

# Curso de programación en C para AVR8

Ing. Damian Corbalan  
Ing. Leandro Lanzieri  
LABFRA - UTN - FRA

Sábados  
9:30 a 12:30 Hs

# Temario

- Arquitectura RISC
- Caract. microcontrolador Atmega128
- Puertos en lenguaje C
- Interrupciones externas y Timers
- Display 7 segmentos y LCD
- Conversor Analogico/Digital
- Comparador analógico
- Comunicaciones serie

- ❑ Interfaz UART.
- ❑ Comunicación con la PC
- ❑ Interfaz I2C. Uso del RTC DS1307
- ❑ Interfaz SPI. Memoria SD.
- ❑ Sistema de archivos sobre memoria SD
- ❑ Proyectos

# Que es un Microcontrolador?

Hay de diferentes formas.....



ESTE VAMOS A USAR



Pero.....

QUE ES  
UN  
MICRO?

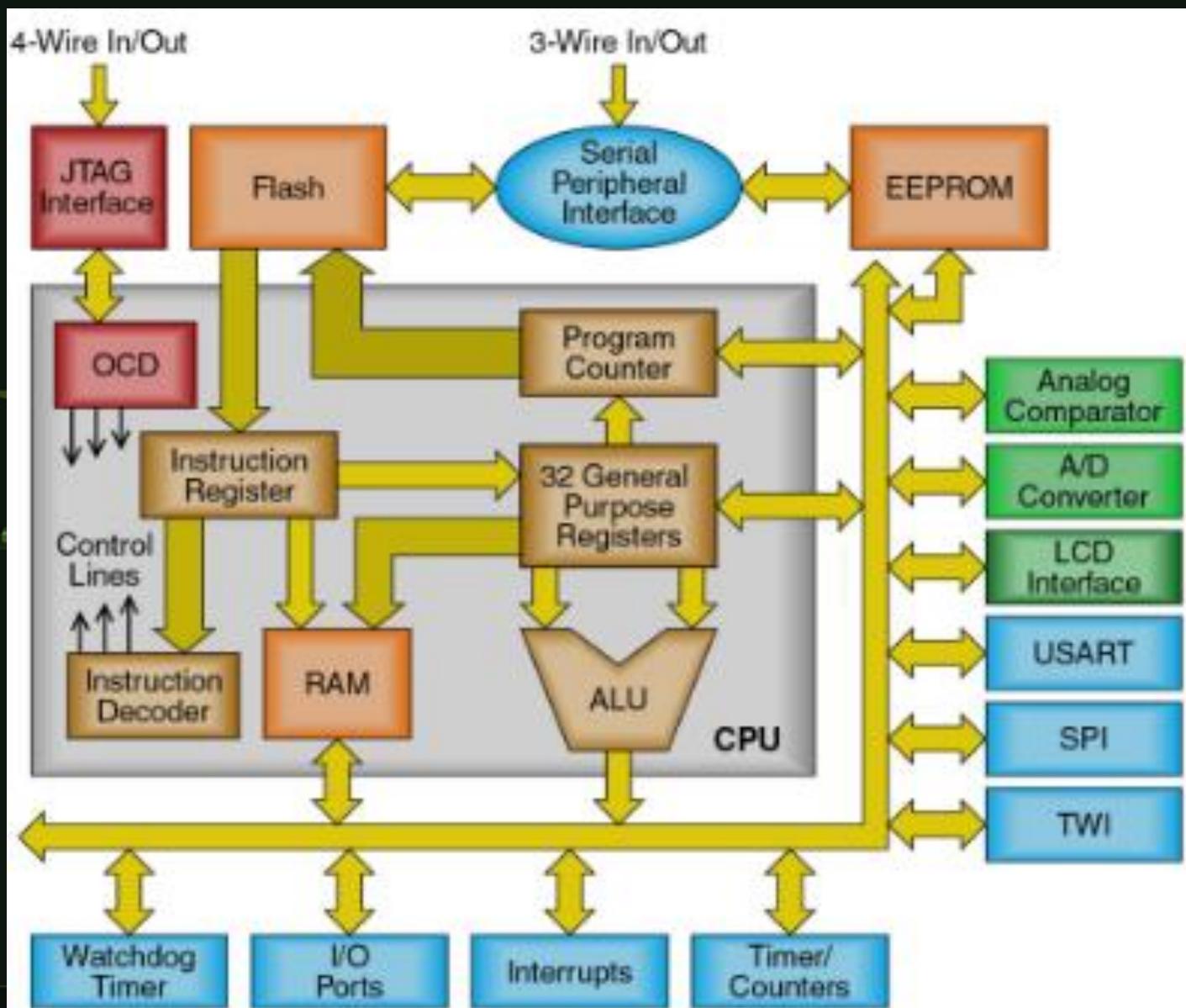
# Que es un Microcontrolador?

Es un circuito integrado que tiene:

- ❖ Procesador
- ❖ Memorias
- ❖ Periféricos
- ❖ Registros
- ❖ Unidad aritmética lógica
- ❖ Buses de conexión

A estos les doy diferentes funciones  
cargando un programa en su memoria.

# Que tiene un Microcontrolador?



# Memorias

- ❖ EPROM\*
- ❖ EEPROM
- ❖ FLASH
- ❖ RAM
- ❖ NVRAM





**Periféricos**  
**Puertos I/O**  
**Timers**  
**Interrupciones**  
**Interfaces de Comunicacion**  
**ADC**  
**DAC**



La cantidad, tipo, tamaño de periféricos y memorias depende del fabricante.

Como elijo  
el micro  
indicado  
entonces

.....

# Los fabricantes ofrecen cuadros comparativos

Device Name	Flash (Kbytes)	Pin Count	Max. Operating Frequency	CPU	# of Touch Channels	Max I/O Pins	Ext Interrupts	SPI	TWI (I2C)	UART	ADC channels	ADC Resolution (bits)	Timers	Output Compare channels	Input Capture Channels	PWM Channels	32kHz RTC	Calibrated RC Oscillator	Packages
ATmega128	128	64	16	8-bit AVR	16	53	8	1	1	2	8	10	4	8	2	7	Yes	Yes	MLF (VQFN) 64M1 64,TQFP 64A 64
ATmega1280	128	100	16	8-bit AVR	16	86	32	5	1	4	16	10	6	16	4	15	Yes	Yes	CBGA 100C1 100,TQFP 100A 100
ATmega16	16	44	16	8-bit AVR	16	32	3	1	1	1	8	10	3	4	1	4	Yes	Yes	MLF (VQFN) 44M1 44,PDIP 40P6 40,TQFP 44A 44
ATmega32	32	44	16	8-bit AVR	16	32	3	1	1	1	8	10	3	4	1	4	Yes	Yes	MLF (VQFN) 44M1 44,PDIP 40P6 40,TQFP 44A 44
ATmega48	4	32	20	8-bit AVR	12	23	24	2	1	1	8	10	3	6	1	6	Yes	Yes	MLF (VQFN) 28M1 28,MLF (VQFN) 32M1-A 32,PDIP 28P3 28,TQFP 32A 32
ATmega8A	8	32	16	8-bit AVR	12	23	2	1	1	1	8	10	3	-	-	3	Yes	Yes	MLF (VQFN) 32M1-A 32,PDIP 28P3 28,TQFP 32A 32

# Arquitectura RISC

Reduced COMPLEXITY Instruction Set Computer.

No significa un número de instrucciones reducido.

Sino que se refiere a la complejidad de los circuitos encargados de la decodificación de las instrucciones , haciéndolos más simples y eficientes.



Firma ATTEL

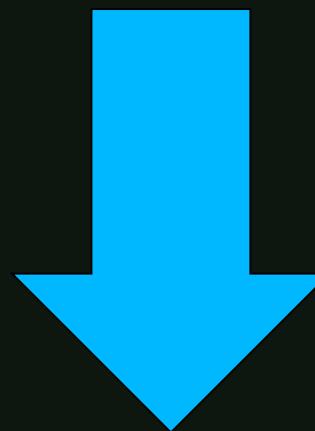
# Arquitectura RISC

- Usa instrucciones de tamaño fijo: 4 instrucciones de 32 bits y el resto son de 16 bits
  - Posee 32 registros de uso general
  - Solo utiliza instrucciones de Load/Store para acceder a la memoria RAM.
  - La mayoría de las instrucciones se ejecutan en 1 ciclo de clock.
  - Esta diseñada especialmente para la programación en C.
  - Posee muy buenas características de bajo consumo

# Arquitectura del bus de comunicaciones

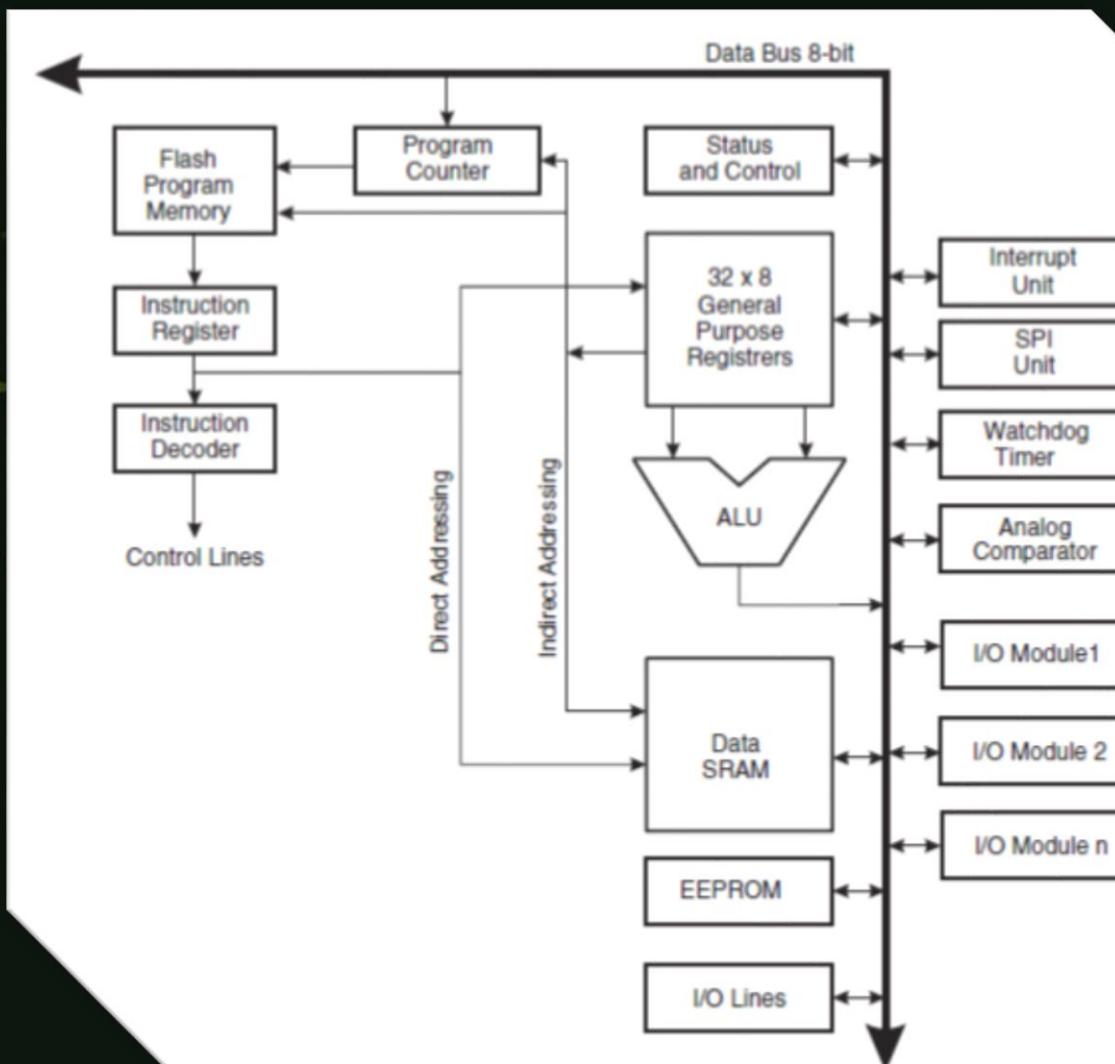
ARQUITECTURA	VON NEUMANN	HARVARD
VELOCIDAD	-A pesar de ser la mas utilizada en nuestros días es mas lenta que la arquitectura de Harvard, esto se debe a su flexibilidad para el uso de diferentes tipos de programas.	-El tener una memoria de programa y una memoria de datos la hace una arquitectura mucho mas estable y con mas velocidad aunque no sea tan utilizada.
USOS DE LA MEMORIA	-Las instrucciones y los datos se almacenan en cachés separadas para mejorar el rendimiento.	-Se utiliza una sola caché para datos e instrucciones (programas), lo cual merma el desempeño.
ES UTILIZADA EN	-Esta arquitectura es la variante adecuada para las PC. Ya que es lenta pero flexible, adaptable y modificable en ciertos casos.	-Por excelencia la utilizada en supercomputadoras, en los microcontroladores, y sistemas embebidos en general.
COSTOS	-Esta arquitectura es la variante adecuada para las PC, porque permite ahorrar una buena cantidad de líneas de E/S, que son bastante costosas, sobre todo para aquellos sistemas como las PC, donde el procesador se monta en algún tipo de socket alojado en una placa madre .	-Consumo muchas líneas de E/S del procesador, por lo que en sistemas donde el procesador está ubicado en su propio encapsulado, solo se utiliza en supercomputadoras.

# AVR



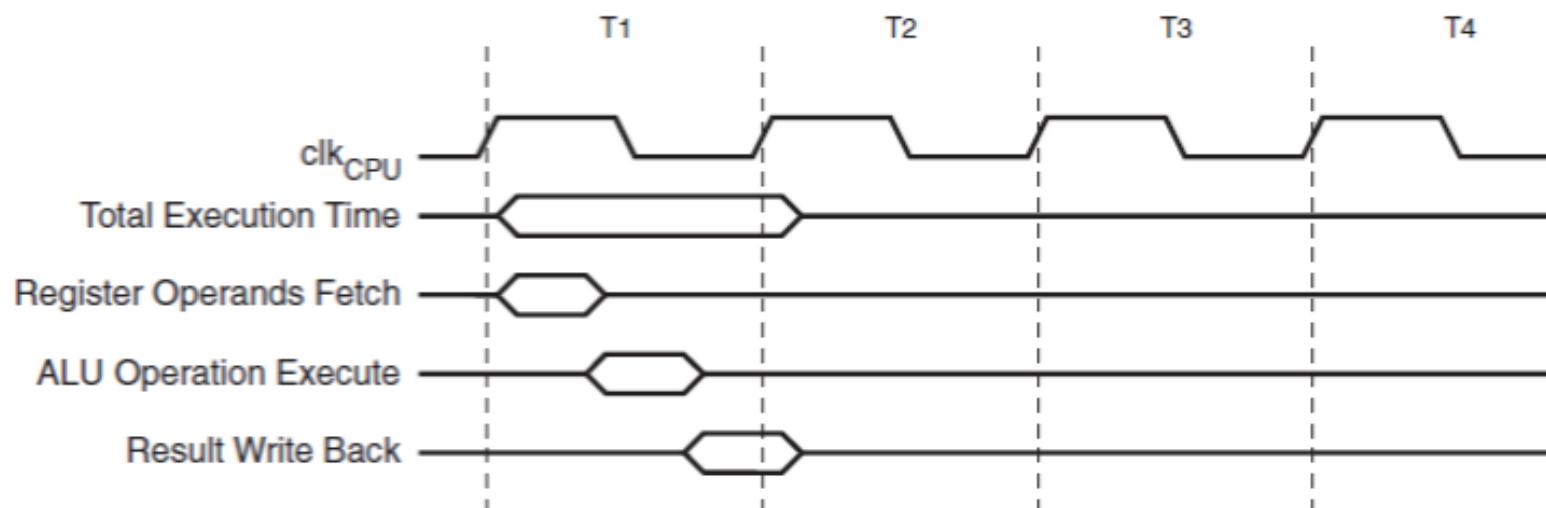
## Arquitectura Harvard

# AVR Core



# AVR Core

En un ciclo de reloj se pueden leer 2 registros que funcionen como operandos para la ALU, realizar la operación y que el resultado quede disponible para escribirse en uno de esos registros.





**ATMEGA 128**

# C A R A C T E R I S T I C A S

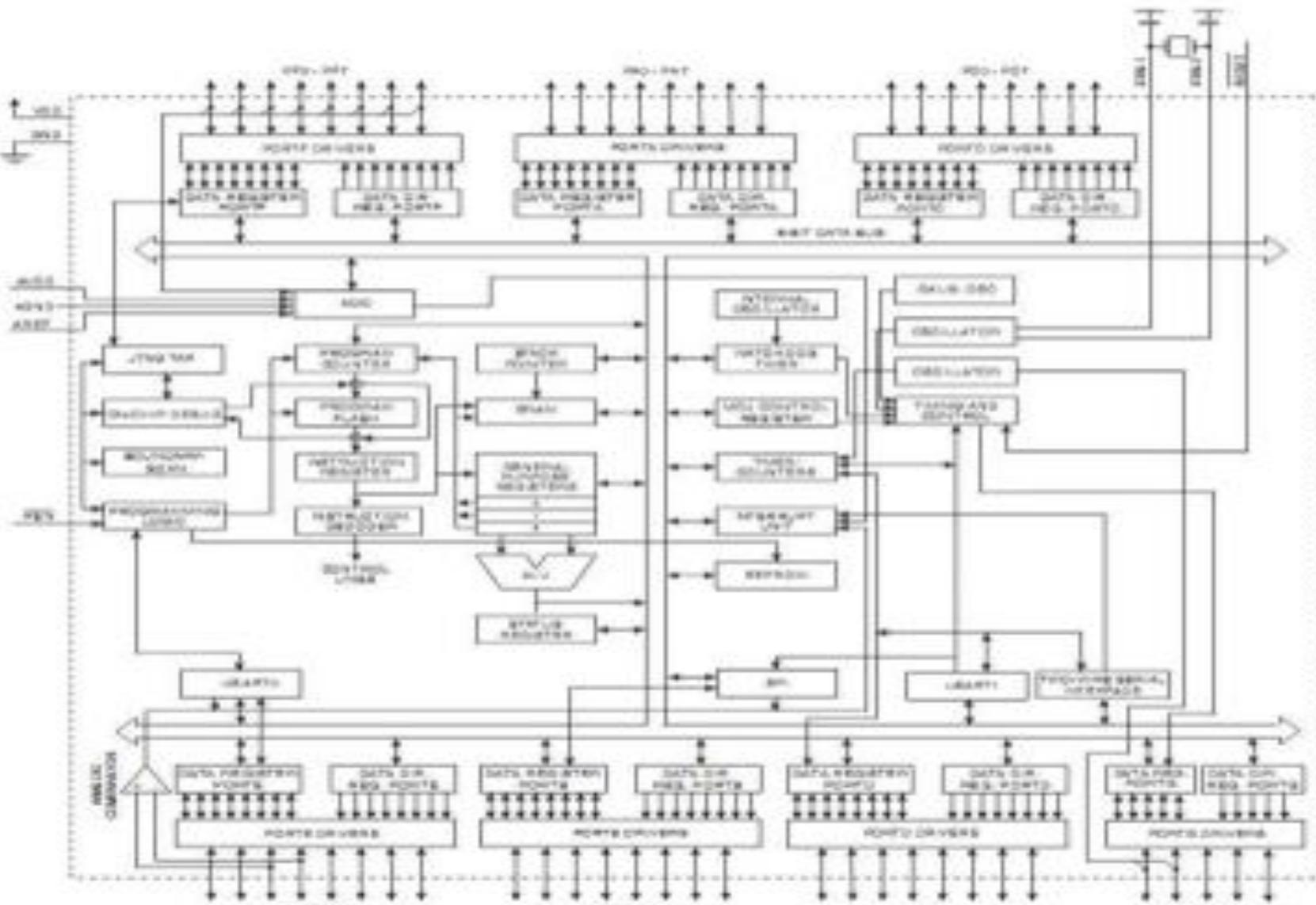
- 133 instrucciones
- 128Kbytes de FLASH
- 4Kbytes de EEPROM
- 4Kbytes de SRAM
- Interface JTAG
- 2 Timer/Counter de 8 bits
- 2 Timer/Counter de 16 bits
- 1 Contador de tiempo real
- 2 canales PWM de 8 bits
- 6 canales PWM con resolución prog.
- ADC de 10bits de 8 canales
- Interfaz serial a 2 hilos TWI (I2C)
- 2 USART programables
- Interfaz SPI
- Comparador analógico integrado
- 53 entradas/salidas programables
- Encapsulado de 64 pines TQFP

10	PB0 (SS)	PA0 (AD0)	51
11	PB1 (SCK)	PA1 (AD1)	50
12	PB2 (MOSI)	PA2 (AD2)	49
13	PB3 (MISO)	PA3 (AD3)	48
14	PB4 (OC0)	PA4 (AD4)	47
15	PB5 (OC1A)	PA5 (AD5)	46
16	PB6 (OC1B)	PA6 (AD6)	45
17	PB7 (OC2OC1C)	PA7 (AD7)	44
25	PD0 (SCLINT0)	PC0 (A8)	35
26	PD1 (SDAINT1)	PC1 (A9)	36
27	PD2 (RXD1INT2)	PC2 (A10)	37
28	PD3 (TXD1INT3)	PC3 (A11)	38
29	PD4 (IC1)	PC4 (A12)	39
30	PD5 (XCK1)	PC5 (A13)	40
31	PD6 (T1)	PC6 (A14)	41
32	PD7 (T2)	PC7 (A15)	42
2	PE0 (RXD0(PDI))	PF0 (ADC0)	61
3	PE1 (TXD0PDO)	PF1 (ADC1)	60
4	PE2 (XCK0AIN0)	PF2 (ADC2)	59
5	PE3 (OC3AAIN1)	PF3 (ADC3)	58
6	PE4 (OC3BINT4)	PF4 (ADC4TCK)	57
7	PE5 (OC3CINT5)	PF5 (ADC5TMS)	56
8	PE6 (T3INT6)	PF6 (ADC6TDO)	55
9	PE7 (IC3INT7)	PF7 (ADC7TDI)	54
33	PG0 (WR)	VCC	52
34	PG1 (RD)	VCC	21
43	PG2 (ALE)	AVCC	64
18	PG3TOSC2	AREF	62
19	PG4TOSC1		
20	RESET	GND	22
1	PEN	GND	53
23	XTAL2	GND	63
24	XTAL1	DAP	65

