Sumario

Midiendo el nivel de una Cisterna.

En muchos lugares es costumbre tener cisternas o depósitos de agua potable sobre todo en los lugares cuando el verano arremete y la presión en la red de agua baja y no es suficiente para cargar estos depósitos.
Entonces resulta interesante tener una forma de poder verificar el nivel de agua que tiene la cisterna para actuar en consecuencia ya sea activando una bomba de llenado o simplemente tener la precaución de agudizar la economía de este vital elemento.
El ejemplo propuesto hace justamente esto, mide la cantidad de agua presente en la cisterna enviando al móvil la cantidad de agua disponible en cinco posibles niveles.
Completo, tres cuartos de cisterna con agua, media cisterna con agua, un cuarto de cisterna con agua o finalmente cisterna vacía.
Para lograr esto cinco electrodos cierran su contacto a través del agua con un electrodo común que envía cinco voltios, este voltaje es interpretado como una señal que se transfiere a una serie de pines de una placa Arduino y un código programado en su memoria decodifica el nivel de agua y lo envía por un enlace Bluetooth.
Esta señal es recibida por el móvil y mostrada en una aplicación para Android que ha sido desarrollada para esta electrónica llamada *Cisterna.apk*.

Cisterna.apk programa para el móvil.

Uno de los principales problemas que tienen los sistemas que usan electrodos sumergidos en agua es la degradación que sufren por efectos de la electrólisis al hacer circular una corriente continua por estos electrodos.

Construcción de los sensores.

Con un cable canal de uso común en instalaciones eléctricas se ha ensamblado el sistema de sensores.

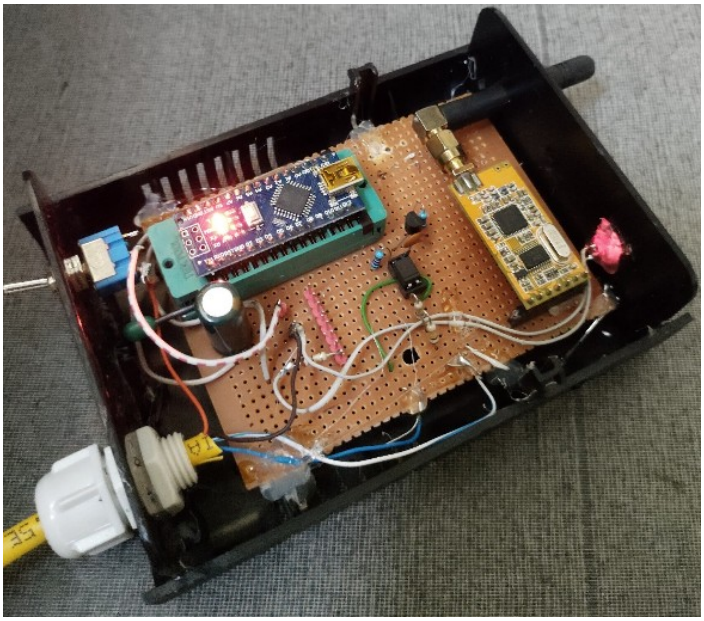
Sistema de sensores ensamblado con un cable canal.

A lo largo del tubo rectangular en su interior se ha dispuesto el cable común que lleva los pulsos generados por el pin 19 de Arduino.
Luego se han realizado una serie de perforaciones donde se han colocado los cables que ofician de sensores colocados cerca del cable común pero sin posibilidad de estos se toquen (esto es importante!!) el único contacto eléctrico debe ocurrir a través del agua cuando esa porción del tubo esta sumergido.
Este es un ejemplo de muy simple construcción y funcionamiento muy eficiente se podrían usar otros dispositivos para medir el nivel como ultrasonidos, medidores de presión, etc. Pero estos son mas complicados de construir, mas costosos de mantener, con una programación mas sofisticada que de repente para un uso “doméstico” con este sistema de contactos es suficiente y si al cabo de unos años los electrodos envejecen por estar sumergidos simplemente se cambian los conductores y todo queda funcionando.
No obstante mas adelante se vera como construir un medidor de nivel de llenado mediante un sensor de presión diferencial que también ha sido construido y se encuentra en funcionamiento.

Enlace de radio APC220

Timbre con enlace inalámbrico.

5V se conecta a +5 voltios.
GND se conecta a GND de 5 voltios.



Unidad transmisora en su caja.

En la imagen anterior se puede ver el prototipo montado en una placa experimental. En este ejemplo se pretende extender el alcance de un timbre ya instalado con una campanilla conectada a 220 voltios entonces mediante un opto-acoplador se detecta la presencia de este voltaje en la campanilla para saber si el timbre de calle a sido activado.
Con el enlace de radio APC220 el alcance efectivo es de alrededor de cien metros en ambientes edificados y mas de mil metros en campo abierto, sin embargo se podría usar cualquier sistema de radio con alcance menor según la necesidad.
Estos enlaces normalmente vienen de pares para usarlos uno como transmisor y el otro como receptor, también se puede conseguir un puente USB/232 para el APC220 necesario si se quiere re-programar algunos ajustes que trae por defecto.
Algunas características del enlace APC220.
Algunas de las características que posee el módulo son los siguientes:
Frecuencia de trabajo programable: 418 MHz a 455 Mhz .
Alimentación: 3.3V a 5.5V
Corriente consumida: 25 a 35 mA.
Rango de transmisión: Hasta 1000m (a 2400 bps).
Interfaz de comunicación: UART/TTL

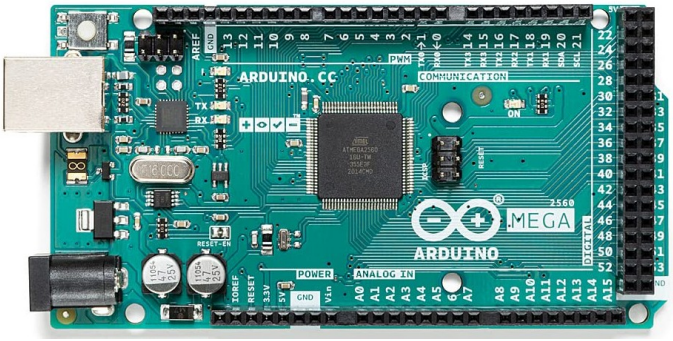
```
wdt_reset();
 if(bandera == 1){
              wdt_reset();
              digitalWrite(led,HIGH );
              startPlayback(sample, sizeof(sample));
              wdt_reset();
              delay(3000);
              wdt_reset();
              startPlayback(sample, sizeof(sample));
              wdt_reset();
              delay(3500);
              wdt_reset();
              stopPlayback();
              digitalWrite(led,LOW );
              bandera = 0;
      } }
ISR(USART_RX_vect){
  dato = UDR0;
  if(dato == 0x55)
      bandera = 1;
   dato = 0;
 }
```

La tabla de números codifica el sonido de campanas que se escucha en el pequeño parlante.

Prototipo ensamblado en un pequeño gabinete de multi-media.

El archivo *PCM.C* y *PCM.h* se pueden descargar de distintos lugares en internet sin embargo el contenido de *PCM.C* es el siguiente:

```
/* speaker_pcm
 *
 * Plays 8-bit PCM audio on pin 11 using pulse-width modulation
(PWM).
```



Arduino Mega usado en el ejemplo.

El circuito completo para su funcionamiento es el siguiente.

