



Sistemas Electrónicos Digitales

Tema #3

2. GPIO

1. Introducción
2. **GPIO: General Purpose Input/Output**
3. Arquitectura Arm Cortex-M4
4. Interrupciones
5. C en ensamblador
6. Temporizadores (Timers)
7. Direct Memory Access
8. Comunicaciones Serie
9. Conversores A/D y D/A

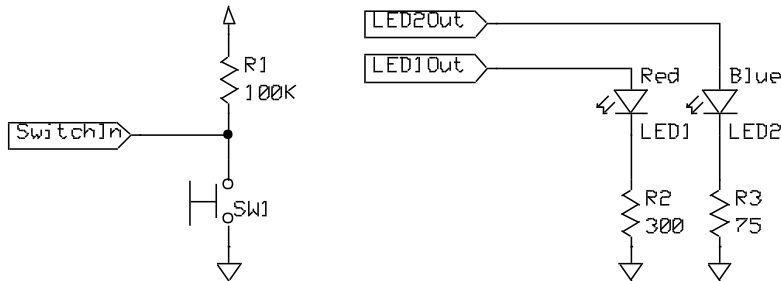


General Purpose Input Output

GPIO

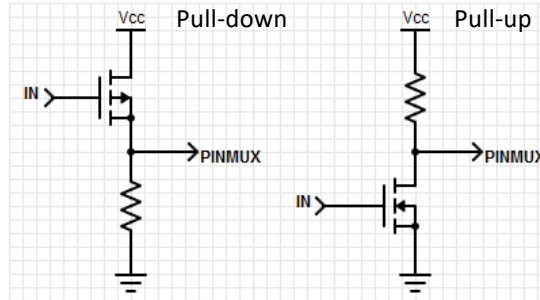
- ¿Cómo hacemos que un programa encienda LEDs en respuesta a un interruptor?
- GPIO
 - Conceptos básicos
 - Circuitería de puertos
 - Funciones alternativas
 - Mapas de memoria
 - Acceso a periféricos en C

Conceptos básicos



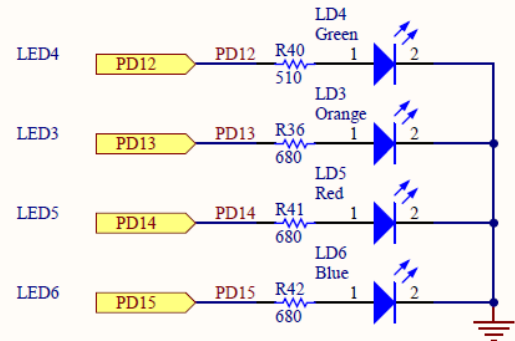
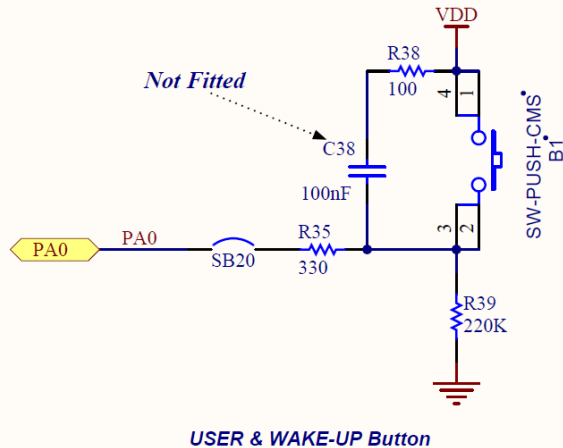
- Objetivo: encender LED1 o LED2 según la posición del interruptor SW1
- GPIO = Entrada y salida de propósito general (digital)
 - Entrada: el programa puede determinar si la señal de entrada es 1 o 0
 - Salida: el programa puede establecer la salida a 1 o 0
- Puede usar esto para interactuar con dispositivos externos
 - Entrada: interruptor
 - Salida: LED

Conceptos básicos: Resistencias Pull-Up & Pull-Down



- Aseguran un valor conocido en la salida si un pin se deja flotando
- En nuestro ejemplo, queremos que el interruptor SW1 “tire” del pin a tierra, por lo que habilitamos el pull-up
- El valor del pin es:
 - Alto cuando SW1 no está presionado
 - Bajo cuando se presiona SW1

