

Sistemas Encaixados

Tema 4: Entrada e Saída

Oscar García Lorenzo

Escola Politécnica Superior de Enxeñaría

Entrada e Saída

- 1 Introducción
- 2 GPIO
 - Hardware
 - Programación
- 3 Temporización
 - Reloxo do Sistema
 - Temporizadores
 - Reloxo de tempo real
- 4 Acceso Directo a Memoria (e buses)
 - Acceso Directo a Memoria
- 5 ADC e DAC
 - ADC
 - DAC
- 6 Protocolos de Comunicación en Serie
 - Protocolos de Comunicación en Serie

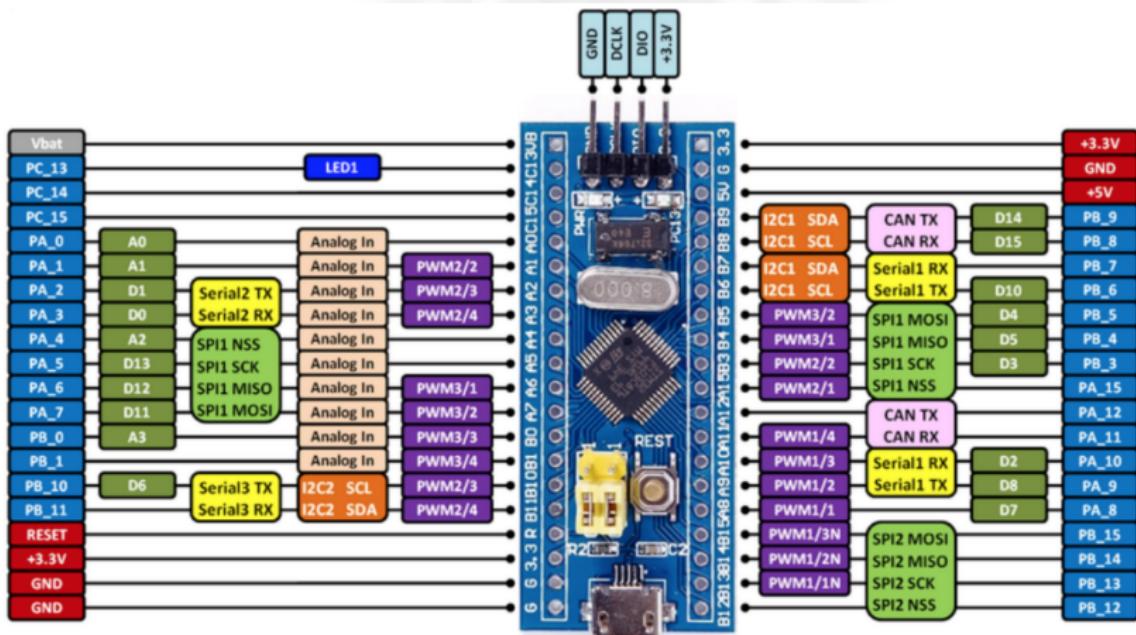
Índice

- 1 Introducción
 - 2 GPIO
 - Hardware
 - Programación
 - 3 Temporización
 - Reloxo do Sistema
 - Temporizadores
 - Reloxo de tempo real
 - 4 Acceso Directo a Memoria (e buses)
 - Acceso Directo a Memoria
 - 5 ADC e DAC
 - ADC
 - DAC
 - 6 Protocolos de Comunicación en Serie
 - Protocolos de Comunicación en Serie

Introducción

- Neste tema veremos a teoría da entrada/saída principalmente
- Volvemos a seguir o libro de Y. Zhu. "Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C"
- Veremos E/S para sistemas encaixados (salvo que se indique o contrario)
 - Recordade, para sistemas con SO complexos (linux, windows) a E/S a ten que realizar o SO, non se pode facer directamente
- O nome dos pins, conexións e demás é o que máis varía de SE en SE, cada fabricante fai o seu
 - Os exemplos do libro son para unha placa STM32L4, pero son extensibles a outros sistemas

Unha placa SMT32F1



Índice

- 1 Introducción
- 2 **GPIO**
 - Hardware
 - Programación
- 3 Temporización
 - Reloxo do Sistema
 - Temporizadores
 - Reloxo de tempo real
- 4 Acceso Directo a Memoria (e buses)
 - Acceso Directo a Memoria
- 5 ADC e DAC
 - ADC
 - DAC
- 6 Protocolos de Comunicación en Serie
 - Protocolos de Comunicación en Serie

GPIO

Pin GPIO

- O número de pins nun SE é limitado
- Un pin que se pode configurar por software é un pin GPIO (general-purpose input/output)
- Da máis flexibilidade aos SE, permítelles realizar máis funcións
- Complica o software, hai que inicializar
- 4 funcións:
 - Entrada dixital que detecta se un voltaxe de entrada é maior ou menor que un valor determinado
 - Saída dixital que controla o voltaxe do pin
 - Funcións analóxicas que realizan DAC ou ADC
 - Outras funcións complexas (LCD, interrupcións externas, USB ...) chamadas funcións alternativas (AF, alternate functions)
- Empezaremos por *entrada dixital e saída dixital*

GPIO

Porto GPIO

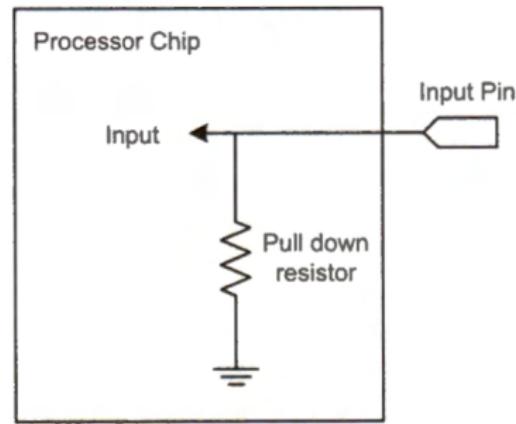
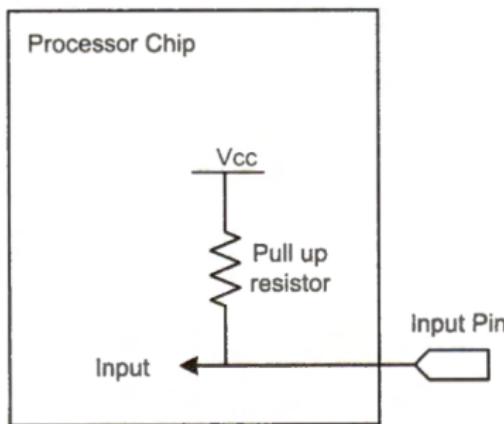
- Os pins agrúpanse en portos GPIO (xeralmente de 8 ou 16)
 - Comparten os mesmos rexistros de entrada e saída
 - Ollo!, non son rexistros coma os do microcontrolador, son rexistros en memoria
 - Cando un pin i está configurado como *entrada* pon o seu valor no bit i do rexistro *input data register* (IDR)
 - Cada bit i ten o valor do pin i
 - Cando un pin i está configurado como *saída* colle o seu valor do bit i do rexistro *output data register* (ODR)
 - Para cambiar o seu valor so hai que cambiar o do bit i , deixando o resto igual
 - Cada pin do porto pódese configurar independentemente

Entrada Dixital

- Unha entrada dixital pode ter 3 estados:
 - Alto voltaxe
 - Baixo voltaxe
 - Alta impedancia (flotante o tri-estado)
- 2 modos de configurar por software:
 - *Pull-up*: O pin conectase internamente cunha resistencia ao voltaxe, o valor é sempre 1 lóxico a non ser que o cambie o circuíto externo
 - *Pull-down*: O pin conectase internamente cunha resistencia a terra, o valor é sempre 0 lóxico a non ser que o cambie o circuíto externo
- Se non se configura quedará en modo alta impedancia e non se poderá saber que valor ten
- O modo pódese cambiar en tempo de execución

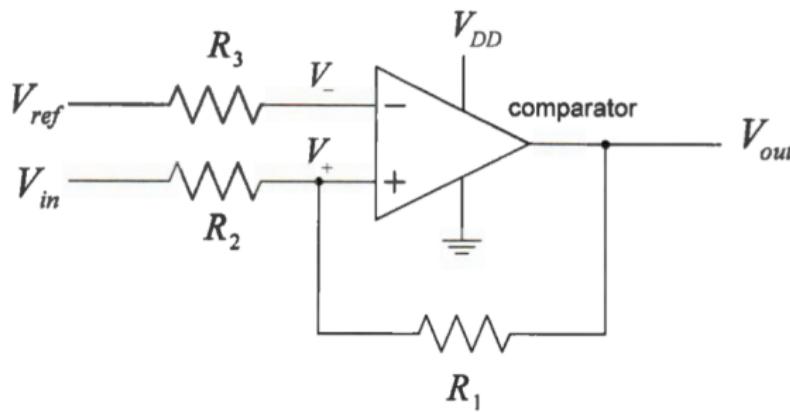
Entrada Dixital, modos

- Cando o circuito externo baixa (fai pull-down) un pin configurado como *pull-up* toma corrente do procesador
- Cando o circuito externo sube (fai pull-up) un pin configurado como *pull-down* mete corrente no procesador
 - Por iso as resistencias internas son moi altas ($>10K\Omega$)
- Se o circuito externo ten moita capacitancia pode ser difícil subir o pin, hai que face o pull-up ou down con resistencias externas.



Entrada Digital, Schmitt Trigger

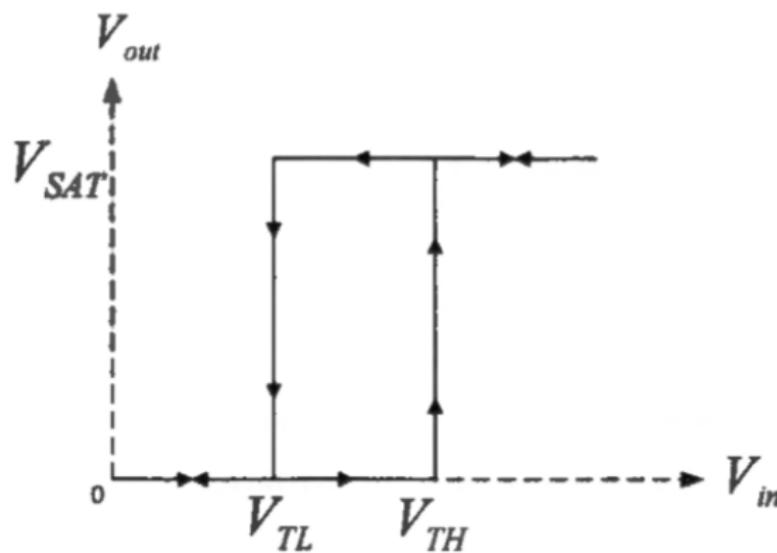
- O modulo GPIO soe incluir un disparador de Schmitt (Schmitt Trigger)
 - Serve para transformar unha sinal con ruído nunha sinal con bordes claros e transicións intantaneas



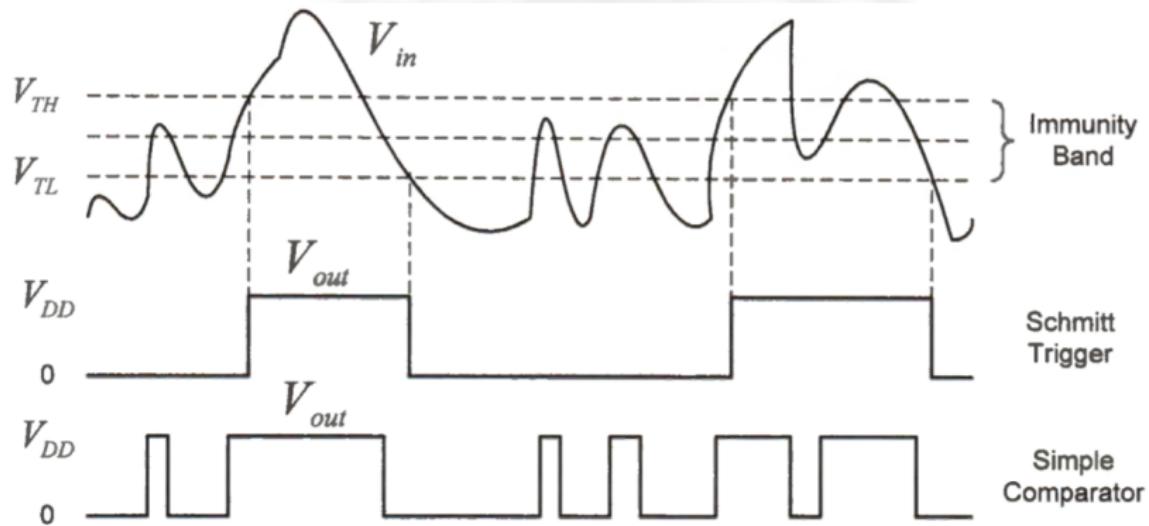
$$V_{out} = \begin{cases} V_{SAT} & \text{if } V_- < V_+ \\ 0 & \text{if } V_- > V_+ \end{cases}$$

Entrada Digital, Schmitt Trigger

- O valor de V_+ depende tanto de V_{out} como de V_{in}
 - Facendo teoría de circuitos obtemos V_{TH} (trigger high) e V_{TL} (trigger low)
 - Para cambiar a alto fai falta máis voltaxe que para cambiar a baixo



