# Embedded C Programming

## 01 - Introduction

Nitin Chandrachoodan, IIT Madras

# Introduction to Embedded Systems

# embedded system

***noun*** [ C ] • COMPUTING

**UK** /ɪmˌbed.ɪd ˈsɪs.təm/ **US** /ɪmˌbed.ɪd ˈsɪs.təm/

Add to word list

**a computer system that does a particular task inside a machine or larger electrical system:**

# Characteristics

- **Dedicated Functionality**
  - Unlike PCs - NOT general purpose
  - Perform specific tasks

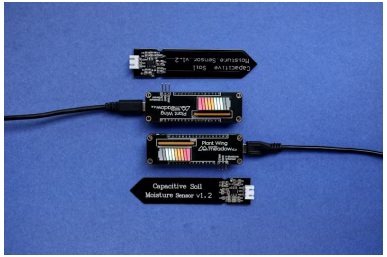
- **Real-time Operations**
  - Respond to changes in input/environment
    - Not the same as “fast”

# Components


Microcontrollers

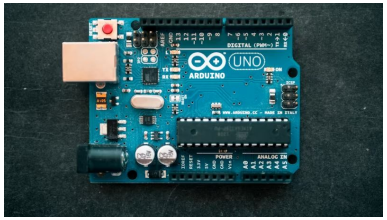
# Components


Microcontrollers

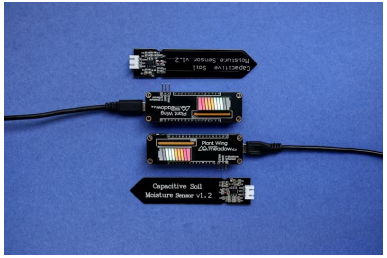



Sensors & Actuators

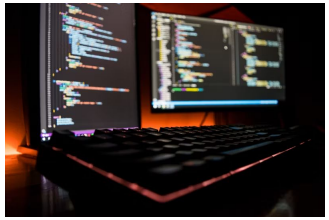
# Components




Microcontrollers

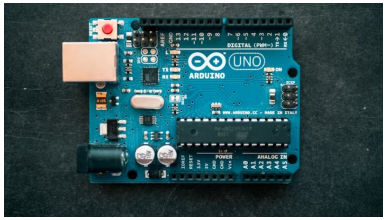



Sensors & Actuators

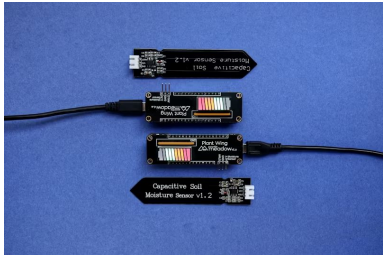


Software

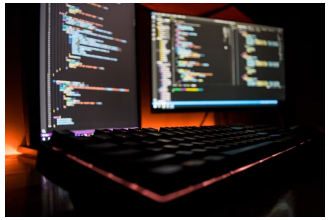
# Components




Microcontrollers




Sensors & Actuators



Software

System

Images src Unsplash - various photographers

# Examples in Everyday Life

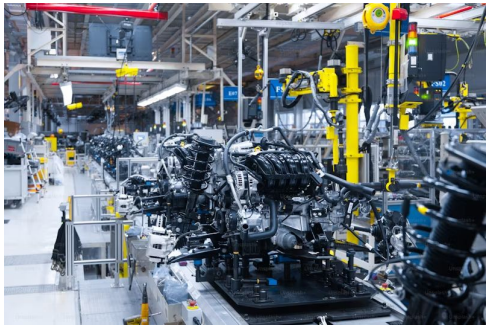

Kitchen

Entertainment

Communication


Travel / Automotive

Fitness



Manufacturing / Industries

Images src Unsplash - various photographers

# Evolution Drivers

- **Semiconductor Technology:**
  - miniaturization has made microcontrollers cheaper and more accessible
  - easier to program a microcontroller than build custom analog circuitry
- **Sensors and Actuators:**
  - Wider range of applications (partly driven by semiconductor evolution)
- **Communication Systems:**
  - Advances in wireless communication
  - Simpler / more flexible wired communication

# Key Application Areas and Impact

# Key Application Areas and Impact

- Automotive:
  - Engine control, fuel injection
  - navigation, climate control

# Key Application Areas and Impact

- Automotive:
  - Engine control, fuel injection
  - navigation, climate control
- Health care:
  - fitness bands / watches
  - continuous monitoring: glucose, blood pressure

# Key Application Areas and Impact

- Automotive:
  - Engine control, fuel injection
  - navigation, climate control
- Health care:
  - fitness bands / watches
  - continuous monitoring: glucose, blood pressure
- Consumer electronics:
  - Household goods, kitchen
  - Entertainment

# Key Application Areas and Impact

- Automotive:
  - Engine control, fuel injection
  - navigation, climate control
- Health care:
  - fitness bands / watches
  - continuous monitoring: glucose, blood pressure
- Consumer electronics:
  - Household goods, kitchen
  - Entertainment

- **Functionality**
  - Allow measurements, control that were not earlier possible

# Key Application Areas and Impact

- Automotive:
  - Engine control, fuel injection
  - navigation, climate control
- Health care:
  - fitness bands / watches
  - continuous monitoring: glucose, blood pressure
- Consumer electronics:
  - Household goods, kitchen
  - Entertainment

- **Functionality**
  - Allow measurements, control that were not earlier possible
- **Safety**
  - Continuous monitoring and feedback - automated

# Key Application Areas and Impact

- Automotive:
  - Engine control, fuel injection
  - navigation, climate control
- Health care:
  - fitness bands / watches
  - continuous monitoring: glucose, blood pressure
- Consumer electronics:
  - Household goods, kitchen
  - Entertainment

- **Functionality**
  - Allow measurements, control that were not earlier possible
- **Safety**
  - Continuous monitoring and feedback - automated
- Overall **user experience**

# General vs Custom

## General

- Run “applications”
  - browser, editor, mail
- General interaction methods:
  - keyboard, mouse, screen
- Programmable
  - by end user
  - loadable applications
- “Software”

# General vs Custom

## General

- Run “applications”
  - browser, editor, mail
- General interaction methods:
  - keyboard, mouse, screen
- Programmable
  - by end user
  - loadable applications
- “Software”

## Custom

- Fixed function
  - Pre-defined tasks
- Limited interaction
  - Keypads, push buttons
- Limited software programmability
- “Firmware”

“Embedded” within larger device

# Smartphones?

- Fixed function:
  - phone, communication
  - Sensors
- Limited interaction:
  - touchscreen, keypad
- Real-time
  - Interactivity
  - output control

- General purpose:
  - Programmable “apps”
- Broad compute capabilities
  - Phone can run Linux, desktop applications
- Operating systems
  - Filesystem access, network

# Requirements of Embedded Systems

- Limitations:
  - compute resource constraints
  - memory usually limited
  - few interaction mechanisms
  - space constraints
  - battery operation
- Real-time
  - Interactive
  - Guaranteed response times (esp. for safety)
  - Continuous operation

Performance Optimization for “Efficiency”

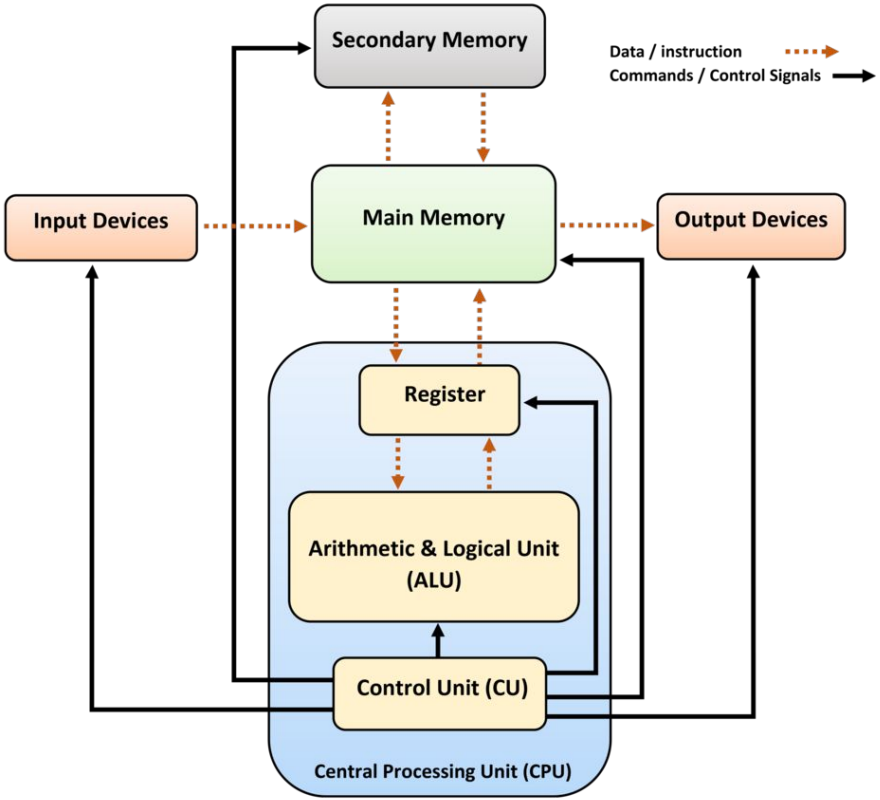
# Summary

- Systems that are “**embedded**” within a larger device
- Resource, space, cost constrained
- Fixed functionality
- Real-time and safety guarantees

- Largely enabled by advanced in semiconductor technology
- Affect every aspect of modern life

# Microcontrollers vs Microprocessors

# General Computer Architecture




Src: Wikipedia

# Definitions

**Microprocessor (MPU)**

- Core processing unit that provides computational power to the system
- Requires separate supporting circuits for memory, I/O etc.

# Definitions

**Microprocessor (MPU)**

- Core processing unit that provides computational power to the system
- Requires separate supporting circuits for memory, I/O etc.

**Microcontroller (MCU)**

- Integrates CPU, memory, I/O peripherals on a single chip
- Usually designed for specific applications

# Integration

- On-chip memory
  - RAM (working memory)
  - ROM - configuration, OS, firmware
  - EEPROM / Flash - flexibility + non-volatile
- Peripheral interfaces
  - Digital and Analog I/O
  - Timers
  - UART, SPI, I2C
  - Networking (WiFi, Bluetooth)
  - Camera / screen / high bandwidth interfaces (less common)

**System-on-Chip**

# Reasons for MCU design

- Minimize number of extra components
  - System-on-Chip design
- Battery lifetime: lower power consumption
- Direct control of physical systems
  - GPIO, bus interfaces for control and sensing
- Communication
  - Built-in communication primitives

**Cost effectiveness**

# Microprocessors Domain

## Computing and Data Processing

- Smartphones, personal computers, workstations, servers requiring:
  - complex computations
  - multi-tasking
- Advantages
  - Higher memory and storage resources - also flexible, expandable
  - Peripheral devices: expandable with USB, PCI etc
  - Higher processing power

# Microcontrollers Domain

## Embedded and Control Applications

- Appliances, Automotive systems, Consumer electronics
  - Fixed (or limited) functionality
  - Real-time guarantees: limited multitasking to ensure quick responses
  - Safety critical: airbags, traction control
- Advantages
  - Cost effective: only minimal resources as needed to solve problem
  - Limited or no expandability: not designed for USB expansion etc.

# Blurred Lines

- Embedded systems:
  - Increasing flexibility, firmware upgrades (washing machine with new cycle type…)
- Microprocessor systems:
  - Increasing integration: access to more peripherals tightly integrated
  - eg. Macbook M1 chip: general purpose processor but almost SoC
- Smartphones: Perfect blend of both sides
  - Downloadable apps: general, extendable, possible high performance
  - Phone, emergency services etc.: fixed function, critical components

Distinction between embedded and general purpose is blurred at best

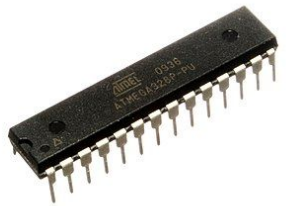
# Microcontroller Families

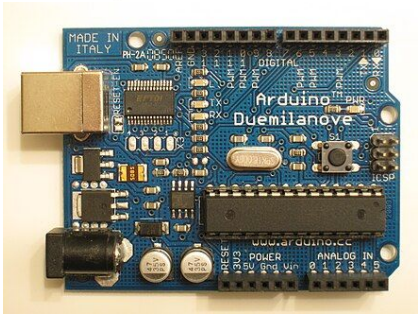
# Microcontroller “Families”

- **Instruction Set**: flashback to Programming in C - how does a processor understand instructions?
- **Instruction Set Architecture**: specific instructions used by a processor
- Equivalence:
  - In principle all ISAs are equivalent (almost… memory size etc. may matter)
  - Common instructions better tuned for certain types of operations (loops etc.)
  - Register-based (RISC), stack, accumulator: different architectures need different optimizations

# MCU family: AVR

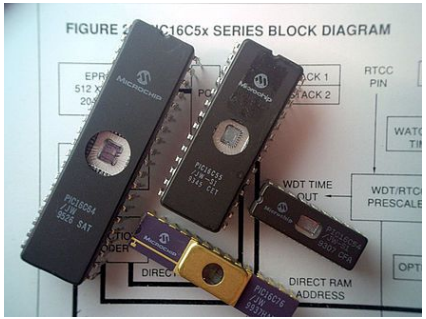
- Atmel (since 1996), Microchip acquired
  - Flash based
  - 8-bit, extended later
- **Arduino**: immensely popular - “democratization” of MCU coding
  - Especially common in hobbyist domains
- Carries over to other areas as well:
  - Consumer electronics, automotive
  - Wearables
  - Art and art installations

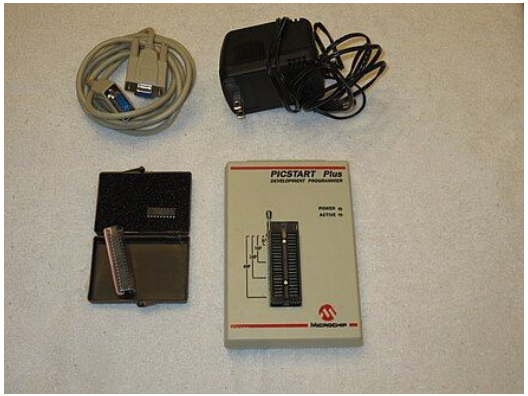




# MCU family: PIC

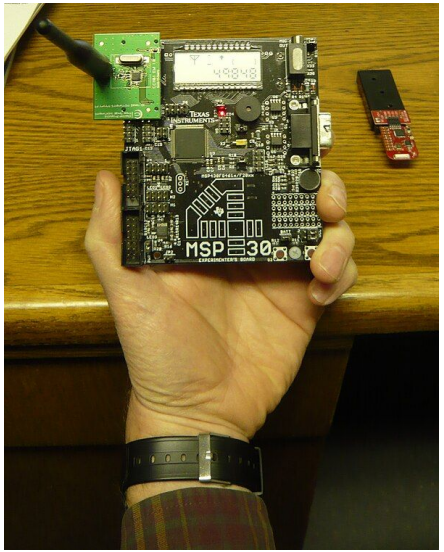
- From 1976 - Microchip Technology Inc.
- EEPROM and similar tech
- Latest PIC32 - MIPS ISA based
- ISA well suited to lookup tables
- Very popular with hobbyists
  - Low cost, ease of use
  - Largely displaced by Arduino
  - Somewhat complex debugging hardware needed



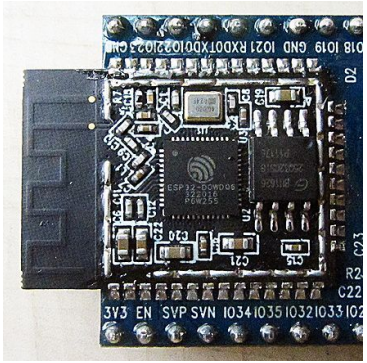
# MCU family: MSP 430

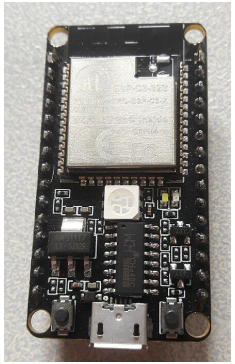
- Texas Instruments (TI) since 1992
- Ultra-low-power application focus
  - 0.1uA RAM retention, 0.7uA RTClock operation
  - Possibly **years of standby** on single coin cell
- Launchpad kits targeted at University programs
  - Easy to use USB-power/debug




# Microcontroller family: ESP32

- Espressif systems (Shanghai based)
- Low-cost, high levels of integration
  - Many peripherals, many communication options
  - Typically include Bluetooth and WiFi
- Tensilica XTensa ISA (Cadence)
- NodeMCU: ultra-low-cost boards



# Which one to choose?

- Highly competitive market:
  - similar features
  - high levels of integration
  - low cost
- Infrastructure:
  - Arduino Processing IDE made a big difference: easier to code
  - Popularity begets popularity…

One ISA/family even more popular in numbers….

# ARM


- “Advanced RISC Machine” (originally Acorn RISC Machine)
- Architecture started ~ mid 1980s: small personal/business computers
- RISC: Reduced Instruction Set Computing
  - Important development in computer architecture - big impact on performance/power tradeoffs

Over **230 Billion** ARM processors produced as of 2022

# IP-based model

- “Intellectual Property”
  - ARM sells “licenses” to use intellectual property, not chips directly
  - No physical products
- “Netlist” - design in synthesizable form - various forms of digital design
  - Gate level
  - RTL (register transfer level)
  - Architectural licenses (Apple, Qualcomm, NVidia, Ampere etc.)
- 32- and 64-bit variants: different application domains

# ARM Cortex Series

Variants of 32-bit architecture

- **A - Application series:**
  - Memory management, full OS support
  - 32- and 64-bit variants: core of most phones, laptop/netbook etc.
- **R - Real-time:**
  - Tightly coupled memory, possibly with error correction, fault tolerance
  - Exception handling support in hardware - safety critical applications
- **M - Microcontroller:**
  - No memory management, mostly standalone mode
  - Low-power, embedded applications - primary focus of this course

# ARM Cortex-M features

- Mostly licensed as synthesizable Verilog code - customizable
- Useful microcontroller features:
  - Handle large number of interrupts
  - Harvard architecture: split instruction / data for more efficient access
  - Limited pipeline - performance is not primarily goal
  - Limited math - may not support floating point in hardware
  - Very good (multiple) low-power modes and switchability
- ISA based on ARMv7 or incremental variants
  - Focus here is on C/C++ coding, not assembly

# Microprocessor Features - Processing Capability

- Processor speed:
  - generally no fixed lower limit, but used mostly in low power modes
  - MSP 430 particularly good at low power - has very low leakage design
  - high speed useful for real-time control response as well as complex control
- 32-b vs 64-b architecture
  - Access more memory
  - Larger registers
  - May mean more power - not preferred for microcontrollers
- 8- or 16-b microcontroller
  - 8051, PIC, AVR: good for limited functionality, can be smaller and lower power

# Microprocessor Features - Power

- Operating power - generally determined by frequency and capabilities
  - Smaller instructions, registers etc good
  - Lower operating frequency
    - But means less compute possible
- Standby power: able to sleep for long durations and wake up to respond
  - Leakage power: fundamental problem in digital circuits
  - Clock gating, power gating, many circuit level techniques applied
  - Target - months or years on button cell batteries: ~ < uA current
  - MSP 430 series strong capabilities
  - Most uCs have various clock and power modes: careful control can extend life

# Microprocessor Features - Real-time and Multitasking

- Handle multiple tasks simultaneously
- Generally requires more memory
- Handle many different peripherals and control in one chip
- Cortex-M series
  - good support for large number of interrupts
  - 32-b processor: handle more memory and switch

# Ecosystem and Development Support

- Traditional coding and debugging:
  - restricted IDE (Integrated Development Environment)
  - Complex terminology, non-trivial to get into
- Arduino - Processing IDE
  - “Democratization” of access
  - Hide most of complexity, accessible to beginners and non-engineers
  - Adopted by multiple others - MBed for ARM, ESP32 IDEs
- Knowledge of detailed debug enables more:
  - Proper control of memory usage
  - Single step debugging of code

