Unidad III
Microcontroladores

David A. Trejo Pizzo

Departamento de sistemas

dtrejopizzo@gmail.com

Marzo, 2015

Estructura

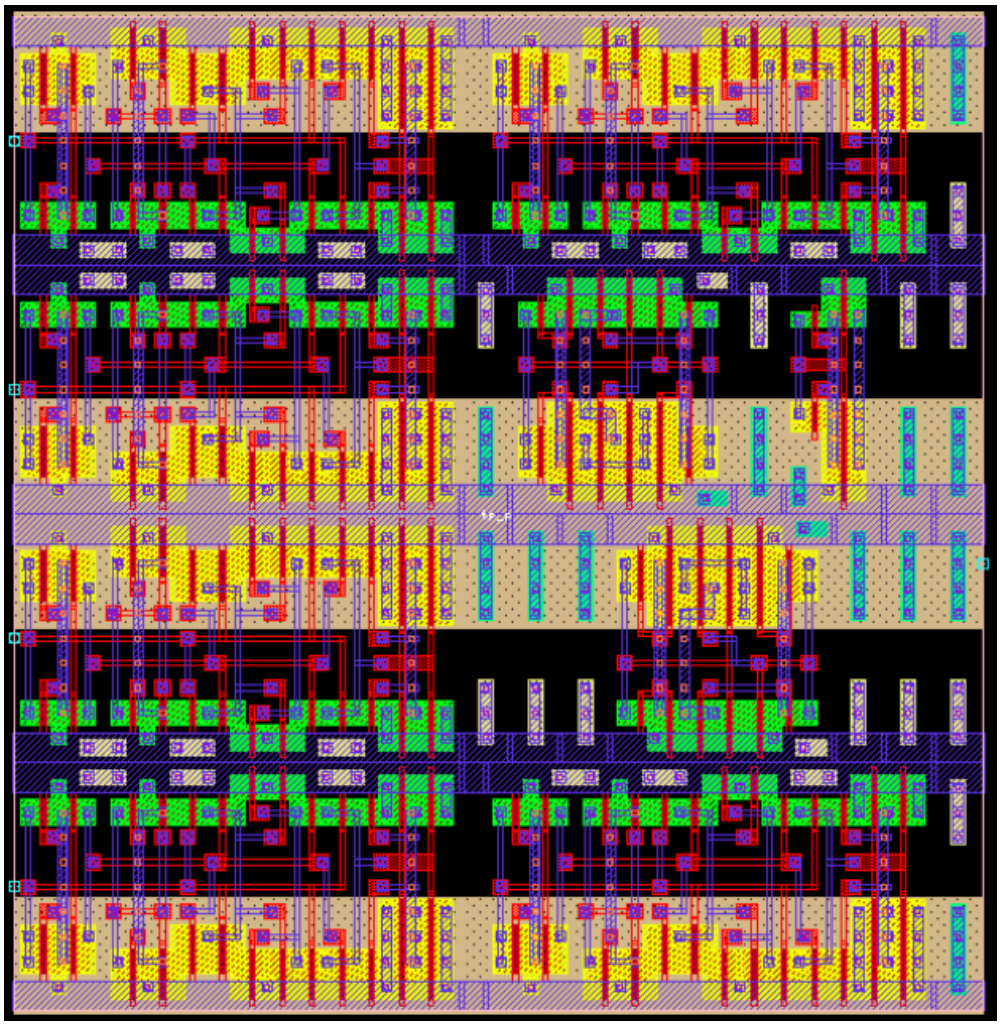
Que es un microcontrolador?

- Un microcontrolador es un circuito integrado que incorpora en su interior un procesador, memorias y una serie de periféricos que lo hacen apto para desempeñar distintas funciones.
- Su funcionamiento depende del programa que tenga cargado en su memoria.

