industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Sistema de control distribuido basado en red inalámbrica de bajo coste

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **José Alberto Andrés Fernández**

Director: José Alfonso Vera Repullo

Codirector: Manuel Jiménez Buendía

Cartagena, Julio de 2016

Universidad Politécnica de Cartagena

Agradecimientos

A mis profesores, José Alfonso Vera y Manuel Jiménez, por su inestimable ayuda y a mis padres José Francisco y Manoli, y mi pareja, Clara por su apoyo moral y por estar ahí hasta en los momentos más difíciles.

Índice de contenidos

Índice de ilustraciones

Índice de tablas

Capítulo 1: Introducción

En este capítulo se presentará una introducción al proyecto que se realiza, así como los objetivos a cumplir.

1.1. Introducción

Se dispone de una zona agrícola de amplia extensión, en la cual se desean monitorizar diversos parámetros de un cultivo, así como establecer pautas de riego eficientes. Para ello se espera distribuir, de forma estratégica, los lisímetros que sean necesarios para permitir la medición de la evapotranspiración del cultivo, además de otras medidas interesantes para el control del cultivo como pueden ser el caudal de agua de riego o la humedad del terreno. Para medir dichos parámetros, los lisímetros contarán con 4 células de carga, de las cuales 3 medirán el peso de la zona cultivada y 1 medirá el peso del agua de drenaje. Asimismo, cada lisímetro incorporará una electroválvula que permita vaciar el depósito de drenaje.

Se necesita por tanto diseñar un sistema que permita la lectura de las células de carga, así como el manejo de las electroválvulas, deberá también ser un sistema inalámbrico, que permita su control y supervisión desde una zona remota y sin alimentación, teniendo la mayor autonomía posible.

En cuanto al sistema de riego, se diseñará una placa que permita controlar las electroválvulas de riego y proporcione una interfaz donde añadir sensores como sondas de temperatura y caudalímetros.

Con la meta de minimizar el coste, el sistema estará basado en software y hardware libres. Las plataformas *Arduino* y *Raspberry Pi* serán el corazón del sistema, dado su bajo precio y el fácil desarrollo de aplicaciones. Para la red de comunicación se hará uso de *openHAB*, un software diseñado para la integración de sistemas domóticos, y *MySensors.org*, un conjunto de librerías para el control y comunicación haciendo uso de módulos de radiofrecuencia. Asimismo, otros componentes como las baterías, electroválvulas y demás hardware serán seleccionados teniendo en cuenta su coste.

1.2. Objetivos

La finalidad del proyecto será por tanto la definición y desarrollo de un sistema de control inalámbrico, basado en los componentes de bajo coste ya mencionados. El sistema será aplicado a la automatización y control de varias zonas de riego, cada cual con su correspondiente lisímetro.

Los puntos a cumplir para la realización del proyecto serán por tanto los siguientes:

- Creación de un servidor web, que actuará tanto de controlador como de base de datos.
- Diseño de un “nodo sensor”, el cual leerá los datos del lisímetro en el que se instale y los enviará tanto al controlador como a un “nodo actuador”.
- Desarrollo de un “nodo actuador” el cual tomará la información de los sensores y del controlador como datos para iniciar y finalizar el riego.
- Crear una aplicación de móvil que facilite el acceso al servidor.

Dado que la red de control se encontrará en zona agrícola, no tendrá acceso a la red eléctrica excepto en algunos puntos. El sistema deberá cumplir, por tanto, ciertas condiciones:

- Ser un sistema inalámbrico, los nodos se deberán comunicar por radio.
- Tener una alta autonomía.
- Disponer de un método de carga de baterías.
- Permitir la configuración del usuario con una interfaz intuitiva.
- Permitir el acceso al control de forma inalámbrica.
- Restaurar los datos ante cortes de luz o reinicios.

Capítulo 2: Estado del arte

En este capítulo se expondrá la base sobre la que se desarrollarán los distintos apartados del proyecto.

2.1. Lisímetros de pesada

Para la medición de las necesidades hídricas de un cultivo, es necesario estimar o calcular la evapotranspiración del mismo. Para ello existen diversos métodos, siendo los más comunes los de balance de agua y los de transferencia de vapor de agua. Para el caso de balance de agua, uno de los sistemas más utilizados son los lisímetros de pesada, o evapotranspirómetros (Shuttleworth, 2008).

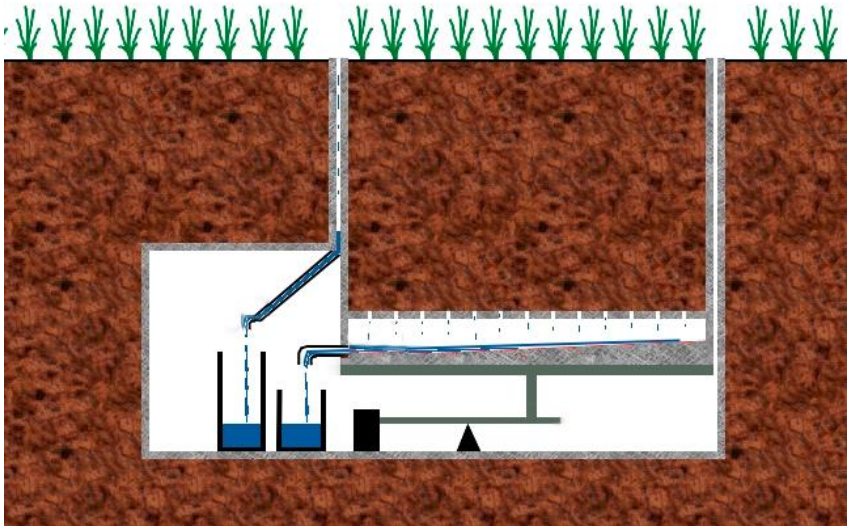

Ilustración 1 - Lisímetro de pesada

Con objeto de llevar a cabo las medidas necesarias, los lisímetros hacen uso de diversos sensores, como pueden ser células de carga y caudalímetros, con las cuales se extrae información acerca del peso del cultivo, el peso del agua drenada y el volumen de agua regada. Otros parámetros climatológicos suelen medirse para minimizar los errores a la hora de realizar los cálculos necesarios. Esta información es luego procesada por un controlador que, de acuerdo a los parámetros medidos, llevará a cabo el riego del cultivo.

Por norma general, los lisímetros de pesada son grandes y costosas estructuras, ocultas bajo tierra las cuales se rellenan con el terreno del mismo lugar para simular las condiciones de las plantas que lo rodean. Para este proyecto se utilizará el lisímetro de bajo coste, patentado por la spin-off de la UPCT, Telenatura (Telenatura, 2015).

Ilustración 2 - Lisímetro utilizado en el diseño

El lisímetro, cuya estructura se puede observar en la ilustración anterior, puede soportar un peso de hasta 90 Kg, dado el límite de sus tres células de carga, de 30 Kg cada una. Para el drenaje, cuenta con un depósito de plástico, con un volumen algo inferior a 4 L.

Respecto a los parámetros a utilizar cabe destacar los siguientes:

- PWFC: Peso de la planta en capacidad de campo, es decir, su peso tras un riego y habiendo completado el drenaje.
- CPW: Peso actual de la planta.
- CDW: Peso actual del drenaje.

Estos serán los que se requieran en el algoritmo de riego por lisimetría. El volumen de agua de riego y el caudal se utilizarán en todos los modos de riego.

2.2. MySensors.org

Tal y como ya se ha comentado, se utilizarán las librerías de MySensors.org como base para la red de comunicación y de control. Las licencias que ofrecen permiten su uso libre en casos de no ser productos comerciales. En caso de serlo, habría que contactar con la empresa.

2.2.1. Introducción a MySensors.org

Ilustración 3 - Logotipo MySensors (MySensors.org, 2016)

El sistema está diseñado para trabajar con distintos nodos, cuya comunicación gestionan las librerías de MySensors.org. Para establecer una "red" de comunicación nombra a los nodos de la siguiente manera:

- Nodos Sensores: Encargados de leer los datos y mandarlos vía radio.
- Nodos Repetidores: Similares a los anteriores, pero con la capacidad de trasmitir mensajes de otros nodos.
- Puerta de enlace (Gateway): Encargada de transformar la información de los nodos a un controlador y viceversa.

Cada nodo tiene asignado, de forma manual o automática, un único *SensorID*, que lo identifica y diferencia de los demás nodos. Si a la hora de programar el Nodo, no se le establece ningún *SensorID*, en su primer arranque intentará obtenerlo de forma automática del *Gateway*. Una vez el Nodo ha conseguido su *ID*, lo guarda en la memoria EEPROM, de forma que al siguiente arranque esté disponible. Una vez ha "seleccionado" su *ID*, intentará encontrar el camino más corto hasta el *Gateway*, es decir, aquel camino que implique un menor número de repetidores hasta llegar a él. Si un Nodo falla 3 veces consecutivas en alcanzar el *Gateway*, considerará que ha perdido la comunicación y reintentará encontrar el camino más corto o directo.

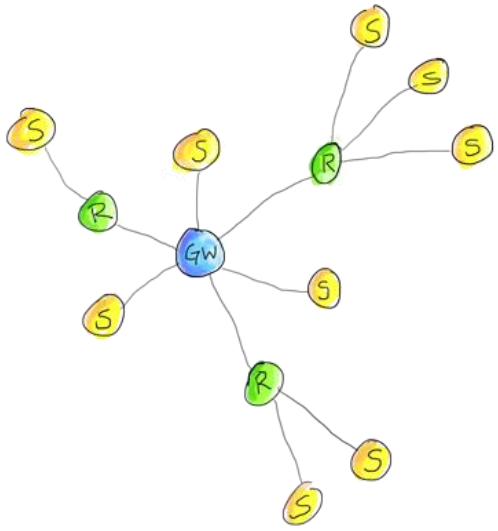

Ilustración 4 - Red Mysensors (MySensors.org, 2016)

En una misma red pueden coexistir hasta 254 Nodos, y cada uno puede enviar datos de hasta 254 *Child Sensors*, es decir, 254 señales diferentes. Por tanto, en el sistema de control se podrían recibir datos de hasta 64516 sensores.

Respecto al *Gateway,* existen 4 tipos predefinidos con las librerías de MySensors.org, dependiendo principalmente del controlador que se quiera usar posteriormente. Éstos son los siguientes:

- *Serial Gateway*: Diseñado para ir conectado directamente al controlador a través de un puerto USB.
- *Ethernet Gateway:* Diseñado para comunicar con el controlador a través de Ethernet.
- *MQTT Gateway:* Muy similar al anterior, con la diferencia de que transforma los mensajes a una forma compatible con el protocolo MQTT, para poder ser utilizado con cualquier sistema compatible con MQTT. (Ver 2.4)
- *ESP8266 Wifi Gateway (Nuevo):* Recientemente MySensors.org ha añadido una versión similar al *Ethernet Gateway* pero con comunicación inalámbrica Wifi.

2.2.2. Protocolo Serie v1.5

El protocolo de comunicación utilizado entre los nodos se basa en una lista de valores separados por punto y coma. El mensaje tendría la siguiente forma:

Ilustración 5 - Formato de comunicación serie (MySensors.org, 2016)

Este mensaje se envía a los nodos siguiendo la ruta calculada previamente hasta el *Gateway*. Cada uno de los valores tiene un significado, éstos son los siguientes:

node-id	La id única del Nodo que envía el mensaje.
child-sensor-id	Identifica el sensor del Nodo que envía el mensaje.
message-type	Tipo de mensaje enviado.
ack	Define si el mensaje es de tipo *acknowledge.*
sub-type	Dependiendo de *message-type* tiene un significado diferente.
payload	El mensaje enviado, o instrucción.

Tabla 1 - Elementos de la estructura (MySensors.org, 2016)

Tal y como se especifica en la tabla anterior, los mensajes pueden ser de diversos tipos, éstos son: *Presentation, set, req, interal y stream,* cada uno de ellos utilizado para algo en concreto. Los más usados serán *presentation* (al inicio del programa) y *set* (establece un mensaje de un sensor a otro, o al *Gateway*). Los mensajes de tipo *internal* se utilizan para enviar variables internas, como estado de batería o el tiempo actual. No son necesarios en un funcionamiento normal.

Los *sub-type*, van asociados entre sí, cada mensaje *presentation* hace referencia a un *sub-type* de tipo *set o req*. Éstos son los siguientes:

Type	Comment	Variables
S_DOOR	Door and window sensors	V_TRIPPED, V_ARMED
S_MOTION	Motion sensors	V_TRIPPED, V_ARMED
S_SMOKE	Smoke sensor	V_TRIPPED, V_ARMED
S_LIGHT	Light Actuator (on/off)	V_STATUS (or V_LIGHT), V_WATT
S_BINARY	Binary device (on/off)	V_STATUS (or V_LIGHT), V_WATT
S_DIMMER	Dimmable device of some kind	V_STATUS (on/off), V_DIMMER (dimmer level 0-100), V_WATT
S_COVER	Window covers or shades	V_UP, V_DOWN, V_STOP, V_PERCENTAGE
S_TEMP	Temperature sensor	V_TEMP, V_ID
S_HUM	Humidity sensor	V_HUM
S_BARO	Barometer sensor (Pressure)	V_PRESSURE, V_FORECAST
S_WIND	Wind sensor	V_WIND, V_GUST
S_RAIN	Rain sensor	V_RAIN, V_RAINRATE
S_UV	UV sensor	V_UV
S_WEIGHT	Weight sensor for scales etc.	V_WEIGHT, V_IMPEDANCE
S_POWER	Power measuring device, like power meters	V_WATT, V_KWH
S_HEATER	Heater device	V_HVAC_SETPOINT_HEAT, V_HVAC_FLOW_STATE, V_TEMP
S_DISTANCE	Distance sensor	V_DISTANCE, V_UNIT_PREFIX
S_LIGHT_LEVEL	Light sensor	V_LIGHT_LEVEL (uncalibrated percentage), V_LEVEL (light level in lux)
S_ARDUINO_NODE	Arduino node device	
S_ARDUINO_REPEATER_NODE	Arduino repeating node device	
S_LOCK	Lock device	V_LOCK_STATUS

Type	Comment	Variables
S_IR	Ir sender/receiver device	V_IR_SEND, V_IR_RECEIVE
S_WATER	Water meter	V_FLOW, V_VOLUME
S_AIR_QUALITY	Air quality sensor e.g. MQ-2	V_LEVEL, V_UNIT_PREFIX
S_CUSTOM	Use this for custom sensors where no other fits.	
S_DUST	Dust level sensor	V_LEVEL, V_UNIT_PREFIX
S_SCENE_CONTROLLER	Scene controller device	V_SCENE_ON, V_SCENE_OFF
S_RGB_LIGHT	RGB light	V_RGB, V_WATT
S_RGBW_LIGHT	RGBW light (with separate white component)	V_RGBW, V_WATT
S_COLOR_SENSOR	Color sensor	V_RGB
S_HVAC	Thermostat/HVAC device	V_HVAC_SETPOINT_HEAT, V_HVAC_SETPOINT_COLD, V_HVAC_FLOW_STATE, V_HVAC_FLOW_MODE, V_HVAC_SPEED
S_MULTIMETER	Multimeter device	V_VOLTAGE, V_CURRENT, V_IMPEDANCE
S_SPRINKLER	Sprinkler device	V_STATUS (turn on/off), V_TRIPPED (if fire detecting device)
S_WATER_LEAK	Water leak sensor	V_TRIPPED, V_ARMED
S_SOUND	Sound sensor	V_LEVEL (in dB), V_TRIPPED, V_ARMED
S_VIBRATION	Vibration sensor	V_LEVEL (Hz), V_TRIPPED, V_ARMED
S_MOISTURE	Moisture sensor	V_LEVEL (%), V_TRIPPED, V_ARMED

Tabla 2 - *Sub-types* & *Variables* (MySensors.org, 2016)

Como se puede observar, MySensors va preparado, por defecto, para enviar mensajes de todo tipo de sensores y a un gran número de actuadores, desde sensores de humedad hasta sistemas de calefacción. Sin embargo, aunque los mensajes incorporen un "tipo de mensaje", no son suficientes para realizar ningún control. Para ello es necesario diseñar y programar las estructuras que recibirán estos datos y los procesarán para comunicarlos con "el mundo real". Una de las piezas necesarias para el correcto funcionamiento de la red, ha de ser un controlador, en el que se puedan implementar algoritmos para todas las variables recibidas o a enviar.

2.2.3. Funciones de las librerías

Con el fin de facilitar la comunicación y el uso de las librerías, MySensors.org ofrece un conjunto de funciones y objetos destinados al control del sistema. La lista completa se puede encontrar en la página oficial o entrando a las librerías. A continuación, se explican las básicas para comenzar a funcionar:

- MySensor *instancia;* Primero hay que crear una instancia.
- *instancia.begin();* Inicializa la radio y la comunicación.
- *instancia.present();* Presenta cada sensor al *Gateway*.
- *instancia.send();* Envía un mensaje a otro nodo o al *Gateway*.
- *instancia.process();* Lee mensajes de radio, esencial para actuadores.
- *instancia.sleep();* Duerme al nodo, es decir, lo pone en bajo consumo.
- *instancia.wait();* Espera un tiempo, mientras llama a *process();*

Con estas y otras funciones se consigue establecer comunicación entre sensores y actuadores. De los parámetros de cada función se hablará posteriormente cuando se explique el código del programa.

Ésta y mucha más información, así como ejemplos, se pueden encontrar en la página oficial, MySensors.org.

2.3. OpenHAB

Actualmente las librerías de MySensors.org pueden trabajar, con mayor o menor compatibilidad, con diversos controladores, entre los que destacan:

- Domoticz
- DomotiGa
- PiDome
- Vera
- OpenHAB

La lista completa se puede encontrar en la página oficial de MySensors.org, con formas de conexión y ejemplos. Todos los controladores para los que existe compatibilidad, están diseñados en su origen para sistemas domóticos, pero podrán adaptarse a nuestro caso de estudio. De entre todos estos controladores, uno de los más versátiles es openHAB, principalmente debido a su compatibilidad con otros sistemas.

OpenHAB es un conjunto de paquetes OSGi[1] definidos en un marco OSGi, por tanto, es una solución en Java con una arquitectura altamente modular, la cual permite añadir o eliminar funcionalidades incluso durante el tiempo de ejecución. En la siguiente ilustración se puede ver un resumen de la arquitectura openHAB:

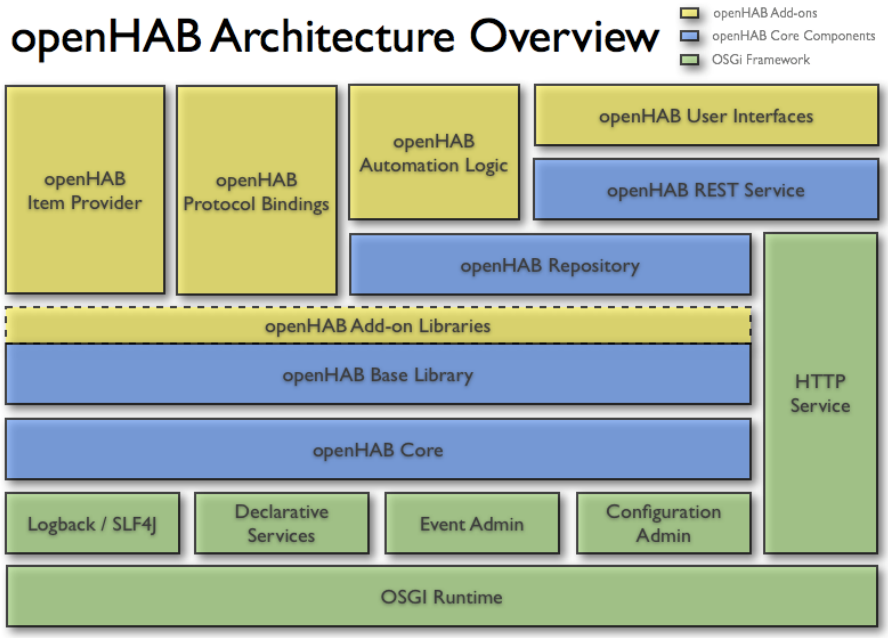

Ilustración 6 - Arquitectura openHAB (openHAB, 2016)

OpenHAB puede ser instalado en la mayoría de sistemas operativos actuales, incluyendo Linux, MAC OS y diversas versiones de Windows. Al instalar openHAB se puede instalar además una configuración *Demo* que permite ver las principales formas de configuración y varios ejemplos de uso.

2.3.1. Comunicación

Respecto a la comunicación interna, openHAB utiliza dos canales: un bus de eventos asíncronos, y un repositorio de estados.

2.3.1.1. El bus de eventos

El bus de eventos es la base de servicio de openHAB. Todos los paquetes que no requieran el uso de estados para su funcionamiento, utilizarán el bus de eventos. Existen dos tipos de eventos: comandos, que desencadenan una acción o un cambio de estado de algún objeto/dispositivo, y actualizaciones de estado, que informan sobre un cambio de estado de algún objeto/dispositivo (normalmente respuestas de comandos).

Todos los enlaces a protocolos (denominados *bindings*, permiten la comunicación con otros sistemas) se comunican haciendo uso del bus de eventos. De esta

[1] OSGi son las siglas de *Open Services Gateway Initiative, una especificación software diseñada para permitir el desarrollo de plataformas compatibles que puedan proporcionar múltiples servicios. OSGi se ha definido con una serie de interfaces de programación de aplicaciones (API) básicas para el desarrollo de servicios. (Wikipedia, 2016)*

forma se asegura el bajo acoplamiento entre paquetes, facilitando la naturaleza dinámica de openHAB.

Es importante destacar que openHAB sirve como integración entre distintos sistemas y como mediador entre los diferentes protocolos que utilicen dichos sistemas. Aunque no es lo usual, es posible conectar diversas instancias de openHAB distribuidas a través del bus de eventos (OpenHAB, 2016).

2.3.1.2. Repositorio de objetos

No toda la comunicación puede cubrirse puramente con servicios de eventos. Por esta razón openHAB ofrece el repositorio de objetos, el cual, conectado al bus de eventos, mantiene la información del estado actual de todos los objetos. El repositorio puede ser usado siempre que sea necesario acceder al estado actual de un objeto. El repositorio evita que cada paquete tenga que almacenar los estados por si mismos para su uso interno.

El repositorio de objetos se asegura asimismo, de que los estados estén sincronizados para todos los paquetes, permitiendo guardar los estados a un sistema de archivos o una base de datos, de forma que se mantengan tras un reinicio del sistema (OpenHAB, 2016). En la Ilustración 7 se puede ver un diagrama que muestra dicha comunicación.

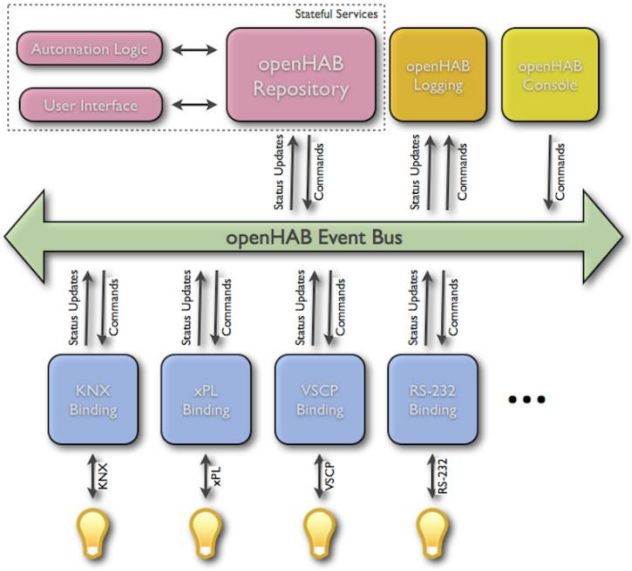

Ilustración 7 - Comunicación openHAB (openHAB, 2016)

2.3.2. *Bindings*

Se denominan *bindings* al conjunto de paquetes que permiten la integración en openHAB de sistemas de diferentes índoles. Estos *bindings* permiten la comunicación con dispositivos, interfaces, y protocolos, como por ejemplo comunicación con dispositivos bluetooth, con Dropbox, o el protocolo HTTP. En la lista a continuación aparecen sólo algunos de los dispositivos y protocolos para los cuales existe un *binding,* es decir, aquellos que se pueden integrar con openHAB:

- Astro: Utilizado para calcular la posición del sol y la luna.

- Bluetooth: Permite emparejar openHAB con dispositivos, permitiendo nuevas funcionalidades.
- Daikin: Permite la comunicación con un dispositivo Daikin, utilizando un controlador especial.
- Dropbox: Principalmente para crear un log en la nube.
- MQTT: Comunicación con el protocolo MQTT, del cual se hablará posteriormente.

La lista completa de *bindings* y su integración con openHAB, se pueden encontrar en la *Wiki* de openHAB en GitHub. Asimismo, la comunidad puede crear nuevos *bindings,* en java, para comunicar con nuevos dispositivos o protocolos, ampliando de forma casi ilimitada la funcionalidad de este sistema.

2.3.3. Objetos

Los objetos, o *Items*, definen las señales de entrada y salida que pueden comunicar con openHAB. Pueden ser leidos o escritos y pueden enlazarse con Bindings, de forma que interactúen con los sistemas y protocolos ya mencionados. Los tipos de objetos que se pueden definir se pueden encontrar en la tabla siguiente:

Tipo	Descripción	Tipos de dato aceptados	Tipos de comandos aceptados
Call	Para objetos de telefonía.	Call	-
Color	Se usa para valores de color.	OnOff, Percent, HSB	OnOff, IncreaseDecrease, Percent, HSB
Contact	Se puede usar para sensores que devuelven señal binaria.	OpenClosed	-
DateTime	Almacena fecha y hora.	DateTime	DateTime
Dimmer	Acepta valores de porcentaje y binarios, para luces.	OnOff, Percent	OnOff, IncreaseDecrease, Percent
Group	Une diferentes *Items*	Depende de los objetos agrupados	Depende de los objetos agrupados.
Location	Almacena posición.	Point	Point
Number	Valor decimal para sensores y contadores.	Decimal	Decimal
Rollershutter	Permite el control de persianas.	UpDown, Percent	UpDown, StopMove, Percent
String	Representa cadenas de texto.	String, DateTime	String
Switch	Interruptores binarios.	OnOff	OnOff

Tabla 3 - Tipos de objetos (openHAB, 2016)

Para definir un objeto, en openHAB hay que seguir una estructura, dentro de un archivo *.items* se definen como se indica a continuación, siendo opcionales las configuraciones entre los símbolos [ ].

Tipo nombredelitem ["etiqueta"] [<icono>] [(grupos)] [{bindings}]

El campo *nombredelitem* debe ser único para cada objeto, siendo este parámetro el que define dicho objeto. En cambio, el parámetro *etiqueta* puede ser igual, al ser su función la de describir el objeto y darle formato a su texto de salida. Este formato se basa en la sintaxis de formato de Java, es decir, ha de seguir la siguiente estructura:

%[argument_index$][flags][width][.precision]conversión

Como ejemplo una definición completa de un objeto, que pertenezca al grupo *Luces*, haga uso de un *binding KNX y* tenga el icono *bombilla,* podría ser:

Switch LuzCocina "La luz está a %d" <bombilla> (Luces) {knx="1/0/1+0/0/1"}

Al mostrar este switch en un servidor web, mostraría algo similar a*: La luz está a 1*. Para poder cambiar el patrón mostrado con %d habrá que hacer uso de transformaciones, como por ejemplo decir que 1=Encendida.

La definición completa de los *Items* y su funcionamiento se puede encontrar en la *Wiki* de openHAB. Los enlaces se dejarán en la bibliografía.

2.3.4. *Sitemap*

La representación de datos, y la forma de interactuar con ellos se hace a través de una interfaz web, en la cual existen *widgets* o formas especiales de visualizar los objetos. Dependiendo del tipo de objeto que se desee representar, será posible utilizar un *widget* u otro. Los *sitemap*, mapas web, se guardan como un archivo de texto con la extensión .sitemap, y al igual que los *ítems* han de seguir una estructura específica.

Lo primero será siempre la definición del *sitemap*, escribiendo, al inicio del archivo, lo siguiente: sitemap [nombredelsitio] [label="<Titulo de la página>"] { }. Una vez más siendo opcional los parámetros entre corchetes. La definición de la página irá dentro de las llaves, quedando como un conjunto.

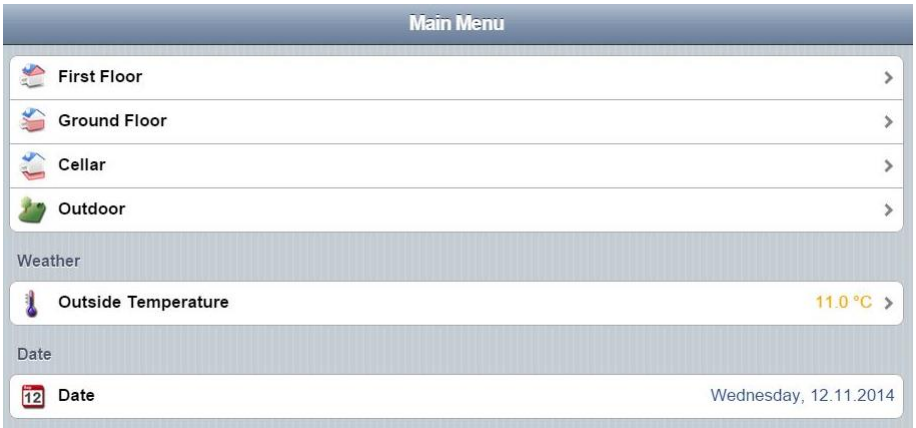

Ilustración 8 - Menu principal. Demo openHAB (openHAB, 2014)

Un ejemplo de distintios *widgets* se puede encontrar en la ilustración a continuación, donde se pueden ver hasta cuatro formas de representar datos, con interruptores, pulsadores arriba/abajo, indicadores numéricos e indicadores binarios.

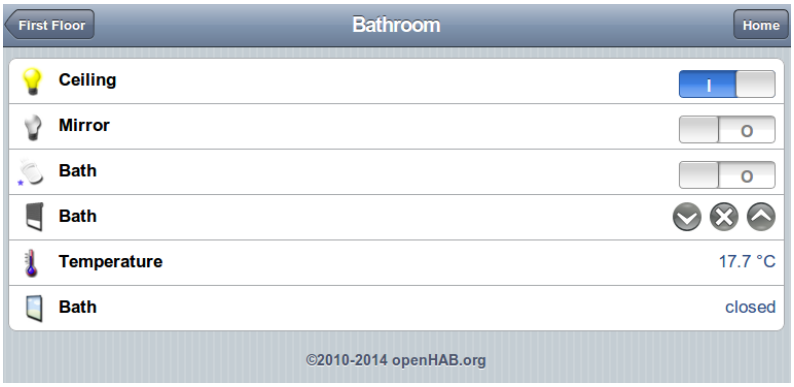

Ilustración 9 - Página "Bathroom". Demo openHAB (openHAB, 2014)

El uso de los distintos tipos de formas de representación y control, permite una personalización muy elevada, consiguiendo así adaptar openHAB a casi cualquier tipo de requerimiento. La lista completa de *widgets* que se pueden utilizar en un *sitemap* se encuentra en la tabla siguiente:

Widget	**Descripción**
Colorpicker	Inserta una rueda de color para elegir valores.
Chart	Añade un gráfico de tiempo con datos.
Frame	Enmarca varias páginas, objetos, u otros frames.
Group	Muestra todos los elementos de un grupo.
Image	Reproduce una imagen.
Selection	Da acceso a una página de selección entre varios valores.
Setpoint	Muestra un valor y permite modificarlo con pulsadores de Arriba/Abajo.
Slider	Muestra una barra deslizante.
Switch	Muestra un interruptor.
Text	Reproduce un elemento de texto.
Video	Muestra un video.

Widget	Descripción
Webview	Navegador web embebido.

Tabla 4 - Widgets openHAB (openHAB, 2016)

Además de la propia funcionalidad de los *widgets*, a éstos se les pueden añadir mapeos, colores, y visibilidad, de forma que se puedan diseñar webs más dinámicas. Todos los widgets y sus formas de uso se pueden encontrar en la *Wiki* oficial de openHAB.

2.3.5. Reglas y scripts

Con el fin de aumentar la capacidad de integración y las funcionalidades de openHAB, éste incluye un sistema de reglas y scripts. Las reglas se escriben en un archivo con extensión *.rules,* en el cual se escriben siguiendo una estructura determinada. Las reglas permiten realizar acciones con la activación de *ítems* o de estados del sistema. Las reglas pueden asimismo llamar a scripts, definidos en un archivo *.script*, los cuales pueden estar escritos en Javascript, Jython y Groovy. Son precisamente las reglas y scripts las que dotan de una alta versatilidad a openHAB.

La estructura que siguen las reglas es la siguiente:

```
rule "nombre de la regla"
        when
                <TRIGGER_CONDITION1> or
                <TRIGGER_CONDITION2> or
                <TRIGGER_CONDITION3>
                ...
        then
                <EXECUTION_BLOCK>
end
```

Donde los *trigger* pueden ser: Eventos de *Items*, eventos de tiempo o eventos de sistema. Y los *execution* serán las acciones a realizar por la regla, pudiendo ser desde llamar a scripts hasta activar una salida.

2.3.6. Persistencia

Almacenamiento y restauración de los datos. En muchas ocasiones es interesante la opción de almacenar datos a lo largo del tiempo, para crear gráficos, comparar resultados, o para restaurar configuraciones tras un reinicio. Como solución, openHAB ofrece servicios de persistencia, es decir, compatibilidad con sistemas y protocolos de almacenaje de datos. Algunos de los servicios más conocidos que ofrece actualmente son los siguientes:

Servicio	Link	Notas
Cosm	cosm.com	Guarda estados de *items en* Cosm/Xively
db4o	db4o.com	Base de datos ligera en Java
InfluxDB	influxdata.com	Base de datos open-source
JDBC	wikipedia	Java Database Connectivity para MySQL y otros
Logging	logback.qos.ch	Escribe estados de *items* a un log.
MapDB	mapdb.org	Guarda el último estado del objeto.
MongoDB	mongodb.com	Base de datos NoSQL.
MQTT	wikipedia	Envía los datos a un bróker MQTT (Ver 2.4)
my.openHAB	my.openHAB.org	Envía los estados a la nube de openHAB.
MySQL	mysql.com	Base de datos clásica MySQL.
RRD4J	RRD4J	Versión Java de RRDtool. Almacena estados numéricos únicamente.

Tabla 5 - Servicios de persistencia (openHAB, 2016)

Una vez elegido el servicio, o los servicios que se desean utilizar, es importante crear las reglas de persistencia, es decir, las reglas que indican qué objetos se guardarán y cuándo se guardarán. Para ello es necesario crear un archivo de configuración, en su carpeta correspondiente de openHAB, con la forma <*nombredelservicio*>.persist. Dentro de dicho archivo se escribirán las estrategias de persistencia como sigue:

```
Strategies {
   <strategyName1> : "<cronexpression1>"
   <strategyName2> : "<cronexpression2>"
   ...

   default = <strategyNameX>, <strategyNameY>
}
```

Seguido de los objetos que seguirán dichas estrategias:

```
Items {
   <itemlist1> [-> "<alias1>"] : [strategy = <strategy1>, <strategy2>, ...]
```

```
  <itemlist2> [-> "<alias2>"] : [strategy = <strategyX>, <strategyY>, ...]
  ...
}
```

De esta forma quedarían definidos los objetos a “recordar” y la frecuencia de guardado de los datos.

2.3.7. OpenHAB Designer

Para poder facilitar, de forma considerable, la configuración de openHAB y evitar en medida de lo posible editar archivos de texto sin ayuda alguna, openHAB dispone de un software llamado openHAB Designer. Este programa permite abrir la configuración de un servidor openHAB y editarla directamente, proporcionando una interfaz agradable, un sistema de validación de sintaxis y un sistema de asistencia a la hora de escribir el código. Incluye asimismo un navegador web para poder visualizar el servidor mientras es editado, sin embargo su funcionamiento no es óptimo y es recomendable utilizar un navegador aparte.

Como se puede observar en la ilustración a continuación, el software permite tener un control de todos los archivos actuales en la configuración de openHAB, así como abrirlos y editarlos, ayudando a su procesamiento. Además se puede observar la interfaz del software y el resaltado de palabras, permitiendo una mayor comodidad a la hora de escribir reglas, *sitemaps* o definir *ítems*.

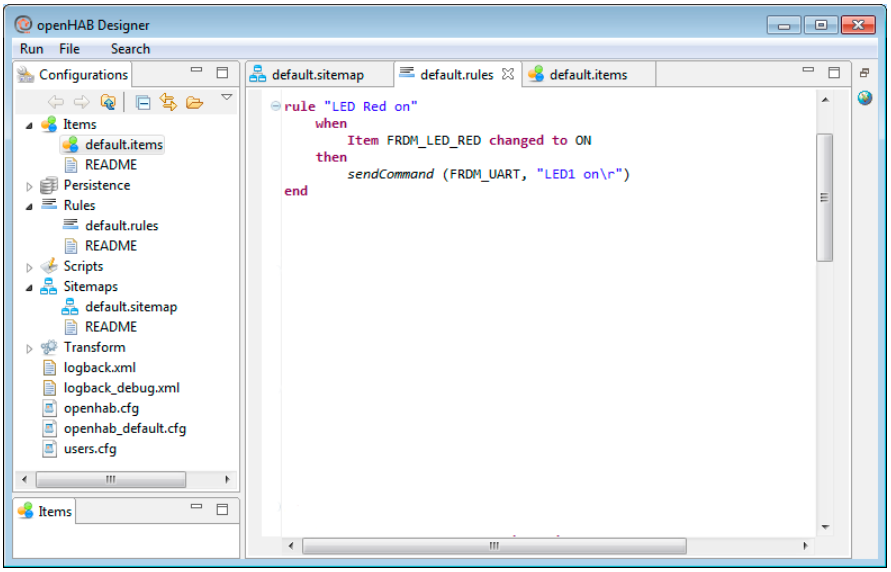

Ilustración 10 - Captura de openHAB Designer

El programa tiene versiones de Linux, Windows y MAC, pudiendo ejecutarlo en el mismo ordenador que el servidor, o desde un ordenador remoto.

2.4. Protocolo MQTT

MQTT es un protocolo de conectividad máquina-a-máquina. Diseñado como un sistema de mensajería *publish-suscribe* extremadamente ligero. Es muy útil para conexiones donde sea necesaria una mínima cantidad de datos. Por ejemplo, se utiliza

para comunicaciones vía satélite o en comunicaciones por línea conmutada (Dial-up), donde limitar la cantidad de datos es crucial. Fue inventado en 1999 por el Dr. Andy Stanford-Clark de IBM, y Arlen Nipper de Arcom (MQTT.org, 2015).

La forma de publicar o recibir mensajes es suscribiéndose a un tema. Los clientes pueden conectarse y enviar información. Existe un bróker que se encarga de gestionar la comunicación, actuando como servidor. Para los temas se respeta una jerarquía establecida por "/". Un ejemplo de tema sería el siguiente:

Sensores/Salón/Temperatura/SensorNorte

De esta forma, el sensor de temperatura *SensorNorte,* publicaría sus mensajes en el tema escrito, mientras que el resto de clientes, como por ejemplo una pantalla, se suscribirían al mismo tema para leer la información.

Existe asimismo la posibilidad de suscribirse a varios temas. Una de las opciones para ello es el uso de los síbolos "+" y "#", que permiten suscribirse a un nivel completo de jerarquía o a todos los niveles restantes, es decir:

Sensores/Salón/Temperatura/+ → Suscripción a todos los sensores de temperatura del salón.

Sensores/+/Temperatura/SensorNorte → Suscripción a todos los *SensorNorte* de las zonas de la casa.

Sensores/Salón/# → Suscripción a todos los sensores del salón y todos los subtipos.

→ Suscripción a todo, equivalente a +/+/+/+ considerando cuatro niveles.

Respecto a la calidad de servicio, QoS, el protocolo MQTT define tres niveles:

- 0: El mensaje se envía 1 vez sin confirmación.
- 1: El mensaje se envía al menos 1 vez, con confirmación.
- 2: El mensaje se envía una vez exactamente, haciendo uso de un *handshake* de 4 pasos.

La descripción completa y detalles sobre el protocolo se dejarán en la carpeta Documentos/*Doc de Interes* del CD del presente TFG.

Los mensajes enviados por el sistema diseñado tendrán la siguiente forma, establecida por MySensors.org:

[PREFIJO_BROKER]/[ID_Nodo]/[ID_Sensor]/V_[Tipo_Sensor]

Capítulo 3: Hardware

En este capítulo se presentará el hardware utilizado, así como algunas alternativas descartadas.

3.1. Control

Para el control del sistema, tanto a nivel de E/S como a nivel de bases de datos y servidor, se hará uso de los componentes de bajo coste que a continuación se mencionan. Las hojas de datos de los componentes principales se encontrarán en la carpeta denominada *Datasheets* dentro del directorio *Documentos* del CD del presente TFG.

3.1.1. Raspberry Pi

Raspberry Pi es un ordenador de placa reducida (SBC) de bajo coste, desarrollada en su origen con la idea de enseñar programación en colegios de todo el mundo. Es precisamente este bajo precio el que ha conseguido introducir la pequeña placa en todo el mundo, generando un gran desarrollo a su alrededor.

