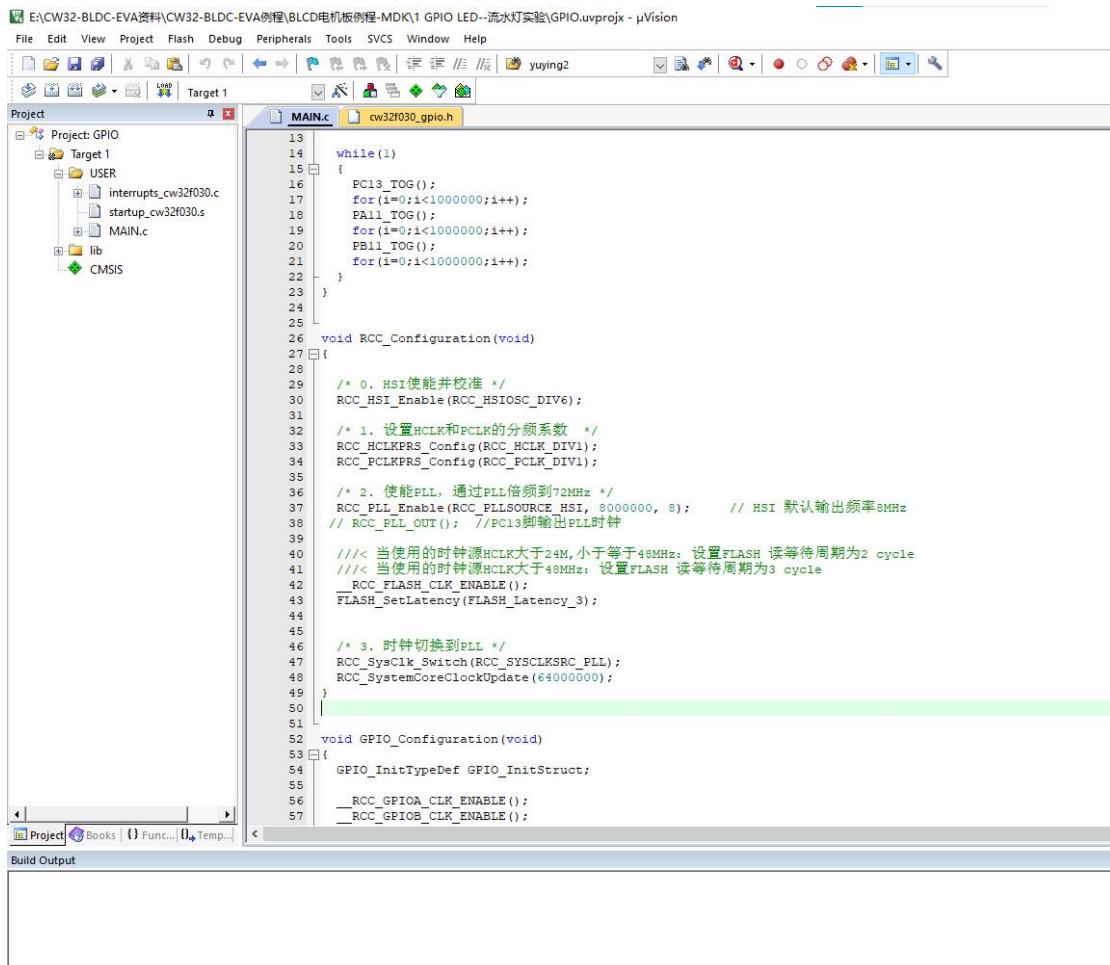


图 4.1.3.3

```

void GPIO_Configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStruct;

    __RCC_GPIOA_CLK_ENABLE();
    __RCC_GPIOB_CLK_ENABLE();
    __RCC_GPIOC_CLK_ENABLE();

    GPIO_InitStruct.IT = GPIO_IT_NONE;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pins = GPIO_PIN_11;
    GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
    GPIO_Init(CW_GPIOB, &GPIO_InitStruct);

    GPIO_InitStruct.Pins = GPIO_PIN_13;
    GPIO_Init(CW_GPIOC, &GPIO_InitStruct);
}

```

```
GPIO_InitStruct.Pins = GPIO_PIN_11;
GPIO_Init(CW_GPIOA, &GPIO_InitStruct);
GPIO_WritePin(CW_GPIOB,GPIO_PIN_11,GPIO_Pin_Set);
GPIO_WritePin(CW_GPIOC,GPIO_PIN_13,GPIO_Pin_Set);
GPIO_WritePin(CW_GPIOA,GPIO_PIN_11,GPIO_Pin_Set);
}

这里是对 LED 灯的初始化
void GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pins, GPIO_PinState PinState)
{}//设置多个引脚，指定引脚电平。
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
配置成推挽输出。
```

#### 4.1.4 软件运行

在下载完之后，我们可以看到 LED1~LED3 循环点亮。

## 4.2 实例二 按键指示灯实验

视频连接: [https://www.bilibili.com/video/BV1G541197Fo?spm\\_id\\_from=333.999.0.0](https://www.bilibili.com/video/BV1G541197Fo?spm_id_from=333.999.0.0)

文件夹路径: CW32\_BLDC\_EVA 电机板资料 V4\LCD 电机板例程-MDK\2 LED&KEY--按键指示灯

### 4.2.1 功能要求

- ① 掌握 CW32 单片机 GPIO 口的输入输出配置。
- ② 熟悉按键工作原理及编程方法。
- ③ 按下按键对应的 LED 灯翻转。

### 4.2.2 硬件设计

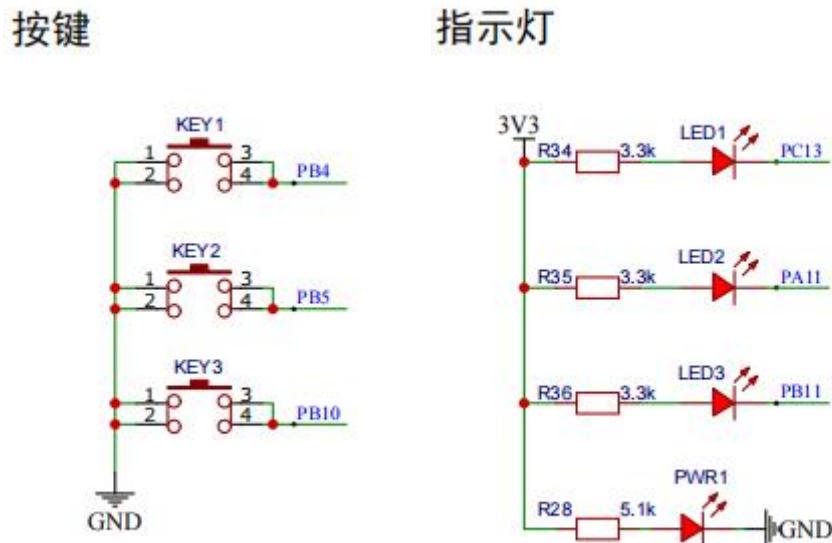
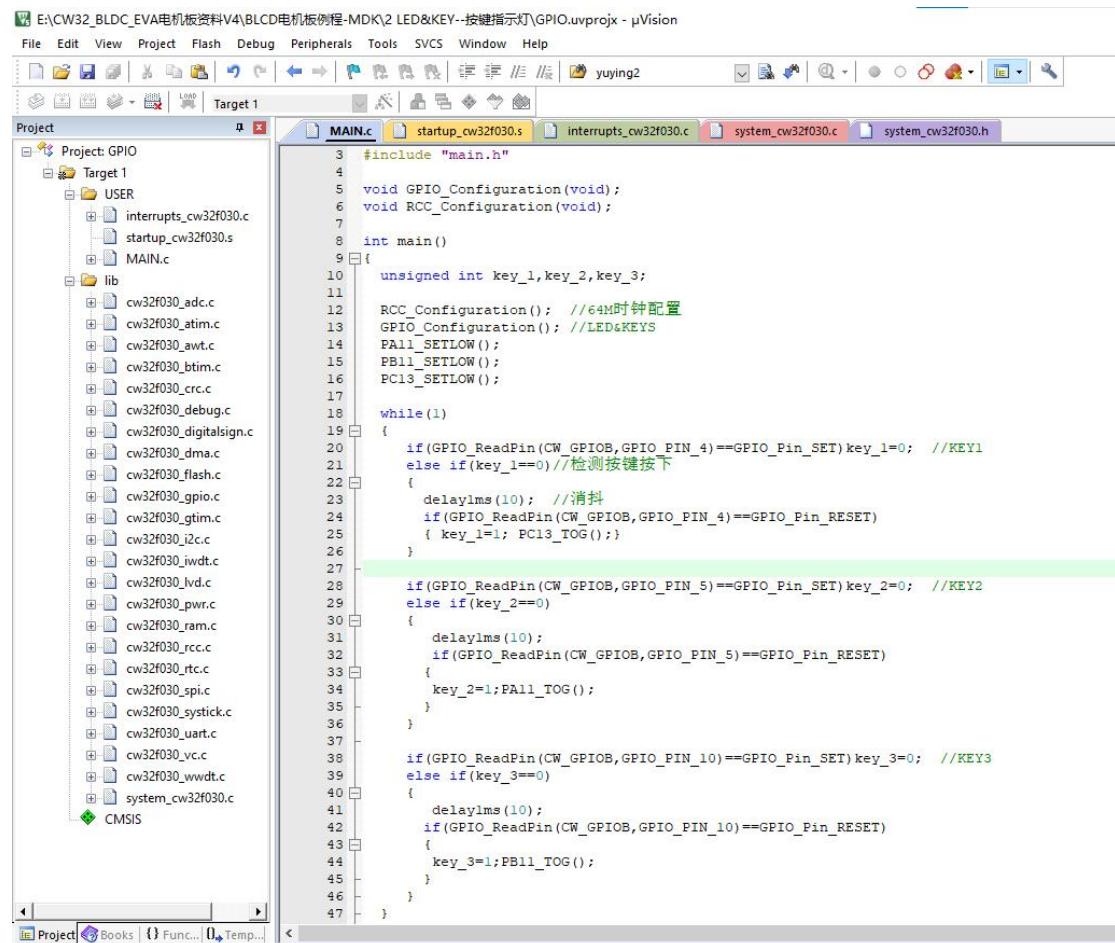


图 4.2.2

它的原理我们前面已经讲过, 这里就不再重复, 这里我们用到了 3 个按键以及 3 个 LED 灯, 对应的 IO 口分别是 KEY1-PB4;KEY2-PB5;KEY3-PB10;  
LED1-PC13;LED2-PA11;LED3-PB11。

### 4.2.3 软件设计

打开 CW32\_BLDC\_EVA 电机板资料 V4\BLCD 电机板例程-MDK\2 LED&KEY--按键指示灯实例



The screenshot shows the MDK-2 IDE interface with the project 'GPIO' open. The 'MAIN.c' file is the active code editor. The code implements a loop to read three GPIO pins (GPIOB pins 4, 5, and 10) and set their corresponding key variables (key\_1, key\_2, key\_3) based on the state. It includes a 10ms delay between reads to debounce the keys.

```
#include "main.h"
void GPIO_Configuration(void);
void RCC_Configuration(void);
int main()
{
    unsigned int key_1, key_2, key_3;
    RCC_Configuration(); //64M时钟配置
    GPIO_Configuration(); //LED&KEYS
    PA11_SETLOW();
    PB11_SETLOW();
    PC13_SETLOW();
    while(1)
    {
        if(GPIO_ReadPin(CW_GPIOB,GPIO_PIN_4)==GPIO_Pin_Set)key_1=0; //KEY1
        else if(key_1==0)//检测按键按下
        {
            delay1ms(10); //消抖
            if(GPIO_ReadPin(CW_GPIOB,GPIO_PIN_4)==GPIO_Pin_Reset)
                { key_1=1; PC13_TOG(); }
        }
        if(GPIO_ReadPin(CW_GPIOB,GPIO_PIN_5)==GPIO_Pin_Set)key_2=0; //KEY2
        else if(key_2==0)
        {
            delay1ms(10);
            if(GPIO_ReadPin(CW_GPIOB,GPIO_PIN_5)==GPIO_Pin_Reset)
            {
                key_2=1; PA11_TOG();
            }
        }
        if(GPIO_ReadPin(CW_GPIOB,GPIO_PIN_10)==GPIO_Pin_Set)key_3=0; //KEY3
        else if(key_3==0)
        {
            delay1ms(10);
            if(GPIO_ReadPin(CW_GPIOB,GPIO_PIN_10)==GPIO_Pin_Reset)
            {
                key_3=1; PB11_TOG();
            }
        }
    }
}
```

图 4.2.3

这里我们采用的是下降沿检测法，delay1ms(10) 延时的作用是进行一个消抖处理，之所以要进行一个消抖是因为按键所用开关为机械弹性开关，当机械触点断开、闭合时，由于机械触点的弹性作用，一个按键开关在闭合时不会马上稳定地接通，在断开时也不会一下子断开。因而在闭合及断开的瞬间均伴随有一连串的抖动，为了不产生这种现象而作的措施就是按键消抖。

关键代码如下：

```
while(1)
{
    if(GPIO_ReadPin(CW_GPIOB,GPIO_PIN_4)==GPIO_Pin_Set)key_1=0;
    //KEY1
    else if(key_1==0)//检测按键按下
    {
        delay1ms(10); //消抖
        if(GPIO_ReadPin(CW_GPIOB,GPIO_PIN_4)==GPIO_Pin_Reset)
        { key_1=1; PC13_TOG(); }
```

```

    }

    if(GPIO_ReadPin(CW_GPIOB, GPIO_PIN_5)==GPIO_Pin_Set)key_2=0;
//KEY2
else if(key_2==0)
{
    delay1ms(10);
    if(GPIO_ReadPin(CW_GPIOB, GPIO_PIN_5)==GPIO_Pin_Reset)
    {
        key_2=1;PA11_TOG();
    }
}

if(GPIO_ReadPin(CW_GPIOB, GPIO_PIN_10)==GPIO_Pin_Set)key_3=0;
//KEY3
else if(key_3==0)
{
    delay1ms(10);
    if(GPIO_ReadPin(CW_GPIOB, GPIO_PIN_10)==GPIO_Pin_Reset)
    {
        key_3=1;PB11_TOG();
    }
}
}