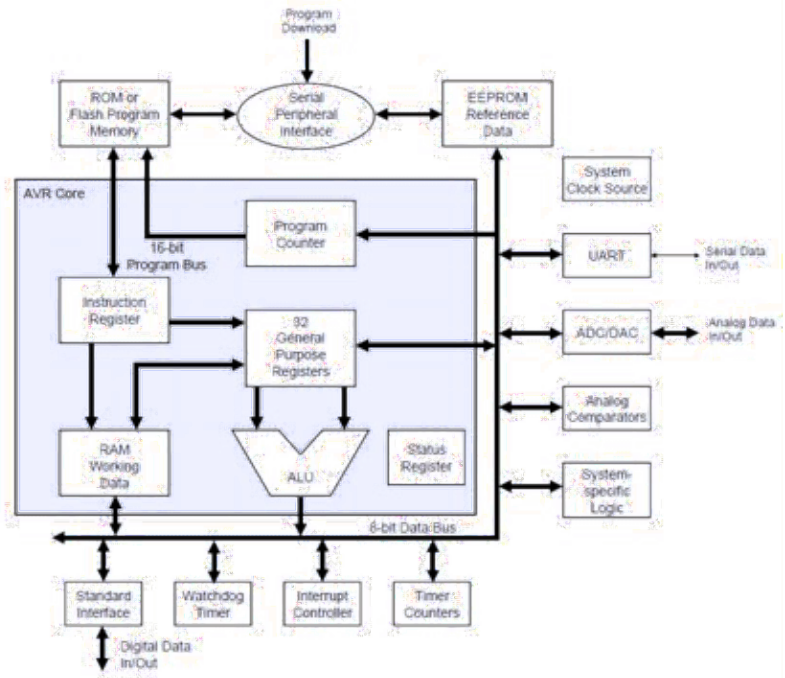
Componentes

- Unidad de control
- Conjunto de registros
- Unidad aritmético y lógica
- Buses de conexión
- Unidad de cálculo de coma flotante

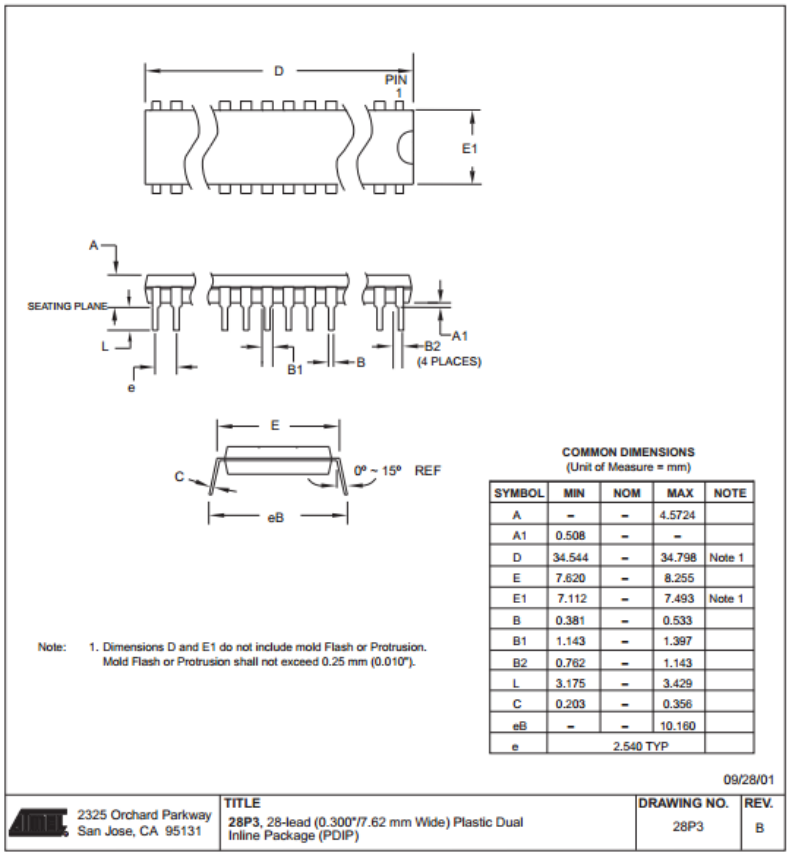
Que es un microcontrolador?



