**Министерство образования и науки Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**“НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**УНИВЕРСИТЕТ ИТМО”**

**ФАКУЛЬТЕТ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**ОТЧЕТ**

**ПО ИТОГОВОЙ РАБОТЕ**

Специальность 09.02.07 «Информационные системы и программирование»

Дисциплина «Разработка программных модулей»

|  |  |
| --- | --- |
| Преподаватель:  Зенкин А.М.  «14» июня 2021г.  Оценка: | Выполнили:  студенты группы Y2334  Панаёт В. Т., Ганькин В. А.,  Панаёт Р. Т., Котлярова С. Ю. |

Санкт-Петербург

2020/2021

ЦЕЛЬ РАБОТЫ

Изучить и проанализировать модули прошивки потенциостата.

ХОД РАБОТЫ

## potentiostat

#ifndef POTENTIOSTAT\_H

#define POTENTIOSTAT\_H

#include "ps\_system\_state.h"

#endif

Исполняет содержимое файла "ps\_system\_state.h", если был открыт впервые.

## ps\_analog\_subsystem

#ifndef PS\_ANALOG\_SUBSYSTEM\_H

#define PS\_ANALOG\_SUBSYSTEM\_H

#include "ps\_hardware\_defs.h"

#include "ps\_pin\_map.h"

#include "ps\_gains.h"

#include "ps\_volt\_range.h"

#include "ps\_curr\_range.h"

#include "ps\_return\_status.h"

namespace ps

{

    class AnalogSubsystem

    {

        public:

            static const uint16\_t DefaultAnalogWriteResolution = 12;

            static const uint16\_t DefaultAnalogReadResolution = 16;

            static const uint16\_t DefaultAnalogReadAveraging = 16;

            static const uint8\_t  DefaultAnalogReference = INTERNAL;

            static const uint16\_t MaxValueAin = uint16\_t((uint32\_t(1) << DefaultAnalogReadResolution) -1);

            static const uint16\_t MaxValueDac = uint16\_t((uint32\_t(1) << DefaultAnalogWriteResolution)-1);

            static const uint16\_t MidValueDac = MaxValueDac/2;

            AnalogSubsystem();

            void initialize();

            void setVolt(float value);

            float getVolt() const;

            float getCurr() const;

            float getRefElectVolt() const;

            void setVoltRange(VoltRangeDac range);

            VoltRangeDac getVoltRange() const;

            bool autoVoltRange(float minVolt, float maxVolt);

            void setCurrRange(CurrRange range);

            CurrRange getCurrRange() const;

            ReturnStatus setVoltRangeByName(String voltRangeName);

            String getVoltRangeName() const;

            ReturnStatus setCurrRangeByName(String currRangeName);

            String getCurrRangeName() const;

#if defined HARDWARE\_VERSION\_0P2

            void setRefElectVoltRange(VoltRangeAdc range);

            VoltRangeAdc getRefElectVoltRange() const;

            ReturnStatus setRefElectVoltRangeByName(String voltRangeName);

            String getRefElectVoltRangeName() const;

            bool autoRefElectVoltRange(float minVolt, float maxVolt);

#endif

        protected:

            uint16\_t valueDac\_;

            VoltRangeDac voltRange\_;

            CurrRange currRange\_;

            void setVoltGain(VoltGain value);

            VoltGain getVoltGain() const;

            void setCurrGainPath(CurrGainPath value);

            CurrGainPath getCurrGainPath() const;

            String getVoltGainString() const;

            String getCurrGainPathString() const;

            void setValueDac(uint16\_t value);

            uint16\_t getValueDac() const;

            uint16\_t getTransAmpAin() const;

            uint16\_t getRefElectAin() const;

            VoltRangeAdc refElectVoltRange\_;

#if defined HARDWARE\_VERSION\_0P2

            void setRefElectVoltGain(VoltGain value);

            VoltGain getRefElectVoltGain() const;

#endif

    };

} // namespace ps

#endif

Подсистема для работы с вольтажом электродов, током трансимпедансного усилителя и отдельно для работы с вольтажом электрода сравнения (только для второй версии железа). Под железо второй версии расширен функционал. Под каждое железо своя настройка пинов по умолчанию. Настройка по умолчанию происходит при первой инициализации.

### Методы

Для пользователя:

1. void initialize() – инициализация, установка значений по умолчанию.
2. void setVolt(float *value*)установление значения исходящего напряжения на электрод сравнения
3. float getVolt() const – получить текущее значение
4. float getCurr() const – получить измерение тока с рабочего электрожа. Проверить на потенциостате.
5. float getRefElectVolt() const - получить измерение напряжения электрода сравнения
6. void setVoltRange(VoltRangeDac *range*) – установить промежуток исходящего напряжения для электрода сравнения. Изменяет значение усиления напряжения.
7. VoltRangeDac getVoltRange() const – возвращает текущий промежуток напряжения.
8. bool autoVoltRange(float *minVolt*, float *maxVolt*) – автоматическое установление промежутка напряжения. Алгоритм:
9. void setCurrRange(CurrRange *range*) – установить диапазон тока для трансимпедансного усилителя.
10. CurrRange getCurrRange() const – получить текущий диапазон тока для трансимпедансного усилителя.
11. ReturnStatus setVoltRangeByName(String *voltRangeName*) - установить диапазон напряжения по его имени.
12. String getVoltRangeName() const – получить имя диапазона напряжения
13. ReturnStatus setCurrRangeByName(String *currRangeName*) - установить диапазон тока по его имени.
14. String getCurrRangeName() const – получить имя диапазона тока;
15. void setRefElectVoltRange(VoltRangeAdc *range*) - установить диапазон напряжения на аналоговом входе электрода для электрода сравнения;
16. VoltRangeAdc getRefElectVoltRange() const - получить диапазон напряжения на аналоговом входе электрода для электрода сравнения;
17. ReturnStatus setRefElectVoltRangeByName(String *voltRangeName*) - установить диапазон напряжения на аналоговом входе электрода по его имени.
18. String getRefElectVoltRangeName() const--получить имя диапазона напряжения на аналоговом входе электрода для электрода сравнения;
19. bool autoRefElectVoltRange (float *minVolt*, float *maxVolt*) – автоматическая установка диапазона напряжения для электрода сравнения;

Внутренние:

1. void setVoltGain(VoltGain *value*) - установить коэф. усиления для значения напряжения электрода сравнения
2. VoltGain getVoltGain() const – считать усиление. Специфично для версии железа и варианта вольтажа
3. void setCurrGainPath(CurrGainPath *value*) - установить путь усиления. Описание путей усиления ниже.
4. CurrGainPath getCurrGainPath() const - получить текущий путь усиления
5. String getVoltGainString() const - строковое представление
6. String getCurrGainPathString() const -строковое представление
7. void setValueDac(uint16\_t *value*) – установить выходное напряжение с ЦАП
8. uint16\_t getValueDac() const – получить выходное напряжение с ЦАП
9. uint16\_t getTransAmpAin() const - считывание аналогового входа, связанного с трансимпедансным усилителем
10. uint16\_t getRefElectAin() const - считывание аналогового входа, связанного с электродом сравнения
11. void setRefElectVoltGain(VoltGain *value*) – устанавливает усиление для электрода сравнения. Только для второй версии железа.
12. VoltGain getRefElectVoltGain() const – получение текущего усиления. Считывает значения электрода сравнения и проверяет их.

## ps\_base\_test

#ifndef PS\_BASE\_TEST\_H

#define PS\_BASE\_TEST\_H

#include <Arduino.h>

#include "ps\_sample.h"

#include "ps\_constants.h"

#include "ps\_return\_status.h"

#include "third-party/ArduinoJson/ArduinoJson.h"

namespace ps

{

    class BaseTest

    {

        public:

            BaseTest();

            virtual bool isDone(uint64\_t t) const;

            virtual uint64\_t getDoneTime() const;

            virtual void reset();

            virtual float getValue(uint64\_t t) const;

            virtual float getMaxValue() const;

            virtual float getMinValue() const;

            virtual void setQuietTime(uint64\_t quietTime);

            virtual uint64\_t getQuietTime() const;

            virtual void setQuietValue(float value);

            virtual void setQuietValueToStart();

            virtual float getQuietValue() const;

            virtual void setSamplePeriod(uint64\_t samplePeriod);

            virtual uint64\_t getSamplePeriod() const;

            virtual void setName(String name);

            virtual String getName();

            virtual void setSampleMethod(SampleMethod sampleMethod);

            virtual SampleMethod getSampleMethod() const;

            virtual bool updateSample(Sample sampleRaw, Sample &sampleTest);

            virtual void getParam(JsonObject &jsonDat);

            virtual ReturnStatus setParam(JsonObject &jsonMsg, JsonObject &jsonDat);

            virtual bool isMuxCompatible();

            virtual void setMuxCompatible(bool value);

        protected:

            uint64\_t quietTime\_ = 0;

            float quietValue\_ = 0.0;

            uint64\_t samplePeriod\_ = 0;

            uint32\_t sampleModulus\_ = 0;

            String name\_ = String("base");

            SampleMethod sampleMethod\_ = SampleGeneric;

            bool muxCompatible\_ = false;

            JsonObject &getParamJsonObject(JsonObject &json, ReturnStatus &status);

            void setQuietValueFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

            void setQuietTimeFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

    };

}

#endif

Модуль для создания основы теста. Имеет четыре важных параметра - время молчания, значение вольт на время молчания, метод выборки и время между выборками.

