KEMENTERIAN PENDIDIKAN TINGGI
JABATAN PENDIDIKAN POLITEKNIK DAN KOLEJ KOMUNITI

POLITEKNIK MALAYSIA
MERLIMAU

EXPLORING EMBEDDED SYSTEMS

Hands-on with Arduino & Augmented Reality

ZAIN RETAS

EXPLORING EMBEDDED SYSTEMS

Hands-on with Arduino & Augmented Reality

ZAIN RETAS

First Edition 2025

ZAIN RETAS

Publish by:
Politeknik Merlimau Melaka
KM 15, Jalan Jasin
77300 Merlimau Melaka

Tel : 06-2636687
Fax: 06-2636678
Website: www.pmm.mypolycc.edu.my

EDITORIAL BOARD

ACKNOWLEDGEMENT

This eBook, "EMBEDDED SYSTEMS: Hands-on with Arduino & Augmented Reality", was made possible through the dedication, support, and collaboration of many individuals and organizations.

First and foremost, I would like to express my sincere gratitude to the Jabatan Kejuruteraan Mekanikal, Politeknik Merlimau Melaka for their unwavering encouragement and for fostering a supportive teaching and learning environment. Special appreciation goes to my studentsyour enthusiasm, feedback, and active participation in the flipped classroom sessions were the driving force behind the development of this eBook.

I would also like to extend my appreciation to Bahagian Instruksional dan Pembangunan Digital (BIPD) and Digital Learning Centre for providing the platform, tools, and support that enabled the transformation of conventional teaching into a more engaging, student centered learning experience.

A heartfelt thank you to my colleagues, mentors, and peer reviewers whose valuable insights and suggestions helped enhance the quality and relevance of this work.

Lastly, to all learners embarking on your embedded systems journeymay this eBook empower and inspire you to explore, create, and innovate with curiosity and confidence.

PREFACE

This eBook, "EMBEDDED SYSTEMS: Hands-on with Arduino & Augmented Reality", is made to help students and beginners learn about embedded systems in a fun and practical way.

Today, smart machines and automation are part of our daily lives. Learning how these systems work is important. In this eBook, you will learn step by step how to build simple projects using Arduino and even use augmented reality (AR) to make your learning more exciting.

This book follows the flipped classroom method, where you learn by doing. It starts with easy projects like blinking an LED, then moves to more advanced systems like automation and IoT. With hands-on activities and clear explanations, you will gain the skills needed to create your own smart solutions.

This eBook was created as part of the ANUGERAH REKABENTUK INSTRUKSIONAL FLEKSIBEL (ARIF) project to support active learning. Whether you are new to embedded systems or looking to improve your skills, we hope this book will guide you, inspire you, and help you enjoy the learning process.

Let’s start small, think big, and have fun creating!

TABLE OF CONTENTS

Starting up

Lesson 1 : Introduction to Embedded System & Arduino Installation

Think Big, Start Small, Have Fun: Your Journey into Embedded Systems Journey

Lesson 2 : Install Arduino IDE

TABLE OF CONTENTS

Lesson 3 : Arduino LED Blink

Lesson 4 : 7 Segment Display

TABLE OF CONTENTS

Lesson 5 : LCD Display

Exercises

List of Table

TABLE OF CONTENTS

List of Figure

WHY SHOULD YOU CARE?

Hands-on learning: You'll build real projects that solve real problems. Forget boring theory—this eBook is all about building, experimenting, and innovating

Career opportunities: These skills are in high demand in industries like robotics, IoT, and automation. Companies are looking for people who can design and build smart systems.

Fun and creativity: You'll get to experiment, innovate, and turn your ideas into reality. Imagine building your own smart home system, a robot, or even a drone. That's what embedded systems can do!

How to Use This eBook

1. **Read the chapters :** Each week, focus on one chapter and complete the activities.
2. **Watch the videos :** Links to YouTube tutorials are provided for extra help.
3. **Scan QR codes :** Use your smartphone to view 3D models and circuits in AR.
4. **Build projects:** Apply what you've learned by building your own projects.

Lesson 1 – Introduction to Embedded Systems & Arduino Installation

Imagine building your own smart home system, a robot, or even a drone. That's what embedded systems can do!

Why learn embedded systems?

Embedded systems are the backbone of modern technology. They're used in everything from smart home devices to medical equipment. By learning embedded systems, you'll gain the skills to create your own smart devices and solve real-world problems.

You'll learn how to control lights, sensors, and motors using Arduino a beginner-friendly tool.

PIC vs Arduino: Which one should you choose?

PIC Microcontroller : Great for advanced projects, but harder to program. If you love coding from scratch and want full control over hardware, PIC is for you.

Arduino: Perfect for beginners easy to use and quick to get results. Arduino has lots of pre-built libraries, so you can focus on building your project instead of writing complex code.

Feature	PIC Microcontroller	Arduino
Programming Complexity	High (More manual coding)	Low (Lots of pre-built libraries)
Hardware Control	Very flexible	Moderate
Best For	Advanced projects	Rapid prototyping & IoT

Table 1.1 : Comparison between PIC Microcontroller and Arduino in terms of programming complexity, hardware control, and best use cases.

Why Arduino Uno is More Popular

Simple to use: Easier for students to understand hardware and code structure.
Huge community: Tons of tutorials, examples, and support available.
Plenty of shields and modules: Compatible with sensors and displays used in common student projects.
Robust I/O pins: Great for basic automation, motor control, LEDs, and LCDs.
No need for Wi-Fi setup: Ideal for offline, hardware-focused projects.

When ESP (ESP8266/ESP32) is Better

Needed if the project requires Wi-Fi/Bluetooth.
Ideal for IoT-based or cloud-connected applications.

Summary

Arduino Uno : best for learning fundamentals, basic control systems, offline interfacing.
ESP32/8266 : best when Wi-Fi, Bluetooth, or more processing power is needed.

Think Big, Start Small, Have Fun: Your Journey into Embedded Systems Journey

What if your room could clean itself, or your plants could water themselves? With embedded systems, that's not science fiction it's a real possibility! In this topic, you'll explore how to bring your big ideas to life, starting with fun and simple projects like blinking LEDs.

As you build your skills, you'll unlock the ability to create powerful systems that automate everyday tasks. Whether you're dreaming of robots or smart gardens, it all starts here.

Dive in, get creative, and enjoy learning by doing!

