Versions

▶ v0.1.0 — private beta-version.
▶ v0.2.0 — development.
▶ v1.0.0 — release.

- New core architecture.
- New library.
- The documentation has been completed.

Содержание

1. Введение

1.1 Микроэлектроника вокруг нас

Мы живем в эпоху кремниевой лихорадки, когда на каждом шагу нас окружает огромное количество высокотехнологичных устройств с микрочипами внутри. Даже самые обыденные бытовые приборы, такие как электрочайник, светодиодная лампа или обогреватель, оказываются оборудованы умными «мозгами», превращая наши дома в настоящие технологические чудеса. Что уж там говорить про различные смартфоны, компьютеры, серверы… Такая массовая распространенность микроэлетроники привела к росту уровня технологий и увеличению масштабов производства. Появилось много микросхем, доступных обычным радиолюбителям. Сейчас, всего за цену одной порции шаурмы, можно заказать из Поднебесной модуль, способный решать задачи с эффективностью компьютера на i486 (процессор Intel 80486, 1989г.), при этом сам он поместится в чайной ложке!

Системой умного дома сейчас мало кого можно удивить. На рынке представлены промышленные решения от именитых брендов. Компании готовы спроектировать и построить дом под любые запросы, либо предоставляют линейку модулей, соединяя которые можно собрать систему под свои потребности. Для таких продвинутых решений не нужны знания программирования и инженерии, все уже продумано разработчиками, требуется только подключить модули согласно инструкции и настроить их взаимодействие в удобном приложении. Линейки модулей достаточно обширные и позволяют контролировать температуру, освещение, включать и выключать приборы по расписанию, общаться с помощью чат-ботов и голосовых помощников и многое другое.

1.2 Актуальность и обоснование проекта

Глядя на это разнообразие, может показаться, что создавать еще одну подобную систему совершенно бессмысленно. Однако, не все так однозначно. Посмотрим, какие же имеются плюсы для тех, кто намерен заняться самостоятельной разработкой?

- Цена. Готовые модули стоят достаточно дорого и собрать на них полноценную систему выйдет далеко не дешево.
- Функциональность. Индивидуальная разработка позволяет реализовать все полностью под себя, вывести доступ к любым настройкам и иметь возможность в любой момент исправить или модернизировать систему.
- Опыт. Во время разработки потребуется изучение программирования на низком и высоком уровне, знание радиоэлектроники, создание электронных схем и плат, пайка, 3D-моделирование и печать (в данном проекте не используется, но ничто не мешает добавить) и пр.
- Возможность наладить коммерческое производство.
- Иметь в портфолио достаточно объемный проект.
- В системе собраны алгоритмы, применение которым можно найти в других разнообразных проектах.

Да, такие мысли приходили в голову многим и аналогичных проектов с подробными статьями множество на просторах интернета. Все они достаточно разнообразны и написаны, в основном, под конкретную ситуацию. Этот проект не является исключением — на выходе получается конкретная система умного дома, но, основу проекта составляет библиотека, которая позволяет реализовать систему “под любой дом”. Поэтому правильнее будет сказать, что ***суть проекта заключается в разработке единого метода реализации системы умного дома, а действительно существующая система — это всего лишь пример применения разработки на практике.***

1.3 Цели и задачи

Главная цель проекта — разработать набор решений и инструментов для создания систем умного дома.

1. Создание инструментов и решений для разработки системы умного дома под любой функционал и условия.
2. Создание рабочего прототипа.
3. Создание демонстрационного макета.

2. Обзор литературы

2.1 Alex Gyver

Во время разработки проекта самыми главными стали материалы, предоставленные в открытом доступе известным в кругах самодельщиков и ардуинщиков блогером AlexGyver. На его сайте есть немало полезных уроков, гайдов и статей по микроконтроллерам. В ***Smart Home System*** многие нюансы не представлены подробно, однако почти все они в деталях грамотно изложены в уроках Gyver'а. Вместе со своим помощником(ами) Alex создал базу легких, оптимизированных и качественно сделанных библиотек, которые используются в ***Smart Home System***: GyverLibs.

Все материалы и разработки Гайвера можно смело рекомендовать для изучения и использования. Большое ему спасибо за огромный и безвозмездный вклад в сообщество программистов-радиолюбителей.

2.2 Виктор Петин

Петин В. А. "Создание умного дома на базе Arduino" — в книге подробно рассматривается конструирование умного дома на Arduino Mega (atmega) и NodeMCU (esp8266). Обстоятельно описаны все этапы: от установки ПО, до подключения датчиков, управления нагрузкой и интернета вещей. Даны примеры подключения практически всех существующих типов датчиков.

2.3 Нил Кэмерон

Neil Cameron. Electronics Projects with the ESP8266 and ESP32: Building WebPages, Applications, and WiFi Enabled — в книге приведен ряд полезных примеров и интересных проектов. Есть переводной вариант от российских издательств.

3. Структура проекта

Проект выложен в репозитории на GitHub и доступен по ссылке: https://github.com/MrRyabena/SmartHomeSystem.

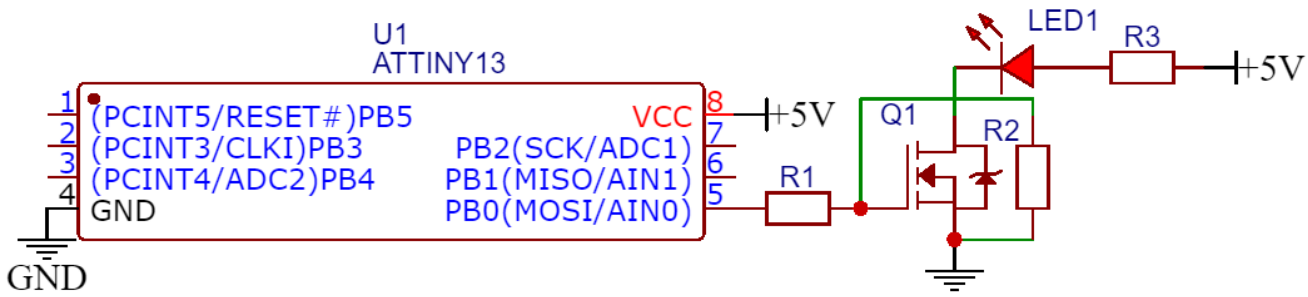
- doc — документация к проекту.
- libraries — сторонние библиотеки, используемые в проекте.
- pitch — презентации проекта и вспомогательные файлы.
- schemes — картинки, схемы, чертежи...
- SHSlogos — логотипы и символика проекта.
- src — ПО проекта.
    - debugging_sketches — наброски отладочных прошивок.

- demo_version — демонстрационная версия проекта.
  - app — приложение под Windows.
  - SmartModules — прошивки модулей.
- SHSapp — приложение под windows.
- SHScore — ядро (бизнес-логика), основная разработка.
- SHSlibrary — библиотека с удобными инструментами, основанная на ядре.
- SmartModules — [beta] устройства и модули.
- SmartModulesAPI — [beta] команды устройств и модулей.
- synchronizer — быстро перекидывает файлы из SHSlibrary в папку с библиотеками (для удобства разработки).

4. Методы и этапы проектирования

4.1 Принцип работы

Все много раз слышали, что работа электроники основана на принципе нулей и единиц (Булева алгебра), где сами значения “0” и “1” — на первый взгляд, очень абстрактные понятия “нет сигнала” и “есть сигнал”. Чтобы понять как, основываясь на этом принципе, построить умный дом, надо копнуть чуть поглубже. Рассмотрим упрощенную схему какого-нибудь модуля.



Первое что нужно — это подать на схему питание. Линий питания две:

- **COM (GND, VSS, земля)** — общий вывод питания, относительно него измеряются все остальные потенциалы в схеме.
- **+V (VCC, VIN)** — положительная линия питания, их может быть несколько, рассчитанных на разные напряжения, например, +12V, +5V, +3.3V. Еще бывает отрицательное напряжение (относительно GND, опять же), но с ним обычно работают усилители и компараторы, в большинстве архитектур ЭВМ для логических цепей оно не применяется.

Вторая часть схемы, которая будет управлять модулем — это набор логических элементов. Наш дом — умный, поэтому схема из таких элементов будет непростая. Собрать самостоятельно ее, используя лишь базовые компоненты (резисторы, транзисторы, диоды…) крайне сложно, объемно и не рационально, да к тому же работать стабильно она вряд ли будет. К счастью, проблему уже давно решили и все необходимые цепи собраны в одном небольшом радиокомпоненте — микросхеме.

Интегральная схема (микросхема) — это электронная схема, помещенная на полупроводниковой (чаще всего кремниевой) подложке, с помощью фотолитографии. Небольшой корпус может содержать

внутри как несложный набор логических элементов, так и целый процессор или контроллер, последний нам и нужен.

Микроконтроллер — это микросхема, которая содержит в себе процессор, ОЗУ, ПЗУ и периферийные устройства. Это целый небольшой компьютер, который может выполнять математические операции и управлять другими устройствами с помощью периферии.

Чтобы микроконтроллер мог принимать и выводить какие-то сигналы он оснащен выводами (контактами, пинами) с интерфейсом GPIO (general-purpose input/output). Такие пины могут работать в двух режимах: `INPUT` (вход) и `OUTPUT` (выход).

- В режиме `INPUT` микроконтроллер сравнивает входящий сигнал с землей (GND) и принимает его за 1, если его потенциал больше GND.
- Аналогично в режиме `OUTPUT` микроконтроллер может формировать на определенном контакте "0" или потенциал, равный его напряжению питания.

Теперь логический сигнал от микроконтроллера надо усилить с помощью силовой части схемы (например, транзистора или реле), что позволит управлять нагрузкой (светом, отоплением, чайником...) в режиме on/off (вкл/выкл).

Для того, чтобы управлять мощностью (интенсивностью работы) нагрузки (т.е. яркостью света, температурой нагревателя), необходимо регулировать подаваемое на нее напряжение. В цифровой электронике для этого применяется PWM (ШИМ — широтно-импульсная модуляция). Проще говоря, микроконтроллер очень быстро включает и выключает нагрузку на разные микропромежутки времени, а за счет ее инертности получается плавное регулирование.

Выше был описан принцип работы цифровой электроники, но в арсенале многих микроконтроллеров есть блоки для работы с аналоговой электроникой — ADC (АЦП) и DAC (ЦАП).

- **Аналого-цифровой преобразователь** позволяет микроконтроллеру измерять потенциал входного сигнала в диапазоне от 0, до опорного напряжения (либо задается от отдельного источника, либо совпадает с напряжением питания микроконтроллера) с некоторой точностью, которая зависит от разрядности АЦП. Он используется для считывания информации с датчиков, которые за счет физических эффектов (фотоэффектов, термоэффектов, эффекта Холла и пр.) изменяют напряжение на своем выходе. (Прим. цифровые датчики имеют встроенный АЦП и микроконтроллер для передачи информации по интерфейсам связи).

- **Цифро-аналоговый преобразователь** позволяет изменять потенциал сигнала в некотором диапазоне, он обычно служит для звуковых сигналов либо в качестве "цифровых потенциометров".

Теперь, вдохновившись идеей и понимая принцип работы, можно попробовать создать свою систему. На одной теории дом не построишь, поэтому разработка требует постоянных экспериментов, которые подробно описаны в основной части документации.

4.2 Этапы разработки

Первым делом необходимо создать физические устройства для решения задач: управления светом, температурой, измерением параметров погоды и пр. Затем реализуются методы взаимодействия и управления. В итоге – получаем автоматизированную систему. Последний шаг в проектировании –

"научить" систему обрабатывать данные и регулировать все устройства корректно в любой ситуации с минимальным вмешательством пользователя.

