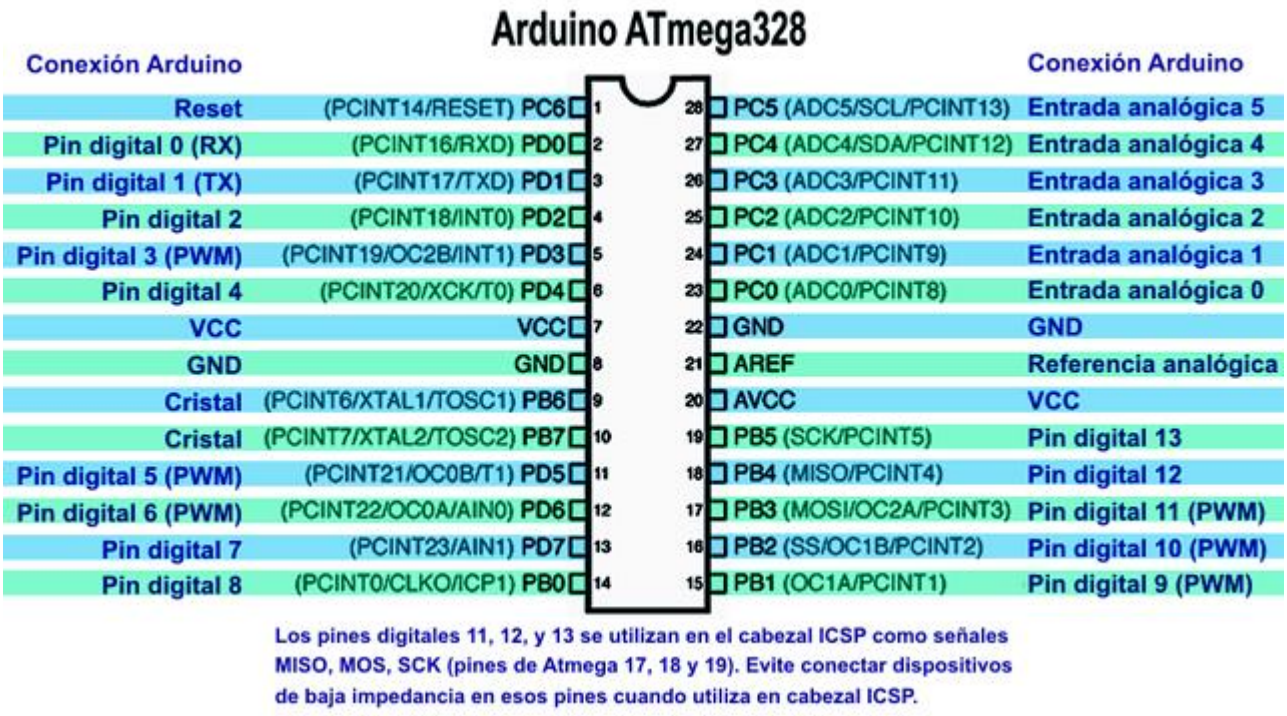
Arduino: Entradas y salidas – Manipulación de puertos

