Муниципальное бюджетное общеобразовательное учреждение
средняя общеобразовательная школа №1
с углубленным изучением английского языка
города Ковдор Мурманской области

Региональный этап Всероссийского научно-технологического конкурса проектов «Большие вызовы»

Проектная работа

Кодовый замок на Arduino

Автор: **Смирнов Максим Анатольевич**,
обучающийся 11 «А» класса МБОУ СОШ № 1
Руководитель: **Маркив Юлия Николаевна**,
учитель информатики МБОУ СОШ № 1

2019-2020
учебный год

Содержание

Введение

С появлением частной собственности возникла необходимость оберегать ее от чужих посягательств. Для этого и были придуманы двери и замки. В современном мире кодовые замки широко применяются на предприятиях, например, в помещениях с ограниченным доступом, а также на различных сейфах и шкафах.

План исследования

Моя увлеченность сборкой небольших электронных схем привела меня к желанию изучить аппаратную платформу Arduino. Мне интересно этим заниматься, поэтому я решил попробовать себя в создании чего-то более серьезного.

Объект исследования – промышленные кодовые замки.

Предмет исследования – принципы работы промышленного кодового замка.

Цель работы: Создание функционального аналога промышленного кодового замка с использованием аппаратно-программной платформы Arduino.

Задачи:

1. Изучить принципы работы кодовых замков.
2. Изучить концепцию программирования на Arduino.
3. Создать рабочий макет устройства аналога промышленного кодового замка.
4. Сделать выводы.

Методы исследования:

- ✓ анализ и синтез;
- ✓ обобщение;
- ✓ эксперимент;
- ✓ моделирование.

1. История создания кодовых замков

Замки для входных дверей и соответственно ключи к замкам появились у ранних цивилизаций, о чем можно прочитать в древних мифах. Есть упоминания о дверных замках даже в Ветхом Завете.

Со времени древних римлян, греков, египтян и китайцев до наших дней дошли висячие замки. Предполагается, что они впервые использовались в качестве дорожных замков, чтобы ограждения товара на древних торговых дорогах от нападений разбойников. Они производились всяческих размеров, от самых маленьких до громадных, и в форме всевозможных геометрических фигур, животных, рыб, птиц. Отворить висячий замок можно было при использовании ключей, каковые поворачивались, ввинчивались, вытягивались и нажимались. Для гарантии более надежной защиты использовались замки с применением букв – кодовые замки, открыть которые, можно было, лишь нажав на определенные буквы или цифры.

Изобретателями первых "секретных" замков являются мастера востока. Такие замки обладали определенным количеством колец, выстроив которые особым образом можно было открыть замок. Наборные замки обладали похожим принципом работы. Открыть такие замки можно было, только зная определенное слово или набор цифр. Один из таких замков «Эврика» запирал сейф казначейства США. Число цифр и букв на нем давало возможность набора 1 073 741 824 комбинаций. Поэтому, открыть его было не реально.

Замок с тумблерными штифтами изобрели отец и сын Ейлс. В 1848 году Линус Ейлс-старший, уроженец Миддлтона, Коннектикут, заинтересовался банковскими замками и адаптировал египетский механизм тумблерных штифтов к современным условиям. В первых моделях тумблеры были встроены в корпус замка, который открывался круглым нарезным ключом. Линус Ейлс-младший создал цилиндрический тумблерный штифт небольших размеров и несколько типов ключей, сначала плоских, позже рифленых, уступивших место парацентрическим, которые используются и ныне. Сейчас во многих странах мира производят замки с тумблерными штифтами одинаковых размеров и конструкций.

Британец Роберт Баррон в 1778 году внес большой вклад в развитие конструкции замков. Он понимал, что защитные выемки ненадежны, и замки, созданные по такому принципу, не обеспечивают полноценной защиты. Баррон создал замок с двумя откидными тумблерами, которые действовали согласованно. Чтобы отодвинуть язычок замка, оба тумблера нужно было поднять на строго определенную высоту. Шесть лет спустя Джозеф Брама,

изобретатель из Йоркшира, запатентовал свой знаменитый замок, который теперь носит его имя. В техническом отношении он сыграл большую роль в развитии замков, хотя коммерческим успехом он не пользовался. Считается, что это первый замок с наконечником или цилиндром и коротким ключом, который непосредственно не касался язычка замка, а отпирал его, приводя в движение вспомогательные детали. Механизм замка состоял из набора скользящих пластинок, которые при повороте ключа вставали в определенное положение и сдвигали язычок.

В истории замков Ч.Чабб известен как изобретатель детекторного замка и рычажных замков высокого качества и повышенной защиты, которые используются вот уже 140 лет.

Справедливости ради стоит отметить, что первый образец кодового замка был создан намного раньше – в XVI веке. Его автором был Джероламо Кардано, сын друга Леонардо Да Винчи.

2. Кодовый замок

Кодовый замок - замок, для открытия которого необходимо ввести с клавиатуры, выставить определённым образом на специальных цилиндрах или иным образом указать кодовую последовательность, которую знает только владелец. В целом, любой замок предназначен для предотвращения доступа в помещение посторонних лиц, или наоборот, ограничения выхода из помещения. Кодовые замки обладают некоторыми ***достоинствами*** перед обычными замками. Во-первых, отсутствует ключ, который можно потерять или который могут украсть. Во-вторых, код можно быстро поменять. В-третьих, код можно передать другому лицу, не потеряв доступ самому. Помимо преимуществ есть имеются и ***недостатки***. Во-первых, код можно забыть. Его можно записать, но при этом понижается конфиденциальность. Во-вторых, при вводе код могут подсмотреть. Поэтому при вводе нужно сохранять скрытность. В-третьих, в некоторых ситуациях в качестве кода указывают даты рождения или общеизвестные числа. Это значительно понижает надежность кода.

Основные разновидности кодовых замков:

- ✓ Механические
- ✓ Электронные
- ✓ Комбинированные

3. Электронный замок

Электронный замок - электронное устройство, в котором кодовая комбинация хранится в памяти электронного блока и вводится обычно с различных датчиков. Это могут быть клавиатуры, сканеры отпечатков пальца, системы дистанционного управления, считыватели магнитных карт, «домофонных» ключей, штрих-кодов и т.п.

В качестве ***исполнительных механизмов*** используются электромеханические и электромагнитные запирающие устройства. Например, электромагнитные щеколды, шпингалеты и т.п.

Замки такого типа обычно состоят из 4 основных частей:

1. Запирающий механизм. Чтобы он открывался и закрывался, необходимо подать на него электрический ток, а именно кратковременный импульс. При соответствии введенного либо направленного кода правильной комбинации замок откроется.

