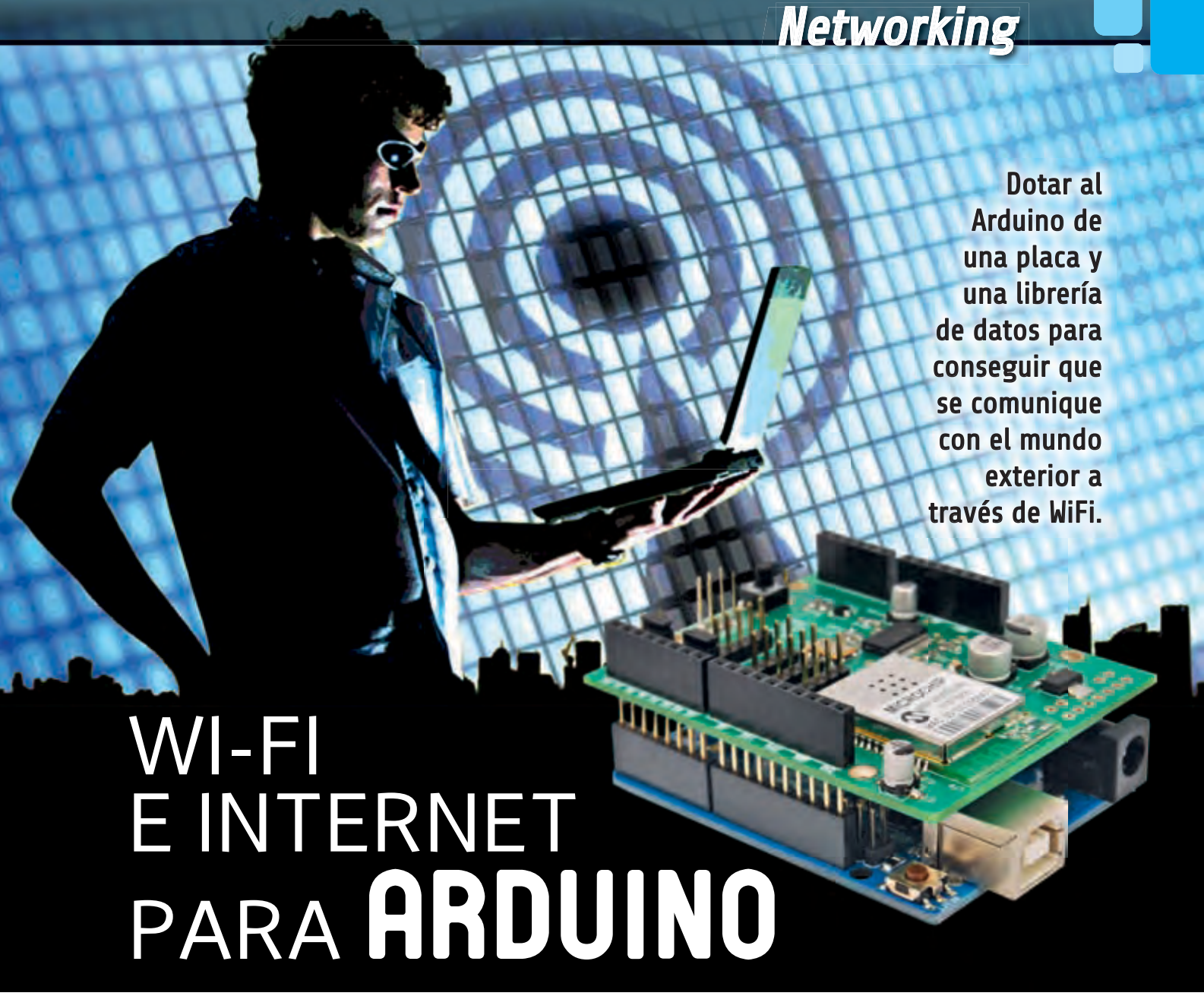


Dotar al Arduino de una placa y una librería de datos para conseguir que se comunique con el mundo exterior a través de WiFi.

