

# Informática II

## Programación del microcontrolador ATmega328

Gonzalo F. Perez Paina

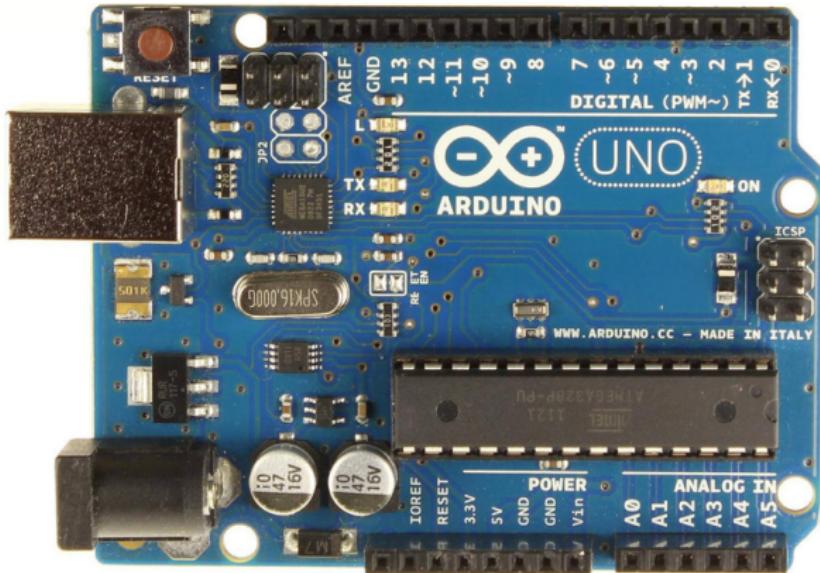


Universidad Tecnológica Nacional  
Facultad Regional Córdoba  
UTN-FRC

– 2021 –

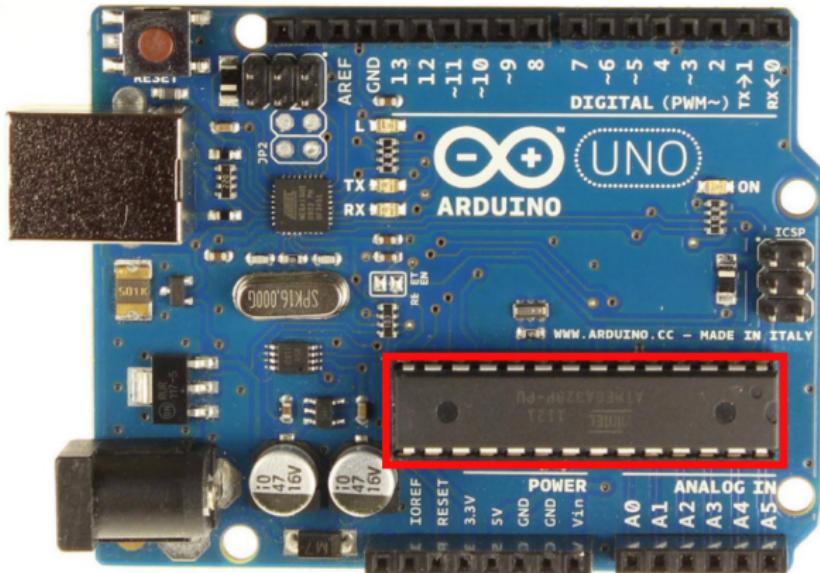
# Arduino UNO Rev3

La placa Arduino UNO tiene un microcontrolador ( $\mu$ C) marca Microchip® (anteriormente Atmel®) modelo ATmega328.



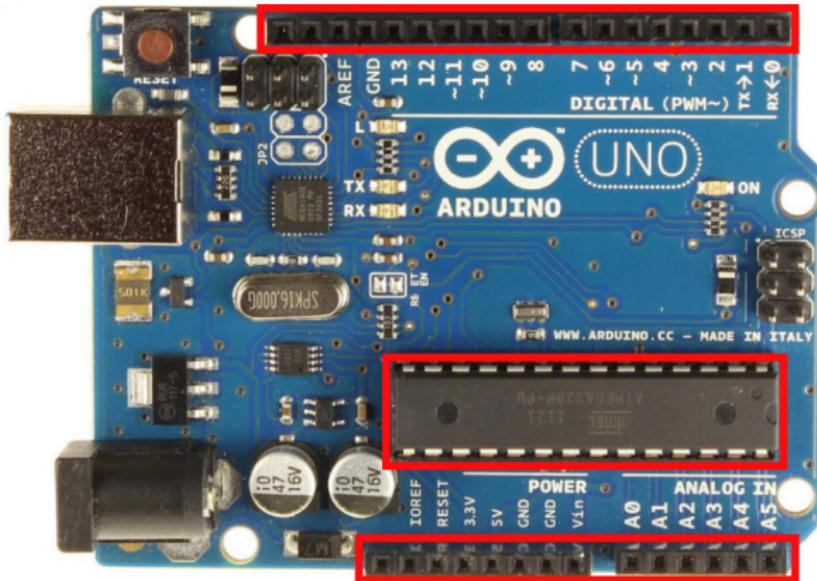
# Arduino UNO Rev3

La placa Arduino UNO tiene un microcontrolador ( $\mu$ C) marca Microchip® (anteriormente Atmel®) modelo ATmega328.



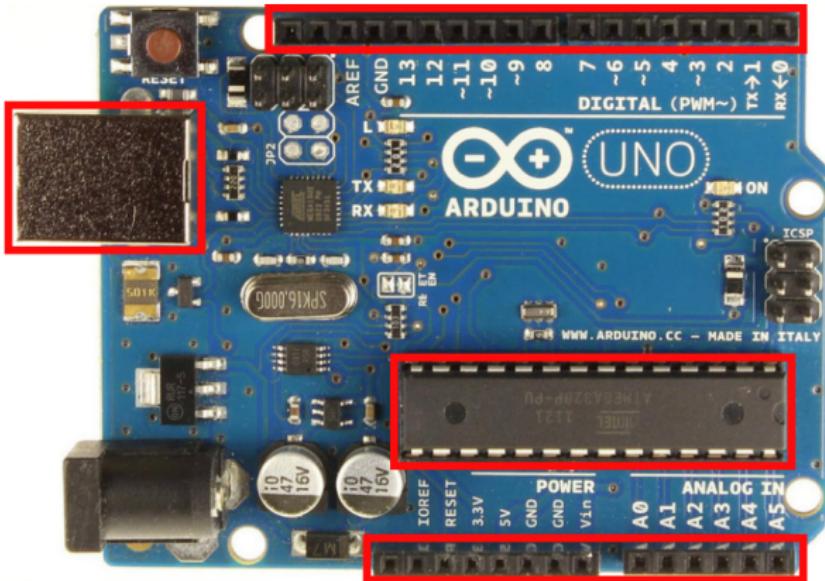
# Arduino UNO Rev3

La placa Arduino UNO tiene un microcontrolador ( $\mu$ C) marca Microchip® (anteriormente Atmel®) modelo ATmega328.



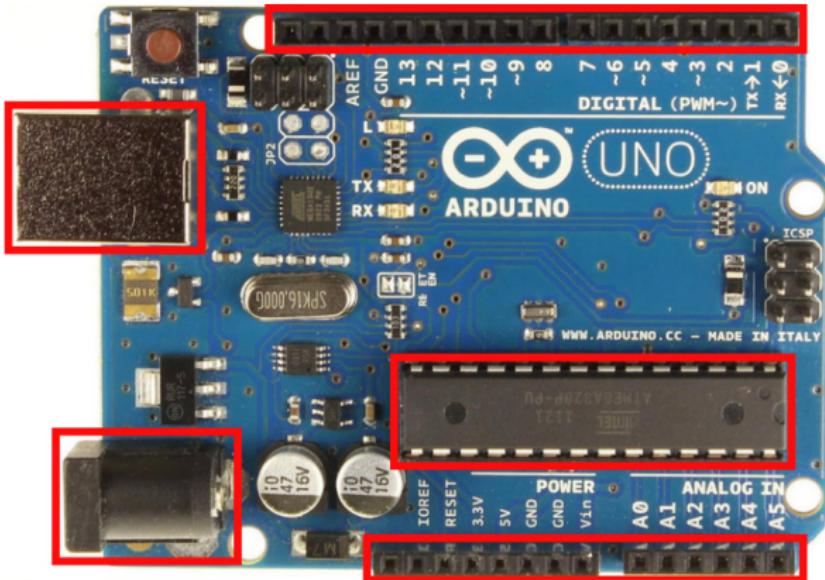
# Arduino UNO Rev3

La placa Arduino UNO tiene un microcontrolador ( $\mu$ C) marca Microchip® (anteriormente Atmel®) modelo ATmega328.



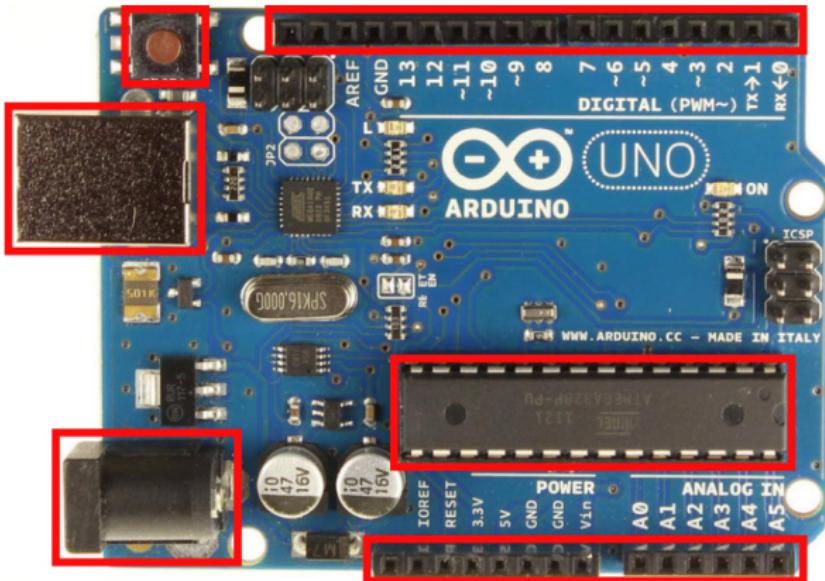
# Arduino UNO Rev3

La placa Arduino UNO tiene un microcontrolador ( $\mu$ C) marca Microchip® (anteriormente Atmel®) modelo ATmega328.



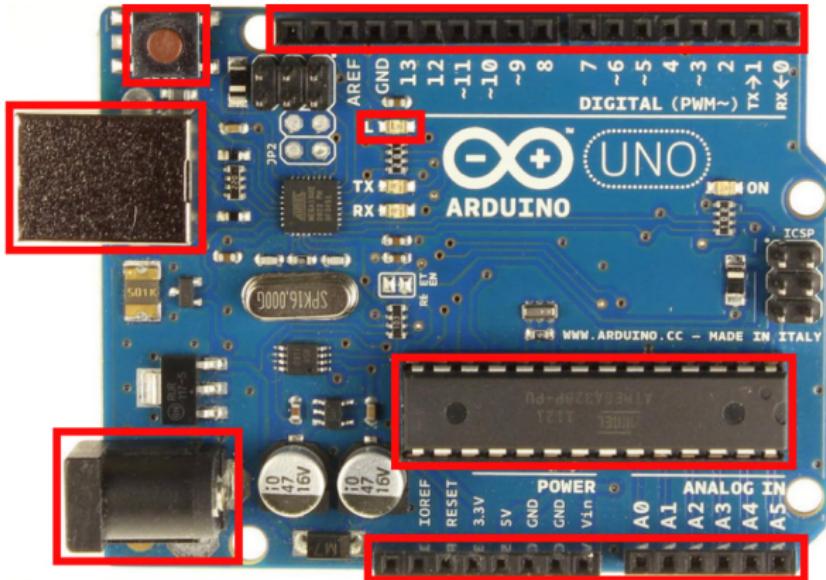
# Arduino UNO Rev3

La placa Arduino UNO tiene un microcontrolador ( $\mu$ C) marca Microchip® (anteriormente Atmel®) modelo ATmega328.