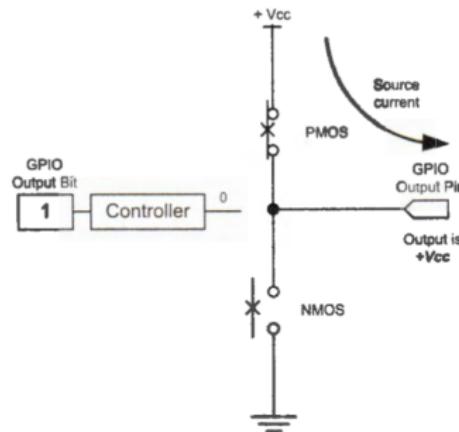
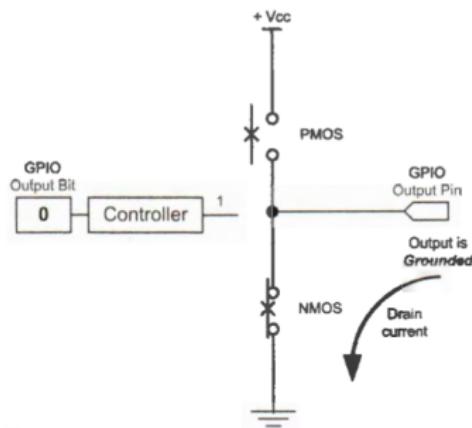
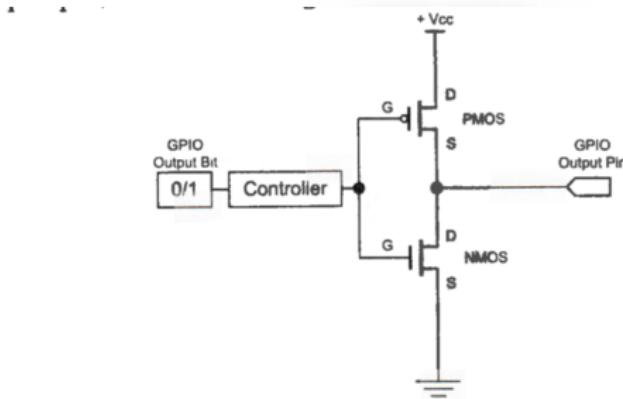
Entrada Dixital, Schmitt Trigger



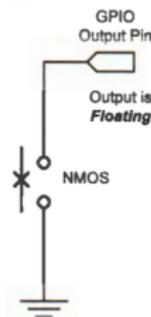
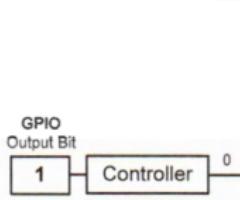
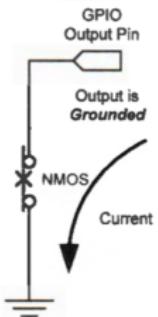
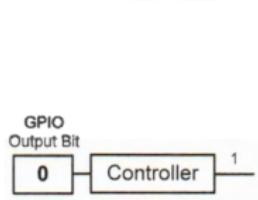
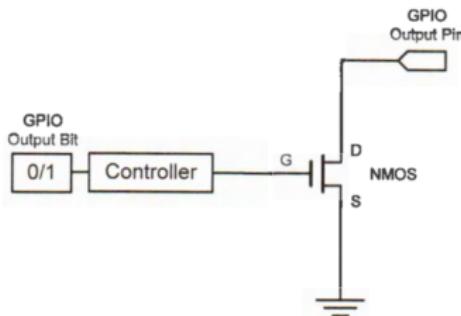
Saída Dixital, modos

- A saída dixital pódese configurar de dous modos:
- *Push-Pull*: Permite subministrar e absorber corrente
 - Par de transistores, so 1 activo en cada momento
 - Cando é 0 lóxico conéctase a terra
 - Cando é 1 lóxico da corrente
- *Open-Drain*: So pode absorber corrente (tamén chamado colector, *collector*)
 - Cando é 0 lóxico conéctase a terra
 - Cando é 1 lóxico queda flotante, en alta impedancia

Saída Dixital, Push-Pull



Saída Digital, Open-Drain

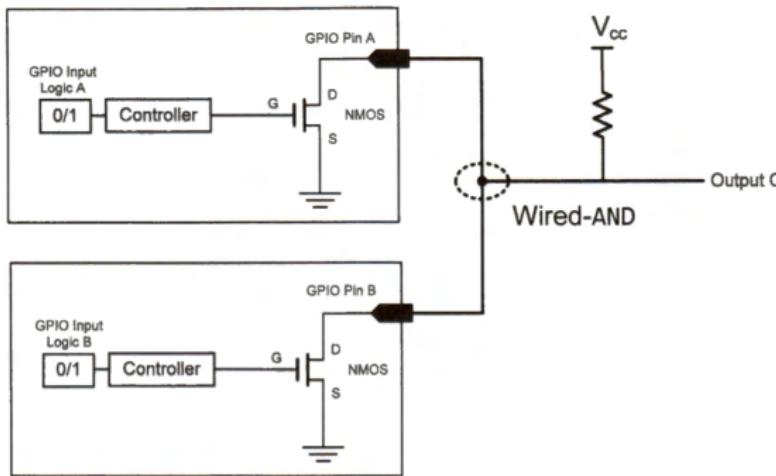


Saída Dixital, Open-Drain

- O Open-Drain permite conectar varias saídas directamente e facer un AND (active high) ou OR (active low) en hardware
 - Múltiples pins conectados con unha mesma resistencia poden ser baixados por calquera deles
 - Todas os pins teñen que estar a 1 para que a saída sexa 1
 - Se dicimos que a voltaxe alta representa un 1 lóxico todos teñen que estar con voltaxe alta (*high*) para obtelo (AND, active high)
 - Se dicimos que a voltaxe baixa representa un 1 lóxico basta un pin que teña voltaxe baixa (*low*) para obtelo (OR, active low)

Saída Digital, Open-Drain, AND cabreado

Inputs				Output
Logic A	Logic B	Circuit A	Circuit B	
0	0	Drain	Drain	0
0	1	Drain	Open	0
1	0	Open	Drain	0
1	1	Open	Open	1



Saída Dixital, comparativa modos

- O Open-Drain permite conectar varias saídas directamente e facer un AND (active high) ou OR (active low) en hardware
 - Non permite dar corrente ao circuíto (acender un led directamente), ten que tomala deste
 - Permite subir a calquera voltaxe, o que pode ser útil ao combinar SE
- O Push-Pull é más rápido nas transicións
 - Permite dar corrente ao circuíto (acender un led)
 - Pode haber un curtocircuíto se se conectan directamente
- Recordade, configurase en software
 - O mesmo circuíto de probas funciona ben con open-drain, curtocircuíto con push-pull

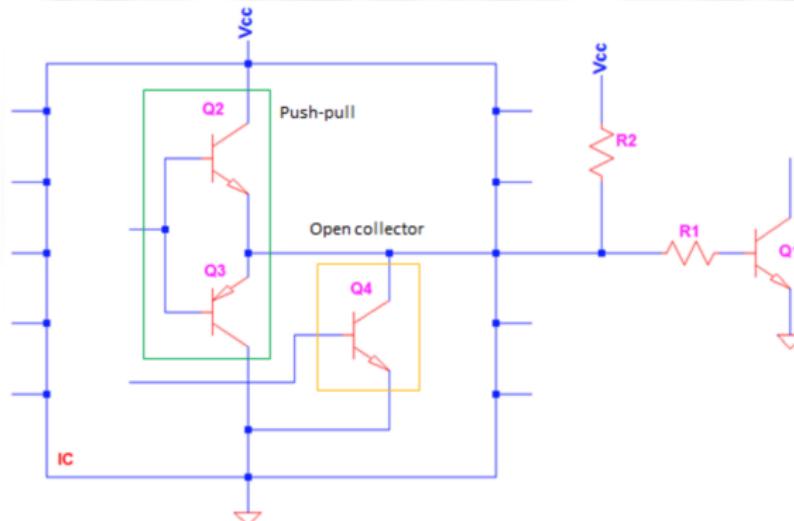
Saída Dixital, velocidade de subida

Velocidade de Subida (*slew rate*)

- A velocidade de subida da GPIO é a velocidade coa que cambia a súa voltaxe
- Se tarda 3s en cambiar de 0V a 3V é 1V/s
- Aplícase tanto ao borde de subida como de baixada
- Unha velocidade alta sería deseñable, pero causa interferencias electromagnéticas aos circuitos veciños (interferencia de radio frecuencia)
- Pódese cambiar por software, modificando a velocidade de saída (*GPIO output speed*)
 - Na STM32L pode ser baixa (low, 400Hz), media (medium, 2MHz), rápida (fast, 10MHz) e alta (high, 40MHz)

Entrada/Saída Dixital

- As imaxes da arquitectura dos pins non son exactamente como están implementados
 - Os pins teñen que implementar todas as configuracións e permitir cambiar entre elas por software



Entrada/Saída Mapeada a Memória

- Un periférico (incluso os interiores ao chip) soe ter rexistros propios: de control, status, entrada de datos, saída de datos ...
 - Tamén pode ter buffers de datos
 - As operacións E/S tratan sobre as comunicacións entre o núcleo do procesador/microcontrolador e esos periféricos
 - Hai dúas maneiras de facelo, en xeral:
 - Mapeando a porto: Usa instrucións máquina especiais para E/S
 - O espazo de direccións da memoria e da E/S está separado
 - A cada periférico se lle asignan un ou varios portos
 - É a típica en sistemas x86 (Intel)
 - Mapeando a memoria: Usa instrucións máquina normais *load* e *store* para E/S
 - O espazo de direccións da memoria e da E/S é compartido
 - A cada rexistro dun periférico se lle asigna unha dirección de memoria (direcionamento más complexo)
 - É a típica en SE

Entrada/Saída Mapeada a Memoria

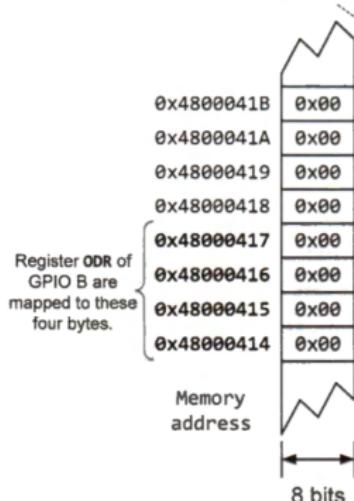
- As direccións usadas para os periféricos son elixidas polo fabricante
- Xeralmente non se poden cambiar por software
- Cada periférico soe ocupar 4 bytes
 - O rexistro ODR do porto B (16 pins) na STM32L4 vai da 0x48000414 a 0x48000417
 - En little-endian
- Para sacar *alto* no pin *i* hai que escribir un 1 na posición *i*
- Se queremos poñer en alto o pin 13:
 - `* (volatile uint32_t *) 0x48000414) I= 1UL<13;`

As Variables `volatile`

- O cualificativo `volatile` indica que o valor da variable pode cambiar aínda que o programa non o faga explicitamente
- Para usar unha variable por primeira vez hai que ler o seu valor de memoria (cargar a dirección, cargar o valor)
- Unha variable que se le moitas veces nun bucle, se non cambia o seu valor, o compilador optimiza o código e garda nun rexistro, sen acceder á memoria
- Pero en E/S ese valor pode cambiar porque o cambia o periférico ou unha interrupción
- Non se pode optimizar!
- Cada vez que se usa hai que cargar o valor (LDR) da dirección de memoria
- En Arduino, como se usan funcións para ler e escribir non é necesario, salvo que se usen interrupcións

Entrada/Saída Mapeada a Memória

The qualifier **volatile** informs the compiler that the value may have changed even though no statements in the program update it.



Method 1: Using numeric memory address directly

Dereferencing the casted pointer

*(volatile uint32_t *) 0x48000414) |= 1UL << 13;

Converting the address to a pointer to a 32-bit unsigned integer

Method 2: Casting an address to a pointer

```
#define GPIOB_ODR ((volatile uint32_t *) 0x48000414)  
*GPIOB_ODR |= 1UL << 13;
```

Method 3: Casting to a pointer and then dereferencing it

```
#define GPIOB_ODR  (*((volatile uint32_t *) 0x480000414))  
GPIOB_ODR |= 1UL<<13;
```

Entrada/Saída Mapeada a Memoria

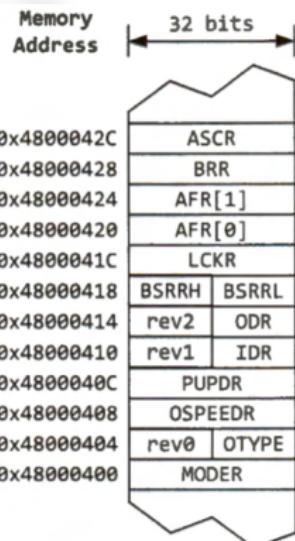
- Tipicamente para acceder aos rexistros dun periférico úsase unha estrutura
- Como están contiguos en memoria isto é fácil de facer en C
- Por exemplo, boa parte das librarías de Arduino son definicións das estruturas de E/S

Entrada/Saída Mapeada a Memoria: Exemplo

```
#define __IO volatile // allows read and write

typedef struct{
    __IO uint32_t MODER;      // Mode register
    __IO uint16_t OTYPER;     // Output type register
    uint16_t rev0;           // Padding two bytes
    __IO uint32_t OSPEEDR;    // Output speed register
    __IO uint32_t PUPDR;      // Pull-up/pull-down register
    __IO uint16_t IDR;        // Input data register
    uint16_t rev1;           // Padding two bytes
    __IO uint16_t ODR;        // Output data register
    uint16_t rev2;           // Padding two bytes
    __IO uint16_t BSRRL;      // Bit set/reset register(low)
    __IO uint16_t BSRRH;      // Bit set/reset register(high)
    __IO uint32_t LCKR;       // Configuration lock register
    __IO uint32_t AFR[2];     // Alternate function registers
    __IO uint32_t BRR;        // Bit reset register
    __IO uint32_t ASCR;       // Analog switch control register
} GPIO_TypeDef

#define GPIOB ((GPIO_TypeDef *) 0x48000400)
```



