**ARDUINO & ESP8266**

**PROGRAMMING**

INDEX

|  |  |
| --- | --- |
| S. No | Title |
| 1 | [PROGRAMMING INTRODUCTION](#PROGRAMMINGINTRODUCTION" \o "PROGRAMMINGINTRODUCTION) |
| 2 | [OPERATORS](#OPERATORS" \o "OPERATORS) |
| 3 | [Conditions](#Conditions" \o "Conditions) |
| 4 | [Loops](#Loops" \o "Loops) |
| 5 | [Functions](#Functions" \o "Functions) |
| 6 | [Array and pointer](#Arrayandpointer" \o "Arrayandpointer) |
| 7 | [About pins](#Aboutpins" \o "Aboutpins) |
| 8 | [Serial Monitor](#SerialMonitor" \o "SerialMonitor) |
| 9 | [Display](#Display" \o "Display) |
| 10 | [SD card](#SDcard" \o "SDcard) |
| 11 | [Keypad](#Keypad" \o "Keypad) |
| 12 | [IR REMOTE CONTROL](#IRREMOTECONTROL" \o "IRREMOTECONTROL) |
| 13 | [Moisture sensor](#Moisturesensor" \o "Moisturesensor) |
| 14 | [Wifi](#Wifi" \o "Wifi) |

**PROGRAMMING INTRODUCTION**

* Android app for Arduino programming [Arduinodroid App](https://arduinodroid.en.uptodown.com/android/download" \o "https://arduinodroid.en.uptodown.com/android/download)
* All functions are written in camel.

Data types:

These are the data types used in Arduino.

|  |  |  |  |
| --- | --- | --- | --- |
| **Data type** | **Example** | **Size, bytes** | **Use** |
| integer | int VARIABLE\_NAME | 2 or 4 | Defining integers numbers |
| Unsigned int | unint8\_t | 2 or 4 | Not negative numbers and uses 8 bit size, it is changable |
| Float | float VARIABLE\_NAME | 4 | Used for defining decimal numbers |
| Double | double VARIABLE\_NAME | 8 | Used for large decimal number |
| Long double | long double | 10 | Large size number |
| character | char VARIABLE\_NAME | 1 | Defining single alphabetic letter |
| Boolean | bool | True or false |  |

OPERATORS:

In Arduino (C++), operators are symbols that perform operations on variables and values. They are essential in controlling the logic, flow, and calculations within your code. Here's a breakdown of the most common operators:

**1. Arithmetic Operators**

Used for basic mathematical calculations.

| Operator | Description | Example |
| --- | --- | --- |
| + | Addition | a + b |
| - | Subtraction | a - b |
| \* | Multiplication | a \* b |
| / | Division | a / b |
| % | Modulus (remainder) | a % b |

Example:

int a = 10;

int b = 3;

int sum = a + b; // 13

int remainder = a % b; // 1

**2. Assignment Operators**

Used to assign values to variables, often with an operation included.

| Operator | Description | Example | Equivalent To |
| --- | --- | --- | --- |
| = | Assignment | a = b |  |
| += | Add and assign | a += b | a = a + b |
| -= | Subtract and assign | a -= b | a = a - b |
| \*= | Multiply and assign | a \*= b | a = a \* b |
| /= | Divide and assign | a /= b | a = a / b |
| %= | Modulus and assign | a %= b | a = a % b |

Example:

int x = 5;

x += 3; // x is now 8

x \*= 2; // x is now 16

**3. Comparison Operators**

| Operator | Description | Example |
| --- | --- | --- |
| == | Equal to | a == b |
| != | Not equal to | a != b |
| > | Greater than | a > b |
| < | Less than | a < b |
| >= | Greater than or equal to | a >= b |
| <= | Less than or equal to | a <= b |

Used to compare two values, returning true or false.

Example:

int temp = 25;

if (temp >= 30) {

Serial.println("It's hot");

} else {

Serial.println("Temperature is normal");

}

**4. Logical Operators**

Used to perform logical operations, often in condition statements.

| Operator | Description | Example |
| --- | --- | --- |
| && | Logical AND | (a > 5) && (b < 10) |
| || | Logical OR | (a > 5) || (b < 10) |
| ! | Logical NOT | !(a > 5) |

Example:

int sensorValue = analogRead(A0);

if (sensorValue > 200 && sensorValue < 800) {

Serial.println("Moderate light detected.");

}

**5. Increment and Decrement Operators**

Used to increase or decrease the value of a variable by 1.

| Operator | Description | Example |
| --- | --- | --- |
| ++ | Increment by 1 | a++ or ++a |
| -- | Decrement by 1 | a-- or --a |

Example:

int count = 0;

count++; // count is now 1

++count; // count is now 2

Difference between Prefix and Postfix:

a++: Returns a, then increments.

++a: Increments a, then returns.

**6. Bitwise Operators**

Used to perform operations at the bit level. Useful in low-level programming.

| Operator | Description | Example |
| --- | --- | --- |
| & | Bitwise AND | a & b |
| | | Bitwise OR | a | b |
| ^ | Bitwise XOR | a ^ b |
| ~ | Bitwise NOT | ~a |
| << | Left Shift | a << 1 |
| >> | Right Shift | a >> 1 |

Example:

int a = 5; // 0b0101

int b = 3; // 0b0011

int c = a & b; // 0b0001, result is 1

**7. Ternary Operator**

The ternary operator is a shorthand for if-else statements.

variable = (condition) ? value\_if\_true : value\_if\_false;

Example:

int temperature = 28;

String status = (temperature > 30) ? "Hot" : "Normal";

Serial.println(status); // Prints "Normal"

**Summary Example Using Multiple Operators:**

void setup() {

Serial.begin(9600);

}

void loop() {

int a = 10;

int b = 5;

// Arithmetic operators

int sum = a + b;

int product = a \* b;

// Comparison and logical operators

if (a > b && b != 0) {

Serial.println("a is greater than b and b is not zero.");

}

// Bitwise and assignment operators

a |= b; // Equivalent to a = a | b;

// Ternary operator

String result = (a == b) ? "a is equal to b" : "a is not equal to b";

Serial.println(result);

delay(1000); // Wait a second before repeating

}

These operators are crucial for controlling the logic and flow in your Arduino code, allowing for everything from basic calculations to complex conditions and bitwise operations. Let me know if you need further explanation on any specific operators!

Conditions:

In Arduino, you can use conditional statements to control the flow of your program based on certain conditions. Here’s an overview of the main types of conditionals you’ll encounter:

1. if Statement

* The if statement checks a condition, and if it’s true, it executes the code inside the block.

if (condition) {

// Code to execute if condition is true

}

Example:

int sensorValue = analogRead(A0);

if (sensorValue > 500) {

Serial.println("Sensor value is high!");

}

2. if-else Statement

* The if-else statement provides an alternative action if the condition is false.

if (condition) {

// Code to execute if condition is true

}

else {

// Code to execute if condition is false

}

Example:

int temperature = 25;

if (temperature > 30) {

Serial.println("It's hot!");

}

else {

Serial.println("Temperature is normal.");

}

3. if-else if-else Statement

* The if-else if-else structure allows for multiple conditions to be checked in sequence.

if (condition1) {

// Code to execute if condition1 is true

} else if (condition2) {

// Code to execute if condition2 is true

} else {

// Code to execute if none of the above conditions are true

}

Example:

int lightLevel = analogRead(A0);

if (lightLevel < 200) {

Serial.println("It's dark.");

} else if (lightLevel >= 200 && lightLevel <= 800) {

Serial.println("It's moderate light.");

} else {

Serial.println("It's bright.");

}

4. switch-case Statement

* The switch statement is a more organized way to handle multiple conditions based on a single variable.

switch (variable) {

case value1:

// Code to execute if variable equals value1

break;

case value2:

// Code to execute if variable equals value2

break;

// Add more cases as needed

default:

// Code to execute if none of the cases are matched

}

Example:

int buttonState = digitalRead(2);

switch (buttonState) {

case HIGH:

Serial.println("Button is pressed.");

break;

case LOW:

Serial.println("Button is not pressed.");

break;

}

5. Ternary Operator

* The ternary operator is a shorthand way to write simple if-else conditions.

variable = (condition) ? value\_if\_true : value\_if\_false;

Example:

int temperature = 28;

String tempStatus = (temperature > 30) ? "Hot" : "Normal";

Serial.println(tempStatus);

**Practical Example in Arduino Code:**

This example checks the temperature from a sensor and controls an LED based on the reading:

void setup() {

Serial.begin(9600);

pinMode(LED\_BUILTIN, OUTPUT); // Initialize the LED pin

}

void loop() {

int tempSensorValue = analogRead(A0);

float temperature = (tempSensorValue / 1023.0) \* 100; // Example conversion

if (temperature > 30) {

digitalWrite(LED\_BUILTIN, HIGH); // Turn on LED if temperature is high

Serial.println("Warning: High temperature!");

}

else if (temperature < 10) {

digitalWrite(LED\_BUILTIN, LOW); // Turn off LED if temperature is low

Serial.println("Temperature is too low.");

}

else {

Serial.println("Temperature is normal.");

}

delay(1000); // Delay for a second before checking again

}

**Summary:**

These conditional statements allow you to control and respond to various conditions in your Arduino program. Whether it's based on sensor data, button presses, or other inputs, conditional statements make your program interactive and responsive.

