GPIO

1. What is GPIO?

GPIO (General Purpose Input/Output) is used for controlling and reading digital signals.

Each GPIO pin can be configured as:

Input Mode (to read signals from sensors, buttons, etc.).

Output Mode (to control LEDs, motors, etc.).

Alternate Function Mode (for peripherals like UART, SPI, I2C).

Analog Mode (for ADC readings).

2. GPIO in Embedded Systems

The "Hello World" program of embedded systems is Blinky (LED Blinking).

GPIO is not limited to LEDs; it is also used for:

Reading digital signals (buttons, sensors).

Generating triggers for external components (timers, interrupts).

Issuing interrupts (external event detection).

Waking up the processor from low-power mode.

3. What is a GPIO Port?

A GPIO Port is a collection of GPIO pins.

Each GPIO port has multiple pins, usually 8, 16, or 32 pins.

GPIO Port A → 16 Pins (PA0 - PA15).

GPIO Port B → 16 Pins (PB0 - PB15).

Each GPIO pin corresponds to a specific bit in the GPIO register.

4. GPIO Pin and Port Width in Different Microcontrollers

Different microcontrollers support different GPIO port sizes.

ARM Cortex M4 microcontrollers use 16-bit wide GPIO ports.

GPIO pins are fundamental to embedded programming.

A GPIO port is a group of GPIO pins (e.g., GPIOA = PA0 - PA15).

GPIOs can be used for Input, Output, Interrupts, and Wake-up functions.

Different microcontrollers have different GPIO port sizes (8, 16, 32 bits).

5. How is a GPIO Pin Implemented Internally?

A GPIO pin is connected to two buffers inside the microcontroller:

Output Buffer → Controls the pin when configured as an output.

Input Buffer → Reads the pin when configured as an input.

Enable Line determines whether the pin is in input or output mode.

6. Understanding Buffers (Using CMOS Transistors)

Each GPIO pin has two transistors:

PMOS (T1) → Pulls the pin HIGH (Logic 1).

NMOS (T2) → Pulls the pin LOW (Logic 0).

The working principle:

Writing '1' to the GPIO Pin:

T1 (PMOS) turns ON, T2 (NMOS) turns OFF → Pin goes HIGH.

Writing '0' to the GPIO Pin:

T2 (NMOS) turns ON, T1 (PMOS) turns OFF → Pin goes LOW.

How the Output Buffer Works:

7. How GPIO Works in Input Mode

When a GPIO pin is configured as input, the input buffer is enabled and the output buffer is disabled.

How Input Mode Works:

If an external voltage drives the pin HIGH → T1 turns ON, T2 turns OFF → Reads HIGH (1).

If an external voltage drives the pin LOW → T1 turns OFF, T2 turns ON → Reads LOW (0).

8. How is GPIO Mode Controlled?

The Enable Line decides whether the pin is in input or output mode.

Enable Line = 0 → Output Mode (Activates Output Buffer, Deactivates Input Buffer).

Enable Line = 1 → Input Mode (Activates Input Buffer, Deactivates Output Buffer).

This configuration is controlled via GPIO control registers.

A GPIO pin has two buffers: Output Buffer and Input Buffer.

Output Buffer uses CMOS transistors (PMOS + NMOS) to set HIGH (1) or LOW (0).

Input Buffer works like an inverted Output Buffer and reads the voltage on the pin.

The Enable Line controls whether a GPIO pin is in Input Mode or Output Mode.

Microcontroller registers configure and control the Enable Line to switch modes.

Important Points to Remember from GPIO Input Mode with High Impedance State

9. What is High Impedance (HI-Z) State?

High Impedance (HI-Z) State means the GPIO pin is not connected to either VDD (high) or GND (low).

This is also called a Floating State because the pin is left disconnected.

By default, all GPIO pins are in HI-Z state after microcontroller power-up.

10. Why is High Impedance Used?

HI-Z prevents unwanted current flow when a pin is not actively driving a signal.

It is useful when multiple devices share the same GPIO pin (e.g., SPI communication, I2C bus).

Prevents short circuits when multiple output devices connect to the same pin.

11. Issues with High Impedance (Floating Pins)

A floating pin can pick up noise, causing unpredictable behavior.

It may lead to leakage currents, increasing power consumption.

If a pin is not pulled to a defined state, it may randomly toggle between HIGH and LOW.

12. How to Avoid Floating Pin Issues?

To prevent floating inputs, we use pull-up or pull-down resistors:

Pull-up resistor (Internal or External)

Connects GPIO pin to VDD through a resistor.

Ensures the pin stays HIGH when not actively driven LOW.

Pull-down resistor (Internal or External)

Connects GPIO pin to GND through a resistor.

Ensures the pin stays LOW when not actively driven HIGH.

HI-Z (Floating) State means a pin is not actively connected to HIGH (VDD) or LOW (GND).

