MCU Memory + GPIO Bare-Metal Notes

Summary of the concepts discussed in this chat (compiled on January 22, 2026).

1. Memory regions on an MCU

MCUs expose a single CPU address space, but different address ranges are wired to different on-chip resources. The 'memory map' tells you which addresses refer to Flash, SRAM, and memory-mapped peripherals.

- **Flash (non-volatile)**: stores firmware instructions (.text), constants (.rodata), and the initial image of initialized globals (.data).
- **SRAM (volatile)**: holds runtime data: stack, globals that change (.data/.bss), buffers, and (optionally) heap.
- **Peripheral address space**: addresses that do not represent RAM cells; they represent hardware registers (GPIO, UART, timers, etc.).
- **Boot ROM / system memory** (vendor-provided): can contain a factory bootloader used for programming in some modes.

2. Start address + size = region range

When a datasheet says 'RAM starts at 0x20000000 and has size 192 KB', it is describing a contiguous address range.

Common formula:

```
end_inclusive = start + size_bytes - 1
range = [start, start + size_bytes)   // half-open form
```

Worked examples (from the discussion):

```
RAM:   start = 0x20000000, size = 192 KB = 192*1024 = 196608 = 0x30000
       end   = 0x20000000 + 0x30000 - 1 = 0x2002FFFF

Flash: start = 0x08000000, size = 2 MB  = 2*1024*1024 = 2097152 = 0x200000
       end   = 0x08000000 + 0x200000 - 1 = 0x081FFFFF
```

Note: some MCUs split RAM into multiple banks (multiple ranges). In that case you follow the memory map table rather than assuming one continuous block.

3. Peripheral regions and 32-bit registers

A peripheral region is an address range reserved for a peripheral's register block. Inside that block, registers are typically 32-bit (4 bytes) wide and appear at fixed offsets from the base address.

- Base address selects the peripheral (e.g., GPIOA base).
- Offset selects a specific register (e.g., MODER, ODR, IDR).
- A 32-bit register occupies 4 consecutive byte addresses; registers typically appear at offsets 0x00, 0x04, 0x08, etc.

- This is memory-mapped I/O: reading/writing those addresses reads/writes hardware state, not normal RAM.

Bit-banding (when available) is an additional aliasing mechanism for individual bits; it is related to address mapping but is not required to understand basic memory-mapped peripherals.

4. GPIO ports, pins, and the MODER bitfield

GPIO is organized into ports (also called banks) like GPIOA, GPIOB, etc. Each port controls up to 16 pins (0-15). For GPIOA, pins are named PA0..PA15; for GPIOB, PB0..PB15, and so on.

The MODER register is 32 bits and packs 2 bits per pin: 16 pins * 2 bits = 32 bits.

```
Pin n mode bits are at positions (2*n + 1 : 2*n)
00 = input
01 = output
10 = alternate function
11 = analog
```

Dev-board header pins are routed to MCU pins (PAx/PBx/etc.). You usually connect Dupont wires to headers, which map to specific MCU pins. Some package pins may not be broken out on every board/package even though the port has bits for all 16 positions.

MODER covers all pins for a single GPIO port. The MCU has one MODER register per port (GPIOA_MODER, GPIOB_MODER, etc.), not one global MODER for the whole chip.

5. Pointers, structs, and volatile for registers

Two 'stars' often appear in register macros because they do different jobs: one defines a pointer type, and one dereferences the pointer.

```
#define GPIOA_BASE  0x40020000u
#define GPIOA_MODER (*(volatile uint32_t *)(GPIOA_BASE + 0x00u))

(volatile uint32_t *)  -> cast an address to a pointer-to-32-bit
*( ... )               -> dereference: access the 32-bit register at that address
```

Using a struct overlay makes register access more readable by matching register offsets to fields:

```
struct gpio {
  volatile uint32_t MODER, OTYPER, OSPEEDR, PUPDR, IDR, ODR, BSRR, LCKR, AFR[2];
};
#define GPIOA ((struct gpio *) 0x40020000)
```

A helper like gpio_set_mode touches only MODER because MODER specifically stores the mode (input/output/AF/analog). Other behaviors (pullups, output type, alternate function selection, etc.) live in other registers and would get their own helper functions.

6. Ports as 'banks' and packed pin IDs

Some STM32 families place GPIO port register blocks at a regular spacing (e.g., 0x400 bytes = 1 KB) so you can compute base addresses.

```
#define GPIO(bank_index) ((struct gpio *) (0x40020000 + 0x400 * (bank_index)))
```

```
// bank_index: A=0, B=1, C=2, ... (by convention)
```

To pass around 'a pin' as a single value, you can pack the bank and pin number into a 16-bit integer:

```
#define PIN(bank, num)   ((((bank) - 'A') << 8) | (num))
#define PINNO(pin)       ((pin) & 255)        // low byte
#define PINBANK(pin)     ((pin) >> 8)         // high byte

Example: PIN('C', 13) -> (('C'-'A')<<8) | 13 -> (2<<8) | 13 -> 0x020D
```

The subtraction (bank - 'A') converts ASCII letters 'A','B','C'... into 0,1,2...; the <<8 shift moves that bank index into the upper byte so the lower byte can hold the pin number.

7. Clearing and setting bits: &= vs |=

To change a 2-bit field inside a shared register, you typically do two steps: clear the old bits, then set the new value. This keeps all other pins' bits unchanged.

```
gpio->MODER &= ~(3U << (n * 2));      // clear the 2-bit field for pin n
gpio->MODER |=  (mode & 3) << (n * 2); // set the new 2-bit value

x &= y  means x = x & y (can force bits to 0)
x |= y  means x = x | y (can force bits to 1)
OR alone cannot turn 1-bits into 0-bits, so clearing first is required.
```

8. XOR loop example and why 14 ^ 90 = 84

The XOR accumulation example uses properties: x^x = 0, x^0 = x, and XOR is commutative/associative. Pairs cancel, leaving the value that occurs an odd number of times.

```
res starts at 0
res ^= 12 -> 12
res ^= 12 -> 0
res ^= 14 -> 14
res ^= 90 -> 84   (intermediate)
...
final result -> 90
```

Binary view of the intermediate step:

```
14  = 00001110
90  = 01011010
XOR = 01010100 = 64 + 16 + 4 = 84
```

Key takeaways

- Datasheet start address + size defines an address range; end = start + size - 1.
- Peripheral address ranges are memory-mapped: addresses point to hardware registers, not RAM.
- GPIO port registers pack per-pin configuration into shared 32-bit registers (MODER is 2 bits per pin).
- Struct overlays and small 'static inline' helpers make bare-metal code readable without hiding what's happening.
- Bitfield updates usually require 'clear then set' (&= ~mask, then |= value).