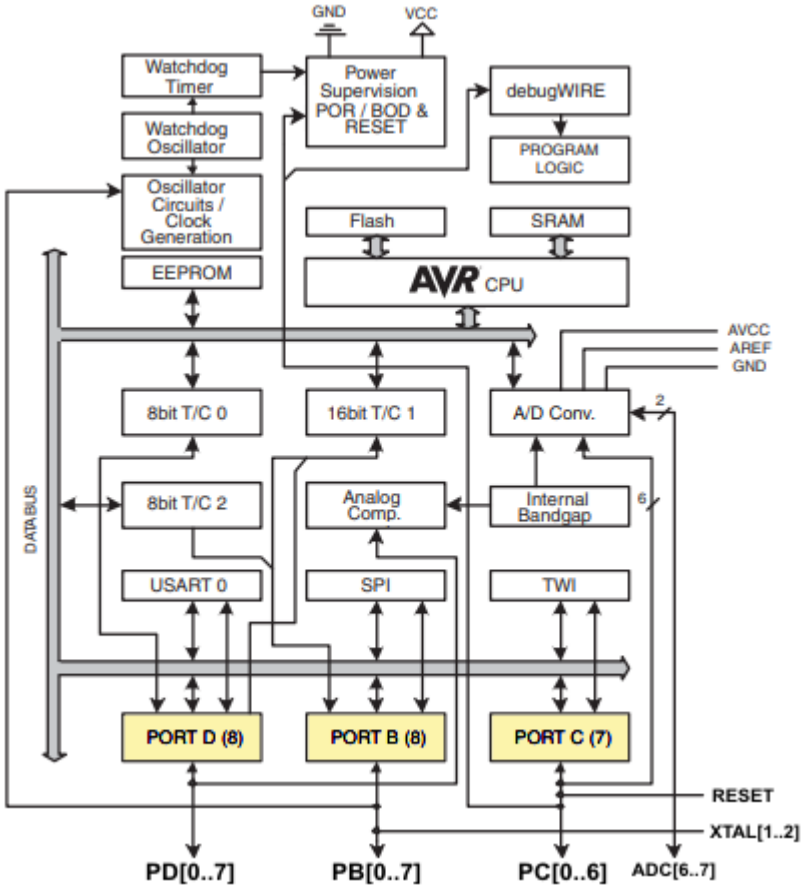
1 respuesta



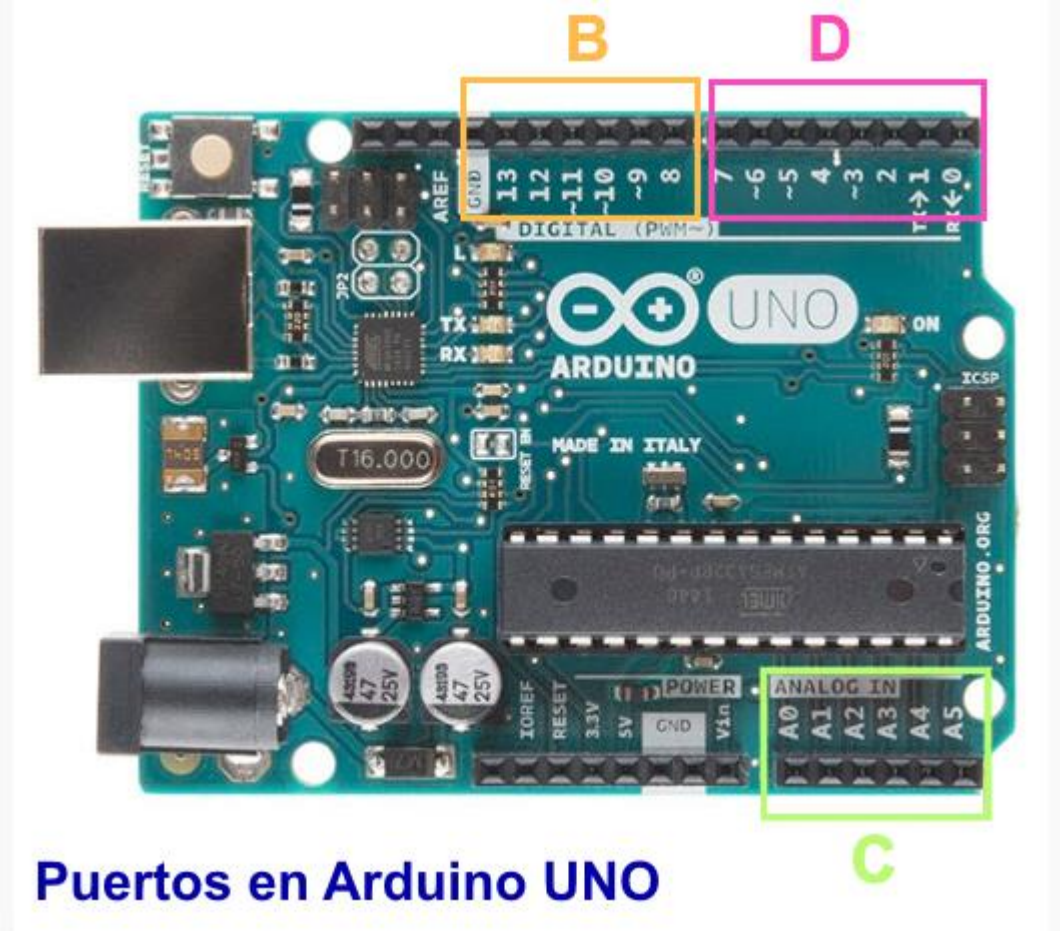
NOTA: *Para ver el uso de los pines del ATmega328P en Arduino recomiendo leer el artículo* ***Arduino UNO R3 – Conectándolo al mundo exterior****.*

Programación avanzada de puertos

En principio, es importante recordar que los puertos de un microcontrolador de 8 bits tienen esa misma cantidad de entradas/salidas, o sea ocho líneas. Esto nos haría pensar que el ATmega328P, que posee tres puertos (B, C y D), dispone de 3 x 8 = 24 líneas de entrada/salida disponibles. Sin embargo, utilizado en un Arduino no es así, como veremos.



La mayoría de los bits de los puertos de los microcontroladores son de uso múltiple, es decir que se comportan de una forma u otra de acuerdo con su configuración. Varias líneas de puertos cumplen funciones vitales en la operación de un Arduino, funciones que no son líneas de entrada/salida de uso general.



El **PORTB** (puerto B) tiene ocupadas dos líneas de entrada/salida que se utilizan para conectar el cristal oscilador. Estos pines, el **PORTB** bit-**6** y **PORTB** bit-**7**, pueden quedar libres si se configura al chip para utilizar el oscilador interno, pero esta opción no podemos utilizarla en el Arduino debido a que ya tiene su sistema basado en la velocidad de cristal de 16 MHz, además de que el cristal está soldado a esos pines en el circuito de la placa.

El **PORTC** tiene dos bits que no están disponibles, uno de ellos, el **PORTC** bit-**6** se utiliza como entrada de reinicio (**RESET**), y el otro bit (**7**) no está cableado hacia el exterior del ATmega328P con cápsula **PDIP** que viene enchufado en el zócalo del Arduino Uno R3, porque no posee suficientes líneas disponibles en su encapsulado de 28 patas. Y cuando se trata de un chip con encapsulado de montaje superficial **TQFP** de 32 pines (como en el **Arduino Nano** y en algunos clones de **Arduino Uno**), las dos líneas faltantes están dedicadas al convertidor analógico digital (**ADC6** y **ADC7**) y no son pines de entrada/salida digital.

