UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

Introducción a los sistemas digitales. Micros

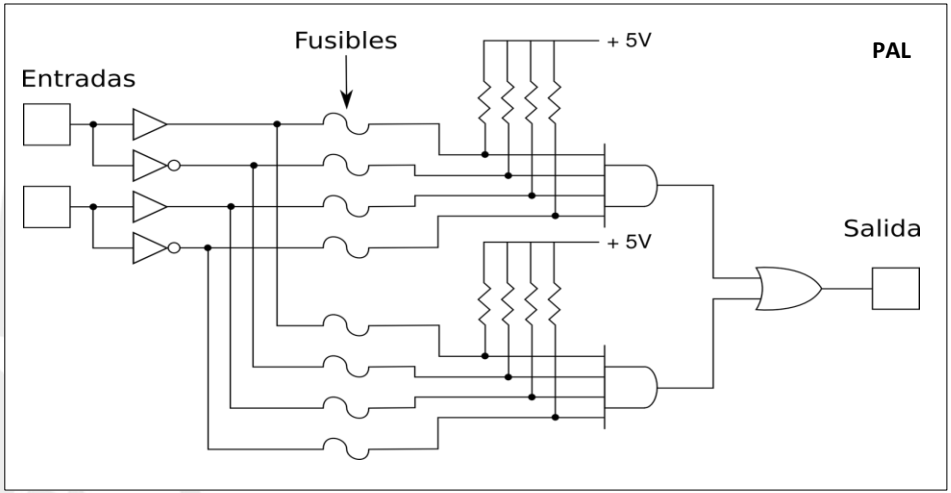
Luis Cucala García

Implantación de sistemas digitales

- Alternativas para implantar un sistema digital:
  - **Lógica discreta**: Mucho espacio, poco flexible, mayor consumo => No usada hoy en día.
  - **Lógica programable**: En un solo chip se puede integrar circuitos muy complejos => poco espacio, reprogramable y por tanto flexible, y menor consumo.
    - **Lógica cableada: CPLD** (Complex Programmable Logic Devices) y **FPGA** (Field Programmable Gate Array).
      - La "inteligencia" del sistema se define por el conexionado de sus componentes.
    - **Lógica programada: Microcontrolador**.
      - La "inteligencia" del sistema se define por el programa ejecutado por el procesador.

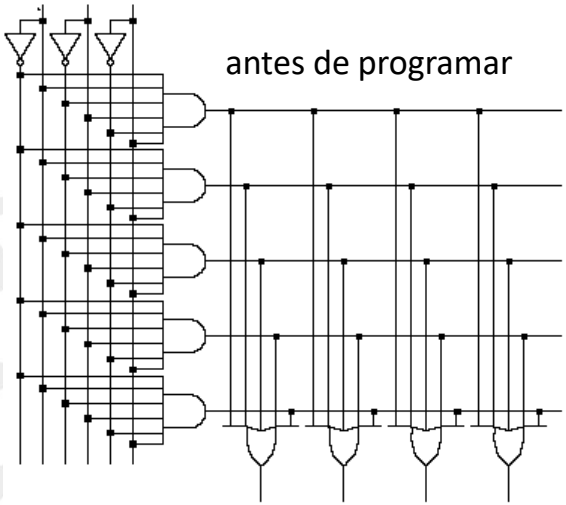
Lógica cableada

- PLD (programable logic device): chips de propósito general para implantar circuitos lógicos.

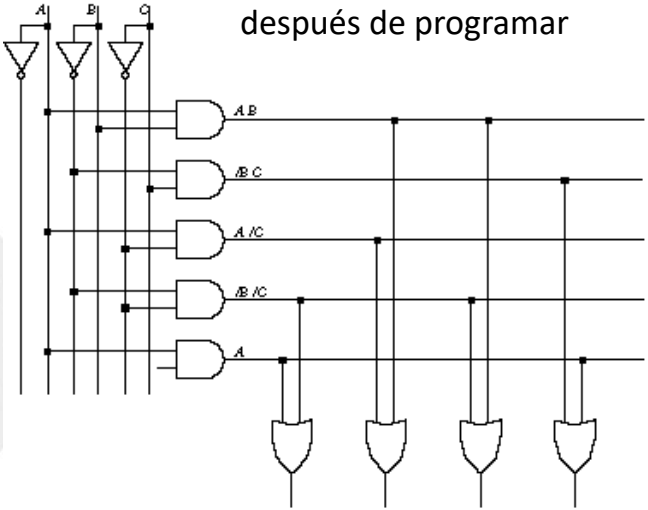


  - PAL (programable array logic): se programan las entradas a las AND.
  - PLA (programable logic array): se programan las entradas a las AND y a las OR
  - CPLD $\approx \Sigma$ PLD + biestables, no volatil
  - FPGA $\approx$ bloques complejos (sumadores, multiplicadores, etc) con flip-flops, volatil

Lógica programable con PLA

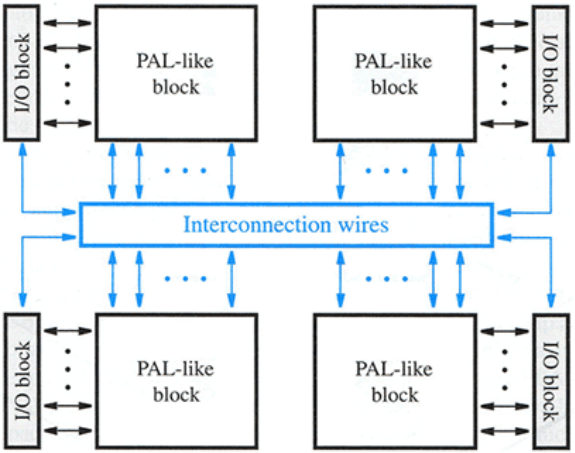


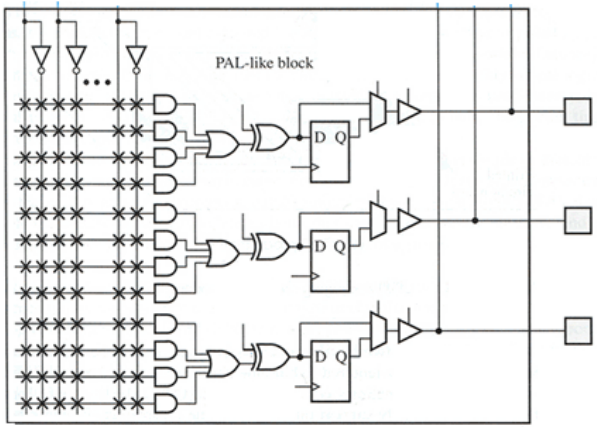
antes de programar



después de programar

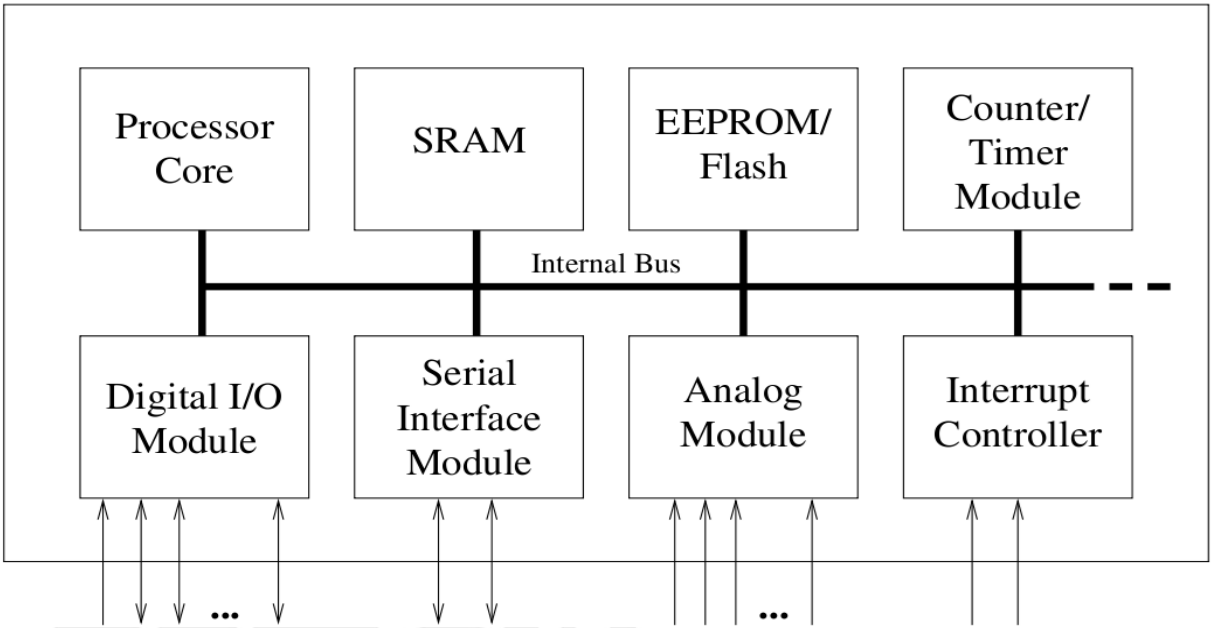
Lógica programada con CPLD





Estructura de un microcontrolador

- Microcontrolador: CPU + memoria + periféricos

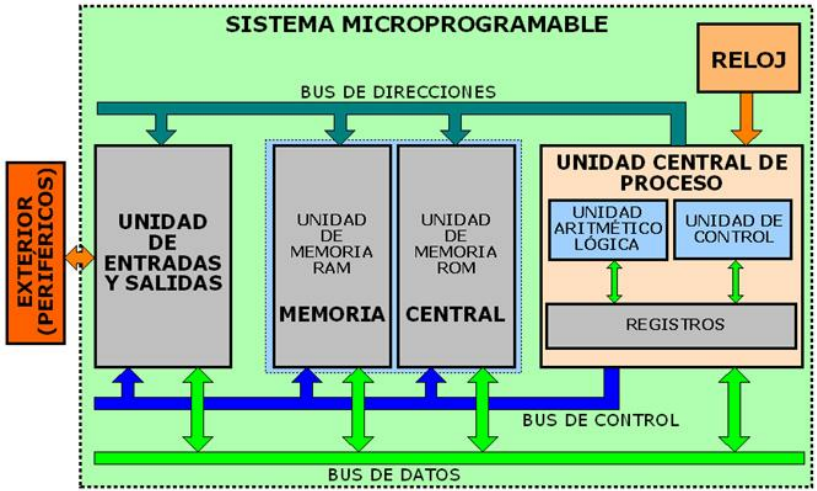


Arquitectura de un sistema basado en CPU

Bus de direcciones (address bus): llevan los bits para seleccionar la posición de memoria o el registro de entrada/salida en el que leer o escribir. Tiene tantas líneas como bits de dirección tenga el sistema, número que determina la cantidad máxima de memoria que puede direccionar el sistema=$2^{\text{Líneas de direcciones}}$

Bus de datos (data bus): bits que componen la información binaria, instrucciones o datos, contenidos en la posición de memoria o en los registros de entrada/salida, seleccionados por el bus de direcciones.

Bus de control (Control Bus): líneas de control, por las que va a circular el conjunto de señales necesarias para la correcta coordinación de todos los elementos del sistema, tales como: órdenes de lectura o escritura, inhabilitación (desactivación) de un dispositivo, etc.

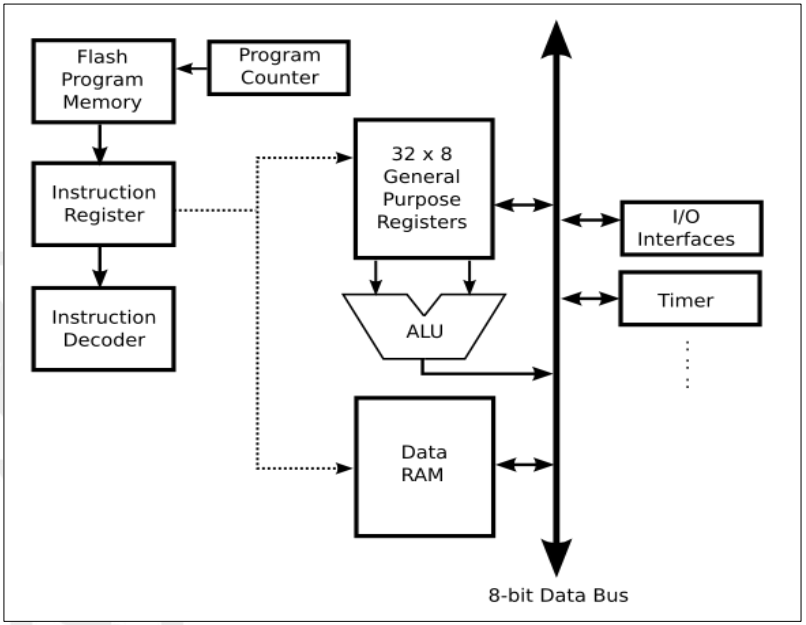


(http://perso.wanadoo.es/pictob/microprg.htm)

Funcionamiento:
Cuando un registro de entrada y salida recibe su dirección, el dispositivo se activa y según la orden de leer o escribir del bus de control, pasa la información que contienen al bus de datos, o se cargan con la información del bus de datos.

La CPU de un microcontrolador

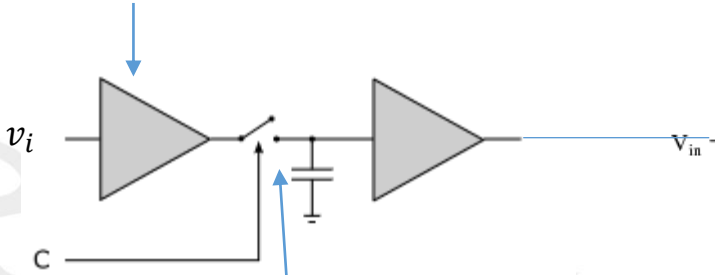
- Registros: valores temporales.
- ALU: circuito combinacional complejo para operaciones aritméticas, lógicas, desplazamientos, etc.
- RAM: datos
- Flash: Programa.
- Cada ciclo de reloj lee una instrucción, la decodifica y la ejecuta.



La CPU

ADC: proceso de muestreo y cuantización

Este circuito se llama “sample & hold” y es el que hace el muestreo

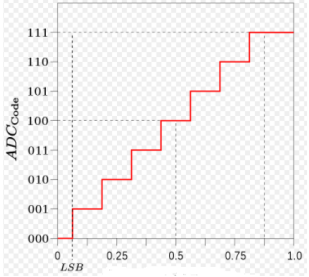


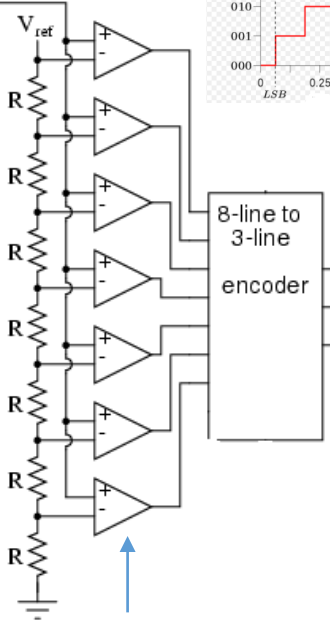
Esta es la señal muestreadora

Cuando se cierra el interruptor, se carga C con una tensión proporcional a la v_i

Y cuando se abre, el valor de la tensión se retiene en el C, de modo que no varía durante el proceso de cuantización

Esquema de codificación con 3 bits





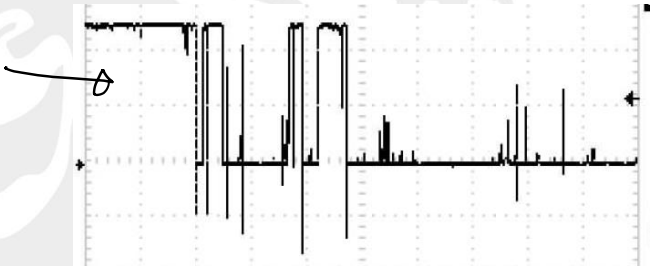
Salida digital con 3 bits por muestra

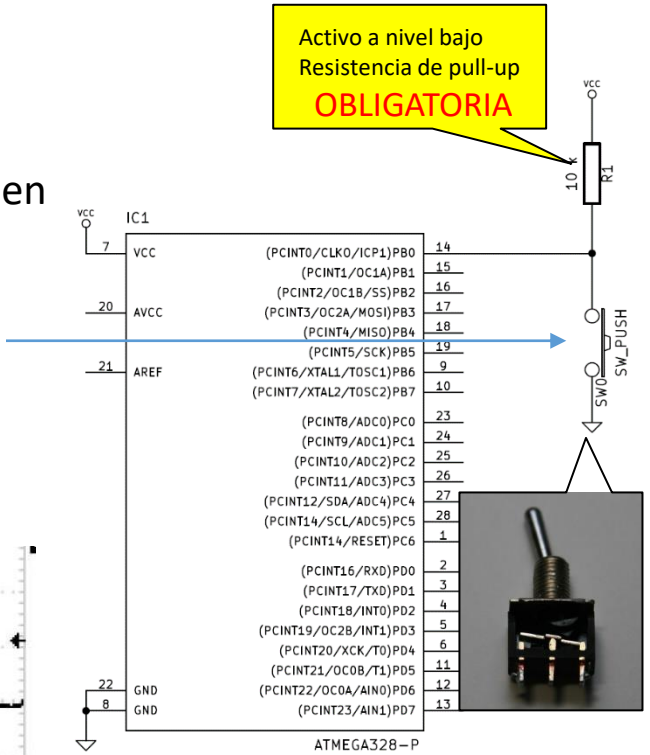
Esto es un ADC tipo Flash, compara la salida del sample & hold con 7 niveles de referencia, y codifica con 3 bits el esquema de conversión

Circuitos de entrada/salida

(o activo a nivel alto, con resistencia de pull-down)

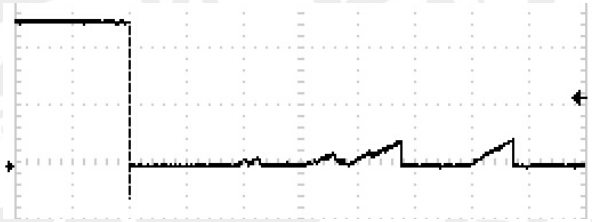
- Interruptores y pulsadores:
  - Permiten introducir un bit en un sistema digital.
  - En el circuito, entrada normalmente a nivel alto hasta pulsación.
  - Rebotes (mecánicos):

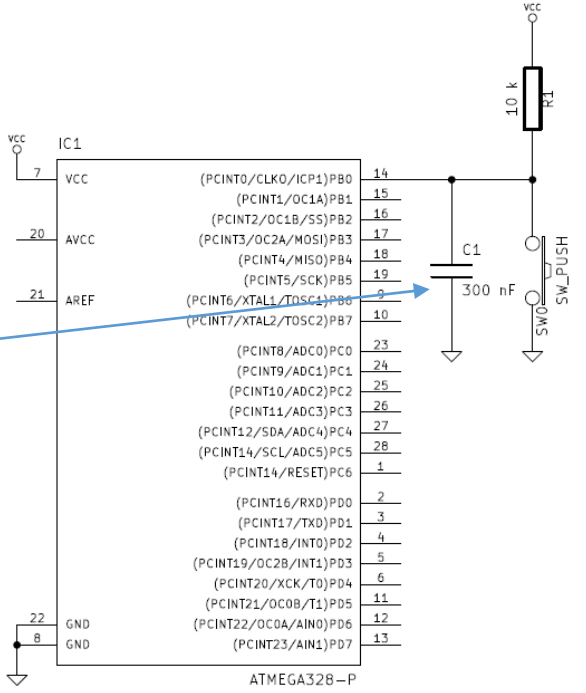




Circuitos de entrada/salida

- Modos de evitar los rebotes:
  - Por software: con un retardo no se vuelve a leer el estado hasta que terminen los rebotes.
  - Por hardware: con un RC





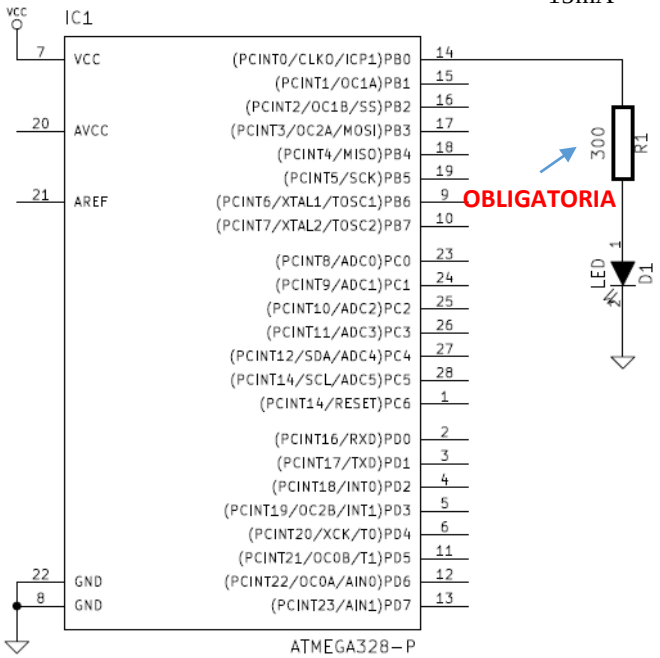
No siempre es obligatorio eliminar los rebotes

Circuitos de entrada/salida

En los diodos LED caen 1,4 V cuando están ON

- **Salida de un puerto a un LED:**
  - Cuando circula corriente directa (p.e. 15 mA), el LED se ilumina y caen **1,4 V** aprox.
  - Los LED se pueden conectar directamente al microcontrolador (el LED necesita 10-20 mA, y el micro puede entregar 40 mA).
  - Si PB0 = 1, el LED se enciende.
  - **R1 ES OBLIGATORIA**, limita la corriente por el LED a la nominal

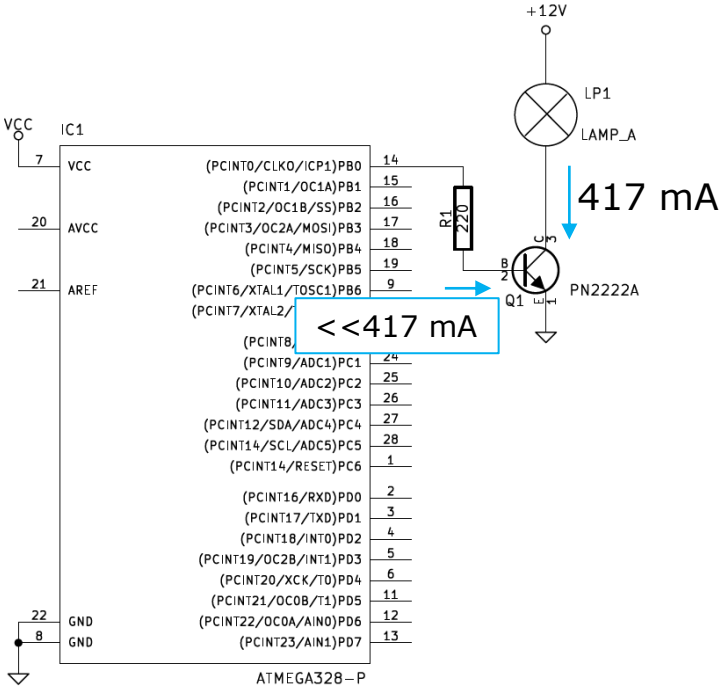
$$R1 \approx \frac{5V - 1{,}4V}{15mA}$$



Circuitos de entrada/salida

Si el micro encendiera directamente la lámpara → $I_{LAMP} = \frac{5W}{5V} = 1A > 40\,mA$

Y si enciende la lámpara por medio de un BJT → $I_{LAMP} = \frac{5W}{12V} = 417\,mA > 40\,mA$

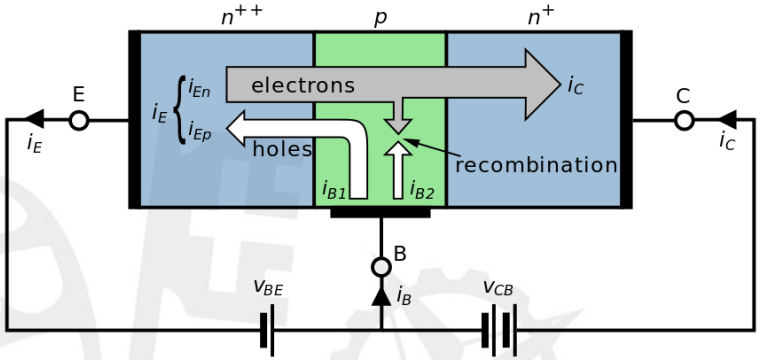


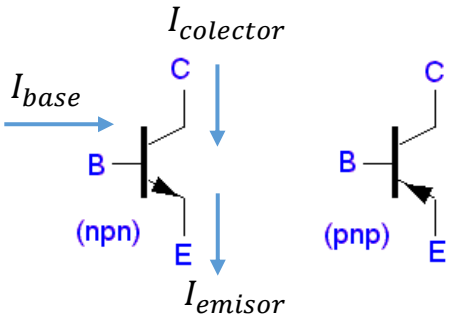
- La salida con BJT se utiliza para poder proporcionar corrientes mayores

  ($I_{max\text{-}puertos} \approx 40mA$ entrante o saliente)
- La salida por el puerto del Micro abre (corta el BJT) o cierra (satura el BJT) el interruptor (BJT) del circuito de la lámpara.
- En lo que sigue se revisa brevemente el funcionamiento de un transistor BJT

