# SmartHomeSystem

/\*  
 \_\_\_\_\_ \_ \_ \_ \_\_\_\_\_ \_   
 / \_\_\_\_| | | | | | | / \_\_\_\_| | |   
 | (\_\_\_ \_ \_\_ \_\_\_ \_\_ \_ \_ \_\_| |\_ | |\_\_| | \_\_\_ \_ \_\_ \_\_\_ \_\_\_ | (\_\_\_ \_ \_ \_\_\_| |\_ \_\_\_ \_ \_\_ \_\_\_   
 \\_\_\_ \| '\_ ` \_ \ / \_` | '\_\_| \_\_| | \_\_ |/ \_ \| '\_ ` \_ \ / \_ \ \\_\_\_ \| | | / \_\_| \_\_/ \_ \ '\_ ` \_ \   
 \_\_\_\_) | | | | | | (\_| | | | |\_ | | | | (\_) | | | | | | \_\_/ \_\_\_\_) | |\_| \\_\_ \ || \_\_/ | | | | |   
 |\_\_\_\_\_/|\_| |\_| |\_|\\_\_,\_|\_| \\_\_| |\_| |\_|\\_\_\_/|\_| |\_| |\_|\\_\_\_| |\_\_\_\_\_/ \\_\_, |\_\_\_/\\_\_\\_\_\_|\_| |\_| |\_|   
 \_\_/ |   
 |\_\_\_/   
\*/

# Note

### Проект в разработке, архитектура ядра, документация и концепции постоянно меняются и дорабатываются. Полная картина будет понятна ближе к релизу.

*Ожидаемая дата релиза:* 05.03.2024

# Versions

* v1.0.0 — private beta-version.
* **v1.1.0 — [current version] development.**

### Announcements

* v1.2.0 — release.
  + New core architecture.
  + New library.
  + Documentation will be updated.