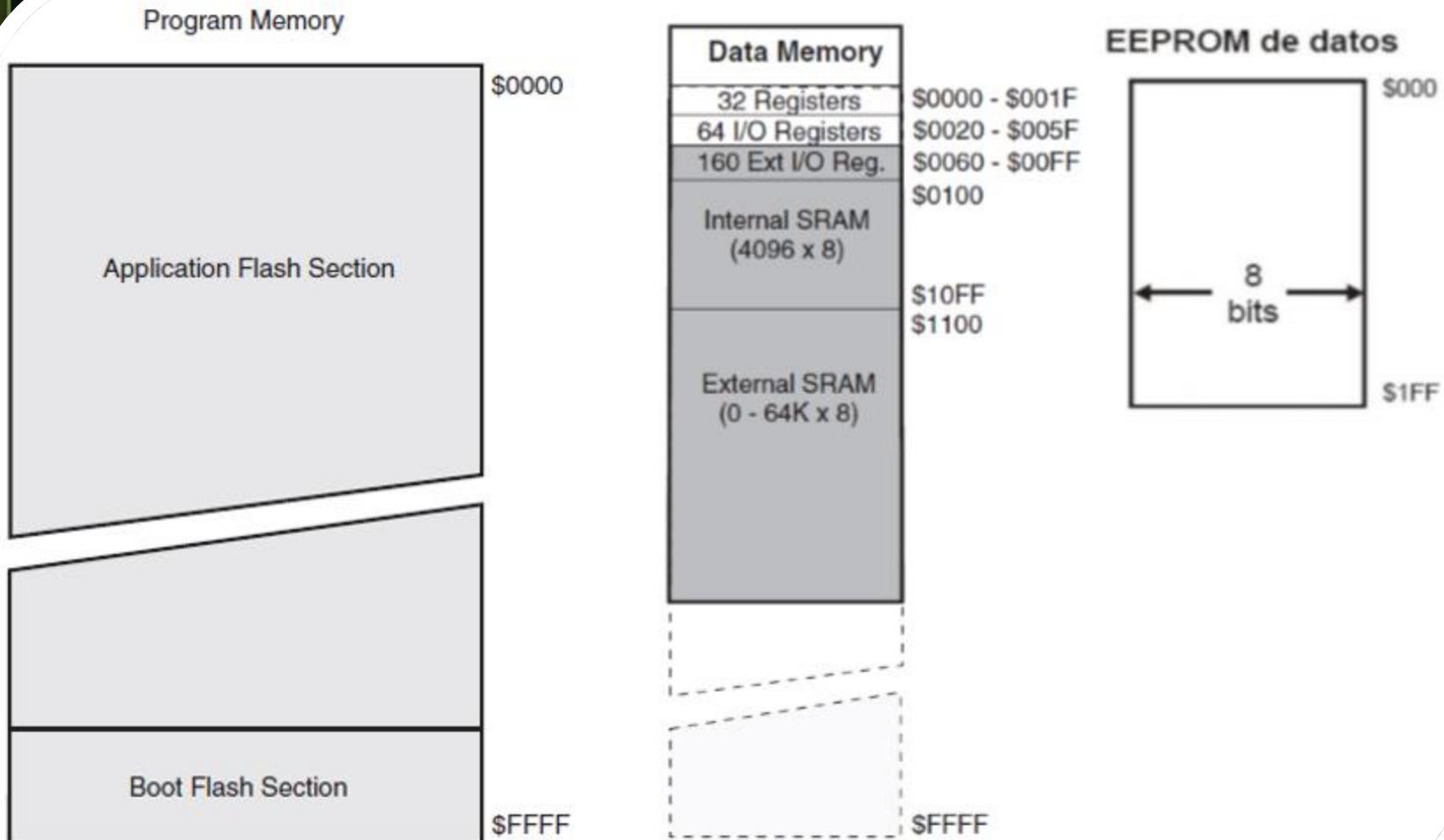
# PINOUT

	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	
PEN																	
RXD0(PDI) PE0																	48 PA3 (AD3)
(TXD0/PDO) PE1																	47 PA4 (AD4)
(XCK0/AIN0) PE2																	46 PA5 (AD5)
(OC3A/AIN1) PE3																	45 PA6 (AD6)
(OC3B/INT4) PE4																	44 PA7 (AD7)
(OC3C/INT5) PE5																	43 PG2(ALE)
(T3/INT6) PE6																	42 PC7 (A15)
(ICP3/INT7) PE7																	41 PC6 (A14)
(SS) PB0																	40 PC5 (A13)
(SCK) PB1																	39 PC4 (A12)
(MOSI) PB2																	38 PC3 (A11)
(MISO) PB3																	37 PC2 (A10)
(OC0) PB4																	36 PC1 (A9)
(OC1A) PB5																	35 PC0 (A8)
(OC1B) PB6																	34 PG1 (RD)
	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33 PG0(WR)
(OC2)/OC1(C) PB7																	
TOSC2/PD3																	
TOSC1/PD4																	
RESET																	
VCC																	
GND																	
XTAL2																	
XTAL1																	
(SCL/INT6) PD0																	
(SDA/INT1) PD1																	
(RXD1/INT2) PD2																	
(TXD1/INT3) PD3																	
(I2C1) PD4																	
(OC2) PD5																	
(T1) PD6																	
(T2) PD7																	

# Diagrama en Bloques



# Mapa de la memoria



# Registros de Propósito General

General  
Purpose  
Working  
Registers

	7	0	Addr.	
R0			\$00	
R1			\$01	
R2			\$02	
...				
R13			\$0D	
R14			\$0E	
R15			\$0F	
R16			\$10	
R17			\$11	
...				
R26			\$1A	X-register Low Byte
R27			\$1B	X-register High Byte
R28			\$1C	Y-register Low Byte
R29			\$1D	Y-register High Byte
R30			\$1E	Z-register Low Byte
R31			\$1F	Z-register High Byte

# ALU - Registro de Estado

La ALU opera en conexión directa con los 32 registros de propósito general. Puede realizar operaciones aritméticas, lógicas y funciones de bit.

El registro de estado contiene información sobre los resultados de la más reciente operación de la ALU.

# Programación

Con el set de instrucciones podemos escribir un programa en lenguaje Assembler, ensamblarlo y cargarlo en la memoria. Así, para usar un puerto por ejemplo podemos escribir un código como el siguiente:

**LDI R4, 0x01 OUT PORTA,R4**

Así ponemos en 1 un bit del puerto A



```
c, state;
nl, nw, nc, state;
OUT;
nc = 0;
c = getchar() != EOF ? nc;
if (c == '\n')
    ++nl;
if (c == ' ' || c == '\r')
    state = OUT;
else if (state == OUT)
    state = IN;
++nw;
printf("%d %d %d\n", nl, nw, nc);
}
}

#include <stdio.h>

#define IN /* inside a word */
#define OUT /* outside a word */
words, and characters in input */
```

Con el compilador gcc nuestro programa se traducirá al assembler del micro AVR y luego en el archivo programable que se graba en el micro.



# Empezando a trabajar



# Herramientas



# Pantalla del Eclipse

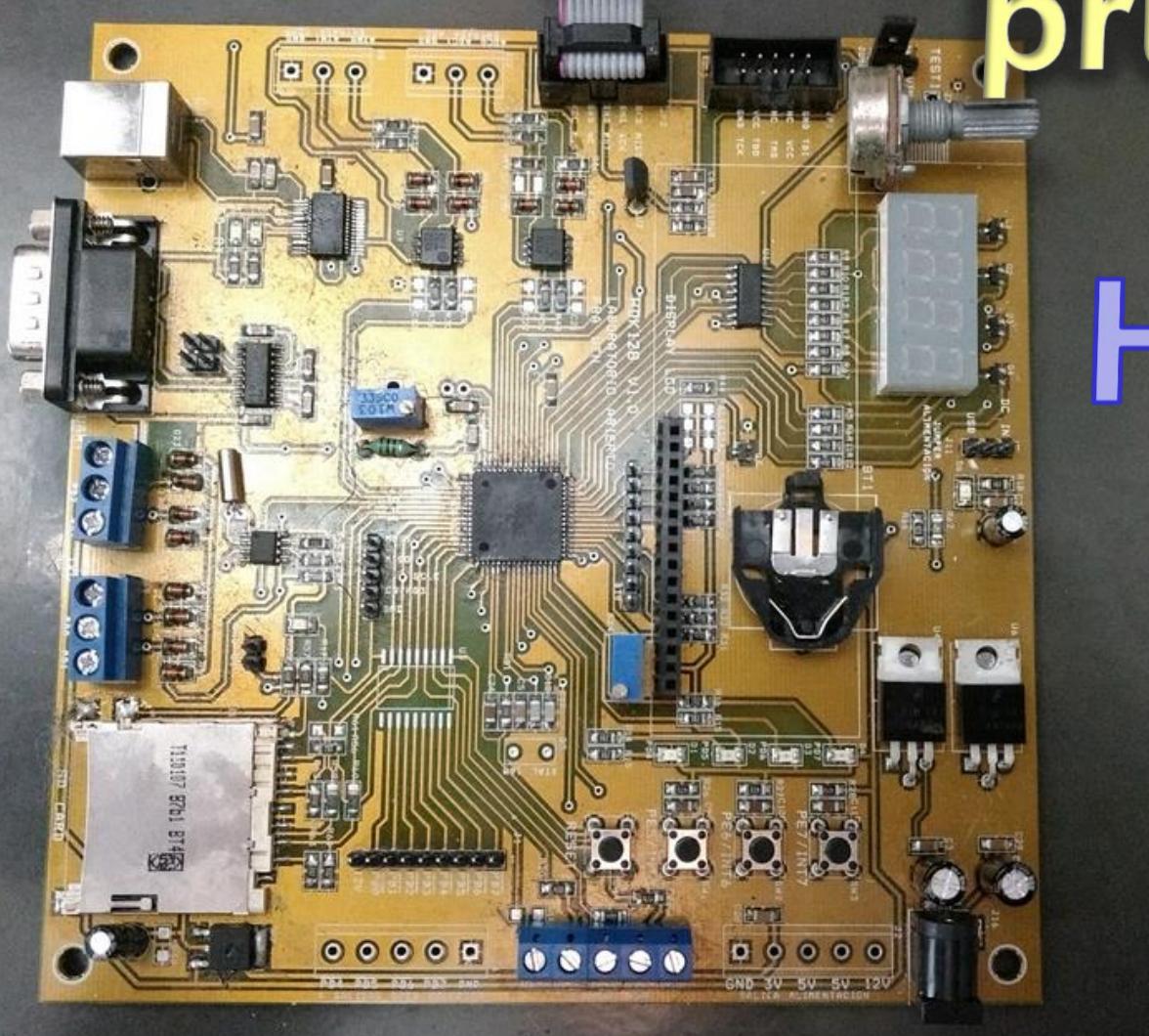
The screenshot shows the Eclipse C/C++ IDE interface. The title bar reads "C/C++ - Prueba\_HDK128/main.c - Eclipse". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, AVR, Run, Window, and Help. The toolbar has various icons for file operations. The Project Explorer view on the left shows a project named "Prueba\_HDK128" with files like main.c, avr\_adc\_api.h, and DS1307.c. The Editor view in the center displays the main.c source code. The Outline view on the right lists the project's header files and a main function. The bottom navigation bar includes Problems, Tasks, Console, Properties, AVR Device Explorer, AVR Supported MCUs, and Search. A status bar at the bottom indicates "0 items selected".

```
#include <util/delay.h>
#include "HDK128.h"
#include "lcd_alfa.h"
#include "adc_settings.h"
#include "DS1307.h"
#include "TWI_master.h"
#include <stdio.h>

int main(void)
{
    unsigned char valor;
    unsigned char hora, min, seg;
    unsigned char dia, mes;
    unsigned anio;
    char cadena_hora[]="18:06:00";
    char cadena_fecha[]="30/12/08";
    char temperatura[] = "20";
```

# Placa de pruebas

HDK128



# Características HDK128

- 4 Leds conectados a un puerto.
- 3 Pulsadores capaces de producir interrupciones.
- Salidas Open Colector a través ULN2803
- 1 Display 7 segmentos de 4 dígitos
- 1 Display LCD 16x2 con backlight
- Sensor de temperatura LM35 conectado al ADC
- Potenciómetro conectado al ADC
- Reloj de tiempo real I2C
- Adaptador USB-Serie TTL
- Adptador RS-232 a TTL
- Memoria SD conectada por SPI
- Además hay conexiones que permite agregar nuevos periféricos

COMPILANDO!

# Lenguaje C

/\* Comentarios en  
varias líneas \*/

// Comentarios en línea simple

# Lenguaje C

## Includes:

Incluye un archivo de cabecera o header, que contiene declaraciones de símbolos y/o macros

**#include <avr/io.h>** Header del sistema.  
**#include “mis\_definiciones.h”** Un header específico de nuestro proyecto, debe estar en la misma carpeta.

# Lenguaje C

## Definiciones:

Sirve para definir un símbolo como una cadena de texto. Cada vez que el preprocesador de C encuentra CANTIDAD la reemplaza por 100. Esto lo hace en tiempo de compilación. También sirve para definir MACROS

**#define CANTIDAD 100**

# Lenguaje C

## Variables:

Cuando se declara una variable se especifica:

- El **tipo**: char, int, float, etc.
- El **ambiente**: local, global, externa.
- El **tiempo de vida**: estático o dinámico.
- El **nombre** de la variable.

**unsigned char contador;**

# Lenguaje C

## Tipos de Variables

Tipo	Tamaño en bits	Rango
Char	8	-128 127
Unsigned char	8	255
signed char	8	-128 127
Int	16	-32768 32767
short int	16	-32768 32767
Unsigned int	16	65535
signed int	16	-32768 32767
long int	32	-2147483648 2147483647
Unsigned long int	32	4294967295
signed long int	32	-2147483648 2147483647
Float	32	$\pm 1.75 \times 10^{-38}$ $\pm 3.402 \times 10^{38}$
Double	32	$\pm 1.75 \times 10^{-38}$ $\pm 3.402 \times 10^{38}$



# Lenguaje C

## Tipos de Datos

Variable o constante	Formato
Decimal	Número
Hexadecimal	0x número hexadecimal
Binario	0b número binario
Octal	0 número octal
Carácter	'a'
Cadena	"esta es una cadena"

# Lenguaje C

## Arrays:

Es un conjunto de datos al que se puede acceder mediante un índice:

```
char datos[10];  
unsigned char vector [] = {0, 1, 0};
```

El primer elemento siempre es el índice 0.

```
x = vector[0];
```

# Lenguaje C

## Operadores aritméticos:

Símbolo	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
%	División Módulo, y el resultado es el residuo

# Lenguaje C

## Operadores de relación:

Operador	Descripción
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor Igual que
==	Igual que
!=	Distinto de
&&	Y también si
	O si

# Lenguaje C

## Operadores a nivel de bits:

Símbolo	Descripción
&	And Bit a Bit
	OR bit a Bit
^	Or exclusivo Bit a Bit
<<	Corrimiento a la Izquierda
>>	Corrimiento a la derecha
~	Complemento a unos (inversión de bits)

# Lenguaje C

## Secuencias de control:

- **if ();**
- **if() else();**
- **while();**
- **do{} while();**
- **for( ; ; );**
- **switch () case: break;**

# Lenguaje C

## Función main:

Todo programa C necesita una función llamada main() :

```
int main (void) {  
    char variable_local;  
    ....  
    return 0;  
}
```

# Puertos de Entrada y Salida

El Atmega128 tiene 53 líneas de entrada/salida organizadas en 7 puertos, 6 de ellos de 8 bits y uno de 5 bits. Cada puerto tiene 3 registros:

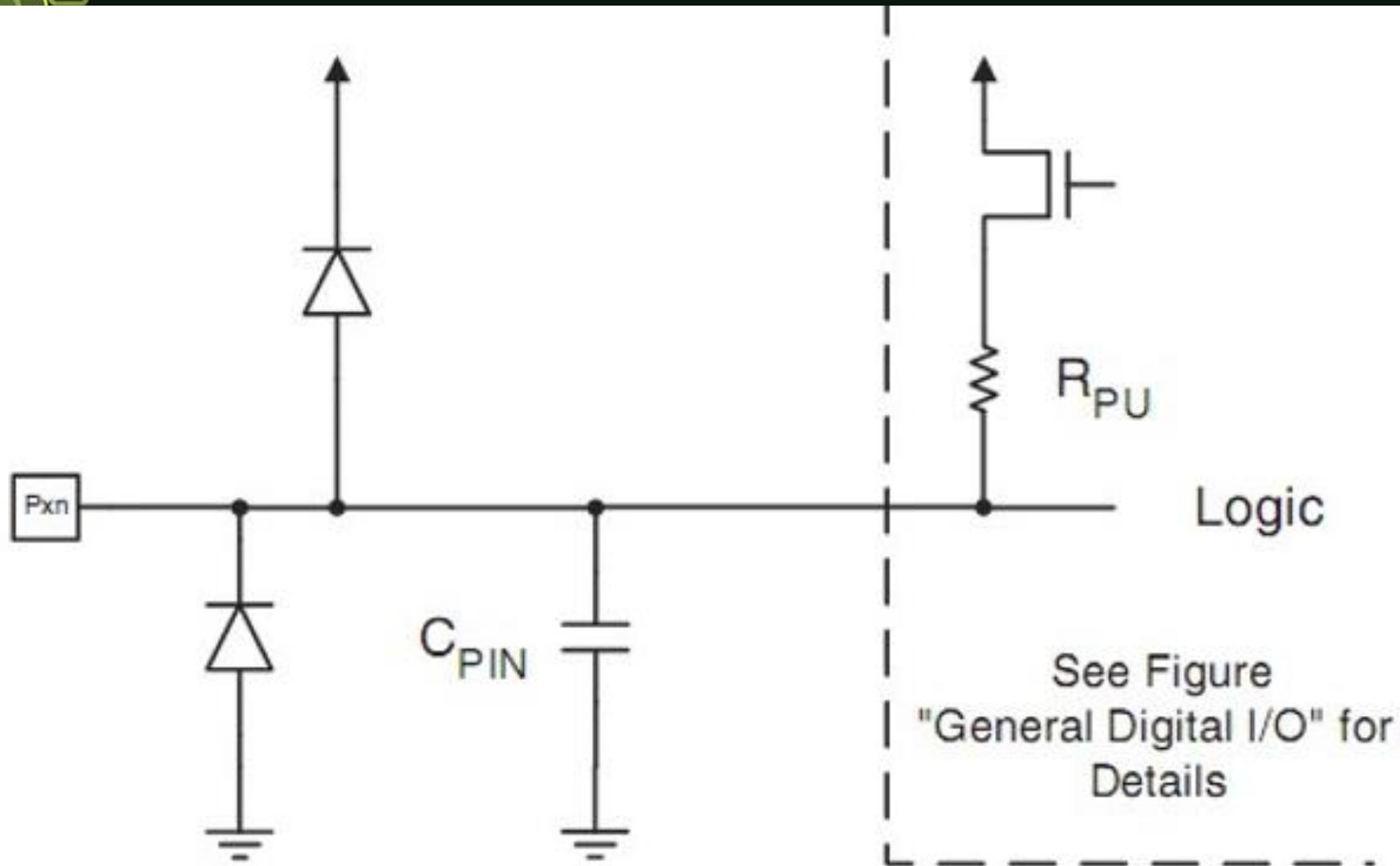
**DDR<sub>x</sub>, PORT<sub>x</sub> y PIN<sub>x</sub>.**

Los **dos** primeros permiten configurar el puerto como **entrada o salida**, y también brindan la posibilidad de poner un **pull-up**.



A través del registro **PINx** se puede consultar el **estado de cada bit** del puerto.