Lesson 2 :Install Arduino IDE and make your first LED blink!

1. Download Arduino IDE: Go to https://www.arduino.cc/en/software and download the software.
2. Connect Your Arduino: Plug in your Arduino Uno using a USB cable.
3. Upload Your First Code: Open the Arduino IDE, go to **File** >
4. Examples > 01.Basics > Blink, and click **Upload**. If the LED on the board blinks, congratulations—you've just written your first embedded system program!

What You Will Learn

★ How to **download** the Arduino library files.
★ How to **instal** the Arduino library in Proteus 8.
★ How to **verify** that the library has been added successfully.

What You Need

Proteus 8 Software: This is the program where you will simulate your circuits.
Arduino Library Files: You will download these files from a link provided in the video.
https://www.youtube.com/watch?v=TKxAkE3837A

Figure 1.1: Video How to Add Arduino Library in Proteus 8

Step by step Instructions

Step 1: Open Proteus 8

Open the Proteus 8 software on your computer.
Go to **Schematic Capture** (this is where you design your circuits).

Step 2: Search for Arduino

In the Device List(where you find components), search for "Arduino".
At first, you will only see **Arduino headers** (small parts of the Arduino board), but not the full Arduino boards.

Step 3: Download the Arduino Library

Open your **web browser** (like Chrome or Firefox).
Go to the **link provided in the video description**. This link will take you to a Google Drive page where you can download the Arduino library files.
Download the **RAR file** (a compressed file) that contains the Arduino library.

Step 4: Extract the Library Files

Find the downloaded **RAR file** on your computer.
Use a program like **WinRAR** or **7-Zip** to **extract** (unzip) the files from the RAR file.
After extracting, you will see two files:
A file with the **.idx** extension (e.g., `modalLibrary.idx`).
A file with the **.lib** extension (e.g., `modalLibrary.lib`).

Step 5: Copy the Library Files to the Proteus Library Folder

Copy the two extracted files (`.idx` and `.lib`).
Now, you need to paste these files into the **Proteus library folder**. Here's how to find it:
Option 1: Go to `C:\Program Files (x86)\Labcenter Electronics\Proteus 8 Professional\LIBRARY`.
Option 2: If you don't see the folder in `Program Files (x86)`, go to `C:\ProgramData\Labcenter Electronics\Proteus 8 Professional\LIBRARY`. (Note: The `ProgramData` folder is usually hidden, so you may need to enable "Show hidden files" in your computer settings.)
Paste the copied files into the `LIBRARY` folder.

Step 6: Restart Proteus

Close the Proteus 8 software completely.

Open it again to make sure the new library files are loaded.

Step 7: Verify the Installation

Go back to **Schematic Capture** in Proteus.

Search for "Arduino" in the **Device List** again.

Now, you should see a full list of Arduino boards and components (like Arduino Uno, Arduino Mega, etc.).

Key Takeaways

This tutorial shows you how to **manually add the Arduino library** to Proteus 8.

By following these steps, you can use **Arduino boards and components** in your Proteus simulations.

This is useful for testing your Arduino projects in a **virtual environment** before building them with real hardware.

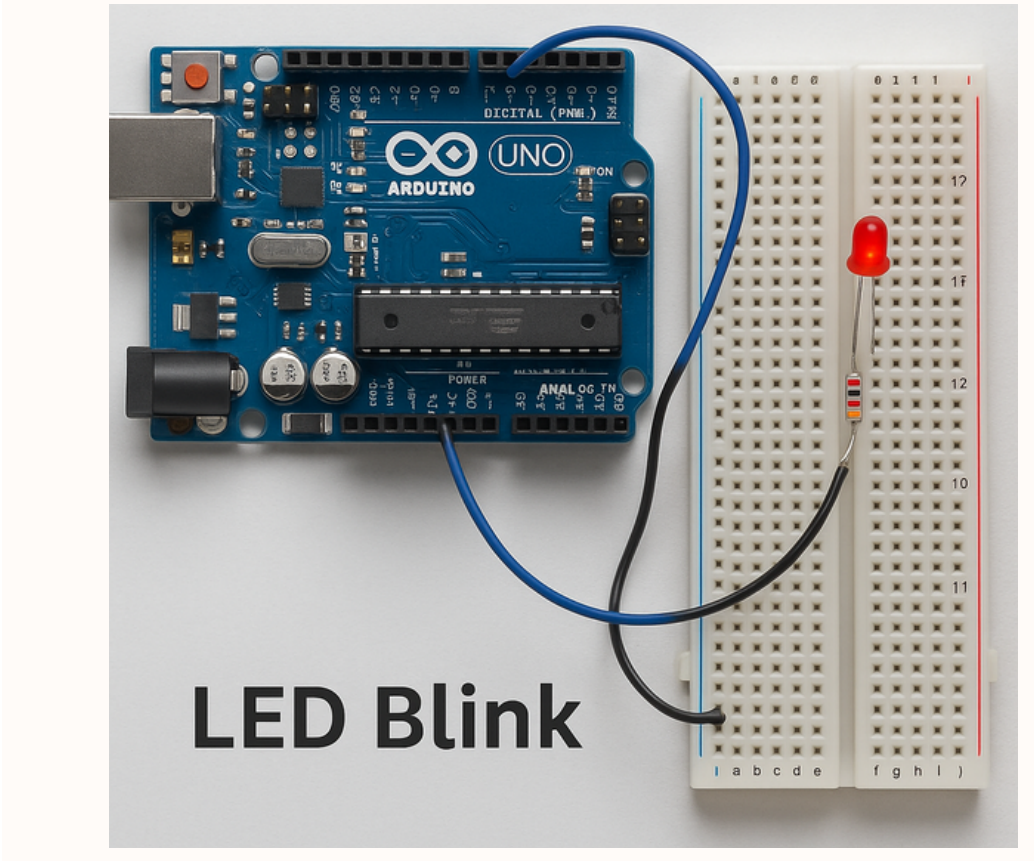
Why This is Useful

Simulate Arduino Projects: You can test your Arduino code and circuits in Proteus without needing real hardware.

Save Time and Money: Simulation helps you find and fix mistakes before building the actual project.

Learn Electronics: Proteus is a great tool for learning how electronic components work together.

Try adding other libraries to Proteus (like sensors or displays).
Start simulating your own Arduino projects in Proteus.