```

#### 4.2.4 下载与验证

下载完成之后，上电 LED 灯全亮；按 KEY1 时，LED1 灯反转；按 KEY2 时，LED2 灯反转；按 KEY3 时，LED3 灯反转。

## 4.3 实例三 蜂鸣器实验

视频连接: [https://www.bilibili.com/video/BV1G541197Fo?spm\\_id\\_from=333.999.0.0](https://www.bilibili.com/video/BV1G541197Fo?spm_id_from=333.999.0.0)

文件夹路径: CW32\_BLDC\_EVA 电机板资料 V4\LCD 电机板例程-MDK\3 BEEP-- 蜂鸣器

### 4.3.1 功能要求

- ① 掌握 CW32 单片机 GPIO 口的输出控制
- ② 熟悉蜂鸣器电路的设计与编程。
- ③ 实现蜂鸣器发声的功能。

### 4.3.2 硬件设计

电路图如图 4.3.2 所示。

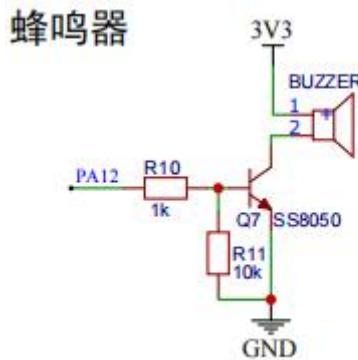


图 4.3.2 蜂鸣器电路图

蜂鸣器是一种一体化结构的电子讯响器，采用直流电压供电，广泛应用于计算机、打印机、复印机、报警器、电子玩具、汽车电子设备、电话机、定时器等电子产品中作发声器件。蜂鸣器主要分为压电式蜂鸣器和电磁式蜂鸣器两种类型。CW32 板载的是有源蜂鸣器。

由图中可知，当端口 ZB 电平置为高电平时，Q7 三极管导通，蜂鸣器电路形成回路，蜂鸣器发声。因此，可以将此端口连至 MCU 普通 GPIO 口，通过程序对 ZB 端口控制制，实现蜂鸣器的发声功能。

实验所用 IO 口: BUZZER--PA12

### 4.3.3 软件设计

打开 CW32\_BLDC\_EVA 电机板资料 V4\LCD 电机板例程-MDK\3 BEEP-- 蜂鸣器实例

The screenshot shows the MDK-ARM IDE interface with the project 'E:\CW32\_BLDC\_EVA电机板资料V4\BLDC电机板例程-MDK\3 BEEP--蜂鸣器(GPIO\_uvprojx - μVision)' open. The Project Explorer on the left shows files like 'interrupts\_cw32f030.c', 'startup\_cw32f030.s', and 'MAIN.c'. The main window displays the 'MAIN.c' source code:

```
1 /* 
2  * 系统时钟配置为64M,上电, LED1亮. 蜂鸣器响
3  * BEEP_IO:PA12
4 */
5
6 #include "main.h"
7
8 void GPIO_Configuration(void);
9 void RCC_Configuration(void);
10
11 int main()
12 {
13     uint32_t i;
14     RCC_Configuration(); // 64M时钟配置
15     GPIO_Configuration(); // LED&BEEP
16
17     while(1)
18     {
19         PA12_SETHIGH();
20         for(i=0;i<1000000;i++);
21         PA12_SETLOW();
22         for(i=0;i<1000000;i++);
23     }
24 }
25
26
27 void RCC_Configuration(void)
28 {
29     /* 0. HSI使能并校准 */
30     RCC_HSI_Enable(RCC_HSIOSC_DIV6);
31
32     /* 1. 设置HCLK和PCLK的分频系数 */
33     RCC_HCLKPR_Config(RCC_HCLK_DIV1);
34     RCC_PCLKPR_Config(RCC_PCLK_DIV1);
35
36     /* 2. 使能PLL, 通过PLL倍频到72MHz */
37     RCC_PLL_Enable(RCC_PLLSOURCE_HSI, 8000000, 8); // HSI 默认输出频率8MHz
38     // RCC_PLL_OUT(); // PC13脚输出PLL时钟
39
40     // < 当使用的时钟源HCLK大于24M, 小于等于48MHz: 设置FLASH 读等待周期为2 cycle
41     // < 当使用的时钟源HCLK大于48MHz: 设置FLASH 读等待周期为3 cycle
42     RCC_FLASH_CLK_ENABLE();
43     FLASH_SetLatency(FLASH_Latency_3);
44
45 }
```

图 4.3.3

这里的 PA12\_SETHIGH 与 PA12\_SETLOW 是官方已经定义好的宏定义，对单个 IO 口进行高低电平操作。

#### 4.3.4 下载与验证

在下载完之后，我们可听到蜂鸣器响一次听一次。

## 4.4 实例四 UART 串口实验

视频连接: [https://www.bilibili.com/video/BV1mU4y12Fv?spm\\_id\\_from=333.999.0.0](https://www.bilibili.com/video/BV1mU4y12Fv?spm_id_from=333.999.0.0)

文件夹路径: CW32\_BLDC\_EVA 电机板资料 V4\LCD 电机板例程-MDK\4 UART--串口实验

### 4.4.1 功能要求

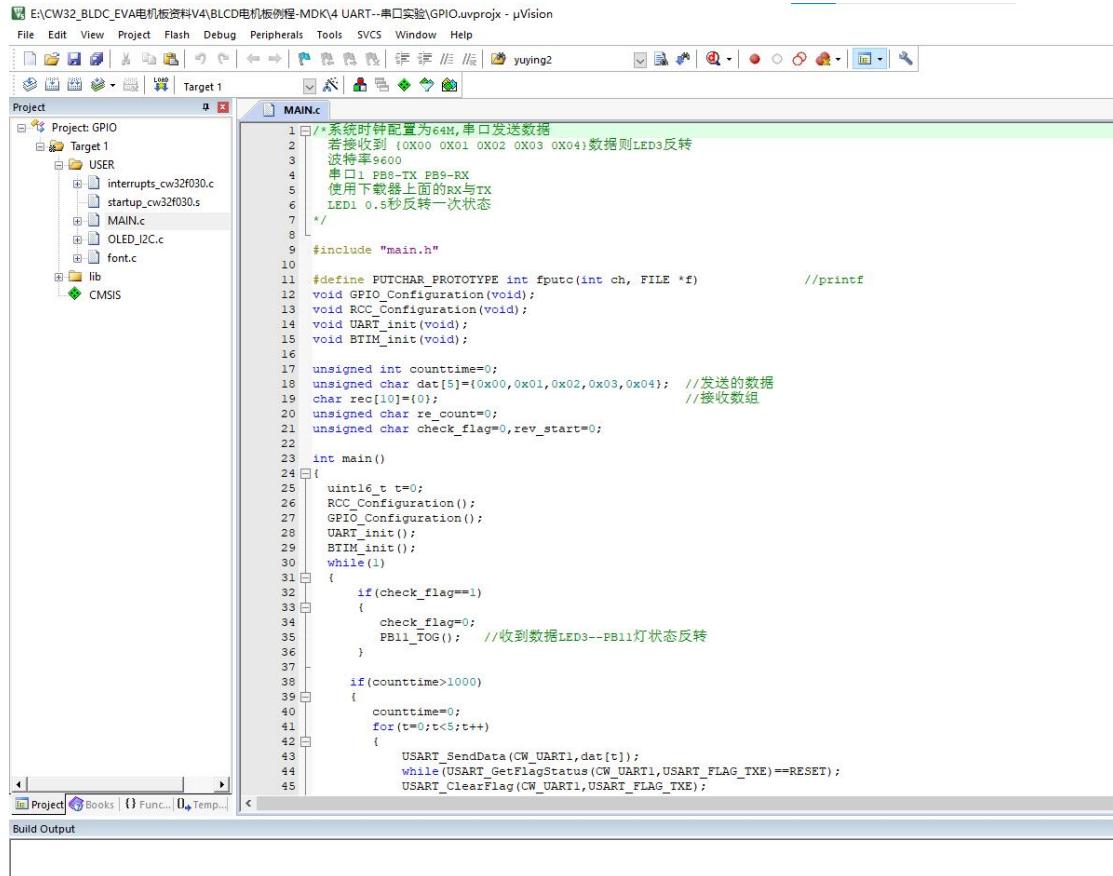
- ①熟悉 CW32 的串口通信原理。
- ②通过串口完成数据收发。

### 4.4.2 硬件设计

所用到的硬件资源有: PB8,PB9 以及 LED1,LED3

### 4.4.3 软件设计

打开 CW32\_BLDC\_EVA 电机板资料 V4\LCD 电机板例程-MDK\4 UART--串口实验实例



The screenshot shows the MDK-ARM IDE interface with the project 'Project GPIO' open. The main window displays the 'MAIN.c' source code. The code is as follows:

```
1 /* 系统时钟配置为64M,串口发送数据
2  若接收到 (0X00 0X01 0X02 0X03 0X04)数据则LED3反转
3  波特率9600
4  串口1 PB8-TX PB9-RX
5  使用下拉器上面的RX与TX
6  LED1 0.5秒反转一次状态
7 */
8
9 #include "main.h"
10
11 #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f) //printf
12 void GPIO_Configuration(void);
13 void RCC_Configuration(void);
14 void UART_Init(void);
15 void BTIM_Init(void);
16
17 unsigned int counttime=0;
18 unsigned char dat[5]={0x00,0x01,0x02,0x03,0x04}; //发送的数据
19 char rec[10]={0}; //接收数组
20 unsigned char re_count=0;
21 unsigned char check_flag=0,rev_start=0;
22
23 int main()
24 {
25     uint16_t t=0;
26     RCC_Configuration();
27     GPIO_Configuration();
28     UART_Init();
29     BTIM_Init();
30     while(1)
31     {
32         if(check_flag==1)
33         {
34             check_flag=0;
35             PB11_TOG(); //收到数据LED3--PB11灯状态反转
36         }
37         if(counttime>1000)
38         {
39             counttime=0;
40             for(t=0;t<5;t++)
41             {
42                 USART_SendData(CW_UART1,dat[t]);
43                 while(USART_GetFlagStatus(CW_UART1,USART_FLAG_TXE)==RESET);
44                 USART_ClearFlag(CW_UART1,USART_FLAG_TXE);
45             }
46         }
47     }
48 }
```

