Arduino bioreaktor Herbion

v0.9.0

Generated by Doxygen 1.12.0

1 main issue 1

1.0.1 this project working by using platformio extension for visual studio code . . . 1

1.0.2 Code is not fully tested . . . 1

1.0.3 all code on develop branch is still in progress it is not recomend to copy/clone any part of it . . . 1

1.1 dokumentacja . . . 1

1.2 Dokumentacja 2 . . . 1

2 Hierarchical Index 3

2.1 Class Hierarchy . . . 3

3 Class Index 5

3.1 Class List . . . 5

4 File Index 7

4.1 File List . . . 7

5 Class Documentation 9

5.1 Config_var Class Reference . . . 9

5.2 ConfigurationVariable Class Reference . . . 9

5.3 DataHMS Class Reference . . . 9

5.4 MeasureArray Class Reference . . . 10

5.5 MemoryManager Class Reference . . . 10

5.6 my_Rotary_encoder Class Reference . . . 10

5.7 MyLCD Class Reference . . . 11

5.8 SdMemoryManager Class Reference . . . 11

5.9 Sensor Class Reference . . . 11

5.10 TimerLowPriority Class Reference . . . 12

6 File Documentation 13

6.1 bioreactor_defined_const.hpp . . . 13

6.2 components.hpp . . . 14

6.3 eeprom_menager.hpp . . . 15

6.4 lcd_display.hpp . . . 15

6.5 my_encoder.hpp . . . 16

6.6 sd_memory.hpp . . . 16

6.7 sensor_config.hpp . . . 18

6.8 Unit_tests.hpp . . . 18

6.9 utility.hpp . . . 18

Index 21

Chapter 1

main issue

1.0.1 this project working by using platformio extension for visual studio code

1.0.2 Code is not fully tested

1.0.3 all code on develop branch is still in progress it is not recomend to copy/clone any part of it

1.1 dokumentacja

```
# Startował
# w wyborach

# prezydenckich
```

1.2 Dokumentacja 2

```
# TODO

# TOMASZ

# HAJTO
```

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Config_var 9
ConfigurationVariable 9
DataHMS 9
LiquidCrystal_I2C
    MyLCD 11
MeasureArray 10
MemoryManager 10
my_Rotary_encoder 10
SdMemoryManager 11
Sensor 11
TimerLowPriority 12

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Config_var 9
ConfigurationVariable 9
DataHMS 9
MeasureArray 10
MemoryManager 10
my_Rotary_encoder 10
MyLCD 11
SdMemoryManager 11
Sensor 11
TimerLowPriority 12

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

bioreactor_defined_const.hpp 13
components.hpp 14
eeprom_menager.hpp 15
lcd_display.hpp 15
my_encoder.hpp 16
sd_memory.hpp 16
sensor_config.hpp 18
Unit_tests.hpp 18
utility.hpp 18

Chapter 5

Class Documentation

5.1 Config_var Class Reference

Public Member Functions

- **Config_var** (float desire_ph, float max_ph_acceptable_deviation, float desire_temp, float max_temp_↩ acceptable_deviation)

The documentation for this class was generated from the following files:

- utility.hpp
- utility.cpp

5.2 ConfigurationVariable Class Reference

Public Member Functions

- **ConfigurationVariable** (MemoryManager &mem_manager)
- int **get_addr** () const
- float **return_config_value** () const
- int **retrieve_config_values_from_eeprom** ()
- void **change_config_value** (float value)

The documentation for this class was generated from the following files:

- eeprom_menager.hpp
- eeprom_menager.cpp

5.3 DataHMS Class Reference

Public Member Functions

- **DataHMS** (long hour, long minute, long second)
- String **return_data** ()

Public Attributes

- long **m_offset**

The documentation for this class was generated from the following files:

- utility.hpp
- utility.cpp

5.4 MeasureArray Class Reference

Public Member Functions

- **MeasureArray** (int size)
- void **init** (float initial_value)
- void **add_measure** (float value)
- float **read_measure** (char index)
- float **get_average** ()

The documentation for this class was generated from the following files:

- utility.hpp
- utility.cpp

5.5 MemoryManager Class Reference

Public Member Functions

- **MemoryManager** (int memory_start, int memory_length)
- int **give_memory** (int require_memory)