Lesson 3 : Arduino LED Blink

What you'll learn:

★ How to control an LED using Arduino.

★ Simulate your project using Tinkercad (no hardware needed!).

★ Write your first Arduino code and see it in action.

Step-by-Step Guide:

1. Connect the LED: Connect an LED to your Arduino using a resistor to protect it from too much current.
2. Write the Code: Use the `digitalWrite()` function to turn the LED on and off.
3. Upload the Code: Upload the code to your Arduino and watch the LED blink!

Figure : 3.1
Augmented Reality (AR) Video: LED Blink with Arduino

Try modifying the blink rate and see how it changes. Can you make the LED blink faster or slower?

The "LED Blink" application is one of the most basic and fundamental projects when starting with Arduino. It helps users get familiar with the basic setup and programming of the Arduino platform. Here are some variations and applications of the "LED Blink" project:

1. Basic LED Blink (Standard Example)

Goal: Blink an LED on the Arduino board on and off with a 1-second interval.

Code Example:

cpp
CopyEdit

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);  // Set the built-in LED pin as output
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // Turn the LED on
  delay(1000);                      // Wait for 1 second
  digitalWrite(LED_BUILTIN, LOW);   // Turn the LED off
  delay(1000);                      // Wait for 1 second
}
```

LED Blink with Different Intervals

- Goal: Blink an LED with different time intervals, e.g., blinking faster or slower.
- Code Example:

cpp
CopyEdit

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // Turn the LED on
  delay(500);                       // 500 ms delay (half a second)
  digitalWrite(LED_BUILTIN, LOW);   // Turn the LED off
  delay(2000);                      // 2-second delay
}
```

Figure 3.2 : How to Blink an LED with Arduino
https://youtu.be/FKekzzj5844?si=c7D-bfrOBXfV-CqR

LED Blink with Button Control

Objective: Use a push button to control the LED's blinking behavior, such as toggling it on and off or adjusting the blink speed.

Code Example:
cpp
CopyEdit

```
int buttonPin = 2;  // Button input pin
int ledPin = LED_BUILTIN;  // LED output pin
int buttonState = 0;  // Variable to store button state

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  buttonState = digitalRead(buttonPin);

  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);  // Turn LED on when button is pressed
  } else {
    digitalWrite(ledPin, LOW);   // Turn LED off when button is not pressed
  }
}
```

Figure 3.3: Blinking LED Using A Button
https://www.youtube.com/watch?v=7_q2x-N5Rl60

```
int buttonState = 0;  // Variable to store button state

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  buttonState = digitalRead(buttonPin);

  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);  // Turn LED on when button is pressed
  } else {
    digitalWrite(ledPin, LOW);   // Turn LED off when button is not pressed
  }
}
```

LED Blink with PWM (Pulse Width Modulation)

Objective: Control the brightness of the LED by varying the PWM value.

Code Example:
cpp
CopyEdit

```
int ledPin = 9;  // LED connected to pin 9 (PWM-capable pin)
int brightness = 0;  // Initial brightness (0 = off, 255 = fully bright)
int fadeAmount = 5;  // Amount by which the brightness changes

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  analogWrite(ledPin, brightness);  // Set LED brightness using PWM
 brightness = brightness + fadeAmount;  // Change brightness for fading effect

  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;  // Reverse the direction of fading when limits are reached
  }

  delay(30);  // Delay to slow down the fading
}
```

Figure 3.4: How Does PWM Work?
https://www.youtube.com/shorts/aeE0u1J-1pg

LED Blink with Random Intervals

Objective: Make an LED blink at unpredictable times by using random time intervals between each blink.

Code Example:

cpp

CopyEdit

```
int ledPin = LED_BUILTIN;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);  // Turn LED on
  delay(random(500, 2000));    // Random delay between 500 ms to 2 seconds
  digitalWrite(ledPin, LOW);   // Turn LED off
  delay(random(500, 2000));    // Random delay between 500 ms to 2 seconds
}
```

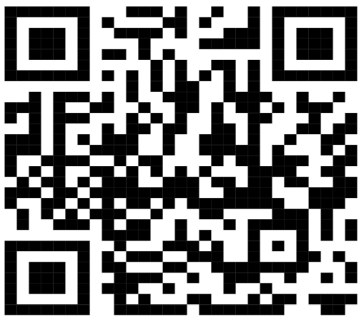


Figure 3.5:Three LED Blink at Random Intervals with Arduino
https://www.youtube.com/watch?v=YvU1NApos1Y

LED Blink with Serial Monitor Feedback

Objective: Display a message on the serial monitor every time the LED blinks to provide real-time feedback.

Code Example:
cpp
CopyEdit

```
int ledPin = LED_BUILTIN;

void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);  // Start serial communication
}

void loop() {
  digitalWrite(ledPin, HIGH);  // Turn LED on
  Serial.println("LED ON");
  delay(1000);                 // Wait for 1 second

  digitalWrite(ledPin, LOW);   // Turn LED off
  Serial.println("LED OFF");
  delay(1000);                 // Wait for 1 second
}
```

Figure 3.6:TURN ON LED USING SERIAL MONITOR
https://youtu.be/gbA0jmrpod4?si=qxVYBcqtG7PNmnXC

LED Blink with Variable Blink Speed via Potentiometer

Objective: Control the blink speed of the LED using a potentiometer.

Code Example:
cpp
CopyEdit

```
int potPin = A0;     // Potentiometer input pin
int ledPin = LED_BUILTIN;  // LED output pin
int potValue = 0;    // Variable to store potentiometer value
int delayTime = 0;   // Variable to store delay time

void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  potValue = analogRead(potPin);       // Read potentiometer value
  delayTime = map(potValue, 0, 1023, 100, 1000);  // Map potentiometer value to delay
range

  digitalWrite(ledPin, HIGH);  // Turn LED on
  delay(delayTime);          // Wait for mapped delay time
  digitalWrite(ledPin, LOW);   // Turn LED off
  delay(delayTime);          // Wait for mapped delay time
}
```

Figure 3.7:Potentiometer controls LED Fade in and out and LED blink speed https://www.youtube.com/shorts/m1WptIf2SI0

LED Blink with External Components (e.g., Transistor or MOSFET)

Objective: Control a higher-power LED (or multiple LEDs) using a transistor or MOSFET.
Application: Use a transistor as a switch to control an external LED array, which requires more current than the Arduino pin can supply.
Each of these variations builds on the basic concept of blinking an LED, but introduces more complexity and interactivity with the hardware, allowing for a range of applications, from simple control to more advanced systems that respond to external input or feedback.