Loops

In Arduino programming, loops are essential for controlling the flow of your program by repeating a set of instructions. Here’s an overview of the most common loops and how to use them in Arduino:

1. The loop() Function

In Arduino, the loop() function itself is a built-in loop that runs continuously after the setup() function has finished executing. It’s automatically called over and over again, which is why you don’t need a while loop to keep the program running.

void setup() {

// Code here runs once

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

// Code here runs continuously

digitalWrite(LED\_BUILTIN, HIGH);

delay(1000);

digitalWrite(LED\_BUILTIN, LOW);

delay(1000);

}

In this example, the built-in LED blinks on and off in a continuous loop.

2. for Loop

The for loop is used to repeat a block of code a specific number of times. It’s typically used when you know how many times you need the loop to execute.

Syntax:

for (initialization; condition; increment) {

// Code to execute in each iteration

}

**Example:**

void setup() {

Serial.begin(9600);

}

void loop() {

for (int i = 0; i < 10; i++) {

Serial.println(i);

delay(500);

}

delay(2000); // Wait before starting the loop again

}

This code will print the numbers 0 to 9 on the Serial Monitor, with a 500 ms delay between each number.

3. while Loop

A while loop runs as long as a specified condition is true. This loop is useful when the number of iterations isn’t known beforehand.

Syntax:

while (condition) {

// Code to execute while the condition is true

}

**Example:**

void setup() {

Serial.begin(9600);

}

void loop() {

int sensorValue = analogRead(A0);

while (sensorValue < 500) {

Serial.println("Sensor reading is low");

delay(500);

sensorValue = analogRead(A0); // Update the sensor value

}

Serial.println("Sensor reading is sufficient");

delay(1000); // Delay before rechecking in the loop

}

In this example, the program checks if a sensor reading is below 500 and continues printing "Sensor reading is low" until the value reaches 500 or above.

4. do...while Loop

The do...while loop is similar to the while loop, but it will always execute at least once, even if the condition is false.

Syntax:

do {

// Code to execute

} while (condition);

**Example:**

void setup() {

Serial.begin(9600);

}

void loop() {

int counter = 0;

do {

Serial.println("Counting: " + String(counter)); // + indicate concatenate, it convert integer into string.

counter++;

delay(500);

} while (counter < 5);

delay(2000); // Wait before restarting the loop

}

In this example, "Counting: 0" through "Counting: 4" will print to the Serial Monitor, even if the initial counter value was greater than or equal to 5.

5. break and continue Statements

break: Exits a loop entirely when triggered.

continue: Skips the current iteration and moves to the next iteration.

**Example with break and continue:**

void setup() {

Serial.begin(9600);

}

void loop() {

for (int i = 0; i < 10; i++) {

if (i == 3) {

continue; // Skip the iteration when i is 3

}

if (i == 8) {

break; // Exit the loop entirely when i is 8

}

Serial.println(i);

delay(300);

}

delay(2000); // Wait before starting the loop again

}

In this example, 3 will be skipped, and the loop will stop entirely when i reaches 8.

Each loop type has its purpose and best use cases in Arduino programming, depending on your project’s requirements. Let me know if you'd like more examples or details on specific loop types!

Functions

In Arduino programming, functions are blocks of code designed to perform a specific task. They help organize code, making it easier to read, reuse, and maintain. Here's an overview of how functions work in Arduino code:

**1. Function Structure**

A function generally has the following structure:

returnType functionName(parameterType parameterName) {

// Code to execute

return value; // Optional, depending on return type

}

* + returnType: The type of value the function returns (e.g., int, void, float). If it doesn't return anything, use void.
  + functionName: A unique name that identifies the function.
  + parameters: Optional inputs that the function can take. They are defined by their type and name.
  + function body: The code that runs when the function is called.

**2. Examples of Functions**

Basic Function Example

Here's a simple example of a function that adds two numbers:

int add(int a, int b) {

return a + b;

}

Using the Function

You can call the function in your loop() or setup():

void setup() {

Serial.begin(9600);

int sum = add(5, 3); // Calls the add function

Serial.println(sum); // Prints 8

}

void loop() {

// Empty loop

}

**3. Built-in Functions**

Arduino has many built-in functions that perform specific tasks, such as:

* digitalWrite(pin, value): Sets a digital pin to HIGH or LOW.
* analogRead(pin): Reads the value from an analog pin.
* delay(milliseconds): Pauses the program for a specified amount of time.

**4. User-defined Functions**

You can create your own functions to encapsulate code that you will reuse throughout your program, improving modularity and readability. For example:

void blinkLED(int pin) {

digitalWrite(pin, HIGH);

delay(1000);

digitalWrite(pin, LOW);

delay(1000);

}

void setup() {

pinMode(LED\_BUILTIN, OUTPUT);

}

void loop() {

blinkLED(LED\_BUILTIN); // Blink the built-in LED

}

**5. Function Overloading**

Arduino supports function overloading, which allows you to define multiple functions with the same name but different parameters:

void display(int number) {

Serial.println(number);

}

void display(float number) {

Serial.println(number, 2); // Print with 2 decimal places

}

**6. Scope and Lifetime**

Local variables: Defined inside a function, only accessible within that function.

Global variables: Defined outside any function, accessible throughout the program.

Array and pointer

In Arduino programming, arrays and pointers are essential for managing collections of data and directly manipulating memory. Here’s an overview of how to use arrays and pointers in Arduino code, along with examples.

Arrays

Arrays are used to store multiple values of the same data type in a single variable. They provide a convenient way to manage lists of related data.

**Declaring and Initializing Arrays**

* You can declare an array by specifying its type and size. Here's an example:

int sensorValues[5]; // Declare an array of integers with 5 elements

* You can also initialize an array at the time of declaration:

int sensorValues[5] = {10, 20, 30, 40, 50}; // Initialize with values

* Accessing Array Elements
* Array elements are accessed using an index (starting from 0):

void setup() {

Serial.begin(9600);

Serial.println(sensorValues[0]); // Prints 10

sensorValues[1] = 25; // Update the second element

}

**Example: Using Arrays**

Here's a simple example that reads values from multiple analog sensors and stores them in an array:

const int numSensors = 5; // Number of sensors

int sensorValues[numSensors]; // Array to hold sensor values

void setup() {

Serial.begin(9600);

}

void loop() {

for (int i = 0; i < numSensors; i++) {

sensorValues[i] = analogRead(i); // Read from analog pins 0 to 4

Serial.print("Sensor ");

Serial.print(i);

Serial.print(": ");

Serial.println(sensorValues[i]);

}

delay(1000); // Wait 1 second before the next reading

}

Pointers

Pointers are variables that store the memory address of another variable. They are powerful tools for memory management and can be used to manipulate data more efficiently.

**Declaring and Using Pointers**

* You can declare a pointer by using the \* symbol:

int myVar = 10;

int\* ptr = &myVar; // Pointer that holds the address of myVar

**Dereferencing Pointers**

To access the value stored at the address a pointer points to, use the \* operator (dereferencing):

void setup() {

Serial.begin(9600);

Serial.print("Value of myVar: ");

Serial.println(\*ptr); // Prints 10

\*ptr = 20; // Change the value of myVar through the pointer

Serial.print("New value of myVar: ");

Serial.println(myVar); // Prints 20

}

**Example: Using Pointers with Arrays**

Pointers can also be used with arrays to manipulate data efficiently. Here’s an example that uses a pointer to traverse an array:

const int numElements = 5;

int values[numElements] = {1, 2, 3, 4, 5};

void setup() {

Serial.begin(9600);

int\* ptr = values; // Pointer to the first element of the array

for (int i = 0; i < numElements; i++) {

Serial.print("Value at index ");

Serial.print(i);

Serial.print(": ");

Serial.println(\*(ptr + i)); // Access array elements using pointer arithmetic

}

}

Key Points

* Arrays are useful for managing collections of similar data types, while pointers allow for more flexible and efficient memory management.
* You can manipulate arrays directly through pointers, which can improve performance, especially with larger datasets.
* Pointers can be complex; it’s important to ensure that they point to valid memory to avoid errors or crashes in your programs.

About pins

***Defining pins:***

Pins can be defined according to GPIO pin number or by name given on board (library required).

Ex : int led\_pin = 5;

const int led\_pin = 5; it is for not changable pin

#define led\_pin 5;

Defining unsigned int of 8 bit data . Ex : Uint8\_t

***Mode of pins:***

It can indicate purpose of pin(either input or output)

Ex : pinMode(led\_pin, OUTPUT);

pinMode(sensor\_pin, INPUT);

***Digital signal:***

It is for reading or writing digital signals,

Ex: digitalwrite(pin\_name or pin\_number, HIGH or LOW);

digitalWrite(led\_pin, HIGH);

digitalRead(IR\_sensor\_pin);

***Analog signal:***

It is for analog writing or reading.

