



Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE-0624 Laboratorio de Microcontroladores

EIE

Escuela de
Ingeniería Eléctrica

Arduino UNO: GPIO, ADC y comunicaciones

MSc. Marco Villalta Fallas - `marco.villalta@ucr.ac.cr`

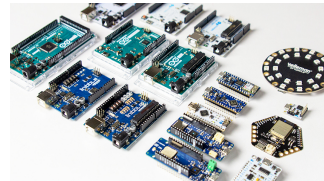
II Ciclo 2023

Arduino

Arduino

Descripción general

- Placas de HW+SW libre con microcontrolador programable
- Mayoría de microcontroladores de la misma familia (AVR)
- IDE y lenguaje(C/C++) inspirados en Processing
- Hardware inspirado en placa libre Wiring (2003)
- Nace en 2005 en el instituto Ivrea (Italia)



Por qué Arduino?

- Es libre y extensible
- Comunidad
- Entorno de programación multiplataforma
- Entorno y lenguaje simples
- Placas baratas, reutilizables y versátiles

Descripción general

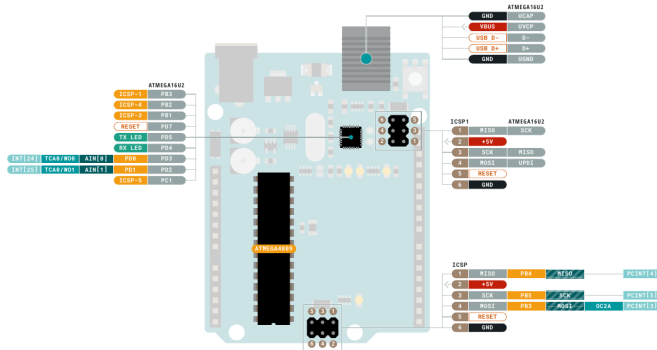
-

 Ground	 Digital Pin	 Analog	 MAXIMUM current per I/O pin is 20mA	 VIN 6-20 V input to the board
 Power	 Analog Pin	 Communication	 MAXIMUM current for +3.3V pin is 50mA	
 LED	 Other Pin	 Timer		
 Internal Pin	 Microcontroller's Port	 Interrupt		
 SWD Pin	 Default	 Sercom		

Arduino UNO

ATMega16

- ATMega16 utilizado como un puente entre el puerto USB y puerto serial del microcontrolador principal
- Firmware se puede actualizar con DFU (no muy necesario)
- ICSP (In-Circuit Serial Programming) Puertos para programar el firmware de un uC



ATMega328P

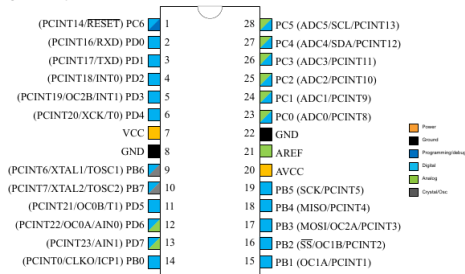
ATMega328

Descripción general

- Microcontrolador AVR de 8 bits
- Arquitectura RISC/Harvard.
- 4/8/16/64Kb Flash, 512b/1/2k bytes de SRAM y 256/512/1k bytes de EEPROM
- 23 GPIOs
- Timer/Counters de 8 y 16 bits.
- Interrupciones
- 8 canales PWM y comparador analogico
- 6 canales 10-bit ADC
- USART
- Maestro/Esclavo SPI (USARTSPI)
- TWI (2-wire), DBG, BTLDR

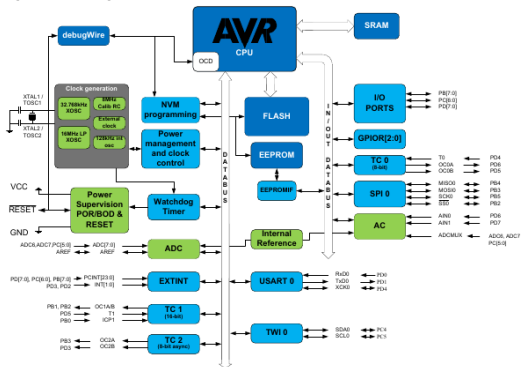
Pin-out

Figure 5-1. 28-pin PDIP



Arquitectura

Figure 4-1. Block Diagram



Uso de perifericos

Estructura general de un sketch

Programa(sketch) que se ejecuta en un Arduino siempre se compone de tres secciones:

- Declaración de variables globales y funciones propias
- Inicialización/configuración de periféricos (**void setup()**)
- Ejecución cíclica (**void loop()**)

```
int var_global;
```

```
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}
```

```
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);                     // wait for a second
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);                     // wait for a second
}
```

E/S digitales

Características eléctricas

- Voltajes: 0 y 5V
- E/S máxima de 20mA, 100mA en total a la vez para todos los pines.
- Resistencia pull-up de 20-50 kOhm

E/S digitales

Uso de entradas y salidas

Las funciones para trabajar con entradas y salidas digitales son:

- **pinMode()**: Configura pin digital. Primer parámetro indica el pin, segundo parámetro indica E/S con INPUT o OUTPUT
- Si se usa entrada se puede activar resistencia pull-up con la constante INPUT_PULLUP en lugar de INPUT. Se recomienda este modo para evitar ruido.
- **digitalWrite()**: Pone en alto o bajo un pin digital. Pin se especifica como primer parámetro y el segundo parámetro el estado
- Si pin es entrada un HIGH con digitalWrite() habilita resistencia pull-up, con LOW se desactiva
- **digitalRead()**: devuelve valor leído del pin digital (primer parámetro)
- **pulseIn()**: pausa la ejecución del sketch y espera a recibir en pin (primer parámetro) una entrada (segundo parámetro). Devuelve en microsegundos el tiempo en volver a cambiar de estado. Mide un pulso.

E/S digitales

Ejemplo de lectura y escritura

```
int ledPin = 13; // LED connected to digital pin 13
int inPin = 7;   // pushbutton connected to digital pin 7
int val = 0;     // variable to store the read value

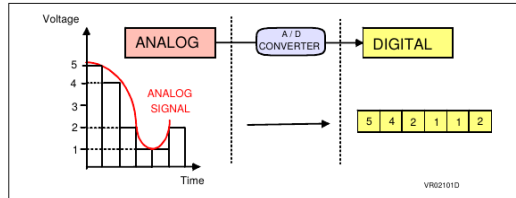
void setup() {
  pinMode(ledPin, OUTPUT); // sets the digital pin 13 as output
  pinMode(inPin, INPUT);   // sets the digital pin 7 as input
}

void loop() {
  val = digitalRead(inPin); // read the input pin
  digitalWrite(ledPin, val); // sets the LED to the button's value
}
```

E/S analógicas

Características eléctricas

- Voltajes: De 0 a 5V
- 6 canales de 10 bits (0-1023 pasos)
- Paso de 5mV



E/S analógicas

Funciones

- **analogWrite():** Envía valor tipo *byte* que representa señal PWM a pin digital configurado como OUTPUT. Se escriben valores entre 0 y 255
- **analogRead():** Devuelve valor leído del pin de entrada analógico, el valor es proporcional a la entrada analógica con respecto al voltaje de referencia a un valor entre 0 y 1023. Si se reduce voltaje de referencia se detectan cambios más pequeños.
- No es necesario usar función *pinMode()* para pines analógicos por estar configurados por defecto como entradas analógicas. También se pueden utilizar como pines digitales.
- Convertidor A/D tarda 100uS en procesar conversión y obtener valor digital.
- El Arduino duo cuenta con las funciones `analogWriteResolution()` y `analogReadResolution()`