2. Считывающее устройство. С его помощью устройство принимает значения (считывает код) и направляет в блок управления.

3. Основной блок управления. В нем осуществляется проверка введенного кода на соответствие правильной комбинации. Если введенный набор цифр соответствует заданной комбинации – дверь откроется, в обратном случае – останется закрытой.

4. Источник питания, который будет обеспечивать постоянную работу устройства. Замок может питаться от сети, но есть варианты с резервным источником питания. Они позволяют устройству работать независимо от наличия или отсутствия электроэнергии в общей сети. Это дает возможность пользоваться замком некоторое время после отключения электроэнергии.

4. Arduino

Arduino — торговая марка аппаратно-программных средств для построения простых систем автоматики и робототехники, ориентированная на непрофессиональных пользователей. Это может использоваться как для создания автономных объектов автоматики, так и для подключения к программному обеспечению на компьютере через стандартные проводные и беспроводные интерфейсы.

4.1. Аппаратная часть

Под торговой маркой Arduino выпускается несколько плат с микроконтроллером. Большинство плат снабжены минимально необходимым набором обвязки для нормальной работы микроконтроллера (стабилизатор питания, кварцевый резонатор, цепочки сброса и т. п.). Также, выпускаются специальные платы расширения (так называемые «шилды»). Они созданы для удобства работы с Arduino.

Некоторые модели микроконтроллерных плат:

В концепции Arduino не предусмотрен корпусный или монтажный конструктив. Автор проекта самостоятельно выбирает способ установки конструкции и его механической защиты. Сторонние производители выпускают различные наборы робототехники и электромеханики, ориентированные на работу совместно с Arduino. Также, выпускается большой ассортимент всевозможных датчиков и модулей в той или иной степени совместимых с Ардуино. Аппаратная часть представляет собой набор смонтированных печатных плат, продающихся как официальными, так и сторонними производителями.

Микроконтроллер

В микроконтроллерах Arduino уже имеется предварительно прошитый загрузчик. С его помощью можно осуществить загрузку прошивки в плату без использования специальных программаторов. Загрузчик соединяется с

компьютером через интерфейс USB (если он есть на плате) или с помощью отдельного переходника UART-USB.

В классической линейке устройств Arduino в основном применяются микроконтроллеры Atmel AVR. Типичные процессорные платы:

- **ATmega2560** (16 МГц, 256к Flash, 8к RAM, 54 порта, из них до 15 с ШИМ и 16 АЦП). Платы «Mega».
- **ATmega32U4** (16 МГц, 32к Flash, 2,5к RAM, 20 портов, из них до 7 с ШИМ и 12 АЦП). Платы «Leonardo», «Micro», «Yun» и др.
- **ATmega328** (16 МГц, 32к Flash, 2к RAM, 14 портов, из них до 6 с ШИМ и 8 АЦП). Платы «Uno», «Mini», «Nano», «Pro» и др.

Периферия

Порты ввода-вывода микроконтроллеров оформлены в виде штыревых контактов. Питание производится от 5 В или 3,3 В, в зависимости от модели платы. Соответственно порты имеют такой же размах допустимых входных и выходных напряжений. У плат имеются некоторые специальные возможности портов, например ШИМ (широтно-импульсная модуляция), АЦП (аналогово-цифровой преобразователь) и различные интерфейсы. Количество и возможности портов ввода-вывода определяются моделью платы.

Память

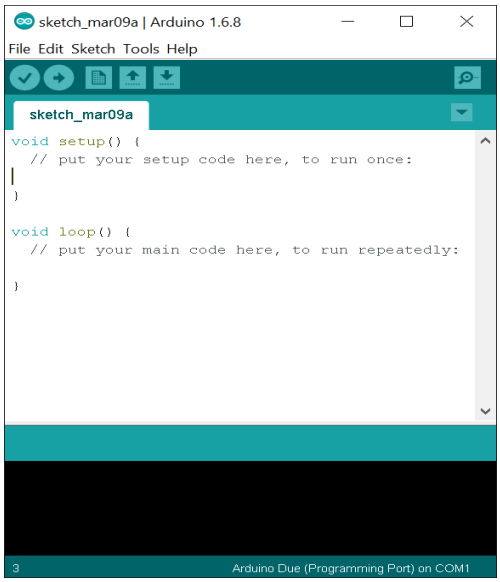
В Arduino имеется и используется несколько типов памяти. Это статическое ОЗУ**,** флэш-память **и** EEPROM (энергонезависимая память). В первом типе памяти хранятся переменные, возникшие при выполнении кода. После отключения платы (питания) вся информация в нем теряется. Второй тип памяти хранит написанные скетчи. Последний – **EEPROM** – можно использовать для сохранения информации на длительное время. Данные и значения, записанные в такую память, сохраняются независимо от питания платы.

4.2. Концепция программирования

Программирование Arduino производится на языке C++ с некоторыми особенностями, а именно на упрощенной версии этого языка. Так же существует множество библиотек, функций, классов и т.п., что значительно облегчает работу.

Для получения возможности программирования Arduino необходимо установить на компьютер интегрированную среду разработки «Arduino IDE». Она может работать в операционных системах Windows, Mac OS и Linux. Это позволяет запрограммировать плату по USB без специального оборудования

для программирования. Для возможности взаимодействия среды программирования с платой Arduino необходимо установить на компьютер специальные USB-драйверы.



Основная кнопка (слева вверху) – «**загрузка**», как понятно по названию – загружает код на плату Arduino, предварительно преобразовав его в исполнительный код. Перед тем, как загрузить код в плату, программа проверяет алгоритм на наличие ошибок. Если допущены ошибки, они высвечиваются в специальной области внизу, а также в самом алгоритме цветом выделяется строка, в которой была допущена ошибка. В этой же области при загрузке кода высвечивается данные о загрузке: успешно ли прошла загрузка, объем памяти занимаемый кодом и т.п.

Рядом с кнопкой «**загрузка**» имеется кнопка «**проверка**». Она просто проверяет правильность написания кода, не загружая код в плату.

Кнопка «**монитор порта**» (справа вверху) открывает окно последовательного интерфейса для связи с Arduino. С его помощью можно просматривать значения различных переменных во время работы платы, что поможет найти неисправность в проекте. Основная часть окна приложения занимает область для написания кода.

Файлы с кодом сохраняются в формате «.ino» и могут храниться только в папках. При сохранении файла он автоматически создает папку, в которой и находится.

4.3.Написание кода

Основные правила написания кода:

- После каждой инструкции необходимо ставить точку с запятой (;)
- Перед объявлением функции нужно указать тип данных, возвращаемый функцией или void если функция не возвращает значение.
- Перед объявлением переменной указывается ее тип.

