**第二十二章 超声波测距**

本章主要介绍有关超声波测距的知识，本章分为如下几个部分：

8.1 超声波测距简介

8.2 硬件设计

8.3 软件设计

8.4 下载验证

8.1 超声波测距简介

**HC-SR04超声波测距模块工作原理：**



1）采用IO口Trig触发测距，给至少10us的高电平信号；

2）模块自动发送8个40KHz的方波，自动检测是否有信号返回；

3）有信号返回，通过IO口Echo输出一个高电平，高电平持续的时间就是超声波从发射到返回的时间。测试距离=（高电平时间\*声速（340M/S））/2；

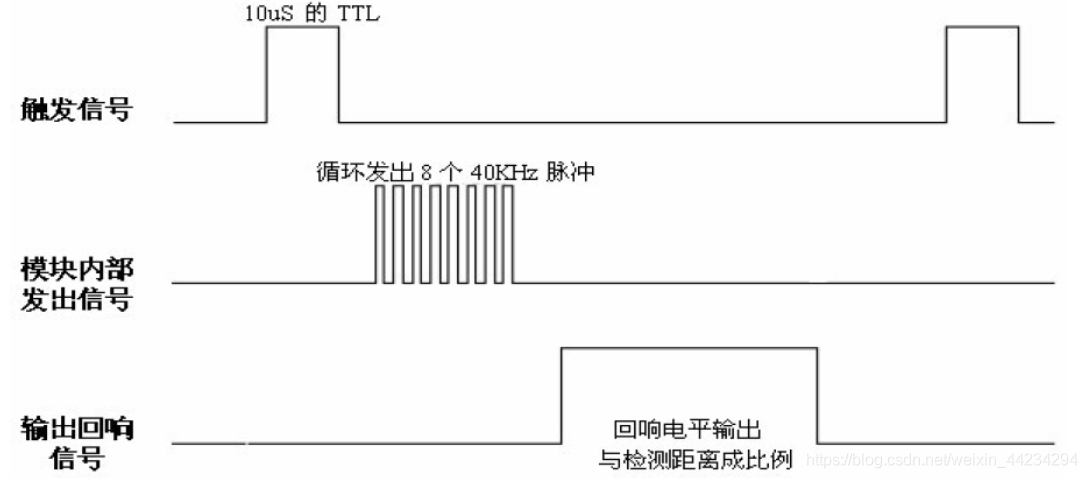
4）本模块使用方法简单，一个控制口发一个10us以上的高电平，就可以在接收口等待高电平输出。一有输出就可以开定时器计时，当此口变为低电平时就可以读定时器的值，此时就为此次测距的时间，方可算出距离。如此不断的周期测，即可以达到你移动测量的值。

图8.1.1 HC-SR04超声波模块的时序触发图

**编程步骤：**

1. 配置好相应GPIO，Trig和Echo引脚，将Trig和Echo端口都置低；

2. 配置定时器，开启中断，并记录中断产生次数；

3. 给模块Trig端口发送大于10us的高电平信号，发出回响信号时，Echo端呈现高电平，此时打开定时器计时；当收到回响信号时，Echo端呈现低电平，此时关闭定时器；

4. 获取Echo高电平时间，利用相关公式计算出距离 = (高电平时间\*声速（340M/S）)/2，可取平均值获取更精准数据。

因为单片机的定时器一般使用us进行计时，所以公式可以转换为，单位为cm。

我们作一下**单位换算**，，单位为cm/us，即为：0.034cm/us

再换一个角度，1/（0.034 cm/us），即：29 us/cm， 1cm就是29us。

但是发送后到接收到回波，声音走过的是2倍的距离，所以实际距离就是1cm，对应58us。换成距离cm，就是除以58，即：。

8.2 硬件设计

本章需要用到的硬件资源有：

1）极风STM32开发板

2）STLINK下载器

3）HC-SR04超声波模块

4）OLED屏

下面介绍一下STM32开发板和HC-SR04超声波模块的连接，由于程序中使用的是TIM2，图8.2.1中红框内为所需要用到的引脚，所以HC-SR04超声波模块的Vcc、Trig、Echo、Gnd分别连在STM32开发板的3.3V、PA2、PA3、GND。总体连接实物图如图8.2.2所示：



