**GPIO** General Purpose Input/output (GPIO) is an interface available on most modern microcontrollers (MCU) to provide an ease of access to the devices internal properties.   
 Generally there are multiple GPIO pins on a single MCU for the use of multiple interaction so simultaneous application. The pins can be programmed as input, where data from some external source is being fed into the system to be manipulated at a desired time and location. Output can also be performed on GPIOs, where formatted data can be transmitted efficiently to outside devices, this provides a simple mechanism to program and retransmit data depending on user desires through a single port interface.

In many applications, the GPIOs can be configured as interrupt lines for a CPU to signal immediate processing of input lines. In many newer designs, they also have the ability to control and use Direct Memory Access (DMA) to transfer blocks of data in a more efficient manner. Essentially all ports can be tailored to fit specific design goals and provide reusability within applications.

The STM32 microcontroller board has 8 general purpose input/output (GPIO) ports named Port A, B, C, D, E, F, G and H. Each port can have up to 16 pins, and each port has associated with 32-bit registers.  
  
**Configuration Register**

* GPIOx\_MODER
* GPIOx\_OTYPER
* GPIOx\_OSPEEDR
* GPIOx\_PUPDR

**Data Register**

* GPIOx\_IDR
* GPIOx\_ODR

**Locking Register**

* GPIOx\_LCKR

**Alternate Function Selection Register**

* GPIOx\_AFRH
* GPIOx\_AFRL

**Set/Reset Register**

* GPIOx\_BSRR

**GPIO port mode register (GPIOx\_MODER)**

This is a 32-bit register where each set of two consecutive bits represent the mode of a single I/O pin. For example bits 0 and 1 of the MODER register associated with GPIOC (GPIOC\_MODER), represent the mode of GPIO pin PC0 and bits 26 and 27 of the same register represent the mode of GPIO pin PC13. These two bits can be set to:

* '00'-> input mode, which allows the GPIO pin to be used as an input pin,
* '01'-> Output mode, which allows the GPIO pin to be used as an output pin,
* '11'-> Analog mode, which allows the GPIO pin to be used as an Analog input pin and finally,
* '10'-> Alternate function mode which allow the GPIO pins to be used by peripherals such as the UART, SPI e.t.c. It is important to note that if a pin's MODE is set to alternate function, any GPIO settings for that pin in the GPIO registers will be overridden by the peripheral. I will be addressing Alternate function mode in more detail in a future entry.

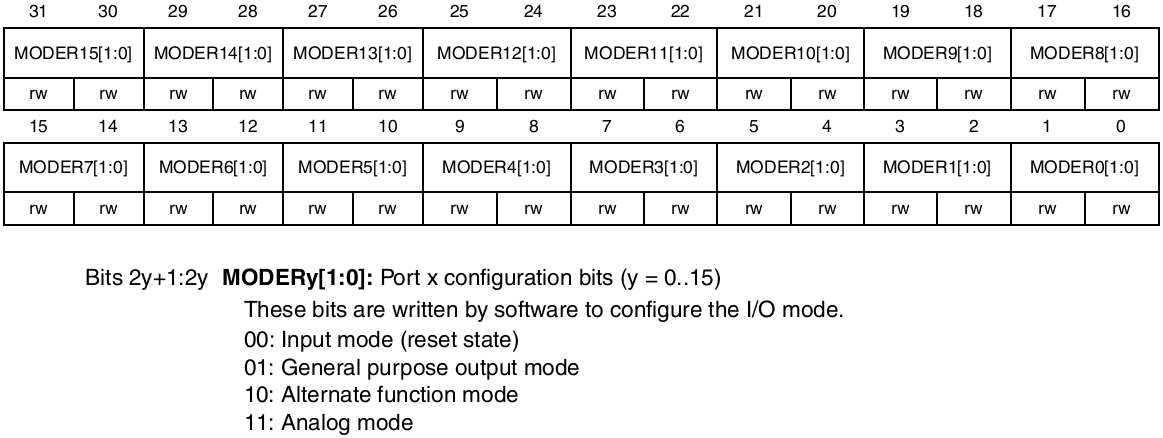


Figure 1 - GPIOx\_MODER register

**GPIO port output type register (GPIOx\_OTYPER)**

This is a 16-bit register where each bit denotes the 'type' of a single pin in the register. This register sets the type of output pins to either push-pull or open drain. For example if pin PC7 is configured as an output pin, clearing bit 7 (or leaving its state at zero) of the OTYPER register associated with GPIOC (GPIOC\_TYPER), will set the output type of the GPIO output pin PC7 to "Push-Pull".

