UNIVERSIDAD DE GRANADA

ICAR

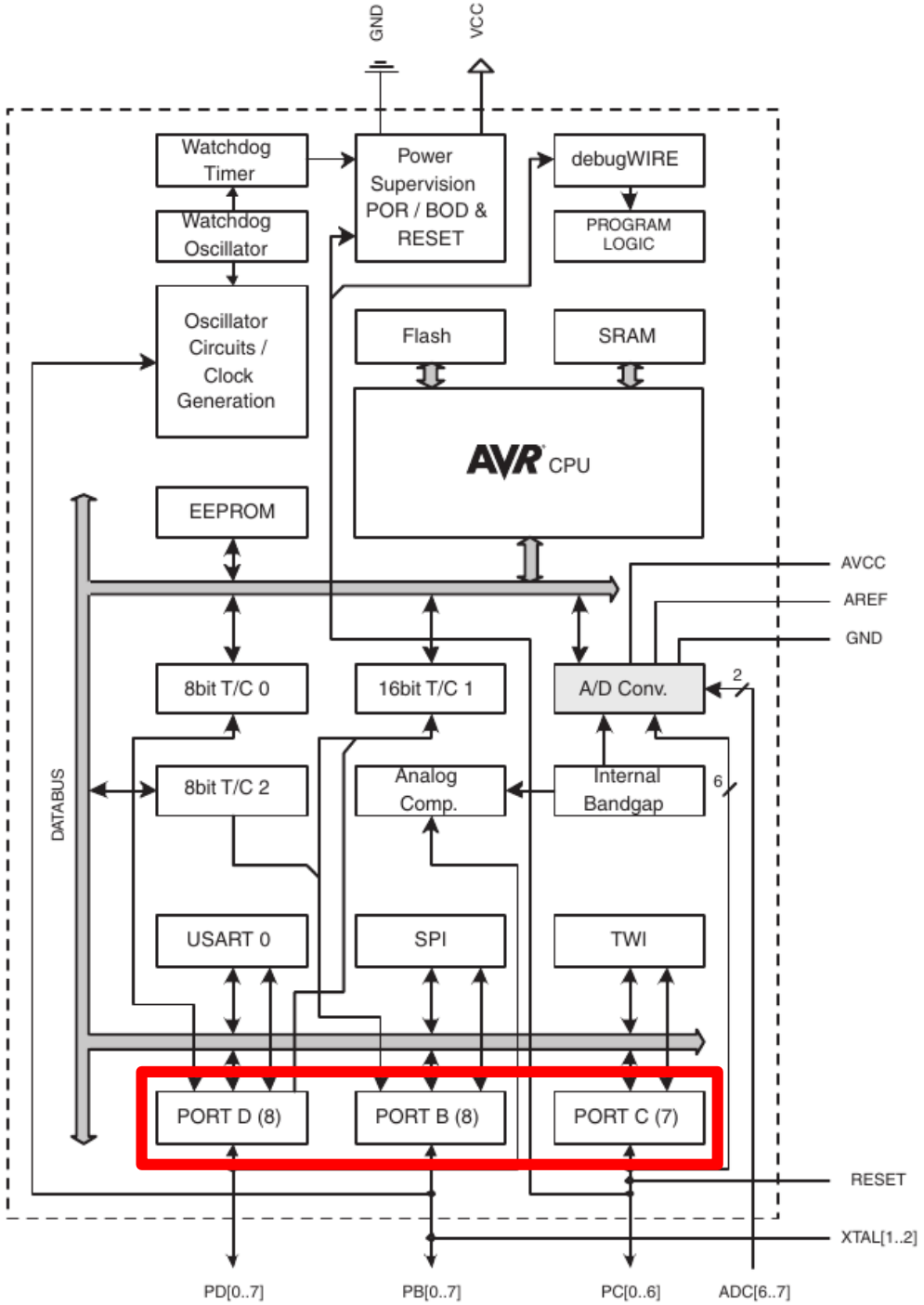
Sistemas con Microprocesadores

3.2. Recursos internos del ATmega328P

3.3. Adaptación e interfaces con sensores

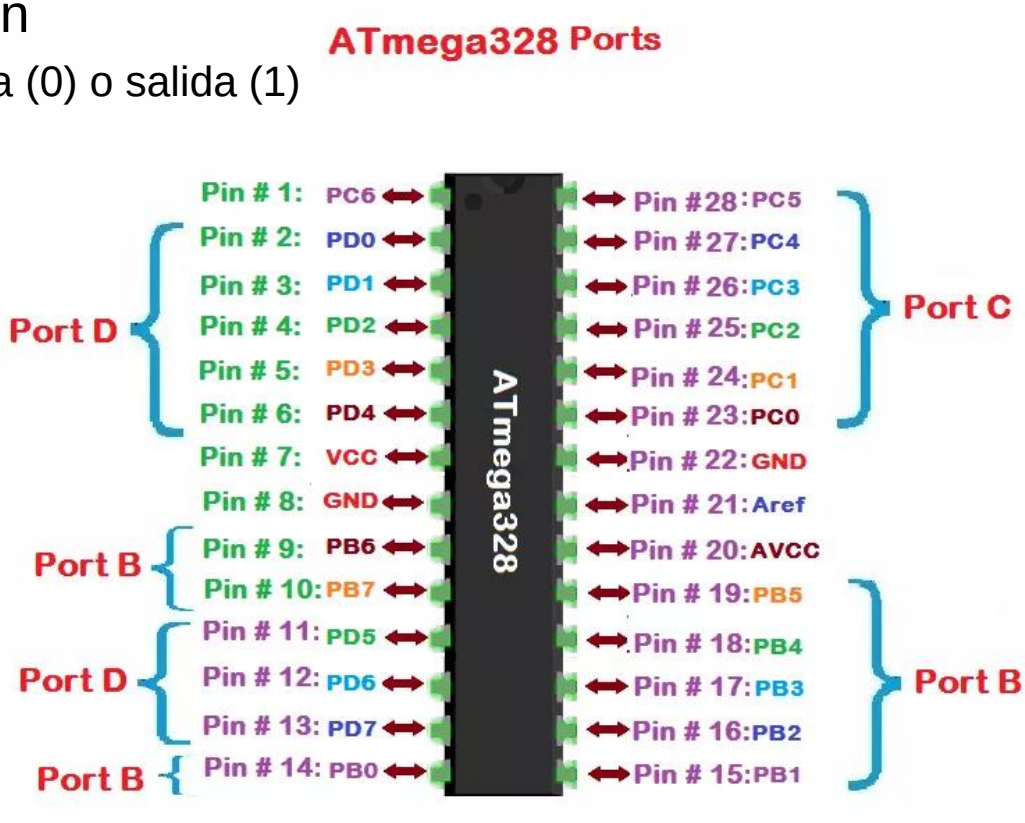
Arquitectura ATmega328P

- Patillas de E/S
  - Cada patilla es configurable:
    - Dirección
    - Valor
    - *Pull-up*
- Analógicas vs. digitales
- Patillas especiales:
  - Reloj
  - *Reset*



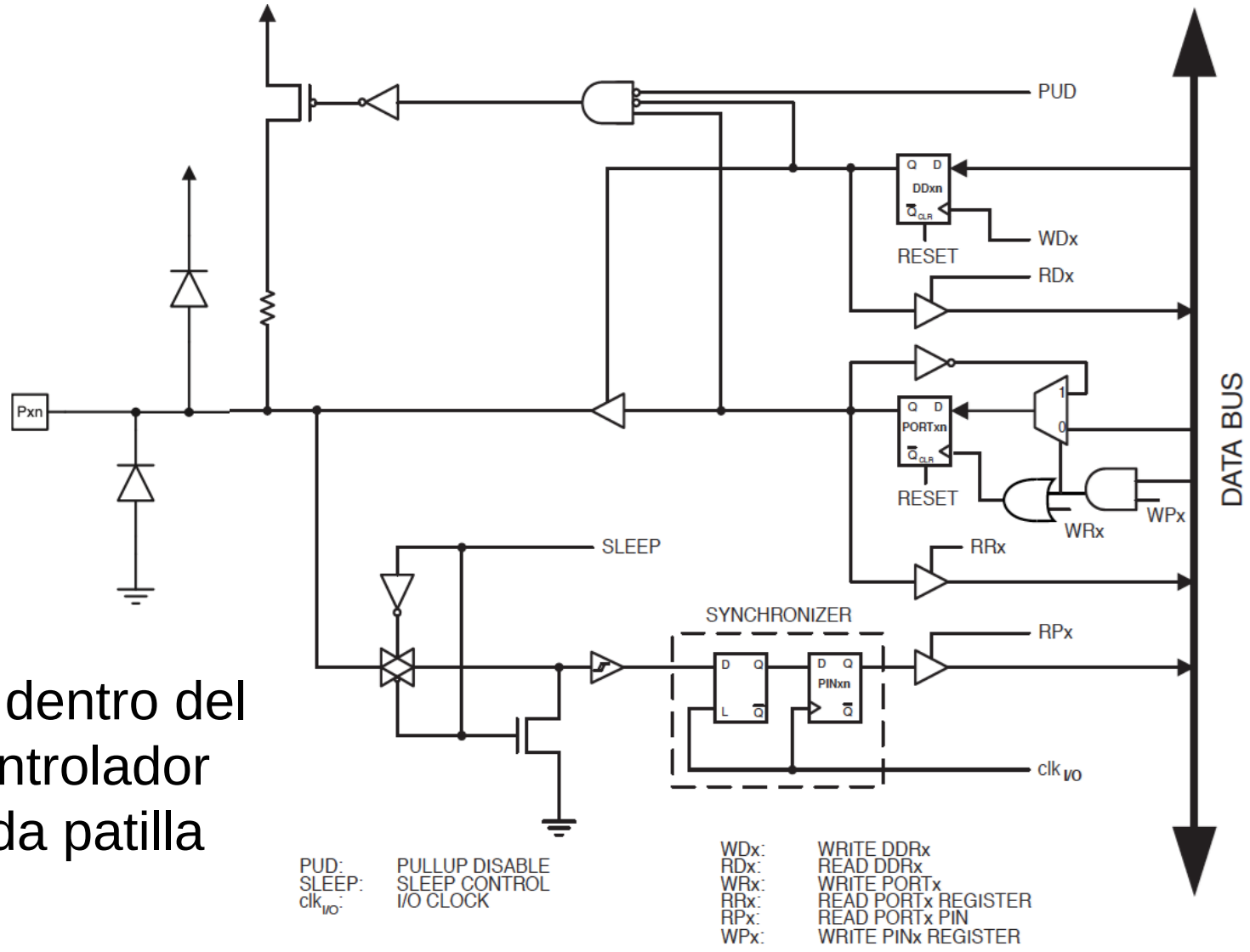
Entradas / Salidas digitales de propósito general (GPIO)

- Hay 3 puertos de 8 bits (B, C y D) conectados a las patillas.
- Cada puerto es controlado por 3 registros de 8 bits:
  - Cada bit controla la patilla correspondiente a su posición
  - **DDRx** – registro de dirección
    - Define si una patilla es entrada (0) o salida (1)
  - **PINx** – valor de entrada
    - Al leer este registro obtenemos el valor que tiene la patilla
  - **PORTx** – valor de salida
    - Al escribir en este registro establecemos el valor de la patilla (o configuramos la resistencia interna de *pull-up* si la patilla está configurada como entrada)



