机器人与嵌入式技术实训教学平台实验教程

目录

1.机器人实训教学平台

1.1 概述

机器人是自动执行工作的机器装置。它既可以接受人类指挥，又可以运行预先编排的程序，也可以根据以人工智能技术制定的原则纲领行动。随着科学技术的不断发展，机器人技术也变得越来越成熟了。未来，机器人将成为我们生产、生活中不可或缺的伙伴。政策的支持和人口红利的持续消退，给机器人产业带来了发展机会。2012年，工信部就发布了《智能制造装备产业“十二五”发展规划》和《服务机器人科技发展“十二五”专向规划》。机器人作为智能制造装备和社会服务的重要组成部分，迎来其战略性的发展机会。2015年，国务院印发“中国制造2025”规划，部署全面推进实施制造强国战略。这是我国实施制造强国战略的行动纲领，明确了9项战略任务和重点领域，其中，包括机器人行业这一重点领域。业内人士认为，未来机器人产业将获得更多政策支持，机器人产业和教育迎来了千载难逢的发展契机。

在大力推广机器人产业化和提高当代大学生创新实践能力的今天，国家已经意识到了机器人教育的重要性，相关部门已经将其纳入到了大学生工程教育实践的教学内容中。在这种背景下，广大学生和教师渴求能有一种可以支持和激发学生创造性思维、寓教于乐、系统化的机器人实训教学平台。鉴于此，开发一款基础性强、覆盖面全、科技含量高、紧跟当代科技潮流的实训教学平台，该实训教学平台包含了三大模块：基于STM32的机器人传感系统与机器人电机控制系统、基于树莓派的嵌入式图像处理系统，涉及了C语言编程、嵌入式系统、LINUX、图像处理、传感器检测以及电机控制技术。用户也可以根据自己所需的应用，在该平台进行自由开发。通过实践验证，通过机器人实训教学平台可以在短时间内对高校内大学生工程教育实践和创新基地的学生进行系统培训，使他们快速掌握移动机器人设计与研发所需的基本技术，不仅可以为大学生参加机器人相关的竞赛做好基础知识准备，同时还可促进大学生就业，并提升大学生就业质量。

1.2 特点

机器人实训教学平台主要以机器人的传感技术、电机运动控制技术、图像处理技术开发为主展开实训教学，具有独立自主的知识产权。与国内外同类产品相比，具有的主要特点如下：

（1）硬件资源丰富，涉及STM32F103ZE单片机、智能外控LED、陀螺仪、加速度计、数字罗盘、语音识别、MP3播放、显示屏、触摸按键、温湿度传感器、蜂鸣器、红外避障传感器、WIFI、串口、红外测距传感器、超声波传感器、直流减速电机、步进电机、无刷电机、数字舵机、机械手臂、树莓派等机器人研发所需的各个主要部件。

（2）低成本：价格适中，可以大大降低大中专院校实验室的建设成本，而且本实训平台是以实验箱的形式供学生学习实践，在教学的过程中，有利于减少不必要的硬件损耗，使机器人技术教学得到更广泛的普及。

（3）多学科知识涉及广泛：不仅可以利用该平台系统地进行智能机器人学科的实验教学，也可以在该平台上开展 C 语言编程、传感器、自动控制、图像处理、嵌入式、无线通信、物联网等学科的实验学习。

（4）平台开放：所有硬件和软件技术资料公开，可供用户进行一系列的综合实验开发。

（5）功能丰富：配套相应的实验教学手册，可开展基于 STM32 的机器人传感实验与机器人电机控制实验、基于树莓派的图像处理与控制实验，累计 24 个实验。后续还将以此为基础更新更多的机器人控制综合实验。

1.3 硬件配置

RobotTTP 主要的硬件配置主要包括以下四部分：机器人传感系统、嵌入式图像处理系统、机器人电机控制系统、机械手臂，具体如下：

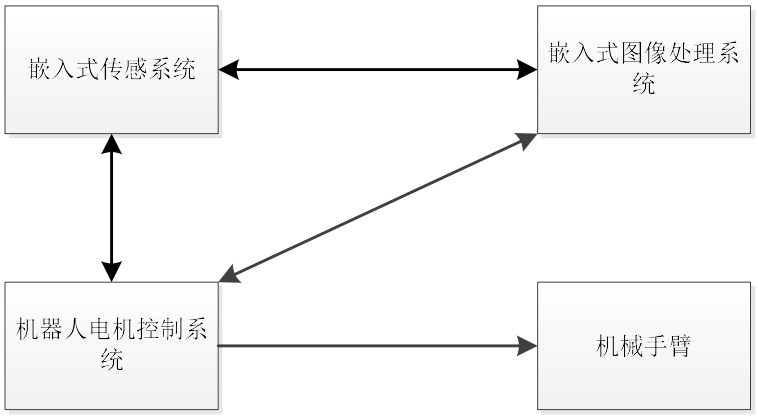


图 1-1 RobotTTP 的硬件配置

（1） 基于 STM32 的机器人传感系统

机器人传感系统采用 STM32 微控制器（具体型号为 STM32F103ZET6）作为主控单元。无论是从性价比还是上手的难易程度考虑，STM32 微控制器目前在嵌入式设计与开发行业里具有很大的优势，也是大学生单片机与嵌入式教学的首选。所选用的 STM32 微控制器的主要特性如下：

1) 内核：基于 ARM 32 位的 Cortex-M3，最高 72MHz 工作频率。
2) 存储器：内嵌 64kB 的 SRAM、512kB 片内 Flash。
3) 外设控制模块：包括 DMA、ADC、UART、SSP/SPI、I2C、定时器、CAN、USB、112 个通用 I/O 等。
4) 低功耗：具有睡眠、停机和待机等模式。
5) 电源：2.0-3.6V 供电。
6) 调试模式：串行单线调试（SWD)和 JTAG 接口。

7) 封装：144 引脚 LQFP 封装。

实验平台上装配了机器人常用的传感器与执行器，主要有：

1) 陀螺仪
2) 加速度计
3) 摄像头
4) 语音识别传感器
5) 触摸按键
6) 温湿度传感器
7) 红外避障传感器
8) 红外测距传感器
9) 超声波传感器
10)LED 灯
11)蜂鸣器
12)智能外控 LED
13)显示屏
14)MP3 播放器
15)WIFI 通信

（2） 机器人电机控制系统

主要配置了以下几种电机：

1) 直流减速电机
2) 直流无刷电机
3) 数字舵机
4) 步进电机
5) 机械手臂

（3） 嵌入式图像处理系统

由树莓派及高清摄像头组成。

2.基于 STM32 的机器人传感实验

STM32 是 ST 公司推出的 32 位基于 Contex-M3 内核的 MCU（微控制器），凭借其品种的多样化、极高的性价比、简单易用的开发方式、应用领域的广泛性迅速占领了中低端 MCU 市场，已经成为嵌入式工程师、大学生嵌入式入门学习的首选。STM32 倡导的是基于固件库的开发方式，和传统基于 8/16 位单片机相比，这是一种全新的开发方式。这种开发方式可以让初学者快速上手，通过调用库里的 API（应用程序接口）就可以根据应用需求快速搭建一个大型的程序，大大降低了学习的门槛和开发周期。在利用 STM32 库函数进行程序开发的过程中，学习者可以从上层的 API 层层追踪到底层，了解 STM32 所有外设寄存器及其内存的分布，对其硬件操作能有深入了解。与此同时，STM32 库函数涉及了大量的 C 语言知识，比如关键字、结构体、指针、类型转换、内联函数、条件编译、宏等，相信通过库的使用和学习，学习者 C 语言的技能将会大大提升。另外，STM32 库函数开发基本上都是兼容的，还可以迁移到 STM32 其它系列微控制器的学习，比如 STM23F1、STM32F3、STM32F4/F7 等。

在本实验教程中，STM32 传感教学实验主要围绕 RobotTTP 自身 STM32 周围的硬件资源进行展开，主要包括以下十五部分内容： MDK（Microcontroller Development Kit，微控制器开发工具包）工程建立、蜂鸣器、触摸按键、温湿度传感器、红外避障传感器、红外测距传感器、超声波传感器、WIFI 通信、陀螺仪、三轴加速度计、数字罗盘、语言识别与 MP3 播放、摄像头、智能外控 LED 和显示屏。每个部分除了原理分析，代码讲解外，都结合 RobotTTP 自身的一些功能实现安排了相应的实验，以此使初学者可以循序渐进地掌握 STM32 以及其在机器人上的使用。

本实验教程对初学者的要求是应具有基本的单片机和 C 语言基础，曾使在诸如 51、AVR 上编写过简单的程序。学习者在学习 STM32 的时候，无需太担心自己基础，参考相关的书籍（在这里，推荐刘火良编著的《STM32 库开发实践指南》），只要有学习这门技术的决心和持之以恒的耐心，相信初学者一定能很快入门。

2.1 工程建立及调试

实验目的

了解 MDK 集成开发环境及安装。

学习掌握 STM32 工程建立方法及其库文件的组成。

学习掌握基于 ST-LINK STM32 程序下载与调试。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件

实验预习

MDK 介绍：

MDK，也称 MDK-ARM，Realview MDK、I-MDK、Keil uVision4 等。MDK-ARM 软件为基于 Cortex-M、Cortex-R4、ARM7、ARM9 处理器设备提供了一个完整的开发环境。 MDK-ARM 专为微控制器应用而设计，不仅易学易用，而且功能强大，能够满足大多数苛刻的嵌入式应用。详细资料参考本目录下的“MDK 介绍.pdf”。

ST-LINK V2 介绍：

ST-LINK V2 是 STM32 微控制器系列的在线调试器和编程器。它的单线接口模块（SWIM）和串行线调试（SWD）接口用于与应用板上的 STM32 微控制器进行通讯，而它的 USB 全速接口与 Atollic、IAR、Keil、TASKING 等集成开发环境通讯。详细资料参考本目录下的“ST-LINK V2 使用说明.pdf”。

STM32 库介绍：

STM32 库是由 ST 公司针对 STM32 提供的函数接口，即 API (Application Program Interface)，开发者可调用这些函数接口来配置 STM32 的寄存器，使开发人员得以脱离最底层的寄存器操作，有开发快速，易于阅读，维护成本低等优点。详细资料参考本目录下的“STM32 库函数架构剖析.pdf”。

实验内容

[1] 安装 Keil。

[2] 启动 Keil。

[3] 新建一个 Keil 工程。

[4] 打开一个 Keil 工程。

[5] 安装 ST-LINK V2 调试器的驱动。

[6] 连接 ST-LINK 与移动微机器人平台。

[7] 设置基于 ST-LINK 的 Keil 仿真选项。

[8] 安装 STM32 ST-LINK Utility 软件。

[9] 使用 STM32 ST-LINK Utility 软件下载程序。

[10] 使用 ST-LINK V2 调试器，进行程序的调试。

实验步骤

[1] 安装 Keil。

[2] 启动 keil。

双击桌面上 Keil uVision4.exe 图标即可。

[3] 新建一个 Keil 工程。

第一步：新建一个文件夹取名 LedShanShuo，在 LedShanShuo 文件夹里建立库函数的工程。在 LedShanShuo 文件下建立四个文件夹 lib、output、startup、user，如图 2-1-4 所示，其中 lib 文件夹存放的是库函数文件，output 存放的是编译时产生的中间文件和输出的 hex 文件，startup 文件存放的是启动文件和内核文件，user 存放的是用户需要编写的文件。将对应的文件放入相应的文件夹，具体流程参考本文件目录下“Keil 工程建立.pdf”。

第二步：新建 Keil 工程，点击 Project->New uVision Project，取名 LedShanShuo 并保存在 LedShanShuo 文件夹里，并选择芯片型号为 STM32F103ZE。点击 Project->Manage->Components,Environment and Books，为项目设置 3 个组分别是 USER、LIB、STARTUP，并分别为各组添加文件，依次如图 2-1-1、2-1-2、2-1-3 所示。

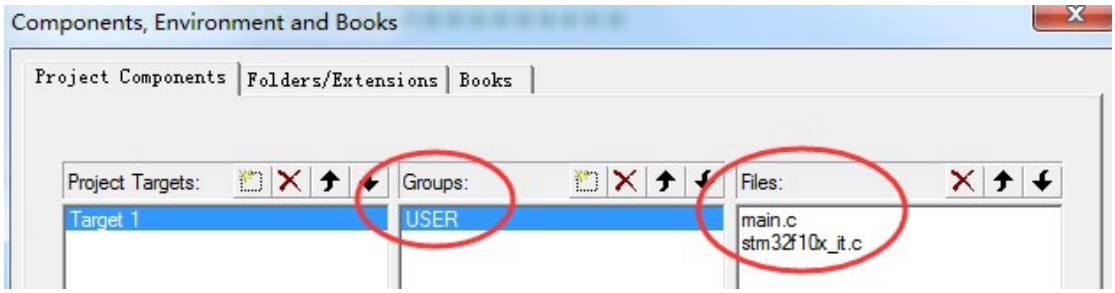


图 2-1-1 USER 组及该组包含的文件

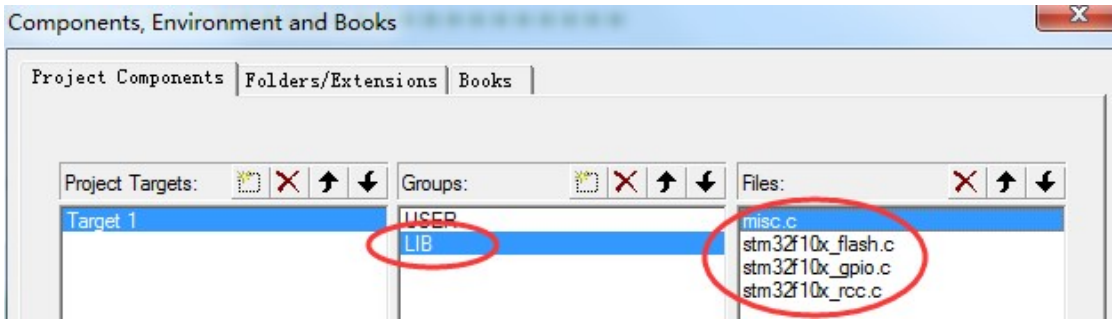


图 2-1-2 LIB 组及该组包含的文件

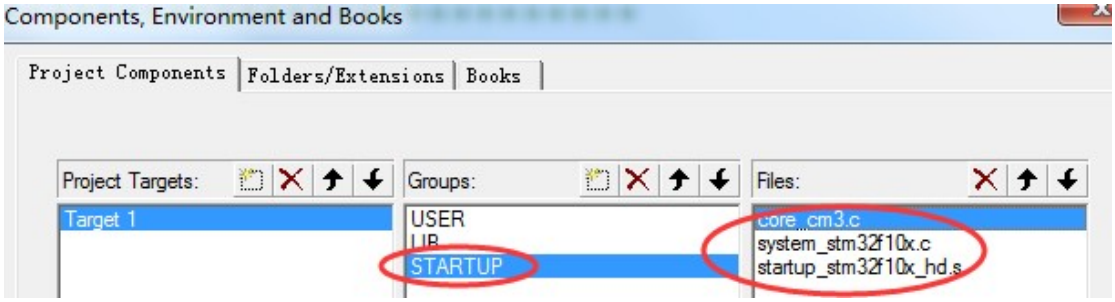


图 2-1-3 STARTUP 组及该组包含的文件

设置成功之后，可以在工程界面左侧栏目中显示出这 3 个文件夹及其文件。如图 2-1-4 所示。具体流程参考本文件目录下“Keil 工程建立.pdf”。

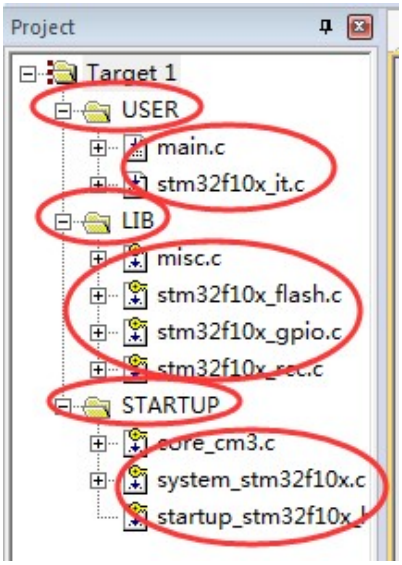


图 2-1-4 组别及其文件显示

第三步：设置编译器，点击 Project->Options for Target ‘Target 1’...，界面如图 2-1-5 所示。具体流程参考本文件目录下“Keil 工程建立.pdf”。

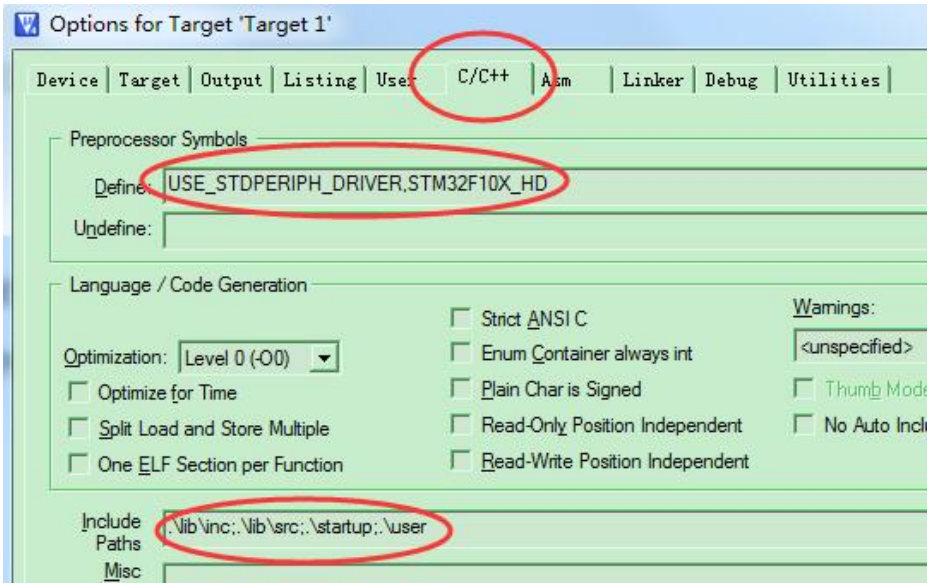


图 2-1-5 设置编译器

第四步：设置输出文件目录，如图 2-1-6，具体流程参考本文件目录下“Keil 工程建立.pdf”。

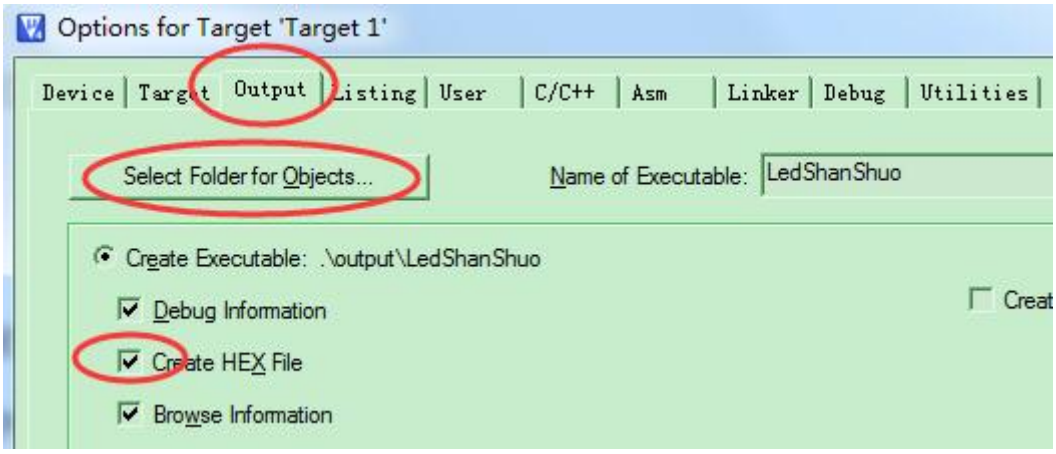


图 2-1-6 选择输出路径

第五步：编译程序，点击 Project->Rebuild all target files。

[4] 打开一个 keil 工程。

点击 Project->Open Project，找到工程文件“LedShanShuo.uvproj”，点击打开。

[5] 安装 ST-LINK V2 调试器驱动。

双击本目录下的“ST-Link_V2_USBdriver.exe”进行安装，具体参考本目录下“ST-LINK V2 使用说明.pdf”。

[6] 连接 ST-LINK 与机器人嵌入式实训教学实验平台。

将 ST-LINK 的插头连接到平台的插座上，如图 2-1-7 所示。

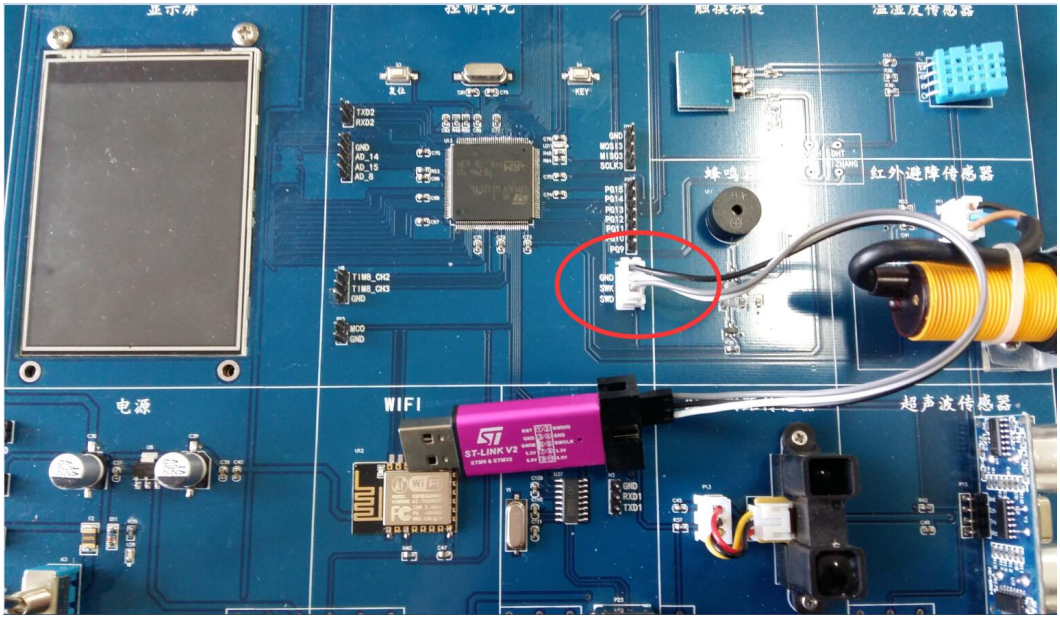


图 2-1-7 ST-LINK 与机器人嵌入式实训教学实验平台连接图

[7] 设置基于 ST-LINK 的 Keil 仿真选项。

启动 Keil，点击 Project->Options for Target ‘Target 1’...，选择 Debug 选项卡，配置如下图 2-1-8，具体参考本目录下“ST-LINK V2 使用说明.pdf”。

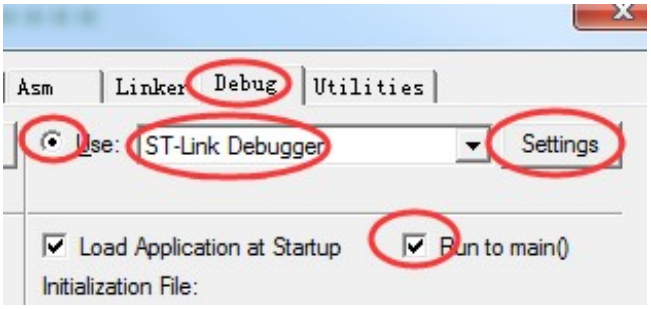


图 2-1-8 设置基于 ST-LINK 的仿真选项

[8] 安装 STM32 ST-LINK Utility 软件。

双击本目录下“STM32 ST-LINK Utility_v3.0.0.exe”进行安装。

[9] 使用 STM32 ST-LINK Utility 软件下载程序。

双击桌面“STM32 ST-LINK Utility.exe”图标，点击 File->Open file...进行程序的选择，点击 Target->Program & Verify...，进行程序的烧写。界面如下图 2-1-9 所示，具体参考本目录下“ST-LINK V2 使用说明.pdf”。

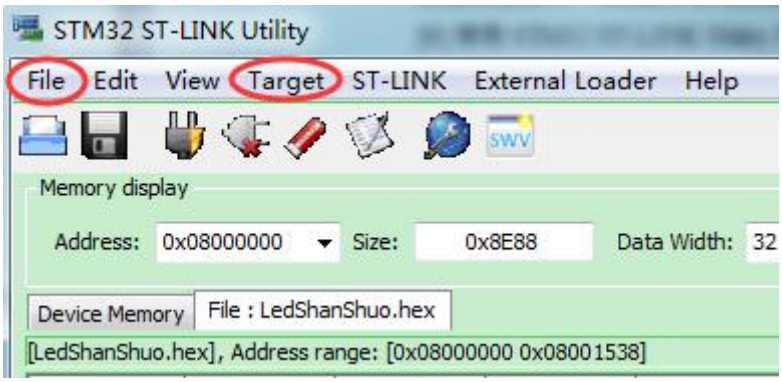


图 2-1-9 下载软件界面

[10] 使用 ST-LINK V2 调试器，进行程序的调试。

打开工程文件“LedShanShuo.uvproj”，点击 Debug->Start/Stop Debug Session。仿真调试界面如图 2-1-10 所示，具体参照本目录下“ST-LINK V2 使用说明.pdf”。

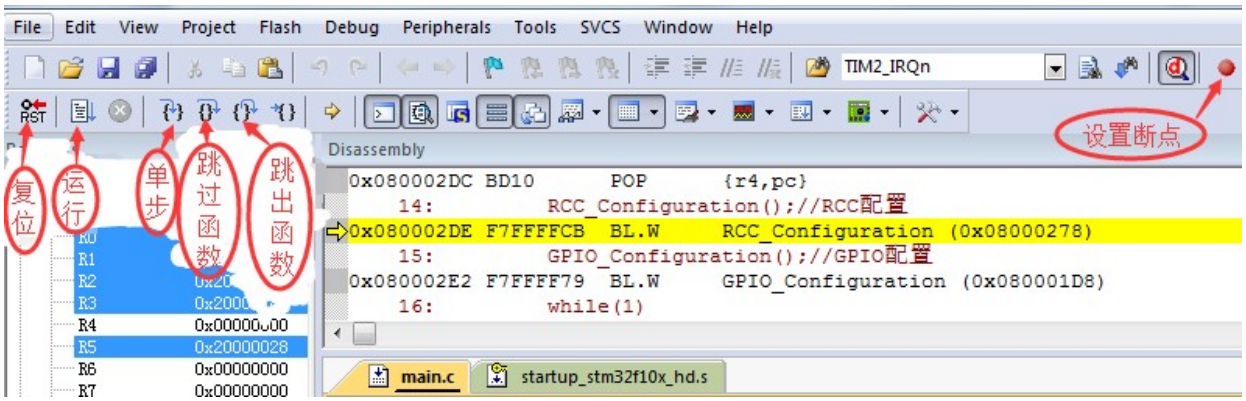


图 2-1-10 仿真调试界面

参考程序

参考程序的工程名为“LedShanShuo.uvproj”，在本实验文件目录下“LedShanShuo”文件夹下。部分参考程序如下：

```
#include "define.h"
void delay(u32 i);//延时子程序
void RCC_Configuration(void);//RCC 子程序
void GPIO_FuWei(void);//GPIO 复位及 SW-DP 配置
void Led_init(void);//LED 初始化

int main(void)
{
	RCC_Configuration();//RCC 配置
	GPIO_FuWei();//GPIO 复位及 SW-DP 配置
	Led_init();//LED 初始化
	while(1)
	{
		Led(ON);//点亮 LED 灯
		delay(2000000);//延时
		Led(OFF);//关闭 LED 灯
		delay(2000000);//延时
	}
}
```

实验小结

通过本小节的学习，学会如何安装 Keil 软件，如何建立、打开、调试一个 Keil 工程，并学会如何使用 STM32 ST-LINK Utility 软件下载程序。

思考题

如何对编写的程序进行单步调试、变量值查看？

2.2 蜂鸣器

实验目的

了解 STM32 时钟源种类，学习掌握 STM32 的时钟配置。

了解 STM32 内部 GPIO（General Purpose Input/Output，通用输入/输出）的工作模式，学习掌握 STM32 GPIO 的控制。

了解 STM32 内部定时器的工作模式，学习掌握 STM32 内部定时器的使用。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件

实验预习

STM32 的时钟源种类：

有五个时钟源，分别为 HSI、HSE、LSI、LSE、PLL。HSI 是高速内部时钟，RC 振荡器，频率为 8MHz。 HSE 是高速外部时钟，可接石英/陶瓷谐振器，或者接外部时钟源，频率范围为 4MHz~16MHz。 LSI 是低速内部时钟，RC 振荡器，频率为 40kHz。LSE 是低速外部时钟，接频率为 32.768kHz 的石英晶体。 PLL 为锁相环倍频输出，其时钟输入源可选择为 HSI/2、HSE 或者 HSE/2。倍频可选择为 2~16 倍，但是其输出频率最大不得超过 72MHz。详细资料参照本文件目录下“STM32F103 中文资料.pdf”。

STM32 GPIO 口的工作模式：

主要有八种，分别是浮空输入、模拟输入、上拉输入、下拉输入、推挽输出、开漏输出、复用推挽输出、复用开漏输出。

1) 浮空输入主要应用于外部按键输入，因为浮空输入状态下，GPIO 口的电平状态是不确定的，完全由外部输入决定。

2) 模拟输入主要用于输入模拟电平，比如 ADC 转换。

3) 上拉输入主要为了保证无信号输入时输入端的电平默认为高电平。

4) 下拉输入主要为了保证无信号输入时输入端的电平默认为低电平。

5) 推挽输出既提高电路的负载能力，又提高开关速度。主要是因为推挽电路是两个参数相同的三极管或 MOSFET，以推挽方式存在于电路中，各负责正负半周的波形放大任务，电路工作时，两只对称的功率开关管每次只有一个导通，所以导通损耗小、效率高，输出既可以向负载灌电流，也可以从负载抽取电流。

6) 开漏输出是用来连接不同电平的器件，用于匹配电平。主要因为开漏输出高电平时必须外接上拉电阻，通过改变上拉电源的电压，便可以改变传输电平。

7) 复用推挽输出及复用开漏输出主要是 GPIO 口第二功能模块的输出，即单片机内部的功能模块映射到 IO 口上，与外部设备相连接。

详细资料参照本文件目录下“STM32F103 中文资料.pdf”。

STM32 单片机定时器介绍：

按功能分类，STM32 单片机含有 2 个高级控制定时器，TIM1、TIM8 分别可产生 6 个通道的 PWM（Pulse Width Modulation，脉宽幅值调制），进而可用于控制三相以下步进电机。

含有 4 个普通定时器 TIM2、TIM3、TIM4、TIM5，每个定时器有 4 个输入捕获/输出比较/PWM/脉冲计数的通道。

含有 2 个基本定时器 TIM6、 TIM7，主要用于产生 DAC 触发信号。

含有 2 个看门狗定时器——独立看门狗和窗口看门狗。

含有 1 个系统时基定时器 SysTick，24 位递减计数器，自动重加载，常用于产生延时。

详细资料参考本文件目录下“STM32F103 中文资料.pdf，页码 157-266”。

关于 STM32 定时器 8 的使用介绍，本实验使用的是定时器 8，该定时器的配置步骤如下：

1、设置自动重装载寄存器的值。

2、设置时钟频率除数的预分频值。

3、设置时钟分割。

4、设置时钟向上计时。

5、清除更新标志。

6、清除中断标志。

7、使能中断。

8、使能定时器 8.

