## 目录

[前言 1](#_bookmark0)

[概述 2](#_bookmark1)

[用途 3](#_bookmark1)

[板上资源及其优势 3](#_bookmark2)

[常规开发板与高频开发板的差异对比表 6](#_TOC_250001)

[准备工作 6](#_bookmark3)

[例程简介 6](#_bookmark4)

[实验 1 流水灯 6](#_bookmark5)

[实验 2 中断检测按键 9](#_bookmark6)

[实验 3 12864 液晶显示 1](#_bookmark7)3

[实验 4 异步串行口（USART）的简单应用 20](#_TOC_250000)

[实验 5 PWM 输出. 2](#_bookmark8)4

[实验 6 ADC 采集数据（采用 DMA 模式） 2](#_bookmark9)8

[实验 7 利用MF58 测出环境温度并显示在 12864 液晶上 32](#_bookmark9)

**负责声明**

### 为了安全有效的使用本产品，请您使用前仔细阅读以下信息；

### 本产品是通过自行外接+24VDC 电源供电，自行外接电源请注意电源极性；

### 请不要在带电时插拔开发板上相关器件；

### 自行扩展搭接导致不良故障，本公司不负任何责任；

### 产品不定时升级，所有更改不另行通知客户，本公司有最终解释权。

# 概述

### DYSTM32F4 高频超声波开发板基于STM32F407VGT6 的微控器，包括一个板载 JLINK 接口，2 路 LED，3 路独立按键，LCD12864 显示接口，原创超声波发射电路，原创超声波接收电路，UART 异步串行口+MAX232 电平转换芯片， 温度传感器MF58 接口。如下图所示。

### STM32F407VGT6 微控器是 ARM Cortex-M4 内核，32 位 MCU，带有 FPU 单元，有 210 DMIPS，多达 1MB FLASH/192+4KB RAM，17 个定时器，3 个ADC， 15 个通讯接口和一个摄像口。

# 用途

### DYSTM32F4 高频超声波开发板，可用做超声波水下测深，探鱼的前期开发； 可以做短距离，高速率的水声通信实验；可做超声测距产品的前期开发，超声传感器的相关实验，可应用于汽车倒车、建筑施工工地以及一些工业现场的位置监控前期开发，也可用于如液位、井深、管道长度的测量等场合的前期开发应用。

# 板上资源及其优势

### STM32F407VGT6 微控制器有 1M 的FLASH 存储器，192K 的RAM，LQFP100

### 封装

### 板上的 JLINK 接口，方便仿真调试

### 板子电源：24VDC

### 1 个电源指示灯LED1（红）

### 2 个用户LED 指示灯 LED2(黄)，LED3（绿），可模拟指示工作状态；

### 3 个按钮（用户按键），可以方便的进行人机互动实验；

### 温感MF58 电路接口，实现实时温度采集显示功能

### 板载LCD12864 显示接口，可以方便显示汉字和图形数据；

### 原创超声波发射电路，实现多级放大，同时含有带通滤波处理；

### 原创超声波接收电路，同时采集预留 2 种采集方式，使用更加灵活；

### UART 异步串行口+MAX232 电平转换芯片，通过 DB9（母头）接口可与电脑的COM 口通讯；

### 温度传感器-- MF58，可以学习到AD 的方法和温度数据处理；

### LQFP100 IO 口的扩展头，可以快速连接成原型板，易于探测；

### 定时器输出频率可高达 84M；

### ADC 可配置分辨率为 12 位、10 位、8 位、6 位；

### ADC 的最大采样频率是 2MSPS;

### 使用MDK 开发环境；

### 适用的换能器频率范围：200K—5M ；

### 众多功能、人性化设计。除普通的实验功能外，还能更大的发挥您的潜能，合理的布局让您在操作的时候更得心应手;

### 硬件设计模块划分清晰，原理明确，符合正常使用习惯;

### 电路简单，安排了最常用最实用的功能电路，接口尽量不复用;

### 无论是程序下载还是仿真调试，让您在学习中事半功倍，从而更快的学习和掌握超声波的应用;

### 标准例程的基础上加配了综合程序，购买者可全方位体验代码的学习，更重要的是能了解产品开发的全过程实例。

# 常规开发板与高频开发板的差异对比表

|  |  |  |
| --- | --- | --- |
| 项目 | DYSTM32 高频开发板 | DY-KFB 常规开发板 |
| CPU | STM32F407VG | ATMEG1284p |
| CPU 内核 | Cortex-M4 | 无 |
| Flash | 1M | 128k |
| SRAM | 192K | 16K |
| EEPROM | 无 | 4k |
| ADC | 12 位、10 位、8 位、6 位 | 10 位 |
| 定时器 | 10 个通用定时器；2 个高级  定时器；2 个基本定时器 | 2 个 8 位定时器；2 个 16 位定时  器 |
| 数据总线宽度 | 32 位 | 8 位 |
| 最高系统时钟  频率 | 168M | 20M |
| 仿真器 | J-LINK | AVR JTAGICE |
| 换能器 | 200K 以上 | 200K 以下（含 200K） |

# 准备工作

### 当用户拿到实验板后的第一件事是先看实验板供电是否正常、下载程序功能是否正常、各芯片功能是否都能成功实现，检测方法如下所示：

### 先把产品包装中的四个铜柱及镙母安装在实验板上。

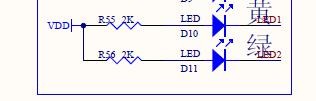
### 把相应的测试程序烧写之STM32F4 中观察个模块测试是否正常。

# 例程简介

**实验 1 流水灯**

### 1、实验描述：运用 ST 官方库来操作I/O 口，使 I/O 口产生置位（0）和复位（0） 信号，从而控制LED 的亮灭。

### 2、局部电路：



3、 用到的库文件

startup\_stm32f4xx.s system\_stm32f4xx.c stm32f4xx\_gpio.h stm32f4xx\_rcc.h

4、用户编写的文件main.c 和 main.h stm32f4xx\_it.c led.c 和 led.h

5、文件部分源码

main.c 文件代码#include "stm32f4xx.h"

#include "main.h"

/\* Private functions \*/

void Delay(u32 nCount) //简单的延时函数

{

for(;nCount != 0; nCount--);

}

int main(void)

{

/\*配置系统时钟 RCC\*/ SystemInit();

/\*LED 端口初始化\*/ LED\_GPIO\_Config();

LED1(OFF); LED2(OFF);

while(1)

{

LED1(ON); //亮

Delay(0xFFFFEF); LED1(OFF); //灭

LED2(ON);

Delay(0xFFFFEF); LED2(OFF);

}

}

led.c 文件代码#include "led.h"

/\*

* 函数名：LED\_GPIO\_Config
* 描述 ：配置 LED 用到的 I/O 口
* 输入 ：无
* 输出 ：无

\*/

void LED\_GPIO\_Config(void) //指示灯初始化

{

/\*定义一个 GPIO\_InitTypeDef 类型的结构体\*/ GPIO\_InitTypeDef GPIO\_InitStructure;

/\* 开 启 GPIOB 的 外 设 时 钟 \*/ RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOB, ENABLE);

/\* 选 择 要 控 制 的 GPIOB 引 脚 \*/ GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_8 | GPIO\_Pin\_9;

/\* 设 置 引 脚 模 式 为 通 用 输 出 \*/ GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_OUT;

/\* 设 置 引 脚 模 式 为 推 挽 输 出 \*/ GPIO\_InitStructure.GPIO\_OType = GPIO\_OType\_PP;

/\* 设 置 引 脚 速 率 为 100MHz \*/ GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_100MHz;

/\* 设 置 引 脚 没 有 上 拉 和 下 拉 \*/ GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_NOPULL;

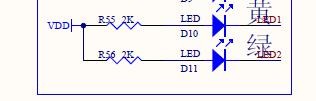
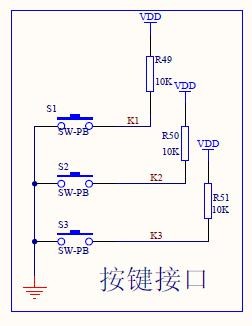
/\*调用库函数，初始化 GPIOB\*/ GPIO\_Init(GPIOB, &GPIO\_InitStructure);