Entradas / Salidas digitales (GPIO)

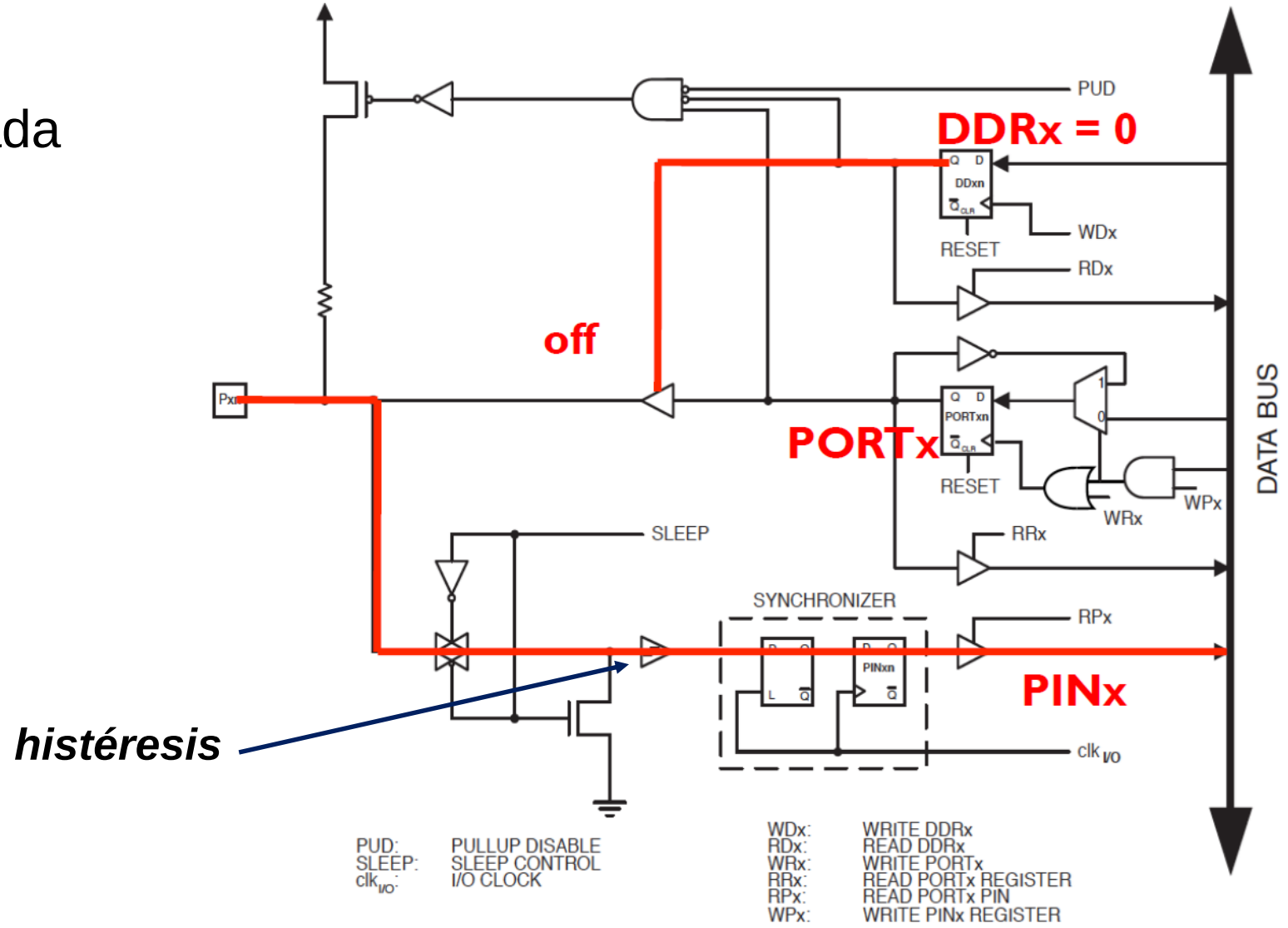
- Circuito dentro del microcontrolador para cada patilla de E/S



Note: 1. WRx, WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

Entradas / Salidas digitales (GPIO)

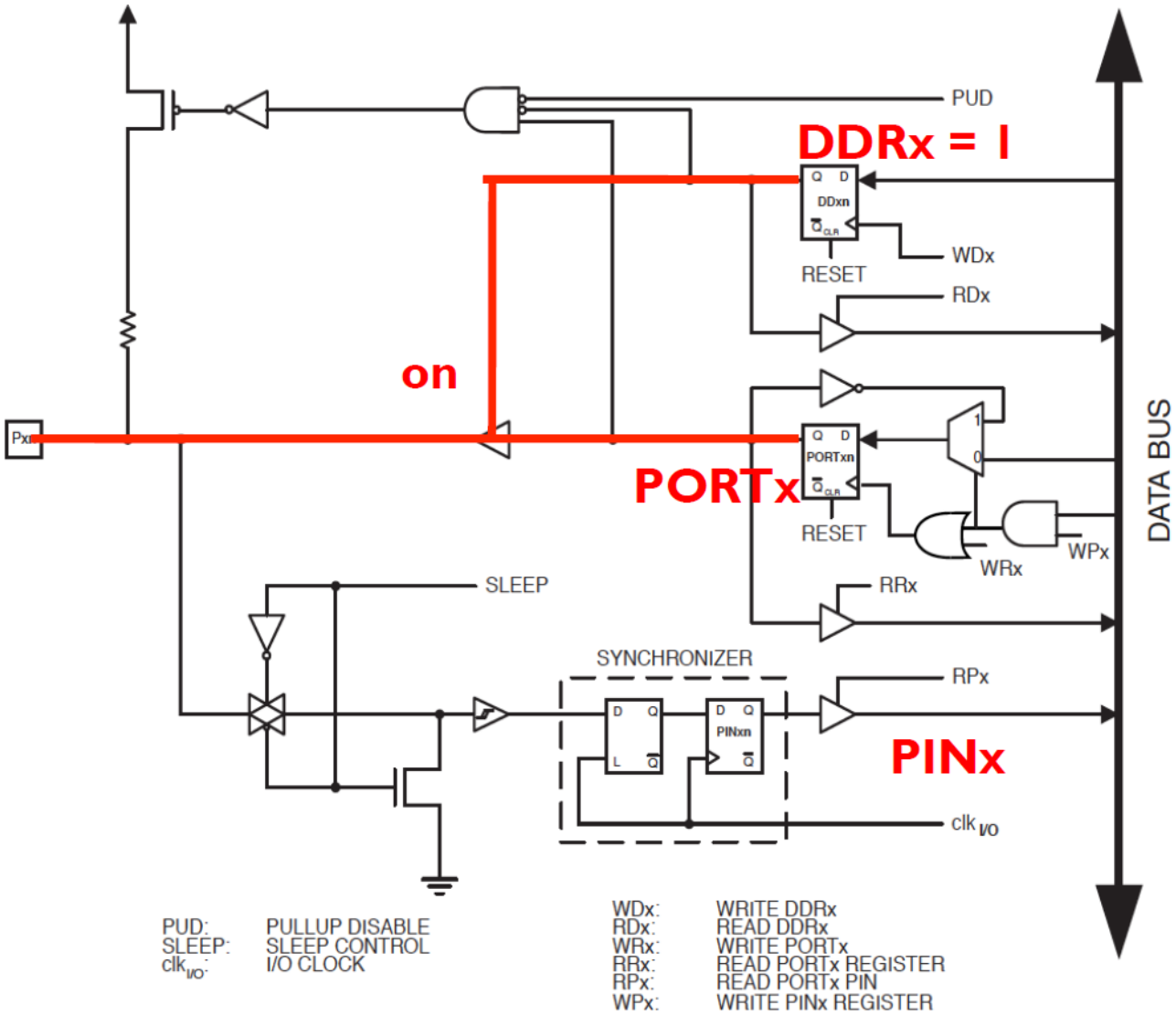
- Entrada



Note: 1. WRx, WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

Entradas / Salidas digitales (GPIO)

- Salida



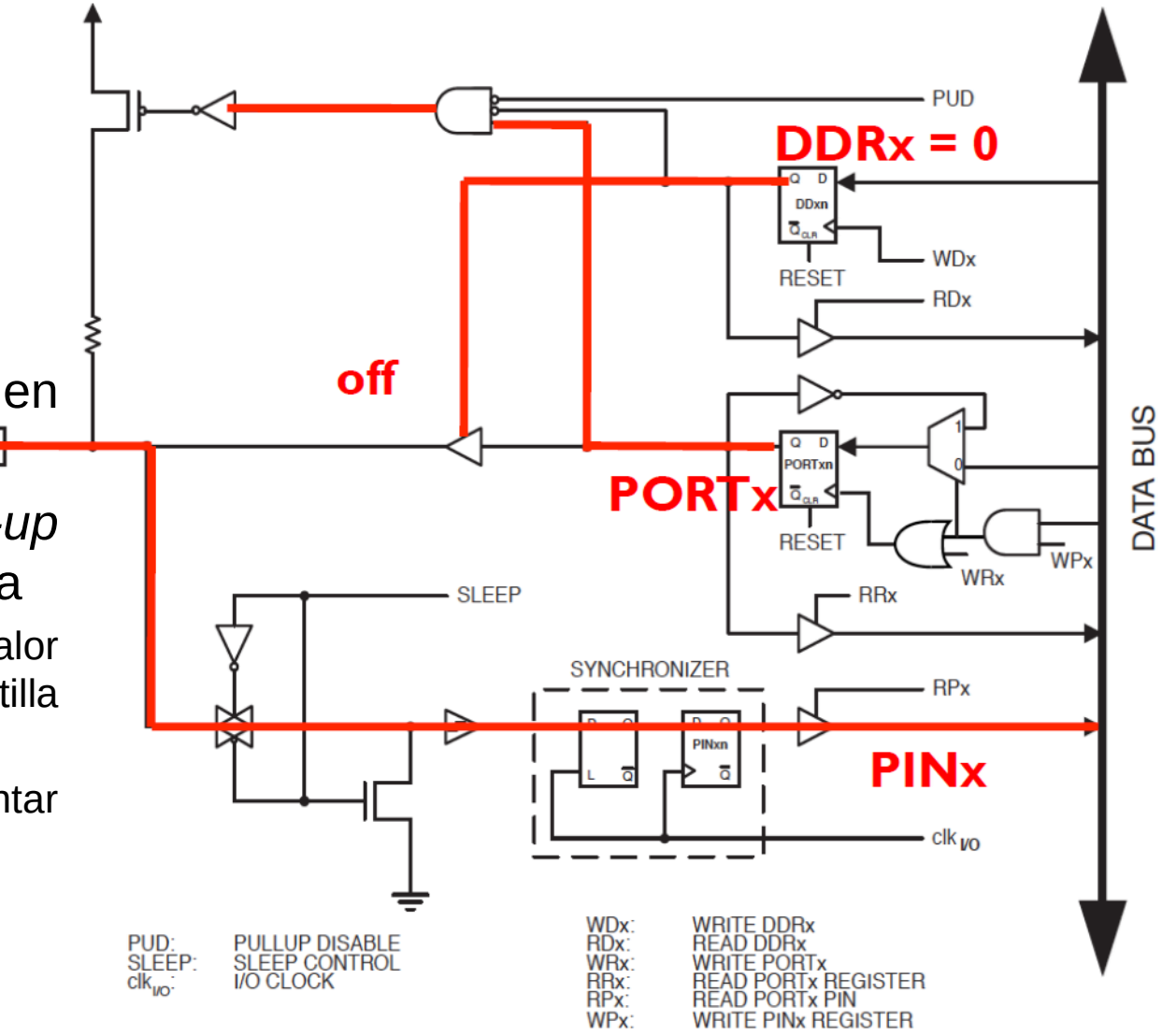
Note: 1. WRx, WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

