Una programación exitosa.

Para minimizar los posibles problemas a la hora de programar Arduino es bueno tener claro algunos conceptos.

- *Debe estar seguro que la fuente de alimentación no tiene problemas. El voltaje es el correcto, no hay falsos contactos ni cables sueltos. Si algo no funciona usted puede tener dudas de su código pero debe tener certeza que su electrónica funciona correctamente.*
- *El lenguaje de programación distingue mayúsculas de minúsculas, si escribe "Dato" y luego hace referencia a "dato" el compilador dará un error indicando que la variable no ha sido declarada o no sabe que es.*
- *No mezcle mayúsculas y minúsculas en un programa, es confuso y poco elegante.*
- *No utilice nombres con espacios, es preferible agregar un "_" para representar un espacio.*
- *No utilice nombres largos para variables o nombres de archivos.*
- *Trabaje prolijo, cree una carpeta donde contener todos sus proyectos que a su vez tendrán cada uno de ellos su propia carpeta individual. En estas carpetas podrá agregar notas explicativas, fotos y toda la información que le servirá en el futuro para recordar rápidamente como realizó el trabajo.*

Trabajar con Arduino es realmente sencillo sin embargo relea los consejos anteriores y asegúrese de cumplirlos se evitará muchos dolores de cabeza y perdidas de tiempo.

El microcontrolador del Arduino UNO.

Un microcontrolador es un circuito integrado capaz de ser programado desde una computadora y seguir la secuencia de acciones escrita en el programa grabado en su memoria.
Cuando el microcontrolador ejecuta una instrucción que definimos en el sketch o programa Arduino, internamente hace muchas operaciones y cada una de esas operaciones se ejecuta al ritmo de un reloj.
Para el ATmega 328p que tiene una frecuencia de 16 MHz, es decir, cada ciclo tarda 0,0000000625 segundos = 0,0625 microsegundos = 62,5 nanosegundos, este es el tiempo que se conoce como **un ciclo de CPU** y las instrucciones o comandos pueden llevar mas de un ciclo de CPU.
Es importante entender que si bien Arduino es muy simple de usar, esto no significa que sea simple su funcionamiento electrónico que es verdaderamente

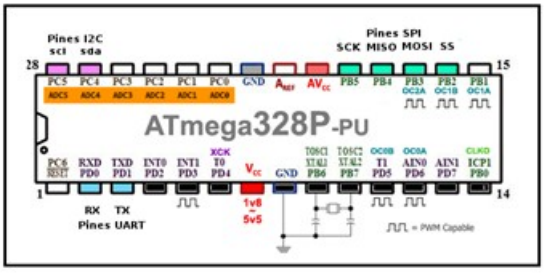
complejo. Un microcontrolador se diferencia de un microprocesador en que el primero centra todos sus recursos en una tarea específica aquella para lo cual fue programado. Todos sus recursos están embebidos en el propio chip, no hay posibilidad de expandir memorias o mejorar sus desempeño a nivel de hardware, solo podemos "afinar" la forma de programar y así obtener un mejor rendimiento del chip.
En el caso de los microprocesadores los recursos de memoria son externos y por tanto se pueden expandir como es el caso de la memoria RAM en una computadora o cambiar el disco para aumentar a capacidad de almacenamiento. En un microcontrolador el disco duro es el equivalente a la memoria de programa o memoria FLASH.
Los microcontroladores se pueden ver como una pequeña ciudad en miniatura donde las calles (*buses*) conectan las distintas casas (*módulos*) incluso muchos de estos módulos son barrios enteros en si mismos, estos módulos sirven para distintas tareas específicas como hacer mediciones, comunicaciones, temporizadores, etc. Dependiendo de la tarea a realizar se invoca a uno de estos módulos para realizar el trabajo.

Recuerde.

> *El que resulte sencillo programar Arduino no significa que esté usando una tecnología simple, el Software está ocultando la complejidad del Hardware para hacerle el trabajo más simple pero no puede hacer magia y resolver un cableado desprolijo o fuentes de alimentación defectuosas.*

Puertos del Arduino UNO.

La mayoría de los pines de los microcontroladores son multipropósito, es decir en función de su configuración se comportan de una forma u otra, tal y como se muestra en la siguiente imagen donde los pines además de la función de puerto tiene funciones secundarias.



Pines del ATMEGA 328

El ATmega328p como cualquier otro microcontrolador tiene registros, si bien

es verdad que el software de programación para Arduino crea una capa de abstracción para el hardware del microcontrolador es interesante conocer algunos conceptos básicos de su estructura interna como por ejemplo un registro que es una posición de memoria que se usa par configurar el funcionamiento del microcontrolador, algunos de estos registros están relacionados con los puertos de entrada/salida, cada puerto tiene un nombre específico y sus registros asociados, de hecho, el 328p tiene el puerto B, C y D, y cada puerto un diferente número de pines (Esta es una restricción del paquete de 28 pines PDIP y no del microcontrolador, ya que un PDIP 40 pines, por ejemplo, tiene 4 puertos con los 8 bits cada uno), el único puerto que tiene el total de sus 8 pines de entradas/salidas es PORTD.
La denominación de los puertos en un órden albabético tiene un parecido a como se ordenaban las unidades de disco en un sistema informatico.
Cada pin puede tener múltiples funciones, como la generación de PWM, o las capacidades de ADC, los pines 6 y 7 del PORTB son los pines de entrada para el oscilador de cristal, y pin 6 del PORTC le corresponde al botón de reinicio.
Como se dijo hay un registro dedicado para cada puerto que define si cada pin es una entrada o una salida, que es el registro de DDRx, donde x es la letra del puerto que queremos configurar, en el caso de la Arduino hay DDRB, DDRC y DDRD. Como toda variable lógica, cada bit en los registros DDRX puede ser 1 ó 0, poner un bit específico de DDRX a 1 configura el pin como salida y ponerla a 0 configura el pin como una entrada. Los pines usados en la placa Arduino poseen tres puertos en el caso de ATmega328p (Arduino Uno):

- B (pines digitales del 8 al 13)
- C (entradas analógicas)
- D (pines digitales del 0 al 7)

El Arduino Mega presenta varios puertos B,C,D,E,F, etc.
Cada puerto es controlado por tres registros, los cuales también están definidos como variables en el lenguaje del Arduino.

- El registro DDR, determina si el pin es una entrada o una salida (1 salida, 0 entrada).
- El registro PORT controla si el pin está en nivel alto (1) o en nivel bajo (0).
- El registro PIN permite leer el estado de un pin. (solo lectura)

Un concepto debe quedar claro, **no puedo conectar cualquier cosa a un pin Arduino**, cada pin tiene características eléctricas muy definidas, recuerde que la placa solo tiene un fusible de protección para el puerto USB, no hay

protección para los pines de los puertos del microcontrolador y puede fácilmente dañarlo de manera permanente si no tiene cuidado a la hora de conectar periféricos a los puertos.
La corriente *máxima absoluta* para un solo pin IO es de 40 mA, básicamente es el límite en el que se Atmel no puede garantizar que el chip no será dañado. Siempre debe asegurarse de que está con seguridad *por debajo de* este límite de corriente.
La corriente total de todos los pines juntos es **de 200 mA máximo no debe superar ese umbral.**
La placa Arduino publica pines de 5V y 3.3V, es bueno conocer algunos detalles de estas salidas.
Los 5V pin de salida tienen alrededor **de** ~400 mA cuando la placa está alimentada desde el USB, y puede entregar alrededor de ~900 mA cuando se utiliza un adaptador de alimentación externa (suponiendo claro que la fuente aplicada tenga esa corriente disponible). Estos 900 mA pueden provenir de un adaptador de 9V que sería un voltaje ideal para el adaptador.

Importante I:

No utilice un adaptador con mucho voltaje de salida porque esto aumenta la cantidad de calor que el regulador tiene que disipar, esto deriva en que la corriente máxima disponible baja a medida que la tensión aumenta y aumenta la temperatura en el regulador, esto se llama *limitación térmica* del regulador. NO utilice adaptadores de más de 12V máximo.

Importante II:

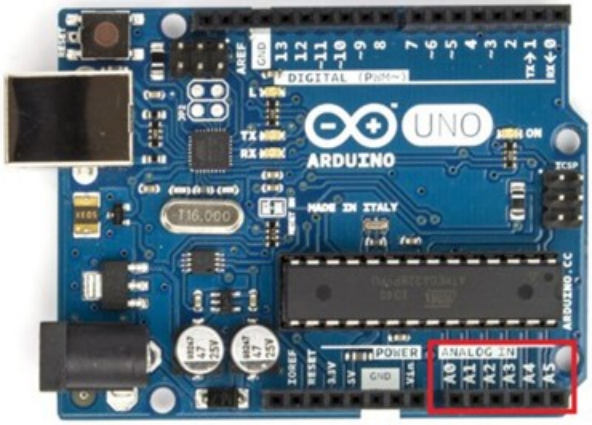
La salida de 3.3 V es capaz de suministrar 150 mA esta salida depende de los 5V, el regulador de 3.3V saca energía de los 5V por lo tanto este consumo se suma a la fuente de 5V y debe considerarlo para no sobrecargar la fuente de 5V.

Es importante mantener los valores eléctricos dentro de los valores recomendados con esto evitará dañar el regulador de voltaje montado en la placa como así también prevenir daños en el propio microcontrolador.

Pines con entradas analógicas y el ADC.

El ATMega328p, al igual que toda la gama ATMega de Atmel y otros microcontroladores, tienen un conversor ADC integrado y no necesita ningún hardware adicional, esto nos permite simplemente conectar un sensor analógico y leer el valor medido por el sensor.

El ADC en microcontroladores AVR utiliza una técnica conocida como aproximación sucesivas donde se compara la tensión de entrada a medir con la mitad de la tensión de referencia generada internamente.
El ADC interno del microcontrolador tiene una resolución de 10 bits, esto significa que la tensión analógica de entrada se convierte en un valor numérico entre 0 y 1023.
Una señal analógica es aquella en la que los valores de la tensión o voltaje varían constantemente y pueden tomar cualquier valor. En el caso de la corriente alterna, la señal analógica incrementa su valor con signo eléctrico positivo (+) durante medio ciclo y disminuye a continuación con signo eléctrico negativo (–) en el medio ciclo siguiente.
Un sistema de control con un microcontrolador, no tiene capacidad alguna para trabajar con señales analógicas, de modo que necesita convertir las señales analógicas en señales digitales para poder trabajar con ellas, es aquí donde aparece el *Conversor Analógico Digital* (ADC) responsable de esta tarea. Arduino UNO tiene una serie de pines asignados para este trabajo, A0, A1, A2, A3, A4 y A5.



Pines analógicos en la placa Arduino Uno.

Estos pines se pueden conectar internamente al módulo del conversor analógico, el microcontrolador tiene un solo módulo y estas entradas se seleccionan con una llave digital interna (multiplexador) para conectar una determinada entrada analógica al módulo analógico y así poder hacer una medición en ese canal analógico.
En el caso de un Arduino Uno, el valor de 0 voltios analógico es expresado en digital como 0x00 y el valor de 5V analógico es expresado en digital como 0x3F (1023).
Es decir que toda medición tendrá un número comprendido entre 0 y 1023, resolución de 10 bits.

El sistema de memoria de Arduino.

Un microcontrolador usado en Arduino tienen varios tipos de memoria integradas en el mismo chip, en el caso de Arduino y los microcontroladores AVR de Atmel usan tres tipos de memorias.
Memoria RAM para el manejo de variables, igual que en una computadora.
La memoria FLASH o memoria de programa, algo como el disco duro de la computadora, y la memoria EEPROM, memoria para el manejo de variables pero que no se pierden cuando el controlador se apaga, su uso es similar al de una memoria SD.

Arduino	Processor	Flash	SRAM	EEPROM
UNO, Uno Ethernet, Menta, Boarduino	Atmega328	32K	2K	1K
Leonardo, Micro, Flora, 32U4 Breakout, Teensy, Esplora	Atmega 32U4	32K	2.5K	1K
Mega, MegaADK	Atmega2560	256K	8K	4K

- ***SRAM :*** *Variables locales, datos parciales. Es la zona de memoria donde el sketch crea y manipula las variables cuando se ejecuta. Es un recurso limitado y debemos supervisar su uso para evitar agotarlo, en un microcontrolador no podemos agregar mas memoria RAM como en una computadora.*
- ***EEPROM:*** *Memoria no volátil para mantener datos después de un reset o cuando el microcontrolador se apaga. Se puede grabar desde el programa del microcontrolador, usualmente, constantes de programa. Las EEPROMs tienen un número limitado de lecturas/escrituras, tener en cuenta a la hora de usarla. Esta memoria solo puede leerse byte a byte y su uso puede se un poco incómodo. También es algo más lenta que la SRAM. La vida útil de la EEPROM es de unos 100.000 ciclos de escritura. En esta memoria se podría almacenar la clave de accesos de una alarma para que el propio usuario pudiera cambiarla sin necesidad de re-programar todo el microcontrolador.*
- ***Flash:*** *Memoria de programa. Usualmente desde 1 Kb a 4 Mb (en controladores de familias grandes). Es donde se guarda el sketch o programa Arduino ya compilado. Sería el equivalente al disco duro de una computadora. En la memoria flash también se almacena del bootloader. Se puede ejecutar un programa desde la memoria flash, pero no es posible modificar los datos, sino que es necesario copiar los datos en la SRAM para modificarlos.*
  La memoria flash usa la misma tecnología que las tarjetas SD, los pen drives o algunos tipos de SSD, esta memoria tiene una vida útil de

unos 100.000 ciclos de escritura, así que cargando 10 programas al día durante 27 años podríamos dañar la memoria flash de manera permanente.