- Этап I. Проектирование автоматических модулей
    - Определение устройств, процессов и параметров, которые необходимо контролировать.
    - Проектирование электронной части модулей.
    - Определение общей структуры всей системы.
- Этап II. Проектирование ядра (бизнес-логики).
    - Дискретизация задач.
    - Создание шаблонов процессов, задач и конфигураций.
    - Создание шаблонов обработки данных и объектов.
    - Создание протоколов взаимодействия и API.
- Этап III. Разработка библиотеки.
    - Создание удобных инструментов, на основе ядра.
    - Создание удобного интерфейса для программистов.
    - Создание инструментов для автоматической конфигурации системы.
- Этап IV. Объединение модулей в единую систему.
    - Настройка стабильной связи между модулями.
    - Настройка протоколов передачи данных.
    - Проектирование API модулей.
    - Проектирование моделей взаимодействия между модулями.
    - Настройка систем обработки ошибок.
- Этап V. Взаимодействие с пользователем.
    - Определение методов взаимодействия с пользователем.
    - Реализация удобных для пользователя методов управления системой.
    - Разработка GUI.
- Этап VI. Обучение системы саморегулированию.
    - Определение факторов, влияющих на поведение системы.
    - Проектирование сценариев поведения системы.
    - Создание нейронной сети, для автоматического регулирования системы.
- Этап VII. Настройка системы.
    - Отладка всех датчиков и модулей.
    - Оптимизация прошивок.
    - Обучение нейронной сети.
    - Тестирование системы на стабильность.
    - Проверка обработки системой критических ситуаций.

4.3 Подход к проектированию

При проектировании выявлено два способа реализации ***Smart Home System***:

1. Создается один большой модуль, который включает в себя несколько микроконтроллеров, их обвязку, систему питания и подключения устройств. Он реализует все необходимые функции.
2. Создается несколько небольших модулей, каждый из которых контролирует один небольшой блок устройств и процессов, имеет собственную систему питания. Все модули связываются между собой по WiFi и образуют единую систему.

Преимущества первого подхода заключаются в удобстве обслуживания: все находится в одном корпусе, не нужно бегать по всему дому, чтобы что-то подключить или поправить. На этом, как оказалось, плюсы заканчиваются, начинаются проблемы. При создании такого модуля получается большая печатная плата, очень большая и сложная, возникает много трудностей при пайке и выявлении ошибок. Самая большая проблема — огромное количество проводов, которые необходимо протянуть по всему дому. Они постоянно отовсюду вылезают, мешают, стоят дорого и наводят помехи друг на друга.

Второй способ оказался более практичным. Мы можем постепенно создавать небольшие схемы, добавлять, менять или переделывать их. Сигнальные и силовые линии не нужно тянуть по всему дому, все получается аккуратно и компактно.
Первый способ имеет смысл только для реализации каких-то небольших систем, по типу контроллера теплицы или какого-то небольшого помещения. Поэтому Smart Home System основан на втором способе.

4.4 Функции

*Выше уже описаны цели и идеи их реализации, здесь речь пойдет более конкретно о задачах, которые можно решить с помощью **Smart Home System**.*

- Первое что приходит в голову, когда речь идет об умном доме — автоматическое включение и регулировка освещения. Свет должен включаться, когда это необходимо, выключаться, но только тогда, когда он действительно никому не нужен и плавно регулироваться, постоянно поддерживая одинаковое значение яркости. Еще одна дополнительная функция точно не оставит равнодушными людей, которым приходится рано вставать — будильник-рассвет — комната плавно заливается теплым светом, эмитируя восход солнца.
  Задача достаточно простая и реализуется даже без дополнительных библиотек.

- • Если уж речь зашла об освещении, то можно добавить красоты и технологичности — RGB-подсветку. Тут необходимы инструменты для работы со светодиодами типа **RGB** (обычные четырехвыводные, где требуется только регулировать яркость каждого канала цвета) и **ARGB** (адресные светодиоды, которые позволяют управлять цветом каждого элемента ленты, независимо от всех остальных). Если первыми можно управлять «вручную», то для адресных точно потребуются дополнительные библиотеки. Когда есть хорошие решения, писать заново библиотеки смысла особого нет, поэтому используя готовую основу в виде пары библиотек останется только создать различные эффекты и режимы работы.

- Климат-контроль. С помощью современных датчиков можно с легкостью отслеживать главные факторы домашнего микроклимата: температуру, уровень влажности и концентрацию углекислого газа. Обрабатывая их, можно настроить регулировку котла или подачу горячей воды, работу увлажнителей воздуха и открывать форточки (или включать систему вентиляции).

- Метеостанция. Измерение температуры, влажности, скорости и направления ветра, атмосферного давления. Обработка полученных результатов и составление прогноза погоды на ближайшее время, рекомендации по одежде и ожидания на день.

- Контроллер теплицы. В доме наверняка есть комнатные растения, а может быть целая теплица или сад. Необходимо реализовать автополив, контроль влажности почвы и досвечивание растений.

- IoT. Реализовать возможность получать данные из интернета, выкладывать их, обмениваться с пользователями различных чат-ботов и другие паттерны Интернета Вещей.
- Smart Bar. Холодильная камера на элементах Пельтье, для охлаждения напитков, чайник с поддержанием температуры воды, автоналиватор напитков.
- Аудиосистема. Качественное проигрывание звука, переключение между колонками, автоматическое воспроизведение, эффекты полного погружения.

4.5 Структура Smart Home System

Module — самостоятельная часть Smart Home System, отвечающая за один или несколько процессов (управление устройством, опрос датчиков, работа с интернетом...). В системе может быть бесконечно много модулей, которые реализуют необходимый функционал:

- SmartChandelier — управляет люстрой.
- SmartLighter — управляет настенными светильниками.
- SmartRGB — управляет эффектами RGB-подсветки.
- SmartBar — холодильник на элементах Пельтье для охлаждения напитков.
- SmartTeapot — подогреет чай к нужному времени и будет поддерживать его температуру.
- SmartGarden — обеспечит автополив и досвечивание растений.
- SmartServer — связывает все модули, обрабатывает данные.
  - SmartHomeBot — реализован в сервере, отвечает за Telegram-бота и общение с пользователем.
  - SmartVoiceControl — система голосового управления.
- SmartMedia — акустическая система, домашний кинотеатр, подсветка экранов (Ambilight).

5. Выбор компонентов

5.1 Микроконтроллеры

Для управления всеми модулями от микроконтроллеров требуется наличие достаточного количества выводов `GPIO`, поддержка (желательно аппаратная) интерфейсов `UART`, `I2C`, `SPI`, `PWM`, `WiFi`, и встроенный `ADC`.

Сейчас на рынке представлено много линеек микроконтроллеров, доступных простым радиолюбителям:

- Компания `Atmel` представляет семейства на архитектуре AVR — `Attiny` и `Atmega`. Первые микроконтроллеры совсем крохотные и использовать их можно только для небольших задач, например, управления одним светильником и связи с остальными. Семейство Atmega более известно под маркой Arduino, т.к. лежит в основе большинства их плат. Эти камни уже могут похвастаться большим арсеналом GPIO, ADC, высоким выходным током с пина. На их основе можно собирать полноценные модули, не хватает только WiFi или другой беспроводной связи.
- Китайцы, в лице `Espressif Systems` разработали свою линейку микроконтроллеров с WiFi на борту, flash-памятью и достаточно мощными (для задач умного дома) вычислительными ядрами.

- В сети много гайдов по созданию сервера умного дома на основе `Raspberry Pi`. Это семейство представляет собой целые миниатюрные компьютеры на архитектуре ARM, позволяющие работать с операционной системой Linux и выводить изображение на монитор. Тема на самом деле достаточно крутая и найти им применение не составит проблем — можно собрать домашний сервер и это выйдет компактнее, чем переделывать какой-нибудь старый компьютер.

  Мощности Малинки для Smart Home System будут избыточные, да и стоят такие платы достаточно дорого.

В итоге выбор пал на 3 основных микроконтроллера: Atmega328P, ESP8266, ESP32.

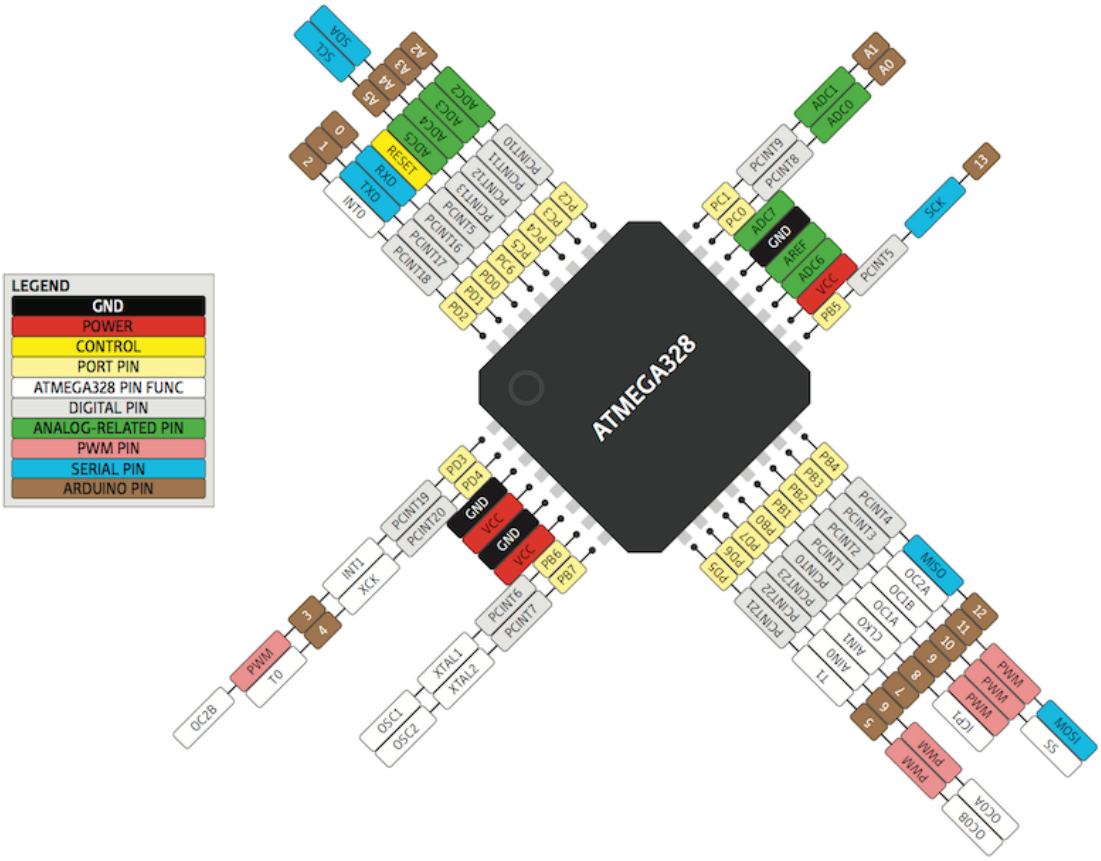
5.1.1 ATmega328P

Микроконтроллер удобен в использовании, крайне неприхотлив в эксплуатации, имеет много аналоговых пинов, аппаратную поддержку ШИМ. WiFi на борту нет, поэтому его придется связывать с ESP по UART.

В проекте используется “голый” микроконтроллер (особенно там, где нужна компактность), либо платы Arduino Nano, Arduino Pro Mini.

