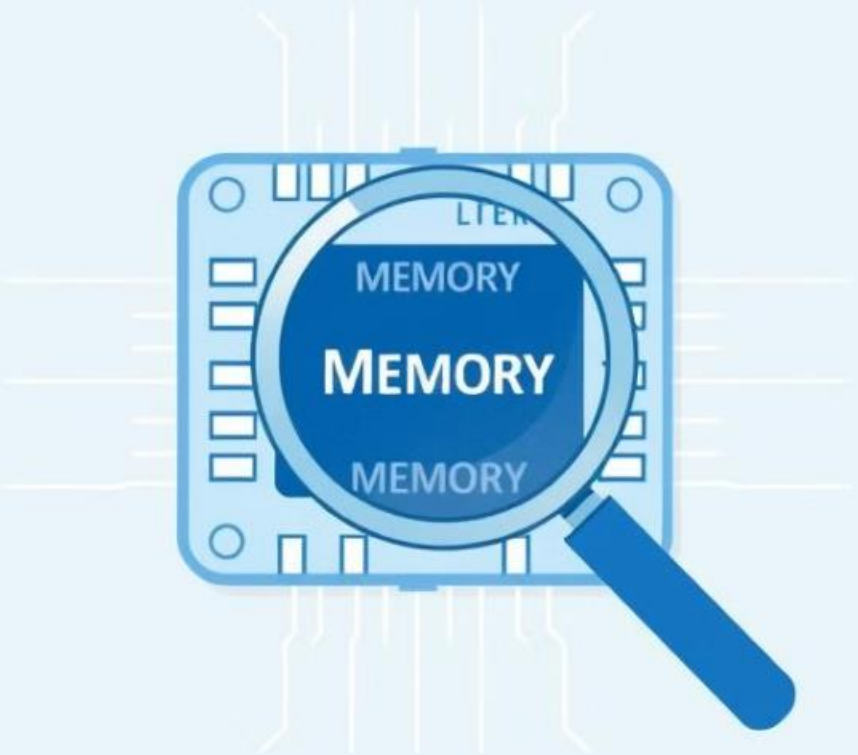
EMBEDDED SYSTEMS (CMPE 30274)
Memory
Module 4



Objectives

In this lesson, we will explore the different type of memory, their characteristics, and their selection criteria for embedded systems.

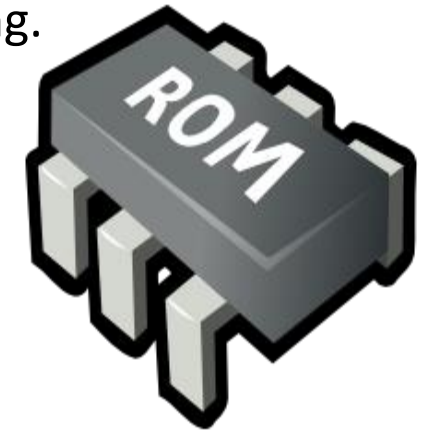
Memory in Embedded Systems

A. Read-Only Memory (ROM)
B. Random Access Memory (RAM)
C. Memory According to the Type of Interface
D. Memory Shadowing
E. Memory Selection for Embedded Systems

A. ROM

ROM

Read-Only Memory (ROM)
- a non-volatile type of memory commonly used in embedded systems.
- data stored in ROM cannot be modified or erased once it is programmed during manufacturing.



Definition and Characteristics:

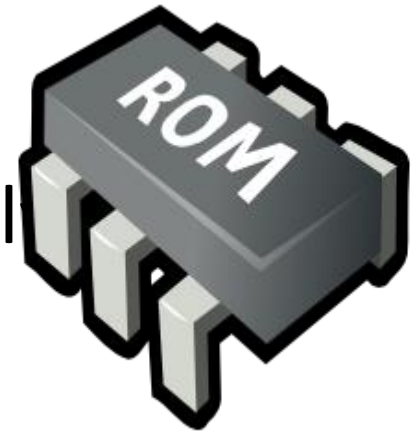
ROM is a type of memory that retains its contents even when the power is turned off.
- used to store firmware, boot loaders, and system configuration data that are essential for the functioning of an embedded system.

ROM is often referred to as "non-volatile" because it retains data without the need for a continuous power supply.

The **data stored** in ROM is **non-erasable and non-modifiable**, making it ideal for storing critical system information.