Memoria de Arduino UNO:

- Flash 32k bytes (el bootloader usa 0.5k)
- SRAM 2k bytes
- EEPROM 1k byte

Memoria de Arduino MEGA:

- Flash 256k bytes (el bootloader 8k)
- SRAM 8k bytes
- EEPROM 4k byte

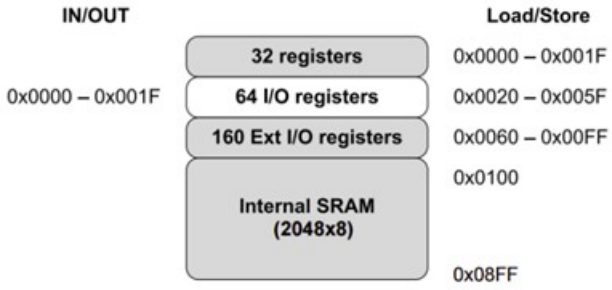
Algunos detalles de la memoria SRAM.

Al ser el recurso más escaso en Arduino hay que entender bien cómo funciona.
La memoria SRAM puede ser leída y escrita desde el programa en ejecución.
La memoria SRAM es usada para varios propósitos:

- ***Static Data:*** *Este bloque de memoria reservado en la SRAM para todas las variables globales y estáticas. Para variables con valores iniciales, el sistema copia el valor inicial desde la flash al iniciar el programa.*
- ***Heap:*** *Es usado para las variables o elementos que asignan memoria dinámicamente. Crece desde el final de la zona de Static Data a medida que la memoria es asignada. Usada por elementos como los objetos y los Strings.*
- ***Stack:*** *Es usado por las variables locales y para mantener un registro de las interrupciones y las llamadas a funciones. La pila crece desde la zona más alta de memoria hacia el Heap. Cada interrupción, llamada de una función o llamada de una variable local produce el crecimiento de la memoria.*
  La mayor parte de los problemas ocurren cuando la pila y el Heap colisionan.
  Cuando esto ocurre una o ambas zonas de memoria se corrompen con resultados impredecibles. En algunos casos se produce un "cuelgue" del programa y en otros casos la corrupción de memoria puede deriva en funcionamientos erráticos.

A partir de la versión 1.6 del IDE al compilar un sketch nos da el tamaño que va a ocupar en la flash el proyecto y el espacio que va a ocupar en la SRAM las variables globales, es decir la zona de static data.
Recuerde que en la memoria SRAM también se encuentran los registros que ocupan las primeras 256 direcciones de memoria. Por lo tanto la memoria

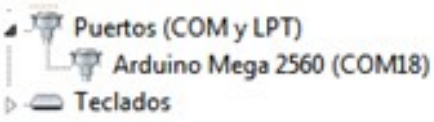
SRAM para el uso de variables del usuario empieza a partir de la dirección 0x0100.
Es decir que la memoria SRAM no esta disponible en su totalidad para el programa del usuario, el propio microcontrolador usa una parte para mantener su propio funcionamiento.
Los registros al estar en la SRAM son volátiles y no conservan su valor después de un reset.
Esto es importante porque ante un fallo de alimentación, corte transitorio, etc. Los registros de memoria pueden iniciar con datos no válidos o corruptos.
Mirando la documentación del microcontrolador se puede ver las direcciones asignadas a las regiones de memoria.



Mapa de memoria para Arduino Uno.

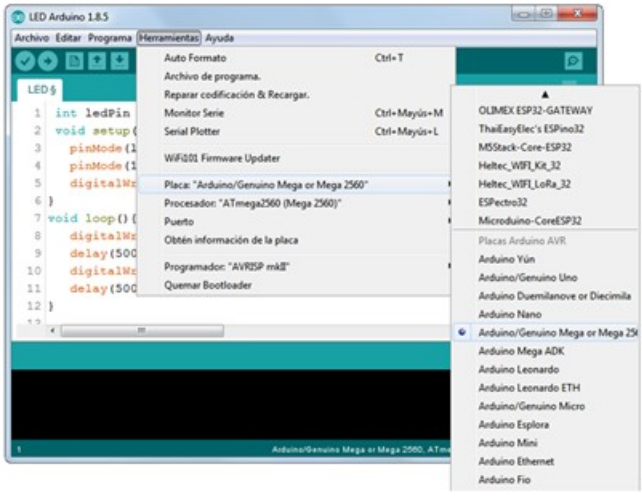
A su vez la memoria SRAM tiene regiones para la gestión de datos variables, la administración de estas regiones se realiza en forma automática pero también depende de la forma en que declaramos una variable en un programa pues esta declaración define en que región de memoria se almacenará.
La primera sección de RAM y se usa para almacenar datos estáticos del programa, como cadenas, estructuras inicializadas y variables globales.
Las variables .bss son la memoria asignada para variables globales y estáticas no inicializadas.
Heap es el área de memoria dinámica.
La pila es el área de memoria ubicada al final de la RAM y crece hacia el área de datos. El área de pila se usa para almacenar el estado del microcontrolador al saltar a una subrutina y recuperarlo luego, valores de retorno en interrupciones, etc. Como ocurre con todos los microcontroladores el manejo inadecuado del Stack o pila puede traer problemas serios ya que si el Stack crece de manera descontrolada destruirá las otras variables del programa y todo el sistema colapsa.
Es importante que siempre exista suficiente memoria libre entre el Stack y el bloque de memoria de datos del usuario. Cuando esta área es demasiado

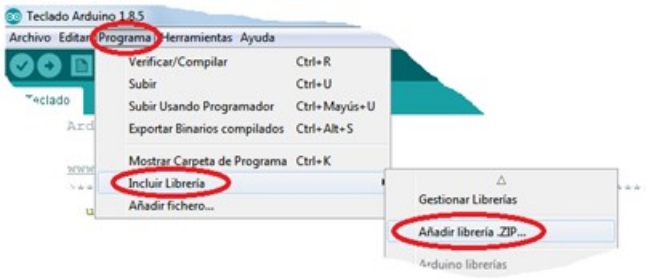
que no necesita ser instalada, se descarga un archivo zip se descomprime y todas las herramientas quedan listas para su uso.
El entorno de trabajo de Arduino es bastante austero si lo comparamos con otros entornos para otras arquitecturas sin embargo es totalmente funcional. Se puede escribir un programa en el, depurarlo e incluso tiene todo lo necesario para programar la memoria del microcontrolador.
Las placas Arduinos son reconocidas por el entorno de trabajo de manera automática si se trata de un Arduino original, conectamos la placa al puerto USB y luego simplemente la seleccionamos de la lista de placas disponibles. Por ejemplo si estamos trabajando con Arduino Mega, en la sección herramientas seleccionamos la placa que corresponde.



Selección de puertos para programar Arduino.

Desde el administrador de dispositivos estando la placa conectado debemos ver algo como la imagen anterior. Si la placa no es reconocida (puede suceder con Mega) cargar los drivers manualmente indicando a Windows que los busque en la carpeta drivers dentro de la carpeta Arduino. (Todos los drivers para las placas originales están en esa carpeta). Normalmente todo lo necesario para trabajar lo tenemos en el propio IDE o dentro de las carpetas que se crean cuando el programa se instala. Para las placas Arduino Compatibles suele ser necesario buscar en la red el driver adecuado.





También se puede instalar directamente desde Internet en el apartado "*Gestionar Librerías*".

Ventajas de Arduino.

- Sin duda una de las grandes ventajas es su bajo costo, con una inversión muy baja tenemos una placa completa y funcional.
- Facilidad de programación y total abstracción del hardware involucrado.
- Gran cantidad de bibliotecas y controladores para distintos periféricos que se puden conetar a la placa.
- Una enorme comunidad de programadores y simpatizantes de Arduino para consultas e intercambio de códigos.

Desventajas de Arduino.

- Considerando que Arduino no fue pensado para desarrollos comerciales no permite la protección de código en su memoria de programa. Esto significa que eventualmente nuestro trabajo queda expuesto.
- El entorno de programación es bastante pobre en cuanto a funciones y herramientas para depurar el código.
- Muchas de las bibliotecas de Arduino no fueron pensadas en función de optimizar el uso de recursos.

Programando Arduino UNO.

El clásico "Hola Mundo" en la programación de microcontroladores es lograr encender y apagar un LED conectado a un pin del controlador.
En el caso de la placa Arduino UNO,ya tiene un LED conectado al pin 13 por lo tanto no necesitamos agregar nada de electrónica, solo escribir el sketch y programar la memoria de Flash.

```
/*********************************************************
Enciende y apaga un LED conectado en el pin 13 al ritmo de un
```

```
segundo.
*************************************************************/
                    CONFIGURACIÓN DEL HARDWARE
void setup(){
   pinMode(13, OUTPUT);
// El pin 13 será una salida digital
   digitalWrite(13,LOW );
// El pin inicia con nivel bajo.
}
                    BUCLE PRINCIPAL DEL PROGRAMA
void loop(){
   digitalWrite(13, HIGH);
// Enciende el LED
   delay(1000);
// Pausa de 1 segundo
   digitalWrite(13, LOW);
// Apaga el LED
   delay(1000);
 // Pausa de 1 segundo
}
/****** Fin de Programa FIRTEC ARGENTINA *********/
```

En el programa anterior la función *pinMode(13,OUTPUT)* es la encargada de configurar el pin como salida y *digitalWrite(13,HIGH)* escribe un “1 (HIGH)” o un “0 (LOW)” en el pin especificado.
La función delay() es la encargada de las esperas de tiempo, el argumento 1000 especifica mil milisegundos (1 Segundo) .
Esta función merece una aclaración adicional, su acción debe ser entendida como “Perder tiempo sin hacer nada”, cuidado con esto, mientras el controlador espera no hace nada mas y todo lo que suceda mientras espera será ignorado, mas bien no será visto. Mientras el controlador está en un delay() es ciego y sordo a lo que pasa en su entorno (pines).
No es práctica prolija el abusar de los delay() y solo se deben emplear para tiempos muy cortos y en secciones del programa que sabemos no hay eventos pendientes de suceder.
En el programa anterior el controlador solo enciende y apaga un LED, no hace nada mas y por lo tanto los tiempos muertos del delay() son irrelevantes pero solo en este caso.

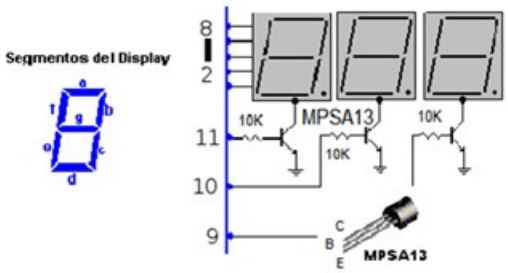
La función millis().

Como vimos en el ejemplo anterior, el uso del delay() para obtener tiempos de espera puede generar complicaciones a la hora de escribir programas donde se esperan eventos que pueden ocurrir mientras el controlador ejecuta el delay(), por ejemplo evaluar el estado de un pin para ver si la señal de un sensor está presente.

```
void loop(){
  estado = digitalRead(boton);
 // Mira el estado del botón
  if (estado == LOW && bandera== 0) {
     contador++;
     if(contador == 10)
        contador =0;
      Muestra_Dato(contador);
      bandera = 1;
  }
  estado = digitalRead(boton);
// Mira el estado del botón
  if (estado == HIGH)
      bandera = 0;
}
                    Muestra el estado de cuenta
void Muestra_Dato(unsigned char number){
  unsigned char pin = 2;
  for (unsigned char j=0; j < 7; j++) {
   digitalWrite(pin, num_array[number][j]);
   pin++;
  }
}
/******* Fin de Programa FIRTEC ARGENTINA ******/
```

Contador de tres dígitos.

Para el siguiente ejemplo vamos a necesitar la siguiente electrónica conectada a nuestra placa Arduino.



Circuito electrónico para el ejemplo propuesto.

Tres dígitos cátodo común controlados por tres transistores que forman un multiplexador serán los encargados de mostrar el estado de cuenta. En este caso el contador cuenta a su propio ritmo de manera automática no hay botón de cuenta. El lector lo puede agregar para contar objetos con un sensor, o contar paquetes en una cinta transportadora.

Los tres dígitos se encuentran conectados en paralelo es decir segmento “a” de uno al segmento “a” del otro y así sucesivamente. Los pines usados para los segmentos son los mismos que en el ejemplo anterior, pines del 2 al 8.
El contador es básicamente un registro, una posición de memoria que se incrementa, si la variable contador ha sido definida como unsigned char o byte solo puede contar hasta 255, esta cuenta se hace en binario es decir una secuencia de números que va desde 00000000 a 11111111, a nivel humano es poco lo que podemos hacer con esto, necesitamos que se lean números decimales.
En el programa encontrará una función destinada a contar que “desarma” este bnario en tres registros Unidad, Decena y Centena.
Estos registros o variables serán mostradas en los dígitos de manera secuencial, primero la Unidad luego la Decena y por último la Centena activando los respectivos transistores a una velocidad tal que el ojo verá las tres cifras como si estuvieran juntas en el mismo instante.
La función de cuenta es la siguiente y se puede ajustar para contar miles o miles de miles si fuera necesario.

```
void Contador(){
    contador++;
    centena = contador / 100;
 // Cuenta cuantas veces centena es mayor que 100.
    resto = contador % 100;
 // Recupera el resto de la división anterior.
    decena = resto /10;
 // Cuentas cuantas veces el resto es mayor que 10.
    unidad = resto % 10;
// Recupera el resto de la división que será la unidad.
  }
```

El siguiente es el código completo del programa.

```
/*************************************************************
Contador de tres dígitos cátodo común. El contador cuenta en
forma automática, el objetivo es ver como se manejan display de
siete segmentos con un multiplexador de tres transistores
* Los segmentos están conectados de la siguiente forma:
* Pin 2 segmento a
* Pin 3 segmento b
* Pin 4 segmento c
* Pin 5 segmento d
* Pin 6 segmento e
* Pin 7 segmento f
* Pin 8 segmento g
*************************************************************/
// ------ Pines de los transistores
int T1 = 9;   // Transistor de la Unidad en pin 9
```

```
int T2 = 10;  // Transistor de la Decena en pin 10
int T3 = 11;  // Transistor de la Centena en pin 11
//----- Codifica los números y segmentos ---------
const unsigned char num_array[10][7] =
  {{ 1,1,1,1,1,1,0 },   // 0
  { 0,1,1,0,0,0,0 },    // 1
  { 1,1,0,1,1,0,1 },    // 2
  { 1,1,1,1,0,0,1 },    // 3
  { 0,1,1,0,0,1,1 },    // 4
  { 1,0,1,1,0,1,1 },    // 5
  { 1,0,1,1,1,1,1 },    // 6
  { 1,1,1,0,0,0,0 },    // 7
  { 1,1,1,1,1,1,1 },    // 8
  { 1,1,1,0,0,1,1 }};   // 9
//---- Se declaran dos funciones ---
void Muestra_Dato(unsigned char);
void Contador(void);
unsigned int retardo =0;
unsigned char decena=0,unidad=0,centena=0,resto=0,contador=0;
```