By default, GPIO pins start in HI-Z mode after power-up.

Floating pins can cause noise, unpredictable behavior, and leakage currents.

Pull-up or Pull-down resistors ensure a defined HIGH or LOW state.

HI-Z is useful when multiple devices share a GPIO pin, but it must be managed correctly.

Important Points to Remember from GPIO Input Mode with Pull-up/Pull-down Resistors

13. What is Open-Drain Output Mode?

In open-drain mode, the PMOS transistor is removed, leaving only an NMOS transistor.

This means the GPIO pin can only:

Pull down (LOW) to GND (when NMOS is ON).

Float (disconnected) when NMOS is OFF.

It cannot actively pull HIGH (VDD) without a pull-up resistor!

Open-drain mode is useless without a pull-up resistor.

14. Why Do We Need a Pull-Up Resistor?

Since open-drain cannot pull HIGH, we need a pull-up resistor to provide a HIGH state.

The pull-up resistor can be:

Internal Pull-Up (Enabled via GPIO configuration register).

External Pull-Up (A physical resistor connected between the GPIO pin and VDD).

15. Applications of Open-Drain Mode

(A) Driving an LED in Open-Drain Mode

LED connected to GPIO with pull-up resistor.

Turning ON LED:

Write 1 → NMOS turns OFF → Pull-up resistor pulls HIGH → LED turns ON.

Turning OFF LED:

Write 0 → NMOS turns ON → Pin pulled to GND → LED turns OFF.

Internal pull-up (10kΩ–50kΩ) or external pull-up can be used.

(B) I2C Communication (SDA & SCL Lines)

I2C bus uses open-drain configuration.

SDA (Serial Data) and SCL (Serial Clock) lines need pull-up resistors.

Pull-ups (4.7kΩ – 10kΩ) ensure proper HIGH logic level.

Can use internal pull-ups or external resistors.

Open-drain mode removes the PMOS transistor, leaving only NMOS.

It can only pull LOW but cannot actively drive HIGH without a pull-up resistor.

Pull-up resistors (internal or external) are necessary for open-drain mode.

Used in LED control, I2C communication, and multi-device bus systems.

Internal pull-ups (10kΩ – 50kΩ) are available, but external pull-ups (4.7kΩ – 10kΩ) are common for I2C.

16. What is Push-Pull Output Mode?

Push-pull mode uses two transistors (PMOS & NMOS) to actively drive the pin HIGH and LOW.

Unlike open-drain mode, push-pull does not require pull-up resistors.

This configuration is called push-pull because:

The top transistor (PMOS) pulls the pin HIGH (VDD).

The bottom transistor (NMOS) pulls the pin LOW (GND).

It ensures a strong HIGH and LOW output without floating states.

17. What is a Floating Input Pin?

A floating input pin means it is not connected to either HIGH (VCC) or LOW (GND).

Due to random noise and interference, the voltage at the pin fluctuates unpredictably.

This may result in leakage current flowing from VCC to GND inside the microcontroller.

18. Why Does Leakage Current Occur in Floating Inputs?

The input buffer inside a microcontroller consists of two transistors (T1 & T2):

T1 (PMOS) turns ON when the input is LOW.

T2 (NMOS) turns ON when the input is HIGH.

When the input pin is floating, the voltage may settle at an intermediate value (~30%-70% of VCC).

At this voltage, both transistors are partially ON, creating a leakage path from VCC to GND.

19. How to Prevent Leakage Current?

✅ Use Pull-up or Pull-down Resistors

Pull-up resistor keeps the pin HIGH (VCC).

Pull-down resistor keeps the pin LOW (GND).

This prevents intermediate voltage and leakage.

✅ Enable Internal Pull-up/Pull-down Resistors via GPIO Configuration Registers

Most modern MCUs have built-in pull-ups and pull-downs.

✅ Use Schmitt Trigger Input Buffers

Modern microcontrollers use Schmitt triggers to filter noise and stabilize input voltage.

This helps in reducing false triggering and leakage issues.

20. How to Enable Pull-up/Pull-down in ST Microcontroller?

In ST Microcontroller, the GPIO Pull-up/Pull-down Register (PUPDR) is used.

Example: Enable Pull-up for GPIOA Pin 5

```
#define GPIOA_PUPDR  (*(volatile uint32_t*) 0x4002000C)//GPIOA Pull-up/Pull-down Register

void enable_pullup_gpioa_pin5()

{

GPIOA_PUPDR &= ~(0x3 << 10); // Clear bits 11:10

GPIOA_PUPDR |=  (0x1 << 10); // Set Pull-up (01)

}
```

21. How Push-Pull Mode Works?

Case 1: Writing 1 to Output Data Register (ODR)

Step 1: 1 is written to the Output Data Register (ODR).

Step 2: The NOT gate inverts the signal, producing 0.

Step 3: PMOS transistor turns ON, while NMOS transistor turns OFF.