Parametr	**Value**
CPU type	8-bit AVR
CPU speed	16 MHz
Current consumption	<<100 mA
Flash memory	32 KB
SRAM	2 KB
EEPROM	1 KB
I/O pins	23
Max current from the pin	40 mA
PWM pins	6
ADC pins	8 channels, 10-bit
DAC	none
External interrupts	3
I2C	1
I2S	none
SPI	1
UART	1
WiFi	none
Bluetooth	none

Parametr	Value
Operating Voltage Range	1.8 — 5.5 V



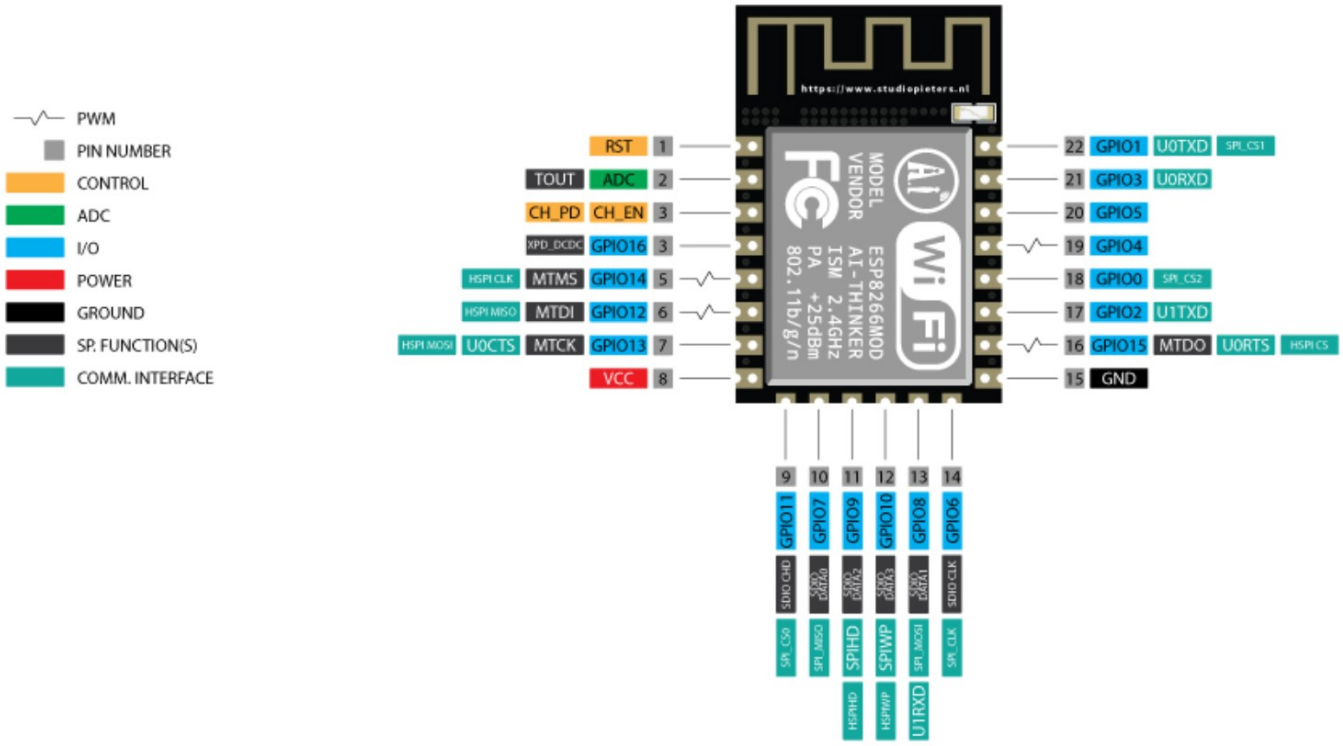
5.1.2 ESP8266

Имеет WiFi, достаточно много оперативной и постоянной памяти – шустрый процессор. Хорошо подходит для взаимодействия с другими цифровыми устройствами и интернетом. Из минусов: слабый ток с пина (12 mA), рабочее напряжение 3.3 V, нет аппаратной поддержки PWM (реализована программно) и всего 1 канал АЦП.

“Голый” микроконтроллер — ESP12F.
Платы — NodeMCU или WemosD1 mini.

Parametr	Value
CPU type	Xtensa L106, 32 bit
CPU speed	80/160 MHz
Current consumption	300 mA
Flash memory	1-16 MB
SRAM	82 KB
EEPROM	4 KB

Parametr	Value
I/O pins	11
Max current from the pin	12 mA
PWM pins	10 (**software!**)
ADC pins	1
DAC	none
External interrupts	10
I2C	1 (software)
I2S	1
SPI	1
UART	1.5
WiFi	802.11 b/g/n 2,4 GHz
Bluetooth	none
Operating Voltage Range	2.2—3.6V

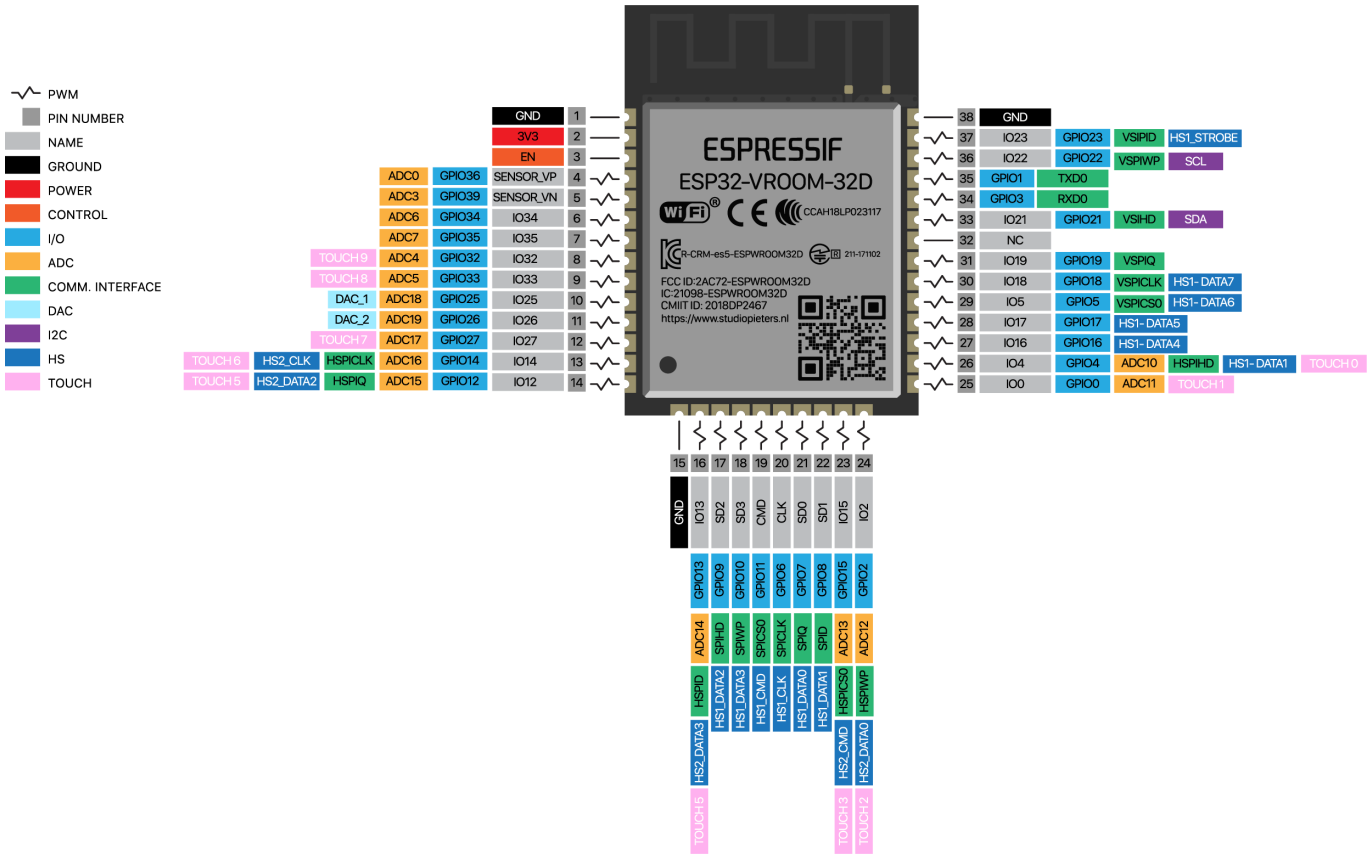


5.1.3 ESP32

Имеет двухъядерный процессор, большой объем памяти, WiFi, PWM, аппаратную поддержку цифровых протоколов, сенсорные пины, много каналов ADC, DAC и выход звукового сигнала. Мощный чип, на

котором будет собран сервер. В проекте используется “голый” микроконтроллер, но также можно взять платы (аналогично ESP8266).

Parametr	**Value**
CPU type	Xtensa LX6, dual-core, 32 bit
CPU speed	160/240 MHz
Current consumption	300 mA
Flash memory	1-16 MB
SRAM	512 KB
EEPROM	4 KB
I/O pins	34
Max current from the pin	12 mA
PWM pins	16
ADC pins	18, 12-bit
DAC	2, 8-bit
External interrupts	34 (10 touch sensors)
I2C	2
I2S	2
SPI	4
UART	3
WiFi	802.11 b/g/n 2,4 GHz
Bluetooth	v4.2 BR/EDR and BLE
Operating Voltage Range	2.2—3.6 V



5.2 Датчики

Чтобы ***Smart Home System*** могла получать информацию об окружающей среде, в ней предусмотрены датчики, позволяющие измерять разнообразные параметры. Датчики можно разделить на две категории: цифровые и аналоговые, о чем было сказано выше. Цифровые в основном подключаются по шинам I2C и SPI, аналоговые — к АЦП через обвязку (если требуется).

5.2.1 Temperature

- **Терморезистор (термистор)** — самый простой способ измерить температуру окружающей среды с приемлемой точностью. Изменяет свое сопротивление с изменением температуры. Подключается в схеме делителя напряжения к ADC и обрабатывается по уравнению Стейнхарта-Харта.
- **DS18B20** — цифровой, достаточно точный датчик температуры, подключается по интерфейсу I2C.
- **Термопара + MAX6675** — термопара и драйвер для измерения ее показаний, измерение температуры в несколько сотен градусов.

5.2.2 Humidity

- **DHT22** — цифровой датчик температуры и влажности воздуха. Не особо точный, в системе не используется.
- **HTU21D** — гораздо более точный датчик температуры и влажности, рекомендуется использовать его.

5.2.3 Pressure

- **BME280/BMP280** — цифровой датчик измерения атмосферного давления. По сути, это целая метеостанция, т.к., помимо давления, измеряет температуру и влажность воздуха (для последнего нужно брать BME280).

5.2.4 Carbon dioxide

- **MH-Z19b** — датчик углекислого газа.

5.2.5 Illumination

- **Фоторезистор** — аналоговый компонент, изменяет свое сопротивление, в зависимости от интенсивности падающего на него света. Подключается к ADC в схеме делителя напряжения.

5.2.6 Other

- **Кнопка/энкодер** — можно организовать удобное управление системой.
- **TTP223** — сенсорная кнопка. Подключается и работает точно так же, как обычная механическая. Можно подпаяться к плате и вывести “антенну” в удобное место. Пробивает толщину в ~0.5mm.
- **Потенциометр** — подключается к ADC, через него тоже можно регулировать настройки системы.

5.3 Силовые компоненты

По управлению мощной нагрузкой есть хорошая статья, там написано грамотно и подробно, ниже лишь указаны некоторые детали.

5.3.1 Транзисторы

- **IRLB8743** — полевой транзистор (мосфет) для вкл./выкл. или ШИМ управления нагрузкой в несколько ампер до 30 V. Может быть не самый лучший вариант (высоковатое сопротивление в открытом состоянии, сильно греется при большом токе или высокой частоте), но дешевый и практичный. Заказать лучше всего сразу пачку-две в Поднебесной.
- **2N7000** — небольшой полевой транзистор для слаботочных цепей (тянет катушку небольшого реле или клапана, кусочки светодиодных лент).
- **BC547** — небольшой биполярный транзистор, полезен для подтяжки цепей к земле, управления слаботочными нагрузками.

5.3.2 Симисторы

ВНИМАНИЕ! Работа с высоким напряжением опасна для жизни! Следует строго соблюдать все меры предосторожности и убеждаться, что цепи отключены от питания, а все конденсаторы разряжены!

В Smart Home System основная часть нагрузок и модулей работает от низковольтного напряжения 3.3V—24V. Не рекомендуется внедрять в систему высоковольтные цепи, однако, если есть необходимость, в системе разработаны решения для управления нагрузкой с помощью реле или симистора. **Все низковольтные цепи необходимо гальванически развязать с высоковольтными (с помощью трансформаторов или оптронов).**

- **BTA208X-1000C** — хорошо подойдет для регулирования осветительных приборов.
- **BTA41-600BRG** — очень мощный симистор, когда-то можно было купить в 8 раз дешевле.

5.4 Обвязка

5.4.1 Резисторы

- **150Om 0.25W** — для ограничения тока с пинов микроконтроллеров с логическим уровнем 5V (Atmega328P, ток с пина ~40mA).
- **330Om 0.25W** — для ограничения тока с пинов микроконтроллеров с логическим уровнем 3.3V (ESP8266, ESP32, ток с пина ~10mA).
- **10kOm 0.25W** — для логической подтяжки пинов.