图 4.4.3.1

CW32F030 内部集成 3 个通用异步收发器 (UART), 支持异步全双工、同步半双工和

单线半双工模式，支持硬件数据流控和多机通信；可编程数据帧结构，可以通过小数波特率发生器提供宽范围的波特率选择。

UART 发送控制环节的时序图如下图所示：

图 18-4 异步工作模式发送控制时序（发送两个字节数据）

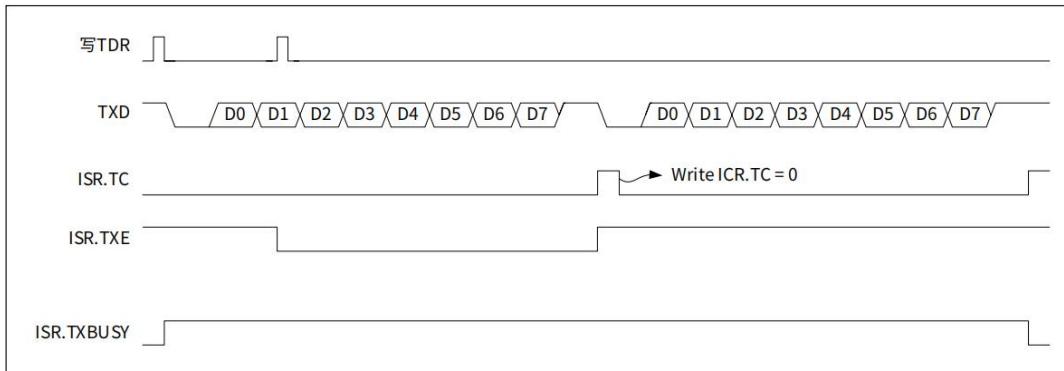


图 4.4.3.2

UART 接收控制环节的时序图如下图所示：

图 18-5 异步工作模式接收控制时序（接收两个字节数据）

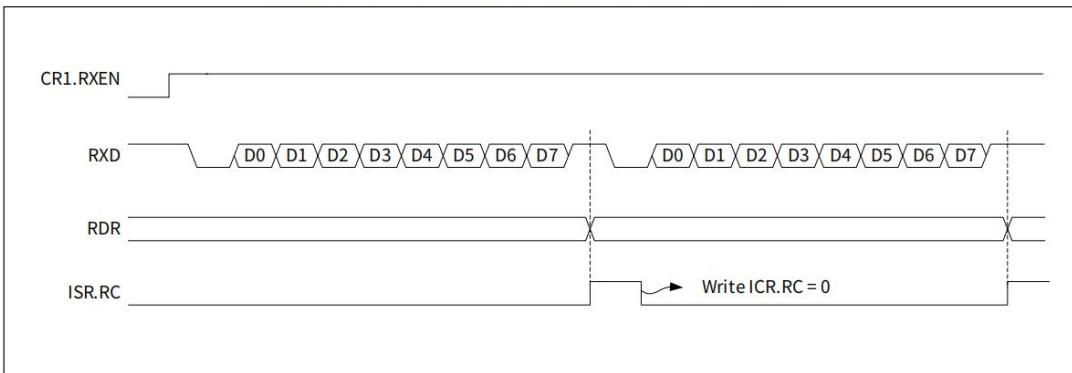


图 4.4.3.3

中断初始化配置代码：

```
void UART_init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;

    RCC_APBPeriphClk_Enable2(RCC_APB2_PERIPH_UART1, ENABLE);
    RCC_AHBPeriphClk_Enable( RCC_AHB_PERIPH_GPIOB, ENABLE);

    PB08_AFx_UART1TXD();
    PB09_AFx_UART1RXD();

    GPIO_InitStructure.Pins = GPIO_PIN_8;
    GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStructure.Speed = GPIO_SPEED_HIGH;
    GPIO_Init(CW_GPIOB, &GPIO_InitStructure);
```

```

GPIO_InitStructure.Pins = GPIO_PIN_9;
GPIO_InitStructure.Mode = GPIO_MODE_INPUT_PULLUP;
GPIO_Init(CW_GPIOB, &GPIO_InitStructure);

USART_InitStructure USART_BaudRate = 9600;
USART_InitStructure USART_Over = USART_Over_16;
USART_InitStructure USART_Source = USART_Source_PCLK;
USART_InitStructure USART_UclkFreq = 64000000;
USART_InitStructure USART_StartBit = USART_StartBit_FE;
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(CW_UART1, &USART_InitStructure);

//使能 UARTx RC 中断
USART_ITConfig(CW_UART1, USART_IT_RC, ENABLE);
//优先级，无优先级分组
NVIC_SetPriority(UART1_IRQn, 0);
//UARTx 中断使能
NVIC_EnableIRQ(UART1_IRQn);

}

void UART1_IRQHandler(void)
{
    unsigned char TxRxBuffer;
    if(USART_GetITStatus(CW_UART1, USART_IT_RC) != RESET)
    {
        USART_ClearITPendingBit(CW_UART1, USART_IT_RC);
        TxRxBuffer = USART_ReceiveData_8bit(CW_UART1);

        rec[re_count]=TxRxBuffer;
        if (re_count == 0&&rec[0]==0x00) //判断帧头
        {
            rev_start=1;
            re_count++;
        }
        else if(rev_start==1)

```

```

{
    if(re_count==1&&rec[1]!=0x01)
    {
        re_count=0;rev_start=0;
    }
    else if(re_count==4)           //计算总个数
    {check_flag=1;re_count=0;rev_start=0;} //如果对，那么将标志位置 1,
并清除里面的值从新接收
    else
        re_count++;
}
}

```

串口中断服务函数里面接收数据，判断帧头是否等于 0X00,如果是则继续接收后面的数据，如果不是则将收到的数据清零从新接收。共接收 0-4,5 个数，接收完成之后将 check\_flag 置 1， 表示数据接收完成。

#### 4.4.4 下载与验证

下载完成之后可以看到 LED1 0.5s 翻转一次，使用我们下载器上面提供的虚拟串口，TX 接到 PB9,RX 接到 PB8 之后打开串口助手勾选十六进制显示可以看到串口 2s 收到一次数据，勾选 十六进制发送 00， 01， 02， 03， 04 可以控制评估板上面 LED2 的



## 4.5 实例五 定时器应用实验

视频连接: [https://www.bilibili.com/video/BV19A4y1Z7c6?spm\\_id\\_from=333.999.0.0](https://www.bilibili.com/video/BV19A4y1Z7c6?spm_id_from=333.999.0.0)

文件夹路径: CW32\_BLDC\_EVA 电机板资料 V4\LCD 电机板例程-MDK\5 ATIMER--定时器

### 4.5.1 功能要求

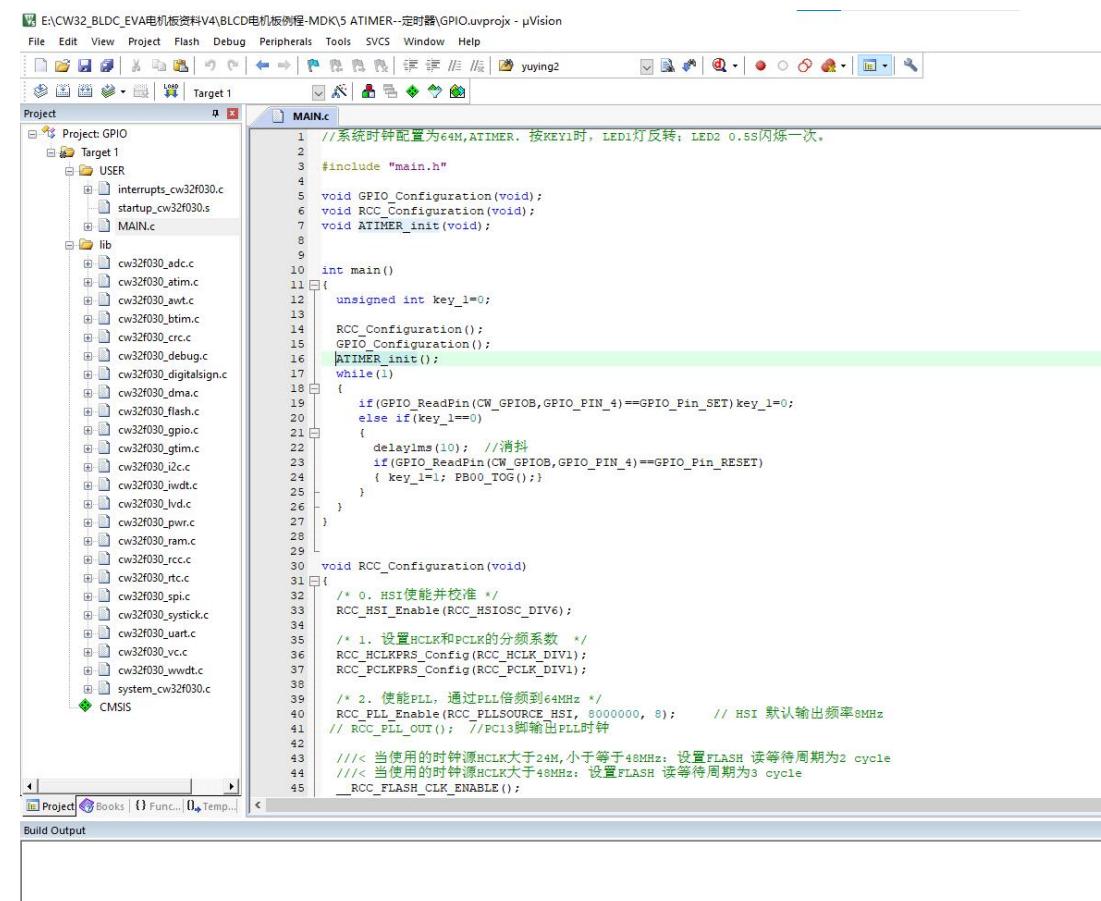
- ①熟悉 CW32 的时钟配置。
- ②CW32\_BLDC\_EVA 评估板上具有 3 个用户 LED 灯，使用这个硬件资源，完成定时器的基本应用实验。
- ③按 KEY1 时，LED1 灯反转；LED2 0.5S 闪烁一次。

### 4.5.2 硬件设计

用到的资源有：LED1,LED2 以及按键 KEY1,高级定时器 ATIMER。

### 4.5.3 软件设计

打开 CW32\_BLDC\_EVA 电机板资料 V4\LCD 电机板例程-MDK\5 ATIMER--定时器实例



The screenshot shows the MDK-5 IDE interface with the following details:

- Project Explorer:** Shows the project structure under "Project: GPIO". It includes a "Target 1" folder containing "USER" and "lib" subfolders. "USER" contains files like "interrupts\_cw32f030.c", "startup\_cw32f030.s", and "MAIN.c". "lib" contains numerous CW32F030 peripheral driver files.
- Code Editor:** Displays the "MAIN.c" file with C code. The code initializes the system clock, configures GPIO pins for KEY1 and LEDs, and sets up an ATIMER. A specific section of the code is highlighted in green:

```
ATIMER_init();
```
- Toolbars and Menus:** Standard MDK-5 menu items like File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, Help are visible at the top.
- Status Bar:** Shows "Build Output" at the bottom.

图 4.5.3.1

CW32F030x6/x8 微控制器内部集成多达四个通用定时器、三个基本定时器和一个高级控制定时器。

定时器类型	定时器	计数器位宽	计数方式	分频因子	DMA请求	捕获 / 比较通道	互补输出
高级定时器	ATIM	16 位	上 / 下 / 上下	$2^N(N=0..7)$	YES	6	3
通用定时器	GTIM1	16 位	上 / 下 / 上下	$2^N(N=0..15)$	YES	4	1
	GTIM2	16 位	上 / 下 / 上下	$2^N(N=0..15)$	YES	4	1
	GTIM3	16 位	上 / 下 / 上下	$2^N(N=0..15)$	YES	4	1
	GTIM4	16 位	上 / 下 / 上下	$2^N(N=0..15)$	YES	4	1
基本定时器	BTIM1	16 位	上	$2^N(N=0..15)$	YES	0	1
	BTIM2	16 位	上	$2^N(N=0..15)$	YES	0	1
	BTIM3	16 位	上	$2^N(N=0..15)$	YES	0	1

图 4.5.3.2

高级定时器 (ATIM)：由一个 16 位的自动重载计数器和 7 个比较单元组成，并由一个可编程的预分频器驱动。ATIM 支持 6 个独立的捕获/比较通道，可实现 6 路独立 PWM 输出或 3 对互补 PWM 输出或对 6 路输入进行捕获。可用于基本的定时 / 计数、测量输入信号的脉冲宽度和周期、产生输出波形 (PWM、单脉冲、插入死区时间的互补 PWM 等)。

通用定时器 (GTIM1..4)：内部集成 4 个通用定时器 (GTIM)，每个 GTIM 完全独立且功能完全相同，各包含一个 16bit 自动重装载计数器并由一个可编程预分频器驱动。GTIM 支持定时器模式、计数器模式、触发启动模式和门控模式 4 种，基本内部集成 4 个通用定时器 (GTIM)，每个 GTIM 完全独立且功能完全相同，各包含一个 16bit 自动重装载计数器并由一个可编程预分频器驱动。GTIM 支持定时器模式、计数器模式、触发启动模式和门控模式 4 种基本工作模式，每组带 4 路独立的捕获 / 比较通道，可以测量输入信号的脉冲宽度 (输入捕获) 或者产生输出波形并由一个可编程预分频器驱动。GTIM 支持定时器模式、计数器模式、触发启动模式和门控模式 4 种基本工作模式，每组带 4 路独立的捕获 / 比较通道，可以测量输入信号的脉冲宽度 (输入捕获) 或者产生输出波形 (输出比较和 PWM)。工作模式，每组带 4 路独形 (输出比较和 PWM)。

基本定时器 (BTIM)：内部集成 3 个基本定时器 (BTIM)，每个 BTIM 完全独立且功能相同，各包含一个 16bit 自动重装载计数器并由一个可编程预分频器驱动。BTIM 支持定时器模式、计数器模式、触发启动模式和门控模式 4 种工作模式，支持溢出事件触发中断请求和 DMA 请求。得益于对触发信号的精细处理设计，使得 BTIM 可以由硬件自动执行触发信号的滤波操作，还能令触发事件产生中断和 DMA 请求。

