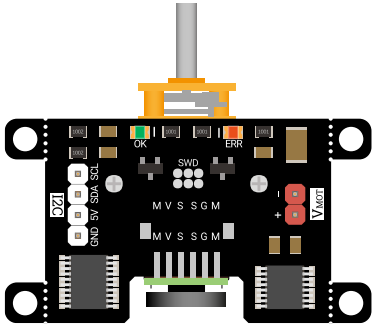
Мотор-редуктор с управляющим контроллером, FLASH-I2C, подключаем к Raspberry



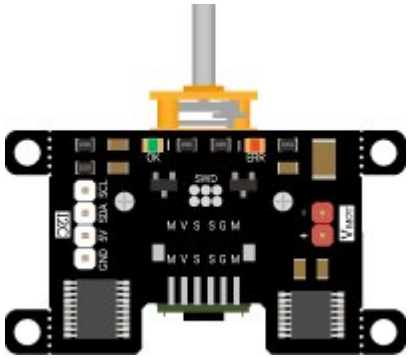
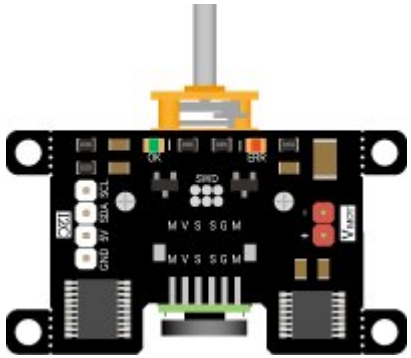
Общие сведения:

Модуль - Мотор-редуктор с управляющим контроллером, I2C-flash - является устройством состоящим из коллекторного двигателя с редуктором и платы управления, подключаемой к шине I2C.

Модуль относится к серии «Flash», а значит к одной шине I2C можно подключить более 100 модулей, так как их адрес на шине I2C (по умолчанию 0x09), хранящийся в энергонезависимой памяти, можно менять программно.

Модуль можно использовать для управления подвижными механизмами (машины, танки, тракторы), а так же для управления роботами и станками.

Модуль выполнен в двух вариантах - с энкодером и без энкодера

	Без энкодера	**С энкодером**
		
Управление скоростью, ШИМ	✓	✓
Управление скоростью, м/с, обороты/с	✗	✓
Остановка по заданному времени	✓	✓
Остановка по заданному расстоянию	✗	✓
Остановка по заданному кол-ву оборотов вала	✗	✓

	Без энкодера	С энкодером
Получение пройденного пути	✗	✓

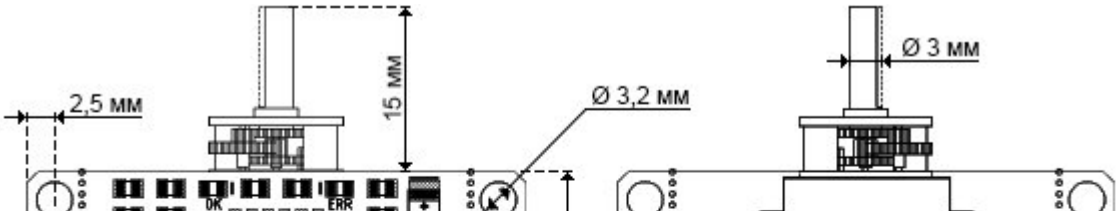
> Функции `setSpeed()`, `getSpeed()`, `setStop()` и `getSum()`, вызванные с параметрами MOT_RPM, MOT_M_S, MOT_MET и MOT_REV поддерживаются только модулем с установленным энкодером.

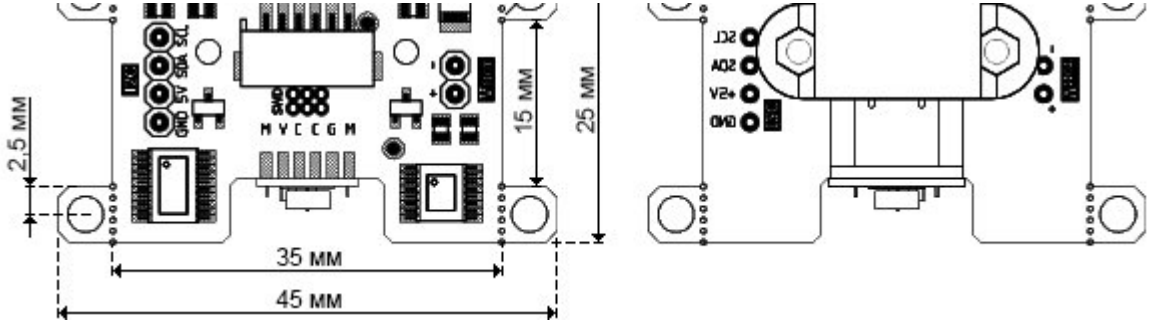
Видео:

Редактируется ...

Спецификация:

- Напряжение питания логики: 5 В (номинально), или 3,3 В.
- Диапазон напряжений мотора поддерживаемый драйвером: 2,7 В ... 12 В.
- Максимальный ток мотора поддерживаемый драйвером: до 3 А (пиковый ток до 4 А).
- Драйвер оснащён защитой от перегрева и перегрузки по току.
- Интерфейс: I2C.
- Скорость шины I2C: 100 кбит/с.
- Адрес на шине I2C: устанавливается программно (по умолчанию 0x09).
- Уровень логической 1 на линиях шины I2C: Vcc.
- Рабочая температура: от -20 до +70 °C.
- Габариты: 45 x 40 мм.
- Вес: 32 г.





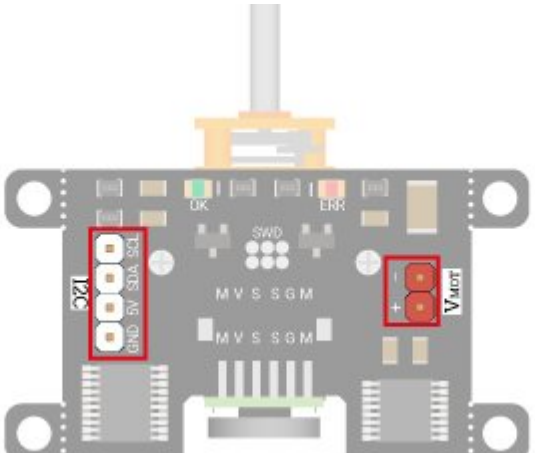
Подключение:

На плате модуля расположен разъем из 4 выводов для подключения к шине I2C.

- **SCL** - вход/выход линии тактирования шины I2C.
- **SDA** - вход/выход линии данных шины I2C.
- **5V** - вход питания +5 В (номинально), или 3,3 В.
- **GND** - общий вывод питания (соединён с выводом питания мотора **-$V_{МОТ}$**).

А так же разъем из 2 выводов для подачи питания на мотор через драйвер модуля.

- **+$V_{МОТ}$** - вход питания мотора от +2,7 В до +12 В.
- **-$V_{МОТ}$** - общий вывод питания (соединён с выводом **GND**).



