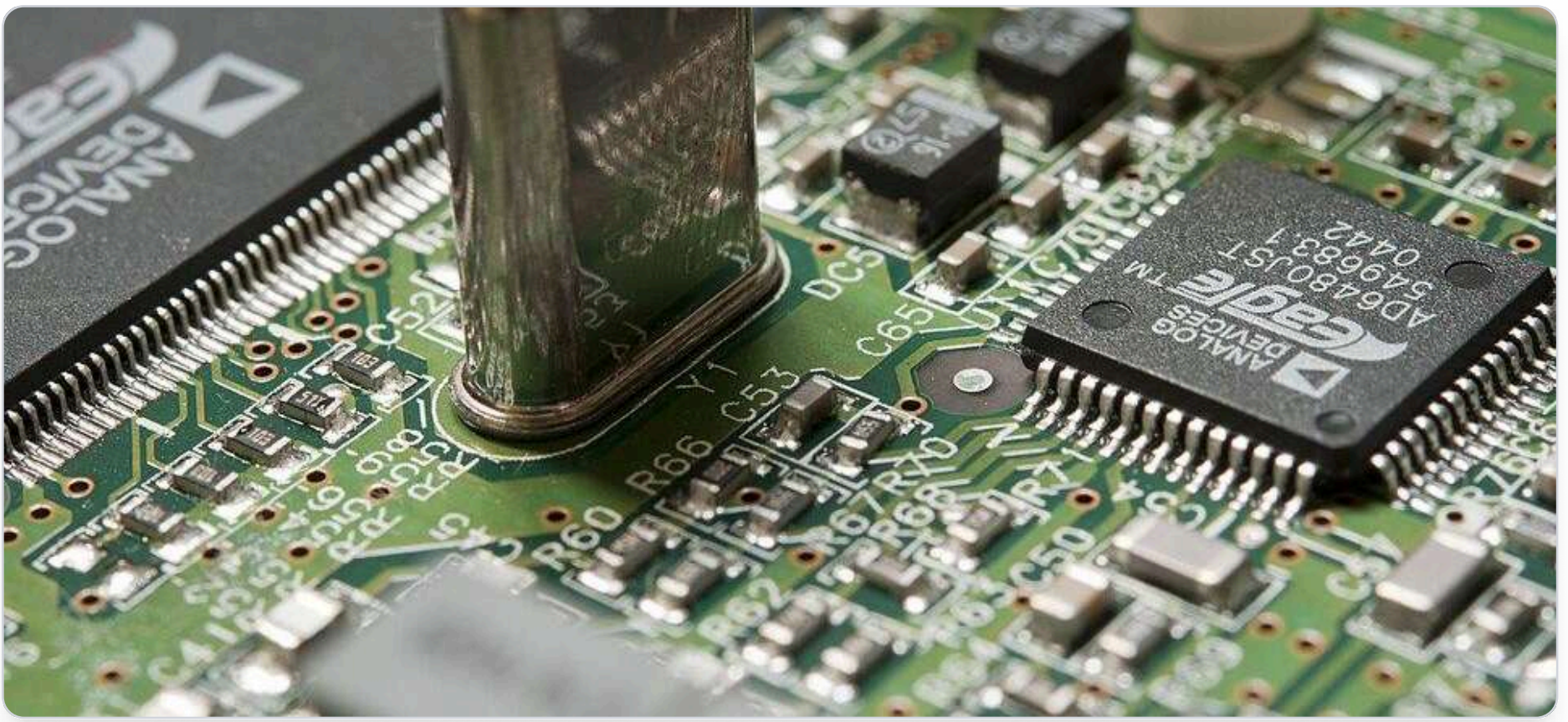
North Star
SCIENTIFIC

Embedded Systems Learning Guide

A comprehensive introduction to embedded systems development for beginners



Version 1.0 · 2026

Table of Contents

Safety & Disclaimer

Please read carefully before beginning

Electrical Safety

Proper Handling

ESD Protection

Read Instructions

DO

- Work in a clean, dry environment
- Disconnect power before making connections
- Use proper insulated tools
- Check voltage ratings before connecting
- Keep components organized and labeled
- Read all instructions before starting

DON'T

- Touch components while powered on
- Work near liquids or moisture
- Force connections that don't fit
- Exceed voltage/current ratings
- Leave circuits unattended while powered
- Skip safety precautions

Important: This kit contains electronic components that can be damaged by improper handling or static discharge. Always disconnect power before making changes.

North Star Scientific is not responsible for damage caused by misuse or failure to follow instructions. Work under adult supervision if under 18 years of age.

ESD Precautions

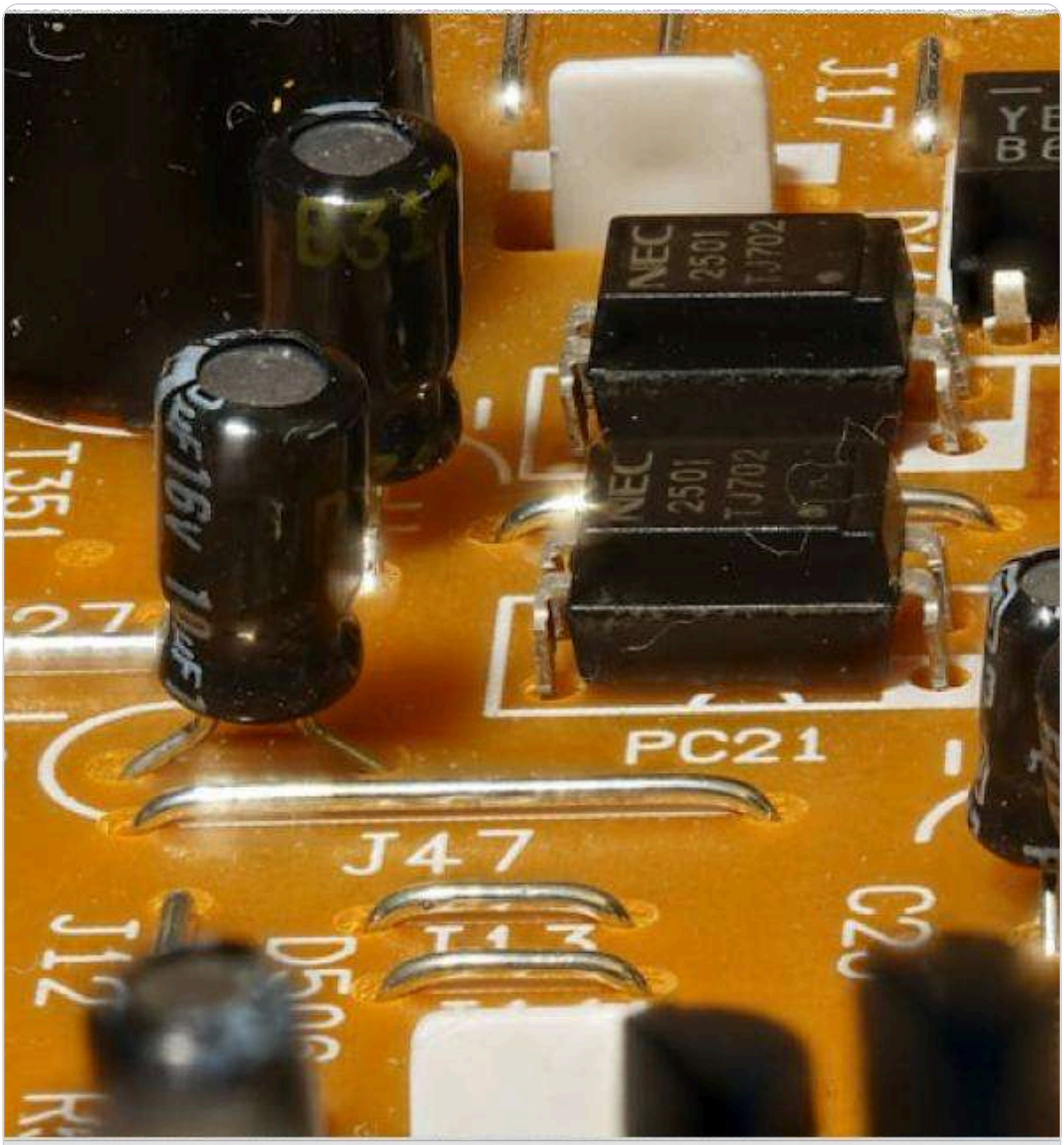
Ground yourself before handling components. Avoid working on carpeted surfaces.

Workspace Setup

Use adequate lighting and ventilation. Keep workspace organized and free of clutter.

01 Background

Understanding the fundamentals of embedded systems, their evolution, and real-world applications that shape modern technology.



Overview of Embedded Systems

What is an Embedded System?

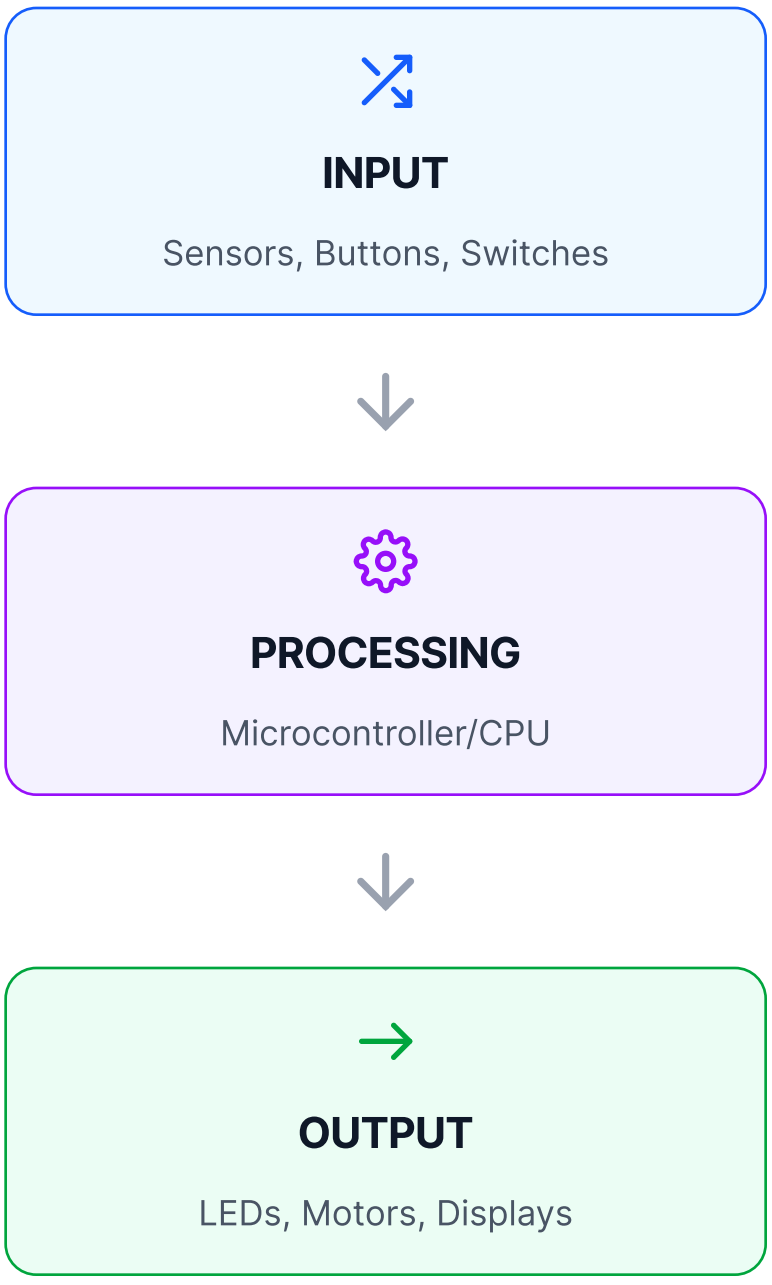
An embedded system is a specialized computer system comprised of both hardware and software designed to perform dedicated functions within a larger system. Unlike general-purpose computers, embedded systems are optimized for specific tasks.