Entradas / Salidas digitales (GPIO)

- Entrada con *pull-up* interno:

$DDRx_i = 0$ y:

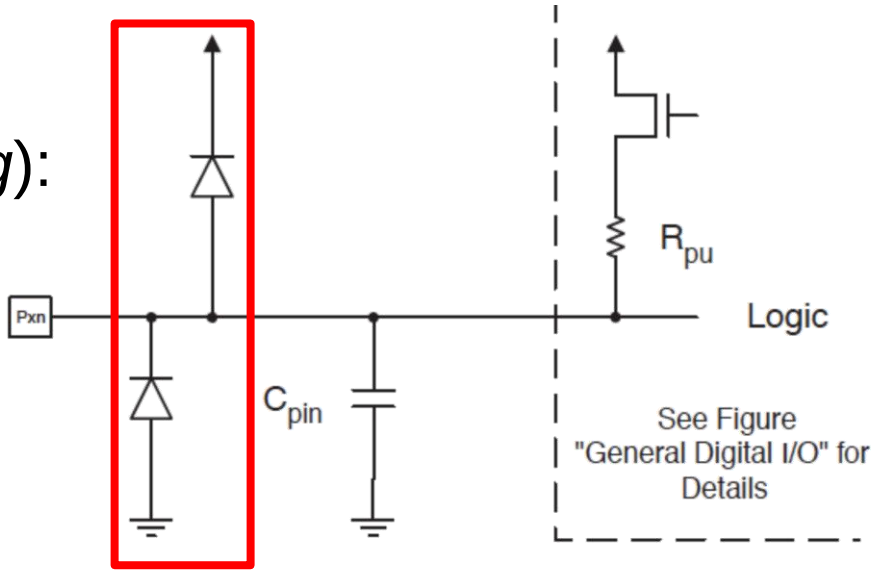
- $PORTx_i = 0$ patilla en alta impendancia
- $PORTx_i = 1$ *pull-up* conectada a la patilla
  - Asegura un valor conocido en la patilla en todo momento
  - Permite implementar OR cableadas



Note: 1. WRx, WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

Entradas / Salidas digitales (GPIO)

- Diodos de protección (*clamping*):



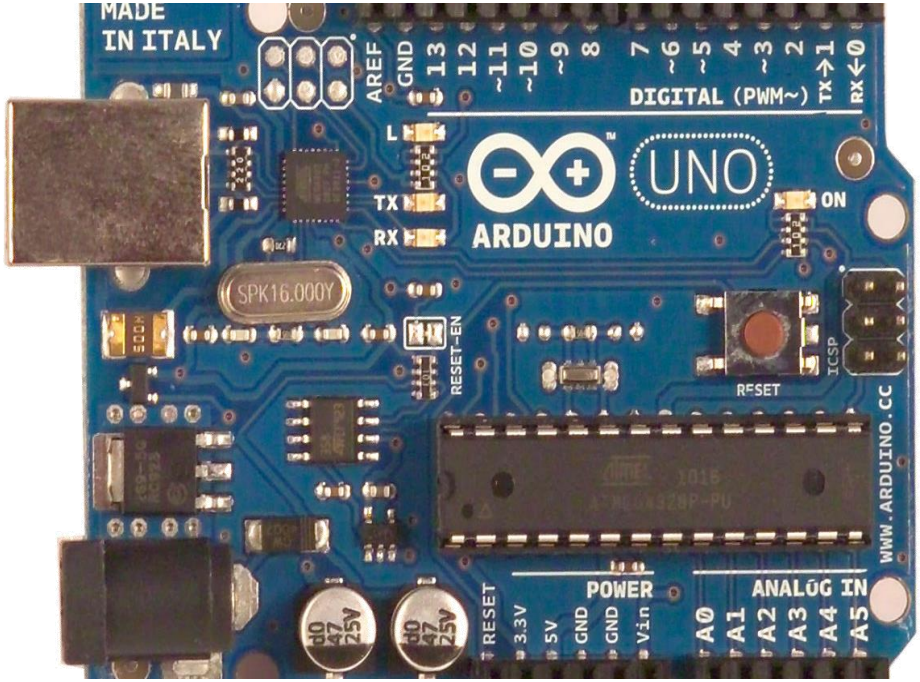
- Límites absolutos:

28.1 Absolute Maximum Ratings* *NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on $\overline{\text{RESET}}$ 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_{CC} and GND Pins	200.0 mA

Conexiones en Arduino

- Pines Digitales:
  - Pines 0 – 7: PORT D [0:7]
  - Pines 8 – 13: PORT B [0:5]
  - Pines 14 – 19: PORT C [0:5] (entradas analógicas Arduino 0 – 5)
  - Patillas 0 y 1 también pueden ser RX y TX para comunicaciones serie
  - Pin 13 conectado a un led en la mayoría de placas Arduino



Arduino

- Control Entrada/Salida:
  - **pinMode(pin, dir)**
    dir=OUTPUT/INPUT/INPUT_PULLUP}
  - Ej: **pinMode(13, OUTPUT)**
- Entrada:
  - **var = digitalRead(pin)**
- Salida: (máx. 40 mA)
  - **digitalWrite(pin,HIGH)**
  - **digitalWrite(pin,LOW)**
- Entrada con *pull-up*:
  - **pinMode(pin,INPUT_PULLUP)**

C AVR

- **DDR*x* =** máscara de bits
  x=B/C/D
- Ej: **DDRB = DDRB | 0x20**

```
*(uint8_t *)0x24 =
    *(uint8_t *)0x24 | 0x20
```

- **var = PIN*x* &** máscara
- **PORT*x* = PORT*x* |** máscara
- **PORT*x* = PORT*x* & ~**máscara
- **PORT*x* = PORT*x* |** máscara
- **PORT*x* = PORT*x* & ~**máscara

Ejemplo (Arduino)

```
// Blink.ino
// El pin 13 de la placa (patilla PB5 del microcontrolador)
// tiene el LED conectado
#define PIN_LED 13

// setup() se ejecuta al principio:
void setup()
{
  pinMode(PIN_LED, OUTPUT); // La configura como salida
}
// loop() se ejecuta una y otra vez continuamente:
void loop()
{
  digitalWrite(led, HIGH);    // Enciende el LED (HIGH)
  delay(1000);                // Espera 1 segundo (1000 ms)
  digitalWrite(led, LOW);     // Apaga el LED (LOW)
  delay(1000);                // Espera 1 segundo (1000 ms)
}
```

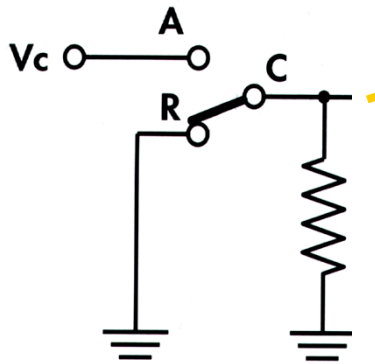
Ejemplo (C AVR)

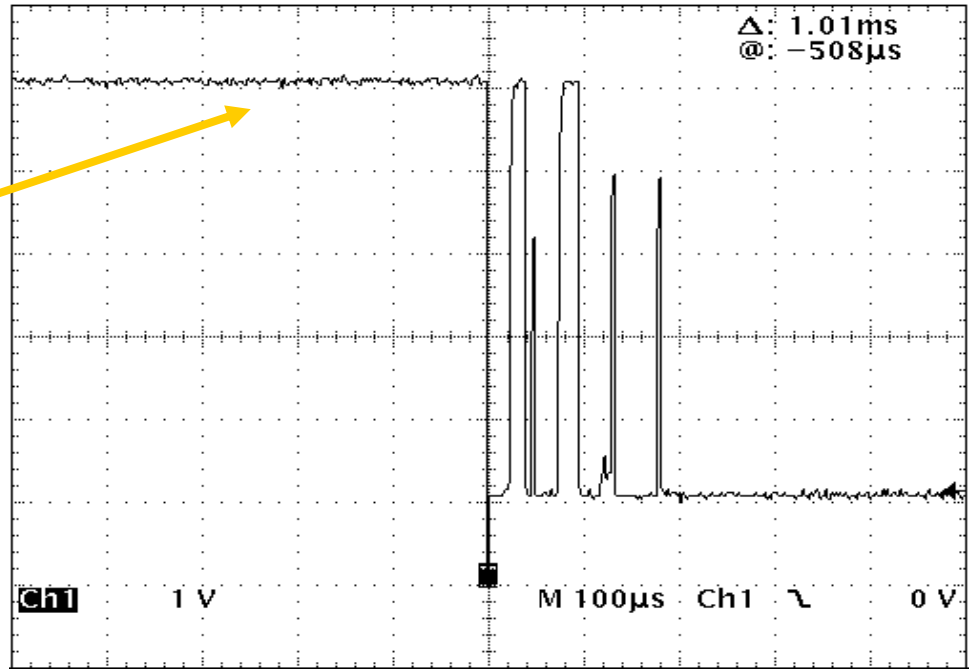
```
// Blink.c
#include <avr/io.h>
#define F_CPU 16000000 // 16 Mhz: frec. cristal del Arduino UNO
#include <util/delay.h>

#define DDR_LED DDRB  // La patilla PB5 tiene el LED conectado
#define PORT_LED PORTB
#define MASK_LED 0x20  // Máscara de bits: 0b00100000 (bit 5 a
1)
int main()
{
  DDR_LED = DDR_LED | MASK_LED; // La configura como salida
  while(1)
  {
    PORT_LED = PORT_LED | MASK_LED; // Enciende el LED (HIGH)
    _delay_ms(1000)                 // Espera 1 segundo (1000 ms)
    PORT_LED = PORT_LED & ~MASK_LED; // Apaga el LED (LOW)
    _delay_ms(1000);                // Espera 1 segundo (1000 ms)
  }
```

Sensores y elementos de entrada digital básicos

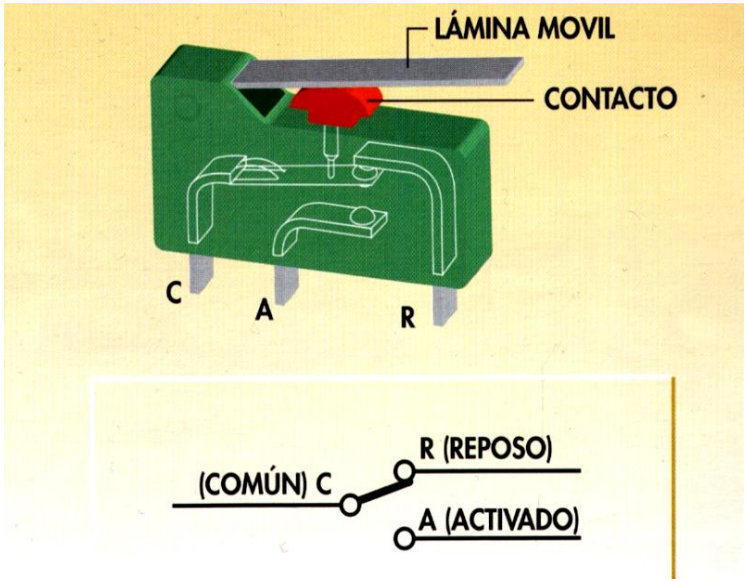
- Sensores mecánicos: Pulsadores, interruptores y conmutadores mecánicos.
- Problema: rebote del botón. Cuando sea crítico, los rebotes deben eliminarse por hardware o por software (ej. efectuando una lectura retardada para leer en la zona estable).