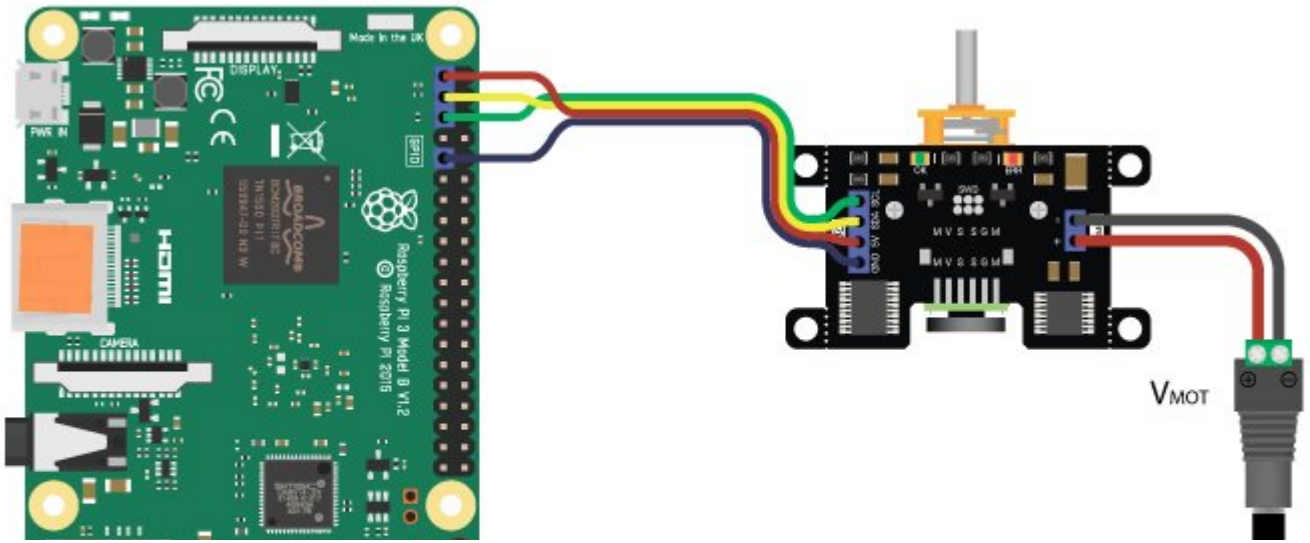
Модуль удобно подключать 2 способами, в зависимости от ситуации:

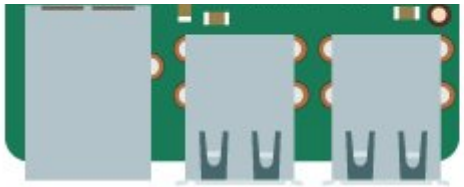
Способ - 1: Используя провода и Raspberry Pi

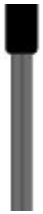
Используя провода «Мама — Мама», подключаем напрямую к Raspberry Pi

В этом случае необходимо питать логическую часть модуля от 3,3 В Raspberry

Вывод Raspberry	Вывод модуля
GPIO 2	SDA
GPIO 3	SCL
3.3V	5V
GND	GND

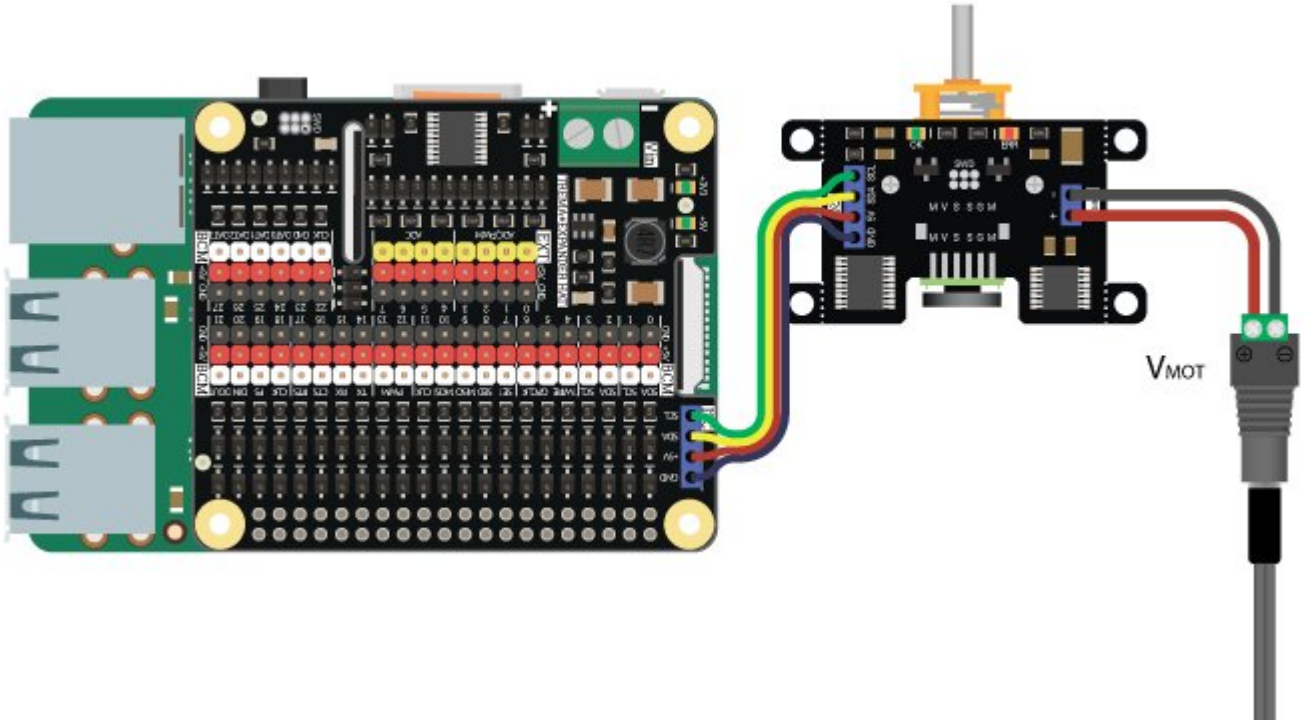






Способ - 2: Используя Trema+Expander Hat

Используя 4-х проводной шлейф, подключаем к Trema+Expander Hat.



Питание:

Входное напряжение питания модуля 5В (номинально), или 3,3В постоянного тока, подаётся на выводы 5V и GND.

Входное напряжение питания мотора от 2,7В до 12В постоянного тока, подаётся на выводы $+V_{MOT}$ и $-V_{MOT}$.

Подробнее о модуле:

Модуль построен на базе двигателя GM12-N20, редуктора, микроконтроллера STM32F030F4 и драйвера DRV8833, снабжен многополюсным магнитным валом, датчиками Холла, и собственным стабилизатором напряжения. Модуль способен поддерживать заданную скорость и направление вращения вала, сверяясь с показаниями датчиков Холла. Модуль самостоятельно обрабатывает данные с датчиков и корректирует скорость. На плате модуля имеется красный светодиод информирующий об отличии реальной скорости от заданной.

Модуль позволяет:

- Менять свой адрес на шине I2C.
- Управлять внутренней подтяжкой линий шины I2C (по умолчанию включена).
- Менять передаточное отношение редуктора мотора (при его замене).
- Менять количество полюсов (одной полярности) магнитного вала (при его замене).
- Менять борт установки мотора (левый мотор / правый мотор).
- Задать скорость вращения вала указав количество оборотов в минуту, ШИМ в % или метры в секунду. Во всех случаях можно указывать отрицательные значения для вращения в обратную сторону.
- Узнать отличается ли заданная скорость вращения вала от реальной, а так же указать процент отклонения при котором будет включаться красный светодиод на плате модуля.
- Узнать текущую скорость вращения вала. Скорость вращения вала определяется по показаниям с датчиков Холла, даже если мотор отключён, а вал вращается по средством внешних сил.
- Узнать количество совершённых полных оборотов вала. Количество оборотов вала определяется по показаниям с датчиков Холла, даже если мотор отключён, а вал вращается по средством внешних сил.
- Остановить двигатель и/или указать тип его остановки. Двигатель может быть остановлен двумя способами: отключением мотора (свободный ход) или торможением (стопор). Заданный тип применяется ко всем последующим остановкам двигателя.
- Остановить двигатель по истечении заданного количества полных оборотов вала, по истечении пройденного пути, или по истечении заданного времени.
- Узнать о наличии ошибки драйвера (перегрузка по току, перегрев, низкое напряжение).