SHScore scheme

# Содержание

* [1. Введение](#X0fc7b51b76a3811f59fc154b037076819f8ec21)
  + [1.1 Микроэлектроника уже повсюду](#X18f8e09909c244579e68bb64434896ab176174d)
  + [1.2 Актуальность и обоснование проекта](#X1a07b1e8b7cf147424a21a70efe70d7095b531c)
  + [1.3 Цели и задачи](#X740a6a02ecbbd24650d20672680974cc69f19f2)
* [2. Обзор литературы](#Xac18bfc6f562edcc949fa7e9e7def7e7e7e2473)
  + [2.1 Alex Gyver](#Xa3c80498e2e696e0628acfffa36e7e0c239b106)
  + [2.2 Виктор Петин](#X1752a7421514dfd07dbf92a86c428b657b436a7)
  + [2.3 Нил Кэмерон](#X1bbd7d28c7e0d106e3048024987837f1eb0b2d6)
* [3. Структура проекта](#Xd1515fdd2df3a8208eb343f797bb9d8ca6e4b12)
* [4. Методы и этапы проектирования](#Xad27a615ded7509a429cd9f03805e5c3fd68efb)
  + [4.1 Принцип работы](#X6e55c454ad3b417207a1eb7acdb1505fb29aacb)
  + [4.2 Этапы разработки](#X0d60b8369b6b4678b42c2a51785a0b08b40a125)
  + [4.3 Подход к проектированию](#X977cd00c04872da1f26de32a4b395356a8affdd)
  + [4.4 Функции](#X6993d4696562524fb3bd466611340fc77e86e2e)
  + [4.5 Структура Smart Home System](#Xad2d0813ca9c959543b7ee05557b37282cd0b15)
* [5. Выбор компонентов](#X06fc230388586df3bd3330df82ba996adb5ac19)
  + [5.1 Микроконтроллеры](#X5ec25a923058421349b59011503cca01a2f289b)
    - [5.1.1 Atmega328P](#X86663bdbb256f8a8ca3b106b6f1d93ec593331b)
    - [5.1.2 ESP8266](#X2deb81666bef4b371755611965b50e7afd9d06a)
    - [5.1.3 ESP32](#Xeadafeeb2859996163061f1941afb86e8d2e322)
  + [5.2 Датчики](#X48e62b97f3bf221cb8921dbaa3302a7bcf5e21d)
    - [5.2.1 Temperature](#Xdd833b1057df29e233f6e183c2574381eaf1129)
    - [5.2.2 Humidity](#X0aa4e877d4dd6ac139f5e2b713fbdb7386ef0ce)
    - [5.2.3 Pressure](#X78e740edd5ad7e5724c5fe25fbd8c2ae24b6e05)
    - [5.2.4 Carbon dioxide](#Xf25a84bb90e3409b78b3fa7062c5a74827fa80b)
    - [5.2.5 Illumination](#X072ee3ebfb7664f76f419477bb871806572f6bf)
    - [5.2.6 Other](#X8d6d5c085705d79f29c74ee8ca55f4b57671e5d)
  + [5.3 Силовые компоненты](#X701d5241a2ff1abc227a018c109a48505fae65c)
    - [5.3.1 Транзисторы](#Xe51a748f238c81ae51b006c7106d6506dab44db)
    - [5.3.2 Симисторы](#Xd89a4e9262e736a14fcd9dc9dd7d58ebf4a352c)
  + [5.4 Обвязка](#X3063105963fc4f3cfb5de6be08b0ba244f3b3d6)
    - [5.4.1 Резисторы](#X6ce1fa0c1b347b9a7aa1bcd85b27b8dcb20ec23)
    - [5.4.2 Конденсаторы](#X634a1e94c77a4086b63318dfd43390e750ce461)
    - [5.4.3 Диоды](#X06e1f1ffcfb54d2ada414445c79749af21e1962)
    - [5.4.4 Стабилитроны](#X082bdea8769e56f25558019ca4be6f0714091a7)
    - [5.4.5 Стабилизаторы напряжения](#X6c8160234c1941cad058276338cb2b7fdddf810)
    - [5.4.6 Оптопары (оптроны)](#X54262cb5ea0052168d11820d2154c30613e6cc2)
    - [5.4.7 Регистры](#Xe52889c3d91628bc7beca99530cb5999915f2a2)
* [6. Этап I](#X49523ec1d66c1adbc6ba370728f4b69fbafa1d2)
  + [6.1 Module](#X3e8fb88bac90a76e1142c96fd4f245c4789dcda)
  + [6.2 Управление нагрузкой](#Xebc854a246628e03ebeb2e239c4600550af21e5)
    - [6.2.1 Силовые схемы](#X054fb7c15c86c4631b1cb5d7ab0633163b9b4a5)
    - [6.2.2 Режим SWITCH](#X8d37f2eebfe8b1939113f479b1d8224a092431f)
    - [6.2.3 Режим PWM](#Xb3ed4fad6b57457bb5dea3c249e7f7a8d819273)
    - [6.2.4 Режим DIMMER](#X393b67813b6a0f94f3f6d7fe1369404f57d0b9b)
    - [6.2.5 Режим PID](#X98fd6aadcb66efdad3fb93d84cc24e688104181)
  + [6.3 Датчики](#X8421474a7ac45b508bbfb85ebb1bf89b5e66468)
    - [6.3.1 Кнопка, энкодер, потенциометр](#Xc7e2d0a8604a0fe7226873d0661fb721dc9e127)
    - [6.3.2 Термистор и фоторезистор](#X4a3e0e0edf97410c711f7feb725972fc4ae7d32)
    - [6.3.3 Термопара](#X4af0da62490c3cef4b6726ec2d638886440974c)
    - [6.3.4 DS18B20](#X334bd6192ed181c42bbdd93b05399b639c2ef0d)
    - [6.3.5 HTU21D](#Xf517137c0638c0a357e16d7d283840f286f82ee)
    - [6.3.6 BME280/BMP280](#Xc327ab24a3430f21d9e27164ce1c3ce886dbb8c)
    - [6.3.7 MH-Z19b](#Xe49634ee22ef87007b1837aa314abdb91c1efea)
    - [6.3.8 HX711](#X7bcb954142f792ce6d5e1db9b4ea71ca14e667b)
* [7. Этап II](#Xb7773dac18a4a968ca3d7e6eb78392a880116f1)
  + [7.1 Containers](#Xefdb2997fdcee65ccb23016c22178231575da01)
    - [7.1.1 ByteCollector](#X1dd6e4e4189529e76a961f6918faf6443e03023)
    - [7.1.2 Handlers](#X0828997ca6082b23a557c4890cb5be6e9e4aed9)
  + [7.2 Protocols](#Xa2a4e536904c9886657a98725731b2308fc4b5e)
    - [7.2.1 SHSDTP](#X9821fd6f03bcf35bcf3f9603da57acbd93eda6b)
    - [7.2.2 API](#X26d0c1d7f0adf503fe23ecb0b6604ccf5f97bd2)
    - [7.2.3 SHSF](#Xfd2a7ef9a9fdda12b0a3f1448a4b86e3812f15a)
    - [7.2.4 TCP/IP](#X3157524841b392e66f7c19e82970b9e58a1499d)
  + [7.3 System](#X2573be6b008e4102067989d20fba526009f8f27)
    - [7.3.1 Process](#X37ac99007c1ab7869d6a37c99a1d483afa29721)
    - [7.3.2 Sensor](#X39bde34eab655b89dbcab37fccc45e014a73759)
    - [7.3.3 Load](#X8a017fdff249c55fdf7a5baaa569e178c16a48f)
    - [7.3.4 ErrorsHandler](#Xb8e288267bf34a1021b7287944ddbd16b898ba9)
    - [7.3.5 Settings](#Xa57a3ef5b5633291184abef1aaa0c7c99b6a292)
  + [7.4 Algorithms](#Xe28a42f0195d3da128349c600296ec2831143bf)
    - [7.4.1 CRC](#Xd6f23d81f8bcb8879b8568606159b3ef3ebc334)
* [8. Этап III](#X27de92ccf2ab8733ebb42cb1d7fe42e0fa4ee86)

