

UNIVERSIDAD DE MÁLAGA

**ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN**



TRABAJO FIN DE MASTER

*METEO: ESTACIÓN DE MONITORIZACIÓN
METEOROLÓGICA REMOTA*

**MASTER EN SISTEMAS ELECTRÓNICOS
PARA ENTORNOS INTELIGENTES**

MÁLAGA, 2016

JOSÉ MIGUEL RÍOS RUBIO

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

METEO: Estación de monitorización meteorológica remota

REALIZADO POR:

José Miguel Ríos Rubio

DIRIGIDO POR:

Alfonso Ariza Quintana

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

TITULACIÓN: Máster en Sistemas Electrónicos para Entornos Inteligentes

PALABRAS CLAVE: Estación Meteorológica Remota (METEO), Meteorología, Sensores, Internet Of Things (IoT), Intel Galileo/Edison, Universal Windows Platform (UWP).

RESUMEN:

Dado el creciente aumento en el número de objetos conectados a Internet, propiciado por el auge del Internet de las cosas, es de una gran importancia ser capaz de recopilar los datos que fluyen por la red para mostrarlos de una manera eficiente. En el presente documento, nos enfocamos en el desarrollo de un sistema de monitorización remota de parámetros atmosféricos; una estación meteorológica que se comunicará por red.

Los datos atmosféricos correspondientes a los parámetros ambientales serán obtenidos a partir de diversos sensores, las medidas de estos sensores se realizarán a través de un sistema empotrado que tratará dichos parámetros y los enviará por red. Un cliente software localizado en un ordenador personal dentro de la misma red se encargará de recopilar y mostrar dichos parámetros al usuario.

Málaga, diciembre de 2016

Tabla de contenidos

SECCIÓN I: Especificación de requisitos y casos de uso

SECCIÓN II: Diseño e implementación

SECCIÓN III: Pruebas

SECCIÓN IV: Conclusiones y Posibles Mejoras

ANEXOS

Problemas en el Proceso de Implementación

CD-ROM

Especificación de requisitos y casos de uso

En este documento vamos a especificar los requisitos y casos de uso del proyecto *METEO* (Estación meteorológica). Esta especificación incluye también los objetivos, alcance, directivas del proyecto, descripción de los interesados y usuarios finales del producto/sistema generado.

TÍTULO DOCUMENTO	Documento de requisitos y casos de uso
FECHA DE ENTREGA	12/12/2016
AUTOR/ES	José Miguel Ríos Rubio

COMENTARIOS	VERSIÓN	FECHA
Versión Inicial	1.0	05/12/2015
Correcciones	1.1	11/10/2016
Adaptación a los cambios de última hora	1.1.2	06/11/2016

ÍNDICE

- 1 Introducción
 - 1.1 Objetivos
 - 1.2 Definiciones, acrónimos y abreviaturas
- 2 Directivas del Proyecto
 - 2.1 Oportunidad de negocio
 - 2.2 Descripción del problema
 - 2.3 Descripción del producto
- 3 Descripción de participantes y usuarios
 - 3.1 Resumen de los participantes
 - 3.2 Resumen y entorno de los usuarios
 - 3.2.1 Entorno de los usuarios
 - 3.3 Perfiles de los participantes
 - 3.3.1 Jefe de la empresa (Director de TFM)
 - 3.3.2 Desarrollador (Alumno de TFM)
 - 3.4 Perfiles de usuario
 - 3.4.1 Operarios METEO (Jurado evaluador de TFM)
 - 3.5 Alternativas y competencia
 - 3.5.1 <http://www.vantagevue.com>
 - 3.5.2 <http://store.oregonscientific.com>
- 4 Requisitos
 - 4.1 Diagrama general
 - 4.2 Precedencia y prioridad
 - 4.3 Requisitos Funcionales
 - 4.3.1 R-1.0 Medición de parámetros ambientales
 - 4.3.2 R-1.1 Temperatura
 - 4.3.3 R-1.2 Humedad
 - 4.3.4 R-1.3 Presión
 - 4.3.5 R-2.0 Control Ordenador
 - 4.3.6 R-3.0 Mostrar Temperatura Ordenador
 - 4.3.7 R-3.1 Mostrar Humedad Ordenador
 - 4.3.8 R-3.2 Mostrar Presión Ordenador
 - 4.3.9 R-4.0 Control Campo

- 4.3.10 R-5.0 Mostrar Temperatura Campo
- 4.3.11 R-5.1 Mostrar Humedad Campo
- 4.3.12 R-5.2 Mostrar Presión Campo
- 4.3.13 R-7.0 Almacenamiento de Datos
- 4.3.14 R-7.3 Exportar datos
- 4.3.15 R-8.0 Datos en Tiempo Real
- 4.3.16 R-8.1 Datos Promediados
- 4.3.17 R-9.0 Control Periodo Promediado
- 4.3.18 R-10.0 Sensado Independiente
- 4.4 Requisitos No Funcionales
 - 4.4.1 R-2.1 Herramienta para el desarrollo de la aplicación de ordenador
 - 4.4.2 R-2.2 Comunicación UDP-TCP/IP con ordenador
 - 4.4.3 R-2.3 Interfaz de comunicación USB con ordenador
 - 4.4.4 R-4.1 Interfaz de Control Campo - Visionado
 - 4.4.5 R-4.2 Interfaz de Control Campo - Control
 - 4.4.6 R-6.0 Intel Edison
 - 4.4.7 R-6.1 Bajo Precio
 - 4.4.8 R-6.2 Alimentación de Red
 - 4.4.9 R-7.1 Capacidad de almacenamiento de Datos
 - 4.4.10 R-7.2 Soporte de Almacenamiento
 - 4.4.11 R-11.0 Fecha de entrega
- 5 Casos de uso
 - 5.1 Actores del sistema
 - 5.2 Diagrama general
 - 5.3 Descripción textual de los casos de uso
 - 5.3.1 C1 Leer parámetros ambientales
 - 5.3.2 C2 Mostrar valores de parámetros en campo
 - 5.3.3 C3 Mostrar valores de parámetros en ordenador
 - 5.3.4 C4 Guardar datos en memoria interna
 - 5.3.5 C5 Exportar datos de la memoria interna
 - 5.3.6 C6 Control del periodo de promediado
- 6 Arquitectura
 - 6.1 Arquitectura Lógica
 - 6.1.1 Vista general
 - 6.1.2 METEO

- 6.1.3 Interfaz de campo
- 6.1.4 Interfaz de ordenador
- 6.1.5 Archivo .txt
- 6.2 Arquitectura Física
 - 6.2.1 Vista general
 - 6.2.2 METEO
 - 6.2.3 Sensor
 - 6.2.4 Temperatura
 - 6.2.5 Humedad
 - 6.2.6 Presión
 - 6.2.7 LCD
 - 6.2.8 Pulsadores
 - 6.2.9 Red TCP/IP
 - 6.2.10 Ordenador
- 7 Planificación temporal y asignación de recursos
 - 7.1 Planificación temporal
 - 7.1.1 Descripción de hitos
 - 7.1.2 Descripción de Tareas
 - 7.2 Asignación de recursos
 - 7.2.1 Asignación del personal por tarea
 - 7.2.2 Vista desglosada de horas del personal por semana
 - 7.3 Estimación de costes
 - 7.3.1 Coste por recurso
 - 7.3.2 Coste por tareas
 - 7.3.3 Distribución del coste entre los recursos

METEO: Estación de monitorización meteorológica



1 Introducción

El objetivo de este documento es capturar, analizar y definir las características y necesidades de alto nivel del sistema *METEO*. Se centrará en describir las expectativas de cada una de las partes del proyecto y de los usuarios finales y por qué estas necesidades existen. Los detalles de cómo el sistema *METEO* cumple estas necesidades se detallan en los casos de uso y en las especificaciones adicionales.

1.1 Objetivos

Se pretende realizar un sistema que permita determinar los parámetros ambientales de temperatura, humedad y presión de un entorno controlado (en el que se conocen los rangos máximos y mínimos que pueden tomar estos valores) con el fin de informar y alertar a los posibles operarios de una empresa, de que estos parámetros se encuentran dentro de los valores esperados, en el entorno en el que se sitúe el sistema.

1.2 Definiciones, acrónimos y abreviaturas

TFM – Trabajo Fin de Master.

METEO – Estación de Monitorización Meteorológica.

2 Directivas del Proyecto

2.1 Oportunidad de negocio

Este sistema permitirá a la empresa mejorar la eficiencia de trabajo, erradicando la necesidad anterior a la implantación de este sistema, de lectura y monitorización de los parámetros de temperatura, humedad y presión por parte de los operarios de forma manual, evitando la necesidad de desplazamiento hacia cada mota sensora para obtener los datos medidos por esta. Así, como mejorar la seguridad laboral de los mismos.

2.2 Descripción del problema

El problema de	Lectura continua de los parámetros ambientales (temperatura, humedad y presión) de un cierto entorno.
Afecta a	Operarios (usuarios) y Jefe de la empresa.
Lo cual tiene como impacto	No tener información sobre los valores de estos parámetros y tener que desplazarse para realizar la lectura de forma manual de los mismos.
Una solución satisfactoria sería	Posicionar un sistema que permita leer, a partir de sus sensores, dichos parámetros e informar de sus valores al operario.

2.3 Descripción del producto

Para	Operarios (usuarios) y Jefe de la empresa.
Los cuales	Necesitan un sistema para conocer los valores de temperatura, humedad y presión de forma automática.
Estación de monitorización meteorológica (METEO)	Es un sistema software y hardware.
Que	Determina, registra y muestra los parámetros de temperatura, humedad y presión de un cierto entorno.
Frente a	Otros productos de competencia (por ejemplo, estaciones de monitorización remota de Vantage o Oregon Scientific).
Nuestro producto	Tiene un coste muy inferior al de la competencia y es un producto a medida que satisface completamente las necesidades de nuestro cliente.

3 Descripción de participantes y usuarios

3.1 Resumen de los participantes

Nombre	Representa	Rol
Jefe de la empresa	Jefe de la empresa ficticia que demanda el proyecto (correspondería con el Director de TFM).	Stakeholder principal, será el que establezca la mayoría de los requisitos, aprobará los presupuestos y validará el resultado del proyecto.
Desarrollador	Empleado de la empresa ficticia que desarrolla el proyecto (correspondería con el alumno de TFM).	Encargado de especificar, diseñar y gestionar los recursos utilizados para desarrollar el sistema, así como el desarrollo del mismo.
Operarios METEO	Usuario final perteneciente a la empresa ficticia que demanda el proyecto (correspondería con el tribunal del TFM).	Se encargará de utilizar y evaluar el sistema desarrollado, determinando el correcto funcionamiento del sistema dentro de la empresa donde se va a utilizar.

3.2 Resumen y entorno de los usuarios

Nombre	Descripción	Participante
Jefe de la empresa	Encargado de revisar los gráficos de resultados mensuales de sensado de temperatura, humedad y presión (promediado temporal).	Jefe de la empresa
Operarios METEO	Encargados de controlar y gestionar el servidor, monitorizar los datos de temperatura, humedad y presión sensados por el sistema y actuar en consecuencia.	Operarios METEO

3.2.1 Entorno de los usuarios

Los usuarios del sistema son expertos en el área. La utilización del sistema implicará el uso de las interfaces del sistema (el cliente software, un programa específico localizado en un ordenador personal, y/o la interfaz de “campo”, la cual corresponde a una pantalla LCD/TFT y botones de control localizados en el mismo hardware de sensado) con el fin de monitorizar las variables de temperatura, presión y humedad sensadas para actuar en consecuencia. El sistema METEO estará conectado en red con diversos ordenadores de la empresa, a partir de los cuales se puede acceder a la mota sensora mediante el cliente software específico.

3.3 Perfiles de los participantes

3.3.1 Jefe de la empresa (Director de TFM)

Representante	Alfonso Ariza Quintana.
Tipo	Doctor Ingeniero de Telecomunicación y Profesor Titular de Universidad en el área de Tecnología Electrónica.
Responsabilidades	Definir los requisitos y especificaciones del sistema. Aprobar el presupuesto del sistema. Validar el sistema.
Criterio de Éxito	Obtener un sistema funcional que cumpla con los requisitos básicos.
Entregables	Documentación sobre el sistema a desarrollar y las especificaciones del sistema y el prototipo hardware y software del proyecto.
Comentarios	Validará el sistema final y aprobará la finalización del mismo.

3.3.2 Desarrollador (Alumno de TFM)

Representante	José Miguel Ríos Rubio.
Tipo	Alumno del Master de Sistemas Electrónicos para Entornos Inteligentes (MSEEI).
Responsabilidades	Recopilar y definir requisitos y especificaciones del sistema, así como desarrollar e implementar el sistema (documentación y prototipo) y defenderlo en la etapa de evaluación.
Criterio de Éxito	Obtener un sistema funcional verificado, que cumpla con los requisitos básicos y avanzados.
Entregables	Documentación del proceso de desarrollo del sistema, manual de uso y características... Memoria técnica del sistema. Prototipo final.
Comentarios	No.

3.4 Perfiles de usuario

3.4.1 Operarios METEO (Jurado evaluador de TFM)

Representantes	A definir.
Descripción	Múltiples operarios de la empresa.
Tipo	Usuarios expertos.
Responsabilidades	Monitorizar el estado de la temperatura, la humedad y la presión del entorno/sistema a controlar para poder actuar en consecuencia en caso de valores fuera de los límites recomendados.
Criterio de Éxito	Poder visualizar a tiempo real los datos sensados de forma simple, eficiente, de forma numérica y gráfica.
Comentarios	No

3.5 Alternativas y competencia

3.5.1 <http://www.vantagevue.com>

La empresa *Vantage* proporciona diversos productos estaciones meteorológicas. Estos sistemas, sin embargo, a pesar de tener un precio semejante al de nuestro producto, o exceden las necesidades de nuestro cliente (y aumenta el precio) o le faltan prestaciones, es decir, no son productos a medida. Por tanto, aunque es una alternativa no lo consideramos como competencia.

3.5.2 <http://store.oregonscientific.com>

Oregon Scientific presenta estaciones meteorológicas profesionales, pero la mayoría de ellas no tienen capacidad para conexión en Red (presentan conexión usb o a lo sumo Bluetooth), requieren de un ordenador personal de por medio que le de dicha funcionalidad y tampoco facilitan un software que se encargue de esto.

4 Requisitos

4.1 Diagrama general

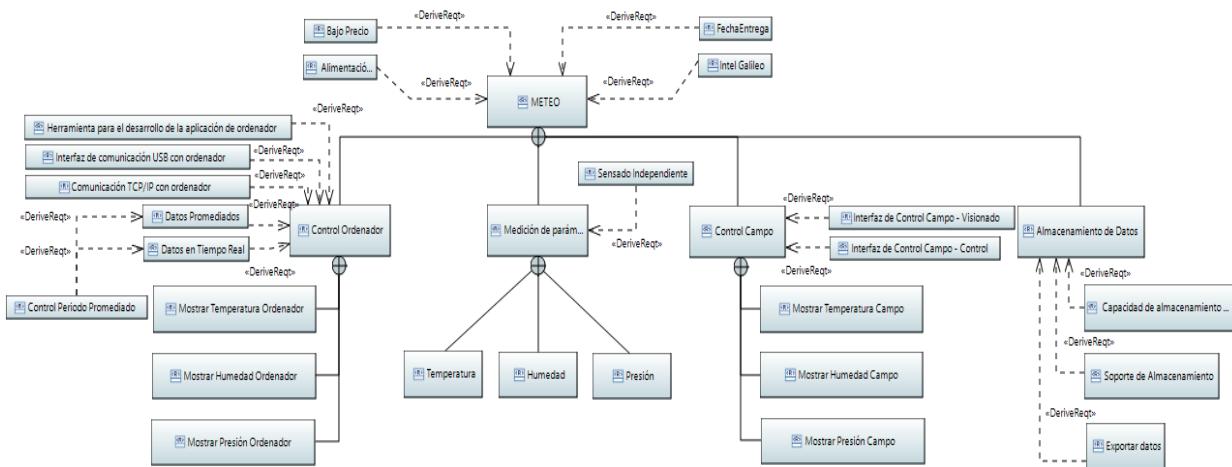


Fig. 1. Diagrama de requisitos generales

4.2 Precedencia y prioridad

Id	Nombre	Prioridad	Precedencia
1.0	Medición de parámetros ambientales	Alta	
1.1	Temperatura	Alta	1.0
1.2	Humedad	Alta	1.0
1.3	Presión	Alta	1.0
2.0	Control Ordenador	Baja	1.x, 2.1, 2.2 y 2,3
2.1	Herramienta para el desarrollo de la aplicación de ordenador	Baja	2.0
2.2	Comunicación UDP-TCP/IP con ordenador	Baja	2.0
2.3	Interfaz de comunicación USB con ordenador	Baja	2.0
3.0	Mostrar Temperatura Ordenador	Alta	1.1, 2.0
3.1	Mostrar Humedad Ordenador	Alta	1.2, 2.0
3.2	Mostrar Presión Ordenador	Alta	1.3, 2.0
4.0	Control Campo	Baja	1.x

4.1	Interfaz de Control Campo - Visionado	Baja	1.x, 4.0
4.1	Interfaz de Control Campo - Control	Baja	1.x, 4.0 y 4.1
5.0	Mostrar Temperatura Campo	Media	1.1, 4.0
5.1	Mostrar Humedad Campo	Media	1.2, 4.0
5.2	Mostrar Presión Campo	Media	1.3, 4.0
6.0	Intel Galileo	Alta	
6.1	Bajo Precio	Alta	
6.2	Alimentación de Red	Alta	
7.0	Almacenamiento de Datos	Media	
7.1	Capacidad de almacenamiento de Datos	Media	7.0
7.2	Soporte de Almacenamiento	Media	7.0, 7.1
7.3	Exportar datos	Baja	7.2
8.0	Datos en Tiempo Real	Alta	
8.1	Datos Promediados	Media	2.0, 7.0
9.0	Control Periodo Promediado	Baja	2.0, 7.0 y 8.1
10.0	Sensado Independiente	Alta	
11.0	Fecha de entrega	Alta	

4.3 Requisitos Funcionales

4.3.1 R-1.0 Medición de parámetros ambientales

El sistema medirá parámetros ambientales.

4.3.2 R-1.1 Temperatura

El sistema medirá la temperatura ambiente.

4.3.3 R-1.2 Humedad

El sistema medirá la humedad ambiente.

4.3.4 R-1.3 Presión

El sistema medirá la presión ambiente.

4.3.5 R-2.0 Control Ordenador

El sistema presentará la información en una aplicación localizada en un ordenador.

4.3.6 R-3.0 Mostrar Temperatura Ordenador

El sistema mostrará la temperatura en la aplicación del ordenador.

4.3.7 R-3.1 Mostrar Humedad Ordenador

El sistema mostrará la humedad en la aplicación del ordenador.

4.3.8 R-3.2 Mostrar Presión Ordenador

El sistema mostrará la presión en la aplicación del ordenador.

4.3.9 R-4.0 Control Campo

El sistema presentará los valores sensados en su propio elemento hardware (sin necesidad de ordenador).

4.3.10 R-5.0 Mostrar Temperatura Campo

El sistema mostrará la temperatura en la pantalla LCD del hardware.

4.3.11 R-5.1 Mostrar Humedad Campo

El sistema mostrará la humedad en la pantalla LCD del hardware.

4.3.12 R-5.2 Mostrar Presión Campo

El sistema mostrará la presión en la pantalla LCD del hardware.

4.3.13 R-7.0 Almacenamiento de Datos

El sistema almacenará los datos sensados.

4.3.14 R-7.3 Exportar datos

El sistema permitirá exportar los datos a un fichero de texto plano en el ordenador desde el que se utilice la aplicación.

4.3.15 R-8.0 Datos en Tiempo Real

La información sensada se mostrará como datos en tiempo real en la aplicación del ordenador.

4.3.16 R-8.1 Datos Promediados

La información sensada se mostrará como un gráfico histórico promedio de datos en la aplicación del ordenador.

4.3.17 R-9.0 Control Periodo Promediado

El usuario podrá definir el rango de promediado a mostrarse gráficamente (segundos, minutos y horas).

4.3.18 R-10.0 Sensado Independiente

Si uno de los sensores del sistema deja de funcionar, el resto de sensores deben de seguir haciéndolo.

4.4 Requisitos No Funcionales

4.4.1 R-2.1 Herramienta para el desarrollo de la aplicación de ordenador

La aplicación localizada en el ordenador se desarrollará mediante Windows Universal Platform, con el framework .NET (C#).

4.4.2 R-2.2 Comunicación UDP-TCP/IP con ordenador

La comunicación del sistema con el ordenador se llevará a cabo a través de los protocolos UDP y TCP en la red local, según corresponda.

4.4.3 R-2.3 Interfaz de comunicación USB con ordenador

El sistema presentará una comunicación serie con el ordenador y se realizará mediante una conexión USB.

4.4.4 R-4.1 Interfaz de Control Campo - Visionado

El sistema presentará una pantalla LCD/TFT para mostrar los datos.

4.4.5 R-4.2 Interfaz de Control Campo - Control

El sistema presentará pulsadores para controlar los datos mostrados.

4.4.6 R-6.0 Intel Edison

El sistema se desarrollará mediante la placa Intel Edison.

4.4.7 R-6.1 Bajo Precio

El sistema no debe de superar un coste máximo de 150 euros.

4.4.8 R-6.2 Alimentación de Red

El sistema permanecerá conectado a la red eléctrica para su funcionamiento (no utilizará batería).

4.4.9 R-7.1 Capacidad de almacenamiento de Datos

El sistema almacenará hasta las últimas 24 horas (último día) de los datos sensados.

4.4.10 R-7.2 Soporte de Almacenamiento

El sistema almacenará los datos sensados en la propia memoria interna.

4.4.11 R-11.0 Fecha de entrega

El proyecto debe finalizar antes del 07/10/2016.

5 Casos de uso

5.1 Actores del sistema

Nombre	Descripción
Operario	Grupo de personas que van a hacer uso de la aplicación.
Temperatura	Representa la temperatura ambiente.
Humedad	Representa la humedad ambiente.
Presión	Representa la presión ambiente.
Tiempo	Representa al reloj interno que invoca a una frecuencia dada ciertos casos de uso.
LCD/TFT	Elemento encargado de mostrar los datos de los parámetros ambientales en campo.
Ordenador	Elemento encargado de mostrar los datos de los parámetros ambientales en ordenador.

5.2 Diagrama general

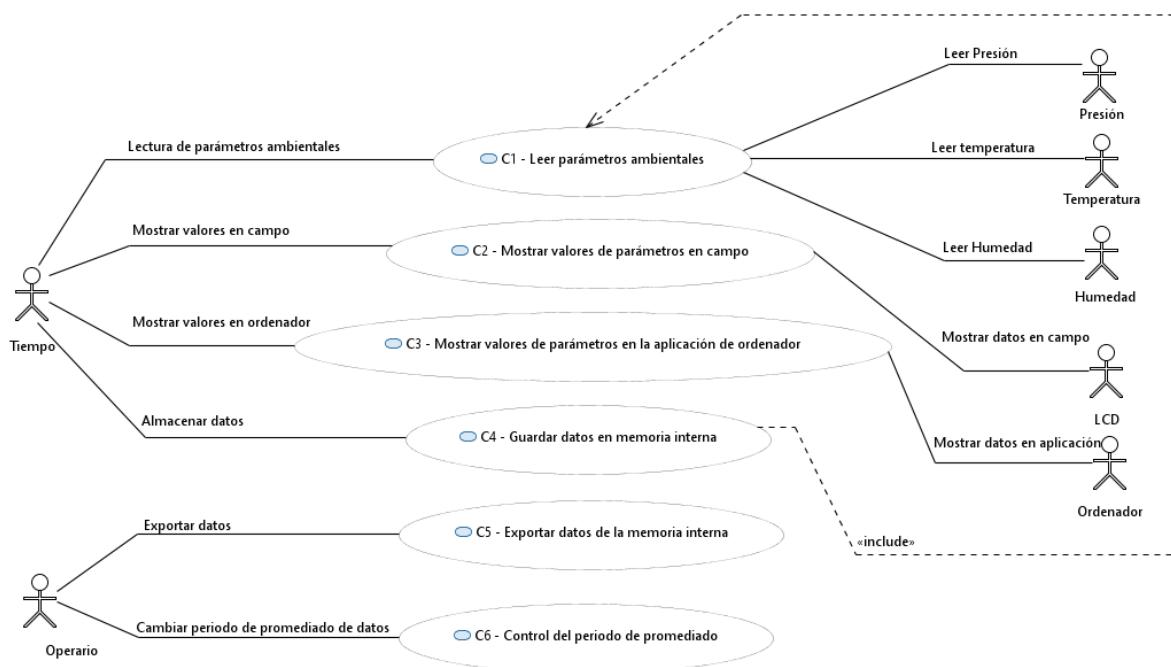


Fig. 2. Diagrama general de casos de uso

5.3 Descripción textual de los casos de uso

5.3.1 C1 Leer parámetros ambientales

5.3.1.1 Contexto de uso:

A una frecuencia dada de 1Hz (cada segundo), se miden los datos de tres sensores situados en el sistema máquina de la cual se le ha solicitado información.