The documentation for this class was generated from the following files:

- eeprom_menager.hpp
- eeprom_menager.cpp

5.6 my_Rotary_encoder Class Reference

Public Member Functions

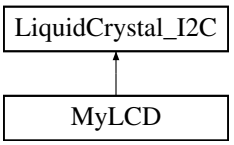
- **my_Rotary_encoder** (uint8_t pinA, uint8_t pinB, uint8_t button, long sensitivity)
- void **init** ()
- int **get_encoder_pos** ()
- int **get_button_state** ()
- long **return_button_inactivate_state_time** ()
- void **check_encoder_pos** ()
- void **reset_encoder_pos** ()
- int **get_encoder_move** ()
- float **set_value** (float initial_value, float step, MyLCD lcd)
- int **get_button_depth** ()
- void **reset__button_depth** ()

The documentation for this class was generated from the following files:

- my_encoder.hpp
- my_encoder.cpp

5.7 MyLCD Class Reference

Inheritance diagram for MyLCD:



Public Member Functions

- **MyLCD** (uint8_t addr, uint8_t column_num, uint8_t row_num)
- void **initialize** ()
- void **send_float_value** (String text, float value, int lcd_row)
- void **send_string** (String text, String value, int lcd_row)

The documentation for this class was generated from the following files:

- lcd_display.hpp
- lcd_display.cpp

5.8 SdMemoryManager Class Reference

Public Member Functions

- **SdMemoryManager** (uint8_t MOSI_pin, uint8_t MISO_pin, uint8_t SCK_pin, uint8_t CS)
- void **init** ()
- void **write_to_st** ()
- void **write_data_frame_to_st** (Sensor &thermometer, Sensor &ph_meter, Sensor &oxygen_meter, DataHMS &data)
- String **DEBUG_write_data_frame** (Sensor &thermometer, Sensor &ph_meter, Sensor &oxygen_meter, DataHMS &data)
- void **save** ()
- void **close_file** ()

The documentation for this class was generated from the following files:

- sd_memory.hpp
- sd_memory.cpp

5.9 Sensor Class Reference

Public Member Functions

- **Sensor** (MeasuringDevice *measuring_dev, ConfigurationVariable &zero_shift, ConfigurationVariable &linear_factor)
- float **get_value_from_measurement** ()
- float **get_value** ()
- void **init** ()

Public Attributes

- ConfigurationVariable **m_zero_shift**
- ConfigurationVariable **m_linear_factor**
- MeasuringDevice * **m_measuring_device**

The documentation for this class was generated from the following files:

- eeprom_menager.hpp
- eeprom_menager.cpp

5.10 TimerLowPriority Class Reference

Public Member Functions

- bool **activate** (int time_to_activate)
- void **reset** ()

The documentation for this class was generated from the following files:

- utility.hpp
- utility.cpp

Chapter 6

File Documentation

6.1 bioreactor_defined_const.hpp

```
00001 #ifndef BIOREACTOR_CONST
00002 #define BIOREACTOR_CONST
00003
00004
00005 //BUTTON SETTING
00006 #define BUTTONDEFAULTSTATE 1 //mean pin state is HIGH
00007 #define BUTTONSTAYONSTATE 2000 //[ms] time button dont change its state
00008 #define STAYINMENUTIME 3000 //[ms] time after you can exit menu
00009
00010
00011 //ROTARY ENCODER SETTINGS
00012
00013 #define SENSITIVITY 150 // [ms] time after re will not change it state after did it before //TODO use
      better world :)
00014
00015
00016 //LCD SETTINGS
00017 // for now not available
00018
00019 //SENSOR DESIRE PARAMETER
00020 #define DESIRE_PH 7
00021 #define MAX_PH_ACCEPTABLE_DEVIATION 0.5
00022 #define DESIRE_TEMP 21.37
00023 #define MAX_TEMP_ACCEPTABLE_DEVIATION 4.2069
00024
00025
00026 //LCD PHYSICAL CONNECTION
00027 // TODO
00028
00029 //SD CARD READER PHYSICAL CONNECTION
00030 #define CSPIN 10
00031 #define MOSIPIN 11
00032 #define MISOPIN 12
00033 #define SCKPIN 13
00034 #define FILENAME "m.txt"
00035
00036 //ROTARY ENCODER PHYSICAL CONNECTION
00037 #define REPINA 3
00038 #define REPINB 4
00039 #define REBUTTONPIN 5
00040
00041 //SENSORS PHYSICAL CONNECTION
00042 #define PH_METER_PIN A1
00043 #define THERMOMETHERPIN A2
00044 #define OXYGEN_METER_PIN A3
00045
00046 //SAMPLE PUMP PARAMETER
00047 #define PUMPFLOWSPEED 60 //[ml/min]
00048 #define SAMPLESIZE 2 //[ml]
00049 #define PUMPSAMPLETAKINGTIME     int(float(SAMPLESIZE)/float(PUMPFLOWSPEED)*60*1000) //[ms] //powinno
      wyjsc 2 sekundy
00050
00051 //PH KEEPERS PUMP
00052 #define TIMEBETWENWORK 10000 //[ms] time between ph correction
00053 #define CORECTIONTIME 100 // [ms] initial time of corection
00054 #define MULTPERDEGRE 3 // it mean if  desire ph is 6 and we have 8 time of pump activate will be
      100*3*3 = 900 ms
00055 #define MAXREACTIONTIME 3000
```

```
00056
00057
00058
00059
00060 #define THERMISTOR_NOMINAL 10000        // [ohm] resistance at 25 degrees C
00061 #define TEMPERATURE_NOMINAL 25          // temp. for nominal resistance (almost always 25 C)
00062 #define B_COEFFICIENT 3950              // [UNIT!] The beta coefficient of the thermistor (usually
      3000-4000)
00063 #define SERIES_RESISTOR 10000           // [ohm] the value of the 'other' resistor
00064
00065
00066
00067 #endif // BIOREACTOR_CONST
```

6.2 components.hpp

```
00001 #include <Arduino.h>
00002 #include <math.h>
00003 #include "bioreactor_defined_const.hpp"
00004 #include "utility.hpp"
00005
00006 #ifndef BIOREACTOR_COMPONENTS
00007 #define BIOREACTOR_COMPONENTS
00008
00009 class SimplePeristalticPump
00010 {
00011 private:
00012 uint8_t m_pin_forward;
00013
00014
00015 public:
00016 SimplePeristalticPump(uint8_t pin_forward);
00017 void init();
00018 void stabilize_ph(float current_ph,float desire_ph);
00019 };
00020
00021 class PeristalticPump
00022 {
00023   private:
00024     const uint8_t m_pwmPin;  //TODO snakekase
00025     const uint8_t m_dir1Pin;
00026     const uint8_t m_dir2Pin;
00027     int m_currentPwmValue;
00028
00029   public:
00030     PeristalticPump(uint8_t PwmPin, uint8_t Dir1Pin, uint8_t Dir2Pin);  //TODO rename variable
00031     void init();
00032     void set_pump_speed(int value); // from -100 (max reverse speed) to 100 (max forward speed)
00033     long get_current_speed() const;
00034     void take_sample();
00035 };
00036
00037 class MeasuringDevice
00038 {
00039   //TODO: add virtual destructor even empty
00040   protected:
00041     const int m_read_pin;
00042     float m_value; // check what sensors return  TODO for now int ; but probably will be change in
      inheritance
00043
00044   public:
00045     MeasuringDevice(int read_pin);
00046     void init();
00047     virtual float get_value();
00048 };
00049
00050 class Thermometer : public MeasuringDevice
00051 {
00052   public:
00053     Thermometer(int read_pin);
00054     float get_value() override;
00055 };
00056
00057 class PhMeter : public MeasuringDevice
00058 {
00059   public:
00060     PhMeter(int read_pin);
00061     float get_value() override;
00062 };
00063
00064 class OxygenMeter : public MeasuringDevice
00065 {
00066   public:
```

```
00067     OxygenMeter(int read_pin);
00068     float get_value() override;
00069 };
00070
00071 #endif // BIOREACTOR_COMPONENTS
```