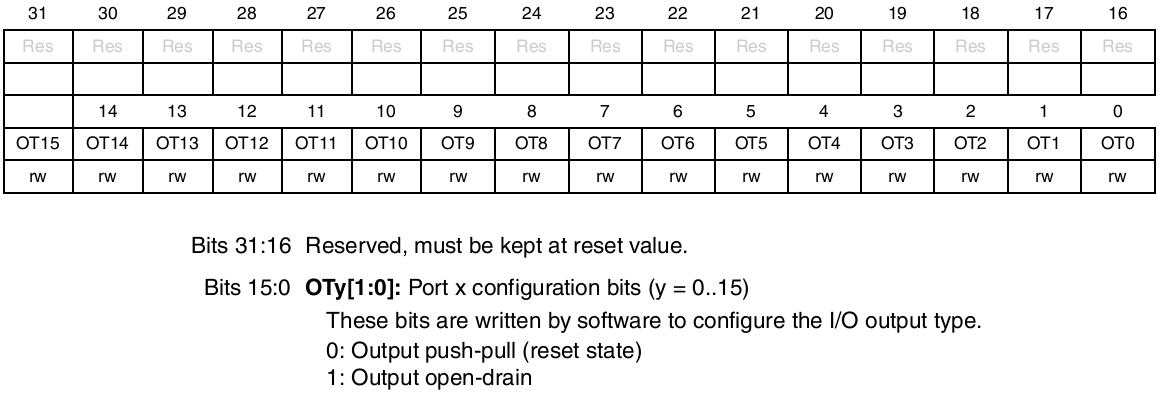


Figure 2. GPIOx\_OTYPER Register

**GPIO port output speed register (GPIOx\_OSPEEDR)**

This is a 32-bit register where each set of two bits represent the speed of a single output pin. For example bits 0 and 1 of the OSPEEDR register associated with port C (GPIOC\_OSPEEDR), represent the speed setting of the output pin PC0 and bits 26 and 27 of the same register represent the speed setting of the output pin PC13. These two bits can be set to:

* 'x0': 2 MHz Low speed
* '01':10 MHz Medium speed
* '11': 50 MHz High speed

So why have a speed setting on I/O ? To save power. On the 2MHz setting the GPIO would consume less current than on the 50 MHz setting I'd imagine but would have relatively longer rise/fall time specs.

The User Manual for the STM32F0 claims that the the output pins fastest toggle speed is every two clock cycles. Assuming a maximum operation speed of 48MHz, the fastest toggle speed for the GPIO on the STM32f0 is 24 MHZ, which means the highest frequency square wave that can be produced by the GPIO is 12MHz.

NOTE: A quick look at the default startup code found in"system\_stm32f0xx.c" will identify that the microcontroller is indeed operating at a maximum speed of 48MHz at startup. In a future entry I will demonstrate how this speed can be changed.

