基于STM32F1的超声波HCSR04模块测距

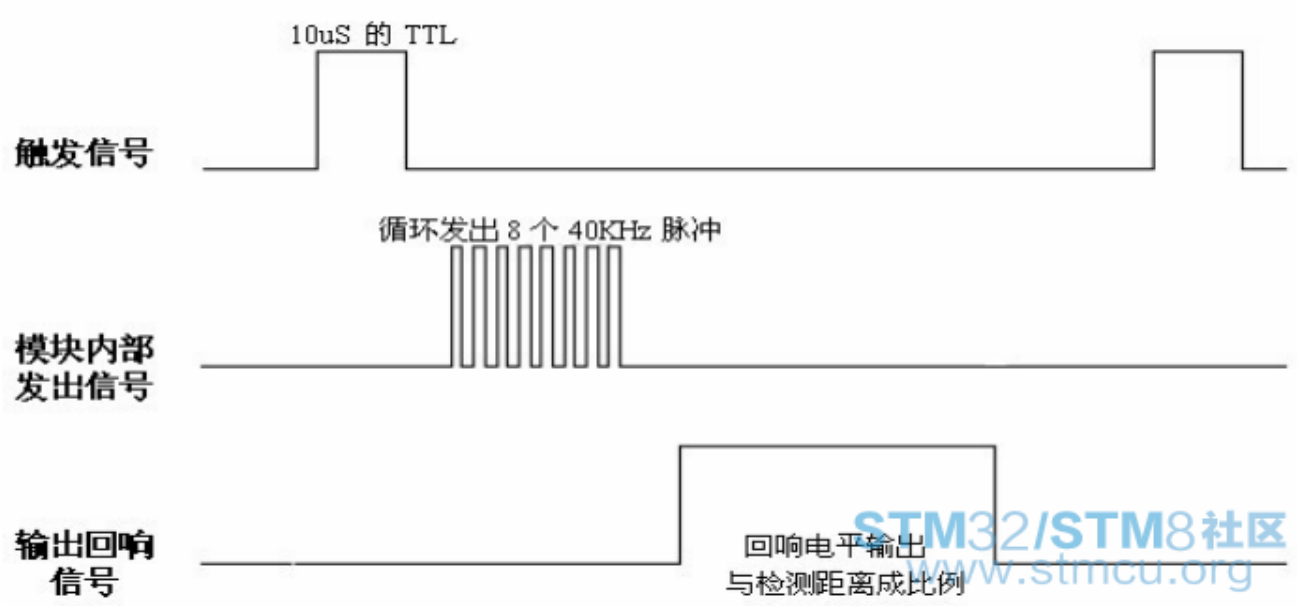
1. HCSR04模块

HC-SR04超声波测距模块可提供**2cm-400cm**的非接触式距离感测功能，测距精度可达高到**3mm**；模块包括**超声波发射器、接收器与控制电路。**像智能小车的测距以及转向，或是一些项目中，常常会用到。智能小车测距可以及时发现前方的障碍物，使智能小车可以及时转向，避开障碍物。

`Vcc-GND 5V电源输入；Trig：触发信号输入；Echo:回响信号输出`

2. 工作原理

- 采用IO口TRIG触发测距，给最少10us的高电平信号。
- 模块自动发送8个40khz的方波，自动检测是否有信号返回。
- 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 $s = \frac{\text{高电平时间}t \times \text{声速（}v=340m/s\text{）}}{2}$



以上时序图表明你只需要提供一个 10uS以上脉冲触发信号，该模块内部将 发出8个 40kHz周期电平并检测回波。一旦检测到有回波信号则输出回响信号。 回响信号的脉冲宽度与所测的距离成正比。 由此通过发射信号到收到的回响信号时间间隔可以计算得到距离。

~~为了防止测试信号对回响信号的影响，建议测量周期~~$\geq 60ms$~~,即频率小于16HZ。~~这句话存疑，根据最大测量距离3m计算，只需要保证周期>20m即可，实际测量最大距离大概为2m-3m。所以实际最大频率可以达到50Hz。