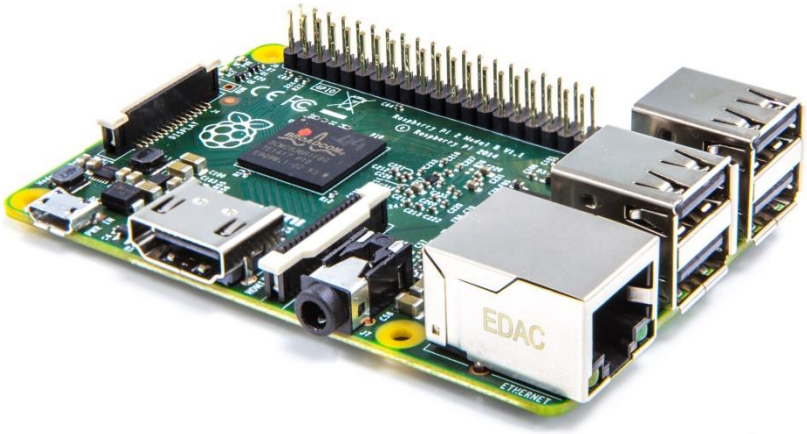

Ilustración 11 - Raspberry Pi 2 Model B (Raspberry Pi Foundation, 2016)

Actualmente se han desarrollado diversos modelos del ordenador, manteniendo el precio de salida en las versiones actualizadas. A la hora de realizar el presente trabajo salió al mercado el modelo Raspberry Pi 2 Model B, por lo que es el ordenador utilizado. Las características principales del mismo son las siguientes:

- **Arquitectura:** ARMv7 (32-bit)
- **SoC:** Broadcom BCM2836
- **CPU:** 900 MHz 32-bit quad-core ARM Cortex-A7
- **GPU:** Broadcom VideoCore IV @ 250 MHz
- **SDRAM:** 1 GB (Compartida con GPU)

Estas especificaciones junto a una buena combinación de puertos USB, puerto Ethernet y un tamaño minúsculo, permiten el uso de Raspberry Pi en diversos proyectos. Para ello la comunidad de desarrolladores y algunas empresas han modificado diversos sistemas operativos para ser compatibles con la placa. Los más destacados son:

- OpenELEC
- RISC OS
- Windows 10 IoT Core
- Raspbian

Exceptuando Windows 10 IoT Core, todos los sistemas son versiones modificadas de Linux que se centran en ciertas funcionalidades específicas, como el contenido multimedia (OpenELEC) o sistemas de propósito múltiple (Raspbian).

En el caso de nuestro controlador, se hará uso de Raspbian, al ser uno de los sistemas que menos recursos consume y uno de los más personalizables. Dado que es un sistema operativo basado en Debian (Linux), se podrá hacer uso de todas las características más conocidas del mismo. El sistema operativo estará dentro de una tarjeta micro SD de 16Gb, la cual permitirá almacenar una gran cantidad de datos.

Durante el desarrollo de este trabajo, Raspberry Pi Foundation diseñó la tercera versión de su conocido ordenador, Raspberry Pi 3 Model B, la cual mejora considerablemente las características de su predecesora, pero el cambio a una nueva versión se deja como una posible mejora u optimización.

3.1.2. Arduino

El "rey" del Open-Source. La conocida plataforma italiana de desarrollo diseñada con un objetivo similar al de Raspberry Pi, el de llevar la programación a todos los rincones del mundo. Su diseño original (Arduino UNO) permite con un simple cable USB la programación de un micro controlador para diversas tareas. Al igual que en el caso de Raspberry Pi, una comunidad de desarrolladores ingente ha aparecido apoyando el proyecto, lo cual hace considerablemente más fácil la creación de proyectos de inimaginables. El software utilizado para su programación (Arduino IDE) es también open-source y está basado en Processing. En este proyecto se hará uso de dos de sus placas más conocidas, Arduino UNO y Arduino Pro Mini.

3.1.2.1. Arduino UNO Rev. 3

Arduino UNO, o Genuino UNO como se llama actualmente en Europa, está basado en el chip ATmega328P, y sus características principales son las siguientes:

- **Pines E/S digitales:** 14 (De los cuales 6 poseen capacidad PWM)
- **Entradas Analógicas:** 6 (Con una resolución de 10 bits)
- **Reloj:** 16 Mhz
- **Memoria Flash:** 32 KB
- **SRAM:** 2 KB
- **EEPROM:** 1 KB
- **Comunicación:** UART, SPI, I2C

Asimismo, el uso de un chip ATmega16U2, programado como conversor USB-serie, permite la carga de código y lectura del puerto serie, a través del USB COM del PC. Como se puede observar, las características de este modelo no son las más altas que se pueden conseguir en un micro controlador, sin embargo, ha sido su facilidad de uso la que lo ha convertido en uno de los principales candidatos a la hora de realizar cualquier proyecto.

Ilustración 12 - Arduino UNO r3 (Arduino, 2016)

Además, la creación por parte de la comunidad de diversos "shields" ha conseguido simplificar aún más el uso de otros componentes que dotan a Arduino de una potencia mucho mayor. Estos *shields*, junto a las librerías diseñadas para ellos, han permitido comunicar fácilmente Arduino con todo tipo de sensores y actuadores, así como otros protocolos de comunicación como Ethernet o Bluetooth.

Uno de los aspectos más importantes será la distribución de sus pines, y su funcionalidad, dado que será crítico a la hora de comunicar y conectar entradas y salidas. A continuación, se puede ver una imagen bastante descriptiva de dichos pines.

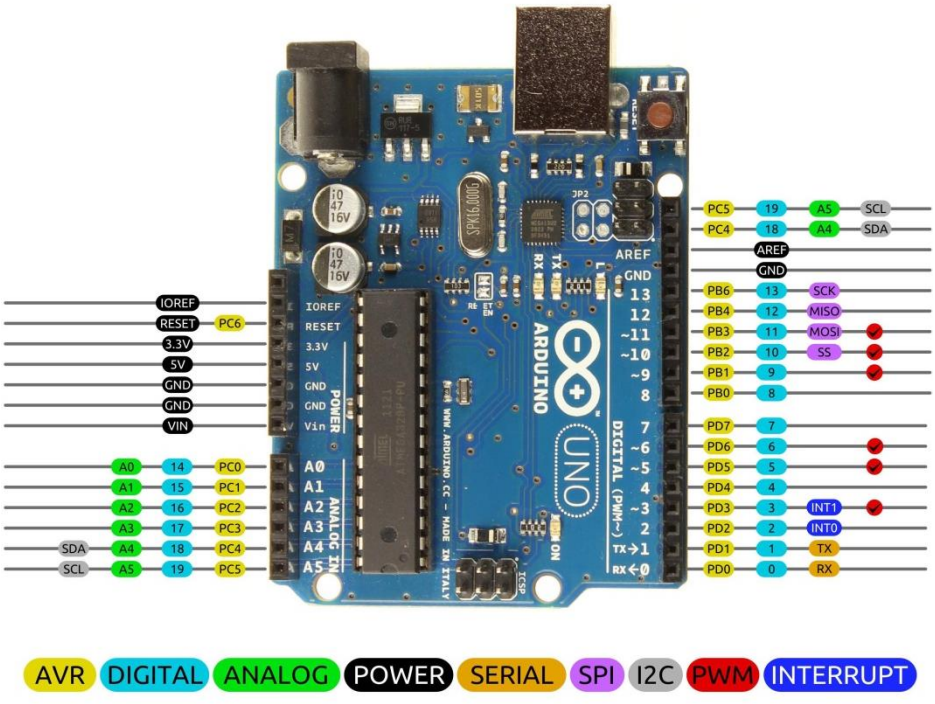

Ilustración 13 - Distribución de pines en Arduino Uno (Arduino.cc, 2014)

3.1.2.2. Arduino Pro Mini 5V

El modelo Pro Mini podría considerarse la miniatura del anterior. En cuanto a características es exactamente igual, utilizando el mismo chip. La principal diferencia es la ausencia del chip ATmega16U2, con lo cual para programarlo es necesario el uso de un FTDI USB-to-serial. Aunque su principal ventaja es su tamaño, en torno a 1/6 parte del tamaño de Arduino UNO. Otra ventaja es que se vende preparado para alimentar a 5V, y a 3.3V, estando el reloj de este último limitado a 8 Mhz, de forma que se pueda utilizar otro tipo de alimentación.

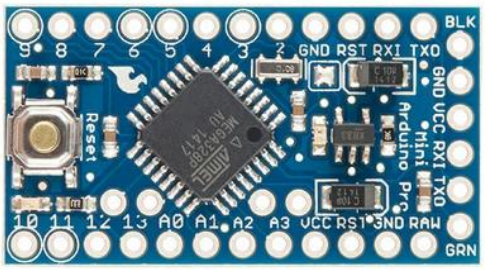

Ilustración 14 - Arduino Pro Mini (Arduino, 2016)

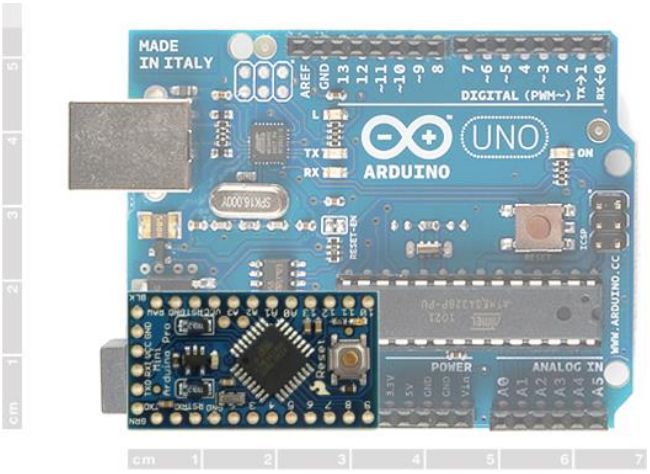

Ilustración 15 - Comparativa Arduino UNO vs Pro Mini (Sparkfun, 2016)

3.2. Comunicación

Respecto al envío y recepción de datos, se utilizarán dos tipos principales de comunicación, Ethernet por parte de Arduino para enviar datos a Raspberry Pi, y radiofrecuencia a 2.4GHz para relacionar distintos Arduino entre sí. Las hojas de datos de los chips se encontrarán en una carpeta denominada *Datasheets* dentro del directorio *Documentos* del CD del presente TFG.

3.2.1. Ethernet

La comunicación tanto con el controlador como con la puerta de enlace se hará mediante Ethernet, para lo cual es necesario añadir un chip que permita dicha comunicación. Las principales alternativas de bajo coste son el chip ENC28J60 de Microchip y el WIZnet W5100.

3.2.1.1. ENC28J60

Para establecer la comunicación, el primer candidato y más barato (apenas 2€ en algunas tiendas) fue el ENC28J60 de Microchip. Se vende normalmente integrado en una placa con el regulador de alimentación y LED de información, normalmente listo para conectar a placas como Arduino directamente. La comunicación con Arduino se realiza a través de SPI. Sin embargo, su bajo precio conlleva una peor gestión de la memoria, lo cual implica un mayor uso de memoria en Arduino y por tanto menos memoria para el resto del programa. No obstante, el hecho que llevó a su descarte fue la inestabilidad que generaba su uso, al no poder mantener una comunicación firme con el dispositivo.

Ilustración 16 - Placa con ENC28J60 (Imarh.ru, 2016)

3.2.1.2. WIZnet W5100

El segundo candidato a probar era el chip W5100 de WIZnet, normalmente vendido en formato “shield”. Su precio es de unos 8€, bastante superior al anterior, no obstante, incluye ranura para tarjeta micro SD, y su formato “shield” permite conectarlo mucho más rápido y sin cableado de por medio.

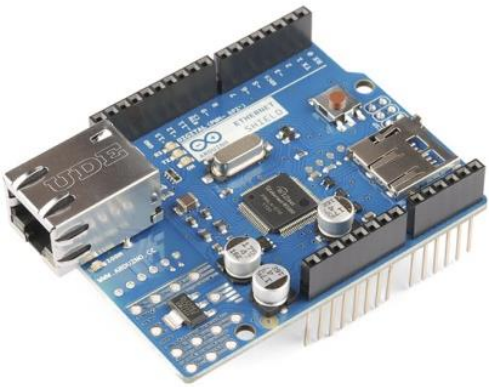

Ilustración 17 - Shield Ethernet WizNET (PatagoniaTec, 2016)

Es precisamente este formato de tarjeta el que permite una conexión directa con Arduino UNO, como se puede observar en la Ilustración 18. Además, una buena gestión por parte de las librerías y del chip, permiten un programa de menor tamaño en Arduino, lo que conlleva un mayor espacio para el programa del usuario.

Tras realizar las conexiones y probar el código se pudo comprobar una estabilidad muy superior al chip ENC28J60, al no aparecer desconexiones, ni fallos de comunicación ni reinicios, problemas bastante frecuentes en su hermano barato. Asimismo, este menor consumo de memoria permite la incorporación de #DEBUG de una de las librerías más utilizadas, facilitando por tanto la programación.

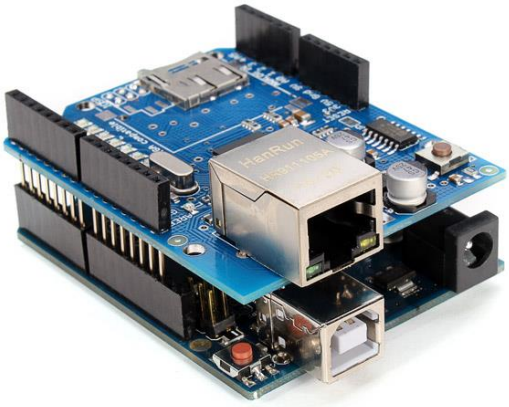

Ilustración 18 - Conjunto Arduino UNO & Shield Ethernet

3.2.2. Radiofrecuencia

Para poder establecer una comunicación inalámbrica entre distintos nodos del sistema, será requisito utilizar una tecnología capaz de enviar, con la suficiente velocidad, una moderada cantidad de datos a una gran distancia. Para distancias superiores a 10m e inferiores a 1km, el mejor candidato es la radiofrecuencia a 2.4GHz. Para ello utilizaremos el chip nRF24L01+ de Nordic Semiconductor.

3.2.2.1. Nordic nRF24L01+

El Nordic nRF24L01+ es un transceptor altamente integrado de muy bajo consumo, para la banda de 2.4GHz. Con corrientes de pico de TX/RX inferiores a 14mA, y modos de *sleep* con consumos del orden de microamperios, se convierte en una de las mejores soluciones de bajo coste para proyectos que requieran una alta autonomía (Nordic Semiconductor ASA, 2015). En cuanto a la alimentación, este módulo trabaja entre 1.9V y 3.6V, pero teniendo la gran ventaja de ser sus pines SPI compatibles con 5V, por lo que no es necesario un circuito intermedio que regule la tensión para comunicar con Arduino, pero sí uno para bajar la tensión para su alimentación.

Ilustración 19 - nRF24L01+ con antena

Respecto a su precio, se pueden encontrar placas preparadas para conectar una antena, (ver Ilustración 19) por un precio aproximado de 2-3€.

Ilustración 20 - Pines nRF24L01+ (MySensors.org, 2016)

Como antena, para el *Gateway* se utilizará una similar a la que aparece en la Ilustración 19, es decir una antena omnidireccional. Para el caso de los nodos se utilizará una StationBox de RF Elements, es decir, un conjunto de carcasa/antena unidireccional, el cual permite hueco para incluir la electrónica necesaria. La antena incluida es de 14 dBi y 2.4 GHz, al igual que los módulos de radio utilizados.

Ilustración 21 - StationBox (RFelements, 2011)

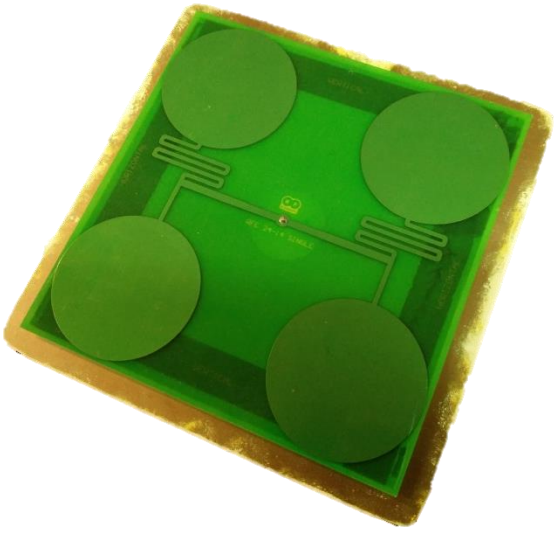

Ilustración 22 - Antena unidireccional 2.4 GHz (RFelements, 2011)

A continuación, se incluye una tabla de pruebas de distancia realizadas con el nodo *Gateway* con antena omnidireccional y un nodo con antena unidireccional enviando y recibiendo respuesta:

Distancia	Paquetes	Perdidos	% pérdidas
50 m	5000	2	0,04
100 m	10000	7	0,07
150 m	10000	12	0,12
200 m	10000	9980	99,8

Tabla 6 - Pruebas de radio con nRF24L01+

Tal y como se observa en los resultados, la distancia máxima obtenida con el módulo de radio es muy inferior a 1 km, anunciado por algunos fabricantes.

3.3. Sensores

Una red de control inalámbrica pierde su funcionalidad si no va acompañada de los sensores necesarios para recopilar la información deseada. Para ello se mostrarán a continuación los sensores utilizados y los conversores A/D requeridos para su uso. Al igual que en el resto de hardware, las hojas de datos correspondientes se encontrarán en la carpeta del CD: /Documentos/Datasheets.

3.3.1. Caudalímetro

La medición del caudal de riego, así como del volumen total de agua regado, será un dato crucial, utilizado en el algoritmo de riego implementado. Para la medición de caudal es necesario el uso de un caudalímetro. Su funcionamiento es similar al de un *encoder*, es decir, ante un volumen de agua responderá enviando una serie de pulsos, indicando dicho volumen.

Ilustración 23 - Caudalímetro PVDF (Clark Sol., 2016)

Para poder realizar esta medición, el caudalímetro hace uso de un emisor/receptor de IR y una turbina. Al pasar el fluido por la turbina, ésta se mueve, cortando el haz de luz y por tanto creando estos pulsos, los cuales se acondicionan y envían a la misma tensión de alimentación (Ver Ilustración 24). El caudalímetro a utilizar será uno de 100000 pulsos por litro, y su rango de trabajo será de 0.06 L/min a 2.0 L/min, alimentado a 5v.

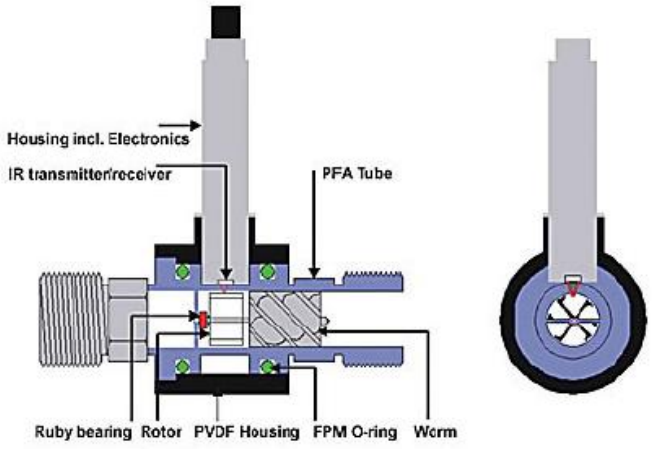

Ilustración 24 - Esquema interior de caudalímetro (Clark Sol., 2016)

3.3.2. Células de carga

El lisímetro que se utilizará para crear la red de control tiene cuatro células de carga, de las cuales tres pesarán la planta al completo, mientras que otra célula pesará el agua drenada. Las tres células principales serán iguales, y de 30 Kg, en cambio, la célula del drenaje será de 10 Kg.

Ilustración 25 - Célula de carga (VETEK, 2016)

Los modelos de células utilizados serán los 108TA de VETEK, con una sensibilidad de 2±10% mV/V, en sus versiones de 10 Kg y 30 Kg. Las resistencias del puente de estas células son de 350 Ω y su carga máxima segura un 150% de su carga nominal, es decir de 10 Kg y 30 Kg.

3.3.3. Conversores A/D

Cuando se trabaja con sensores analógicos, como pueden ser una célula de carga, o la tensión de una batería, es necesario hacer uso de circuitos que sean capaces de "leer" y transformar los valores a señales digitales, entendibles por los micro-controladores utilizados. Para esta transformación, se hace uso de conversores A/D, cuya característica principal es su resolución en bits, es decir, con cuanta precisión pueden transformar un valor analógico a digital.

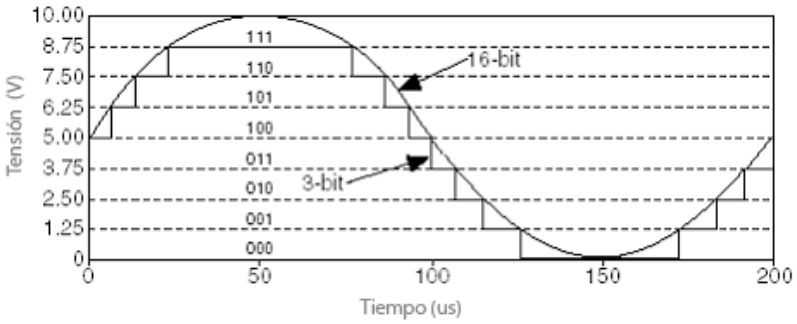

Ilustración 26 - Conversión A/D 3 bits

En casos como el de las células de carga, es necesario ampliar la señal medida antes de realizar la conversión, dado que la variación es tan pequeña, que apenas es apreciable. Para realizar esta tarea existen conversores A/D que incluyen amplificadores de instrumentación, y van por tanto preparados para eliminar el ruido que pueda aparecer y obtener la mejor señal posible.

3.3.3.1. Conversor A/D integrado en ATmega328P

El chip ATmega328P de Atmel, es decir, el controlador utilizado por Arduino, incluye un conjunto de 7 entradas analógicas conectadas a un conversor A/D, embebido en el propio chip.

Éste conversor tiene una resolución de 10 bits, lo cual permite transformar la señal analógica de entrada en 2^10 valores, o lo que es lo mismo, 1024. Por tanto ante una entrada de 0V-5V, tendríamos una resolución de 5V/1024=4.88 mV. Para muchas aplicaciones, esta resolución puede ser más que suficiente, para un sensor de temperatura ambiental, que no requiera una alta precisión, este conversor podría ser suficiente, o incluso demasiado. Sin embargo, el problema llega cuando se requiere medir una variación ínfima, como ocurre en el caso de las células de carga, para las cuales, la resolución de este conversor, queda insuficiente, por lo que hay que explorar alternativas.

3.3.3.2. CS5534-ASZ

Tal y como se ha comentado en el punto anterior, para poder realizar la medición de células de carga con precisión, es necesario el uso de un conversor de mayor resolución, que además permita amplificar la señal del puente de Wheatstone de las células. El candidato elegido ha sido el CS5534-ASZ de Cirrus Logic Inc. dada su amplia capacidad de configuración y sus características. Entre ellas destacan las siguientes:

- Amplificador de instrumentación: Programable, x1 a x64.
- Canales: 4, Multiplexados
- Comunicación: SPI, Microwire
- Banda de rechazo configurable: 50 o 60 Hz
- Distintas configuraciones de alimentación: +5V, ±2.5V, ±3V

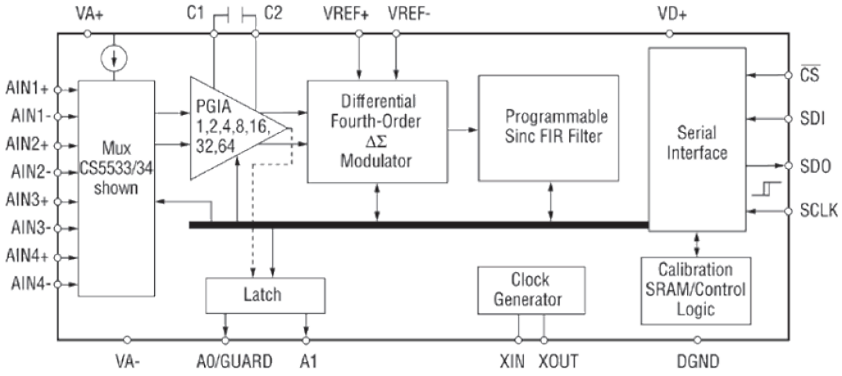

Ilustración 27 - Diagrama funcional CS5534 (Cirrus Logic Inc, 2016)

La elección de este componente como conversor A/D se ha debido principalmente a dos de las características, la gran capacidad de configuración que ofrece, y la posibilidad de leer hasta 4 canales, multiplexados, dado que en los lisímetros utilizados son 4 las células de carga a leer.

En cuanto a desventajas, es importante tener en cuenta la falta de librerías para su uso, por lo que ha sido necesaria la creación de diversas funciones para facilitar su uso.

3.3.3.3. HX711

Por el contrario, el chip HX711 de Avia Semiconductor, dispone de librerías para la mayoría de lenguajes y controladores, sin embargo, la menor posibilidad de configuración, y el hecho de que sólo disponga de dos canales suponen un gran inconveniente, al tener que implementar dos en paralelo para poder leer las 4 células de carga. Conjuntamente, el tener que ampliar el número de pines usados en Arduino implica otro gran reto, al estar todos los pines digitales ya en uso.

Como ya se ha definido, sus características son similares al CS5534-ASZ, teniendo el HX711 algunas ventajas como pueden ser las siguientes:

- Amplificador de instrumentación: Programable, x32 x64 x128
- Banda de rechazo simultanea: 50 y 60 Hz

3.4. Actuadores

Para que cualquier proyecto tenga repercusión sobre el medio que lo rodea, es necesario el uso de actuadores, los cuales se encargarán de transformar las señales eléctricas en movimiento, caudal, temperatura, y un largo etcétera. En el presente proyecto se desea realizar el control de riego de un cultivo, para lo cual será necesario el uso de electroválvulas, que sean capaces de dejar fluir, o cortar el paso del agua. Cabe destacar que la situación especial que genera el hecho de no tener acceso a la red, requiere del uso de electroválvulas de bajo consumo, que puedan desactivarse una vez abiertas o cerradas. Una vez más, las hojas de datos disponibles se encontrarán en el CD presentado, en la carpeta /Documentos/Datasheets.

3.4.1. Electroválvulas 5V

Tal y como ya se ha citado, en las zonas de campo aisladas, no se dispondrá de acceso a la red eléctrica, por lo que es necesario el uso de electroválvulas especiales. Para ello se han buscado aquellas que respondan a las características deseadas, encontrado un buen candidato.

Las válvulas elegidas disponen de ciertas particularidades que las hacen especialmente interesantes para esta aplicación, como son:

- Válvulas a 3 hilos: Neutro, Apertura y Cierre.
- Bajo consumo: Menor a 100 mA tanto a la apertura como el cierre.
- Consumo nulo sin alimentación: Permite mantener las e.v. sin consumo.
- Alimentación a 5v: Se puede alimentar junto al resto de componentes.
- Válvulas motorizadas: Menos probabilidad de fallo.

Ilustración 28 - Electroválvula 5V y 3 hilos (Flow-Controls, 2016)

En cada lisímetro se incorporarán dos de estas electroválvulas, una entre la planta y el depósito de drenaje, y la otra al final del depósito. De esta forma, cerrando siempre una electroválvula antes de abrir la siguiente, controlaremos el agua de drenaje y evitaremos pérdidas.

3.4.2. Relés para E.V. de 230V

Dado que el control de riego se encontrará en una zona con alimentación eléctrica, donde se encuentren asimismo las electroválvulas principales de riego, no es necesario hacer uso de electroválvulas especiales. Por ello al nodo encargado de gestionar estas válvulas se le incorporará un relé que permita comandar cualquier tipo de válvula comercial.

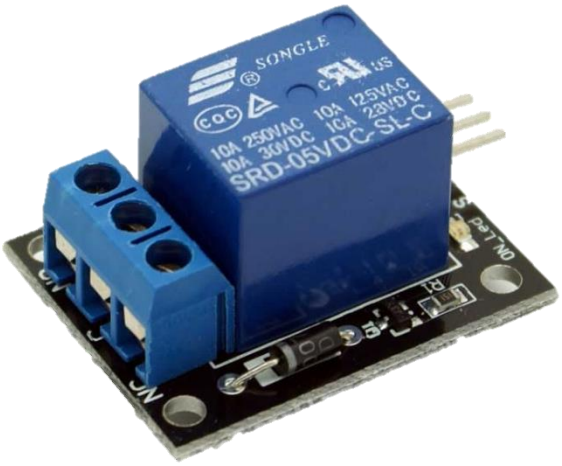

Ilustración 29 - Relé a 5V para cargas de 250V

El relé mostrado en la ilustración anterior viene montando sobre una placa que acondiciona la salida de 5V, de placas como Arduino, para activar cargas de tensiones de hasta 250V y 10A. El circuito permite la conexión "directa" a Arduino, eliminando la necesidad de diseñar un circuito específico para ello, es decir, facilitando el uso de cargas de mayor voltaje. Se pueden encontrar en cualquier tienda online de electrónica y su precio suele ser menor a 1€.

3.4.3. MOSFET Canal P

Si bien no son el actuador en sí, son la pieza que les dará fuerza a los mismos. Con el fin de manejar tanto las electroválvulas de 5V, como de dar alimentación selectiva a las células de carga, es necesario el uso de un interruptor que permita el flujo de corriente directo desde la alimentación hasta su objetivo sin afectar a los puertos del controlador. El encargado de realizar dicha tarea será un transistor MOSFET de canal P, cuyas características permitan una intensidad superior a 100mA, y una tensión umbral baja. El transistor seleccionado será el VISHAY SI2301BDS-T1-E3, cuyas características principales son las siguientes:

- Intensidad Drenador Continua Id -2.4A
- Tensión Drenador/Fuente (Vds) -20V
- Resistencia en Estado ON (Rds) 0.1ohm
- Tensión Vgs de Medición Rds(on) -4.5V
- Tensión Umbral Vgs -950mV
- Disipación de Potencia Pd 900mW
- Diseño de Transistor SOT-23
- Temperatura de Trabajo Máx. 150°C

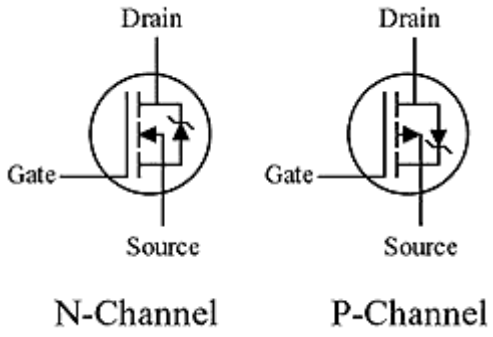

Ilustración 30 - Diagrama MOSFET canal P y N

Las características del transistor elegido permitirán al controlador conmutar su estado, sin comprometer la estabilidad, siendo la tensión umbral superior al intervalo de salida de las señales digitales del ATmega en estado LOW.

Habrá que tener en cuenta, dado que el MOSFET es de canal P, que dependiendo del circuito que se aplique, se invertirá la lógica de salida.

3.5. Alimentación

Como ya se ha referenciado anteriormente, el sistema tendrá que cumplir ciertas condiciones, que le permitan trabajar en un entorno específico. Entre ellas destaca la necesidad de una autonomía alta, y que permita, asimismo, la recarga de las baterías, para poder considerarse un sistema completamente autónomo. Por ello es necesario establecer tanto las baterías, como las tensiones a las que trabajará nuestro sistema, y si es posible, realizar una valoración del consumo para obtener una estimación de la autonomía de dicho sistema.

3.5.1. Consumos

Para poder establecer los requerimientos de nuestra alimentación, es necesario tener una estimación del consumo de nuestro diseño. Para ello se ha conectado el hardware y se han realizado pruebas para el *Nodo Sensor*, es decir, aquel que debe ser autónomo. Los datos más importantes serán el consumo diario medio de corriente y la corriente máxima demandada por el circuito.

Las pruebas y sus resultados han sido los siguientes:

- Corriente en apertura/cierre de las electroválvulas: 100 mA
- Corriente en modo sueño del conjunto: 6 mA
- Corriente en modo normal: 16 mA
- Corriente en modo lectura/envío de datos: 60 mA

Dado el esquema del programa, no hay momentos en los que se den de forma simultánea el modo lectura/envío de datos, con la apertura/cierre de las electroválvulas, por lo que el consumo máximo del circuito se estima en torno a los 116 mA.

Estimando una corriente media, según la duración de los consumos, tendríamos una corriente media de unos 20 mAh, considerando 4 riegos diarios. Por lo que con una

batería de 5000 mAh obtendríamos una autonomía de unos 4 días (sin carga solar). Este consumo se puede disminuir cambiando los modos de sueño del conversor, y eliminando componentes innecesarios dentro de la placa, como el regulador de tensión incorporado en Arduino, el cual no se utiliza, o los LED informativos de los componentes.

3.5.2. Baterías

Teniendo en cuenta el consumo del nodo, hay que establecer los requerimientos de autonomía del sistema. Considerando que su única forma de carga será la energía solar fotovoltaica, se decide implementar una batería con la mayor capacidad posible, y que quepa dentro del sistema, más concretamente dentro del StationBox ya mencionado, considerando toda la circuitería incluida en su interior.

Dada la forma de la placa en la StationBox quedan dos zonas para la instalación de baterías, cuya situación se puede ver en las ilustraciones a continuación:

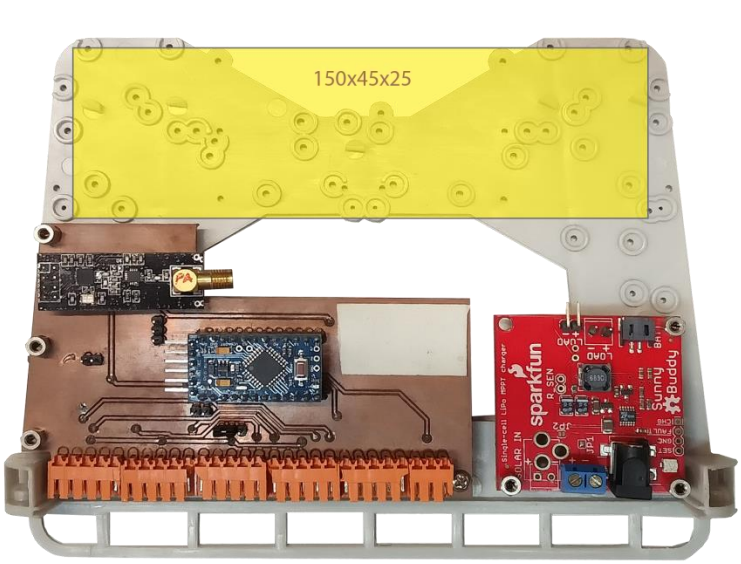

Ilustración 31 - Zona libre para batería. Opción 1.

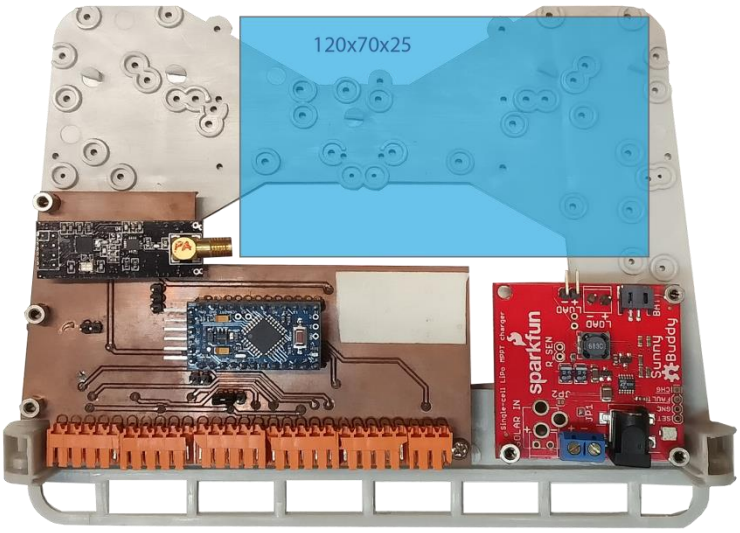

Ilustración 32 - Zona libre para batería. Opción 2.

Como se puede observar, el espacio que queda es bastante amplio, considerando el tamaño de la antena. Esto nos permite incluir baterías de hasta 6000 mAh. Para realizar las pruebas se ha utilizado una batería de 1530 mAh y una celda, probando simplemente que el circuito de alimentación funciona correctamente. La alternativa elegida para la instalación final, tendrá que ser una batería que soporte varios días sin carga alguna, y que permita además una carga rápida para poder reestablecer su autonomía. La batería seleccionada será una *Turnigy 5000mAh* de una celda y una capacidad de descarga 20C. Su alta capacidad en una batería Li-Po, permitirá tanto una elevada autonomía como un alto nivel de carga[2], de forma que con una placa solar de cierta potencia se recarga la batería a la par que se alimenta el circuito.

Las dimensiones de la batería son 128 x 42 x 10mm, por lo que se ajustaría perfectamente en la opción 1 de distribución.

[2] En baterías Li-Po, la corriente de carga recomendada está entre 0,5C y 1C, siendo C la capacidad de la batería en mAh.

Ilustración 33 - Turnigy 5000mAh 1S 20C Li-Po (HobbyKing, 2016)

3.5.3. Placa Solar Fotovoltaica

Como ya se ha descrito, es requisito indispensable permitir la carga de baterías sin conexión a la red eléctrica, y para ello la mejor solución es la energía solar fotovoltaica.

3.5.3.1. Conceptos básicos ESF

Antes de continuar, es necesario definir algunos de los conceptos más importantes de la energía solar fotovoltaica y su aprovechamiento con paneles solares.

La energía solar fotovoltaica se produce mediante generadores fotovoltaicos, compuestos por células solares. Para conseguir una conversión eficiente de energía, es necesario orientar e inclinar las placas de forma adecuada. Normalmente los cálculos y mediciones de parámetros en placas fotovoltaicas se realizan con condiciones estándar (CEM), por lo que los parámetros reales cambiarán según el clima y la estación del año. Dependiendo además del material constructivo de la célula, se pueden obtener eficiencias desde un 6% hasta un 18%, siendo de menos a más eficientes las de silicio amorfo, silicio policristalino y silicio monocristalino.

Para conocer bien el funcionamiento de las placas solares fotovoltaicas, es necesario comprender las curvas características de las placas solares. La curva más importante para las células será la curva I-V, que denotará la respuesta, en condiciones estándar, de la potencia de una placa solar fotovoltaica. Cabe destacar que la corriente se refiere a la corriente en cortocircuito de la placa, mientras que la tensión se refiere al voltaje en circuito abierto entre los contactos de la misma. A continuación se puede observar un ejemplo de curva I-V para la placa solar utilizada durante el desarrollo del presente trabajo para distintas radiaciones.

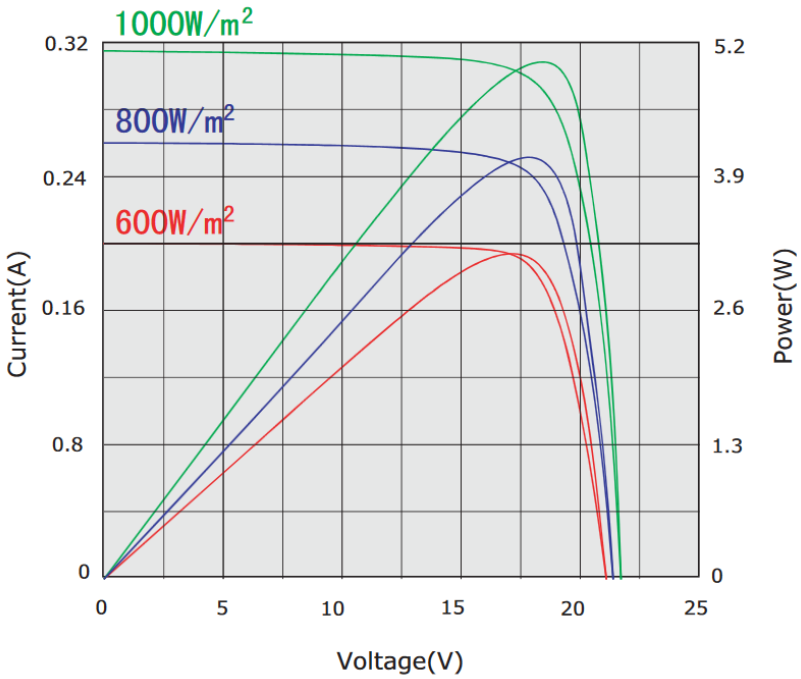

Ilustración 34 - Gráfica I-V. ET-M53605 (ET Solar, 2015)

Como se puede apreciar en la ilustración, la potencia dependerá directamente de la irradiancia recibida. Sin embargo. Se puede observar también que, dependiendo del voltaje al que trabaje la placa, ésta podrá ofrecer una mayor o menor intensidad. El punto en el cual la potencia es la mayor, en torno a 18V en este ejemplo, se denomina Punto de Máxima Potencia (MPP en inglés). Con objeto de mantener la placa en su MPP, los controladores de carga conectados a éstas suelen incorporar una tecnología denominada MPPT, cuya función es la de intentar mantener la tensión de la placa lo más cercana posible a su valor en MPP, obteniendo así la mayor potencia posible.

Otros factores, como la temperatura, influyen sustancialmente en la potencia, reduciendo de forma significativa esta última con el aumento de la primera. Además, para evitar un consumo de corriente de la placa durante los momentos de "sombra", éstas incorporan diodos de Bypass, cuya función es evitar la corriente en dirección a la placa.

3.5.3.2. Placa durante el desarrollo

La placa que se ha utilizado durante el desarrollo del presente trabajo, ha sido la ET-M53605 de ET Solar, cuyas características se detallan a continuación[3]:

- Pmax: 5 Wp
- Voc: 21.96 V
- Isc: 0.315 A
- Vmpp: 17.82 V
- Impp: 0.285 A

[3] Valores medidos bajo condiciones estándar (CEM): 1000 W/m², 25 ºC y con espectro AM1.5.