Специально для работы с модулем - Мотор-редуктор с управляющим контроллером, I2C-flash, нами разработана библиотека pyiArduinoI2Cmotor которая позволяет реализовать все функции модуля.

Для работы с модулем необходимо включить шину I2C.
Ссылка на подробное описание как это сделать.

Для подключения библиотеки необходимо сначала её установить. Сделать это можно в менеджере модулей в Thonny Python IDE (тогда библиотека установится локально для этого IDE), в виртуальной среде командой `pip3 install pyiArduinoI2Cmotor` или в терминале Raspberry (тогда библиотека будет системной) командой:

```
sudo pip3 install pyiArduinoI2Cmotor
```

Подробнее об установке библиотек можно узнать в этой статье.

Примеры:

Смена адреса модуля на шине I2C:

Пример позволяет указать адрес модулю, даже если его текущий адрес Вам неизвестен.

```
# Подключаем библиотеку для работы с модулем
from pyiArduinoI2Cmotor import *
import sys

# Объявляем объект module для работы с функциями и методами библиотеки pyiArduinoI2Cled.
# Если при объявлении объекта указать адрес, например, module(0x0B),
# то пример будет работать с тем модулем, адрес которого был указан.
module = pyiArduinoI2Cmotor(None, NO_BEGIN)

# Если сценарию не были переданы аргументы
```

```
if len(sys.argv) < 2:
    # Назначаем модулю адрес (0x07 < адрес < 0x7F).
    newAddress = 0x09

# Иначе
else:
    # Новый адрес - первый аргумент
    newAddress = int(sys.argv[1])

# Если модуль найден
if module.begin():
    print("Найден модуль %#.2x" % module.getAddress())

    # Если адрес удалось изменить
    if module.changeAddress(newAddress):
            print("Адрес изменён на %#.2x" % module.getAddress())

    else:
            print("Адрес не изменён!")

else:
    print("Модуль не найден!")
```

Для работы данного примера, на шине I2C должен быть только один мотор.

Данный скетч демонстрирует не только возможность смены адреса на указанный в переменной `newAddress`, но и обнаружение, и вывод текущего адреса модуля на шине I2C.

Запуск мотора с указанием скорости:

Пример позволяет запустить мотор указав количество оборотов в минуту.

```
#   Подключаем библиотеку для работы с мотором I2C-flash.
```

```
from pyiArduinoI2Cmotor import *
from time import sleep

#   Объявляем объект для работы с функциями и методами библиотеки pyiArduinoI2Cmotor, указывая адрес модуля на шине I2C.
mot = pyiArduinoI2Cmotor(0x09)

while True:
    #   Запускаем мотор на скорости  120 об/мин и ждём 5 секунд.
    mot.setSpeed( 120, MOT_RPM)
    sleep(5)

    #   Останавливаем мотор и ждём 5 секунд.
    mot.setStop()
    sleep(5)

    #   Запускаем мотор на скорости -120 об/мин и ждём 5 секунд.
    mot.setSpeed(-120, MOT_RPM)
    sleep(5)

    #   Останавливаем мотор и ждём 5 секунд.
    mot.setStop()
    sleep(5)
```

После загрузки данного примера, начнёт выполняться цикл состоящий из 4 действий: запуск мотора на скорости 120 об/мин на 5 секунд, остановка мотора на 5 секунд, запуск мотора на скорости 120 об/мин на 5 секунд в противоположную сторону, остановка мотора на 5 секунд.

Второй параметр функции `setSpeed()` указывает как задана скорость:

- `MOT_RPM` - скорость задана количеством оборотов в минуту от 0 до ±32'767 об/мин.
- `MOT_PWM` - скорость задана коэффициентом заполнения ШИМ от 0 до ±100.0%.
- `MOT_M_S` - скорость задана в м/сек. (должен быть указан радиус колеса `mot.radius`).

Запуск мотора для движения на указанное расстояние:

Пример запускает мотор однократно, указав модулю самостоятельно остановить мотор после преодоления определённого расстояния.

```
#   Подключаем библиотеку для работы с мотором I2C-flash.
from pyiArduinoI2Cmotor import *
from time import sleep

#   Объявляем объект для работы с функциями и методами библиотеки pyiArduinoI2Cmotor, указывая адрес модуля на шине I2C.
mot = pyiArduinoI2Cmotor(0x09)

mot.radius = 12.2                           #  Указываем радиус колеса в мм.
#mot.setSpeed(100, MOT_RPM,  5.5, MOT_REV)  #  Запускаем мотор на скорости 100 об/мин с остановкой через 5 с половиной оборо
#mot.setSpeed(100, MOT_RPM,    2, MOT_SEC)  #  Запускаем мотор на скорости 100 об/мин с остановкой через 2 секунды.
mot.setSpeed(100, MOT_RPM, 0.05, MOT_MET)   #  Запускаем мотор на скорости 100 об/мин с остановкой через 0.05 метров = 5см =
```

Данный пример запускает мотор на скорости 100 об/мин, передав модулю условие остановки мотора.

Второй параметр функции `setSpeed()` указывает как задана скорость:

- `MOT_RPM` - скорость задана количеством оборотов в минуту от 0 до ±32'767 об/мин.
- `MOT_PWM` - скорость задана коэффициентом заполнения ШИМ от 0 до ±100.0%.
- `MOT_M_S` - скорость задана скоростью движения в м/сек.

Четвёртый параметр функции `setSpeed()` указывает как задано условие остановки:

- `MOT_REV` - условие остановки задано количеством оборотов вала от 0,01 до 167'772,15.
- `MOT_SEC` - условие остановки задано временем от 0,001 до 16'777,215 секунд.
- `MOT_MET` - условие остановки задано расстоянием пройденного пути в метрах.

Обратите внимание на то, что если скорость задана в м/сек (`MOT_M_S`) или условием остановки является расстояние пройденного пути в

метрах (`MOT_MET`), то необходимо однократно указать радиус колеса в миллиметрах (`mot.radius=РАДИУС`) используемого для движения.

Остановка мотора с освобождением ротора или без такового:

Пример определяет поведение мотора при остановке. Мотор может быть остановлен двумя способами: отключением мотора (свободный ход) или торможением (стопор). Тип остановки заданный функцией `setStopNeutral()` применяется ко всем последующим остановкам двигателя.

```
#   Подключаем библиотеку для работы с мотором I2C-flash.
from pyiArduinoI2Cmotor import *
from time import sleep

#   Объявляем объект для работы с функциями и методами библиотеки pyiArduinoI2Cmotor, указывая адрес модуля на шине I2C.
mot = pyiArduinoI2Cmotor(0x09)

while True:
    #   Запускаем и плавно останавливаем мотор:
    #   Запускаем мотор на максимальной скорости, указав максимальное значение ШИМ = 100%.
    mot.setSpeed(100.0, MOT_PWM)
    sleep(5)

    #   Указываем освободить мотор при его остановке. Ротор остановленного мотора можно вращать.
    mot.setStopNeutral(True)

    #   Останавливаем мотор. Обратите внимание на то, что ротор мотора останавливается плавно.
    mot.setStop()
    sleep(1)

    #   Запускаем и резко останавливаем мотор:
    #   Запускаем мотор на максимальной скорости, указав максимальное значение ШИМ = 100%.
    mot.setSpeed(100.0, MOT_PWM)
    sleep(5)
```

```
    #   Указываем не освобождать мотор при его остановке. Ротор остановленного мотора будет застопорен.
    mot.setStopNeutral(False)

    #   Останавливаем мотор. Обратите внимание на то, что ротор мотора останавливается резко.
    mot.setStop()
    sleep(1)
```

После загрузки данного примера, мотор будет запускаться и останавливаться, но остановка мотора будет либо плавной (ротор свободно останавливается), либо резкой (ротор стопорится).