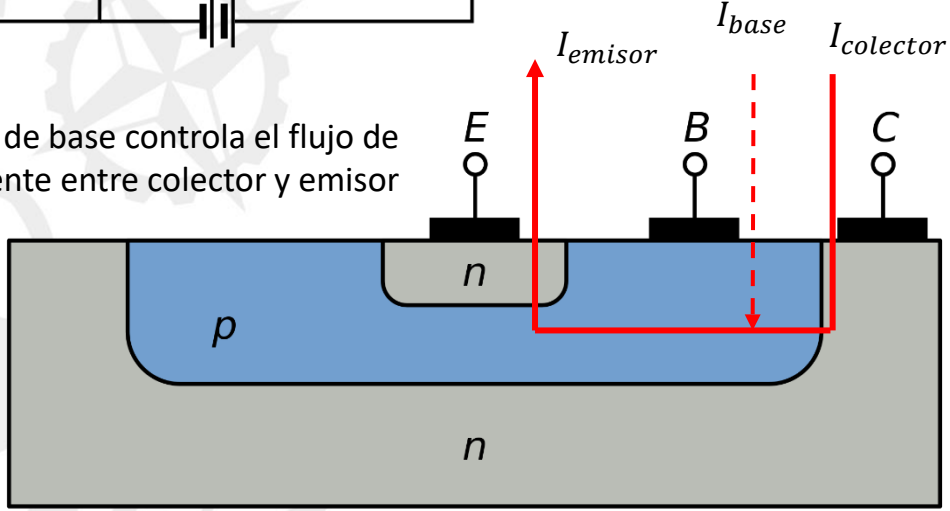
El transistor bipolar

BJT: Bipolar Junction Transistor



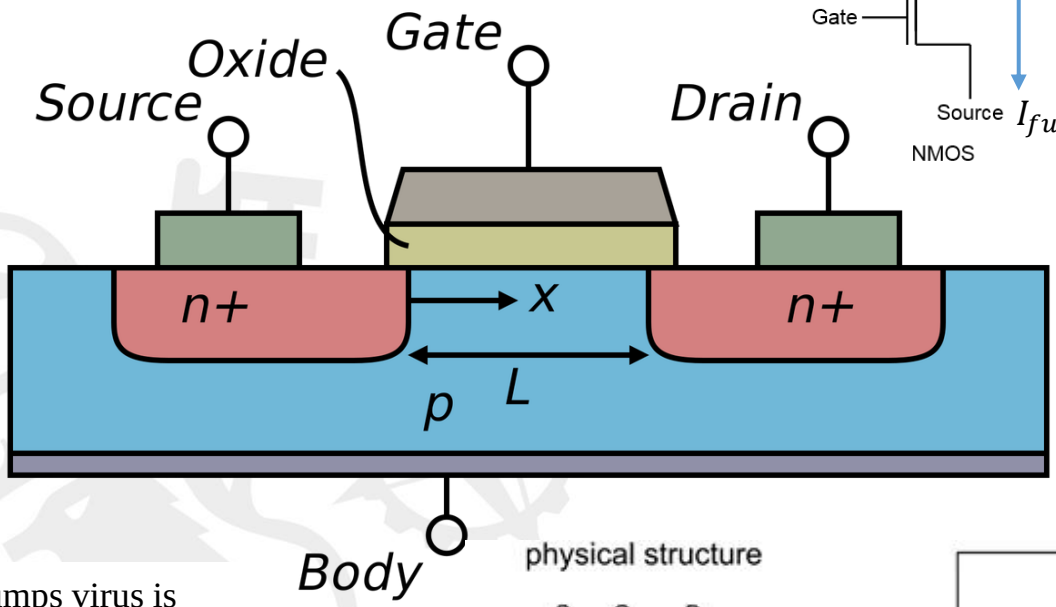


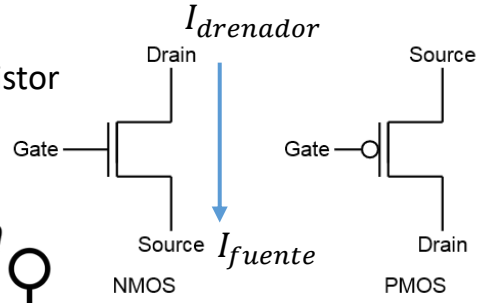
La corriente de base controla el flujo de corriente entre colector y emisor



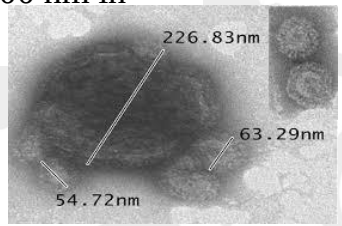
El transistor MOSFET

MOSFET: Metal Oxide Semiconductor Field Effect Transistor

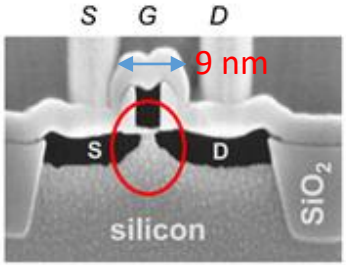




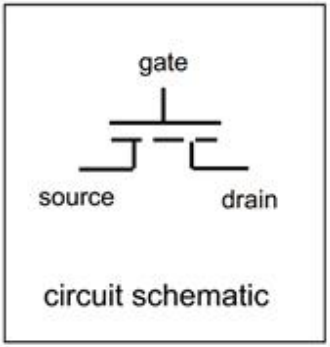
The mumps virus is primarily spherical shaped and roughly 200 nm in size...



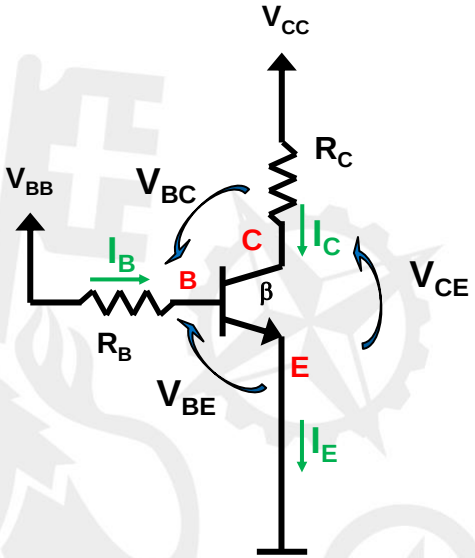
physical structure



(Texas Instruments, 1997)



Modos de funcionamiento del BJT NPN



siempre :

$$I_C + I_B = I_E$$

$$V_{BE} = V_{BC} + V_{CE}$$

(Esto son las reglas de Kirchoff)

Zona de corte del BJT NPN

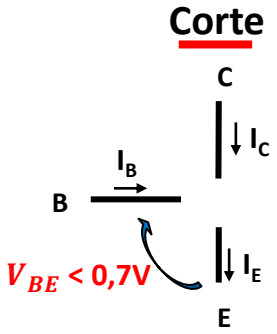
Funciona como un conmutador abierto

Corte:

$I_C = I_B = I_E = 0$

si se cumple

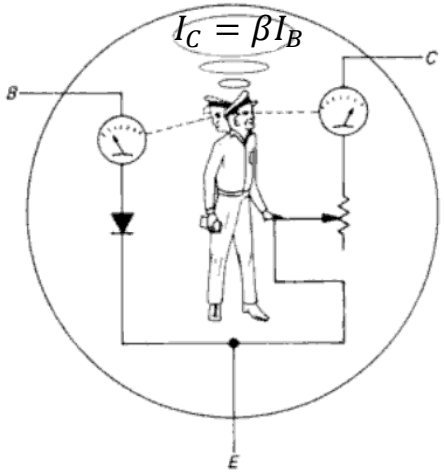
$V_{BE} < 0{,}7V$



$$se\ cumple\ que\ I_C = I_B = I_C = 0\ ,\ V_{BE} < 0{,}7\ V$$

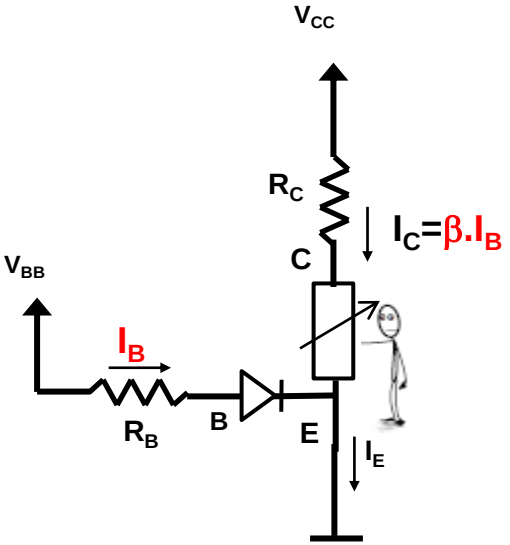
Zona activa del BJT NPN

$$se\ cumple\ que\ I_C = \beta I_B\ ,\ V_{BE} = 0{,}7\ V\ ,\ V_{CE} > 0{,}2\ V$$



("Transistor Man". The Art of Electronics. Horowitz)

Funciona como un amplificador de corriente



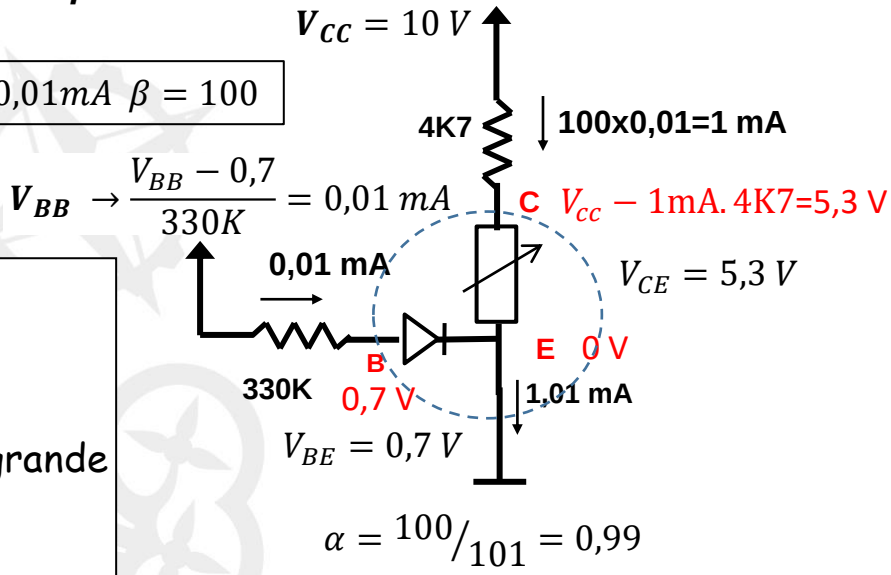
Zona activa del BJT NPN

se cumple que $I_C = \beta I_B$, $V_{BE} = 0{,}7\ V$, $V_{CE} > 0{,}2\ V$

$$I_C + I_B = I_E \rightarrow I_B(\beta + 1) = I_E \rightarrow I_B = I_E \frac{1}{\beta + 1}$$

$$I_C + I_E \frac{1}{\beta + 1} = I_E \rightarrow I_C = I_E \frac{\boldsymbol{\beta}}{\boldsymbol{\beta + 1}} = I_E \boldsymbol{\alpha}$$

Ejemplo para $I_B = 0{,}01mA$ $\beta = 100$



$$\alpha = {}^{100}/_{101} = 0{,}99$$

Activo :

$V_{BE} = 0{,}7V$,

$I_B = I_E/(\beta+1) = I_C/\beta$

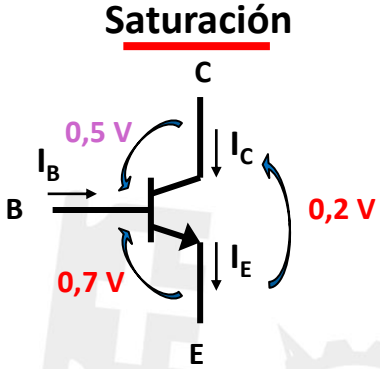
y si $\alpha = \beta/(\beta+1)$, β grande

$I_C = \alpha \cdot I_E \approx I_E$

si se cumple

$I_C, I_B, I_E > 0$, $V_{CE} > 0{,}2V$

Zona de saturación del BJT NPN



Funciona como un conmutador cerrado, con caída de tención 0,2 V

Modelo

$V_{BE} = 0{,}7V$

$V_{CE} = 0{,}2V$

$I_C + I_B = I_E$

Verificar:

$I_C < \beta I_B$

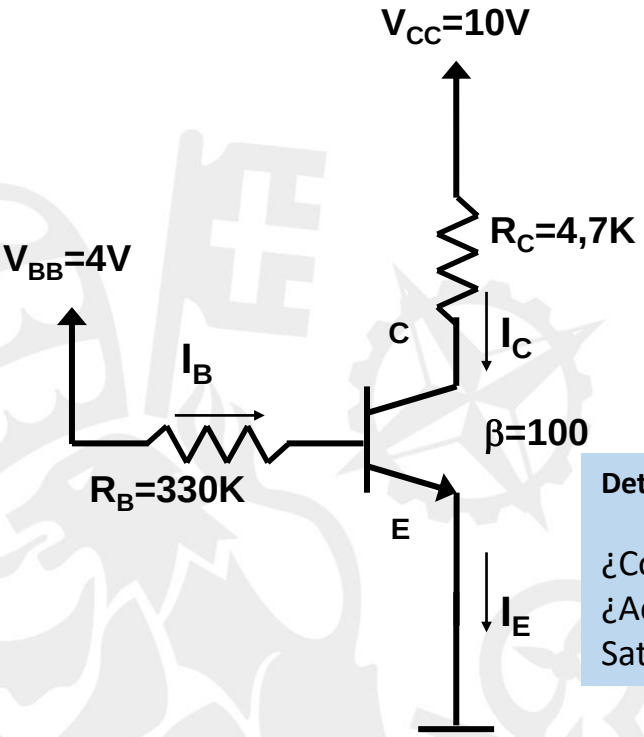
$I_C, I_B, I_E > 0$

$se\ cumple\ que\ I_C < \beta I_B\ ,$
$V_{BE} = 0{,}7\ V\ , V_{CE} = 0{,}2\ V$

$NO\ se\ cumple\ que\ I_C = \beta I_B$

Típicamente para saturación se diseña tal que $\frac{I_C}{I_B} = \frac{\beta}{n^{\underline{o}}\ (\text{factor de seguridad})}$

BJT: ejemplo 1



Determinar las tensiones y corrientes del circuito.

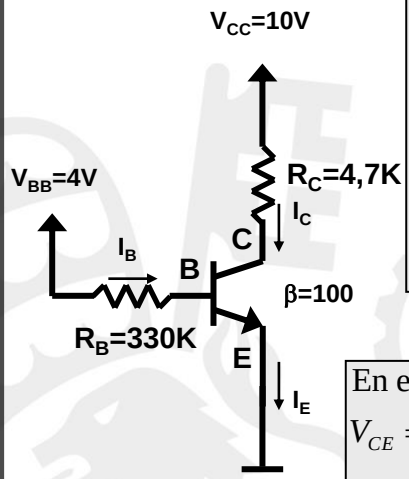
¿Corte? Ver si $V_{BE} < 0,7V$
¿Activo? Ver si $V_{BE} = 0,7V$, $V_{CE} > 0,2V$
Saturación? Ver si $V_{BE} = 0,7V$, $V_{CE} = 0,2V$

Qué ocurre si V_{BB} ↑ o si V_{BB} ↓

Calcular V_{BB} o R_B para Q SAT.

BJT: ejemplo 1

¿Corte? Ver si $V_{BE} < 0,7V$
¿Activo? Ver si $V_{BE} = 0,7V$, $V_{CE} > 0,2V$
Saturación? Ver si $V_{BE} = 0,7V$, $V_{CE} = 0,2V$



Suponemos Q Activo $(\text{si } I_i > 0 \; y \; V_{CE} > 0.2V)$

$$4V = I_B R_B + V_{BEon} \Rightarrow I_B = \frac{4V - 0,7V}{330k} = 0,01mA$$

si activo entonces $I_C = \beta I_B = 1\ mA$

¡ACTIVO!

comprobamos que $I_C, I_B, I_E > 0 \;\; y \; V_{CE} > 0,2\ V$

$$I_C, I_E, I_B > 0 \qquad I_C = 100 \; x \; 0,01\ mA = 1\ mA$$

$$V_{CE} = V_C = 10\ V - 4,7\ K\Omega \; x \; 1\ mA = 5,3V > 0,2V$$

Qué ocurre si $V_{BB} \uparrow$ o si $V_{BB} \downarrow$.
Calcular V_{BB} o R_B para Q SAT.

En el umbral de SAT

$$V_{CE} = 10V - I_{C\_SAT} \times 4,7k = 0,2V$$

$$\Rightarrow I_{C\_SAT} = \frac{10V - 0,2V}{4,7k} = 2,09mA \Rightarrow I_{B\_SAT} = \frac{I_{C\_SAT}}{\beta} = \frac{2,09mA}{100} = 0,02mA$$

Con R_B cte si $V_{BB} \uparrow, I_B, I_C \uparrow, V_{CE} \downarrow$

Con V_{BB} cte si $R_B \downarrow, I_B, I_C \uparrow, V_{CE} \downarrow$

Nos acercamos a la saturación

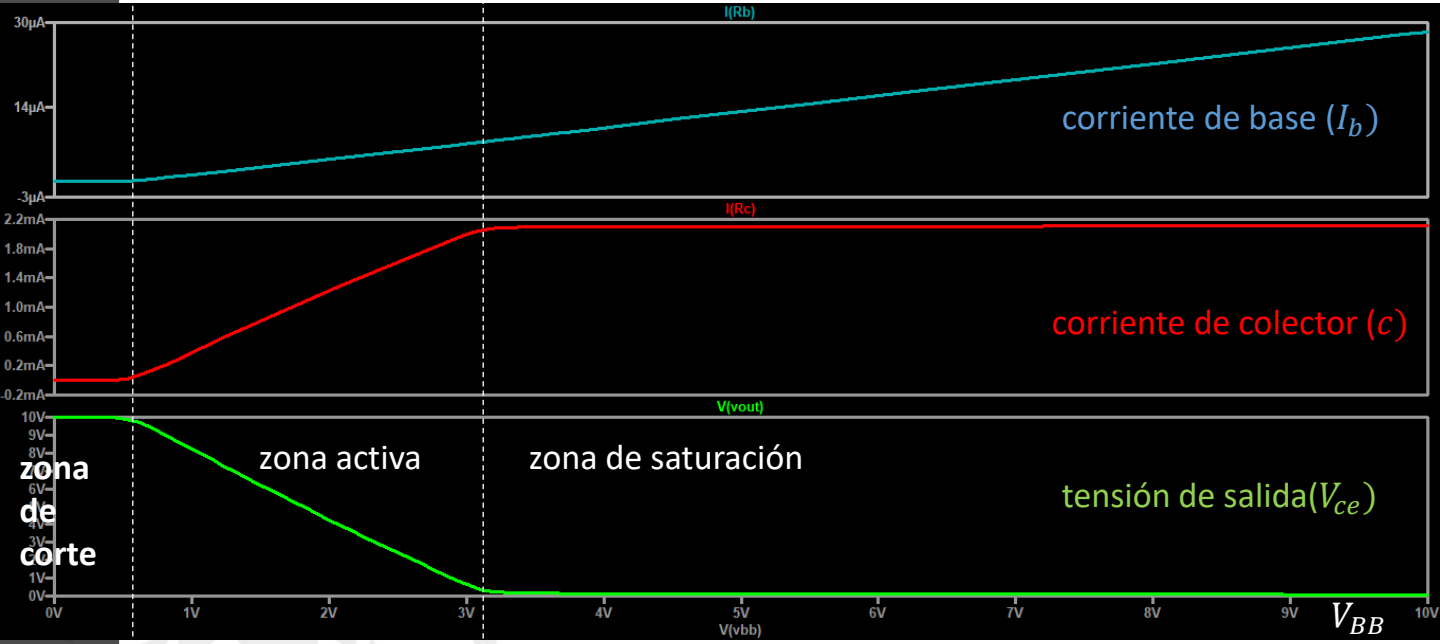
$\Rightarrow$

Para $R_B = 330K \Rightarrow \frac{V_{BB} - 0,7}{R_B} = 0,02 \Rightarrow V_{BB} \geq 7,3\ V\ para\ sat.$

o bien

Para $V_{BB} = 4V \Rightarrow \frac{V_{BB} - 0,7}{R_B} = 0,02 \Rightarrow R_B \leq 165K\ para\ sat.$

BJT: ejemplo 1



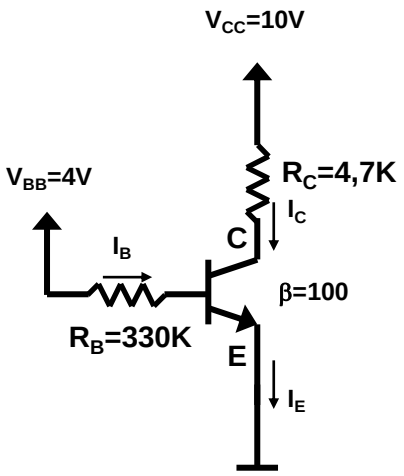
La diferencia en la $V_{BB}(saturación)$ con respecto a lo calculado, se debe a que el TRT elegido para la simulación tiene una β mayor

por eso para saturación diseñamos con un factor de seguridad

$$\frac{I_C}{I_B} = \frac{\beta}{nº\,(\text{factor de seguridad})}$$

para asegurarnos que estamos claramente en saturación

BJT: ejemplo 1



Cálculo de V_{BB} o R_B para estar en saturación, con factor de seguridad, p.e. 5

para estar en saturación

5 veces más que antes

$$I_{c\,sat} = \frac{10\ -0{,}2}{4K7} = 2{,}09\ mA \rightarrow \boldsymbol{I_B} = \frac{I_{c\,sat}}{\beta/_{fac.seg.}} = fac.seg.\frac{I_{c\,sat}}{\beta} = 5\frac{2{,}09\ mA}{100} = \mathbf{104{,}5\ \mu A}$$

muchos voltios...

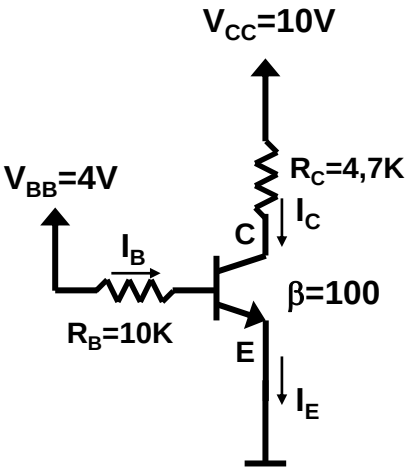
$$I_B = 104{,}5\ \mu A = \frac{V_{BB} - 0{,}7}{R_B}$$

para $R_B = 330K \rightarrow 104{,}5\ \mu A = \frac{V_{BB}-0{,}7}{330K} \rightarrow V_{BB} = 35{,}2\ V$

para $V_{BB} = 4\ V \rightarrow 104{,}5\ \mu A = \frac{4-0{,}7}{R_B} \rightarrow R_B = 31{,}6\ K\Omega$

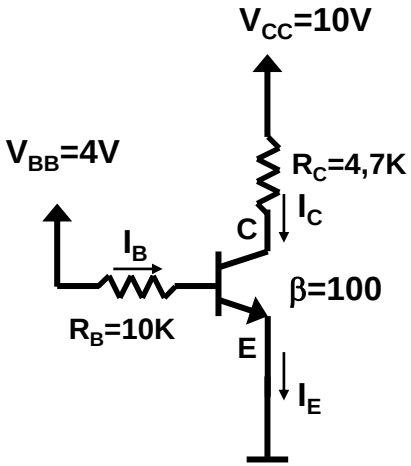
mejor esta opción

BJT: ejemplo 2



Determinar las tensiones y corrientes del circuito.