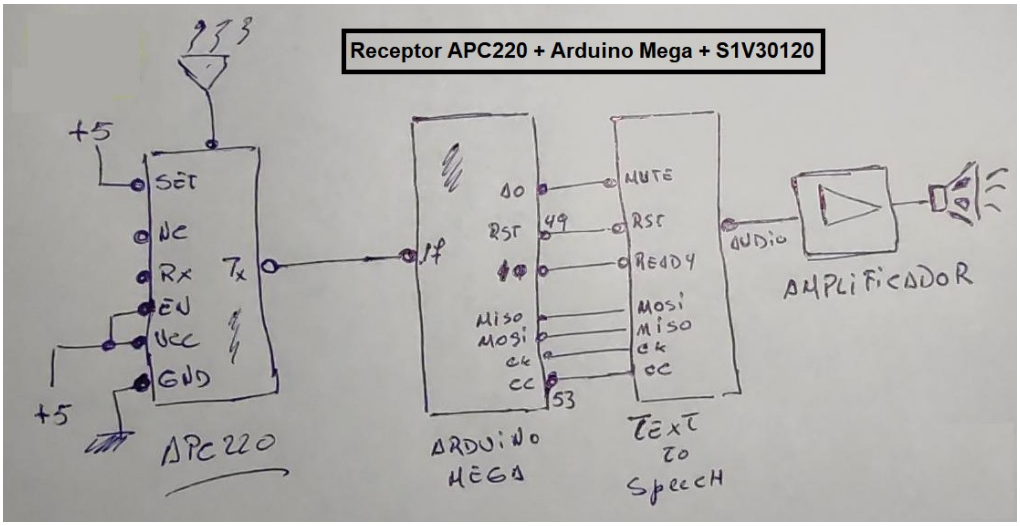


Diagrama electrónico con Arduino Mega.

Para controlar el S1V30120 necesitamos el puerto SPI de Arduino Mega, este puerto de comunicaciones tiene cuatro pines: MOSI, MISO, CK y CC.
El MOSI se conecta al MISO de la placa destino y el MISO al MOSI CK al CK y el CC o selección de chip lo hemos fijado en el pin 53 del Mega.
El ejemplo reproduce tres posibles mensajes según la necesidad pero el emisor podría enviar varios códigos o comando que corresponden a distintas puertas o estados y el sistema reproduciría un mensaje para cada comando informando de que puerta se ha activado.
El código completo para el Arduino Mega es el siguiente:

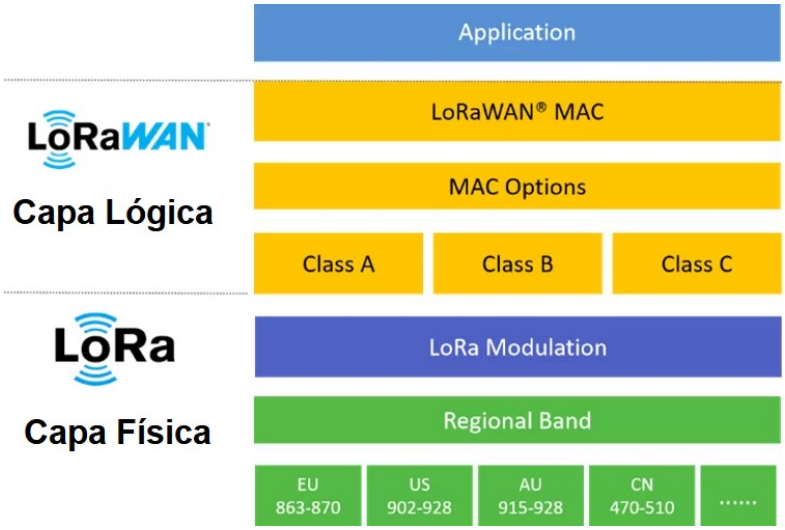
```
#include <avr/pgmspace.h>
#include <avr/wdt.h>
```

Nota 2: El archivo *text_to_speech_img.h* es demasiado extenso para transcribirlo puede solicitarlo en daniel@firtec.com.ar y con gusto se lo estaremos enviando.

Enlace de Radio LoRa.

Que es LoRa.

Es una tecnología relativamente nueva desarrollada en 2012 por la empresa Cycleo que luego paso a ser Semtech y es quien se encarga de impulsarla.
Su funcionamiento se basa en un tipo de modulación llamada de amplio espectro, ideal para no ser perturbada por el ruido y para que una señal realice caminos múltiples hasta encontrar su destino.
Presenta un ancho de banda reducido, pero adaptado a las necesidades de los dispositivos que ayuda a conectar.
No se debe confundir LoRa con LoRaWAN. LoRa es la modulación que emplean los dispositivos para lograr una cobertura con baja potencia, equivale a la capa física de la red que dicta las frecuencias de trabajo con las que se obtienen largas distancias y bajo consumo.
Por otro lado tenemos LoRaWAN que es el protocolo que define como se procesan los paquetes de datos que se envían y reciben desde los distintos elementos de la red.



Estructura de LoRa.

Actualmente LoRa Alliance es la encargada de mantener y desarrollar las especificaciones de LoRaWAN y a su vez, se encarga de certificar los

Cobertura LoRaWAN.

LoRa se creó con la finalidad de ofrecer cobertura a nivel mundial y para ello existe varias formas de conectarse a una red LoRa, por ejemplo comunidades abiertas, redes privadas o redes ofrecidas por operadoras
Las comunidades abiertas son generalmente creadas por comunidades de desarrolladores y las redes privadas, creadas para dar servicios a redes propias en aplicaciones como agricultura, logística, etc.

Sirena para alarma con enlace LoRa.

Visto las características generales del sistema de radio LoRa se propone controlar el funcionamiento de una sirena vinculada a un sistema de alarma. Si bien el sistema de alarma instalado ya es inalámbrico el desafío es sumar una sirena ubicada fuera del área de alcance del sistema de radio de la propia alarma pero que se active cuando esta se dispara como si fuera parte del propio sistema.

Sirena ya modificada usada en el ejemplo.

Como todo el sistema de enlace con la sirena se construirá a partir de cero se eligió usar una sirena simple del tipo que normalmente se instala cableada esto porque en realidad lo único necesario es el gabinete donde se instalará el nuevo sistema que dispare la sirena.

```
minutos
  wdt_reset();
  contador1++;
  if(contador1 == 850){  // +- 1 MiliSegundo
    contador1 = 0;
    contador2++;
    if(contador2 == 3000){
      contador2 =0;
      bandera = 0;
// Borra bandera para que la próxima IRQ sea valida
// y se re-envie el comando de activación porque la
// la alarma sigue activa.
     }
   }
  }
 }
}

/************************************************
  Rutina para detectar el disparo de la alarma
************************************************/
void ISR_Alarma_SI(){
  bandera_1 = 1;  // ALARMA ACTIVADA!!!!
  }
```

En el diagrama electrónico del receptor se puede ver que la señal de disparo se aplica mediante un opto-acoplador PC817 a un transistor 2A92 que a su vez activa un pequeño relevador y controla el disparo de la sirena.

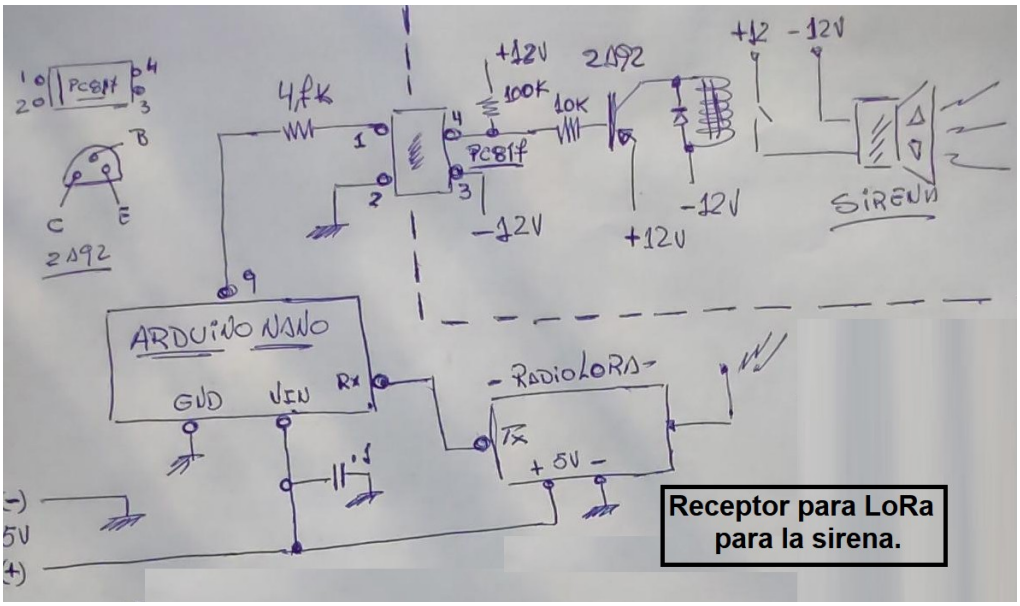


Diagrama electrónico del receptor LoRa.