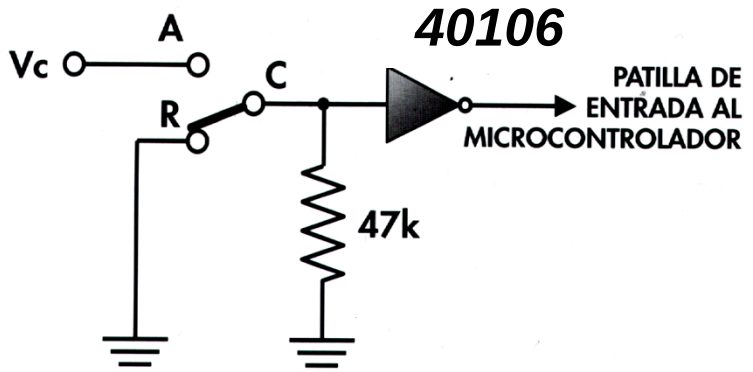


Sensores y elementos de entrada digital básicos

- Un ejemplo de sensor mecánico: el sensor de contacto o de fin de carrera (*bumper*).

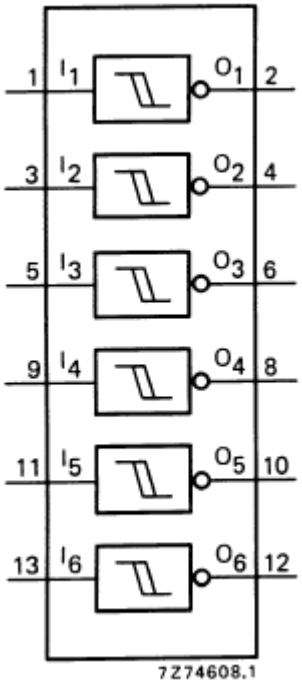


Ejemplo de conexión al microcontrolador con *Schmitt trigger* externo:

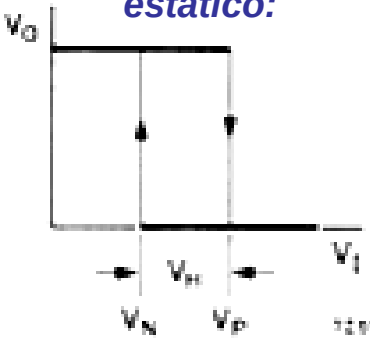


Sensores y elementos de entrada digital básicos

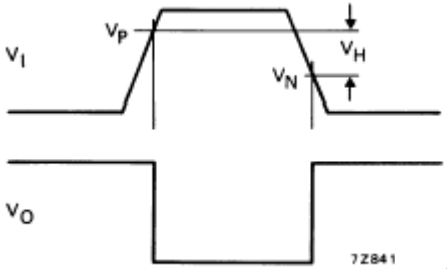
- Discretización de niveles de entrada mediante el C.I. 40106:
- 6 inversores digitales con histéresis “Schmitt *trigger*”

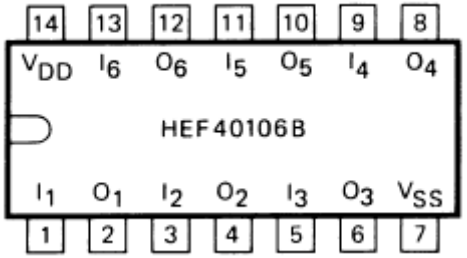


Comportamiento estático:



Comportamiento dinámico:





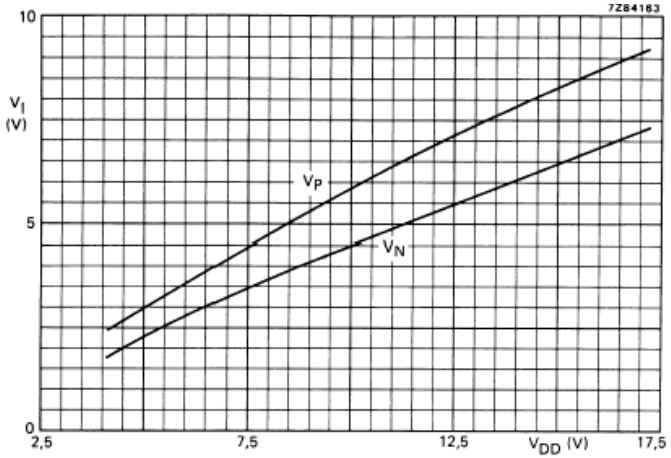
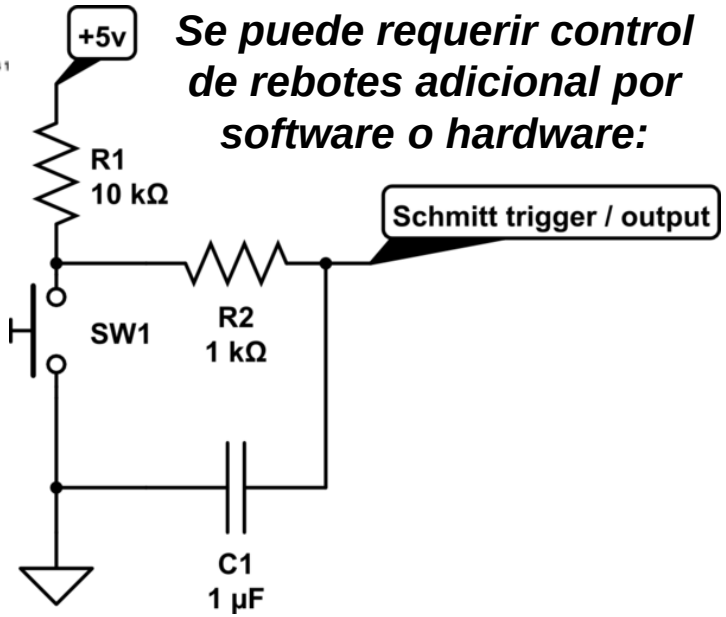
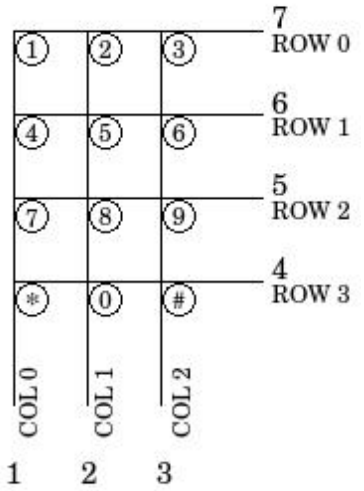


Fig.9 Typical switching levels as a function of supply voltage V_{DD}; T_{amb} = 25 °C.

Se puede requerir control de rebotes adicional por software o hardware:



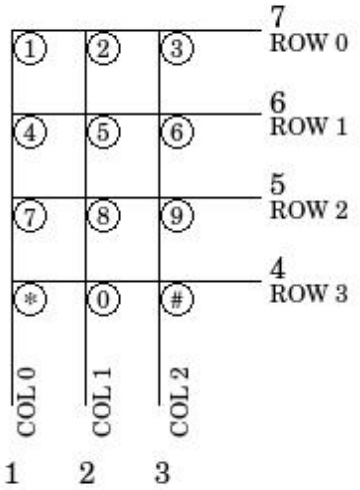
Teclado matricial




```
#include <Keypad.h>
#define ROWS 4 // 4 filas
#define COLS 3 // 3 columnas
#define PIN_LED 13
// Define el teclado
char keys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'#','0','*'}};
// Conectar teclado ROW0, ROW1, ROW2 y ROW3 a esos pines
uint8_t rowPins[ROWS] = { 9, 8, 7, 6 };
// Conectar teclado COL0, COL1 y COL2 a esos pines
uint8_t colPins[COLS] = { 12, 11, 10 };
// Crea el teclado
Keypad kpd = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
void setup() {
  pinMode(PIN_LED, OUTPUT);
  digitalWrite(PIN_LED, HIGH);
  Serial.begin(9600);
}
```

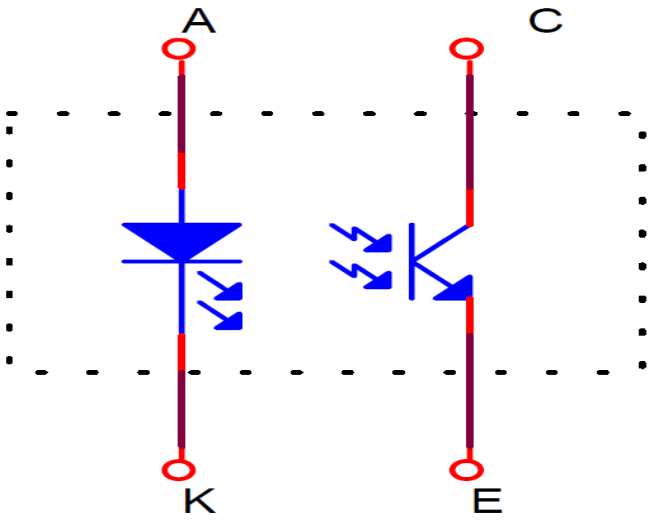
Teclado matricial

```
void loop()
{
  char tecla = kpd.getKey();
  if(tecla)
  {  // comprobar si hay una tecla válida
    switch (tecla)
    {
      case '*':
        digitalWrite(PIN_LED, LOW);
        break;
      case '#':
        digitalWrite(PIN_LED, HIGH);
        break;
      default:
        Serial.println(tecla);
    }
  }
}
```



Sensores y elementos de entrada digital básicos

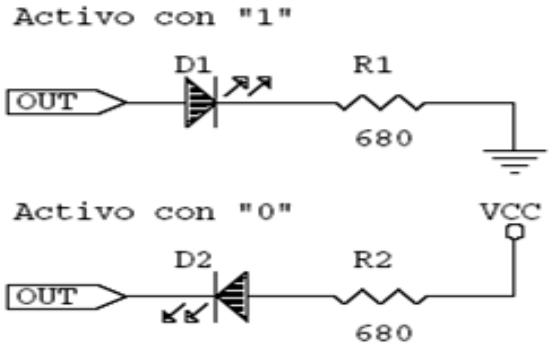
- Sensores ópticos: El principio básico es el “acoplamiento óptico”, creado por una pareja emisor (diodo) y receptor (fototransistor). Se utiliza en entradas y salidas con las que se desea un aislamiento eléctrico (comunicación por luz).
- Existen variantes de encapsulado utilizadas como sensores .