详细资料参考本文件目录下“STM32F103 中文资料.pdf，页码 211-256”。

实验内容

[1] 新建一个 Keil 工程。

[2] 配置 STM32 系统时钟、GPIO 时钟。

[3] 配置 GPIO 口工作模式。

[4] 配置定时器工作模式。

[5] 编写主函数。

[6] 进行基于 ST-LINK 的 Keil 仿真。

[7] 使用 STM32 ST-LINK Utility 软件下载程序。

[8] 观察实验现象。

实验步骤

[1] 新建一个 Keil 工程。

新建一个 Keil 工程。

[2] 配置 STM32 系统时钟、GPIO 时钟。按照如下步骤进行：

1) 恢复 RCC 时钟到默认值：RCC_DeInit()；
2) 开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；
3) 等待外部晶振启动：RCC_WaitForHSEStartUp()；
4) 代码延时两个周期：FLASH_SetLatency(FLASH_Latency_2)；
5) 半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；
6) 预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；
7) 选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；
8) 使能 PLL：RCC_PLLCmd(ENABLE)；
9) PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；
10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；
11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；
12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；
13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);
14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；
15)使能 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、TIM8、AFIO 时钟（这些和 RobotTTP 自身硬件配置相关，以避免上电\复位时相关 GPIO 口电平不确定）：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOE|RCC_APB2Periph_TIM8|RCC_APB2Periph_AFIO, ENABLE);
用户开发程序时，一般只需要从上面第 15 项开始修改。库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3] 配置 GPIO 口的工作模式。按照如下步骤进行：
1) 恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();
2) 配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);
3) 配置 TIM8 通道 4 管脚 PC9，用于驱动蜂鸣器：
考虑到在本实验中采用 PC9 管脚驱动蜂鸣器（如图 2-2-1 所示），而该管脚外部没有接上拉电阻，所以要配置为推挽输出，否则无法输出高电平：
GPIO_InitStructure.GPIO_Pin =GPIO_Pin_8;//选中 PE8
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//输出频率为 50MHz
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//IO 工作模式为推挽输出
GPIO_Init(GPIOE, &GPIO_InitStructure);//对选中管脚初始化
GPIO_ResetBits(GPIOE,GPIO_Pin_8);//PE8 输出低电平

详细的库函数使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf”。

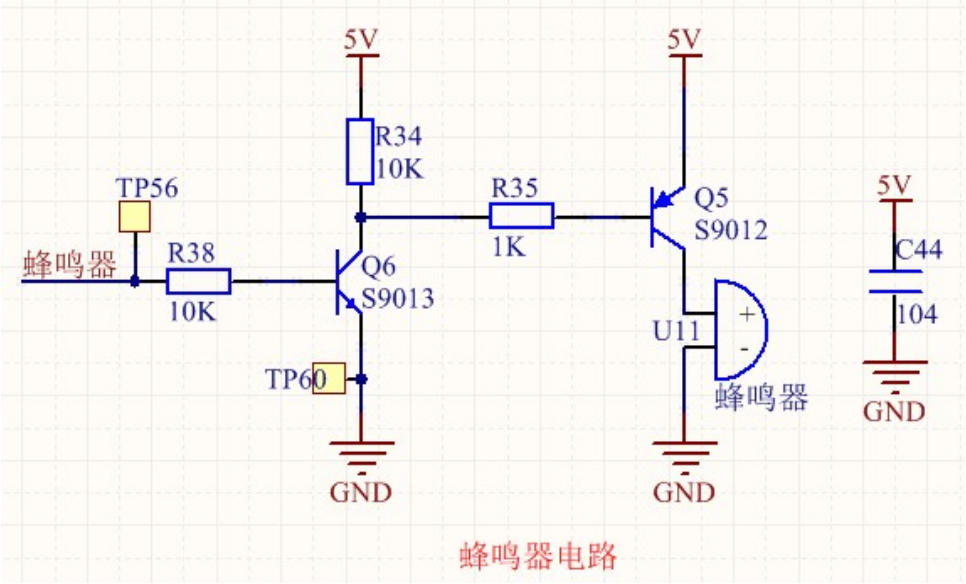


图 2-2-1 LED 驱动电路原理图

[4] 配置定时器 8 工作模式。

```
void TIM8_Int_Init(u32 arr,u16 psc)//TIM8 初始化
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_OCInitTypeDef TIM_OCInitStructure;
    TIM_DeInit(TIM8);//TIM8 寄存器为默认值
    TIM_TimeBaseStructure.TIM_Period = arr; //设置自动重装载寄存器的值
    TIM_TimeBaseStructure.TIM_Prescaler =psc; //设置时钟频率除数的预分频值
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1; //设置时钟分割
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //TIM 向上
    TIM_TimeBaseInit(TIM8, &TIM_TimeBaseStructure); //初始化 TIM8 时基
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2;//配置 PWM
    TIM_OCInitStructure.TIM_Pulse = arr/2;//CCR4 的值
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;//该通道输出
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;//通道 4 极性为低
    TIM_OC4Init(TIM8, &TIM_OCInitStructure);//初始化通道 4
    TIM_ITConfig(TIM8,TIM_IT_Update,DISABLE); //禁止中断
    TIM_ARRPreloadConfig(TIM8, ENABLE);//使能预装载
    TIM_OC4PreloadConfig(TIM8, TIM_OCPreload_Enable);//通道 4 预装载使能
    TIM_Cmd(TIM8, ENABLE); //使能 TIM8
    TIM_CtrlPWMOutputs(TIM8, ENABLE);
}
```

[5] 编写主函数。

第一步：包含头文件：

```
#include "stm32f10x.h"
```

第二步：声明程序中的子函数：

```
void delay(u32 i);//延时子程序
void RCC_Configuration(void);//RCC 配置子程序
void GPIO_Configuration(void);//IO 口初始化子程序
void TIM8_Int_Init(u32 arr,u16 psc);//TIM8 初始化
```

第三步：编写主函数函数体：

```
int main(void)
{
    RCC_Configuration();//RCC 配置
    GPIO_Configuration();//GPIO 配置
    while(1)
    {
            Music(Music_Code[choose][i]);
            i++;
            jiepai(Music_Code[choose][i]);
            i++;
            if(PlayWord==0)
                    TIM8_Int_Init(0,499);//定时器 PWM 频率改变
            else
                    TIM8_Int_Init(144000/PlayWord,499);//定时器 PWM 频率改变
            Delay(jie);
            if((choose==0)&&(i>=152))
            {choose=1;i=0;}
            else if((choose==1)&&(i>=38))
            {choose=2;i=0;}
            else if((choose==2)&&(i>=42))
            {choose=0;i=0;}
    }
}
```

[6] 进行基于 ST-LINK 的 Keil 仿真。

点击 Debug->Run，可以听到蜂鸣器播放音乐。

[7] 使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 FengMingQi_DingShiQi_YinYue.hex 文件烧写到 RobotTTP。

[8] 观察实验现象。

观察 LED 闪烁的现象。

参考程序

参考程序的工程名为“FengMingQi_DingShiQi_YinYue.uvproj”，在本实验文件目录下“FengMingQi_DingShiQi_YinYue”文件夹下。

实验小结

通过本节的学习，了解 STM32 时钟的种类及其 GPIO 口的工作模式，了解 STM32 定时器的工作模式，并掌握 STM32 时钟及其 GPIO 口的配置方法，并学会定时器驱动蜂鸣器。

思考题

如何用普通 IO 口驱动蜂鸣器？

2.3 触摸按键

实验目的

了解单片机中断及外部中断的概念。

学会 STM32 内部 EXTI（External Interrupt，外部中断）的使用方法。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件

实验预习

触摸按键的原理：

根据工作原理不同，触摸式按键可分为两大类：电阻式触摸按键与电容式感应按键。

电阻式的触摸按键原理非常类似于触摸屏技术，需要由多块导电薄膜上面按照按键的位置印制成的，因此这种按键需要在设备表面贴一张触摸薄膜。电阻式触摸屏一直由于其低廉的价格而深受厂商的喜爱，但是由于导电薄膜的耐用性较低，并且也会降低透光性，因此已经被越来越多的厂家所抛弃。

电容式触摸按键主要是为了克服电阻屏的耐用性所提出的，电容式触摸按键的结构与电阻式的相似，但是其采用电容量为判断标准。简单来说，就是一个 IC 控制的电路，该电路包括一个能放置在任何介质面板后的简单阻性环形电极组件，因此，按键的操作界面可以是一整块普通绝缘体（如有机玻璃一般材料都可），不需要在界面上挖孔，按键在介质下面，人手接近界面和下面的电极片形成电容，靠侦测电容量的变化来感应。温度，静电，水，灰尘等外界因素一般不会影响，界面没有太多要求，可以加上背光，音效等，靠人手感应，整个界面没有按键的存在，便于清洁，让产品在外观上更加高档美观，由于按键没有接点，使用寿命也是非常的长久，一般来说是半永久性。

中断的概念：

对于单片机来讲，中断是指 CPU 在处理某一事件 A 时，发生了另一事件 B，请求 CPU 迅速去处理（中断发生）；CPU 接到中断请求后，暂停当前正在进行的工作（中断响应），转去处理事件 B（执行相应的中断服务程序），待 CPU 将事件 B 处理完毕后，再回到原来事件 A 被中断的地方继续处理事件 A（中断返回），这一过程称为中断。

外部中断的概念：

外部中断就是外部事件引起的中断。STM32 中，每一个 GPIO 都可以触发一个外部中断，然而，GPIO 的中断是以组为一个单位的，同组间的外部中断同一时间只能使用一个。比如说，PA0，PB0，PC0，PD0，PE0，PF0，PG0 这些为 1 组，如果使用 PA

0 作为外部中断源，那么别的就不能够再使用了，在此情况下，我们只能使用类似于 PB1，PC2 这种末端序号不同的外部中断源。每一组使用一个中断标志 EXTIx。EXTI0 – EXTI4 这 5 个外部中断有着自己独立的中断响应函数，EXTI5-EXTI9 共用一个中断响应函数，EXTI10-EXTI15 共用一个中断响应函数。 对于中断的控制，STM32 有一个专用的管理机构：NVIC（嵌套向量中断控制器）。详细资料参考本文件目录下“STM32 外部中断详解.pdf”。

触摸按键简介：

首先是键的本身，它是直接利用 PCB 制做的，上边再覆盖一层绝缘层而成。结构上，触片的周围和背面都是地线。当手触摸时，则会改变触片极与地之间的等效电容量，使之加大，通过电路检测这一变化就可以判断出来触片被触。

实验内容

[1] 新建一个 Keil 工程。

[2] 配置 STM32 系统时钟、GPIO 时钟。

[3] 配置 GPIO 口工作模式。

[4] 外部中断配置。

[5] 中断向量配置。

[6] 编写中断处理函数。

[7] 编写主函数。

[8] 使用 STM32 ST-LINK Utility 软件下载程序。

[9] 观察实验现象。

实验步骤

[1] 新建一个 Keil 工程。

新建一个 Keil 工程。

[2] 配置 STM32 系统时钟、GPIO 时钟。按照如下步骤进行：

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE 、AFIO 时钟（这些和 RobotTP 自身硬件配置相关，以避免上电\复位时相关 GPIO 口电平不确定）：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOE|RCC_APB2Periph_AFIO, ENABLE);

库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3] 配置 GPIO 口的工作模式。按照如下步骤进行：

1)恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();

2)配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

3)配置端口：

本实验利用 PC12 管脚检测外部中断，将其配置成上拉输入模式。配置程序如下：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;//外部中断脚
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;//上拉输入
GPIO_Init(GPIOC, &GPIO_InitStructure);//对选中管脚初始化
```

[4] 外部中断配置。

```
void EXTI_Configuration(void)
{
 EXTI_InitTypeDef EXTI_InitStructure;  //初始化结构
 GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource12);//选择 PC12 管脚作为外部线路触发引脚
 EXTI_ClearITPendingBit(EXTI_Line12);//清除中断标志位 EXTI_Line12 对应相应的中断线
 EXTI_InitStructure.EXTI_Mode =EXTI_Mode_Interrupt; //选择中断模式请求
 EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;/上升沿沿触发
 EXTI_InitStructure.EXTI_Line = EXTI_Line12;  //选择使能的外部中断线
 EXTI_InitStructure.EXTI_LineCmd = ENABLE;  //定义选中线的新状态使能
 EXTI_Init(&EXTI_InitStructure);//配置上述参数
}
```

[5] 中断向量配置。

```
void NVIC_Configuration(void)//嵌套向量中断控制器配置
{
    NVIC_InitTypeDef NVIC_InitSructure;
    NVIC_InitSructure.NVIC_IRQChannel = EXTI15_10_IRQn;//开放线 12 的中断
    NVIC_InitSructure.NVIC_IRQChannelCmd = ENABLE;  //使能指定通道
    NVIC_Init(&NVIC_InitSructure);
}
```

[6] 编写中断处理函数

```
void EXTI15_10_IRQHandler(void)
{
    EXTI_ClearITPendingBit(EXTI_Line12);//清除 PC12 挂起位
    if(GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_12))//如果是高电平
    {
            delay(500000);//延时去抖动
            if(GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_12))
            {
                    LED_flag++;
                    if(LED_flag>1)
                            LED_flag=0;
                    if(LED_flag)
                            GPIO_SetBits(GPIOE,GPIO_Pin_0);//点亮 LED 灯
                    else
                            GPIO_ResetBits(GPIOE,GPIO_Pin_0);//关闭 LED 灯
            }
    }
}
```

[7] 编写主函数。

第一步：包含头文件：

```
#include "stm32f10x.h"
```

第二步：声明程序中的子函数：

```
void delay(u32 i);//延时子程序
void RCC_Configuration(void);//RCC 配置子程序
void GPIO_Configuration(void);//IO 口初始化子程序
void EXTI_Configuration(void);//外部中断配置
void NVIC_Configuration(void);//中断向量配置
```

第三步：编写主函数函数体：

```
void main(void)//主函数函数名
{
```

```
    RCC_Configuration();//RCC 配置
    GPIO_Configuration();//GPIO 配置
     EXTI_Configuration();//外部中断配置
    NVIC_Configuration();//中断向量配置
    while(1)//主循环
    {
            ;
    }
}
```

[8] 使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 ChuMoJian.hex 文件烧写到 RobotTTP。

[9] 观察实验现象。

用手触摸触摸按键，观察 LED 闪烁情况。

参考程序

参考程序的工程名为“ChuMoJian.uvproj”，在本实验文件目录下“ChuMoJian”文件夹下。

实验小结

通过本小节的学习，了解 STM32 中断的种类及其外部中断的原理，并掌握 STM32 外部中断的配置和使用方法。

思考题

本节实验中断返回具体是什么样的一个过程？

2.4 温湿度传感器

实验目的

了解单总线通信原理。

了解 DHT11 数字温湿度传感器的原理。

掌握运用 DHT11 温湿度传感器应用到实际产品中的能力。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件

实验预习

单总线原理介绍：

1-wire 单总线是 Maxim 全资子公司 Dallas 的一项专有技术。与目前多数标准串行数据通信方式，如 SPI/I2C/MICROWIRE 不同，它采用单根信号线，既传输时钟，又传输数据，而且数据传输是双向的。它具有节省 I/O 口线资源、结构简单、成本低廉、便于总线扩展和维护等诸多优点。1-wire 单总线适用于单个主机系统，能够控制一个或多个从机设备。当只有一个从机位于总线上时，系统可按照单节点系统操作；而当多个从机位于总线上时，则系统按照多节点系统操作。详细资料参考本文件目录下“单总线原理.pdf”。

DHT11 数字温湿度传感器概述：

DHT11 数字温湿度传感器是一款含有已校准数字信号输出的温湿度复合传感器。它应用专用的数字模块采集技术和温湿度传感技术， 确保产品具有极高的可靠性与卓越的长期稳定性。传感器包括一个电阻式感湿元件和一个 NTC 测温元件，并与一个高性能 8 位单片机相连接。因此该产品具有品质卓越、超快响应、抗干扰能力强、性价比极高等优点。每个 DHT11 传感器都在极为精确的湿度校验室中进行校准。校准系数以程序的形式储存在 OTP 内存中，传感器内部在检测信号的处理过程中要调用这些校准系数。 单线制串行接口， 使系统集成变得简易快捷。超小的体积、极低的功耗， 信号传输距离可达 20 米以上， 使其成为各类应用甚至最为苛刻的应用场合的最佳选则。详细资料参考本文件目录下“DHT11 说明书.pdf”。

实验内容

[1] 新建一个 Keil 工程。

[2] 配置 STM32 系统时钟、GPIO 时钟。

[3] 配置 GPIO 口工作模式。

[4] 编写温湿度读取函数。

[5] 编写主函数。

[6] 使用 STM32 ST-LINK Utility 软件下载程序。

[7] 观察实验现象。

实验步骤

[1] 新建一个 Keil 工程。

新建一个 Keil 工程。

[2] 配置 STM32 系统时钟、GPIO 时钟。按照如下步骤进行：

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、GPIOF、GPIOG、USART1、AFIO 时钟：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOE|RCC_APB2Periph_GPIOF|RCC_APB2Periph_GPIOG|RCC_APB2Periph_USART1|RCC_APB2Periph_AFIO, ENABLE);

库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3] 配置 GPIO 口的工作模式。按照如下步骤进行：

1)恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();

2)配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

3)配置端口：

考虑到在本实验中采用 PD3 读取数字温湿度传感器 DHT11 的数据，IO 口的方向要在通信时进行切换，所以暂时设置为浮空输入。

详细的库函数使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 120-133”。

[4] 编写温湿度读取函数。

```
void RH(void)
{
    //主机拉低 18ms
    ZongXian_Out;
    ZongXian_L;
    delay(144000);
    ZongXian_H;
```

```
//总线由上拉电阻拉高 主机延时 20us
Delay_10us();
Delay_10us();
Delay_10us();
Delay_10us();
//主机设为输入 判断从机响应信号
ZongXian_In;
//判断从机是否有低电平响应信号 如不响应则跳出，响应则向下运行
if(!ZongXian_DianPing)
{
        U8FLAG=2;
        //判断从机是否发出 80us 的低电平响应信号是否结束
        while((!ZongXian_DianPing)&&U8FLAG++);
        U8FLAG=2;
        //判断从机是否发出 80us 的高电平，如发出则进入数据接收状态
        while((ZongXian_DianPing)&&U8FLAG++);
        //数据接收状态
        COM();
        U8RH_data_H_temp=U8comdata;
        COM();
        U8RH_data_L_temp=U8comdata;
        COM();
        U8T_data_H_temp=U8comdata;
        COM();
        U8T_data_L_temp=U8comdata;
        COM();
        U8checkdata_temp=U8comdata;
        ZongXian_Out;
        ZongXian_H;
        //数据校验
        U8temp=(U8T_data_H_temp+U8T_data_L_temp+U8RH_data_H_temp+U8
        RH_data_L_temp);
        if(U8temp==U8checkdata_temp)
        {
                U8RH_data_H=U8RH_data_H_temp;//湿度整数部分
                U8RH_data_L=U8RH_data_L_temp;//湿度小数部分
                U8T_data_H=U8T_data_H_temp;//温度整数部分
                U8T_data_L=U8T_data_L_temp;//温度小数部分
                U8checkdata=U8checkdata_temp;//上面 4 个字节的校验和
                printf("WenDu=%d\r\n",U8T_data_H);
                printf("ShiDu=%d\r\n",U8RH_data_H);
        }
```

```
    }
}
```

[5] 编写主函数。

第一步：包含头文件：

```
#include "stm32f10x.h"
```

第二步：声明程序中的子函数：

```
void delay(u32 i);//延时子程序
void RCC_Configuration(void);//RCC 子程序
void GPIO_Configuration(void);//IO 初始化程序
void RH(void);//温湿度读取
void Delay_10us(void);//延时 10us
void  COM(void);
void USART_Config(void);
void NVIC_Configuration(void);//嵌套向量中断控制器配置
```

第三步：编写主函数函数体：

```
int main(void)
{
    RCC_Configuration();//RCC 配置
    GPIO_Configuration();//GPIO 配置
    USART_Config();
    NVIC_Configuration();
    while(1)
    {
            RH();
            delay(16000000);//2S 采集一次
    }
}
```

[6] 使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 WenShiDu.hex 文件烧写到 RobotTTP。

[7] 观察实验现象。

观察串口调试助手打印出的温湿度数据，如图 2-4-1 所示。

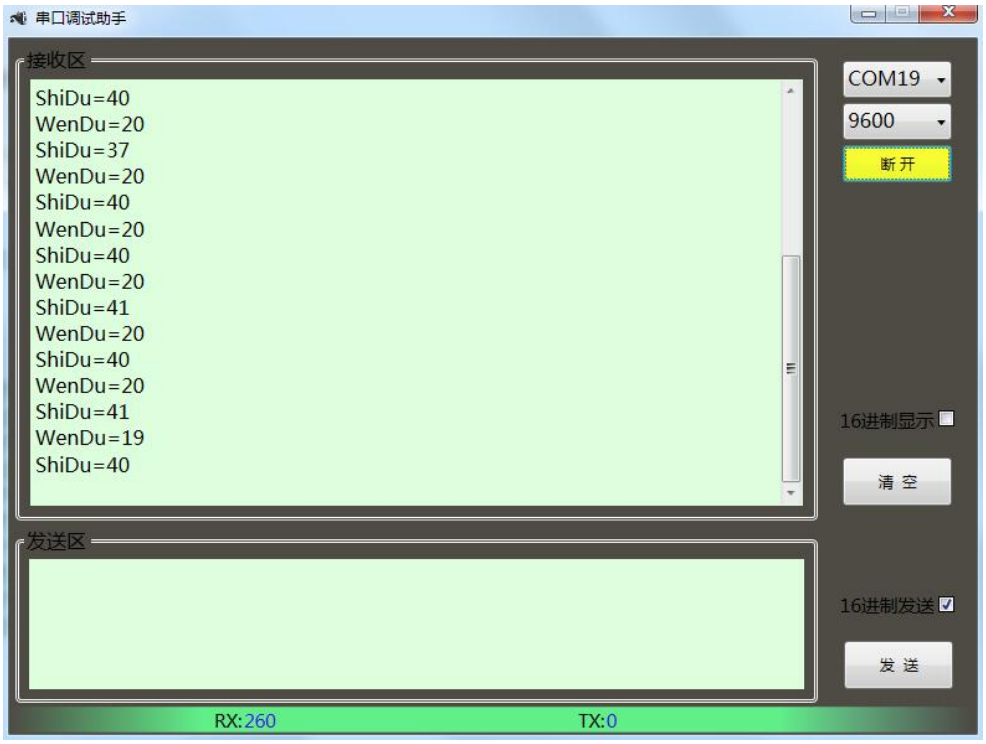


图 2-4-1 实验现象图示

参考程序

参考程序的工程名为“WenShiDu.uvproj”，在本实验文件目录下“WenShiDu”文件夹下。

实验小结

通过本节学习，了解单总线通信原理。了解 DHT11 数字温湿度传感器的原理。掌握运用 DHT11 温湿度传感器应用到实际产品中的能力。

思考题

运用 DHT11 做一个温度报警设备。

2.5 红外避障传感器

实验目的

了解红外避障传感器的原理。

学习掌握红外避障传感器在机器人应用中的使用方法。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件

实验预习

红外避障传感器的原理：

利用红外光的反射特性。在一定范围内，如果没有障碍物，发射出去的红外线，因为传播距离越远而逐渐减弱，最后消失。如果有障碍物，红外线遇到障碍物，被反射到达传感器接收头。传感器检测到这一信号，就可以确认正前方有障碍物，并送给处理器一个信号，告诉处理器前方出现障碍物。

实验内容

[1]新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

[3]配置 GPIO 口工作模式。

[4]配置外部中断。

[5]中断向量配置。

[6]编写中断处理函数。

[7]编写主函数。

[8]使用 STM32 ST-LINK Utility 软件下载程序。

[9]观察实验现象。

实验步骤

[1]新建一个 Keil 工程。

新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。按照如下步骤进行：

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 时钟：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOE|RCC_APB2Periph_AFIO, ENABLE);

库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3]配置 GPIO 口的工作模式。按照如下步骤进行：

1) 恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();

2) 配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

3) 配置端口：

考虑到在本实验中采用 PA15 检测外部中断信号，所以要配置为上拉输入：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15;//PA15---BIZHANG
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;//模式上拉输入
GPIO_Init(GPIOA, &GPIO_InitStructure);//对选中管脚初始化
```

详细的库函数使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 120-133”。

[4]配置外部中断。

```
void EXTI_Configuration(void)
{
 EXTI_InitTypeDef EXTI_InitStructure;  //初始化结构
 GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource15);//选择 PA15
 EXTI_ClearITPendingBit(EXTI_Line15);//清除中断标志位 EXTI_Line15
```

```
 EXTI_InitStructure.EXTI_Mode =EXTI_Mode_Interrupt; //选择中断模式请求
 EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;//下降沿触发
 EXTI_InitStructure.EXTI_Line = EXTI_Line15;  //选择使能的外部中断线
 EXTI_InitStructure.EXTI_LineCmd = ENABLE;   //定义选中线的新状态使能
 EXTI_Init(&EXTI_InitStructure);//配置上述参数
}
```

[5]中断向量配置。

```
void NVIC_Configuration(void)//嵌套向量中断控制器配置
{
    NVIC_InitTypeDef NVIC_InitSructure;
    NVIC_InitSructure.NVIC_IRQChannel = EXTI15_10_IRQn;//开放线 15 的中断
    NVIC_InitSructure.NVIC_IRQChannelCmd = ENABLE;  //使能指定通道
    NVIC_Init(&NVIC_InitSructure);
}
```

[6]编写中断处理函数。

```
void EXTI15_10_IRQHandler(void)
{
    EXTI_ClearITPendingBit(EXTI_Line15);//清除 PA15 挂起位
    if((GPIO_ReadInputData(GPIOA)&0x8000) == 0)
    {
            delay(500000);//延时去抖动
            if((GPIO_ReadInputData(GPIOA)&0x8000) == 0)
            {
                    GPIO_SetBits(GPIOE,GPIO_Pin_0);
            }
    }
    else
    {
            delay(500000);//延时去抖动
            if((GPIO_ReadInputData(GPIOA)&0x8000) == 1)
            {
                    GPIO_ResetBits(GPIOE,GPIO_Pin_0);
            }
    }
}
```

[7]编写主函数。

第一步：包含头文件：

```
#include "stm32f10x.h"
```

第二步：声明程序中的子函数：

```
void delay(u32 i);//延时子程序
void RCC_Configuration(void);//RCC 子程序
void GPIO_Configuration(void);//IO 初始化程序
void EXTI_Configuration(void);
void NVIC_Configuration(void);//嵌套向量中断控制器配置
第三步：编写主函数函数体：
int main(void)
{
    RCC_Configuration();//RCC 配置
    GPIO_Configuration();//GPIO 配置
    EXTI_Configuration();//初始化 PE7 外部中断
    NVIC_Configuration();//嵌套向量中断控制器配置
    while(1)
    {
            if(GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_15))
            {
                    delay(500000);//延时去抖动
                    if(GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_15))
                            GPIO_ResetBits(GPIOE,GPIO_Pin_0);
            }
    }
}
```

[8]使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 HongWaiBiZhang.hex 文件烧写到 RobotTTP。

[9]观察实验现象。

有障碍物时，可以看到 LD11 亮；没有障碍物时，可以看到 LD11 熄灭。

参考程序

参考程序的工程名为“HongWaiBiZhang.uvproj”，在本实验文件目录下“HongWaiBiZhang”文件夹下。

实验小结

通过本节学习，了解红外避障传感器的原理，并掌握红外避障传感器的使用方法。

思考题

对本实验的程序进行修改，实现有障碍物靠近时，让 LD11 处于闪烁状态，障碍物离开后，停止闪烁。

2.6 红外测距传感器

实验目的

了解红外测距传感器的原理。

学习掌握红外测距传感器在机器人领域的应用。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件

实验预习

红外测距传感器的原理：

红外测距传感器基于三角测量原理。红外发射器按照一定的角度发射红外光束，当遇到物体以后，光束 会反射回来，如图 2-6-1 所示。反射回来的红外光线被 CCD 检测器检测到以后，会获得一个偏移值 L，利用三角关系，已知发射角度 a，偏移距 L，中心矩 X，以及滤镜的焦距 F 以后，传感器到物体的距离 D 就可以通过几何关系计算出来了。

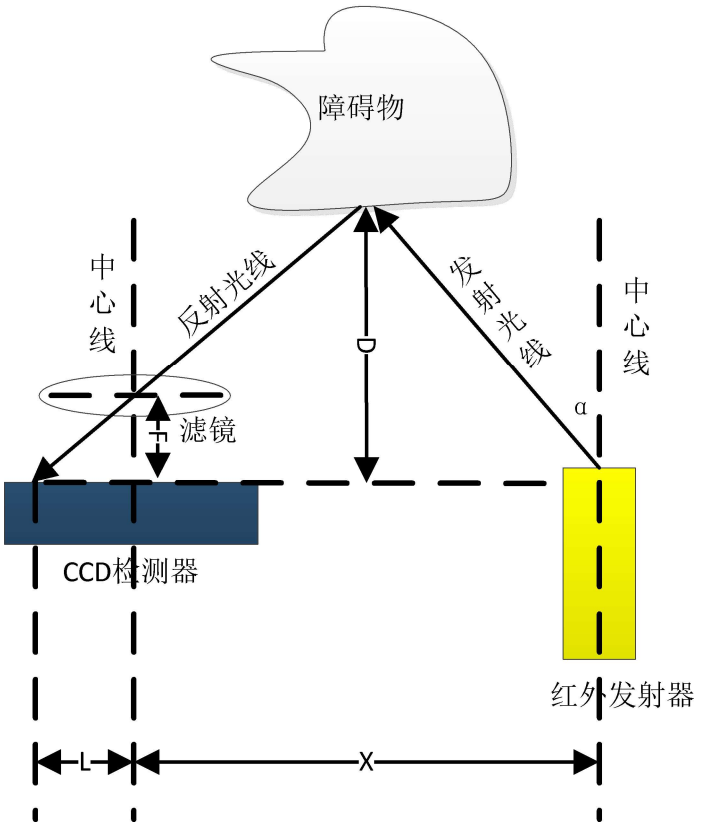


图 2-6-1 红外测距原理

ADC 简介：

ADC，Analog-to-Digital Converter 的缩写，指模/数转换器或者模拟/数字转换器。是指将连续变化的模拟信号转换为离散的数字信号的器件。真实世界的模拟信号，例如温度、压力、声音或者图像等，需要转换成更容易储存、处理和发射的数字形式。模/数转换器可以实现这个功能，在各种不同的产品中都可以找到它的身影。

典型的模拟数字转换器将模拟信号转换为表示一定比例电压值的数字信号。然而，有一些模拟数字转换器并非纯的电子设备，例如旋转编码器，也可以被视为模拟数字转换器。

数字信号输出可能会使用不同的编码结构。通常会使用二进制二补数（也称作“补码”）进行表示，但也有其他情况，例如有的设备使用格雷码（一种循环码）。

模拟信号在时域上是连续的，因此可以将它转换为时间上连续的一系列数字信号。这样就要求定义一个参数来表示新的数字信号采样自模拟信号速率。这个速率称为转换器的采样率（sampling rate）或采样频率（sampling frequency）。

可以采集连续变化、带宽受限的信号（即每隔一时间测量并存储一个信号值），然后可以通过差值将转换后的离散信号还原为原始信号。这一过程的精确度受量化误差的限制。然而，仅当采样率比信号频率的两倍还高的情况下才可能达到对原始信号的忠实还原，这一规律在采样定理有所体现。

由于实际使用的模拟数字转换器不能进行完全实时的转换，所以对输入信号进行一次转换的过程中必须通过一些外加方法使之保持恒定。常用的有采样－保持电路，在大多数的情况里，通过使用一个电容器可以存储输入的模拟电压，并通过开关或门电路来闭合、断开这个电容和输入信号的连接。许多模拟数字转换集成电路在内部就已经包含了这样的采样－保持子系统。

实验内容

[1]新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

[3]配置 GPIO 口工作模式。

[4]配置串口。

[5]配置 ADC。

[6]配置 TIM2。

[7]编写获取 AD 程序。

[8]编写主函数。

[9]使用 STM32 ST-LINK Utility 软件下载程序。

[10]观察实验现象。

实验步骤

[1]新建一个 Keil 工程。

新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。按照如下步骤进行：

1)恢复 RCC 时钟到默认值：RCC_DeInit()；
2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；
3)等待外部晶振启动：RCC_WaitForHSEStartUp()；
4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；
5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；
6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；
7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；
8)使能 PLL：RCC_PLLCmd(ENABLE)；
9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；
10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；
11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；
12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；
13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);
14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；
15)使能时钟：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOE|RCC_APB2Periph_AFIO|RCC_APB2Periph_ADC1|RCC_APB2Periph_USART1, ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3]配置 GPIO 口的工作模式。按照如下步骤进行：
1)恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();
2)配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);
3)配置端口：
本实验用到的电压检测引脚是 PB1，对应的是 AD1 的通道 9，配置如下：
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;//PB1 是电池检测
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;//模拟输入
GPIO_Init(GPIOB, &GPIO_InitStructure);//对选中管脚初始化

[4]配置串口。
```
void USART_Config(void)
{
```

```
 USART_InitTypeDef USART_InitStructure;
 USART_DeInit(USART1);//复位串口 1 寄存器
 USART_InitStructure.USART_BaudRate = 9600;//波特率 9600bps
 USART_InitStructure.USART_WordLength = USART_WordLength_8b;//数据位 8 位
 USART_InitStructure.USART_StopBits = USART_StopBits_1; //停止位 1 位
 USART_InitStructure.USART_Parity = USART_Parity_No; //无校验位
 USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowContr
 ol_None;   //无硬件流控
 USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
 USART_Init(USART1, &USART_InitStructure);//配置串口参数
 USART_ITConfig(USART1, USART_IT_RXNE, ENABLE); //使能接收中断
 USART_ClearFlag(USART1, USART_FLAG_TC );//清除发送完成标志位
 USART_Cmd(USART1, ENABLE);//使能串口
}
```

[5]配置 ADC。

```
void HongWaiCeJu_ADC_Configuration(void)
{
    ADC_InitTypeDef ADC_InitStructure;
    ADC_DeInit(ADC1);//复位 ADC1
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;//ADC 工作在独立模式
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;//单通道
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;//单次转换
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;//软件
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;//转换数据右对齐
    ADC_InitStructure.ADC_NbrOfChannel = 1;//转换通道 1 个通道
    ADC_Init(ADC1, &ADC_InitStructure);//开始初始化以上参数
   ADC_RegularChannelConfig(ADC1, ADC_Channel_9,1,ADC_SampleTime_239Cyc
   les5);//配置规则通道的通道 9 的采样顺序及采样周期
    ADC_Cmd(ADC1, ENABLE);//使能 ADC，并启动转换
    ADC_ResetCalibration(ADC1);//重置 ADC1 校准寄存器
   while(ADC_GetResetCalibrationStatus(ADC1));//获取 ADC1 重置校准寄存器的状
   态,等待校准寄存器初始化完成
    ADC_StartCalibration(ADC1);//开始 AD 校准
    while(ADC_GetCalibrationStatus(ADC1));//获取 ADC1 校准的状态,等待校准完成
}
```