Conceptos básicos: STM32F4 Discovery



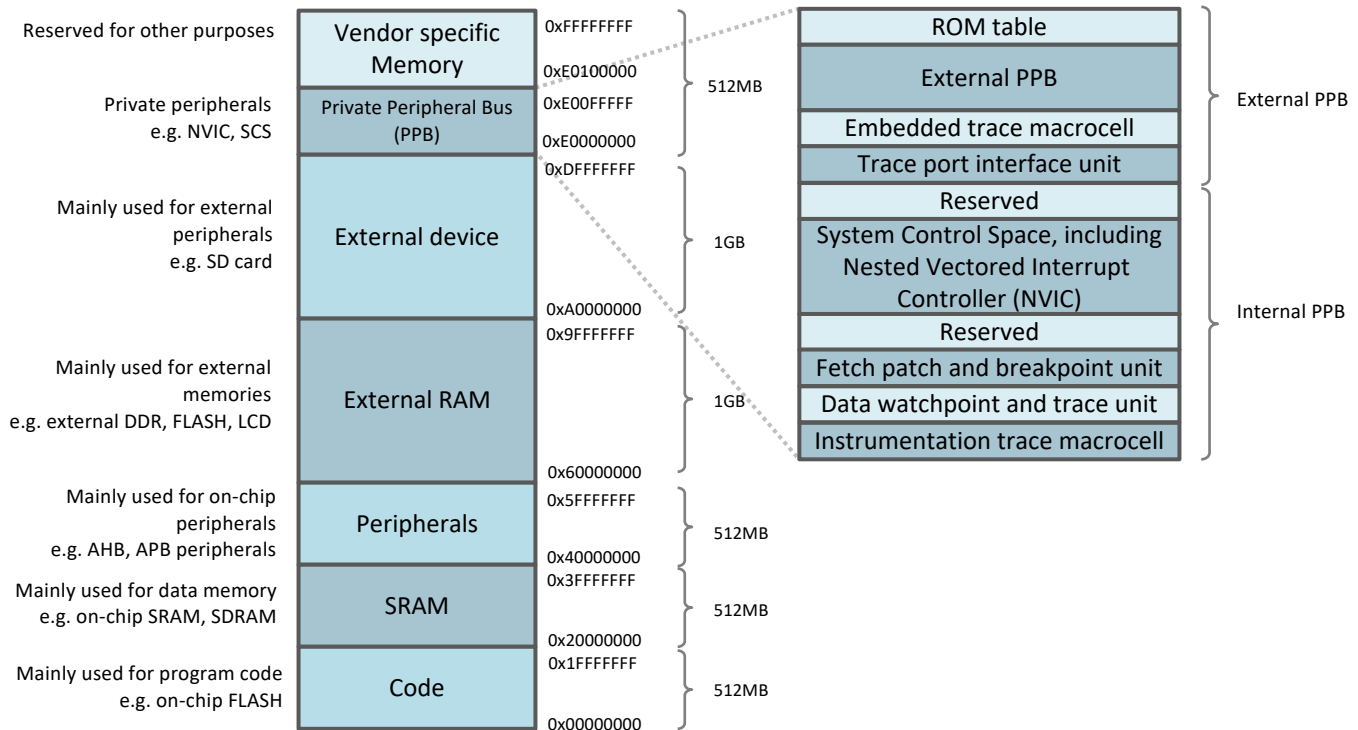
Ejemplo Con Lib

```
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    while (1)
    {
        if (HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)){
            HAL_GPIO_WritePin(GPIOD,GPIO_PIN_12,1);
        }else{
            HAL_GPIO_WritePin(GPIOD,GPIO_PIN_12,0);
        }
    }
}
```

Cortex-M4: mapa de memoria

- El procesador Cortex-M4 tiene **4 GB de espacio de direcciones de memoria**
- Este espacio se divide en **regiones**
 - Cada región se da para un **uso recomendado**
 - Fácil para el programador de software **conectar puertos** entre diferentes dispositivos
- Sin embargo, el uso real del mapa de memoria también puede ser definido de manera **flexible** por el usuario (excepto algunas direcciones de memoria fija, como el bus periférico interno privado)

Cortex-M4: mapa de memoria (II)