5.4.2 Конденсаторы

- **0.1uF** — керамический конденсатор, стоит ставить как можно ближе к микроконтроллерам, для поглощения помех и пульсаций питания.
- **470—2000uF** — стоит ставить рядом с микроконтроллерами и другими чувствительными устройствами (адресными светодиодными лентами, датчиками...), особенно если используется некачественный блок питания или в этой же цепи есть мощная нагрузка.

5.4.3 Диоды

- **1N4007** — практически на все случаи жизни.
- **1N5817/1N5819** — диоды Шоттки, могу пригодиться, когда мешает падение напряжения на классическом диоде (например, для реализации защиты схемы от неправильной полярности входного питания, без потерь напряжения, что особенно критично для 3.3V).

5.4.4 Стабилитроны

- **1N4728** — 3.3V, подойдет для выравнивания логических уровней.
- **BZX55C5V1** — 5V, можно сделать источник опорного напряжения.

5.4.5 Стабилизаторы напряжения

- **LD1117/L78L33** — 3.3V 1.5/0.1A.
- **L7805/L78L05** — 5V 1.5A/0.1A.
- **L7812C** — 12V.
- **LM317** — регулируемый выход ~1—24V.

5.4.6 Оптопары (оптроны)

- **АОТ101ГС** — транзисторная, двухканальная. Удобна для гальванической развязки цепей связи типа UART.
- **PC814/FOD814** — транзисторная, хорошо подходит для цепей постоянного тока.

- **PC817/EL817C** — симисторная, хорошо подходит для цепей переменного тока.
- **MOC3063** — симисторная, с детектором нуля (для управления нагрузкой в режиме on/off).
- **MOC3021** — симисторная, без детектора нуля (для диммера).

5.4.7 Регистры

- **SN74HC595** — восьмиканальный сдвиговый регистр, можно использовать как расширитель пинов.
- **CD4051** — аналоговый восьмиканальный мультиплексор, можно использовать как переключатель аналоговых входов/выходов (особенно актуально для ESP8266).

6. Этап I. Проектирование автоматических модулей

В этом этапе рассказаны принципы построения системы умного дома. В ***Smart Home System*** алгоритмы управления уже реализованы в ядре и библиотеке (см. Этап II и Этап III), что упрощает и ускоряет разработку системы.

6.1 Module

Module — самостоятельный компонент Smart Home System, состоящий из одного или нескольких микроконтроллеров, блока питания, обвязки для управления нагрузками и датчиками для сбора данных.

- Может управлять устройствами и нагрузками.
- Может опрашивать датчики и обрабатывать информацию с них.
- Должен корректно функционировать самостоятельно, при потере связи с другими модулями.
- Должен иметь связь с сервером.

Каждый модуль имеет свой `ID`, по которому к нему можно обратиться, понять, что данные собраны и отправлены им.

Для Module созданы отдельные классы (см. SHSmodule.h), которые являются удобной оберткой (по сути мини операционной системой), контролирующей модуль и позволяющей по одному шаблону настроить и запустить все процессы, корректно связаться и взаимодействовать с системой. Это позволило вынести одинаковые куски кода по настройке каждого модуля в пару удобных функций.

6.2 Server

За обработку всех данных, запросов, команд и принятие решений отвечает главный элемент системы — сервер. По идее, в качестве сервера может выступать любое устройство, например, конкретный модуль или компьютер. В ***Smart Home System*** сервер реализован на отдельном микроконтроллере, т.к. компьютер нет смысла гонять впустую (не выключая) из-за какого-то процесса умного дома, а другие модули заняты своими вычислениями и их мощности будет маловато.

6.2.1 Основные требования к серверу

- Поддерживать одновременное подключение нескольких устройств по TCP/IP.

- Иметь достаточную вычислительную мощность для быстрой обработки всех данных.
- Иметь хранилище данных (ПЗУ) для сохранения собранной информации и ведения статистики.
- Эффективно и надежно поддерживать непрерывную работу.

Выбранным микроконтроллером стал ESP32, его двухъядерный процессор и достаточно большой объем RAM идеально подходят для работы с TCP/IP и обработкой данных.

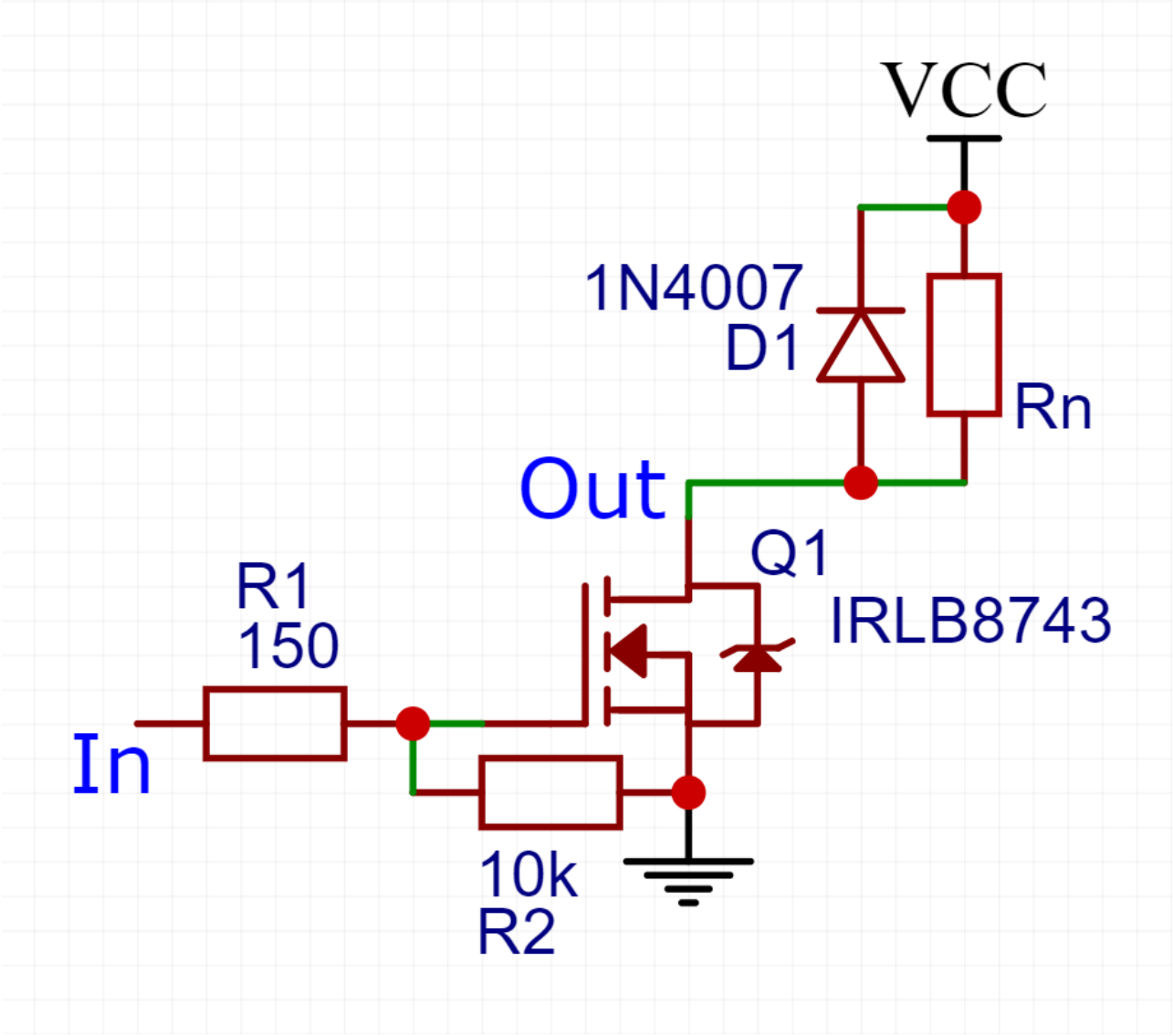
6.3 Управление нагрузкой

6.3.1 Силовые схемы

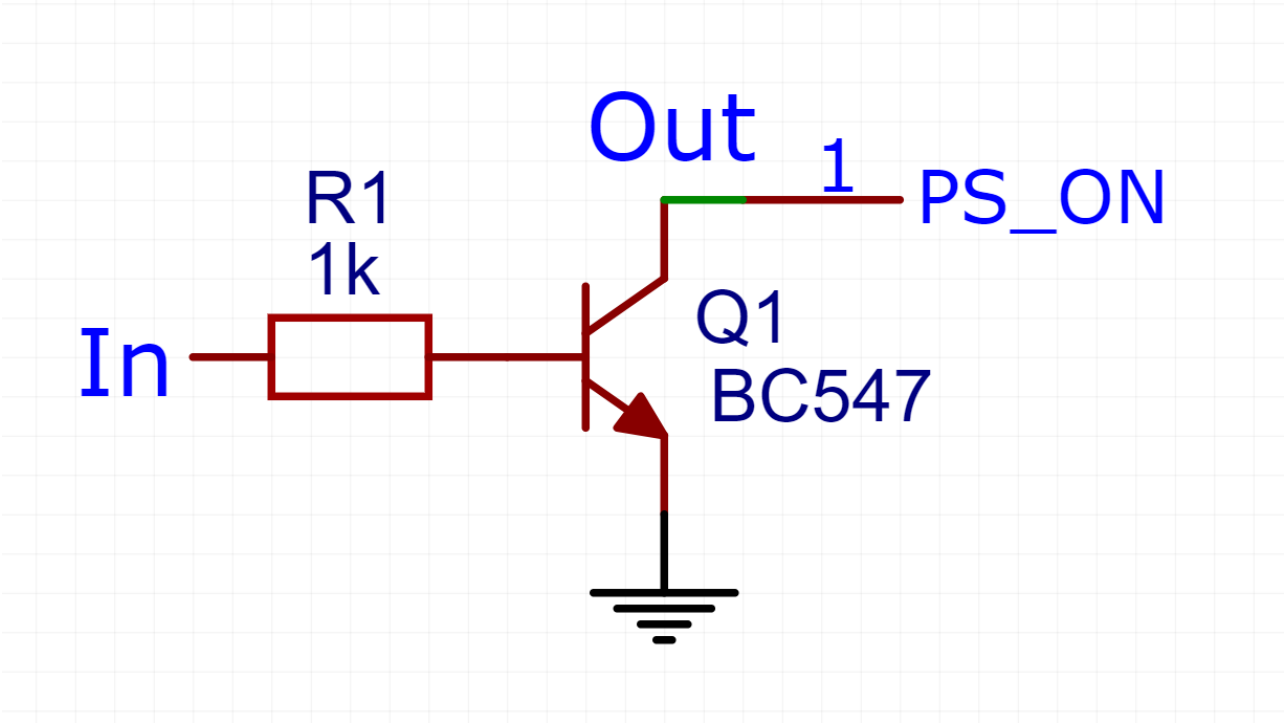
Одной из самых главных задач ***Smart Home System***, является управление различными бытовыми приборами. Для этого, управляющий сигнал микроконтроллера подается на специальные схемы управления мощной нагрузкой. В основном в проекте используются три силовые схемы:

- Полевой транзистор.
    - Подходит для коммутации нагрузки при напряжении 1—30V и токах в несколько А.
    - Поддерживает режимы on/off и PWM (даже на высоких частотах).
    - Может греться при больших токах и высоких частотах, в таком случае требуется теплоотвод.

Диод D1 необходим, если нагрузка индуктивная, чтобы не спалить транзистор.