Types of ROM

1. Mask ROM
2. PROM (Programmable Read-Only Memory)
3. EPROM (Erasable Programmable Read-Only Memory)
4. EEPROM (Electrically Erasable Programmable Read-Only Memory)
5. Flash Memory



Types of ROM

Mask ROM: This type of ROM is manufactured with a fixed pattern during the chip fabrication process.
The data is permanently embedded in the ROM, and it cannot be changed or updated.

PROM (Programmable ROM): PROM allows users to program the memory once by using special programming equipment.
Once programmed, the data becomes permanent and cannot be altered.

Types of ROM

EPROM (Erasable Programmable ROM)

EPROM is similar to PROM but can be erased and reprogrammed using ultraviolet light exposure. It requires a **special erasing device to remove the existing data** before reprogramming.

EEPROM (Electrically Erasable Programmable ROM)

EEPROM can be electrically erased and reprogrammed using specific voltage levels. Unlike EPROM, EEPROM **does not require UV light for erasing** and is more convenient for in-circuit updates.

Flash Memory:

Flash is a type of EEPROM that allows multiple-byte erasure. Widely used in embedded systems due to its high density, fast erasure, and reprogramming capabilities.

In Arduino Uno,

Mask ROM: the mask ROM would contain low-level instructions and initialization code that are hardcoded into the microcontroller during manufacturing.

PROM (Programmable Read-Only Memory): PROM is not commonly used for storing user programs. However, it could be used to store custom bootloaders or specialized firmware if a user wants to replace the default bootloader with a customized one.

EPROM (Erasable Programmable ROM): does not typically use EPROM for user programs. However, it can be used for storing custom bootloaders or firmware that may require occasional updates or modifications during development.

EEPROM (Electrically Erasable Programmable ROM): EEPROM is commonly used for storing system configuration data, such as default settings, calibration parameters, and user preferences.
For example, it can store values like the brightness level of an LCD display or the last state of an input/output pin.

Flash Memory: the majority of the flash memory is used for storing user programs or sketches. **When you write a program using the Arduino IDE and upload it to the board, it gets stored in the flash memory**, allowing the microcontroller to execute the program's instructions. Arduino's flash memory can be thought of as the long-term storage space of the Arduino board where your programs, called sketches, are stored.

Application and Limitations

Applications :

Firmware Storage: ROM is commonly used to store firmware, which includes the program code that controls the embedded system's operation. It provides a reliable and permanent storage medium for critical software components.

Boot Loaders: Boot loaders, responsible for initializing the system and loading the operating system, are often stored in ROM. This ensures that the bootloader remains intact and cannot be modified by unauthorized sources.

System Configuration Data: ROM is useful for storing configuration data, such as calibration parameters, default settings, and system-specific information that needs to be retained even during power cycles.

Limitations: One limitation of ROM is that **it cannot be modified or updated once programmed**. This makes it unsuitable for storing frequently changing data or dynamic information.
Additionally, certain types of ROM, such as Mask ROM, have **higher manufacturing costs** and **require upfront design decisions** as they cannot be changed after fabrication.

B. Random Access Memory (RAM)

B. Random Access Memory (RAM):

Definition and Characteristics:

- Volatile memory
- Allows read and write operations
- Examples: Static RAM (SRAM), Dynamic RAM (DRAM)

Applications and Limitations:

- Storing program code, data, and variables during runtime
- Faster access but larger in size compared to ROM

RAM

B. Random Access Memory (RAM):

Random Access Memory (RAM) is a type of volatile memory commonly used in embedded systems. It is called "random access" because it allows for both read and write operations, and data can be accessed in any order, unlike sequential access memory.

Volatile Memory:

RAM is volatile, which means that it requires a continuous power supply to retain its stored data. When the power is turned off or lost, the data stored in RAM is lost as well.

The volatile nature of RAM makes it suitable for storing temporary data and variables that are needed during the runtime of a program but are not required to be retained once the power is off.

RAM

Applications :

Storing Program Code, Data, and Variables:. RAM is used in embedded systems to store variables, functions during the runtime of a program.

Faster Access but Larger in Size: RAM provides faster access to data compared to ROM because it allows both read and write operations. However, it typically requires more physical space compared to ROM due to the additional circuitry required for read and write capabilities.

Limitations of RAM:

Limited Capacity: RAM has a limited capacity compared to other forms of storage like ROM. The amount of RAM available on a microcontroller or embedded system is fixed and can impose constraints on the complexity and size of programs that can be executed.