ATIM 的功能框图如下图所示：

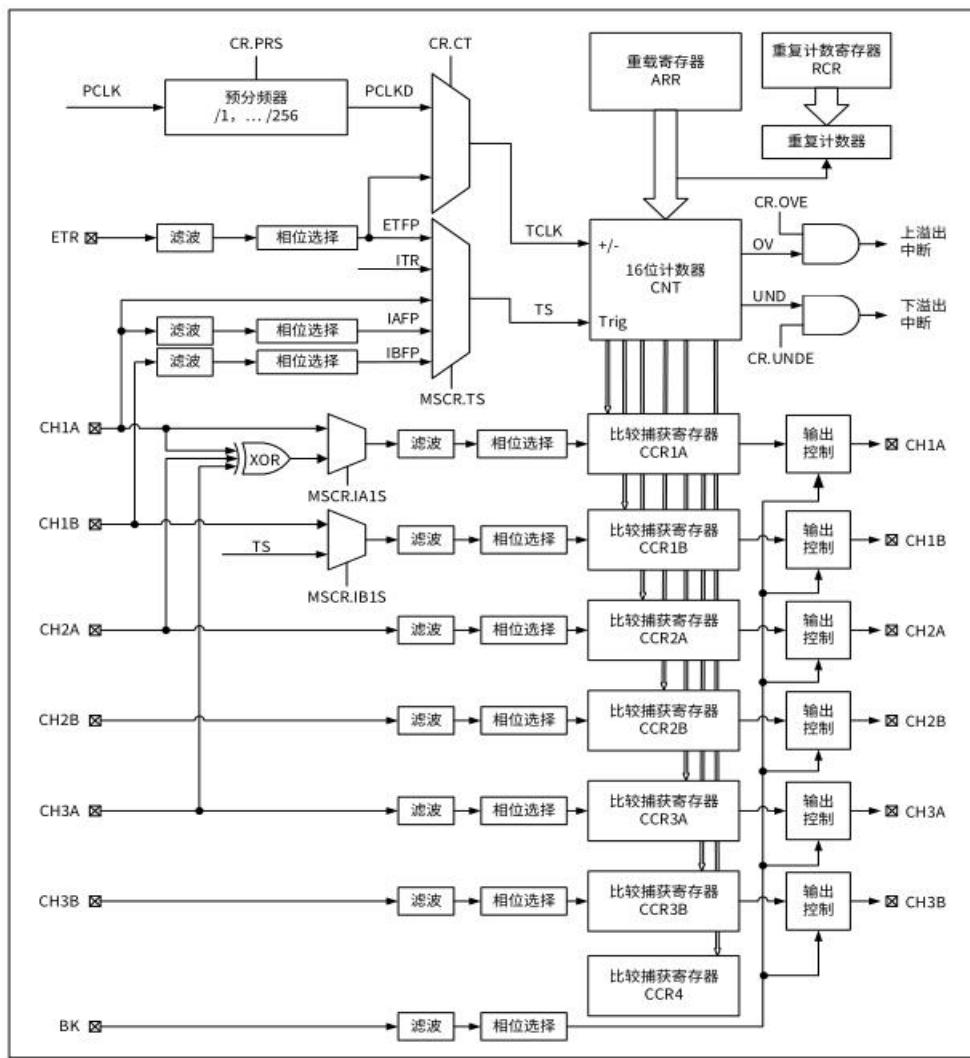


图 4.5.3.3

更多内容请参考 CW32F030 用户手册。

ATM 初始化配置如下，选择向上计数，主频为 64M，8 分频。

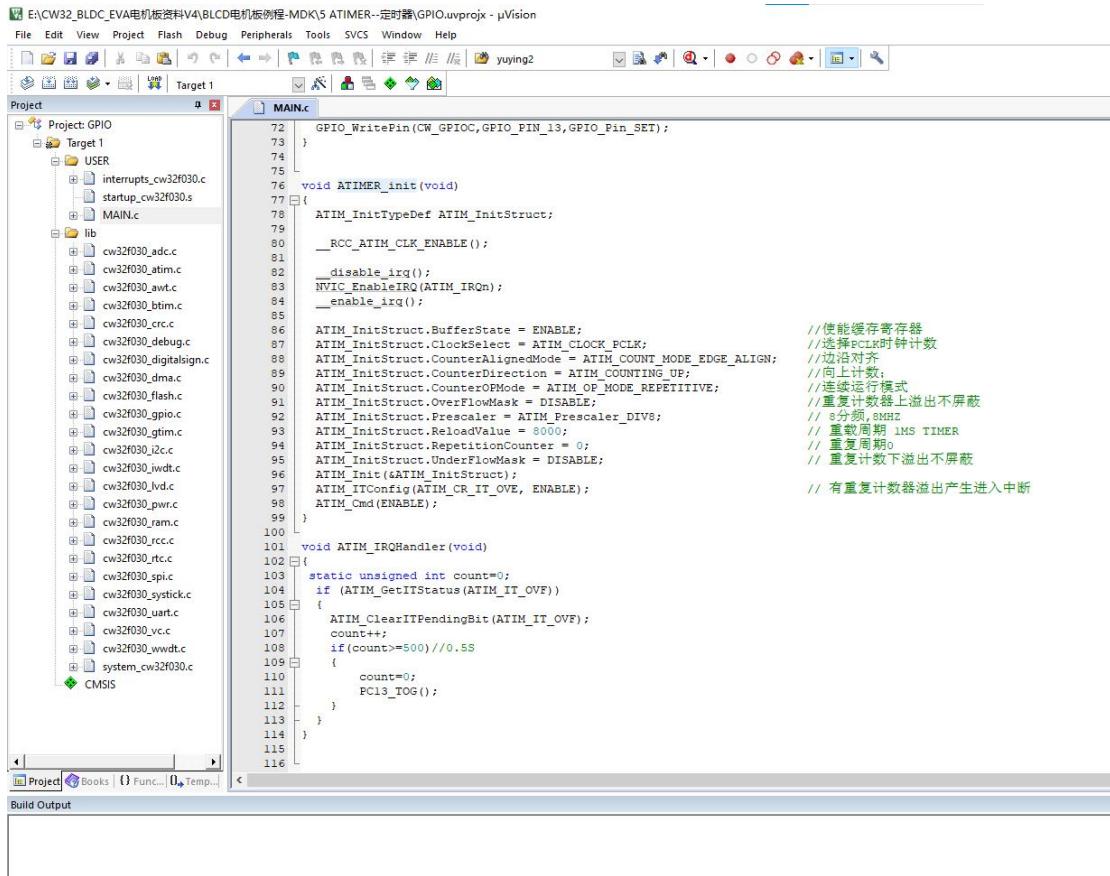


图 4.5.3.4

### ATM 定时器中断服务函数代码

```
void ATIM_IRQHandler(void)
{
    static unsigned int count=0;
    if(ATIM_GetITStatus(ATIM_IT_OVF))
    {
        ATIM_ClearITPendingBit(ATIM_IT_OVF);
        count++;
        if(count>=500)//0.5S
        {
            count=0;
            PC13_TOG();
        }
    }
}
```

}在中断服务函数中，定时器定时 1ms，count 做++运算;500Ms 翻转一次 LED 灯。

### 4.4.4 下载与验证

下载完成之后按 KEY1 时 LED1 灯反转同时 LED2 0.5S 翻转一次。

## 4.6 实例六 OLED 显示实验

视频连接: [https://www.bilibili.com/video/BV1x4y1Z7Uc?spm\\_id\\_from=333.999.0.0](https://www.bilibili.com/video/BV1x4y1Z7Uc?spm_id_from=333.999.0.0)

文件夹路径: CW32\_BLDC\_EVA 电机板资料 V4\BLCD 电机板例程-MDK\8 0.91OLED 实验

### 4.6.1 功能要求

- ①熟悉 OELD 显示原理。
- ②完成 OLED 的显示。

### 4.6.2 硬件设计

OLED 电路图

OLED屏/接口

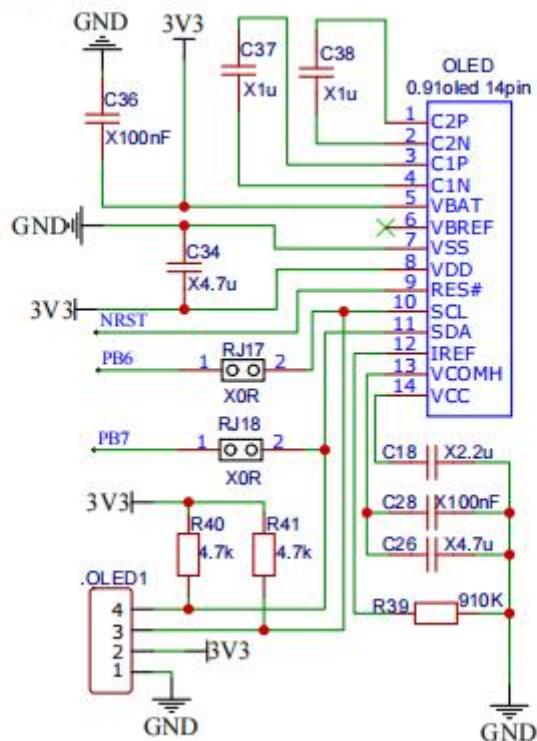


图 4.6.2.1

OLED，即有机发光二极管（Organic Light-Emitting Diode），又称为有机电激光显示（Organic Electroluminescence Display，OLED）。OLED 由于同时具备自发光，不需背光源、对比度高、厚度薄、视角广、反应速度快、可用于挠曲性面板、使用温度范围广、构造及制程较简单等优异之特性，被认为是下一代的平面显示器新兴应用技术。

由图可知我们 OLED 的 SCL 和 SDA 接到了 PB6, PB7 上面，复位引脚接到了 MCU 的复位引脚上，所以复位 MCU 可以复位我们的 OLED 屏。

## I2C 通信协议

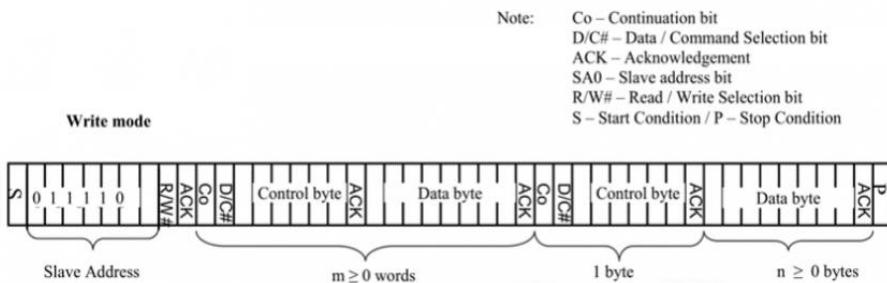


图 4.6.2.2

在 I2C 通信时，先发送一个 7bit 的从设备地址+1bit 的读写位，等待设备的响应。

在从设备应答后，接着发送一个控制字节，该字节决定了后面发送的字节是命令还是数据，然后再等待从设备应答。在从设备再次应答之后，若发送命令，则只发送一字节的命令。若发送数据，可以只发送一个字节的数据连着发送，视情况而定。

### 4.6.3 软件设计

打开 CW32\_BLDC\_EVA 电机板资料 V4\BLCD 电机板例程-MDK\8 0.91OLED 实验实例

The screenshot shows the MDK-8 IDE interface. The project structure on the left includes a 'Project: GPIO' folder containing 'Target 1', 'USER' (with 'interrupts\_cw32f030.c' and 'startup\_cw32f030.s'), 'MAIN.c', 'OLED\_I2C.c', 'font.c', 'lib', and 'CMSIS'. The main window displays the 'MAIN.c' source code:

```
1 /*  
2  系统时钟配置为64M  
3  LED1长亮  
4  OLED屏显示  
5 */  
6  
7 #include "main.h"  
8 #include "OLED_I2C.h"  
9  
10 void GPIO_Configuration(void);  
11 void RCC_Configuration(void);  
12  
13  
14 int main()  
15 {  
16     RCC_Configuration(); //系统时钟  
17     GPIO_Configuration(); //LED初始化  
18     I2C_Init(); //I2C初始化  
19     I2C_OLED_Init();  
20     I2C_OLED_Clear(1);  
21     I2C_OLED_ShowString(0,0,"OLED Init OK");  
22     I2C_OLED_UFdata();  
23     while(1);  
24 }  
25  
26 void RCC_Configuration(void)  
27 {  
28     /* 0. HSI使能并校准 */  
29     RCC_HSI_Enable(RCC_HSIOSC_DIV6);  
30  
31     /* 1. 设置HCLK和PCLK的分频系数 */  
32     RCC_HCLKPFRS_Config(RCC_HCLK_DIV1);  
33     RCC_PCLKPFRS_Config(RCC_PCLK_DIV1);  
34  
35     /* 2. 使能PLL，通过PLL倍频到64MHz */  
36     RCC_PLL_Enable(RCC_PLLSOURCE_HSI, 8000000, 8); // HSI 默认输出频率8MHz  
37 // RCC_PLL_OUT(); //FC13脚输出PLL时钟  
38  
39     //当使用的时钟源HCLK大于24M, 小于等于48MHz: 设置FLASH 读等待周期为2 cycle  
40     //当使用的时钟源HCLK大于48MHz: 设置FLASH 读等待周期为3 cycle  
41     _RCC_FLASH_CLK_ENABLE();  
42     FLASH_SetLatency(FLASH_Latency_3);  
43  
44     /* 3. 时钟切换到PLL */  
45     RCC_SysClk_Switch(RCC_SYSCLKSRC_PLL);
```

图 4.6.3.1

I2C 初始化代码:

```
void I2C_init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    I2C_InitTypeDef I2C_InitStruct;

    __RCC_I2C1_CLK_ENABLE();
    __RCC_GPIOB_CLK_ENABLE();

    PB06_AFx_I2C1SCL();
    PB07_AFx_I2C1SDA();

    GPIO_InitStructure.Pins = GPIO_PIN_6 | GPIO_PIN_7;
    GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_OD;      ///I2C 必须开漏出
    GPIO_InitStructure.Speed = GPIO_SPEED_HIGH;
    GPIO_Init(CW_GPIOB, &GPIO_InitStructure);

    I2C_InitStruct.I2C_BaudEn = ENABLE;
    I2C_InitStruct.I2C_Baud = 0x08;
    I2C_InitStruct.I2C_FLT = DISABLE;
    I2C_InitStruct.I2C_AA = DISABLE; //DISABLE ENABLE

    I2C1_DeInit();
    I2C_Master_Init(CW_I2C1,&I2C_InitStruct); //初始化模块
    I2C_Cmd(CW_I2C1,ENABLE); //模块使能
}
```

此部分代码是对 I2C 的初始化，将 PB06,PB07 分别复用到 SCL 和 SDA,输出配置为开漏输出。

OLED 写函数代码:

```
void     I2C_MasterWriteEepromData1(I2C_TypeDef      *I2Cx,uint8_t      u8Addr,uint8_t
*pu8Data,uint32_t u32Len)
{
    uint8_t u8i=0,u8State;
    I2C_GenerateSTART(I2Cx, ENABLE);
    while(1)
    {
        while(0 == I2C_GetIrq(I2Cx))
        {};
        u8State = I2C_GetState(I2Cx);
        switch(u8State)
        {
            case 0x08: //发送完 START 信号
```

```

I2C_GenerateSTART(I2Cx, DISABLE);
I2C_Send7bitAddress(I2Cx, OLED_ADDRESS, 0X00); //从设备地址发送
break;
case 0x18: //发送完 SLA+W 信号,ACK 已收到
    I2C_SendData(I2Cx, u8Addr); //从设备内存地址发送
    break;
case 0x28: //发送完 1 字节数据：发送 EEPROM 中 memory 地址也会产生,
发送后面的数据也会产生
    I2C_SendData(I2Cx, pu8Data[u8i++]);
    break;
case 0x20: //发送完 SLA+W 后从机返回 NACK
case 0x38: //主机在发送 SLA+W 阶段或者发送数据阶段丢失仲裁 或者
主机在发送 SLA+R 阶段或者回应 NACK 阶段丢失仲裁
    I2C_GenerateSTART(I2Cx, ENABLE);
    break;
case 0x30: //发送完一个数据字节后从机返回 NACK
    I2C_GenerateSTOP(I2Cx, ENABLE);
    break;
default:
    break;
}
if(u8i > u32Len)
{
    I2C_GenerateSTOP(I2Cx, ENABLE); //此顺序不能调换，出停止条件
    I2C_ClearIrq(I2Cx);
    break;
}
I2C_ClearIrq(I2Cx);
}

#define OLED_ADDRESS 0x78 //通过调整 OR 电阻 0x78
#define COM 0x00 // OLED 指令（禁止修改）写命令
#define DAT 0x40 // OLED 数据（禁止修改）写数据

