

# Устройства ввода данных

Антон Парамонов  
Инженер-программист АСУ ТП



# Проверка связи





## Если у вас нет звука:

- убедитесь, что на вашем устройстве и на колонках включён звук
- обновите страницу вебинара (или закройте страницу и заново присоединитесь к вебинару)
- откройте вебинар в другом браузере
- перезагрузите компьютер (ноутбук) и заново попытайтесь зайти



## Поставьте в чат:

-  если меня видно и слышно
-  если нет

# Антон Парамонов

О спикере:

- Инженер-программист АСУ ТП
- С 2019 года работаю в проектах по модернизации производственных мощностей
- Занимался обслуживанием и модернизацией установок полного цикла изготовления алюминиевого каркаса двигателей Volkswagen MPI
- Имею опыт автоматизации на оборудовании Siemens, Omron, Allen-Bradley



# Вспоминаем прошрое занятие

**Вопрос:** Что такое коэффициент заполнения ШИМ?



# Вспоминаем прошрое занятие

**Вопрос:** Что такое коэффициент заполнения ШИМ?

**Ответ:** отношение длительности импульса к периоду



# Вспоминаем прошрое занятие

**Вопрос:** Какие обязательные файлы входят в состав библиотеки ARduino?



# Вспоминаем прошрое занятие

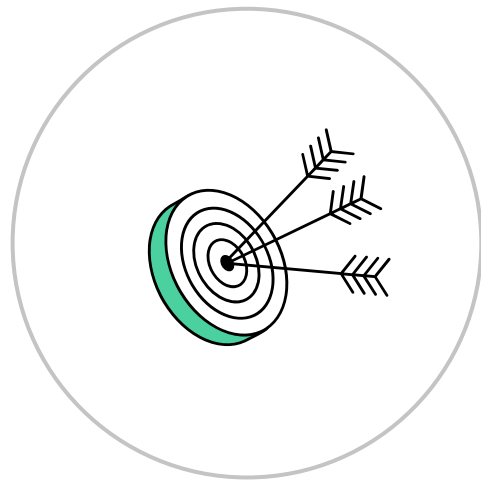
**Вопрос:** Какие обязательные файлы входят в состав библиотеки ARduino?

**Ответ:** заголовочный файл .h и файл реализации .cpp



# Цели занятия

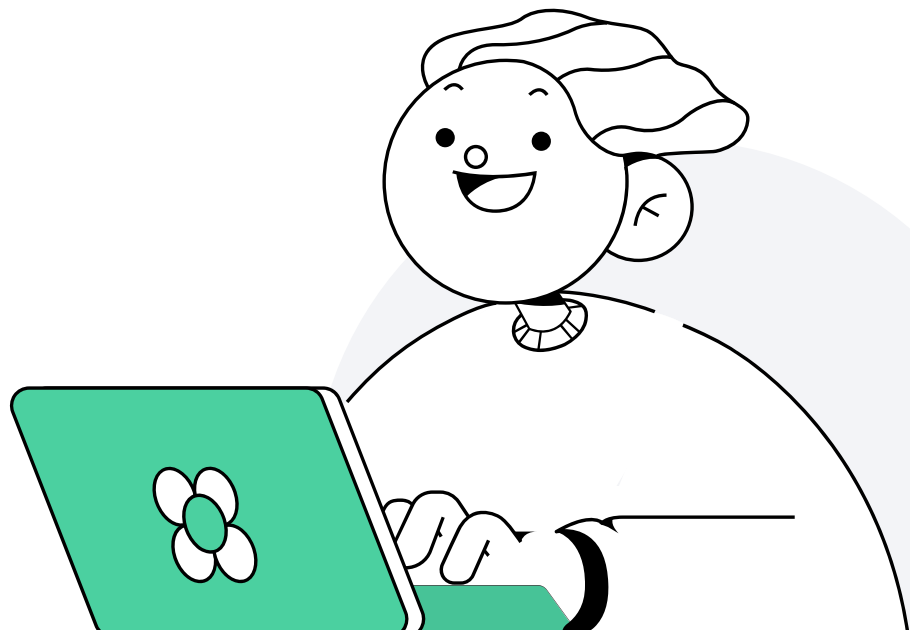
- Узнаем, как обрабатывать аналоговый сигнал
- Научимся подключать потенциометр и обрабатывать сигнал от него
- Научимся подключать аналоговый джойстик и обрабатывать сигнал от него
- Научимся подключать матричную клавиатуру





# План занятия

- 1 Как обрабатывать аналоговый сигнал
- 2 Как обрабатывать сигнал от потенциометра
- 3 Как обрабатывать сигнал от аналогового джойстика
- 4 Как подключить матричную клавиатуру
- 5 Итоги



