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 с OLED дисплеем 128x64
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 A в попугаях в режиме WORK_MODE_PID_I2C
    REG_PARROT_1 = 0x0B  # скорость вращения мотора A в попугаях в режиме WORK_MODE_PID_I2C
    REG_DIR_0 = 0x0C  # направление вращения мотора A
    REG_PWM_0 = 0x0D  # ШИМ задаваемый мотору A 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()
```