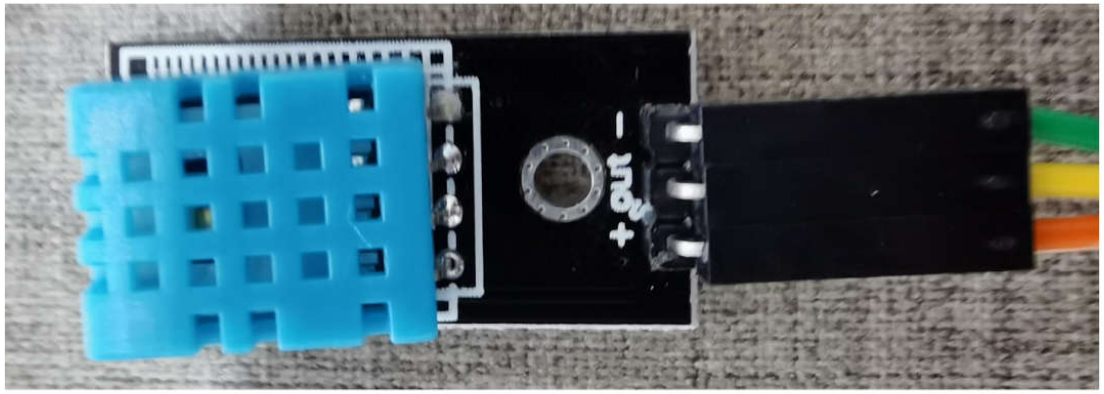
智能系统与控制

树莓派：温度湿度-DHT11



于泓
鲁东大学
信息与电气工程学院
2021.10.22

数字温度湿度传感器 DHT11 是一种复合传感器，包含温度和湿度的校准数字信号输出。采用专用的数字模块采集技术和温度湿度传感技术，确保产品具有高可靠性和稳定性。

该传感器包含一个电阻湿感元件和一个 NTC 温度测量设备，与内置的高性能的 8 位微控制器连接。

3 个引脚 +、 -分别接 VCC 与 GND。中间引脚 out 是一条数据线，通过它可以向传感器发送应答信号并返回 40 位的温度湿度数据。

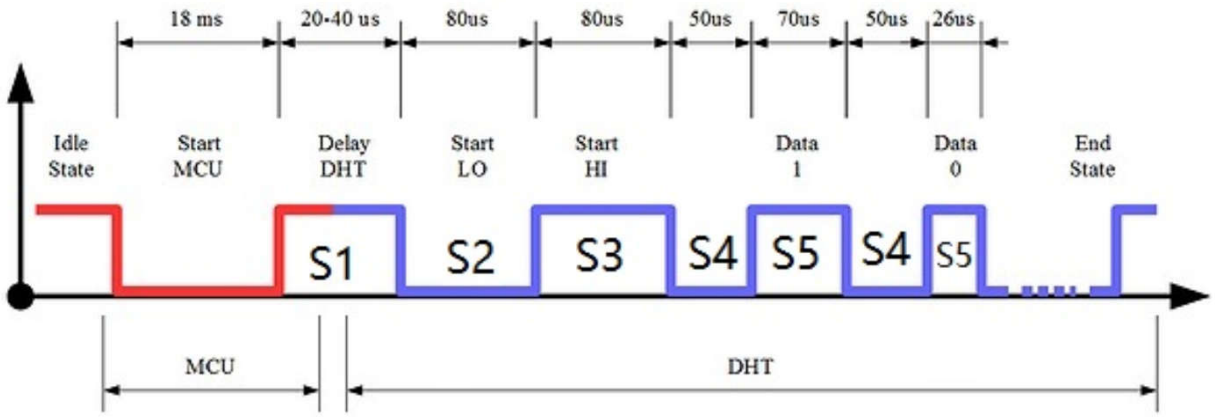
DHT11 的工作时序如图 所示。

1、首先由树莓派将数据线拉高进入**空闲状态（IdleState），**

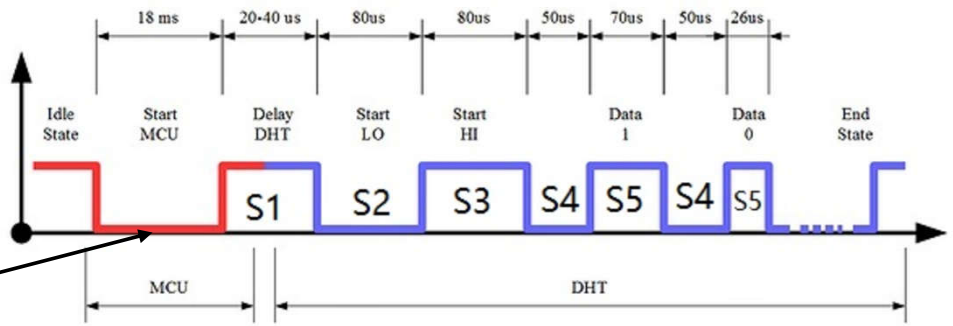
2、然后再把**数据线拉低**至少 18ms 通知 DHT11 需要进行数据采集（Start MCU），
然后放弃总线的控制权。