Ex: analogRead(led\_pin);

analogWrite(potentiometer\_pin);

***PULL\_UP:***

It make the pin high as default.

Ex: int sensor\_pin = 5;

pinMode(sensor\_pin, INPUT\_PULLUP);

Serial Monitor

The Serial Monitor in the Arduino IDE (or other development environments) allows you to communicate with your Arduino board over a serial connection, usually via a USB cable. It displays data sent from the board, which can be extremely helpful for debugging and monitoring purposes. Here’s a list of commonly used Serial functions and methods in Arduino programming:

1. Begin and End Communication

1. Serial.begin(speed): Initializes the serial communication at the specified baud rate (e.g., Serial.begin(9600);).
2. Serial.end(): Ends the serial communication, freeing up the serial port.

2. Printing Data to the Serial Monitor

1. Serial.print(data): Sends data to the Serial Monitor without a newline.
2. Serial.println(data): Sends data to the Serial Monitor followed by a newline character, moving the cursor to the next line.
3. Serial.printf(format, ...): Formats and prints data (similar to printf in C/C++). Available on some boards like ESP32.
4. Serial.write(data): Sends binary data (characters or bytes) directly to the Serial Monitor. This can be a single byte, an array, or a string.

3. Reading Data from the Serial Monitor

1. Serial.available(): Returns the number of bytes available to read from the serial buffer. Useful to check if data has been received before reading it.
2. Serial.read(): Reads the next byte from the serial buffer. If no data is available, returns -1.
3. Serial.readBytes(buffer, length): Reads multiple bytes into a buffer until the specified length is reached or the buffer is empty.
4. Serial.readString(): Reads the entire incoming serial data as a string until it reaches a newline character or a timeout.
5. Serial.readStringUntil(char): Reads characters into a string until the specified terminating character is reached or timeout occurs.
6. Serial.parseInt(): Reads and parses an integer from the serial buffer.
7. Serial.parseFloat(): Reads and parses a float (decimal) from the serial buffer.

4. Setting Communication Parameters

1. Serial.setTimeout(milliseconds): Sets the maximum time (in milliseconds) to wait for data during read operations.
2. Serial.flush(): Waits for all outgoing serial data to be transmitted (useful when ensuring all data is sent before performing other tasks).

5. Serial Control and Status Functions

1. Serial.peek(): Returns the next byte in the serial buffer without removing it from the buffer.
2. Serial.availableForWrite(): Returns the amount of space available in the outgoing serial buffer (useful to check if there’s room to send more data).

**Example** Usage of Serial Functions:

void setup() {

Serial.begin(9600); // Start serial communication at 9600 baud

Serial.setTimeout(1000); // Set a timeout of 1000 ms for read operations

}

void loop() {

if (Serial.available() > 0) { // Check if data is available

int receivedInt = Serial.parseInt(); // Read an integer from Serial Monitor

Serial.print("Received: "); // Print the received integer to the Serial Monitor

Serial.println(receivedInt);

}

Serial.print("LED status: ");

Serial.println(digitalRead(LED\_BUILTIN)); // Print the status of the built-in LED

delay(1000); // Wait a second

}

Display

In Arduino projects, displays are commonly used to show data, status messages, or interactive elements. Each type of display has its own functions and libraries. Here’s an overview of popular types of displays and their primary functions:

A. Character LCDs (LiquidCrystal Library)

Character LCDs are basic text-only displays. They are available in different sizes (like 16x2 and 20x4, indicating columns x rows) and are commonly interfaced using the LiquidCrystal library.

These functions apply to character LCDs, such as 16x2 or 20x4 displays, using the LiquidCrystal library.

1. Setup & Initialization

* + LiquidCrystal lcd(rs, enable, d4, d5, d6, d7);: Initializes the LCD with specified pins.
  + lcd.begin(cols, rows);: Initializes the display with the specified number of columns and rows.

2. Writing Text and Characters

* lcd.print("text");: Prints text at the current cursor position.
* lcd.write(byte);: Writes a single character to the display (useful for custom characters).

3. Cursor Position and Display Control

* lcd.setCursor(col, row);: Sets the cursor position to the specified column and row.
* lcd.clear();: Clears the display and resets the cursor to (0,0).
* lcd.home();: Moves the cursor to the home position (0,0) without clearing the display.
* lcd.cursor();: Turns on the cursor (underline).
* lcd.noCursor();: Hides the cursor (underline).
* lcd.blink();: Makes the cursor blink.
* lcd.noBlink();: Turns off cursor blinking.
* lcd.display();: Turns the display on.
* lcd.noDisplay();: Turns the display off without clearing content.

4. Scrolling Text

* lcd.scrollDisplayLeft();: Scrolls the display content left.
* lcd.scrollDisplayRight();: Scrolls the display content right.
* lcd.autoscroll();: Enables automatic scrolling of text.
* lcd.noAutoscroll();: Disables automatic scrolling.

5. Custom Characters

* lcd.createChar(location, charmap);: Creates a custom character in one of eight memory slots (0–7).

**Example:**

#include <LiquidCrystal.h>

LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

void setup() {

lcd.begin(16, 2);

lcd.print("Hello, World!");

}

void loop() {

// Code for other operations

}

B. OLED Displays (Adafruit\_SSD1306 Library)

OLED displays (Organic Light Emitting Diode) are high-contrast, graphical displays available in various sizes, such as 128x64 and 128x32 pixels. Libraries such as Adafruit\_SSD1306 and U8g2 are often used for OLEDs.

Key Functions (Adafruit\_SSD1306 Library):

These functions are specific to OLED displays, using the Adafruit\_SSD1306 library.

1. Setup & Initialization

* display.begin(SSD1306\_SWITCHCAPVCC, address);: Initializes the display with the I2C address.
* display.clearDisplay();: Clears the display content.
* display.display();: Updates the display with any new changes.

2. Text Handling

* display.setCursor(x, y);: Sets the cursor at (x, y).
* display.setTextSize(size);: Sets the text size.
* display.setTextColor(color);: Sets the text color (e.g., SSD1306\_WHITE).
* display.print("text");: Prints text on the display.
* display.println("text");: Prints text with a new line.

3. Drawing Shapes and Pixels

* display.drawPixel(x, y, color);: Draws a pixel at (x, y).
* display.drawLine(x1, y1, x2, y2, color);: Draws a line from (x1, y1) to (x2, y2).
* display.drawRect(x, y, width, height, color);: Draws a rectangle.
* display.fillRect(x, y, width, height, color);: Draws a filled rectangle.
* display.drawCircle(x, y, radius, color);: Draws a circle.
* display.fillCircle(x, y, radius, color);: Draws a filled circle.
* display.drawRoundRect(x, y, width, height, radius, color);: Draws a rounded rectangle.
* display.fillRoundRect(x, y, width, height, radius, color);: Draws a filled rounded rectangle.
* display.drawTriangle(x0, y0, x1, y1, x2, y2, color);: Draws a triangle.
* display.fillTriangle(x0, y0, x1, y1, x2, y2, color);: Draws a filled triangle.

**Example:**

#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

#define SCREEN\_WIDTH 128

#define SCREEN\_HEIGHT 64

Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire);

void setup() {

display.begin(SSD1306\_SWITCHCAPVCC, 0x3C);

display.clearDisplay();

display.setTextSize(1);

display.setTextColor(SSD1306\_WHITE);

display.setCursor(0,0);

display.print("Hello OLED");

display.display();

}

void loop() {

// Additional code

}

C. TFT LCD Displays (Adafruit\_ILI9341 Library)

TFT (Thin-Film Transistor) displays are full-color graphical displays that allow for images, shapes, and text. They come in sizes like 1.8", 2.4", and larger. Libraries like Adafruit\_ILI9341 and TFT\_eSPI are commonly used.

Key Functions (Adafruit\_ILI9341 Library):

Functions for color TFT displays, commonly controlled by the Adafruit\_ILI9341 library.

1. Setup & Initialization

* tft.begin();: Initializes the display.
* tft.setRotation(rotation);: Sets the screen rotation (0 to 3).
* tft.invertDisplay(invert);: Inverts the display colors if invert is true.

2. Color and Text Settings

* tft.setTextSize(size);: Sets the text size.
* tft.setTextColor(color);: Sets the color for text.
* tft.setCursor(x, y);: Sets the text cursor position.
* tft.print("text");: Prints text at the cursor.
* tft.fillScreen(color);: Fills the screen with a specified color.