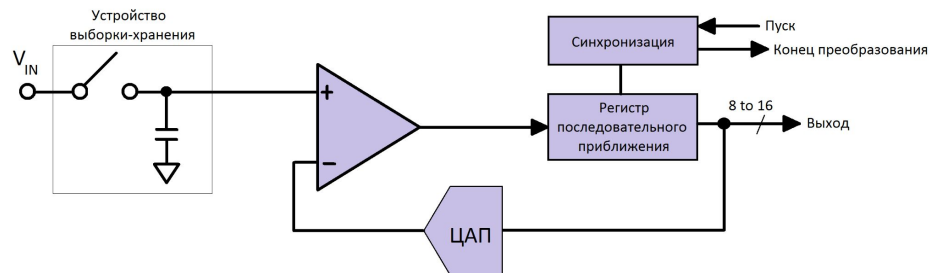
# Как обрабатывать аналоговый сигнал



1

# Структура модуля АЦП микроконтроллера

Модуль АЦП большинства микроконтроллеров реализован по архитектуре АЦП последовательного приближения. Такая архитектура позволяет строить относительно медленные АЦП (у AVR частота дискретизации до 10 кГц)



# Особенности модуля АЦП в платах Arduino

- Большинство плат имеют 6 каналов для подключения аналогового сигнала (максимум 16 каналов - у Mega)
- У большинства плат результат преобразования 10-и разрядный
- Считывание значения занимает примерно 100 мкс, в течение которого микроконтроллер не выполняет другие задачи
- Максимальное напряжение входного сигнала для большинства плат - 5 В (3,3 В у плат Zero и Due)
- При напряжении питания 5 В один разряд выходного кода соответствует примерно 4,9 мВ

# Функции аналогового ввода/вывода

`int analogRead(uint8_t pin)` — считывает значение с указанного аналогового входа

Параметры:

- `pin`: номер порта аналогового входа с которого будет производиться считывание, допустимые значения для большинства плат: 0 ... 5

Возвращаемое значение:

- цифровой код в диапазоне от 0 до 1023, пропорциональный аналоговому напряжению на входе

# Функции аналогового ввода/вывода

`void analogReference(uint8_t mode)` — определяет опорное напряжение относительно которого происходят аналоговые измерения

Параметры:

- `mode`: определяет используемое опорное напряжение
  - `DEFAULT`: стандартное опорное напряжение 5 В (на платформах с напряжением питания 5 В) или 3,3 В (на платформах с напряжением питания 3,3 В)
  - `INTERNAL`: встроенное опорное напряжение 1,1 В на микроконтроллерах ATmega168 и ATmega328, и 2,56 В на ATmega8
  - `EXTERNAL`: внешний источник опорного напряжения, подключенный к выводу AREF

Возвращаемое значение: нет

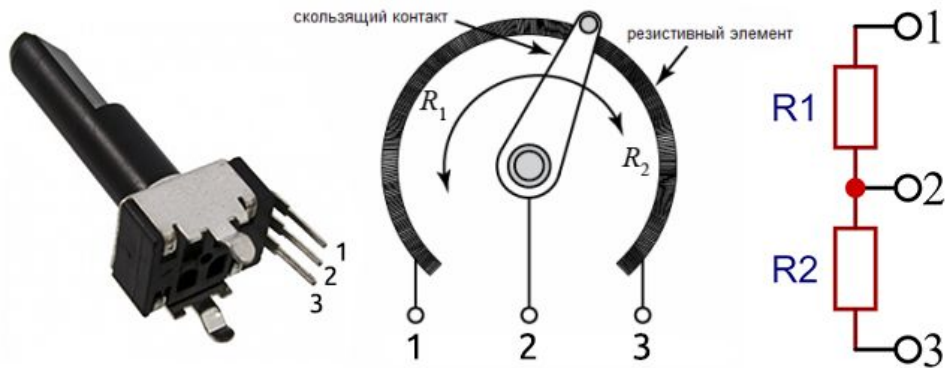
# Как обрабатывать сигнал от потенциометра



2

# Потенциометр

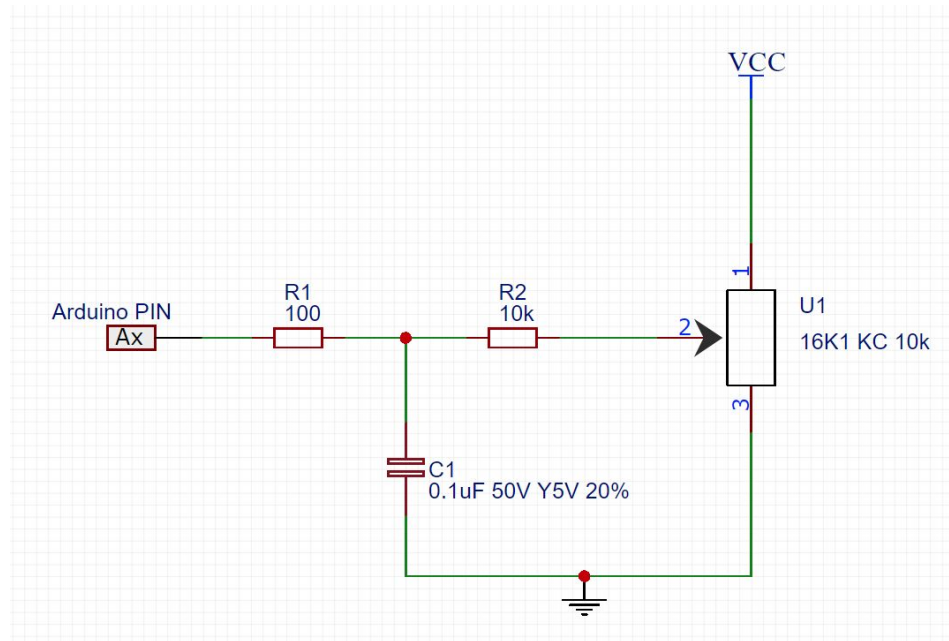
Потенциометр – переменный резистор, представляющий собой резистивный делитель напряжения с подвижной средней точкой. При подключении выводов №1 и №3 на источник напряжения (например GND и 5 В) на выводе №2 появится напряжение (относительно GND), пропорциональное положению ручки потенциометра.