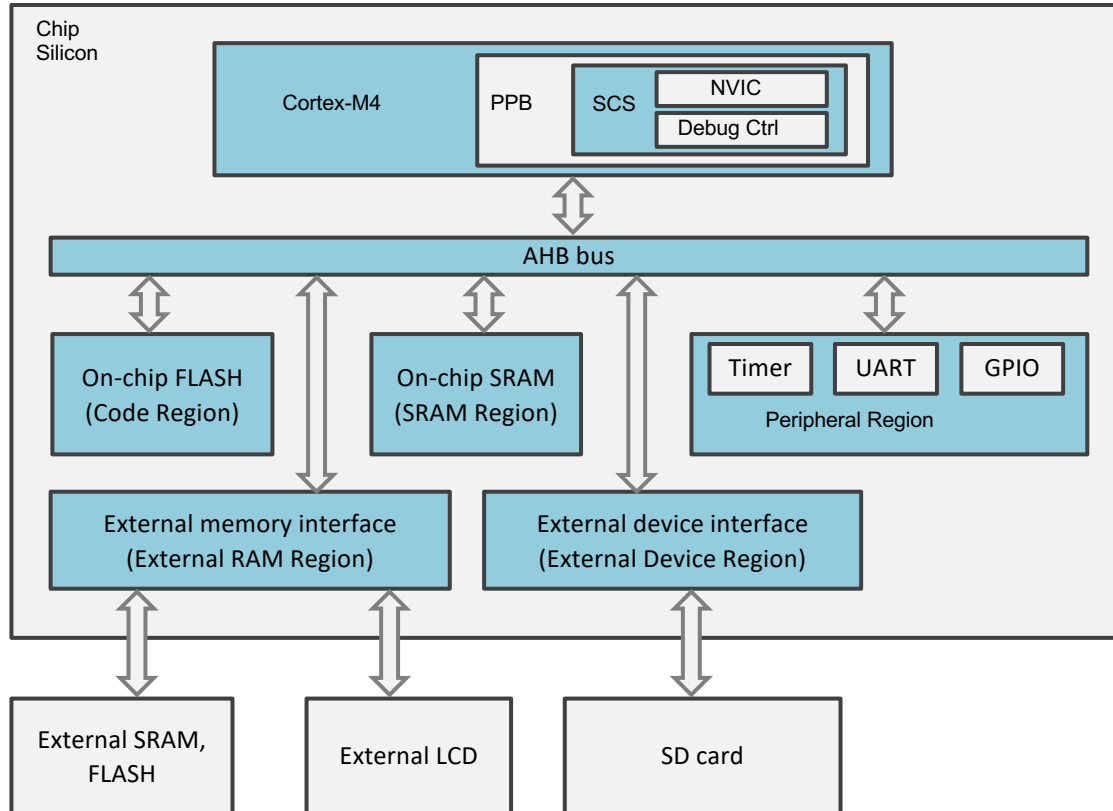
Cortex-M4: mapa de memoria (III)

- Región de **Código**
 - Código de programa (principalmente)
 - Opcionalmente para memoria de datos
 - On-chip memory, ej. on-chip FLASH
- Región **SRAM**
 - Para datos (principalmente), ej. heaps and stacks
 - Opcionalmente para código de programa
 - On-chip memory; puede ser SRAM, SDRAM u otros tipo
- Región de **periféricos**
 - Periféricos (principalmente), ej. Advanced High-performance Bus (AHB) o Advanced Peripheral Bus (APB)
 - On-chip peripherals

Cortex-M4: mapa de memoria (IV)

- Región de **RAM externa**
 - Primarily used to store large data blocks, or memory caches
 - Off-chip memory, slower than on-chip SRAM region
- Región de **dispositivos externos**
 - Principalmente para mapear dispositivos externos
 - Off-chip devices, ej. SD card
- Internal **Private Peripheral Bus (PPB)**
 - Usado dentro del core del procesador para control interno
 - Dentro del PPB, una zona especial de memoria se denomina System Control Space (**SCS**)
 - Es parte del SCS el “Nested Vectored Interrupt Controller” (**NVIC**)

Cortex-M4: ejemplo mapa de memoria



GPIO: acceso a nivel bajo

```
#define PERIPH_BASE      0x40000000U /*!< Peripheral base address
#define GPIOA            ((GPIO_TypeDef *) PERIPH_BASE+ 0x0020000U)
#define GPIOB            ((GPIO_TypeDef *) PERIPH_BASE+ 0x0020400U)
```

```
Uint32_t* p = 0x40020000U;
*p = valor_que_quiero_escribir_en_puerto
```

