ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Desarrollo de un servidor web con Arduino para monitorización y control de sensores y actuadores

AUTOR: JOSE CARLOS GONZÁLEZ VIDAL
DIRECTOR: DR. FERNANDO LOSILLA LÓPEZ
DICIEMBRE / 2013

Autor	Jose Carlos González Vidal
E-mail del Autor	**txecar@hotmail.com**
Director	Fernando Losilla López
E-mail del Director	fernando.losilla@upct.es
Codirector(es)	
Título del PFC	Desarrollo de un servidor web con Arduino para monitorización y control de sensores y actuadores
Descriptores	

Resumen

El trabajo que se presenta en la memoria de este proyecto fin de carrera se adentra en la implementación de un servidor web sobre Arduino el cual es capaz de controlar sensores y actuadores conectados a la plataforma Arduino.

A lo largo de esta memoria se describirán los distintos protocolos y estándares que han sido tenidos en cuenta a la hora de su realización. Se continuara introduciendo una pequeñas nociones al lenguaje específico de Arduino, para después hacer una descripción del hardware Arduino MEGA 2560 que soportara el servidor y los actuadores, también cabe destaca el XBee que permitirá las comunicaciones inalámbricas entre Arduinos. Para culminar se describirá el funcionamiento de la aplicación servidor.

Titulación	I.T.T. Especialidad Telemática
Intensificación	
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Diciembre 2013

Agradecimientos

Agradecer a mi director de proyecto Fernando por permitirme realizar el proyecto con él, que aunque teniendo un año muy apretado siguió sacando tiempo para resolver mis dudas, aun siendo fin de semana.

Me gustaría agradecer a mis compañeros y amigos por hacer de estos años de carrera sean unos años que no deseo olvidar, que por duros que fueran en ciertos momentos todos podíamos superarlos.

Por último me gustaría agradecer a mi familia por el apoyo que me han dado a lo largo de los años, siempre aconsejándome sobre que podría hacer en los momentos difíciles de mi vida. Siempre he querido hacer caso de sus consejos, pero siempre acababa por el camino largo, equivocarme y después hacer caso del consejo que ellos me habían dado. Agradecer a mi hermano por siempre estar ahí para resolver mis dudas o por lo menos intentarlo y agradecer a mis padres por tener la confianza en que podía hacerlo mejor, se que sin vosotros no habría llegado a ser el que soy hoy en día.

INDICE

1 - Introducción

La utilización de Internet en nuestro día a día ha aumentado de manera exponencial desde hace unas décadas. No hay día en que uno no se pueda levantar de la cama y no mirar el correo electrónico para comprobar si hay algún mensaje nuevo. Esto nos lleva a pensar que si podemos comunicarnos con cualquier persona que se encuentre en cualquier lugar del mundo con una conexión a Internet porque no podernos comunicarnos con nuestros aparatos electrónicos.

El aumento de las ventas de aparatos electrónicos en los últimos años, la creación de aparatos nuevos con gran frecuencia y el uso de Internet en una gran cantidad de estos dispositivos ha llevado a la sociedad de hoy en día a ansiar el control de estos aparatos o dispositivos, a los que dan uso en su vida cotidiana de un modo remoto, a través de Internet, por lo que es necesario la introducción de un pequeño hardware conectado al aparato para poder llevar a cabo esta función.

Con el uso de Arduino, una plataforma de hardware y software de código abierto (open-source), que lleva en su placa un microcontrolador y ofrece además su propio entorno de desarrollo, se pretende proporcionar Internet a ciertos aparatos o dispositivos. El hecho de tener código abierto da grandes posibilidades a los desarrolladores para crear proyectos que corran sobre ellos, ya que no es necesaria una licencia para su creación. El código abierto junto con el bajo coste de la plataforma Arduino y de los dispositivos que pueden ser conectados a ella, ha llevado a una gran proliferación de su uso. Estos usos pueden llevarse a la docencia para la creación de prototipos que controlen robots, coches teledirigidos, etc. ya que no es necesario que estos estén conectados a un ordenador para su funcionamiento.

Por estas razones se plantea la realización de este proyecto, en el que se pretende desarrollar un servidor web sobre Arduino con el que se puedan controlar sensores y actuadores conectados a una plataforma Arduino. En una primera fase del proyecto se abordara la creación de un servidor web para un único Arduino, una vez quede realizado este servidor se aborda un acceso inalámbrico a más de un Arduino.

Para el desarrollo de este proyecto se plantean una serie de tareas y retos que deben abordarse:

- Aprendizaje de uso del hardware de Arduino a través de su leguaje

- Comunicación serie Arduino – PC. En el caso que se trata deben poder enviarse datagramas IP por el puerto serie que conecta el Arduino y el PC. Se ha optado por el protocolo SLIP para permitir este tipo de comunicaciones y por la librería SerialIP para el manejo de las comunicaciones TCP/IP.

- Procesado de peticiones HTTP. Dado que HTTP es el protocolo fundamental en la Web se debe desarrollar un sistema capaz de procesar correctamente peticiones HTTP

básicas en dispositivos con capacidades limitadas. En este proyecto se procesan los métodos GET y PUT que harán las llamadas al servidor web.

- Comunicaciones inalámbricas entre dispositivos Arduino. Con el fin de que no solamente se puede acceder a los dispositivos conectados mediante una conexión serie se contempla el uso de conexiones inalámbricas entre dispositivos Arduino, en concreto mediante módulos hardware Xbee, que permite comunicación mediante el protocolo 802.15.4

2 - Estándares y Protocolos

2.1 Introducción

Como primer paso hacia la realización de este proyecto final de carrera se ha de realizar un estudio de los estándares y protocolos que se deberán implementar para el correcto funcionamiento del mismo.

2.2 HTTP

HTTP es un protocolo de transferencia de hipertexto (HyperText Transfer Protocol) que trabaja a nivel de aplicación utilizado para la trasferencia de información. HTTP define la sintaxis y semántica que es utilizada para la comunicación entre los diversos elementos de la arquitectura del software. Este protocolo es usado por arquitecturas cliente-servidor con un esquema petición-respuesta.

El protocolo sigue una especie de esquema que podría ser el siguiente. Un cliente realiza una petición que estará formada por un método, una URI, y una versión del protocolo en la primera línea, a continuación se mandaran las cabeceras de la petición y por último un posible dato a introducir. El servidor contesta con una línea de estado que incluye la línea del protocolo y un código de éxito o error, en las siguientes líneas enviara las cabeceras de la petición y por último y si fuera necesario algún tipo de datos.

HTTP es un protocolo sin estado, lo que significa que no guarda ninguna información de sus conexiones anteriores. Para dar una solución a este problema se crearon las cookies, las cuales son información que el servidor guarda en los clientes. Las cookies abren un abanico de posibilidades como la creación de sesiones, la compra por Internet.

Petición

HTTP define 8 métodos: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS y CONNECT.

GET

Pide una representación del recurso especificado. Por seguridad no debería ser usado por aplicaciones que causen efectos ya que transmite información a través de la URI agregando parámetros a la URL.

HEAD

El método HEAD es igual que el método GET, salvo que el servidor solo tiene que devolver las cabeceras. Estas cabeceras son las mismas como si se utilizara el método GET.

POST

El método POST es utilizado para realizar peticiones en las que el servidor destino acepta el contenido de la petición como un nuevo subordinado del recurso pedido. Otro de sus usos es el de cambiar el estado del servidor.

PUT

El método PUT sube, carga o realiza un upload de un recurso, es la manera más eficiente de subir archivos a un servidor.

DELETE

El método DELETE es utilizado para que el servidor borre un recurso indicado por la URI de la petición.

TRACE

El método TRACE es utilizado para comprobar la existencia del receptor del mensaje y usar la información para realizar un diagnostico.

OPTIONS

El método OPTIONS devuelve los métodos HTTP que el servidor soporta para un URL específico. Esto puede ser utilizado para comprobar la funcionalidad de un servidor web mediante petición.

CONNECT

El método CONNECT es utilizado para saber si un proxy nos da acceso a un host bajo condiciones especiales.