Dos bits del **PORTD**, el **PORTD** bit-**0** y el **PORTD** bit-**1**, se utilizan durante la programación del Arduino, ya que están conectados a la interfaz **USB**, además de ser los pines **TX** y **RX** utilizados para la comunicación serie. Estos pines se pueden utilizar para comunicación serie asincrónica hacia el exterior, y también como entradas o salidas cuando no se está grabando un programa. Pero no deben tener conexiones instaladas mientras se programa el Arduino.

En consecuencia, no se llega a disponer de la cantidad de 24 entradas/salidas que ofrecerían tres puertos de 8 bits.

El ATmega328P, como cualquier otro microcontrolador, tiene registros para cada puerto con los cuales se define si cada bit del puerto será usado como entrada o como salida, y en varios casos otra función. El

ATmega328P tiene tres puertos: **PORTB**, **PORTC** y **PORTD**, por lo cual hay tres bancos de registros de configuración, uno para cada puerto.

BANCO	**2^7**	**2^6**	**2^5**	**2^4**	**2^3**	**2^2**	**2^1**	**2^0**
PORTB			Digital 13	Digital 12	Digital 11	Digital 10	Digital 9	Digital 8
PORTC			A5	A4	A3	A2	A1	A0
PORTD	Digital 7	Digital 6	Digital 5	Digital 4	Digital 3	Digital 2	Digital 1	Digital 0
VALOR	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**

Bancos y puertos de ATmega 328p

Registros

El ATmega328P tiene tres registros de 8 bits con los que administra estos tres puertos:

- **DDRx** (donde **x** es **B**, **C** o **D** en este caso) determina si un bit es entrada (fijándolo en **0**) o salida (fijándolo en **1**)
- **PORTx** controla si el pin está en nivel **ALTO** (HIGH) o **BAJO** (LOW). También define la existencia o no de un resistor de polarización a Vcc (**pull-up**, en inglés) si es una entrada.
- **PINx** permite leer el estado de los pines de un puerto (solo es para lectura)

PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0

Antes de entrar de lleno a explicar el uso de los registros, veamos primero para qué podría servir este esfuerzo.

Ventajas de usar registros:

- Cada instrucción de máquina necesita un ciclo de reloj a 16 MHz. Las funciones **digitalRead()** y **digitalWrite()** se componen cada una de ellas de varias instrucciones de máquina, lo que puede influir negativamente en aplicaciones muy dependientes del tiempo. El uso de los registros **PORTx** puede hacer el mismo trabajo en muchos menos ciclos de reloj.
- Si es necesario cambiar de estado varios pines simultáneamente en lugar de ir haciéndolo de a uno con un ciclo **for**, que tomaría mucho tiempo, es posibles escribir directamente al registro y establecer los valores de varios pines de una sola vez.
- Si el código está llegando al límite de memoria de programa disponible (memoria **Flash**), es posible usar este método para hacer que el código use menos bytes de programa.

Desventajas de usar registros:

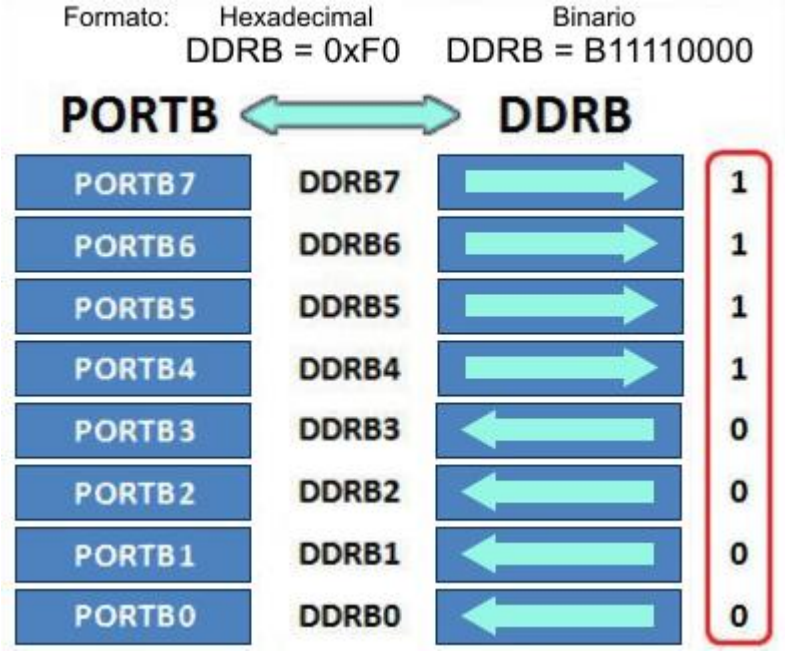
- El código no es fácil de entender para novatos.
- Es mucho mas fácil cometer errores de difícil depuración. Por ejemplo con **DDRD = B11111111** se pone a todos los pines, **D0** a **D7** como salida, inclyendo el pin **D0** (**RX**), lo que causará que el puerto serie deje de funcionar.

En las librerías es muy recomendable usar la manipulación directa de registros, de modo de hacerlas mucho más eficientes.

Registro DDRx – Definición de pines como entrada o salida

Al utilizar los registros **DDR** tenemos la ventaja de que con solo una instrucción podemos declarar el pin como entrada o salida, en cambio, utilizando **pinMode()** necesitaríamos 8 instrucciones.

Veamos un ejemplo con el registro DDRB del PORTB:



El ejemplo de la imagen define los 4 bits bajos o **menos significativos** (0 a 3) como entradas, y los 4 bits altos o **más significativos** (4 a 7) como salidas. Veamos cómo se verían este y otros ejemplos en el programa del IDE de Arduino.