# Аппаратная фильтрация помех, наводимых на аналоговый вход

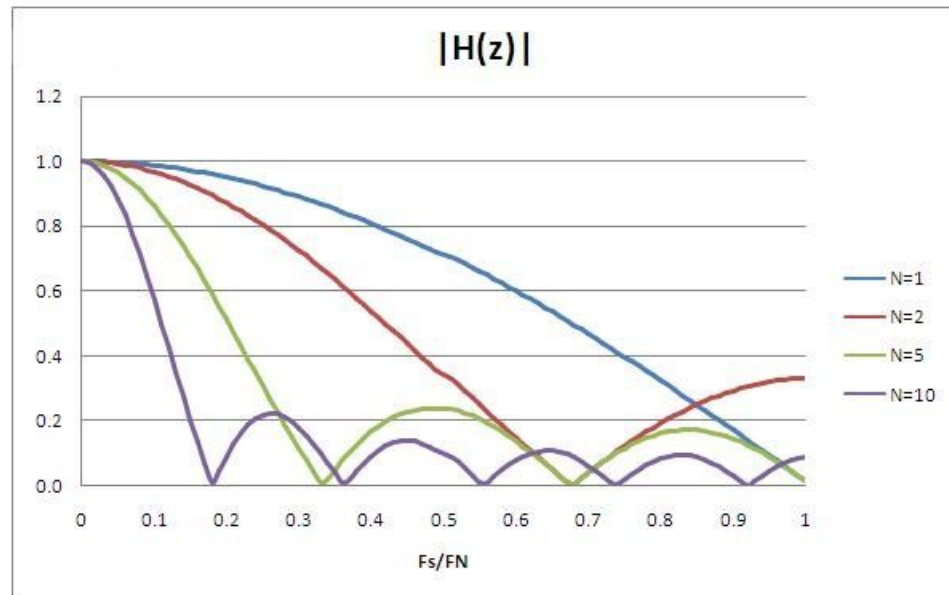
Для уменьшения уровня помех можно использовать ФНЧ, реализованный в виде RC-цепи



# Программная фильтрация помех, наводимых на аналоговый вход

Для сглаживания уровня помех можно использовать фильтр скользящего среднего:

$$Y[n] = \frac{1}{N+1} \sum_{i=0}^N X[n-i]$$



# Управление яркостью светодиода с помощью потенциометра

Пример программы:

```
const int led = 9;           //светодиод подключен к контакту 9
const int pot = A0;          //потенциометр подключен к контакту A0

void setup()
{
    Serial.begin(9600);       //последовательный порт для отладки

    pinMode(led, OUTPUT);     //контакт светодиода - выход
    digitalWrite(led, LOW);   //на выходе лог. 0

    pinMode(pot, INPUT);      //контакт потенциометра - вход
    analogReference(DEFAULT); //опорное напряжение - 5 В
}
```

# Управление яркостью светодиода с помощью потенциометра

Пример программы:

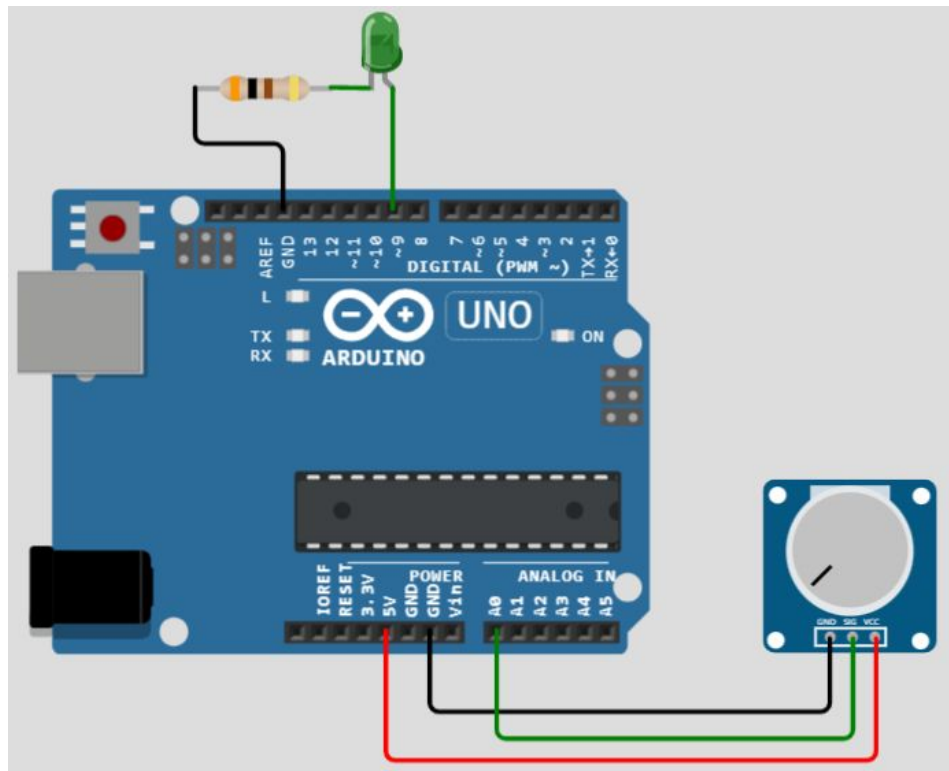
```
void loop()
{
    int x; //для напряжение потенциометра

    x = analogRead(pot);           //максимальное значение 1023
    Serial.println(x);             //вывод значения в COM-порт для отладки
    x = x/4;                       //максимальное значение 255
    analogWrite(led, x);

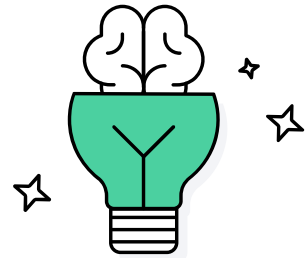
    delay(100);                   //задержка для наглядности
}
```