2.3 SERVIDOR HTTP

Un servidor web es un programa que atiende y responde las diferentes peticiones de los usuarios, proporcionando los recursos solicitados haciendo uso del protocolo HTTP o HTTPS. El servidor web básico es el que acostumbra a realizar las siguientes acciones en un bucle

1. Espera peticiones en el puerto TCP indicado
2. Recibe una petición.
3. Busca el recurso.
4. Envía el recurso utilizando la misma conexión por la que recibió petición.
5. Vuelve al segundo punto.

Un servidor web que siga este esquema cumplirá todos los requisitos básicos de los servidores HTTP.

A partir del anterior esquema se han diseñado y desarrollado todos los servidores de HTTP que existen, variando sólo el tipo de peticiones que pueden atender, en función de que sean o no sean multi-proceso o multi-thread, etc.

2.4 SLIP

El protocolo SLIP es un estándar de transmisión de datagramas IP para líneas serie. Está diseñado para trabajar a través de puerto serie y conexión de modem.

SLIP ha sido sustituido en PC por el protocolo PPP (Point-to-Point Protocol) que posee mejores características, pero esto no ha llevado a que SLIP cayera en desuso. SLIP sigue siendo utilizado por microcontroladores debido a que su encapsulación para paquetes IP utiliza cabeceras de tamaño reducido.

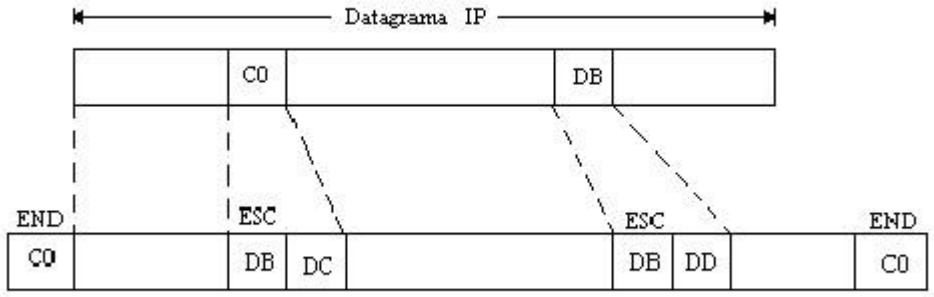
SLIP es un protocolo con una implementación sencilla y debido a ello no proporciona ni direccionamiento, ni identificación de tipo de paquete, ni detección/corrección de errores ni mecanismos de compresión. Se puede hacer uso de SLIP en líneas con velocidades comprendidas entre 1200bps y 19.2kbps.

Formato de los paquetes

SLIP modifica cada datagrama IP añadiéndole al final un carácter especial C0 que permite distinguir entre diferentes tramas. Para evitar que se produzca algún problema con el ruido de línea se manda otro al principio de cada datagrama.

Si el carácter C0 se presenta en el contenido del datagrama, se utiliza la secuencia de dos bytes DB, DC. El carácter DB es el carácter de escape de SLIP (distinto al valor ASCII de ESC –1B--).

Si en el contenido se presenta el carácter de escape; se reemplaza por la secuencia DB, DD.



Tipo

- **SLIP dinamico**

El servidor proveedor de Internet identifica al ordenador proporcionándole una dirección IP. La dirrecion es asignada dinámicamente de entre un conjunto de direcciones y solo dura el tiempo de la conexión.

- **SLIP estatico**

Es servidor proveedor de Internet asigna una dirección fija al ordenador para su uso en todas las sesiones.

Inconvenientes

- **Direccionamiento**

Los dos hosts necesitan conocer previamente la dirección IP del otro extremo.

- **Identificación de tipo**

SLIP al no poseer una identificación de tipo provoca que no se pueda hacer uso de otros protocolos simultáneamente.

- **Contraseñas**

Transmite las contraseñas de autentificación como texto, lo que puede provocar un grave problema de seguridad.

Comparativa SLIP vs PPP

Ambos cumplen la misma función aunque entre ellos mantienen ciertas diferencias

SLIP	PPP
Fácil de implementar	Compleja implementación
Introduce muy pocos bytes de overhead	Incluye más datos de overhead
Utilizado por muchas aplicaciones	Su uso está en crecimiento
Efectúa comprensión de overhead (CSLIP)	Puede configurarse para que use comprensión de overhead
No es estándar	Estándar
No efectúa detección ni corrección de errores	Suma de verificación (CRC) en cada marco.
Solo reconoce IP	Múltiples protocolos
Debe conocerse la IP de cada extremo	Permite la asignación dinámica de IP
No confiable	Transmisión confiable opcional
Estático	Configurable a través de LCP

2.5 802.15.4

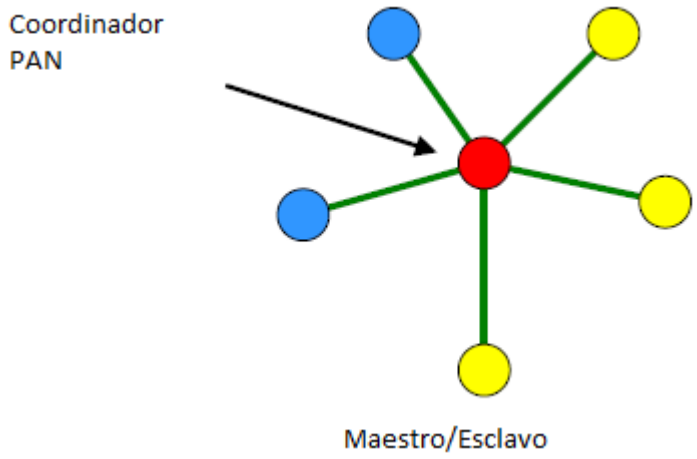
2.5.1 Introducción

IEEE 802.15.4 es un estándar que define el nivel físico y el control de acceso al medio de redes inalámbricas de área personal con tasas bajas de transmisión de datos. Ha sido desarrollado por el Grupo de Tareas 802.15.4 dentro del IEEE y define la capa física (PHY) y el control de acceso al medio (MAC) especificaciones de la capa para las redes de área personal inalámbricas baja tasa de transmisión (LR WPAN). Este tipo de redes se limitan por lo general a un espacio operativo personal de unas decenas de metros y hace participar a escasas infraestructuras. La norma prevé baja complejidad, bajo consumo de energía, conectividad inalámbrica de baja velocidad de datos entre una amplia gama de dispositivos de bajo costo. Entre otros, las redes inalámbricas de sensores parecen ser un escenario de aplicación adecuada para redes 802.15.4.

2.5.2 Las topologías de red

Topología en estrella

En esta topología los dispositivos se comunican con un único coordinador central PAN. Mientras RFDs actúan como sólo los puntos finales de comunicación, un coordinador PAN lo hace por las vías adyacentes. Diferentes redes en estrella operan independientemente de otras. Esto se logra mediante el uso de un identificador de PAN que es único dentro del alcance de la radio. Después de que el coordinador del PAN ha elegido un identificador de PAN, puede permitir que otros dispositivos, tanto FFDs y RFDs, puedan unirse a la red.



Topología punto a punto

La topología de punto a punto también tiene un coordinador PAN, pero, además, los dispositivos pueden comunicarse directamente entre sí. Esto permite topologías más complejas, tales como una topología de malla o cluster-tree.

2.5.3 Transferencia de datos en redes beacon-enabled y non-beacon

Hay dos modos de funcionamiento en una red 802.15.4, modo beacon-enabled y modo non-beacon.

En el modo beacon-enabled, se utiliza una estructura de supertrama opcional. Es definido y limitado por los beacons emitidos por el coordinador. Los beacons se utilizan para sincronizar disponsitivos, identificar el PAN y definir la estructura de la supertrama. La supertrama se divide en 16 ranuras del mismo tamaño. Un beacon es emitido en el primer slot de cada supertrama.

Los dispositivos que deseen comunicarse durantes el periodo de contención de acceso (CAP) entre dos beacons compiten con otros dispositivos usando el mecanismo de ranurado CSMA-CA (Carrier Sense Multiple Access - Collision Avoidance). La trama se puede dividir en dos partes una activa y otra inactiva. Durante la parte inactiva el coordinador no interactúa con el PAN y puede entrar en un modo de bajo consumo. El coordinador podrá dedicar partes de la supertrama activa a slots de tiempo garantizados (GTSs) para aplicaciones de baja latencia o garantizar ancho de banda.

