Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Воронежский государственный университет»

Физический факультет
Кафедра электроники

Разработка системы интеллектуального управления освещением

03.04.03 «Радиофизика»
Профиль «Информационные системы и технологии»

Обучающийся		В. А. Сафонов
Руководитель	к.т.н.	А. С. Жабин
Зав. кафедрой	д.ф.-м. н., профессор	А. М. Бобрешов

Воронеж

2020

Содержание

Введение

Моя магистерская работа посвящена разработке системы интеллектуального управления освещением для оптимизации энергопотребление осветительных приборов. Её основными преимуществами являются эффективное энергопотребление, и, как следствие, экологичность. Добавив к этим свойствам ещё и дешевизну с простотой в эксплуатации, мы получаем систему, которая будет пользоваться спросом на современном рынке аналогичных систем. Особенно полезна она будет в больших офисных зданиях или на промышленных объектах. Выгодность данного проекта подтверждается отчетом Navigant Research [1], согласно которому мировой доход от сетевых средств контроля освещения будет расти с 1,7 млрд. долларов США в год в 2013 году до более чем 5,3 млрд. долларов США к 2021 году . Вся работа была разделена на 4 части:

1. Разработка общего вида системы. Постановка целей на будущее
2. Разработка технической части системы
3. Разработка алгоритма оптимизации освещения.
4. Разработка сети. Сбор и обработка статистики

В последней части моей работы мы должны подобрать оптимальный способ сбора, хранения и обработки информации от наших устройств. Т.е. необходимо подобрать протокол передачи данных, способ хранения данных и сделать какие то выводы из данных.

1. Теоретическая часть

1.1. Описание компонент системы

1.1.1. Микроконтроллер STM32F100

STM32F100 младшая линейка дешевых микроконтроллеров от компании STMicroelectronics на базе ядра Cortex-M3. Микроконтроллеры включают в себя широкий набор интерфейсов и большой объем встроенной памяти: ядро Cortex-M3 с частотой процессора до 24 МГц, Flash до 512 кБ, до 32 кБ RAM, большее количество таймеров, часы реального времени (RTC), до 5 UART, до 2 I2C, до 3 SPI, 12-битный АЦП и 12-битный ЦАП, встроенный температурный датчик, а также контроллер внешней памяти (EMC). Микроконтроллеры семейства STM32F100x («Value Line») предназначены для различных крайне чувствительных к стоимости применений, где возможностей 16-битного микроконтроллера уже недостаточно, а функциональность обычных 32-битных микроконтроллеров избыточна. Выпускаются в корпусах: LQFP48, LQFP64, TFBGA64, LQFP100, LQFP144.

Основные характеристики линейки STM32F100x («Value Line»):

- Максимальная тактовая частота 24 МГц (30 DMIPS)
- Умножение и деление за 1 такт
- Напряжения питания 2.0 – 3.6 В
- От 4 до 8 Кб ОЗУ
- От 16 до 128 Кб флэш-памяти
- Два встроенных и откалиброванных тактовых генератора на 40 КГц и 8 МГц
- 7-канальный DMA контроллер
- 16-канальный 12-битный АЦП (1.2 мкс) с датчиком температуры
- Два 12–битных ЦАП
- До 80 быстрых портов ввода – вывода (есть совместимость с 5 В)
- 16 внешних прерываний

- Два сторожевых таймера (IWDG и WWDG)
- До 10 таймеров общего и расширенного назначений
- До 2x I2C(SMBus/PMBus), до 3x USART (Lin, IrDa, modem control), до 2 SPI(2 Мбит/с), HDMI (CEC), RTC
- Управление питанием и сбросом (3 режима низкого потребления, PVD, BOR)
- Аппаратный расчет CRC
- 96–битный уникальный идентификатор (ID)

1.1.2. **Фотодатчик KY-018**

Принцип работы данного датчика довольно прост: Чем ярче освещен фоторезистор, тем ниже его сопротивление. Сопротивление фоторезистора при изменении освещенности меняется в широких пределах от единиц кОм и до сотен кОм или МОм. Точно выяснить изменение сопротивление фоторезистора следует экспериментально с помощью омметра. Контакты и схема модуля KY-018 позволяют использовать только фоторезистор или фоторезистор в составе делителя напряжения. Схема устройства на рисунке 2 в приложении Б.

Для этого на плате установлен резистор 10 кОм. Питание модуля подают на контакт +5 В. С увеличением освещенности на выходе модуля фоторезистора напряжение будет падать, при ярком свете напряжение выхода будет около половины напряжения питания. Величина напряжения на выходе зависит от типа фоторезистора. В темноте напряжение выхода будет близко к напряжению контакта +5 В.

При работе совместно с МК выход модуля фоторезистора соединяют с входом АЦП микроконтроллера. Так как изменение сопротивления фоторезистора при освещении значительно, то с помощью АЦП можно легко фиксировать наступление темноты или включение освещения. Схема АЦП МК STM32F100 представлена на рисунке 5 в приложении Б.

1.1.3. **Wi-fi модуль esp8266 - 01**

ESP8266 — микроконтроллер китайского производителя Espressif с интерфейсом Wi-Fi. Помимо Wi-Fi микроконтроллер отличается возможностью исполнять программы из внешней флеш-памяти с интерфейсом SPI.

Основное применение ESP8266 находит в управлении разнообразными бытовыми приборами через беспроводные сети. Концепцию такого управления часто называют «Internet of Things» (IoT, «интернет вещей»). Верхний уровень IoT представлен разнообразными приложениями под популярные платформы (Android, iOS, Windows). Эти приложения позволяют разработчику прибора адаптировать приложение под управление его прибором и передать пользователю готовое решение.

Всего существует 13 модификаций данного модуля от esp8266-01 до esp8266-13. Они отличаются размерами, антеннами, наличием или отсутствием экрана, а также количеством и назначением ножек на плате.

На плате имеется свой встроенный микроконтроллер с параметрами:

- 80 MHz 32-bit процессор Tensilica L106. Возможен негарантированный разгон до 160 МГц.
- IEEE 802.11 b/g/n Wi-Fi. Поддерживается WEP и WPA/WPA2.
- 14 портов ввода-вывода(из них возможно использовать 11), SPI, I²C, I²S, UART, 10-bit АЦП.
- Питание 2,2…3,6 В. Потребление до 215 мА в режиме передачи, 100 мА в режиме приема, 70 мА в режиме ожидания. Поддерживаются три режима пониженного потребления, все без сохранения соединения с точкой доступа: Modem sleep (15 мА), Light sleep (0.4 мА), Deep sleep (15 мкА).