### Методы

Для пользователя:

virtual bool isDone(uint64\_t *t*) const - узнать исполнен ли тест

            virtual uint64\_t getDoneTime() const - узнать время исполнения теста

            virtual void reset() - сброс теста

Получение значений:

            virtual float getValue(uint64\_t *t*) const;

            virtual float getMaxValue() const;

            virtual float getMinValue() const;

Установление и получение времени молчания

            virtual void setQuietTime(uint64\_t *quietTime*);

            virtual uint64\_t getQuietTime() const;

Установление и получение значения на время молчания

            virtual void setQuietValue(float *value*);

            virtual void setQuietValueToStart() - в начале следующего теста

            virtual float getQuietValue() const;

Установление и получение периода времени между выборками

            virtual void setSamplePeriod(uint64\_t *samplePeriod*);

            virtual uint64\_t getSamplePeriod() const;

Установление и получение имени теста

            virtual void setName(String *name*);

            virtual String getName();

Установление и получение выборки данных при тесте

            virtual void setSampleMethod(SampleMethod *sampleMethod*);

            virtual SampleMethod getSampleMethod() const;

Редактировать данные

            virtual bool updateSample(Sample *sampleRaw*, Sample &*sampleTest*) - метод не работает. М. б. дописать?

Работа с JSON файлами

            virtual void getParam(JsonObject &*jsonDat*) - получить текущие параметры () в JSON файле

            virtual ReturnStatus setParam(JsonObject &*jsonMsg*, JsonObject &*jsonDat*) - установление параметров из JSON файла

Совместим ли тест с мультиплексором?

            virtual bool isMuxCompatible();

            virtual void setMuxCompatible(bool *value*); установить да или нет

Внутренние:

1. JsonObject &getParamJsonObject(JsonObject &json, ReturnStatus &status) - получить параметр с JSON файла. Обработаны исключения когда подан не JSON файл и когда нет параметров в файле.
2. void setQuietValueFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status) - установить значение для тест из JSON файла. Обработаны исключения, когда тип не float.
3. void setQuietTimeFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status) - установить время на тест из JSON файла. Обработаны исключения, когда тип не unsigned long (время не может быть отрицательным).

## ps\_circular\_buffer

#ifndef PS\_CIRCULAR\_BUFFER\_H

#define PS\_CIRCULAR\_BUFFER\_H

#include <Arduino.h>

#include "third-party/Array/Array.h"

namespace ps

{

    template<typename T, size\_t MAX\_SIZE>

    class CircularBuffer

    {

        public:

            CircularBuffer();

            T& front();

            T& back();

            T& operator[](const size\_t index);

            T  operator[](const size\_t index) const;

            void push\_back(const T &value);

            void push\_front(const T &value);

            void pop\_front();

            void pop\_back();

            void clear();

            bool empty() const;

            bool full() const;

            size\_t size() const;

            size\_t max\_size() const;

            size\_t pos\_front() const;

            size\_t pos\_back() const;

        protected:

            Array<T,MAX\_SIZE+1> data\_;

            volatile size\_t  pos\_front\_ = 0;

            volatile size\_t  pos\_back\_  = 0;

    };

Реализация циркулярного буфера для messege\_receiver.

Циркулярный буфер — это структура данных, использующая единственный буфер фиксированного размера, как будто бы после последнего элемента сразу же снова идет первый. Такая структура легко предоставляет возможность буферизации потоков данных. Перезаписывает наиболее старые данные на новые.

## ps\_command\_table

#ifndef PS\_COMMAND\_TABLE\_H

#define PS\_COMMAND\_TABLE\_H

#include <Arduino.h>

#include "ps\_return\_status.h"

#include "ps\_keyvalue\_command.h"

#include "ps\_constants.h"

#include "third-party/ArduinoJson/ArduinoJson.h"

#include "third-party/Array/Array.h"

namespace ps

{

    template<typename T, size\_t MAX\_SIZE>

    class CommandTable

    {

        public:

            CommandTable(T \*client=nullptr);

            void clear();

            void clearTable();

            size\_t size();

            size\_t maxSize();

            void setClient(T \*client);

            void registerMethod(String key, String value, ReturnStatus (T::\*method)(JsonObject&,JsonObject&));

            ReturnStatus apply(String key, JsonObject &jsonMsg, JsonObject &jsonDat);

        protected:

            T \*client\_;

            Array<KeyValueCommand<T>,MAX\_SIZE> table\_;

    };

реализация командной строки.

#### Методы

1. Table – массив с зарегистрированными методами.
2. clearTable – очистить табло
3. clear – очистить табло и клиента, создавшего табло
4. size – размер табло
5. maxSize – максимальный размер табло (установлен в коде)
6. registerMethod – зарегистрировать метод в табло
7. apply – получить результат команды. Прописаны следующие исключение: клиент не найден, команда не найдена.

## ps\_constant\_test

#ifndef PS\_CONSTANT\_TEST\_H

#define PS\_CONSTANT\_TEST\_H

#include "ps\_base\_test.h"

namespace ps

{

    class ConstantTest : public BaseTest

    {

        public:

            ConstantTest();

            void setDuration(uint64\_t duration);

            uint64\_t getDuration() const;

            void setValue(float value);

            float getValue();

            virtual float getValue(uint64\_t t) const override;

            virtual bool isDone(uint64\_t t) const override;

            virtual uint64\_t getDoneTime() const override;

            virtual float getMaxValue() const override;

            virtual float getMinValue() const override;

            virtual void getParam(JsonObject &jsonDat) override;

            virtual ReturnStatus setParam(JsonObject &jsonMsg, JsonObject &jsonDat) override;

        protected:

            uint64\_t duration\_ = 5000000;

            float value\_ = 1.0;

            void setDurationFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

            void setValueFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

    };

} // namespace ps

#endif

тест, где потенциал между рабочим и эталонным электродами остается постоянным. Как и другие испытания, испытание постоянным напряжением включает в себя период молчания, в течение которого выходное напряжение удерживается, и постоянное значение (напряжение молчания) в течение фиксированной продолжительности до начала испытания.

Параметры:

|  |  |  |
| --- | --- | --- |
| Параметр | Ед. изм | Тип данных |
| время молчания | мс. | uint\_64 |
| напряжение во время молчания | В. | float |
| напряжение | мс. | float |
| продолжительность теста | В. | uint\_64 |

Для пользователя:

ConstantTest() - конструктор, в котором имя теста ставят constant и включают совместимость с мультиплексором.

**Установить и получить продолжительность теста**

* void setDuration(uint64\_t duration);
* uint64\_t getDuration() const;

**Установить и получить исходящее напряжение**

* void setValue(float value);
* float getValue();
* virtual float getValue(uint64\_t t) const override; - получить quiet\_value

**Готов ли тест и за сколько**

* virtual bool isDone(uint64\_t t) const override; - готов ли тест
* virtual uint64\_t getDoneTime() const override; - время выполнения теста

**Максимальное и минимальное значение напряжения**

* virtual float getMaxValue() const override;
* virtual float getMinValue() const override;

**JSON**

* virtual void getParam(JsonObject &jsonDat) override - Получить тек. параметры в JSON файл
* virtual ReturnStatus setParam(JsonObject &jsonMsg, JsonObject &jsonDat) override; - установить параметры из JSON файла

Внутренние:

**Установить продолжительность и значение из JSON файла**

1. void setDurationFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);
2. void setValueFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

## ps\_constants

– константы, используемые в зависимости от версии железа, входного тока и напряжения

## ps\_curr\_range

#ifndef PS\_CURR\_RANGE\_H

#define PS\_CURR\_RANGE\_H

#include <Arduino.h>

#include "ps\_range.h"

#include "ps\_gains.h"

namespace ps

{

    class CurrRange : public Range<CurrGainPath,uint16\_t>

    {

        public:

            CurrRange() : Range<CurrGainPath,uint16\_t>()  {};

            CurrRange(String name, float minValue, float maxValue, CurrGainPath currGainPath, uint16\_t maxInt)

                : Range<CurrGainPath,uint16\_t>(name, minValue, maxValue, currGainPath, maxInt) {};