Configuración del Hardware

```
 void setup(){
  pinMode(13, OUTPUT);   // Pin del LED de la placa
  digitalWrite(13,LOW ); // LED apagado

// -- Configuración de los pines de los transistores
  pinMode(T1, OUTPUT);
  pinMode(T2, OUTPUT);
  pinMode(T3, OUTPUT);
// -- Configuración de los pines para los segmentos
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
}
```

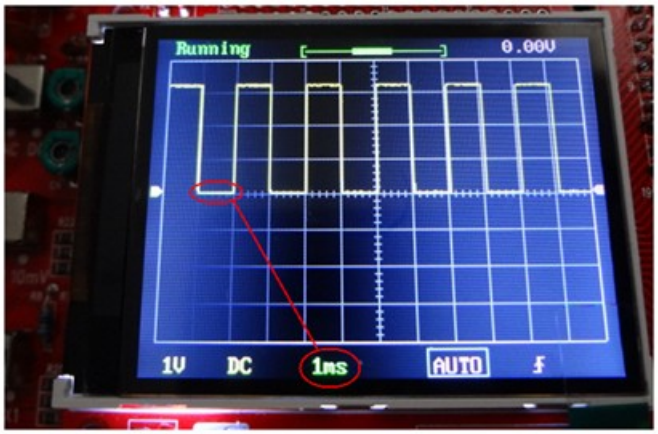
Bucle principal del programa

```
void loop(){
   retardo++;
   if(retardo == 200){
      Contador();
      retardo =0;
   }
   Muestra_Dato(unidad);
   digitalWrite(T1, HIGH);
   delay(1);
   digitalWrite(T1, LOW);
   Muestra_Dato(decena);
   if((decena > 0) | (centena > 0))
```

```
   digitalWrite(T2, HIGH);
   delay(1);
   digitalWrite(T2, LOW);
   Muestra_Dato(centena);
   if(centena >0)
   digitalWrite(T3, HIGH);
   delay(1);
   digitalWrite(T3, LOW);
}
                        Función para contaar
void Contador(){
    contador++;
    centena = contador / 100;
// Se define la centena
    resto = contador % 100;
// Recupera el resto de div.
    decena = resto /10;
// Se define la decena
    unidad = resto % 10;
// Define la unidad
  }
          Función para mostrar el estado del contador
void Muestra_Dato(unsigned char number){
  unsigned char pin = 2;
  for (unsigned char j=0; j < 7; j++) {
   digitalWrite(pin, num_array[number][j]);
   pin++;
  }
}
/********* Fin de Programa FIRTEC ARGENTINA ******/
```

El multiplexador es controlado por el siguiente trozo de código.

```
 Muestra_Dato(unidad);
   digitalWrite(T1, HIGH);
   delay(1);
   digitalWrite(T1, LOW);
   Muestra_Dato(decena);
   if((decena > 0) | (centena > 0))
   digitalWrite(T2, HIGH);
   delay(1);
   digitalWrite(T2, LOW);
   Muestra_Dato(centena);
   if(centena >0)
   digitalWrite(T3, HIGH);
   delay(1);
   digitalWrite(T3, LOW);
```



El tiempo de encendido/apagado de cada dígito es de 1mS.

Observe que se coloca la unidad en los display, se activa el transistor que energiza el display de la unidad por un tiempo de 1 milisegundo, se lo apaga y se pasa la decena repitiendo los tiempos de encendido, debido la persistencia de la imagen anterior en la retina del ojo, este sigue viendo el número anterior como si todavía estuviera presente en el visor y se forma la ilusión óptica que permite ver tres cifras distintas en lo que parece ser el mismo instante.
Si la decena está en cero no se muestra lo mismo que la centena, no tiene sentido mostrar tres ceros generando el consumo de mas segmentos cuando no hay información útil para ver.

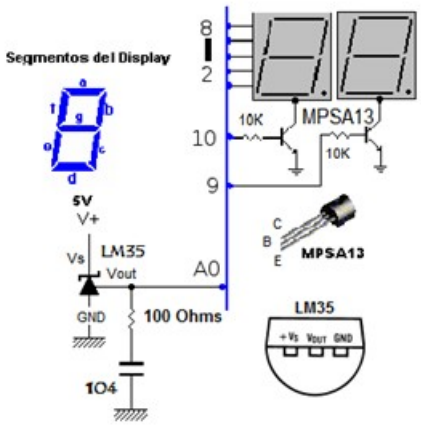
El conversor analógico (ADC).

En muchos proyectos tenemos que tratar con las señales o información del mundo analógico, como la temperatura, presión, corriente, etc .
Estas señales son analógicas de forma predeterminada y en la mayoría de los casos se utilizan sensores que convierten estas señales a analógico de tensión eléctrica que será presentada en un pin del microcontrolador para hacer algún trabajo.
Por desgracia, los microcontroladores son digitales y no pueden hacer frente a las señales analógicas por lo que estas señales deben ser convertidas en señales digitales que sean comprensibles por el núcleo del microcontrolador.
Básicamente digitalizar algo es simplemente convertirlo a una secuencia de números de tal forma que si hacemos el proceso inverso de leer esos números podemos reconstruir la información original.
Para el siguiente ejemplo usaremos la electrónica que se muestra a continuación.

Si bien el conversor del ATmega328 no es de los mas rápidos, es lo suficientemente ágil y preciso para la mayoría de los problemas a resolver con una placa Arduino.

Sensor LM35.

El LM35 es la versión para grados centígrados de un legendario sensor que mide en grados Kelvin, el LM355.
Ambos son parecidos solo que miden en escalas diferentes, **no son reemplazo uno del otro**.
El LM35 tiene un funcionamiento muy sencillo y seguramente encontrará en la red incontables sitios que lo describen en profundidad. Solo diremos que por su salida envía un voltaje que es proporcional a los grados de temperatura a la está expuesta su cápsula.
El voltaje de salida sube o baja a razón de 10 mV por cada grado, 0 mV son cero grados centígrados y 250 mV equivalen a 25 grados C.
Como imaginará su uso con Arduino es bastante simple, solo debemos medir el voltaje que el sensor envía al ADC y hacer unos pequeños cálculos para conocer la temperatura.
En el ejemplo propuesto necesitamos la siguiente electrónica conectada a la placa Arduino.



El condensador de 01 uF y la resistencia de 100 Ohms forman un filtro para eliminar las posibles interferencias que se pudieran captar entre la conexión del sensor y el pin A0 de la placa Arduino.
Recuerde que básicamente está midiendo el voltaje que envía el sensor, si coloca un cable demasiado largo (varios metros) se provoca una caída de tensión en la línea y la medición no será fiable.
El siguiente es el código de funcionamiento para el sensor LM35 y el ADC, básicamente es el mismo trabajo anterior solo se han agregado las rutinas para

```
  muestras =0;
 }
          Función para el encendido de los segmentos.
void Muestra_Dato(unsigned char number){
  unsigned char pin = 2;
  for (unsigned char j=0; j < 7; j++) {
   digitalWrite(pin, num_array[number][j]);
   pin++;
  }
}
/***** Fin de Programa FIRTEC ARGENTINA ******/
```

Interrupciones con Arduino UNO.

Las interrupciones son señales recibidas por el microcontrolador, indicando que debe "interrumpir" el curso de ejecución actual y pasar a ejecutar un código específico para tratar esta situación.
Una interrupción suspende la ejecución temporalmente de un programa, para pasar a ejecutar una subrutina de servicio de interrupción (ISR).
Las interrupciones surgen de las necesidades que tienen los periféricos de enviar información al controlador, por ejemplo el controlador esta ejecutando el programa de una balanza y una línea de interrupción esta conectada a un sensor que mide la presión de vapor de una caldera, en determinado momento la presión del vapor baja, el sensor detecta el problema y activa la interrupción, el controlador interrumpe su tarea y salta a la ISR que trata el evento del vapor.
Las interrupciones son muy importantes cuando se trabaja con microcontroladores, permiten hacer varias cosas al mismo tiempo sin que la propia CPU esté al pendiente de todo el trabajo que hacen los módulos internos, estos pueden realizar una tarea y llamar la atención de la CPU solo cuando la tarea se terminó.
Esto suena bien sin embargo hay un par de cosas a tener en cuenta cuando se trabaja con interrupciones.

1. *Debe entrar y salir rápidamente de una ISR (Rutina de Interrupción). No escriba un largo código para un servicio de interrupción.*
2. *Las interrupciones no son re-entrantes, si está en una no puede activar otra y ninguna otra será atendida mientras no termine la que está en curso.*
3. *Las variables que se usan en una ISR siempre deben ser del tipo volatile.*

Recuerde, si está en una ISR su microcontrolador (su placa Arduino) esta

```
      Muestra_Dato(decena);
      if(decena > 0)
         digitalWrite(T2, HIGH);
       break;
    }
    case 3:{
      digitalWrite(T2, LOW);
      marca =0;
      break;
    }
  }
}
void Leer_Conversor(void){
  M0 += analogRead(A0);
// Lee el A/D y acumula el dato en M0
  NOP;
  NOP;
  NOP;
  muestras++;
// Incrementa el contador de muestras
if(muestras ==49){
// Se tomaron 50 muestras ??
  conversion = M0/50;
// Se busca el promedio de las 50 muestras
  conversion= (conversion * 500) >> 10;
  decena = (conversion / 10) % 10;
  unidad = conversion % 10;
  M0 =0;
  muestras =0;
  }
 }
void Muestra_Dato(unsigned char dato){
  unsigned char pin = 2;
  for (unsigned char j=0; j < 7; j++) {
   digitalWrite(pin, num_array[dato][j]);
   pin++;
  }
}
```

Funcionamiento del Timer 1.

Un temporizador es en esencia un contador que cuenta a la velocidad de CPU. Usando un cristal de 16Mhz como tiene la placa Arduino, el tiempo de CPU es 1/16 = 62,5 nS. Este es el tiempo de una instrucción de un solo ciclo por ejemplo la instrucción NOP.
Y esta es la velocidad de cuenta de un Timer, es decir que cada cuenta toma 62,5 nS, conociendo esto podemos fácilmente calcular tiempos.
Un Timer de 16 bits puede contar desde 0 a 65535 a la velocidad de 62,5 nS por cada número contado. Podemos usar un “modulo”, un registro que le dice

- **TCCR1B** configura el prescalador o divisor de tiempo, CS10, CS11 y CS12 son los bits para ajustar los tiempos de división de acuerdo a la siguiente tabla.

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0		1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	clkI/O/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

- **TIMSK1** es el registro de control donde se encuentra el bit de configuración para activar la interrupción TOIE1.

Para cnfigurar el Timer_1 y generarla señal de 50Hz los pasos serían los siguientes:

```
noInterrupts();           // No hay interrupciones
TCNT1 = 64911;            // Ajusta modulo a ~10mS
TCCR1B |= (1 << CS12);    // Divisor ajustado a 256
TIMSK1 |= (1 << TOIE1);  // Habilita la mascara de interrupción
interrupts();     // Las interrupciones se activan nuevamente.
```

Recordar que se debe definir nuevamente el módulo una vez que la interrupción se activa caso contrario el Timer inicia siempre en cero.
Como se comentó anteriormente, las interrupciones están vectorizadas, cada una de ellas cuando se active buscará el código ISR en una dirección específica. Esta dirección corresponde con la forma en que se escribe el identificador de la ISR.
La siguiente es la lista completa de vectores donde se ha marcado el vector del Timer_1.

Vector No	Program Address(2)	Source	Interrupts definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 0
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2_COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2_COMPB	Timer/Coutner2 Compare Match B
10	0x0012	TIMER2_OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1_CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1_COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1_COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1_OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0_COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0_COMPB	Timer/Coutner0 Compare Match B
17	0x0020	TIMER0_OVF	Timer/Counter0 Overflow
18	0x0022	SPI STC	SPI Serial Transfer Complete
19	0x0024	USART_RX	USART Rx Complete
20	0x0026	USART_UDRE	USART Data Register Empty
21	0x0028	USART_TX	USART Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface (I^2C)
26	0x0032	SPM READY	Store Program Memory Ready

INT0 y INT1.

El microcontrolador ATMega328 tiene dos pines asignados a interrupciones por hardware externo, en la placa Arduino estos dos pines son el pin 2 y 3. Son dos interrupciones altamente configurables y muy útiles a la hora de detectar eventos por cualquiera de esos pines.

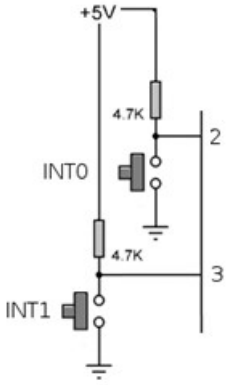
Se pueden configurar para que la interrupción se active por un cambio de bajo a alto (flanco de subida) o cambio de alto a bajo (flanco de bajada).