}

**实验 2 中断检测按键**

### 1、实验描述：PD0 连接到K3,PD1 连接到K2,PD2 连接到K1,PD0、PD1、PD2 配置为线中断模式，当 K3、K2、K1 按下时，进入线中断处理函数，LED 状态取反。

2、局部电路：



3、 用到的库文件

startup\_stm32f4xx.s system\_stm32f4xx.c stm32f4xx\_gpio.h stm32f4xx\_rcc.h stm32f4xx\_exti.h stm32f4xx\_syscfg.h misc.h

4、用户编写的文件

main.c 和 main.h stm32f4xx\_it.c led.c 和 led.h key.c 和 key.h delay.c 和 delay.h

5、 文件部分源码

main.c 文件代码#include "stm32f4xx.h"

#include "main.h"

/\*

* 函数名：main
* 描述 ：主函数
* 输入 ：无
* 输出 ：无

\*/

int main(void)

{

/\*配置系统时钟 RCC\*/ SystemInit();

/\*LED 端口初始化\*/ LED\_GPIO\_Config();

/\*KEY 端口初始化\*/ Key\_GPIO\_Config();

EXTI\_KEY\_Config(KEY1);//按键 1 中断模式EXTI\_KEY\_Config(KEY2); EXTI\_KEY\_Config(KEY3);

LED1(ON);

LED2(ON);

while(1);

}

key.c 文件代码#include "key.h"

/\*

* 函数名：Key\_GPIO\_Config
* 描述 ：配置 KEY 用到的 I/O 口
* 输入 ：无
* 输出 ：无

\*/

void Key\_GPIO\_Config(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOD, ENABLE); GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_0 | GPIO\_Pin\_1 | GPIO\_Pin\_3; GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IN; GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_NOPULL; GPIO\_Init(GPIOD, &GPIO\_InitStructure);

}

/\*

* 函数名：EXTI\_KEY\_Config
* 描述 ：配置 KEY 用到的 I/O 口为中断口
* 输入 ：无
* 输出 ：无

\*/

void EXTI\_KEY\_Config(Key\_Def key)//将按键设置成中断方式

{

EXTI\_InitTypeDef EXTI\_InitStructure; NVIC\_InitTypeDef NVIC\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_SYSCFG, ENABLE); SYSCFG\_EXTILineConfig(EXTI\_PortSourceGPIOD, KEY\_PNS[key]);

EXTI\_InitStructure.EXTI\_Line = KEY\_SRC[key]; EXTI\_InitStructure.EXTI\_Mode = EXTI\_Mode\_Interrupt; if(key==KEY3)

EXTI\_InitStructure.EXTI\_Trigger = EXTI\_Trigger\_Rising; else

EXTI\_InitStructure.EXTI\_Trigger = EXTI\_Trigger\_Falling; EXTI\_InitStructure.EXTI\_LineCmd = ENABLE; EXTI\_Init(&EXTI\_InitStructure);

NVIC\_PriorityGroupConfig(NVIC\_PriorityGroup\_0); //嵌套分组为组 0

NVIC\_InitStructure.NVIC\_IRQChannel = KEY\_IRQ[key]; NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority = 0x01; NVIC\_InitStructure.NVIC\_IRQChannelSubPriority = 0x01; NVIC\_InitStructure.NVIC\_IRQChannelCmd = ENABLE; NVIC\_Init(&NVIC\_InitStructure);

}

stm32f4xx\_it.c 部分文件代码void EXTI0\_IRQHandler(void)

{

Delay(0x0FFFF); if(EXTI\_GetITStatus(EXTI\_Line0) != RESET)

{

LED1(OFF); //灭

EXTI\_ClearITPendingBit(EXTI\_Line0);

}

}

void EXTI1\_IRQHandler(void)

{

Delay(0x0FFFF); if(EXTI\_GetITStatus(EXTI\_Line1) != RESET)

{

LED2(OFF);

EXTI\_ClearITPendingBit(EXTI\_Line1);

}

}

void EXTI3\_IRQHandler(void)

{

Delay(0x0FFFF); if(EXTI\_GetITStatus(EXTI\_Line3) != RESET)

{

LED1(ON);

LED2(ON);

EXTI\_ClearITPendingBit(EXTI\_Line3);

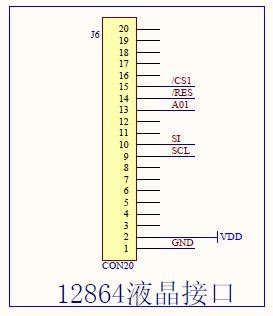
}

}

**实验 3 12864 液晶显示**

### 1、实验描述：初步掌握STM32 控制基本液晶显示信息。

### 2、局部电路：



3、 用到的库文件

startup\_stm32f4xx.s system\_stm32f4xx.c stm32f4xx\_gpio.h stm32f4xx\_rcc.h stm32f4xx\_syscfg.h misc.h

4、用户编写的文件

main.c 和 main.h stm32f4xx\_it.c delay.c 和 delay.h

lcd12864.c 和 lcd12864.h

5、 文件部分源码

main.c 部分文件代码

#include "stm32f4xx.h" #include "main.h"

/\*

* 函数名：main
* 描述 ：主函数
* 输入 ：无
* 输出 ：无

\*/

int main(void)

{

/\* 配置系统时钟 RCC \*/ SystemInit();

/\* LCD 端口初始化\*/ LCD\_12864\_Config();

/\* LCD 复位\*/ ResetLcd();

/\* 清 屏 \*/ ClrScreen();

printFront(4,0,0);//距printFront(4,16,1);//离printFront(4,32,2);//: printNumbers(4,48,0);//0 printNumbers(4,56,0);//0 printNumbers(4,64,10);//. printNumbers(4,72,0);//0 printNumbers(4,80,0);//0 printFront(4,96,3);//厘printFront(4,112,4);//米

while(1);

}

lcd12864.c 部分文件代码

#include "lcd12864.h" #include "delay.h"

void LCD\_12864\_Config(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;//定义一个 GPIO\_InitStructure 类型的结构体

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOD, ENABLE); //开启 GPIOD 的外设时钟

// GPIO\_Pin\_8 控制 LCD RES ,GPIO\_Pin\_9 控制 LCD A0, GPIO\_Pin\_10 控制 LCD Data6 ,GPIO\_Pin\_11

控制 LCD Data7

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_3 |GPIO\_Pin\_4 | GPIO\_Pin\_5 | GPIO\_Pin\_6 | GPIO\_Pin\_7; GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_OUT;

GPIO\_InitStructure.GPIO\_OType = GPIO\_OType\_PP; GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz; GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_NOPULL; GPIO\_Init(GPIOD, &GPIO\_InitStructure);

GPIO\_ResetBits(GPIOD, GPIO\_Pin\_3 |GPIO\_Pin\_4 | GPIO\_Pin\_5 | GPIO\_Pin\_6 | GPIO\_Pin\_7); //输出低电

平

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* 功能描述：初始化 LCD
* 隶属模块：公开函数模块
* 函数属性：
* 参数说明：
* 返回说明：
* 注：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void ResetLcd(void)

{

ClrRES;

delay\_us(200);

SetRES; //2010.6.29 增加以上三句，解决一体液晶屏经常花屏的问题

delay\_us(200);

Write\_COM(Disp\_Off); //Display OFF Write\_COM(SetBias9); // 设置 BIAS 为 1/9 Write\_COM(ADC\_Normal); //从左至右输入Write\_COM(OUTNormal); //com1 --> com64

Write\_COM(V5Ratio); // Select internal resistor ratio(Rb/Ra)mode

Write\_COM(V5OutVol); // Set the v5 output voltage Write\_COM(SetBright); //调节液晶屏的亮度

Write\_COM(POWERSet); //voltage follower ON regulator ON booster ON Write\_COM(DispNormal); // 从左至右显示

Write\_COM(AllNormal); //Entire Display Disable

Write\_COM(Disp\_On); //Display ON ClrScreen();

Write\_COM(SetStartLine); //Set Display Start Line = com0 Write\_COM(SetPage); //Set Page Address = 0 Write\_COM(SetColMsb); //Set Column Address = 0,set 4 higher bits

Write\_COM(SetColLsb); //set 4 lower bits of column of display data RAM in register

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* 功能描述：传递数据到 LCD (串行)
* 隶属模块：公开函数模块
* 函数属性：
* 参数说明：
* 返回说明：
* 注：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void Write\_SPI(uint8\_t data)