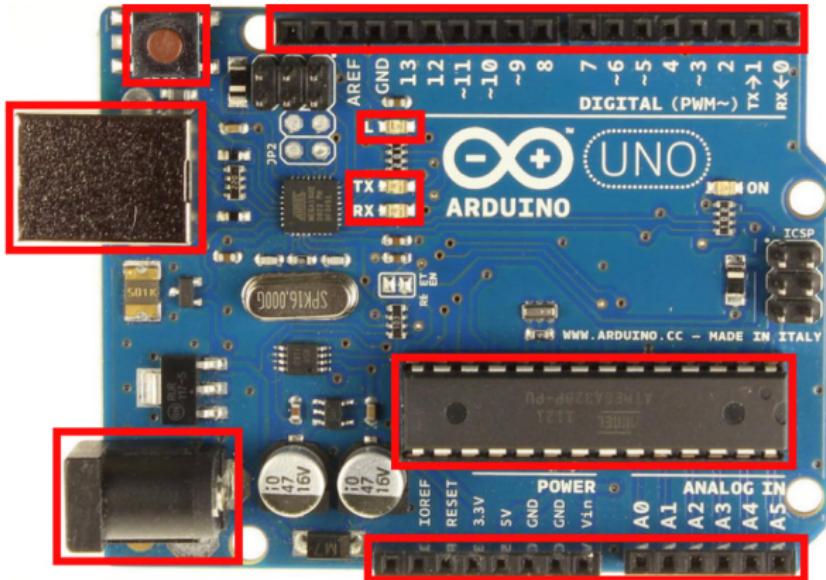
# Arduino UNO Rev3

La placa Arduino UNO tiene un microcontrolador ( $\mu$ C) marca Microchip® (anteriormente Atmel®) modelo ATmega328.



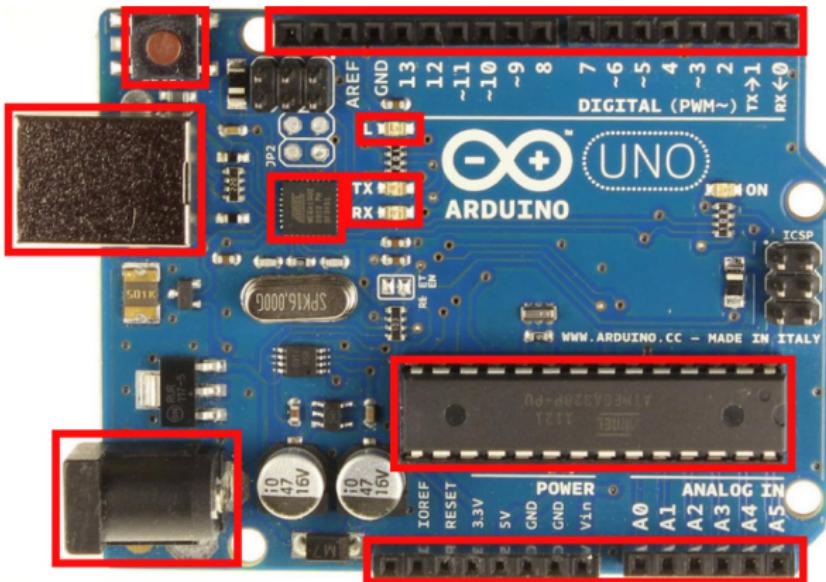
# Arduino UNO Rev3

La placa Arduino UNO tiene un microcontrolador ( $\mu$ C) marca Microchip® (anteriormente Atmel®) modelo ATmega328.



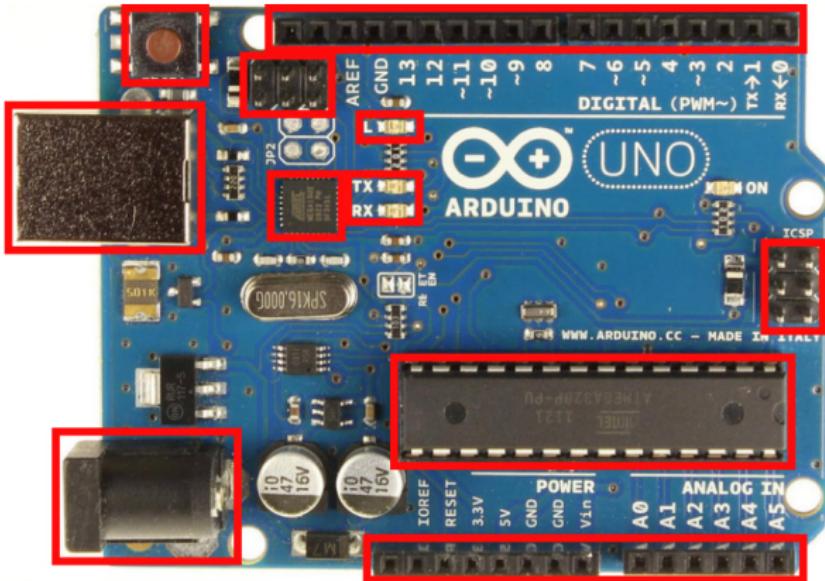
# Arduino UNO Rev3

La placa Arduino UNO tiene un microcontrolador ( $\mu$ C) marca Microchip® (anteriormente Atmel®) modelo ATmega328.



# Arduino UNO Rev3

La placa Arduino UNO tiene un microcontrolador ( $\mu$ C) marca Microchip® (anteriormente Atmel®) modelo ATmega328.

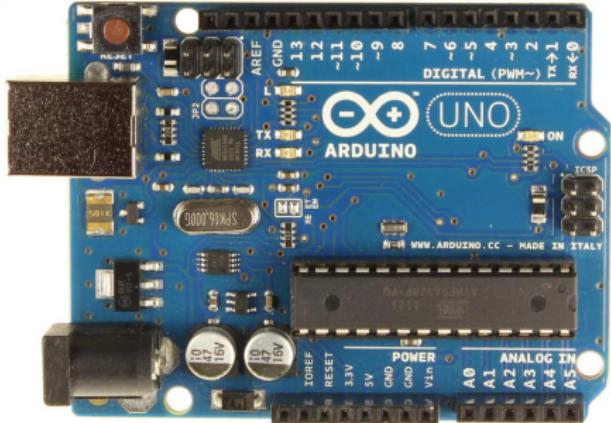


# Arduino UNO Rev3 – Algunas características

Algunas de las características de esta placa son:

- ▶ Tensión de funcionamiento de 5V
- ▶ Tensión de entrada de 7 a 12V (de 6 a 20V como rango máximo)
- ▶ 14 pines digitales de entrada/salida
- ▶ 6 pines digitales con función PWM
- ▶ 6 pines de entrada analógica

# Arduino UNO – Versiones



Arduino UNO Rev3:

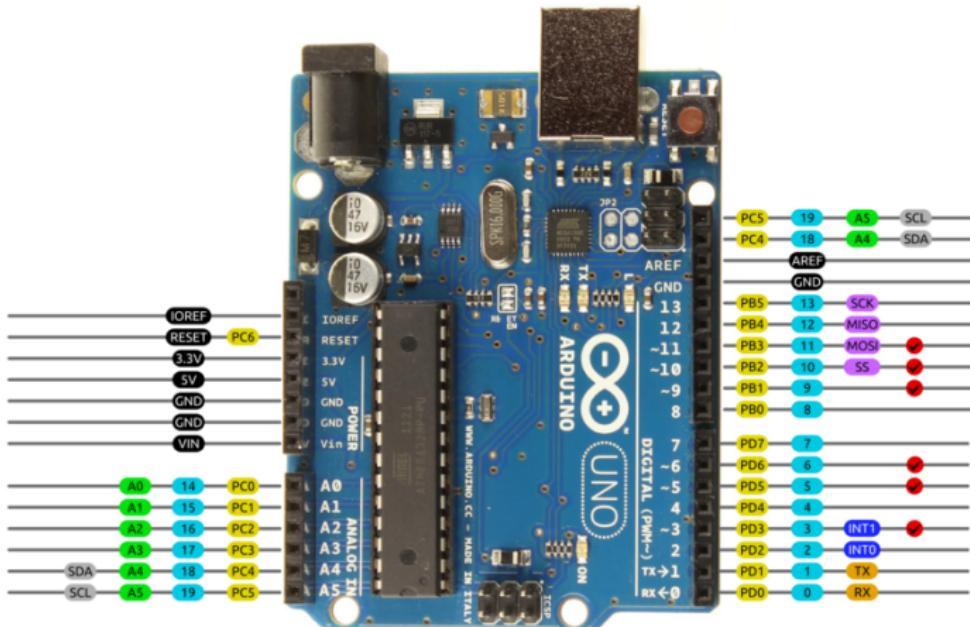
- ▶ ATmega328P DIP (Dual In-line Package)
- ▶ ATmega16U2  $\mu$ C con controlador USB, QFN (Quad-flat No-leads)



Arduino UNO CH340:

- ▶ ATmega328P TQFP (Thin Quad Flat Package)
- ▶ CH340, Chip USB a serial

# Entrada/salida (pinout) y periféricos



AVR DIGITAL ANALOG POWER SERIAL SPI I2C PWM INTERRUPT



2014 by Bouni  
Photo by Arduino.cc

# El ATmega328

Es un microcontrolador de 8-bits de arquitectura AVR RISC mejorado.  
Algunas de las características de este  $\mu$ C son:

- ▶ Arquitectura RISC avanzada:
  - ▶ 131 instrucciones (la mayoría de las cuales se ejecutan en un ciclo de reloj)
  - ▶ 32 registros de propósitos generales de 8-bits
  - ▶ Máximo de 20MIPS (million instructions per second) @ 20MHz de frecuencia de reloj
  - ▶ Multiplicación por hardware

# El ATmega328

Es un microcontrolador de 8-bits de arquitectura AVR RISC mejorado.  
Algunas de las características de este  $\mu$ C son:

- ▶ Arquitectura RISC avanzada:
  - ▶ 131 instrucciones (la mayoría de las cuales se ejecutan en un ciclo de reloj)
  - ▶ 32 registros de propósitos generales de 8-bits
  - ▶ Máximo de 20MIPS (million instructions per second) @ 20MHz de frecuencia de reloj
  - ▶ Multiplicación por hardware
- ▶ Memoria:
  - ▶ 32KB de memoria FLASH
  - ▶ 2KB de memoria SRAM (Static Random Access Memory)
  - ▶ 1KB de memoria EEPROM (Electrically Erasable Programmable Read-Only Memory)

# El ATmega328

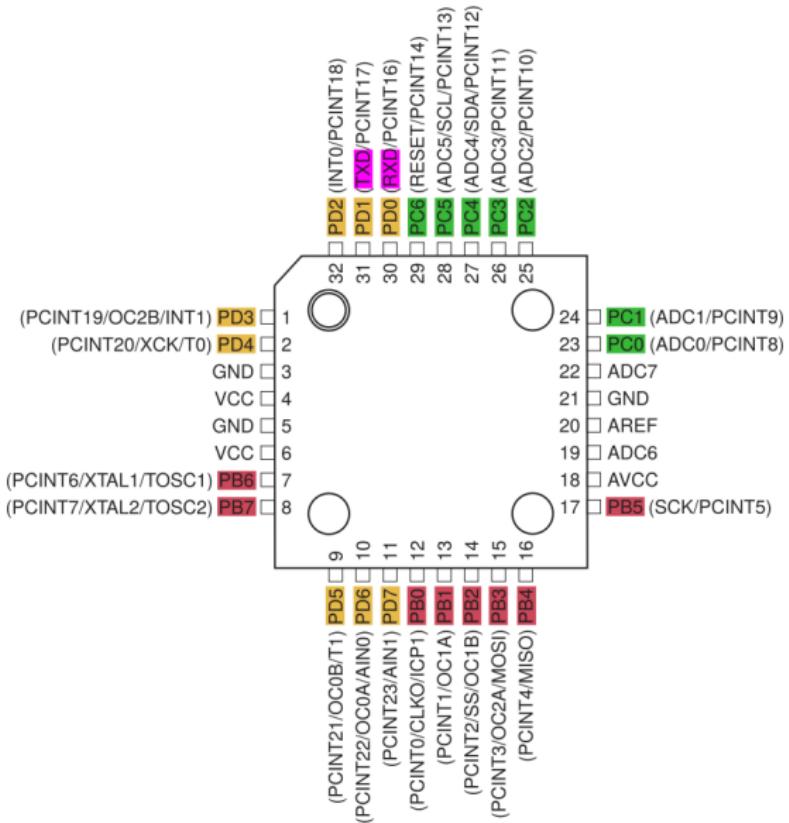
- ▶ Periféricos:
  - ▶ 23 entrada-salida de propósito general (GPIO: General Purpose Input-Output)
  - ▶ 6 canales de PWM (Pulse Width Modulation)
  - ▶ 6/8 canales de ADC (Analog to Digital Converter) de 10-bits
  - ▶ Puerto serial programable (USART: Universal Synchronous/Asynchronous Receiver/Transmitter)
  - ▶ Interfaz serial SPI (Serial Peripheral Interface) maestro-esclavo
  - ▶ Interfaz serial de 2-cables (compatible con el I<sup>2</sup>C de Philips)
  - ▶ Interrupción externa por cambio de nivel en entrada digital

# El ATmega328

28 PDIP	
(PCINT14/RESET) <b>PC6</b>	1
(PCINT16/ <b>RXD</b> ) <b>PD0</b>	2
(PCINT17/ <b>TXD</b> ) <b>PD1</b>	3
(PCINT18/INT0) <b>PD2</b>	4
(PCINT19/OC2B/INT1) <b>PD3</b>	5
(PCINT20/XCK/T0) <b>PD4</b>	6
VCC	7
GND	8
(PCINT6/XTAL1/TOSC1) <b>PB6</b>	9
(PCINT7/XTAL2/TOSC2) <b>PB7</b>	10
(PCINT21/OC0B/T1) <b>PD5</b>	11
(PCINT22/OC0A/AIN0) <b>PD6</b>	12
(PCINT23/AIN1) <b>PD7</b>	13
(PCINT0/CLKO/ICP1) <b>PB0</b>	14
	15
	16
	17
	18
	19
	20
	21
	22
	23
	24
	25
	26
	27
	28
<b>PC5</b> (ADC5/SCL/PCINT13)	
<b>PC4</b> (ADC4/SDA/PCINT12)	
<b>PC3</b> (ADC3/PCINT11)	
<b>PC2</b> (ADC2/PCINT10)	
<b>PC1</b> (ADC1/PCINT9)	
<b>PC0</b> (ADC0/PCINT8)	
GND	
AREF	
AVCC	
<b>PB5</b> (SCK/PCINT5)	
<b>PB4</b> (MISO/PCINT4)	
<b>PB3</b> (MOSI/OC2A/PCINT3)	
<b>PB2</b> (SS/OC1B/PCINT2)	
<b>PB1</b> (OC1A/PCINT1)	

# El ATmega328

## 32 TQFP Top View



# El ATmega328 – Memoria

Tiene dos espacios de memorias diferentes (arquitectura Harvard):

1. una es la memoria de programa y
2. otra la memoria de datos.