图8.2.1 引脚分配图

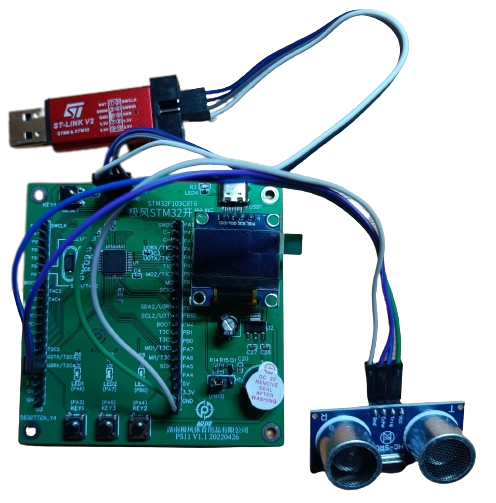


图8.2.2 总体连接实物图

8.3 软件设计

（1）程序实现的功能

①利用TIM2来实现定时器计时来获取Echo高电平时间，并在OLED屏上显示平均距离数据（这里的平均距离数据是取5次距离数据的平均值）；

②还使用了TIM3来进行1s计时，可打开串口查看1s内测量距离的所有数据、平均距离数据以及平均距离数据的刷新次数。

（2）程序的实现

打开**实验1 超声波测距\motor.uvprojx**，我们可以看到工程中拥有9个源文件，分别是led.c、led.h、oled.c、oled.h、ultrasonsic.c、ultrasonsic.h、atim.c、atim.h、main.c。其中ultrasonsic.c存放TIM2和超声波模块配置的代码，实现程序功能①；atim.c存放TIM3的驱动代码，实现程序功能②，main.c文件存放应用代码。

打开ultrasonsic.c文件，代码如下：

#include "ultrasonsic.h"

#include "delay.h"

u8 msHcCount = 0; // ms计数

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

\* 函数全称:

\* void Hcsr04\_Init()

\*

\* 函数作用:

\* 初始化超声波模块和TIM2

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

void Hcsr04\_Init()

{

TIM\_TimeBaseInitTypeDef TIM\_TimeBaseStructure;

NVIC\_InitTypeDef NVIC\_InitStruct;

GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA,ENABLE);

/\* IO初始化 \*/

GPIO\_InitStructure.GPIO\_Pin =GPIO\_Pin\_2; // 发送电平引脚TX

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_Out\_PP; // 设置推挽输出

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

GPIO\_ResetBits(GPIOA,GPIO\_Pin\_2); //一开始低电平

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_3; // 返回电平引脚

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IN\_FLOATING; //浮空输入

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

GPIO\_ResetBits(GPIOA,GPIO\_Pin\_3); // 默认低电平

/\* 定时器初始化 使用TIM2 \*/

RCC\_APB1PeriphClockCmd(TIMx\_CLK, ENABLE); // 使能对应RCC时钟

TIM\_DeInit(TIMx); // 配置定时器基础结构体

//自动重装载值寄存器的值，累计TIMx\_Period+1个频率后产生一个更新或者中断

TIM\_TimeBaseStructure.TIM\_Period = TIMx\_Period; // 设置周期为1000us

TIM\_TimeBaseStructure.TIM\_Prescaler = TIMx\_Prescaler; // 时钟预分频数

TIM\_TimeBaseStructure.TIM\_ClockDivision = TIM\_CKD\_DIV1; // 时钟分频因子

//计数器计数模式，设置向上计数

TIM\_TimeBaseStructure.TIM\_CounterMode = TIM\_CounterMode\_Up;

TIM\_TimeBaseInit(TIMx, &TIM\_TimeBaseStructure);

TIM\_ClearFlag(TIMx, TIM\_FLAG\_Update); //清除定时器的状态标志位

TIM\_Cmd(TIMx,ENABLE);

TIM\_ITConfig(TIMx,TIM\_IT\_Update,ENABLE); //打开定时器更新中断

NVIC\_InitStruct.NVIC\_IRQChannel = TIM2\_IRQn; //设置中断来源

NVIC\_InitStruct.NVIC\_IRQChannelCmd = ENABLE;

NVIC\_InitStruct.NVIC\_IRQChannelPreemptionPriority = 1; //设置主优先级

NVIC\_InitStruct.NVIC\_IRQChannelSubPriority = 1; //设置次优先级

NVIC\_Init(&NVIC\_InitStruct); //初始化

}

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

\* 函数全称:

\* void TIM2\_IRQHandler(void)

\*

\* 函数作用:

\* 定时器2中断服务程序

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

void TIM2\_IRQHandler(void)