Но в ходе данной работы использовать непосредственно этот микроконтроллер мы не будем, так как у нас уже есть внешний программируемый микроконтроллер stm32f100.

Модуль esp8266-01 является простейшей модификацией платы esp8266. Она отличается простотой в использовании, полосковой антенной (PCB antenna) с радиусом действия до 400 метров на открытой местности. Модуль управляется AT командами.

1.1.4. **Датчик движения HC-SR501**

Модуль представляет собой датчик движения человека. При вхождении человека в зону обзора датчика фиксируется присутствие. Принцип работы модуля HC-SR501 заключается в регистрации инфракрасного излучения от подвижного объекта. Чувствительный элемент – пироэлектрический датчик 500BP. Он состоит из двух элементов заключенных в одном корпусе. Чувствительный элемент закрыт белым куполом – линзой Френеля. Особенности линзы Френеля таковы, что инфракрасное излучение от подвижного объекта попадает сначала на один элемент датчика 500BP, затем на другой. Электроника модуля HC-SR501 регистрирует поочередное поступление сигналов от двух элементов из состава 500BP и при фиксации движения выходная цепь модуля формирует логический сигнал.

Технические характеристики “HC-SR505”:

- Напряжение питания постоянного тока, В: 4.5-20
- Потребляемый ток (в покое), мкА: < 60
- Потребляемый ток (при срабатывании), мкА: < 80
- Выходной уровень, В: высокий 3.3 / низкий 0
- Длительность высокого уровня на выходе после срабатывания, С: 8 ± 30%
- Угол зоны обнаружения, градусов: < 100
- Дальность обнаружения, м: 2-3
- Рабочая температура: -20…+80
- Диаметр линзы датчика, мм: 10
- Размеры платы, мм: 10*23

1.1.5. Message Queue Telemetry Transport

MQTT – это простой открытый протокол, работающий поверх TCP/IP, обмена информацией. Расшифровывается как Message Queue Telemetry Transport. В данный момент это наиболее популярный проток в сфере интернета вещей.

Основные особенности протокола MQTT:

- •Асинхронный протокол
- •Компактные сообщения
- •Работа в условиях нестабильной связи на линии передачи данных
- •Поддержка нескольких уровней качества обслуживания (QoS)
- •Легкая интеграция новых устройств

Обмен информацией происходит между клиентом, который может работать в режиме издателя или подписчика, и брокером. Издатель отправляет данные на MQTT брокер, указывая в сообщении определенную тему, топик. Подписчики могут получать разные данные от множества издателей в зависимости от подписки на соответствующие топики.

Каждое сообщение в MQTT имеет нескольких частей (рис. А2):

1. Байт 1: содержит тип сообщения (запрос клиента на подключение, подтверждение подписки, запрос ping и т. д.), флаг дублирования, инструкции для сохранения сообщений и информацию об уровне качества обслуживания (QoS).

2. Байт 2: содержит информацию об оставшейся длине сообщения, включая полезную нагрузку и любые данные в заголовке необязательной переменной.

Флаг QoS в байте 1 заслуживает особого внимания, так как он лежит в основе переменной функциональности, которую поддерживает MQTT. Флаги QoS содержат следующие значения, основанные на намерении и срочности сообщения:

1. 0 = не более одного раза: сервер срабатывает и забывает. Сообщения могут быть потеряны или продублированы

2. 1 = по крайней мере один раз: получатель подтверждает доставку. Сообщения могут дублироваться, но доставка гарантирована

3. 2 = ровно один раз: сервер обеспечивает доставку. Сообщения поступают точно один раз без потери или дублирования

В качестве брокера сообщений был выбран mosquitto.

Eclipse Mosquitto — брокер сообщений с открытым исходным кодом (лицензии EPL/EDL), который реализует протоколы MQTT версий 5.0, 3.1.1 и 3.1. Mosquitto лёгкий и подходит для использования на всех устройствах: от маломощных одноплатных компьютеров до полноценных серверов.

1.1.6. Способ хранения данных

В качестве средства хранения информации была выбрана база данных InfluxDB. Т.к. данные у нас простые и им не требуются сложные таблицы, но в тоже время мы будем получать постоянный поток данных. Поэтому используем базу данных временных рядов. База данных временных рядов оптимизирована для быстрого приёма данных. Такие системы используют индексацию данных, объединённых со временем. Как следствие, скорость загрузки не уменьшается со временем и остается достаточно стабильной (от 50 до 100 тыс. строк в секунду на одном узле). К тому же, использование именно этой бд решает проблему последующей визуализации т. к. она работает с большинством инструментов мониторинга вроде Grafana.

Недостатки:

- нет update. для этого используется вставка с тем же ключем (временем)

- delete очень долгий (от нескольких секунд на удаление одной записи)

- при добавления нового тега, фильтр по этому тегу для существующих данных не доступен

Также к недостаткам данной БД можно причислить специфичный язык запросов — InfluxQL.

Важной концепцией данной БД является привязка всех данных ко времени. Т.е. каждая запись имеет свою временную метку в виде timestamp, определенный в стандарте RFC3339.

1.1.7. Средства визуализации данных

Grafana - это кроссплатформенное веб-приложение для аналитики и интерактивной визуализации с открытым исходным кодом . Она позволяет отображать данные в виде диаграмм, графиков и дэшбордов.

Её плюсами является возможно тонкой настройки средств визуализации. Она берет данные из БД и отображает в удобной нам форме.

Кроме того, она позволяет не только банально строить графики, но проводить сначала анализ этих данных, а потом уже отображать обработанные данные.

1.2. Основные принципы работы с уровнем освещения

1.2.1. Определение комфортного уровня освещенности

Для определения значения среднего комфортного уровня освещенности воспользуемся принятыми нормами и стандартами. Для работы с освещенностью будем использовать люксы. Люкс – это единица измерения освещённости в Международной системе единиц. Люкс равен освещённости поверхности площадью 1 м² при световом потоке падающего на неё излучения, равном 1 лм. Соответственно, выполнятся: 1 лк = 1 лм/м2. С другой стороны, люкс равен освещённости поверхности сферы радиусом 1 м, создаваемой точечным источником света, находящимся в её центре, сила света которого составляет 1 кд. Таким образом, с основными единицами СИ люкс связан соотношением: 1 лк = 1 кд/м^2.

В качестве основного нормирующего документа используем ГОСТ Р 55710-2013. Называется он «Освещение рабочих мест внутри зданий. Нормы и методы измерений» и был разработан обществом с ограниченной ответственностью "Всероссийский научно-исследовательский, проектно-конструкторский светотехнический институт им.С.И.Вавилова" (ООО "ВНИСИ"). Данный документ устанавливает нормы освещения рабочих мест, обеспечивающие безопасные и комфортные условия труда. Данный документ утверждает, например, что в административных зданиях эксплуатационная освещенность рабочих мест с монитором должна быть равна 500лк, а в проходах и зонах движения 300лк. Эти значения мы примем за комфортные и будет использовать при дальнейших расчётах.