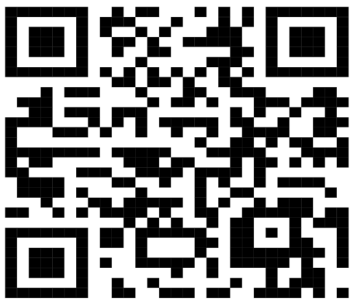


Figure 3.8:Simple led flasher using bc547
https://youtube.com/shorts/F3jVPovUI80?si=v8tDKg-7SnPW6OPF

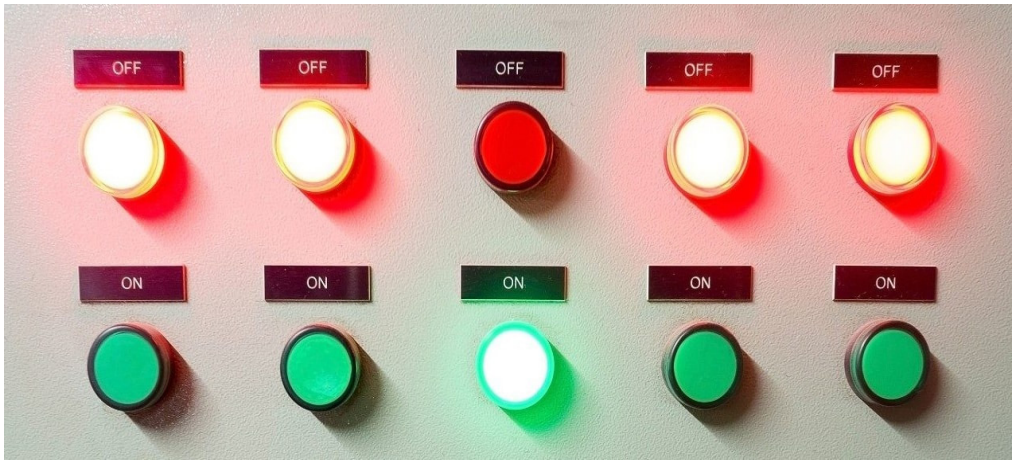
Figure 3.9:Transistor controlled by Arduino (as amplifier)
https://www.youtube.com/shorts/uywv1ZedFFU

Applications of "LED Blink with Different Intervals"

Varying the blink intervals of an LED opens up practical and creative applications across industries. Here are some key use cases:

Status Indicators for Devices

Devices often use distinct LED blink patterns to indicate their current states: for example, a network router may show slow blinks (e.g., 2 seconds on/off) to indicate standby, fast blinks (0.5 seconds on/off) during active data transmission, and irregular double-blinks to signal errors or connection issues. Similarly, smart home devices might blink slowly when searching for a Wi-Fi network, blink rapidly when actively syncing or updating, and flash in irregular bursts to warn of low battery or hardware faults.



Traffic/Pedestrian Signals

Traffic and pedestrian signals rely on timed blinking patterns to ensure safety and regulate movement. For instance, pedestrian crossing lights often blink slowly at first, then speed up as the “walk” phase nears its end, prompting people to finish crossing. Railway signals may use alternating or flashing lights to indicate an approaching train. Examples include countdown timers at crosswalks that combine numeric displays with blinking lights, and construction zone warning lights that flash in rhythmic or alternating patterns to alert drivers of lane changes or hazards ahead.

Figure 3.10 : Traffic/Pedestrian Signals

Interactive Art/Entertainment

Dynamic visual effects are created using programmable blink patterns that respond to various inputs or are synchronized with external stimuli. For example, LEDs can blink in time with music beats, using varying intervals to match the rhythm and tempo, creating an immersive audio-visual experience. Additionally, sensors like motion or light detectors can trigger real-time changes in the blink patterns, making the display interactive. Examples include light installations that change colors and patterns as people move nearby, and stage lighting systems that blink, pulse, or fade in coordination with concerts or theater performances to enhance the mood and storytelling.

Figure 3.11 : Interactive Art or Entertainment

Emergency/Alert Systems

Urgency signaling is achieved by using varying blink speeds to convey the severity of a situation. For instance, slow blinks are often used for non-critical warnings, such as a low fuel alert in a vehicle, allowing users to take action without immediate panic. In contrast, rapid blinking is used for critical alerts, like fire alarms or system failures, demanding immediate attention. Common examples include car dashboard indicators such as a blinking check engine or tire pressure warning and industrial equipment fault lights, which may blink faster as the issue becomes more severe or remains unresolved.

Figure 3.12 :Emergency or Alert Systems

Wearable Tech/Health Monitoring

Wearable technology and health monitoring devices use visual blink patterns to provide real-time feedback on health metrics. For example, LEDs may simulate a heartbeat with irregular or pulsing blinks that reflect the user's actual pulse rate. Similarly, changes in blink intervals can indicate rising or falling values, such as increasing heart rate or fluctuating glucose levels. Common applications include fitness trackers that blink to remind users to move or to signal goal completion, and medical devices that use flashing alerts to warn of abnormal readings like high blood pressure or irregular heart rhythms.

Figure 3.13 :Wearable Tech/Health Monitoring

Educational Tools

Educational tools often use blinking LEDs to teach fundamental concepts in timing, coding, and electronics. For instance, students can program loops and delays to control LED blink rates, helping them understand how timing works in code. They can also experiment with analog vs. digital signals, learning how pulse width modulation or binary states affect behavior. Examples include classroom kits for learning Arduino basics, where blinking an LED is often the first hands-on exercise, and DIY science projects like simulating Morse code, which reinforce both programming logic and communication principles.