BJT: ejemplo 2



Suponemos Q Activo (si $I_E > 0$ y $V_{CE} > 0.2V$)

$$I_B = \frac{4V - 0{,}7V}{10k} = 0{,}33mA$$

si activo entonces $I_C = \beta \cdot I_B = 33mA$

comprobamos que

$$I_C, I_E, I_B > 0$$

$$V_{CE} = V_C = 10V - 4{,}7k\Omega \times 33mA = -145V < 0{,}2V \Rightarrow \text{Q SAT}$$

si Q Sat

$$I_C = \frac{10V - 0{,}2V}{4{,}7K} = 2{,}09mA < \beta \cdot I_B$$

$$I_E = I_B + I_C = 2{,}42mA$$

$$\Rightarrow \frac{I_C}{I_B} = \frac{2{,}09}{2{,}42 - 2{,}09} \approx 6 << \beta$$

Si se quiere forzar Q SAT se diseña con I_C/I_B del orden de 10 o 20.
Así se garantiza que $I_B >> I_C/\beta$
Es lo que se hace en electrónica digital cuando se utiliza un BJT como interruptor

Circuitos de entrada/salida

Este esquema se usa cuando la salida del micro no puede suministrar bastante corriente a la carga

- **Salida mediante transistor BJT**:
  - Si PB0=0 (0V) transistor en corte
    ⇒bombilla apagada
  - Si PB0=1 (5V) transistor saturado
    ⇒bombilla encendida
  - Intensidad en la bombilla

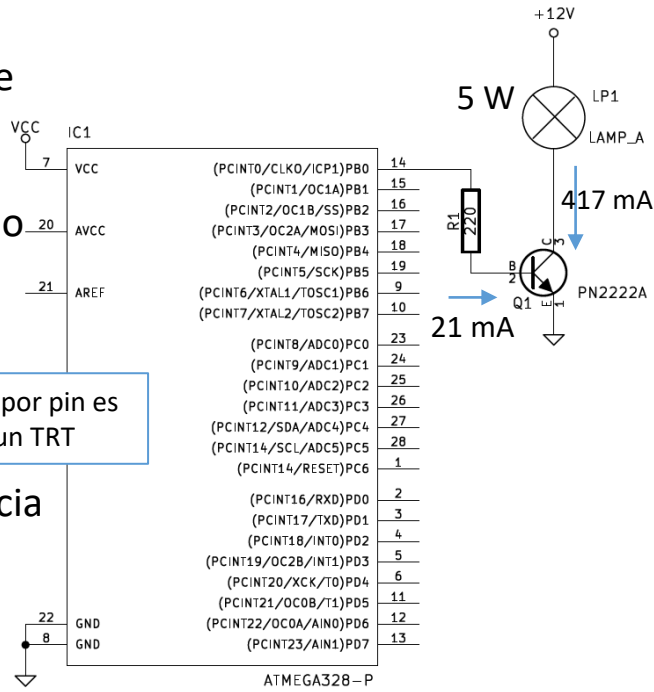
$$I_{LAMP} = \frac{5W}{12V} = 417mA$$

pero en ATMEGA328, la $I_{out\ max}$ por pin es 40 mA, por eso hay que usar un TRT

  - Intensidad en la base y resistencia en la base

$$I_B = \frac{I_C}{20} = 21mA \Rightarrow R1 = \frac{5V - 0{,}7V}{21mA} \approx 200\Omega$$

Valor normalizado $R1 = 220\Omega$

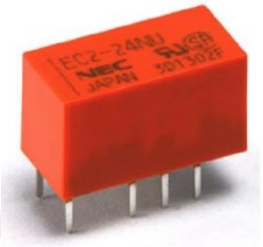


esta corriente ya la puede suministrar el micro

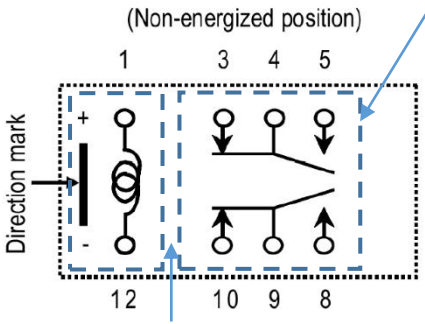
hemos empleado un factor de seguridad de 5 (100/5=20)

Circuitos de entrada/salida

- **Salida mediante Relé**:
  - El relé permite trabajar con tensiones alternas de 220 V y corrientes de varios amperios.
  - Proporciona aislamiento galvánico (el circuito que excita al relé está aislado eléctricamente del conmutador que es activado en el relé)
  - Inconvenientes:
    - Velocidad de conmutación baja
    - Número de conmutaciones limitado



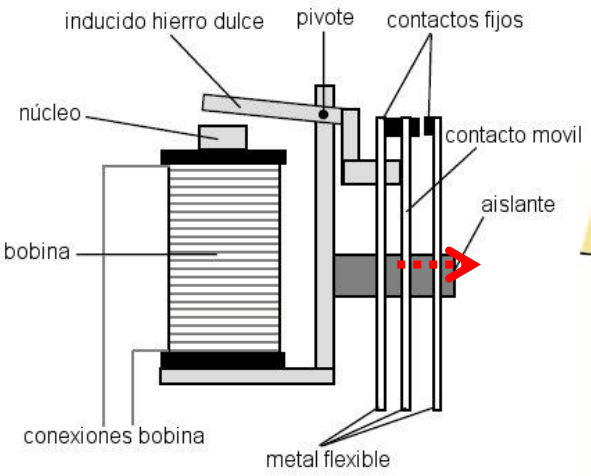
conmutadores internos que son activados por el relé

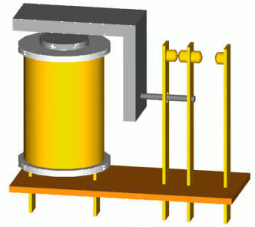


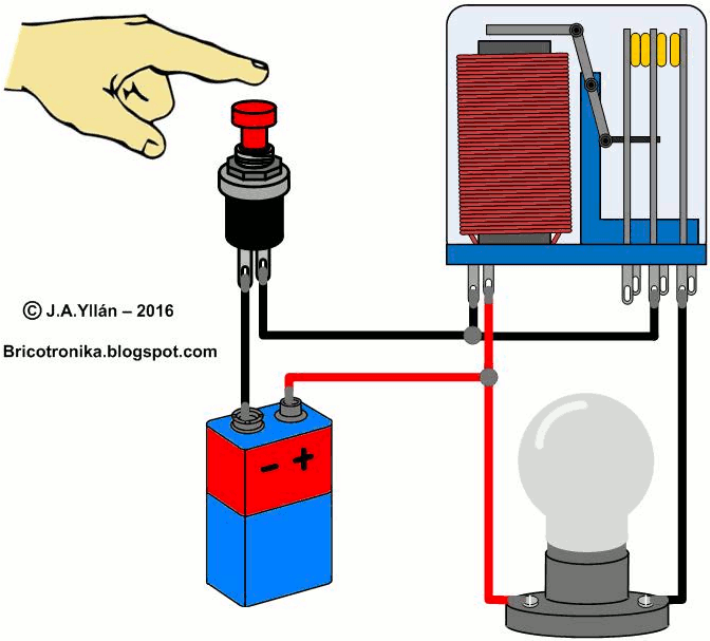
entre estas 2 zonas hay aislamiento eléctrico

Circuitos de entrada/salida

- **Salida mediante Relé:**







https://thumbs.gfycat.com/AcademicDelayedAxisdeer.webp

Circuitos de entrada/salida

cuando deja de pasar corriente por la bobina, $\frac{di}{dT} \rightarrow -\infty$ de modo que $V_{CE} \rightarrow \infty$

- **Salida mediante BJT y relé:**

Diodo de libre cirulación :

$v_{bobina} = L\frac{di}{dt}$

¡obligatorio ponerlo!

  - Si el relé tiene una resistencia de bobina de 178 Ω (interna, no se representa) y lo activamos con un BJT

$$I_C = \frac{5V - 0{,}2V}{178\Omega} = 27mA$$

  - Para asegurar la saturación

Si $\beta = 75$ escogemos $\frac{I_C}{I_B} = 15$ para Q SAT

$$I_B = \frac{I_C}{15} = 1{,}8mA$$

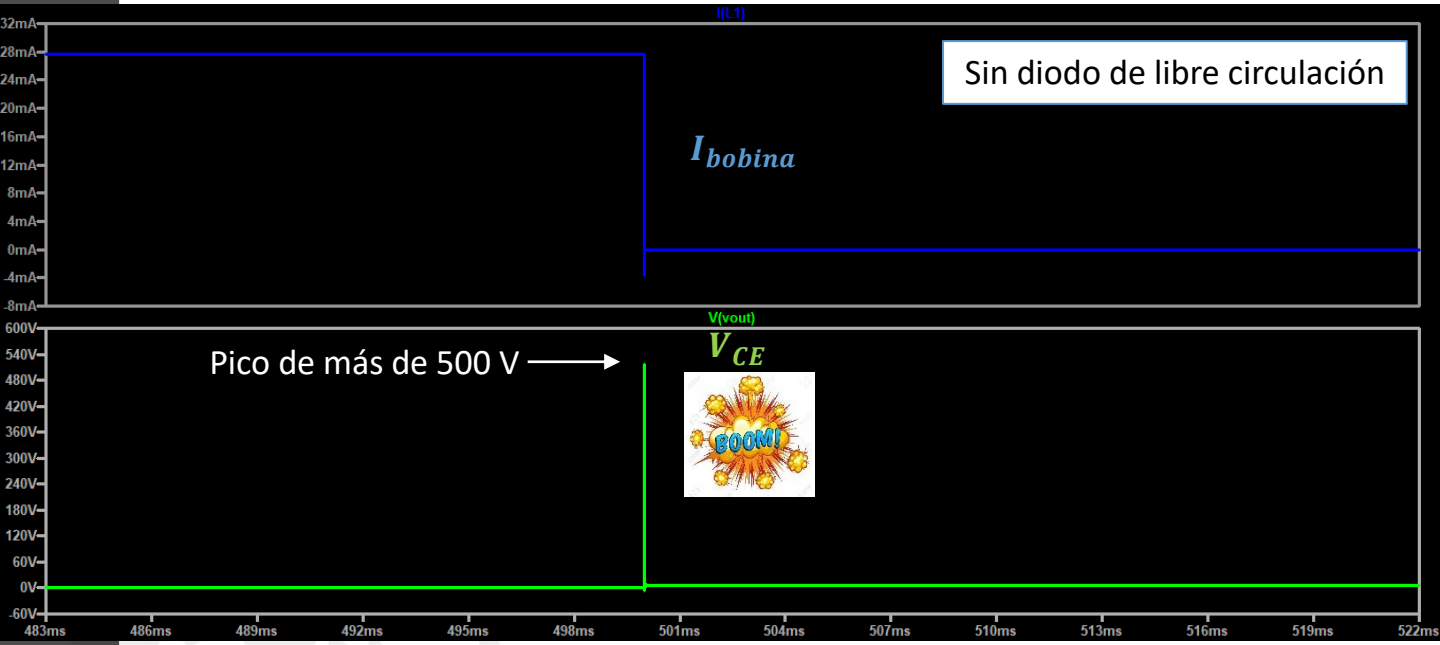
$$\Rightarrow R1 = \frac{5V - 0{,}7V}{1{,}8mA} = 2{,}4k\Omega$$

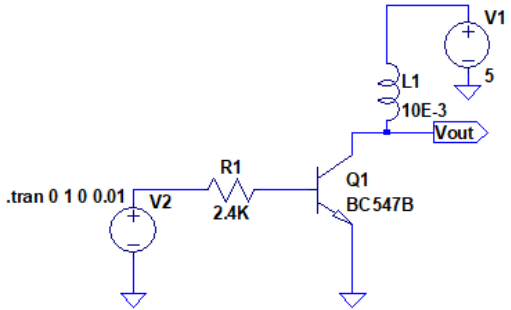
hemos empleado un factor de seguridad de 5 (75/5 = 15)

Cuando el relé está activado, el conmutador está a la izq. y **sí** pasa corriente

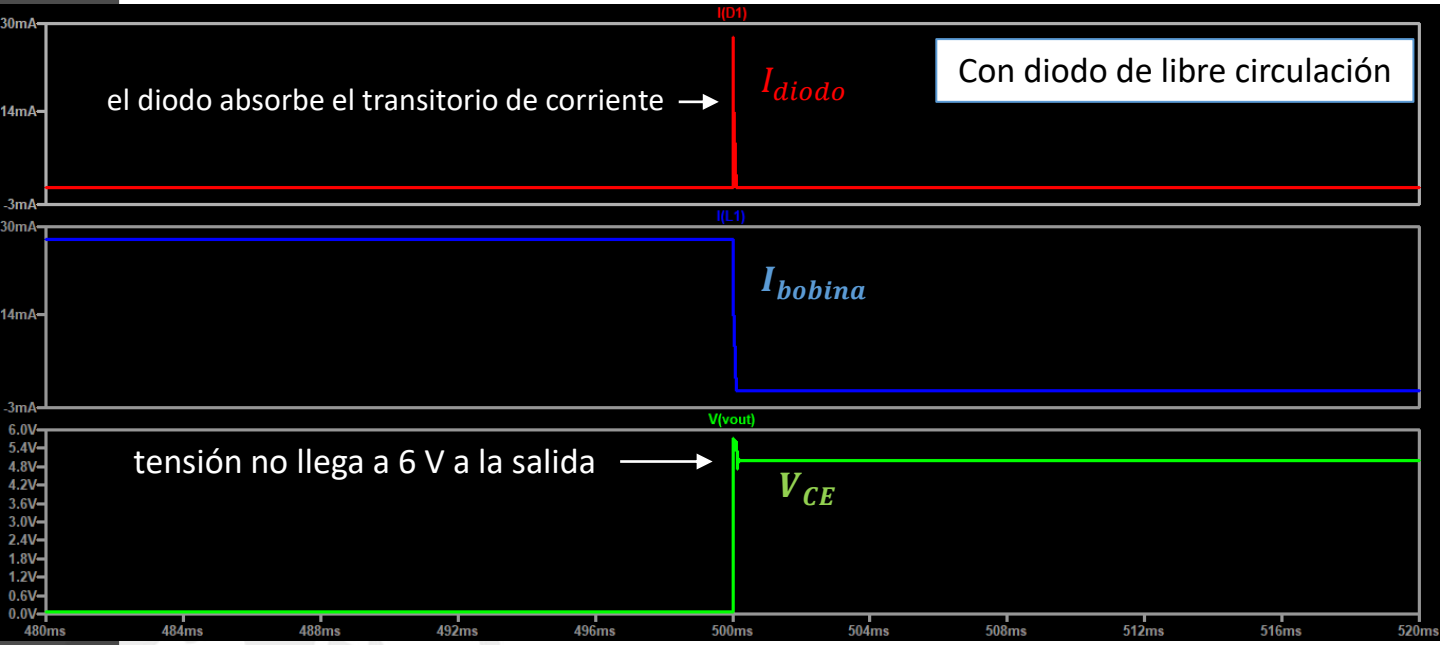
Cuando el relé no está activado, el conmutador está a la dcha. y **no** pasa corriente entre R y N

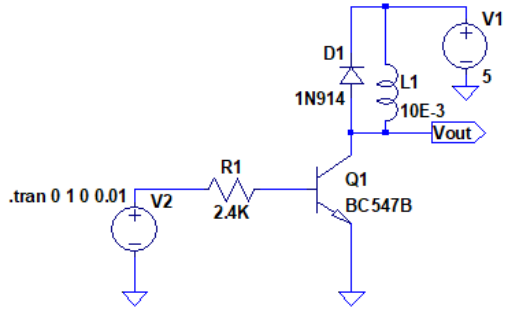
Circuitos de entrada/salida





Circuitos de entrada/salida





Hexadecimal

Dec	Hex	Bin
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

- Sistema de numeración de 16 dígitos (**base 16**):

  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

- Muy utilizado para representar la información que manejan los micros, que son palabras múltiplos del byte(=8bits)
- Hexadecimal son 4 bits=16 combinaciones.
- En lenguaje C: 0x indica que el número está en hex

```
0xF1        11110001
0xB5        10110101
```


Tipos de memoria

Volátiles: Se borran cuando se desconecta la alimentación eléctrica.

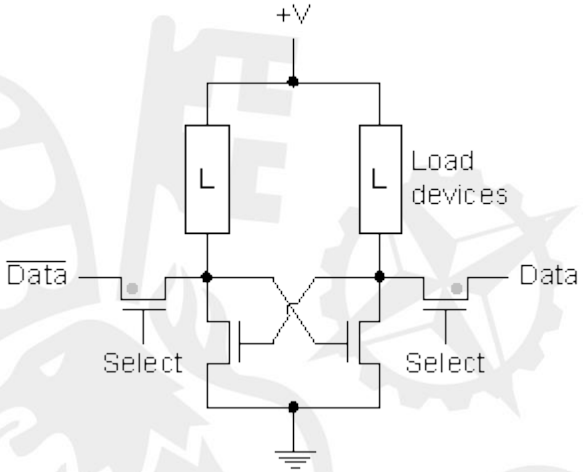
- DRAM: Dynamic RAM. Cada bit se almacena en un condensador, que se descarga con el tiempo, por lo que hay que reescribir el contenido periódicamente.
- SRAM: Static RAM. Cada bit se almacena en un par de transistores, por lo que no hace falta refrescar.

No volátiles: No se borran al desconectar alimentación.

- No reprogramables: son memorias que, una vez escritas, no se pueden volver a escribir
  - ROM: memoria que se escribe en fábrica.
  - PROM (Programmable ROM): posibilidad de escribirlas una vez después de la fabricación.
- Reprogramables: son memorias que se pueden escribir varias veces
  - EPROM: Erasable PROM (borrable con luz ultravioleta, antigua)
  - EEPROM: Electrically Erasable PROM (borrable eléctricamente)
  - Flash: EEPROM borrable por bloque de datos (estas son las memorias USB, las microSD, discos SSD)

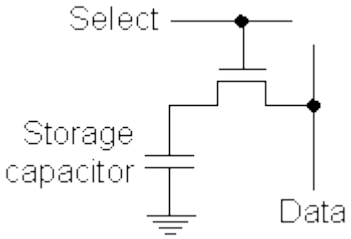
SRAM y DRAM

Static Random Access Memory



- No se borra mientras haya alimentación
- Muy rápida
- Consume energía
- Requiere área en el circuito (4 transistores)

Dynamic Random Access Memory



- Se borra al cabo de un tiempo aunque haya alimentación
- Es necesario reescribirla periódicamente
- Menos rápida
- Consume poca energía
- Requiere poca área en el circuito

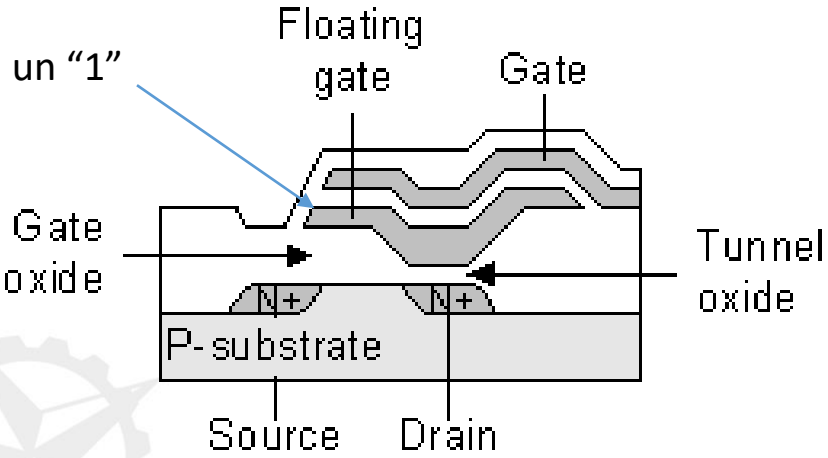
EPROM borrable por ultravioleta

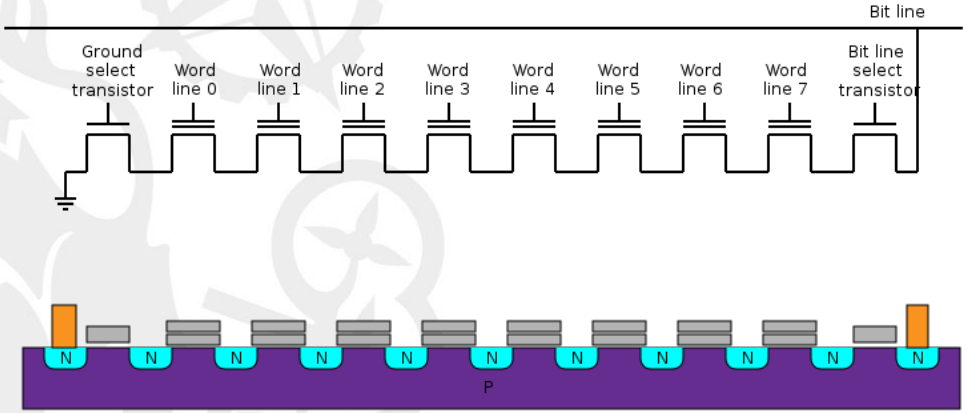
Esto solo es una curiosidad, ya no se usa:
Por esta ventana entra luz ultravioleta, que moviliza las cargas almacenadas en los condensadores donde se almacenan los datos, borrándolos

Memoria EEPROM (FLASH)

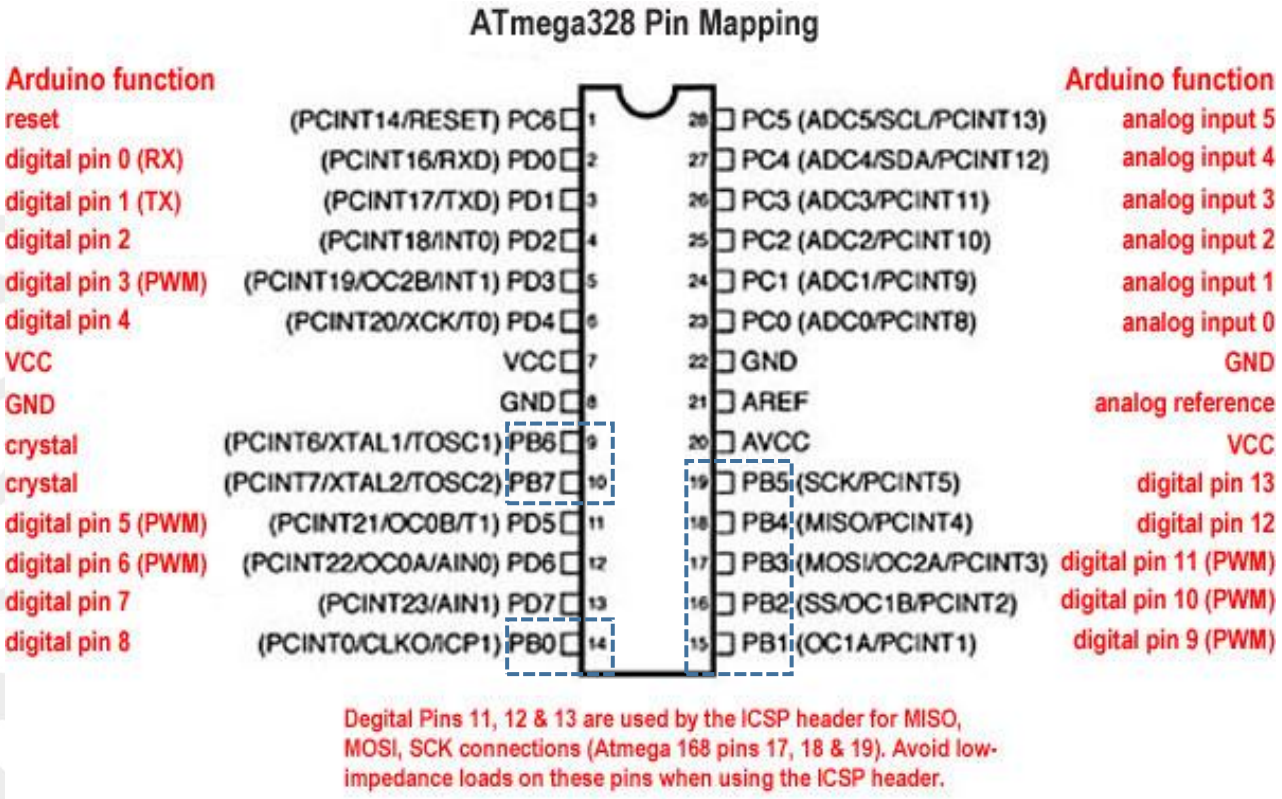
La carga que representa un “1” se almacena aquí

No se borra aunque se quite la alimentación





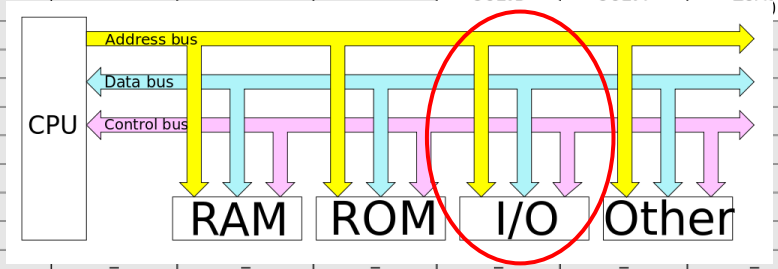
Patillaje del Arduino vs ATmega 328



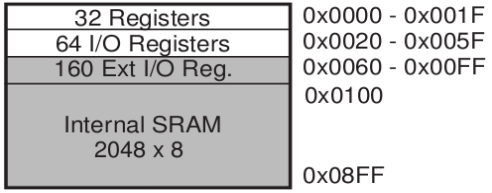
Programación de los puertos del microcontrolador

- Los puertos en el mapa de memoria (puertos B, C y D)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
(0x35)	TIFR0	–	–						
(0x34)	Reserved	–	–						
(0x33)	Reserved	–	–						
(0x32)	Reserved	–	–						
(0x31)	Reserved	–	–						
(0x30)	Reserved	–	–						
(0x2F)	Reserved	–	–						
(0x2E)	Reserved	–	–						
(0x2D)	Reserved	–	–						
(0x2C)	Reserved	–	–	–	–	–	–	–	–
(0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
(0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
(0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
(0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
(0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
(0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
(0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
(0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
(0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
(0x22)	Reserved	–	–	–	–	–	–	–	–
(0x21)	Reserved	–	–	–	–	–	–	–	–
(0x20)	Reserved	–	–	–	–	–	–	–	–

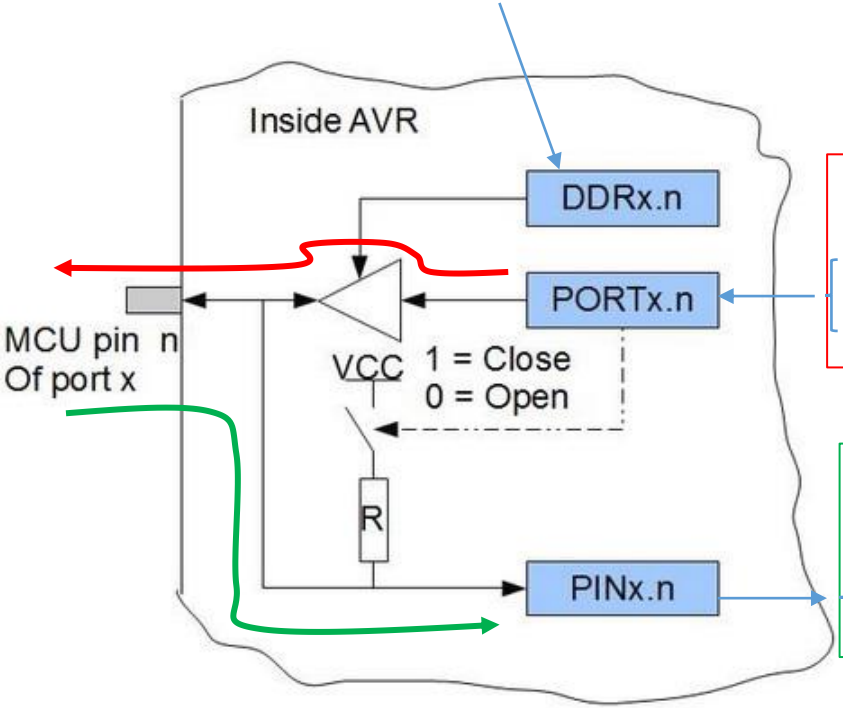


PORTB. Contiene el byte a escribir en los pines de salida
DDRB. Indica si el puerto es de entrada o de salida
PINB. Registra el byte leído desde los pines de entrada



Configuración de los puertos de microcontrolador

gobierna si los pines son de salida o entrada



Si declaro el puerto B como **salida** (**DDRB=1**), el byte almacenado en PORTB se presentará en los pines (en forma de 0 o 5V)

Si declaro el puerto B como **entrada** (**DDRB=0**), la señal presente en los pines (0 o 5V) se registrará en PINB (como "0" o "1" lógicos)

NOTA. Normalmente trabajaremos con cada uno de los 8 pines del puerto por separado

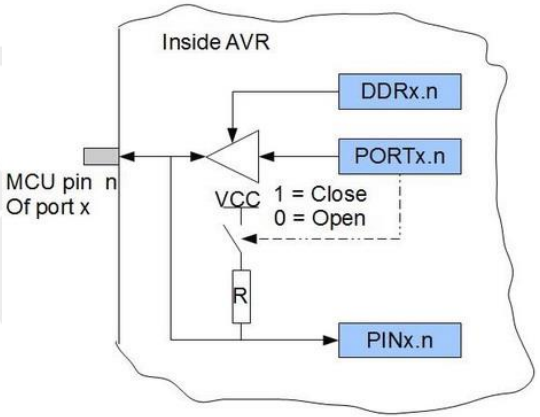
Configuración de los puertos de microcontrolador

Debemos **configurar los pines no usados como entradas**, para evitar cortocircuitos accidentales si se toca una patilla sin usar que esté programada como salida y por tanto con tensión, ya sea 0 o 5V.

