Титульный лист Edubot-Shield

import threading

import smbus

import time

\_SHUNT\_OHMS = 0.01 # значение сопротивления шунта на плате EduBot

\_MAX\_EXPECTED\_AMPS = 2.0

try:

import Adafruit\_SSD1306 # sudo pip3 install Adafruit-SSD1306

display = Adafruit\_SSD1306.SSD1306\_128\_64(rst=None) # создаем обект для работы c OLED дисплеем 128х64

except ImportError:

display = None

try:

from ina219 import INA219 # sudo pip3 install pi-ina219

ina = INA219(\_SHUNT\_OHMS, \_MAX\_EXPECTED\_AMPS) # создаем обект для работы с INA219

ina.configure(INA219.RANGE\_16V) # конфигурируем INA219

except ImportError:

ina = None

class Registers:

""" Класс, хранящий регистры драйвера моторов """

REG\_WHY\_IAM = 0x00 # регистр, возвращающий 42

REG\_ONLINE = 0x01

REG\_SERVO\_0 = 0x02

REG\_SERVO\_1 = 0x03

REG\_MOTOR\_MODE = 0x04 # режимы работы

REG\_KP = 0x05 # пропорциональный коэффициент

REG\_KI = 0x06 # интегральный коэффициент

REG\_KD = 0x07 # дифференциальный коэффициент

REG\_INT\_SUMM = 0x08 # предел интегральной суммы

REG\_PID\_PERIOD = 0x09 #

REG\_PARROT\_0 = 0x0A # скорость вращения мотора А в попугаях в режиме WORK\_MODE\_PID\_I2C

REG\_PARROT\_1 = 0x0B # скорость вращения мотора А в попугаях в режиме WORK\_MODE\_PID\_I2C

REG\_DIR\_0 = 0x0C # направление вращения мотора A

REG\_PWM\_0 = 0x0D # ШИМ задаваемый мотору А в режиме WORK\_MODE\_PWM\_I2C

REG\_DIR\_1 = 0x0E # направление вращения мотора B

REG\_PWM\_1 = 0x0F # ШИМ задаваемый мотору B в режиме WORK\_MODE\_PWM\_I2C

REG\_RESET\_ALL\_MOTOR = 0x10 # сброс всех внутренних параметров

REG\_BEEP = 0x11

REG\_BUTTON = 0x12

class MotorMode:

""" Класс, хранящий режимы работы драйвера моторов """

MOTOR\_MODE\_PWM = 0x00 # режим работы - напрямую от ШИМ вилки на плате

MOTOR\_MODE\_PID = 0x01 # режим работы - от ШИМ вилки на плате через ПИД-регулятор

class Direction:

""" Класс, хранящий возможные направления """

FORWARD = 0x00 # вперед

BACKWARD = 0x01 # назад

class EduBot:

""" Класс работы с шилдом едубота """

def \_\_init\_\_(self, bus, addr=0x27):

self.\_bus = bus # шина i2c

self.\_addr = addr # адресс устройства

self.online = False # флаг, определяющий шлется онлайн метка или нет

self.\_\_exit = False # метка выхода из потоков

def whoIam(self):

""" Должен вернуть 42 """

return self.\_bus.read\_byte\_data(self.\_addr, Registers.REG\_WHY\_IAM)

def setMotorMode(self, mode):

""" Устанавливает режим работы драйвера """

self.\_bus.write\_byte\_data(self.\_addr, Registers.REG\_MOTOR\_MODE, mode)

def \_setDirection0(self, direction):

""" Устанавливает направление вращения мотора 0 """

self.\_bus.write\_byte\_data(self.\_addr, Registers.REG\_DIR\_0, direction)

def \_setDirection1(self, direction):

""" Устанавливает направление вращения мотора 1 """

self.\_bus.write\_byte\_data(self.\_addr, Registers.REG\_DIR\_1, direction)

def setParrot0(self, parrot):

""" Устанавливает скорость вращение мотора 0 в попугаях """

parrot = min(max(-100, parrot), 100) # проверяем значение parrot

if parrot < 0:

self.\_setDirection0(Direction.FORWARD)

else:

self.\_setDirection0(Direction.BACKWARD)

self.\_bus.write\_byte\_data(self.\_addr, Registers.REG\_PARROT\_0, abs(parrot))

def setParrot1(self, parrot):

""" Устанавливает скорость вращение мотора 1 в попугаях """

parrot = min(max(-100, parrot), 100) # проверяем значение parrot

if parrot < 0:

self.\_setDirection1(Direction.FORWARD)

else:

self.\_setDirection1(Direction.BACKWARD)

self.\_bus.write\_byte\_data(self.\_addr, Registers.REG\_PARROT\_1, abs(parrot))

def setPwm0(self, direction, pwm):

""" Устанавливает скорость через параметры шима """

pwm = min(max(-255, pwm), 255) # проверяем значение pwm

self.\_setDirection0(direction)

self.\_bus.write\_byte\_data(self.\_addr, Registers.REG\_PWM\_0, abs(pwm))

def setPwm1(self, direction, pwm):

""" Устанавливает скорость через параметры шима """

pwm = min(max(-255, pwm), 255) # проверяем значение pwm

self.\_setDirection1(direction)

self.\_bus.write\_byte\_data(self.\_addr, Registers.REG\_PWM\_1, abs(pwm))

def setKp(self, kp):

""" Устанавливает пропорциональный коэффициент регулятора """

self.\_bus.write\_byte\_data(self.\_addr, Registers.REG\_KP, abs(int(kp \* 10)))

def setKi(self, ki):

""" Устанавливает интегральный коэффициент регулятора """

self.\_bus.write\_byte\_data(self.\_addr, Registers.REG\_KI, abs(int(ki \* 10)))

def setKd(self, kd):

""" Устанавливает дифференциальный коэффициент регулятора """

self.\_bus.write\_byte\_data(self.\_addr, Registers.REG\_KD, abs(int(kd \* 10)))

def setServo0(self, pos):

""" Установка позиции 0 сервы """

pos = min(max(0, pos), 250) # проверяем значение pos

self.\_bus.write\_byte\_data(self.\_addr, Registers.REG\_SERVO\_0, pos)

def setServo1(self, pos):

""" Установка позиции 1 сервы """

pos = min(max(0, pos), 250) # проверяем значение pos

self.\_bus.write\_byte\_data(self.\_addr, Registers.REG\_SERVO\_1, pos)

def beep(self):

""" Бибикнуть """

self.\_bus.write\_byte\_data(self.\_addr, Registers.REG\_BEEP, 3)

def \_\_onlineThread(self):

""" поток отправляющий онлайн метки """

while not self.\_\_exit:

if self.online: # если включена посылка онлайн меток

self.\_bus.write\_byte\_data(self.\_addr, Registers.REG\_ONLINE, 1)

time.sleep(1)

def start(self):

threading.Thread(target=self.\_\_onlineThread, daemon=True).start() # включаем посылку онлайн меток

def exit(self):

self.\_\_exit = True

if \_\_name\_\_ == "\_\_main\_\_":

bus = smbus.SMBus(1)

bot = EduBot(bus)

bot.start()

print(bot.whoIam())

bot.setMotorMode(MotorMode.MOTOR\_MODE\_PID)

bot.setParrot0(0x05)

time.sleep(5)

bot.exit()