

General Purpose Input/Output (GPIO)

Eduardo Barrera

Julián Nieto

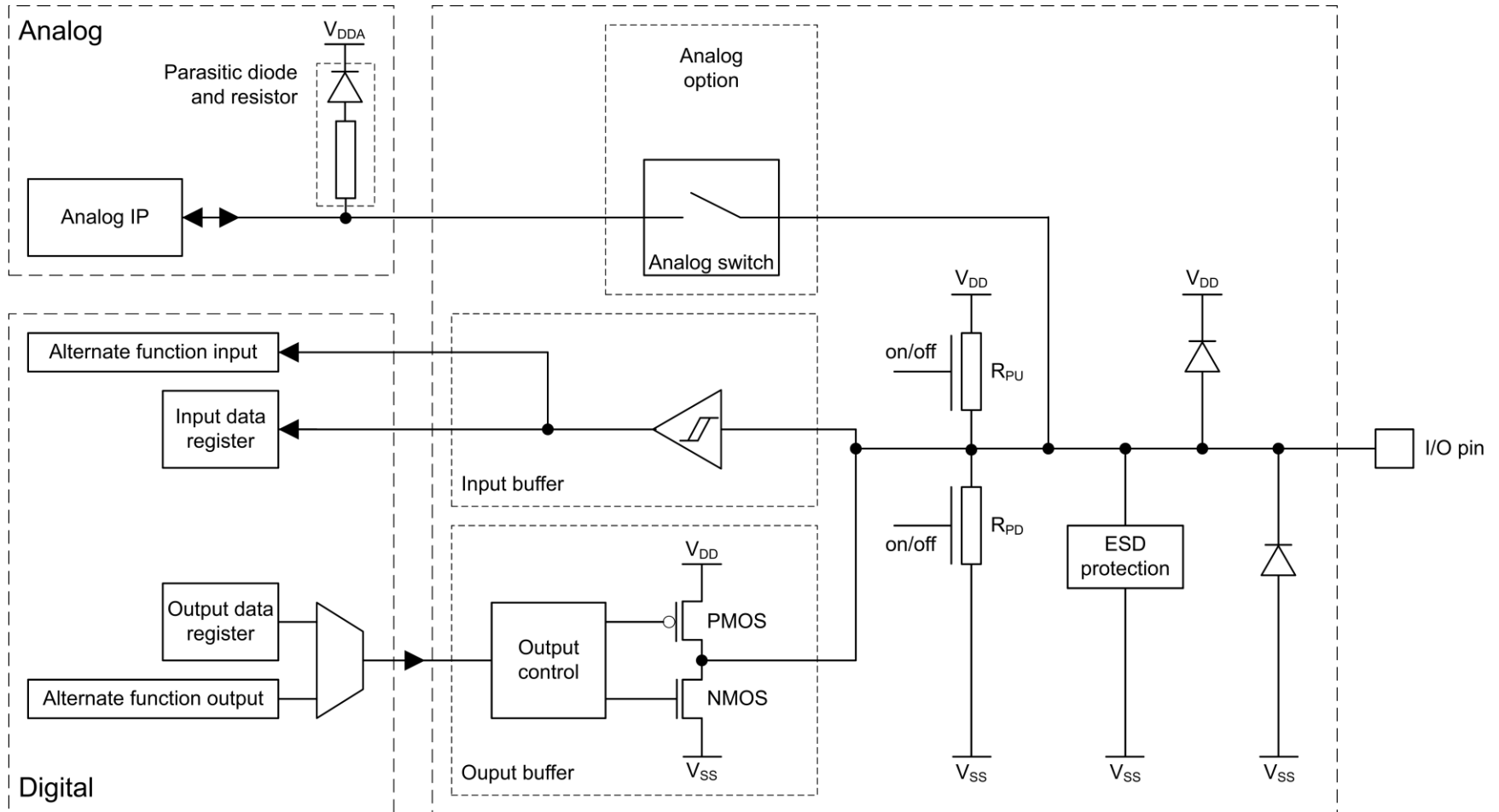
Juan Manuel López

Mariano Ruiz

Multifunction Pins

- A pin in the microcontroller can have multiple functions:
 - Digital I/O pin (input or output)
 - Specific peripheral function pin (for example the output of a hardware timer or an input clock signal) (Alternate function)
 - Analog input to connect to an ADC or DAC
- The pin can be configured in Open Drain, Pull-up or pull-down, high-speed or low-speed.

