全国大学生电子设计竞赛

获奖证书

全国大学生电子设计竞赛组织委员会

目录

基于边沿检测的声传播智能定位系统

摘要

我们采用了

关键字： 上升沿检测　　TDOA 算法

1 设计方案

1.1 方案的选择与比较

1.1.1 控制电路的选择

方案一：选用 K210 芯片

优点：K210 芯片搭载了双核 64 位 RISC-V 处理器和专门的 AI 加速器，具有高性能的计算能力；支持常见的图像处理任务，并且有多个 GPIO 接口、UART、I2C、SPI 等常用接口，方便与其他外设进行连接和扩展，满足不同应用场景的需求。

缺点：作为较新的芯片型号，某些功能可能尚未得到广泛验证和使用，因此在使用过程中可能存在一些潜在的问题或缺陷。

方案二：选用树莓派

优点：树莓派搭载了 ARM 架构的处理器，性能较强，能够处理多种复杂的算法和任务；具有多个 GPIO 接口、USB 接口、HDMI 接口等常见接口；拥有庞大的开源社区支持，有大量的资源、文档和示例代码可供参考和使用。

缺点：功耗较大，在长时间工作或需要低功耗的应用中不太适合；掌握起来比较困难。

方案三：选用树莓派 + STM32

优点：树莓派具有较强的计算能力和丰富的计算资源可以处理性能较强，能够处理多种复杂的算法和任务，STM32 微控制器具有强大的计算能力和丰富的接口，以及较低的功耗，可以承担控制算法和系统管理任务。

缺点：使用树莓派 + STM32 方案需要对图像处理、机器学习算法和嵌入式系统开发有一定的了解，对开发者而言具有一定的挑战性；采用这个方案需要同时使用树莓派和 STM32，增加了硬件的复杂性和成本。

综合考虑采用方案一，因为 STM32 平衡了性能与功耗，并且在发挥部分也能够更好地完成任务，适合该系统的需求

1.1.2 显示电路的比较与选择

方案一：选用数码管

优点：功耗低

缺点：难以显示复杂的内容，并且电路复杂，扩展能力差

方案二：选用 oled 屏

优点：功耗较低

缺点：对 gui 的支持差，绘图较复杂，开发时间较长

方案三：串口屏

优点：开发快速，图形绘制快捷，交互触摸功能完善

缺点：成本较高，功耗较高

综合考虑采用方案三，因为串口屏保证了开发速度，并且能够直观显示出实验结果，而且带有触摸功能，能够方便快捷地进行交互。

1.1.3 信号处理电路的比较与选择

方案一：敲击边沿转换

采用转换电路，将敲击信号经过变换，使之成为脉冲信号，根据脉冲的上升沿到达时间差，采用 TDOA 算法对声源进行计算定位。

优点：较易实现

缺点：扩展部分需要另外搭建电路

方案二：ADC 直采

对信号进行放大后，进行 AD 转换后，直接采集数据

优点：能够较为清楚地得到接收到的信号的每个时间的电压值

缺点：数据过多，难以处理

综合考虑采用方案一，保证基础部分以及发挥部分的完成进度。

1.2 方案的设计与论证

1.2.1 整体电路组成

电路主要部分有：恒压、恒流电源模块、信号处理电路、控制电路、显示电路

1.2.2 整体电路框架图

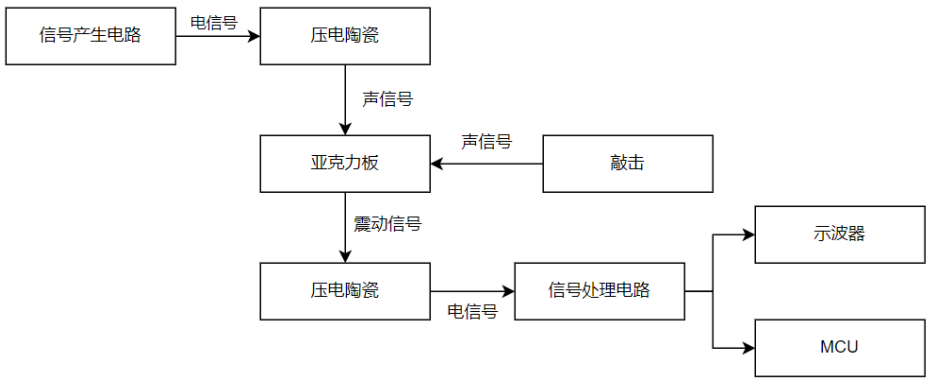

图 1　电路框架图

2 符号说明

表 **1** 符号与常数

符号	含义	单位或常数值
t_{prop}	传感器接收到信号的时间差	μs
d	距离	cm
x	敲击点横坐标	cm
y	敲击点纵坐标	cm
t_{OFF}	时间偏置	μs
T	温度	$°C$
v	声信号在亚克力板中的传播速度	$cm/\mu s$