# Управление яркостью светодиода с помощью потенциометра

Симулятор не позволяет имитировать помехи, наводимые на аналоговые линии, поэтому рассматривается упрощенная схема



# Практическое задание N°1



# Практика: управление яркостью светодиода с помощью потенциометра

## Задание:

- 1) соберите схему в симуляторе WOKWI, подключив светодиод к выводу 9, а потенциометр - к выводу A0;
- 2) создайте скетч с текстом, приведенным выше;
- 3) проведите моделирование работы

**Как выполнять:** напишите в чат об удачной работе схемы

**Время выполнения:** 5 минут



# Как обрабатывать сигнал от аналогового джойстика

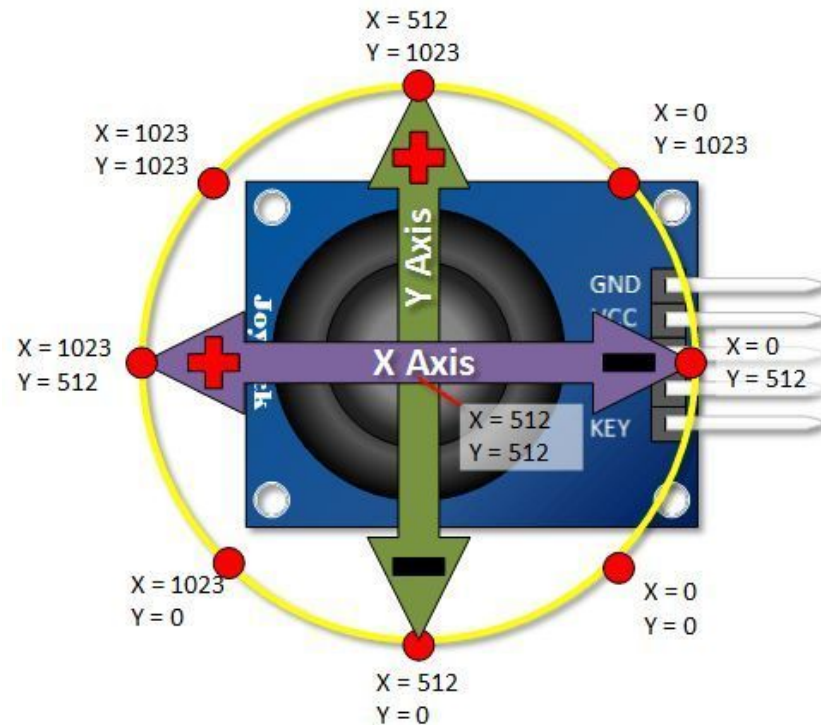


3



# Аналоговый джойстик

Аналоговый джойстик выглядит как ручка, которая закрепляется на шарнире с двумя потенциометрами, определяющими оси X и Y, и кнопкой Z. Наклон или поворот ручки вращает специальный подвижный контакт, из-за чего изменяется выходное напряжение



# Особенности аналогового джойстика

- Наличие пружины не позволяет ручке точно вернуться в центральное положение из-за трения в механических деталях. Это приводит к тому, что необходимо программно определять диапазон значений, которому соответствует центральное положение.
- Наличие “мертвых зон”. Два крайних значения при наибольших отклонениях должно быть равным 0 В и напряжению питания. В действительности эти значения могут различаться, так как не используется весь электрический диапазон изменения сопротивления. Для решения этой проблемы крайние точки могут соответствовать некоторым ненулевым значениям.