Функция `setStopNeutral()` не останавливает мотор, а указывает его поведение при остановке. Останавливается мотор функцией `setStop()`.

Второй параметр функции `setSpeed()` указывает как задана скорость:

- `MOT_RPM` - скорость задана количеством оборотов в минуту от 0 до ±32'767 об/мин.
- `MOT_PWM` - скорость задана коэффициентом заполнения ШИМ от 0 до ±100.0%.
- `MOT_M_S` - скорость задана в м/сек. (должен быть указан радиус колеса `mot.radius`).

Изменение установки мотора (мотор слева / мотор справа):

При использовании модулей для управления подвижным механизмом, например, машиной, чаще всего устанавливают два мотора слева и справа. Если при такой установке задать обоим моторам одинаковую скорость (для движения механизма по прямой), то один мотор будет двигать механизм вперёд, а второй назад. Избежать такого поведения можно отправляя скорость с разными знаками для разных моторов, а можно однократно указать тип установки моторов при помощи функции `setDirection()`.

```
#   Подключаем библиотеку для работы с мотором I2C-flash.
from pyiArduinoI2Cmotor import *
from time import sleep

#   Объявляем объект mot1 для работы с функциями и методами библиотеки pyiArduinoI2Cmotor, указывая адрес модуля на шине I2C.
mot1 = pyiArduinoI2Cmotor(0x09)
```

```
#   Объявляем объект mot2 для работы с функциями и методами библиотеки pyiArduinoI2Cmotor, указывая адрес модуля на шине I2C.
mot2 = pyiArduinoI2Cmotor(0x0A)
#   При наличии нескольких моторов на шине I2C нельзя объявлять объект без указания адреса.

#   Задаём направление вращения для 1 мотора: по часовой стрелке при положительных скоростях и против при отрицательных.
mot1.setDirection(True)

#   Задаём направление вращения для 2 мотора: против часовой стрелки при положительных скоростях и по при отрицательных.
mot2.setDirection(False)

while True:
    mot1.setSpeed(120, MOT_RPM)         #   Запускаем мотор 1 на скорости  120 об/мин.
    mot2.setSpeed(120, MOT_RPM)         #   Запускаем мотор 2 на скорости  120 об/мин.
    sleep(5)                            #   Ждём 5 секунд.
    mot1.setStop()                      #   Останавливаем мотор 1.
    mot2.setStop()                      #   Останавливаем мотор 2.
    sleep(5)                            #   Ждём 5 секунд.
```

Моторы, колеса которых вращаются по часовой стрелке, при положительных скоростях, устанавливают по правому борту, а против часовой стрелки, по левому.

Если моторы расположить: 1 справа, 2 слева. То при одинаковых положительных скоростях механизм будет двигаться строго вперёд, а при одинаковых отрицательных, строго назад.

Описание функций библиотеки:

В данном разделе описаны функции библиотеки iarduino_I2C_Motor для работы с модулем - Мотор-редуктор с управляющим контроллером, I2C-flash.

Данная библиотека может использовать как аппаратную, так и программную реализацию шины I2C. О том как выбрать тип шины I2C

рассказано в статье Wiki - расширенные возможности библиотек iarduino для шины I2C.

Подключение библиотеки:

- Если адрес модуля известен (в примере используется адрес 0x09):

```
from pyiArduinoI2Cmotor import *       #   Подключаем библиотеку для работы с мотором I2C-flash.
mot = pyiArduinoI2Cmotor(0x09)         #   Объявляем объект mot1 для работы с функциями и методами библиотеки pyiArduinoI2Cmot
```

- Если адрес модуля неизвестен (адрес будет найден автоматически):

```
from pyiArduinoI2Cmotor import *       #   Подключаем библиотеку для работы с мотором I2C-flash.
mot = pyiArduinoI2Cmotor()             #   Создаём объект mot для работы с функциями и методами библиотеки iarduino_I2C_Motor,
```

- При создании объекта без указания адреса, на шине должен находиться только один модуль.

Функция reset()

- Назначение: Перезагрузка модуля.
- Синтаксис: reset()
- Параметры: Нет.
- Возвращаемое значение: - результат перезагрузки (True или False).
- Пример:

```
if mot.reset():
    print( "Модуль    перезагружен" )
else:
    print( "Модуль не перезагружен" )
```

Функция changeAddress()

- Назначение: Смена адреса модуля на шине I2C.
- Синтаксис: changeAddress(АДРЕС)
- Параметр:
  - АДРЕС - новый адрес модуля на шине I2C (целое число от 0x08 до 0x7E)
- Возвращаемое значение: - результат смены адреса (True или False).
- Примечание:
  - Адрес модуля сохраняется в энергонезависимую память, а значит будет действовать и после отключения питания.
  - Текущий адрес модуля можно узнать функцией getAddress().
- Пример:

```
if mot.changeAddress(0x12):
    print( "Адрес модуля изменён на 0x12" )
else:
    print( "Не удалось изменить адрес"    )
```

Функция getAddress()

- Назначение: Запрос текущего адреса модуля на шине I2C.
- Синтаксис: getAddress()
- Параметры: Нет.
- Возвращаемое значение: АДРЕС - текущий адрес модуля на шине I2C (от 0x08 до 0x7E)
- Примечание: Функция может понадобиться если адрес модуля не указан при создании объекта, а обнаружен библиотекой.
- Пример:

```
print( "Адрес модуля на шине I2C = 0x" % mot.getAddress() )
```

Функция getVersion()

- Назначение: Запрос версии прошивки модуля.
- Синтаксис: getVersion()

- Параметры: Нет
- Возвращаемое значение: ВЕРСИЯ - номер версии прошивки от 0 до 255.
- Пример:

```
print( "Версия прошивки модуля " )
print( mot.getVersion() )
```

Функция setPullI2C()

- Назначение: Управление внутрисхемной подтяжкой линий шины I2C.
- Синтаксис: setPullI2C([ФЛАГ])
- Параметр:
  - ЛАГ требующий установить внутрисхемную подтяжку линий шины I2C (True или False).
- Возвращаемое значение:
  - ▪ результат включения / отключения внутрисхемной подтяжки (True или False).
- Примечание:
  - Вызов функции без параметра равносилен вызову функции с параметром True - установить.
  - Флаг установки внутрисхемной подтяжки сохраняется в энергонезависимую память модуля, а значит будет действовать и после отключения питания.
  - Внутрисхемная подтяжка линий шины I2C осуществляется до уровня 3,3 В, но допускает устанавливать внешние подтягивающие резисторы и иные модули с подтяжкой до уровня 3,3 В или 5 В, вне зависимости от состояния внутрисхемной подтяжки модуля.
- Пример:

```
if mot.setPullI2C(True ):
    print( "Внутрисхемная подтяжка установлена." )
if mot.setPullI2C(False):
    print( "Внутрисхемная подтяжка отключена."   )
```

