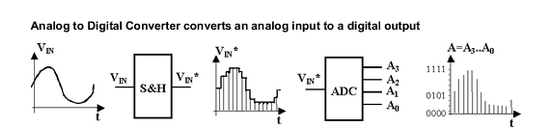
Vacas Martinez, José Antonio

# Tema 8 - Expandiendo Arduino

En este tema veremos distintas formas de ampliar Arduino, bien sea en cómo controlar otros dispositivos o como tener una mayor control sobre sus funcionalidades.

## ADC: Analog TO Digital Converter



ADC

Un ADC es un sistema que mide una magnitud analógica proporcionando un valor digital. Se utilizan para medir voltajes normalmente.

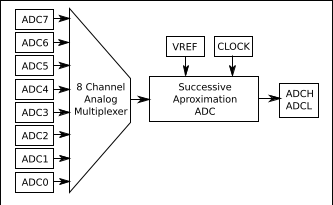
En Arduino el ADC es de 10 bits, es decir divide el rango de voltaje que mide (normalmente 0-5V) en 1024 partes.

Se pueden conseguir 9600 muestras (samples) por segundo con el sistema Arduino. Podemos modificar esta velocidad usando el preescaler (del que ya hablamos cuando lo hicimos sobre los Timers), en [este enlace](http://yaab-arduino.blogspot.com.es/2015/02/fast-sampling-from-analog-input.html) se comenta cómo hacerlo.

Usando el IDE Arduino la forma natural de obtener valores analógicos como ya hemos visto es usando

analogRead(EntradaAnalógica)

Aunque en un Arduino UNO disponemos de 6 entradas analógicas, internamente sólo existe un medidor, que mediante un multiplexor accede a cada una de las entradas.



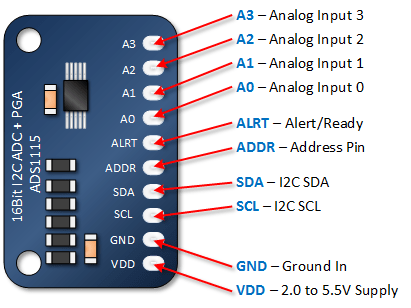
ADC\_Multiplexor

El hardware de arduino permite  leer datos ADC en 2 modos:

* Single Conversion Mode: una sola muestra
* Free Running: medida continua

Por defecto se usa la segunda opción. Podría ser deseable usar la primera para bajar el consumo.

Por supuesto que podemos añadir hardware externo más avanzado con capacidad de usar más bits de resolución y con mayor velocidad, por ejemplo usando el chip ADS1115 podrías alcanzar resoluciones de 16 bits. Además este chip nos permite amplificar la señal usando lo que se conoce como PGA (Ganancia). Os dejo un [excelente tutorial](https://www.luisllamas.es/entrada-analogica-adc-de-16-bits-con-arduino-y-ads1115/) sobre cómo usarlo conectándolo vía I2C.



ADS1115

### Cambiando el rango de medida

Podemos cambiar el rango de voltaje a medir usando la función analogReference que configura el voltaje de referencia usado por la entrada analógica. La función analogRead() devolverá un valor de 1023 para aquella tensión de entrada que sea igual a la tensión de referencia. Las opciones son: \* DEFAULT: Es el valor de referencia analógico que viene por defecto que es de 5 voltios en placas Arduino y de 3,3 voltios en placas Arduino que funcionen con 3,3 voltios. \* INTERNAL: Es una referencia de tensión interna de 1.1 voltios en el ATmega168 o ATmega328 y de 2.56 voltios en elATmega8. \* EXTERNAL: Se usará una tensión de referencia externa que tendrá que ser conectada al pin AREF. En el siguiente vídeo veremos esto en detalle.

En [este vídeo](https://www.youtube.com/embed/J3vdHNQJI54) se habla del ADC.

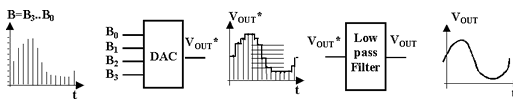
### Referencias

Para ampliar sobre el tema puedes leer:

[Midiendo el propio Vin/Vcc](http://jeelabs.org/2012/05/04/measuring-vcc-via-the-bandgap/)

[ADC Avanzado](https://www.pjrc.com/teensy/adc.html)

## DAC: Digital To Analog Converter



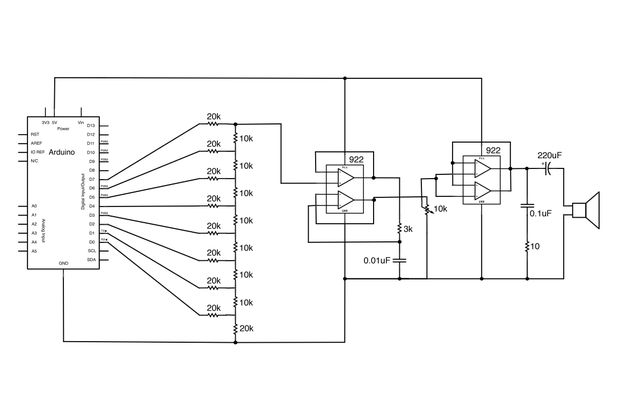
DAC

Podemos decir que el DAC (Digital to Analog Converter) es el proceso inverso al ADC. Es decir a partir de un conjunto de valores digitales obtenemos una señal analógica. Para mejorar la calidad de la señal se la somete a un filtro paso bajo para suavizar.

El número de bits caracteriza la calidad del DAC, es decir, si usamos un DAC de 10 bits, tendremos una señal con una calidad de V/1024 valores.