Al configurar una patilla como entrada, si se queda sin conexión externa, por las características CMOS del chip, con alta impedancia de entrada, es fácil que el valor de entrada se ponga a cambiar de forma indeterminada, debido al ruido externo.

Por ello **se recomienda (DEBE) poner una resistencia de pull-up**, que puede ser **externa**, **o** bien **interna** en aquellos micros que dispongan de ella **(ATmega la incluye) .**

Para el **Arduino** es suficiente **con programar el puerto como entrada y escribir un 1 en el bit correspondiente de PORTB.**



DDRB = 0
PORTB = 1

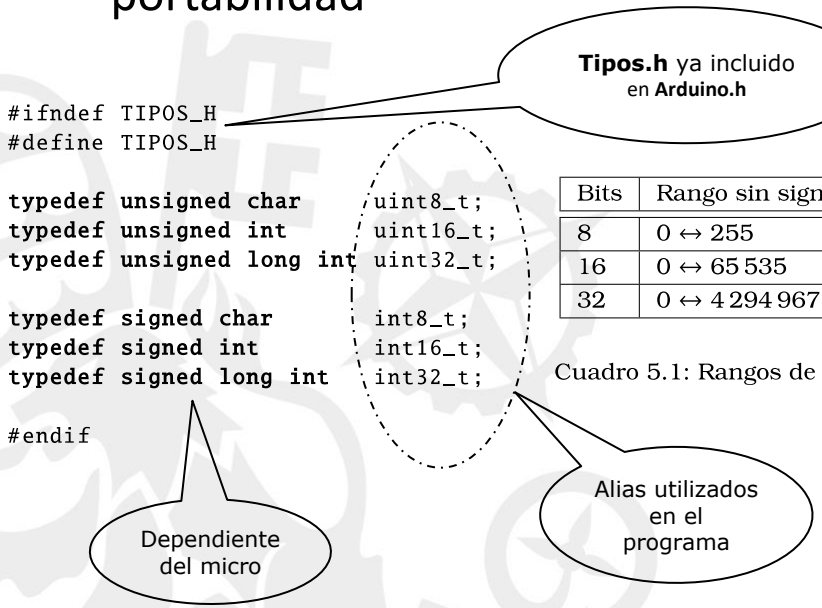
NOTA. Normalmente trabajaremos con cada uno de los 8 pines del puerto por separado

Comandos de control de los puertos del microcontrolador

- DDRx. Por ejemplo DDRB. **Controla si el puerto B es de entrada (0) o de salida (1)** (lo hacemos para cada pin individual del puerto)
  - Ejemplo. DDRB = 0x0F; // PB7-PB4 In, PB3-PB0 Out (0x0F es 0000 1111)
- PORTx. Por ejemplo PORTB. **Controla la salida del puerto (L, H, pull up)**
  - Si el puerto es de salida. Controla el estado **H ("1", 5V)** o **L ("0", 0V)** del puerto
  - Ejemplo. PORTB = 0x03; // PB7-PB2 L PB1 y PB0 H (0x03 es 0000 0011)
  - Si el puerto es de entrada. Controla la **conexión (1) o desconexión (0)** de la resistencia de pullup
  - Ejemplo.
    - DDRB = 0x08; /*0000 1000, PB7-PB4 no usadas y declaradas In, PB3 Out, PB2-PB0 sí usadas y declaradas In */
    - PORTB = 0xF0; /*1111 0000 Activamos las resistencias de Pull-Up de las entradas no usadas */
- PINx. Por ejemplo PINB. **Contiene el byte leído por los pines de entrada**
  - Ejemplo: unaVariable = PINB; /*leerá los pines que son de entrada en el puerto B y los asigna a "unaVariable" */
  - Ejemplo: printf("El valor obtenido del puerto B es: %x\n", PINB);

Lenguaje C para micros: tipos de datos

- Tipos de datos enteros: definirlos en header file para portabilidad



Bits	Rango sin signo	Rango con signo
8	$0 \leftrightarrow 255$	$-128 \leftrightarrow 127$
16	$0 \leftrightarrow 65\,535$	$-32\,768 \leftrightarrow 32\,767$
32	$0 \leftrightarrow 4\,294\,967\,296$	$-2\,147\,483\,648 \leftrightarrow 2\,147\,483\,647$

Cuadro 5.1: Rangos de enteros con y sin signo de 8, 16 y 32 bits.

Manipulación de bits. Operadores a nivel de bit

- Desplazamiento. A la izquierda (<<) y a la derecha (>>)
  - Ejemplo. Si a = 10101001 , b = a << 4; (desplaza los bits de a 4 posiciones a la izq.) entonces b = 10010000 (lo que se va por la izq. se pierde, por la dcha. entran ceros)
- Operadores lógicos bit a bit de dos operandos
  - AND. &
    - Ejemplo. a = 01101101 b = 10101001 entonces a & b = 00101001
  - OR. |
    - Ejemplo. a = 01101101 b = 10101001 entonces a | b = 11101101
  - XOR. ^
    - Ejemplo. a = 01101101 b = 10101001 entonces a ^ b = 11000100
  - Operador NOT. ~
    - Ejemplo. Si a = 0xF0 (11110000), ~a valdrá 0x0F (00001111)

Manipulación de bits. Bits individuales y uso de máscaras

Notación a usar en un byte → B7 (bit 7, más significativo), B6,…, B1, B0 (bit 0, menos significativo)

- Verificar el estado de un bit de una variable.
  - Se hace AND entre la variable y una máscara con todos los bits a cero excepto el bit que se quiere comprobar
  - Ejemplo. if (PINB & (1<<5)) /* pasamos de 0000 0001 a 0010 0000, y comprobamos si el bit 5 de PINB es 1, pues XXXX XXXX AND 0010 0000 = 00X0 0000 */
- Poner un bit a 1
  - Se hace OR entre la variable y una máscara con todos los bits a cero, excepto el bit que se quiere poner a 1.
  - Ejemplo. PORTB |= (1<<2); /* pasamos de 00000001 a 00000100, y ponemos bit 2 de PORTB a 1, pues XXXX XXXX OR 0000 0100 = XXXX X1XX */
- Poner un bit a 0
  - Se construye una máscara con todos los bits a 1 excepto el bit n. Para ello, primero se crea una máscara con todos los bits a cero excepto el bit n y luego se invierte la máscara con el operador NOT
  - Luego se hace AND entre la variable y la máscara
  - Ejemplo. PORTB &= ~(1<<2); /* pasamos de 0000 0001 a 0000 0100, invertimos y queda a 1111 1011, y ponemos el bit 2 de PORTB a 0, pues XXXX XXXX AND 1111 1011 = XXXX X0XX */
- Invertir un bit
  - Se hace una XOR con una máscara igual a la usada para ponerlo a 1
  - Ejemplo. PORTB ^= (1<<2); /*pasamos de 0000 0001 a 0000 0100, y bit 2 se invierte, pues XXXX XXXX XOR 0000 0100 = XXXXX $X\bar{X}$XX */

Uso de máscaras para mejorar la legibilidad

- Pueden definirse máscaras para acceder a bits individuales, y darles a estas máscaras los nombres de los bits
- Ejemplo #define LED_D1 (1<<1)
  - Pasamos de 00000001 a 00000010
  - De esta forma tenemos una máscara así, 0000 0010, que nos ayudará a indicar que un LED está en el pin PB1 del PORTB
  - Para poner a 1 ese pin, se hace un OR con esa máscara.

  PORTB |= LED_D1 (PORTB = XXXX XXXX OR 0000 0010 = XXXX XX1X)
  - Para poner a 0 el pin, se hace una AND con su máscara negada.

  PORTB &= ~LED_D1 (PORTB = XXXX XXXX AND 1111 1101 = XXXX XX0X)

Programación de micros. El bucle scan

```
main()
{
  /* Inicialización del sistema */
  while(1){   /* Bucle infinito */
    /* Leer las entradas */
    /* Ejecutar algoritmo de control */
    /* Actualizar las salidas */
```

- En IDE Arduino, el bucle scan (función main) ya está predefinido en la función loop()

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

}

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

}
```

C para micros: el bucle principal

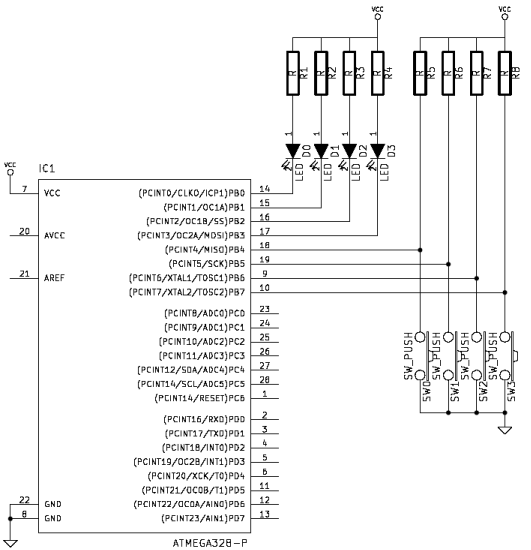
- Técnicas de programación:
  - El bucle scan: ejemplo de pulsadores y LED

```
#include "Arduino.h"

main()
{
  uint8_t entradas, salidas;

  DDRB = 0x0F;            /* PB7-PB4 In, PB3-PB0 Out */
  while(1){                          /* Bucle infinito */
    entradas = PINB;          /* Lee los pulsadores */
    salidas = (entradas >> 4) & 0x0F; /* Los aisla */
    PORTB = salidas;           /* Actualiza salidas */
  }
}
```

includes

Hex, conf. puerto

Definimos variables

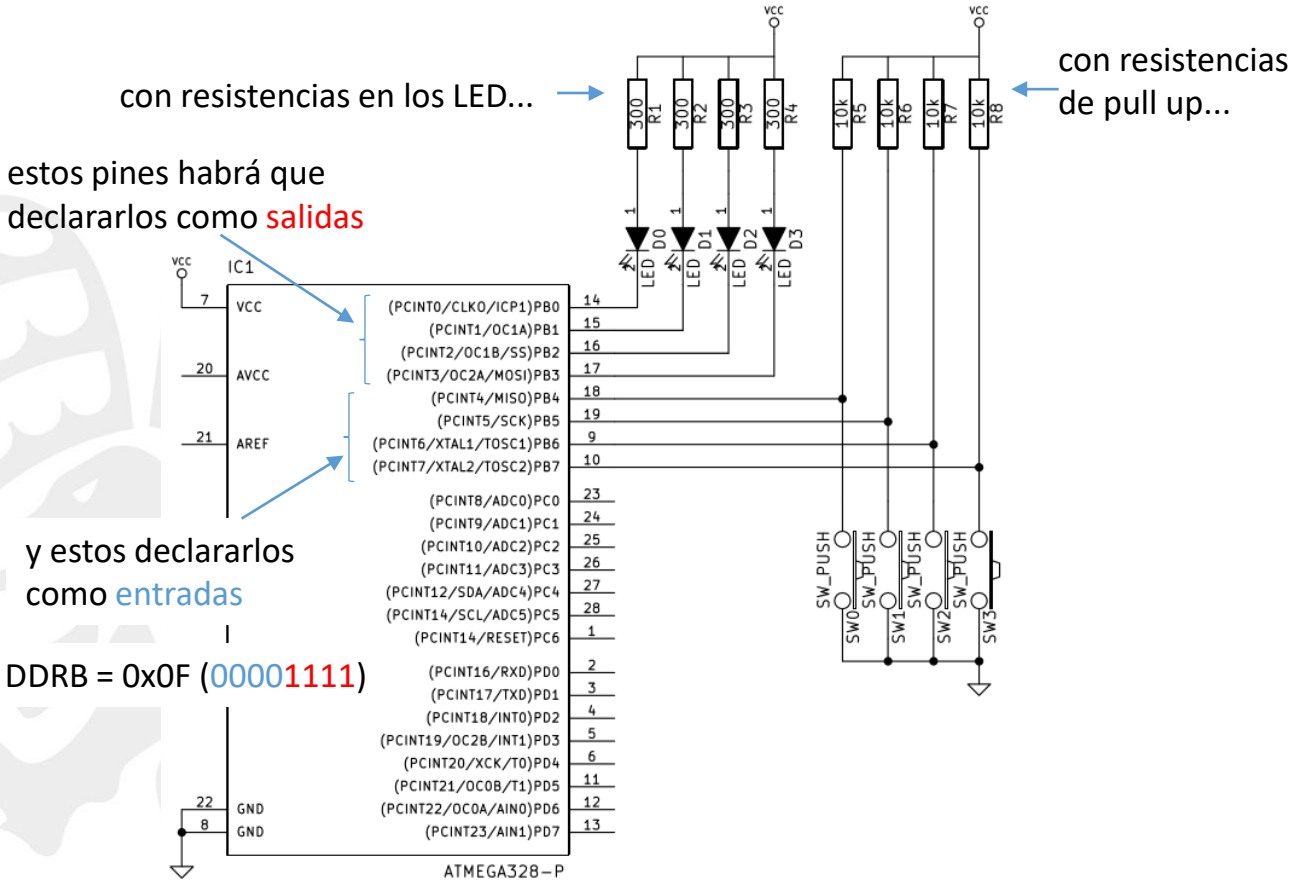


Nota

```
salidas = (entradas >> 4) & 0x0F; /* Los aisla */
```

i7 i6 i5 i4 x x x x >> 4 = 0 0 0 0 i7 i6 i5 i4

Ejemplo pulsadores y LEDs

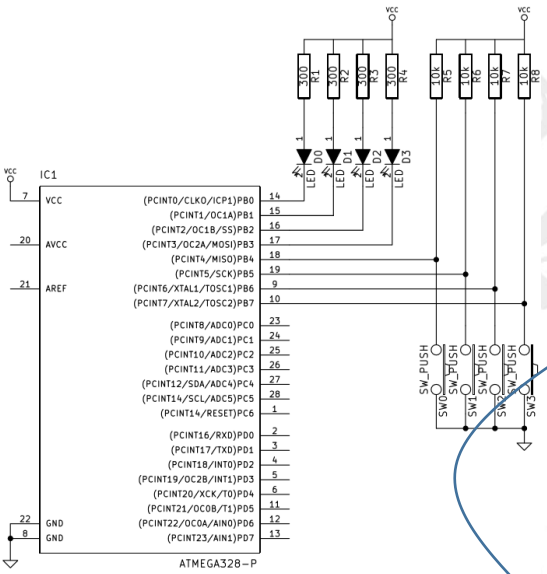


Ejemplo pulsadores y LEDs

- Programa que muestra en los LEDs el estado de los pulsadores, de forma que cuando se pulse el pulsador SW0 **se enciende** el LED0, **cuando se pulse** SW1 se enciende el LED1 y así sucesivamente

NOTA: según el esquema, para encender un LED la salida correspondiente debe ser un "0"

NOTA: según el esquema, cuando pulsamos la entrada correspondiente es un "0"




```
#include "Arduino.h"

main()
{
  uint8_t entradas, salidas;

  DDRB = 0x0F;            /* PB7-PB4 In, PB3-PB0 Out */
  while(1){                          /* Bucle infinito */
    entradas = PINB;          /* Lee los pulsadores */
    salidas = (entradas >> 4) & 0x0F; /* Los aisla */
    PORTB = salidas;           /* Actualiza salidas */
  }
}
```

includes

Hex, conf. puerto 00001111

Definimos variables

Nota

```
salidas = (entradas >> 4) & 0x0F; /* Los aisla */

i7 i6 i5 i4 x x x x >> 4 = 0 0 0 0 i7 i6 i5 i4
```

Ejemplo pulsadores y LEDs con Arduino (código un poco primitivo)

```
// EJEMPLO PULSADORES Y LEDS CON ARDUINO
/* defino otros pines porque en Arduino UNO
PB6 PB7 están conectados al cristal */
#define SW0 5
#define SW1 6
#define SW2 7
#define SW3 8
#define LED0 9
#define LED1 10
#define LED2 11
#define LED3 12
```

```
void setup() {
pinMode(SW0, INPUT);
pinMode(SW1, INPUT);
pinMode(SW2, INPUT);
pinMode(SW3, INPUT);
pinMode(LED0, OUTPUT);
pinMode(LED1, OUTPUT);
pinMode(LED2, OUTPUT);
pinMode(LED3, OUTPUT);
}

void loop() {
int sw0 = digitalRead(SW0);
int sw1 = digitalRead(SW1);
int sw2 = digitalRead(SW2);
int sw3 = digitalRead(SW3);
digitalWrite(LED0,sw0);
digitalWrite(LED1,sw1);
digitalWrite(LED2,sw2);
digitalWrite(LED3,sw3);
}
```

Ejemplo pulsadores y LEDs con Arduino (más elegante)

```
int numPinsSW = 4;
int pinsSW[] = {5, 6, 7, 8};
int numPinsLED = 4;
int pinsLED[] = {9, 10, 11, 12};

void setup() {
  for (int i = 0; i < numPinsSW; i++)
  {
    pinMode(pinsSW[i], INPUT);
  }
  for (int i = 0; i < numPinsLED; i++)
  {
    pinMode(pinsLED[i], OUTPUT);
  }
        }

void loop() {
  int lecturas[numPinsSW];
  for (int i = 0; i < numPinsSW; i++)
  {
     lecturas[i] = {digitalRead(pinsSW[i])};
  }
  for (int i = 0; i < numPinsLED; i++)
  {
    digitalWrite(pinsLED[i], lecturas[i]);
  }
       }
```

Problema 1. Sistema de alarma para casa

- Sistema de alarma tal que: interruptor para activar el sistema (ON), sensor que detecta apertura puerta (P), sensor que detecta apertura ventana (V), y salida para activar bocina (Al) al abrirse la puerta o la ventana.

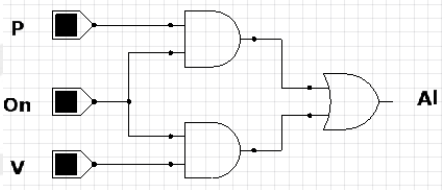
(a) Tabla de verdad

Nº Fila	ON	P	V	Al
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

(b) Diagrama de Karnaugh

ON \ P V	00	01	11	10
0	0_0	0_1	0_3	0_2
1	0_4	1_5	1_7	1_6

Figura 2.2: Obtención del diagrama de Karnaugh para una función.

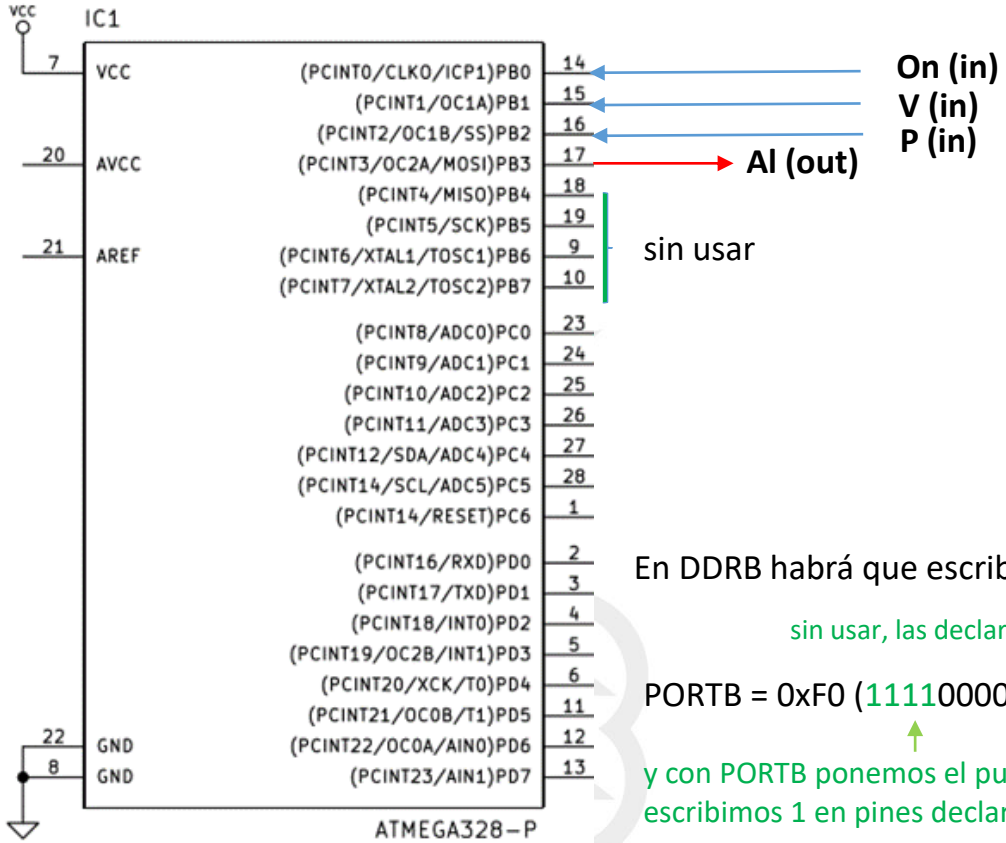


$$Al = ON \cdot V + ON \cdot P$$

Se pide diseñar el software de un sistema equivalente con el ATmega328P

Problema 1. Sistema de alarma para casa

PB7-PB4 no usadas (In), PB3 Out, PB2-PB0 In



PB3 OUT

En DDRB habrá que escribir 00001000 (0x08)

PB2-0 IN

sin usar, las declaramos IN

PORTB = 0xF0 (11110000)

y con PORTB ponemos el pull up, pues escribimos 1 en pines declarados como entradas

Problema 1. Sistema de alarma para casa

```
#include <Arduino.h>
#define ON (0) /* Números de bit de las entradas y salidas */
#define V (1)
#define P (2)
#define AL (3)

main()
{
  uint8_t entradas;
  uint8_t on, v, p, al;

  DDRB  = 0x08; /* PB7-PB4 no usadas (In), PB3 Out, PB2-PB0 In */
  PORTB = 0xF0; /* Activamos las resistencias de Pull-Up de las entradas
                   no usadas e inicializamos la salida a '0'*/

  while(1){
    entradas = PINB; /* Lee las entradas */
    on = (entradas >> ON) & 1; /* Copia el bit a su variable */
    v  = (entradas >> V ) & 1;
    p  = (entradas >> P ) & 1;
    /* Calculamos la función lógica*/
    al = on & v | on & p;
    /* Y actualizamos el bit de salida */
    if( al == 0x01 ){
      PORTB |= (1 << AL); /* Pone a 1 el bit */
    }else{ PORTB &= ~(1 << AL); /* Pone a 0 el bit */
    }
  }
}
```

includes

Para facilitar la manipulación de bits

$$Al = ON \cdot V + ON \cdot P$$

Definimos variables

Configuramos puerto, in-out

Lectura entradas

Aislamos los bits en variables

Función lógica

Actualizamos salidas

Problema 1. Sistema de alarma para casa. Paso a paso

```
#include <Arduino.h>
#define ON (0) /* Números de bit de las entradas y salidas */
#define V (1)
#define P (2)
#define AL (3)

main()
{
  uint8_t entradas;
  uint8_t on, v, p, al;
```

En MAYÚSCULAS. Asocia a cada identificador un número, que nos recuerda en que pin está cada cosa
Las entradas ON, V, P están en PBO,PB1, PB2, y la salida AL en PB3 (entradas y salidas se definen más abajo)

variables de tipo uint_t, consistentes en 8 bits. Atención, en minúscula, no son lo mismo que los #define previos

PB3 out

DDRB = 0x08 , o sea, 00001000

PB2-0 in

sin usar

```
  DDRB  = 0x08; /* PB7-PB4 no usadas (In), PB3 Out, PB2-PB0 In */
  PORTB = 0xF0; /* Activamos las resistencias de Pull-Up de las entradas
                   no usadas e inicializamos la salida a '0'*/
```

PORTB = 0xF0 , o sea, 11110000, de esta forma en los pines de entrada PB7-PB6 ponemos resistencias de pull-up. PB3 queda igual a 0 lógico (0 voltios) y queda inicializado así para que la alarma (salida) no esté activa al principio. Y al escribir ceros en PB2-PB0 quedan estás entradas sin que se active su pull-up, pues las vamos a usar y necesitaremos leer ahí el valor que haya (si ponemos el pull-up, se quedan fijas a 5 voltios, un 1 lógico)

Problema 1. Sistema de alarma para casa. Paso a paso

Esta condición se cumple siempre, de modo que tenemos un conjunto de instrucciones que se repiten por siempre