Key Characteristics

- Real-time operation and response
- Low power consumption
- Compact size and dedicated hardware
- Task-specific programming
- Embedded software (firmware)

Did you know? Over 90% of all microprocessors manufactured are used in embedded systems, from your microwave to your car.

Basic System Flow



History of Embedded Systems

Embedded systems have evolved from room-sized computers to tiny, powerful devices that power everything from spacecraft to everyday appliances. Understanding this evolution helps us appreciate their modern capabilities.

1960s

Apollo Guidance Computer

One of the first embedded systems, used in NASA's Apollo moon missions

1971

Intel 4004 Microprocessor

First commercial microprocessor, enabling compact embedded designs

1980s

Home Automation Era

Embedded systems enter consumer electronics and appliances

2000s

IoT Revolution

Connected embedded devices become ubiquitous in daily life

2020s

AI & Edge Computing

Embedded systems with machine learning capabilities

50B+
Embedded devices worldwide (2025)

98%
Of microprocessors in embedded use

$250B
Global market value (2025)

Modern Day Embedded Systems

Today, embedded systems are everywhere. From the smartphone in your pocket to the car you drive, these specialized computers make modern life possible. Here are key application areas:

Smart Home

Thermostats, security systems, lighting control

Automotive

Engine control, ABS, infotainment systems

Medical Devices

Pacemakers, insulin pumps, monitors

Industrial

PLCs, robotics, process control

Consumer Electronics

Wearables, cameras, smart appliances

Aerospace

Flight controls, navigation, telemetry

10+

Embedded devices per person (average)

24/7

Always-on reliability required

∞

Limitless application possibilities

PARTS & COMPONENTS

Getting Started

Explore these project ideas that demonstrate real-world applications of embedded systems. Each project showcases different components and concepts you'll learn in this guide. You may use these for your own learning to grow your understanding of embedded systems after finishing our first 2 mini projects!

Smart Lighting System

Beginner

Automated room lighting with motion detection and ambient light sensing. Perfect for learning sensor integration.

Key Components:

PIR Sensor · LED · Photoresistor

Climate Monitor

Intermediate

Real-time temperature and humidity monitoring with data logging and OLED display output.

Key Components:

DHT22 · OLED · SD Card

Air Quality Sensor

Intermediate

Measures air quality levels and displays alerts on OLED screen with color-coded warnings.

Key Components:

MQ-135 · OLED · RGB LED

Smart Doorbell

Beginner

Motion-activated alert system with customizable notification sounds and distance measurement.

Key Components:

Ultrasonic · Buzzer · Button

What You'll Learn

Programming
C/C++ Basics

Circuit Design
Wiring & Layout

Sensors
Data Reading

Debugging
Troubleshooting

Integration
System Build

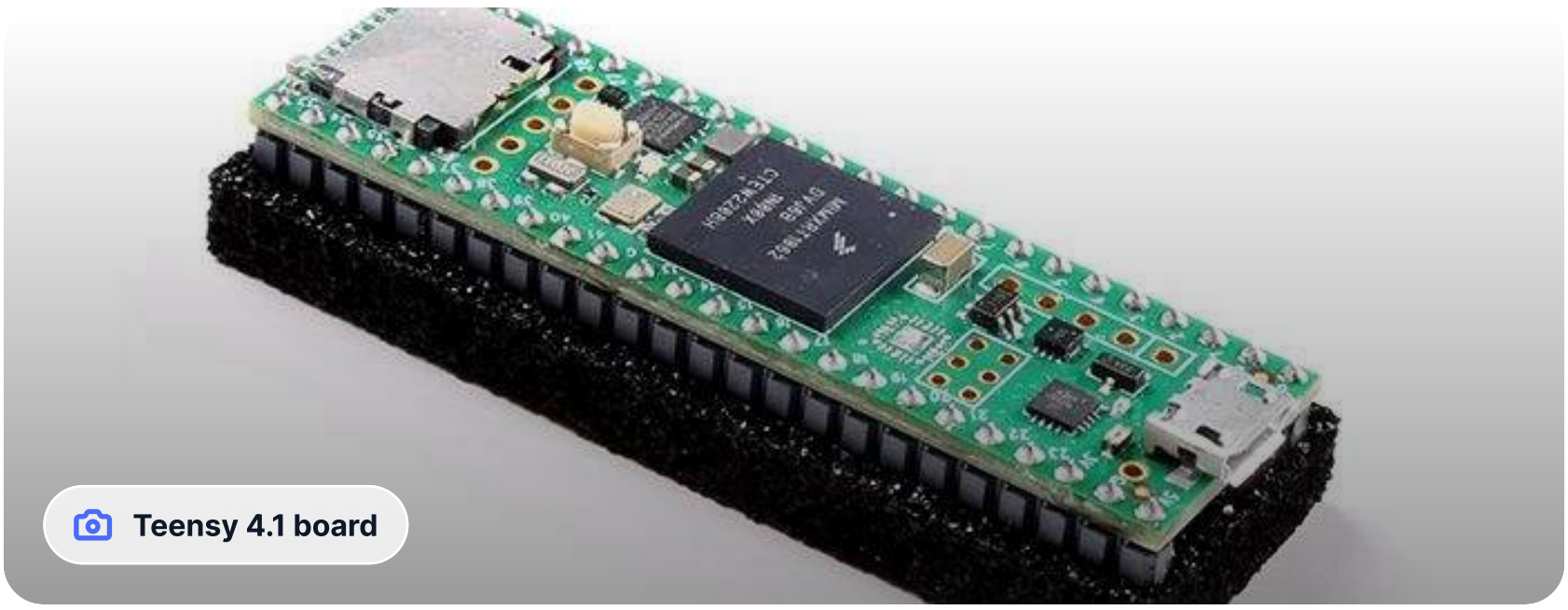
02 Components

Detailed breakdown of the Teensy microcontroller board, its specifications, and pin configuration for building your embedded projects.

Teensy Board

Featured

The Teensy 4.1 is a powerful microcontroller development board, featuring a high-speed ARM processor and extensive I/O capabilities. It's Arduino-compatible and ideal for embedded systems projects.



Teensy 4.1 board

Key Specifications

8 specs

- **Processor** ARM Cortex-M7 @ 600 MHz
- **Flash Memory** 8 MB
- **RAM** 1 MB
- **Digital I/O Pins** 40
- **Analog Inputs** 14
- **PWM Outputs** 30
- **Operating Voltage** 3.3V
- **USB** High-Speed (480 Mbit/s)

Breadboard Insertion Warning

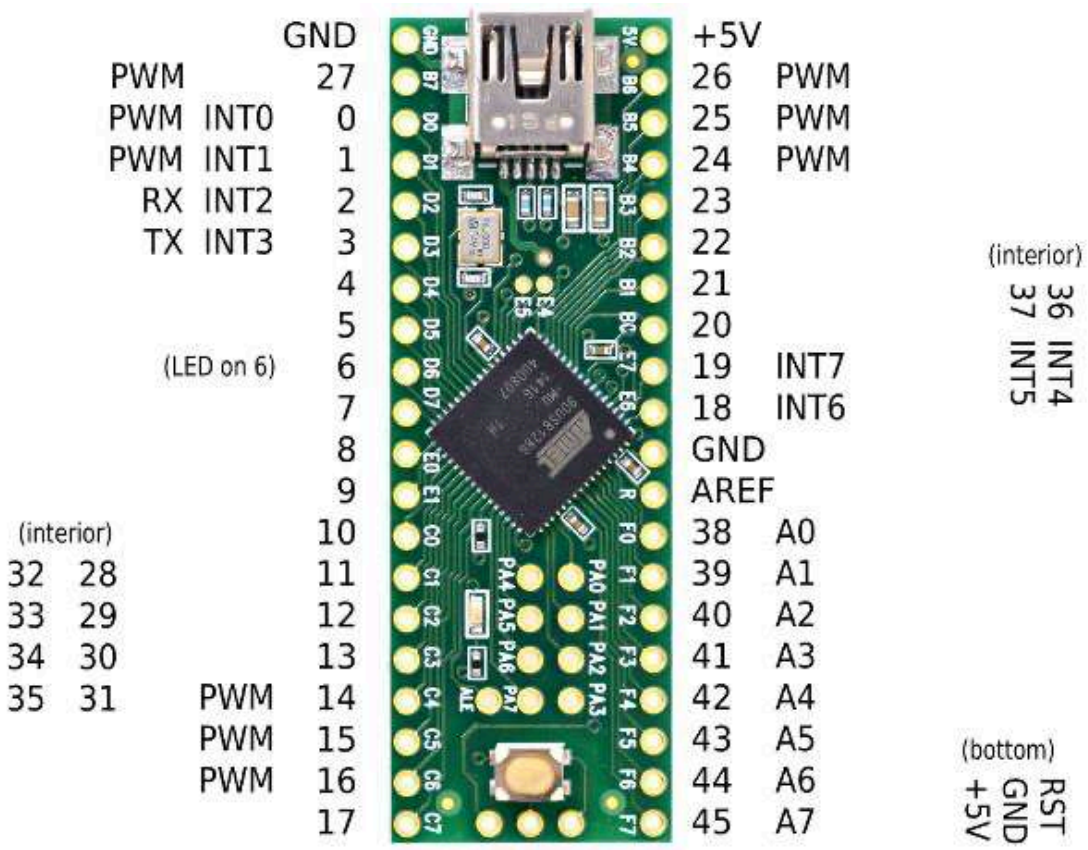
Due to the Teensy's high pin count and the slightly undersized breadboard, inserting the board fully may be difficult and can result in a less secure connection near the center. Use your best judgment when inserting the Teensy. If a secure connection is already established, avoid pressing the board in further. If the board becomes stuck, gently rock it from side to side or carefully pry it out rather than pulling forcefully.

Note

Always connect the Teensy via USB before uploading code. The board is powered through USB or external power.

Teensy Continued

Pin Configuration & Board Features



Teensy 4.1
IMXRT1062DVJ6 Cortex-M7 32 & 64 600 912
7936 66 65536
1024
4284 (emu)
32
55 42 3.3V 10mA 3.3V Only 55

Digital I/O

0-39

General purpose digital input/output pins, **3.3V** logic level

Analog Input

A0-A13

Analog-to-digital converter inputs, 12-bit resolution

PWM Output

30 total

Serial/I2C/SPI

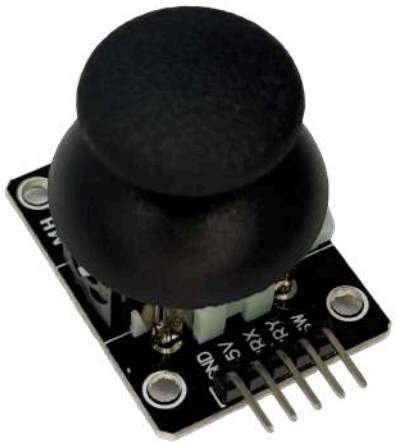
Multiple

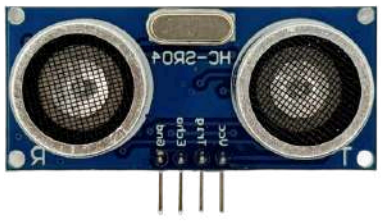
Important

- Teensy operates at 3.3V logic - do NOT connect 5V signals directly
- Maximum current per pin: 10mA (use transistor for higher loads)
- Refer to official PJRC pinout card for detailed specifications

03 Inputs

Explore the various input devices including buttons, joysticks, and sensors that allow your embedded system to interact with the physical world.





Push Button & Joystick

Push Button

A push button is a momentary contact switch that creates or breaks an electrical connection when pressed. Unlike toggle switches that maintain their state, push buttons only remain active while physical pressure is applied, making them ideal for user input, reset functions, and trigger mechanisms.

In embedded systems, push buttons are typically wired with either pull-up or pull-down resistors to ensure a defined state when not pressed. The Teensy reads the button state as a digital signal: HIGH (pressed) or LOW (not pressed), depending on the circuit configuration.