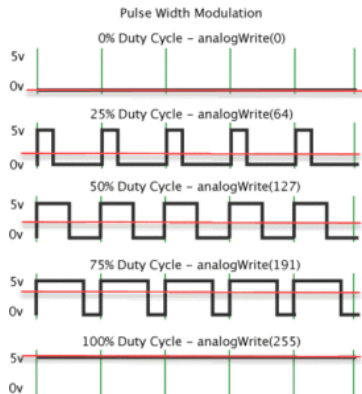
E/S analógicas

Cambio voltaje de referencia

- Reducción de voltaje de referencia permite detectar cambios más pequeños. Puede reducirse por HW y/o SW
- En HW modificar voltaje en pin AREF
- En SW utilizar **analogReference()**: Recibí un único parámetro cuyo valor puede ser:
 - DEFAULT: equivale a establecer el voltaje de referencia 3.3V o 5V
 - INTERNAL: Voltaje de referencia es 1.1V
 - EXTERNAL: Utiliza voltaje aplicado en AREF
- AREF no puede ser menor a 0V ni mayor de 5V

E/S analógicas

PWM



- Método para reducir la potencia promedio entregada por una señal eléctrica, controlada por la conmutación de un switch entre la fuente y la carga
- Forma de controlar dispositivos analógicos con una salida digital
- Ciclo de trabajo: proporción/porcentaje de encendido en intervalos regulares de tiempo
- $V_{medio} = DC/100 * V_{alto}$
- Usos: intensidad de leds, dirección servos, actuadores, speakers

E/S analógicas

Funciones PWM

- **analogWrite()**: Envía valor tipo *byte* que representa señal PWM a pin digital configurado como OUTPUT.
- Una llamada de `analogWrite()` es en una escala de 0 - 255. Ejemplo: `analogWrite(255)` requiere 100 % del DC, `analogWrite(127)` requiere 50 %
- `analogWrite()` no tiene relación con pines analógicos de entrada (A0...An), solo con pines tipo PWM

Timers y PWM

Arduino Mega

- Las funciones PWM que son controladas por hardware emplean los módulos Timer para generar la onda de salida
- En el arduino Mega:
 - Timer0 controla salidas PWM 4 y 13.
 - Timer1 controla salidas PWM 11 y 12.
 - Timer2 controla salidas PWM 9 y 10.
 - Timer3 controla salidas PWM 2, 3 y 5.
 - Timer4 controla salidas PWM 6, 7 y 8.
 - Timer5 controla salidas PWM 44, 45 y 46.
- Frecuencia de ciclo PWM: frecuencia estándar es de 490 Hz para todos los pines, excepto para el 4 y 13 cuya frecuencia es de 980 Hz
- Uso de Timer no es exclusivo para PWM

E/S analógicas

Ejemplo de lectura y escritura

```
int ledPin = 9;           // LED connected to digital pin 9
int analogPin = 3;       // potentiometer connected to analog pin 3
int val = 0;             // variable to store the read value

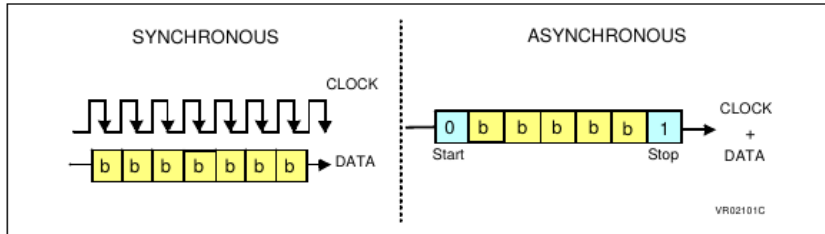
void setup() {
  pinMode(ledPin, OUTPUT); // sets the pin as output
}

void loop() {
  val = analogRead(analogPin); // read the input pin
  analogWrite(ledPin, val / 4); // analogRead values go from 0 to 1023, analogWrite values from 0 to 255
}
```

Comunicaciones

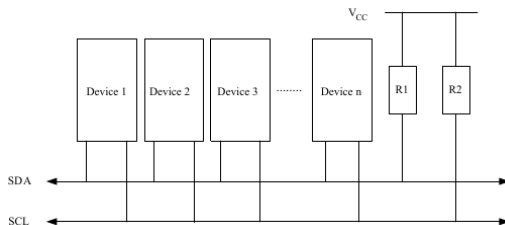
Comunicaciones con/en uC

- Mayoría de comunicaciones con MCU son en serie
- Información es transmitida bit a bit.
- Existen varios protocolos y estándares
- Diferencias de protocolos: modo de sincronización, velocidad, tamaño de paquete de datos, mensajes de conexión y desconexión, voltajes
- TWI/I2C,SPI,UART