[6]配置 TIM2。

```
void TIM2_Int_Init(u16 arr,u16 psc)//tim2 初始化
{
```

```
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_TimeBaseStructure.TIM_Period = arr; //设置自动重装载寄存器的值
    TIM_TimeBaseStructure.TIM_Prescaler =psc; //设置时钟频率除数的预分频值
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1; //设置时钟分割
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //TIM 向上
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure); //初始化 TIM2
    TIM_ClearFlag(TIM2, TIM_FLAG_Update); //清除更新标志位
    TIM_ClearITPendingBit(TIM2, TIM_FLAG_Update); //清除 TIM2 等待中断标志
    位
    TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE); //允许更新中断
    TIM_Cmd(TIM2, ENABLE); //使能 TIM2
}
```

[7]编写获取 AD 值程序。

```
u16 get_adc(void)
{
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);//使用外部触发，开始转换规则
    while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC));//获取转换结束标志位
    return ADC_GetConversionValue(ADC1);
}
```

[8]编写主函数。

第一步：包含头文件：

```
#include "stm32f10x.h"
```

第二步：声明程序中的子函数：

```
void delay(u32 i);//延时子程序
void RCC_Configuration(void);//RCC 子程序
void GPIO_Configuration(void);//IO 初始化程序
void NVIC_Configuration(void);//嵌套向量中断控制器配置
void TIM2_Int_Init(u16 arr,u16 psc);//tim2 初始化
```

第三步：编写主函数函数体：

```
int main(void)
{
    float dianya=0;
    RCC_Configuration();//RCC 配置
    GPIO_Configuration();//GPIO 配置
    USART_Config();
    TIM2_Int_Init(999,359);//tim2 初始化
    HongWaiCeJu_ADC_Configuration();
    NVIC_Configuration();//嵌套向量中断控制器配置
    while(1)
```

```
{
        if(tim2_count>=20)//5ms*20=100ms
        {
                tim2_count=0;
                dianya=get_adc();
                if(dianya>V20) {printf("JuLiXiaoYu20cm");}
                else if(dianya>V30) {
                        dianya=30-(dianya-V30)/K2030;
                        printf("JuLi=%2.0fcm",dianya); }
                else if(dianya>V40) {
                        dianya=40-(dianya-V40)/K3040;
                        printf("JuLi=%2.0fcm",dianya); }
                else if(dianya>V50) {
                        dianya=50-(dianya-V50)/K4050;
                        printf("JuLi=%2.0fcm",dianya); }
                else if(dianya>V60) {
                        dianya=60-(dianya-V60)/K5060;
                        printf("JuLi=%2.0fcm",dianya); }
                else if(dianya>V70) {
                        dianya=70-(dianya-V70)/K6070;
                        printf("JuLi=%2.0fcm",dianya); }
                else if(dianya>V80) {
                        dianya=80-(dianya-V80)/K7080;
                        printf("JuLi=%2.0fcm",dianya); }
                else if(dianya>V90) {
                        dianya=90-(dianya-V90)/K8090;
                        printf("JuLi=%2.0fcm",dianya); }
                else if(dianya>V100) {
                        dianya=100-(dianya-V100)/K90100;
                        printf("JuLi=%2.0fcm",dianya); }
                else if(dianya>V110) {
                        dianya=110-(dianya-V110)/K100110;
                        printf("JuLi=%2.0fcm",dianya); }
                else if(dianya>V120) {
                        dianya=120-(dianya-V120)/K110120;
                        printf("JuLi=%2.0fcm",dianya); }
                else if(dianya>V130) {
                        dianya=130-(dianya-V130)/K120130;
                        printf("JuLi=%2.0fcm",dianya); }
                else if(dianya>V140) {
                        dianya=140-(dianya-V140)/K130140;
                        printf("JuLi=%2.0fcm",dianya); }
                else if(dianya>=V150) {
                        dianya=150-(dianya-V150)/K140150;
```

```
                    printf("JuLi=%2.0fcm",dianya); }
                else{printf("JuLiDaYu150cm");}
            }
        }
    }
```

[9]使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 HongWaiCeJu.hex 文件烧写到 RobotTTP。

[10]观察实验现象。

观察实验现象。

参考程序

参考程序的工程名为“HongWaiCeJu.uvproj”，在本实验文件目录下“HongWaiCeJu”文件夹下。

实验小结

通过本节学习，了解红外测距传感器的原理及相关知识，并掌握 STM32 内部 ADC 的配置及使用方法，能够基于红外测距传感器应用于机器人领域。

思考题

对本实验的程序进行修改，编写相应的程序，当距离小于某个值时进行报警，提示机器人进行避让。

2.7 超声波测距传感器

实验目的

了解超声波传感器相关知识及其在机器人系统中应用。

学习掌握超声波传感器的使用方法。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件

实验预习

超声波测距传感器原理：

超声波测距原理是通过超声波发射器向某一方向发射超声波,在发射时刻的同时开始计时,超声波在空气中传播时碰到障碍物就立即返回来,超声波接收器收到反射波就立即停止计时。超声波在空气中的传播速度为 v ,而根据计时器记录的测出发射和接收回波的时间差△t ,就可以计算出发射点距障碍物的距离 S ,即:

$$S = v \cdot \triangle t / 2 \qquad ①$$

这就是所谓的时间差测距法。

实验内容

[1]新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

[3]配置 GPIO 口工作模式。

[4]配置 TIM1。

[5]配置中断向量。

[6]编写 TIM1 中断服务函数。

[7]编写主函数。

[8]使用 STM32 ST-LINK Utility 软件下载程序。

[9]观察实验现象。

实验步骤

[1] 新建一个 Keil 工程。

新建一个 Keil 工程。

[2] 配置 STM32 系统时钟、GPIO 时钟。按照如下步骤进行：

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能时钟：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOE|RCC_APB2Periph_TIM1|RCC_APB2Periph_AFIO|RCC_APB2Periph_USART1, ENABLE);

库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3] 配置 GPIO 口的工作模式。按照如下步骤进行：

1)恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();

2)配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

3)配置端口：

本实验用到的触发端口是 PA12，用到的输入捕获端口是 PA11。

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;//
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//模式通用推挽输出
GPIO_Init(GPIOA, &GPIO_InitStructure);//对选中管脚初始化
GPIO_ResetBits(GPIOA, GPIO_Pin_12);//
```

[4] 配置 TIM1。

```
void TIM1_Int_Init(void)//TIM1 初始化
{
```

```
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_DeInit(TIM1);//TIM1 寄存器为默认值
    TIM_InternalClockConfig(TIM1);
    //预分频系数为 36000-1，这样计数器时钟为 72MHz/36000 = 2kHz
    TIM_TimeBaseStructure.TIM_Prescaler = 3600 - 1;
    //设置时钟分割
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    //设置计数器模式为向上计数模式
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    //设置计数溢出大小，每计 2000 个数就产生一个更新事件
    TIM_TimeBaseStructure.TIM_Period = 0XFFFF-1;//2000-1
    //将配置应用到 TIM1 中
    TIM_TimeBaseInit(TIM1,&TIM_TimeBaseStructure);
    //禁止 ARR 预装载缓冲器
    TIM_ARRPreloadConfig(TIM1, DISABLE);
    //下面是对 TIM_ICInitStructure 的配置
    TIM_ICInitStructure.TIM_Channel = TIM_Channel_4;
    TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;//;//上升沿
    TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
    TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
    TIM_ICInitStructure.TIM_ICFilter = 0x0;
     TIM_ICInit(TIM1, &TIM_ICInitStructure);
    //开启 TIM1 的中断
    TIM_ClearITPendingBit(TIM1, TIM_IT_CC1);
    TIM_ITConfig(TIM1,TIM_IT_CC4,ENABLE);//允许捕获中断
    TIM_Cmd(TIM1, ENABLE);  //使能 TIMx 外设
}
```

[5] 配置中断向量。

```
void NVIC_Configuration(void)//嵌套向量中断控制器配置
{
    NVIC_InitTypeDef NVIC_InitSructure;
    NVIC_InitSructure.NVIC_IRQChannel = TIM1_CC_IRQn;//定时器 1 捕获中断
    NVIC_InitSructure.NVIC_IRQChannelCmd = ENABLE;  //使能指定通道
    NVIC_Init(&NVIC_InitSructure);

    NVIC_InitSructure.NVIC_IRQChannel = USART1_IRQn;//开放串口 1 中断
    NVIC_InitSructure.NVIC_IRQChannelCmd = ENABLE;  //使能指定通道
    NVIC_Init(&NVIC_InitSructure);
}
```

[6] 编写 TIM1 中断服务函数。

```
void TIM1_CC_IRQHandler(void)
{
 if(TIM_GetITStatus(TIM1, TIM_IT_CC4) == SET)//如果是捕获中断
 {
        TIM_ClearITPendingBit(TIM1, TIM_IT_CC4);
        ShangShengXiaJiangYan_Count++;
        if(ShangShengXiaJiangYan_Count>1)
                ShangShengXiaJiangYan_Count=0;
        if(ShangShengXiaJiangYan_Count)//如果是上升沿捕获
        {
                FanZhuan=TIM_GetCapture4(TIM1);
                TIM_OC4PolarityConfig(TIM1, TIM_ICPolarity_Falling);//设置为
        }
        else//如果是下降沿捕获
        {
                FanZhuan1=TIM_GetCapture4(TIM1);
                TIM_OC4PolarityConfig(TIM1, TIM_ICPolarity_Rising); //设置
                if(FanZhuan1>FanZhuan)//未溢出
                        FanZhuan2=FanZhuan1-FanZhuan;
                else//溢出
                        FanZhuan2=0xFFFF+FanZhuan1-FanZhuan;
                JuLi=FanZhuan2;
                JuLi = JuLi * 0.85;
                printf("JuLi=%fCM",JuLi);
        }
 }
}
```

[7] 编写主函数。

第一步：包含头文件：

```
#include "stm32f10x.h"
```

第二步：声明程序中的子函数：

```
void delay(u32 i);//延时子程序
void RCC_Configuration(void);//RCC 子程序
void GPIO_Configuration(void);//IO 初始化程序
void Delay_10us(void);
void TIM1_Int_Init(void);//TIM1 初始化
void NVIC_Configuration(void);//嵌套向量中断控制器配置
void USART_Config(void);
```

第三步：编写主函数函数体：

```
int main(void)
```

```
{
    RCC_Configuration();//RCC 配置
    GPIO_Configuration();//GPIO 配置
    TIM1_Int_Init();
    USART_Config();
    NVIC_Configuration();
    while(1)
    {
            ShangShengXiaJiangYan_Count=0;//Count 清零，防止数大
            TIM_OC4PolarityConfig(TIM1, TIM_ICPolarity_Rising); //设置为上升沿
            Trig_L;//首先将控制端拉低
            Delay_10us();
            Trig_H;//加一个 20us 的高电平
            Delay_10us();
            Delay_10us();
            Trig_L;//
            delay(8000000);//延时 1S
    }
}
```

[8] 使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 ChaoShengBo.hex 文件烧写到 RobotTTP。

[9] 观察实验现象。

观察实验现象。

参考程序

参考程序的工程名为“ChaoShengBo.uvproj”，在本实验文件目录下“ChaoSheng Bo”文件夹下。

实验小结

通过本节学习，了解超声波测距传感器的原理，并掌握超声波测距传感器的使用。

思考题

对本实验的程序进行修改，编写相应的程序，当距离小于某个特定值时，进行报警。

2.8 WIFI 通信

实验目的

了解 WiFi 相关知识。

学习 WiFi 的 AT 指令。

学会使用 RobotTTP 本体上 WiFi 模块进行通信。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）、笔记本（或台式机加便携式无线网卡）（一台）

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件，TTCDS-IMR 软件，网络调试助手软件

实验预习

WiFi 简介：

WiFi 全称 Wireless Fidelity，又称 802.11b 标准，它的最大优点就是传输速度较高，可以达到 11Mbps，另外它的有效距离也很长，同时也与已有的各种 802.11 DSSS 设备兼容。迅驰技术就是基于该标准的，无线上网已经成为现实。

IEEE 802.11b 无线网络规范是 IEEE 802.11 网络规范的变种，最高带宽为 11 Mbps，在信号较弱或有干扰的情况下，带宽可调整为 5.5Mbps、2Mbps 和 1Mbps，带宽的自动调整，有效地保障了网络的稳定性和可靠性。其主要特性为：速度快，可靠性高，在开放性区域，通讯距离可达 305 米，在封闭性区域，通讯距离为 76 米到 122 米，方便与现有的有线以太网络整合，组网的成本更低。

WiFi—WirelessFidelity，无线保真技术与蓝牙技术一样，同属于在办公室和家庭中使用的短距离无线技术。该技术使用的是 2.4GHz 附近的频段，该频段目前尚属没用许可的无线频段。其目前可使用的标准有两个，分别是 IEEE802.11a 和 IEEE802.11b。该技术由于有着自身的优点，因此受到厂商的青睐。

AT 指令简介：

AT 指令集是从终端设备（Terminal Equipment，TE)或数据终端设备（Data Terminal Equipment，DTE)向终端适配器(Terminal Adapter，TA)或数据电路终端设备(Data Circuit Terminal Equipment，DCE)发送的。

其对所传输的数据包大小有定义：即对于 AT 指令的发送，除 AT 两个字符外，最多可以接收 1056 个字符的长度（包括最后的空字符）。

每个 AT 命令行中只能包含一条 AT 指令；对于由终端设备主动向 PC 端报告的 URC 指示或者 response 响应，也要求一行最多有一个，不允许上报的一行中有多条指示或者响应。AT 指令以回车作为结尾，响应或上报以回车换行为结尾。

AT 指令举例：

指令	响应	功能
AT	OK	测试 AT 启动
AT+CWMODE	OK	设置 WiFi 模式
AT+RST	OK	重启 WiFi
AT+CIPMUX	OK	启用多连接或者单连接
AT+CIPSERVER	OK	建立服务器
AT+CIPSEND	SEND OK	发送数据

实验内容

[1]新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

[3]配置 GPIO 口工作模式。

[4]配置串口模块。

[5]编写串口中断处理函数。

[6]编写主函数。

[7]使用 STM32 ST-LINK Utility 软件下载程序。

[8]依次用 AT 指令配置 WIFI。

[9]使用网络调试助手连接 WIFI，观察实验现象。

实验步骤

[1]新建一个 Keil 工程。

新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。按照如下步骤进行：

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能时钟：RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1,ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOE|RCC_APB2Periph_AFIO|RCC_APB2Periph_USART1, ENABLE);//使能
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3,ENABLE);

[3]配置 GPIO 口的工作模式。按照如下步骤进行：

1) 恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();
2) 配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);
3) 配置端口：

本实验用到的串口 1 发送管脚是 PA9，串口 3 发送管脚 PB10，配置如下：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;//PA9 是串口发送
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;//复用推挽输出
GPIO_Init(GPIOA, &GPIO_InitStructure);//对选中管脚初始化
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;//PB10---TxD3
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;//复用推挽输出
GPIO_Init(GPIOB, &GPIO_InitStructure);//对选中管脚初始化
```

[4]配置串口模块。

```
oid USART_Config(void)
{
 USART_InitTypeDef USART_InitStructure;
 USART_DeInit(USART1);//复位串口 1 寄存器
 USART_InitStructure.USART_BaudRate = 115200;//波特率 9600bps
 USART_InitStructure.USART_WordLength = USART_WordLength_8b;//数据位 8 位
 USART_InitStructure.USART_StopBits = USART_StopBits_1;//停止位 1 位
 USART_InitStructure.USART_Parity = USART_Parity_No;
   //无校验位
 USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowContr
 ol _None;  //无硬件流控
 USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
 USART_Init(USART1, &USART_InitStructure);//配置串口参数
 USART_ITConfig(USART1, USART_IT_RXNE, ENABLE); //使能接收中断
```

```
 USART_ClearFlag(USART1, USART_FLAG_TC );//清除发送完成标志位
 USART_Cmd(USART1, ENABLE);//使能串口
 USART_DeInit(USART3);//复位串口 3 寄存器
 USART_InitStructure.USART_BaudRate = 115200;//波特率 115200
 USART_InitStructure.USART_WordLength = USART_WordLength_8b; //数据位
 USART_InitStructure.USART_StopBits = USART_StopBits_1; //停止位 1 位
 USART_InitStructure.USART_Parity = USART_Parity_No;//无校验位
 USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowContr
 ol_None;   //无硬件流控
 USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;   //收
 USART_Init(USART3, &USART_InitStructure);//配置串口参数
 USART_ITConfig(USART3, USART_IT_RXNE, ENABLE); //使能接收中断
 USART_ClearFlag(USART3, USART_FLAG_TC );//清除发送完成标志位
 USART_Cmd(USART3, ENABLE);//使能串口
}
```

[5]编写串口中断处理函数。

```
void USART1_IRQHandler(void)
{
    u8 num;
    u8 i;

    if(USART_GetITStatus(USART1,USART_IT_IDLE) == SET)
    {
            USART_ReceiveData(USART1); // Clear IDLE interrupt flag bit//清掉空闲
            DMA_Cmd(DMA1_Channel5,DISABLE);//关闭接收 DMA
            DMA_ClearITPendingBit(DMA1_IT_GL5);//
            DMA_ClearFlag( DMA1_FLAG_GL5 );  // 清 DMA 标志位
            num = UART1_BUFF_SIZE - DMA1_Channel5->CNDTR;
            for(i=0;i<num;i++)
            {
                    usart3_Send[i]=usart1_Receive[i];//将串口 1 收到的数据转到
                    usart1_Receive[i]=0;
            }
            if(usart3_Send[num-1]=='\n')
            {
                    usart1Receive_Count=num;//串口 1 接收字节数计数
                    usart1Receive_Flag=1;//串口 1 接收完一帧标志
            }
            else//出错
            {
```

```
            USART_SendData(USART1,0xaa);//发送回复帧
            while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
            USART_SendData(USART1,0x55);//发送回复帧
            while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
            }
            DMA1_Channel5->CNDTR = UART1_BUFF_SIZE;
            DMA_Cmd(DMA1_Channel5,ENABLE);
    }
}
void USART3_IRQHandler(void)
{
    if(USART_GetITStatus(USART3,USART_IT_RXNE))
    {
            USART_ClearITPendingBit(USART3,USART_IT_RXNE);//清除 USART3
            usart3_Receive[usart3Receive_Count]=USART_ReceiveData(USART3);
            if(usart3_Receive[usart3Receive_Count]=='\n')//一帧数据结束标志
            {
                    usart3Receive_Flag=1;//接收到完整的一帧数据置标志
            }
            usart3Receive_Count++;//接收到的数据加 1
    }
}
```

[6]编写主函数。

第一步：包含头文件：

```
#include "stm32f10x.h"
```

第二步：声明程序中的子函数：

```
void delay(u32 i);//延时子程序
void RCC_Configuration(void);//RCC 子程序
void GPIO_Configuration(void);//IO 初始化程序
void NVIC_Configuration(void);//嵌套向量中断控制器配置
void USART_Config(void);
void KeHuDuan_TouChuan(void);//工作在客户端透传模式
void uart_uart1_send(u8 num);//串口 1
void ShangChuanShuJuToUsart1(void);//上传数据到串口 1
void FaSongShuJuToUart3(void);//发送数据到串口 3
int fputc(int ch,FILE* f);
```

第三步：编写主函数函数体：

```
int main(void)
{
    RCC_Configuration();//RCC 配置
```

```
    GPIO_Configuration();//GPIO 配置
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);//分组 2
    USART_Config();
    NVIC_Configuration();
    delay(8000);//延时 1ms
    KeHuDuan_TouChuan();//设置模块工作在客户端透传模式
    while(1)
    {
    ShangChuanShuJuToUsart1();//把收到的串口 3 的数据通过串口 1 发送到上位机
    FaSongShuJuToUart3();//把收到的串口 1 的数据通过串口 3 发送到 WIFI 模块
    }
}
```

[7]使用 STM32 ST-LINK Utility 软件下载程序。
将本工程生成的 WIFI.hex 文件烧写到 RobotTTP。

[8]依次用 AT 指令配置 WIFI。

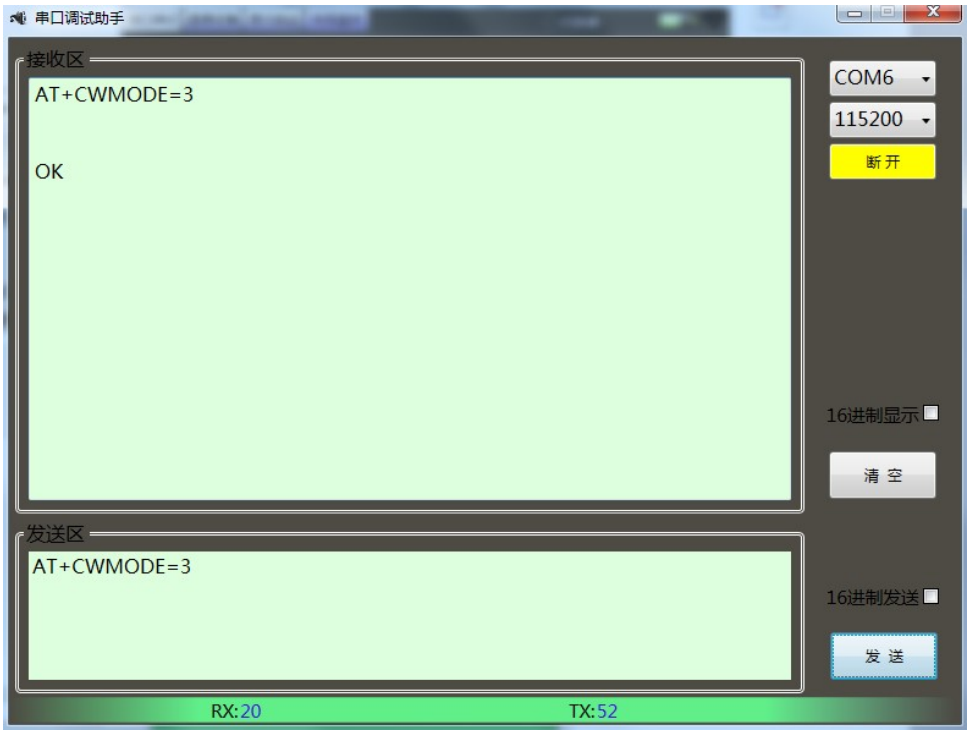


图 2-8-1 配置 WIFI 为 softAP+station 共存模式

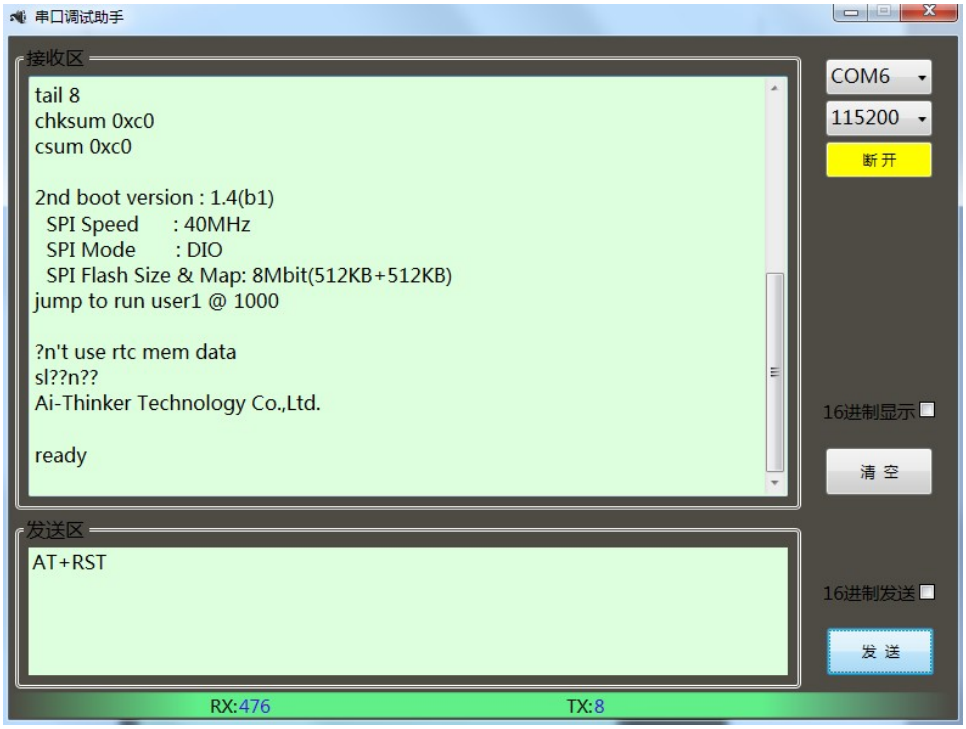


图 2-8-2 重启 WIFI 使设置生效

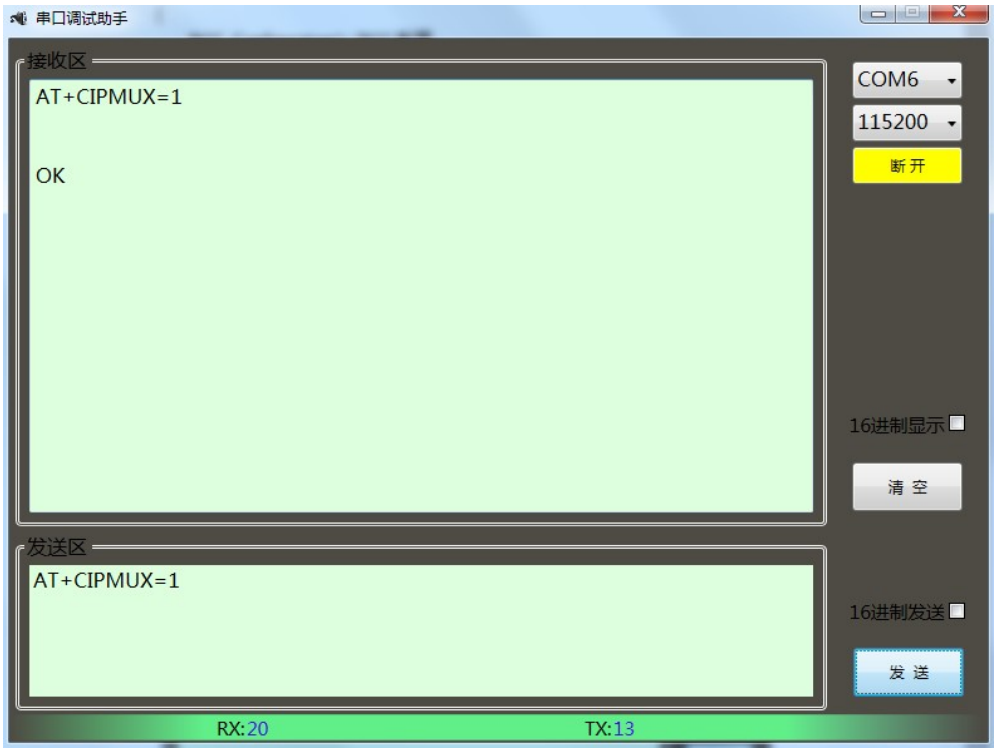


图 2-8-3 启用多连接

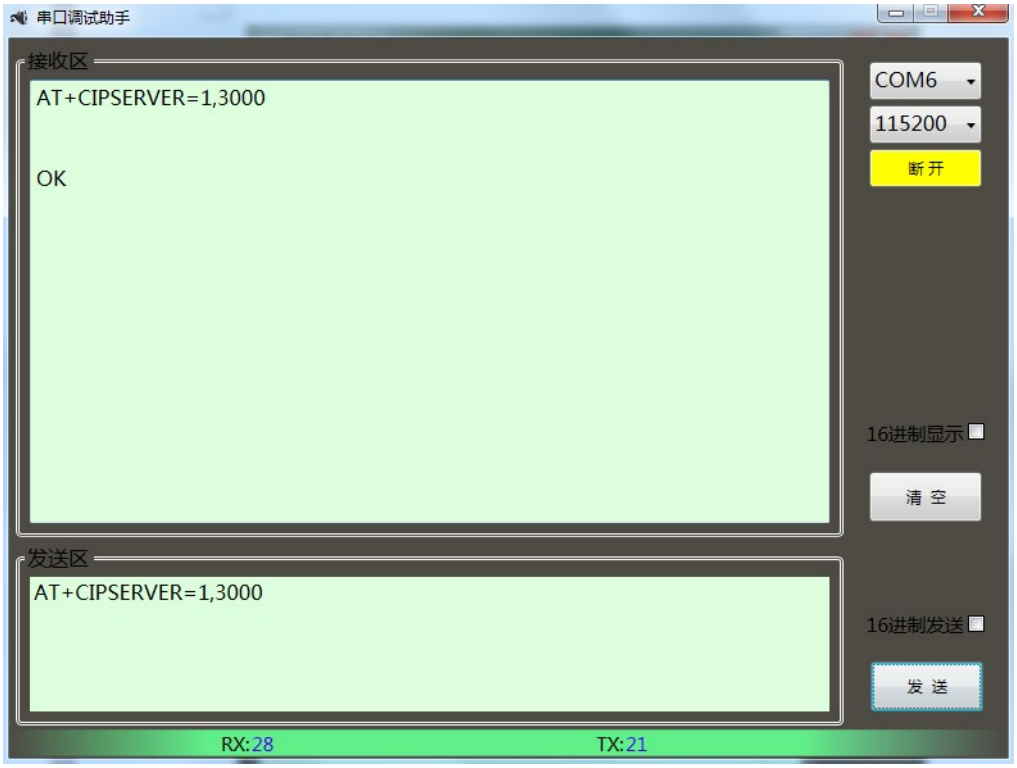


图 2-8-4 建立 server

图 2-8-5 电脑连接 RobotTTP 本体 WIFI

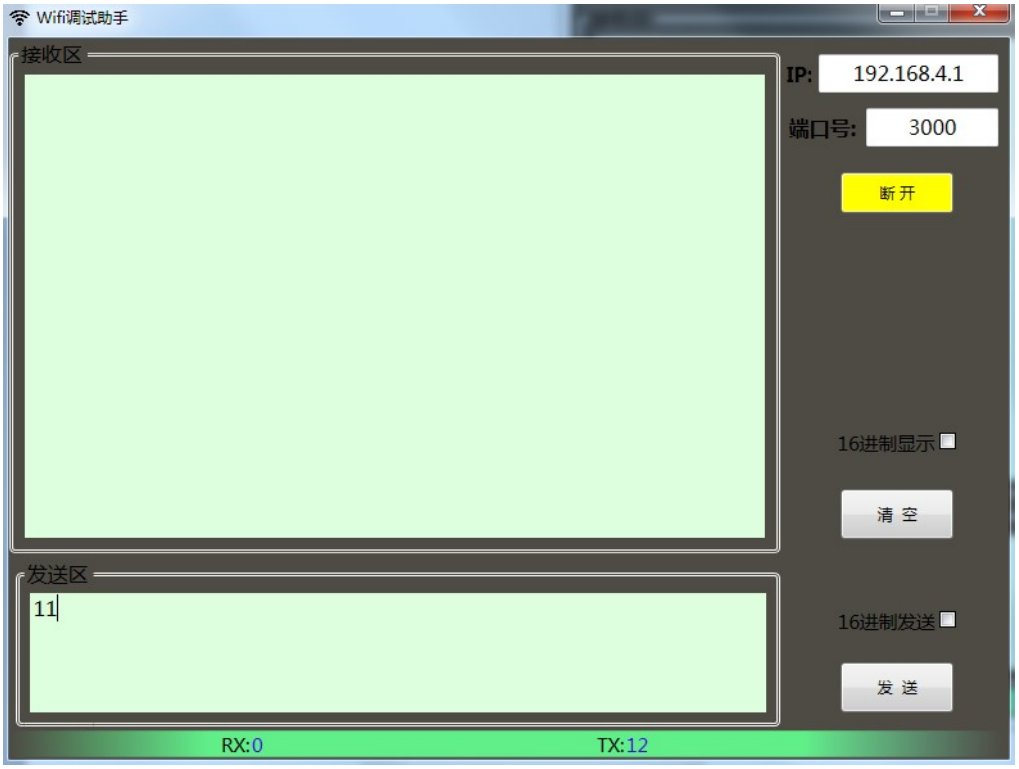


图 2-8-6 网络调试助手发送数据界面

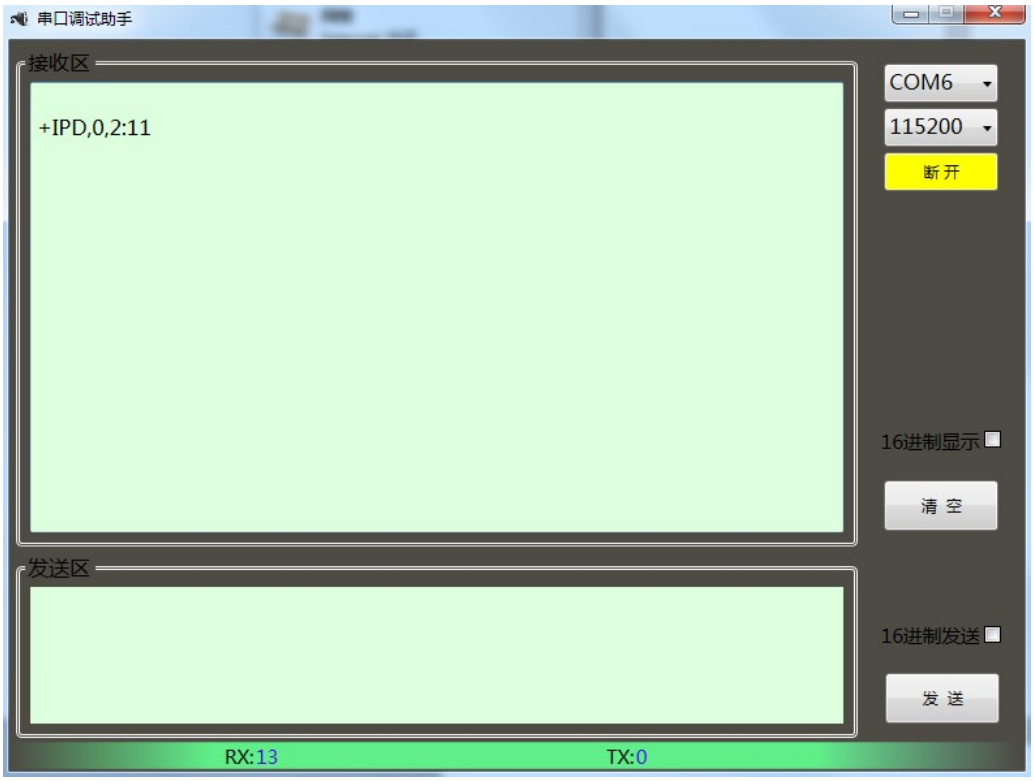


图 2-8-7 上位机软件接收数据界面

[9]使用网络调试助手发送数据，观察实验现象。

如图 2-8-6 所示，网络调试助手发送数据 11，上位机软件可以将 WIFI 收到的数据 11 显示出来。

参考程序

参考程序的工程名为“WIFI.uvproj”，在本实验文件目录下“WIFI”文件夹下。

实验小结

通过本节学习，了解 AT 指令，学习 WiFi 的 AT 指令。掌握使用 RobotTTP 本体上 WiFi 模块进行通信。

思考题

编写程序，完成 2 个 RobotTTP 进行 WIFI 通信。

2.9 陀螺仪

实验目的

了解陀螺仪原理及其相关知识。

学习掌握基于 STM32 的陀螺仪使用方法。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件

实验预习

陀螺仪简介：

绕一个支点高速转动的刚体称为陀螺(top)。通常所说的陀螺是特指对称陀螺，它是一个质量均匀分布的、具有轴对称形状的刚体，其几何对称轴就是它的自转轴。陀螺仪的原理就是，一个旋转物体的旋转轴所指的方向在不受外力影响时，是不会改变的。人们根据这个道理，用它来保持方向，制造出来的东西就叫陀螺仪。陀螺仪在工作时要给它一个力，使它快速旋转起来，一般能达到每分钟几十万转，可以工作很长时间。然后用多种方法读取轴所指示的方向，并自动将数据信号传给控制系统。