Arduino

```
1 DDRB = B11110000;  // PORTB7-PORTB4 como salidas, PORTB0-PORTB3 como entradas.
```

```
2 DDRD = B11111111;  // PORTD0-PORTD7 como salidas.

3 DDRB = B00000000;  // PORTB0-PORTB7 como entradas.

4 DDRB = B00000111;  // PORTB0-PORTB2 salidas, PORTB3-PORTB7 como entradas.

5 PORTD = B11111111; // PORTD0-PORTD7 en ALTO (HIGH).

6 PORTD = B00000000; // PORTD0-PORTD7 en BAJO (LOW).

7 PORTB = B00000101; // PORTB0 y PORTB2 en ALTO (HIGH), PORTB1, PORTB3-PORTB7 en BAJO (LOW).

8 byte variable = PIND; // Guarda en una variable 'byte' el estado de PORTD0-PORTD7.
```

Mediante estos registros también podemos controlar las resistencias internas de pull-up que se usan, básicamente, para no dejar los pines de entrada al aire, ya que esto genera ruido eléctrico. Se puede resolver el problema de dos maneras: poner una resistencia externa de 10K a Vcc (+5V) o usar los pull-up internos del microcomputador, que polarizan de la misma forma las entradas y hacen más simple el circuito.

Para habilitar las resistencias pull-up, primero tenemos que configurar como entrada el puerto (con el bit 0), mediante registro DDRx, y luego escribir un 1 en el registro PORTx.

Arduino

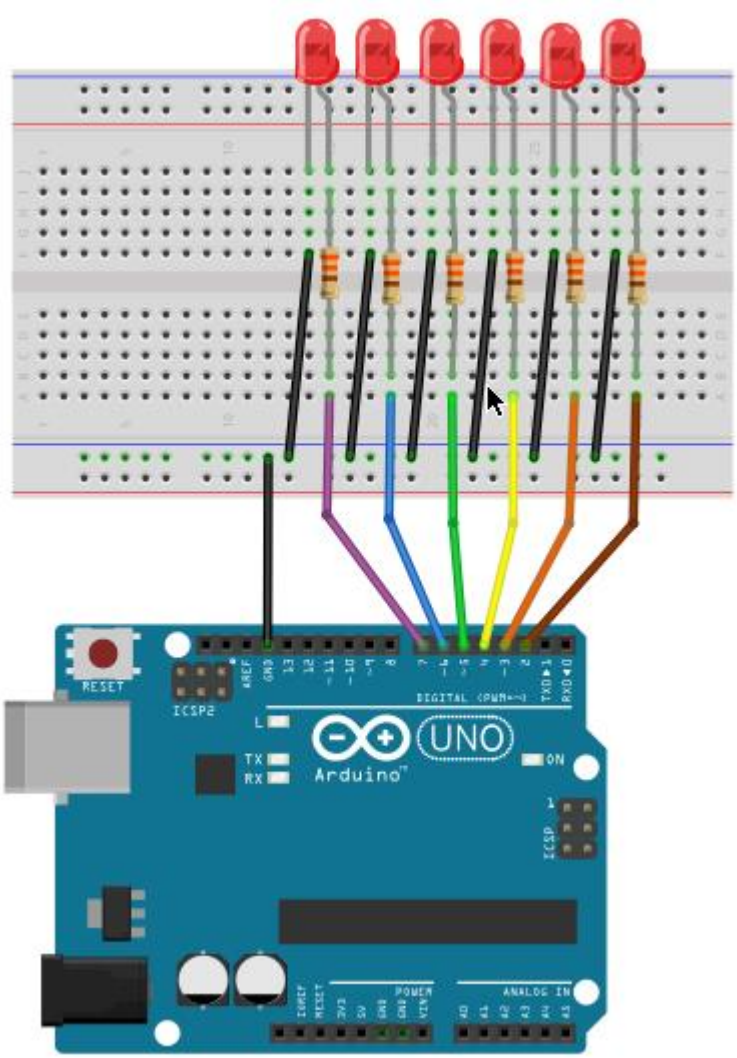
```
1 DDRD = B00000000;  // Configura los pines del puerto PORTD (D0-D7) como entradas.

2 PORTD = B00001111; // Habilitar las pull-ups de los pines D0-D3.
```

Es el equivalente de usar varias veces las funciones **digitalRead()** y **digitalWrite()** de Arduino. Sin embargo, con acceso directo al puerto se puede ahorrar espacio en la memoria flash, porque cada operación con funciones de leer estados de un puerto ocupa varios bytes de instrucciones, y también se puede ganar mucha velocidad, porque las funciones Arduino puede tomar más de 40 ciclos de reloj para leer o escribir un solo bit en un puerto.

Nada mejor que un ejemplo concreto para entender las instrucciones de programa. Veamos un ejemplo con Leds.