Signal Type	Voltage	Contact	Debounce
Digital ON/OFF	**3.3V logic**	**Momentary**	**Required**

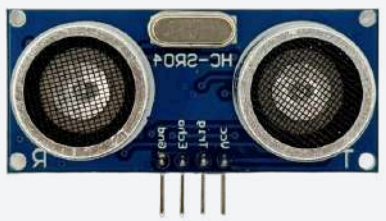
Analog Joystick

An analog joystick provides two-axis positional input through potentiometers that vary resistance based on stick position. Unlike digital buttons, joysticks offer continuous analog values, allowing for precise directional control and variable speed inputs in robotics and gaming applications.

The joystick outputs analog voltages on the X and Y axes, typically ranging from 0V (full left/down) to 3.3V (full right/up), with approximately 1.65V at center position. Most modules also include a push button activated by pressing down on the stick.

Signal Type	Voltage	Axes	Center
Analog	**0-3.3V**	**X + Y**	**1.65V**

Ultrasonic Sensor



Distance Measurement Technology

The HC-SR04 ultrasonic sensor uses sound waves to measure distance with remarkable accuracy. By emitting high-frequency sound pulses (40 kHz, beyond human hearing) and measuring the time it takes for the echo to return, the sensor can determine the distance to objects up to 4 meters away with millimeter precision.

This technology mirrors how bats and dolphins navigate using echolocation. The calculation is straightforward: since sound travels at approximately 343 meters per

```
distance = (time × 0.034) / 2
```

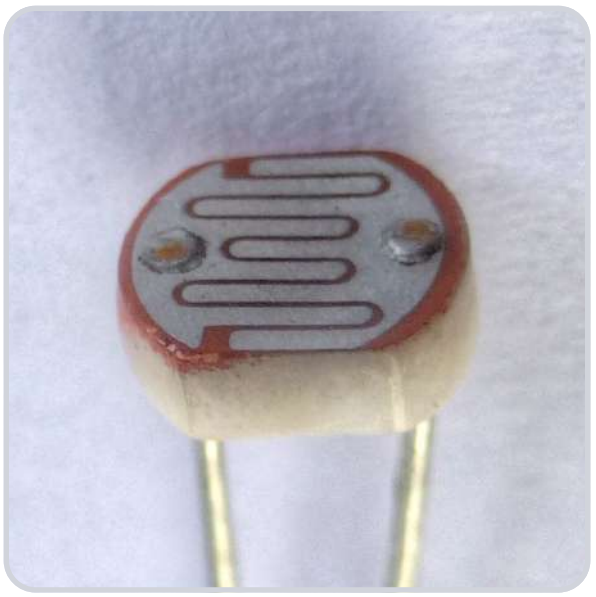
where time is measured in microseconds and the division by 2 accounts for the round-trip journey.

Voltage	Range	Accuracy	Frequency	Angle	Trigger
5V DC	2-400 cm	±3 mm	40 kHz	15° cone	10 µs

Operation Sequence

1. **Trigger Pulse**
   Send a 10µs HIGH pulse to the Trig pin to initiate measurement.
2. **Ultrasonic Burst**
   Sensor emits eight 40kHz ultrasonic pulses and waits for echo return.
3. **Echo Detection**
   Echo pin goes HIGH when pulse is sent, returns LOW when echo received.
4. **Distance Calculation**
   Microcontroller measures pulse duration and calculates distance using speed of sound.

Photoresistor



Passive Light Detection

A photoresistor, or Light-Dependent Resistor (LDR), is a component that changes its resistance based on light intensity. In the dark, it has high resistance, making it difficult for electricity to flow. When light hits its surface, it knocks electrons loose, causing the resistance to drop. Essentially, the brighter the light, the more easily current can pass through the device.

These sensors are the "eyes" for simple electronics, commonly used in streetlights that turn on at dusk or night lights that react to a dark room. By using a photoresistor in a circuit, you can allow a device to "sense" its environment and trigger actions based on brightness. They are inexpensive, reliable tools for all embedded projects.

Key Components

Protective Insulated casing

Eliminates possibility of external interference with the wiring during use

Semiconductive Wire

A wire that changes resistance based on brightness of light

Signal Processing

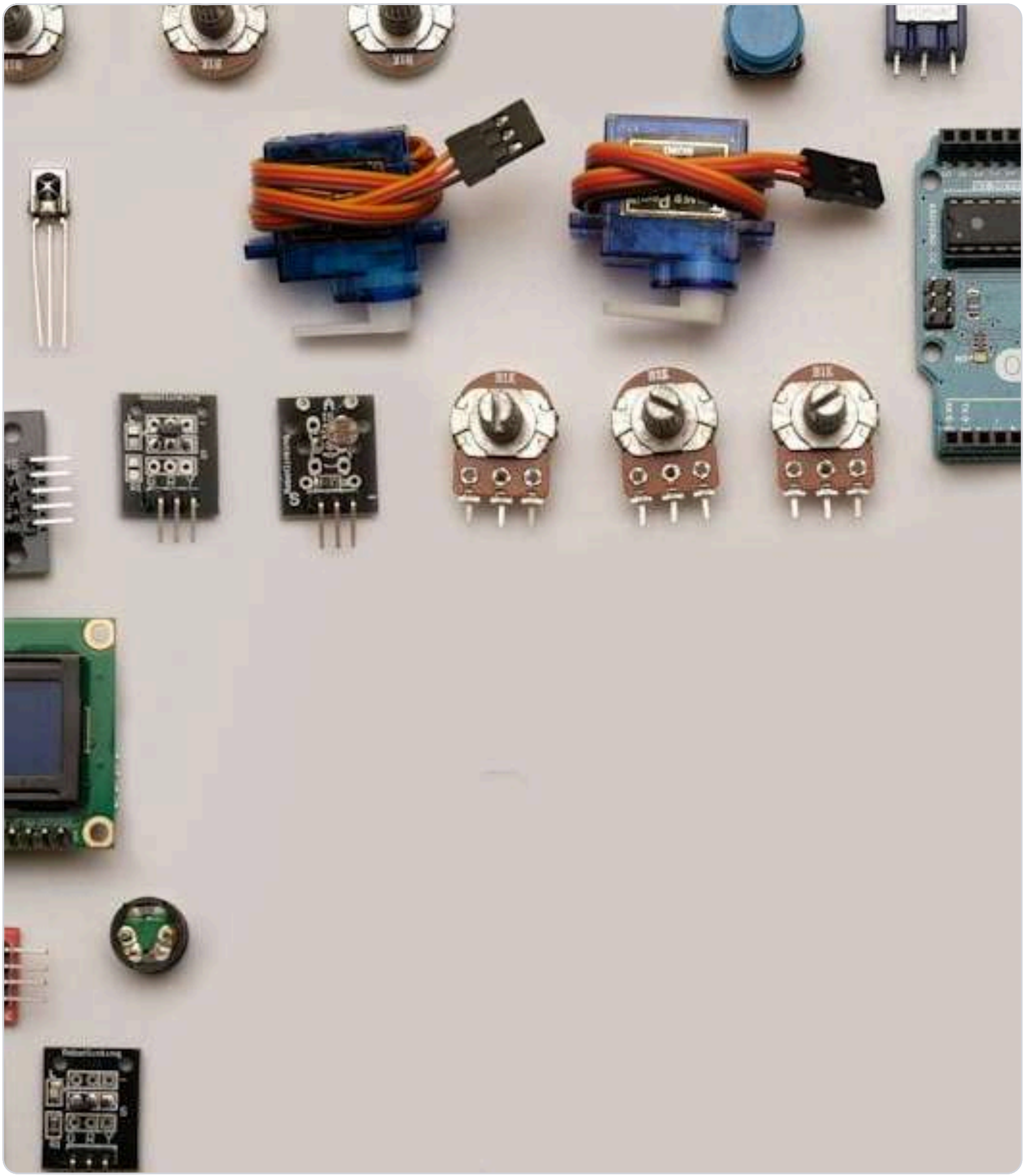
Drops in resistance changes voltage on negative side

Adjustable Parameters

Sensitivity	Voltage	Current
0-10,000 Lux	3.3-5V	5-10 mA max

04 Outputs

Learn about output devices including servos, displays, and speakers that allow your embedded system to interact with and control the physical environment.



Servo Motor

Mechanical Position Control

A servo motor is a rotary actuator that allows for control of angular position in mechanical systems. Unlike continuous rotation motors, hobby servos like the SG90 can accurately rotate to a specific angle (typically 0-180 degrees) and maintain that position. This makes them ideal for applications requiring controlled movement such as robotic arms, camera gimbals, and RC vehicles.

Servos are controlled using Pulse Width Modulation (PWM) signals. The pulse is set at a specific voltage, and the frequency of the pulse determines the movement of the angle. Power is supplied continuously to the servo, but the PMW signal closes the circuit rapidly to allow the motor to move.

PWM Control Signals

1.0 ms	1.5 ms	2.0 ms
0° Position	**90° Position**	**180° Position**
Fully counterclockwise	Center neutral angle	Fully clockwise

SG90 Specifications

Voltage	Range	Frequency	Torque	Current	Weight
4.8-6V	0-180°	50Hz	1.8 kg·cm	100-250mA	9 grams

OLED Display

High-Contrast Visual Output

OLED (Organic Light-Emitting Diode) displays are self-illuminating screens that produce bright, high-contrast images without requiring a backlight. Each pixel contains organic compounds that emit light when electrical current passes through them, allowing for true black pixels (completely off) and excellent visibility in various lighting conditions.

The SSD1306 is a monochrome OLED driver chip commonly paired with 128×64 pixel displays. It communicates via I2C or SPI protocols, requiring only two wires (plus power) for I2C communication. The driver includes built-in RAM to store the display buffer, allowing the Teensy to update graphics without constantly refreshing.

I2C Pin Connections

VCC Pin	GND Pin	SDA Pin	SCL Pin
3.3V / 5V	Ground	Pin 18 (Data)	Pin 19 (Clock)

Graphics Capabilities

Aa

Text Rendering

Multiple font sizes, scaling, wrapping

Shapes

Lines, circles, rectangles, triangles

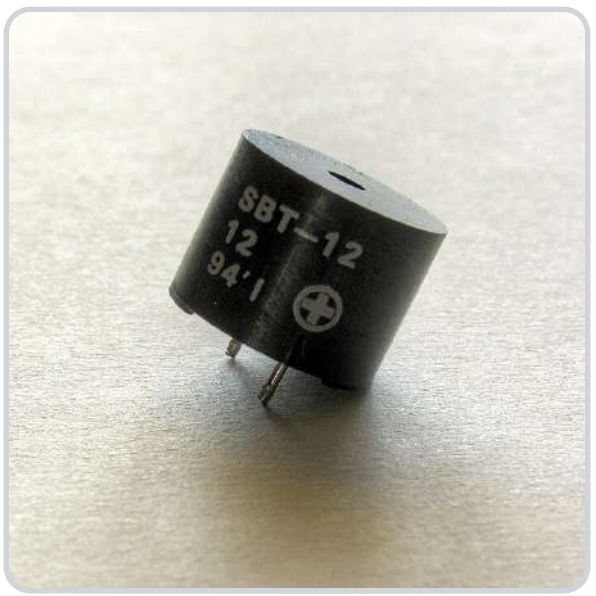
Bitmaps

Icons, logos, custom graphics

Technical Specifications

Resolution	Color	Viewing	Voltage	Current	Interface
128×64 px	Mono	>160°	3.3-5V	20 mA	I2C/SPI

Mini Speaker



Sound Generation Technology

This "cavity" speaker comes pre-installed in a tiny plastic box that acts like a miniature megaphone to amplify the sound. By trapping air behind the speaker driver, the enclosure boosts the volume and provides a much clearer sound than a bare speaker could. This self-contained design makes it a convenient "all-in-one" solution for adding audio to small gadgets without needing a bulky external cabinet.

Since an Teensy lacks the "muscle" to move the speaker cone on its own, a transistor acts as a high-speed switch to handle the heavy lifting. It uses a small signal to pulse a larger current through the speaker, turning electrical frequencies into the audible pitches you hear. By changing how fast the transistor switches on and off, you can control the exact pitch and tone of the sounds being produced.