```

#### 4.6.4 下载与验证

下载完成之后可以看到 LED1 常亮，OLED 显示屏显示字符串。

## 4.7 实例七 ADC 电位器实验

视频连接: [https://www.bilibili.com/video/BV1bY4y1B7aX?spm\\_id\\_from=333.999.0.0](https://www.bilibili.com/video/BV1bY4y1B7aX?spm_id_from=333.999.0.0)

文件夹路径: CW32\_BLDC\_EVA 电机板资料 V4\BLCD 电机板例程-MDK\9 AD 电位器实验

### 4.7.1 功能要求

- ①熟悉 ADC 工作原理。
- ②完成 AD 值与电压值的采集与显示。

### 4.7.2 硬件设计

电位器原理图:

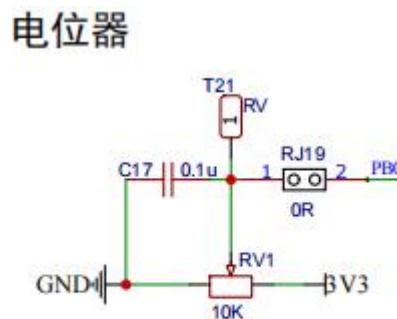


图 4.7.2

CW32F030C8T6 内置 12 位模数转换器具有多达 13 个外部通道和 3 个内部通道（温度传感器、电压基准、VDDA/3），支持单通道或序列模式转换。ADC 属于 CW32 内部资源，实际上我们只需要软件设置就可以正常工作，不过我们需要在外部连接其端口到被测电压上面。这里，我们通过 ADC1 的通道 8 (PB0) 来读取外部电压值。

ADC 通道与 GPIO 对应关系如下：

- \* 通道 0 输入 PA00
- \* 通道 1 输入 PA01
- \* 通道 2 输入 PA02
- \* 通道 3 输入 PA03
- \* 通道 4 输入 PA04
- \* 通道 5 输入 PA05
- \* 通道 6 输入 PA06
- \* 通道 7 输入 PA07
- \* 通道 8 输入 PB00
- \* 通道 9 输入 PB01
- \* 通道 10 输入 PB02
- \* 通道 11 输入 PB10
- \* 通道 12 输入 PB11

```

* 1/3 VDDA(必须使用输入增益)
* BGR_TS(必须使用输入增益)
* Vref1P2(必须使用输入增益)
*/

```

这里我们使用的端口是 PB0, 通道 8。

### 4.7.3 软件设计

打开 CW32\_BLDC\_EVA 电机板资料 V4\BLCD 电机板例程-MDK\9 AD 电位器实验实例

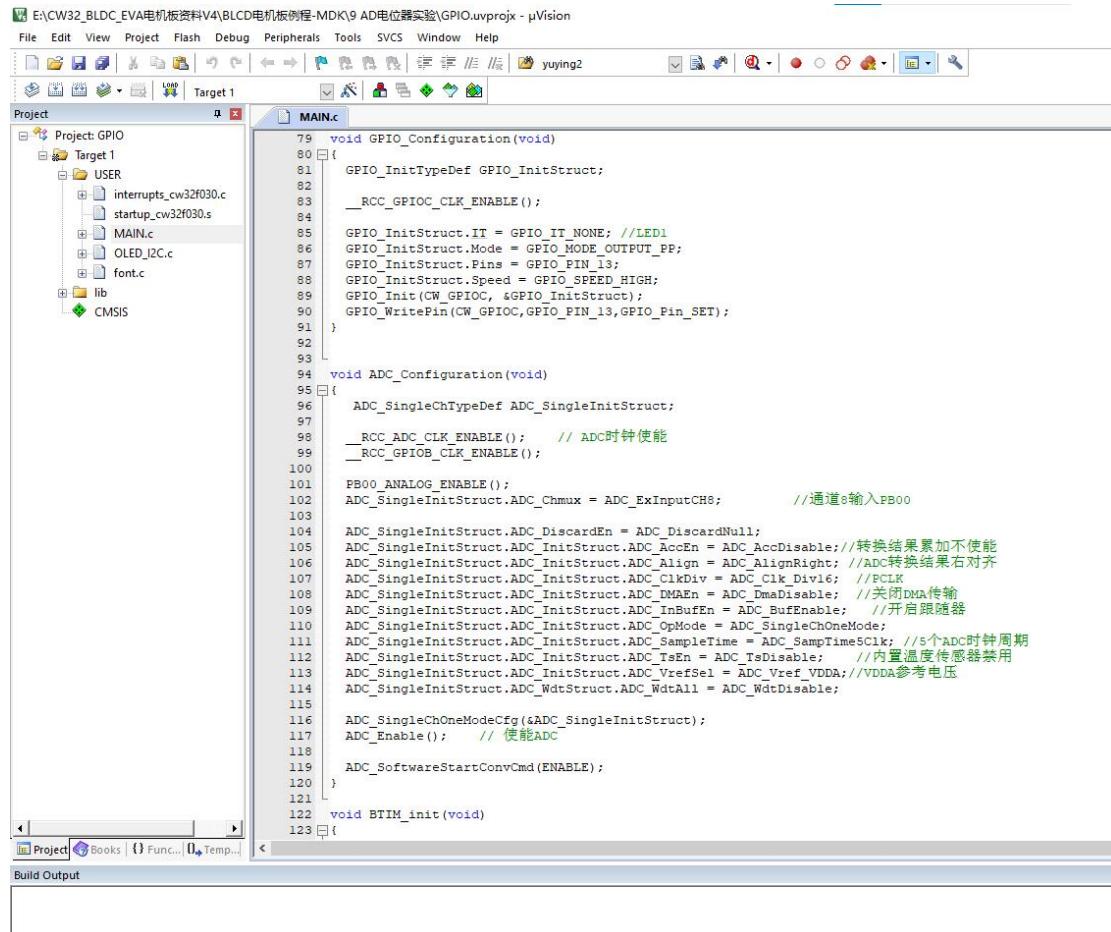


图 4.7.3.1

下面看一下 ADC 的初始化配置：

```

void ADC_Configuration(void)
{
    ADC_SingleChTypeDef ADC_SingleInitStruct;

    __RCC_ADC_CLK_ENABLE(); // ADC 时钟使能
    __RCC_GPIOB_CLK_ENABLE();

    PB00_ANALOG_ENABLE();
    ADC_SingleInitStruct.ADC_Chmux = ADC_ExInputCH8; //通道 8 输入 PB00
}

```

```

ADC_SingleInitStruct.ADC_DiscardEn = ADC_DiscardNull;
ADC_SingleInitStruct.ADC_InitStruct.ADC_AccEn = ADC_AccDisable;//转换结果累加
不使能
ADC_SingleInitStruct.ADC_InitStruct.ADC_Align = ADC_AlignRight; //ADC 转换结果
右对齐
ADC_SingleInitStruct.ADC_InitStruct.ADC_ClkDiv = ADC_Clk_Div16; //PCLK
ADC_SingleInitStruct.ADC_InitStruct.ADC_DMAEn = ADC_DmaDisable; //关闭 DMA 传
输
ADC_SingleInitStruct.ADC_InitStruct.ADC_InBufEn = ADC_BufEnable; //开启跟随
器
ADC_SingleInitStruct.ADC_InitStruct.ADC_OpMode = ADC_SingleChOneMode;
ADC_SingleInitStruct.ADC_InitStruct.ADC_SampleTime = ADC_SampTime5Clk; //5 个
ADC 时钟周期
ADC_SingleInitStruct.ADC_InitStruct.ADC_TsEn = ADC_TsDisable; //内置温度传
感器禁用
ADC_SingleInitStruct.ADC_InitStruct.ADC_VrefSel = ADC_Vref_VDDA;//VDDA 参考电
压
ADC_SingleInitStruct.ADC_WdtStruct.ADC_WdtAll = ADC_WdtDisable;

ADC_SingleChOneModeCfg(&ADC_SingleInitStruct);
ADC_Enable(); // 使能 ADC

ADC_SoftwareStartConvCmd(ENABLE);
}

```

此部分代码函数用于初始化 ADC，开通了 CH8 通道。

AD 转换计算代码：

```

while(1)
{
    if(timecount>200)//每 200ms 单次转换一次，并通过 OLED 显示
    {
        timecount=0;
        ADC_SoftwareStartConvCmd(ENABLE);
        while(ADC_GetITStatus(ADC_IT_EOC))
        {
            ADC_ClearITPendingBit(ADC_IT_EOC);
            adcvalue=ADC_GetConversionValue();
            sprintf(temp_buff, "AD/VAL:%d ", adcvalue);
            I2C_OLED_ShowString(0, 15, temp_buff);
            I2C_OLED_UPdata();
            temp=(float)adcvalue*(3.3/4096);
            sprintf(temp_buff1, "AD/VOL:%0.1f V ", temp);
            I2C_OLED_ShowString(0, 0, temp_buff1);
            I2C_OLED_UPdata();
        }
    }
}

```

```
    }  
}  
}
```

使用定时器做软变量，200ms 转换一次并显示在 OLED 显示屏上  
`temp=(float)adcvalue*(3.3/4096); //电压值换算-做线性运算得到电压值。`

#### 4.7.4 下载与验证

下载完成之后可以看到 LED1 0.5s 翻转一次提示系统运行，旋转电位器查看 OLED 显示屏上的 AD 值和电压值。

## 8 实例八 霍尔开环电机实例

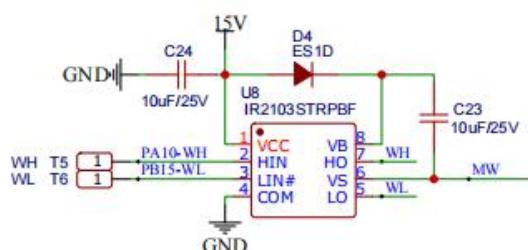
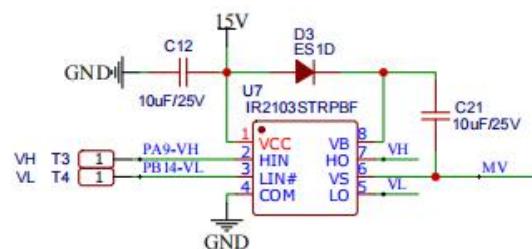
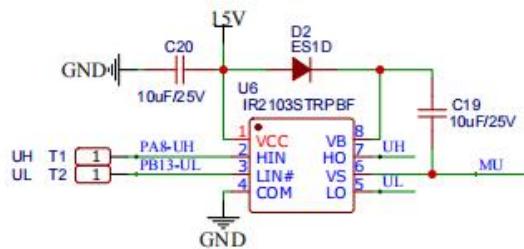
视频连接: <https://space.bilibili.com/1407060490/channel/seriesdetail?sid=2323858>  
文件夹路径: CW32\_BLDC\_EVA 电机板资料 V4\LCD 电机板例程-MDK\11 CW32 BLDC haLL openspeed ok 实例

### 4.8.1 功能要求

- ①熟悉电机工作原理。
- ②完成霍尔值的采集。

### 4.8.2 硬件设计

电机接口电路图与霍尔电路图如下:



## 电桥

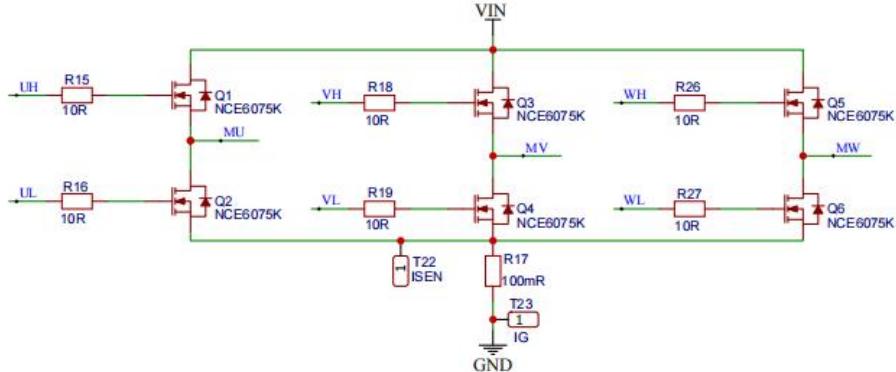


图 4.8.2

由图可知我们电机接口 PA8--UH; PB13--UL;PA9--VH;PB14--VL;PA10--WH;PB15--WL;  
霍尔接口 HA--PA15;HB--PB3;HC--PA2;