3. Drawing Functions

* tft.drawPixel(x, y, color);: Draws a single pixel.
* tft.drawLine(x0, y0, x1, y1, color);: Draws a line.
* tft.drawRect(x, y, w, h, color);: Draws a rectangle.
* tft.fillRect(x, y, w, h, color);: Draws a filled rectangle.
* tft.drawCircle(x, y, radius, color);: Draws a circle.
* tft.fillCircle(x, y, radius, color);: Draws a filled circle.
* tft.drawRoundRect(x, y, w, h, radius, color);: Draws a rounded rectangle.
* tft.fillRoundRect(x, y, w, h, radius, color);: Draws a filled rounded rectangle.
* tft.drawTriangle(x0, y0, x1, y1, x2, y2, color);: Draws a triangle.
* tft.fillTriangle(x0, y0, x1, y1, x2, y2, color);: Draws a filled triangle.

4. Bitmap Display

* tft.drawBitmap(x, y, bitmap, w, h, color);: Draws a bitmap image.

Example:

#include <Adafruit\_GFX.h>

#include <Adafruit\_ILI9341.h>

#define TFT\_CS 10

#define TFT\_DC 9

Adafruit\_ILI9341 tft = Adafruit\_ILI9341(TFT\_CS, TFT\_DC);

void setup() {

tft.begin();

tft.fillScreen(ILI9341\_BLACK);

tft.setTextColor(ILI9341\_WHITE);

tft.setTextSize(2);

tft.setCursor(10, 10);

tft.print("Hello TFT");

}

void loop() {

// Additional code for display operations

}

D. LED Matrix Displays (MD\_MAX72XX Library)

For LED matrices, like 8x8 or 32x8, with the MD\_MAX72XX library.

1. Setup & Initialization

* matrix.begin();: Initializes the LED matrix.
* matrix.clear();: Clears the display.
* matrix.setIntensity(intensity);: Sets the brightness.

2. Drawing and Displaying Text

* matrix.print("text");: Prints text on the matrix.
* matrix.write();: Writes buffered data to the display.
* matrix.setTextAlignment(alignment);: Sets text alignment.
* matrix.scrollText("text", PA\_LEFT, speed);: Scrolls text across the display.

3. Pixel Control

* matrix.drawPixel(x, y, state);: Lights up or turns off a pixel.
* matrix.fillScreen(state);: Turns all LEDs on or off.

Example:

#include <MD\_MAX72XX.h>

#include <SPI.h>

#define HARDWARE\_TYPE MD\_MAX72XX::FC16\_HW

#define MAX\_DEVICES 4

#define CS\_PIN 10

MD\_MAX72XX matrix = MD\_MAX72XX(HARDWARE\_TYPE, CS\_PIN, MAX\_DEVICES);

void setup() {

matrix.begin();

matrix.setIntensity(5); // Set brightness level

matrix.print("Hello");

}

void loop() {

matrix.scrollText("Scrolling Text", PA\_LEFT, 100);

}

These functions provide comprehensive control over each type of display, allowing you to handle text, graphics, animations, and custom settings tailored to your Arduino project. Let me know if you’d like further examples or explanations of any specific functions!

SD card

In Arduino, working with SD cards typically involves using the SD library. This library provides a set of functions that allow you to read from and write to SD cards, making it easy to store data like sensor readings, images, or logs.

Here’s an overview of the most commonly used functions in the Arduino SD library, along with explanations and examples.

1. SD.begin()

* Initializes communication with the SD card. This function needs to be called at the beginning of your code to set up the SD card.
* Syntax: SD.begin(chipSelectPin);
* Parameters: chipSelectPin – the pin connected to the SD card's CS (Chip Select) line, typically pin 4 or 10 on most boards.
* Returns: true if initialization was successful, false otherwise.

Example:

#include <SD.h>

void setup() {

Serial.begin(9600);

if (!SD.begin(10)) { // Assuming CS is connected to pin 10

Serial.println("SD card initialization failed!");

return;

}

Serial.println("SD card initialized successfully.");

}

void loop() {}

2. SD.open()

* Opens a file on the SD card, allowing you to read from or write to it. Use FILE\_WRITE mode to open a file for writing and FILE\_READ for reading.
* Syntax: File myFile = SD.open(filename, mode);
* Parameters:
  + filename – name of the file, e.g., "data.txt".
  + mode – FILE\_WRITE or FILE\_READ.
* Returns: A File object if the file is opened successfully; otherwise, returns false.

Example:

File dataFile = SD.open("data.txt", FILE\_WRITE);

if (dataFile) {

dataFile.println("Hello, SD card!"); // Write data to file

dataFile.close(); // Always close the file after writing

} else {

Serial.println("Error opening file.");

}

3. File.close()

* Closes the file on the SD card. It’s essential to close a file after you’re done with it to ensure all data is written to the SD card.
* Syntax: myFile.close();
* Returns: None.

Example:

File dataFile = SD.open("data.txt", FILE\_WRITE);

if (dataFile) {

dataFile.println("Writing data...");

dataFile.close(); // Closes the file to save data

}

4. File.read()

* Reads a byte from the file and moves the file pointer to the next byte. This is often used in loops to read an entire file, byte by byte.
* Syntax: int data = myFile.read();
* Returns: The next byte in the file, or -1 if no more data is available.

Example:

File dataFile = SD.open("data.txt");

if (dataFile) {

while (dataFile.available()) {

Serial.write(dataFile.read());

}

dataFile.close();

} else {

Serial.println("Error opening file.");

}

5. File.write()

* Writes data to the file. This can be used to write strings, bytes, or other data types to an SD card file.
* Syntax: myFile.write(data);
* Returns: The number of bytes written, or 0 if it fails.

Example:

File dataFile = SD.open("data.txt", FILE\_WRITE);

if (dataFile) {

dataFile.write("Writing text to SD card.");

dataFile.close();

}

6. File.available()

* Checks if there is data available for reading from the file.
* Syntax: myFile.available();
* Returns: true if there is more data to read, false otherwise.

Example:

File dataFile = SD.open("data.txt");

if (dataFile) {

while (dataFile.available()) {

Serial.write(dataFile.read());

}

dataFile.close();

}

7. File.seek()

* Moves the file pointer to a specified position, which is helpful if you need to start reading or writing from a particular point.
* Syntax: myFile.seek(position);
* Parameters: position – the byte offset where you want the pointer to move.
* Returns: true if successful, false otherwise.

Example:

File dataFile = SD.open("data.txt");

if (dataFile) {

dataFile.seek(10); // Move pointer to the 10th byte

Serial.write(dataFile.read()); // Read from the new position

dataFile.close();

}

8. SD.remove()

* Deletes a file from the SD card.
* Syntax: SD.remove(filename);
* Parameters: filename – the name of the file to delete.
* Returns: true if the file was deleted successfully, false otherwise.

Example:

if (SD.remove("oldfile.txt")) {

Serial.println("File deleted.");

} else {

Serial.println("File not found or could not be deleted.");

}

9. SD.exists()

* Checks if a file exists on the SD card.
* Syntax: SD.exists(filename);
* Parameters: filename – the name of the file to check.
* Returns: true if the file exists, false otherwise.

Example:

if (SD.exists("data.txt")) {

Serial.println("File exists.");

} else {

Serial.println("File does not exist.");

}

10. File.size()

* Returns the size of a file in bytes.
* Syntax: myFile.size();
* Returns: The size of the file in bytes.

Example:

File dataFile = SD.open("data.txt");

if (dataFile) {

Serial.print("File size: ");

Serial.println(dataFile.size());

dataFile.close();

}

**Summary**

These functions give you powerful tools for handling data storage on SD cards with Arduino. Remember to always initialize with SD.begin(), close files after you’re done with them using close(), and handle files properly to avoid corruption.

Keypad

A 4x4 keypad is a commonly used input device in Arduino projects, providing a way to enter data like numbers and commands. It’s typically organized in a grid format with 4 rows and 4 columns, which allows for 16 buttons (often labeled 0-9 and A-D). This grid-like arrangement enables efficient data entry for applications like passcodes, calculators, or any system needing user input.

**How the Keypad Works**

A matrix keypad (like the 4x4) is set up by wiring the rows and columns to the Arduino’s digital I/O pins. Each button is located at the intersection of a specific row and column. When a key is pressed, it connects the row and column at that intersection, completing a circuit. The Keypad library helps read these button presses by scanning the rows and columns to detect the pressed button.

**Interfacing the Keypad with Arduino**

The 4x4 keypad is connected to Arduino using 8 pins: 4 for rows and 4 for columns. When integrated with the Keypad library, the Arduino can check which button was pressed by reading the row and column signals.

Here’s an example of the basic wiring setup for a 4x4 keypad:

1. Row Pins: Connect the keypad’s row pins to 4 of the Arduino’s digital I/O pins.

2. Column Pins: Similarly, connect the keypad’s column pins to another 4 digital I/O pins.

3. Define the keys and map them in code using the Keypad library, allowing easy recognition of which button corresponds to which character.