WI-FI E INTERNET PARA ARDUINO

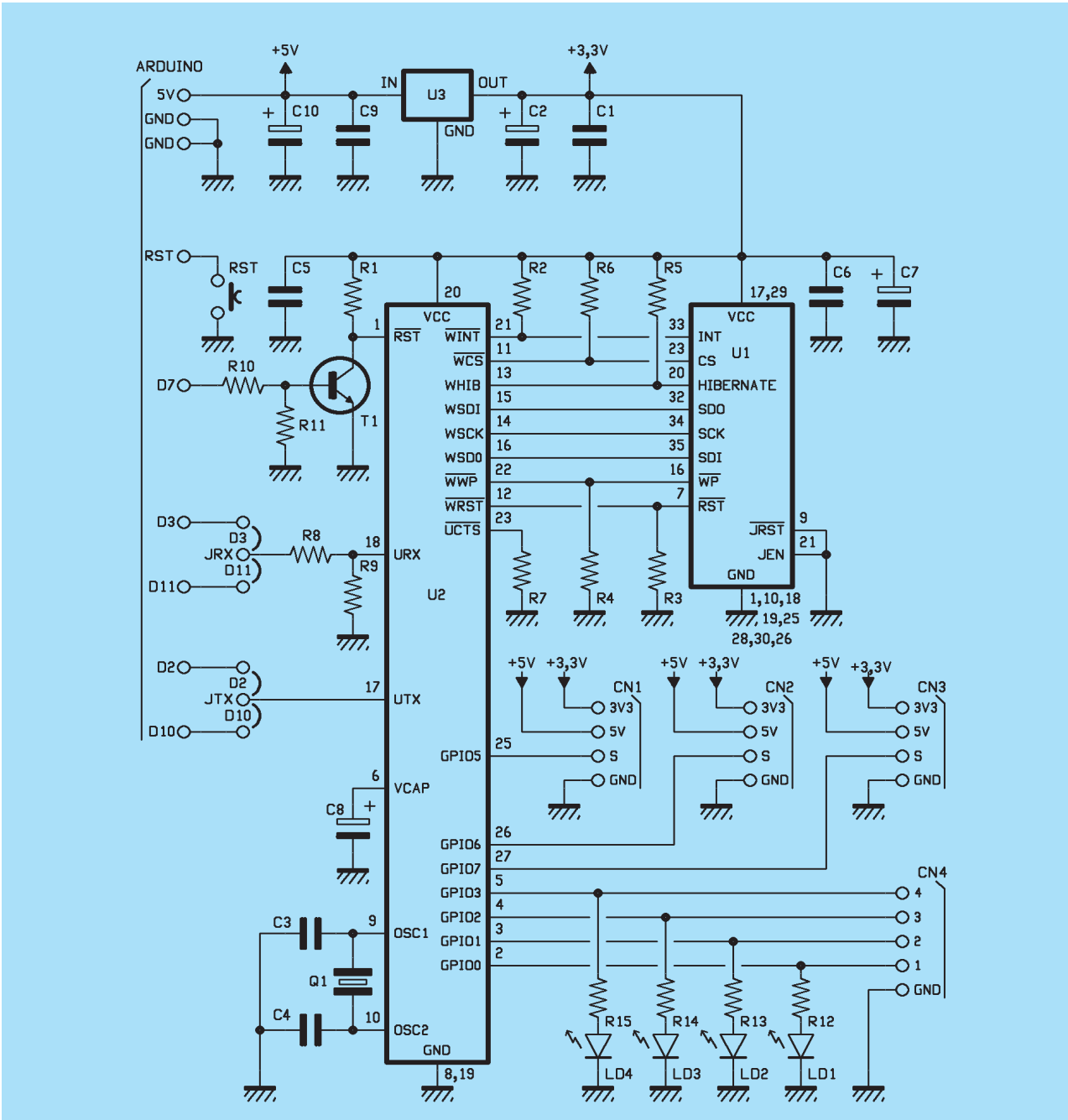
DANIELE DENARO

La placa Arduino ha supuesto una verdadera revolución en el campo de la electrónica, demostrándose particularmente adecuada para los no expertos y al mismo tiempo lo suficientemente versátil como para ser capaz de manejar una amplia gama de aplicaciones en diversos campos. La hemos visto, por ejemplo, afrontar tareas complicadas, como la conexión con el mundo exterior, bien usando las simples interfaces básicas (serie, I²C-Bus), bien con lo que llamamos conectividad real, es decir, interfaces inalámbricas y red Ethernet. En ambos casos, los límites de Arduino se manifiestan con claridad, porque la placa del equipo Aruduino, como sabes, se basa en un hardware mínimo, mientras las conexiones con las redes y aún más la conectividad a Internet, requieren una gestión asincrónica y protocolos de comunicación esencialmente multitarea. Por esta razón, la conexión de Arduino con Internet siempre ha sido un poco dificultosa, tanto como para empujar a sus creadores a implementar un sistema híbrido como el YÚN, donde el procesador Atheros tiene la función principal de administrar las comunicaciones. YÚN sin embargo, no es la única solución. De hecho como demostraremos en estas páginas, Arduino puede disponer de conectividad a Internet a través de una placa dotada con un

administrador de pila TCP/IP, liberando al ATmega de algunas tareas básicas. Pero incluso con el gestor, todavía es necesario superar varias dificultades en la gestión de las conexiones a Internet, tanto con el sencillo Socket2 (ver más adelante), como con el protocolo Http (el utilizado por el mundo de la Web). Por esta razón, es esencial acompañar la tarjeta de una librería que, además de comunicarse con el administrador del protocolo TCP/IP, simplifique al máximo la vida de quién quiere programar el Arduino y quiere comunicarse con otros equipos a través de Internet. Empecemos con la descripción del hardware del nuevo módulo WiFi y posteriormente describiremos la librería de los equipos y su uso.

EL MÓDULO WIFI

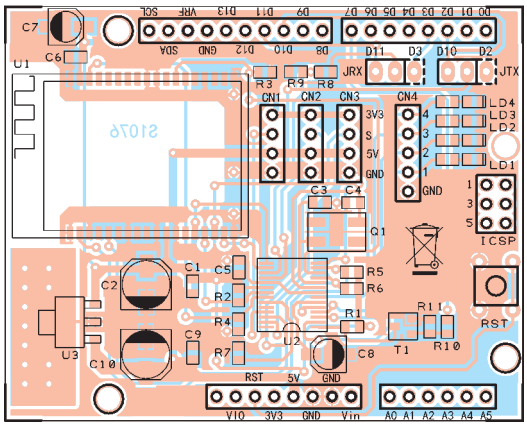
Si recuerdas, hace ya algún tiempo (en concreto, en la edición n° 160 de Elettronica In) se publicó un proyecto para dotar a una placa de Arduino de conectividad