3、随后数据线会被 **DHT11 拉高（20-40us），**然后 DHT11 将发送**一个 80us 的低电平与 80us 的高电平**数据开始信号通知树莓派准备接受数据。

4、随后将发送 40位的 0,1 脉冲信号，
其中 **0 脉冲包括 50us 低电平， 26us 高电平**；**而 1 脉冲由 50us 低电平与 70us 的高电平组成。**
数据发送完毕之后数据总线被长时间拉高，总线又进入空闲模式。

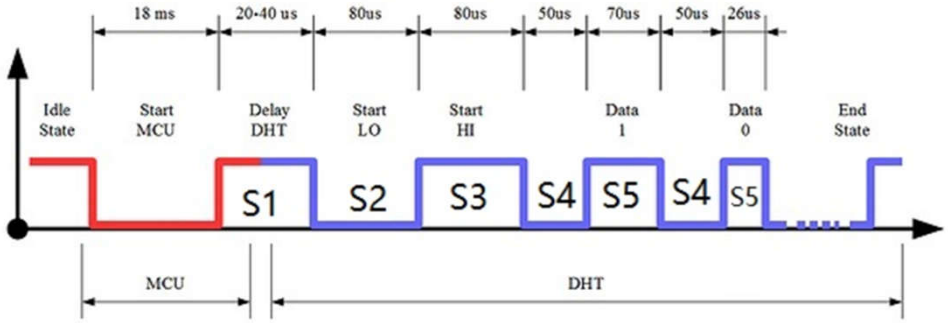


```python
class DHT11(object):

    def __init__(self,pin_D):
        self._pin = pin_D

    # 读取温度、湿度
    def read_DHT(self):

        # 设置为输出引脚
        GPIO. setup (self._pin , GPIO.OUT)

        # 输出高电平
        GPIO.output(self._pin, GPIO.HIGH)
        time.sleep(0.05)

        # 输出低电平
        GPIO.output(self._pin, GPIO.LOW)
        time.sleep(0.02)

        # 放弃总线控制权，设置为输入引脚，上拉模式
        GPIO.setup(self._pin, GPIO.IN, GPIO.PUD_UP)

        # 收集从数据线上传来的数据
        data = self.collect_input()

        # 从收集的信号中获取数据位，高电平持续时间
        high_state_lengths_data = self.get_high_state_lengths_data(data)

        # 数据应当有40位，即有40段高电平 包括4字节数据和一字节校验不对的话退出
        if len(high_state_lengths_data) != 40:
            return False,0

        # 根据上升脉冲的长度计算二进制bit
        bits = self.calculate_bits(high_state_lengths_data)

        # 将二进制bit变换为字节
        the_bytes = self.bits_to_bytes(bits)

        # 进行校验 校验失败 返回错误信息
        checksum = self.calculate_checksum(the_bytes)
```