Keypad Library Functions Overview

The Keypad library for Arduino offers several functions to simplify keypad interaction:

* getKey(): Checks if any key is currently pressed.
* waitForKey(): Pauses the program until a key is pressed.
* isPressed(): Tests if a specific key is pressed.
* getState(): Retrieves the state of the last key (e.g., pressed, held, or released).
* addEventListener(): Allows defining custom event responses, like a beep or LED flash on each key press.
* setHoldTime() and setDebounceTime(): Configure timing to improve responsiveness and accuracy.

**Example Code for Basic Keypad Setup**

* The following example shows how to initialize and read a 4x4 keypad:

#include <Keypad.h>

const byte ROWS = 4; // Four rows

const byte COLS = 4; // Four columns

// Define the keypad layout

char keys[ROWS][COLS] = {

{'1', '2', '3', 'A'},

{'4', '5', '6', 'B'},

{'7', '8', '9', 'C'},

{'\*', '0', '#', 'D'}

};

byte rowPins[ROWS] = {9, 8, 7, 6}; // Connect to the row pins of the keypad

byte colPins[COLS] = {5, 4, 3, 2}; // Connect to the column pins of the keypad

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

void setup() {

Serial.begin(9600);

}

void loop() {

char key = keypad.getKey();

if (key != NO\_KEY) {

Serial.println(key); // Print the pressed key to the serial monitor

}

}

In this example:

* getKey() is used to check if any key is pressed.
* Each pressed key is printed to the Serial Monitor.

Practical Applications of the Keypad

1. Security Systems: Enter passcodes for authentication.
2. Home Automation: Control devices with keypad commands.
3. Menu Navigation: Input selections for different options in a user interface.
4. Data Entry: Enter numerical data, like in a calculator project.

The 4x4 keypad is versatile and highly useful in many Arduino-based projects, providing both functionality and simplicity in user interaction.

**Here’s a comprehensive list of functions:**

1. Keypad Library Initialization

* The Keypad library must be included, and a Keypad object created to interact with the keypad.
* Code for Initialization

#include <Keypad.h>

// Define the size of the keypad

const byte ROWS = 4; // Number of rows

const byte COLS = 4; // Number of columns

// Define the characters on the keypad

char keys[ROWS][COLS] = {

{'1', '2', '3', 'A'},

{'4', '5', '6', 'B'},

{'7', '8', '9', 'C'},

{'\*', '0', '#', 'D'}

};

// Define row and column pins connected to Arduino

byte rowPins[ROWS] = {9, 8, 7, 6}; // Arduino pins for rows

byte colPins[COLS] = {5, 4, 3, 2}; // Arduino pins for columns

// Create a Keypad object

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

2. getKey()

* The getKey() function checks if any key is currently pressed. It returns the character of the key if pressed or NO\_KEY if no key is pressed.
* Syntax: char key = keypad.getKey();
* Returns: Character of the pressed key or NO\_KEY if no key is pressed.

Example:

void loop() {

char key = keypad.getKey();

if (key != NO\_KEY) {

Serial.println(key); // Prints the pressed key

}

}

3. waitForKey()

* The waitForKey() function blocks the code execution until a key is pressed, then returns the character of the pressed key.
* Syntax: char key = keypad.waitForKey();
* Returns: Character of the pressed key.

Example:

void loop() {

char key = keypad.waitForKey(); // Wait for any key press

Serial.print("Key Pressed: ");

Serial.println(key);

}

4. isPressed()

* The isPressed() function checks if a specific key is currently pressed.
* Syntax: bool isPressed = keypad.isPressed('key');
* Parameters: key - Character of the key to check (e.g., '1', 'A').
* Returns: true if the specified key is pressed, false otherwise.

Example:

void loop() {

if (keypad.isPressed('A')) {

Serial.println("Key 'A' is pressed!");

}

}

5. getState()

* The getState() function returns the current state of the last key event. The possible states are:
* IDLE - No activity.
* PRESSED - Key is pressed.
* HOLD - Key is being held down.
* RELEASED - Key has been released.
* Syntax: KeyState state = keypad.getState();
* Returns: The state of the last key event (IDLE, PRESSED, HOLD, RELEASED).

Example:

void loop() {

char key = keypad.getKey();

if (key) {

KeyState state = keypad.getState();

Serial.print("Key: ");

Serial.print(key);

Serial.print(" State: ");

Serial.println(state);

}

}

6. addEventListener()

* The addEventListener() function allows you to define a custom event handler function that executes when a key event occurs. This is useful for handling PRESSED, HOLD, and RELEASED events separately.
* Syntax: keypad.addEventListener(eventHandlerFunction);
* Parameters: eventHandlerFunction - Name of the function to handle key events.
* Returns: None.

Example: Define a custom event handler:

void setup() {

Serial.begin(9600);

keypad.addEventListener(keypadEvent);

}

void keypadEvent(KeypadEvent key) {

switch (keypad.getState()) {

case PRESSED:

Serial.print("Pressed: ");

Serial.println(key);

break;

case HOLD:

Serial.print("Hold: ");

Serial.println(key);

break;

case RELEASED:

Serial.print("Released: ");

Serial.println(key);

break;

}

}

7. setHoldTime()

* The setHoldTime() function sets how long a key needs to be held down to register a HOLD event. The time is specified in milliseconds.
* Syntax: keypad.setHoldTime(milliseconds);
* Parameters: milliseconds - Duration to consider a key held.
* Returns: None.

Example:

void setup() {

keypad.setHoldTime(1000); // Set hold time to 1 second

}

8. setDebounceTime()

* The setDebounceTime() function sets the debounce time for the keypad, helping to prevent multiple detections of a single key press due to electrical noise.
* Syntax: keypad.setDebounceTime(milliseconds);
* Parameters: milliseconds - Debounce duration.
* Returns: None.

Example:

void setup() {

keypad.setDebounceTime(50); // Set debounce time to 50 ms

}

9. getKeys()

* The getKeys() function returns the current state of all keys in a multi-keypress environment. This function enables handling of simultaneous key presses.
* Syntax: bool keyStatus = keypad.getKeys();
* Returns: true if there’s an active keypress event, false if no event.

Example:

void loop() {

if (keypad.getKeys()) {

for (int i = 0; i < LIST\_MAX; i++) {

if (keypad.key[i].stateChanged) { // Check if key state changed

Serial.print("Key: ");

Serial.print(keypad.key[i].kchar);

Serial.print(" State: ");

Serial.println(keypad.key[i].kstate);

}

}

}

}

10. Other Useful Definitions and Variables

1. LIST\_MAX

* Defines the maximum number of simultaneous keypresses the library can track (default is 10).

1. NO\_KEY

* A constant representing no key pressed.

1. keypad.key[]

* An array that stores information about each key being pressed, held, or released. Each entry contains:

1. kchar - The character of the key.
2. kstate - The state of the key (PRESSED, HOLD, RELEASED).
3. stateChanged - true if the key state changed.

**Summary of All Functions**

With these functions, you can customize and manage key inputs on a 4x4 keypad for various applications, from simple input to handling multi-keypresses and long holds.

IR REMOTE CONTROL

For using an IR remote control with Arduino, the IRremote library is the most commonly used library. It allows Arduino to receive and decode signals from an infrared remote control. Below is a comprehensive list of functions provided by the IRremote library, along with their explanations, parameters, and example code.

**Key Functions of the IRremote Library**

1. IRrecv

* The IRrecv object is created to interface with the IR receiver module connected to the Arduino.
* Syntax: IRrecv irrecv(pin);
* Parameters:
* pin: The digital pin to which the IR receiver’s output is connected.

Example:

IRrecv irrecv(11); // IR receiver connected to digital pin 11

2. enableIRIn

* This function initializes the IR receiver to start listening for incoming signals.
* Syntax: irrecv.enableIRIn();
* Parameters: None.

Example:

void setup() {

irrecv.enableIRIn(); // Start the IR receiver}

3. decode

* The decode function checks if an IR signal is received and, if so, stores the decoded data in a decode\_results variable.
* Syntax: bool success = irrecv.decode(&results);
* Parameters:
* &results: A reference to a decode\_results structure that will store the decoded signal.
* Returns: true if a signal was successfully decoded, false otherwise.

Example:

decode\_results results;

void loop() {

if (irrecv.decode(&results)) {

Serial.println(results.value, HEX); // Print received code in hexadecimal

irrecv.resume(); // Prepare to receive the next signal

}

}

4. resume

* Once a signal is received and decoded, resume clears the receiver buffer so it can receive the next signal.
* Syntax: irrecv.resume();
* Parameters: None.

Example:

void loop() {

if (irrecv.decode(&results)) {

Serial.println(results.value, HEX);

irrecv.resume(); // Ready for the next signal

}

}

5. results.value

* results.value is a property of the decode\_results structure that holds the decoded value of the received IR signal.
* Syntax: results.value
* Returns: The decoded value of the received signal (usually in hexadecimal format).

Example:

if (irrecv.decode(&results)) {

Serial.println(results.value, HEX); // Print decoded value in hex

}

6. results.decode\_type

* This property provides information about the type of encoding used by the remote. Common protocols include NEC, Sony, RC5, and RC6.
* Syntax: results.decode\_type
* Returns: The type of encoding used by the remote.