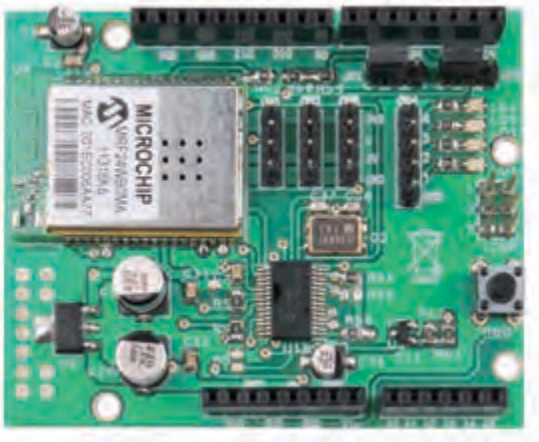


Lista de materiales

R1, R4, R11: 10 kohm (0805)
R2, R10: 4,7 kohm (0805)
R3: 100 kohm (0805)
R5: 1 Mohm (0805)
R6: -
R7: -
R8: 1 kohm (0805)
R9: 1,5 kohm (0805)
R12÷R15: 330 ohm (0805)
C1, C5, C6, C9: 100 nF multicapa (0805)
C2, C10: 220 μF 6,3 VL electrolítico (D)
C3, C4: 22 pF cerámico (0805)
C7, C8: 10 μF 35 VL electrolítico (B)
U1: MRF24WB0MA/RM
U2: MCW1001A
U3: TC1262-3.3 (SOT-223)
T1: BC817
Q1: Cuarzo 8 MHz (HCX-7SB) > RS: 675-4703
RST: Micro interruptor
LD1÷LD4: LED verde (0805)

Varios:
- Tira de pines macho 3 vías (2 pzas.)
- Tira de pines macho 4 vías (3 pzas.)
- Tira de pines macho 5 vías (1 pzas.)
- Tira de pines macho/hembra 6 vías (1 pzas.)
- Tira de pines macho/hembra 8 vías (2 pzas.)
- Tira de pines macho/hembra 10 vías (1 pzas.)
- Tira de pines macho/hembra 3 vías (2 pzas.)
- Puente (2 uds.)
- Circuito impreso





WiFi; el que se describe aquí es una versión mejorada del anterior, que se basaba en un módulo Microchip que ahora está fuera de producción, y dispone de hardware que le permite gestionar por sí mismo el protocolo TCP/IP. Más exactamente, la diferencia sustancial entre la placa antigua y esta nueva está en que en la primera era el Arduino (mediante una librería adecuada) quién controlaba el módulo WiFi y manejaba el protocolo TCP/IP, mientras que en este nuevo circuito, hemos insertado un procesador del interfaz que se ocupa de la gestión del protocolo TCP/IP, descargando así al Arduino de una tarea que, por la experiencia del primer módulo, llegó a ser demasiado exigente.

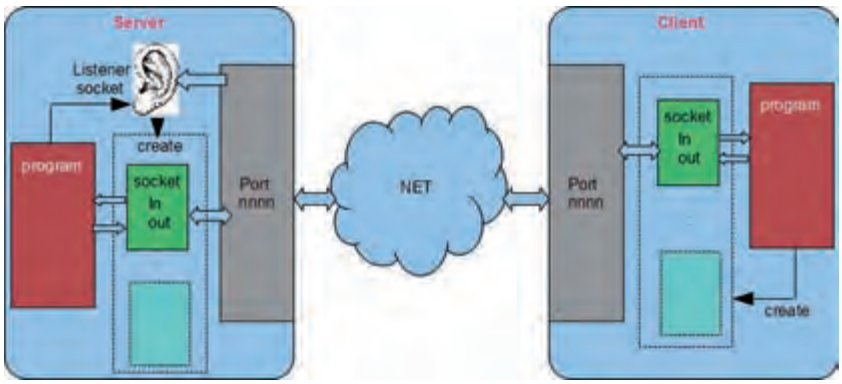
Sin embargo, una tarjeta de red llega a ser realmente útil si está controlada por un software que simplifique tanto como sea posible su uso; por este motivo, hemos dotado a la tarjeta de una librería que teníamos intención de utilizar en Arduino, con un grado de simplificación mayor que la de un hardware más complejo (aunque más flexible) como la de YÚN. Llegando a la descripción del hardware, podemos ver que el shield (que tiene el mismo tamaño que una palca de Arduino) se basa esencialmente en un módulo WiFi MRF24WB0MA y un procesador MCW1001A, ambos Microchip.

La fuente de alimentación es de 3,3V (obtenida mediante un regulador de tensión alimentado desde la línea 5V de Arduino) pero el procesador soporta también niveles TTL.

El módulo de radio tiene la antena integrada.

El procesador gobierna el módulo WiFi a través del bus SPI (líneas SDO, SDI, SCK) y otras líneas de control como CS, INT, RST, HIBERNATE, WP; de lado del host (es decir, Arduino) el procesador está conectado a una interfaz en serie de dos hilos en lógica TTL (RX,TX). Esto nos obliga a utilizar un puerto serie simulado por software con una versión personalizada de la librería SoftwareSerial. La personalización es necesaria a causa de la

Fig. 1 *- Comunicación entre dos aplicaciones.*



modalidad con dos bit de stop, pero también para ajustar la velocidad optima y hacer transparente el uso de la librería. De hecho esta versión del SoftwareSerial se incluye en la librería. Este método de conexión, sin embargo, impide la posibilidad de utilizar la librería del SoftwareSerial estándar para otros usos en el mismo sketch. El puerto serie simulado se puede colocar, a nivel de hardware, sobre dos pares de pines de Arduino: usando dos puentes se puede escoger si se utiliza D3 o D11 para el canal RX (D3 es el valor predeterminado) y D2 o D10 para el TX (D2 es el predefinido). En el caso de que decida modificar los valores predefinidos, debe editar el archivo MWIFI.h y modificar las líneas:

```
#define RXPIN 3
#define TXPIN 2
```