Core


Program Download
ROM or Flash Program Memory
Serial Peripheral Interface
EEPROM Reference Data
System Clock Source
AVR Core
Program Counter
16-bit Program Bus
UART
Serial Data In/Out
Instruction Register
32 General Purpose Registers
ADC/DAC
Analog Data In/Out
Analog Comparators
RAM Working Data
ALU
Status Register
System-specific Logic
8-bit Data Bus
Standard Interface
Watchdog Timer
Interrupt Controller
Timer Counters
Digital Data In/Out

Implementacion fisica



COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	4.5724	
A1	0.508	–	–	
D	34.544	–	34.798	Note 1
E	7.620	–	8.255	
E1	7.112	–	7.493	Note 1
B	0.381	–	0.533	
B1	1.143	–	1.397	
B2	0.762	–	1.143	
L	3.175	–	3.429	
C	0.203	–	0.356	
eB	–	–	10.160	
e	2.540 TYP			

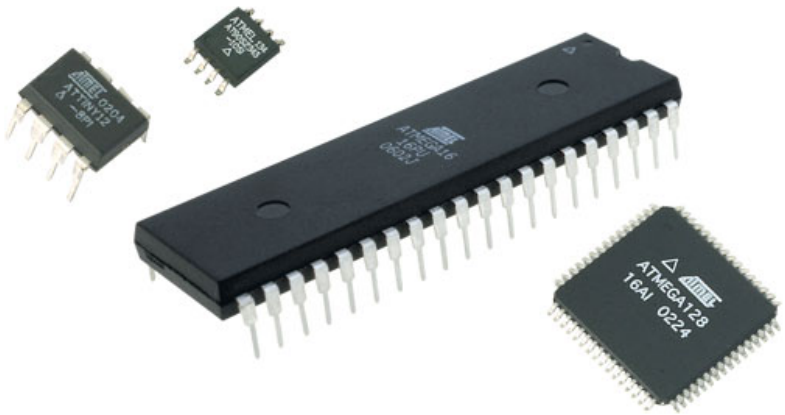
Note: 1. Dimensions D and E1 do not include mold Flash or Protrusion. Mold Flash or Protrusion shall not exceed 0.25 mm (0.010").

09/28/01

2325 Orchard Parkway San Jose, CA 95131	TITLE	DRAWING NO.	REV.
	28P3, 28-lead (0.300"/7.62 mm Wide) Plastic Dual Inline Package (PDIP)	28P3	B

AVR

El AVR fue diseñado desde un comienzo para la ejecución eficiente en lenguaje C.
La familia de microcontroladores es bastante numerosa, con más de 70 dispositivos y variantes que poseen el mismo núcleo; solo varían la cantidad de memoria y los periféricos.



Periféricos

- Puertos de Entradas / Salidas
- Timers
- Generador de interrupciones
- Interfaces de comunicación
- Conversor Analógico / Digital
- Conversor Digital / Analógico

Memoria y periféricos

El tipo y cantidad de periféricos, va a depender de cada fabricante.
Lo mismo el tamaño y tipo de las memorias.
Se debe elegir al microcontrolador de acuerdo a las necesidades del proyecto a implementar.
Los fabricantes ofrecen cuadros comparativos entre sus distintas familias, los cuales permiten ver de forma detallada las distintas características de cada producto.

Arquitectura RISC

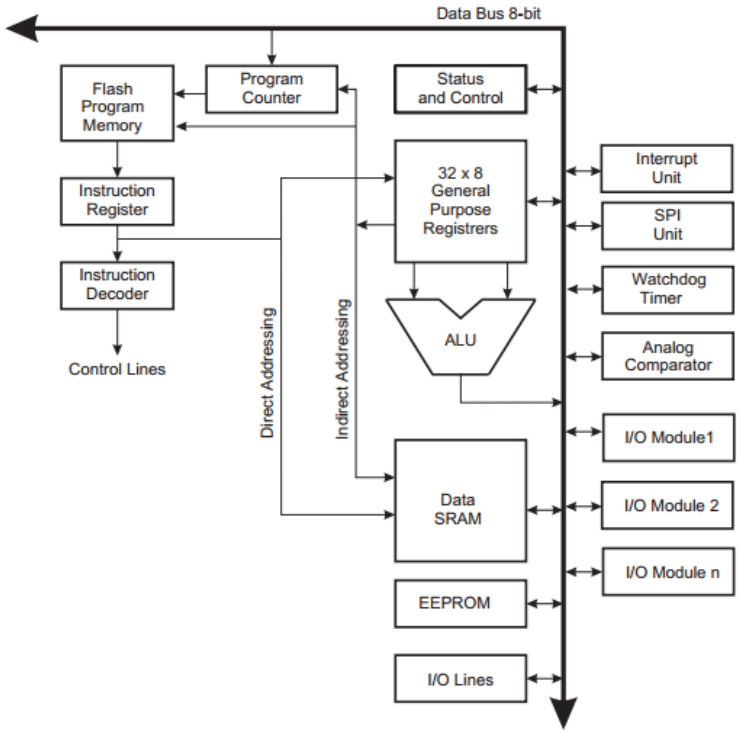
Reduced COMPLEXITY Instruction Set Computer.