Musical Note Frequencies

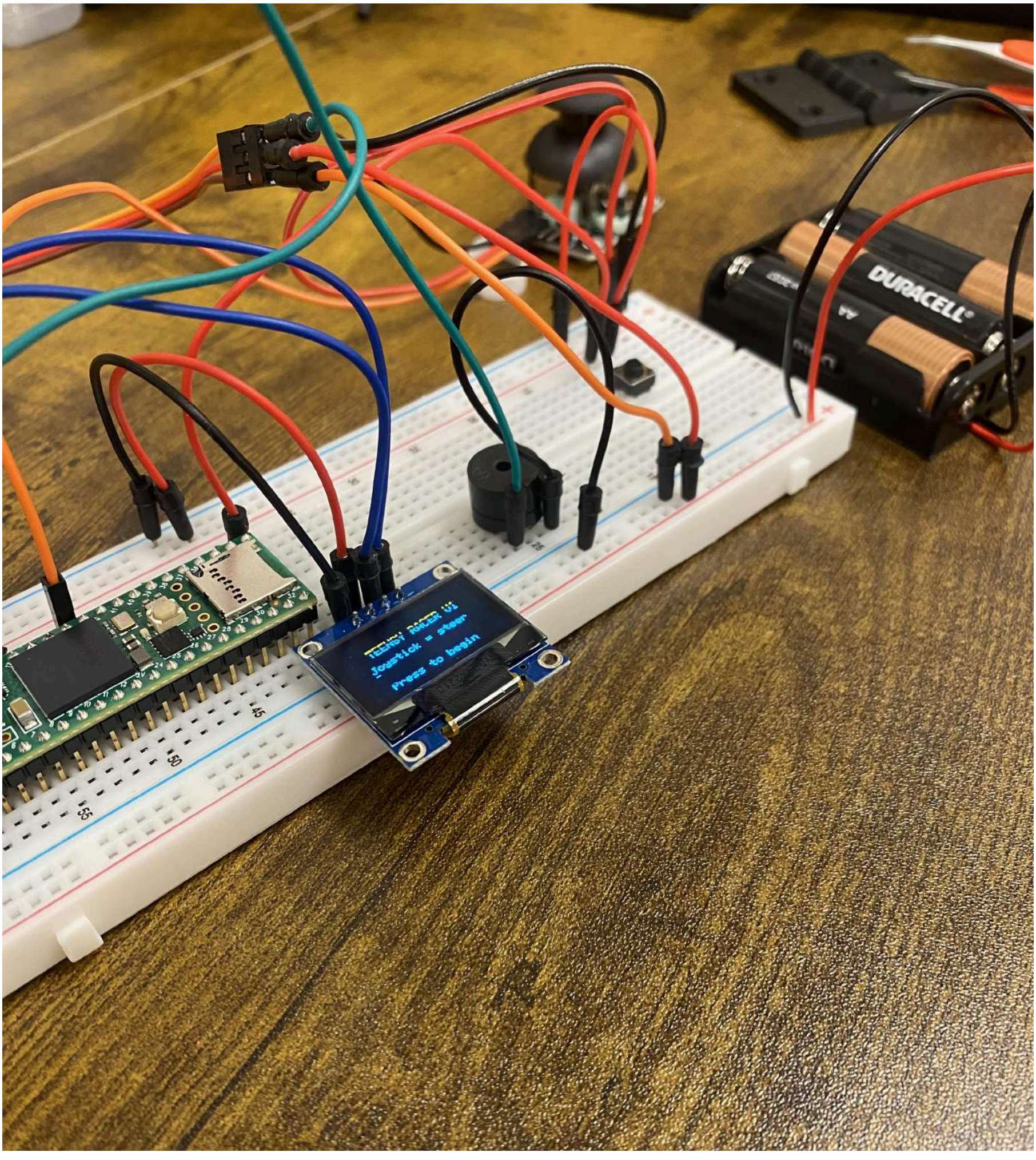
C4	D4	E4	F4	G4	A4
262 Hz	294 Hz	330 Hz	349 Hz	392 Hz	440 Hz

Technical Specifications

Voltage	Current	SPL	Optimal	Type	Polarity
3-12V	~350mA	82-85 dB	.3-15 kHz	Passive	+/–

05 Mini Project 1

Build your first hands-on project: a mini-racer game with live feedback from a servo and a buzzer.



Full Code → Page 30

Project 1: Mini-Racer Game

Project Description

In this beginner-friendly project, you'll practice wiring together components on a breadboard. Then, you will learn how to upload code to the computer. Finally, you will get to play your mini-racer game!

Learning Objectives

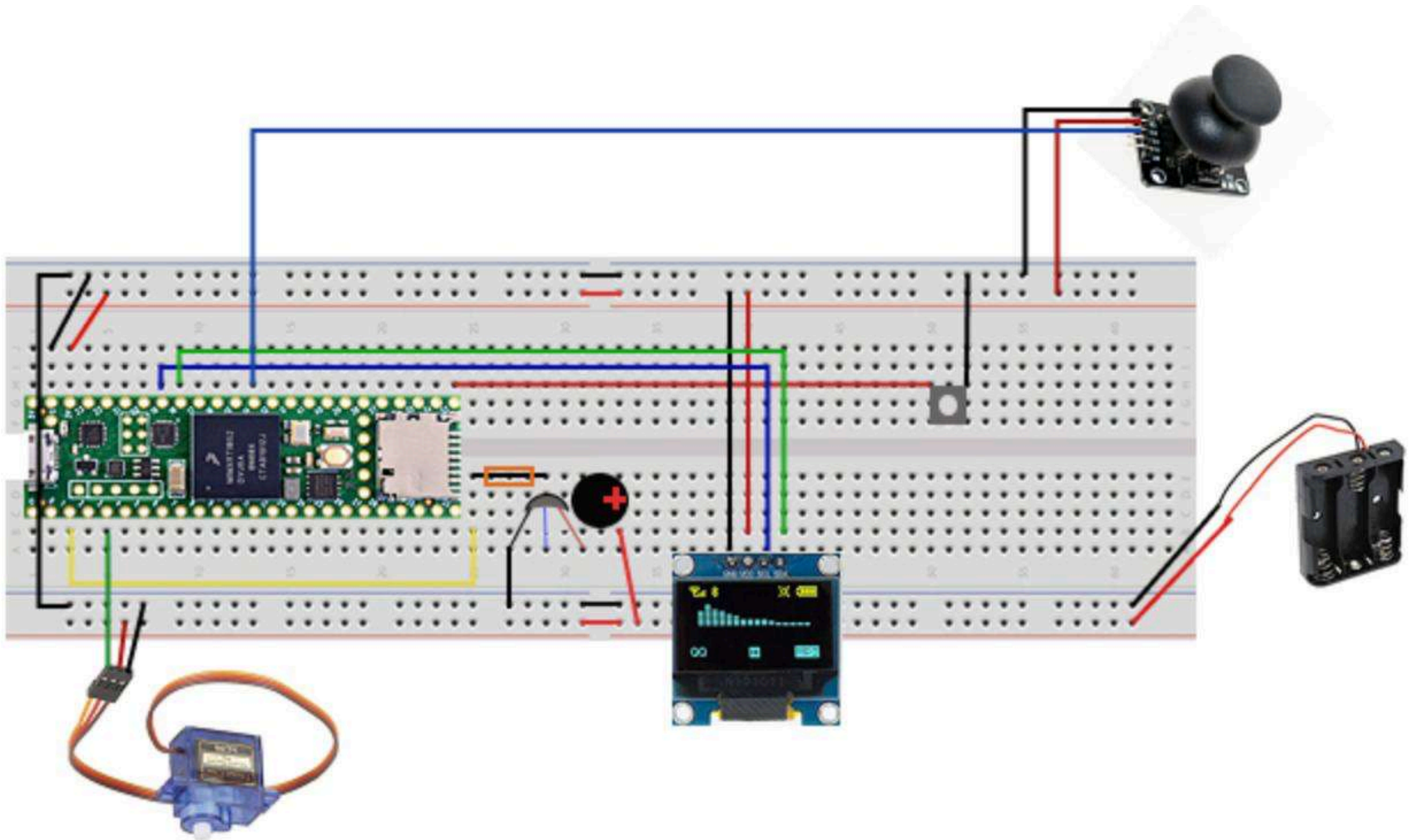
- Understanding digital input (button reading)
- Controlling digital output (LED)
- Learning software uploads using Arduino IDE
- Basics of circuits

Difficulty: Intermediate **Time:** 60 minutes

Components Needed

- ✓ Battery Pack
- ✓ MiniOLED
- ✓ Buzzer
- ✓ Joystick
- ✓ Push Button
- ✓ Servo Motor
- ✓ Jumper Wires
- ✓ Breadboard
- ✓ Orange Resistor (1kΩ)
- ✓ Transistor

Wiring Diagram:



Full Code → Page 30

Project 1: Instructions (Part 1)

Hardware Assembly

1 **Set Up Breadboard**

Place the Teensy board on the breadboard, ensuring the pins are firmly inserted into the breadboard rows.

2 **Connect Servo**

Connect dark brown wire to the lower GND rail, connect the orange wire to the PWR rail, and yellow wire to pin 3

3 **Buzzer Setup**

Insert the transistor with the flat side facing you. Connect the left pin to the GND rail. Connect Pin 1 to the transistor's middle pin through a 1 kΩ resistor. Then connect the buzzer's negative lead to the transistor's right pin and the buzzer's positive lead to the power rail.

4 **Mini OLED Display**

Connect VCC to the upper PWR rail, GND to the upper GND rail, SCL to pin 19, and SDA to pin 18

5 **Joystick Connection**

Connect joystick VCC to upper PWR rail, GND to upper GND rail, and RX to pin 14

6 **Button**

Connect one side of button to pin 33, other end to upper GND

7 **Power**

Connect Battery pack to lower PWR and GND rail, and connect Teensy 3.3V and GND to upper PWR and GND rail

8 **Ground**

Use a jumper wire to connect the upper and lower GND rails

Component Checklist

- Battery Pack
- Push button
- Servo Motor
- Breadboard
- Orange Resistor (1kΩ)
- Joystick
- Buzzer
- Mini OLED Display
- Jumper Wires
- Transistor

Important: Do not connect power until all connections are verified. Teensy pins only take 3.3V, you can damage the board if you connect any pin directly to the 5V lower PWR rail!

Full Code → Page 30

Project 1: Instructions (Part 2)

</> Programming the Teensy

9 **Download and Open Arduino IDE**

Download the latest version of the Arduino IDE from the official website. Follow the installation instructions here: https://www.pjrc.com/teensy/td_download.html

10 **Configure Board**

- Go to Tools → Board
- Select Teensy 4.1
- Set USB Type: Serial

11 **Copy & Paste**

Copy the code from the Code Appendix on the North Star Scientific Website. Make sure you use Project 1 Code.

12 **Upload to Board**

- Connect Teensy via USB
- Click Upload button (→)
- Wait for confirmation

Software Requirements

- Arduino IDE 1.8.x or 2.x
- **Teensyduino add-on (required)**
- USB drivers (auto-installed on most systems)

```
// Code Preview

#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Servo.h>

// ========================================
// V1 Racing Game Firmware for Teensy 4.1
// Revised for transistor-driven speaker on separate 5V rail
// ----------------------------------------------------------
// Pins:
// OLED SDA → 18
// OLED SCL → 19
// Joystick X-axis → 14
// Shift button → 33
// Servo PWM → 3
// Speaker control → 10  (to transistor base through 1k)
// ========================================

// -------------------- Display --------------------
constexpr int SCREEN_WIDTH  = 128;
constexpr int SCREEN_HEIGHT = 64;
constexpr int OLED_RESET    = -1;
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT

// -------------------- Pins ----------------------
constexpr int JOYSTICK_PIN = 14;
constexpr int BUTTON_PIN   = 33;
constexpr int SERVO_PIN    = 3;
constexpr int SPEAKER_PIN  = 1;   // changed from pin 1

*** See full version in Code Appendix
```

Upload Tips

- Press Teensy button if upload doesn't start
- Check correct COM port is selected
- Verify no syntax errors before uploading
- Red LED on board will blink immediately after upload

Next Steps

After uploading, you should see the Title screen for the racing game appear on the Mini OLED.