Entrada/Saída Mapeada a Memoria: Exemplo

- A estrutura anterior define o porto B da GPIO, 16 pins
- Indica as direccións de cada rexistro, definido polo fabricante
- MODER é o rexistro do modo, o OSPEED a velocidade de subida, ODR o rexistro de datos de saída
 - Neste caso necesitamos 2 bits para o modo e 2 para velocidad ($16 \times 2 = 32$)
 - Para os datos de saída basta con 1 bits (16)
 - Os rexistros están aliñados a 4 bytes (palabra), por iso hai bytes inútiles de *padding*
- Unha vez que temos definida a estrutura podemos facer un *cast* a súa dirección base de memoria
 - `#define GPIOB ((GPIO_TypeDef *) 0x48000400)`

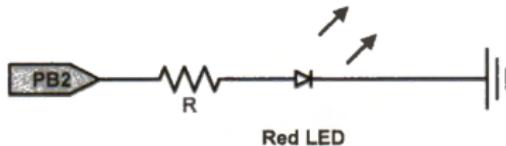
Entrada/Saída Mapeada a Memória: Exemplo

- Para cambiar un valor *i* usamos 1UL (1 unsigned long, 32 bits) e o desprazamos *i*
 - `GPIOB->ODR |= 1UL< <6; // Set bit 6`
- Pódense reutilizar as estruturas
 - `#define GPIOA ((GPIO_TypeDef *) 0x48000000)`
 - `#define GPIOC ((GPIO_TypeDef *) 0x48000800)`
- Pódense pasar as estruturas por punteiro
 - `void GPIO_Init (GPIO_TypeDef * GPIO);`
 - `void main(void) { GPIO_Init(GPIOA);`
`GPIO_Init(GPIOB); }`

Exemplo: Acender un LED

- Acender un LED
- Dous pasos:
 - Acender o reloxo do porto GPIO
 - Configurar o pin 2 como saída *push-pull*
 - Despois, sacar un 1 para acender
- Veremos C e ensamblador
- En ensamblador pódese usar EQU para as definicións (.equ en GNU)
- O reloxo da GPIO está apagado por defecto, hai que acendelo
 - O reloxo é un periférico en si mesmo
 - Ten rexistros onde se pode activar para cada outro periférico
- Hai que inicializar a GPIO
- En Arduino varias de estas cousas as fai automaticamente

Exemplo: Acender un LED



```
GPIOB_BASE EQU 0x48000400      ; Base memory address
GPIO_ODR    EQU 20              ; Byte offset of ODR from the base

LDR r7, =GPIOB_BASE           ; Load GPIO port B base address
LDR r1, [r7, #GPIO_ODR]        ; Read GPIOB->ODR
ORR r1, r1, #(1<<2)          ; Set bit 2
STR r1, [r7, #GPIO_ODR]        ; Write to GPIOB->ODR
```

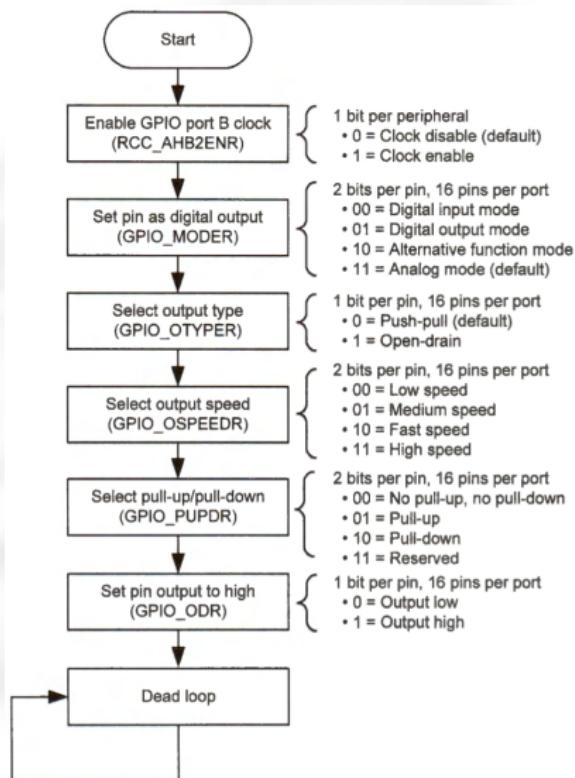
Exemplo: Acender un LED

```
// Reset and clock control
typedef struct {
    __IO uint32_t CR;          // Clock control register
    __IO uint32_t ICSCR;        // Internal clock sources calibration register
    __IO uint32_t CFGR;        // Clock configuration register
    ...
    __IO uint32_t AHB1ENR;      // AHB1 peripheral clocks enable register
    __IO uint32_t AHB2ENR;      // AHB2 peripheral clocks enable register
    __IO uint32_t AHB3ENR;      // AHB3 peripheral clocks enable register
    ...
} RCC_TypeDef;

#define RCC ((RCC_TypeDef *) 0x40021000)
```

```
#define RCC_AHB2ENR_GPIOBEN (0x00000002)
RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
```

Exemplo: Acender un LED



Exemplo: Acender un LED

- Cando se cambia o valor de bits específicos nun rexistro hai que recordar usar OR (`| =`) e non dar valor directamente (`=`)
- Cando se cambian varios bits é bo resetear os bits antes
 - Ao usar OR, se o seu valor era 1 seguirán a 1 aínda que queiramos poñelos a 0
 - Primeiro poñer todos a 0
 - Despois facer un OR cos valores que queremos
 - Rexistro R, 4 bits menos significativos a 1001:
 - `R &= ~0xF;`
 - `R |= 0x9;`
- Cada porto GPIO ten un rexistro ODR (output data register) e outro IDR (input data register)
 - Para poñer un pin en voltaxe alto hai que escribir un 1 no seu lugar no ODR
 - No IDR podemos ler o seu valor

Exemplo: Acender un LED, programa C

```
// Red LED is connected PB 2 (GPIO port B pin 2)
void GPIO_Clock_Enable(){
    // Enable the clock to GPIO port B
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
}

void GPIO_Pin_Init(){
    // Set mode of pin 2 as digital output
    // 00 = digital input,          01 = digital output
    // 10 = alternate function,    11 = analog (default)
    GPIOB->MODER &= ~(3UL<<4); // Clear mode bits
    GPIOB->MODER |= 1UL<<4;      // mode = 01, digital output

    // Set output type of pin 2 as push-pull
    // 0 = push-pull (default)
    // 1 = open-drain
    GPIOB->OTYPER &= ~(1<<2);

    // Set output speed of pin 2 as Low
    // 00 = Low speed,            01 = Medium speed
    // 10 = Fast speed,          11 = High speed
    GPIOB->OSPEEDR &= ~(3UL<<4); // Clear speed bits

    // Set pin 2 as no pull-up, no pull-down
    // 00 = no pull-up, no pull-down 01 = pull-up
    // 10 = pull-down,             11 = reserved
    GPIOB->PUPDR &= ~(3UL<<4); // no pull-up, no pull-down
}
```

Exemplo: Acender un LED, programa C

```
int main(void){  
    GPIO_Clock_Enable();  
    GPIO_Pin_Init();  
    GPIOB->ODR |= 1UL<<6; // Set bit 6 of output data register (ODR)  
    while(1); // Dead Loop & program hangs here  
}
```

Exemplo: Acender un LED, programa ensamblador

```
; Constants defined in file stm32l476xx_constants.s
;
; Memory addresses of GPIO port B and RCC (reset and clock control) data
; structure. These addresses are predefined by the chip manufacturer.
GPIOB_BASE      EQU    0x48000400
RCC_BASE        EQU    0x40021000

; Byte offset of each variable in the GPIO_TypeDef structure
GPIO_MODER      EQU    0x00
GPIO_OTYPER     EQU    0x04
GPIO_RESERVED0  EQU    0x06
GPIO_OSPEEDR    EQU    0x08
GPIO_PUPDR      EQU    0x0C
GPIO_IDR        EQU    0x10
GPIO_RESERVED1  EQU    0x12
GPIO_ODR        EQU    0x14
GPIO_RESERVED2  EQU    0x16
GPIO_BSRRL      EQU    0x18
GPIO_BSRRH      EQU    0x1A
GPIO_LCKR       EQU    0x1C
GPIO_AFR0        EQU    0x20 ; AFR[0]
GPIO_AFR1        EQU    0x24 ; AFR[1]
GPIO_AFRL        EQU    0x20
GPIO_AFRH        EQU    0x24

; Byte offset of variable AHB2ENR in the RCC_TypeDef structure
RCC_AHB2ENR     EQU    0x4C
```

Exemplo: Acender un LED, programa ensamblador

```
INCLUDE stm32l476xx_constants.s
AREA main, CODE, READONLY
EXPORT __main           ; make __main visible to Linker
ENTRY

__main PROC
; Enable the clock to GPIO port B
; Load address of reset and clock control (RCC)
LDR r2, =RCC_BASE          ; Pseudo instruction
LDR r1, [r2, #RCC_AHB2ENR] ; r1 = RCC->AHB2ENR
ORR r1, r1, #2              ; Set bit 2 of AHB2ENR
STR r1, [r2, #RCC_AHB2ENR] ; GPIO port B clock enable

; Load GPIO port B base address
LDR r3, =GPIOB_BASE         ; Pseudo instruction

; Set pin 2 I/O mode as general-purpose output
LDR r1, [r3, #GPIO_MODER]   ; Read the mode register
BIC r1, r1, #(3 << 4)      ; Direction mask pin 6, clear bits 5 and 4
ORR r1, r1, #(1 << 4)       ; Set mode as digital output (mode = 01)
STR r1, [r3, #GPIO_MODER]   ; Save to the mode register

; Set pin 2 the push-pull mode for the output type
LDR r1, [r3, #GPIO_OTYPER]   ; Read the output type register
BIC r1, r1, #(1<<2)        ; Push-pull(0), open-drain (1)
STR r1, [r3, #GPIO_OTYPER]   ; Save to the output type register

; Set I/O output speed value as Low
LDR r1, [r3, #GPIO_OSPEEDR]  ; Read the output speed register
BIC r1, r1, #(3<<4)        ; Low(00), Medium(01), Fast(01), High(11)
STR r1, [r3, #GPIO_OSPEEDR]  ; Save to the output speed register

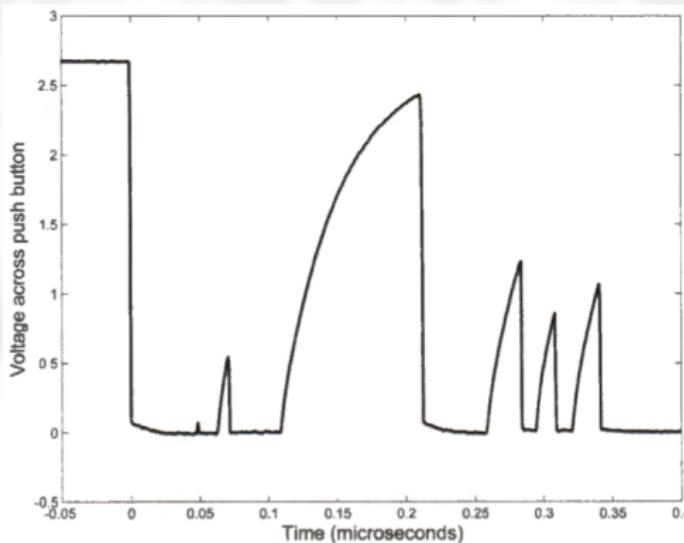
; Set I/O as no pull-up, no pull-down
LDR r1, [r3, #GPIO_PUPDR]    ; r1 = GPIOB->PUPDR
BIC r1, r1, #(3<<4)        ; No PUPD(00), PU(01), PD(10), Reserved(11)
STR r1, [r3, #GPIO_PUPDR]    ; Save pull-up and pull-down setting

; Light up LED
LDR r1, [r3, #GPIO_ODR]      ; Read the output data register
ORR r1, r1, #(1<<2)         ; Set bit 2
STR r1, [r3, #GPIO_ODR]      ; Save to the output data register

stop
B stop ; dead Loop & program hangs here
ENDP
END
```

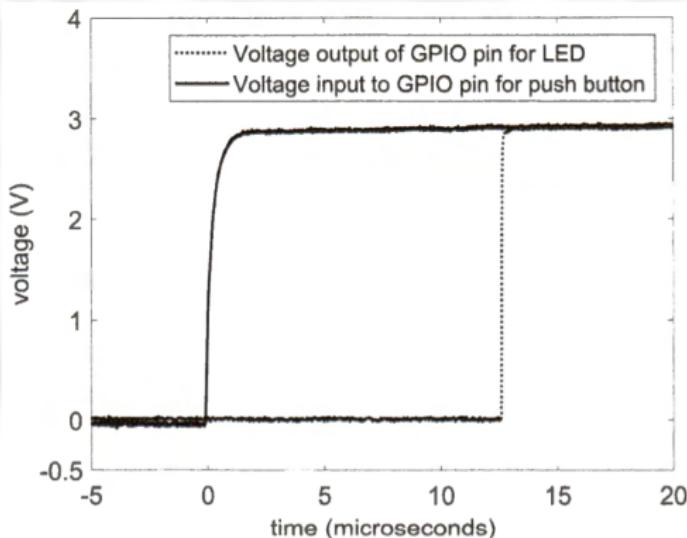
Exemplo: Pulsar un botón

- Pulsar un botón, acender un LED
 - Cando se pulsa un botón as placas metálicas rebotan e crean múltiples sinais (rebotes, *bounce*)
 - O microcontrolador pode pensar que se pulsou varias veces