Vendor-specific memory	511MB	0xFFFFFFFF
Private peripheral bus	1.0MB	0xE0100000 0xE00FFFFF
External device	1.0GB	0xE0000000 0xDFFFFFFF
		0xA000 0000 - 0xA000 0FFF FSMC
External RAM	1.0GB	0xA0000000 0x9FFFFFFF
		0x60000000 0x5FFFFFFF
		0x4002 0000 - 0x4002 03FF GPIOA Peripheral
SRAM	0.5GB	0x40000000 0x3FFFFFFF
		0x20000000 0x1FFFFFFF
Code	0.5GB	+rápida 0x00000000

GPIO: acceso a nivel bajo (II)

```
#define PERIPH_BASE      0x40000000U /*!< Peripheral base address
#define GPIOA            ((GPIO_TypeDef *) PERIPH_BASE+ 0x00020000U)
#define GPIOB            ((GPIO_TypeDef *) PERIPH_BASE+ 0x00020400U)
```

typedef struct

```
{
  __IO uint32_t MODER; /*!< GPIO port mode register,      Address offset: 0x00 */
  __IO uint32_t OTYPER; /*!< GPIO port output type register,    Address offset: 0x04 */
  __IO uint32_t OSPEEDR; /*!< GPIO port output speed register,    Address offset: 0x08 */
  __IO uint32_t PUPDR; /*!< GPIO port pull-up/pull-down register, Address offset: 0x0C */
  __IO uint32_t IDR; /*!< GPIO port input data register,      Address offset: 0x10 */
  __IO uint32_t ODR; /*!< GPIO port output data register,    Address offset: 0x14 */
  __IO uint32_t BSRR; /*!< GPIO port bit set/reset register,    Address offset: 0x18 */
  __IO uint32_t LCKR; /*!< GPIO port configuration lock register, Address offset: 0x1C */
  __IO uint32_t AFR[2]; /*!< GPIO alternate function registers,    Address offset: 0x20-0x24 */
} GPIO_TypeDef;
```

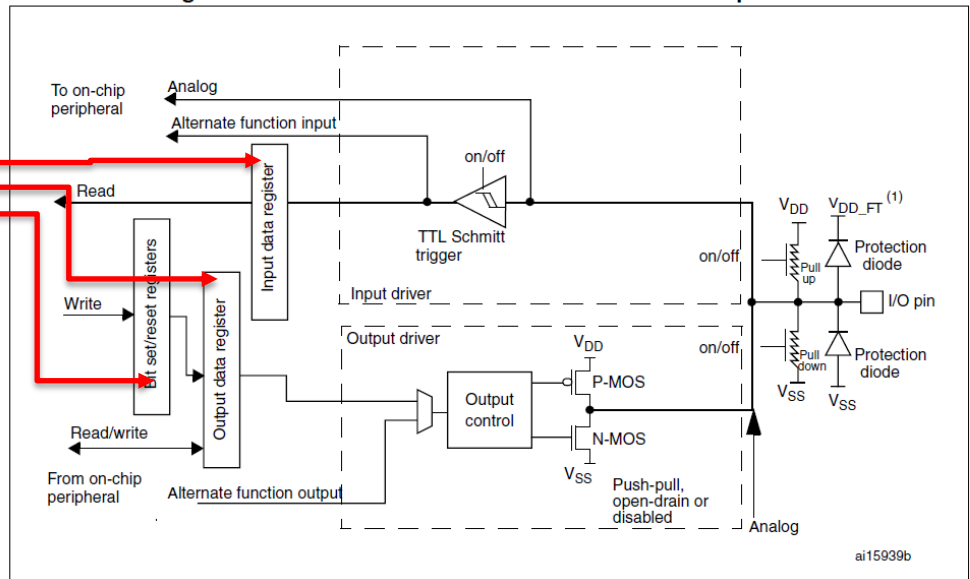
Vendor-specific memory	511MB	0xFFFFFFF
Private peripheral bus	1.0MB	0xE0100000 0xE00FFFFF
External device	1.0GB	0xE0000000 0xDFFFFFFF
FSMC		0xA000 0000 - 0xA000 0FFF
External RAM	1.0GB	0xA0000000 0x9FFFFFFF
GPIOA Peripheral	0.5GB	0x60000000 0x5FFFFFFF
SRAM	0.5GB	0x40000000 0x3FFFFFFF
Code	0.5GB	0x20000000 0x1FFFFFFF +rápida
		0x00000000