Hay una diferencia entre las transferencias de datos desde un dispositivo a un coordinador y vice-versa. Para la transferencia de datos desde un dispositivo a un coordinador, el dispositivo usa slots CSMA-CS para transmitir la trama de datos. Para la transferencia de datos desde un dispositivo, el coordinador indica en un beacon que los datos son de espera para el dispositivo. El dispositivo escucha periódicamente beacons. Si la transferencia de datos está pendiente de él, este transmite un comando MAC solicitando la transferencia de datos. Este comando MAC se transmite utilizando ranurado CSMA-CA. A la recepción de la orden MAC el coordinador podrá decidir si se utiliza el modo non-beacon. Entonces no hay estructura de supertrama y hay dispositivos de uso no ranurado CSMA-CA. Para la transferencia de datos coordinador-dispositivo, el dispositivo tiene que solicitar los datos de transferencia usando un comando MAC. Si hay pendientes datos para el dispositivo, estos son transmitidos en la trama de datos. De lo contrario, una trama de longitud cero con carga útil se transmite, indicando que no hay datos pendientes para el dispositivo.

2.5.4 Robustez

La robustez en las redes 802.15.4 se logra mediante el uso de un marco opcional acknowledgments, mecanismos CSMA-CA y verificación de datos.

2.5.5 Seguridad

Varios servicios de seguridad tales como el mantenimiento de una lista de control de acceso (ACL) y utilización de criptografía de clave simétrica para proteger las tramas transmitidas que son especificadas en el estándar forman parte de la seguridad del estándar.

Usando estos servicios, los dispositivos pueden funcionar de los siguientes modos: modo no seguro, ACL y modo seguro. En el modo no seguro los servicios de seguridad no son utilizados. En el modo ACL los dispositivos mantiene listas ACL de los dispositivos de los que están dispuestos a recibir tramas. Los dispositivos que funcionan en modo seguro usan criptografía junto con ACL.

2.6 uIP

La pila uIP TCP/IP está diseñada para hacer posible la comunicación usando el protocolo TCP/IP, incluso en pequeños microcontroladores de 8 bits. El tamaño del código tiene del orden de unos pocos kilobytes y uso de la RAM puede ser configurado para ser tan bajo como unos pocos cientos de bytes.

2.7 Protosocket

La biblioteca protosocket proporciona una interfaz a la pila uIP que es parecida a la tradicional de socket BSD y su funcionamiento solo es posible en conexiones TCP. Otra de sus características es que los programas escritos con la librería protosocket se ejecutan de forma secuencial y no tienen que ser ejecutados como máquinas de estados explícitas.

La biblioteca protosocket utiliza protothreads para proporcionar un flujo de control secuencial. Esto hace a los protosockets ligeros en lo que a términos de memoria se refiere, aunque también hace que adquieran las limitaciones de los prothreads.

3 - Tutorial de Arduino

3.1 Introducción del tutorial

El lenguaje de programación de Arduino proviene del lenguaje C y posee la siguiente estructura básica de programa o sketch.

La estructura básica se compone de dos partes setup y loop.

```
void setup()
{
//inicializaciones;
}
void loop()
{
//ejecución;
}
```

Setup() es la parte encargada de recoger la configuración y loop() es la parte que contiene el programa que se ejecutará cíclicamente. Las dos funciones son necesarias para el funcionamiento del programa.

La función “setup” debe contener la declaración de las variables. Es la primera función que se ejecuta en el programa, se ejecuta solo una vez y se utiliza para configurar comunicaciones serie y otras. Debe ser incluido en los programas aunque no haya declaraciones que ejecutar.

```
{
pinMode(LEDpin, OUTPUT); // configura 'LEDpin' como salida
}
```

La función “loop” se ejecutara constantemente de forma cíclica y es la que realiza la mayor parte del trabajo. Posibilita que el programa responda constantemente a los eventos que se produzcan en la placa.

```
void loop()
{
digitalWrite(LEDpin, HIGH); // pone en uno ´LEDpin´
delay(1000); // espera un segundo
digitalWrite(LEDpin, LOW); // pone en cero ‘LEDpin’
delay(1000); //
}
```

3.2 Constantes

3.2.1 HIGH/LOW

Las constantes high y low son utilizadas para la escritura y lectura digital de las patillas. High se define como ON y por lo tanto LOW como OFF.

digitalWrite(13, HIGH); // activa la salida 13 a nivel alto

3.2.2 INPUT/OUTPUT

Las constantes input y output son utilizadas para definir el funcionamiento de los pines si son de entrada o salida. Mediante la instrucción pinMode, que será ejecutada dentro de setup() estableceros que pines serán utilizados como entrada o salida.

pinMode(13, OUTPUT); // designamos que el PIN 13 como salida

3.3Entradas y Salidas Digitales

3.3.1 pinMode(pin,mode)

Como ya se comento anteriormente esta instrucción es ejecutada en la parte de configuración y configura los pines como entrada o salida.

Los terminales Arduino, están configurados como entradas por defecto, por lo que no es necesario hacer un pinMode, si van a a trabajar como entrada. Los pines de entrada se encuentran en estado de alta impedancia.

Los pines tienen una resistencia de 20KΩ a las que se puede acceder como muestran las siguientes líneas de código

pinMode(LEDpin, INPUT); // configura el ‘pin’ como entrada
digitalWrite(LEDpin, HIGH);

Los pines que son configurados como OUTPUT se encuentran a un nivel de baja impedancia y podrán proporcionar 40 mA(miliamperios) a otros circuitos y dispositivos. Con ello se puede llegar a alimentar LED por ejemplo.

Se debe tener cuidado a la hora de conectar dispositivos o circuitos externos, ya que si por error provocamos un cortocircuito podría darse el caso de dañar el chip Atmega. Esto lleva a hacer la recomendación de usar una resistencia externa de 470 Ω o de 1000 Ω.

3.3.2 digitalRead(pin)

Lee el valor al que se encuentra el pin y da una respuesta HIGH o LOW. El dato que le pasamos es un valor entre 0 y 13.

valor = digitalRead(LEDPin);

3.3.3 digitalWrite(pin, value)

Escribe en el pin indicado (que el parte de de configuración debió ser configurado como OUTPUT) el valor de la variable value que será HIGH o LOW.

digitalWrite(LEDpin, HIGH); //pone 'LEDpin' a alto

3.4 E/S analógicas

3.4.1 analogRead(pin)

Lee el valor de un pin configurado como entrada analógica. Los pines que encontramos como entradas analógicas son los comprendidos entre 0 y 5. Estos pines no necesitan ser declarados como entradas ya que están configurados así por defecto. El valor que nos puede ser devuelto se encuentra entre 0 y 1023.

valor = analogRead(pin); // asigna a valor que lee de la entrada

3.5 Control del tiempo

3.5.1 delay(ms)

La función delay() provoca una interrupción en el programa por un tiempo determinado que es introducido como parámetro. Este tiempo debe ser introducido en milisegundos, por lo que 5000 es igual a 5 seg.

delay(5000); // espera 5 segundos

3.5.2 millis()

Devuelve el tiempo que ha transcurrido desde que se inicio el programa en Arduino hasta el momento actual. Este valor debe ser reseteado si quiere utilizarse debido que tras 9 horas el numero es desbordado.

valor = millis(); // valor recoge el número de milisegundos

3.6 Matemáticas

3.6.1 min(x, y)

Devuelve el número de menor tamaño entre dos valores introducidos. Esta instrucción facilita por ejemplo la ordenación de arrays.

valor = min(valor, 100);

3.6.2 max(x, y)

Esta instrucción devuelve el número de mayor tamaño de entre ambos valores introducidos.

valor = max(valor, 100);

3.7 Aleatorios

3.7.1 randomSeed(seed)

Crea un valor aleatorio a partir de un valor o semilla. Sin embargo se plantea el problema de que Arduino no puede crear valores aleatorios de manera real, por lo que una manera de crear un aleatorio seria introduciendo el valor de la instrucción millis() como semilla.

randomSeed(valor);

3.7.2 random(max)

Genera un número aleatorio cuyo valor máximo está determinado por un valor que es introducido como parámetro en la función. Debido a que Arduino no es capaz de generar verdaderos números aleatorios antes de hacer uso de esta instrucción deberemos hacer una llamada a la función randomSeed(seed).