# Математические функции преобразования

`long map(long x, long in_min, long in_max, long out_min, long out_max)` — пропорционально переносит значение (x) из текущего диапазона значений (in\_min ... in\_max) в новый диапазон (out\_min ... out\_max), заданный параметрами

Параметры:

- x: значение для переноса
- in\_min: нижняя граница текущего диапазона
- in\_max: верхняя граница текущего диапазона
- out\_min: нижняя граница нового диапазона, в который переноситься значение
- out\_max: верхняя граница нового диапазона

Возвращаемое значение: значение в новом диапазоне

# Управление яркостью светодиодов с помощью джойстика

```
const int pinX    A2  // ось X джойстика
const int pinY    A1  // ось Y джойстика
const int ledX     5  // светодиод на Pin 5
const int ledY     6  // светодиод на Pin 6

void setup()
{
    pinMode(ledX, OUTPUT);
    pinMode(ledY, OUTPUT);

    pinMode(pinX, INPUT);
    pinMode(pinY, INPUT);
}
```

```
void loop()
{
    int x, y;                                // для напряжения джойстика

    int x = analogRead(pinX);                // считываем значение оси X
    int y = analogRead(pinY);                // считываем значение оси Y

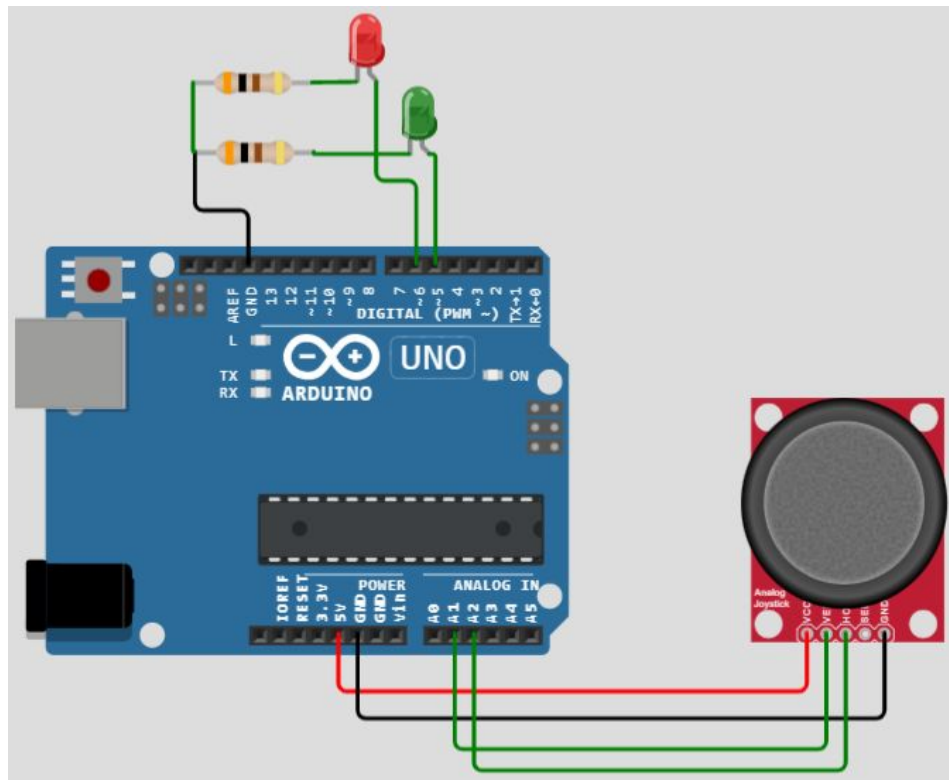
    x = map(x, 0, 1023, 0, 255); // преобразуем значение X в
    // другой диапазон
    y = map(y, 0, 1023, 0, 255); // преобразуем значение Y в
    // другой диапазон

    analogWrite(ledX, x);                    // включаем светодиоды с
    // разной яркостью
    analogWrite(ledY, y);

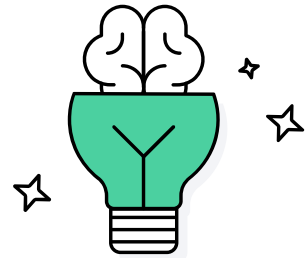
}
```

# Управление яркостью светодиодов с помощью джойстика

В примере не используется кнопка джойстика



# Практическое задание №2



# Практика: управление яркостью светодиодов с помощью джойстика

## Задание:

- 1) соберите схему в симуляторе WOKWI, подключив светодиоды к выводам 5 и 6, а джойстик - к выводам A1 и A2;
- 2) создайте скетч с текстом, приведенным выше;
- 3) проведите моделирование работы