6.3 eeprom_menager.hpp

```
00001 #include "components.hpp"
00002 #include <EEPROM.h>
00003
00004 #ifndef TEST
00005 #define TEST
00006
00007 class MemoryManager
00008 {
00009   private:
00010     const int m_memory_start;
00011     const int m_memory_size;
00012     int m_memory_pointer;
00013
00014   public:
00015     MemoryManager(int memory_start, int memory_length);
00016     int give_memory(int require_memory);
00017 };
00018
00019 class ConfigurationVariable
00020 {
00021   private:
00022     int m_memory_addr;
00023     float m_value = -21.37;
00024
00025   public:
00026     ConfigurationVariable(MemoryManager &mem_manager);
00027     int get_addr() const;
00028     float return_config_value() const;
00029     int retrieve_config_values_from_eeprom();
00030     void change_config_value(float value);
00031 };
00032
00033 class Sensor
00034 {
00035   private:
00036     float m_value = 10;
00037
00038   public: // bad practice TODO in free time try to change to private
00039     ConfigurationVariable m_zero_shift, m_linear_factor; //TODO - make reference to this object not
      copy
00040     MeasuringDevice *m_measuring_device;
00041
00042   public:
00043     Sensor(MeasuringDevice *measuring_dev, ConfigurationVariable &zero_shift, ConfigurationVariable
      &linear_factor);
00044     float get_value_from_measurement();
00045     float get_value();
00046     void init();
00047 };
00048
00049 #endif // TEST
```

6.4 lcd_display.hpp

```
00001 #ifndef LCD_DISPLAY_BR
00002 #define LCD_DISPLAY_BR
00003
00004 #include<LiquidCrystal_I2C.h>
00005 // not refactored
00006
00007 class MyLCD : public LiquidCrystal_I2C
00008 {
00009
00010   public:
00011   MyLCD(uint8_t addr, uint8_t column_num, uint8_t row_num);
00012   void initialize(); //TODO: better name
00013   void send_float_value(String text, float value, int lcd_row); //TODO: change to template
00014   void send_string(String text, String value, int lcd_row);
00015 };
00016
00017 #endif // LCD_DISPLAY_BR
```

6.5 my_encoder.hpp

```
00001 #ifndef MY_ENCODER
00002 #define MY_ENCODER
00003
00004 #include <Arduino.h>
00005 #include "lcd_display.hpp"
00006
00007
00008 class my_Rotary_encoder
00009 {
00010   private:
00011     const uint8_t m_pinA;
00012     const uint8_t m_pinB;
00013     const uint8_t m_pin_button;
00014
00015     uint8_t m_button_depth;
00016     long m_sensitivity;
00017     long m_button_inactivate_state_time; //TODO rename
00018     long m_last_change;
00019
00020     uint8_t m_aVal = 0;
00021     uint8_t m_pinALast = 0;
00022     uint8_t m_button_state = 1;
00023     int m_encoderPosCount;
00024
00025   public:
00026     my_Rotary_encoder(uint8_t pinA, uint8_t pinB, uint8_t button, long sensitivity);
00027     void init();
00028     int get_encoder_pos();
00029     int get_button_state();
00030     long return_button_inactivate_state_time();
00031     void check_encoder_pos();
00032     void reset_encoder_pos();
00033     int get_encoder_move();
00034     float set_value(float initial_value, float step, MyLCD lcd);
00035     int get_button_depth();
00036     void reset__button_depth();
00037 };
00038
00039 #endif
```