由于回响电平持续时间为us级，对以上公式可以化简：$s = \frac{us \times 340 \times 100}{10^6 \times 2} cm = 0.017 \times us \approx \frac{us}{58} cm(\text{或者 } \frac{us}{59} cm)$

3. HCSR04测距实现

3.1 硬件连接

- 超声波模块电源：`Vcc-GND: 5V输入`
- 超声波模块测距触发引脚：`Trig：PA1`
- 超声波模块测距信号返回引脚：`Echo:PA0`

3.2 软件控制

1. **实现方法**：20ms触发一次测距，在定时器100us的中断里面计算模块Echo返回信号高电平的时间，然后通过公式计算出距离，并串口打印显示。注意：20ms触发一次测距，说明最远只能测3.4m。
2. **代码部分**：含有`主函数main.c,超声波模块控制函数（timer.h,timer.c）,程序运行状态指示函数（led.c,led.h）`五个程序文件。

```
//led.h
#ifndef __LED_H
#define __LED_H
#include "sys.h"
#define LED0 PAout(8)   // PA8
#define LED1 PDout(2)   // PD2

void LED_Init(void);//初始化

#endif
```

```
/**************************************************************/
//timer.h
#ifndef __TIMER_H
#define __TIMER_H
#include "sys.h"



void TIM1_PWM_Init(u16 arr,u16 psc);
void TIM2_Cap_Init(u16 arr,u16 psc);
void Wave_SRD_Strat(void);

#endif
/**************************************************************/
//led.c
#include "led.h"
//初始化PA.8和PD.2为输出口.并使能这两个口的时钟
//LED IO初始化
void LED_Init(void)
{

 GPIO_InitTypeDef  GPIO_InitStructure;

 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOD, ENABLE);  //使能PA,PD端口时钟

 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;               //LED0-->PA.8 端口配置
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;        //推挽输出
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;       //IO口速度为50MHz
 GPIO_Init(GPIOA, &GPIO_InitStructure);                  //根据设定参数初始化GPIOA.8
 GPIO_SetBits(GPIOA,GPIO_Pin_8);                         //PA.8 输出高

 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;               //LED1-->PD.2 端口配置, 推挽输出
 GPIO_Init(GPIOD, &GPIO_InitStructure);                  //推挽输出 ，IO口速度为50MHz
 GPIO_SetBits(GPIOD,GPIO_Pin_2);                         //PD.2 输出高
}


/**************************************************************/
//timer.c
#include "timer.h"
#include "led.h"
#include "usart.h"
#include "sys.h"
#include "delay.h"

#define Trig GPIO_Pin_1

#define Echo GPIO_Pin_0

//PWM输出初始化
//arr：自动重装值
//psc：时钟预分频数
void TIM1_PWM_Init(u16 arr,u16 psc)
```

```
{
     GPIO_InitTypeDef GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    TIM_OCInitTypeDef  TIM_OCInitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);//
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA , ENABLE);  //使能GPIO外设时钟使能


   //设置该引脚为复用输出功能,输出TIM1 CH2的PWM脉冲波形
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8; //TIM_CH2
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;  //复用推挽输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);


    TIM_TimeBaseStructure.TIM_Period = arr;//设置在下一个更新事件装入活动的自动重装载寄存器周期的值
80K
    TIM_TimeBaseStructure.TIM_Prescaler =psc; //设置用来作为TIMx时钟频率除数的预分频值  不分频
    TIM_TimeBaseStructure.TIM_ClockDivision = 0; //设置时钟分割:TDTS = Tck_tim
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  //TIM向上计数模式
    TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure); //根据TIM_TimeBaseInitStruct中指定的参数初始化
TIMx的时间基数单位


    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2; //选择定时器模式:TIM脉冲宽度调制模式2
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; //比较输出使能
    TIM_OCInitStructure.TIM_Pulse = 0; //设置待装入捕获比较寄存器的脉冲值
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High; //输出极性:TIM输出比较极性高
    TIM_OC1Init(TIM1, &TIM_OCInitStructure);  //根据TIM_OCInitStruct中指定的参数初始化外设TIMx

  TIM_CtrlPWMOutputs(TIM1,ENABLE);  //MOE 主输出使能

    TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);  //CH1预装载使能

    TIM_ARRPreloadConfig(TIM1, ENABLE); //使能TIMx在ARR上的预装载寄存器

    TIM_Cmd(TIM1, ENABLE);  //使能TIM1


}

//定时器2通道1输入捕获配置

TIM_ICInitTypeDef  TIM2_ICInitStructure;

void TIM2_Cap_Init(u16 arr,u16 psc)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);    //使能TIM2时钟
```

```
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);  //使能GPIOA时钟
    //Echo
    GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_0;  //PA0 清除之前设置
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD; //PA0 输入
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    GPIO_ResetBits(GPIOA,GPIO_Pin_0);                        //PA0 下拉
    //Trig
    GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_1;  //PA1 清除之前设置
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; //PA1 输入出