Example:

if (irrecv.decode(&results)) {

Serial.print("Encoding type: ");

Serial.println(results.decode\_type); // Print the encoding type

}

7. results.bits

* This property holds the number of bits in the received IR signal. Different remotes use different numbers of bits for each command.
* Syntax: results.bits
* Returns: The number of bits in the received IR signal.

Example:

if (irrecv.decode(&results)) {

Serial.print("Bits: ");

Serial.println(results.bits); // Print the number of bits

}

8. IRsend

* The IRsend object is used to transmit IR signals, allowing Arduino to mimic a remote control and send commands to other devices.
* Syntax: IRsend irsend;
* Parameters: None.

Example:

IRsend irsend; // Initialize an IR transmitter

9. sendNEC, sendSony, sendRC5, etc.

* These functions allow Arduino to transmit signals in various IR protocols, such as NEC, Sony, and RC5. Each protocol has its specific timing and bit structure, and these functions format the signal accordingly.
* Syntax:
* irsend.sendNEC(value, bits);
* irsend.sendSony(value, bits);
* Parameters:
* value: The code to be sent (in hexadecimal).
* bits: Number of bits in the code (depends on the protocol).

Example:

void loop() {

irsend.sendNEC(0xFFA25D, 32); // Sends NEC signal with 32 bits

delay(2000); // Wait 2 seconds between transmissions

}

10. enableIROut

* Configures the IR LED pin as an output at a specified frequency (usually 38 kHz for most IR remote controls).
* Syntax: irsend.enableIROut(frequency);
* Parameters: frequency - The frequency in kHz (e.g., 38 for NEC remotes).

Example:

void setup() {

irsend.enableIROut(38); // Set up to send IR at 38 kHz

}

**Full Example: Using an IR Remote to Control an LED**

This example demonstrates how to use an IR remote to toggle an LED on and off based on the received IR codes.

#include <IRremote.h>

const int receiverPin = 11; // Pin connected to IR receiver

const int ledPin = 13; // LED connected to pin 13

IRrecv irrecv(receiverPin); // Initialize IR receiver

decode\_results results; // Store decoded results

void setup() {

Serial.begin(9600);

irrecv.enableIRIn(); // Start the IR receiver

pinMode(ledPin, OUTPUT);

}

void loop() {

if (irrecv.decode(&results)) {

Serial.print("Received code: ");

Serial.println(results.value, HEX); // Print the received code in hexadecimal

// Check for specific IR codes to control the LED

if (results.value == 0xFFA25D) { // Example code for "Power" button

digitalWrite(ledPin, !digitalRead(ledPin)); // Toggle LED

}

irrecv.resume(); // Receive the next signal

}

}

In this example:

* The irrecv.decode(&results) checks for incoming IR signals.
* results.value stores the received IR code, which is printed and used to toggle the LED.
* 0xFFA25D is the hexadecimal code for the "Power" button on most NEC remotes, but codes vary by remote.

**Example for sending and receiving string:**

To send a string as a binary signal through an IR remote, you’ll need to:

1. Convert each character in the string to its ASCII value.

2. Convert the ASCII value to binary.

3. Send the binary representation as IR signals.

The IRremote library primarily works with specific protocols like NEC, Sony, and RC5. To send custom binary data, you would typically encode the string manually into binary bits and send those over IR, bit-by-bit, or as a whole number if the protocol allows.

Here’s a simple example of how you could convert a string into binary and send each bit through an IR transmitter.

Step 1: Convert String to Binary

First, let’s break down the string into individual characters, convert each character to binary, and store the result.

Step 2: Send Each Binary Bit through IR

In this example, we’ll use a custom encoding where each bit (1 or 0) is sent individually with a delay to distinguish bits.

#include <IRremote.h>

IRsend irsend; // Initialize IR transmitter

// Function to convert character to binary and send through IR

void sendBinaryChar(char c) {

// Convert character to ASCII value, then to binary

for (int i = 7; i >= 0; i--) {

bool bit = (c >> i) & 1; // Get the ith bit from the left

if (bit == 1) {

irsend.sendNEC(0xFFFFFF, 32); // Use a unique 32-bit code for '1'

} else {

irsend.sendNEC(0x000000, 32); // Use a unique 32-bit code for '0'

}

delay(100); // Small delay between bits

}

}

// Function to send a string through IR

void sendBinaryString(String message) {

for (int i = 0; i < message.length(); i++) {

sendBinaryChar(message[i]); // Send each character

}

}

void setup() {

Serial.begin(9600);

irsend.enableIROut(38); // Set frequency to 38 kHz

}

void loop() {

String myMessage = "Hello"; // Message to send

sendBinaryString(myMessage); // Send the message through IR

delay(2000); // Wait before sending again

}

Explanation

* sendBinaryChar(char c): This function takes a character, converts it to binary, and sends each bit as an IR signal. Here, 0xFFFFFF is sent to represent 1, and 0x000000 for 0.
* sendBinaryString(String message): This function takes a string and iterates over each character, calling sendBinaryChar to send it as binary.

Notes

1. IR Codes for Bits: The values 0xFFFFFF and 0x000000 are placeholders and should be adjusted if your receiver needs a specific pattern.

2. Timing: delay(100) is used to differentiate between bits. Adjust based on your requirements and receiver sensitivity.

3. Receiver Code: The receiving Arduino would need to interpret these codes and reconstruct the original message based on the binary bits received.

Received

Here's the code to receive the binary-encoded message sent via IR and reconstruct the original string. This code will work with an Arduino that has an IR receiver module connected to it.

The receiver will:

1. Capture incoming IR signals.

2. Decode each signal into bits (0 or 1).

3. Reconstruct the original characters from the bits and output the final string.

This example assumes the following:

* 0xFFFFFF is used to represent a binary 1.
* 0x000000 is used to represent a binary 0.

Each character in the message consists of 8 bits (standard ASCII).

Receiver Code Example

#include <IRremote.h>

const int receiverPin = 11; // IR receiver connected to pin 11

IRrecv irrecv(receiverPin); // Create an IR receiver instance

decode\_results results; // Store decoded results

String receivedMessage = ""; // Variable to store the received message

byte currentByte = 0; // Temporary storage for constructing each character

int bitCount = 0; // Counter for bits in the current character

void setup() {

Serial.begin(9600);

irrecv.enableIRIn(); // Start the IR receiver

}

void loop() {

if (irrecv.decode(&results)) { // Check if a signal is received

// Check if the received code is 0xFFFFFF (1) or 0x000000 (0)

if (results.value == 0xFFFFFF) {

currentByte = (currentByte << 1) | 1; // Shift left and add a 1

} else if (results.value == 0x000000) {

currentByte = (currentByte << 1); // Shift left and add a 0

}

bitCount++; // Increment the bit count

// If 8 bits are received, a full character is formed

if (bitCount == 8) {

receivedMessage += char(currentByte); // Convert byte to character and add to message

bitCount = 0; // Reset bit count

currentByte = 0; // Reset byte for next character

}

irrecv.resume(); // Ready to receive the next bit

}

// Print the received message periodically (every 2 seconds)

static unsigned long lastPrintTime = 0;

if (millis() - lastPrintTime > 2000) {

Serial.print("Received Message: ");

Serial.println(receivedMessage);

lastPrintTime = millis();

}

}

Explanation

* irrecv.decode(&results): Checks if an IR signal was received and decodes it.

Bit Construction:

* For each received bit, the code checks if it matches 0xFFFFFF (interpreted as 1) or 0x000000 (interpreted as 0).
* currentByte = (currentByte << 1) | 1; adds a 1 bit to currentByte.
* currentByte = (currentByte << 1); adds a 0 bit to currentByte.
* 8-Bit Character Formation: Once 8 bits are received, currentByte is converted into a character, which is then appended to receivedMessage.
* Periodic Display: The message is printed every 2 seconds.

Note

* This approach assumes a simple protocol where:
* 0xFFFFFF and 0x000000 are used consistently for binary 1 and 0.
* Timing between bits may vary based on IR transmission speed, so you may need to adjust delay timings in the transmitter code to ensure stability.

Moisture sensor

Moisture sensors are devices that measure the amount of water content in soil or another medium. They are commonly used in agriculture, gardening, and automation to ensure plants receive adequate water or to monitor conditions for specific applications. When used with a microcontroller (such as an Arduino or Raspberry Pi), moisture sensors can automate processes, like activating a pump to water plants or alerting a user when soil is dry. Here’s a breakdown of how moisture sensors work, coding functions, parameters, and examples.

**How a Moisture Sensor Works**

A typical soil moisture sensor consists of two metal probes that are inserted into the soil. These probes act as an open circuit, and when placed in the soil, the water content changes the resistance between the probes. The sensor provides either an analog or digital output that correlates with moisture level:

**Analog Output**: A continuous signal (e.g., from 0 to 1023 on a 10-bit ADC) that varies based on soil moisture. The lower the value, the drier the soil; higher values mean the soil is moist.