传统的惯性陀螺仪主要是指机械式的陀螺仪，机械式的陀螺仪对工艺结构的要求很高，结构复杂，它的精度受到了很多方面的制约。现在的陀螺仪分为，液浮陀螺仪，气浮陀螺仪，磁悬浮陀螺仪，静电陀螺仪，微机械（MEMS)陀螺仪，光纤陀螺仪，激光陀螺仪，振动陀螺仪。它们都是广义上的陀螺仪，是根据近代物理学原理制成的具有陀螺效应的传感器。因其无活动部件，称为固态陀螺仪，具有结构紧凑，灵敏度高，工作可靠等等优点，完全取代了机械式的传统的陀螺仪，成为现代导航仪器中的关键部件。

MPU6050 简介：

MPU-6000（6050）为全球首例整合性 6 轴运动处理组件，相较于多组件方案，免除了组合陀螺仪与加速器时之轴间差的问题，减少了大量的包装空间。MPU-6000（6050）整合了 3 轴陀螺仪、3 轴加速器，并含可藉由第二个 I2C 端口连接其他厂牌之加速器、磁力传感器、或其他传感器的数位运动处理(DMP: Digital Motion Processor)硬件加速引擎，由主要 I2C 端口以单一数据流的形式，向应用端输出完整的 9 轴融合演算技术 InvenSense 的运动处理资料库，可处理运动感测的复杂数据，降低了运动处理运算对操作系统的负荷，并为应用开发提供架构化的 API。MPU-6000（6050）的角速度全格感测范围为±250、±500、±1000 与±2000°/sec (dps)，可准确追縱快速与慢速动作，并且，用户可程式控制的加速器全格感测范围为±2g、±4g、±8g 与±16g。产品传输可透过最高至 400kHz 的 IC 或最高达 20MHz 的 SPI（MPU-6050 没有 SPI）。MPU-6000 可在不同电压下工作，VDD 供电电压介为 2.5V±5%、3.0V±5%或 3.3V±

5%，逻辑接口 VVDIO 供电为 1.8V± 5%（MPU6000 仅用 VDD）。MPU-6000 的包装尺寸 4×4×0.9mm(QFN)，在业界是革命性的尺寸。其他的特征包含内建的温度感测器、包含在运作环境中仅有±1%变动的振荡器。

实验内容

[1]新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

[3]配置 GPIO 口工作模式。

[4]配置串口模块。

[5]编写 IIC 初始化函数。

[6]编写主函数。

[7]使用 STM32 ST-LINK Utility 软件下载程序。

[8]观察实验现象。

实验步骤

[1]新建一个 Keil 工程。

新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。按照如下步骤进行：

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能时钟：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOE|RCC_APB2Periph_AFIO|RCC_APB2Periph_USART1, ENABLE);//

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3]配置 GPIO 口的工作模式。按照如下步骤进行：

1)恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();

2)配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

3)配置端口：

本实验用到的串口 1 发射引脚是 PA9，配置如下：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;//PA9---TxD1
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;//复用推挽输出
GPIO_Init(GPIOA, &GPIO_InitStructure);//对选中管脚初始化
```

[4]配置串口模块。

```
void USART_Config(void)
{
 USART_InitTypeDef USART_InitStructure;
 USART_DeInit(USART1);//复位串口 1 寄存器
 USART_InitStructure.USART_BaudRate = 115200;//波特率 115200bps
 USART_InitStructure.USART_WordLength = USART_WordLength_8b; //数据位
 USART_InitStructure.USART_StopBits = USART_StopBits_1; //停止位 1 位
 USART_InitStructure.USART_Parity = USART_Parity_No; //无校验位
 USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;  //无硬件流控
 USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;  //收
 USART_Init(USART1, &USART_InitStructure);//配置串口参数
 USART_ITConfig(USART1, USART_IT_RXNE, ENABLE); //使能接收中断
 USART_ClearFlag(USART1, USART_FLAG_TC );//清除发送完成标志位
 USART_Cmd(USART1, ENABLE);//使能串口
}
```

[5]编写 IIC 初始化函数。

```
void init_IIC2(void)   //标准模式 200KHz
{
    GPIO_InitTypeDef GPIO_Struct;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_I2C,ENABLE);
    GPIO_Struct.GPIO_Pin=SCL | SDA;
```

```
    GPIO_Struct.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_Struct.GPIO_Mode=GPIO_Mode_Out_PP;
    GPIO_Init(GPIO_I2C,&GPIO_Struct);
    SDA_H;
    delay(20);
    SCL_H;
    delay(20);
}
```

[6]编写主函数。

第一步：包含头文件：

```
#include "stm32f10x.h"
```

第二步：声明程序中的子函数：

```
void RCC_Configuration(void);//RCC 子程序
void GPIO_Configuration(void);//IO 初始化程序
void NVIC_Configuration(void);//嵌套向量中断控制器配置
void USART_Config(void);
int fputc(int ch,FILE* f);
void TIM2_Int_Init(u16 arr,u16 psc);//tim2 初始化
```

第三步：编写主函数函数体：

```
int main(void)
{
    RCC_Configuration();//RCC 配置
    GPIO_Configuration();//GPIO 配置
    TIM2_Int_Init(999,359);//tim2 初始化
    USART_Config();
    NVIC_Configuration();//嵌套向量中断控制器配置
    init_IIC2();
    delay(0xffffff);//延时
    Delay(1000000);
    if(MPU6050_Config())//MPU6050 配置
            printf("MPU6050_Config_Success");
    while(1)
    {
            if(tim2_count>=20)//20*5ms=100ms
            {
                    tim2_count=0;
                    MPU6050Update();
                    printf("G_Roll=%4f;",G_Roll);
                    printf("G_Pitch=%4f;",G_Pitch);
                    printf("G_Yaw=%4f;",G_Yaw);
                    printf("G_Gyro_X=%4f;",G_Gyro_X);
```

```
                printf("G_Gyro_Y=%4f;",G_Gyro_Y);
                printf("G_Gyro_Z=%4f\r\n",G_Gyro_Z);
            }
        }
    }
```

[7]使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 MPU6050.hex 文件烧写到 RobotTTP。

[8]观察实验现象。

观察实验现象。

参考程序

参考程序的工程名为“MPU6050.uvproj”，在本实验文件目录下“MPU6050”文件夹下。

实验小结

通过本节学习，了解陀螺仪原理及相关知识，能够利用 STM32 通过 RobotTTP 本体上的陀螺仪测量倾角等。

思考题

对本实验的程序进行修改，编写相应的程序，能够实现加速度的测量。

2.10 加速度计

实验目的

了解三轴加速度计原理及其相关知识。

学习掌握基于 STM32 的三轴加速度计使用方法。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件

实验预习

三轴加速度计（ADXL345）简介：

ADXL345 是一款小而薄的超低功耗 3 轴加速度计，分辨率高(13 位)，测量范围达 ± 16g。数字输出数据为 16 位二进制补码格式，可通过 SPI(3 线或 4 线)或 I2C 数字接口访问。ADXL345 非常适合移动设备应用。它可以在倾斜检测应用中测量静态重力加速度，还可以测量运动或冲击导致的动态加速度。其高分辨率(3.9mg/LSB)，能够测量不到 1.0°的倾斜角度变化。该器件提供多种特殊检测功能。活动和非活动检测功能通过比较任意轴上的加速度与用户设置的阈值来检测有无运动发生。敲击检测功能可以检测任意方向的单振和双振动作。自由落体检测功能可以检测器件是否正在掉落。这些功能可以独立映射到两个中断输出引脚中的一个。正在申请专利的集成式存储器管理系统采用一个 32 级先进先出(FIFO)缓冲器，可用于存储数据，从而将主机处理器负荷降至最低，并降低整体系统功耗。

实验内容

[1]新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

[3]配置 GPIO 口工作模式。

[4]配置串口模块。

[5]编写 IIC 初始化函数。

[6]编写主函数。

[7]使用 STM32 ST-LINK Utility 软件下载程序。

[8]观察实验现象。

实验步骤

[1]新建一个 Keil 工程。

新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。按照如下步骤进行：

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能时钟：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOE|RCC_APB2Periph_AFIO|RCC_APB2Periph_USART1, ENABLE);//RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3]配置 GPIO 口的工作模式。按照如下步骤进行：

1)恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();

2)配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

3)配置端口：

本实验用到的串口 1 发射引脚是 PA9，配置如下：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;//PA9---TxD1
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;//复用推挽输出
GPIO_Init(GPIOA, &GPIO_InitStructure);//对选中管脚初始化
```

[4]配置串口模块。

```
void USART_Config(void)
```

```
{
 USART_InitTypeDef USART_InitStructure;
 USART_DeInit(USART1);//复位串口 1 寄存器
 USART_InitStructure.USART_BaudRate = 115200;//波特率 115200bps
 USART_InitStructure.USART_WordLength = USART_WordLength_8b; //数据位
 USART_InitStructure.USART_StopBits = USART_StopBits_1; //停止位 1 位
 USART_InitStructure.USART_Parity = USART_Parity_No; //无校验位
 USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowContr
 ol_None;  //无硬件流控
 USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;  //收
 USART_Init(USART1, &USART_InitStructure);//配置串口参数
 USART_ITConfig(USART1, USART_IT_RXNE, ENABLE); //使能接收中断
 USART_ClearFlag(USART1, USART_FLAG_TC );//清除发送完成标志位
 USART_Cmd(USART1, ENABLE);//使能串口
}
```

[5]编写 IIC 初始化函数。

```
void init_IIC2(void)  //标准模式 200KHz
{
    GPIO_InitTypeDef GPIO_Struct;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_I2C,ENABLE);
    GPIO_Struct.GPIO_Pin=SCL | SDA;
    GPIO_Struct.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_Struct.GPIO_Mode=GPIO_Mode_Out_PP;
    GPIO_Init(GPIO_I2C,&GPIO_Struct);
    SDA_H;
    delay(20);
    SCL_H;
    delay(20);
}
```

[6]编写主函数。

第一步：包含头文件：

```
#include "stm32f10x.h"
```

第二步：声明程序中的子函数：

```
void RCC_Configuration(void);//RCC 子程序
void GPIO_Configuration(void);//IO 初始化程序
void NVIC_Configuration(void);//嵌套向量中断控制器配置
void USART_Config(void);
int fputc(int ch,FILE* f);
void TIM2_Int_Init(u16 arr,u16 psc);//tim2 初始化
```

第三步：编写主函数函数体：

```
int main(void)
{
    RCC_Configuration();//RCC 配置
    GPIO_Configuration();//GPIO 配置
    TIM2_Int_Init(999,359);//tim2 初始化
    USART_Config();
    NVIC_Configuration();//嵌套向量中断控制器配置
    init_IIC2();
    delay(0xffffff);//延时
    Delay(1000000);
    if(ADXL345_Init()==0)
    {
        printf("initial success"); //initial success
    }
    else
    {
        printf("fail");//fail
    }
    while(1)
    {
        if(tim2_count>=20)//20*5ms=100ms
        {
            tim2_count=0;
            angle_deal();//角度处理
        }
    }
}
```

[7]使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 ADXL345.hex 文件烧写到 RobotTTP。

[8]观察实验现象。

观察实验现象。

参考程序

参考程序的工程名为“ADXL345.uvproj”，在本实验文件目录下“ADXL345”文件夹下。

实验小结

通过本节学习，了解三轴加速度计原理及相关知识，能够利用 STM32 通过 RobotTTP 本体上的三轴加速度计测量加速度等。

思考题

对本实验的程序进行修改，编写相应的程序，能够实现物体姿态的判断。

2.11 数字罗盘

实验目的

了解数字罗盘原理及其相关知识。

学习掌握基于 STM32 的数字罗盘使用方法。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件

实验预习

数字罗盘（HMC5883L）简介：

霍尼韦尔 HMC5883L 是一种表面贴装的高集成模块，并带有数字接口的弱磁传感器芯片，应用于低成本罗盘和磁场检测领域。HMC5883L 包括最先进的高分辨率 HMC118X 系列磁阻传感器，并附带霍尼韦尔专利的集成电路包括放大器、自动消磁驱动器、偏差校准、能使罗盘精度控制在 1°~2°的 12 位模数转换器.简易的 I2C 系列总线接口。HMC5883L 是采用无铅表面封装技术，带有 16 引脚，尺寸为 3.0×3.0×0.9mm。HMC5883L 的所应用领域有手机、笔记本电脑、消费类电子、汽车导航系统和个人导航系统。

HMC5883L 采用霍尼韦尔各向异性磁阻(AMR)技术，该技术的优点是其他磁传感器技术所无法企及。这些各向异性传感器具有在轴向高灵敏度和线性高精度的特点.传感器带有的对于正交轴低敏感行的固相结构能用于测量地球磁场的方向和大小，其测量范围从毫高斯到 8 高斯(gauss)。霍尼韦尔的磁传感器在低磁场传感器行业中是灵敏度最高和可靠性最好的传感器。

实验内容

[1]新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

[3]配置 GPIO 口工作模式。

[4]配置串口模块。

[5]编写 IIC 初始化函数。

[6]编写主函数。

[7]使用 STM32 ST-LINK Utility 软件下载程序。

[8]观察实验现象。

实验步骤

[1]新建一个 Keil 工程。

新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。按照如下步骤进行：

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能时钟：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOE|RCC_APB2Periph_AFIO|RCC_APB2Periph_USART1, ENABLE);//RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3]配置 GPIO 口的工作模式。按照如下步骤进行：

1)恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();

2)配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

3)配置端口：

本实验用到的串口 1 发射引脚是 PA9，配置如下：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;//PA9---TxD1
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;//复用推挽输出
GPIO_Init(GPIOA, &GPIO_InitStructure);//对选中管脚初始化
```

[4]配置串口模块。

```
void USART_Config(void)
{
 USART_InitTypeDef USART_InitStructure;
 USART_DeInit(USART1);//复位串口 1 寄存器
 USART_InitStructure.USART_BaudRate = 115200;//波特率 115200bps
 USART_InitStructure.USART_WordLength = USART_WordLength_8b; //数据位
 USART_InitStructure.USART_StopBits = USART_StopBits_1; //停止位 1 位
 USART_InitStructure.USART_Parity = USART_Parity_No; //无校验位
 USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowContr
 ol_None;  //无硬件流控
 USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;  //收
 USART_Init(USART1, &USART_InitStructure);//配置串口参数
 USART_ITConfig(USART1, USART_IT_RXNE, ENABLE); //使能接收中断
 USART_ClearFlag(USART1, USART_FLAG_TC );//清除发送完成标志位
 USART_Cmd(USART1, ENABLE);//使能串口
}
```

[5]编写 IIC 初始化函数。

```
void init_IIC2(void)  //标准模式 200KHz
{
    GPIO_InitTypeDef GPIO_Struct;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_I2C,ENABLE);
    GPIO_Struct.GPIO_Pin=SCL | SDA;
    GPIO_Struct.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_Struct.GPIO_Mode=GPIO_Mode_Out_PP;
    GPIO_Init(GPIO_I2C,&GPIO_Struct);
    SDA_H;
    delay(20);
    SCL_H;
    delay(20);
}
```

[6]编写主函数。

第一步：包含头文件：

```
#include "stm32f10x.h"
```

第二步：声明程序中的子函数：

```
void RCC_Configuration(void);//RCC 子程序
void GPIO_Configuration(void);//IO 初始化程序
void NVIC_Configuration(void);//嵌套向量中断控制器配置
void USART_Config(void);
```

```
int fputc(int ch,FILE* f);
void TIM2_Int_Init(u16 arr,u16 psc);//tim2 初始化
```

第三步：编写主函数函数体：

```
int main(void)
{
    u16 angle;
    RCC_Configuration();//RCC 配置
    GPIO_Configuration();//GPIO 配置
    TIM2_Int_Init(999,359);//tim2 初始化
    USART_Config();
    NVIC_Configuration();//嵌套向量中断控制器配置
    init_IIC2();
    HMC5883_Init();
    delay(0xffffff);//延时
    Delay(1000000);
    while(1)
    {
            if(tim2_count>=20)//20*5ms=100ms
            {
                    tim2_count=0;
                    angle=HMC5883_RD_XYZ();
                    printf("angle=%d\r\n",angle);
            }
    }
}
```

[7]使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 HMC5883.hex 文件烧写到 RobotTTP。

[8]观察实验现象。

观察实验现象。

参考程序

参考程序的工程名为“HMC5883.uvproj”，在本实验文件目录下“HMC5883”文件夹下。

实验小结

通过本节学习，了解数字罗盘原理及相关知识，能够利用 STM32 通过 RobotTTP 本体上的数字罗盘测量地磁角度等。

思考题

对本实验的程序进行修改，编写相应的程序，能够实现指南针的功能。

2.12 语音识别及 MP3 播放

实验目的

了解语音识别及其相关知识。
了解 MP3 播放器及其相关知识。
熟悉和掌握语音识别传感器 LD3320A 在 RobotTTP 的中使用。
学习掌握基于 STM32 的 MP3 播放方法。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）
软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件，语音调试助手

实验预习

MP3 简介：

MP3 是 Internet 上最流行的音乐格式，最早起源于 1987 年德国一家公司的 EU147 数字传输计划，它利用 MPEGAudioLayer3 的技术，将声音文件用 1：12 左右的压缩率压缩，变成容量较小的音乐文件，使传输和储存更为便捷，更利于互联网用户在网上试听或下载到个人计算机。

同时，MP3 格式音乐的流行也带动了 MP3 专用播放装置的出现，并在近年来得到快速发展。目前市场上流行的 MP3 设备从功能和性能上已经远远超出了原来"播放器"的范畴，逐步发展成为集音频播放（包括 MP3 之外的音乐格式）、录音复读、文本阅读、移动存储、FM 收音等功能为一体的多媒体掌上设备。有的高端 MP3 设备甚至还集成了音频编辑处理、电影播放等功能。

MP3 的优点有许多，主要有三点：一是由于大大压缩了文件的体积，所以相同的空间能存储更多的信息；二是由于没有机械元件，全部是电子元件，所以不存在防震问题，更加适合运动时欣赏音乐；三是可以随心所欲编辑自己喜爱的歌。

有一利便有一弊，MP3 也有一些缺点。MP3 音频压缩技术是一种失真压缩，因为人耳只能听到一定频段内的声音，而其他更高或更低频率的声音对人耳是没有用处的，所以 MP3 技术就把这部分声音去掉了，从而使得文件体积大为缩小。虽然听上去 MP3 音乐仍旧具有接近 CD 的音质，但毕竟要比 CD 稍逊一些。

语音识别的概念：

与机器进行语音交流，让机器明白你说什么，这是人们长期以来梦寐以求的事

情。中国物联网校企联盟形象得把语音识别比作为“机器的听觉系统”。语音识别技术就是让机器通过识别和理解过程把语音信号转变为相应的文本或命令的高技术。语音识别技术主要包括特征提取技术、模式匹配准则及模型训练技术三个方面。语音识别技术车联网也得到了充分的应用，例如在翼卡车联网中，只需按一键通客服人员口述即可设置目的地直接导航，安全、便捷。

语音识别芯片 LD3320A 简介：

LD3320 芯片是一款“语音识别”专用芯片，由 ICRoute 公司设计生产。该芯片集成了语音识别处理器和一些外部电路，包括 AD、DA 转换器、麦克风接口、声音输出接口等。本芯片在设计上注重节能与高效，不需要外接任何的辅助芯片如 Flash、RAM 等，直接集成在现有的产品中即可以实现语音识别/声控/人机对话功能。并且识别的关键词语列表是可以任意动态编辑的。

LD3320 芯片支持 MP3 播放功能，无需外围辅助器件，主控 MCU 将 MP3 数据依次送入 LD3320 芯片内部就可以从芯片的相应 PIN 输出声音。产品设计可以选择从立体声的耳机或者单声道喇叭来获得声音输出。支持 MPEG1(ISO/IEC11172-3), MPEG2(ISO/IEC13818-3)和 MPEG2.5layer3 等格式。

非特定语音识别简介：

非特定人系统是用很广泛的说话人语音来训练识别系统模型，在保证有足够的数据来精确刻画语音单元的各种复杂的时变特性和协同发音的同时，也可忽略说话人之间的差异，从而降低了系统对于单个的说话人建模的精度，因此相对特定人语音识别，非特定人语音识别则可用于不同的用户，这种识别系统的通用性好、应用面广，具有更广阔的研究前景，但难度也较大。

声音提取过程流程图：如图 2-12-1 所示，为声音提取过程流程图。

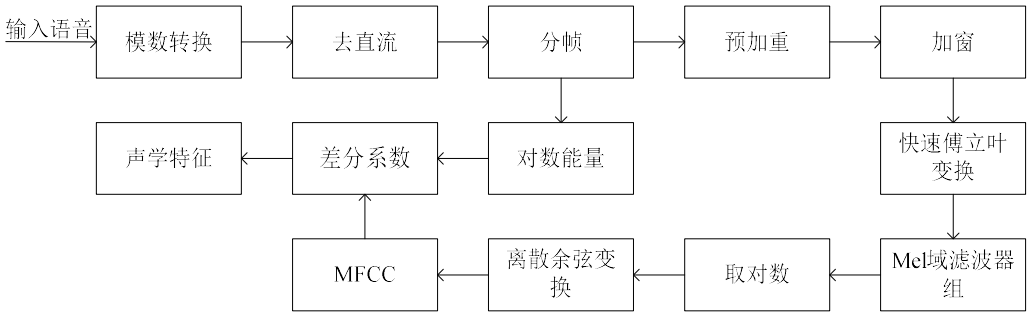


图 2-12-1 声音提取过程流程图

如何利用 TTCDS-IMR 软件下载识别语句：

如图 2-12-2 所示，为下载识别语句界面。用户可以在语句输入栏输入想要识别的词语、短句，例如“ni hao”、“ni shi shui”等，每个字之间加 1 个空格。点击右边的

叉号可以清除相应的语句内容，点击右下角的“写入”按钮，可以将这些语句写入语音板。拉动左上角麦克风灵敏度后面的滑块，可以选择麦克风的灵敏度，点击后面的写入按钮，可以将麦克风灵敏度写入语音板。

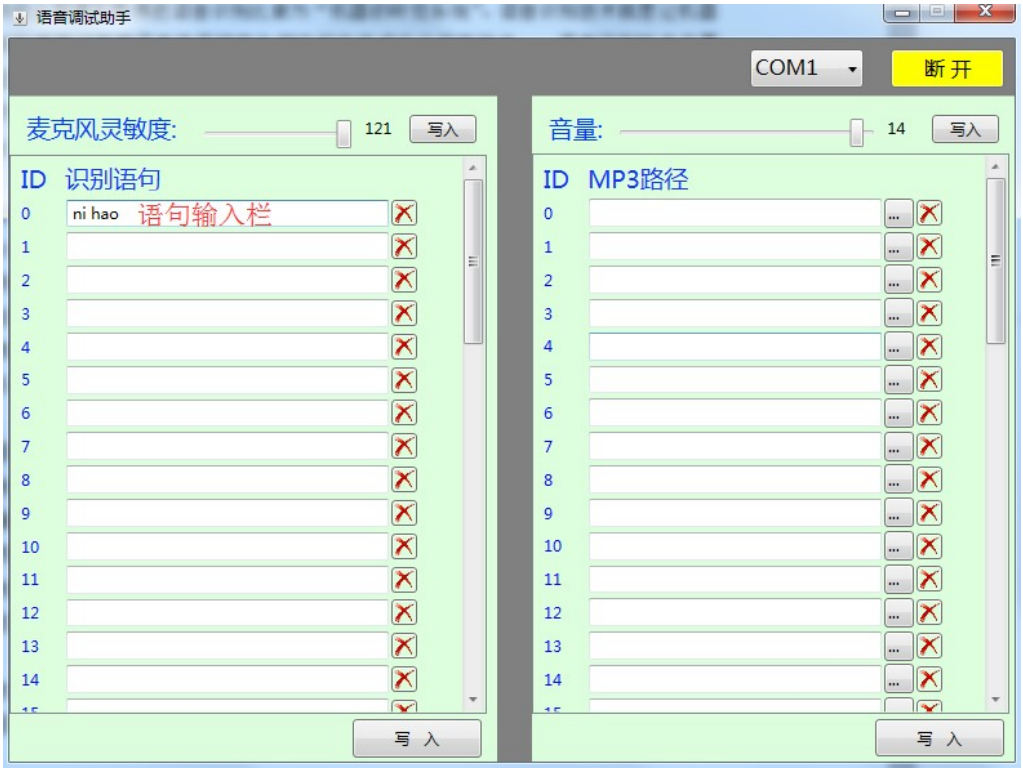


图 2-12-2 TTCDS-IMR 软件下载识别语句

TTCDS-IMR 软件下载 MP3 文件使用说明：

如图 2-12-1 所示，为 TTCDS-IMR 软件下载 MP3 文件界面。其中 ID 一栏是表示 MP3 文件下载到语音板后，在语音板内的存储序号，需要播放的 MP3 文件时，需要对应相应的 ID 号。MP3 路径一栏表示加载 MP3 文件的路径，后面的加载按钮是加载相应的 MP3 文件，后面的红色叉号是 MP3 文件的删除键。点击右下角的写入按钮，可以将加载的全部 MP3 文件写入语音板。拉动音量后面的滑块，可以选择音量的大小，点击右上角的写入可以将音量写入语音板。

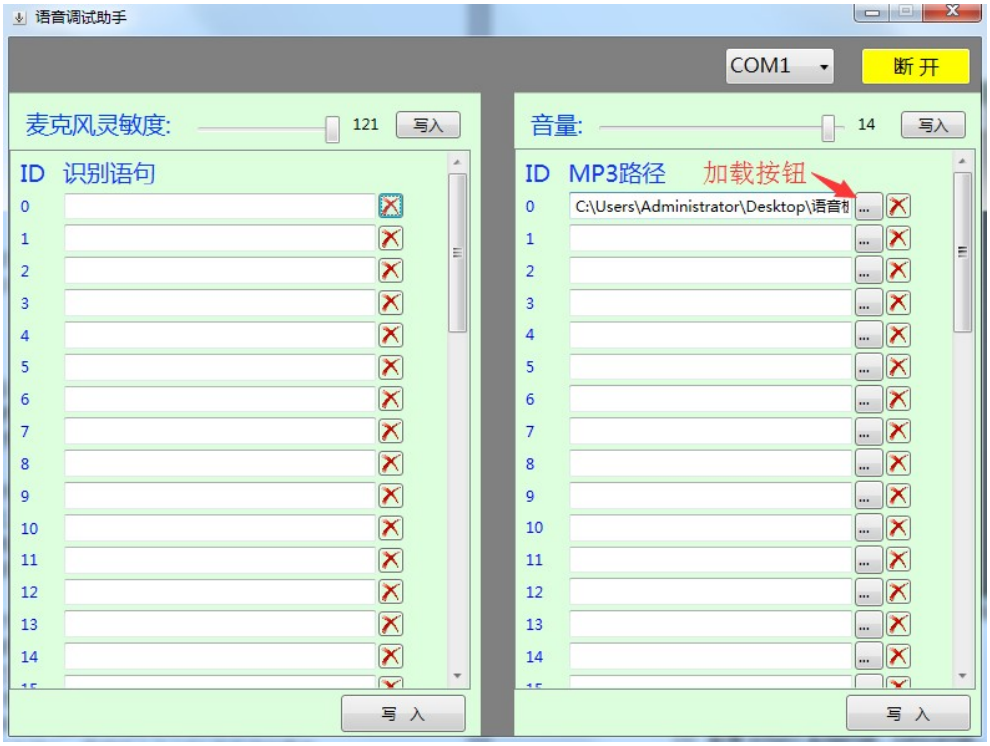


图 2-12-3 语音调试助手软件界面

实验内容

[1]使用 TTCDS-IMR 软件下载需要识别的语句。

[2]新建一个 Keil 工程。

[3]配置 STM32 系统时钟、GPIO 时钟。

[4]配置 GPIO 口工作模式。

[5]配置串口模块。

[6]编写 USART1 串口中断处理函数。

[7]编写主函数。

[8]使用 STM32 ST-LINK Utility 软件下载程序。

[9]观察实验现象。

实验步骤

[1]使用 TTCDS-IMR 软件下载需要识别的语句。

如图 2-12-2 所示，在 ID 为 0 的语句输入栏中输入“ni hao”，在 ID 为 1 的语句输入栏中输入“ni shi shui”。然后点击右下角的“写入”按钮，将识别语句写入语音板。

[2]新建一个 Keil 工程。

新建一个 Keil 工程。

[3]配置 STM32 系统时钟、GPIO 时钟。按照如下步骤进行：

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能时钟：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOE|RCC_APB2Periph_GPIOF|RCC_APB2Periph_GPIOG|RCC_APB2Periph_USART1|RCC_APB2Periph_AFIO, ENABLE);//使能 RCC_APB1PeriphClockCmd(RCC_APB1Periph_UART4, ENABLE);

库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[4]配置 GPIO 口的工作模式。按照如下步骤进行：

1)恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();

2)配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

3)配置各端口：

本实验用到的串口 1 发射引脚是 PA9，配置如下：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;//PA9---TxD1
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;//复用推挽输出
GPIO_Init(GPIOA, &GPIO_InitStructure);//对选中管脚初始化
```

[5]配置串口模块。

```
void USART_Config(void)
```

```
{
 USART_InitTypeDef USART_InitStructure;
 USART_DeInit(USART1);//复位串口 1 寄存器
 USART_InitStructure.USART_BaudRate = 9600;//波特率 9600bps
 USART_InitStructure.USART_WordLength = USART_WordLength_8b; //数据位 8
 USART_InitStructure.USART_StopBits = USART_StopBits_1; //停止位 1 位
 USART_InitStructure.USART_Parity = USART_Parity_No; //无校验位
 USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowContr
 ol_None;   //无硬件流控
 USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;   //收
 USART_Init(USART1, &USART_InitStructure);//配置串口参数
 USART_ITConfig(USART1, USART_IT_RXNE, ENABLE); //使能接收中断
 USART_ClearFlag(USART1, USART_FLAG_TC );//清除发送完成标志位
 USART_Cmd(USART1, ENABLE);//使能串口
}
```