Figure 3.14 :Educational Tools

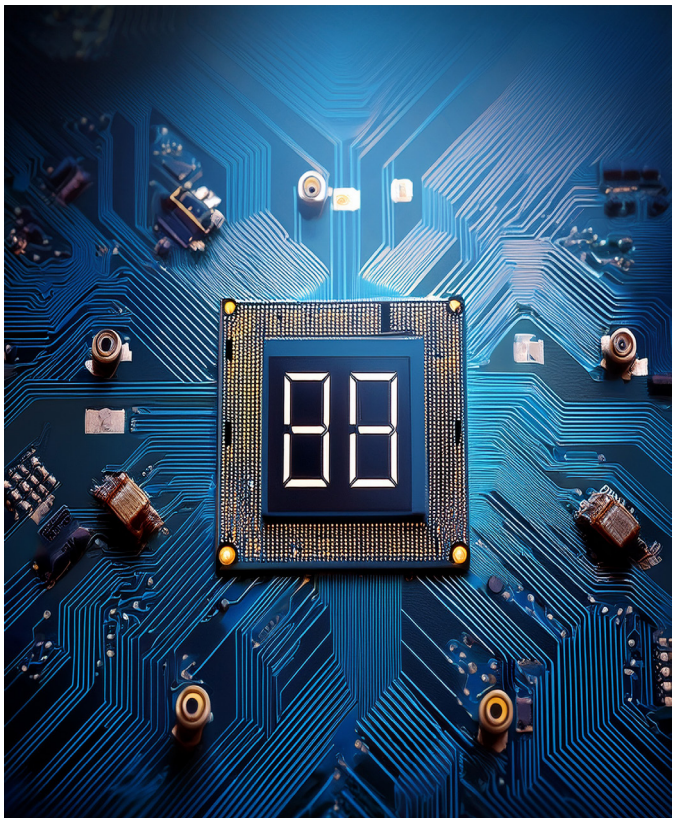
Automotive Lighting

Automotive lighting uses adaptive blink patterns to improve visibility and enhance safety on the road. For example, turn signals may blink faster during quick lane changes to draw more attention, while brake lights can flash rapidly during sudden or emergency stops to warn drivers behind more effectively. Modern vehicles often feature dynamic turn signals that create a flowing light effect to indicate direction more clearly. Similarly, bicycle taillights come with multiple blink modes steady, pulsing, or strobe to increase visibility in various conditions and alert nearby traffic.

Figure 3.15 :Automotive Lighting

Security Systems

Security systems often use unpredictable blink patterns to deter intruders and enhance protection. One method is to randomize LED blink intervals to simulate occupancy in an empty house, making it appear as though someone is home. In more active systems, multiple LEDs may blink in sync when an alarm is triggered, drawing attention to a potential threat. Examples include fake security cameras that feature blinking LEDs to give the illusion of active surveillance, and smart security lights that flash or change patterns in response to motion, alerting homeowners and scaring off potential intruders.



Lesson 3 : 7-Segment Display

What you'll learn:

How to use 7-segment and LCD displays.
Build a project using Proteus (a simulation tool).
Troubleshoot common issues and improve your coding skills

A **7-segment display** is a simple electronic component used to **display numbers** and some **letters**. It is commonly found in devices like digital clocks, calculators, and electronic meters. The name "7-segment" comes from the fact that it has **7 individual light segments** (labeled A to G) that can be turned on or off to form different numbers or characters.

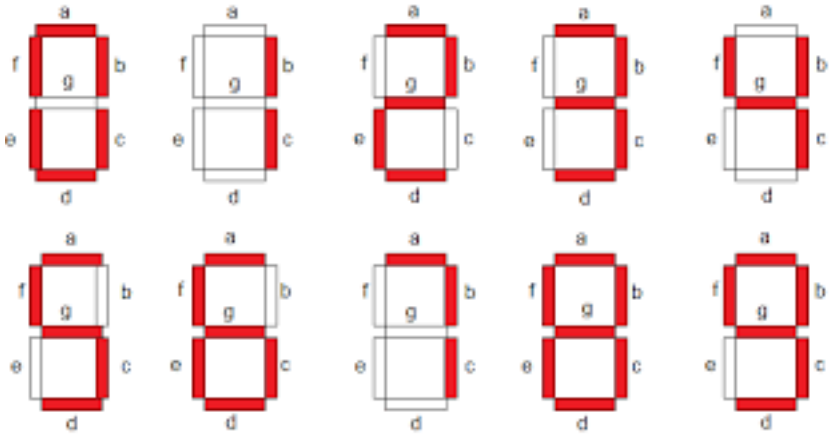


Figure 3.16 : 7 Segment Display

How Does a 7-Segment Display Work?

A **7-segment display** consists of **seven individual LED segments** arranged in a figure-eight pattern. Each segment is labeled (A through G) and can be **controlled separately** to form various numbers or letters.

For example:

* To display the number **"1"**, only segments **B** and **C** are illuminated.
* To display the number "8", **all seven segments (A to G)** are turned on simultaneously.

By selectively turning segments on or off, the display can represent digits from 0 to 9 and some alphabetic characters.

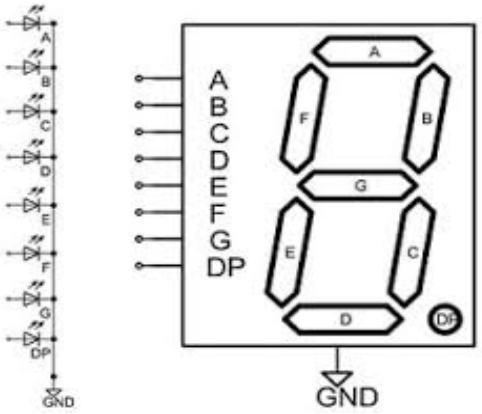


Figure 3.17 : 7 Segment pin

Figure 3.18 : Augmented Reality (AR) Video, 7 Segment with Arduino

Types of 7-Segment Display

There are two main types of 7-segment displays:

1. **Common Cathode (CC)**:
   All the **negative terminals** (cathodes) of the LEDs are connected together.
   To turn on a segment, you need to supply **positive voltage** to that segment.

2. **Common Anode (CA):**
   All the **positive terminals** (anodes) of the LEDs are connected together.
   To turn on a segment, you need to **ground** (connect to negative) that segment.