```
while(1){
```

En PINB (y en "entradas") tendremos 8 bits así: 1111 X PB2 PB1 PB0
Los 1 se deben a los pines no usados, que hemos declarado como entradas y puestos los pull-up, la X hace referencia al pin PB3 que es de salida y no sabremos que valor tendrá almacenado el registro PINB3, y PB2, PB1, PB0 son los valores leídos en los pines que hemos declarado como entrada y sí usamos

```
  entradas = PINB; /* Lee las entradas */
```

entradas se corre ON saltos (0) a la derecha, y al hacer AND con 00000001 queda 0000000 PB0

```
  on = (entradas >> ON) & 1; /* Copia el bit a su variable */
```

entradas se corre V saltos (1) a la derecha, y al hacer AND con 00000001 queda 0000000 PB1

```
  v  = (entradas >> V ) & 1;
```

entradas se corre P saltos (2) a la derecha, y al hacer AND con 00000001 queda 0000000 PB2

```
  p  = (entradas >> P ) & 1;
  /* Calculamos la función lógica*/
  al = on & v | on & p;
```

el resultado será 0000000Y, donde Y será el resultado de hacer la función on.v + on.p, de modo que será o bien 0000000**1** (se genera alarma) o 0000000**0** (no se genera). Observad que toda la operación se hace bit a bit, pero y el resultado de la función lógica solo interesa en el bit menos significativo de cada variable, los demás bits son todos 0

Problema 1. Sistema de alarma para casa. Paso a paso

NOTA: En C, cualquier valor diferente de 0 se evalúa como TRUE

```
/* Y actualizamos el bit de salida */
```

si al es 00000001, donde el 1 final es el resultado final de la operación lógica

```
if( al == 0x01 ){
```

Escribimos un 1 en la posición de salida correspondiente a AL (PB3). El detalle es este: tomamos 00000001 y los corremos AL veces (3) a la izquierda, queda 00001000. Si PORTB era 11110000 (aunque podría suponer algo genérico como XXXXXXXX), si hago 11110000 OR 00001000 queda 11111000, he escrito un 1 en la posición correspondiente a PB3 (si PORTB fuera XXXXXXXX es la misma idea, el resultado sería XXXX1XXX, escribiría un 1 en PB3)

```
PORTB |= (1 << AL); /* Pone a 1 el bit */
```

Y si al es 00000000, escribimos un 0 en la posición de salida correspondiente a AL (PB3). El detalle es este: tomamos 00000001 y los corremos AL veces (3) a la izquierda, queda 00001000, y después de negarlo queda 11110111. Si PORTB era 11110000 (aunque podría suponer algo genérico como XXXXXXXX), si hago 11110000 AND 11110111 queda 11110000, he escrito un 0 en la posición correspondiente a PB3 (si PORTB fuera XXXXXXXX es la misma idea, el resultado sería XXXX0XXX, escribiría un 0 en PB3)

```
    }else{ PORTB &= ~(1 << AL); /* Pone a 0 el bit */
    }
  }
}
```

Problema 2. Detector de números BCD divisibles por 3

Recordamos (tema 4) que gracias a Karnaugh se había obtenido la función que detecta que un número BCD es divisible por 3:

$$S = D_3 \cdot D_0 + D_2 \cdot D_1 \cdot \overline{D_0} + \overline{D_2} \cdot D_1 \cdot D_0$$

y con lógica combinacional se hace así

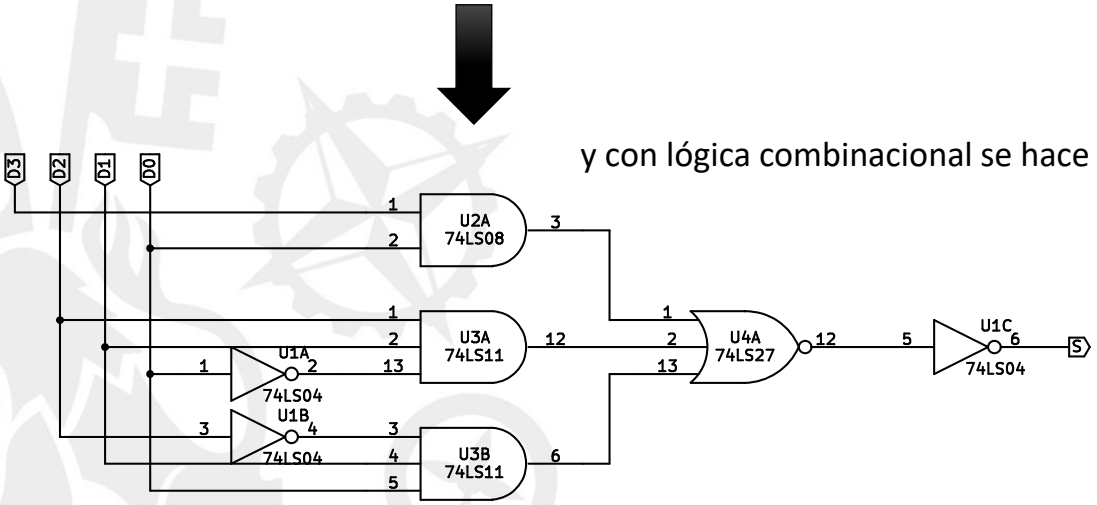
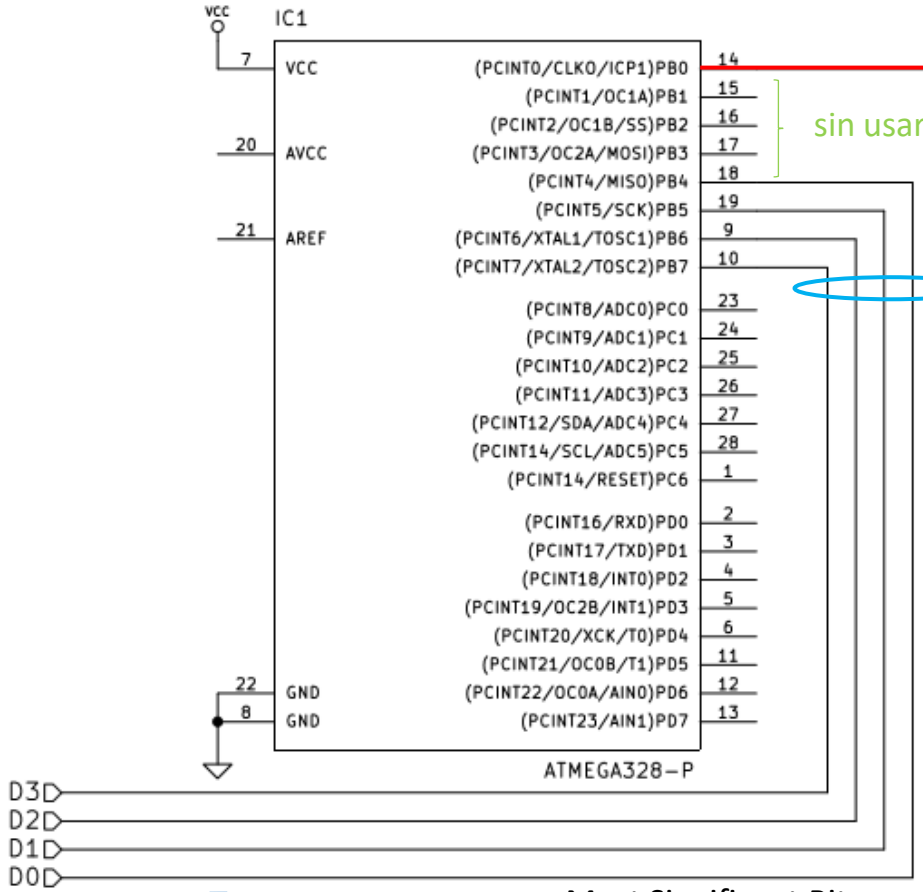


Figura 5.2: Detector de dígito BCD divisible entre 3 usando puertas lógicas.

Problema 2. Detector de números BCD divisibles por 3



En DDRB habrá que escribir 0000 0001 (0x01)

PB7-4 in — sin usar — PB0 out

Y para poner las resistencias de pull-up:
PORTB = 0000 1110 (0x0E)

Most Significant Bit

Least Significant Bit

Número BCD de entrada, codificado como D3 (MSB)-D2-D1-DO (LSB)

Problema 2. Detector de números BCD divisibles por 3

```
#include <Arduino.h>

#define S  (0) /* Números de bit de las entradas y salidas */
#define D0 (4)
#define D1 (5)
#define D2 (6)
#define D3 (7)

main()
{
  uint8_t entradas;
  uint8_t d3, d2, d1, d0, s;

  DDRB  = 0x01; /* PB7-PB4 In, PB3-PB1 No usadas (In) PB0 Out */
  PORTB = 0x0E; /* Resistencias Pull-up ON en PB3-PB1 y salida a '0'*/
  while(1){
    entradas = PINB; /* Lee las entradas */
    d3 = (entradas >> D3) & 1; /* Copia el bit a su variable */
    d2 = (entradas >> D2) & 1;
    d1 = (entradas >> D1) & 1;
    d0 = (entradas >> D0) & 1;

    /* Calculamos la función lógica*/
    s = d3 & d0 | d2 & d1 & ((~d0 ) & 0x01) | ((~d2) & 0x01) & d1 & d0;

    /* Y actualizamos el bit de salida */
    if( s == 0x01 ){
      PORTB |= (1 << S); /* Pone a 1 el bit */
    }else{
      PORTB &= ~(1 << S); /* Pone a 0 el bit */
    }
  }
}
```

includes

Para facilitar la manipulación de bits

$$S = D_3 \cdot D_0 + D_2 \cdot D_1 \cdot \overline{D_0} + \overline{D_2} \cdot D_1 \cdot D_0$$

Definimos variables

Configuramos puerto, in-out

Lectura entradas

Aislamos los bits

Función lógica

Actualizamos salidas

Problema 2. BCD divisibles por 3. Paso a paso

```
#include <Arduino.h>

#define S  (0) /* Números de bit de las entradas y salidas */
#define D0  (4)
#define D1  (5)
#define D2  (6)
#define D3  (7)

main()
{
  uint8_t entradas;
  uint8_t d3, d2, d1, d0, s;

  DDRB   = 0x01; /* PB7-PB4 In, PB3-PB1 No usadas (In) PB0 Out */

  PORTB = 0x0E; /* Resistencias Pull-up ON en PB3-PB1 y salida a '0'*/
```

En mayúscula. Asocia a cada identificador un número, que nos recuerda en que pin está cada cosa
Las entradas D3, D2, D1, D0 están en PB7,PB6, PB5, PB4, y la salida S en PB0 (entradas y salidas se definen más abajo)

variables de tipo uint_t, consistentes en 8 bits. Atención, en minúscula, no son lo mismo que los #define previos

PB3 out

DDRB = 0x01 , o sea, 0000 0001

PB72-4 in

sin usar

PORTB = 0x0E , o sea, 00001110, de esta forma en los pines de entrada PB3-PB1 ponemos resistencias de pull-up. PB1 queda igual a 0 lógico (0 voltios) y queda inicializado así para que la salida no esté activa al principio. Y al escribir ceros en PB7-PB4 quedan estás entradas sin que se active su pull-up, pues las vamos a usar y necesitaremos leer ahí el valor que haya (si ponemos el pull-up, se quedan fijas a 5 voltios, un 1 lógico)

Problema 2. BCD divisibles por 3. Paso a paso

Esta condición se cumple siempre, de modo que tenemos un conjunto de instrucciones que se repiten por siempre

```
while(1){
    entradas = PINB; /* Lee las entradas */
```

En PINB (y en "entradas") tendremos 8 bits así: PB7 PB6 PB5 PB4 111 X
Los 1 se deben a los pines no usados, que hemos declarado como entradas y puestos los pull-up, la X hace referencia al pin PB0 que es de salida y no sabremos que valor tendrá almacenado el registro PINB, y PB7, PB6, PB5, PB4 son los valores leídos en los pines que hemos declarado como entrada y sí usamos

entradas se corre D3 saltos (7) a la derecha, y al hacer AND con 00000001 queda 0000000 PB7

```
d3 = (entradas >> D3) & 1; /* Copia el bit a su variable */
```

entradas se corre D2 saltos (6) a la derecha, y al hacer AND con 00000001 queda 0000000 PB6

```
d2 = (entradas >> D2) & 1;
```

entradas se corre D1 saltos (5) a la derecha, y al hacer AND con 00000001 queda 0000000 PB5

```
d1 = (entradas >> D1) & 1;
```

entradas se corre D0 saltos (4) a la derecha, y al hacer AND con 00000001 queda 0000000 PB4

```
d0 = (entradas >> D0) & 1;
```

Problema 2. BCD divisibles por 3. Paso a paso

$$S = D_3 \cdot D_0 + D_2 \cdot D_1 \cdot \overline{D_0} + \overline{D_2} \cdot D_1 \cdot D_0$$

```
   /* Calculamos la función lógica*/
```

el resultado será 0000000Y, donde Y será el resultado de hacer la función lógica, de modo que será o bien 0000000**1** (divisible por 3) o 0000000**0** (no divisible). Observad que toda la operación y el resultado de la función lógica se hace bit a bit, pero solo interesa el bit menos significativo de cada variable, los demás bits son todos 0

```
    s = d3 & d0 | d2 & d1 & ((~d0 ) & 0x01) | ((~d2) & 0x01) & d1 & d0;
```

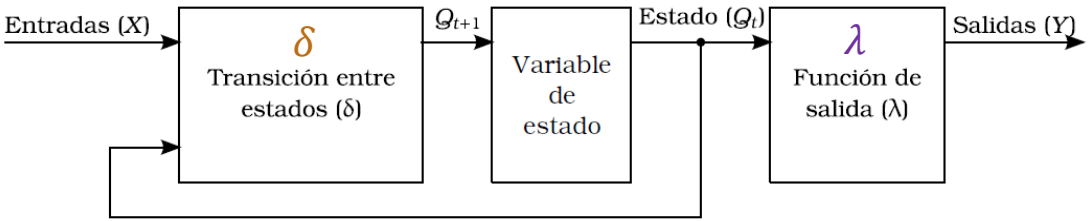
Atención a este paso. d0 es 0000000PB4 y $\sim d0$ es 1111111$\overline{PB4}$ hacemos AND con 00000001 para que quede así: 0000000 $\overline{PB4}$ de modo que dejamos todos los bits a 0 salvo el menos significativo

Aquí hacemos lo mismo

```
    /* Y actualizamos el bit de salida */
    if( s == 0x01 ){
      PORTB |= (1 << S); /* Pone a 1 el bit */
    }else{
      PORTB &= ~(1 << S); /* Pone a 0 el bit */
    }
  }
}
```

Máquinas de estados finitos

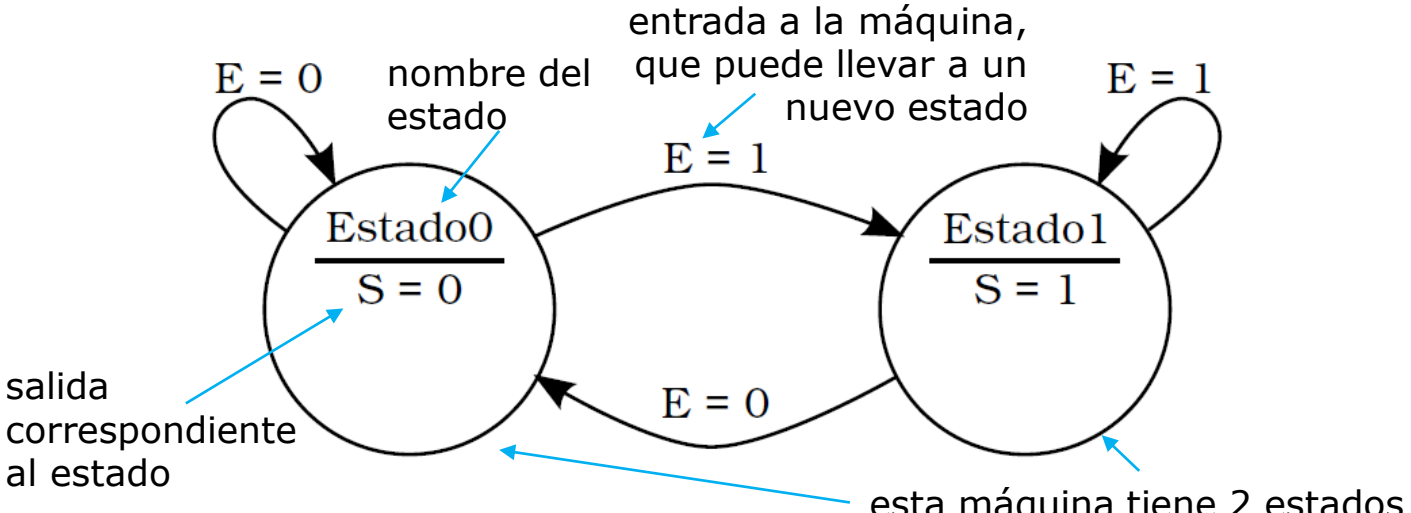
- Son circuitos que permiten "tomar decisiones" en función del estado actual del circuito y de sus entradas.
- Dos tipos: Moore y Mealy. Estudiaremos sólo Moore
- Las máquinas de Moore disponen de:
  - X entradas e Y salidas.
  - Q variables de estado que se almacenan en una memoria.
  - Dos funciones lógicas: δ para transición entre estados y λ para obtener las salidas a partir del estado actual.



Máquinas de estados finitos

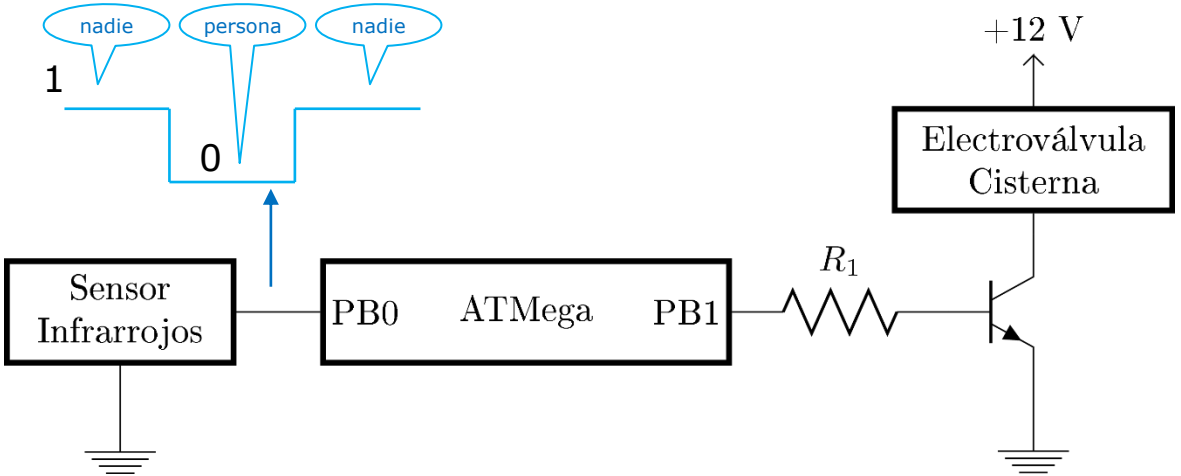
Representación gráfica:

- Los estados se representan mediante círculos con el nombre del estado y el valor de las salidas para ese estado.
- Las transiciones se representan por flechas, con el valor de las entradas que provocan la transición



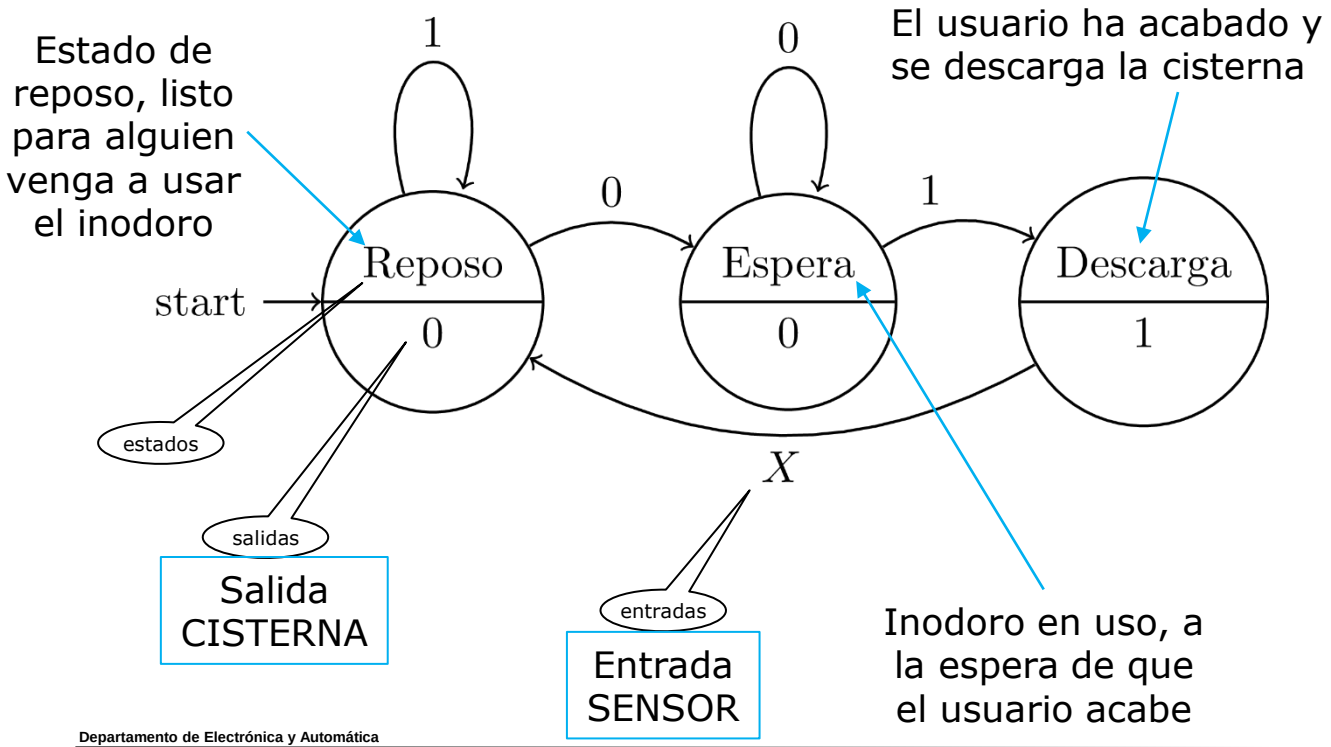
Problema 5: Cisterna automática

- Se desea automatizar la cisterna de un inodoro. Cuando llegue una persona al inodoro, se esperará hasta que se vaya y en ese momento se activará la cisterna.
- Para ello se instala un sensor de infrarrojos que detecta cuando una persona está en el inodoro, dando un nivel bajo. Por otro lado se instala una electroválvula que funciona a 12 V para descargar la cisterna.



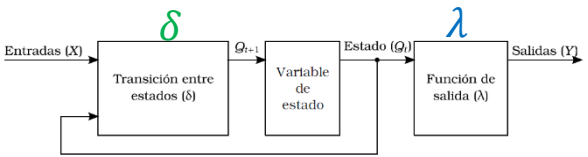
Problema 5: Cisterna automática

- Para implantar este automatismo es necesario una máquina de estados:



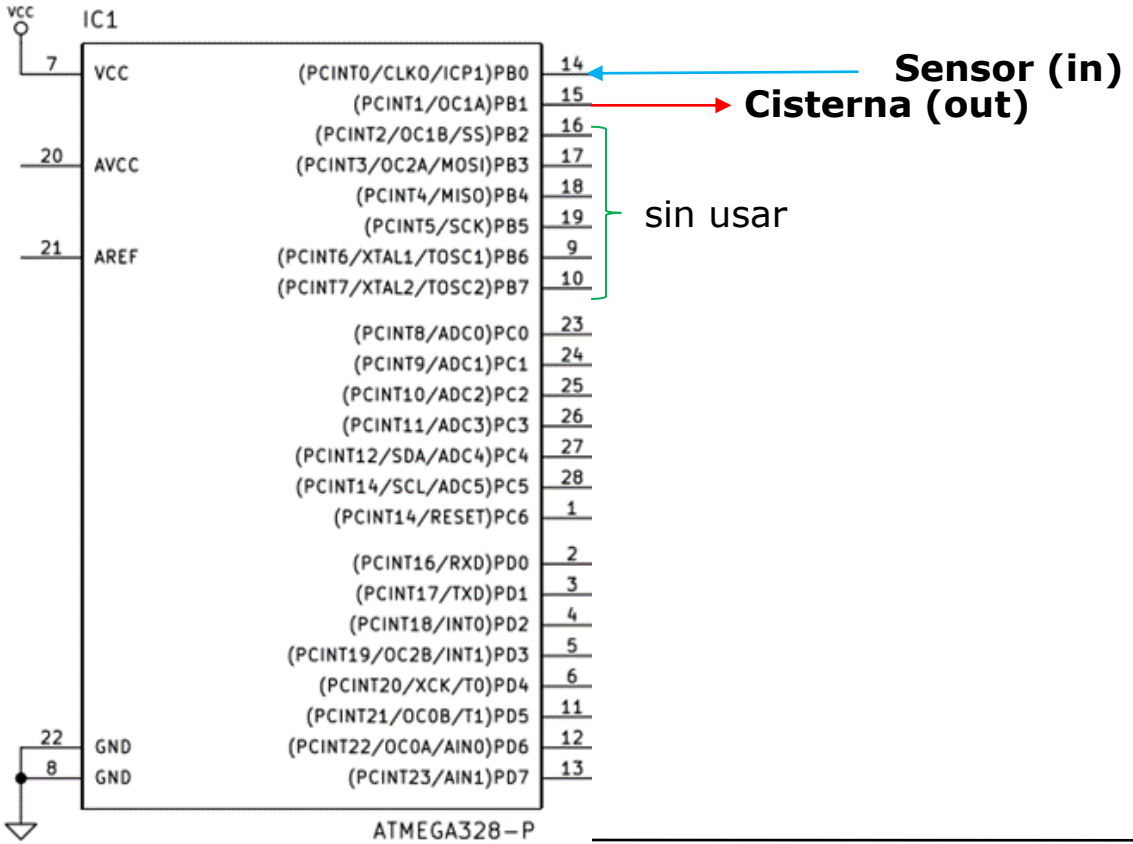
Problema 5: Cisterna automática