**Как выполнять:** напишите в чат об удачной работе схемы

**Время выполнения:** 5 минут



# Как подключить матричную клавиатуру

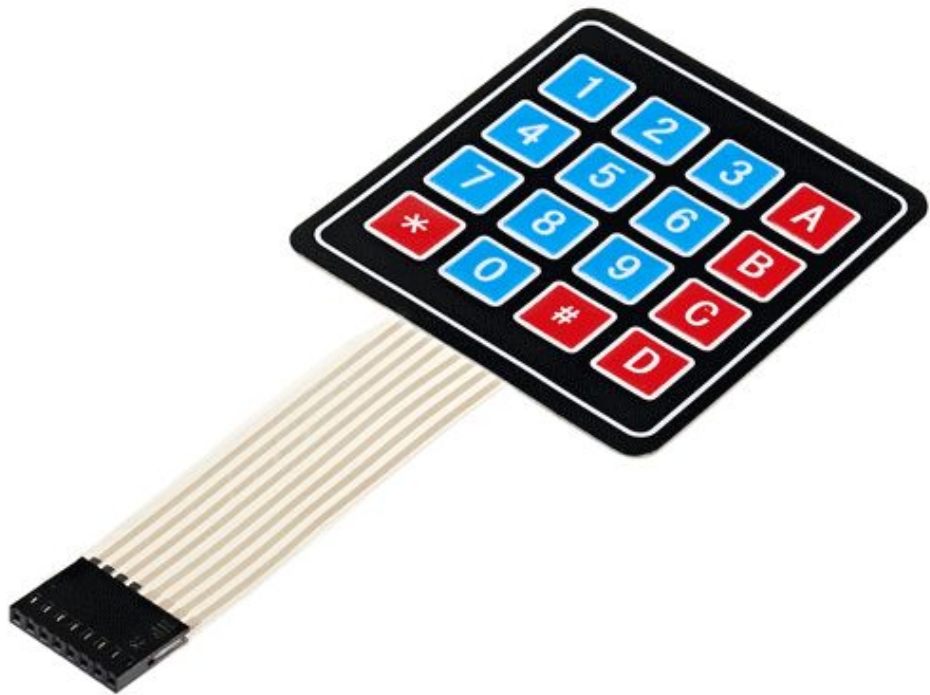


4



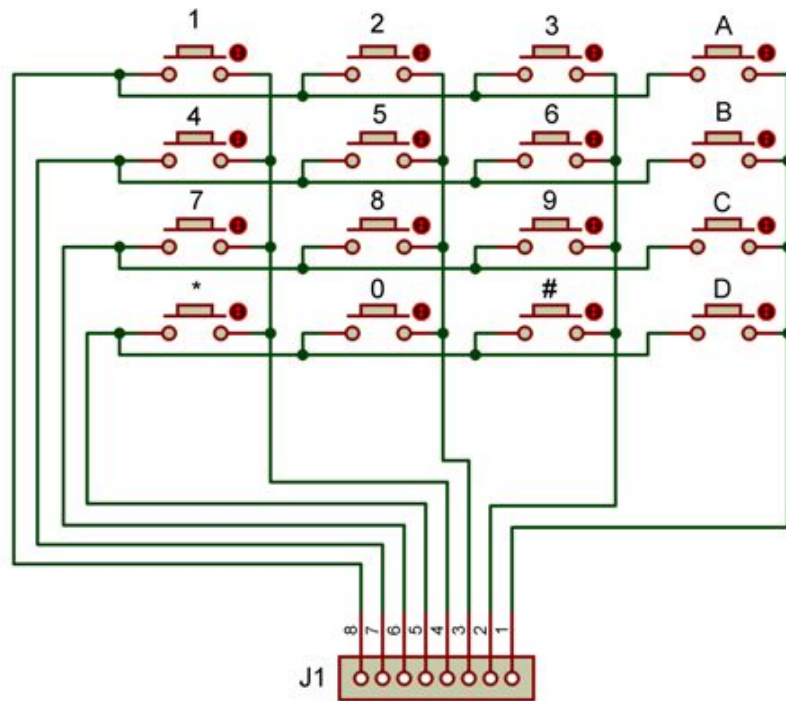
# Матричная клавиатура

Если в проекте необходимо обрабатывать большое количество кнопок, то для уменьшения количества портов ввода/вывода микроконтроллера целесообразно использовать матричную клавиатуру



# Принцип работы матричной клавиатуры

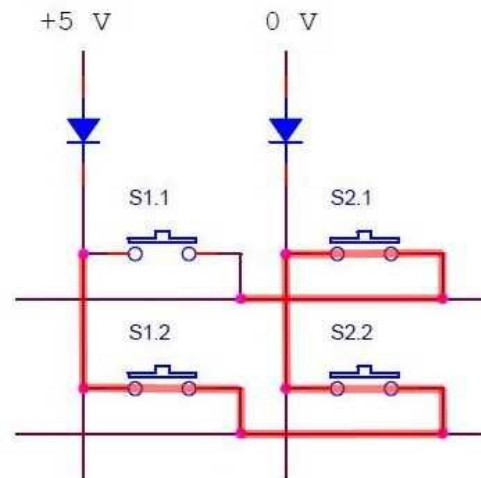
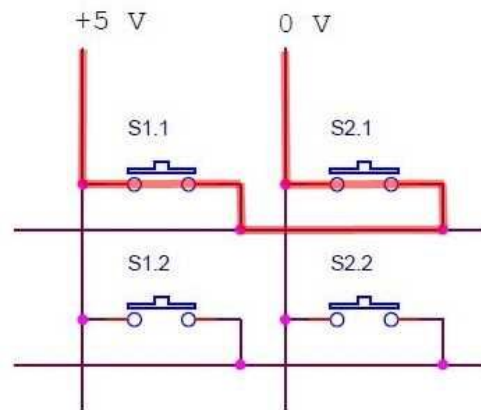
Каждая кнопка, при нажатии на нее, замыкает контакты между конкретным столбцом и конкретным рядом, создавая цепь, которую можно программно обнаружить. Например, если мы нажмем верхнюю левую кнопку на схеме, изображенной выше, мы замкнем контакты A и 1.



