

I2C接口与超声测距

2022/10/29

电子系统导论教学团队

实验目的

- 了解串行通信的基本概念
- 掌握树莓派I2C接口的编程方法
- 掌握通过I2C接口的超声模块进行测距

串行通信

■ 串行通信

- 采用串行通信协议（**serial communication**）在一条信号线上将数据一个比特一个比特地逐位进行传输的通信模式
- 收发两端波特率一致
- 包括同步通信与异步通信
- 优点：节省端口
- 缺点：底层控制逻辑复杂

■ 通讯方式

- 单工模式：数据传输是单向的
- 半双工模式：可以接收和发送数据，但不能同时收发
- 全双工模式：可以同时收发数据

串行同步通信

■ 串行同步通信

- 发送端与接收端时钟信号保持同步，包括频率和相位
- 需传输时钟信号
- 数据组成信息帧进行传输，由同步字符、数据字符和校验字符（**CRC**）组成。同步字符位于帧开头，标志数据字符的开始；数据字符在同步字符之后，个数没有限制，由所需传输的数据块长度来决定；校验字符有1到2个。
- 优点：传输数据的位数不受限制，通信效率较高
- 缺点：时钟信号需保持精确同步
- 典例：I2C通信

串行异步通信

■ 串行异步通信

- 又称起止式异步通信，以字符为单位进行传输，字符中每位时间间隔固定，字符间时间间隔不固定
- 收发同步通过在字符格式中设置起始位和停止位实现
- 停止位和下一个起始位之间有不定长的空闲位
- 起始位为低电平，停止位与空闲位为高电平
- 收发双方时钟源可彼此独立，可互不同步
- 典例：RS232通信

I2C总线协议

- I2C总线是一种双向二线制同步串行总线，用于总线上设备间的信息传输。两根线分别称为SDA（串行数据线）和SCL（串行时钟线），都是双向I/O线，接口电路为开漏输出。需通过上拉电阻接电源VCC。
- 特征
 - 树莓派上已集成**I2C**接口控制器，不需要额外的接口电路，且片上接口电路的滤波器可滤去总线数据上的毛刺
 - 总线可连接多个**I2C**设备，每个设备即可作为主机又可作为从机，但同一时刻只允许有一个主机
 - 连到总线上的设备只受总线最大电容限制
 - 有三种工作模式：标准(可达100kbit/s)、快速(可达400kbit/s)和高速模式(可达3.4Mbit/s)。向下兼容、向上不兼容
 - 总线具有极低电流消耗，高抗噪声干扰

注：树莓派默认**I2C**通信速率100kbit/s

I2C总线时序

- 空闲状态
 - SDA与SCL信号线均为高电平
- 起始信号
 - 当SCL为高期间，SDA由高到低的跳变。基于边沿检测。
- 停止信号
 - 当SCL为高期间，SDA由低到高的跳变。基于边沿检测。