[6]编写 USART1 串口中断处理函数。

```
void USART1_IRQHandler(void)
{
    u8 usart1_data;
    usart1_data=USART_ReceiveData(USART1);
    while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
    USART_SendData(USART1,usart1_data);//发送回复帧
    while(USART_GetFlagStatus(UART4,USART_FLAG_TXE)==RESET);
    USART_SendData(UART4,0xFE);//发送回复帧
    while(USART_GetFlagStatus(UART4,USART_FLAG_TXE)==RESET);
    USART_SendData(UART4,0xFE);//发送回复帧
    while(USART_GetFlagStatus(UART4,USART_FLAG_TXE)==RESET);
    USART_SendData(UART4,0xFE);//发送回复帧
    while(USART_GetFlagStatus(UART4,USART_FLAG_TXE)==RESET);
    USART_SendData(UART4,0xFE);//发送回复帧
    while(USART_GetFlagStatus(UART4,USART_FLAG_TXE)==RESET);
    USART_SendData(UART4,0xEA);//发送回复帧
    while(USART_GetFlagStatus(UART4,USART_FLAG_TXE)==RESET);
    USART_SendData(UART4,0xA2);//发送回复帧
    while(USART_GetFlagStatus(UART4,USART_FLAG_TXE)==RESET);
    USART_SendData(UART4,0x00);//发送回复帧
    while(USART_GetFlagStatus(UART4,USART_FLAG_TXE)==RESET);
    USART_SendData(UART4,usart1_data);//发送回复帧
    while(USART_GetFlagStatus(UART4,USART_FLAG_TXE)==RESET);
    USART_SendData(UART4,0xD3);//发送回复帧
```

```
}
```

[7]编写主函数。

第一步：包含头文件：

```
#include "stm32f10x.h"
```

第二步：声明程序中的子函数：

```
void delay(u32 i);//延时子程序
void RCC_Configuration(void);//RCC 子程序
void GPIO_FuWei(void);//GPIO 复位及 SW-DP 配置
void NVIC_Configuration(void);
void USART_Config(void);
void USART_Config_4(void);
```

第三步：编写主函数函数体：

```
int main(void)
{
    RCC_Configuration();//RCC 配置
    GPIO_FuWei();//GPIO 复位及 SW-DP 配置
    USART_Config();
    NVIC_Configuration();
    delay(8000000);//延时
    YuYinDianYuan(ON);//打开语音电路电源
    USART_Config_4();
    while(1)
    {
            Led(ON);//点亮 LED 灯
            delay(2000000);//延时
            Led(OFF);//关闭 LED 灯
            delay(2000000);//延时
    }
}
```

[8]使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 YuYinShiBieJiBoFang.hex 文件烧写到 RobotTTP。

[9]观察实验现象。

观察实验现象。

参考程序

参考程序的工程名为“YuYinShiBieJiBoFang.uvproj”，在本实验文件目录下“YuYinShiBieJiBoFang”文件夹下。

实验小结

通过本小节的学习，了解语音识别的原理及 MP3 播放的相关知识，并掌握 RobotTTP 和 TTCDS-IMR 软件做语音识别实验及 MP3 播放实验的方法。

思考题

对本实验的程序进行修改，编写相应的程序，能够实现声控播放指定的 MP3 文件。

2.13 智能外控 LED

实验目的

了解智能外控 LED 原理。

掌握控制智能外控 LED 的能力。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件

实验预习

智能外控 LED（WS2812B）介绍：

WS2812 是一个集控制电路与发光电路于一体的智能外控 LED 光源。其外型与一个 5050LED 灯珠相同，每个元件即为一个像素点。像素点内部包含了智能数字接口数据锁存信号整形放大驱动电路，还包含有高精度的内部振荡器和 12V 高压可编程定电流控制部分，有效保证了像素点光的颜色高度一致。数据协议采用单线归零码的通讯方式，像素点在上电复位以后，DIN 端接受从控制器传输过来的数据，首先送过来的 24bit 数据被第一个像素点提取后，送到像素点内部的数据锁存器，剩余的数据经过内部整形处理电路整形放大后通过 DO 端口开始转发输出给下一个级联的像素点，每经过一个像素点的传输，信号减少 24bit。像素点采用自动整形转发技术，使得该像素点的级联个数不受信号传送的限制，仅仅受限信号传输速度要求。LED 具有低电压驱动，环保节能，亮度高，散射角度大，一致性好，超低功率，超长寿命等优点。将控制电路集成于 LED 上面，电路变得更加简单，体积小，安装更加简便。

实验内容

[1] 新建一个 Keil 工程。

[2] 配置 STM32 系统时钟、GPIO 时钟。

[3] 配置 GPIO 口工作模式。

[4] 编写颜色发送函数。

[5] 编写主函数。

[6] 使用 STM32 ST-LINK Utility 软件下载程序。

[7] 观察实验现象。

实验步骤

[1] 新建一个 Keil 工程。

新建一个 Keil 工程。

[2] 配置 STM32 系统时钟、GPIO 时钟。按照如下步骤进行：

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、GPIOF、GPIOG、USART1、AFIO 时钟：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOE|RCC_APB2Periph_GPIOF|RCC_APB2Periph_GPIOG|RCC_APB2Periph_USART1|RCC_APB2Periph_AFIO, ENABLE);

库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3] 配置 GPIO 口的工作模式。按照如下步骤进行：

1)恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();

2)配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

3)配置端口：

考虑到在本实验中采用 PA1 作为智能外控 LED 的控制 IO，所以设置为推挽输出。

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;//PA1---DATALED
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//模式通用推挽输出
GPIO_Init(GPIOA, &GPIO_InitStructure);//对选中管脚初始化
GPIO_ResetBits(GPIOA, GPIO_Pin_1);//DATALED 配置为输出低电平
```

详细的库函数使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 120-133”。

[4] 编写颜色发送函数。

```
void SendData(u32 *p)
{
 u32 i,j=0;
 ZongXian_L;
 delay(500);
 while(j < 16)//16 个数
 {
   for(i=0;i<24;i++)
   {
   if(*p&(1<<(23-i)))
   {
   ZongXian_H;  //0.85us_H,0.4us_L
   __nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
   __nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
   __nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
   __nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
   __nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
   ZongXian_L;
   }
  else
   {
   ZongXian_H;  //0.4us_H,0.85us_L
   __nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
   ZongXian_L;
  __nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
  __nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
  __nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
   }
 }//for
 p++;j++;
 }//while
}
```

[5] 编写主函数。

第一步：包含头文件：

```
#include "stm32f10x.h"
```

第二步：声明程序中的子函数：

```
void delay(u32 i);//延时子程序
void RCC_Configuration(void);//RCC 子程序
```

```
void GPIO_Configuration(void);//IO 初始化程序
void SendData(u32 *p);//发送灯的颜色
```

第三步：编写主函数函数体：

```
int main(void)
{
    u8 i;
    RCC_Configuration();//RCC 配置
    GPIO_Configuration();//GPIO 配置
    for(i=0;i<16;i++)
    {colordata[i]=0x000010;}//0x101010//GRB//绿红蓝
    SendData(colordata);
    while(1)
    {
            for(i=0;i<16;i++)
            {
                    if(colordata[i]<0x100)
                            colordata[i]++;
                    else if(colordata[i]<0x10000)
                            colordata[i] += 0x100;
                    else if(colordata[i]<0x1000000)
                            colordata[i] += 0x10000;
                    else
                            colordata[i] =0;
            }
            SendData(colordata);
            delay(80000);
    }
}
```

[6] 使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 CaiDeng.hex 文件烧写到 RobotTTP。

[7] 观察实验现象。

观察智能外控 LED 颜色的变化。

参考程序

参考程序的工程名为“CaiDeng.uvproj”，在本实验文件目录下“CaiDeng”文件夹下。

实验小结

通过本节学习，了解智能外控 LED 原理。掌握控制智能外控 LED 颜色变化的能力。

思考题

修改程序，制作一个花式跑马灯。

2.14 显示屏

实验目的

了解 TFT 相关知识及其在嵌入式系统中应用。

学习掌握 RobotTTP 本体上 TFT（ThinFilmTransistor，是指薄膜晶体管）液晶屏的使用方法。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件

实验预习

TFT-LCD 介绍：

TFT(Thin Film Transistor)LCD 即薄膜场效应晶体管 LCD，是有源矩阵类型液晶显示器(AM-LCD)中的一种。液晶平板显示器，特别 TFT-LCD，是目前唯一在亮度、对比度、功耗、寿命、体积和重量等综合性能上全面赶上和超过 CRT 的显示器件，它的性能优良、大规模生产特性好，自动化程度高，原材料成本低廉，发展空间广阔，将迅速成为新世纪的主流产品，是 21 世纪全球经济增长的一个亮点。和 TN 技术不同的是，TFT 的显示采用“背透式”照射方式——假想的光源路径不是像 TN 液晶那样从上至下，而是从下向上。这样的作法是在液晶的背部设置特殊光管，光源照射时通过下偏光板向上透出。由于上下夹层的电极改成 FET 电极和共通电极，在 FET 电极导通时，液晶分子的表现也会发生改变，可以通过遮光和透光来达到显示的目的，响应时间大大提高到 80ms 左右。因其具有比 TN-LCD 更高的对比度和更丰富的色彩，荧屏更新频率也更快，故 TFT 俗称“真彩”。相对于 DSTN 而言，TFT-LCD 的主要特点是为每个像素配置一个半导体开关器件。由于每个像素都可以通过点脉冲直接控制。因而每个节点都相对独立，并可以进行连续控制。这样的设计方法不仅提高了显示屏的反应速度，同时也可以精确控制显示灰度，这就是 TFT 色彩较 DSTN 更为逼真的原因。

TFT 屏与微控制器的连接示意图如图 2-14-1 所示：

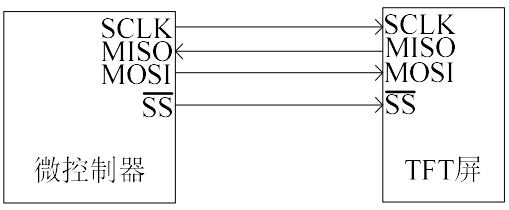


图 2-14-1 TFT 屏与微控制器的连接示意图

TFT 屏与微控制器的通信也是通过 SPI 接口实现的，上一节我们已经详细地介绍了 SPI 通信接口的原理及使用方法，这里就不在赘述了。

TFT 屏显示原理：

液晶是一种有机化合物，是液体。但是其分子具有固体水晶石分子一样的光学特性，我们知道水晶石对光有优秀的透射性能。那么液晶的分子对光也有优秀的透射性能。同时液晶的分子对电场又极其的敏感（类似于铁分子对磁场敏感的现象），液晶分子周边的电场发生变化，液晶的分子会随其变化产生扭曲，通过液晶分子的扭曲可以使通过的光线受到控制（通过、阻断）从而形成图像。液晶显示器就是利用液晶的这两项重要特性使图像的显示成为现实。

液晶显示屏是把图像信号先由一个时序控制电路转化为；“水平”（行）和“垂直”（列）的驱动信号，再加到液晶屏的矩阵电路上，经过“寻址”把在液晶屏上产生能影响光线通过的“点”排列成图像，再在背光的作用下形成明亮的图像。液晶的这种驱动方式称为“矩阵”驱动方式，液晶屏的驱动电路有列驱动和行驱动，由时序控制电路（俗称 T-CON 电路）把数字的图像信号转化成列、行驱动信号。列驱动信号是反映图像内容的像素信息，行驱动信号是驱使上下扫描的位移脉冲，一行一行的驱使列信号的显示，一行线上的列信号是同时显示的，这一点和 CRT 上一行像素信号是逐个显示的不同。

液晶的本身并不能发光，它只能对通过的光进行控制，为了产生逼真明亮的图像，所以液晶电视的显示屏都有一个高亮度，光谱范围宽类似太阳光的背光源，一般采用组合冷阴极荧光灯（CCFL）作为液晶显示屏的背光源。

RobotTTP 本体上的 TFT 屏的分辨率是 320*240，代表屏幕的长宽分别有 320 和 240 个像素点，每个像素点的颜色由红（Red）、绿（Green）、蓝（Blue）三种色光按照不同的比例混合而成。采用的 RGB 格式是 RGB565，RGB565 使用 16 位表示一个像素，这 16 位中的 5 位用于 R，6 位用于 G，5 位用于 B。程序中通常使用一个字（WORD，一个字等于两个字节）来操作一个像素。当读出一个像素后，这个字的各个位意义如下：R R R R R G G G G G G B B B B B。依次给屏幕中的每个像素点都赋予特定的颜色，那么一副图片就可以显示出来。

BMP 格式的图片如何在 RobotTTP 本体上的 TFT 屏显示：

1、打开图片取模软件，取模软件的界面如下图 2-14-2 所示。

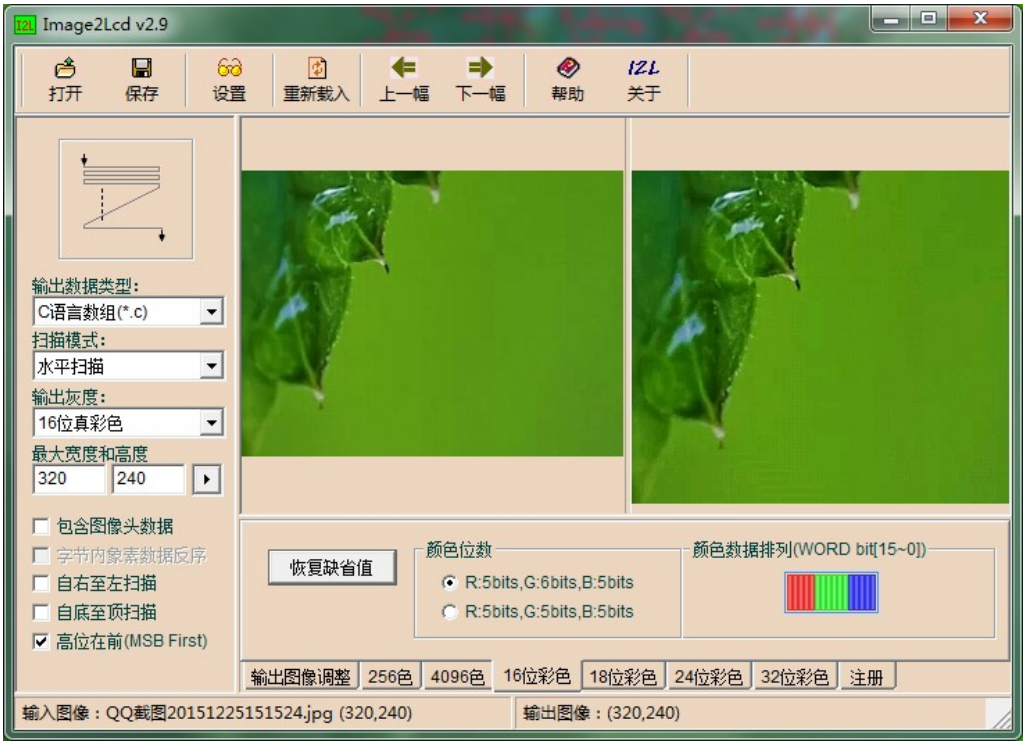


图 2-14-2 图片取模软件界面

2、点击“打开”，选择需要加载的图片。

3、点击保存，将取到的数模文件保存。

4、编写程序，将数模文件中的数据发送到 TFT 屏就可以显示出图片。

TFT 屏显示汉字的原理：

RobotTTP 本体上的 TFT 屏的分辨率是 320*240，所以有 76800 个像素点。点亮指定的像素点，就可以在屏幕上显示出特定的汉字。字模指的是点阵字模，我们知道屏幕是由一个个的像素点组成的，点亮若干个像素点，在屏幕上看到的现象就是有若干个亮点，如果这些亮点是按照汉字的轨迹排列的，那么我们看到的就是一个汉字。把汉字转换成点阵字模的过程称为取模。

如图 2-14-3 所示，为汉字取模软件界面。在汉字输入栏输入待取模的汉字，点击左边的“生成字模”按钮，会在下方的字模生成区产生相应的字模，点击“保存字模”按钮可以把字模保存起来，或者选中字模生成区的字模复制粘贴。

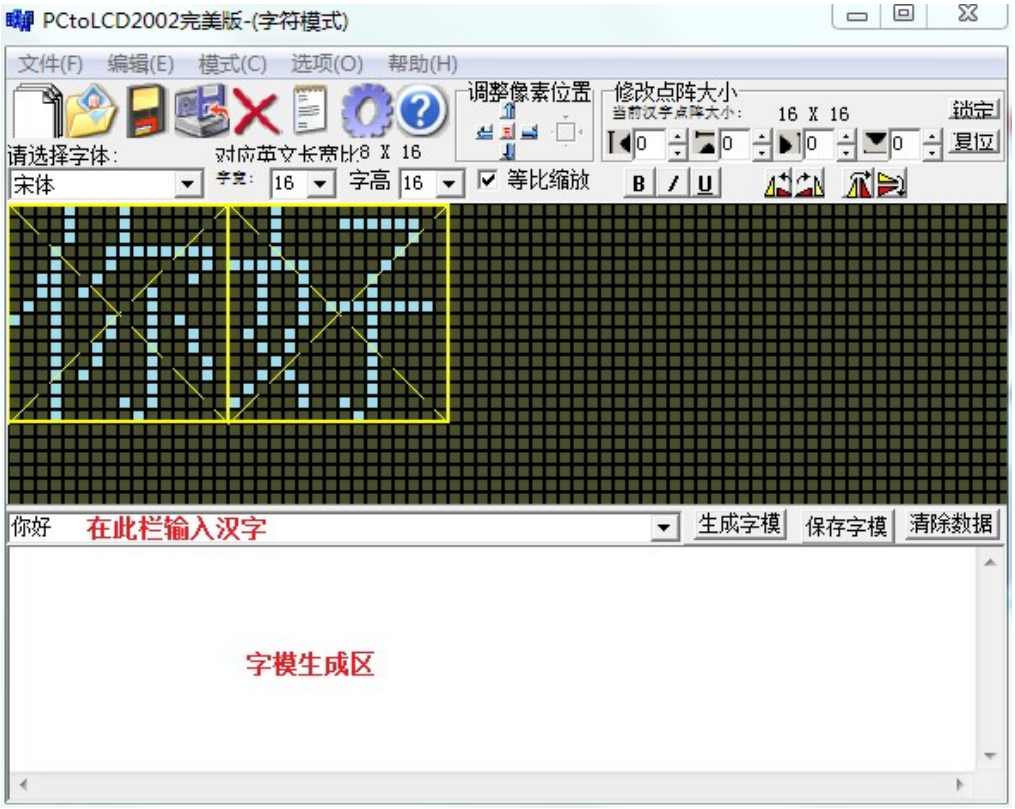


图 2-14-3 汉字取模软件界面

FSMC 简介：

FSMC(Flexible Static Memory Controller，可变静态存储控制器)是 STM32 系列中内部集成 256 KB 以上 FlaSh，后缀为 xC、xD 和 xE 的高存储密度微控制器特有的存储控制机制。之所以称为"可变"，是由于通过对特殊功能寄存器的设置，FSMC 能够根据不同的外部存储器类型，发出相应的数据、地址、控制信号类型以匹配信号的速度，从而使得 STM32 系列微控制器不仅能够应用各种不同类型、不同速度的外部静态存储器，而且能够在不增加外部器件的情况下同时扩展多种不同类型的静态存储器，满足系统设计对存储容量、产品体积以及成本的综合要求。

实验内容

[1]新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

[3]配置 GPIO 口工作模式。

[4]配置 FSMC。

[5]编写主函数。

[6]使用 STM32 ST-LINK Utility 软件下载程序。

[7]观察实验现象。

实验步骤

[1] 新建一个 Keil 工程。

新建一个 Keil 工程。

[2] 配置 STM32 系统时钟、GPIO 时钟。按照如下步骤进行：

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能时钟：RCC_AHBPeriphClockCmd(RCC_AHBPeriph_FSMC,ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOE|RCC_APB2Periph_GPIOF|RCC_APB2Periph_GPIOG|RCC_APB2Periph_USART1|RCC_APB2Periph_AFIO, ENABLE);

库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3] 配置 GPIO 口的工作模式。

```
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode =  GPIO_Mode_AF_PP;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0  | GPIO_Pin_1  | GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 | GPIO_Pin_14 | GPIO_Pin_15;
GPIO_Init ( GPIOD, & GPIO_InitStructure );
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 | GPIO_Pin_11 | GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
GPIO_Init ( GPIOE,  & GPIO_InitStructure );
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode =  GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
    GPIO_Init (GPIOD, & GPIO_InitStructure );
```

```
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_Init (GPIOD, & GPIO_InitStructure );
    macILI9341_CS_APBxClock_FUN ( macILI9341_CS_CLK, ENABLE );
    GPIO_InitStructure.GPIO_Pin = macILI9341_CS_PIN;
    GPIO_Init ( macILI9341_CS_PORT, & GPIO_InitStructure );
    macILI9341_DC_APBxClock_FUN ( macILI9341_DC_CLK, ENABLE );
    GPIO_InitStructure.GPIO_Pin = macILI9341_DC_PIN;
    GPIO_Init ( macILI9341_DC_PORT, & GPIO_InitStructure );
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    macILI9341_RST_APBxClock_FUN ( macILI9341_RST_CLK, ENABLE );
    GPIO_InitStructure.GPIO_Pin = macILI9341_RST_PIN;
    GPIO_Init ( macILI9341_RST_PORT, & GPIO_InitStructure );
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    macILI9341_BK_APBxClock_FUN ( macILI9341_BK_CLK, ENABLE );
    GPIO_InitStructure.GPIO_Pin = macILI9341_BK_PIN;
    GPIO_Init ( macILI9341_BK_PORT, & GPIO_InitStructure );
```

[4] 配置 FSMC。

```
void ILI9341_FSMC_Config ( void )
{
    FSMC_NORSRAMInitTypeDef  FSMC_NORSRAMInitStructure;
    FSMC_NORSRAMTimingInitTypeDef  p;
    RCC_AHBPeriphClockCmd ( RCC_AHBPeriph_FSMC, ENABLE );
    p.FSMC_AddressSetupTime      = 0x02;  //地址建立时间
    p.FSMC_AddressHoldTime       = 0x00;  //地址保持时间
    p.FSMC_DataSetupTime         = 0x05;    //数据建立时间
    p.FSMC_BusTurnAroundDuration = 0x00;
    p.FSMC_CLKDivision           = 0x00;
    p.FSMC_DataLatency           = 0x00;
    p.FSMC_AccessMode            = FSMC_AccessMode_B;  //
    FSMC_NORSRAMInitStructure.FSMC_Bank   = macFSMC_Bank1_NORSRAMx;
   FSMC_NORSRAMInitStructure.FSMC_DataAddressMux= FSMC_DataAddressMux
   _Disable;
    FSMC_NORSRAMInitStructure.FSMC_MemoryType= FSMC_MemoryType_NOR;
   FSMC_NORSRAMInitStructure.FSMC_MemoryDataWidth= FSMC_MemoryDataW
   idth_16b;
   FSMC_NORSRAMInitStructure.FSMC_BurstAccessMode= FSMC_BurstAccessMo
   de_Disable;
   FSMC_NORSRAMInitStructure.FSMC_WaitSignalPolarity= FSMC_WaitSignalPola
   rity_Low;
    FSMC_NORSRAMInitStructure.FSMC_WrapMode= FSMC_WrapMode_Disable;
```

```
    FSMC_NORSRAMInitStructure.FSMC_WaitSignalActive= FSMC_WaitSignalActiv
    e_BeforeWaitState;
    FSMC_NORSRAMInitStructure.FSMC_WriteOperation= FSMC_WriteOperation_En
    able;
     FSMC_NORSRAMInitStructure.FSMC_WaitSignal= FSMC_WaitSignal_Disable;
    FSMC_NORSRAMInitStructure.FSMC_ExtendedMode= FSMC_ExtendedMode_Dis
    able;
     FSMC_NORSRAMInitStructure.FSMC_WriteBurst= FSMC_WriteBurst_Disable;
     FSMC_NORSRAMInitStructure.FSMC_ReadWriteTimingStruct = & p;
     FSMC_NORSRAMInitStructure.FSMC_WriteTimingStruct     = & p;
     FSMC_NORSRAMInit ( & FSMC_NORSRAMInitStructure );
     FSMC_NORSRAMCmd ( macFSMC_Bank1_NORSRAMx, ENABLE );
}
```

[5] 编写主函数。

第一步：包含头文件：

```
#include "stm32f10x.h"
#include "bsp_lcd.h"
#include "palette.h"
```

第二步：声明程序中的子函数：

```
void RCC_Configuration(void);
```

第三步：编写主函数函数体：

```
int main ( void )
{
    RCC_Configuration();
    LCD_Init ();         //LCD 初始化
     Palette_Init();     //触摸取色板初始化
    while ( 1 )
    {
              strType_XPT2046_Coordinate strDisplayCoordinate;
              if ( ucXPT2046_TouchFlag == 1 )
              {
                    if ( XPT2046_Get_TouchedPoint ( & strDisplayCoordinate, & strXPT2
                   046_TouchPara ) )  //获取触摸点的坐标
                    Palette_draw_point ( strDisplayCoordinate .x, strDisplayCoordinate .y );
              }
     }
}
```

[6] 使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 TFTChuMoPing.hex 文件烧写到 RobotTTP。

[7] 观察实验现象。

观察实验现象。

参考程序

参考程序的工程名为“TFTChuMoPing.uvproj”，在本实验文件目录下“TFTChu MoPing”文件夹下。

实验小结

通过本节学习，了解 TFT 液晶屏的读写和显示原理，并掌握在 TFT 液晶屏上显示文字和图片的使用方法。

思考题

对本实验的程序进行修改，编写相应的程序，利用图像处理知识，将图片以反色的形式显示在屏幕上。

2.15 摄像头

实验目的

了解 RobotTTP 本体摄像头模块的工作原理及其相关知识。
熟悉和掌握摄像头在 RobotTTP 中的使用。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）。
软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件。

实验预习

OV7670 图像传感器简介：

OV7670/OV7171 CAMERACHIPTM 图像传感器，体积小、工作电压低，提供单片 VGA 摄像头和影像处理器的所有功能。通过 SCCB 总线控制，可以输出整帧、子采样、取窗口等方式的各种分辨率 8 位影响数据。该产品 VGA 图像最高达到 30 帧/秒。用户可以完全控制图像质量、数据格式和传输方式。所有图像处理功能过程包括伽玛曲线、白平衡、饱和度、色度等都可以通过 SCCB 接口编程。OmmiVision 图像传感器应用独有的传感器技术，通过减少或消除光学或电子缺陷如固定图案噪声、托尾、浮散等，提高图像质量，得到清晰的稳定的彩色图像。详细资料参考本文件目录下“OV 7670 中文版数据手册.pdf”。

FIFO 存储器简介：

FIFO 是英文 First In First Out 的缩写，是一种先进先出的数据缓存器，没有外部读写地址线，但只能顺序写入、读出数据，其内部读写指针自动加 1，不能决定读取或写入某个指定的地址。FIFO 一般用于不同时钟域之间的数据传输。对于单片 FIFO 来说，主要有两种结构：触发导向结构和零导向传输结构。触发导向传输结构的 FIFO 是由寄存器阵列构成的，零导向传输结构的 FIFO 是由具有读和写地址指针的双口 RAM 构成，如图 2-15-1：

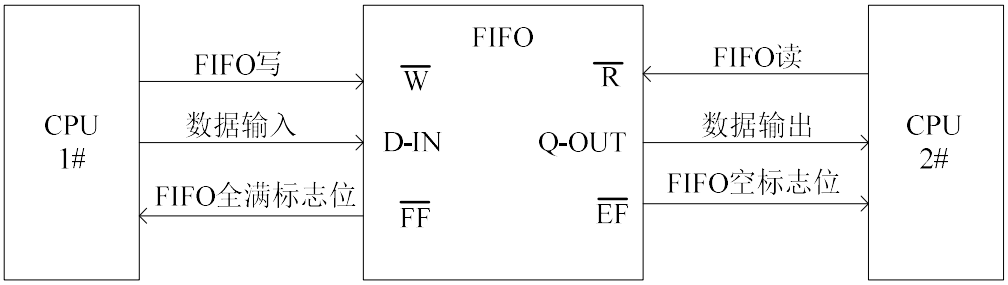


图 2-15-1 FIFO 工作原理示意图

FIFO 芯片 AL422B 简介：

AL422B 是由 AverLogic 公司推出的存储容量为 3Mbits 的视频帧存储器，由于目前 1 帧图像信息通常包含 640×480 或 720×480 个字节， 而市面上很多视频存储器由于容量有限只能存储 1 场图像信息，无法存储 1 帧图像信息。AL422B 由于容量很大，可存储 1 帧图像的完整信息，其工作频率达 50MHz。

实验内容

[1]新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

[3]配置 GPIO 口工作模式。

[4]配置外部中断程序。

[5]编写中断服务函数。

[6]编写显示图像程序。

[7]编写主函数。

[8]使用 STM32 ST-LINK Utility 软件下载程序。

[9]观察实验现象。

实验步骤

[1]新建一个 Keil 工程。

新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

1) 恢复 RCC 时钟到默认值：RCC_DeInit()；
2) 开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；
3) 等待外部晶振启动：RCC_WaitForHSEStartUp()；
4) 代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；
5) 半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；
6) 预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；
7) 选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；
8) 使能 PLL：RCC_PLLCmd(ENABLE)；
9) PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；
10) 系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；
11) AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；
12) AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；
13) APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);
14) 关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；
16) 使能时钟：RCC_AHBPeriphClockCmd(RCC_AHBPeriph_FSMC,ENABLE);

RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOE|RCC_APB2Periph_GPIOF|RCC_APB2Periph_GPIOG|RCC_APB2Periph_USART1|RCC_APB2Periph_AFIO, ENABLE);

库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3]配置 GPIO 口工作模式。

```
GPIO_InitStructure.GPIO_Pin =GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//通用推挽输出
GPIO_Init(GPIOC, &GPIO_InitStructure);//对选中管脚初始化
GPIO_ResetBits(GPIOC,GPIO_Pin_13|GPIO_Pin_14);//
GPIO_SetBits(GPIOC,GPIO_Pin_15);
GPIO_InitStructure.GPIO_Pin =GPIO_Pin_5;//OV_WEN
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//通用推挽输出
GPIO_Init(GPIOE, &GPIO_InitStructure);//对选中管脚初始化
GPIO_ResetBits(GPIOE,GPIO_Pin_5);//
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;//OV_VSYNC 外部中断脚
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;//上拉输入
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOE, &GPIO_InitStructure);//对选中管脚初始化
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1| GPIO_Pin_2| GPIO_Pi
n_3|
GPIO_Pin_4| GPIO_Pin_5| GPIO_Pin_6| GPIO_Pin_7;//OV_D0-OV_D7
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;//模式输入上拉
GPIO_Init(GPIOF, &GPIO_InitStructure);//对选中管脚初始化
GPIO_InitStructure.GPIO_Pin =GPIO_Pin_8|GPIO_Pin_9|GPIO_Pin_10;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//通用推挽输出
GPIO_Init(GPIOF, &GPIO_InitStructure);//对选中管脚初始化
GPIO_ResetBits(GPIOF,GPIO_Pin_8|GPIO_Pin_10);//
GPIO_SetBits(GPIOF,GPIO_Pin_9);
```

[4]配置外部中断程序。

```
void EXTI_Configuration(void)
{
    EXTI_InitTypeDef EXTI_InitStructure;  //初始化结构
    NVIC_InitTypeDef NVIC_InitSructure;
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource6);//
```

```
    EXTI_ClearITPendingBit(EXTI_Line6);//清除中断标志位 EXTI_Line6
    EXTI_InitStructure.EXTI_Mode =EXTI_Mode_Interrupt; //选择中断模式请求
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;//下降沿触发
    EXTI_InitStructure.EXTI_Line = EXTI_Line6;  //选择使能的外部中断线
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;   //定义选中线的新状态使能
    EXTI_Init(&EXTI_InitStructure);//配置上述参数
    EXTI_GenerateSWInterrupt(EXTI_Line6);
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    NVIC_InitSructure.NVIC_IRQChannel = EXTI9_5_IRQn;//开放线 6 的中断
    NVIC_InitSructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitSructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitSructure.NVIC_IRQChannelCmd = ENABLE;  //使能指定通道
    NVIC_Init(&NVIC_InitSructure);
}
```

[5]编写中断服务函数。

```
void EXTI9_5_IRQHandler(void)
{
    if ( EXTI_GetITStatus(EXTI_Line6) ==SET )
    {
            if(sct_flag==0)
            {
                    FIFO_WRST_L;
                    FIFO_WE_H;
                    sct_flag=1;
                    FIFO_WRST_H;
            }
            else if(sct_flag==1)
            {
                    sct_flag=2;
            }
    }
    EXTI_ClearITPendingBit(EXTI_Line6);//清除 PE6 挂起位
}
```

[6]编写显示图像程序。

```
void XianShi_TuXiang(void)
{
    u32 i;//局部变量
    u16 j;
    u16 cm_d;
    FIFO_RRST_L;
```