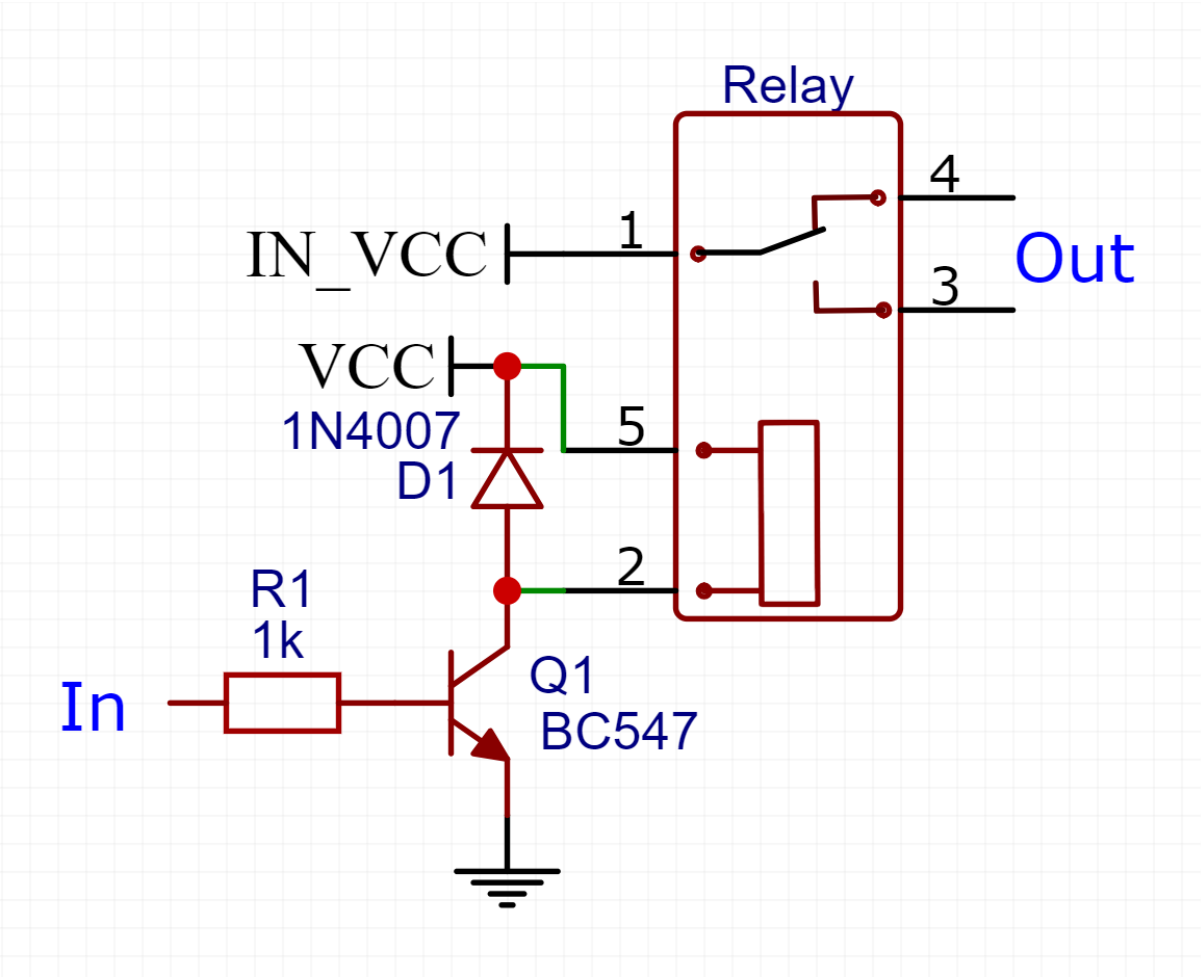


- Биполярный транзистор.
    - Практически полностью вытеснен полевиками.
    - Бывает полезен для различных подтяжек к земле (например, контакта PS_ON, для включения компьютерного блока питания) или для управления слаботочными реле.



- Реле.
    - Может работать как с низким, так и с высоким напряжением, при постоянном или переменном токе.
    - Поддерживает режим on/off, когда не требуется частых переключений.
    - Издает звуки при работе, в отличии от других компонентов.
    - Имеет короткий срок эксплуатации (за счет физического износа контактов при переключении).

При большом токе рекомендуется использовать искрогасящие цепи.



- Симистор.
  - Работает с переменным током.
  - Поддерживает режимы on/off и диммер.
  - Греется при мощной нагрузке.

6.3.2 Режим SWITCH

В этом режиме нагрузку можно только включить или выключить.

```
#include <Arduino.h>
// set pin for output
pinMode(myPIN, OUTPUT);

// on
digitalWrite(myPIN, HIGH);

// off
digitalWrite(myPIN, LOW);
```

6.3.3 Режим PWM

Режим позволят плавно управлять мощностью нагрузки, за счет широтно-импульсной модуляции. Нагрузка получает питание импульсами, а за счет высокой частоты и своей инертности импульсы

сглаживаются.

Рекомендуется повысить стандартную частоту PWM, если это позволяет сделать силовая схема (не перегревается). Удобнее всего это сделать с помощью библиотеки GyverPWM, либо вручную установить нужные регистры. **Работает только для ATmega328.** ESP выдают слабый ток и напряжение с пина, поэтому плохо подходят для ШИМ-контроллеров, если очень нужно изменить стандартные настройки — см. документацию на микроконтроллер и его ядро (где-то есть функции, т.к. у ESP8266 ШИМ в принципе реализован программно).

Более подробно можно прочитать в статьях про ШИМ и увеличение его частоты на сайте у Гайвера.

```
#include <Arduino.h>
#include <GyverPWM.h>

// optimal settings:
// only ATmega328!
PWM_prescaler(3, 1);
PWM_prescaler(9, 1);
PWM_TMR1_8BIT();

// PWM
// value: [0; 256) (if 8-bit PWM)
analogWrite(myPIN, value);
```

6.3.4 Режим DIMMER

Необходим для плавного управления нагрузкой переменного тока.

Удобнее всего использовать библиотеку GyverDimmer. *Лучше использовать алгоритм Брезенхема.*

6.3.5 Режим PID

Крутой алгоритм для поддержания заданной установки. Требует времени на подбор коэффициентов. Идеален для поддержания инертных процессов (температура, влажность...).

Удобнее всего использовать библиотеку GyverPID.

6.4 Датчики

6.4.1 Кнопка, энкодер, потенциометр

Механическая, либо сенсорная кнопка — самый простой датчик, но, несмотря на это, за счет обработки количества и времени нажатий, можно организовать удобное многофункциональное управление устройством.

Для удобной обработки нажатий в проекте используется библиотека EncButton. Помимо кнопки, она предлагает инструменты для работы с энкодером.

Для обработки потенциометра особых хитростей не надо, достаточно получить значение с АЦП и привести его в нужный диапазон.

```
#include <Arduino.h>

pinMode(Apin, INPUT);

int value = analogRead(Apin);
value = map(value, 0, 1023, 0, 100);
// or for [0, 256):
// value >>= 2;
```

6.4.2 Термистор и фоторезистор

Для преобразования показаний **термистора** используется уравнение Стейнхарта—Харта. Можно воспользоваться готовой библиотекой GyverNTC, либо создать функцию самостоятельно.

Для **фоторезистора** нужно измерить показания с АЦП. Других преобразований сделать не получится, единственный вариант — запомнить значения минимальной и максимальной освещенности и перевести в проценты яркости.

6.4.3 Термопара

С помощью термопары можно измерить высокие температуры в несколько сотен градусов. К ней требуется драйвер-усилитель, для работы с которым есть хорошая библиотека GyverMAX6675.

6.4.4 DS18B20

Достаточно точный цифровой датчик температуры.
Библиотека: microDS18B20

6.4.5 HTU21D

Точный датчик температуры и влажности воздуха. Для работы с ним есть легкая библиотека GyverHTU21D.

6.4.6 BME280/BMP280

На основе BME280 можно собрать полноценную метеостанцию, измеряющую температуру, влажность воздуха и атмосферное давление.

Удобная библиотека: GyverBME280.

6.4.7 MH-Z19b

Современный датчик, позволяющий определять концентрацию углекислого газа (CO2) в воздухе. Очень полезен для информирования об опасном уровне или организации автоматической форточки (системы проветривания).

По работе с ним есть статья, позже в проекте появится своя библиотека.

6.4.8 HX711

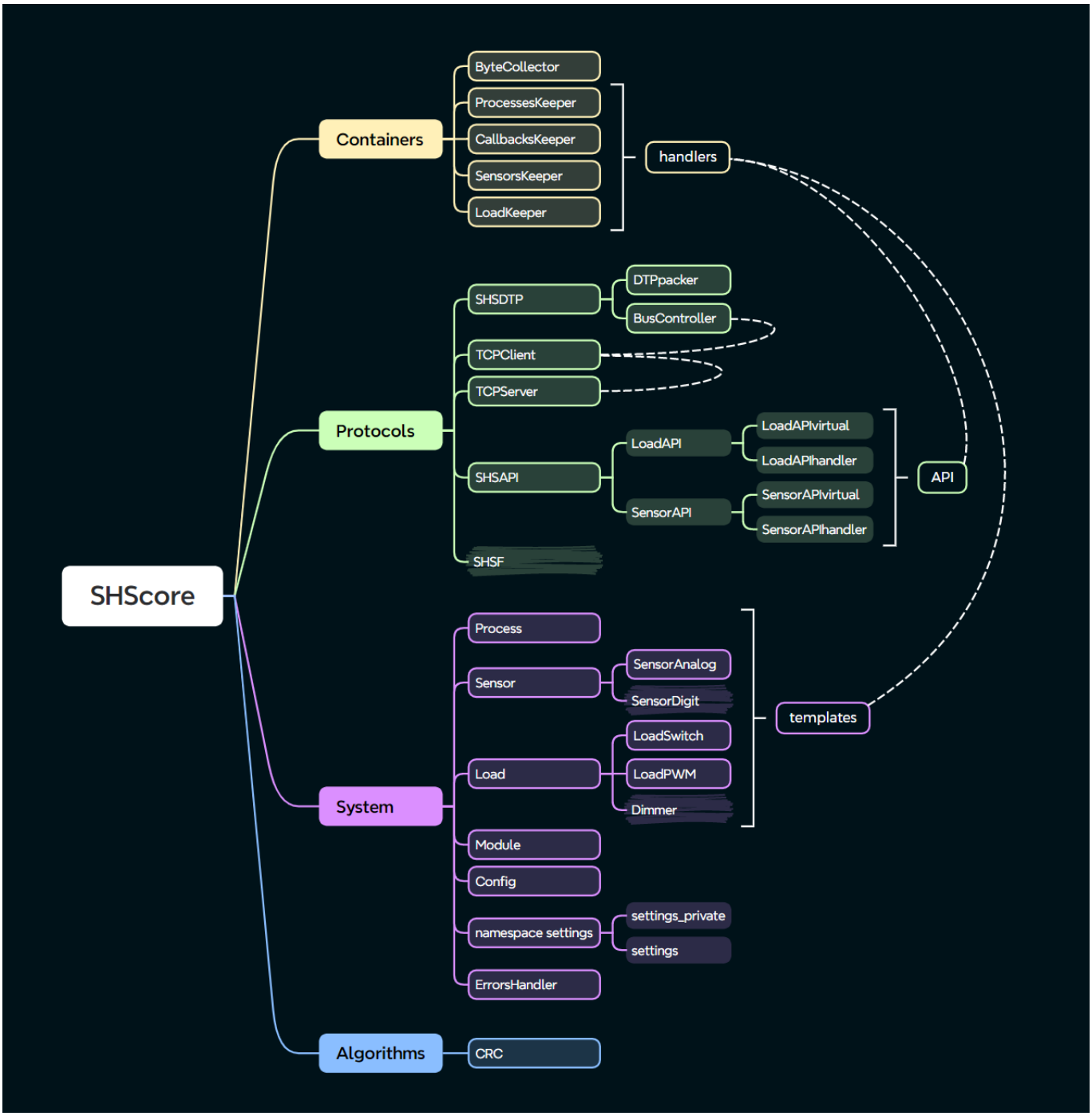
Драйвер для различных датчиков веса. Можно отслеживать или отмерять жидкости по весу.

Библиотека: GyverHX711.

7. Этап II. Проектирование ядра (бизнес-логики)

SHScore — логика работы системы, основанная на парадигме **ООП** и принципах **SOLID**. В ядре описаны абстрактные классы для оперируемых системой объектов и процессов, а также обработчики для них, протоколы связи и передачи данных. Ядро дискретизирует и описывает действия системы, что позволяет быстро и удобно создавать и изменять конфигурацию системы и последующие классы высокого уровня (библиотеки для конкретных задач).

