ARM-Embedded-Path

Bitmasking

Pavel Pys

October 30, 2025

Bit-Masking Basics

How does `((GPIOB->IDR & (1<<5)) == 0)` actually work?

Goal: We want to know: "Is Pin 5 LOW (0)?"

- We need to isolate Pin 5's state...
  - ...from 31 other (irrelevant) bits.
- We use "Bit-Masking" for this.

The 3 Steps:

1. `(1<<5)` $\rightarrow$ Create the "mask" (template).
2. `...  &  ...` $\rightarrow$ Apply the mask (Isolate).
3. `...  == 0` $\rightarrow$ Check the result.

Step 1: The "Mask" (1«5)

« is the "Bit-Shift Operator". (Shift bits left)

1. **Start with number 1:**

```
0000 0000 0000 0000 0000 0000 0000 0001
```

2. **Shift it 5 positions left:**

```
... 0000 0001  (Start)
... 0000 0010  (<< 1)
... 0000 0100  (<< 2)
... 0000 1000  (<< 3)
... 0001 0000  (<< 4)
... 0010 0000  (<< 5)  <- This is Pin 5!
```

Result (1«5) is our mask:

```
0000 0000 ... 0000 0010 0000  (Value: 32)
```

Step 2: Isolation (Bitwise-AND / &)

`&` applies our mask to the read value. Works like a "light filter".

Rule (Bit by bit):

- `1 & 1` $\rightarrow$ `1` (light passes through)
- `1 & 0` $\rightarrow$ `0`
- `0 & 1` $\rightarrow$ `0`
- `0 & 0` $\rightarrow$ `0`

What happens: Wherever our mask has **0**, the result is **ALWAYS 0** (regardless of `IDR` value). Only at the **single** position (Pin 5), where our mask has **1**, the actual value of Pin 5 "shines through"!

Step 2: Case A (Button NOT pressed)

Assumption: Pin 5 is HIGH (1).

```
... 0010 0100   (Example: GPIOB->IDR)
(Pin 5 is 1, Pin 2 is 1)
& 000 0010 0000   (Our mask (1<<5))
-----------------------------------------
= 000 0010 0000   (Result is 32, so NOT ZERO)
```

Explanation:

- At Pin 2: `IDR(1) & MASK(0) = 0` (filtered out)
- At Pin 5: `IDR(1) & MASK(1) = 1` (came through!)

The result is `(1«5)`.

Step 2: Case B (Button PRESSED)

Assumption: Pin 5 is LOW (0).

```
... 0000 0100   (Example: GPIOB->IDR)
(Pin 5 is 0, Pin 2 is 1)
& 000 0010 0000   (Our mask (1<<5))
-----------------------------------------
= 000 0000 0000   (Result is 0, so ZERO)
```

Explanation:

- At Pin 2: `IDR(1) & MASK(0) = 0` (filtered out)
- At Pin 5: `IDR(0) & MASK(1) = 0` (came through, but was 0!)

The result is `0`.

Step 3: Comparison `(... == 0)`

This part is now super simple. In step 2 we "erased" all 31 irrelevant bits.

We just ask: "Is the overall result of our isolation operation ZERO?"

```
if ( ((... & (1<<5)) == 0) )
```

- **If result is `0`:**
  - (Case B occurred)
  - The bit at Pin 5 *must* have been 0.
  - → **Button is pressed!**
- **If result is `not 0` (i.e., 32):**
  - (Case A occurred)
  - The bit at Pin 5 *must* have been 1.
  - → **Button is released!**

Bit-Masking: Conclusion

`((GPIOB->IDR & (1<<5)) == 0)`

- `(1<<5)` $\rightarrow$ "Build me a mask with a hole only at Pin 5."
- `&` $\rightarrow$ "Apply the mask to the register to isolate Pin 5."
- `== 0` $\rightarrow$ "Check if the isolated value (i.e., Pin 5) was zero."

What about `GPIO_IDR_IDR5`?

- That's just the CMSIS name for `(1<<5)`.
- `((GPIOB->IDR & GPIO_IDR_IDR5) == 0)` is exactly the same code.

The 4 Most Important Bit Operators

Your "Swiss Army knife" for registers.

- `&` **(AND)**: **The Isolator / Filter**
  - "Show me *only* the bits I care about."
  - `1 & 1 = 1`, `1 & 0 = 0`, `0 & 0 = 0`
  - (Used for *checking* in previous slides)
- `|` **(OR)**: **The Setter / Combiner**
  - "Set this bit to 1, but leave the rest alone."
  - `1 | 1 = 1`, `1 | 0 = 1`, `0 | 0 = 0`
- `~` **(NOT)**: **The Inverter**
  - "Flip *all* 32 bits."
  - `~1 = 0`, `~0 = 1`
- `^` **(XOR)**: **The Toggler**
  - "Flip *only* this one bit (0->1 or 1->0)."
  - `1 ^ 1 = 0`, `1 ^ 0 = 1`, `0 ^ 0 = 0`

The C Shortcut: |= and &=

These are "assignment operators".

