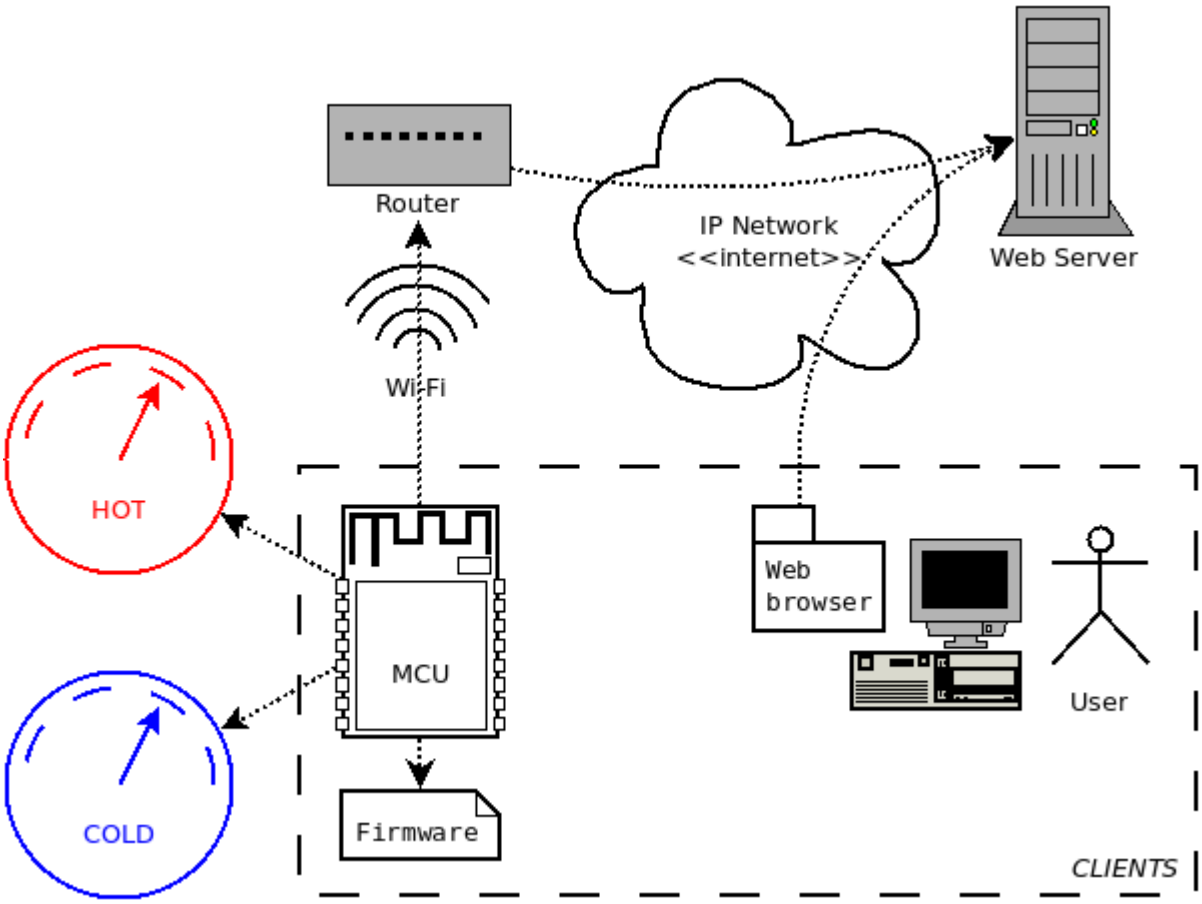
Оглавление

1. Общая схема комплекса



Комплекс включает в себя следующие элементы:

- **HOT** и **COLD** — счетчики воды с импульсными выходами
- **MCU** (Micro-Controller Unit) — микроконтроллер
- **Firmware** — программа для микроконтроллера
- Пользователь **User**, получающий доступ к интерфейсу комплекса через **Web browser**
- **Router** — коммутатор локальной сети, с которым **MCU** соединяется с помощью **Wi-Fi**
- **Web Server** — сервер, обслуживающий запросы **MCU** и предоставляющий пользовательский интерфейс по протоколу **HTTP**

Клиент **User** и сервер **Web Server** могут размещаться как в той же локальной сети, что и микроконтроллер **MCU**, так и в произвольном месте подключения к IP-сети (в частности, сети Internet). Комплекс не накладывает на это ограничений.

2. Счетчики

Используются счетчики воды с импульсными выходами. Схема работы счетчиков — **ГЕРКОН**, на каждые 10 литров расхода воды происходит замыкание импульсного выхода. Альтернативные импульсные схемы — **НАМУР** и **БЕТАР** на данный момент не поддерживаются.