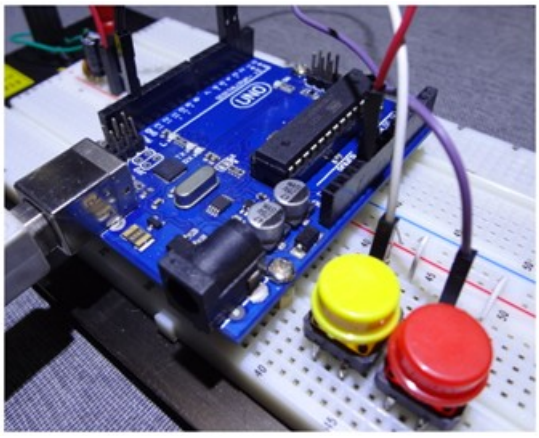
Para su uso solo basta con usar la función de configuración para cada pin. Para configurar el pin 2 que corresponde a INT0 por un flanco de bajada necesitamos escribir lo siguiente *attachInterrupt(0, ISR_BotonCuenta, LOW)* donde 0 es INT0, *ISR_BotonCuenta* es el nombre del servicio de interrupción y *LOW* indica que se activa por flanco de bajada. O mismo es para INT1 donde solo cambia el 0 por el 1 y el nombre del servicio ISR, attachInterrupt(1, ISR_BotonReset, LOW).

En el ejemplo propuesto para verificar el funcionamiento de estas dos interrupciones vamos crear un contador controlado por dos botones, el botón en INT0 será encargado de incrementar el contador y el botón en INT1 pondrá a cero el contador.

En este ejemplo el estado del contador lo enviamos por la UART que al estar conectados por el cable USB ya podemos recibir los datos en el terminal del

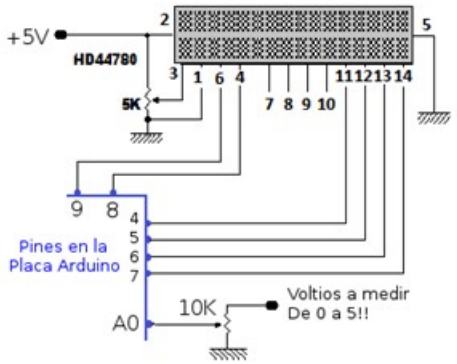
propio IDE Arduino, para usar el terminal fácilmente lo tenemos disponible desde el icono que parece una lupa en el margen superior derecho del IDE de trabajo o también lo podemos llamar desde el menú Herramientas.
Para verificar el funcionamiento del ejemplo necesitamos la siguiente electrónica conectada a nuestra placa Arduino.





La electrónica usada es muy sencilla, y el sketch completo es el siguiente.

```
/************************************************************
*  Muestra el uso de las interrupciones INT0 y INT1 para los
pines 2 y 3 del Arduino UNO. Las interrupciones se han
configurado para ser activas con un flanco de bajada.
*  En la configuración de los pines se han activado las
resistencias PULLUP que llevan el pin a nivel alto (contrario
al disparo).
*  El botón en el pin 2 incrementa el contador, el botón del
pin 3 pone a cero el contador.
*  En la ISR del botón de cuenta se comprueba si el valor
actual de millis es mayor en 20 ms a la última vez que paso por
```




```
/************************************************************
* Lectura de AN0 usando el ADC mediante
* interrupción del conversor
* Los datos se muestran en una pantalla LCD 16*2
************************************************************/
#include "LiquidCrystal.h"
#define ledPin 13
const byte adc_Pin = 0; // Canal donde mide A0
volatile unsigned char muestras=0;
volatile unsigned int M0=0;
volatile float conversion;
volatile char* buffn="";
char buffer[10]=" ";
// Pines del LCD RS E  datos
LiquidCrystal lcd(8,9,4,5,6,7);

void setup (){
  pinMode(A0, INPUT);
  lcd.begin(16,2);
  pinMode(ledPin, OUTPUT);
// Pin del LED de la placa
  digitalWrite(13,LOW );   // LED inicia apagado
  ADCSRA =  bit (ADEN);    // ADC activado
  ADCSRA |= bit (ADPS0) |  bit (ADPS1) | bit (ADPS2);
 // Prescaler en 128
  ADMUX  =  bit (REFS0) | (adc_Pin & 0x07);
// Todos los pines
  lcd.setCursor(0,0);
  lcd.print("FIRTEC ARGENTINA");
// Muestra cartel 1
  lcd.setCursor(0,1);
  lcd.print("Voltios: ");
// Muestra cartel 2
 }
ISR (ADC_vect){
```

```
  M0 += ADC;
// Acumula las conversiones en M0.
  Muestras++;
// Incrementa el contador de muestras.
  if(muestras == 50){
// Se tomaron 50 muestras?
      conversion = M0/50;
// Busca el promedio
      conversion = (conversion *5.0)/1024;
// Escala la conv.
      dtostrf(conversion,2,2,buffn);
// Dato a cadena
      M0 =0;
      muestras =0;
      digitalWrite(ledPin, digitalRead(ledPin) ^ 1);
   }
  }
void loop (){
    sprintf(buffer,"%s", buffn);
// Ajusta la cadena a mostrar
    lcd.setCursor(8,1);
// Mueve el cursor hasta la ubicación
    lcd.print(buffer);
// Muestra el voltaje en pantalla
    ADCSRA |= bit (ADSC) | bit (ADIE);
// Inicia nueva conv.
}
/********** Fin de Programa FIRTEC ARGENTINA *****/
```

NOTA I:
Recuerde que dentro de una interrupción las variables usadas deben ser del tipo volatile. Usar variables convencionales puede dar lugar a funcionamientos erráticos.
NOTA II:
La función clásica de ANSI-C sprintf() en su uso convencional no es soportada por Arduino, por lo que hay que hacer una serie de pasos para poder usarla. Un movimiento un tanto redundante de cadenas sobre cadenas pero como el uso de sprintf() resuelve varios puntos en la representación de información en pantallas LCD vale la pena su uso.

Midiendo con el ADC por cuatro canales.

Habiendo visto la medición en un canal veamos un ejemplo que mide por cuatro canales, a la electrónica del ejemplo anterior solo tenemos que agregar tres potenciómetros mas conectados a los pines A1, A2 y A3. El resultado del ejemplo funcionando es como se aprecia en el siguiente imagen.

```
  lcd.print("Temp en Grados");
// Muestra carteles iniciales
  lcd.setCursor(0,2);
  lcd.print("----[      ]----");
 }
void loop (){
   // Envía el comando para medir la temperatura
   sensors.requestTemperatures();
   // Obtiene la temperatura en °C
   float temp= sensors.getTempCByIndex(0);
   dtostrf(temp,2,1,buffn);
   sprintf(buffer,"%s", buffn);
   lcd.setCursor(6,1);
   lcd.print(buffer);
   delay(100);
}
```

Notará que en el programa se envían comandos, esto es así porque el sensor DS18B20 es básicamente un microcontrolador y como todo microcontrolador recibe y procesa comandos igual que la placa Arduino. Las bibliotecas se encargan de hacer transparente al programador toda la gestión de estos comandos.
Podemos encontrar tres tipos básicos de sensores, el DS1820 similar al DS18S20 y el DS18B20, los dos primeros miden con saltos de 0.5 grados (24.0, 24.5, 26.0, 26.5), el DS18B20 tiene más exactitud generando datos 24.0, 24.1, 24.2, etc.

Obteniendo el ID de un sensor DS18x20.

Estos sensores tiene un ID de 64 bits que permite direccionarlos en una red con varios sensores.
Es interesante poder conocer esta ID particular de cada sensor, el ejemplo siguiente tiene la misma electrónica solo que en lugar de obtener solo la temperatura, leemos también su identificador de 64 bits.
El siguiente es el código que extrae el ID.

```
/*********************************************************
* Muestra el ID de 64 bits (8 Bytes) del sensor
* DS18B20. También muestra la temperatura en
* grados.
* Los datos se muestran en pantalla 16x2
*********************************************************/
#include <OneWire.h>
#include <DallasTemperature.h>
OneWire ourWire(2);
//Se establece el pin 2  como bus 1-Wire
DallasTemperature sensors(&ourWire);
#include "LiquidCrystal.h"
```

```
   sensors.requestTemperatures();
// Lee la temperatura por el ID del sensor
   float temp= sensors.getTempC(ID_1);
   dtostrf(temp,2,1,buffn);
   sprintf(buffer,"%s", buffn);
   lcd.setCursor(12,1);
   lcd.print(buffer);
   delay(100);
 }
}
```

Medición de temperatura y humedad con DHT22.

El DHT11 y el DHT22 son dos modelos de una misma familia de sensores, que permiten realizar la medición simultánea de temperatura y humedad**.** Estos sensores disponen de un procesador interno que realiza el proceso de medición, proporcionando la medición mediante una señal digital, por lo que resulta muy sencillo obtener la medición desde una placa como Arduino**.** Ambos sensores presentan un encapsulado de plástico similar. Podemos distinguir ambos modelos por el color del mismo**,** el DHT11 tiene una carcasa azul, mientras que en el caso del sensor DHT22 el exterior es blanco.
De ambos modelos, el DHT11 es el hermano pequeño de la familia, y es bastante inferior en prestaciones que su hermano mayor el DHT22, modelo bastante superior pero tiene un precio también más alto.
El DHT22 tiene las siguientes características:

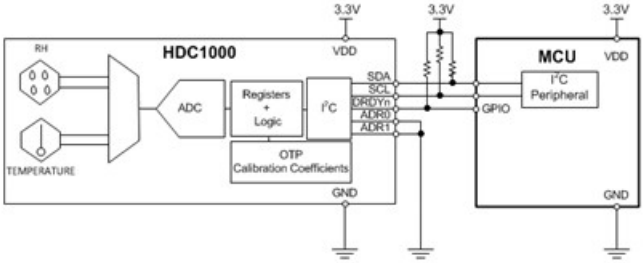
- Medición de temperatura entre -40 a 125, con una precisión de 0.5ºC
- Medición de humedad entre 0 a 100%, con precisión del 2-5%.
- Frecuencia de muestreo de 2 muestras por segundo (2 Hz)

EL DHT22 que sin llegar a ser un sensor de alta precisión tiene unas características aceptables para su uso en proyectos de domótica o control que requieran una precisión media.
Los sensores DHT11 y DHT22 usan su propio protocolo de comunicación bidireccional mediante un único conductor, empleando señales temporizadas. Una filosofía similar a 1-Wire sin embargo los protocolos son totalmente diferentes
En cada envío de medición el sensor envía un total de 40bits, en 4ms. Estos 40 bits corresponden con 2 Bytes para la medición de humedad, 2 Bytes para la medición de temperatura, más un Byte final para la comprobación de errores (8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit check sum)
Podemos leer los datos del sensor directamente generando y leyendo las señales temporizadas según el protocolo del DHTxx lo cual presupone un arduo trabajo en la programación.

Si toma mediciones muy rápidas el sensor no responderá enviando códigos de error o simplemente no midiendo.

Medición de temperatura y humedad con HDC1000.

Con este sensor podemos medir temperatura con un rango que va desde -40 grados a +125 grados y humedad de 0 a 100%. Utiliza un conversor analógico de 14 bits lo que da una precisión mas que aceptable y sin duda superior al sensor DHT22, es la alternativa cuando lo que se necesita es un grado de exactitud mas alto.

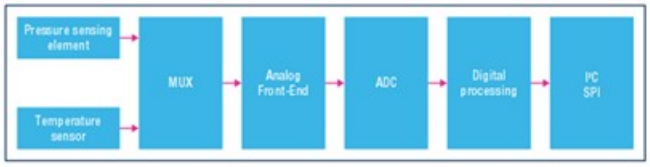


Desarrollado por Texas Instruments y ha sido calibrado en fábrica por lo que no requiere ajuste alguno.
Este sensor tiene una serie de registros de 16 bits para su configuración y uso. En la dirección 0x00 se guardan dos Bytes que corresponden a la temperatura y en la dirección 0x01 dos Bytes para la humedad.
En nuestro ejemplo y para simplificar las cosas, estamos usando un sensor ya montado en una placa impresa construido por MikroElektronika y por su bajo costo no justifica construirla. (Sin embargo el fabricante de la placa brinda todos los diagramas de conexionado).
Para el manejo de este sensor se ha usado la siguiente biblioteca *https://github.com/hotchpotch/Arduino-HDC1000/archive/master.zip* para el HDC1000.
El sensor utiliza el formato de MikroBUS de Mikroelektronika y para facilitar su conexionado vamos a usar un Shield para Arduino UNO que da soporte a MikroBUS.

El resultado del código anterior se puede ver en la imagen. Temperatura y humedad son enviados por el puerto UART.

Sensor Barométrico LPS25HB.

El LPS25HB es un sensor piezo-resistivo para medir presión absoluta que funciona como barómetro.
El dispositivo se puede comunicar mediante el protocolo I2C o SPI, en la imagen siguiente se puede ver el diagrama de bloques internos de este sensor.



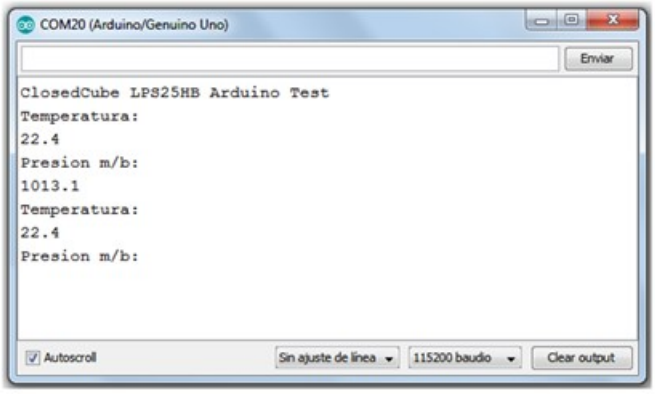
Este sensor es de uso muy popular y se lo encuentra incluso en muchos de los actuales teléfonos celulares y tablets. Algunas de sus características más relevantes son las siguientes:

- Rango de presión de 260 a 1260 hPa.
- High-resolution mode: 0.01 hPa RMS
- High overpressure capability: 20x full scale.
- Embedded temperature compensation
- 24-bit pressure data output
- ODR from 1 Hz to 25 Hz
- SPI and I^2 C interfaces
- Embedded FIFO
- Interrupt functions: Data Ready, FIFO flags, pressure thresholds
- Supply voltage: 1.7 to 3.6 V
- High shock survivability: 10,000 g
- ECOPACK® lead-free compliant

Este sensor también mide la temperatura, si bien utiliza esta medición internamente en el cálculo de presión, su valor está disponible para poder usarlo.