Analogy: You know this from math:

- `x = x + 10;` is the same as `x += 10;`

For bit operations it's exactly the same:

- `x = x | MASK;` (long) $\rightarrow$ **`x |= MASK;`** (short)
- `x = x & MASK;` (long) $\rightarrow$ **`x &= MASK;`** (short)
- `x = x ^MASK;` (long) $\rightarrow$ **`x ^= MASK;`** (short)

Remember: |= and &= aren't "magic", just C syntax sugar (shortcuts) for a **"Read-Modify-Write" operation**.

Pattern 1: SET a Bit (with |=)

Goal: We want to turn Pin 5 (LED) ON (set to 1). But we must not change the other 31 bits!

Tool: | (OR)

- `...Original...` | 0 → `...Original...` (unchanged)
- `...Original...` | 1 → `1` (forced to 1)

The Operation:

```
// This is the "Set-Pin-5" command
GPIOB->ODR |= (1U<<5); // (or GPIO_ODR_ODR5)
```

```
... 0000 0100   (Original value in ODR register)
| ... 0010 0000   (Our mask (1<<5))
----------------------------------------
= ... 0010 0100   (Result: Pin 5 is ON, Pin 2 stayed ON)
```

Pattern 2: CLEAR a Bit (Concept)

Goal: We want to turn Pin 5 (LED) OFF (set to 0). But we must not change the other 31 bits!

Tools: ~ (NOT) and & (AND) This is the "pro" move. (You used this unconsciously in your `LED_Init` with `&= ~(0xF<<20)`!)

Step 1: The mask (Pin 5) $\rightarrow$ `(1<<5)`

```
... 0010 0000
```

Step 2: The "Anti-Mask" (Invert) $\rightarrow$ `~(1<<5)`

```
... 1101 1111  (All 1s, EXCEPT at Pin 5)
```

Pattern 2: CLEAR a Bit (Code &= ~)

Step 3: The Filtering (AND) The "anti-mask" is &-combined.

The Operation:

```
// This is the "Clear-Pin-5" command
REG &= ~(1U<<5);
```

```
... 0010 0100   (Original value in register)
& ... 1101 1111   (Our "anti-mask" from slide 11)
---------------------------------------
= ... 0000 0100 (Result: Pin 5 is OFF, Pin 2 stayed ON)
```

STM32F1: Atomic Set/Reset with `BSRR` (Recommended)

Why? Avoid read-modify-write hazards on `ODR` and update a single bit atomically.

Use `GPIOx_BSRR`:

- **Set bit** `y`: `GPIOB->BSRR = (1U << y);` (writes to `BSy`)
- **Reset bit** `y`: `GPIOB->BSRR = (1U << (y+16));` (writes to `BRy`)
- (On F1 there is also `BRR`: `GPIOB->BRR = (1U << y);` to reset.)

Note (RM0008): *For atomic bit set/reset, the ODR bits can be individually set and cleared by writing to the `GPIOx_BSRR` register.*

Pro Tip: TOGGLE a Bit (with ≙)

Goal: "Flip" Pin 5's state (ON->OFF, OFF->ON). Ideal for blinkers, without `if/else`!

Tool: ^ (XOR)

- `...Original...` $\hat{0} \rightarrow$ `...Original...` (unchanged)
- `...Original...` $\hat{1} \rightarrow$ `...Inverted...` (flipped)

The Operation:

```
// This is the "light switch" command
GPIOB->ODR ^= (1U<<5);
```

Pro Tip: TOGGLE (The 2 Cases)

Case A (LED was OFF):

```
... 0000 0000   (Original ODR)
^ ... 0010 0000   (Mask (1<<5))
--------------------------------------
= ... 0010 0000   (Result: LED is ON)
```

Case B (LED was ON):

```
... 0010 0000   (Original ODR)
^ ... 0010 0000   (Mask (1<<5))
--------------------------------------
= ... 0000 0000   (Result: LED is OFF)
```

Cheat Sheet: Bit Operations (Part 1)

Your toolbox for register programming

- **Isolate / Check (if LOW):**
  - `if ((REG & (1<<5)) == 0) ...`
- **SET Bit (to 1):**
  - `REG |= (1<<5);`

Cheat Sheet: Bit Operations (Part 2)

Your toolbox for register programming

- **CLEAR Bit (to 0):**
  - `REG &= ~(1U<<5);`
- **TOGGLE Bit (flip):**
  - `REG ^= (1U<<5);`
- **CLEAR 4-Bit Block (e.g., Pin 13):**
  - `REG &= ~(0xFU << 20);`