Salvo Arduino DUE, las placas Arduino no disponen de DAC, si no que nos permiten simularlo con PWM. En todos ellos usaremos la función analogWrite para obtener un valor.

Podemos crear un DAC usando una red de resistencias



DAC\_resustencias.png

Si conectamos este montaje al puerto B (pines 0-7), escribiendo un valor en el puerto, obtendremos un salida analógica proporcional al valor utilizado.

¿Para qué podemos usar un DAC?

* Para producir sonido de más calidad
* Para generar señales más precisas
* Generar señales de vídeo ... Utilizando un sistema de refresco suficientemente rápido y bien sincronizado y un montaje, muy, muy sencillo (sólo 2 resistencias)  se puede generar una señal analógica capaz de visualizarse como vídeo en una TV. Más detalles en [este instructable](http://www.instructables.com/id/MRETV-Video-Stereo-Sound-and-much-more-from-a-s/)



DAC\_Video.png

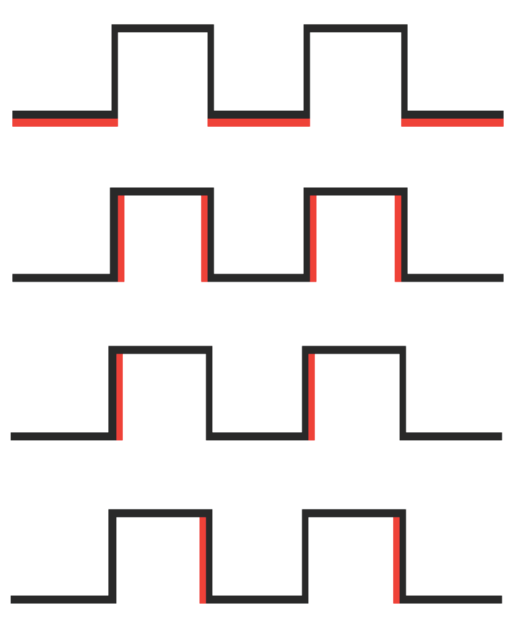
En [este vídeo](https://www.youtube.com/embed/ZNeFd8kK9h0) hablamos del DAC.

## Interrupciones hardware

Para que se produzca una interrupción hardware, el microcontrolador debe disponer de una electrónica capaz de hacerlo.

Los microcontroladores utilizados en Arduino (ATMega) disponen de varias patillas capaces de generar interrupciones hardware ante cambios de su estado.

En concreto en arduino UNO, las interrupciones externas llamadas INT0 e INT1 están asociadas a los pines 2 y 3. El Arduino Mega tiene otras 4: 2 (pin 21), 3 (pin 20), 4 (pin 19) y 5 (pin 18) )



InterrupcionHardware.png

Podemos configurarlas dependiendo del cambio de estado como

* estado bajo -> LOW
* cambio -> CHANGE
* de subida -> RISING
* de bajada -> FALLING

Para activar una interrupción sólo tenemos que llamar al método attachInterrupt con el número de interrupción a usar, la función a la que se llamará cuando se produzca y el tipo de cambio a detectar

Veamos un ejemplo donde hemos conectado un led al pin 11 (con su resistencia) y un pulsador en el pin 2 (interrupción hardware 0)

const int led\_pin = 11;  
int estado = LOW ;  
  
void setup() {  
pinMode(led\_pin,OUTPUT);  
// configuramos interrupcion  
attachInterrupt(0 ,parpadeo , CHANGE ); // Pin 2  
}  
  
void loop() {  
 digitalWrite(led\_pin , estado);  
}  
  
// Funcion que se llamara en la interrupcion  
void parpadeo(){  
 if(estado == HIGH ) { // estado = !estado;  
 estado = LOW ;  
 } else {  
 estado = HIGH;  
 }  
}

Donde vemos que la función parpadeo se encarga de cambiar el estado del pin cada vez que pulsamos el pulsador

Podríamos estar tentados de mover la llamada de digitalWrite al método parpadeo. Lo hemos dejado así por 2 razones:

* El código ejecutado en las funciones de las interrupciones debe ser lo más corto posible. No olvidemos que estamos interrumpiendo el código "central"
* Así demostramos que estamos haciendo parte del procesamiento en 2 sitios distintos

Hay que tener cuidado con los cambios que se hacen a las variables en una interrupción pues podrían entrar en conflicto con los valores que tenían, por eso se marcan esas variables como "volatile" para el compilador lo tenga en cuenta.

Vamos a añadir al ejemplo un contador que se incrementará con cada pulsación

const int led\_pin = 11;  
int estado = LOW ;  
volatile int contador = 0;  
  
void setup() {  
 Serial.begin(9600);  
 pinMode(led\_pin,OUTPUT);  
 // configuramos interrupcion  
 attachInterrupt(0 ,parpadeo , CHANGE ); // Pin 2  
  
}  
  
void loop() {  
 digitalWrite(led\_pin , estado);  
 Serial.println(contador);  
 delay(100);  
}  
  
// Funcion que se llamara en la interrupcion  
void parpadeo(){  
 if(estado == HIGH ) { // estado = !estado;  
 estado = LOW ;  
 } else {  
 estado = HIGH;  
 }  
 contador = contador +1;  
}

Los micro ATMega tienen muchas más posibilidades de interrupciones, pero no todas están soportadas por arduino. En [esta página](https://sites.google.com/site/qeewiki/books/avr-guide/external-interrupts-on-the-atmega328) hablan de ello.

En el [siguiente vídeo](https://www.youtube.com/embed/n5tkYR5PT0c) se habla de interrupciones hardware.