Al igual que el sensor HDC1000, el sensor que probamos tiene el formato MikroBUS ya montado sobre una placa con toda su electrónica.
El siguiente es el código para el manejo de este sensor.

```
/************************************************************
** Descripción: Manejo del sensor LPS25HB para medir presión y
** temperatura. El sensor se conecta al puerto I2C de la placa
** Arduino. Pines SDA y SCL.
** Target        : Arduino UNO
** ToolChain     : Arduino IDE 1.8.12
** https://www.firtec.com.ar/
************************************************************/
#include <Wire.h>
#include "ClosedCube_LPS25HB.h"
char* buffn="";
ClosedCube_LPS25HB lps25hb;
void setup(){
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  Serial.begin(115200);
  Serial.println("ClosedCube LPS25HB Arduino Test");
  lps25hb.begin(0x5D);
}
void loop(){
  float t = 0.0, p = 0.00;
  t = lps25hb.readTemperature();
  p = lps25hb.readPressure();
  dtostrf(t,2,1,buffn);   // Procesa la temperatura
  Serial.println("Temperatura:");
  Serial.println(buffn);
  dtostrf(p,4,1,buffn);    // Procesa la presion
  Serial.println("Presion m/b:");
  Serial.println(buffn);
  delay(2000);
}
//****** Fin del archivo Firtec Argentina ********
```



El resultado práctico del código anterior es el que se aprecia en la imagen anterior.

Medición del índice de Radiación Ultravioleta.

Puede resultar interesante contar con un dispositivo que nos informe la intensidad de la radiación ultravioleta, por ejemplo en los días de verano cuando las playas se llenan de gente informar el índice de radiación UV para protegerse adecuadamente del sol.
El ejemplo propuesto funciona con el nuevo sensor SI1145 de SiLabs con un algoritmo de detección de luz ya calibrado que puede calcular el índice UV.
En realidad no contiene un elemento de detección de UV real, sino que lo calcula a partir de la radiación infrarroja y la cantidad de luz visible del sol.
Es un sensor realmente muy preciso, digital que funciona con I2C, por lo que cualquier microcontrolador puede usarlo.
Como el sensor también tiene elementos para la detección del radiación infrarroja y luz visible, es posible cuantificar estas variables, sin embargo se remarca que los algoritmos internos de este sensor están ajustados para la medición del índice de radiación ultravioleta.
Los resultados son realmente interesantes, comprobará el lector las diferentes lecturas de radiación UV incluso midiendo a través del cristal de una ventana se aprecia como se filtran los rayos UV.
Las escalas, advertencias o alarmas se ajustarán de acuerdo a la necesidad, este ejemplo solo pretende mostrar el uso de este sensor.
El sensor probado está ensamblado por Adafruit y para usarlo necesitamos descargar la correspondiente librería desde Adafruit.

El código para su funcionamiento es el siguiente.

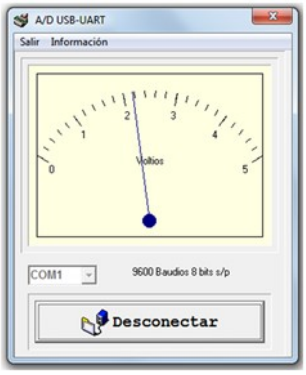
```
/*************************************************************
** Descripción  : Manejo del sensor SI1145 para medir el índice
** de Radiación Ultravioleta.
** El sensor se conecta al puerto I2C de la placa Arduino.
** Conexiones: Pin: Vin +5V GND Pines SDA y SCL.
** Target       : Arduino UNO
** ToolChain    : Arduino IDE 1.8.12
** www.firtec.com.ar
```

```
  if(RX_Byte!=0x08 & a <16){
// Todo esta bien para mostrar?
    lcd.setCursor(base,1);
// Mueve el cursor
    lcd.print(RX_Byte);
// Muestra carácter recibido
    mySerial.write(RX_Byte);
// Eco del carácter
    base++;
// Desplaza el cursor
    a++;
// Contador de caracteres
    }
  }
}
//******** Fin del archivo Firtec Argentina ****
```

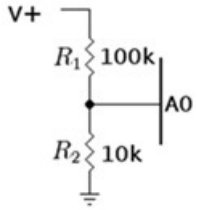
Si en lugar de un puente USB-RS-232 usamos un enlace Bluetooth, e instalamos un Hiper Terminal Android en nuestro móvil podremos hacer lo mismo pero desde el móvil a la placa Arduino.
El protocolo RS-232 tiene “parientes” cercanos como son el RS-422 y RS-485, protocolos de uso industrial por par diferencial que tienen longitudes de cables superiores a los 1000 metros y una alta inmunidad al ruido eléctrico.

Voltímetro UART.

El siguiente ejemplo envía por el puerto UART los datos de un canal analógico para ser mostrados en un instrumento virtual en la computadora.



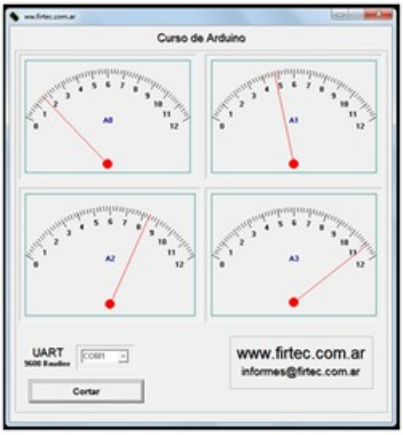
En la imagen anterior se puede ver el resultado al ejecutar el ejemplo propuesto. El instrumento virtual “pide” los datos bajo ciertas condiciones. Al enviar el dato 0XFE espera recibir el byte menos significativo de la conversión, para solicitar el byte mas significativo envía al Arduino el dato



Este atenuador conecta el voltaje a medir con la entrada analógica A0. El valor de voltaje máximo que se podría aplicar a la entrada esta determinado por la siguiente fórmula:
Vmax = 5.0/(R2/(R1+R2)), aproximadamente 55 voltios.

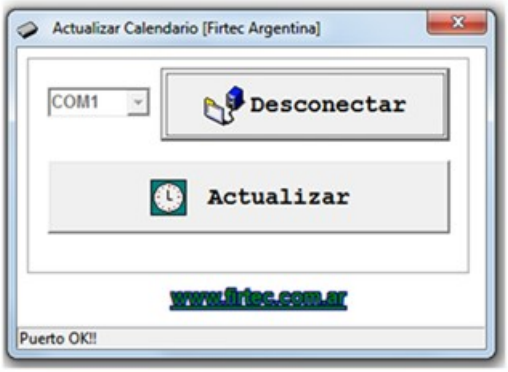
El software informático utiliza el siguiente algoritmo para obtener el voltaje que muestra el instrumento.
Voltaje = (entrada_A0 * 5.5)/1024, Voltaje_Final = (Voltaje/R2(R1+R2)).

Note que siempre suponemos que los valores de las resistencias son exactas por lo que el error de lectura estará determinado por el error que tengan las resistencias en sus valores en Ohmios.
También podríamos leer el voltaje en las entradas A0, A1, A2 y A3 y mostrar los datos en otro instrumento virtual.



Necesitamos cuatro divisores resistivo uno para cada canal analógico. El código de funcionamiento es el siguiente.

```
/*************************************************************
* Lee cuatro canales analógicos y envía datos por
* el puerto UART.
* Utilizar la aplicación Conversor2.exe
* Descarga desde
            https://github.com/firtec/Arduino_Programas
```



Esta aplicación se conecta al programa Arduino que vemos seguidamente y actualiza los registros del chip DS3231 con los valores del calendario que tenga la computadora donde se ejecuta el programa.

```
/************************************************************
* Descripción: Ajusta el DS3231 mediante el puerto
* UART y Software de ajuste.
* Placa Arduino: Arduino UNO
* Arduino IDE: 1.8.19
* Utilizar la aplicación DS3231.exe
* Descargar desde:
*          https://github.com/firtec/Arduino_Programas
************************************************************/
#include <Wire.h> // Biblioteca para 1-Wire
#include "RTClib.h"
// Biblioteca para el RTC
RTC_DS3231 rtc;
char daysOfTheWeek[7][12] = {"Domingo", "Lunes", "Martes",
"Miercoles", "Jueves", "Viernes", "Sabado"};
byte datos[6];
bool bandera = false;
byte rx = 0, offset = 0;
void setup (){
Serial.begin(9600);
delay(3000);
if (!rtc.begin()) {
Serial.println("Opsss!!!... no encuentro el RTC!!!");
while (1);
}
}
void loop () {
if (Serial.available()){ // Espera datos
bandera = true; // Se ha recibido un dato
rx = (uint8_t)Serial.read();
datos[offset] = rx; // Los datos se almacenan en datos
offset++; // Incrementa el offset de datos
if(offset == 6){ // se recibieron seis datos?
```

```
// Actualizar el calendario
rtc.adjust(DateTime(datos[0] + 2000, datos[1], datos[2],
datos[3], datos[4], datos[5]));
offset = 0;
bandera = false;
}
}
if(bandera == false){
DateTime now = rtc.now();
Serial.print(now.year(), DEC);
Serial.print('/');
Serial.print(now.month(), DEC);
Serial.print('/');
Serial.print(now.day(), DEC);
Serial.print(" (");
Serial.print(daysOfTheWeek[now.dayOfTheWeek()]);
Serial.print(") ");
Serial.print(now.hour(), DEC);
Serial.print(':');
Serial.print(now.minute(), DEC);
Serial.print(':');
Serial.print(now.second(), DEC);
Serial.println();
delay(2000);
}}
```

Lora Radio.

LoRa es una comunicación inalámbrica de datos, tecnología desarrollada por la empresa Semtech.
LoRa utiliza bandas de radio que no requieren licencias en rangos de frecuencias de 169 MHz, 433 MHz, 868 MHz y 915 MHz.
Permite transmisiones de muy larga distancias (más de 10 km en zonas rurales), con bajo consumo de energía y no muchas exigencias de antenas, la diferencia con otras tecnologías (WiFi,Bluetooth, etc) lo que diferencia el enlace Lora es el mayor alcance.
Lora Radio es una tecnología de uso creciente en IOT y su implementación es bastante simple con Arduino gracias a las librerías disponibles.
El siguiente ejemplo envía mediante mediante un enlace de Radio Lora el estado de un contador, el emisor esta soportado por un microcontrolador Arduino Uno y el receptor tiene un Arduino Mega.

Las conexiones en el receptor controlado por Arduino Mega las conexiones son las siguientes.

Para usar los módulos Lora Radio de Adafruit necesitamos instalar dos librerías.

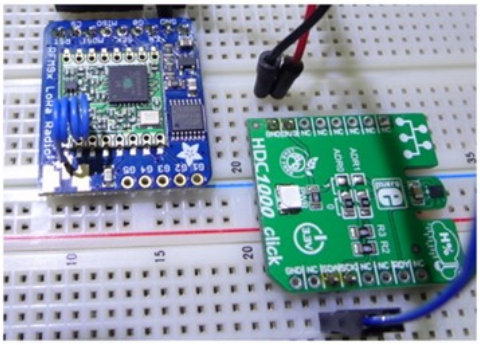
El código completo del transmisor para Arduino UNO es el siguiente.

```
/*********************************************************
* Ejemplo para un transmisor de Radio Lora con
* Arduino UNO
* Usando la placa Lora Radio RFM9x de Adafruit.
* Placa Arduino: UNO R3
*********************************************************/
#include <SPI.h>
// Archivo de cabecera para el puerto SPI
#include <RH_RF95.h>
// Archivo de cabecera para el driver
#define RFM95_INT 3
#define RFM95_CS 4
#define RFM95_RST 2
#define LED 7
#define RF95_FREQ 915.0
// Frecuencia del enlace de radio
RH_RF95 rf95(RFM95_CS, RFM95_INT);
void setup() {
pinMode(RFM95_RST, OUTPUT);
digitalWrite(RFM95_RST, HIGH);
Serial.begin(115200);
while (!Serial) {
delay(1);
}
delay(100);
Serial.println("LoRa TX Test!");
digitalWrite(RFM95_RST, LOW);
```

```
uint8_t len = sizeof(buf);
void loop(){
if (rf95.available()){
if (rf95.recv(buf, &len)){
digitalWrite(LED, HIGH);
Serial.print("Recibido: "); Serial.println((char*)buf);
// Enviando respuesta
uint8_t data[] = "Recibido OK!!";
rf95.send(data, sizeof(data));
rf95.waitPacketSent();
Serial.println("Enviando respuesta!!");
digitalWrite(LED, LOW);
}
Else
{
Serial.println("ERROR en recepción");
}}}
```

Enviando datos por un enlace Lora Radio.

Para ver el funcionamiento de un enlace Lora Radio con un ejemplo que envía datos de un sensor vamos a transmitir los datos de temperatura de un sensor HDC1000.



La electrónica es la misma que en el ejemplo anterior, solo se ha agregado el sensor HDC1000 conectado al puerto I2C de la placa Arduino UNO.
El resultado obtenido se puede ver en la siguiente imagen.

```
buffer[5] = 0;
// Apagar pin remoto
else
buffer[5] = 1; // Activar pin remoto
sensor.requestTemperatures(); // Medir temperatura
temp = sensor.getTempCByIndex(0);
// Obtener el resultado
dtostrf(temp, 2, 1, buffer); // Convertir ASCII
Serial.write( (char*)buffer, sizeof(buffer) );
// Enviar trama
delay(1000);
// Esperar un segundo para nuevos datos
}
```

Tecnología ZigBee.

ZigBee es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica por enlace de radio de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal. Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y un muy bajo consumo de energía.
En principio, el ámbito donde esta tecnología tiene mayor presencia es en domótica. La razón de ello son diversas las características que lo diferencian de otras tecnologías.

- Su bajo consumo.
- Su topología de red en malla.
- Su fácil integración (se pueden fabricar nodos con muy poca electrónica).