El uso del opto-acoplador es para separar el voltaje de 5 voltios de los 12 voltios de la propia sirena.
En la instalación donde el sistema se encuentra funcionando todos los voltajes son provistos por una UPS que a su vez alimenta todo el sistema de alarma y el enlace Wi-Fi para el envío remoto de alertas.

UPS encargada de alimentar todo el sistema.

Para la UPS se ha usado un gabinete genérico y se ha montado en su interior un sistema de alimentación ininterrumpida con autonomía de alrededor de cuatro horas para una eventual falla eléctrica.

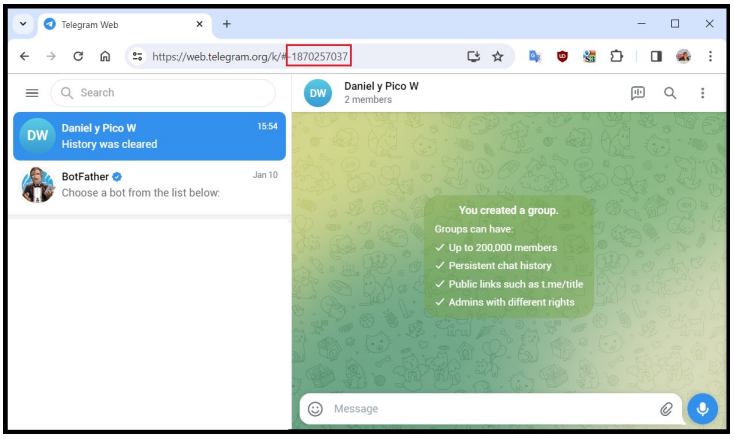
Algunas consideraciones a tener en cuenta.
1. Si bien el sistema es muy confiable desde el punto de vista del protocolo ya que solo responde si el comando adecuado es recibido lo que lo hace inmune a toda interferencia de ruido eléctrico que podría provocar un disparo accidental de la sirena, se debe prestar mucha atención al disparo accidental de la interrupción en el emisor puesto que esto genera el envío del comando de disparo. Para esto se ha usado un PC817 que sirve también como aislante y filtro para la señal de disparo. No estaría de mas colocar condensadores a tierra y sumar filtros sobre todo para las interferencias generadas por descargas atmosféricas que suelen ser de alta intensidad.
2. En la alimentación del voltaje de la sirena no está de mas colocar en algún lugar poco visible un interruptor para desconectar la sirena si algo sucediera con el Arduino, el opto, etc que llevara la sirena a una activación fuera del control de apagado por comando.

Creando un Bot para Telegram.

Está claro que para construir esto y hacerlo funcionar tenemos que tener instalado Telegram en el dispositivo móvil.
Lo primero que hacemos es crear un grupo dentro de Telegram y luego mediante el BotFather creamos un bot que agregamos al grupo creado.

BotFather de Telegram para crear bots.

El proceso de crear un bot es muy sencillo y la propia aplicación lo guía de como hacerlo con el comando /newbot.
Pero lo importante aquí es el Token, algo parecido a esto 123456:ABC-DEF1234ghIkl-zyx57W2v1u123ew11 este será el ID del bot y lo necesitará para configurar el acceso y poder enviar mensajes al bot.
Lo siguiente es obtener el ID del usuario, es importante que este bot este agregado al grupo, **luego desde un computador** enviamos un mensaje al bot y tomamos nota del número ID que aparece en la barra del navegador.
Observe la siguiente imagen donde se puede en la barra del navegador el nombre del bot y el ID correspondiente.



Obteniendo el ID de usuario.

Con esto ya tenemos los datos necesarios para conectar Telegram con el bot, ahora solo necesitamos un programa en Pico W que haga el trabajo.
La aplicación propuesta puede funcionar de dos formas.

1. **Modo Usuario:** Controla el estado de las puertas asignadas.
2. **Modo Programación:** El sistema crea una red Wi-Fi propia abierta con el SSID firadmin. En esa red se accede http://firtec.net que básicamente es una página de configuración donde se cargan las credenciales de acceso tanto a la red Wi-Fi adonde se conectará Pico W como las credenciales de acceso a Telegram.

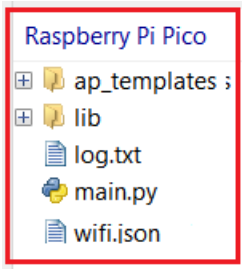
En la siguiente imagen se puede ver el aspecto de la página web de configuración para el sistema.

http://firtec.net configura el acceso al sistema de control.

Una vez que los datos son salvados la red *firadmin* desaparece y el sistema pasa al modo usuario intentando validar las credenciales de acceso a la red Wi-Fi y cuando sea necesario las credenciales de Telegram.
Para entrar al modo programación se ha colocado un pequeño pulsador en el GPIO_16 que si se oprime el sistema se desconecta de la red Wi-Fi, borra todas las credenciales y pasa a modo programación.

```
requests.post(sendURL + "?chat_id=" +
wifi_credentials['telegramDmUid'] + "&text=" +
message)
```

Puede ver que en esta línea se encuentran todos los elementos necesarios para ensamblar el mensaje, *URL* + *ID* + *Token* + *Texto*.



Archivos en la memoria de Pico W.

En la imagen anterior se pueden ver los archivos que deberían estar en memoria en la memoria del controlador Pico W.
En la siguiente imagen se pueden ver los mensajes enviados por dos dispositivos conectados a distintas puertas.

El archivo *index.html* tiene el siguiente contenido.

hidroponia, el sistema NFT o el Mixto estos dos sistema a su vez tienen derivados pero sin importar el sistema elegido todos necesitan de una bomba que impulse el agua con nutrientes a través de las raíces de las plantas y a su vez mantenga el liquido en movimiento para evitar su estancamiento.
En el siguiente ejemplo tenemos un Arduino Nano controlando el tiempo de apagado y funcionamiento de una bomba que impulsa los nutrientes por los tubos del sistema hidropónico mixto construido con tubos de PVC de 110 mm usados en descargas pluviales .

Sistema Hidropónico Mixto.

En la imagen anterior se puede ver el deposito con capacidad de 50 litros para contener los nutrientes, el sistema también tendrá un sensor de temperatura para poder ver en todo momento la temperatura de la solución.
Como se dijo el control lo realiza un pequeño Arduino Nanoquien será el encargado del control de la bomba de nutrientes, medir la temperatura de la solución nutritiva y la temperatura y humedad del exterior para tener una idea de las condiciones en donde está instalado el sistema.

Caja de control para el sistema hidropónico.