3 理论分析与计算

假设我们有两个传感器，位置分别是 ($\mathbf{p}_1$) 和 ($\mathbf{p}_2$)，信号源的位置是 ($\mathbf{x}$)。设传感器 1 和传感器 2 接收到信号的时间分别为 (t_1) 和 (t_2)。

则，信号源到传感器 1 和传感器 2 的距离分别是：

$$d_1 = |\mathbf{x} - \mathbf{p}1| \quad \text{和} \quad d_2 = |\mathbf{x} - \mathbf{p}2|$$

根据声音的速度 (v)，我们可以得到两个传感器之间的声音传播时间：

$$t_{\text{prop}} = \frac{d_1 - d_2}{v}$$

而传感器测量到的到达时间差（TDOA）为 ($t_1 - t_2$)。假设传感器的时钟同步误差可以忽略不计，则有：

$$t_1 - t_2 = t_{\text{prop}}$$

因此，我们可以将 (t_{prop}) 表示为测量值 (Δt)：

$$\Delta t = t_1 - t_2 = \frac{d_1 - d_2}{v}$$

这就是 TDOA 算法的基本方程。

接下来，我们可以将距离 (d_1) 和 (d_2) 表示为欧氏距离：

$$d_1 = \sqrt{(x_1 - p_{1x})^2 + (x_2 - p_{1y})^2 + (x_3 - p_{1z})^2}$$

$$d_2 = \sqrt{(x_1 - p_{2x})^2 + (x_2 - p_{2y})^2 + (x_3 - p_{2z})^2}$$

其中，(x_1),(x_2),(x_3) 分别是信号源的 x, y, z 坐标，$(p_{1x}), (p_{1y}), (p_{1z})$ 和 $(p_{2x}), (p_{2y}), (p_{2z})$ 分别是传感器 1 和传感器 2 的 x, y, z 坐标。

然后，我们可以将上述方程组带入 TDOA 的基本方程中，得到关于 $(x_1), (x_2), (x_3)$ 的非线性方程组。通常情况下，我们需要使用数值优化方法（如最小二乘法、梯度下降等）来求解这个方程组，从而估计出信号源的位置 $((x_1, x_2, x_3))$。

这就是 TDOA 算法的完整推导过程，其中包括了声速传播、距离计算和非线性方程求解。

4　硬件电路设计

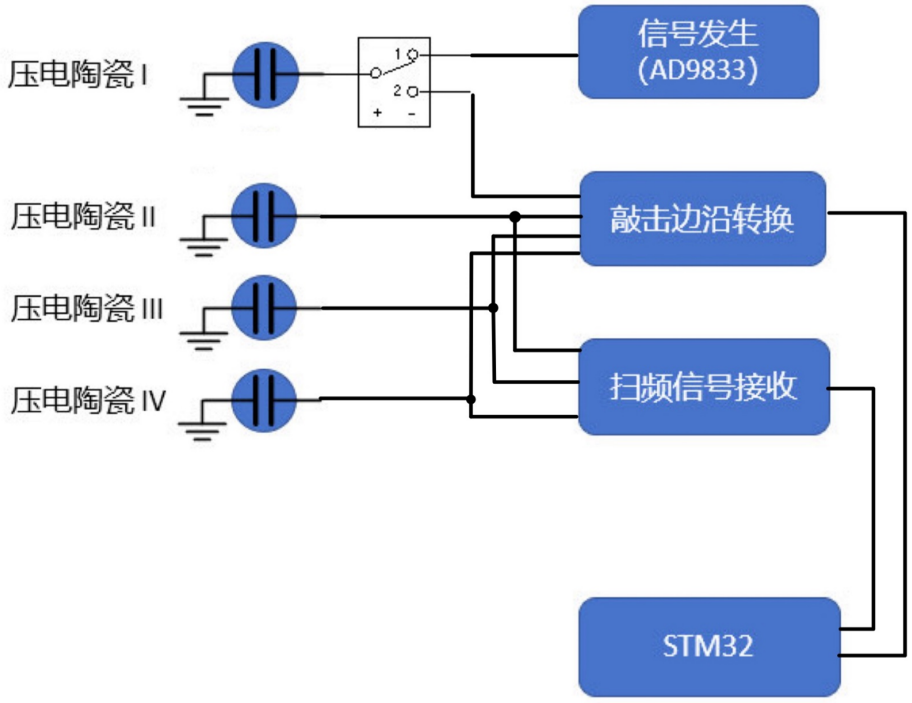

图 **2**　硬件总体框图

4.1　基本要求 1

我们采用模块 AD9833 进行扫频信号的输出，并且用 STM32F407 作为主控，控制电信号的输出。然后通过压电陶瓷片，将电信号转化为声信号，再通过亚克力板传播。