Ilustración 35 - Placa ET-M53605 5W (ET Solar, 2015)

3.5.3.3. Placa para el diseño final

La potencia de la placa utilizada durante el desarrollo es muy superior a las necesidades del circuito. Estando el consumo de nuestro circuito estimado menor a 100 mA con una tensión de 5V, tendríamos una potencia de 0.5W, sin embargo, es requisito indispensable una rápida carga de la batería con el fin de aprovechar al máximo las horas de sol. Aun siendo la autonomía cercana a una semana, no es posible asegurar que el clima mantenga una radiación suficiente durante 7 días. Haciendo uso de una placa de mayor potencia, se podría alimentar al circuito a la vez que cargar la batería de forma muy rápida. Para ello se ha escogido una célula solar de 3.5W de potencia, con las siguientes características:

- Pmax: 3.69 Wp
- Voc: 7 V
- Vmpp: 6 V
- Impp: 0.615 A

Su pequeño tamaño, 210mm x 113mm x 5mm, permitirá colocarla en cualquier sitio sin ser ello un problema.

3.5.4. Sunny Buddy Sparkfun

Como ya se ha explicado en el punto anterior, la tensión a la que trabajará la placa solar fotovoltaica estará por encima de la tensión de la batería y el resto de los componentes. Para acondicionar la tensión será necesario hacer uso de un controlador o sistema que permita seguir el punto de máxima potencia de la placa (MPPT) y disminuya, asimismo, la tensión a los valores adecuados para la carga de la batería. Para ello existen

en el mercado diversos chips, que bien ajustados realizan dicha tarea y proporcionan una salida adecuada a la carga de una Li-Po, no obstante, en este caso aprovecharemos un diseño ya existente de la empresa SparkFun, preparado para placas solares de entre 6V y 20V (MPP) y para baterías Li-Po de una celda.

La placa que se utilizará se denomina Sunny Buddy y, como se ha descrito, es diseño de la empresa SparkFun. Como principal característica, destaca el MPPT, o seguimiento del punto de máxima potencia, que permite un rendimiento óptimo de la placa solar, además, por defecto su corriente de salida viene limitada a 450mA, permitiendo así la carga de baterías a partir de los 450mAh.

Ilustración 36 - Sunny Buddy (SparkFun, 2016)

Como añadido, la placa viene preparada para conectar hasta dos placas, e incluye los conectores más utilizados tanto para la alimentación (Jack) como para la batería (JST).

La regulación del circuito es muy simple: Se conecta la placa solar y con el pequeño potenciómetro se establece una tensión cercana a la Vmpp de la placa, entre los puntos SET y GND. A partir de ese punto, el LT3652 (Chip que utiliza Sunny Buddy) se encarga de mantener el punto de máxima potencia y cargar la batería que se conecte, manteniendo la tensión correcta para la carga de la misma y el MPP en la placa solar.

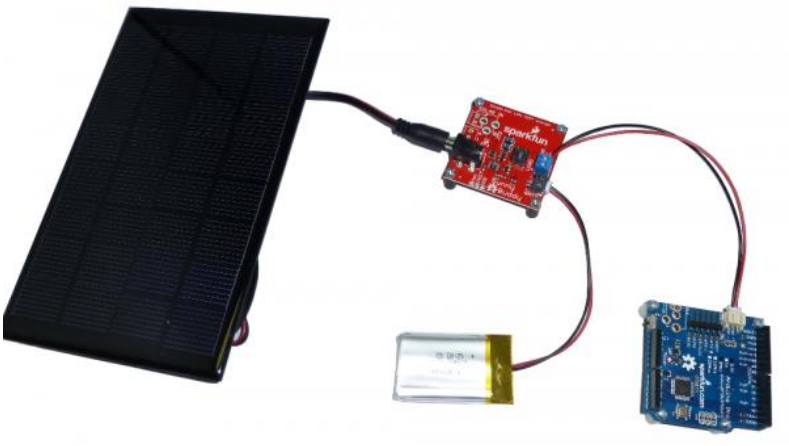

Ilustración 37 - Esquema de conexionado Sunny Buddy (SparkFun, 2016)

3.5.5. Regulador 5V

Dado que las baterías son de una celda, la tensión nominal que ofrecen es de 3.7V, tensión con la que no podemos alimentar ningún componente del diseño. Para poder hacer uso de dichas baterías, es necesario llevar esta tensión hasta 5V, voltaje al que se alimentan todos los componentes, excepto el módulo de radio. Para subir y mantener la tensión de 3.7V a 5V se hará uso de un convertidor DC-DC, y con el fin de disminuir tiempo de diseño y búsqueda de componentes, se recurrirá a un convertidor previamente diseñado, el cual se podrá encontrar en internet por un bajo precio.

El convertidor elegido será un *booster* de 1V-5V a 5V, es decir, ante cualquier tensión de entrada desde 1V hasta 5V, a la salida dará 5V, perfecto para el uso con baterías y pilas de una célula. En cuanto a la intensidad que ofrece, dependerá tanto de la entrada, como del convertidor, pero dado que las baterías Li-Po ofrecen ratios de descarga muy elevados, la limitación la impondrá el convertidor, siendo esta de 600mA, más que suficiente para alimentar todo el circuito durante sus ciclos de mayor consumo.

Respecto al convertidor elegido, cabe destacar su precio, costando este apenas 1,5€, por lo que se ahorra una enorme cantidad de tiempo y esfuerzo comprando esta placa.

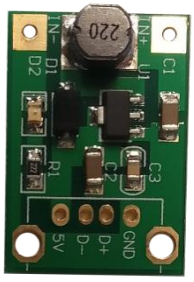

Ilustración 38 - Convertidor DC-DC 5V (DealExtreme, 2016)

Como se puede apreciar en la imagen, está diseñado para conectar a la salida un puerto USB, sin embargo, para el presente trabajo se conectarán cables.

3.5.6. Regulador a 3.3V

Existe una excepción en cuanto a alimentación se refiere. El chip de radiofrecuencia utilizado tiene una tensión de trabajo de 1.9V a 3.3V, consecuentemente, no es posible alimentarlo directamente con los 5V que llegan a la placa. Para proporcionarle tensión será necesario reducir dicha alimentación de 5V a 3.3V, para lo que se utilizará un simple regulador de tensión lineal, un LM1117.

Ilustración 39 - Regulador a 3.3V LM1117

Capítulo 4: Nodos y placas diseñadas

En este capítulo se presentarán los nodos que incorporará el sistema, así como la programación incluida y los circuitos diseñados.

Para el presente proyecto se trabajará sólo con un lisímetro, por lo que se montarán físicamente las partes necesarias para el control de riego de una zona con un lisímetro. No obstante, el programa y diseño que se realicen estarán preparados para la comunicación de dos lisímetros y por tanto la gestión de dos zonas de riego.

Los nodos que se requerirán para la puesta en marcha del sistema serán: Un nodo *Gateway* que haga tanto de bróker MQTT como de centro de comunicación, un nodo de entradas o nodo *sensor* que reciba los datos del lisímetro y los envíe a través de radiofrecuencia, un nodo *actuador* que gestione el riego y por último, un nodo *Servidor* al cual nos conectaremos para programar el riego y visualizar los datos.

4.1. Funcionamiento general del sistema

Antes de comenzar a explicar los nodos de forma independiente, se explicará el funcionamiento global del sistema, de forma que a la hora de leer los nodos se comprenda mejor la información.

El sistema estará compuesto, como ya se ha definido, por varios nodos, dependiendo del número de lisímetros. El mínimo número de nodos será 4, es decir: Controlador, actuador, sensor y *Gateway,* siendo la estructura la que aparece a continuación:

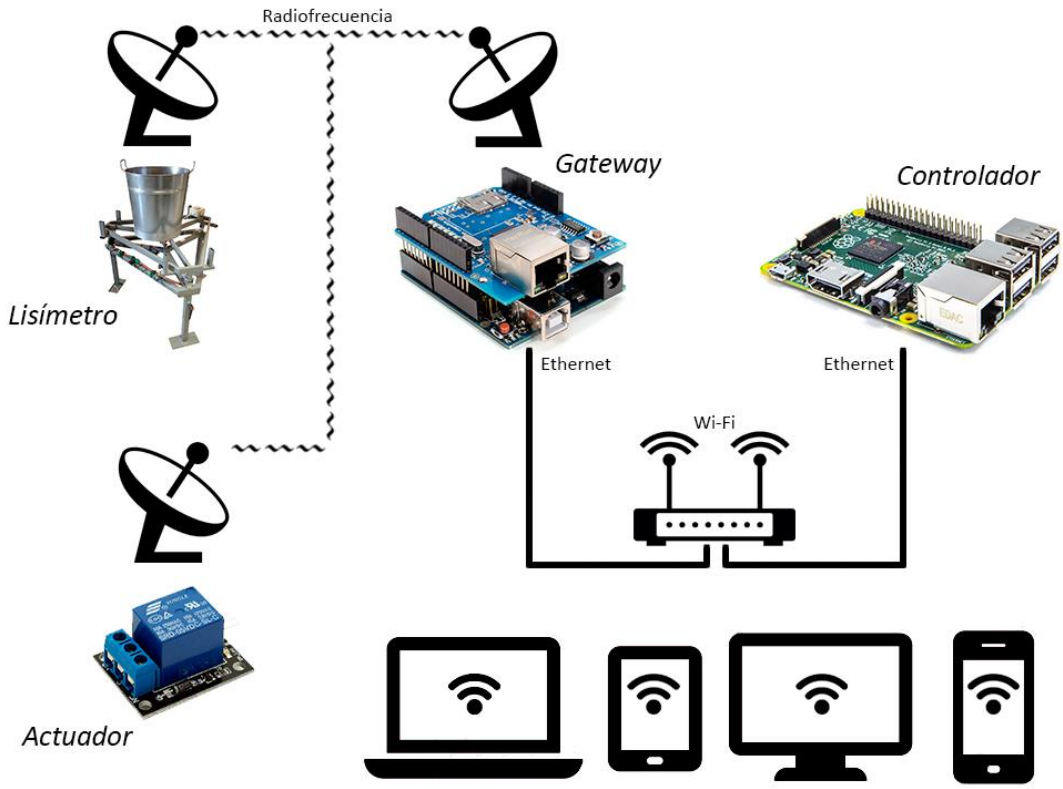

Ilustración 40 - Esquema general del diseño

En cuanto a la gestión de riego, se definirán diversos algoritmos, es decir, no habrá un único modo de riego. Desde el controlador se podrá definir, para cada zona, el modo de inicio de riego y el modo de fin de riego. Los modos son los siguientes:

- **Inicio de riego**
  - Inicio lisimétrico (Disminución de PWFC)
  - Inicio temporizado (Número de riegos y hora del día)
- **Fin de riego**
  - Drenaje (Porcentaje del PWFC)
  - Tiempo (Duración en minutos del regadío)
  - Volumen de agua (regada)

Por tanto, dependiendo de esta configuración, se necesitarán unos parámetros u otros para gestionar el riego. Esta configuración, modificada en el controlador, se enviará al actuador, que establecerá el riego haciendo uso de los parámetros que envíe el sensor. El nodo sensor será "tonto", es decir, no tendrá información ninguna acerca de los otros nodos, dado que para ahorrar batería dormirá siempre que pueda. En cuanto al orden de arranque será importante, dado que facilitará las conexiones entre los componentes, y deberá ser el siguiente:

Gateway → Controlador → Sensor → Actuador

4.2. Nodo *Gateway*

El nodo *Gateway* será el más importante de todos, encargado de gestionar la comunicación entre los demás nodos y de llevar a cabo las transformaciones de datos de radio a Ethernet. En el presente proyecto, como se ha descrito anteriormente, se hará uso del protocolo MQTT, por lo que el tipo de nodo a utilizar será un *Gateway MQTT* (Ver 2.2.1 y 2.4), es decir, además de centro de comunicaciones y puerta de enlace, hará de bróker MQTT y "traductor" a este protocolo. El nodo será asimismo uno de los más sencillos en cuanto a hardware se refiere. En cuanto a software se utilizará el código proporcionado por MySensors.org, con la única diferencia de añadir un LED de estado.

4.2.1. Diseño hardware

Siendo su función la de gestionar la comunicación por radio y hacer de puerta de enlace entre ésta y Ethernet, es imperativo que el nodo incorpore un módulo de radio y un módulo Ethernet. Por ello estará compuesto por: Un Arduino Uno R3 (Ver 3.1.2.1) donde se programe todo el algoritmo de comunicación y gestión; Un *shield* Ethernet W5100 (Ver 3.2.1.2) que permita la comunicación con el *router* y por tanto con el controlador; Y un módulo de radio nRF24L01+ (Ver 3.2.2.1) con antena omnidireccional, para hacer de "base de comunicaciones", es decir, para permitir el intercambio de información entre todos los nodos.

4.2.1.1. Diagrama de conexiones

Existen ciertas particularidades a la hora de conectar nuestros módulos. En concreto, existe una pequeña traba con la comunicación por SPI (Ver Anexo correspondiente) del módulo de Ethernet de WizNET, la cual tiene conflictos a la hora de compartir dicha comunicación con otros dispositivos. Cabe añadir, que siendo el

módulo de Ethernet de tipo *shield,* no existe la posibilidad de cambiar el conexionado con la placa (Ver Ilustración 18), sin embargo, el módulo de radio sí permite dicho cambio.

El conexionado, teniendo en cuenta los cambios de conexionado del módulo de radio, se observan en la Ilustración 41. En esta se puede observar asimismo un condensador entre la tensión de 3.3V y GND, es decir, entre los pines de alimentación del módulo de radio. Dicho condensador, de 47uF, servirá como desacoplamiento y proporcionará estabilidad a la radio. Si se prescinde de dicho condensador, la comunicación por radio comienza a fallar.

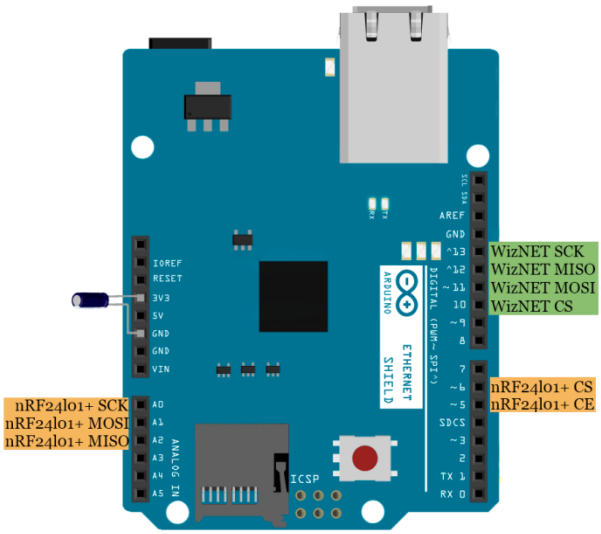

Ilustración 41 - Conexionado *Gateway: Radio y Ethernet.*

Existe, además, la posibilidad de conectar un LED en el PIN 9, el cual indicaría el correcto inicio de la placa, es decir, el correcto arranque de los módulos de Ethernet y radio.

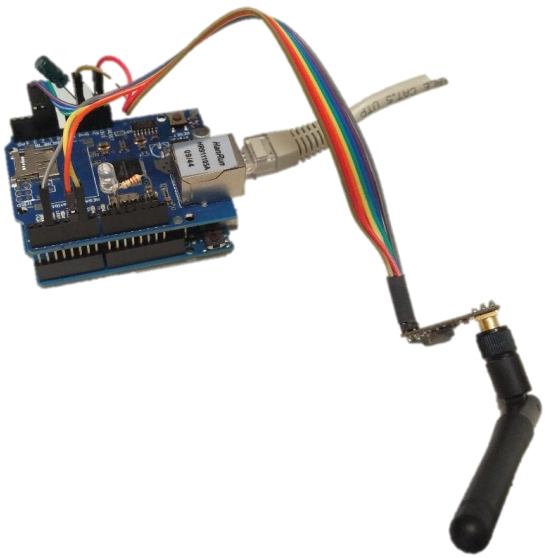

Ilustración 42 - Nodo *Gateway* con radio y LED

En cuanto a otros requerimientos, será indispensable proporcionar a la placa alimentación permanente, y una conexión Ethernet a un *router*, de forma que pueda comunicarse con el controlador.

4.2.2. Programación del nodo

El software que incorporará este nodo será a la par el más complejo y sencillo de implementar. Por suerte, MySensors.org ofrece su código de *Gateway MQTT* de forma libre, por lo que lo podremos usar directamente. El código gestionará todas las conexiones y hará tanto de puerta de enlace entre la radiofrecuencia y Ethernet como de "traductor" a MQTT, es decir, todos los mensajes que se envíen desde la puerta de enlace al controlador, o lleguen desde el controlador, irán codificados siguiendo el protocolo MQTT, permitiendo de esta forma la comunicación con casi cualquier sistema.

Para comenzar a programar es necesario tener presentes los archivos de las librerías de MySensors.org, dado que en ellos aparecerán las configuraciones necesarias para programar de forma correcta los nodos. El código del *Gateway* se encontrará en el directorio *MQTTGatewayWiz* de la carpeta *Programación/Arduino* del directorio raíz del presente TFG.

Sin embargo, antes de comenzar a programar el nodo hay que tener en cuenta algo que ya se ha mencionado, la memoria de Arduino y la comunicación SPI. Dado que el módulo WizNET W5100 no es compatible con la comunicación por radio, es necesario definir una comunicación aparte para esta última.

En primer lugar hay que definir qué archivos de las librerías será necesario modificar. Todos los archivos a modifica se encontrarán dentro de la carpeta MySensors, en el directorio donde se hayan instalado las librerías para Arduino, excepto el archivo MyMQTT.h, que se encontrará dentro del directorio donde se guarde el programa del *Gateway* para Arduino.

El primer archivo a modificar será ***MyConfig.h***, donde habrá que modificar lo siguiente:

- Definir el parámetro que indica una configuración especial de SPI. En concreto habrá que activar *#SOFTSPI*, para ello bastará con eliminar las barras de comentario antes de la línea *#define SOFTSPI*. Es muy importante tener en cuenta que esta configuración será sólo y exclusivamente para el *Gateway* con el *shield* WizNET W5100. **Una vez programado el *Gateway* deberá ser comentado de nuevo. Si antes de programar cualquier otro nodo, no se comenta *#SOFTSPI*, el nodo no comunicará,** dado que intentará conectar con el módulo de radio a través de los puertos incorrectos.

- Si se está utilizando el chip WizNET W5100, activar el *DEBUG* puede ser muy interesante, dado que en los primeros inicios será crucial para comprobar el correcto funcionamiento del sistema. Para ello dejar sin comentar la línea *#define DEBUG*. Es importante destacar que activar el *DEBUG* es un proceso que consume memoria de Arduino, por lo que en casos donde la memoria sea un problema, habrá que desactivarlo. Éste es el caso del chip ENC28J60, el cual al necesitar más memoria, requiere desactivar el *DEBUG,* comentando la línea mencionada.

- En este archivo se deberá configurar asimismo la tasa de envío de datos de los módulos de radio y el canal que utilizan, así como la potencia de envío. En el presente proyecto se han definido los siguientes:

  - *RF24_PA_LEVEL* *RF24_PA_MAX*
  - *RF24_PA_LEVEL_GW* *RF24_PA_MAX*
  - *RF24_CHANNEL* *76*
  - *RF24_DATARATE* *RF24_2MBPS*

Una vez modificada la configuración, será necesario modificar el archivo ***MyMQTT.h***, localizado, como ya se ha descrito, en la carpeta donde se guarde el archivio *.ino* del *Gateway*. En este documento habrá que modificar lo siguiente:

- *MQTT_FIRST_SENSORID* valor a partir del cual se gestionarán ID de forma automática, para desactivarlo poner 255.
- *MQTT_BROKER_PREFIX* nombre del bróker, ha de ser corto y el mismo utilizado luego en el controlador. En este caso será "*ArdMQTT*"
- Otras definiciones como las unidades en métrico o imperial se pueden modificar, pero no será necesario.

Hecho todo esto, basta con implementar el código proporcionado por MySensors.org, modificando ciertas líneas, en el nodo *Gateway,* conectado como se indica en el punto correspondiente. Antes de cargar el programa en el controlador, habrá que modificar lo siguiente[4]:

```
#define TCP_PORT 3000
IPAddress TCP_IP (192,168,1,150);
uint8_t TCP_MAC[] = { 0x02, 0xDE, 0xAD, 0xAA, 0xDE, 0xAE };
```

Estas tres líneas definirán la comunicación por *Ethernet* con el controlador, estableciendo la IP del *Gateway*, el puerto de comunicación, y la dirección física del mismo. Los valores mostrados en el texto anterior son válidos y los utilizados en el presente trabajo. Respecto a la dirección física, su primer byte debe estar en rango local, es decir *x2, x6, xA o xE*, donde *x* puede ser cualquier valor hexadecimal.

En el código utilizado se ha añadido un *LED* informativo, de forma que al arrancar el nodo, éste se encienda, mostrando su correcto arranque. En la programación esto se traduce a la activación del PIN 9 en HIGH al final del *setup();*

4.3. Nodo *Sensor*

Su función principal será la de monitorizar las variables y señales producidas en el lisímetro, es decir, monitorizar el peso de la planta, el drenaje de la misma, y gestionar la apertura y cierre de las electroválvulas de drenaje. Con el fin de realizar un diseño adecuado es necesario establecer los requerimientos que ha de cumplir, estos son:

- Control para dos electroválvulas de 5V
- Alimentación con batería
- Medición de carga de batería

[4] Se mostrará el caso del *Gateway* con el controlador de WizNET.

- Recarga solar con MPPT
- Lectura de 4 canales de células de carga
- Lectura de caudalímetro
- Alto alcance de radio

Teniendo en cuenta los requisitos descritos, es posible comenzar con el diseño del nodo. Para ello nos ceñiremos al tamaño permitido en el interior de la StationBox, de forma que obtengamos un mayor alcance con su antena unidireccional y permita su unión a un mástil alto.

4.3.1. Diseño hardware

Lo primero que hay que diseñar será el circuito que incorporará el nodo, dado que será la base para añadir o eliminar componentes, así como la base para el software. Para ello se ha utilizado la versión gratuita del software *Eagle* de *CadSoftUSA*. Como se describe en el anexo IV, correspondiente a dicho software, existen ciertas limitaciones en cuanto al uso del programa, sin embargo, es suficiente para el presente trabajo.

El diseño de una placa se compone de dos partes: El diagrama de componentes, es decir, el esquema eléctrico con conexiones y cableado, y el diseño de la placa física, con las pistas reales y los componentes ocupando una superficie.

4.3.1.1. Esquema eléctrico

Comenzaremos, como es necesario, con el diseño a nivel electrónico. El diseño lo podremos dividir en varias sub-partes, permitiendo una mejor visualización y mayor facilidad a la hora de diseñarlo.

Lo primera parte y más importante será el controlador, dado que proporcionará la mayoría de entradas y salidas necesarias, así como los protocolos de comunicación utilizados. Para ello utilizaremos la librería de Arduino Pro Mini de SparkFun, pero sustituyendo los *PAD* redondos por unos alargados con más espaciado entre sí, para poder introducir pistas en la siguiente fase del diseño.

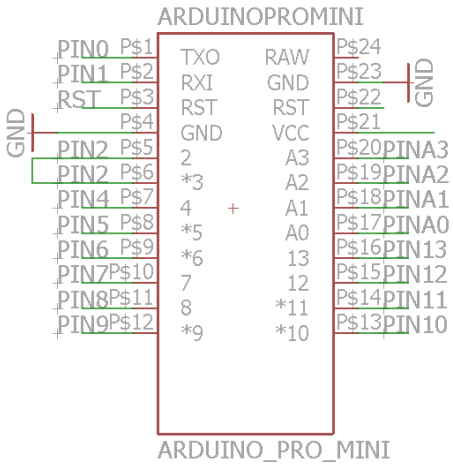

Ilustración 43 - Esquema Arduino Pro Mini .Nodo *Sensor*.

Lo primero que destaca es la conexión entre los pin 2 y 3 del componente. Esta conexión se debe a un inconveniente encontrado durante la fase de programación. Dicha

dificultad consistía en el uso de una misma señal de interrupción para dos funciones completamente diferentes. Las soluciones previstas fueron varias, entre ellas las más evaluadas fueron: Modificar las librerías de MySensors.org con la correspondiente problemática, y la alternativa, utilizar la misma señal digital en las dos interrupciones de Arduino, de forma que se ejecutasen una tras otra. Por suerte, el micro controlador permite el uso de ambas interrupciones al tener un sistema de prioridades, por lo que se implementó la segunda opción.

Una vez definido Arduino dentro del software Eagle y habiendo nombrado sus pines, es posible continuar. El siguiente módulo a conectar será el de radio. En este caso, la conexión de radio puede conectarse a los pines SPI normales de Arduino, al no existir conflicto con el módulo Ethernet (No utilizado en este nodo). De esta forma quedaría así:

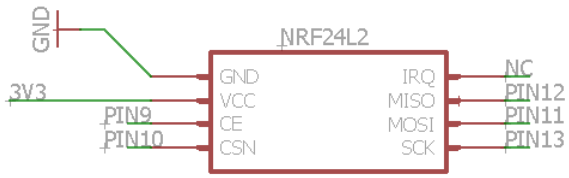

Ilustración 44 - Esquema nRF24L01+. Nodo *Sensor*.

Continuamos con los módulos SPI, dado que su conexión será similar. El siguiente será el conversor A/D seleccionado, el cs5534ASZ, su conexionado se realiza respetando las indicaciones del fabricante. En el conexionado se incorporarán además los conectores para las células de carga:

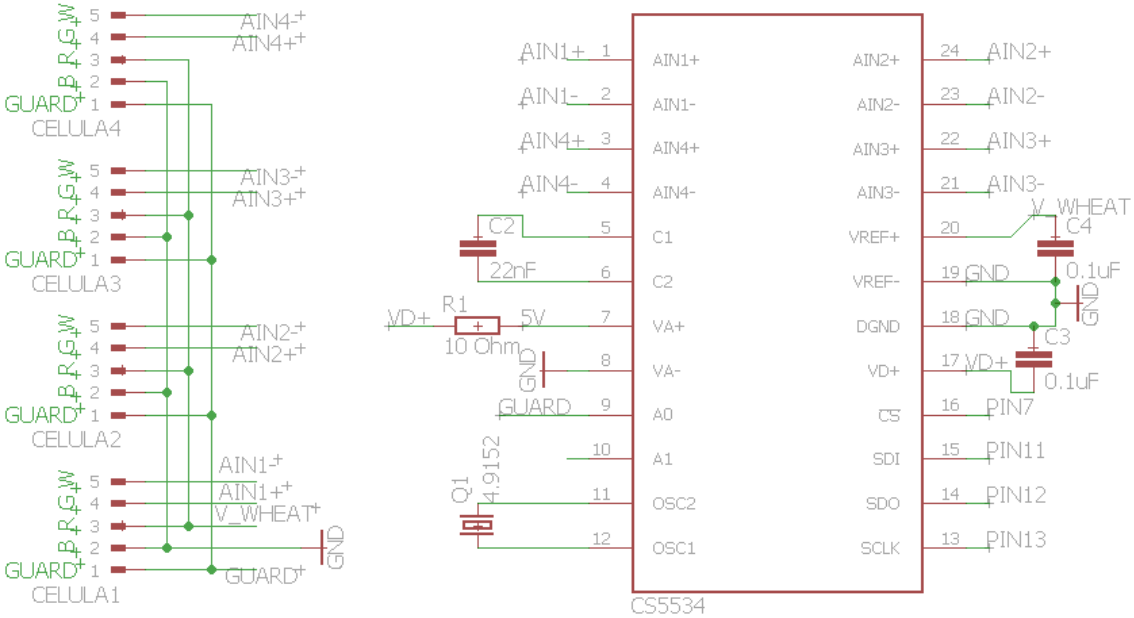

Ilustración 45 - Esquema CS5534ASZ. Nodo *Sensor*.

Dadas las características de la comunicación SPI, el conversor A/D y el módulo de radio compartirán ciertos pines con Arduino. En el esquema anterior es importante destacar la tensión V_WHEAT, que será la tensión de alimentación de las células de carga y por tanto hay que enviarla al conversor A/D como referencia. El resto de componentes son los aconsejados por el fabricante.

Teniendo conectados los módulos SPI, le llega la hora al resto de sensores y actuadores. El primer sensor y más sencillo será la medición de carga de la batería, que constará de un divisor de tensión estabilizado con un pequeño condensador:

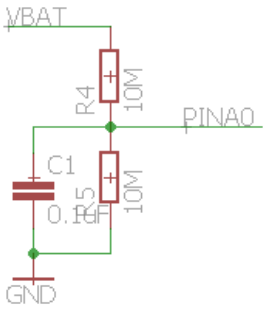

Ilustración 46 - Esquema medición de carga. Nodo *Sensor*.

En cuanto al resto de sensores y actuadores, en el siguiente esquema se puede ver el conexionado de las electroválvulas y el caudalímetro, así como un *jumper* que permitirá seleccionar la alimentación de las células de entre 5V fijos, y 5V controlados por un MOSFET:

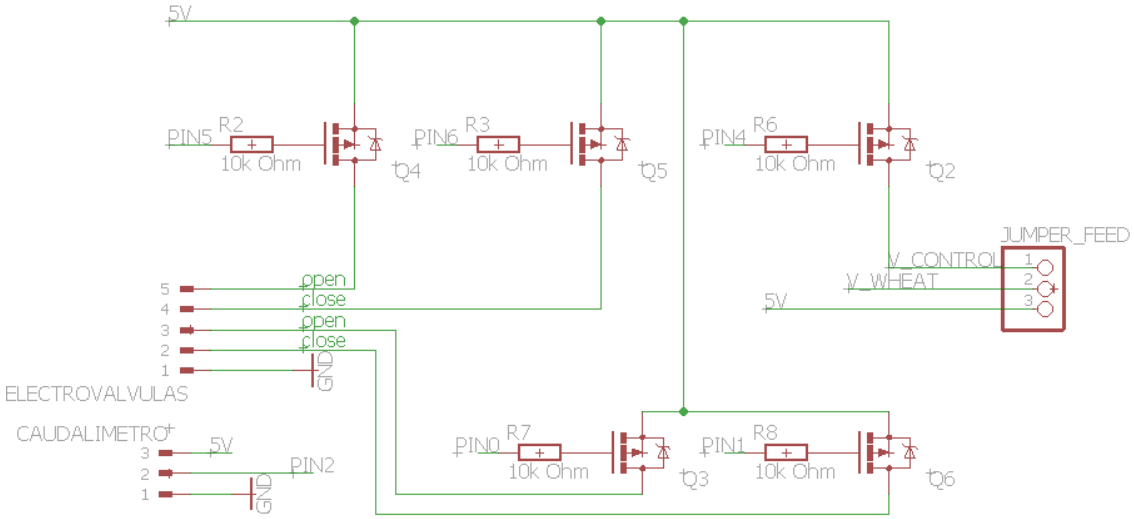

Ilustración 47 - Esquema electroválvulas y caudalímetro. Nodo *Sensor*.

En este esquema se pueden observar además los conectores de las electroválvulas y el caudalímetro, así como la particularidad de la alimentación, que al establecer un HIGH en los pines de los MOSFET obtendremos LOW en la salida, y viceversa. Esto será un aspecto importante a tener en cuenta en la programación. El caudalímetro además irá alimentado siempre a 5V, dado que será el que proporcione las interrupciones al PIN2, como se aprecia en el diagrama.

En cuanto a sensores, se ha decidido dejar acceso a tres de los pines analógicos de Arduino, dado que, aun teniendo sólo 10 bits de resolución, pueden ser útiles en un futuro para añadir otros sensores como sondas de temperatura o sensores de humedad.

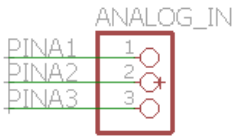

Ilustración 48 - Pines analógicos Arduino. Nodo *Sensor*.

Por último faltaría la alimentación, es decir, la conexión de la batería, los reguladores y convertidores de tensión y la conexión de la placa solar fotovoltaica. En el esquema inferior se puede apreciar el circuito de regulación de 5V a 3.3V para la alimentación del módulo de radio, y dos conectores para la batería y los 5V de alimentación:

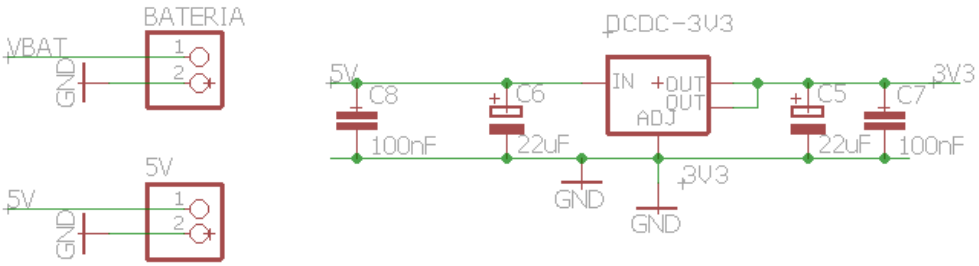

Ilustración 49 - Esquema de alimentación. Nodo *Sensor*.

Sin embargo, en dicho esquema falta la conversión de la tensión de la batería a 5V, y la conexión de la placa solar a su circuito controlador y a la batería. Dado que estas conexiones se realizarán con placas atornilladas y con cables aéreos, no se han incluido en el esquema de *Eagle*. No obstante, su conexión se puede ver en la ilustración a continuación:

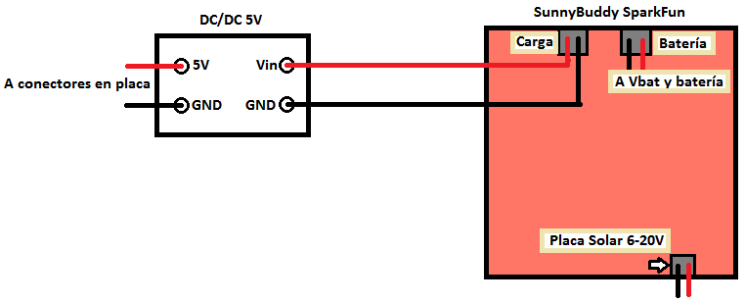

Ilustración 50 - Conexionado SunnyBuddy y DC/DC 5v. Nodo *Sensor*.

Con esto quedaría el circuito completamente conectado y listo. Las células conectadas al conversor A/D, las electroválvulas comandadas por los MOSFET, el caudalímetro como interrupción de Arduino, etc. Quedaría por tanto el diseño a nivel físico.

Como resumen, el pinout de Arduino quedaría como sigue:

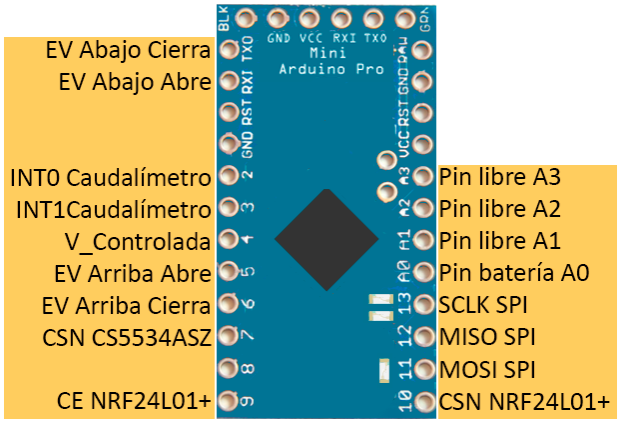

Ilustración 51 - Pinout Arduino Pro Mini 5V. Nodo *Sensor*.

Como se puede observar, los pines 0 y 1 de Arduino serán utilizados, por lo que habrá que deshabilitar la comunicación serie una vez programado (Ver apartado de programación).

4.3.1.2. Distribución física de los componentes

Con el fin de ocupar el menor espacio posible dentro de la StationBox, se diseñará la placa intentando ajustarla a la forma permitida. Las dimensiones aproximadas de la misma se pueden ver en la ilustración siguiente:

Ilustración 52 - Dimensiones aproximadas StationBox. Nodo *Sensor*.

Dado que en la zona más interna de la estructura no existe la posibilidad de atornillar componentes, se hará uso de la zona más exterior, ajustando la forma del

diseño a ésta. Habrá que tener en cuenta, asimismo, el volumen ocupado por la placa SunnyBuddy, dado que su tamaño ocupará cerca de 1/6 del espacio libre.

Como idea de diseño se ha incorporado el tamaño aproximado de la placa SunnyBuddy al esquema anterior, visualizando mejor el espacio restante para el circuito.

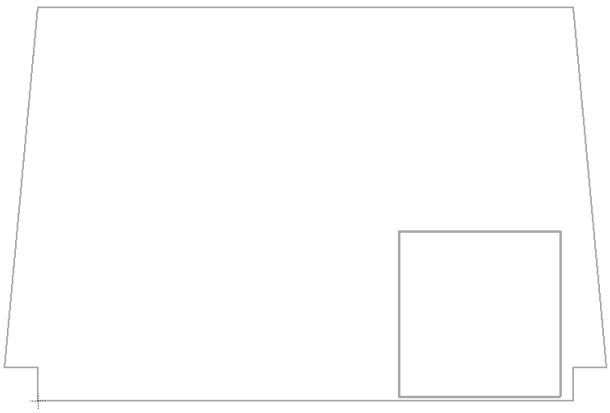

Ilustración 53 - Tamaño SunnyBuddy en StationBox

Con el área restante se diseñará el posicionado y conexionado de los componentes. Con el fin de minimizar el espacio ocupada por los componentes se colocarán todos los elementos de inserción en la capa superior (quedando la soldadura por la inferior) y los componentes SMD por la capa inferior, es decir, dejando todas las soldaduras posibles en la capa inferior.

Una vez colocados los componentes de ambas capas y realizado el ruteado de las pistas, las capas quedan como sigue:

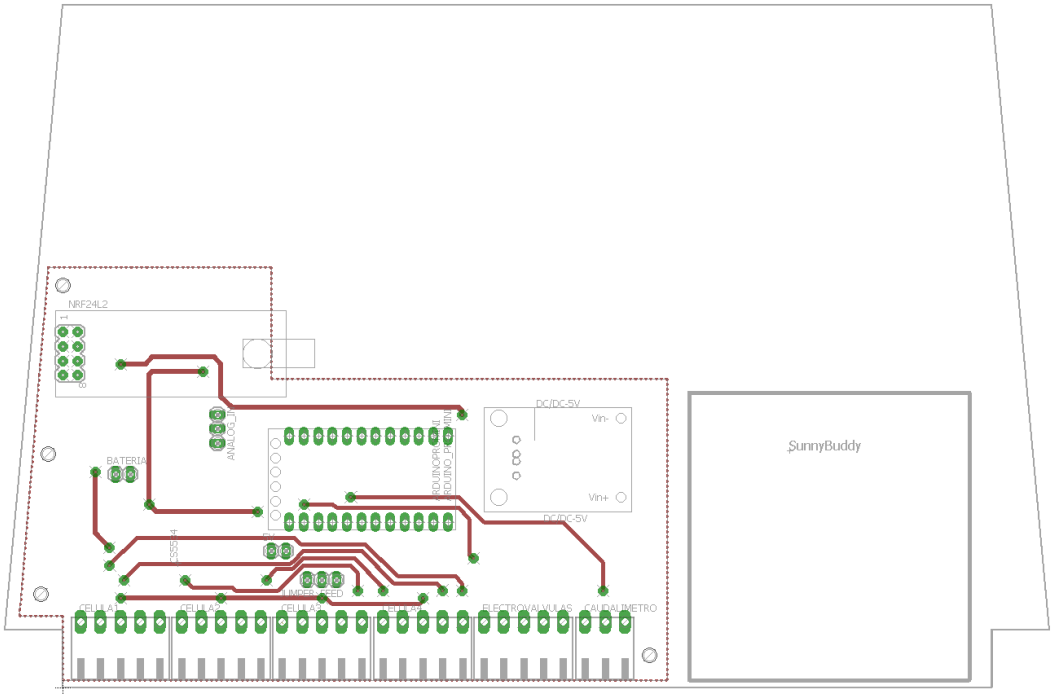

Ilustración 54 - Capa superior Eagle. Nodo *Sensor*.

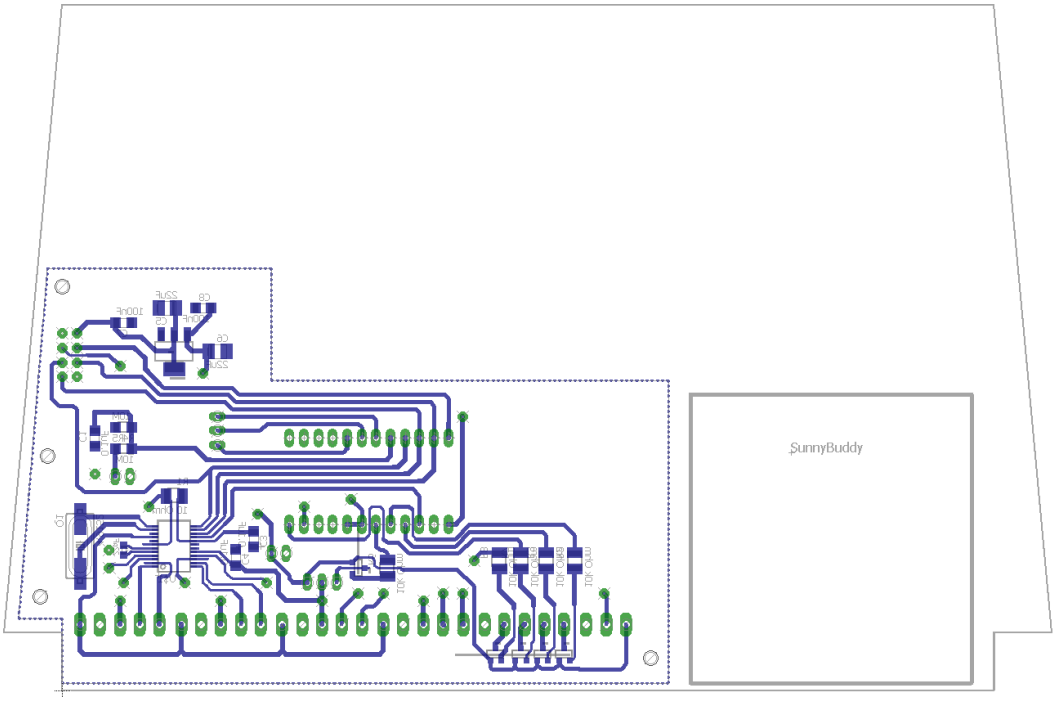

Ilustración 55 - Capa inferior Eagle. Nodo *Sensor*.