5.3.1.2 Actor Principal:

Tiempo.

5.3.1.3 Participantes y Objetivos:

Participante	Objetivo
Tiempo	Activar la lectura de los parámetros ambientales.
Temperatura	Proveer los datos de su sensor.
Humedad	Proveer los datos de su sensor.
Presión	Proveer los datos de su sensor.

5.3.1.4 Pre Condiciones:

Sistema en funcionamiento.

5.3.1.5 Garantías mínimas:

Se almacenan los valores de cada parámetro (válidos o no).

5.3.1.6 Escenario de éxito principal:

1. El tiempo solicita a METEO la lectura de los parámetros.
2. METEO solicita a Temperatura, Humedad y Presión la información de sus sensores.
3. Temperatura envía su información a METEO.
4. Humedad envía su información a METEO.
5. Presión envía su información a METEO.
6. METEO determina que los valores leídos son correctos.

5.3.1.7 Escenario secundario 1:

3. Temperatura no envía su información a METEO.
- 3a. METEO determina un valor de medición erróneo para temperatura.

5.3.1.8 Escenario secundario 2:

4. Humedad no envía su información a METEO.
- 4a. METEO determina un valor de medición erróneo para humedad.

5.3.1.9 Escenario secundario 2:

5. Presión no envía su información a METEO.
- 5a. METEO determina un valor de medición erróneo para presión.

5.3.2 C2 Mostrar valores de parámetros en campo

5.3.2.1 Contexto de uso:

A una frecuencia dada se muestran los datos de temperatura, humedad y presión almacenados del ambiente en el LCD localizado en el mismo hardware del sistema *METEO*.

5.3.2.2 Actor Principal:

Tiempo.

5.3.2.3 Participantes y Objetivos:

Participante	Objetivo
Tiempo	Activar el proceso de mostrar los parámetros ambientales.

5.3.2.4 Pre Condiciones:

Datos de temperatura, humedad y presión almacenados.

5.3.2.5 Garantías mínimas:

El Sistema METEO muestra los valores de los parámetros a tiempo real.

5.3.2.6 Escenario de éxito principal:

1. El Tiempo solicita al Sistema METEO que muestre los datos de los parámetros ambientales.
2. El Sistema METEO muestra los valores almacenados de los parámetros ambientales en LCD.

5.3.3 C3 Mostrar valores de parámetros en ordenador

5.3.3.1 Contexto de uso:

A una frecuencia dada se muestran los datos de temperatura, humedad y presión almacenados del ambiente en la aplicación localizada en el ordenador.

5.3.3.2 Actor Principal:

Tiempo.

5.3.3.3 Participantes y Objetivos:

Participante	Objetivo
Tiempo	Activar el proceso de mostrar los parámetros ambientales.

5.3.3.4 Pre Condiciones:

Datos de temperatura, humedad y presión almacenados.

5.3.3.5 Garantías mínimas:

El Sistema METEO muestra los valores de los parámetros a tiempo real.

5.3.3.6 Escenario de éxito principal:

1. El Tiempo solicita al Sistema METEO que muestre los datos de los parámetros ambientales.
2. El Sistema METEO muestra los valores almacenados de los parámetros ambientales en ORDENADOR.

5.3.4 C4 Guardar datos en memoria interna

5.3.4.1 Contexto de uso:

A una frecuencia dada se almacenan los datos de temperatura, humedad y presión, leídos previamente, en la memoria interna del sistema.

5.3.4.2 Actor Principal:

Tiempo.

5.3.4.3 Participantes y Objetivos:

Participante	Objetivo
Tiempo	Activar el proceso de almacenado de datos correspondientes a los parámetros ambientales.

5.3.4.4 Pre Condiciones:

Datos de temperatura, humedad y presión sensados.

5.3.4.5 Garantías mínimas:

El Sistema METEO almacenará los valores de cada parámetro ambiental hasta un mínimo de valores correspondientes a tres meses.

5.3.4.6 Escenario de éxito principal:

1. El Sistema METEO realiza la lectura de cada uno de los sensores de TEMPERATURA, HUMEDAD y PRESIÓN (C1).
2. El Sistema METEO almacena en su memoria interna los valores leídos.

5.3.5 C5 Exportar datos de la memoria interna

5.3.5.1 Contexto de uso:

El operario (usuario) interactúa con el sistema a través de la aplicación de ordenador con el fin de exportar los datos de los parámetros ambientales, localizados en la memoria interna del sistema, en un fichero de texto plano.

5.3.5.2 *Actor Principal:*

Operario.

5.3.5.3 *Participantes y Objetivos:*

Participante	Objetivo
Operario	Activar la exportación de los datos de los tres parámetros ambientales.

5.3.5.4 *Pre Condiciones:*

Datos almacenados previamente en la memoria interna del sistema METEO.

5.3.5.5 *Garantías mínimas:*

Datos exportados en fichero de texto plano.

5.3.5.6 *Escenario de éxito principal:*

1. El OPERARIO activa la exportación de datos, especificando la ruta y el nombre del archivo.
2. El Sistema METEO realiza la exportación de los datos.

5.3.6 C6 Control del periodo de promediado

5.3.6.1 *Contexto de uso:*

El operario (usuario) debe establecer el periodo de tiempo en que se quiere graficar los valores promediados de los parámetros ambientales con el fin de visualizar la variación de estos a lo largo del tiempo (ya sea a lo largo del último minuto, La última hora, o las últimas 24 horas).

5.3.6.2 *Actor Principal:*

Operario.

5.3.6.3 *Participantes y Objetivos:*

Participante	Objetivo
Operario	Establecer el periodo de tiempo de promediado que se quiere graficar.

5.3.6.4 *Pre Condiciones:*

Datos almacenados previamente en la memoria interna del sistema METEO.

5.3.6.5 Garantías mínimas:

Mostrar gráfica de valores promediados de las variables ambientales en rangos prefijados (no personalizados), por ejemplo, segundos, minutos u horas.

5.3.6.6 Escenario de éxito principal:

1. El usuario establece un periodo temporal en el que quiere visualizar la variación de valores de los parámetros ambientales.
2. El Sistema METEO accede a su archivo de memoria que almacena el promediado de los valores para adaptarse al rango de tiempo establecido.
3. El Sistema METEO muestra en una gráfica, del periodo temporal preestablecido, los datos promediados de los parámetros ambientales.

6 Arquitectura

6.1 Arquitectura Lógica

6.1.1 Vista general

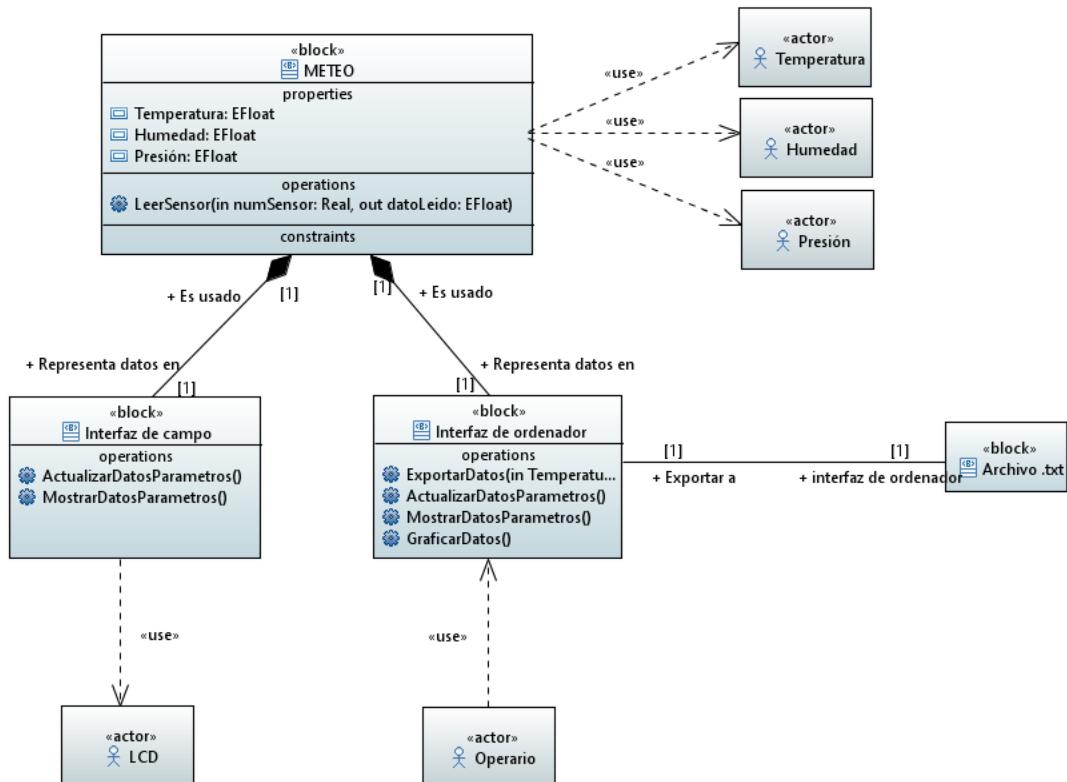


Fig. 3. Arquitectura lógica del sistema

6.1.2 METEO

Modela el conjunto de la mayoría del software del sistema (la “lógica” interna), siendo el núcleo del sistema. El sistema *METEO* adquiere (mediante su operación de lectura) la información de la temperatura, humedad y presión del ambiente y, ésta a su vez, es mostrada en las dos interfaces que componen al sistema, la interfaz de campo LCD/TFT) y/o la interfaz de ordenador.

6.1.3 Interfaz de campo

Modela la parte software gráfica y de control correspondiente con mostrar la información de los datos leídos en el LCD/TFT y de controlar lo mostrado con los pulsadores.

6.1.4 Interfaz de ordenador

Modela la parte software gráfica y de control correspondiente con mostrar la información de los datos leídos en la aplicación con un ordenador y de controlar lo mostrado con el mismo. Sería la aplicación de control propiamente dicha con la que los operarios interaccionarían en el ordenador personal.

6.1.5 Archivo .txt

Modela el archivo de texto plano, generado dentro del ordenador con el que se está utilizando la interfaz, en el que se almacenarán todas las lecturas de los datos dentro del periodo que el usuario establece.

6.2 Arquitectura Física

6.2.1 Vista general

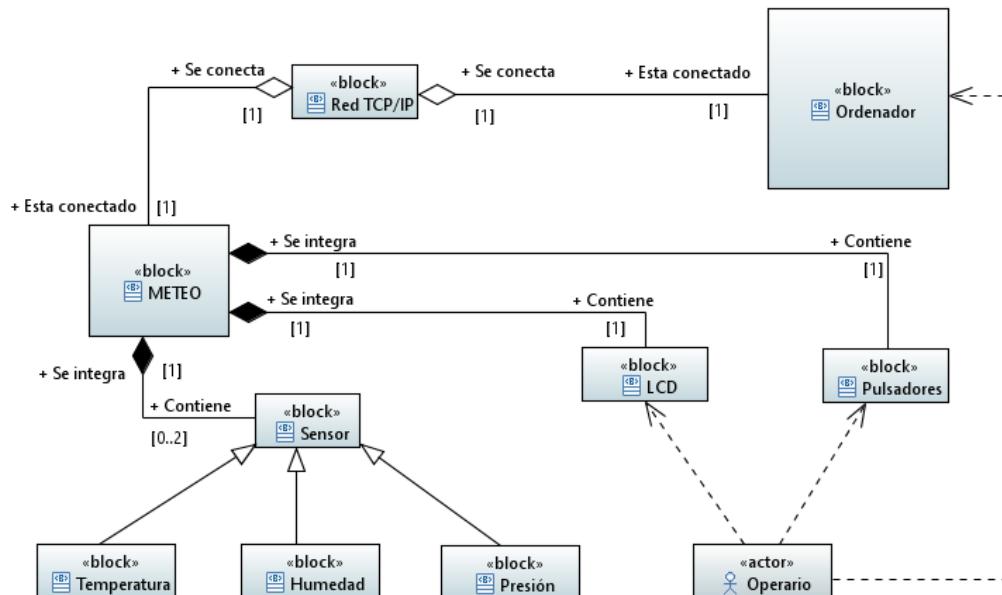


Fig. 4. Arquitectura física del sistema

6.2.2 METEO

Representa el conjunto del hardware conformado por el sistema de desarrollo Intel Galileo.

6.2.3 Sensor

Representa la generalización de los tres sensores correspondientes a cada parámetro ambiental.

6.2.4 Temperatura

Representa al sensor de Temperatura.

6.2.5 Humedad

Representa al sensor de Humedad.

6.2.6 Presión

Representa al sensor de Presión.

6.2.7 LCD

Representa al LCD/TFT utilizado para mostrar los datos en campo.

6.2.8 Pulsadores

Representa al conjunto de pulsadores que permiten interactuar a los usuarios con el sistema en campo.

6.2.9 Red TCP/IP

Representa a la red TCP/IP utilizada para la transmisión de datos hacia la aplicación del ordenador.

6.2.10 Ordenador

Representa al ordenador que contiene la aplicación que sirve de interfaz para el sistema a través de la Red.

7 Planificación temporal y asignación de recursos

7.1 Planificación temporal

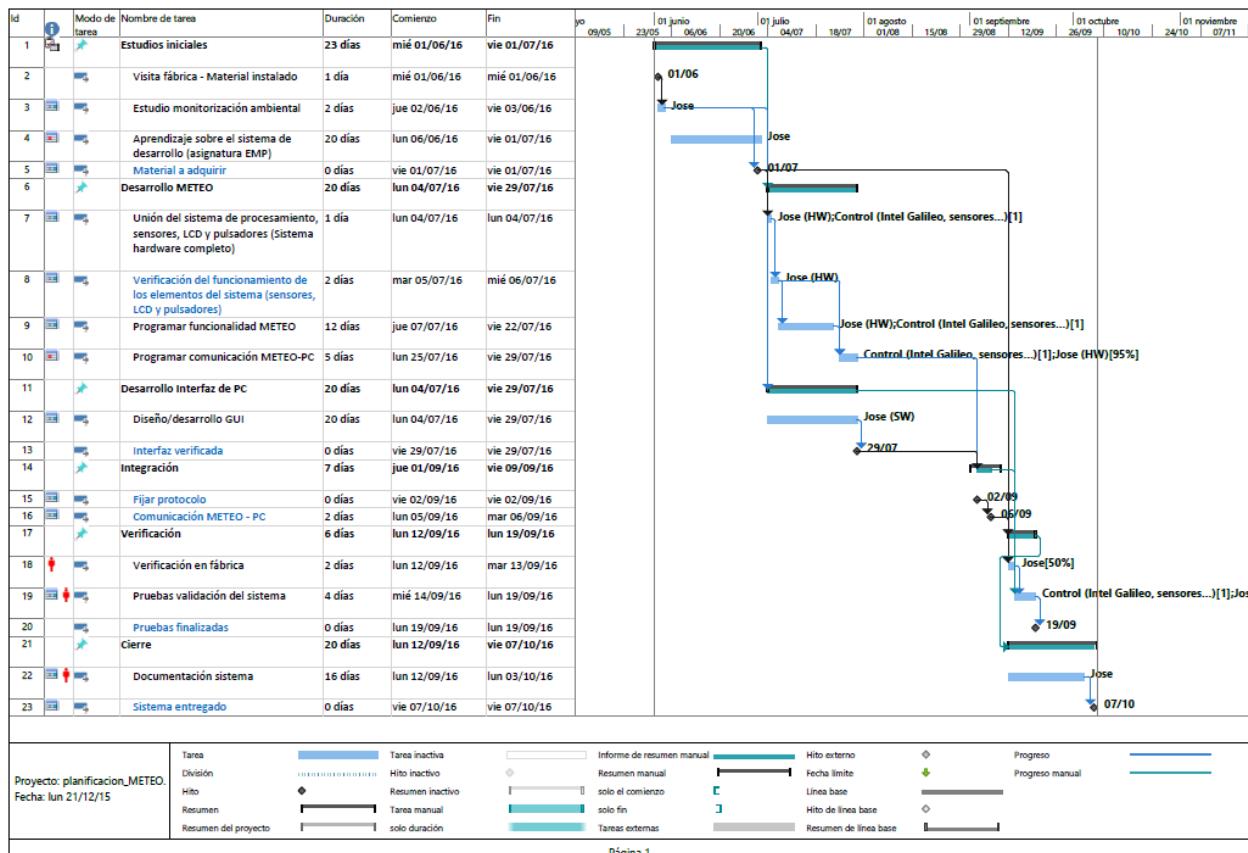


Fig. 5. Planificación temporal

7.1.1 Descripción de hitos

ID	Hitos	Descripción	Comienzo	Fin
2	Visita fábrica – Material instalado.	Visitar la fábrica para determinar claramente el entorno en el que se desplegará nuestro sistema.	01/06/16	01/06/16
5	Material a adquirir.	Reunión para decidir qué material comprar.	01/07/16	01/07/16
8	Verificación del funcionamiento de los elementos del sistema	Sensores, pantalla LCD, pulsadores e Intel Galileo verificados individualmente para corroborar su correcta funcionalidad.	05/07/16	07/07/16

	(sensores, LCD y pulsadores).			
13	Interfaz verificada.	La interfaz de usuario para el PC funciona correctamente y permite las opciones de monitorización y control requeridas por el cliente.	29/07/16	29/07/16
15	Fijar protocolo.	Protocolo de comunicación establecido y puesto en común para todo el equipo de trabajo.	02/09/16	02/09/16
16	Comunicación METEO-PC.	El sistema se comunica correctamente con el PC. Verificación del correcto funcionamiento de cada comando de intercambio de información.	05/09/16	06/09/16
20	Pruebas finalizadas.	Cierre de las pruebas de verificación y validación final del sistema completo.	19/09/16	19/09/16
23	Sistema entregado.	Entrega del sistema al cliente.	07/10/16	07/10/16

7.1.2 Descripción de Tareas

Id	Tarea	Descripción
1	Estudios iniciales	Estudios previos a las fases de desarrollo y prueba
3	Estudio monitorización ambiental.	Análisis del entorno en el que posicionar el sistema en la fábrica: como monitorizar el ambiente o qué posibilidades de control se requiere. Búsqueda de soluciones similares.
4	Aprendizaje sobre el sistema de desarrollo (asignatura EMP).	Aprendizaje sobre el uso de la Intel Galileo. Asignatura del master Microprocesadores Empotrados.
6	Desarrollo METEO	Creación del sistema METEO como tal
7	Unión del sistema de procesamiento, sensores, LCD y pulsadores (Sistema hardware completo).	Integración de todos los elementos para conformar al sistema.
9	Programar funcionalidad METEO.	Programación de la funcionalidad genérica del sistema (excluyendo la parte de comunicación con el PC).
10	Programar comunicación METEO-PC.	Programación de la comunicación del sistema METEO para permitir el envío de información a la aplicación del PC

11	Desarrollo interfaz de PC	Diseño de la interfaz de usuario que deba albergar el PC que gestiona el sistema
12	Diseño/desarrollo GUI.	Diseño y desarrollo de la interfaz de usuario con las funcionalidades requeridas. Pruebas de usuario.
14	Integración	Integración del sistema completo METEO junto al PC de gestión
17	Verificación	Pruebas de verificación y validación globales al sistema
18	Verificación en fábrica.	Pruebas de verificación del sistema implantado en la fábrica.
19	Pruebas validación del sistema.	Pruebas de validación final del sistema completo en la fábrica.
21	Cierre	Procesos de cierre
22	Documentación sistema.	Documentación del sistema (Manual de instalación, Manual de Usuario).

7.2 Asignación de recursos

7.2.1 Asignación del personal por tarea

Nombre de Tarea	Trabajo	Duración	Comienzo	Fin
Estudios iniciales	184 horas	23 días	mié 01/06/16	vie 01/07/16
Estudio monitorización máquinas	16 horas	2 días	jue 02/06/16	vie 03/06/16
Jose	16 horas	2 días	jue 02/06/16	vie 03/06/16
Aprendizaje sobre el sistema de desarrollo (asignatura EMP)	160 horas	20 días	lun 06/06/16	vie 01/07/16
Jose	160 horas	20 días	lun 06/06/16	vie 01/07/16
Desarrollo METEO	158 horas	20 días	jue 04/07/16	jue 29/07/16
Unión del sistema de procesamiento, sensores, LCD y pulsadores (Sistema hardware completo)	8 horas	1 días	lun 04/07/16	lun 04/07/16
Jose (HW)	8 horas	1 días	lun 04/07/16	lun 04/07/16

<i>Control (Intel Galileo, sensores...)</i>			lun 04/07/16	lun 04/07/16
Programar funcionalidad METEO	96 horas	12 días	jue 07/07/16	vie 22/07/16
Jose (HW)	96 horas	12 días	jue 07/07/16	vie 22/07/16
<i>Control (Intel Galileo, sensores...)</i>				
Programar comunicación METEO-PC	38 horas	5 días	lun 25/07/16	vie 29/07/16
Jose (HW)	38 horas	5 días	lun 25/07/16	vie 29/07/16
<i>Control (Intel Galileo, sensores...)</i>				
Desarrollo interfaz de PC	160 horas	20 días	lun 04/07/16	vie 29/07/16
Diseño/desarrollo GUI	160 horas	20 días	lun 04/07/16	vie 29/07/16
Jose (SW)	160 horas	20 días	lun 04/07/16	vie 29/07/16
Verificación	40 horas	6 días	lun 12/09/16	lun 19/09/16
Verificación en fábrica	8 horas	2 días	lun 12/09/16	mar 13/09/16
Jose	8 horas	2 días	lun 12/09/16	mar 13/09/16
Pruebas validación del sistema	32 horas	4 días	mié 14/09/16	lun 19/09/16
Jose	48 horas	6 días	mié 14/09/16	lun 19/09/16
Cierre	128 horas	20 días	lun 12/09/16	vie 07/10/16
Documentación sistema	128 horas	16 días	lun 12/09/16	lun 03/10/16
Jose	128 horas	16 días	lun 12/09/16	lun 03/10/16