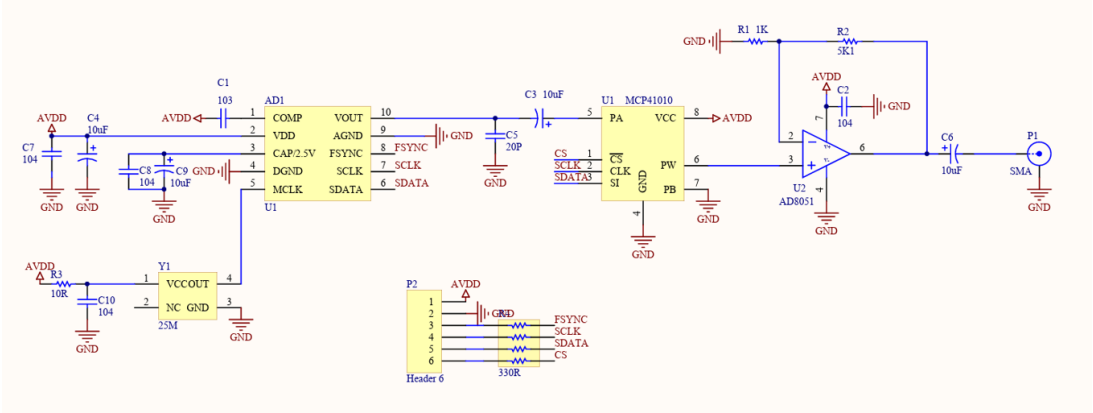

图 **3**　扫频信号发生（**AD9833**）

4.2　基本要求 2

我们采用敲击边沿转换的方式对信号时间差进行测量。首先对输入的信号进行全波整流，保证信号处于正半区，保护 GPIO，避免输入负电压。然后，对信号进行过零比较，变换为脉冲信号，便于对上升沿进行中断触发。