Multifunction PINs



MSv46872V1

Configuration (using registers)

a.- Configuration process

- 1.- GPIO port mode register (**GPIOx_MODER**)
- 2.- GPIO port output type register (**GPIOx_OTYPER**)
- 3.- GPIO port output speed register (**GPIOx_OSPEEDR**)
- 4.- GPIO port pull-up/pull-down register (**GPIOx_PUPDR**)

MODER(i) [1:0]	OTYPER(i)	OSPEEDR(i) [B:A]	PUPDR(i) [1:0]	I/O configuration	
01	0	SPEED [B:A]	0	0	GP output PP
	0		0	1	GP output PP + PU
	0		1	0	GP output PP + PD
	0		1	1	Reserved
	1		0	0	GP output OD
	1		0	1	GP output OD + PU
	1		1	0	GP output OD + PD
	1		1	1	Reserved (GP output OD)
10	0	SPEED [B:A]	0	0	AF PP
	0		0	1	AF PP + PU
	0		1	0	AF PP + PD
	0		1	1	Reserved
	1		0	0	AF OD
	1		0	1	AF OD + PU
	1		1	0	AF OD + PD
	1		1	1	Reserved
00	x	x	0	0	Input Floating
	x	x	0	1	Input PU
	x	x	1	0	Input PD
	x	x	1	1	Reserved (input floating)
11	x	x	0	0	Input/output Analog
	x	x	0	1	Reserved
	x	x	1	0	
	x	x	1	1	

Source: ST. RM0090 Reference manual

Direct Programming of GPIO

This code shows how manage I/O access using pointers.

```
volatile uint32_t *pGPIOA_MODER = 0;
volatile uint32_t *pGPIOA_ODR = 0;

pGPIOA_MODER = (uint32_t*)0x48000000; // Address of the GPIOA MODER register
pGPIOA_ODR = (uint32_t*)(0x48000000 + 0x14); // Address of the GPIOA ODR register

// Before use a peripheral, it must be enabled and connected to the AHB1 bus
// This will be done using the HAL

*pGPIOA_MODER = *pGPIOA_MODER | 0x04; // Sets MODER[3:2] = 0x1 and configure like Output
*pGPIOA_ODR = *pGPIOA_ODR | 0x02; // Sets PA1 high
```

Pointer declaration and initialization

Assign pointer specific peripheral address

Access peripheral:
a.- Configure pin.
b.- Access pin.



POLITÉCNICA



UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
DE SISTEMAS Y TELECOMUNICACIÓN



Sistemas Basados en Microprocesador

Programming GPIO

- Using HAL to manage the I/O peripheral.

```
GPIO_InitTypeDef GPIO_InitStructure = {0};  
  
//Init HAL. (Reset of all peripherals, Initializes the Flash interface and the SysTick.)  
HAL_Init();  
  
/* GPIO Port A Clock Enable */  
__HAL_RCC_GPIOA_CLK_ENABLE();  
  
/*Configure PA5 and PA2 like output */  
GPIO_InitStructure.Pin = GPIO_PIN_5 | GPIO_PIN_2;  
GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;  
GPIO_InitStructure.Pull = GPIO_NOPULL;  
GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;  
HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);  
  
/*Set GPIO pin Output Level to LOW */  
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
```

a.- Configuration process

b.- Read/Write the I/O pins

IMPORTANT

Before use **any** peripheral, the peripheral clock must be **ENABLED**

Fill in the handler and initialize the HAL

Handler initialization

HAL initialization **must be done** before using it

HAL GPIO Functions and Handlers

- Initialization and de-initialization

```
void HAL_GPIO_Init(GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_Init);  
void HAL_GPIO_DeInit(GPIO_TypeDef *GPIOx, uint32_t GPIO_Pin);
```

```
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);  
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState);  
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);  
HAL_StatusTypeDef HAL_GPIO_LockPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);  
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin);  
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin);
```