7.2.2 Vista desglosada de horas del personal por semana

7.2.2.1 José

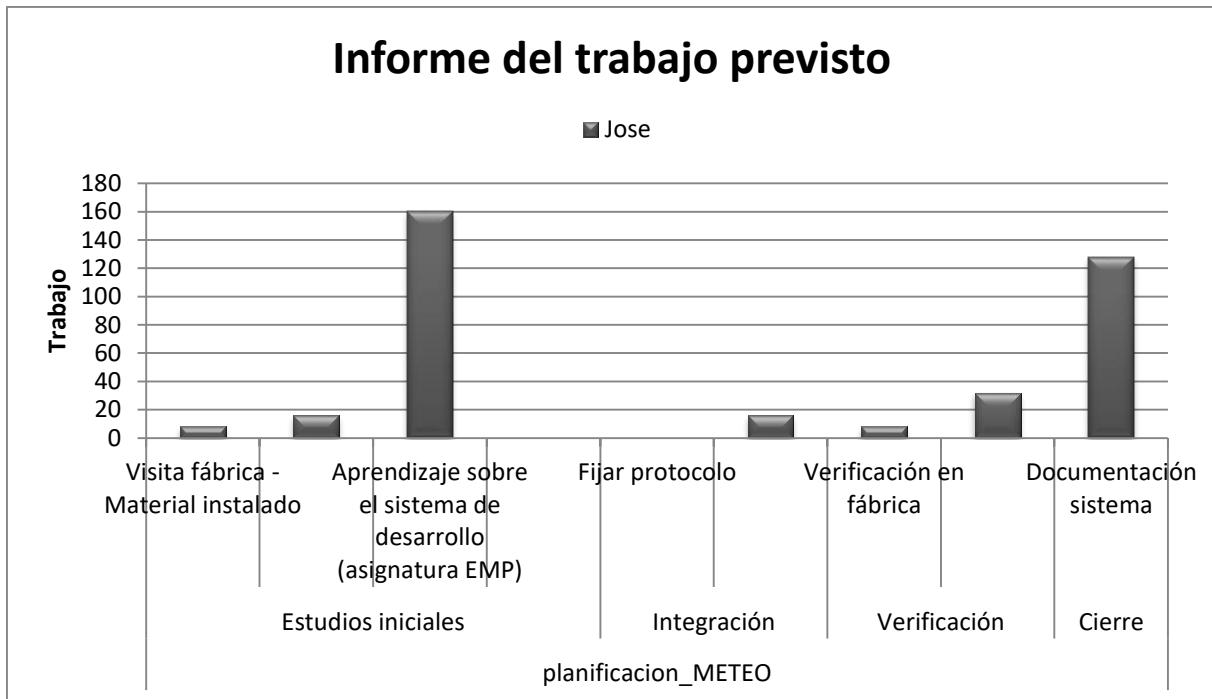


Fig. 6. Trabajo previsto para José

7.2.2.2 José (HW)

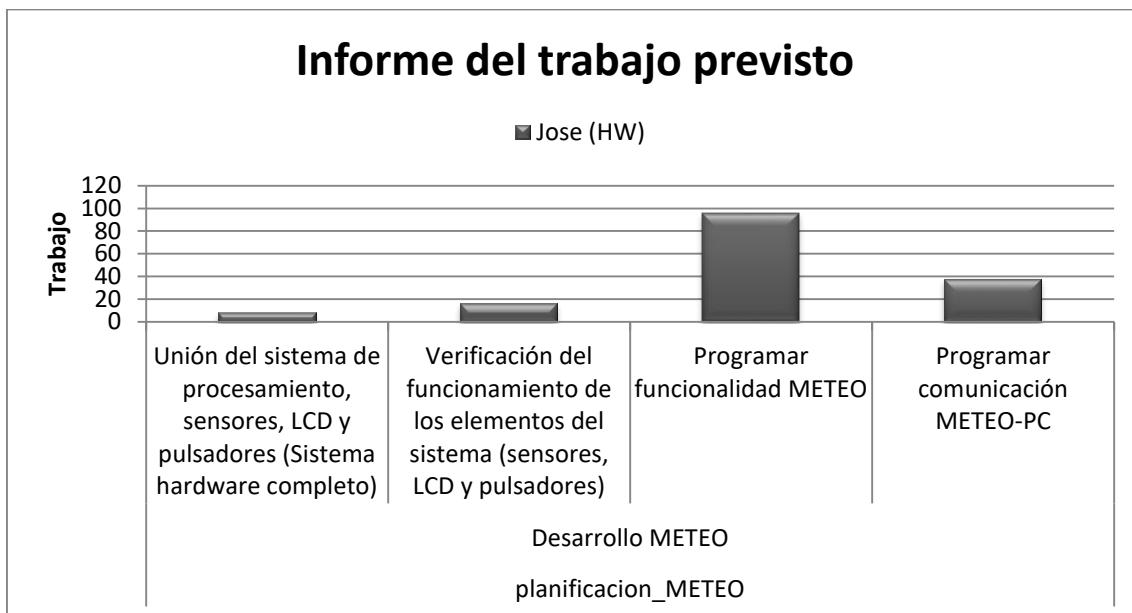
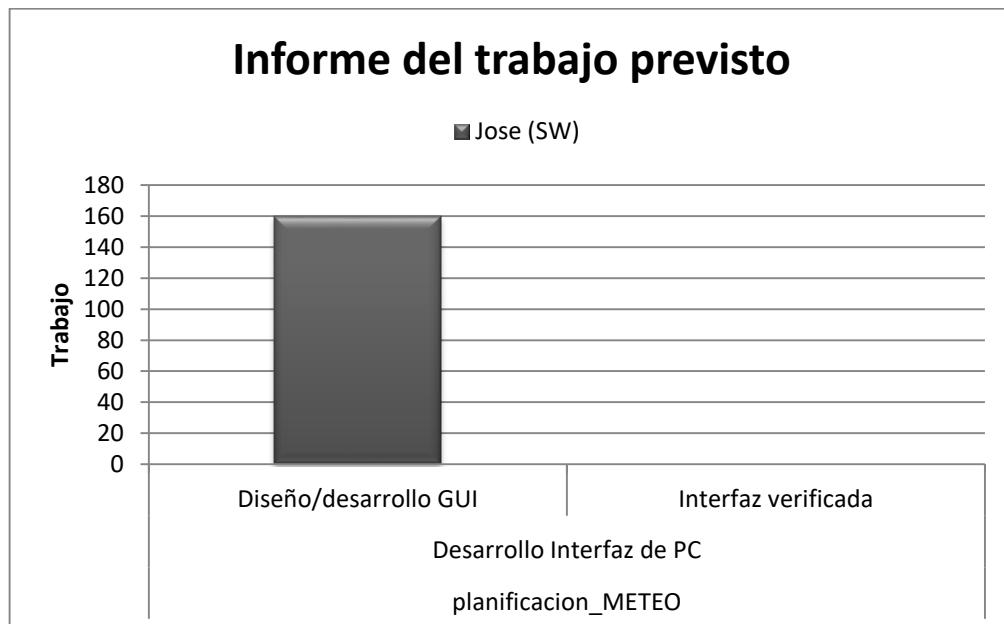


Fig. 7. Trabajo previsto para José en aspectos Hardware

7.2.2.3 José (SW)

**Fig. 8.** Trabajo previsto para José en aspectos Software

7.3 Estimación de costes

7.3.1 Coste por recurso

Nombre del Recurso	Tipo	Iniciales	Tasa estándar	Costo/Uso
Jose	Trabajo	J	18,00 €/hora	0,00 €
Jose (HW)	Trabajo	JHW	9,00 €/hora	0,00 €
Jose (SW)	Trabajo	JSW	9,00 €/hora	0,00 €
Control (Intel Edison, sensores, LCD...)	Material	C	0,00 €	100,00 €

7.3.2 Coste por tareas

Nombre	Costo
Estudios iniciales	3.680,00 €
Estudio monitorización ambiental	320,00 €
Aprendizaje sobre el sistema de desarrollo (asignatura EMP)	3.200,00 €
Desarrollo METEO	2.970,00 €
Unión del sistema de procesamiento, sensores, LCD y pulsadores (Sistema hardware completo)	320,00 €
Programar funcionalidad METEO	1.640,00 €
Programar comunicación METEO-PC	770,00 €
Desarrollo interfaz de PC	2.400,00 €
Diseño/desarrollo GUI	2.400,00 €
Integración	320,00 €
Verificación	1.000,00 €
Verificación en fábrica	160,00 €
Pruebas validación del sistema	840,00 €
Cierre	2.560,00 €
Documentación sistema	2.560,00 €
TOTAL	12.930,00 €

7.3.3 Distribución del coste entre los recursos

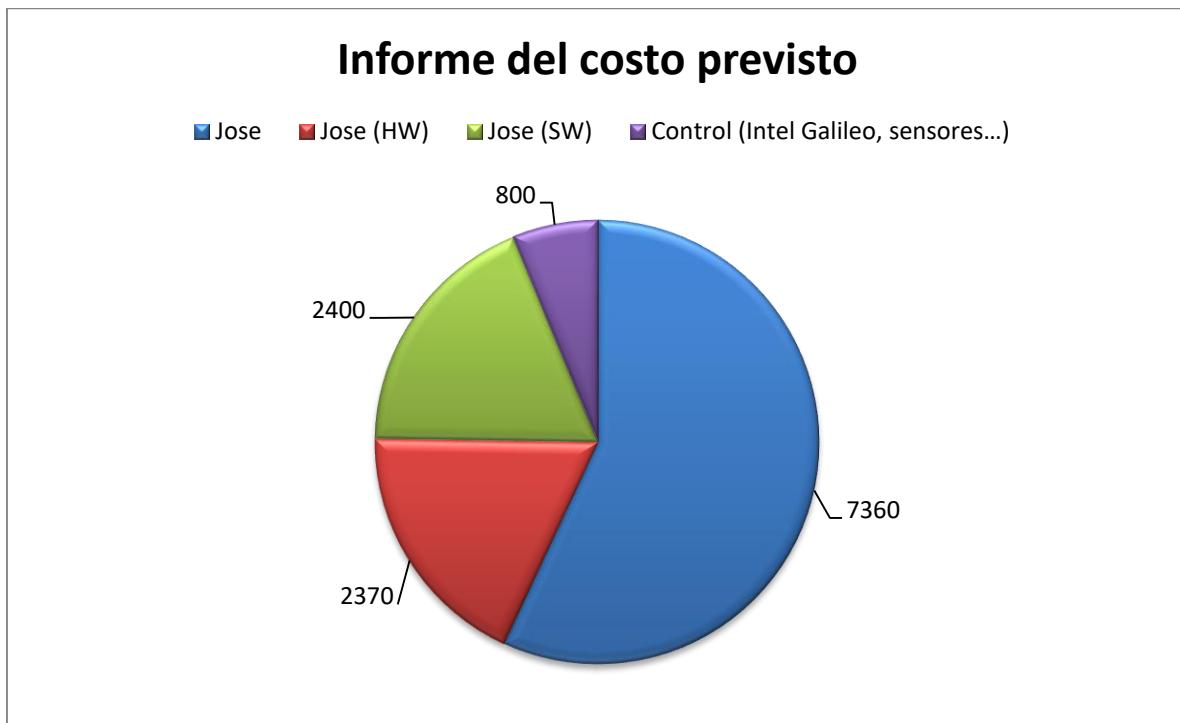


Fig. 9. Diagrama de distribución de costes entre productos

Diseño e Implementación

Descripción inicial de la arquitectura (sistema, subsistemas e interfaces) y posterior análisis a más bajo nivel de la implementación de los elementos que la forman.

TÍTULO DOCUMENTO	Documento de diseño e implementación del sistema
FECHA DE ENTREGA	12/12/2016
AUTOR/ES	José Miguel Ríos Rubio

COMENTARIOS	VERSIÓN	FECHA
Versión Inicial	1.0	25/11/2016

ÍNDICE

- 1 Introducción
- 2 Aspectos Generales del Sistema
- 3 Aspectos Específicos del Sistema
 - 3.1. Intel Edison
 - 3.2. Weather Shield
 - 3.1 Pantalla LCD
 - 3.2 Pulsadores
 - 3.5. Fotorresistencia
 - 3.3 Aplicación UWP (Universal Windows Platform)
- 4 Implementación del Sistema
 - 4.1 Hardware (Sistema METEO)
 - 4.2 Software (Sistema METEO)
 - 4.2.1 Preparando el Sistema Intel Edison
 - 4.2.2 Preparando el Proyecto de Intel System Studio
 - 4.2.3 Desarrollando la Inteligencia del Sistema METEO
- 5 Programa de Monitorización y Control (MeteoPanel)
 - 5.1 Preparando el Equipo para Desarrollar Aplicaciones Universales (UWP)
 - 5.2 Desarrollando la Aplicación Universal

METEO: Estación de monitorización meteorológica



1 Introducción

A partir de las especificaciones y requisitos del proyecto, definidos en la sección anterior, se procede a dar una visión global del sistema a desarrollar; y con ella, seremos capaces de proporcionar un enfoque más detallado y específico del mismo. Con esto, estaremos definiendo el **diseño** de nuestro sistema y una vez que lo tengamos, pasaremos a la **implementación** del mismo.

De este modo, comenzaremos por establecer el diagrama de bloques general y, a partir de ahí, saldremos del nivel de *ingeniería de sistemas* y pasaremos a determinar los componentes exactos que conforman al sistema, así, entraremos de lleno en el proceso de desarrollo del mismo, dentro del cual se recorrerá todo el proceso de su implementación, tanto de los aspectos Hardware, como del Software.

2 Aspectos Generales del Sistema

Dada la necesidad existente de conocer los parámetros ambientales y según las especificaciones, podemos determinar que nuestro sistema estará compuesto por una serie de **sensores atmosféricos**. Estos sensores se encargarán de medir los valores de temperatura, humedad y presión proporcionados como requisitos en el apartado anterior, además, de forma adicional vamos a añadir un sensor de luminosidad para que el sistema sea más versátil. Todos estos valores medidos deben de ser tratados mediante algún **sistema “inteligente”** que permita registrar y manejar dicha información y, este, conformaría el núcleo del sistema. Una vez tratados, los valores correspondientes a los parámetros atmosféricos deben de ser **informados a los usuarios** tanto por el propio hardware del sistema, haciendo uso de algún elemento de **representación visual**, como mediante un **software específico** que se ejecute en un ordenador personal. A su vez, el usuario debe de tener **herramientas para interaccionar** con el sistema y vendrán ligadas a los elementos informativos (tanto en el *Hardware* como en el *Software*).

Según las especificaciones relacionadas con las partes que componen al sistema, determinamos el diagrama de bloques general del mismo, el cual estará compuesto por cada uno de los elementos anteriores y se muestra en la figura 1.

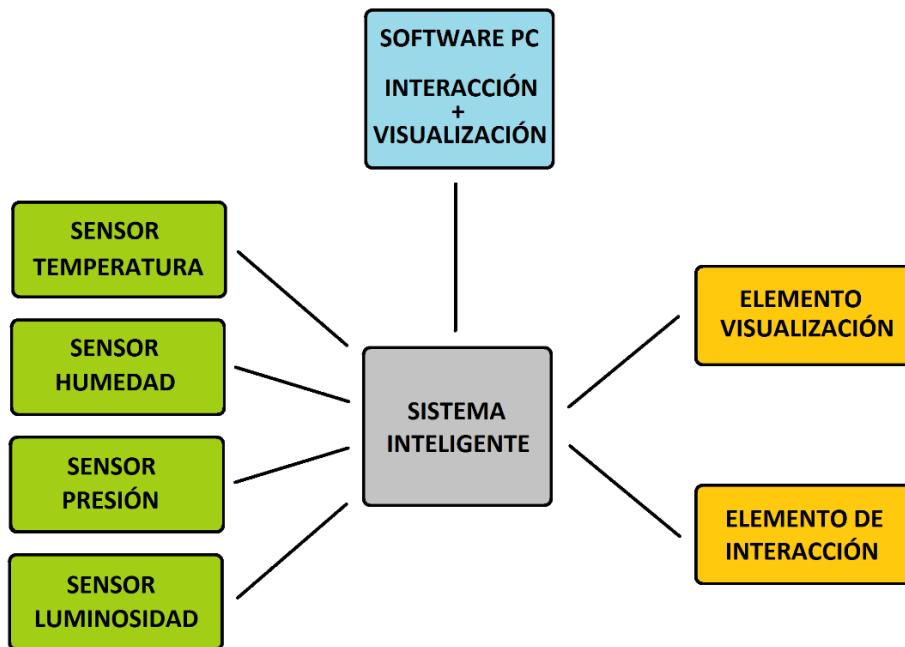


Fig. 1. Diagrama de bloques genérico

3 Aspectos Específicos del Sistema

Una vez identificados los distintos bloques que conforman al sistema, pasamos a identificar la forma de relacionarse que tienen cada uno de estos elementos y los componentes reales específicos a los que se corresponden. Comenzaremos por determinar los elementos partícipes en nuestro sistema y el porqué de la elección de los mismos para, posteriormente, entrar a detallar las características de cada uno de ellos.

Nuestro sistema hardware, con todos los elementos que lo componen, puede clasificarse como un *sistema empotrado/embibido* (del inglés, *Embedded system*) ya que cumplirá con funciones dedicadas de propósito específico.

Como núcleo del sistema, el elemento inteligente, se utilizará una placa ***Intel Edison** (como su propio nombre indica, de Intel), una solución empotrada enfocada al diseño y desarrollo de sistemas *IoT* (*Internet of Things*) específicos. Por tanto, no debe de confundirse con ordenadores de placa reducida o *SBC* (del inglés, *Single Board Computer*) como las plataformas *Raspberry Pi*, ya que estas son de propósito general.

Cada uno de los sensores de temperatura, humedad y presión deben de ser capaces de comunicarse con la Intel Edison. Las alternativas posibles van desde sensores de salida analógica, salida de voltaje analógico y lectura de la misma mediante el *conversor analógico-digital* o *ADC* (del inglés, *Analog to Digital Converter*) de la Intel

* En un principio se comenzó el proyecto con la Intel Galileo Gen 2, no obstante, debido a problemas técnicos con la misma se tuvo que trasladar el proyecto a la Intel Edison (Consultar el anexo “Problemas en el proceso de Implementación”).

Edison, hasta sensores de salida digital, salida de voltaje digital a través de algún protocolo de comunicación (*I2C, SPI, 1-WIRE...*) que debe de poder ser leído desde la Intel Edison. Entre estas dos posibilidades, optamos por **sensores digitales**, esto debido a las ventajas que proporcionan:

- Tratamiento previo, realizado por el propio sensor, de las lecturas de los parámetros a medir, reduciendo los errores medibles y aumentando la precisión y fiabilidad de las lecturas provenientes del transductor.
- Calibrado interno que los hacen más fiables.
- Simplicidad del conexionado, debido a un menor número de conexiones por el hecho de utilizar protocolos que comparten bus de comunicación.

De entre los sensores digitales disponibles en el mercado, buscamos aquellos que, cumpliendo con los rangos mínimos y máximos de lectura, necesarios por un sistema de monitorización meteorológica y, teniendo en cuenta de que no necesitamos una precisión milimétrica, sean de un coste asequible. Otro parámetro considerable es la disponibilidad de librerías implementadas que nos agilicen y simplifiquen el proceso de desarrollo software, facilitando el control de los sensores. Teniendo presente los factores anteriores se seleccionan los siguientes sensores: el **sensor de presión atmosférica y temperatura MPL3115A2**, y el **sensor de humedad y temperatura HTU21D**.

Para simplificar la conexión de los sensores, aprovechamos la compatibilidad de la placa Intel Edison con las placas Arduino, esto es, que presentan una disposición de pines *GPIO* (*General Purpose Input Output*) compatibles con los *Shields de Arduino*, que permite, si se usa un *Shield* que contenga los sensores, una integración directa y compacta entre el elemento inteligente y los sensores, por ello, pasará a utilizarse el **Wheater Shield**, de SparkFun, que integra los sensores presentados anteriormente y permite una comunicación con estos a través del protocolo **I2C** (*Inter-Integrated Circuit*).

Por otro lado, el **sensor de luminosidad** adicional a utilizar, es el basado en **fotorresistencia** (*LDR*, del inglés, *Light Dependent Resistor*), con una salida de voltaje analógica y lectura de ella mediante el **ADC** de la Intel Edison, pues no se requiere de un valor exacto del nivel de luz en una escala del sistema internacional de medida, como por ejemplo el Lumen, sino que nos basta con determinar en un rango de 0 – 100 el porcentaje de luminosidad existente respecto a un día normal (siendo 0% completa oscuridad, y 100% el momento del día en el que haya más luz ambiente).

Como elemento **visualizador hardware**, para mostrar los datos leídos de los parámetros ambientales, se recurre a un ***LCD** de 16 columnas por 2 filas controlado por la Intel Edison mediante **salidas digitales**; mientras que el elemento de **interacción hardware** se compondrá por un ***conjunto de 3 botones** conectados con la Intel Edison a través de **entradas digitales**.

* Inicialmente se procedió a utilizar un Shield que contenía un LCD y varios pulsadores y permitía una comunicación I2C, sin embargo, el cambio de la Intel Galileo a la Intel Edison generó una incompatibilidad que nos obligó a utilizar un LCD y pulsadores por separado (Consultar el anexo “Problemas en el proceso de Implementación”).

Por último, el **elemento software** de nuestro sistema, el cual complementará al sistema hardware y conformará, junto a éste, la plataforma *METEO* completa, vendrá dado por un programa de ordenador que permite la visualización e interacción, por parte del usuario, con el sistema *METEO*. La comunicación entre la Intel Edison y este programa se debe de realizar (según las especificaciones) en Red mediante **TCP y/o UDP** y las alternativas de desarrollo posibles son variadas. En un principio se planteó la posibilidad de utilizar el *Framework QT*, que brinda la capacidad de elaborar una interfaz de usuario gráfica y permite desarrollar programas compatibles tanto en sistemas basados en *Unix* (como *Linux*), *Mac OS X*, sistemas *Windows* e incluso *Android*. No obstante, como ya se posee el conocimiento en desarrollo mediante dicho Framework y por propia búsqueda de nuevos conocimientos, se optó por investigar alguna otra solución que nos permita satisfacer nuestras necesidades. De entre las diversas opciones, se buscaba alguna herramienta novedosa con posibilidades de implantarse en el futuro como solución prometedora para el desarrollo de este tipo de programas con entorno gráfico; debido a esto se seleccionó lo que se conoce por **Universal Windows Platform (UWP)** una plataforma de desarrollo de aplicaciones (entendiéndose estas como software), de Microsoft, que permite elaborar aplicaciones para la plataforma *Windows 10*, esto es, que no se enfoca en el desarrollo de un tipo de dispositivo con un sistema operativo en concreto, sino que se centra en un desarrollo por plataforma, por lo que desarrollar una aplicación UWP permite que esta sea compatible en cualquier tipo de dispositivo que presente *Windows 10*, ya sea éste un ordenador personal, un Smartphone, una Tablet, un dispositivo de realidad aumentada (como las futuras *Hololens*), un sistema empotrado, etc.

Conocidos todos los componentes específicos que participan en nuestro sistema, podemos volver a representar el diagrama de bloques correspondiente y, esta vez, con los elementos reales que lo componen. La figura 2 muestra este nuevo diagrama.

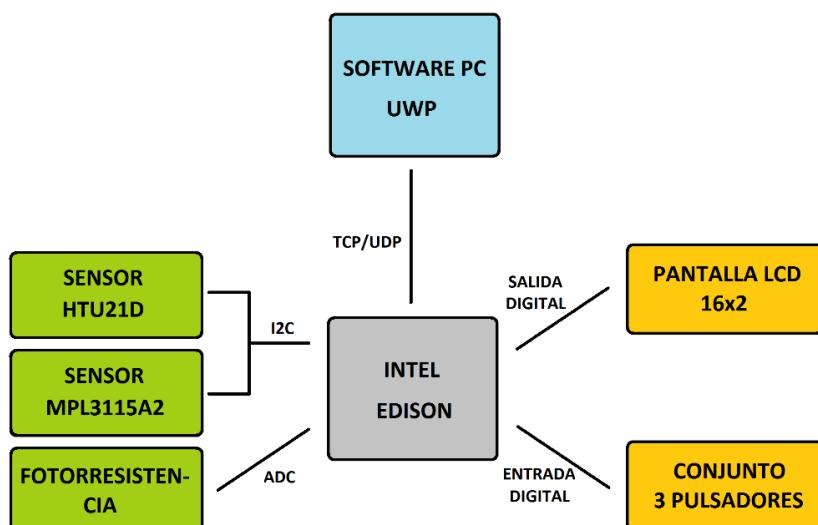


Fig. 2. Diagrama de bloques específico

10.1 Intel Edison

Como hemos dicho, el núcleo de nuestro sistema lo formará la Intel Edison, la cual se puede clasificar como un sistema empotrado enfocado en el desarrollo de productos de IoT.