{

if(TIM\_GetITStatus(TIMx,TIM\_IT\_Update) != RESET){

TIM\_ClearITPendingBit(TIMx,TIM\_IT\_Update); //清除更新中断标志位

msHcCount++; //溢出次数

}

}

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

\* 函数全称:

\* u32 GetEchoTimer(void)

\*

\* 函数作用:

\* 获取定时器时间

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

u32 GetEchoTimer(void)

{

u32 time = 0;

/\*当回响信号很长是，计数值溢出后重复计数，msHcCount用中断来保存溢出次数\*/

time = msHcCount\*1000;//msHcCount毫秒，time微妙

time += TIM\_GetCounter(TIMx);//获取计TIM2数寄存器中的计数值，以便计算回响信号时间

TIMx->CNT = 0; //将TIM2计数寄存器的计数值清零

delay\_ms(50);

return time;

}

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

\* 函数全称:

\* float Hcsr04GetLength(void)

\*

\* 函数作用:

\* 获取距离数据

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

float Hcsr04GetLength(void)

{

/\*测5次数据计算一次平均值\*/

float length = 0;

float t = 0;

float sum = 0;

u16 j = 0;

while(j != 5){

GPIO\_SetBits(GPIOA,GPIO\_Pin\_2); // Trig拉高信号，发出高电平

delay\_us(20); //持续时间超过10us

GPIO\_ResetBits(GPIOA,GPIO\_Pin\_2);

/\* Echo发出信号 等待回响信号 \*/

/\* 输入方波后，模块会自动发射8个40KHz的声波，与此同时回波引脚（Echo）端的电平会由0变为1（此时应该启动定时器计时）；

当超声波返回被模块接收到时，回波引脚端的电平会由1变为0（此时应该停止定时器计数），定时器记下的这个时间即为超声波由发射到返回的总时长；\*/

while(GPIO\_ReadInputDataBit(GPIOA,GPIO\_Pin\_3) == 0); //Echo等待回响

/\* 开启定时器 \*/

TIM\_SetCounter(TIMx,0); //清除计数器

msHcCount = 0;

TIM\_Cmd(TIMx,ENABLE); //使能定时器

j = j+1; //每收到一次回响信号+1，收到5次就计算均值

while(GPIO\_ReadInputDataBit(GPIOA,GPIO\_Pin\_3) == 1);

/\* 关闭定时器 \*/

TIM\_Cmd(TIMx,DISABLE);

/\* 获取Echo高电平时间时间 \*/

t = GetEchoTimer();

length = (float)t/58;//单位为cm

printf("dis = %fcm\r\n",length);//打印当前计算出的距离数据

sum += length;

}

length = sum/5;//五次平均值，求平均距离数据

printf("dis\_average = %fcm\r\n",length); //打印平均距离

return length;

}

下面我们看看头文ultrasonsic.h的代码，代码中包含一些头文件定义，数据类型、函数声明，代码如下：

#ifndef \_ULTRASONSIC\_H

#define \_ULTRASONSIC\_H

#include "stdlib.h"

#include "stm32f10x.h"

/\*TIM2\*/

#define TIMx TIM2

#define TIMx\_CLK RCC\_APB1Periph\_TIM2

#define TIMx\_IRQn TIM2\_IRQn

#define TIMx\_Period (1000-1)

#define TIMx\_Prescaler (72-1)

void Hcsr04\_Init(void); //定时中断初始化函数

float Hcsr04GetLength(void); //获取距离数据函数

#endif

最后，我们在main函数里编写应用代码，main.c文件如下：

打开atim.c文件，代码如下：

#include "atim.h"

#include "oled.h"

extern int i;

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

\* 函数全称:

\* void TIM3\_init(uint16\_t arr, uint16\_t pre)

\*

\* 函数作用:

\* 初始化TIM3

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

void TIM3\_init(uint16\_t arr, uint16\_t pre)

{

TIM\_TimeBaseInitTypeDef TIM\_TimeBaseStructure;

NVIC\_InitTypeDef NVIC\_InitStruct;

/\* 定时器初始化 使用TIM3 \*/

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM3, ENABLE); //使能对应RCC时钟

TIM\_DeInit(TIM3);//配置定时器基础结构体

//自动重装载值寄存器的值，累计TIMx\_Period+1个频率后产生一个更新或者中断

TIM\_TimeBaseStructure.TIM\_Period = arr;//设置周期为1s

TIM\_TimeBaseStructure.TIM\_Prescaler = pre;//时钟预分频数

TIM\_TimeBaseStructure.TIM\_ClockDivision = TIM\_CKD\_DIV1; //时钟分频因子

//计数器计数模式，设置向上计数，

TIM\_TimeBaseStructure.TIM\_CounterMode = TIM\_CounterMode\_Up;

