

**单片机及嵌入式系统综合实习** **课程设计**

**设计(论文)题目基于STM32F0系列芯片的声 音采集系统**

**学院名称 核技术与自动化工程学院**

**专业名称 测控技术与仪器**

**学生姓名 追梦少年南南**

**学生学号 202006010336**

**任课教师 周伟 王敏**

**设计（论文）成绩**

**教务处 制**

2022年12月05日

填写说明

1. 专业名称填写为专业全称，有专业方向的用小括号标明；
2. 格式要求：格式要求：
3. 用A4纸双面打印（封面双面打印）或在A4大小纸上用蓝黑色水笔书写。
4. 打印排版：正文用宋体小四号，1.5倍行距，页边距采取默认形式（上下2.54cm，左右2.54cm，页眉1.5cm，页脚1.75cm）。字符间距为默认值（缩放100%，间距：标准）；页码用小五号字底端居中。
5. 具体要求：

**题目**（二号黑体居中）；

**摘要**(“摘要”二字用小二号黑体居中，隔行书写摘要的文字部分，小4号宋体)；

**关键词**（隔行顶格书写“关键词”三字，提炼3-5个关键词，用分号隔开，小4号黑体)；

正文部分采用三级标题；

**第1章** ××(小二号黑体居中，段前0.5行)

**1.1** ×××××小三号黑体×××××（段前、段后0.5行）

**1.1.1**小四号黑体（段前、段后0.5行）

**参考文献**（黑体小二号居中，段前0.5行），参考文献用五号宋体，参照《参考文献著录规则（GB/T 7714－2005）》。

摘 要

本文详细描述了用咪头进行声音采集与波形展示的全过程，从绘制原理图、制作PCB到使用代码实现相关功能、进行仿真与波形展示。本文就声音采集部分进行详细描述，并且深入分析与讨论电路原理，深入解析代码逻辑结构，最终呈现完美的作品。

**关键词：STM32Cube；声音采集；ADC**

目录

[第一章 引言 6](#_Toc121693443)

[1.1 单片机技术介绍 6](#_Toc121693444)

[1.2实习内容及要求 6](#_Toc121693445)

[第二章 系统硬件电路设计 7](#_Toc121693446)

[2.1 原理图电路设计 7](#_Toc121693447)

[2.1.1 MCU——STM32F030C8T6 7](#_Toc121693448)

[2.1.2 稳压电路——AMS1117 7](#_Toc121693449)

[2.1.3串口通信电路——CH340C 8](#_Toc121693450)

[2.1.4 程序下载电路——Serial Wire 9](#_Toc121693451)

[2.1.5 复位电路 9](#_Toc121693452)

[2.1.6 整体电路 10](#_Toc121693453)

[2.2 PCB走线设计 10](#_Toc121693454)

[2.2.1 走线 10](#_Toc121693455)

[2.2.2 铺铜 10](#_Toc121693456)

[2.2.3 其他 11](#_Toc121693457)

[第三章 系统软件设计 12](#_Toc121693458)

[3.1 系统软件整体结构 12](#_Toc121693459)

[3.1.1 STM32CubeIDE简介 12](#_Toc121693460)

[3.1.2 STM32CubeMonitor简介 12](#_Toc121693461)

[3.2 CubeMX初始化配置部分 13](#_Toc121693462)

[3.2.1 RCC时钟和SYS部分 13](#_Toc121693463)

[3.2.2 ADC采集部分 14](#_Toc121693464)

[3.2.3 USART串口部分 15](#_Toc121693465)

[3.3 各个模块以及主函数部分代码 15](#_Toc121693466)

[3.3.1 ADC初始化adc.c 15](#_Toc121693467)

[3.3.2 串口初始化uart.c 17](#_Toc121693468)

[3.3.3 主函数main.c 17](#_Toc121693469)

[第四章 系统焊接及调试 20](#_Toc121693470)

[4.1 实物焊接器件 20](#_Toc121693471)

[4.2 系统调试及结果 20](#_Toc121693472)