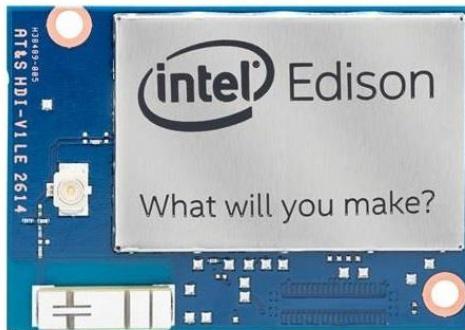


Fig. 3. Módulo de computación Intel Edison

El elemento principal de un sistema Intel Edison es un pequeño módulo de cómputo, representado en la figura 3 [1], el cual presenta las especificaciones recopiladas en la siguiente tabla [1]:

PHYSICAL	
Form factor	Board with 70-pin connector
Dimensions	35.5 × 25.0 × 3.9 mm (1.4 × 1.0 × 0.15 inches) max
C/M/F	Blue PCB with shields / No enclosure
Connector	Hirose DF40 Series (1.5, 2.0, or 3.0 mm stack height)
Operating temperature	32 to 104°F (0 to 40°C)
EXTERNAL INTERFACES	
Total of 40 GPIOs, which can be configured as:	
SD card	1 interface
UART	2 controllers (1 full flow control, 1 Rx/Tx)
I2C	2 controllers
SPI	1 controller with 2 chip selects
I2S	1 controller
GPIO	Additional 12 (with 4 capable of PWM)
USB 2.0	1 OTG controller
Clock output	32 kHz, 19.2 MHz
MAJOR EDISON COMPONENTS	
SoC	22 nm Intel® SoC that includes a dual-core, dual-threaded Intel® Atom™ CPU at 500 MHz and a 32-bit Intel® Quark™ microcontroller at 100 MHz
RAM	1 GB LPDDR3 POP memory (2 channel 32bits @ 800MT/sec)
Flash storage	4 GB eMMC (v4.51 spec)
WiFi	Broadcom® 43340 802.11 a/b/g/n; Dual-band (2.4 and 5 GHz) Onboard antenna
Bluetooth	Bluetooth 4.0
POWER	
Input	3.3 to 4.5 V
Output	100 ma @3.3 V and 100 ma @ 1.8 V
Power	Standby (No radios): 13 mW Standby (Bluetooth 4.0): 21.5 mW (BTLE in Q4-14) Standby (Wi-Fi): 35 mW
FIRMWARE + SOFTWARE	
CPU OS	Yocto Linux® v1.6
Development environments	Arduino® IDE Eclipse supporting: C, C++, and Python Intel XDK supporting: Node.JS and HTML5
MCU OS	RTOS
Development environments	MCU SDK and IDE

Tabla. 1. Especificaciones del módulo de cómputo Intel Edison

Este módulo posee un conector Hirose DF40 de **70 pines** que sacan las líneas directamente del SOC (System On Chip) y **no son adecuadas para su uso directo**. La solución a esto se basa en el uso de una placa de desarrollo que nos convierta el nivel de voltaje de las señales de salida de este conector (que se encuentran con valores de 1.28V y 3.3V) a valores adecuados y comunes para nuestro sistema (3.3V y/o 5V), adapte las interfaces de los periféricos del sistema (líneas de I2C, SPI, UART...) y proporcione conectores tanto para los pines GPIO, como para conectar la fuente de alimentación, y conectores **USB** (*Universal Serial Bus*), de programación/depuración, y que permita el uso de la capacidad **USB-OTG** (*On The Go*) del Intel Edison.



Fig. 4. Breakout Boards Intel Edison

Existen varias placas de desarrollo para Intel Edison, como se puede apreciar en la figura 4; de entre todas ellas, se utilizará la llamada **Arduino Breakout Board**, mostrada en la figura 5, la cual proporciona compatibilidad con los sistemas Arduino. Esta compatibilidad se presenta en forma de una disposición de pines **GPIO** posicionados de la **misma forma y distancias** que los presentes en las **placas Arduino estándar** (como los Arduino UNO, por ejemplo) lo que permite el uso de Shields Arduino en la misma (que nos permitirá, como vimos en el apartado anterior, conectar el Shield que contiene los sensores) además, la placa presenta un **conector de alimentación**, un conector **USB** hembra (USB Host) y **dos micro-USB** (uno para alimentar la placa y programarla, y otro para la depuración a través de un puerto serie).

Otro aspecto destacable es la presencia de diversos **Jumpers** que nos permiten configurar propiedades relacionadas con las señales GPIO de la placa, como por ejemplo la configuración de **voltajes de referencia de las señales** (3.3V o 5V), el **voltaje de referencia del ADC** y **configuraciones** relacionadas a las **señales PWM**.

Cabe mencionar la presencia de un **zócalo** para una tarjeta **micro-sd** que le da al sistema la posibilidad de almacenar y arrancar una distribución Linux de mayor capacidad que la que se encuentra en la eMMC.

Por último, la placa presenta **5 pulsadores** en ella para el control de diversos aspectos de la misma [2]: apagado/encendido de la placa, reset del núcleo Intel Edison, reset de los Shields conectados, recuperación de imágenes corruptas y recuperación del firmware de la placa; como se puede apreciar, todos estos pulsadores tienen un **propósito específico** dentro del sistema y **no están pensados para ser usados** por el usuario, **en la aplicación que se desarrolle**.

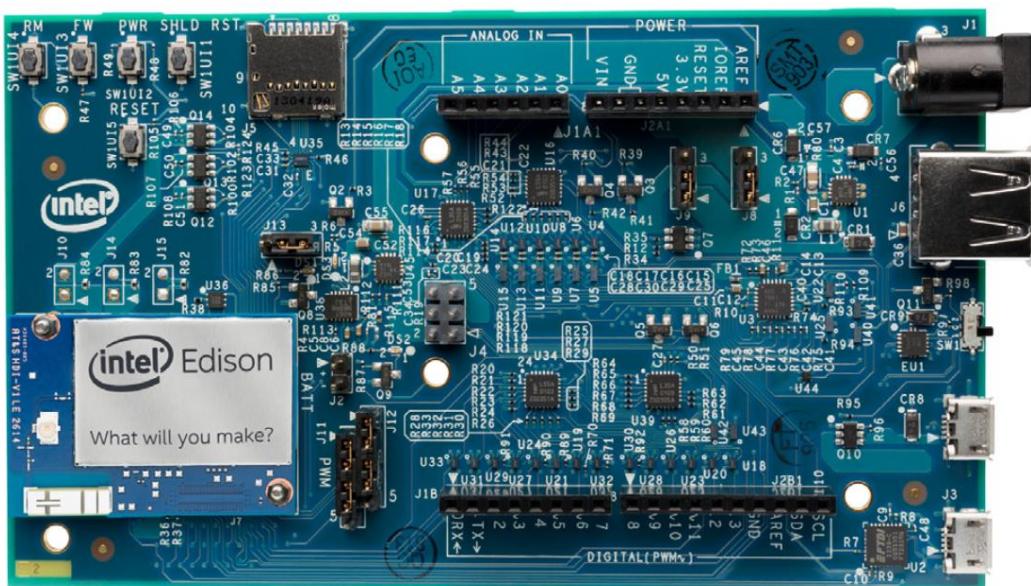


Fig. 5. Intel Edison Arduino Breakout Board

Una vez repasados todos los aspectos hardware más destacables de la placa, vamos a profundizar en su forma de uso y las posibilidades de desarrollo que ofrece.

Como ya se ha dicho, la Intel Edison está planteada para el desarrollo de productos IoT, sin embargo, no ha llegado a tener un gran éxito, y no es de extrañar, dado que el coste monetario para su adquisición, se aleja mucho de otros sistemas que representan competencia.

Queriendo conservar la simplicidad que proporciona el caso de éxito que ha supuesto la plataforma Arduino, la placa **Intel Edison** da **soporte y compatibilidad a una programación** a través de la propia capa de abstracción HAL (del inglés, *Hardware Abstract Layer*) de **Arduino**, esto es, que la placa puede programarse con el mismo “lenguaje” que las placas Arduino y a través del mismo entorno de desarrollo Arduino IDE (*Integrated Development Environment*). Sin embargo, la plataforma Arduino está enfocada a un público que será su propio consumidor, no está orientada exclusivamente

a desarrolladores con conocimientos avanzados, sino a un público que consigue un aprendizaje rápido e intuitivo sin precisar de conocimientos técnicos sobre electrónica, para hacer funcionar un proyecto, lo que permite que la plataforma llegue a cualquier persona. Al contrario de esto, la plataforma Intel Edison, debido a su precio, está más enfocada en ser un sistema de desarrollo para empresas del sector tecnológico, que permita un desarrollo rápido de productos, o instituciones académicas que hacen uso de la misma para enseñanza (como es el caso de nuestra Universidad). Por esto mismo, ya que los usuarios de Intel Edison son desarrolladores con conocimientos más avanzados, se proporcionan otras opciones de desarrollo adaptadas a diferentes lenguajes de programación, que pueden ser: **C/C++, JAVA, Python, JavaScript y Node.js**.

Resumiendo lo anterior, podemos decir que la plataforma Intel Edison puede ser programada mediante dos líneas principales diferentes [3]:

- A través del **Arduino IDE** y como si una placa Arduino se tratase (con la **simplicidad y versatilidad del código Arduino**).
- Mediante **otros IDEs** (a excepción de Python) **permitiendo depuración y aspectos más avanzados** del mismo (**Intel System Studio IoT Edition**, para C/C++/JAVA; o **Intel XDK**, para JavaScript/Node.js).

En el **firmware** interno del módulo eMMC del núcleo de computo **Intel Edison** se encuentra un sistema operativo estándar basado en **Yocto** [4] (un proyecto de código abierto que proporciona plantillas, herramientas y métodos necesarios para la creación de sistemas Linux empotrados personalizados).

Como se verá en el apartado *Implementación*, de entre estas dos posibles líneas de desarrollo, en este proyecto nos enfocaremos en la segunda de ellas, para sobretodo, conseguir la ventaja de la depuración, no obstante, además, se consigue añadir la capa de abstracción de *Arduino* obteniendo la ventaja de simplicidad de código y el poder utilizar librerías ya implementadas, obteniendo así, las ventajas de las dos líneas de desarrollo.

10.2 Weather Shield

El **Wheater Shield**, que es mostrado en la figura 6, es un Shield Arduino de *SparkFun*, orientado a satisfacer las necesidades de medición de parámetros ambientales, por lo que es óptimo para el desarrollo de estaciones meteorológicas. Este Shield integra los sensores de presión barométrica **MPL3115A2**, de humedad **HTU21D** y de luz **ALS-PT19**; además presenta diversos conectores que permiten acoplar sensores externos de lluvia, viento y localización *GPS* (del inglés, *Global Positioning System*) ampliando así la funcionalidad del mismo.

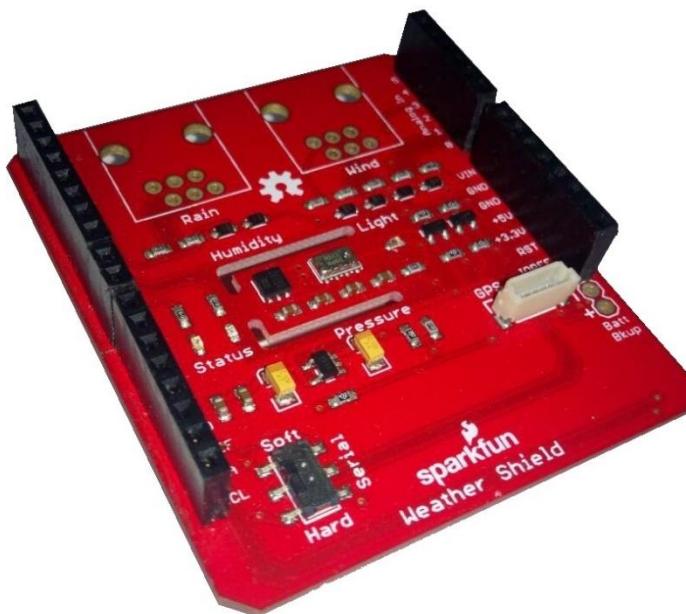


Fig. 6. Sparkfun Weather Shield

Los sensores presentes en el Shield se comunican mediante el **protocolo I2C** a través de las líneas de dato **SDA** (*Serial Data*) y reloj **SCL** (*Serial Clock*) del bus, que corresponden conjuntamente con los pines **A4-A5** y **SDA-SCL** de los pines de entrada/salida del Shield.

El sensor **MPL3115A2** es un sensor de **presión** barométrica [5] con interfaz digital I2C y un rango de operación de **20KPa** a **100KPa**. El sensor está compuesto por un transductor piezoelectrónico de tecnología microelectromecánica **MEM** (*MicroElectroMechanical*) que permite obtener los parámetros de presión y un sensor de **temperatura** con un rango de **-40°C** a **85°C**. La medición de temperatura permite compensar los errores de la lectura de presión debido a esta. Ambas mediciones son tratadas internamente por un ADC de alta resolución, digitalizándose y devolviendo sus valores en pascales y grados centígrados; la figura 7 [5] muestra el diagrama de bloque interno de este sensor. La capacidad de medir la presión atmosférica permite obtener la **altitud respecto al nivel del mar**, a la que nos encontramos, gracias a la relación existente entre ellas (a nivel del mar nos encontramos a 100KPa), por lo que se podría obtener el valor de altitud en el rango de **-698m** a **11775m**. Como resumen, podemos decir que con este único sensor se puede determinar **presión, temperatura y altitud**.

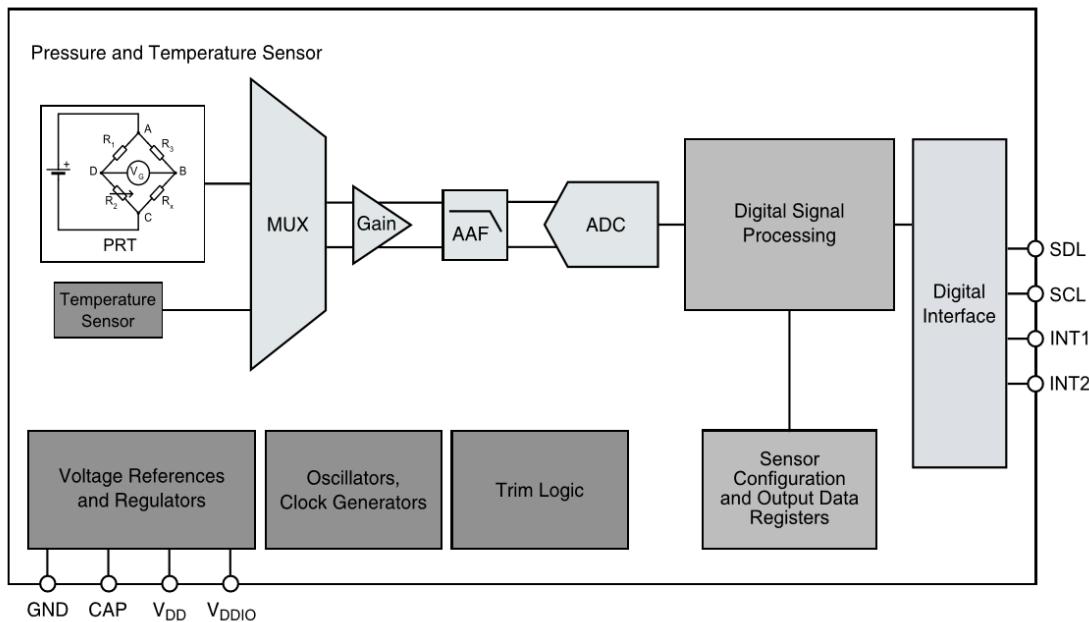


Fig. 7. Diagrama de bloques interno del sensor MPL3115A2

Por otro lado, el sensor **HTU21D** es un sensor de **humedad relativa** (relación existente entre la cantidad de vapor de agua que tiene una masa de aire respecto a la máxima que podría tener a dicha temperatura) [6] con interfaz digital I2C y un rango de operación de **0%** a **100%**. Este sensor presenta una lectura completamente **calibrada** y compensada en temperatura, para ello, al igual que con el sensor de presión, se realiza una lectura interna de **temperatura** y, esta, también es entregada en la salida del sensor. Es un sensor orientado a aplicaciones de bajo consumo y la resolución de las lecturas de humedad y temperatura puede ser configurada. Con este sensor, es posible obtener medidas de *humedad* y *temperatura*.

El sensor **ALS-PT19** es un sensor lumínico basado en fototransistor [7] y de salida analógica, el cual, **no se utilizará en este proyecto**, por lo que no definiremos ni las especificaciones ni las características de este. Esto es debido a la posibilidad de que el sistema se encuentre en el interior de una caja/carcasa de protección (que podría provocar que el sensor quedara aislado de luminosidad). En cambio, se utilizará un sensor externo basado en **foterresistencia**.

El **Wheater Shield** trabaja **internamente con un voltaje de 3.3 voltios**, aunque requiere de una **alimentación de 5V o más** (hasta 12V), voltaje que es regulado internamente por el propio Shield. Así mismo, ya que se presenta un voltaje interno de 3.3V, éste es utilizado en un conversor de voltaje para las líneas I2C, de modo que es compatible tanto para sistemas con niveles lógicos de 0-3.3V como de 0-5V.

Como único elemento de interacción en el Shield, se encuentra un *micro-interruptor* cuya función se corresponde con la configuración de los pines del puerto serie a utilizar (los

de la *UART*, hardware, o para un puerto serie software). Este puerto serie permite la comunicación con el módulo GPS externo opcional, que se puede conectar al Shield, por lo que será superfluo para este proyecto (ya que no se utiliza ningún módulo GPS).

3.1 Pantalla LCD

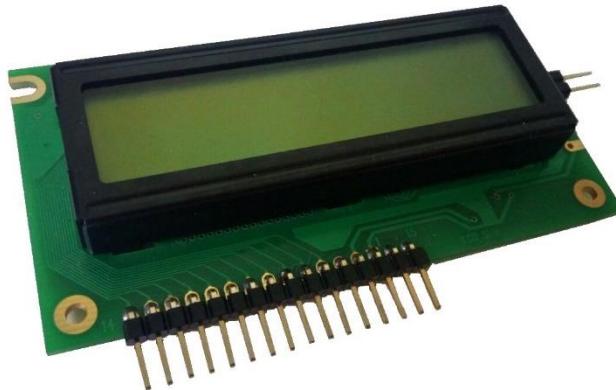


Fig. 8. Módulo LCD

En la figura 8 se muestra la pantalla LCD que se va a utilizar en el proyecto, se corresponde con el modelo **LMB162HBC de Topway** (figura 8), una pantalla **LCD alfanumérica matricial de 2 líneas y 16 columnas**, lo que permite representar 32 caracteres visibles. Entre otras características cabe destacar la **posibilidad de regular el contraste** de la matriz y la **disponibilidad de iluminación de fondo** o *Backlight* para permitir la visibilidad de la pantalla cuando hay ausencia de luz natural.

La **alimentación** del módulo depende de los últimos caracteres del modelo en cuestión (LMB162HBX, donde X corresponde a la alimentación del módulo). En este caso, para el modelo LMB162HBC, la alimentación es de **5V** y esto también implica que el nivel lógico '1' (alto) de las señales han de ser de 5V.

El LCD posee una **interfaz digital paralela**, esto es, que presenta un conjunto de pines para controlar tanto aspectos de configuraciones como la representación de datos [8].

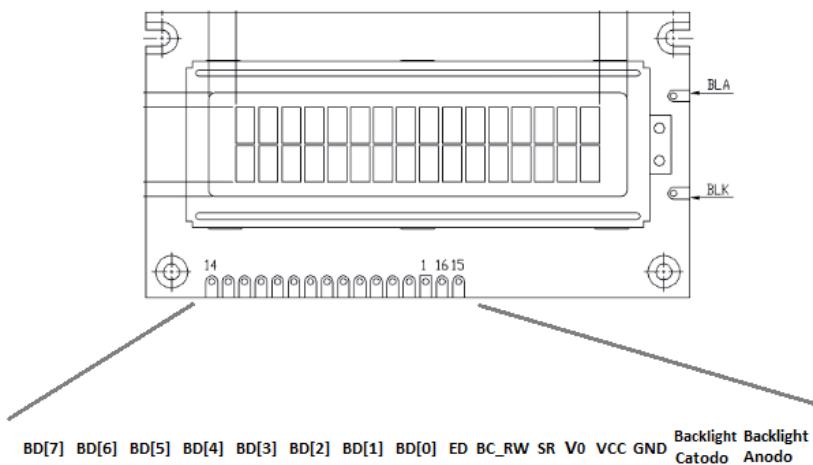


Fig. 9. Disposición de pines del LCD [8]

De la imagen anterior, la figura 9 [8], se puede observar que el LCD presenta un bus con 8 líneas de datos (*BD*), un pin de habilitación para estos datos (*ED*), una señal de control de escritura/lectura (*BC_RW*), una señal de selección de registro (*SR*), un pin para regular el nivel de contraste (*V₀*) y los pines correspondientes a la alimentación y el ánodo y cátodo de la iluminación del fondo (*Backlight*).

3.2 Pulsadores

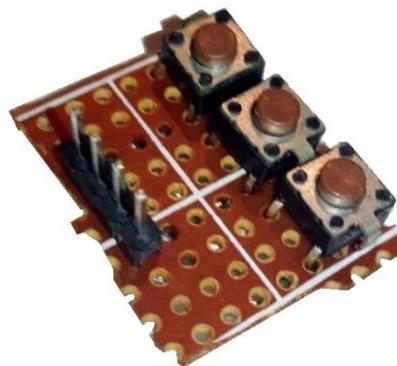


Fig. 10. Módulo Pulsadores

Para poder interaccionar con el sistema se utiliza un módulo que ha sido desarrollado a mano sobre una placa perforada de prototipado y que está compuesto por tres pulsadores con salidas independientes. La figura 10 muestra dicha placa.

El proceso de desarrollo, así como la conexión exacta, la funcionalidad de cada pulsador y la utilización del módulo se detallará en profundidad en el apartado de Implementación.

10.5 Fotorresistencia



Fig. 11. Módulo Fotorresistencia

Como se ha comentado, para evitar posibles problemas en la recepción de luz mediante el sensor que se encuentra en la placa *Weather Shield*, cuando ésta se encuentra integrada en una caja, recurrimos al uso de un sensor basado en fotorresistencia elaborado sobre una placa perforada de prototipado, y que está compuesto por una fotorresistencia y una resistencia, conectadas como divisor de tensión, y tres pines machos, de modo que constituya un módulo conectable mediante cables, lo que permite que éste se sitúe en un lugar óptimo para la recepción de luz. La figura 11 muestra el módulo desarrollado.

El proceso de desarrollo, así como la conexión exacta, y la utilización del módulo se detallará en profundidad en el apartado de Implementación.

3.3 Aplicación UWP (Universal Windows Platform)



Fig. 12. Universal Windows Platform (UWP) [9]

La **plataforma universal de Windows (UWP)** [9], desarrollada y perteneciente a Microsoft, es una plataforma orientada al desarrollo de las llamadas **aplicaciones universales**, aplicaciones que son **comunes en código** y orientadas a sistemas que presenten **Windows 10** (y sus variantes), **indistintamente del tipo de dispositivo** (esto es representado en la figura 12), un ordenador personal, un Smartphone, un sistema empotrado... una aplicación universal será compatible en todo tipo de dispositivos.

Para conseguir esta compatibilidad, la UWP presenta no solo una **única base de código**, sino también un mismo conjunto de **APIs (Application Programming Interface)** y un **interfaz de usuario adaptable a cada dispositivo**, con un **lenguaje de diseño común** y un **único conjunto de controles**.

La UWP hace uso del llamado *Windows Runtime*, una *API* nativa localizada dentro del sistema y que permite crear aplicaciones universales mediante los lenguajes de programación más reconocidos para los desarrolladores de sistemas de *Microsoft*: **C#, C++, Visual Basic y JavaScript**.

Para el diseño de la interfaz gráfica se utiliza **XAML** (*eXtensible Application Markup Language*), un lenguaje de modelado/marcado basado en *XML*.

Analizando lo que la UWP supone, podemos apreciar que **Microsoft busca estandarizar el proceso de desarrollo software** de la gran variedad de familias de dispositivos que se pueden encontrar hoy en día en el mercado, así como de los futuros dispositivos que vendrán, todo ello con el fin de posicionarse en un mejor lugar frente a la competitividad que, sobre todo, están suponiendo los sistemas *Android* en dispositivos portátiles inteligentes, como los *Smartphones* (evidentemente *Android* se ha extendido y es el sistema líder, muy superior frente a sistemas *Windows Phone*).

Este proceso de estandarización comenzó con la creación de *Windows 8*, un sistema operativo a medio camino entre un sistema portátil (Smartphone y Tablet) y un ordenador personal dotándole al sistema de características de ambos tipos de dispositivos. No obstante, *Windows 8* se encontraba más cercano a las interfaces de usuario de dispositivos portátiles, que al de los ordenadores personales y esto provocó rechazo por parte de los usuarios. Una vez adquirida la experiencia y reacción del público sobre *Windows 8*, Microsoft dio un paso más y desarrollo *Windows 10*, un sistema operativo al que denominaron por **universal** dada su compatibilidad con todo tipo de dispositivos y el cual establece un tipo de interfaz de usuario dependiendo del dispositivo que se utilice (para que no pasará como con *Windows 8*).