**Digital Output**: A simple on/off signal that turns on when the soil moisture crosses a predefined threshold.

**Common Functions for Moisture Sensor in Arduino Code**

When using a moisture sensor with an Arduino, you can use a few key functions to interact with the sensor and obtain data. Below are some commonly used functions with explanations, parameters, and examples.

1. analogRead()

* Description: This function reads the analog output of the moisture sensor on the specified pin. It returns an integer value (0-1023) that indicates the soil’s moisture level, where higher values mean higher moisture content.
* Syntax:
  + int analogRead(pin);
* Parameters:
  + pin: The analog pin on the Arduino where the sensor’s analog output is connected (e.g., A0).

Example:

int moisturePin = A0; // Pin connected to the sensor

int moistureValue = 0; // Variable to store moisture reading

void setup() {

Serial.begin(9600); // Start the Serial monitor

}

void loop() {

moistureValue = analogRead(moisturePin); // Read analog value from sensor

Serial.print("Moisture Level: ");

Serial.println(moistureValue); // Print moisture level to Serial monitor

delay(1000); // Wait 1 second before next reading

}

2. digitalRead()

* Description: If your moisture sensor has a digital output, you can use digitalRead() to read a HIGH or LOW value. This approach can be useful for simpler projects, where you only need to know if the soil is "dry" or "moist."
* Syntax:
  + int digitalRead(pin);
* Parameters:
  + pin: The digital pin on the Arduino where the sensor’s digital output is connected.

Example:

int moisturePin = 7; // Digital pin connected to the sensor

void setup() {

pinMode(moisturePin, INPUT); // Set pin as input

Serial.begin(9600);

}

void loop() {

int state = digitalRead(moisturePin); // Read digital output of sensor

if (state == HIGH) {

Serial.println("Soil is moist");

} else {

Serial.println("Soil is dry");

}

delay(1000); // Wait 1 second before next reading

}

3. map()

* Description: The map() function can be useful to map the sensor's analog readings to a different range, such as 0 to 100, representing percentage moisture level. This can make data more interpretable for users.
* Syntax:
  + long map(value, fromLow, fromHigh, toLow, toHigh);
* Parameters:
  + value: The sensor reading you want to map.
  + fromLow: The minimum value in the input range (e.g., 0).
  + fromHigh: The maximum value in the input range (e.g., 1023).
  + toLow: The minimum value in the output range (e.g., 0).
  + toHigh: The maximum value in the output range (e.g., 100).

Example:

int moisturePin = A0; // Pin connected to the sensor

int moistureValue = 0;

void setup() {

Serial.begin(9600);

}

void loop() {

moistureValue = analogRead(moisturePin);

int moisturePercent = map(moistureValue, 0, 1023, 0, 100); // Map to percentage

Serial.print("Moisture Percentage: ");

Serial.print(moisturePercent);

Serial.println("%");

delay(1000);

}

4. digitalWrite()

* Description: The digitalWrite() function sends a HIGH or LOW signal to a digital pin, which can be used to control devices like pumps or LEDs in response to the sensor readings. This is useful when you want to activate something when the soil reaches a certain moisture level.
* Syntax:
  + void digitalWrite(pin, value);
* Parameters:
  + pin: The digital pin on the Arduino to which the device is connected.
  + value: The state of the pin, either HIGH or LOW.

Example:

int moisturePin = A0; // Analog pin for moisture sensor

int pumpPin = 13; // Digital pin connected to a pump

void setup() {

pinMode(pumpPin, OUTPUT);

Serial.begin(9600);

}

void loop() {

int moistureValue = analogRead(moisturePin);

Serial.println(moistureValue);

if (moistureValue < 300) { // If soil is too dry

digitalWrite(pumpPin, HIGH); // Turn on the pump

Serial.println("Pump ON");

} else {

digitalWrite(pumpPin, LOW); // Turn off the pump

Serial.println("Pump OFF");

}

delay(1000);

}

**Combining Functions for Automation**

In many projects, these functions work together to provide automated responses based on soil moisture levels. Here’s an example where a sensor’s analog reading is used to decide when to activate a pump and LED:

int moisturePin = A0; // Analog pin for sensor

int pumpPin = 8; // Digital pin for pump

int ledPin = 9; // Digital pin for LED

void setup() {

pinMode(pumpPin, OUTPUT);

pinMode(ledPin, OUTPUT);

Serial.begin(9600);

}

void loop() {

int moistureValue = analogRead(moisturePin);

int moisturePercent = map(moistureValue, 0, 1023, 0, 100); // Convert to percentage

Serial.print("Moisture Level: ");

Serial.print(moisturePercent);

Serial.println("%");

if (moisturePercent < 30) { // If soil is less than 30% moist

digitalWrite(pumpPin, HIGH); // Turn on the pump

digitalWrite(ledPin, HIGH); // Turn on LED

Serial.println("Watering...");

} else {

digitalWrite(pumpPin, LOW); // Turn off the pump

digitalWrite(ledPin, LOW); // Turn off LED

Serial.println("Soil is moist.");

}

delay(1000);

}

In this example:

* The analog sensor reading is mapped to a percentage value.
* If the soil moisture is below 30%, the pump and LED are turned on to indicate watering.
* Otherwise, they remain off.

**Conclusion**

Moisture sensors provide a simple and effective way to measure soil moisture and automate irrigation systems. By using functions like analogRead(), digitalRead(), map(), and digitalWrite(), you can create a responsive, efficient system to ensure plants are watered appropriately.

Wifi

The WiFi.h library is a core component in Arduino programming for ESP8266 and ESP32 microcontrollers, enabling these devices to connect to WiFi networks and manage wireless connectivity functions. This library allows developers to add Internet of Things (IoT) functionality to their projects by letting devices connect to WiFi, access remote servers, and communicate with other networked devices. Here’s a breakdown of what the WiFi.h library includes and how it works:

**Key Features of WiFi.h**

1. Connection Management:

WiFi.h offers multiple methods to connect to a WiFi network, whether it's using a predefined SSID and password (WiFi.begin()), Smart Config (WiFi.beginSmartConfig()), or WPS (WiFi.beginWPSConfig()).

It also provides functions to disconnect from a network and check connection status, making it easy to manage connectivity.

2. Network Information:

The library can retrieve detailed information about the network, such as the SSID, MAC address, IP address, subnet mask, gateway, and signal strength (RSSI).

This is useful for diagnostics and ensuring that the device is properly connected to the correct network with the desired settings.

3. Static IP Configuration:

WiFi.h provides functionality for configuring a static IP address (WiFi.config()), allowing devices to maintain a consistent IP on the network, which is useful in more controlled or closed network environments.

4. Access Point Mode:

Besides connecting to existing WiFi networks, WiFi.h can configure the ESP module as a Soft Access Point (AP), enabling it to act as a standalone WiFi network. This is useful for direct communication with other devices without needing a router.

Functions like WiFi.softAP() and WiFi.softAPConfig() set up and manage the access point mode, including IP configuration and security settings.

5. Network Scanning:

The library includes functionality to scan for nearby WiFi networks (WiFi.scanNetworks()), which is helpful for determining available networks, their signal strength, and encryption type.

6. Smart Config and WPS:

Smart Config and WPS (WiFi Protected Setup) provide alternative ways to connect to WiFi networks without manually entering the SSID and password on the device. These features make the setup process user-friendly, especially for IoT applications.

**Basic Example of Connecting to WiFi**

Below is an example of how to use WiFi.h to connect an ESP32 to a WiFi network:

#include <WiFi.h> // Include the WiFi library

// Define the network credentials

const char\* ssid = "yourSSID";

const char\* password = "yourPASSWORD";

void setup() {

Serial.begin(115200); // Start serial communication for debugging

WiFi.begin(ssid, password); // Attempt to connect to WiFi

// Wait until the device is connected

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.println("Connecting to WiFi...");

}

Serial.println("Connected to WiFi");

Serial.print("IP Address: ");

Serial.println(WiFi.localIP()); // Display the local IP address

}

void loop() {

// Your main code here

}

1. WiFi.begin(ssid, password)

* Description: Connects to a WiFi network with a given SSID (network name) and password.