Exemplo: Pulsar un botón

- En hardware os botóns poden usar un DEBOUNCER para evitalo
- Usa un condensador para suavizar



Exemplo: Pulsar un botón

- En software pódese usar *esperar e ver*
- Cando detectas unha pulsación esperas un tempo a ver se sigue pulsado

```
bool is_button_pressed(){
    read button input;
    if (button is not pressed) return false;
    wait 50 ms;
    read button input again;
    if (button is not pressed) return false;
    return true;
}
```

Exemplo: Pulsar un botón

- En software pódese usar *counter debouncer*
- Comprobas o valor cada menos tempo, incrementas o contador cada vez que está pulsado, reseteas se non
- Requires que o contador chegue a certo valor

```
bool is_button_pressed(){

    read button input;
    if (button is not pressed)
        return false;

    counter = 0;

    for(i = 0; i < 10; i++){
        wait 5 ms;
        read button input;
        if (button is not pressed) {
            counter = 0;           // bounce, reset counter
        } else {
            counter = counter + 1; // stable, increase counter
            if (counter >= 4)     // require 4 consecutive positive readings
                return true;
        }
    }
}
```

Exemplo: Pulsar un botón

- Para as esperas e os retardos pódese usar un bucle `for` ou `while` moi grande
- É más eficiente usar interrupcións do reloxo, con temporizadores
- É o que fai Arduino con `delay()`

```
volatile uint8_t counter = 0;
volatile uint8_t pressed = 0;

// Set up timer 4 to generate an interrupt every 5 ms
...
void TIM4_IRQHandler(void) {
    ...
    if((GPIOA->IDR & 0x1) == 0x1){ // check input on pin PA.0
        counter++; // button is pressed
        if (counter >= 4) {
            pressed = 1; // set the flag
            counter = 0; // reset counter
        } else { // button is not pressed
            counter = 0; // reset counter
        }
    }
}
```

Exemplo: Pulsar un botón

- Se temos un botón con *debouncer* por hardware podemos facer o programa sinxelo
- Mostraxe (*polling*) con espera activa (*busy waiting*)
- Moi ineficiente e non válido para tempo real
- Mellor usar interrupcións

```
// Enable the clock to GPIO port A and B
RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN | RCC_AHB2ENR_GPIOBEN;

// Set mode of pin 0 as general-purpose input
// 00 = Digital input,          01 = Digital output
// 10 = Alternate function,    11 = Analog
GPIOA->MODER &= ~3UL;           // Set mode as input (00)

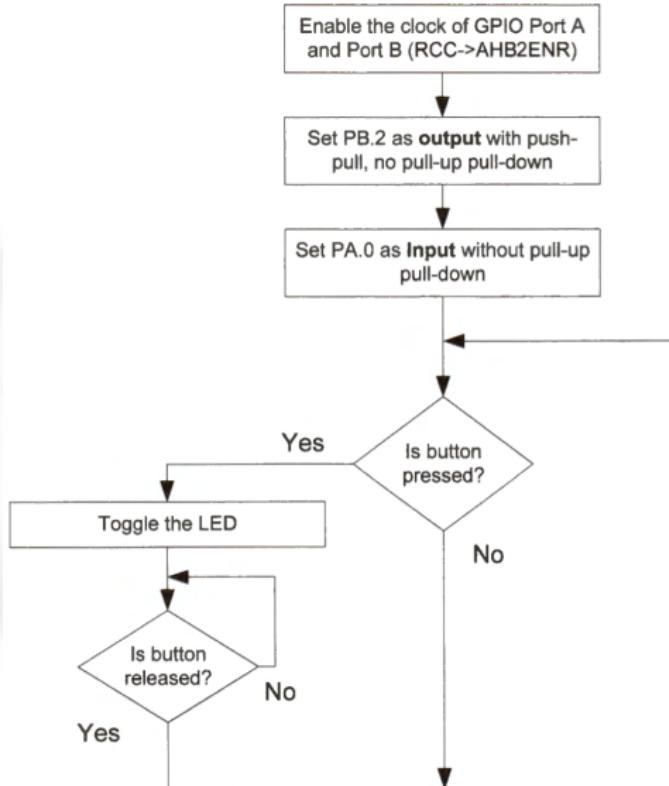
// Set I/O as no pull-up, no pull-down
// 00 = No pull-up/pull down, 01 = Pull-up
// 10 = Pull-down,             11 = Reserved
GPIOA->PUPDR &= ~3UL;           // Pull-up pull-down mask
```

Exemplo: Pulsar un botón

● Continuación

```
// Set PB.2 as digital output with push-pull, no pull-up/pull-down
// See Example 14-1
...
while (1) {
    // Toggle red LED when button PA.0 is pushed
    if((GPIOA->IDR & 0x1) == 0x1){
        GPIOB->ODR ^= GPIO_ODR_ODR_2;          // Toggle pin PB.2
        while((GPIOA->IDR & 0x1) != 0x00); // Wait until button is released
    }
}
```

Exemplo: Pulsar un botón



Interrupciones en Arduino

- En Arduino poden programar ISRs para interrupciones
- Para definir interrupciones necesitamos:
 - O pin que recibirá a sinal que disparará a interrupción
 - A condición de disparo
 - A función que se executará (*callback function, ISR*)
- As condicións de disparo poden ser:
 - LOW, a interrupción disparase cando a voltaxe é baixa
 - CHANGE, cando a voltaxe pasa de alta a baixa ou ao revés
 - RISING, en flanco de subida (de baixa a alta)
 - FALLING, en flanco de baixada (de alta a baixa)
 - HIGH, cando a voltaxe está alta (so para o Due)
- A función é `attachInterrupt()`

Interrupciones en Arduino

- Diferentes placas tienen diferente gestión de interrupciones e diferentes limitaciones
- Cada placa de Arduino soporta interrupciones en pins diferentes
 - As Uno, Mini e otras solo nos pins 2 e 3
 - As Mega nos 2, 3, 18, 19, 20, 21
 - As Due en todos os pins digitales
- A función `digitalPinToInterrupt(pin)` devolve o número de interrupción asociado ao pin
- Dentro das funciones ISR no se puede haber otras interrupciones, así que no funcionan `delay()` ou `millis()`
- As variables globais modificadas pola ISR tienen que ser `volatile`

Interrupciones en Arduino, pulsar botón, acender LED

```
1  const byte ledPin = 13;
2  const byte interruptPin = 2;
3  volatile byte state = LOW;
4
5  void setup() {
6    pinMode(ledPin, OUTPUT);
7    pinMode(interruptPin, INPUT_PULLUP);
8    attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
9  }
10
11 void loop() {
12   digitalWrite(ledPin, state);
13 }
14
15 void blink() {
16   state = !state;
17 }
```

Interrupciones en Arduino, onda cadrada

```
1 const int emuPin = 10;
2
3 const int LEDPin = 13;
4 const int intPin = 2;
5 volatile int state = LOW;
6
7 void setup() {
8     pinMode(emuPin, OUTPUT);
9     pinMode(LEDPin, OUTPUT);
10    pinMode(intPin, INPUT_PULLUP);
11    attachInterrupt(digitalPinToInterrupt(intPin), blink, RISING);
12 }
13
14 void loop() {
15     //esta parte es para emular la salida
16     digitalWrite(emuPin, HIGH);
17     delay(150);
18     digitalWrite(emuPin, LOW);
19     delay(150);
20 }
21
22 void blink() {
23     state = !state;
24     digitalWrite(LEDPin, state);
25 }
```

Interrupcóns en Arduino

- Hai más cousas sobre interrupcóns, incluso en Arduino, isto é só o uso básico
- Son a maneira de programar multitarefa en Arduino, sen usar a libraría de FreeRTOS

Índice

- 1 Introducción
- 2 GPIO
 - Hardware
 - Programación
- 3 Temporización
 - Reloxo do Sistema
 - Temporizadores
 - Reloxo de tempo real
- 4 Acceso Directo a Memoria (e buses)
 - Acceso Directo a Memoria
- 5 ADC e DAC
 - ADC
 - DAC
- 6 Protocolos de Comunicación en Serie
 - Protocolos de Comunicación en Serie

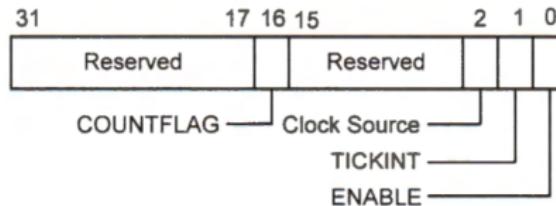
O reloxo do sistema

- O reloxo/temporizador de tick do sistema (*system tick timer, SysTick*) é un contador de 24 bits que decrece de forma constante
- Serve para crear retardos ou xerar interrupcóns periódicas
- O contador decrece de $N - 1$ a 0 e xera unha interrupción ao chegar a 0
- Ao chegar a 0 carga o valor no rexistro *SysTick Reload* e volve a empezar
- Non para de contar co procesador parado
- Tamén serve para controlar o planificador nos SO de tempo real

O reloxo do sistema

- Hai 4 rexistros para configuralo (cambian de nome segundo o fabricante, pero non de dirección)
- SysTick_CTRL (0xE000E010)
 - Para configurar e ver o estado
- SysTick_LOAD (0xE000E014)
 - Garda o valor ao que se reinicia o temporizador
- SysTick_VAL (0xE000E018)
 - Garda o valor actual do temporizador
- SysTick_CALIB EQU (0xE000E01C)
 - Garda o valor para contar 10ms
 - Pódese usar o seu valor para convertir entre segundos e ticks

SysTick_CTRL



- CLKSOURCE indica a fonte do reloxo
 - MSI: multi-speed internal, reloxo interno (oscilador) de frecuencia variable
 - HSI: high-speed internal, reloxo interno a 16MHz
 - PLL: Interno, lazo de seguimento de fase, o que usa o microcontrolador para as suas tarefas
 - HSE: high-speed external, reloxo externo
 - *LSI: Low-speed internal, reloxo interno (oscilador) a 32MHz, pouco consumo*
 - Diferentes placas poden ter más ou menos
- TICKINT indica se xerar ou non interrupción ao chegar a 0
- ENABLE acende o contador
- COUNTFLAG a 1 indica se chegou a cero desde a última lectura

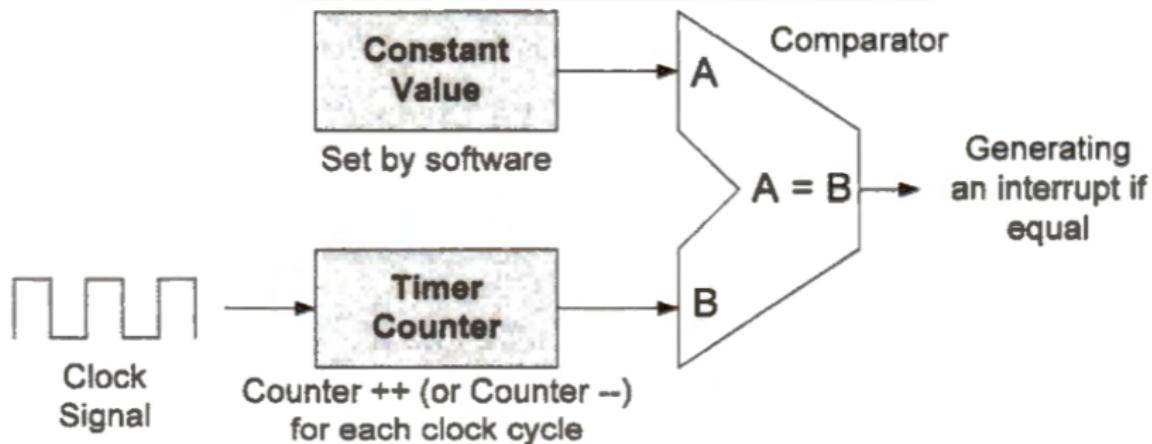
Reloxos

- Os reloxos internos son menos precisos
- Varían coa temperatura
- Desgástanse
- Mellor sempre son os externos
- Úsase un preescalador (*prescaler*) para frenar ou acelerar o reloxo

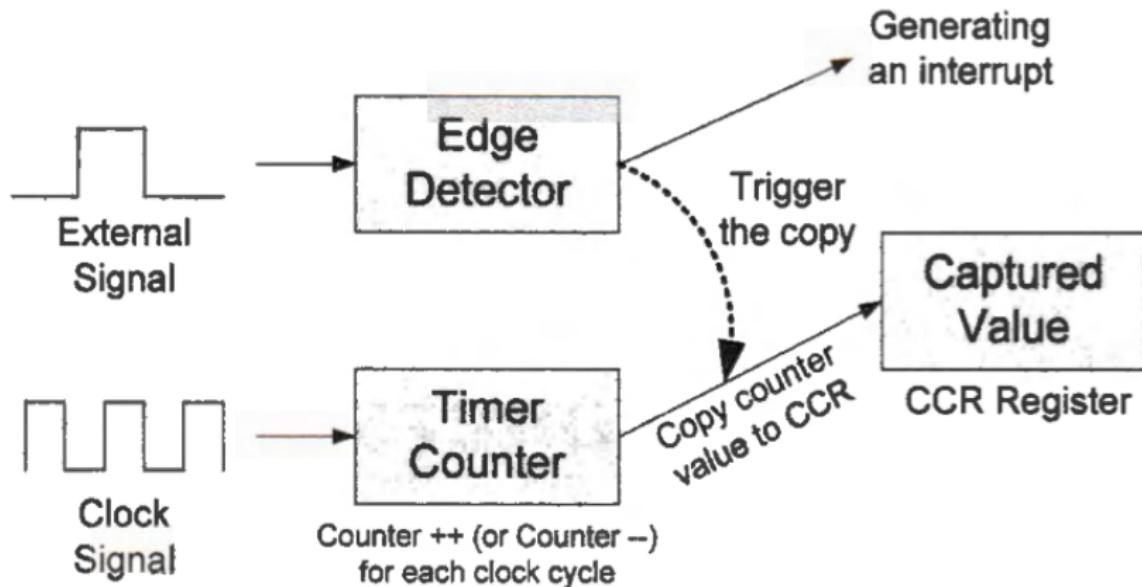
Temporizadores

- Hai temporizadores que poden funcionar directamente con entradas e saídas (o número varía coa placa)
- Como saída (*output compare*)
 - Compara o contador cunha constante programable
 - Cando son iguais xera unha saída ou interrupción
- Como entrada (*input compare*)
 - Detecta os flancos dunha sinal de entrada
 - Copiase o momento de subida/baixada da entrada automáticamente a un rexistro especial (CCR)
 - Xera unha interrupción
 - Así pódense medir tempos