{

uint8\_t Count;

for (Count=0; Count<8; Count++)

{

if (data & 0x80)

{

}

else

{

}

Set\_SI;

Clr\_SI;

data <<= 1; Set\_CLK; delay\_us(1); Clr\_CLK;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* 功能描述：写命令到 LCD (串行)
* 隶属模块：公开函数模块
* 函数属性：
* 参数说明：
* 返回说明：
* 注：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void Write\_COM(uint8\_t cmdcode)

{

Clr\_CLK; delay\_us(1); ClrA0; delay\_us(1); ClrCS1;

Write\_SPI(cmdcode) ; SetCS1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* 功能描述：写数据到 LCD (串行)
* 隶属模块：公开函数模块
* 函数属性：
* 参数说明：
* 返回说明：
* 注：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void WriteData(uint8\_t dispdata)

{

Clr\_CLK; delay\_us(1); SetA0; delay\_us(1); ClrCS1;

Write\_SPI(dispdata) ; SetCS1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* 功能描述：清除 LCD 全屏
* 隶属模块：公开函数模块
* 函数属性：
* 参数说明：
* 返回说明：
* 注：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void ClrScreen(void)

{

uint8\_t i, j;

Write\_COM(SetStartLine); //Set Display Start Line = com0

for (i=0; i<8; i++)

{

Write\_COM(SetPage+i); //Set Page Address

for (j=0; j<128; j++)

{

Write\_COM(SetColLsb|(j&0x0f)); Write\_COM(SetColMsb|((j&0xf0)>>4)); WriteData(0x00);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* 功能描述：在任意位置显示任意大小点阵
* 隶属模块：公开函数模块
* 函数属性：
* 参数说明： startCol起始最左列

startL 起始最上端行 \* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void printNumbers(uint8\_t startL,uint8\_t startCol,int16\_t code)//输入阿拉伯数字 8\*16 字模

{

uint8\_t col;

Write\_COM(0x10+(startCol>>4)); Write\_COM(startCol&0x0f); Write\_COM(0xB0+startL); for(col = 0; col < 8; col++)

{

WriteData(LcdFont[code][col]);

}

Write\_COM(0x10+(startCol>>4)); Write\_COM(startCol&0x0f); Write\_COM(0xB0+startL-1); for(col = 0; col < 8; col++)

{

WriteData(LcdFont[code][col+8]);

}

}

void printFront(uint8\_t startL,uint8\_t startCol,int16\_t uChar) //输入字 16\*16 字模

{

uint8\_t col;

Write\_COM(0x10+(startCol>>4)); Write\_COM(startCol&0x0f); Write\_COM(0xB0+startL); for(col = 0; col < 16; col++)

{

WriteData(Font[uChar][col]);

}

Write\_COM(0x10+(startCol>>4)); Write\_COM(startCol&0x0f); Write\_COM(0xB0+startL-1); for(col = 0; col < 16; col++)

{

WriteData(Font[uChar][col+16]);

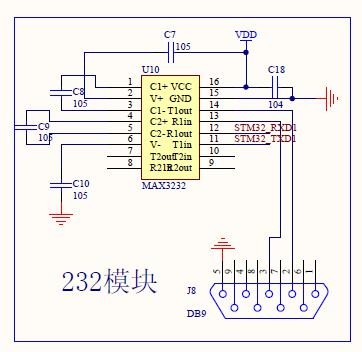
}

}

**实验 4 异步串行口（USART）的简单应用**

### 1、实验描述：初步掌握STM32 的USART 的应用。

### 2、局部电路图：



3、 用到的库文件

startup\_stm32f4xx.s system\_stm32f4xx.c stm32f4xx\_gpio.h stm32f4xx\_rcc.h stm32f4xx\_usart.h

misc.h

4、用户编写的文件main.c 和 main.h stm32f4xx\_it.c delay.c 和 delay.h usart.c 和 usart.h led.c 和 led.h

5、 文件部分源码

main.c 部分文件代码#include "stm32f4xx.h"

#include "main.h" uint16\_t temp;

int main(void)

{

/\*配置系统时钟 RCC\*/ SystemInit();

/\*LED 端口初始化\*/ LED\_GPIO\_Config();

/\*USART 端口初始化\*/ USART\_Config(); NVIC\_USART\_Config();

LED1(OFF); LED2(OFF);

temp=0;

while(1)

{

USART1\_Puts("Hello Wrold!\n"); LED1(ON);

Delay(0xFFFFFF); LED1(OFF);

//while (USART\_GetFlagStatus(USART1,USART\_FLAG\_RXNE) == RESET); //

串口是否接受到数据

Delay(0xFFFFFF); LED2(OFF);

USART1\_Puts("Get Data From USART1:"); temp = USART\_ReceiveData(USART1); USART\_SendData(USART1,temp);

while (USART\_GetFlagStatus(USART1, USART\_FLAG\_TXE) == RESET); //串口发

送是否完成

Delay(0xFFFFFF); USART1\_Puts("\r\n");

}

}

usart.c 部分文件代码#include "usart.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: void USART\_Config(void)

Description: USART 配置函数

Input: 无Output:无Return:无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void USART\_Config(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure; USART\_InitTypeDef USART\_InitStructure; USART\_ClockInitTypeDef USART\_ClockInitStruct;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_USART1, ENABLE); //开启 USART1 时钟RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOA, ENABLE); //开启 GPIOA 时钟GPIO\_PinAFConfig(GPIOA, GPIO\_PinSource9, GPIO\_AF\_USART1);//这相当于 M3 的开启复

用时钟？只配置复用的引脚，

GPIO\_PinAFConfig(GPIOA, GPIO\_PinSource10, GPIO\_AF\_USART1);//

GPIO\_StructInit(&GPIO\_InitStructure); //缺省值填入

/\* 配 置 GPIOA\_Pin9 为 TX 输 出 \*/ GPIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_9;

GPIO\_InitStructure.GPIO\_Mode=GPIO\_Mode\_AF; //设置为复用，必须为 AF，OUT 不行

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz; GPIO\_InitStructure.GPIO\_PuPd= GPIO\_PuPd\_NOPULL; //没有上拉和下拉GPIO\_InitStructure.GPIO\_OType= GPIO\_OType\_PP; //输出推免GPIO\_Init(GPIOA,&GPIO\_InitStructure);

/\* 配 置 GPIOA\_Pin10 为 RX 输 入 \*/ GPIO\_InitStructure.GPIO\_Pin=GPIO\_Pin\_10;

GPIO\_InitStructure.GPIO\_Mode=GPIO\_Mode\_AF; //这也必须为复用，与 M3 不同！

// GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz; GPIO\_Init(GPIOA,&GPIO\_InitStructure);

/\*配置 USART1\*/

USART\_StructInit(&USART\_InitStructure);//这个要屏蔽是不行的 2014.3.20

USART\_InitStructure.USART\_BaudRate =115200; USART\_InitStructure.USART\_WordLength = USART\_WordLength\_8b; USART\_InitStructure.USART\_StopBits = USART\_StopBits\_1; USART\_InitStructure.USART\_Parity = USART\_Parity\_No; USART\_InitStructure.USART\_HardwareFlowControl = USART\_HardwareFlowControl\_None; USART\_InitStructure.USART\_Mode = USART\_Mode\_Rx | USART\_Mode\_Tx; USART\_Init(USART1, &USART\_InitStructure);

/\* Enable USART \*/

USART\_ClockStructInit(&USART\_ClockInitStruct); //填充外设时钟默认值USART\_ClockInit(USART1, &USART\_ClockInitStruct); //初始化外设时钟USART\_ITConfig(USART1, USART\_IT\_RXNE, ENABLE); //使能 USART1 中断USART\_Cmd(USART1, ENABLE); //使能 USART1

}

void NVIC\_USART\_Config()

{

NVIC\_InitTypeDef NVIC\_InitStructure; NVIC\_PriorityGroupConfig(NVIC\_PriorityGroup\_1); // 嵌 套 优 先 级 分 组 为 1 NVIC\_InitStructure.NVIC\_IRQChannel = USART1\_IRQn; //嵌套通道为 USART1\_IRQn NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority = 0; //抢占优先级为 0

