

义务教育信息科技教学指南

硬件编程说明

2024 年 8 月

一、引言

为促进义务教育信息科技课程开设，通过程序验证发展全体学生数字素养与技能。义务教育信息科技教学指南编写组对涉及硬件编程的内容给出了一种建议程序供参考。其目的在于加强不同厂商生产的主控板之间的兼容性，简化教育资源的开发及使用过程，并提高学习效率。遵循以下原则：

- 简洁性原则：强调代码的简洁性，倡导尽可能缩短代码长度，以简化编程教学和学习过程。
- 可读性原则：确保代码易于理解，与教学资源紧密结合，明确地展现其功能和实现方式，以便教师和学生能够轻松掌握。
- 统一性原则：对于功能相似的硬件调用，应统一格式和接口，以降低学习和应用的复杂度，保证教学效果的一致性。
- 经济性原则：鼓励使用成本效益高的硬件，以促进技术的广泛普及和应用。

二、适用范围

本程序说明主要供教学参考。需要可编程主控板须支持运行 **MicroPython** 脚本程序。希望有更多的主控板在固件中支持 **educore** 模块。

三、术语解释

本说明中使用的专业术语和定义如下：

可编程主控板：指一种集成了微处理器、内存和输入/输出接口的电路板，用户可以通过编程控制其行为。

兼容性：不同厂商生产的主控板使用相同的或者兼容的编程代码执行相同功能。

互操作性：不同厂商生产的主控板能够在同一教学案例中互相通信和协同工作。

四、代码解释

定义了 **educore** 模块，适用于教学中常见的传感器、执行器等设备。

1. 引脚 IO 控制

(1) 读取引脚电平

```
from educore import pin #从 educore 模块导入 pin 类
p0=pin(0) #实例化引脚 0
d=p0.read_digital() #读取引脚电平值，0 为低电平，1 为高电平
```

注：该方法用于读取输出开关量信号的各类传感器数据。如按键传感器、触摸传感器、红外热释电传感器等。

(2) 读取引脚模拟值

```
from educore import pin
p1=pin(1) #实例化引脚 1
v=p1.read_analog() #读取引脚模拟值，采样精度为 10 位，返回值范围为 0-4095
```

注：对于不具备 ADC 功能的引脚，返回值为 0 或者 4095，其中 0 表示低电平，4095 表示高电平

注：该方法用于读取输出模拟量信号的各类传感器数据。如光敏传感器、声音传感器、可调电位器、气敏传感器等。

(3) 输出引脚电平

```
from educore import pin
p0=pin(0)
p0.write_digital(value=0) #设置 p0 引脚输出为低电平
p0.wiite_digital(value=1) #设置 p0 引脚输出为高电平
```

注：调用 write_digital 方法时，需主动配置引脚为内部强上拉模式(如果硬件支持)

(4) 输出引脚模拟值(PWM 输出)

```
from educore import pin
```

```
p1=pin(1)
```

```
p1.write_analog(value=511[,freq=5000]) #设置 p1 的模拟输出值为 511
```

注：模拟值范围为 0-1023，freq 为可选参数，默认为 5000Hz

(5) 引脚事件

引脚事件包含 event_rising(上升沿)，event_falling(下降沿)。

```
from educore import pin
```

```
def myfunction():
```

```
#自定义函数代码
```

```
p1=pin(1)
```

```
p1.event_rising=myfunction #当 p1 引脚上升沿时回调 myfunction 函数
```

注：某些硬件不支持同时设置上升沿和下降沿事件，此情况下以最后一次绑定事件为准

2. 执行器控制

(1) 显示屏控制

a. 显示屏初始化

```
from educore import oled
```

```
oled.init(sda=0,scl=1) #通过 IIC 总线连接外部 OLED 屏时，需要通过该语句初始化
```

```
oled.init(sda=pin(0),scl=pin(1)) #等价于 oled.init(sda=0,scl=1)
```

注：若使用板载显示屏，则无须初始化语句

b. 显示字符

```
oled.print("String") #在显示屏显示字符串 “String”
```

```
oled.print("Str\ning") #在显示屏显示字符串，并支持换行符
```