Pin Configuration

A 7-segment display usually has 10 pins:
7 pins for the segments (A to G).
1 pin for the decimal point (optional, used for showing decimals like “5.2”).
2 pins for the common cathode or common anode (depending on the type of display).How to Use a 7-Segment Display

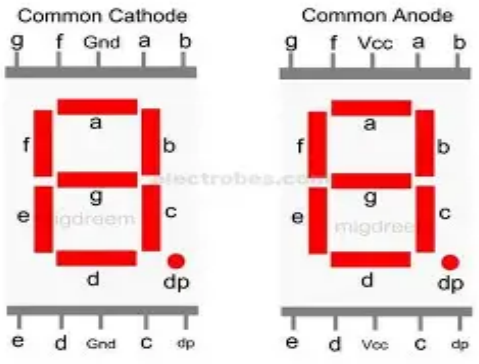


Figure : 3.19 Segment pin configurations for the common cathode or common anode

How to Use a 7-Segment Display

To use a 7-segment display, you need to:
1. **Connect the display** to a microcontroller (like Arduino) or a driver circuit.
2. **Control the segments**by sending signals to the pins.
3. **Write code** to turn on/off the segments to display the desired number or character.

Here’s how you can display the number "3" on a **common cathode** 7-segment display:

- **Turn on segments: A, B, C, D, G.**
- Turn off segments: **E, F**.

Example: Displaying Numbers

For a **common anode** display, you would do the opposite:

- Turn off segments: **A, B, C, D, G**.
- Turn on segments: **E, F**.

Applications of 7-Segment Displays

- **Digital Clocks**: Used to show hours, minutes, and seconds.
- **Calculators**: Used to display numbers and results.
- **Electronic Meters**: Used in devices like voltmeters, ammeters, and thermometers.
- **Scoreboards**: Used in sports to show scores or timers.

Limitations of 7-Segment Displays

- **Limited characters**: Can only display numbers and a few letters (like A, B, C, D, E, F).
- Not suitable for complex graphics: Cannot display images or detailed text.

Example Code for Arduino

Here's a simple Arduino code to display the number **"5"** on a **common cathode** 7-segment display:

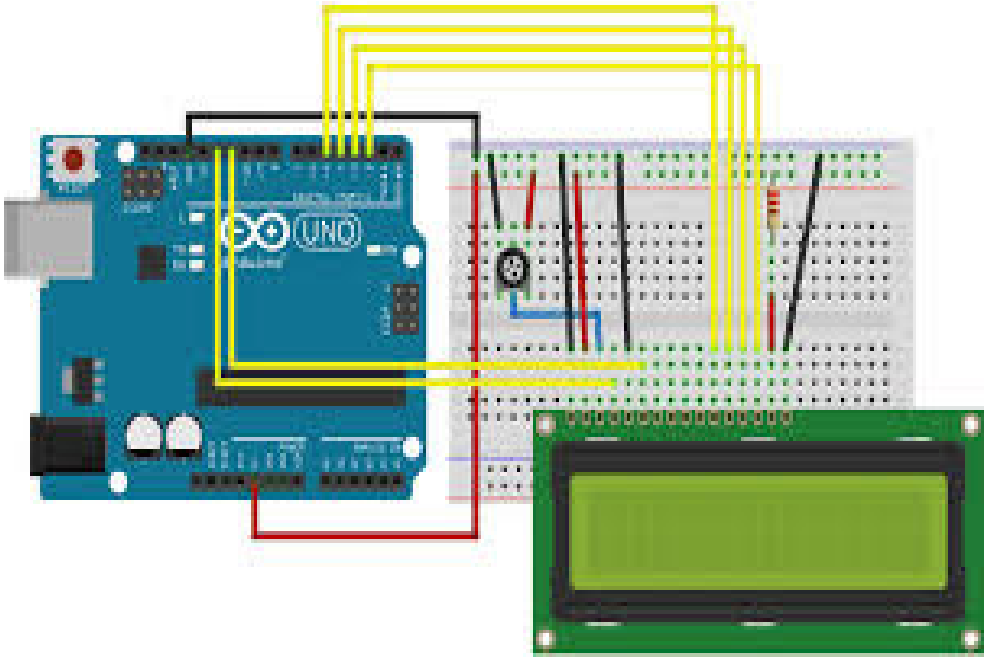
```cpp
void setup() {
  // Set pins 2 to 8 as OUTPUT
  for (int i = 2; i <= 8; i++) {
    pinMode(i, OUTPUT);
  }
}

void loop() {
  // Display "5"
  digitalWrite(2, HIGH);  // Turn on segment A
  digitalWrite(3, HIGH);  // Turn on segment B
  digitalWrite(4, LOW);   // Turn off segment C
  digitalWrite(5, HIGH);  // Turn on segment D
 digitalWrite(6, LOW);   // Turn off segment E
  digitalWrite(7, HIGH);  // Turn on segment F
  digitalWrite(8, HIGH);  // Turn on segment G
}
```

Summary

A **7-segment display** is a simple and versatile component used to show numbers and some letters. It is widely used in electronics because it is **easy to control**, **low cost**, and **energy efficient**. Whether you're building a digital clock, a calculator, or a scoreboard, a 7-segment display is a great choice for displaying numeric information.

Can you create a countdown timer using the 7-segment display?



LESSON 4 :LCD DISPLAY

How Does an LCD Display Work?

- An LCD consists of **liquid crystals** sandwiched between two layers of glass or plastic.
- These liquid crystals can change their alignment when an **electric current** is applied, which controls how much light passes through.
- The display is **backlit** (usually with LEDs), so when light passes through the liquid crystals, it creates visible text or images.
- Each character or pixel on the display is made up of smaller segments (like dots or bars) that can be turned on or off.

Character LCD:

- Used to display **text and simple symbols**.
- Common sizes: 16x2 (16 characters per line, 2 lines) or 20x4 (20 characters per line, 4 lines).
- Example: Used in calculators, small displays for appliances, or Arduino projects.

Graphical LCD:

- Used to display **images, graphics, and complex text.**
- Example: Used in smartphones, computer monitors, and advanced embedded systems.

Pin Configuration of a Character LCD

1. A typical **16x2 character LCD** has **16 pins**, but not all are always used. Here are the most important pins:
2. **VSS**: Ground (0V).
3. **VDD/VEE**: Power supply (usually 5V).
4. **V0**: Contrast control (adjusts display brightness).
5. **RS**: Register Select (switches between command and data modes).
6. **RW**: Read/Write (used to read or write data).
7. **E**: Enable (starts reading/writing data).
8. **D0-D7**: Data pins (send data or commands to the display).
9. **A**: Backlight anode (+).
10. **K**: Backlight cathode (-).