    };

}

#endif

Range по текущему CurrGainPath

## ps\_device\_id\_eeprom

#ifndef PS\_DEVICE\_ID\_EEPROM\_H

#define PS\_DEVICE\_ID\_EEPROM\_H

#include "ps\_constants.h"

#include "ps\_return\_status.h"

#include "third-party/ArduinoJson/ArduinoJson.h"

namespace ps

{

    class DeviceId\_EEPROM

    {

        public:

            DeviceId\_EEPROM(uint32\_t address=EEPROM\_DeviceIdAddress);

            ReturnStatus set(JsonObject &jsonMsg, JsonObject &jsonDat);

            void get(JsonObject &jsonDat);

        protected:

            uint32\_t address\_;

    };

} // namespace ps

#endif

сохранение в памяти id устройства

## ps\_electrode\_switch

#ifndef PS\_ELECTRODE\_SWITCH\_H

#define PS\_ELECTRODE\_SWITCH\_H

#include "ps\_hardware\_defs.h"

#if defined HARDWARE\_VERSION\_0P2

namespace ps

{

    class ElectrodeSwitch

    {

        public:

            ElectrodeSwitch();

            void initialize();

            bool connected(int pin);

            void setConnected(int pin, bool value);

            bool ctrConnected();

            void setCtrConnected(bool value);

            bool refConnected();

            void setRefConnected(bool value);

            bool wrkConnected();

            void setWrkConnected(bool value);

            bool allConnected();

            void setAllConnected(bool value);

    };

}

#endif // if defined HARDWARE\_VERSION

#endif // ifndef PS\_ELECTRODE\_SWITCH\_H

переключатель и проверка состояния «соединён» электродов. Работает только на второй версии железа.

## ps\_filter

#ifndef PS\_FILTER\_H

#define PS\_FILTER\_H

namespace ps

{

    struct LowPassParam

    {

        float cutoffFreq;   //  cutoff frequency (-3dB) for fileter cascade

        float initialValue;

        unsigned int order;

    };

    // Cascaded first order low-pass filters

    class LowPass

    {

        public:

            static const unsigned int MaxOrder\_ = 5;

            static const unsigned int MinOrder\_ = 1;

            LowPass(float cutoff\_freq=1.0, unsigned int order=1, float value=0.0);

            LowPass(LowPassParam param);

            void setParam(float cutoff\_freq, unsigned int order, float value);

            void setParam(LowPassParam param);

            LowPassParam param();

            float cutoffFreq();

            void setCutoffFreq(float cutoffFreq);

            float initialValue();

            void setInitialValue(float initialValue);

            float order();

            void setOrder(unsigned int order);

            float singleStageRC();

            float singleStageCutoffFreq();

            void update(float value, float dt);

            void reset();

            float value() const;

        protected:

            LowPassParam param\_;

            volatile float state\_[MaxOrder\_+1];

            float rc\_;

            float elemCutoffFreq\_;

            void initializeState();

    };

} // namespace filter

#endif

модуль реализуют фильтр нижних частот.

## ps\_gains

#ifndef PS\_GAINS\_H

#define PS\_GAINS\_H

#include <Arduino.h>

#include "ps\_hardware\_defs.h"

namespace ps

{

#if defined(VOLTAGE\_VARIANT\_AD8250) || defined(VOLTAGE\_VARIANT\_10V)

    enum VoltGain // Analog output voltage scaling factor

    {

        VoltGain1X  = 0,   // [-1V,  +1V]

        VoltGain2X  = 1,   // [-2V,  +2V]

        VoltGain5X  = 2,   // [-5V,  +5V]

        VoltGain10X = 3,   // [-10V, +10V]

        NumVoltGain = 4

    };

    const String VoltGainStringArray[NumVoltGain] =

    {

        String("VoltGain1X"),

        String("VoltGain2X"),

        String("VoltGain5X"),

        String("VoltGain10X")

    };

#elif defined VOLTAGE\_VARIANT\_AD8251

    enum VoltGain // Analog output voltage scaling factor

    {

        VoltGain1X  = 0,   // [-1V,  +1V]

        VoltGain2X  = 1,   // [-2V,  +2V]

        VoltGain4X  = 2,   // [-4V,  +4V]

        VoltGain8X  = 3,   // [-8V,  +8V]

        VoltGain10X = 4,   // [-10V, +10V] // used for reference input

        NumVoltGain = 5

    };

    const String VoltGainStringArray[NumVoltGain] =

    {

        String("VoltGain1X"),

        String("VoltGain2X"),

        String("VoltGain4X"),

        String("VoltGain8X"),

        String("VoltGain10X")

    };

#else

#   error "VOLTAGE\_VARIANT must be specified"

#endif

    enum CurrGainPath // TransImpedance Amplifiler Current gain path

    {

        CurrGainPathIn1 = 0,  // [-1uA,    +1uA]    w/ default resistors

        CurrGainPathIn2 = 1,  // [-10uA,   +10uA]

        CurrGainPathIn3 = 2,  // [-100uA,  +100uA]

        CurrGainPathIn4 = 3,  // [-1000uA, +1000uA]

        CurrGainPathErr = 4,  // Incorrect path setting

        NumCurrGainPath = 5

    };

    const String CurrGainPathStringArray[NumCurrGainPath] =

    {

        String("CurrGainPathIn1"),

        String("CurrGainPathIn2"),

        String("CurrGainPathIn3"),

        String("CurrGainPathIn4"),

        String("CurrGainPathErr")

    };

    String voltGainToString(VoltGain value);

    String currGainPathToString(CurrGainPath value);

}

#endif

методы для строковой презентации коэффициентов усиления. Используется в аналоговой подсистеме.

## ps\_multistep\_test

#ifndef PS\_MULTISTEP\_TEST\_H

#define PS\_MULTISTEP\_TEST\_H

#include "ps\_base\_test.h"

#include "ps\_constants.h"

#include "ps\_time\_utils.h"

#include "third-party/Array/Array.h"

#include "third-party/ArduinoJson/ArduinoJson.h"

namespace ps

{

    template<size\_t MAX\_SIZE>

    class MultiStepTest : public BaseTest

    {

        public:

            //static const String StepArrayKey;

            MultiStepTest(size\_t numStep=5);

            virtual void setStepValue(size\_t n, float value);

            virtual float getStepValue(size\_t n) const;

            virtual void setStepDuration(size\_t n, uint64\_t duration);

            virtual uint64\_t getStepDuration(size\_t n) const;

            virtual uint64\_t getDuration() const;

            virtual void setNumStep(size\_t numStep);

            virtual size\_t getNumStep() const;

            virtual size\_t getMaxNumStep() const;

            virtual bool isDone(uint64\_t t) const override;

            virtual uint64\_t getDoneTime() const override;

            virtual void reset() override;

            virtual float getValue(uint64\_t t) const override;

            virtual float getMaxValue() const override;

            virtual float getMinValue() const override;

            virtual void getParam(JsonObject &jsonDat) override;

            virtual ReturnStatus setParam(JsonObject &jsonMsg, JsonObject &jsonDat) override;

        protected:

            Array<float, MAX\_SIZE> valueArray\_;

            Array<uint64\_t, MAX\_SIZE> durationArray\_;

            size\_t numStep\_;

            void setValueAndDurationFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

    };

тест, где каждое изменение напряжения задается шагом.

Параметры:

|  |  |  |
| --- | --- | --- |
| Параметр | Ед. изм | Тип данных |
| время молчания | мс | uint\_64 |
| напряжение молчания | В | float |
| шаг\* - (продолжительность, напряжение) | (мс, В) | (uint\_64, float) |

\*макс кол-во шагов = 50

#### Методы:

Для пользователя:

MultiStepTest(size\_t numStep=5) - конструктор, в котором имя теста ставят multiStep, готовят массивы значений и продолжительности, устанавливают количество шагов (по умолчанию 5) и включают совместимость с мультиплексором.

**Установить или получить значение в шаге**

* virtual void setStepValue(size\_t n, float value)
* virtual float getStepValue(size\_t n) const

**Установить или получить продолжительность в шаге**

* virtual void setStepDuration(size\_t n, uint64\_t duration);
* virtual uint64\_t getStepDuration(size\_t n) const;

**Получить продолжительность теста**

* virtual uint64\_t getDuration() const;

**Установить или получить кол-во шагов**

* virtual void setNumStep(size\_t numStep);
* virtual size\_t getNumStep() const;

**Получить максимально возможное кол-во шагов**

* virtual size\_t getMaxNumStep() const;

**Готов ли тест и за сколько**

* virtual bool isDone(uint64\_t t) const override; - готов ли тест
* virtual uint64\_t getDoneTime() const override; - время исполнения теста

**Сброс**

* virtual void reset() override

**Получить текущее значение**

* virtual float getValue(uint64\_t t) const override

**Максимальное и минимальное значение**

* virtual float getMaxValue() const override;
* virtual float getMinValue() const override;

**JSON**

* virtual void getParam(JsonObject &jsonDat) override - получить параметры в JSON файл
* virtual ReturnStatus setParam(JsonObject &jsonMsg, JsonObject &jsonDat) override - установить параметры из JSON файла - value and duration

Внутренние:

**Установить значение и продолжительность из JSON файла.**

Обработаны след. исключения:

* для кол-ва шагов:
  + не найдено кол-во шагов;
  + на вход подан не массив;
  + массив слишком большой
* для value and duration:
  + продолжительность не int;
  + значение не float;
  + размер массива не 2;
  + на вход подан не массив.