# El ATmega328 – Memoria

Tiene dos espacios de memorias diferentes (arquitectura Harvard):

1. una es la memoria de programa y
2. otra la memoria de datos.

## Memoria de programa (Flash)

- ▶ La memoria Flash, para almacenar el programa, es de  $16K \times 16\text{-bits}$  haciendo un total de 32KBytes ( $32K \times 8\text{-bits}$ ).
- ▶ El contador de programa (PC: Program Counter) es de 14-bits lo que permite direccionar 16K ubicaciones de la memoria de programa.

# El ATmega328 – Memoria

Tiene dos espacios de memorias diferentes (arquitectura Harvard):

1. una es la memoria de programa y
2. otra la memoria de datos.

## Memoria de programa (Flash)

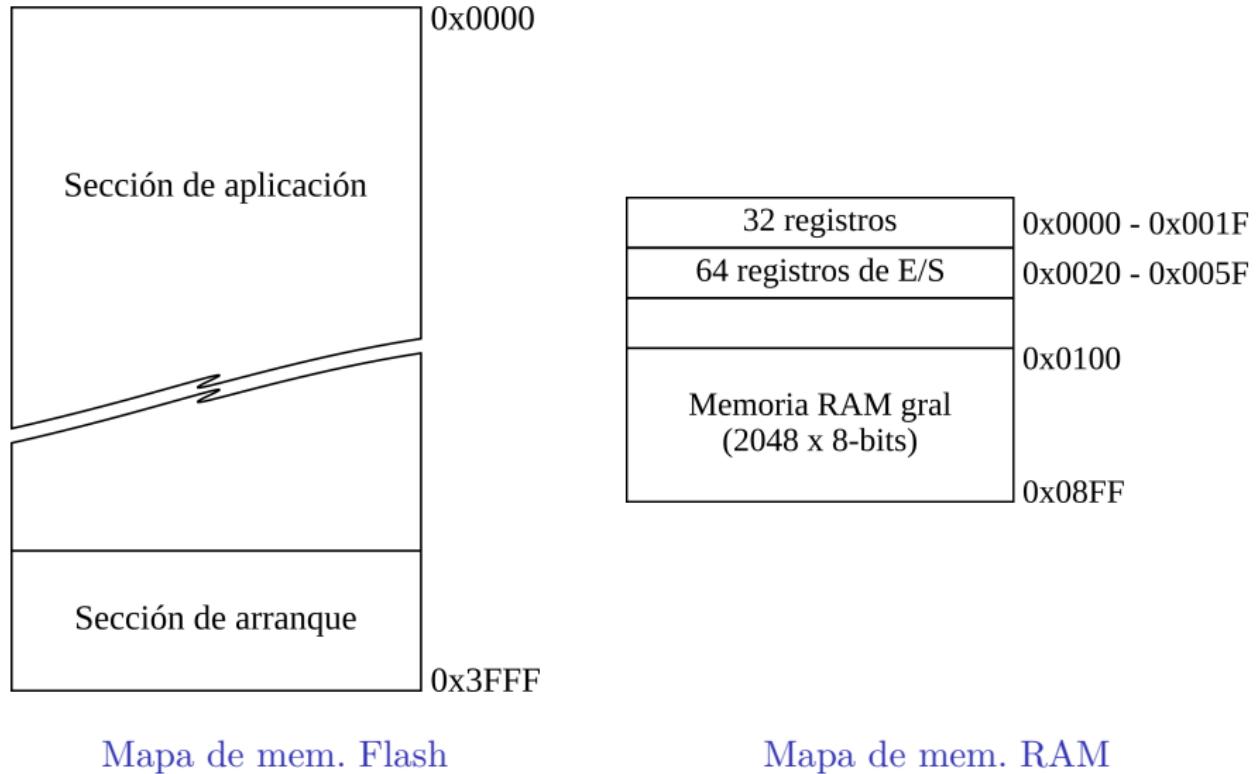
- ▶ La memoria Flash, para almacenar el programa, es de  $16K \times 16\text{-bits}$  haciendo un total de 32KBytes ( $32K \times 8\text{-bits}$ ).
- ▶ El contador de programa (PC: Program Counter) es de 14-bits lo que permite direccionar 16K ubicaciones de la memoria de programa.

## Memoria de programa (RAM)

La memoria RAM se organiza en diferentes secciones.

- ▶ Las primeras direcciones para acceder a los registros de propósito general y los de entrada-salida,
- ▶ las siguientes para almacenar las variables del programa.

# El ATmega328 – Memoria





# Puerto digital de entrada/salida

Para el manejo de los puertos digitales de entrada/salida se utilizan 3 registros: DDRx, PORTx y PINx (x: B, C o D).

# Puerto digital de entrada/salida

Para el manejo de los puertos digitales de entrada/salida se utilizan 3 registros: DDRx, PORTx y PINx (x: B, C o D).

- ▶ Los puertos digitales de entrada/salida suelen denominarse también como GPIO de General Purpose Input/Output.

# Puerto digital de entrada/salida

Para el manejo de los puertos digitales de entrada/salida se utilizan 3 registros: DDRx, PORTx y PINx (x: B, C o D).

- ▶ Los puertos digitales de entrada/salida suelen denominarse también como GPIO de General Purpose Input/Output.

## Registros

- ▶ DDRx: registro de dirección (Data Direction Register) del puerto x
- ▶ PORTx: registro de datos (Data Register) del puerto x
- ▶ PINx: registro de lectura (Input Pin Register) del puerto x

# Puerto digital de entrada/salida

Para el manejo de los puertos digitales de entrada/salida se utilizan 3 registros: DDRx, PORTx y PINx (x: B, C o D).

- ▶ Los puertos digitales de entrada/salida suelen denominarse también como GPIO de General Purpose Input/Output.

## Registros

- ▶ DDRx: registro de dirección (Data Direction Register) del puerto x
- ▶ PORTx: registro de datos (Data Register) del puerto x
- ▶ PINx: registro de lectura (Input Pin Register) del puerto x

Los registros de manejo de los puertos digitales de entrada/salida se encuentran mapeados en la memoria RAM de  $\mu$ C, o sea que tienen direcciones de memorias fijas.

# Puerto digital de entrada/salida

Registros para el manejo de puertos digitales de E/S

Bit	7	6	5	4	3	2	1	0	
	DDx7	DDx6	DDx5	DDx4	DDx3	DDx2	DDx1	DDx0	<b>DDR<sub>x</sub></b>
Read/Write	R/W								
Initial value	0	0	0	0	0	0	0	0	
	PORTx7	PORTx6	PORTx5	PORTx4	PORTx3	PORTx2	PORTx1	PORTx0	<b>PORT<sub>x</sub></b>
Read/Write	R/W								
Initial value	0	0	0	0	0	0	0	0	
	PINx7	PINx6	PINx5	PINx4	PINx3	PINx2	PINx1	PINx0	<b>PIN<sub>x</sub></b>
Read/Write	R/W								
Initial value	N/A								

# Puerto digital de entrada/salida

## Registros para el manejo de puertos digitales de E/S

Bit	7	6	5	4	3	2	1	0	
	DDx7	DDx6	DDx5	DDx4	DDx3	DDx2	DDx1	DDx0	
Read/Write	R/W								
Initial value	0	0	0	0	0	0	0	0	
	PORTx7	PORTx6	PORTx5	PORTx4	PORTx3	PORTx2	PORTx1	PORTx0	
Read/Write	R/W								
Initial value	0	0	0	0	0	0	0	0	
	PINx7	PINx6	PINx5	PINx4	PINx3	PINx2	PINx1	PINx0	
Read/Write	R/W								
Initial value	N/A								

Los bits DDxn del registro DDx seleccionan si el pin correspondiente actuará como entrada o salida:

- ▶ Si se escribe un 1 lógico a DDxn el pin Pxn se configura como salida.
- ▶ Si se escribe un 0 lógico a DDxn, el pin Pxn se configura como entrada.

# Puerto digital de entrada/salida

	PINx	DDRx	PORTx
<b>Puerto B</b>	0x23	0x24	0x25
<b>Puerto C</b>	0x26	0x27	0x28
<b>Puerto D</b>	0x29	0x2A	0x2B

Dirección de los registros de manejo de puertos digitales de E/S



# Puerto serie USART

El  $\mu$ C ATmega328 posee un periférico denominado USART (Universal Synchronous and Asynchronous Receiver and Transmitter) que permite la comunicación serie con otro dispositivo (p.e. una PC).

# Puerto serie USART

El  $\mu$ C ATmega328 posee un periférico denominado USART (Universal Synchronous and Asynchronous Receiver and Transmitter) que permite la comunicación serie con otro dispositivo (p.e. una PC).

- ▶ La USART del ATmega328 se denomina USART0.

# Puerto serie USART

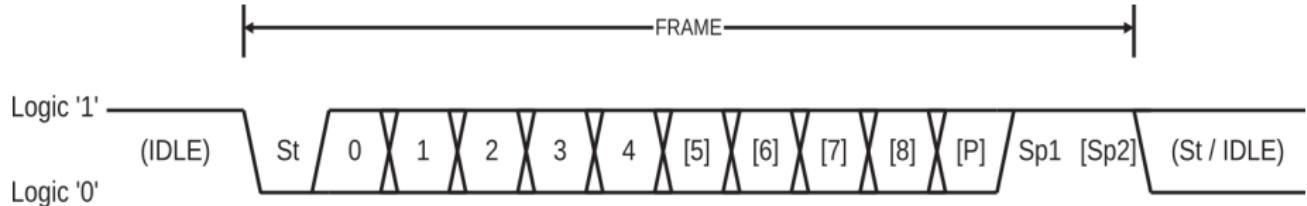
El  $\mu$ C ATmega328 posee un periférico denominado USART (Universal Synchronous and Asynchronous Receiver and Transmitter) que permite la comunicación serie con otro dispositivo (p.e. una PC).

- ▶ La USART del ATmega328 se denomina USART0.