注：使用 print 方法时，会先清除原屏幕中显示的内容。在显示字符串时支持自动换行（超出显示屏显示范围时将截断字符串）。此外，厂商可自定义内

置表情图像，自定义表情图像显示时，使用 \: 实现转义，如 `oled.print("\:HAPPY")` 可表示为显示一个笑脸图像，`oled.print("\:SAD")` 表示为显示哭脸图像。

c. 屏幕内容清除

```
oled.clear() #清除屏幕显示内容
```

(2) 蜂鸣器控制

a. 蜂鸣器初始化

```
from educore import speaker  
s1=speaker(1) #将蜂鸣器连接到 1 号引脚  
s1=speaker(pin(1)) #等价于 s1=speaker(1)  
s1=speaker() #调用板载蜂鸣器
```

b. 播放声音

```
s1.tone(freq,dur)  
freq:播放频率  
dur:持续时间（毫秒）
```

注：当 `dur` 参数为空时，蜂鸣器将持续播放该频率声音，直到通过 `stop` 方法停止声音。

对于有 DAC 硬件支持的主控板，建议支持和弦声音的播放：

```
s1.tone(freq=[2000,3000],dur=1000) #播放由 2000Hz、3000Hz 正弦波叠加和弦声音
```

注：考虑到语句兼容性，即便主控板不支持和弦声音播放，也要支持 `freq` 参数为列表类型。此时可取 `freq[0]` 作为实际播放声音频率。

c. 停止播放声音

```
s1.stop()
```

(3) 电机控制

a.电机初始化

```
from educore import parrot
```

```
M1=parrot(in0=0,in1=1) #将电机控制线连接到 0 号和 1 号引脚
```

注：in0 和 in1 引脚用于控制电机方向，当 in0 引脚给 0，in1 引脚给 1 时，电机正转；当 in0 引脚给 1，in1 引脚给 0 时，电机反转。

```
M1=parrot(parrot.M1) #初始化板载电机接口 M1
```

注：当主控板有两个电机接口时，编号分别为 M1、M2。在程序实现时，可通过判断传参类型来确定调用的是板载电机接口还是外部电机接口。

b.电机转速控制

```
M1.speed(speed)
```

speed 参数：用于控制 PWM 值进行调速。speed 参数范围:-100-100，其中 0 表示停止，负数表示反转，绝对值越大转速越快。

(4) 舵机控制

a.舵机初始化

```
from educore import servo
```

```
s1=servo(1) #将舵机连接到 1 号引脚
```

```
s1=servo(pin(1)) #等价于 s1=servo(1)
```

b.舵机输出角度

```
s1.angle(value=180) #设置舵机转动到 180° 位置,角度范围为:0~180°
```

注：为了传值错误导致程序报错，需对参数 value 进行判断，超过 180 时修正为 180，小于 0 时修正为 0。

(5) 步进电机（编码电机）控制

a.步进电机初始化

```
from educore import stepper
```

```
sp=stepper(sda=0,scl=1) #将电机控制线连接到 0 号和 1 号引脚
```

```
sp=stepper(sda=pin(0),scl=pin(1)) #等价于 sp=stepper(sda=0,scl=1)
```

注：

1.考虑到程序非阻塞运行，建议在步进电机模块中内置下位机。并通过总线方式进行通信。

2.该方法同样适用于可精确控制与获取转速、位置等参数的编码电机、伺服电机。

b.步进电机转速控制

```
sp.speed(speed)
```

用于精确控制步进电机转速。

speed 参数绝对值表示转速，单位为 rpm（转每分钟），正负符号表示转动方向。speed=0 表示电机停止转动。

c.步进电机转动控制

```
sp.goto(r,speed,wait=True)
```

用于控制步进电机以何速度转动多少圈。

参数 r 为需要转动的圈数(支持浮点数类型)。当参数为正数时电机正转，参数为负数时电机反转。

参数 speed 表示转速(rpm)。当参数为正数时电机正转，参数为负数时电机反转。

参数 wait=True 时表示该方法为阻塞方法，wait=False 时该方法为非阻塞方法。

示例如下。

sp.goto(3.5,100)和 sp.goto(-3.5,-100)都表示以 100rpm 的速度顺时针转动 3.5 圈。

sp.goto(-3.5,100)、sp.goto(3.5,-100)都表示以 100rpm 的速度逆时针转动 3.5 圈。

d.步进电机停止转动

```
sp.stop()
```

无条件让步进电机立即停止转动

e.查询步进电机状态

```
sp.status()
```

返回步进电机的状态，返回状态为包含 ready 和 steps 状态的列表类型数据。

如: [ready,angle]

ready:当前步进电机的运转状态，True 表示当前正在转动，False 表示当前停转。

angle:当前步进电机所处角度位置，支持浮点数类型。（在电机上电时 angle 清零），如 angle=540.0，表示该步进电机当前位置与上电后顺时针旋转了 1.5 圈时一致。