No significa un número de instrucciones reducido como a menudo se piensa. Significa que se reduce la complejidad de los circuitos encargados de la decodificación de las instrucciones , haciéndolos más simples y eficientes.

ARV RISC

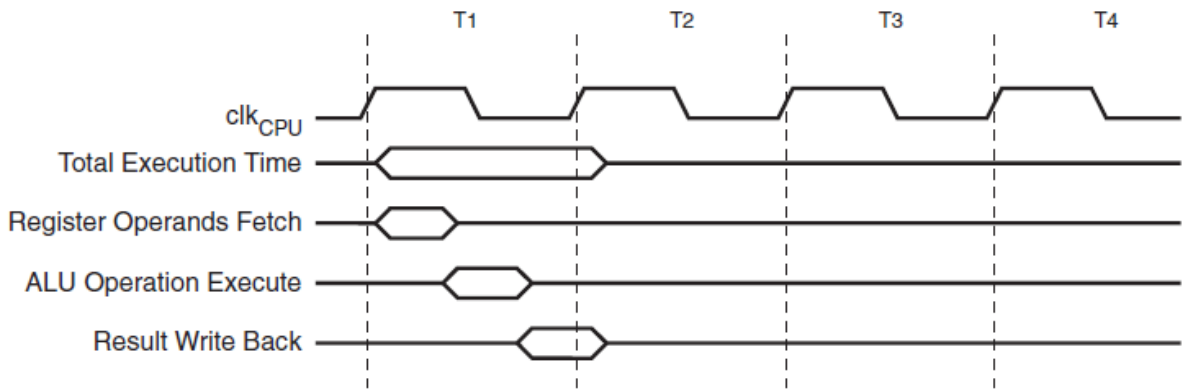
- Utiliza instrucciones de tamaño fijo. Solo 4 instrucciones de 32 bits y el resto son de 16 bits
- Posee 32 registros de uso general
- Solo utiliza instrucciones de Load/Store para acceder a la memoria RAM.
- La mayoría de las instrucciones se ejecutan en 1 ciclo de clock.
- Esta diseñada especialmente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

Arquitectura AVR



Ciclos

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



Bus de comunicaciones

- La arquitectura de un microcontrolador no solo está definida por su set de instrucciones, sino también por la arquitectura del bus de comunicaciones.
- Arquitectura de Von Newmann: los buses de acceso a la memoria de programa y a la memoria de datos son los mismos, resultando así un único mapa de memoria.
- Arquitectura Harvard: se utilizan buses separados para la memoria de programa y la de datos.

Los microcontroladores AVR poseen una Arquitectura Harvard.

El lenguaje C

El lenguaje C modificado

Operadores

Aritmeticos

- = (asignar)
- + (suma)
- - (resta)
- * (multriplicacion)
- / (division)
- % (modulo)

Comparacion

- == (igual a)
- != (distinto a)
- < (menor que)
- > (mayor que)
- <= (menor o igual que)
- >= (mayor o igual que)

Boleanos

- && (y)
- || (o)
- ! (no)

Otros operadores

Compuestos

- $++$ (incremento)
- $--$ (decrementar)

Punteros

- * desreferencia
- & referencia

Bit a bit

- & (y)
- $|$ (o)
- $\wedge$ (xor)
- $\sim$ (no)
- $<<$ (bit shift a la izquierda)
- $>>$ (bit shift a la derecha)

Tipos de datos

- void
- booleano
- char
- unsigned char
- byte
- int
- unsigned int
- word
- long
- unsigned long
- short
- float
- double
- string (char[])
- String (objeto)
- array

Sentencia IF

- Si $X \longrightarrow Y$

```
if (X) {
    // Y}
```

- Si $X \longrightarrow Y \vee Z$

```
if (X) {
    // Y}
else {
    // Z}
```

- Si $W \longrightarrow X \vee Y \longrightarrow Z$

```
if (W) {
    // X}
else if (Y) {
    // Z}
```

Switch

La estructura **Switch** funciona como un menu. Cuando se encuentra una sentencia **case** cuyo valor coincide con el de la variable, se ejecuta el código en esa declaración de caso.