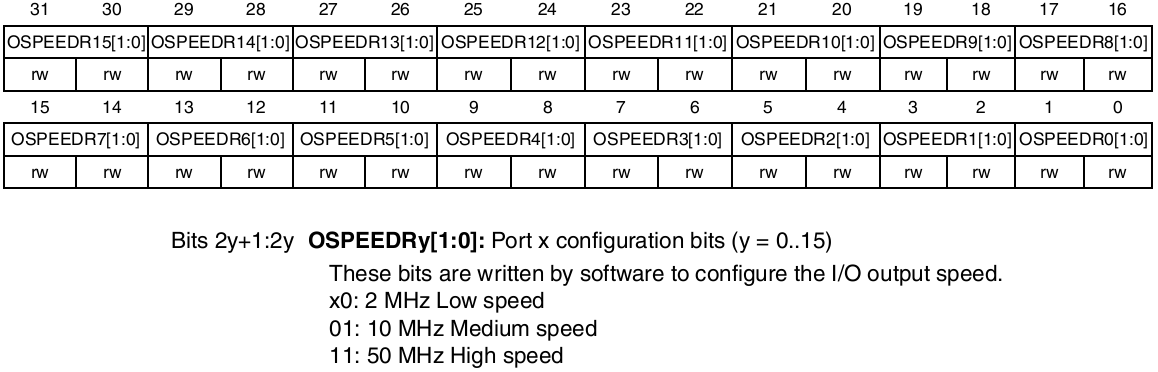


Figure 3. GPIOx\_OSPEEDR register

**GPIO port pull-up/pull-down register(GPIOx\_PUPDR)**

The GPIOx\_PUPDR registers configures the internal pull-ups and pull-down resistors on each I/O pin. The internal pull-up/down resistors can be configured on GPIO pins set as input or output (though I'd imagine they'd be more popular on input pins). The Pull-up/down resistors have a typical value of 40 KOhms but can range from 30-50 KOhms.

Again each two consecutive bits represent the internal pull-up/down resistor setting for each pin within a single port.

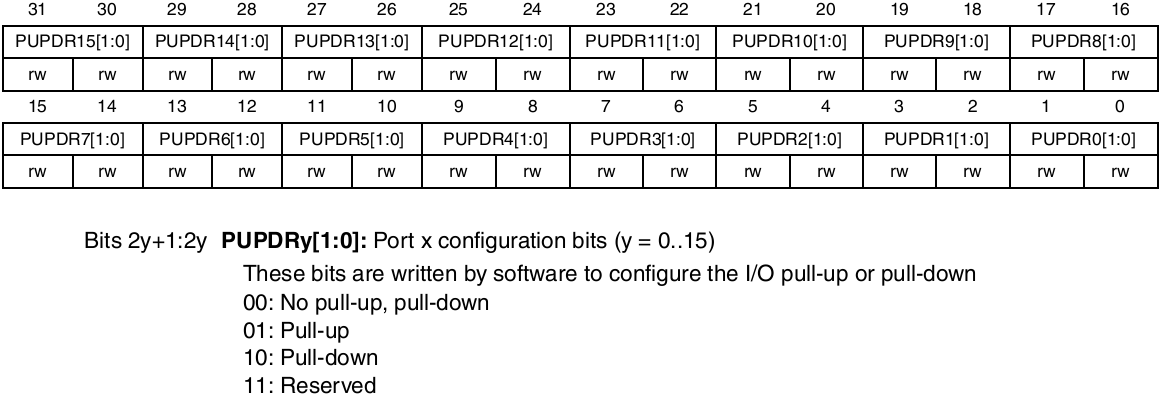
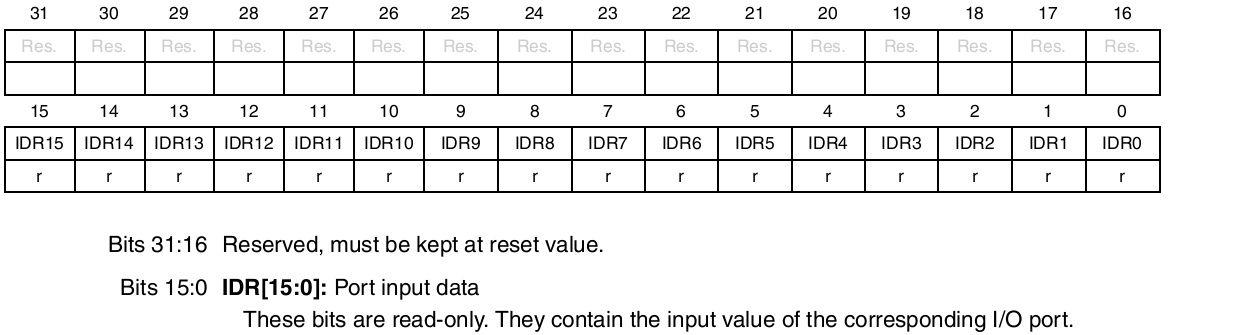


Figure 4. GPIOx\_PUPDR Register

**GPIO port input data register (GPIOx \_IDR)**

This is a 16-bit read-only register. Each bit represents the input value on a corresponding pin. Reading a '0' in bit 8 of this GPIOC \_IDR register indicates that the voltage on PC8 is 0V (GND). While reading a '1' in bit 8 of this GPIOC \_IDR register indicates that the voltage on PC8 is 3.3V (VDD)