接下来，为了避免信号持续时间过短，信号难以采集，我们加上电容，起到峰值检测作用。然后再加上一级比较器，使其信号的电压放大到可以正常输入 GPIO。

然后，在 STM32 中设置上升沿触发，然后通过中断函数，记录中断时的时间，求得时间差，即可通过算法解得声源。

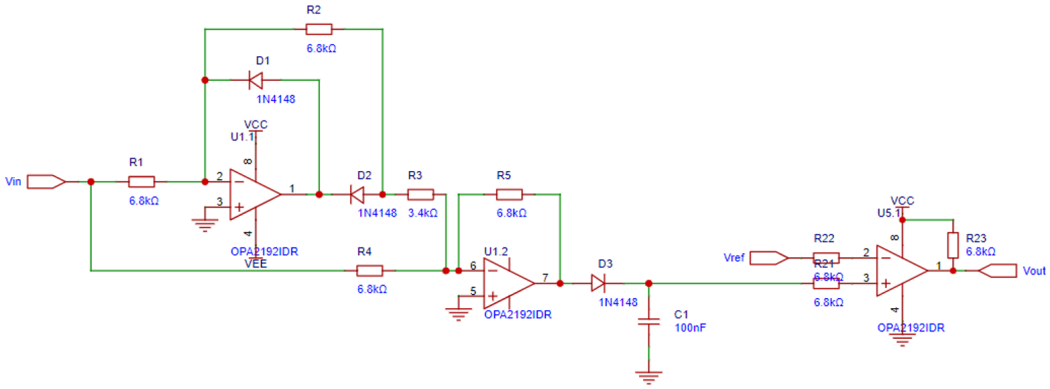

图 **4**　信号处理电路

5 软件设计

5.1 信号采集算法

采用 STM32 的 Timer 的中断捕获功能。将中断设置为上升沿触发，从而使得经过硬件电路处理过的脉冲信号能够触发中断。然后，编写中断函数。

中断函数中，首先读取中断标志位，判断是哪一路信号触发了中断，判断成功后，读取 CCR 的值，通过所得值乘以时钟周期，即可得到计数时间。由于考虑到精度较高，所以时钟采用 $1MHZ$，故非常容易计满溢出。所以，更新中断也要打开，中断函数中进行判断，如果是更新中断，则采用变量 cnt 计数，并且最终计算时间时，应该加上变量 cnt 的值乘以计满周期。

```
// TIM4
#include "Serial.h"
#include "TODA.h"
#include "math.h"

#define ARR_NUM 10000
#define PSC_NUM 84
#define SCALE 1240
#include "stm32f4xx.h"

int triggerOrder = 0;
int nowTime = 0;
int triggerTime[4];
int triggerTimeCCR[4];
int t[3];

void TIM_Init(void)
{
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

   TIM_InternalClockConfig(TIM4); // choose internal clock

   TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;
   TIM_TimeBaseInitStruct.TIM_ClockDivision = TIM_CKD_DIV1;
   TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up;
   TIM_TimeBaseInitStruct.TIM_Period = ARR_NUM - 1; // ARR
   TIM_TimeBaseInitStruct.TIM_Prescaler = PSC_NUM - 1; // PSC
   TIM_TimeBaseInitStruct.TIM_RepetitionCounter = 0; // Advanced TIM
   TIM_TimeBaseInit(TIM4, &TIM_TimeBaseInitStruct);

   // IC1 PB6
   TIM_ICInitTypeDef TIM_ICInitStruct;
   TIM_ICInitStruct.TIM_Channel = TIM_Channel_1;
   TIM_ICInitStruct.TIM_ICFilter = 0xf;
```

```
    TIM_ICInitStruct.TIM_ICPolarity = TIM_ICPolarity_Rising;
    TIM_ICInitStruct.TIM_ICPrescaler = TIM_ICPSC_DIV1;
    TIM_ICInitStruct.TIM_ICSelection = TIM_ICSelection_DirectTI;
    TIM_ICInit(TIM4, &TIM_ICInitStruct);

    // IC2 PB7
    TIM_ICInitStruct.TIM_Channel = TIM_Channel_2;
    TIM_ICInit(TIM4, &TIM_ICInitStruct);

    // IC3 PB8
    TIM_ICInitStruct.TIM_Channel = TIM_Channel_3;
    TIM_ICInit(TIM4, &TIM_ICInitStruct);

    // IC4 PB9
    TIM_ICInitStruct.TIM_Channel = TIM_Channel_4;
    TIM_ICInit(TIM4, &TIM_ICInitStruct);

    TIM_ITConfig(TIM4, TIM_IT_Update, ENABLE);
    TIM_ITConfig(TIM4, TIM_IT_CC1, ENABLE);
    TIM_ITConfig(TIM4, TIM_IT_CC2, ENABLE);
    TIM_ITConfig(TIM4, TIM_IT_CC3, ENABLE);
    TIM_ITConfig(TIM4, TIM_IT_CC4, ENABLE);
    TIM_Cmd(TIM4, ENABLE);

    NVIC_InitTypeDef NVIC_InitStruct;
    NVIC_InitStruct.NVIC_IRQChannel = TIM4_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);
}

int diffTime = 0;
int area;

int s1,s2;
int x_AB[2];
int y_12[2];
int cnt = 0;
void TIM4_IRQHandler(void)
{
    if (TIM_GetFlagStatus(TIM4, TIM_IT_Update) == SET)
    {
        nowTime++;
        TIM_ClearITPendingBit(TIM4, TIM_IT_Update);
        nowTime %= 100000000;
        if(cnt>=20)
```

```
        {
            cnt = 0;
            triggerOrder = 0;