### 4.8.3 软件设计

打开 CW32\_BLDC\_EVA 电机板资料 V4\BLCD 电机板例程-MDK\11 CW32 BLDC hall openspeed ok 实例。

```
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
E:\CW32_BLDC_EVA\BLCD\motor\MDK\11\cw32f030.hall.openspeed.ok\GPIO.uvprojx - pVision
Project: GPIO
Target 1
MAIN.c font.c OLED_I2C.c MOTORCONTROL.C cw32f030_gpio.h main.h motorcontrol.l
56     RCC_Configuration(); //64MHz时钟配置
57     GPIO_Configuration(); //LEDs&KEYS
58     I2C_Init(); //I2C init
59     I2C_OLED_Init();
60     ADC_Configuration(); //AD PONITER
61     BTIM_Init(); //10MS USER TIMER
62     ATIM_Init(); //FWN TIMER
63     GTIM_Init(); //HALL TIMER
64     I2C_OLED_Clear(1);
65
66     sprintf(temp_buff1,"PWN: %d %s STOP...",PWNFLA); //输出显示占空比
67     sprintf(temp_buff2,"Real:%d rpm ",RealS); //显示电机转速
68     I2C_OLED_ShowString(0,1,temp_buff1);
69     I2C_OLED_ShowString(0,0,temp_buff2);
70     I2C_OLED_UFdata();
71
72     GPIO_WritePin(CW_GPIOC,GPIO_PIN_13,GPIO_Pin_RESET); //LED ON
73
74     while(1)
75     {
76         if ((CW_DMA->ISR_f.TC1)
77             { //AD DMA 启动
78                 CW_DMA->ICR_f.TC1 = 0;
79                 CW_DMACHANNEL1->CNT=bv16|60000; //MUST RET AGAIN BEFORE CW_DMACHANNEL1->CNT=0
80                 CW_DMACHANNEL1->CSR_f.EN = 1;
81             }
82
83         if(timecount>=10)
84         {
85             timecount=0;//100ms
86
87             //电位器采集的AD代码值在3~4000范围内有效 3对应最大输出, 4000对应最小输出
88             if(ADC_Result_Array>=4000) test=0;
89             else if(ADC_Result_Array<3) test=4000;
90             else test=4000-ADC_Result_Array;
91
92             t=test*0.8;//40*32;
93             test=t; //计算当前 应当输出的值
94
95             if(OutPwm<test)
96
97         }
98     }
99 }
```

图 4.8.3.1

ATIIM 初始化代码：

```
void ATIIM_init(void)
{
    ATIM_InitTypeDef ATIM_InitStruct;
    ATIM_OCIInitTypeDef ATIM_OCInitStruct;
    GPIO_InitTypeDef GPIO_InitStruct;

    __RCC_ATIM_CLK_ENABLE();
    __RCC_GPIOA_CLK_ENABLE();
    __RCC_GPIOB_CLK_ENABLE();

    PA08_AFx_ATIMCH1A();
    PA09_AFx_ATIMCH2A();
    PA10_AFx_ATIMCH3A();

    GPIO_InitStruct.IT = GPIO_IT_NONE;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pins = GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
    GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
    GPIO_Init(CW_GPIOB, &GPIO_InitStruct);

    GPIO_InitStruct.IT = GPIO_IT_NONE;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pins = GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10;
    GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
    GPIO_Init(CW_GPIOA, &GPIO_InitStruct);

    PWM_AL_OFF;
    PWM_BL_OFF;
    PWM_CL_OFF;

    ATIM_InitStruct.BufferState = DISABLE;
    ATIM_InitStruct.ClockSelect = ATIM_CLOCK_PCLK;
    ATIM_InitStruct.CounterAlignedMode = ATIM_COUNT_MODE_EDGE_ALIGN;
    ATIM_InitStruct.CounterDirection = ATIM_COUNTING_UP;
    ATIM_InitStruct.CounterOPMode = ATIM_OP_MODE_REPEATITIVE;
    ATIM_InitStruct.OverFlowMask = DISABLE;
    ATIM_InitStruct.Prescaler = ATIM_Prescaler_DIV1;      // 计算时钟 1MHz
    ATIM_InitStruct.ReloadValue = TS;        // 20K
    ATIM_InitStruct.RepetitionCounter = 0;
    ATIM_InitStruct.UnderFlowMask = DISABLE;
    ATIM_Init(&ATIM_InitStruct);
```

```

ATIM_OCInitStruct.BufferState = DISABLE;
ATIM_OCInitStruct.OCDMAState = DISABLE;
ATIM_OCInitStruct.OCInterruptSelect = ATIM_OC_IT_UP_COUNTER;
ATIM_OCInitStruct.OCInterruptState = ENABLE;
ATIM_OCInitStruct.OCMode = ATIM_OCMODE_PWM1;
ATIM_OCInitStruct.OCPolarity = ATIM_OCPOLARITY_NONINVERT;
ATIM_OC1AInit(&ATIM_OCInitStruct);
    ATIM_OC2AInit(&ATIM_OCInitStruct);
    ATIM_OC3AInit(&ATIM_OCInitStruct);

ATIM_SetCompare1A(0);
ATIM_SetCompare2A(0);
ATIM_SetCompare3A(0);
ATIM_PWMOutputConfig(OCREFA_TYPE_SINGLE, OUTPUT_TYPE_COMP, 0);
ATIM_CtrlPWMOutputs(ENABLE);
ATIM_Cmd(ENABLE);
}

```

```

void ATIM_IRQHandler(void)
{
    if(ATIM_GetITStatus(ATIM_IT_OVF))
    {
        ATIM_ClearITPendingBit(ATIM_IT_OVF);
    }
}

```

如图 4.8.3.3 是 120° 霍尔换相表，以霍尔信号从高到低是 ABC。那么第一个是 101=5 那么依次下去是 5-4-6-2-3-1。

而我们的电子电流 ABC 相就一次是 0-5 依次顺序排列。

STEP\_TAB[6]={5,3,4,1,0,2};这个换相表是 霍尔信号的 5 对应电子电流的 0 (A 高 B 低) , 所以 STEP\_TAB[6]的第 5 位放的是 0=AB, 依次 1 的位放的是 1 (AC) ,这样就会得到我们的 STEP\_TAB[6]={5,3,4,1,0,2}; 而我们的电子电流信号是一一对应的 0:AB; 1:AC; 2:BC; 3:BA; 4:CA; 5:CB

```

//输出上桥
if(step==0||step==1){
CW_ATIM->CH1CCRA=OutPwmValue;CW_ATIM->CH2CCRA=0;CW_ATIM->CH3CCRA=0;
} //0:AB; 1:AC
if(step==2||step==3){
CW_ATIM->CH1CCRA=0;CW_ATIM->CH2CCRA=OutPwmValue;CW_ATIM->CH3CCRA=0;
} //2:BC; 3:BA
if(step==4||step==5){
CW_ATIM->CH1CCRA=0;CW_ATIM->CH2CCRA=0;CW_ATIM->CH3CCRA=OutPwmValue;
} //4:CA; 5:CB

```

```

//输出下桥
if(step==0||step==5){PWM_AL_OFF; PWM_CL_OFF;PWM_BL_ON;} //AB CB ; B
下桥导通
else if(step==1||step==2){    PWM_AL_OFF;    PWM_BL_OFF;
PWM_CL_ON;}//AC BC; C 下桥导通
else if(step==3||step==4){    PWM_BL_OFF;    PWM_CL_OFF;
PWM_AL_ON;}//BA CA; A 下桥导通

```

我们读取霍尔信号的时候一点检查 IOI 口对应关系 是写的 A 是高还是低,对应我们读取的霍尔表示 A 高还是低, 要一一对应

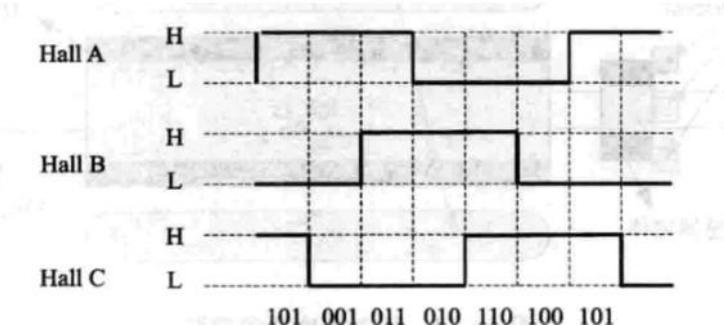
```

if(PA15_GETVALUE()!=0)x=1;
if(PB03_GETVALUE()!=0)x|=0x2;
if(PA02_GETVALUE()!=0)x|=0x4;

```

这里就写出了 PA15 是最低位, 如果 PA15 最低位那么霍尔换相表数字是--- 5-1-3-2-6-4。

### 三路霍尔信号图



120°Hall换相真值表

Hall 编码	正 转		反 转	
	绕组通电顺序	导通功率开关	输出相电压	导通功率开关
5	+A, -B	T1, T4	+B, -A	T3, T2
4	+A, -C	T1, T6	+C, -A	T5, T2
6	+B, -C	T3, T6	+C, -B	T5, T4
2	+B, -A	T3, T2	+A, -B	T1, T4
3	+C, -A	T5, T2	+A, -C	T1, T6
1	+C, -B	T5, T4	+B, -C	T3, T6

图 4.8.3.3

### 4.8.4 下载与验证

下载完成之后按下复位按键, 转动我们的电位器, 电机开始转动, 上电, PC13 指示灯常亮。旋转电位器 电机运转后 PC13 灯闪烁。

如果有示波器可以查看 UVW 任意一个的波形是梯形如图:

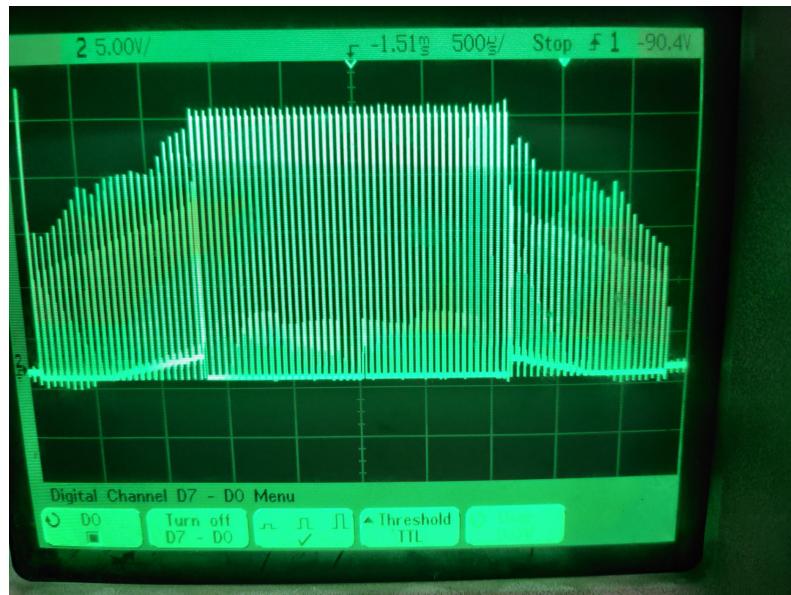


图 4.8.4.1

## 4.9 实例九 霍尔与 PID 电机实例

视频连接: <https://space.bilibili.com/1407060490/channel/seriesdetail?sid=2323858>

文件夹路径: CW32\_BLDC\_EVA 电机板资料 V4\BLCD 电机板例程-MDK\12 CW32 BLDC haLL pid OK 实例

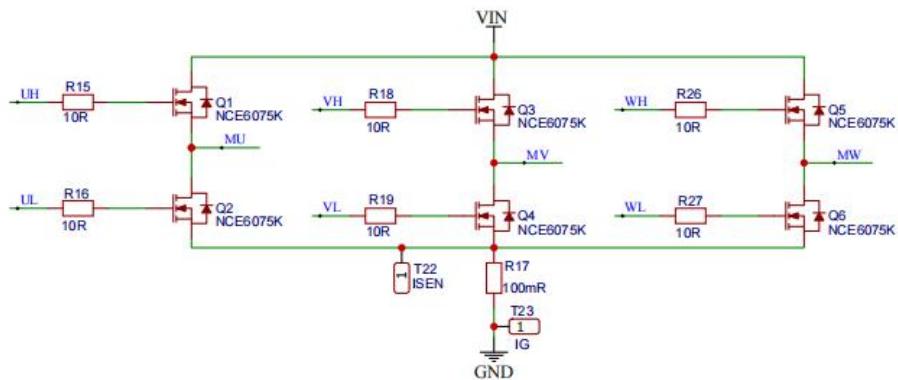
### 4.9.1 功能要求

- ①熟悉电机工作原理。
- ②PID 简单的算法。

### 4.9.2 硬件设计

霍尔电路图与电机驱动电路图如下:

#### 电桥



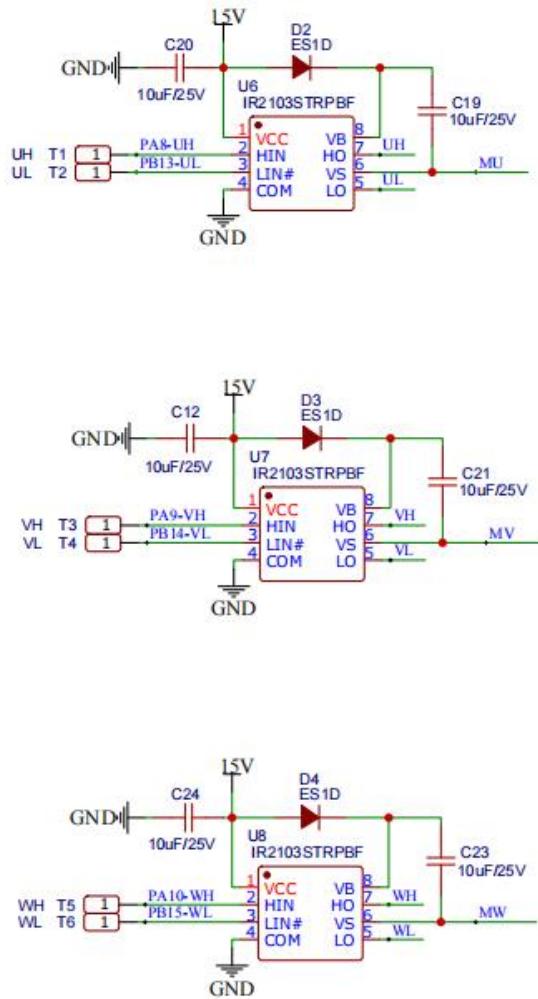


图 4.9.2

由图可知我们电机接口：

PA8--UH; PB13--UL;PA9--VH;PB14--VL;PA10--WH;PB15--WL;