1. void setValueAndDurationFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

## ps\_periodic\_test

#ifndef PS\_PERIODIC\_TEST\_H

#define PS\_PERIODIC\_TEST\_H

#include "ps\_base\_test.h"

#include "ps\_constants.h"

namespace ps

{

    class PeriodicTest : public BaseTest

    {

        public:

            static constexpr float DefaultAmplitude = 1.0;

            static constexpr float DefaultOffset = 0.0;

            static constexpr float DefaultShift = 0.0;

            static constexpr uint64\_t DefaultPeriod = UINT64\_C(1000000);

            static constexpr uint32\_t DefaultNumCycles = UINT32\_C(10);

            PeriodicTest();

            virtual void setAmplitude(float amplitude);

            virtual float getAmplitude() const;

            virtual void setOffset(float offset);

            virtual float getOffset() const;

            virtual void setPeriod(uint64\_t period);

            virtual uint64\_t getPeriod() const;

            virtual void setNumCycles(uint32\_t numCycles);

            virtual uint32\_t getNumCycles() const;

            virtual void setShift(float lag);

            virtual float getShift() const;

            virtual uint32\_t getCycleCount(uint64\_t t) const;

            virtual bool isDone(uint64\_t t) const override;

            virtual uint64\_t getDoneTime() const override;

            virtual float getValue(uint64\_t t) const override;

            virtual float getMaxValue() const override;

            virtual float getMinValue() const override;

            virtual void getParam(JsonObject &jsonDat) override;

            virtual ReturnStatus setParam(JsonObject &jsonMsg, JsonObject &jsonDat) override;

        protected:

            float amplitude\_ = DefaultAmplitude;     // 12-bit Dac int

            float offset\_ = DefaultOffset;           // 12-bit Dac int

            uint64\_t period\_ = DefaultPeriod;        // Waveform period (us)

            uint32\_t numCycles\_ = DefaultNumCycles;  // Number of cycles to perform

            float shift\_ = DefaultShift;             // Waveform shift as fraction of period [0,1]

            uint64\_t shiftInUs\_ = 0;                 // Waveform shift in us;

            void setAmplitudeFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

            void setOffsetFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

            void setPeriodFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

            void setNumCyclesFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

            void setShiftFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

            void updateShiftInUs();

    };

}

#endif

Параметры:

|  |  |  |
| --- | --- | --- |
| Параметр | Ед. изм | Тип данных |
| время молчания | мс | uint\_64 |
| напряжение молчания | В | float |
| амплитуда | В | float |
| смещение по y | В | float |
| период | мс | uint\_64 |
| кол-во циклов | шт. | uint\_32 |
| фазовый сдвиг (смещение по x) |  | float |

#### Методы

**Для пользователя:**

PeriodicTest::PeriodicTest() - конструктор, в котором имя теста = periodic и рассчитывается перемещение в секундах.

Установить или получить амплитуду

* virtual void setAmplitude(float amplitude);
* virtual float getAmplitude() const;

Установить или получить смещение (относительно оси ординат)

* virtual void setOffset(float offset);
* virtual float getOffset() const;

Установить или получить период

* virtual void setPeriod(uint64\_t period);
* virtual uint64\_t getPeriod() const;

Установить или получить кол-во итераций

* virtual void setNumCycles(uint32\_t numCycles);
* virtual uint32\_t getNumCycles() const;

Установить или получить смещение относительно оси абсцисс

* virtual void setShift(float lag);
* virtual float getShift() const;

Текущая итерация

* virtual uint32\_t getCycleCount(uint64\_t t) const;

Готов ли тест и за сколько

* virtual bool isDone(uint64\_t t) const override
* virtual uint64\_t getDoneTime() const override
* virtual float getValue(uint64\_t t) const override; - получение текущего значения. Создан для обязательной инициализации в классах-наследниках.

Мин и Макс знач. напряжения

* virtual float getMaxValue() const override;
* virtual float getMinValue() const override;

Скинуть параметры в JSON файл

* virtual void getParam(JsonObject &jsonDat) override;

Установить параметры из JSON файла

* virtual ReturnStatus setParam(JsonObject &jsonMsg, JsonObject &jsonDat) override;

**Внутренние:**

Установить амплитуду, смещение, период, кол-во циклов, перемещение из JSON файла. Прописаны исключения на тип и для перемещения на значение [между 0 и 1]

* 1. void setAmplitudeFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);
  2. void setOffsetFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);
  3. void setPeriodFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);
  4. void setNumCyclesFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);
  5. void setShiftFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);
  6. void updateShiftInUs() - смещение по x в секундах

## ps\_cyclic\_test.h

#ifndef PS\_CYCLIC\_TEST\_H

#define PS\_CYCLIC\_TEST\_H

#include <Arduino.h>

#include "ps\_periodic\_test.h"

namespace ps

{

    class CyclicTest : public PeriodicTest

    {

        public:

            CyclicTest();

            virtual float getValue(uint64\_t t) const override;

    };

} // namespace ps

#endif

Наследник periodic\_test. Тест, где потенциал между рабочим и опорным электродами циклически наращивается вверх и вниз кусочно-линейным способом - в треугольной форме волны.

Параметры:

|  |  |  |
| --- | --- | --- |
| Параметр | Ед. изм | Тип данных |
| время молчания | мс | uint\_64 |
| напряжение молчания | В | float |
| амплитуда | В | float |
| смещение по y | В | float |
| период | мс | uint\_64 |
| кол-во циклов | шт. | uint\_32 |
| фазовый сдвиг (смещение по x) |  | float |

#### Методы

**Для пользователя:**

CyclicTest() - - конструктор, в котором имя теста = cyclic и multiplexer вкл.

virtual float getValue(uint64\_t t) const override - получение текущего значения.

## ps\_pin\_map

#ifndef PS\_PIN\_MAP\_H

#define PS\_PIN\_MAP\_H

#include <Arduino.h>

#include "ps\_hardware\_defs.h"

namespace ps

{

#if defined HARDWARE\_VERSION\_0P1

    const int AD8250\_GAIN\_A0 = 0;

    const int AD8250\_GAIN\_A1 = 1;

    const int TIA\_SW1\_IN1 = 2;

    const int TIA\_SW1\_IN2 = 5;

    const int TIA\_SW1\_IN3 = 6;

    const int TIA\_SW1\_IN4 = 7;

    const int TIA\_SW2\_IN1 = 8;

    const int TIA\_SW2\_IN2 = 9;

    const int TIA\_SW2\_IN3 = 22;

    const int TIA\_SW2\_IN4 = 23;

#elif defined HARDWARE\_VERSION\_0P2

    const int DAC\_GAIN\_A0 = 0;

    const int DAC\_GAIN\_A1 = 1;

    const int TIA\_GAIN\_A0 = 5;

    const int TIA\_GAIN\_A1 = 6;

    const int REF\_GAIN\_A0 = 20;

    const int REF\_GAIN\_A1 = 21;

    const int SW\_CTR\_ELECT = 7;

    const int SW\_REF\_ELECT = 8;

    const int SW\_WRK\_ELECT = 9;

#else

#   error "HARDWARE\_VERSION must be specified"

#endif

    const int DAC\_UNI\_PIN = A14;

    const int TIA\_OUT\_UNI\_PIN = A1;

    const int REF\_ELECT\_UNI\_PIN = A2;

    // Multiplexer switch pins

    const int MUX\_WRK1\_TO\_TIA = 24;

    const int MUX\_WRK1\_TO\_GND = 28;

    const int MUX\_WRK2\_TO\_TIA = 25;

    const int MUX\_WRK2\_TO\_GND = 29;

    const int MUX\_WRK3\_TO\_TIA = 26;

    const int MUX\_WRK3\_TO\_GND = 30;

    const int MUX\_WRK4\_TO\_TIA = 27;

    const int MUX\_WRK4\_TO\_GND = 31;

    const int MUX\_WRK5\_TO\_TIA = 4;

    const int MUX\_WRK5\_TO\_GND = 11;

    const int MUX\_WRK6\_TO\_TIA = 19;

    const int MUX\_WRK6\_TO\_GND = 13;

    const int MUX\_WRK7\_TO\_TIA = 18;

    const int MUX\_WRK7\_TO\_GND = 10;

    const int MUX\_CTR\_CONN = 3;

    const int MUX\_REF\_CONN = 12;

}

#endif

нумерование пинов для изменения их состояния в коде.