            //Serial_Printf(" Clear! ");
        }
        if(triggerOrder!=0)
        {
            cnt++;
        }
    }
    if (TIM_GetFlagStatus(TIM4, TIM_IT_CC1) == SET)
    {

        TIM_ClearITPendingBit(TIM4, TIM_IT_CC1);
        triggerTime[0] = nowTime; // 单位: 10ms
        triggerTimeCCR[0] = TIM_GetCapture1(TIM4); // 以CCR1为基准 单位: 1us
        //Serial_Printf("CC1-%d0ms,%dus\n \n", triggerTime[0],triggerTimeCCR[0]);
        triggerOrder++;

    }
    if (TIM_GetFlagStatus(TIM4, TIM_IT_CC2) == SET)
    {

        TIM_ClearITPendingBit(TIM4, TIM_IT_CC2);
        triggerTime[1] = nowTime;
        triggerTimeCCR[1] = TIM_GetCapture2(TIM4); // 以CCR1为基准
        //Serial_Printf("CC2-%d0ms,%dus\n \n", triggerTime[1],triggerTimeCCR[1]);
        t[0] = triggerTime[1]*10+triggerTimeCCR[1];
        triggerOrder++;
        if(triggerOrder % 3 == 0)
        {
            nowTime = 0;
            triggerOrder = 0;
            TODA((t[1]-t[0])*1.0/SCALE,(t[2]-t[0])*1.0/SCALE, x_AB, y_12);
            //Serial_Printf(" %d %d \n", t[1]-t[0],t[2]-t[0]);
            //Serial_Printf("(%c%c,%2d%2d )",x_AB[0],x_AB[1],y_12[0],y_12[1]);
        }
    }
    if (TIM_GetFlagStatus(TIM4, TIM_IT_CC3) == SET)
    {

        TIM_ClearITPendingBit(TIM4, TIM_IT_CC3);
        triggerTime[2] = nowTime;
        triggerTimeCCR[2] = TIM_GetCapture3(TIM4); // 以CCR1为基准
        //Serial_Printf("CC3-%d0ms,%dus\n \n", triggerTime[2],triggerTimeCCR[2]);
        t[1] = triggerTime[2]*10+triggerTimeCCR[2];
        triggerOrder++;
```

```
        if(triggerOrder % 3 == 0)
        {
          nowTime = 0;
          triggerOrder = 0;
          TODA((t[1]-t[0])*1.0/SCALE,(t[2]-t[0])*1.0/SCALE, x_AB, y_12);
          //Serial_Printf(" %d %d \n", t[1]-t[0],t[2]-t[0]);
          //Serial_Printf("(%c%c,%2d%2d )",x_AB[0],x_AB[1],y_12[0],y_12[1]);
        }
    }
    if (TIM_GetFlagStatus(TIM4, TIM_IT_CC4) == SET)
    {

      TIM_ClearITPendingBit(TIM4, TIM_IT_CC4);
      triggerTime[3] = nowTime;
      triggerTimeCCR[3] = TIM_GetCapture4(TIM4); // 以CCR1为基准
      //Serial_Printf("CC4-%d0ms,%dus\n \n", triggerTime[3],triggerTimeCCR[3]);
      t[2] = triggerTime[3]*10+triggerTimeCCR[3];
      triggerOrder++;
      if(triggerOrder % 3 == 0)
      {
        nowTime = 0;
        triggerOrder = 0;
        TODA((t[1]-t[0])*1.0/SCALE,(t[2]-t[0])*1.0/SCALE, x_AB, y_12);
        //Serial_Printf(" %d %d \n", t[1]-t[0],t[2]-t[0]);
        //Serial_Printf("(%c%c,%2d%2d )",x_AB[0],x_AB[1],y_12[0],y_12[1]);
      }
    }
}

/*
void PWMSetCCRPercent(uint16_t percent)
{
  percent = percent % 101;
  uint16_t compare = percent*ARR_NUM/100;
  TIM_SetCompare3(TIM3, compare);
}
*/

#define SQUARE_TIME 500
```