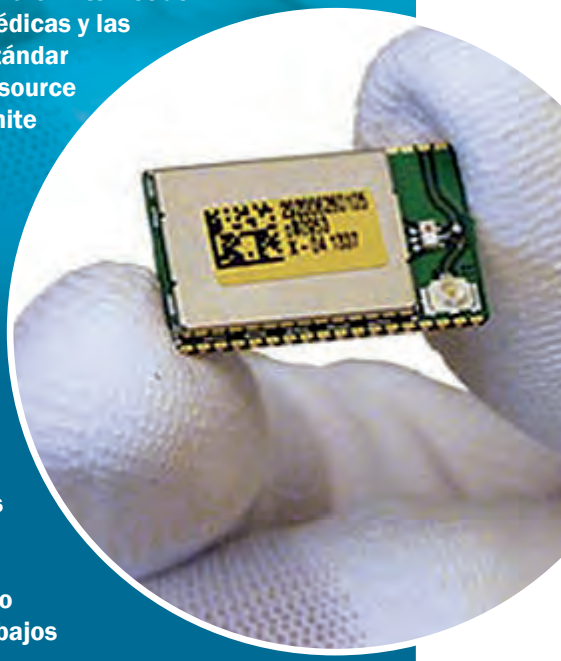
Los pines seleccionados, junto con el pin D7 utilizado para restablecer la tarjeta WiFi, no pueden utilizarse para otros fines. Pero el procesador MCW1001A proporciona pines digitales adicionales, de los que los cuatro primeros están conectados a otros tantos LED, mientras que los otros tres hemos están presentes en otros tantos conectores junto a la alimentación positiva (5V o 3V3) y a GND. De hecho, estos tres pines (GPIO5, GPIO6, GPIO7) pueden suministrar también niveles de 5V con una corriente máxima de alrededor de 25 mA (IN/OUT). Los pines GPIO se pueden desactivar con funciones de la librería, pero los dos primeros LED están reservados a la librería que señalan inicio y conexión en red. Como se ha mencionado, el procesador MCW1001A se ocupa de la pila TCP/IP, administrando el protocolo básico mediante la utilización de su RAM interna para la trasmisión y recepción, así como para el almacenamiento de varios parámetros. Hay que decir que no dispone de memoria no volátil interna, el procesador debe ser reconfigurado en cada reinicio. El protocolo de comunicación entre el host y el procesador y sus funciones están bien descritas en la hoja de características del componente, que se encuentra en la página web: ***www.microchip.com/wwwproducts/Devices.aspx?dDocName=en555844***. En todo caso, la gestión de bajo nivel se asigna a la librería que proporciona un conjunto de funciones para abrir el socket, o administrar el protocolo HTTP. La placa tiene cuatro LED, dos de los cuales son utilizados por la librería. Más precisamente: el LED0 se activa al encender y poner en marcha la tarjeta mientras el LED1 se enciende cuando se establece la conexión con la red. Los otros dos pueden ser utilizados por el usuario. Para comprender las opciones de diseño en términos de hardware y software de Arduino, usamos un par de párrafos para describir los problemas inherentes a la comunicación TCP/IP: a nivel de aplicación, cuando un equipo decide ponerse en contacto con otro, es necesario que éste último se encuentre en escucha de los elementos que están enviando los datos; por tanto, siempre hay una figura de *servidor* (y listener, o sea el oyente) y una figura de *llamante* (cliente), de otra forma sería imposible sincronizar la llamada. La librería cubre estos dos tipos de funciones: por ejemplo, se puede utilizar cuando se quiere dar al Arduino la tarea de servidor, de modo que reacciona y responde a los mensajes de otro equipo. O bien puede ser utilizado para dar al Arduino la posibilidad de enlazar por su iniciativa con un equipo servidor para enviar, por ejemplo, las mediciones realizadas por los sensores. La conexión se establece en TCP/IP a través de las entidades denominadas socket, que se corresponden con el concepto de "toma" que conecta un cable de comunicación. Existe por lo tanto un socket para cada uno de los dos equipos que necesitan comunicarse entre ellos. Un socket se crea como parte de un número de puerto. Los números de puerto son subdirecciones de la aplicación en una dirección IP. Los números inferiores a 1.000 son reservados a los servicios públicamente reconocidos como Mail, Ftp, Web (puerto 80), y así sucesivamente.

LA LIBRERÍA MWIFIS

Bueno, en éste punto, podemos describir la librería para Arduino que implementará la gestión de conexión en TCP/ IP. La librería se puede descargar desde nuestro sitio web ***www.elettronicain.it*** junto a los demás archivos del proyecto. Vamos a proceder con orden: el MwiFi va situado

WiFi 5 GHz + Bluetooth

ConnectBlue presenta OWL355, un módulo WLAN capaz de funcionar tanto en la banda clásica a 2,4 GHZ, como en la nueva a 5 GHz, así como interconectar con cualquier estándar Bluetooth. Diseñado para el Internet de las cosas, las aplicaciones biomédicas y las industriales, supera el actual estándar IPC que es AQL. Gracias al open source Linux host driver, el módulo permite implementar con facilidad una conexión wireless en cualquier dispositivo electrónico, proporcionando hardware modular homologado, con certificación EMC, calificación Bluetooth, una amplia gama de antenas. Entre otras características, señalar la tecnología Castellation Package que utiliza los relieves metálicos a los lados del módulo para hacer mucho más fácil la soldadura (también manualmente) permitiendo el uso del módulo en aplicaciones con bajos volúmenes de producción.