GPIO: acceso a nivel bajo (III)

```
typedef struct
```

```
{
  __IO uint32_t MODER;
  __IO uint32_t OTYPER;
  __IO uint32_t OSPEEDR;
  __IO uint32_t PUPDR;
  __IO uint32_t IDR;
  __IO uint32_t ODR;
  __IO uint32_t BSRR;
  __IO uint32_t LCKR;
  __IO uint32_t AFR[2];
} GPIO_TypeDef;
```

Figure 25. Basic structure of a five-volt tolerant I/O port bit



GPIO: acceso a nivel bajo (IV)

```
#define PERIPH_BASE      0x40000000U /*!< Peripheral base address
#define GPIOA            ((GPIO_TypeDef *) PERIPH_BASE+ 0x00020000U)
#define GPIOB            ((GPIO_TypeDef *) PERIPH_BASE+ 0x00020400U)
```

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

0x4002 2000 - 0x4002 23FF	GPIOI	APB2	Section 8.4.11: GPIO register map on page 286
0x4002 1C00 - 0x4002 1FFF	GPIOH		
0x4002 1800 - 0x4002 1BFF	GPIOG		
0x4002 1400 - 0x4002 17FF	GPIOF		
0x4002 1000 - 0x4002 13FF	GPIOE		
0x4002 0C00 - 0x4002 0FFF	GPIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		
0x4001 6800 - 0x4001 6BFF	LCD-TFT		Section 16.7.26: LTDC register map on page 510
0x4001 5800 - 0x4001 5BFF	SAI		Section 20.47.0: SAI register map on page 266

GPIO: acceso a nivel bajo (V)

typedef struct

```
{
    __IO uint32_t MODER;
    __IO uint32_t OTYPER;
    __IO uint32_t OSPEEDR;
    __IO uint32_t PUPDR;
    __IO uint32_t IDR;
    __IO uint32_t ODR;
    __IO uint32_t BSRR;
    __IO uint32_t LCKR;
    __IO uint32_t AFR[2];
} GPIO_TypeDef;
```

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

GPIO: acceso a nivel bajo (VI)

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

8.4.5 GPIO port input data register (GPIOx_IDR) (x = A..I/J/K)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy**: Port input data (y = 0..15)

These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

Ejemplo Sin Lib

```
GPIO_TypeDef* p_led = ((GPIO_TypeDef*)(AHB1PERIPH_BASE + 0x0400 * 3)); //0x40020c00
GPIO_TypeDef* p_switch = ((GPIO_TypeDef*)(AHB1PERIPH_BASE + 0x0400 * 0)); //0x40020000

int main(void)
{
    RCC->AHB1ENR|=1UL<<0; //enable clock output
    RCC->AHB1ENR|=1UL<<3; //enable clock output

    p_led->MODER = p_led->MODER | 0x55000000;
    p_switch->MODER = p_switch->MODER & 0xfffffff; //0b1111 1111 1111 1111 1111 1111 1111 1100 ;

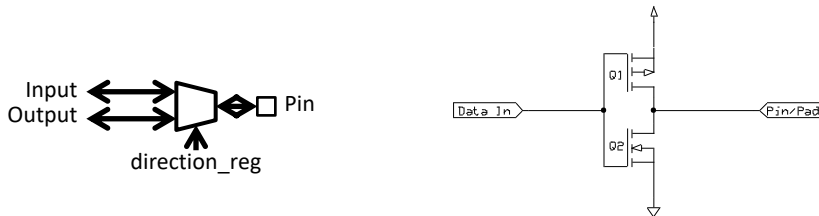
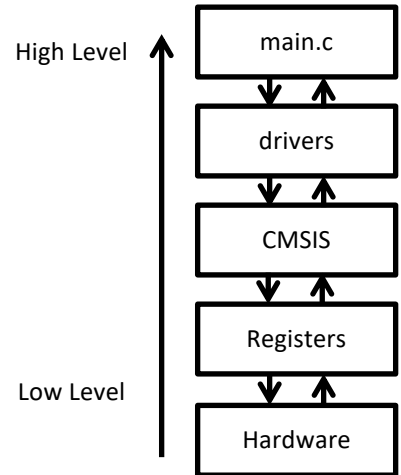
    while (1) {
        if (p_switch->IDR & (1<<0)) { //0b0000 0000 0000 0000 0000 0000 0000 0001
            p_led->ODR|=1UL<<12;
        } else {
            p_led->ODR&=~(1UL<<12);
        }
    }
}
```