Output compare



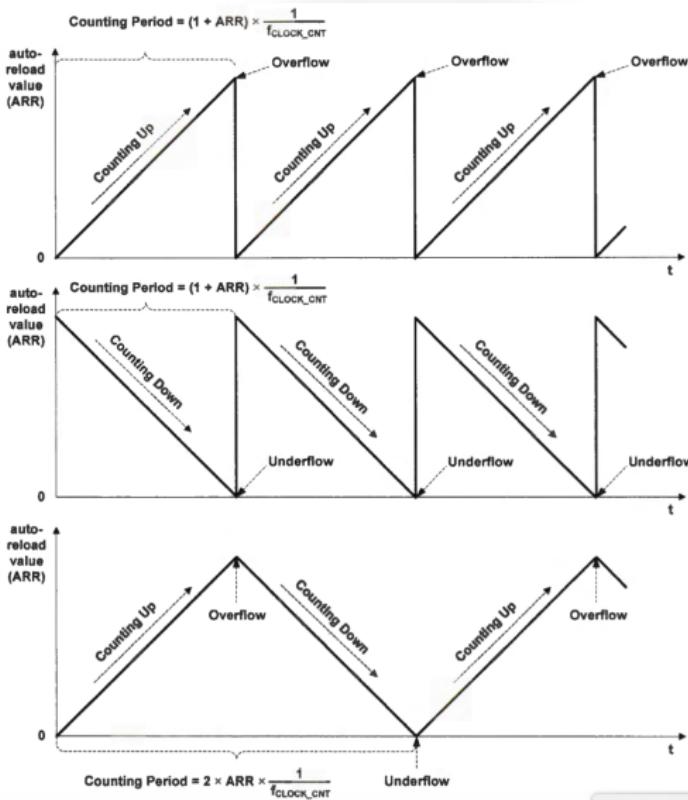
Input compare



Temporizadores

- O contador pode programarse para contar cara arriba (*up-counting*), cara cero (*down-counting*) ou alternar (*center-aligned counting*)
- Ao chegar ao final resetease co valor gardado no rexistro ARR, que é o programable
- Hai dous eventos *underflow* e *overflow*

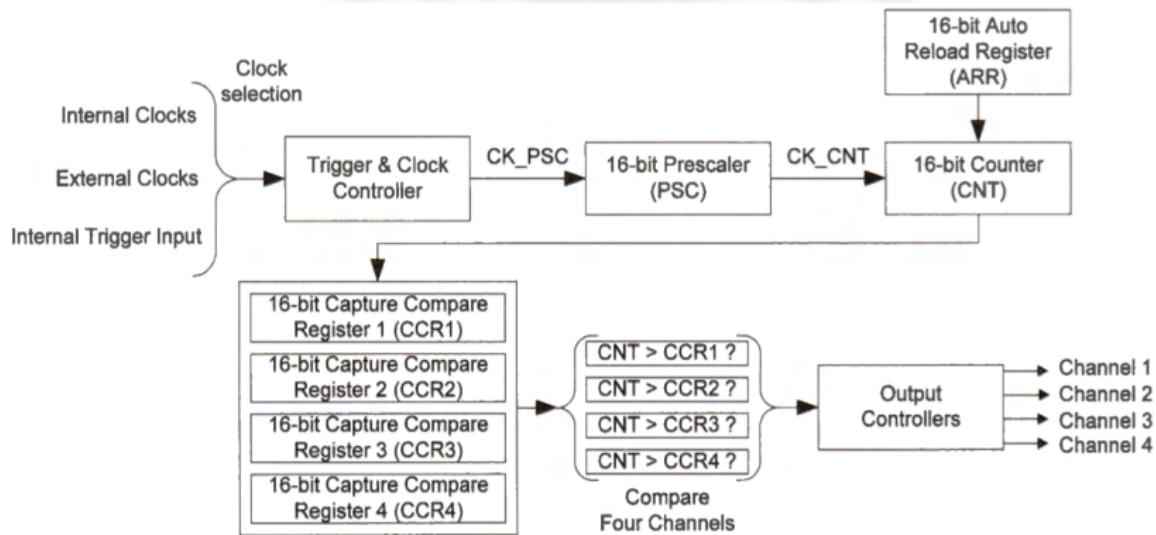
Configuraciones do contador



Comparar saída

- Pódese comparar o reloxo con 4 rexistros CCR diferentes (na STM32L)
- Pódese escalar o reloxo para frealo ou aceleralo
- O rexistro do contador do temporizador chámase CNT
- A saída pode ter valores diferentes dependendo de como se configure
- Pódese xerar unha interrupción cando $CNT=CCR$ (se se programa)

Comparar saída



Comparar saída

Output Compare Mode (OCM)	Timer Reference Output (OCREF)
Timing mode (0000)	Frozen
Active mode (0001)	Logic high if CNT = CCR
Inactive mode (0010)	Logic low if CNT = CCR
Toggle mode (0011)	Toggle if CNT = CCR
Forced inactive mode (0100)	Forced logic low (always low)
Forced active mode (0101)	Forced logic high (always high)
PWM output mode 1 (0110)	In up-counting: Logic high if $CNT < CCR$, else logic low In down-counting: Logic high if $CNT \leq CCR$, else logic low
PWM output mode 2 (0111)	In up-counting: Logic high if $CNT \geq CCR$, else logic low In down-counting: Logic high if $CNT > CCR$, else logic low

Table 15-1. Control of timer channel output

Comparar saída: Acender un LED

- Queremos pestanexar un LED, 2 segundos aceso e outros 2 segundos apagado
- Buscar que pins soportar como AF (Alternative Function) os temporizadores
 - Como non é algo tan usado as AF repártense, non todos teñen as mesmas
- Programar un temporizador
 - Hai que calcular o preescalador segundo a frecuencia de reloxo e o tempo que queremos
 - Se o reloxo vai a 80MHz, queremos 2KHz: $80M/2K = 39999$
 - Así no ARR podemos por 1999 para contar cada 2 segundos
 - En CCR poñemos un valor ente 0 e 1999
 - Dicimos que conte cara abaixo
 - Poñemos saída en modo *Toggle*

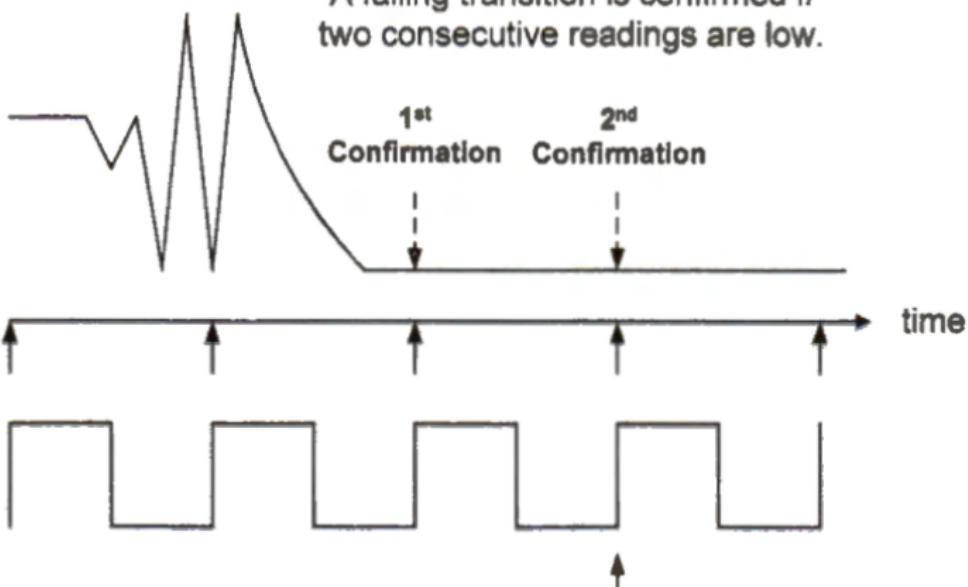
Comparar entrada

- Unha entrada pode ter flancos de subida e de baixada
 - O temporizador programase para capturar en flancos de subida, de baixada ou ambos
- Cando capture garda o CNT no CCR e xera unha interrupción
- Así pódese medir o tempo entre dous flancos (período da sinal ou tamaño do pulso)
- Cada canle de entrada:
 - Pode ser unha sinal en 1 ou varios pins (varios temporizadores) ou unha sinal interna
 - Detector de flancos (subida, baixada ou ambos)
 - Filtro de entrada, principalmente para filtrar ruído
 - Se a sinal tarda 10 ciclos en estabilizarse pódese poñer un filtro de más de 10 ciclos
- Configurase a frecuencia de mostraxe, o número de lecturas consecutivas válidas

Comparar entrada, filtro

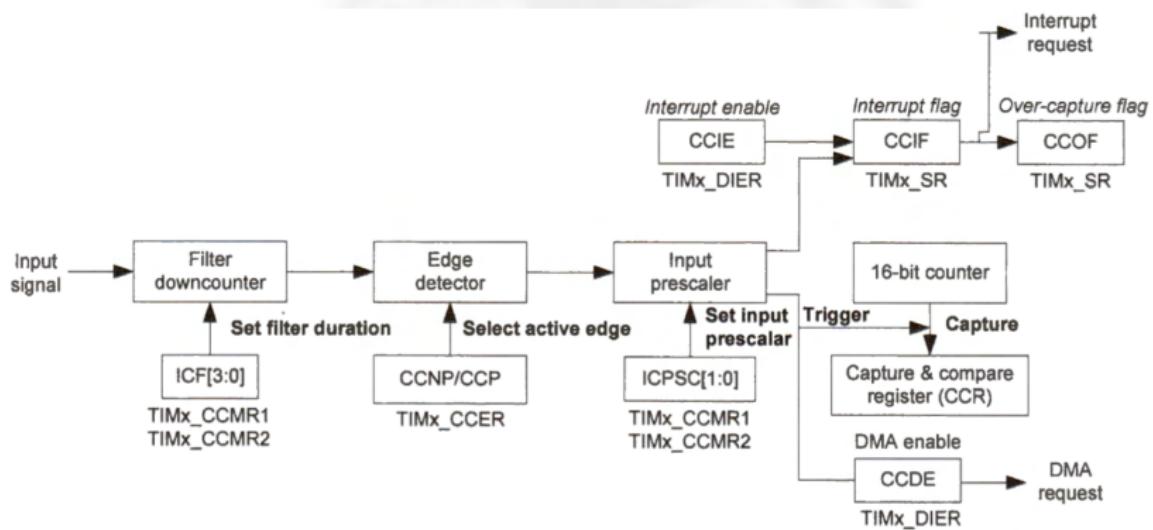
A falling transition is confirmed if two consecutive readings are low.

External
Signal



A falling transition is confirmed.

Comparar entrada



Comparar entrada

- Pódese usar co detector de distancia por ultrason
- Cando detecta unha sinal reflectida manda un pulso de 5V
- A duración do pulso é directamente proporcional á distancia ao obxecto

Reloxo de tempo real

- O reloxo de tempo real (real-time clock, RTC) úsase para levar conta do tempo de calendario
- Pódese usar para programar tarefas, espertar ao sistema, calibrar o resto de reloxos ...
- Consumen moi pouca enerxía
- Teñen unha batería propia para manter a data aínda que se apague o sistema

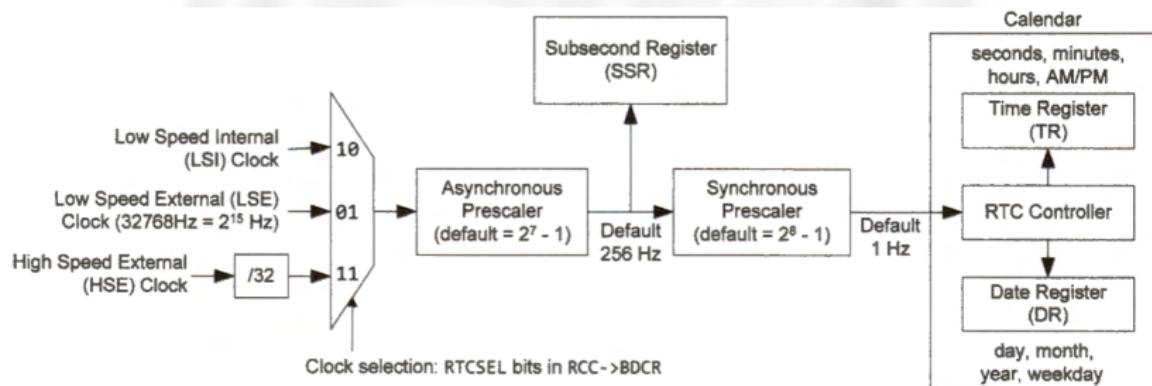
Reloxo de tempo real

Epoch Time, Unix Time

- É o tempo medido como un so enteiro
- 0 = 00:00:00 UTC do xoves 1 de xaneiro de 1970
- Enteiro con signo de 32 bits
 - 20:45:52 UTC do 13 de decembro de 1901
 - 03:14:07 UTC do 19 de xaneiro de 2038 (efecto 2038)
- A Lúa frea a rotación da Terra, 1.4ms por século
 - Engádense ás veces 1 segundo ao ano para que o sol do mediodía caia enriba de Greenwich
 - 27s desde 1972, Epoch non os ten en conta
 - Problemas con aplicacións astronómicas e satélites
- Algúns sistemas (o STM32L) gardan tamén a data en forma humana:
 - "08: 30: 45: 09 AM, MON AUG 14, 2017"