## ps\_hardware\_defs

//защита от повторного включения файла

#ifndef PS\_HARDWARE\_DEFS\_H

#define PS\_HARDWARE\_DEFS\_H

    #if 1

        // Hardware version 0.2

        // -----------------------------------------------------------------------

        #define HARDWARE\_VERSION\_0P2

        #define VOLTAGE\_VARIANT\_10V

        #define CURRENT\_VARIANT\_MICRO\_AMP   // select from (NANO, MICRO, MILL)

    #else

        // Hardware version 0.1

        // -----------------------------------------------------------------------

        #define HARDWARE\_VERSION\_0P1

        #define VOLTAGE\_VARIANT\_AD8250      // select from (AD8250 or AD8251)

        #define CURRENT\_VARIANT\_MICRO\_AMP   // select from (NANO, MICRO or MILL)

    #endif

#endif

Описание файла: Прописана защита от повторного включения файла и версия hardware(оборудование)

## ps\_keyvalue\_command

#ifndef PS\_KEYVALUE\_COMMAND\_H

#define PS\_KEYVALUE\_COMMAND\_H

#include "ps\_constants.h"

namespace ps

{

    template<typename T>

    class KeyValueCommand

    {

        public:

            KeyValueCommand() {};

            KeyValueCommand(String key, String value, ReturnStatus (T::\*method)(JsonObject&,JsonObject&));

            String key();

            void setKey(String key);

            String value();

            void setValue(String value);

            void setMethod(ReturnStatus (T::\*method)(JsonObject&,JsonObject&));

            ReturnStatus applyMethod(T\* client, JsonObject &jsonMsg, JsonObject &jsonDat);

        protected:

            String key\_;

            String value\_;

            ReturnStatus (T::\*method\_)(JsonObject&,JsonObject&) = nullptr;

    };

Описание файла: класс,через который происходит ввод ключевых команд в Командую строку(Command Table).  
Имеет три атрибута:

String key\_;

String value\_;

ReturnStatus (T::\*method\_)(JsonObject&,JsonObject&) = nullptr;

Для всех них прописаны геттеры и сеттеры.

## ps\_linearsweep\_test

#ifndef PS\_LINEARSWEEP\_TEST\_H

#define PS\_LINEARSWEEP\_TEST\_H

#include "ps\_base\_test.h"

namespace ps

{

    class LinearSweepTest : public BaseTest

    {

        public:

            LinearSweepTest();

            void setStartValue(float value);

            float getStartValue() const;

            void setFinalValue(float value);

            float getFinalValue() const;

            void setDuration(uint64\_t duration);

            uint64\_t getDuration() const;

            virtual bool isDone(uint64\_t t) const override;

            virtual uint64\_t getDoneTime() const override;

            virtual float getValue(uint64\_t t) const override;

            virtual float getMaxValue() const override;

            virtual float getMinValue() const override;

            virtual void getParam(JsonObject &jsonDat) override;

            virtual ReturnStatus setParam(JsonObject &jsonMsg, JsonObject &jsonDat) override;

        protected:

            float startValue\_ = -0.5;

            float finalValue\_ =  0.5;

            uint64\_t duration\_ = 2000000;

            void setStartValueFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

            void setFinalValueFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

            void setDurationFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

    };

} // namespace ps

#endif

Тест линейной развертки

Параметры:

|  |  |  |
| --- | --- | --- |
| Параметр | Ед. изм | Тип данных |
| Стартовое значение | В. | float |
| Финальное значение | В. | float |
| продолжительность теста | Сек. | uint\_64 |

#### Методы:

Для пользователя:

**Установить или получить стартового значения**

* void setStartValue(float value);
* float getStartValue() const;

**Установить или получить конечного значения**

* void setFinalValue(float value);
* float getFinalValue() const;

**Получить или установить продолжительность теста**

* void setDuration(uint64\_t duration);
* uint64\_t getDuration() const;

**Установить или получить кол-во шагов**

* virtual void setNumStep(size\_t numStep);
* virtual size\_t getNumStep() const;

**Получить максимально возможное кол-во шагов**

* virtual size\_t getMaxNumStep() const;

**Готов ли тест и за сколько**

* virtual bool isDone(uint64\_t t) const override; - готов ли тест
* virtual uint64\_t getDoneTime() const override; - время исполнения теста

**Получить текущее значение**

* virtual float getValue(uint64\_t t) const override

**Максимальное и минимальное значение**

* virtual float getMaxValue() const override;
* virtual float getMinValue() const override;

**JSON**

* virtual void getParam(JsonObject &jsonDat) override - получить параметры в JSON файл
* virtual ReturnStatus setParam(JsonObject &jsonMsg, JsonObject &jsonDat) override - установить параметры из JSON файла - value and duration

Внутренние:

**Установить значение и продолжительность из JSON файла.**

void setStartValueFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

void setFinalValueFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

void setDurationFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

Обработаны след. исключения:

* для кол-ва шагов:
  + не найдено кол-во шагов;
  + на вход подан не массив;
  + массив слишком большой
* для value and duration:
  + продолжительность не int;
  + значение не float;
  + размер массива не 2;
  + на вход подан не массив.

## ps\_lookup\_table

#ifndef PS\_LOOKUP\_TABLE\_H

#define PS\_LOOKUP\_TABLE\_H

#include "third-party/Array/Array.h"

namespace ps

{

    template<typename T, size\_t SIZE>

    class LookupTable

    {

        public:

            LookupTable();

            T& operator[](const size\_t ind);

            T  operator[](const size\_t ind) const;

            T getValue(T pos) const;  // pos >= 0  and pos < SIZE - 1

        protected:

            Array<T,SIZE> data\_;

    };

Описание файла: Файл, в котором объявлен класс LookupTable. Этот класс используется для того чтобы просматривать данные класса Array. В нём есть перегруженный оператор [] для удобного обращения, так же это можно сделать методом getValue, передав туда индекс.

## ps\_message\_parser

#ifndef PS\_MESSAGE\_PARSER\_H

#define PS\_MESSAGE\_PARSER\_H

#include <Arduino.h>

#include "ps\_constants.h"

#include "third-party/ArduinoJson/ArduinoJson.h"

namespace ps

{

    class MessageParser

    {

        public:

            MessageParser();

            JsonObject& parse(String &message, StaticJsonBuffer<JsonMessageBufferSize> &jsonBuffer);

    };

} // namespace ps

#endif

Описание файла: переводит сообщение типа String в JSON объект

## ps\_message\_receiver

#ifndef PS\_MESSAGE\_RECEIVER\_H

#define PS\_MESSAGE\_RECEIVER\_H

#include <Arduino.h>

#include "ps\_constants.h"

#include "ps\_circular\_buffer.h"

namespace ps

{

    class MessageReceiver

    {

        public:

            MessageReceiver();

            void reset();

            void readData();

            String next();

            bool available() const;

            uint32\_t getMessageCnt() const;

            uint32\_t getTotalMessageCnt() const;

        protected:

            CircularBuffer<char,SerialBufferSize> serialBuffer\_;

            bool overflow\_ = false;

            uint32\_t messageCnt\_ = 0;

            uint32\_t totalMessageCnt\_ = 0;

    };

} // namespace ps

#endif

Описание файла:   
Класс Message receiver отвечает за принятие сообщения. Состоит из ранее упомянутого циркулярного буфера, флага заполнения и счетчика принятых сообщений без переполнения и счётчика всех принятых сообщений.

В нем есть след методы:

void reset(); - обнуляет кол-во принятых сообщений без переполнения и снимает флаг переполнения

void readData(); - посимвольно принимает информацию

String next(); - копирует данные из буфера в строку

bool available() const; - проверка доступности

uint32\_t getMessageCnt() const; - возвращает счётчик принятых сообщений без переполнения

uint32\_t getTotalMessageCnt() const; - возвращает счётчик всех принятых сообщений.

## ps\_message\_sender

#ifndef PS\_MESSAGE\_SENDER\_H

#define PS\_MESSAGE\_SENDER\_H

#include <Arduino.h>

#include "ps\_constants.h"

#include "ps\_sample.h"

#include "ps\_return\_status.h"

#include "third-party/ArduinoJson/ArduinoJson.h"

namespace ps

{

    class MessageSender

    {

        public:

            MessageSender();

            void sendSample(Sample sample);

            void sendSampleEnd();

            void sendCommandResponse(ReturnStatus status, JsonObject &jsonDat);

    };

} // namespace ps

#endif

Описание файла: Класс Message sender отвечает за запись данных в JSON объект.