Como se puede apreciar en las imágenes anteriores, al mantener todas las soldaduras en la capa inferior se aprovecha de forma excepcional la superficie, permitiendo liberar la capa superior y por tanto dejar superficie para vías y puentes.

En la imagen a continuación se pueden ver ambas capas con más detalle:

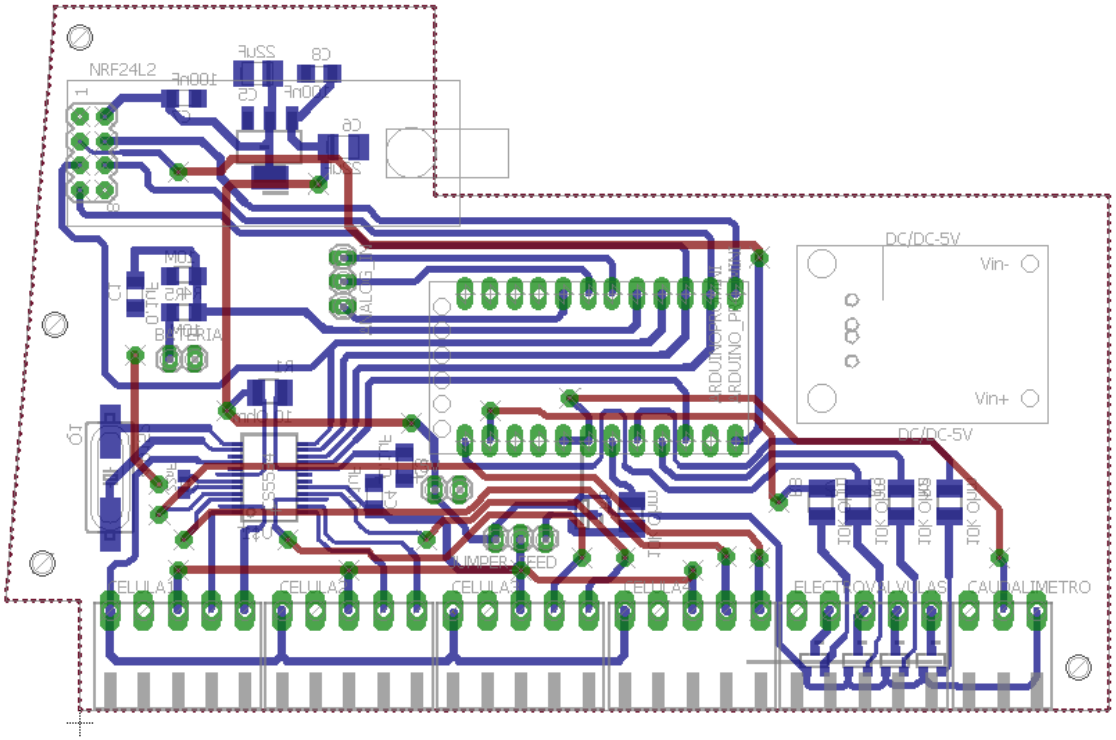

Ilustración 56 - Detalle de ambas capas Eagle. Nodo *Sensor*.

Como se puede apreciar, el conversor DC/DC 5V queda sobre la placa de cobre, por lo que no aparece en el diseño eléctrico pero si en el *layout* de los componentes. Para evitar cortocircuitos, se unirá a la placa mediante un sellado aislante. Además se han preparado los orificios necesarios para adherir la placa a la StationBox correspondiente.

En el caso de Arduino y el módulo de radio, en lugar de soldar los componentes, se soldarán tiras de pines macho-hembra, de forma que se puedan sustituir los componentes en caso de ser dañados o en alguna actualización. Además este diseño permite un gran hueco en la parte superior para incorporar una batería de gran tamaño. Tal y como se define en el apartado correspondiente, se incorporará una batería de 5000 mAh.

Los archivos del diseño se encontrarán en la carpeta *Archivos Eagle* del presente trabajo de fin de grado. En dicha carpeta se encontrarán asimismo los *Gerbers* para la micro-fresadora utilizada.

4.3.2. Programación del nodo

Siendo el nodo más aislado, es indispensable gestionar algoritmos adecuados de funcionamiento y reposo, que permitan la cantidad adecuada de datos, así como una larga duración de la batería. Siendo el objetivo una autonomía en torno a los 7 días, el nodo entrará en modo sueño siempre que pueda.

Con el fin de comprender lo mejor posible el código, éste será analizado por partes. Uno de los fragmentos del código estará dedicado a las variables y funciones necesarias para el control del chip CS5534ASZ. La explicación de dicho código y de las funciones se encontrará en el anexo correspondiente.

Nota: Dado que se utilizan los pines digitales 0 y 1 de Arduino, habrá que deshabilitar la comunicación por serie, y por tanto el DEBUG. Para ello, basta con comentar la línea #define ENABLED_SERIAL del archivo MyConfig.h. En caso contrario no funcionarán correctamente las electroválvulas.

Antes de comenzar con el código, será necesaria una breve explicación de cuál será el funcionamiento general del mismo:

El nodo inicializará y comenzará a tomar lecturas de datos, cada minuto realizará la media de las lecturas y las enviará al controlador. El nodo sabrá cuándo comienza el riego por la interrupción del caudalímetro, por lo que cambiará su frecuencia de lectura de datos hasta terminar dicho riego, definido asimismo por el caudalímetro. Finalizado el riego drenará y enviará nuevos datos al controlador, volviendo a su estado normal.

4.3.2.1. Definiciones

En primer lugar, al inicio del código aparece la inclusión de las librerías necesarias, que en este caso serán cuatro, la propia de MySensors.org, la que permite la comunicación por SPI y dos necesarias para la gestión de tiempo y alarmas:

```
#include <MySensor.h>
#include <SPI.h>
#include <Time.h>
#include <TimeAlarms.h>
```

Una vez incluidas las librerías se procede a definir las constantes y variables relacionadas con MySensor.h que se necesitarán:

```
#define LIS_NODE_ID 1
#define LIS_NODE_NAME "Lisim. 1"
#define LIS_NODE_VER  "v1"
//Definition of children. Essential to check in Openhab.
#define CHILD_CELL_1 1
#define CHILD_CELL_2 2
#define CHILD_CELL_3 3
#define CHILD_CELL_4 4
#define CHILD_VALVE 5
#define CHILD_CAUDAL 6
#define CHILD_VOL 7
#define CHILD_OK  10
#define CHILD_BAT 12
//Definition of children used in Output node.
#define CH_PWFC 20
#define CH_CPW  21
#define CH_CDW  22
```

Por orden, tendremos la ID asignada al nodo junto a su nombre y un texto de versión. A continuación tendremos que definir las ID de los sensores del nodo, para poder crear los mensajes asociados a ellos. Una vez definidos, es turno de los mensajes:

```
MySensor gw;   //MySensor NAME(cepin,cspin);
//Definition of msg to be used.
MyMessage msgCell1(CHILD_CELL_1, V_WEIGHT); //Load Cell 1
MyMessage msgCell2(CHILD_CELL_2, V_WEIGHT); //Load Cell 2
MyMessage msgCell3(CHILD_CELL_3, V_WEIGHT); //Load Cell 3
MyMessage msgCell4(CHILD_CELL_4, V_WEIGHT); //Load Cell 4
MyMessage msgValve(CHILD_VALVE, V_STATUS); //ON/OFF
MyMessage msgCaudal(CHILD_CAUDAL, V_FLOW); //Flow Meter
MyMessage msgVol(CHILD_VOL,V_VOLUME);//Water vol
MyMessage msgPWFC(CH_PWFC,V_WEIGHT);
MyMessage msgCPW(CH_CPW,V_WEIGHT);
MyMessage msgCDW(CH_CDW,V_WEIGHT);
MyMessage msgOK(CHILD_OK, V_VAR1);
MyMessage battrep(CHILD_BAT,V_VOLTAGE);
```

Para definir los mensajes hay que crear primero una instancia de la librería. En este caso la instancia se llamará *gw* (en la primera línea). Hecho esto hay que definir el mensaje para cada sensor. La estructura será siempre la misma, el nombre del mensaje seguido de la ID del sensor asociado y su tipo, entre paréntesis. Recordemos que la lista completa de tipos se puede encontrar tanto en el apartado 2.2 como en la página oficial.

En cuanto a los tipos no es un dato importante, simplemente sirve para recordar que unidades tendrá, pero no afecta al valor enviado.

Las siguientes definiciones serán las propias de Arduino, es decir, los pines de entrada y salida y las variables utilizadas durante el programa:

```
#define PIN_BATTERY A0
#define PIN_CAUDAL 3
#define PIN_BOT_OPEN 1
#define PIN_BOT_CLOSE 0
#define PIN_TOP_OPEN 5
#define PIN_TOP_CLOSE 6
#define PIN_V_WHEAT 4
#define CSN_AD 7

unsigned long SLEEP_TIME = 5000;
unsigned long volumen_agua=0;
float lastbatlvl=0.0;
float newbatlvl=0.0;
float DrenajeAntiguo=0.0;
float DrenajeActual;
int valve_opened=0;
int valve_closed=0;
int IDAlarmaDrenaje=0;
int IDEndDrain=0;
float vol_final=0.0;
float pulsos_caudal=100000.0;
float mean1=0.0;float mean2=0.0;float mean3=0.0;float mean4=0.0;
int cuentaseis=0;
float PotWeightCapField=0.0;
float CurrentPotWeight=0.0;
float CurrentDrainWeight=0.0;
```

4.3.2.2. Funciones auxiliares

Definidas las variables, se puede comenzar con el *setup()* y *void()* del programa, sin embargo, dado que se hará uso de diversas funciones, corresponde explicar primero éstas. La primera función y más utilizada será la función *modo_normal(modo);* que se encargará de leer y mandar los datos durante el día.

```
void modo_normal(int modo){
  int i=0;
    wakeupAD();
    if(modo==0){
      for(i=0;i<2;i++){//2readings
        mean1=mean1+read_CH(1);
        mean2=mean2+read_CH(2);
        mean3=mean3+read_CH(3);
        mean4=mean4+read_CH(4);
      }
    sleepAD();
    cuentaseis++;  //1min
      if(cuentaseis==6){
        cuentaseis=0;
        mean1=mean1/12.0;
        mean2=mean2/12.0;
        mean3=mean3/12.0;
        mean4=mean4/12.0;
        gw.send(msgCell1.set(mean1/1000.0,4));  //Mandamos la media al gateway
        gw.send(msgCell2.set(mean2/1000.0,4));
        gw.send(msgCell3.set(mean3/1000.0,4));
        gw.send(msgCell4.set(mean4/1000.0,4));
        //Mandamos la suma para el inicio lisimetrico (Al nodo outputs)
        gw.send(msgCPW.setDestination(10).set((mean1+mean2+mean3)/1000.0,3));
        mean1=0.0;
        mean2=0.0;
        mean3=0.0;
        mean4=0.0;
      }
    }else{
        mean1=read_CH(1);
        mean2=read_CH(2);
        mean3=read_CH(3);
        mean4=read_CH(4);
        gw.send(msgCell1.set(mean1/1000.0,4));  //Mandamos la media al gateway
        gw.send(msgCell2.set(mean2/1000.0,4));
        gw.send(msgCell3.set(mean3/1000.0,4));
        gw.send(msgCell4.set(mean4/1000.0,4));
        gw.send(msgCDW.setDestination(10).set(mean4/1000.0,4));
        mean1=0.0;
        mean2=0.0;
        mean3=0.0;
        mean4=0.0;
    }
```

Lo primero que realizará será comprobar el modo, dado que si se llama a la función con un parámetro u otro realizará tareas diferentes. En caso de llamarla con un cero, la función realizará una medida de las cuatro células de carga y almacenará el resultado. Al haber realizado 6 medidas, hará la media de los datos y los enviará al controlador. Asimismo calculará la suma de las tres células de la planta y la enviará al nodo *actuador* para su uso, evitando reenvíos innecesarios.

En el caso de pasar un parámetro distinto de cero a la función, realizará una medida de todas las células y la enviará al controlador. Enviando asimismo, al nodo *actuador,* la medida de la célula de drenaje.

La siguiente función a definir será *modo_riego(),* la cual se mantendrá en un bucle *while*, mientras se esté regando. Ésta llamará a *modo_normal(1),* de forma que cada 10 segundos realizará una medida y mandará los datos. La función *modo_riego()*

comprobará el estado del riego, esperando a que éste acabe, y mandando en ese momento la señal de volumen de riego cero.

```
void modo_riego(){
  vol_final=0;
  while(volumen_agua>0){
    modo_normal(1);
      if (calc_caudal()==0){
        vol_final=volumen_agua;
        volumen_agua=0;
        gw.send(msgVol.set(0.00000,5));
        gw.send(msgVol.setDestination(10).set(0.00000,5));
      }
    gw.wait(10000);
  }
}
```

La siguiente función, *checkdrainage(),* se encargará de comprobar el estado del depósito de drenaje, dado que el volumen de ésta es relativamente bajo. Así, si se riega de forma manual o si llueve, el depósito se descargará mandando los nuevos datos a los nodos:

```
void checkdrainage(){
  if(volumen_agua==0){
  wakeupAD();
  float currentdrainageweight=read_CH(4);
  sleepAD();
    if((currentdrainageweight>500)&&(volumen_agua==0)){
      drain_open();
      wakeupAD();
      while((read_CH(4)>20)&&(volumen_agua==0)){
        //sleepAD();
        modo_normal(1);
        gw.sleep(2,RISING,30000);
        //wakeupAD();
      }
      sleepAD();
      drain_close();
      if(volumen_agua==0){
      //Aprovechamos para enviar el PWFC
      wakeupAD();
      float sumaPWFC=read_CH(1)+read_CH(2)+read_CH(3);
      sleepAD();
      float pwfcfinal=(mean1+mean2+mean3)/1000.0;
      gw.send(msgPWFC.setDestination(10).set(pwfcfinal,3));
      gw.send(msgPWFC.set(pwfcfinal,3));
      }
    }
  }
}
```

El algoritmo diseñado para conocer si ha llovido o se ha regado manual comienza leyendo el peso del drenaje, en caso de ser este superior a un valor (500g, ajustable), se comprobará que no se está regando. Si no se está regando drenará hasta terminar, una vez termine enviará los datos del nuevo peso en capacidad de campo a los nodos. Se

considera que ha vaciado cuando el peso de drenaje es menor a 20g (Es posible modificar este valor).

Continuando con el drenaje, una vez termina un riego, se procederá a drenar, cuya gestión lo realizará la función *drenaje_programado()*:

```
void drenaje_programado(){
  //Despues del riego, chequeo periodico de variacion de drenaje.
  do{
    wakeupAD();//Despertamos pra medir el drenaje
    DrenajeAntiguo=read_CH(4);//Leemos el drenaje actual
    sleepAD();
    modo_normal(1);
    gw.sleep(60000);//Duerme 1 min...
    wakeupAD();
    DrenajeActual=read_CH(4);
    sleepAD();
    }while(((DrenajeActual-DrenajeAntiguo)>20)&&(volumen_agua==0));

    drain_open();
    wakeupAD();
    while((read_CH(4)>20)&&(volumen_agua==0)){
      //sleepAD();
      modo_normal(1);
      gw.sleep(30000);
      //wakeupAD();
    }
    sleepAD();
    drain_close();

    //Aprovechamos para enviar el PWFC
    wakeupAD();
    float sumaPWFC=read_CH(1)+read_CH(2)+read_CH(3);
    sleepAD();
    float pwfcfinal=(mean1+mean2+mean3)/1000.0;
    gw.send(msgPWFC.setDestination(10).set(pwfcfinal,3));
    gw.send(msgPWFC.set(pwfcfinal,3));
}
```

Esta función realizará una medida del drenaje y esperará un minuto, al minuto comprobará si la nueva medida es similar a la anterior, lo que significaría que se ha estabilizado el drenaje, es decir, que ha terminado de drenar. Esto lo hace comparando con un valor de 20g, comprobando asimismo si no hay riego. Una vez estabilizado el drenaje, maniobra las electroválvulas y vacía el mismo, considerando el mismo algoritmo, es decir, cuando el peso del drenaje sea menor a 20g. Hecho esto, al igual que en la función anterior, leerá el peso de la planta y lo mandará como PWFC, tanto al controlador como al nodo *actuador*.

La función *calc_caudal(),* utilizada en varias otras funciones, se encarga de calcular el caudal de riego durante un periodo de tiempo (500ms, ajustable), para ello medirá el volumen de agua proporcionado por el caudalímetro antes y después de ese tiempo y dividirá. La función pasará el volumen tanto al controlador como al nodo *actuador* y el caudal sólo al controlador. Para utilizarla como comprobación de riego, la función retornará el caudal medido:

```
float calc_caudal(){
  //Calcula el caudal durante 500ms
  float tiempoA,tiempoB=0;
  float volA,volB;
  tiempoA=millis();
  volA=volumen_agua;
  do{
    tiempoB=millis();
    volB=volumen_agua;
   }while((tiempoB-tiempoA)<500);
   float caudal=(volB-volA)/(tiempoB-tiempoA);
   float caudalenvio=caudal*(0.06/pulsos_caudal);
   float volumenenreal=(volumen_agua/pulsos_caudal);
   //caudal en pulsos/ms, pasar a l/min.
   gw.send(msgVol.set(volumenenreal,5));
   gw.send(msgVol.setDestination(10).set(volumenenreal,5));
   gw.send(msgCaudal.set(caudalenvio,5));
   return caudal;
}
```

La función *incomingMessage()* no será utilizada, pero su objetivo es ser la función ejecutada cuando se recibe un mensaje de radio. Dado que el nodo estará gran parte del tiempo en *sleep*, no recibirá ningún mensaje.

Respecto a la función *sendbatterylvl(),* como su propio nombre indica, enviará el estado de la batería al controlador. Simplemente leerá el pin analógico y realizará el cálculo de la tensión real de la batería, en ese momento enviará el valor.

En cuanto a la función *water_volume(),* será la que se ejecute para medir caudal. Para ello, dado que el caudalímetro envía pulsos de forma asíncrona, es necesario hacer uso de las interrupciones de Arduino. En concreto, ésta será la interrupción de prioridad baja, quedando la de prioridad alta el despertar a Arduino si se riega en modo *sleep*. (Se verá más adelante).

Para la gestión de las electroválvulas, se han creado tres funciones, de forma que se visualice mejor el código en el resto del programa:

```
void drain_open(){
  digitalWrite(PIN_TOP_OPEN,HIGH);
  delay(20);
  digitalWrite(PIN_TOP_CLOSE,LOW);
  delay(5000);
  digitalWrite(PIN_BOT_CLOSE,HIGH);
  delay(20);
  digitalWrite(PIN_BOT_OPEN,LOW);
  delay(5000);
  gw.send(msgValve.set("ON"));
  drain_stop();
}
```

La primera, *drain_open()*, mandará cerrar la electroválvula superior, dándole 5 segundos para hacerlo. Una vez hecho esto y considerando que se ha cerrado, abrirá el drenaje, permitiendo otros 5 segundos de margen. Para terminar mandará la información del drenaje al controlador y quitará las señales con *drain_stop()*.

```
void drain_close(){
  digitalWrite(PIN_BOT_OPEN,HIGH);
  delay(20);
  digitalWrite(PIN_BOT_CLOSE,LOW);
  delay(5000);
  digitalWrite(PIN_TOP_CLOSE,HIGH);
  delay(20);
  digitalWrite(PIN_TOP_OPEN,LOW);
  //toca esperar a que cierre
  delay(5000);
  gw.send(msgValve.set("-"));
  drain_stop();
}
```

Similar a la anterior, pero con la secuencia contraria, *drain_close()*, cerrará la electroválvula de drenaje y hecho esto abrirá la superior, permitiendo que la planta vuelva a drenar sobre el depósito. Al final enviará el estado del drenaje al controlador.

```
void drain_stop(){//DONE
  //Takes feed out from both signals.
  digitalWrite(PIN_BOT_OPEN,HIGH);
  digitalWrite(PIN_BOT_CLOSE,HIGH);
  digitalWrite(PIN_TOP_OPEN,HIGH);
  digitalWrite(PIN_TOP_CLOSE,HIGH);
}
```

Por último, *drain_stop(),* simplemente pondrá a *HIGH* todas las salidas (invertidas por el MOSFET P) para evitar consumos.

El resto de funciones utilizadas pertenecen al conversor A/D y se explicarán en su correspondiente anexo.

Definidas todas las funciones a utilizar, estamos en condiciones de explicar el código de *setup() y loop()*.

4.3.2.3. Función *setup ( )*

Comenzamos, como marca la lógica, por la función *setup()*, la cual se puede dividir en 3 zonas bien marcadas:

```
void setup(){
  pinMode(PIN_CAUDAL, INPUT);
  pinMode(PIN_BOT_OPEN, OUTPUT);
  pinMode(PIN_BOT_CLOSE, OUTPUT);
  pinMode(PIN_TOP_OPEN,OUTPUT);
  pinMode(PIN_TOP_CLOSE,OUTPUT);
  pinMode(CSN_AD, OUTPUT);
  digitalWrite(CSN_AD,HIGH); //CSN AD OFF (1). WON'T LISTEN GW COMMANDS.
  digitalWrite(PIN_BOT_OPEN,HIGH);//VALVE OPEN TO HIGH MEANS NO VOLTAGE.
  digitalWrite(PIN_BOT_CLOSE,LOW);//VALVE CLOSE TO LOW MEANS VALVE CLOSES.
  digitalWrite(PIN_TOP_CLOSE,HIGH);
  digitalWrite(PIN_TOP_OPEN,LOW);
  delay(5000);
  digitalWrite(PIN_BOT_OPEN,HIGH);//VALVE OPEN TO HIGH MEANS NO VOLTAGE.
  digitalWrite(PIN_BOT_CLOSE,HIGH);//VALVE CLOSE TO LOW MEANS VALVE CLOSES.
  digitalWrite(PIN_TOP_CLOSE,HIGH);
  digitalWrite(PIN_TOP_OPEN,HIGH);

  attachInterrupt(digitalPinToInterrupt(PIN_CAUDAL), water_volume, RISING);
```

La primera parte, define los pines como entradas o salidas, según corresponda y manda cerrar la electroválvula de abajo y abrir la de arriba, por si se quedaron mal en un apagado inesperado. En ese momento quita la señal a las electroválvulas y define la interrupción del caudalímetro, es decir, la que nos permitirá conocer el estado del riego y el volumen del mismo.

```
  //First thing to do, as it initializes SPI Comm.
  gw.begin(incomingMessage, LIS_NODE_ID, false);
  gw.wait(20);//Wait until initialization completed.
  gw.sendSketchInfo(LIS_NODE_NAME, LIS_NODE_VER);
  gw.present(CHILD_CELL_1, S_WEIGHT);// Register sensors
  gw.wait(20);//Wait 20 ms to let the msg arrive properly.
  gw.present(CHILD_CELL_2, S_WEIGHT);
  gw.wait(20);
  gw.present(CHILD_CELL_3, S_WEIGHT);
  gw.wait(20);
  gw.present(CHILD_CELL_4, S_WEIGHT);
  gw.wait(20);
  gw.present(CHILD_VALVE, S_LIGHT); //Boolean
  gw.wait(20);
  gw.present(CHILD_BAT, S_MULTIMETER);
  gw.wait(20);
  gw.present(CHILD_CAUDAL, S_WATER);
  gw.wait(20);
  gw.present(CHILD_VOL, S_WATER);
  gw.wait(20);
  gw.present(CH_PWFC,S_WEIGHT);
  gw.wait(20);
  gw.present(CH_CDW,S_WEIGHT);
  gw.wait(20);
  gw.present(CH_CPW,S_WEIGHT);
  gw.wait(20);
  gw.present(CHILD_OK, S_CUSTOM);
```

En la segunda parte, inicializa la instancia creada, teniendo como función de entrada (de radiofrecuencia) la función *incomingMessage(),* como ID del nodo, la

asignada en las definiciones y por último *false,* indicando que no será un nodo repetidor. La definición completa de la función *begin* se encuentra tanto en el código implementado como en la página MySensors.org. Una vez inicializado el nodo, se presenta al controlador, mandando su nombre y versión y posteriormente, todos los sensores asociados a dicho nodo, es decir, los definidos en los parámetros iniciales. La presentación se realiza con S_[Tipo de sensor]. La información se encuentra en el punto correspondiente a MySensors.org.

```
  //FINALIZA SETUP GW || COMIENZA SETUP AD
  Serial.println("Inicio reset AD.");
  resetAD(); //Reset inicial de puerto serie y AD
  Serial.println("Inicio conf AD.");
  confAD(); //Configuración de modo normal
  confSetups();
  Serial.println("");
  sleepAD();//Send the ADC to sleep (low power)
  gw.send(msgValve.set(0));//Mandamos drenaje OFF
  Serial.println("Finalizado el setup");
  gw.send(msgOK.set("OK"));//Mandamos OK
}
```

Por último, en el *setup*, se inicia el conversor A/D y se manda el estado del drenaje y un *OK* al controlador.

4.3.2.4. Función *loop ( )*

A continuación la implementación de la función *loop():*

```
void loop(){

  checkdrainage();

  if(volumen_agua==0){
    Serial.println("Modo Normal");
    Serial.println(volumen_agua);
    modo_normal(0);
    gw.sleep(2,RISING,10000);
  }else{
    Serial.println("Modo Riego");
    modo_riego();
    Serial.println("Drenaje prog");
    drenaje_programado();
  }
    sendbatterylvl();
}
```

Una vez inicializado el nodo y terminada la configuración, el programa entrará en el bucle *loop()*, donde comprobará al inicio el estado del drenaje, y posteriormente, dependiendo de si se está regando o no (interrupción del caudalímetro), entrará en un modo "normal", donde ejecutará la función *modo_normal()*¸ya descrita, y dormirá 10 segundos (excepto interrupción en PIN 2, es decir, caudalímetro). En caso de estar regando, activará la función *modo_riego(),*igualmente descrita, y tras terminarla un drenaje programado.

Tras terminar cualquiera de los modos de lectura de datos, ejecutará la función *sendbatterylvl ()*.

4.4. Nodo *Actuador*

Será el encargado de recibir la programación de riego, y por tanto gestionar las electroválvulas. El *nodo actuador* recibirá datos tanto del *nodo controlador* como de los nodos sensores, proporcionando a este la información necesaria para iniciar o detener el riego.

4.4.1. Diseño hardware

Este nodo estará compuesto por un Arduino Uno R3 y tantos módulos de relés como salidas se quieran controlar. En principio el sistema estará preparado para controlar dos zonas de riego con dos lisímetros, por tanto el nodo tendrá conectados dos módulos de relés (Ver 3.4.2) de 5V. Tendrá asimismo conectado un módulo de radio, que le permita recibir la información necesaria. Al igual que en el caso anterior, no existe un módulo Ethernet que genere conflictos con la radiofrecuencia, por lo que se puede instalar en los puertos SPI de Arduino, recordando siempre incorporar el condensador de 47 uF. En resumen, las conexiones quedarían como sigue:

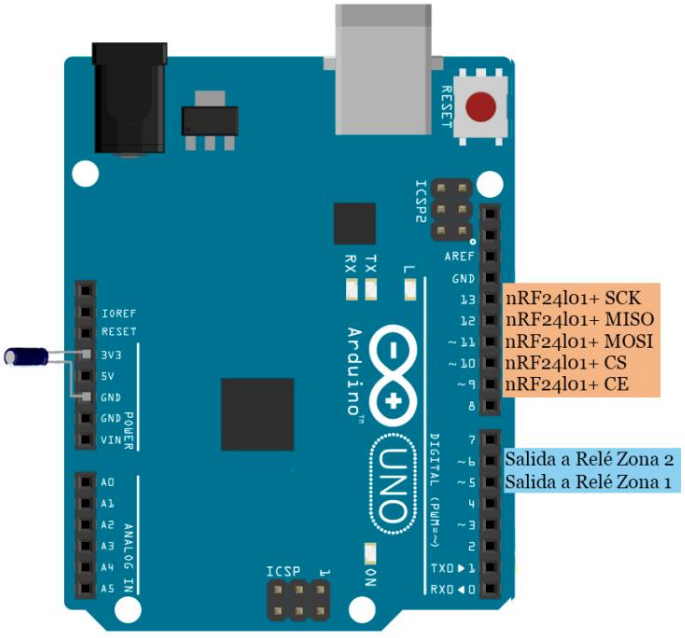

Ilustración 57 - Conexionado del *nodo actuador*.

4.4.2. Programación del nodo

El presente nodo tendrá una de las funciones más importantes de todos, gestionar el riego. Para ello hará uso de los parámetros enviados desde el nodo *sensor* y el *controlador*, y decidirá en qué momento se deberá activar y detener el riego.

El nodo deberá cumplir ciertas condiciones para ser funcional y evitar problemas, como puede ser, la capacidad para recuperar el estado y configuración tras un reinicio. Asimismo deberá conocer el día y hora en todo momento para permitir riegos en momentos determinados.

4.4.2.1. Definiciones

Comenzamos por tanto, incluyendo las librerías que necesitaremos:

```
#include <MySensor.h>
#include <SPI.h>
#include <Time.h>
#include <TimeAlarms.h>
#include <EEPROM.h>
```

Las librerías serán las mismas incluidas para el nodo *sensor*, añadiendo la librería que permite la gestión de la memoria EEPROM de Arduino.

Posteriormente, y al igual que en el nodo anterior, definiremos la ID del nodo, su nombre y los sensores que incorporará:

```
#define LIS_NODE_ID 10 //Nodo de salidas 10.
#define LIS_NODE_NAME "Riego"
#define LIS_NODE_VER  "v1"
//Definition of children.
#define CHILD_EV_1 11
#define CHILD_EV_2 12
#define CHILD_EV_3 13
#define CHILD_EV_4 14

#define CH_ZONE   100
#define CH_MODO_I 101
#define CH_MODO_F 102
#define CH_DIA_I  103
#define CH_HORA_I 104
/* Inicio por tiempo */
#define CH_INT_R  105
#define CH_NUM_R  106
#define CH_HORA_R1  107
#define CH_HORA_R2  108
#define CH_HORA_R3  109
#define CH_HORA_R4  110
```

```
/* Inicio por lisimetria */
#define CH_WRED 111
/* Fin por tiempo */
#define CH_TIEMP_RIEGO  112
/* Fin por drenaje */
#define CH_DRAIN_F  113
/* Fin por volumen */
#define CH_VOL_F  114
/* VALIDACION DATOS */
#define CH_VALIDATE 120

#define CHILD_VALVE 5
#define CHILD_CAUDAL 6
#define CHILD_VOL 7
#define CHILD_OK  10
#define CHILD_BAT 12
```

Los sensores incluidos aquí serán en cierta forma especiales, al no ser ni sensores ni actuadores reales. Estos serán los definidos a partir del 100, y servirán como núcleo de recepción de los datos del controlador.

A continuación, hay que definir la instancia de la librería y los mensajes que enviará el nodo, los cuales serán sólo 5:

```
MySensor gw;   //MySensor COSA(cepin,cspin);

/* MESSAGES TO SEND&RECEIVE */
MyMessage msgEV1(CHILD_EV_1, V_LIGHT);
MyMessage msgEV2(CHILD_EV_2, V_LIGHT);
MyMessage msgEV3(CHILD_EV_3, V_LIGHT);
MyMessage msgEV4(CHILD_EV_4, V_LIGHT);

MyMessage msgOK(CHILD_OK, V_VAR1);
```

Dado que sólo enviará datos relevantes al estado de las electroválvulas, sólo necesitará mensajes para ellas, además de un mensaje de *OK* para el controlador.

Hecho esto, hay que definir las entradas y salidas del nodo, así como las direcciones que se utilizarán para guardar las configuraciones en memoria. De hecho, este nodo no incorporará entradas, sino, sólo salidas. Las direcciones de EEPROM se definirán a partir de la 512, dado que las librerías de MySensors.org utilizan las anteriores. Se definirán asimismo direcciones para las dos zonas o lisímetros controlados, 1 y 2.

```
#define PIN_EV1 5
#define PIN_EV2 6
#define PIN_EV3 7
#define PIN_EV4 8
//Definimos las direcciones eeprom>512
#define ADD_DATA_STORED1  512 //Byte
#define ADD_MODOINICIO1   513 //Byte
#define ADD_MODOFIN1      514 //Byte
#define ADD_WATERTIMEINT1 515 //Byte
#define ADD_NUMRIEGOS1    516 //Byte
#define ADD_HORARIEGO11   517 //Byte
#define ADD_HORARIEGO12   518 //Byte
#define ADD_HORARIEGO13   519 //Byte
#define ADD_HORARIEGO14   520 //Byte
#define ADD_WEIGHT1       521 //Float!
#define ADD_WATERTIMEDUR1 525 //Byte
#define ADD_DRAINWEIGHT1  526 //Byte
#define ADD_VOLTOFIN1     527 //Float!
#define ADD_PWFC1         531 //Float!
#define ADD_PWFC1_SET     535 //Byte

#define ADD_DATA_STORED2  612 //Byte
#define ADD_MODOINICIO2   613 //Byte
#define ADD_MODOFIN2      614 //Byte
#define ADD_WATERTIMEINT2 615 //Byte
#define ADD_NUMRIEGOS2    616 //Byte
#define ADD_HORARIEGO21   617 //Byte
#define ADD_HORARIEGO22   618 //Byte
#define ADD_HORARIEGO23   619 //Byte
#define ADD_HORARIEGO24   620 //Byte
#define ADD_WEIGHT2       621 //Float!
#define ADD_WATERTIMEDUR2 625 //Byte
#define ADD_DRAINWEIGHT2  626 //Byte
#define ADD_VOLTOFIN2     627 //Float!
#define ADD_PWFC2         631 //Float!
#define ADD_PWFC2_SET     635 //Byte
```

Leyendo la serie de direcciones a almacenar, es posible conocer el número de parámetros necesarios para la gestión del riego y entender el significado de la mayoría. Los parámetros incluirán los modos de inicio y fin, las horas de riego, el PWFC, volumen de agua, etc. Todos ellos se visualizarán mejor en las pantallas del controlador.

Las siguientes definiciones serán las correspondientes a las variables necesarias durante el programa, que incluirán una copia de todos los parámetros de riego, así como otras variables necesarias para la gestión del mismo. Dado que es importante tener una validación de parámetros antes de su almacenamiento, se han de crear variables temporales que las almacenen hasta dicha confirmación. Por ello se han creado las variables y una copia con el prefijo *t_*, indicando variable temporal.

```
byte zonariego=0;
byte modoinicioriego=0;
byte modofinriego1=0;
byte modofinriego2=0;

//byte diainicio=0;//NU
//byte horainicio=0;//NU
byte watertimeinterval1=0;
byte watertimeinterval2=0;
byte numriegos=0;
//byte intervaloriegos=0;//NU

byte horariego1=0;
byte horariego2=0;
byte horariego3=0;
byte horariego4=0;

float weight1=0.0;
float weight2=0.0;
byte watertimeduration1=0;
byte watertimeduration2=0;

byte drainpercent1=0;
byte drainpercent2=0;
float voltofin1=0.0;
float voltofin2=0.0;
byte actualizarconf=0;

byte t_zonariego=0;
byte t_modoinicioriego=0;
byte t_modofinriego=0;


byte t_diainicio=0;
byte t_horainicio=0;
byte t_watertimeinterval=0;
byte t_numriegos=0;
byte t_intervaloriegos=0;

byte t_horariego1=0;
byte t_horariego2=0;
byte t_horariego3=0;
byte t_horariego4=0;

float t_weight=0;
byte t_watertimeduration=0;

byte t_drainpercent=0;
float t_voltofin=0.0;
byte t_actualizarconf=0;
```

Y a continuación las variables relacionadas con el resto del programa.

```
byte diaultimoriego1=1;
byte diaultimoriego2=1;
//Variables para el inicio/stop riego.
float volumenregado1=0.0;
float volumenregado2=0.0;
float pesodrenaje1=0.0;
float pesodrenaje2=0.0;
float CDW1=0.0;
float CDW2=0.0;
float CPW1=0.0;
float CPW2=0.0;
float PWFC1=0.0;
float PWFC2=0.0;
float SPW1=0.0;
float SPW2=0.0;
float SDW1=0.0;
float SDW2=0.0;
```

Además, será necesario crear unas ID de alarmas, que se encargarán de gestionar los riegos en su hora correspondiente, así como de comprobar las condiciones de inicio y fin en caso de ser riego paramétrico.

Se han definido 5 alarmas para cada zona de riego:

```
int ID_Alarm_11=0;
int ID_Alarm_12=0;
int ID_Alarm_13=0;
int ID_Alarm_14=0;
int ID_Alarm_15=0;
//int ID_Alarm_16=0;
//int ID_Alarm_17=0;

int ID_Alarm_21=0;
int ID_Alarm_22=0;
int ID_Alarm_23=0;
int ID_Alarm_24=0;
int ID_Alarm_25=0;
//int ID_Alarm_26=0;
//int ID_Alarm_27=0;
```

Es importante destacar que para el uso de un número elevado de alarmas, habrá que modificar el archivo ***TimeAlarms.h,*** de la librería TimeAlarms, donde se define dicho máximo. Por defecto es un número inferior a 10 y por tanto no se podrían utilizar estas alarmas. El parámetro a modificar será: `#define dtNBR ALARMS 50  // max is 255` donde como se puede observar, ya se ha modificado a 50.

4.4.2.2. Función *setup ( )*

Será muy sencilla, siendo su único objetivo el de parametrizar las salidas como tales, presentarse como nodo y recuperar los valores de PWFC de la memoria EEPROM:

```
void setup(){

  pinMode(PIN_EV1,OUTPUT);
  pinMode(PIN_EV2,OUTPUT);
  pinMode(PIN_EV3,OUTPUT);
  pinMode(PIN_EV4,OUTPUT);

  digitalWrite(PIN_EV1,LOW);
  digitalWrite(PIN_EV2,LOW);
  digitalWrite(PIN_EV3,LOW);
  digitalWrite(PIN_EV4,LOW);
```

Comienza con las salidas, estableciéndolas a *LOW*, dado que son relés significará que las electroválvulas estarán cerradas, y no habrá flujo de agua.

```
  gw.begin(incomingMessage, LIS_NODE_ID, true);
  gw.wait(20);//Wait until initialization completed.
  gw.sendSketchInfo(LIS_NODE_NAME, LIS_NODE_VER);
  gw.present(CHILD_EV_1, S_LIGHT);// Register sensors
  gw.wait(20);//Wait 20 ms to let the msg arrive
  gw.present(CHILD_EV_2, S_LIGHT);
  gw.wait(20);
  gw.present(CHILD_EV_3, S_LIGHT);
  gw.wait(20);
  gw.present(CHILD_EV_4, S_LIGHT);
  gw.wait(20);
  gw.present(150, S_CUSTOM);
  gw.wait(20);
  gw.present(CHILD_OK, S_CUSTOM);
```

Posteriormente se presenta a sí mismo y a los mensajes de envío necesarios, entre los que se observa un mensaje con ID 150 y de tipo *CUSTOM*, que se utilizará para solicitar la fecha al controlador.

```
  //Cargamos el ultimo valor de PWFC de EEPROM.
  EEPROM.get(ADD_PWFC1,PWFC1);
  EEPROM.get(ADD_PWFC2,PWFC2);


  Serial.println("Finalizado el setup");
  gw.send(msgOK.set("OK"));//Mandamos OK
}
```

Por último, cargamos valores de EEPROM y enviamos OK.

4.4.2.3. Función *loop ( )*

Sin lugar a dudas la más sencilla de las funciones. Su única tarea será la de recibir los mensajes de radio y comprobar el estado de las alarmas, es decir, la potencia del programa se encontrará en las alarmas y los mensajes de radio recibidos:

```
void loop(){

gw.process();
Alarm.delay(0);

}
```

Comenzamos por tanto con las funciones que gestionarán realmente el riego.

4.4.2.4. Función *incomingMessage ( )*

Función ejecutada a la recepción de un mensaje de radio. Permite analizar el mensaje y extraer la información del mismo. Dado que toda la configuración será enviada a través de la radio, será esta función la más compleja, gestionando todo el programa.