# Puertos de Entrada y Salida



# Puertos de Entrada y Salida

## Absolute Maximum Ratings\*

Operating Temperature ..... -55°C to +125°C

Storage Temperature ..... -65°C to +150°C

Voltage on any Pin except RESET  
with respect to Ground ..... -0.5V to V<sub>CC</sub>+0.5V

Voltage on RESET with respect to Ground.....-0.5V to +13.0V

Maximum Operating Voltage ..... 6.0V

DC Current per I/O Pin ..... 40.0 mA

DC Current V<sub>CC</sub> and GND Pins ..... 200.0 - 400.0mA

# Puertos de Entrada y Salida

**PORTA y PORTC:** Líneas de direccionamiento de memoria externa.

**PORTB:** Salidas PWM y puerto SPI.

**PORTD:** entradas de timer, interrupciones externas, puerto I2C, puerto UART1

**PORTE:** Interrupciones externas, salidas PWM, entradas de comparador analógico, puerto UART0, líneas de programación

**PORTF:** Entradas ADC y puerto JTAG.

**PORTG:** Entradas de Oscilador, líneas de control para memoria externa.

# Puerto A

## Conjunto de registros

Port A Data Register –  
PORTA

Bit	7	6	5	4	3	2	1	0	PORTA
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

Port A Data Direction  
Register – DDRA

Bit	7	6	5	4	3	2	1	0	DDRA
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

Port A Input Pins  
Address – PINA

Bit	7	6	5	4	3	2	1	0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A								

# Como configuro un puerto?



# Lenguaje C

## Puertos:

Iniciar el puerto como salida:

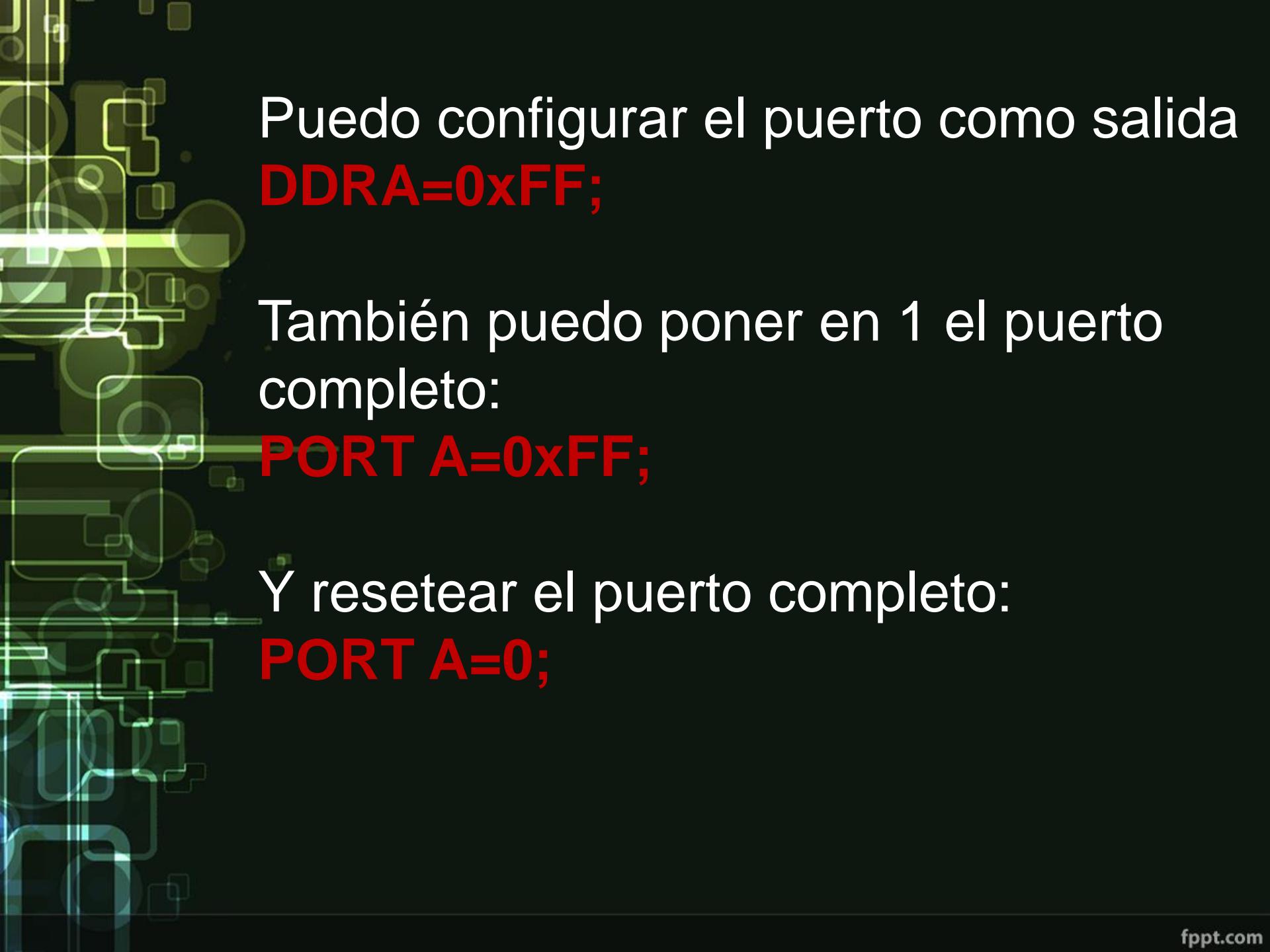
**DDRA=0x01;**

Escribir en el puerto A :

**PORTA=0x01;**

Con lo cual se pone a 1 el bit 0 del puerto A.  
análogamente para resetear el bit:

**PORTA=0x00;**



Puedo configurar el puerto como salida  
**DDRA=0xFF;**

También puedo poner en 1 el puerto completo:

**PORT A=0xFF;**

Y resetear el puerto completo:

**PORT A=0;**

# Lenguaje C

## Ejemplo:

Con el puerto configurado como salida

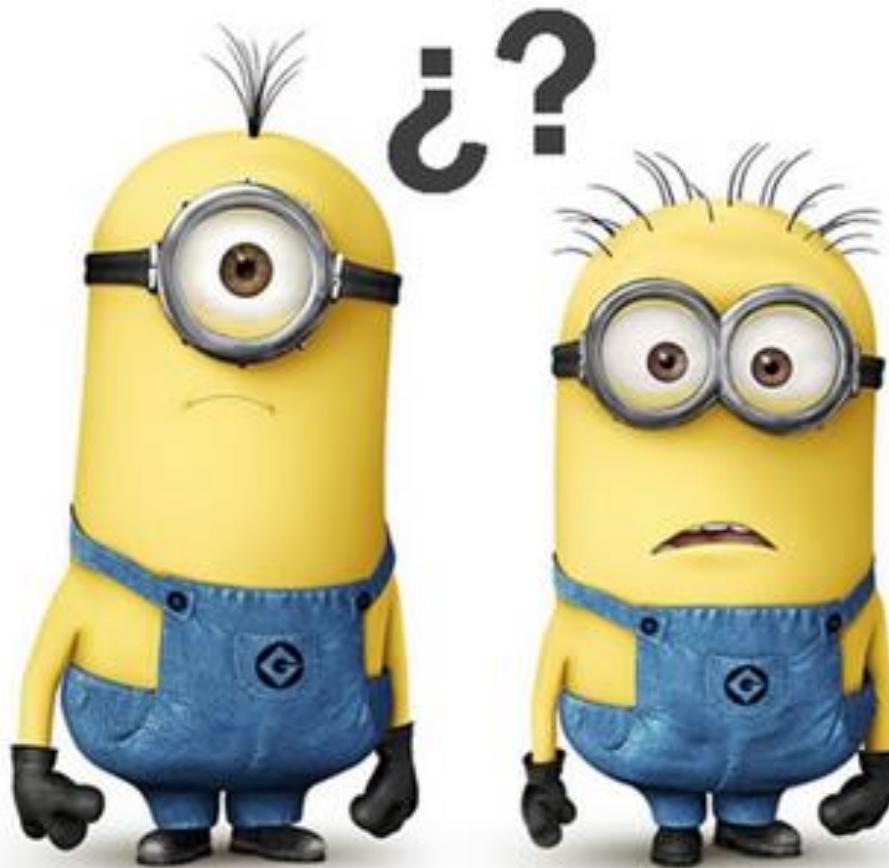
**PORTA|=0x01;** // Setea únicamente el bit 0

**PORTA&=0x01;** // Resetea todos menos el bit 0

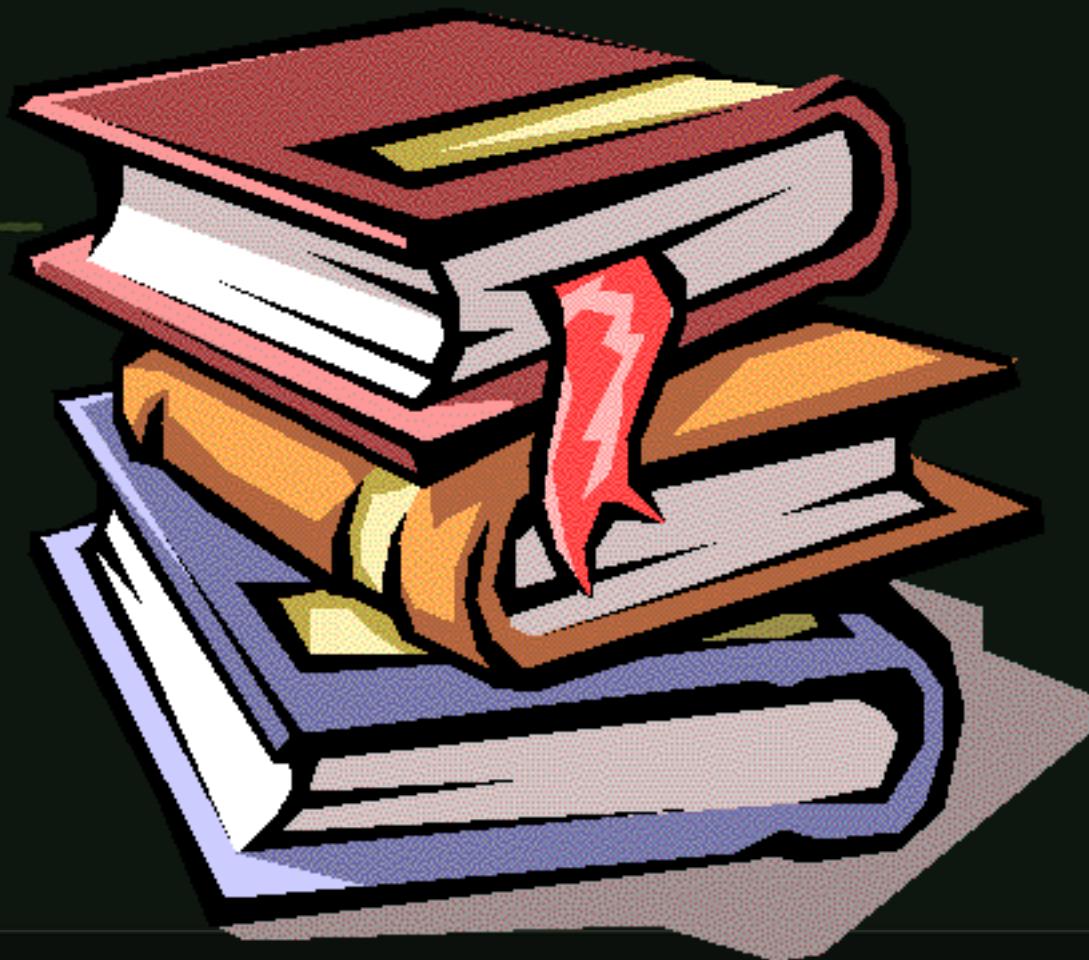
**PORTA|=(1<<7);** // Setea únicamente el bit 7

**PORTA&=~(1<<5);** // Resetea el bit 5

# ALGUNA DUDA?



# Continuemos con la clase....



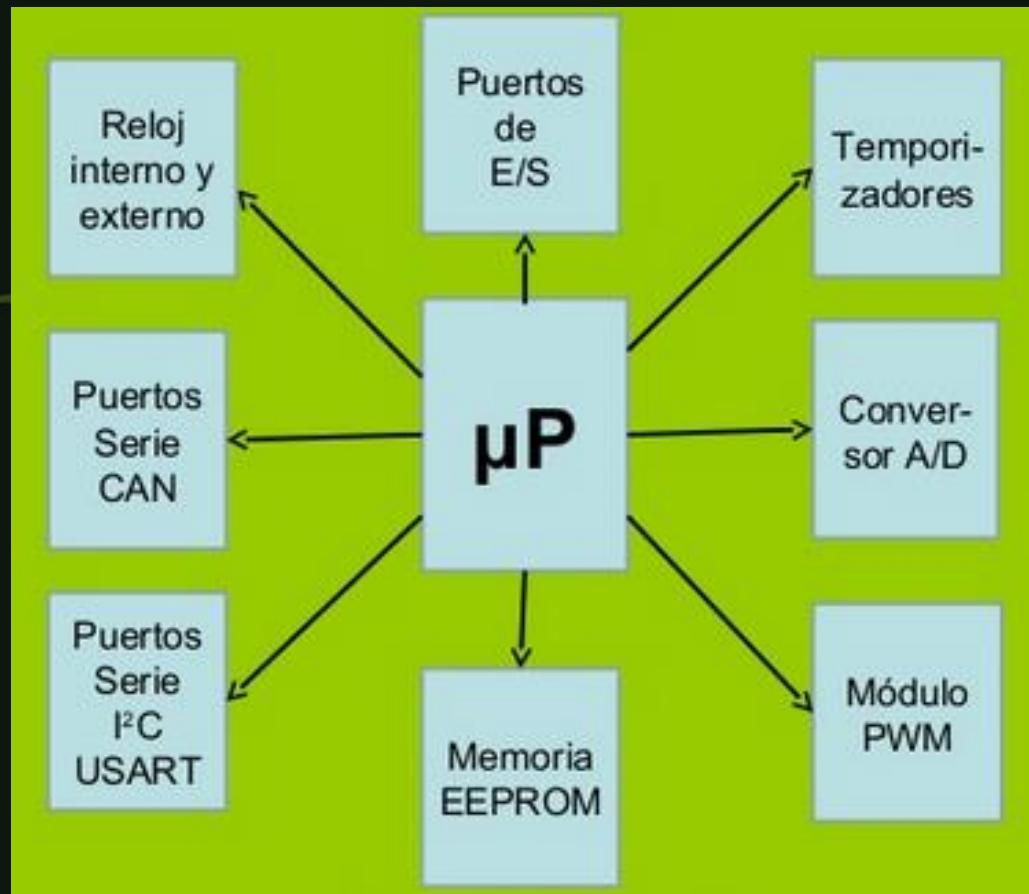
# INTERRUPCIONES EXTERNAS

Con una interrupción altera la ejecución del programa, haciendo que el micro salte a otro punto de la memoria para ejecutar la rutina de atención de esa interrupción, y finalmente volver al curso normal del programa principal



# INTERRUPCIONES EXTERNAS

- Pueden ser generadas por periféricos internos  
O externos.



# INTERRUPCIONES EXTERNAS

El Atmega128 posee 8 entradas de interrupciones externas:

**4 en el PORTD y 4 en el PORTE.**

Estas pueden habilitarse o deshabilitarse durante la ejecución del programa.

# INTERRUPCIONES EXTERNAS

Para la configuración de estas interrupciones se utilizan **4** registros:

- **EICRA**
- **EICRB**
- **EIMSK**
- **EIFR**

# EICRA

Se utiliza para definir si las interrupciones 0 al 3 son por nivel o por flanco, de acuerdo a la siguiente tabla:

<b>ISCn1</b>	<b>ISCn0</b>	<b>Description</b>
0	0	The low level of INTn generates an interrupt request.
0	1	Reserved
1	0	The falling edge of INTn generates asynchronously an interrupt request.
1	1	The rising edge of INTn generates asynchronously an interrupt request.

# Bits de configuración de EICRA:

**EICRB**

Se utiliza para definir si las interrupciones 4 al 7 son por nivel o por flanco, de acuerdo a la siguiente tabla:

<b>ISCn1</b>	<b>ISCn0</b>	<b>Description</b>
0	0	The low level of INTn generates an interrupt request.
0	1	Any logical change on INTn generates an interrupt request
1	0	The falling edge between two samples of INTn generates an interrupt request.
1	1	The rising edge between two samples of INTn generates an interrupt request.

# Bits de configuración de EICRB:

# EIMSK

Se utiliza para habilitar las interrupciones. Si el bit correspondiente a la interrupción está en 1, la interrupción está habilitada.

Bit	7	6	5	4	3	2	1	0	EIMSK
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

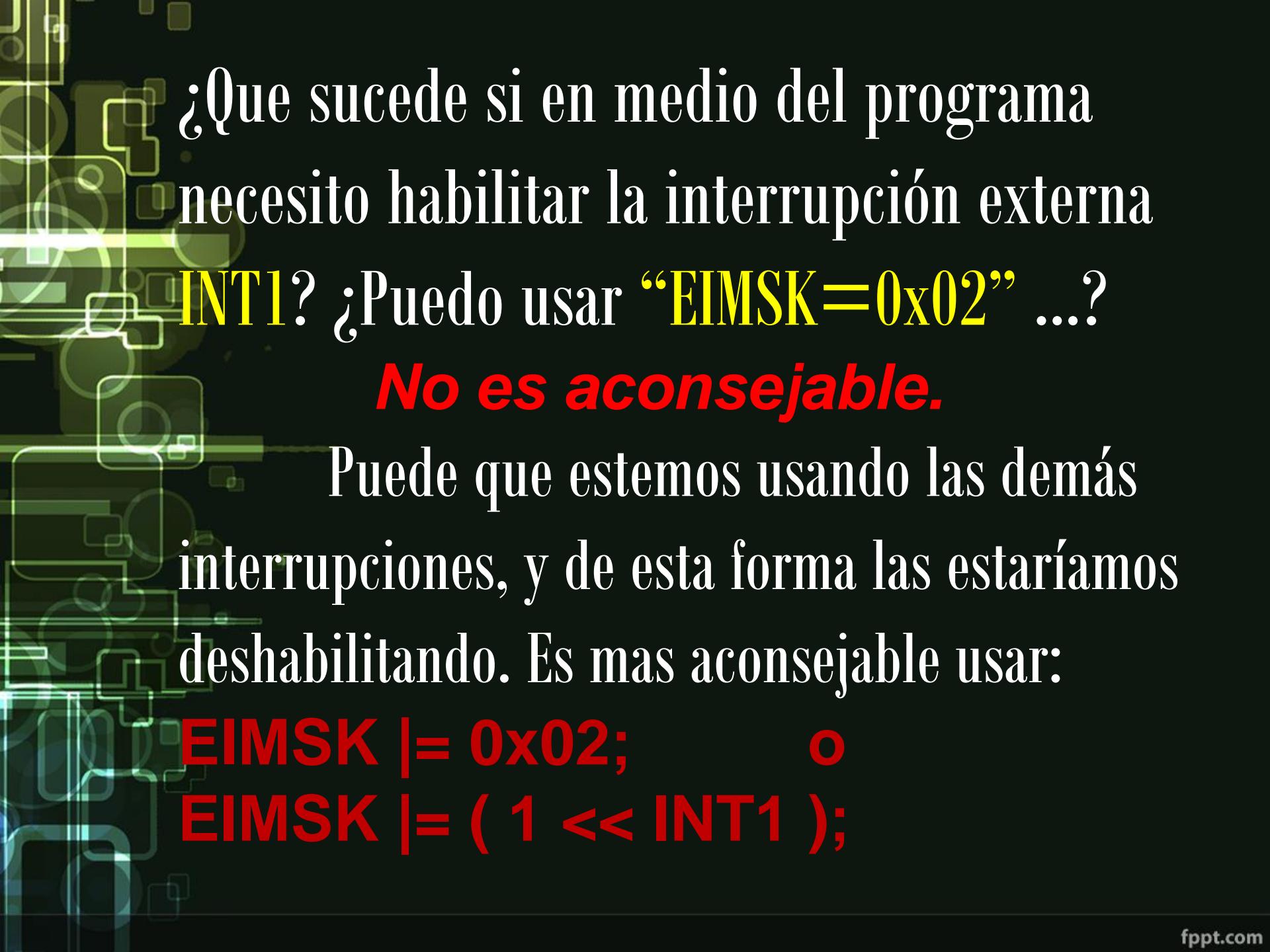
Por ejemplo para habilitar las interrupciones INT5, INT6 y INT7, basta con hacer:

**EIMSK=0xE0;**

# EIFR

Contiene los flags de interrupción.

Bit	7	6	5	4	3	2	1	0	EIFR
Read/Write	R/W	IINTFO							
Initial Value	0	0	0	0	0	0	0	0	0



¿Que sucede si en medio del programa  
necesito habilitar la interrupción externa  
**INT1**? ¿Puedo usar “**EIMSK=0x02**” ...?

***No es aconsejable.***

Puede que estemos usando las demás  
interrupciones, y de esta forma las estaríamos  
deshabilitando. Es mas aconsejable usar:

**EIMSK |= 0x02;**      o

**EIMSK |= ( 1 << INT1 );**

# TIMERS



# QUE es un TIMER?

Es un contador de pulsos

CONFIGURABLE

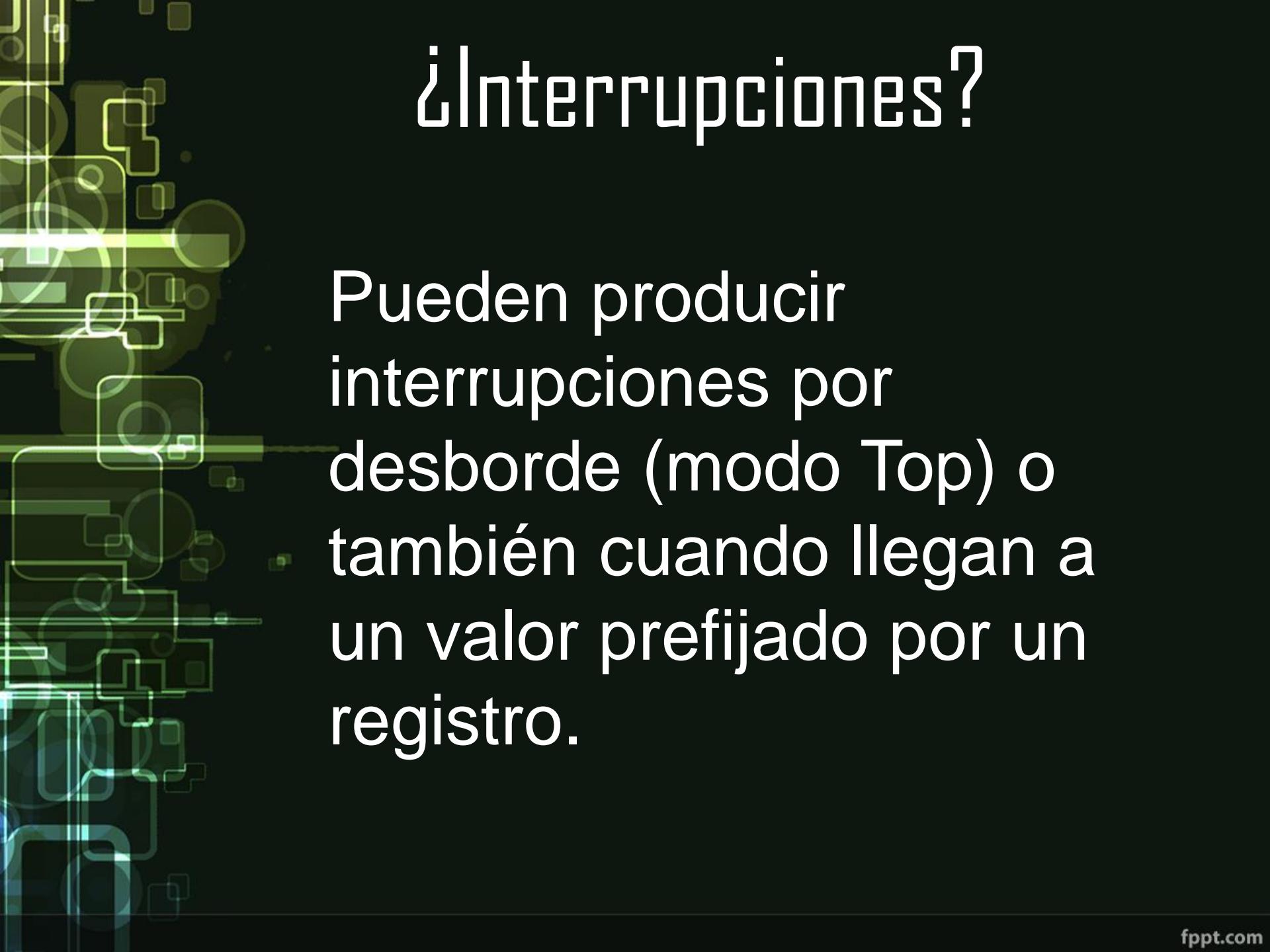
CAPAZ DE  
PRODUCIR  
INTERRUPCIONES

# ¿Clock configurable?

En la mayoría de los timers se puede elegir como fuente de señal de reloj el propio del sistema (o submúltiplos) o alguna entrada configurada para tal fin.