Aunque desde la implantación de *Windows 8* ya se hablaba de una “plataforma universal”, no fue hasta que se introdujera *Windows 10* cuando pasó a ser la **Windows Universal Platform** tal y como se conoce actualmente. Así, *Microsoft* introdujo *Windows 10* como apuesta para un sistema operativo “universal”, y en la *Build 2015* (conferencia de *Microsoft* para desarrolladores, realizada en abril de 2015), presentó la *Windows Universal Platform (UWP)* y sus posibilidades [9].



Fig. 13. Capa de abstracción UWP

La universalidad y amplia compatibilidad que ofrece la UWP puede ser caracterizada como una **capa de abstracción**, representada mediante la figura 13, que permite desarrollar aplicaciones universales que funcionen en distintas familias de dispositivos. La única variación en un proyecto UWP para dos familias de dispositivos distintos radicaría en el entorno gráfico del mismo. De este modo, una aplicación universal puede ser desarrollada de forma que según el dispositivo en el que se ejecute, la misma tome las configuraciones necesarias de la familia de dispositivos correspondientes.

4 Implementación del Sistema (Dispositivo)

Una vez que tenemos una idea exacta del diseño de la plataforma METEO, y los componentes participantes en la misma, procedemos a detallar el proceso de implementación.

En este apartado se repasarán los aspectos más destacables del proceso de desarrollo del **hardware del sistema** (tanto la interconexión de los módulos como el porqué de dichas conexiones), del **software correspondiente al sistema inteligente** (*Intel Edison*) y del **software correspondiente al programa de monitorización y control** del sistema METEO, desarrollado a través de *UWP*.

4.1 Hardware (Sistema METEO)

Según el resultado del diseño establecido, los componentes físicos que conformarán al sistema METEO son: el sistema **Intel Edison con su breakout board Arduino correspondiente** (a partir de ahora, nos referiremos a este conjunto simplemente como Intel Edison), como elemento “inteligente”, el **Weather Shield**, para la adquisición de los parámetros ambientales de presión humedad y temperatura, el módulo **fotorreceptor**, para la luminosidad, un **LCD**, que mostrará los resultados de las medidas, y **tres pulsadores** que nos permitan interaccionar con el sistema.

La implementación hardware del sistema comienza por la conexión de la Intel Edison con la *Weather Shield*, para ello, debería bastar con insertar este *Shield* en los pines correspondientes de la placa *Intel Edison Breakout Board Arduino* (ya que esta debería de ser totalmente compatible con los *Shield Arduino*) no obstante, existe una cierta **variación en la circuitería** de la Intel Edison que **impide que esto funcione***.

SPARKFUN WEATHER SHIELD

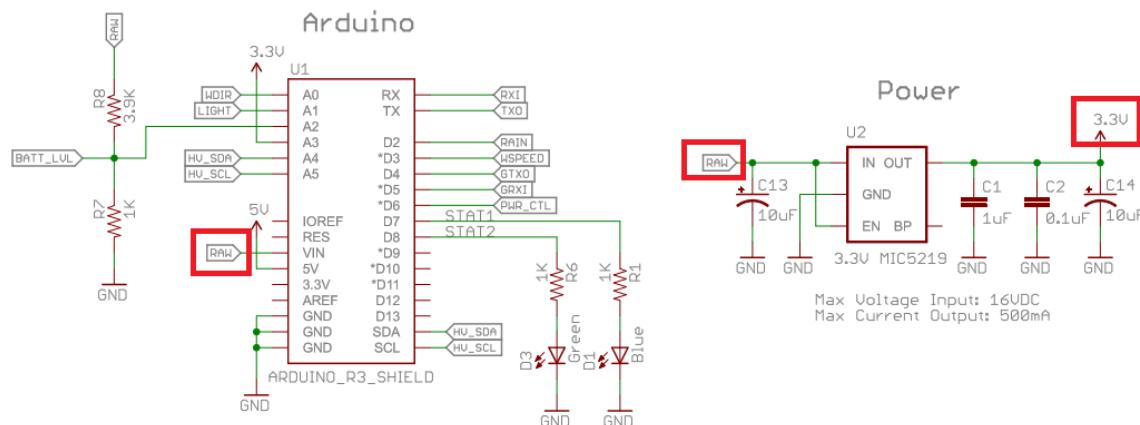


Fig. 14. Fragmento del esquema interno del Weather Shield [10]

La *Weather Shield* utiliza el pin V_{IN} para adquirir la alimentación de la placa y regularla a los 3.3V internos utilizados en ella, esto se puede ver en el diagrama eléctrico del *Shield*, correspondiente a la figura 14 [10]. En los *Arduino* [11] y en las *Intel Galileo* [12], este pin va directamente al regulador utilizado para regular a 5V el voltaje correspondiente a la tensión de entrada que se suministre por el conector DC y, cuando no se alimenta la placa a partir de dicho conector, la señal V_{IN} tomará un valor cercano a 5V (pues el voltaje localizado en la salida del regulador de la placa, como bien se dice en el *datasheet* de, por ejemplo, el regulador *NCP1117* [13] presente en los *Arduino UNO*, presenta un diodo de protección interno que interconecta la salida del regulador con la entrada del mismo, por ello el voltaje a la entrada se corresponderá con el voltaje a la salida (5V) menos el voltaje que cae en dicho diodo (~0.4V), es decir unos 4.6V). En la figura 15 se muestra el diagrama eléctrico de este regulador [13].

* Cuando se comenzó el desarrollo con la Intel Galileo Gen 2, no existía dicha incompatibilidad, pues se alimentaba la placa a través del conector dc y ese voltaje estaba presente en el pin V_{IN} (Consultar el anexo “Problemas en el proceso de Implementación”).

Protection Diodes

The NCP1117 family has two internal low impedance diode paths that normally do not require protection when used in the typical regulator applications. The first path connects between V_{out} and V_{in} , and it can withstand a peak surge current of about 15 A...

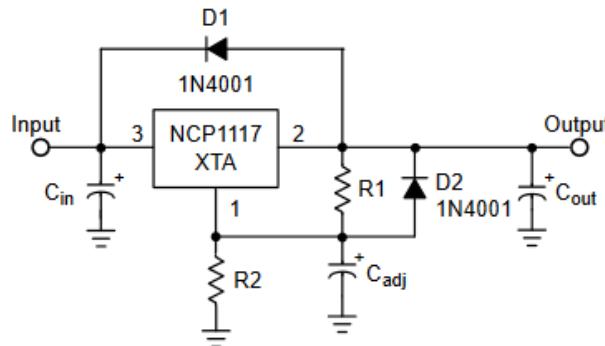


Fig. 15. Diodos internos del NCP1117 [13]

Por tanto, cuando no se suministra voltaje por el conector DC (solo se alimenta por USB), el pin V_{IN} debería presentar 5V, sin embargo, revisando el esquemático de la Intel Edison [14], resulta que ese pin está aislado (mediante un diodo) de tanto el voltaje USB, como de la alimentación DC, es exclusivamente un pin de entrada y, por tanto, no presenta el voltaje necesario (~5V) para ser regulado y utilizado por la *Weather Shield*. La siguiente imagen, la figura 16, muestra la comparación de este aspecto en los diagramas eléctricos de los sistemas *Arduino UNO*, *Intel Galileo* e *Intel Edison*.

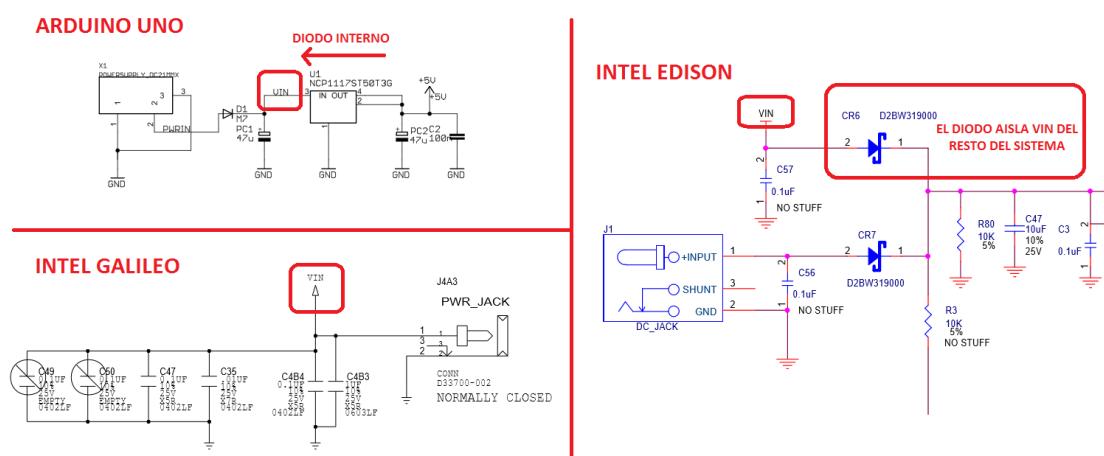


Fig. 16. Diferencias en V_{IN} según plataformas [11] [12] [14]

Debido a que la Intel Edison no suministra el voltaje necesario por el pin de alimentación V_{IN} , debemos de suministrarle al pin correspondiente de la *Weather Shield* el voltaje que

proporciona la *Intel Edison* por su pin 5V. Por tanto, la conexión a realizar entre la *Intel Edison* y el *Weather Shield* se corresponderá con la integración directa del *Shield*, a salvedad de los pines V_{IN} y 5V, los cuales deberán ser conectados de modo que **se proporcione los 5V de la placa Intel Edison al pin V_{IN} del Weather Shield**. En la siguiente imagen, la figura 17, se puede ver la conexión exacta de ambas placas, en ella, podemos diferenciar la conexión particular de **5V-V_{in}** mediante el cable de color **rojo**, además, el pin de 5V de la *Intel Edison* no se conectarán (a partir de ahora los 5V se encontrarán en el pin V_{IN} de la *Weather Shield*). Por otro lado, podemos apreciar las conexiones de **GND** mediante cables de color **negro**, las líneas correspondientes al **bus I2C** de color **verde**, conexiones de datos, de diversos **aspectos de los sensores de viento, lluvia y luz** (no utilizados en nuestra aplicación), en **azul**, y el **puerto serie** para la comunicación con el modulo GPS (tampoco utilizado en nuestro sistema) en **amarillo**. Por último, cabe decir que en la imagen se representa la totalidad de las conexiones, incluyendo aquellas que no presentan funcionalidad alguna en el *Weather Shield*, estas conexiones son las representadas con cables de color **gris** y, por tanto, quedan **disponibles y utilizables** para las futuras necesidades.

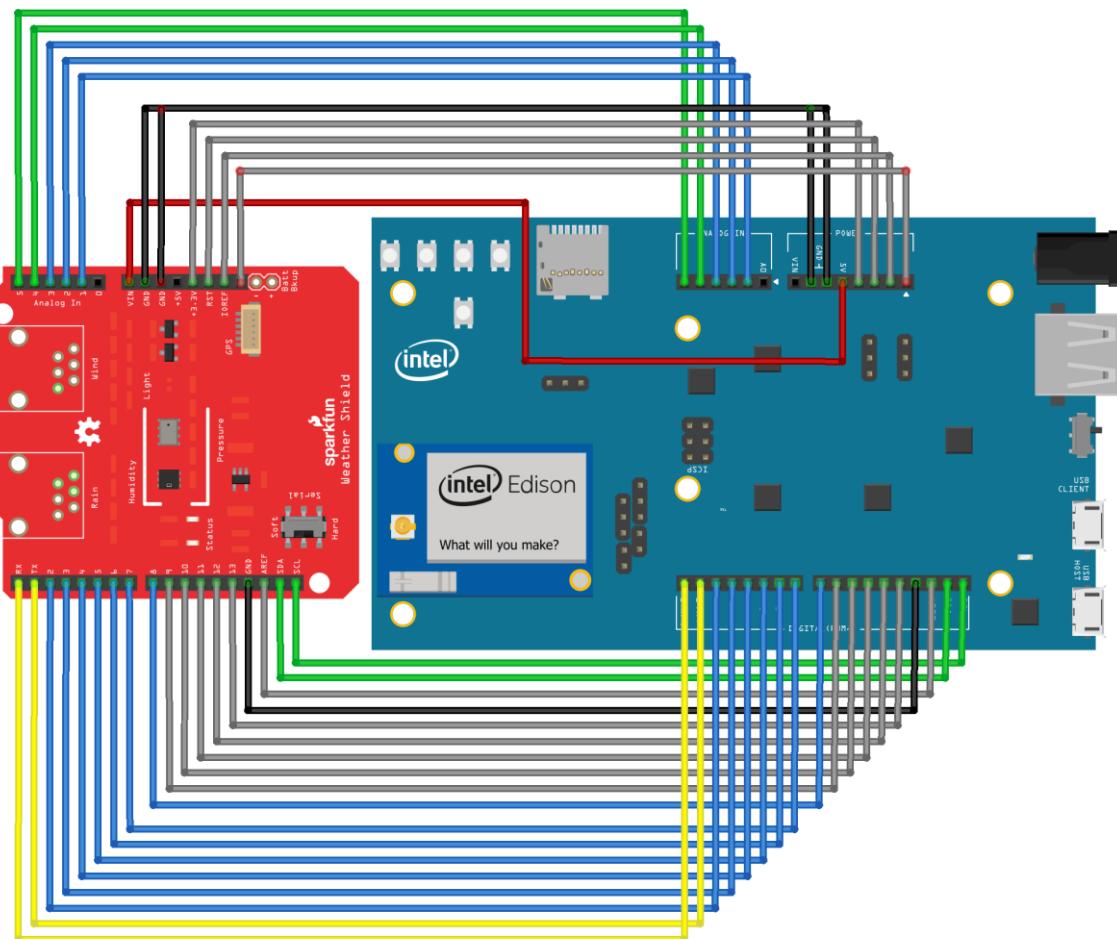


Fig. 17. Conexión Intel Edison con Weather Shield

Para realizar esta conexión en los componentes reales, comenzamos por **flexionar hacia dentro** los pines inferiores V_{IN} y $5V$ de la *Weather Shield*, de modo que, a la hora de integrar el *Shield* en la *Intel Edison*, estos **no encajen** con sus iguales, tras esto, preparamos un cable que conectamos al pin V_{IN} flexionado, tal y como se puede ver en la figura 18. Ya que necesitaremos un pin del ADC para realizar la lectura analógica del sensor (externo a la placa) basado en fotorresistencia, también flexionaremos el pin A_0 para, posteriormente conectar dicho sensor.

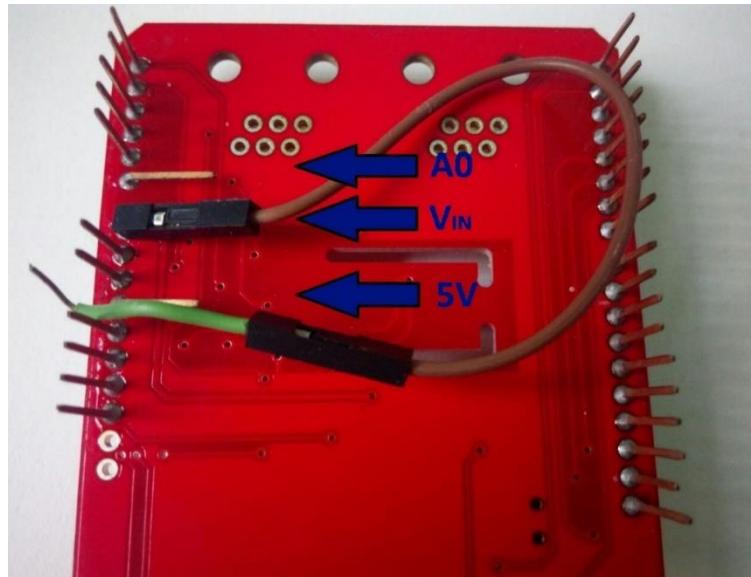


Fig. 18. Flexionado de los pines de la Weather Shield

Una vez hecho esto, conectamos el cable proveniente de V_{IN} al pin de $5V$ de la *Intel Edison* y **acoplamos** el *Shield* en ella, de forma que quede como en la figura 19.

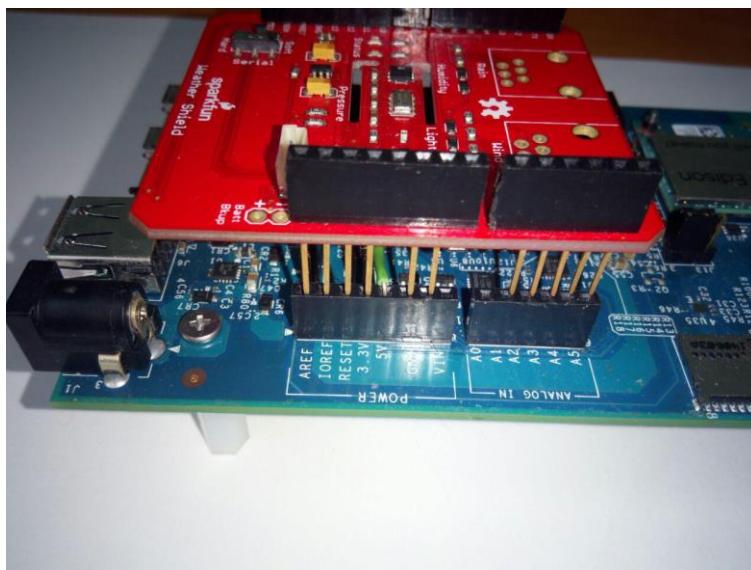


Fig. 19. Acoplando la Weather Shield en la Intel Edison

Llegados a este punto, se habrá **completado la conexión del Weather Shield con la Intel Edison**, quedando el sistema como puede verse en la figura 20, y, así, teniendo en cuenta las conexiones inferiores de los pines V_{IN} y 5V, podremos simplificar la representación del esquema de conexión tal y como se muestra en la figura 21.



Fig. 20. Intel Edison con *Weather Shield* integrada

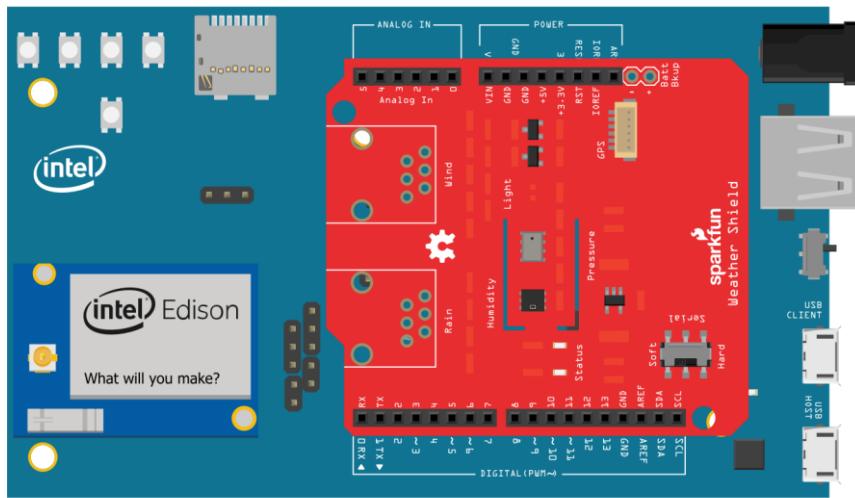


Fig. 21. Esquemático simplificado de conexión *Intel Edison* y *Weather Shield*

A continuación, procedemos a añadir el último sensor del sistema, el **sensor de luz**, el cual es ajeno al *Shield* utilizado, y ello permite posicionarlo en el lugar más óptimo para la recepción de luz.

Como se comentó en el apartado de diseño, el sensor lumínico estará basado en una **fotorresistencia**, por lo que, para determinar las variaciones del nivel de luz del entorno, se requiere de una configuración de esta como **divisor de tensión** junto a una resistencia fija que nos permita transformar el nivel de luz recibido en un valor de voltaje analógico. Para conseguir que el nivel de luminosidad se encuentre en el rango aceptable por el ADC (5V-0V o 3.3V-0V) establecemos un **voltaje de entrada** del divisor correspondiente al voltaje de **3.3V** que suministra la Intel Edison.

El fabricante de la fotorresistencia utilizada en nuestro sistema es desconocido al igual que el modelo de esta, por lo que no tenemos un *Datasheet* específico con el que poder calibrar su valor de lectura en lúmenes, ni tampoco consultar el rango de valores de resistencia que toma según el nivel de luz que recibe, sin embargo, el problema de los Lúmenes nos es superfluo, pues representaremos el nivel de luz mediante un **rango porcentual de 0 a 100**. Aunque no se conoce el rango exacto de valores de resistividad eléctrica que puede tomar la fotorresistencia, podemos decir que en general se encuentran entre los valores de **0Ω a 1MΩ** [15] (dato consultado en varios *Datasheets* de fotorresistencias), siendo 0 la máxima luminosidad perceptible por el sensor y 1MΩ la completa oscuridad.

Para que el valor de voltaje a la salida del divisor corresponda con un nivel de **0V para oscuridad y 3.3V para la máxima luminosidad**, se recurre a una configuración del divisor con resistencia de **pull-down**. El valor óptimo de esta resistencia se ha hallado mediante **prueba** con un **potenciómetro** conectado como **resistencia variable**, en el **exterior un día de luz solar normal**, y el resultado nos da un valor cercano al comercial de **1kΩ**.

De este modo, el circuito correspondiente al módulo sensor lumínico conectado a la Intel Edison quedará como en la figura 22.

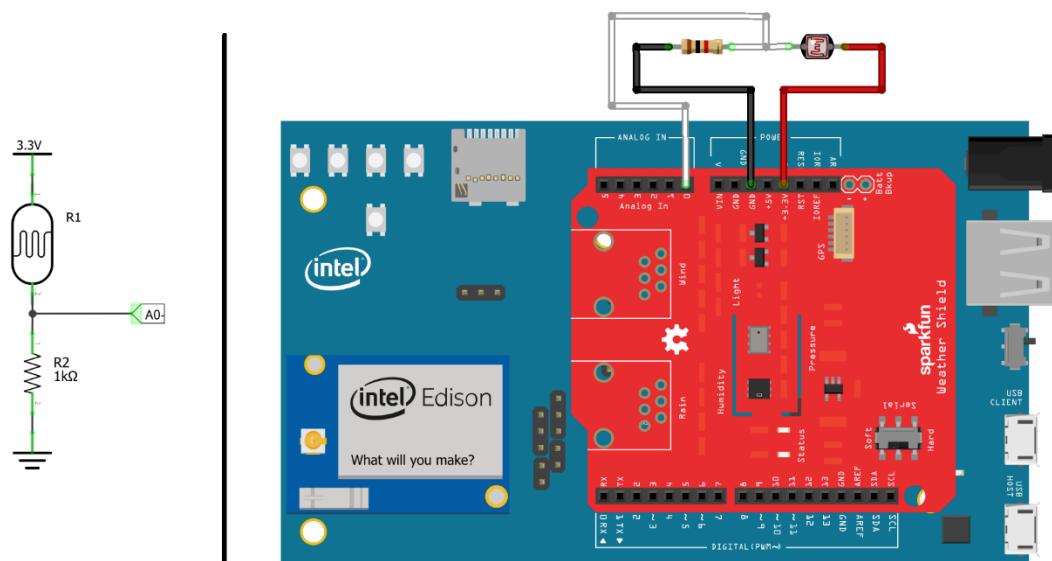


Fig. 22. Esquemático de conexión Intel Edison, Weather Shield y sensor de luz

Para conectar de forma cómoda y simple el circuito del sensor de luz, se recurre al **desarrollo de un pequeño módulo** basado en una placa perforada para prototipado de baquelita. Sobre esta placa se disponen la fotorresistencia, la resistencia de pull-down y **3 pines machos** para poder conectarnos a través de cables a las señales de **voltaje de entrada del divisor, voltaje de salida del divisor y voltaje de referencia (GND)** respectivamente. Una vez posicionados de forma adecuada, se procede con la soldadura de los componentes a través de la parte inferior de la placa (cara donde se encuentran las pequeñas superficies de cobre para facilitar la soldadura). El resultado final del módulo es el correspondiente a la figura 23.