RCC: Reset and Clock Control

AHB1ENR: peripheral clock enable register

Code Structure

- El código principal habla con los drivers, produciendo código fácil de leer y entender
 - `gpio_set_mode(P2_5, Output)`
- Los drivers usan la librería CMSIS y agrupan acciones relevantes:
 - `port_struct->direction_reg = output`
- CMSIS transforma los registros mapeados en memoria en estructuras de C
 - `#define PORT0 ((struct PORT*)0x2000030)`
- Los registros controlan el hardware directamente:



- El hardware controla los pines de E/S físicamente

BLINK!!!!

```
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    while (1)
    {
        HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);
        HAL_Delay(500);
    }
}
```



UNIVERSIDAD
POLITÉCNICA
DE MADRID

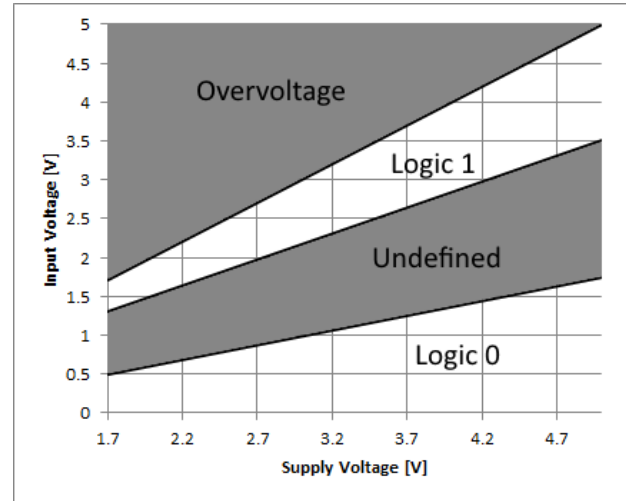
POLITÉCNICA

Inputs and Outputs, Ones and Zeros, Voltages and Currents

INTERFACING

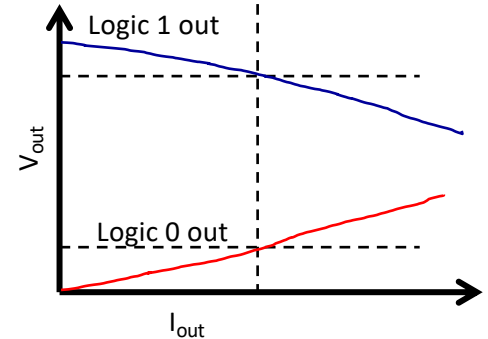
Entradas: ¿Qué es un Cero y un Uno?

- El valor de la señal de entrada se determina por voltaje
- Los voltajes de umbral de entrada dependen del voltaje de alimentación VDD
- Exceder VDD o GND puede dañar el chip



Outputs: What's a One? A Zero?

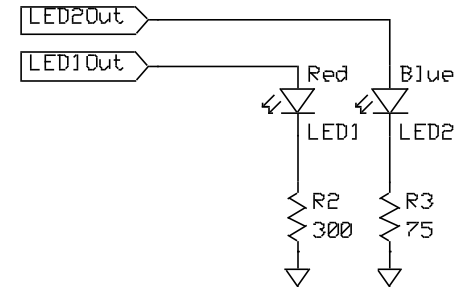
- Voltajes de salida nominales
 - 1: $V_{DD}-0.5\text{ V}$ to V_{DD}
 - 0: 0 to 0.5 V



- Nota: El voltaje de salida depende de la corriente consumida por la carga conectada al pin

Ejemplo de salida: LEDs

- Es necesario limitar la corriente a un valor que sea seguro tanto para el LED como para el controlador de puerto MCU
- Use resistencia limitadora de corriente
 - $R = (VDD - V_{LED})/I_{LED}$
- Supongamos: $I_{LED} = 4 \text{ mA}$
- V_{LED} depende del tipo de LED (sobre todo del color)
 - Red: $\sim 1.8 \text{ V}$
 - Blue: $\sim 2.7 \text{ V}$
- Calcular R dado $VDD = \sim 3.0 \text{ V}$
 - Red: 300Ω
 - Blue: 75Ω



Output Example: Driving a Speaker

- Crear una onda cuadrada con una salida GPIO
- Use el capacitor para bloquear el valor de CC
- Utilice la resistencia para reducir el volumen

```
void beep(void) {  
    unsigned int period = 20;  
    while (1) {  
        gpio_toggle(P_SPEAKER);  
        delay_ms(period/2);  
    }  
}
```