I2C总线时序

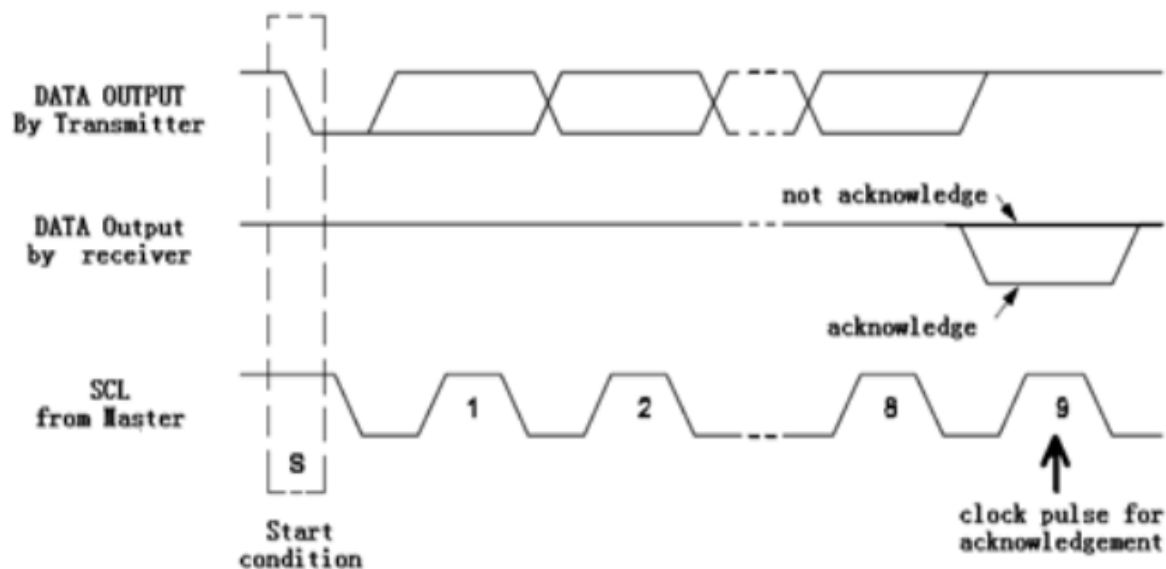
■ 应答位

- 发送器每发送一个字节，就在第9个时钟脉冲期间释放数据线，由接收器反馈一个应答信号。有效应答位为低电平，简称应答位（**ACK**）；非应答位（**NACK**）为高电平，一般表示该字节未接收成功。
- 应答位要求，接收器在第9个时钟脉冲之前的低电平期间将**SDA**线拉低，并确保在该时钟脉冲高电平期间保持为稳定的低电平。
- 若接收器是主控器，则在收到最后一个字节后，发送一个非应答位（**NACK**），通知发送器结束数据发送，并释放SDA线，随后接收器发送一个停止信号P。