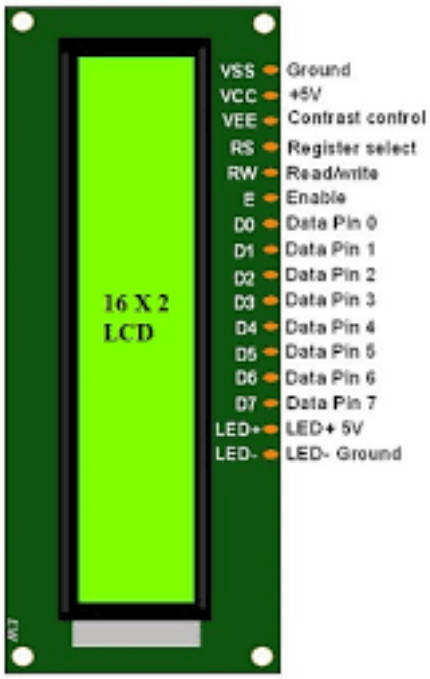


Figure 3.20 : Pin configuration of LCD

How to Use an LCD Display

To use an LCD display, you first need to connect it to a microcontroller like an Arduino or a driver circuit using the appropriate pins for power, ground, data, and control. Once connected, you send initialization commands to set up the display, such as clearing the screen or setting the cursor position. After that, you can send data like text or numbers which the LCD will show on its screen.

Example: Connecting a 16x2 LCD to Arduino

Here’s how you can connect a **16x2 LCD** to an Arduino and display "Embedded Robotic 2":

Wiring
- **VSS** → Arduino GND
- **VDD** → Arduino 5V
- **V0** → Potentiometer (to adjust contrast)
- **RS** → Arduino Pin 12
- **RW** → Arduino GND
- **E** → Arduino Pin 11
- **D4-D7** → Arduino Pins 5, 4, 3, 2
- **A** → Arduino 5V (for backlight)
- **K** → Arduino GND (for backlight)

Figure 3.21 :Augmented Reality (AR) Video LCD with Arduino

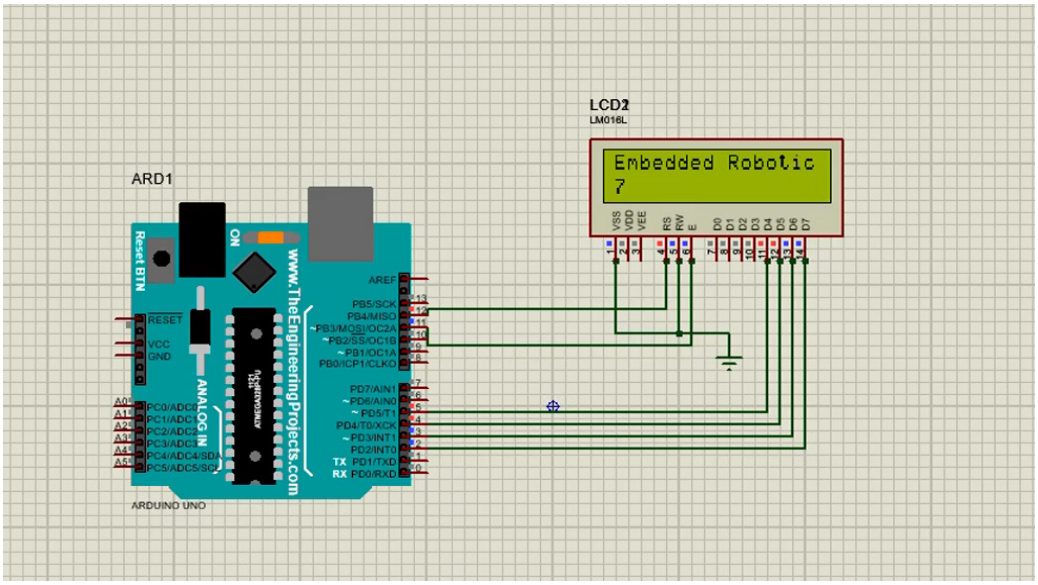


Figure 3.22 : Wiring Circuit: LCD with Arduino

Arduino Code

```cpp
#include <LiquidCrystal.h>  // Include the LCD library

// Initialize the LCD with the correct pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // Set up the LCD's number of columns and rows
  lcd.begin(16, 2);  // 16 columns, 2 rows
  lcd.print(" Embedded Robotic 2");  // Display text
}

void loop() {
  // Move the cursor to the second line
  lcd.setCursor(0, 1);
  lcd.print("LCD Tutorial");  // Display more text
}
```

Applications of LCD Displays

LCD displays are widely used across various fields due to their versatility and clarity. In **consumer electronics**, they are commonly found in TVs, smartphones, and computer monitors. In **industrial equipment**, LCDs serve as interfaces in control panels and machinery displays. **Medical devices** use them for patient monitoring and diagnostic screens, providing clear and accurate readouts. Additionally, in **embedded systems**, LCDs are frequently integrated with platforms like Arduino and Raspberry Pi to display data in real-time, making them essential in both educational and practical applications.

Figure 3.23 : LCD in consumer electronics

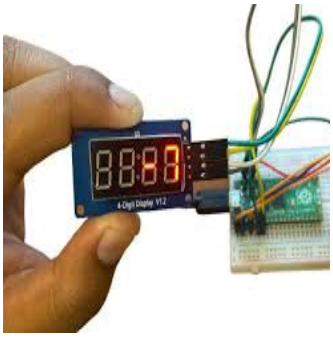


Figure 3.24 : interfaces in control panels and machinery displays

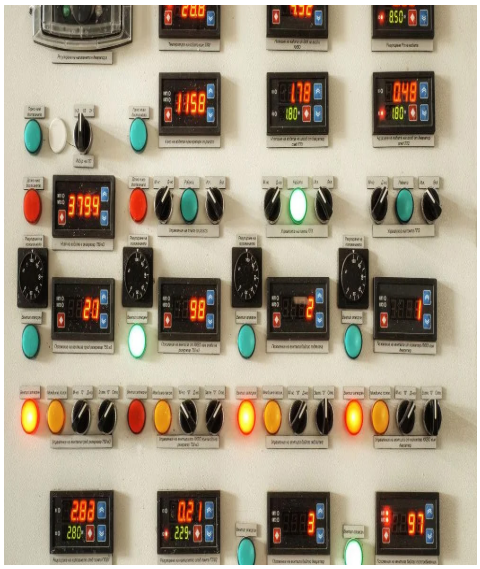


Figure 3.25: Interfaces in control machinery displays

Advantages of LCD Displays

- **Low power consumption**: Uses less energy compared to other display types.
- **Lightweight and thin**: Easy to integrate into small devices.
- **Good visibility**: Works well in most lighting conditions.
- **Affordable**: Cost-effective for many applications.