1.2.2. Определение текущего уровня освещения в точке

За определение текущего уровня освещенности в данной точке помещения отвечает модуль KY-018, подключенный к АЦП микроконтроллера. Как известно, сопротивление фоторезистора, лежащего в основе модуля, будет меняться с изменением уровня освещения. Диапазон сопротивлений для данной платы от 48 до 500 кОм, где 500кОм — теневое сопротивление. Сопротивление при 10 люкс: 24 ± 12 кОм. В итоге данные на АЦП микроконтроллера будут меняться. При этом напряжение, подаваемое на вход АЦП, будет конвертировано в число от 0 до 1024. В этом месте нам необходимо соотнести значения в 500лк и 300лк с соответствующими значениями на выходе АЦП. Произвести это соотношение возможно только экспериментальным путём. В зависимости от этих значений наше устройство в будущем будет принимать множество значений. Назовём эти значения E_{500} и E_{300}.

1.2.3. Определение освещенности в помещении

Очевидно, что решение об изменении уровня освещения по данным от всего лишь одного датчика не всегда будет верно. Так как важно контролировать не только освещение на точкой измерения его уровня, но и в зоне непосредственного окружения. Зона непосредственного окружения — это зона шириной не менее 0,5 м, окружающая зону зрительной работы внутри поля зрения . Для рабочих мест в этой зоне значение освещенности должно быть не менее 300лк, а для зон движения не менее 200лк. Из этого следуют, что для принятия каких-либо решений микроконтроллер должен знать не только данные от своего датчика освещенности и своё местоположение (рабочее место или зона движения), но и данные и местоположение своих соседей. При этом важен такой параметр как равномерность освещенности. Равномерность освещенности - это отношение значения минимальной освещенности к значению средней на заданной поверхности. Для офисных помещений должен быть равен не менее 0.6. Отсюда следует, что для верного принятия решений помещение должно быть верно и непрерывно оборудовано датчиками освещенности.

1.3. Распределение датчиков по помещению

1.3.1. Распределение датчиков освещенности

Конечно же, для каждого помещения наиболее точным будет индивидуальное расположение датчик, но всегда будет ряд параметров, которым расположение должно удовлетворять. Попробуем их перечислить:

- Датчик обязательно должен быть в зоне зрительной работы.
- Если рабочая зона локально не ограничена стенами, то устройство, принимающее решение об изменении уровня освещенности должно иметь данные от ближайших соседей.
- На открытом пространстве у каждого датчика должны быть соседи, образующие зону непосредственного окружения
- Датчики, образующие зону непосредственного окружения должны находится на расстоянии не менее 0.5м.

Это минимальный набор условий для правильного размещения датчика освещенности.

1.3.2. Расположение датчиков освещенности на примере офисного помещения

Рассмотрим простейший пример офисного помещения, а конкретно офис, спланированный по принципу High-panelled cubicles, где для принятия решений об управлении освещением не потребуются «соседи». Смоделируем это помещение и естественное световое освещение в нём в программе Velux daylight visualizer 2. Общий план можно увидеть в приложении А на рисунке А.1.1. На рисунке А.1.2. видно распределение естественного освещения по комнате.

Каждый из офисов можно принять за рабочую поверхность, а сам стол непосредственно за зону зрительной работы. Именно в районе рабочего стола и должен находится датчик, ответственный за измерение уровня освещенности данного офиса.

Отдельной зоной будет коридор в середине помещения. Пусть он освещается тремя осветительными приборами и окном. Он достаточно протяженный поэтому, казалось бы, тут потребуется несколько датчиков. Но для рационального управления достаточно всего лишь одного датчика, измеряющего освещенность в районе окна, так как в других зонах коридора нет естественных источников света, и управлять на основе данных об освещенности имеет смысл только осветительным прибором, дублирующим окно. Все остальные решения об освещение в коридоре будут приниматься на основе датчика движения.

Финальное размещение датчиков освещения на рисунке А.1.3.

1.4. Применение датчиков движения для отслеживания присутствия

1.4.1. Обоснование применения датчиков движения

Для эффективного управления освещением в помещении необходимо отслеживать присутствие человека внутри этого помещения. В большинстве случаев в помещении должно быть выключены все осветительные приборы при отсутствии человека, а включать их следует только при присутствии человека, и отсутствии иных способов освещения.

Одиночные датчики движения плохо себя зарекомендовали для использования в офисных помещения, так как требуют постоянного движения и перемещений, а современный офис — это, прежде всего, люди, сидящие за мониторами.

Подходящим прибором был бы датчик присутствия, который в, свою очередь, имеет схожий принцип работы. Но за счёт использования в десятки раз большего количества линз, чем датчик движения, он более чувствителен и способен заметить даже малейшие движения, такие как удары пальце по клавиатуре или даже дрожь. Из-за этих линз использовать данный датчик невыгодно по экономическим соображениям, так как стоимость его вырастает в десятки раз. Так что даже использование нескольких датчиков движения экономически более целесообразно, чем один датчик присутствия, а при использовании специального алгоритма полностью избавиться от недостатков датчика движения.

1.4.2. Особенности применения датчиков движения в системе

В нашей системе датчики движения используются не для отслеживания движения, и не как более дешевая версия датчика присутствия, а для подсчёта количества людей в помещении в данный момент.

В каждом помещении есть одно или несколько мест, через которые необходимо пройти, чтобы войти в помещение или выйти из него. Обычно это дверь, рама, арка и т.д. Зная сколько человек прошло через эти места и в каком направлении, мы можем сделать вывод о наличие или соприсутствие людей в помещении.

Для измерения потока людей через проход достаточно одного датчика движения, но это не даст нам никакой информации о направлении движения. Например, если датчик сработал 200 раз, это значит, что через проход прошло 200 человек, но сколько людей сейчас в помещении: 0 или 200 (или любое другое количество в промежутке $0 \leq N \leq 200$)?

Однако, если взять два датчика и последовательно расположить их в проходе, то из последовательности, в которой сработают датчики, можно сделать вывод о направлении движения. Например, разделим проход (арку, коробку двери) на две части: ближнюю к помещению (назовём её выходом, так как при выходе её проходят первой) и дальнюю (назовём её входом по аналогии с предыдущей частью). Установим датчик движения А на „вход”, а датчик Б на „выход”. Если сначала сработал датчик А, а потом Б, то человек зашел в помещение и общий счётчик инкременируется, и наоборот, если сначала датчик Б, а потом А, то кто-то покинул помещение, следовательно общий счётчик декрементируется. Если общий счётчик равен нулю, то количество вошедших равно количеству вышедших, следовательно, помещение в данный момент пусто.