randomSeed(millis());

```
randNumber = random(255);
delay(500);
```

3.7.3 random(min, max)

Esta función actúa igual que random(max) con la única singularidad de que también se debe introducir el valor mínimo que podrá alcanzar el numero aleatorio.

```
randomSeed(millis());
randNumber = random(100,255);
delay(500);
```

3.8 Comunicación serie

3.8.1 Serial.begin(rate)

Establece la comunicación entre el Arduino y el ordenador a través del puerto serie estableciendo la velocidad a la que serán transmitidos los datos, esta unidad vendrá definida en baudios.

```
void setup()
{
Serial.begin(9600); // abre el Puerto serie
}
```

3.8.2 Serial.println(data)

Esta función envía por el puerto serie los datos deseados junto con un salto de línea. Estos datos serán mostrados en el Monitor Serie del software.

```
Serial.println(“Hola”);
```

3.8.3 Serial.print(data, data type)

Al igual que Serial.println(data) envía por el puerto serie los datos deseados, sin embargo en esta ocasión no se introducirá un salto de línea. En esta función podemos introducir el tipo de carácter que vamos a enviar, si se deja en blanco será considerado como ASCII.

```
Serial.print(b);
```

```
int b = 79;
Serial.print(b);
Serial.print(b, HEX);
Serial.print(b, OCT);
Serial.print(b, BIN);
Serial.print(b, BYTE);
```

3.8.4 Serial.avaible()

Esta función devuelve el número de bytes provenientes del bus serie que pueden ser leídos. Se debe tener cuidado con las limitaciones del hadware para los bytes que serán recibidos por el bus serie.

3.8.5 Serial.Read()

Lee un byte proveniente del puerto serie, si no hubiera ningún byte que leer la función devolverá -1.

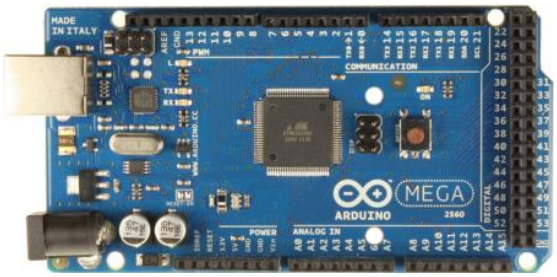
```
int incomingByte = 0;
void loop() {
incomingByte = Serial.read();
Serial.print("I received: ");
Serial.println(incomingByte, DEC);
}
```

4 - Entorno de Trabajo

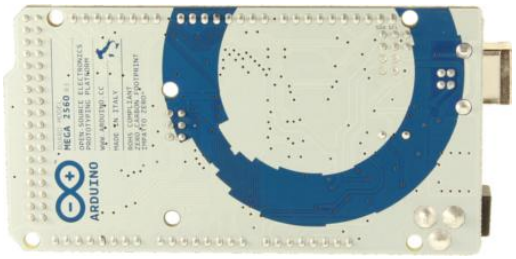
4.1 Introducción

En este capítulo se pretenden describir las principales herramientas hadware y software sobre las que se ha llevado a cabo este proyecto fin de carrera, de una manera detallada.

4.2 Arduino Mega 2560

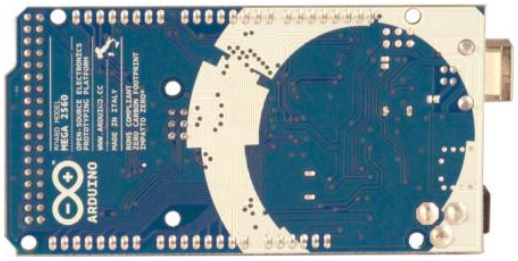


Arduino Mega 2560 R3 delantero



Arduino Mega2560 R3

Arduino Mega 2560 Frontal



Arduino Mega 2560

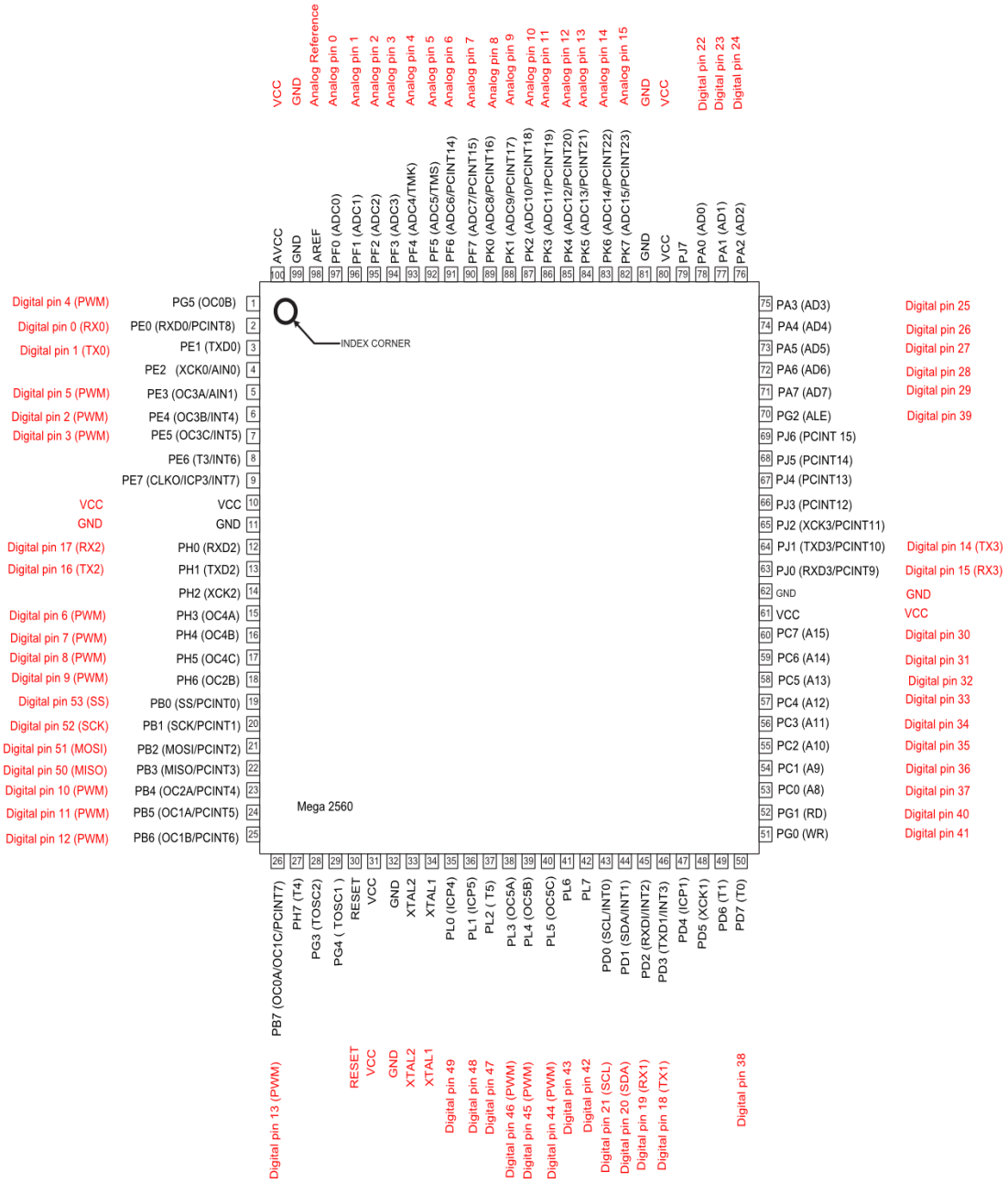
4.2.1 Vision General

El Arduino Mega 2560 es una placa mmicrocontrolador basada en el microprocesador Atmega2560. Tiene 54 entradas/salidas digitales (de las cuales 15 pueden utilizarse para salidas PWM), 16 entradas analógicas, 4 UARTs (puertos serie por hardware), un oscilador de 16MHz, una conexión USB, entrada de corriente, conector ICSP y botón de reset. Contiene todo lo necesario para hacer funcionar el microcontrolador, simplemente conectarlo a un ordenador con un cable USB, o alimentarlo con un adaptador de corriente AC a DC para empezar. Mega es compatible con la mayoría de los shield diseñados para Arduino Duemilanove o Diecimila.