Конечно, парадигма ООП с длинными цепочками наследования, виртуальными функциями и указателями расходует гораздо больше памяти и вычислительных ресурсов, чем функциональная. Такой выбор сделан намерено, с учетом того, что система разбита на большое количество модулей и каждый микроконтроллер будет содержать небольшое количество оперируемых объектов. Зато ООП позволяет организовать систему очень удобно, быстро и без лишнего дублирования кода.



7.1 Containers

7.1.1 ByteCollector

Для удобной упаковки данных в массив байтов и дальнейшей их обработки разработан легкий класс: SHSByteCollector.h.

Класс используется в протоколах передачи данных и API.

```
class shs::ByteCollector
{
public:
    uint8_t *buf{};      // array
    uint8_t *ptr{};      // current position
    uint8_t *readPtr{};  // read position
```

```
    explicit ByteCollector(uint8_t size);

    /*
      The bytes argument specifies how many bytes
      to write from the passed type.

      int value = 1000;
      bc.add(value, 2);  // will add 2 bytes
      bc.size();         // will return 2, not 4!

      bc.add(value);     // will add sizeof(value)
      bc.size();         // will return 6

    */
    // add to the end
    template <typename T>
    void add(const T &value, uint8_t bytes = sizeof(T));

    // add to the beginning
    template <typename T>
    void addBefore(const T &value, uint8_t bytes = sizeof(T));

    // unpack data
    template <typename T>
    void get(T &var, uint8_t bytes = sizeof(T))

    // reserve bytes for more size
    void reserve(uint8_t size);
    void reserveBefore(uint8_t size)

    uint16_t size();
};
```

7.1.2 Handlers

Классы-контейнеры для обработчиков процессов. Они позволяют оперировать несколькими объектами как одним, что дает возможность легко масштабировать систему, без вмешательства в логику обработки данных.

- ProcessesKeeper — позволяет добавлять различные процессы, например, опросы датчиков, циклические вычисления, парсинг сайтов...

```
class shs::ProcessesKeeper : public shs::Process
{
public:
    explicit ProcessesKeeper();

    void attach(shs::Process *object);
    void detach(shs::Process*object);
```

```
    void begin();
    void tick();
    void end();

protected:
    std::vector<shs::Process *> m_ptrs;
    uint8_t m_find(const shs::Process*object);
};
```

- CallbacksKeeper — в него добавляются API датчиков и модулей, которые вызываются при поступлении новых данных.

```
class shs::CallbacksKeeper
{
public:
    explicit CallbacksKeeper() {}

    void attach(shs::API *object);
    void detach(shs::API *object);

    uint8_t handler(shs::ByteCollector &data);

protected:
    std::vector<shs::API *> m_ptrs;
    uint8_t m_find(shs::API *object);
};
```

- SensorsKeeper — хранит набор датчиков и позволяет вызывать методы конкретного датчика по ID.

```
class shs::SensorsKeeper : public Sensor
{
public:
    explicit SensorsKeeper();
    void attach(shs::Sensor *object);
    void detach(shs::Sensor *object);

    void setup() override;

    uint8_t find(const shs::settings::shs_ID_t ID);

    int16_t getValueI(const shs::settings::shs_ID_t ID) override;
    shs::settings::shs_float_t getValueF(const shs::settings::shs_ID_t ID)
override;
    shs::settings::shs_double_t getValueD(const shs::settings::shs_ID_t ID)
override;

    int16_t getAverageI(const shs::settings::shs_ID_t ID) override;
```

```
    shs::settings::shs_float_t getAverageF(const shs::settings::shs_ID_t ID)
override;
    shs::settings::shs_double_t getAverageD(const shs::settings::shs_ID_t
ID) override;

protected:
    std::vector<shs::Sensor *> m_ptrs;
    uint8_t m_find(const shs::Sensor *object);
};
```

- LoadKeeper — аналогично SensorsKeeper управляет нагрузками.

```
 class shs::LoadKeeper : public shs::Load
{
public:
    explicit LoadKeeper();

    void attach(shs::Load *object);
    void detach(shs::Load *object);

    uint8_t find(shs::settings::shs_ID_t ID);

    void setup() override;

    void on(const uint8_t value = 255, const uint8_t smoothing = 0, const
shs::settings::shs_ID_t ID = 0) override;
    void on(const uint16_t value = UINT16_MAX, const uint16_t smoothing = 0,
const shs::settings::shs_ID_t ID = 0) override;

    void off(const uint16_t smoothing = 0, const shs::settings::shs_ID_t ID
= 0) override;


protected:
    std::vector<shs::Load *> m_ptrs;
    uint8_t m_find(const shs::Load *object);
};
```

7.2 Protocols

7.2.1 SHSDTP

Smart Home System Data Transmission Protocol — единый протокол передачи данных, разработанный для передачи информации между всеми модулями. Идея взята из GyverBus.

Для отправки данные нужно обработать, а потом распаковать обратно, для этого создан SHSdtp.h. Он добавляет к пакету данные об отправителе и получателе, общее количество байт и CRC. Затем данные отправляются любым способом, основанным на классе `Stream`. Если не используется стандартная

библиотека `<Arduino.h>`, то класс `Stream` необходимо реализовать отдельно, унаследовавшись от абстрактного класса shs::Stream:

```
class shs::Stream
{
public:
    virtual uint8_t write(const uint8_t *buf, uint16_t size) = 0;
    virtual uint8_t read() = 0;
    virtual uint8_t available() = 0;
};
```

В конструкторе `DTP` принимает указатель на объект `Stream`.

```
/*
  Smart Home System Data Transmission Protocol

  The idea is taken from https://github.com/GyverLibs/GyverBus
*/

/*
  WARNING! It is necessary to include Stream-class befor this file.
  If the <Arduino.h> is not used, you need to create your own class inherited from
  abstract class shs::Stream (SHSStream.h)
*/

namespace shs::settings
{
#ifndef SILENCE_TIMEOUT
#define SILENCE_TIMEOUT 120000
#endif

    inline const uint8_t DTP_OFFSETbeg = 5;
};

namespace shs
{
    namespace DTPcommands
    {
        enum DTPcommands : uint8_t;
    };

    namespace DTPhandlerStatus
    {
        enum DTPhandlerStatus : uint8_t;
    };

    struct DTPdata;
    class DTP;
    class DTPpacker;
    typedef void (*DTPhandler_t)(shs::DTPdata &);
```

```
};

enum shs::DTPcommands::DTPcommands : uint8_t
{
    answer = 252,
    error,
    request,
};

struct shs::DTPdata
{
    shs::settings::shs_ModuleID_t to{};
    shs::settings::shs_ModuleID_t from{};
    shs::settings::shs_ID_t apiID{};
    uint8_t datasize{};
};

/*
  ----------------------------------------
  DTPpacker
  ----------------------------------------
*/
class shs::DTPpacker
{
public:
    // uint8_t packDTP(shs::ByteCollector *bc, const uint8_t to, const int16_t
apiID);
    uint8_t packDTP(shs::ByteCollector *bc, const shs::settings::shs_ModuleID_t
to, const shs::settings::shs_ID_t apiID, const shs::settings::shs_ModuleID_t
from);
    uint8_t checkDTP(shs::ByteCollector *bc);
    uint8_t parseDTP(shs::ByteCollector *bc, shs::DTPdata &data);

protected:
    shs::CRC8 _crc;
};

/*
  ----------------------------------------
  DTP
  ----------------------------------------
*/
class shs::DTP : public shs::DTPpacker, public shs::CallbacksKeeper
{
public:
    explicit DTP(Stream *bus, const shs::settings::shs_ModuleID_t ID);
    ~DTP();

    uint8_t tick();
    uint8_t checkBus(uint8_t len = UINT8_MAX);

    uint8_t sendPacket(shs::ByteCollector *bc, const shs::settings::shs_ModuleID_t
to);
```

```
    uint8_t sendPacket(shs::ByteCollector *bc, const shs::settings::shs_ModuleID_t
to, const shs::settings::shs_ID_t api_ID, const shs::settings::shs_ModuleID_t
from);
};
```

7.2.2 SHSF

Smart Home System File — протокол записи данных в файл, а так же само расширение файла (.shsf). Содержит инструменты для записи, сжатия, чтения данных, сохранения настроек и конфигураций.

В данной версии проекта имеются наброски, но реализация еще не готова.

7.2.3 TCP/IP

Для передачи данных по WiFi используется протокол TCP/IP. Для системы хватает стандартной реализации протокола в ядрах esp, однако для удобства добавлены расширения.

SHSTcpClient.h и SHSTcpServer.h упрощают процессы подключения и опроса входящих потоков.

```
class shs::TcpServer
{
public:
    WiFiServer server;
    WiFiClient *clients;
    shs::DTP *dtp{};

    const uint8_t *IP{};
    uint8_t maxClients{};

    TcpServer(const uint8_t *IPaddress, uint16_t port = 50000, uint8_t max_clients
= 6);
    ~TcpServer();

    void begin();
    void tick();

    uint8_t sendPacket(shs::ByteCollector *bc, const shs::settings::shs_ModuleID_t
to,
                       const shs::settings::shs_ID_t api_ID);
};
```

7.3 System

7.3.1 Process

SHSProcess.h — абстрактный класс, описывающий интерфейс процессов в системе:

- `begin()` — вызывается один раз во время конфигурации процесса (например, блок `setup`).
- `tick()` — постоянно вызывается в цикле программы.
- `end()` — вызывается для завершения процесса.

```
class shs::Process
{
public:
    virtual void begin() = 0;
    virtual void tick() = 0;
    virtual void end() = 0;
};
```

7.3.2 Sensor

- Класс `shs::Sensor` (SHSSensor.h) является абстрактным и описывает интерфейс для получения значений с датчиков. В `type` хранится тип датчика, которых с развитием поддерживаемых системой устройств может быть несколько десятков.
- В классе `shs::SensorAnalog` (SHSSensorAnalog.h) реализованы методы считывания значений с аналоговых выводов микроконтроллера и подсчет среднего значения из выборки измерений.

На основе этих классов в SHSlibrary разрабатываются классы для конкретных моделей датчиков, расширяя набор поддерживаемых устройств.

```
// SHSsensor.h
enum shs::SensorType::SensorType : uint8_t
{
    unknown,
    analogPin,
    thermistor,
    photoresistor,
};

class shs::Sensor
{
public:
    shs::SensorType::SensorType type{};

public:
    explicit Sensor(const shs::settings::shs_ID_t ID = 0, const
shs::SensorType::SensorType stype = shs::SensorType::unknown);

    void setID(const shs::settings::shs_ID_t ID);
    shs::settings::shs_ID_t getID() const;

    virtual void setup() = 0;

    virtual int16_t getValueI(const shs::settings::shs_ID_t ID = 0) = 0;
    virtual shs::settings::shs_float_t getValueF(const shs::settings::shs_ID_t ID
= 0) = 0;
    virtual shs::settings::shs_double_t getValueD(const shs::settings::shs_ID_t ID
= 0) = 0;

    virtual int16_t getAverageI(const shs::settings::shs_ID_t ID = 0) = 0;
```

```
    virtual shs::settings::shs_float_t getAverageF(const shs::settings::shs_ID_t
ID = 0) = 0;
    virtual shs::settings::shs_double_t getAverageD(const shs::settings::shs_ID_t
ID = 0) = 0;

protected:
    shs::settings::shs_ID_t m_sensorID{};
};

// SHSSensorAnalog.h
class shs::SensorAnalog : public shs::Sensor
{
public:
    explicit SensorAnalog(const shs::settings::shs_ID_t ID = 0, const
shs::SensorType::SensorType stype = shs::SensorType::unknown, const uint8_t pin =
A0);

    void setup() override;
    shs::settings::shs_float_t getValueF(const shs::settings::shs_ID_t ID = 0)
override;
    shs::settings::shs_double_t getValueD(const shs::settings::shs_ID_t ID = 0)
override;
    int16_t getValueI(const shs::settings::shs_ID_t ID = 0) override;

    shs::settings::shs_float_t getAverageF(const shs::settings::shs_ID_t ID = 0)
override;
    shs::settings::shs_double_t getAverageD(const shs::settings::shs_ID_t ID = 0)
override;
    int16_t getAverageI(const shs::settings::shs_ID_t ID = 0) override;

protected:
    uint8_t m_pin{};
    uint16_t m_getSamplesSum(const uint8_t count =
shs::settings::SENSOR_AVERAGE_SAMPLES);
};
```

7.3.3 Load

SHSLoad.h — абстрактный класс, описывающий интерфейс взаимодействия с нагрузками.