El funcionamiento del sistema es bastante simple, los datos de funcionamiento programados se almacenan en la memoria EEPROM del microcontrolador por si el sistema se apaga o queda sin energía este pueda recuperar el último estado de funcionamiento. Si no encuentra datos útiles inicia un funcionamiento con valores por defecto de 10 minutos de funcionamiento y 10 de parada para la bomba.
El sistema pasa a modo programación si se mantiene el botón conectado al pin 2 por mas de 5 segundos.

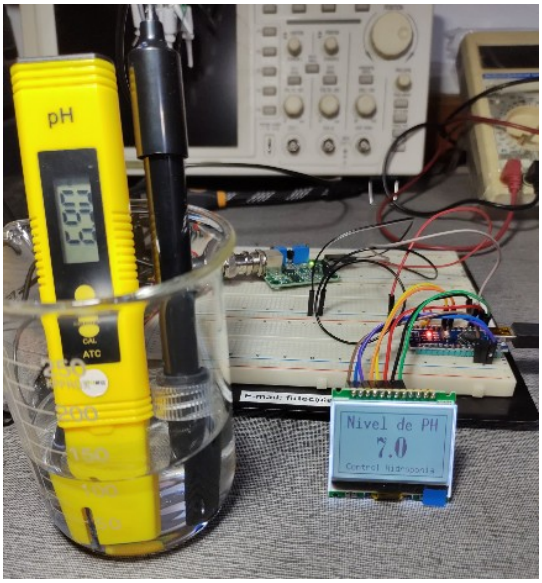
Sistema Hidropónico en funcionamiento.

Medición de pH.

El pH del suelo o medio de cultivo es una medida de acidez o alcalinidad en los suelos, esto está en relación con los iones hidrógeno.
El índice varía de 1 a 14, siendo 7 neutro, un pH por debajo de 7 es ácido y por encima de 7 alcalino.
El pH del suelo es considerado como una de las principales variables en los suelos o medios de cultivo ya que de el dependen muchos procesos químicos que afectan a las plantas en general.
Afecta específicamente la disponibilidad de los nutrientes de las plantas, la absorción de los distintos minerales se ven afectados por el nivel de acidez.
El rango de pH óptimo para la mayoría de las plantas oscila entre 5,5 y 7,0,1 sin embargo muchas plantas requieren de suelos mas ácidos y otras se desarrollan mejor en suelos alcalinos.
En la imagen siguiente se puede ver como el nivel de pH tiene incidencia directa con la absorción de los minerales que forman los llamados macro y micro nutrientes necesarios para un correcto desarrollo vegetal.
Minerales como el Nitrógeno, Fósforo o Magnesio, etc tienen una absorción óptima con un pH 7 pero por ejemplo el Hierro indispensable en la producción de clorofila y por ende en los procesos de fotosíntesis tiene su mayor absorción con pH inferior a 7.

Voltaje de Alimentación: 5VDc +/- 0.2V.
Corriente: 5 a 10mA.
Rango de PH: 0 a 14.
Rango de temperatura: 0 a 80°C.
Tiempo de Respuesta: 5seg.
Tiempo de estabilización: 60Sg.
Temperatura de trabajo: 10 a 50ºC Ideal 20ºC.
Humedad de trabajo: 95 RH sin condensación.
Dimensiones de la tarjeta: 42.5 X 32.6 X 20 mm.
Mediciones: Temperatura y PH.
Disposición de los pines.
To: Salida Analógica de Temperatura.
Do: Salida Digital de PH limite.
Po: Salida Analógica de PH.
G: Tierra.
Vcc: 5V.

El sensor de PH-4502C es un dispositivo de alta calidad y precisión que es ideal para aplicaciones de monitoreo y control del pH en diversos campos como el control de calidad del agua, la industria alimentaria, la agricultura y la investigación.



Resultados obtenidos con el sistema propuesto.

En la imagen anterior se puede ver el resultado de la medición con el sistema propuesto respecto a la medición obtenida con un sistema comercial, como se aprecia el resultado es mas que aceptable.

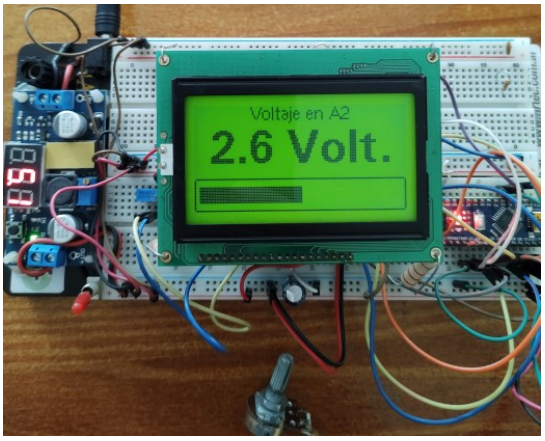
Este modo dos requiere un 60% menos de RAM que el modo uno, si por el contrario tenemos un controlador rápido con mucha RAM la elección acertada sería el modo uno.
El modo tres no tiene funciones gráficas, es el modo mas rápido y no requiere casi memoria pero solo admite texto.
Los distintos modos se seleccionan en el momento de crear el constructor para la biblioteca por ejemplo la siguiente línea es el constructor para el manejo de la pantalla KS0108.

U8G2_KS0108_128X64_**1** u8g2(U8G2_R0, ……..);

Observe el número “1” que aparece en el constructor, esto es para indicar que se usará el modo dos (Modo de Página) si en lugar de uno (también puede ser un dos) tendríamos un carácter “**F**” estaríamos seleccionando el modo uno (Modo Buffer).
Para el modo tres cada pantalla tiene su particular forma de seleccionarla, en el caso de la pantalla KS0108 el constructor tendría el siguiente formato.

U8G2_KS0108_128X64(U8G2_R0, ……..);

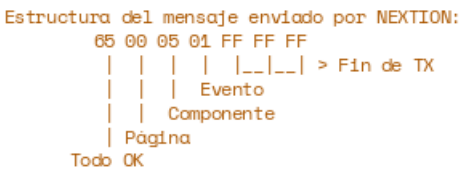
La biblioteca hace una copia de la memoria de la pantalla en la propia RAM del controlador y es en esa memoria donde se realiza el trabajo para luego copiar esta memoria a la memoria de la pantalla.
Para ahorrar memoria RAM en el controlador la pantalla se divide en dos mitades que se procesan en forma independiente es por esto que en el modo página existen dos llamados a las páginas.



Ejemplo con la pantalla gráfica KS0108

Pantallas Nextion.

Nextion es probablemente una de las mejores opciones para crear proyectos con electrónica. Son compatibles con Arduino, Raspberry PI, Atmel, Microchip, ARM, etc, además podemos utilizarlos de forma independiente contando la propia placa con una serie de pines de uso general incluso un RTC para sistemas de adquisición de datos.
Para crear una interfaz en una pantalla Nextion usaremos el editor Nextion, un software provisto por el propio fabricante que hace la tarea del diseño realmente simple.
Una vez que tenemos todo los elementos de la interfaz desplegados en la pantalla se envía la programación a la propia pantalla usando una simple conexión serial.
Este editor de Nextion es la mejor solución, en realidad casi la única para crear proyectos con las pantallas Nextion.
Lamentablemente solo existe en versión para Windows que se descarga a través del enlace oficial de Nextion.
La pantalla dispone solamente de 4 pines. Dos de ellos son de alimentación (cable rojo y negro) y los otros dos son de recepción (RX) y transmisión (TX) de datos a través de su puerto serie.
La pantalla se programa por medio de una conexión serial a 115200 baudios desde el propio editor Nextion, y también se conecta con nuestra placa (electrónica del usuario) por la misma conexión serial pero a 9600 baudios.
Esta claro que no se puede tener la pantalla conectada a la electrónica del usuario y también conectada al editor Nextion.
Los mensajes que intercambia la pantalla con el microcontrolador tienen el siguiente formato.



Formato del mensaje Nextion.