NVIC\_InitStructure.NVIC\_IRQChannelSubPriority = 0; //响应优先级为 0 NVIC\_InitStructure.NVIC\_IRQChannelCmd = ENABLE; //通道中断使能NVIC\_Init(&NVIC\_InitStructure);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function: void USART1\_Puts(char \* str)

Description: USART1 发送数据Input: 待发送数据指针Output:无

Return:无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ void USART1\_Puts(char \* str)

{

while (\*str)

{

USART\_SendData(USART1, \*str++);

/\* Loop until the end of transmission \*/

while (USART\_GetFlagStatus(USART1, USART\_FLAG\_TXE) == RESET); //当 TXE 被置起时，一帧数据传输完成

}

}

stm32f4xx\_it.c 部分文件代码

void USART1\_IRQHandler(void)

{

if (USART\_GetITStatus(USART1, USART\_IT\_RXNE) != RESET) //判断为接收中断

{

USART\_ClearITPendingBit(USART1,USART\_IT\_RXNE);//清除中断挂起位USART\_SendData(USART1, USART\_ReceiveData(USART1)); //发送收到的数据while (USART\_GetFlagStatus(USART1, USART\_FLAG\_TXE) == RESET); LED2(ON); //点亮 LED，起到中断指示作用

}

}

**实验 5 PWM 输出**

### 1、实验目的：利用通用定时器 TIM2 产生占空比可调，频率可调的 PWM 信号。

### 2、局部电路：见原理图 PDF

3、 用到的库文件

startup\_stm32f4xx.s system\_stm32f4xx.c stm32f4xx\_gpio.h stm32f4xx\_rcc.h stm32f4xx\_tim.h misc.h

4、用户编写的文件

main.c 和 main.h stm32f4xx\_it.c ssp.c 和 ssp.h

5、 文件部分源码

main.c 部分文件代码#include "stm32f4xx.h"

#include "main.h"

/\* Private functions \*/

int main(void)

{

uint32\_t i=0;

//配置系统时钟 RCC SystemInit(); NVIC\_Config(); GPIO\_Config();

SSP\_Configuration();// 配置 TIM2 的脉冲输出为 10Hz

while(1)

{

if(TIM2\_Pulse\_TIM3\_Counter\_OK == 0) Output\_Pulse(PulseNum);

else if(TIM2\_Pulse\_TIM3\_Counter\_OK == 1)

{

for(i=0; i<10000000; i++); // Delay TIM2\_Pulse\_TIM3\_Counter\_OK = 0;

}

}

}

ssp.c 部分文件代码#include "ssp.h"

uint8\_t TIM2\_Pulse\_TIM3\_Counter\_OK = 0;

/\*

* 函数名：TIM1\_GPIO\_Config
* 描述 ：配置 TIM1 复用输出 PWM 时用到的 I/O
* 输入 ：无
* 输出 ：无

\*/

void GPIO\_Config(void)

{

/\*定义了一个 GPIO\_InitStructure 的结构体，方便一下使用 \*/ GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOA , ENABLE);

//RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM2 | RCC\_APB1Periph\_TIM3, ENABLE);

/\* Timer2 Channel 1, PA0 \*/ GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AF; GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_100MHz; GPIO\_InitStructure.GPIO\_OType = GPIO\_OType\_PP; GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_NOPULL; GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_1; GPIO\_Init(GPIOA, &GPIO\_InitStructure);

GPIO\_PinAFConfig(GPIOA, GPIO\_PinSource1, GPIO\_AF\_TIM2);

}

void SSP\_Configuration(void)

{

uint16\_t nPDTemp ;

TIM\_TimeBaseInitTypeDef TIM\_TimeBaseStructure; TIM\_OCInitTypeDef TIM\_OCInitStructure;

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM2 | RCC\_APB1Periph\_TIM3, ENABLE);

TIM\_Cmd(TIM2, DISABLE);

nPDTemp = 84000000/PulseFrequency;

// 时基配置：配置 PWM 输出定时器——TIM2

// Time base configuration TIM\_TimeBaseStructure.TIM\_Period = nPDTemp - 1; TIM\_TimeBaseStructure.TIM\_Prescaler = 0;

TIM\_TimeBaseStructure.TIM\_ClockDivision = 0; TIM\_TimeBaseStructure.TIM\_CounterMode = TIM\_CounterMode\_Up; TIM\_TimeBaseStructure.TIM\_RepetitionCounter = 0; TIM\_TimeBaseInit(TIM2, &TIM\_TimeBaseStructure);

// 输出配置：配置 PWM 输出定时器——TIM2

// PWM1 Mode configuration: Channel1 TIM\_OCInitStructure.TIM\_Pulse = nPDTemp >> 1;//50% TIM\_OCInitStructure.TIM\_OCMode = TIM\_OCMode\_PWM2; TIM\_OCInitStructure.TIM\_OCPolarity = TIM\_OCPolarity\_Low; TIM\_OCInitStructure.TIM\_OutputState = TIM\_OutputState\_Enable; TIM\_OC2Init(TIM2, &TIM\_OCInitStructure); TIM\_OC2PreloadConfig(TIM2, TIM\_OCPreload\_Enable); TIM\_ARRPreloadConfig(TIM2, ENABLE);

// 时基配置：配置脉冲计数寄存器——TIM3 TIM\_TimeBaseStructure.TIM\_Period = 0xFFFF; TIM\_TimeBaseStructure.TIM\_Prescaler = 0;

TIM\_TimeBaseStructure.TIM\_ClockDivision = 0; TIM\_TimeBaseStructure.TIM\_CounterMode = TIM\_CounterMode\_Up; TIM\_TimeBaseStructure.TIM\_RepetitionCounter = 0; TIM\_TimeBaseInit(TIM3, &TIM\_TimeBaseStructure);

// Output Compare Active Mode configuration: Channel1 TIM\_OCInitStructure.TIM\_OCMode = TIM\_OCMode\_Inactive; TIM\_OCInitStructure.TIM\_OCPolarity = TIM\_OCPolarity\_High; TIM\_OCInitStructure.TIM\_OutputState = TIM\_OutputState\_Enable; TIM\_OCInitStructure.TIM\_Pulse = 0xFFFF; // 这里的配置值意义不大TIM\_OC1Init(TIM3, &TIM\_OCInitStructure);

// 配置 TIM2 为主定时器

// Select the Master Slave Mode

TIM\_SelectMasterSlaveMode(TIM2, TIM\_MasterSlaveMode\_Enable);

// Master Mode selection

TIM\_SelectOutputTrigger(TIM2, TIM\_TRGOSource\_Update);

// 配置 TIM3 为从定时器

// Slave Mode selection: TIM3 TIM\_SelectSlaveMode(TIM3, TIM\_SlaveMode\_Gated); TIM\_SelectInputTrigger(TIM3, TIM\_TS\_ITR1); TIM\_ITRxExternalClockConfig(TIM3, TIM\_TS\_ITR1);

//TIM\_ITConfig(TIM3, TIM\_IT\_CC1, ENABLE);

TIM\_Cmd(TIM2, DISABLE); TIM\_Cmd(TIM3, DISABLE);

TIM\_CtrlPWMOutputs(TIM2, ENABLE);//使能 TIM2 的 PWM 输出，

}

void Output\_Pulse(u16 Num)

{

TIM3->CCR1 = Num; TIM3->CNT = 0;

TIM\_Cmd(TIM3, ENABLE); TIM\_ITConfig(TIM3, TIM\_IT\_CC1, ENABLE); TIM\_Cmd(TIM2, ENABLE);

TIM2\_Pulse\_TIM3\_Counter\_OK = 1;

}

//==============================

//

// 配置中断向量表及优先级

//

//==============================

void NVIC\_Config(void)

{

NVIC\_InitTypeDef NVIC\_InitStructure;

#ifdef VECT\_TAB\_RAM NVIC\_SetVectorTable(NVIC\_VectTab\_RAM, 0x0);

#else

NVIC\_SetVectorTable(NVIC\_VectTab\_FLASH, 0x0); #endif

NVIC\_InitStructure.NVIC\_IRQChannel = TIM3\_IRQn; NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority = 0;

NVIC\_InitStructure.NVIC\_IRQChannelSubPriority = 0; NVIC\_InitStructure.NVIC\_IRQChannelCmd = ENABLE; NVIC\_Init(&NVIC\_InitStructure);