La función se puede dividir según el mensaje recibido, más concretamente, según el remitente y el sensor objetivo del mensaje. Esta función comienza definiendo quién ha enviado el mensaje y a quién:

```
void incomingMessage (const MyMessage &message){
  int sensorobj=message.sensor;
  int remitente=message.sender;
  Serial.print("Hemos recibido mensaje para ");
  Serial.println(sensorobj);
```

De esta forma se puede empezar a clasificar. El primer mensaje que se espera será la respuesta del nodo *controlador* al *OK* del nodo *actuador,* con la información correspondiente a la fecha y hora actuales.

```
  if(sensorobj==150){//CH 150 lo envia openhab con info
    unsigned long timegw=atol(message.getString());
    Serial.println(timegw);
    setTime(timegw);
```

Esto será lo primero que ocurrirá, dado que sin fecha ni hora establecidas, las alarmas no funcionarán. Como se puede observar, el *sensor* objetivo es el 150, presentado anteriormente. Hecho esto, dentro del *if* comprobará con otras dos estructuras *if*, si existen datos almacenados para cada una de las dos zonas de riego, y en caso de ser así, los recupera de la memoria EEPROM y los restaura, actualizando las alarmas:

```
if(EEPROM.read(ADD_DATA_STORED1)==0b00000001){//Si hay datos de alarmas guardados
  //Retrieve data
  Serial.println("Hay datos en EEPROM para Zona 1:");
  zonariego=1;
  modoinicioriego=EEPROM.read(ADD_MODOINICIO1);Serial.println(modoinicioriego);
  modofinriego1=EEPROM.read(ADD_MODOFIN1);
  watertimeinterval1=EEPROM.read(ADD_WATERTIMEINT1);
  numriegos=EEPROM.read(ADD_NUMRIEGOS1);
  horariego1=EEPROM.read(ADD_HORARIEGO11);
  horariego2=EEPROM.read(ADD_HORARIEGO12);
  horariego3=EEPROM.read(ADD_HORARIEGO13);
  horariego4=EEPROM.read(ADD_HORARIEGO14);
  watertimeduration1=5*(EEPROM.read(ADD_WATERTIMEDUR1));
  drainpercent1=EEPROM.read(ADD_DRAINWEIGHT1);
  EEPROM.get(ADD_WEIGHT1,weight1);
  EEPROM.get(ADD_VOLTOFIN1,voltofin1);
  updatealarms();

}
if(EEPROM.read(ADD_DATA_STORED2)==0b00000001){//Si hay datos de alarmas guardados
  //Retrieve data
  Serial.println("Hay datos en EEPROM para Zona 2:");
  zonariego=2;
  modoinicioriego=EEPROM.read(ADD_MODOINICIO2);Serial.println(modoinicioriego);
  modofinriego2=EEPROM.read(ADD_MODOFIN2);
  watertimeinterval2=EEPROM.read(ADD_WATERTIMEINT2);
  numriegos=EEPROM.read(ADD_NUMRIEGOS2);
  horariego1=EEPROM.read(ADD_HORARIEGO21);
  horariego2=EEPROM.read(ADD_HORARIEGO22);
  horariego3=EEPROM.read(ADD_HORARIEGO23);
  horariego4=EEPROM.read(ADD_HORARIEGO24);
  watertimeduration2=5*(EEPROM.read(ADD_WATERTIMEDUR2));
  drainpercent2=EEPROM.read(ADD_DRAINWEIGHT2);
  EEPROM.get(ADD_WEIGHT2,weight2);
  EEPROM.get(ADD_VOLTOFIN2,voltofin2);
  updatealarms();
}
}
```

Así finalizaría en caso de ser la respuesta del controlador a la petición de hora. En caso de ser un mensaje destinado a otro *sensor* se gestionará con una estructura *switch-case:*

```
switch(sensorobj){
  case CHILD_EV_1://Riego manual 1
  digitalWrite(PIN_EV1, atoi(message.getString("1")) ? HIGH:LOW);//Iniciamos temp
  Alarm.timerOnce(1800, ev_stop1);
  break;
  case CHILD_EV_2://Riego manual 2
  digitalWrite(PIN_EV2, atoi(message.getString("1")) ? HIGH:LOW);
  Alarm.timerOnce(1800, ev_stop2);
  break;
  case CHILD_EV_3://Riego manual 3
  //digitalWrite(PIN_EV3, atoi(message.getString("1")) ? HIGH:LOW);
  //Alarm.timerOnce(1800, ev_stop3);
  break;
  case CHILD_EV_4://Riego manual 4
  //digitalWrite(PIN_EV4, atoi(message.getString("1")) ? HIGH:LOW);
  //Alarm.timerOnce(1800, ev_stop4);
```

En caso de ser orden manual a una electroválvula, abriría o cerraría la misma, añadiendo un temporizador de parada a los 30 minutos, evitando que se quede regando

de forma indefinida. Posteriormente aparece donde se leen y guardan las configuraciones temporales.

```
break;
case CH_ZONE:
t_zonariego=atoi(message.getString("1"));
Serial.print("Zona ");
Serial.println(message.getString("1"));
break;
case CH_MODO_I:
t_modoinicioriego=atoi(message.getString("1"));
Serial.print("Modo I ");
Serial.println(atoi(message.getString("1")));
break;
case CH_MODO_F:
t_modofinriego=atoi(message.getString("1"));
Serial.print("Modo F ");
Serial.println(atoi(message.getString("1")));
break;
case CH_DIA_I:
t_diainicio=atoi(message.getString("1"));
break;
case CH_HORA_I:
t_horainicio=atoi(message.getString("1"));
break;
case CH_INT_R:
t_watertimeinterval=atoi(message.getString("1"));
//t_intervaloriegos=atoi(message.getString("1"));
Serial.print("Interval ");
Serial.println(atoi(message.getString("1")));
break;
case CH_NUM_R:
t_numriegos=atoi(message.getString("1"));
Serial.print("Riegos ");
Serial.println(atoi(message.getString("1")));
break;
```

```
    case CH_HORA_R1:
    t_horariego1=atoi(message.getString("2"));
    Serial.print("Hora1: ");
    Serial.println(atoi(message.getString("2")));
    break;
    case CH_HORA_R2:
    t_horariego2=atoi(message.getString("2"));
    Serial.print("Hora2: ");
    Serial.println(atoi(message.getString("2")));
    break;
    case CH_HORA_R3:
    t_horariego3=atoi(message.getString("2"));
    Serial.print("Hora3: ");
    Serial.println(atoi(message.getString("2")));
    break;
    case CH_HORA_R4:
    t_horariego4=atoi(message.getString("2"));
    Serial.print("Hora4: ");
    Serial.println(atoi(message.getString("2")));
    break;
    case CH_WRED:
    t_weight=atof(message.getString("3"));
    Serial.print("Peso ");
    Serial.println(atof(message.getString("3")));
    break;
    case CH_TIEMP_RIEGO:
    t_watertimeduration=atoi(message.getString("2"));
    Serial.print("Duracion ");
    Serial.println(atoi(message.getString("2")));
    break;
    case CH_DRAIN_F:
    t_drainpercent=atoi(message.getString("3"));
    Serial.print("Drenaje ");
    Serial.println(atoi(message.getString("3")));
    break;
    case CH_VOL_F:
    t_voltofin=atof(message.getString("3"));
    Serial.print("Vol final ");
    Serial.println(atof(message.getString("3")));
    break;
  }
```

Tal y como se ha comentado, todas las configuraciones se almacenarán de forma temporal hasta enviar una señal de validación, la cual se incluye a continuación, dentro de la misma función.

```
/* VALIDACION DE PARAMETROS *///If we receive the validate signal.
if((sensorobj==CH_VALIDATE)&&(atoi(message.getString("1"))==1)){
  //Check if we didn't receive some values, so we use the last ones...
  if(t_zonariego==0){t_zonariego=zonariego;}
  if(t_modoinicioriego==0){t_modoinicioriego=modoinicioriego;}
  if(t_modofinriego==0){if(zonariego==1){t_modofinriego=modofinriego1;}
                        if(zonariego==2){t_modofinriego=modofinriego2;}}
  if(t_weight==0){if(zonariego==1){t_weight=weight1;}
                  if(zonariego==2){t_weight=weight2;}}
  if(t_watertimeduration==0){if(zonariego==1){t_watertimeduration=watertimeduration1;}
                             if(zonariego==2){t_watertimeduration=watertimeduration2;}}
  if(t_watertimeinterval==0){if(zonariego==1){t_watertimeinterval=watertimeinterval1;}
                             if(zonariego==2){t_watertimeinterval=watertimeinterval2;}}
  if(t_numriegos==0){t_numriegos=numriegos;}
  if(t_horariego1==0){t_horariego1=horariego1;}
  if(t_horariego2==0){t_horariego2=horariego2;}
  if(t_horariego3==0){t_horariego3=horariego3;}
  if(t_horariego4==0){t_horariego4=horariego4;}
  if(t_drainpercent==0){if(zonariego==1){t_drainpercent=drainpercent1;}
                        if(zonariego==2){t_drainpercent=drainpercent2;}}
  if(t_voltofin==0){if(zonariego==1){t_voltofin=voltofin1;}
                    if(zonariego==2){t_voltofin=voltofin2;}}
  updateirrigationvalues();
  updatealarms();
  Serial.print("ALARMAS ACTUALIZADAS. ");
}
```

Si se recibe validación, se comprobará si hay valores temporales para toda la configuración, en caso de no haber recibido alguno, se utilizará el guardado anteriormente. En ese momento se actualizarán los valores a permanentes y se activarán las alarmas.

Existirán otros dos casos dentro de la misma función, que serán aquellos en los que en lugar del controlador, sean los nodos *sensor,* de los lisímetros, los que envíen datos, en ese caso se leerán y almacenarán, para posteriormente en las alarmas utilizarlos:

```
    if(remitente==1){//Si el nodo 1 envia datos.
        if(sensorobj==7){//Datos de volumen de agua de riego //Fin riego
          volumenregado1=message.getFloat();
        }else if(sensorobj==22){//Datos de peso de drenaje actual
          CDW1=message.getFloat();//Current drain weight //Fin riego
        }else if(sensorobj==20){//Pot Weight Field Capacity //Fin riego
          PWFC1=message.getFloat();
          EEPROM.put(ADD_PWFC1,PWFC1);
          EEPROM.update(ADD_PWFC1_SET,0b00000001);
        }else if(sensorobj==21){//Current Pot Weight //Inicio riego
          CPW1=message.getFloat();
        }
    }else if(remitente==2){//Nodo 2 envia datos
        if(sensorobj==7){//Datos de volumen de agua de riego //Fin riego
          volumenregado2=message.getFloat();
        }else if(sensorobj==22){//Datos de peso de drenaje actual
          CDW1=message.getFloat();//Current drain weight //Fin riego
        }else if(sensorobj==20){//Pot Weight Field Capacity //Fin riego
          PWFC2=message.getFloat();
          EEPROM.put(ADD_PWFC2,PWFC2);
          EEPROM.update(ADD_PWFC2_SET,0b00000001);
        }else if(sensorobj==21){//Current Pot Weight //Inicio riego
          CPW1=message.getFloat();
        }

    }


}
```

De esta forma terminaría la función *incomingMessage ( )*. Cabe destacar ciertos aspectos acerca de esta función:

- En lugar de hacer uso de diversas estructuras, se podría incluir todo en una, pero dado que no se dan varios casos al mismo tiempo, se ha considerado más ordenado y mejor visible de esta manera.
- La recepción de datos del controlador tiene un problema, se reciben más datos de los que realmente se envían, es decir, se recibe el dato deseado, pero además le sigue una secuencia de "basura" del envío anterior. Para evitar utilizar datos erróneos se recupera el mensaje con el buffer deseado.
- En el caso de recibir mensajes de los nodos *sensor*, no es necesario hacer ninguna conversión especial.
- Al leer configuraciones nuevas y validarlas, éstas se almacenarán en la memoria EEPROM, de forma que, si se vuelve a validar, y no se envían datos nuevos, se puedan utilizar los últimos, al igual que al reiniciar el dispositivo.

4.4.2.5. Función *updateirrigationvalues ( )*

Como su propio nombre indica, esta función actualizará los valores de irrigación, es decir, los pasará de valores temporales a permanentes, almacenándolos asimismo en la memoria EEPROM. A esta función se la llamará desde diversas funciones.

```
void updateirrigationvalues(){//Copies all the temporary values
  zonariego=t_zonariego;
   modoinicioriego=t_modoinicioriego;

   //diainicio=t_diainicio;
   //horainicio=t_horainicio;
   numriegos=t_numriegos;
   //intervaloriegos=t_intervaloriegos;
   horariego1=t_horariego1;
   horariego2=t_horariego2;
   horariego3=t_horariego3;
   horariego4=t_horariego4;
   if(zonariego==1){
   weight1=t_weight;//Hay que diferenciar.
   watertimeduration1=5*t_watertimeduration;//Multiplicamos por 5
   drainpercent1=t_drainpercent;
   voltofin1=t_voltofin;
   watertimeinterval1=t_watertimeinterval;
   modofinriego1=t_modofinriego;
   }else{
   weight2=t_weight;//Hay que diferenciar.
   watertimeduration2=5*t_watertimeduration;//Multiplicamos por 5
   drainpercent2=t_drainpercent;
   voltofin2=t_voltofin;
   watertimeinterval2=t_watertimeinterval;
   modofinriego2=t_modofinriego;
   }
```

Lo primero será por tanto copiar los valores a permanentes.

```
if(t_zonariego==1){
  EEPROM.update(ADD_DATA_STORED1,0b00000001);
  EEPROM.update(ADD_MODOINICIO1,t_modoinicioriego);
  EEPROM.update(ADD_MODOFIN1,t_modofinriego);
  EEPROM.update(ADD_WATERTIMEINT1,t_watertimeinterval);
  EEPROM.update(ADD_NUMRIEGOS1,t_numriegos);
  EEPROM.update(ADD_HORARIEGO11,t_horariego1);
  EEPROM.update(ADD_HORARIEGO12,t_horariego2);
  EEPROM.update(ADD_HORARIEGO13,t_horariego3);
  EEPROM.update(ADD_HORARIEGO14,t_horariego4);
  EEPROM.update(ADD_WATERTIMEDUR1,t_watertimeduration);
  EEPROM.update(ADD_DRAINWEIGHT1,t_drainpercent);
  EEPROM.put(ADD_WEIGHT1,t_weight);
  EEPROM.put(ADD_VOLTOFIN1,t_voltofin);
}else if(t_zonariego==2){
  EEPROM.update(ADD_DATA_STORED2,0b00000001);
  EEPROM.update(ADD_MODOINICIO2,t_modoinicioriego);
  EEPROM.update(ADD_MODOFIN2,t_modofinriego);
  EEPROM.update(ADD_WATERTIMEINT2,t_watertimeinterval);
  EEPROM.update(ADD_NUMRIEGOS2,t_numriegos);
  EEPROM.update(ADD_HORARIEGO21,t_horariego1);
  EEPROM.update(ADD_HORARIEGO22,t_horariego2);
  EEPROM.update(ADD_HORARIEGO23,t_horariego3);
  EEPROM.update(ADD_HORARIEGO24,t_horariego4);
  EEPROM.update(ADD_WATERTIMEDUR2,t_watertimeduration);
  EEPROM.update(ADD_DRAINWEIGHT2,t_drainpercent);
  EEPROM.put(ADD_WEIGHT2,t_weight);
  EEPROM.put(ADD_VOLTOFIN2,t_voltofin);
}
```

A continuación, almacenarlos en EEPROM, según a que zona pertenezcan.

```
  t_zonariego=0;
  t_modoinicioriego=0;
  t_modofinriego=0;
  t_diainicio=0;
  t_horainicio=0;
  t_watertimeinterval=0;
  t_numriegos=0;
  t_intervaloriegos=0;//NU
  t_horariego1=0;
  t_horariego2=0;
  t_horariego3=0;
  t_horariego4=0;
  t_weight=0;
  t_watertimeduration=0;
  t_drainpercent=0;
  t_voltofin=0;
  t_actualizarconf=0;
}
```

Y por último, resetear los valores temporales.

4.4.2.6. Función *updatealarms ( )*

Otra de las funciones importantes, se encargará de activar las alarmas correspondientes con todos los datos tomados de EEPROM y el controlador. Sólo podrá activarse estando la hora y día establecidos, dado que, en caso contrario, las alarmas no funcionarán. La función está dividida en dos, con una estructura *if-else,* separando las alarmas para cada zona:

```
void updatealarms(){
  if(zonariego==1){
      Serial.println("Alarmas para la zona 1: ");
      Alarm.free(ID_Alarm_11);//First thing to do is setting all alarms free s
      Alarm.free(ID_Alarm_12);
      Alarm.free(ID_Alarm_13);
      Alarm.free(ID_Alarm_14);
      Alarm.free(ID_Alarm_15);
      //Alarm.free(ID_Alarm_16); //NU
      //Alarm.free(ID_Alarm_17); //NU
      //Once we get rid of all the alarms, we set them with the new parameters
        if(modoinicioriego==1){//Inicio por tiempo
          Serial.println("Creadas alarmas de inicio temporal.");
          if(numriegos==1){
          ID_Alarm_11=Alarm.alarmRepeat(horariego1, 0, 0, f_iniciotemporal1);
          //Con esta puedo chequear todos los dias y func solo si es el bueno.
          }else if(numriegos==2){
          ID_Alarm_11=Alarm.alarmRepeat(horariego1, 0, 0, f_iniciotemporal1);
          ID_Alarm_12=Alarm.alarmRepeat(horariego2, 0, 0, f_iniciotemporal1);
          }else if(numriegos==3){
          ID_Alarm_11=Alarm.alarmRepeat(horariego1, 0, 0, f_iniciotemporal1);
          ID_Alarm_12=Alarm.alarmRepeat(horariego2, 0, 0, f_iniciotemporal1);
          ID_Alarm_13=Alarm.alarmRepeat(horariego3, 0, 0, f_iniciotemporal1);
          }else if(numriegos==4){
          ID_Alarm_11=Alarm.alarmRepeat(horariego1, 0, 0, f_iniciotemporal1);
          ID_Alarm_12=Alarm.alarmRepeat(horariego2, 0, 0, f_iniciotemporal1);
          ID_Alarm_13=Alarm.alarmRepeat(horariego3, 0, 0, f_iniciotemporal1);
          ID_Alarm_14=Alarm.alarmRepeat(horariego4, 0, 0, f_iniciotemporal1);
          }
        }else if(modoinicioriego==2){//Inicio por lisimetria
          ID_Alarm_11=Alarm.timerRepeat(30,f_iniciolisimetrico1);
        }
```

Lo primero que hará, una vez seleccionada la zona a modificar, será liberar las alarmas anteriores de dicha zona, es decir, borrarlas para poder sobrescribirlas, al no poderse sobrescribir directamente. Posteriormente, según el modo de inicio de riego, activará alarmas a las horas indicadas, o activará una vigilancia que active el riego cuando se cumplan los parámetros. Para ello llamarán a las funciones *f_iniciotemporal1* y *f_iniciolisimétrico1*, que se verán posteriormente. Para la zona 2 será exactamente igual, sustituyendo todos los "1" por "2".

```
}else if (zonariego==2){
  Serial.println("Alarmas para la zona 2: ");
  Alarm.free(ID_Alarm_21);//First thing to do is setting all alarms free
  Alarm.free(ID_Alarm_22);
  Alarm.free(ID_Alarm_23);
  Alarm.free(ID_Alarm_24);
  Alarm.free(ID_Alarm_25);
  //Alarm.free(ID_Alarm_26); //NU
  //Alarm.free(ID_Alarm_27); //NU
  //Once we get rid of all the alarms, we set them with the new parameter
  if(modoinicioriego==1){//Inicio por tiempo
      if(numriegos==1){
      ID_Alarm_21=Alarm.alarmRepeat(horariego1, 0, 3, f_iniciotemporal2);
      //Con esta puedo chequear todos los dias y func solo si es el bueno
      }else if(numriegos==2){
      ID_Alarm_21=Alarm.alarmRepeat(horariego1, 0, 3, f_iniciotemporal2);
      ID_Alarm_22=Alarm.alarmRepeat(horariego2, 0, 3, f_iniciotemporal2);
      }else if(numriegos==3){
      ID_Alarm_21=Alarm.alarmRepeat(horariego1, 0, 3, f_iniciotemporal2);
      ID_Alarm_22=Alarm.alarmRepeat(horariego2, 0, 3, f_iniciotemporal2);
      ID_Alarm_23=Alarm.alarmRepeat(horariego3, 0, 3, f_iniciotemporal2);
      }else if(numriegos==4){
      ID_Alarm_21=Alarm.alarmRepeat(horariego1, 0, 3, f_iniciotemporal2);
      ID_Alarm_22=Alarm.alarmRepeat(horariego2, 0, 3, f_iniciotemporal2);
      ID_Alarm_23=Alarm.alarmRepeat(horariego3, 0, 3, f_iniciotemporal2);
      ID_Alarm_24=Alarm.alarmRepeat(horariego4, 0, 3, f_iniciotemporal2);
      }
    }else if(modoinicioriego==2){//Inicio por lisimetria
      ID_Alarm_21=Alarm.timerRepeat(30,f_iniciolisimetrico2);
    }


}

}
```

Definidas las funciones que controlarán los parámetros y las alarmas, toca explicar las funciones a las que llamarán las anteriores. Se explicarán sólo para un nodo, dado que será igual para los demás.

4.4.2.7. Función *f_iniciotemporal1 ( )*

La función se activará, si está configurado su modo de inicio, todos los días a las horas indicadas, sin embargo, al inicio de la función comprobará si el día que se ejecuta hay que regar o no. Para ello comprobará cuando fue el último riego y el intervalo de riego deseado. Al activarse, actualizará el último día regado y enviará la información al controlador. Posteriormente activará el modo de fin de riego adecuado.

```
void f_iniciotemporal1(){//Inicia el riego y activa el fin que corresponda.
  //Primero comprobar si hoy toca o no, según watertimeinterval
  Serial.println("Se riega hoy");
  if(watertimeinterval1*numriegos==diaultimoriego1){

    diaultimoriego1=1;//Establece a 1
    //Al iniciar el riego guarda el valor actual de drenaje y peso
    SDW1=CDW1;
    SPW1=CPW1;
    digitalWrite(PIN_EV1,HIGH);
    //Alarm.enable(ID_Alarm_15);//Activamos la parada correspondiente.
    gw.send(msgEV1.set("ON"));
        if(modofinriego1==1){//Fin por tiempo.
          ID_Alarm_15=Alarm.timerOnce((watertimeduration1*60),f_finriego1);
          Serial.println("Modo fin por tiempo.");
        }else if(modofinriego1==2){//Fin por drenaje
          ID_Alarm_15=Alarm.timerRepeat(30,f_checkdrainage1);
        }else if(modofinriego1==3){//Fin por volumen de agua
          ID_Alarm_15=Alarm.timerRepeat(30,f_checkvolume1);
        }
  }else {
    diaultimoriego1++;
  }

}
```

4.4.2.8. Función *f_iniciolisimetrico1 ( )*

Esta función se ejecutará de forma periódica llamada por la alarma correspondiente. Su algoritmo será:

1. Comprobar si hay datos de PWFC, dado que si no existen tendrá que realizar un riego normal.
2. Si existen datos, comprobar si se cumple la condición establecida para iniciar el riego. Si no es así terminará la función hasta que se vuelva a llamar. Si se cumple dicha condición, comenzará el riego, actualizando los valores de peso de drenaje y planta actuales y enviando los datos al controlador.
3. Iniciado el riego, activará, según la parametrización, las alarmas que controlarán el fin del riego, dependiendo, por supuesto, del modo de fin de riego indicado.

```
void f_iniciolisimetrico1(){//Comprueba la condición de lisimetría
  //Cond. lisimetria?
  if(EEPROM.read(ADD_PWFC1_SET)!=0b00000001){//Aun no se han tomado medidas
    diaultimoriego1=1;
    f_iniciotemporal1();//Comenzamos un riego estándar.
  }else if(CPW1<(PWFC1-weight1)){
    SDW1=CDW1;
    SPW1=CPW1;
    digitalWrite(PIN_EV1,HIGH);
    //Alarm.enable(ID_Alarm_15);//Activamos la parada correspondiente.
    gw.send(msgEV1.set("ON"));
      if(modofinriego1==1){//Fin por tiempo.
          ID_Alarm_15=Alarm.timerOnce((watertimeduration1*60),f_finriego1);
          Serial.println("Modo fin por tiempo.");
        }else if(modofinriego1==2){//Fin por drenaje
          ID_Alarm_15=Alarm.timerRepeat(30,f_checkdrainage1);
        }else if(modofinriego1==3){//Fin por volumen de agua
          ID_Alarm_15=Alarm.timerRepeat(30,f_checkvolume1);
        }
  }

}
```

4.4.2.9. Funciones de fin de riego

Dependiendo del modo de fin de riego seleccionado, las funciones se ejecutarán una sola vez o de forma periódica.

La primera será *f_checkvolume1 ( ),* que comprobará si el volumen de agua regado es superior al seleccionado, cuando así sea, detendrá el riego e informará de ello, además desactivará la alarma para poder ser activada de nuevo. Se llamará de forma periódica a esta función.

```
void f_checkvolume1(){//Fin por volumen de agua regado.
  if(volumenregado1>=voltofin1){
    digitalWrite(PIN_EV1,LOW);
    Alarm.free(ID_Alarm_15);//Desactivamos la parada correspondiente.
    gw.send(msgEV1.set("OFF"));
  }
}
```

La función *f_finriego1 ( )* será llamada una sola vez, y tras el tiempo de riego correspondiente, por lo que simplemente cortará el agua de riego e informará de ello, desactivando la alarma que la llamó.

```
void f_finriego1(){//Fin por timepo
    digitalWrite(PIN_EV1,LOW);
    Alarm.free(ID_Alarm_15);//Desactivamos la parada correspondiente.
    gw.send(msgEV1.set("OFF"));
}
```

Por último, *f_checkdrainage1 ( ),* se llamará de forma periódica para comprobar el estado del drenaje, cuando se cumpla la condición elegida (si este es el modo de finalización seleccionado) detendrá el riego y al igual que las anteriores, informará de ello y eliminará la alarma correspondiente.

```
void f_checkdrainage1(){//Fin por drenaje.
  if((CDW1-SDW1)>=((CPW1-SPW1)*(drainpercent1/100))){//StartingPotWeight
    digitalWrite(PIN_EV1,LOW);
    Alarm.free(ID_Alarm_15);//Desactivamos la parada correspondiente.
    gw.send(msgEV1.set("OFF"));
  }
}
```

4.4.2.10. Función *ev_stop ( )*

Función llamada tras un tiempo determinado, en caso de activar un riego manual y no desactivarlo dentro de dicho tiempo. Simplemente detendrá el riego e informará al controlador.

```
void ev_stop1(){
  //Takes feed out from both signals.
  digitalWrite(PIN_EV1,LOW);
  gw.send(msgEV1.set("OFF"));
}
void ev_stop2(){
  //Takes feed out from both signals.
  digitalWrite(PIN_EV2,LOW);
  gw.send(msgEV2.set("OFF"));
}
```

4.5. Nodo *Controlador*

El nodo *controlador,* o controlador sólo, será el encargado de generar el servidor web que se utilice para la gestión de la configuración, además de almacenar los datos en un servidor MySQL y gestionar tanto dicha database como los objetos del sistema.

4.5.1. Diseño Hardware

En cuanto a la parte física, nos encontraremos con una placa Raspberry Pi, conectada vía Ethernet con un router. A ésta se le conectará además un LED en su GPIO 4, que servirá para indicar el correcto arranque de openHAB.

Ilustración 58 - Raspberry Pi con LED de estado

En caso de instalar el controlador en una zona agrícola, sería necesario incorporarlo dentro de una caja estanca con algún sistema de refrigeración. En Raspberry Pi 2 y 3 se recomienda el uso de disipadores de cobre o aluminio.

4.5.2. Programación del nodo

La programación de este nodo será de las más complejas, al tener que integrar diversos sistemas. Este apartado se dividirá por tanto según las distintas configuraciones y opciones a modificar.

4.5.2.1. General. Configuración de IP estática

Como ya se ha comentado, el controlador irá conectado a un router, sin embargo, para poder acceder a él, será necesario conocer su IP en todo momento. Para ello se le otorgará una IP estática, de forma que ésta, no varíe.

Para establecer dicha IP habrá que escribir lo siguiente en la consola de Linux:

Sudo nano /etc/network/interfaces

Se abrirá un archivo de texto, en el cual modificaremos el apartado correspondiente a Ethernet, en concreto al adaptador eth0, dejando la configuración como sigue:

```
auto eth0
allow-hotplug eth0
iface eth0 inet static
address 192.168.1.106
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
gateway 192.168.1.1
```

De esta forma la IP estática pasará a ser la *192.168.1.106,* será importante anotarla, dado que será la dirección a la que nos conectemos para acceder al servidor web.

4.5.2.2. MySQL. Auto-exportación y limpieza

La base de datos generada, irá actualizándose y aumentando de tamaño conforme vayan cambiando los estados de los diferentes objetos. Es decir, de forma natural irá incrementando el espacio que ocupa en disco. Dado que la memoria de Raspberry Pi se reduce al tamaño de la tarjeta micro SD insertada en ella, tendremos una limitación importante, por lo que será requisito indispensable eliminar datos antiguos. Además, se ha considerado útil la creación de una copia de seguridad, de forma que no se puedan perder datos.

Para realizar las tareas automáticas o periódicas se hará uso de *crontab* (Ver anexo correspondiente). En concreto se realizarán tres tareas:

- La copia de seguridad se hará de forma que se puedan restaurar los datos posteriormente a la base de datos, es decir será una copia de seguridad con el formato MySQL. Para realizarla se utilizará mysqldump, una herramienta integrada en MySQL. Exportados los datos se comprimirán: *10 10 1 1/3 * root mysqldump -u root -proot openhabdb | gzip > /home/pi/BackupDB/database_ `date '+%Y%m%d'`.sql.gz*
- Asimismo, se guardarán las tablas con un formato CSV (Comma Separated Value) con el siguiente comando: *0 * * * * root mysqldump -u*

root -proot --fields-terminated-by='\t' --tab /home/pi/BackupDB/ openhabdb

- Por último se ejecutará cada 3 meses (modificable) un script de borrado de datos antiguos (6 meses), manteniendo el tamaño en memoria en un rango adecuado: *10 10 1 1*3 * root mysql -u root -proot openhabdb <delete_old.sql*

4.5.2.3. OpenHAB. Arranque y auto-arranque

Una vez instalado y configurado openHAB, es posible arrancarlo ejecutando desde una consola el archivo *start.sh* dentro del directorio de openHAB. Para ello escribir en la consola lo siguiente: *sudo /opt/openhab/start.sh*

Sin embargo, si lo que se desea es un auto-arranque del servicio, habrá que configurar openHAB como tal. Para ello habrá que utilizar un *script* y un archivo de configuración, establecer los permisos necesarios y por último incorporarlo dentro de la secuencia de arranque de Raspberry Pi.

Para establecer el auto arranque seguiremos la guía de la página web *homeautomatmionforgeeks.com*. Para ello seguiremos los siguientes pasos:

1. Descargaremos el *script* que ofrece y copiaremos su contenido dentro de */etc/init.d/openhab*.
2. Daremos permisos al archivo: *sudo chmod 777 /etc/init.d/openhab*
3. Para hacer el script autoejecutable, escribiremos: *sudo update-rc.d openhab defaults*
4. *Listo*. Para deshacer el auto-arranque basta con repetir el comando anterior, sustituyendo *defaults* por *remove*.

Terminado el arranque podremos acceder al servidor desde la dirección:

http://192.168.1.106:8080/openhab.app?sitemap=main

4.5.2.4. OpenHAB. *Binding* incluidos

Los *binding* incluidos para ampliar la funcionalidad al nodo controlador son los siguientes:

- *action.mqtt* y *binding.mqtt:* Necesarios para el uso del protocolo MQTT.
- *io.gpio y binding.gpio:* Para el uso de los pines digitales de Raspberry Pi.
- *binding.http*: No se utiliza, pero se deja para posibles futuras implementaciones.
- *binding.ntp:* No se utiliza, pero se deja para futuras implementaciones.
- *binding.systeminfo:* Proporciona información del sistema, en este caso Raspberry Pi, acerca de memoria, RAM, procesador…
- *binding.weather:* Permite comunicar con servicios proveedores de clima.
- *persistence.mysql:* Permite la comunicación con el servidor MySQL.

4.5.2.5. OpenHAB. *openhab.cfg*

El archivo principal de configuración. Se lee antes de iniciar openHAB y ofrece algunas de las configuraciones más importantes. A continuación, se detallan las utilizadas, clasificadas igual que en archivo:

```
##################################################################################################
#####                              General configurations                                    #####
##################################################################################################

# Configuration folders (must exist as a subdirectory of "configurations"; the value
# tells the number of seconds for the next scan of the directory for changes. A
# value of -1 deactivates the scan).
# A comma separated list can follow after the refresh value. This list defines a filter
# for valid file extensions for the models.
folder:items=30,items
folder:sitemaps=30,sitemap
folder:rules=30,rules
folder:scripts=30,script
folder:persistence=30,persist

# configures the security options. The following values are valid:
# ON = security is switched on generally
# OFF = security is switched off generally
# EXTERNAL = security is switched on for external requests
#            (e.g. originating from the Internet) only
# (optional, defaults to 'OFF')
#security:option=

# the Netmask to define a range of internal IP-Addresses which doesn't require
# authorization (optional, defaults to '192.168.1.0/24')
security:netmask=192.168.1.1/200

# The name of the default persistence service to use
persistence:default=mysql
```

En la configuración general sólo se cambia la frecuencia de refresco de las carpetas, las IP internas a utilizar, y el servicio de persistencia por defecto.

```
############################## SQL Persistence Service ###################################
# the database url like 'jdbc:mysql://<host>:<port>/<user>'
mysql:url=jdbc:mysql://localhost:3306/openhabdb

# the database user
mysql:user=openhab

# the database password
mysql:password=openhab

# the reconnection counter
mysql:reconnectCnt=1

# the connection timeout (in seconds)
#mysql:waitTimeout=

# optional tweaking of mysql datatypes
# example as described in https://github.com/openhab/openhab/issues/710
mysql:sqltype.string=VARCHAR(20000)
```

En el servicio de persistencia MySQL los datos de usuario y dirección de la base de datos. Como se puede observar, openHAB tiene su propio usuario de acceso. Al final del archivo se añade una opción de configuración que mejora el funcionamiento con los tipos de datos.

```
################################################################################################
#####                          Transport configurations                                    #####
################################################################################################

################################## MQTT Transport ######################################
#
# Define your MQTT broker connections here for use in the MQTT Binding or MQTT
# Persistence bundles. Replace <broker> with a id you choose.
#

# URL to the MQTT broker, e.g. tcp://localhost:1883 or ssl://localhost:8883
mqtt:ArdMQTT.url=tcp://192.168.1.150:3000

# Optional. Client id (max 23 chars) to use when connecting to the broker.
# If not provided a default one is generated.
mqtt:ArdMQTT.clientId=Openhab

# Optional. User id to authenticate with the broker.
#mqtt:ArdMQTT.user=rpi

# Optional. Password to authenticate with the broker.
#mqtt:ArdMQTT.pwd=raspberry

# Optional. Set the quality of service level for sending messages to this broker.
# Possible values are 0 (Deliver at most once),1 (Deliver at least once) or 2
# (Deliver exactly once). Defaults to 0.
mqtt:ArdMQTT.qos=0
```

Una de las configuraciones más importantes, la de transporte MQTT, habrá que establecer la IP del bróker, es decir la del nodo *Gateway, y el puerto de comunicación.* Como opción, además, la calidad del servicio que se deja en "0" al producir errores en las otras dos configuraciones. El resto de parámetros no se utilizarán.

```
############################### Systeminfo Binding ###################################
#
# Interval in milliseconds when to find new refresh candidates
# (optional, defaults to 1000)
#systeminfo:granularity=

# Data Storage Unit, where B=Bytes, K=kB, M=MB, T=TB (optional, defaults to M)
#systeminfo:units=
```

La configuración de SystemInfo no se modifica al ser la configuración por defecto la deseada.

```
################################### GPIO Binding #######################################

# Optional directory path where "sysfs" pseudo file system is mounted, when isn't
# specified it will be determined automatically if "procfs" is mounted
#gpio:sysfs=/sys

# Optional time interval in miliseconds when pin interrupts are ignored to
# prevent bounce effect, mainly on buttons. Global option for all pins, can be
# overwritten per pin in item configuration. Default value if omitted: 0
#gpio:debounce=10

# Boolean controlling whether already exported pin should be forcibly taken under
# control of openHAB. Usefull after unclean shutdown. May cause serios issue if
# other software/system is also controlling GPIO and there is a configuration
# mistake for pin name/number. Default value if omitted: false.
#gpio:force=true
```

La configuración referente a las GPIO tampoco se modifica al ser utilizado simplemente para activar un LED. En caso de tener entradas o un control más amplio, sería necesario modificar dicha configuración. Además habrá que modificar el script *Init.d* de forma que

exporte los pines después de cerrar openHAB, en caso contrario se quedan “colgados” y no responden.

```
################################## Weather Binding ###################################
#
# The apikey for the different weather providers, at least one must be specified
# Note: Hamweather requires two apikeys: client_id=apikey, client_secret=apikey2
#weather:apikey.ForecastIo=57f362a6dc4a12a02c04fd25f6622ea3
#weather:apikey.OpenWeatherMap=c6ec2e0bc04ab63e81595c7ebd168d6b
#weather:apikey.WorldWeatherOnline=
#weather:apikey.Wunderground=5f203b1b23b51178
#weather:apikey.Hamweather=
#weather:apikey2.Hamweather=

# location configuration, you can specify multiple locations
#weather:location.<locationId1>.name=
#weather:location.home.woeid=756324
weather:location.home.latitude=37.6067
weather:location.home.longitude=-0.9880
weather:location.home.provider=Wunderground
weather:location.home.language=sp
weather:location.home.updateInterval=10

#weather:location.home2.latitude=37.6067
#weather:location.home2.longitude=-0.9880
#weather:location.home2.provider=OpenWeatherMap
#weather:location.home2.language=sp
#weather:location.home2.updateInterval=10

weather:location.home3.woeid=756324
weather:location.home3.provider=Yahoo
weather:location.home3.language=sp
weather:location.home3.updateInterval=10
```

Por último la configuración de las API de clima, con las correspondientes claves API. En la programación final se deshabilita al consumir muchos recursos de la Raspberry Pi y ralentizar considerablemente el arranque.

4.5.2.6. OpenHAB. *Items*

Para realizar cualquier acción o mostrar cualquier tipo de elemento en el servidor web que genera openHAB, será necesario primero crear un objeto o *ítem* asociado. En el presente trabajo se han definido dos archivos *.items* clasificando algo mejor todos los objetos utilizados.

Comenzamos con los objetos de sistema, *system.items*:

```
/* SystemInfo */
String  UptimeFormatted "Uptime [%s horas encendido]" (System) {systeminfo="UptimeFormatted:300000"}
/* RamInfo */
Number  memUsedPercent  "RAM Usada [%.1f %%]" <ram> (RAMInfo) {systeminfo="MemUsedPercent:300000"}
Number  memActualFree   "RAM Libre [%.1f MB]" <ram> (RAMInfo) {systeminfo="MemActualFree:300000"}
Number  memActualUsed   "RAM Usada [%.1f MB]" <ram> (RAMInfo) {systeminfo="MemActualUsed:300000"}
Number  memTotal        "RAM Total [%.1f MB]" <ram> (RAMInfo) {systeminfo="MemTotal:300000"}

/* DiskInfo */
Number  FileSystemUsagePercent  "Disco Usado [%.1f %%]" <micro_sd>  (DiskInfo)  { systeminfo="FileSystemUsagePercent:300000:/" }
Number  FileSystemFree          "Disco Libre [%.1f MB]" <micro_sd>  (DiskInfo)  { systeminfo="FileSystemFree:300000:/" }
Number  FileSystemUsed          "Disco Usado [%.1f MB]" <micro_sd>  (DiskInfo)  { systeminfo="FileSystemUsed:300000:/" }
Number  FileSystemTotal         "Disco Total [%.1f MB]" <micro_sd>  (DiskInfo)  { systeminfo="FileSystemTotal:300000:/" }
/* BatteryInfo */
Number  Battery1          "Nivel de bateria  Nodo 1 [%d V]"   <battery>   (mqtt="<[ArdMQTT:ArdMQTT/1/12/V_VOLTAGE:state:default]"}
Number  Battery2          "Nivel de bateria  Nodo 2 [%d V]"   <battery>   (mqtt="<[ArdMQTT:ArdMQTT/2/12/V_VOLTAGE:state:default]"}
Number  Battery3          "Nivel de bateria  Nodo 3 [%d V]"   <battery>   (mqtt="<[ArdMQTT:ArdMQTT/3/12/V_VOLTAGE:state:default]"}
Number  Battery4          "Nivel de bateria  Nodo 4 [%d V]"   <battery>   (mqtt="<[ArdMQTT:ArdMQTT/4/12/V_VOLTAGE:state:default]"}
```

Éstos serán los objetos asociados a elementos de sistema, es decir, todos los correspondientes al *binding systeminfo*, y además los estados de las baterías. Dado el largo tamaño del código, es recomendable abrir el archivo desde el software *openHAB Designer*.

Continuando con el resto de objetos en *main.items*, tendremos:

```
/* Grupos Generales */
//type  name                "label"                    <icon>      (groups)

Group   AllItems            "All"
Group   System              "System"                   <info>
Group   RAMInfo             "RamInfo"                  <info>      (System)
Group   DiskInfo            "DiskInfo"                 <info>      (System)

/* Grupos Lisímetros */
Group   Lysimeters          "Lysimeters"               <plant>     (AllItems)

Group   Lysimeter1          "Lysimeter 1"              <plant>     (Lysimeters)
Group   Lysimeter2          "Lysimeter 2"              <plant>     (Lysimeters)
/*
Group   Lysimeter3          "Lysimeter 3"              <plant>     (Lysimeters)
Group   Lysimeter4          "Lysimeter 4"              <plant>     (Lysimeters)
*/

/* Grupos Válvulas */
Group   Valves              "Valves"                   <valve>     (AllItems)
Group   ValvesLys1          "Valves Lys. 1"         <valve>     (Valves)
Group   ValvesLys2          "Valves Lys. 2"         <valve>     (Valves)
/*
Group   ValvesLys3          "Valves Lys. 3"         <valve>     (Valves)
Group   ValvesLys4          "Valves Lys. 4"         <valve>     (Valves)
*/
Group   DrenajeGeneral      "Drain All"             <valve>     (AllItems)
Group   RiegoGeneral        "Irrigate All [30 min]"             <valve>     (AllItems)

Group   ConfRiego           "Irrigation Conf."                          (AllItems)
```

Primero una definición de grupos, donde se conectarán posteriormente otros *ítems*. En estas definiciones tendremos grupos de entradas y salidas. Se pueden utilizar para la visualización web, sin embargo, en el presente trabajo sólo se utilizan para agrupar objetos y visualizarlos mejor.

```
String  node1               "Node 1 [%d V]"
String  node2               "Node 2 [%d V]"
/*
String  node3               "Nodo 3 [MAP(en.map):%s]"
String  node4               "Nodo 4 [MAP(en.map):%s]"
*/

String  node_outputs        "Outputs Node [MAP(en.map):%s]"

Number  chart_period        ""
Number  ram_chart_period    ""
Number  disk_chart_period   ""
Number  lys_chart_period    ""
Number  weather_chart_period ""
```

A continuación varias definiciones de ítems útiles como los periodos de los cuadros y los ítems utilizados finalmente para la tensión de las baterías. No se incluyen los *binding* ni los grupos en los códigos mostrados dado que no se visualizaría bien. El código completo se haya en la configuración real.