Mega 2560 es una actualización de la placa Arduino Mega , al que sustituye.

Mega2560 difiere de todas las placas anteriores ya que no utiliza el chip controlador de USB a serial FTDI. En su lugar, ofrece el ATmega16U2 programado como convertidor USB a serie. Esto implica que se trabaje con /dev/ACM.

4.2.2 Mapeado de Pin



4.2.3 Resumen

Microcontrolador	ATmega2560
Voltaje de funcionamiento	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (limite)	6-20V
Pines E/S digitales	54 (15 proporcionan salida PWM)
Pines de entrada analógica	16
Intensidad por pin	40 mA
Intensidad en pin 3.3V	50 mA
Memoria Flash	256 KB de las cuales 8 KB las usa el gestor de arranque(bootloader)
SRAM	8 KB
EEPROM	4 KB
Velocidad de reloj	16 MHz

4.2.4 Alimentación

El Arduino Mega puede ser alimentado por la conexión USB o por una fuente de alimentación externa. El origen de la alimentación es seleccionado automáticamente.

Las fuentes de alimentación externas (no-USB) pueden ser un transformador o una batería. El transformador se puede conectar usando un conector macho de 2.1mm con centro positivo en el conector hembra de la placa. Los cables de la batería pueden conectarse a los pines GND y Vin en los conectores de alimentación (POWER).

La placa puede trabajar con una alimentación externa de entre 6 a 20 voltios. Si el voltaje suministrado es inferior a 7V el pin de 5V puede proporcionar menos de 5 Voltios y la placa puede volverse inestable, si se usan mas de 12V los reguladores de voltaje se pueden sobrecalentar y dañar la placa. El rango recomendado es de 7 a 12 voltios.

Los pines de alimentación son los siguientes:

- **VIN.** La entrada de voltaje a la placa Arduino cuando se está usando una fuente externa de alimentación (en contraposición a los 5 voltios de la conexión USB). Se puede proporcionar voltaje a través de este pin, o, si se está alimentado a través de la conexión de 2.1mm, acceder a ella a través de este pin.

- **5V.** La fuente de voltaje estabilizado usado para alimentar el microcontrolador y otros componentes de la placa. Esta puede provenir de VIN a través de un regulador

integrado en la placa, o proporcionada directamente por el USB u otra fuente estabilizada de 5V.

- **3V3.** Una fuente de voltaje a 3.3 voltios generada en el chip FTDI integrado en la placa. La corriente máxima soportada 50mA.
- **GND.** Pines de toma de tierra.
- **IOREF.** Este pin proporciona la referencia de tensión con la que opera el microcontrolador. Un shield configurado puede leer el voltaje pin IOREF y seleccionar la fuente de alimentación adecuada o habilitar traductores tensión en las salidas para trabajar con los 5V o 3.3V.

4.2.5 Memoria

El ATmega2560 tiene 256KB de memoria flash para almacenar código (8KB son usados para el arranque del sistema (bootloader)).El ATmega2560 tiene 8 KB de memoria SRAM y 4KB de EEPROM.

4.2.6 Entradas y Salidas

Cada uno de los 54 pines digitales en el Mega pueden utilizarse como entradas o salidas usando las funciones pinMode(), digitalWrite(), y digitalRead() . Las E/S operan a 5 voltios. Cada pin puede proporcionar o recibir una intensidad máxima de 40mA y tiene una resistencia interna (desconectada por defecto) de 20-50kOhms. Además, algunos pines tienen funciones especializadas:

- **Serie: 0 (RX) y 1 (TX), Serie 1: 19 (RX) y 18 (TX); Serie 2: 17 (RX) y 16 (TX); Serie 3: 15 (RX) y 14 (TX).** Usado para recibir (RX) transmitir (TX) datos a través de puerto serie TTL. Los pines Serie: 0 (RX) y 1 (TX) están conectados a los pines correspondientes del chip ATmega16U2.
- **Interrupciones Externas: 2 (interrupción 0), 3 (interrupción 1), 18 (interrupción 5), 19 (interrupción 4), 20 (interrupción 3), y 21 (interrupción 2).** Estos pines se pueden configurar para lanzar una interrupción en un valor LOW (0V), en flancos de subida o bajada (cambio de LOW a HIGH (5V) o viceversa), o en cambios de valor.
- **PWM: de 2 a 13 y 44 a 46.** Proporciona una salida PWM (Pulse Wave Modulation, modulación de onda por pulsos) de 8 bits de resolución (valores de 0 a 255) a través de la función analogWrite().

- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** Estos pines soportan comunicación SPI utilizando la biblioteca de SPI . Los pines SPI también se desglosan en la cabecera ICSP, que es físicamente compatible con el Uno, Duemilanove y Diecimila.

- **LED: 13.** Hay un LED integrado en la placa conectado al pin digital 13, cuando este pin tiene un valor HIGH(5V) el LED se enciende y cuando este tiene un valor LOW(0V) este se apaga.

- **TWI: 20 (SDA) y 21 (SCL)** Apoyar la comunicación TWI utilizando la librería Wire.

El Mega2560 tiene 16 entradas analógicas, cada una de las cuales proporcionan una resolución de 10 bits (1.024 valores diferentes). Por defecto se miden desde el tierra a 5 voltios, aunque es posible cambiar la cota superior de su rango utilizando el pin AREF y función analogReference ().

Hay unos otros pines en la placa:

- **AREF.** Voltaje de referencia para las entradas analógicas. Usado por analogReference().

- **Reset.** Suministrar un valor LOW (0V) para reiniciar el microcontrolador. Típicamente usado para añadir un botón de reset a los shields que no dejan acceso a este botón en la placa.

4.2.7 Comunicación

El Arduino Mega2560 tiene una serie de facilidades para la comunicación con un ordenador, otro Arduino, u otros microcontroladores. El ATmega2560 proporciona cuatro UART hardware para TTL (5V) de comunicación serie. Un ATmega16U2 canaliza a uno de ellos sobre el USB y proporciona un puerto com virtual para software al equipo (máquinas de Windows tendrá un archivo .inf, pero las máquinas OSX y Linux reconocerán la placa automáticamente). El software de Arduino incluye un monitor serie que permite enviar datos desde y hacia la placa. Los LEDs RX y TX de la placa parpadearán cuando se están transmitiendo datos a través de ATmega8U2/ATmega16U2 chip y la conexión USB al ordenador (pero no para la comunicación serial en los pines 0 y 1).

La biblioteca SoftwareSerial permite la comunicación en serie en cualquiera de los pines digitales del Mega2560.

El ATmega2560 también soporta TWI y la comunicación SPI.

4.2.8 Programación

El ATmega2560 en el Arduino Mega viene precargado con un gestor de arranque (bootloader) que permite cargar nuevo código sin necesidad de un programador por hardware externo. Se comunica utilizando el protocolo STK500 original.

También se puede saltar el gestor de arranque y programar directamente el microcontrolador a través del puerto ISCP (In Circuit Serial Programming).

4.2.9 Reinicio Automático por Software

En vez de necesitar reiniciar presionando físicamente el botón de reset antes de cargar, el Arduino Mega2560 está diseñado de manera que es posible reiniciar por software desde el ordenador al que esté conectado. Una de las líneas de control de flujo (DTR) del ATmega8U2 está conectada a la línea de reinicio del ATmega2560 a través de un condensador de 100 nanofaradios. Cuando la línea se pone a LOW (0V), la línea de reinicio también se pone a LOW el tiempo suficiente para reiniciar el chip. El software de Arduino utiliza esta característica para permitir cargar los sketches con solo apretar un botón del entorno. Dado que el gestor de arranque tiene un lapso de tiempo para ello, la activación del DTR y la carga del sketch se coordinan perfectamente.

Esta configuración tiene otras implicaciones. Cuando el Mega2560 se conecta a un ordenador con Mac OS X o Linux, este reinicia la placa cada vez que se realiza una conexión desde el software (vía USB). El medio segundo aproximadamente posterior, el gestor de arranque se ejecutará. A pesar de estar programado para ignorar datos mal formateados (ej. cualquier cosa que la carga de un programa nuevo) intercepta los primeros bytes que se envían a la placa justo después de que se abra la conexión. Si un sketch que se está ejecutando en la placa recibe algún tipo de configuración inicial u otro tipo de información al inicio del programa, asegúrese de que el software con el cual se comunica espera un segundo después de abrir la conexión antes de enviar los datos.