6.6 sd_memory.hpp

```
00001 #ifndef SM_MEMORY
00002 #define SM_MEMORY
00003
00004 #include <SD.h>
00005 #include "eeprom_menager.hpp"
00006 #include "utility.hpp"
00007
00008 class SdMemoryManager
00009 {
00010   private:
00011     // pins
00012     const uint8_t m_MOSI_pin;
00013     const uint8_t m_MISO_pin;
00014     const uint8_t m_SCK_pin;
00015     const uint8_t m_CS;
00016
00017     uint8_t m_write_number=0;
00018
00019     File m_file;
00020     String m_filename;
00021     String m_dataBuffer;
00022     unsigned long m_last_write = 0;
00023
00024   public:
00025     SdMemoryManager(uint8_t MOSI_pin, uint8_t MISO_pin, uint8_t SCK_pin, uint8_t CS);
00026     void init();
00027     void write_to_st();
00028     void write_data_frame_to_st(Sensor& thermometer, Sensor& ph_meter, Sensor& oxygen_meter, DataHMS&
      data);
00029     String DEBUG_write_data_frame(Sensor& thermometer, Sensor& ph_meter, Sensor& oxygen_meter,
      DataHMS& data);
00030     void save();
00031     void close_file();
00032 };
00033
00034 #endif
00035
00036 /*
00037
```

```
00038 #include <SD.h>
00039
00040 const int chipSelect = 10;
00041
00042 // file name to use for writing
00043 const char filename[] = "datalog.txt";
00044
00045 // File object to represent file
00046 File myFile;
00047 // string to buffer output
00048 String dataBuffer;
00049 // last time data was written to card:
00050 unsigned long lastMillis = 0;
00051
00052 void setup() {
00053   // Open serial communications and wait for port to open:
00054   Serial.begin(9600);
00055   // reserve 1 kB for String used as a dataBuffer
00056   dataBuffer.reserve(1024);
00057
00058   // set LED pin to output, used to blink when writing
00059   pinMode(LED_BUILTIN, OUTPUT);
00060
00061   // wait for Serial Monitor to connect. Needed for native USB port boards only:
00062   while (!Serial);
00063
00064   Serial.print("Initializing SD card...");
00065
00066   if (!SD.begin(chipSelect)) {
00067     Serial.println("initialization failed. Things to check:");
00068     Serial.println("1. is a card inserted?");
00069     Serial.println("2. is your wiring correct?");
00070     Serial.println("3. did you change the chipSelect pin to match your shield or module?");
00071     Serial.println("Note: press reset button on the board and reopen this Serial Monitor after fixing
      your issue!");
00072     while (true);
00073   }
00074
00075   Serial.println("initialization done.");
00076
00077   // If you want to start from an empty file,
00078   // uncomment the next line:
00079   //  SD.remove(filename);
00080   // try to open the file for writing
00081
00082   myFile = SD.open(filename, FILE_WRITE);
00083   if (!myFile) {
00084     Serial.print("error opening ");
00085     Serial.println(filename);
00086     while (true);
00087   }
00088
00089   // add some new lines to start
00090   myFile.println();
00091   myFile.println("Hello World!");
00092   Serial.println("Starting to write to file...");
00093 }
00094
00095 void loop() {
00096   // check if it's been over 10 ms since the last line added
00097   unsigned long now = millis();
00098   if ((now - lastMillis) >= 10) {
00099     // add a new line to the dataBuffer
00100     dataBuffer += "Hello ";
00101     dataBuffer += now;
00102     dataBuffer += "\r\n";
00103     // print the buffer length. This will change depending on when
00104     // data is actually written to the SD card file:
00105     Serial.print("Unsaved data buffer length (in bytes): ");
00106     Serial.println(dataBuffer.length());
00107     // note the time that the last line was added to the string
00108     lastMillis = now;
00109   }
00110
00111   // check if the SD card is available to write data without blocking
00112   // and if the dataBuffered data is enough for the full chunk size
00113   unsigned int chunkSize = myFile.availableForWrite();
00114   if (chunkSize && dataBuffer.length() >= chunkSize) {
00115     // write to file and blink LED
00116     digitalWrite(LED_BUILTIN, HIGH);
00117     myFile.write(dataBuffer.c_str(), chunkSize);
00118     digitalWrite(LED_BUILTIN, LOW);
00119     // remove written data from dataBuffer
00120     dataBuffer.remove(0, chunkSize);
00121   }
00122 }
00123
```

```
00124 */
```

6.7 sensor_config.hpp

```
00001 #include "bioreactor_defined_const.hpp"
00002 #include "components.hpp"
00003 #include "eeprom_menager.hpp"
00004 #include <Arduino.h>
00005
00006 #ifndef MY_SC
00007 #define MY_SC
00008
00009 Sensor setup_thermometer_sensors(MemoryManager &manager);
00010 Sensor setup_ph_sensors(MemoryManager &manager);
00011 Sensor setup_oxygen_sensors(MemoryManager &manager);
00012 void test_sensor(Sensor &sensor, float value_new, String sensor_name = "Sensor");
00013
00014 #endif
```

