实验报告 5

I^2C 总线与专家系统简介

范皓年 1900012739　信息科学技术学院

2020 年 11 月 12 日

目录

1 实验目的

1. 了解 I^2C 总线传输的原理。
2. 编写 Python 程序，访问扩展板上的 I^2C 设备。
3. 熟悉并使用 I^2C 调试工具。
4. 了解专家系统的基本概念和设计方法。

2 实验原理

2.1 I^2C 总线传输原理

I^2C 总线为“Inter-Intergrated Circuit”的简写，最初由荷兰 Philips 实验室提出，用于解决引脚数目和成本的瓶颈；相比 SPI，I^2C 连接线更少，使得印刷电路板更加简单。

出于减少接线的考虑，I^2C 取消了额外的地址译码器及片选信号，由于不涉及片选，可以构成多从（multi-slave）的系统，多个器件可以简单的连接到同一条 I^2C 总线，多个从系统的调度通过逻辑信号实现。当然，这样的结果是信号的逻辑结构变得复杂，一个直接的表现是 I^2C 的传输速率低于全双工的 SPI。主设备发起/结束一次传输，并维护时钟信号，从设备根据主设备的寻址来响应数据传输。在同一时刻允许一个设备发送数据至总线，其余设备接收总线的信号。I^2C 总线由两根线 SDA 及 SCL 两根线连接主从设备，两根线都采用上拉方式连接到 VDD。

I^2C 支持多种模式下速率传输，分别是标准模式、快速模式、快速模式 +、高速模式。传输速率从 100kbit/s（标准模式）至 3.4Mbit/s（高速模式）。一个典型传输过程如图1：

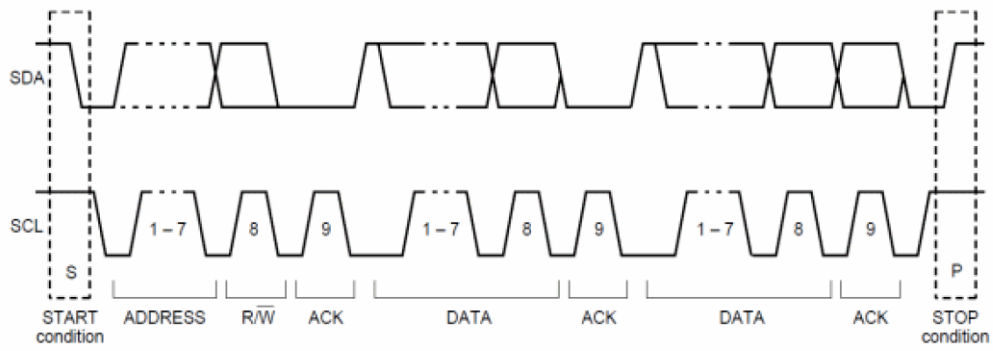
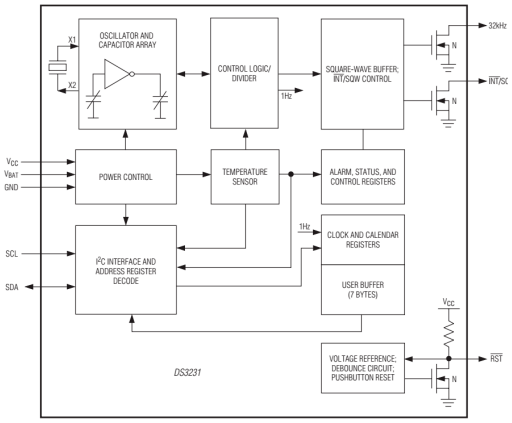