5.2 TDOA 定位算法

本电路中有三块压电陶瓷片接收信号，作为“信号站”。在接收到信号后，STM32给出时间差，然后采用 TDOA 定位算法计算出声源的位置。

具体代码如下。

并且，为了提高精度，我们充分利用四个接收信号的压电陶瓷产生的三组时间差，进行求均值处理，得到更加可信的结果。

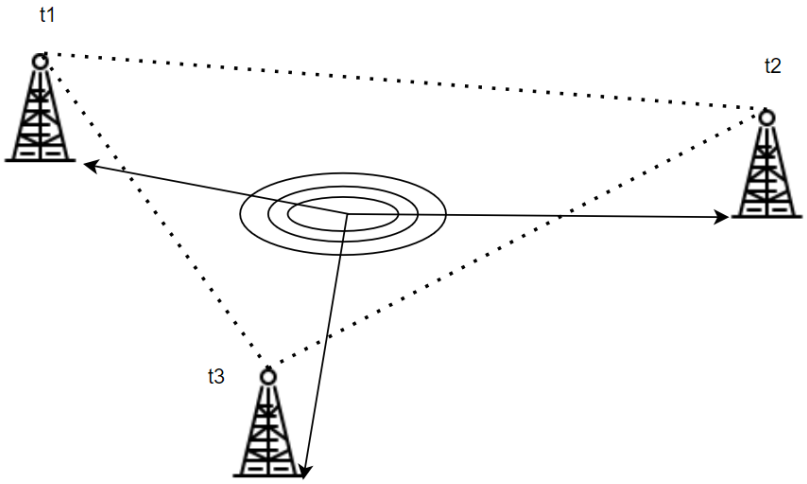

图 5　**TDOA**：信号接收与处理

```
#include<stdio.h>
#include<math.h>
#include "Serial.h"

#define x1 21
#define y1 21
#define x2 -21
#define y2 21
#define x3 -21
#define y3 -21

//float s1,s2;  //input
//float x_AB,y_12; //output
//int x_AB[2]; // AB~KL
//int y_12[2]; // 12~1112


void TODA(float s1,float s2, int x_AB[], int y_12[])
{
   float p1,q1,p1_m,p1_d,p2,q2,p2_m,p2_d;
   float k1,k2,k3;
   float a,b,c,d;
   float r1,r2;
   float xx,yy;
   k1 = pow(x1, 2) + pow(y1, 2);
   k2 = pow(x2, 2) + pow(y2, 2);
```

```
    k3 = pow(x3, 2) + pow(y3, 2);

    p1_m=(y2-y1)*s2*s2-(y3-y1)*s1*s1+(y3-y1)*(k2-k1)-(y2-y1)*(k3-k1);
    p1_d=(x2-x1)*(y3-y1)-(x3-x1)*(y2-y1);
    p1=p1_m/(p1_d*2);
    q1=((y2-y1)*s2-(y3-y1)*s1)/((x2-x1)*(y3-y1)-(x3-x1)*(y2-y1));

    p2_m=(x2-x1)*s2*s2-(x3-x1)*s1*s1+(x3-x1)*(k2-k1)-(x2-x1)*(k3-k1);
    p2_d=(x3-x1)*(y2-y1)-(x2-x1)*(y3-y1);
    p2=p2_m/(p2_d*2);
    q2=((x2-x1)*s2-(x3-x1)*s1)/((x3-x1)*(y2-y1)-(x2-x1)*(y3-y1));

    a=q1*q1+q2*q2-1;
    b=-2*((x1-p1)*q1+(y1-p2)*q2);
    c=(x1-p1)*(x1-p1)+(y1-p2)*(y1-p2);

    d=b*b-4*a*c;
    r1=(-b+sqrt(d))/(2*a);
    r2=(-b-sqrt(d))/(2*a);

    if(r1>0)
    {
       xx=p1+q1*r1;
       yy=p2+q2*r1;
    }
    else
    {
       xx=p1+q1*r2;
       yy=p2+q2*r2;
    }
    xx*=100;
    yy*=100;

    //映射
    int x;//计数
    int y;
    // 初值设定
    float x_low = -15;
    float x_high = -10;
    float y_low = 10;
    float y_high = 15;
    x_AB[0] = 'A';
    x_AB[1] = 'B';
    y_12[0] = 1;
    y_12[1] = 2;
    // 边界区域
    float x_min = -15;
```

```
    float x_max = 15;
    float y_min = -15;
    float y_max = 15;
    // 定位x
    for (x = 0; x < 6; x++) {
      if (xx < x_min || xx >= x_max) {
        x = -1;
        break;
      }
      else if (xx < x_high && xx >= x_low)
      break;
      else {
        x_low += 5;
        x_high += 5;
      }
    }
    // 定位y
    for (y = 0; y < 6; y++) {
      if (yy < y_min || yy >= y_max) {
        y = -1;
        break;
      }
      else if (yy < y_high && yy >= y_low)
      break;
      else {
        y_low -= 5;
        y_high -= 5;
      }
    }
    // 转换板子位置
    if (x == -1 || y == -1) {//超出坐标范围，则为0
      x_AB[0] = '*';
      x_AB[1] = '*';
      y_12[0] = '*';
      y_12[1] = '*';
    } else {
      x_AB[0] += 2 * x;
      x_AB[1] += 2 * x;
      y_12[0] += 2 * y;
      y_12[1] += 2 * y;
    }
    //Serial_Printf("---- %f %f ----",xx,yy);
    Serial_Printf("va0.val=%d",((x_AB[0]-'A'+2)/2)*30);
    Serial_End();
    Serial_Printf("va1.val=%d",((y_12[0]+1)/2)*30);
    Serial_End();
    Serial_Printf("va2.val=1");
```

```
    Serial_End();
}
```

5.3 坐标映射算法

6 系统测试

6.1 测试仪器及测试方法

示波器

6.2 测试过程及结果分析

6.2.1 硬件测试

在硬件电路搭建完毕后，首先将输出连接到示波器，将时基与幅度调至适当量程，之后将示波器设置为单次触发模式，上升沿触发，并且将触发电平调至 2V，然后敲击亚克力板，观察三路信号的输出是否正常，以及上升沿是否正常。