Siguiendo con los ítems tenemos:

```
/* Modos de riego */
String  ZonaRiego            "Zone "              (ConfRiego)

String  ModoInicioRiego      "Irrigation Start"       (ConfRiego)
String  ModoFinRiego         "Irrigation End"             (ConfRiego)

Number  DiaInicio            "Starting day"               (ConfRiego)
Number  HoraInicio           "Starting time [%d:00]" (ConfRiego)
Number  WaterTimeInterval    "Irrigation interval"   (ConfRiego)
Number  NumRiegos            "Number of irrigations"      (ConfRiego)
//Number    IntervaloRiegos      "Intervalo entre riegos [%d Horas]"(ConfRiego)  //

String  HoraRiego1           "Starting time 1"   (ConfRiego)
String  HoraRiego2           "Starting time 2"   (ConfRiego)
String  HoraRiego3           "Starting time 3"   (ConfRiego)
String  HoraRiego4           "Starting time 4"   (ConfRiego)

Number  WeightPercent        "Weight decrease [%.1f Kg]" (ConfRiego)

Number  WaterTimeDuracion    "Run time"       (ConfRiego)
//String    UdDuracion           "Unidad"                 (ConfRiego)//NU

Number  DrainPercent         "Drainage variation [%d %% of PWFC]"     (ConfRiego)
Number  VoltoFin             "Water volume    [%.1f litros]"  (ConfRiego)
Switch  ActualizarConfRiego "Validate new conf."
Switch  SendConf             {mqtt=">[ArdMQTT:ArdMQTT/10/120/V_VAR3:command:ON:1]"}
```

Los ítems asociados a modos de riego, que se encontrarán duplicados para su envío simultáneo una vez establecidos los correctos.

```
/* PARAMETROS QUE ENVIAN REALMENTE */
String  ZonaRiegoENV            "Zone "              (ConfRiego)

String  ModoInicioRiegoENV      "Irrigation Start"       (ConfRiego)
String  ModoFinRiegoENV     "Irrigation End"             (ConfRiego)

//Number    DiaInicioENV            "Starting day"
//Number    HoraInicioENV           "Starting time [%d:00]"
Number  WaterTimeIntervalENV    "Irrigation interval"    (ConfRiego)
Number  NumRiegosENV            "Number of irrigations"      (ConfRi
//Number    IntervaloRiegos     "Intervalo entre riegos [%d Horas]"

String  HoraRiego1ENV           "Starting time 1"    (ConfRiego)
String  HoraRiego2ENV           "Starting time 2"    (ConfRiego)
String  HoraRiego3ENV           "Starting time 3"    (ConfRiego)
String  HoraRiego4ENV           "Starting time 4"    (ConfRiego)

String  WeightPercentENV        "Weight decrease [%.1f Kg]" (ConfRi

String  WaterTimeDuracionENV    "Run time"       (ConfRiego)
//String    UdDuracion          "Unidad"                 (ConfRiego)

String  DrainPercentENV     "Drainage variation [%d %% of PWFC]"
String  VoltoFinENV         "Water volume    [%.1f l]"    (ConfRiego)
```

Son por tanto éstos los que incluyen el *binding* MQTT que envía los datos al *Gateway*.

A continuación los correspondientes a la hora actual (Recordemos, el *sensor 150)* y los objetos asociados a los lisímetros:

```
/* Lysimeters */
/* Lysimeter 1 */
Number  W_1_L_1             "Weight 1 Lysimeter 1 [%.3f Kg]"    <weight>
Number  W_2_L_1             "Weight 2 Lysimeter 1 [%.3f Kg]"    <weight>
Number  W_3_L_1             "Weight 3 Lysimeter 1 [%.3f Kg]"    <weight>
Number  W_D_L_1             "Drainage Weight Lysimeter 1 [%.3f Kg]"
Contact DrenajeLys1         "Drainage Lysimeter 1 [MAP(en.map):%s]"
Number  Caudal1             "Irrigation Flow Lysimeter 1 [%.2f l/min]"
Number  VolumenRiego1       "Volume of water irrigated [%.3f l]"
Number  PesoMaxPlanta1      "Pot Weight Field Cap. [%.3f]"

/* Lysimeter 2 */
Number  W_1_L_2             "Weight 1 Lysimeter 2 [%.3f Kg]"    <weight>
Number  W_2_L_2             "Weight 2 Lysimeter 2 [%.3f Kg]"    <weight>
Number  W_3_L_2             "Weight 3 Lysimeter 2 [%.3f Kg]"    <weight>
Number  W_D_L_2             "Weight Drenaje Lysimeter 2 [%.3f Kg]"
Contact DrenajeLys2         "Drainage Lysimeter 2 [MAP(en.map):%s]"
Number  Caudal2             "Irrigation Flow Lysimeter 2 [%.2f l/min]"
Number  VolumenRiego2       "Volume of water irrigated [%.3f l]"
Number  PesoMaxPlanta2      "Pot Weight Field Cap. [%.3f]"
```

Se incluyen al final los ítems de riego manual y algunos útiles, como la suma de peso de las células de los lisímetros:

```
/* Outputs */
Switch  RiegoLys1           "Man. Irrigation Zone 1 [30 min]"       <valve>
Switch  RiegoLys2           "Man. Irrigation Zone 2 [30 min]"       <valve>
/*
Switch  RiegoLys3           "Man. Irrigation Zone 3 [30 min]"       <valve>
Switch  RiegoLys4           "Man. Irrigation Zone 4 [30 min]"       <valve>
*/
Switch  LedEncendido        "OpenhabStarted"            <light> { gpio="pin:4" }

/*Groups with functions*/
//Group:type:function(args) name            "label"
Group:Number:SUM            SUMLysimeter1   "Total Weight Lysimeter 1 [%.3f Kg]"
Group:Number:SUM            SUMLysimeter2   "Total Weight Lysimeter 2 [%.3f Kg]"
/*
Group:Number:SUM            SUMLysimeter3   "Weight Total Lisímetro 3 [%.3f Kg]"
Group:Number:SUM            SUMLysimeter4   "Weight Total Lisímetro 4 [%.3f Kg]"
*/
```

Si se observa el código implementado, aparecen los objetos asociados al clima, pero con sus *binding* comentados, de forma que si en un futuro se implementa un nodo climático, esté preparado el servidor.

4.5.2.7. OpenHAB. *Persistence*

El archivo *mysql.persist* definirá las reglas de persistencia de nuestros objetos, o lo que es lo mismo, qué objetos se almacenarán y cada cuánto tiempo.

Dichas reglas serán simples, primero definiremos las estrategias que podrán seguir:

```
Strategies {
    everyMinute : "0 * * * * ?"
    everyTenMins: "0 0/10 * * * ?"
    everyHour   : "0 0 * * * ?"
    everyDay    : "0 0 0 * * ?"
    default = everyChange
}
```

Una vez hecho esto, se aplican a los objetos:

```
Items {
    /* Restore on wakeUP */
    ConfRiego*:strategy=restoreOnStartup
    // persist all items
    * : strategy = everyChange
}
```

Se almacenará todo cambio en los objetos y en el caso de configuraciones de riego se restaurarán tras un reinicio.

4.5.2.8. OpenHAB. *Rules*

Las reglas que se definirán, en el documento *main.rules,* serán 4, y para ellas habrá que importar algunas librerías de java, en concreto las que permitirán enviar la fecha y hora con el formato adecuado:

```
import org.joda.time.*
import org.openhab.core.library.types.DateTimeType
```

A continuación se incluyen las reglas:

```
rule "ActualizaTime"
when
    Item node_outputs received update OK
then
    val long Ahora = now.plusHours(2).getMillis/1000
    sendCommand(Today,Ahora.toString)
end
```

Esta regla enviará el *timestamp* del tiempo actual al nodo de *salidas* cuando éste envíe la señal de *OK*, es decir, al terminar de inicializarse. Como respuesta, el controlador enviará la hora codificada en tiempo *unix* y como cadena de texto.

```
rule "FloatReset"
when
    Item ModoInicioRiego received update
then
    sendCommand(VoltoFin,5.0)
    sendCommand(WeightPercent,5.0)
end
```

La regla *FloatReset* reseteará los números en coma flotante del servidor web a un valor conocido, dado que con el tiempo, el valor del servidor variaba e inducía errores.

```
rule "ENVIO_REAL"
when
    Item ActualizarConfRiego received command   ON
then
    sendCommand(ActualizarConfRiego,OFF)
    ZonaRiegoENV.sendCommand(ZonaRiego.state.toString)
    ModoInicioRiegoENV.sendCommand(ModoInicioRiego.state.toString)
    ModoFinRiegoENV.sendCommand(ModoFinRiego.state.toString)
    //DiaInicioENV.sendCommand(DiaInicio.state.toString)
    //HoraInicioENV.sendCommand(HoraInicio.state.toString)
    WaterTimeIntervalENV.sendCommand(WaterTimeInterval.state.toString)
    NumRiegosENV.sendCommand(NumRiegos.state.toString)
    HoraRiego1ENV.sendCommand(HoraRiego1.state.toString)
    HoraRiego2ENV.sendCommand(HoraRiego2.state.toString)
    HoraRiego3ENV.sendCommand(HoraRiego3.state.toString)
    HoraRiego4ENV.sendCommand(HoraRiego4.state.toString)
    WeightPercentENV.sendCommand(WeightPercent.state.toString)
    WaterTimeDuracionENV.sendCommand(WaterTimeDuracion.state.toString)
    DrainPercentENV.sendCommand(DrainPercent.state.toString)
    VoltoFinENV.sendCommand(VoltoFin.state.toString)
    SendConf.sendCommand(ON)
end
```

ENVIO_REAL será la regla que actualice los valores enviados al actuador con los parámetros elegidos en el servidor. En lugar de enviar cada valor con cada cambio, que sería la forma por defecto, los cambios no se almacenan y se copia el valor final

únicamente. El sistema ha sido probado de ambas formas y el uso de esta regla es la que mejor resultado ha dado.

```
rule "LedAvisoEncendido"
when
    System started
    or
    Item LedEncendido received update OFF or Item LedEncendido received update Undefined
then
    LedEncendido.sendCommand(ON)
end
```

Por último, una pequeña regla que mantiene el LED del pin 4 encendido ante cualquier cambio a apagado o indefinido, de forma que si el LED se muestra apagado es posible que haya habido algún problema y haya que revisar el LOG de openHAB.

4.5.2.9. OpenHAB. *Transform*

Con el fin de visualizar el contenido en la web de la forma preferida, se ha creado un archivo de transformaciones, *en.map,* el cual sustituirá los valores indicados a la izquierda por los de la derecha, en los casos en los que se utilice dicha transformación:

```
CLOSED=closed
OPEN=open
undefined=unknown
-=OFF
Relays=OK
1=ON
0=OFF
ON=ON
OK=OK
OFF=OFF
```

4.5.2.10. OpenHAB. *Sitemap*

El archivo *main.sitemap* será el que defina la estructura y funcionamiento del servidor web, dado que será el documento leído por openHAB para generarlo. Para poder analizar correctamente la programación será necesario tener presente dicho servidor, por lo que comenzaremos analizando la estructura de la pantalla principal y sub-pantallas:

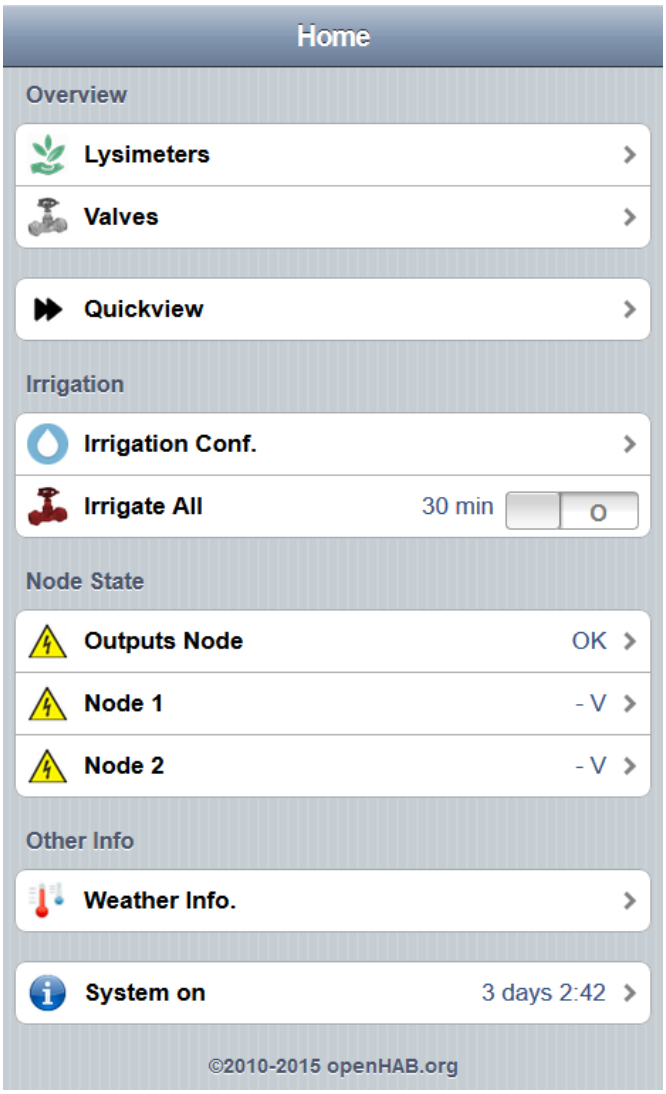

Ilustración 59 - Servidor web. *Main*.

En la ilustración de la izquierda se puede observar el menú principal, desde el cual podremos acceder a las distintas ventanas de visualización y configuración del sistema. La estructura estará dividida en cuatro partes: *Overview, Irrigation, Node State* y *Other Info*.

En el apartado *Overview,* existen tres sub-apartados: *Lysimeters,* que nos permitirá acceder a la visualización del estado de los lisímetros; *Valves,* que de forma similar al anterior nos permitirá ver el estado de las diferentes electroválvulas y activar de forma manual las de riego; Y por último, *Quickview*, que mostrará una sub-pantalla donde se verá un resumen de la información de todos los lisímetros.

En el siguiente apartado, *Irrigation*, habrá dos opciones: Acceder al menú de configuración de riego, e iniciar el riego de todas las zonas, con un límite de 30 minutos.

Posteriormente se puede observar el estado de los nodos "independientes", indicando el estado de tensión de las baterías de los nodos autónomos y si el nodo actuador mandó la señal de *OK*.

Por último, se dará acceso a información del clima y del sistema, viendo además en la de sistema el número de días, horas y minutos que el controlador lleva encendido.

A continuación, se muestran dichas pantallas en el orden definido.

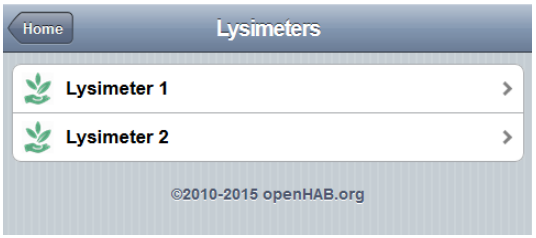

Ilustración 60 - Servidor web. *Lysimeters.*

Desde el menú *Lysimeters* se podrá acceder a los distintos lisímetros instalados. Dado que el sistema se encuentra preparado para dos, aparecen sólo dos. Seleccionando uno de ellos iremos a su pantalla:

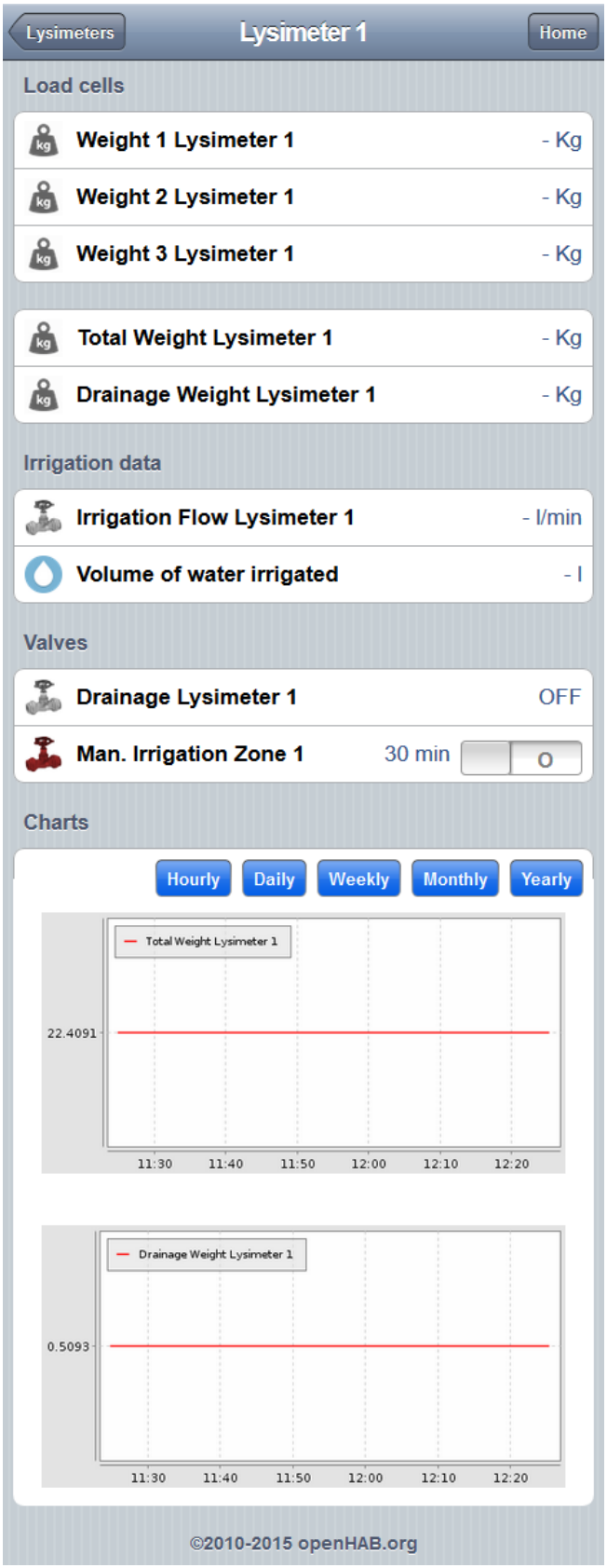

Ilustración 61 - Servidor web. *Lysimeter 1*

El correspondiente al lisímetro 2 no se incluye al ser exactamente igual al lisímetro 1.

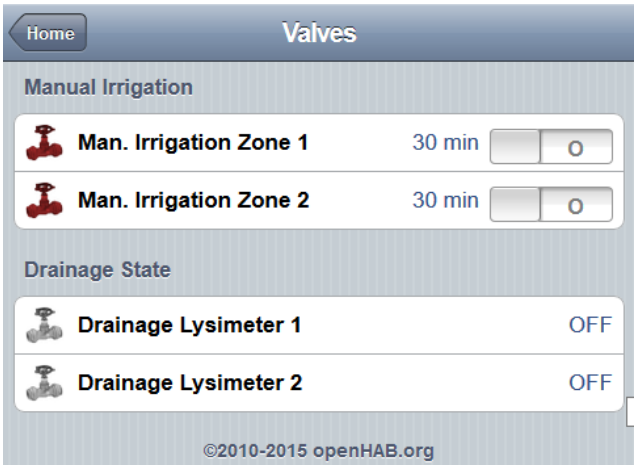

Ilustración 62 - Servidor web. *Valves.*

Desde el menú *valves* se visualizará el estado de todas las electroválvulas y se podrán comandar las relativas al riego manual, con la limitación de 30 minutos.

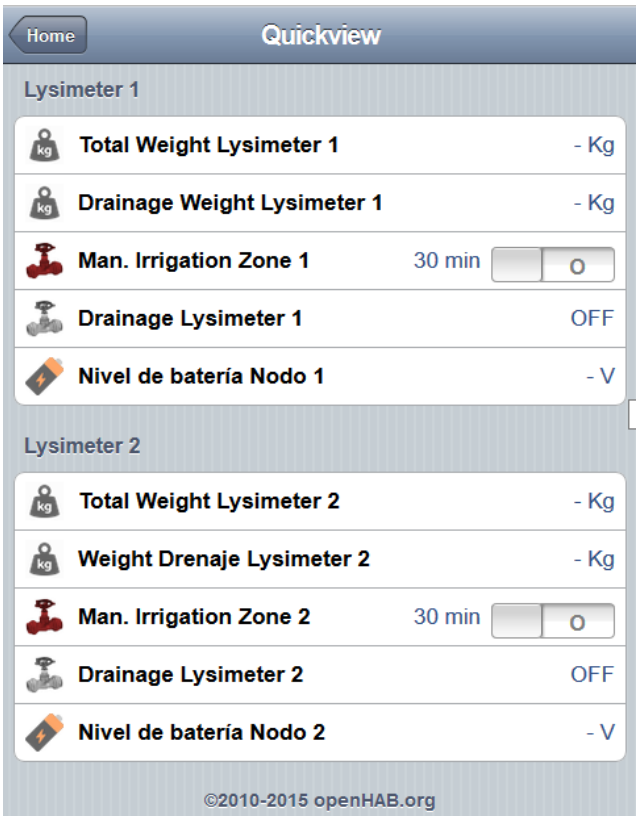

Ilustración 63 - Servidor web. *Quickview.*

Este menú mostrará tanto entradas como salidas de los nodos instalados.

El siguiente, será el más complejo, el de configuración de modos de riego, dentro del cual, dependiendo de los modos elegidos aparecerán unas configuraciones u otras. Existen hasta 6 configuraciones posibles, con dos de inicio y tres de fin de riego.

Clasificándolas según el modo inicial, para inicio lisimétrico tenemos las siguientes capturas:

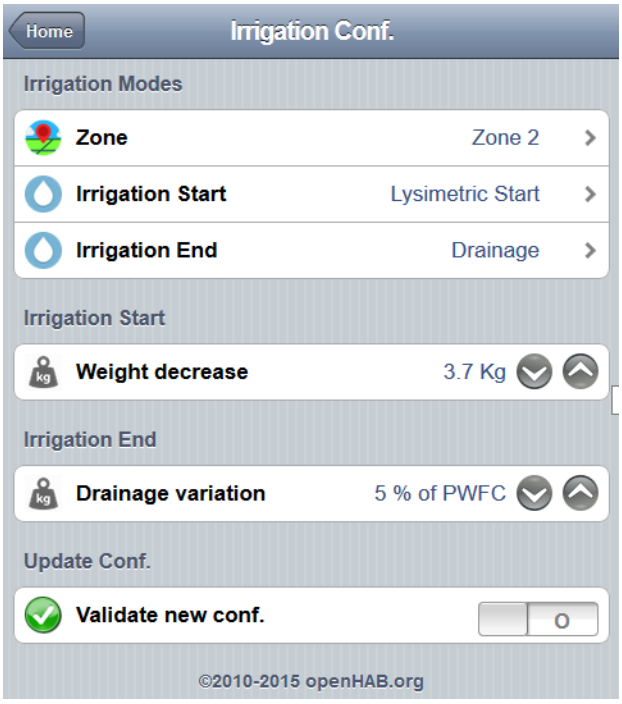

Ilustración 64 - Servidor web. *Irrigation conf 1.*

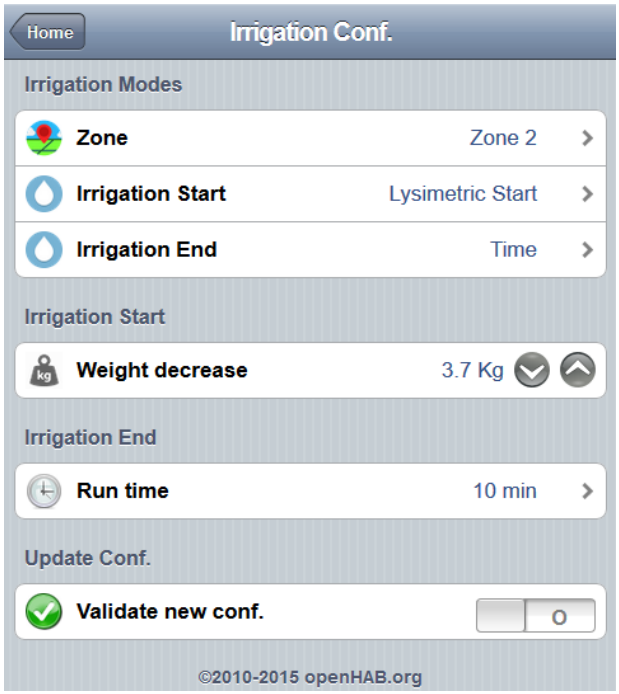

Ilustración 65 - Servidor web. *Irrigation conf 2.*

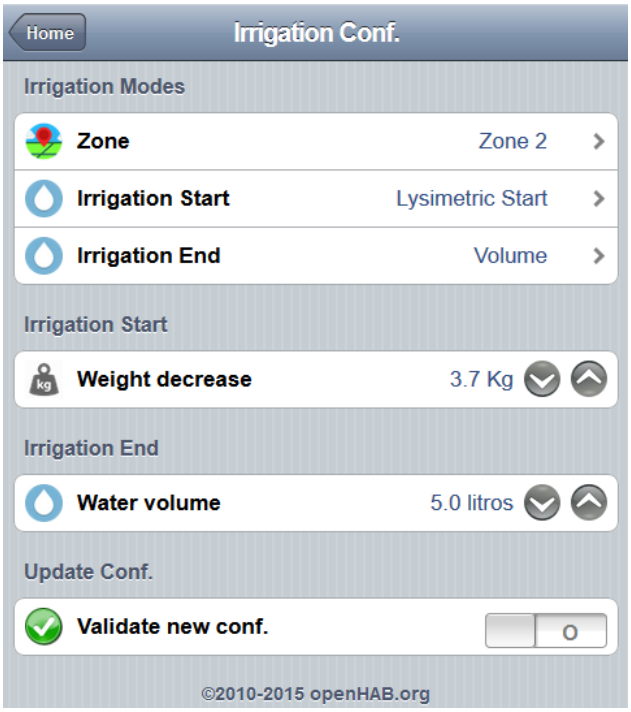

Ilustración 66 - Servidor web. *Irrigation conf 3.*

Para inicio temporizado tendremos las tres que aparecen a continuación:

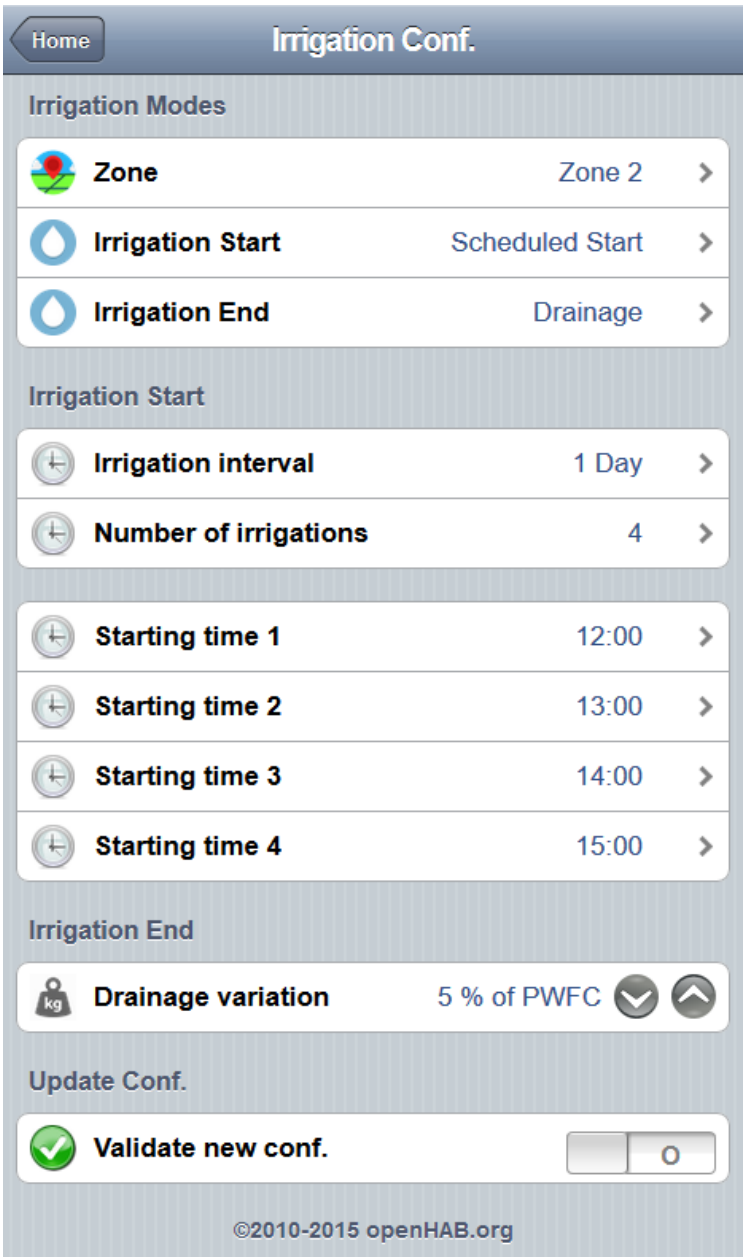

Ilustración 67 - Servidor web. *Irrigation conf 4.*

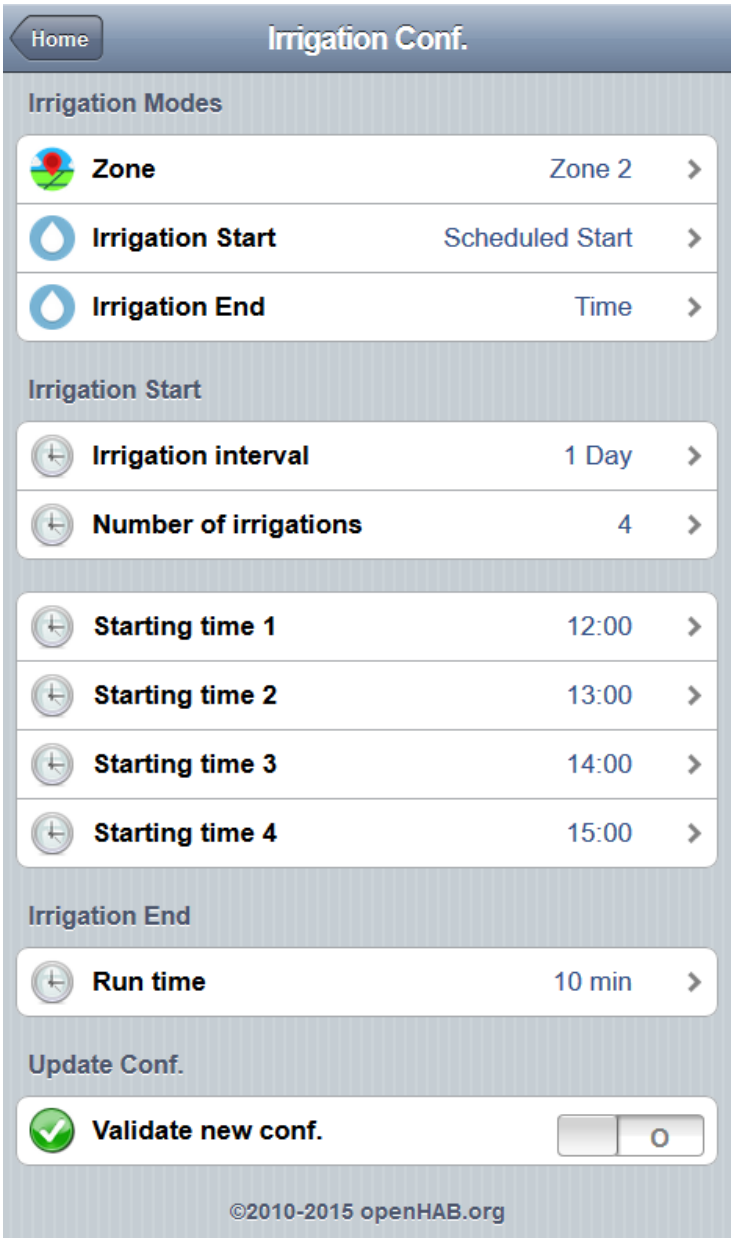

Ilustración 68 - Servidor web. *Irrigation conf 5.*

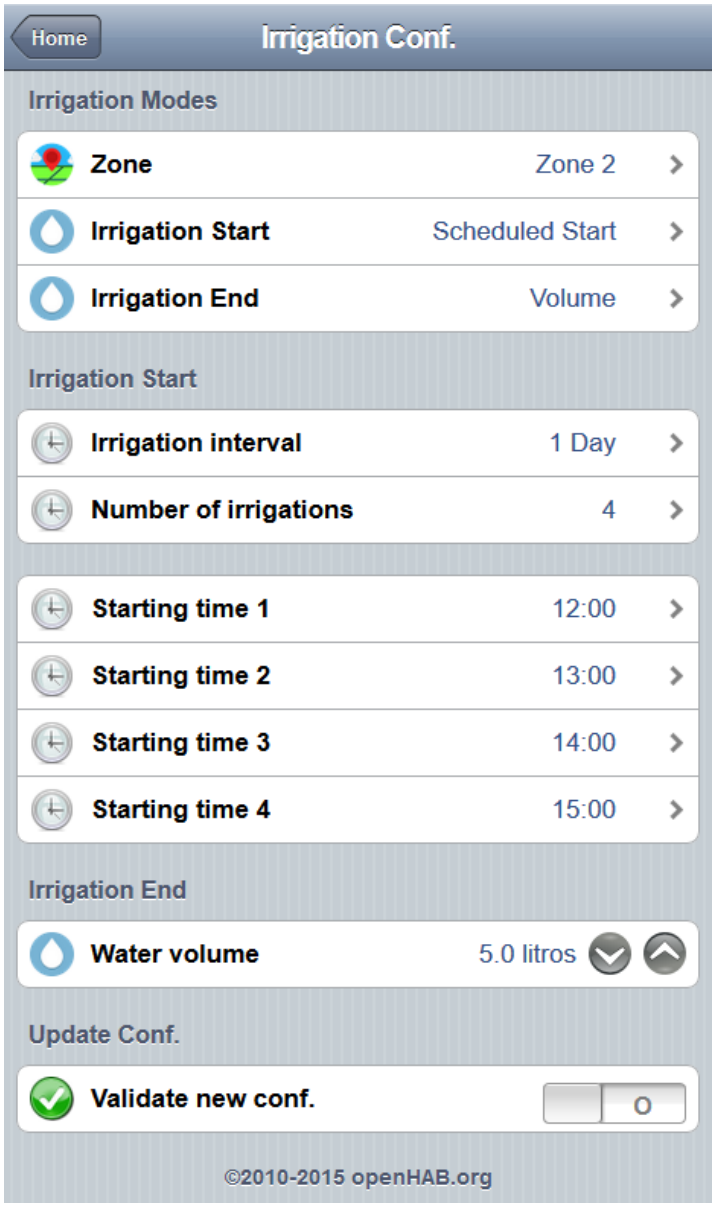

Ilustración 69 - Servidor web. *Irrigation conf 6.*

Con todas las posibilidades se podría establecer el riego según las necesidades del cultivo. Pudiendo elegir para cada una de las zonas una configuración totalmente diferente.

Continuando con el servidor web, tendríamos a continuación el estado de los nodos, donde pulsando obtendríamos una gráfica con los últimos valores de tensión.

Al final del todo se encuentra la información climática y de sistema, cuyas capturas se muestran a continuación:

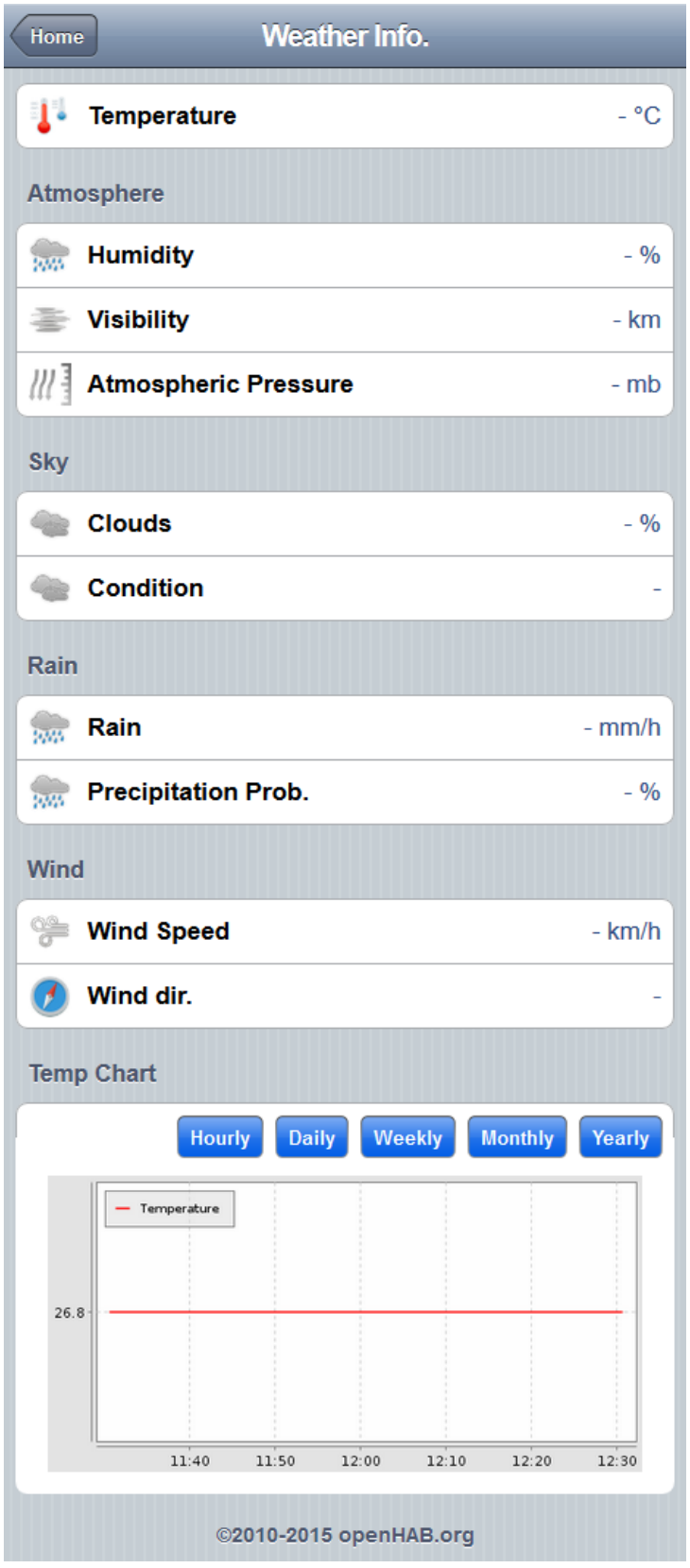

Ilustración 70 - Servidor web. *Weather Info.*

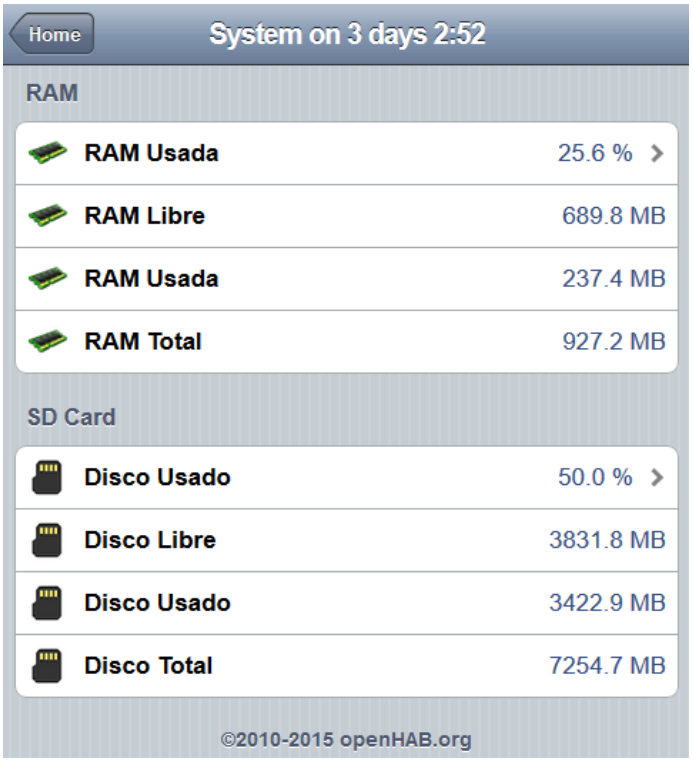

Ilustración 71 - Servidor web. *System Info.*

Con esta última captura queda definida la funcionalidad del servidor, con lo que ya es posible describir la programación. Ésta es bastante similar a los cuadros visualizados, es decir, la página principal estará compuesta por varios *Frames:*

```
sitemap main label="Home"
{
    Frame   label="Overview" {

    Frame {
    Frame label="Irrigation"{
    Frame label="Node State"{
/* Meteorología */
    Frame label="Other Info"{
/* Datos de sistema */
    Frame {
}
```

Los *Frame* sin *label* son aquellos que crean una ligera diferencia, sin parecer un cuadro totalmente diferente.