(después de descomprimirlo) en el directorio "libraries" del IDE de Arduino, al igual que las otras librerías; se debe utilizar incluyendo en el sketch el archivo <MwiFi.h >.
Se llama como un objeto y se puede utilizar; la primera función llamada es begin (), que sirve para inicializar la tarjeta:

```
#include <MWiFi.h>
MWiFi WIFI;
void setup()
{
 WIFI.begin();
  :
```

El arranque de la tarjeta determina el encendido del primer LED. Probamos, ahora, a conectar Arduino en red utilizando un punto de acceso, que puede ser el router WiFi de casa, por ejemplo asignando el nombre (SSID) "D-Link-casa":

```
WIFI.ConnSetOpen("D-Link-casa");
```

válido en el caso de que la red no esté protegida, o bien:

```
WIFI.ConnSetWPA("D-Link-casa","mipassword");
```

en el caso de que la red tenga protección WPA con contraseña "`mipassword`". Las funciones anteriores se utilizan para preparar la conexión de la tarjeta; pero la verdadera y real conexión se obtiene llamando a la función:

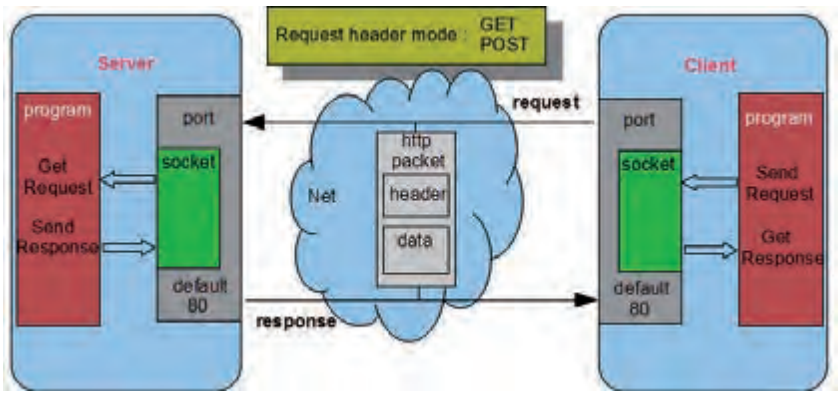
```
WIFI.Connect();
```

Si la conexión se realiza correctamente, se enciende el segundo LED. Hay que tener en cuenta que en el caso de que la red esté protegida, la conexión puede tardar más de medio minuto, ya que el controlador de la tarjeta debe codificar la clave con la contraseña. La versión actual de la librería proporciona un reset automático de Arduino en caso de que la conexión se pierda; la restauración automática se produce incluso si los errores son detectados por el controlador. De este modo, se evita un posible bloqueo y el sistema puede ser "unattended" (funcionar sin supervisión humana). También se han añadido funciones de conexión directa, que preparan la conexión y la ejecutan en un único paso. Además se ha añadido la posibilidad de generar la clave de contraseña numérica, para utilizar sucesivamente ésta en lugar de la contraseña; de hecho el acceso con contraseña, como necesita cada vez la elaboración de la clave numérica, puede tardar hasta un minuto, mientras que el acceso con clave es muy rápido. El router asigna al Arduino una dirección IP dinámica, porque ese es el comportamiento por defecto de la tarjeta; pero si se desea se puede imponer una dirección IP fija. La dirección asignada puede ser necesaria para una función de la librería.
En este punto elegimos hacer funcionar Arduino como servidor: primero llamamos a la función **openServerTCP()**, que crea el oyente sobre un puerto determinado (por ejemplo 5000), y luego ponemos en bucle la recepción de una posible solicitud de link (enlace):

```
int ssocket=WIFI.openServerTCP(5000);
void loop()
{
 int csocket=WIFI.
pollingAccept(ssocket);
  :
```

Las variables enteras ssocket y csocket son de las referencias (handle), respectivamente, para el servidor y la conexión del socket con cualquier equipo que quie-

Fig. 2 *- Comunicación HTTP.*



ra efectuar el link. La función **pollingAccept()** devuelve un número menor de 255 si se ha solicitado el enlace, o 255 si no hay ninguna solicitud de conexión entrante. En el caso de que la conexión haya sido establecida podremos enviar o recibir mensajes haciendo referencia a este csocket. Por ejemplo, para recibir un registro, que es una cadena que termina con un salto de línea, puede utilizar la función:

```
char *line=WIFI.
readDataLn(csocket);
```