В нем есть след методы:

void sendSample(Sample sample); - данные результата теста для отправки

void sendCommandResponse(ReturnStatus status, JsonObject &jsonDat); -

## ps\_multiplexer

#ifndef PS\_MULTIPLEXER\_H

#define PS\_MULTIPLEXER\_H

#include "ps\_pin\_map.h"

#include "ps\_constants.h"

#include "ps\_return\_status.h"

#include "third-party/Array/Array.h"

namespace ps

{

    class Multiplexer

    {

        public:

            static const int MuxSwitchPin[NumMuxPin];

            static const int MuxToTiaPin[NumMuxChan];

            static const int MuxToGndPin[NumMuxChan];

            static const int NotConnected = -1;

            Multiplexer();

            void setupSwitchPins();

            void clearSwitchPins();

            void connectCtrElect();

            void disconnectCtrElect();

            void connectRefElect();

            void disconnectRefElect();

            void connectWrkElect(int electNum);

            void disconnectWrkElect();

            int currentWrkElect();

            bool isConnectedWrk();

            bool isConnectedCtr();

            bool isConnectedRef();

            void connectFirstEnabledWrkElect();

            void connectNextEnabledWrkElect();

            void start();

            void stop();

            bool isRunning();

            void enableWrkElect(int electNum);

            void disableWrkElect(int electNum);

            void enableAllWrkElect();

            void disableAllWrkElect();

            void setEnabledWrkElect(Array<int,NumMuxChan> enabledArray);

            Array<int,NumMuxChan> getEnabledWrkElect();

            bool isWrkElectEnabled(int electNum);

            int numEnabledWrkElect();

            int electNumToIndex(int electNum);

            int indexToElectNum(int index);

        protected:

            bool running\_ = false;

            volatile int currWrkElect\_ = NotConnected;

            int numEnabled\_ = 0;

            Array<bool,NumMuxChan> enabledTable\_;

            void setAllChanToGnd();

            void initializeEnabledTable(bool value);

            int countNumEnabled();

    };

Описание файла: Реализация мультиплексора.

Mультипле́ксор — устройство, им0еющее несколько сигнальных входов, один или более управляющих входов и один выход. Мультиплексор позволяет передавать сигнал с одного из нескольких входов на один выход; при этом выбор желаемого входа осуществляется подачей соответствующей комбинации управляющих сигналов.  
  
Методы:

Публичные:

void setupSwitchPins(); - установка контактов переключателя мультиплексора

void clearSwitchPins(); - очистка контактов переключателя мультиплексора

void connectCtrElect(); - подключение вспомогательного электрода к трансимпедантному усилителю

void disconnectCtrElect(); - отключение вспомогательного электрода от трансимпедансного усилителя

void connectRefElect(); - подключение сравнительного(референсный) электрода к трансимпедантному усилителю

void disconnectRefElect(); - отключение сравнительного(референсный) электрода от трансимпедансного усилителя

void connectWrkElect(int electNum); - Подключение рабочего электрода к трансимпедантному усилителю

void disconnectWrkElect(); - Отсоединение рабочего электрода от трансимпедансного усилителя

int currentWrkElect(); - возвращает текущий работающий электрод

bool isConnectedWrk(); - подключён ли работающий электрод

bool isConnectedCtr(); - подключён ли вспомогательный электрод

bool isConnectedRef(); - подключён ли сравнительный(референсный) электрод

void connectFirstEnabledWrkElect(); - подключение первого включенного рабочего электрода

void connectNextEnabledWrkElect(); - подключение следующего(после первого) включенного рабочего электрода

void start(); - включение мультиплексора

void stop(); - выключение мультиплексора

bool isRunning(); - работает ли мультиплексор?

void enableWrkElect(int electNum); - включение рабочего электода по номеру электрода

void disableWrkElect(int electNum); - выключение рабочего электорда по номеру электрода

void enableAllWrkElect(); - включение всех рабочих электродов

void disableAllWrkElect(); - выключение всех рабочих электородов

void setEnabledWrkElect(Array<int,NumMuxChan> enabledArray); - включение всех рабочих электродов из массива

Array<int,NumMuxChan> getEnabledWrkElect(); - возвращает массив включенных рабочих электродов

bool isWrkElectEnabled(int electNum); - включён ли рабочий электрод

int numEnabledWrkElect(); - кол-во включенных рабочих электродов

int electNumToIndex(int electNum); - перевод номера электрода в индекс для массива

int indexToElectNum(int index); - перевод индекса в номер электрода

Защищённые:

void setAllChanToGnd(); - установка всех каналов к общему проводу

void initializeEnabledTable(bool value); - иницализация таблицы включённых электродов

int countNumEnabled(); - подсчёт включенных электродов

## ps\_range.h

#ifndef PS\_RANGE\_H

#define PS\_RANGE\_H

namespace ps

{

    template<typename GainType, typename IntType>

    class Range

    {

        public:

            Range() {};

            Range(String name, float minValue, float maxValue, GainType gain, IntType maxInt);

            String name() const;

            float minValue() const;

            float maxValue() const;

            GainType gain() const;

            IntType maxInt() const;

            inline IntType valueToInt(float volt) const;

            inline float intToValue(IntType value) const;

        private:

            String name\_;

            float minValue\_;

            float maxValue\_;

            GainType gain\_;

            IntType maxInt\_;

    };

Модуль описывает шаблон класса Range и шаблоны функций:

name() - возвращает имя;

minValue() - возвращает минимальное значение;

maxValue() - возвращает максимальное значение;

gain() - возвращает коэффициент усиления;

maxInt() - возвращает maxInt;

valueToInt() - приводит передаваемое значение volt к int типу + возвращает значение ограниченное 0**≤**value**≤**maxInt\_:

IntType value = IntType(float(maxInt\_)/(maxValue\_ - minValue\_)\*(maxValue\_ - volt));

intToValue() - обратная по отношению к valueToInt операция.

## ps\_return\_status

#ifndef PS\_RETURN\_STATUS\_H

#define PS\_RETURN\_STATUS\_H

#include <Arduino.h>

namespace ps

{

    class ReturnStatus

    {

        public:

            ReturnStatus() { };

            void appendToMessage(String value);

            bool success = true;;

            String message;

    };

} // namespace ps

#endif

сохраняет статус записывая в переменную message статусы тестов через запятую.

## ps\_sample.h

#ifndef PS\_SAMPLE\_H

#define PS\_SAMPLE\_H

namespace ps

{

    class Sample

    {

        public:

            uint64\_t t;

            float volt;

            float curr;

            uint8\_t chan;

    };

}

#endif

## ps\_sinusoid\_test

#ifndef PS\_SINUSOID\_TEST\_H

#define PS\_SINUSOID\_TEST\_H

#include "ps\_periodic\_test.h"

#include "ps\_lookup\_table.h"

#include <Arduino.h>

namespace ps

{

    class SinusoidTest : public PeriodicTest

    {

        public:

            static constexpr uint32\_t LookupTableSize = 300;

            SinusoidTest();

            virtual void setAmplitude(float amplitude) override;

            virtual void setOffset(float offset) override;

            virtual float getValue(uint64\_t t) const override;

            void updateLookupTable();

        protected:

            LookupTable<float,LookupTableSize> lookupTable\_;

    };

} // namespace ps

#endif

Наследник periodic\_test. Этот тест проверяет, является ли ток в потенциостате синусоидальным.

|  |  |  |
| --- | --- | --- |
| Параметр | Ед. изм | Тип данных |
| время молчания | мс | uint\_64 |
| напряжение молчания | В | float |
| амплитуда | В | float |
| смещение по y | В | float |
| период | мс | uint\_64 |
| кол-во циклов | шт. | uint\_32 |
| фазовый сдвиг (смещение по x) | В | float |

#### Методы

SinusoidTest(); - конструктор, в котором устанавливается название теста (sinusoid) и включается поддержка мультиплексера.

virtual void setAmplitude(float amplitude) override; - сеттер(установка значения) для амплитуды

virtual void setOffset(float offset) override; - сеттер(установка значения) для смещения по y

virtual float getValue(uint64\_t t) const override; - получение значения в период времени

## 

## ps\_squarewave\_test

Тест прямоугольной вольтамперометрии.