Full Code → Page 30

</> Project 1: Instructions (Part 3)

Learning Code

1 IDE

An IDE, or integrated development enviornment, is a software that lets you write and run code within one place

2 Header Files

These files act as a library that the computer reads so it knows how to perform a certain function

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
```

3 Setup Function

This function is where your computer reads the libraries you provide and gives initial values before running any actions

```
void setup() {
  pinMode(trigPin, OUTPUT); pinMode(echoPin, INPUT);
  pinMode(ledPin, OUTPUT); pinMode(buzzerPin, OUTPUT);
  Serial.begin(9600);analogReadResolution(10);
}
```

4 Loop Function

This function is where your computer runs the code and tells the connected electronics what to do

```
void loop() {
  lightValue = analogRead(photoPin);
  distance = getDistance();
  Serial.print("Light: "); Serial.print(lightValue);
  Serial.print("   Distance: "); Serial.println(distance);
```

5 Other Functions

Here are some other common functions worth exploring:

- While
- If-Else
- Switch
- Custom Functions

▷ Testing the Project

1. OLED Display should turn ON
2. Press button to start game
3. Use joystick to move character on screen left and right
4. Buzzer and Servo should activate, eventually stopping
5. Use button to “shift gears”, changing buzzer and servo position

ⓘ Troubleshooting

Mini OLED not on

Button not responding

Servo not moving

Success! You've completed your first embedded systems project. Try modifying the code to add LED blinking or multiple button inputs.

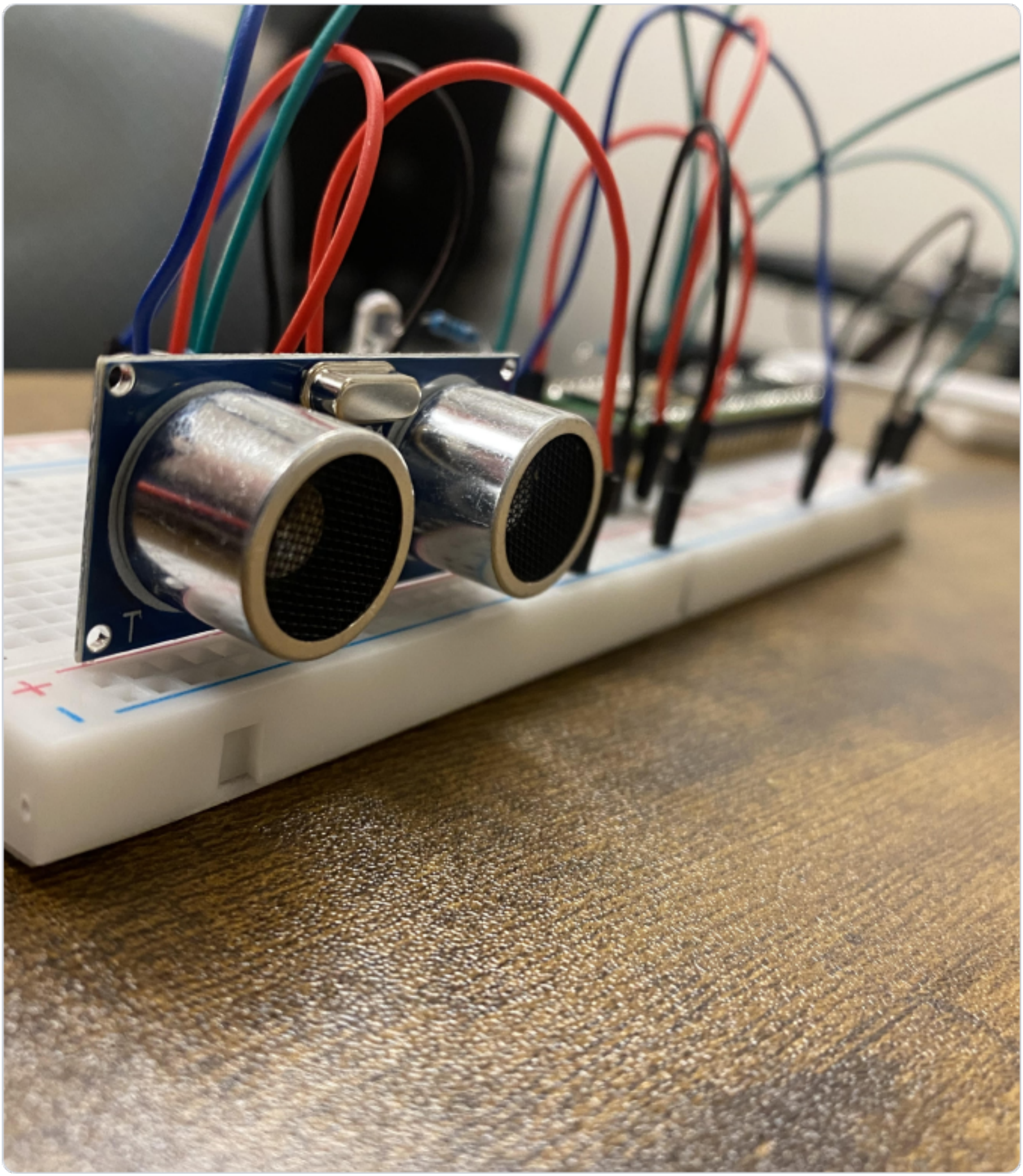
Challenge Ideas

- Add the second Joystick direction toggles +/- Y
- Make an LED blink to indicate a gear shift
- Add an acceleration physics feature like a real car

Full Code → Page 30

06 Mini Project 2

Advance your skills with a multi-component project: a passive nightlight that uses motion detection and a photoresistor so it turns on when you need it.



Full Code → Page 30

Project 2: Night Light

Project Description

Build a system that uses motion detection and the light-sensing photo resistor to turn on an LED nightlight and a small chime. Using a while loop and an if-else statement, it is an easy project to learn the basics of software code.

Learning Objectives

- Reading analog sensor data (ultrasonic)
- Conditional logic and thresholds
- Controlling multiple outputs simultaneously
- Generating audio alerts

Difficulty: Beginner **Time:** 30 minutes

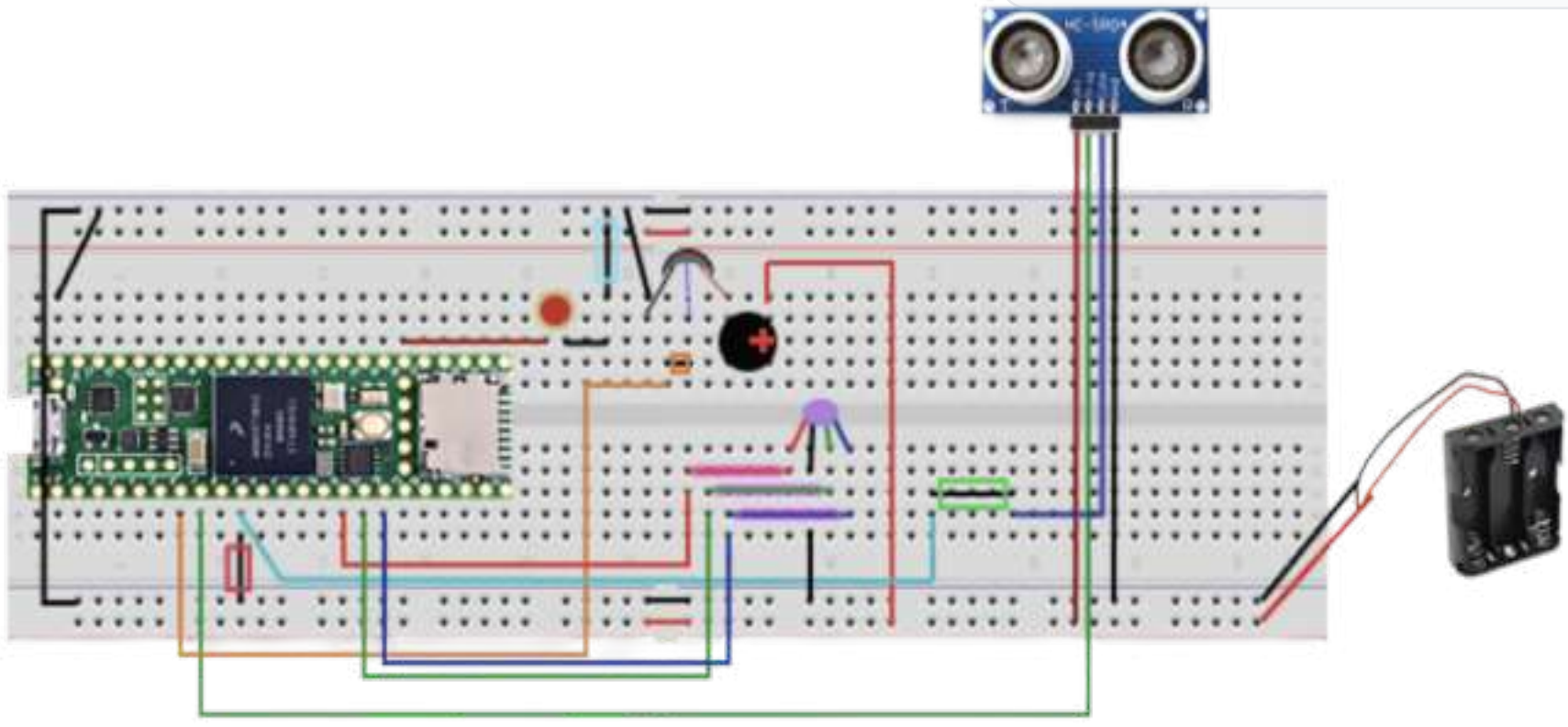
Components Needed

- ✓ Photoresistor
- ✓ Ultrasonic sensor (HC-SR04)
- ✓ Buzzer/Speaker
- ✓ LED (red)
- ✓ Resistors (see below)
- ✓ Jumper wires
- ✓ Breadboard
- ✓ Battery Pack
- ✓ Transistor

Resistor Values:

Blue: 10k Ω
Pink: 470 Ω
Green: 1.8k Ω
Red: 3.3k Ω
Orange: 1k Ω

System Diagram:



Full Code → Page 30

Project 2: Instructions (Part 1)

Hardware Assembly & Setup

1 Connect Ultrasonic Sensor

Connect VCC to 5V, GND to ground, Trig to pin 7, Echo to resistor then to pin 9. Also add the second resistor from pin 9 to GND

2 Connect Buzzer

Insert the transistor with the flat side facing you. Connect the left pin to the ground rail. Connect Pin 6 to the transistor's middle pin through a 1 kΩ resistor. Then connect the buzzer's negative lead to the transistor's right pin and the buzzer's positive lead to the lower power rail.

3 Photoresistor Setup

Connect photoresistor to A0 (pin 38) and then other end to the resistor before connecting that to GND

4 Add LED

Connect pins 24, 25, and 26 to the Red, Green, and Blue LED legs, respectively, using a 470 Ω resistor for each connection. Connect the LED's long leg to the GND rail.

5 Connect Battery

Place battery pack in bottom power rails, and put in batteries

6 Common Ground

Use a jumper wire to connect the Teensy GND pin to the ground rail. Then, connect the two ground rails with another jumper wire.

7 Upload Code

Upload the code in the following steps!

Component Checklist

- Battery Pack
- Photoresistor
- LED
- Jumper wires
- Transistor
- Ultrasonic sensor
- Buzzer
- Resistors
- Breadboard

Pin Summary

Pin 24: LED R
Pin 25: LED G
Pin 26: LED B
Pin 6: Buzzer
Pin 7: Ultrasonic Trig
Pin 9: Ultrasonic Echo
Pin 38 (A0): Photoresistor

Note: Ultrasonic sensor echo outputs 5V. Ensure voltage divider (2 resistors) is in circuit before pin 9 connection.

Full Code → Page 30

Project 2: Instructions (Part 2)

</> Programming Steps

7 Create New Sketch

Open Arduino IDE and create a new sketch. Name it "Night Light"

8 Define Pins and Constants

```
int photoPin = A0; int trigPin = 7; int echoPin = 9; int redPin = 24; int greenPin = 25; int bluePin = 26; int buzzerPin = 6; int lightValue = 0; int darkLevel = 500; int closeDistance = 50; float distance = 0;
```

9 Set Up Initial State

```
void setup() {
pinMode(trigPin, OUTPUT); pinMode(echoPin, INPUT);
pinMode(redPin, OUTPUT);
pinMode(greenPin, OUTPUT); pinMode(bluePin, OUTPUT);
pinMode(buzzerPin, OUTPUT);
// Start RGB LED off, setLED(false); Serial.begin(9600);
analogReadResolution(10);
}
```

10 Begin Sensing

```
void loop() { lightValue = analogRead(photoPin);
distance = getDistance(); Serial.print("Light: ");
Serial.print(lightValue); Serial.print(" Distance: ");
Serial.println(distance);
if (
lightValue < darkLevel &&
distance > 0 &&
distance < closeDistance) {
// Same behavior as old single-color LED: turn the light ON. All three RGB channels create white.
setLED(true);
if (!alreadyTriggered) { playLullaby(); alreadyTriggered = true;
}}
```

11 If-Else Loop

Checks the variable is true or false. If it is a true statement, the if code runs. False statement, then the else code runs.