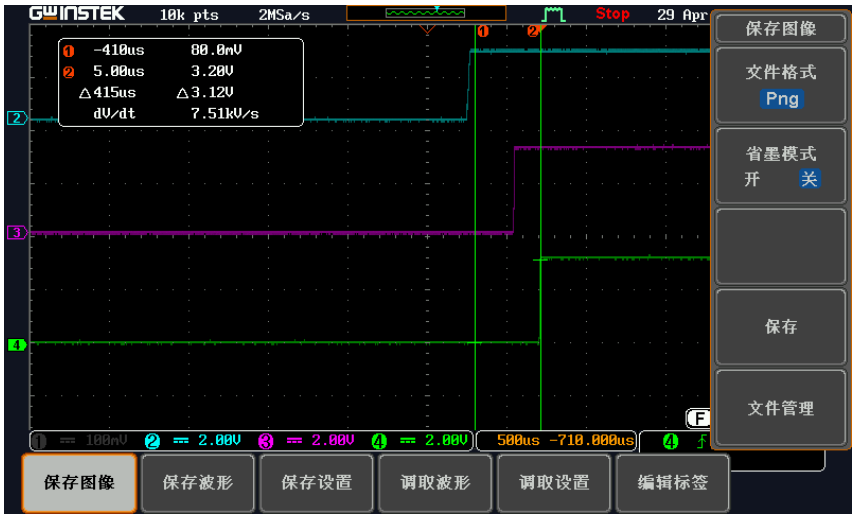

图 6　三路信号

当确定波形正常之后，再重复敲击同一位置，然后观察波形、时间差是否相同。然后，从串口输出时间差，观察输出的时间差，与从示波器上读到的值是否一致。

当一致性良好时，即可认为硬件基本良好。

6.2.2 软件测试

在软件测试中，由于需要信号在亚克力板中的传播速度，我们首先在紧邻一片压电陶瓷进行敲击，然后利用示波器测量另一块接收到信号的时间差，从而求得速度。并且

多次测量取平均，保证数据的可信度。

随后，我们

7 结论

8 参考文献与引用

参考文献

[1] Yamasaki, Ryota, et al. ”TDOA location system for IEEE 802.11 b WLAN.” IEEE Wireless Communications and Networking Conference, 2005. Vol. 4. IEEE, 2005.