第一章 引言

1.1 单片机技术介绍

单片机又称单片微控制器，它不是完成某一个逻辑功能的芯片，而是把一个计算机系统集成到一个芯片上。相当于一个微型的计算机，和计算机相比，单片机只缺少了I/O设备。概括的讲：一块芯片就成了一台计算机。它的体积小、质量轻、价格便宜、为学习、应用和开发提供了便利条件。而STM32系列MCU（以下简称STM32）是国内应用非常广泛的一种32位MCU，它拥有种种资源以便开发者进行开发，另外其公司意法半导体（以下简称ST）对STM32的开发者做出了极大的贡献，ST的生态是非常完美的，这令许多开发者在进行开发时更快上手，少走弯路。

1.2实习内容及要求

本次实验要求学生了解并掌握STM32的开发流程，从硬件开始制作一款属于自己的单片机开发板，并完成对声音信号采集的功能。

第二章 系统硬件电路设计

2.1 原理图电路设计

2.1.1 MCU——STM32F030C8T6

在本次实验中，我采用的是STM32F030C8T6单片机（图2.1），STM32F103C8T6是一款基于ARM Cortex-M 内核[STM32](https://baike.baidu.com/item/STM32/9133302)系列的32位的[微控制器](https://baike.baidu.com/item/%E5%BE%AE%E6%8E%A7%E5%88%B6%E5%99%A8/6688343)，[程序存储器](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F%E5%AD%98%E5%82%A8%E5%99%A8/7661398)容量是64KB，需要电压2V—3.6V，工作温度为-40°C—85°C。它主频最大为48MHz，内部有一个具有9个ADC通道的ADC单元，具有两个通用同步异步收发器（USART）等等。

在进行原理图设计的时候，需要额外注意单片机的供电部分、晶振部分以及复位电路部分等等。

图示, 示意图

描述已自动生成

图2.1 MCU主体

2.1.2 稳压电路——AMS1117

由于STM32芯片需要使用3.3V电压为其供电，而在电脑USB端口的输出电压为5V，所以我们需要用到AMS1117芯片（图2.2）。这是一个可以将5V电压转换为3.3V电压的稳压芯片，在STM32上是必不可少的东西。那么根据AMS1117的特性，我设计的如下电路（图2.3），它可以很好的将5V的供电电压转换为3.3V供单片机工作。

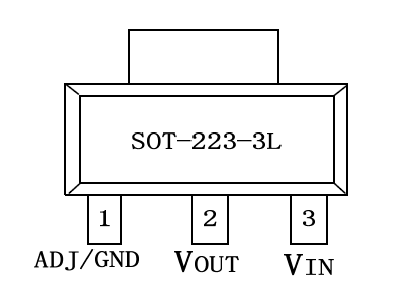


图2.2 AMS1117引脚分配

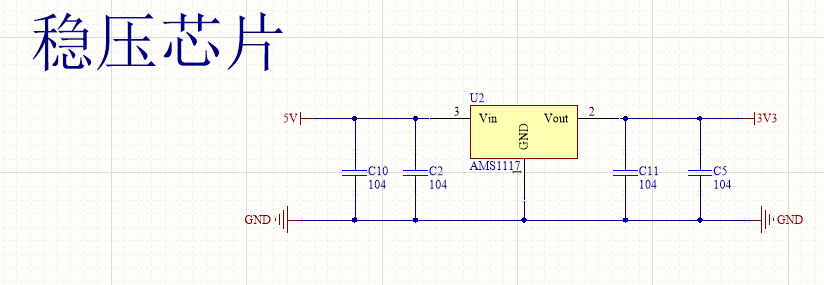


图2.3 自己设计的AMS1117电路部分

2.1.3串口通信电路——CH340C

CH340C是一款USB转串口芯片，属于CH340系列芯片（图2.4）。CH340可以实现USB转串口或USB转打印口的功能，在串口方式下，CH340提供常用的MODEM联络信号，用于为计算机扩展异步串口，或者将普通的串口设备直接升级到USB总线。