Volatility: RAM is volatile, meaning it cannot retain data without a continuous power supply. This limitation necessitates the need for saving critical data to non-volatile memory like EEPROM or external storage devices to avoid data loss during power cycles.

Static RAM (SRAM) and Dynamic RAM (DRAM)

Static RAM (SRAM)

- a type of volatile memory that uses bistable latching circuitry to store each bit.
- It retains data as long as power is supplied.

Characteristics:

- **Speed**: Faster than DRAM, making it suitable for cache memory in CPUs.
- **Structure**: Consists of multiple transistors (typically 6 per bit) without needing refresh cycles.
- **Power Consumption**: Consumes more power than DRAM when idle.
- **Cost**: More expensive per bit compared to DRAM due to its complexity.

Use Cases: Often used for cache memory, small buffers, and in applications requiring high-speed data access.

Dynamic RAM (DRAM)

• another type of volatile memory that stores each bit of data in a separate capacitor within an integrated circuit. It needs to be refreshed periodically to maintain data.

•**Characteristics**:

- **Speed**: Slower than SRAM, but faster than non-volatile memory types.
- **Structure**: Requires fewer transistors (typically 1 transistor and 1 capacitor per bit), which allows for higher density.
- **Power Consumption**: Consumes less power when idle compared to SRAM, but uses more during refresh cycles.
- **Cost**: Cheaper per bit than SRAM, making it suitable for larger memory applications.

Use Cases: Used in embedded systems where larger capacity is needed.

SRAM: Fast, more expensive, used for small, high-speed applications.

DRAM: Slower, cheaper, used for larger memory needs.

C. Memory According to the Type of Interface

Memory According to the Type of Interface:

1. Parallel Memory Interfaces

Parallel memory interfaces allow multiple bits of data to be transmitted simultaneously.
This type of memory is typically **faster** due to the simultaneous transfer of data bits.

•SRAM (Static Random Access Memory):

- **Interface:** Typically uses a parallel interface.
- **Characteristics:** Fast, low latency, and volatile. Ideal for cache memory in microcontrollers.

•DRAM (Dynamic Random Access Memory):

- **Interface:** Also commonly employs a parallel interface.
- **Characteristics:** Slower than SRAM but offers higher density. Requires periodic refreshing to maintain data.

Memory According to the Type of Interface:

2. Serial Memory Interfaces

Serial memory interfaces transmit data one bit at a time, which can lead to lower complexity and reduced pin count.

- **SPI (Serial Peripheral Interface):**
  - **Characteristics:** Full-duplex, allowing simultaneous send and receive.

  Commonly used for connecting microcontrollers to peripherals like sensors and flash memory.

- **I2C (Inter-Integrated Circuit):**
  - **Characteristics:** Multi-master, multi-slave, and uses a two-wire interface (SDA and SCL).

  Suitable for short-distance communication between multiple devices.

- **UART (Universal Asynchronous Receiver-Transmitter):**
  - **Characteristics:** Asynchronous serial communication.
  - Often used for debugging or communication between microcontrollers.

Memory According to the Type of Interface:

3. Memory-Mapped Interfaces

In embedded systems, certain memory types can be accessed as part of the system's address space:

•Flash Memory:

- **Interface:** Often memory-mapped.
- **Characteristics:** Non-volatile, used for storing firmware and application code. Can be accessed similarly to RAM but is slower.

•EEPROM (Electrically Erasable Programmable Read-Only Memory):

- **Interface:** Typically provides a memory-mapped interface.
- **Characteristics:** Non-volatile and allows selective byte-wise data erasure and rewriting.

Memory

Memory-mapped interface - a method of interfacing hardware devices with a microcontroller through memory addresses.

Specific memory addresses in the system's address space are allocated for hardware devices, allowing them to be accessed like regular memory locations.

Hardware devices (like sensors) are assigned specific addresses within the main memory range.

This means that the CPU can read from or write to these devices using standard memory operations.

Flash Memory

•**Description:** Most embedded systems, such as microcontrollers (e.g., Arduino), use flash memory to store sketches.

Flash memory retains its contents without power and can be electrically erased and reprogrammed.

•**Characteristics:**

- **Non-volatile:** Retains data without power.
- **Writable:** Can be written to and erased in blocks.
- **Durability:** Limited number of write/erase cycles.

Memory According to the Type of Interface:

4. Direct Memory Access (DMA)

DMA allows peripherals to access memory without CPU intervention, optimizing data transfer rates:

•**Characteristics:** Reduces CPU load and improves system performance. Commonly used in applications requiring high-speed data transfer, like audio and video processing.