```
if (state) {
digitalwrite(redPin, HIGH);
digitalWrite(greenPin, HIGH);
digitalWrite(bluePin, HIGH); } 

else {
digitalWrite(redPin, LOW);
digitalWrite(greenPin, LOW);
digitalWrite(bluePin, LOW);}
```

12 Custom Function

You can also define your own function:

```
float getDistance() {

long time;

digitalWrite(trigPin, LOW);
delayMicroseconds(2);

digitalWrite(trigPin, HIGH);
delayMicroseconds(10);

digitalWrite(trigPin, LOW);

time = pulseIn(
echoPin,HIGH,30000);

return time * 0.034 / 2;
}
```

Code Structure

setup(): Initialize pins and code

loop(): Measure distance, check light threshold

If-Else: Checks conditions to decide action

Action: Activate buzzer and LED when triggered

Distance Calculation

Ultrasonic sensors measure time for sound to bounce back:

```
duration = pulseIn(echoPin, HIGH);
distance = duration * 0.034 / 2;
```

Speed of sound: 343 m/s (0.034 cm/µs)

Used Functions

pinMode() - set pin input/output

Serial.print() - print to terminal

delay() - pause program

getDistance() - ultrasonic sensor

Alarm Logic

distance < threshold: buzzer ON + LED HIGH. Otherwise: both OFF.

Full Code → Page 30

</> Project 2: Instructions (Part 3)

Complete Code

```
// Nightlight System
// Photoresistor + Ultrasonic + RGB LED + Gentle Melody
// Teensy 4.1
int photoPin = A0; int trigPin = 7; int echoPin = 9; int redPin = 24; int greenPin = 25;
int bluePin = 26; int buzzerPin = 6; int lightValue = 0; int darkLevel = 500; int
closeDistance = 50; float distance = 0;
bool alreadyTriggered = false;
void setup() {
  pinMode(trigPin, OUTPUT); pinMode(echoPin, INPUT); pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT); pinMode(bluePin, OUTPUT); pinMode(buzzerPin,
OUTPUT);
  // Start RGB LED off, setLED(false); Serial.begin(9600); analogReadResolution(10);
}
void loop() { lightValue = analogRead(photoPin); distance = getDistance();
Serial.print("Light: "); Serial.print(lightValue); Serial.print("    Distance: ");
Serial.println(distance);
if (
    lightValue < darkLevel &&
    distance > 0 &&
    distance < closeDistance) {
// Same behavior as old single-color LED: turn the light ON. All three RGB channels
create white.
setLED(true);
    if (!alreadyTriggered) { playLullaby(); alreadyTriggered = true;
    }
  } else {
    setLED(false);
    noTone(buzzerPin);
    alreadyTriggered = false;
  }
  delay(100);
}
void setLED(bool state) {
  if (state) {
    digitalWrite(redPin, HIGH);
    digitalWrite(greenPin, HIGH);
    digitalWrite(bluePin, HIGH);
  } else {
    digitalWrite(redPin, LOW);
    digitalWrite(greenPin, LOW);
    digitalWrite(bluePin, LOW);
  }
}
float getDistance() {

  long time;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  time = pulseIn(
    echoPin,
    HIGH,
    30000
  );
  return time * 0.034 / 2;
}
void playLullaby() {
  tone(buzzerPin, 523, 250);
  delay(300);
  tone(buzzerPin, 587, 250);
  delay(300);
  tone(buzzerPin, 659, 300);
  delay(350);
  tone(buzzerPin, 698, 300);
  delay(350);
  tone(buzzerPin, 659, 300);
  delay(350);
  tone(buzzerPin, 587, 250);
  delay(300);
  tone(buzzerPin, 523, 350);
  delay(400);
  tone(buzzerPin, 587, 250);
  delay(300);
  tone(buzzerPin, 523, 500);
  delay(550);
  noTone(buzzerPin);
}
```

Testing

1. Upload code to Teensy via USB
2. Turn off lights and make room dark
3. Move hand near sensor (<20cm) - chime triggers and light comes on
4. Move hand away - light turns off

Troubleshooting

- No LED: Ensure right resistance
- Incorrect distance: Verify sensor wiring and power
- No alarm: Check buzzer polarity and pin connections
- Constant alarm: Adjust threshold or check sensor placement

Enhancements

Variable Tone: Change buzzer frequency based on distance (closer = higher pitch)

Bar Graph: Display distance as graphical bar on OLED screen

Adjustable Threshold: Add buttons to increase/decrease alarm distance

Congratulations! You've completed both projects and learned key embedded systems concepts. Keep experimenting and building!

CODE — Copy & paste into Arduino IDE

</> Project 1: Complete Code

Complete Code

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Servo.h>
//
============================================================
// V1 Racing Game Firmware for Teensy 4.1
// Revised for transistor-driven speaker on separate 5V rail
// ------------------------------------------------------------
// Pins:
// OLED SDA -> 18
// OLED SCL -> 19
// Joystick X-axis -> 14
// Shift button -> 33
// Servo PWM -> 3
// Speaker control -> 10 (to transistor base through 1k)
//
============================================================
// -------------------- Display --------------------
constexpr int SCREEN_WIDTH = 128;
constexpr int SCREEN_HEIGHT = 64;
constexpr int OLED_RESET = -1;
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
// -------------------- Pins ----------------------
constexpr int JOYSTICK_PIN = 14;
constexpr int BUTTON_PIN = 33;
constexpr int SERVO_PIN = 3;
constexpr int SPEAKER_PIN = 1; // changed from pin 1
// -------------------- Servo ---------------------
Servo rpmServo;
// -------------------- Timing --------------------
elapsedMillis roadTimer;
elapsedMillis soundTimer;

elapsedMillis uiTimer;
// -------------------- Game State ----------------
enum GameMode {
MODE_TITLE,
MODE_PLAYING,
MODE_CRASH
};
GameMode mode = MODE_TITLE;
// -------------------- Road Model ----------------
int roadCenter[SCREEN_HEIGHT];
// -------------------- Player --------------------
float playerX = SCREEN_WIDTH / 2.0f;
constexpr int CAR_Y = SCREEN_HEIGHT - 10;
constexpr int CAR_W = 8;
constexpr int CAR_H = 6;
// -------------------- Joystick ------------------
int joystickCenter = 512;
constexpr int JOY_DEADBAND = 40;
// -------------------- Shifting / Speed ----------
int gear = 1;
constexpr int MAX_GEAR = 4;
float rpm = 1200.0f;
const float gearMinRPM[MAX_GEAR + 1] = {0, 1000, 1500, 1700, 1900};
const float gearMaxRPM[MAX_GEAR + 1] = {0, 4200, 5200, 6200, 7200};
const float gearAccel[MAX_GEAR + 1] = {0, 11.0, 9.5, 8.0, 6.5};
const float gearDrag[MAX_GEAR + 1] = {0, 1.5, 1.3, 1.2, 1.0};
const float turnAuthority[MAX_GEAR + 1] = {0, 2.0, 2.4, 2.8, 3.2};
int roadScrollMs = 80;
// -------------------- Button Debounce -----------
bool lastButtonReading = HIGH;
bool stableButtonState = HIGH;

unsigned long lastDebounceTime = 0;
constexpr unsigned long DEBOUNCE_MS = 30;
// -------------------- Sound ---------------------
bool shiftBeepActive = false;
unsigned long shiftBeepEnd = 0;
unsigned long crashStartTime = 0;
//
============================================================
// Utility Functions
//
============================================================
```

```
int clampInt(int x, int lo, int hi) {
if (x < lo) return lo;
if (x > hi) return hi;
return x;
}
float clampFloat(float x, float lo, float hi) {
if (x < lo) return lo;
if (x > hi) return hi;
return x;
}
float mapFloat(float x, float inMin, float inMax, float outMin, float
outMax) {
if (fabs(inMax - inMin) < 0.0001f) return outMin;
return (x - inMin) * (outMax - outMin) / (inMax - inMin) + outMin;
}
//
============================================================
// Input
//
============================================================
void calibrateJoystickCenter() {
long sum = 0;
const int samples = 40;
for (int i = 0; i < samples; i++) {
sum += analogRead(JOYSTICK_PIN);
delay(5);

}
joystickCenter = sum / samples;
}
bool buttonPressedEvent() {
bool reading = digitalRead(BUTTON_PIN);
if (reading != lastButtonReading) {
lastDebounceTime = millis();
}
if ((millis() - lastDebounceTime) > DEBOUNCE_MS) {
if (reading != stableButtonState) {
stableButtonState = reading;
if (stableButtonState == LOW) {
lastButtonReading = reading;
return true;
}
}
}
lastButtonReading = reading;
return false;
}
float readSteeringInput() {
int raw = analogRead(JOYSTICK_PIN);
int delta = raw - joystickCenter;
if (abs(delta) < JOY_DEADBAND) return 0.0f;
float steer = (float)delta / 512.0f;
steer = clampFloat(steer, -1.0f, 1.0f);
return steer;
}
//
============================================================
// Road Generation
//
============================================================

void initRoad() {
int startCenter = SCREEN_WIDTH / 2;
for (int y = 0; y < SCREEN_HEIGHT; y++) {
roadCenter[y] = startCenter;
}
}
void scrollRoad() {
for (int y = SCREEN_HEIGHT - 1; y > 0; y--) {
roadCenter[y] = roadCenter[y - 1];
}
static int currentCenter = SCREEN_WIDTH / 2;
int wander = random(-3, 4);
currentCenter += wander;
currentCenter = clampInt(currentCenter, 30, SCREEN_WIDTH - 30);
roadCenter[0] = currentCenter;
}
int currentRoadWidth() {
int width = 44 - (gear - 1) * 4;
return clampInt(width, 28, 44);
}
```