Функция getPullI2C()

- Назначение: Запрос состояния внутрисхемной подтяжки линий шины I2C.
- Синтаксис: getPullI2C()
- Параметры: Нет.
- Возвращаемое значение: - ФЛАГ включения внутрисхемной подтяжки (True или False).
- Пример:

```
if mot.getPullI2C():
    print( "Внутрисхемная подтяжка включена."  )
else:
    print( "Внутрисхемная подтяжка отключена." )
```

Функция setSpeed()

- Назначение: Установка скорости.
- Синтаксис: setSpeed(СКОРОСТЬ, ТИП СКОРОСТИ [, УСЛОВИЕ, ТИП УСЛОВИЯ])
- Параметры:
- ТИП СКОРОСТИ - принимает одно из трёх значений:
- MOT_RPM - скорость задана количеством оборотов в минуту.
- MOT_PWM - скорость задана коэффициентом заполнения ШИМ.
- MOT_M_S - скорость задана в м/сек.
- СКОРОСТЬ - значение зависит от указанного типа скорости.
- Если тип скорости задан значением MOT_RPM, то скорость указывается количеством оборотов в минуту, от 0 до ±32'767.
- Если тип скорости задан значением MOT_PWM, то скорость указывается коэффициентом заполнения ШИМ, от 0 до ±100,0%. (шаг 0,025%).
- Если тип скорости задан значением MOT_M_S, то скорость указывается в м/сек.
- ТИП УСЛОВИЯ - принимает одно из тёх значений:
- MOT_REV - условие остановки задано количеством полных оборотов вала.
- MOT_SEC - условие остановки задано временем.
- MOT_MET - условие остановки задано расстоянием.
- УСЛОВИЕ - условие остановки, значение зависит от указанного типа условия:
- Если тип условия задан значением MOT_REV, то в качестве условия указывается количество полных оборотов до остановки, от 0.01 до

167'772.15 оборотов.

- Если тип условия задан значением MOT_SEC, то в качестве условия указывается время до остановки, от 0,001 до 16'777,215 секунд.
- Если тип условия задан значением MOT_MET, то в качестве условия указывается расстояние до остановки в метрах.
- Возвращаемое значение: - результат установки скорости (True или False).
- Примечание:
  - СКОРОСТЬ может быть отрицательной, знак указывает на направление вращения.
  - Если функция указана без параметров УСЛОВИЕ и ТИП УСЛОВИЯ, то мотор будет запущен на указанной скорости, пока не будет остановлен функцией `setStop()` .
  - Если функция указана с параметрами УСЛОВИЕ и ТИП УСЛОВИЯ, то мотор будет остановлен модулем самостоятельно, по истечении указанного условия остановки.
  - Если в качестве типа скорости указано значение `MOT_M_S` (скорость в м/сек) или в качестве типа условия остановки указано значение `MOT_MET` (расстояние в метрах), то до обращения к данной функции должен быть указан радиус колеса `mot.radius=РАДИУС` . Радиус указывается в мм, его можно указать однократно в коде `setup()` .
  - Если скорость задана не значением ШИМ, то пока мотор не достигнет указанной скорости, на плате модуля будет светиться красный светодиод, а функция `getError()` будет возвращать ошибку скорости `MOT_ERR_SPD` .
  - Задавать скорость через ШИМ удобно в тех случаях, когда мотор требуется запустить в процентах от его максимальной скорости.
  - Если на роторе мотора нет кольцевого магнита, или в модуле нет датчиков Холла, тогда:
    - Установка скорости с параметром `MOT_RPM` или `MOT_M_S` будет проигнорирована.
    - Условие остановки с параметром `MOT_REV` или `MOT_MET` будет проигнорировано.
  - Проверить наличие магнита и датчиков Холла можно функцией `getMagnet()` .
- Пример:

```
mot.setSpeed(60,   MOT_RPM)               # Запускаем мотор на скорости 60 об/мин.
mot.setSpeed(60,   MOT_RPM, 0.1, MOT_MET)  # Запускаем мотор на скорости 60 об/мин с остановкой мотора через 0.1 метра.
mot.setSpeed(60,   MOT_RPM, 3.5, MOT_REV)  # Запускаем мотор на скорости 60 об/мин с остановкой через 3,5 оборота вала.
mot.setSpeed(60,   MOT_RPM,   2, MOT_SEC)  # Запускаем мотор на скорости 60 об/мин с остановкой через 2 секунды.
mot.setSpeed(50.0, MOT_PWM)               # Запускаем мотор на 50%.
mot.setSpeed(50.0, MOT_PWM, 0.1, MOT_MET)  # Запускаем мотор на 50% с остановкой мотора через 0.1 метра.
mot.setSpeed(50.0, MOT_PWM, 3.5, MOT_REV)  # Запускаем мотор на 50% с остановкой через 3,5 оборота вала.
```

```
mot.setSpeed(50.0, MOT_PWM,   2, MOT_SEC)  # Запускаем мотор на 50% с остановкой через 2 секунды.
mot.setSpeed(0.01, MOT_M_S)                # Запускаем мотор на скорости 0,01 м/сек.
mot.setSpeed(0.01, MOT_M_S, 0.1, MOT_MET)  # Запускаем мотор на скорости 0,01 м/сек с остановкой мотора через 0.1 метра.
mot.setSpeed(0.01, MOT_M_S, 3.5, MOT_REV)  # Запускаем мотор на скорости 0,01 м/сек с остановкой через 3,5 оборота вала.
mot.setSpeed(0.01, MOT_M_S,   2, MOT_SEC)  # Запускаем мотор на скорости 0,01 м/сек с остановкой через 2 секунды.
```

Функция getSpeed()

- Назначение: Получение реальной скорости или ШИМ.
- Синтаксис: getSpeed(ТИП)
- Параметр: ТИП - тип получаемого значения:
  - MOT_RPM - получить реальную скорость количеством оборотов в минуту.
  - MOT_PWM - получить текущий коэффициент заполнения ШИМ.
  - MOT_M_S - получить реальную скорость в м/сек.
- Возвращаемое значение: - реальная скорость, значение зависит от запрошенного типа:
  - Если тип задан значением MOT_RPM, то функция возвращает реальную скорость количеством оборотов в минуту от 0 до ±32'767.
  - Если тип задан значением MOT_PWM, то функция возвращает текущий коэффициент заполнения ШИМ от 0 до ±100.0% (шаг 0,025%).
  - Если тип задан значением MOT_M_S, то функция возвращает реальную скорость в м/сек.
- Примечание:
  - Скорость вращения вала определяется по показаниям с датчиков Холла, вне зависимости от того, как запущен мотор, функцией `setSpeed()` или его вал вращается по средством внешних сил, даже если мотор отключён.
  - Возвращаемое значение может быть отрицательным, знак указывает на направление вращения.
  - Если в качестве типа получаемого значения указано `MOT_M_S` (получить скорость в м/сек), то до обращения к данной функции должен быть указан радиус колеса `mot.radius=РАДИУС`. Радиус указывается в мм, его можно указать однократно в коде `setup()`.
  - Если на роторе мотора нет кольцевого магнита, или в модуле нет датчиков Холла, тогда функция будет возвращать 0, если её вызвать с параметром `MOT_RPM` или `MOT_M_S`.
  - Проверить наличие магнита и датчиков Холла можно функцией `getMagnet()`.
- Пример:

```
i = mot.getSpeed( MOT_RPM )  # Получить реальную скорость в об/мин.
j = mot.getSpeed( MOT_M_S )  # Получить реальную скорость в м/сек.
k = mot.getSpeed( MOT_PWM )  # Получить текущий коэффициент заполнения ШИМ.
```

