Embedded System – Arduino Framework

MD Shariful

Introduction To Embedded System

A Basic Overview of microcontroller, Sensor, Display, communication protocol and M5Stack Core 2

Overview

What is an Embedded System?

Basic components of an Embedded System

Simplified overview of microcontroller, sensor and display

GPIO and communication protocols (UART, I2C and SPI)

Introduction to M5Stack Core 2

What is an Embedded System?

A specialized computing system designed to perform a specific task

Combines hardware (e.g., microcontroller, sensors) and software (e.g., firmware)

Common in everyday devices: smart watch, smart TV, microwave oven, etc.

Car control systems, medical devices, robotics, etc. are some examples

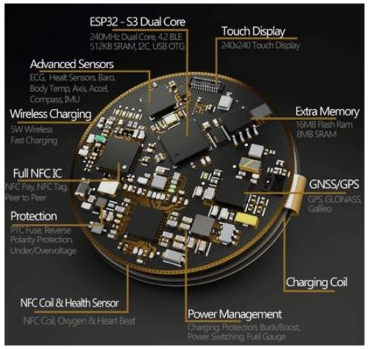

ESP32 - S3 Dual Core
240MHz Dual Core, 4.2 BLE
512KB SRAM, I2C, USB OTG
Touch Display
240x240 Touch Display
Advanced Sensors
ECG, Healt Sensors, Baro, Body Temp, Axis, Accel, Compass, IMU
Wireless Charging
5W Wireless Fast Charging
Extra Memory
16MB Flash Ram
8MB SRAM
Full NFC IC
NFC Pay, NFC Tag, Peer to Peer
GNSS/GPS
GPS, GLONASS, Galileo
Protection
PTC Fuse, Reverse Polarity Protection, Under/Overvoltage
Charging Coil
NFC Coil & Health Sensor
NFC Coil, Oxygen & Heart Beat
Power Management
Charging, Protection, Buck/Boost, Power Switching, Fuel Gauge

Fig: Smart Watch – An embedded System

Microcontroller: The Brain of Embedded Systems

A microcontroller (MCU) is a small computer on a single chip

It has a CPU, memory, and programmable input/output peripherals

Common microcontrollers: ATmega328p, STM32, ESP32, PIC

Typically used in embedded systems for control and data processing tasks

Fig: A STM32F103 microcontroller

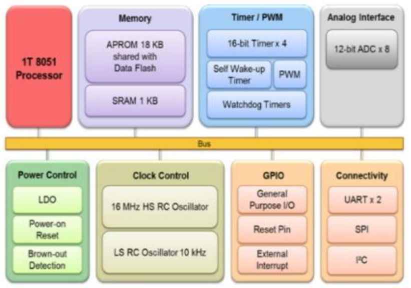


Fig: Block diagram of an 8051 microcontroller

Sensors in Embedded Systems

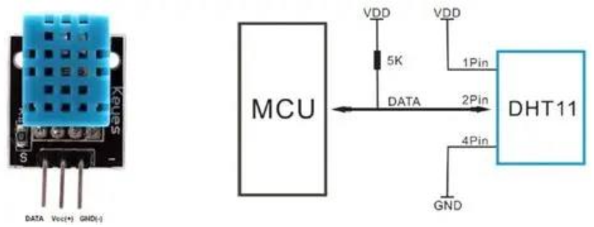
Sensors detect changes in physical or environmental conditions

Sensors convert real-world data into electrical signals

Types of sensors: Temperature, humidity, pressure, motion, light, etc.

Example: A temperature sensor (e.g., DHT11) converts temperature and humidity into digital data for a microcontroller

Fig: A DHT11 sensor is connected with a microcontroller



Display Devices in Embedded Systems

Displays are used to show information or feedback in embedded systems

Types of displays: LCD, OLED, 7-segment, TFT, etc.

Often used in devices like calculators, clocks, or digital thermometers

Displays communicate with microcontrollers via protocols like SPI or I2C

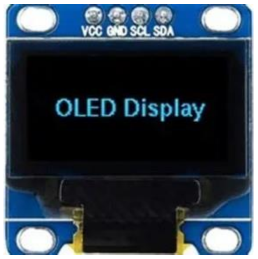


Fig: An OLED display

General Purpose Input/Output (GPIO)

GPIO pins allow microcontrollers to interface with the external world

Can be configured as input (e.g., reading sensors) or output (e.g., controlling LEDs)