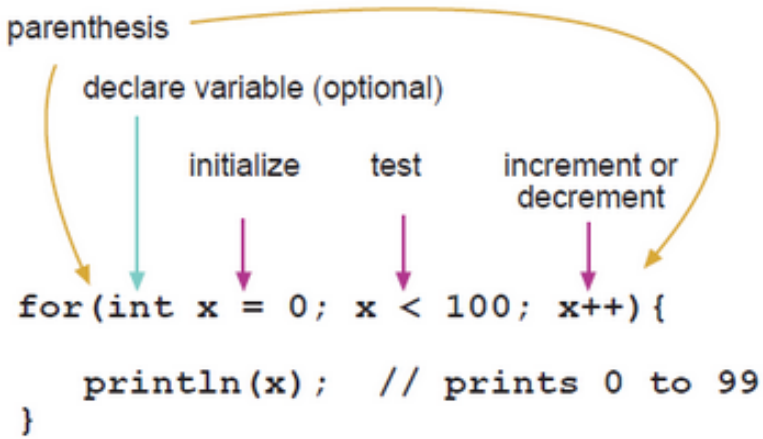
Sintaxis:

```
switch (var) {
  case label:
    // statements
    break;
  case label:
    // statements
    break;
  default:
    // statements
}
```

Ejemplo:

```
switch (var) {
    case 1:
      //do
      break;
    case 2:
      //do
      break;
    default:
  }
```

Sentencia FOR




```
void loop()
{
    for (int i=0; i <= 255; i++)
    {
        analogWrite(10, i);
        delay(10);
    }
}
```

While

El ciclo **While** se repetirá de forma continua hasta que la expresión dentro del paréntesis sea falsa.

Sintaxis:

```
while(expresion){
  // tarea(s)
}
```

Ejemplo:

```
var = 0;
while(var < 200){
  var++;
}
```

Do while

El ciclo **Do while** es similar al ciclo While, con la excepción de que la condición se comprueba al final del bucle, por lo que el bucle se ejecutará al menos una vez.

Sintaxis:

```
do
{
    // statement block
} while (test condition);
```

Ejemplo:

```
do
{
  delay(50);
  x = readSensors();
} while (x < 100);
```

Break

La declaración **Break** se usa para salir de un ciclo **do**, **for** o **while**, saltando la condicion normal del ciclo.

Ejemplo:

```
for (x = 0; x < 255; x ++)
{
    digitalWrite(PWMpin, x)
        ;
    sens = analogRead(0);
    if (sens > threshold){
        x = 0;
        break;
    }
    delay(50);
}
```

Continue

La declaración **Continue** se usa para salir de un ciclo **do**, **for** o **while**, saltando el resto del ciclo.

Ejemplo:

```
for (x = 0; x < 255; x ++)
{
    if (x > 40 && x < 120){
        continue;
    }
    digitalWrite(PWMpin, x)
        ;
    delay(50);
}
```

Return

La declaración **Return** se usa para terminar una función y retornar un valor a otra funcon o al programa principal.

Ejemplo:

```
int checkSensor(){
v = analogRead(0);
    if (v > 400) {
        return 1;
    else{
        return 0;
    }
}
```

Goto

La declaración **Goto** transfiere el flujo del programa a otro punto.