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    GPIO_ResetBits(GPIOA,GPIO_Pin_1);                        //PA0 下拉

    //初始化定时器2 TIM2
    TIM_TimeBaseStructure.TIM_Period = arr; //设定计数器自动重装值
    TIM_TimeBaseStructure.TIM_Prescaler =psc;   //预分频器
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1; //设置时钟分割:TDTS = Tck_tim
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  //TIM向上计数模式
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure); //根据TIM_TimeBaseInitStruct中指定的参数初始化
TIMx的时间基数单位

    //初始化TIM2输入捕获参数
    TIM2_ICInitStructure.TIM_Channel = TIM_Channel_1; //CC1S=01     选择输入端 IC1映射到TI1上
    TIM2_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;    //上升沿捕获
    TIM2_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI; //映射到TI1上
    TIM2_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;   //配置输入分频,不分频
    TIM2_ICInitStructure.TIM_ICFilter = 0x00;//IC1F=0000 配置输入滤波器 不滤波
    TIM_ICInit(TIM2, &TIM2_ICInitStructure);

    //中断分组初始化
    NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;  //TIM2中断
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;  //先占优先级2级
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;  //从优先级0级
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //IRQ通道被使能
    NVIC_Init(&NVIC_InitStructure);  //根据NVIC_InitStruct中指定的参数初始化外设NVIC寄存器

    TIM_ITConfig(TIM2,TIM_IT_Update|TIM_IT_CC1,ENABLE);//允许更新中断 ,允许CC1IE捕获中断

  TIM_Cmd(TIM2,ENABLE );    //使能定时器2

}


u8  TIM2CH1_CAPTURE_STA=0;  //输入捕获状态
u16 TIM2CH1_CAPTURE_VAL;    //输入捕获值

//定时器5中断服务程序
void TIM2_IRQHandler(void)
{

    if((TIM2CH1_CAPTURE_STA&0X80)==0)//还未成功捕获
    {
        if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET)
```

```
        {
            if(TIM2CH1_CAPTURE_STA&0X40)//已经捕获到高电平了
            {
                if((TIM2CH1_CAPTURE_STA&0X3F)==0X3F)//高电平太长了
                {
                    TIM2CH1_CAPTURE_STA|=0X80;//标记成功捕获了一次
                    TIM2CH1_CAPTURE_VAL=0XFFFF;
                }else TIM2CH1_CAPTURE_STA++;
            }
        }
    if (TIM_GetITStatus(TIM2, TIM_IT_CC1) != RESET)//捕获1发生捕获事件
        {
            if(TIM2CH1_CAPTURE_STA&0X40)        //捕获到一个下降沿
            {
                TIM2CH1_CAPTURE_STA|=0X80;      //标记成功捕获到一次上升沿
                TIM2CH1_CAPTURE_VAL=TIM_GetCapture1(TIM2);
                TIM_OC1PolarityConfig(TIM2,TIM_ICPolarity_Rising); //CC1P=0 设置为上升沿捕获
            }else                               //还未开始,第一次捕获上升沿
            {
                TIM2CH1_CAPTURE_STA=0;          //清空
                TIM2CH1_CAPTURE_VAL=0;
                TIM_SetCounter(TIM2,0);
                TIM2CH1_CAPTURE_STA|=0X40;      //标记捕获到了上升沿
                TIM_OC1PolarityConfig(TIM2,TIM_ICPolarity_Falling);//CC1P=1 设置为下降沿捕获
            }
        }
    }

    TIM_ClearITPendingBit(TIM2, TIM_IT_CC1|TIM_IT_Update); //清除中断标志位

}


void Wave_SRD_Strat(void)
{
    GPIO_SetBits(GPIOA,Trig);   //将Trig设置为高电平
    delay_us(20);               //持续大于10us触发，触发超声波模块工作
    GPIO_ResetBits(GPIOA,Trig);

}

/**************************************************************************/
//main.c
#include "led.h"
#include "delay.h"
#include "sys.h"
#include "timer.h"
#include "usart.h"

extern u8  TIM2CH1_CAPTURE_STA;     //输入捕获状态
extern u16  TIM2CH1_CAPTURE_VAL;    //输入捕获值

float distance;
```

```
 int main(void)
 {

    u32 temp=0;
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);// 设置中断优先级分组2
    delay_init();            //延时函数初始化
    uart_init(9600);                //9600
    LED_Init();         //初始化与LED连接的硬件接口
    TIM1_PWM_Init(899,0);           //不分频；PWM频率=72000/(899+1)=80Khz
    TIM2_Cap_Init(0XFFFF,72-1);     //以1Mhz的频率计数
    while(1)
    {
        delay_ms(10);
        Wave_SRD_Strat();
//      TIM_SetCompare1(TIM1,TIM_GetCapture1(TIM1)+1);
//      if(TIM_GetCapture1(TIM1)==300)TIM_SetCompare1(TIM1,0);
        if(TIM2CH1_CAPTURE_STA&0X80)//成功捕获到了一次高电平
        {
            temp=TIM2CH1_CAPTURE_STA&0X3F;
            temp*=65536;                    //溢出时间总和
            temp+=TIM2CH1_CAPTURE_VAL;      //得到总的高电平时间
            distance = temp/59.0;
            printf("HIGH:%.2f cm\r\n",distance);    //打印总的高点平时间
            LED1=!LED1;
            TIM2CH1_CAPTURE_STA=0;          //开启下一次捕获
        }
    }
}
```