ZigBee es muy similar al Bluetooth pero con algunas diferencias y ventajas para domótica:

- Una red ZigBee puede constar de un máximo de 65535 nodos distribuidos en subredes de 255 nodos, frente a los ocho máximos de una subred (Piconet) Bluetooth.
- Menor consumo eléctrico que el Bluetooth. En términos exactos, ZigBee tiene un consumo de 30 mA transmitiendo y de 3 μA en reposo, frente a los 40 mA transmitiendo y 0,2 mA en reposo que tiene el Bluetooth. Este menor consumo se debe a que el sistema ZigBee se queda la mayor parte del tiempo dormido, mientras que en una comunicación Bluetooth esto no se puede dar, y siempre se está transmitiendo y/o recibiendo.
- Tiene una velocidad de hasta 250 kbit/s, mientras que en Bluetooth es de hasta 3000 kbs.
- Debido a las velocidades de cada uno, uno es más apropiado que el otro para ciertas cosas. Por ejemplo, mientras que el Bluetooth se usa

para aplicaciones como los teléfonos móviles y la informática casera, la velocidad del ZigBee se hace insuficiente para estas tareas, desviándolo a usos tales como la Domótica, los productos dependientes de la batería, los sensores médicos, y en artículos de juguetería, en los cuales la transferencia de datos es menor.

En ZigBee nos encontramos con tres categorías de nodos.

- *El primero y más importante ya que ha de existir obligatoriamente en una red. El **Coordinador ZigBee** es el nodo más completo y se encarga de controlar toda la red y los caminos para su comunicación.*
- *Por debajo tenemos el **Router ZigBee** que interconecta los nodos para poder ejecutar código del usuario, es decir, ofrece un nivel de aplicación dentro de la torre de protocolos.*
- *Por último, el **Dispositivo final ZigBee** sólo recibe información y se comunica únicamente con el nodo padre. La ventaja de este dispositivo es que puede permanecer dormido y 'despertarse' en ciertos momentos para alargar la duración de batería.*

Sin duda que ZigBee es una alternativa interesante para la comunicación entre distintos dispositivos que necesiten de un consumo energético reducido. Al contrario que el Bluetooth o el WiFi, ZigBee es una opción a tener en cuenta a la hora de **domotizar una casa**. Además, el Internet de las Cosas puede recurrir a esta tecnología en lugar de a dispositivos conectados por WiFi.
Por ejemplo, creando un nodo central con conexión a internet y con suministro energético constante se podrían verificar todos los aspectos de una vivienda.

Topologías de red para ZigBee.

ZigBee permite tres topologías de red.

- Topología en estrella: el coordinador se sitúa en el centro.
- Topología en árbol: el coordinador será la raíz del árbol.
- Topología de malla: al menos uno de los nodos tendrá más de dos conexiones.

La topología más interesante es la topología de malla. Ésta permite que si, en un momento dado, un nodo del camino falla y se cae, pueda seguir la comunicación entre todos los demás nodos debido a que se rehacen todos los caminos. La gestión de los caminos es tarea del coordinador.
El paso siguiente es ver el ZigBee en funcionamiento, para esto tenemos el siguiente ejemplo que envía el estado de cuenta de un contador por ZigBee.

```
/*************************************************************
Descripción : Recibe el valor de un contador por un enlace
ZigBee usando MRF24J40MA a 2.4 Ghz
```

Enlace de radio NRF24L01.

En la actualidad se consiguen varios tipos de módulos del NRF24L01, todos operan en la banda de 2.4GHz, son muy usados por su funcionalidad, bajo consumo y bajo costo, los más populares son los que se muestran en la imagen anterior, El más básico y económico es el que se muestra en la imagen, básicamente es el chip NRF24L01 y sus componentes necesarios para su funcionamiento.

Otro modelo es más completo, aparte del NRF24L01 posee un circuito amplificador de potencia (PA), un circuito amplificador de bajo ruido (LNA) además de una antena SMA que en conjunto le permiten lograr un rango de hasta 1000m.

La alimentación del NRF24L01 (VCC) va conectado al pin 3V3 del Arduino, esto porque el modulo funciona con 3.3V. NO conectar a 5V porque podemos quemar al módulo, los pines de datos los estamos conectando directamente al Arduino a pesar que los niveles lógicos del NRF24L01 son también de 3.3V, esto debido a que el NRF24L01 es tolerante a los 5V en su bus de datos sin embargo para aplicaciones en donde el sistema pase largos períodos de funcionamiento enviado y recibiendo datos es conveniente el uso de un adaptador de niveles digitales.

En el siguiente ejemplo enviamos por el enlace de radio la medición de temperatura con un sensor DS18B20.

```
/************************************************************
** Descripción : Envía el valor de temperatura
** leído desde un sensor DS18B20 usando un
** transponder de 2.4 Ghz NRF24L01
** Target : Arduino UNO
** ToolChain : Arduino IDE 1.8.19 bajo Linux Debian
************************************************************/
#include <SoftwareSerial.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <RF24_config.h>
#include <SPI.h>
#define CE_PIN 9
// Pines asignados al hardware NRF24L01
#define CSN_PIN 10
OneWire ourWire(2);
```

```
RF24 radio(CE_PIN, CSN_PIN);
// Crea el objeto radio (NRF24L01)
char datos[10]; // Buffer para los datos recibidos
void setup(){
radio.begin();
//inicializamos el NRF24L01
/********* Posibles ajustes de potencia **********
RF24_PA_MIN = -18dBm, RF24_PA_LOW = -12dBm
RF24_PA_HIGH = -6dBm, RF24_PA_MAX = 0dBm
*************************************************/
// Potencia máxima (irrelevante en modo escucha)
radio.setPALevel(RF24_PA_MAX);
Serial.begin(9600);
radio.setChannel(100); // 125 canales posibles
// Modo escucha siempre llamar antes que radio.startListening()
radio.openReadingPipe(1, direccion); radio.startListening(); //
Modo escucha activo
}
void loop() {
if ( radio.available()) // Datos disponibles?
{
radio.read(datos,sizeof(unsigned long));
// Leer los datos
Serial.println(datos);
// Mostrar los datos en terminal
}
delay(1000);
}
```

Medición de radiación infrarroja.

El sensor MLX90614 parece un sensor convencional para medir temperatura, sin embargo este sensor tiene una característica que lo hace diferente, este sensor puede medir la temperatura a distancia, lo único que se debe hacer es apuntar hacia el objeto del que necesitas medir la temperatura.
Esta es la característica importante del pequeño sensor, medir a través de ondas infrarrojas. Aprovechando un fenómeno físico conocido, la temperatura de un cuerpo se puede medir a través de las ondas infrarrojas que se irradia en el ambiente.
Los rangos de temperatura que puede medir este sensor son: de -70°C a +380°C.
Toma las mediciones en ángulos de visión hasta por sobre 90 grados.
Con estas características, es mucho más práctico determinar la temperatura de un área específica.
El sensor ya viene calibrado directa de fábrica, las temperaturas óptimas de operación para el sensor son de -40°C a +125°C.
Rango de temperatura de medición en los objetos -70°C a +380°C con una

precisión de 0.5°C.
Tiene un campo de visión de 90°, una alimentación de 4.5V a 5.5V (para la versión de 5V).
Alimentación: 2.6V a 3.6V (para la versión de 3V), el protocolo es I2C con una dirección fija de 7 bits 0x5A.
El siguiente es un ejemplo de uso simple para este sensor.

```
#include <Wire.h>
#include <Adafruit_MLX90614.h>
Adafruit_MLX90614 mlx = Adafruit_MLX90614();
void setup() {
Serial.begin(9600);
mlx.begin();
}
void loop() {
Serial.print("Ambiente = ");
Serial.print(mlx.readAmbientTempC());
Serial.print("°C\tObjeto = ");
Serial.print(mlx.readObjectTempC()); Serial.println("°C");
delay(500);
}
```

Usando una pequeña pantalla se pueden obtener efectos interesantes, por ejemplo tener un dispositivo que muestre tanto la temperatura ambiente como la temperatura de un objeto frente al sensor
El ejemplo funcionando se lo puede ver en nuestro canal de yotube (**firadmin)**

```
/*************************************************************
Descripción: Ejemplo para el sensor de temperatura por
infrarrojos MLX90614.
El ejemplo también usa una pantalla OLED de 96x39
pixeles, controlador I2C SSD1306.
Códigos disponibles en:
https://github.com/firtec/Arduino_Ejemplos
Placa Arduino: UNO
Arduino IDE: 1.8.19
*************************************************************/
```

el duty por lo que queda claro que el motor se puede mover cada 20mS. La biblioteca *Servo.h* se encarga del manejo del PWM a través de un pin asignado para eso, el pin 9 en nuestro ejemplo.
El motor solo tiene tres conexiones, "+", "-" y el pin de control, el siguiente es el sketch para el manejo del servo.

```
#include <Servo.h>
Servo servoMotor; // Crea una instancia del servo
void setup() {
  servoMotor.attach(9);
// Definimos el pin de control
}
void loop() {
  servoMotor.write(0);
// Desplaza a la posición 0°
  delay(1000);   // Espera 1 segundo
  servoMotor.write(90);
// Desplaza a la posición 90°
  delay(1000);   // Espera 1 segundo
  servoMotor.write(180);
// Desplaza a la posición 180°
  delay(1000);           // Espera 1 segundo
}
/********* Fin de Programa FIRTEC ARGENTINA **/
```

Control de un motor paso a paso.

El siguiente ejemplo para el manejo de un motor paso a paso ha sido extraído de la biblioteca de ejemplos de Adafruit.
Utiliza un Kit de motor + Interfaz de potencia 2PH640 con un integrado ULN2003, que viene como accesorio en muchos kit de Arduino.

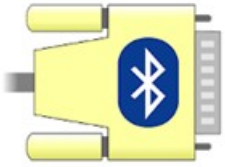
El manejo del motor es realmente sencillo sin embargo se debe remarcar que cuando se alimenta todo de la placa Arduino **el regulador de 5 voltios de la placa se calienta en exceso**, si no vigila la temperatura de este regulador se dañara.

tiempo vuelve a intentar los canales marcados como no aptos y si la interferencia ha cesado el canal se puede utilizar.
El hardware de un Bluetooth tiene dos partes.

- *Un dispositivo de radio, encargado de modular y transmitir la señal.*
- *Una CPU o controlador de Enlace y de las interfaces con el dispositivo anfitrión.*

En nuestro ejemplo usaremos el enlace Bluetooth HC-05, un clásico en la electrónica con microcontroladores.

Este dispositivo es muy similar al HC-06 sin embargo este se diferencia del HC-05 en que no espera un terminador de línea como “ \n”.
Simplemente espera un tiempo y salta a la siguiente línea, se debe tener en cuenta el detalle a la hora de hacer los programas, además el HC-06 solo funciona como esclavo mientras que el HC-05 pude ser maestro o esclavo. La cantidad de pines también es diferentes, solo cuatro para el HC-06 mientras que el HC-05 tiene seis pines de los cuales solo usaremos los pines de alimentación, TX y RX. El ejemplo propuesto lee la temperatura de un sensor DS18B20 y lo envía mediante un enlace Bluetooth a un teléfono Android.En el móvil corre la aplicación “*Serial Bluetooth*” que se descarga desde *Play Store*. Esta aplicación nos permite conectar con HC-05 y leer los datos enviados por el sensor.



Icono de Serial Bluetooth en el Play Store.

Básicamente los datos son leídos desde el sensor DS18B20 y enviados al puerto UART que a su vez se encuentra conectado a los pines TX y RX de la placa Bluetooth.

```
  lcd.setCursor(10,3);
  lcd.print("~");
}
/****** FUNCIÓN PARA LEER LAS VARIABLES ATMOSFERICAS *********/
void Sensor_DHT22(void){
//M1:
  t = dht.readTemperature(); // Lee sensor temperatura
  h = dht.readHumidity();  // Lee sensor humedad

  if (isnan(h) || isnan(t)) { // Verifica si error de lectura
              lcd.setCursor(14,1);
              lcd.print("Error");
              lcd.setCursor(14,2);
              lcd.print("Error");
              }
            else{ // No hay error, procesa los datos

   dtostrf(t, 4, 1, Temperatura); // Procesa la temperatura
   dtostrf(h, 4, 1, Humedad);     // Procesa la humedad
   lcd.setCursor(15,1);
   lcd.print(Temperatura);  // Muestra la temperatura
   lcd.print((char)223);
   lcd.setCursor(15,2);
   lcd.print(Humedad);    // Muestra la humedad
   lcd.print("%");

   }
 }
```

Midiendo PH con Arduino.

La Pantalla LCD 128x64 JLX12864G-086 es una una estupenda pantalla con pines para una interfaz SPI.
Este modelo cuenta con un regulador de voltaje integrado lo que permite conectarla directamente a 5 Voltios.
La pantalla LCD 128x64 JLX12864G-086 es una gran opción para ser utilizada en lugar de la pantalla del Nokia 5110.
Entre sus principales características destaca su diseño sencillo, compacto y estando integrado en una placa que dispone de 4 perforaciones para su fijación, con todos los componentes necesarios para su funcionamiento.
Dispone de LED (backlight) que facilita la visualización de datos, contribuyendo así a la mejora de los proyectos realizados por los diseñadores.
Para poder usar esta pantalla necesitamos la biblioteca **U8glib** que se puede

La interfaz que vincula el sensor al Arduino tiene dos pre-set, el que esta mas próximo al conector BNC será el que se usa para calibrar el sensor (no viene calibrado). La medición del sensor es lineal y para ajustarlo necesitamos dos buffers que generalmente vienen con el propio sensor, uno de ph 4.01 y otro de ph 6.86.
Considerando que el sensor es lineal con estos dos buffers podemos usar un poco de matemática para convertir el voltaje medido en cada uno de estos dos puntos de ph.
La formula general sería A = MX + B, por lo que tenemos tenemos que calcular M y B ya que X sería el voltaje y A el ph.
El resultado es A = -5.70x + 21.34, en nuestro caso usamos un medidor de ph comercial para ajustar el sensor bajo prueba.