Reloxo de tempo real

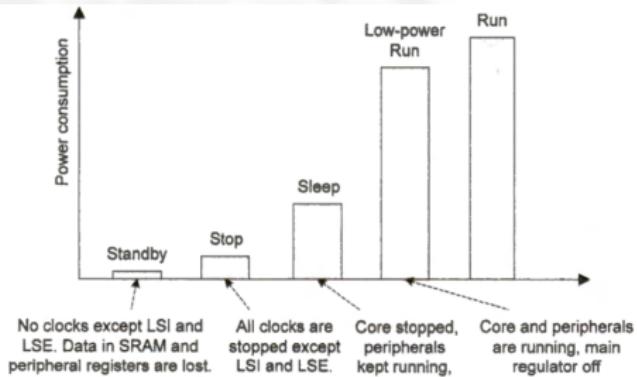
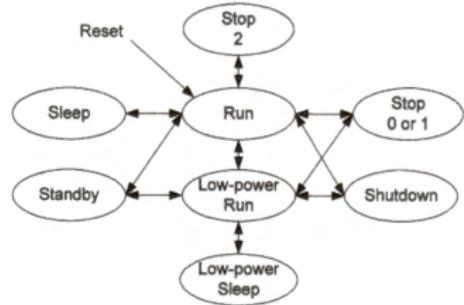
- Incrementase cada 1Hz, hai que escoller reloxo, preescalado
- Protexido contra escritura, operacións especiais para inicializalo
- Usa Binary Coded Decimal (BCD) para gardar en forma humana
 - Cada cifra 0 – 9 codifícase por separado
 - Fai máis sinxelo mandar os díxitos a unha pantalla, non hai divisións nin multiplicacións



Alarma

- Pode programar alarmas para realizar tarefas (ata dúas no STM32L)
- Cando coincide a data gardada na alarma coa real xera unha interrupción
- Pode enmascarar o día da semana, hora, minuto ou segundo para que non se comparen
- Pode usarse para espertar ao sistema:
 - Na STM32L hai 8 modos de enerxía
 - En *sleep* so se para o núcleo microcontrolador, os periféricos seguen acendidos
 - En *standby* so están acendidos os reloxos de baixo consumo (LSI, LSE)
 - Hai a instrución `WFI` (Wait For Interrupt) and `WFE` (Wait For Event) para entrar en *sleep*
 - O RTC pode xerar unha interrupción coa alarma

Modos de Enerxía



Índice

- 1 Introducción
 - 2 GPIO
 - Hardware
 - Programación
 - 3 Temporización
 - Reloxo do Sistema
 - Temporizadores
 - Reloxo de tempo real
 - 4 Acceso Directo a Memoria (e buses)
 - Acceso Directo a Memoria
 - 5 ADC e DAC
 - ADC
 - DAC
 - 6 Protocolos de Comunicación en Serie
 - Protocolos de Comunicación en Serie

Acceso Directo a Memoria

Direct Memory Access (DMA)

- É unha técnica que permite transferir datos directamente entre os periféricos e a memoria (ou memoria e memoria)
- O procesador programa o *controlador DMA* e despois esquecese
- Durante a transferencia de memoria o procesador está libre e pode facer outras cousas
 - Con periféricos lentos evita que o procesador teña que esperar
 - Con periféricos rápidos mellora a transferencia de información, xa que non ten que pasar pola CPU
 - Con periféricos de alta velocidade reduce o número de interrupcións xeradas, xunto coa sua sobrecarga
- Con DMA os periféricos non teñen por que ter memoria propia

Buses

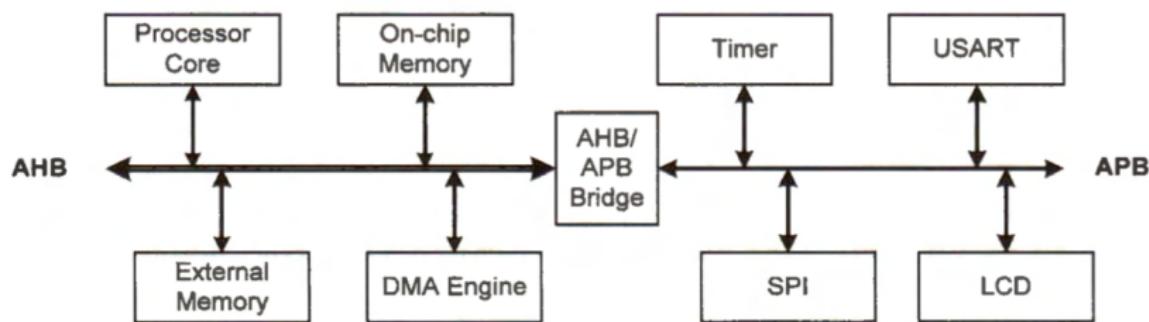
Advanced Microcontroller Bus Architecture (AMBA)

- É un estándar libre de comunicación dentro do chip para microcontroladores
- Especifica a arquitectura e os protocolos de 3 estándares de bus:
 - Advanced High-performance Bus (AHB)
 - Advanced System Bus (ASB)
 - Advanced Peripheral Bus (APB)
- Un bus é un conxunto de conexións (cables) que permite comunicarse a dous ou máis compoñentes
 - Usa un protocolo de comunicación
- Hai camiños distintos para datos, instrucións e control
- AMBA usa o modelo mestre/escravo, so o mestre pode iniciar a comunicación

Buses

- O AHB ou ASB é o bus principal, de alto rendemento e frecuencia de reloxo
 - Soporta *pipelining* para aumentar o ancho de banda
 - Permite modo ráfaga con arbitro
 - Pode haber varios mestres, o arbitro decide a que lle toca usar o bus
 - Envíase toda a información e se libera o bus
 - Cortex-M usa AHB (mellor rendemento)
 - O APB é máis sinxelo e ten menos consumo
 - Válido para periféricos lentos: USART, SPI, LCD ...
 - Non usa *pipelining*, menor ancho de banda
 - Todos os periféricos son escravos (non poden iniciar transferencia)

Buses



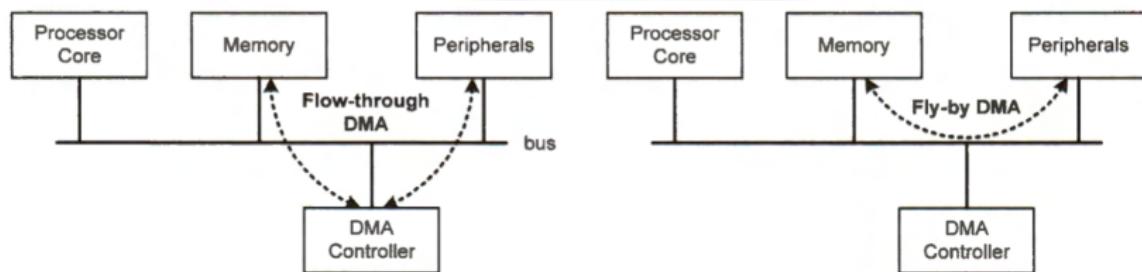
- O AHB está conectado a unha matriz de buses
 - Usa un planificador de quenda circular con prioridade para asegurar que todos poidan transmitir
 - Tamén arbitra conflitos entre peticións simultáneas
- O AHB e o APB están conectados por unha ponte
 - Escrava no AHB, mestra do APB
 - Ten buffers para xestionar a diferenza de velocidades
 - Traduce entre os protocolos

Controlador DMA

Controlador DMA

- O *controlador DMA* reside no bus AHB. Ten dous portos:
 - O porto escravo pode recibir datos e comandos do procesador para programar as transferencias
 - O porto mestre pode iniciar a transferencia no AHB ou pola ponte AHB/APB
- Ten varias canles programables independentemente
 - Mesma interface ao AHB, diferentes para os periféricos
 - Permiten realizar varias transferencias á vez
- As transferencias DMA poden ser *flow-through* ou *fly-by*

Controlador DMA



- **flow-through:** Os datos lense un a un, gárdanse no controlador DMA e logo escríbense no destino
 - Permite transferir entre aparellos con diferentes tamaños de rexistro (16, 32 bits...)
 - É a que ten a STM32
- **fly-by:** Os datos van directos sen pasar polo controlador
 - Máis eficientes
 - Non se poden usar entre memorias se non se pode escribir e ler no mesmo ciclo

Comparación DMA/Non DMA

- A STM32L ten 2 buses APB, dous controladores DMA con 7 canles cada un
- Sen DMA o procesador está ocupado coa transmisión
 - Cada lectura son 2 operacións, un `LDR` da dirección do periférico a un rexistro e un `STR` do rexistro á memoria
 - Ao revés coa escritura
 - Ten que usar espera activa ou interrupcións
 - Con periféricos rápidos e interrupcións pode ter moita sobrecarga
 - Pasas máis tempo facendo cambios de contexto que operacións útiles
- Con DMA o procesador configura a transferencia e a inicia
 - Pode configurar unha canle para responder ao USART
 - Se hai un evento no USART responde directamente o controlador DMA

Comparación DMA/Non DMA

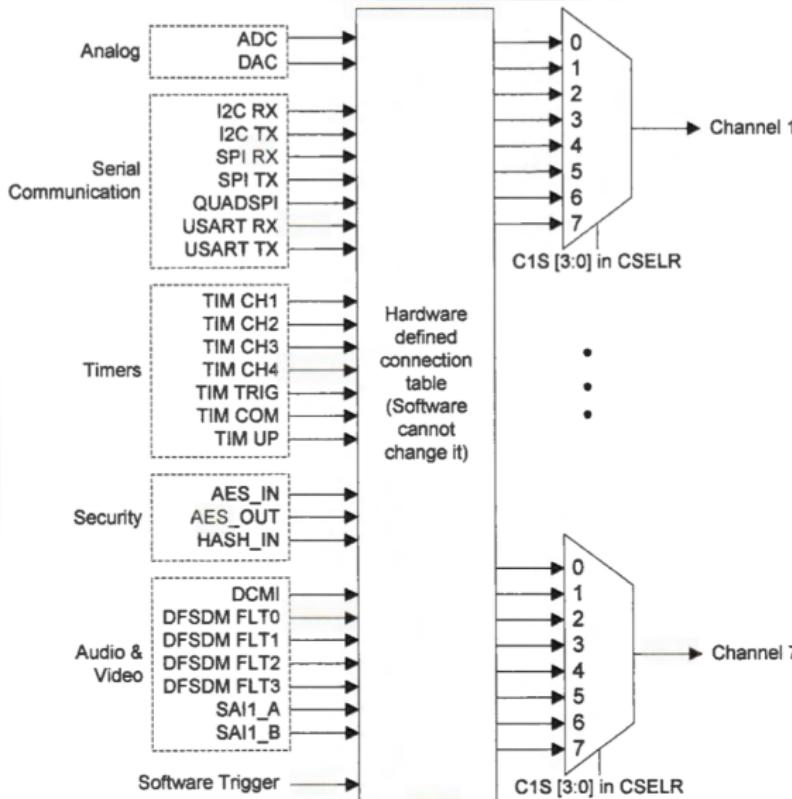
- Con DMA:

- Agora o procesador non se encarga de escribir e ler
 - Usa *flow-through*, os datos pasan polo controlador DMA
 - Séguese necesitando o mesmo número de transferencias
 - Pódese ocultar a súa latencia, xa que o procesador fai cousas nese tempo
 - O controlador DMA pode xerar unha interrupción para indicar que acabou ou se realizou unha transferencia
- O procesador pode pausar a transferencia se necesita usar o bus para acceder á mesma memoria ou periférico

Canles DMA

- A STM32L ten 2 buses APB, dous controladores DMA con 7 canles cada un
 - Cada canle pode realizar transferencias independentes
 - Cada canle ten o seu/súa propio/a:
 - Fonte, Destino, Dirección de transferencia, Ancho de transferencia, Disparador
 - Unha canle activada pode escoller entre diferentes eventos disparadores (8 na STM23L, diferentes para cada canle)
 - O disparador software está en todas (iniciar a transferencia con código)
 - Cada canle ten 2 prioridades, en caso de conflicto ten o bus o de máis prioridade
 - *Prioridade Software*: 4 niveis configurables por software
 - *Prioridade Hardware*: A canle 1 ten a maior prioridade, a 7 a menor, úsase para desempatar se a *software* é a mesma