See: *stm32l4xx_hal_gpio.h*

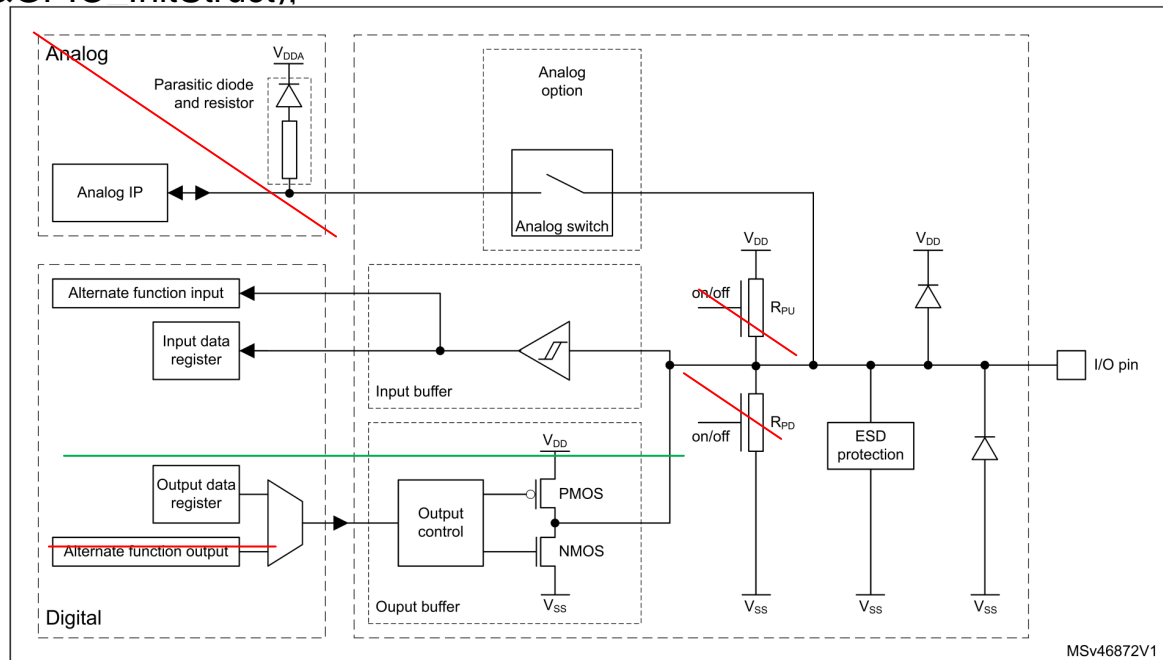
```
typedef struct {  
    volatile uint32_t MODER;  
    volatile uint32_t OTYPER;  
    volatile uint32_t OSPEEDR;  
    volatile uint32_t PUPDR;  
    volatile uint32_t IDR;  
    volatile uint32_t ODR;  
    volatile uint32_t BSRR;  
    volatile uint32_t LCKR;  
    volatile uint32_t AFR[2];  
    volatile uint32_t BRR;  
}  
GPIO_TypeDef;
```

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

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

Examples (Digital output)

```
GPIO_InitTypeDef GPIO_InitStructure = {0};
__HAL_RCC_GPIOB_CLK_ENABLE();
GPIO_InitStructure.Pin = GPIO_PIN_0;
GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStructure.Pull = GPIO_NOPULL;
GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStructure);
```