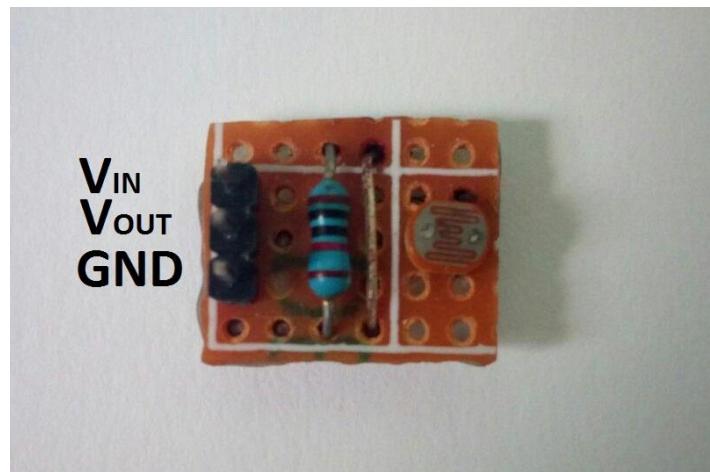


Fig. 23. Módulo sensor de luz

Una vez **creado el módulo**, y aprovechando la disponibilidad de la conexión al **pin A0** de la *Intel Edison*, que obtuvimos cuando flexionamos el pin correspondiente de la *Weather Shield*, **procedemos a conectarlo al sistema**, tal y como se muestra en la figura 24.

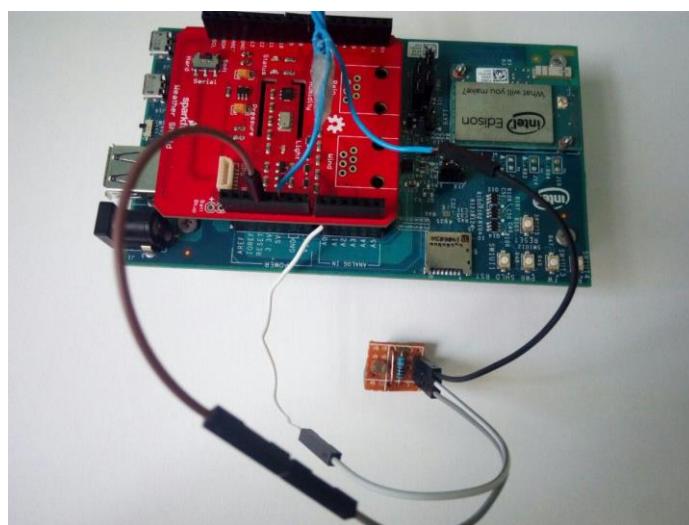


Fig. 24. Conexión *Intel Edison* y sensor de luz

Ya tenemos conectados e integrados todos los sensores necesarios por el sistema para determinar los parámetros ambientales, a continuación, procederemos con los elementos hardware que otorgan la posibilidad de interacción por parte del usuario, la **pantalla LCD** y los **pulsadores**.

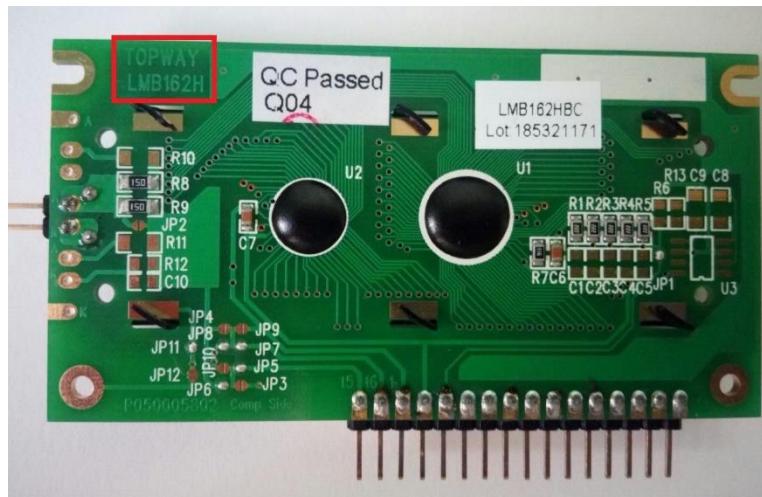


Fig. 25. Parte trasera del LCD

Como se dijo en el apartado de diseño, el modelo de la pantalla LCD que se va a utilizar es **LMB162HBC**, de **Topway**, determinado a partir de la parte posterior del módulo LCD, tal y como se ve en la figura 25, la cual presenta una interfaz digital paralela compuesta por 16 pines. De entre todas las señales de datos (BD), solo será necesario utilizar las 4 correspondientes a los bits más significativos, por tanto, las **conexiones** a realizar con esos 16 pines se reducen a **12**.

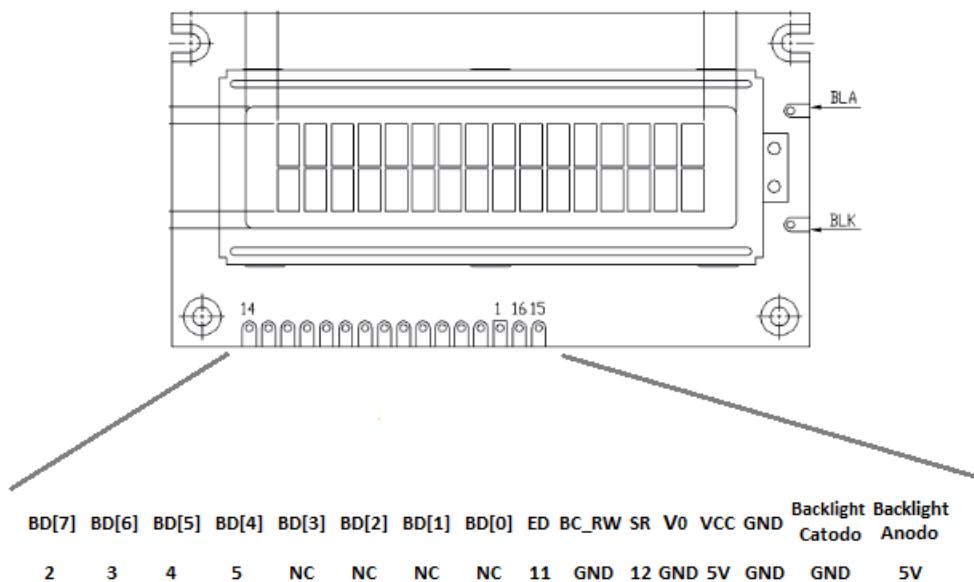


Fig. 26. Pinout del LCD

Como se verá en el siguiente apartado sobre la implementación del software de la Intel Edison, se utilizará una librería ya implementada para el control del LCD, por lo que, las conexiones a realizar vendrán dadas por dicha librería y deberán ser justo como se puede apreciar en la imagen anterior, la figura 26 (las siglas “NC” se corresponden a No Conectado).

Si al esquema que ya tenemos implementado sobre los elementos hardware que componen el sistema METEO, le sumamos las conexiones pertinentes al LCD, obtenemos la representación de la figura 27.

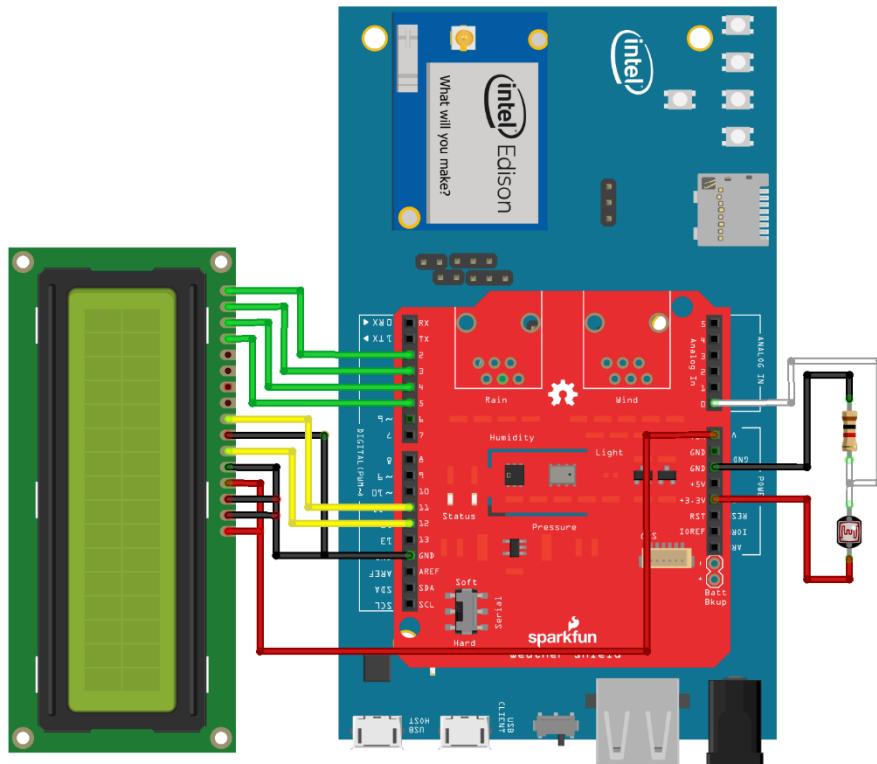


Fig. 27. Esquemático de conexión *Intel Edison*, *Weather Shield*, sensor de luz y *LCD*

Cabe destacar que, si retrocedemos al momento en que realizamos la conexión del *Weather Shield*, se puede apreciar que los pines digitales del 2 al 5 (que son utilizados para manejar el *LCD*) tienen función dentro de la *Weather Shield* (figura 17) no obstante, consultando el esquemático interno del *Shield* podemos observar que todos estos están asociados a los sensores opcionales (viento, lluvia y *GPS*) y **no generan ninguna incompatibilidad con el *LCD*** cuando no están presentes dichos sensores.

Por último, al igual que con el *fotorreceptor*, para los **pulsadores también crearemos un módulo** sobre una placa perforada. Cada pulsador será independiente e irá conectado a una entrada digital específica de la *Intel Edison*, las cuales presentan **resistencias internas de pull-up configurables**, por lo que **no necesitaremos de resistencias externas en el módulo** de los pulsadores. De esta forma, el circuito

correspondiente al módulo de los pulsadores quedaría tan simple como se muestra en la figura 28:

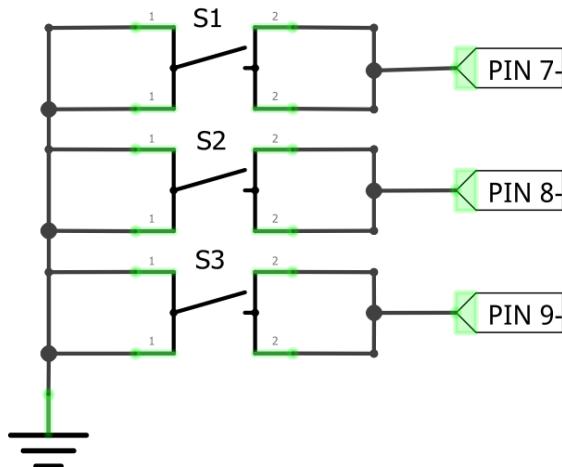


Fig. 28. Circuito eléctrico del módulo de los pulsadores

Como se observa, se ha representado cada uno de los **pines digitales de entrada de la Intel Edison**, que permitirán conocer el estado de los pulsadores, estos han sido elegidos según la **disponibilidad de pines restantes** tras la conexión del resto de elementos hardware; se han tomado los pines **7, 8 y 9** por comodidad a la hora de la conexión física y la disposición de los cables. Estas tres señales, junto al voltaje de referencia, conforman los pines físicos del módulo desarrollado, el cual se realizó siguiendo el mismo proceso que el módulo sensor de luz: posicionamiento de elementos, fijación de estos a la placa e interconexión mediante soldadura en la cara inferior de la placa. El resultado final del módulo se corresponde con la figura 29.

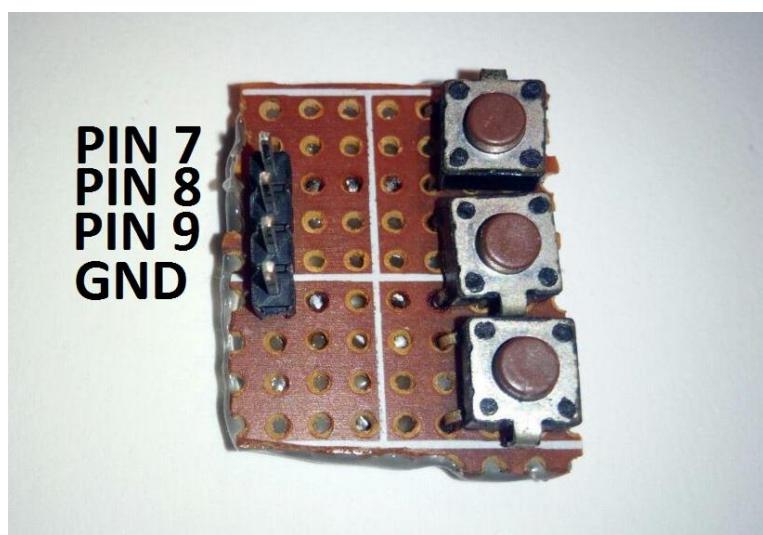


Fig. 29. Módulo de pulsadores

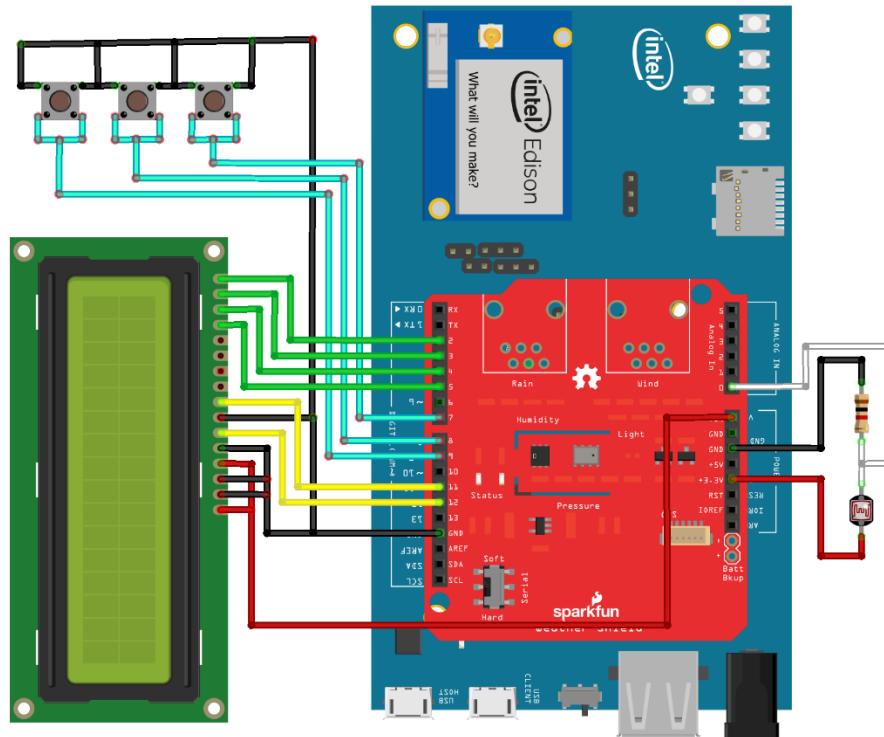


Fig. 30. Esquemático de conexión completo del sistema METEO

Como se puede ver en la figura 30, para finalizar con el aspecto hardware del sistema METEO, se conecta el módulo de los pulsadores. Así, el aspecto final del sistema real se muestra en la figura 31:

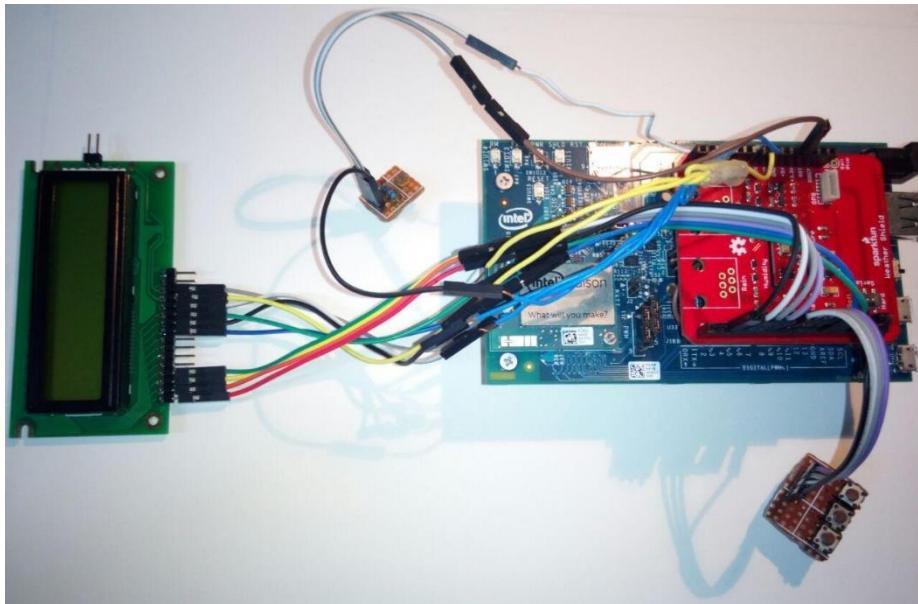


Fig. 31. Sistema METEO (hardware)

4.2 Software (Sistema METEO)

Una vez que tenemos completamente definido el hardware del sistema, se procede a desarrollar la “inteligencia” del mismo, es decir, el software que contendrá la Intel Edison. Comencemos por repasar los aspectos más característicos de esta placa (vistos en el apartado de diseño) en relación al Software:

- Se puede desarrollar aplicaciones mediante los siguientes lenguajes: C/C++, Python, JavaScript y Node.js.
- Se puede desarrollar aplicaciones a través del **Arduino IDE** y como si una placa Arduino se tratase (con la **simplicidad y versatilidad del código Arduino**), o mediante **otros IDEs**, permitiendo depuración y aspectos más avanzados del mismo (como el **Intel System Studio IoT Edition**, para C/C++/JAVA; o **Intel XDK**, para JavaScript/Node.js).
- En su memoria interna (eMMC) alberga un sistema *Linux* basado en *Yocto*.

En un principio, de entre las posibles opciones de desarrollo, nos interesamos por el uso de algún entorno que nos permitiera depurar el sistema, no obstante, usar el entorno **Arduino ofrece la posibilidad de utilizar librerías ya implementadas** para controlar tanto la *Weather Shield* como el LCD. Esto nos hizo decidirnos **inicialmente por el entorno Arduino**, sin embargo, posteriormente llegamos al punto en el que **necesitábamos tener un mayor control sobre el sistema**, por lo que tuvimos que **cambiar al entorno Intel System Studio IoT Edition** (en el cual se utiliza **C/C++**, lenguaje que nos es más cómodo que JavaScript/Node.js).

El entorno *Intel System Studio IoT Edition* está basado en el IDE Eclipse, está pensado para desarrollar en C++ o Java, y posee la capacidad de integrar fácilmente las librerías de Intel: **MRAA**, “librería Linux para comunicación E/S a baja velocidad con soporte para una gran variedad de dispositivos y lenguajes de programación” [16], y **UPM**, “repositorio de alto nivel que provee controladores software para una gran variedad de sensores y actuadores conocidos permitiendo interaccionar a bajo nivel con ellos mediante las APIs de MRAA” [17], **para controlar sensores**; además presenta numerosas plantillas y herramientas para iniciar nuevos proyectos de forma rápida.

Por tanto, el **desarrollo** de aplicaciones en la **Intel Edison** mediante el entorno **Intel System Studio IoT Edition**, consiste en **programación C/C++** haciendo uso de la **librería MRAA**, la cual supone una **capa de abstracción hardware**, y de las **librerías contenidas en el repositorio UPM**, para **comunicarnos con los sensores**.

Como se verá más adelante, la **librería** localizada en el repositorio **UPM** para el control del sensor de humedad/temperatura **HTU21D** localizado en la *Weather Shield* no funciona de forma adecuada y **da problemas con la comunicación I2C**, por lo que acabaremos incluyendo la propia capa de abstracción hardware de *Arduino* en el entorno *Intel System Studio IoT Edition* de modo que obtendremos los beneficios de esta, junto a la potencia de este entorno, así como la capacidad de depuración.

4.2.1 Preparando el Sistema Intel Edison

Para comenzar, necesitamos preparar a la placa Intel Edison, para ello se procede a acceder a la página principal de la zona de desarrolladores de Intel Edison (<https://software.intel.com/en-us/iot/hardware/edison>) y accedemos a la sección de “Get Started”, tal como se muestra en la figura 32.

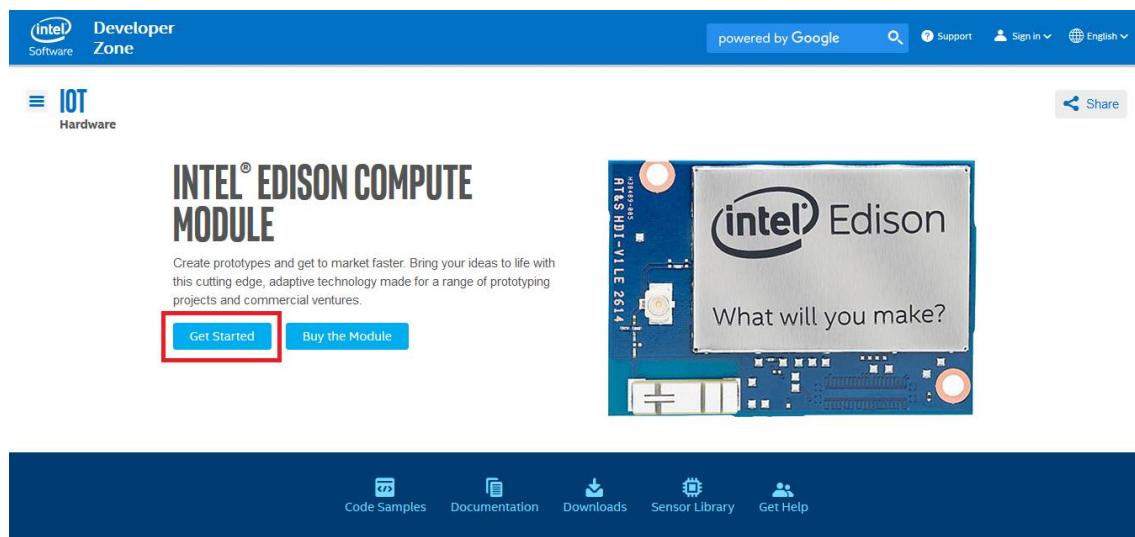


Fig. 32. Zona de Desarrolladores de Intel Edison

Siguiendo todos los pasos de esta sección, se revisan los componentes que vienen en el kit, se realiza el ensamblaje del módulo de computación Intel Edison en la placa de desarrollo *Arduino Breakout Board* y se conecta la placa en el ordenador. El sistema operativo del ordenador utilizado para desarrollar el software es Windows 10 y tal como se indica en la sección en la que nos encontramos, la conexión de la placa con el ordenador debe seguir el siguiente proceso (representado de forma gráfica en la figura 33):

1. Posicionar el micro-switch de la placa en la dirección más cercana al micro-USB (con esto se consigue que el micro USB se encuentre en modo de dispositivo, en otro caso estaría en modo Host).
2. Conectamos el cable USB del ordenador al conector micr-USB más cercano al micro-switch (lo que le dará alimentación a la placa).

3. Conectamos el segundo cable USB para la comunicación por puerto serie (puerto COM).
4. La Intel Edison puede ser alimentada exclusivamente por el USB, sin embargo, en caso de necesitar más corriente si conectaría ahora la fuente dc. En nuestro caso la placa tendrá un bajo consumo y no requerirá la fuente dc, se alimentará solo por USB.