Comunicaciones: TWI/I2C

- Interfaz de comunicación serial con 2 líneas
- SCL: Señal de reloj, SDA: Señal de datos
- Modelo Maestro - Esclavo
- Maestro inicia/termina transmisión. Genera reloj
- Direcccionamiento de 7 bit (128 esclavos)
- Velocidad de 100kHz(standard) y 400kHz (fast) en transmisión y recepción (Half duplex)
- Voltajes típicos: +5V,+3.3V



Comunicaciones: SPI

- Interfaz de comunicación serial con 4 líneas
- SCLK: Señal de reloj, SS: Slave Select(activo en bajo), MOSI: Master Output Slave Input (salida datos de maestro, MISO: Master Input Slave Output (salida datos de esclavo)
- Modelo Maestro - Esclavo
- Full duplex
- Sin limitación de velocidad

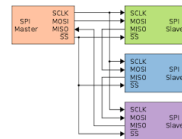


Figura: Bus SPI típico

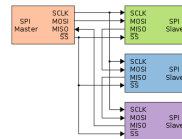
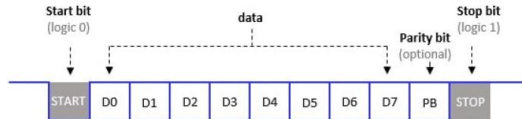


Figura: Bus SPI en daisy chain

Comunicaciones: UART/USART

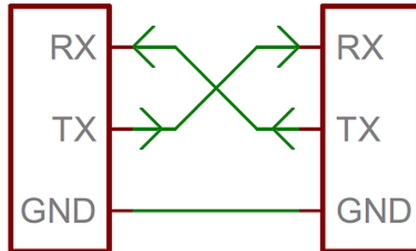
- UART: Universal Asynchronous Receiver Transmitter (sin reloj, orientado a transmisiones de bytes)
- USART: Universal Synchronous/Asynchronous Receiver Transmitter (ambos modos, más rápido)
- Estructura del data-frame:
 - Start bit: Marca inicio y siempre es bajo
 - Data: Puede ser de 5 a 9 bits
 - Stop bit: Puede ser de 1-2 bits y siempre es alto
- USART utiliza clock y permite enviar paquetes de datos mayores que una UART



Comunicaciones entre Arduino y PC

Conceptos

- Comunicaciones se establecen a una velocidad o *baud rate*, algunos valores típicos pueden ser 9600,56000,115200..etc
- Baud rate y otros parámetros deben ser igual en ambos dispositivos
- Se pueden establecer varios parámetros en la comunicación como paridad, stop bit, start bit.
- En el Arduino UNO se utilizan pines digitales 0 (RX) y 1(TX)
- Ver imagen para conexión física.



Comunicaciones entre Arduino y PC

Funciones para enviar datos

- **Serial.begin():** Abre comunicación, va en setup(). Primer parámetro es velocidad y segundo parámetro configuración
- **Serial.print():** Envía dato de cualquier tipo (especificado como parámetro). Cada tipo de dato tiene opciones. Transmisión asincrónica. Devuelve cantidad de bytes enviados.
- **Serial.flush():** Espera hasta transmisión sea completa volviendo al print bloqueante.
- **Serial.println():** Hace lo mismo que print pero agrega el carácter de retorno y el de nueva línea.
- **Serial.write():** Envía un dato (especificado como parámetro). Dato solo puede ocupar un byte. Devuelve cantidad de bytes enviados.
- **Serial.end():** Cierra comunicación