Examples (digital input - interrupt)

```
GPIO_InitTypeDef GPIO_InitStructure = {0};
__HAL_RCC_GPIOF_CLK_ENABLE();
GPIO_InitStructure.Pin = GPIO_PIN_6;
GPIO_InitStructure.Mode = GPIO_MODE_INPUT;
GPIO_InitStructure.Pull = GPIO_PULLUP; //GPIO_PULLDOWN //GPIO_NOPULL
HAL_GPIO_Init(GPIOF, &GPIO_InitStructure);
```

```
GPIO_InitTypeDef GPIO_InitStructure = {0};
__HAL_RCC_GPIOC_CLK_ENABLE();
GPIO_InitStructure.Pin = GPIO_PIN_13;
GPIO_InitStructure.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStructure.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOC, &GPIO_InitStructure);
```

Examples (Analog)

```
__HAL_RCC_GPIOA_CLK_ENABLE();  
GPIO_InitStruct.Pin = GPIO_PIN_3;  
GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;  
GPIO_InitStruct.Pull = GPIO_NOPULL;  
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

Examples (Alternate Function)

Table 12. STM32F427xx and STM32F429xx alternate function mapping

Port	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
	SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11	I2C1/2/3	SPI1/2/3/4/5/6	SPI2/3/SAI1	SPI3/USART1/2/3	USART6/USART4/5/7/8	CAN1/2/TIM12/13/14/LCD	OTG2_HS/OTG1_FS	ETH	FMC/SDIO/OTG2_FS	DCMI	LCD	SYS
Port A	PA0	TIM2_CH1/TIM2_ETR	TIM5_CH1	TIM8_ETR	-	-	-	USART2_CTS	UART4_TX	-	-	ETH_MII CRS	-	-	-	EVEN TOUT
	PA1	TIM2_CH2	TIM5_CH2	-	-	-	-	USART2_RTS	UART4_RX	-	-	ETH_MII RX_CLK/ETH_RMII_REF_CLK	-	-	-	EVEN TOUT
	PA2	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	-	-	USART2_TX	-	-	-	ETH_MDIO	-	-	-	EVEN TOUT
	PA3	TIM2_CH4	TIM5_CH4	TIM9_CH2	-	-	-	USART2_RX	-	-	OTG_HS_ULPI_D0	ETH_MII_COL	-	-	LCD_B5	EVEN TOUT
	PA4	-	-	-	-	SPI1_NSS	SPI3_NSS/I2S_WS	USART2_CLK	-	-	-	-	OTG_HS_DFS	DCMI_USING	LCD_USING	EVEN TOUT
	PA5	TIM2_CH1/TIM2_ETR	-	TIM8_CH1N	-	SPI1_SCK	-	-	-	-	-	-	-	-	-	-
	PA6	TIM1_BKIN	TIM3_CH1	TIM8_BKIN	-	SPI1_MISO	-	-	-	-	-	-	-	-	-	-
	PA7	TIM1_CH1N	TIM3_CH2	TIM8_CH1N	-	SPI1_MOSI	-	-	-	-	-	-	-	-	-	-
	PA8	MCO1	TIM1_CH1	-	-	I2C3_SCL	-	-	USART1_CK	-	-	-	-	-	-	-
	PA9	TIM1_CH2	-	-	-	I2C3_SMBA	-	-	USART1_TX	-	-	-	-	-	-	-
	PA10	TIM1_CH3	-	-	-	-	-	USART1_RX	-	-	-	-	-	-	-	-
	PA11	TIM1_CH4	-	-	-	-	-	USART1_CTS	-	CAN1_RX	OTG_FS_DM	-	-	-	LCD_R4	EVEN TOUT
	PA12	TIM1_ETR	-	-	-	-	-	USART1_RTS	-	CAN1_TX	OTG_FS_CS	-	-	-	LCD_B5	EVEN TOUT

```

__HAL_RCC_GPIOA_CLK_ENABLE();
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF1_TIM2;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

```

```

__HAL_RCC_GPIOA_CLK_ENABLE();
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.Alternate = GPIO_AF1_TIM2;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

```

Conclusions

- Some STM32 microcontroller pins can be configured for different functions
 - Use HAL to configure the PIN in the initialization of your system
 - Check the reference manual to see the Alternate Functions (to be used for timers, SPI, I2C, etc)
 - Review the HAL_GPIO code in Keil environment