El Mega2560 contiene una pista que puede ser cortada para deshabilitar el auto-reset. Las terminaciones a cada lado pueden ser soldadas entre ellas para rehabilitarlo. Están etiquetadas con "RESET-EN". También se puede deshabilitar el auto-reset conectando una resistencia de 110 Ω desde el pin 5V al pin de reset.

4.2.10 Características Físicas y Compatibilidad de Shields

La longitud y amplitud máxima de la placa Mega2560 es de 4 y 2.1 pulgadas respectivamente, con el conector USB y la conexión de alimentación sobresaliendo de estas dimensiones. Tres agujeros para fijación con tornillos permiten colocar la placa en superficies y cajas. Se debe tener en cuenta que la distancia entre los pines digitales 7 y 8 es 160 mil (0,16").

El Mega está diseñado para ser compatible con la mayoría de shields diseñados para el Uno, Diecimila o Duemilanove. Pines digitales 0 a 13 (y los pines AREF y GND adyacentes), las entradas analógicas de 0 a 5, los conectores de alimentación, y los conectores ICSP están todos ubicados en posiciones equivalentes. Además la UART principal (puerto serie) se encuentra en los mismos pines (0 y 1), al igual que las interrupciones externas 0 y 1 (pines 2 y 3, respectivamente). SPI está disponible a través de la cabecera ICSP tanto en el Mega2560 y Duemilanove / Diecimila.

4.3 XBee AP1-001

El módulo XBee 802.15.4 OEM RF es una solución incrustada que proporciona conectividad end-point inalámbrica a los dispositivos. Los módulos utilizan el protocolo de red IEEE 802.15.4 para un rápido punto-a-multipunto o PPP en redes. Están diseñados para aplicaciones de alto rendimiento que requieren baja latencia y tiempo de comunicación predecibles.

Los módulos XBee están preparados para trabajar a baja potencia y aplicaciones de bajo costo. Parte de la familia de productos de RF XBee son fáciles de usar, tienen el mismo

tamaño, y son totalmente interoperables con otros productos XBee utilizando la misma tecnología.

Características principales:

- Alcance en espacios abiertos hasta la línea de visión de 90 m
- Alcance interior de hasta 30m
- La velocidad de datos de hasta 250 Kbps
- 2,4 GHz banda de frecuencia (aceptada en todo el mundo)

4.4 Netcat

Netcat es una herramienta de red potente y sensible que permite a través de intérprete de comandos y con una sintaxis sencilla abrir puertos TCP/UDP en un HOST (quedando netcat a la escucha), asociar una shell a un puerto en concreto y forzar conexiones UDP/TCP.

Entre sus múltiples aplicaciones, es frecuente la depuración de aplicaciones de red. También es utilizada a menudo para abrir puertas traseras en un sistema.

Introducción

La forma más básica de operar de netcat consiste en:

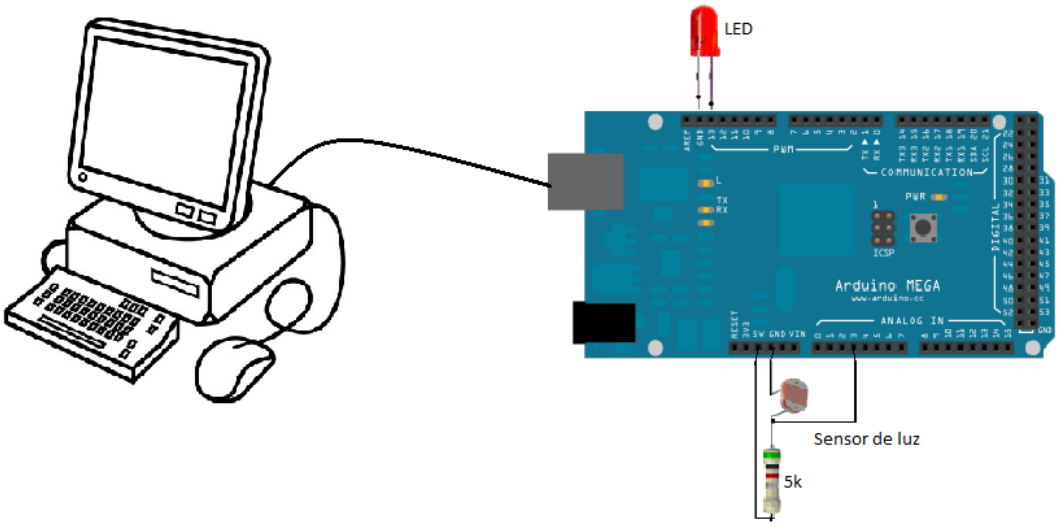
1. Crea un socket con el destino indicado si es cliente, o en el puerto indicado, si es servidor
2. Una vez conectado, envía por el socket todo lo que llegue en su entrada estándar y envía a su salida estándar todo lo que llegue por el socket

5 – Sistema Desarrollado

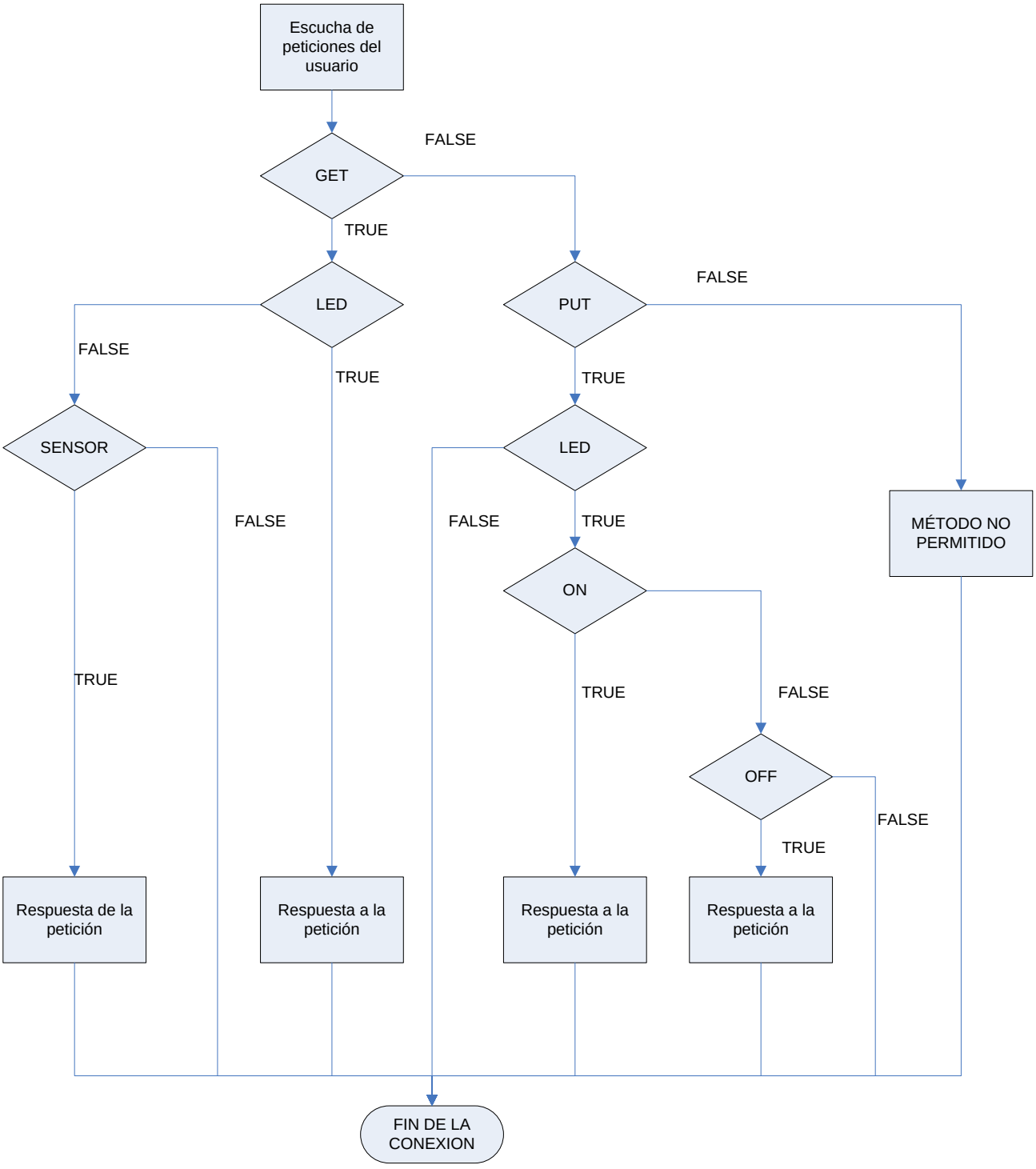
Para el correcto funcionamiento del servidor web que es necesario disponer de unos elementos básicos que son un PC con conexión a Internet y un Arduino que llevara conectados ciertos actuadores y sensores.