- El transistor receptor deja pasar (entre C y E) una corriente proporcional a la que circula por el diodo emisor (de A a K).

- Para conectar el transistor a una patilla de entrada se requiere al menos una resistencia de *pullup*.

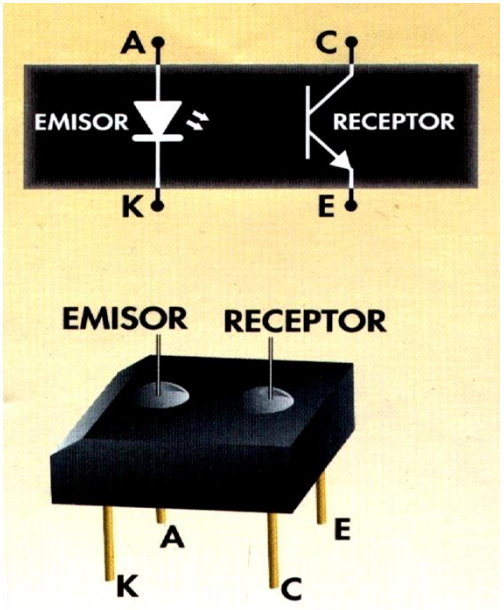
- Para conectar el diodo al una patilla de salida, se emplea resistencia en serie de protección (limita la corriente que pasa por el diodo), igual que cuando conectamos un LED:



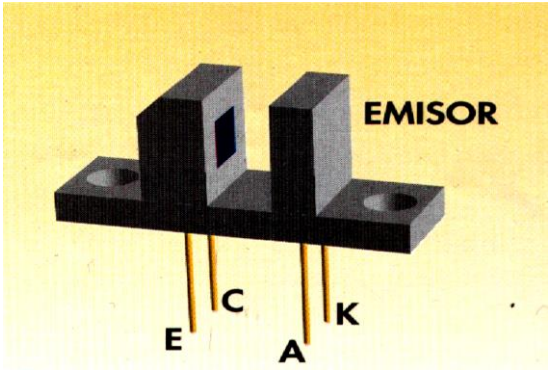
Sensores y elementos de entrada digital básicos

- Unos ejemplos de sensores ópticos por infrarrojos son el CNY70 (por reflexión) y el H21A1 (directo).

CNY70

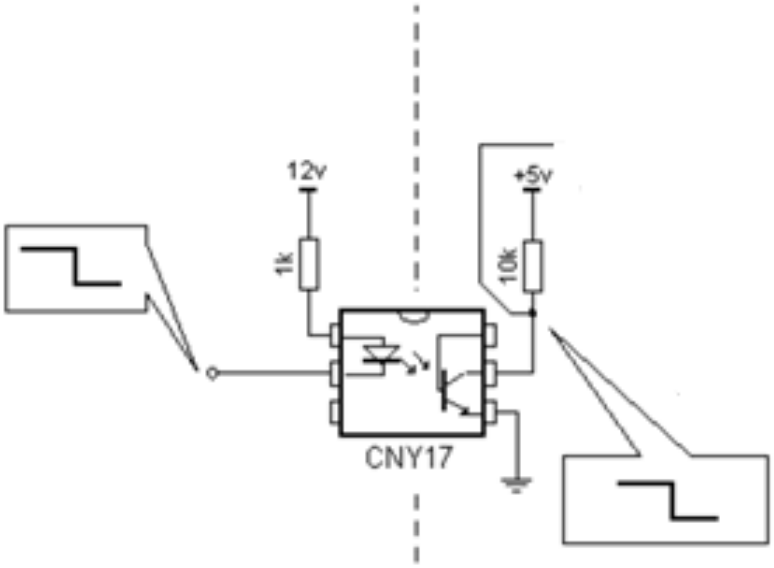


H21A1

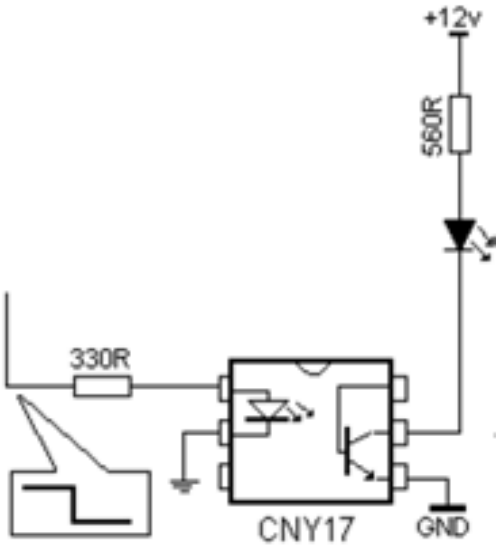


Aplicaciones en seguimiento de líneas, control de velocidad en cintas y motores, *encoder* de desplazamiento angular o lineal. También como entrada con acoplamiento óptico.

Optoacopladores



Entrada optoacoplada

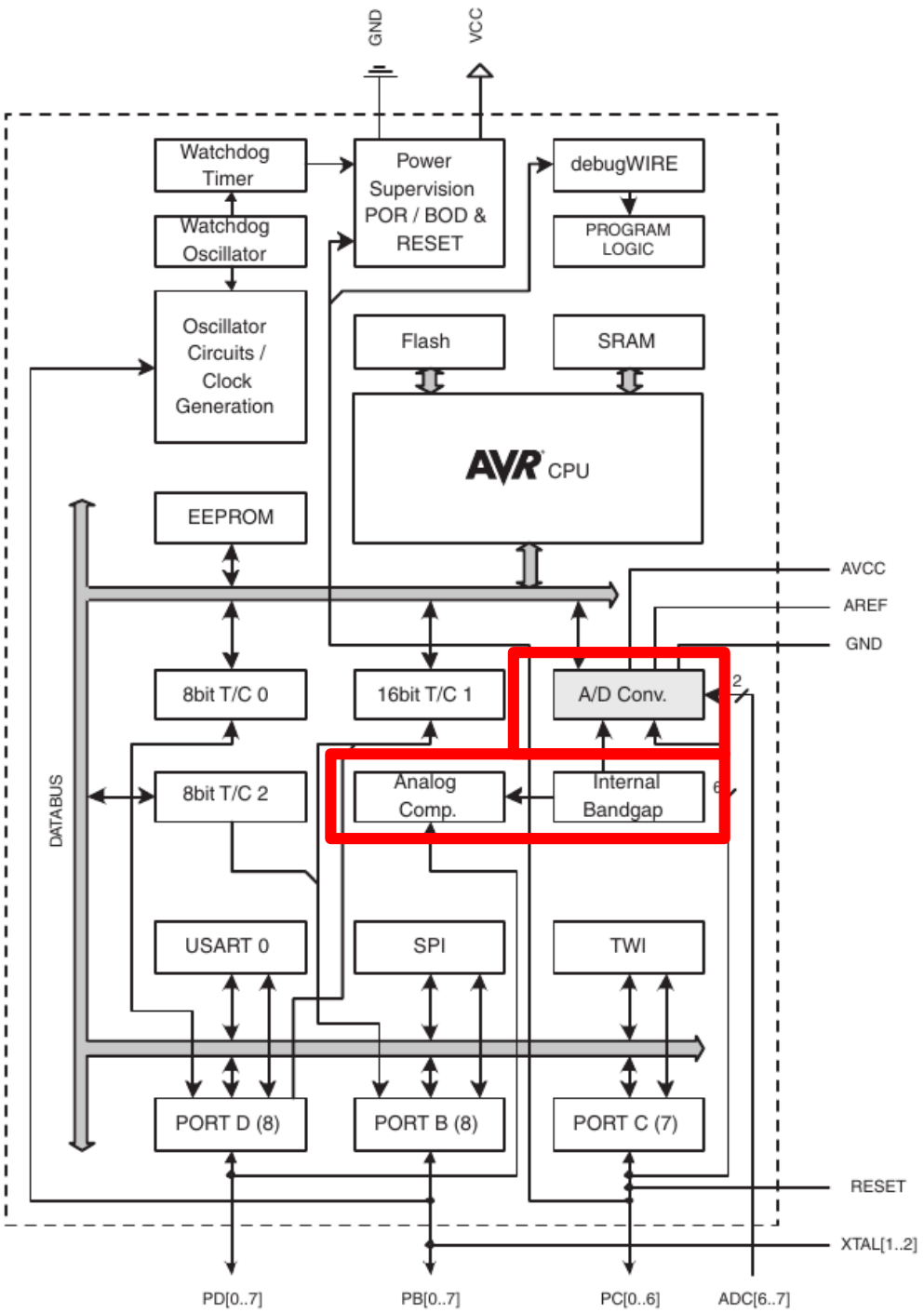


Salida optoacoplada

Tipos de optoacopladores: convencional (aislamiento), Darlintong (intensidad más alta de salida), con Triac (control de cargas AC), lineales (calidad de señal), de alta velocidad (comunicaciones), abiertos (encoders) ver diapositiva anterior.

Arquitectura ATmega328P

- Entradas analógicas
- Conversor A/D:
  - Voltaje a 10 bits digital
- Se puede usar referencia externa (defecto 0-5V)