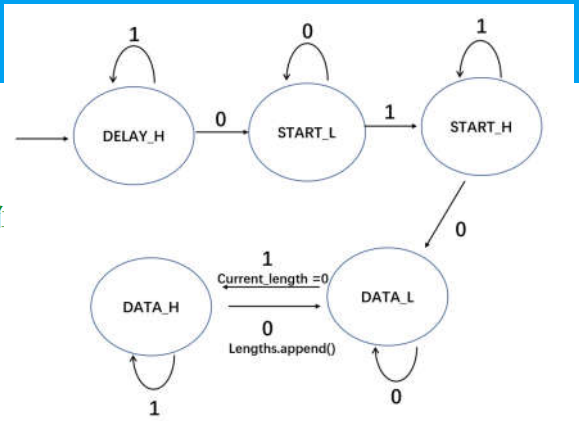


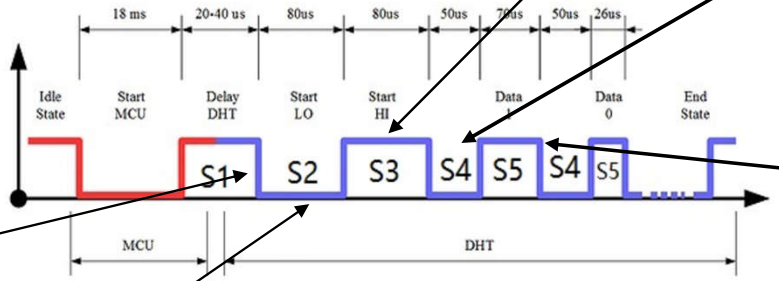

```python
        if the_bytes[4] != checksum:
            return False,0

        # 四个字节0-3 分别是湿度的整数、湿度的小数、温度的整数、温度的小数
        temperature = the_bytes[2] + float(the_bytes[3]) / 10
        humidity = the_bytes[0] + float(the_bytes[1]) / 10

        # 正确接收返回接收正确标志以及计算得到的温度湿度值
        return True, [temperature, humidity]
```

```
def collect_input(self):
    # 记录持续信号时长
    unchanged_count = 0

    # 信号持续的最大长度，用来判断数据传输是否结束
    max_unchanged_count = 100

    last = -1
    data = []
    while True:
        # 不断采集数据
        current = GPIO.input(self._pin)
        data.append(current)

        # 记录信号持续的时间
        # 如果有变化就开启一段新的记录
        if last != current:
            unchanged_count = 0
            last = current
        # 没有变化时长+1
        else:
            unchanged_count += 1
            if unchanged_count > max_unchanged_count:
                break

    return data
```



```
# 从收集的信号中获取数据位高电平的持续时间
def get_high_state_lengths_data(self, data):

    # 设置5个状态 分别是 初始延时 起始位低电平 起始位
    STATE_DELAY_H = 1
    STATE_START_L = 2
    STATE_START_H = 3
    STATE_DATA_L = 4
    STATE_DATA_H = 5

    state = STATE_DELAY_H

    lengths = [] # 记录每个数据周期中高电平持续时间
    current_length = 0 # 记录前一个状态的持续时间

    for i in range(len(data)):
        current = data[i]
        current_length += 1

        # 等待的高电平
        if state == STATE_DELAY_H:
            if current == GPIO.LOW:
                state = STATE_START_L
                continue
            else:
                continue

        # 起始的低电平
        if state == STATE_START_L:
            if current == GPIO.HIGH:
                state = STATE_START_H
                continue
            else:
                continue
```






```
        # 起始的高电平
        if state == STATE_START_H:
            if current == GPIO.LOW:
                state = STATE_DATA_L
                continue
            else:
                continue

        # 数据的低电平
        if state == STATE_DATA_L:
            if current == GPIO.HIGH:
                current_length = 0
                state = STATE_DATA_H
                continue
            else:
                continue

        # 数据的高电平
        if state == STATE_DATA_H:
            if current == GPIO.LOW:
                lengths.append(current_length)
                state = STATE_DATA_L
                continue
            else:
                continue

    return lengths
```

```
# 通过记录的数据高电平的持续时长来进行0、1解码
def calculate_bits(self,high_state_lengths_data):
    # 找到最长和最短的时长
    shortest_pull_up = 1000
    longest_pull_up = 0

    for i in range(0, len(high_state_lengths_data)):
        length = high_state_lengths_data[i]
        if length < shortest_pull_up:
            shortest_pull_up = length
        if length > longest_pull_up:
            longest_pull_up = length

    # 用中间值作为阈值
    halfway = (longest_pull_up + shortest_pull_up) / 2
    bits = []

    # 大于阈值判定为1 ， 否则判定为0
    for i in range(0, len(high_state_lengths_data)):
        bit = False
        if high_state_lengths_data[i] > halfway:
            bit = True
        bits.append(bit)

    return bits
```

```
# 每8个bit一组 将bit转换成字节（byte）
def bits_to_bytes(self, bits):
    the_bytes = []
    byte = 0

    for i in range(0, len(bits)):
        byte = byte << 1
        if (bits[i]):
            byte = byte | 1
        else:
            byte = byte | 0
        if ((i + 1) % 8 == 0):
            the_bytes.append(byte)
            byte = 0

    return the_bytes


# 计算校验值 前四个字节相加取低8位作为校验码
def calculate_checksum(self, the_bytes):
    return the_bytes[0] + the_bytes[1] + the_bytes[2] + the_bytes[3] & 255
```

```
if __name__ == "__main__":
    # 设置引脚及工作方式
    pin_dht=  pin_dic['G16']
    GPIO.setmode(GPIO.BOARD)

    m_DHT11 =  DHT11(pin_dht)


    try:
        while True:
            flag, result = m_DHT11.read_DHT()

            if flag:
                print("温度: %-3.1f C\n" % result[0])
                print("湿度: %-3.1f %% \n" % result[1])
            else:
                print("ERROR")
            time.sleep(2)
    except KeyboardInterrupt:
        print('\n Ctrl + C QUIT')
    finally:
        GPIO.cleanup()
```