（6）RGB 灯带控制

a.RGB 灯带初始化

```
from educore import rgb
```

```
pxs=rgb() #初始化板载 RGB 灯（若主控板支持）
```

```
pxs=rgb(1) #将灯带连接到 1 号引脚
```

```
pxs=rgb(pin(1)) #等价于 pxs=rgb(1)
```

b.控制 RGB 灯带

```
pxs.write(index=[0,1,2],r,g,b) #让 0、1、2 号灯珠根据 r、g、b 值显示色彩
```

注：rgb 颜色分量值范围为 0-255

```
pxs.clear() #熄灭所有灯珠
```


3. 传感器读取

(1) 声音传感器

a. 声音传感器初始化

```
from educore import sound
```

```
s=sound() #初始化板载声音传感器
```

```
s=sound(0) #初始化连接在 0 号口(支持 ADC 的引脚)上的声音传感器
```

```
s=sound(pin(0)) #等价于 s=sound(0)
```

b. 读取声音传感器值

```
v=s.read() #读取声音传感器值，返回值范围为 0-4095
```

(2) 光敏传感器

a. 光敏传感器初始化

```
from educore import light
```

```
lgt=light() #初始化板载光敏传感器
```

```
lgt=light(0) #初始化连接在 0 号口(支持 ADC 的引脚)上的光敏传感器
```

```
lgt=light(pin(0)) #等价于 lgt=light(0)
```

b. 读取光敏传感器值

```
v=lgt.read() #读取光敏传感器值，返回值范围为 0-4095
```

(3) 按键传感器

a. 按键传感器初始化

```
from educore import button
```

```
bt=button(0) #初始化连接在 0 号口上的按键传感器
```

```
bt=button(pin(0)) #等价于 bt=button(0)
```

若主控板上本身板载了按键传感器或触摸传感器，则可以通过以下语句初始化：

```
bt=button(button.a) #初始化板载按键传感器 A
```

```
bt=button(button.b) #初始化板载按键传感器 B
```

b.读取按键状态

```
bt.status() #返回按键 A 的状态，0 表示释放，1 表示按下
```

c.按键事件

```
def myfunction():
```

```
#自定义函数代码
```

```
button_a.event_pressed =myfunction #绑定按键事件，当按键 A 被按下时回调 myfunction 函数。
```

注：在代码中，按键传感器也可以通过引脚 IO 控制实现。

（4）加速度传感器

a.加速度传感器初始化

```
from educore import accelerometer
```

```
acc=accelerometer()#初始化板载加速度传感器
```

```
acc=accelerometer(sda=0,scl=1) #初始化外部加速度传感器
```

```
acc=accelerometer(sda=pin(0),scl=pin(1)) #等价于
```

```
acc=accelerometer(sda=0,scl=1)
```

b.读取加速度传感器值

```
acc.X() #获得 x 轴方向（板子横向）加速度值
```

```
acc.Y() #获得 y 轴方向（板子纵向）加速度值
```

```
acc.Z() #获得 z 轴方向（垂直板子）加速度值
```

```
acc.shake() #若检测到摇晃，则返回 True，否则返回 False
```

(5) DHT 温湿度传感器

a. 温湿度传感器初始化

```
from educore import dht  
  
dhtsensor=dht(0)#初始化连接在 0 号口上的温湿度传感器  
  
dhtsensor=dht(pin(0)) #等价于 dhtsensor=dht(0)
```

注：常见的数字温湿度传感器有 dht11 和 dht22 模块。主控板厂商可适配自己的模块型号。有些主控板会自适应 dht11 和 dht22 两种型号的模块。

b. 读取温湿度值

```
temp,hum=dhtsensor.read() #读取摄氏温度值和湿度百分比  
也可以通过以下代码直接读取温湿度值：  
  
temp,hum=dht(0).read() #读取连接在 0 号口的温湿度传感器值  
  
temp,hum=dht(pin(0)).read() #等价于 temp,hum=dht(0).read()
```