Ejemplo propuesto funcionando.

Como se puede ver en la imagen la exactitud del sensor respecto de uno comercial es muy buena, en realidad la única diferencia que encontramos de este sensor con sus referentes comerciales es la duración en un uso intensivo y continuo.
Otra cosa a tener en cuenta es que es recomendable montar la interfaz de tal forma de tener acceso al pre-set de calibración ya que si el sensor pasa mucho

```
* Programa para medir electroconductibilidad y partes por
* millon.
* Sensor de temperatura DS18B20.
* Pantalla JLX12864G-086 (Graphic LCD 128x64)
*
* Placa Arduino: ARDUINO NANO
* Arduino IDE: 1.8.15
*
* www.firtec.com.ar
*
*************************************************************/
#include "U8glib.h"
#include <OneWire.h>
#include <DallasTemperature.h>
const int analogInPin = A0;
int sensorValue = 0;
unsigned long int conversor;
float b;
//int buf[10],temp;
char buffer[20] = "0";
char buf_temp[5] ="0";
char buf_ppm[10] ="0";
U8GLIB_NHD_C12864 u8g(19, 18, 8, 10, 9);    // Arduino NANO
//U8GLIB_NHD_C12864 u8g(13, 11, 8, 10, 9);  // Arduino UNO
/*-------------------------------------------------------------
| JLX12864G-086 (Graphic LCD 128x64)
|
|-------------------------------------------------------------
| Pin Connect :
|
|          CS : 8         VDD : 3v3
|
|         RST : 9         VSS : GND
|
|          RS : 10       LEDA : 5v
|
| SPI:    SDA : 11
|
| SPI:    SCK : 13
|
-------------------------------------------------------------*/
int R1= 4700;
int Ra=25; // Resistencia para alimentar la sonda
int ECPin= A0;
int ECGround= 3;
int ECPower = 4;  // Pin para alimentar la sonda

float PPMconversion=0.7;
float TemperatureCoef = 0.019;
```

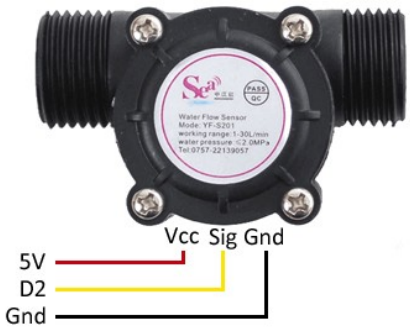
Como el nivel de conducción eléctrica varía con la temperatura es necesario medirla para compensar la lectura del valor eléctrico.

Medidor de caudal con Arduino.

Los caudalímetros como el YF-S201, FS300A y el FS400A están constituidos por una carcasa plástica estanca y un rotor con paletas en su interior. Al atravesar el fluido el interior el sensor el caudal hace girar el rotor.
La velocidad de giro se determina mediante un imán fijado al rotor, que es detectado mediante un sensor hall externo a la carcasa. Por tanto, ninguna parte eléctrica está en contacto con el fluido.
La salida del sensor es una onda cuadrada cuya frecuencia es proporcional al caudal atravesado.

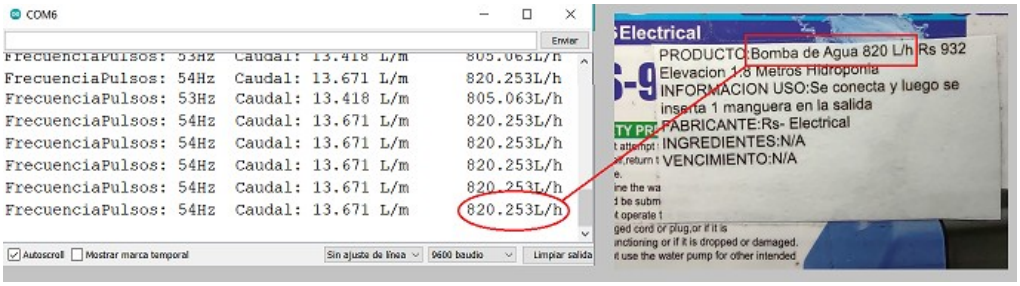
$$f\,(Hz) = K \cdot Q\,(l/min) \Rightarrow Q\,(l/min) = \frac{f(Hz)}{K}$$

El factor K de conversión entre frecuencia (Hz) y caudal (L/min) depende de los parámetros constructivos del sensor. El fabricante proporciona un valor de referencia en sus Datasheet. No obstante, la constante K depende de cada caudalímetro. Con el valor de referencia podemos tener una precisión de +-10%. Si queremos una precisión superior deberemos realizar un ensayo para calibrar el caudalímetro.
La conexión del caudalímetro es muy sencilla. Por un lado alimentamos el sensor conectando Vcc y Gnd, respectivamente, a 5V y Gnd en Arduino. Por otro lado, conectamos la salida del sensor SIG a un pin digital que permita emplear interrupciones.



Para realizar la lectura del caudalímetro debemos calcular la frecuencia de la señal de salida del sensor. Para ello emplearemos una interrupción que cuente pulsos en un determinado intervalo, y dividiendo el número de pulsos entre el intervalo en segundos, obtendremos la frecuencia.
El siguiente código convierte la medición de frecuencia a caudal, para lo cuál

```
  Serial.println ("L/h");
}
```



Resultado obtenido con el ejemplo.

NOTA:

En la imagen anterior se puede ver que la lectura de caudal es muy próxima a la informada por el fabricante de la bomba que se uso en el ejemplo.

Manejo de un teclado matricial.

Un teclado matricial es un dispositivo que agrupa varios botones y permite controlarlos mediante una matriz usando menos conductores que conectando botón por botón.

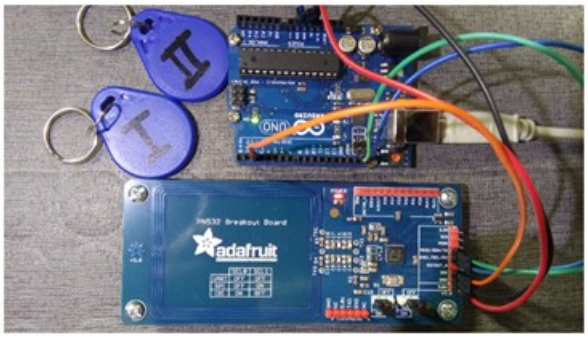
El teclado se organiza en filas y columnas que forman esta matriz, una de las desventajas de usar un teclado matricial es que pueden causar problemas cuando se pulsa más de una tecla en el mismo momento.

El uso de estos teclados es muy común, computadoras, teléfonos, alarmas, etc. Existen incluido circuitos integrados diseñados específicamente apara el manejo de estos teclados.

En la imagen siguiente e puede ver la matriz de un teclado de cuatro filas por tres columnas (12 teclas) como el usado en el ejemplo propuesto.

corriente eléctrica, suficiente para el funcionamiento del circuito integrado CMOS del *tag* y la transmisión de información al lector.
La distancia de aplicación de estos *tags* es para uso cercano, unos pocos centímetros entre el *tag* y el lector.

Estos tag son de uso común para el acceso a edificios, cocheras, expendedoras de bebidas, y mil aplicaciones mas.
Como receptor usaremos el una placa de Adafruit con el chip PN532. La ventaja de este chip es que no solo puede leer tag pasivos, también puede escribir los tag que admiten escritura.
Para usar esta placa necesitamos descargar la librería *Adafruit_PN532*.



Esta placa es muy versátil y con un costo muy razonable, puede funcionar de varias maneras, SPI, I2C, RS-232, el alcance de su receptor es bastante bueno lo que permite esconderla en una caja plástica para usarla en una aplicación real.
Tiene dos puentes que se deben configurar para indicar si la conectamos por SPI o I2C, en el ejemplo funciona mediante I2. (No verificamos su funcionamiento por SPI).
Solo son necesarias cinco conexiones a la placa Arduino, dos de alimentación (5V), SDA (A4), SCL (A5) y el pin IRQ que hemos conectado al pin 2.

```
#include <Wire.h> // Biblioteca para el I2C
#include <Adafruit_PN532.h>
// Librería de Adafruit
#define PN532_IRQ   (2)
```

Funcionamiento general de un PID.

Para el correcto funcionamiento de un controlador PID que controle un proceso o sistema se necesita, al menos:

1. *Un sensor, que determine el estado del sistema.*
2. *Un controlador, que genere la señal que gobierna al actuador.*
3. *Un actuador, que modifique al sistema de manera controlada.*

El sensor proporciona la información al controlador, la cual representa el punto actual en el que se encuentra el proceso o sistema.
El controlador lee una señal externa que representa el valor que se desea alcanzar. Esta señal recibe el nombre de punto de consigna (o punto de referencia), la cual es de la misma naturaleza y tiene el mismo rango de valores que la señal que proporciona el sensor.

Control de temperatura con PID.

El siguiente ejemplo mantendrá la temperatura dentro de una caja cerrada, como sensor de temperatura usamos el LM35 y supervisamos el funcionamiento con una pantalla LCD.
El manejo de potencia lo haremos con un relay de estado solido que será controlado por una salida PWM vinculado al PID.
Con un Osciloscopio conectado a la salida PWM podemos ver como la potencia es modulada por el canal PWM de acuerdo a lo calculado por el algoritmo PID.

```
/********************************************************
Descripción: Ejemplo para PID usando las bibliotecas de Arduino
El objetivo es mantener la temperatura dentro de una caja de
acuerdo al valor de un Set Point.
El sensor usado es un LM35 conectado en AN0 y la salida para el
control PID en el pin 3 (PWM).
Kp, Ki y Kd se ajustan de manera experimental.
Placa Arduino: UNO
Arduino IDE: 1.8.5
********************************************************/
#include <PID_v1.h>
#include <LiquidCrystal.h>
int salidaPWM = 3; // Salida de señal PWM
double temp, error, Setpoint, Output;
// Variables del PID
double Kp = 8, Ki = 1.5, Kd = 1; // Ajustes del PID
// PID myPID(&temp, &Output, &Setpoint, Kp, Ki, Kd,
// REVERSE);
PID myPID(&temp, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
```

```
void loop() {
sensor_dato = analogRead(0); // Lee el valor del sensor en A0
lcd.setCursor(0,1);
lcd.print("Aire:");
if(sensor_dato > 254)
lcd.print("Contaminado");
else if((sensor_dato < 254) && (sensor_dato > 150))
lcd.print("Bueno ");
else if(sensor_dato < 150)
lcd.print("Excelente!!");
delay(100);
}
```

Para comprobar el correcto funcionamiento se pude colocar unas gotas de perfume en un envase y cubrir el sensor con el envase, inmediatamente se verá como los valores de contaminación se disparan.

Ethernet y protocolos de RED.

El modelo OSI.

El modelo de referencia OSI (Open System Interconnection o interconexión de sistemas abiertos)
fue creado en el año 1984 por la ISO (Organización Internacional para la Estandarización). El objetivo era solucionar los problemas de compatibilidad entre los sistemas provenientes de distintos fabricantes. Este modelo define una estructura de siete capas, en donde cada una es completamente independiente del resto y soluciona un aspecto particular del sistema de comunicación.

Capa 1: FÍSICA

La capa física es la que se encarga de transmitir los bytes por el medio que sea. En esta capa se define el hardware de conexión, los niveles de tensión en los cuales se representan los unos y los ceros, los conectores empleados, el tipo de medio (por ejemplo, coaxial, par trenzado o fibra óptica), la velocidad de transmisión, etc.
En definitiva, todos los aspectos físicos requeridos para transportar los bits de información.

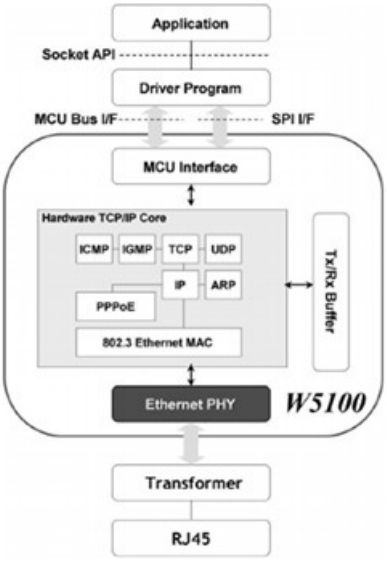
Capa 2: ENLACE DE DATOS

La función principal de esta capa es permitir la comunicación con el dispositivo, y que la comunicación sea libre de errores. Esta capa delimita la

Aquí tenemos la lectura del pin AN0 de la placa Arduino visto desde un sitio web, el usuario podría estar en cualquier parte del mundo y viendo lo que está pasando en su placa Arduino.
El diseño de la parte gráfica puede ser tan bonito y atractivo como conocimientos de diseño web tengamos.
Este enfoque cambia sin duda los alcances entre un diseño clásico con una pantalla gráfica conectada al equipo y un diseño con conectividad TCP-IP.

Ethernet Shield con WS5100.

El Arduino Ethernet Shield nos da la capacidad de conectar un Arduino a una red Ethernet.
Es la parte física que implementa la pila de protocolos TCP/IP.
Este escudo está basada en el chip Ethernet Wiznet W5100, el Wiznet W5100 provee de una pila de red IP capaz de soportar TCP y UDP. Soporta hasta cuatro conexiones de sockets simultáneas y usa la librería Ethernet para leer y escribir los flujos de datos que pasan por el puerto Ethernet.
En la imagen siguiente se pude ver la estructura interna del W5100.



El escudo provee un conector Ethernet estándar RJ45. También dispone de unos conectores que permiten conectar a su vez otras placas encima.
Arduino usa los pines digitales 10, 11, 12, y 13 (SPI) para comunicarse con el W5100. Estos pines no pueden ser usados para otra cosa mientras se usa Ethernet..

Que es Ajax?

AJAX (*Asynchronous JavaScript And XML*) no es una tecnología en si mismo, mas bien se trata de varias tecnologías independientes que se unen de formas nuevas para obtener resultados distintos de los que originalmente se esperaban. Ajax utiliza las siguientes tecnologías.