```
    FIFO_RCK_L;
    FIFO_RCK_H;
    FIFO_RCK_L;
    FIFO_RRST_H;
    FIFO_RCK_H;
    for(i = 0; i < 76800; i++)//76800
    {
            FIFO_RCK_L;
            cm_d= GPIOF->IDR & 0xff;
            FIFO_RCK_H;
            cm_d<<=8;
            FIFO_RCK_L;
            cm_d |= GPIOF->IDR & 0xff;
            FIFO_RCK_H;
            ILI9341_Write_Data(cm_d);
    }
}
```

[7]编写主函数。

```
int main ( void )
{
    u8 lightmode=0,saturation=2,brightness=2,contrast=2;
    u8 effect=0;
    RCC_Configuration();
    LCD_Init ();        //LCD 初始化
    SheXiang_GPIO_Init();
    USART_Config();
    if(OV7670_Init()==0)
    {
            USART_SendData(USART1,0x55);
            while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
    }//摄像头初始化
    delay(8000000);
    delay(8000000);
    OV7670_Light_Mode(lightmode);
    OV7670_Color_Saturation(saturation);
    OV7670_Brightness(brightness);
    OV7670_Contrast(contrast);
    OV7670_Special_Effects(effect);
    EXTI_Configuration();//外部中断配置，用于摄像头
    OV7670_Window_Set(12,178,240,320);//设置窗口
    ILI9341_GramScan( 2 );
    while(1)
```

```
    {
        if(sct_flag>1)//有摄像头图像信息
        {
            XianShi_TuXiang();//显示图像
            sct_flag=0;
        }
    }
}
```

[8]使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 SheXiangTou.hex 文件烧写到 RobotTTP。

[9]观察实验现象。

观察液晶屏显示。

参考程序

参考程序的工程名为“SheXiangTou.uvproj”，在本实验文件目录下“SheXiangTou”文件夹下。

实验小结

通过对本节的学习，了解 RobotTTP 本体摄像头模块的工作原理。熟悉和掌握摄像头在 RobotTTP 中的使用。

思考题

对本实验的程序进行修改，编写相应的程序，改变当前图像的采集速率和大小，分析图像采集速率和大小之间的关系。

3.基于 STM32 的机器人电机控制实验

机器人是一种集机械、电子、传感器技术、控制技术等多种现代化技术与一体的电机一体化产品。随着科学技术的发展，机器人发挥着越来越越重要的作用。学习机器人相关知识与技术已经成为当前理工科类大学生必备技能。机器人一般是由机械本体、控制系统和电机驱动系统等部分组成。机器人接受控制系统的指令，按照预定的速度和轨迹运动到预定的位置，或者完成一系列复杂的操作。这些动作都是有电机控制系统来完成的，而电机控制系统既不开控制中心，又离不开驱动电机，只有学习并实训操作才能掌握机器电机控制系统。实现机器人如人一样的自由运动，既要有敏捷的“脚”，又要有灵活的“手”。因此对电机的基本要求是：

（1） 反应速度快；

（2） 运动速度及精度高；

（3） 负载能力强，可靠性高。

在常见的机器人电机使用中有步进电机、直流电机、交流电机、伺服电机、舵机等种类。不同的电机运用于机器人的不同部位，例如步进电机和直流减速电机常用于机器人行走，舵机常用于机器人手臂驱动。

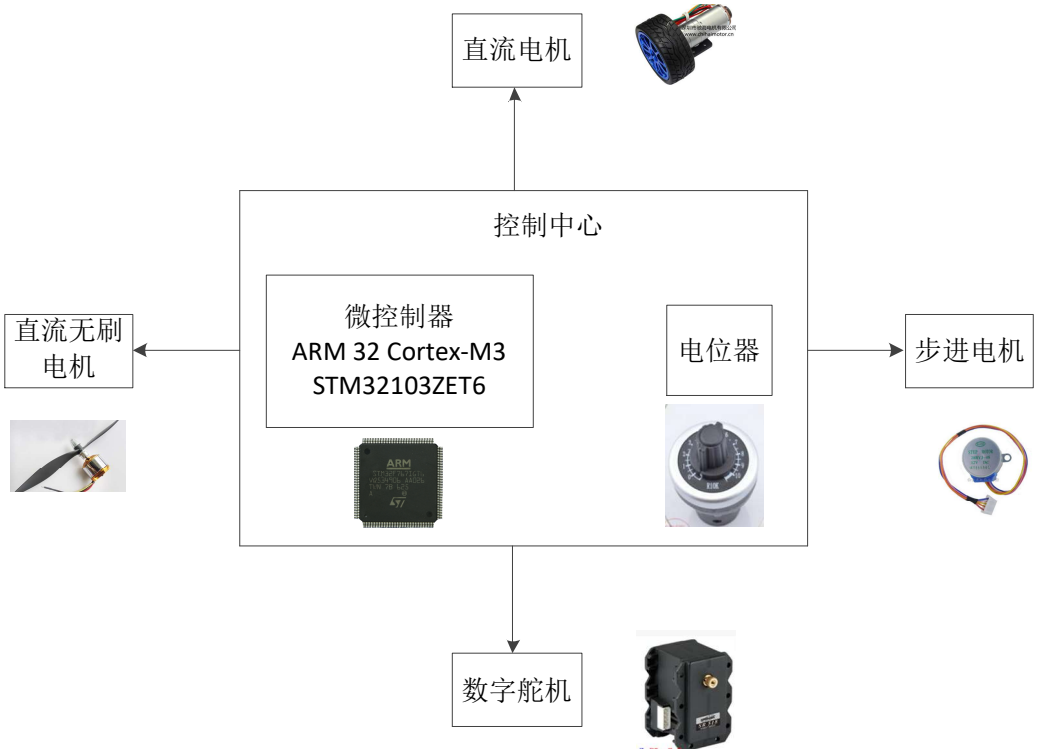


图 3-1 机器人电机控制系统结构图

在本实验教程中，机器人电机控制系统主要围绕基本的电机控制展开，包括了直流电机、步进电机、直流无刷电机、数字舵机以及机械手臂。通过以上基本实验让学者初步了解机器人电机控制系统的一些基本知识，为将来的机器人制作等领域提供基础。

本实验教程对初学者的要求是应具有基本的 STM32 嵌入式和自动控制基础，对诸如电路原理图设计、STM32 嵌入式编程、电机控制等应有基本的了解。

3.1 直流电机

实验目的

了解直流电机工作原理及电气特性。

掌握直流电机启动方法。

编程实现电机调速控制。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）。

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件。

实验预习

直流电机参数：

电机轴长	12mm D 长 8mm	电机轴径	4mm D 字型
轮子直径	65mm		
额定电压	DC12.0V		
空载转速参数	350RPM 0.2A		
最大效率点参数	负载 2.0kg•cm/285rpm/5.0W/0.65A		
最大功率点参数	负载 5.8kg•cm/180rpm/9.0W/1.65A		
堵转参数	12.0kg•cm 堵死电流 5.5A		
减速器减速比	1：34		
霍尔分辨率	霍尔分辨率 11×精确减速比 34.02=341.2PPR		

直流电机工作原理：

此实训平台采用的是双霍尔编码器减速电机，拥有超长的使用寿命，噪音低，力矩大，性能稳定可靠，可以实现正反转。

直流电机主要由定子、转子、转换器等构成。定子产生磁场，支撑电机。转子产生电磁转矩和感应电动势，进行能量转换。转换器与电刷配合使用，在直流电动机中，将外加直流电源转换为电枢线圈中的交变电流，使电磁转矩的方向恒定不变。电机的工作原理是建立在电磁感应定律、电磁力定律、电路定律等基本理论之上。

霍尔编码器是一种通过磁电转换将输出轴上的机械几何位移量转换成脉冲或数字量的传感器。霍尔编码器是由霍尔码盘和霍尔元件组成。霍尔码盘是在一定直径的圆板上等分地布置有不同的磁极。霍尔码盘与电动机同轴，电动机旋转时，霍尔元件检测输出若干脉冲信号，为判断转向，一般输出两组存在一定相位差的方波信号。

直流电机采用双功率两组 H 桥驱动器驱动，具体型号是 L298P，具体技术参数和原理参考本文件目录下的“L298.pdf”。L298P 可以直接驱动 2 个直流电机，驱动电流

达 2A,电机输出端采用 8 只高速肖特基二极管作为保护。L289P 的管脚分布图如图 3-1-1 所示。

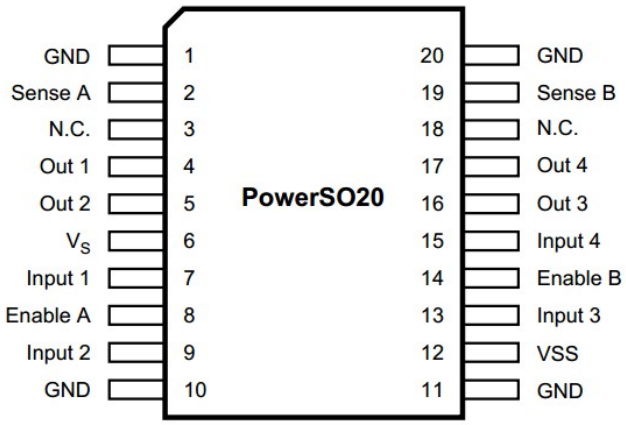


图 3-1-1 L298P 管脚分布图

每个直流电机都由 L298P 驱动，如图 3-1-2 所示，要使直流电机运转，必须使其对对角线上的一对三极管导通。根据不同三极管对的导通情况，电流可能会从左至右或从右至左流过电机，从而控制电机的转向。

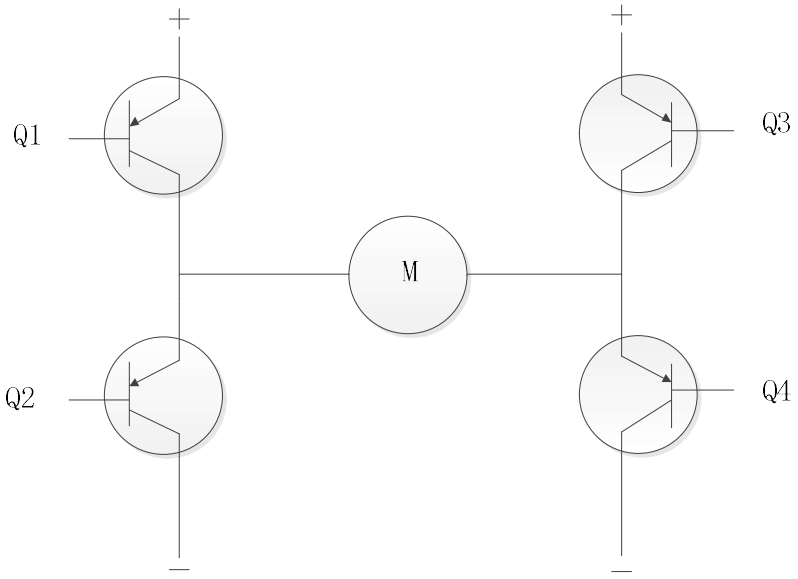


图 3-1-2 H 桥驱动电路原理图

通过 STM32 中央微处理器 Enable 使能使驱动芯片 L298P 和电机通道打开，Ctrl 控制驱动芯片输出 OUT1、OUT2 高（或低）电平，实现电机的运转，同时还可实现电机的正反转。A、B 相是编码器反馈出电机正转、反转。如图 3-1-3 所示为电路原理图。

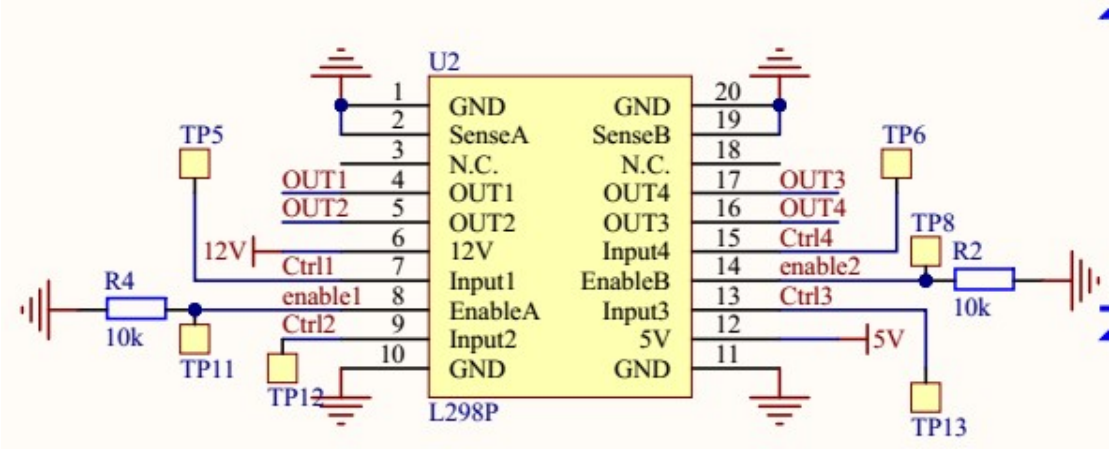


图 3-1-3 电路原理图

实验内容

[1]新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

[3]配置 GPIO 口工作模式。

[4] 配置 TIM2 模块，实现驱动电机。

[5]编写主函数。

[6]使用 STM32 ST-LINK Utility 软件下载程序。

[7]通过调节电机调速装置，改变电机的速度，观察实验现象。

实验步骤

[1]新建一个 Keil 工程。

新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能 GPIOA、GPIOB、GPIOE 时钟：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOE|RCC_APB2Periph_ADC1, ENABLE)；

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3|RCC_APB1Periph_TIM4, ENABLE);

库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3]配置 GPIO 口工作模式。

1)恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();

2)配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

3)配置各端口：

第一步：

由于 PB2、PB4、PB9 都是外部模块的电源控制端，所以均需要配置成输出低电平，可以降低系统的功耗。如：

GPIO_InitStructure.GPIO_Pin =GPIO_Pin_2|GPIO_Pin_4|GPIO_Pin_9;

GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;//速度

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//推挽输出

GPIO_Init(GPIOB, &GPIO_InitStructure);

GPIO_ResetBits(GPIOB,GPIO_Pin_2|GPIO_Pin_4|GPIO_Pin_9);设定管脚输出低电平。

第二步：

本实验用到的电机驱动管脚是，配置如下：PA0、PA1、PA1、PA3、PA4、PA5、PA6、PA7，配置如下：

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3|GPIO_Pin_1;//PA1/3-TIM2

GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;//复用推挽输出

GPIO_Init(GPIOA, &GPIO_InitStructure);

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5|GPIO_Pin_4|GPIO_Pin_2|GPIO_Pin_0;

GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;//速度

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//推挽输出

GPIO_Init(GPIOA, &GPIO_InitStructure);

GPIO_ResetBits(GPIOA,GPIO_Pin_5|GPIO_Pin_4|GPIO_Pin_2|GPIO_Pin_0);

[4] 配置 TIM2 模块，实现驱动电机。

```
TIM_TimeBaseStructure.TIM_Period=15999;         //200us  5KHz
TIM_TimeBaseStructure.TIM_Prescaler=8;          //设置预分频：72/9=8MHz
TIM_TimeBaseStructure.TIM_ClockDivision=0;     //设置时钟分频系数：不分频
TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;//向上计数溢出模式
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
TIM_OCInitStructure.TIM_OCMode=TIM_OCMode_PWM1; //配置为 PWM 模式 1
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 0; //设置通道 2 的电平跳变值，当计数器计数到这个值时，电平发生跳变
TIM_OCInitStructure.TIM_OCPolarity=TIM_OCPolarity_High;  //当定时器计数值小于 CCR2 时为高电平
TIM_OC2Init(TIM2,&TIM_OCInitStructure);         //使能通道 2
TIM_OC2PreloadConfig(TIM2, TIM_OCPreload_Enable);
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 0;  //设置通道 4 的电平跳变值，当计数器计数到这个值时，电平发生跳变
TIM_OCInitStructure.TIM_OCPolarity=TIM_OCPolarity_High;  //当定时器计数值小于 CCR4 时为高电平
TIM_OC4Init(TIM2,&TIM_OCInitStructure);          //使能通道 4
TIM_OC4PreloadConfig(TIM2, TIM_OCPreload_Enable);
TIM_ARRPreloadConfig(TIM2,ENABLE);       //使能 TIM2 重载寄存器 ARR
TIM_Cmd(TIM2, ENABLE);
```

[5]编写主函数。

```
GPIO_SetBits(GPIOA, GPIO_Pin_4|GPIO_Pin_5);         //电机使能
USART_Config(USART1, 115200);
TIM1_cnt_Config(999,359);
TIM2_PWM_Config();
TIM3_Encoder_Configuration();
TIM4_Encoder_Configuration();
AD_Single_Config();
while(1)
{
 if(tim1_cntr>=50)
 {
  tim1_cntr=0;
  GPIOB->ODR^=(1<<9);
  dcmotor_deal();
```

```
TIM_SetCompare4(TIM2,((speed3_set-0XC8)*14)%15999);
TIM_SetCompare2(TIM2,((speed4_set-0XC8)*14)%15999);
  }
 if(rcv1_flag==1){rcv1_flag=0;com1_order();}
 delay_s(0x7f);
}
```

[6] 使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 ZhiLiu.hex 文件烧写到 RobotTTP。

[7]通过调节电机调速装置，改变电机的速度，观察实验现象。

调节实验箱上的电机调速模块中的电位器，可以改变电机的转速大小。调节装置如果 3-1-4 所示。

3-1-4 电机调速装置

参考程序

参考程序的工程名为“ZhiLiu.uvproj”，在本实验文件目录下“ZhiLiu”文件夹下。

实验小结

通过对本节的学习，了解直流电机的内部结构、运行机理、驱动控制原理。学会电机调速控制。

思考题

1、如何通过程序改变电机的转速？

2、如何改变使电机实现正反转？

3.2 步进电机

实验目的

了解步进电机的内部构造及其驱动控制原理。

熟悉和掌握基于 RobotTTP 中步进电机的使用方法。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）。

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件。

实验预习

步进电机的技术参数如表 1 所示：

参数	值	参数	值
电压（V）	12	空载嵌入频率（Hz）	≥500
直流电阻（25℃±7%）	85	空载牵出频率（Hz）	≥800
步距角	5.625°/64	绝缘耐压	600
减速比	1/64	温升（K）	≤55
牵入转矩（100Hz 时）gf•cm	550	噪音 dB	≤40
自定位转矩	300	驱动方式	四相八拍

表 1 技术参数

步进电机工作原理：

虽然步进电机已被广泛地应用，但是步进电机并不能像普通的直流电机、交流电机那样在常规下使用。它必须由双环形脉冲信号、功率驱动电路等组成控制系统方可使用。用好步进电机却非易事，它涉及到机械、电机、电子及计算机等专业知识。

步进电机是将电脉冲信号转变为角位移或线位移的开环控制电机。在非超载的情况下，电机的转速、停止位置只取决于脉冲信号的频率和脉冲数，而不受负载变化的影响，当步进驱动器接收到一个脉冲信号，它就驱动步进电机按设定的方向转动一个固定的角度，称为“步距角”，它的旋转是以固定的角度一步一步运行的。可以通过控制脉冲个数来控制角位移量，从而达到准确定位的目的；同时可以通过控制脉冲频率来控制电机转动的速度和加速度，从而达到调速的目的。

RobotTTP 采用的是型号为 28BYJ48 型的 4 相 5 线步进电机。当对步进电机施加一系列连续不断的控制脉冲时，它可以连续不断地转动。每一个脉冲信号对应步进电机的某一组或二组绕组的通电状态改变一次，也就对应转子转过一定的角度（一个步距

角）。当通电状态的改变完成一个循环时，转子转过一个齿距。该电机为四相八拍，其内部结构如图 3-2-1 所示。四相步进电机可以在不同的通电方式下运行，常见得通电方式有单（单相绕组通电）四拍（A-B-C-D-A-…）,双（双相绕组通电）四拍（AB-BC-CD-DA-AB-…），八拍（A-AB-B-BC-C-CD-D-DA-A-…）。表 2 列出了相应步进电机的通电逻辑时序。

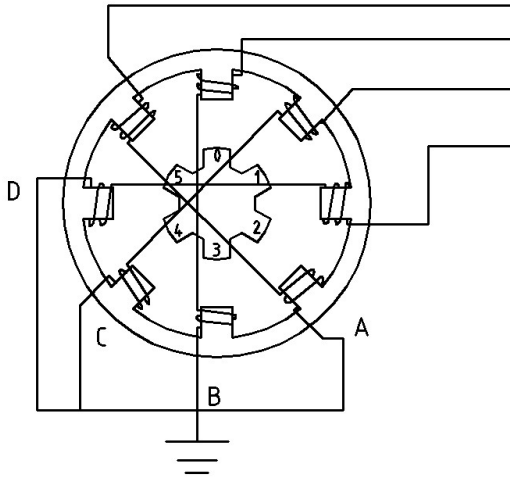


图 4-1-1 四相八拍制步进电机内部构造图

步进	01	02	03	04	真值表	通电
0	ON	OFF	ON	OFF	1010	AC
1	ON	OFF	OFF	OFF	1000	A
2	ON	OFF	OFF	ON	1001	AD
3	OFF	OFF	OFF	ON	0001	D
4	OFF	ON	OFF	ON	0101	BD
5	OFF	ON	OFF	OFF	0100	B
6	OFF	ON	ON	OFF	0110	BC
7	OFF	OFF	ON	OFF	0010	C
8	ON	OFF	ON	OFF	1010	AC

表 2 八拍驱动方式逻辑时序

由于步进电机的驱动电流较大，单片机不能直接驱动，一般都采用型号为 ULN2003 驱动芯片进行驱动，该芯片具体的技术参数和原理参考本文件目录下的“ULN2003_ST.pdf”。ULN2003 管脚分布如图 3-2-2 所示。

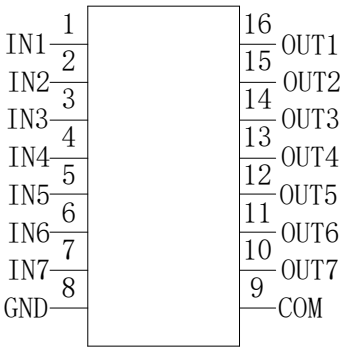


图 3-2-2 ULN2003 管脚分布图

通过 STM32 中央微处理器的 PB12、PB13、PB14、PB15 脚作为 ULN2003 的 1-4 口输入端所构成的步进电机驱动电路，其电路原理如图 3-2-3 所示。

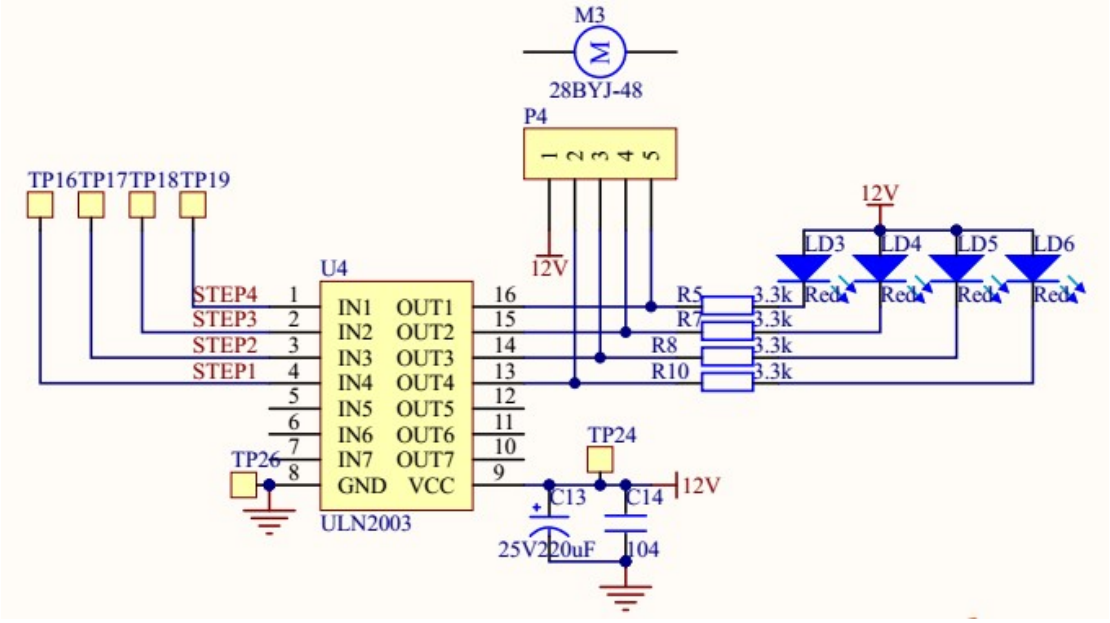


图 3-2-3 电路原理图

实验内容

[1]新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

[3]配置 GPIO 口工作模式。

[4]运用 GPIO 控制方式实现驱动电机。

[5]编写主函数。

[6]使用 STM32 ST-LINK Utility 软件下载程序。

[7]通过调节电机调速装置，改变电机的速度，观察实验现象。

实验步骤

[1]新建一个 Keil 工程。

新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能 GPIOA、GPIOB、GPIOE 时钟：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOE|RCC_APB2Periph_ADC1, ENABLE)；

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3|RCC_APB1Periph_TIM4, ENABLE);

库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3]配置 GPIO 口工作模式。

1)恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();

2)配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

3)配置各端口：

第一步：

由于 PA4、PA5、PB2、PB4 都是外部模块的电源控制端，所以均需要配置成输出低电平，可以降低系统的功耗。以 A 端口为例：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4|GPIO_Pin_5;//PA4 PA5
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//推挽输出
```

GPIO_Init(GPIOA, &GPIO_InitStructure);
GPIO_ResetBits(GPIOA, GPIO_Pin_4|GPIO_Pin_5);
第二步：
本实验用到的电机驱动管脚是 PB12、PB13、PB14、PB15，配置如下：
GPIO_InitStructure.GPIO_Pin=|GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//推挽输出
GPIO_Init(GPIOB, &GPIO_InitStructure);

[4]运用 GPIO 控制方式实现驱动电机。

```
switch(motor_step)
  {
        case 0:
              if(systick_flag==1)
                    {
                          systick_flag=0;    //关闭定时器
                          A_H;B_L;C_L;D_L;  //1
                          motor_step=1;
                    }
              break;
        case 1:
              if(systick_flag==1)
                    {
                          systick_flag=0;    //关闭定时器
                          A_H;B_H;C_L;D_L;  //2
                          motor_step=2;
                    }
              break;
        case 2:
              if(systick_flag==1)
                    {
                          systick_flag=0;    //关闭定时器
                          A_L;B_H;C_L;D_L;  //3
                          motor_step=3;
                    }
              break;
        case 3:
              if(systick_flag==1)
                    {
                          systick_flag=0;    //关闭定时器
                          A_L;B_H;C_H;D_L;  //4
                          motor_step=4;
```

```
                }
            break;
        case 4:
            if(systick_flag==1)
                {
                    systick_flag=0;    //关闭定时器
                    A_L;B_L;C_H;D_L;  //5
                    motor_step=5;
                }
            break;
        case 5:
            if(systick_flag==1)
                {
                    systick_flag=0;    //关闭定时器
                    A_L;B_L;C_H;D_H;  //6
                    motor_step=6;
                }
            break;
        case 6:
            if(systick_flag==1)
                {
                    systick_flag=0;    //关闭定时器
                    A_L;B_L;C_L;D_H;  //7
                    motor_step=7;
                }
            break;
        case 7:
            if(systick_flag==1)
                {
                    systick_flag=0;    //关闭定时器
                    A_H;B_L;C_L;D_H;  //8
                    motor_step=0;
                }
            break;
    }
```

[5]编写主函数。

```
GPIO_ResetBits(GPIOB, GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15);
while(1)
 {
   if(tim2_cntr>=20)
        {
            tim2_cntr=0;
```

```
                    GPIOB->ODR^=(1<<9);

                    step_deal();
                }
    if(rcv1_flag==1){rcv1_flag=0;com1_order();}
    step_control();
    delay_s(0x7f);
  }
```

[6]使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 Stepmotor.hex 文件烧写到 RobotTTP。

[7]通过调节电机调速装置，改变电机的速度，观察实验现象。

调节实验箱上的电机调速模块中的电位器，可以改变电机的转速大小。调节装置如果 3-2-4 所示。

3-2-4 电机调速装置

参考程序

参考程序的工程名为“Stepmotor.uvproj”，在本实验文件目录下“Stepmotor”文件夹下。

实验小结

通过对本节的学习，了解四相八拍制步进电机的内部结构、运行机理、驱动控制原理。学会电机调速控制。

思考题

1、通过实验确定步进电机的最大旋转速度。

2、测量电机四相电（A、B、C、D），观察其现象。

3.3 直流无刷电机

实验目的

了解直流无刷电机的工作原理及电气特性。

掌握直流无刷电机控制方法。

编程实现直流无刷电机调速控制。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）。

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件。

实验预习

直流无刷电机的参数：

尺寸	27.5×30mm	轴径	3.17mm
工作电压	12V	工作电流	4-10A
螺旋桨直径	50mm	KV 值	100

直流无刷电机工作原理：

此实验箱采用的是型号为新西达 A2212 的直流无刷电机，该电机具有高效率、超强劲、超暴力的性能。

直流无刷电机是同步电机的一种，也就是说电机转子的转速受电机定子旋转磁场的速度及转子极数（P）的影响：

$$N = \frac{120 \bullet f}{P}$$

在转子极数固定的情况下，改变定子旋转磁场的频率就可以改变转子的转速。

直流无刷电机主要由电动机本体、位置传感器和电子开关线路三部分组成。其定子绕组一般制成多相（三相、四相、五相不等），转子由永久磁钢按一定极对数（2P=2,4，…）组成，此电机为三相直流无刷电机。无刷直流电机的运行需依靠转子位置传感器检测出转子的位置信号，通过换相驱动电路驱动与电枢绕组连接的哥功率开关管的导通与关闭，从而控制定子绕组的通电，在定子上产生旋转磁场，拖动转子旋转。关于电机运行的换相步骤，需严格按照以下的换相顺序如图 3-2-1 所示，应用中需要调换电机的转动方向，只需把电机的任意两根相线对调即可。

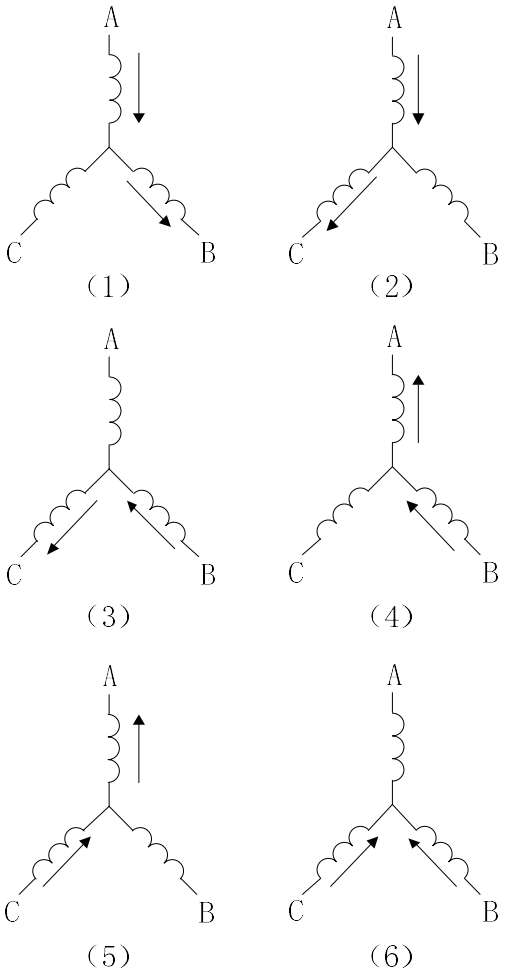


图 3-3-1 换相顺序图

直流电机的调速是通过直流电压来控制的，电压越高，转得越快；电压越低，转得越慢。不过只依靠单片机使不能输出可调的直流电压，于是只好变通，用脉宽调制（PWM）方式控制电机的输入电压。PWM 占空比越高，等效电压越高，占空比越低，等效电压就越低。

步进电机是将电脉冲信号转变为角位移或线位移的开环控制电机。在非超载的情况下，电机的转速、停止位置只取决于脉冲信号的频率和脉冲数，而不受负载变化的影响，当步进驱动器接收到一个脉冲信号，它就驱动步进电机按设定的方向转动一个固定的角度，称为“步距角”，它的旋转是以固定的角度一步一步运行的。可以通过控制脉冲个数来控制角位移量，从而达到准确定位的目的；同时可以通过控制脉冲频率来控制电机转动的速度和加速度，从而达到调速的目的。

当然，单片机给出的 PWM 波形只是控制信号，其能量并不足以驱动无刷直流电机，且直流无刷电机自身不能完成换相任务，只有通过电调的作用。电调是电子调速器的简称，电调都是用 PWM 方式来调速，主要作用就是调速，当然，运用于无刷电机中海油电子换相的作用。此实验箱采用的型号为好盈天行者（SKYWALKER）的电调，产品规格如下所示：