development

* [~~7. Этап II~~](#stageII)
  + [~~7.1 Организация соединения~~](#connection-organization)
    - [~~7.1.1 Покрытие сети~~](#network-coverage)
    - [~~7.1.2 Подключение модулей~~](#connecting-modules)
    - [~~7.1.3 TCP/IP~~](#X099c72975b37a1bc7bae8dc97f11b73eb139ddb)
* [~~8. Этап III~~](#stageIII)
* [~~9. Этап IV]~~(#stageIV)
* [~~10. Этап V~~](#stageV)
* [Версии](#versions)
* [Источники информации](#sources)

# 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 широкая 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](https://AlexGyver.ru). На его сайте есть много 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’a. Вместе со своим помощником(ами) Alex создал базу легких, оптимизированных и качественно сделанных библиотек, которые используются в ***Smart Home System***: [GyverLibs](https://github.com/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>.

### ~~Структура изменена, здесь будет поправлекна чуть позже!~~

### Структура может изменяться

* [libraries](libraries/) — сторонние библиотеки, используемые в проекте.
* [presentation](presentation/) — презентации проекта и вспомогательные файлы.
* [schemes](schemes/) — картинки, схемы, чертежи...
* [SHSlogos](SHSlogos/) — логотипы и символика проекта.
* [src](src/) — ПО проекта.
  + [debugging\_sketches](src/debugging_sketches/) — наброски отладочных прошивок.
  + [SHSapp](src/SHSapp/) — приложение под windows.
  + [SHScore](src/SHScore/) — ядро (бизнес-логика), основная разработка.
  + [SHSlibrary](src/SHSlibrary/) — библиотека с удобными инструментами, основанная на ядре.
    - [SHSnerualnet](src/SHSlibrary/SHSnerualnet/) — разработка нейросети и алгоритмов работы с ней.
  + [SmartModules](src/SmartModules/) — [beta] устройства и модули.
  + [SmartModulesAPI](src/SmartModulesAPI/) — [beta] команды устройств и модулей.
* [synchronizer](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 (VСС, 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. Так, 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 V. 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 много 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.