}

**实验 6 ADC 采集数据（采用 DMA 模式）**

### 1、实验描述：使用 ADC 时采用DMA 传输方式，又 DMA 把ADC 外设转换的数据传输到SRAM,再进行处理。

### 2、局部电路：见原理图 PDF

3、 用到的库文件

startup\_stm32f4xx.s system\_stm32f4xx.c stm32f4xx\_gpio.h stm32f4xx\_rcc.h stm32f4xx\_tim.h stm32f4xx\_adc.h stm32f4xx\_dma.h misc.h

4、用户编写的文件main.c 和 main.h stm32f4xx\_it.c ssp.c 和 ssp.h adc.c 和 adc.h

5、 文件部分源码

main.c 部分文件代码#define LONGSAMPNUM 100

uint16\_t DATASMOOTH[LONGSAMPNUM]={0};

/\* Private functions \*/

int main(void)

{

uint32\_t i=0;

//配置系统时钟 RCC SystemInit(); NVIC\_Config(); GPIO\_Config();

SSP\_Configuration();// 配置 TIM2 的脉冲输出

ADC3\_GPIO\_Config(); ADC3\_Mode\_Config(); ADC\_SoftwareStartConv(ADC3);

while(1)

{

if(TIM2\_Pulse\_TIM3\_Counter\_OK == 0) Output\_Pulse(PulseNum);

else if(TIM2\_Pulse\_TIM3\_Counter\_OK == 1)

{

for(i=0; i<10000000; i++); // Delay TIM2\_Pulse\_TIM3\_Counter\_OK = 0;

}

ProceAD(DATASMOOTH,LONGSAMPNUM);

}

}

adc.c 部分文件代码#include "adc.h"

IO uint16\_t ADC3ConvertedValue = 0;

/\*

* 函数名：ADC1\_GPIO\_Config
* 描述 ：使能 ADC1 和 DMA1 的时钟，初始化 PC.01
* 输入 : 无
* 输出 ：无
* 调用 ：内部调用

\*/

void ADC3\_GPIO\_Config(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure; RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOA, ENABLE);

/\* Configure PC.01 as analog input \*/ GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_2; GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AN; GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_NOPULL;

GPIO\_Init(GPIOA, &GPIO\_InitStructure); // PC1,输入时不用设置速率

}

/\* 函数名：ADC1\_Mode\_Config

* 描述 ：配置 ADC1 的工作模式为 MDA 模式
* 输入 : 无
* 输出 ：无
* 调用 ：内部调用

\*/

void ADC3\_Mode\_Config(void)

{ //AD 初始化

ADC\_InitTypeDef ADC\_InitStructure; ADC\_CommonInitTypeDef ADC\_CommonInitStructure; DMA\_InitTypeDef DMA\_InitStructure;

/\* Enable ADC3, DMA2 and GPIO clocks \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_DMA2 , ENABLE); RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC3, ENABLE);

/\* DMA2 Stream0 channel0 configuration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ DMA\_InitStructure.DMA\_Channel = DMA\_Channel\_2; DMA\_InitStructure.DMA\_PeripheralBaseAddr = (uint32\_t)ADC3\_DR\_ADDRESS; DMA\_InitStructure.DMA\_Memory0BaseAddr = (uint32\_t)&ADC3ConvertedValue; DMA\_InitStructure.DMA\_DIR = DMA\_DIR\_PeripheralToMemory; DMA\_InitStructure.DMA\_BufferSize = 1;

DMA\_InitStructure.DMA\_PeripheralInc = DMA\_PeripheralInc\_Disable; DMA\_InitStructure.DMA\_MemoryInc = DMA\_MemoryInc\_Disable; DMA\_InitStructure.DMA\_PeripheralDataSize = DMA\_PeripheralDataSize\_HalfWord; DMA\_InitStructure.DMA\_MemoryDataSize = DMA\_MemoryDataSize\_HalfWord; DMA\_InitStructure.DMA\_Mode = DMA\_Mode\_Circular; DMA\_InitStructure.DMA\_Priority = DMA\_Priority\_High; DMA\_InitStructure.DMA\_FIFOMode = DMA\_FIFOMode\_Disable; DMA\_InitStructure.DMA\_FIFOThreshold = DMA\_FIFOThreshold\_HalfFull; DMA\_InitStructure.DMA\_MemoryBurst = DMA\_MemoryBurst\_Single; DMA\_InitStructure.DMA\_PeripheralBurst = DMA\_PeripheralBurst\_Single; DMA\_Init(DMA2\_Stream0, &DMA\_InitStructure);

DMA\_Cmd(DMA2\_Stream0, ENABLE);

/\* ADC Common Init \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ ADC\_CommonInitStructure.ADC\_Mode = ADC\_Mode\_Independent; ADC\_CommonInitStructure.ADC\_Prescaler = ADC\_Prescaler\_Div2; ADC\_CommonInitStructure.ADC\_DMAAccessMode = ADC\_DMAAccessMode\_Disabled; ADC\_CommonInitStructure.ADC\_TwoSamplingDelay = ADC\_TwoSamplingDelay\_5Cycles; ADC\_CommonInit(&ADC\_CommonInitStructure);

/\* ADC3 Init \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ ADC\_StructInit(& ADC\_InitStructure);

ADC\_InitStructure.ADC\_Resolution = ADC\_Resolution\_12b; ADC\_InitStructure.ADC\_ScanConvMode = DISABLE; ADC\_InitStructure.ADC\_ContinuousConvMode = ENABLE; ADC\_InitStructure.ADC\_ExternalTrigConvEdge = ADC\_ExternalTrigConvEdge\_None; ADC\_InitStructure.ADC\_DataAlign = ADC\_DataAlign\_Right; ADC\_InitStructure.ADC\_NbrOfConversion = 1;

ADC\_Init(ADC3, &ADC\_InitStructure);

/\* ADC3 regular channel12 configuration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ ADC\_RegularChannelConfig(ADC3, ADC\_Channel\_2, 1, ADC\_SampleTime\_3Cycles);

/\* Enable DMA request after last transfer (Single-ADC mode) \*/ ADC\_DMARequestAfterLastTransferCmd(ADC3, ENABLE);

/\* Enable ADC3 DMA \*/ ADC\_DMACmd(ADC3, ENABLE);

/\* Enable ADC3 \*/ ADC\_Cmd(ADC3, ENABLE);

}

void ProceAD(uint16\_t \*ADIn,uint32\_t ADLen)

{

while(ADLen>0)

{

\*ADIn = ADC3ConvertedValue; ADIn++; //调整指针ADLen--;

}

}

**实验 7 利用 MF58 测出环境温度并显示在 12864 液晶上**

### 1、实验描述：温度传感器-- MF58，可以学习到AD 的方法和温度数据处理局部电路图：

3、 用到的库文件

startup\_stm32f4xx.s system\_stm32f4xx.c stm32f4xx\_gpio.h stm32f4xx\_rcc.h stm32f4xx\_syscfg.h stm32f4xx\_adc.h stm32f4xx\_dma.h

4、用户编写的文件main.c 和 main.h stm32f4xx\_it.c delay.c 和 delay.h mf58.c 和 mf58.h

5、 文件部分源码

main.c 部分文件代码#include "stm32f4xx.h"

#include "main.h"

/\*

* 函数名：main
* 描述 ：主函数
* 输入 ：无
* 输出 ：无

\*/

int main(void)

{

double okdata;

uint16\_t num1,num2,num3,num4; uint32\_t numdata;

/\*配置系统时钟 RCC\*/ SystemInit();

/\*LCD 端口初始化\*/ LCD\_12864\_Config();

/\*LCD 复位\*/ ResetLcd();

/\* 清 屏 \*/ ClrScreen();

ADC1\_GPIO\_Config(); ADC1\_Mode\_Config(); ADC\_SoftwareStartConv(ADC1);

while(1)

{

Temperature = GetTemp(); okdata = Temperature; okdata = okdata \*100; numdata = okdata;

num1 = numdata / 1000; numdata = numdata % 1000; num2 = numdata / 100; numdata = numdata % 100; num3 = numdata / 10;

num4 = numdata % 10;

printFront(4,0,0);//温printFront(4,16,1);//度printFront(4,32,2);//: printNumbers(4,48,num1);//0 printNumbers(4,56,num2);//0 printNumbers(4,64,10);//. printNumbers(4,72,num3);//0 printNumbers(4,80,num4);//0 printFront(4,96,3);//℃ Delay(0xFFFFEF); Delay(0xFFFFEF);