· Комментарии обозначаются: // Строчный и /* блочный */. Это не программный код, а пояснения, помогающие понять, что происходит в коде.

Типы данных

void	0	Тип void задается при объявлении функции, которая не возвращает значения
char	1 байт	ASCII символ, например, 'A'. Char хранит информацию о коде, то есть число 66 - это 'B'. Тип хранит значения от -128 до 127
unsigned char	1 байт	ASCII символ, записанный в виде числа из интервала 0 - 255
byte	1 байт	Число из интервала 0 - 255
int	2 байта	Число от -32 768 до 32 767
unsigned int	2 байта	Число из интервала 0 - 65535
word	2 байта	Число из интервала 0 - 65535
long	4 байта	Число от -2 147 483 648 до 2 147 483 647
unsigned long	4 байта	Число из интервала 0 - 4 294 967 295
float	4 байта	Число из интервала -3.4028235 E+38 до 3.4028235 E+38
double	4 байта	В Arduino как float
string	...	Таблица символов, например, char Str1[15]; char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'}; char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', ''}; char Str4[]="arduino";
String	...	Класс String позволяет выполнять операции над текстовыми объектами.
array	...	Массив любого типа, например, int tab[10]; char text[8]="arduino";

4.3.1. Функция

Функция – фрагмент программного кода, в котором выполняются действия. В любой прошивке Arduino должны быть 2 обязательные функции – «setup» и «loop».

setup()

Выполняется в начале, всего 1 раз, при подаче питания на плату или после ее перезагрузки. Эта функция предназначена для выполнения однократных операций. Обычно в этой функции настраиваются режимы контактов платы (вход/выход), открывают необходимые протоколы связи, настраивают подключенные библиотеки, устанавливают соединения с дополнительными модулями. Даже если в коде не требуется ничего из перечисленного, функция все равно должна быть объявлена. Вот стандартный пример этой функции:

```
1 void setup() {
2  Serial.begin(9600); // Открываем serial соединение
3  pinMode(9, INPUT); // Назначаем 9 пин входом
4  pinMode(13, OUTPUT); // Назначаем 13 пин выходом
5 }
```

loop()

Выполняется сразу после функции setup(). Loop в переводе с английского означает «петля». Это означает, что функция зациклена, то есть команды внутри нее будут постоянно повторяться. Например, микроконтроллер ATmega328, который установлен в большинстве плат Arduino, будет выполнять функцию loop около 10 000 раз в секунду (если не используются задержки и сложные вычисления). Как только в этой функции завершится выполнение последней строки, программа возвращается к первой команде функции, и все повторяется.

```
void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

Здесь приведен пример программы, которая будет мигать лампочкой, подсоединенной к 13 контакту на плате Ардуино.

Условные оператор - это конструкция, обеспечивающая выполнения фрагмента кода при условии истинности некоторого логического выражения. Конструкция if ... else используется в тех случаях, когда при соблюдении условия необходимо выполнить одну часть кода, а в обратном случае – другую. Можно обойтись и без блока else, если в этом нет необходимости. Для работы можно использовать следующие сравнения: == (равно), < (меньше), >(больше), <= (меньше или равно), >= (больше или равно), != (не равно).

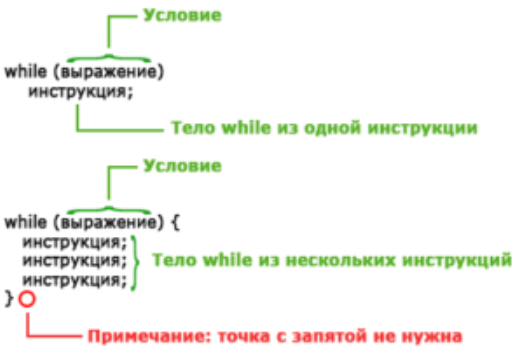
```
if (условие) { /*код*/ }