## 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 света. 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, эмитирую восход солнца. Задача достаточно простая и реализуется даже без дополнительных библиотек.
* Если уж речь пошла об освещении, то можно добавить красоты и технологичности — RGB-подсветку. Тут необходимы инструменты для работы с **RGB** (обычные четырезвыводные, где требуется только регалировать яркость каждого канала цвета) и **ARGB** (адресные светодиоды, позволяют управлять цветом каждого светодиода, независимо от всех остальных) светодиодами. Если первые все еще можно ковырять 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 важности и концентрацию углекислого газа. Обрабатывая их, можно настроить регулировку котла или подачи горячей воды, работу увлажнителей воздуха и открывать форточки (или включать систему вентиляции).
* Метеостанция. Измерение температуры, влажности, скорости и направления ветра, атмосферного давления. Обработка результатов и составление прогноза погоды на ближайшее время, рекомендаций по одежде и ожиданий 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 функионал:

* 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, орабатывает данные.
  + SmartHomeBot — реализован в серевере, 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 маркойArduino, 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 где нужна компактность), либо платы Arduino Nano, Arduino Pro Mini.

|Parametr|Value| |:-|:-| |CPU type|8-bit AVR| |CPU speed|16 MHz| |Сurrent 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| |Operating Voltage Range|1.8 — 5.5 V|

ATmega328P pinout

### 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| |Сurrent consumption|300 mA| |Flash memory|1-16 MB| |SRAM|82 KB| |EEPROM|4 KB| |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|

EPS8266 pinout

### 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| |Сurrent 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|

ESP32 pinout

## 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 уже писалось 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](https://alexgyver.ru/lessons/arduino-load/), 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-1000С** — 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 c 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

В этом этапе рассказаны принципы построения системы умного дома. В ***Smart Home System*** алгоритмы управления уже реализованы в ядре и библиотеке (см. [Этап II](#Xb7773dac18a4a968ca3d7e6eb78392a880116f1) и [Этап III](#X27de92ccf2ab8733ebb42cb1d7fe42e0fa4ee86)), 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](SHSlibrary/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 Управление нагрузкой

### 6.2.1 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 используется 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.*
* transistor-scheme
* 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.
* bc547-scheme
* 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.*

relay-scheme

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

### 6.2.2 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);

### 6.2.3 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](https://github.com/GyverLibs/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](https://alexgyver.ru).

#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);

### 6.2.4 Режим DIMMER

Необходим для плавного управления нагрузкой переменного тока.

Удобнее всего использовать библиотеку [GyverDimmer](https://github.com/GyverLibs/GyverDimmer). *Лучше использовать алгоритм Брезенхема.*

### 6.2.5 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](https://github.com/GyverLibs/GyverPID).

## 6.3 Датчики

### 6.3.1 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](https://github.com/GyverLibs/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;

### 6.3.2 Термистор и фоторезистор

Для преобразования показаний **термистора** используется уравнение Стейнхарта—Харта. Можно воспользоваться готовой библиотекой [GyverNTC](https://github.com/GyverLibs/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.

### 6.3.3 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](https://github.com/GyverLibs/GyverMAX6675).

### 6.3.4 DS18B20

Достаточно точный цифровой датчик температуры. Библиотека: [microDS18B20](https://github.com/GyverLibs/microDS18B20)

### 6.3.5 HTU21D

Точный датчик температуры и влажности воздуха. Для работы с ним есть легкая библиотека [GyverHTU21D](https://github.com/GyverLibs/GyverHTU21D).

### 6.3.6 BME280/BMP280

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

Удобная библиотека: [GyverBME280](https://github.com/GyverLibs/GyverBME280).

### 6.3.7 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](https://habr.com/ru/articles/391157/), позже в проекте появится своя библиотека.

### 6.3.8 HX711

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

Библиотека: [GyverHX711](https://github.com/GyverLibs/GyverHX711).

# 7. Этап II

[SHScore](src/SHScore/) — логика работы системы, основанная на парадигме **ООП** и принципах **SOLID**. В ядре описаны абстрактные классы для оперируемых системой объектов и процессов, а так же обработчики для них, протоколы связи и передачи данных. Ядро дискретизирует и описывает действия системе, 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 не большое количество оперируемых объектов. Зато, ООП позволяет организовать систему очень удобно, быстро и без лишнего дублирования кода.

SHScore-scheme

## 7.1 Containers

### 7.1.1 ByteCollector

Для удобной упаковки данных в массив байтов и дальнейшей их обработки разработан легкий класс: [SHSByteCollector.h](SHSlibrary/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);  
 ~ByteCollector();  
  
 /\*  
 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](src/SHScore/SHSProcessesKeeper.h) — 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](src/SHScore/SHSCallbacksKeeper.h) — 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](src/SHScore/SHSSensorsKeeper.h) — 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](src/SHScore/SHSLoadKeeper.h) — 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](https://github.com/GyverLibs/GyverBus).