```
#include …
#define … // posiciones entradas y salidas

int main()
{
  // declaración de variables
  // declaración de estados (enum)
  // programación del puerto
  // inicializaciones (ej: estado inicial)

  while(1){
  // lectura entradas
  // bloque de transición de estados (switch case), función delta
  // bloque de cálculo de salidas, función lamda
  // actualización salidas

  }
}
```



Problema 5: Cisterna automática

PB7-PB2 no usadas (In), PB1 Out, PB0 In



Problema 5: Cisterna automática

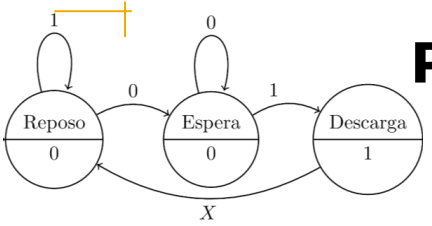
```
/* Control automatico para la descarga de una cisterna */

#include <Arduino.h>
#define SENSOR (0) // Números de bit de las entradas y salidas
#define CISTERNA (1) // SENSOR en PB0, y CISTERNA en PB1

int main()
{
  // declaración de variables y estados
  uint8_t sensor, cisterna; //variables
  enum {Reposo, Espera, Descarga} estado;  //estados

  // programación del puerto
  DDRB  = 0x02; // PB7-PB2 No usados (In), PB1 Out, PB0 In
  PORTB = 0xFC; // Resistencias de Pull-Up ON en PB7 -PB2, salida PB1
a'0', y entrada PB0 sin pull up
  // Inicializaciones: variable estado al estado inicial
  estado = Reposo;

  // bucle infinito
  while(1){ sensor = (PINB >> SENSOR) & 1; //Se lee la entrada
```

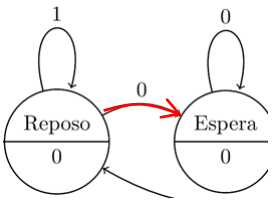
00000010

11111100

111111XPB0 AND 00000001 = 0000000PB0

Problema 5: Cisterna automática

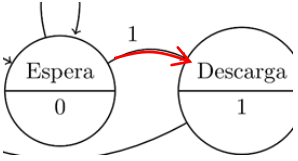


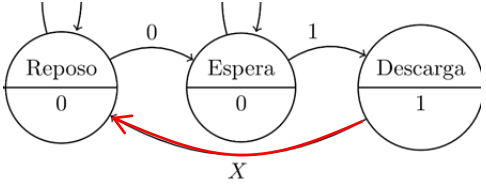



```
// bloque de transición de estados (función delta)
switch (estado){
case Reposo:
  if (sensor == 0){ // Ha llegado alguien al inodoro
    estado = Espera;
  }
  break;
case Espera:
  if (sensor == 1){ // El usuario ha terminado
    estado = Descarga;
  }
  break;
case Descarga:
  estado = Reposo;
  break;
}
```

si sensor = 0000000PB0 = 0000000 (leemos un 0 en PBO)

si sensor = 0000000PB0 = 0000001 (leemos un 1 en PBO)





Problema 5: Cisterna automática

```
    // bloque de cálculo de salidas (función lambda)
    if (estado == Descarga){
      cisterna = 1;
    }else{
      cisterna = 0;
    }

    // bloque para la actualización de las salidas
    if( cisterna == 1 ){
      PORTB |= (1 << CISTERNA); // Pone a 1 el bit PB1
    }else{
      PORTB &= ~(1 << CISTERNA); // Pone a 0 el bit PB1
    }
  } // fin while(1)
} // fin main
```

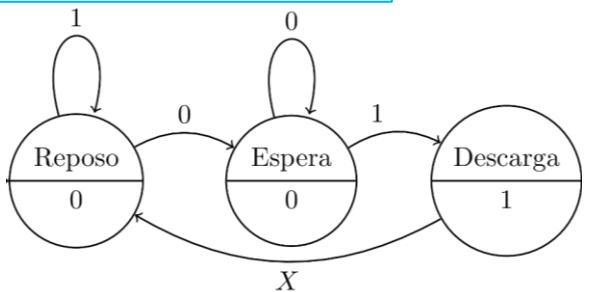
cisterna = 00000001

cisterna = 00000000

XXXXXXXX OR 00000010 = XXXXXX1X

de 00000001 a 00000010

XXXXXXXX AND 11111101 = XXXXXX0X

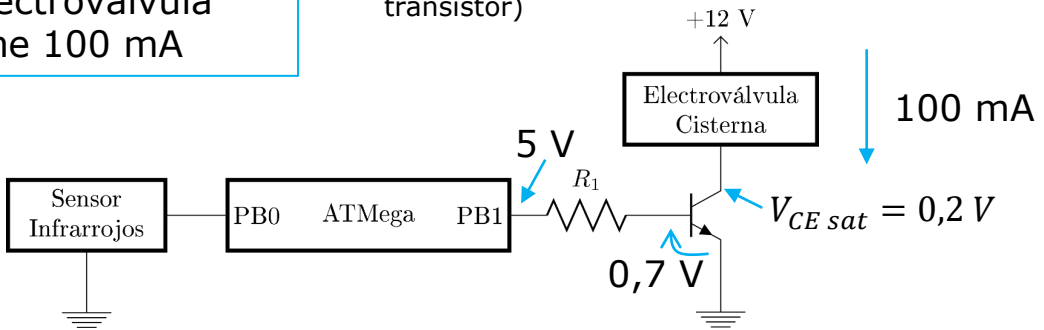


Problema 5: Cisterna automática

- Vamos a completar el problema con el diseño de la salida

supongamos que nos dicen que la electroválvula consume 100 mA

(**No hay que calcular ninguna R equivalente** de la válvula, ya nos dan la corriente, que será la $I_{C\ SAT}$ del transistor)



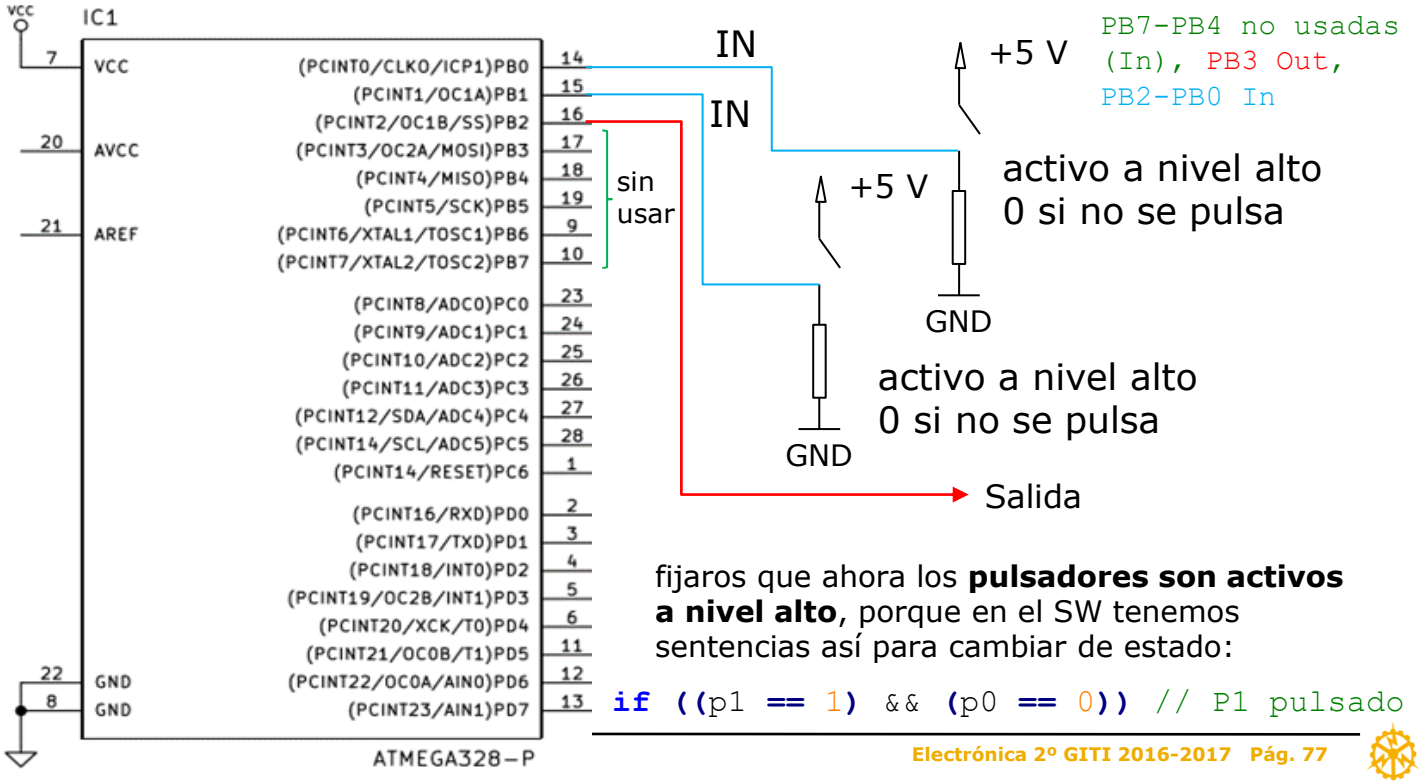
diseñamos la salida para que el TRT esté saturado, $\beta = 100$ y factor de seguridad de 5

$$I_B = 5\frac{100\ mA}{100} = 5\ mA = \frac{5\ V - 0{,}7\ V}{R_1} \rightarrow \boldsymbol{R_1 = 860\ \Omega}$$

(o nos podrían haber dado la potencia de la válvula, y de ahí tendríamos que calcular su corriente)

Problema 6: cerradura electrónica

- Se desea diseñar una cerradura electrónica que consta de dos pulsadores P1 y P0 y un relé de salida para abrir la puerta.
- La salida se activará cuando se pulse P1, luego P0 y por último otra vez P0. La salida permanecerá activa hasta que se suelte la tecla P0.

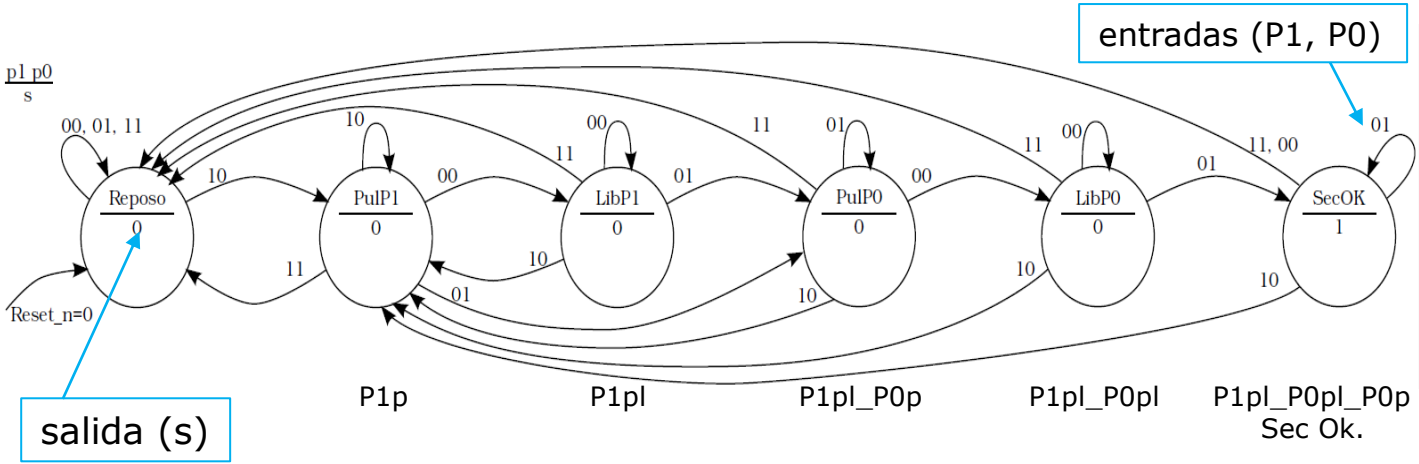


fijaros que ahora los **pulsadores son activos a nivel alto**, porque en el SW tenemos sentencias así para cambiar de estado:

```
if ((p1 == 1) && (p0 == 0)) // P1 pulsado
```

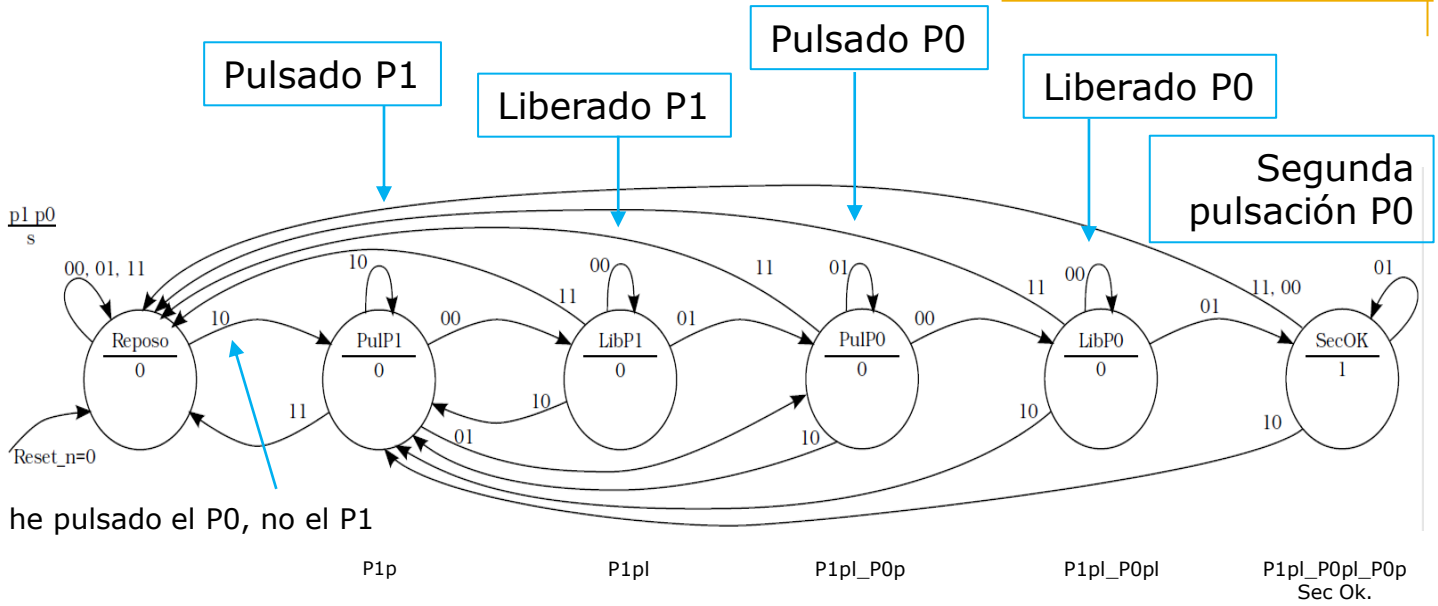
Problema 6: cerradura electrónica

- Se desea diseñar una cerradura electrónica que consta de dos pulsadores P1 y P0 y un relé de salida para abrir la puerta.
- La salida se activará cuando se pulse P1, luego P0 y por último otra vez P0. La salida permanecerá activa hasta que se suelte la tecla P0.



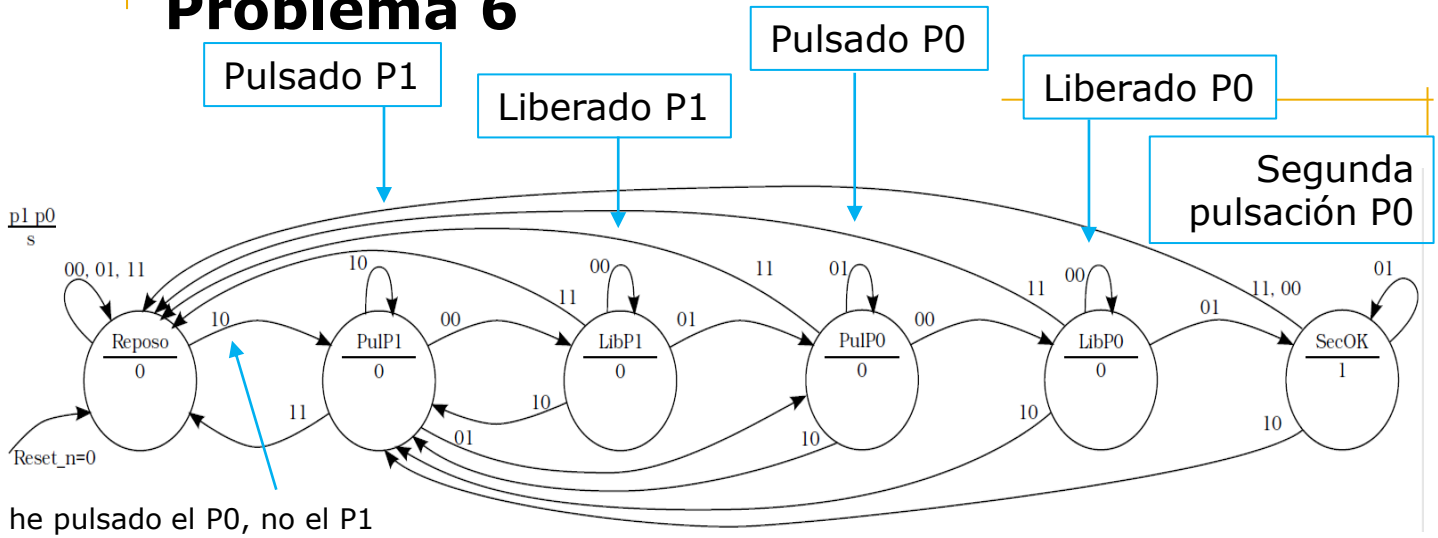
- Se detecta la secuencia de teclas P1-P0-P0, que se traduce en: P1pl_P0pl_P0p
- Se controla cuando se pulsa y se libera cada tecla.

Problema 6: cerradura electrónica

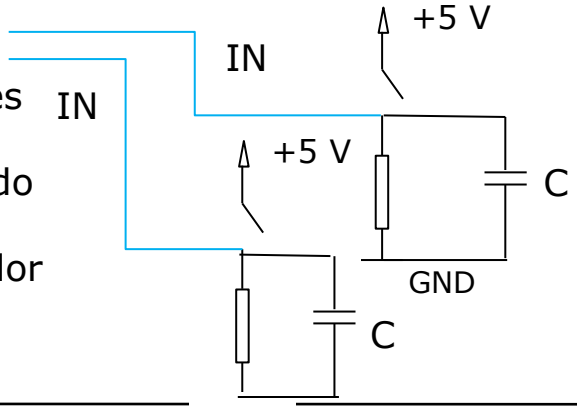


- Se detecta la secuencia de teclas P1-P0-P0, que se traduce en: P1pl_P0pl_P0p
- Se controla **cuando se pulsa y se libera cada tecla**.

Problema 6



OJO: para evitar que los rebotes provoquen por ejemplo la secuencia "pulsado P0", "liberado P0", "segunda pulsación P0", habría que poner un condensador en los pulsadores (o un "delay" en el código)



Problema 6: cerradura electrónica

```
/* Cerradura electrónica implantada mediante una máquina de estados */
#include <Arduino.h>

#define P0 (0) /* Números de bit de las entradas y salidas */
#define P1 (1)
#define S  (2)

int main()
{
  uint8_t entradas;
  uint8_t p1, p0, s;
  /* Definición de la variable para almacenar los estados */
  enum {Reposo, PulP1, LibP1, PulP0, LibP0, SecOK} estado;

  DDRB  = 0x04; /* PB7-PB3 No usados (In), PB2 Out, PB1-PB0 In */
  PORTB = 0xF8; /* Resistencias de Pull-Up ON en PB7-PB3, salida '0',
entradas sin pull up */

  estado = Reposo; /* Arrancamos en el estado inicial */
```

00000100

11111000

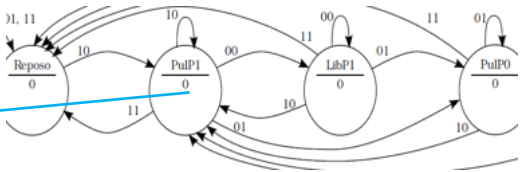
Problema 6: cerradura electrónica

```
while(1){
    entradas = PINB;            /* Lee las entradas */
    p1 = (entradas >> P1) & 1; /* Copia el bit a su variable */
    p0 = (entradas >> P0) & 1;
    switch (estado){ // Transición de estados (función delta)
    case Reposo:
      if ((p1 == 1) && (p0 == 0)){ // P1 pulsado
    estado = PulP1;
      }
      break;
    case PulP1:
      if ((p1 == 0) && (p0 == 0)){ // P1 liberado
    estado = LibP1;
      }else if ((p1 == 0) && (p0 == 1)){// P1 liberado y P0 pulsado al
    estado = PulP0;                    // mismo tiempo
      }else if ((p1 == 1) && (p0 == 1)){// P1 y P0 pulsados a la vez
    estado = Reposo;                   // se considera un error
      }
      break;
```

entradas = 11111XPB1PB0

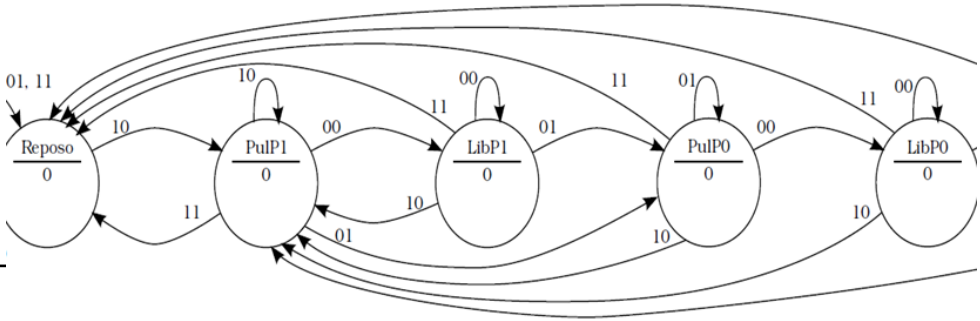
p1 =0000000PB1

p0 =0000000PB0



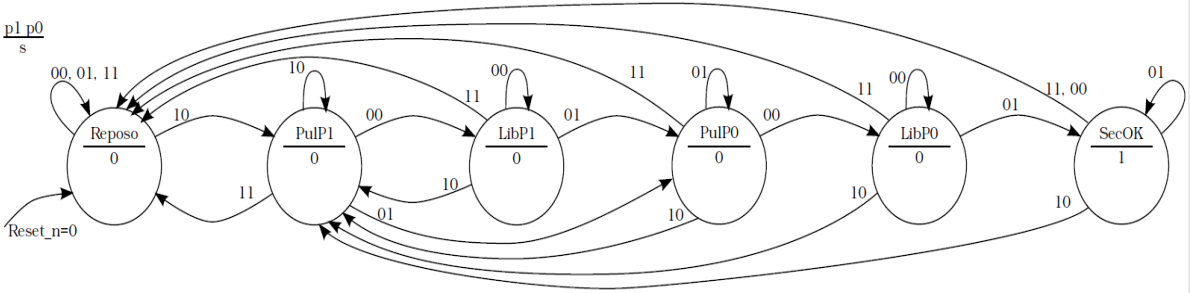
Problema 6: cerradura electrónica

```
case LibP1:
  if ((p1 == 0) && (p0 == 1)){ // siguiente tecla de la secuencia
estado = PulP0;
  }else if ((p1 == 1) && (p0 == 0)){ // al principio de la
estado = PulP1;                      // secuencia.
  }else if ((p1 == 1) && (p0 == 1)){
estado = Reposo;
  }
  break;
case PulP0:
  if ((p1 == 0) && (p0 == 0)){ // P0 liberado
estado = LibP0;
  }else if ((p1 == 1) && (p0 == 0)){ // al principio de la
estado = PulP1;                      // secuencia.
  }else if ((p1 == 1) && (p0 == 1)){
estado = Reposo;
  }
  break;
```



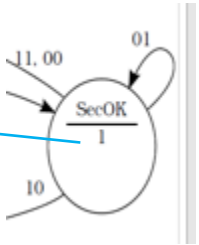
Problema 6: cerradura electrónica

```
case LibP0:
  if ((p1 == 0) && (p0 == 1)){ // Segunda pulsación de P0
estado = SecOK;
  }else if ((p1 == 1) && (p0 == 0)){// al principio de la
estado = PulP1;                        // secuencia.
  }else if ((p1 == 1) && (p0 == 1)){
estado = Reposo;
  }
  break;
case SecOK:
  if ((p1 == 1) &&(p0 == 0)){ // al principio de la secuencia
estado = PulP1;
  }else if ((p1 == 1) &&(p0 == 1)){
estado = Reposo;
  }else if ((p1 == 0) &&(p0 == 0)){
estado = Reposo;
  }
  break;
}
```



Problema 6: cerradura electrónica