else { /*код*/ }
```

4.3.2. Цикл

Цикл - управляющая конструкция, предназначенная для многократного выполнения одного и того же набора инструкций. Использование циклов позволяет выполнять код заданное количество раз или повторять его до

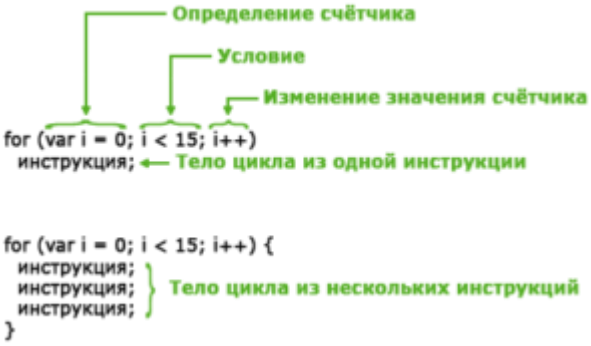
момента, пока определенное условие не изменится. Стоит отметить, что функция loop()также вызывается в бесконечном цикле.

Оператор **WHILE** выполняется до тех пор, пока не будет соблюдено определенное условие, или в то время, пока условие выполняется.



В качестве условий может использоваться любая конструкция языка, возвращающая логическое значение. Это могут быть операции сравнения, функции, константы, переменные.

Использование оператора **FOR** удобно в тех случаях, когда действия необходимо совершать определенное количество раз.



5. Макет промышленного кодового замка

На основании изученного, планируется сделать простой электронный кодовый замок на основе Arduino.

5.1. Электронная часть аналога кодового замка

Как я описывал ранее, замки такого типа обычно состоят из 4 частей:

1. Запирающий механизм. В своей версии я использую мощную электромагнитную щеколду. Её рабочее напряжение – 12 В, потребляемый ток – 1.1 А. Включение производится при помощи полевого MOSFET транзистора, т.к. с платы Arduino выходит уровень низкого сигнала – всего 5 В. В своем проекте я использую IRF3205.

При выборе транзистора самым главным параметром является пороговое напряжение на затворе – оно должно быть меньше 5 В.

Для защиты от случайных срабатываний между крайними ногами транзистора стоит резистор на 10 кОм, а между управляющим выводом Arduino и транзистором стоит резистор примерно на 100 Ом для защиты вывода платы.

2. Считыватель кода. Здесь я использую 16 – кнопочную мембранную клавиатуру для Arduino.

3. Блок управления. Как было описано ранее, в качестве основного блока будет использоваться аппаратная платформа Arduino, а именно плата Arduino UNO.

4. Источник питания. В моем случае для демонстрации работы макета был использован мощный аккумулятор на 12 В. По-хорошему можно осуществить работу замка от сети (через блок питания). Для моего макета использование аккумулятора оказалось более удобным и простым способом обеспечить питание системы.

На данном этапе я столкнулся с проблемой срабатывания щеколды. Дело в том, что данный вид запирающего механизма слишком мощный (потребляемый ток 1.1 А), и ток, проходящий через транзистор, просто не может открыть эту щеколду. Решением данной проблемы является установка конденсаторов общей суммарной емкостью приблизительно 6500 мкФ параллельно питанию щеколды. В момент включения конденсаторы накапливают заряд, а в момент открытия отдают мощный импульс на щеколду, достаточный для ее открытия. Впоследствии щеколда просто удерживается в открытом состоянии напряжением аккумулятора.

В хорошем кодовом замке электронного типа предусмотрена возможность отпирания вручную с внутренней стороны помещения. Обычно для этих целей служит небольшая кнопка, размещенная в корпусе. Собственно, я так и сделал. На обратной стороне макета была установлена тактовая кнопка.

5.2. Макет двери

Замок устанавливается на дверь, поэтому было принято решение сделать макет дверцы. Для этого понадобилось несколько деревянных брусков, кусок ДСП и дверная петля.

5.3. Программная часть аналога кодового замка

Первоначальная идея алгоритма для кодового замка мне пришла случайно еще в 2017 году. Изучив Arduino и простейшие команды программирования, я начал придумывать идеи для проектов, которые возможно создать на данной платформе.

5.3.1. Первая версия

Идея алгоритма такова: есть 10 переменных, по умолчанию они все равны 0. В них записываются данные о нажатых кнопках (если нажата кнопка «3», прибавляем к переменной «3» единицу). После этого, идет проверка при помощи оператора «if» : равны ли переменные заданным значениям. Также, в алгоритме есть еще одна переменная – счетчик. При каждом нажатии кнопки она прибавляется. При проверке происходит сравнение значений переменных с заданными, а также значение счетчика. Если все совпало, дверь откроется. Приложение 1

В теории, все должно было работать, но опытным путем выяснилось, что данная версия оказалась нерабочей.

Главной проблемой являлось неправильное срабатывание кнопок. Существует такое понятие как дребезг кнопок. Мир не идеален, поэтому в момент нажатия на кнопку в месте соединения контакты соединяются не мгновенно, микронеровности на поверхности не позволяют пластинам мгновенно соединиться. Из-за этого в короткий промежуток времени на границе пластинок меняется и сопротивление, и взаимная емкость, вследствие чего возникает ряд изменений уровня тока и напряжения. Проще

говоря, возникают помехи. Они происходят очень быстро и исчезают за доли миллисекунд. Мы редко их замечаем в повседневной жизни, например, включая свет. При нажатии на кнопку выключателя помехи также присутствуют. Но мы их не замечаем, т.к. лампа накаливания не может менять яркость с такой скоростью. Однако в данном случае ситуация иная. Обрабатывая сигнал с кнопки, микроконтроллер замечает все изменения сигнала. В идеале, форма сигнала после нажатия на кнопку должна быть строго прямоугольная. В реальных же условиях можно заметить все скачки напряжения. Как отразится дребезг на проекте? Микроконтроллер будет принимать на входе совершенно случайный набор значений. Пауза между двумя вызовами loop составляет микросекунды, и поэтому будут считаны все мелкие изменения. Существует 2 способа устранения дребезга – аппаратный и программный.

Также, в данной версии проверка правильности пароля осуществлена не самым удобным образом. Если вдруг захочется поменять пароль, придется изменять код, и чтобы это осуществить, придется в этом разобраться, что не очень просто. Мной было внесено много изменений, было написано много похожих алгоритмов, и, пожалуй, более-менее рабочим можно признать следующий вариант.

5.3.2. Вторая версия

Здесь я постарался максимально исправить ошибки первой версии. Для этого пришлось полностью изменить алгоритм. Его идея такова: Есть 4 переменные, в них хранится верный код. Есть еще 4 переменные, в них записывается номер нажатой кнопки. После каждого нажатия сравниваются переменные с верным кодом и с номерами нажатых кнопок. Также, работает счетчик нажатий, как в первом случае. Если все совпало, и счетчик насчитал нужное количество нажатий, открываем дверь.

Также, было принято решение устранить дребезг кнопок. Во-первых, контакты с подключенными кнопками я обозначил как «INPUT_PULLUP». Такое действие подключает внутренние подтягивающие резисторы Arduino, что должно в некоторой степени устранить дребезг. Полностью устранить его не удалось, поэтому я попытался исправить это другим способом. Сделано это было, как выяснилось позже, не самым лучшим образом. Для устранения дребезга я использовал задержки при помощи команды delay(). Проблема использования этой команды заключается в том, что при ее вызове

останавливается весь код. После вызова приложение не будет получать никаких данных с датчиков. Это является самым большим недостатком использования функции delay в Arduino.

Еще одной важной доработкой является возможность смены пароля прямо с устройства. Идея была такова: Нажимаем отдельную кнопку на панели, вводим действующий пароль; если проверка пароля прошла успешно, переходим непосредственно к смене пароля. При нажатии кнопок полученные значения должны записываться в переменные с верным кодом. Однако возникает проблема: для хранения переменных в процессе выполнения программы используется статическое ОЗУ, т.е. оперативная память. После перезагрузки Arduino она теряет всю информацию, т.е. переменные, измененные во время работы, обнуляются. Таким образом, если мы поменяем пароль, то после перезагрузки платы он останется прежним. Для решения данной проблемы была использована энергонезависимая память EEPROM. Для ее работы нужна специальная библиотека. Чтобы задействовать библиотеку в скетче, подключаем её директивой *#include EEPROM.h***.** Переменные с верным кодом я заменил элементами энергонезависимой памяти, и при выполнении программы введенный пароль сравнивается с ними. При смене пароля, эти элементы просто заполняются другими значениями.