GPIO pins are the fundamental way to interact with devices in embedded systems

Example: Turning an LED on/off using GPIO pins

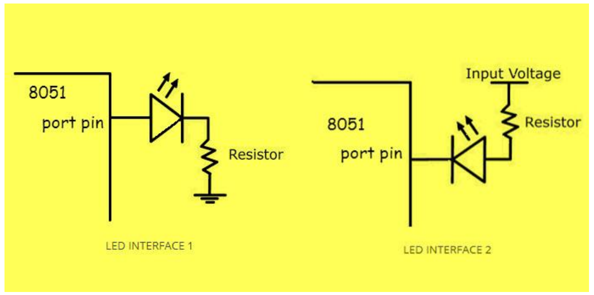


Fig: A LED is connected to a microcontroller GPIO

Universal Asynchronous Receiver-Transmitter (UART) Protocol

Simple, asynchronous and full-duplex

Generally, uses TX (Transmit) and RX (Receive) and no clock is required

GPS, Bluetooth, Wi-Fi modules, etc. use UART to transmit data

Commonly used to communicate with a PC. Also used for debugging

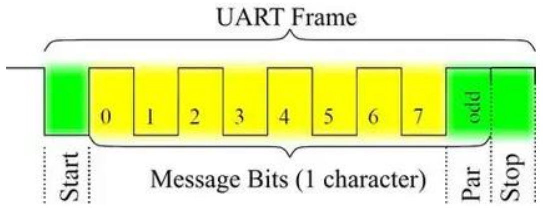


Fig: A UART frame

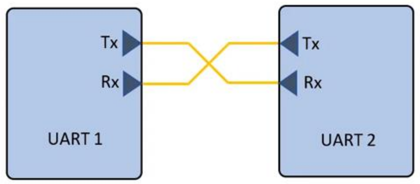


Fig: UART communication between two devices

Inter-Integrated Circuit (I2C) Protocol

A synchronous communication protocol

Uses only two wires: SDA (Serial Data) and SCL (Serial Clock).

Multiple devices with unique addresses can be connected up to a limit.

Used in interfacing sensors, real-time clocks, displays, EEPROM etc.

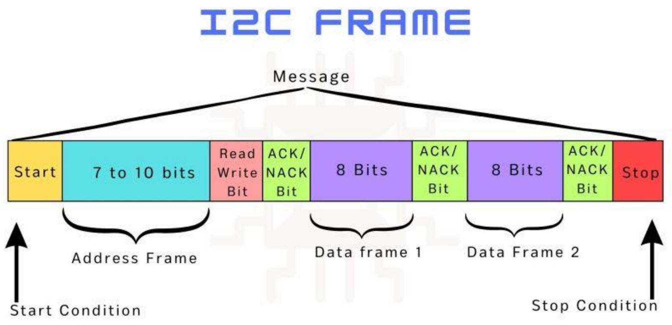


Fig: An I2C frame

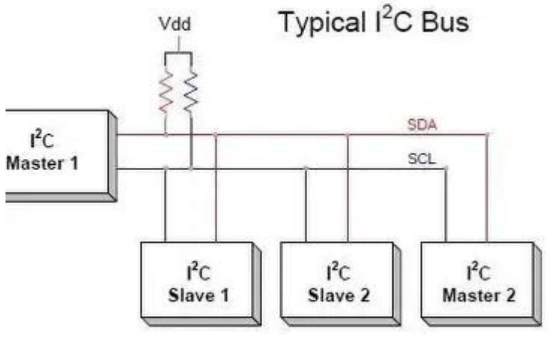


Fig: A I2C bus with multiple devices

Serial Peripheral Interface (SPI) Protocol

A synchronous, full-duplex high-speed communication protocol

Uses four wires: MOSI (Master Out Slave In), MISO (Master In Slave Out), SCK (Serial Clock), and CS (Chip Select).

Data flows simultaneously in both directions (MOSI & MISO).

Common Uses: Flash memory, SD cards, sensors, displays, and other high-speed devices.

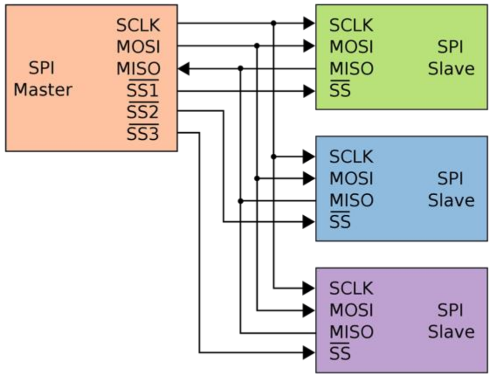


Fig: Devices are connected through SPI

M5Stack: A Powerful Embedded Development Platform

M5Stack is a modular and stackable embedded development kit.

Based on the ESP32 microcontroller, it provides Wi-Fi and Bluetooth connectivity.