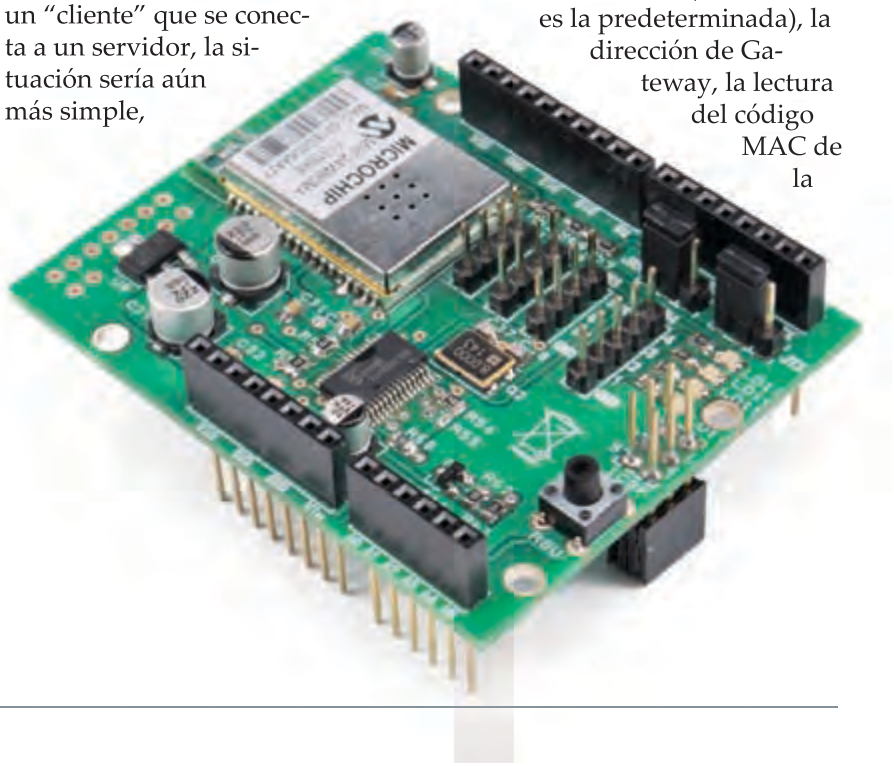
Será restituido una "cadena terminada null" pero sin el line-feed. En este caso, la librería utiliza un buffer predefinido de 81 caracteres (pero su longitud puede ser modificada en su definición) por eso no es necesario que nos proporcione dicha información. Sin embargo, hay otras posibilidades. Si queremos responder podemos utilizar la función:

```
WIFI.writeDataLn(csocket,answer);
```

La variable que llamamos answer (respuesta) corresponde a un buffer de char, pero tiene que ser una "cadena terminada en null". Utilizar un "cadena terminada en null" significa que nosotros no proporcionamos la longitud de los caracteres útiles en la matriz, porque la función la calcula automáticamente, pudiendo basarse sobre el carácter de terminación null. La mayor parte de las funciones en C administran y producen este tipo de cadena, que no debe confundirse con el objeto String presente también en el lenguaje de referencia de Arduino. De este simple modo hemos establecido y utilizado una conexión WiFi con un equipo remoto. En la librería, entre los ejemplos indicados, hay un ejemplo de servidor llamado *CommandServer*, que permite controlar a Arduino utilizando un programa del tipo *telnet* en el equipo remoto. Para simplificar las pruebas se añadió un programa de Java que funciona como telnet. Si, en cambio, quisiéramos hacer actuar a Arduino como un "cliente" que se conecta a un servidor, la situación sería aún más simple, porque tenemos que crear solo un socket de enlace, así:

```
int csocket=WIFI.openSoc --
kTCP("192.168.1.2",5000);
```

Y si csocket es válido (menor de 255), quiere decir que se ha establecido la conexión con el ordenador a la dirección 192.168.1.2 sobre el puerto 5000. En este punto podemos utilizar las funciones de lectura y escritura vistas anteriormente. Entre los ejemplos, hay uno (SendData) que se enlaza a un servidor remoto para enviar a intervalos regulares las lecturas de los sensores; en este caso es necesario un programa de servidor instalado en el equipo remoto. Para facilitar las pruebas, junto con la librería se proporciona un programa Java que recibe los datos y los descarga en un archivo añadiendo una marca de tiempo (para obviar la falta de un RTC, Real Time Clock, en Arduino). Además de estas características básicas, la librería cuenta con todas las funciones necesarias para definir diversos parámetros como la dirección de enmascaramiento de la red (255.255.0.0 es la predeterminada), la dirección de Gateway, la lectura del código MAC de la



tarjeta, etc. En particular, están presentes las funciones para detectar puntos de acceso presentes y visibles en el entorno.
Por ejemplo, para detectar todas las redes presentes, se usa la instrucción:

```
int nn=WIFI.scanNets();
```

La variable entera nn contendrá el número de redes detectadas. En cambio la función:

```
char net=WIFI.getNetScanned(i);
```

restituirá las características (bajo la forma de registros) del "i-esima" red detectada. Por último, por conveniencia, se proporciona una función que restituye el nombre de la red no protegida que proporciona la señal más potente. Nos referimos a la documentación de la librería para la descripción de todas las funcionalidades disponibles. La librería contiene un *help* (ayuda) y está documentada en los archivos de código (en particular los archivos .h). Pero la librería no se limita a la simple gestión de conexión y socket; de hecho, en ella se incluye una clase derivada (y por lo tanto especializada) que administra el protocolo HTTP. Éste último es un protocolo que proporciona siempre una solicitud y una respuesta; tanto la solicitud como la respuesta hacen viajar por la red paquetes formados de algunos encabezamientos (header) y datos reales (tales como las páginas html, imágenes, video o también simple texto). Para descargar al usuario de todos estos problemas, la librería de HTTP se molestó en dar forma a estos paquetes utilizando, además, la modalidad PROGMEM, es decir, la posibilidad de colocar en el área Flash constantes de programa y en particular los textos. Hay que tener en cuenta que el protocolo HTTP es un protocolo textual: utiliza únicamente caracteres. El uso de la memoria Flash para los textos permite ahorrar espacio en la memoria RAM de Arduino.