图 1

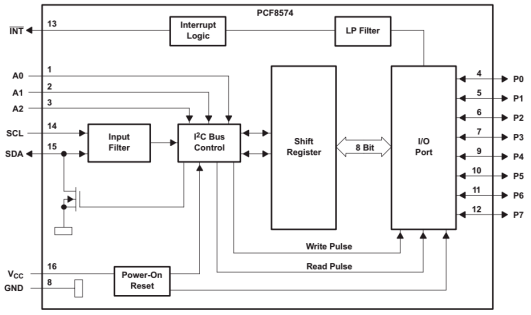
I^2C 数据传输由主设备产生一个起始位开始，然后传递 7 个地址位（对于 7 位地址模式），指定通信的从设备。接下来传递的一位是读写位，0 表示写入，1 表示读出。再接下来的一位由从设备驱动，如果从设备的地址与主设备发出的地址相同，从设备将把 SDA 信号拉低，表示确认这次数据传输（ACK）。当从设备地址被确认，真正的数据传输就开始了。如果是写入数据，则数据线仍由主设备驱动，从设备在每个字节传输之后也要驱动一位的 ACK 信号。如果是读出数据，则数据线由从设备驱动，主设备仅在应答时输出 ACK 位。最后由主设备产生一个停止位表示数据传输结束。所有的 I2C 数据与地址，都是先传递最高位，最后传递最低位。

2.2 扩展板上的 I^2C 设备

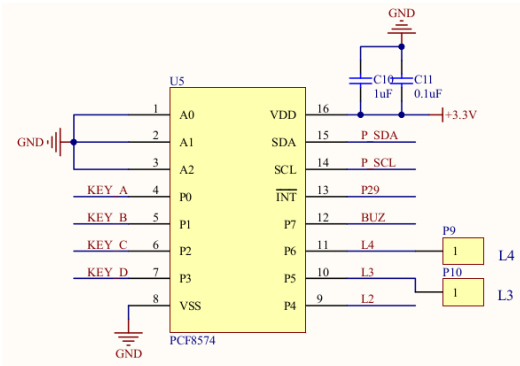
在扩展板上连接了多个 I^2C 设备，有 DS3231、BMP280 和 PCF8574。DS3231 是一款高度集成的 RTC（Real Time Clock）时钟芯片，自动维护时分秒、年月日时间信息。其内部框图如图2(a)所示，内部寄存器如图2(b)所示。PCF8574 是基于 I^2C 接口的 8bit IO 接口扩展芯片，通过其接收方向杆的输入及控制蜂鸣器的输出，其内部框图如图2(c)所示，外设连接如图2(d) 所示。从图2(d)中可以看出：五方向摇杆按钮的四个方向接到了 PCF8574 的 P0 到 P3 端口（分别对应左上下上四个方向），P4 端口接了发光管 LED2，P7 端口接了蜂鸣器输出。



(a) DS3231 内部框图



(b) DS3231 内部寄存器



(c) PCF8574 内部框图

SPPR2	SPPR1	SPPR0	SPR2	SPR1	SPR0	BaudRateDivisor	BuadRate
0	0	0	0	0	0	2	12.5MHz

(d) PCF8574 外设连接

图 2: I^2C 扩展板

2.3 I^2C 设备访问

在树莓派中开发环境中已经安装了安装 smbus 库和 i2c-tools 工具。Smbus 完成了一些基本 I2C 函数的封装，可以使用 pydoc smbus 获取库帮助信息。

i2c-tools 中 i2cdetect 完成树莓派 I2C 设备的扫描，如下所示：

```
1  pi@raspberrypi:~ $ sudo i2cdetect -y 1
2  0 1 2 3 4 5 6 7 8 9 a b c d e f
3  00: -- -- -- -- -- -- -- -- -- -- -- -- --
```

```
    10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
    20: 20 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
    30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
    40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
```

根据扫描的结果，树莓派共接入了四个 I2C 设备，其中地址 20 为 PCF8574 IO 扩展芯片（参考图 2(d) 和图2(c)，它连接了五方向按键的其中四个方向、LED2 发光管和 BUZ 蜂鸣器）；地址 48 为 PCF8591 AD/DA 扩展芯片（参考图2(c)）；地址 68 为 DS3231 实时钟芯片（参考图2(c)）；地址 76 为 BMP280 芯片（为气压传感器，本实验暂未使用）。

扩展板 RTC DS3231 的地址为 0x68，挂载在 i2cbus-1 上，下面示例给出了 RTC 的访问流程：

```python
import smbus
import time # 包含相关库文件
address = 0x68
register = 0x00
bus = smbus.SMBus(1) # 初始化 i2c Bus
# FixTime 定义为 2019 年 6 月 12 日 18 时
FixTime = [0x00,0x00,0x18,0x03,0x12,0x06,0x19]
# 定义时钟操作函数
def ds3231SetTime():
    bus.write_i2c_block_data(address,register,FixTime)
def ds3231ReadTime():
    return bus.read_i2c_block_data(address,register,7);
ds3231SetTime() # 设置时间
ds3231ReadTime() # 读出时间
```

write_i2c_block_data 中的第一个参数是 I2C 的地址，第二个参数是传递给从设备的命令，DS3231 的 I2C 协议规定这个命令就是后续传递数据的起始地址。根据图2(b) 中寄存器的描述。ds3231SetTime 写入了前 7 个地址，分别设置了年月日时分秒和星期的内容。ds3231ReadTime 读出前 7 个地址，也包含了全部的时间信息。注意这里 DS3231 中保存的数据都是 BCD 编码类型，也就是 0x19 代表十进制数 19，在操作和应用的时候要注意转换。

2.4 专家系统和 Prolog 编程语言

所谓的专家系统（Expert System）是一种利用知识进行推理的程序。专家系统具有某些领域的知识的规则库（rules），例如鸟类识别专家系统关于几种鸟类的规则库可能是：

```
喜鹊是留鸟、颜色黑白、体型中等
白鹭是旅鸟、颜色白、体型大
翠鸟是留鸟、颜色蓝、体型小
```

当给出一些事实（facts），专家系统可以利用它们进行推理，得到特定的结论。例如给出

```
居留类型：留鸟
颜色：蓝
体型：小
```

专家系统就可以得出查询的鸟是翠鸟的答案。

当然前面的示例过于简单，并不能体现专家系统的优势。一个复杂的规则库和完备的推理程序可以完成只有人类专家才可以做到的任务，这类任务往往规则库非常复杂，人类需要多年的学习和积累经验才能完成，这也是为什么这类软件被称为专家系统的原因。常用的应用方向有工程、医药、军事等。

编写专家系统一般需要特殊的编程语言，这类语言具有逻辑推理机制，可以提高编程的效率，减少程序出错的概率。其中比较有代表性的就是 Prolog。

Prolog 是一种逻辑型程序设计语言，它的英文就是 Programming in Logic 的缩写。它具有自动搜索、模式匹配、回溯等性能，主要特点是以谓词逻辑为理论基础。用 Prolog 编程时，人们注重和关心对问题的描述，而不是问题的求解过程。例如前面小节中的规则库和事实库可以用 Prolog 描述如下：

```
1 bird(magpie) :- color(black_white), season(all_year), size(medium).
2 bird(egret) :- color(white), season(spring_autumn), size(large).
3 bird(kingfisher) :- color(blue), season(all_year), size(small).
4 season(all_year).
5 color(blue).
6 size(small).
```

在这个示例中我们可以看到，Prolog 每行语句由句点结束，包含:- 的表示规则，代表蕴含的关系，例如 bird(magpie) 为真的条件是:- 后面的三个函数都为真。不包含:- 的具有为事实，例如 color(blue) 这个函数永远为真。

swipl 是 Prolog 的一个实现，下面的交互示例假定前面的代码列表保存在 bird.pl 中。

```
1 $ swipl bird.pl
2 Welcome to SWI-Prolog (threaded, 32 bits, version 8.0.2)
3 SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
4 Please run ?- license. for legal details.
5 For online help and background, visit http://www.swi-prolog.org
6 For built-in help, use ?- help(Topic). or ?- apropos(Word).
7 ?- bird(kingfisher).
8 true.
9 ?- bird(magpie).
```

```
false.
?- bird(X).
X = kingfisher.
?- halt.
```

大写开始的单词表示变量，因此 bird(X) 的查询结果给出了变量 X 的值。

2.5 语言补充：pyswip 和 python dict

pyswip 是 Python 语言与 swipl 的接口，现在将 bird.pl 中的事实部分删除，运行如下 Python 程序，可以获得对 bird(X) 的查询结果。

```
from pyswip import Prolog # 导入模块
prolog = Prolog()
prolog.consult('bird.pl') # 导入Prolog 的代码（规则库）
prolog.assertz("color(blue)") # 添加事实
prolog.assertz('season(all_year)')
prolog.assertz('size(small)')
for result in prolog.query('bird(X)'): # query 代表查询
    print(result["X"])
```

pyswip 还支持设计 Prolog 中的外部函数，例如：

```
from pyswip import Prolog, registerForeign
def hello(t): # 包含一个参数，返回值为布尔类型
    print("Hello,", t)
hello.arity = 1 # 这个属性是必须的
registerForeign(hello)
prolog = Prolog()
prolog.assertz("father(michael,john)") # 事实1: michael 是john 的父亲
prolog.assertz("father(michael,gina)") # 事实2: michael 是gina 的父亲
print(list(prolog.query("father(michael,X), hello(X)"))) # 查询
```

和列表类似，字典也是可以保存多个元素的变量类型，其每个元素都是一个冒号分隔的键值对，例如：

```
dict = {'Alice': '1234', 'Bob': '5678', 'Charlie': '90'}
```

可以通过“键”对“值”进行查询，也可以对“键”进行赋值，例如：

```
print(dict['Alice'])
dict['John'] = '45'
```

字典也包含一些内置的函数，如 clear() 清除字典内容；has_key(key) 查询键是否存在；len(dict) 查询字典元素个数等。

3 实验内容

3.1 I^2C 设备的用法

3.1.1 通过 I^2C 接口访问并调用蜂鸣器

运行代码见附录A.1

3.1.2 RTC DS3231 的时间访问

扩展板上的时间存储是通过 16 进制实现的，所以特别注意编码问题，对应的时间值都是 16 进制数，比如当前是 20 年，存储为 0x20，设定初值时应该存入十进制的 16。为了解决编码问题，我们需要如下的函数：

```
def hex2dec(num):
    return num // 10 * 16 + num % 10
def dec2hex(num):
    return num // 16 * 10 + num % 16
```

前者将十六进制时间变成十进制数，后者将十进制变成十六进制。

从而我们可以将当前时间转码为一个向量，等待存入系统：

```
t = str(dt.datetime.now())

year = hex2dec(int(t[2:4]))
month = hex2dec(int(t[5:7]))
day = hex2dec(int(t[8:10]))
hour = hex2dec(int(t[11:13]))
minute = hex2dec(int(t[14:16]))
second = hex2dec(int(t[17:19]))
# FixTime[3]是星期数
FixTime = [second, minute, hour, 0x04, day, month, year]
```

设定时间和读取时间利用如下的语句（已经使用函数进行封装）

```
def ds3231SetTime():
    bus.write_i2c_block_data(address,register,FixTime)
def ds3231ReadTime():
    return bus.read_i2c_block_data(address,register,7);
```

随后每一秒进行一次读取，并将时间格式化输出：（仍然注意进制转换）

```
print('%d-%d-%.2d %s %.2d:%.2d:%.2d' % \
    (2000+dec2hex(t[6]),dec2hex(t[5]), dec2hex(t[4]), \
    weekday[t[3]],dec2hex(t[2]), dec2hex(t[1]), dec2hex(t[0])))
```

完整代码见A.2。

3.2 专家系统动物判别实例

用 prolog 写成的专家系统程序，类似于利用 SQL 在数据库中查询有效数据，只不过可以实现更加复杂的逻辑，尽管这个例子仍然是简明的。给出一组动物的属性如A.3。

随后利用 python 的 pyswip 组件编写前端。借助 registerForeign 编写外部函数，替代 prolog 规则库中的 verify 完成属性选项，prolog 函数如下；

```
verify(S) :-
    (yes(S)
    ->
    true ;
    (no(S)
    ->
    fail ;
    ask(S))).
```

即输入是需要验证的事实（这个事实是通过遍历 prolog 规则库提取出的），返回值为布尔类型，表示需要确认是否具有这个事实。

实现如下：（其中 t 为遍历出的作为判断依据的属性）

```
def verify(t):
    choice = input("Is it true? {}(y/n):".format(t))
    if choice == 'y':
        features[t] = 'yes'
        return 1
    elif choice == 'n':
        features[t] = 'no'
        return 0
```

这样就利用了 Python 函数进行了 Prolog 代码的用户 IO 和逻辑判定。

在重新实现的过程中，函数可以通过一个字典（dict）记录每个回答的结果，这样面对重复的问题就可以不必每次都回答，也可以增加代码的健壮性（以防止同一个问题不同回答导致问题）。代码如下：

```
features = {}
def verify(t):
    if t in features:
        if features[t] == 'yes':
            return 1
        elif features[t] == 'no':
            return 0
    choice = input("Is it true? {}(y/n):".format(t))
    if choice == 'y':
        features[t] = 'yes'
        return 1
    elif choice == 'n':
```

```
        features[t] = 'no'
        return 0
```

整个程序与 prolog 的关联按照GitHub 上的 API 说明翻版如下：

```
verify.arity = 1
registerForeign(verify)
prolog = Prolog()
prolog.consult('animal.pl') # 导入 Prolog 的代码（规则库）
for result in prolog.query('hypothesize(X)'):
    print('It is', result["X"])
```

每次都按照 animals.pl 文件中的逻辑顺序进行进行判断。完整代码见A.4

A 附录代码

A.1 I^2C 总线蜂鸣器

```
#!/usr/bin/python3
import smbus
import RPi.GPIO as GPIO


buzz_flag=0

def MyInterrupt(ch):
    global buzz_flag
    buzz_flag = not buzz_flag
    print("KEY PRESS")

class buzz:
    def __init__(self):
        self.address = 0x20
        self.bus = smbus.SMBus(1)

    def buzz_on(self):
        self.bus.write_byte(self.address,0x7F)

    def buzz_off(self):
        self.bus.write_byte(self.address,0xFF)

def main():
    global buzz_flag

    # setup the gpio
    KEY = 20
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(KEY, GPIO.IN, GPIO.PUD_UP)
    GPIO.add_event_detect(KEY, GPIO.FALLING, MyInterrupt, 200)

    mybuzz = buzz()

    while True:
        if buzz_flag:
            mybuzz.buzz_on()
```

```
38          else:
39              mybuzz.buzz_off()
40
41  #          time.sleep(1)
42
43  if __name__=="__main__":
44      try:
45          main()
46      except KeyboardInterrupt:
47          print('exit')
```

A.2 RTC 时间读取

```
1   import smbus
2   import time # 包含相关库文件
3   import datetime as dt
4   address = 0x68
5   register = 0x00
6   bus = smbus.SMBus(1) # 初始化 i2c Bus
7
8   def hex2dec(num):
9       return num // 10 * 16 + num % 10
10  def dec2hex(num):
11      return num // 16 * 10 + num % 16
12
13  # FixTime 定义为 2019 年 6 月 12 日 18 时
14  t = str(dt.datetime.now())
15  print(t)
16
17  year = hex2dec(int(t[2:4]))
18  month = hex2dec(int(t[5:7]))
19  day = hex2dec(int(t[8:10]))
20  hour = hex2dec(int(t[11:13]))
21  minute = hex2dec(int(t[14:16]))
22  second = hex2dec(int(t[17:19]))
23
24  FixTime = [second, minute, hour, 0x04, day, month, year]
25  # FixTime=[0x00,0x00,0x18,0x03,0x12,0x06,0x19]
26
27  print(FixTime)
28  # 定义时钟操作函数
```

```
def ds3231SetTime():
    bus.write_i2c_block_data(address,register,FixTime)
def ds3231ReadTime():
    return bus.read_i2c_block_data(address,register,7);
ds3231SetTime() # 设置时间
print(FixTime)
FixTime = ds3231ReadTime() # 读出时间
print(FixTime)

weekday = ['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']
try:
    while True:
        t = ds3231ReadTime()
        print('%d-%d-%.2d %s %.2d:%.2d:%.2d' % \
        (2000+dec2hex(t[6]),dec2hex(t[5]), dec2hex(t[4]), \
        weekday[t[3]],dec2hex(t[2]), dec2hex(t[1]), dec2hex(t[0])))
        time.sleep(1)
except KeyboardInterrupt:
    pass
```

A.3 专家系统规则库

```
/* Prolog ver. of the animal identification game (simple expert system) */
/* presented in a Lisp program in Chapter 6 of Winston and Horn (1985). */

/* hypotheses to be tested */
hypothesize(ostrich)   :- ostrich, !.
hypothesize(tiger)     :- tiger, !.
hypothesize(cheetah)   :- cheetah, !.
hypothesize(giraffe)   :- giraffe, !.
hypothesize(zebra)     :- zebra, !.
hypothesize(penguin)   :- penguin, !.
hypothesize(albatross) :- albatross, !.
hypothesize(unknown).             /* no diagnosis */

/* animal identification rules */
tiger :- mammal,
         carnivore,
         verify(has_tawny_color),
         verify(has_black_stripes).
cheetah :- mammal,
```

```
           carnivore,
           verify(has_tawny_color),
           verify(has_dark_spots).
giraffe :- ungulate,
           verify(has_long_neck),
           verify(has_long_legs).
zebra :- ungulate,
         verify(has_black_stripes).

ostrich :- bird,
           verify(does_not_fly),
           verify(has_long_neck).
penguin :- bird,
           verify(does_not_fly),
           verify(swims),
           verify(is_black_and_white).
albatross :- bird,
             verify(appears_in_story_Ancient_Mariner),
             verify(flys_well).

/* classification rules */
mammal    :- verify(has_hair), !.
mammal    :- verify(gives_milk).
bird      :- verify(has_feathers), !.
bird      :- verify(flys),
             verify(lays_eggs).
carnivore :- verify(eats_meat), !.
carnivore :- verify(has_pointed_teeth),
             verify(has_claws),
             verify(has_forward_eyes).
ungulate :- mammal,
            verify(has_hooves), !.
ungulate :- mammal,
            verify(chews_cud).
```

A.4 动物识别程序的 verify 函数

```
from pyswip import Prolog, registerForeign # 导入模块

features = {}
def verify(t):
```

```
    if t in features:
        if features[t] == 'yes':
            return 1
        elif features[t] == 'no':
            return 0
    choice = input("Is it true? {}(y/n):".format(t))
    if choice == 'y':
        features[t] = 'yes'
        return 1
    elif choice == 'n':
        features[t] = 'no'
        return 0

verify.arity = 1
registerForeign(verify)
prolog = Prolog()
prolog.consult('animal.pl') # 导入 Prolog 的代码（规则库）
for result in prolog.query('hypothesize(X)'):
    print('It is', result["X"])
```