Работа с EEPROM

Мной был написан пробный алгоритм для того, чтобы понять принцип заполнения и изменения ячеек EEPROM.

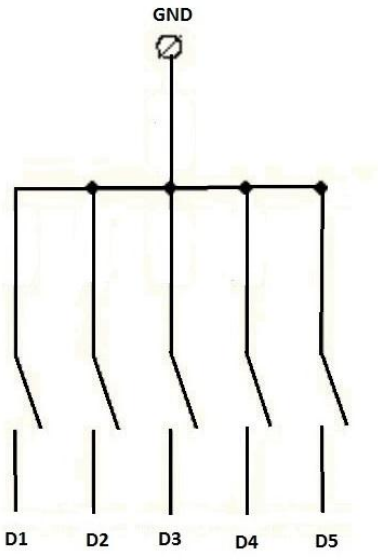
```
#include <EEPROM.h>
// начальный адрес памяти EEPROM
int address = 0;
byte value;
byte otg=0;
void setup(){
 Serial.begin(9600);
  pinMode(3,INPUT_PULLUP);
   pinMode(2,INPUT_PULLUP);}
void loop(){
 if (digitalRead(3)==0){ EEPROM.write(1, +5);delay(50);}
   if (digitalRead(2)==0){ EEPROM.write(1, 0);delay(50);}
  // считываем значение по текущему адресу EEPROM
  value = EEPROM.read(address);
  Serial.print(EEPROM.read(1));
  Serial.println();
  // устанавливаем следующую ячейку памяти
  address = address + 1;
  // EEPROM содержит всего 512 байт: от 0 до 511, поэтому
  // если адрес достиг 512, то снова переходим на 0
  if (address == 512)
    address = 0;
  delay(100);}
```

Опытным путем удалось проверить работу данного алгоритма. При нажатии на определенные кнопки в ячейки памяти записывались заданные значения. После перезагрузки платы они сохранялись. Именно этот тип памяти позволяет сохранять значение пароля вне зависимости от питания платы.Версия доступна для просмотра в приложении 2

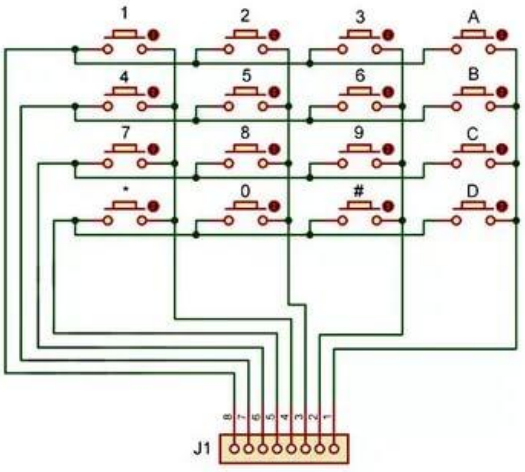
Как видно, код получился достаточно громоздким, да и дребезг кнопок здесь нормально устранить не получилось. Поэтому, при нажатии кнопок они либо не срабатывали, либо контроллер засекал несколько нажатий вместо одного. Этот вариант оказался рабочим, но ужасно нестабильным. Было решено снова переписать код, но в этот раз учесть все ошибки и по возможности исправить их, используя другие пути написания программы.

5.3.3. Третья версия (итоговая)

Первыми очень важным изменением является использование специальной библиотеки для работы с клавиатурой <Keypad.h>. Отличается и способ подключения клавиатуры. В прошлых версиях я использовал самодельную клавиатуру из тактовых кнопок, подключенных каждой к своему входу, с общим минусом (Gnd). В этом варианте я использую готовую мембранную клавиатуру, предназначенную как раз для работы с Arduino. Кнопки в ней подключены по типу матрицы (к входам платы подключаются столбцы и строки клавиатуры).



Подключение с общим минусом



Подключение по типу матрицы

Использование библиотеки полностью устранило дребезг кнопок. В новой версии для хранения нажатых кнопок используется массив, а не переменные, как в прошлых версиях. Контроллер опрашивает клавиатуру на наличие нажатых кнопок, если они есть, то их номера записываются в массив. Верный код все также хранится в массиве энергонезависимой памяти. После введения кода (нажаты 4 кнопки) производится проверка совпадений. Сравнивается массив нажатых кнопок с энергонезависимой памятью, если все совпало, открываем дверь. Изменился и алгоритм смены пароля. При нажатии определенной кнопки на клавиатуре (в моем случае – кнопка «A») программа переходит в цикл проверки верного кода. Если все совпало, программа переходит в цикл самой смены пароля. Значения нажатых кнопок записываем в отдельный массив, при этом работает счетчик нажатий. Если все кнопки нажаты, записываем в энергонезависимую память элементы массива с новым кодом. Так же, есть кнопка сброса. Если мы набирали код и ошиблись, можно нажать на кнопку сброса. При ее нажатии все переменные обнуляются, заканчиваются все циклы, контроллер переходит обратно в режим считывания нажатых кнопок. Еще одной полезной доработкой является таймер сброса. После 10 секунд бездействия (например, мы вводили код, и нас отвлекли) все переменные обнуляются, циклы завершаются, аналогично нажатию кнопки сброса. Контроллер также переходит в режим проверки кнопок. Код можно посмотреть в приложении 3

По сравнению с прошлыми версиями данный вариант кода получился максимально доработанным и стабильным. Во-первых, большим плюсом стало устранение дребезга кнопок за счет использования библиотеки для работы с клавиатурой. Во-вторых, этот вариант отличается минимально возможным количеством функций delay. Благодаря этому программа не приостанавливается там, где это не нужно, тем самым не тормозит работу всего алгоритма. В-третьих, вместо большого количества операторов if были использованы циклы с массивами, что также сыграло важную роль в оптимизации.

Мой итоговый вариант обладает следующими плюсами и минусами:

Плюсы:

+ Написанный код полностью рабочий, его работоспособность демонстрируется на макете. Можно считать готовым продуктом.

+ В алгоритме предусмотрен сброс после 10 секунд бездействия.

+ При смене пароля новая последовательность цифр не будет записана в память, пока не введен полностью весь новый пароль. Это полезно в случаях, когда, например, отключается электричество. Допустим, мы меняли пароль, уже ввели 2 новые цифры, и внезапно отключилось электричество. Новый пароль при этом записан не будет.
+ Электронная часть позволяет использовать довольно мощные щеколды.
+Цена. Проект обошелся приблизительно в 700 рублей.
+Универсальность. Можно поставить как на дверь, так и на шкафчики или выдвижные ящики.

Минусы:

- В моем проекте не предусмотрено питание системы. Его можно осуществить от блока питания на 12 В с максимальным током 2 А или более.
-При питании от сети большой проблемой будет являться отключение электроснабжения. Щеколда в выключенном состоянии закрыта, поэтому при отключении электроэнергии замок заблокируется.

Что можно доработать?

+ Есть возможность снизить энергопотребление платы за счет программного отключения неиспользуемых узлов платы.
+ Исправить проблему зависимости от электросети. Можно добавить резервный источник питания (аккумулятор), который будет поддерживать работу замка на протяжении нескольких часов.
+ При желании можно заменить тип считывающего или запирающего устройства, однако код придется дописывать под соответствующие доработки. Моя тема – кодовый замок, поэтому при написании кода не была учтена возможность смены считывающего устройства.

Выводы

При выполнении работы были сделаны следующие выводы:

1) При выборе деталей стоит учитывать все характеристики заранее. Из -за слишком мощной щеколды мне пришлось добавлять в схему конденсаторы для полноценного открытия замка.

2)Перед тем, как приступить к написанию программы хорошим решением будет тщательно изучить концепцию программирования, так как это может сыграть довольно важную роль в оптимизации кода. В данном случае сделан вывод, что при возможности лучшим решением является использование специальных библиотек.

3) При желании данное устройство можно доработать под себя, добавлять новые удобные варианты открытия замка. Моей целью являлось показать возможность создания именно кодовых замков на основе Arduino. Тему электронных замков на данной платформе можно развивать дальше. Устройство имеет возможность дальнейших доработок и преобразований.

4) Подобное может создать не каждый, но при большом желании и стремлении похожую схему может сделать даже подросток. Для этого необходимо иметь базовые знания в использовании Arduino, а также некоторые принципы написании программ на данной платформе. Может пригодиться и некоторый опыт в сборке электрических схем, а также умение паять. При отсутствии данных навыков есть возможность сборки схем на готовых модулях, предназначенных специально для этого.

Источники информации:

1. **Саймон Монк**. Мейкерство. Arduino и Raspberry Pi. Управление движением, светом и звуком
2. https://ru.m.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B7%D0%B0%D0%BC%D0%BE%D0%BA
3. https://ru.m.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4%D0%BE%D0%B2%D1%8B%D0%B9_%D0%B7%D0%B0%D0%BC%D0%BE%D0%BA
4. https://ru.m.wikipedia.org/wiki/Arduino
5. https://www.dverizamki.org/forum/index.php?topic=10669.0

Приложение 1 (Назад)