Smart Home System предлагает на выбор 3 режима управления нагрузкой:

- `SWITCH` — нагрузку можно только полностью включить или полностью выключить.
- `PWM` — можно управлять мощностью нагрузки в сети постоянного тока с помощью ШИМ.
- `DIMMER` — аналог ШИМ для сети переменного тока.

Параметр `smothing` работает только для режимов `PWM` и `DIMMER`. Это коэффициент, устанавливающий скорость изменения управляющего сигнала, например, для плавного регулирования света. Коэффициент задает время, за которое значение `value` нагрузки будет увеличено на 1.

Для работы `smoothing` необходимо передать в класс объект типа `shs::ProcessesKeeper*`. Когда потребуется выполнить сглаживание, класс сам добавится, а затем удалится из массива keeper'а.

```
enum shs::LoadType::LoadType : uint8_t
{
    UNKNOWN,
    SWITCH,
    PID,
    PWM,
    DIMMER,
};

class shs::Load
{
public:
    shs::LoadType::LoadType type;

public:
    explicit Load(shs::settings::shs_ID_t ID = 0, shs::LoadType::LoadType ltype =
shs::LoadType::UNKNOWN);

    void setID(const shs::settings::shs_ID_t ID);
    shs::settings::shs_ID_t getID() const;

    virtual void setup() = 0;

    virtual void on(const uint8_t value = 255, const uint8_t smoothing = 0, const
shs::settings::shs_ID_t ID = 0) = 0;
    virtual void on(const uint16_t value = UINT16_MAX, const uint16_t smoothing =
0, const shs::settings::shs_ID_t ID = 0) = 0;

    virtual void off(const uint16_t smoothing = 0, const shs::settings::shs_ID_t
ID = 0) = 0;

protected:
    shs::settings::shs_ID_t m_loadID{};
}
```

7.3.4 ErrorsHandler

В класс `shs::ErrorsHandler` (SHSErrorsHandler) передаются коды ошибок, которые обрабатываются подключенными обработчиками.

Класс рассчитан на два обработчика: первый используется системой, второй — пользователем. Возможно, в будущем будет рассчитан на любое число обработчиков.

Коды всех ошибок находятся в отдельном файле SHSerrors.h. При необходимости, `enum shs::errors::Errors` может быть дополнено пользователем.

```
class shs::ErrorsHandler
{
public:
    ErrorsHandler(shs::errorsCallback_t system_callback = nullptr,
```

```
    shs::errorsCallback_t user_callback = nullptr);

    void attachFirstHandler(shs::errorsCallback_t callback);
    void attachSecondHandler(shs::errorsCallback_t callback);

    void error(const shs::errors::Errors error);

    [[nodiscard]] shs::errors::Errors getLastError();
    [[nodiscard]] uint8_t getCount();
};
```

7.3.5 Settings

Все настройки обернуты в `namespace shs::settings`. Файл SHSsettings.h является пользовательским шаблоном настроек. Там можно разово задать общие параметры для всей системы. В файл SHSsettings_private.h рекомендуется не вносить изменения, во избежание некорректной работы системы.

7.4 Algorithms

Для обработки информации в системе потребуется разработать много легких и эффективных алгоритмов: сортировка и сжатие данных, поиск и оптимизация хранения, собсвтенные аллокаторы для работы с памятью, хеш-функции и пр.

7.4.1 CRC

Циклическая контрольная сумма — очень важный алгоритм для проверки целостности данных при передаче или хранении в файле.

Ядро предлагает реализацию самых удобных и востребованных для ***Smart Home System*** типов на 8, 16 и 32-bit.

Ниже приведен класс для CRC-32, остальные организованы полностью аналогично, см. SHSCRC.h.

```
namespace shs
{
    class CRC8;
    class CRC16;
    class CRC32;

    const uint8_t CRC8_beg = 0x00;
    const uint16_t CRC16_beg = 0xFFFF;
    const uint32_t CRC32_beg = 0x00000000;
};

class shs::CRC32
{
public:
    uint32_t crc{};
    explicit CRC32() : crc(shs::CRC32_beg) {}
```

```
    void add(const uint8_t value);
    void clear();

    void update(uint32_t &crc, uint8_t data);
    uint32_t crcBuf(const uint8_t *ptr, uint16_t size);
};
```

8. Этап III. Разработка библиотеки

SHSlibrary — следующий уровень программной архитектуры ***Smart Home System***. Библиотека реализует более функциональные и направленные решения, API к устройствам и библиотекам.

Пока что большая часть библиотеки находится в разработке. Сейчас в ней представлен класс SHSAutoProcesses, позволяющий выполнять автоматическую конфигурацию и управление модулем.

Позже в библиотеку будут добавлены классы для работы с датчиками, более продвинутые модели API, инструменты конфигурации и сохранения системы, обработки с файлов и данных.

```
class shs::AutoProcesses : public shs::Process
{
public:
    WiFiClient tcp;
    shs::DTP dtp;
    shs::LoadAPIhandler loadAPI;
    shs::SensorAPIhandler sensorsAPI;

    explicit AutoProcesses();

    void begin() override;
    void tick() override;
    void end() override;
};
```

9. Этап IV. Объединение модулей в единую систему

9.1 Организация соединения

9.1.1 Покрытие сети

Все модули Smart Home System обмениваются между собой данными по WiFi. Для стабильной работы всех устройств необходимо обеспечить высокий уровень сигнала общей сети для каждого модуля.

Если помещение небольшое и установлен мощный маршрутизатор (WiFi-роутер), то скорее всего никаких дополнительных действий не потребуется. В противном случае, проблему можно решить путем установки WiFi-repeater'ов. Они подключаются в сеть, увеличивают ее радиус покрытия и уровень сигнала.

Вторым шагом необходимо научить все устройства находить друг друга в сети. Для этого им надо раздать статические IP-адреса. Такой адрес привязывается к MAC-адресу платы (он идет с завода, но при желании можно поменять). Делается все в настройках роутера, процедура несложная. Теперь, зная IP устройства, можно будет посылать и принимать данные.

Для удобства настройки в ***Smart Home System*** модули имеют схожие MAC-адреса. Подробнее см. ниже.

9.1.2 Подключение модулей

После настройки маршрутизатора, остается подключить и настроить сами микроконтроллеры. Для этого в SHScore есть набор функций, реализованных в файле SHSconnectWiFi.h.

```
namespace shs
{
        // set mac-address SHSma + id:
        // 53:48:53:6D:61:xx xx = 0x(id)
        void setMac(const uint8_t id);

        // set mac-address
        void setMac(const uint8_t *mac);

        // wi-fi connection, called 1 time
        // default:
        // ssid = WIFI_SSID
        // pass = WIFI_PASSWORD
        // from the SHSsettings.h
        void connectWiFi(const char *ssid, const char *pass);

        // call constantly, will cause a reboot if the board is
        // disconnected from WiFi for a long time
        void checkReconnection();
};
```

Задать `SSID` и `PASSWORD` сети можно один раз для всех устройств в файле SHSsettings.h.

MAC-адрес для удобства зашифрован кодом `SHSma` (53:48:53:6D:61:) (т.е. сокращение от **Smart Home System MAC Address**), а на последнем месте указывается уникальный `ID` модуля, таким образом устройства удобно отслеживать и настраивать в маршрутизаторе — достаточно изменить только последнее число.

9.1.3 TCP/IP

TCP/IP — основной протокол передачи данных в Интернете. Через него организуется соединение между всеми модулями. Для этого в ядре ESP есть классы `WiFiClient` и `WiFiServer`.

На всех модулях создаются объекты класса `WiFiClient`, которые подключаются к серверу. В SHScore есть свой класс, но необходимости в нем на данном этапе проекта нет, в отличии от SHSTcpServer.h, который отвечает за обработку всех клиентов. Подробнее об этих классах было написано выше в п. 7.2.4 TCP/IP.

9.2 API

Все устройства связаны и имеют доступ друг к другу. Чтобы они могли запрашивать и принимать данные, нужно определить соответствующие команды и обработчики для них. `API` каждого модуля состоит из перечисления(`enum`), где каждой команде соответствует числовой код. В пакет данных передается команда, а затем дополнительные параметры, если она их требует. В таком же порядке данные и будут расшифровываться на стороне приемника.

Для разработки `API` следует наследоваться от абстрактного класса `shs::API`. API имеет смысл разделять на реальные (т.е. непостредственно управляющие устройством) и виртуальные (непосредственно управляющие устройством). API можно написать для управления микроконтроллером или его частью, какой-то библиотекой или сущностью, целым модулем или подсистемой.

Каждому типу API следует задавать свой `shs::settings::shs_ID_t apiID`, для правильной и быстрой расшифровки протоколами передачи данных.

10. Этап V. Взаимодействие с пользователем

В Smart Home System реализованы четыре метода взаимодействия с пользователем:

1. Кнопки и датчики.
2. Чат-боты.
3. GUI (приложение).
4. Голосовое управление (и даже своя колонка-ассистент).

Каждый из них позволяет изменять набор параметров и регулировать систему.

10.1 Кнопки и датчики

Датчики движения позволяют оценить нахождение пользователя в пространстве и скорректировать поведение системы. Таким образом, когда пользователь вне дома, включается досвечивание растений, шумные приборы (например, вытяжки, фильтры, насосы).

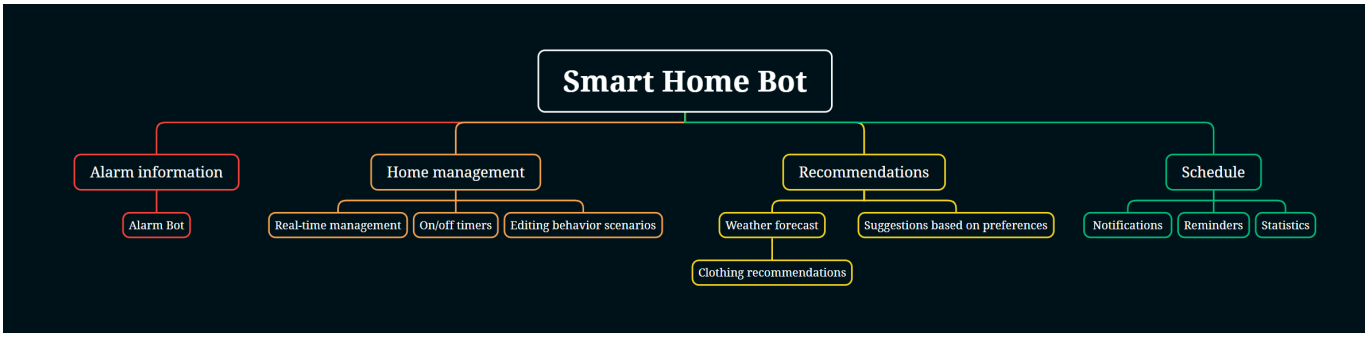
Кнопки лучше всего расположить в наиболее удобных местах: у входов в комнаты, около рабочего места, рядом с кроватью. Таким образом, выходя из комнаты, можно дать команду перейти в ждущий режим, за рабочим местом — добавить света, а лежа в кровати — включить тихий режим и подготовиться ко сну.

10.2 Telegram bot

@SHS_sbot

Масштабная разработка, которая позволяет не только управлять умным домом, но и пользоваться дополнительными функциями для составления расписания, напоминаний, получения различной информации. Кроме того, бот может предупредить об ухудшении погоды или присутствии посторонних рядом с домом.

Бот умеет показывать меню с кнопками, нажимая на которые можно очень быстро управлять системой. По специальным алгоритмам меню корректируются и высвечиваются актуальные на данный момент кнопки. Например — к вечеру меню предложит гасить свет и переводить систему в бесшумный режим; когда пользователь возвращается домой, меню может предложить кнопки для включения чайника, света, отопления и пр.