（1） 输出能力：持续电流 30A，短时电流 40A（不少于 10 秒）；

（2） 电源输入：2-3 节锂电池组或 5-9 节镍氢/镍镉电池组；

（3） BEC 输出：5V@2A；

（4） 油门信号频率范围：50Hz-432Hz；

（5） 最高转速：2 极马达 210000 转/分钟，6 极马达 70000 转/分钟，12 极马达 35000 转/分钟；

（6） 尺寸：68mm*25mm*8mm。

电调一边的 3 根线是连接电机，连接顺序可随意（其顺序只影响电机的转向）；另一边的两根电源线和 3 根信号线，3 根信号线分别输出电压（5V），接收脉冲信号（PWM），以及地线。

同时，这里运用一个型号为 APM4953 的双 P 沟道增强型场效应管，主要是用来做 PWM 开关，具体技术参数与原理参考本文件目录下的“APM4953.pdf”。APM4953 的管脚分布图如图 3-2-2 所示。

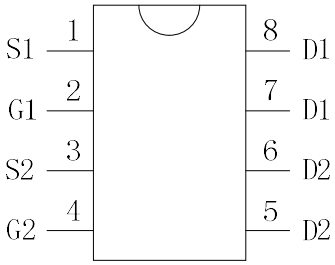


图 3-3-2 APM4953 管脚分布图

通过 STM32 中央微处理器发送高电平信号导通三极管（型号为 9013），三极管导通从而导通 QPM4953，电调电源信号导通，同时 STM32 中央微处理器给予电调 PWM 信号，便可驱动电机转动。其电路原理如图 3-2-3 所示。

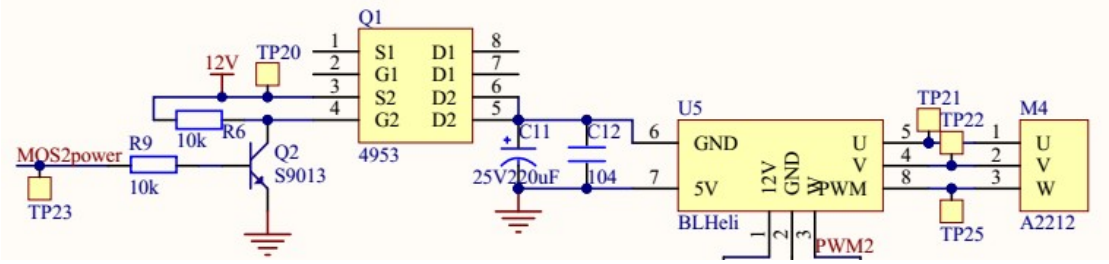


图 3-3-3 电路原理图

实验内容

[1]新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

[3]配置 GPIO 口工作模式。

[4]配置 TIM1、TIM2 模块，实现 PWM 方式驱动电机。

[5]编写主函数。

[6]使用 STM32 ST-LINK Utility 软件下载程序。

[7]通过调节电机调速装置，改变电机的速度，观察实验现象。

实验步骤

[1]新建一个 Keil 工程。

新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能 GPIOA、GPIOB、GPIOE 时钟：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOE|RCC_APB2Periph_ADC1, ENABLE)；

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3|RCC_APB1Periph_TIM4, ENABLE);

库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3]配置 GPIO 口工作模式。

1)恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();

2)配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

3)配置各端口：

第一步：

由于 PA4、PA5、PB2 都是外部模块的电源控制端，所以均需要配置成输出低电平，可以降低系统的功耗。以 A 端口为例：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4|GPIO_Pin_5;//PA4 PA5
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//推挽输出
GPIO_Init(GPIOA, &GPIO_InitStructure);
GPIO_ResetBits(GPIOA, GPIO_Pin_4|GPIO_Pin_5);
GPIO_InitStructure.GPIO_Pin =GPIO_Pin_2;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//推挽输出
GPIO_Init(GPIOB, &GPIO_InitStructure);
GPIO_ResetBits(GPIOB,GPIO_Pin_2);
```

第二步：

本实验用到的电机驱动管脚是 PA11、PB4、PB9，配置如下：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;//PA11-PWM2
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;//复用推挽输出
GPIO_Init(GPIOA, &GPIO_InitStructure);
GPIO_InitStructure.GPIO_Pin =GPIO_Pin_4|GPIO_Pin_9;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//推挽输出
GPIO_Init(GPIOB, &GPIO_InitStructure);
GPIO_ResetBits(GPIOB,GPIO_Pin_4|GPIO_Pin_9); //PB4，PB9 初始化，置低
```

[4]配置 TIM1 模块，实现 PWM 方式驱动电机。

TIM1 模块配置参考本目录下参考程序。

用 PWM 驱动电机，需要定时器功能映射到 IO 口上，而且需要重新配置电机驱动管脚，程序如下：

```
TIM_TimeBaseStructure.TIM_Period=50000-1; //周期 20ms//20000--20MS
TIM_TimeBaseStructure.TIM_Prescaler=71;  //设置预分频：预分频=72，即为 72/72=1MHz,1 个时钟 1uS
TIM_TimeBaseStructure.TIM_ClockDivision=0;  //设置时钟分频系数：不分频
TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up; //向上计数溢出模式
TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);
TIM_OCInitStructure.TIM_OCMode=TIM_OCMode_PWM1; //配置为 PWM 模式 1
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse=val1; //设置通道 1 的电平跳变值，输出另外一个占空比的 PWM
```

```
TIM_OCInitStructure.TIM_OCPolarity=TIM_OCPolarity_High; //当定时器计数值小于 CCR1 时为高电平
TIM_OC1Init(TIM1,&TIM_OCInitStructure); //使能通道 1
TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse=val4; //设置通道 4 的电平跳变值，当计数器计数到这个值时，电平发生跳变
TIM_OCInitStructure.TIM_OCPolarity=TIM_OCPolarity_High; //当定时器计数值小于 CCR4 时为高电平
TIM_OC4Init(TIM1,&TIM_OCInitStructure);  //使能通道 4
TIM_OC4PreloadConfig(TIM1, TIM_OCPreload_Enable);
TIM_ARRPreloadConfig(TIM1,ENABLE);  //使能 TIM1 重载寄存器 ARR
TIM_Cmd(TIM1,ENABLE);          //使能 TIM1
TIM_CtrlPWMOutputs(TIM1,ENABLE);
```

[5]编写主函数。

```
GPIO_SetBits(GPIOB, GPIO_Pin_4);  // PB 置高电平
USART_Config(USART1, 115200);
TIM2_Config(999,359);
TIM1_PWM_Config(0,1000);
AD_Single_Config();
while(1)
 {
  if(tim2_cntr>=20)
    {
            tim2_cntr=0;
            tim_start++;
            GPIOB->ODR^=(1<<9);
        if((brushless_flag==1)||(tim_start>=50))
                            {
                                    brushless_flag=1;
                                    brushless_deal();
                            }
    }
    if(rcv1_flag==1){rcv1_flag=0;com1_order();}
    delay_s(0x7f);
 }
```

[6]使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 Brushless.hex 文件烧写到 RobotTTP。

[7]通过调节电机调速装置，改变电机的速度，观察实验现象。

调节实验箱上的电机调速模块中的电位器，可以改变电机的转速大小。调节装置如果 3-3-4 所示。

3-3-4 电机调速装置

参考程序

参考程序的工程名为“Brushless.uvproj”，在本实验文件目录下“Brushless”文件夹下。

实验小结

通过对本节的学习，了解直流无刷电机的内部结构、运行机理、驱动控制原理。学会电机调速控制。

思考题

1、通过实验确定直流无刷电机的最大转速。

2、测量电机三相电（A、B、C），观察其现象。

3、如何通过程序控制电机从最小转速达到最大转速？

3.4 数字舵机

实验目的

了解数字舵机的内部构造及其工作原理。

熟悉和掌握基于 RobotTTP 中数字舵机的使用方法。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）。

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件。

实验预习

该实验平台采用的型号为 SR518D 的数字舵机，该舵机属于一种集电机、伺服驱动、总线式通讯接口为一体的集成伺服单元，主要用于微型机器人的关节、轮子、履带驱动，也可用于其他简单位置控制场合。SR518D 的特点如下所示：

（1） 扭矩大；

（2） 精度高；

（3） 散热效果突出；

（4） 总线连接，理论上可串接 254 个单元；

（5） 通讯波特率高达 1M，兼容 Robotis Dvnamixel 通讯协议；

（6） 具备位置、温度、速度、电压反馈。

该舵机具体技术参数如下表 1 所示：

尺寸（mm）	50.9*35.7*36	重量（g）	75
类型	智能	运转角度（ °）	300
脉冲范围（ms）	可连续旋转	齿轮数量（个）	6
7.4V	速度 sec/60°	0.21	
	扭矩 kg•cm	17.3	

表 1 技术参数

数字舵机工作原理：

数字舵机主要是由马达、减速齿轮、控制电路等组成，数字舵机与模拟舵机的最大不同在于控制电路上，数字舵机在控制电路上添加了微处理器和晶振。

数字舵机区别于传统的模拟舵机，模拟舵机需要给它不停的发送 PWM 信号，才能使它保持在规定的位置或者让它按照某个速度转动，数字舵机则只需要发送一个 PWM 信号就能保持在规定的某个位置。

数字舵机有两大优势：第一，因为微处理器的关系，数字舵机可以在将动力脉冲发送到舵机马达之前，对输入的信号根据设定的参数进行处理。这意味着动力脉冲的宽度，即激励马达的动力，可以根据微处理器的程序运算而调整，以适应不同的功能

要求，并优化舵机的性能。第二，数字舵机以较高的频率向马达发送动力脉冲。虽然，频率提高了，每个动力脉冲的宽度被减小了，但是马达在同一时间内接收到更多的激励信号，转动变的更快。同时也意味着不仅仅舵机马达以更高的频率响应发射信号，而且“无反应区”变小；反应变得更快；加速和减速也更迅速。

S518D 数字舵机电气接口如下图 3-4-1 所示，两组引脚定义一致的接线端子可将舵机逐机串联起来。

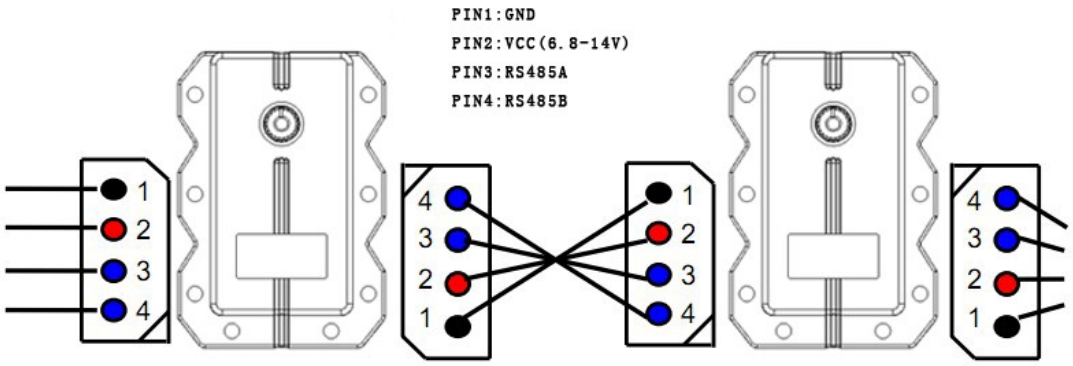


图 3-4-1 数字舵机电气特性图

舵机采用异步串行总线通讯方式，理论上至多 254 个舵机可以通过总线组成链型，通过 UART 异步串行接口统一控制。每个舵机可以设定不同的节点位置，多个舵机可以统一运动也可以单个独立控制。舵机的通讯指令集开放，通过异步串行接口与用户的上位机（控制器或 PC 机）通讯。当然可以对其进行参数设置、功能控制。通过异步串行接口发送指令，舵机可以设置为电机控制模式或位置控制模式。在电机控制模式下，舵机可以作为直流减速电机使用，速度可调；在位置控制模式下，该舵机在 0-300° 范围内转动，在此范围内具备精确位置控制性能，速度可调。

只要符合协议的半双工 UART 异步接口都可以和舵机进行通讯，对舵机进行各种控制。主要有一下两种方式：

方式一：通过调试器控制

PC 机会将调试器识别为串口设备，上位机软件通过串口发送符合协议格式的数据包，经调试器转发给舵机。舵机会执行数据包的指令，并且返回应答数据包。

方式二：通过专用控制器控制

方式一可以快捷地调试舵机，修改各种性能和功能参数，但是，这种方式离不开 PC 机，不能搭建独立的模型。可通过设计专用的控制器，通过控制器的 UART 端口控制舵机。

RS-485 是美国电气工业联合会制定的利用平衡双绞线作传输线的多点通讯标准，采用的是差分信号进行传输。SP485 接口芯片是 RS-485 芯片的一种，工作电压为 +5V，采用半双工通讯方式。它完成将 TTL 电平转换为 RS-485 电平的功能，该芯片具体的技术参数和原理参考本文件目录下的“SP485.pdf”其管脚分布如图 3-4-2 所示。

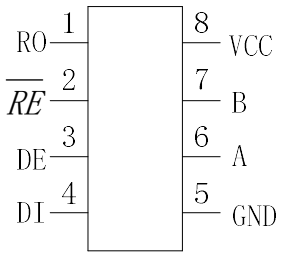


图 3-4-2 SP-485 管脚分布图

通过 STM32 中央微处理器串口 3 发送数据到 SP485，经调试器转发给舵机，实现对舵机的控制，其电路原理如图 3-4-3 所示。

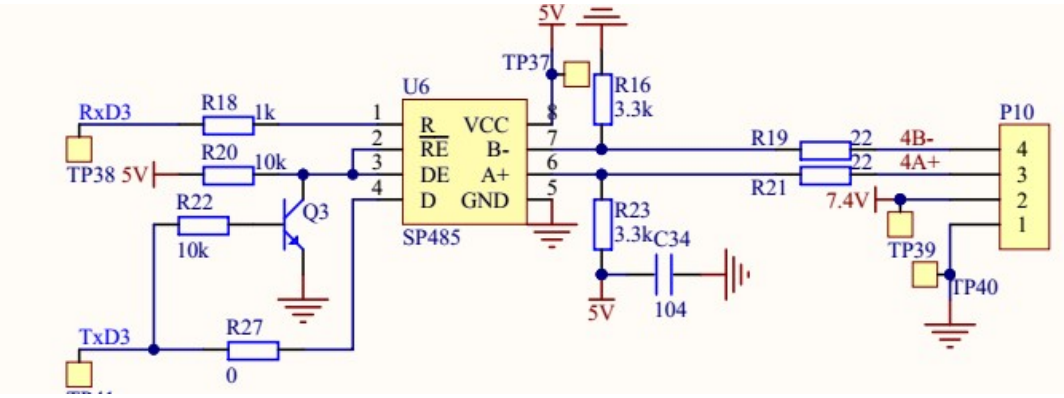


图 3-4-3 电路原理图

实验内容

[1]新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

[3]配置 GPIO 口、串口工作模式。

[4]编写主函数。

[5]使用 STM32 ST-LINK Utility 软件下载程序。

[6]通过调节舵机位置控制装置，改变舵机转动位置，观察实验现象。

实验步骤

[1]新建一个 Keil 工程。

新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能 GPIOA、GPIOB、GPIOE 时钟：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOE|RCC_APB2Periph_ADC1, ENABLE)；

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3|RCC_APB1Periph_TIM4, ENABLE);

库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3]配置 GPIO 口、串口工作模式。

1)恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();

2)配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

3)配置各端口：

第一步：

由于 PA4、PA5、PB12、PB13、PB14、PB15 都是外部模块的电源控制端，所以均需要配置成输出低电平，可以降低系统的功耗。以 A 端口为例：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4|GPIO_Pin_5;//PA4 PA5
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//推挽输出
GPIO_Init(GPIOA, &GPIO_InitStructure);
GPIO_ResetBits(GPIOA, GPIO_Pin_4|GPIO_Pin_5);
```

第二步：

本实验用到管脚 PB2、PB10，配置如下：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;//PB10-TXD3,
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;//复用推挽输出
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

```
GPIO_InitStructure.GPIO_Pin=GPIO_Pin_2;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//推挽输出
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

4)配置串口：

```
void USART_Config(USART_TypeDef* USARTx,u32 baudrate)
{
USART_InitTypeDef USART_InitStructure;
USART_DeInit(USARTx);//复位串口 1 寄存器
USART_InitStructure.USART_BaudRate = baudrate;//波特率 115200
USART_InitStructure.USART_WordLength = USART_WordLength_8b; //数据位 8 位
USART_InitStructure.USART_StopBits = USART_StopBits_1;//停止位 1 位
USART_InitStructure.USART_Parity = USART_Parity_No; //无校验位
USART_InitStructure.USART_HardwareFlowControl                                    =
USART_HardwareFlowControl_None;   //无硬件流控
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;    // 收
发模式
USART_Init(USARTx, &USART_InitStructure);  //配置串口参数
USART_ITConfig(USARTx, USART_IT_RXNE, ENABLE); //使能接收中断
USART_ClearFlag(USARTx, USART_FLAG_TC );//清除发送完成标志位
USART_Cmd(USARTx, ENABLE);//使能串口
}
```

[4]编写主函数。

```
GPIO_SetBits(GPIOB, GPIO_Pin_2);
USART_Config(USART1, 115200);
USART_Config(USART3, 115200);
TIM2_Config(999,359);
AD_Single_Config();
GPIO_SetBits(GPIOB, GPIO_Pin_9);
delay_s(0xaffffff);    //延时等待舵机上电稳定，最少 0x7fffff
delay_s(0xaffffff);
delay_s(0xaffffff);
delay_s(0xaffffff);
delay_s(0xaffffff);
 while(1)
 {
   if(tim2_cntr>=20)
            {
                    tim2_cntr=0;
                    GPIOB->ODR^=(1<<9);
                    sr518_deal();
```

```
            }
    if(rcv1_flag==1){rcv1_flag=0;com1_order();}
    if(rcv3_flag==1){rcv3_flag=0;com3_order();}
    delay_s(0x7f);
  }
```

[5]使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 Duoji.hex 文件烧写到 RobotTTP。

[6]通过调节舵机位置控制装置，改变舵机转动位置，观察实验现象。

调节实验箱上的电机调速模块中的电位器，可以改变舵机的转动位置。调节装置如果 3-4-4 所示。

3-4-4 电机调速装置

参考程序

参考程序的工程名为“Duoji.uvproj”，在本实验文件目录下“Duoji”文件夹下。

实验小结

通过对本节的学习，了解数字舵机的工作原理以及通讯方式，同时学习串口通讯的原理 及其方法。

思考题

如何通过串口调试软件直接控制舵机转动？

3.5 机械手臂

实验目的

了解机械手臂的组成及性能指标。

了解机械手臂的机械系统和控制系统的组成、工作原理。

实验设备

硬件：PC 机（一台），RobotTTP（一台），ST-LINK 调试编程器（一个）。

软件：Keil uVision4 开发工具，STM32 ST-LINK Utility 软件。

实验预习

该机械手臂的具体技术参数如下表 1 所示：

高度（mm）	210	旋转半径（mm）	150
工作电压（V）	5	机构极限角度（°）	180
机械爪	长度（mm）	102	
	宽度（mm）	105	
	厚度（mm）	20	

表 1 机械手臂技术参数

该机械手臂是三自由度的串联式结构，机械手臂的各关节采用的是模拟舵机，型号为 MG996R，该模拟舵机的具体参数如下表 2 所示：

重量（g）	55	工作电压（V）	4.8-7.2
尺寸（mm）	40．7*19.7*42.9	工作电流（mA）	>100
反应速度（sec/60°）	0.17（4.8V） 0.14（6V）	产品拉力（kg/cm）	9.4（4.8V） 11（6V）
工作死区（us）	5	齿轮形式	金属齿轮

表 2 舵机技术参数

模拟舵机工作原理：

模拟舵机是一种位置伺服的驱动器，适用于那些需要角度不断变化并保持的控制系统。其工作原理是：控制信号由接收机的通道进入信号调制芯片，获得直流偏置电压。它内部有一个基准电路，产生周期为 20ms，宽度为 1.5ms 的基准信号，将获得的直流偏置电压与电位器的电压比较，获得电压差输出。最后，电压差的正负输出到电机驱动芯片决定电机的正反转。当电机转速一定时，通过级联减速齿轮带动电位器旋转，使得电压差为 0，电机停止转动。

MG996R 舵机有三根连接线，红色为电源线，棕色为 GND，黄色为信号线。驱动舵机转动需要 PWM 信号，单片机 STM32 作为其控制单元，输出 PWM 信号，其电路原理如图 3-5-1 所示。

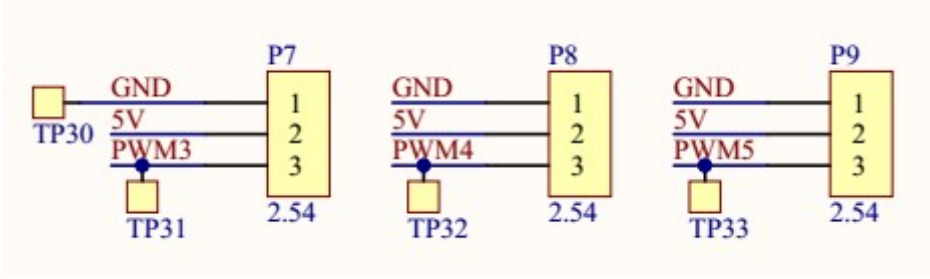


图 3-5-1 电路原理图

机械手臂的三自由度的限制为：

咬合舵机：1200-1949 (1200 闭合，1949 完全张开)

伸直舵机：1199-2149 (1199 向上伸直，2149 向下贴到底盘)

旋转舵机：600-2160（旋转 180°）

实验内容

[1]新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

[3]配置 GPIO 口工作模式。

[4]配置 TIM1 模块，实现 PWM 方式驱动舵机

[5]编写主函数。

[6]使用 STM32 ST-LINK Utility 软件下载程序。

[7]改变 PWM 占空比，使机械手机转动到不同位置，观察实验现象。

实验步骤

[1]新建一个 Keil 工程。

新建一个 Keil 工程。

[2]配置 STM32 系统时钟、GPIO 时钟。

1)恢复 RCC 时钟到默认值：RCC_DeInit()；

2)开启外部高速时钟：RCC_HSEConfig(RCC_HSE_ON)；

3)等待外部晶振启动：RCC_WaitForHSEStartUp()；

4)代码延时两个个周期：FLASH_SetLatency(FLASH_Latency_2)；

5)半周期访问失能：FLASH_HalfCycleAccessCmd(FLASH_HalfCycleAccess_Disable)；

6)预取指缓存使能：FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable)；

7)选择 HSE 的 9 倍频作为 PLL 时钟：RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9)；

8)使能 PLL：RCC_PLLCmd(ENABLE)；

9)PLL 作为系统时钟：RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK)；

10)系统时钟作为 AHB 时钟：RCC_HCLKConfig(RCC_SYSCLK_Div1)；

11)AHB 时钟的一半作为 APB1 时钟：RCC_PCLK1Config(RCC_HCLK_Div2)；

12)AHB 时钟作为 APB2 时钟：RCC_PCLK2Config(RCC_HCLK_Div1)；

13)APB2 时钟的六分之一作为 ADC 时钟：RCC_ADCCLKConfig(RCC_PCLK2_Div6);

14)关闭外部低速时钟：RCC_LSEConfig(RCC_LSE_OFF)；

15)使能 GPIOA、GPIOB、GPIOE 时钟：RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOE|RCC_APB2Periph_ADC1, ENABLE)；

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3|RCC_APB1Periph_TIM4, ENABLE);

库函数的详细使用说明，参照本文件目录下“STM32 固件库使用手册_v3.5 版本.pdf，页码 193-213”。

[3]配置 GPIO 口、串口工作模式。

1)恢复 GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、AFIO 寄存器到默认值：GPIO_DeInit(GPIOA);GPIO_DeInit(GPIOB);GPIO_DeInit(GPIOC);GPIO_DeInit(GPIOD);GPIO_DeInit(GPIOE);GPIO_AFIODeInit();

2)配置 JTAG 使能、SWD 使能：GPIO_PinRemapConfig(GPIO_Remap_SWJ_JTAGDisable, ENABLE);

3)配置各端口：

第一步：

由于 PA1、PA3、PA4、PA5、PB12、PB13、PB14、PB15 都是外部模块的电源控制端，所以均需要配置成输出低电平，可以降低系统的功耗。以 A 端口为例：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4|GPIO_Pin_5;//PA4 PA5
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//推挽输出
GPIO_Init(GPIOA, &GPIO_InitStructure);
GPIO_ResetBits(GPIOA, GPIO_Pin_4|GPIO_Pin_5);
```

第二步：

本实验用到管脚 PE9、PE11、PE13，配置如下：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9|GPIO_Pin_11|GPIO_Pin_13;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;//复用推挽输出
GPIO_Init(GPIOE, &GPIO_InitStructure);
```

4)配置串口：

```
void USART_Config(USART_TypeDef* USARTx,u32 baudrate)
{
USART_InitTypeDef USART_InitStructure;
```

USART_DeInit(USARTx);//复位串口 1 寄存器
USART_InitStructure.USART_BaudRate = baudrate;//波特率 115200
USART_InitStructure.USART_WordLength = USART_WordLength_8b; //数据位 8 位
USART_InitStructure.USART_StopBits = USART_StopBits_1;//停止位 1 位
USART_InitStructure.USART_Parity = USART_Parity_No; //无校验位
USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowControl
_None; //无硬件流控
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx; //收发模式
USART_Init(USARTx, &USART_InitStructure); //配置串口参数
USART_ITConfig(USARTx, USART_IT_RXNE, ENABLE); //使能接收中断
USART_ClearFlag(USARTx, USART_FLAG_TC);//清除发送完成标志位
USART_Cmd(USARTx, ENABLE);//使能串口
}

[4]配置 TIM1 模块，实现 PWM 方式驱动舵机。
TIM_TimeBaseStructure.TIM_Period=20000-1; //周期 20ms
TIM_TimeBaseStructure.TIM_Prescaler=71; //设置预分频：预分频=72，即为 72/72=1MHz,1 个时钟 1uS
TIM_TimeBaseStructure.TIM_ClockDivision=0; //设置时钟分频系数：不分频
TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up; //向上计数溢出模式
TIM_TimeBaseStructure.TIM_RepetitionCounter=0;
TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_OutputNState=TIM_OutputNState_Enable;
TIM_OCInitStructure.TIM_Pulse=1199; //设置通道 1 的电平跳变值，输出另外一个占空比的 PWM
TIM_OCInitStructure.TIM_OCPolarity=TIM_OCPolarity_High; //当定时器计数值小于 CCR1 时为高电平范围在 600-2160
TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
TIM_OC1Init(TIM1,&TIM_OCInitStructure); //使能通道 1
TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_OutputNState=TIM_OutputNState_Enable;
TIM_OCInitStructure.TIM_Pulse=1199; //设置通道 2 的电平跳变值，当计数器计数到这个值时，电平发生跳变
TIM_OCInitStructure.TIM_OCPolarity=TIM_OCPolarity_High; //当定时器计数值小于 CCR2 时为高电平范围在 1199-2149

```
TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
TIM_OC2Init(TIM1,&TIM_OCInitStructure);      //使能通道 2
TIM_OC2PreloadConfig(TIM1, TIM_OCPreload_Enable);
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_OutputNState=TIM_OutputNState_Enable;
TIM_OCInitStructure.TIM_Pulse=1199; //设置通道 3 的电平跳变值，输出另外一个占空比的 PWM
TIM_OCInitStructure.TIM_OCPolarity=TIM_OCPolarity_High; //当定时器计数值小于 CCR3 时为高电平范围在 945-1949
TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
TIM_OC3Init(TIM1,&TIM_OCInitStructure); //使能通道 3
TIM_OC3PreloadConfig(TIM1, TIM_OCPreload_Enable);
TIM1->CR2=0;
TIM_ARRPreloadConfig(TIM1,ENABLE);  //使能 TIM1 重载寄存器 ARR
TIM_Cmd(TIM1,ENABLE);          //使能 TIM1
TIM_CtrlPWMOutputs(TIM1, ENABLE);
```

[5]编写主函数。

```
NVIC_Configuration();
USART_Config();
TIM1_PWM_Config();
GPIO_SetBits(GPIOB,GPIO_Pin_9);
USART_SendData(USART1,0x11);//发送回复帧
while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
USART_SendData(USART1,0x12);//发送回复帧
while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
USART_SendData(USART1,0x11);//发送回复帧
while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
USART_SendData(USART1,0x12);//发送回复帧
 while(1)
 {
   if(rcv_flag==2){rcv_flag=0;com1_order();}
   delay_s(0x1ff);
  }
```

[6]使用 STM32 ST-LINK Utility 软件下载程序。

将本工程生成的 jxs.hex 文件烧写到 RobotTTP。

[7]改变 PWM 占空比，使机械手机转动到不同位置，观察实验现象。

根据各个舵机的范围，改变其 PWM 占空比，转动机械手臂到不同的位置，观察其实验现象。

参考程序

参考程序的工程名为“jxs.uvproj”，在本实验文件目录下“Jixieshoubi”文件夹下。

实验小结

通过对本节的学习，了解机械手臂的组成，学习模拟舵机的工作原理以及控制方式。

思考题

1、与数字舵机进行对比，学习其两者的不同之处。

2、如何通过动作调试软件进行对机械手臂的控制。

4.基于树莓派的机器人图像处理系统

目前，图像处理已经在许多不同的应用领域中得到重视，并取得了巨大成就。根据应用领域要求的不同，图像处理分为许多分支技术，例如：图像变换、图像增强与复原、图像压缩编码、图像分割等。不同的图像处理技术应用与不同的领域，发展出不同的分支学科，如遥感图像处理、医学图像处理等，其它如计算机图形学、模式识别、人工智能和机器人视觉等学科领域也与图像处理有着密切的关系。

随着科技的发展，人们对于图像的处理可以概括为以下三个目的：

（1）提高图像的视感质量。如去除图像中的噪声，改变图像中的亮度和颜色，增强图像中的某些成分与抑制某些成分，对图像进行几何变换等。

（2）提取图像中所包含的某些特征或特殊信息。如常用作模式识别和计算机视觉的预处理等，这些都便于计算机进行分析。

（3）对图像数据进行变换、编码和压缩，以便于图像的存储和传输。

此实训教学平台是基于树莓派（Raspberry Pi）进行的图像处理，既学习了嵌入式中树莓派的使用方法，又学习了图像处理的基本知识。本实验教程包含了树莓派 GPIO 口的控制实验，摄像头操作使用，图像二值化处理实验以及人脸检测实验。

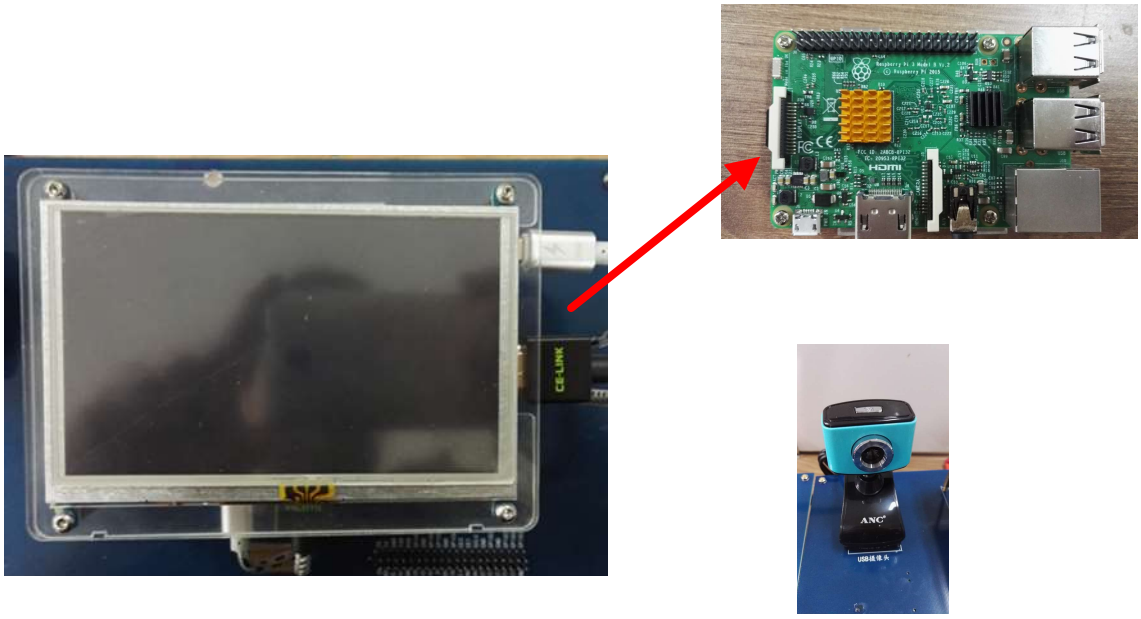


图 4-1 嵌入式图像系统结构图

本实验教程对初学者要求是应具有基本的树莓派操作使用基础，对编程语言、控制指令等应有基本的了解。

4.1 GPIO 控制 LED 灯

实验目的

了解树莓派基本的使用方法。

学习并掌握树莓派 GPIO 的操作过程。

实验设备

硬件：PC 机（一台），RobotTTP（一台），mini 键盘。

软件：Terminal 软件。

实验预习

Raspberry Pi 本质上就是一台廉价的 Linux 电脑，但是与我们平时用于上网、写邮件或进行文字处理的台式电脑或笔记本仍有一些不同之处。其中最明显的一个差别就是，Raspberry Pi 的主板上提供了一组 GPIO 接口，这些接口可以直接用于开发一些电子相关的项目，如图 4-1-1 所示。