Así por ejemplo se puede usar como temporizador basado en la frecuencia del sistema o como un contador de eventos.

# ¿Interrupciones?



Pueden producir interrupciones por desborde (modo Top) o también cuando llegan a un valor prefijado por un registro.

# Timer/Counter de 8 bits

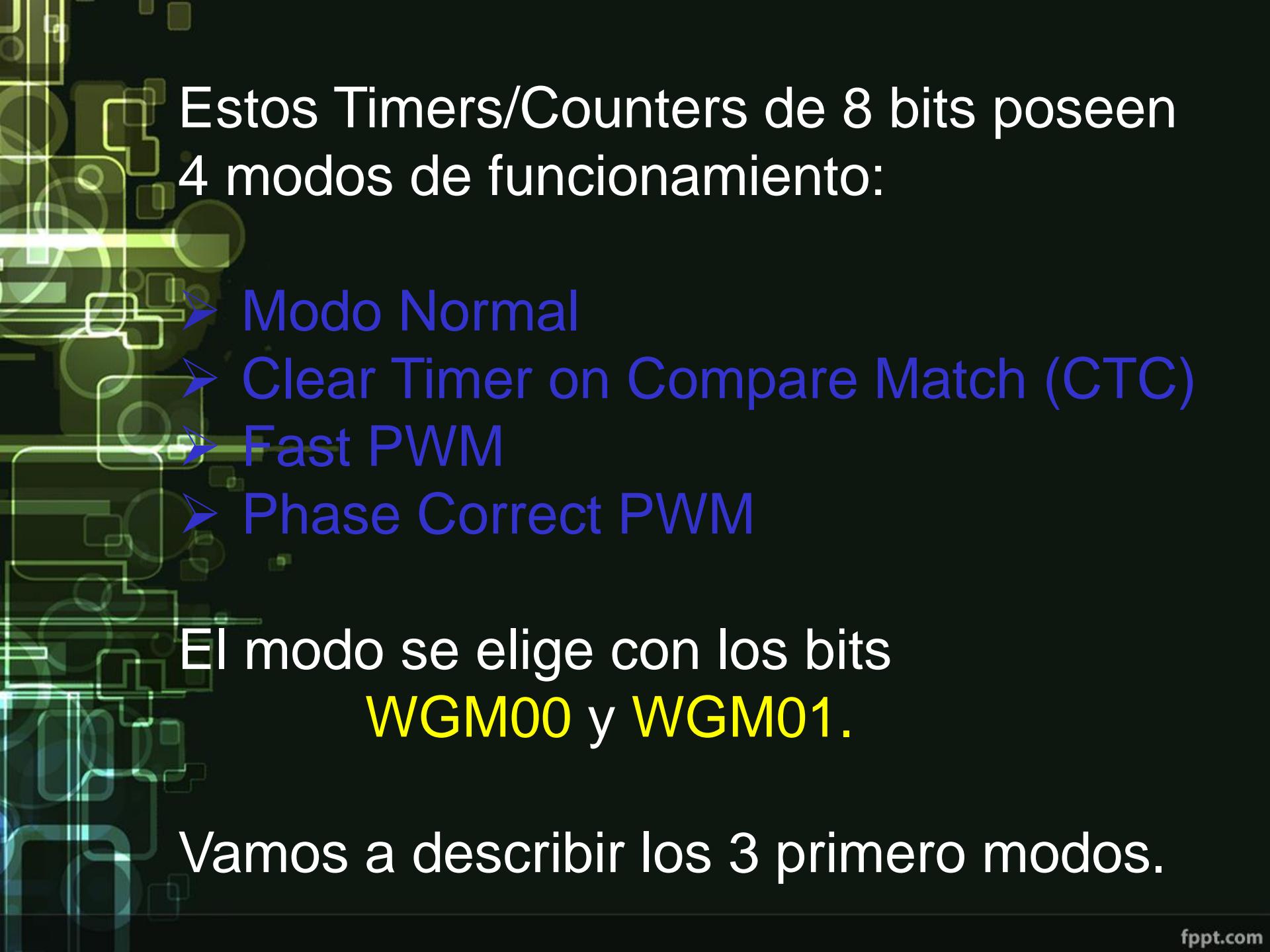
El Atmega128 posee 2 timers/counters de 8 Bits:

## **Timer0 y Timer2.**

La diferencia entre ambos está en que el **Timer 0** está preparado para trabajar con el clock generado por un cristal de 32KHz, con lo cual puede trabajar en forma asincrónica con el sistema.

En cambio el **Timer 2** está preparado para contar eventos en forma sincrónica.

A pesar de estas diferencias, sus modos de funcionamiento son muy parecidos, lo cual nos permite analizarlos en forma conjunta



Estos Timers/Counters de 8 bits poseen 4 modos de funcionamiento:

- Modo Normal
- Clear Timer on Compare Match (CTC)
- Fast PWM
- Phase Correct PWM

El modo se elige con los bits  
**WGM00** y **WGM01**.

Vamos a describir los 3 primero modos.

# Modo Normal: (WGM01:0 = 0)

En el modo **normal** el Timer/Counter incrementa su valor en cada pulso de clock hasta llegar al valor máximo de 255; luego con el siguiente pulso de clock se desborda y “avanza” a cero. Es posible generar una interrupción en el momento del desborde.

# Clear Timer on Compare Match (CTC) : (WGM01:0 = 2)

El Timer/Counter se incrementa hasta el valor definido por OCRn (OCR0 u OCR2). Al alcanzar dicho valor el contador se resetea a cero y vuelve a contar. Se puede configurar para que se genere una interrupción cuando el contador alcanza el valor OCRn.

# Clear Timer on Compare Match (CTC) : (WGM01:0 = 2)

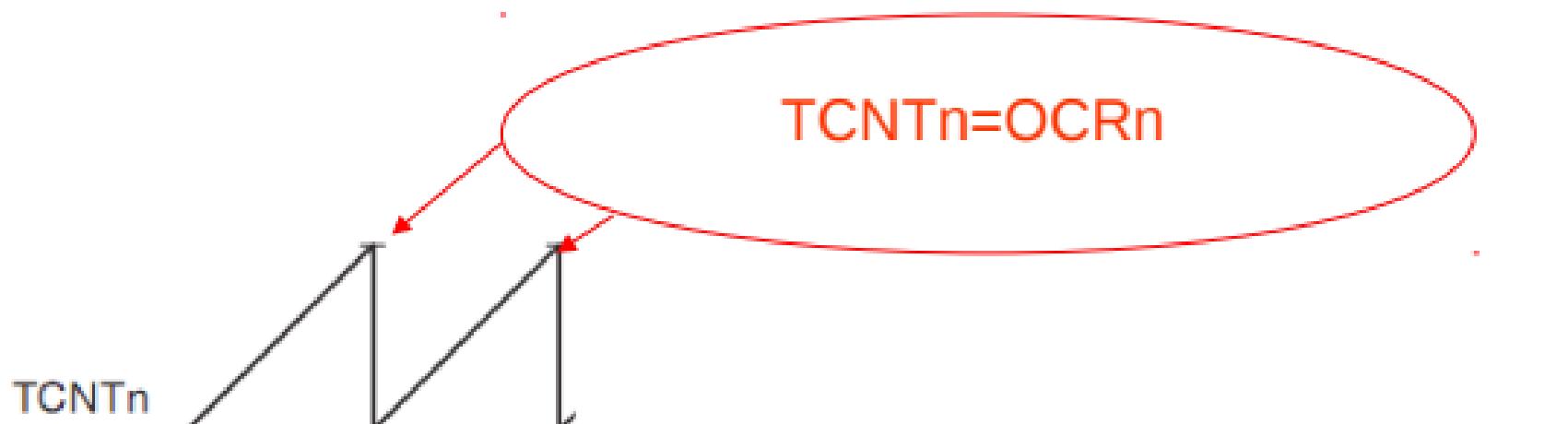
También es posible configurar la salida OCn (OC0 u OC2) para que cambie su estado lógico cada vez que el contador llega al valor máximo. Para que esto ocurra el pin OCn debe estar configurado como salida. Esto nos sirve por ejemplo para generar un tren de pulsos.



# Clear Timer on Compare Match (CTC) : (WGM01:0 = 2)



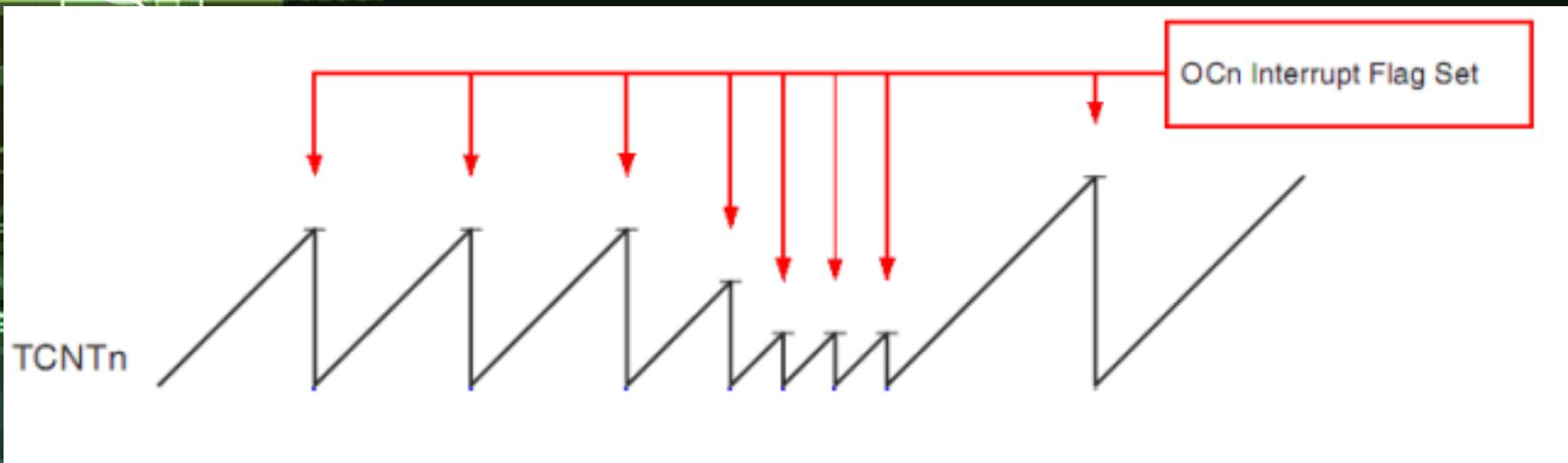
# Clear Timer on Compare Match (CTC) : (WGM01:0 = 2)



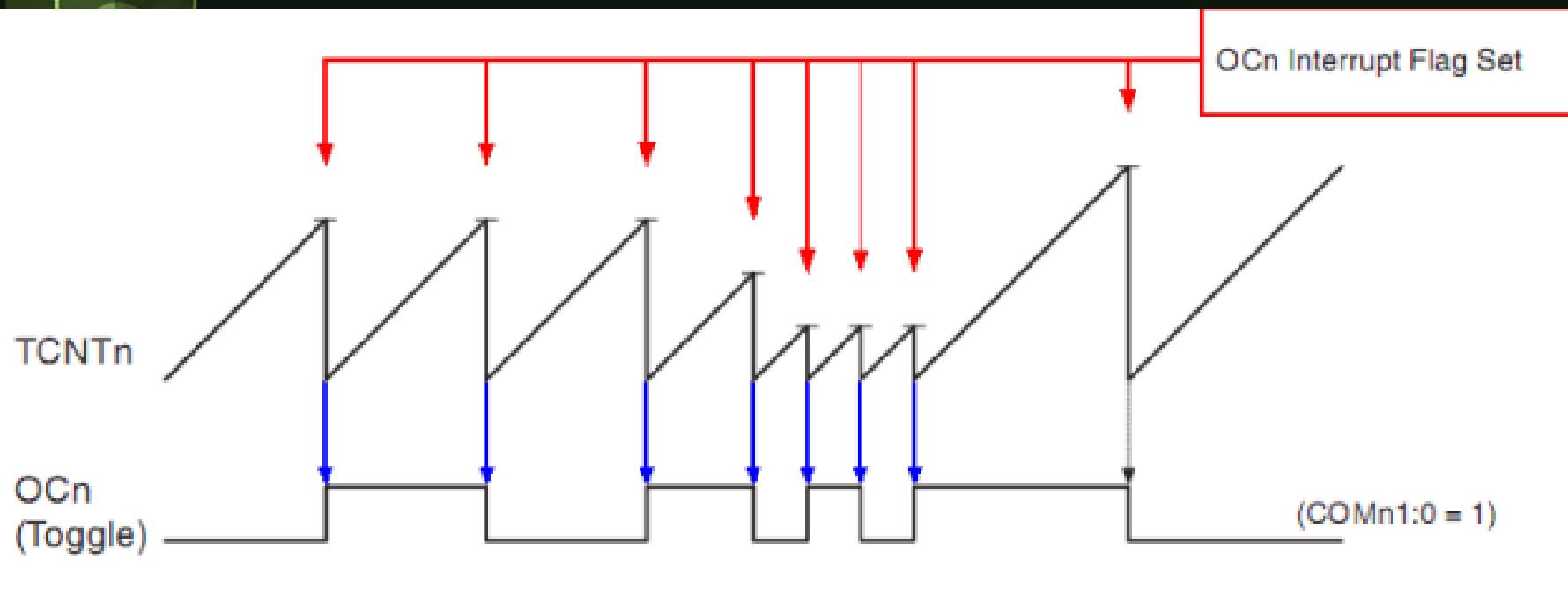
# Clear Timer on Compare Match (CTC) : (WGM01:0 = 2)



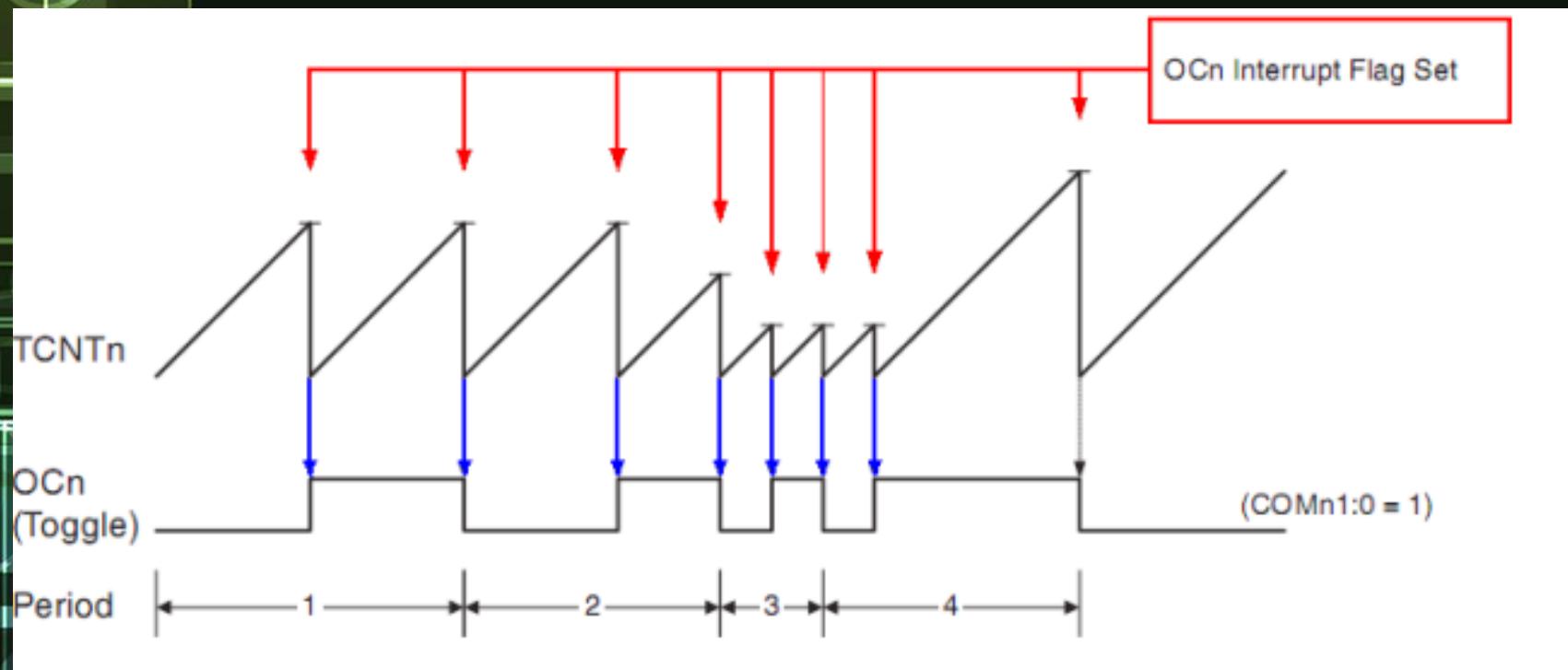
# Clear Timer on Compare Match (CTC) : (WGM01:0 = 2)



# Clear Timer on Compare Match (CTC) : (WGM01:0 = 2)



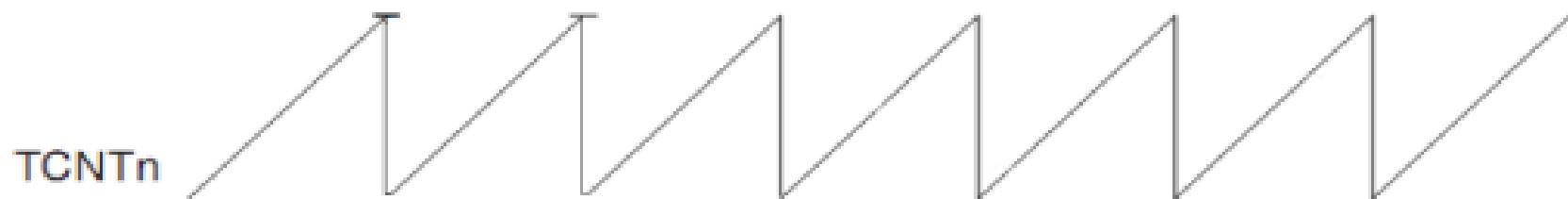
# Clear Timer on Compare Match (CTC) : (WGM01:0 = 2)



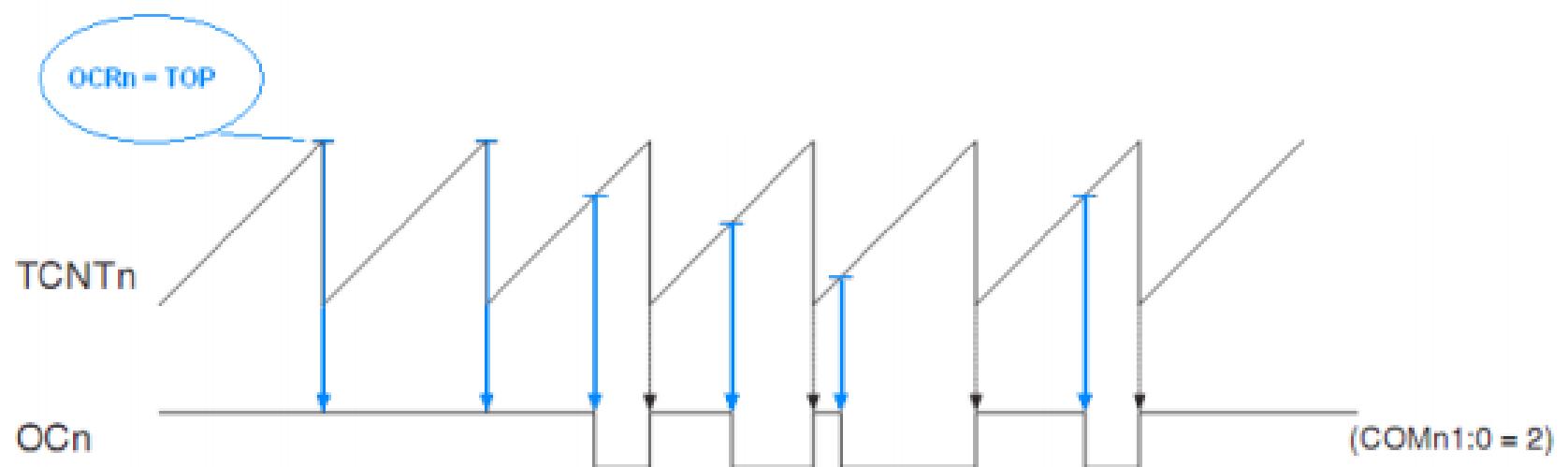
# FAST PWM : (WGM01:0 = 3)

En este modo el Timer/Counter se incrementa hasta el valor definido por OCRn (OCR0 u OCR2). Al alcanzar dicho valor el contador cambia de estado la salida OCn (OC0 u OC2) y sigue la cuenta hasta llegar al valor máximo, donde se resetea y vuelve a cambiar nuevamente el estado de OCn. Dependiendo de los valores de los bits del registro COMn, será el estado de la salida OCn en cada cambio, dando lugar a un PWM no invertido y a un PWM invertido.