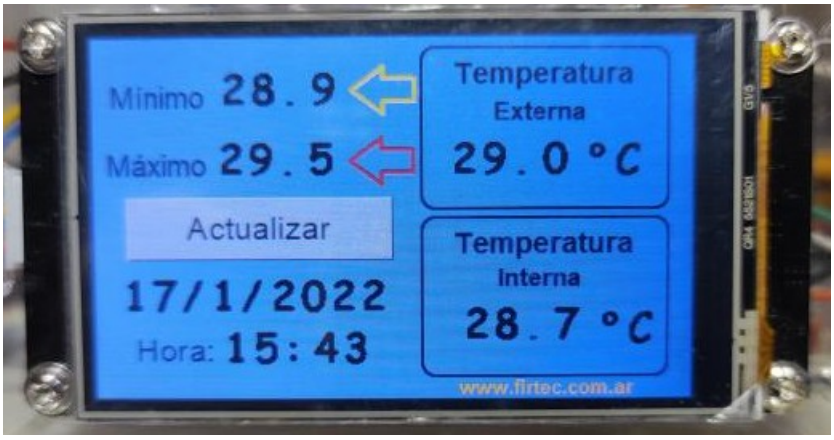
Son siete los Bytes enviados y siempre terminado con *FF FF FF.*
Por ejemplo para capturar la acción sobre un botón tenemos que decodificar la página donde se encuentra el botón, el componente (el botón) y la acción.
Cuando queremos mostrar un dato en la pantalla solo "*apuntamos*"al componente de la pantalla donde queremos enviar los datos y enviamos la información que se mostrará donde se indicó, por ejemplo imaginemos que tenemos un termómetro dibujado en la pantalla y queremos enviar los datos

```
def main():
    while True:
        global cont
        rx = sm.get() >> 24
        if cont == 0:
            a = rx
        elif cont == 1:
            enviar("rtc1="+ str(rx))
        elif cont == 2:
            enviar("rtc2="+ str(rx))
        elif cont == 3:
            enviar("rtc3="+ str(rx))
        elif cont == 4:
            enviar("rtc4="+ str(rx))
        elif cont == 5:
            enviar("rtc5="+ str(rx))
        cont += 1
        if cont == 6:
            cont = 0

if __name__ == '__main__':
    main()
```

Para el receptor serial que recibe los datos de la aplicación de ajuste se ha usado código ensamblador PIO del controlador RP2040 controlado por una interrupción.

Por ejemplo podemos construir un registrador de datos para registrar y almacenar la temperatura en una memoria SD.



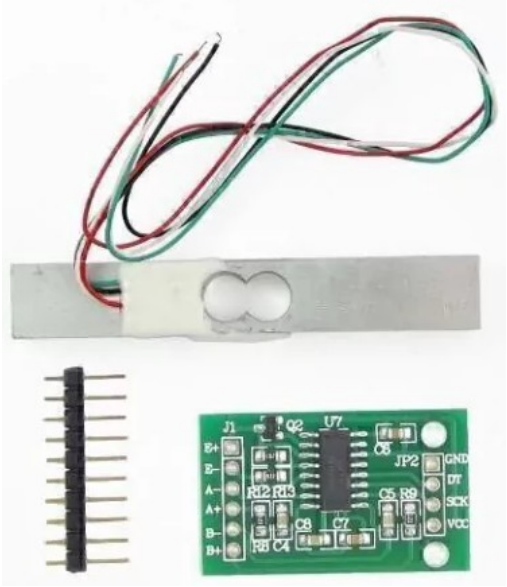
Ejemplo para un registrador de datos.

Como se puede ver el uso de pantallas gráficas no solo se limita a detectar botones y desplegar datos, también podemos escribir programas completos dentro de la propia pantalla para relacionarla con la electrónica a la que está conectada y si le sumamos que solo necesitamos un puerto serial para interactuar con estas pantallas su uso es muy simple siendo solo su costo lo

que condiciona el uso de esta tecnología.

Celdas de Carga.

Las celdas de carga son transductores que soportan cargas de compresión, tensión o flexión y las convierten en una magnitud eléctrica proporcional a la carga o el peso aplicado.
La aplicación principal de las celdas de carga es la medición de peso o fuerza aplicada a la celda por ejemplo medir la deformación de un determinado objeto cuando es sometido a carga.
Las celdas de carga poseen normalmente 4 galgas extensiométricas conectadas en una configuración de puente *Wheatstone*. Esta configuración permite leer de forma precisa las variaciones de resistencia en las galgas.
Para poder usarlas necesitamos de una interfaz que interprete la información enviada por el puente de *Wheatstone* y la transfiera al microcontrolador que se este usando.
El módulo HX711 cumple con ese trabajo y comunica la celda de carga y un microcontrolador como Arduino, permitiendo leer el peso en la celda de manera sencilla.



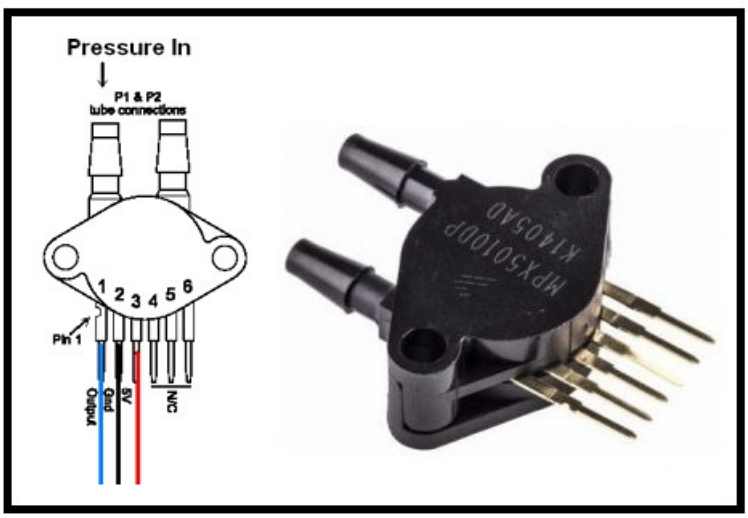
Celda de carga con su interfaz HX711.

El chip HX711 posee internamente la electrónica para la lectura del puente de *Wheatstone* formado por la celda de carga y también un conversor ADC de 24 bits. Se comunica con el microcontrolador por medio de un protocolo de tipo serial mediante dos pines (*Clock y Data*).

Es decir hallar el factor de conversión para convertir valor de lectura en un valor con unidades de peso. La escala es diferente para cada celda y cambia de acuerdo a la forma de instalar, al peso máximo o modelo de celda de carga, incluso así se trate del mismo modelo de celdas no necesariamente tienen el mismo valor de escala.
Lo ideal es tener un objeto con peso conocido, es recomendable que el peso conocido sea cercano al valor máximo del rango de trabajo de la celda de carga.
El programa se ejecuta la primera vez sin colocar peso puesto que se debe calcular la tara de la celda esto es un valor que se debe restar para que la celda marque cero si no hay peso colocado.
Este valor es único para cada celda, incluso puede tener un ligero cambio con el tiempo de uso es por esto que normalmente las balanzas se re-calibran cada cierto tiempo.

Sensor de presión diferencial.

