Introduction

Every STM32 microcontroller has a variable number of general programmable I/Os, depending on package chosen, family of microcontroller and usage of external crystals.

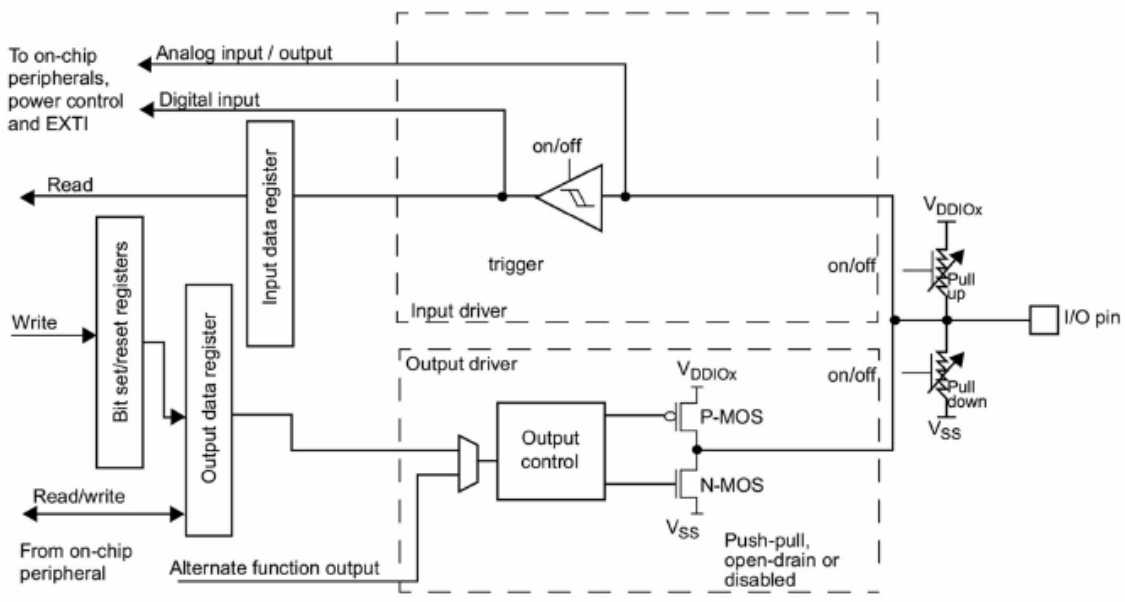

Figure 4: Basic structure of an I/O port bit

GPIO configuration

To configure a GPIO we use the `HAL_GPIO_Init(GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_Init)` function. GPIO_InitTypeDef is the C struct used to configure the GPIO, and it is defined in the following way:

```
typedef struct {
    uint32_t Pin;
    uint32_t Mode;
    uint32_t Pull;
    uint32_t Speed;
    uint32_t Alternate;
} GPIO_InitTypeDef;
```

where:

- Pin: it is the number, starting from 0, of the pins we are going to configure. For example, for PA5 pin it assumes the value GPIO_PIN_55 . We can use the same GPIO_InitTypeDef instance to configure several pins at once, doing a bitwise OR (e.g., GPIO_PIN_1 | GPIO_PIN_5 | GPIO_- PIN_6).
- Mode: it is the operating mode of the pin, and it can assume one of the values in the table Pin modes.
- Pull: specifies the Pull-up or Pull-Down activation for the selected pins, according to table Pull modes.
- Speed: defines the pin speed.
- Alternate: specifies which peripheral to associate to the pin.

GPIO Modes

Table 1: Pin modes

Mode	Description
GPIO_MODE_INPUT	Input Floating Mode 6
GPIO_MODE_OUTPUT_PP	Output Push Pull Mode
GPIO_MODE_OUTPUT_OD	Output Open Drain Mode
GPIO_MODE_AF_PP	Alternate Function Push Pull Mode
GPIO_MODE_AF_OD	Alternate Function Open Drain Mode
GPIO_MODE_ANALOG	Analog Mode
GPIO_MODE_IT_RISING	External Interrupt Mode with Rising edge trigger detection
GPIO_MODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
GPIO_MODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection
GPIO_MODE_EVT_RISING	External Event Mode with Rising edge trigger detection
GPIO_MODE_EVT_FALLING	External Event Mode with Falling edge trigger detection
GPIO_MODE_EVT_RISING_FALLING	External Event Mode with Rising/Falling edge trigger detection

GPIO Pull

Table 2: Pull modes

Pull mode	Description
GPIO_NOPULL	No Pull-up or Pull-down activation
GPIO_PULLUP	Pull-up activation
GPIO_PULLDOWN	Pull-down activation

GPIO Speed

Speed mode	Description
GPIO_SPEED_FREQ_LOW	range up to 5 MHz, refer to the product datasheet
GPIO_SPEED_FREQ_MEDIUM	range 5 MHz to 25 MHz, refer to the product datasheet
GPIO_SPEED_FREQ_HIGH	range 25 MHz to 50 MHz, refer to the product datasheet
GPIO_SPEED_FREQ_VERY_HIGH	range 50 MHz to 80 MHz, refer to the product datasheet

Main modes

The modes in which we are interested initially are `GPIO_MODE_INPUT` and `GPIO_MODE_OUTPUT*`;

When the I/O is configured as `GPIO_MODE_INPUT`:

- The output buffer is disabled.
- The Schmitt trigger input is activated.
- The pull-up and pull-down resistors are activated depending on the value of the Pull field.
- A read access to the input data register provides the I/O state.

When the I/O port is programmed as `GPIO_MODE_OUTPUT*`:

- The output buffer is enabled as follow:
  - if mode is GPIO_MODE_OUTPUT_OD: A 0 in the Output register (ODR) activates the N-MOS whereas a 1 leaves the port in Hi-Z (the P-MOS is never activated);
  - if mode is GPIO_MODE_OUTPUT_PP: A 0 in the ODR activates the N-MOS whereas a 1 activates the P-MOS.
- The Schmitt trigger input is activated.
- The pull-up and pull-down resistors are activated depending on the value of the Pull field.
- A read access to the input data register gets the I/O state.
- A read access to the output data register gets the last written value.

Initializing a GPIO

```
void HAL_GPIO_Init(GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)
```

- `GPIOx`: where x can be (A..H) to select the GPIO peripheral for STM32L4 family

for example, suppose to want to initialize the GPIOA5 (Port A, pin 5), configuring it as output, push-pull, without pullup, i am going to call the function like this:

```
// A way to initialize a struct filled with 0s
GPIO_InitTypeDef GPIO_InitStruct = { 0 };

GPIO_InitStruct.Pin = GPIO_PIN_5;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;

void HAL_GPIO_Init(GPIOA, GPIO_InitTypeDef * GPIO_Init);
```

Driving a GPIO

Main functions are:

- To read the status of an I/O pin:

```
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
```

- To change the state of an I/O pin:

```
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
```

- To invert the state of an I/O pin:

```
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
```

For example, to put the GPIOA5 to high level, you will write:

```
GPIO_PinState state = GPIO_PIN_SET;
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, state);
```

given that

```
typedef enum
{
  GPIO_PIN_RESET = 0U,
  GPIO_PIN_SET
} GPIO_PinState;
```

De-init a GPIO

It is possible to set a GPIO pin to its default reset status (that is in Input Floating Mode) by mens of the function:

```
void HAL_GPIO_DeInit(GPIO_TypeDef *GPIOx, uint32_t GPIO_Pin)
```

CubeMX

Most configuration code is automatically generated by a graphical tool called CubeMX.

Useful links

CubeMX

STM32L4 Api Reference