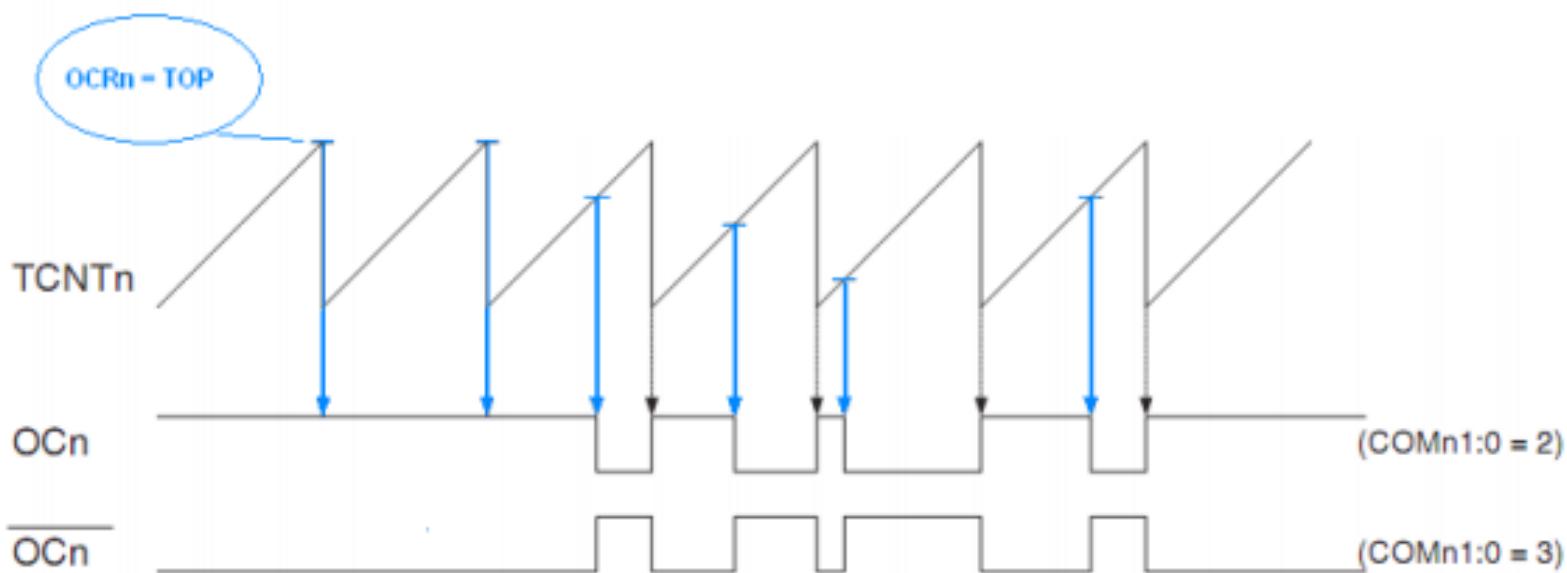
# FAST PWM : (WGM01:0 = 3)



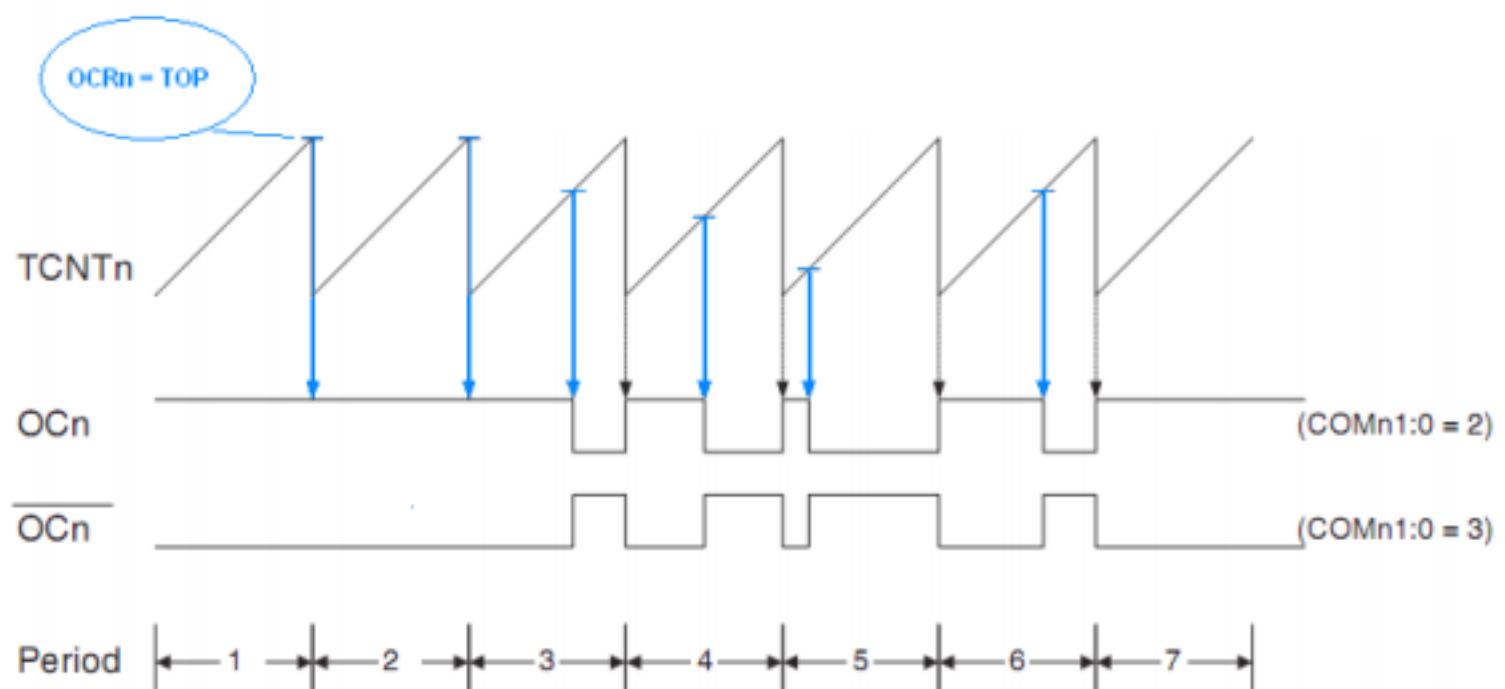
# FAST PWM : (WGM01:0 = 3)



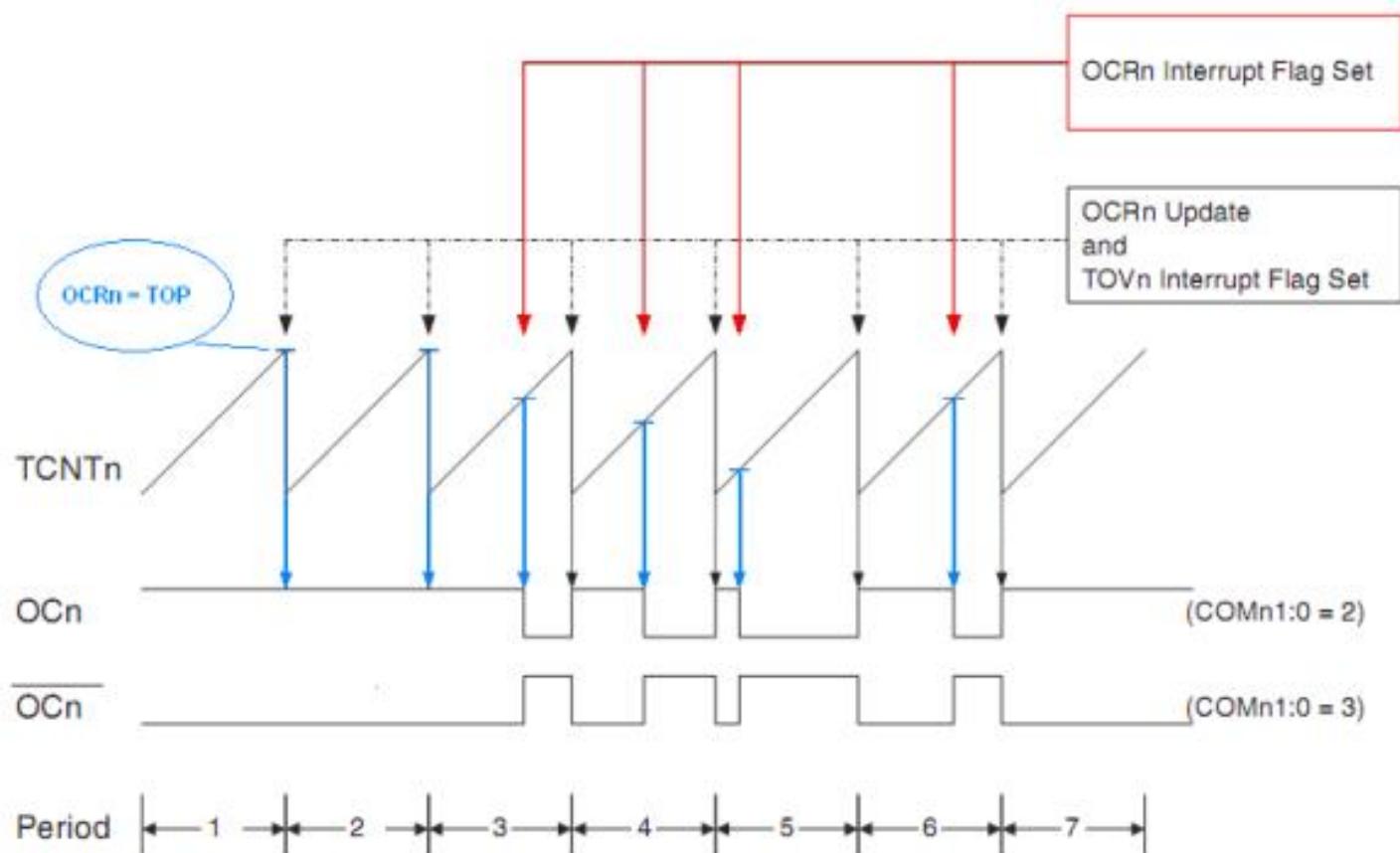
# FAST PWM : (WGM01:0 = 3)



# FAST PWM : (WGM01:0 = 3)



# FAST PWM : (WGM01:0 = 3)



# FAST PWM : (WGM01:0 = 3)

Se puede configurar de manera de generar interrupciones cuando el Timer alcanza el valor OCRn y cuando alcanza el valor máximo. Esto nos sirve para ir cambiando el valor de OCRn y por consiguiente cambiar el ciclo de actividad.

# TCCR<sub>n</sub>

## Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	TCCR0
Read/Write	W	R/W							
Initial Value	0	0	0	0	0	0	0	0	

**FOC<sub>n</sub>:** Force Output Compare. Sirve para forzar la salida Ocn en modos no PWM

**WGM<sub>n1:0</sub>:** Wave Generator Mode. Estos bits fijan el modo del Timer

Mode	WGM21 (CTC2)	WGM20 (PWM2)	Timer/Counter Mode of Operation	TOP	Update of OCR2 at	TOV2 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR2	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

# TCCR<sub>n</sub>

## Timer/Counter Control Register

**COM<sub>n1:0</sub>:** Compare Match Output Mode. Define el funcionamiento de la salida OC<sub>n</sub> de acuerdo al modo seleccionado.

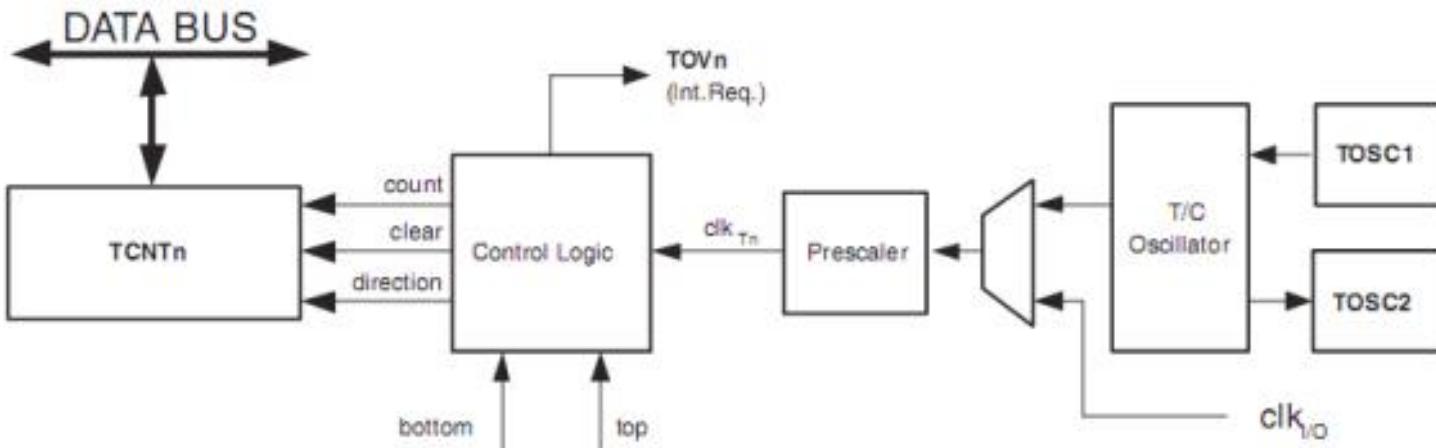
Los bits CS<sub>n0</sub>, CS<sub>n1</sub> y CS<sub>n2</sub> se utilizan para definir la fuente de clock del timer/counter

Compare Output Mode, Non-PWM Mode		
COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Toggle OC2 on compare match
1	0	Clear OC2 on compare match
1	1	Set OC2 on compare match

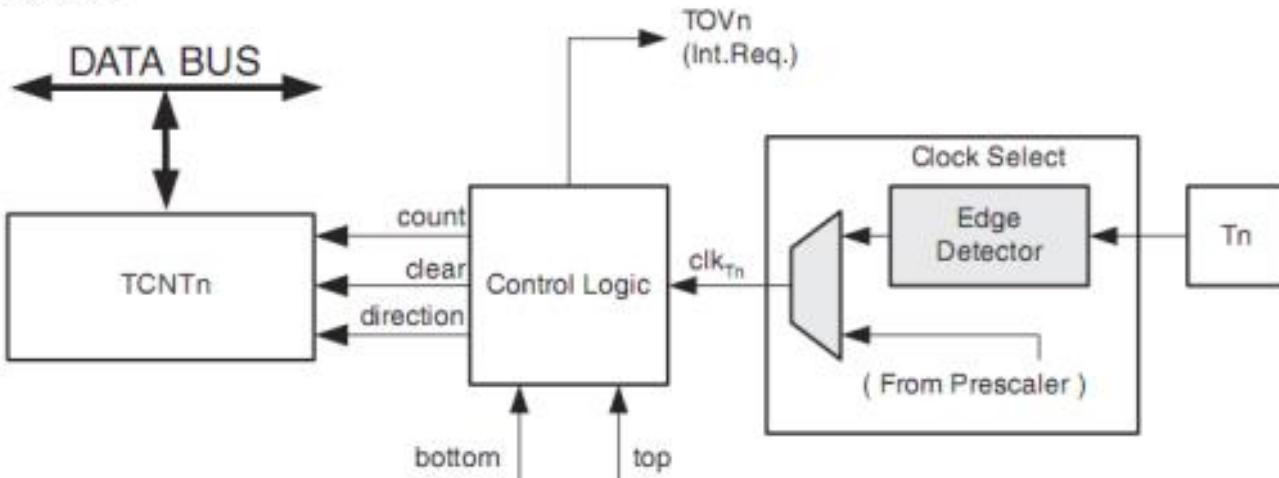
Compare Output Mode, Fast PWM Mode <sup>(1)</sup>		
COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Reserved
1	0	Clear OC2 on compare match, set OC2 at BOTTOM, (non-inverting mode)
1	1	Set OC2 on compare match, clear OC2 at BOTTOM, (inverting mode)

# Fuentes de clock del Timer/Counter

## TIMER/COUNTER 0



## TIMER/COUNTER 2



# Registros del Timer/Counter

**TCNT<sub>n</sub>**: permite el acceso de lectura o escritura de la unidad de cuenta.

**OCR<sub>n</sub>**: Output Compare Register. Mantiene el valor que se compara continuamente con el valor del contador TCNT<sub>n</sub>.

**TIMSK**: Timer/Counter Interrupt Mask Register. Contiene los bits de habilitación de las interrupciones de los Timers

# Timer/Counter de 16 bits

El Atmega128 posee 2 timers/counters de 16 Bits:

## **Timer1 y Timer3.**

La primera diferencia esencial entre estos timers y los de 8 bits, es que el valor máximo de cuenta ya no es 255 sino 65535.

Además de eso:

- Tienen un modo de operación más: Fase and Frequency Correct PWM .
- El modo de operación normal es igual al de 8 bits excepto por el valor máximo de cuenta.
- El modo CTC es igual al de 8 bits solo que con 16 bits.

# Timer/Counter de 16 bits

En el modo **Fast PWM** hay 2 grandes diferencias respecto a de 8 bits.

- El valor máximo ya no es fijo 255, sino que pueden seleccionarse distintos valores de resolución fija (512, 1024, etc) o puede ser variable a través de un registro.

- Se pueden tener hasta 3 salidas PWM con el mismo timer.

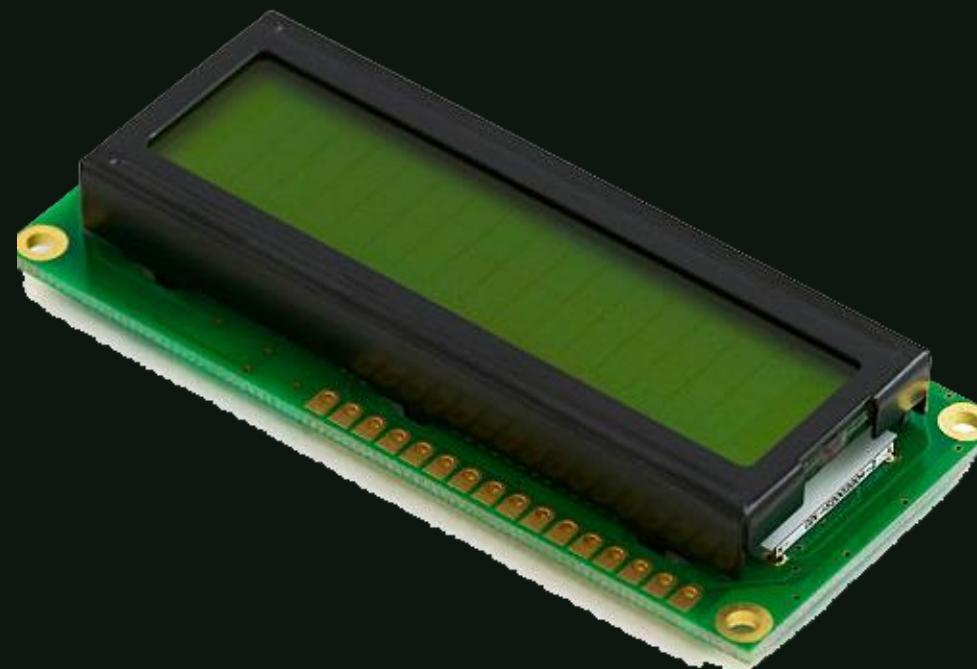
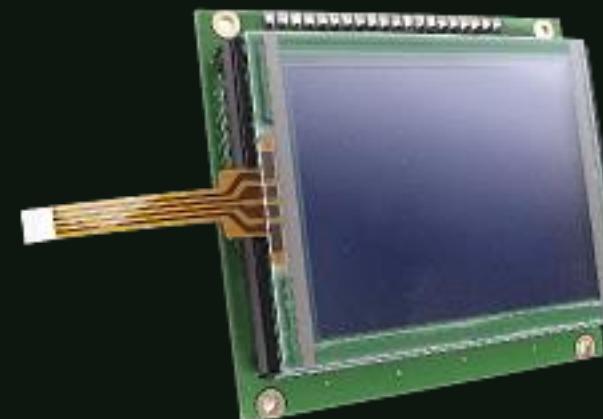
En el modo Fase Correct PWM también están estas 2 diferencias mencionadas.

# Los registros del timer son muuuuchos

LEER  
HOJA DE  
DATOS



# Usando Displays



# Displays 7 Segmentos

Vamos a ver en forma práctica como se implementa el multiplexado de 4 dígitos 7 segmentos controlados por un circuito integrado CD4511 y 4 transistores.

Este tipo de solución es bastante sencilla y no requiere mayores cambios para escalarla a otra cantidad de dígitos.



El IC 4511 es nuestro decodificador BCD a 7 Segmentos. Sacando por un nibble valores de 0 a 9, y adicionalmente, usando transistores que habiliten cada uno de los displays podemos barrerlos y ver una presentación fluida de los datos a mostrar.

# Display LCD Alfanumericos

Se suele hablar de display LCD inteligentes dado que éstos poseen un grado de “inteligencia”. Esto se debe a que estos display incorporan un pequeño controlador que les permite realizar las distintas funciones de control y presentación de los caracteres presentados en la pantalla, sin necesidad de intervención del usuario.

Esto significa que para poder escribir caracteres en la pantalla se le deben enviar los comandos apropiados al controlador. Estos comandos en general son los mismos para las distintas marcas de displays, siempre y cuando los controladores sean “compatibles”.

# Display LCD Alfanumericos

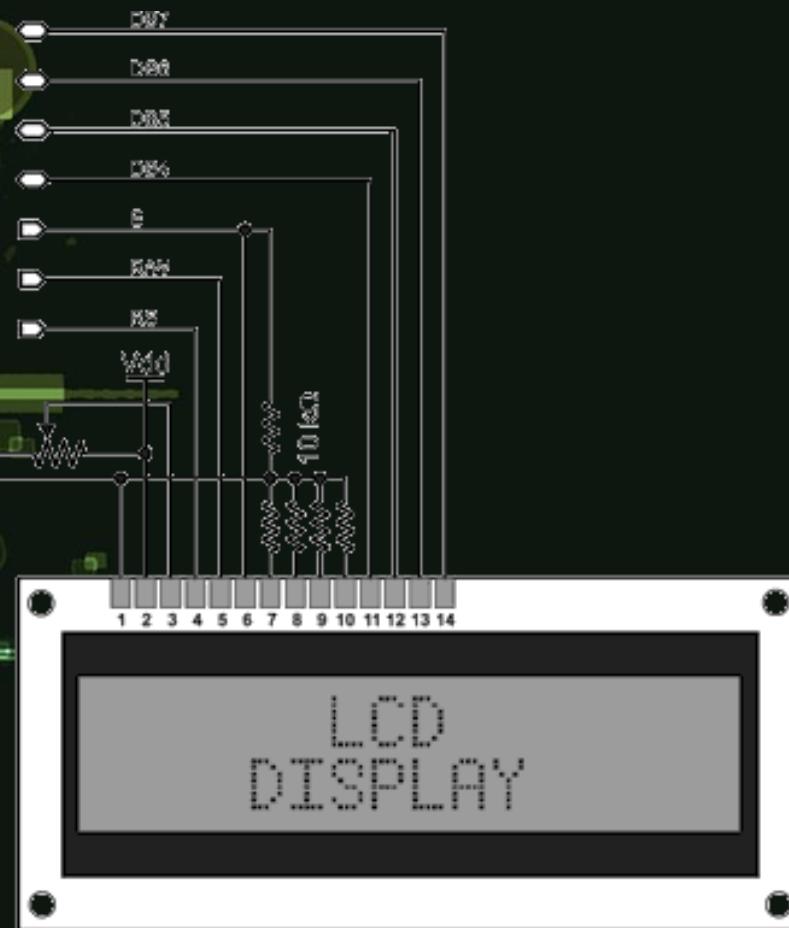
El display posee 2 tipos de líneas de conexión:

- Líneas de control (3)
- Líneas de datos (8 o 4)

Las líneas de control nos van a servir para indicarle al controlador de display si le vamos a enviar un comando o un dato/argumento, o si vamos a leer o escribir.