Example:

#include <WiFi.h>

const char\* ssid = "yourSSID";

const char\* password = "yourPASSWORD";

void setup() {

Serial.begin(115200);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.println("Connecting...");

}

Serial.println("Connected to WiFi!");

}

void loop() {}

2. WiFi.beginSmartConfig()

* Description: Starts the Smart Config process, allowing the ESP to be configured over WiFi by a mobile app.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.beginSmartConfig();

while (!WiFi.smartConfigDone()) {

delay(1000);

Serial.println("Waiting for SmartConfig...");

}

Serial.println("SmartConfig done.");

}

void loop() {}

3. WiFi.beginWPSConfig()

* Description: Starts the WPS (WiFi Protected Setup) connection.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.beginWPSConfig();

if (WiFi.status() == WL\_CONNECTED) {

Serial.println("WPS Config successful.");

}

}

void loop() {}

4. WiFi.BSSID()

* Description: Returns the MAC address of the connected access point as an array.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.begin("yourSSID", "yourPASSWORD");

while (WiFi.status() != WL\_CONNECTED) delay(1000);

Serial.println(WiFi.BSSID());

}

void loop() {}

5. WiFi.BSSIDstr()

* Description: Returns the MAC address of the connected access point as a String.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.begin("yourSSID", "yourPASSWORD");

while (WiFi.status() != WL\_CONNECTED) delay(1000);

Serial.println(WiFi.BSSIDstr());

}

void loop() {}

6. WiFi.channel()

* Description: Returns the WiFi channel of the connected network.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.begin("yourSSID", "yourPASSWORD");

while (WiFi.status() != WL\_CONNECTED) delay(1000);

Serial.println("Connected on channel: " + String(WiFi.channel()));

}

void loop() {}

7. WiFi.config(local\_ip, gateway, subnet)

* Description: Sets a static IP address.

Example:

#include <WiFi.h>

IPAddress local\_ip(192, 168, 1, 100);

IPAddress gateway(192, 168, 1, 1);

IPAddress subnet(255, 255, 255, 0);

void setup() {

Serial.begin(115200);

WiFi.config(local\_ip, gateway, subnet);

WiFi.begin("yourSSID", "yourPASSWORD");

while (WiFi.status() != WL\_CONNECTED) delay(1000);

Serial.println("Connected with static IP: " + WiFi.localIP().toString());

}

void loop() {}

8. WiFi.disconnect()

* Description: Disconnects from the current WiFi network.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.begin("yourSSID", "yourPASSWORD");

delay(5000);

WiFi.disconnect();

Serial.println("Disconnected from WiFi");

}

void loop() {}

9. WiFi.encryptionType()

* Description: Returns the encryption type of the currently connected WiFi.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.begin("yourSSID", "yourPASSWORD");

while (WiFi.status() != WL\_CONNECTED) delay(1000);

Serial.println("Encryption Type: " + String(WiFi.encryptionType()));

}

void loop() {}

10. WiFi.gatewayIP()

* Description: Returns the IP address of the gateway.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.begin("yourSSID", "yourPASSWORD");

while (WiFi.status() != WL\_CONNECTED) delay(1000);

Serial.println("Gateway IP: " + WiFi.gatewayIP().toString());

}

void loop() {}

11. WiFi.getNetworkInfo()

* Description: Retrieves details about a specified network (only works with WiFi.scan).

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

int n = WiFi.scanNetworks();

for (int i = 0; i < n; ++i) {

Serial.println("Network " + String(i) + ": " + WiFi.SSID(i) + ", Channel: " + String(WiFi.channel(i)));

}

}

void loop() {}

12. WiFi.hostByName(host, ip)

* Description: Resolves a hostname to an IP address.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

IPAddress ip;

WiFi.hostByName("example.com", ip);

Serial.println("IP Address of example.com: " + ip.toString());

}

void loop() {}

13. WiFi.hostname()

* Description: Gets or sets the hostname of the device.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.setHostname("ESP\_Device");

Serial.println("Hostname set to: ESP\_Device");

}

void loop() {}

14. WiFi.isHidden()

* Description: Checks if the network is hidden or visible.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

int n = WiFi.scanNetworks();

for (int i = 0; i < n; ++i) {

Serial.print("Network " + WiFi.SSID(i) + " is ");

Serial.println(WiFi.isHidden(i) ? "Hidden" : "Visible");

}

}

void loop() {}

15. WiFi.localIP()

* Description: Returns the local IP address of the ESP when connected to a WiFi network.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.begin("yourSSID", "yourPASSWORD");

while (WiFi.status() != WL\_CONNECTED) delay(1000);

Serial.println("Local IP: " + WiFi.localIP().toString());

}

void loop() {}

16. WiFi.macAddress()

* Description: Returns the MAC address of the ESP as a String.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

Serial.println("MAC Address: " + WiFi.macAddress());

}

void loop() {}

17. WiFi.mode()

* Description: Sets or gets the WiFi mode of the ESP (e.g., STA, AP, or both).
* Parameters:

mode: A WiFiMode\_t value (WIFI\_STA, WIFI\_AP, or WIFI\_AP\_STA).

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.mode(WIFI\_STA); // Set to Station mode

WiFi.begin("yourSSID", "yourPASSWORD");

}

void loop() {}

18. WiFi.printDiag()

* Description: Prints diagnostic information about the current WiFi configuration.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.begin("yourSSID", "yourPASSWORD");

while (WiFi.status() != WL\_CONNECTED) delay(1000);

WiFi.printDiag(Serial);

}

void loop() {}

19. WiFi.RSSI()

* Description: Returns the signal strength (RSSI) of the connected WiFi network.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.begin("yourSSID", "yourPASSWORD");

while (WiFi.status() != WL\_CONNECTED) delay(1000);

Serial.println("Signal Strength (RSSI): " + String(WiFi.RSSI()) + " dBm");

}

void loop() {}

20. WiFi.scanComplete()

* Description: Returns the number of available networks after a scan is complete.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.scanNetworks(true); // Asynchronous scan

while (WiFi.scanComplete() == -1) {

delay(1000);

Serial.println("Scanning...");

}

Serial.println("Networks found: " + String(WiFi.scanComplete()));

}

void loop() {}

21. WiFi.scanDelete()

* Description: Deletes the results of the last network scan to free memory.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.scanNetworks();

WiFi.scanDelete();

Serial.println("Scan results deleted.");

}

void loop() {}

22. WiFi.scanNetworks()

* Description: Initiates a scan to detect available WiFi networks.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

int n = WiFi.scanNetworks();

for (int i = 0; i < n; i++) {

Serial.println(WiFi.SSID(i));

}

}

void loop() {}

23. WiFi.smartConfigDone()

* Description: Checks if Smart Config is complete.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.beginSmartConfig();

while (!WiFi.smartConfigDone()) delay(1000);

Serial.println("SmartConfig done.");

}

void loop() {}

24. WiFi.softAP(ssid, password)

* Description: Sets up the ESP as a Soft Access Point (AP).
* Parameters:
* ssid: Network name for the AP.
* password: Password for the AP.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.softAP("ESP\_AP", "password123");

Serial.println("SoftAP IP: " + WiFi.softAPIP().toString());

}

void loop() {}

25. WiFi.softAPConfig()

* Description: Configures the IP settings for the Soft AP.
* Parameters:
  + local\_ip, gateway, subnet: IP configuration as IPAddress objects.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.softAPConfig(IPAddress(192, 168, 1, 1), IPAddress(192, 168, 1, 1), IPAddress(255, 255, 255, 0));

WiFi.softAP("ESP\_AP", "password123");

}

void loop() {}

26. WiFi.softAPdisconnect()

* Description: Disconnects the ESP from its Soft AP mode.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.softAP("ESP\_AP", "password123");

delay(5000);

WiFi.softAPdisconnect();

Serial.println("Soft AP disconnected.");

}

void loop() {}

27. WiFi.softAPmacAddress()

* Description: Returns the MAC address of the ESP’s Soft AP as a String.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.softAP("ESP\_AP", "password123");

Serial.println("Soft AP MAC Address: " + WiFi.softAPmacAddress());

}

void loop() {}

28. WiFi.softAPIP()

* Description: Returns the IP address of the ESP’s Soft AP.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.softAP("ESP\_AP", "password123");

Serial.println("Soft AP IP: " + WiFi.softAPIP().toString());

}

void loop() {}

29. WiFi.SSID()

* Description: Returns the SSID of the current WiFi network.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.begin("yourSSID", "yourPASSWORD");

while (WiFi.status() != WL\_CONNECTED) delay(1000);

Serial.println("SSID: " + WiFi.SSID());

}

void loop() {}

30. WiFi.status()

* Description: Returns the connection status of the WiFi.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.begin("yourSSID", "yourPASSWORD");

if (WiFi.status() == WL\_CONNECTED) {

Serial.println("Connected");

} else {

Serial.println("Not connected");

}

}

void loop() {}

31. WiFi.stopSmartConfig()

* Description: Stops the ongoing Smart Config process.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.beginSmartConfig();

delay(5000);

WiFi.stopSmartConfig();

Serial.println("Smart Config stopped.");

}

void loop() {}

32. WiFi.subnetMask()

* Description: Returns the subnet mask of the current network.

Example:

#include <WiFi.h>

void setup() {

Serial.begin(115200);

WiFi.begin("yourSSID", "yourPASSWORD");

while (WiFi.status() != WL\_CONNECTED) delay(1000);

Serial.println("Subnet Mask: " + WiFi.subnetMask().toString());

}

void loop() {}

33. WiFi.waitForConnectResult()

* Description: Waits for the WiFi

**ELECTRIC AND ELECTRONICS**

Resistor: