Reading and Writing Multiple Bytes

EEPROM allows reading and writing only **one byte at a time**, but some data types (e.g., `int`, `float`) require multiple bytes.

- **Example problem:**
  - An `int` is 2 bytes, but `EEPROM.write()` handles only 1 byte.
  - Solution: Store the **low byte** and **high byte** separately.

Writing an `int` (2 bytes) to EEPROM:

```
int value = 258;
EEPROM.write(0, value & 0xFF);         // Low byte
EEPROM.write(1, (value >> 8) & 0xFF); // High byte
```

Reading an `int` back from EEPROM:

```
int value = EEPROM.read(0) | (EEPROM.read(1) << 8);
```

- **Bitwise masking** is used to extract individual bytes:
  - `value & 0xFF` → Extracts the **low byte**.
  - `(value >> 8) & 0xFF` → Extracts the **high byte**.
- Data is reconstructed by combining the **low and high bytes**.

Understanding the Code: Storing and Retrieving a 2-Byte Integer in EEPROM

The code is handling an **`int` value (258)** and storing it in an EEPROM memory that writes data in **single-byte (8-bit) chunks**. Since an `int` in this case is **2 bytes (16 bits)**, it must be broken into two separate **8-bit values** before storing in EEPROM.

Let's break it down step by step.

Step 1: Understanding `258` in Binary

An **`int` (2 bytes = 16 bits)** is used to store `258`. In **binary (16-bit representation)**:

```
258 (decimal) = 0000 0001 0000 0010 (binary)
```

This consists of:

- **Low Byte** (first 8 bits) → `0000 0010` (decimal: `2`)
- **High Byte** (next 8 bits) → `0000 0001` (decimal: `1`)

Step 2: Writing `258` to EEPROM

EEPROM can store only **1 byte (8 bits) per memory address**, so we must store `258` in **two separate bytes**:

```
EEPROM.write(0, value & 0xFF);         // Low byte
EEPROM.write(1, (value >> 8) & 0xFF); // High byte
```

Breaking Down Each Write Operation

1. **Storing the Low Byte (`0000 0010` = `2`) at address `0`**

```
EEPROM.write(0, value & 0xFF);
```

- `value & 0xFF` means **bitwise AND** with `0xFF` (which is `1111 1111` in binary).
- This extracts **only the lowest 8 bits**.
- **Result:** `0000 0010` (decimal: `2`) is stored in EEPROM at address `0`.

Storing the High Byte (`0000 0001` = `1`) at address `1`

```
EEPROM.write(1, (value >> 8) & 0xFF);
```

- `(value >> 8)` shifts the bits **8 places to the right**:

```
  0000 0001 0000 0010  (original: 258)
→ 0000 0000 0000 0001  (after shift, decimal: `1`)
```

- `& 0xFF` ensures we keep only the **lowest 8 bits** of the result.
- **Result:** `0000 0001` (decimal: `1`) is stored in EEPROM at address `1`.

Step 3: Reading 258 Back from EEPROM

Now we need to **reconstruct** the `int` from two separate bytes stored in EEPROM:

```
int value = EEPROM.read(0) | (EEPROM.read(1) << 8);
```

Breaking Down Each Read Operation

1. **Reading Low Byte from address `0` (contains `0000 0010` = `2`)**

```
EEPROM.read(0)    → 0000 0010 (decimal: `2`)
```

2. Reading High Byte from address `1` (contains `0000 0001` = `1`)

```
EEPROM.read(1) << 8
```

- `EEPROM.read(1)` gives `1` (`0000 0001`).
- **Shifting left by 8 bits (`<< 8`)** moves it back into its original position:

```
0000 0001 → 0000 0001 0000 0000 (decimal: 256)
```

3. Reconstructing the Original Value

```
int value = EEPROM.read(0) | (EEPROM.read(1) << 8);
```

- **Bitwise OR (`|`)** combines the two bytes:

```
   0000 0001 0000 0000  (256)
OR 0000 0000 0000 0010  (2)
----------------------
   0000 0001 0000 0010  (258)
```

    - The reconstructed `value` is **258**.

Summary

1. **Writing an int (`258`) to EEPROM:**
    - Store the **low byte** (`2`) at address `0`.
    - Store the **high byte** (`1`) at address `1`.
2. **Reading the int (`258`) from EEPROM:**
    - Retrieve the **low byte** from `EEPROM.read(0)`.
    - Retrieve the **high byte**, shift it left by 8 bits, and combine both using **bitwise OR** (`|`).

C bitwise operators