Las líneas de datos nos van a servir para enviar el código del comando o un argumento del mismo.

# Display LCD Alphanumericos



También, como es de esperarse, el display va a tener entradas de alimentación. Y en algunos casos vamos a encontrar también dos terminales adicionales que corresponden al LED de backlight. A continuación el diagrama en bloque simplificado de un display tomado de una hoja de datos.

Instruction	Instruction code										Description	Execution time (fosc= 270 KHZ)
	RS	R/M	DB5	DB4	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRA and set DDRAM address to "00H" from AC	1.53ms
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to "00H" From AC and return cursor to Its original position if shifted. The contents of DDRAM are not changed.	1.53ms
Entry mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction And blinking of entire display	39us
Display ON/OFF control	0	0	0	0	0	0	1	D	C	B	Set display (D), cursor (C), and Blinking of cursor (B) on/off Control bit.	
Cursor or Display shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display Shift control bit, and the Direction, without changing of DDRAM data.	39us
Function set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL: 8-bit/4-bit), numbers of display Line (N: =2-line/1-line) and, Display font type (F: 5x11/5x8)	39us
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address Counter.	39us
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address Counter.	39us
Read busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal Operation or not can be known By reading BF. The contents of Address counter can also be read.	0us
Write data to Address	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	43us
Read data From RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	43us

# Display LCD Alphanumericos

Nuevamente esto se va complicando...

Pero por suerte existen librerías que nos permiten controlar este tipo de displays utilizando funciones de alto nivel.

## Funciones de manejo de display:

`char lcd_init(unsigned char dispAttr)`

Inicializa el display, lo borra completamente y posiciona el puntero en el comienzo (columna 0 y fila 0).

Con el parámetro dispAttr indicamos el modo del cursor.

# Display LCD Alphanumericos

## Funciones de Control

`void lcd_clrscr(void);`

Borra completamente el LCD y posiciona el puntero en (0,0).

`void lcd_home(void);`

Posiciona el puntero en (0,0).

`void lcd_gotoxy(unsigned char x, unsigned char y)`

Posiciona el puntero en la posición (x,y).

# Display LCD Alphanumericos

## Funciones de Escritura

`void lcd_putc(char c)`

Escribe el caracter 'c' en la posición actual.

`void lcd_puts(char *str)`

Escribe una cadena ubicada en RAM a partir de la posición actual.

`void lcd_puts_p(char flash *str)`

Escribe una cadena ubicada en la flash a partir de la posición actual. Todas incrementan la posición luego de escribir

# Display LCD Alphanumericos

## Funciones de Bajo Nivel

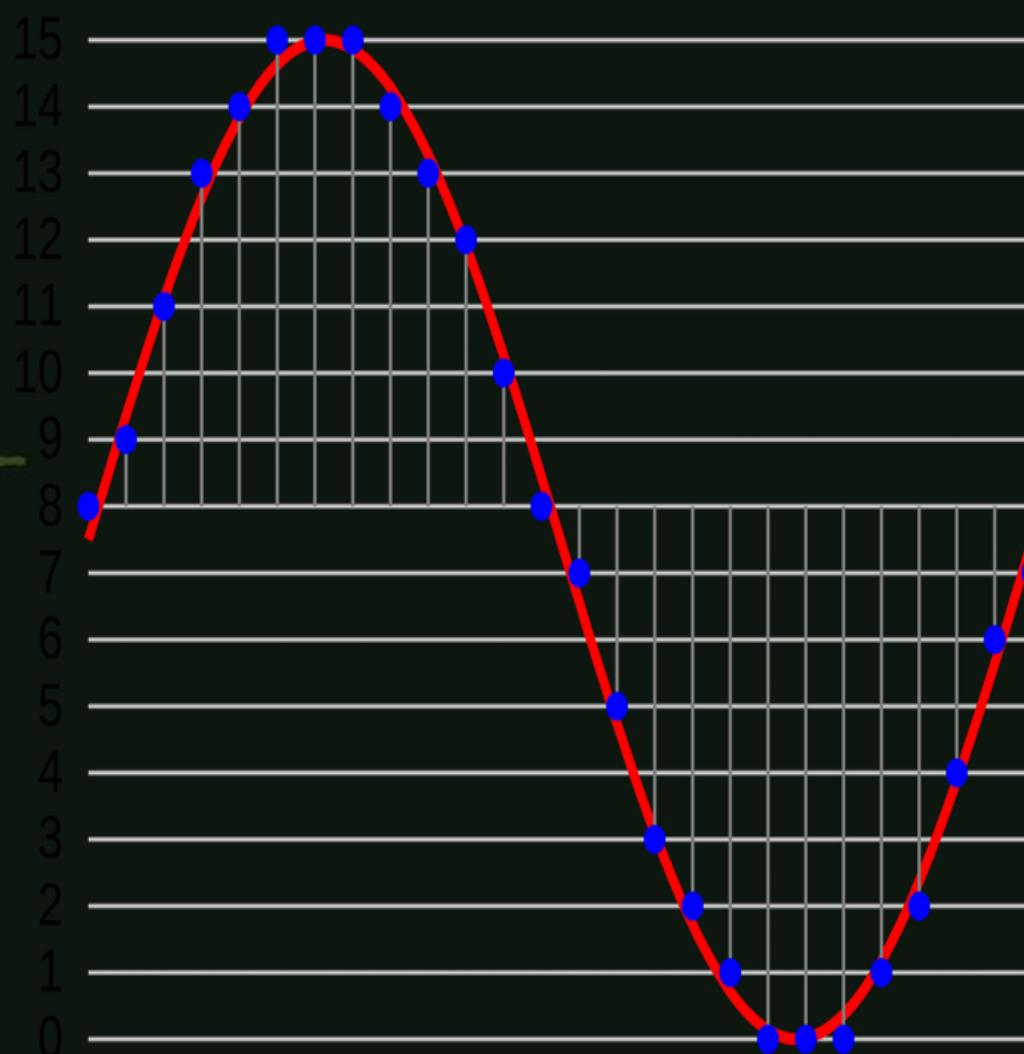
**vlcd\_command(unsigned char cmd)**

Escribe un byte en el registro de instrucciones.  
Sirve para configurar el display.

Estas funciones nos sirven por ejemplo para activar el cursor para que este destelle o para escribir caracteres especiales

# Conversor Analógico Digital

ADC



# Conversor Analógico Digital



Un conversor Analógico a Digital (o ADC) transforma una señal **analógica** a un **número binario**. La cantidad de dígitos binarios definen la resolución del ADC. Este número digital es solo una aproximación de la señal analógica, dado que solo puede ser representada en pasos discretos. La exactitud de la conversión dependerá también de la resolución del ADC.

# Conversor Analógico Digital

El Atmega128 posee un conversor A/D de aproximaciones sucesivas.

Características relevantes:

- 10 bits de resolución
- 8 canales en modo común multiplexados
- Hasta 7 canales en modo diferencial.
- Hasta 15KSPS
- Referencia seleccionable (Externa o 2.56V internos)
- Interrupción al completar la conversión

# Conversor Analógico Digital

Para utilizar el conversor A/D y realizar una conversión simple es necesario:

- Habilitar el ADC mediante el bit ADEN.
- Seleccionar la referencia (AREF, AVCC o 2.56V Int).
- Seleccionar el canal y el modo (común o diferencial)
- Poner en un 1 en el bit ADC Start Conversion ADSC. Una vez hecho esto se debe esperar que el ADC haga la conversión y cuando esta finaliza, se pone en 1 el flag ADIF y, si está habilitada, se genera una interrupción. En estas condiciones se puede leer el resultado contenido en los registros ADCH y ADCL.

# Conversor Analógico Digital

El resultado de la conversión en modo común responde a la siguiente fórmula:

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

Por ejemplo, si VREF es 2.56V y la entrada VIN es 1V, el resultado de la conversión es 400 (ADCH=01h y ADCL=90h)

# Conversor Analógico Digital

El resultado de la conversión en modo diferencial responde a la siguiente fórmula:

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot GAIN \cdot 512}{V_{REF}}$$

Es presentado en complemento a 2, desde -512 (ADCH=02h y ADCL=00h) hasta +511 (ADCH=01h y ADCL=FFh).

La polaridad del resultado puede obtenerse leyendo el 2do LSB del registro ADCH.

# Conversor Analógico Digital

Registros: ADMUX: ADC Multiplexer Selection Register.

Bit	7	6	5	4	3	2	1	0	ADMUX
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

REFS1:0: Reference Selection Bits. Permiten seleccionar la referencia.

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

# Conversor Analógico Digital

Registros: ADMUX: ADC Multiplexer Selection Register.

Bit	7	6	5	4	3	2	1	0	ADMUX
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

ADLAR: ADC Left Adjust Result. Se utiliza para ajustar el resultado de la conversion.

MUX4:0: Permite seleccionar uno de los 32 modos de conexión de las entrada al ADC.

# Conversor Analógico Digital

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A	N/A	1x
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000 <sup>(1)</sup>		ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010 <sup>(1)</sup>		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x

# Conversor Analógico Digital

## ADCSRA: ADC Control and Status Register A.

Bit	7	6	5	4	3	2	1	0	ADCSRA
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

ADCEN: ADC Enable. Habilita el ADC. Si no se usa debe estar en 0.

ADSC: ADC Start Conversion. Se utiliza para inicial la conversión simple.

ADFR: ADC Free Running Select. Habilita el modo Free running\*\*\*

ADIF: ADC interrupt Flag. Se pone en 1 cuando se completa la conversión.

ADIE: ADC Interrupt Enable. Habilita la interrupción del ADC.

ADPS2:0: ADC Preescaler Select Bits: Determinan el factor de división del clock.

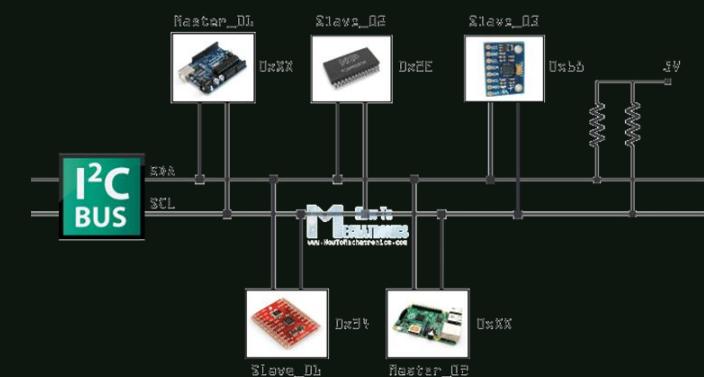
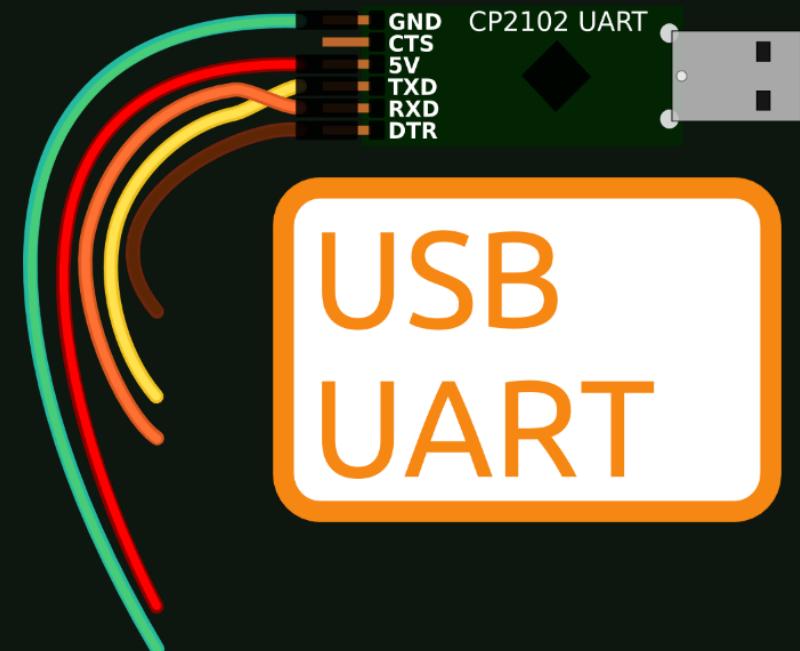
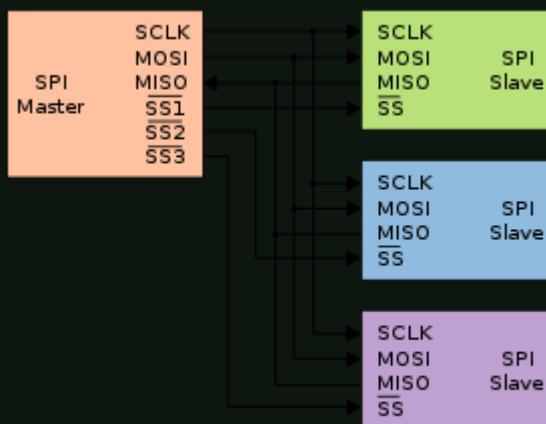
# Conversor Analógico Digital

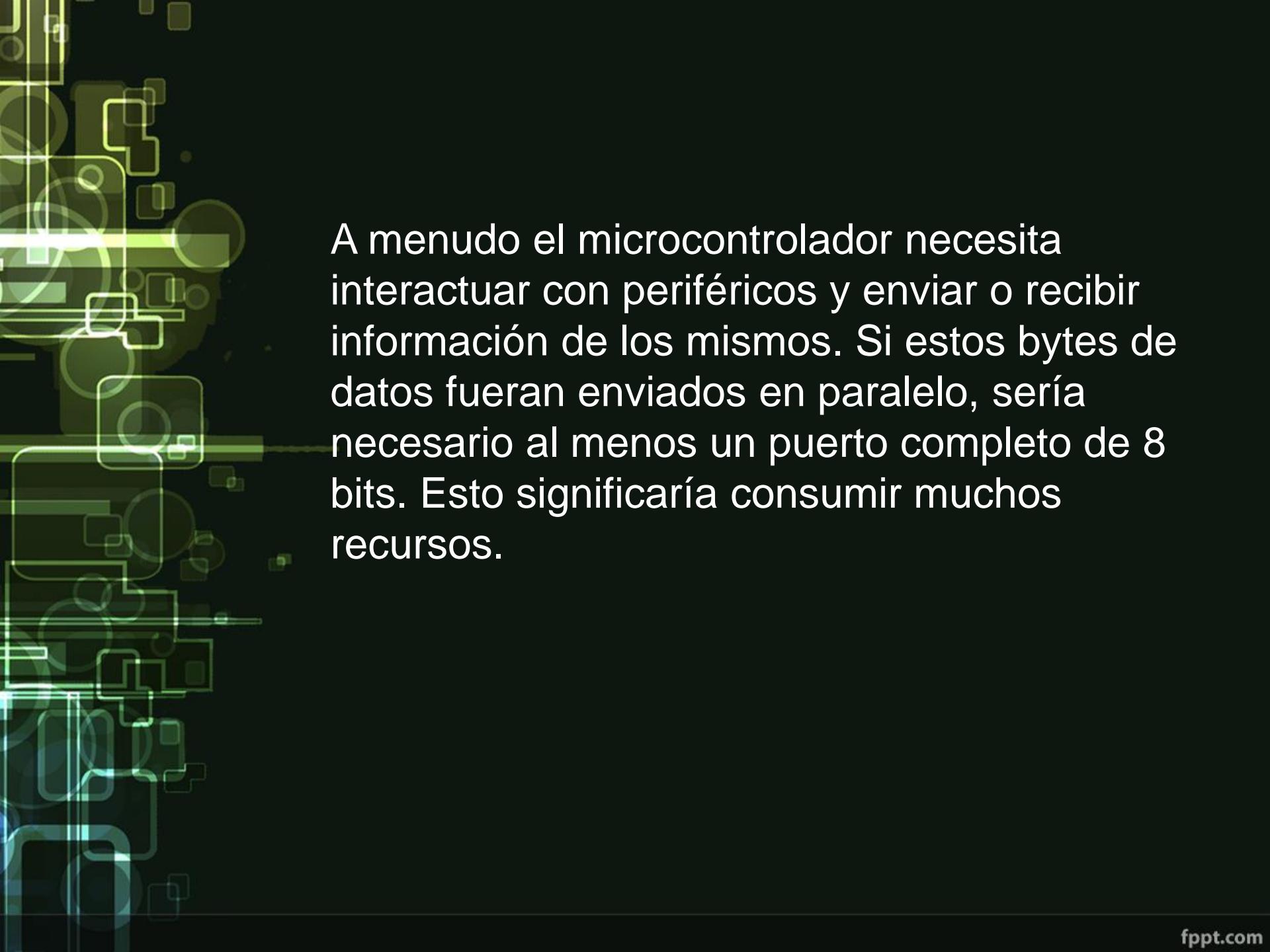
Haremos nuestras Funciones para usar el ADC.

Primero la haremos bloqueante.

Y luego la haremos no bloqueante.

# Comunicación Serie

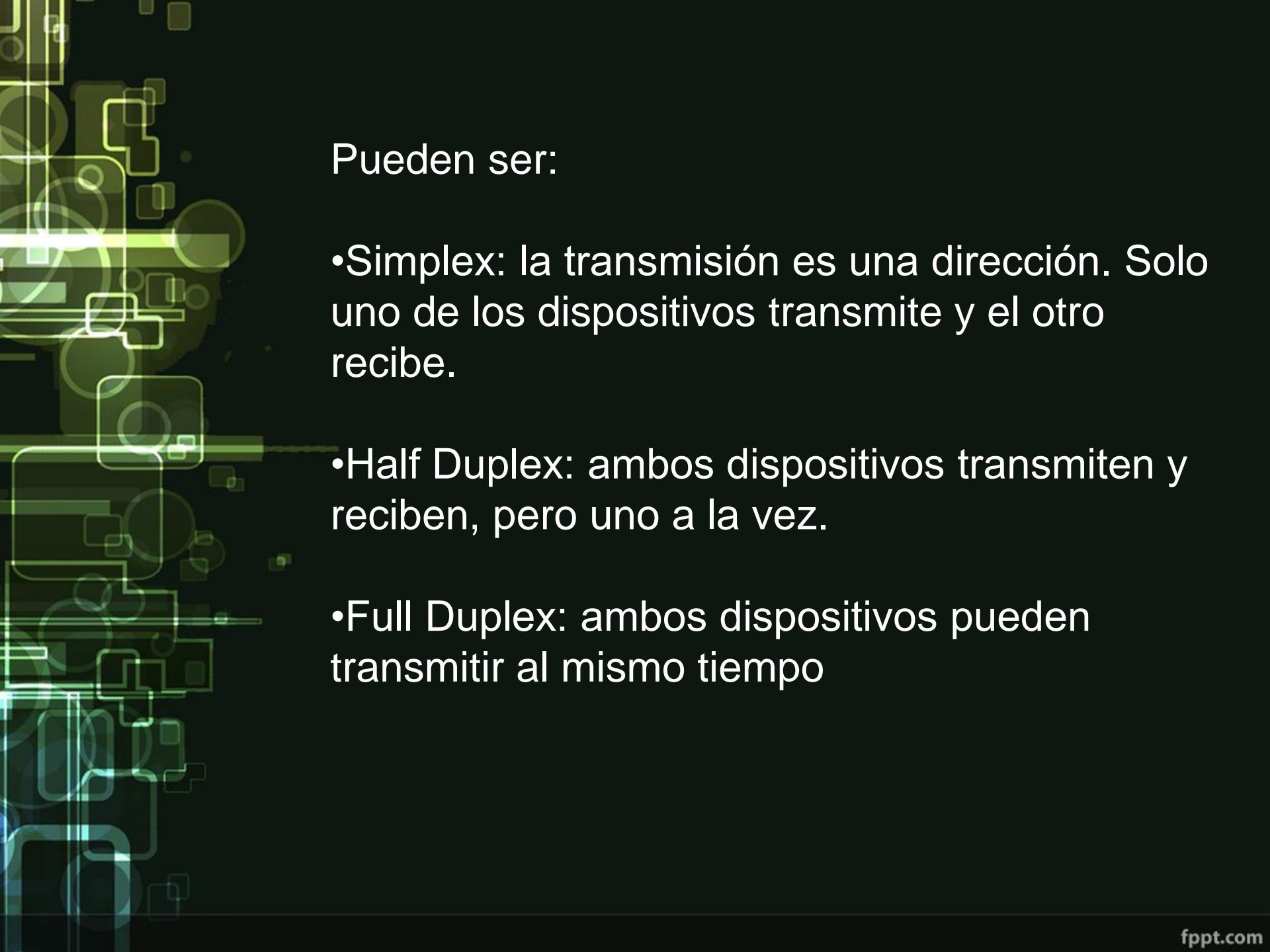




A menudo el microcontrolador necesita interactuar con periféricos y enviar o recibir información de los mismos. Si estos bytes de datos fueran enviados en paralelo, sería necesario al menos un puerto completo de 8 bits. Esto significaría consumir muchos recursos.