# Защита от короткого замыкания

При нажатии двух кнопок в одной строке схема будет работать некорректно (вплоть до короткого замыкания и выхода платы из строя)

Решение: установка диодов на сканирующие линии



# Обработка матричной клавиатуры

```
const int P[] = {11, 10, 9, 8}; // выходы строк строк (выходы)
const int M[] = {7, 6, 5, 4};   // выходы столбцов (входы)
const char k4x4 [4][4] = {      // коды символов на клавиатуре
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}
};

void setup()
{
    for (int i = 0; i <= 3; i++) // настройка выходов и входов
    {
        pinMode(P[i], OUTPUT);
        pinMode(M[i], INPUT_PULLUP); //входы с внутренней подтяжкой
        digitalWrite(P[i], HIGH);
    }
    Serial.begin(9600);
    Serial.println("begin");
}
```

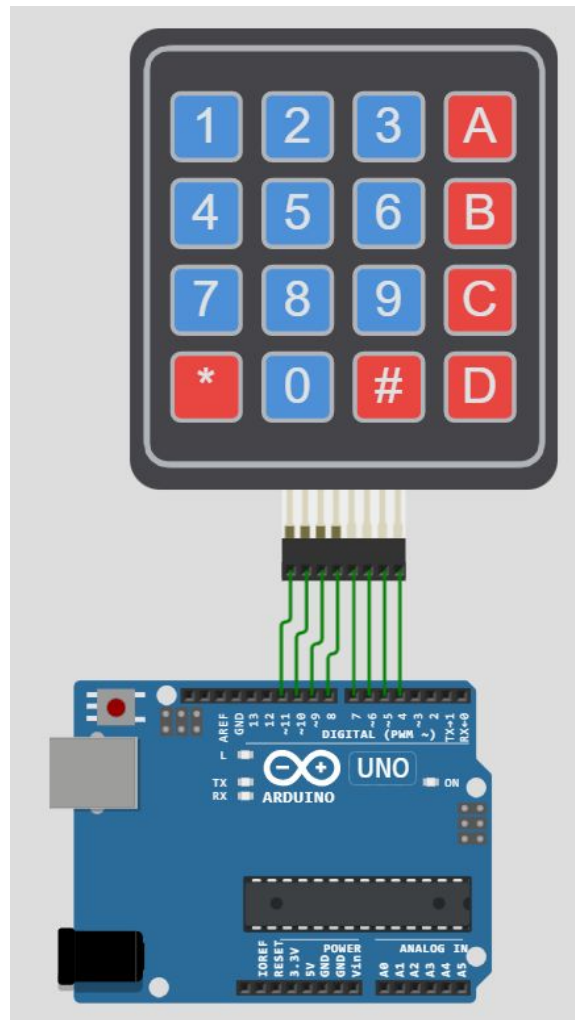
```
void loop() {
    char a = GetKey4x4(); // опрос клавиатуры
    if (a != 0) // при нажатии - вывод кнопки в порт
    {
        Serial.print(a);
    }
}
```

# Обработка матричной клавиатуры

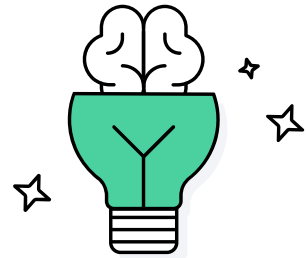
```
/*Функция опроса клавиатуры*/
char GetKey4x4()
{
    static unsigned long timer; // для подавления дребезга
    static char olda;           //старый код нажатой клавиши
    char a = 0;                 //код нажатой клавиши
    if ((timer + 50) > millis()) return 0; // пауза для подавления дребезга
    for (byte p = 0; p <= 3; p++) // последовательно выставляем по одной строке в LOW
    {
        digitalWrite(P[p], LOW);
        for (byte m = 0; m <= 3; m++) // и считываем столбцы, анализируя, где LOW происходит
        {
            if (!digitalRead(M[m]))
            {
                a = k4x4[p][m]; // считываем соответствующий символ для комбинации столбца и строки
            }
        }
        digitalWrite(P[p], HIGH); // возвращем строку в HIGH и крутим дальше
    }
    timer = millis();
    if (a == olda) return 0; //маскируем удержание
    olda = a;
    return a;
}
```

# Обработка матричной клавиатуры

В примере не показаны защитные диоды



# Практическое задание №3



# Практика: обработка матричной клавиатуры

## Задание:

- 1) соберите схему в симуляторе WOKWI, подключив матричную клавиатуру к выводам 4 ... 11;
- 2) создайте скетч с текстом, приведенным выше;
- 3) проведите моделирование работы

**Как выполнять:** напишите в чат об удачной работе схемы

**Время выполнения:** 5 минут





# Библиотека Keypad для работы с клавиатурой

`Keypad(char *userKeymap, byte *row, byte *col, byte numRows, byte numCols)` — конструктор, создающий объект Keypad

Параметры:

- `userKeymap`: указатель на массив, хранящий коды клавиш
- `row`: указатель на массив, хранящий номера выводов строк
- `col`: указатель на массив, хранящий номера выводов столбцов
- `numRows`: количество строк
- `numCols`: количество столбцов

Возвращаемое значение: нет

# Библиотека Keypad для работы с клавиатурой

`void begin(char *userKeymap)` — инициализирует внутреннюю раскладку клавиатуры, чтобы та соответствовала userKeymap