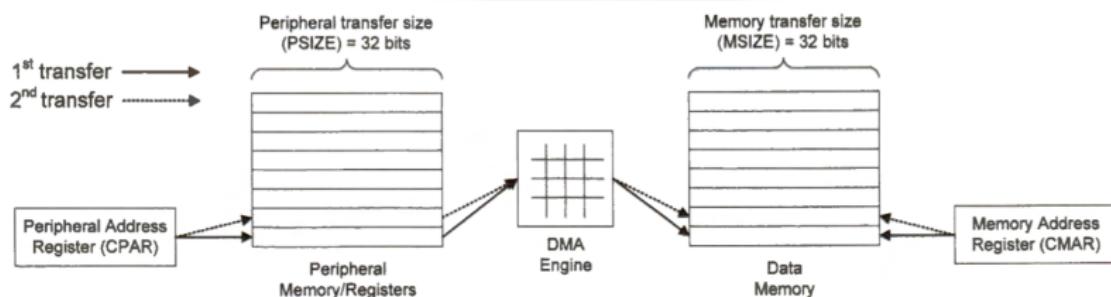
Canles DMA



Programar DMA

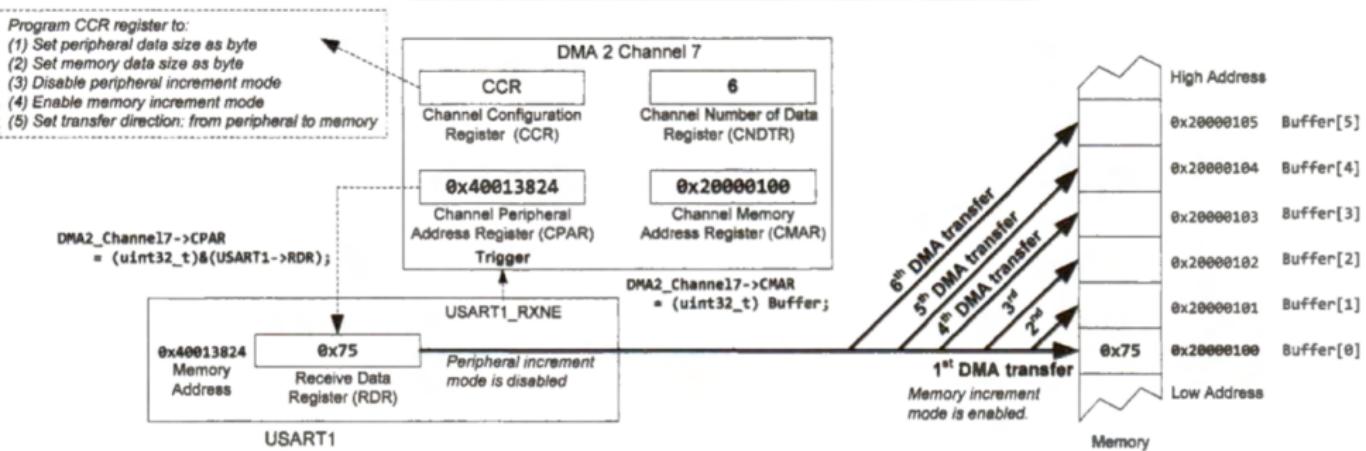
- Cada canle ten 4 rexistros:
 - Channel memory address register (CMAR): inicio da zona de memoria
 - Channel peripheral address register (CPAR): inicio do periférico
 - Channel number of data register (CNDTR): número de datos totais
 - Channel configuration register (CCR): configuración
 - Dirección da transferencia
 - Úsase modo de incremento?
 - Úsase modo circular?
 - Prioridade
 - Habilitar certas interrupcións

Programar DMA



- O modo incremento indica se se incrementan automaticamente os valores en CMAR e CPAR despois de cada transferencia
 - O disparador programase co channel selection register (CSELR) ao escoller a canle
 - Diferentes canles teñen diferentes disparadores, hai que ver o manual
 - O seguinte exemplo programa o USART1 no DMA 2 canle 7

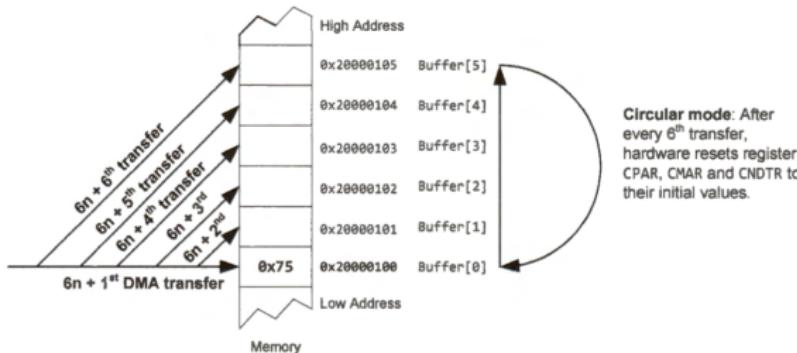
Exemplo: programar DMA



Exemplo: programar DMA

```
RCC->AHB1ENR |= RCC_AHB1ENR_DMA2EN;           // Enable DMA clock
DMA2_Channel17->CCR &= ~DMA_CCR_EN;           // Disable DMA channel
DMA2_Channel17->CCR &= ~DMA_CCR_PSIZE;         // Peripheral data size 00 = 8 bits
DMA2_Channel17->CCR &= ~DMA_CCR_MSIZE;         // Memory data size: 00 = 8 bits
DMA2_Channel17->CCR &= ~DMA_CCR_PINC;          // Disable peripheral increment mode
DMA2_Channel17->CCR |= DMA_CCR_MINC;           // Enable memory increment mode
DMA2_Channel17->CCR &= ~DMA_CCR_DIR;           // Transfer direction: to memory
DMA2_Channel17->CNDTR = 6;                      // Number of data to transfer
DMA2_Channel17->CPAR = (uint32_t)&(USART1->RDR); // Peripheral address
DMA2_Channel17->CMAR = (uint32_t) Buffer;        // Receive buffer address
DMA2_CSELR->CSELR &= ~DMA_CSELR_C6S;          // See Table 19-2
DMA2_CSELR->CSELR |= 2<<24;                  // Map channel 7 to USART1_RX
DMA2_Channel17->CCR |= DMA_CCR_EN;              // Enable DMA channel 7
```

DMA: Modo circular



- Modo normal:
 - En cada transferencia decrementease CNDTR (número de datos totais) en 1
 - Cando chega a 0 acábase
 - CNDTR non se pode modificar durante a transferencia
 - Modo circular
 - Cando CNDTR chega a cero CPAR, CMAR e CNDTR volven aos seus valores orixinais
 - Pode iniciarse unha nova ronda de transferencias
 - Útil para fluxos continuos de datos

DMA: Interrupciones

```
// Enable the transfer complete interrupt
DMA2_Channel17->CCR |= DMA_CCR_TCIE;

// Disable the half transfer interrupt
DMA2_Channel17->CCR &= ~DMA_CCR_HTIE;

// Set the priority as the most urgent
NVIC_SetPriority(DMA2_Channel17 IRQn, 0);

// Enable NVIC interrupt
NVIC_EnableIRQ(DMA2_Channel17 IRQn);
```

- Pódense xerar 3 interrupciones, habilitables por separado
 - Transferencia finalizada (transfer finished (TC))
 - A medio terminar (half-finished (HT))
 - Erro (transfer error (TE))
 - O controlador DMA desactiva a canle automáticamente

Índice

- 1 Introducción
 - 2 GPIO
 - Hardware
 - Programación
 - 3 Temporización
 - Reloxo do Sistema
 - Temporizadores
 - Reloxo de tempo real
 - 4 Acceso Directo a Memoria (e buses)
 - Acceso Directo a Memoria
 - 5 ADC e DAC
 - ADC
 - DAC
 - 6 Protocolos de Comunicación en Serie
 - Protocolos de Comunicación en Serie

Analog to Digital Converter ADC

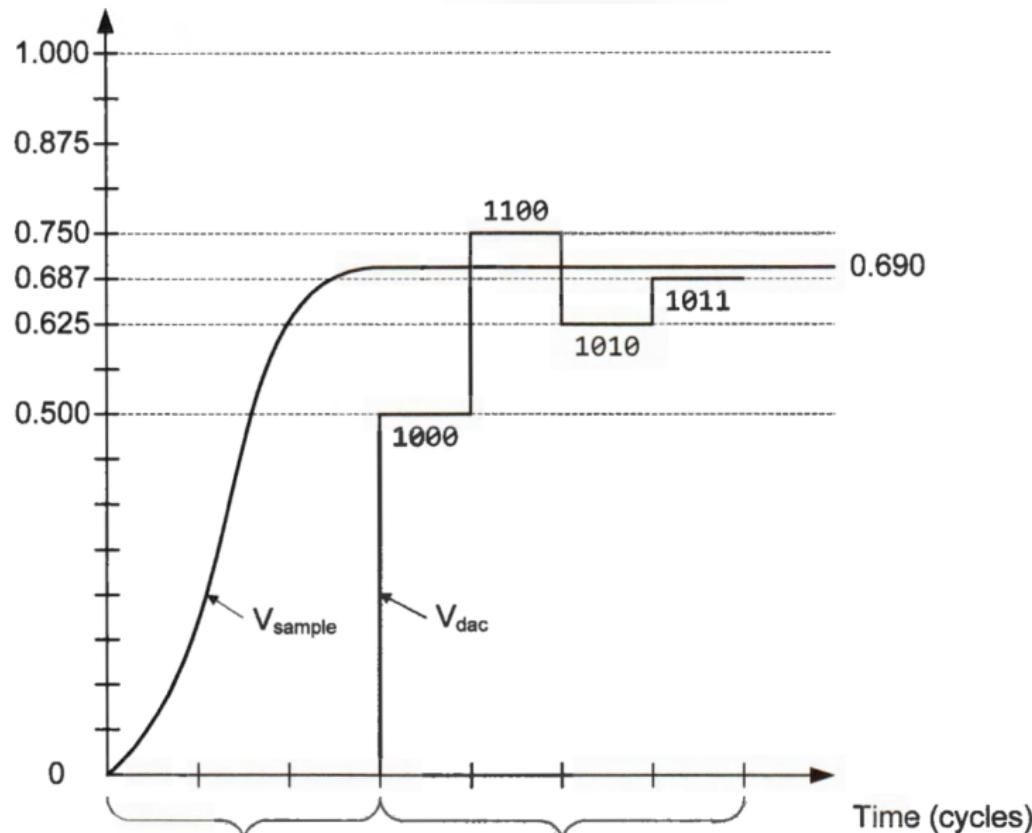
Analog to Digital Converter ADC

- O conversor analóxico a dixital produce un número enteiro de precisión finita para aproximar un voltaxe analóxico de entrada
- Os tres parámetros de rendemento:
 - Ratio de mostraxe (*sampling rate*): Cantas conversíons ADC por segundo. Varios millóns ou miles de millóns por segundo
 - Resolución: o número de bits da saída. Entre 6 e 24 bits. Con 12 ou 24 bits soe chegar
 - Disipación de potencia. Nos sistemas encaixados o rendemento soe estar limitado polo presuposto de enerxía
- Hai varias formas de crear a arquitectura, as STM32L usan aproximacións sucesivas (*successive-approximation (SAR)*)

ADC, aproximacións sucesivas

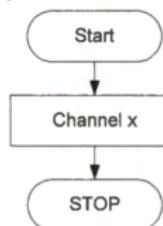
- Usa o DAC (Digital to Analog Converter) para aproximar a sinal de entrada
- Vai cambiando a sinal do DAC aumentando ou reducindo por $1/2^n$
- Así sácanse os n bits ata que se acabe a resolución

ADC, aproximacións sucesivas



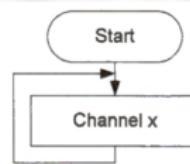
ADC

O ADC pode ter 1 ou varias canles de entrada, as converte por quenda. Pode funcionar en modo continuo, volvendo a empezar



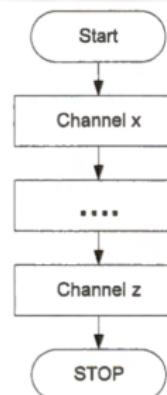
Single Channel, Single Conversion

CONT in ADC_CFGR = 0



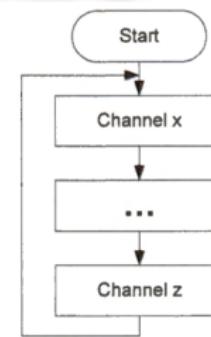
Single Channel, Continuous Conversion

CONT in ADC_CFGR = 1



Scan with Single Conversion

CONT in ADC_CFGR = 0



Scan with Continuous Conversion

CONT in ADC_CFGR = 1

Regular channel:

1. Set bit ADSTART in register ADC_CR
2. Channel is selected by SQ1[4:0] in SQR1
3. Result stored in ADC_DR
4. EOC is set after conversion
5. Interrupt is generated if EOcie is set

Injected channel:

1. Set JADSTART in register ADC_CR
2. Channel is selected by JSQ1[4:0] in JSQR
3. Result is stored in ADC_JDR1
4. JEOC is set after conversion
5. Interrupt is generated if JEOCIE is set

1. Channels are selected by ADC_SQRx registers for regular channels, and by ADC_JSQR register for injected channels
2. All channels in a regular group share the same result register ADC_DR. Make sure to read data between consecutive sampling.

ADC, disparadores

- O ADC pode dispararse por software
- Pode dispararse por eventos externos
 - Temporizadores
 - Entradas por Pin
 - Escoller flanco de subida, baixada...