```
byte one =0;
byte two =0;
byte thre =0;
byte four =0;
byte five =0;
byte six =0;
byte seven =0;
byte eight =0;
byte nine =0;
byte nul =0; // 10 переменных для
byte set = 0; // счетчик нажатий
void setup() {

pinMode(1,INPUT);
pinMode(2,INPUT);
pinMode(3,INPUT);
pinMode(4,INPUT);
pinMode(5,INPUT);
pinMode(6,INPUT);
pinMode(7,INPUT);
pinMode(8,INPUT);
pinMode(9,INPUT);
pinMode(10,INPUT); // Настраиваем
```

```
void loop() {

 if (digitalRead(1) == 1){
   one++;}
 if (digitalRead(2) == 1){
   two++;}
   if (digitalRead(3) == 1){
   thre++;}
   if (digitalRead(4) == 1){
   four++;}
   if (digitalRead(5) == 1){
   five++;}
   if (digitalRead(6) == 1){
   six++;}
   if (digitalRead(7) == 1){
   seven++;}
   if (digitalRead(8) == 1){
   eight++;}
   if (digitalRead(9) == 1){
   nine++;}
   if (digitalRead(10) == 1){
   nul++;}                    /

   if(one ==1){set++;};
   if(two==1 && set==1){set++;};
   if(thre==1 && set==2){set++;};
   if(one==2 && set==3){
     digitalWrite(13,HIGH);}; //E
```

```
#include <Servo.h> // Библиотека для работы с мотором
Servo motor;
#include <EEPROM.h> // Энергонезависимая память для хранения пароля

byte set = 0; // Счетчик нажатий кнопок
byte code1 =0;
byte code2 =0;
byte code3 =0;
byte code4 =0; // Четыре переменные, в них будут записываться данные о нажатии кнопок

byte setpin=0; // Счетчик ввода пароля
byte codeMode=0; // Переменная для смены пароля

  void setup() {
    Serial.begin(9600);

motor.attach(A0); // сюда подключен мотор

 motor.write(0);
// закрываем дверь


pinMode(0,INPUT_PULLUP);
pinMode(12,INPUT_PULLUP);
pinMode(2,INPUT_PULLUP);
pinMode(3,INPUT_PULLUP);
pinMode(4,INPUT_PULLUP);
pinMode(5,INPUT_PULLUP);
pinMode(6,INPUT_PULLUP);
pinMode(7,INPUT_PULLUP);
pinMode(8,INPUT_PULLUP);
pinMode(9,INPUT_PULLUP);  // Кнопки от 0 до 9

pinMode(10,INPUT_PULLUP); // Кнопка смены пароля
pinMode(13,INPUT_PULLUP); //Кнопка ввода
```

```
//-------------------------------------СМЕНА ПАРОЛЯ-----------------------------
delay(100);

if(digitalRead(10)==0 && codeMode==0){codeMode++; tone(A1, 1600); delay(70);noT
if(setpin==4 && codeMode==1){codeMode++;delay(10);}

if(digitalRead(10)==0 && codeMode==2){codeMode++;tone(A1,1600);delay(50); noTon



if (codeMode==3 && digitalRead(0)==0){EEPROM.write(1,+1);codeMode++;delay(115);
if (codeMode==3 && digitalRead(12)==0){EEPROM.write(1,+2);codeMode++;delay(115)
if (codeMode==3 && digitalRead(2)==0){EEPROM.write(1,+3);codeMode++;delay(115);
if (codeMode==3 && digitalRead(3)==0){EEPROM.write(1,+4);codeMode++;delay(115);
if (codeMode==3 && digitalRead(4)==0){EEPROM.write(1,+5);codeMode++;delay(115);
if (codeMode==3 && digitalRead(5)==0){EEPROM.write(1,+6);codeMode++;delay(115);
if (codeMode==3 && digitalRead(6)==0){EEPROM.write(1,+7);codeMode++;delay(115);
if (codeMode==3 && digitalRead(7)==0){EEPROM.write(1,+8);codeMode++;delay(115);
if (codeMode==3 && digitalRead(8)==0){EEPROM.write(1,+9);codeMode++;delay(115);
if (codeMode==3 && digitalRead(9)==0){EEPROM.write(1,+10);codeMode++;delay(115)

if (codeMode==4 && digitalRead(0)==0){EEPROM.write(2,+1);codeMode++;delay(115);
if (codeMode==4 && digitalRead(12)==0){EEPROM.write(2,+2);codeMode++;delay(115)
if (codeMode==4 && digitalRead(2)==0){EEPROM.write(2,+3);codeMode++;delay(115);
if (codeMode==4 && digitalRead(3)==0){EEPROM.write(2,+4);codeMode++;delay(115);
if (codeMode==4 && digitalRead(4)==0){EEPROM.write(2,+5);codeMode++;delay(115);
if (codeMode==4 && digitalRead(5)==0){EEPROM.write(2,+6);codeMode++;delay(115);
if (codeMode==4 && digitalRead(6)==0){EEPROM.write(2,+7);codeMode++;delay(115);
if (codeMode==4 && digitalRead(7)==0){EEPROM.write(2,+8);codeMode++;delay(115);
if (codeMode==4 && digitalRead(8)==0){EEPROM.write(2,+9);codeMode++;delay(115);
if (codeMode==4 && digitalRead(9)==0){EEPROM.write(2,+10);codeMode++;delay(115)