Счётчик людей в помещении может отдать команду на переход в спящий режим (или выход из него) всем устройствам, находящимся в зоне, за которую он отвечает. Разбиение по зонам идёт именно по этим счётчика, т.е по входам\выходам.

2. Алгоритм работы системы управления освещением

2.1. Общее описание алгоритма работы

Назначение данного алгоритма — это наиболее эффективное управление освещением в определенной зоне.

Для эффективного управления освещением система должна решать данные основные проблемы:

1. Работа освещения без необходимости
2. Недостаточность освещения
3. Избыточность освещение

Все решения принимаются на основе данных от:

1. Датчика освещенности
2. Датчика движения
3. Информации от соседних устройств

В результате обработки данных может быть принято одно из решений:

1. Отключение освещения
2. Включение освещения на определенный уровень
3. Корректировка уровня освещенности

Корректировка уровня освещенности происходит по известным нам уровням E_{500} и E_{300}. Так как существует не только проблема недостаточного освещение, но и проблема переосвещения, то мы будем придерживаться именно этих рекомендованных уровней. Кроме того при использовании более яркого освещения идёт перерасход энергии, а одним из обязательных качеств нашей системы является энергоэффективность.

Сам алгоритм можно разбить на 2 части: сбор информации, анализ информации и принятие решения.

Отдельным алгоритмом является корректировка освещения.

2.2. Алгоритм работы

Сам алгоритм можно разбить на 2 части: сбор информации, анализ информации и принятие решения. В общем виде его можно представить так:

1. Сбор информации

1.1. Обход списка соседей на соответствие их уровня освещенности значению освещенности зоны непосредственного окружения и формирование списка ответов

1.2. Получение значений со своих датчиков

2. Анализ информации и принятие решений

2.1. Обход списка ответов от соседей. Если какой-либо из ответов false, то отправляем этому соседу приказ на корректировку освещения

2.2. Если значения уровня освещенности не лежат в пределе $0.95E_{необ} \leq E_{текущая} \leq 1.05E_{необх}$, то корректируем собственное освещение

Блок схему алгоритма можно найти на рисунке А.2.1.

Далее рассмотрим алгоритм корректировки алгоритма.

1. Определение текущего уровня освещенности

2.1. Если освещение недостаточно, то

2.1.1. Проверяем включено или выключено освещение

2.2.1. Если выключено, то включаем со стандартным уровнем яркости

2.2.2. Если включено, то постепенно увеличиваем яркость пока не попадем в промежуток $0.95E_{необ} \leq E_{текущая} \leq 1.05E_{необх}$

3.1. Если освещение избыточно, то

3.1.1. Проверяем включено или выключено освещение

3.2.1. Если выключено, то ничего не делаем

3.2.2. Если включено, то постепенно уменьшаем яркость пока не попадем в промежуток $0.95E_{необ} \leq E_{текущая} \leq 1.05E_{необх}$

3.2.3. Если яркость минимальна, а $E_{текущая} \geq 1.3E_{необх}$, то выключаем освещение

2.3. Выводы ко второму разделу

Разработанный набор алгоритмов корректно решает такие критические для оптимального и эффективного освещения, как

1. Работа освещения без необходимости
2. Недостаточность освещения
3. Избыточность освещение

Кроме того была гарантирована надежность работы системы, так как “соседи“ в ходе работы гарантируют корректность работы друг друга.

3. Создание сети

3.1. Сервер и настройка MQTT брокера

В качестве сервера используется пк с Debian 10. Нам необходимо установить и настроить брокер. В качестве брокера, как уже писалось выше, был выбрать Mosquitto. Установим брокер и настроем шифрование для обеспечения безопасности передаваемых данных.

Установить mosquitto можно стандартными средствами Debian,а конкретно, командой «sudo apt install mosquitto mosquitto-clients».

Наиболее важной частью настройки брокера является установка SSL-шифрования. Для этого нам необходим сертификат Let’s Encrypt, и указать брокеру на место его хранения, и перезапустить серверв после этого.

3.2. Настройка MQTT на конечных устройствах

Частью моего устройства является wi-fi модуль esp8266, что позволяет легко использовать выбранный протокол. Используем открытую библиотеку esp_mqtt и настроим наши устройства в качестве клиентов.

Модуль через равные промежутки времени подключается к точке доступа wi-fi и устанавливает соединение с брокером, далее передается информация.

Код прошивки можно увидеть в приложении Б.

3.3. Передача данных от конечных устройств в БД

В данный момент информация от устройств доходит только до mqtt брокера. Для хранения и дальнейшей визуализации мы должны далее переправить данные в БД. Этот разрыв можно ликвидировать несколькими путями. Например, оформление брокера и бд в виде Docker контейнеров могло бы ликвидировать данный разрыв, но я не буду излишне усложнять свою систему и пойду более простым путём. Обеспечим передачу данных от брокера к базе данных путем создания так называемого «моста».

Для того, чтобы создать «мост» между mosquitto и influxDB на потребуется создать программу выполняющую данные ниже пять шагов:

1. ждем пока какой-нибудь клиент подключится к брокеру
2. подписываемся на этого клиента
3. получаем данные от брокера
4. Переводим данные в понятный для БД вид
5. отправляем их в БД

Стоит отметить, что взаимодействие с influxDB отличается от взаимодействия с SQL базами данных. Выборка, удаление, изменение, подключение к БД — всё это построено на основе rest api. Поэтому мы будем использовать специальную библиотеку, чтобы сократить размер скрипта и избежать ошибок.

В качестве «моста» был написан небольшой скрипт на языке программирования python.

Скрипт работает в режиме бесконечного цикла, что позволяет нам не пропустить время передачи данных от клиента к брокеру,т.е. исключается всякая вероятность потерять какие-либо данные.

Увидеть код скрипта можно в приложении Б2

3.4. Визуализация данных

Теперь, когда данные в нашей БД, попробуем визуализировать их.

После установки и запуска на сервере Grafana, нам остается сделать последний шаг, и указать адрес нашей базы данных в InfluxDB. Вся работа с grafana идет через графический интерфейс, что позволяет упростить взаимодействия. Остается лишь написать запрос в БД, который вернет нам набор данных, который необходимо визуализировать. Далее grafana все сделает сама. Пример получившегося графика можно увидеть в приложении А3.1

3.5. Расчёт себестоимости

Очевидно, что себестоимость внедрения системы будет зависеть от размера, типа и конфигурации помещения, в котором она будет установлена. В ходе предыдущих работ была рассчитана себестоимость отдельных модулей системы:

- Модуль управления освещением — 464.58 руб.
- Датчик освещенности - 298.88 руб.
- Датчик движения - 321.08 руб.