Для отправки данные нужно обработать, а потом распаковать обратно, для этого создан [SHSdtp.h](SHSlibrary/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, с тремя обязательными функциями-членами:

class Stream  
{  
 uint8\_t write(const uint8\_t \*buf, size\_t size);  
 uint8\_t read();  
 uint8\_t available();  
};

В конструкторе DTP принимает указатели на объект Stream и функцию-обработчик входящих пакетов.

/\*  
 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:  
 #include <Arduino.h>  
 or write your class:  
 uint8\_t Stream::write(const uint8\_t\* buf, size\_t size);  
 uint8\_t Stream::read();  
 uint8\_t Stream::available();  
  
\*/  
namespace settings  
{  
#ifndef SILENCE\_TIMEOUT  
#define SILENCE\_TIMEOUT 120000  
#endif  
  
#define DTP\_OFFSETbeg 5  
};  
  
enum shs::DTPcommands : uint8\_t  
{  
 answer = 252,  
 error,  
 request,  
};  
  
struct shs::DTPdata  
{  
 uint8\_t to{};  
 uint8\_t from{};  
 int16\_t apiID{};  
 uint8\_t datasize{};  
 shs::ByteCollector &data{};  
};  
  
class shs::DTPpacker  
{  
  
public:  
 uint8\_t packDTP(shs::ByteCollector \*bc, const uint8\_t to);  
 uint8\_t packDTP(shs::ByteCollector \*bc, const uint8\_t to, const uint8\_t from);  
 uint8\_t checkDTP(shs::ByteCollector \*bc);  
 uint8\_t parseDTP(shs::ByteCollector \*bc, shs::DTPdata &data);  
};  
  
class shs::DTP : public shs::DTPpacker, public shs::CallbacksKeeper  
{  
public:  
 explicit DTP(Stream \*bus);  
 ~DTP();  
  
 inline uint8\_t tick();  
 uint8\_t checkBus();  
  
 inline uint8\_t sendPacket(shs::ByteCollector \*bc, const uint8\_t to);  
 uint8\_t sendPacket(shs::ByteCollector \*bc, uint8\_t to, uint8\_t from);  
  
 template <typename T>  
 uint8\_t sendRAW(const T &data, const uint8\_t to);  
};

### 7.2.2 API

В файле [API.h](src/SHScore/SHSAPI.h) описан абстрактный класс для реализации API. В handler() передается пакет данных, который нужно расшифровать и отправить, если требутся, ответ. Можно реализовать управление отдельным датчиком, нагрузкой или библиотекой. Таким образом в ***Smart Home System*** всегда возможно интегрировать любое устройство, достаточно реализовать для него API и процессы его поведения.

class shs::API  
{  
public:  
 API(const shs::settings::shs\_ID\_t ID = 0);  
  
 void setID(const shs::settings::shs\_ID\_t ID);  
 shs::settings::shs\_ID\_t getID() const;  
  
 virtual uint8\_t handler(shs::ByteCollector &data) = 0;  
  
protected:  
 shs::settings::shs\_ID\_t m\_ID{};  
};

### 7.2.3 SHSF

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

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

### 7.2.4 TCP/IP

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

[SHSTcpClient.h](src/SHScore/SHSTcpClient.h) и [SHSTcpServer.h](src/SHScore/SHSTcpServer.h) упрощают процесы подключения и опрса входящих потоков.

// SHSTcpClient.h  
class shs::TcpClient : WiFiClient  
{  
public:  
 inline void tick();  
};  
  
  
// SHSTcpServer.h  
class shs::TcpServer  
{  
public:  
 WiFiServer \*server;  
 WiFiClient \*clients;  
 shs::DTP \*dtp;  
  
 const uint8\_t \*IP{};  
 uint8\_t maxClients{};  
  
 TcpServer(const uint8\_t \*IPaddress, void (\*TCPhandle)(shs::DTPdata &), uint16\_t port = 50000, uint8\_t max\_clients = 6);  
 ~TcpServer();  
  
 inline void begin();  
 inline void tick();  
  
 uint8\_t sendPacket(shs::ByteCollector \*col, uint8\_t id);  
 void sendRAW(uint8\_t \*buf, uint8\_t size);  
}

## 7.3 System

## 7.3.1 Process

[SHSProcess.h](src/SHScore/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 поцесса.