## Algunas características

- ▶ Permite operación Full-duplex (tiene registros independientes de transmisión y recepción)
- ▶ Permite operación síncrona (modo SPI) y asíncrona (modo UART)
- ▶ Soporta tramas seriales con 5, 6, 7, 8 y 9 bits de datos, y 1 o 2 bits de parada
- ▶ Generación de paridad impar o par y verificación de paridad por hardware

# Puerto serie USART – Trama

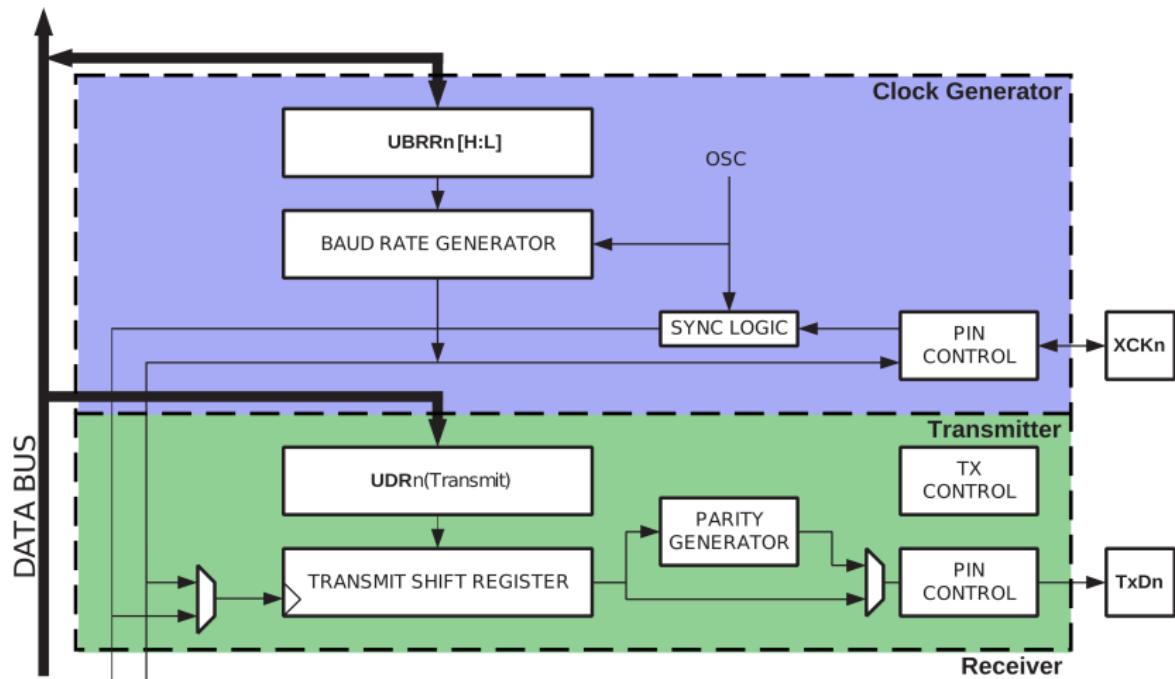


Donde:

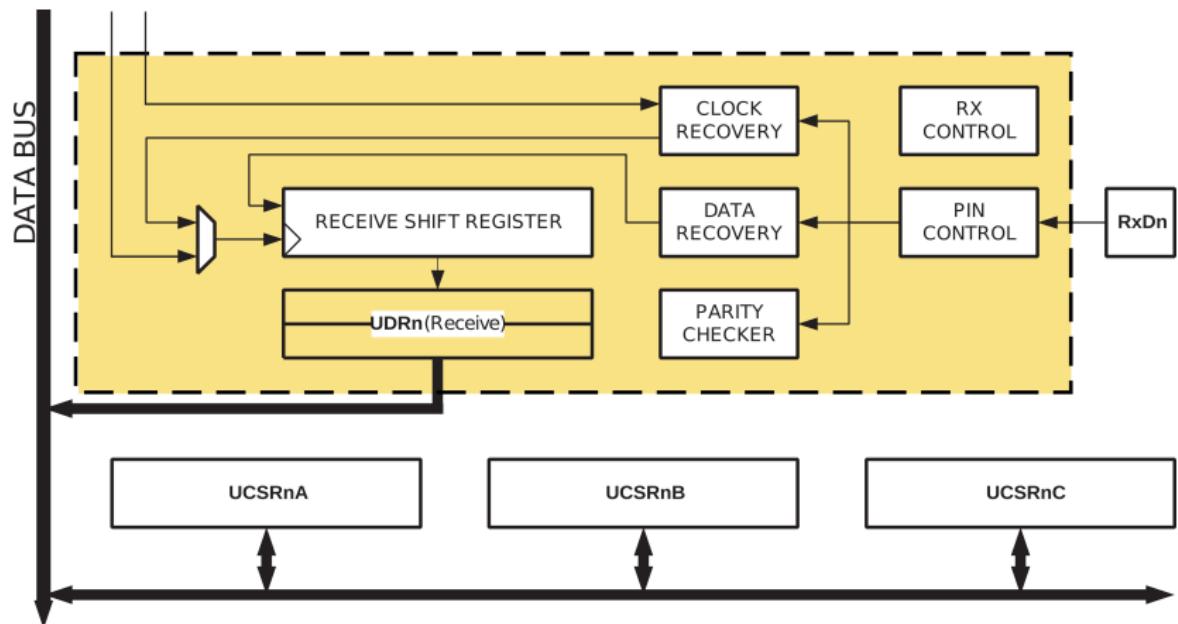
- ▶ **St:** Bit de inicio de trama (nivel bajo)
- ▶ **(n):** Bits de datos (0 a 8)
- ▶ **P:** Bit de paridad, que puede ser impar o par
- ▶ **Sp:** Bit de parada (nivel alto)
- ▶ **IDLE:** No hay transferencia de datos sobre las líneas de comunicación (RXD o TXD) (nivel alto)

Se envía primer el bit menos significativo (LSb: Least Significant bit)

# Puerto serie USART – Diagrama en bloques (1/2)



# Puerto serie USART – Diagrama en bloques (2/2)



# Puerto serie USART – Velocidad de comunic.

- ▶ El registro Baud Rate (UBRRn) en conjunto con un contador descendente (down-counter) funcionan como un prescaler o generador de baud-rate programable.
- ▶ El contador descendente (corriendo a  $f_{osc}$ ) se carga con el valor del registro UBRRn cada vez que alcanza el valor cero y genera un pulso de reloj.
- ▶ El reloj del generador de baud-rate tiene una frecuencia  $f_{osc}/(UBRRn + 1)$  la cual se divide luego por 2, 8 y 16 dependiendo del modo de la USART.

# Puerto serie USART – Velocidad de comunic.

- ▶ El registro Baud Rate (UBRRn) en conjunto con un contador descendente (down-counter) funcionan como un prescaler o generador de baud-rate programable.
- ▶ El contador descendente (corriendo a  $f_{osc}$ ) se carga con el valor del registro UBRRn cada vez que alcanza el valor cero y genera un pulso de reloj.
- ▶ El reloj del generador de baud-rate tiene una frecuencia  $f_{osc}/(UBRRn + 1)$  la cual se divide luego por 2, 8 y 16 dependiendo del modo de la USART.

En el modo asíncrono normal la tasa de transmisión de datos (BR) es

$$BR = \frac{f_{osc}}{16 \cdot (UBRRn + 1)},$$

por lo tanto

$$UBRRn = \frac{f_{osc}}{16 \cdot BR} - 1.$$

donde UBRRn es el valor del registro de velocidad.

# Puerto serie USART – Velocidad de comunic.

Baud rate	UBRRn	Error (%)	Baud rate	UBRRn	Error (%)
2400	416	-0.1	28.8K	34	-0.8
4800	207	0.2	38.4K	25	0.2
9600	103	0.2	57.6K	16	2.1
14.4K	68	0.6	76.8K	12	0.2
19.2K	51	0.2	115.2K	8	-3.5

Velocidades de comunicación y valor del registro UBRRn  
para  $f_{osc} = 16MHz$

# Puerto serie USART – Inicialización

- ▶ Para poder utilizar la USART en una comunicación es necesario su inicialización.
- ▶ En general, el proceso de inicialización consiste en configurar el baud-rate, el formato de la trama y habilitar el transmisor y/o el receptor.

# Puerto serie USART – Inicialización

- ▶ Para poder utilizar la USART en una comunicación es necesario su inicialización.
- ▶ En general, el proceso de inicialización consiste en configurar el baud-rate, el formato de la trama y habilitar el transmisor y/o el receptor.

Configuración:

1. Configuración del baud-rate: se realiza escribiendo los registros **UBRR0[H:L]**

# Puerto serie USART – Inicialización

- ▶ Para poder utilizar la USART en una comunicación es necesario su inicialización.
- ▶ En general, el proceso de inicialización consiste en configurar el baud-rate, el formato de la trama y habilitar el transmisor y/o el receptor.

Configuración:

1. Configuración del baud-rate: se realiza escribiendo los registros **UBRR0[H:L]**
2. El formato de la trama de comunicación se configura utilizando el registro **UCSR0C**

# Puerto serie USART – Inicialización

- ▶ Para poder utilizar la USART en una comunicación es necesario su inicialización.
- ▶ En general, el proceso de inicialización consiste en configurar el baud-rate, el formato de la trama y habilitar el transmisor y/o el receptor.

Configuración:

1. Configuración del baud-rate: se realiza escribiendo los registros **UBRR0[H:L]**
2. El formato de la trama de comunicación se configura utilizando el registro **UCSR0C**
3. La habilitación de la transmisión y recepción se realiza a través del registro **UCSR0B**

# Puerto serie USART – Registros

Bit	7	6	5	4	3	2	1	0	
0xC6	<b>RXB[7:0]</b>								<b>UDR0 (Read)</b>
	<b>TXB[7:0]</b>								<b>UDR0 (Write)</b>
Read/Write Initial value	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	
0xC0	<b>RXC0</b>	<b>TXC0</b>	<b>UDRE0</b>	<b>FE0</b>	<b>DOR0</b>	<b>UPE0</b>	<b>U2X0</b>	<b>MPCM0</b>	<b>UCSR0A</b>
Read/Write Initial value	R 0	R/W 0	R 1	R 0	R 0	R 0	R/W 0	R/W 0	
0xC1	<b>RXCIE0</b>	<b>TXCIE0</b>	<b>UDRIE0</b>	<b>RXEN0</b>	<b>TXEN0</b>	<b>UCSZ02</b>	<b>RXB80</b>	<b>TXB80</b>	<b>UCSR0B</b>
Read/Write Initial value	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R 0	R/W 0	
0xC2	<b>UMSEL01</b>	<b>UMSEL00</b>	<b>UPM01</b>	<b>UPM00</b>	<b>USBS0</b>	<b>UCSZ01</b>	<b>UCSZ00</b>	<b>UCPOL0</b>	<b>UCSR0C</b>
Read/Write Initial value	R/W 0	R/W 0	R/W 0	R/W 0	R/W 0	R/W 1	R 1	R/W 0	
0xC5	-	-	-	-	<b>UBRR0[11:8]</b>				
0xC4	<b>UBRR0[7:0]</b>								<b>UBRR0H</b>
Read/Write Initial value	R R/W 0	R R/W 0	R R/W 0	R R/W 0	R/W R/W 0	R/W R/W 0	R/W R/W 0	R/W R/W 0	<b>UBRR0L</b>

# USART0 – Registro de datos

Los registros búfer de transmisión y de recepción de datos de la USART comparten la misma dirección de entrada/salida como registro UDR0 (USART Data Register).

- ▶ Cuando se realiza una escritura en UDR0, los datos se almacenan en el registro búfer de transmisión (TXB).

# USART0 – Registro de datos

Los registros búfer de transmisión y de recepción de datos de la USART comparten la misma dirección de entrada/salida como registro UDR0 (USART Data Register).

- ▶ Cuando se realiza una escritura en UDR0, los datos se almacenan en el registro búfer de transmisión (TXB).
- ▶ Cuando se lee UDR0 se obtiene el valor del registro búfer de recepción de datos (RXB).

# USART0 – Registro de datos

Los registros búfer de transmisión y de recepción de datos de la USART comparten la misma dirección de entrada/salida como registro UDR0 (USART Data Register).

- ▶ Cuando se realiza una escritura en UDR0, los datos se almacenan en el registro búfer de transmisión (TXB).
- ▶ Cuando se lee UDR0 se obtiene el valor del registro búfer de recepción de datos (RXB).
- ▶ El búfer de transmisión puede escribirse cuando el bit UDRE0 del registro UCSROA este a 1.

# USART0 – Registro de datos

Los registros búfer de transmisión y de recepción de datos de la USART comparten la misma dirección de entrada/salida como registro UDR0 (USART Data Register).

- ▶ Cuando se realiza una escritura en UDR0, los datos se almacenan en el registro búfer de transmisión (TXB).
- ▶ Cuando se lee UDR0 se obtiene el valor del registro búfer de recepción de datos (RXB).
- ▶ El búfer de transmisión puede escribirse cuando el bit UDRE0 del registro UCSROA este a 1.
- ▶ El búfer de recepción consta de una FIFO de dos niveles, cuyo estado cambia cada vez que se acceda al búfer de recepción.

# USART0 – Registro de control y estado A

Funciones de los bits más utilizados del registro UCSR0A:

- ▶ Bit 7 – RXC0 (USART Receive Complete): bit que actúa como bandera (flag) para indicar que hay un dato no leído en el búfer de recepción y se borra cuando el búfer de recepción está vacío.

# USART0 – Registro de control y estado A

Funciones de los bits más utilizados del registro UCSR0A:

- ▶ Bit 7 – RXC0 (USART Receive Complete): bit que actúa como bandera (flag) para indicar que hay un dato no leído en el búfer de recepción y se borra cuando el búfer de recepción está vacío.
- ▶ Bit 6 – TXC0 (USART Transmit Complete): este bit de bandera se pone a 1 cuando la trama completa en el registro de desplazamiento de transmisión ha sido serializado y no hay datos nuevos en el búfer de transmisión.

# USART0 – Registro de control y estado A

Funciones de los bits más utilizados del registro UCSR0A:

- ▶ Bit 7 – RXC0 (USART Receive Complete): bit que actúa como bandera (flag) para indicar que hay un dato no leído en el búfer de recepción y se borra cuando el búfer de recepción está vacío.
- ▶ Bit 6 – TXC0 (USART Transmit Complete): este bit de bandera se pone a 1 cuando la trama completa en el registro de desplazamiento de transmisión ha sido serializado y no hay datos nuevos en el búfer de transmisión.
- ▶ Bit 5 – UDRE0 (USART Data Register Empty): este flag indica si el búfer de transmisión está listo para recibir un dato nuevo. Si está a 1 el búfer está vacío y listo para ser escrito.