霍尔接口： HA--PA15;HB--PB3;HC--PA2;

### 4.9.3 软件设计

打开 CW32\_BLDC\_EVA 电机板资料 V4\BLCD 电机板例程-MDK\12 CW32 BLDC haLL pid OK 实例。

上一节我们就说了怎么运用霍尔的采集去换相使电机运转起来，下面我们就针对 PID 运算的讲解。

PID 算法：

PID 即：Proportional（比例）、Integral（积分）、Differential（微分）的缩写。顾名思义，PID 控制算法是结合比例、积分和微分三种环节于一体的控制算法，它是连续系统中技术最为成熟、应用最为广泛的一种控制算法

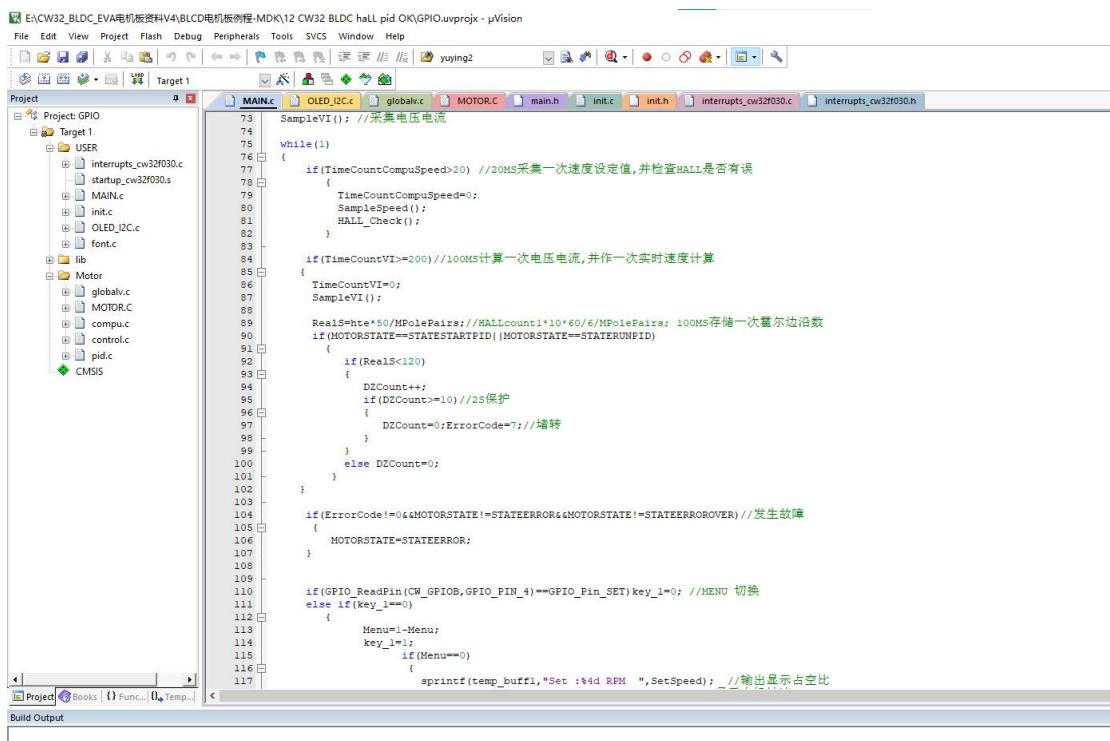


图 4.9.3.1

PID 初始化代码：

```
void PID_init(void)
{
    SError=0;SSumError=0;SLastError=0;
```

```
SPEED_P=(float)SP/1000;
SPEED_I=(float)SI/1000;
SPEED_D=(float)SD/1000;
```

```
SSumError=OutPwmValue;
SSumError=SSumError/SPEED_I;
}
```

```
void PIDcompute(unsigned int Target,unsigned int Real)
```

```
{
    float j=0.0,i;
```

```
if(Target*M PolePairs<600)Target=600/M PolePairs; //限速， 4 对极下 150RPM
```

```
SError =(Target-Real); //本次偏差
```

```

        if(SSumError<500)SSumError=500;
        else if(SSumError*SPEED_I>PWM_PERIOD)
        {
            SSumError=PWM_PERIOD;
            SSumError=SSumError/SPEED_I;
        } // 输出到最大值后，不再累计偏差，抗积分饱和
        else SSumError +=SError;

SdError=SError-SLastError; //微分项，偏差的变化
SError=SLastError; //记录本次偏差，用于下次计算

i=SPEED_P;
j=SError*i;
i=SPEED_I;
j=j+SSumError*i;
i=SPEED_D;
j=j+SdError*i;

if(j>PWM_PERIOD)OutPwmValue=PWM_PERIOD;
else if(j<1)OutPwmValue=1;
else OutPwmValue=j;

UPPWM();
}

```

#### 4.9.4 下载与验证

2页菜单：1：设定速度、实时速度； 2：母线电压、电流显示。

KEY1：按键切换速度及电压电流菜单

KEY2：方向切换按键。电机运行时也可以切换。LED2 亮灭表示不同的方向。

KEY3：电机启停控制，LED3 灯亮时，电机处于启动状态； LED3 灯来时，电机处于停止状态；

电位器调速，切换到速度菜单，可以显示当前设定转速。

如果有示波器可以查看 UVW 任意一个的波形是梯形如图：

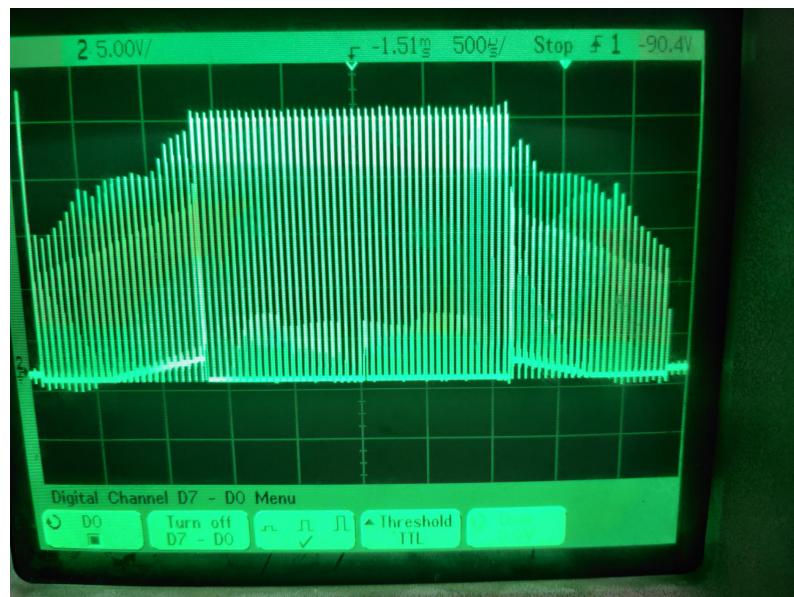


图 4.9.4.1

## 4.10 实例十 无霍尔开环电机实例

视频连接: <https://space.bilibili.com/1407060490/channel/seriesdetail?sid=2323858>

文件夹路径: CW32\_BLDC\_EVA 电机板资料 V4\LCD 电机板例程-MDK\13 CW32 BLDC sensorless openspeed 105V OK 实例

### 4.10.1 功能要求

- ①熟悉电机工作原理。
- ②了解无霍尔的“三段式”启动。

### 4.10.2 硬件设计

电机接口电路图如下:

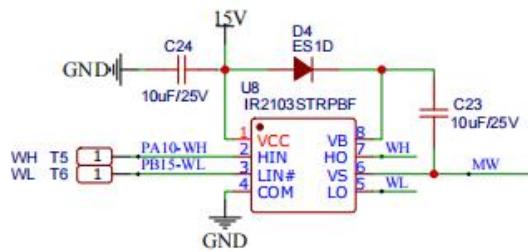
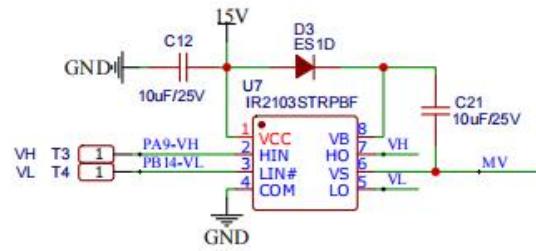
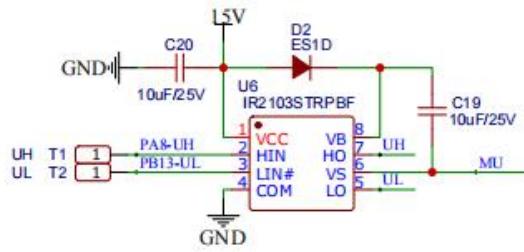


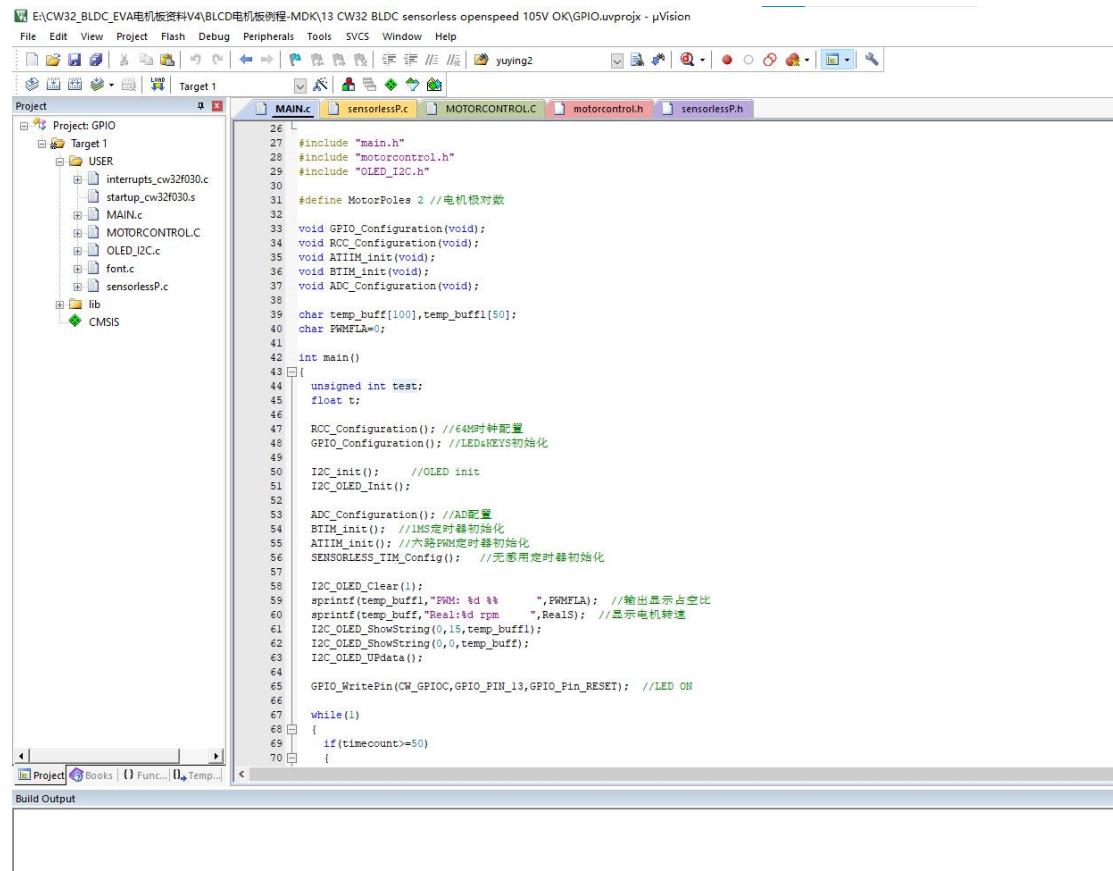
图 4.10.2

由图可知我们电机驱动接口：  
PA8--UH; PB13--UL;PA9--VH;PB14--VL;PA10--WH;PB15--WL;

### 4.10.3 软件设计

打开 CW32\_BLDC\_EVA 电机板资料 V4\BLCD 电机板例程 -MDK\13 CW32 BLDC sensorless openspeed 105V OK 实例。

这是针对于无霍尔开环的电机如何让它转动。



```

E:\CW32_BLDC_EVA电机板资料V4\BLCD电机板例程-MDK\13 CW32 BLDC sensorless openspeed 105V OK\GPIO.uvprojx - μVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Project Target 1
Project: GPIO
Target 1
USER
  interrupts_cw32f030.c
  startup_cw32f030.s
  MAIN.c
  MOTORCONTROL.C
  OLED_I2C.c
  font.c
  sensorlessP.c
lib
CMSIS
MAIN.c
sensorlessP.c
MOTORCONTROL.C
motorcontrol.h
sensorlessP.h
26 L
27 #include "main.h"
28 #include "motorcontrol.h"
29 #include "OLED_I2C.h"
30
31 #define MotorPoles 2 //电机板对数
32
33 void GPIO_Configuration(void);
34 void RCC_Configuration(void);
35 void ATIM_init(void);
36 void BTIM_init(void);
37 void ADC_Configuration(void);
38
39 char temp_buff[100],temp_buff1[50];
40 char PWMFLA=0;
41
42 int main()
43 {
44     unsigned int test;
45     float t;
46
47     RCC_Configuration(); //64MHz时钟配置
48     GPIO_Configuration(); //LED+KEYS初始化
49
50     I2C_init(); //OLED init
51     I2C_OLED_Init();
52
53     ADC_Configuration(); //AD配置
54     BTIM_init(); //IMS定时器初始化
55     ATIM_init(); //六路FRM定时器初始化
56     SENSORLESS_TIM_Config(); //无感用定时器初始化
57
58     I2C_OLED_Clear();
59     sprintf(temp_buff1,"PWM: %d %%      ",PWMFLA); //输出显示占空比
60     sprintf(temp_buff,"%d rpm      ",RealS); //显示电机转速
61     I2C_OLED_ShowString(0,15,temp_buff1);
62     I2C_OLED_ShowString(0,0,temp_buff);
63     I2C_OLED_UPdata();
64
65     GPIO_WritePin(CW_GPIOC,GPIO_PIN_13,GPIO_Pin_RESET); //LED ON
66
67     while(1)
68     {
69         if(timecount>=50)
70         {