Figure 5. GPIOx\_IDR Register

**GPIO port output data register (GPIOx\_ODR)**

This is a 16-bit read/write register. Each bit represents the output value on a corresponding pin. Writing a '0' in bit 8 of this GPIOC \_ODR register indicates that the voltage on PC8 is driven by the micro to 0V (GND). While writing a '1' in bit 8 of this GPIOC \_ODR register indicates that the voltage on PC8 is driven by the micro to 3.3V (VDD).

Writing to the ODR register is good if you want to write to the entire port.e.g.

GPIOC->ODR = 0xF0FE

The above statement changes the state of every pin on the GPIOC peripheral from its previous (and now discarded) state, to the one indicated by the statement; 0xF0FE (0b1111000011111110).

However if you want to set only a single pin; lets say PC8 without affecting the state of the rest of the pins on GPIOC, you have to perform a read-modify-write (RMW) access. To set pin PC8 independent of all other pins on GPIOC (RMW) you could use:

GPIOC->ODR |= 0x00000100; *//( 0b00000000000000000000000100000000)*

To clear pin PC8 independent of all other pins on GPIOC (RMW) you could use:

GPIOC->ODR &= ~(0x00000100); *//( 0b00000000000000000000000100000000)*

This works just fine, but you have to **read** the ODR register, OR (|) or AND(&) **(modify)** it with a mask and then **write** it back to the ODR register. This means that at the assembly language level, at least three instructions are used to set/clear an I/O which can significantly slow down toggling speed. A better way would be to use the BSRR register and the BRR registers for setting and clearing pins. They enable 'atomic' access that allows the I/O pin to be clear/set in as short a time as possible.

**GPIO port bit set/reset register (GPIOx \_BSRR)**

As mentioned in the preceding paragraph the BSRR register allows us to set/clear a particular pin (or groups of pins) while preserving the state of the rest of the pins on a GPIO peripheral atomically i.e. a fast as possible, without having to resort to the slower read-modify-write (RMW) accesses. The least significant 16 bits are used to atomically set pin values to VDD whereas the most significant 16 bits are used to atomically clear pin values to GND.

So if I wanted to set PC8 independent of all other pins on GPIOC I could use:

GPIOC->BSRR = 0x00000100;*//( 0b00000000000000000000000100000000)*

or

GPIOC->BSRR = (1<<8);

To clear pin PC8 independent of all other pins on GPIOC you could use:

GPIOC->BSRR = 0x01000000; *//( 0b00000001000000000000000000000000)*

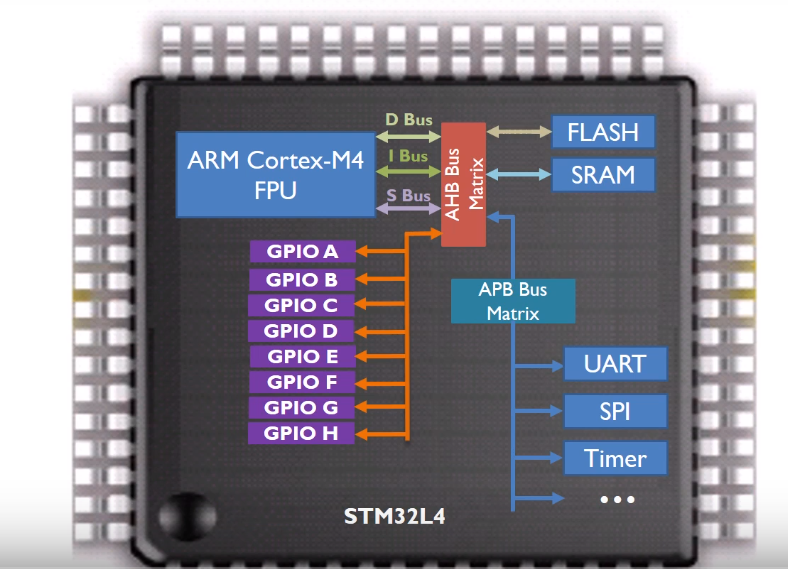
or

GPIOC->BSRR = (1<<24);