而CH340C是CH340系列的其中一款芯片，除去供电引脚，我们使用到的只有四个引脚，分别是TXD、RXD、UD+、UD-。TXD和RXD是串行数据输入输出，分别连接单片机USART1的RXD和TXD。UD+和UD-是USB信号端，直接连到USB总线的D+、D-端，无需串联电阻。因此，结合着供电电路部分，我设计了如下电路（图2.5）。

示意图

低可信度描述已自动生成

图2.4 CH340C引脚分配

图示, 示意图

描述已自动生成

图2.5 自己设计的CH340C电路部分

2.1.4 程序下载电路——Serial Wire

Serial Wire简称SW，Arm串行调试接口使用了一个单一的双向数据连接。电路分为SWDIO以及SWCLK两条线。它的工作原理是：外部的调试器向DP发送一个请求，DP就是请求的目标，DP发送应答告知调试器可以传输数据。通过目标机应答允许，数据读或写得命令被执行。当通信占用或故障时请求不被执行。读命令执行时目标机向宿主机传输数据。写命令执行时宿主机向目标机传输数据。而此处Cortex-M0的DP指的就是SW的DP。下图是SW下载部分（图2.6）。

图示, 示意图

描述已自动生成

图2.6 SW的下载部分

2.1.5 复位电路

STM32F的NRST是异步复位脚。当NRST输入低电平的时候，MCU处于复位状态，重设所有的内部寄存器，及片内几十KB的SRAM。当NRST从低电平变高时，PC指针从0开始。STM32中的NRST有施密特功能，大概在输入电压低于1V的时候将芯片复位（图2.7）。

图示, 示意图

中度可信度描述已自动生成

图2.7 复位电路

2.1.6 整体电路

整体电路结构如图所示。

图片包含 图示

描述已自动生成

图2.8 整体原理图

2.2 PCB走线设计

2.2.1 走线

输入和输出端的导线应尽量避免相邻平行。在 PCB 布线时，相邻层的走线方向成正交结构，避免将不同的信号线在相邻层走成同一方向，以减少不必要的层间窜扰。当 PCB 布线受到结构限制（如某些背板）难以避免出现平行布线时，特别是在信号速率较高时，应考虑用地平面隔离各布线层，用地线隔离各信号线。

2.2.2 铺铜

PCB的地线较多，我们应当采取铺铜的操作。将原理图上所有的GND用一层铜连接起来，以达到减小信号对外干扰的目的。对于孤岛铜，我们往往采取打过孔的形式进行操作，简单而有效。

2.2.3 其他

丝印部分细节没有很特别，值得注意的一点是丝印要控制大小并且打在醒目的位置即可。另外，在设计全部完成之后，我还添加了泪滴以达到美观且平滑信号线阻抗的目的。最后的PCB图如下所示。

图片包含 游戏机, 电子, 电路

描述已自动生成图示

描述已自动生成

图2.9 PCB顶层与底层设计

示意图, 地图

描述已自动生成

图2.10 PCB的3D视角

第三章 系统软件设计

3.1 系统软件整体结构

3.1.1 STM32CubeIDE简介

STM32CubeIDE是ST官方提供的免费软件开发工具，也是STM32Cube生态系统的一员大将。它基于[Eclipse](https://so.csdn.net/so/search?q=Eclipse&spm=1001.2101.3001.7020)/CDT框架，GCC编译工具链和GDB调试工具，支持添加第三方功能插件。同时，STM32CubeIDE还集成了部分STM32CubeMX和STM32CubeProgrammer的功能，是一个“多合一”的STM32开发工具。用户只需要STM32CubeIDE这一个工具，就可以完成从芯片选型，项目配置，代码生成，到代码编辑，编译，调试和烧录的所有工作。此次实习中，我选择使用此软件进行编辑，编译，调试，烧录的工作。

图形用户界面, 文本, 应用程序

描述已自动生成

图3.1 STM32CubeIDE界面

3.1.2 STM32CubeMonitor简介

2020年2月，ST公司推出了一款全新的软件STM32CubeMonitor 1.0.0。这是一个实时监测变量并将其加以图形化显示的软件。STM32CubeMonitor是基于NODE-RED开发的，使用基于流（Flow）的图形化编程技术，可供用户通过图形化编程方法设计程序，无需编写任何代码。此次实习中，我选择使用此软件进行声音波形的查看。

图示

中度可信度描述已自动生成

图3.2 STM32CubeMonitor界面

3.2 CubeMX初始化配置部分

3.2.1 RCC时钟和SYS部分

我们选择使能外部高速时钟HSE，在时钟树上选择6倍的倍频器，刚好达到48MHz的最大主频。而在SYS部分，我们选择SW作为调试方式。

图形用户界面, 文本, 应用程序

描述已自动生成

图3.3 RCC时钟配置

图示

描述已自动生成

图3.4 时钟树配置

图形用户界面, 文本, 应用程序

描述已自动生成

图3.5 SYS配置

3.2.2 ADC采集部分

我们需要ADC通道的一个单元进行ADC采集即可，此处我选择使用PA0引脚作为ADC的一个采集单元，我们只需要将咪头模块的输出接到ADC的输入部分即可。我选择12位数据右对齐，采样方式使用循环采样。

图形用户界面

描述已自动生成

图3.6 ADC采集配置

3.2.3 USART串口部分

我们仅需要将采集到的电压信号用串口进行打印即可。在此我结合硬件电路选择USART1进行通信，波特率选择115200bit/s，数据长八位，一个停止位，无奇偶校验位。

图形用户界面, 应用程序

描述已自动生成

图3.7 串口配置

3.3 各个模块以及主函数部分代码

3.3.1 ADC初始化adc.c

1. #include "adc.h"

2. ADC\_HandleTypeDef hadc;

3. void MX\_ADC\_Init(void)

4. {

5. ADC\_ChannelConfTypeDef sConfig = {0};

6. hadc.Instance = ADC1;

7. hadc.Init.ClockPrescaler = ADC\_CLOCK\_ASYNC\_DIV1;

8. hadc.Init.Resolution = ADC\_RESOLUTION\_12B;

9. hadc.Init.DataAlign = ADC\_DATAALIGN\_RIGHT;

10. hadc.Init.ScanConvMode = ADC\_SCAN\_DIRECTION\_FORWARD;

11. hadc.Init.EOCSelection = ADC\_EOC\_SINGLE\_CONV;

12. hadc.Init.LowPowerAutoWait = DISABLE;

13. hadc.Init.LowPowerAutoPowerOff = DISABLE;

14. hadc.Init.ContinuousConvMode = ENABLE;

15. hadc.Init.DiscontinuousConvMode = DISABLE;

16. hadc.Init.ExternalTrigConv = ADC\_SOFTWARE\_START;

17. hadc.Init.ExternalTrigConvEdge = ADC\_EXTERNALTRIGCONVEDGE\_NONE;

18. hadc.Init.DMAContinuousRequests = DISABLE;

19. hadc.Init.Overrun = ADC\_OVR\_DATA\_PRESERVED;

20. if (HAL\_ADC\_Init(&hadc) != HAL\_OK)

21. {

22. Error\_Handler();

23. }

24. sConfig.Channel = ADC\_CHANNEL\_0;

25. sConfig.Rank = ADC\_RANK\_CHANNEL\_NUMBER;

26. sConfig.SamplingTime = ADC\_SAMPLETIME\_7CYCLES\_5;

27. if (HAL\_ADC\_ConfigChannel(&hadc, &sConfig) != HAL\_OK)

28. {

29. Error\_Handler();

30. }

31. }

32. void HAL\_ADC\_MspInit(ADC\_HandleTypeDef\* adcHandle)

33. {

34. GPIO\_InitTypeDef GPIO\_InitStruct = {0};

35. if(adcHandle->Instance==ADC1)

36. {

37. \_\_HAL\_RCC\_ADC1\_CLK\_ENABLE();

38. \_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

39. GPIO\_InitStruct.Pin = GPIO\_PIN\_0;

40. GPIO\_InitStruct.Mode = GPIO\_MODE\_ANALOG;

41. GPIO\_InitStruct.Pull = GPIO\_NOPULL;

42. HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

43. }

44. }

3.3.2 串口初始化uart.c

1. #include "usart.h"

2. UART\_HandleTypeDef huart1;

3. void MX\_USART1\_UART\_Init(void)

4. {

5. huart1.Instance = USART1;

6. huart1.Init.BaudRate = 115200;

7. huart1.Init.WordLength = UART\_WORDLENGTH\_8B;

8. huart1.Init.StopBits = UART\_STOPBITS\_1;

9. huart1.Init.Parity = UART\_PARITY\_NONE;

10. huart1.Init.Mode = UART\_MODE\_TX\_RX;

11. huart1.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

12. huart1.Init.OverSampling = UART\_OVERSAMPLING\_16;

13. huart1.Init.OneBitSampling = UART\_ONE\_BIT\_SAMPLE\_DISABLE;

14. huart1.AdvancedInit.AdvFeatureInit = UART\_ADVFEATURE\_NO\_INIT;

15. if (HAL\_UART\_Init(&huart1) != HAL\_OK)

16. {

17. Error\_Handler();

18. }

19. }

20. void HAL\_UART\_MspInit(UART\_HandleTypeDef\* uartHandle)

21. {

22.

23. GPIO\_InitTypeDef GPIO\_InitStruct = {0};

24. if(uartHandle->Instance==USART1)

25. {

26. \_\_HAL\_RCC\_USART1\_CLK\_ENABLE();

27. \_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

28. GPIO\_InitStruct.Pin = GPIO\_PIN\_9|GPIO\_PIN\_10;

29. GPIO\_InitStruct.Mode = GPIO\_MODE\_AF\_PP;

30. GPIO\_InitStruct.Pull = GPIO\_NOPULL;

31. GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_HIGH;

32. GPIO\_InitStruct.Alternate = GPIO\_AF1\_USART1;

33. HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

34. }

35. }

3.3.3 主函数main.c

1. #include "main.h"

2. #include "adc.h"

3. #include "usart.h"

4. #include "gpio.h"

5. #include "retarget.h"

6. void SystemClock\_Config(void);

7. uint32\_t val\_x=0;

8. uint32\_t Volt\_x=0;

9. int main(void)

10. {

11. HAL\_Init();

12. SystemClock\_Config();

13. MX\_GPIO\_Init();

14. MX\_ADC\_Init();

15. MX\_USART1\_UART\_Init();

16. RetargetInit(&huart1);//串口重定向函数，官方提供，直接使用即可

17. HAL\_ADC\_Start(&hadc);

18. while (1)

19. {

20. if(HAL\_ADC\_PollForConversion(&hadc,200)==HAL\_OK)//adc采样

21. {

22. val\_x=HAL\_ADC\_GetValue(&hadc);

23. Volt\_x=3300\*val\_x;

24. Volt\_x=Volt\_x>>12; //将得到的结果进行转换

25. }

26. printf("%d\r\n",Volt\_x);//通过串口打印出来

27. HAL\_Delay(10);

28. }

29. }

30. void SystemClock\_Config(void)

31. {

32. RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

33. RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

34. RCC\_PeriphCLKInitTypeDef PeriphClkInit = {0};

35. RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI14|RCC\_OSCILLATORTYPE\_HSE;

36. RCC\_OscInitStruct.HSEState = RCC\_HSE\_ON;

37. RCC\_OscInitStruct.HSI14State = RCC\_HSI14\_ON;

38. RCC\_OscInitStruct.HSI14CalibrationValue = 16;

39. RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

40. RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSE;

41. RCC\_OscInitStruct.PLL.PLLMUL = RCC\_PLL\_MUL6;

42. RCC\_OscInitStruct.PLL.PREDIV = RCC\_PREDIV\_DIV1;

43. if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

44. {

45. Error\_Handler();

46. }

47. RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

48. |RCC\_CLOCKTYPE\_PCLK1;

49. RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

50. RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

51. RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;

52. if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_1) != HAL\_OK)

53. {

54. Error\_Handler();

55. }

56. PeriphClkInit.PeriphClockSelection = RCC\_PERIPHCLK\_USART1;

57. PeriphClkInit.Usart1ClockSelection = RCC\_USART1CLKSOURCE\_PCLK1;

58. if (HAL\_RCCEx\_PeriphCLKConfig(&PeriphClkInit) != HAL\_OK)

59. {

60. Error\_Handler();

61. }

62. }

63.

第四章 系统焊接及调试

4.1 实物焊接器件

经过实物焊接，最终的板子如下图所示。

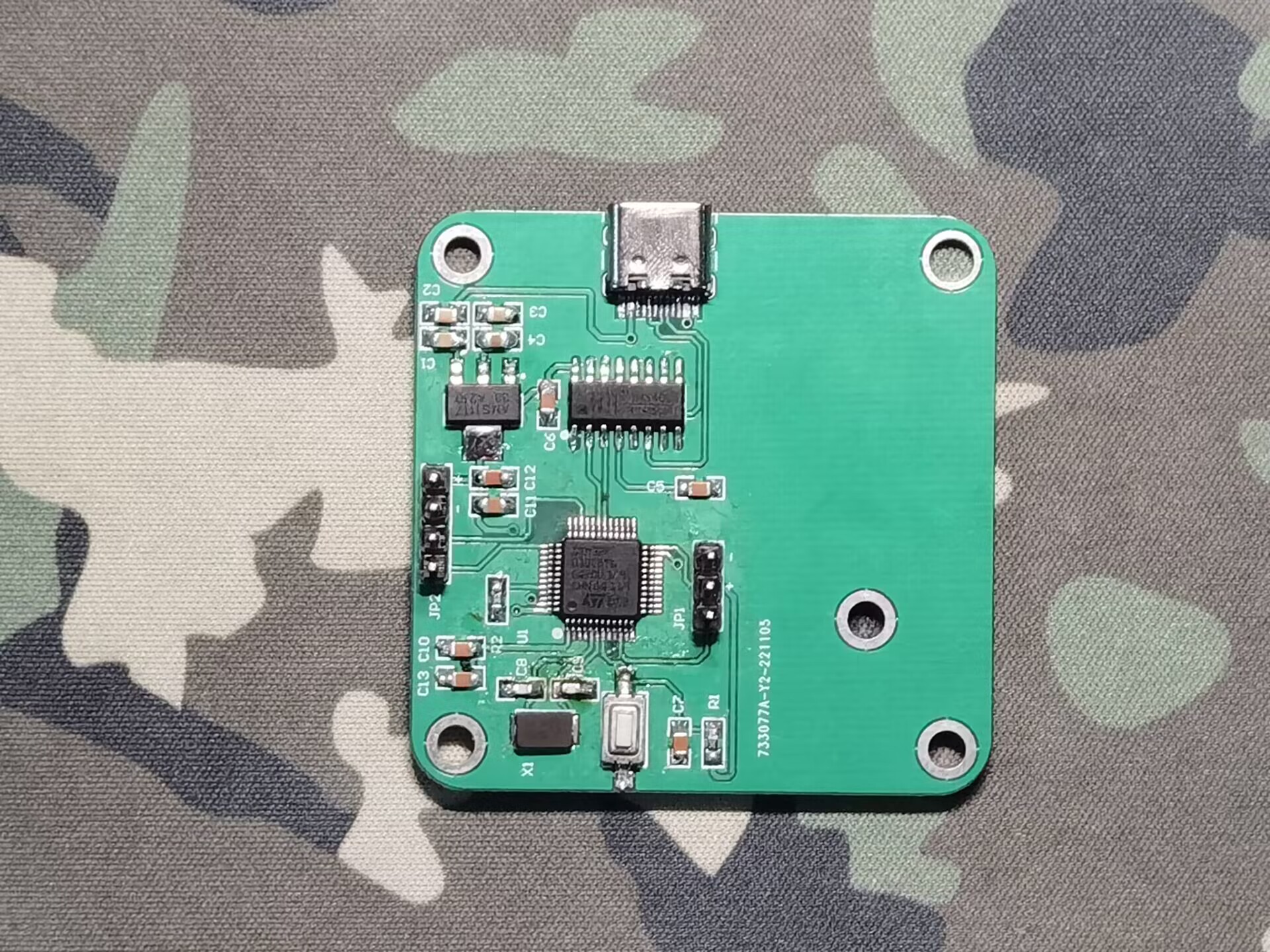


图4.1 PCB实物与焊接后

声音传感器如下图所示。

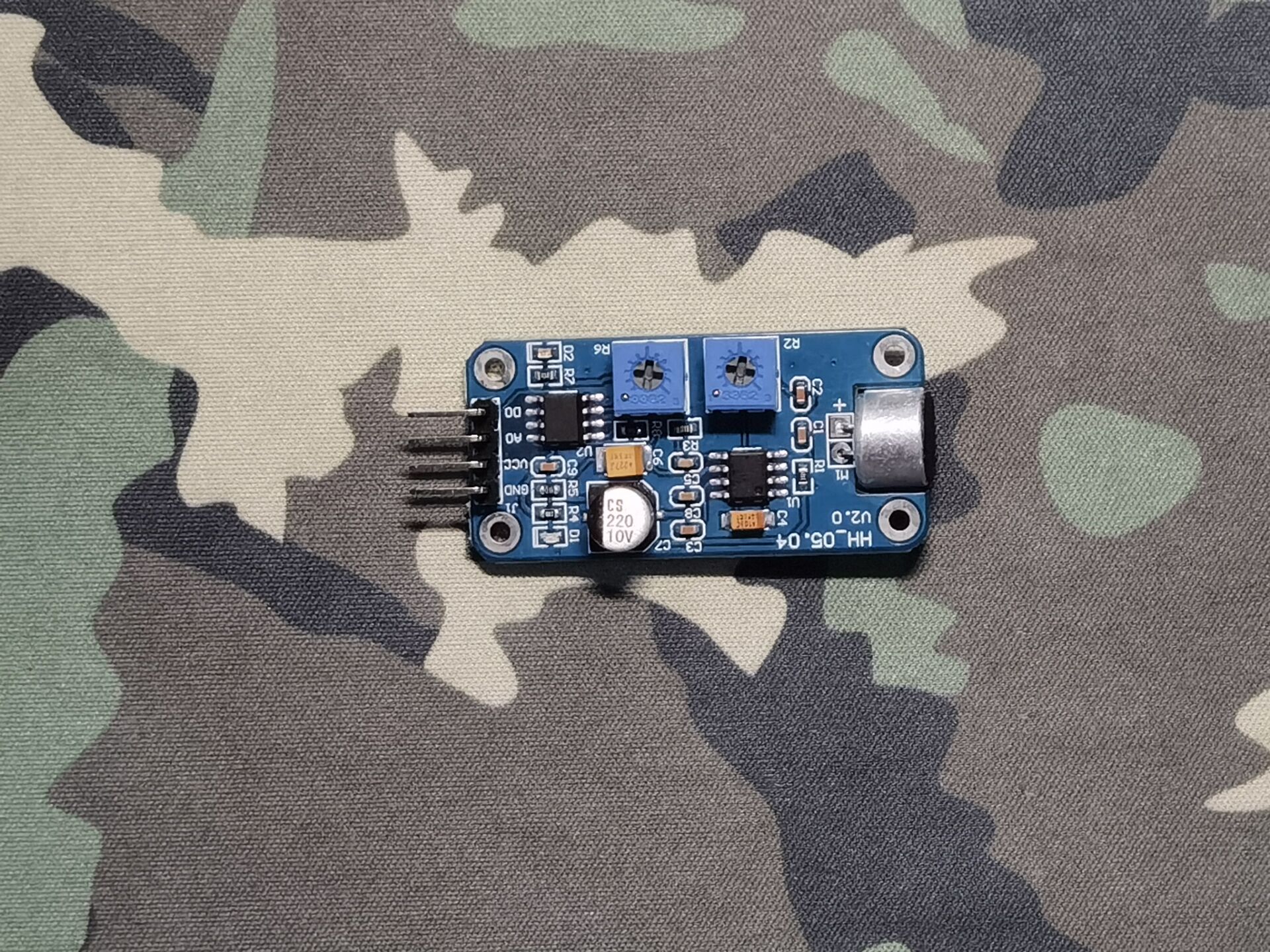


图4.2 声音传感器

4.2 系统调试及结果

经过不断的调试与测试，最终在STM32CubeMonitor上检测的电压变量随时间变化图如图4.3所示，我同样尝试了将串口打印出的数据在matlab上进行绘制，结果如图4.4所示。（注：两幅图的数据不是同一组数据）

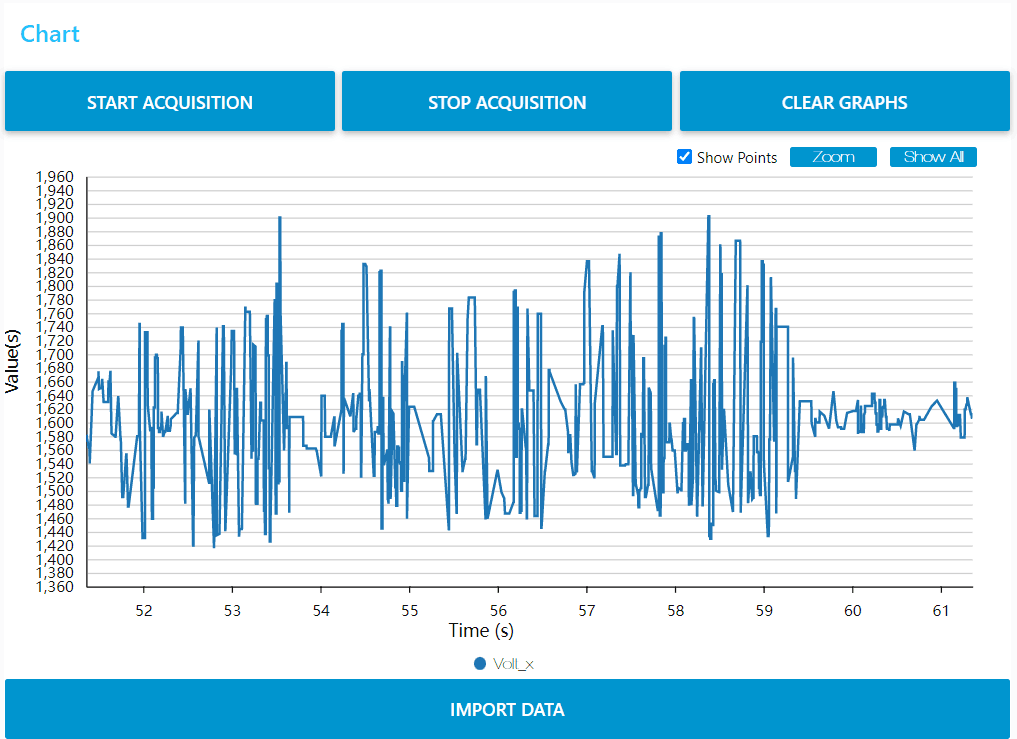


图4.3 电压变量监测

图表, 条形图

描述已自动生成

图4.4 串口打印后用matlab打印的波形图

|  |  |
| --- | --- |
| **学生学习心得** | 在本次实习中，我清楚明白的掌握了一个项目从开始到实现的整个流程与步骤，从原理图到PCB到实物器件焊接，从代码到原理的实现流程。在整个实验过程中，我全程以高昂的兴趣参与进来，完成实验后，我还将实验中的每一步加以自己的理解。我的梦想是做一名优秀的嵌入式工程师，这门课程让我离我的梦想又靠近了一步。  学生（签名）：  2022年12月11日 |
| **诚信承诺** | 本人郑重声明所呈交的课程报告是本人在指导教师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注的地方外，论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同学对本文研究所做的贡献均已在报告中作了明确的说明并表示谢意。  学生（签 名）： |
| **任课**  **教师**  **评语** | 成绩评定：  任课教师（签名）：  年 月 日 |