Функция setStop()

- Назначение: Остановка мотора с условием или без.
- Синтаксис: setStop([УСЛОВИЕ, ТИП])
- Параметры:
- ТИП - тип условия остановки, принимает одно из тёх значений:
- MOT_REV - условие остановки задано количеством полных оборотов вала.
- MOT_SEC - условие остановки задано временем.
- MOT_MET - условие остановки задано расстоянием.
- УСЛОВИЕ - условие остановки, значение зависит от типа условия:
- Если тип задан значением MOT_REV, то в качестве условия указывается количество полных оборотов до остановки, от 0.01 до 167'772.15 оборотов.
- Если тип задан значением MOT_SEC, то в качестве условия указывается время до остановки, от 0,001 до 16'777,215 секунд.
- Если тип задан значением MOT_MET, то в качестве условия указывается расстояние до остановки в метрах.
- Возвращаемое значение: - результат записи данных остановки в модуль.
- Примечание:
  - Если функция указана без параметров, то мотор будет остановлен сразу.
  - Если функция указана с параметрами УСЛОВИЕ и ТИП, то мотор будет остановлен модулем самостоятельно, по истечении указанного условия остановки.
  - Если в качестве типа условия остановки указано значение `MOT_MET` (расстояние в метрах), то до обращения к данной функции должен быть указан радиус колеса `mot.radius=РАДИУС`. Радиус указывается в мм, его можно указать однократно в коде `setup()`.
  - Если в качестве условия остановки указать 0, то мотор не остановится, а ранее заданное условие того же типа будет отменено, при этом типы `MOT_REV` и `MOT_MET` приравнены.
  - Если на роторе мотора нет кольцевого магнита, или в модуле нет датчиков Холла, тогда условие остановки с параметром `MOT_REV` или `MOT_MET` будет проигнорировано.

- Проверить наличие магнита и датчиков Холла можно функцией `getMagnet()` .
- Пример:

```
mot.setStop()               # Остановить мотор сразу (без условий).
mot.setStop( 0.5, MOT_MET)  # Остановить мотор через 0,5 метров пути.
mot.setStop( 3.5, MOT_REV)  # Остановить мотор через 3,5 оборота вала.
mot.setStop( 6.5, MOT_SEC)  # Остановить мотор через 6,5 секунд.
mot.setStop(   0, MOT_MET)  # Отменить заданную ранее остановку мотора по пройденному пути и количеству оборотов.
mot.setStop(   0, MOT_REV)  # Отменить заданную ранее остановку мотора по количеству оборотов и пройденному пути.
mot.setStop(   0, MOT_SEC)  # Отменить заданную ранее остановку мотора по времени.
```

Функция getStop()

- Назначение: Получение значения оставшегося до остановки.
- Синтаксис: getStop(ТИП)
- Параметр: ТИП - тип получаемого значения:
  - MOT_REV - получить количество оборотов вала оставшихся до остановки.
  - MOT_SEC - получить время оставшееся до остановки.
  - MOT_MET - получить расстояние оставшееся до остановки.
- Возвращаемое значение: - одно из значений:
  - Если тип задан значением MOT_REV, то функция возвращает количество оборотов оставшихся до остановки, от 0.01 до 167'772.15 полных оборотов.
  - Если тип задан значением MOT_SEC, то функция возвращает время оставшееся до остановки, от 0,001 до 16'777,215 секунд.
  - Если тип задан значением MOT_MET, то функция возвращает расстояние оставшееся до остановки в метрах.
- Примечание:
  - Функция `getStop()` возвращает значение оставшееся до остановки мотора, заданное ранее функциями `setStop()` или `setSpeed()` , того же типа.
  - Если тип условия остановки заданный функциями `setStop()` или `setSpeed()` отличается от запрашиваемого типа, то функция возвращает значение рассчитанное с учётом текущей скорости вращения вала. Это значение может сильно отличаться от действительного, пока мотор не набрал заданную ему скорость.

- Если запрошено расстояние оставшееся до остановки `MOT_MET`, то до обращения к данной функции должен быть указан радиус колеса `mot.radius=РАДИУС`. Радиус указывается в мм, его можно указать однократно в коде `setup()`.
- Если на роторе мотора нет кольцевого магнита, или в модуле нет датчиков Холла, тогда функция сможет вернуть только время до остановки `MOT_SEC`, при том что условие остановки так же было задано функциями `setStop()` или `setSpeed()` через время `MOT_SEC`, иначе функция вернёт 0.
- Проверить наличие магнита и датчиков Холла можно функцией `getMagnet()`.

- Пример:

```
mot.setSpeed(100, MOT_RPM, 50, MOT_REV)   # Запускаем мотор на скорости 100 об/мин с остановкой мотора после 50 полных оборотс
while mot.getStop(MOT_REV):               # Если до остановки ещё есть не пройденное количество оборотов.
    print( mot.getStop(MOT_REV) )         # Выводим оставшееся оставшееся количество оборотов до остановки.
```

Функция setStopNeutral()

- Назначение: Установка нейтрального положения при остановке мотора.
- Синтаксис: setStopNeutral(ФЛАГ)
- Параметр: ФЛАГ - указывает переводить ротор в нейтральное положение при остановке.
- Возвращаемое значение: - результат применения данных.
- Примечание:
  - Функция не останавливает мотор, а определяет его поведение при остановке.
  - Если вызвать функцию с параметром `True`, то при остановке мотор будет отключён, при этом ротор можно вращать, как при нейтральном положении машины.
  - Если вызвать функцию с параметром `False`, то при остановке мотор будет застопорен, при этом ротор будет трудно вращать.
  - Выбранное состояние мотора будет применяется ко всем последующим остановкам.
  - По умолчанию ротор можно вращать при остановке.
- Пример:

```
mot.setStopNeutral(True)   # Указываем освободить мотор при его остановке. Ротор остановленного мотора можно вращать.
mot.setStopNeutral(False)  # Указываем не освобождать мотор при его остановке. Ротор остановленного мотора будет застопорен.
```

Функция getStopNeutral()

- Назначение: Получение установленного поведения мотора при остановке.
- Синтаксис: getStopNeutral()
- Параметр: Нет.
- Возвращаемое значение: ФЛАГ - наличия нейтрального положения при остановке.
- Примечание: Функция возвращает значение заданное ранее функцией `setStopNeutral()` .
- Пример:

```
i = mot.getStopNeutral()
```

Функция getSum()

- Назначение: Получение количества совершённых оборотов или пройденного пути.
- Синтаксис: getSum(ТИП)
- Параметр: ТИП - тип получаемого значения:
  - MOT_REV - получить количество совершённых оборотов вала.
  - MOT_MET - получить пройденное расстояние.
- Возвращаемое значение: - количество совершённых оборотов или пройденный путь:
  - Если тип задан значением MOT_REV, то функция возвращает количество совершённых оборотов с момента их сброса, от 0.01 до 167'772.15 полных оборотов.
  - Если тип задан значением MOT_MET, то функция возвращает пройденный путь с момента его сброса в метрах
- Примечание:
  - Функция `getSum()` возвращает количество совершённых оборотов или пройденный путь с момента их сброса функцией `delSum()` .
  - Количество совершённых оборотов и пройденный путь определяются по показаниям с датчиков Холла, вне зависимости от того, как запущен мотор, функцией `setSpeed()` или его вал вращается по средством внешних сил, даже если мотор отключён.
  - Если запрошен пройденный путь `MOT_MET` , то до обращения к данной функции должен быть указан радиус колеса `mot.radius=РАДИУС` . Радиус указывается в мм, его можно указать однократно в коде setup().