//重复计数器的值

//TIM\_TimeBaseStructure.TIM\_RepetitionCounter=0 ;

TIM\_TimeBaseInit(TIM3, &TIM\_TimeBaseStructure);

TIM\_ClearFlag(TIM3, TIM\_FLAG\_Update); //清除定时器的状态标志位

TIM\_Cmd(TIM3,ENABLE);

TIM\_ITConfig(TIM3,TIM\_IT\_Update,ENABLE); //打开定时器更新中断

NVIC\_InitStruct.NVIC\_IRQChannel = TIM3\_IRQn; //设置中断来源

NVIC\_InitStruct.NVIC\_IRQChannelCmd = ENABLE;

NVIC\_InitStruct.NVIC\_IRQChannelPreemptionPriority = 1; //设置主优先级

NVIC\_InitStruct.NVIC\_IRQChannelSubPriority = 1; //设置次优先级

NVIC\_Init(&NVIC\_InitStruct); //初始化

}

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

\* 函数全称:

\* void TIM3\_IRQHandler(void)

\*

\* 函数作用:

\* 定时器3中断服务程序

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

void TIM3\_IRQHandler(void)

{

if(TIM\_GetITStatus(TIM3,TIM\_IT\_Update) != RESET){

TIM\_ClearITPendingBit(TIM3,TIM\_IT\_Update); //清除更新中断标志位

printf("cnt=%d\n",i); //打印1s内平均距离数据的刷新次数

i=0;

}

}

下面我们看看头文atim.h的代码，代码中包含一些头文件定义，数据类型、函数声明，代码如下：

#ifndef \_ATIM\_H

#define \_ATIM\_H

#include "stdlib.h"

#include "stm32f10x.h"

#include "usart.h"

#include "ultrasonsic.h"

void TIM3\_init(uint16\_t arr, uint16\_t pre);

#endif

最后，我们在main函数里编写应用代码，main.c文件如下：

#include "stm32f10x.h"

#include <string.h>

#include <stdlib.h>

#include "stdio.h"

#include "usart.h"

#include "delay.h"

#include "oled.h"

#include "led.h"

#include "ultrasonsic.h"

#include "atim.h"

int i=0;

int main(void)

{

float length;

char str[32];

uart\_init(115200); /\* 初始化串口 \*/

LED\_Init(); /\* 初始化LED \*/

OLED\_Init(); /\* 初始化OLED \*/

OLED\_Clear(); /\* 清屏 \*/

Hcsr04\_Init();

TIM3\_init(10000 -1, 7200 -1);

{

//蜂鸣器初始化

GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB, ENABLE);

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_13; /\* BEEP引脚 \*/

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_Out\_PP; /\* 推挽输出 \*/

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz; /\* 高速 \*/

GPIO\_Init(GPIOB, &GPIO\_InitStructure);

GPIO\_ResetBits(GPIOB, GPIO\_Pin\_13); /\* 关闭BEEP \*/

}

while(1)

{

i++;

length=Hcsr04GetLength(); //获取距离

OLED\_ShowCHinese(0, 0, 6); //距

OLED\_ShowCHinese(16, 0, 7); //离

OLED\_ShowCHinese(32, 0, 5 ); //：

sprintf(str,"%f",length);

OLED\_ShowString(40,1,str,8); //OLED屏显示平均距离数据

if(length < 10)

{//小于10cm

GPIO\_ResetBits(GPIOA, GPIO\_Pin\_6); //LED1灭

GPIO\_SetBits(GPIOA, GPIO\_Pin\_7); //LED2亮

GPIO\_SetBits(GPIOB, GPIO\_Pin\_13); /\* 打开BEEP \*/

delay\_ms(300);

}else

{//正常距离

GPIO\_SetBits(GPIOA, GPIO\_Pin\_6); //LED1亮

GPIO\_ResetBits(GPIOA, GPIO\_Pin\_7); //LED1灭

GPIO\_ResetBits(GPIOB, GPIO\_Pin\_13);/\* 关闭BEEP \*/

}

}

}

至此，我们的软件设计部分就结束了。

8.4 下载验证

在代码编译成功，下载程序完成后，可观察到OLED屏显示平均距离数据，打开串口可以查看1s内测量距离的所有数据、平均距离数据以及平均距离数据的刷新次数，即成功完成实验1 超声波测距。