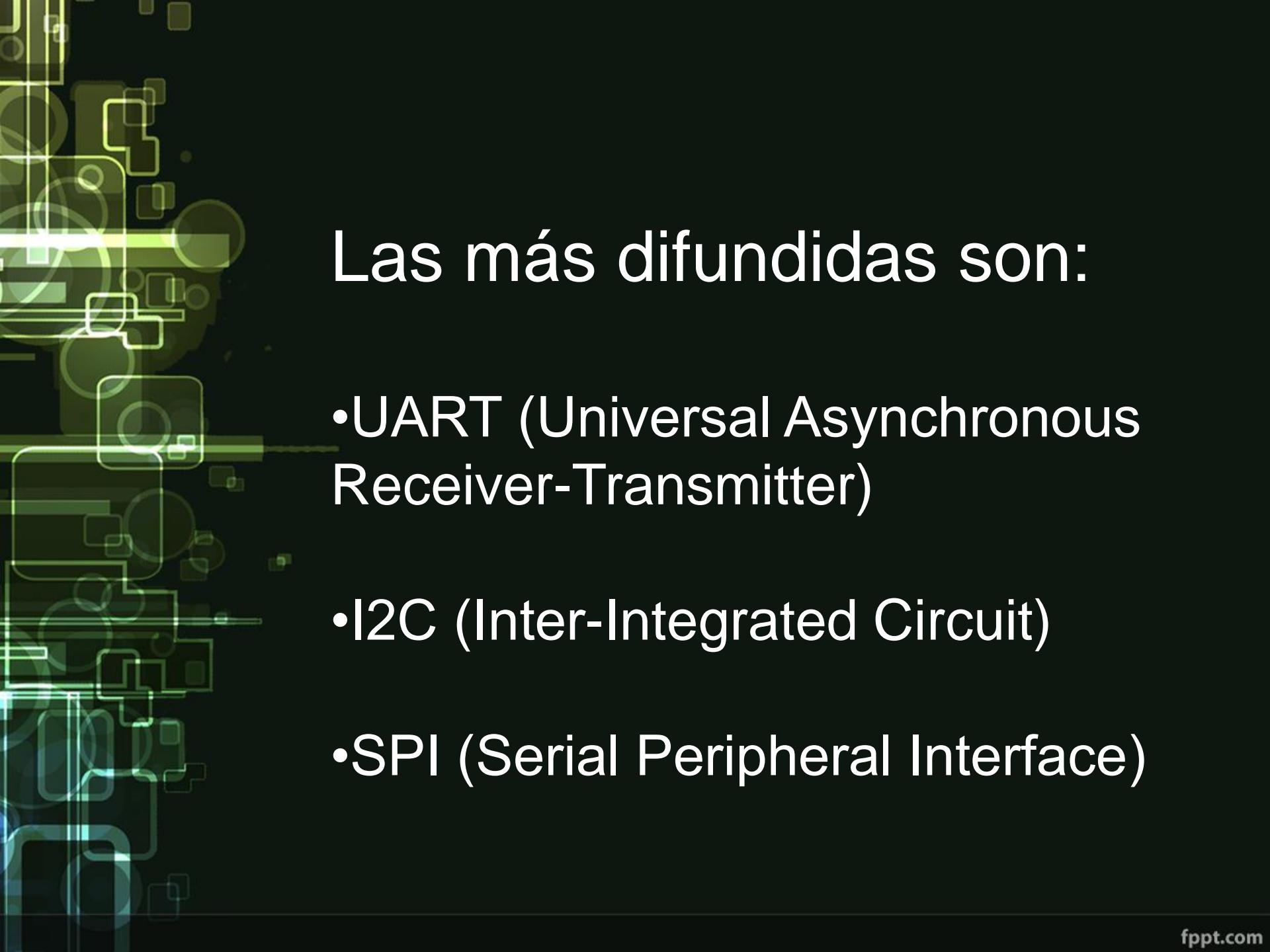


Afortunadamente muchos de los periféricos tienen hoy en día distintos tipos de comunicación serie. Este tipo de comunicación consiste simplemente en enviar cada byte bit por bit, para que luego en el lado receptor se reconstruya nuevamente el byte original. Para esto se utilizan técnicas con registros de desplazamiento. La transmisión puede ser sincrónica o asincrónica, y la cantidad de líneas de conexión dependerá del protocolo usado.



Pueden ser:

- Simplex: la transmisión es una dirección. Solo uno de los dispositivos transmite y el otro recibe.
- Half Duplex: ambos dispositivos transmiten y reciben, pero uno a la vez.
- Full Duplex: ambos dispositivos pueden transmitir al mismo tiempo



# Las más difundidas son:

- UART (Universal Asynchronous Receiver-Transmitter)
- I2C (Inter-Integrated Circuit)
- SPI (Serial Peripheral Interface)

# UART

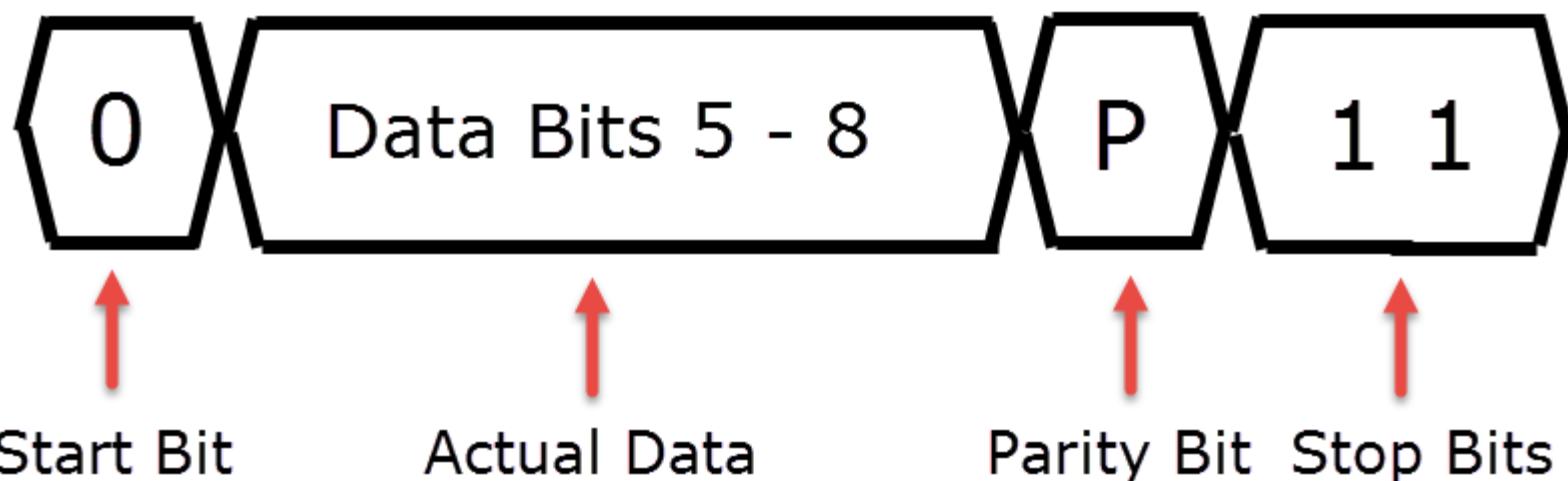
(Universal Asynchronous Receiver-Transmitter)

Transmisor-Receptor Asincrónico Universal. Este tipo de comunicación utiliza 2 líneas de datos, una de transmisión y otra de recepción. Es asincrónica ya que no se utiliza ninguna de línea de reloj. La velocidad de transmisión debe ser acordada antes de iniciar la comunicación

# UART

En estado normal u ocioso, la línea de transmisión estéan alto. Para comenzar la transmisión el TX genera el bitde START poniendo la línea en estado bajo durante un pulso de clock. A continuación envía los bits de datos, seguidos de un bitde paridad y del (o los) bits de stop.

## UART Packet



# UART

El micro Atmega128 posee 2 USART (Universal Synchronous and Asynchronous serial Receiver and Transmitter), que además del modo UART permiten una comunicación sincrónica. Ambos, USART0 y USART1, permiten ser configurados en cuanto a la cantidad de bits de datos, paridad par e impar, bits stop y velocidad de transmisión.

# UART

## Registros:

URDn: USART I/O Data Register. Atmgea128 posee registros de transmisión y recepción separados, pero que comparten la misma dirección. Así, cuando se lee UDRn se accede el registro de recepción. En cambio cuando se escribe UDRn, se accede al registro de transmisión.

Bit	7	6	5	4	3	2	1	0	UDRn (Read)	UDRn (Write)
Read/Write	R/W									
Initial Value	0	0	0	0	0	0	0	0		

# UART

UCSRnA: USART Control and Status Register A: Contiene bits de control y status.

Bit	7	6	5	4	3	2	1	0	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

RXCn: USART ReceiveComplete. Se pone en 1 cuando hay datos disponibles en el buffer de recepción.

TXCn: USART TransmitComplete. Se pone en 1 cuando se transmitió una trama completa y no hay mas datos para enviar en el buffer.

UDREn: USART Data Register Empty. Este bit indica, cuando estén 1, que el buffer de transmisión está vacío.

Los 3 pueden producir interrupciones si están habilitadas.

# UART

Bit	7	6	5	4	3	2	1	0	UCSRnA
Read/Write	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	
Initial Value	0	0	1	0	0	0	0	0	

FEn: Frame Error. Este bit indica cuando se produce un error de trama recibida.

DORn: Data Over Run. Este bit se pone a uno cuando hay saturación. Sigue cuando el buffer está lleno y se detecta condición de START.

UPEn: Parity Error. Se pone a uno cuando hay error en la paridad.

U2Xn: Doble de USART Transmission Speed. Para modo asincrónico.

MPCMn. Multiprocesor Comunication Mode. Habilita el modo de comunicación multiprocesador cuando se escribe un 1.

# UART

UCSRnB: USART Control and status Register B: Contiene bits de control y status.

Bit	7	6	5	4	3	2	1	0	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

RXCIEn, TXCIEny UDRIEn: permiten habilitar las interrupciones.

RXENn: ReceiverEnable. Permite habilitar la recepción (1=En).

TXENn: TransmitterEnable. Permite habilitar la transmisión (1=En).

UCSZn2: CharacterSize. Define la cantidad de bits junto con UCSZn1 u UCSZn0.

RXB8ny TXB8n: 9no bitrecibido o a transmitir.

# UART

UCSRnC: USART Control and Status RegisterC: Contiene bits de control y status.

Bit	7	6	5	4	3	2	1	0	UCSRnC
Read/Write	-	R/W	UCSRnC						
Initial Value	0	0	0	0	0	1	1	0	

UMSELn: USART ModeSelect. 1 = Sincrónico. 0 = Asincrónico

UPMn1:0: ParityMode. Definen la paridad.

USBSn: Stop BitSelect. Define la cantidad de bits de stop. 0=1bit y 1=2 bits.

UCSZn1:0: CharacterSize. Definen junto a UCSZn2 la cantidad de bits.

UCPOLn: Clock Polarity. Define la polaridad del clock para el modo sincrónico.

# UART

- UBRRn: USART BaudRateRegister: Definen el baud rate.

Bit	15	14	13	12	11	10	9	8	UBRRnH	UBRRnL			
	-	-	-	-	UBRRn[11:8]								
					UBRRn[7:0]								
	7	6	5	4	3	2	1	0					
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W					
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W					
Initial Value	0	0	0	0	0	0	0	0					
	0	0	0	0	0	0	0	0					

Operating Mode	Equation for Calculating Baud Rate <sup>(1)</sup>	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

Como transmito  
los datos?

# UART

- 1 primero configuro el Baud Rate
- 2 habilito la transmisión
- 3 verificar no haya TX en curso (UDRE)
- 4 escribir buffer de transmisión UDR



# UART

## EJEMPLO

```
while(! UDRE);
```

```
UDR0=dato;
```

**Como recibo  
los datos?**

# UART

- 1 primero configuro el Baud Rate
- 2 habilito la recepción
- 3 verificar estado flag recepción (RXC)
- 4 leer el buffer de recepción UDR

UART

# EJEMPLO

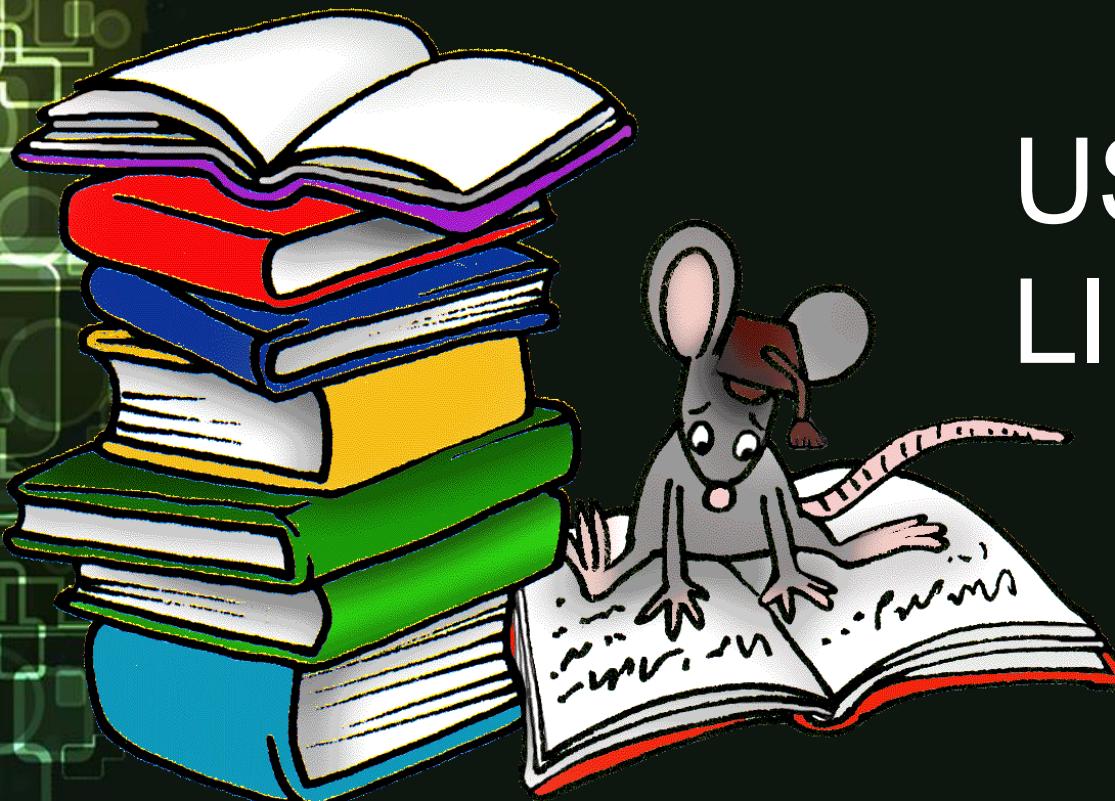
```
while(! RXC);
```

```
dato=UDR;
```

UART

SIMPLIFICANDO  
NUESTRO TRABAJO.....

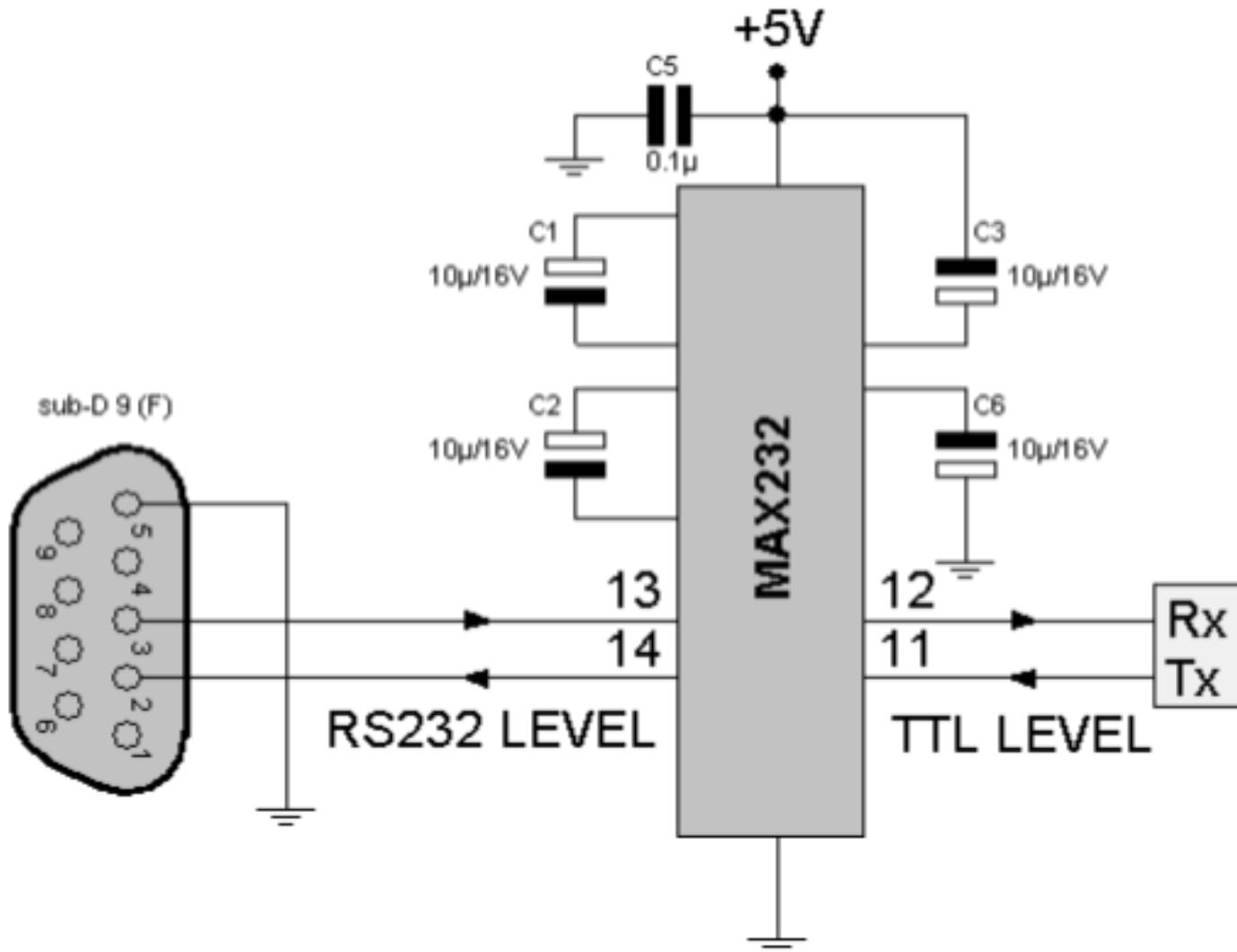
USAREMOS  
LIBRERIAS



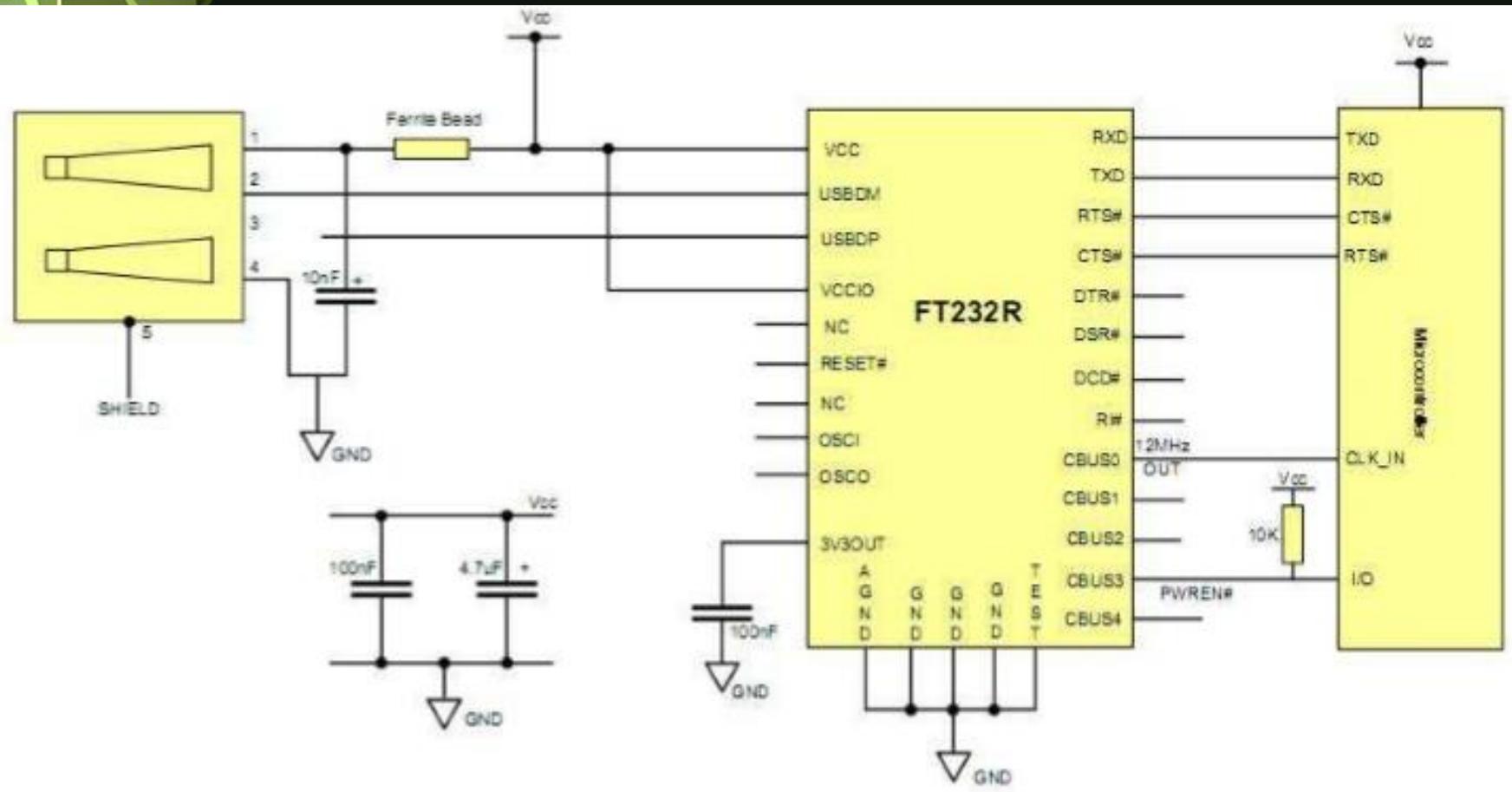
- Las alternativas anteriores se basan en hacer **pooling** de los registros de la UART, lo cual no siempre es óptimo.

- En estos casos es posible trabajar por **interrupciones**.