> 工程文件见ult文件夹。

3.3 测试结果

将`.hex`文件烧录到stm32中，连接硬件，打开串口下载软件，设置波特路为9600，得到测距返回结果。


XCOM V2.0
HIGH:20.53 cm
HIGH:136.49 cm
HIGH:24.22 cm
HIGH:123.53 cm
HIGH:20.37 cm
HIGH:16.27 cm
HIGH:14.47 cm
HIGH:9.46 cm
HIGH:7.63 cm
HIGH:7.12 cm
HIGH:0.05 cm
HIGH:73.97 cm
HIGH:20.14 cm
HIGH:109.25 cm
HIGH:105.08 cm
HIGH:28.27 cm
HIGH:86.73 cm
HIGH:95.14 cm
HIGH:17.07 cm
HIGH:8.27 cm
HIGH:12.17 cm
HIGH:4.32 cm
HIGH:6.07 cm
HIGH:14.68 cm
HIGH:6.76 cm
H
打印结果
波特率设置为9600
串口选择
COM4:USB-SERIAL
波特率
9600
停止位
1
数据位
8
奇偶校验
无
串口操作
打开串口
保存窗口
清除接收
16进制显示
白底黑字
RTS
DTR
时间戳(以换行回车断帧)
单条发送
多条发送
协议传输
帮助
发送
清除发送
定时发送
周期:
1000
ms
打开文件
发送文件
停止发送
16进制发送
发送新行
0%
开源电子网：www.openedv.com
www.openedv.com
S:0
R:14513
CTS=0 DSR=0 DCD=0
当前时间 12:50:58