附录 A　源程序代码

AD9833

```
#include "AD9833.h"

//***************************
//   Pin assign
//      STM32      AD9833
//   GPIOB_Pin_15    ---> FSYNC
//   GPIOB_Pin_14    ---> SCK/CLK
//   GPIOB_Pin_13    ---> DAT
//   GPIOB_Pin_12    ---> CS
//***************************

/*端口定义 */
#define PIN_GROUP GPIOB

#define PIN_FSYNC GPIO_Pin_15

#define PIN_SCK GPIO_Pin_14

#define PIN_DAT GPIO_Pin_13

#define PIN_CS GPIO_Pin_12 //数字电位器片选

#define RESET_FSYNC() GPIO_ResetBits(PIN_GROUP, PIN_FSYNC)
#define SET_FSYNC() GPIO_SetBits(PIN_GROUP, PIN_FSYNC)

#define RESET_SCK() GPIO_ResetBits(PIN_GROUP, PIN_SCK)
#define SET_SCK() GPIO_SetBits(PIN_GROUP, PIN_SCK)

#define RESET_DAT() GPIO_ResetBits(PIN_GROUP, PIN_DAT)
#define SET_DAT() GPIO_SetBits(PIN_GROUP, PIN_DAT)

#define RESET_CS() GPIO_ResetBits(PIN_GROUP, PIN_CS)
#define SET_CS() GPIO_SetBits(PIN_GROUP, PIN_CS)

void AD9833InitConfig(void) {
  // 开启GPIOB时钟
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB,ENABLE);

  // 定义GPIO结构体
  GPIO_InitTypeDef GPIO_InitStruct;
  GPIO_InitStruct.GPIO_Pin = PIN_FSYNC|PIN_SCK|PIN_DAT|PIN_CS;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
```

```
45          GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
46          GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
47
48          GPIO_Init(PIN_GROUP,&GPIO_InitStruct);
49      }
50
51      // 延迟
52      void AD9833_Delay() {
53          for(int i=0; i<1; i++);
54      }
55
56      /*
57      *************************************
58      * 函 数 名: AD9833_Write
59      * 功能说明: 向SPI总线发送16个bit数据
60      * 形    参: TxData : 数据
61      * 返 回 值: 无
62      *************************************
63      */
64      void AD9833_Write(unsigned int TxData)
65      {
66          unsigned char i;
67
68          SET_SCK();
69          //AD9833_Delay();
70          SET_FSYNC();
71          //AD9833_Delay();
72          RESET_FSYNC();
73          //AD9833_Delay();
74          for(i = 0; i < 16; i++)
75          {
76              if (TxData & 0x8000) // 获取最高位
77              SET_DAT(); // 数据为设为1
78              else
79              RESET_DAT(); // 数据为设为0
80
81              AD9833_Delay();
82              RESET_SCK();
83              AD9833_Delay();
84              SET_SCK();
85
86              TxData <<= 1;
87          }
88          SET_FSYNC();
89      }
90
91      /*
```

```
92      ************************************
93      * 函 数 名: AD9833_AmpSet
94      * 功能说明: 改变输出信号幅度值
95      * 形    参: 1.amp : 幅度值 0- 255
96      * 返 回 值: 无
97      ************************************
98      */
99      void AD9833_AmpSet(unsigned char amp)
100     {
101        unsigned char i;
102        unsigned int temp;
103
104        RESET_CS();
105        temp =0x1100|amp;
106        for(i=0;i<16;i++)
107        {
108           RESET_SCK();
109           if(temp&0x8000)
110           SET_DAT();
111           else
112           RESET_DAT();
113           temp<<=1;
114           SET_SCK();
115           AD9833_Delay();
116        }
117
118        SET_CS();
119     }
120
121     /*
122     ************************************
123     * 函 数 名: AD9833_WaveSeting
124     * 功能说明: 向SPI总线发送16个bit数据
125     * 形    参: 1.Freq: 频率值, 0.1 hz - 12Mhz 单位:HZ
126     2.Freq_SFR: 0 或 1
127     3.WaveMode: TRI_WAVE(三角波),SIN_WAVE(正弦波),SQU_WAVE(方波)
128     4.Phase : 波形的初相位
129     * 返 回 值: 无
130     ************************************
131     */
132     void AD9833_WaveSeting(double Freq,unsigned int Freq_SFR,unsigned int WaveMode,unsigned
            int Phase )
133     {
134        int frequence_LSB,frequence_MSB,Phs_data;
135        double  frequence_mid,frequence_DATA;
136        long int frequence_hex;
137
```

```
138         /***************计算频率的16进制值*****************/
139         frequence_mid=268435456/25;//适合25M晶振
140         //如果时钟频率不为25MHZ，修改该处的频率值，单位MHz ，AD9833最大支持25MHz
141         frequence_DATA=Freq;
142         frequence_DATA=frequence_DATA/1000000;
143         frequence_DATA=frequence_DATA*frequence_mid;
144         frequence_hex=frequence_DATA;
                //这个frequence_hex的值是32位的一个很大的数字，需要拆分成两个14位进行处理;
145         frequence_LSB=frequence_hex; //frequence_hex低16位送给frequence_LSB
146         frequence_LSB=frequence_LSB&0x3fff;//去除最高两位，16位数换去掉高位后变成了14位
147         frequence_MSB=frequence_hex>>14; //frequence_hex高16位送给frequence_HSB
148         frequence_MSB=frequence_MSB&0x3fff;//去除最高两位，16位数换去掉高位后变成了14位
149
150         Phs_data=Phase|0xC000; //相位值
151         AD9833_Write(0x0100); //复位AD9833,即RESET位为1
152         AD9833_Write(0x2100); //选择数据一次写入，B28位和RESET位为1
153
154         if(Freq_SFR==0)          //把数据设置到设置频率寄存器0
155         {
156            frequence_LSB=frequence_LSB|0x4000;
157            frequence_MSB=frequence_MSB|0x4000;
158            //使用频率寄存器0输出波形
159            AD9833_Write(frequence_LSB); //L14，选择频率寄存器0的低14位数据输入
160            AD9833_Write(frequence_MSB); //H14 频率寄存器的高14位数据输入
161            AD9833_Write(Phs_data); //设置相位
162            //AD9833_Write(0x2000); /**设置FSELECT位为0，芯片进入工作状态,频率寄存器0输出波形**/
163         }
164         if(Freq_SFR==1)         //把数据设置到设置频率寄存器1
165         {
166            frequence_LSB=frequence_LSB|0x8000;
167            frequence_MSB=frequence_MSB|0x8000;
168            //使用频率寄存器1输出波形
169            AD9833_Write(frequence_LSB); //L14，选择频率寄存器1的低14位输入
170            AD9833_Write(frequence_MSB); //H14 频率寄存器1为
171            AD9833_Write(Phs_data); //设置相位
172            //AD9833_Write(0x2800);
173            /**
174            设置FSELECT位为0，设置FSELECT位为1，即使用频率寄存器1的值，芯片进入工作状态，
175            频率寄存器1输出波形
176            **/
177         }
178         if(WaveMode==TRI_WAVE) //输出三角波波形
179         AD9833_Write(0x2002);
180         if(WaveMode==SQU_WAVE) //输出方波波形
181         AD9833_Write(0x2028);
182         if(WaveMode==SIN_WAVE) //输出正弦波形
183         AD9833_Write(0x2000);
```

```
184     }
```

AD9833