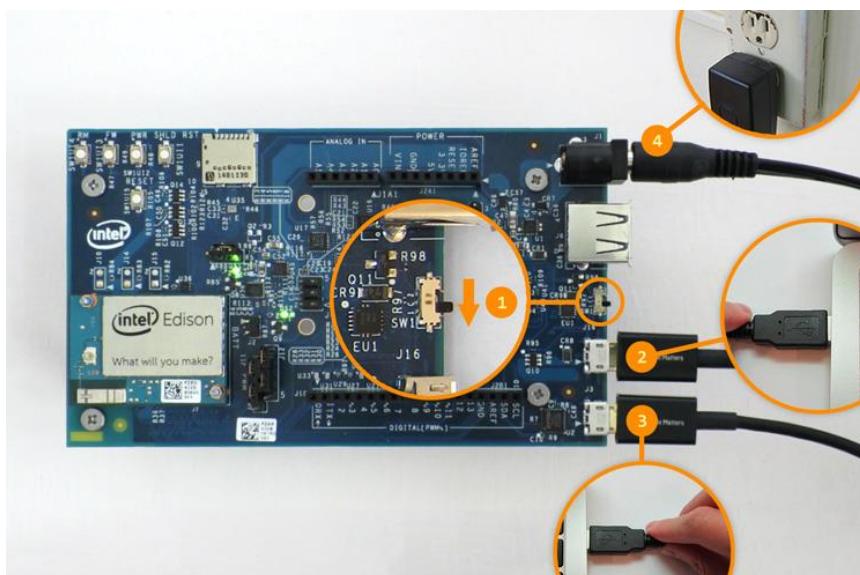


Fig. 33. Conexión del Sistema Edison al ordenador

El siguiente paso consiste en usar la **herramienta de configuración** que proporciona Intel para preparar tanto nuestro sistema, como la Intel Edison. Esta herramienta simplifica y facilita en gran medida esto, ya que permite **instalar los Drivers** adecuados según el sistema utilizado, **actualizar el firmware**, **habilitar el servidor SSH** (Secure Shell) y **establecer la conexión wifi** de la placa.

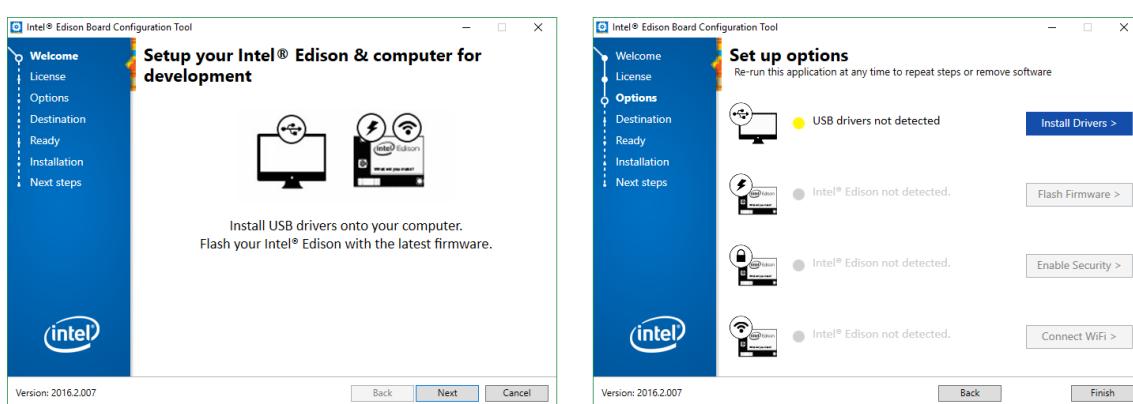


Fig. 34. Herramienta de configuración de Intel Edison

Procedemos a descargar la herramienta de configuración, cuyo enlace hacia la descarga se puede encontrar en el segundo paso de la sección “Get Started” en la que nos encontrábamos (<https://software.intel.com/iot/hardware/edison/downloads>). Una vez descargada la ejecutamos y podemos apreciar que se detecta la ausencia, en el sistema, de los Drivers (figura 34). A continuación, presionamos en el botón para instalar los Drivers y esperamos a que se complete el proceso, como se puede observar en la figura 35.

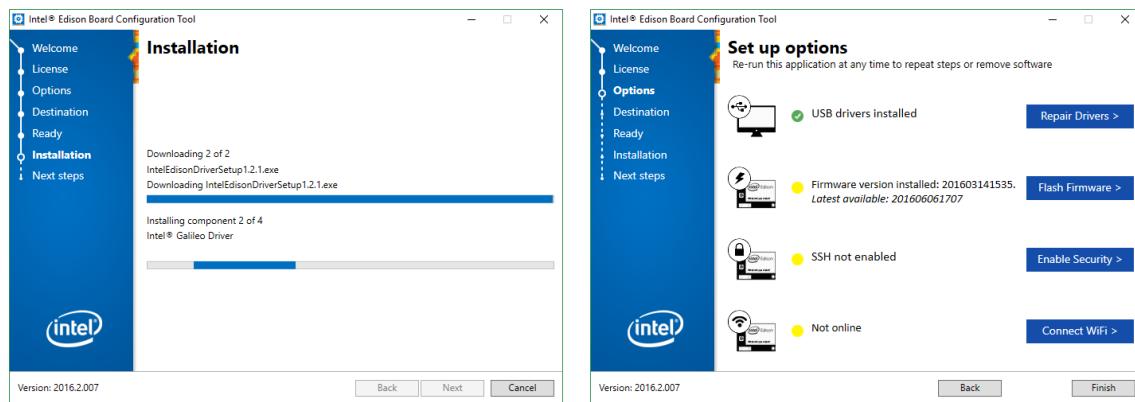


Fig. 35. Instalación de Drivers

Con los Drivers instalados, el sistema es capaz de comunicarse con la placa y detectar tanto la versión del Firmware interno, como el estado de las conexiones (servicio SSH y wifi). Como se puede ver, el Firmware que trae la placa no es el último, este se puede actualizar, sin embargo, la herramienta de configuración (al menos en esta versión, la 2016.2.007) falla al intentar actualizarlo, por lo que tenemos que proceder manualmente tal y como indica Intel (<https://software.intel.com/es-es/flashing-firmware-on-your-intel-edison-board-windows>), por lo que cerramos la herramienta de configuración. Como se especifica, debemos seguir los siguientes pasos:

1. Descargar la última **imagen de Linux Yocto**: <https://software.intel.com/en-us/iot/hardware/edison/downloads>
2. Descargar las últimas versiones de los archivos **dfu-util.exe** y **libusb-1.0.dll**: <http://dfu-util.sourceforge.net/releases/>
3. Descomprimir la imagen Yocto en una carpeta vacía e insertar los archivos dfu-util.exe y libusb-1.0.dll en ella.
4. Abrimos un terminal de Windows con privilegios de administrador en la ruta correspondiente a la imagen descomprimida (o accedemos a dicha ruta) y ejecutamos el comando: **flashall.bat**
5. Esperamos a que se complete el proceso y se reinicie la placa, este paso está representado en la figura 36. Todo el proceso puede durar hasta 10 minutos.



```

Administrator: C:\WINDOWS\system32\cmd.exe - flashall.bat
C:\Users\PCP2023\Downloads\iot-devkit-prof-dev-image-edison-20160606>flashall.bat
Using U-boot target: edison-blankndis
Now waiting for dfu device 8087:0999
Please plug and reboot the board
Dfu device found
Flashing IFWI
Download [=====] 100% 4194304 bytes
Download done.
Download [=====] 100% 4194304 bytes
Download done.
Flashing U-Boot
Download [=====] 100% 237568 bytes
Download done.
Flashing U-Boot Environment
Download [=====] 100% 65536 bytes
Download done.
Flashing U-Boot Environment Backup
Download [=====] 100% 65536 bytes
Download done.
Rebooting to apply partiton changes

Dfu device found
Flashing boot partition (kernel)
Download [=====] 100% 6144000 bytes
Download done.
Flashing rootfs, (it can take up to 5 minutes... Please be patient)
Download [=====] 100% 1373149184 bytes
Download done.
Rebooting
U-boot & Kernel System Flash Success...
Your board needs to reboot to complete the flashing procedure, please do not unplug it for 2 minutes.

C:\Users\PCP2023\Downloads\iot-devkit-prof-dev-image-edison-20160606>

```

Fig. 36. Actualización de Firmware

Volvemos a abrir la herramienta de configuración de Intel Edison y comprobamos la nueva versión del firmware. En la figura 37 se puede ver como la placa se encuentra ahora en la última versión (hasta la fecha).

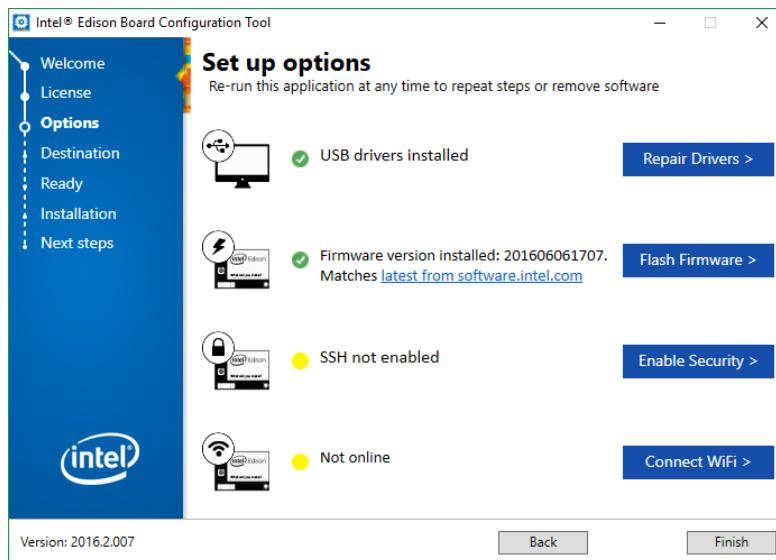


Fig. 37. Herramienta de configuración. Firmware actualizado

Por último, configuraremos el servicio SSH, el cual será necesario para cargar los programas desde el *Intel System Studio IoT Edition* y la conexión wifi, proceso representado en la figura 38. Al seleccionar la opción del SSH le damos un nombre (por ejemplo: intel_edison) y una contraseña (por ejemplo: intel1234) y al configurar el wifi seleccionamos la red a la que nos queremos conectar mediante el SSID de nuestro punto de acceso y facilitamos la contraseña. Una vez configurada la red wifi, se establecerá la conexión y se le asociará una IP (proporcionada por el servidor DHCP del punto de acceso) a la Intel Edison.

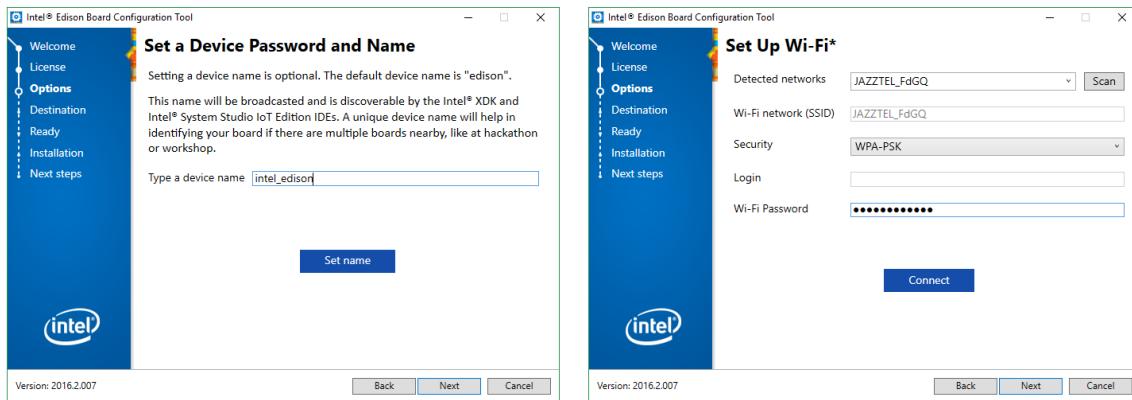


Fig. 38. Herramienta de configuración. Configuración SSH y Wifi

Tal y como se puede apreciar en la figura 39, tras todo este proceso, tendríamos completamente configurada la Intel Edison, y podríamos cerrar la herramienta de configuración.

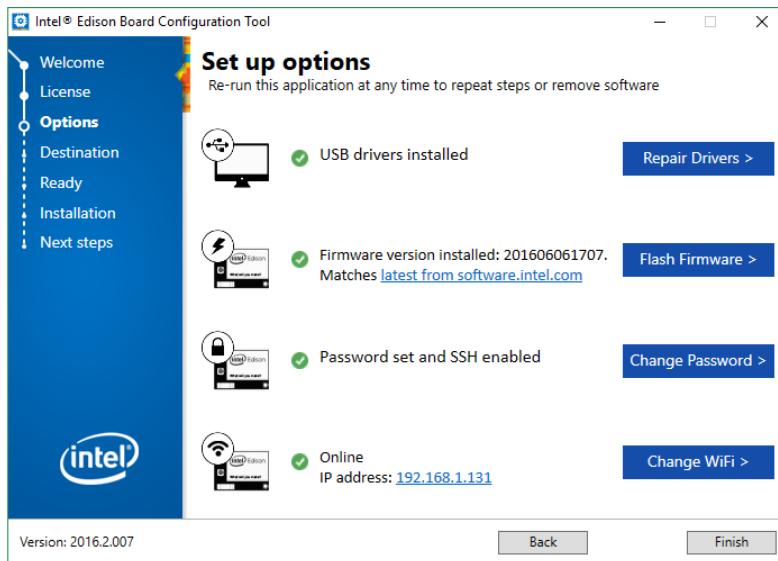


Fig. 39. Herramienta de configuración. Configuración completada

El siguiente paso corresponde a la sección 3 del “Get Started” de la Intel Edison y consiste en seleccionar un entorno de desarrollo y descargarlo. Como se ha dicho, seleccionaremos el entorno *Intel System Studio IoT Edition* (<https://software.intel.com/en-us/iot/tools-ide/ide>) y procederemos a su descarga. En la figura 40 se puede ver la web de entornos de desarrollo empotrado de *Intel*.



Intel Software Developer Zone

powered by Google Support Sign in English

IOT Tools & IDEs Share

INTEGRATED DEVELOPMENT ENVIRONMENTS

The Intel® IoT Developer Kit is programmable using Arduino®, C/C++, JavaScript®, Node.js®, and Python*. Explore the list below to find the best solution for you.

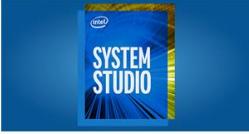
 Intel® XDK	 Arduino®	 Intel® System Studio IoT Edition
Create web interfaces, add sensors to your project, and work with the cloud. This developer kit includes companion templates to get your project up and running quickly.	With readily available code from a variety of manufacturers, quickly add sensors to your project with this intuitive interface.	This Eclipse*-based IDE comes with the built-in capability to easily integrate sensors via UPM and MRAA libraries, which you can develop in C/C++ or Java.

Fig. 40. Descarga de entorno de desarrollo para Intel Edison

El IDE *Intel System Studio IoT Edition* es un programa portable y como tal, no necesita instalación, sin embargo, al estar basado en Eclipse (el cual está desarrollado en JAVA) requiere que el sistema tenga instalado JAVA para poder ejecutarse. De este modo, vamos a descargar el kit de desarrollo Java SE Development Kit (JDK), el cual también incluye el entorno Java SE Runtime Environment (JRE), para ello, accedemos a <http://www.oracle.com/technetwork/java/javase/downloads/index.html> y descargamos el JDK apropiado para nuestra versión del sistema, en este caso para Windows de 64 bits, como se muestra en la figura 41.

Java SE Development Kit 8...

Java SE Development Kit 8u102

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	160.35 MB	jdk-8u102-linux-i586.rpm
Linux x86	175.03 MB	jdk-8u102-linux-i586.tar.gz
Linux x64	158.35 MB	jdk-8u102-linux-x64.rpm
Linux x64	173.03 MB	jdk-8u102-linux-x64.tar.gz
Mac OS X	227.35 MB	jdk-8u102-macosx-x64.dmg
Solaris SPARC 64-bit	139.59 MB	jdk-8u102-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	98.98 MB	jdk-8u102-solaris-sparcv9.tar.gz
Solaris x64	140.02 MB	jdk-8u102-solaris-x64.tar.Z
Solaris x64	96.24 MB	jdk-8u102-solaris-x64.tar.gz
Windows x86	189.2 MB	jdk-8u102-windows-i586.exe
Windows x64	194.68 MB	jdk-8u102-windows-x64.exe

Java SE Development Kit 8u101 Demos and Samples Downloads

You must accept the Oracle BSD License. to download this software.

Fig. 41. Descarga del Java JDK



Una vez descargado lo instalamos, y así ya podremos ejecutar el *Intel System Studio IoT Edition*. La ejecución da como resultado la figura 42, donde se observa que se ha detectado JAVA en el sistema, tras esta ventana, se nos abre el entorno de desarrollo.

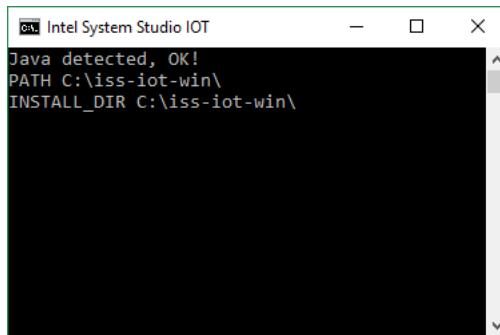


Fig. 42. Inicio correcto de *Intel System Studio* y detección de Java

4.2.2 Preparando el Proyecto de Intel System Studio

En este sub-apartado detallaremos el proceso de **creación desde cero de un proyecto** en el entorno de desarrollo *Intel System Studio IoT Edition* y la preparación de este para poder comenzar a desarrollar nuestra aplicación *METEO* (**importar el núcleo Arduino, incluir las librerías a utilizar, etc.**).

Comenzamos por el proceso de creación de un nuevo proyecto:

1. Abrimos el Intel System Studio IoT Edition.
2. Creamos un nuevo proyecto mediante: *File/New/Create a new Intel(R) IoT Project* (figura 43):

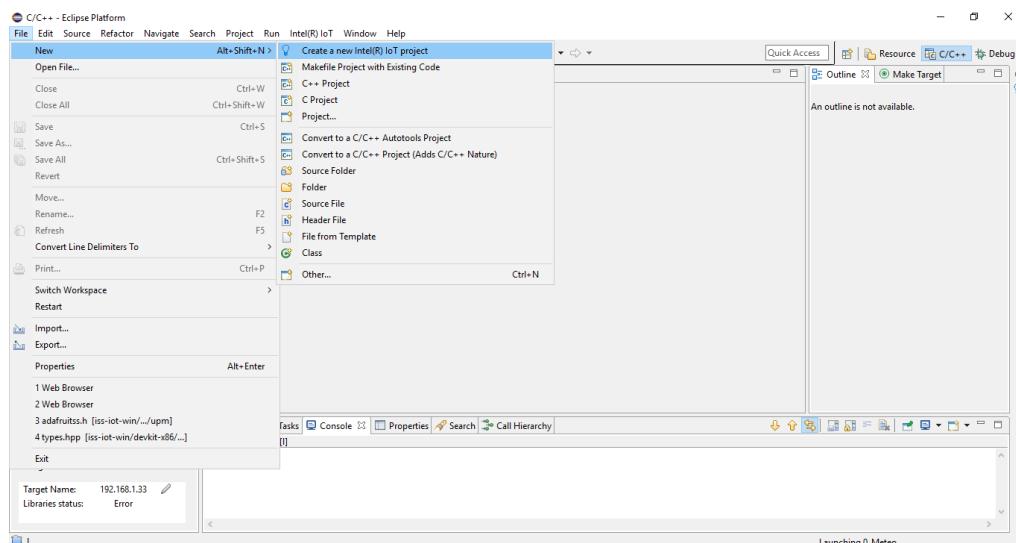


Fig. 43. Creación de proyecto *Intel System Studio*. Crear proyecto

3. Seleccionamos “C/C++ Project” y pulsamos en *Next* (figura 44):

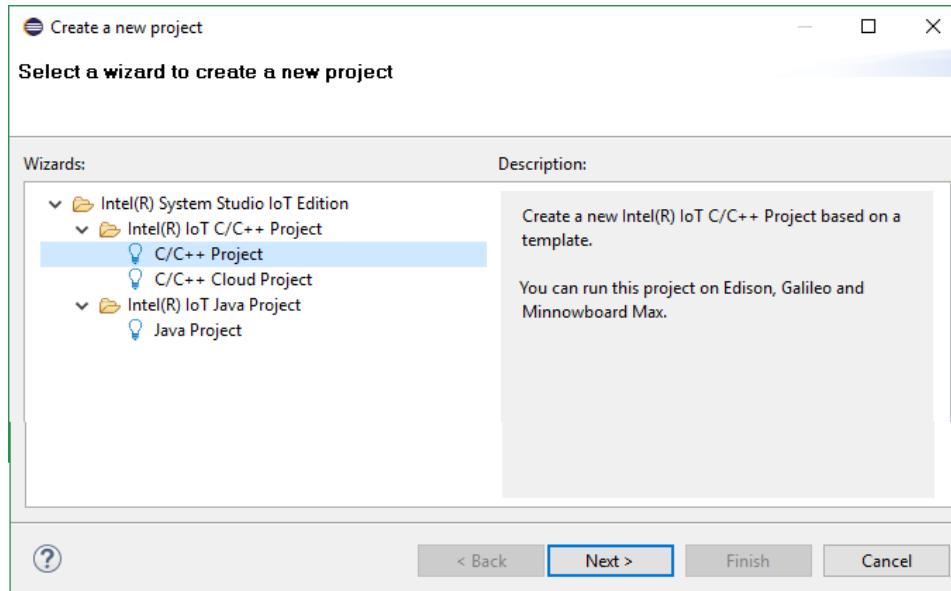


Fig. 44. Creación de proyecto *Intel System Studio*. Tipo de proyecto

4. Lo siguiente que hay que hacer se corresponde con la figura 45, en ella, establecemos el nombre del proyecto (por ejemplo, PlataformaMeteo), seleccionamos un proyecto de base (plantilla) “On Board Led Blink C++”, para que se importen las librerías MRAA y UPM, y pulsamos en *Next*:

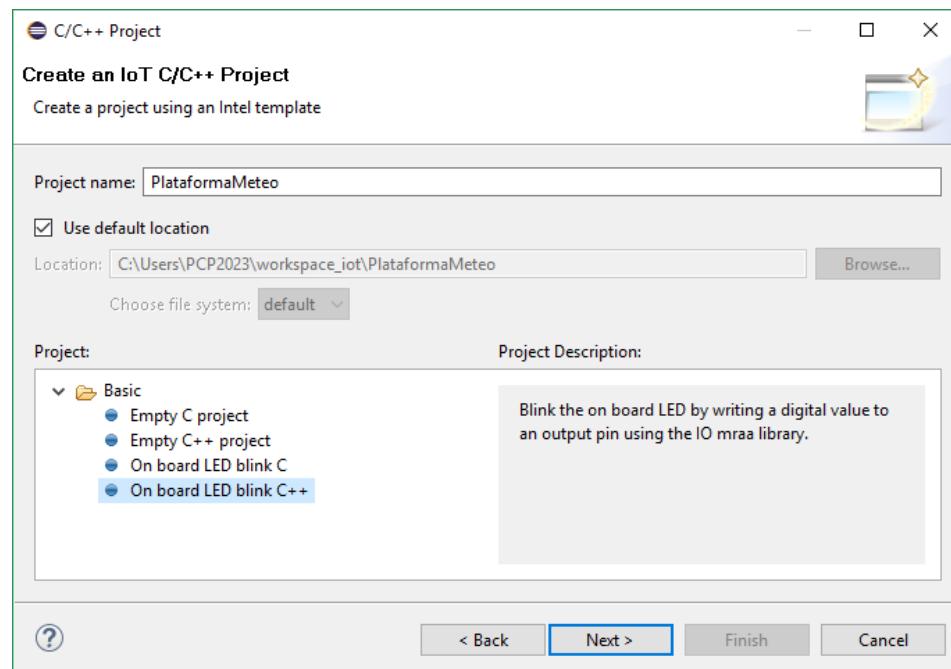


Fig. 45. Creación de proyecto *Intel System Studio*. Nombre del proyecto



5. Buscamos nuestra placa presionando el botón "Search Target", a continuación, le damos un nombre de conexión (por ejemplo, IntelEdison) y añadimos la dirección IP correspondiente a la placa (la cual aparecía en la herramienta de configuración de Intel Edison), todo esto tal y como se muestra en la figura 46. Tras ello, presionamos en *Finish*.