Параметры:

- userKeymap: указатель на массив, хранящий коды клавиш

Возвращаемое значение: нет

`char waitForKey()` — ожидание нажатия клавиши. **Предупреждение:** она блокирует весь остальной код, пока клавиша не будет нажата

Параметры: нет

Возвращаемое значение: код нажатой клавиши

# Библиотека Keypad для работы с клавиатурой

`char getKey()` — возвращает код нажатой клавиши, если такая есть. Данная функция неблокирующая

Параметры: нет

Возвращаемое значение: код нажатой клавиши

`KeyState getState()` — возвращает текущее состояние любой из клавиш

Параметры: нет

Возвращаемое значение: одно из четырех состояний: IDLE, PRESSED, RELEASED и HOLD

# Библиотека Keypad для работы с клавиатурой

`bool keyStateChanged()` — сообщает об изменении состояния любой из клавиш. Например, вместо проверки нужной клавиши, можно проверить, когда клавиша была нажата

Параметры: нет

Возвращаемое значение: `true` - было изменение состояния, `false` - без изменения

`void setHoldTime(uint hold)` — устанавливает количество миллисекунд, которое пользователь должен удерживать кнопку нажатой, чтобы было вызвано состояние HOLD

Параметры:

- `hold`: время в мс

Возвращаемое значение: нет

# Библиотека Keypad для работы с клавиатурой

`void setDebounceTime(uint debounce)` — устанавливает количество миллисекунд, которое клавиатура ждет перед тем, как применить новое нажатие кнопки или событие кнопки. Это «время задержки» в методе обработки дребезга контактов.

Параметры:

- `debounce`: время в мс

Возвращаемое значение: нет

`void addEventListener(void (*listener)(char))` — подключает функцию обратного вызова для обработки нажатия клавиши

Параметры:

- `listener`: имя функции обратного вызова

Возвращаемое значение: нет

# Обработка матричной клавиатуры с применением библиотеки Keypad

```
#include "Keypad.h"

const byte Rows= 4; // количество строк на клавиатуре
const byte Cols= 4; // количество столбцов на клавиатуре
// определяем массив символов соответствующий распределению кнопок на клавиатуре:
char keymap[Rows][Cols]=
{
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}
};

// соединения клавиатуры с выводами Arduino:
byte rPins[Rows]= {11,10,9,8};
byte cPins[Cols]= {7,6,5,4};

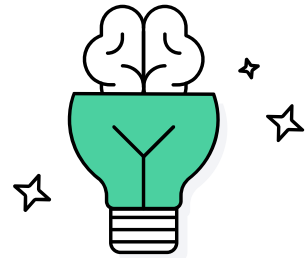
// создаем объект класса Keypad
Keypad kpd= Keypad(makeKeymap(keymap), rPins, cPins, Rows, Cols);
```

# Обработка матричной клавиатуры с применением библиотеки Keypad

```
void setup()
{
  Serial.begin(9600); // инициализация монитора последовательного порта
}

// Если кнопка нажата, эта кнопка сохраняется в переменной keypressed.
// Если keypressed не равна NO_KEY, то выводим значение в последовательный порт.
void loop()
{
  char keypressed = kpd.getKey();
  if (keypressed != NO_KEY)
  {
    Serial.println(keypressed);
  }
}
```

# Практическое задание N°4





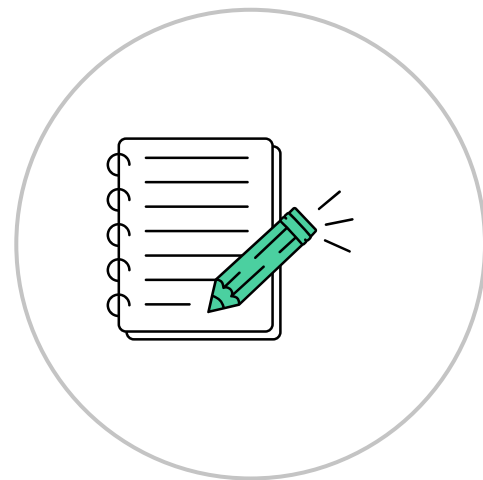
# Практика: обработка матричной клавиатуры с применением библиотеки Keypad

## Задание:

- 1) подключите в симуляторе WOKWI библиотеку Keypad
- 2) соберите схему в симуляторе WOKWI, подключив матричную клавиатуру к выводам 4 ... 11;
- 3) создайте скетч с текстом, приведенным выше;
- 4) проведите моделирование работы

**Как выполнять:** напишите в чат об удачной работе схемы

**Время выполнения:** 5 минут



# Итоги



5

# Итоги занятия

Сегодня мы

- 1 Узнали особенности обработки аналогового сигнала
- 2 Научились подключать потенциометр и обрабатывать сигнал от него
- 3 Научились подключать аналоговый джойстик и обрабатывать сигнал от него
- 4 Научились подключать матричную клавиатуру и обрабатывать ее с помощью библиотеки и без нее



# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- 1 Вопросы по домашней работе задавайте в чате группы
- 2 Задачи можно сдавать по частям
- 3 Зачёт по домашней работе ставят после того, как приняты все задачи



**Задавайте вопросы  
и пишите отзыв о лекции**