Comes with various built-in sensors, displays, and modules.

Ideal for rapid prototyping and IoT projects.

Supports programming in Arduino IDE, MicroPython, and more.

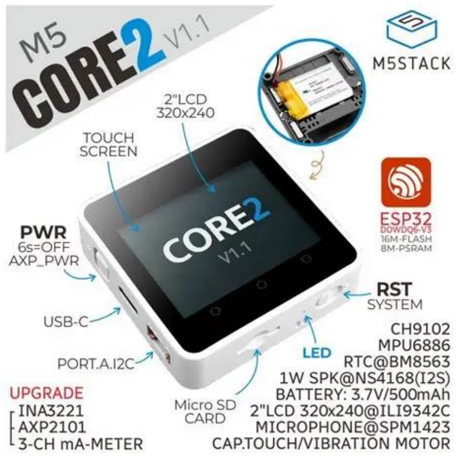


Fig: M5Stack Core 2

Introduction to Arduino

An overview of Arduino IDE

Overview

- Introduction to Arduino Platform
- Why is Arduino useful?
- Arduino boards and IDE
- Arduino IDE interface
- Programming in Arduino Platform

Introduction to Arduino

Open-source electronic prototyping platform

Consists of hardware (Arduino Boards) and software (Arduino IDE)

Designed for rapid prototyping and learning by doing

Why Arduino?

Easy entry to embedded systems

No deep electronics or programming background required

Widely used in academia and industry

Mostly used in prototyping, robotics, automation, IoT and smart systems.

Arduino Ecosystem

- Arduino Board (e.g., Uno, Nano, Mega)
- Arduino IDE
- Sensors & Actuators
- Board Support Packages
- Libraries
- Community & documentation

Popular Arduino Boards

Arduino Uno (most popular)

Arduino Nano (compact)

Arduino Mega (more I/O pins)

Arduino Due (advanced)

Key differences are in size, memory, number of pins and cost

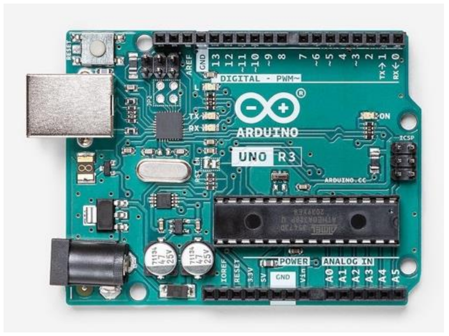


Fig: Arduino UNO – a popular Arduino Board

What is Arduino IDE?

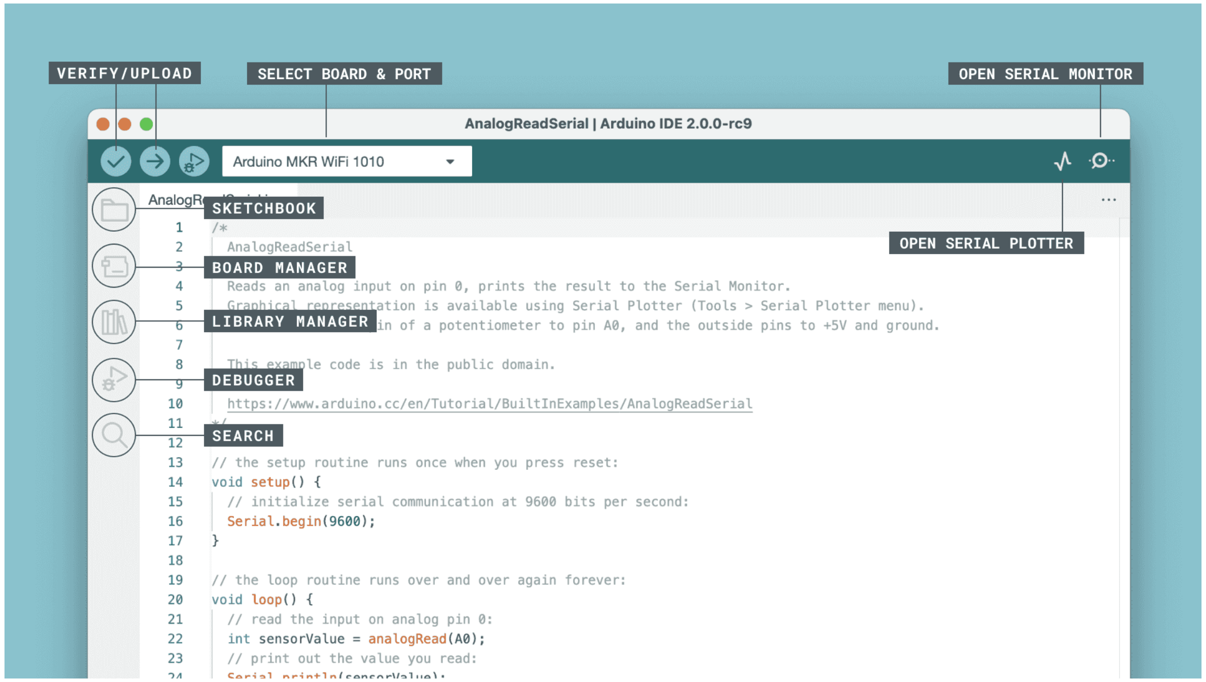
A software (Integrated Development Environment) used to