```
// Función Lambda
if (estado == SecOK){
  s = 1;
}else{
  s = 0;
}

/* Y actualizamos los bits de salida */
if( s == 0x01 ){
  PORTB |=  (1 << S); /* Pone a 1 el bit */
```



pasamos de 00000001 a 00000100, y al hacer el OR en PORTB queda XXXXX1XX

```
}else{
  PORTB &= ~(1 << S); /* Pone a 0 el bit */
}
  }
}
```

de 00000001 a 00000100, luego 11111011, y con el AND en PORTB queda XXXXX0XX

Problema 7: Control de aparcamiento

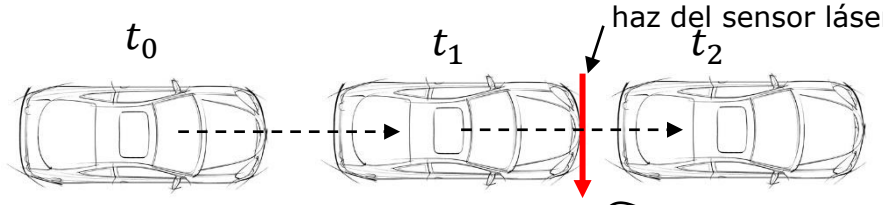
- Una tienda dispone de un pequeño aparcamiento para sus clientes con capacidad para 4 coches.

- Diseñar la máquina de estados necesaria para automatizarlo, de forma que se conozca en todo momento el número de vehículos dentro del parking, y un semáforo verde indique la existencia de plazas libres y otro en rojo que el parking está completo.

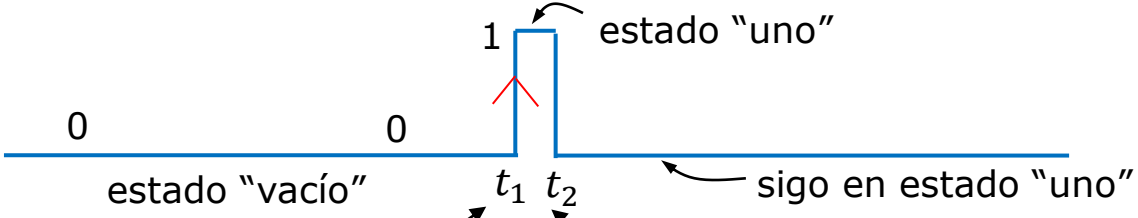
- Se dispone de un sensor E que detecta la entrada de un coche, y de otro sensor S que detecta la salida.

Vamos a diseñarlo con un detector de flancos

Problema 7: Control de aparcamiento



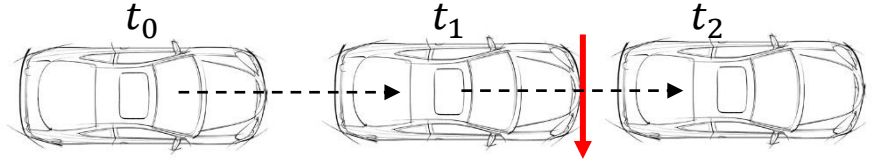
CON FLANCOS



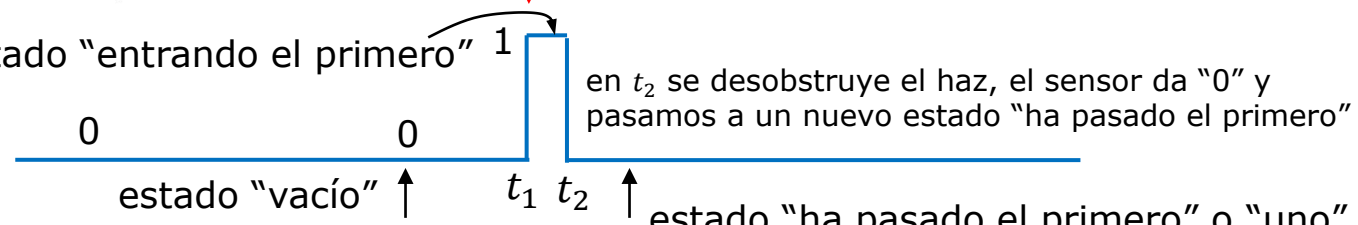
en t_1 se obstruye el haz, el sensor da "1" y pasamos al estado "uno"

en t_2 se desobstruye el haz, el sensor da "0" y seguimos en estado "uno". Solo miramos el flanco de subida

---

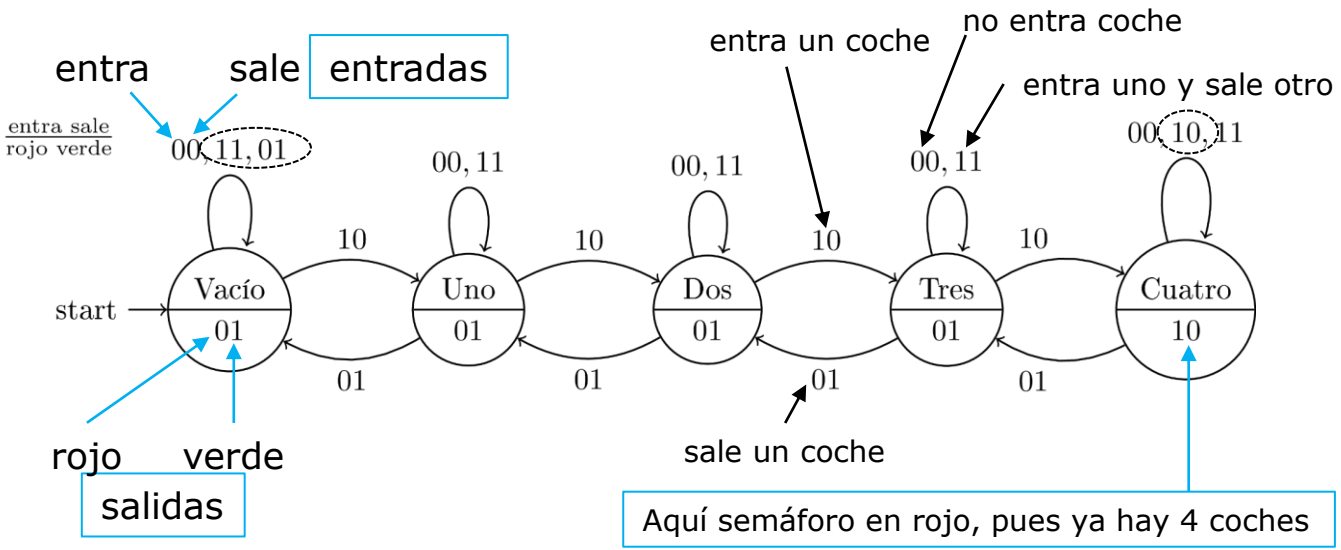


SIN FLANCOS



Problema 7: Control de aparcamiento

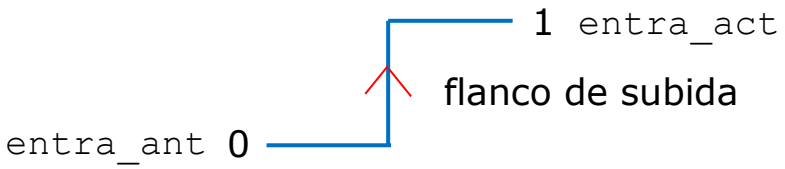
- Una tienda dispone de un pequeño aparcamiento para sus clientes con capacidad para 4 coches.
- Diseñar la máquina de estados necesaria para automatizarlo, de forma que se conozca en todo momento el número de vehículos dentro del parking, y un semáforo verde indique la existencia de plazas libres y otro en rojo que el parking está completo.
- Se dispone de un sensor "entra" que detecta la entrada de un coche, y de otro sensor "sale" que detecta la salida.



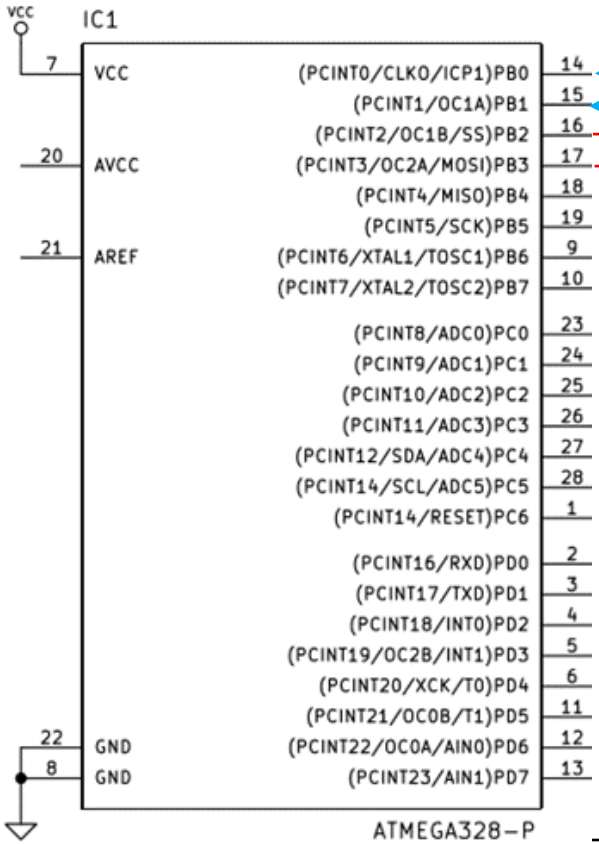
Problema 7: Control de aparcamiento

- A diferencia de los ejemplos anteriores, en este caso es necesario detectar flancos en las entradas:
  - Si se aumenta el contador de coches mientras el sensor *entra* está a 1 contaremos múltiples coches mientras que solo uno está entrando (NOTA. Se puede hacer sin flancos, pero saldrían más estados, como en el problema anterior, deberíamos detectar el paso de 0 a 1 (entra el morro del coche), y luego de 1 a 0 (pasa la parte de atrás))
  - Sólo hay que incrementar el contador de coches cuando la señal del sensor *entra* pase de 0 a 1: por tanto hay que detectar su flanco de subida:
  - Utilizamos dos variables: entra_act (valor actual de entra) y entra_ant (valor anterior de entra) y la detección de flanco de subida se hace con:

```
if ((entra_act != entra_ant) && (entra_act == 1)) { …
```



Problema 7: Control de aparcamiento



Sensor ENTRA(in)

Sensor SALE (in)

ROJO (out)

VERDE (out)

sin usar

```
DDRB  = 0x0C   00001100
PORTB = 0xF0   11110000
```


Problema 7: Control de Aparcamiento

```
/* Control del semaforo de un aparcamiento */
#include <Arduino.h>
#define ENTRA (0) // Números de bit de las entradas y salidas
#define SALE  (1) // Están conectadas al puerto B
#define ROJO  (2)
#define VERDE (3)

int main() {
  uint8_t entradas, entra, sale, rojo, verde;
  // Variables para detectar los flancos en las entradas
  uint8_t entra_ant, entra_act, sale_ant, sale_act;
  // Estados
  enum {Vacio, Uno, Dos, Tres, Cuatro} estado;
  DDRB  = 0x0C; // PB7-PB4 No usados (In), PB3-PB2 Out, PB1-PB0 In
  PORTB = 0xF0; // Pull-Up en PB7-PB4, salidas a'0', entradas sin pull up
  // Se inicializan las señales para los detectores de flanco
  entradas = PINB;
  entra_ant = (entradas >> ENTRA) & 1;
  sale_ant = (entradas >> SALE) & 1;
```

00001100 11110000

entradas = 1111XXPB1PB0

1111XXPB1PB0 & 00000001=0000000PB0

01111XXPB1 & 00000001=0000000PB1

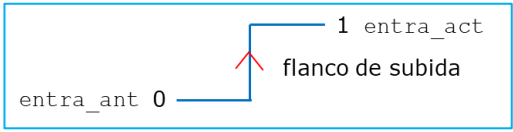
Problema 7: Control de Aparcamiento

```
// Se inicializa la variable estado al estado inicial
estado = Vacio;
while(1){
  entradas = PINB;            // Se leen las entradas
  entra_act = (entradas >> ENTRA) & 1;
  sale_act  = (entradas >> SALE)  & 1;

  // Se detectan flancos de subida
  if ((entra_act != entra_ant) && (entra_act == 1)){
    entra = 1;
  }else{
    entra = 0;
  }
  if ((sale_act != sale_ant) && (sale_act == 1)){
    sale = 1;
  }else{
    sale = 0;
  }
  entra_ant = entra_act;
  sale_ant = sale_act;
```

volvemos a leer las entradas (PINB)

0000000PB0 (la nueva lectura de PB0)

0000000PB1 (la nueva lectura de PB1)



Ha entrado un coche

Ha salido un coche

actualizamos las variables, lo que era una lectura actual pasa a ser una lectura antigua

Problema 7: Control de Aparcamiento

```
// Transición de estados (función delta)
switch (estado){
case Vacio:
  if ((entra == 1) && (sale == 0)){ // Entra un coche
    estado = Uno;
  }
  break;
case Uno:
  if ((entra == 1) && (sale == 0)){ // Entra un coche
    estado = Dos;
  }else if ((entra == 0) && (sale == 1)){ // Sale un coche
    estado = Vacio;
  }
  break;
```

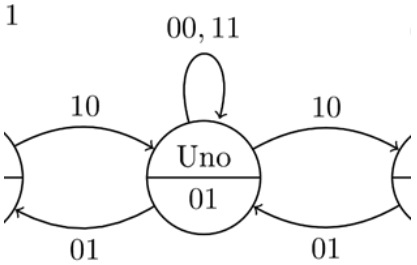
esto podría sobrar, si estamos en "Vacio" no pueden salir coches...

pasamos de estado "vacío" a "uno" solo si entra un coche, el cualquier otro caso seguimos en vacío

pasamos de "uno" a "vacío" si no entra un coche y sale un coche

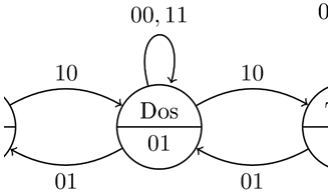
pasamos de estado "uno" a "dos" si entra un coche y no sale ninguno

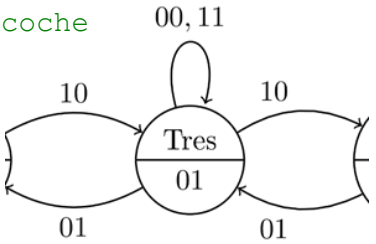
En las demás combinaciones (11, 00) permanecemos en estado "uno"

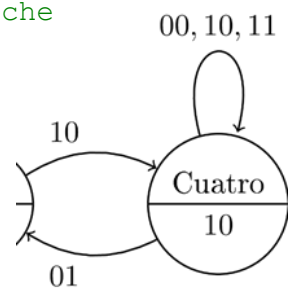


Problema 7: Control de Aparcamiento

```
case Dos:
  if ((entra == 1) && (sale == 0)){ // Entra un coche
    estado = Tres;
  }else if ((entra == 0) && (sale == 1)){ // Sale un coche
    estado = Uno;
  }
  break;
case Tres:
  if ((entra == 1) && (sale == 0)){ // Entra un coche
    estado = Cuatro;
  }else if ((entra == 0) && (sale == 1)){ // Sale un coche
    estado = Dos;
  }
  break;
case Cuatro:
  if ((entra == 0) && (sale == 1)){ // Sale un coche
    estado = Tres;
  }
  break;
}
```







Problema 7: Control de Aparcamiento

```
    // Función Lambda
    if ((estado == Vacio) || (estado == Uno) || (estado == Dos) ||
        (estado == Tres)){
      verde = 1;
      rojo  = 0;
    }else{ // Estaremos en el estado Cuatro
      verde = 0;
      rojo  = 1;
    }

    // Se actualizan los bits de salida
    if( verde == 1 ){
      PORTB |=  (1 << VERDE); // Pone a 1 el bit
    }else{
      PORTB &= ~(1 << VERDE); // Pone a 0 el bit
    }
    if( rojo == 1 ){
      PORTB |=  (1 << ROJO); // Pone a 1 el bit
    }else{
      PORTB &= ~(1 << ROJO); // Pone a 0 el bit
    }
  }
}
```

00000001

00000000

00000000

00000001

pasamos de 00000001 a 0000 1000, y al hacer el OR en PORTB queda XXXX 1XXX

de 00000001 a 00000100, luego 1111 0111, y con el AND en PORTB queda XXXX 0XXX

Problema 8: campana extractora

- A un primo suyo se le acaba de romper el circuito de control de la campana extractora de la cocina. Como sabe que usted es un experto en electrónica, le pide que diseñe uno nuevo. La campana tiene dos botones, uno para pararla (P) y otro para arrancarla y para seleccionar la velocidad (M). Cuando la campana está parada, la primera pulsación del botón M pondrá el motor del extractor en marcha a velocidad lenta. Si se vuelve a pulsar se pasará a la velocidad rápida y si se pulsa otra vez el motor conmutará a la velocidad ultra-rápida. La siguiente pulsación de M cambiará a la velocidad lenta, repitiéndose el ciclo indefinidamente. En cualquier momento si se pulsa el botón P la campana se parará. Si se pulsan los dos botones a la vez la campana no modificará su régimen de funcionamiento. El motor de la campana extractora dispone de tres entradas para controlar su velocidad: L (lenta), R (rápida) y UR (ultra-rápida).

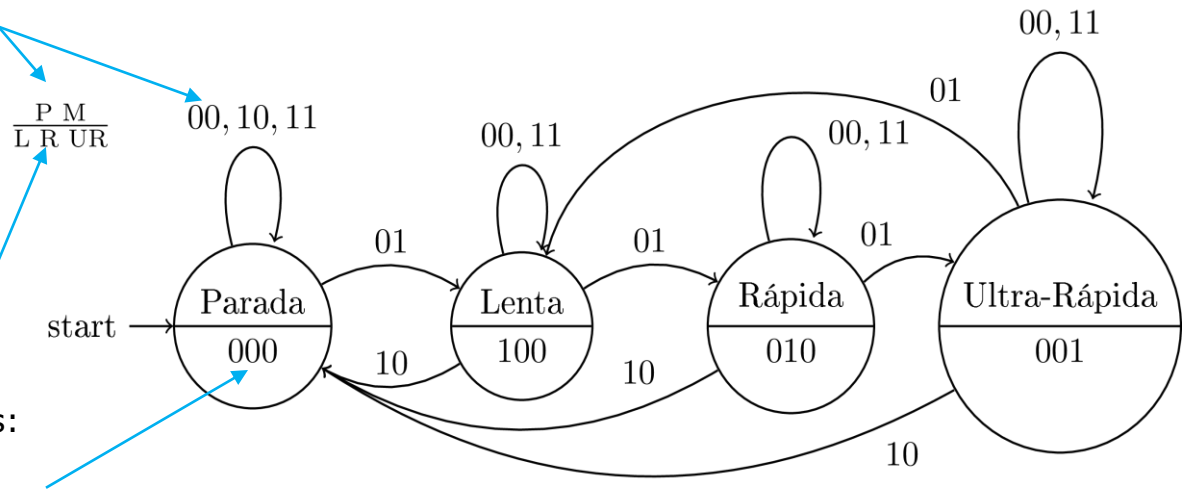
  que serán salidas del microcontrolador

  El sistema a diseñar se implantará mediante una máquina de estados usando un microcontrolador.

- El diseño ha de incluir:
  - El diagrama de estados.
  - La asignación de entradas y salidas a las patillas del Microcontrolador ATmega328P. Use la figura 4.15 como punto de partida.
  - El código en C del microcontrolador para implantar la máquina de estados.

Problema 8: campana extractora

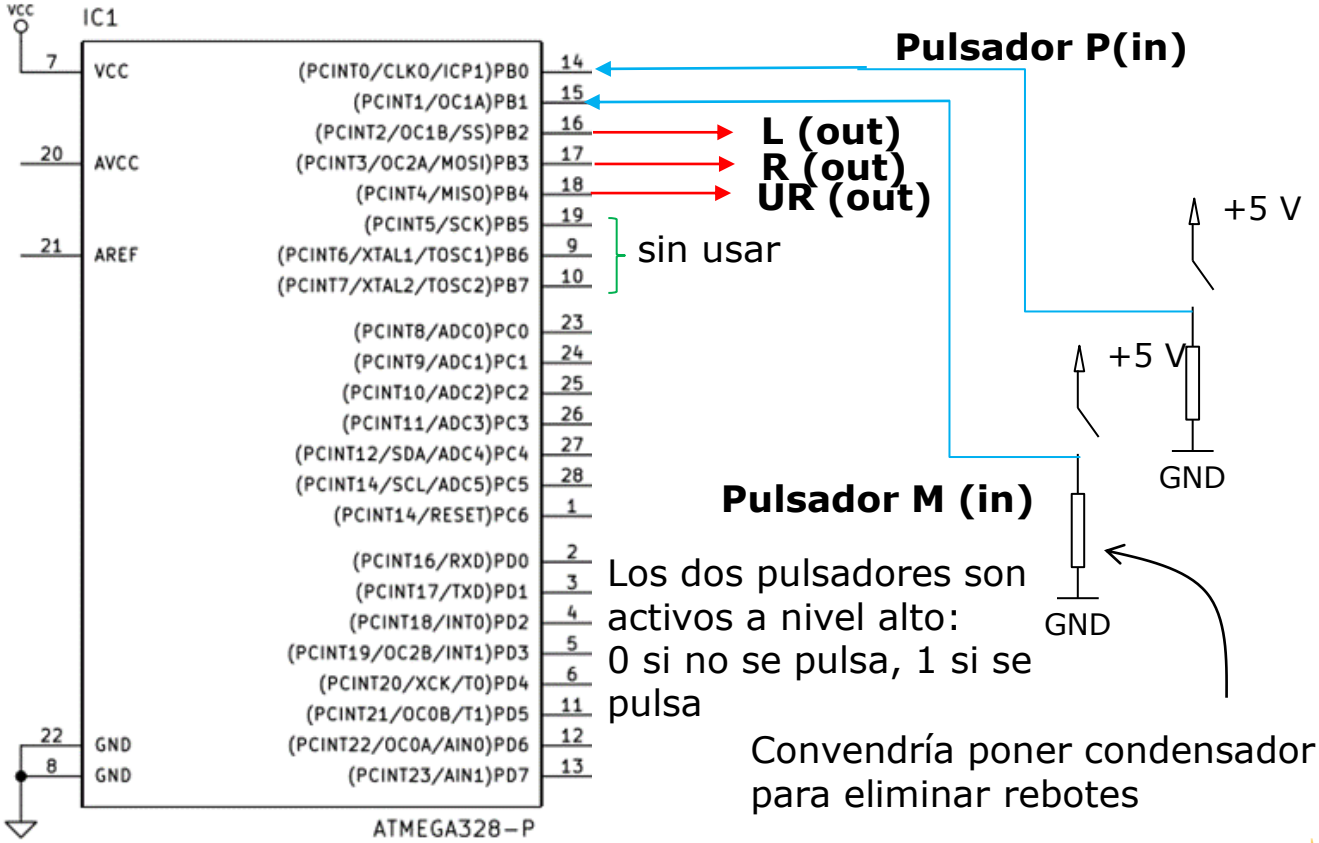
Entradas (pulsadores):
Parada
Motor



Salidas:
Lenta
Rápida
Ultra**R**ápida

(NOTA: son salidas desde el punto de vista de la máquina de estados finitos, el micro, y son entradas para el motor)

Problema 8: campana extractora



Los dos pulsadores son activos a nivel alto: 0 si no se pulsa, 1 si se pulsa

Convendría poner condensador para eliminar rebotes

Problema 8: Campana extractora

```
/* Control de una campana extractora */
#include <Arduino.h>

#define P (0) // Números de bit de las entradas
#define M (1)
#define L (2) // y salidas
#define R (3)
#define UR (4)

int main(){
  uint8_t p, m, l, r, ur;
  //variables para detectar los flancos en los pulsadores
  uint8_t p_ant, p_act, m_ant, m_act;
  // Estados
  enum {Parada, Lenta, Rapida, UltraRapida} estado;

  DDRB = 0x1C; // PB7-PB5 No usados (In), PB4-PB2 Out, PB1-PB0 In
  PORTB = 0xE0; // R de pull-up ON en PB7-PB5. Resto a 0
```

fijaros que en este ejemplo prescindimos de declarar una variable llamada "entradas"

00011100

11100000

Problema 8: Campana extractora