# Conexión a la PC por puerto RS-232



# Conexión a la PC por puerto USB



# Inter-Integrated Circuit

I2C

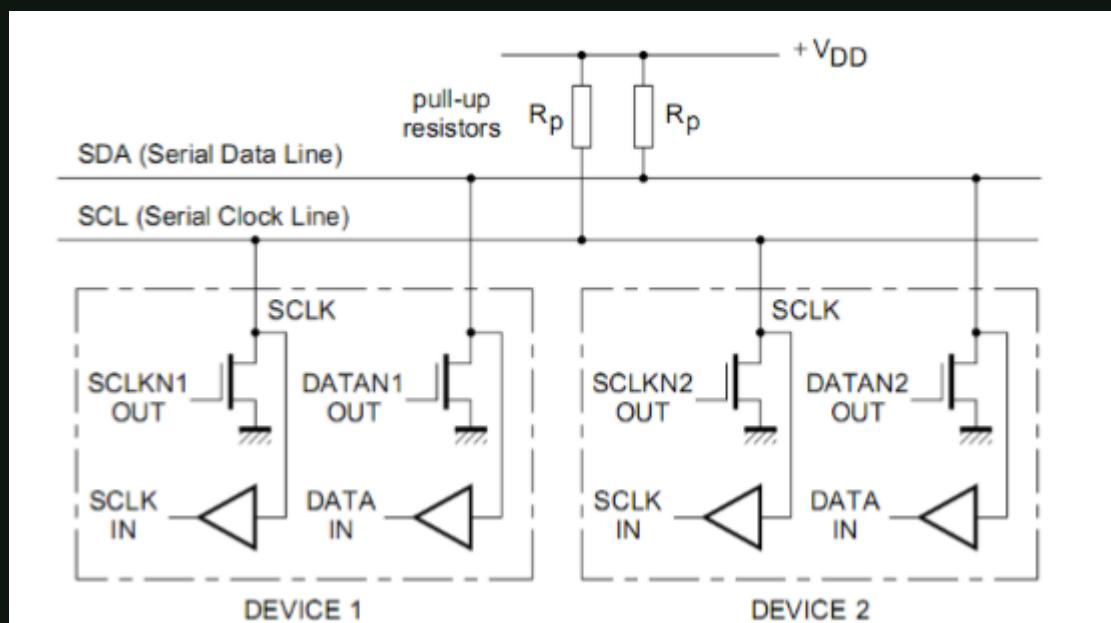
Este protocolo de comunicación fue diseñado por Phillips con la idea de hacer más simple a nivel hardware la conexión de los procesadores a los periféricos. Consiste en un bus de dos líneas de datos bidireccionales y puede alcanzar velocidades de hasta 100Kbit/s en el modo estándar y hasta 400Kbit/s en el Fast Mode. Es un tipo de comunicación Master –Slave

# Inter-Integrated Circuit

## Capa física.

I2C

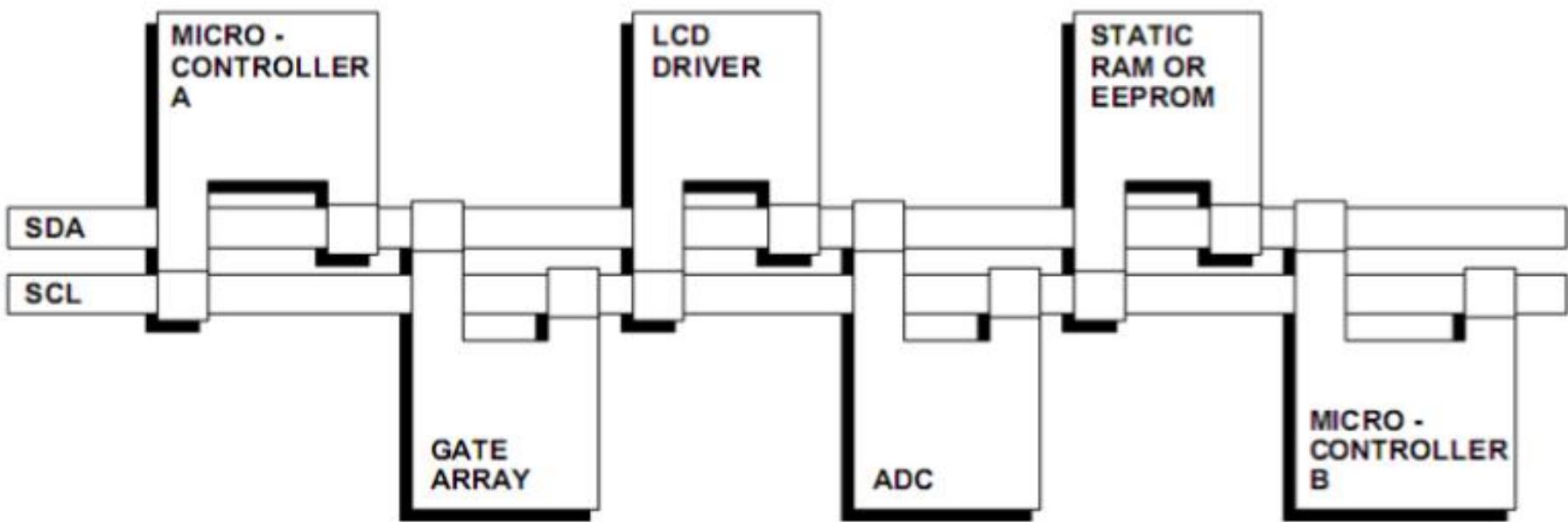
El bus consiste de dos líneas: una de clock y otra de datos. Ambas líneas del bus son open-drain y llevan siempre resistores de pull-up, de manera que en estado ocioso ambas están en nivel alto. Los valores típicos de resistencia de pull-up van entre 10K y 47K.



# Inter-Integrated Circuit

I2C

Generalmente hay un único Master que arbitra el bus y es el encargado de generar la señal de reloj. Todos los periféricos que se encuentran conectados al bus reciben dicho clock. La comunicación comienza cuando el master genera la condición de START y termina cuando genera la condición de STOP.

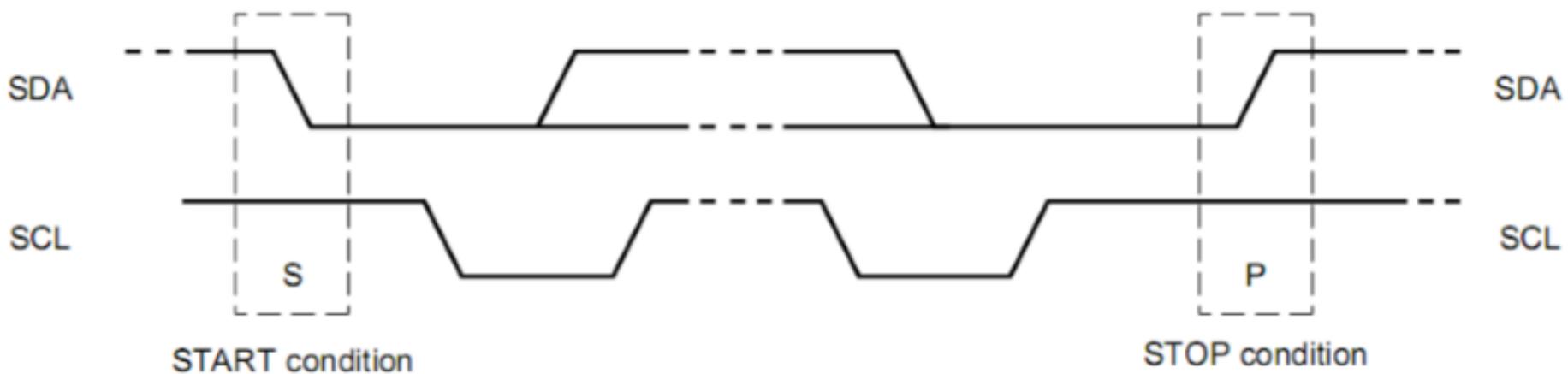


# Inter-Integrated Circuit

## Condiciones de Start y Stop

I2C

La condición se Start se produce cuando el master genera una transición de alto a bajo en SDA mientras SCL está en alto. La condición de Stop se produce cuando el master genera una transición de bajo a alto mientras SCL esta en estado alto. Durante la comunicación las transiciones de SDA se hacen mientras SCL está a nivel bajo.



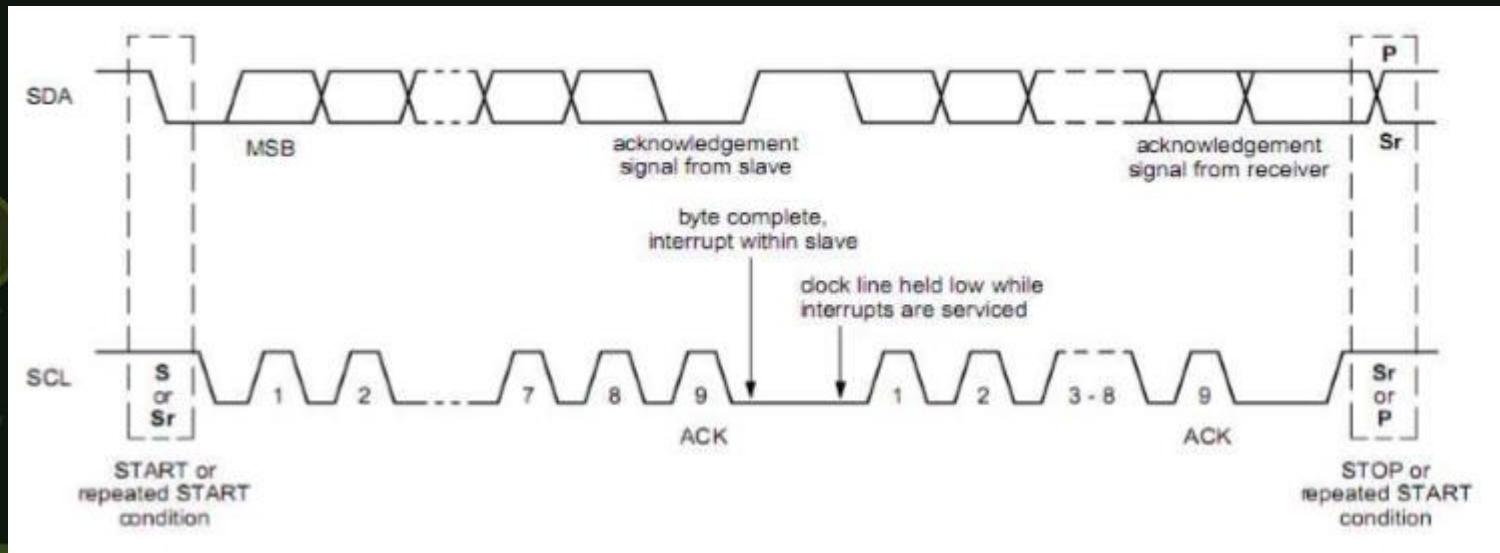
# Inter-Integrated Circuit

I2C

Habiendo tantos periféricos que reciben los mismos datos y clock, como se identifica al receptor de la información? Más bien, como sabe un SLAVE que los datos que vienen le corresponden a él?

Se utiliza una dirección que va intercalada en la TRAMA de datos. Cada periférico conectado al bus debe tener una dirección distinta.

## FORMATO DE TRAMA:



El Master genera la condición de Start.

El Master envía el byte a transmitir.

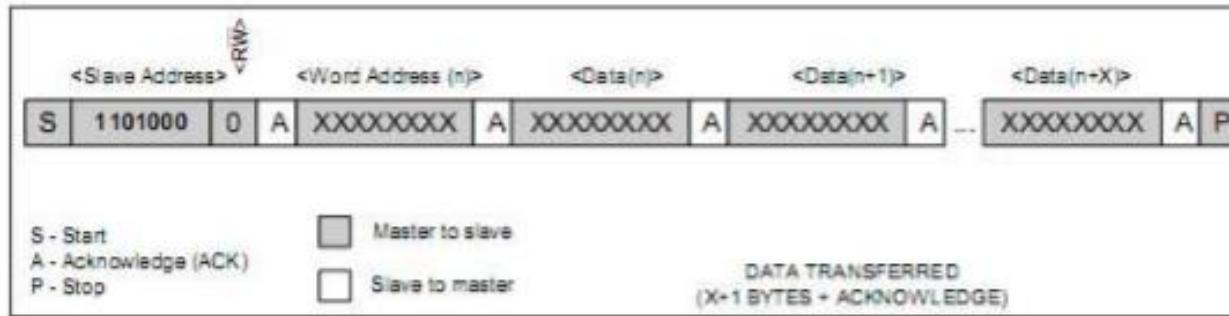
El Slave genera el reconocimiento (ACK) mientras SCL está en alto.

Si el Slave no puede transmitir/recibir otro byte puede poner la SCL en bajo para forzar al Master a esperar.

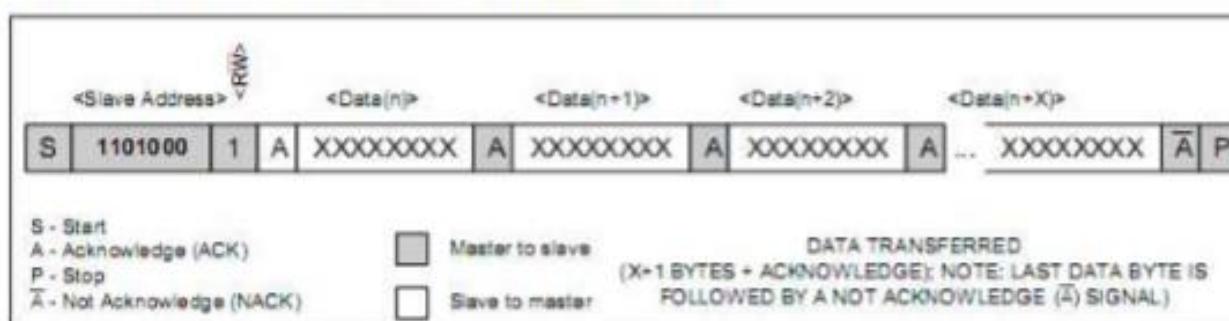
Transmitido/recibido el byte, y recibido/enviado el ACK, el Master envía señal de STOP

# Trama con dirección Fija: DS1307

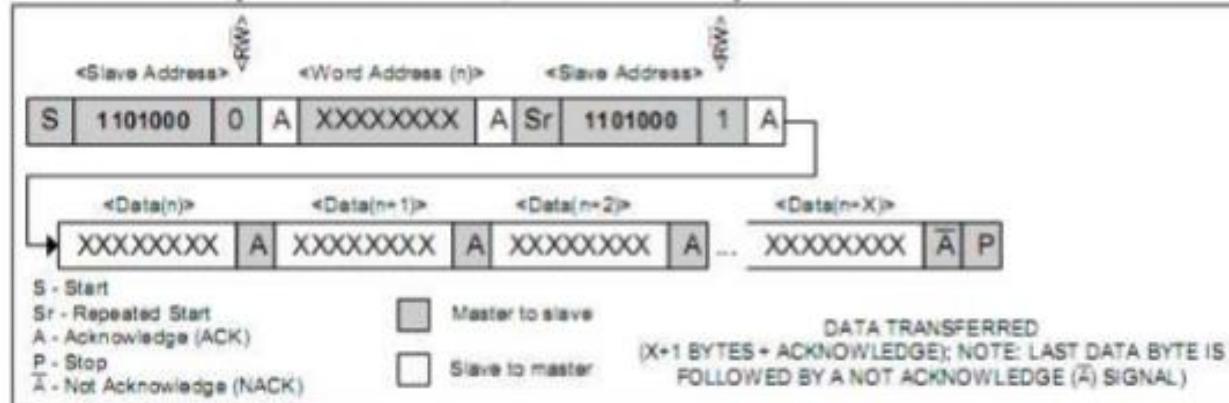
## Data Write—Slave Receiver Mode



## Data Read—Slave Transmitter Mode



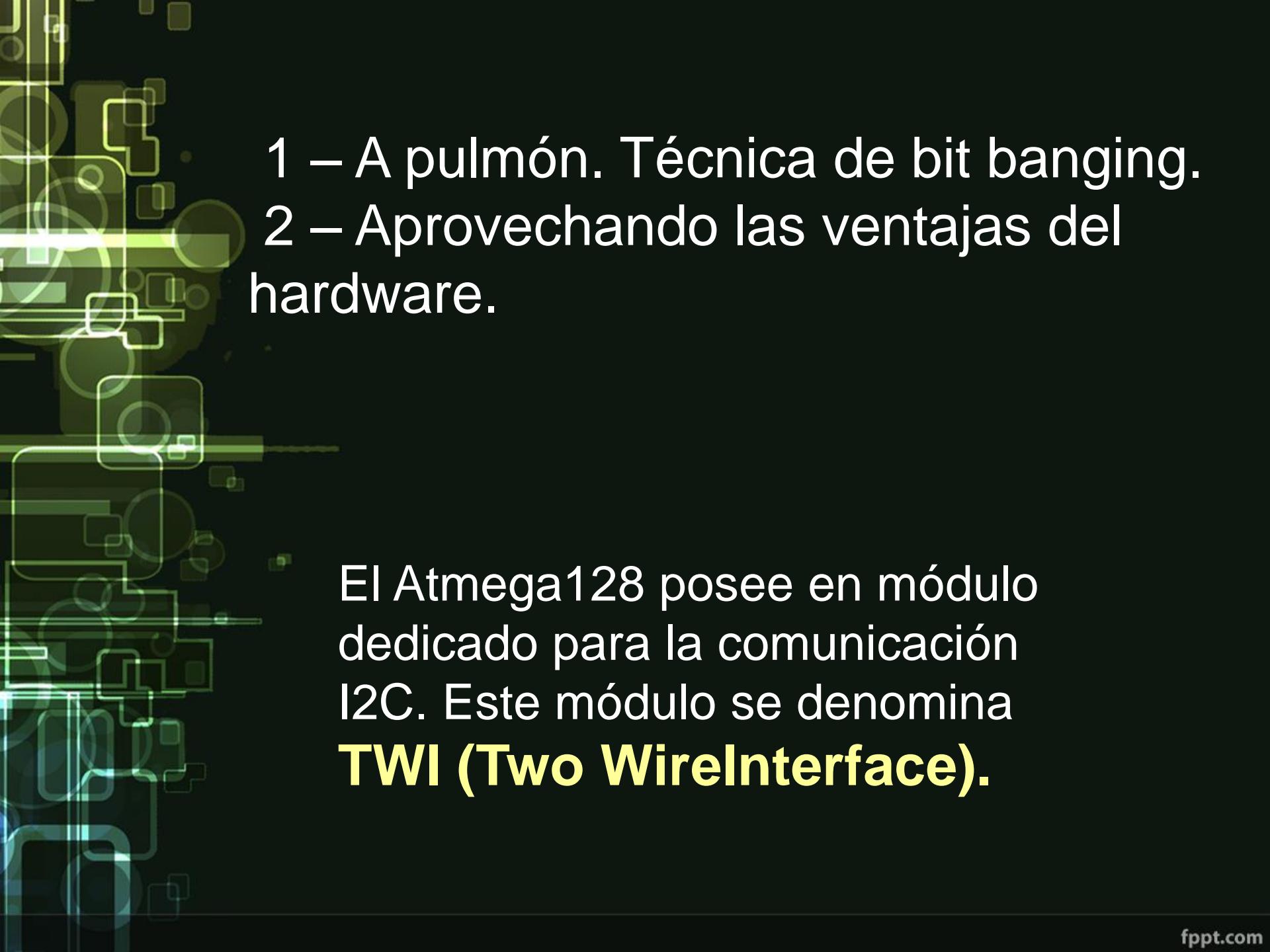
## Data Read (Write Pointer, Then Read)—Slave Receive and Transmit





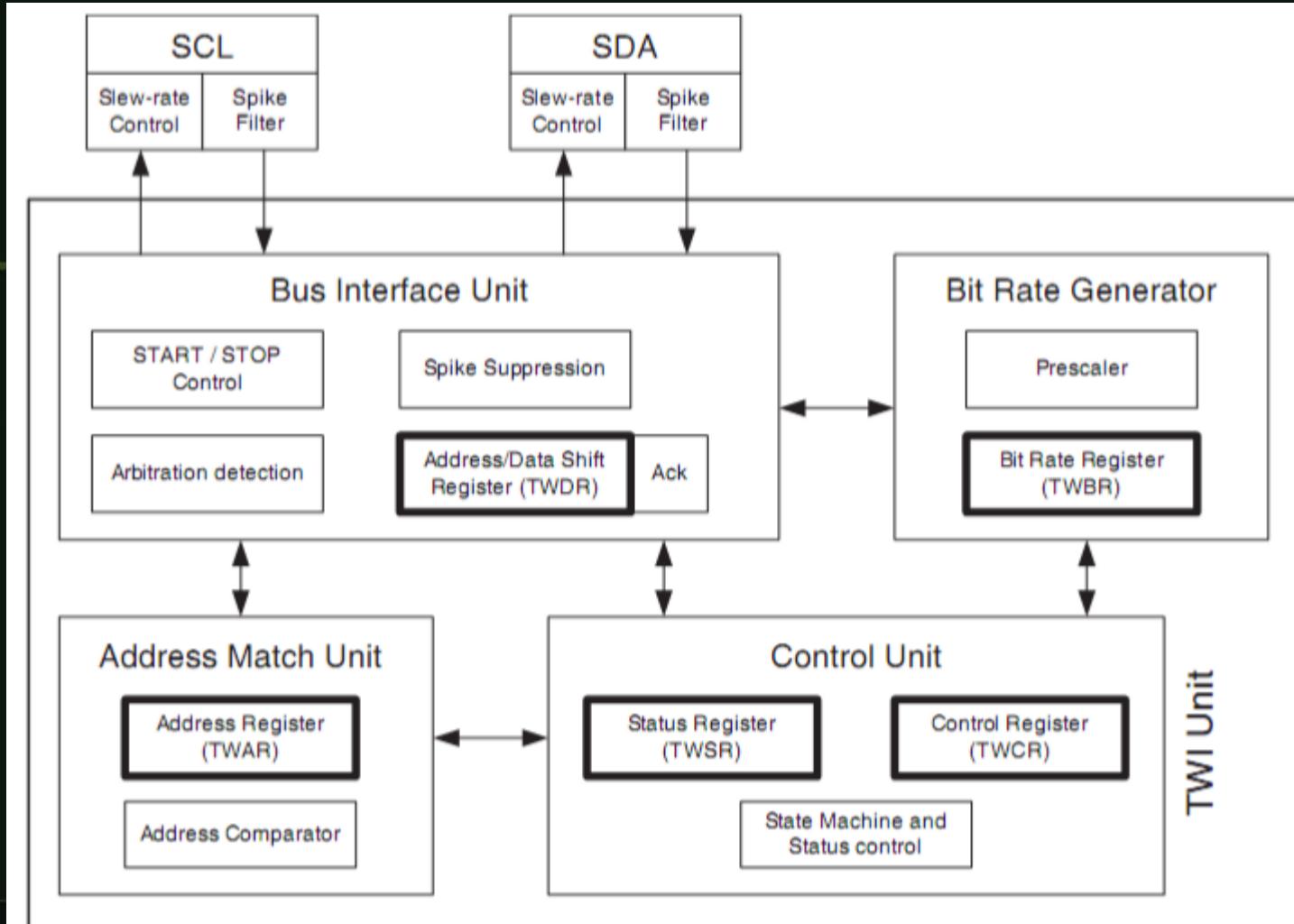
Cómo  
implementamos  
este tipo de  
comunicación  
en un Micro



- 
- 1 – A pulmón. Técnica de bit banging.
  - 2 – Aprovechando las ventajas del hardware.

El Atmega128 posee en módulo dedicado para la comunicación I2C. Este módulo se denomina **TWI (Two Wire Interface)**.

# Two Wire Interface (TWI)





En el curso usaremos  
Funciones para controlar  
nuestro RTC por **TWI**