**Edison Yang**

**U1298759**

**ECE 5780 - Post Lab1 Questions**

**Link to my Github Repo: [https://github.com/EYANG27/ECE5780](https://github.com/EYANG27/ECE5780)**

**1. What are the GPIO control registers that the lab mentions? Briefly describe each of their functions.**

1. GPIOx_MODER (Mode Register): This register is crucial in defining the operational mode of individual GPIO pins within a given port 'x'. It allocates two bits per pin to delineate among various modes such as digital input, digital output, alternate function (interfacing with onboard peripherals), and analog mode. The binary values assigned to these bits directly influence the pin's behavior and its interaction with external circuitry.

2. GPIOx_OTYPER (Output Type Register): In the context of output configuration, this register specifies the output signal's drive characteristics. It distinguishes between 'push-pull' (a bidirectional drive capable of asserting both high and low states actively) and 'open-drain' (unidirectional drive where the pin can either pull the line to ground or leave it in a high-impedance state). Each pin's output type is controlled by a single bit, facilitating the configuration of heterogeneous output schemes within the same GPIO port.

3. GPIOx_OSPEEDR (Output Speed Register): The electrical characteristics of digital signals are partly defined by their transition speeds. This register, through per-pin configuration, controls the slew rate of GPIO pins configured as outputs. The granularity provided by two bits per pin allows for the selection of speed categories ranging from low to very high, directly impacting the signal rise and fall times, and by extension, the electromagnetic characteristics and power consumption.

4. GPIOx_PUPDR (Pull-Up/Pull-Down Register): To mitigate issues related to floating inputs and ensure defined logic levels, this register enables the activation of internal pull-up or pull-down resistors. Configuration options per pin include the disabling of pull resistors, enabling pull-up, enabling pull-down, or a reserved state, thereby ensuring signal integrity in various electrical environments.

5. GPIOx_IDR (Input Data Register): Real-time monitoring of GPIO pin states is facilitated through this register. It provides a direct readout of the logic levels present at the pins configured as inputs, thus serving as a critical interface for sensing external digital events or states.

6. GPIOx_ODR (Output Data Register): Analogous to the IDR but for output operations, this register holds the logic levels to be driven on the pins configured as outputs. Writing to this register enacts a change in the output state, enabling the microcontroller to interact with external components such as LEDs, motors, and other digital systems.

7. GPIOx_BSRR (Bit Set/Reset Register): This register is engineered for atomic bit manipulation, allowing for the setting or clearing of output states without the risk of read-modify-write errors. It is bifurcated into a 'set' and 'reset' section, each affecting the corresponding pin state when written with a '1', thereby providing a mechanism for high-reliability state changes.

8. GPIOx_LCKR (Configuration Lock Register): To safeguard against inadvertent modifications of GPIO configurations, this register implements a locking mechanism. Once engaged, the configuration of the GPIO pins is frozen until a system reset, ensuring the stability of critical hardware interfaces.

9. GPIOx_BRR (Bit Reset Register): This register complements the BSRR by offering a dedicated pathway for resetting pin states. A write operation with a '1' to the appropriate bit position will clear the corresponding pin's output, enhancing the control granularity.

**2. What values would you want to write to the bits controlling a pin in the GPIOx_MODER register in order to set it to analog mode?**

The values you would want to write to the bits that control a pin in the GPIOx_MODER register to set it to analog mode would be writing “11”, in binary, to the two bits corresponding to that pin.

**3. Examine the bit descriptions in the GPIOx_BSRR register: which bit would you want to set to clear the fourth bit in the ODR?**

You would want to set the **20th** bit to clear the fourth bit in the ODR, this is because the 20th bit corresponds to the fourth bit in the ODR when it comes to clearing a bit. Setting the 20th bit of the BSRR to 1 will clear the fourth bit of the ODR. The GPIOx_BSRR (Bit Set/Reset Register) is used to set or clear bits in the GPIO Output Data Register (ODR) atomically, without affecting other bits. The BSRR is typically a 32-bit register, where the lower 16 bits (0-15) are used to set the corresponding bits in the ODR (writing a 1 sets the bit in the ODR to 1), and the upper 16 bits (16-31) are used to clear the bits in the ODR (writing a 1 to these bits clears the corresponding bit in the ODR to 0).

**4. Perform the following bitwise operations: • 0xAD | 0xC7 = ? • 0xAD & 0xC7 = ? • 0xAD & ~(0xC7) = ? • 0xAD ^0xC7 = ?**

0xAD | 0xC7 = 0xEF, which is 239 in decimal
0xAD & 0xC7 = 0x85, which is 133 in decimal
0xAD & ~(0xC7) = 0x28, which is 40 in decimal
0xAD ^ 0xC7 = 0x6A, which is 106 in decimal

**5. How would you clear the 5th and 6th bits in a register while leaving the other's alone? 38 Memory-Mapped Peripherals and the GPIO**

To clear the 5th and 6th bits in a register while leaving the others unchanged, you can use a bitwise AND operation with a mask that has all bits set to 1 except for the 5th and 6th bits, which should be set to 0. In bit numbering, if we start counting from 0 (i.e., the least significant bit is bit 0), the 5th and 6th bits are actually the bits in positions 5 and 6.

**6. What is the maximum speed the STM32F072R8 GPIO pins can handle in the lowest speed setting? • Use the chip datasheet: lab section 1.4.1 gives a hint to the location. You'll want to search the I/O AC characteristics table. You will also need to view the OSPEEDR settings to find the bit pattern indicating the slowest speed.**

The maximum speed the STM32F072R8 can handle in its lowest speed setting is 1 MHz.

**7. What RCC register would you manipulate to enable the following peripherals: (use the comments next to the bit defines for better peripheral descriptions) • TIM1 (TIMER1) • DMA1 • I2C1**

We will need to use the RCC_AHBENR (for DMA1) and RCC_APB2ENR (for TIM1 and I2C1), according to the stm32f0xx.h file..