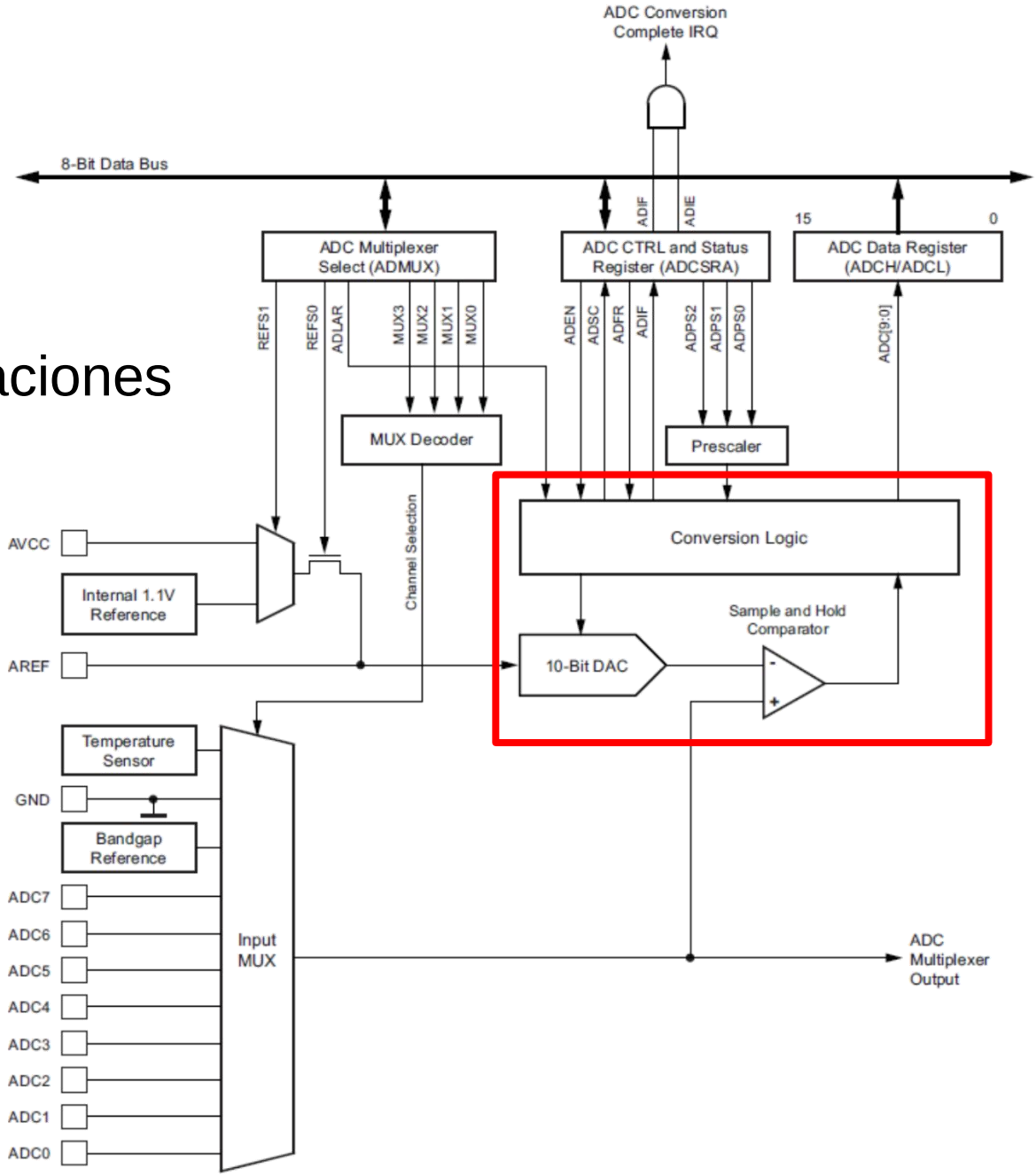


Entradas analógicas

- Entradas analógicas pines: 0 – 5
  - Si se utilizan como entradas/salidas digitales pueden referenciarse como A0...A5
- Funciones entrada analógica:
  - **`x = analogRead(pin);`**
  - Tarda unos 100 uS → hasta 10.000 muestras/segundo
  - Convierte voltaje 0 – 5 V a 10 bits (0 – 1023)
  - No es imprescindible utilizar **pinMode**
  - **`analogReference(tipo)`** Donde tipo puede ser
    - DEFAULT: Es el valor de referencia analógico que viene por defecto de 5 V en placas Arduino de 5 V, y de 3,3 V en placas Arduino que funcionen con 3,3 V.
    - INTERNAL: Es una referencia de tensión interna de 1,1V en el ATmega328P.
    - EXTERNAL: Se usará una tensión de referencia externa que tendrá que ser conectada al pin AREF.

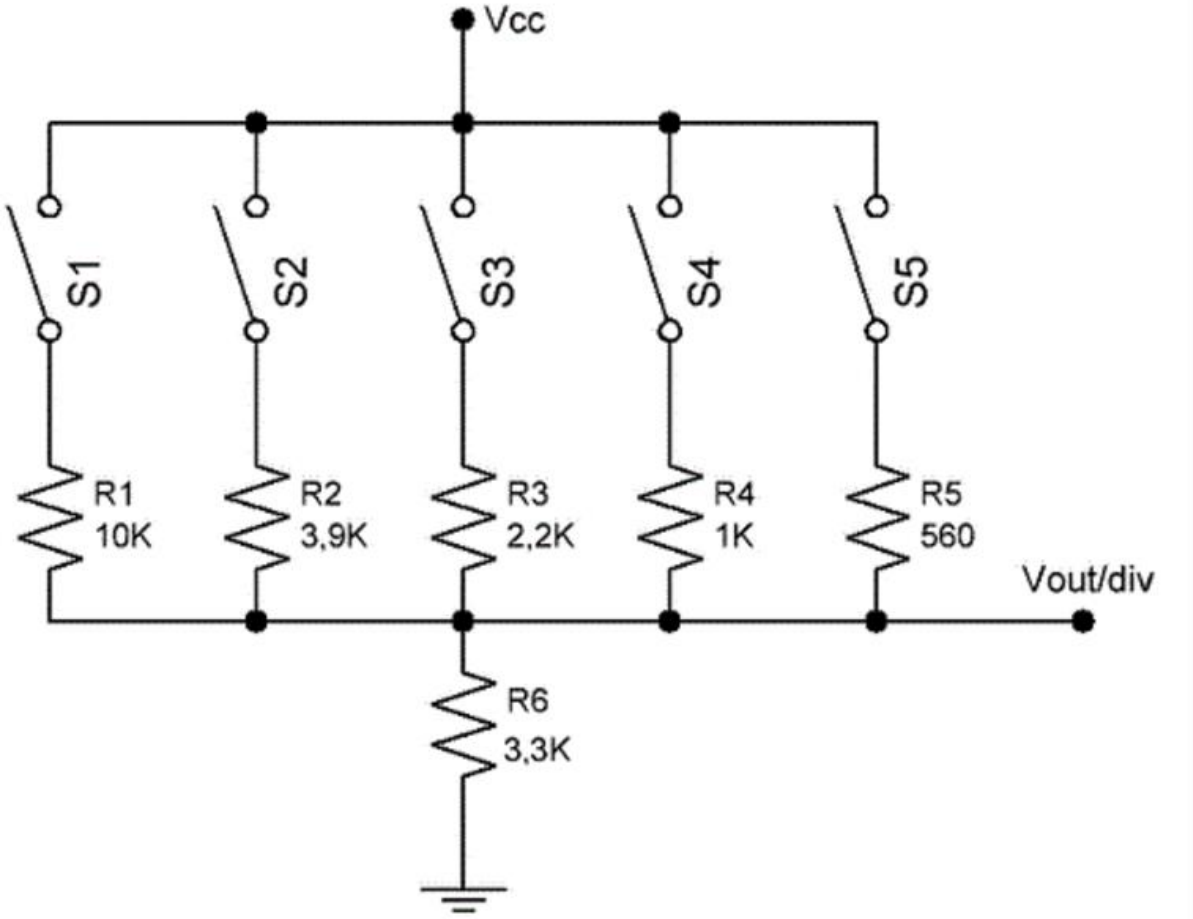
Conversor A/D

- Aproximaciones sucesivas

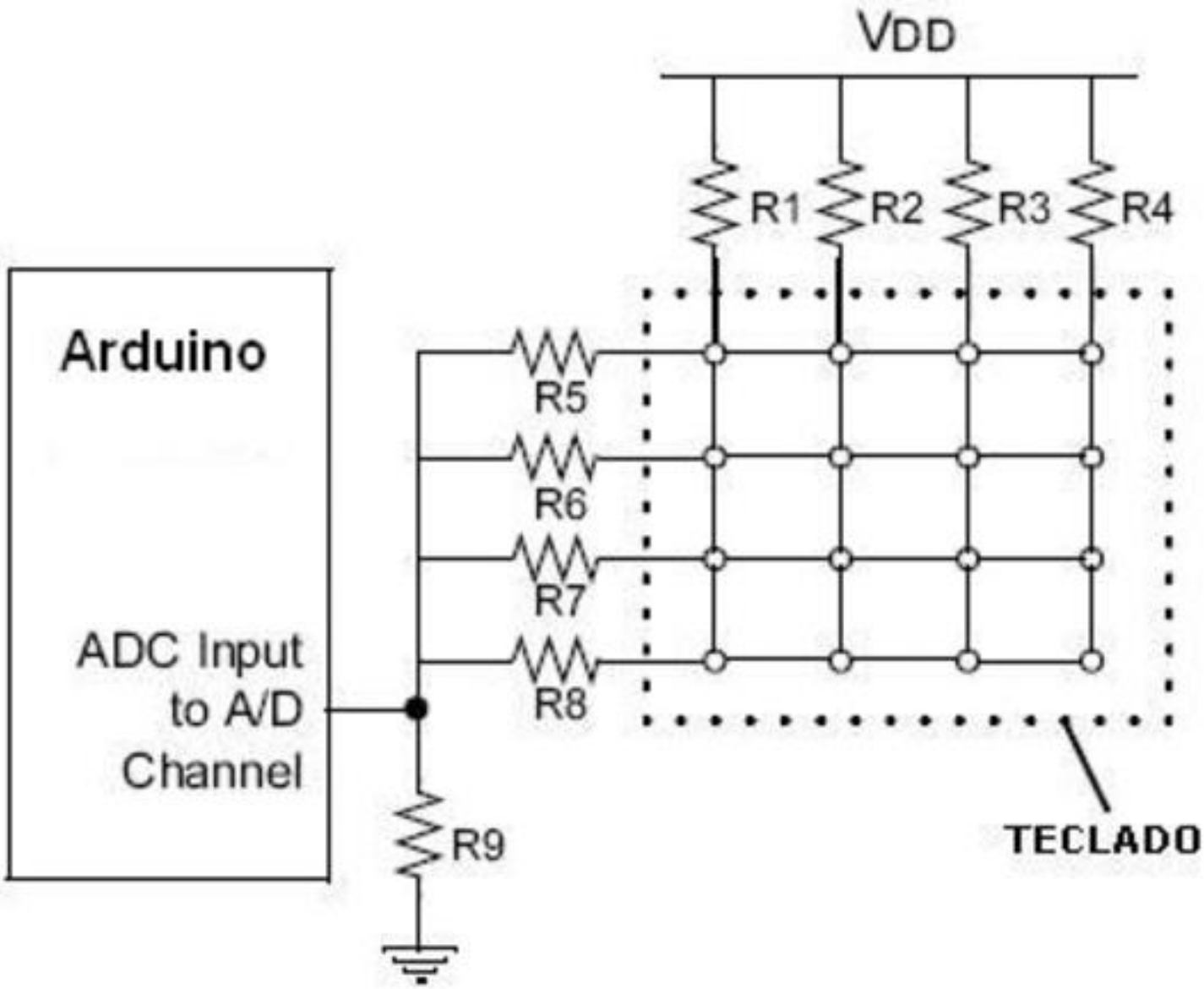


Teclado analógico

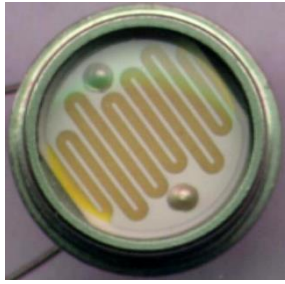
- Permiten la identificación de una tecla pulsada en función del voltaje obtenido.