Diagrama:



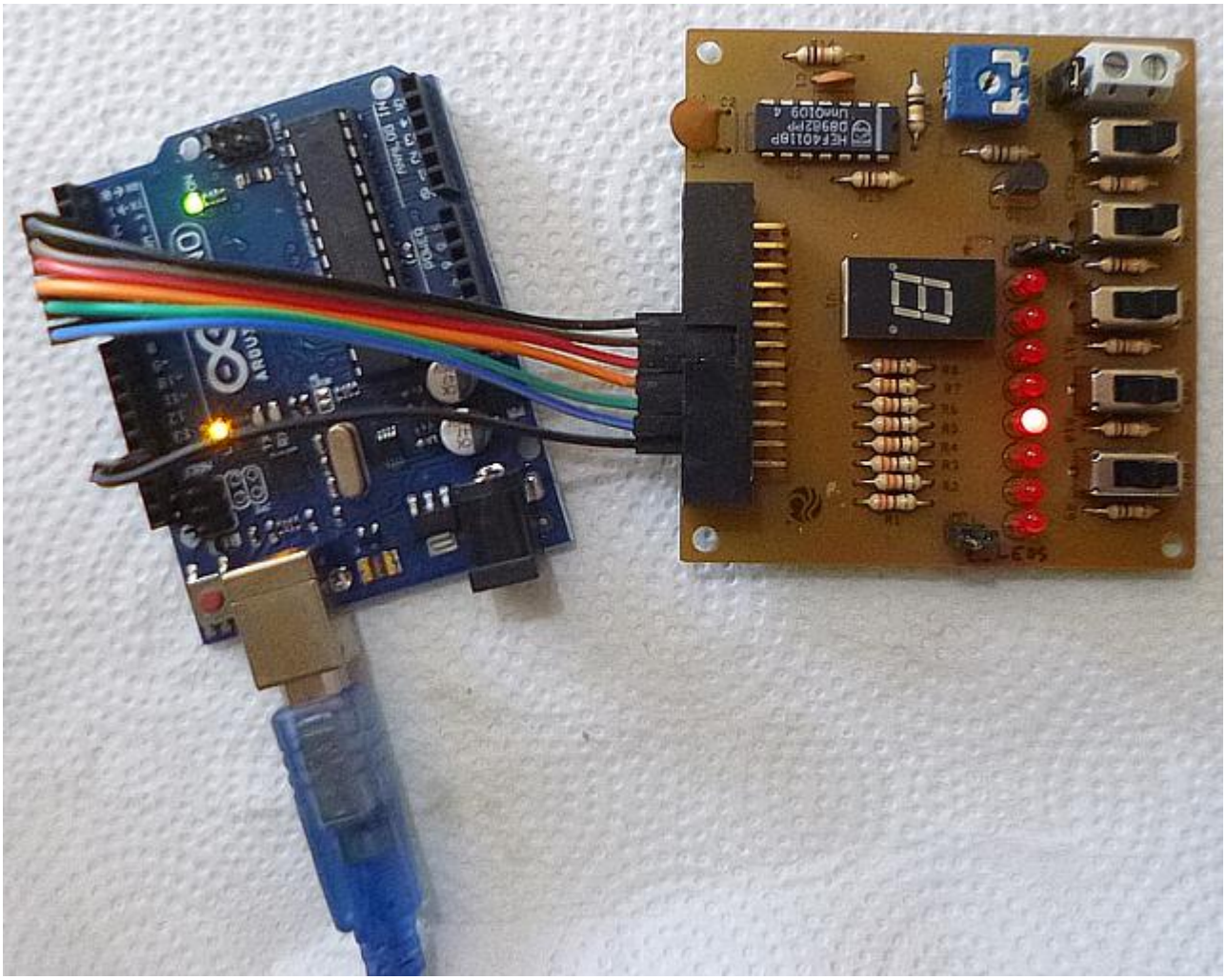
En este ejemplo, durante dos segundos todos los leds encienden, durante otros dos segundos se encienden los impares, luego los pares, y durante dos más se apagan todos.

Abrimos Arduino IDE y escribimos el siguiente código:

Prueba con registros directos
Arduino

```
1  void setup(){
2    DDRD= B11111100; // Utilizamos seis bits del puerto D.
3
4  // En este ejemplo no utilizamos los pines 0 y 1 del Arduino
```

```
5  // para no interferir con la comunicación serie.
6
7  /* Esta instrucción equivale a hacer esto:
8    pinMode(2,OUTPUT);
9    pinMode(3,OUTPUT);
10   pinMode(4,OUTPUT);
11   pinMode(5,OUTPUT);
12   pinMode(6,OUTPUT);
13   pinMode(7,OUTPUT);
14 */
15 }
16 void loop(){
17   PORTD= B11111100; // aquí encenderemos todos los leds
18   delay(2000);
19   PORTD= B10101000; // aquí encenderemos solo una mitad.
20   delay(2000);
21   PORTD= B01010100; // aquí encenderemos la otra mitad.
22   delay(2000);
23   PORTD= B00000000; // aquí los apagaremos todos.
24   delay(2000);
25 }
```



Un segundo ejemplo que puede resultar más divertido. Enciende un led en secuencia hacia a un lado y hacia el otro. No hice más sofisticado el programa para claridad y comprensión. Se puede hacer dentro de una estructura **for**, por ejemplo, tomando el dato a poner el puerto de una matriz. O usando otros recursos más complicados, como desplazar el dato sobre un registro antes de ponerlo al puerto. Pero prefiero que sea didáctico y comprensible para todos.

La secuencia suelen llamarla «El auto fantástico». Hasta se vende un dispositivo con ese efecto de luces para adornar los autos.