Comunicaciones entre Arduino y PC

Funciones para recibir datos

- **Serial.available():** devuelve número de bytes disponibles para ser leídos
- **Serial.read():** devuelve primer byte aún no leído en buffer y lo borra del buffer
- **Serial.peek():** devuelve primer byte aún no leído en buffer y **NO** lo borra del buffer
- **Serial.find():** lee datos del buffer hasta encontrar cadena de caracteres especificado como parámetro
- **Serial.readBytes():** Lee del buffer cantidad de bytes especificada como segundo parámetro
- Otras funciones: Serial.findUntil(), Serial.readBytesUntil(), Serial.setTimeout(), Serial.parseFloat(), Serial.parseInt()

Comunicaciones entre Arduino y PC

Software Serial

- En caso de necesitar más de un puerto serial se puede utilizar la biblioteca SoftwareSerial
- Utiliza 2 pines digitales

```
#include <SoftwareSerial.h>
```

```
SoftwareSerial mySerial(2, 3); // RX, TX
```

```
void setup()
```

```
{
  Serial.begin(115200); // Open serial communications and wait for port to open:
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Native USB only
  }
}
```

```
Serial.println("Goodnight moon!");
// set the data rate for the SoftwareSerial port
mySerial.begin(38400);
mySerial.println("Hello, world?");
}
```

```
void loop() // run over and over
```

```
{
  if (mySerial.available())
    Serial.write(mySerial.read());
  if (Serial.available())
    mySerial.write(Serial.read());
}
```

Hola Arduino USART

```
void setup() {
  Serial.begin(115200,SERIAL_8N1);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB
  }
}

void loop() {
  Serial.write(45); // send a byte with the value 45

  int bytesSent = Serial.write("hello"); //send the string "hello" and return the length of the string.

  // send data only when you receive data:
  if (Serial.available() > 0) {
    // read the incoming byte:
    incomingByte = Serial.read();

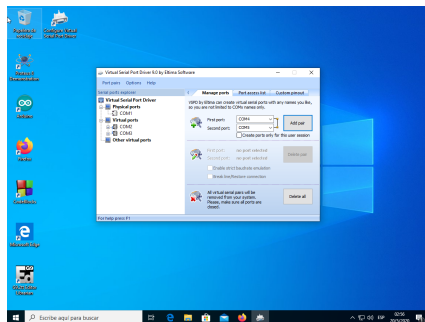
    // say what you got:
    Serial.print("I received: ");
    Serial.println(incomingByte, DEC);
  }
}
```

Comunicación entre PC y Simulide

- Es posible conectar un puerto serial dentro de Simulide con el mundo externo
- Es necesario simular un puerto serial de entrada y un puerto serial de salida
- Windows: Utilizar com2com / Virtual serial port / vspd
- Linux: Utilizar socat

```
socat -d -d pty,raw,echo=0 pty,raw,echo=0
```

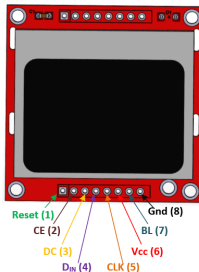
```
socat PTY,link=/tmp/ttyS0,raw,echo=0 PTY,link=/tmp/ttyS1,raw,echo=0
```



Pantalla PCD8544

Pinout

- RST: Reset Pin
- SCE: chip select Pin
- D/C: Selector de operación en modo (LOW) /comando (HIGH)
- DN (Data Pin): Serial Data In
- SCLK: Serial Clock
- VCC: Alimentación de 2.7 a 3.3V
- LED/BL: Backlight LED. Voltaje de 3.3V
- GND: Ground



Hola PCD8544

Utilizando biblioteca PCD8544

<https://github.com/carlosefr/pcd8544/archive/master.zip>

```
#include <PCD8544.h>
```

```
void setup() {  
  lcd.begin(); // default resolution is 84x48  
}
```

```
void loop() {  
  lcd.clear();  
  // Write some text on the first line...  
  lcd.setCursor(0, 0);  
  lcd.print("Hello, World!");  
  
  // Write the counter on the second line...  
  lcd.setCursor(0, 1);  
  lcd.print(counter);  
  
  delay(200);  
  counter++;  
}
```