3. Микроконтроллер

Используется микроконтроллер семейства **ESP-8266**, конкретно, **ESP-12S**. Выбранный микроконтроллер обладает следующими определяющими характеристиками:

- Низкая цена
- Наличие достаточного количества GPIO (General Purpose Input-Output) портов
- Наличие Wi-Fi модуля в базовой комплектации
- Поддержка программирования в среде Arduino IDE

Микроконтроллер использует порты GPIO для опроса счетчиков воды и индикации посредством светодиодов:

- GPIO 4 в режиме INPUT — для опроса счетчика холодной воды
- GPIO 5 в режиме INPUT — для опроса счетчика горячей воды
- GPIO 12 в режиме OUTPUT — для управления зеленым светодиодом
- GPIO 13 в режиме OUTPUT — для управления красным светодиодом

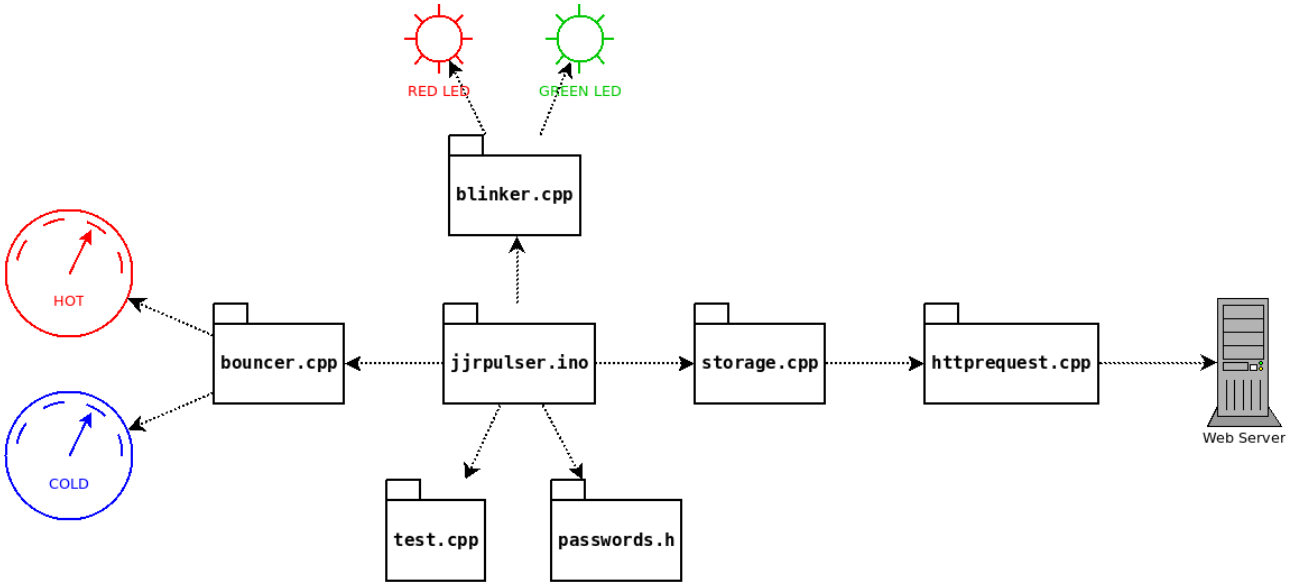
Микроконтроллер соединяется с локальной сетью посредством Wi-Fi.

Электрическая схема подключения микроконтроллера прилагается в виде отдельного документа.

Программное обеспечение микроконтроллера описано в отдельной главе.

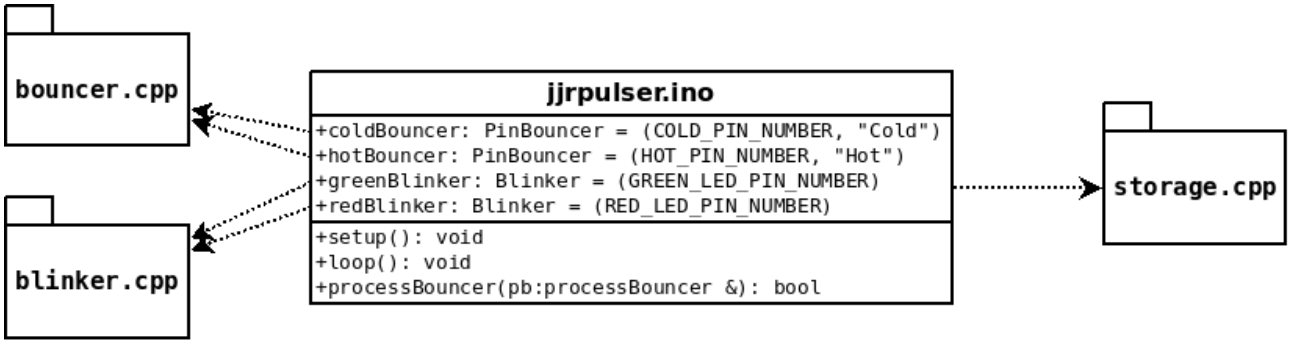
4. Программа для микроконтроллера

Программа микроконтроллера написана на языке C++, компилируется и загружается в микроконтроллер с помощью Arduino IDE. Программа разбита на модули, содержащие логически выделенные алгоритмы:



- **jjrpulser.ino** — содержит основные функции работы микроконтроллера (согласно архитектуре Arduino IDE)
- **blinker.cpp** — предоставляет удобное управление индикацией светодиодов
- **bouncer.cpp** — реализует алгоритмы компенсации дребезга контактов счетчиков
- **storage.cpp** — содержит логику подсчета импульсов, формирования статистики, и отправки их на сервер
- **httprequest.cpp** — предоставляет удобную отправку асинхронных HTTP GET-запросов на сервер
- **passwords.h** — заголовочный файл, в который вынесена индивидуальная для инсталляции информация
- **test.cpp** — логика нагрузочного тестирования, не используется в штатной работе