El funcionamiento del servidor web implementado sobre Arduino explicado de una forma sencilla y sin entrar en detalles seguirá los siguientes pasos:

1. Establecimiento de una conexión Arduino con el PC mediante el protocolo SLIP
2. Recepción de las peticiones del cliente
3. Tratamiento de las peticiones del cliente
4. Respuesta a las peticiones del cliente ya sae haciendo uso de un actuador o devolviendo el valor de un sensor.



He aquí un diagrama de flujo que muestra como se dará respuesta a las peticiones de los clientes:



Pasemos ahora a una explicación más detallada del funcionamiento del código.

Para comenzar a programar el Arduino conectado mediante el protocolo SLIP se han tenido en cuenta las dos partes principales de la programación en Arduino la parte setup() en la que se configuraran ciertos parámetros para la configuración de la placa y la parte loop() que ejecutara la parte principal del programa.

Comencemos por explicar la parte setup() del código ya que sin ella el código no podría comenzar a funcionar.

Se ha configurado la velocidad del bus serie a 115200, que transmitirá la información por tx0 y la recibirá por rx0, el hecho de emitir por estos pines lo tiene asignado la placa por hardware.

Se hace uso de de la función SerialIP.use_device() para asignar la ruta por la que los datos de entrada y salida serán transmitidos y tratados, gracias a SerialIP. Se ha decidido manejar los eventos uIP de manera autónoma por lo que se le indica al Arduino con SerialIP.set_uip_callback() y como parámetro la función que tratara los eventos.

Configuramos la IP y la máscara de la red ya que son necesarias para establecer la conexión SLIP con el PC y establecemos el puerto por el que queremos que fluyan los datos.

```
void setup() {

  // Velocidad de nuestra conexion serie.
  Serial.begin(115200);

  // Llamada a SerialIP con el puerto serie a utilizar
  SerialIP.use_device(Serial);

  // Manejaremos los eventos uIP nosotros mismos.
  SerialIP.set_uip_callback(uip_callback);

  // Establecemos la direccion de la IP que vamos a utilizar
  IP_ADDR myIP = {192,168,5,2};
  IP_ADDR subnet = {255,255,255,0};
  SerialIP.begin(myIP, subnet);

  SerialIP.listen(80);

  pinMode(ledPin, OUTPUT);

}
```

En la parte de loop() se usará solo una función que será la encargada de comprobar el puerto serie y procesar los datos, esta función es SerialIP.tick(). Esta función debe ser llamada de forma regular y puede ser la única función llamada en loop() como es el caso.

A continuación se comenzara con el análisis de los eventos uIP por lo que se comprobara que se ha establecido una conexión uIP antes de realizar cualquier otra función o asignación.

Si se ha establecido la conexión, inicializamos el protosocket y lo arrancamos; con esto ya estamos preparados para comenzar a recibir mensajes de los posibles clientes. El servidor web del Arduino se pondrá en espera a la recepción de peticiones por parte de los clientes, estas peticiones han tenido que ser reducidas en tamaño debido a las limitaciones del hardware. Por lo que se ha decidido realizar las peticiones al servidor web a través de netcat ya que con el podemos limitar el tamaño de la petición.

Solo queda en estos momentos analizar la petición del usuario y darle una respuesta en consecuencia, para ello utilizaremos las siguientes funciones:

Esta primera función lee los datos a través de la conexión SLIP hasta el salto de línea.

```
PSOCK_READTO(&s->p, '\n');
```

Esta envía los datos del buffer y añade un terminal nulo al final.

```
PSOCK_SEND_STR(&s->p, buffer);
```

Por último queda desconectar a este cliente del servidor y cerrar el protosocket que se había creado para mantenerla, y ya estará listo para recibir otras conexiones.

Mediante del uso del sniffer de la aplicación wireshark, se han realizado unas capturas de tráfico para comprobar el correcto funcionamiento del servidor web. A continuación se procederá al análisis de una captura GET.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.5.1	192.168.5.2	TCP	60	54337 > http [SYN] Seq=0 Win=2560 Len=0 MSS=256 SACK_PERM=1 TSval=5346786 TSecr=0 W
2	0.023543	192.168.5.2	192.168.5.1	TCP	44	http > 54337 [SYN, ACK] Seq=4294967288 Ack=1 Win=88 Len=0 MSS=88
3	0.023574	192.168.5.1	192.168.5.2	TCP	40	54337 > http [ACK] Seq=1 Ack=4294967289 Win=2560 Len=0
4	0.025085	192.168.5.1	192.168.5.2	HTTP	60	GET /led HTTP/1.1 Continuation or non-HTTP traffic
5	0.028530	192.168.5.1	192.168.5.2	TCP	40	54337 > http [FIN, ACK] Seq=21 Ack=4294967289 Win=2560 Len=0
6	0.039123	192.168.5.2	192.168.5.1	HTTP	48	Continuation or non-HTTP traffic
7	0.039159	192.168.5.1	192.168.5.2	TCP	40	54337 > http [ACK] Seq=22 Ack=1 Win=2560 Len=0
8	0.051906	192.168.5.2	192.168.5.1	HTTP	64	Continuation or non-HTTP traffic
9	0.051928	192.168.5.1	192.168.5.2	TCP	40	54337 > http [ACK] Seq=22 Ack=25 Win=2560 Len=0
10	0.066373	192.168.5.2	192.168.5.1	HTTP	41	Continuation or non-HTTP traffic
11	0.066415	192.168.5.1	192.168.5.2	TCP	40	54337 > http [ACK] Seq=22 Ack=26 Win=2560 Len=0
12	0.082976	192.168.5.2	192.168.5.1	HTTP	53	Continuation or non-HTTP traffic
13	0.083181	192.168.5.1	192.168.5.2	TCP	40	54337 > http [ACK] Seq=22 Ack=39 Win=2560 Len=0
14	0.094510	192.168.5.2	192.168.5.1	TCP	40	http > 54337 [FIN, ACK] Seq=39 Ack=21 Win=88 Len=0
15	0.094545	192.168.5.1	192.168.5.2	TCP	40	54337 > http [ACK] Seq=22 Ack=40 Win=2560 Len=0
16	0.258224	192.168.5.1	192.168.5.2	TCP	40	54337 > http [FIN, ACK] Seq=21 Ack=40 Win=2560 Len=0
17	0.281889	192.168.5.2	192.168.5.1	TCP	40	http > 54337 [ACK] Seq=40 Ack=22 Win=88 Len=0

En los paquetes de 1 a 3 se establece la conexión entre la maquina que ejecuta la petición con el servidor web instalado en el Arduino. Cabe destacar en el segundo paquete el

MSS=88 que limita el tamaño de los segmentos que puede recibir el Arduino.

```
http > 54337 [SYN, ACK] Seq=4294967288 Ack=1 Win=88 Len=0 MSS=88
```

Una vez establecida la conexión entre el cliente y el servidor, se puede observar que el paquete 4 se envía la petición y en paquete 5 el cliente cierra su parte de conexión por lo que no enviará más datos, pero si recibirá los datos provenientes del Arduino.

```
GET /led HTTP/1.1 Continuation or non-HTTP traffic
54337 > http [FIN, ACK] Seq=21 Ack=4294967289 Win=2560 Len=0
```

A partir de este momento el servidor tratara la petición del cliente y enviara una respuesta en consecuencia a su petición. En el paquete 14 el Arduino informa sobre el cierre de su mitad de conexión.

```
http > 54337 [FIN, ACK] Seq=39 Ack=21 Win=88 Len=0
```

Ampliación del proyecto

A partir de este servidor web que ha quedado operativo, se procede a su modificación para poder transmitir las peticiones de los usuarios a través de él, esto dará acceso a los sensores y actuadores de otros Arduinos que no dispongan de una conexión a Internet pero si de un dispositivo XBee como el Arduinoo que funciona como servidor web, ya que a través de este se enviara una petición tratada debidamente para redireccionarla hacia ellos.