Listado 1

```
WEBRES rs[8]=
{
 {"/index",pindex},
 {"/Analog",panalog},
 {"/RDigital",rdigital},
 {"/Wdigital",wdigital},
 {"/wdig",wdig},
 {"/Pwm",pwmpage},
 {"/PwmSet",pwmset},
 {"/End",sessend}
};

void loop()
{
  WIFI.getRequest(csocket,8,rs);
    :
```

Listado 2

```
prog_char pagerdigital[] PROGMEM=
      :
"<tr>"
"<td><div align='center'>@</div></td>"
"<td><div align='center'>@</div></td>"
"<td><div align='center'>@</div></td></tr>"
      :

void rdigital(char *query)
{
  char *val[3];
  if(digitalRead(4)) val[0]=ON;else val[0]=OFF;
  if(digitalRead(5)) val[1]=ON;else val[1]=OFF;
  if(digitalRead(12))val[2]=ON;else val[2]=OFF;
  WIFI.sendDynResponse(csocket,pagerdigital,3,val);
}
```

LA LIBRERÍA HTTP

Siendo una clase derivada de MwiFi, la librería HTTP hereda todas las funciones de MwiFi; sin embargo, si desea utilizar las nuevas funciones es necesario incluir el archivo HTTPlib.h (en lugar de MwiFi.h) y crear una instancia de un objeto HTTP:

```
#include HTTPlib.h HTTP WIFI;
```

Respecto a la conexión con un punto de acceso y a la gestión del socket, todo permanece como antes (tal vez hora escogeremos el puerto 80); pero también esta vez hay que decidir si hacer funcionar al Arduino como servidor (es vez del servidor Web) o como cliente que se accede a un servidor de aplicaciones Web (como Tomcat, GlassFish, Jboss, PHP, etc.). Supongamos que desea crear un servidor Web para ser consultado por cualquier browser (navegador): para hacer funcionar a Arduino como un servidor Web tenemos que preparar los recursos que él puede poner a disposición, es decir, las páginas html de respuesta. Estas páginas se almacenan en áreas PROGMEM por los motivos mencionados anteriormente. Por ejemplo:

```
prog_char pageindex[] PROGMEM=
"<html><head>"
"<title>Index</title>"
```

En este punto, tenemos que conectar estos buffer de memoria con los nombres de los recursos invocados a través del browser. Los nombres de los recursos son la parte de la ruta de acceso local de la dirección URL (o URI), que es, en esencia, el nombre del archivo en la URL completa (por ejemplo: http:// www.mio.es/mipag.html). En este contexto, los recursos mínimos para ser invocados se identifican sólo por su nombre sin extensión. Así que se trata de asociar una página memorizada con su nombre de Internet correspondiente (por ejemplo "/indice"). En realidad esta página no lo enviará sólo puesto que necesita enlazar el nombre del recurso con una función que se ocupará de enviarlo. Para hacer el mecanismo

Listado 3

```
void pwmset(char *query)
{
  char *pwmval;
  pwmval=WIFI.getParameter(query,strlen(query),"PWM10");
  if (pwmval!=NULL)
   {int pv;sscanf(pwmval,"%d",&pv);
    analogWrite(10,pv);d10=pv;}
  pwmval=WIFI.getParameter(query,strlen(query),"PWM11");
  if (pwmval!=NULL)
    {int pv;sscanf(pwmval,"%d",&pv);
     analogWrite(11,pv);d11=pv;}
   pwmpage(query);
}
```

lo más automático posible, se ha preparado una estructura o más exactamente un *tipedef* llamado WEBRES; esta estructura se forma de la unión de dos campos: el nombre del recurso y el nombre de la función (que en C corresponde a una dirección). Se trata de formar tantas parejas nombre-función para pasar a la función **getRequest()**, que se ocupará de poner en marcha la función correcta (función call-back), o enviar un mensaje estándar "Not Found" de respuesta, en caso de que el nombre no coincida con ninguno de los predispuestos. En el Listado 1 se muestra un ejemplo de la construcción de un conjunto de 8 estructuras WEBRES y su inclusión en la llamada a la función **getRequest()**. Cada segundo campo de la estructura corresponde a la función call-back que **getRequest()** lanzará. La función de call-back (devolución de llamada) se tendrá que preocupar de enviar el buffer correspondiente a la página elegida y su prototipo se espera que sea de tipo void (es decir, de retorno nulo) y tiene un solo argumento; en la práctica, un puntero de una cadena terminada en cero suministrada por la persona que llama (véase a continuación):

```
void pindex(char *query)
{
  WIFI.sendResponse(csocket,pagei
ndex);
}
```