(6) DS18B20 温度传感器

a. 温度传感器初始化

```
from educore import ds18b20  
  
ds=ds18b20(0)#初始化连接在 0 号口上的温度传感器  
  
ds=ds18b20(pin(0)) #等价于 ds=ds18b20(0)
```

b. 读取温度值

```
temp=ds.read() #读取摄氏温度值  
也可以通过以下代码直接读取温度值：  
  
temp=ds18b20(0).read() #读取连接在 0 号口上 DS18B20 温度传感器数据  
  
temp=ds18b20(pin(0)).read()#等价于 temp=ds18b20(0).read()
```

(7) 超声波测距传感器

a.超声波传感器初始化

```
from educore import ultrasonic
```

```
sonic=ultrasonic(trig=1,echo=0) #连接 trig 和 echo 引脚
```

```
sonic=ultrasonic(trig=pin(1),echo=pin(0))
```

```
#等价于 sonic=ultrasonic(trig=1,echo=0)
```

注：对于使用 IIC 总线的超声波传感器，可通过以下方式初始化：

```
sonic=ultrasonic(sda=1,scl=0) #使用 IIC 接口连接超声波传感器
```

b.读取超声波传感器测距值

```
sonic.distance() #返回距离，单位 cm，数据类型为整数型
```

(8) RFID 读卡器模块

a.读卡器初始化

```
from educore import rfid
```

```
scan_rfid=rfid(sda=0,scl=1) #使用 IIC 接口连接读卡器
```

```
scan_rfid=rfid(sda=pin(0),scl=pin(1))
```

```
#等价于 scan_rfid=rfid(sda=0,scl=1)
```

b.尝试读卡

```
rf=scan_rfid.scanning([wait=True]) #尝试读取并返回电子标签信息
```

注：该方法为阻塞方法。其中 wait 参数为可选参数，默认为 True。该参数表示等待刷卡时间，当 wait=True 时，会一直等待刷卡；wait 参数也可接受整数型数据，当传入整数型数据时表示等待 wait 秒后结束阻塞。无论是否成功刷卡，该函数都应该返回一个 rf 对象。

c.获取电子标签序列号（编号）

```
rf.serial_number() #获取电子标签编号，若读取不成功则该方法返回 None
```

注：

- 1.除非再次读取，否则 `rf` 中的读取结果不变。
- 2.返回的电子标签编号的数据类型为字符串类型。

完整示例：

```
scan_rfid = rfid(sda=pin(0),scl=pin(1))  
  
while True:  
  
    rf = scan_rfid.scanning()  
  
    sn= rf.serial_number()  
  
    oled.print(rf.serial_number())
```

(9) 红外热释电传感器

a.红外热释电传感器初始化

```
from educore import tsd  
  
dev=tsd(0) ##初始化连接在 0 号口上的红外热释电传感器  
  
dev=tsd(pin(0))#等价于 dev=tsd(0)
```

b.读取红外热释电传感器值

```
dev.read() #返回传感器值，0 为无人，1 为有人
```

注：因为人体热释电传感器一般输出高低电平信号，因此也可以通过引脚 IO 控制实现读取。

10) 大气压强传感器

a.大气压强传感器初始化

```
from educore import pressure  
  
ps=pressure() #初始化板载大气压强传感器（若主控板支持）  
  
ps=pressure(sda=0,scl=1) #通过 IIC 总线连接外部大气压强传感器
```

b.读取大气压强

```
ps.read() #返回大气压强浮点数值，单位为百帕
```

(11) 磁力传感器

a.磁力传感器初始化

```
from educore import compass
```

```
cp=compass() #初始化板载指南针传感器（若主控板支持）
```

```
cp=compass(sda=0,scl=1) #通过 IIC 总线连接外部指南针传感器
```

b.磁力模块校准

```
cp.adjust() #执行指南针校准程序
```

c.读取方向值

```
cp.direction() #返回方向角数据，范围为 0-359
```

(12) 称重传感器

a.称重传感器初始化

```
from educore import force
```

```
fs=force(sda=0,scl=1) #使用 IIC 接口连接拉力传感器
```

b.称重传感器归零

```
fs.zero() #使拉力传感器归零
```

c.读取值

```
fs.read(mass=True) #返回质量值，单位为克。
```

```
#默认返回质量，当参数 mass=False 时，返回力的单位 N（牛顿）
```