Бот лежит на сервере и имеет быстрый доступ к информации хранилища и управлению Smart Home System. Для реализации есть две библиотеки FastBot и FastBot2 (бот потихоньку переезжает на вторую), планируется расширение `SHStbot` с дополнительным функционалом.

10.3 Приложение

GUI — графический интерфейс пользователя, т.е. различные приложения и web-страницы. Один из самых удобных способов организовать управление системой – визуализировать графики и настройки, сделать меню для конфигурации системы и настройки связей. Разработка хорошего приложения требует много времени, но результат того стоит.

Главное приложение находится в папке SHSapp, но пока что детально не разработано. Рабочая упрощенная версия есть в демонстрационной части проекта: demo_version.

10.4 Голосовое управление

Построено на специальном контроллере распознавания голоса. Библиотека для него достаточно неудобная и нефункциональная, поэтому в ближайших версиях в проекте появится своя. Разработки

уже ведутся.

Модуль имеет хранилище распознавателя и общее хранилище команд. Таким образом одновременно он сравнивает 7 команд, затем передает код распознанной команды, основываясь на ней, управляющий микроконтроллер загружает в распознаватель следующую порцию команд. Таким образом можно строить цепочки сложных команд и организовать своего голосового помощника.

В ближайшее время в проекте появится своя умная колонка с голосовым ассистентом "Сом" (т.е. Smart hOMe).

11. Этап VI. Обучение системы саморегулированию

11.1 Алгоритмы поведения системы

11.1.1 Автоматизация на основе датчиков

Система постоянно опрашивает датчики и получает с них информацию. На основе полученных данных система может сама контролировать процессы.

Первым делом стоит автоматизировать все процессы, которые возможно. Свет может включаться по датчикам уровня освещенности, вентиляторы и вытяжки, когда зашкаливает датчик углекислого газа, ночью система не должна шуметь и издавать звуков.

11.1.2 Модели и режимы поведения

Далее следует разделить поведение системы на несколько моделей примерно так:

1. `Normal` — система работает в обычном режиме, все устройства функционируют без ограничений.
2. `StandBy` — система функционирует, продолжает опрашивать датчики и обрабатывать данные, но практически не включает никаких устройств.
3. `Silence` — система функционирует, но стремится ограничить все шумные процессы. Таким образом, обороты вентилятора будут максимально снижены, а реле, вытяжки, моторы и другие шумные приборы без крайней необходимости включаться не будут.

Разумеется, тремя режимами "умный" дом ограничиваться не должен, поэтому список можно продолжать, описывая различные ситуации и подстраивая под них систему.

У структуры `shs::Config` есть поле `mode`, отвечающее за текущее состояние системы, однако таким образом можно устанавливать режим как для всей системы, так и для ее отдельных частей (подсистем, модулей или конкретных устройств (процессов)).

11.1.3 Автоматический выбор режима

Чтобы система использовала актуальный режим, надо построить логику выбора модели поведения, основываясь на времени. Переключение общих режимов (`Normal`, `StandBy`, `Silence`...) можно привязать ко времени суток. Отдельные приборы можно настроить по более точному графику, связав время и дни недели, таким образом, что чайник будет горячим по утрам в будни и ближе к обеду в

выходные дни. Расписание работы шумных приборов можно назначить на время отсутствия пользователя дома.

Правильно задав все параметры, можно добиться стабильной работы системы с оптимальным выбором режима.

11.2 SHSNeuralNet

SHSNeuralNet — разработка легкой нейронной сети для автоматического управления микросистемами и устройствами. За основу взята нейронная сеть из проекта Winderton'a.

11.2.1 Оригинальное устройство SHSNeuralNet

Сама нейронная сеть состоит из двух классов:

1. `neuron` — отдельный нейрон. Для активации выбрана функция гиперболического тангенса.
2. `net` — соединяет в себе нейроны.

Далее для взаимодействия с сетью автор предложил класс `trainingSet`, он может построить сеть по указанной топологии и обучить ее на представленных данных.

11.2.2 Реализация в Smart Home System

Для использования в ***Smart Home System*** класс `trainingSet` не подходит, поэтому на его основе будет создан другой.

Вдобавок к существующим будет разработано еще несколько классов-инструментов для работы с нейронной сетью. Они будут сохранять ее в энергонезависимую память, разворачивать обратно, автоматически подбирать наилучшую топологию (структуру) сети, собирать информацию для обучения сети и проводить его в реальном времени. Эти инструменты позволят расширить функционал и заменять одну сеть на другую, просто вытащив ее из постоянной памяти.

Классы будут оптимизированы и доработаны. Такая нейросеть будет спокойно запускаться и функционировать на ресурсах одного микроконтроллера, а смена одной нейронной сети на другую расширит возможности ее применения все еще на том же контроллере.

Конечно, такая нейросеть достаточно примитивная и для сложного анализа большого объема данных она никак не подходит, однако это идеальный вариант в рамках умного дома.

11.2.3 Управление системой

SHSNeuralNet принимает на вход множество факторов, обрабатывает их и выдает результат на выходе. Таким образом, она может контролировать процессы, которые сложно или долго описывать вручную. Она может определять, когда надо начать будить пользователя, как долго его не будет дома, когда подогреть чайник к его приходу и в какой момент пора начинать подготовку ко сную. Все это происходит в режиме реального времени, основываясь на показаниях датчиков, времени нахождения пользователя в разных частях дома, его занятиях и предпочтениях, которые обусловлены временными, внешними и внутренними факторами.

Обрабатывая большое количество факторов, нейросеть способна очень эффективно управлять системой и ее отдельными процессами, корректируясь (в рабочем режиме), что актуально, если,

например, у пользователя наступил отпуск и сменилась его манера поведения.

11.2.4 Предсказание событий

SHSNeuralNet может взять в "голову" много факторов и находить среди них закономерности. Основываясь на них, нейросеть может попробовать предсказать событие. Такое применение очень актуально для некоторых устройств и процессов:

- Для предсказания изменения погоды людьми давно выведены закономерности, основанные на наблюдении за давлением, влажностью, температурой воздуха и прочими параметрами. Очень сложно запрограммировать и учитывать все факторы и законы одновременно. Поэтому встроенная в метеостанцию нейросеть после грамотного обучения будет способна очень точно предсказывать изменение погоды в ближайшие несколько часов. Кроме того, нейросеть сможет сформировать рекомендации по выбору одежды для выхода на улице.
- ***Smart Home System*** предлагает огромное количество настроек системы. Разобраться с ними частично помогают графические интерфейсы. Чтобы пользователю было удобнее управлять системой, она должна предлагать небольшой интерфейс, который позволяет быстро и удобно скорректировать необходимые параметры. Для формирования такого интерфейса отлично подойдет **SHSNeuralNet**.Она обрабатывает время, прошлые предпочтения пользователя и его поведение, после чего формирует наиболее востребованный интерфейс на данный момент. Например, ближе к вечеру система предложит кнопку для плавного выключения света и подготовки ко сну, а утром наоборот — расположит кнопки для включения всех устройств и настройки расписания на день.

11.3 AdvancedNeuralNet

Идея заключается в распределении нейронной сети по всем модулям. Современные нейросети создаются на больших серверах и требуют огромных вычислительных ресурсов. ***Smart Home Syste*** предлагает встраивать нейроны или мини-нейронные сети прямо в датчики и модули, а потом объединять их в единую сеть.

Такая технология позволит снизить технические требования к главному серверу и сможет учитывать не только показания с датчиков, но и вес этих датчиков в системе, т.к., по сути, они будут являться отдельными нейронами.

Для более точного понимая возможностей такой сети необходимо подготовить базу, собрать много данных и провести серию экспериментов. Все это будет постепенно разрабатываться в следующих версиях проекта.

12. Этап VII. Настройка системы

Конфигурация и настройка системы — масштабный этап ее разработки, к которому стоит подходить максимально ответственно. Пока что в ***Smart Home System*** мало инструментов и интерфейсов, упрощающих процесс конфигурации и отладки. В скором времени пользователю будут доступны удобные GUI, что избавит его от необходимости писать код или вносить в него какие-то изменения.

12.1 Настройка на уровне ядра

Пока удобных инструментов не создано, настройка системы производится в специальных файлах. Когда же они будут созданы, этот пункт будет полезен разработчикам и продвинутым пользователям.

12.1.1 SHSsettings.h

В SHScore лежит файл SHSsettings_template.h, его необходимо скопировать и положить куда-нибудь в папку с библиотеками (для Ардуино), например, создать отдельную папку SHSsettings.

В файле надо указать данные для подключения к WiFi и IP адрес, если он не подходит.

Для оптимизации вычислений с плавающей точкой и ускорения работы можно раскомментировать `#define USE_FLOAT_FOR_DOUBLE`. После этого тип данных `shs::settings::shs_double_t` будет синонимом `float`.

12.1.2 SHSsettings_private.h

Файл лежит в глубине ядра и отвечает за настройки, которые не рекомендуется изменять. Там объявлены некоторые типы данных и заданы системные константы. Некоторые параметры можно скорректировать в файле SHSsettings.h, остальные должны оставаться неизменными, в противном случае поведение системы может быть непредсказуемым или выпадут ошибки при компиляции.

12.2 Конфигурация системы

Аналогично настройкам, удобные интерфейсы для конфигурации еще в разработке. Сейчас в системе нет строго стандарта конфигурации. Ее можно задать как в файле настроек, дописав туда отдельные поля (так сделано, например в demo_version), так и в файлах прошивок.

Позже будет написан парсер файлов конфигурации, который избавит пользователя от написания кода. Затем, останется создать только графический интерфейс, чтобы пользователю вообще не приходилось редактировать какие-либо файлы.

13. Resources

Для повторения проекта рекомендую изучить следующие ресурсы. Это только малая часть необходимой информации, но изучая ее, пользователь найдет много ссылок на сторонние источники, а также поймет, как и где найти необходимую информацию.

13.1 Уроки и гайды

- https://alexgyver.ru/arduino-first/ — настройка среды разработки.
- https://alexgyver.ru/lessons/ — уроки по программированию и микроконтроллерам.
- https://alexgyver.ru/arduino_lessons/ — еще уроки с крутыми примерами и небольшими проектами.
- https://amperka.com — у них есть база гайдов по подключению датчиков.

13.2 Документация и библиотеки

- https://github.com/arduino/ArduinoCore-avr — ядро Arduino (avr) и документация.
- https://github.com/AlexGyver/GyverCore — улучшенный аналог предыдущего пункта, рекомендуется использовать его.
- https://github.com/esp8266/Arduino — ядро esp8266 и документация.
- https://github.com/espressif/arduino-esp32 — ядро esp32 и документация.

13.3 Поиск компонентов

- https://alexgyver.ru/arduino_shop/ — база микроконтроллеров, датчиков и модулей.
- https://www.chipdip.ru — магазин радиодеталей.

14. Contacts

Tg: https://t.me/MrRyabena
Mail: daniilrazanov82349@gmail.com

I will be glad to receive suggestions for improving the project, cooperation, and feedback.

Daniil Ryazanov (Даниил Рязанов)

Заключение

Smart Home System — масштабный проект, предлагающий удобные решения и инструменты в современной реализации. В скором времени пользоваться разработками будет удобно и легко даже пользователю, не владеющему навыками программирования и проектирования.

У проекта большой простор для развития и расширения. Впереди полно работы и многие инструменты недоработаны, но уже сейчас на базе ***Smart Home System*** можно создавать автоматизированные системы.

Подготовив линейку модулей и пользовательские интерфейсы, можно наладить коммерческое производство модульных систем.

Многие инструменты в проекте поддерживают индивидуальную работу, поэтому ***Smart Home System*** можно использовать в отдельных проектах и устройствах, ускорив процесс их создания.

Источники информации

1. https://alexgyver.ru
2. Петин В. А. Создание умного дома на базе Arduino. — М.: ДМК Пресс, 2018. — 180 с.
3. Neil Cameron. Electronics Projects with the ESP8266 and ESP32: Building WebPages, Applications, and WiFi Enabled Devices. Apress; 1st ed. edition (December 18, 2020)
4. https://amperka.com