}

}

Mf58.c 部分文件代码#include "mf58.h"

#include "math.h" #include "arm\_math.h"

IO uint16\_t ADC1ConvertedValue = 0;

/\*

* 函数名：ADC1\_GPIO\_Config
* 描述 ：使能 ADC1 和 DMA1 的时钟，初始化 PC.01
* 输入 : 无
* 输出 ：无
* 调用 ：内部调用

\*/

void ADC1\_GPIO\_Config(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure; RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOA, ENABLE);

/\* Configure PC.01 as analog input \*/ GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_3; GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AN; GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_NOPULL;

GPIO\_Init(GPIOA, &GPIO\_InitStructure); // PC1,输入时不用设置速率

}

/\* 函数名：ADC1\_Mode\_Config

* 描述 ：配置 ADC1 的工作模式为 MDA 模式
* 输入 : 无
* 输出 ：无
* 调用 ：内部调用

\*/

void ADC1\_Mode\_Config(void)

{ //AD 初始化

ADC\_InitTypeDef ADC\_InitStructure; ADC\_CommonInitTypeDef ADC\_CommonInitStructure; DMA\_InitTypeDef DMA\_InitStructure;

/\* Enable ADC3, DMA2 and GPIO clocks \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_DMA2 , ENABLE); RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC1, ENABLE);

/\* DMA2 Stream0 channel0 configuration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ DMA\_InitStructure.DMA\_Channel = DMA\_Channel\_0; //指定通道为 2（根据芯片资料查找） DMA\_InitStructure.DMA\_PeripheralBaseAddr = (uint32\_t)ADC1\_DR\_ADDRESS;// 指 定

ADC3 的数据寄存器地址（根据芯片资料查找）

DMA\_InitStructure.DMA\_Memory0BaseAddr = (uint32\_t)&ADC1ConvertedValue;

DMA\_InitStructure.DMA\_DIR = DMA\_DIR\_PeripheralToMemory;//数据传输方向：从外设到内存

DMA\_InitStructure.DMA\_BufferSize = 1; //指定缓存器大小：1 个单位DMA\_InitStructure.DMA\_PeripheralInc = DMA\_PeripheralInc\_Disable;//禁止外设地址自加DMA\_InitStructure.DMA\_MemoryInc = DMA\_MemoryInc\_Disable;//禁止内存地址自加DMA\_InitStructure.DMA\_PeripheralDataSize = DMA\_PeripheralDataSize\_HalfWord;// 外 设

数据宽度：16 位

DMA\_InitStructure.DMA\_MemoryDataSize = DMA\_MemoryDataSize\_HalfWord; //内存数据宽度：16 位

DMA\_InitStructure.DMA\_Mode = DMA\_Mode\_Circular; //DMA 模式：循环模式DMA\_InitStructure.DMA\_Priority = DMA\_Priority\_High; //优先级：高DMA\_InitStructure.DMA\_FIFOMode = DMA\_FIFOMode\_Disable;//FIFO 禁止使用DMA\_InitStructure.DMA\_FIFOThreshold = DMA\_FIFOThreshold\_HalfFull; DMA\_InitStructure.DMA\_MemoryBurst = DMA\_MemoryBurst\_Single; DMA\_InitStructure.DMA\_PeripheralBurst = DMA\_PeripheralBurst\_Single; DMA\_Init(DMA2\_Stream0, &DMA\_InitStructure);

DMA\_Cmd(DMA2\_Stream0, ENABLE);

/\* ADC Common Init \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ ADC\_CommonInitStructure.ADC\_Mode = ADC\_Mode\_Independent; //AD 执行模式：独立模

式

ADC\_CommonInitStructure.ADC\_Prescaler = ADC\_Prescaler\_Div2; //时钟预分频：2 分频

ADC\_CommonInitStructure.ADC\_DMAAccessMode = ADC\_DMAAccessMode\_Disabled;//

这个是用于多通道模式，所以此处选择禁止

//ADC\_CommonInitStructure.ADC\_TwoSamplingDelay = ADC\_TwoSamplingDelay\_5Cycles;

//两次采样之间延时 5 个时钟周期

ADC\_CommonInitStructure.ADC\_TwoSamplingDelay = ADC\_TwoSamplingDelay\_10Cycles; ADC\_CommonInit(&ADC\_CommonInitStructure);

/\* ADC3 Init \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ ADC\_StructInit(& ADC\_InitStructure);

ADC\_InitStructure.ADC\_Resolution = ADC\_Resolution\_12b;//指定 AD 转换的位数为 12 位

ADC\_InitStructure.ADC\_ScanConvMode = DISABLE; //SCAN 模式用于多路同时采集，此处选择禁止

ADC\_InitStructure.ADC\_ContinuousConvMode = ENABLE; //允许循环采集ADC\_InitStructure.ADC\_ExternalTrigConvEdge = ADC\_ExternalTrigConvEdge\_None; //无外

部触发源

ADC\_InitStructure.ADC\_DataAlign = ADC\_DataAlign\_Right;//数据对齐方式：右对齐ADC\_InitStructure.ADC\_NbrOfConversion = 1; //采集通道数：1 个通道ADC\_Init(ADC1, &ADC\_InitStructure);

/\* ADC3 regular channel12 configuration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// ADC\_RegularChannelConfig(ADC3, ADC\_Channel\_3, 1, ADC\_SampleTime\_3Cycles); ADC\_RegularChannelConfig(ADC1, ADC\_Channel\_3, 1, ADC\_SampleTime\_480Cycles);

/\* Enable DMA request after last transfer (Single-ADC mode) \*/ ADC\_DMARequestAfterLastTransferCmd(ADC1, ENABLE);

/\* Enable ADC3 DMA \*/ ADC\_DMACmd(ADC1, ENABLE);

/\* Enable ADC3 \*/ ADC\_Cmd(ADC1, ENABLE);

}

double GetTemp(void)

{

uint16\_t Temp[10]; uint16\_t AverTemp; uint16\_t Temp\_T; uint8\_t i,j;

double Temp\_Value; double AverTemp1; double RT;

Temp\_Value = 0;

AverTemp = 0;

AverTemp1 =0; RT=0;

Temp\_T =0;

for(i=0;i<10;i++) Temp[i] =0;

for(i=0;i<10;i++)//连续２４次 AD 采样

{

Temp[i] = ADC1ConvertedValue;

}

for(i = 0;i<10;i++)//把 AD 采样值排序

{

for(j=0;j<10-i;j++)

{

if(Temp[j] >= Temp[j+1]) Temp\_T =Temp[j]; Temp[j] = Temp[j+1]; Temp[j+1] = Temp\_T;

}

}

for(i = 1;i<9;i++)//去处最小的四个，最大的四个，进行平均AverTemp += Temp[i];

AverTemp >>= 3;

AverTemp1=(AverTemp\*3.3)/4096.0;

RT=(AverTemp1\*10000)/(3.3- AverTemp1)-1000;//2012.11.22 要减去串到温度中的1K 电阻

Temp\_Value=1/((log(RT/5000))/3470+1/298.15)-273.15;

if( (Temp\_Value<(-45.0) )|| ( Temp\_Value>105.0))//超出温度范围Temp\_Value=15.0;

return Temp\_Value;

}

实验 7 利用超声换能器实现测距功能

综述：我司的开发板+换能器（测距型）实现测距显示功能是通过单片机在 PA1 口输出与换能器同频的PWM 信号（反向的），经过反向，放大升压成实际的驱动换能器的信号，信号在空间传输碰到被测物体反射回来，被换能器接收到，经过放大后从 PA2 口进入单片机。我们通过 AD 采样获取到信号后，通过阀值判断出发射波和回波，得出时间间隔 t,由 s=1/2\*V\*t 计算出距离。

1、换能器

我们换能器又称超声波探头，主要由压电陶瓷构成，既可以发射声波又可以接受声波。在接受到高频电压驱动信号时，压电陶瓷产生振动，激励周围空气质点产生振动，由于空气具有惯性和弹性，在空气质点的相互作用下，振动物体周围的空气就交替地产生压缩与膨胀，并且逐渐向外传播而形成声波。（反之， 声波信号传到换能器时，换能器会输出电压信号）。