#ifndef PS\_SQUAREWAVE\_TEST\_H

#define PS\_SQUAREWAVE\_TEST\_H

#include "ps\_base\_test.h"

namespace ps

{

    class SquareWaveTest : public BaseTest

    {

        public:

            SquareWaveTest();

            void setStartValue(float value);

            float getStartValue();

            void setFinalValue(float value);

            float getFinalValue();

            void setStepValue(float value);

            float getStepValue();

            void setAmplitude(float value);

            float getAmplitude();

            void setWindow(float value);

            float getWindow();

            virtual bool isDone(uint64\_t t) const override;

            virtual uint64\_t getDoneTime() const override;

            virtual void reset();

            virtual float getValue(uint64\_t t) const override;

            virtual float getStairValue(uint64\_t t) const;

            virtual float getMaxValue() const override;

            virtual float getMinValue() const override;

            virtual void setSamplePeriod(uint64\_t samplePeriod) override;

            virtual bool updateSample(Sample sampleRaw, Sample &sampleTest);

            virtual void getParam(JsonObject &jsonDat) override;

            virtual ReturnStatus setParam(JsonObject &jsonMsg, JsonObject &jsonDat) override;

        protected:

            float startValue\_ = -0.5;

            float finalValue\_ = 0.5;

            float stepValue\_ = 0.005;

            float amplitude\_ = 0.025;

            float window\_ = 0.2;

            float maxValue\_ = 0.0;

            float minValue\_ = 0.0;

            float stepSign\_ = 1.0;

            uint64\_t doneTime\_ = 0;

            uint64\_t halfSamplePeriod\_ = 0;

            uint64\_t windowLenUs\_ = 0;

            bool isFirst\_ = true;

            uint64\_t testCnt\_ = 0;

            uint64\_t numForward\_ = 0;

            uint64\_t numReverse\_ = 0;

            float currForward\_ = 0.0;

            float currReverse\_ = 0.0;

            void updateDoneTime();

            void updateMaxMinValues();

            void updateWindowLenUs();

            void updateStepSign();

            void setStartValueFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

            void setFinalValueFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

            void setStepValueFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

            void setAmplitudeFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

            void setWindowFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status);

    };

} // namespace ps

#endif

Параметры:

|  |  |  |
| --- | --- | --- |
| Параметр | Ед. изм | Тип данных |
| Стартовое значение | В. | float |
| Финальное значение | В. | float |
| Шаг | (мс, В) | float |
| Амплитуда | В. | float |
| Окно | % | float |
| Продолжительность теста | Сек. | uint\_64 |

Методы:

SquareWaveTest() - конструктор, в котором устанавливается название теста (squareWave) и отключается поддержка мультиплексера.

setStartValue(float value) - устанавливает стартовое значение;

getStartValue() - получает стартовое значение;

setFinalValue(float value), getFinalValue() - установить/получить финальное значение;

setStepValue(float value), getStepValue() - установить/получить значение шага;

setAmplitude(float value), getAmplitude() - установить/получить амплитуду;

setWindow(float value), getWindow() - установить/получить окно;

getMaxValue() const, getMinValue() const - получить максимальное/минимальное значение;

setSamplePeriod(uint64\_t SamplePeriod) - установить период выборки;

isDone(uint64\_t t) const - провера выполненности теста;

getDoneTime() const - получить время выполнения;

reset() - сброс всех значений к нулю;

getValue(uint64\_t t) const - получение значений;

getStairValue(uint64\_t t) const - получение значения ступени;

updateSample(Sample sampleRaw, Sample &sampleTest) - обновить образец;

getParam(JsonObject &jsonDat), setParam(JsonObject &jsonMsg, JsonObject &jsonDat) - получить/установить значения из json файла;

updateDoneTime() - время обновления данных;

updateMaxMinValues() - обновить минимальное и максимальное значения;

updateWindowLenUs() - обновление значения окна;

updateStepSign() - изменяет знак шага(+/-);

setStartValueFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status), setFinalValueFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status) - устанавливает начальное и финальное значение из json файла;

setStepValueFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status) - устанавливает значение шага из json файла;

setAmplitudeFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status) - устанавливает амплитуду из json файла;

setWindowFromJson(JsonObject &jsonMsgPrm, JsonObject &jsonDatPrm, ReturnStatus &status) - устанавливает окно из json файла;

## ps\_system\_state.cpp

#ifndef PS\_SYSTEM\_STATE\_H

#define PS\_SYSTEM\_STATE\_H

#include <Arduino.h>

#include "ps\_hardware\_defs.h"

#include "ps\_constants.h"

#include "ps\_return\_status.h"

#include "ps\_analog\_subsystem.h"

#include "ps\_electrode\_switch.h"

#include "ps\_message\_receiver.h"

#include "ps\_message\_sender.h"

#include "ps\_message\_parser.h"

#include "ps\_command\_table.h"

#include "ps\_circular\_buffer.h"

#include "ps\_voltammetry.h"

#include "ps\_sample.h"

#include "ps\_filter.h"

#include "ps\_multiplexer.h"

#include "third-party/Array/Array.h"

#define ARDUINOJSON\_USE\_DOUBLE 0

#include "third-party/ArduinoJson/ArduinoJson.h"

namespace ps

{

    class SystemState

    {

        public:

            SystemState();

            void initialize();

            void processMessages();

            void updateMessageData();

            void serviceDataBuffer();

            ReturnStatus onCommandRunTest(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandStopTest(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetVolt(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetVolt(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetCurr(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetRefVolt(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetTestParam(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetTestParam(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetVoltRange(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetVoltRange(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetCurrRange(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetCurrRange(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetDeviceId(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetDeviceId(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetSamplePeriod(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetSamplePeriod(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetTestDoneTime(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetTestNames(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetVersion(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetVariant(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetMuxEnabled(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetMuxEnabled(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetEnabledMuxChan(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetEnabledMuxChan(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetMuxTestNames(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetMuxRefElectConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetMuxRefElectConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetMuxCtrElectConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetMuxCtrElectConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetMuxWrkElectConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetMuxWrkElectConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandDisconnAllMuxElect(JsonObject &jsonMsg, JsonObject &jsonDat);

#if defined HARDWARE\_VERSION\_0P2

            ReturnStatus onCommandSetRefElectConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetRefElectConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetCtrElectConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetCtrElectConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetWrkElectConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetWrkElectConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetAllElectConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetAllElectConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetElectAutoConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetElectAutoConn(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandSetRefElectVoltRange(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetRefElectVoltRange(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus onCommandGetHardwareVersion(JsonObject &jsonMsg, JsonObject &jsonDat);

#endif

            void startTest();

            void stopTest();

            void setSamplePeriod(uint32\_t samplePeriod);

            uint32\_t getSamplePeriod();

            void setTestTimerCallback(void(\*func)());

            void updateTestOnTimer();

        protected:

            volatile bool testInProgress\_;

            volatile bool lastSampleFlag\_;

            AnalogSubsystem analogSubsystem\_;

            Multiplexer multiplexer\_;

#if defined HARDWARE\_VERSION\_0P2

            ElectrodeSwitch electrodeSwitch\_;

            bool electrodeAutoConnect\_;

#endif

            MessageReceiver messageReceiver\_;

            MessageParser messageParser\_;

            MessageSender messageSender\_;

            CommandTable<SystemState,CommandTableMaxSize> commandTable\_;

            CircularBuffer<Sample,DataBufferSize> dataBuffer\_;

            Voltammetry voltammetry\_;

            IntervalTimer testTimer\_;

            void (\*testTimerCallback\_)() = dummyTimerCallback;

            volatile uint64\_t timerCnt\_;

            uint32\_t samplePeriod\_;

            uint32\_t sampleModulus\_;

            Array<LowPass,NumMuxChan> currLowPass\_;

            float lowPassDtSec\_;

            BaseTest \*test\_;

            static void dummyTimerCallback() {};

            void updateSampleModulus();

    };

} // namespace ps

#endif

## Первым делом настраивает состояние системы по умолчанию – тестов нет, таймер не запущен, значения фильтра нижних частот тоже по умолчанию(частота среза – 200), период на образец – 0.1с.

## Далее в командное табло регистрируются методы по формуле обращения «command <команда>» и устанавливается текущий клиент. Их много, позволяют настраивать все элементы описанные выше.

## После включается аналоговая система с выходящим значением напряжения 0 и обнуляется циркулярный буфер приемщика сообщений.

## Если версия устройства вторая, то производится автоматическая настройка электродов на вывод. Устанавливать состояние «присоединен» надо вручную в любом случае.

## Методы system\_state. Начнём с базовых, они доступны для любой версии железа.

## Отдельно остановимся на тестах:

## Запустить или остановить тест в безопасном режиме, т.к. входящие данные теста и если они неправильного синтаксиса, то тест не запустится. (это относится ко всем методам с set). Если мультиплексор включён, то тест должен его поддерживать, иначе теста не будет. Если на мультиплексоре заняты рабочие электроды, то тест также не запустится. При успешном прохождении проверок происходит подключение электродов к мультиплексору.