Step 4: The pin is pulled HIGH (VCC).

Step 5: Current flows out of the GPIO pin → Sourcing Current.

22. How Open Drain Mode Works?

Case 1: Writing 0 to Output Data Register (ODR)

Step 1: 0 is written to Output Data Register (ODR).

Step 3: NMOS transistor turns ON, creating a direct path to GND.

Step 4: The pin is pulled LOW (0V/GND).

➡️ If an LED is connected, it will turn ON.

Case 2: Writing 1 to Output Data Register (ODR)

Step 1: 1 is written to Output Data Register (ODR).

Step 3: NMOS transistor turns OFF.

Step 4: The pin is left floating (HIGH-Z state).

➡️ If no pull-up resistor is present, the pin will float (undefined voltage).

23. How to Configure GPIO for Alternate Function Mode?

Step 1: Configure GPIO Mode Register (MODER)

Set MODER bits to 10 for Alternate Function Mode.

```
GPIOA->MODER |= (2 << (2 * PIN_NUMBER));  // Set mode to AF
```

Step 2: Select Alternate Function in AFR Register

The AFR register (GPIO Alternate Function Register) maps each pin to a specific function.

Each pin supports multiple alternate functions (AF0 to AF15).

Example: Selecting USART1 TX (AF7) on GPIOA Pin 9

```
GPIOA->AFR[1] |= (7 << ((9 - 8) * 4)); // Select AF7 (USART1_TX)
```

Step 3: Configure Output Type & Speed (Optional)

Choose Push-Pull or Open-Drain Output Type in OTYPER.

Set Output Speed (OSPEEDR) to match the peripheral requirements.

```
GPIOA->OTYPER &= ~(1 << 9); // Push-Pull Mode (default)
GPIOA->OSPEEDR |= (2 << (9 * 2)); // High Speed Output
```

24. Alternate Function Mapping in STM32

Each GPIO pin can have multiple alternate functions.

For example, PA9 can be:

AF1 → TIM1_CH2

AF3 → I2C3_SMBA

AF7 → USART1_TX

AF9 → CAN1_TX

Example: Configuring PA9 for USART1 TX

Scenario: Configure PA9 as USART1 TX using AF7.

```
// Enable Clock for GPIOA and USART1
RCC->AHB1ENR |= (1 << 0);  // GPIOA clock enable
RCC->APB2ENR |= (1 << 4);  // USART1 clock enable
// Set PA9 as Alternate Function Mode
GPIOA->MODER &= ~(3 << (9 * 2)); // Clear bits
GPIOA->MODER |= (2 << (9 * 2));  // Set to Alternate Function
// Set PA9 to AF7 (USART1_TX)
```

```
GPIOA->AFR[1] &= ~(0xF << ((9 - 8) * 4)); // Clear bits
GPIOA->AFR[1] |= (7 << ((9 - 8) * 4));   // Select AF7
// Configure PA9 Output Type and Speed
GPIOA->OTYPER &= ~(1 << 9);  // Push-Pull
GPIOA->OSPEEDR |= (3 << (9 * 2)); // Very High Speed
```

25. What is the Purpose of OSPEEDR Register?

The GPIO Port Output Speed Register (OSPEEDR) is used to control the slew rate (switching speed) of a GPIO pin when it is configured as an output.

- The OSPEEDR register determines:

How fast the GPIO pin transitions from HIGH to LOW and vice versa.

The maximum frequency at which the GPIO can toggle.

The drive strength and electromagnetic interference (EMI) characteristics of the GPIO.

4. How to Write to GPIO Pins Using ODR?

```
// Set PA5 as Output Mode
GPIOA->MODER &= ~(3 << (5 * 2)); // Clear mode bits
GPIOA->MODER |= (1 << (5 * 2));  // Set Output mode (01)
// Turn ON LED (PA5 HIGH)
GPIOA->ODR |= (1 << 5);
// Turn OFF LED (PA5 LOW)
GPIOA->ODR &= ~(1 << 5);
```

5. Reading GPIO Output States Using ODR

To check the state of all 16 GPIO pins:

```
uint16_t pin_states = GPIOA->ODR;
```

To check if PA5 is HIGH:

```
if (GPIOA->ODR & (1 << 5))
        // PA5 is HIGH
```

5. How to Configure Alternate Functionality?

Let's assume we want to configure PA8 as TIM1_CH1 (AF1).

```
GPIOA->MODER &= ~(3 << (8 * 2)); // Clear bits

GPIOA->MODER |= (2 << (8 * 2));  // Set to Alternate Function Mode (10)

// Select AF1 (TIM1_CH1) in AFRH Register

GPIOA->AFR[1] &= ~(0xF << ((8 - 8) * 4)); // Clear existing bits

GPIOA->AFR[1] |= (1 << ((8 - 8) * 4));    // Set AF1
```