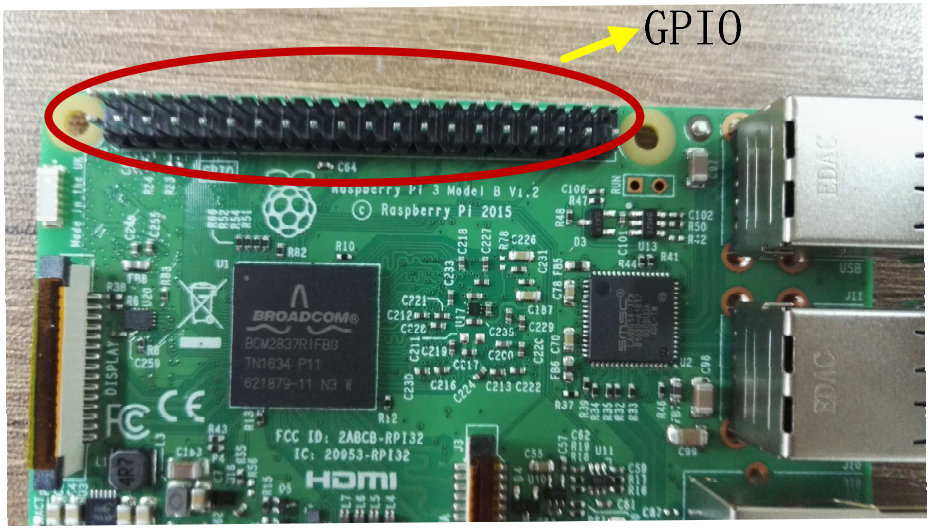


图 4-1-1 GPIO 接口

这组 GPIO 一共有 40 个引脚，不同的引脚有不同的作用，具体引脚说明如下图 4-1-2 所示。

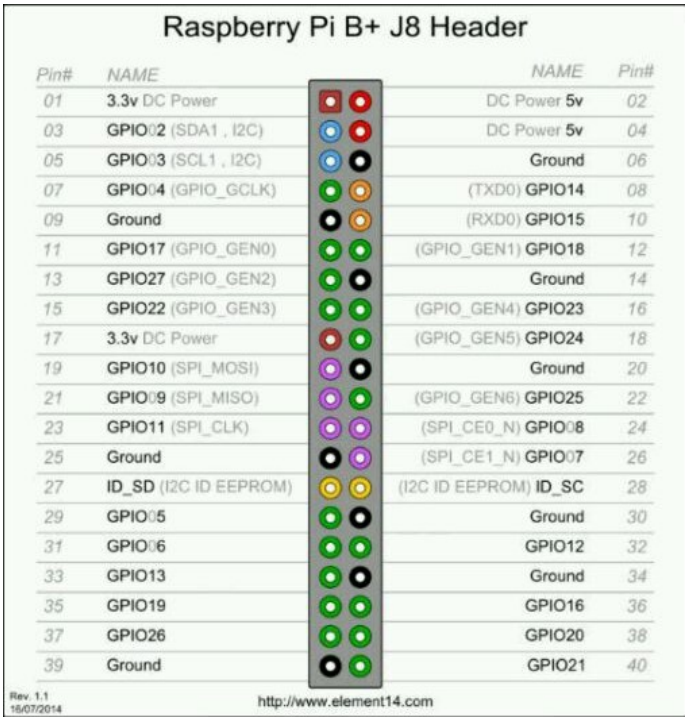


图 4-1-2 树莓派引脚图

这些 GPIO 接口可以作为控制信号输出口来控制一些硬件设备，如发光二极管（LED）、电机或继电器。它们也可以作为信号输入接口，用于读取按键或开关的状态，获取温度、光线、运动和距离等各种传感器的状态。

如果电脑具备 GPIO，它的最大优势在于可以编写程序从 GPIO 口上读取状态或根据不同的条件向 GPIO 口输出值，就与给普通台式电脑编写程序一样方便。Raspberry Pi 与其它设备 GPIO 口的单片机开发板不同之处在于，它还具备键盘、鼠标、显示器及以太网等输入输出设备。

具备键盘、鼠标、显示器等接口不是 Raspberry Pi 区别于其它单片机开发板的唯一优点，在开发电子项目时，它还具备以下优点。

（一） 文件系统

可以直接读写 Linux 文件系统，会使很多项目实现起来更为容易。

（二） Linux 工具

Raspberry Pi 的 Linux 发行版中包含了一系列的命令行工具，可以使用这些工具来操作文件、控制进程或把很多任务进行自动化处理。这些灵活的工具都可以在项目中直接使用。

（三） 编程语言

在 Raspberry Pi 这样的嵌入式 Linux 系统中，系统支持多种开发语言，包括 C、Java、Perl、Shell、Python 等开发语言。

实验内容

[1]准备相关设备。

[2]GPIO 与 LED 连接。

[3]打开终端（Terminal）。

[4]获取 root 权限。
[5]引脚从内核空间暴露到用户空间。
[6]改变目录。
[7]设置 GPIO25 接口为输出模式。
[8]点亮 LED 灯。
[9]熄灭 LED 灯。

实验步骤

[1]准备相关设备。
准备实验所用的杜邦线、mini 键盘。

[2]GPIO 与 LED 连接。
首先确认 LED 的极性，用杜邦线使 LED 的正极与 Raspberry Pi 的 GPIO 25 相连，负极与 Raspberry Pi 的地（GND）相连。

[3]打开终端（Terminal）。
在 Raspberry Pi 桌面双击打开 Terminal 图标，如图 4-1-3 所示进入命令操作窗口。

图 4-1-3 Terminal

[4]获取 root 权限。
为了从命令行操作 Raspberry Pi 的输入输出接口，需要以 root 权限（管理员权限）来进行相关的命令。通过输入 sudo su 并回车，可以把命令行切换到管理员权限模式下：

```
pi@raspberrypi:~ $ sudo su
root@raspberrypi:/home/pi#
```

可以注意到，命令行的提示符发生了变化，提示后续的命令都可以以 root 权限运行。

[5]引脚从内核空间暴露到用户空间。
在开始使用命令来操作 GPIO25 上所连接的 LED 前，需要先把这个引脚的操作接口从内核空间暴露到用户空间中，使用以下命令：

```
root@raspberrypi:/home/pi# echo 25 > /sys/class/gpio/export
```

使用 echo 命令把操作的 GPIO 编号（25）写入/sys/class/gpio 下的 export 文件，写入这个文件之后，/sys/class/gpio 下会自动新建一个 /sys/class/gpio25 目录，里面包含了操作这个 GPIO 口所需的控制文件。

[6]改变目录。

用 cd 命令切换到新建的目录下，并用 ls 命令列出文件清单：

```
root@raspberrypi:/home/pi# cd /sys/class/gpio/gpio25
root@raspberrypi:/sys/class/gpio/gpio25# ls
active_low  device  direction  edge  power  subsystem  uevent  value
```

cd 命令是 Change Directory（改变目录）的缩写，能把当前的工作目录切换到指定的目录下，这样就不需要每次都输入目录的完整路径。ls 命令可以列出目录下的文件和子目录。在这个目录下，需要操作两个文件：direction 和 value。

[7]设置 GPIO25 接口为输出模式。

Direction 文件用于指定这个 GPIO 接口会被用于输入或输出。由于目前需要控制的 LED 连接在 GPIO25 接口上，所以应该把这个接口设置为输出模式：

```
root@raspberrypi:/sys/class/gpio/gpio25# echo out > direction
```

[8]点亮 LED 灯。

使用 echo 命令向 value 文件输出 1，可以点亮这个 GPIO 口上所接的 LED：

```
root@raspberrypi:/sys/class/gpio/gpio25# echo 1 > value
```

[9]熄灭 LED 灯。

```
root@raspberrypi:/sys/class/gpio/gpio25# echo 0 >value
```

实验小结

通过对本节的学习，了解了树莓派的基本操作方法，同时也学习了 GPIO 口的控制使用。

思考题

1、改变 GPIO 接口，实现对 LED 的控制。

2、如何从 GPIO 读入数据。

4.2 摄像头使用

实验目的

了解树莓派基本操作命令。

学习并掌握树莓派摄像头使用方法。

实验设备

硬件：PC 机（一台），RobotTTP（一台），mini 键盘，摄像头。

软件：Terminal 软件，SimpleCV 软件，luvcview 软件。

实验预习

使用 Raspberry Pi 这样的平台开发各种项目一个很大的优势就是可以使用各种 USB 设备。不仅可以连接键盘、鼠标，还可以使用摄像头。该实训平台采用的摄像头像素为 300 万，最大分辨率为 1024*768。

树莓派可以使用两种类型的摄像头，一种为 CSI 接口，一种为 USB 接口。由于 Pi 所使用的 Broadcom 芯片组原本是为手机和平板电脑设计的，CSI 接口就是这些移动设备厂商用来连接摄像头的接口。但是在现实中，CSI 接口的摄像头使用不多，所以我们使用 USB 摄像头。

实验内容

[1]准备相关设备。

[2]USB 摄像头与 Raspberry Pi 连接。

[3]打开终端（Terminal）。

[4]测试摄像头。

[5]安装并测试 SimpleCV。

[6]调用 USB 摄像头。

[7]获取一张拍摄图片。

实验步骤

[1]准备相关设备。

准备实验所用的 USB 摄像头、mini 键盘等设备。

[2]USB 摄像头与 Raspberry Pi 连接。

将 USB 摄像头与 Raspberry Pi 主板上的 USB 相连接。

[3]打开终端（Terminal）。

在 Raspberry Pi 桌面双击打开 Terminal 图标，如图 4-2-1 所示进入命令操作窗口。

图 4-2-1 Terminal

[4]测试摄像头。

市面上有很多型号的摄像头，没办法保证每一种都能直接在 Raspberry Pi 上正常工作，现需要对摄像是否能正常使用进行测试。

（一） 安装 luvcview 软件（已安装好，可以跳过此步骤）

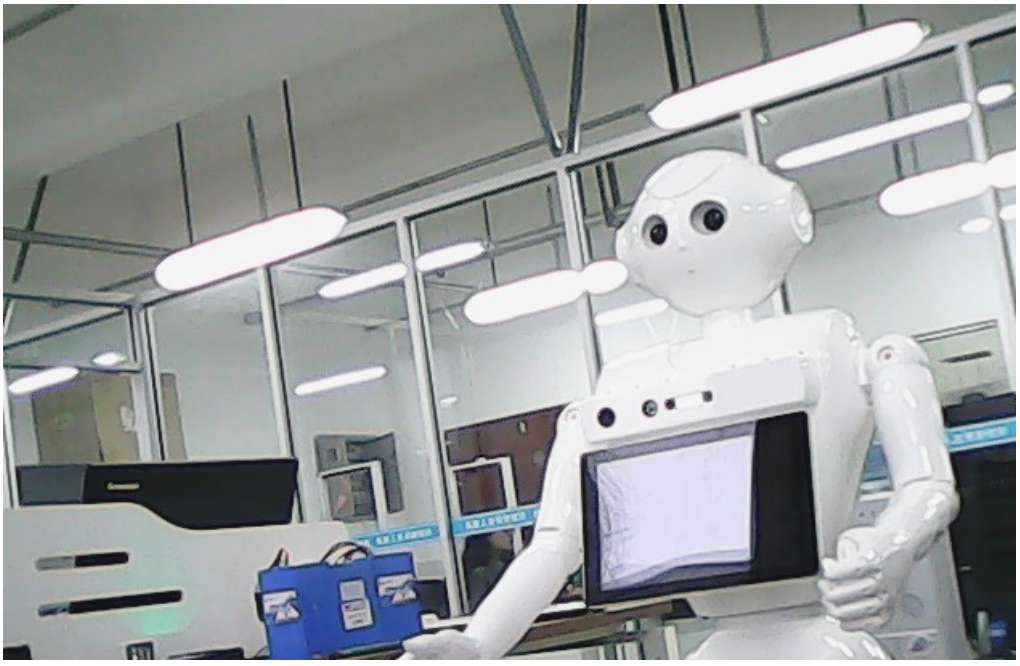
在终端输入以下命令：

```
pi@raspberrypi:~ $ sudo apt-get install luvcview
```

（二） 调用 luvcview 软件

程序安装完成之后，在图形化桌面环境的终端中输入 luvcview 运行这个程序。Luvcview 会打开新的窗口显示摄像头采集到的图像，如图 4-2-2 所示。画图的尺寸会在终端上显示出来。如果显示的视频图像上有一些波纹，可以试着缩小画图的尺寸。例如，默认的视频大小是 640x480，可以进行一下设置：

```
pi@raspberrypi:~ $ luvcview -s 320x480
```



4-2-2 图像显示

[5]安装并测试 SimpleCV。

注：已安装好，可跳过此步骤。

这里将在 Python 中使用 SimpleCV 库来操作摄像头，SimpleCV 是与电脑视觉相关的开源库。通过使用 SimpleCV，可以很方便地从摄像头采集图像并显示在屏幕上或保存为文件。但 SimpleCV 最强大功能在于它包含的计算机视觉相关的算法，可以用来实现一些让人惊叹的效果。除了基本的图像变换，SimpleCV 还可以用来跟踪、检测和识别图像或视频中的特定对象。

在安装 SimpleCV 前，需要安装一些它所依赖的库，安装命令如下所示：

```
pi@raspberrypi:~ $ sudo apt-get install ipython python-opencv python-scipy pytho
n-numpy python-pygame python-setuptools
```

安装 SimpleCV：

```
pi@raspberrypi:~ $ sudo pip install https://github.com/sightmachine/SimpleCV/zip
ball/develop
```

[6]调用 USB 摄像头。

（一） 新建文件夹 simplecv-test。

（二） 在 simplecv-test 文件夹下新建文件 basic-camera.py，双击新建的 basic-camera.py 文件

（三） 输入如下图 4-2-3 所示代码。

basic-camera.py - /home/pi/simplecv-test/basic-camera.py (2.7.9)

File Edit Format Run Options Windows Help

```
from SimpleCV import Camera,Display
from time import sleep

myCamera = Camera(prop_set={"width":320,"height":240})

myDisplay = Display(resolution=(320,240))

while not myDisplay.isDone():
    myCamera.getImage().save(myDisplay)
    sleep(.1)
```

图 4-2-3 代码

（四） 保存 basic-camera.py 文件，在终端运行，摄像头将显示画像，如图 4-2-4 所示。

```
pi@raspberrypi:~ $ cd simplecv-test/
pi@raspberrypi:~/simplecv-test $ python basic-camera.py
```

图 4-2-4 图像

[7]获取一张拍摄图片。

在文件 basic-camera.py 文件下输入以下代码，并在终端运行可捕获一张图片并保存为.jpg 文件，代码如下图 4-2-5 所示：

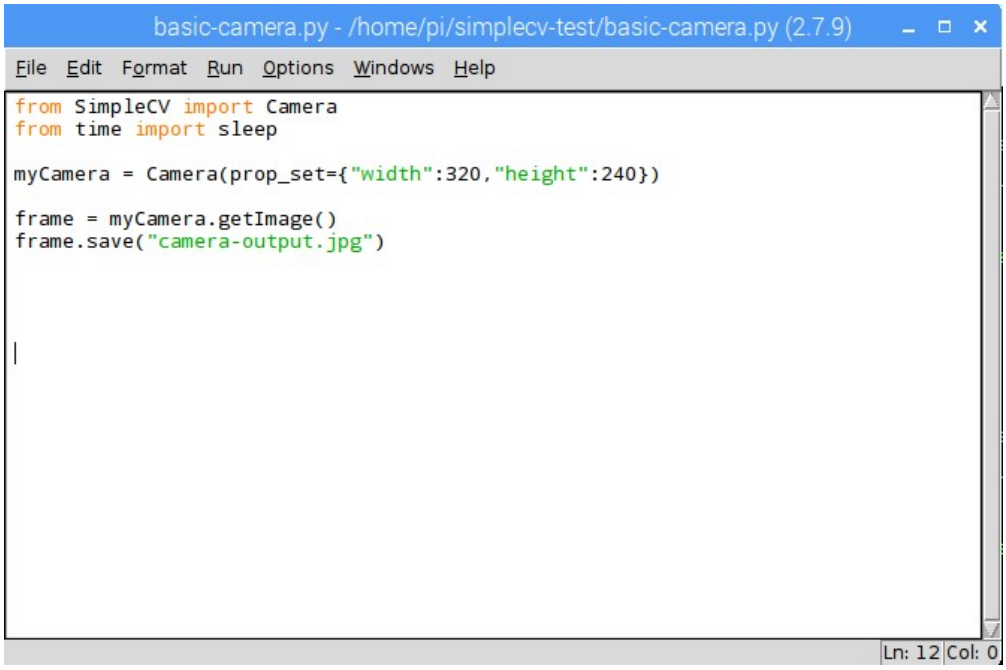


图 4-2-4 代码

实验小结

通过对本节的学习，了解了树莓派的基本操作命令，同时学习了树莓派操作使用 USB 摄像头的方法。

思考题

如何使用 USB 摄像头实现网络监控功能。

4.3 图像处理

实验目的

了解基本的图像形态学知识。

学习并掌握用树莓派对图像进行二值化处理。

实验设备

硬件：PC 机（一台），RobotTTP（一台），mini 键盘，摄像头。

软件：Terminal 软件，SimpleCV 软件。

实验预习

目前，图像处理已经在许多不同的应用领域中得到重视，并取得了巨大成就。根据应用领域要求的不同，图像处理分为许多分支技术，例如：图像变换、图像增强与复原、图像压缩编码、图像分割等。不同的图像处理技术应用与不同的领域，发展出不同的分支学科，如遥感图像处理、医学图像处理等，其它如计算机图形学、模式识别、人工智能和机器人视觉等学科领域也与图像处理有着密切的关系。

二值化是图像分割的一种方法。在二值化图像的时候把大于某个临界灰度值的像素灰度设为灰度极大值，把小于这个值的像素灰度设为灰度极小值，从而实现二值化。

彩色图像中的每个像素的颜色有 R、G、B 三个分量决定，而每个分量有 255 种值可取，这样一个像素点可以有 1600 多万的颜色变化范围。而灰度图像是 R、G、B 三个分量相同的一种特殊的彩色图像，一个像素点的变化范围为 255 种，所以在图像处理中一般先将各种格式的图像转变成灰度图像以使后续的图像的计算量变得少一些。灰度图像的描述与彩色图像一样仍然反映了正幅图像的整体和局部的色度和亮度等级的分布和特征。图像的灰度化处理可先求出每个像素点的 R、G、B 三个分量的平均值，然后将这个平均值赋予给这个像素的三个分量。

图像的二值化处理就是将图像上的点的灰度置为 0 或 255，也就是使整个图像呈现出明显的黑白效果。即将 256 个亮度等级的灰度图像通过适当的阈值选取而获得仍然可以反映图像整体和局部特征的二值化图像。

实验内容

[1]准备相关设备。

[2]获取一张图片。

[3]打开终端（Terminal）。

[4]安装并测试 SimpleCV。

[5]二值化处理。

[6]察看处理后图片。

实验步骤

[1]准备相关设备。

准备实验所用 mini 键盘等设备。

[2]获取一张图片。

树莓派链接网络，下载一些图片，或用本机所有的一些图片。

[3]打开终端（Terminal）。

在 Raspberry Pi 桌面双击打开 Terminal 图标，如图 4-3-1 所示进入命令操作窗口。

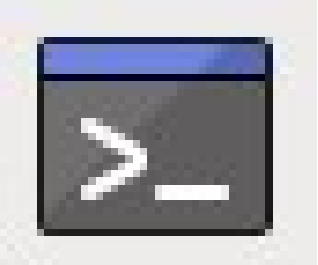


图 4-3-1 Terminal

[4]安装并测试 SimpleCV。

注：已安装好，可跳过此步骤。

这里将在 Python 中使用 SimpleCV 库来操作摄像头，SimpleCV 是与电脑视觉相关的开源库。通过使用 SimpleCV，可以很方便地从摄像头采集图像并显示在屏幕上或保存为文件。但 SimpleCV 最强大功能在于它包含的计算机视觉相关的算法，可以用来实现一些让人惊叹的效果。除了基本的图像变换，SimpleCV 还可以用来跟踪、检测和识别图像或视频中的特定对象。

在安装 SimpleCV 前，需要安装一些它所依赖的库，安装命令如下所示：

```
pi@raspberrypi:~ $ sudo apt-get install ipython python-opencv python-scipy pytho
n-numpy python-pygame python-setuptools
```

安装 SimpleCV：

```
pi@raspberrypi:~ $ sudo pip install https://github.com/sightmachine/SimpleCV/zip
ball/develop
```

[5]二值化处理。

（一） 新建文件夹 simplecv-test。

（二） 在 simplecv-test 文件夹下新建文件 image-processing.py，双击新建的 image-processing.py 文件

（三） 输入如下图 4-3-2 所示代码。

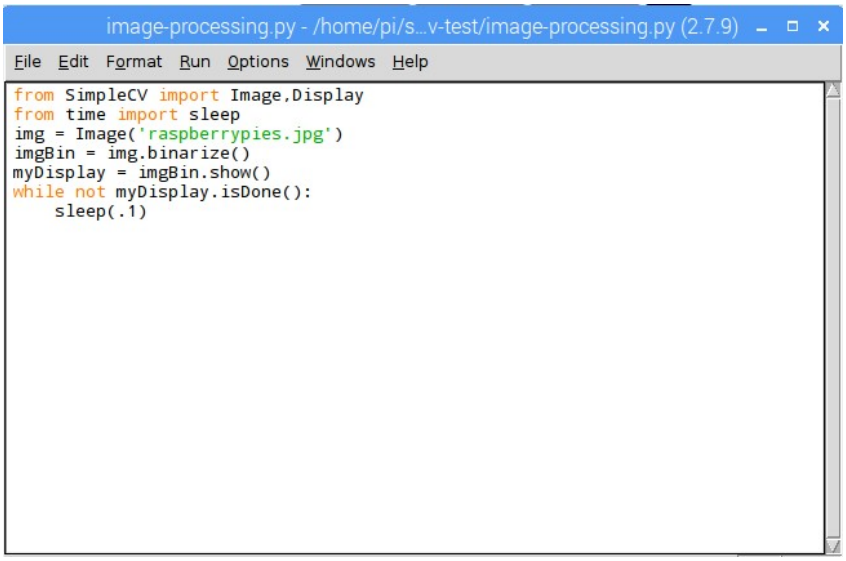


图 4-3-2 代码

（四） 保存 image-processing.py 文件。

[6]察看处理后图片。

原图为“raspberrypies.jpg”，如图 4-3-3 所示。在终端输入以下代码，得到处理后的图片，如图 4-3-4 所示。

```
pi@raspberrypi:~ $ cd simplecv-test/
pi@raspberrypi:~/simplecv-test $ python image-processing.py
```

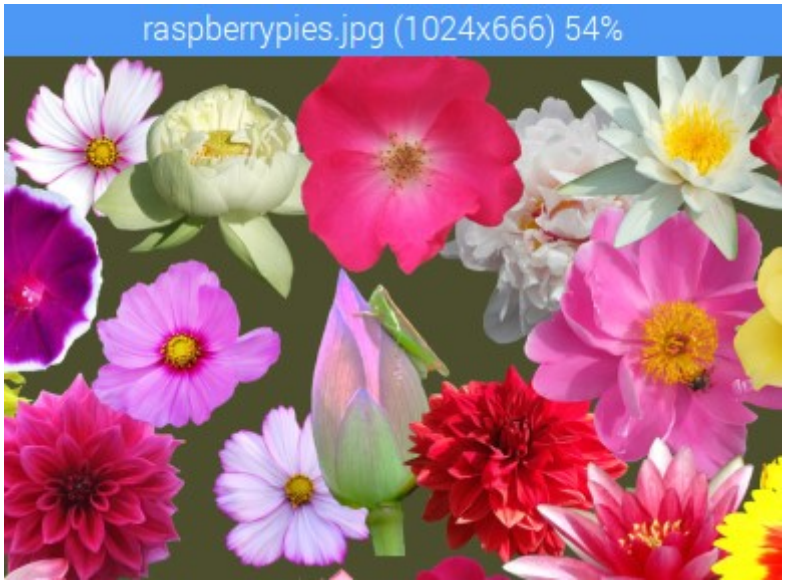


图 4-3-3 原图

图 4-3-4 处理后图片

实验小结

通过对本节的学习，了解图像处理的基本知识，同时掌握了树莓派用于图像的二值化处理的基本方法。

思考题

如何利用 SimpleCV 对图像进行其它技术处理。

4.4 人脸检测

实验目的

了解 SimpleCV 基本函数。

学习并掌握树莓派利用摄像头进行人脸检测。

实验设备

硬件：PC 机（一台），RobotTTP（一台），mini 键盘，摄像头。

软件：Terminal 软件，SimpleCV 软件。

实验预习

人脸检测是机器视觉和模式识别领域最富有挑战性的课题之一，同时也具有较为广泛的应用意义。人脸检测技术是一个非常活跃的研究领域，它覆盖了数字图像处理、模式识别、计算机视觉、神经网络、心理学、生理学、数学等诸多学科的内容。如今，在这方面的研究已经取得了一些成果。

迄今为止，机器视觉的发展已经经历了一个漫长的过程。计算机视觉处理的一个重要内容，就是人脸的视觉处理。人脸分析的相关研究希望用户的身份、状态和意图的信息能够从图像中提取出来，然后计算依此做出反应（比如通过观察用户人脸部表情来分析心情并进行相应反应）。人脸检测技术应用背景十分广泛，可用于公安系统刑侦破案件罪犯身份识别、身份证及驾驶证等证件验证、银行及海关的监控、自动门卫系统、视频会议、机器人的智能化研究以及医学等方面。

SimpleCV 有一个很强大的函数—findHaarFeatures，这是一个在图像中搜索匹配某一种特定模式（或称 cascade）的算法，在 SimpleCV 中自带几种模式，包括脸、鼻子、眼睛、嘴和身体，同时，也可以下载或生成自己模式文件。FindHaarFeatures 可以分析图像并从中匹配出对应的模式，然后返回匹配到的部分在图像中的位置。这意味着，可以从图像或摄像头捕获的图像中匹配汽车、动物或人。

实验内容

[1]准备相关设备。

[2]USB 摄像头与 Raspberry Pi 连接。

[3]打开终端（Terminal）。

[4]测试摄像头。

[5]安装并测试 SimpleCV。

[6]编写代码。

[7]获取检测结果。

实验步骤

[1]准备相关设备。

准备实验所用的 USB 摄像头、mini 键盘等设备。

[2]USB 摄像头与 Raspberry Pi 连接。

将 USB 摄像头与 Raspberry Pi 主板上的 USB 相连接。

[3]打开终端（Terminal）。

在 Raspberry Pi 桌面双击打开 Terminal 图标，如图 4-4-1 所示进入命令操作窗口。

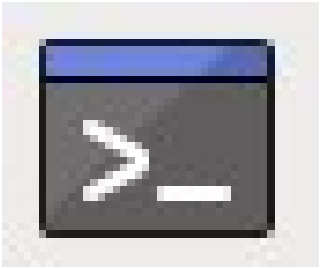


图 4-4-1 Terminal

[4]测试摄像头。

市面上有很多型号的摄像头，没办法保证每一种都能直接在 Raspberry Pi 上正常工作，现需要对摄像是否能正常使用进行测试。

（一） 安装 luvcview 软件（已安装好，可以跳过此步骤）

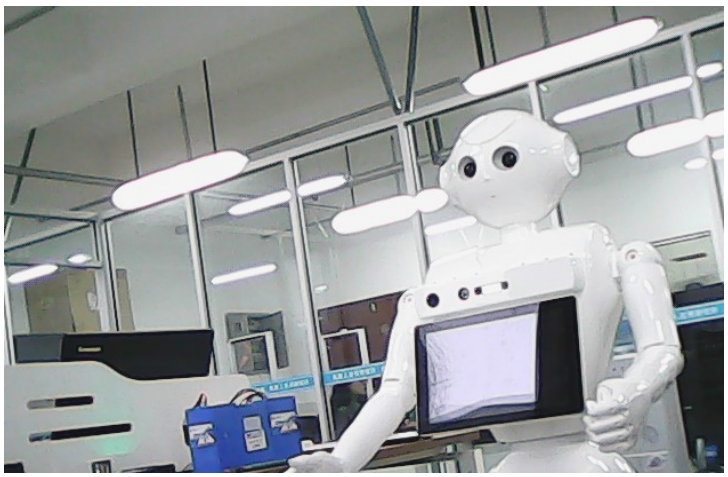
在终端输入以下命令：

```
pi@raspberrypi:~ $ sudo apt-get install luvcview
```

（二） 调用 luvcview 软件

程序安装完成之后，在图形化桌面环境的终端中输入 luvcview 运行这个程序。Luvcview 会打开新的窗口显示摄像头采集到的图像，如图 4-4-2 所示。画图的尺寸会在终端上显示出来。如果显示的视频图像上有一些波纹，可以试着缩小画图的尺寸。例如，默认的视频大小是 640x480，可以进行一下设置：

```
pi@raspberrypi:~ $ luvcview -s 320x480
```



[5]安装并测试 SimpleCV。

注：已安装好，可跳过此步骤。

这里将在 Python 中使用 SimpleCV 库来操作摄像头，SimpleCV 是与电脑视觉相关的开源库。通过使用 SimpleCV，可以很方便地从摄像头采集图像并显示在屏幕上或保存为文件。但 SimpleCV 最强大功能在于它包含的计算机视觉相关的算法，可以用来实现一些让人惊叹的效果。除了基本的图像变换，SimpleCV 还可以用来跟踪、检测和识别图像或视频中的特定对象。

在安装 SimpleCV 前，需要安装一些它所依赖的库，安装命令如下所示：

```
pi@raspberrypi:~ $ sudo apt-get install ipython python-opencv python-scipy pytho
n-numpy python-pygame python-setuptools
```

安装 SimpleCV：

```
pi@raspberrypi:~ $ sudo pip install https://github.com/sightmachine/SimpleCV/zip
ball/develop
```

[6]编写代码。

（一） 新建文件夹 simplecv-test。

（二） 在 simplecv-test 文件夹下新建文件 face-detector.py，双击新建的 face-detector.py 文件

（三） 输入如下图 4-4-3 所示代码。

face-detector.py - /home/pi/simplecv-test/face-detector.py (2.7.9)

File Edit Format Run Options Windows Help

```
from SimpleCV import *
import time
display = Display()
my = Camera()
while display.isNotDone():
    Img = my.getImage()
    faces = Img.findHaarFeatures('face')
    if faces:
        for face in faces:
            print "Face at"+ str(face.coordinates())
            face.draw(Color.RED,1)
    Img.show()
```

图 4-4-3 代码

（四） 保存 face-detector.py 文件。

[7]获取检测结果。

在终端运行，获得检测结果，如下图 4-4-4 所示。

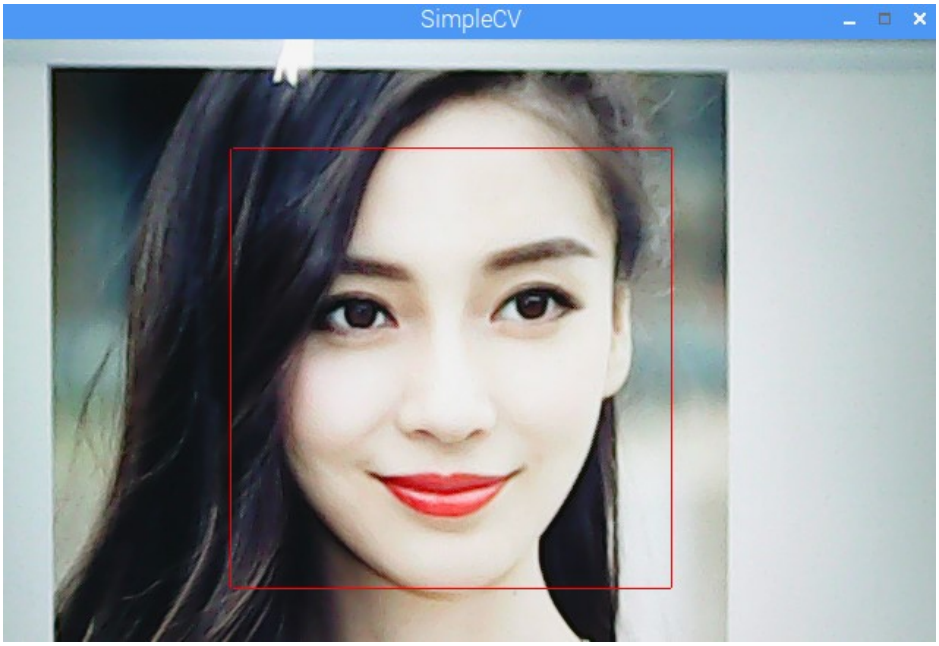


图 4-4-4 检测结果显示

实验小结

通过对本节的学习，了解 SimpleCV 的基本函数，同时学习了树莓派进行人脸检测的方法。

思考题

1、尝试多人脸的图像进行检测。

2、怎么利用 SimpleCV 显示出人脸的坐标位置。