换能器盲区：发射波和回波能明显分开的最小距离

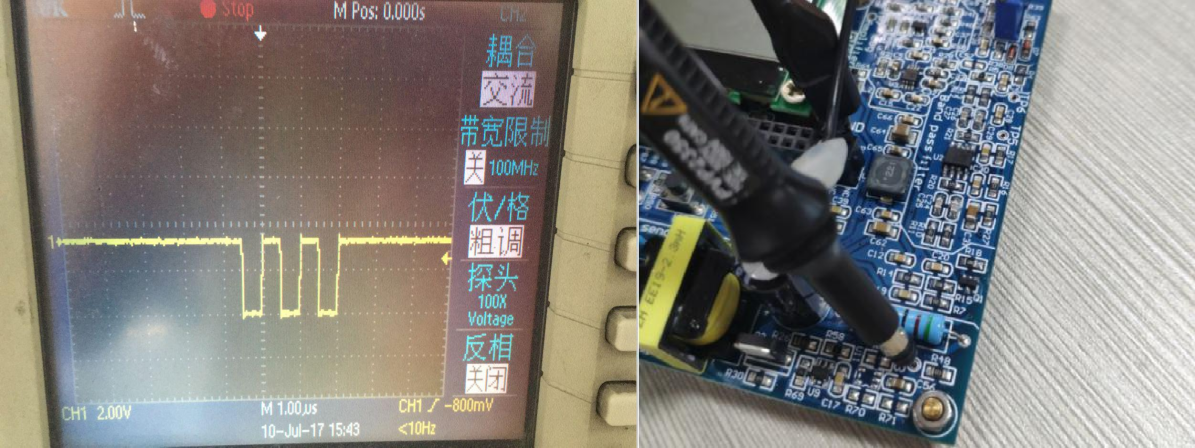
换能器信号拖尾：高频驱动电压结束后，换能器的振动不能立即停止，在驱动信号后就会产生拖尾（实际 发射波数量大于驱动信号的数量）

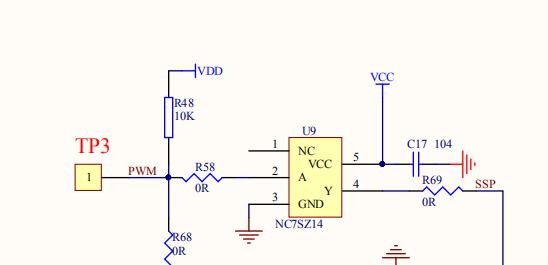
探头的指向性：有小角度的（例如半功率角@-3dB:4.1°±1，锐度角 9.7°±2）适用于测距也有水平无方向的，不适用与单向测距

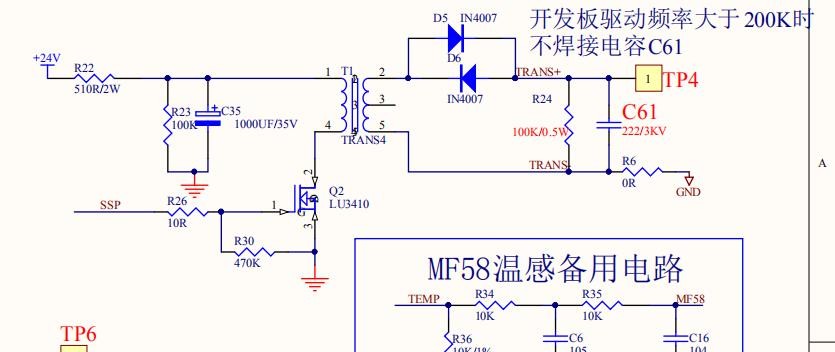
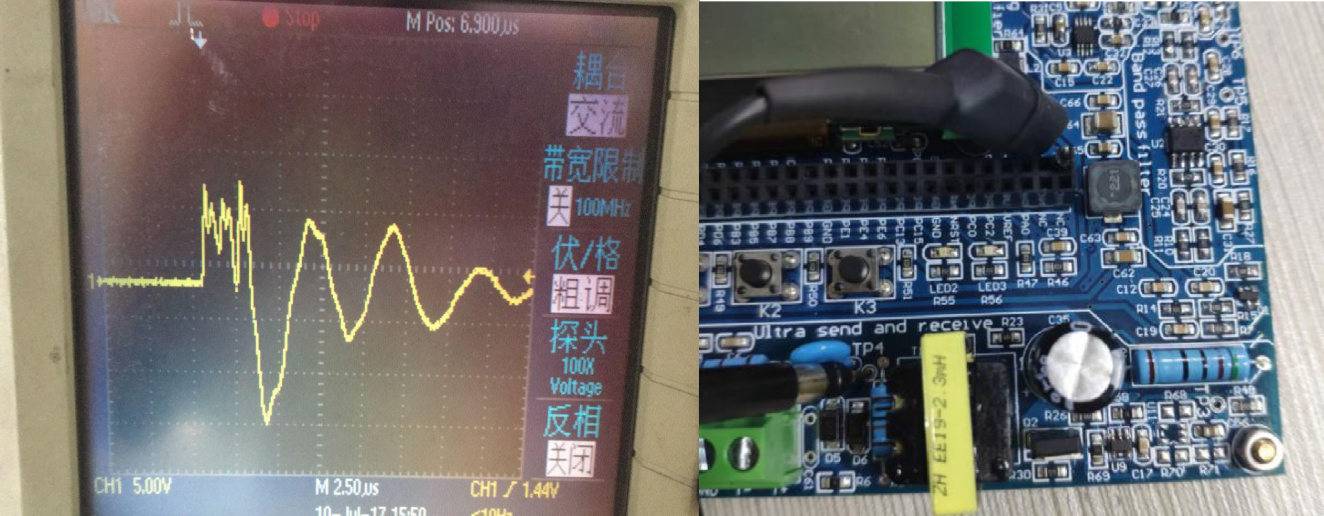
2、硬件信号（以空气中 300K 换能器为例） 开发板需要用户自行外接 24V 电源。

在 TP1 测试电是 5V 电源信号在 TP2 测试点是 3.3V 电源信号

首先单片机在 PA1 口产生一定数量的（一般 1-50 个脉冲）3.3V 的反向 PWM 信号

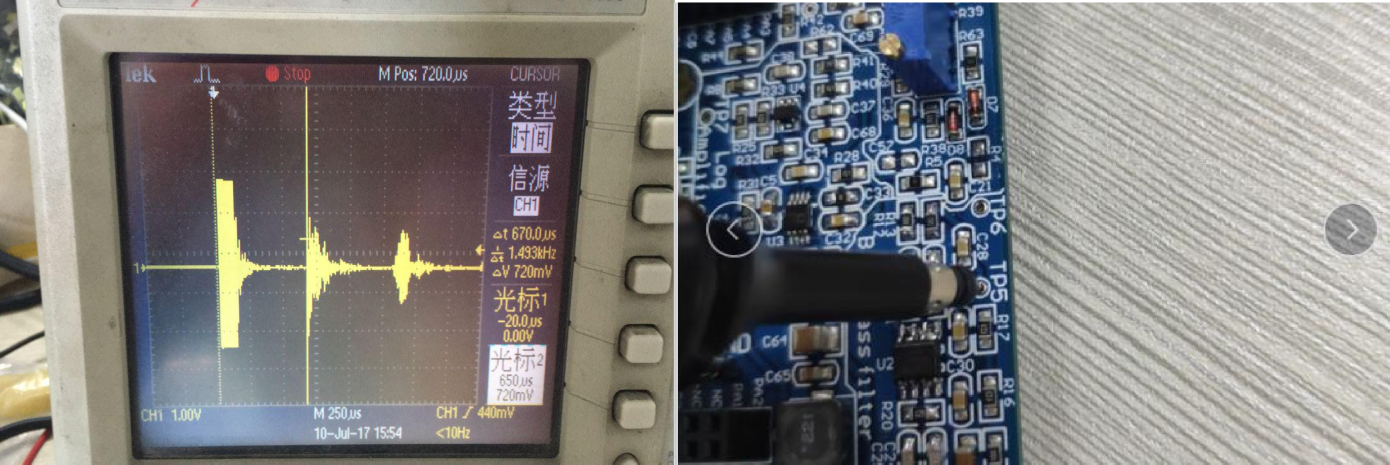


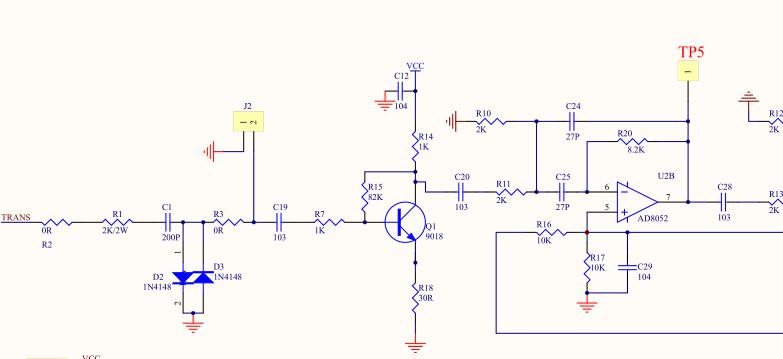


信号经过反向放大,通过 mos 管，变压器放大后得到实际的驱动信号，在 TP4 可测的峰值 200 多 V.