4. 物联通信相关

(1) 蓝牙通信（模拟 HID 设备）

a.设备初始化

```
from educore import hid
```

```
ble_hid=hid(name='mpy_hid') #初始化蓝牙 HID 设备并命名为 mpy_hid
```

b.设备连接状态

```
ble_hid.isconnected() #判断是否已经连接， True 为连接， False 为未连接
```

注：该方法为非阻塞运行

c.模拟鼠标键盘

```
from educore import keycode
```

```
ble_hid.keyboard_send (keycode.SPACE) #模拟键盘按下空格
```

```
ble_hid.keyboard_send([keycode.CTRL,keycode.a]) #按下组合键 CTRL+a
```

```
ble_hid.mouse_key(keycode.CLICK) #鼠标单击
```

```
ble_hid.mouse_key(keycode.DCLICK) #鼠标双击
```

注：keycode 定义：

按键或操作	keycode	示例
大写字母	A、 B、 C、 DZ	keycode.A
小写字母	a、 b、 c、 d... .. z	keycode.a
数字字符	_1、 _2、 _3... .._9	keycode._9(不能省略下划线)
空格	SPACE	keycode.SPACE
回车换行	ENTER	keycode.ENTER
CONTROL	CTRL	keycode.CTRL
SHIFT	SHIFT	keycode.SHIFT
ALT	ALT	keycode.ALT
方向按键	LEFT、 RIGHT、 UP、 DOWN	keycode.LEFT
鼠标单击	CLICK	keycode.CLICK
鼠标双击	DCLICK	keycode.DCLICK

(2) 连接 Wi-Fi

```
from educore import wifi
```

```
wifi.connect(ssid="mywifi",psd="xxx"[,timeout=10000]) #尝试连接 wifi 网络。
```

可以不写参数名称。timeout 为可选参数，表示连接超时时长，默认超时时间为 10000 毫秒，即 10 秒。

注：该方法为阻塞运行，若连接超时则直接报错中断运行。

```
wifi.status() #返回网络连接状态，True 表示已连接，False 表示未连接
```

```
wifi.info() #返回包含当前 IP 地址、子网掩码、网关等信息的字符串
```

(3) 使用 MQTT 协议

a.连接 MQTT 平台

```
from educore import mqttclient
```

```
mqttclient.connect(server= "xxxxxxxxxxxxx",#MQTT 服务器域名或 IP
```

```
port=1883, #IOT 服务器端口
```

```
client_id="设备 ID", #设备 Client_id (根据需要)
```

```
user= "用户名", #设备接入用户名 (根据需要)
```

```
psd= "密码") #设备接入密码 (根据需要)
```

注：该方法为阻塞运行，默认超时时间为 3 秒

b.判断 MQTT 连接状态

```
mqttclient.connected() #返回连接状态，True 表示连接成功，False 表示未连接成功
```

c.发布 MQTT 消息

```
mqttclient.publish(topic='xxxxxx', #向对应主题发送消息
```

```
content= 'xxxxxx')#content 为发送内容
```

d.获取主题消息

`msg=mqttclient.message(topic='xxxxxx')` #获取对应主题接收的消息，若该主题无消息，则返回 None

e.主题消息订阅回调

`mqttclient.received (topic='xxxxxx', #对应主题收到消息时，回调函数
callback=receivedfunction)` #通过 callback 指定回调函数