if (codeMode==5 && digitalRead(0)==0){EEPROM.write(3,+1);codeMode++;delay(115);
if (codeMode==5 && digitalRead(12)==0){EEPROM.write(3,+2);codeMode++;delay(115)
if (codeMode==5 && digitalRead(2)==0){EEPROM.write(3,+3);codeMode++;delay(115);
if (codeMode==5 && digitalRead(3)==0){EEPROM.write(3,+4);codeMode++;delay(115);
if (codeMode==5 && digitalRead(4)==0){EEPROM.write(3,+5);codeMode++;delay(115);
if (codeMode==5 && digitalRead(5)==0){EEPROM.write(3,+6);codeMode++;delay(115);
if (codeMode==5 && digitalRead(6)==0){EEPROM.write(3,+7);codeMode++;delay(115);
if (codeMode==5 && digitalRead(7)==0){EEPROM.write(3,+8);codeMode++;delay(115);
if (codeMode==5 && digitalRead(8)==0){EEPROM.write(3,+9);codeMode++;delay(115);
if (codeMode==5 && digitalRead(9)==0){EEPROM.write(3,+10);codeMode++;delay(115)

if (codeMode==6 && digitalRead(0)==0){EEPROM.write(4,+1);codeMode++;delay(115);
if (codeMode==6 && digitalRead(12)==0){EEPROM.write(4,+2);codeMode++;delay(115)
if (codeMode==6 && digitalRead(2)==0){EEPROM.write(4,+3);codeMode++;delay(115);
if (codeMode==6 && digitalRead(3)==0){EEPROM.write(4,+4);codeMode++;delay(115);
if (codeMode==6 && digitalRead(4)==0){EEPROM.write(4,+5);codeMode++;delay(115);
if (codeMode==6 && digitalRead(5)==0){EEPROM.write(4,+6);codeMode++;delay(115);
if (codeMode==6 && digitalRead(6)==0){EEPROM.write(4,+7);codeMode++;delay(115);
if (codeMode==6 && digitalRead(7)==0){EEPROM.write(4,+8);codeMode++;delay(115);
if (codeMode==6 && digitalRead(8)==0){EEPROM.write(4,+9);codeMode++;delay(115);
if (codeMode==6 && digitalRead(9)==0){EEPROM.write(4,+10);codeMode++;delay(115)


//-------------------------------------------------------------------------------
void(* resetFunc) (void) = 0;

if(digitalRead(13)==0 && setpin!=4){tone(A1,100);delay(150);resetFunc();}

if(digitalRead(13)==0 && setpin==4){tone(A1,1300);delay(150);noTone(A1);motor.w
// Это действия, выполняемые после правильного введения кода

if(codeMode==7){delay(50);tone(A1,400);delay(90);tone(A1,1000);delay(90);resetF

if(set==4 && codeMode==1){tone(A1,100);delay(200);resetFunc();}
}
```

```
// ------------------------------ПРОГРАММНАЯ ЧАСТЬ-----------------
 Serial.println(set);

if(digitalRead(0)==0 || digitalRead(12)==0 || digitalRead(2)==0 ||
 digitalRead(3)==0 || digitalRead(4)==0 || digitalRead(5)==0 ||
 digitalRead(6)==0 || digitalRead(7)==0 || digitalRead(8)==0 ||
 digitalRead(9)==0){tone(A1,15,700);delay(50);}

if (digitalRead(0) == 0 && set==0){code1+=1;set++;delay(95);}
if (digitalRead(12) == 0 && set==0){code1+=2;set++;delay(95);}
if (digitalRead(2) == 0 && set==0){code1+=3;set++;delay(95);}
if (digitalRead(3) == 0 && set==0){code1+=4;set++;delay(95);}
if (digitalRead(4) == 0 && set==0){code1+=5;set++;delay(95);}
if (digitalRead(5) == 0 && set==0){code1+=6;set++;delay(95);}
if (digitalRead(6) == 0 && set==0){code1+=7;set++;delay(95);}
if (digitalRead(7) == 0 && set==0){code1+=8;set++;delay(95);}
if (digitalRead(8) == 0 && set==0){code1+=9;set++;delay(95);}
if (digitalRead(9) == 0 && set==0){code1+=10;set++;delay(95);}

if(code1 == EEPROM.read(1) && setpin==0){setpin++;delay(50);}

if (digitalRead(0) == 0 && set==1){code2+=1;set++;delay(95);}
if (digitalRead(12) == 0 && set==1){code2+=2;set++;delay(95);}
if (digitalRead(2) == 0 && set==1){code2+=3;set++;delay(95);}
if (digitalRead(3) == 0 && set==1){code2+=4;set++;delay(95);}
if (digitalRead(4) == 0 && set==1){code2+=5;set++;delay(95);}
if (digitalRead(5) == 0 && set==1){code2+=6;set++;delay(95);}
if (digitalRead(6) == 0 && set==1 ){code2+=7;set++;delay(95);}
if (digitalRead(7) == 0 && set==1){code2+=8;set++;delay(95);}
if (digitalRead(8) == 0 && set==1){code2+=9;set++;delay(95);}
if (digitalRead(9) == 0 && set==1){code2+=10;set++;delay(95);}

if(code2 == EEPROM.read(2) && setpin==1){setpin++;delay(50);}
```

```
if (digitalRead(0) == 0 && set==2){code3+=1;set++;delay(95);}
if (digitalRead(12) == 0 && set==2){code3+=2;set++;delay(95);}
if (digitalRead(2) == 0 && set==2){code3+=3;set++;delay(95);}
if (digitalRead(3) == 0 && set==2){code3+=4;set++;delay(95);}
if (digitalRead(4) == 0 && set==2){code3+=5;set++;delay(95);}
if (digitalRead(5) == 0 && set==2){code3+=9;set++;delay(95);}
if (digitalRead(6) == 0 && set==2){code3+=7;set++;delay(95);}
if (digitalRead(7) == 0 && set==2){code3+=8;set++;delay(95);}
if (digitalRead(8) == 0 && set==2){code3+=9;set++;delay(95);}
if (digitalRead(9) == 0 && set==2){code3+=10;set++;delay(95);}