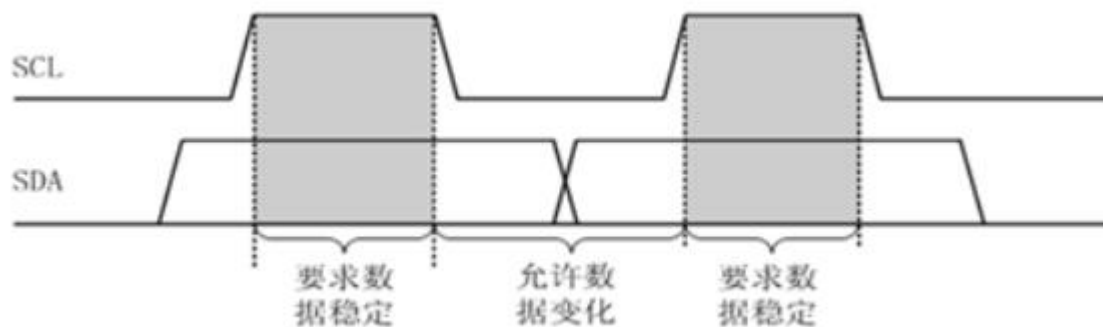
■ 数据的有效性

数据传送时，时钟信号为高电平期间，数据线上的数据必须保持稳定，只有在时钟信号低电平期间，数据线上的数据才能改变。

I2C总线时序

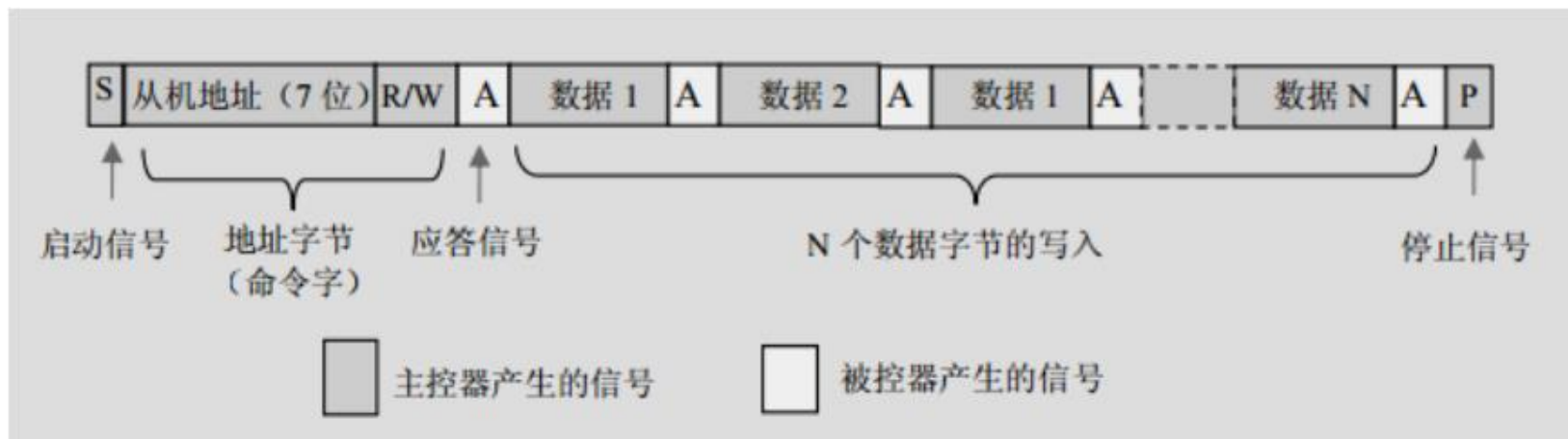


I²C 总线的响应



I2C数据传输

■ 主机发送数据流程



1. 主机在检测到总线为“空闲状态”（即 SDA、SCL 线均为高电平）时，发送一个启动信号“S”，开始一次通信的开始
2. 主机接着发送一个命令字节。该字节由 7 位的外围器件地址和 1 位读写控制位 R/W 组成（此时 R/W=0）
3. 相对应的从机收到命令字节后向主机回馈应答信号 ACK（ACK=0）

I2C数据传输

■ 主机发送数据流程

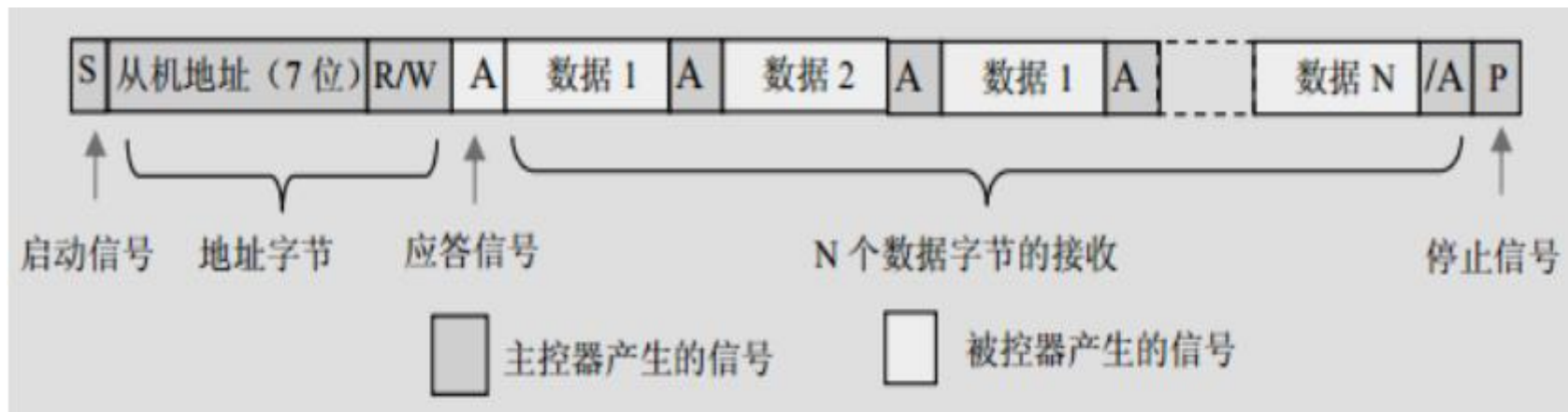
4. 主机收到从机的应答信号后开始发送第一个字节的数据
5. 从机收到数据后返回一个应答信号 ACK
6. 主机收到应答信号后再发送下一个数据字节
7. 当主机发送最后一个数据字节并收到从机的 ACK 后，通过向从机发送一个停止信号P结束本次通信并释放总线。从机收到P信号后也退出与主机之间的通信