4.1. jjrpulser.ino — главный модуль



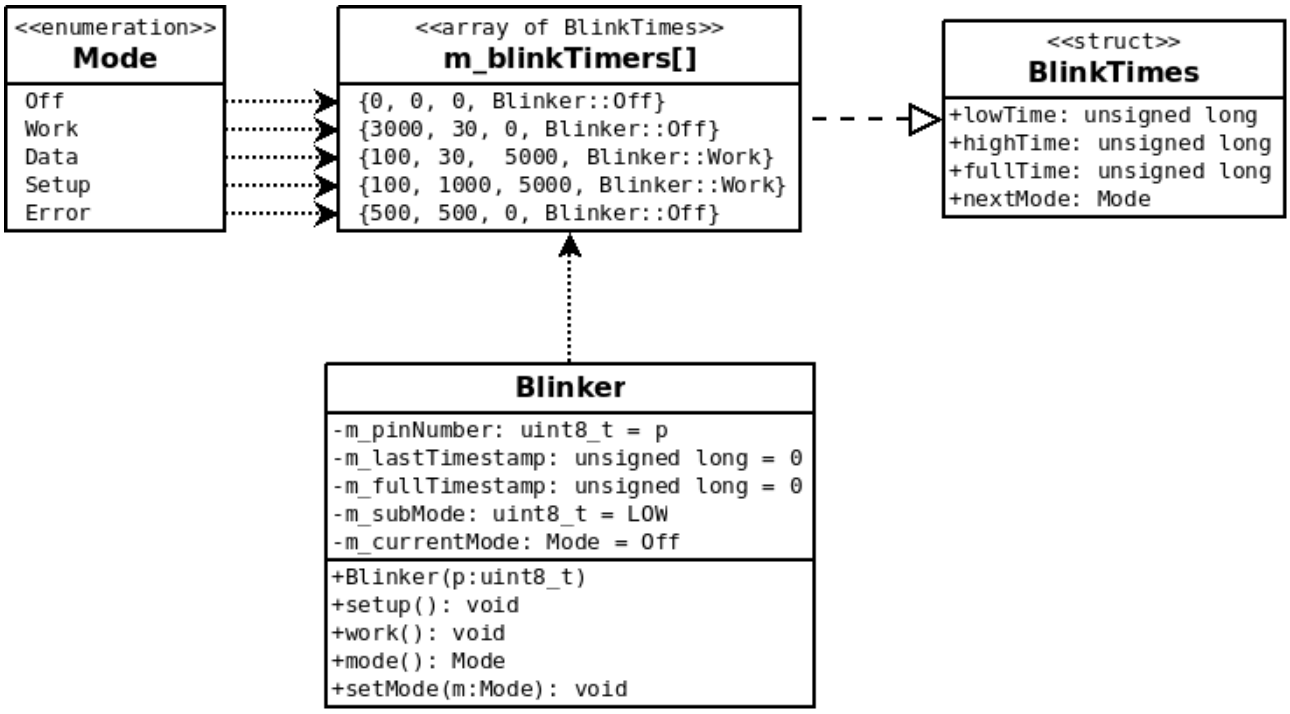
В модуле объявлены глобальные переменные:

- **coldBouncer** — объект типа **PinBouncer**, регистрирующий показания счетчика холодной воды. Работает с GPIO **COLD_PIN_NUMBER** = 4, имеет символьное имя **"Cold"**, эти данные передаются, как параметры при конструировании объекта
- **hotBouncer** — объект типа **PinBouncer**, регистрирующий показания счетчика горячей воды. Работает с GPIO **HOT_PIN_NUMBER** = 5, имеет символьное имя **"Hot"**
- **greenBlinker** — объект типа **Blinker**, управляющий работой зеленого светодиода через GPIO **GREEN_LED_PIN_NUMBER** = 12
- **redBlinker** — объект типа **Blinker**, управляющий работой красного светодиода через GPIO **RED_LED_PIN_NUMBER** = 13

В модуле объявлены функции:

- **setup()** — среда Arduino выполняет эту функцию один раз, при запуске программы. В функции настраиваются параметры последовательного порта, глобальных объектов и модуля **DataStorage** (реализовано, как вызовы методов **setup**)
- **loop()** — функция главного цикла среды Arduino. Среда бесконечно вызывает эту функцию на протяжении всего времени работы программы. В функции опрашиваются счетчики через объекты **coldBouncer**, **hotBouncer** и функцию **processBouncer**, отсылается статистика через модуль **DataStorage**, и все объекты и модули получают свой квант времени для работы (реализовано, как вызовы методов **work**)
- **processBouncer()** — в эту функцию вынесена логика работы с объектами типа **PinBouncer**. В нее (из функции **loop**) передаются на обработку объекты **coldBouncer** и **hotBouncer**, над которыми, таким образом, производится одинаковая последовательность действий без дублирования кода

4.2. blinker.cpp — модуль управления светодиодами



Модуль содержит следующие вспомогательные элементы:

- Перечисление режимов работы **Mode**, включает в себя режимы:
  - **Off** — светодиод выключен
  - **Work** — устройство находится в обычном рабочем режиме
  - **Data** — устройство передает данные на сервер
  - **Setup** — устройство получило скорректированные показания с сервера
  - **Error** — последняя операция завершилась ошибкой
- Структуру **BlinkTimes**, описывающую тайминги повторяющегося периода мигания:
  - **lowTime** — сколько миллисекунд светодиод выключен
  - **highTime** — сколько миллисекунд светодиод включен
  - **fullTime** — через сколько миллисекунд (от переключения в текущий режим) произойдет переключение в следующий режим **nextMode**. Если 0, то переключение в следующий режим не осуществляется
  - **nextMode** — следующий режим, в который происходит автоматическое переключение по истечению **fullTime** миллисекунд. Как пример — через вызов метода **setMode()** осуществляется явное переключение в режим **Setup**, он мигает 5 секунд, потом автоматически переключается в режим **Work**
- Массив **m_blinkTimers**, заполненный структурами **BlinkTimes** для каждого режима **Mode**:
  - **Первый элемент** — для режима **Off**. Светодиод всегда выключен
  - **Второй элемент** — для режима **Work**. Выключен 3000мс, потом вспыхивает на 30мс, режим не переключается
  - **Третий элемент** — для режима **Data**. Выключен 100мс, потом вспыхивает на 30мс, через 5000мс переключается в режим **Work**
  - **Четвертый элемент** — для режима **Setup**. Гаснет на 100мс, потом горит 1000мс, через 5000мс переключается в режим **Work**
  - **Пятый элемент** — для режима **Error**. Выключен 500мс, потом горит 500мс, режим не переключается

Используемым номерам GPIO через макросы даны следующие имена:

- **GREEN_LED_PIN_NUMBER** — GPIO 12, через который осуществляется управление зеленым светодиодом
- **RED_LED_PIN_NUMBER** — GPIO 13, через который осуществляется управление красным светодиодом

Основной интерфейс управления светодиодами реализован через объекты класса **Blinker**, предоставляющего следующие методы:

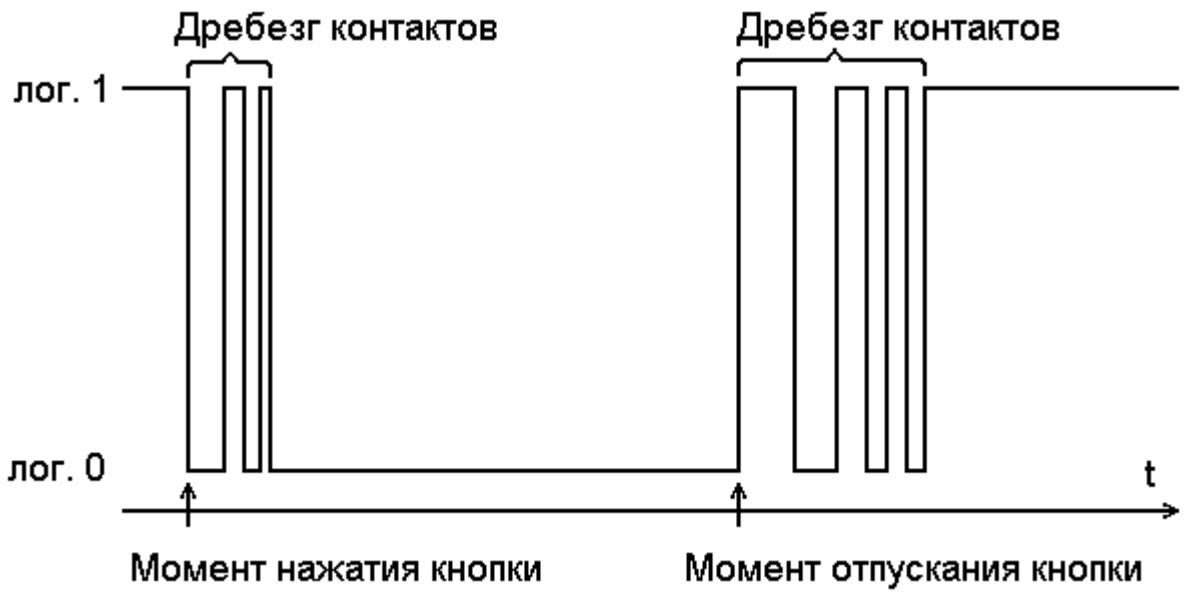
- **Blinker(uint8_t p)** — конструктор. В качестве аргумента **p** принимает номер GPIO, через который осуществляется управление светодиодом
- **void setup()** — метод, вызываемый в функции **setup()** главного модуля **jjrpulser.ino**. В нем осуществляется первоначальная настройка — назначение OUTPUT-режима работы GPIO, первичное запоминание временных меток
- **void work()** — метод, осуществляющий полезную работу в выделяемый квант времени. Вызывается в функции **work()** главного модуля **jjrpulser.ino**. Алгоритм вычисляет, сколько прошло времени от зафиксированной метки, и, при необходимости, гасит, или зажигает светодиод, согласно информации, содержащейся для текущего режима в нужном элементе массива **m_blinkTimers**
- **Mode mode()** — метод возвращает текущий режим работы
- **void setMode(Mode m)** — метод устанавливает текущий режим работы, передаваемый, как аргумент **m**

Ниже перечислены приватные члены класса **Blinker**. Это дополнительная информация для углубленного понимания деталей реализации алгоритмов.

- **uint8_t m_pinNumber** — здесь конструктор сохраняет номер GPIO, через который осуществляется управление светодиодом
- **unsigned long m_lastTimestamp** — здесь хранится время последнего переключения светодиода вкл/выкл, чтобы отсчитывать от него **BlinkTimes::lowTime** или **BlinkTimes::highTime**, в зависимости от **m_subMode**
- **unsigned long m_fullTimestamp** — здесь хранится время последнего переключения режима работы **m_currentMode**, чтобы отсчитывать от него **BlinkTimes::fullTime**
- **uint8_t m_subMode** — здесь хранится текущее состояние GPIO, **LOW** или **HIGH**, т. е. выключен сейчас светодиод, или включен
- **Mode m_currentMode** — здесь хранится текущий режим работы

4.3. bouncer.cpp — модуль сглаживания дребезга контактов

Во время начала и окончания импульса на короткое время возникают переходные электрические процессы, которые приводят ко множественным ложным значениям при считывании с GPIO:



По этому считывание производится не напрямую, а через объект класса **PinBouncer**, отбрасывающий ложные значения и детектирующий настоящее переключение сигнала, не меняющееся на протяжении определенного времени.

PinBouncer
-m_pinNumber: uint8_t = p -m_pinName: String = n -m_timeout: unsigned long = t -m_lastPinState: int = HIGH -m_lastStableState: int = HIGH -m_lastTimestamp: unsigned long = 0 -m_newValue: bool = false -m_isStable: bool = false
+PinBouncer(p:uint8_t,n:const char *,t:unsigned long=500) +setup(): void +work(): void +stable(): bool +newValue(): bool +resetNewValue(): void +value(): int +name(): const char *

Используемым номерам GPIO через макросы даны следующие имена:

- **COLD_PIN_NUMBER** — GPIO 4, к которому присоединен импульсный выход счетчика холодной воды
- **HOT_PIN_NUMBER** — GPIO 5, к которому присоединен импульсный выход счетчика горячей воды

Работа с объектами класса **PinBouncer** производится через следующие методы:

- **PinBouncer(uint8_t p, const char *n, unsigned long t = 500)** — конструктор, принимает следующие аргументы инициализации:
  - **p** — номер GPIO, на который поступают импульсы со счетчика

    - **n** — символьное имя объекта
    - **t** — если в течение этого времени (в миллисекундах) сигнал GPIO не менялся, считается, что получено стабилизированное (без дребезга) значение. По умолчанию задано 500мс
- **void setup()** — метод, вызываемый в функции **setup()** главного модуля **jjrpulser.ino**. В нем осуществляется первоначальная настройка — назначение INPUT-режима работы GPIO, первичное запоминание значения сигнала GPIO и текущей временной метки
- **void work()** — метод, осуществляющий полезную работу в выделяемый квант времени. Вызывается в функции **work()** главного модуля **jjrpulser.ino**. Алгоритм проверяет, изменился ли сигнал GPIO с последнего вызова **work()**, вычисляет, сколько прошло времени от последней смены сигнала, и, при необходимости, выставляет, или, наоборот, сбрасывает флаги стабилизации и наличия нового стабилизированного сигнала
- **bool stable()** — возвращает **true**, если сигнал уже стабилизировался (не менялся на протяжении заданного времени). Иначе возвращает **false**
- **bool newValue()** — возвращает **true**, если текущий стабилизированный сигнал изменился относительно последнего, т. е. произошла настоящая, не ложная, смена значения сигнала. Иначе возвращает **false**
- **void resetNewValue()** — когда программист обработал возврат метода **newValue()** == **true**, он вызывает этот метод, чтобы сбросить текущий детект переключения сигнала и перейти к ожиданию нового переключения
- **int value()** — метод возвращает текущее значение GPIO
- **const char * name()** — метод возвращает символьное имя объекта, заданное при конструировании

Класс **PinBouncer** имеет следующие приватные члены, используемые в реализации алгоритмов:

- **uint8_t m_pinNumber** — здесь конструктор сохраняет номер GPIO, к которому присоединен импульсный выход счетчика воды
- **String m_pinName** — здесь конструктор сохраняет символьное имя объекта, на данный момент используемое только в отладочных целях
- **unsigned long m_timeout** — здесь конструктор сохраняет время, необходимое для стабилизации сигнала
- **int m_lastPinState** — здесь алгоритмы хранят последнее считанное значение GPIO
- **int m_lastStableState** — здесь алгоритмы хранят последнее стабилизированное значение GPIO
- **unsigned long m_lastTimestamp** — здесь алгоритмы хранят время последнего изменения значения GPIO

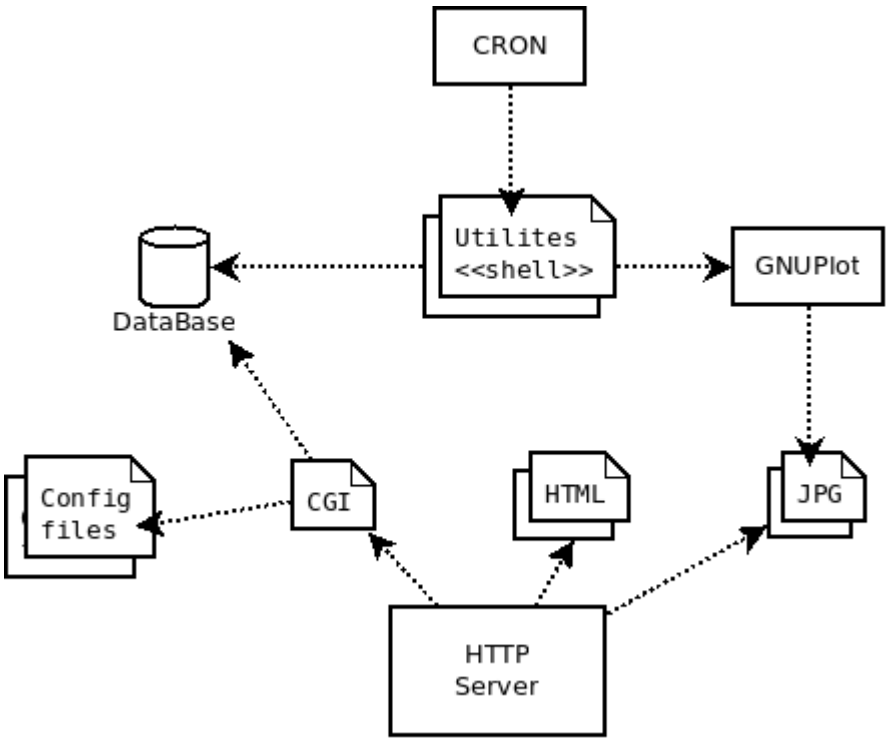
- **bool m_newValue** — этот флаг алгоритмы выставляют при смене стабилизированного значения GPIO
- **bool m_isStable** — этот флаг алгоритмы выставляют при стабилизации значения GPIO

4.4. httprequest.cpp — модуль отправки http-запросов

4.5. storage.cpp — модуль управления показаниями и статистикой

5. Web Server

Используется аппаратный сервер в виде одноплатного компьютера FriendlyElec NanoPi Fire3 под управление операционной системы (дистрибутива Linux) Ubuntu. Программный комплекс состоит из следующих компонент:



5.1. HTTP сервер

Предоставляет доступ к статическим страницам **HTML**, картинкам **JPG** и взаимодействует с **CGI**-программами. Статический контент **HTML**/**JPG**, используемый при построении интерфейса, представлен следующими файлами:

- **index_header.html** — «шапка» html-страницы, заголовок, цветовая гамма
- **index_input.html** — форма корректировки показаний

- **index_graphs.html** — таблица для размещения картинок с графиками
- **index_footer.html** — «подвал» html-страницы, набор стандартных html-тэгов, завершающих объявление страницы
- **index.css** — каскадная таблица стилей (описание деталей внешнего вида), используемая элементами html-страницы
- **jjrpulser.js** — java script (язык программирования, используемый внутри браузера) логика, отвечает за загрузку и переключение картинок с графиками
- **water_cold.png**, **water_hot.png** — картинки со счетчиками, поверх которых выводятся цифры показаний

Используется программный сервер **LigHttpd**, входящий в состав дистрибутива Ubuntu.

5.2. CGI программы

Динамически генерируют содержимое страниц, отображаемых **http-сервером**. Так же обрабатывают запросы, приходящие в **http-сервер**. Реализованы, как набор скриптов на языке shell/**bash**. Используются следующие скрипты:

- **index.sh** — основной скрипт, динамически генерирующий пользовательский интерфейс, отображающий показания счётчиков, графики потребления воды, форму корректировки показаний и статистику работы микроконтроллера
- **cmd.sh** — скрипт, обслуживающий служебные http/get запросы от микроконтроллера и пользовательского интерфейса. Взаимодействует с базой данных и конфигурационными файлами. Обслуживает следующие команды:
  - **setup** — с помощью этой команды форма корректировки передает скорректированные значения. Скрипт помещает эти значения в конфигурационный файл **setup_counters.txt**. Значения из этого файла будут переданы в микроконтроллер при его обращении к серверу. Так же будут отображаться в интерфейсе (их считает скрипт **index.sh**), как ожидающие отправки в устройство. Отрицательные значения игнорируются. Имена используемых парамеров — **cold, hot**. Пример get-запроса:

    `http://server.my/cmd.sh?cmd=setup&cold=123&hot=456`
  - **add_value** — с помощью этой команды микроконтроллер передает показания, считанные со счетчиков. Скрипт заносит показания в базу данных, откуда они считываются при отображении в интерфейсе и построении графиков. Микроконтроллер так же передает свой MAC-адрес, который сейчас нигде не используется. Имена используемых парамеров — **cold, hot, mac**. Пример get-запроса:

    `http://server.my/cmd.sh?cmd=add_value&cold=123&hot=456&mac=11:22:33:44:55:66`
    Если на момент запроса в файле **setup_counters.txt** есть данные, они передаются микроконтроллеру в теле ответа строками вида:

    ```
    setup_new_cold = 123
    ```

`setup_new_hot = 456`
после чего файл **setup_counters.txt** очищается

- **statistics** — с помощью этой команды микроконтроллер передает служебную отладочную статистику раз в 15 минут. Скрипт сохраняет статистику в конфигурационный файл **statistics.txt**, откуда она считывается (скриптом **index.sh**) при отображении в интерфейсе. Статистика включает в себя время работы микроконтроллера без перезагрузки, его свободную память и отчет об отправленных **http-серверу** запросах. Имена используемых парамеров — **mac**, **uptime_days**, **uptime_hours**, **uptime_minutes**, **uptime_seconds**, **uptime_millis**, **free_heap**, **http_req_sent**, **http_req_commited**, **http_req_failed**. Пример get-запроса:

  `http://server.my/cmd.sh?cmd=statistics&mac=11:22:33:44:55:66&uptime_days=10&uptime_hours=2&uptime_minutes=15&uptime_seconds=30&uptime_millis=0&free_heap=64000&http_req_sent=12456&http_req_commited=123456&http_req_failed=0`

- **mysql_settings.sh** — подключаемый модуль. В него вынесены настройки базы данных, функции SQL-запросов, записи и чтения конфигурационных файлов

5.3. Конфигурационные файлы

Когда нецелесообразно хранить временные данные небольшого объема в базе данных, они записываются в конфигурационные файлы. На данный момент используются следующие конфигурационные файлы:

- **setup_counters.txt** — для хранения корректировки значений
- **statistics.txt** — для хранения статистики микроконтроллера

Более подробно использование этих файлов описано в главе CGI программы

5.4. База данных

Показания счетчиков хранятся в базе данных с поддержкой SQL-запросов. База данных участвует в следующих сценариях:

- Микроконтроллер передает показания **http-серверу** в get-запросе с командой **add_value**. Сервер запускает **CGI**-скрипт **cmd.sh**. Скрипт разбирает параметры запроса и сохраняет их в **базе данных**, используя следующие SQL-запросы:

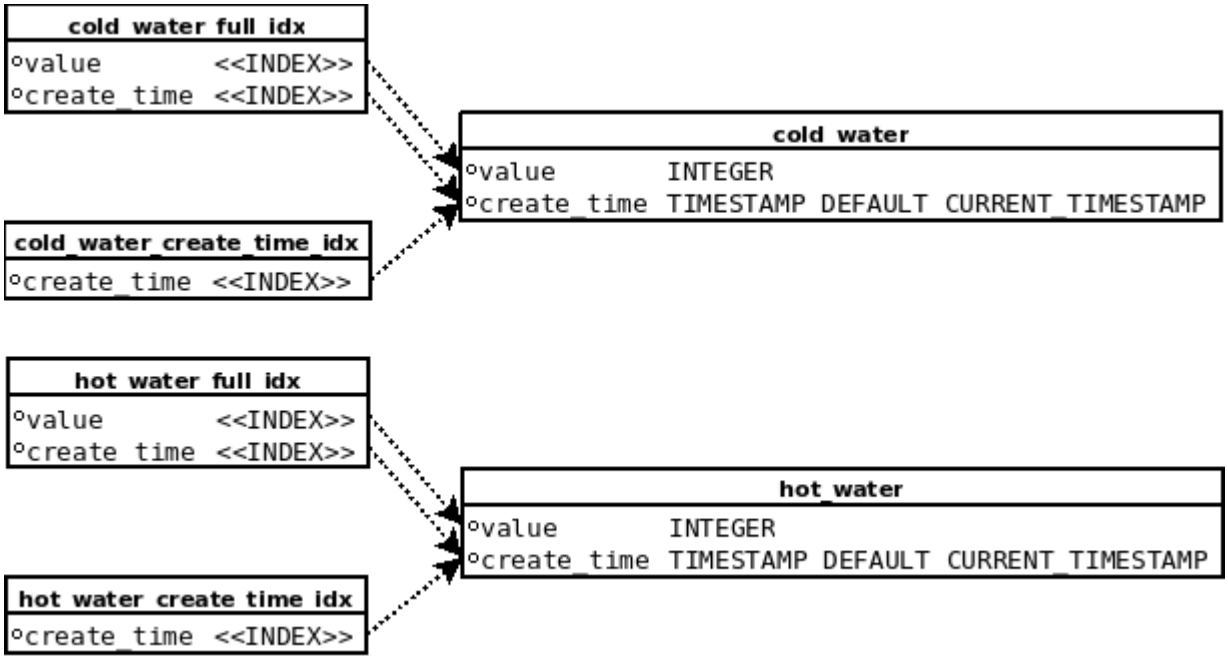
  ```
  INSERT INTO cold_water(value) VALUES(123)
  INSERT INTO hot_water(value) VALUES(456)
  ```
  Поле **create_time** явно не указывается, согласно спецификации в таблице, оно автоматически заполняется текущей датой и временем

- Пользователь обращается к **http-серверу** через браузер. Сервер запускает **CGI**-скрипт **index.sh**. Скрипт, кроме прочего, отображает в генерируемой http-странице текущие показания, запрашивая их из **базы данных** с помощью следующих SQL-запросы:

  ```
  SELECT create_time, value FROM cold_water ORDER BY create_time DESC LIMIT 1
  SELECT create_time, value FROM hot_water ORDER BY create_time DESC LIMIT 1
  ```

- Планировщик **Cron** запускает по расписанию **утилиты**, генерирующие графики. Утилиты обращаются к **базе данных** и складывают результаты запросов в файлы. Графопостроитель **GNU Plot** строит графики на основе этих файлов.

  Данные для графиков формируются сложными вложенными SQL-запросами с группировками по полю **create_time**. Ознакомиться с ними можно в скриптах **mysql_generate_graph.sh** и **mysql_generate_graph2.sh**



База данных содержит следующие элементы:

- **cold_water** — таблица показаний холодной воды, состоит из
  - **value** — целочисленное поле с абсолютным значением показания
  - **create_time** — поле, хранящее время занесения показания. В штатном режиме время явно не указывается, вместо этого база данных подставляет текущее
- **cold_water_create_time_idx** — служебный индекс, ускоряющий поиск по полю времени
- **cold_water_full_idx** — служебный индекс, ускоряющий поиск по комбинации обоих полей
- Для горячей воды структура полностью аналогична

В качестве базы данных используется **MariaDB** (бывшая **MySQL**), входящая в состав дистрибутива Ubuntu.

5.5. Утилиты

Некоторые функции системы выделены в отдельные утилиты, реализованные, как shell/bash скрипты. Основные утилиты, запускаемые планировщиком:

- **mysql_generate_graph.sh** — утилита, генерирующая графики потребления воды в абсолютных показателях. Принимает, как аргумент, следующие диапазоны генерации:

    - **day** — за прошедший день, на выходе изображение **mysql_jjrpulser_day.png**
    - **week** — за прошедшую неделю, на выходе изображение **mysql_jjrpulser_week.png**
    - **month** — за прошедший месяц, на выходе изображение **mysql_jjrpulser_month.png**
    - **year** — за прошедший год, на выходе изображение **mysql_jjrpulser_year.png**
- **mysql_generate_graph2.sh** — утилита, генерирующая графики потребления воды в виде набора относительных интервалов. Принимает в виде аргумента, следующие диапазоны генерации, в которых выделены соответствующие интервалы:
    - **day** — за прошедший день, с разбивкой по часам, на выходе изображение **mysql_jjrpulser2_day.png**
    - **week** — за прошедшую неделю, с разбивкой по часам, на выходе изображение **mysql_jjrpulser2_week.png**
    - **month** — за прошедший месяц, с разбивкой по дням, на выходе изображение **mysql_jjrpulser2_month.png**
    - **year** — за прошедший год, с разбивкой по месяцам, на выходе изображение **mysql_jjrpulser2_year.png**

Так же существует набор утилит, не используемый при штатной работе сервера, но предоставляющий удобные функции при развертывании, настройке и тестировании:

- **force_generate_graph.sh** — принудительно, без участия планировщика, запускает перегенерацию изображений всех возможных графиков
- **mysql_clear.sh** — удаляет все данные, таблицы и базу данных. Заново создает пустую базу данных и пустые таблицы
- **mysql_refill-db.sh** — генерирует файл **jjrpulser.sql**, содержащий SQL-инструкции для заполнения базы данных фальшивыми показаниями за период с 2017-01-01 по 2019-06-01

5.6. CRON

Чтобы избежать перегрузки сервера, графики потребления воды генерируются не каждый раз, при обращении пользователя к интерфейсу, а по расписанию. Интерфейс показывает заранее сгенерированные по расписанию картинки с графиками. Используется наиболее распространенный в Linux дистрибутивах планировщик — **CRON**. Он запускает по расписанию утилиты **mysql_generate_graph.sh** и **mysql_generate_graph2.sh**, которые генерируют изображения с графиками. Расписание запуска таково:

- Дневные графики строятся раз в 5 минут
- Недельные графики строятся раз в 15 минут
- Месячные графики строятся раз в час

- Годовые графики строятся раз в сутки

В формате crontab это выглядит следующим образом:

```
1-59/5 * * * * /var/www/html/jjrpulser/mysql_generate_graph.sh day
1-59/5 * * * * /var/www/html/jjrpulser/mysql_generate_graph2.sh day

3-59/15 * * * * /var/www/html/jjrpulser/mysql_generate_graph.sh week
3-59/15 * * * * /var/www/html/jjrpulser/mysql_generate_graph2.sh week

5 * * * * /var/www/html/jjrpulser/mysql_generate_graph.sh month
5 * * * * /var/www/html/jjrpulser/mysql_generate_graph2.sh month

0 3 * * * /var/www/html/jjrpulser/mysql_generate_graph.sh year
0 3 * * * /var/www/html/jjrpulser/mysql_generate_graph2.sh year
```

Эта конфигурация может быть найдена в файле **example.crontab**

5.7. GNU Plot

Утилиты для построения графиков запрашивают данные из базы данных и формируют файлы со столбцами «дата, время» и «значение», разделенными символом «|» :

```
2019-03-30 23:32:03 | 190
2019-03-31 22:12:06 | 470
2019-04-01 01:15:43 | 480
```

Для генерации из этих файлов изображений с графиками используется графопостроитель **GNU Plot**. Кроме файлов с входными данными, ему требуется программа, описывающая, как эти данные необходимо интерпретировать. Утилиты генерируют для него программу генерации графика потребления воды в абсолютных показателях (имена файлов и значения полей условно-показательные, в реальном режиме работы используются динамически вычисляемые утилитами):

```
set terminal png truecolor size 1024, 600
set output "output_image.png"
set datafile separator "|"
set timefmt '%Y-%m-%d %H:%M:%S'
set xdata time
set format x "%b\n%Y"
set style data steps
set xtics autofreq
set xtics out
unset mxtics
set tics font ", 8"
set grid
set timestamp

set xrange [ "2018-04-09 01:49:04" : "2019-04-04 01:49:04" ]
set yrange [ 0 : * ]

plot "file.cold" using 1:2 notitle with impulses lc rgb "#ADD8E6", \
     "file.hot" using 1:2 notitle with impulses lc rgb "#CCFF69B4", \
     "file.cold" using 1:2 title "Cold water" lc rgb "blue" lw 2, \
     "file.hot" using 1:2 title "Hot water" lc rgb "red" lw 2
```

И программу генерации графика потребления воды в виде набора относительных интервалов (имена файлов и значения полей условно-показательные, в реальном режиме работы используются динамически вычисляемые утилитами):

```
set terminal png truecolor size 1024, 600
set output "output_image.png"
```

```
set datafile separator "|"
set timefmt '%Y-%m-%d %H:%M:%S'
set xdata time
set format x "%H:%M\n%d.%m"
set boxwidth 1800
set style fill solid
set xtics 3600
set xtics out
unset mxtics
set tics font ", 8"
set grid
set timestamp

set xrange [ "2019-04-03 01:49:04" : "2019-04-04 01:49:04" ]
set yrange [ 0 : * ]

plot "file.cold" using (timecolumn(1)):2 title "Cold water" with boxes lc rgb
"#770000FF", \
     "file.hot" using (timecolumn(1) + 450):2 title "Hot water" with boxes lc
rgb "#77FF0000"
```