if(code3 == EEPROM.read(3) && setpin==2){setpin++;delay(50);}


if (digitalRead(0) == 0 && set==3){code4+=1;set++;delay(95);}
if (digitalRead(12) == 0 && set==3){code4+=2;set++;delay(95);}
if (digitalRead(2) == 0 && set==3){code4+=3;set++;delay(95);}
if (digitalRead(3) == 0 && set==3){code4+=4;set++;delay(95);}
if (digitalRead(4) == 0 && set==3){code4+=5;set++;delay(95);}
if (digitalRead(5) == 0 && set==3){code4+=6;set++;delay(95);}
if (digitalRead(6) == 0 && set==3){code4+=7;set++;delay(95);}
if (digitalRead(7) == 0 && set==3){code4+=8;set++;delay(95);}
if (digitalRead(8) == 0 && set==3){code4+=9;set++;delay(95);}
if (digitalRead(9) == 0 && set==3){code4+=10;set++;delay(95);}

if(code4 == EEPROM.read(4) && setpin==3){setpin++;delay(50);}
```

```
#include <EEPROM.h> // энергонезависимая память для хранения пароля
#include <Keypad.h> // для клавиатуры
#define MOSFET 10 // транзистор
#define NUM_KEYS 4 // количество знаков в коде
boolean c=0;
byte done = 0;
uint32_t myTimer1;
char key;
char key1;
char newcode[NUM_KEYS];
char button_pressed[NUM_KEYS]; //массив для хранения нажатых кнопок
int k=0; // счетчик нажатий
int s=0; // счетчик совпадений нажатых кнопок с верными
const byte ROWS = 4; // количество строк в матрице клавиатуры
const byte COLS = 4; // количество столбцов
char keys[ROWS][COLS] = { // таблица соответствия кнопок символам
{'1','2','3','A'},
{'4','5','6','B'},
{'7','8','9','C'},
{'*','0','#','D'}};
byte rowPins[ROWS] = {5, 4, 3, 2}; // пины подключенных строк
byte colPins[COLS] = {9, 8, 7, 6}; // пины подключенных столбцов

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS ); // создаем

void setup() {Serial.begin(9600);
//EEPROM.write(0,'4');EEPROM.write(1,'4');EEPROM.write(2,'4');EEPROM.write(3,'4');

pinMode(MOSFET, OUTPUT); //  управление замком
pinMode(11, INPUT_PULLUP); // Кнопка внутри
}
```

```
done = 1; // Запоминаем, что проверка прошла успешно
 k=0;
s=0;
}
else {c=!c; // если не все кнопки совпали с верным кодом
tone(A1,100);
delay(150);
noTone(A1);
done=2;
k=0; // обнуляем счетчики, чтобы начать все заново
s=0;
]}
}
if (done==1 || done==2){ c=!c;} //Если проверка прошла успешно, выходим из цик
}
if(done==2)done=0;
//-----------------------------------------------------------------СМЕНА ПАРОЛЯ--

while(done==1){

if (millis() - myTimer1 >= 10000) {   // ищем разницу
    myTimer1 = millis();              // сброс таймера
    c=0; k=0; s=0; done=0; tone(A1,100);delay(100);noTone(A1);} //Если с после
      key1 = keypad.getKey();         //Ждем пароль
if (key1 == 'B'){c=0;k=0;s=0;done=0; tone(A1,100);delay(100);noTone(A1);key1=0;

    if ( key1 != NO_KEY )
    {tone(A1,15,100);delay(50);noTone;  myTimer1 = millis();
 newcode [k] = key1;
k = k + 1;
if(k == NUM_KEYS)
{ tone(A1,400);delay(90);tone(A1,1000);delay(70);noTone(A1); k=0; s=0;  done=0;
  for(int i=0;i<=NUM_KEYS;i++){
    EEPROM.write(i,newcode[i]);  // Меняем пароль
  }}}}
```

```
void loop (){
  Serial.println(key);

if(digitalRead(11) == 0){      // Если нажали кнопку изнутри
  digitalWrite (MOSFET, HIGH);
delay (250);                   // Открываем дверь
digitalWrite (MOSFET, LOW);
}

  key = keypad.getKey(); // спрашиваем у клавиатуры, есть нажатая кнопка?
// ----------------------------------------------------------ПРОВЕРКА ПАРОЛЯ ДЛЯ СМЕНЫ----------

 if (key == 'A'){c=1; tone(A1, 1600); delay(70);noTone(A1);k=0;s=0;myTimer1 = millis(); }// Если нажата a, войт
      while(c==1){
       if (millis() - myTimer1 >= 10000) {   // ищем разницу
    myTimer1 = millis();              // сброс таймера
    c=0; k=0; s=0; tone(A1,100);delay(100);noTone(A1);} //Если с последнего нажатия прошло 10 секунд - выходим

    key = keypad.getKey();   //Ждем пароль
     if (key == 'B'){c=0;k=0;s=0;done=0; tone(A1,100);delay(100);noTone(A1);key=0;} //Если передумали - выходим

     if ( key != NO_KEY )
{tone(A1,15,100);delay(50);noTone;  myTimer1 = millis();  // В режиме смены пароля даем другой звук
  button_pressed [k] = key;
k = k + 1;
if(k == NUM_KEYS)
{for ( uint8_t i = 0; i < NUM_KEYS; i++)
{
if (button_pressed[i] == EEPROM.read(i))
{s = s + 1; }}
if(s == NUM_KEYS){c=0; //если все кнопки совпали с кодом
tone(A1,1200);
  delay(50);
   noTone(A1);
   delay(60);
```

```
// ----------------------------------------------------------ОСНОВНОЙ ЦИК
if (key == 'B'){c=0;k=0;s=0;done=0; tone(A1,100);delay(100);noTone(A1);key=0;}

if ( key != NO_KEY ) //если мы не меняем пароль и кнопка нажата
{tone(A1,15,700);delay(50);noTone(A1); //Делаем звук
  button_pressed [k] = key;                                    //Заполняем ма
k = k + 1;                  // переключаем номер элемента массива
if(k == NUM_KEYS)                                              // если нажали
for ( uint8_t i = 0; i < NUM_KEYS; i++) // пройдемся по всему массиву
{
if (button_pressed[i] == EEPROM.read(i)) // сравниваем нажатые кнопки с верны
{s = s + 1; // плюсуем счетчик совпадений
 }}
if(s == NUM_KEYS){ //если все кнопки совпали с кодом, открываем дверь
digitalWrite (MOSFET, HIGH);
tone(A1,1600); delay(150);noTone(A1);
delay (250);
digitalWrite (MOSFET, LOW);
 k=0; // Обнуляем счетчики
s=0;
}

else { // если не все кнопки совпали с верным кодом
  delay(50);
tone(A1,100);
delay(70);
noTone(A1);
delay(30);
tone(A1,100);
delay(70);
noTone(A1);

k=0; // обнуляем счетчики
s=0;
}}}
```