6.8 Unit_tests.hpp

```
00001 #include "utility.hpp"
00002 #include <AUnit.h>
00003
00004 #ifndef MY_UNIT_TEST
00005 #define MY_UNIT_TEST
00006
00007 // not refactored
00008
00009 int dumb_func()
00010 {
00011     return 1;
00012 }
00013
00014 MeasureArray m1(10), m2(3); // 10 ; 3
00015
00016 float test_mes_array(MeasureArray &ma)
00017 {
00018     ma.add_measure(20);
00019     ma.add_measure(10);
00020     ma.add_measure(5);
00021     ma.add_measure(5);
00022     ma.add_measure(60);
00023     return (ma.get_average());
00024 }
00025
00026 float test_mes_array2(MeasureArray &ma)
00027 {
00028     ma.add_measure(20);
00029     ma.add_measure(10);
00030     ma.add_measure(2137);
00031
00032     return ma.read_measure(2);
00033 }
00034
00035 test(test_mes_arrayTest)
00036 {
00037     assertEqual(test_mes_array(m1), 10.0);
00038 }
00039
00040 test(test_mes_array2Test)
00041 {
00042     assertEqual(test_mes_array2(m2), 2137.0);
00043 }
00044
00045 test(dumb_funcTest)
00046 {
00047     assertEqual(dumb_func(), 1);
00048 }
00049
00050 #endif // MY_UNIT_TEST
```

6.9 utility.hpp

```
00001 #ifndef MY_UTILITY
```

```
00002 #define MY_UTILITY
00003
00004 #include<string.h>
00005 #include<Arduino.h>
00006
00007 #include "my_encoder.hpp"
00008 #include "eeprom_menager.hpp"
00009 #include "lcd_display.hpp"
00010
00011 class Config_var   //more like struct but... TODO check if we can change it to real struct
00012 {
00013   private:
00014   float m_desire_ph;
00015   float m_max_ph_acceptable_deviation;
00016
00017   float m_desire_temp;
00018   float m_max_temp_acceptable_deviation;
00019
00020   public:
00021   Config_var(float desire_ph, float max_ph_acceptable_deviation, float desire_temp, float
      max_temp_acceptable_deviation);
00022 };
00023
00024
00025
00026
00027 class MeasureArray // TODO is not good enough make it better
00028 {
00029   private: //TODO: inhertinace after arduino vector
00030   //TODO: is not good enough make it better
00031     const char m_array_size; // max 256 (but                not recommended to use more than 100)
00032     float *measurement;      // Pointer array, will be initialized              in constructor
00033     int m_memory_cursor = 0;
00034     float m_oldest_measure;
00035
00036   public:
00037     MeasureArray(int size);
00038     void init(float initial_value); //TODO make it vector
00039     void add_measure(float value);
00040     float read_measure(char index);
00041     float get_average(); //TODO: add const
00042 };
00043
00044 class TimerLowPriority
00045 {
00046   private:
00047     unsigned long m_start_time;
00048     unsigned long m_end_time;
00049
00050   public:
00051     TimerLowPriority();
00052     bool activate(int time_to_activate);
00053     void reset();
00054 };
00055
00056 // TODO high priority timer
00057
00058 class DataHMS
00059 {
00060   public:
00061     long m_offset;
00062
00063   public:
00064     DataHMS(long hour, long minute, long second);
00065     String return_data(); //TODO const
00066 };
00067
00068 void print_config_menu(my_Rotary_encoder &encoder, MyLCD &lcd, // main sensors
00069                        Sensor &term, Sensor &ph, Sensor &oxygen, PeristalticPump& pump);
00070
00071 #endif
```

Index

bioreactor_defined_const.hpp, 13

components.hpp, 14
Config_var, 9
ConfigurationVariable, 9

DataHMS, 9

eeprom_menager.hpp, 15

lcd_display.hpp, 15

main issue, 1
MeasureArray, 10
MemoryManager, 10
my_encoder.hpp, 16
my_Rotary_encoder, 10
MyLCD, 11

sd_memory.hpp, 16
SdMemoryManager, 11
Sensor, 11
sensor_config.hpp, 18

TimerLowPriority, 12

Unit_tests.hpp, 18
utility.hpp, 18