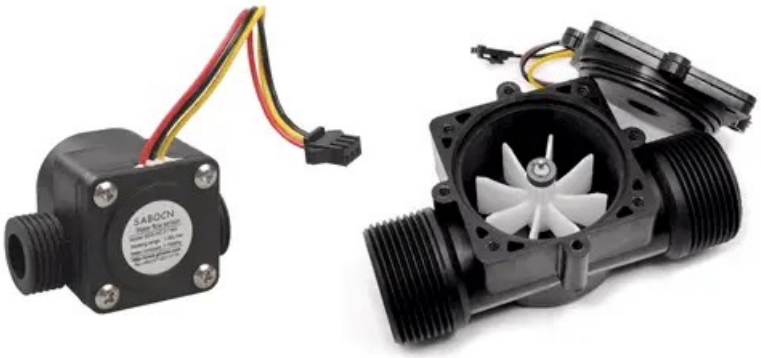
Existen diferentes sensores de presión diferencial en el mercado de los mas conocidos son los sensores tipo MPX.
Por ejemplo el MPX53DP, MPX53GP, MPX2010DP, MPX2010GP, MPX2050DP, MPX10DP, MPX5010DP, etc, y básicamente varían en la capacidad de presión que es capaz de medir el sensor.



Sensor de presión diferencial MPX5010DP.

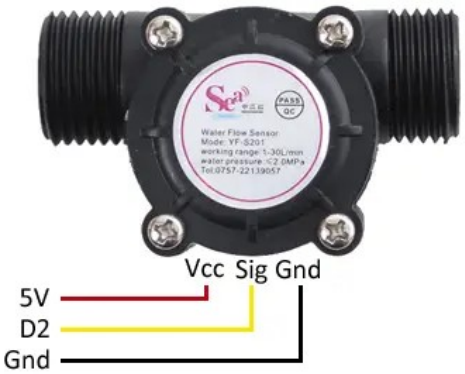
Dependiendo de la aplicación se selecciona el sensor adecuado, recordando que muchos de esos sensores necesitarán ser acoplados a circuitos amplificadores de voltaje para poder ser leídos por un microcontrolador.

Dentro del campo de caudalímetros que podemos emplear en nuestros proyectos de electrónica y domótica caseros tenemos diversos modelos como el YF-S201, FS300A, FS400A.
Cada uno dispone de distintas características, aunque el criterio de selección entre estos tres será el diámetro de la tubería.
Podemos usar un caudalímetro por ejemplo para saber cuanta agua está circulando por un sistema hidropónico, controlar el llenado de un depósito o controlar un sistema de riego, etc.
El caudalímetro YF-S201 con conexión de ½ pulgada es muy económica y fácil de conseguir..



Aspecto exterior e interior del caudalímetro YF-S201.

La conexión del caudalímetro es muy sencilla, por un lado alimentamos el sensor conectando Vcc y Gnd, respectivamente, a 5V y Gnd. Por otro lado, conectamos la salida del sensor SIG a un pin digital que permita emplear interrupciones por ejemplo en Arduino sería el pin 2.

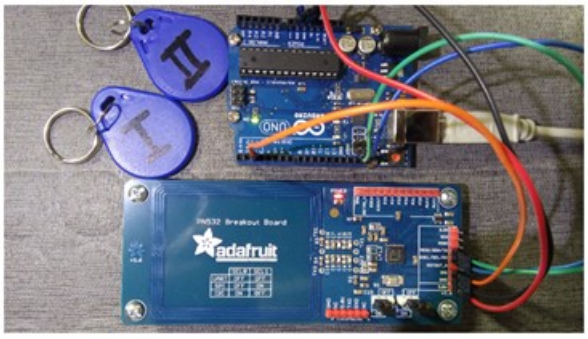


Conexionado del caudalímetro de media pulgada.

etiquetas semi-pasivas emplean una batería para hacer funcionar los circuitos del microchip, pero se comunican obteniendo energía del lector. Las etiquetas activas y semi-pasivas son útiles para realizar el seguimiento o rastreo de objetos de alto valor que deben ser escaneados a largas distancias, como ser los vagones de ferrocarril sobre rieles. Las etiquetas activas o semi-pasivas cuestan más que las etiquetas pasivas, lo que significa que no se las puede usar en objetos de bajo costo. Actualmente se está trabajando para bajar el costo de estas tarjetas.

Tags de solo lectura.

En el ejemplo propuesto usamos los típicos *tags* de uso común para reglamentar el ingreso a edificios, control de alarmas, etc. Su funcionamiento se basa en la señal que le llega de los lectores. Ésta induce una pequeña 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*.



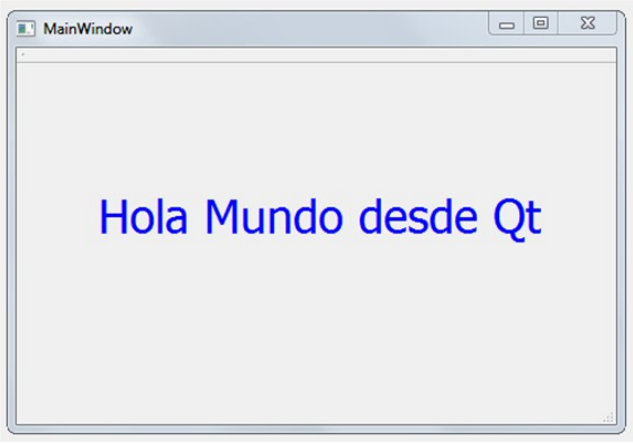
Placa con chip PN532 de Adafruit.

Esta placa es muy versátil (sin embargo con otras se obtienen los mismos resultados a un costo mas bajo), 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.

- QTextEdit: Es similara QLineEdit pero permitiendo mostrar texto, incluso en formato HTML.
- QPushButton: Widget Esta clase sirve para desplegar botones.
- QFrame: Widget Sirve para generar marcos.
- QMessageBox: Widget Despliega los Message Box.
- QLayout: Es la clase que gestiona la geometría de los objetos mostrados.
- QBoxLayout: Alinea los widgets de manera vertical u horizontal.
- QThread: Clase que gestiona hilos de ejecución independientes del SO que se esté usando.
- QTimer: Clase que proporciona una interfaz de alto nivel para los temporizadores.

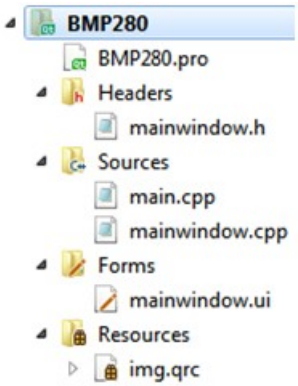
Mi primer programa con Qt.

Vamos a iniciar el primer ejemplo con el clásico “Hola mundo”.



No es la idea profundizar en los distintos aspectos que tiene la programación con QT ya que eso supera en mucho a estas pocas líneas donde se intenta presentar QT como una alternativa interesante para el diseño de interfaces gráficas usadas en electrónica.
Sin embargo y dado que es relativamente simple su programación intentaremos realizar algunos ejemplos simples iniciando con una ventana y un texto.
Una ventana simple con un objeto *Label* y un texto serán suficientes para este primer ejemplo.
Primero seleccionamos el tipo de programa que vamos a crear, en este caso será una aplicación Qt.

comandando por un temporizador que cada un segundo envía una “signal” que es recibida por un “slot” (método que atiende el evento del temporizador) y es en este método donde se decide que comando enviar.
El árbol de archivos que conforman el proyecto contiene el archivo PRO con la descripción general del proyecto, esto se puede ver en la siguiente imagen.



Los archivos *CPP* y *H* del método principal como de la ventana y un archivo de recursos para una imagen que se agrega a la ventana principal.
El resultado final de la interfaz que vamos a diseñar es la siguiente.

Resultado obtenido con el ejemplo propuesto.

Sin duda se pueden crear aplicaciones elegantes que no envidian en nada a las interfaces creadas por sistemas mas complejos y costosos.