# USART0 – Registro de control y estado A

Funciones de los bits más utilizados del registro UCSR0A:

- ▶ Bit 7 – RXC0 (USART Receive Complete): bit que actúa como bandera (flag) para indicar que hay un dato no leído en el búfer de recepción y se borra cuando el búfer de recepción está vacío.
- ▶ Bit 6 – TXC0 (USART Transmit Complete): este bit de bandera se pone a 1 cuando la trama completa en el registro de desplazamiento de transmisión ha sido serializado y no hay datos nuevos en el búfer de transmisión.
- ▶ Bit 5 – UDRE0 (USART Data Register Empty): este flag indica si el búfer de transmisión está listo para recibir un dato nuevo. Si está a 1 el búfer está vacío y listo para ser escrito.
- ▶ Bit 4 – FEO (Frame Error):

# USART0 – Registro de control y estado A

Funciones de los bits más utilizados del registro UCSR0A:

- ▶ Bit 7 – RXC0 (USART Receive Complete): bit que actúa como bandera (flag) para indicar que hay un dato no leído en el búfer de recepción y se borra cuando el búfer de recepción está vacío.
- ▶ Bit 6 – TXC0 (USART Transmit Complete): este bit de bandera se pone a 1 cuando la trama completa en el registro de desplazamiento de transmisión ha sido serializado y no hay datos nuevos en el búfer de transmisión.
- ▶ Bit 5 – UDRE0 (USART Data Register Empty): este flag indica si el búfer de transmisión está listo para recibir un dato nuevo. Si está a 1 el búfer está vacío y listo para ser escrito.
- ▶ Bit 4 – FEO (Frame Error):
- ▶ Bit 3 – DOR0 (Data OverRun):

# USART0 – Registro de control y estado A

Funciones de los bits más utilizados del registro UCSR0A:

- ▶ Bit 7 – RXC0 (USART Receive Complete): bit que actúa como bandera (flag) para indicar que hay un dato no leído en el búfer de recepción y se borra cuando el búfer de recepción está vacío.
- ▶ Bit 6 – TXC0 (USART Transmit Complete): este bit de bandera se pone a 1 cuando la trama completa en el registro de desplazamiento de transmisión ha sido serializado y no hay datos nuevos en el búfer de transmisión.
- ▶ Bit 5 – UDRE0 (USART Data Register Empty): este flag indica si el búfer de transmisión está listo para recibir un dato nuevo. Si está a 1 el búfer está vacío y listo para ser escrito.
- ▶ Bit 4 – FEO (Frame Error):
- ▶ Bit 3 – DOR0 (Data OverRun):
- ▶ Bit 2 – UPE0 (USART Parity Error):

# USART0 – Registro de control y estado B

Funciones de los bits más utilizados del registro UCSR0B:

- ▶ Bit 4 – RXENO (Receiver Enable): al escribir un 1 se habilita la recepción de datos y se configura el pin RxD0.

# USART0 – Registro de control y estado B

Funciones de los bits más utilizados del registro UCSR0B:

- ▶ Bit 4 – RXENO (Receiver Enable): al escribir un 1 se habilita la recepción de datos y se configura el pin RxD0.
- ▶ Bit 3 – TXENO (Transmitter Enable): al escribir un 1 se habilita la transmisión de datos y se configura el pin TxD0.

# USART0 – Registro de control y estado B

Funciones de los bits más utilizados del registro UCSR0B:

- ▶ Bit 4 – RXENO (Receiver Enable): al escribir un 1 se habilita la recepción de datos y se configura el pin RxD0.
- ▶ Bit 3 – TXENO (Transmitter Enable): al escribir un 1 se habilita la transmisión de datos y se configura el pin TxD0.
- ▶ Bit 2 – UCSZ02 (Character Size): este bit, en combinación con los bits UCSZ01:0, configuran la cantidad de bits de datos (Character SiZe) de la trama de transmisión y recepción.

# USART0 – Registro de control y estado C

Funciones de los bits más utilizados del registro UCSR0C:

- ▶ Bits 7:6 – UMSEL<sub>01:0</sub> (USART Mode Select): estos bits seleccionan el modo de operación de la USART0, donde para el modo asíncrono los valores deben ser 00.

# USART0 – Registro de control y estado C

Funciones de los bits más utilizados del registro UCSR0C:

- ▶ Bits 7:6 – UMSEL<sub>1:0</sub> (USART Mode Select): estos bits seleccionan el modo de operación de la USART0, donde para el modo asíncrono los valores deben ser 00.
- ▶ Bits 5:4 – UPM<sub>1:0</sub> (Parity Mode): fijan el tipo de paridad utilizada. Estos son: 00, paridad desactivada; 01, reservado; 10, paridad par; y 11, paridad impar.

# USART0 – Registro de control y estado C

Funciones de los bits más utilizados del registro UCSR0C:

- ▶ Bits 7:6 – UMSEL01:0 (USART Mode Select): estos bits seleccionan el modo de operación de la USART0, donde para el modo asíncrono los valores deben ser 00.
- ▶ Bits 5:4 – UPM01:0 (Parity Mode): fijan el tipo de paridad utilizada. Estos son: 00, paridad desactivada; 01, reservado; 10, paridad par; y 11, paridad impar.
- ▶ Bit 3 – USBS0 (Stop Bit Select): seleccionan la cantidad de bits de stop agregados en la transmisión (el receptor ignora esta configuración). Las opciones son 0: 1-bit de parada, o 1: 2-bits de parada.

# USART0 – Registro de control y estado C

Funciones de los bits más utilizados del registro UCSR0C:

- ▶ Bits 7:6 – UMSEL01:0 (USART Mode Select): estos bits seleccionan el modo de operación de la USART0, donde para el modo asíncrono los valores deben ser 00.
- ▶ Bits 5:4 – UPM01:0 (Parity Mode): fijan el tipo de paridad utilizada. Estos son: 00, paridad desactivada; 01, reservado; 10, paridad par; y 11, paridad impar.
- ▶ Bit 3 – USBS0 (Stop Bit Select): seleccionan la cantidad de bits de stop agregados en la transmisión (el receptor ignora esta configuración). Las opciones son 0: 1-bit de parada, o 1: 2-bits de parada.
- ▶ Bit 2:1 – UCSZ01:0 (Character Size): estos bits, en conjunto con el bit UCSZ02 en UCSR0B, configuran la cantidad de bits de datos (Character SiZe) utilizada en la trama de transmisión y recepción.

# USART0 – Tamaño de carácter (UCSR0B/C)

UCSZ02	UCSZ01	UCSZ00	Tamaño de carácter
0	0	0	5 bits
0	0	1	6 bits
0	1	0	7 bits
0	1	1	8 bits
1	0	0	Reservado
1	0	1	Reservado
1	1	0	Reservado
1	1	1	9 bits



# Programación con IDE Arduino



## Arduino IDE 1.8.13

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any Arduino board.

Refer to the [Getting Started](#) page for installation instructions.

### SOURCE CODE

Active development of the Arduino software is [hosted by GitHub](#). See the instructions for [building the code](#). Latest release source code archives are available [here](#). The archives are PGP-signed so they can be verified using [this](#) gpg key.

### DOWNLOAD OPTIONS

**Windows** Win 7 and newer

**Windows** ZIP file

**Windows app** Win 8.1 or 10



**Linux** 32 bits

**Linux** 64 bits

**Linux** ARM 32 bits

**Linux** ARM 64 bits

**Mac OS X** 10.10 or newer

[Release Notes](#) [Checksums \(sha512\)](#)

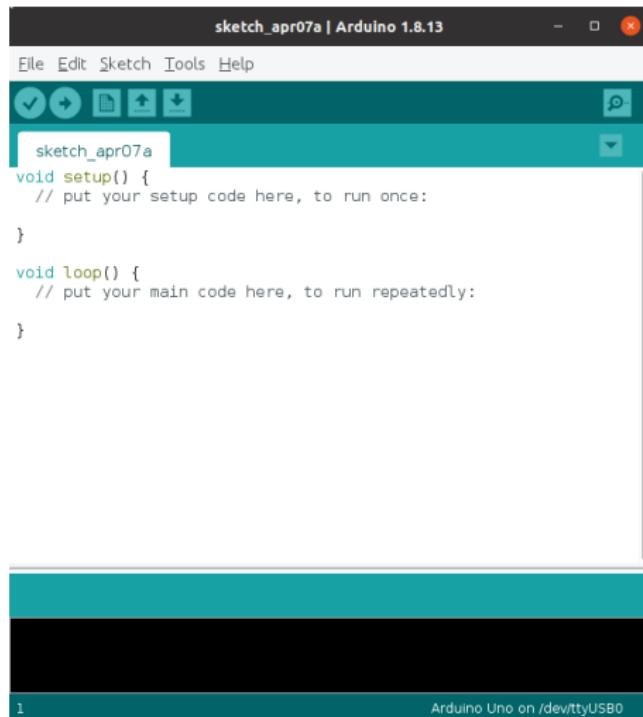
- ▶ Descargar y descomprimir el archivo (`arduino-1.8.13-linux64.tar.xz`)
- ▶ Ejecutar el script de instalación (`./install.sh`)

# Programación con IDE Arduino

IDE: Conjunto de herramientas de Sw que permite a los programadores desarrollar (básicamente escribir y probar) sus propios programas.

# Programación con IDE Arduino

IDE: Conjunto de herramientas de Sw que permite a los programadores desarrollar (básicamente escribir y probar) sus propios programas.



The screenshot shows the Arduino IDE interface. The title bar reads "sketch\_apr07a | Arduino 1.8.13". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for save, upload, and search. A dropdown menu shows "sketch\_apr07a". The main code editor contains the following code:

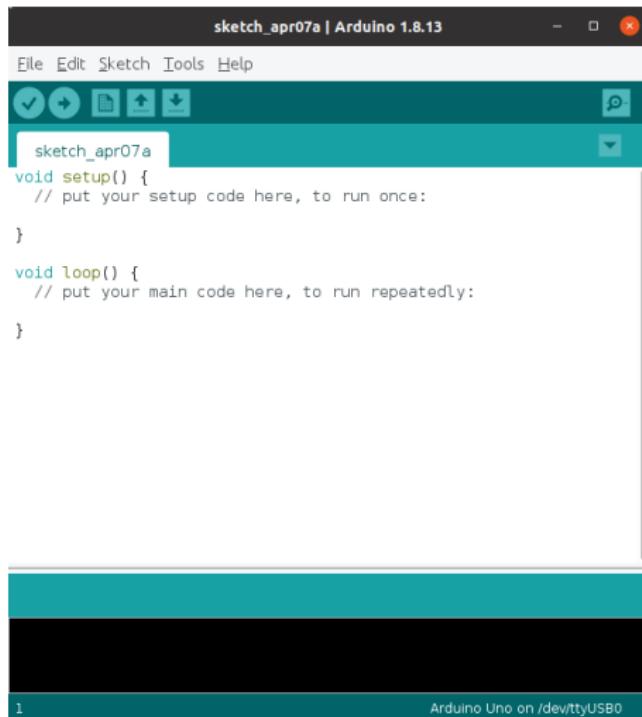
```
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

At the bottom of the screen, there is a status bar with the text "1" and "Arduino Uno on /dev/ttyUSB0".

# Programación con IDE Arduino

IDE: Conjunto de herramientas de Sw que permite a los programadores desarrollar (básicamente escribir y probar) sus propios programas.

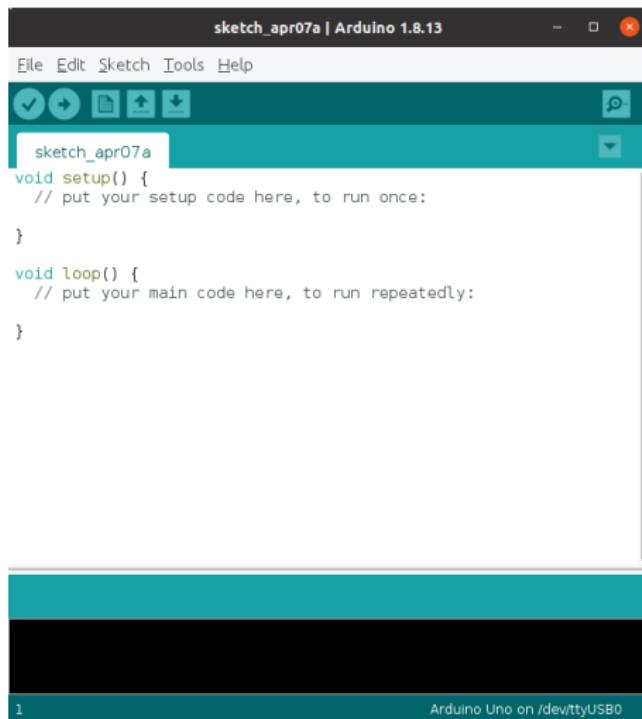


Se divide en 4 áreas:

1. Barra de menús
2. Barra de botones
3. Editor
4. Ventana de mensajes

# Programación con IDE Arduino

IDE: Conjunto de herramientas de Sw que permite a los programadores desarrollar (básicamente escribir y probar) sus propios programas.



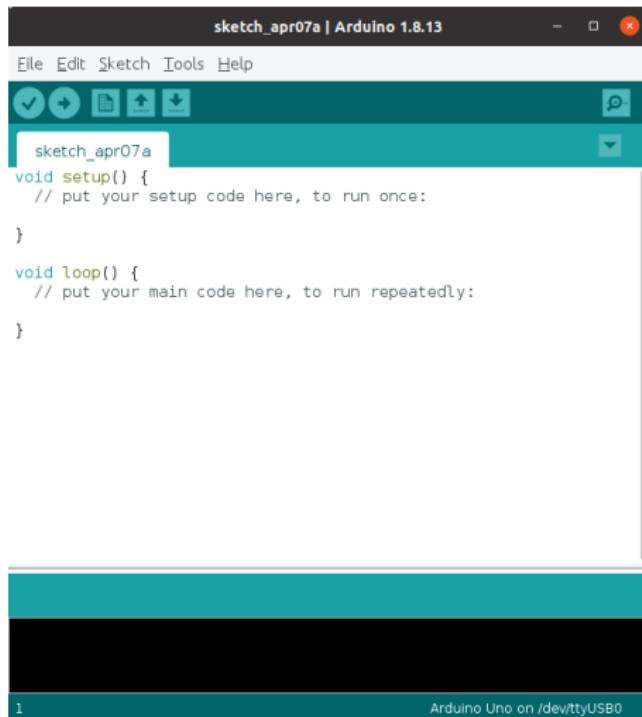
Se divide en 4 áreas:

1. Barra de menús
2. Barra de botones
3. Editor
4. Ventana de mensajes

sketch Arduino – 2 bloques:

# Programación con IDE Arduino

IDE: Conjunto de herramientas de Sw que permite a los programadores desarrollar (básicamente escribir y probar) sus propios programas.



Se divide en 4 áreas:

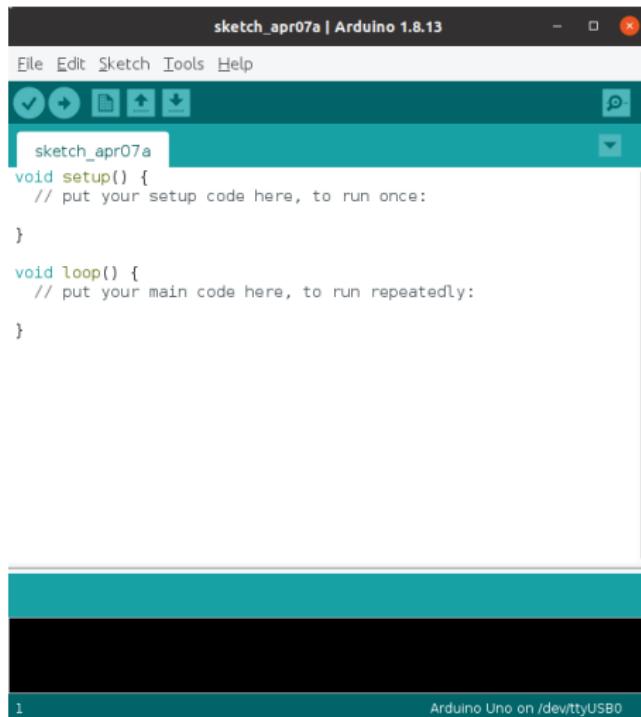
1. Barra de menús
2. Barra de botones
3. Editor
4. Ventana de mensajes

sketch Arduino – 2 bloques:

- ▶ **setup()**: se ejecuta una única vez cuando se enciende o resetea la placa

# Programación con IDE Arduino

IDE: Conjunto de herramientas de Sw que permite a los programadores desarrollar (básicamente escribir y probar) sus propios programas.



Se divide en 4 áreas:

1. Barra de menús
2. Barra de botones
3. Editor
4. Ventana de mensajes

sketch Arduino – 2 bloques:

- ▶ **setup()**: se ejecuta una única vez cuando se enciende o resetea la placa
- ▶ **loop()**: se ejecuta de forma constante (bucle)

# Sketch ejemplo: blink

File->Examples->01.Basics->Blink

---

```
1 // the setup function runs once when you press reset or
2 // power the board
3 void setup() {
4     // initialize digital pin LED_BUILTIN as an output.
5     pinMode(LED_BUILTIN, OUTPUT);
6 }
7
8 // the loop function runs over and over again forever
9 void loop() {
10    digitalWrite(LED_BUILTIN, HIGH);      // turn the LED on
11    delay(1000);                      // wait for a second
12    digitalWrite(LED_BUILTIN, LOW);     // turn the LED off
13    delay(1000);                      // wait for a second
14 }
```

---

# Sketch ejemplo: puerto serie

---

```
1 #define MENSAJE "Hola mundo"
2
3 // La función 'setup' se ejecuta una única vez al
4 // presionar reset o encender la placa
5 void setup() {
6     // Inicializa el puerto serie (UART) a 9600 bps.
7     Serial.begin(9600);
8 }
9
10 // La función 'loop' corre indefinidamente una y otra vez
11 void loop() {
12     // Imprime (envía) la cadena MENSAJE por puerto serie.
13     Serial.println(MENSAJE);
14     delay(500);
15 }
```

---



# Programación con Toolchain GCC para AVR

El Toolchain de GCC para arquitectura AVR está compuesto por:

`gcc`: el compilador de C y C++

# Programación con Toolchain GCC para AVR

El Toolchain de GCC para arquitectura AVR está compuesto por:

`gcc`: el compilador de C y C++

`binutils`: colección de herramientas para la manipulación de archivos binarios

# Programación con Toolchain GCC para AVR

El Toolchain de GCC para arquitectura AVR está compuesto por:

`gcc`: el compilador de C y C++

`binutils`: colección de herramientas para la manipulación de archivos binarios

`libc-avr`: subconjunto de la biblioteca estándar de C con funciones específicas para arquitectura AVR

# Programación con Toolchain GCC para AVR

El Toolchain de GCC para arquitectura AVR está compuesto por:

`gcc`: el compilador de C y C++

`binutils`: colección de herramientas para la manipulación de archivos binarios

`libc-avr`: subconjunto de la biblioteca estándar de C con funciones específicas para arquitectura AVR

`avrdude`: programa para escribir y leer la memoria Flash del  $\mu$ C

# Programación con Toolchain GCC para AVR

El Toolchain de GCC para arquitectura AVR está compuesto por:

`gcc`: el compilador de C y C++

`binutils`: colección de herramientas para la manipulación de archivos binarios

`libc-avr`: subconjunto de la biblioteca estándar de C con funciones específicas para arquitectura AVR

`avrdude`: programa para escribir y leer la memoria Flash del  $\mu$ C

La instalación del Toolchain en Ubuntu se realiza instalando los siguientes paquetes: `gcc-avr`, `binutils-avr`, `avr-libc` y `avrdude`.

# Programación con Toolchain GCC para AVR

El Toolchain de GCC para arquitectura AVR está compuesto por:

`gcc`: el compilador de C y C++

`binutils`: colección de herramientas para la manipulación de archivos binarios

`libc-avr`: subconjunto de la biblioteca estándar de C con funciones específicas para arquitectura AVR

`avrdude`: programa para escribir y leer la memoria Flash del  $\mu$ C

La instalación del Toolchain en Ubuntu se realiza instalando los siguientes paquetes: `gcc-avr`, `binutils-avr`, `avr-libc` y `avrdude`.

- ▶ El compilador GCC para AVR (`avr-gcc`) es en realidad es un compilador cruzado (cross-compiler).
- ▶ Un compilador cruzado es aquel que genera código máquina para una arquitectura diferente a aquella en la cual se ejecuta el compilador.

# Programación con Toolchain GCC para AVR

## Archivo 'blink.c'

---

```
1 #include <avr/io.h>
2 #include <util/delay.h>
3
4 #define BLINK_DELAY_MS 1000
5
6 int main (void)
7 {
8     /* Inicializa el pin digital como salida */
9     DDRB |= _BV(DDB5);
10
11    while(1)
12    {
13        PORTB |= _BV(PORTB5); /* Enciende LED */
14        _delay_ms(BLINK_DELAY_MS);
15
16        PORTB &= ~_BV(PORTB5); /* Apaga LED */
17        _delay_ms(BLINK_DELAY_MS);
18    }
19    return 0;
20 }
```

---

# Programación con Toolchain GCC para AVR

Construcción del programa y grabación en la memoria del  $\mu$ C ATmega328 de la placa Arduino UNO:

# Programación con Toolchain GCC para AVR

Construcción del programa y grabación en la memoria del  $\mu$ C ATmega328 de la placa Arduino UNO:

1. Construcción del código fuente (.c) hasta la etapa de ensamblado, de la cual se obtiene un archivo objeto (.o):

```
avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -c -o  
blink.o blink.c
```

# Programación con Toolchain GCC para AVR

Construcción del programa y grabación en la memoria del  $\mu$ C ATmega328 de la placa Arduino UNO:

1. Construcción del código fuente (.c) hasta la etapa de ensamblado, de la cual se obtiene un archivo objeto (.o):

```
avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -c -o  
blink.o blink.c
```

2. Enlazado con bibliotecas, del cual se obtiene un archivo binario (.elf):  
`avr-gcc -mmcu=atmega328p blink.o -o blink.elf`

# Programación con Toolchain GCC para AVR

Construcción del programa y grabación en la memoria del  $\mu$ C ATmega328 de la placa Arduino UNO:

1. Construcción del código fuente (.c) hasta la etapa de ensamblado, de la cual se obtiene un archivo objeto (.o):

```
avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -c -o  
blink.o blink.c
```

2. Enlazado con bibliotecas, del cual se obtiene un archivo binario (.elf):  
`avr-gcc -mmcu=atmega328p blink.o -o blink.elf`

3. Conversión del código binario (.elf) a formato objeto hexadecimal Intel (.hex)

```
avr-objcopy -O ihex -R .eeprom blink.elf blink.hex
```

# Programación con Toolchain GCC para AVR

Construcción del programa y grabación en la memoria del  $\mu$ C ATmega328 de la placa Arduino UNO:

1. Construcción del código fuente (.c) hasta la etapa de ensamblado, de la cual se obtiene un archivo objeto (.o):

```
avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -c -o  
blink.o blink.c
```

2. Enlazado con bibliotecas, del cual se obtiene un archivo binario (.elf):  
`avr-gcc -mmcu=atmega328p blink.o -o blink.elf`

3. Conversión del código binario (.elf) a formato objeto hexadecimal Intel (.hex)  
`avr-objcopy -O ihex -R .eeprom blink.elf blink.hex`

4. Grabación en la memoria de programa del  $\mu$ C

```
avrdude -F -V -c arduino -p ATMEGA328P -P /dev/ttyACM0  
-b 115200 -U flash:w:blink.hex
```



# Programación de los puertos digitales – salidas (1)

- ▶ El código fuente `blink.c` es un ejemplo de manejo de un puerto digital de entrada/salida.
- ▶ Se utiliza el bit 5 del puerto B (`PB5`) configurado como salida para encender y apagar un LED conectado al pin correspondiente.
- ▶ La placa Arduino UNO incluye un LED conectado a dicho pin

# Programación de los puertos digitales – salidas (1)

- ▶ El código fuente `blink.c` es un ejemplo de manejo de un puerto digital de entrada/salida.
- ▶ Se utiliza el bit 5 del puerto B (PB5) configurado como salida para encender y apagar un LED conectado al pin correspondiente.
- ▶ La placa Arduino UNO incluye un LED conectado a dicho pin

Configuración del pin PB5 como salida:

```
DDRB |= _BV(DDB5); /* Equiv. a DDRB = DDRB | _BV(DDB5); */
```

# Programación de los puertos digitales – salidas (1)

- ▶ El código fuente `blink.c` es un ejemplo de manejo de un puerto digital de entrada/salida.
- ▶ Se utiliza el bit 5 del puerto B (PB5) configurado como salida para encender y apagar un LED conectado al pin correspondiente.
- ▶ La placa Arduino UNO incluye un LED conectado a dicho pin

Configuración del pin PB5 como salida:

```
DDRB |= _BV(DDB5); /* Equiv. a DDRB = DDRB | _BV(DDB5); */
```

que usa la macro `_BV()` definida como:

```
#define _BV(bit) (1 << (bit))
```

y la constante simbólica DDB5 que vale 5.

# Programación de los puertos digitales – salidas (1)

- ▶ El código fuente `blink.c` es un ejemplo de manejo de un puerto digital de entrada/salida.
- ▶ Se utiliza el bit 5 del puerto B (PB5) configurado como salida para encender y apagar un LED conectado al pin correspondiente.
- ▶ La placa Arduino UNO incluye un LED conectado a dicho pin

Configuración del pin PB5 como salida:

```
DDRB |= _BV(DDB5); /* Equiv. a DDRB = DDRB | _BV(DDB5); */
```

que usa la macro `_BV()` definida como:

```
#define _BV(bit) (1 << (bit))
```

y la constante simbólica DDB5 que vale 5. Por lo que la operación queda:

```
DDRB = DDRB | (1 << (5)); /* DDRB = DDRB | 0x40 */
```

# Programación de los puertos digitales – salidas (1)

- ▶ El código fuente `blink.c` es un ejemplo de manejo de un puerto digital de entrada/salida.
- ▶ Se utiliza el bit 5 del puerto B (PB5) configurado como salida para encender y apagar un LED conectado al pin correspondiente.
- ▶ La placa Arduino UNO incluye un LED conectado a dicho pin

Configuración del pin PB5 como salida:

```
DDRB |= _BV(DDB5); /* Equiv. a DDRB = DDRB | _BV(DDB5); */
```

que usa la macro `_BV()` definida como:

```
#define _BV(bit) (1 << (bit))
```

y la constante simbólica DDB5 que vale 5. Por lo que la operación queda:

```
DDRB = DDRB | (1 << (5)); /* DDRB = DDRB | 0x40 */
```

que pone a 1 el bit 5 del registro DDRB.

# Programación de los puertos digitales – salidas (1)

Luego en el bucle principal se enciende y apaga el LED cada 1000ms, poniendo a 1 y a 0 respectivamente el bit 5 del registro de datos del puerto B (**PORTB5**).

# Programación de los puertos digitales – salidas (1)

Luego en el bucle principal se enciende y apaga el LED cada 1000ms, poniendo a 1 y a 0 respectivamente el bit 5 del registro de datos del puerto B (**PORTB5**).

El bit se pone a 1 con la línea:

```
PORTB |= _BV(PORTB5);
```

# Programación de los puertos digitales – salidas (1)

Luego en el bucle principal se enciende y apaga el LED cada 1000ms, poniendo a 1 y a 0 respectivamente el bit 5 del registro de datos del puerto B (**PORTB5**).

El bit se pone a 1 con la línea:

```
PORTB |= _BV(PORTB5);
```

y se pone a 0 con la línea:

```
PORTB = PORTB & ~(1 << (5)); /* PORTB = PORTB & 0xBF; */
```

la cual utiliza el operador AND y NOT a nivel de bit. La constante simbólica **PORTB5** vale 5.

# Programación de los puertos digitales – salidas (1)

Luego en el bucle principal se enciende y apaga el LED cada 1000ms, poniendo a 1 y a 0 respectivamente el bit 5 del registro de datos del puerto B (**PORTB5**).

El bit se pone a 1 con la línea:

```
PORTB |= _BV(PORTB5);
```

y se pone a 0 con la línea:

```
PORTB = PORTB & ~(1 << (5)); /* PORTB = PORTB & 0xBF; */
```

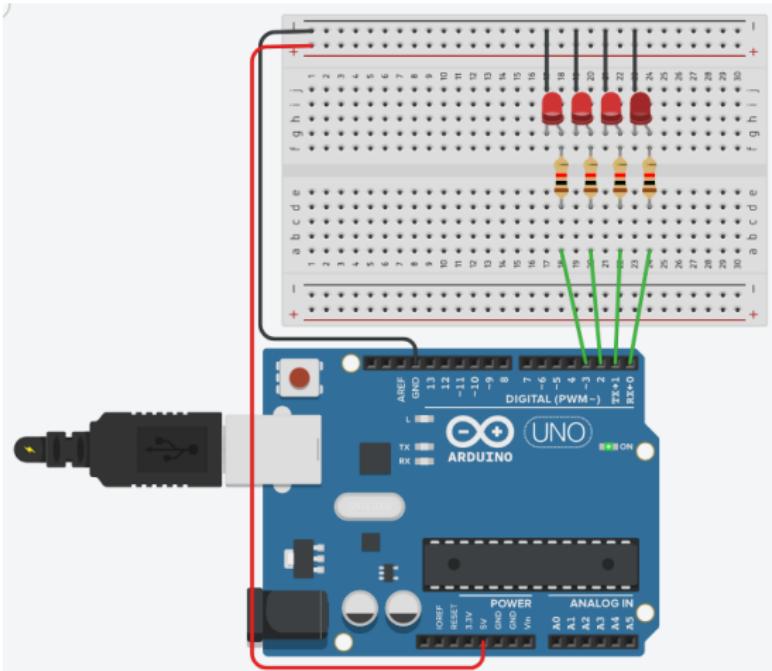
la cual utiliza el operador AND y NOT a nivel de bit. La constante simbólica **PORTB5** vale 5.

Otra forma de hacer lo mismo es usando el operador XOR, reemplazando el bucle principal por:

```
while(1)
{
    PORTB ^= _BV(PORTB5); /* Toggle LED */
    _delay_ms(BLINK_DELAY_MS);
}
```

# Programación de los puertos digitales – salidas (2)

Circuito contador de 4 bits



Utiliza los 4 bits menos significativos del puerto D (PB0 a PB3).

# Programación de los puertos digitales – salidas (2)

## Archivo 'led\_hex\_counter.c'

---

```
1 #include <avr/io.h>
2 #include <util/delay.h>
3
4 #define LED_DDR DDRD
5 #define LED_PORT PORTD
6 #define LED_MASK 0x0F
7
8 #define DELAY_MS 500
9
10 int main (void)
11 {
12     unsigned char hex = 0;
13
14     /* Inicializa salidas digitales */
15     LED_DDR |= LED_MASK;
16
17     while(1)
18     {
19         LED_PORT &= ~(LED_MASK); // Apaga los LEDs
20         LED_PORT |= hex; // Muestra valor
21 }
```

---

# Programación de los puertos digitales – salidas (2)

## Archivo 'led\_hex\_counter.c' (cont.)

---

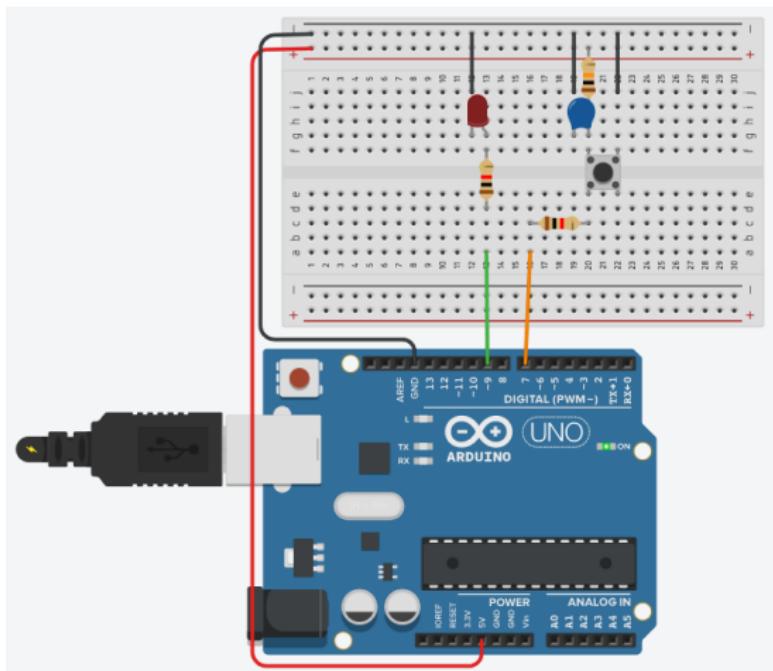
```
22     if(++hex > 16) // Controla rango máximo
23         hex = 0;
24
25     _delay_ms(DELAY_MS);
26 }
27 return 0;
28 }
```

---

Los LEDs muestran una cuenta binario de 4 bits, o sea del rango de valores que va desde  $0d = 0000b = 0x00$  hasta  $15d = 1111b = 0x0F$ .

# Programación de los puertos digitales – entradas

## Circuito con pulsador y LED



El pulsador se conecta al pin 7 que es el bit 7 del puerto D (PD7) el cual debe configurarse como entrada y el LED se conecta al pin 9 que es el bit 1 del puerto B (PB1) que debe configurarse como salida.

# Programación de los puertos digitales – entradas

## Archivo 'led\_sw.c'

---

```
1 #include <avr/io.h>
2 #include <util/delay.h>
3
4 /*
5  * LED: pin 9 --> PB1 (Salida)
6  * Sw: pin 7 --> PD7 (Entrada)
7  */
8 #define SW_MASK _BV(PIND7)
9 #define SW_PRES 0
10
11 int main(void)
12 {
13     uint8_t sw;
14
15     /* Configuración de entrada/salida digitales */
16     DDRB |= _BV(DDB1); /* LED como salida */
17     DDRD &= ~_BV(DDD7); /* Sw como entrada */
18     PORTB &= ~_BV(PORTB1); /* Apaga el LED */
19
```

---

# Programación de los puertos digitales – entradas

## Archivo 'led\_sw.c' (cont.)

```
20  while(1)
21  {
22      /* Lee el valor del pulsador */
23      sw = (PIND & SW_MASK) >> PIND7;
24
25      if(sw == SW_PRES)
26      {
27          PORTB |= _BV(PORTB1); /* Enciende LED */
28          _delay_ms(1000);
29          PORTB &= ~_BV(PORTB1); /* Apaga LED */
30      }
31      _delay_ms(1);
32  }
33  return 0;
34 }
```

El programa lee el valor de la entrada digital conectada al pulsador y, en caso de estar presionado, encender el LED durante 1 seg. y luego apagarlo.

# Programación de los puertos digitales – entradas

La constante simbólica `SW_MASK` se utiliza como máscara para la lectura de la entrada digital (línea 23).

# Programación de los puertos digitales – entradas

La constante simbólica `SW_MASK` se utiliza como máscara para la lectura de la entrada digital (línea 23).

La línea que realiza la lectura de la entrada es:

```
sw = (PIND & SW_MASK) >> PIND7;
```

donde se utiliza el valor del registro `PIND` para la lectura del Puerto D, la que se enmascara con `SW_MASK`, o sea que se ponen a cero todos los bits menos aquel cuyo valor interesa (a esto se le denomina máscara).

# Programación de los puertos digitales – entradas

La constante simbólica `SW_MASK` se utiliza como máscara para la lectura de la entrada digital (línea 23).

La línea que realiza la lectura de la entrada es:

```
sw = (PIND & SW_MASK) >> PIND7;
```

donde se utiliza el valor del registro `PIND` para la lectura del Puerto D, la que se enmascara con `SW_MASK`, o sea que se ponen a cero todos los bits menos aquel cuyo valor interesa (a esto se le denomina máscara).

Este valor luego se desplaza hacia la derecha para ocupar el bit menos significativo de la variable de 8 bits de nombre `sw`, que se utiliza luego en la estructura de selección `if` para determinar si hay que encender el LED.

# Programación de los puertos digitales – entradas

La constante simbólica **SW\_MASK** se utiliza como máscara para la lectura de la entrada digital (línea 23).

La línea que realiza la lectura de la entrada es:

```
sw = (PIND & SW_MASK) >> PIND7;
```

donde se utiliza el valor del registro **PIND** para la lectura del Puerto D, la que se enmascara con **SW\_MASK**, o sea que se ponen a cero todos los bits menos aquel cuyo valor interesa (a esto se le denomina máscara).

Este valor luego se desplaza hacia la derecha para ocupar el bit menos significativo de la variable de 8 bits de nombre **sw**, que se utiliza luego en la estructura de selección **if** para determinar si hay que encender el LED.

La constante simbólica **SW\_PRES** fija el valor que tendrá la entrada digital si el pulsador se encuentra presionado, lo cual dependerá de cómo este armado el circuito.



# Programación de la UART – Configuración

La configuración de la USART consiste en:

- ▶ seleccionar el modo de funcionamiento asíncrono (UART),
- ▶ definir el tipo de trama,
- ▶ velocidad de comunicación (bps) y
- ▶ habilitar el transmisor y/o el receptor.

# Programación de la UART – Configuración

La configuración de la USART consiste en:

- ▶ seleccionar el modo de funcionamiento asíncrono (UART),
  - ▶ definir el tipo de trama,
  - ▶ velocidad de comunicación (bps) y
  - ▶ habilitar el transmisor y/o el receptor.
- 
- ▶ El tipo de trama queda determinada por la cantidad de bits de datos, y de parada y si se habilita o no la detección de errores mediante paridad y el tipo de paridad (par o impar).

# Programación de la UART – Configuración

La configuración de la USART consiste en:

- ▶ seleccionar el modo de funcionamiento asíncrono (UART),
  - ▶ definir el tipo de trama,
  - ▶ velocidad de comunicación (bps) y
  - ▶ habilitar el transmisor y/o el receptor.
- 
- ▶ El tipo de trama queda determinada por la cantidad de bits de datos, y de parada y si se habilita o no la detección de errores mediante paridad y el tipo de paridad (par o impar).
  - ▶ La configuración de la USART0 del  $\mu$ C ATmega328 se utilizan mediante los registros de control y estado B y C, UCSR0B y UCSR0C.

# Programación de la UART – Configuración

La configuración de la USART consiste en:

- ▶ seleccionar el modo de funcionamiento asíncrono (UART),
  - ▶ definir el tipo de trama,
  - ▶ velocidad de comunicación (bps) y
  - ▶ habilitar el transmisor y/o el receptor.
- 
- ▶ El tipo de trama queda determinada por la cantidad de bits de datos, y de parada y si se habilita o no la detección de errores mediante paridad y el tipo de paridad (par o impar).
  - ▶ La configuración de la USART0 del  $\mu$ C ATmega328 se utilizan mediante los registros de control y estado B y C, UCSR0B y UCSR0C.
  - ▶ El formato de trama suele indicarse por una combinación de dos números y una letra, p.e.: 8N1, 5E2, 7O1, etc.

# Programación de la UART – Configuración

Definición de constantes simbólicas (antes de la función `main()`):