El programa es:

Arduino

```
1  void setup(){
2    DDRD= B11111100; // Utilizamos seis bits del puerto D.
```

```
3
4  // En este ejemplo no utilizamos los pines 0 y 1 del Arduino
5  // para no interferir con la comunicación serie.
6
7  /* Esta instrucción equivale a hacer esto:
8    pinMode(2,OUTPUT);
9    pinMode(3,OUTPUT);
10   pinMode(4,OUTPUT);
11   pinMode(5,OUTPUT);
12   pinMode(6,OUTPUT);
13   pinMode(7,OUTPUT);
14 */
15 }
16 void loop(){
17   PORTD= B00000100; // aquí encenderemos un primer led.
18   delay(200);
19   PORTD= B00001000; // aquí el siguiente.
20   delay(200);
21   PORTD= B00010000; // y sigue la secuencia.
22   delay(200);
23   PORTD= B00100000; // sigue.
24   delay(200);
25   PORTD= B01000000; // sigue.
26   delay(200);
27   PORTD= B10000000; // sigue.
28   delay(200);
29   PORTD= B01000000; // y ahora regresa.
30   delay(200);
31   PORTD= B00100000; // sigue.
32   delay(200);
33   PORTD= B00010000; // sigue.
34   delay(200);
```

```
35   PORTD= B00001000; // sigue.
36   delay(200);
37 }
```

Reproductor de vídeo

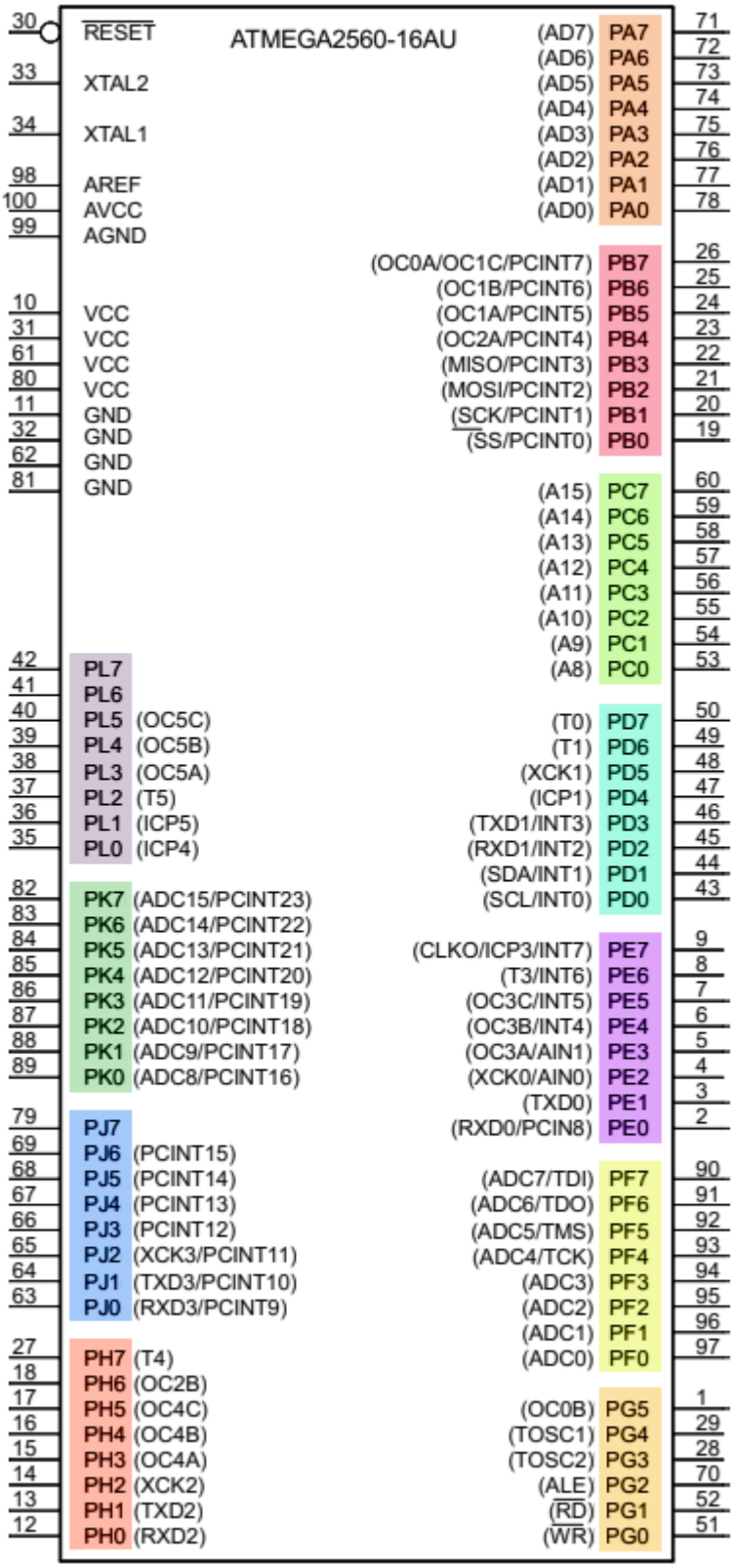
00:00
00:11

@nbsp;