Limitations of LCD Displays

LCD displays have some limitations, including **restricted viewing angles**, which can cause the screen to appear dim or distorted when viewed from the side. They also tend to have a **slower response time**, making them less suitable for fast-moving visuals like those in gaming. Additionally, LCDs **require a backlight** to be visible, which means they can be difficult to read in low-light conditions if the backlight is insufficient or turned off.

Summary

An **LCD display** is a versatile and widely used component for showing text, numbers, or graphics in electronic devices. It is **easy to use**, **energy-efficient**, and **affordable**, making it a popular choice for both simple and complex projects. Whether you're building a digital clock, a weather station, or a small information display, an LCD is a great option.

Review Questions

Section 1: Introduction to Embedded Systems

1. What is an embedded system? Give two real-life examples.
2. How are embedded systems different from general-purpose computers?
3. Why is Arduino popular in embedded systems learning?

Section 2: Getting Started with Arduino

4. What is the function of a microcontroller in an embedded system?
5. Write a simple Arduino code to blink an LED.
6. Name three basic components commonly used with Arduino.

Section 3: Sensors and Inputs

7. What is a sensor? Name two types and their functions.
8. How does a temperature sensor work with Arduino?
9. What is the purpose of analog and digital pins?

Section 4: Automation Projects

10. Describe a project where Arduino can be used to automate a task at home.
11. How would you use a moisture sensor to water plants automatically?
12. What components would you need to build a room-cleaning robot?

Section 5: Augmented Reality Integration

13. What is augmented reality (AR), and how can it help you learn electronics?
14. List one benefit of using AR in embedded systems education.
15. How can AR be used to visualize circuit components or wiring?

Bonus: Think & Reflect

16. If you could invent a device using Arduino, what would it be? Describe how it works.
17. Why is it important to “start small” in embedded systems projects?
18. How did the flipped classroom model help you understand the topic better?

Review Questions and Answers

Section 1: Introduction to Embedded Systems

1. What is an embedded system? Give two real-life examples.

An embedded system is a computer system with a dedicated function within a larger system.

Examples: Microwave oven, Smartwatch.

2. How are embedded systems different from general-purpose computers?

Embedded systems are designed for specific tasks, while general-purpose computers can run multiple programs and functions.

3. Why is Arduino popular in embedded systems learning?**

Arduino is beginner-friendly, low-cost, open-source, and has a large online community for support and tutorials.

Section 2: Getting Started with Arduino

4. What is the function of a microcontroller in an embedded system?**

A microcontroller processes input from sensors and controls output devices based on programmed instructions.

5. Write a simple Arduino code to blink an LED.**

```cpp
void setup() {
  pinMode(13, OUTPUT);
}
void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

6. Name three basic components commonly used with Arduino.

LED, push button, temperature sensor

Section 3: Sensors and Inputs

7. What is a sensor? Name two types and their functions.

A sensor detects changes in the environment and sends data to the system.

Examples:

Temperature sensor: detects heat

Motion sensor: detects movement

8. How does a temperature sensor work with Arduino?

It measures temperature and sends analog signals to Arduino, which converts them into readable data.

9. What is the purpose of analog and digital pins?

Analog pins read continuous voltage values (e.g., sensors), while digital pins read or output only HIGH or LOW (ON/OFF).

Section 4: Automation Projects

10. Describe a project where Arduino can be used to automate a task at home.

A smart light system that turns on automatically when someone enters a room using a motion sensor and Arduino.

11. How would you use a moisture sensor to water plants automatically?

Connect the sensor to Arduino to detect soil moisture. If the level is low, the Arduino activates a water pump.

12. What components would you need to build a room-cleaning robot?

Microcontroller, motors, wheels, sensors (ultrasonic/IR), battery, and chassis.

Section 5: Augmented Reality Integration

13. What is augmented reality (AR), and how can it help you learn electronics?

AR overlays digital content onto the real world, helping learners visualize circuits and components in 3D.

14. List one benefit of using AR in embedded systems education.

AR makes complex hardware easier to understand through interactive, 3D visualizations.

15. How can AR be used to visualize circuit components or wiring?

AR can project 3D models of circuits or components on-screen, showing how they connect without needing physical parts.

Bonus: Think & Reflect

16. If you could invent a device using Arduino, what would it be? Describe how it works.

Example answer: A smart pet feeder that dispenses food at set times using a servo motor and real-time clock module.

17. Why is it important to "start small" in embedded systems projects?

Starting with simple projects builds confidence and understanding before moving to more complex systems.

18. How did the flipped classroom model help you understand the topic better?

Example answer: It allowed me to learn theory at my own pace and focus on hands-on practice during class time.

REFERENCES

Arduino. (n.d.). Arduino IDE Downloads. Arduino.cc. https://www.arduino.cc/en/software

Labcenter Electronics. (2023). Proteus Design Suite – Simulation & PCB Design Software. https://www.labcenter.com/

Espressif Systems. (2022). ESP32 Series Datasheet and Resources. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/

Monk, S. (2016). Programming Arduino: Getting Started with Sketches (2nd ed.). McGraw-Hill Education.

McWhorter, P. (2020). Arduino Tutorial #1 - Getting Started and LED Blink [Video]. YouTube. https://www.youtube.com/watch?v=fJWR7dBuc18

Simon, J. (2011). Practical Electronics for Inventors (3rd ed.). McGraw-Hill Education.

Banzi, M., & Shiloh, M. (2014). Getting Started with Arduino (3rd ed.). Maker Media, Inc.

YouTube. (2022). The Arduino Starter Kit Tutorials. https://www.youtube.com/playlist?list=PLT6ZzNi5fYd8VZPFQh3a9tsC34G4wVbVW

Tinkercad. (2011). Circuits – Learn & Simulate Arduino Projects Online. https://www.tinkercad.com/learn/circuits

MathWorks. (2013.). Simulink Support Package for Arduino Hardware. https://www.mathworks.com/hardware-support/arduino-simulink.html

APPENDICES

1. Padlet Embedded Systems

2. Survey
https://forms.gle/zjp4ujr1aVaUXEoi6