```
#define BAUD_RATE_4800 207
#define BAUD_RATE_9600 103
#define BAUD_RATE_38400 25
#define BAUD_RATE_57600 16
#define BAUD_RATE_115200 8
```

En la función `main`:

```
uint16_t ubrr = BAUD_RATE_115200;

/* Configuración el baud-rate */
UBRROH = (ubrr >> 8) & 0xFF; /* Byte más significativo */
UBRROL = ubrr & 0xFF; /* Byte menos significativo */

UCSROC = _BV(UCSZ00) | _BV(UCSZ01); /* UART con trama 8N1 */
```

La última línea equivale a escribir el valor  $6d = 00000110b = 0x06$  en el registro.

# Programación de la UART – Transmisión

## Archivo 'hola\_mundo.c'

---

```
1 #include <avr/io.h>
2 #include <util/delay.h>
3 #include <stdint.h>
4
5 #define BAUD_RATE_115200 8
6
7 int main()
8 {
9     int i;
10    uint16_t ubrr = BAUD_RATE_115200;
11    uint8_t dato[] = "Hola mundo\n";
12
13    /* Configuración el baud-rate */
14    UBRROH = (ubrr >> 8) & 0xFF; /* Byte más significativo */
15    UBRROL = ubrr & 0xFF; /* Byte menos significativo */
16
17    UCSROC = _BV(UCSZ00) | _BV(UCSZ01); /* UART con trama 8N1 */
18    UCSROB |= _BV(TXENO); /* Habilita el transmisor */
19
```

---

# Programación de la UART – Transmisión

Archivo 'hola\_mundo.c' (cont.)

---

```
20 while(1)
21 {
22     for(i = 0; dato[i] != '\0'; i++)
23     {
24         /* Espera a que el búfer de transmisión esté vacío */
25         while( !(UCSROA & _BV(UDRE0)) ) ;
26         UDR0 = dato[i]; /* Escribe el búfer de transmisión */
27     }
28     _delay_ms(500);
29 }
30 return 0;
31 }
```

---

# Programación de la UART – Transmisión

Archivo 'hola\_mundo.c' (cont.)

---

```
20 while(1)
21 {
22     for(i = 0; dato[i] != '\0'; i++)
23     {
24         /* Espera a que el búfer de transmisión esté vacío */
25         while( !(UCSROA & _BV(UDRE0)) ) ;
26         UDR0 = dato[i]; /* Escribe el búfer de transmisión */
27     }
28     _delay_ms(500);
29 }
30 return 0;
31 }
```

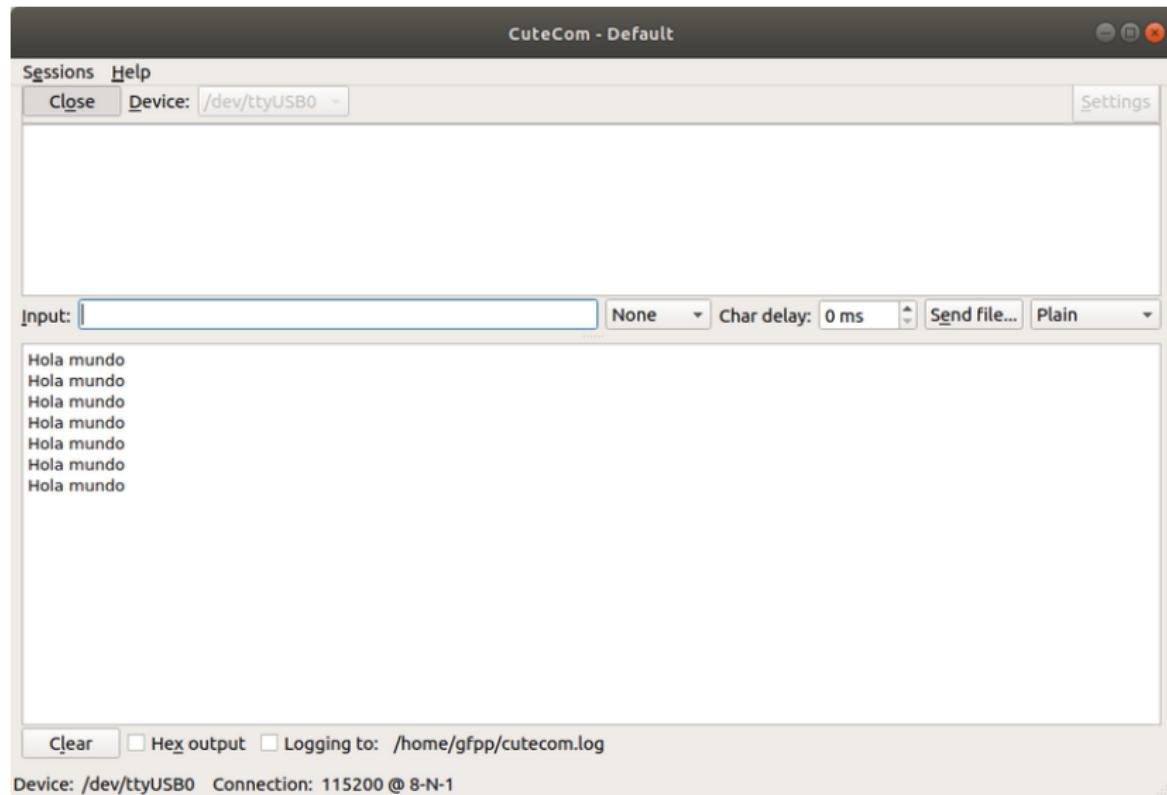
---

Se escribe el registro de transmisión de datos, UDR0, habiendo verificado que el búfer de transmisión está vacío. La línea:

```
while( !(UCSROA & _BV(UDRE0)) ) ;
```

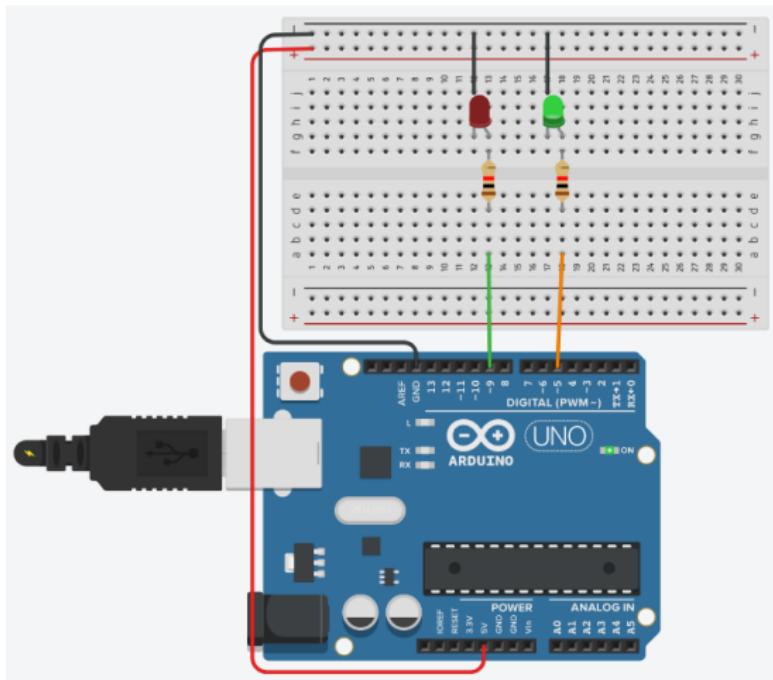
espera a que el búfer de transmisión este vacío.

# Programación de la UART – Transmisión



# Programación de la UART – Recepción

## Circuito para la recepción de la UART



- ▶ El LED rojo, en PB1, cambia de estado al recibir el carácter 'r'
- ▶ El LED verde, en PD5, cambia de estado al recibir el carácter 'v'

# Programación de la UART – Recepción

## Archivo 'uart\_read.c'

---

```
1 #include <avr/io.h>
2 #include <stdint.h>
3
4 #define BAUD_RATE_115200 8
5
6 int main()
7 {
8     uint16_t ubrr = BAUD_RATE_115200;
9
10    DDRB |= _BV(DDB1); /* Led rojo, salida */
11    DDRD |= _BV(DDD5); /* Led verde, salida */
12
13    /* Configuración el baud-rate */
14    UBRROH = (ubrr >> 8) & 0xFF; /* Byte más significativo */
15    UBRROL = ubrr & 0xFF; /* Byte menos significativo */
16
17    UCSROC = _BV(UCSZ00) | _BV(UCSZ01); /* UART con trama 8N1 */
18    UCSROB |= _BV(RXENO); /* Habilita el receptor */
19
```

---

# Programación de la UART – Recepción

## Archivo 'uart\_read.c' (cont.)

---

```
20 while(1)
21 {
22     /* Espera a recibir un dato */
23     while( !(UCSROA & _BV(RXC0)) ) ;
24
25     switch(UDR0) {
26         case 'r':
27         case 'R':
28             PORTB ^= _BV(PORTB1); /* Cambia estado de LED rojo */
29             break;
30
31         case 'v':
32         case 'V':
33             PORTD ^= _BV(PORTD5); /* Cambia estado de LED verde */
34             break;
35     }
36 }
37 return 0;
38 }
```

---

# Programación de la UART – Recepción

En el bucle principal la línea:

```
while( !(UCSROA & _BV(RXCO)) ) ;
```

espera hasta recibir un carácter.

# Programación de la UART – Recepción

En el bucle principal la línea:

```
while( !(UCSROA & _BV(RXCO)) ) ;
```

espera hasta recibir un carácter.

Luego se utiliza la estructura de selección **switch-case** para actuar sobre el LED adecuado dependiendo del carácter recibido.

# Programación de la UART – Trans. y recep.

## Archivo 'to\_upper.c'

---

```
1 #include <avr/io.h>
2 #include <stdint.h>
3 #define BAUD_RATE_57600 16
4
5 int main()
6 {
7     uint16_t ubrr = BAUD_RATE_57600;
8     uint8_t car;
9
10    /* Configuración el baud-rate */
11    UBRROH = (ubrr >> 8) & 0xFF; /* Byte más significativo */
12    UBRROL = ubrr & 0xFF; /* Byte menos significativo */
13
14    UCSROC = _BV(UCSZ00) | _BV(UCSZ01); /* UART con trama 8N1 */
15    UCSROB |= _BV(TXENO); /* Habilita el transmisor */
16    UCSROB |= _BV(RXENO); /* Habilita el receptor */
17
```

---

# Programación de la UART – Trans. y recep.

## Archivo 'to\_upper.c' (cont.)

---

```
18  while(1)
19  {
20      /* Espera a recibir un dato */
21      while( !(UCSROA & _BV(RXCO)) ) ;
22      car = UDR0; /* Lee carácter */
23
24      if( (car >= 'a') && (car <= 'z') )
25          car -= 32;
26
27      /* Espera a que el búfer de transmisión esté vacío */
28      while( !(UCSROA & _BV(UDRE0)) ) ;
29      UDR0 = car; /* Escribe el búfer de transmisión */
30  }
31  return 0;
32 }
```

---

# Programación de la UART – Trans. y recep.

## Archivo 'to\_upper.c' (cont.)

---

```
18  while(1)
19  {
20      /* Espera a recibir un dato */
21      while( !(UCSROA & _BV(RXCO)) ) ;
22      car = UDR0; /* Lee carácter */
23
24      if( (car >= 'a') && (car <= 'z') )
25          car -= 32;
26
27      /* Espera a que el búfer de transmisión esté vacío */
28      while( !(UCSROA & _BV(UDRE0)) ) ;
29      UDR0 = car; /* Escribe el búfer de transmisión */
30  }
31  return 0;
32 }
```

---

O bien utilizando funciones de la biblioteca estándar de C, como:

```
if( isalpha(car) )
    car = toupper(car);
```

para lo cual es necesario incluir el archivo de cabecera `ctype.h`.