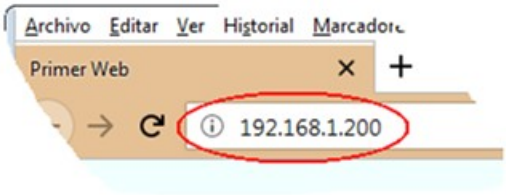
- HTML y CSS para generar páginas webs clásicas.
- DOM para la interacción dinámica de la presentación.
- XML, XSLT y JSON para el intercambio de información entre el servidor y el cliente.
- XMLHttp solicitudes para el intercambio asincrónico de información.
- JavaScript para unir todas estas tecnologías.

En un dialogo web existe un servidor y un cliente, cuando un cliente (el navegador) se conecta al servidor web (normalmente por el puerto 80), este responde enviando una página web.
La página se envía en su totalidad una vez, solo una vez, por eso es que en ocasiones debemos actualizar la pagina con F5, actualizar la página es simplemente cargarla de nuevo. Observe que aquí ya tenemos una situación problemática para trabajar con electrónica.
Imaginemos que vamos a encender un LED con un botón en la página web, cuando actuamos sobre el botón el cliente envía al servidor un mensaje HTPP con un código que identifica la acción sobre el botón, el servidor decodifica este mensaje y actúa en consecuencia encendiendo el LED.
Pero ahora imaginemos que también queremos reportar en la web el estado del LED, que nos informe si el pin del LED está a nivel alto o bajo, y es aquí donde surge la dificultad porque la página ya está cargada en el navegador, lo que se muestra ya está ahí y la única forma de cambiar algo es recargar toda la página, no estamos trabajando con una computadora en el servidor, no tenemos los mismos recursos ni velocidad y si en cada cambio o lectura de nueva información hay que recargar la página todo se torna lento y pesado.
Es aquí donde aparece Ajax que permite actualizar solo el dato sin necesidad de cargar toda la página, para los que han trabajado con pantallas gráficas es exactamente lo mismo que actualizar las variables sin necesidad de tener que "pintar" de nuevo toda la pantalla.
No importa lo densa o cargado que sea el sitio web que estamos mostrando, la lentitud de carga será la primera vez luego los datos se actualizan como si estuviéramos conectados físicamente a la pantalla.
Entonces está claro que la estrella para trabajar con electrónica y servidores web embebidos es Ajax.
Veamos un ejemplo simple para ver el funcionamiento de las funciones Ajax.

```
delay(1);
// Conexión cerrada.
navegador.stop();
Serial.println(F("Navegador desconectado!"));
 }
}
```

Resultado en el navegador web.

Obviamente este ejemplo no hace nada especial solo responde enviando un sitio web cada vez que el navegador se conecta a la IP 192.168.1.200.



En este caso la IP se ha definido fija luego veremos como colocar IP's dinámicas provistas por el servidor DHCP del propio router.
Observe que como en toda conexión de red debemos proveer de una dirección MAC que es el identificador de hardware de la tarjeta de red que estamos usando.
Como es una conexión http usamos el puerto 80 que es el puerto por defecto para este tipo de conexiones.
En la función setup() se realizan las configuraciones básicas, UART , IP y MAC del escudo Ethernet.
Una de las mayores condicionantes de la placa Arduino a la hora de escribir

navegador.println(F("<meta http-equiv=\"refresh\" content=\"1\">"))
Si apretamos o soltamos el pulsador se reflejará en la página web el estado eléctrico del pin3.
La función encargada de verificar el estado del pin y enviarlo a la página web es *Pin_Status()* y llevará como argumento el nombre que le hemos dado al cliente, en este caso "*navegador*".
La función *Pin_Status()* contiene el siguiente código.

```
void Pin_Status(EthernetClient navegador){
if (digitalRead(3)) {
navegador.println(F("Estado del pin:<font color='red'><b>
ALTO</b></font>"));
}
else {
navegador.println(F("Estado del pin:<font color='green'><b>
BAJO</b></font>"));
}
}
```

Se lee el pin 3 y según sea su estado se envía una cadena de caracteres con indicaciones de color para ser mostrados en la página web.
El mensaje se verá exactamente en la línea donde se llamó a la función *Pin_Status(),* la línea donde la función es llamada se ha pintado de azul en el código web.
El siguiente es el aspecto de la página web en el navegador.

El código completo es el siguiente.

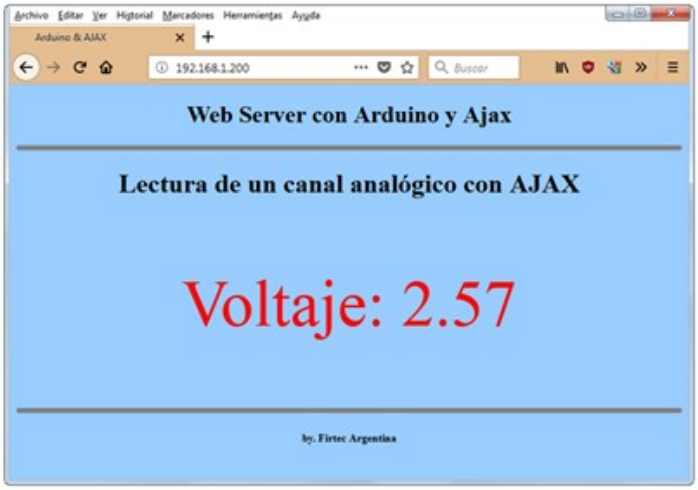
```
/*************************************************************
Descripción: Lectura del nivel eléctrico del pin 3 mediante una
web embebida.
Placa Arduino: UNO
```

```
cl.print(F("<p>Voltaje: "));
cl.print(voltaje);
cl.println(F("</p>"));
}
```

El bloque de código que controla el funcionamiento dinámico de la pagina web es el siguiente.

```
if (cliente.available()) { // Un navegador conectado!!!
char c = cliente.read();
// Lee 1 byte, un carácter desde el navegador
HTTP_req += c; // Guardo caracteres de uno en uno
if (c == '\n' && currentLineIsBlank) {
// Ejecuta cuando el mensaje se ha recibido completamente
if (HTTP_req.indexOf("ajax_AD") > -1) {
// Solicitud viene de la función AJAX?
Leer_Conversor(cliente);
 // Leo el valor de voltaje en A0 con la función creada
}
// Si la solicitud no es un comando cargar la pagina
else {
            Aquí se carga la página web
}
```

Usando correctamente la función *indexOf( )* recorremos el mensaje recibido hasta encontrar "*ajax_AD*" que desencadena el llamado a la función *Leer_Conversor( )* que es la encargada de leer el voltaje y finalmente enviarlo como respuesta a la página web.
La página obtenida será la siguiente.



```
cl.println(Temperatura);
cl.println(F(" &degC"));
cl.println(F("</p>"));
cl.print(F("<p>Presión: "));
cl.print(Presion);
cl.println(F(" Mb"));
cl.println(F("</p>"));
cl.println(F("</p>"));
cl.println(F("</font>"));
cl.print(F("</center>"));
}
```

La página obtenida es la siguiente.

El funcionamiento del programa sigue la misma línea de los ejemplos anteriores, una función Ajax embebida en el código HTML que mediante *GET* envía el mensaje *bmp280* como consulta al servidor, se decodifica el mensaje y se llama a la función que procesa los datos del sensor. Todo esto en la siguientes líneas:

```
if (HTTP_req.indexOf("bmp280") > -1) {
Leer_Sensor(cliente);
}
```

Sensor BME280 con WEB & Ajax.

El sensor *BME280* integra en un solo dispositivo sensores de presión atmosférica, temperatura y humedad relativa, con gran precisión, bajo consumo energético y un formato ultra compacto. Basado en tecnología

original de *BOSCH* piezo-resistiva tiene una alta precisión y linealidad, así como estabilidad a largo plazo.
Se conecta directamente a la placa Arduino a través de SPI o I2C siendo esta la conexión de prueba para ejemplo propuesto.
Su funcionamiento es similar al sensor *BMP280* visto en el ejemplo anterior.
Este tipo de sensores pueden ser utilizados para calcular la altitud con gran precisión (barómetro), por lo que es un sensor muy utilizado en sistemas para Drones entregando medidas de altitud con una precisión de hasta 1m. Otras aplicaciones como monitoreo de clima, Internet de las Cosas, Domótica, Aire acondicionado, etc.
Como se puede ver en la siguiente imagen, en los datos reportados a la web ahora también se muestra el porcentual de humedad.

Sitio web para el sensor BME280.

La capacidad de este sensor para medir la humedad es la diferencia principal con el *BMP280*, también su costo siendo el *BME280* un poco mas caro que su hermano el *BMP280*.

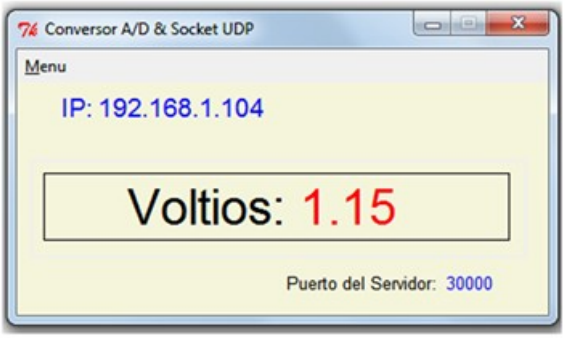
```
#include <SPI.h>
#include <Ethernet.h>
#include <Wire.h>
#include "SparkFunBME280.h"
BME280 mySensorA; //Uses default I2C address 0x77
BME280 mySensorB;
byte mac[] = {
0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
IPAddress ip(192, 168, 1, 200);
EthernetServer server(80);
```

Que es Python?

Python es un lenguaje de programación concebido a finales de los años 80 y principios de los 90 que ha sido rápidamente aceptado por desarrolladores gracias a su potencia, sencillez y legibilidad del código. Como lenguaje de programación resulta muy fácil de aprender, con estructuras de datos eficientes y de alto nivel con un enfoque simple pero efectivo. Permite un desarrollo rápido para aplicaciones en diversas áreas y sobre la mayoría de las plataformas.
Nació a fines de los 80 de la mano de ***Guido van Rossum***, un programador de origen holandés.
El nombre fue inspirado en el grupo de humoristas Monty Python siendo Van Rossum quien siguió a la cabeza de los desarrollos de este lenguaje hasta el año 2000, para esa época Python ya era bastante popular.
Desde el año 2005, Guido van Rossum trabaja en Google creando aplicaciones con Python.
El intérprete de Python y sus bibliotecas son de acceso y uso libre para distintas plataformas, desde el sitio web de Python, *https://www.python.org/* se pueden descargar y distribuirse libremente. El mismo sitio contiene también distribuciones y enlaces de muchos módulos libres de Python de terceros, programas y herramientas, y documentación adicional.
El intérprete de Python puede extenderse fácilmente con nuevas funcionalidades, tipos de datos y bibliotecas para el desarrollo de interfaces gráficas e interactuar con dispositivos electrónicos.
Lo interesante de trabajar con Python es que fácilmente podemos construir interfaces para visualizar en computadoras datos generados en Arduino.
Trabajar con el puerto UART puede ser interesante pero trabajar con conexiones en un Socket UDP lo será mucho mas, con este tipo de conexiones la placa Arduino y la computadora pueden estar separados en cualquier lugar del planeta y vinculados por Internet. Python puede crear este tipo de conexiones con gran facilidad, nació en el mundo de Internet y se mueve en Internet como pez en el agua y por eso es tan interesante en la conectividad electrónica.

Trabajando con Python.

Como interfaz de programación para correr los ejemplos propuestos podemos usar el IDLE de Python3 o cualquier programa para la edición de código.
Cuando se instala el interprete Python también se instala Tkinter que es la biblioteca gráfica que nos permite crear ventanas, botones, etc.
Si bien Tkinter se instala por defecto y está lista para funcionar, también podemos instalar *Glade* tanto para Linux como Windows. Glade es una



El programa Python es muy simple, solo muestra los caracteres que llegan por el socket, estos caracteres debieron ser formateados antes de ser enviados, recuerde por la red solo pueden viajar caracteres por lo tanto toda información que no se ajuste a ese formato debe ser convertida, en este caso en la placa Arduino.
El código Python es el siguiente.

```
# -*- coding: utf-8 -*-
#!/usr/bin/env python
import socket
import sys
import select
import errno
import time
from tkinter import *
from tkinter.messagebox import showinfo
from tkinter import Frame
from tkinter import Text
from tkinter import Label
#from Tkinter import *
#from tkMessageBox import showinfo
class MyGui(Frame):
    def __init__(self, parent=None):
        Frame.__init__(self, parent)
def get_ip():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        s.connect(('10.255.255.255', 0))
        IP = s.getsockname()[0]
    except:
        IP = '127.0.0.1'
    finally:
        s.close()
    return IP
UDP_PORT = 30000  # Puerto del socket
Dir_IP = get_ip()
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # UDP
```

Vemos el siguiente código que hace el trabajo propuesto, notará que salvo muy pequeñas diferencias el código es el mismo que el usado con la placa Ethernet.
Esto es interesante porque cualquier programa que funcionara con el escudo de Ethernet puede fácilmente ser portado a Wi-Fi usando ESP32.

```
/*************************************************
** Descripción : Crea un servidor WEB y una página
** vacía que es mostrada en un navegador.
** Target : ESP32
** ToolChain : Arduino
** www.firtec.com.ar/
*************************************************/
#include <WiFi.h>
WiFiServer server(80);
const char* ssid = "Firtec";
const char* password = "12345";
String buffer;
void setup() {
Serial.begin(115200); // Configura la UART
while(!Serial) {
;
}
Serial.println(); // Muestra carteles
Serial.println();
Serial.print("Conectado con ");
Serial.println(ssid);
WiFi.begin(ssid, password);
// Intenta conectar con la red WiFi
while(WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println("");
Serial.println("WiFi conectado");
// Muestra carteles
```