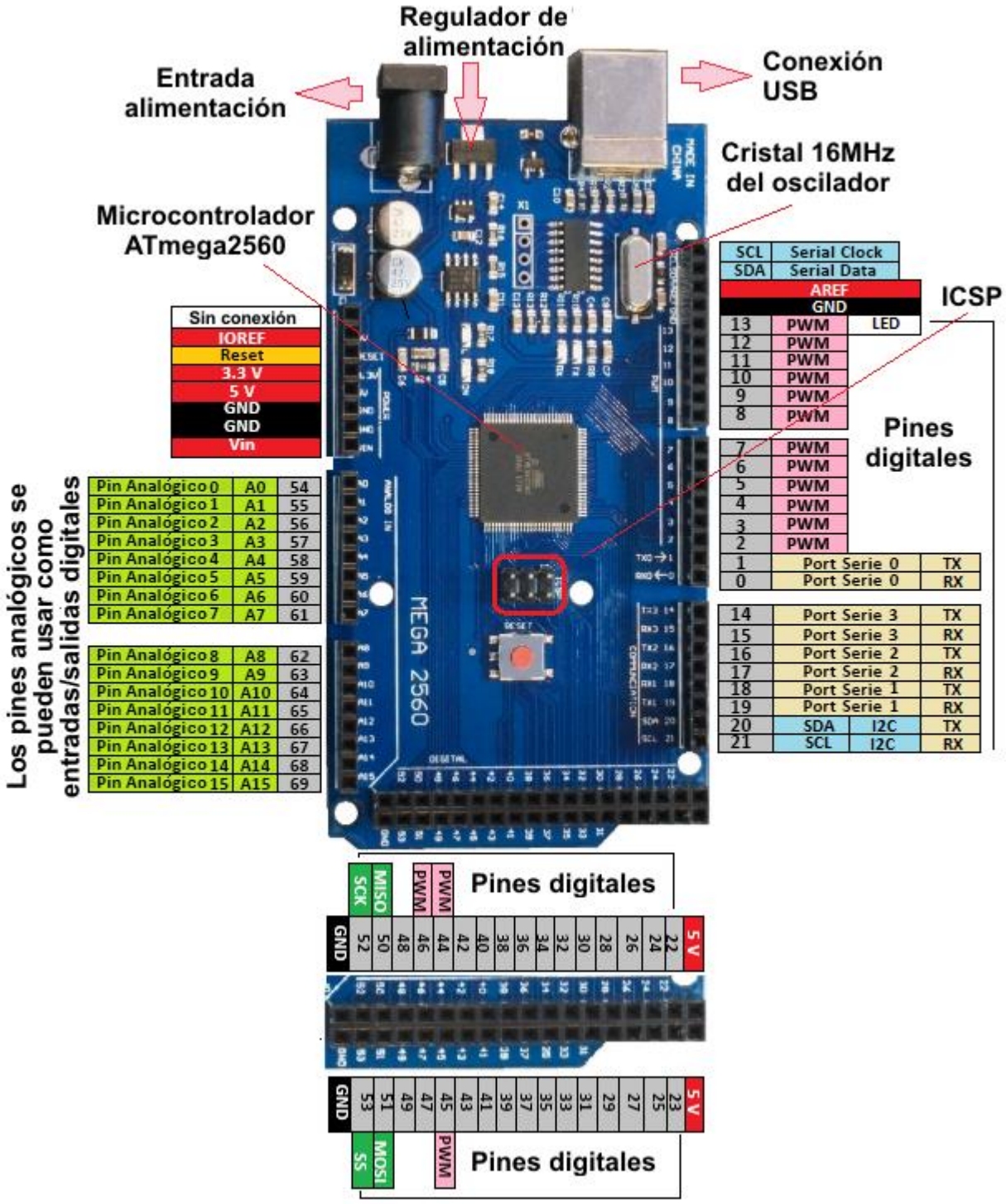
Un chip de la liga mayor:

Sólo por completar la información, y dar una idea de la importancia de este tipo de ahorro de código en otros microcontroladores, voy a mostrar algunos datos de los puertos del ATMega2560, el núcleo del Arduino Mega 2560 R3 (esquemático). Un tremendo chip con nada menos que 100 pines y ONCE puertos, que van desde el PORTA al PORTL. ¡Imagínense el banco de registros que hay para manejar!