Dentro del cuadro *Overview* tendremos el ítem *Lysimeters* con los dos lisímetros instalados y el ítem *valves* con las electroválvulas:

```
Frame   label="Overview" {
    Text    item=Lysimeters icon="plant"{
        Text item=Lysimeter1{
        Text item=Lysimeter2{

    }
    Text    item=Valves  {
        Frame label="Manual Irrigation"{
            Switch item=RiegoLys1
            Switch item=RiegoLys2
            /*
            Switch item=RiegoLys3
            Switch item=RiegoLys4
            */
        }
        Frame label="Drainage State"{
            Text item=DrenajeLys1
            Text item=DrenajeLys2
            /*
            Text item=DrenajeLys3
            Text item=DrenajeLys4
            */
        }
    }

}
```

A continuación el cuadro sin nombre para el *quickview:*

```
Frame {
    Text label="Quickview" icon="fast"{
        Frame label="Lysimeter 1" {
            Text item=SUMLysimeter1
            Text item=W_D_L_1
            Switch item=RiegoLys1
            Text item=DrenajeLys1
            Text item=Battery1
        }
        Frame label="Lysimeter 2" {
            Text item=SUMLysimeter2
            Text item=W_D_L_2
            Switch item=RiegoLys2
            Text item=DrenajeLys2
            Text item=Battery2
        }
    }
}
```

En el cuadro *Irrigation* comenzará la programación para las opciones posibles, para ver el código abrir el archivo *main.sitemap* de la carpeta *configurations* de openHAB, en los documentos del presente trabajo.

Luego el estado de los nodos, la información climática (desactivada por su alto consumo de recursos) y la información del sistema.

Como se puede apreciar, normalmente es suficiente con escribir el ítem para su visualización, sin embargo, para los objetos seleccionables como las horas de riego, es necesario mapear todos los valores posibles, creando una larga lista con dichos valores.

Para poder crear configuraciones y visualizaciones diferentes se ha hecho uso de la opción *visibility*, que permite ocultar un objeto ante ciertas condiciones. La gran desventaja de usar este modo, es el alto consumo de recursos, dado que el objeto sigue ahí aun cuando no se está visualizando.

4.5.2.11. OpenHAB. App *Lysimeter*

Realizando pruebas se comprobó que el acceso desde un Smartphone suponía varios inconvenientes, por lo que se desarrolló una pequeña aplicación. La aplicación, creada con *App Inventor* de *Google*, no será más que un navegador web apuntando constantemente a la dirección del servidor instalado. Así se mejora considerablemente la interfaz y el rendimiento con dispositivos móviles, facilitando pues su uso.

Capítulo 5: Presupuesto

En este apartado se presentará el estudio de costes de materiales para el desarrollo del sistema de control propuesto.

El presupuesto se establecerá para el conjunto de la red de control y sistemas asociados. A continuación, se presentará dicho presupuesto para los materiales del proyecto, no incluidos gastos derivados de envíos, producción, mano de obra u otros componentes. Es decir, para la instalación de un producto completo, red de control unida a los lisímetros y en una localización final, habría que añadir al presupuesto aquí descrito, el precio de la estructura del lisímetro con las células de carga y de los componentes requeridos como electroválvulas, caudalímetros, cajas estancas, cableado y sistema de riego.

Componente	Cantidad	Precio Unitario	Total
Arduino UNO R3	2	3,50 €	7,00 €
Arduino Pro Mini 5V	1	1,50 €	1,50 €
Conversor DC/DC	1	1,50 €	1,50 €
Condensadores y resistencias*	1	1,00 €	1,00 €
StationBox	1	12,00 €	12,00 €
SunnyBuddy	1	25,00 €	25,00 €
Placa Solar	1	30,00 €	30,00 €
CS5534ASZ	1	11,00 €	11,00 €
Raspberry Pi y accesorios	1	50,00 €	50,00 €
Shield Ethernet	1	4,50 €	4,50 €
Batería	1	12,00 €	12,00 €
Relés Arduino	2	0,50 €	1,00 €
Módulos de radio	3	2,00 €	6,00 €
MOSFET canal P	4	2,00 €	8,00 €
TOTAL			170,50 €

*Respecto a resistencias y condensadores es difícil estimar un precio exacto dado que se utilizaron componentes del laboratorio, sin embargo, se aproxima el precio de todos los componentes. Habría que añadir asimismo el precio de fabricación de la placa.

Cabe destacar que los precios son aproximados dada la alta variabilidad de los mismos en el mercado, así como su disponibilidad.

Capítulo 6: Conclusiones, mejoras y optimización

En este apartado se presentarán las opciones de mejora consideradas para el presente trabajo, así como posibles optimizaciones que se podrían llevar a cabo.

Conclusión

Como conclusión, se puede afirmar que el uso de componentes *open-source* y *low-cost* consigue disminuir, de forma muy significativa, el coste total del proyecto, consiguiendo los objetivos planteados al inicio del mismo. Cabe destacar asimismo la facilidad de uso de la interfaz y su alta funcionalidad, permitiendo gestionar el riego de forma eficiente y operativa, pese a las limitaciones impuestas por los servicios utilizados. Por tanto, el sistema ha cumplido con los requisitos planteados en un principio, manteniendo el bajo precio como una ventaja, al conseguir un margen de mejora considerable con un ligero incremento del presupuesto.

En cuanto al tiempo de ejecución del proyecto, se pudo cumplir en su mayor parte, habiendo ocurrido retrasos tanto en la recepción de componentes, como en causas ajenas al proyecto que frenaron su desarrollo. La diferencia total, en tiempo de ejecución del proyecto ha sido de 4 semanas de retraso, siendo los recursos los previstos al inicio del mismo.

Futuras líneas de mejora

Dada la rápida evolución de los componentes actuales, todos los proyectos tienen una gran capacidad de mejora. Asimismo, consultando con ingenieros y agricultores, han aparecido posibles cambios en el sistema implementado. A continuación, se incluyen varias líneas de mejora consideradas:

Mejora del controlador

Con el fin de mejorar el rendimiento del controlador existe la posibilidad de actualizar éste a una Raspberry Pi 3, la cual mejora considerablemente su rendimiento respecto a su predecesora, o como alternativa, instalar el controlador en un servidor ajeno, permitiendo el acceso al mismo para el *Gateway*. Asimismo, haciendo uso del REST API de openHAB, se podría diseñar un nuevo servidor web más rápido e intuitivo.

Consumo y autonomía en nodo *sensor*

Eliminando LED innecesarios o desconectando el regulador de tensión integrado en Arduino, se podría reducir el consumo del nodo sensor. También sería posible minimizar el mismo, revisando los ciclos de sueño de los componentes e intentando ajustarlos al máximo. Además, dado el espacio libre en la placa, sería posible incorporar una batería con mayor capacidad, aumentando la autonomía del sistema.

En cuanto al puente de alimentación de las células, está preparado para ser alimentado con tensión controlada, disminuyendo el consumo global del nodo.

Diseño de un nodo climático

Con el fin de disminuir la carga de trabajo del controlador, se deshabilitó la descarga de parámetros climáticos desde APIs en internet. Para mantener información del estado del tiempo en la zona donde se instale el sistema, podría diseñarse un pequeño nodo que haga de estación meteorológica, es decir, que mida todas las variables

importantes como temperatura del aire y el suelo, velocidad del viento, humedad… y las envíe al controlador.

Ampliación a *n* zonas con *m* lisímetros

El diseño realizado está preparado para controlar dos zonas, de las cuales se utilizará la información de dos lisímetros, uno para cada una. Una gran línea de desarrollo sería la modificación del código permitiendo *n* zonas con *m* lisímetros, donde cada zona pueda tener información de más de un lisímetro o de ninguno, configurable por parámetros. *n y m* serían por tanto números diferentes y podrían ser uno mayor que el otro.

Mayor análisis y control de parámetros agrarios

Por último, y con el fin de mejorar el desarrollo de las plantas, se podrían modificar los nodos de forma que calculen y tomen datos de interés para el crecimiento de las plantas, como pueden ser análisis de sales, nutrientes o parámetros como la evapotranspiración de cultivo. Un sistema con dichos análisis podría gestionar de manera más eficiente el desarrollo del cultivo, y automatizarlo mejorando el estado general y evolución del mismo.

Capítulo 7: Bibliografía

En este capítulo se presentará la bibliografía consultada, así como enlaces de interés para el desarrollo del presente trabajo y otros trabajos de índole similar.

Allen, R. G., Pereira, L. S., Raes, D., & Smith, M. (1998). Crop evapotranspiration - Guidelines for computing crop water. *FAO - Food and Agriculture Organization of the United Nations*.

Arduino. (2016). *Arduino*. Obtenido de https://www.arduino.cc

Autodesk, Inc. (2016). *instructables*. Obtenido de http://www.instructables.com

DigitalOcean Inc. (s.f.). *DigitalOcean*. Obtenido de https://www.digitalocean.com

eLinux.org. (2016). *eLinux*. Obtenido de http://elinux.org

Evapotranspiración del cultivo. (2006). *FAO - ORGANIZACIÓN DE LAS NACIONES UNIDAS PARA LA AGRICULTURA*.

Geeky Theory. (2016). *Geeky Theory*. Obtenido de https://geekytheory.com

General Wireless Operations Inc. (2016). *RadioShack*. Obtenido de http://support.radioshack.com

GitHub, Inc. (2016). *GitHub*. Obtenido de https://github.com/

Grupo ADSL Zone. (2016). *RedesZone*. Obtenido de http://www.redeszone.net

Home Automation for geeks. (2015). *Home Automation for geeks*. Obtenido de http://www.homeautomationforgeeks.com

IoT Zone. (2016). *DZone*. Obtenido de https://dzone.com/

Liu, C., Zhang, X., & Zhang, Y. (2002). Determination of daily evaporation and evapotranspiration. *Agricultural and forest meteorology*.

ModMyPi LTD. (2012). *MODMYPI*. Obtenido de http://www.modmypi.com/

MQTT.org. (2015). *MQTT.org*. Obtenido de http://mqtt.org/

Nordic Semiconductor ASA. (2015). *NORDIC SEMICONDUCTOR*. Obtenido de http://www.nordicsemi.com/

Oracle Corporation. (2016). *MySQL*. Obtenido de https://dev.mysql.com

Philippe, & Javier. (2016). *Domótica Doméstica*. Obtenido de http://www.domoticadomestica.com/

Raspberry Pi Foundation. (2016). *Raspberry Pi*. Obtenido de https://www.raspberrypi.org/

Samba. (2016). *Samba*. Obtenido de https://www.samba.org/

Sastre, T. R. (s.f.). *ELECTROENSAIMADA*. Obtenido de http://www.electroensaimada.com

Shuttleworth, W. J. (Febrero de 2008). *SouthWest Hydrology*.

Stack Exchange Inc. (2016). *stackoverflow*. Obtenido de http://stackoverflow.com

Styger, E. (2016). *MCU on Eclipse*. Obtenido de https://mcuoneclipse.com

Terracota Inc. (2016). *QUARTZ Job Scheduler*. Obtenido de http://www.quartz-scheduler.org

Toro, L., Alejandro, Acosta, E., Castagnino, P., & Nano. (2016). *DesdeLinux*. Obtenido de http://blog.desdelinux.net

Vera-Repullo, J., Ruiz-Peñalver, L., Jiménez-Buendía, M., Rosillo, J., & Molina-Martínez, J. (2014). Software for the automatic control of irrigation using weighing-drainage lysimeters. *Agricultural Water Management*.

Anexo I: Comunicación

En este capítulo se presentarán algunas ideas sobre los tipos de comunicación más utilizados entre componentes.

Bus SPI

Del inglés *Serial Peripheral Interface*, es un estándar de comunicaciones basado en el sistema maestro-esclavo. Al ser un sistema de tipo Bus, minimiza el número de conexiones y cableado para su uso. Dado que es un protocolo síncrono, utiliza una señal de reloj que sincroniza la comunicación. Las señáles que utiliza para dicha comunicación son las siguientes:

- SCLK: Reloj que marca la sincronización, donde en cada pulso se lee/escribe un bit.
- MOSI (Master Output Slave Input): Traduciendo directamente del inglés, salida de datos del master y entrada de datos de los esclavos.
- MISO (Master Input Slave Output): El contrario al anterior, salida de datos de los esclavos y entrada de datos del maestro.
- SS/CS (*Chip Select*): Con el fin de seleccionar el esclavo con el que comunicar se hace uso de un bit señalador. El esclavo cuyo bit esté a cero estará escuchando/trasmitiendo. Si está a 1, no comunicará.

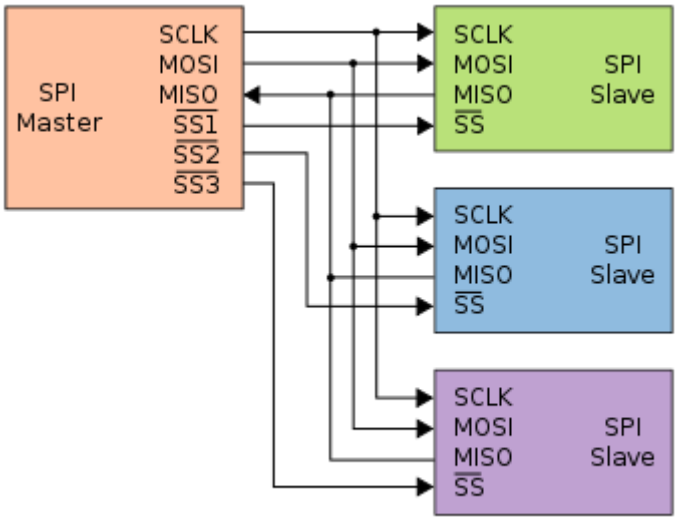

Ilustración 72 - Conexionado SPI (Cburnett, 2006)

Respecto a este sistema existen diversas configuraciones, a éstas se les llaman polaridad y fase de la trasmisión, e indican la forma de envío de los datos:

- Con flanco de subida sin retraso.
- Con flanco de bajada sin retraso.
- Con flanco de subida con retraso.
- Con flanco de bajada con retraso.

En el presente TFG, la comunicación SPI la gestionan las librerías de MySensors.org, por lo que simplemente se comprobará si los dispositivos a utilizar pueden comunicar con esa configuración.

Como principales ventajas ofrece comunicación Full Duplex (Ambas direcciones al mismo tiempo), buena velocidad trasmisión de datos (Mayor a I²C) y una señal única para cada esclavo, es decir, el CS. Por tanto para varios esclavos es una solución perfecta.

Como inconvenientes destaca el número de pines utilizados, dado que para un solo esclavo consume cuatro. Tampoco es bueno en distancias largas, por lo que su uso se reduce a las cortas y no tiene señal de asentimiento o *acknowledge*, es decir, el maestro puede enviar datos sin conocer si los esclavos los están recibiendo.

En Arduino hay que conocer la siguiente función:

SPI.transfer(DATA); Donde se pueden dar 2 casos (simultáneos o no) dependiendo de el valor de *DATA*.

En el caso de realizar: *SPI.transfer(oxoo);* Enviando un byte vacío, es decir dos ceros en hexadecimal, estaríamos realizando una lectura, por tanto el *return* de la función sería el dato importante.

En caso de enviar datos reales: *SPI.transfer(oxFF);* Enviando datos se realizaría tanto el envío como lectura, es decir, si enviamos un byte de datos, en el *return* tendremos además una lectura.

I²C

Diseñado, al igual que el anterior, como un bus maestro-esclavo. El protocolo utiliza dos líneas de comunicación, una de datos (SDA) y una de reloj (SCL), además requiere de unas resistencias pull-up entre las líneas y VDD. El bus trabaja con lógica positiva, es decir un nivel alto de tensión corresponde a un "1" mientras que un nivel bajo a un "0".

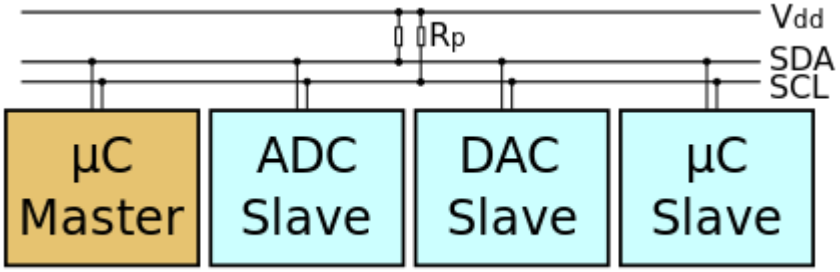

Ilustración 73 - Esquema I2C (Cburnett, 2006)

En cuanto a modos de trabajo, depende de una velocidad máxima de trasmisión. Para el modo más rápido, a 5 Mbit/s, la comunicación pasa a ser unidireccional. El direccionamiento se realiza con el primer byte enviado por el maestro.

Su principal ventaja es el mínimo uso de pines que conlleva. Sin embargo, su extremada simpleza hace que sea bastante susceptible a interferencias.

Puerto Serie (Arduino)

El puerto serie de Arduino está compuesto por dos pines principalmente, TX y RX. TX es utilizado para la trasmisión serie de datos, mientras que RX para la recepción. Los otros pines utilizados son alimentación y masa. Haciendo uso de esta comunicación, es posible comunicar Arduino con el ordenador, de forma que permite tanto programarlo como enviar y recibir mensajes. Dado que es una comunicación en serie, consisten en el envío de largas cadenas de "0" y "1", es decir, si los puertos están siendo utilizados para comunicar, y se conectase un led, éste se encontraría parpadeando la mayor parte del tiempo. Esto es importante, sobre todo en los casos en los que se quiera utilizar los pines RX y/o TX como pines normales, dado que, si se permite la comunicación, "ésta machacará" el valor dado al pin.

Respecto a velocidades, normalmente referidas en baudios, van desde los 75 baudios hasta los 115200. En Arduino los más comunes son 9600 y 115200 baudios. Los puertos serie clásicos, incorporan una serie de pines que facilitan la comunicación, como *Terminal Ready* o *Request to Send*, sin embargo, Arduino utiliza sólo los de trasmisión.

En las comunicaciones con PC se emulan los puertos serie a través de los puertos USB. Esta virtualización es la que permite la comunicación entre Arduino y un ordenador, al permitir virtualizar la velocidad en baudios, los bits de datos, paridad, etc.

Anexo II: *Getting Started:* Arduino

En este anexo se detallará la forma de instalación del IDE de Arduino y de las librerías necesarias para la programación de los distintos nodos utilizados.

Puesta en marcha

Para utilizar Arduino será necesario el uso de ciertos componentes, como serán un ordenador, cables USB y en ciertos casos una FTDI. Toda la información se puede encontrar en la página oficial de Arduino, Arduino.cc.

Instalación del software necesario

Para poder programar el controlador, una vez conectado utilizaremos un software específico. Dicho software será el IDE de Arduino (entorno de desarrollo integrado). Para instalarlo bastará con ir a la página web oficial del producto, https://www.arduino.cc/en/Main/Software, y descargar la versión adecuada al sistema operativo.

Para instalarlo bastará con seguir los pasos:

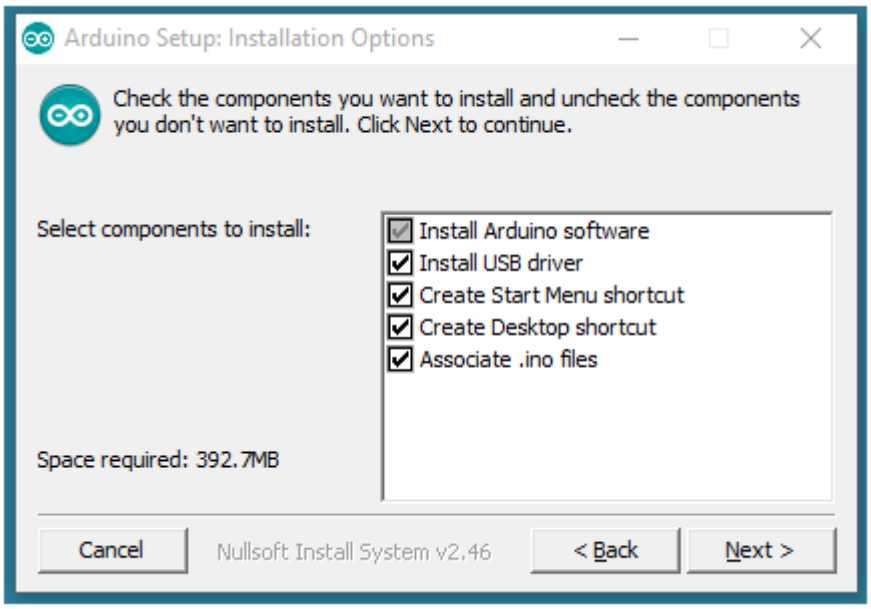

Ilustración 74 - Paso 1: Instalación IDE (Arduino.cc, 2016)

Elegir los componentes a instalar.

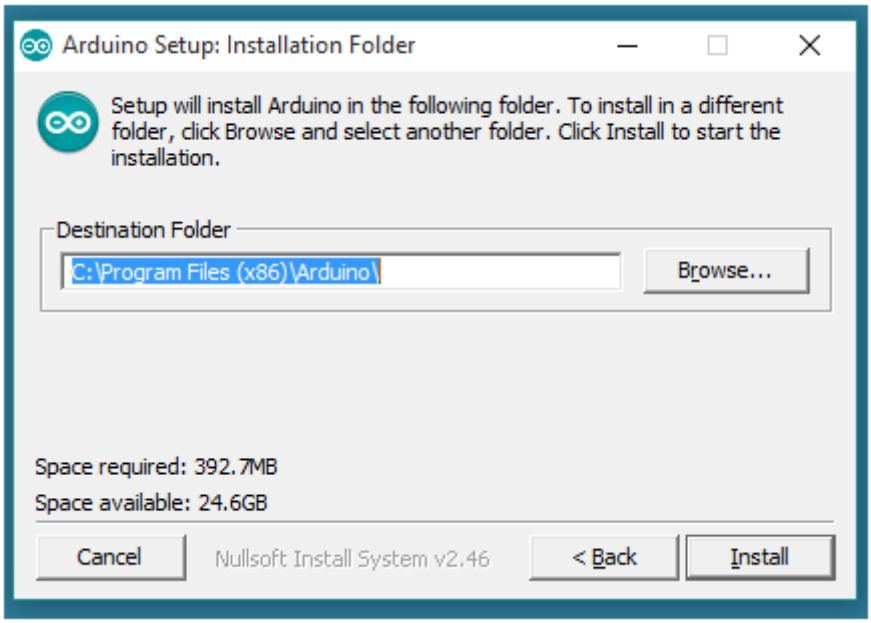

Ilustración 75 - Paso 2: Instalación IDE (Arduino.cc, 2016)

Elegir el directorio.

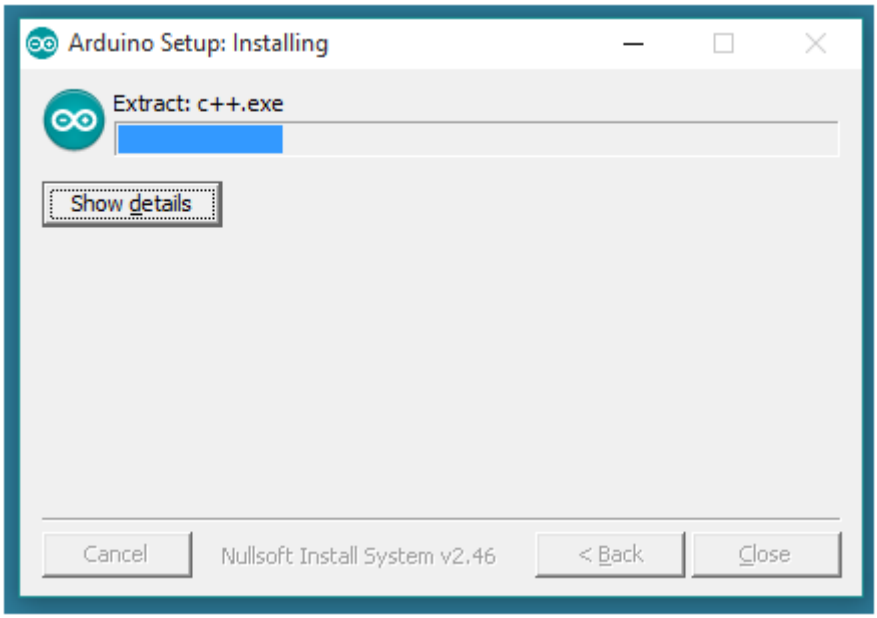

Ilustración 76 - Paso 3: Instalación IDE (Arduino.cc, 2016)

Y esperar a que finalice. Hecho esto es posible conectar Arduino y comenzar a programar.

Conexión de Arduino al PC

Dependiendo del tipo de Arduino a utilizar, la conexión al PC será directa con un cable USB o necesitará de un componente intermedio. Nos centraremos en los modelos utilizados en el presente trabajo, es decir, Arduino UNO R3 y Arduino Pro Mini 5V:

- En el caso de Arduino Uno R3, bastará con un cable USB de tipo B, es decir, los utilizados con impresoras. Los drivers se instalarán de forma automática en sistemas Windows con versiones posteriores a Windows XP.

Ilustración 77 - USB tipo B a USB tipo A (Arduino.cc, 2016)

- En el caso de Arduino Pro Mini 5V, dado que no incorpora un puerto USB, será necesario utilizar una interfaz que transforme el puerto serie del controlador a un USB. Para ello se hará uso de una FTDI, en este caso de 5V. El conexionado será el siguiente:

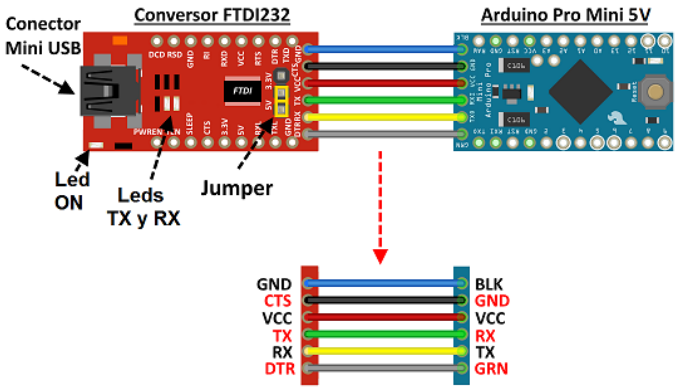

Ilustración 78 - Conexión con FTDI232

Uso del software

Una vez abierto el programa, basando en Processing, nos encontraremos la siguiente ventana:

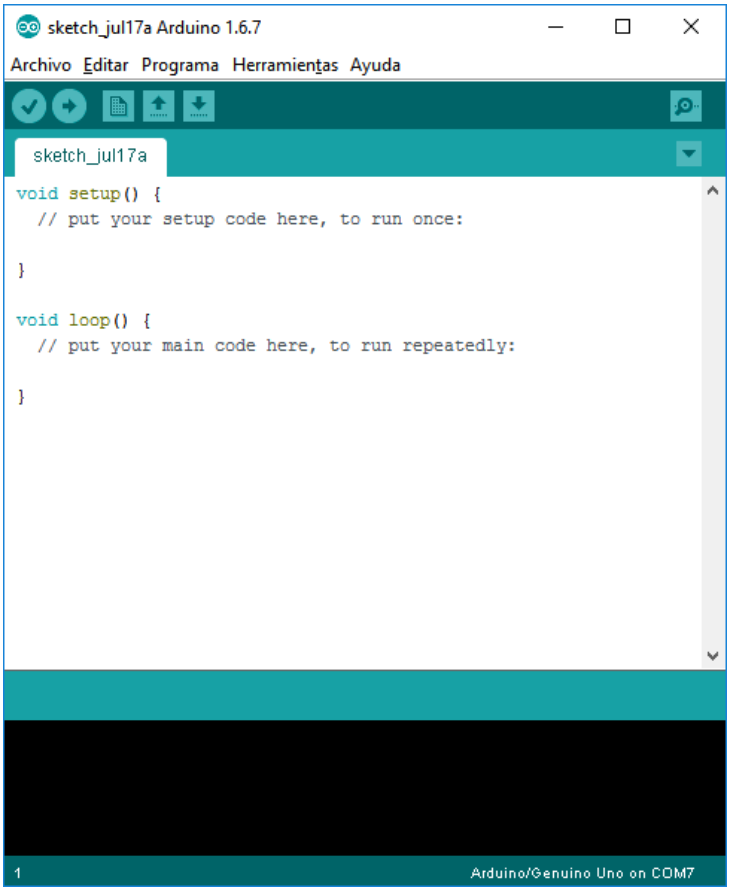

Ilustración 79 - Arduino IDE

Ésta será la ventana de programación, donde insertaremos el código que se desee implementar en el controlador. Para ello bastará con seleccionar el controlador correcto del desplegable *Herramientas* y pulsar en el botón con la flecha indicando a la derecha.

Para abrir la comunicación o monitor serie, habrá que pulsar el icono de la lupa a la derecha de la barra de herramientas.

Instalación y uso de librerías

Con el objetivo de ampliar las funcionalidades de Arduino, será necesario hacer uso de librerías de terceros, que permitan funciones de gestión de tiempo y comunicación con dispositivos. Para implementar las librerías, de forma que el software las reconozca y se puedan utilizar, existen dos métodos principalmente:

- Usar el gestor: El programa de Arduino incorpora un gestor de librerías, de forma que éstas se pueden descargar y actualizar desde internet. Para acceder al gestor, dentro del software, habrá que pulsar en *Programa, Incluir librería, Gestionar librerías*.
- Añadirlas manualmente: Al instalar el programa aparece, en la carpeta documentos, un directorio denominado *Arduino/Libraries*, dentro del cual bastará con añadir, sin comprimir, las librerías descargadas. En caso de no existir dicho directorio, el la carpeta de instalación del software

aparece un directorio similar, el cual incluye, por defecto, las librerías más utilizadas de Arduino.

En el presente trabajo se han utilizado las librerías de MySensors.org y otras incluidas por defecto con el software Arduino IDE.

Anexo III: Software Raspberry Pi

Con el fin de organizar el contenido, en este anexo se mostrará la instalación de los componentes necesarios para poner en funcionamiento el controlador con Raspberry Pi.

El uso de Raspberry Pi como controlador tiene grandes ventajas, entre las que destacan la gran comunidad que rodea al pequeño ordenador. Su uso, sin embargo, puede plantear algunas desventajas, como la necesidad de teclado y ratón para su uso, así como un monitor, o como alternativa el uso de protocolos como SSH.

En este apartado se explicará el uso e instalación del software y protocolos utilizados, para facilitar el uso de Raspberry Pi, poderlo controlar desde un ordenador y utilizar otras funciones, como un servidor de archivos.

Raspbian

El sistema operativo que se utilizará será Raspbian, una versión optimizada del famoso sistema, basado en Linux, Debian. Al ser Linux podremos utilizar la consola de comandos para realizar gran parte de las tareas.

Para instalar Raspbian existen dos formas principales, las cuales se comentan a continuación:

- Hacer uso de *NOOBS*, un conjunto de sistemas operativos para Raspberry Pi, desde el cual se puede elegir la instalación de Raspbian, u otros sistemas, tales como OpenELEC o RaspBMC.
- Es posible descargar directamente Raspbian e instalarlo, encontrando así una versión Lite que ocupa menos espacio en disco.

Ambas opciones se encuentran en la página oficial de Raspberry Pi, raspberrypi.org, desde la cual se ofrece además una guía de instalación.

MobaXterm

MobaXterm, es un software que ofrece todas las conexiones necesarias para comunicar con diversos dispositivos, entre sus modos de conexión destacan la posibilidad de ser servidor SSH, cliente SSH, herramientas de edición, cliente FTP…

En nuestro caso lo utilizaremos para conectar, desde Windows, vía SSH con Raspberry Pi y poder acceder a una consola de la misma, para ver un escritorio virtual de la misma, y para la transferencia de archivos vía SFTP.

La versión gratuita del mismo se puede encontrar en la página oficial de MobaTek, y ofrece las siguientes características:

- Full X server and SSH support
- Remote desktop (RDP, VNC, Xdmcp)
- Remote terminal (SSH, telnet, rlogin, Mosh)
- X11-Forwarding
- Automatic SFTP browser
- Plugins support
- Portable and installer versions
- Full documentation
- Max. 12 sessions
- Max. 2 SSH tunnels

- Max. 4 macros

Como se puede observar, ofrece todo lo necesario para una comunicación satisfactoria con nuestro controlador. Para instalarlo bastará con bajarlo de la página oficial y seguir las instrucciones.

Una vez instalado, o abierto como portable, aparece la siguiente ventana:

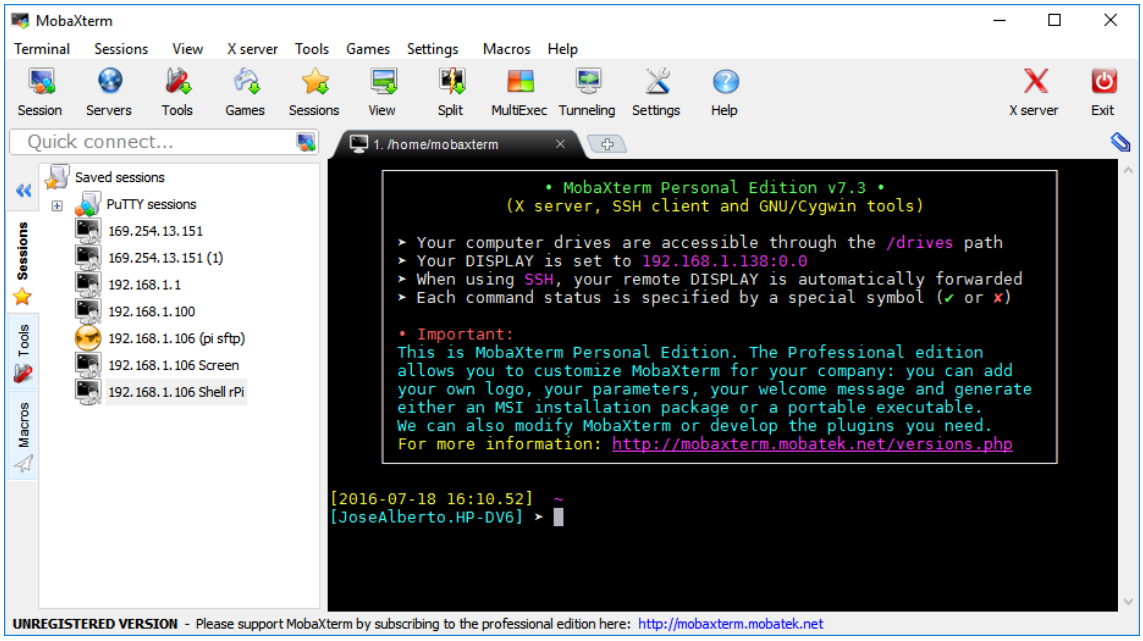

Ilustración 80 - Ventana principal MobaXterm

Desde esta ventana se podrán crear sesiones nuevas, abrir sesiones antiguas, o modificar las opciones del programa, así como muchas otras características que incluye. En nuestro caso utilizaremos las más comunes.

Si pulsamos en *Session* aparecerá una ventana emergente con las nuevas opciones, que servirán para definir una nueva conexión:

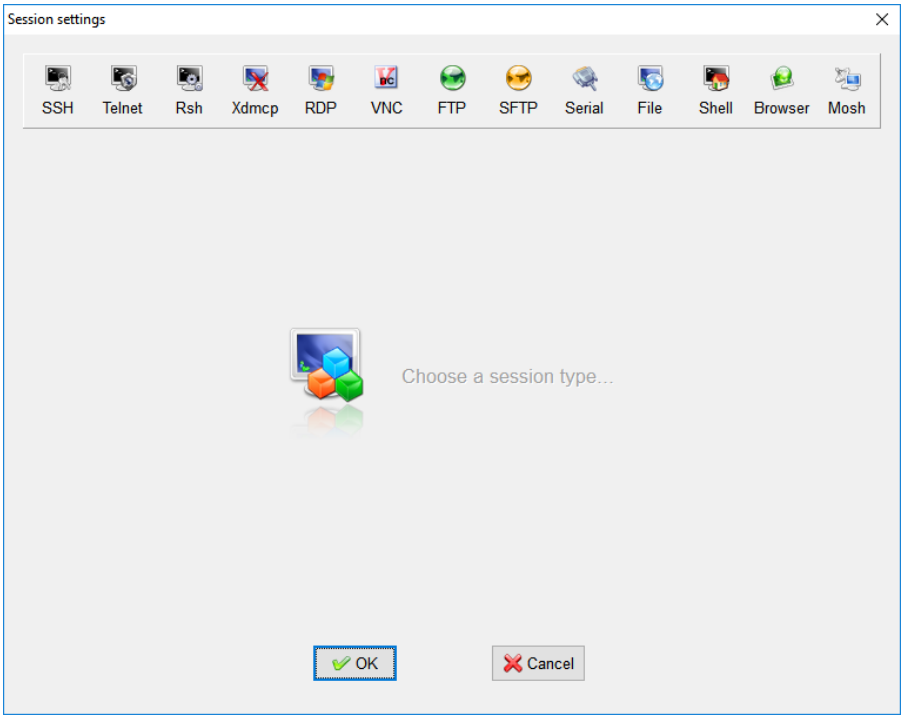

Ilustración 81 - Opciones de sesión, MobaXterm.

Haciendo clic ahora en SSH tendremos casi todas las opciones necesarias para comunicar con Raspberry Pi. A continuación se muestra un ejemplo para abrir una vía de comunicación y mostrarla como un escritorio virtual, bastará con establecer la dirección IP de la Raspberry Pi:

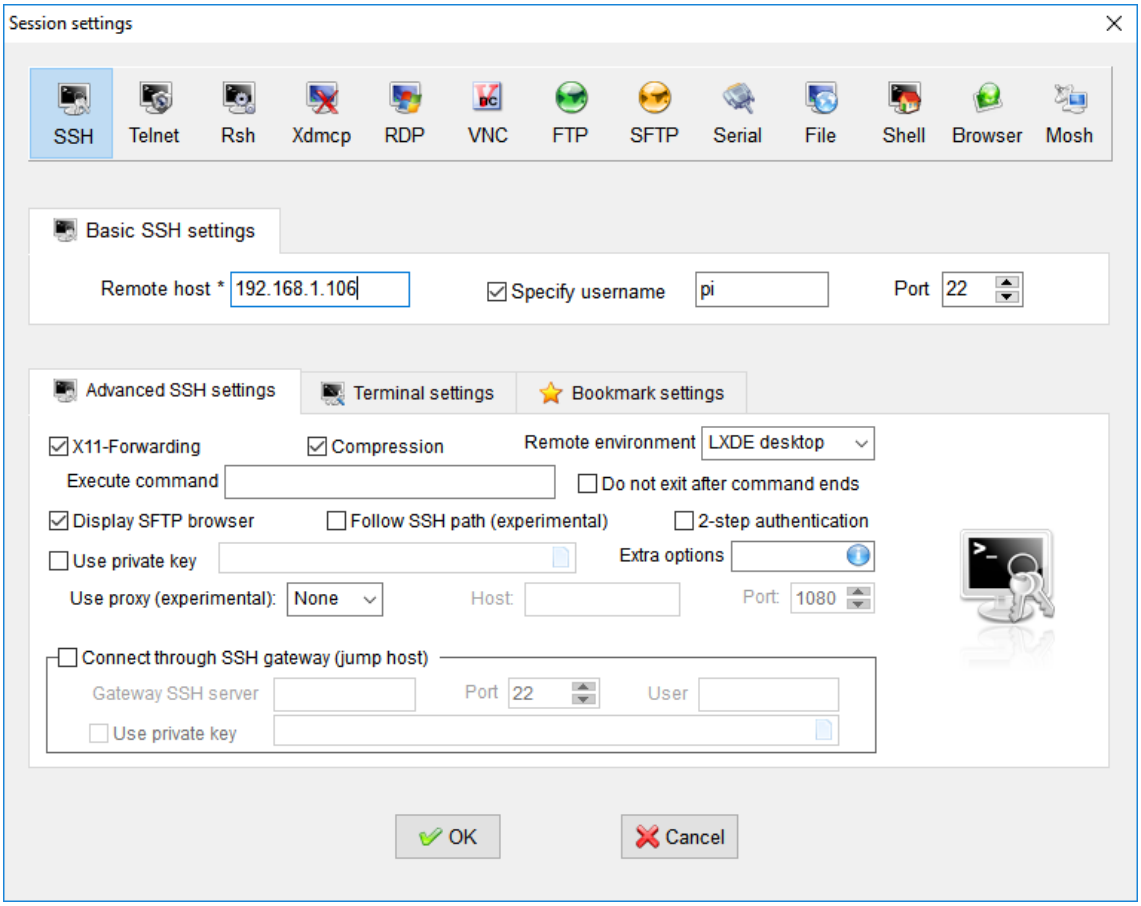

Ilustración 82 - Ejemplo de configuración para escritorio LXDE, MobaXterm.

En caso de preferir una comunicación estándar por SSH habrá que cambiar la opción *Remote environment* a *Interactive Shell*. Si se abre la comunicación con un *Shell* inteactivo, tendremos además de forma automática una conexión por SFTP, mostrada a la izquierda de la consola.