Ejemplo:

```
for(byte r = 0; r < 255; r++){
    for(byte g = 255; g > -1; g--){
        for(byte b = 0; b < 255; b++){
            if (analogRead(0) > 250){ goto bailout;}
            // otras declaraciones ...
        }
    }
}
bailout:
```

Sentencia FOR

```
int PWMpin = 10; // LED in series with 470 ohm resistor
     on pin 10

void setup()
{
  // no setup needed
}

void loop()
{
   for (int i=0; i <= 255; i++){
      analogWrite(PWMpin, i);
      delay(10);
   }
}
```

Declaraciones

Declaraciones

Declaraciones

El conversor analogico-digital

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

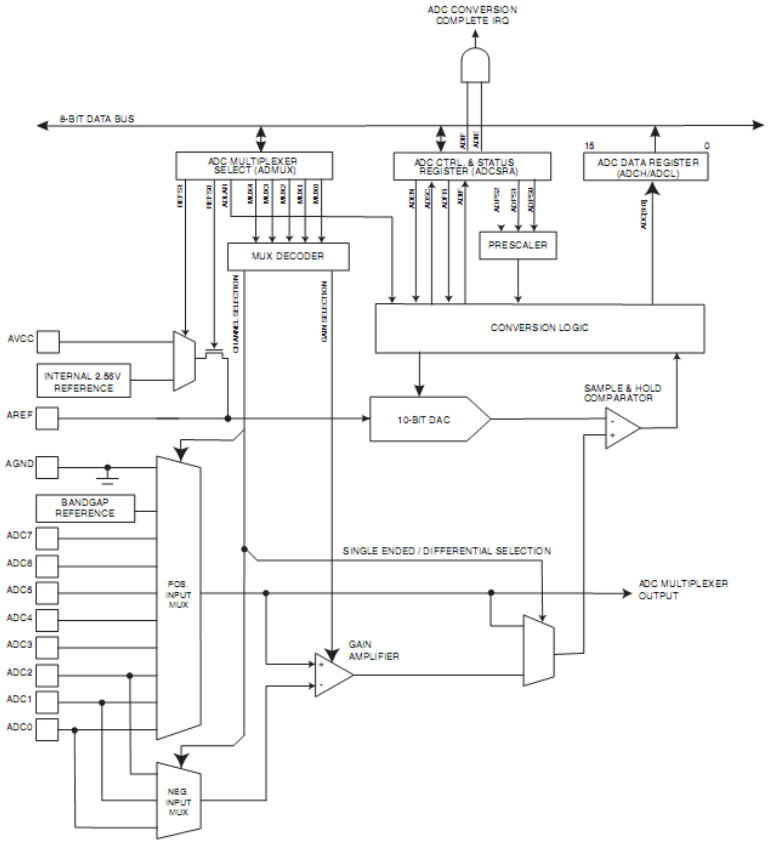
Tipos

- Simple o doble rampa.
- Tipo FLASH
- De aproximaciones sucesivas.
- VCO ADC
- Sigma Delta.

El ADC del ATmega328

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

Esquema interno



Conversión en Modo comun

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

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

Por ejemplo, si $V_{REF} = 2.56V$ y la entrada $V_{IN} = 1V$, el resultado de la conversión es 400.

Conversión en modo diferencial

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

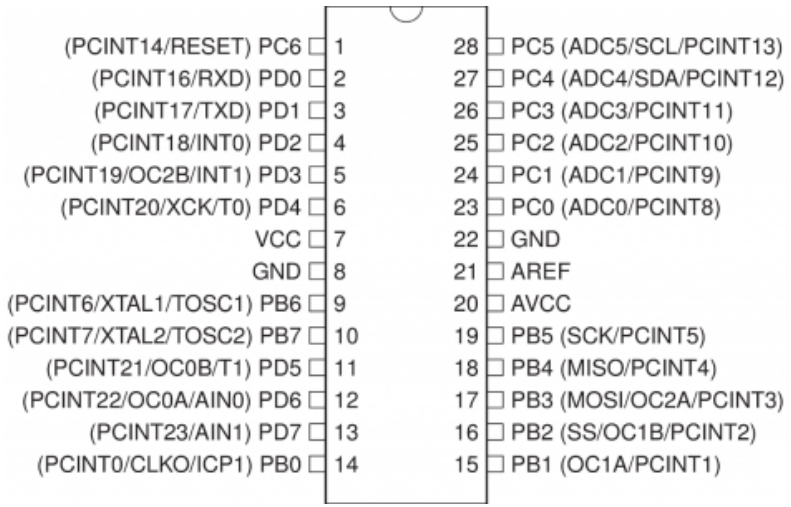
$$ADC = \frac{(V_{POS} - V_{NEG}) * GAIN * 512}{V_{REF}}$$

Es presentado en complemento a 2, desde -512 (ADCH=02h y ADCL=00h) hasta +511 (ADCH=01h y ADCL=FFh). La polaridad del resultado puede obtenerse leyendo el 2do LSB del registro ADCH.