```

图 4.10.3.1

While 初始化代码：

```

while(1)
{
    if(timecount>=50)
    {
        timecount=0;//100ms

```

//电位器采集的 AD 代码值在 3-4000 范围内有效 3 对应最大输出，4000 对应

最小输出

```
if(SampleData[5]>=4000)    test=0;
else if(SampleData[5]<3)    test=4000;
else      test=4000-SampleData[5];

t=test*0.8;//40*32;
//t=t/4000;
//t=t*TSPeriod;
test=t; //计算当前 应当输出的值

if(OutPwm<test)
{
    OutPwm+=TSPeriod*0.05; //缓启动
    if(OutPwm>=test)OutPwm=test;
}
else OutPwm=test;

if(Motor_Start_F==1)//电机启动后，及时更新 PWM
UPPWM();// 及时更新输出占空比

if(test<OUTMINPWM) //小于最小输出时，电机停止运行
{
    if(Motor_Start_F==1)
    {
        Motor_Start_F=0; //电机停止状态
        MOTOR_STOP(); //停止电机
        GPIO_WritePin(CW_GPIOC,GPIO_PIN_13,GPIO_Pin_RESET);
    }
    else if(test>OUTMINPWM+50&&Motor_Start_F==0&&ErrorCode==0) //大于最小输出时，开始转
    {
        Motor_Start_F=1; //改变启停状态
        Sensorless_MOTOR_START(); //启动电机
    }
}

if(timecount1>=500) //1 秒计算一次速度      MotorPoles-- 极对数
HALLcount--脉冲个数  RealS: 实测转速
{
    timecount1=0;
    //1 秒计算速度值并显示
```

```

RealS=HALLcount*20/MotorPoles; //计算实测电机 RPM
HALLcount=0;
sprintf(temp_buff,"Real:%d rpm ",RealS); //显示电机转速
I2C_OLED_ShowString(0,0,temp_buff);
PWMFLA=OutPwm/32;
sprintf(temp_buff1,"PWM: %d %% ",PWMFLA);

I2C_OLED_ShowString(0,15,temp_buff1); //显示输出占空比
I2C_OLED_UPdata();
I2C_OLED_UPdata();
}

} // while(1)
}

```

#### 4.10.4 下载与验证

上电，PC13 指示灯常亮。旋转电位器 电机运转后 PC13 灯闪烁。

如果有示波器可以查看 UVW 任意一个的波形是梯形如图：

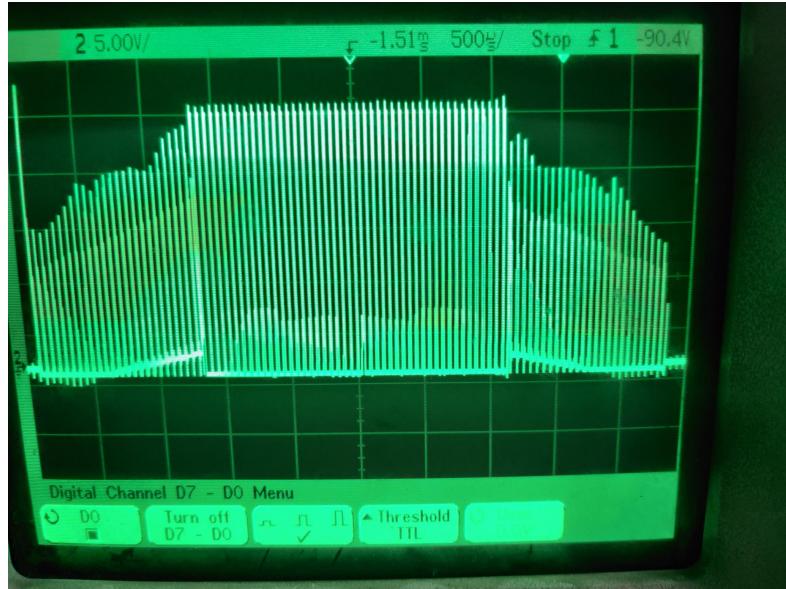


图 4.10.4.1

## 4.11 实例十一 无霍尔闭环电机实例

视频连接: <https://space.bilibili.com/1407060490/channel/seriesdetail?sid=2323858>

文件夹路径: CW32\_BLDC\_EVA 电机板资料 V4\LCD 电机板例程-MDK\14 CW32 BLDC sensorless closespeed 105V OK 实例

### 4.11.1 功能要求

- ①熟悉电机工作原理。
- ②了解无霍尔的“三段式”启动。

### 4.11.2 硬件设计

电机接口电路图如下:

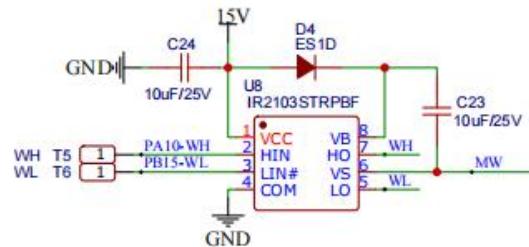
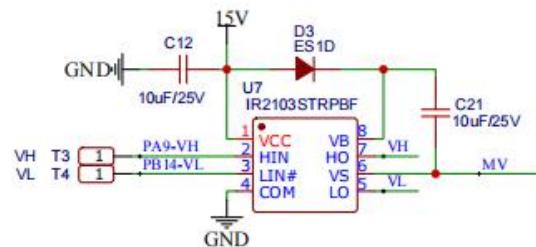
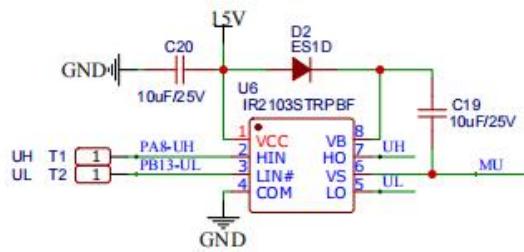


图 4.11.2

由图可知我们电机驱动接口:

PA8--UH; PB13--UL;PA9--VH;PB14--VL;PA10--WH;PB15--WL;

### 4.11.3 软件设计

打开 CW32\_BLDC\_EVA 电机板资料 V4\BLCD 电机板例程 -MDK\14 CW32 BLDC sensorless closespeed 105V OK GPIO.uvprojx - μVision

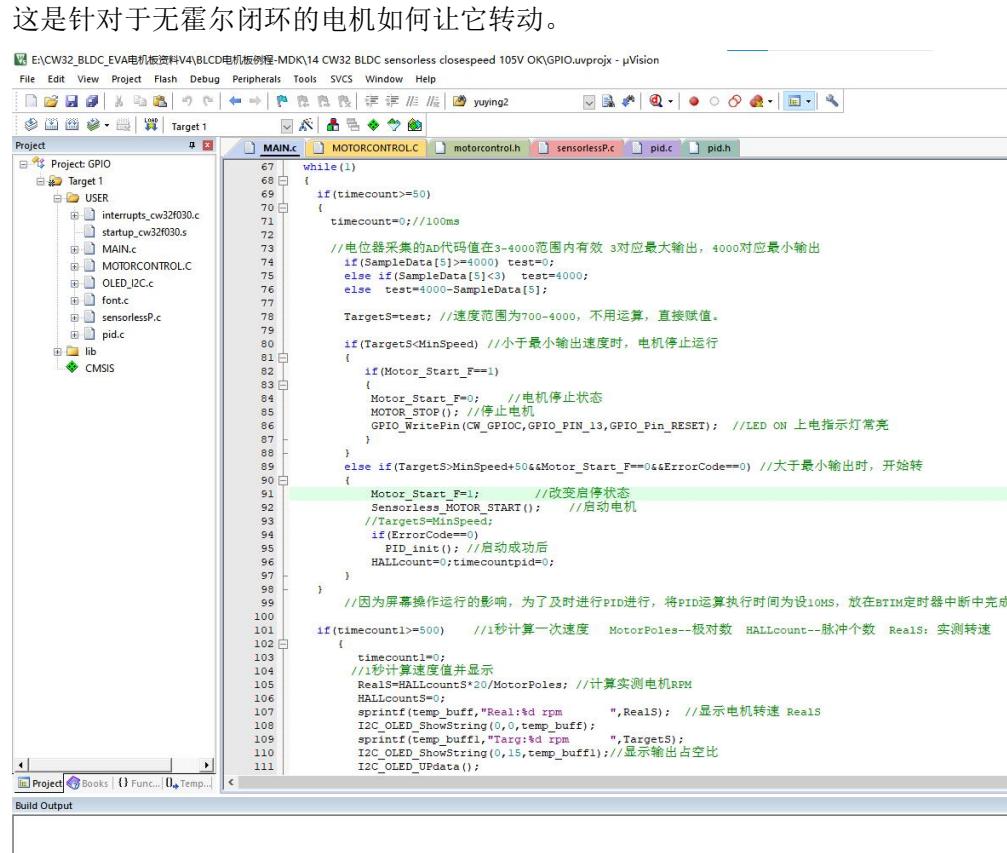


图 4.11.3.1

While 初始化代码：

```
while(1)
{
    if(timecount>=50)
    {
        timecount=0;//100ms
```

//电位器采集的 AD 代码值在 3-4000 范围内有效 3 对应最大输出，4000 对应最小输出

```
if(SampleData[5]>=4000)    test=0;
else if(SampleData[5]<3)    test=4000;
else      test=4000-SampleData[5];
```

TargetS=test; //速度范围为 700-4000，不用运算，直接赋值。

```

if(TargetS<MinSpeed) //小于最小输出速度时，电机停止运行
{
    if(Motor_Start_F==1)
    {
        Motor_Start_F=0;      //电机停止状态
        MOTOR_STOP(); //停止电机
        GPIO_WritePin(CW_GPIOC,GPIO_PIN_13,GPIO_Pin_RESET);

//LED ON 上电指示灯常亮
    }
}
else if(TargetS>MinSpeed+50&&Motor_Start_F==0&&ErrorCode==0) // 大于最小输出时，开始转
{
    Motor_Start_F=1;          //改变启停状态
    Sensorless_MOTOR_START(); //启动电机
    //TargetS=MinSpeed;
    if(ErrorCode==0)
        PID_init(); //启动成功后
    HALLcount=0;timecountpid=0;
}
}

//因为屏幕操作运行的影响，为了及时进行 PID 进行，将 PID 运算执行时间为设 10MS，放在 BTIM 定时器中断中完成。

if(timecount1>=500)      //1 秒计算一次速度   MotorPoles--极对数   HALLcount--
脉冲个数  RealS: 实测转速
{
    timecount1=0;
    //1 秒计算速度值并显示
    RealS=HALLcountS*20/MotorPoles; //计算实测电机 RPM
    HALLcountS=0;
    sprintf(temp_buff,"Real:%d rpm      ",RealS); // 显示电机转速
RealS
    I2C_OLED_ShowString(0,0,temp_buff);

    sprintf(temp_buff1,"Targ:%d rpm      ",TargetS);
    I2C_OLED_ShowString(0,15,temp_buff1); //显示输出占空比
    I2C_OLED_UPdata();
}
}

} // while(1)

```

#### 4.11.4 下载与验证

上电，PC13 指示灯常亮。旋转电位器 电机运转后 PC13 灯闪烁

如果有示波器可以查看 UVW 任意一个的波形是梯形如图：

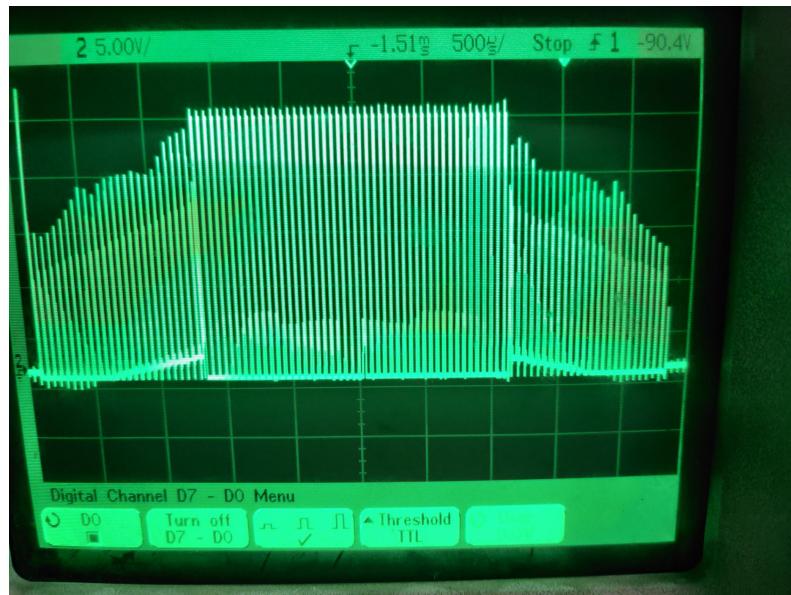


图 4.11.4.1