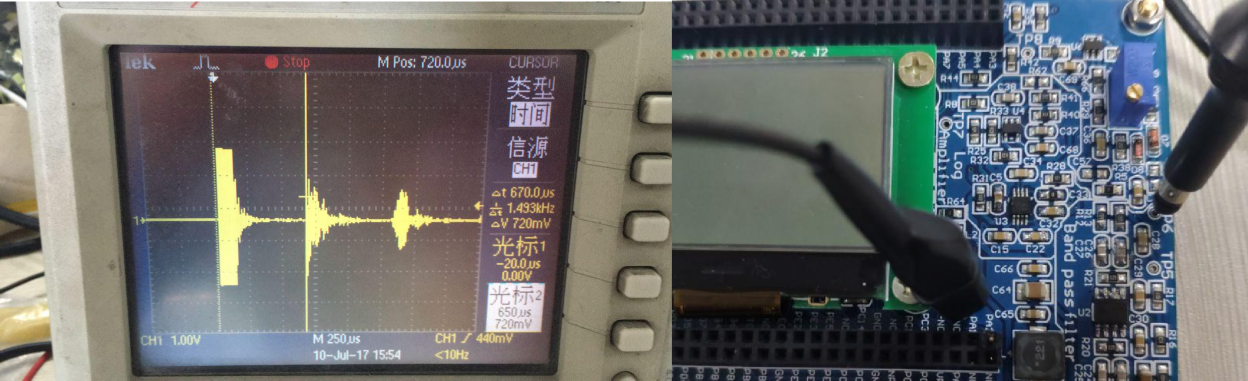
自发自收信号通过 TRANS 过来经过限幅（双探头从 J2 过来）经过 3 级管，带通滤波一级放大后可在

TP5 点测的信号

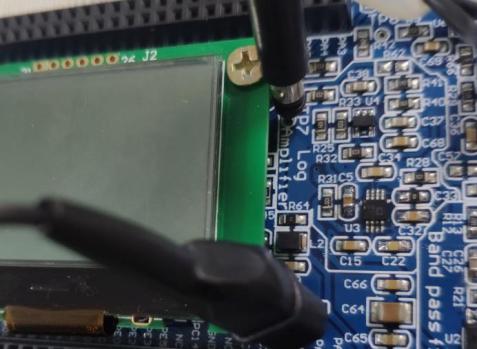




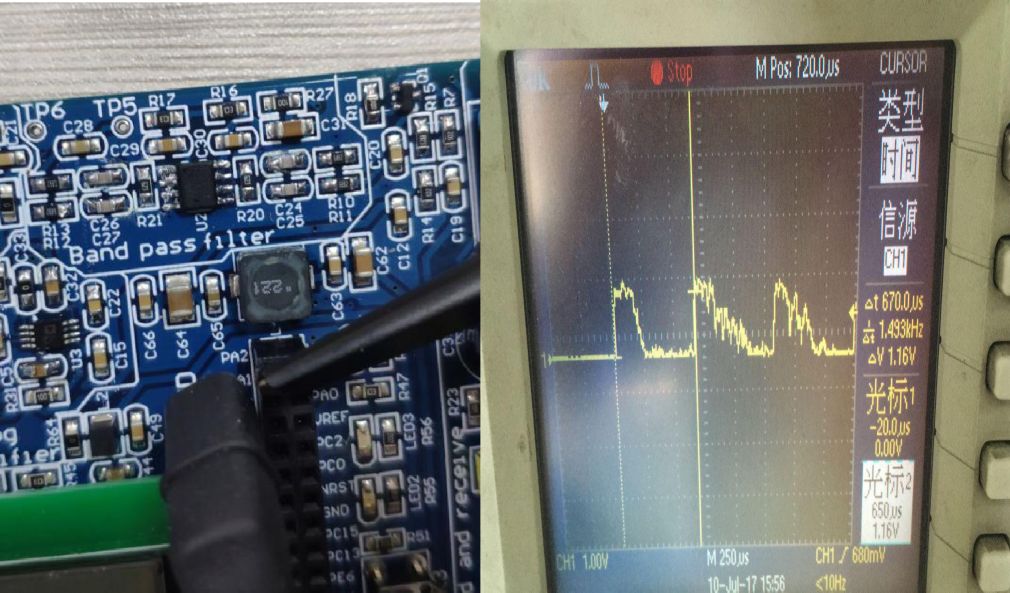
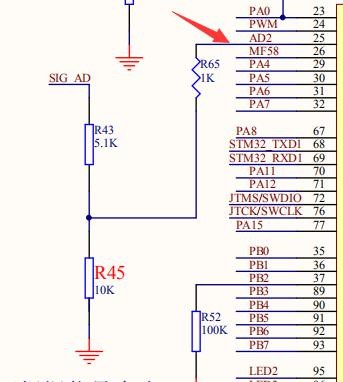
再经过带通滤波第二级放大，可在 TP6 点上测的信号



默认采用 A/D 采样方式 2，经过 AD8310 放大整形成单向包罗信号，再由 LM358 运放放大后可在 TP7 点测的信号



经过限幅进入单片机可在 PA2 测的信号

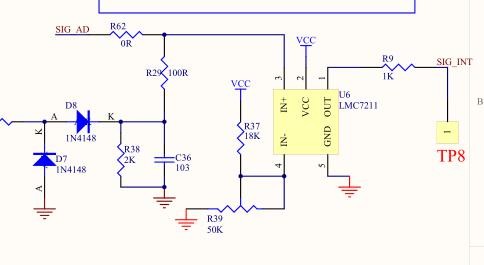
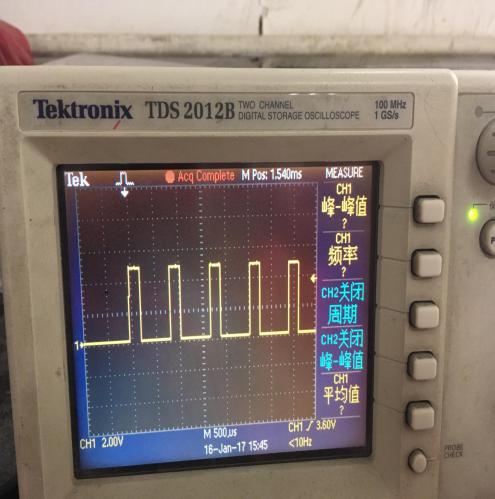


补充说明：使用小于 200K 的换能器时，驱动电压会达到 800 多 V

使用不同的探头的，在不同的距离下，信号的幅度，发射波和回波的时间差，以及盲区会有所差异，以实测为准

使用水平无方向的换能器是不能正确测量距离的。

AD 采样方式 1 是经过 rc 滤波后通过比较器，输出脉冲信号的。大于阈值的输出高电平，小于阈值的输出低电平，可通过调节可调电阻改变比较器的阈值。这个信号留给客户自行发挥。



3 软件

我们的简易测距代码是通过在 PA1 口输出 PWM，在 PA2 通过（在一定的采样周期）AD 采样获取发射波和回波的信号放在一个数组里。通过判断回波幅度（阈值法）判断发射波和回波的所在位置，有点数乘以采样周期的出发射波和回波的时间差，再乘以声速得出距离值。