Teclado analógico

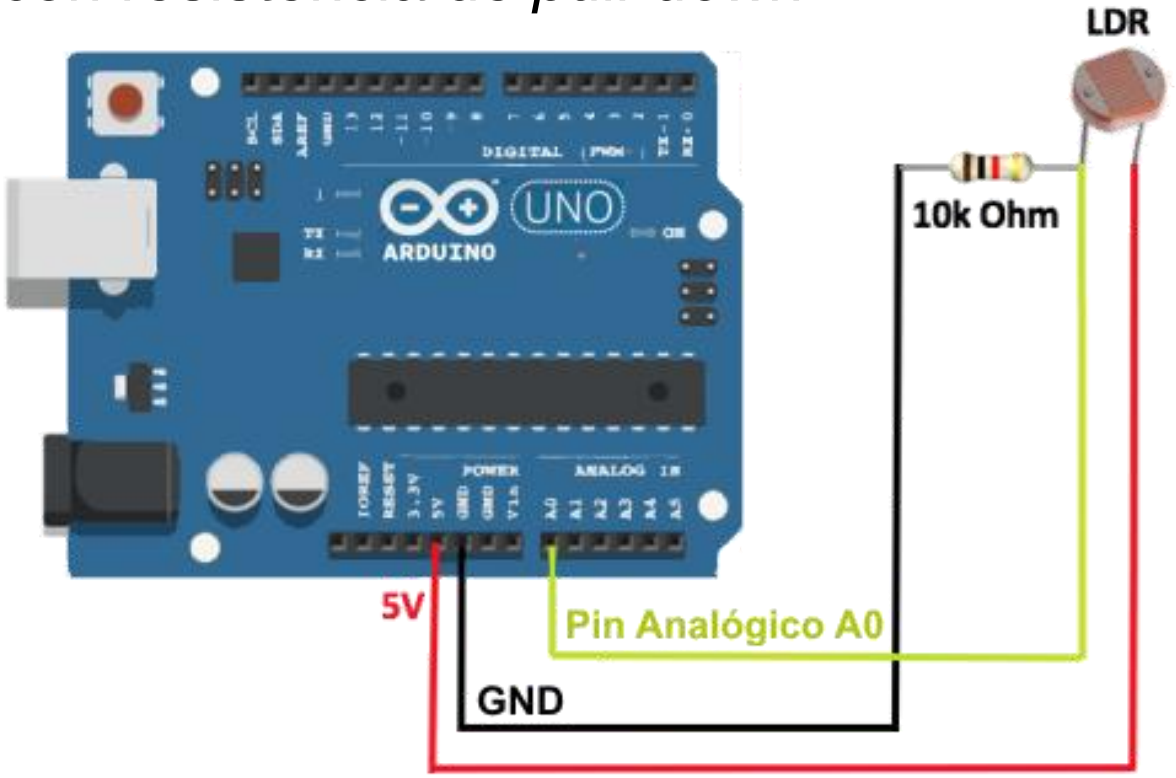


Fotorresistencia (LDR)



- Resistencia variable en función de la luz incidente.

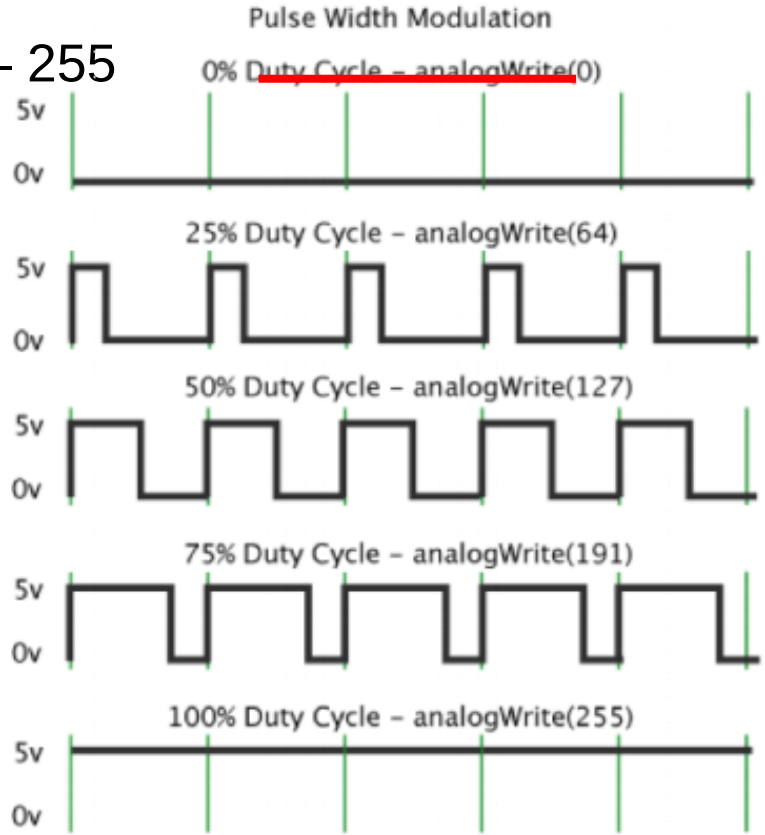
- Conexión como si se tratase de un botón pulsador (*push-button*) con resistencia de *pull-down*.



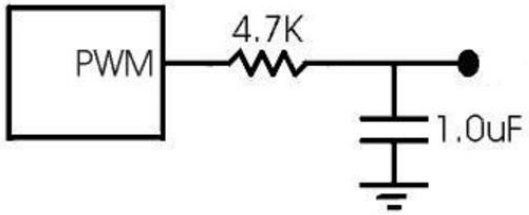
Salidas “analógicas” (PWM): modulación por ancho de pulsos

- Los pines con salidas PWM genrado por hardware son: 3, 5, 6, 9, 10, 11 (pines digitales)

- Salida “analógica” (mediante PWM)
  - **`analogWrite(pin, valor)`** valor: 0 – 255
  - “*duty-cycle*” representa el valor.
  - Frecuencia: 490 Hz
    (980 Hz en pines 5 y 6)



- Se puede convertir en una señal analógica mediante un filtro integrador



Utilidades PWM

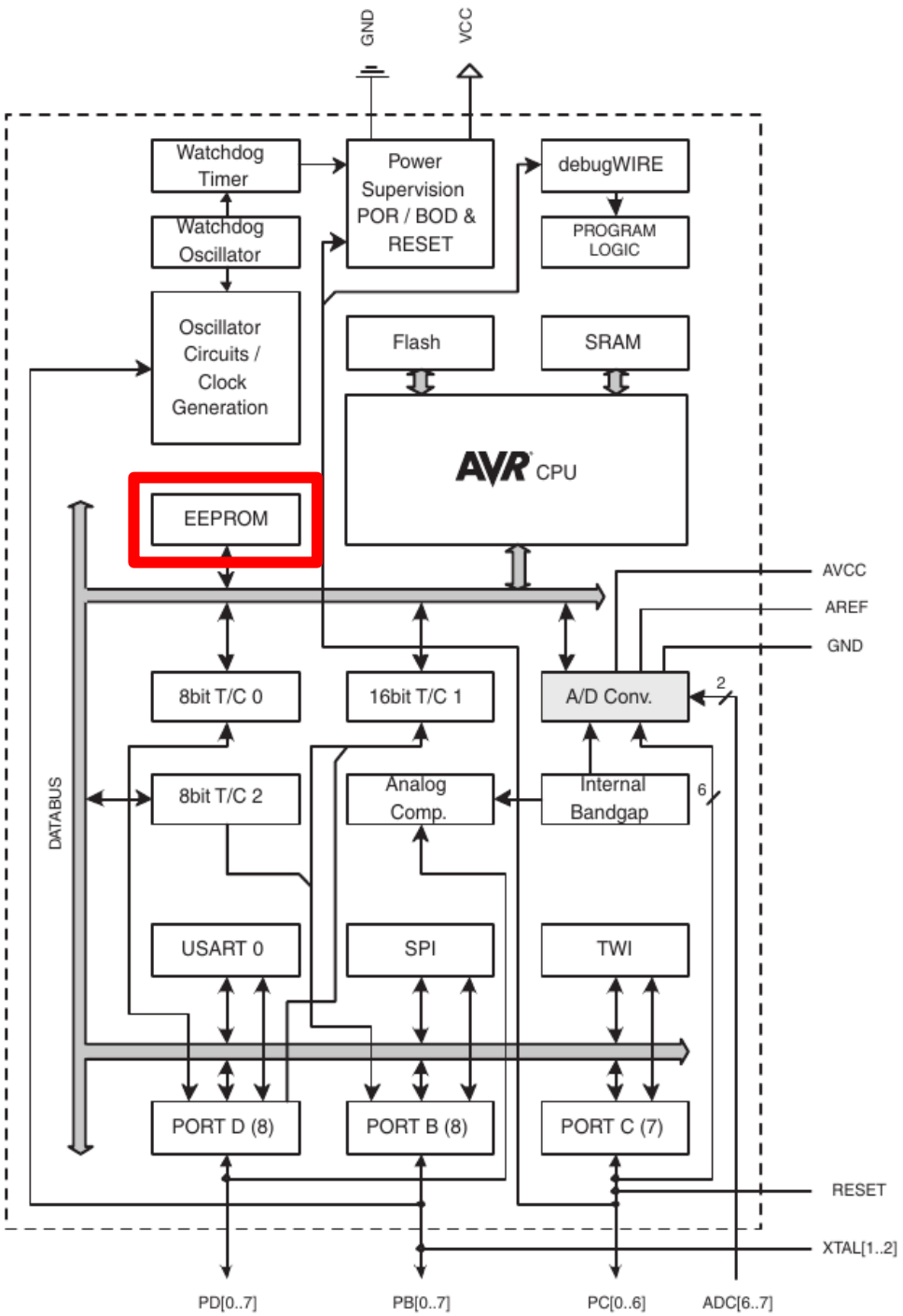
- Control de velocidad de motores
- Control de servomotores
- Control de luminosidad de un LED
- Generación de tonos y sonidos
- ¿Alguna más?