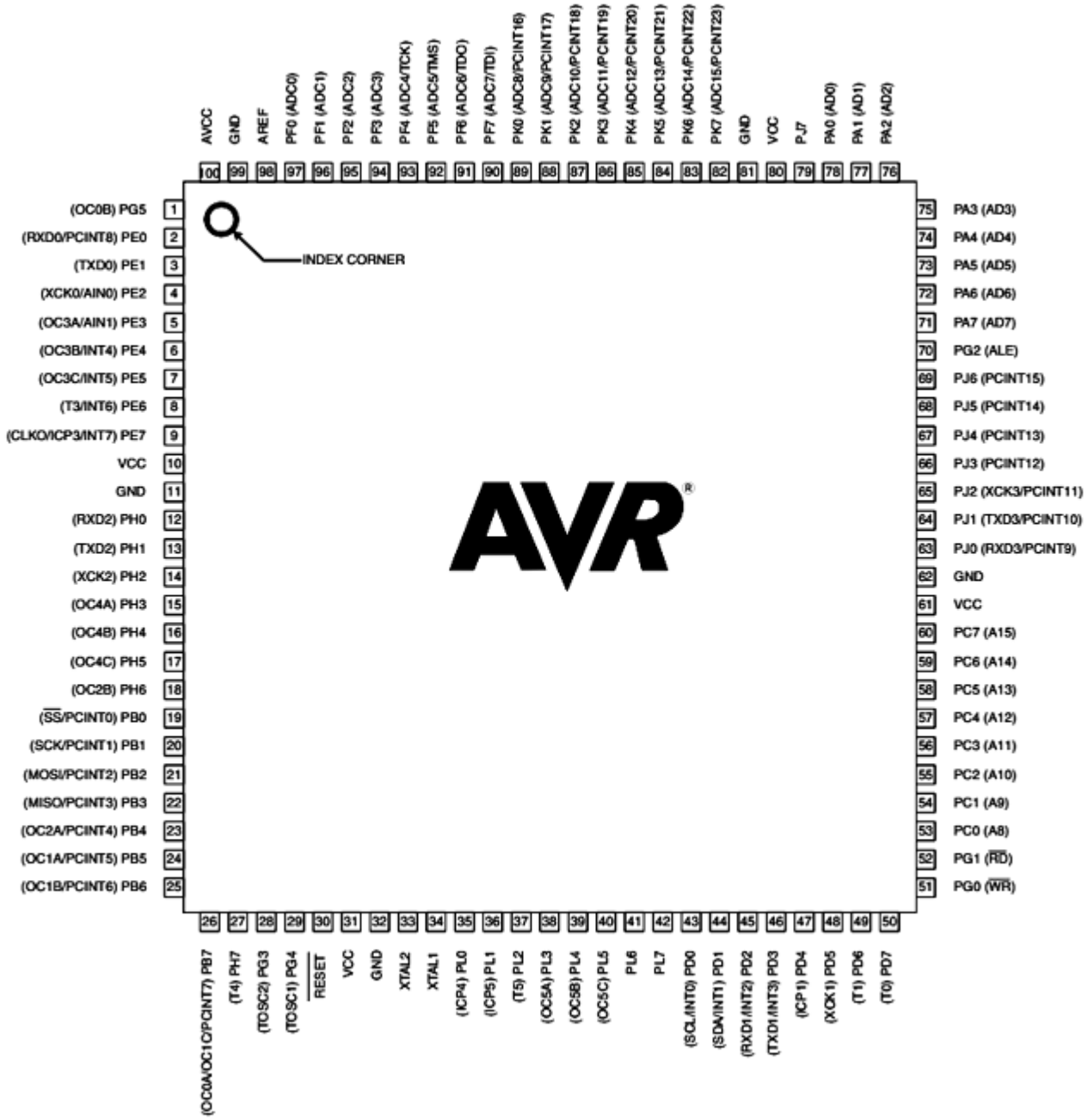
Microcontrolador ATmega2560-16AU



Placa Arduino Mega 2560 R3



Encapsulado del ATmega2560-16AU utilizado en el Mega 2560


AVCC 100
GND 99
AREF 98
PF0 (ADC0) 97
PF1 (ADC1) 96
PF2 (ADC2) 95
PF3 (ADC3) 94
PF4 (ADC4/TCK) 93
PF5 (ADC5/TMS) 92
PF6 (ADC6/TDO) 91
PF7 (ADC7/TDI) 90
PK0 (ADC8/PCINT16) 89
PK1 (ADC9/PCINT17) 88
PK2 (ADC10/PCINT18) 87
PK3 (ADC11/PCINT19) 86
PK4 (ADC12/PCINT20) 85
PK5 (ADC13/PCINT21) 84
PK6 (ADC14/PCINT22) 83
PK7 (ADC15/PCINT23) 82
GND 81
VCC 80
PJ7 79
PA0 (AD0) 78
PA1 (AD1) 77
PA2 (AD2) 76
(OC0B) PG5 1
(RXD0/PCINT8) PE0 2
(TXD0) PE1 3
(XCK0/AIN0) PE2 4
(OC3A/AIN1) PE3 5
(OC3B/INT4) PE4 6
(OC3C/INT5) PE5 7
(T3/INT6) PE6 8
(CLKO/ICP3/INT7) PE7 9
VCC 10
GND 11
(RXD2) PH0 12
(TXD2) PH1 13
(XCK2) PH2 14
(OC4A) PH3 15
(OC4B) PH4 16
(OC4C) PH5 17
(OC2B) PH6 18
(SS/PCINT0) PB0 19
(SCK/PCINT1) PB1 20
(MOSI/PCINT2) PB2 21
(MISO/PCINT3) PB3 22
(OC2A/PCINT4) PB4 23
(OC1A/PCINT5) PB5 24
(OC1B/PCINT6) PB6 25
INDEX CORNER
PA3 (AD3) 75
PA4 (AD4) 74
PA5 (AD5) 73
PA6 (AD6) 72
PA7 (AD7) 71
PG2 (ALE) 70
PJ6 (PCINT15) 69
PJ5 (PCINT14) 68
PJ4 (PCINT13) 67
PJ3 (PCINT12) 66
PJ2 (XCK3/PCINT11) 65
PJ1 (TXD3/PCINT10) 64
PJ0 (RXD3/PCINT9) 63
GND 62
VCC 61
PC7 (A15) 60
PC6 (A14) 59
PC5 (A13) 58
PC4 (A12) 57
PC3 (A11) 56
PC2 (A10) 55
PC1 (A9) 54
PC0 (A8) 53
PG1 (RD) 52
PG0 (WR) 51
(OC0A/OC1C/PCINT7) PB7 26
(T4) PH7 27
(TOSC2) PG3 28
(TOSC1) PG4 29
RESET 30
VCC 31
GND 32
XTAL2 33
XTAL1 34
(ICP4) PL0 35
(ICP5) PL1 36
(T5) PL2 37
(OC5A) PL3 38
(OC5B) PL4 39
(OC5C) PL5 40
PL6 41
PL7 42
(SCL/INT0) PD0 43
(SDA/INT1) PD1 44
(RXD1/INT2) PD2 45
(TXD1/INT3) PD3 46
(ICP1) PD4 47
(XCK1) PD5 48
(T1) PD6 49
(T0) PD7 50