Si no ha tenido éxito al abrir el puerto COM desplegamos un *QMessageBox* informando la novedad.

```
if(!arduino->isOpen()){
error = true;
QMessageBox::warning(this, "ERROR", "EL PUERTO YA ESTA ABIERTO
O NO EXISTE!!!");
}
```

Si todo ha salido bien nos aseguramos que todos los *buffers* del COM estén limpios antes de iniciar el dialogo con Arduino.

```
arduino->clear(QSerialPort::AllDirections);
```

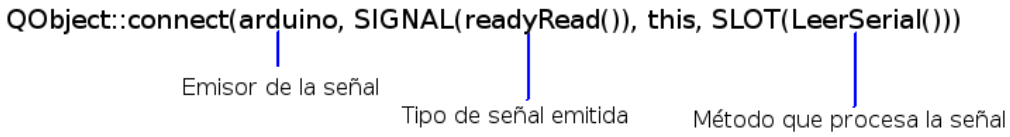
Paso siguiente configurar la velocidad en baudios que se ha seleccionado desde el *QComboBox* que hemos llamado *baudBox*.

```
arduino → setBaudRate(ui → baudBox → currentData().toInt());
```

El resto de los parámetros de configuración para el puerto COM se "programan" de manera directa en las siguientes líneas.

```
arduino->setDataBits(QSerialPort::Data8);
arduino->setFlowControl(QSerialPort::NoFlowControl);
arduino->setParity(QSerialPort::NoParity);
arduino->setStopBits(QSerialPort::OneStop);
```

Finalmente necesitamos conectar la señal generada cuando se recibe un dato con el método (slot) encargado de procesar este dato.

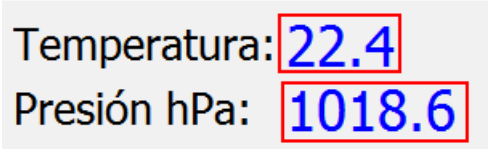


En la imagen anterior puede ver la forma en que se conecta una señal con su slot.
Luego si se ha establecido conexión con la placa Arduino es prolijo quitar la posibilidad de actuar sobre los QComboBox encargados de seleccionar el puerto y los baudios.

dato este almacenado de alguna transacción anterior fallida o abortada.
El slot encargado de interpretar los datos del puerto COM es LeerSerial() y su código es el siguiente.

```
void MainWindow::LeerSerial(){
if(estado == 1){
serialData1 = arduino->read(4);
ui->t->setText(serialData1);
serialData1.clear();
serialData2.clear();
estado = 2;
}
else if(estado == 2) {
serialData2 = arduino->read(6);
ui->p->setText(serialData2);
estado = 0;
serialData1.clear();
serialData2.clear();
}
}
```

En la ventana principal se han definido dos *Label*, uno identificado como *"t"* y el otro *"p"*colocados junto a otros *Label's* con un texto que identifica el tipo de datos que estamos viendo.
Cada uno de ellos destinado a mostrar la temperatura o la presión según sea el caso.
Un detalle para remarcar es que los datos enviados desde Arduino están en formato texto, una secuencia de caracteres ASCII contienen los datos del sensor BMP280.



En el programa Arduino se puede identificar las líneas en que el dato es formateado.

```
t = bmp.readTemperature();
dtostrf(t,2,1,buffn);
sprintf(Temperatura,"%s", buffn);
```

Primero se convierte la variable "t" (temperatura leída desde el sensor) a una cadena de texto, luego con función *sprintf()* da el formato final que se envía por la UART de la placa Arduino.

Conectando Arduino por Internet.

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.Sabido es que cualquier aplicación decente que pretenda competir en el campo del "Internet de las Cosas" o IoT debe tener conectividad a nivel de red.
Arduino UNO con su microcontrolador ATmega328 tiene un potencial de hardware mas bien acotado para ejecutar aplicaciones que requieran control sobre redes de datos, la solución viene con el chip WS5100 montado sobre el escudo para Ethernet.

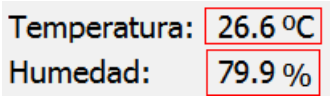
Conectando Arduino a Internet con el escudo Ethernet.

Este escudo se monta directamente sobre los pines de Arduino UNO y tiene incluso un zócalo para conectar una memoria SD pudiendo así alojar en esta memoria los futuros sitos web que el servidor embebido en Arduino manejará.
A continuación vemos un ejemplo con QT que mide temperatura y humedad con un sensor DHT22 mediante un socket UDP.
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.

Cuando se comento el programa *Arduino* para esta aplicación se hizo referencia al hecho que los datos son enviados en una larga cadena donde donde viene la temperatura y la humedad separados por una coma “,” en el programa *Arduino* la línea que hace esto es la siguiente.

sprintf(cadena,"%s,%s",Temperatura,Humedad);

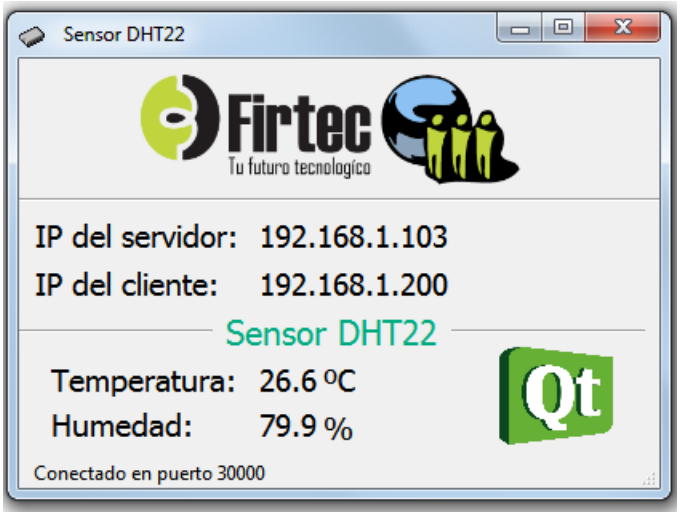
Observe que la coma es el separador de campo para los datos enviados, en Qt se divide la cadena enviada desde Arduino usando la “,” como separador de campo de datos.
La siguiente línea hace justamente eso *QStringList datos = recibido_socket.split(",")* .En la variable datos[0] se guardará la temperatura y en datos[1] el valor de humedad que se muestran los correspondientes *Label's* llamados *“l1”* y *“l2”*.



Es importante no confundir el *Label* que muestra los datos (*recuadros rojos*) con los *Label* que sirven de carteles indicadores de la variable recibida *“Temperatura:”* y *“Humedad:”*.
El siguiente es el código completo del archivo *mainwindow.cpp*.

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QNetworkInterface>
#include <QUdpSocket>
MainWindow::MainWindow(QWidget *parent) :
QMainWindow(parent),
ui(new Ui::MainWindow){
ui->setupUi(this);
QList<QHostAddress> list = QNetworkInterface::allAddresses();
foreach (const QHostAddress &address,
QNetworkInterface::allAddresses()) {
if (address.protocol() == QAbstractSocket::IPv4Protocol &&
address != QHostAddress(QHostAddress::LocalHost))
string = address.toString();
ui->iplocal->setText(string);
}
QHostAddress address(QHostAddress::Any);
address.setAddress(string);
socket = new QUdpSocket(this);
socket->bind(address , 30000);
// Conecta una dirección y un puerto a un socket
statusBar()->showMessage(tr("Desconectado!!"));
connect(socket, SIGNAL(readyRead()), this,
SLOT(processPendingDatagrams()),Qt::QueuedConnection);
```