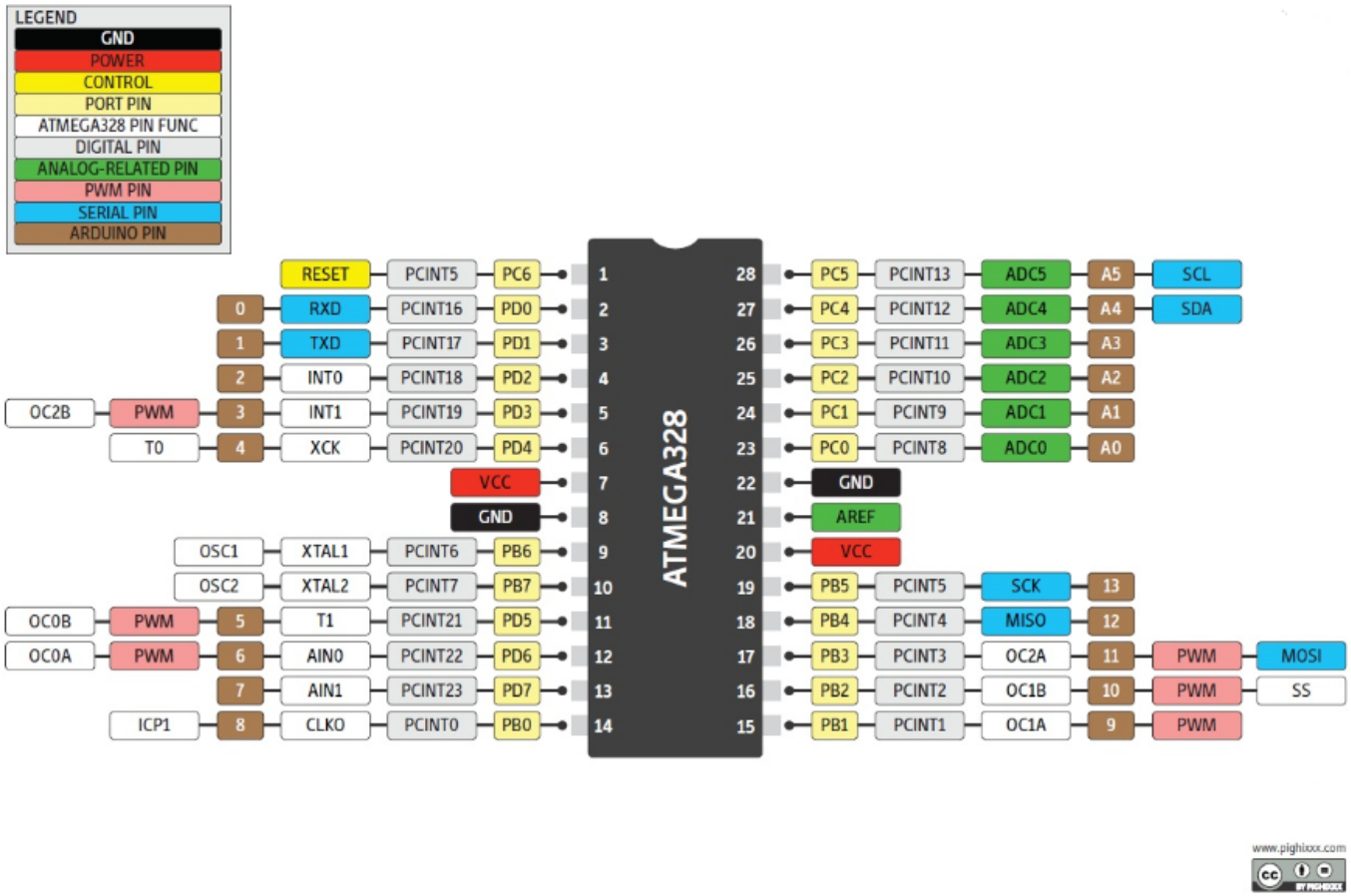
Modos de conversión

Modos de conversión

Pinout


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

Esquema de pines



Lectura analogica

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

void loop() {
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue);
}
```

Puertos analogicos

Puertos analogicos