## · ReturnStatus onCommandRunTest(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandStopTest(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Установить или получить значение напряжения

## · ReturnStatus onCommandGetVolt(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandSetVolt(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Получить ток или получить напряжение на электроде сравнения

## · ReturnStatus onCommandGetCurr(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetRefVolt(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## ·

## · ReturnStatus onCommandSetTestParam(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetTestParam(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Установить или получить промежуток изменения напряжения

## · ReturnStatus onCommandSetVoltRange(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetVoltRange(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Установить или получить промежуток изменения тока

## · ReturnStatus onCommandSetCurrRange(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetCurrRange(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Установить или получить код устройства

## · ReturnStatus onCommandSetDeviceId(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetDeviceId(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Установить или получить время между выборками с командной строки

## · ReturnStatus onCommandSetSamplePeriod(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetSamplePeriod(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Получить время выполнения теста, имена всех тестов, версию прошивки и вариант железа

## · ReturnStatus onCommandGetTestDoneTime(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetTestNames(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetVersion(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetVariant(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Вкл/выкл мультиплексор или проверить его состояние

## · ReturnStatus onCommandSetMuxEnabled(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetMuxEnabled(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Сделать доступным канал мультиплексора (тем самым в вкл. некоторое кол-во рабочих электродов. Кол-во зависит от канала) или проверить его состояние по индексу

## · ReturnStatus onCommandSetEnabledMuxChan(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetEnabledMuxChan(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Получить имена тестов на мультиплексоре

## · ReturnStatus onCommandGetMuxTestNames(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Изменить состояние «присоединён» или получить текущее состояние электрода сравнения, вспомогательного электрода или рабочего электрода на мультеплексоре.

## · ReturnStatus onCommandSetMuxRefElectConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetMuxRefElectConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandSetMuxCtrElectConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetMuxCtrElectConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandSetMuxWrkElectConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetMuxWrkElectConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Разъединить все электроды мультиплексора.

## · ReturnStatus onCommandDisconnAllMuxElect(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## Установить или получить время между выборками

## void setSamplePeriod(uint32\_t samplePeriod);

## uint32\_t getSamplePeriod();

## Буфер для хранения образцов

## void SystemState::serviceDataBuffer()

## Чтение информации с буфера

## void SystemState::updateMessageData()

## Вывод сообщений во время обработки JSON

## void SystemState::processMessages()

## установить колбэк таймера

## void setTestTimerCallback(void(\*func)());

## 

## void updateTestOnTimer() - изменение значений теста во время его исполнения

## void startTest() – начинает тест

## void stopTest() – останавливает тест.

## Версия железа 2

## · Изменить состояние «присоединён» или получить текущее состояние электрода сравнения, вспомогательного электрода или рабочего электрода.

## · ReturnStatus onCommandSetRefElectConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetRefElectConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandSetCtrElectConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetCtrElectConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandSetWrkElectConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetWrkElectConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Все электроды в состояние «присоединён» или получить состояние всех

## · ReturnStatus onCommandSetAllElectConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetAllElectConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Автоматическое подключение всех электродов

## · ReturnStatus onCommandSetElectAutoConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetElectAutoConn(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Установить или получить промежуток изменения напряжения для электрода сравнения

## · ReturnStatus onCommandSetRefElectVoltRange(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · ReturnStatus onCommandGetRefElectVoltRange(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## · Получить версию железа

## · ReturnStatus onCommandGetHardwareVersion(JsonObject &*jsonMsg*, JsonObject &*jsonDat*);

## ps\_time\_utils.h

#ifndef PS\_TIME\_UTILS\_H

#define PS\_TIME\_UTILS\_H

namespace ps

{

    inline uint32\_t convertUsToMs(uint64\_t t)

    {

        return uint32\_t(t/UINT64\_C(1000));

    }

    inline uint64\_t convertMsToUs(uint32\_t t)

    {

        return uint64\_t(t)\*UINT64\_C(1000);

    }

} // namespace ps

#endif

Вспомогательные методы для преобразования секунд в миллисекунды и обратно

## ps\_volt\_range

#ifndef PS\_VOLT\_RANGE\_H

#define PS\_VOLT\_RANGE\_H

#include <Arduino.h>

#include "ps\_range.h"

#include "ps\_gains.h"

namespace ps

{

    class VoltRange : public Range<VoltGain,uint16\_t>

    {

        public:

            VoltRange() : Range<VoltGain,uint16\_t>()  {};

            VoltRange(String name, float minValue, float maxValue, VoltGain voltGain, uint16\_t maxInt)

                : Range<VoltGain,uint16\_t>(name, minValue, maxValue, voltGain, maxInt) {};

    };

    class VoltRangeDac : public VoltRange

    {

        //using VoltRange::VoltRange;

        public:

            VoltRangeDac() : VoltRange()  {};

            VoltRangeDac(String name, float minValue, float maxValue, VoltGain voltGain, uint16\_t maxInt)

                : VoltRange(name, minValue, maxValue, voltGain, maxInt) {};

    };

    class VoltRangeAdc : public VoltRange

    {

        //using VoltRange::VoltRange;

        public:

            VoltRangeAdc() : VoltRange()  {};

            VoltRangeAdc(String name, float minValue, float maxValue, VoltGain voltGain, uint16\_t maxInt)

                : VoltRange(name, minValue, maxValue, voltGain, maxInt) {};

    };

}

#endif

Конструкторы VoltRange, VoltRangeDac, VoltRangeAdc на основе шаблона класса, описанного в классе [ps\_range.h](#_heading=h.jivksuj5we02)

## ps\_voltammetry

#ifndef PS\_VOLTAMMETRY\_H

#define PS\_VOLTAMMETRY\_H

#include "ps\_constants.h"

#include "ps\_return\_status.h"

#include "ps\_base\_test.h"

#include "ps\_cyclic\_test.h"

#include "ps\_sinusoid\_test.h"

#include "ps\_constant\_test.h"

#include "ps\_linearsweep\_test.h"

#include "ps\_multistep\_test.h"

#include "ps\_squarewave\_test.h"

#include "third-party/ArduinoJson/ArduinoJson.h"

#include "third-party/Array/Array.h"

namespace ps

{

    class Voltammetry

    {

        public:

            Voltammetry();

            BaseTest \*getTest(String name);

            ReturnStatus getTest(JsonObject &jsonMsg, JsonObject &jsonDat, BaseTest\* &testPtr);

            ReturnStatus getParam(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus setParam(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus getTestDoneTime(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus getTestNames(JsonObject &jsonMsg, JsonObject &jsonDat);

            ReturnStatus getMuxTestNames(JsonObject &jsonMsg, JsonObject &jsonDat);

            void setSamplePeriod(uint64\_t samplePeriod);

            BaseTest baseTest;

            CyclicTest cyclicTest;

            SinusoidTest sinusoidTest;

            ConstantTest constantTest;

            SquareWaveTest squareWaveTest;

            LinearSweepTest linearSweepTest;

            MultiStepTest<2> chronoampTest;

            MultiStepTest<MultiStepMaxSize> multiStepTest;

            static const String TestKey;

        protected:

            Array<BaseTest\*,AvailableTestsMaxSize> availableTests\_;

    };

}

#endif

В .h файле описывается класс Voltametry, с public методами:

\*getTest(String name) - указатель на имя теста;

ReturnStatus getTest(JsonObject &jsonMsg, JsonObject &jsonDat, BaseTest\* &testPtr) - сохраняет в json тест;

ReturnStatus getParam(JsonObject &jsonMsg, JsonObject &jsonDat) - сохраняет в json параметры теста;

ReturnStatus setParam(JsonObject &jsonMsg, JsonObject &jsonDat) - устанавливает параметры из json файла;

ReturnStatus getTestDoneTime(JsonObject &jsonMsg, JsonObject &jsonDat) - сохраняет в json файл время выполнения теста;

ReturnStatus getTestNames(JsonObject &jsonMsg, JsonObject &jsonDat) - сохраняет в json файл имена тестов;

ReturnStatus getMuxTestNames(JsonObject &jsonMsg, JsonObject &jsonDat) - сохраняет в json файл мультиплексорные имена тестов;

void setSamplePeriod(uint64\_t samplePeriod) - устанавливает период времени между выборками;

Затем создаются все тесты описанные в файлах ранее:

BaseTest baseTest;

CyclicTest cyclicTest;

SinusoidTest sinusoidTest;

ConstantTest constantTest;

SquareWaveTest squareWaveTest;

LinearSweepTest linearSweepTest;

MultiStepTest<2> chronoampTest;

MultiStepTest<MultiStepMaxSize> multiStepTest;

Создается переменная: static const String TestKey;

protected методы:

Array<BaseTest\*,AvailableTestsMaxSize> availableTests\_;

В .cpp файле описывается как работают перечисленные выше методы.

ВЫВОД

Были изучены и проанализированы модули прошивки потенциостата.