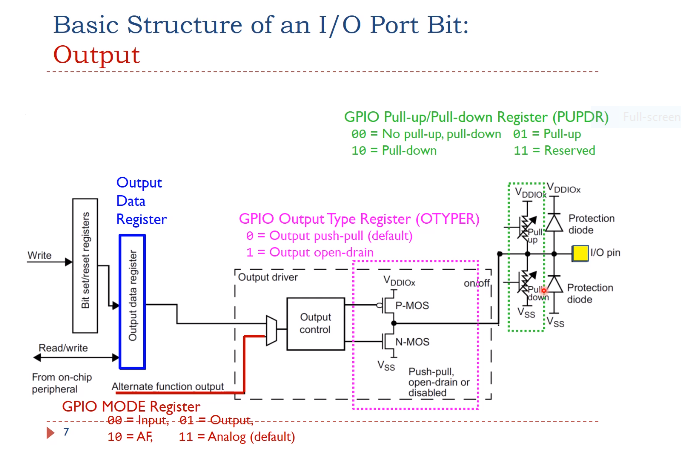
Notice how in both scenarios a simple assignment operator '=' (atomic) was used rather than an '|=' or an '&=' which denote RMW accesses. Furthermore, note that to clear the pin value of PC8 to GND, I had to set the 24th bit in the BSRR (8th bit of the most significant 16 bits). While to set the pin value of PC8 to VDD, I had to set the 8th bit in the BSRR. The awkwardness of atomic clearing being mapped to the most significant 16-bits of the BSRR register is compensated for by the inclusion of the BRR register. The BRR register maps the most significant 16-bits of the BSRR register into itself. So to clear pin PC8 independent of all other pins on GPIOC you could use:

GPIOC->BRR = (1<<8);

Finally there are three more registers; the GPIOx\_AFRH, GPIOx\_AFRL, and the GPIOx\_LCKR registers. The first two allow GPIO pins to be used for alternate functions. There is a really neat pin muxing mechanism that allows each GPIO to be mapped to multiple alternate functions depending on how these two registers are set. I will spend more time on the AFRL/AFRH registers in future entries. The last GPIOx\_LCKR register can be used once GPIO is configured to 'lock' the configuration so that it does not change until the micro is reset. I encourage you to look up these three registers in the user manual.

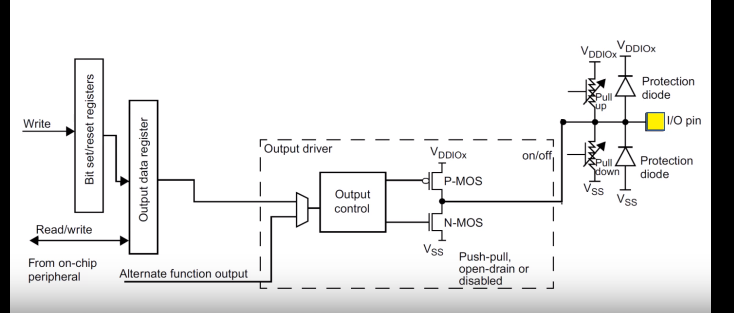


Configuration of a pin to general purpose output mode  
  
 Previously talked about all the registers of a GPIO. Now, time to tell about the configuration of a pin to GP(general purpose) output mode.   
 For output mode of a port, we associate with 4 registers: GPIOx\_MODER, GPIOx\_OTYPER, GPIOx\_PUPDR, GPIOx\_ODR, GPIOx\_BSRR.

  
Each GPIO port can have up to 16 pins. All pins in a GPIO port are controlled as a group. Each pin can be configured by software at runtime to perform various functions

Here is the basic structure that is designed

for digital output on one GPIO pin.



We can use the mode register to configure the pin as output. The output type register sets the output pin as, either push-pull or open-drain. The pull-up pull-down register specifies whether the pin is pulled up to a high voltage by a register, or the pin is pulled down to the ground via another register. When the pin is configured as output, the value written to the output data register is output on the GPIO pin. However, whether the pin has a high voltage output, a low voltage output, or a high impedance output, depends on the configuration of the output type register, and the pull-up pull-down register.