Arquitectura ATmega328P

- Memoria EEPROM 1 kB

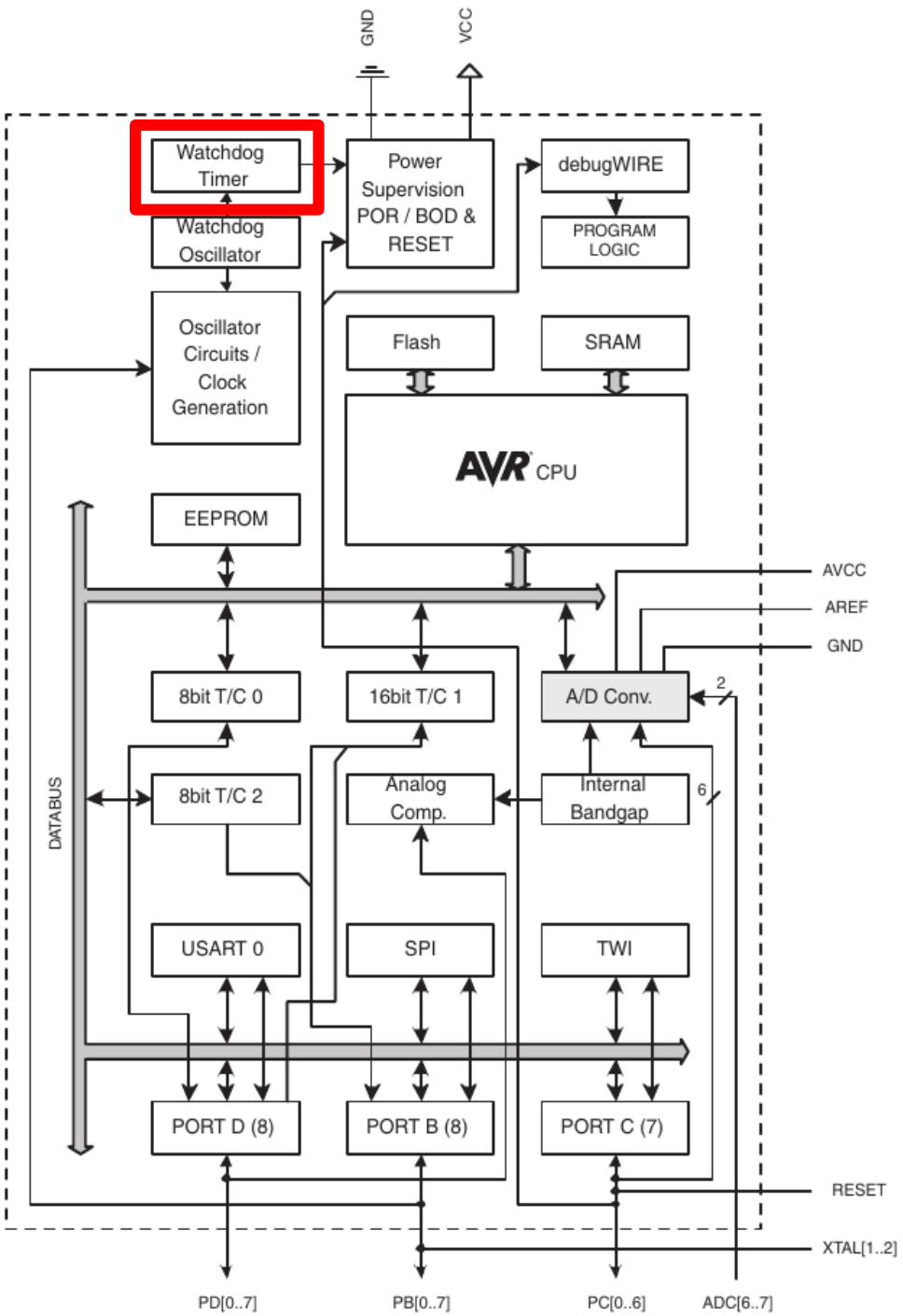
```
#include <EEPROM.h>
...
valor = EEPROM.read(dirección);
...
EEPROM.write(dirección, valor);
```

- Tiempo escritura: 3,3 ms
- 100.000 ciclos de escritura



Arquitectura ATmega328P

- *Watchdog* (perro guardián)
- Reinicia el microcontrolador en caso de que se bloquee la ejecución de programa.



Watchdog (perro guardián) Arduino y C AVR

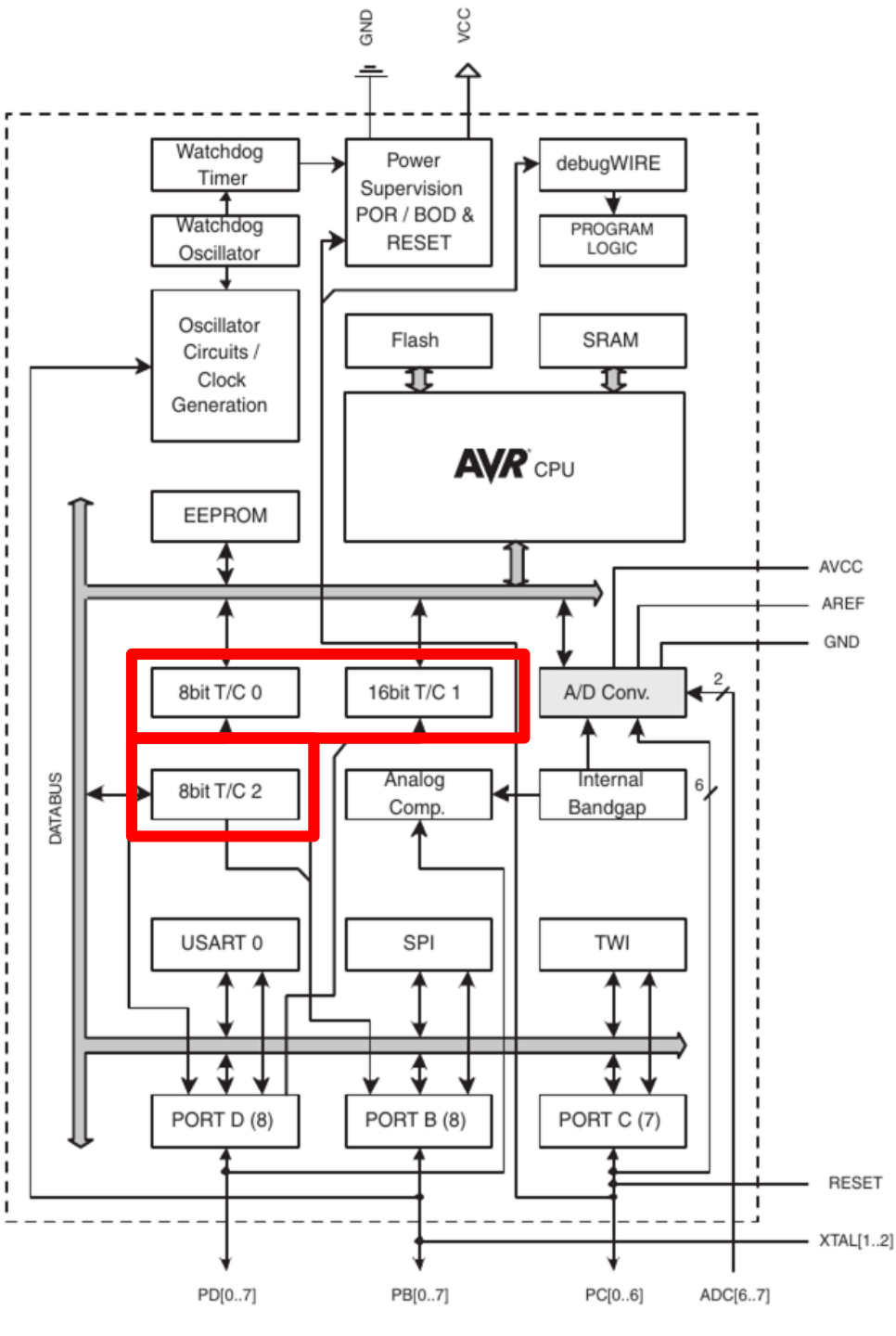
```
#include <avr/wdt.h>
// Al comenzar (en la función setup la 1ª línea para código
// Arduino) lo podemos deshabilitar si lo tenemos habilitado
// por defecto y el código de setup es lento
wdt_disable();

// En la última línea de la función setup lo activamos e
// indicamos el tiempo para reinicio, en este caso 250 ms
wdt_enable(WDTO_250MS);

// Valores posibles: WDTO_15MS, WDTO_30MS, WDTO_60MS,
// WDTO_120MS, WDTO_250MS, WDTO_500MS, WDTO_1S, WDTO_2S,
// WDTO_4S,WDTO_8S
// En la función loop llamamos a esta función que reinicia el
// contador del perro guardián
wdt_reset();
```

Arquitectura ATmega328P

- 3 Temporizadores o *timers*
- Registros con autoincremento: 8 bits (T0 y T2) – 16 bits (T1)
- Configurables:
  - Frecuencia de reloj
  - Permiten valor de "desbordamiento"
  - Generan interrupciones
  - Generan señales PWM



Temporización

Arduino

- **`delay(ms)`**

Produce una espera de `ms` milisegundos

- **`delayMicroseconds(us)`**

Produce una espera de `us` microsegundos

- **`unsigned long millis()`**

Devuelve los milisegundos transcurridos desde el comienzo de la ejecución

- **`unsigned long micros()`**

Devuelve los microsegundos desde el comienzo de la ejecución, con una resolución de 4 microsegundos

C AVR

- **`_delay_ms(ms)`**

- **`_delay_us(us)`**

Previamente hay que configurar un temporizador y hacer lecturas de sus registros para obtener las mediciones de tiempo.

Interrupciones

- Permiten al programa responder a ciertos eventos cuando ocurren
- Le evitan al programa tener que estar continuamente leyendo y comprobando valores de entrada (*polling*)
- Eventos externos: recepción UART, cambio de nivel en una patilla, detección de flancos en una patilla, …
- Eventos internos: fallo de alimentación, perro guardián, eventos de temporizadores, …

Interrupciones (vectores)

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B
14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow
18	0x0022	SPI, STC	SPI serial transfer complete
19	0x0024	USART, RX	USART Rx complete
20	0x0026	USART, UDRE	USART, data register empty
21	0x0028	USART, TX	USART, Tx complete
22	0x002A	ADC	ADC conversion complete
23	0x002C	EE READY	EEPROM ready
24	0x002E	ANALOG COMP	Analog comparator
25	0x0030	TWI	2-wire serial interface
26	0x0032	SPM READY	Store program memory ready

Interrupciones

- Para atender una interrupción determinada, debe estar habilitada en el correspondiente registro de configuración.

- Existe un bit global de habilitación de interrupciones que también debe estar habilitado para que se produzca una interrupción.

- Algunas fuentes de interrupción tienen asociado un *flag* que se activa cuando ocurre el evento, independientemente de que la interrupción estén activadas o no. Ese *flag* persiste mientras no lo borre el programa.

Interrupciones

- Cuando se produce una interrupción, automáticamente:
  - Se almacena en la pila el valor actual del contador de programa (PC)
  - Se desactiva el bit global de interrupción
    - Se deshabilitan las interrupciones anidadas
    - ¿Cómo se podrían habilitar?
  - Se transfiere el control al vector correspondiente (CALL)

- La rutina de atención a interrupción (ISR):
  - Trata el evento que la ha desencadenado
  - Termina con instrucción especial RETI
  - Automáticamente se activa el bit global de interrupción
  - Continúa la ejecución del programa por donde se había interrumpido

Interrupciones con C AVR

- Bit global de habilitación de interrupciones:
  - Se encuentra en el registro de estado SREG
  - Permite desactivar todas las interrupciones con un solo bit
  - `sei()` – activa el bit
  - `cli()` – desactiva el bit

- La prioridad de las interrupciones (ocurrencia simultánea) viene determinada por el orden de sus vectores
  - Direcciones más bajas tienen mayor prioridad

- `ISR(vector)` – Definición de rutina de interrupción
- `reti()` – Función llamada al final de la rutina de interrupción. Solo hay llamarla explícitamente en en ISR desnudas (`ISR_NAKED`)

Interrupciones externas
Arduino / C AVR

- **`attachInterrupt(`**`interrupt, function, mode`**`)`**

`interrupt`: **`digitalPinToInterrupt(pin)`**

Pines 2 y 3 en Arduino Uno

`function`: nombre de la función a ejecutar

`mode`: **`LOW, CHANGE, RISING, FALLING`**

- **`detachInterrupt(`**`interrupt`**`)`**

- **`interrupts()`** – Habilita interrupciones: **sei()**

- **`noInterrupts()`** – Deshabilita interrupcioned: **cli()**