```
// Se inicializan las señales para los detectores de flanco
p_ant = (PINB >> P) & 1;
m_ant = (PINB >> M) & 1;
// Se inicializa la variable estado al estado inicial
estado = Parada;
while(1){
  p_act = (PINB >> P) & 1; // Se leen las entradas
  m_act = (PINB >> M) & 1;
  // Se detectan flancos de subida
  if ((p_act != p_ant) && (p_act == 1)){
    p = 1;
  }else{
    p = 0;
  }
  if ((m_act != m_ant) && (m_act == 1)){
    m = 1;
  }else{
    m = 0;
  }
  p_ant = p_act;
  m_ant = m_act;
```

111XXXPB1PB0 & 00000001=0000000PB0

0111XXXPB1 & 00000001=0000000PB1

leemos las entradas (PINB) sin necesidad de una variable "entradas"

0000000PB0 (la nueva lectura de PB0)

0000000PB1 (la nueva lectura de PB1)

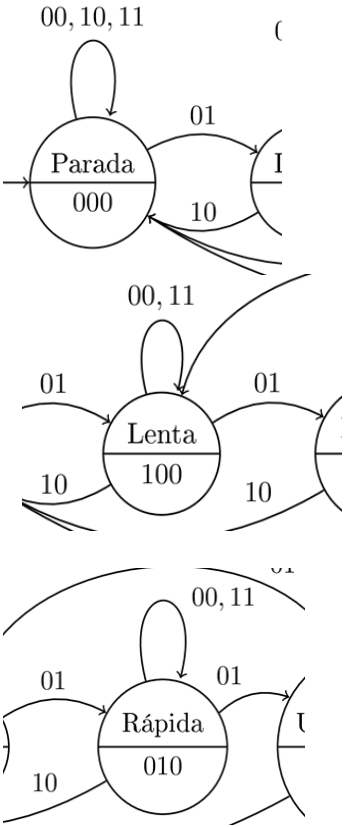
Se ha pulsado P

Se ha pulsado M

actualizamos las variables, lo que era una lectura actual pasa a ser una lectura antigua

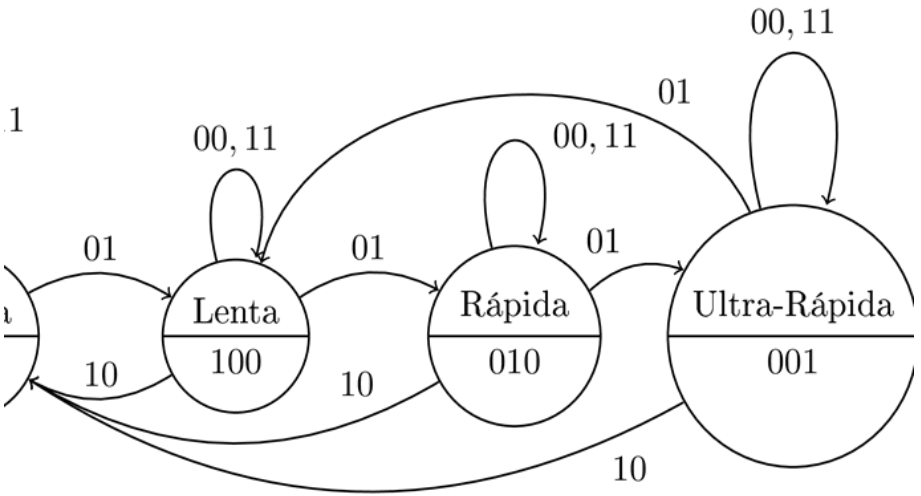
Problema 8: Campana extractora

```
// Transición de estados (función delta)
switch (estado){
case Parada:
  if ((p == 0) && (m == 1)){ // Puesta en marcha
    estado = Lenta;
  }
  break;
case Lenta:
  if ((p == 0) && (m == 1)){ // Aumento de velocidad
    estado = Rapida;
  }else if ((p == 1) && (m == 0)){// Parada
    estado = Parada;
  }
  break;
case Rapida:
  if ((p == 0) && (m == 1)){ // Aumento de velocidad
    estado = UltraRapida;
  }else if ((p == 1) && (m == 0)){// Parada
    estado = Parada;
  }
  break;
```



Problema 8: Campana extractora

```
case UltraRapida:
  if ((p == 0) && (m == 1)){ // "Aumento" de velocidad. Se pasa a vel. Lenta
    estado = Lenta;
  }else if ((p == 1) && (m == 0)){// Parada
     estado = Parada;
  }
  break;
}
```



Problema 8: Campana extractora

```
// Función Lambda
    if (estado == Lenta){
      l  = 1;
      r  = 0;
      ur = 0;
    }else if(estado == Rapida){
      l  = 0;
      r  = 1;
      ur = 0;
    }else if(estado == UltraRapida){
      l  = 0;
      r  = 0;
      ur = 1;
    }else{ // Estamos en el estado Parada
      l  = 0;
      r  = 0;
      ur = 0;
    }
```


Problema 8: Campana extractora

```
    // Se actualizan los bits de salida
    if( l == 1 ){
      PORTB |=  (1 << L); // Pone a 1 el bit
    }else{
      PORTB &= ~(1 << L); // Pone a 0 el bit
    }
    if( r == 1 ){
      PORTB |=  (1 << R); // Pone a 1 el bit
    }else{
      PORTB &= ~(1 << R); // Pone a 0 el bit
    }
    if( ur == 1 ){
      PORTB |=  (1 << UR); // Pone a 1 el bit
    }else{
      PORTB &= ~(1 << UR); // Pone a 0 el bit
    }
  }
}
```


UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

Anexo para Arduino

Comandos de control de los puertos de Arduino mediante IDE Arduino

- pinMode. Controla si el pin es de entrada, salida , o pull up
  - pinMode (pin, modo)
  - Pin es el pin de la placa de Arduino (no del ATMega)
  - Mode puede ser INPUT, INPUT_PULLUP, OUTPUT
- digitalWrite. Controla el estado del pin (L, H)
  - digitalWrite (pin, modo)
  - pin es el pin de la placa de Arduino (no del ATMega)
  - Mode puede ser LOW, HIGH
- analogWrite. Controla la tensión en el pin (modulación PWM)
  - analogWrite (pin, pwmvalue)
  - pin es el pin de la placa de Arduino (no del ATMega)
  - pwmvalue va de 0 a 255 (8 bits) indica la modulación PWM desde 0% (pwmvalue=0) hasta 100% (pwmvalue=255)
- digitalRead. Lee el bit en el pin indicado
  - digitalRead (pin)
- analogRead. Lee el valor analógico en el pin indicado
  - analogRead (pin)
  - El valor se entrega como un número entero entre 0 (0 V) y 1023 (5 V) (precisión de 10 bits)

Otros detalles sobre los pines de Arduino

- La corriente máxima de salida por pin es 40 mA
- La corriente agregada máxima de salida de todos los pines es 200 mA
- Si un pin no se configura, por defecto está en modo INPUT
- Los pines de entrada analógicos no es necesario declararlos como INPUT

Operadores aritméticos

OPERADOR	SIGNIFICADO
=	Asignación
+	Suma
-	Resta
*	Multiplicación
/	División (truncada cuando manejamos tipo int)
%	Resto de la división (con tipo int)
pow	Potencia. Pow(x,y) = x^y (hay que incluir Math.h)
sqrt	Raiz cuadrada. (hay que incluir Math.h)

Operadores lógicos de comparación (resultado Verdadero / Falso)

OPERADOR	SIGNIFICADO
<	Menor que
>	Mayor que
<=	Menor o igual
>=	Mayor o igual
==	Igual que
!=	No igual que

- TRUE, HIGH y 1 son equivalentes
- FALSE, LOW y 0 son equivalentes

Operadores Booleanos (resultado Verdadero / Falso)

OPERADOR	SIGNIFICADO
&&	AND
||	OR
^	XOR
!	NOT (el ! se pone antes)

Operadores Booleanos bit a bit

- Afectan a los operandos bit a bit, dentro de los n bytes que contiene cada operando

OPERADOR	SIGNIFICADO
&	AND bit a bit
|	OR bit a bit
^	XOR bit a bit
~	Complemento a 1 bit a bit (el ~ se pone antes)
<<	desplazar bits a la izq. (p.e. variable<<3)
>>	desplazar bits a la der.. (p.e. variable>>3)

Asignaciones compuestas

OPERADOR	SIGNIFICADO
x ++	x = x + 1
x --	x = x - 1
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y

Comandos if, for, while

```
if( condición)
  {
   sentencias;
  }
```

```
if( condición)
  {
   sentencias;
  } else {
    otras sentencias;
          }
```

```
for (int indice; condicion_basada_en_i; i++)
  {
    sentencias;
  }
```

```
while(condicion)
  {
    sentencias;
    sentencia_que_varia_condicion;
  }
```

```
//igual que while, pero asegura que al menos el bucle
//se ejecuta una vez
do
{
sentencias;
 {
while(condicion)
```

Comandos propios de Arduino. Tiempo y registros

- delay (retardo). Retardo expresado como un nº decimal en milisegundos. Durante ese retardo el micro no hace nada, ni leer puertos o pines
- delayMicroseconds (retardo en microsegundos).
- millis(). Devuelve el tiempo en milisegundos desde el arranque del programa. El resultado debe asignarse a una variable
  - p.e. tiempo = millis()
  - milis() se almacena en un registro de 32 bits, capacidad para 49 días y 17 horas
- pulseIn(pin, valor). Mide la duración en milisegundos de un pulso que entra por el pin indicado. valor puede ser HIGH (para medir un pulso positivo) o LOW.
- shiftOut(dataPin, clockPin, bitOrder, value). Manda un dato a un registro de desplazamiento externo. El dato sale por dataPin, el reloj por clockPin, bitOrder puede ser LSBFIRST o MSBFIRST, y value es el dato.
  - Ejemplo: shiftOut(dataPin, clockPin, LSBFIRST, 255);
- tone (pin, frec_en_Hez, duración_opcional). Genera un tono en un pin digital, mediante una onda cuadrada. Si no se indica duración, se genera de forma continua sin parar
- NoTone(pin). Cesa la generación del tono

Funciones propias de Arduino. Matemáticas

- min (x, y). Devuelve el valor mínimo de los dos
- max (x, y). Devuelve el valor máximo de los dos
- abs(x). Devuelve el valor absoluto de x
- constrain(x, a, b). Devuelve x si está entre a y b, si está por debajo de a devuelve a, y si está por encima de b devuelve b
- map(value, fromLow, fromHigh, toLow, toHigh). Mapea un valor que está entre fromLow – fromHigh al rango que va de toLow – toHigh
  - Muy útil para mapear lecturas de los puertos analógicos (10 bits) a las salidas analógicas (8 bits)
  - Ejemplo: val = map(analogRead(0),0,1023,100, 200); /* mapea el valor de analog 0 a un valor entre 100 y 200 */
- pow(base, exponent)
- sqrt(x)
- sin(rad), cos(rad), tan(rad)
- log(x)

Funciones propias de Arduino. Nº aleatorios

- randomSeed(seed) . Resetea el generador de números aleatorios con un nuevo número “semilla”.
  - Si alguna de las entradas analógicas no está conectada, puede su valor de tensión flotante para generar esta semilla. randomSeed(analogRead(5));
  - O podemos fijar seed: randomSeed(8);
- random (min, max). random (max). Genera un nº aleatorio entre min y max. Si min no se especifica, por defecto es 0.
  - Ejemplo: randNumber = random(10, 20);

Funciones propias de Arduino. Manipulación de bits

- bitRead(variable_a_leer, posición)
  - Entrega el bit que está en una determinada posición en una variable
  - La posición comienza en 0 (bit menos significativo) y acaba en 15 (para variables de 16 bits). Ejemplo:

  int x = 0b11001100 //0b indica que es un nº binario (0x indicaría hexadecimal)

  int bit = bitRead (x, 1) // asigna 0 a bit, 0 es el 2º LSB de x

- bitWrite(variable_a_escribir, posición, valor_del_bit)
  - Escribe un bit en una determinada posición en una variable
  - La posición comienza en 0 (bit menos significativo) y acaba en 15 (para variables de 16 bits). Ejemplo:

  int x = 0b11001000

  int bit = bitWrite (x, 2, 1) // asigna 1 al bit 2, ahora queda x = 11001100

Declaración y uso de una función en Arduino

```
void setup() {

}

void loop() {

  funcion (parametro1, parametro2);
  /*podemos usar nombres diferentes a los usados en la declaración como argumentos
  de la función*/
}

/*a continuación la declaración y definición de la función*/
void funcion (int param1, int param2) /* puede ser cualquier tipo y número de parámetros*/
{
    sentencias;
    /* cualquier parámetro usado en la definición de la función es una variable local*/
}
```

Interrupciones en Arduino

```
const int interruptPin = 2;
const int ledPin = 13;
int period = 500;

void setup()
{
 pinMode(ledPin, OUTPUT);
 pinMode(interruptPin, INPUT_PULLUP);
 attachInterrupt(0, goFast, FALLING);
}

void loop()
{
 digitalWrite(ledPin, HIGH);
 delay(period);
 digitalWrite(ledPin, LOW);
 delay(period);
}

void goFast()
{
 period = 100;
}
```

- Hay 2 pines en Arduino para recibir interrupciones, el D2 y el D3
- attachInterrupt(pin, función, modo);
  - pin puede ser 0 (D2) o 1 (D3)
  - función indica la función a ejecutar cuando se detecte la interrupción
  - modo puede ser CHANGE (interrupción cuando hay un cambio en pin) , RISING (interrupción por paso 0 a 1), FALLING (por paso de 1 a 0)
- Cuando se detecta la interrupción, Arduino deja lo que estaba haciendo y ejecuta función, y luego vuelve a lo que hacía (en el ejemplo, cambia el periodo a 100)
- noInterrupts() deshabilita todas las interrupciones en D2,D3
- interrupts habilita de nuevo las interrupciones

Uso del monitor serie

```
//SOS en Morse usando matrices
const int ledPin = 13;
int duraciones[]={200,200,200,500,500,500,200,200,200};
/*definición de matriz con las duraciones de parpadeo,
 * 200 para parpadeo corto, 200-200-200 es punto-punto-punto,
 * una letra s, y lo mismo con 500 para la letra o*/

void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600); /*inicializa el monitor serie, se declara en el setup*/
}

void loop() {
  for (int i = 0; i<9; i++)
  {
    flash(duraciones[i]);
    /*encendemos el LED 200ms (punto) o 500 (raya)
    accedemos al contenido de la matriz como parámetro
    de flash*/
    Serial.println(duraciones[i]); /*muestra duraciones[i] en el monitor serie*/
   }
   delay(1000);
}
```

Inicialización y lectura del puerto serie

- Serial.begin. Inicialización, en la parte de setup, indicando la velocidad del puerto

```
void setup() {
Serial.begin(9600);
}
```

- Serial.read. Lectura, en la parte de loop, es necesario almacenar el resultado de la lectura en una variable

```
void loop() {
valor_de_la_lectura = Serial.read();
}
```

- Serial.available. En la parte de loop. Comprueba si hay datos en el puerto serie (el buffer máximo de Arduino es 64 bytes)
  - Toma un valor entero, con el número de bytes disponibles en el buffer del puerto serie

```
if (Serial.available() > 0) //comprueba si hay datos en el buffer
 {
valor_de_la_lectura = Serial.read();
}
```

Escritura en el puerto serie: Serial.print y Serial.println

- La diferencia entre ellos es que Serial.printl introduce un salto de línea al final de lo que se va a imprimir
- Formato general: **Serial.print** (dato, tipo de dato). Si no se indica tipo de dato se trata como texto
  - Tipo de dato es opcional, si no se indica se imprimirá como decimal
  - Tipo BIN
  - Tipo OCT
  - Tipo DEC
  - Tipo HEX
  - Tipo BYTE. Imprime el equivalente e
  - Si el dato es float, por defecto se imprimirá con 2 decimales, para cambiar por ejemplo para que imprima 3 se hace así: Serial.print (dato_float, 3)

```
Serial.print(78, BIN);   // manda el dato "1001110"
Serial.print(78, OCT);   // manda el dato "116"
Serial.print(78, DEC);   // manda el dato "78"
Serial.print(78, HEX);   // manda el dato "4E"
```

- Para imprimir cadenas de caracteres: Serial.print (“Hola mundo”)
- **Serial.println** (dato, tipo de dato) equivale a:
  - Serial.print (dato, tipo de dato);
  - Serial.print (“\n”);
- **Serial.flush()**. Vacia el buffer donde Arduino va almacenando los datos que entran por el puerto serie.

Uso de matrices

```
//SOS en Morse usando matrices
const int ledPin = 13;
int duraciones[]={200,200,200,500,500,500,200,200,200};
/*definición de matriz con las duraciones de parpadeo, 200 para parpadeo corto, 200-200-200 es
punto-punto-punto, una letra s, y lo mismo con 500 para la letra o*/

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  for (int i = 0; i<9; i++)
  {
    flash(duraciones[i]);
    /*encendemos el LED 200ms (punto) o 500 (raya)
    accedemos al contenido de la matriz como parámetro
    de flash*/
   }
   delay(1000);
}
 /*defino flash como una función que enciende el LED durante el tiempo deseado*/
 void flash (int retardo) {
  digitalWrite(ledPin, HIGH);
  delay(retardo);
  digitalWrite(ledPin, LOW);
  delay(retardo);
 }
```

Punteros y matrices

- Declaración del puntero: tipo *nombre_del_puntero;
  - Ejemplo. int ***ip**, ar[20]; //**declaramos un puntero llamado** ip y un array,
- Operador & (dirección)
  - ip = **&**ar[6] /*hacemos que el puntero apunte al elemento 6 del array (sería el 7º pues siempre se comienza a contar desde 0) */
- Operador * (contenido)
  - *ip = 0 /* hacemos que el elemento al que apunta el puntero, ar[6], sea igual a 0 */
- Podemos aplicar operadores aritméticos a los punteros

```
int *ip, ar[20];
ip = &ar[6];
ip++; //ahora ip apunta a &ar[7]
```

Cadenas de caracteres (trings) como matrices tipo char

- Un literal tipo cadena (string) se escribe entre dobles comillas, p.e. “Hello”
- Esta cadena es en realidad una matriz, donde cada elemento es de tipo char
- La cadena “Hello” equivale entonces a
  - **char palabra[] = “Hello”**
  - palabra[] es la matriz
  - = “Hello” es una forma alternativa de asignar valores a la matriz
  - Otra forma de asignación es: palabra []={‘H’,’e’,’l’,’l’,’o’} (con comillas simples)
  - Otra forma es palabra[0]=‘H’; palabra[1]=‘e’; etc…
  - La matriz contiene los caracteres: desde H (su byte en ASCII es 72)… hasta el carácter especial vacío \0 (en ASCII 0)
- Otra forma de definir la matriz es mediante un puntero:
  - char *palabra = “Hello”
  - *palabra* es la dirección del primer elemento tipo char de la matriz
  - Esto se debe a que *palabra* es equivalente a *&palabra[0]*
  - *Con *palabra = “Hello” inicializamos el contenido de la dirección (el valor a donde apunta el puntero) y los contenidos de las direcciones siguientes hasta inicializar la matriz completa*

Tablas de caracteres ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Variables globales, locales y estáticas

```
const int ledPin = 13; /*const indica que es una constante, podría sobrar, indica al compilador que esta variable no va a cambiar*/
int delayPeriod = 100; /* variable global */

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  static int count = 0; /* global estática, solo se ejecuta la primera vez que se pasa por loop*/
  int global = 1; /*global, local a nivel de loop, se ejecuta cada vez que pasamos por loop*/
  digitalWrite(ledPin, HIGH);
  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);
  count ++; /*al ser estática crece en 1 cada vez que pasamos por el loop*/
   if( count == 20 )
   {
     count = 0;
     delay(500);
   }
  global ++;
  if( global == 20 )
   {
     global = 0;/*esto nunca se ejecutará, global solo llegará a 2*/
     delay(8000);
   }
}
```

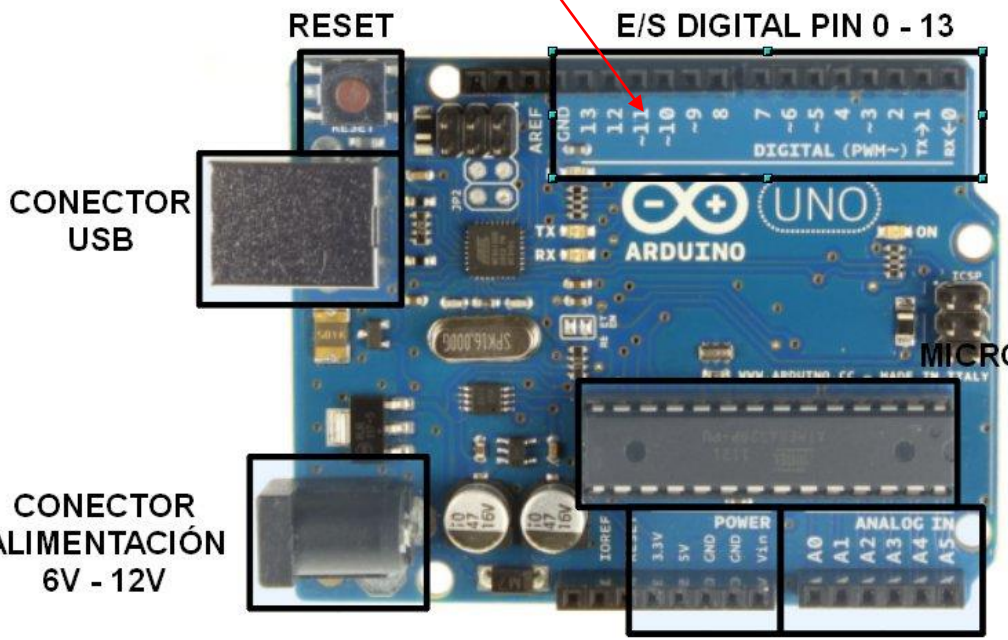
Constantes. Uso de const y define

- Dos formas equivalentes de definir una constante:
  - const float pi = 3.14159265;
  - # DEFINE pi = 3.14159265 (ojo, sin ;)
- Con # DEFINE se nombran las constantes antes de compilar el programa. Las constantes definidas de esta forma no ocupan memoria de programa, pues el compilador las sustituye por el valor definido al compilar

Patillaje del Arduino UNO

Los pines de este lado se denominan en el código Arduino con el nº de pin, p.e. 1

Los pines con un ~ pueden usarse como salidas analogWrite

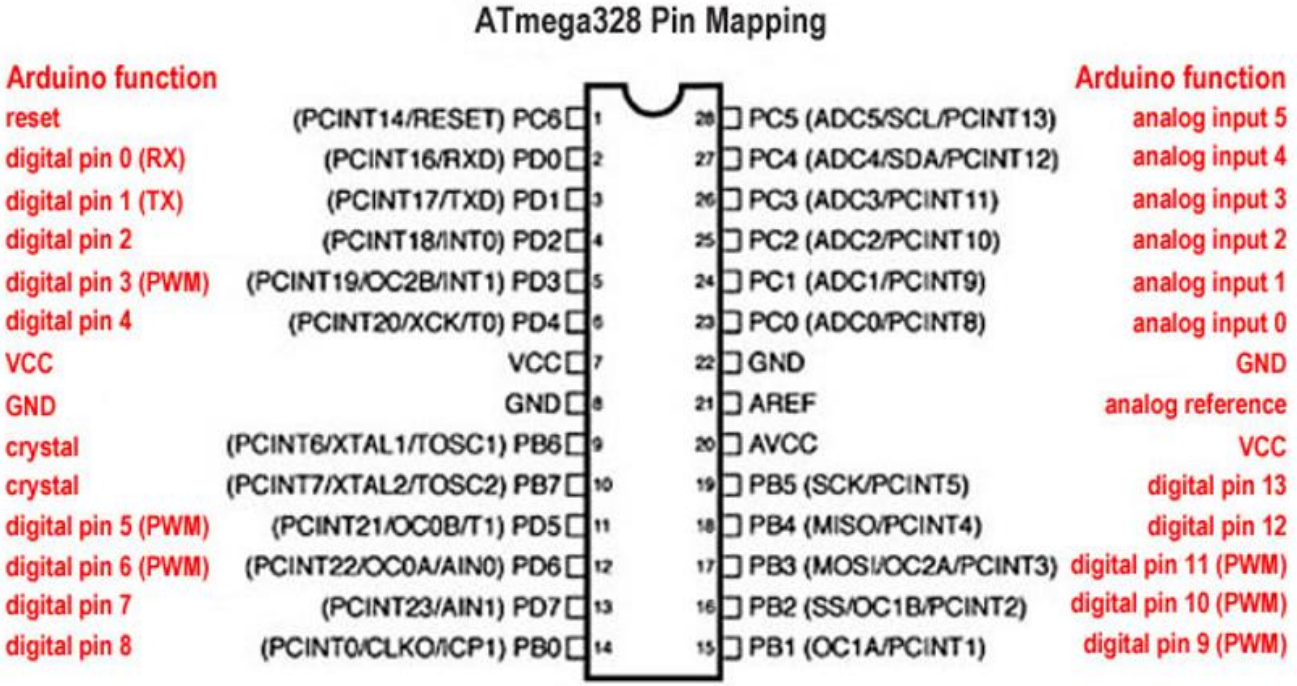


Los pines de este lado se denominan en el código Arduino con el nº de pin, p.e. 1, o como A1

Comentarios sobre los pines de la placa de Arduino uno

- Hay un pin “Reset”, que si se pone a 0 hace que el sketch en el micro comience desde la primera línea
- Los pines marcados como entradas analógicas se pueden configurar como entradas / salidas digitales (p.e. digitalRead(A1), digitalWrite(A1))
- Conviene evitar el uso de los pines digitales 0 y 1, pues se comparten con la comunicación serie USB

Correspondencia entre el patillaje de Arduino UNO y los pines de ATMega



Tipos en C para micros

- Definimos tipos para facilitar la portabilidad del SW

```
#ifndef TIPOS_H
#define TIPOS_H

typedef unsigned char      uint8_t;
typedef unsigned int       uint16_t;
typedef unsigned long int  uint32_t;

typedef signed char        int8_t;
typedef signed int         int16_t;
typedef signed long int    int32_t;

#endif
```

- Con los siguientes rangos

Bits	Rango sin signo	Rango con signo
8	$0 \leftrightarrow 255$	$-128 \leftrightarrow 127$
16	$0 \leftrightarrow 65\,535$	$-32\,768 \leftrightarrow 32\,767$
32	$0 \leftrightarrow 4\,294\,967\,296$	$-2\,147\,483\,648 \leftrightarrow 2\,147\,483\,647$

Tipos generales y detalle para Arduino

TIPO	MEMORIA (BYTES)	RANGO	COMENTARIOS
boolean	1	verdadero / Falso	
char	1	-128 a 127	caracteres ASCII
byte	1	0 a 255	
int	2	-32.768 a 32.767	
unsigned int	2	0 a 65.535	
long	4	-2.147.483.648 a 2.147.483.647	
unsigned long	4	0 a 4.294.967.295	
float	4	-3.4028235 E+38 a 3.4028235 E+38	Escribir los nº con un . (p.e. 3.0) para asegurar que son tratados como float
double	4	-3.4028235 E+38 a 3.4028235 E+38	Normalmente debería ser 8 bytes, pero en Arduino el mismo rango que float