Se ha añadido la librería XBee.h la cual servirá para hacer uso del XBee conectado al PC y también deberá ser incluida en los Arduinos no conectados, esto se tratara más adelante. La inclusión de esta librería permite la creación de variables que nos permitirán realizar la comunicación.

Cabe destacar que es necesario crear dos tipos de variable distintos dependiendo de si van a ser utilizados para recepción o envío de datos. Otro punto que se debe en tener en cuenta es que en la librería XBee.h hace distinción de estas variables según el tamaño de la MAC. Los XBee de los cuales se disponen para la realización de este proyecto poseen una MAC de 64-bit por lo que se elegirán las variables para estos tamaños. Las variables que configuran la comunicación serán las siguientes:

Variables para la transmisión:

XBeeAddress64 addr64;

Tx64Request tx1;

TxStatusResponse txStatus = TxStatusResponse();

Variables para la recepción:

```
XBeeResponse response = XBeeResponse();

Rx64Response rx64 = Rx64Response();
```

Configuramos la entrada por la que le llegaran los datos y por la que se enviaran datos a través del XBee, este es uno de los cambios más relevantes en la función setup().

La parte que más cambios ha sufrido es la que se encarga de la gestión de los datos ya que ahora no debe solo dar una respuesta adecuada a una petición sino que ahora es encargada de leer la información que llega desde la conexión SLIP y redirigirla hacia el Arduino que se indique en la petición. Cabe destacar que las peticiones que lleguen ahora a nuestro servidor web ya no pueden ser las mismas que para simplemente el servidor web conectado al PC, sino que además deberán incluir un campo en el que se indique hacia que Arduino va dirigida la petición.

Una vez recibida la petición esta será analizada por una función dependiendo de si es una petición GET o una petición PUT, después la función reescribirá la petición para ser enviada hacia el Arduino remoto. Estas funciones son muy parecidas simplemente se diferencian en la inclusión de los datos que se desean introducir con un PUT. Al igual que cabía destacar las nuevas variables que introducía la librería XBee.h cabe destacar las funciones para envío y recepción de datos.

Se ha dejado la demostración de la inicialización de la MAC para este momento ya que es después de recibir la petición del usuario cuando se sabrá a que Arduino va dirigida la petición y así poder enviar los datos.

```
addr64 = XBeeAddress64(0x0013a200, 0x408d40b9);

memcpy(dat_env,payload,20);
```

Como ya se indicó antes es necesaria la MAC del XBee para poder enviar los datos hacia el Arduino que se indico en la petición, además a la función hay que introducirle un buffer del tipo uint8_t y su tamaño.

```
tx1 = Tx64Request(addr64, payload, sizeof(payload));
```

Una vez realizada la transmisión de los datos hacia otros Arduinos nos quedamos a la espera de recibir la respuesta. Se leerán todos los bytes hasta que se analiza un paquete, se produce un error, o el buffer está vacío.

```
xbee.readPacket();
```

Tras recibir la respuesta desde el Arduino remoto llamado, se creara la respuesta

correspondiente a la petición realizada por el usuario. Una vez enviada la respuesta se cierra el protosocket que establecía la conexión con el PC y se queda a la espera de nuevas peticiones.

6 - Conclusiones

Este Proyecto Final de Carrera se ha desarrollado un servidor web para Arduino que es capaz de responder a peticiones GET y PUT de pequeño tamaño enviadas des un PC no necesariamente conectado al Arduino. Estas peticiones hacen llamadas a los sensores y actuadores conectados al dispositivo pudiendo ir dirigidas directamente a recursos del Arduino en el que se ejecuta el servidor web o a otros Arduino conectados a él de forma inalámbrica.

Como primer paso para la realización del proyecto, se ha configurado la comunicación entre Arduino y PC mediante el protocolo SLIP. A lo largo del estudio de esta comunicación, se ha encontrado que la librería SerialIP era capaz de soportar funciones de Protosocket por lo que para el análisis de los datos de entrada y el envío se ha decidido hacer uso de la librería Protosocket.h, aunque para la configuración de las conexiones se ha utilizado SerialIP. Protosocket.h ha servido para crear un hilo y establecer el buffer en el que se almacenara la información, que nos llegara del usuario, aparte de encargarse de la gestión de envío o recepción de datos como se hizo mención antes.

Se ha conseguido hacer uso de los actuadores y sensores conectados a los Arduinos según la petición que llega al servidor y se le proporciona una respuesta correcta al cliente. Esto nos ha dado como resultado un servidor web completamente operativo entre Arduino y PC, que es uno de los grandes objetivos por los que se ha llevado a cabo este proyecto.

A la hora de realizar las peticiones se ha utilizado netcat para controlar el tamaño de los paquetes que llegan al Arduino ya que este tiene una limitación hardware que limita el número de bytes que puede recibir en un solo segmento TCP.

Tras la realización del servidor web para un único Arduino, se decidió realizar una ampliación del proyecto con la que hacer llegar las peticiones que llegan al servidor hacia otros Arduinos a través de comunicaciones inalámbricas mediante módulos XBee. Como primer paso para realizar esta ampliación se ha comprobado la comunicación inalámbrica entre Arduinos de manera independiente al servidor web. Una vez llegados a este punto se encontró el problema de que para poder realizar la conexión a Internet, junto con la comunicación inalámbrica de los Arduinos de forma simultánea es necesario un adaptador del que no disponemos. Se ha decidido dejar la comprobación de la comunicación simultánea del proyecto para una ampliación futura debido a la falta de tiempo.

Líneas de desarrollo futuro

La línea de desarrollo más inmediata que se debe llevar a cabo es la realización de una comprobación del código que ha sido creado para hacer llegar la petición desde el Arduino conectado al PC a un Arduino remoto mediante la tecnología XBee y corrección de los errores que pudieran surgir.

Una vez se tuviera el servidor en correcto funcionamiento, se podría proveer a este de algún tipo de mecanismo con el que fuera capaz de obtener conocimiento de las distintas MACs de los Arduinos que se encuentran en su radio de alcance, ya que en esta ocasión se han introducido de un modo manual.

De la misma manera también podrían implementarse mecanismos para que el servidor web pueda devolver información y qué recursos ofrecen. De esta forma los clientes no tendrían que tener un conocimiento previo de los dispositivos que forman parte de la red, lo que facilitaría la realización de cambios en la red.

Finalmente, otra posible ampliación que podría llevarse a cabo consistiría en dar soporte a peticiones de cualquier tamaño. Con esto se podría acceder al servidor desde cualquier navegador web, que típicamente envían peticiones que por su tamaño son segmentadas, lo que dificulta su procesamiento en un dispositivo como un Arduino. Para resolver este problema sería necesario cambiar la forma en la que se procesan las peticiones, ya que actualmente solo se procesa el primer segmento para garantizar la estabilidad del programa. Además de reconstruir las peticiones originales en el buffer de recepción del Arduino se deberían implementar mecanismos que aseguren que éste no va a quedar bloqueado a la espera de segmentos que podrían no llegar.

Bibliografía

http://tools.ietf.org/html/rfc1055

http://www.uhu.es/josel_alvarez/NvasTecnProg/recursos/ProtocoloHTTP.pdf

http://arduino.cc/es/Main/ArduinoBoardMega

http://arduino.cc/en/Main/arduinoBoardMega2560

http://researcher.nsc.gov.tw/public/8905780/Attachment/791017365871.pdf

http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-series1-module#overview

http://mharvan.net/docs/msc-proposal.pdf

http://www.cibernetia.com/manuales/instalacion_servidor_web/1_conceptos_basicos.php

http://contiki.sourceforge.net/docs/2.6/a01793.html#_details

http://contiki.sourceforge.net/docs/2.6/a01689.html