```
}
MainWindow::~MainWindow(){
delete ui;
}
void MainWindow::processPendingDatagrams() {
QHostAddress sender;
uint16_t port;
QByteArray datagram;
QString dato1;
QString dato2;
datagram.resize(socket->pendingDatagramSize());
// Tamaño del datagrama a leer.
while (socket->hasPendingDatagrams()) {
socket-
>readDatagram(datagram.data(),datagram.size(),&sender,&port);
ui->IP->setText(sender.toString());
// Mustra la IP a la que se conecta
QString recibido_socket(datagram);
// Convierte QByteArray a QString
QStringList datos = recibido_socket.split(",");
// Busca la marca separadora de campos ","
ui->l1->setText(datos[0]);
ui->l2->setText(datos[1]);
statusBar()->showMessage(tr(" Conectado en puerto
%1").arg(port));
}}
```



Resultado obtenido con el ejemplo anterior.

Control de pines Arduino + Qt + Socket de red.

En el ejemplo anterior la placa *Arduino* envía los datos de un sensor *DHT22* a un programa Qt mediante un socket UDP, es decir que el programa Qt solo

puede leer el voltaje presente en el canal analógico A0.
Sin duda el resultado final de las pantallas obtenidas son muy satisfactorias incluso para aplicaciones comerciales .
En los pines del Arduino podríamos tener conectados relevadores u opto-acopladores para controlar luces, motores, etc y en lugar del voltaje podríamos estar leyendo una temperatura o cualquier dato de un sensor.

En estos ejemplos se han agregado imágenes a la ventana principal, cosa que no es necesaria para su funcionamiento pero mejora la estética.
Estos ejemplos suponen que un específico programa QT se está ejecutando en un computador.
Esto significa que en cada computador deberá existir el programa QT.
Sería incluso mejor que cualquier usuario con un simple navegador pudiera tener los mismos datos ya sea en una tableta, móvil o computador.
Entonces porque no construir una "*pantalla*" que se pueda ver desde cualquier lado y con cualquier equipo que tenga un navegador de esta forma la electrónica y pantalla gráfica la pone el propio usuario con su móvil.

Servidor Web con Arduino.

Lo primero a tener claro es que estamos trabajando con las mismas herramientas que se usan en los grandes sistemas informáticos pero está claro que no es lo mismo porque el escenario es distinto.
Lo mismo pasa con la programación de webs embebidas, en esencia es lo mismo pero también diferente, aquí buscamos que códigos en la web activen electrónica o lea una temperatura mientras que en la "*web de computadoras*"

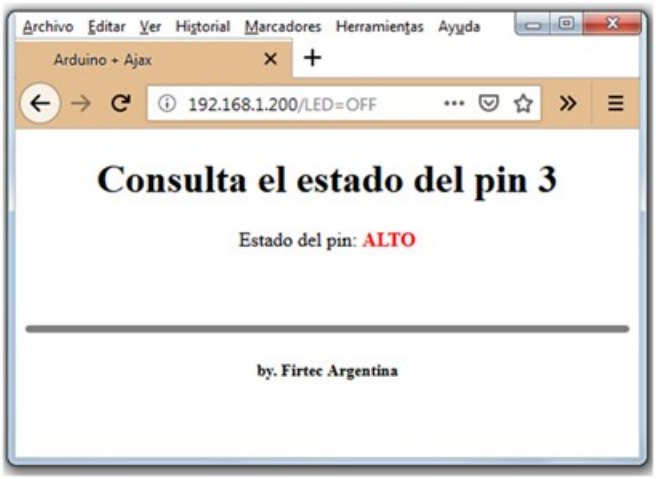
```
navegador.println(F("Estado del pin:<font color='green'><b>
BAJO</b></font>"));
}
}
```

Lectura de un pin Arduino con AJAX.

Viendo el funcionamiento del ejemplo anterior se hace evidente lo molesto que es tener que actualizar toda la página web cada vez que se quieran visualizar nuevos datos.
El siguiente ejemplo no presenta grandes cambios en la parte gráfica, la estética de la web es la misma salvo que no hemos dado color al fondo y se ha cambiado el texto de la pestaña que muestra el navegador.
Sin embargo lo que si ha cambiado totalmente es su comportamiento, ya no es necesario cargar toda la pagina web para actualizar los datos. Una función en Java Script sen encarga de manejar los datos que el servidor envía a la función Ajax identificada con el nombre es ***EstadoDelPin()***, para marcar donde inicia un código Java Script primero se envían los marcadores que la definen.

```
navegador.println(F("<script>"));
.
.
navegador.println(F("</script>"));
```

Luego del marcador del inicio del código Ajax se declara el nombre de la función.
navegador.println(F("function EstadoDelPin() {"));



El ejemplo anterior pero usando Ajax.

Sin embargo para entender el funcionamiento del Java Script será necesario profundizar en algunos conceptos que hacen a su funcionamiento.

Está claro que la visualización de los datos enviados por los sistemas electrónicos es parte fundamental en todo proyecto ya sea para uso personal o comercial.
Una plataforma interesante para construir interfaces muy funcionales es Python.

Hablemos de 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.
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.

En la misma ventana se leen los datos de conexión.

Recuerde es necesario que la placa Arduino se conecte a la IP y puerto del servidor de lo contrario la conexión nunca ocurrirá.
El funcionamiento de este programa Python es muy sencillo, cuando se oprime el botón Apagar LED se envía el carácter “4” y cuando se oprime Encender LED se envía el carácter “5”. Estos caracteres se interpretan como comandos que son decodificados por Arduino para controlar el estado del pin 5.
El código Pyton es el siguiente.

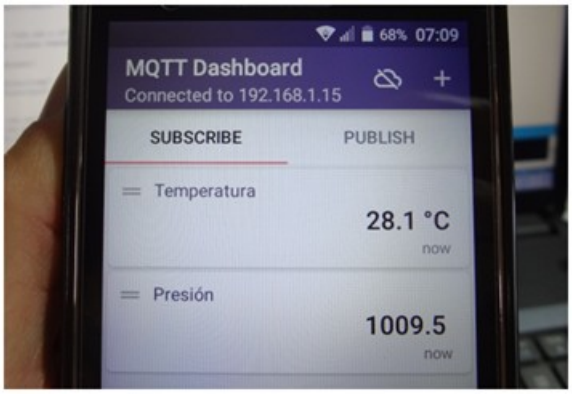
```
# -*- coding: utf-8 -*-
#!/usr/bin/env python
import socket
import sys
import errno
import time
from tkinter import *
from tkinter.messagebox import showinfo
from tkinter import Frame
from tkinter import Text
from tkinter import Label
bandera = 0
class MyGui(Frame):
    def __init__(self, parent=None):
        Frame.__init__(self, parent)

#*** Función para conocer el IP del servidor *****
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'
```

es bastante mas lento pero mas seguro. Elegir el nivel de seguridad depende de lo que estamos transmitiendo, generalmente con el nivel 0 es suficiente.

Un ejemplo con el sensor BMP280 y MQTT.

Como vimos en ejemplos anteriores el BMP280 mide la presión barométrica y la humedad.
En realidad este sensor también se puede usar como altímetro sin embargo en este ejemplo solo mostramos presión y temperatura.
Para el manejo del sensor se debe cargar la biblioteca correspondiente desde el administrador de librerías igual que en los casos anteriores.
Para visualizar los datos hemos usado uno de los tantos *brokers* disponibles en el *Play Store* y el resultado es el siguiente.



Arduino tiene una biblioteca para MQTT, *PubSubClient* es la encargada del control de MQTT y el broker que usemos para suscribirnos a los datos maneje la versión 3.1.1 de MQTT (muchos solo manejan 3.1.0)
El código completo para el manejo del sensor del sensor BMP280 y publicación de datos en la red MQTT es el siguiente.

```
#include <SPI.h>
#include <Ethernet.h>
#include <PubSubClient.h> // Biblioteca MQTT
#include "SPI.h"
#include <Adafruit_Sensor.h>
#include "Adafruit_BMP280.h"
Adafruit_BMP280 bmp;
byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
// IP del servidor MQTT
const char* mqtt_server = "192.168.1.15";
#define TEMP_TOPIC "firtec/temp"
#define PRES_TOPIC "firtec/pre"
float p;
```