    - Функция не будет работать если на роторе мотора нет кольцевого магнита, или в модуле нет датчиков Холла.
    - Проверить наличие магнита и датчиков Холла можно функцией `getMagnet()` .
- Пример:

```
i = mot.getSum( MOT_REV )  # Получить количество совершённых полных оборотов вала.
j = mot.getSum( MOT_MET )  # Получить пройденный путь в метрах.
```

Функция delSum()

- Назначение: Сброс количества совершённых оборотов и пройденного пути.
- Синтаксис: delSum()
- Параметр: Нет.
- Возвращаемое значение: - результат сброса совершённых оборотов и пройденного пути.
- Примечание:
    - Функция сбрасывает количество совершённых оборотов и пройденного пути, которые можно получить функцией `getSum()` .
    - Сброс количества совершённых оборотов и пройденного пути так же осуществляется обращением к функциям `setStop()` или `setSpeed()` вызванным с параметром `MOT_REV` или `MOT_MET` .
- Пример:

```
mot.delSum()  # Сбросить количество совершённых оборотов и пройденный путь в 0.
```

Функция setDirection()

- Назначение: Установка направления вращения вала.
- Синтаксис: setDirection(ФЛАГ)
- Параметры:
- ФЛАГ - флаг вращения вала в прямом направлении.
- Если ФЛАГ установлен, то при положительных скоростях вал будет вращаться по часовой стрелке, а при отрицательных скоростях, против часовой стрелки.
- Если ФЛАГ сброшен, то при положительных скоростях вал будет вращаться против часовой стрелки, а при отрицательных скоростях, по

часовой стрелке.

- Возвращаемое значение: - результат применения настройки.
- Примечание:
  - Функцию удобно использовать при установке моторов на подвижные механизмы, слева и справа. Если не обращаться к функции `setDirection()` и задать двум моторам одинаковую скорость функцией `setSpeed()`, то один мотор будет двигать механизм вперёд, а второй назад. Механизм будет разворачиваться на месте, как танк.
  - Если для правого мотора вызвать функцию `setDirection(True)`, а для левого мотора `setDirection(False)`, то задав положительную скорость обоим моторам, они будут двигать механизм в прямом направлении, а при отрицательных скоростях в обратном.
  - Функцию достаточно однократно вызвать в коде `setup()` для каждого мотора.
  - По умолчанию установлено прямое направление вращения вала.
- Пример:

```
mot1.setDirection(True)  # Задаём прямое   направление вращения. Вращение по ч.с. при положительных скоростях.
mot2.setDirection(False) # Задаём обратное направление вращения. Вращение против ч.с. при положительных скоростях.
```

Функция getDirection()

- Назначение: Получение направления вращения вала.
- Синтаксис: getDirection()
- Параметр: Нет.
- Возвращаемое значение: - ФЛАГ вращения вала в прямом направлении.
  - Если ФЛАГ == 1, то при положительных скоростях вал будет вращаться по часовой стрелке, а при отрицательных скоростях, против часовой стрелки.
  - Если ФЛАГ == 0, то при положительных скоростях вал будет вращаться против часовой стрелки, а при отрицательных скоростях, по часовой стрелке.
- Пример:

```
if mot.getDirection() == True:
```

```
    print("При положительных скоростях вал вращается по ч.с.")
else:
    print("При положительных скоростях вал вращается против ч.с.")}
```

Функция getError()

- Назначение: Получение наличия ошибки модуля.
- Синтаксис: getError()
- Параметр: Нет.
- Возвращаемое значение: - может принимать следующие значения:
  - 0 - модуль работает без ошибок.
  - MOT_ERR_SPD - ошибка скорости. Ошибка возникает при отличии реальной скорости от заданной функцией `setSpeed()` с параметром `MOT_RPM` или `MOT_M_S` .
  - MOT_ERR_DRV - ошибка драйвера. Ошибка возникает при перегрузке по току, перегреве чипа и при низком напряжении Vmot.
- Примечание:
  - При наличии двух ошибок `MOT_ERR_SPD` и `MOT_ERR_DRV` , будет возвращена первая.
  - Если на роторе мотора нет кольцевого магнита, или в модуле нет датчиков Холла, тогда в качестве ошибки можно получить только ошибку драйвера `MOT_ERR_DRV` .
  - Проверить наличие магнита и датчиков Холла можно функцией `getMagnet()` .
- Пример:

```
i = mot.getError()  # Получить ошибку модуля.
```

Функция getVoltage()

- Назначение: Получение номинального напряжения электродвигателя.
- Синтаксис: getVoltage()
- Параметр: Нет.
- Возвращаемое значение: - номинальное напряжение питания мотора в Вольтах.
- Примечание:

    - Функция возвращает напряжение на которое рассчитан электродвигатель установленный в модуле, а не текущее напряжение на нём.
    - Возвращаемое значение можно изменить функцией `setVoltage()`, до отключения питания или перезагрузки модуля. Но это не изменит номинальное напряжение мотора.
    - Если обратиться к функции `setVoltage()` указав любое значение, то функция `getVoltage()` будет возвращать не номинальное напряжение питания мотора, а указанное вами значение, до отключения питания или перезагрузки модуля.
- Пример:

```
i = mot.getVoltage()  # Получить напряжение на которое рассчитан мотор.
```

Функция getNominalRPM()

- Назначение: Получение номинальной скорости вращения вала.
- Синтаксис: getNominalRPM()
- Параметр: Нет.
- Возвращаемое значение: - номинальная скорость вращения от 0 до 65'535 об.мин.
- Примечание:
    - Функция возвращает номинальную скорость вращения вала редуктора. Скорость заявленную производителем при номинальном напряжении и 100% коэффициенте заполнения ШИМ.
    - Возвращаемое значение можно изменить функцией `setNominalRPM()`, до отключения питания или перезагрузки модуля. Но это не изменит номинальную скорость.
    - Если обратиться к функции `setNominalRPM()` указав любое значение, то функция `getNominalRPM()` будет возвращать не номинальную скорость вращения, а указанное вами значение, до отключения питания или перезагрузки модуля.
- Пример:

```
i = mot.getNominalRPM()  # Получить номинальную скорость вращения вала.
```

Функция getMagnet()

- Назначение: Получение количества полюсов многополюсного магнитного вала.
- Синтаксис: getMagnet()

- Параметр: Нет.
- Возвращаемое значение: КОЛИЧЕСТВО - значение от 1 до 255, или 0 - нет магнита.
- Примечание:
  - Если функция вернула 0, значит на роторе мотора нет кольцевого магнита, или в модуле отсутствуют датчики Холла. По этой причине модуль не может отслеживать вращение вала, следовательно, функции: `setSpeed()`, `getSpeed()`, `setStop()`, `getStop()` и `getSum()` не будут работать с параметрами: `MOT_RPM`, `MOT_M_S`, `MOT_MET` и `MOT_REV`, а функция `getError()` не будет возвращать ошибку скорости `MOT_ERR_SPD`.
- Пример:

```
i = mot.getMagnet()  # Получаем сохранённое количество полюсов одной полярности.
```

Функция getReducer()

- Назначение: Получение передаточного отношения редуктора.
- Синтаксис: getReducer()
- Параметр: Нет.
- Возвращаемое значение: ОТНОШЕНИЕ - значение от 0.01 до 167'772.15.
- Пример:

```
i = mot.getReducer()  # Получаем сохранённое передаточное отношение редуктора.
```

Функции библиотеки используемые для настройки:

Некоторые данные указываемые / получаемые следующими функциями хранятся в Flash памяти модуля, а значит сохраняются и после отключения питания:

Функция setInvGear()

- Назначение: Установка флагов инверсии механизма.
- Синтаксис: setInvGear(РЕДУКТОР, МОТОР)
- Параметры:

    - РЕДУКТОР - флаг инверсии вращения редуктора. Флаг должен быть установлен если вал редуктора вращается в сторону противоположную вращению ротора мотора.
    - МОТОР - флаг инверсии полярности мотора. Флаг должен быть установлен при обратном подключении выводов мотора, если ротор мотора вращается против ч.с.
- Возвращаемое значение: - результат применения новых данных.
- Примечание:
    - Заданные значения не сохраняются в энергонезависимую память модуля.
    - Функция может быть полезной, только при смене редуктора или мотора.
    - По умолчанию установлены значения в соответствии с типом редуктора и способа подключения мотора в модуле.
- Пример:

```
mot.setInvGear(False, False) # Редуктор НЕ инвертирует направление вращения, ротор мотора вращается НЕ против часовой стрелки.
mot.setInvGear(True , True ) # Редуктор    инвертирует направление вращения, ротор мотора вращается    против часовой стрелки.
```

Функция getInvGear()

- Назначение: Получение флагов инверсии механизма.
- Синтаксис: getInvGear()
- Параметр: Нет.
- Возвращаемое значение: БАЙТ - значение от 0 до 3.
    - 0 - редуктор без инверсии вращения, ротор мотора вращается по ч.с.
    - 1 - редуктор без инверсии вращения, ротор мотора вращается против ч.с.
    - 2 - редуктор с инверсией вращения, ротор мотора вращается по ч.с.
    - 3 - редуктор с инверсией вращения, ротор мотора вращается против ч.с.
- Примечание:
    - Нулевой бит возвращаемого байта является флагом инверсии полярности мотора.
    - Первый бит возвращаемого байта является флагом инверсии вращения редуктора.
- Пример:

```
InvReducer = mot.getInvGear() & (1 << 1) # Получаем флаг инверсии редуктора.
InvMotor   = mot.getInvGear() & (1 << 0) # Получаем Флаг инверсии полярности мотора.
```

Функция setFreqPWM()

- Назначение: Установка частоты ШИМ подаваемого на мотор.
- Синтаксис: setFreqPWM(ЧАСТОТА)
- Параметр: ЧАСТОТА - значение от 1 до 1000 Гц.
- Возвращаемое значение: - результат применения новой частоты.
- Примечание:
  - Заданное значение сохраняется в энергонезависимую память модуля, при этом модуль не сохраняет значение повторно, если оно не изменилось.
  - Функции задания скорости позволяют задавать уровень ШИМ (коэффициент заполнения) не влияя на частоту. А данная функция позволяет изменить частоту ШИМ (период следования импульсов) не влияя на уровень ШИМ.
  - Частота по умолчанию 500 Гц.
- Пример:

```
mot.setFreqPWM(100)  # Задаём частоту ШИМ равную 100 Гц.
```

Функция setError()

- Назначение: Установка процента максимального отклонения скорости до установки ошибки.
- Синтаксис: setError(ПРОЦЕНТ)
- Параметр: ПРОЦЕНТ - значение от 1 до 100.
- Возвращаемое значение: - результат сохранения нового процента отклонения.
- Примечание:
  - Заданное значение сохраняется в энергонезависимую память модуля, при этом модуль не сохраняет значение повторно, если оно не изменилось.
  - При отличии реальной скорости от заданной функцией `setSpeed()` с параметром `MOT_RPM` или `MOT_M_S` на плате модуля включается красный светодиод, а функция `getError()` начинает возвращать ошибку скорости `MOT_ERR_SPD`. Это происходит при

отличии реальной скорости от заданной на указанный функцией `setError()` процент. Меняя этот процент Вы меняете поведение светодиода и функции `getError()`.
    - Значение по умолчанию 10%
- Пример:

```
mot.setError(10)  # Установить максимальное отклонение скорости до установки ошибки как 10%.
```

Функция setVoltage()

- Назначение: Установка значения возвращаемого функцией getVoltage().
- Синтаксис: setVoltage(НАПРЯЖЕНИЕ)
- Параметр: НАПРЯЖЕНИЕ - значение от 0,0 до 25,5.
- Возвращаемое значение: - результат сохранения нового значения.
- Примечание:
    - Заданное значение не сохраняется в энергонезависимую память модуля.
    - Заданное значение не влияет на реальное номинальное напряжение мотора.
    - Значение по умолчанию зависит от типа мотора установленного в модуле.
- Пример:

```
mot.setVoltage(9.2)  # Установить значение которое будет возвращать функция getVoltage() до отключения питания.
```

Функция setNominalRPM()

- Назначение: Установка значения возвращаемого функцией getNominalRPM().
- Синтаксис: setNominalRPM(СКОРОСТЬ)
- Параметр: СКОРОСТЬ - значение от 0 до 65'535 об.мин.
- Возвращаемое значение: - результат сохранения нового значения.
- Примечание:
    - Заданное значение не сохраняется в энергонезависимую память модуля.
    - Заданное значение не влияет на реальную номинальную скорость вращения.

    - Значение по умолчанию зависит от типа мотора и редуктора в модуле.
- Пример:

```
mot.setNominalRPM(1000)  # Установить значение которое будет возвращать функция getNominalRPM() до отключения питания.
```

Функция setMagnet()

- Назначение: Установка количества полюсов многополюсного магнитного вала.
- Синтаксис: setMagnet(КОЛИЧЕСТВО)
- Параметр: КОЛИЧЕСТВО - значение от 1 до 255, или 0 - нет магнита.
- Возвращаемое значение: - результат сохранения нового количества полюсов.
- Примечание:
    - Заданное значение не сохраняется в энергонезависимую память модуля.
    - В качестве параметра указывается количество полюсов одной полярности.
    - Функция может быть полезной, только при смене магнитного вала.
    - Магнитный вал закреплён на роторе мотора возле датчиков Холла, по показаниям которого определяется скорость.
    - Если вызвать функцию с параметром 0 (указать что на роторе мотора нет магнита), то функции: `setSpeed()` , `getSpeed()` , `setStop()` , `getStop()` и `getSum()` не будут работать с параметрами: `MOT_RPM` , `MOT_M_S` , `MOT_MET` и `MOT_REV` , а функция `getError()` не будет возвращать ошибку скорости `MOT_ERR_SPD` .
    - Значение по умолчанию зависит от магнита установленного на ротор мотора.
- Пример:

```
mot.setMagnet(7)  # Указываем что магнитный вал содержит 7 полюсов одной полярности.
```

Функция setReducer()

- Назначение: Установка передаточного отношения редуктора.
- Синтаксис: setReducer(ОТНОШЕНИЕ)
- Параметр: ОТНОШЕНИЕ - значение от 0.01 до 167'772.15.
- Возвращаемое значение: - результат сохранения нового передаточного отношения.

- Примечание:
  - Заданное значение не сохраняется в энергонезависимую память модуля.
  - Если указать отношение равное 1, значит редуктор отсутствует.
  - Если указать значение выше 1, значит редуктор понижающий.
  - Если указать значение меньше 1, значит редуктор повышающий.
  - Функция может быть полезной, только при смене редуктора.
  - Значение по умолчанию зависит от типа редуктора установленного в модуле.
- Пример:

```
mot.setReducer(49.2)  # Указываем передаточное отношение редуктора как 1:49.2
```