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](src/SHScore/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](src/SHScore/SHSSensorAnalog.h)) реализованы методы считывания значений с аналоговых выводов микроконтроллера и подсчет среднего значения из выборки измерений.

На основе этих классов в [SHSlibrary](src/SHSlibrary/) разрабатываются классы для конкретных моделей датчиков, расширяя набор поддерживаемых устройств.

// SHSsensor.h  
  
enum shs::SensorType : uint8\_t  
{  
 unknown,  
 analogPin,  
 thermistor,  
 photoresistor,  
};  
  
class shs::Sensor  
{  
public:  
 shs::SensorType type{};  
  
public:  
 explicit Sensor(const shs::settings::shs\_ID\_t ID = 0,   
 const shs::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() = 0;  
 virtual shs::settings::shs\_float\_t getValueF() = 0;  
 virtual shs::settings::shs\_double\_t getValueD() = 0;  
  
 virtual int16\_t getAverageI() = 0;  
 virtual shs::settings::shs\_float\_t getAverageF() = 0;  
 virtual shs::settings::shs\_double\_t getAverageD() = 0;  
  
protected:  
 shs::settings::shs\_ID\_t m\_ID{};  
};  
  
// SHSSensorAnalog.h  
  
class shs::SensorAnalog : public shs::Sensor  
{  
public:  
 explicit SensorAnalog(const shs::settings::shs\_ID\_t ID = 0,   
 const shs::SensorType stype = shs::SensorType::unknown, const uint8\_t pin = A0);  
  
 void setup() override;  
 shs::settings::shs\_float\_t getValueF() override;  
 shs::settings::shs\_double\_t getValueD() override;  
 int16\_t getValueI() override;  
  
 shs::settings::shs\_float\_t getAverageF() override;  
 shs::settings::shs\_double\_t getAverageD() override;  
 int16\_t getAverageI() 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](src/SHScore/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 полнотью выключить.
* PWM — можно управлять мощностью нагрузкив сети постоянного тока с помошью ШИМ.
* DIMMER — аналог ШИМ для сети перенного 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 : uint8\_t  
{  
 UNKNOWN,  
 SWITCH,  
 PWM,  
 DIMMER,  
};  
  
class shs::Load  
{  
public:  
 shs::LoadType type;  
  
public:  
 explicit Load(shs::settings::shs\_ID\_t ID = 0, shs::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) = 0;  
 virtual void on(const uint16\_t value = UINT16\_MAX, const uint16\_t smoothing = 0) = 0;  
  
 virtual void off(const uint16\_t smoothing = 0) = 0;  
  
protected:  
 shs::settings::shs\_ID\_t m\_ID{};  
};

### 7.3.4 ErrorsHandler

В класс shs::ErrorsHandler ([SHSErrorsHandler](src/SHScore/SHSErrorsHandler.h)) передаются коды ошибок, которые обрабатываются подключенными обработчиками.

### 7.3.5 Settings

Все настройки обернуты в namespace shs::settings. Файл [SHSsettings.h](src/SHScore/SHSsettings.h) является пользовательским шаблоном настроек. Там можно разово задать общие параметры для всей системы. В файл [SHSsettings\_private.h](src/SHScore/SHSsettings_private.h) рекомендуется не вносить изменения, во избежание некорректной работы системы.