注:

- ① 主机的一次发送通信，其发送的数据数量不受限制，主机通过 P 信号通知发送的结束
- ② 主机的每一次发送后都是通过从机的 ACK 信号了解从机的接收状况，如果应答错误则重发。

I2C数据传输

■ 主机接收数据流程



1. 主机发送启动信号后，接着发送命令字节（其中 $R/W=1$ ）
2. 对应的从机收到地址字节后，返回一个应答信号并向主机发送数据
3. 主机收到数据后向从机反馈一个应答信号

I2C数据传输

■ 主机接收数据流程





















4. 从机收到应答信号后再向主机发送下一个数据
5. 当主机完成接收数据后，向从机发送一个“非应答信号（ACK=1）”，从机收到非应答信号后便停止发送
6. 主机发送非应答信号后，再发送一个停止信号，释放总线结束通信

注：

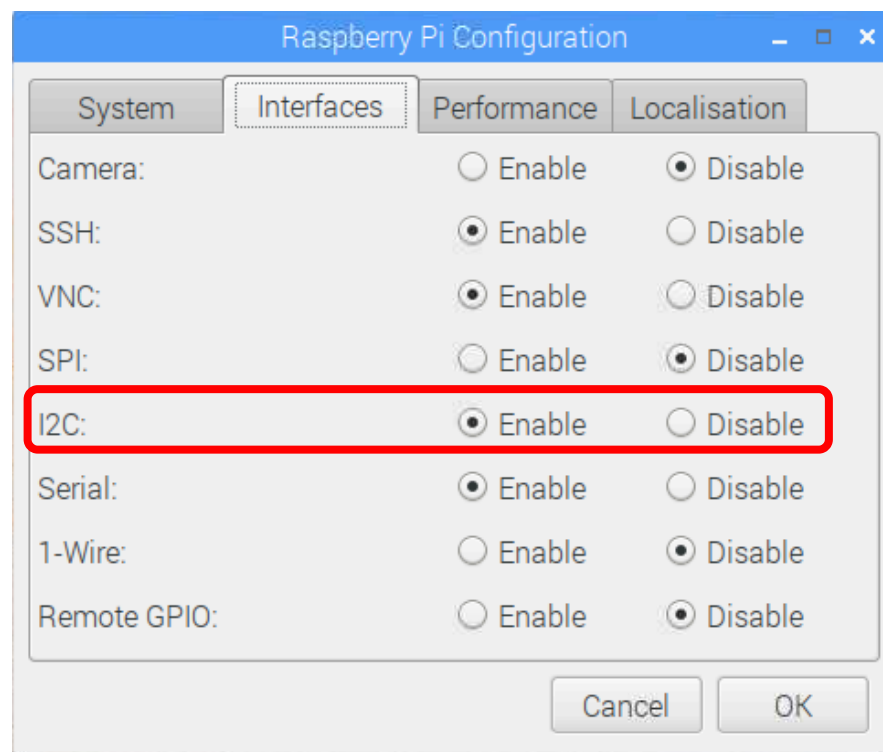
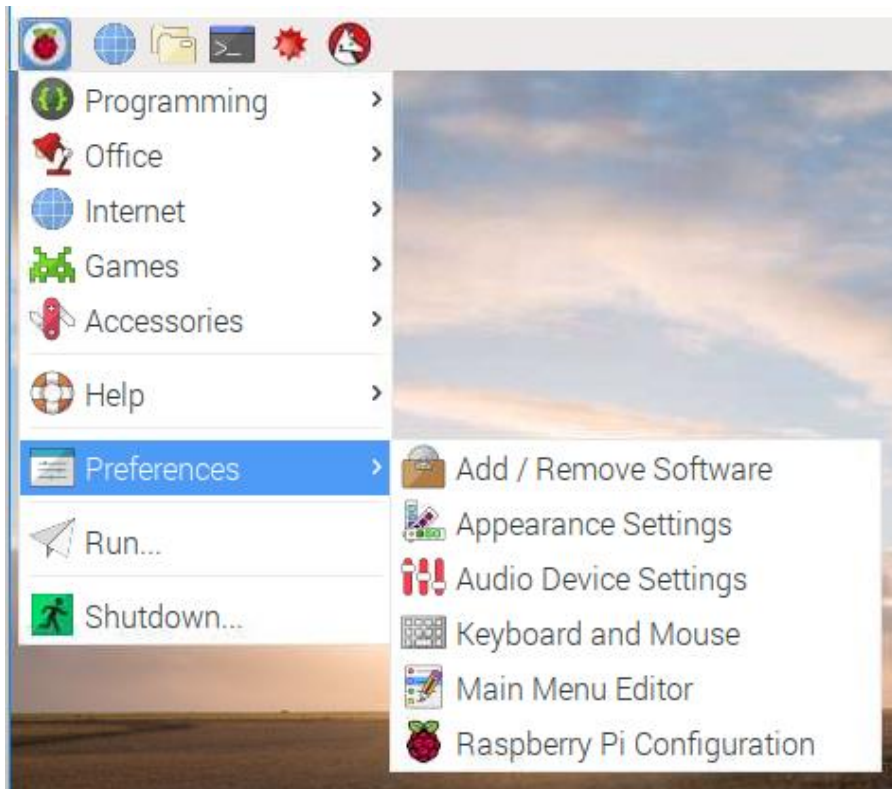
主机所接收数据的数量是由主机自身决定，当发送“非应答信号/A”时从机便结束传送并释放总线（非应答信号的两个作用：前一个数据接收成功，停止从机的再次发送）

树莓派I2C通信

- 开启I2C功能
- I2C设备KS103简介
- 安装I2C通信工具
 - i2c-tools
 - smbus
 - pigpio
 - wiringpi
 - I2C编程实践

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)		DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)		(I ² C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