# How to choose?

Traditionally:

- ESP32 - connected devices
- AVR (Arduino) - hobby and educational
- PIC - robust industrial
- MSP 430 - low power
- ARM - scalable, high performance

# How to choose?

Traditionally:

- ESP32 - connected devices
- AVR (Arduino) - hobby and educational
- PIC - robust industrial
- MSP 430 - low power
- ARM - scalable, high performance

Now:

- Most processors have overlapping features
- Understand capabilities and choose based on connectivity, memory
- Ecosystem plays an important role

# Language Choices

# Approaches to Programming

- Assembly language: direct control
- High level languages:
  - C, C++, Rust
- Visual programming languages
  - Labview, Blockly
- Beginner friendly languages
  - Processing

# Assembly Language

- Direct manipulation of the instruction code
- Requires detailed knowledge of the ISA (Instruction Set Architecture)
- Fragile:
  - Strongly dependent on specific microcontroller
  - Custom extensions / instructions make it hard to change
- “Structured programming” difficult

# High-level languages

- Structured programming:
  - Functions
  - Iteration
  - Data types, structures
- C / C++: closest to hardware, good control possible
  - Allow arbitrary memory allocation, manipulation
  - Registers, IO
  - Potentially unsafe access to memory
- Rust, Go (tinygo), MicroPython
  - relatively new languages in embedded space
  - Solve many of the memory safety problems, provide higher abstraction levels

# Embedded Programming

Patterns not common in regular C

- Direct register access
- Memory Mapped IO
- Volatile variables
- Interrupts
- Timer

```c
#include <stdint.h>
#define GPIO_PORTF_DATA_R       (*((volatile uint32_t *)0x400253FC))
#define GPIO_PORTF_DIR_R        (*((volatile uint32_t *)0x40025400))
#define GPIO_PORTF_DEN_R        (*((volatile uint32_t *)0x4002551C))

int main() {
    GPIO_PORTF_DIR_R |= 0x04;    // Set PF2 as output
    GPIO_PORTF_DEN_R |= 0x04;    // Enable PF2
    GPIO_PORTF_DATA_R |= 0x04;   // Set PF2 to high
}
```

```c
#include <stdint.h>
#define GPIO_PORTA_DATA_R       (*((volatile uint32_t *)0x400043FC))

void TimerOIntHandler(void) {
    TimerOICR = 0x01;           // Clear the timer interrupt
    GPIO_PORTA_DATA_R ^= 0x01; // Toggle PA0
}
```

# C for Embedded Systems

- Proximity to Hardware
  - Code to directly access memory, Registers
  - Interrupts, Timers, Peripherals
- Portability
  - Machine specific details isolated in header files
  - Logic focuses on bit manipulation, register/memory level control
- Legacy
  - Lot of old code surviving in C - other languages too but smaller numbers
  - Maintenance and Upgrades

# C++ - Abstraction and Generalization

- Encapsulate common functions into a “class”
  - Similar to C struct but with functions (methods) etc.
- Templates
  - Generic programming that allows similar code for different data types
- Limited use of object-oriented programming
  - Similar peripherals can benefit from common functions and code
  - Minimal inheritance used: code needs to be explicit in most cases

# Other Languages

## Very high level

- Processing - Arduino etc.
  - Targeted more towards arts, education
  - Focus on functionality, not control
- Python / Micropython
  - Interesting trade off between control and ease
  - Interpreter overhead difficult for resource constrained systems

## Visual Programming

- NI Labview, Blockly, …
  - Ease of use for non-programmers
  - Limited control

## Traditional

- Java
  - VM and bytecode overhead
  - Large installed base and well known
- Go / TinyGo
  - Easy to pick up, good degree of control
  - Memory management more difficult

## Rust

- Very promising for the future
- Good memory safety features
- High performance
- Strong challenger to C/C++

# Summary - why C?

- Good tradeoff between practicality and ease of use
  - Large installed codebase, well tested
  - Good control over memory use, memory/register management, close to Assembly
- Practicality
  - Industry standards exist
  - Safety critical systems very reluctant to try new languages
- Tools and Educational Material
  - Bulk of training material still in C
  - Community support, Forums etc.