CODE — Copy & paste into Arduino IDE

</> Project 1: Complete Code

Complete Code

```
bool playerOffRoad() {
int roadW = currentRoadWidth();
int center = roadCenter[CAR_Y + CAR_H / 2];
int leftEdge = center - roadW / 2;
int rightEdge = center + roadW / 2;
int carLeft = (int)(playerX - CAR_W / 2);
int carRight = (int)(playerX + CAR_W / 2);
return (carLeft < leftEdge || carRight > rightEdge);
}
//
============================================================
// Sound / Servo
//
============================================================

void speakerSelfTest() {
// Useful for immediate hardware verification
tone(SPEAKER_PIN, 1000);
delay(250);
tone(SPEAKER_PIN, 1600);
delay(250);
tone(SPEAKER_PIN, 2200);
delay(250);
noTone(SPEAKER_PIN);
delay(150);
}
void triggerShiftBeep() {
shiftBeepActive = true;
shiftBeepEnd = millis() + 100;
}
void updateSound() {
if (mode == MODE_TITLE) {
noTone(SPEAKER_PIN);
return;
}
if (mode == MODE_CRASH) {
static int crashFreq = 2200;
static unsigned long lastCrashStep = 0;
if (millis() - lastCrashStep > 40) {
lastCrashStep = millis();
crashFreq -= 180;
if (crashFreq < 700) crashFreq = 2200;
tone(SPEAKER_PIN, crashFreq);
}
return;
}
if (shiftBeepActive) {
if (millis() < shiftBeepEnd) {
tone(SPEAKER_PIN, 2800);

return;
} else {
shiftBeepActive = false;
}
}
// Higher range for tiny speaker / transistor setup
int freq = (int)mapFloat(rpm, 1000.0f, 7200.0f, 900.0f, 2600.0f);
freq = clampInt(freq, 900, 2600);
tone(SPEAKER_PIN, freq);
}
void updateServo() {
int angle = (int)mapFloat(rpm, 1000.0f, 7200.0f, 15.0f, 165.0f);
angle = clampInt(angle, 15, 165);
rpmServo.write(angle);
}
//
============================================================
// Game Logic
//
============================================================
void resetGame() {
gear = 1;
rpm = 1200.0f;
playerX = SCREEN_WIDTH / 2.0f;
roadScrollMs = 80;
initRoad();
shiftBeepActive = false;
mode = MODE_PLAYING;
}
```

```
void startCrash() {
mode = MODE_CRASH;
crashStartTime = millis();
}
void tryShiftUp() {
if (gear >= MAX_GEAR) return;

if (rpm > gearMaxRPM[gear] * 0.72f) {
gear++;
rpm *= 0.68f;
if (rpm < gearMinRPM[gear]) rpm = gearMinRPM[gear];
triggerShiftBeep();
} else {
rpm -= 250.0f;
if (rpm < 1000.0f) rpm = 1000.0f;
triggerShiftBeep();
}
}
void updatePhysics(float dt) {
rpm += gearAccel[gear] * dt * 100.0f;
rpm -= gearDrag[gear] * dt * 20.0f;
if (rpm > gearMaxRPM[gear]) {
rpm -= 10.0f * dt * 100.0f;
}
rpm = clampFloat(rpm, 1000.0f, 7200.0f);
float steer = readSteeringInput();
float speedFactor = mapFloat(rpm, 1000.0f, 7200.0f, 0.5f, 1.35f);
playerX += steer * turnAuthority[gear] * speedFactor;
playerX = clampFloat(playerX, 4.0f, SCREEN_WIDTH - 4.0f);
roadScrollMs = (int)mapFloat(rpm, 1000.0f, 7200.0f, 95.0f, 28.0f);
roadScrollMs -= (gear - 1) * 4;
roadScrollMs = clampInt(roadScrollMs, 20, 95);
}
//
============================================================
// Drawing
//
============================================================
void drawCar(int x, int y) {
display.fillRect(x - 3, y - 2, 6, 4, SSD1306_WHITE);
display.drawPixel(x - 2, y - 3, SSD1306_WHITE);

display.drawPixel(x + 2, y - 3, SSD1306_WHITE);
display.drawPixel(x - 2, y + 3, SSD1306_WHITE);
display.drawPixel(x + 2, y + 3, SSD1306_WHITE);
}
void drawRoad() {
int roadW = currentRoadWidth();
for (int y = 0; y < SCREEN_HEIGHT; y++) {
int center = roadCenter[y];
int leftEdge = center - roadW / 2;
int rightEdge = center + roadW / 2;
display.drawPixel(leftEdge, y, SSD1306_WHITE);
display.drawPixel(rightEdge, y, SSD1306_WHITE);
if ((y / 4) % 2 == 0) {
display.drawPixel(center, y, SSD1306_WHITE);
}
}
}
void drawHUD() {
display.setTextSize(1);
display.setTextColor(SSD1306_WHITE);
display.setCursor(2, 2);
display.print("G:");
display.print(gear);
display.setCursor(38, 2);
display.print("RPM:");
display.print((int)rpm);
}
void drawTitleScreen() {
display.clearDisplay();
display.setTextSize(1);
display.setTextColor(SSD1306_WHITE);

display.setCursor(22, 14);
display.println("TEENSY RACER V1");
display.setCursor(14, 30);
display.println("Joystick = steer");
display.setCursor(14, 40);
display.println("Button = shift/start");
display.setCursor(18, 54);
display.println("Press to begin");
display.display();
}
```

CODE — Copy & paste into Arduino IDE

</> Project 1: Complete Code

Complete Code

```
void drawCrashScreen() {
display.clearDisplay();
drawRoad();
drawCar((int)playerX, CAR_Y);
display.fillRect(18, 22, 92, 18, SSD1306_BLACK);
display.drawRect(18, 22, 92, 18, SSD1306_WHITE);
display.setTextSize(1);
display.setTextColor(SSD1306_WHITE);
display.setCursor(42, 27);
display.print("CRASH!");
display.setCursor(22, 44);
display.print("Press to retry");
display.display();
}
void drawGame() {
display.clearDisplay();
drawRoad();
drawCar((int)playerX, CAR_Y);
drawHUD();
display.display();

}
//
============================================================
// Setup / Loop
//
============================================================
void setup() {
pinMode(BUTTON_PIN, INPUT_PULLUP);
pinMode(SPEAKER_PIN, OUTPUT);
digitalWrite(SPEAKER_PIN, LOW);
analogReadResolution(10);
Wire.begin();
Wire.setClock(400000);
if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
while (true) {
delay(1000);
}
}
display.clearDisplay();
display.display();
rpmServo.attach(SERVO_PIN);
rpmServo.write(15);
randomSeed(analogRead(JOYSTICK_PIN) + micros());
calibrateJoystickCenter();
initRoad();
// Speaker hardware verification at boot
speakerSelfTest();
mode = MODE_TITLE;
}
void loop() {

if (buttonPressedEvent()) {
if (mode == MODE_TITLE) {
resetGame();
} else if (mode == MODE_PLAYING) {
tryShiftUp();
} else if (mode == MODE_CRASH) {
resetGame();
}
}
if (mode == MODE_PLAYING) {
static unsigned long lastPhysicsMs = millis();
unsigned long now = millis();
float dt = (now - lastPhysicsMs) / 1000.0f;
lastPhysicsMs = now;
dt = clampFloat(dt, 0.0f, 0.05f);
updatePhysics(dt);
if (roadTimer >= (unsigned long)roadScrollMs) {
roadTimer = 0;
scrollRoad();
if (playerOffRoad()) {
startCrash();
}
}
updateServo();
if (uiTimer >= 16) {
uiTimer = 0;
drawGame();
}
}
else if (mode == MODE_TITLE) {
drawTitleScreen();
rpm = 1000.0f;
updateServo();

}
else if (mode == MODE_CRASH) {
if (millis() - crashStartTime > 1200) {
drawCrashScreen();
noTone(SPEAKER_PIN);
rpmServo.write(20);
} else {
drawCrashScreen();
}
}
if (soundTimer >= 20) {
soundTimer = 0;
updateSound();
}
}
```

CODE — Copy & paste into Arduino IDE

</> Project 2: Complete Code

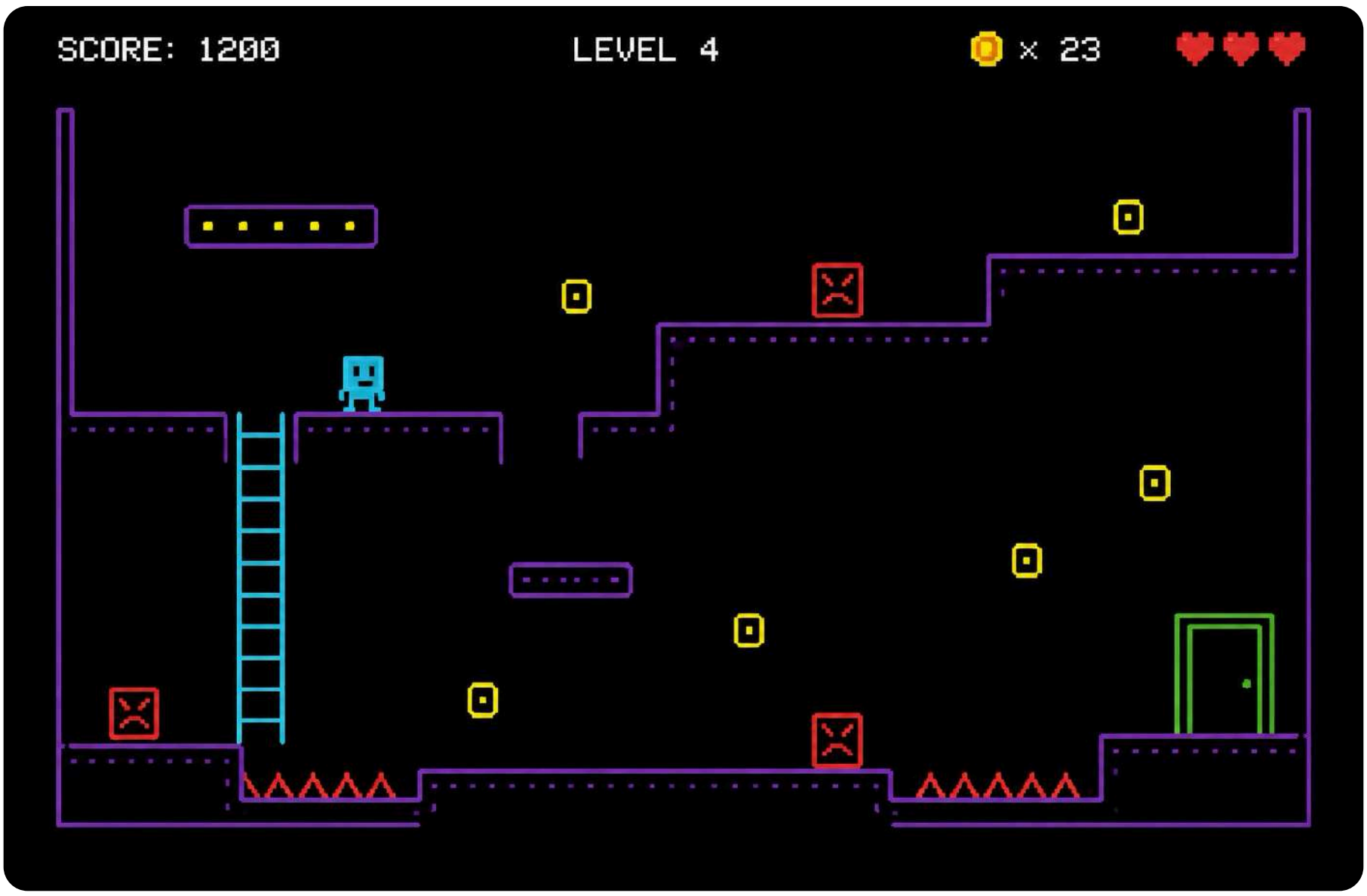
Complete Code

```
// Nightlight System
// Photoresistor + Ultrasonic + RGB LED + Gentle Melody
// Teensy 4.1
int photoPin = A0; int trigPin = 7; int echoPin = 9; int redPin = 24; int greenPin = 25;
int bluePin = 26; int buzzerPin = 6; int lightValue = 0; int darkLevel = 500; int
closeDistance = 50; float distance = 0;
bool alreadyTriggered = false;
void setup() {
  pinMode(trigPin, OUTPUT); pinMode(echoPin, INPUT); pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT); pinMode(bluePin, OUTPUT); pinMode(buzzerPin,
OUTPUT);
   // Start RGB LED off, setLED(false); Serial.begin(9600); analogReadResolution(10);
}
void loop() { lightValue = analogRead(photoPin); distance = getDistance();
Serial.print("Light: "); Serial.print(lightValue); Serial.print("   Distance: ");
Serial.println(distance);
if (
    lightValue < darkLevel &&
    distance > 0 &&
    distance < closeDistance) {
// Same behavior as old single-color LED: turn the light ON. All three RGB channels
create white.
setLED(true);
    if (!alreadyTriggered) { playLullaby(); alreadyTriggered = true;
    }
  } else {
    setLED(false);
    noTone(buzzerPin);
    alreadyTriggered = false;
  }
  delay(100);
}
void setLED(bool state) {
  if (state) {
    digitalWrite(redPin, HIGH);
    digitalWrite(greenPin, HIGH);
    digitalWrite(bluePin, HIGH);
  } else {
    digitalWrite(redPin, LOW);
    digitalWrite(greenPin, LOW);
    digitalWrite(bluePin, LOW);
  }
}
float getDistance() {

  long time;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  time = pulseIn(
    echoPin,
    HIGH,
    30000
  );
  return time * 0.034 / 2;
}
void playLullaby() {
  tone(buzzerPin, 523, 250);
  delay(300);
  tone(buzzerPin, 587, 250);
  delay(300);
  tone(buzzerPin, 659, 300);
  delay(350);
  tone(buzzerPin, 698, 300);
  delay(350);
  tone(buzzerPin, 659, 300);
  delay(350);
  tone(buzzerPin, 587, 250);
  delay(300);
  tone(buzzerPin, 523, 350);
  delay(400);
  tone(buzzerPin, 587, 250);
  delay(300);
  tone(buzzerPin, 523, 500);
  delay(550);
  noTone(buzzerPin);
}
```

06 Vibe-Code Your Own Game

Now that you have built and programmed the Mini-Racer, it is time to make something of your own! Use AI as a game-design and coding partner to turn your ideas into working Teensy code.