开启I2C功能



KS103简介

- KS103是一款简单好用的超声测距器件，配备I2C接口。
- 功能
 - 探测范围1cm ~ 550cm
 - 工作电压：3.0V~5.5V 直流电源（实验选择3.3V）
 - 工作时瞬间最大电流：10.6mA @5.0V, typical
 - 工作电流：1.6-2.7 mA @5.0V, typical
 - 休眠时最大耗电量：500uA @5.0V, typical
 - 使用I2C 接口与主机通信，自动响应主机的I2C 控制指令
 - I2C 通信速率建议不要高于20kbit/s。
 - 短距探测量程由10cm、20cm、.....、至470cm，满足快速近距离探测
 - 快速、高精度的温度探测,快速光强探测



KS103简介

■ 探测指令

□ 格式:

I2C 地址	寄存器 2	8 位数据
--------	-------	-------

- 测距(写): 寄存器 2, 命令 0xb0。 (0-5m范围, 返回探测距离(mm), 数据存放在2号, 3号寄存器中(下同)。探测最大耗时约33ms)
- 测距(写): 寄存器 2, 命令 0xb2。 (0-5m范围, 返回超声传播时间(us)。探测最大耗时约32ms)
- 测距(读): 寄存器 2, 返回探测数据的高8位 (高位在前)
- 测距(读): 寄存器 3, 返回探测数据的低8位 (高位在前)
- 测温(写): 寄存器 2, 命令0xc9/0xca/0xcb/0xcc, 返回9/10/11/12位精度的温度数据, 按DS18B20格式

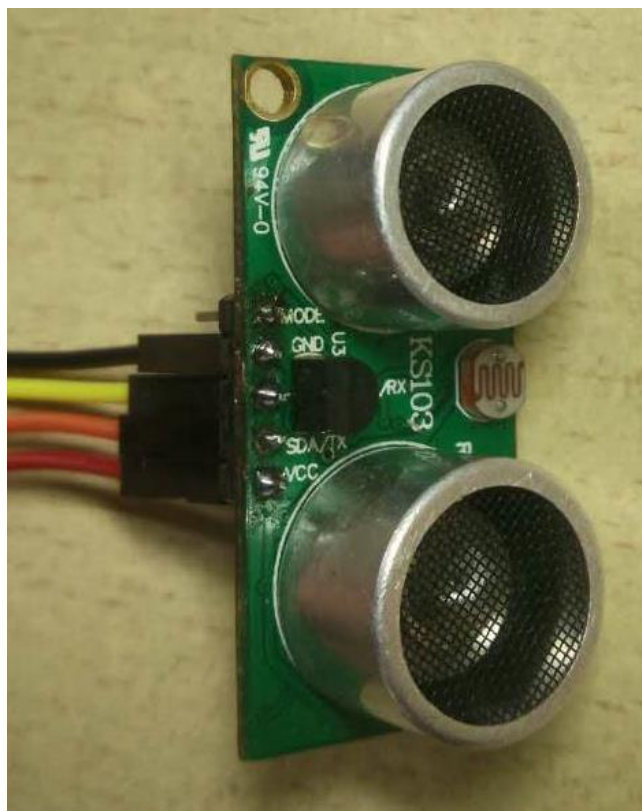
注: DS18B20格式, 9/10/11/12位精度对应的温度精度分别为 0.5/0.25/0.125/0.0625℃, 得到16位数据, 补码表示, 前5位为符号位, 乘0.0625得到摄氏温度值

树莓派与KS103连接

- 将树莓派与KS103的SDA、SCL管脚对应连接，不必接上拉电阻。KS103的VCC、GND管脚分别接树莓派的3.3V DC Power管脚和GND管脚



- GND
- SCL
- SDA
- VCC



i2c-tools

- 安装i2c-tools（已安装好 了解安装方式）
 - i2c-tools是一个简单好用的I2C设备调试工具
 - 安装: `sudo apt-get install i2c-tools` (Raspbian已预装)
- i2c-tools的执行函数
 - `i2cdetect` – 列举I2C bus和上面所有的设备
 - `i2cdump` – 显示设备上所有寄存器的值
 - `I2cget` – 读取设备上某个寄存器的值
 - `i2cset` – 将值写入设备上某个寄存器

i2c-tools

■ I2C总线扫描

```
pi@raspberrypi:~ $ i2cdetect -l  
i2c-1    i2c                bcm2835 I2C adapter
```

I2C adapter

■ I2C设备查询

- 指令: `i2cdetect -y 1`
- `-y`表示取消交互, 1表示I2C总线编号
- 表里74为设备地址, 即0x74 (7位地址格式)

```
pi@raspberrypi:~ $ i2cdetect -y 1  
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  
40:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  
60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  
70:  --  --  --  --  74  --  --  --  --  --  --  --  --  --  --
```

i2c-tools

- 寄存器内容导出
 - 指令：i2cdump -y 1 0x74
 - 导出指定设备中所有寄存器的内容

```
pi@raspberrypi:~ $ i2cdump -y 1 0x74
No size specified (using byte-data access)
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f      0123456789abcdef
00: 0c 1d 00 00 00 00 2d 00 00 00 00 00 00 00 00 00  ??.....-.....
10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
```

i2c-tools

■ 寄存器内容写入

- ❑ 指令: `i2cset -y 1 0x74 0x2 0xb0`
- ❑ 0x74 表示设备总线编号
- ❑ 0x2 表示寄存器地址
- ❑ 0xb0 表示写入寄存器的值

■ 寄存器内容读出

- ❑ 指令: `i2cget -y 1 0x74 0x2`
- ❑ 0x74 表示设备总线编号
- ❑ 0x2 表示寄存器地址
- ❑ 返回0x2寄存器的值

超声测距-基于smbus

- **System Management BUS**，即系统管理总线，有两条信号线，分别为SMCLK和SMDAT，总线规范基于I2C总线规范，但有一定的区别。在本例中，通用。
- 安装： `sudo apt-get install python-smbus` (Raspbian已预装)
- 0x74 表示设备总线编号
- 函数列表
 - `read_byte_data` 读一个字节
 - `write_byte_data`
 - `read_word_data` 读两个字节，小端读取，第一个字节认为是低位
 - `write_word_data`
 - `read_i2c_block_data`
 - `write_i2c_block_data`

超声测距-基于smbus

- KS103 I2C接口有一个数据缓冲区，里面有较多的寄存器。可认为一个寄存器内存储一字节数据。
- 写数据：

`write_byte_data(addr,reg,value)`

其中，addr为设备地址，reg为寄存器编号，value为写入寄存器的值，根据KS103说明书，reg=0x2, value若采用0xb0, 则返回距离值，单位mm；若采用0xb2, 则返回声音传播时间，单位us。测出数据为16位二进制数，高8位存在2号寄存器中，低8位存在3号寄存器中。

- 读数据：

`read_byte_data(addr,reg)`

读出2号、3号寄存器内的值，并计算测得数据值

注：距离探测需要一定时间，发出探测指令后，至少等33ms才能读数据

超声测距-基于smbus

```
import smbus
import time

bus = smbus.SMBus(1) #open /dev/i2c-1
address = 0x74 #i2c device address
wr_cmd = 0xb0 #range 0-5m, return distance(mm)
#rd_cmd = 0xb2
##range 0-5m, return flight time(us), remember divided by 2
try:
    while True:
        bus.write_byte_data(address, 0x2, wr_cmd)
        time.sleep(1) #MIN ~ 0.033
        HighByte = bus.read_byte_data(address, 0x2)
        LowByte = bus.read_byte_data(address, 0x3)
        Dist = (HighByte << 8) + LowByte
        print('Distance:', Dist/10, 'cm')
        #time.sleep(2)
except KeyboardInterrupt:
    pass
bus.close()
print('Range over!')
```

详见*i2c_range_smbus.py*

超声测距-基于pigpio

- pigpio是一个用C语言编写的高效的树莓派GPIO库（系统已预装）。

- 使用方法

- 启动pigpiod. 在终端输入

```
sudo pigpiod
```

- 查看pigpiod是否已成功开启

```
ps -A | grep pigpiod
```

- 查看是否连接成功

```
if not pi.connected:  
    exit()
```

- 创建一个连接daemon的实例 pi

```
import pigpio  
pi = pigpio.pi()
```

超声测距-基于pigpio

打开I2C设备:

```
handle = pi.i2c_open(1, addr)
```

其中, **addr**为设备地址, 返回设备对象句柄

写指令:

```
pi.i2c_write_byte_data(handle, reg, value)
```

其中, **handle**为设备对象句柄, **reg**为寄存器编号, **value**为写入寄存器的值。

读数据:

```
pi.i2c_read_byte_data(handle, reg)
```

读出2号、3号寄存器内的值, 并计算测得数据值

关闭I2C设备:

```
pi.i2c_close(handle)
```

超声测距-基于pigpio

```
import pigpio
import time

pi = pigpio.pi()
if not pi.connected:
    exit()
address = 0x74 #i2c device address
h = pi.i2c_open(1,address) #open device at address on bus 1
wr_cmd = 0xb0 #range 0-5m, return distance(mm)
#rd_cmd = 0xb2
##range 0-5m, return flight time(us), remember divided by 2
try:
    while True:
        pi.i2c_write_byte_data(h, 0x2, wr_cmd)
        time.sleep(1) #MIN ~ 0.033
        HighByte = pi.i2c_read_byte_data(h, 0x2)
        LowByte = pi.i2c_read_byte_data(h, 0x3)
        Dist = (HighByte << 8) + LowByte
        print('Distance:', Dist/10, 'cm')
        #time.sleep(2)
except KeyboardInterrupt:
    pass
pi.i2c_close(h)
print('Range over!')
```

详见*i2c_range_pigpio.py*

超声测距-基于wiringpi

- WiringPi是应用于树莓派平台的GPIO控制库函数，使用C或者C++开发并且可以被其他语言包装，例如python、ruby或者PHP等
- 安装
 - pip3 install wiringpi
- 函数列表
 - wiringPiI2CSetup() #初始化i2c设备，返回设备对象（句柄）
 - wiringPiI2CRead() #读数据
 - wiringPiI2CReadReg16() #读取16位数据
 - wiringPiI2CReadReg8() #读取8位数据
 - wiringPiI2CWrite() #写数据
 - wiringPiI2CWriteReg16() #写入16位数据
 - wiringPiI2CWriteReg8() #写入8位数据

超声测距-基于wiringpi

- 初始化设备:

`wiringPiI2CSetup(addr)`

其中, `addr`为设备地址, 返回设备对象句柄

- 写指令:

`wiringPiI2CReadReg8(handle, reg, value)`

其中, `handle`为设备对象句柄, `reg`为寄存器编号, `value`为写入寄存器的值。

- 读数据:

`read_byte_data(handle, reg)`

读出2号、3号寄存器内的值, 并计算测得数据值

超声测距-基于wiringpi

```
import wiringpi as wpi

address = 0x74 #i2c device address
h = wpi.wiringPiI2CSetup(address) #open device at address
wr_cmd = 0xb0 #range 0-5m, return distance(mm)
#rd_cmd = 0xb2
##range 0-5m, return flight time(us), remember divided by 2
try:
    while True:
        wpi.wiringPiI2CWriteReg8(h, 0x2, wr_cmd)
        wpi.delay(1000) #unit:ms MIN ~ 33
        HighByte = wpi.wiringPiI2CReadReg8(h, 0x2)
        LowByte = wpi.wiringPiI2CReadReg8(h, 0x3)
        Dist = (HighByte << 8) + LowByte
        print('Distance:', Dist/10, 'cm')
except KeyboardInterrupt:
    pass
print('Range over!')
```

详见*i2c_range_wiringpi.py*

文件列表

1. i2c_range_smbus.py 超声测距，使用smbus库
2. i2c_range_pigpio.py 超声测距，使用pigpio库
3. i2c_range_wiringpi.py 超声测距，使用wiringpi库
4. i2c_temp.py 温度测量，使用smbus库（补充）

参考资料

1. KS103 datasheet, KS103_ZH.pdf
2. I2C相关知识, <http://blog.csdn.net/hygzxf/article/details/17416725>
3. DS18B20数据格式, <https://baike.baidu.com/item/DS18B20>
4. pigpio library python 接口: <http://abyz.me.uk/rpi/pigpio/python.html#stop>

实验内容

- 在树莓派初始化I2C接口，并完成超声测距

实验报告中需要回答的问题

1. 如何确定I2C总线上从设备的地址？
2. 简述从KS103上读取距离的流程。

致谢

- 本课件由以下同学协助编写
 - 邱云辉(14307130225)