Memory-Mapped I/O

```
// Memory-mapped I/O example using Arduino Uno

// Define the memory addresses for the I/O device
const int LED_PIN = 0x20;  // Address for the LED pin

void setup() {
  // Configure the LED pin as an output
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  // Turn on the LED
  digitalWrite(LED_PIN, HIGH);
  delay(1000);  // Wait for 1 second

  // Turn off the LED
  digitalWrite(LED_PIN, LOW);
  delay(1000);  // Wait for 1 second
}
```

D. Memory Shadowing

D. Memory Shadowing:

Definition and Purpose:

- Duplicating memory content in different address spaces
- Enhances system performance and flexibility

Shadow RAM and Shadow ROM:

- Using RAM or ROM as a shadow copy of another memory region
- Applications and benefits

Memory

Memory shadowing is a technique used to improve performance and reliability by creating a duplicate or "shadow" copy of a section of memory.

- **Duplicating critical data** or program code in a secondary memory space.
- Purpose is to ensure that the system can **continue to operate correctly** even in the event of a fault or error in the primary memory.

Key Characteristics of Memory Shadowing:

1.Redundancy: Provides a backup of critical data or instructions, mitigating the risk of data loss or corruption.
2.Performance Improvement: By keeping frequently accessed data in a faster, shadow memory, systems can reduce access times.
3.Error Detection and Recovery: Allows for the detection of discrepancies between primary and shadow memory, enabling systems to recover from errors.

Example of Memory Shadowing in a Weather Temperature Monitor

1. **Data Collection**: The monitor collects temperature readings from sensors.
2. **Primary Memory Storage**: New readings are stored in primary memory (e.g., RAM).
3. **Shadow Memory Storage**: Each reading is simultaneously written to shadow memory (e.g., EEPROM).
4. **Data Integrity Checks**: The system regularly compares the primary and shadow memory. If discrepancies are found, it reverts to the shadow memory for accurate data.
5. **Error Recovery**: In case of primary memory failure, the monitor uses the shadow memory to maintain access to the last known good readings.

E. Memory Selection for Embedded Systems

Common Factors for Selecting Memory in Embedded Systems

1. **Type of Memory**:
   Choose between volatile (e.g., SRAM, DRAM) and non-volatile (e.g., Flash, EEPROM) based on the application needs.
2. **Performance Requirements**:
   Assess access speed and data throughput required for the application, especially for real-time processing.
3. **Capacity Needs**:
   Determine the required memory size for current and future data storage, taking into account firmware and data logging.
4. **Power Consumption**:
   Evaluate energy efficiency, especially for battery-operated devices. Consider the power profiles of different memory types.
5. **Cost**:
   Analyze the budget available for memory. Balance cost with performance and capacity needs.

Common Factors for Selecting Memory in Embedded Systems

6. **Reliability and Durability:**
   Consider endurance (write/erase cycles) and error correction capabilities for critical applications.
7. **Interface Compatibility:**
   Ensure the selected memory is compatible with the microcontroller or processor's bus architecture.
8. **Data Retention:**
   Assess how long the data must be retained without power and choose memory types accordingly.
9. **Size and Form Factor:**
   Consider the physical size of the memory chip and how it fits into the design of the embedded system.
10. **Availability and Support:**
    Ensure the selected memory is readily available and supported by the manufacturer for long-term projects.

Comparison of ESP32 and Arduino Uno

Specification	Arduino Uno	ESP32
Microcontroller	ATmega328P	Dual-core Tensilica LX6
Operating Voltage	5V	3.3V
Input Voltage (recommended)	7-12V	5V (varies by module)
Digital I/O Pins	14 (6 PWM)	34 (varies by module)
Analog Input Pins	6	18 (12-bit ADC)
Clock Speed	16 MHz	Up to 240 MHz
Flash Memory	32 KB (0.5 KB for bootloader)	Typically 4 MB (some models 8 MB)
SRAM	2 KB	520 KB
EEPROM	1 KB	None (but can use flash for storage)
Wi-Fi	No	Yes (802.11 b/g/n)
Bluetooth	No	Yes (v4.2)

Reminder :

1. Quiz 1 next face to face meeting, March 17, 2025
Coverage : Module 1,2,3,4 and laboratory activities

2. Group seatwork to be presented on next online meeting March 25, 2025.

Reminder :

Quiz 1 next face to face meeting, March 17, 2025
Coverage : Module 1,2,3,4 and laboratory activities

End of presentation