注：通过以下方法实现对应主题消息的接收与回调：

`def receivedfunction():`

`msg=mqttclient.message(topic='mytopic')`

`oled.print(msg)`

`mqttclient.received (topic='mytopic',callback=receivedfunction)`

5. 人工智能调用

(1) 人工智能摄像头模块

a.摄像头模块初始化

`from educore import smartcamera`

`camera=smartcamera()` #连接板载人工智能摄像头

`camera=smartcamera(tx=16,rx=15)` #通过串口连接人工智能摄像头模块

`camera=smartcamera(tx=pin(16),rx=pin(15))`

#等价于 `camera=smartcamera(tx=16,rx=15)`

b.加载模型

`camera.init("FACE_RECOGNIZE")` #加载人脸识别模型

`camera.init("FACE_DETECT")` #加载人脸检测模型

注：对于内置人工智能摄像头的主控板（本身具备模型推理能力，可对加载模型过程进行封装。如加载人脸识别模型时，可调用人脸录入过程来录入人脸；对于外部人工智能摄像头模块，其使用过程中由摄像头模块独立完成数据采集、推理等工作，并将推理结果通过串口发送给主控板。

c.模型推理调用

camera.result() #获取模型推理结果

注：模型推理的返回结果是一个字典。不同的加载模型其返回的推理结果所包含的字段不同。针对于人脸识别模型约定所需返回的结果为：

{"id":1, #当前识别到的人脸（多个人脸结果返回人脸面积最大者）id

"label":"xxx", #当前人脸 id 对应的标签名称

"similarity":0.726, #置信度

"status": 1, #识别状态，1 表示识别成功，0 表示失败

"blink":0, #5 秒内眨眼次数

"mouth_open":0 } #5 秒内张嘴次数

status 的规则为当识别到人脸库中的人脸且置信度大于 0.6 时为 1，否则为 0。即当 status 为 0 时，其他参数无意义。

（2）网页版人工智能摄像头调用

a.连接摄像头

from educore import webcamera

webcamera.connect(id="xxxxxx") #连接网络摄像头

注：id 为网页摄像头提供的 id 号，关联网页与主控板之间的关联

b.模块推理调用

webcamera.result() #获取推理结果

注 1：模型推理的返回结果是一个字典。不同的加载模型其返回的推理结果所包含的字段不同。针对于人脸识别模型约定所需返回的结果数据格式与人工智能摄像头模块返回结果格式一致：

{"id":1, #当前识别到的人脸（多个人脸结果返回人脸面积最大者）id

"label":"xxx", #当前人脸 id 对应的标签名称

"similarity":0.726, #置信度

"status":1, #识别状态，1 表示识别成功，0 表示失败

"blink":0, #5 秒内眨眼次数

"mouth_open":0} #5 秒内张嘴次数

注 2：网页版摄像头通过浏览器打开网页，并调用电脑上的摄像头来获取图像、通过 WebGL 或 WebGPU 技术来进行模型推理，最后通过网络将识别结果发送给主控板。因此使用网页版摄像头前，要求用户先连接网络。

注 3：人工智能摄像头成本较高，本功能旨在降低成本。建议主控板能适配的网页版人工智能摄像头。一般的，来自网页的推理结果可以通过 MQTT 协议传递给主控板。因此在使用支持外部 MQTT 连接的网页版人工智能摄像头时，也可以通过直接订阅 MQTT 主题消息的方法获得识别结果。

6. 其他辅助功能

(1) 数据格式转换

a. 从文本文件中生成字典

```
from educore import get_dict_from_file
```

```
dic=get_dict_from_file('tushu.txt') #读取文本文件内容并生成字典
```

注：若文本文件内容为：

001 张三

002 李四

则返回的字典为：dic={"001":张三,"002":李四"}

若存在重复键名，则以第一次出现的为准

b. 根据字符串生成字典

```
from educore import get_dict_from_str
```

```
s="001 张三; 002 李四"
```

```
dic=get_dict_from_str(s) #生成字典{"001":张三,"002":李四"}
```

注：字符串中的分号为行分隔符，也可以用\n。即 s="001 张三\n002 李四" 时，运行结果相同。

（2）获取硬件唯一标识

```
from educore import uuid  
  
_id=uuid() #获取设备唯一标识
```

注：该唯一标识可以是设备 mac 地址、芯片 id 等。返回结果为字符串格式，且不应该包含任何空格、冒号等分隔符。

（3）其他所需函数或模块

为了完成物联网相关实验，主控板还需要提供以下 MicroPython 标准模块：

模块名称	用途
time	时间模块
ujson	JSON 解析
urequests	网络请求

五、其他说明

本程序建议仅供教学参考，教师在教学中可根据本地硬件使用情况及学情，选择主控板支持的程序语言及模块。

目前已有部分主控板支持本说明。