Si queremos exclusivamente la comunicación SFTP, podríamos utilizar la siguiente configuración:

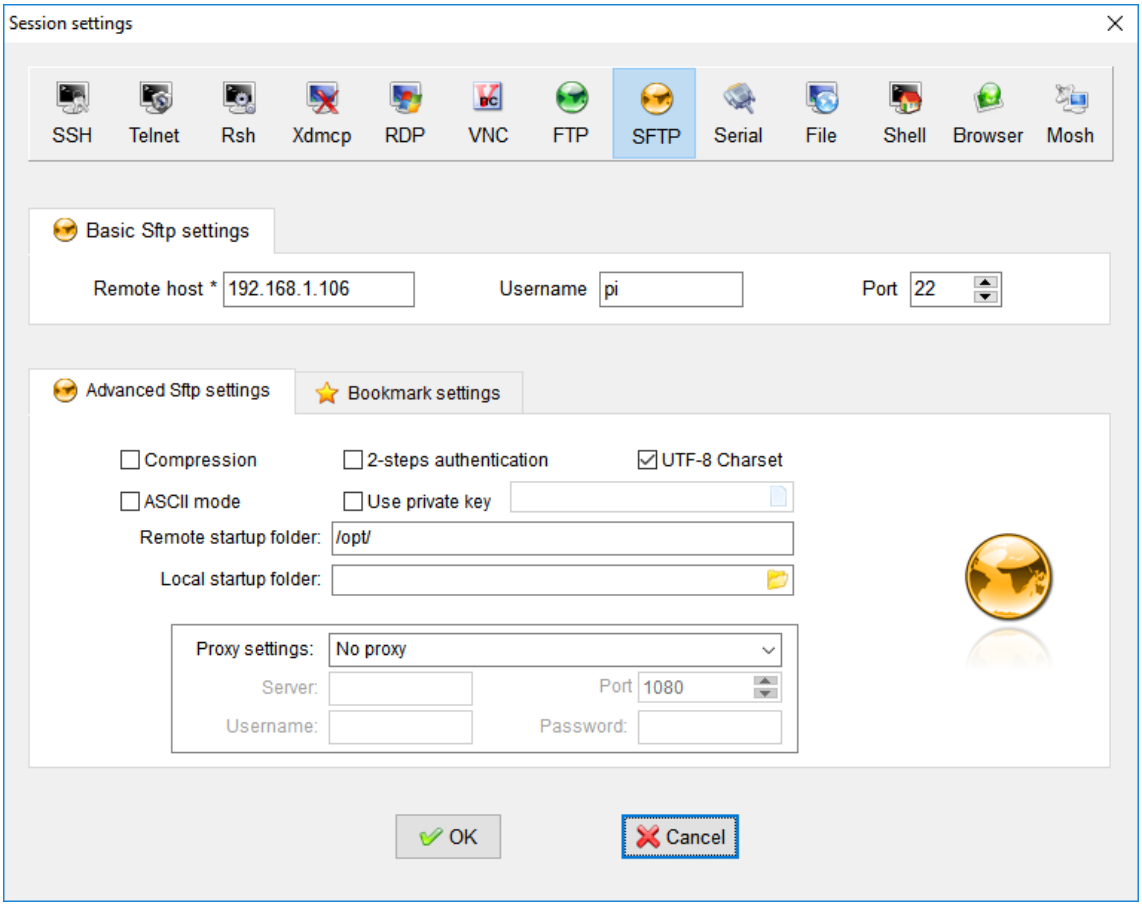

Ilustración 83 - SFTP con MobaXterm

Es importante destacar, que vía SFTP podremos acceder sólo a las carpetas a las que se tenga permiso, por lo que establecer los permisos adecuados será primordial.

Samba

Continuando con la comunicación y control de Raspberry Pi desde otro ordenador, con Windows en este caso, será muy útil crear un servidor Samba, que permita compartir los directorios de Raspberry Pi en la red local. Así será posible modificar los archivos y configuraciones deseados sin necesidad de abrir otros programas, bastará con usar los directorios como si de directorios locales se tratase.

A diferencia de con el software anterior, para poder utilizar un servidor Samba, habrá que instalar éste primero. En el caso de SSH, viene instalado por defecto. Para instalar el servidor Samba y dar permisos de acceso a Windows habrá que seguir una serie de pasos:

1. En primer lugar instalar los paquetes necesarios, para ello escribir en consola: *sudo apt-get install samba samba-common-bin*
2. Instalados los paquetes, hay que modificar la configuración: *sudo nano /etc/samba/smb.conf*. En dicho archivo modificar las siguientes líneas:
    a. workgroup = *GRUPODERED (puede ser cualquiera mientras coincida con el del resto de la red)*

   b. wins support = yes. Si aparece comentada con "#", descomentarla.
3. Ahora, dentro del mismo archivo, hay que crear las definiciones de carpetas compartidas, es decir, indicar qué se va a compartir. Para ello iremos al apartado *Share Definitions*. Copiaremos la siguiente estructura:

   [Nombredelperfil]
   comment = *Comentario*
   path = /opt/ *Carpeta a compartir*
   browseable = yes *Carpeta navegable*
   writeable = yes C*arpeta de lectura/escritura*
   only guest = no *Autenticación*
   create mask = 0777 *Permisos Linux*
   directory mask = 0777 *Permisos Linux*
   public = no *Carpeta pública*

4. Creado el perfil, se guarda el archivo. Es momento de crear un usuario para poder conectar a dicha carpeta. Escribimos *sudo smbpasswd –a pi* seguido de la contraseña dos veces.
5. Hecho esto se reinicia el servicio con: *sudo service samba restart*
6. En este momento se podrá acceder al directorio desde la red de Windows, como si de una carpeta local más se tratase, es decir, se podrá usar con el software del PC.
7. Para agregar más directorios, repetir los pasos 3 y 5.

Crontab

Cron es un administrador de procesos en segundo plano (*daemon*), que ejecuta procesos o guiones a intervalos regulares. Viene instalado por defecto en Linux, y su funcionamiento consiste en revisar cada minuto una tabla de tareas, almacenada, en Raspberry Pi, en el directorio */etc/* y se denomina *crontab*.

Crontab es por tanto un archivo de texto plano en el que se guardan las tareas que se realizarán en ciertos momentos indicados. El archivo *crontab* dentro del directorio */etc/* corresponde a la tabla del usuario *root,* que será la que utilicemos.

La tabla tiene la siguiente forma: *m h dom mon dow user command* donde:

- **m** corresponde al minuto en que se va a ejecutar el script, va de 0 a 59
- **h** la hora exacta, en formato 24 horas, de 0 a 23.
- **dom** hace referencia al día del mes, por ejemplo se puede especificar 15 si se quiere ejecutar cada día 15.
- **dow** significa el día de la semana, puede ser numérico (0 a 7, donde 0 y 7 son domingo) o las 3 primeras letras del día en inglés: mon, tue, wed, thu, fri, sat, sun.
- **user** define el usuario que va a ejecutar el comando, puede ser root.
- **command** se refiere al comando o a la ruta absoluta del script a ejecutar, ejemplo: */home/usuario/scripts/elquesea.sh*

De esta forma se pueden definir acciones concretas en momentos concretos. Se puede usar además el símbolo asterisco, "*", para denotar cualquier momento que

cumpla las condiciones. Es decir, se puede establecer un "*" en día del mes, para que todos los días a cierta hora realice la tarea indicada.

MySQL server

Para poder almacenar datos e información existen varios métodos. Uno de ellos es una base de datos, por ejemplo en MySQL, que permita el uso de comandos y permisos. Sin embargo, antes de usar una base de datos en MySQL habrá que instalar ésta y configurarla.

Instalación

Lo primero será por tanto la instalación de la base de datos, e instalaremos asimismo *bindings* para poder utilizar Python con MySQL:

sudo apt-get install mysql-server python-mysqldb

Durante la instalación, se pedirá una contraseña para el usuario *root*, a la que pondremos *root*.

Hecho esto estaremos en disposición de crear bases de datos y modificarlas.

Login en consola

Podremos crear asimismo usuarios con acceso a las distintas bases de datos. Para hacer *login*, una vez abierta la consola, habrá que escribir lo siguiente:

Mysql –u [usuario] –p[contraseña] Para *root: mysql –u root –proot*

De esta forma accederemos a MySQL y podremos trabajar sobre el servidor.

Crear usuarios

Con el fin de permitir a ciertos usuarios el acceso a ciertas tablas, habrá que definir a estos usuarios. Para ello escribir lo siguiente, una vez abierto MySQL:

CREATE USER 'user'@'localhost' IDENTIFIED BY 'password';

Hecho esto, habrá que proporcionarle permisos al usuario *user:*

GRANT ALL PRIVILEGES ON database.table TO 'user'@'localhost';

Y por último, para que los cambios surtan efecto:

FLUSH PRIVILEGES;

Así el usuario *user* tendrá acceso con su contraseña *password* a la tabla *tables* de la base de datos *database*. Otras opciones, como eliminar usuarios o cambiar contraseñas se pueden realizar desde la misma consola, en la bibliografía se dejarán enlaces de interés.

OpenHAB

Se ha decidido dejar la instalación de openHAB aparte para facilitar la lectura en el capítulo en el que se explica su uso.

Java Runtime

Para instalar openHAB será necesario tener, en el sistema a instalarlo, una versión de *java runtime* superior a 1.7. Por defecto en todas las versiones de Raspbian, viene incluido, por lo que bastará con comprobar la versión, en caso de existir la duda. Para comprobarlo bastará con escribir, en la consola:

java –version

Y si todo es correcto, nos devolverá la información sobre la versión instalada, de una forma similar a lo siguiente:

```
pi@raspberrypi ~ $ java -version
java version "1.8.0"
Java(TM) SE Runtime Environment (build 1.8.0-b132)
Java HotSpot(TM) Client VM (build 25.0-b70, mixed mode)
```

Instalación openHAB

Una vez se ha comprobado si en efecto java se encuentra instalado, es posible continuar con la instalación de openHAB, que será muy sencilla. Bastará con descargar en *runtime* desde la página oficial de openHAB, y descomprimir el contenido en el directorio */opt/openhab* de Raspberry Pi.

Hecho esto habrá que navegar al directorio y dentro de la carpeta *configurations* se encontrará un archivo denominado *openhab_default.cfg*. Este archivo será el "modelo" de archivo de configuración, bastará con realizar una copia y renombrarla como *openhab.cfg* para terminar. Por supuesto, en el presente trabajo ya se ha realizado dicha tarea y el archivo *openhab.cfg* incluye todas las configuraciones necesarias.

Instalación *bindings*

Tal y como se comentó en el capítulo correspondiente, la versatilidad le viene dada a openHAB por los *bindings* instalados, por lo que a continuación se explicará cómo instalarlos:

1. Descargar los archivos *binding* de configuración desde la página oficial de openHAB.
2. Descomprimir todos los archivos en una carpeta cualquiera, no tiene por qué ser dentro de openHAB.
3. Seleccionar, de dicha carpeta, los *binding* que se desean instalar y copiarlos dentro de la carpeta *addons* de openHAB.
4. Algunos *binding* necesitarán de una configuración especial dentro del archivo *openhab.cfg* mencionado anteriormente.

NOTA: Un mayor número de *bindings* instalados, implica un rendimiento mucho menor de openHAB, por lo que hay que mantener los *binding* justos y necesarios para el funcionamiento deseado.

Passwords

Durante la ejecución del presente trabajo, se han ido estableciendo contraseñas para diferentes apartados, desde la propia Raspberry Pi hasta el servidor MySQL, a continuación se recopilan todas ellas, en el caso de necesitarlas:

SERVICIO	USER	PASSWD
Raspberry Pi	*pi*	*raspberry*
MySQL	*root*	*root*
MySQL	*openhab*	*openhab*
Samba	*pi*	*raspberry*

Anexo IV: Software Eagle

Breve presentación de las capacidades y uso de dicho software, utilizado para el desarrollo de la placa.

Qué es Eagle

Eagle es un software de tipo CAD para el diseño de diagramas electrónicos y PCBs, desarrollado por la empresa CadSoft Computer. Una de las grandes ventajas que ofrece es una licencia *freeware*, que en un tamaño limitado permite la funcionalidad casi del total del programa. Entre las limitaciones destacan dos: Un espacio de trabajo muy limitado (100x80mm), y un máximo de 2 capas en una PCB.

En cuanto a su funcionamiento es muy sencillo, basta con colocar los componentes y conectarlos, establecer el *layout,* y conectar las pistas. Además ofrece una función de *auto-routing,* la cual permite, bajo ciertas condiciones, establecer el camino para las pistas, evitando hacer este trabajo de forma manual.

Al abrir el software aparece una ventana de panel de control, desde la cual se gestionan tanto librerías como proyectos.

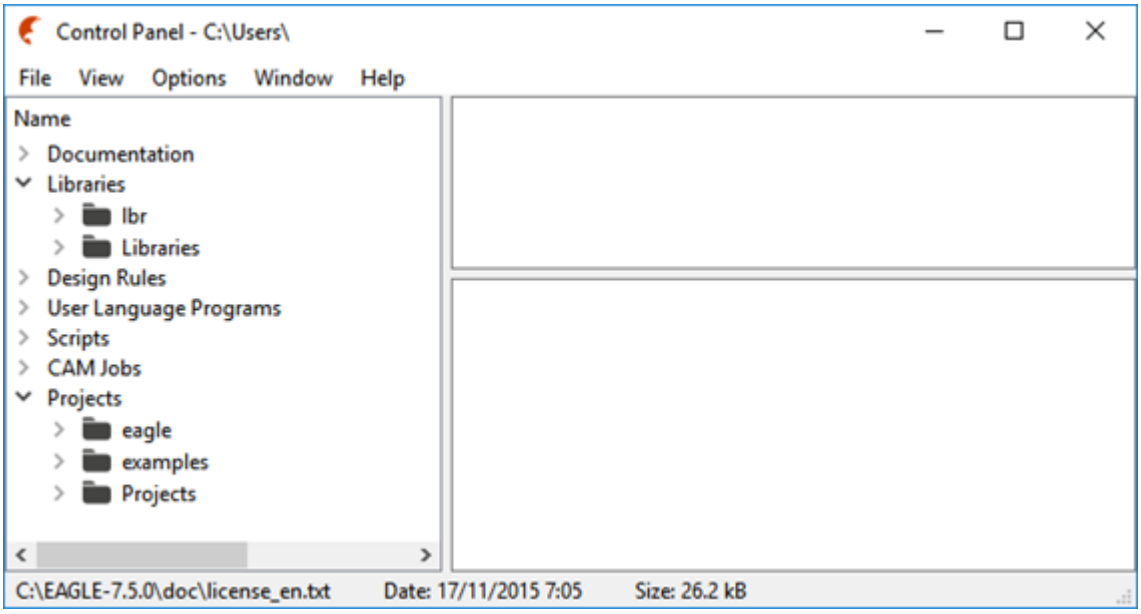

Ilustración 84 - Panel de control de Eagle

Desde este panel podemos crear un proyecto nuevo, un esquemático, una librería... Dentro de un proyecto normalmente se tienen esquemáticos y *layout*, es decir, la disposición de los componentes en una placa.

Diseño de esquemáticos

Una vez creado un proyecto, es posible crear uno, o varios, esquemáticos para lo cual se abre una ventana nueva:

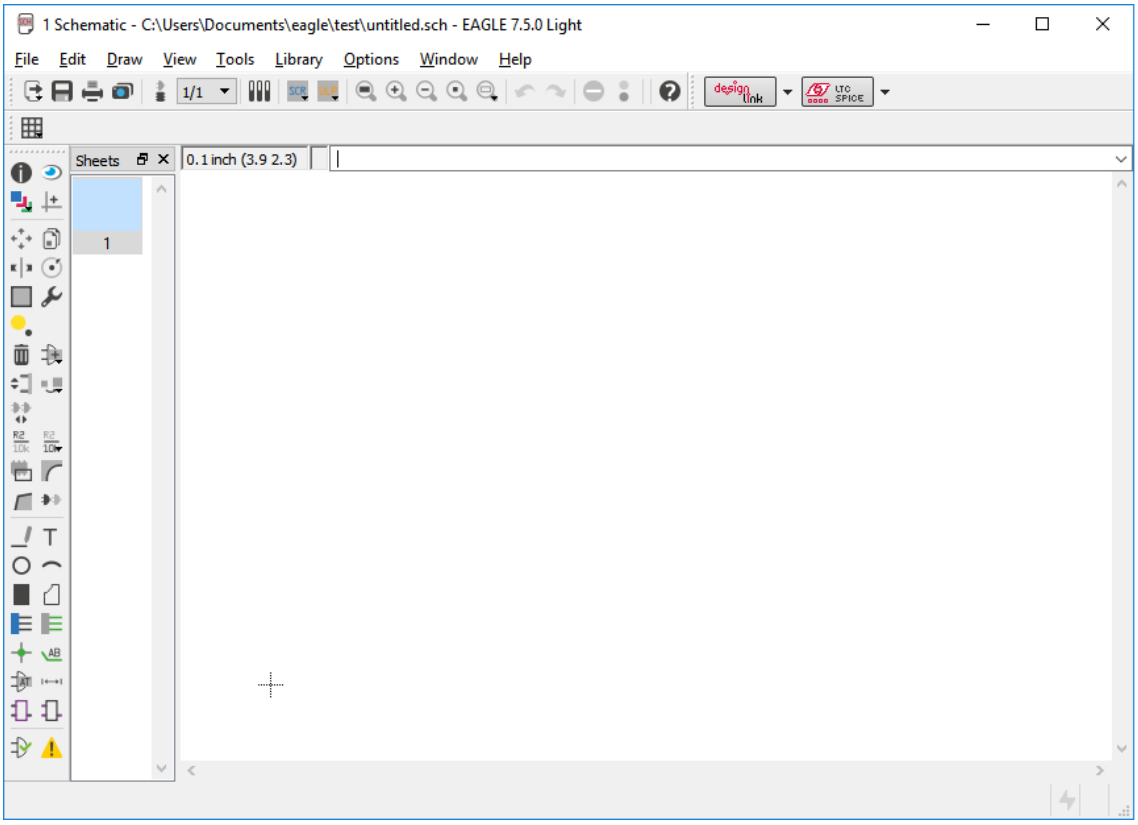

Ilustración 85 - Pantalla de edición de esquemáticos, Eagle.

En esta ventana aparecen todas las herramientas necesarias para el diseño del diagrama, desde herramientas de *espejo* o bus, hasta una herramienta simple de *Mover* o *Borrar*. Haciendo uso de la barra de herramientas, a la izquierda, y de las opciones que ofrece, es posible realizar circuitos de alta complejidad, incluso con la versión gratuita del programa.

Una vez diseñado el circuito basta con hacer clic en el botón *Generate board,* a la izquierda de la escala, para entrar en el modo de diseño de placa.

Diseño de PCB

Pulsando el botón mencionado en el apartado anterior, se abre una nueva ventana, con los componentes insertados en el apartado anterior, de forma que simplemente bastará con arrastrarlos a una zona de trabajo y conectarlos como va indicando el software.

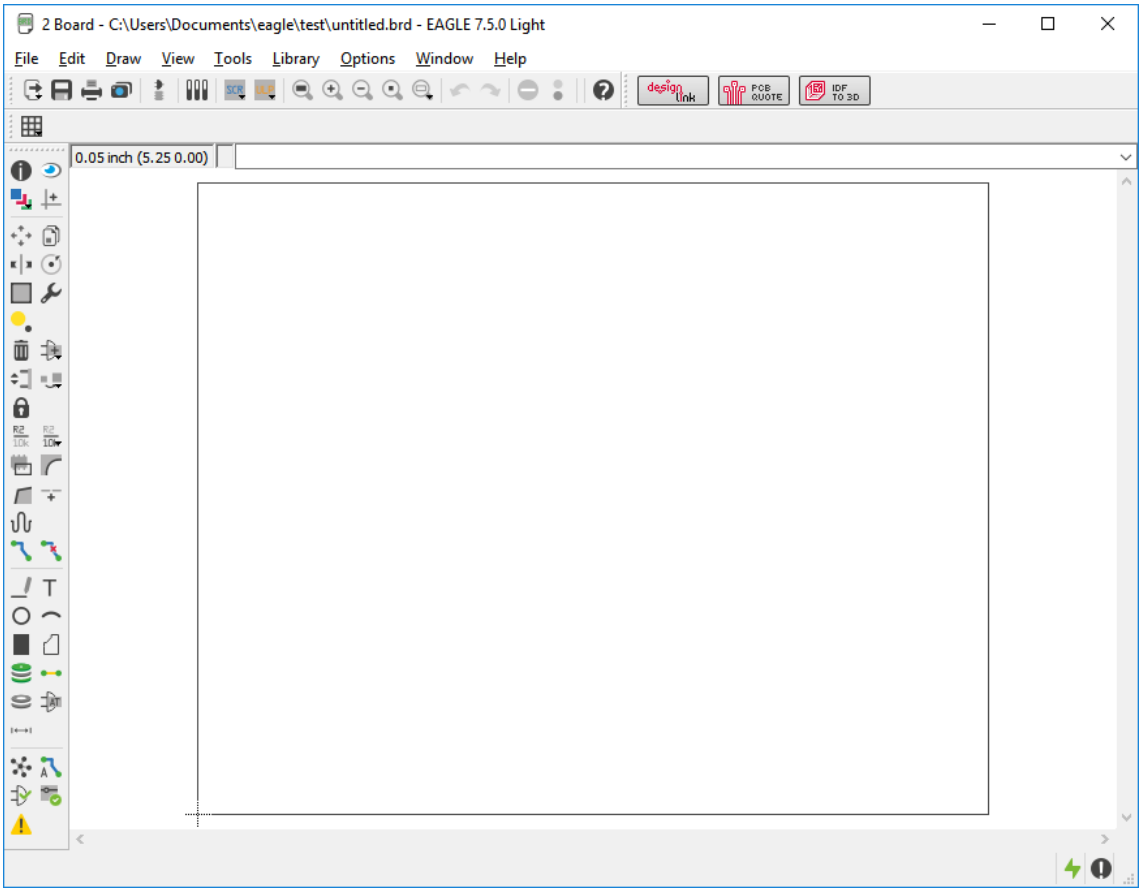

Ilustración 86 - Pantalla de diseño de PCB, Eagle.

Al igual que en el caso anterior, para facilitar el diseño, incluye una serie de herramientas a la izquierda, dichas herramientas permiten el conexionado con pistas, cable aéreo, puentes o vías de los componentes integrados en el proyecto, o la ya mencionada *auto-routing*, que basándose en todas las anteriores y en la posición actual de los componentes, crea dichas conexiones. Es en este paso en el que las limitaciones se hacen palpables, al recortar el tamaño para las conexiones de pistas a 100 x 80 mm, tamaño escaso para ciertas aplicaciones.

Instalación y uso de librerías

En el caso de querer utilizar componentes poco comunes o no comerciales, será necesario el uso de librerías, ya sean de creación propia o de terceros. En el trabajo que se presente se ha tenido que utilizar de ambos tipos. A continuación se explica cómo instalar librerías de terceros:

1. El primer paso será descargar las librerías, normalmente en formato .zip, es decir, comprimidas.
2. Hecho esto hay que descomprimirlas dentro de la carpeta /lbr/ del directorio de instalación de *Eagle*. Realizando esto, aparecerán en el software para su uso.
3. De no ser así, dentro del proyecto es posible añadir las librerías a mano, sin importar el directorio en el que se encuentren. Para ello bastará con ir al menú superior de la página de diseño de esquemático, y pulsar en

Library, donde aparecerá la opción de *Open* y *Use,* con las cuales se podrán abrir las librerías sin importar donde se encuentren, siempre que no estén comprimidas.

Existen librerías para casi todos los productos comerciales, un buen ejemplo son las librerías que ofrece *SparkFun*, que incluyen gran contenido de elementos comerciales, así como librerías de productos que la empresa vende. Existen además tutoriales en internet acerca de la creación de librerías propias, para componentes no comerciales, o de los cuales no se encuentren librerías. Para el presente proyecto se han utilizado las ya mencionadas librerías de *SparkFun* junto a algunas diseñadas manualmente. Todas ellas se pueden encontrar en la carpeta del proyecto.

Anexo V: Funciones CS5534

En el presente anexo se mostrarán y explicarán las funciones y parámetros necesarios para el correcto control del conversor CS5534ASZ, dada la ausencia de una librería preexistente.

Con el objetivo de facilitar el uso del conversor seleccionado, es recomendable la creación de funciones que gestionen dicho componente. Para utilizar el conversor es necesario escribir unas secuencias de bytes por SPI, a las cuales el conversor reacciona devolviendo una respuesta. Todos los registros y configuraciones se pueden encontrar en la hoja de datos del componente, no obstante aquí se detallarán las utilizadas.

Los registros de 4 bytes se escribirán en de byte en byte, esperando al final una confirmación o la respuesta.

Registros y constantes

El primer paso antes de crear las funciones, será definir todos los registros y comandos a enviar, de forma que posteriormente sea más eficiente el diseño de dichas funciones.

```
//CHANNEL SETUP REGISTERS 4x32
//SETUP 1 y 2 Channel 1 y 2 Valores para ganancia 64 unipolar 6.25 Sps REG1x32
const byte CHANNELREG11 = 0b00110010; //Setup1
const byte CHANNELREG12 = 0b01000000; //Setup1
const byte CHANNELREG13 = 0b01110010; //Setup2
const byte CHANNELREG14 = 0b01000001; //Setup2
//SETUP 3 y 4 Channel 3 y 4 Valores para ganancia 64 unipolar 6.25 Sps REG 1x32
const byte CHANNELREG21 = 0b10110010; //Setup3
const byte CHANNELREG22 = 0b01000010; //Setup3
const byte CHANNELREG23 = 0b11110010; //Setup4
const byte CHANNELREG24 = 0b01000011; //Setup4
//SETUP 5,6 & 7,8 not used
```

Lo primero será definir las opciones o *setups* que podrá utilizar el conversor. Este conversor ofrece hasta 8 configuraciones de lectura, de las cuales configuraremos 4. La configuración será la que aparece en el código, un *setup* para cada canal de células de carga, con la máxima ganancia, 64, y la transferencia más rápida (aunque menos precisa) de los datos.

```
//CONFIGURATION REGISTER 1x32 ESTANDAR, SIN SLEEP
const byte CONFIGREG11 = 0b10000100; //Página 26 Datasheet
const byte CONFIGREG12 = 0b00001000; //No hay que cambiarlos
const byte CONFIGREG13 = 0b00000000; //NULL (not used)
const byte CONFIGREG14 = 0b00000000; //NULL (not used)
//CONFIGURATION REGISTER 1x32 ESTANDAR, SLEEP MODE
const byte CONFIGREG21 = 0b11000100; //Página 26 Datasheet
const byte CONFIGREG22 = 0b00001000; //No hay que cambiarlos
const byte CONFIGREG23 = 0b00000000; //NULL (not used)
const byte CONFIGREG24 = 0b00000000; //NULL (not used)
//CONFIGURATION REGISTER 1x32 RESET MODE
const byte CONF_RESET1 = 0b00100000;
const byte CONF_RESET2 = 0b00000000;
const byte CONF_RESET3 = 0b00000000;
const byte CONF_RESET4 = 0b00000000;
```

Posteriormente, definir las configuraciones en cuanto a modos de trabajo. Se definen tres: Modo estándar, es decir, de lectura; Modo sueño, o ahorro de batería; Y por último, el modo *RESET*, utilizado únicamente para reiniciar el dispositivo.

Tras esto, queda definir los comandos que se enviarán al conversor:

```
const byte COMMAND_CONFGREG = 0b00000011;  //ESCRIBIR EN CONF REG
const byte COMMAND_READCONF = 0b00001011;  //LEER DE CONF REG
const byte COMM_GAIN_1 = 0b00000010; //WRITE GAIN REG.1
const byte COMM_GAIN_2 = 0b00010010; //WRITE GAIN REG.2
const byte COMM_GAIN_3 = 0b00100010; //WRITE GAIN REG.3
const byte COMM_GAIN_4 = 0b00110010; //WRITE GAIN REG.4
const byte COMM_OFFSET_1 = 0b00000001; //WRITE OFFSET REG.1
const byte COMM_OFFSET_2 = 0b00010001; //WRITE OFFSET REG.2
const byte COMM_OFFSET_3 = 0b00100001; //WRITE OFFSET REG.3
const byte COMM_OFFSET_4 = 0b00110001; //WRITE OFFSET REG.4
const byte COMMAND_CHANNELREG1 = 0b00000101; //ESCRIBIR EN CHANNEL 1 SETUP 1 y 2
const byte COMMAND_CHANNELREG2 = 0b00010101; //ESCRIBIR EN CHANNEL 2 SETUP 3 y 4
//const byte COMMAND_CHANNELREG3 = 0b00100101; //ESCRIBIR EN CHANNEL 3 SETUP 5 y 6
//const byte COMMAND_CHANNELREG4 = 0b00110101; //ESCRIBIR EN CHANNEL 4 SETUP 7 y 8
const byte READ11 = 0b10000000; //Conversión sencilla con Setup1 Pagina 20 Datasheet
const byte READ21 = 0b10001000; //Conversión sencilla con Setup2
const byte READ31 = 0b10010000; //Conversión sencilla con Setup3
const byte READ41 = 0b10011000; //Conversión sencilla con Setup4
```

Se definirán los comandos:

- Escribir en el registro de configuración
- Leer el registro de configuración
- Escribir la ganancia de los registros 1, 2, 3 y 4 (No utilizado)
- Escribir el offset de los registros 1, 2, 3 y 4 (No utilizado)
- Escribir la configuración de los *setups* 1, 2, 3 y 4
- Leer los canales de 1, 2, 3 y 4 con conversión sencilla.

La conversión sencilla implica que el conversor tomará medidas de las células hasta que éstas se estabilicen y responderá con la medida correcta. Existe la alternativa de una medida en continuo, pero es eficiente sólo en el caso de leer un canal de forma prolongada, y en el caso presentado se leen los 4 canales de forma puntual.

Funciones creadas

Una vez definidos los registros y comandos a utilizar, es posible comenzar a diseñar las funciones que los utilizarán.

read_CH (int ch)

La función más utilizada. Se encargará de realizar la lectura del canal indicado, así como de realizar la transformación de los datos leídos, siguiendo una recta de calibración, a datos reales (Kg). Para facilitar su entendimiento dividiremos la función en varias partes, de principio a fin:

```
float read_CH(int ch){
  digitalWrite(CSN_AD,LOW); //Activa comm. AD
  byte inByte = 0;           // incoming byte from the SPI for the SUM
  byte cola_conversion = 0;  //LEE LOS 8 de over-range y channel indicator
  unsigned long result = 0;   // result to return 32bits, MSB 0x00 24 result
  unsigned long result_blank = 0;   // SDO CLEAR
  float CHgain = 0.0;
  float CHoffset = 0.0;
  float FINALRESULT=0;
```

Al comienzo de la función, lo primero será establecer el pin CS del conversor a *LOW*, de forma que podamos iniciar la comunicación. Definiremos asimismo las variables necesarias para la lectura y conversión.

```
switch(ch){
  case 1:
    SPI.transfer(READ11); //HERE YOU CAN MODIFY THE GAIN AND OFFSET OF CHANNEL 1
    CHgain = 0.006966561;
    CHoffset = -175.25;
  break;
  case 2:
    SPI.transfer(READ21); //HERE YOU CAN MODIFY THE GAIN AND OFFSET OF CHANNEL 2
    CHgain = 0.005;
    CHoffset = 0.0;
  break;
  case 3:
    SPI.transfer(READ31); //HERE YOU CAN MODIFY THE GAIN AND OFFSET OF CHANNEL 3
    CHgain = 0.005;
    CHoffset = 0.0;
  break;
  case 4:
    SPI.transfer(READ41); //HERE YOU CAN MODIFY THE GAIN AND OFFSET OF CHANNEL 4
    CHgain = 0.006907320351;
    CHoffset = -390.770399070308;
  break;
  default:
    SPI.transfer(READ11); //DO NOT TOUCH, JUST HERE TO SHOW 0.0 WHEN WRONG CHANNEL REQUESTED
    CHgain = 0.0;
    CHoffset = 0.0;
  break;
}
```

Entonces, dependiendo del parámetro pasado a la función enviará por SPI el comando de lectura del canal deseado e inicializará las variables de ganancia y offset (recta de calibración) a los valores calculados o estimados. Hecho esto, el conversor comenzará la medida y responderá escribiendo un "0" en su pin SDO, o MISO. Además se enviará un byte de datos en blanco para limpiar la "bandera" SDO.

```
while (digitalRead(12)==HIGH); //Esperamos a que SDO sea 0
result_blank = SPI.transfer(0x00); //CLEAN SDO FLAG
```

En este punto el conversor comienza a mandar la lectura del canal indicado, por lo que recibiremos los 3 bytes (24 bits) de respuesta, y una cola de conversión que indicará si el resultado ha sido correcto o si ha aparecido *over-flow*:

```
result = SPI.transfer(0x00); //LEE PRIMEROS 8 BIT
for(int i = 0;i<2;i++) { //AÑADE los 8 y 8 siguientes.
result = result << 8;
inByte = SPI.transfer(0x00);
result = result | inByte; //dado que result comienza siendo 0, el OR suma la lectura
}
cola_conversion = SPI.transfer(0x00);
digitalWrite(CSN_AD,HIGH); //Desactiva comm. AD
```

Como se puede ver, al final se vuelve a poner *HIGH* el pin CS, terminando la comunicación con el conversor. Teniendo los datos en variables, estamos en disposición de realizar el cálculo final, con los parámetros anteriores y devolverlo:

```
  FINALRESULT=(result*CHgain)+CHoffset;
  byte over_range = 0b00000100;
      #ifdef DEBUG
     if((cola_conversion&over_range)==over_range){
      Serial.println("Over-range flag activated");
     }
      #endif
return FINALRESULT;//AQUI EL RESULTADO EN FLOAT.
}
```

readConf ( )

Simplemente como método de comprobación, se ha definido una función que lea el registro de configuración:

```
void readConf(){
  digitalWrite(CSN_AD,LOW); //Activa comm. AD
  byte CONF_READED1 = 0;
  byte CONF_READED2 = 0;
  byte CONF_READED3 = 0;
  byte CONF_READED4 = 0;

  SPI.transfer(COMMAND_READCONF);

  CONF_READED1 = SPI.transfer(0x00);
  CONF_READED2 = SPI.transfer(0x00);
  CONF_READED3 = SPI.transfer(0x00);
  CONF_READED4 = SPI.transfer(0x00);

  digitalWrite(CSN_AD,HIGH);//Desactiva comm. AD
  #ifdef DEBUG
  Serial.print(" Registro de CFG: ");
  Serial.print(CONF_READED1,BIN);
  Serial.print(":");
  Serial.print(CONF_READED2,BIN);
  Serial.print(":");
  Serial.print(CONF_READED3,BIN);
  Serial.print(":");
  Serial.println(CONF_READED4,BIN);
  #endif
}
```

Al igual que en el caso anterior, es necesario activar la comunicación antes de comenzar. Hecho esto, se crean las variables donde se leerá la configuración y se envía el comando. El conversor A/D responde con la misma y se muestra por pantalla.

resetAD ( )

Una de las más complejas, requiere seguir una secuencia, establecida por el fabricante, para resetear el dispositivo. Como en todos los casos, es indispensable comenzar activando la comunicación SPI con el dispositivo. Hecho esto, se definen las variables utilizadas para la lectura y dos variables de comprobación. La secuencia a seguir pasa a ser la siguiente:

1. Se envía al menos 15 veces el comando 0xFF (SYNC1) iniciando el *reset* del puerto serie. Posteriormente 0xFE, finalizando dicho *reset*.

```
for(int i = 0;i<15;i++){
 SPI.transfer(0xFF);//SYNC1 15 veces, inicia reset del puerto serie
  }
 SPI.transfer(0xFE);//SYNC0 finaliza reset
```

2. Terminado el *reset* de SPI, hay que reinicializar el chip CS5534, para lo cual hay que escribir en el bit RS, del registro de configuración, el valor "1" y esperar 20 microsegundos (Ver *datasheet con registros)*.

```
SPI.transfer(COMMAND_CONFGREG);
SPI.transfer(CONF_RESET1);
SPI.transfer(CONF_RESET2);
SPI.transfer(CONF_RESET3);
SPI.transfer(CONF_RESET4);
delayMicroseconds(20);
```

3. Para comprobar si se ha realizado correctamente el paso anterior, es necesario leer el registro de configuración y comprobar que el bit RV se encuentre a "1".

```
SPI.transfer(COMMAND_READCONF);
CONF_READED1 = SPI.transfer(0x00);
CONF_READED2 = SPI.transfer(0x00);
CONF_READED3 = SPI.transfer(0x00);
CONF_READED4 = SPI.transfer(0x00);
#ifdef DEBUG
  Serial.println(CONF_READED1,BIN);
if((CONF_READED1&RV_ON)==RV_ON){
  Serial.println("OK. RV activado.");
}else {
  Serial.println("ERROR. RV no activado.");
}
#endif
```

4. Si todo es correcto, hay que escribir un "0" en el registro de configuración, más concretamente en el bit RS, pero se realiza para todo el registro.

```
delayMicroseconds(20);
SPI.transfer(COMMAND_CONFGREG);  // COMMAND_CONFGREG = (x03)
SPI.transfer(CONFIGREG11); // x00
SPI.transfer(CONFIGREG12); // x00
SPI.transfer(CONFIGREG13); // x00
SPI.transfer(CONFIGREG14); // x00
```

5. Una vez más hay que comprobar si se ha realizado correctamente el paso anterior, para ello RV ha de estar a cero:

```
  SPI.transfer(COMMAND_READCONF); // COMMAND_READCONF

  CONF_READED1 = SPI.transfer(0x00);
  CONF_READED2 = SPI.transfer(0x00);
  CONF_READED3 = SPI.transfer(0x00);
  CONF_READED4 = SPI.transfer(0x00);
  #ifdef DEBUG
  if((CONF_READED1&RV_ON)==RV_OFF){
    Serial.println("OK. RV desactivado.");
  }else {
    Serial.println("ERROR. RV activado.");
  }
  #endif
  digitalWrite(CSN_AD,HIGH); //Desactiva comm. AD
}
```

Si todo es correcto, el chip estará reseteado y listo para funcionar.

confAD ( )

Función simple que activará un filtro de rechazo a 50Hz y establecerá el pin del conversor, A0 como *GUARD (*Ver *Datasheet).* Asimismo el conversor A/D quedará en modo normal, es decir sin bajo consumo.

```
void confAD(){
  digitalWrite(CSN_AD,LOW); //Activa comm. AD
     //CONFIGURATION REGISTER 1x32
  //Ponemos el Fsr a 1 para activar el filtro a 50Hz y el A0 como GUARD. Func. Normal
  delay(20);
  SPI.transfer(COMMAND_CONFGREG);
  SPI.transfer(CONFIGREG11);
  SPI.transfer(CONFIGREG12);
  SPI.transfer(CONFIGREG13);
  SPI.transfer(CONFIGREG14);
  digitalWrite(CSN_AD,HIGH); //Desactiva comm. AD
}
```

confSetups ( )

Con el objetivo de minimizar el código en el programa, la configuración de los modos de lectura en los diversos *setups* se realizará con la siguiente función:

```
void  confSetups(){
  digitalWrite(CSN_AD,LOW); //Activa comm. AD
     //CHANNEL-SETUP REGISTERS
  // Configuramos el CSR #1 (Setup 1 y 2)
  delay(20);
  SPI.transfer(COMMAND_CHANNELREG1);
  SPI.transfer(CHANNELREG11);//SETUP1
  SPI.transfer(CHANNELREG12);//SETUP1
  SPI.transfer(CHANNELREG13);//SETUP2
  SPI.transfer(CHANNELREG14);//SETUP2
  // Configuramos el CSR #2 (Setup 3 y 4)
  delay(20);
  SPI.transfer(COMMAND_CHANNELREG2);
  SPI.transfer(CHANNELREG21);//SETUP3
  SPI.transfer(CHANNELREG22);//SETUP3
  SPI.transfer(CHANNELREG23);//SETUP4
  SPI.transfer(CHANNELREG24);//SETUP4
  digitalWrite(CSN_AD,HIGH); //Desactiva comm. AD
  }
```

Simplemente enviará la configuración deseada a los registros correspondientes, es decir, aquellos que incluyen los 4 primeros *setups*.

sleepAD ( ) y wakeupAD ( )

Las funciones que gestionarán el modo normal y "sueño" del conversor.

```
void sleepAD(){
  digitalWrite(CSN_AD,LOW); //Activa comm. AD
    //CONFIGURATION REGISTER 1x32
  SPI.transfer(COMMAND_CONFGREG);
  SPI.transfer(CONFIGREG21);
  SPI.transfer(CONFIGREG22);
  SPI.transfer(CONFIGREG23);
  SPI.transfer(CONFIGREG24);
  digitalWrite(CSN_AD,HIGH); //Desactiva comm. AD
}
```

El caso de la función sueño enviará la configuración al registro correspondiente, evitando que el controlador pueda realizar lecturas y disminuyendo considerablemente su consumo.

La función para despertar devolverá al controlador a su estado "normal", es decir, listo para realizar lecturas y conversiones:

```
void wakeupAD(){
  digitalWrite(CSN_AD,LOW); //Activa comm. AD
  delay(20);
     //CONFIGURATION REGISTER 1x32
  SPI.transfer(COMMAND_CONFGREG);
  SPI.transfer(CONFIGREG11);
  SPI.transfer(CONFIGREG12);
  SPI.transfer(CONFIGREG13);
  SPI.transfer(CONFIGREG14);
  delay(20);
  digitalWrite(CSN_AD,HIGH); //Desactiva comm. AD
}
```

Haciendo uso de todas estas funciones será posible controlar el conversor A/D y por tanto realizar lecturas de las células de carga. En caso de necesitar la creación de más funciones, los registros y opciones de configuración aparecen en la hoja de datos del componente.