What You'll Learn

- Turning an idea into a simple game
- Communicating technical ideas to AI
- Connecting game mechanics to physical hardware
- Reading and modifying generated code
- Testing and debugging embedded software
- Iterating on a design after it works

Difficulty:

Beginner–Intermediate

Time Commitment:

30–90 minutes

Vibe Coding: Instructions (Part 1)

Start Your Game

1. **Keep Your Project Connected**
   Begin with the hardware from Mini Project 1 assembled and working on your breadboard.

2. **Open Your AI Assistant**
   Open ChatGPT or another AI coding assistant. Start a completely new conversation to keep the context clean.

3. **Attach This Learning Guide**
   Upload the Embedded Systems Learning Guide or reference code files directly into the conversation window.

4. **Copy the Game Designer Prompt**
   Copy and paste the provided AI Game Designer Prompt from your kit resources into the conversation.

Component Checklist

- ✓ Mini OLED Display
- ✓ Joystick
- ✓ Push Button
- ✓ Servo Motor
- ✓ Buzzer/Speaker
- ✓ Teensy 4.1

⚠ Important Notice:

Start with the existing wiring whenever possible. If AI suggests changing wiring or adding components, check the connection and voltage requirements before powering the circuit.

Pro-Tip:

Don't ask for code immediately! Designing the game first usually produces a much more interesting project.

Vibe Coding: Instructions (Part 2)

Design Your Game

5 **Answer One Question at a Time**

The AI will begin asking questions about what you want to create. There are no wrong answers. Consider:

- Game name and theme
- Player controls (buttons, joystick)
- Win/lose conditions
- Display layout (score, lives, level)
- Sound effects and feedback
- Difficulty progression
- Servo and buzzer behavior

6 **Be Creative!**

You are not limited to a simple racing game. The hardware can support many types of gaming genres.

Explore Diverse Genres:

Space battle · Dungeon explorer · Fishing game · Maze solver · Reaction challenge · Rhythm game · Monster battle · Sports game

7 **Review Your Game Design**

Before generating code, verify the AI has documented a short specification covering: Name, Goal, Controls, Mechanics, and Sound outputs.

Example Revision Prompts:

"Can you make the game slightly faster as time goes on?"

"Let's make the servo act like a speed gauge for the racer."

Vibe Coding: Instructions (Part 3)

Build the Game

8 Generate the Code

Ask AI to generate the complete Arduino/ Teensy program. Before uploading, check the pin definitions near the top.

9 Copy Into Arduino IDE

1. Open Arduino IDE
2. Create a new sketch
3. Delete the example code
4. Paste in the AI-generated program
5. Select Teensy 4.1 under Tools
6. Connect through USB
7. Click Upload

10 Test Your Game Piece by Piece

- ✓ Does the OLED turn on?
- ✓ Does the game start?
- ✓ Does the joystick register?
- ✓ Does the button work?
- ✓ Does the servo move?
- ✓ Does buzzer make sound?
- ✓ Can you play the game?
- ✓ Can you win or lose?

11 Tell AI What Happened (Bugs)

Avoid: "It doesn't work."

Good: "OLED works & game starts, but left joystick moves player right."

Better: "Code uploads ok. OLED is active and joystick responds, but X-direction is reversed."

12 Change Something / Upgrade

Another level · Increasing difficulty · High score · Lives · Power-ups · New enemies · Better sound effects · Servo animations · Start screen · Game-over animation · Secret mode

Quick Reference

Use the checklist to verify each component is responding as expected before reporting bugs to AI.

</> Testing Your Game

Pre-Flight System Check

- ✓ Game starts without compilation errors
- ✓ Joystick controls the intended player action
- ✓ Servo responds correctly during gameplay
- ✓ Winning and losing conditions work
- ✓ OLED graphics display clearly & correctly
- ✓ Button performs its designated game action
- ✓ Buzzer produces clear, intended sounds
- ✓ Game can be restarted and played again

Troubleshooting Guide

Problem	Try This Solution
Code will not compile	Copy the Arduino IDE error log and send it to AI
OLED screen is blank	Check 3.3V/GND power, SDA/SCL pins, and I2C address
Joystick moves wrong	Tell AI exactly which direction or axis is reversed
Button does nothing	Verify pin number 33 and check the ground connection
Servo does not move	Check brown GND, orange PWR, and yellow signal pin 3
Buzzer is silent	Verify wiring is strictly to pin 1 and lower GND rail
Game too fast/slow	Ask AI which delay or speed variable controls timing
Game isn't fun	Describe what feels dry and ask AI for 3 new mechanics

⚠ Core Circuit Check:

Teensy IO pins are NOT 5V tolerant. Connecting a raw 5V signal (such as raw echo output) straight to Teensy pins can instantly fry your microcontroller! Always use the voltage divider resistors where instructed.

BEST PRACTICES

Vibe Coding Tips

Mastering the art of vibe coding requires a structured approach. These six core principles will help you navigate the AI-assisted development process with clarity and efficiency.

Start Simple

Get the basic game working before adding lots of features. Build a stable foundation first.

Change One Thing at a Time

If you change five things and the game breaks, you won't know which change caused it. Test incrementally.

Describe Behavior, Not Code

You don't need to know the exact function. Tell AI what you want the physical hardware to do.

Test Often

Upload and play the game after each major change. Catch errors early before they compound.

Ask Why

When you see an interesting part of the code, ask AI to explain it. Vibe coding is a doorway to programming.

You Are the Designer

AI can suggest ideas and write lines of code, but you decide whether the game is actually fun to play.

Vibe Coding Tip

AI-generated code is not guaranteed to be correct. Always check pin numbers, never connect 5V directly to Teensy GPIO, compare wiring against the guide, and disconnect power before rewiring.

PROJECT COMPLETE

You Did It!

You successfully completed the project from idea to deployment. You've mastered the core workflow of describing your vision, building the mechanics, and testing the game on real hardware.

Your Journey

Step 1
Idea
You started with a clear game vision.

Step 2
Design
You turned the idea into game mechanics.

Step 3
Build
You connected mechanics to real hardware.

Step 4
Test
You generated software and tested it.

Step 5
Improve
You iterated until the game felt right.

Challenge Ideas

Build a Different Game
BEGINNER

Using the same hardware, explore a new genre or mechanic.

Add Difficulty Levels
INTERMEDIATE

Modify your existing codebase to scale challenge.

Physical Score Gauge
INTERMEDIATE

Use the servo to visualize health, speed, or points.

Unique Sound Effects
BEGINNER

Create sounds for win, lose, and tick events.

Hidden Game Mode
ADVANCED

Hold a button for a secret experience.

Refine with Feedback
BEGINNER

Let another person play and iterate.

Vibe Coding Tip

Keep every working version saved. As you experiment with new features, you can always roll back to a stable build.

RESOURCE

AI Game Designer Prompt

Copy and paste this prompt into ChatGPT, Claude, or your preferred AI assistant to start designing your own game.

SYSTEM PROMPT TEMPLATE

COPY PROMPT

You are my AI game designer and embedded-systems coding partner.

I am using the North Star Scientific Embedded Systems Learning Kit. Read the attached learning guide first, especially Mini Project 1 and its complete code, so you understand the available hardware, wiring, Teensy 4.1, OLED display, joystick, push button, servo, buzzer/speaker, and existing libraries.

Your goal is to help me invent and build my own fun, creative, interactive game using this hardware.

Step 1: Design the game with me
Lead me through 5–10 short questions, one at a time. Ask about things like:
• What type of game sounds fun to me
• Theme or setting
• What the player should control
• What the goal should be
• Enemies, obstacles, challenges, or puzzles
• How scoring or winning should work
• How the joystick and button could be used
• How the servo and sound could make the game more interactive
• Difficulty, levels, power-ups, surprises, or special mechanics
Be very creative and open-ended. Do not limit ideas to racing games. If I don't know an answer, always give me 3–5 specific, interesting ideas I can choose from. Examples could include spaceship battles, dungeon games, fishing games, survival games, reaction games, maze games, rhythm games, monster battles, sports games, treasure hunts, stealth games, or completely original ideas.

Step 2: Turn the idea into a game
After you understand my idea, briefly show me:
• Game Name
• Concept
• Controls
• Goal
• Game Mechanics
• How each piece of hardware is used
Let me approve or change the idea before coding.

Step 3: Vibe-code it
Once I approve it, generate the complete Arduino/Teensy code needed to play the game. Use the attached guide's existing hardware setup, pins, libraries, display configuration, and wiring wherever possible. Do not require extra components unless you clearly label them as optional.

Make the code:
• Complete and copy/paste ready
• Beginner-friendly
• Commented clearly
• Fun and polished rather than just a technical demo
• Designed to run smoothly on the Teensy 4.1
• Compatible with the kit's OLED, joystick, button, servo, and buzzer/speaker

Then give me very short instructions for:
• Copying the code into Arduino IDE
• Uploading it
• Playing the game
• Fixing the most likely problems

After it works, suggest 3 cool upgrades I could ask you to add next.

Start by reading the guide and then ask me Question 1.

HARDWARE SETUP

Wiring Your Game Controller

Fill in your wiring below, then screenshot or copy this table and paste it into your AI assistant. This tells the AI exactly how your hardware is connected so the generated code works with your setup.

→ Send this to your AI assistant — Copy & paste

GPIO Pin Assignments

9 CONNECTIONS

COMPONENT / SIGNAL	TEENSY PIN	TYPE / MODE
Joystick X-Axis	A0	Analog Input
Joystick Y-Axis	A1	Analog Input
Joystick Button	Pin 2	Digital Input (Pull-up)
OLED Display SDA	Pin 18	I2C Data
OLED Display SCL	Pin 19	I2C Clock
Buzzer	Pin 10	PWM Output
Button A (Action)	Pin 3	Digital Input (Pull-up)
Button B (Start)	Pin 4	Digital Input (Pull-up)
LED Indicator	Pin 13	Digital Output

Setup Checklist

- ✓ Double-check all wiring before powering on
- ✓ Use pull-up resistors for button inputs
- ✓ Test each component individually before combining

Example prompt

Example prompt: "Here is my wiring setup: Joystick X on A0, Joystick Y on A1, Buzzer on Pin 10, OLED on pins 18/19. Write me a Snake game that uses these exact pins."

North Star
SCIENTIFIC

Want More Hands-On Learning Kits?

Explore more electronics kits, robotics projects, and STEM learning resources.

nstarscientific.com

Join our community on Discord →