电子系统导论实验报告

实验 5 模数转换

指导教师： 万景

学生姓名： 彭堃 学生姓名： 吴磊 学生姓名： 徐洋

学 号： 22307110109 学 号： 22307130218 学 号： 20300290037

专 业： 保密技术 专 业： 技科 专 业： 计算机

日 期： 2024.03.28

0.1 实验目的：

· 了解模数转换的基本概念
· 初步了解信号发生器、示波器的使用
· 以 ADC0832 为例，掌握 ADC 的基本使用方法

0.2 实验原理：

0.2.1 什么是模数转换

1. 模数是指模拟量与数字量
2. 模数转换即是模拟量和数字量之间的转换
3. 目的是实现模拟电信号与数字电信号之间的转换，以进一步实现物理量和数字电信号的转换，达到联系计算机和生产或实验现场的效果

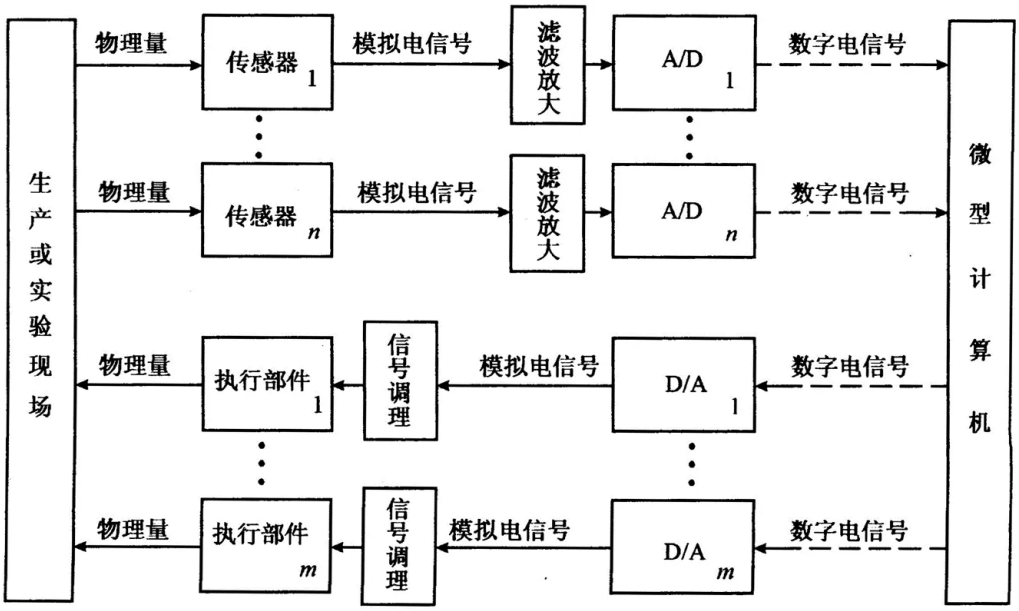


图 1: 模数转换原理示意图

0.2.2 如何实现模数转换

1. 模数转换器 ADC

· 能够将模拟电压量转换成离散数值，通常用二进制或 16 进制表示 $D = Vin/\Delta$

· 实现方式有很多，比如逐次逼近型

· 操作过程如下图所示

- 输入连续信号
- 依据 cpu 的时钟周期进行等距采样——时间离散化
- 通过特定的方式对不同大小的数据进行编码（会丢失细节）——幅度离散化
- 输出数字信号

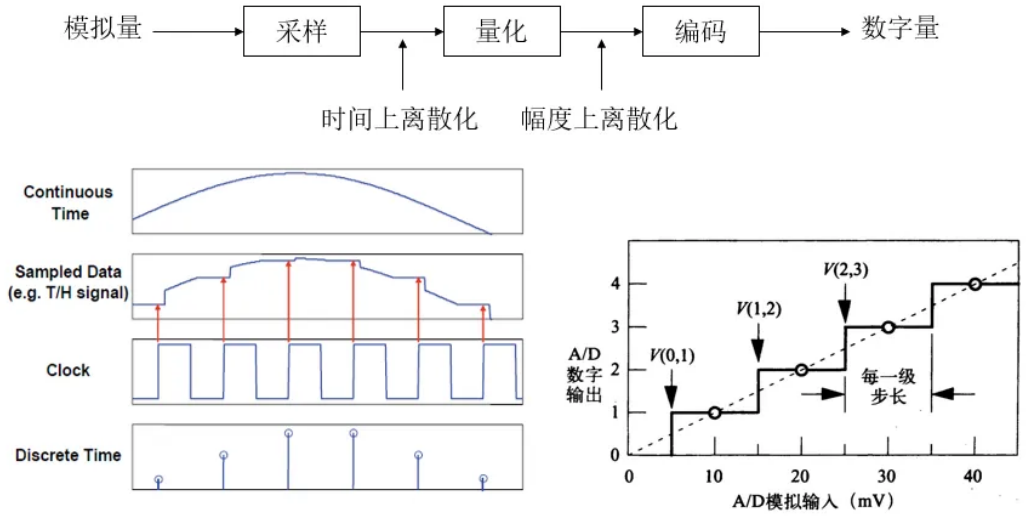


图 2: 模数转换操作示意图

2.ADC0832（逐次逼近型）简介

ADC0832 是美国国家半导体公司生产的一种 8 位分辨率、双通道 A/D 转换芯片。其工作原理为逐次逼近型。

编码原理如下，逻辑很简单

· 首先以 32 为起始比较，并标记最后一个被加的数字为“头”

  ◦ 如果比较结果是 1，就再加上头的一半，此时头转移到新加的数字

  ◦ 如果比较结果是 0，就将头减半，头的位置不变

  ◦ 再次比较

· 随后不断运行上述逻辑进行比较和编码，直到头指向 1 为止

· 不难发现这样产生的编码总是一个 6 位的二进制数字，因为头总是从 32 开始一直被被除以 2

· 其编码范围为 0—65，精度为二进制 1，同时其向下取整，即若右边实际值为 1.5，则通过模数转换后得到的数字信号为 1

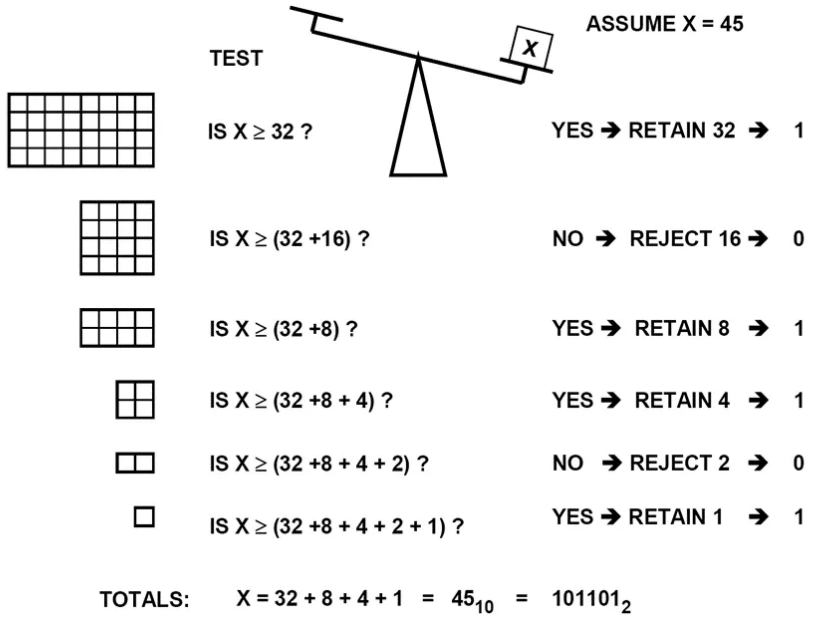


图 3: Enter Caption

特点：· 输入输出电平与 TTL/CMOS 相兼容；

·5V 电源供电时输入电压在 0 5V 之间；

· **该 ADC 的满量程参考电压直接设置为电源电压；**

· 工作频率为 250KHz，转换时间为 32 S；一般功耗为 15mW；

·8P、14P—DIP（双列直插）、PICC 多种封装；

· 商用级芯片温宽为 0℃ to +70℃ ,工业级芯片温宽为 −40℃ to +125℃；

Note：本实验中用 GPIO3.3V 供电，输入电压在 0~3.3V 之间；

3. 关于芯片接口

·CS 片选使能，低电平芯片使能。

·CH0 模拟输入通道 0，或作为 IN+/-使用。

·CH1 模拟输入通道 1，或作为 IN+/-使用。

·GND 芯片参考 0 电位（地）。

·DI 数据信号输入，选择通道控制。

·DO 数据信号输出，转换数据输出。

·CLK 芯片时钟输入。

·Vcc/REF 电源输入及参考电压输入（复用）。

4. 应用接口、工作时序和工作原理

· 正常情况下 ADC0832 与单片机的接口应为 4 条数据线，分别是-CS、CLK、DO、DI。

· 但由于 D0 端与 DI 端在通信时输入输出信号不会同时产生，与树莓派的接口之间的通信是双向的，所以电路设计时可以将 D0 和 DI 并联后接入同一根数据线上使用。

· 当-CS 输入端高电平时芯片禁用。当要进行 A/D 转换时，须先将-CS 置于低电平并且保持低电平直到转换完全结束。

· **在第 1 个时钟脉冲的下降沿之前 Dl 端必须是高电平，表示启始信号。在第 2、3 个脉冲下降沿之前 DI 端应输入 2 位数据用于选择模拟信号通道**。

- 当这两位数据为 “1”、“0” 时，设置 CH0 为单端输入；
- 当 2 位数据为 “1”、“1” 时，设置 CH1 为单端输入；
- 当 2 位数据为 “0”、“0” 时，将 CH0 作为正输入端 IN+，CH1 作为负输入端 IN-进行差分输入；
- 当 2 位数据为 “0”、“1” 时，将 CH0 作为负输入端 IN-，CH1 作为正输入端 IN+ 进行差分输入。

· 到第 3 个脉冲的下降沿之后输出 D0 进行转换数据的读取。从第 4 个脉冲下降沿开始由 DO 端输出转换数据最高位 DATA7，随后在每一个脉冲的下降沿 D0 端输出下一位数据，直到第 11 个脉冲时发出最低位数据 DATA0。· 紧接着，从此位开始输出一个相反顺序的数据，即从第 11 个字节的下降沿输出 DATD0，随后输出 8 位数据，到第 19 个脉冲时数据输出完成。

0.2.3 代码分析

```
# 库引入
#
    RPi.GPIO库用于树莓派GPIO控制，wiringpi库也是用于GPIO控制，time库用于时间相关功能
import RPi.GPIO as GPIO
import wiringpi
import time

# 定义ADC模块的引脚
ADC_CS = 17 #11
ADC_CLK = 18 #12
ADC_DIO = 27 #13
usdelay = 2 # 最大时钟频率为400KHz，这里设置为2*2us
T_convert = 8*2*usdelay # ADC转换时间

# 使用默认引脚以保持向后兼容性
def setup(cs=17,clk=18,dio=27): #11,12,13
    global ADC_CS, ADC_CLK, ADC_DIO
    ADC_CS=cs
    ADC_CLK=clk
    ADC_DIO=dio
    GPIO.setwarnings(False)
```

```
    GPIO.setmode(GPIO.BCM)      # 以BCM编号引脚
    GPIO.setup(ADC_CS, GPIO.OUT)   # 将引脚设为输出
    GPIO.setup(ADC_CLK, GPIO.OUT)  # 将引脚设为输出

def destroy(): # 清理GPIO设置
    GPIO.cleanup()

# 使用通道0作为默认通道以保持向后兼容性
def getResult(channel=0):              # 获取ADC结果，选择输入通道
    GPIO.setup(ADC_DIO, GPIO.OUT) # 将DIO引脚设为输出

    GPIO.output(ADC_CS, 0) # 使能片选信号，低电平表明开始工作

# 下面便是根据ADC0832的工作协议进行通道选择
    # 输入第一个时钟脉冲下降沿，并向DIO输出高电平，表示启始信号
    GPIO.output(ADC_CLK, 0)
    GPIO.output(ADC_DIO, 1); wiringpi.delayMicroseconds(usdelay)
    # 重新拉高时钟脉冲
    GPIO.output(ADC_CLK, 1); wiringpi.delayMicroseconds(usdelay)

    # 在第2、3个时钟脉冲下降沿向DIO输出一个高电平和一个channel
    #
        依照协议，当channel为0时，便是CH0单端输入，channel为1时，便是CH1单端输入
    GPIO.output(ADC_CLK, 0)
    GPIO.output(ADC_DIO, 1); wiringpi.delayMicroseconds(usdelay)

    GPIO.output(ADC_CLK, 1); wiringpi.delayMicroseconds(usdelay)

    GPIO.output(ADC_CLK, 0)
    GPIO.output(ADC_DIO, channel);
        wiringpi.delayMicroseconds(usdelay)
```

```
    GPIO.output(ADC_CLK, 1); wiringpi.delayMicroseconds(usdelay)

# 第3个脉冲下降沿之后，输出DO进行转换数据的读取
#
    我们这里将DIO设置为输入，并开始等待设置的T_convert时间让ADC0832进行处理
    GPIO.output(ADC_CLK, 0)
    GPIO.setup(ADC_DIO, GPIO.IN);
        wiringpi.delayMicroseconds(T_convert)

# 根据工作协议，DIO将进行两次输出，我们将分别记录下来
#
    第一次是从高位DATA7开始输出，第二次是从低位DATA0开始输出，所以处理方式不同
# 这里设置dat1和dat2是为了保证数据正确
    dat1 = 0 # 初始化为0
    for i in range(0, 8): # 循环输入时钟脉冲并读取数据，进行数据处理
        GPIO.output(ADC_CLK, 1); wiringpi.delayMicroseconds(usdelay)
        GPIO.output(ADC_CLK, 0); wiringpi.delayMicroseconds(usdelay)
        dat1 = dat1 << 1 | GPIO.input(ADC_DIO) # 左移dat1

    # 都是下降沿读取数据
    dat2 = 0
    for i in range(0, 8):
        dat2 = dat2 | GPIO.input(ADC_DIO) << i # 左移DIO输出
        GPIO.output(ADC_CLK, 1); wiringpi.delayMicroseconds(usdelay)
        GPIO.output(ADC_CLK, 0); wiringpi.delayMicroseconds(usdelay)

# 这里在一个时钟脉冲里将CS设为高电平，使ADC0832停止工作
    GPIO.output(ADC_CLK, 1)
    GPIO.output(ADC_CS, 1); wiringpi.delayMicroseconds(usdelay)
    GPIO.output(ADC_CLK, 0); wiringpi.delayMicroseconds(usdelay)
```

```
    if dat1 == dat2: # 比较两次读取的数据，确保数据正确
        return dat1
    else:
        return 0

def loop(): # 循环读取，两个模式都进行
    while True:
        res0 = getResult(0)
        res1 = getResult(1)
        print ('res0 = %d, res1 = %d' % (res0,res1))
        time.sleep(0.4)

# 当本文件是主文件时运行的内容，但本质上不会运行，本文件当作库使用
if __name__ == '__main__':
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

0.3 实验内容

0.3.1 实验 1：树莓派控制 ADC0832 测量电位器电压

1. 实验准备

· 库安装——使用 apt 包管理

· 导入代码——直接用 git，配置好后 pull 下来

· 连接基本线路

  ○ 第一个实验只连接电位器和示波器

◦ 确认衰减开关为 1x 还是 10x，应选择 1x

2. 具体操作上

· 我们选择先实现电位器、ADC0832 和树莓派部分的连接

· 运行代码，发现树莓派上有输出，调整电位器，输出变化

· 然后再将示波器接入，使示波器有输出

· 我们使用 scrot 库实现直接在树莓派上截图，并通过语雀实现在树莓派上直接记录数据并上传至云端

· 调整电位器，在 0 3.3V 直接等间距测量数据，我们选择的是 0.3V 的间距，进行记录

0.3.2 实验 2：树莓派控制 ADC0832 测量正弦电压

1. 实验操作

· 不改变树莓派和 ADC0832 间的电路连接关系

· 拆除电位器相关电路，并将输入转换为信号发生器输入

◦ 在电路连接完成前将输入关闭

· 调整示波器相关电路，进行测试

· 首次打开输入，运行代码便可以得到正常的数据输出，但示波器有问题

· 调整示波器

◦ 我们先使用 auto mode 进行初始化，然后调整到手动模式确定了直流耦合，但输出依旧异常

◦ 在接入示波器自身提供的测试输入时，同样有不明干扰

◦ 我们怀疑是输入的频率太低，便将输入调整到了 100Hz，此次再使用 auto mode，发现示波器能够正常输出了，树莓派上的数字输出也没有问题

根据图 4. 可以看到示波器左下角正常显示了 99.9999Hz

◦ 之后我们又将输入调整到 10Hz，示波器的输出又正常了，原因可能是 10Hz 的信号未达到示波器 AUTO 功能的最低使用频率，在成功识别 100Hz 的信号后，示波器才找到电压信号。

◦ 调整输入电压（默认偏置为最高电压的一半），进行多次定时采样和波形记录

◦ 之后针对需要回答的问题又设置了一组无偏置的测试

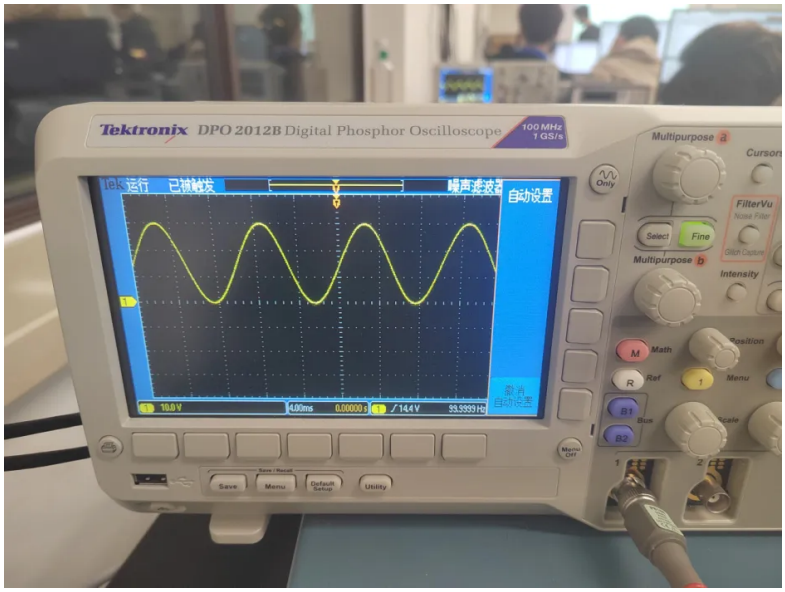


图 4: 示波器展示图

2. 实验记录

1. 1.0Vpp 500mV

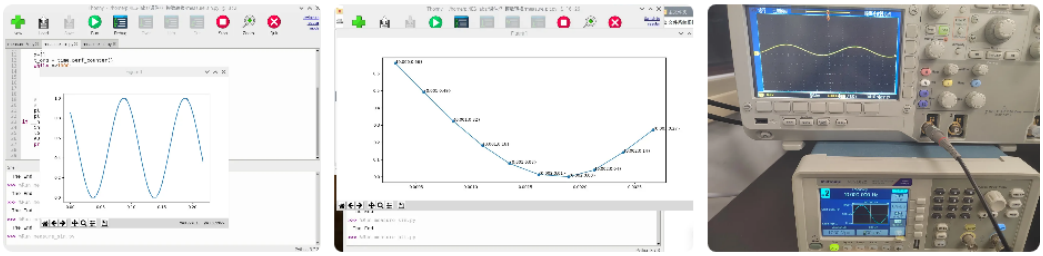


图 5: 实验图片 1

2. 2.0Vpp 1V

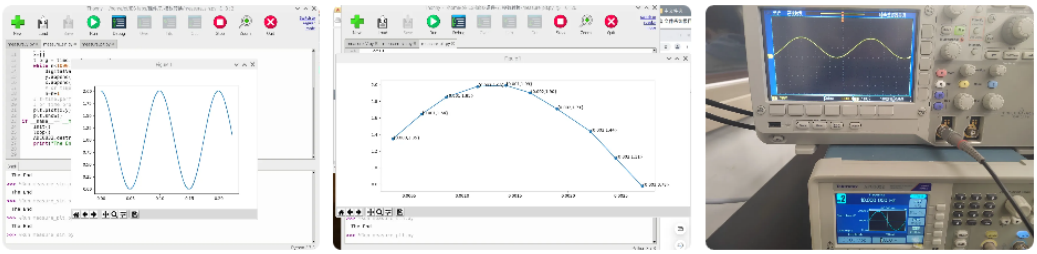


图 6: 实验图片 2

3. 3.0Vpp 1.5V

图 7: 实验图片 3

4. 3.3Vpp 1.65V

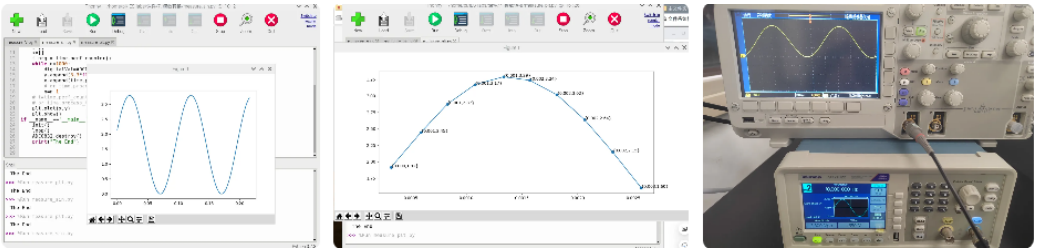


图 8: 实验图片 4

5. 3.0Vpp 0v

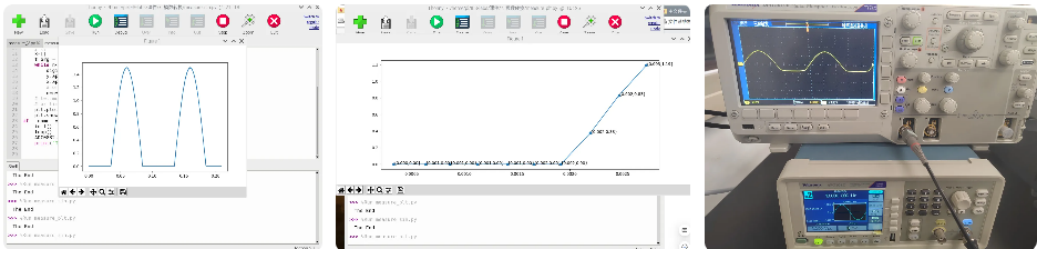


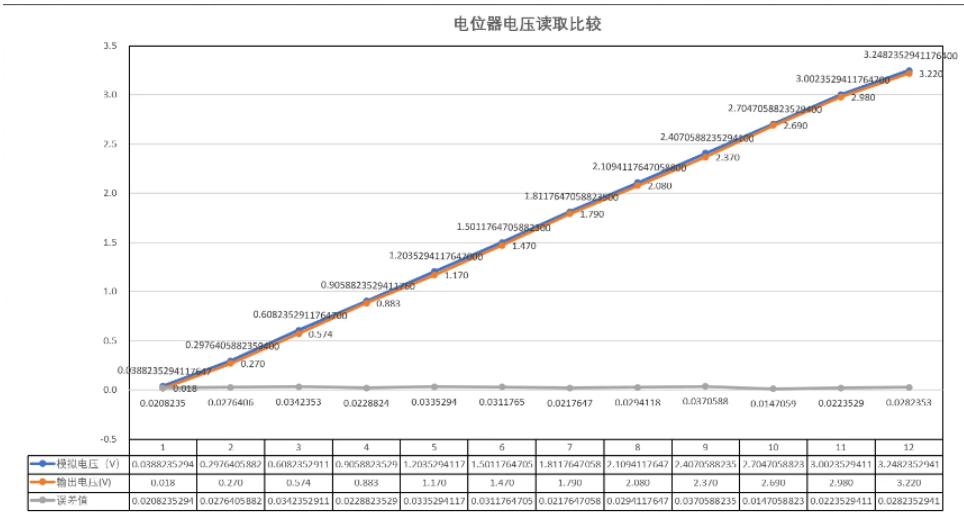
图 9: 实验图片 5

0.4 实验分析

0.4.1 电位器电压读取比较

使用 Excel 进行数据处理:

· 数据总结图:



	1	2	3	4	5	6	7	8	9	10	11	12
模拟电压（V）	0.0388235294	0.2976405882	0.6082352911	0.9058823529	1.2035294117	1.5011764705	1.8117647058	2.1094117647	2.4070588235	2.7047058823	3.0023529411	3.2482352941
输出电压(V)	0.018	0.270	0.574	0.883	1.170	1.470	1.790	2.080	2.370	2.690	2.980	3.220
误差值	0.0208235294	0.0276405882	0.0342352911	0.0228823529	0.0335294117	0.0311764705	0.0217647058	0.0294117647	0.0370588235	0.0147058823	0.0223529411	0.0282352941

图 10: 实验数据图

· 误差分析:

模拟电压（V）	输出电压(V)	误差值
0.0388235294117647	0.018	0.0208235294117647
0.2976405882359400	0.270	0.0276405882359400
0.6082352911764700	0.574	0.0342352911764701
0.9058823529411760	0.883	0.0228823529411760
1.2035294117647000	1.170	0.0335294117647000
1.5011764705882300	1.470	0.0311764705882300
1.8117647058823500	1.790	0.0217647058823500
2.1094117647058800	2.080	0.0294117647058800
2.4070588235294100	2.370	0.0370588235294100
2.7047058823529400	2.690	0.0147058823529402
3.0023529411764700	2.980	0.0223529411764698
3.2482352941176400	3.220	0.0282352941176396
误差均值	0.0269847547	
误差方差	0.0000397174	
误差标准差	0.0063021704	
两组数据的相关系数	0.9999813174	

图 11: 误差分析结果

示波器显示电压与树莓派显示电压间约有 0.027v 的差距，且相关系数较高，各组值较为恒定，认为其产生原因大致由实验设计而出现，偶然误差较小。

0.4.2 示波器探头的直流耦合和交流耦合间的区别

· 示波器探头的直流耦合和交流耦合是指探头对输入信号的响应方式。它们的区别在于探头对输入信号中直流分量的处理方式不同。

· 直流耦合:

  ◦ 直流耦合会传输所有频率的信号，包括直流信号和交流信号。

◦ 在直流耦合下，探头会传输输入信号的完整波形，无论它是直流偏置还是交流信号。

◦ 这种耦合方式适用于需要查看信号的直流偏置或慢速变化的情况。

· 交流耦合：

◦ 交流耦合只传输输入信号的交流成分，而将直流部分阻止。

◦ 通过交流耦合，示波器可以忽略输入信号的直流偏置，仅显示信号的变化部分。

◦ 这种耦合方式适用于需要关注信号变化的情况，忽略信号的直流成分。

Q: 为什么本次实验需要使用直流耦合？

A: 本次实验中设置了直流偏置，需要在示波器中呈现出来，查看整体的电压值情况。

0.4.3 为什么信号发生器的输出要加直流偏置

如果信号发生器的输出不加直流偏置，那么正弦信号的波形将以 0 点为中心对称地在 x 轴上下波动。这样的信号波形的特点是，其周期的一半处于正电平（大于 0），而另一半处于负电平（小于 0）。

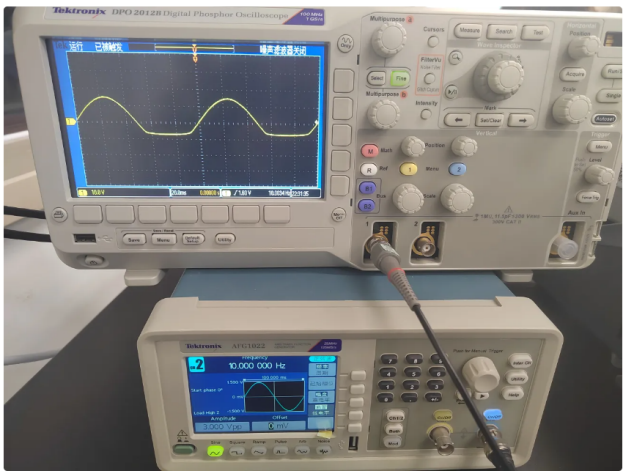


图 12: 示波器探测到的信号图

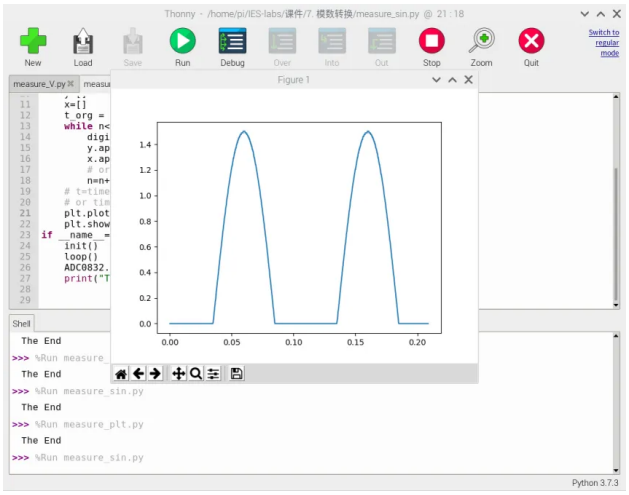


图 13: 程序反馈的信号图

尝试输入非直流信号，可以看到在树莓派以及示波器中，负电压值均未正常显示。考虑到电路:

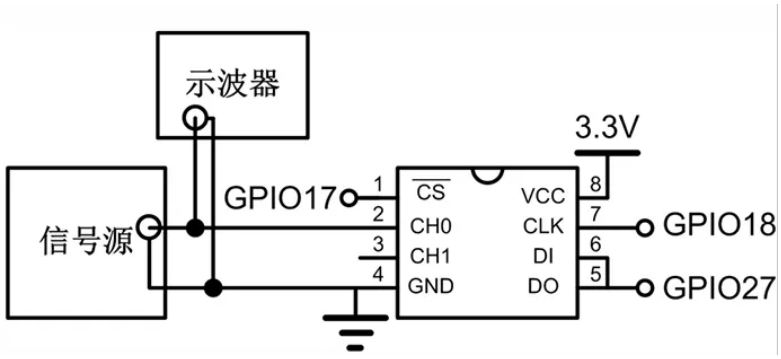


图 14: 连接电路示意图

在用 ADC0832.getResult()，信号源中产生负电压信号时，其得到的数字信号值为 0。

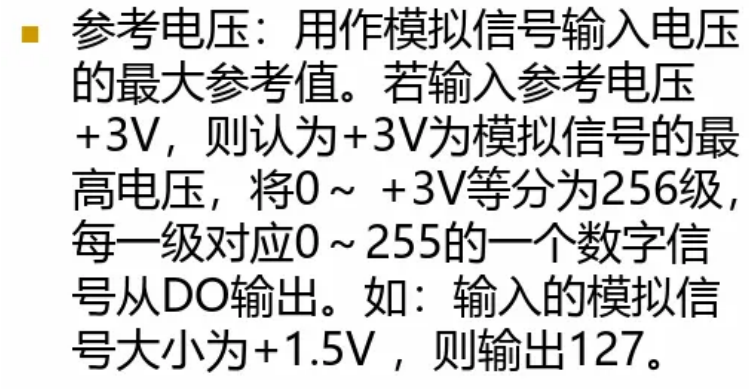


图 15: 参考电压的运用

然而，ADC0832 是一个单端输入的模数转换器，它只能接受正电压信号。如果输入信号存在负电平部分，ADC0832 将无法正确地进行模数转换，可能导致不准确的读数或者损坏转换器。

ADC0832 中因没有 ref-脚（接地），无法表示负值，在用信号源输入负值时，其值为 0，而输入模拟电压为树莓派 3.3v 电压，其所测电压值也就只能在 0 3.3v 之间。为了解决这个问题，需要给正弦信号加上直流偏置，使得信号的波形完全位于正电平以上。这样，在整个信号周期内，信号都保持正电压状态，从而确保 ADC0832 正确地进行模数转换。

故本次实验必须加上直流偏置，使得整个数字信号产生值位于 0 3.3v 之间，而不产生负值，其偏置值设为最高电压差值的一半。

这也解释了为什么我们在实验过程中为什么一直设置偏置为最高值的一半并打开直流耦合.

0.5 总结与思考:

1. 本次实验通过使用模数转换器 ADC 来模拟示波器的工作运行，深入了解模数转换器 ADC 的工作原理和工作过程，切实的感受了模数转换器将连续的模拟信号转换成离散的数字信号的过程。

2. 模数转换是对输入侧信息处理的过程，在各种输入端的器件中的数据处理中均存在，对于接下来学习的内容的理解有重要意义。