Попробуем рассчитать себестоимость установки данной системы в офис, который был описан в предыдущих главах.

Финальное расположение датчиков можно посмотреть на рис А.4., где чёрными точками отмечены датчики движения, красными - датчики освещенности, а желтыми — модули управления освещением.

Т.е. нам потребуется:

- 10 датчиков движения на входе в отдельные офисы + 1 на входе в само помещение
- 13 модулей управления освещением
- 11 датчиков освещенности

Итого: 10993.6 рублей.

Заключение

В ходе данной работы были выбраны оптимальные протокол передачи данных, способ хранения информации и средства визуализации. Также было разработано программное обеспечение отвечающее за получение данных, хранение данных, визуализацию данных. Важной частью этого ПО является алгоритм управления освещением.

В данный момент можно сказать, что вся работа прошла по принятому ранее плану, и была полностью выполнена в соответствии со сроками

Список литературы

Основная литература

1. Бройдо, В. Л. Вычислительные системы, сети и телекоммуникации: Учебник для вузов / В. Л. Бройдо. – Санкт-Петербург : ПИТЕР, 2004. – 702 с.
2. Предко, М. Руководство по микроконтроллерам. Том 1 / М. Предко. – Москва : Постмаркет, 2001. – 416 с.
3. Предко, М. Руководство по микроконтроллерам. Том 2 / М. Предко. – Москва : Постмаркет, 2001. – 418 с.

Дополнительная литература

4. Joseph, Yiu. the Definitive Guide to the ARM Cortex-M3 / Yiu. Joseph. – Burlington : Elsevier Inc, 2007. – 531 с.
5. Trevor, Martin. The Insider’s Guide To The STM32 ARM®Based Microcontroller / Martin. Trevor. – Coventry : Hitex (UK) Ltd., 2008. – 96с.
6. Матюшин А.О Программирование микроконтроллеров. Стратегия и тактика. - 1-е изд. - М.: ДМК Пресс, 2017.
7. Per Arnold Andersen, Karsten Duer, Peter Foldbjerg Daylight, Energy and Indoor Climate Basic Book. - 3-е изд. VELUX Knowledge Centre for Daylight, Energy and Indoor Climate (DEIC) , 2014.
8. Коннолли Т., Бегг К. Database Systems: A Practical Approach to Design, Implementation, and Management. - 3-е изд. - М.: Williams , 2017.
9. Martin Fowler, Pramodkumar J. Sadalage NoSQL Distilled. - 1-е изд. Williams , 2014.

Приложение А. Рисунки

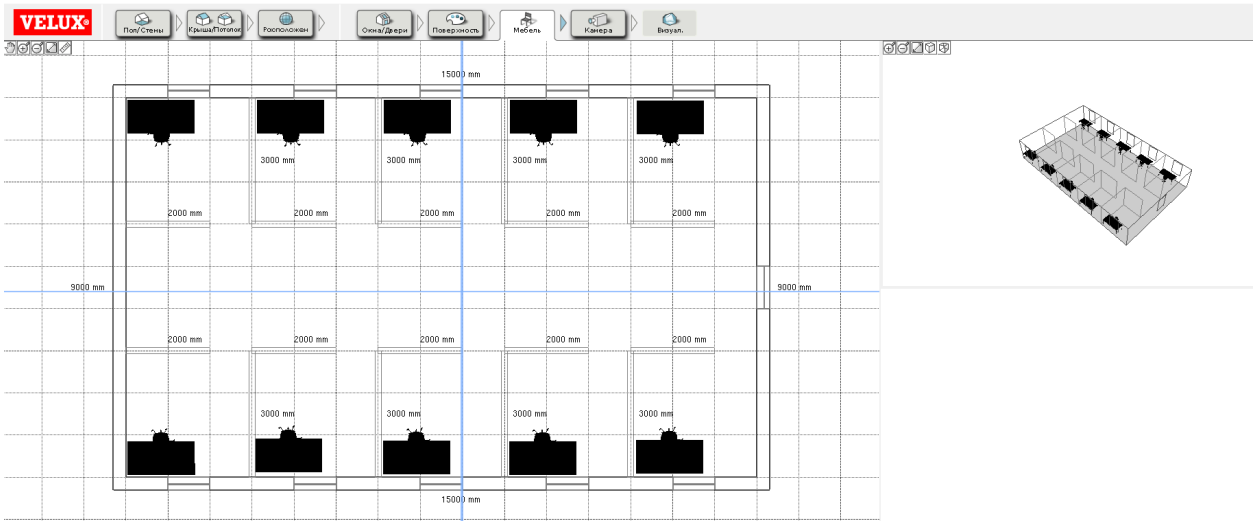

Рисунок А.1.1. Общий план офиса

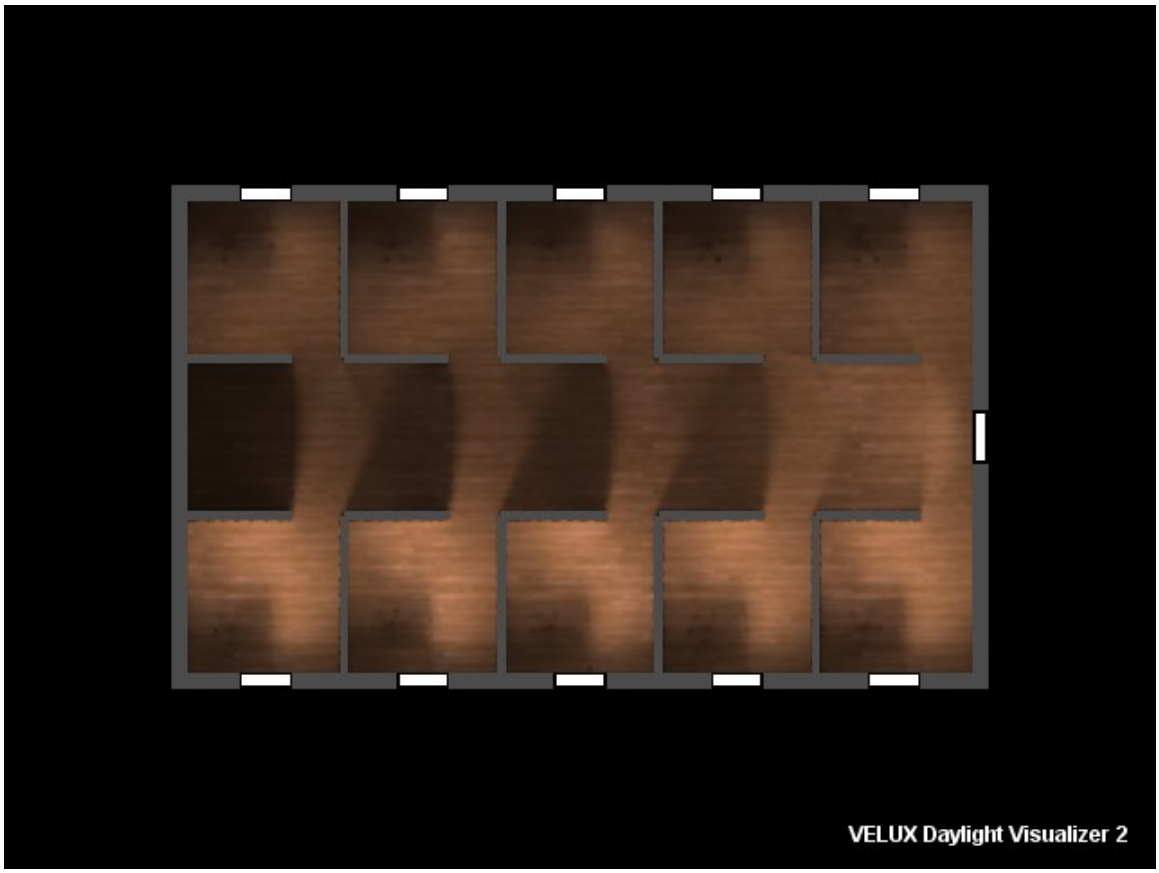

Рисунок А.1.2. Распределение естественного света в офисе

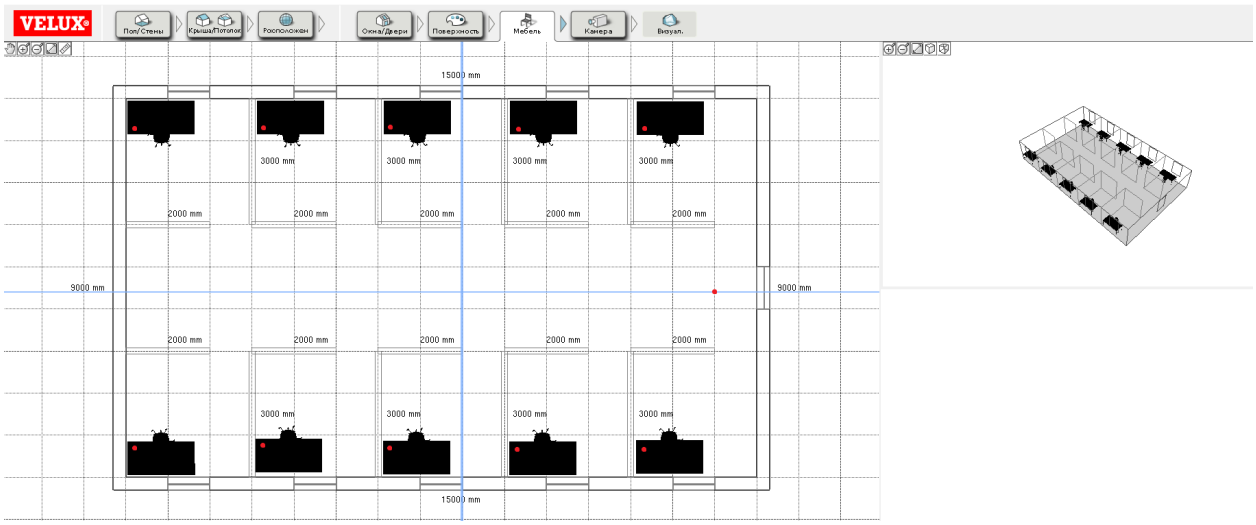

Рисунок А.1.3. Расстановка датчиков освещенности по офису

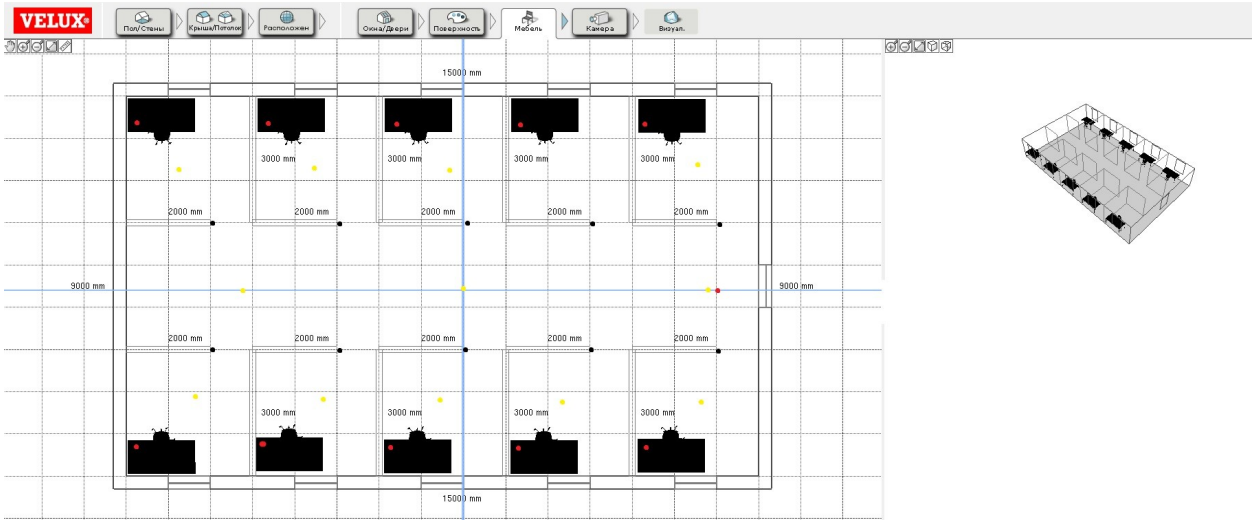

Рисунок А1.4. Финальная расстановка датчиков

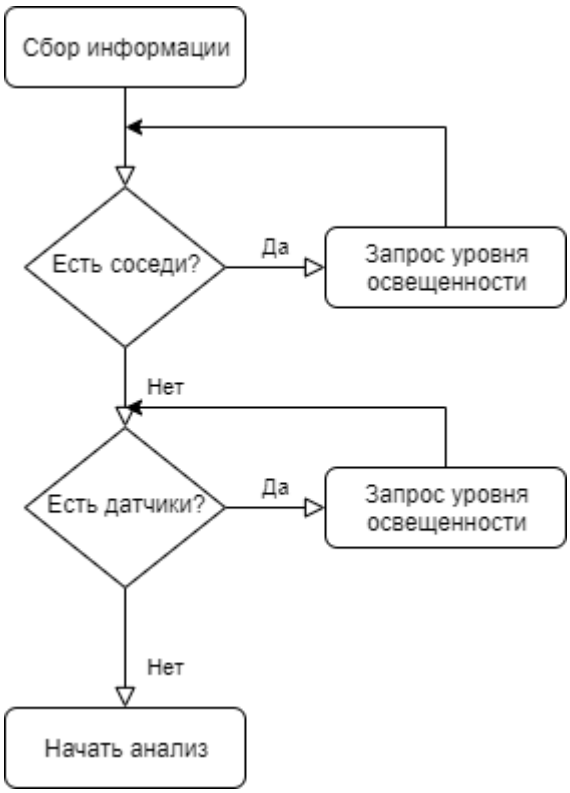

Рисунок А2.1. Алгоритм сбора информации

Рисунок А2.2. Алгоритм анализа информации

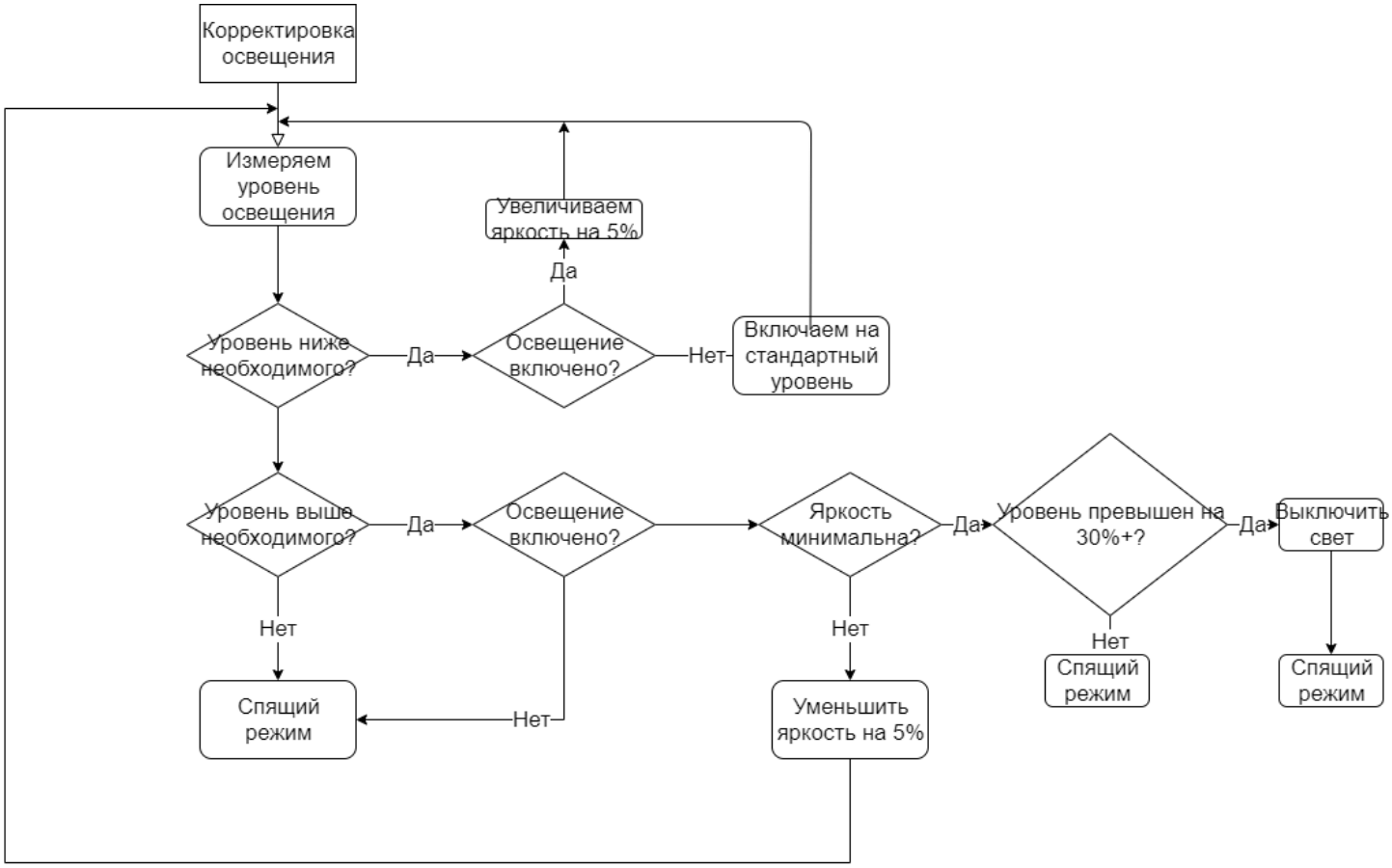

Рисунок А2.3 Алгоритм корректировки освещения

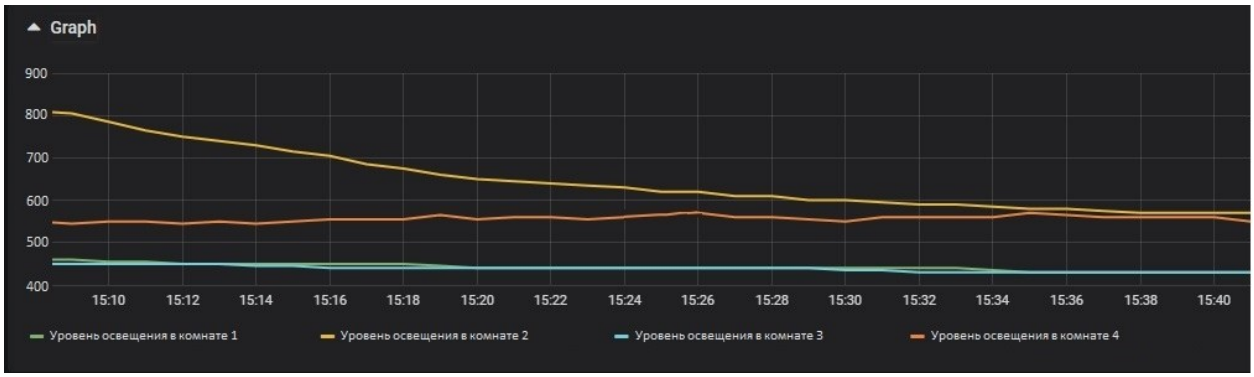

Рисунок А3.1 График в grafana

Рисунок А3.2. Структура сообщений MQTT

Рисунок А3.3. Взаимодействие брокера с другими элементами системы

Приложение Б. Код

Б1. Настройка устройства в качестве mqtt клиента

```
#include "ets_sys.h"
#include "driver/uart.h"
#include "osapi.h"
#include "mqtt.h"
#include "wifi.h"
#include "config.h"
#include "debug.h"
#include "gpio.h"
#include "user_interface.h"
#include "mem.h"

MQTT_Client mqttClient;

void wifiConnectCb(uint8_t status)
{
	if(status == STATION_GOT_IP){
		MQTT_Connect(&mqttClient);
	} else {
		MQTT_Disconnect(&mqttClient);
	}
}
void mqttConnectedCb(uint32_t *args)
{
	MQTT_Client* client = (MQTT_Client*)args;
	INFO("MQTT: Connected\r\n");
	MQTT_Subscribe(client, "/mqtt/topic/0", 0);
```

```
	MQTT_Subscribe(client, "/mqtt/topic/1", 1);
	MQTT_Subscribe(client, "/mqtt/topic/2", 2);

	MQTT_Publish(client, "/mqtt/topic/0", "hello0", 6, 0, 0);
	MQTT_Publish(client, "/mqtt/topic/1", "hello1", 6, 1, 0);
	MQTT_Publish(client, "/mqtt/topic/2", "hello2", 6, 2, 0);

}

void mqttDisconnectedCb(uint32_t *args)
{
	MQTT_Client* client = (MQTT_Client*)args;
	INFO("MQTT: Disconnected\r\n");
}

void mqttPublishedCb(uint32_t *args)
{
	MQTT_Client* client = (MQTT_Client*)args;
	INFO("MQTT: Published\r\n");
}

void mqttDataCb(uint32_t *args, const char* topic, uint32_t topic_len, const
char *data, uint32_t data_len)
{
	char *topicBuf = (char*)os_zalloc(topic_len+1),
			*dataBuf = (char*)os_zalloc(data_len+1);

	MQTT_Client* client = (MQTT_Client*)args;
```

```
    os_memcpy(topicBuf, topic, topic_len);
    topicBuf[topic_len] = 0;

    os_memcpy(dataBuf, data, data_len);
    dataBuf[data_len] = 0;

    INFO("Receive topic: %s, data: %s \r\n", topicBuf, dataBuf);
    os_free(topicBuf);
    os_free(dataBuf);
}


void user_init(void)
{
    uart_init(BIT_RATE_115200, BIT_RATE_115200);
    os_delay_us(1000000);

    CFG_Load();

    MQTT_InitConnection(&mqttClient, sysCfg.mqtt_host, sysCfg.mqtt_port, sysCfg.security);
    //MQTT_InitConnection(&mqttClient, "192.168.11.122", 1880, 0);

    MQTT_InitClient(&mqttClient, sysCfg.device_id, sysCfg.mqtt_user, sysCfg.mqtt_pass, sysCfg.mqtt_keepalive, 1);
    //MQTT_InitClient(&mqttClient, "client_id", "user", "pass", 120, 1);

    MQTT_InitLWT(&mqttClient, "/lwt", "offline", 0, 0);
    MQTT_OnConnected(&mqttClient, mqttConnectedCb);
```

```
    MQTT_OnDisconnected(&mqttClient, mqttDisconnectedCb);
    MQTT_OnPublished(&mqttClient, mqttPublishedCb);
    MQTT_OnData(&mqttClient, mqttDataCb);

    WIFI_Connect(sysCfg.sta_ssid, sysCfg.sta_pwd, wifiConnectCb);

    INFO("\r\nSystem started ...\r\n");
}
```

Б2. «Мост» между mosquitto и influxDB

```
import re
from typing import NamedTuple

import paho.mqtt.client as mqtt
from influxdb import InfluxDBClient

INFLUXDB_ADDRESS = '192.168.0.8'
INFLUXDB_USER = 'mqtt'
INFLUXDB_PASSWORD = 'mqtt'
INFLUXDB_DATABASE = 'DB'

MQTT_ADDRESS = '192.168.0.8'
MQTT_USER = 'cdavid'
MQTT_PASSWORD = 'cdavid'
MQTT_TOPIC = 'home/+/+'
MQTT_REGEX = 'home/([^/]+)/([^/]+)'
MQTT_CLIENT_ID = 'MQTTInfluxDBBridge'

influxdb_client = InfluxDBClient(INFLUXDB_ADDRESS, 8086, INFLUXDB_USER, INFLUXDB_PASSWORD, None)



class SensorData(NamedTuple):
    location: str
    measurement: str
    value: float
```

```
def on_connect(client, userdata, flags, rc):
    print('Connected with result code ' + str(rc))
    client.subscribe(MQTT_TOPIC)


def on_message(client, userdata, msg):
    print(msg.topic + ' ' + str(msg.payload))
                        sensor_data      =      _parse_mqtt_message(msg.topic,
msg.payload.decode('utf-8'))
    if sensor_data is not None:
        _send_sensor_data_to_influxdb(sensor_data)


def _parse_mqtt_message(topic, payload):
    match = re.match(MQTT_REGEX, topic)
    if match:
        location = match.group(1)
        measurement = match.group(2)
        if measurement == 'status':
            return None
        return SensorData(location, measurement, float(payload))
    else:
        return None


def _send_sensor_data_to_influxdb(sensor_data):
    json_body = [
        {
```

```
            'measurement': sensor_data.measurement,
            'tags': {
                'location': sensor_data.location
            },
            'fields': {
                'value': sensor_data.value
            }
        }
    ]
    influxdb_client.write_points(json_body)


def _init_influxdb_database():
    databases = influxdb_client.get_list_database()
        if len(list(filter(lambda x: x['name'] == INFLUXDB_DATABASE,
databases))) == 0:
        influxdb_client.create_database(INFLUXDB_DATABASE)
    influxdb_client.switch_database(INFLUXDB_DATABASE)


def main():
    _init_influxdb_database()

    mqtt_client = mqtt.Client(MQTT_CLIENT_ID)
    mqtt_client.username_pw_set(MQTT_USER, MQTT_PASSWORD)
    mqtt_client.on_connect = on_connect
    mqtt_client.on_message = on_message

    mqtt_client.connect(MQTT_ADDRESS, 1883)
```

```
    mqtt_client.loop_forever()
main()
```

Приложение В. Ссылки

[1]. Networked Lighting Controls Will Surpass $5.3 Billion in Annual Revenue by 2020. [Online]. Available: http://www.navigantresearch.com/newsroom/networked-lighting-controls-will-surpass-5-3-billion-in-annual-revenue-by-2020, accessed Sep. 2013.