- Write code
- Compile code
- Upload code to Arduino Boards

Free and cross-platform. It runs on Windows, MacOS, Linux

Arduino IDE Interface

- Code Editor
- Toolbar
- Message Area
- Console
- Board & Port Selection



```
/*
  AnalogReadSerial

  Reads an analog input on pin 0, prints the result to the Serial Monitor.
  Graphical representation is available using Serial Plotter (Tools > Serial Plotter menu).
  in of a potentiometer to pin A0, and the outside pins to +5V and ground.

  This example code is in the public domain.

  https://www.arduino.cc/en/Tutorial/BuiltInExamples/AnalogReadSerial
*/

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
```

Fig: Arduino IDE interface

Arduino Programming Language

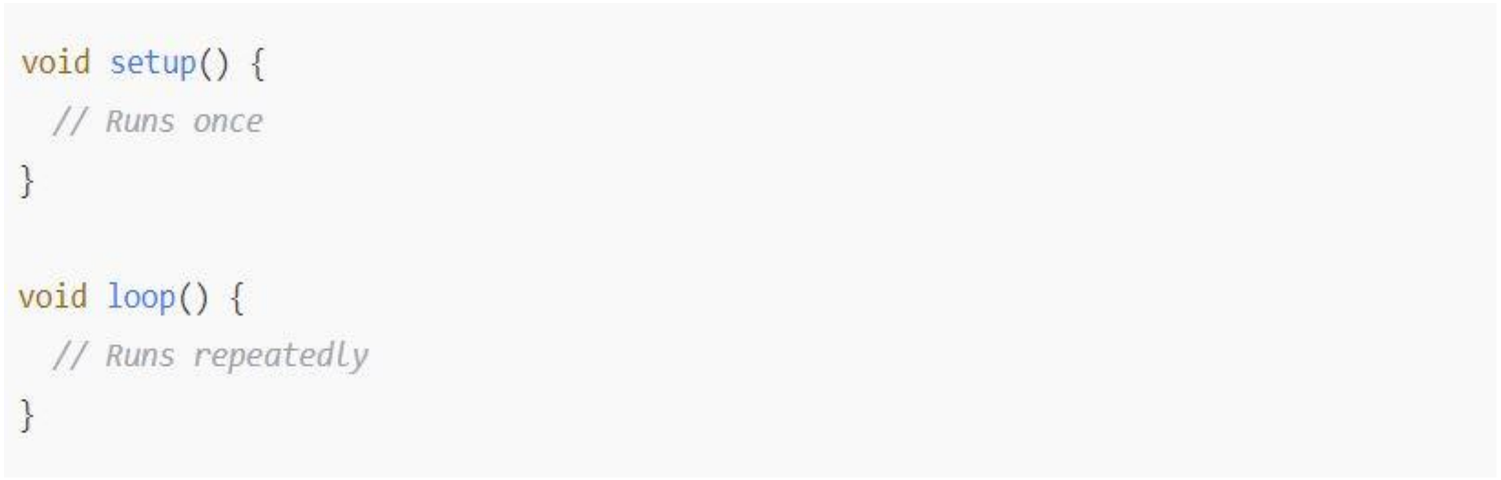
Based on:

- C
- C++

Simplified for beginners

Uses ready-made libraries

Structure of an Arduino Program



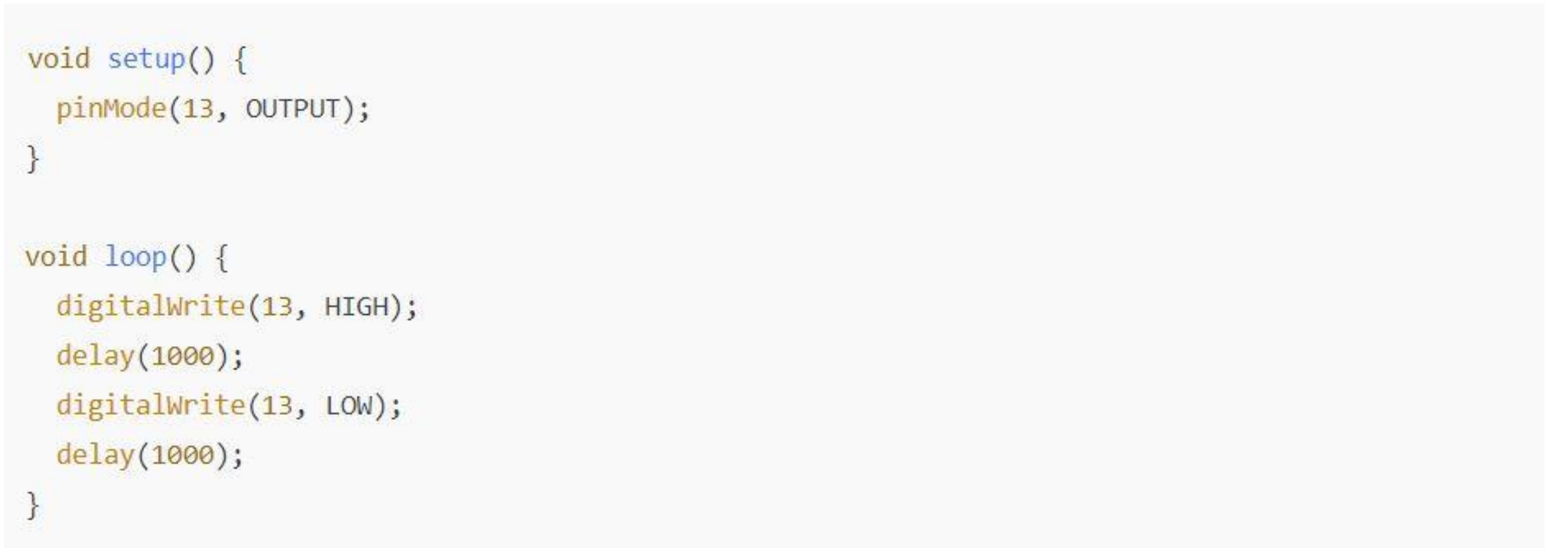
```
void setup() {
  // Runs once
}

void loop() {
  // Runs repeatedly
}
```

`setup()` → Initialization

`loop()` → Main logic

Example: Blink LED



```
void setup() {
  pinMode(13, OUTPUT);
}

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

Turns LED ON and OFF every second
“Hello World” of Arduino

Introduction to C Programming for Arduino

A very basic Overview

Overview

Basic C syntax rules

Variables

Operators

Conditional statements

Loops

Functions

Basic C Syntax Rules

- C is case-sensitive
- Every statement ends with ;
- Blocks of code use { }
- Comments:

```
// Single line comment
/* Multi-line
   comment */
```

Variables (Storing Data)

- Variables store values in memory.

```
int ledPin = 13;
//int → integer number
//ledPin → variable name
//13 → value stored
```

- Common variable types:
  - int → whole numbers
  - float → decimal numbers
  - char → single character
  - bool → true / false

Using Variables in Arduino

```
int ledPin = 13;

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

- Variables make code:
    - Easier to read
    - Easier to modify
    - Change pin number in one place

Operators (Basic Math & Logic)

Arithmetic operators:

- + addition
- - subtraction
- * multiplication
- / division

Comparison operators:

- == equal to
- != not equal
- <, >, <=, >=

Conditional Statements (if–else)

- Used to make decisions

```
if (value > 500) {
  digitalWrite(13, HIGH);
} else {
  digitalWrite(13, LOW);
}
```

- Arduino reacts based on conditions
- Essential for sensors

Loops (Repeating Actions) : ‘for’ loop

Runs code fixed number of times

```
for (int i = 0; i < 5; i++) {
  digitalWrite(13, HIGH);
  delay(500);
  digitalWrite(13, LOW);
  delay(500);
}
```

Loops (Repeating Actions) : ‘while’ loop

- Runs as long as condition is true
- Useful for waiting for events

```
while (buttonState == LOW) {
  digitalWrite(13, HIGH);
}
```

Functions (Reusable Code)

- Functions help organize code.

```
void blinkLED() {
  digitalWrite(13, HIGH);
  delay(500);
  digitalWrite(13, LOW);
  delay(500);
}
```

Calling a function

```
blinkLED();
```

Serial Communication (Debugging)

```
Serial.begin(9600);
Serial.println(sensorValue);
```

- Used to:
  - Print values
  - Debug programs
  - View output in Serial Monitor