## 7.4 Algorithms

### 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*](src/SHScore/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) {}  
  
 inline void add(const uint8\_t value);  
 inline void clear();  
  
 void update(uint32\_t &crc, uint8\_t data);  
 uint32\_t crcBuf(const uint8\_t \*ptr, uint16\_t size);  
};

# 8. Этап III

[SHSlibrary](src/SHSlibrary/) — следующий уровень 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 представлены классы [SHSModule](src/SHSlibrary/SHSModule.h), хранящий шаблон конфигурации модуля и [SHSAutoProcesses](src/SHSlibrary/SHSAutoProcesses.h), позволяющий выполнять автоматические конфигурацию и управление модулем.

namespace shs  
{  
 namespace settings;  
  
 class Module;  
 extern Module module;  
};  
  
class shs::Module  
{  
public:  
 GyverNTP ntp(3, 1800);  
 shs::Config config;  
 shs::ErrorsHandler errorsHandler;  
 shs::ProcessesKeeper processes;  
 shs::SensorsKeeper sensors;  
 shs::LoadKeeper load;  
};  
  
class shs::AutoProcesses : public shs::Process  
{  
public:  
 WiFiClient tcp;  
 shs::DTP dtp;  
 shs::ProcessesKeeper keeper;  
  
 explicit AutoProcesses();  
 ~AutoProcesses();  
  
 void begin() override;  
 void tick() override;  
 void end() override;  
};

# Этап VI

## Организация соединения

### .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 не сложная. 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.

### .1.2 Подключение модулей

После настройки маршрутизатора, остается подключить и настроить сами микроконтроллеры. Для этого в [SHScore](src/SHScore/) есть набор функций, реализованных в файле [SHSconnectWiFi.h](src/SHScore/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](SHSlibrary/SHSsettings.h).

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

## .2 Передача данных

### .2.3 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 [SHSlibrary](SHSlibrary/) есть свой клаcc, но необходимости в нем на данном этапе проекта нет, в отличии от [SHSTcpServer.h](SHSlibrary/SHSTcpServer.h), который отвечает за обработку всех клиентов.

class shs::TcpServer  
{  
public:  
 WiFiServer \*server;  
 WiFiClient \*clients;  
 shs::DTP \*dtp;  
  
 const uint8\_t \*IP{};  
 uint8\_t maxClients{};  
  
 TcpServer(const uint8\_t \*IPaddress, void (\*TCPhandle)(shs::DTPdata &), uint16\_t port = 50000, uint8\_t max\_clients = 6);  
 ~TcpServer();  
  
 void begin();  
 void tick();  
  
 uint8\_t sendPacket(shs::ByteCollector \*col, uint8\_t id);  
 void sendRAW(uint8\_t \*buf, uint8\_t size);  
  
private:  
 void (\*\_TCPhandle)(shs::DTPdata &){};  
 uint8\_t \*lens{};  
 uint8\_t i{};  
};

## .3 API

Все устройства связаны и имеют доступ друг к другу. Чтобы они могли запрашивать и принимать данные, нужно определить соотвествующие команды и обработчики для них. API каждого модуля состоит из перечисления(enum), где каждой команде соответствует численный 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.

API всех устройств собраны в [SmartModulesAPI](SmartModulesAPI/).

# . Этап IV

В Smart Home System реализованы четыре метода взаимодействия с пользователем:

1. Кнопки и датчики.
2. Чат-боты.
3. Приложение.
4. Голосовое управление.

Каждый из них позволяет изменять набор параметров и регулировать систему.

## .1 Кнопки и датчики

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

Кнопки лучше всего расположить в частодоступных местах: у входа 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.

## Telegram bot

[@SHS\_sbot](https://t.me/SHS_sbot)

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

Smart Home Bot scheme

Лежит на сервере и имеет быстрый доступ к информации хранилища и управлению Smart Home System. Для реализации есть две библиотеки [FastBot](https://github.com/GyverLibs/FastBot) и [FastBot2](https://github.com/GyverLibs/FastBot2) (бот потихоньку переезжает на вторую), планируется расширение SHStbot с дополнительным функционалом.

## Приложение

Приложение является очень мощным инструментом. В нем собраны все настройки и кнопки управления, сценарии поведения, состояния всех модулей, графики разных параметров и визуализация работы системы. Разработка находится в папке [SHSapp](SHSapp/).

## Голосовое управление

Построено на специальном контроллере распознавания голоса.

## 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 миктроконтроллере, т.к. компьютер нет смысла гонят впустую (не выключать) из-за какого-то процесса умного дома, а другие модули заняты своими вычислениями и их мощности будет маловато.

### Основные требования к серверу

* Поддерживать одновременное подключение нескльких 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 неприрывную работу.
* Иметь возможности реализовать функции:
  + Telegram-bot

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

# 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)