ADC e DMA

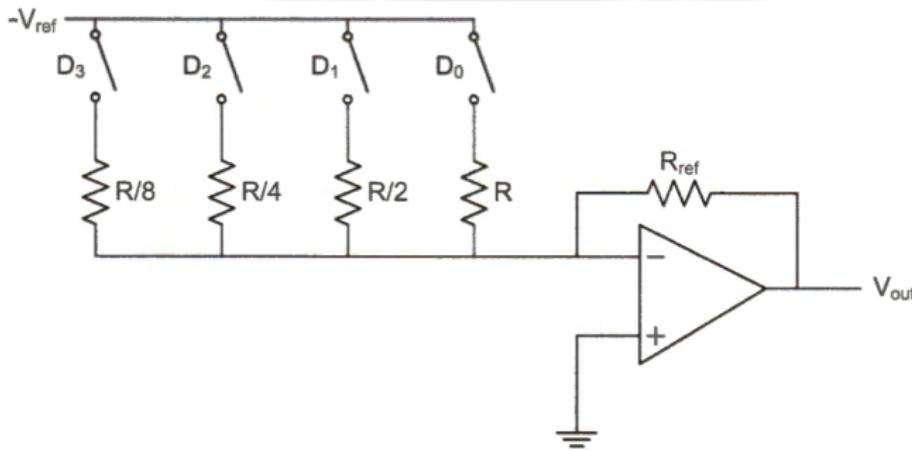
- O ADC sóese usar con DMA
- Con DMA circular pódese reutilizar o buffer para sinais continuas
- Pódense usar dous buffers DMA:
 - O DMA escribe nun deles (escritura)
 - A CPU procesa o outro (lectura)
 - Tras cada transferencia DMA intercámbianse os buffers lectura/escritura
 - Así solapase o procesamento

Digital to Analog Converter DAC

Digital to Analog Converter DAC

- O conversor digital a analóxico produce un voltaxe analóxico de saída a partires dun número enteiro de precisión finita
- Os tres parámetros de rendemento:
 - Resolución: É o menor cambio na voltaxe que se pode producir (en V). Para n bits os posibles niveis de saída son 2^n . Como é directamente proporcional sóese medir en bits por simplicidade
 - Para rango de 0-5V e 8 bits: $5/2^8 = 0,0195V = 19,5mV$
 - Tempo de estabilización: Tempo que tarda en estabilizarse a sinal desde que se cambia (dentro de certo porcentaxe, como 0.025 % para 12 bit)
 - *glitch* (sobrepico): É o primeiro pico transitorio que aparece ao crear a sinal. Idealmente debería subir monotónicamente, pero non soe pasar

DAC



$$V_{out} = V_{ref} \times \left(\frac{D_3}{R/8} + \frac{D_2}{R/4} + \frac{D_1}{R/2} + \frac{D_0}{R} \right) \times R_{ref}$$

$$V_{out} = V_{ref} \times \frac{R_{ref}}{R} \times (D_3 \times 2^3 + D_2 \times 2^2 + D_1 \times 2 + D_0)$$

DAC

- As STM32 teñen dous DAC que poden funcionar sincronizarse entre eles
- Poden usar disparadores software, temporizadores internos e externos
- Para evitar usar punto flotante úsanse táboas de lookup
 - Gardas o resultado dunha operación (como $\sin(x)$) nunha táboa
 - Usa máis memoria e ten menos precisión pero é mais rápido

Índice

- 1 Introducción
- 2 GPIO
 - Hardware
 - Programación
- 3 Temporización
 - Reloxo do Sistema
 - Temporizadores
 - Reloxo de tempo real
- 4 Acceso Directo a Memoria (e buses)
 - Acceso Directo a Memoria
- 5 ADC e DAC
 - ADC
 - DAC
- 6 Protocolos de Comunicación en Serie
 - Protocolos de Comunicación en Serie

Protocolos de Comunicación en Serie

Comunicación en Serie

- Transmiten 1 bit de cada vez
 - Poden usar 1 so cable ou 1 por dirección (2 cables)
 - A maiores poden necesitar cables de Vcc, terra, reloxo...
 - En principio más lentas que en paralelo, más baratas
 - Pero hoxe en día tan rápidas que compensa
 - USB, Serial ATA ... case todo é xa en serie
 - Half-duplex
 - Úsase 1 so cable e se transmite nas dúas direccións, non simultaneamente
 - Full-duplex
 - Transmitese nas dúas direccións simultaneamente, con 2 cables, frecuencias separadas...

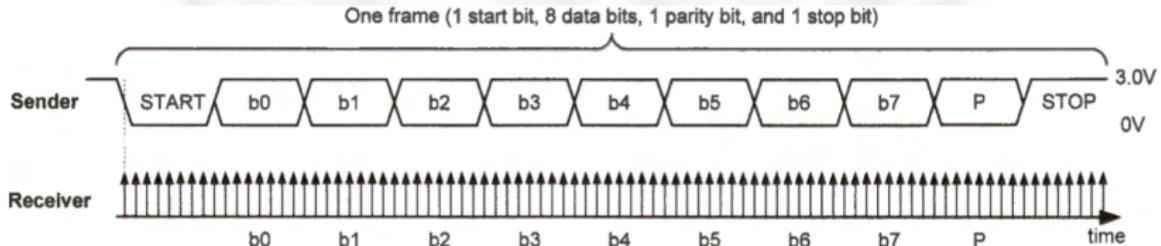
Universal Asynchronous Receiver and Transmitter

Universal Asynchronous Receiver and Transmitter UART

- Comunicación asíncrona, pero pode configurarse como síncrona (USART)
- Universal implica que é programable
- Ambos extremos pónense de acordo na taxa de transmisión (bitrate, bits por segundo)
- Liñas TX (transmite) e RX (recibe)
 - En half-duplex so TX
- Liñas CTS (clear to send) e RST (request to send) para o control
- Marco de comunicación (communication frame)
 - Os datos divídense para envialos
 - Lonxitude dos datos (7,8,9 bits)
 - Bits de parada (0.5, 1, 1.5, 2 bits)
 - Bit de paridade (par, impar, sen paridade)

Universal Asynchronous Receiver and Transmitter

- Bit de inicio: voltaxe baixo. Bit de final: voltaxe alto
 - Paridade: indica se o número de 1 é par ou impar (configurable)
 - Faise sobresampleo para asegurarse ler ben (pode baixarse no modo síncrono)
 - Orden de bits: LSB ou MSB



Universal Asynchronous Receiver and Transmitter

- Poden usarse diferentes métodos para comunicarse
- Sondaxe (polling):
 - Pouco eficiente, a non ser que haxa comunicación continua
- Interrupciós:
 - Poden ser varias. Tan só como exemplo:
 - transmission data register empty (TxE)
 - transmission complete (TC)
 - received data register not empty (RXNE)
 - overrun error detected (ORE)
 - idle line detected (IDLE)
 - parity error (PE)
- DMA
 - Soe ser o más eficiente, combinado con interrupciós

Inter-Integrated Circuit

Inter-Integrated Circuit I^2C

- Protocolo de bus para comunicar procesador e os seus periféricos
- 2 cables, reloxo (serial clock line, SCL) e datos (serial data line, SDA)
- Mestre- escravo
 - Pode haber varios mestres
 - Tanto os mestres como os escravos teñen unha dirección
- O mestre inicia a comunicación:
 - Bit de inicio (start S) e final (stop P): transicións baixo-alto
 - Xera a sinal de reloxo
 - Todos os escravos leen, mandan ACK ou NACK (aviso de recibo)
 - A CLK está en *open-drain*, fai un AND hardware
 - Así sincronizase o reloxo e detéctanse colisiones

Inter-Integrated Circuit

- Envíase a dirección da transferencia (ler ou escribir) e a dirección do periférico
- Se a dirección é moi grande envíase por partes
- Pode haber varios *start S* antes do *stop P*, así non se libera o bus
- Pode usar Sondaxe ou DMA

S	Slave Address	R/ \bar{W}	A	Data	A	Data	A	P
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

S	Slave Address (higher 2 bits)	R/ \bar{W}	A1	Slave Address (lower 8 bits)	A2	S	Slave Address (higher 2 bits)	R/ \bar{W}	A3	Data	A	Data	A	P
1 bit	7 bits (11110xx)	1 bit (0)	1 bit	8 bits	1 bit	1 bit	7 bits (11110xx)	1 bit (0)	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

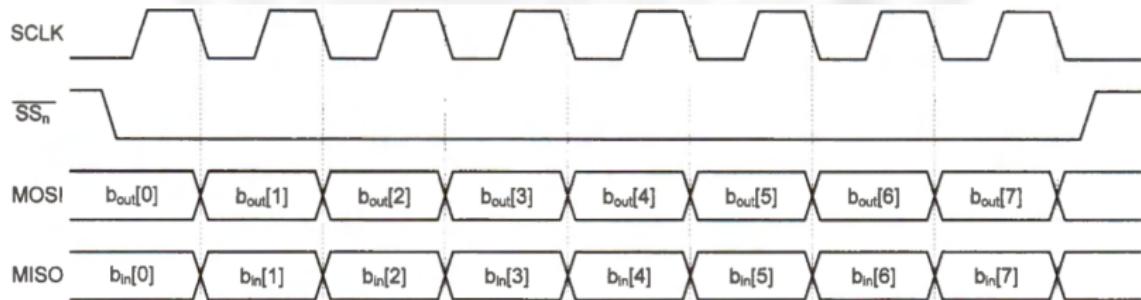
Serial Peripheral Interface Bus

Serial Peripheral Interface SPI

- Protocolo de bus para comunicar procesador e periféricos en xeral, síncrono
- Usa 4 cables
 - master-in-slave-out data line (MISO)
 - master-out- slave-in data line (MOSI)
 - serial clock line (SCLK)
 - slave select line (SS)
 - Unha SS diferente para cada periférico, o resto compartidas
- So pode haber 1 mestre
 - Os escravos non poden iniciar comunicación ou variar a velocidade

Serial Peripheral Interface Bus

- Cada ciclo de reloxo 1 dato sae pola MISO e outro entra pola MOSI
- Cando se escribe nun flanco (baixada ou subida) lese no oposto do mesmo ciclo (subida ou baixada)
- Sempre full-duplex, pódese ter que enviar datos inútiles (*dummy data*)

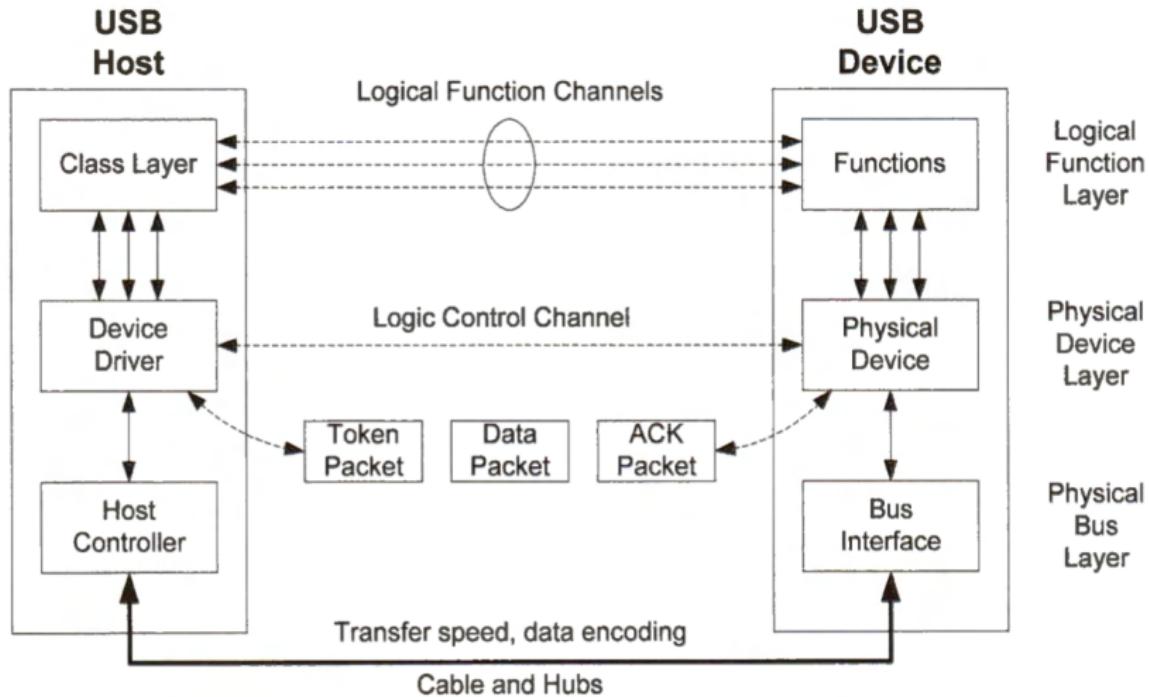


Universal Serial Bus

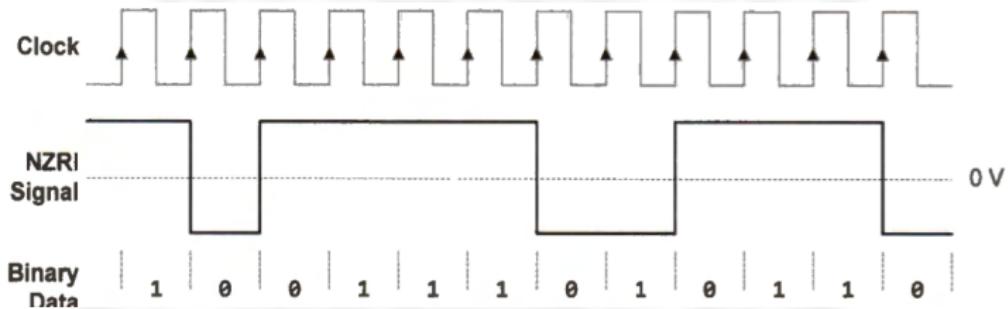
Universal Serial Bus USB

- Para conectar múltiples periféricos a un host
- Moitas vantaxes (plug-and-play, da enerxía, baixo custo e consumo...)
- 4 cables (voltaxe, terra e par trenzado)
- Protocolo en capas:
 - Bus: conexión dos cables, transmisión dos datos, codificación
 - Device (aparello): control, configuración, detección de errores, segmentación
 - Lógica: Funcións de alto nivel
- Bastante complejo, máis da materia de *Redes e Comunicacións*

Universal Serial Bus



Universal Serial Bus



Referencias

Imaxes desde:

Y. Zhu. "Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C". Third Edition. E-Man Press LLC. 2017. ISBN-13: 978-0982692660