En resumen: **getRequest()** puesto en el bucle, se hará cargo de toda la gestión de la solicitud y detectará la modalidad utilizada GET o POST, comportándose en consecuencia (los datos están de diferente modo) y lanzará la función correspondiente a la solicitud (o el mensaje "Not Found"). La función de call-back **pindex()** descrita anteriormente, sin embargo no hace más que enviar en respuesta una página html estática, que se define de una manera fija.
El servidor web Arduino, definido así, no es muy útil, porque se supone que se puede utilizar para leer los valores proporcionados por los sensores (Fig. 4) o activar salidas (Fig. 5); para lograr esto, la página html de respuesta tiene que construirse en el momento, insertando los valores que se quieren leer. Tiene que, en definitiva, ser una página dinámica. Pero es demasiado costoso de construir, dentro de la función call-back, toda la página entera; para simplificar la tarea se prevé definir la página "única" como una página estática, pero puede ser capaz de entrar en el interior de las etiquetas (etiquetas marcadoras de posición) en la posición que desea completar en el momento. Para este fin existe una función alternativa a **sendResponse()**, la función **sendDynResponse()** que se encarga de encontrar y sustituir las etiquetas y enviar la página. La sustitución pasa secuencialmente recorriendo una matriz de cadenas dispuestas sobre el momento; a continuación, la primera etiqueta encontrada se sustituye con la primera cadena de la matriz, y así sucesivamente. La etiqueta utiliza el carácter @; se debe utilizar sólo una, independientemente de la longitud de la cadena que será reemplazada. En el ejemplo que se encuentra en el listado 2, se sustituyen tres etiquetas por el mismo número de cadenas que representan los valores de tres entradas digitales. A la función **SendDynResponse()** se debe pasar una matriz de cadenas

Fig. 3 *- Página de inicio del servidor Web.*

Fig. 4

Fig. 5

Listado 4

```
#include <MwiFi.h>
 MwiFi WIFI;
setup()
{
        WIFI.begin();
        WIFI.ConnectWPAwithPSW("MioAcp","pippo");
        server=WIFI.openServerTCP(5000);
}
loop()
{
        if (!OpenCom)
                socket=WIFI.pollingAccept(server);
        if(socket<255) OpenCom=true;
        if (OpenCom)
                record=WIFI.readDataLn(socket);
        WIFI.writeDataLn(socket,".......");
}
```

y de su tamaño. Para asegurarse de que Arduino actúa en consecuencia, los siguientes comandos son lanzados desde el navegador (por ejemplo mediante botones de formulario), es necesario leer los datos enviados por la solicitud junto al nombre del recurso. La situación es diferente si la petición llega en forma de GET en lugar de POST en un formulario (los dos métodos fundamentales del protocolo HTTP). Es, en todo caso, para utilizar finalmente ese argumento pasado a la función de call-back de **getRequest()**.
En el primer caso, los datos están representados por parejas nombre-valor que identifican un parámetro. Los parámetros se añaden al nombre del recurso en un formato que codifica espacios y caracteres especiales y que podemos llamar cadena de consulta. La cadena de consulta se proporciona siempre (aunque tenga longitud cero) a la función call-back (en efecto es parte del prototipo). Entonces podemos utilizar la función **getParameter()** para recuperar el valor (siempre en forma de cadena) del parámetro con un cierto nombre (ver Listado 3).
En el segundo caso, en cambio, la cadena de consulta contendrá valores que pueden estar en formato cadena de consulta (como en general hacen las formularios) o bien en un formato cualquiera.
De todos modos, hay que tener presente que el buffer que contiene la cadena de consulta lo proporciona la librería y tiene una longitud de 64 caracteres (pero se puede definir mediante la definición de HTTPlib.h). Los datos en exceso se pierden.
Entre los ejemplos hay un servidor Web completo que le permite leer valores analógicos y valores digitales, activar y desactivar una salida digital y regular dos salidas PWM. El sketch es particularmente compacto (la mitad está constituida de las páginas html en PROGMEM) gracias a la automatización producida de las funciones **getRequest()** y **sendDynResponse().** Si, en cambio, se quiere utilizar Arduino como cliente de un servidor de aplicaciones Web (o de una CGI más sencilla), utilizaremos las funciones **sendRequest()** y **getResponse(). SendRequest()** está efectivamente constituido de dos funciones separadas en base a la modalidad que se quiere utilizar: GET o POST. Si tuviéramos que utilizar **sendRequestGET()** proporcionaríamos tanto el nombre del recurso como los parámetros en una única cadena de consulta. En el caso de que utilizáramos **sendRequestPOST()** proporcionaríamos separadamente el nombre del recurso y los datos situados en un buffer de tipo cadena terminada en null, de este modo:

```
WIFI.addParameter(query,128,"/
TestClient",NULL); WIFI.
addParameter(query,128,"A1", sa1);
:
WIFI.sendRequestGET (csocket,query);
```

O bien así:

```
sprintf(rec,"%d %d %d
%d",an1,an2,d1,d2); WIFI.sendReques-
tPOST (csocket,"/ TestClient",rec);
```

En el caso de utilizar **sendRequestGET()**, la cadena de consulta se formaría con la ayuda de la función **addParameter()**. La primera vez que se inicia la cadena de consulta sería con el nombre del recurso (valor nulo), y luego con las parejas de nombre-valor para los parámetros individuales. La función **getResponse()**, es aquella que se utiliza para recuperar la respuesta desde el servidor. Ésta puede estar también formada por numerosos datos en varios formatos: desde la página HTML, a los datos en formato XML, JSON o csv (valores separados por comas). Si los datos no pueden ser contenidos en un único buffer, es posible volver a llamar a la función **getNextResponseBuffer()** en un bucle hasta que no retorne 0. En el Listado 4 hay un ejemplo elemental de conexión para recibir y enviar registros.

El MATERIAL

Con la información contenida en este artículo, cualquier persona podría poner en práctica este proyecto. Sin embargo, hemos decidido ofrecer éste módulo WiFi ya montado y probado para dar a todos la oportunidad de dotar de conexión WiFi la propia placa de Arduino. El módulo ya montado, código FT1076M, tiene un precio de **56,00 Euros**.

Los precios incluyen IVA.
Los gastos de envío no van incluidos.
Puede hacer su pedido en:
pedidos@nuevaelectronica.com