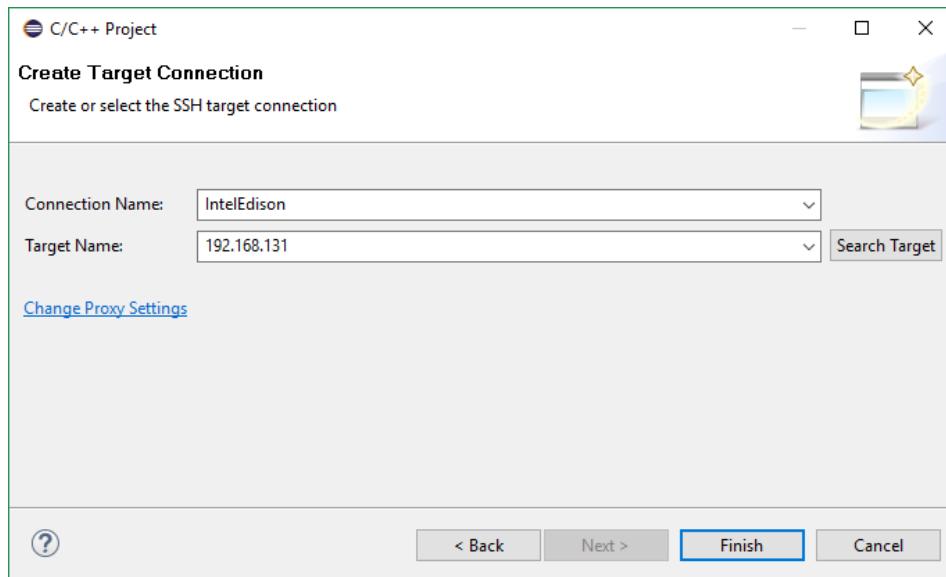


Fig. 46. Creación de proyecto Intel System Studio. Configurando conexión con placa

Hecho lo anterior, habremos creado el proyecto base y comprobaremos el correcto funcionamiento del proceso de compilación y carga del código dado. Para ello, presionamos el botón correspondiente a lanzar la carga del código (figura 47).

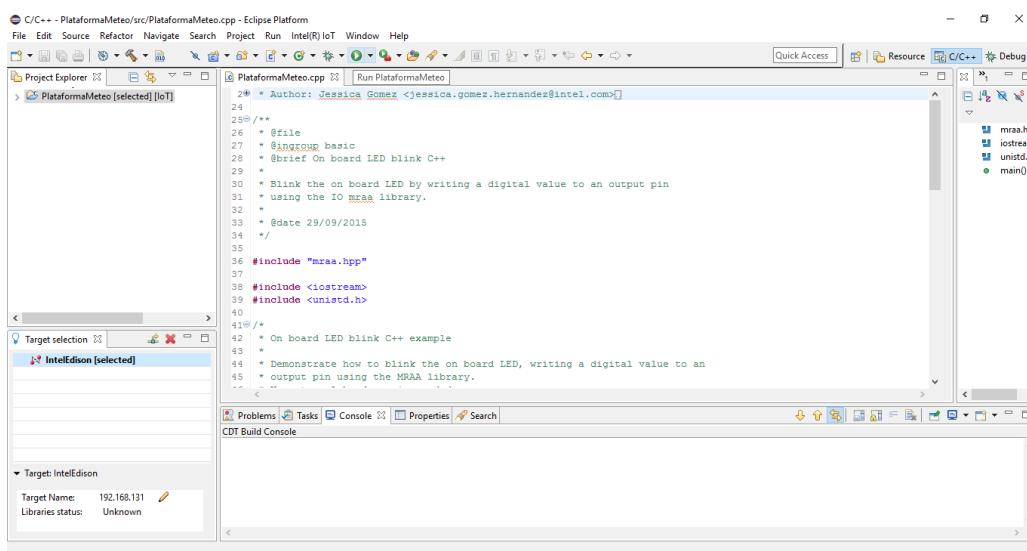


Fig. 47. Intel System Studio. Proyecto base y carga del código en el dispositivo

Al presionar este botón, se intentará conectar con el dispositivo correspondiente a la configuración de conexión que dimos previamente y se nos pedirá los datos de conexión del servicio SSH. Así, como se ve en la figura 48, introducimos la contraseña SSH que establecimos mediante la herramienta de configuración de la Intel Edison, en este caso, intel1234 y presionamos OK.

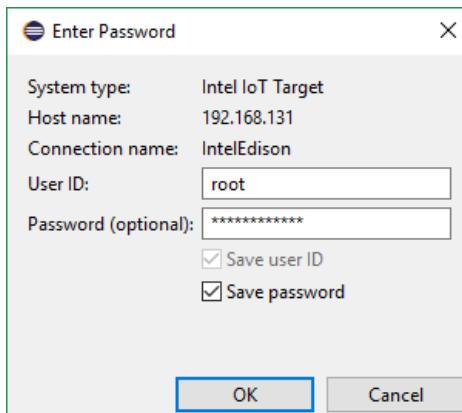


Fig. 48. Intel System Studio. Conexión con el dispositivo

Tras eso, esperamos a que se cargue en la placa y comprobamos el correcto parpadeo del LED.

Para hacer un uso rápido y sencillo tanto del *Weather Shield* como del *LCD*, existen varias librerías implementadas para *Arduino* pero, para poder utilizarlas en nuestro proyecto, debemos de **insertar el "Core" o "Núcleo" de Arduino**, es decir todos aquellos archivos que componen la capa de abstracción de *Arduino* para la Intel Edison. Para este proceso se requiere de tener instalado en el equipo, el entorno de programación *Arduino IDE 1.6.5 o superior* y el paquete de placas correspondientes a la Intel Edison (Instalado mediante el gestor de tarjetas). Una vez con esto, seguimos el siguiente proceso:

1. En Windows, creamos una carpeta nueva de nombre *Arduino_core*
2. Copiamos dentro de esta carpeta los archivos localizados en:
 - C:/Users/USUARIOPC/AppData/Local/Arduino15/packages/Intel/hardware/686/VERSION/cores/Arduino
 - C:/Users/USUARIOPC/AppData/Local/Arduino15/packages/Intel/hardware/686/VERSION/variants/edison_fab_c
 - C:/Users/USUARIOPC/AppData/Local/Arduino15/packages/Intel/hardware/686/VERSION/libraries/Wire/src

3. En *Intel System Studio IoT Edition*, importamos en nuestro proyecto, como sistema de archivo, la carpeta creada: *File/Import*.
4. Marcamos *File System* y presionamos *Next* (figura 49).

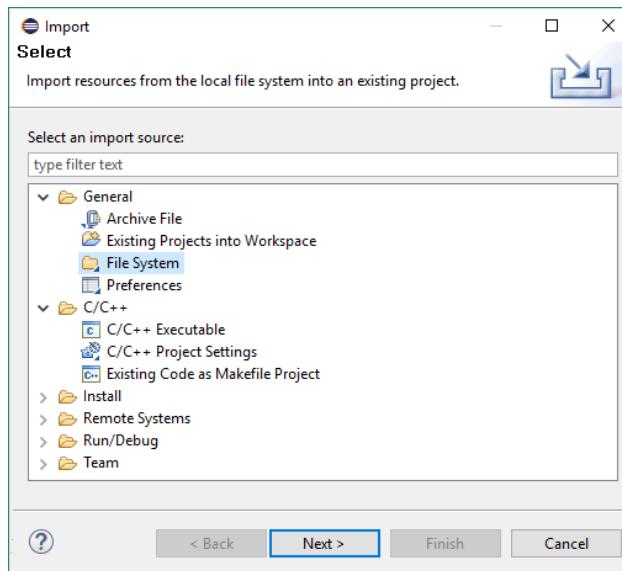


Fig. 49. *Intel System Studio*. Importar sistema de archivos

5. Seleccionamos la carpeta “Arduino_core” creada y presionamos en *Finish*, como se muestra en la figura 50.

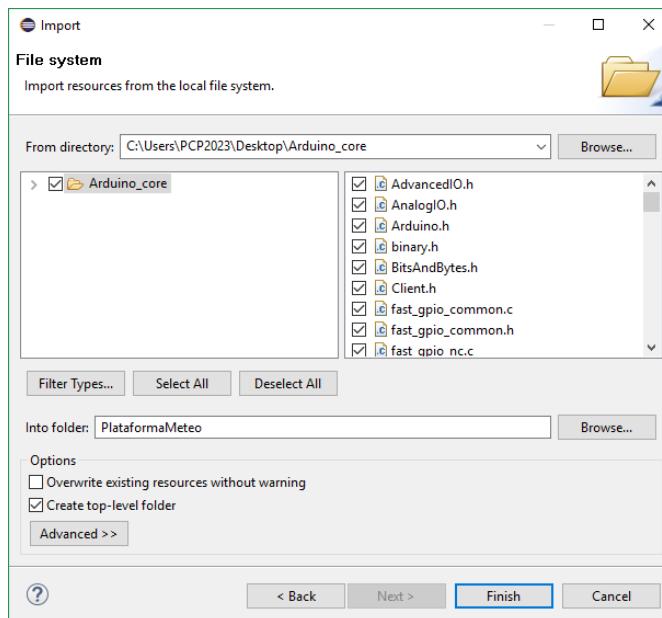


Fig. 50. *Intel System Studio*. Importando el núcleo Arduino

- A continuación, accedemos a las opciones del proyecto desde la nueva subcarpeta Arduino_core dentro del proyecto (figura 51).

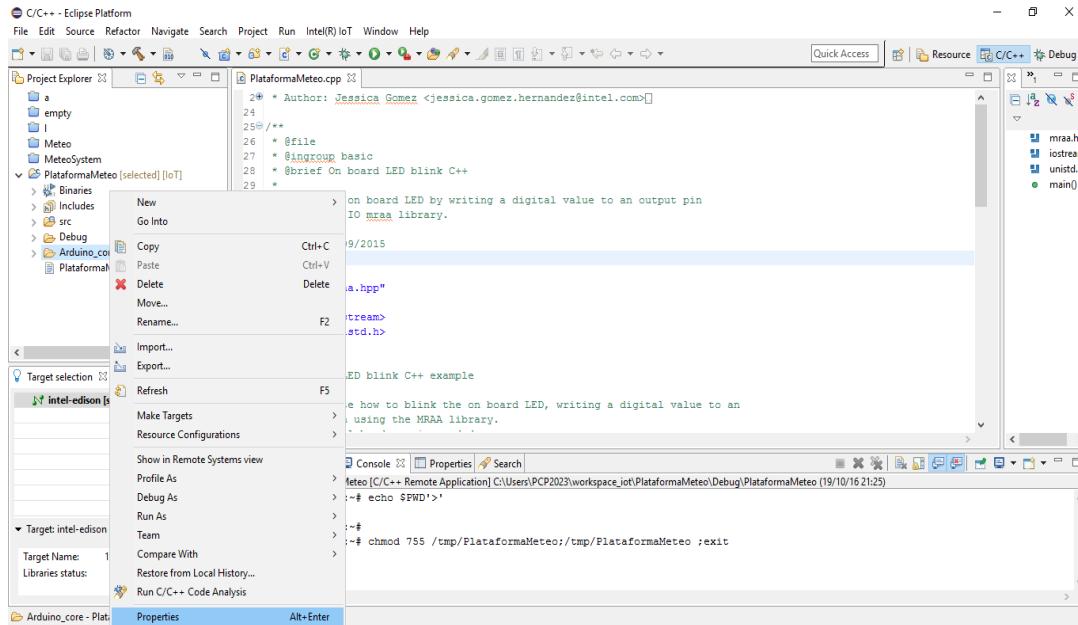


Fig. 51. Intel System Studio. Opciones de proyecto en sub-carpeta

- Dentro de C/C++ Build desmarcamos el check de la opción "exclude resource from build" (para que se incluya en la compilación dicha carpeta) y presionamos Apply, tal y como se puede observar en la figura 52.

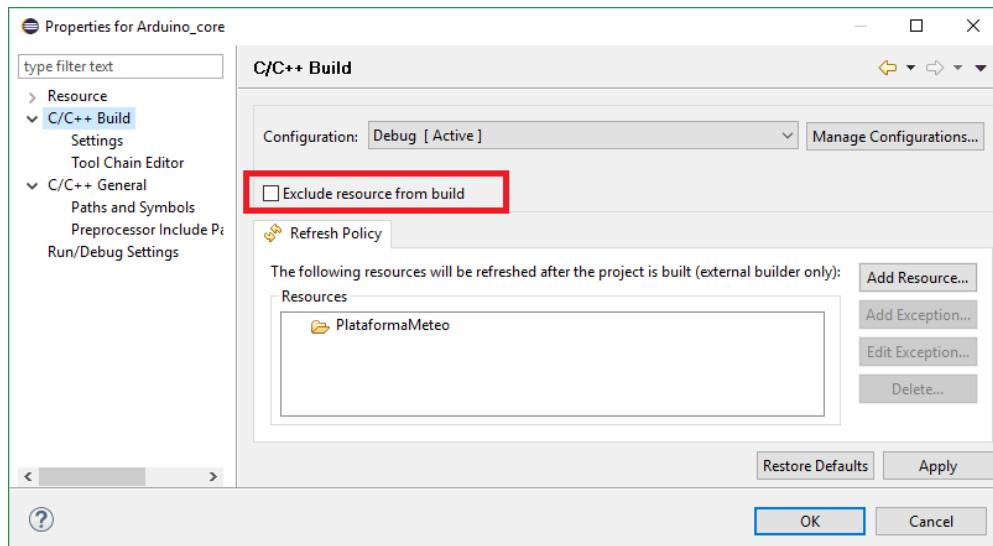


Fig. 52. Intel System Studio. Incluir en la compilación

- Dentro de *C/C++ General/Paths and symbols* quitamos las referencias a las librerías MRAA y UPM en los lenguajes de C y C++ seleccionándolas y pulsando *Delete*, esto es necesario para evitar problemas de incompatibilidades. En la figura 53 se representa este paso.

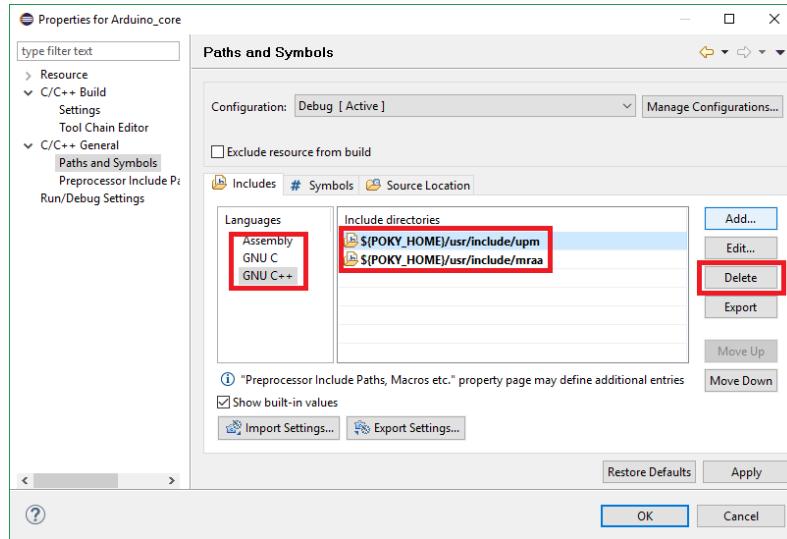


Fig. 53. Intel System Studio. Eliminando librerías MRAA y UPM

- Ahora, seleccionamos el lenguaje GNU C++, presionamos el botón *Add* y añadimos para todas las configuraciones y todos los lenguajes la propia carpeta Arduino_Core, buscándola en el workspace; hecho esto presionamos *Apply* y cerramos las opciones. En la figura 54 se muestra la ventana para añadir las rutas de directorios.

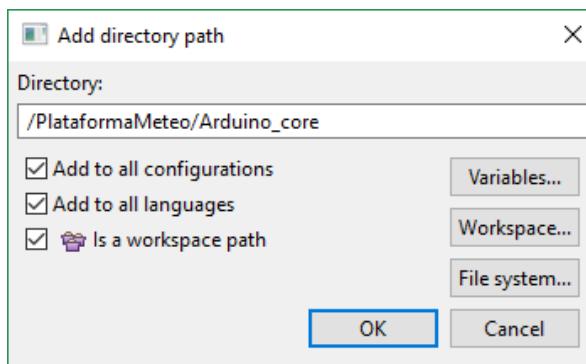


Fig. 54. Intel System Studio. Añadiendo la propia carpeta Arduino_core

- Por último, accedemos a las opciones del proyecto completo: *Project/Properties*, e incluimos el Núcleo Arduino tal y como se ha hecho en el paso anterior.

Con lo anterior, tendríamos el núcleo de *Arduino* en nuestro proyecto listo para ser utilizado. Lo siguiente que vamos a hacer es **importar las librerías Arduino** a utilizar en el proyecto. Estas serían la del **sensor de humedad HTU21D** (el cual da problemas si se utiliza la librería UPM) y la del **LCD**. Para ello seguimos los siguientes pasos:

1. Ejecutamos el *Arduino IDE* y abrimos el gestor de librerías: *Programa/Incluir Librería/Gestionar Librerías* (figura 55).

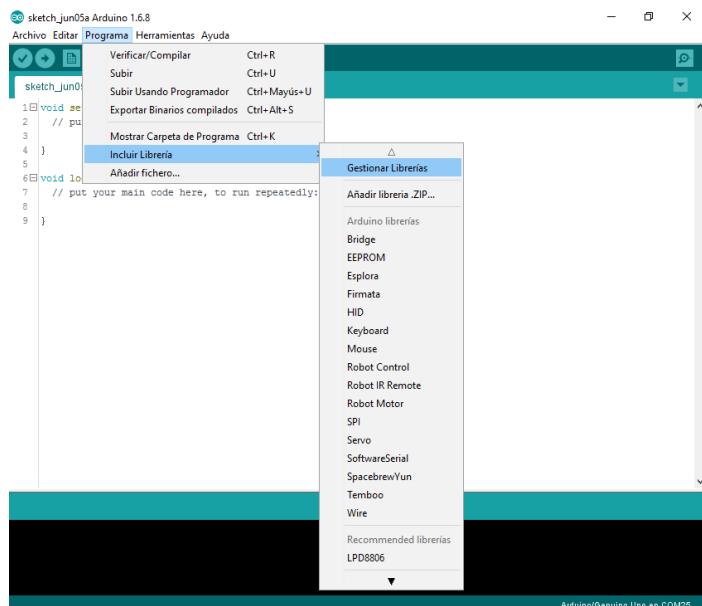


Fig. 55. Arduino IDE. Abriendo el gestor de librerías

2. Buscamos la librería de control del sensor *HTU21D* desarrollada por *Sparkfun*, y la instalamos, buscando por *Sparkfun*, como puede verse en la siguiente imagen, la figura 56.

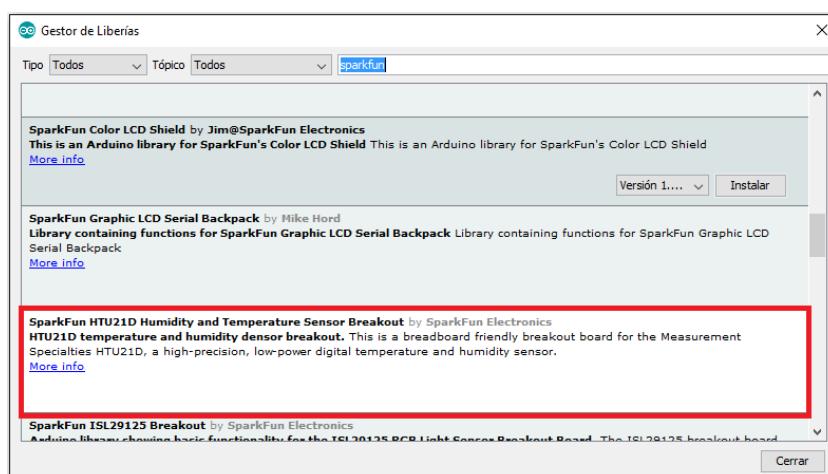


Fig. 56. Arduino IDE. Gestor de librerías

3. Una vez instalada, cerramos el gestor de librerías y el entorno Arduino IDE.
4. Nos aseguramos que se encuentra esta nueva librería “**SparkFun_HTU21D_Humidity_and_Temperature_Sensor_Breakout**” en el directorio de librerías descargadas por el entorno *Arduino*:
 - C:\Users\PCP2023\Documents\Arduino\libraries
5. La librería LCD a utilizar es la librería “**LiquidCrystal**” la cual viene por defecto en el entorno IDE. Nos aseguramos de localizarla en el directorio de librerías por defecto del Arduino IDE:
 - C:\Program Files (x86)\Arduino\libraries

Con estas **dos librerías** localizadas en el equipo, procedemos a **importarlas** en *Intel System Studio IoT Edition*. Para ello, volvemos a realizar los pasos realizados al importar el núcleo *Arduino* pero, esta vez, para estas dos librerías (importar como sistema de archivos, incluir su compilación y referenciar tanto a sí mismo como al Núcleo de *Arduino*).

Así, las opciones del proyecto completo y de cada librería deben quedar tal y como se ve en la siguiente figura número 57:

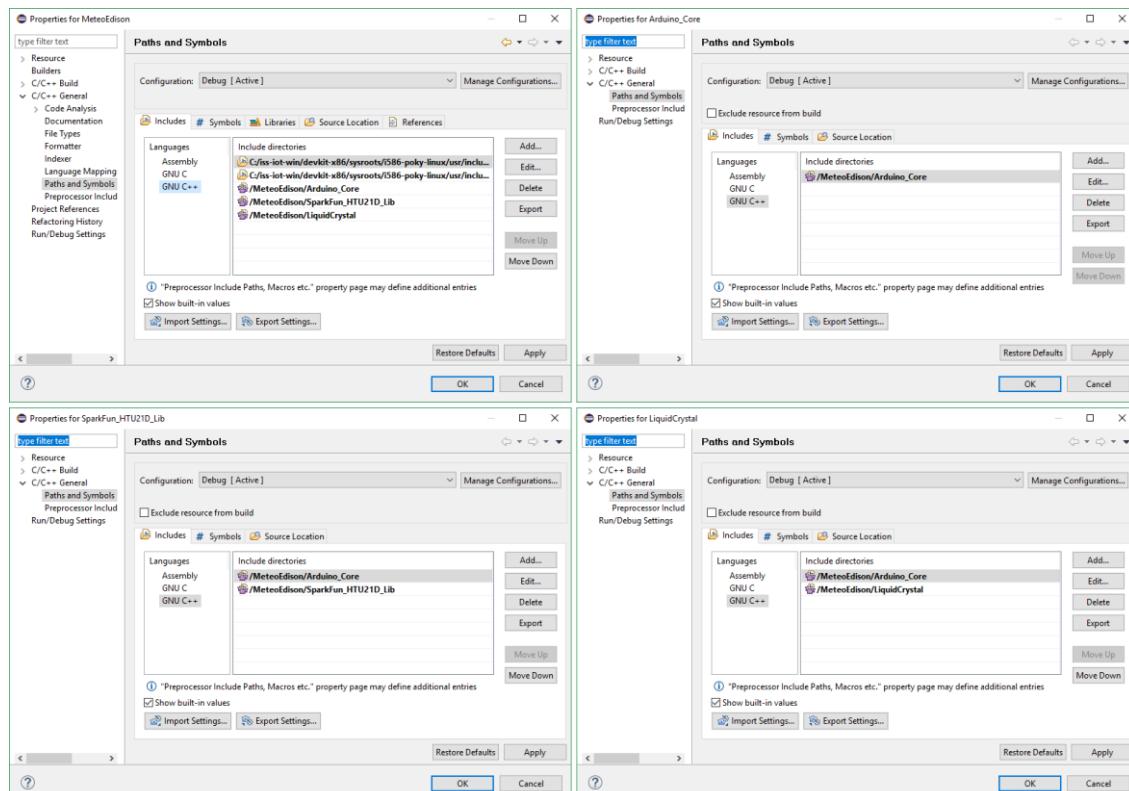


Fig. 57. Opciones del proyecto completo y de cada sub-directorio

Para finalizar, se deben **referenciar las librerías y el núcleo Arduino** en la carpeta del proyecto que contendrá los archivos del programa que realicemos, la carpeta “**src**”. En la figura 58 se puede observar cómo han de quedar las referencias en esa carpeta.

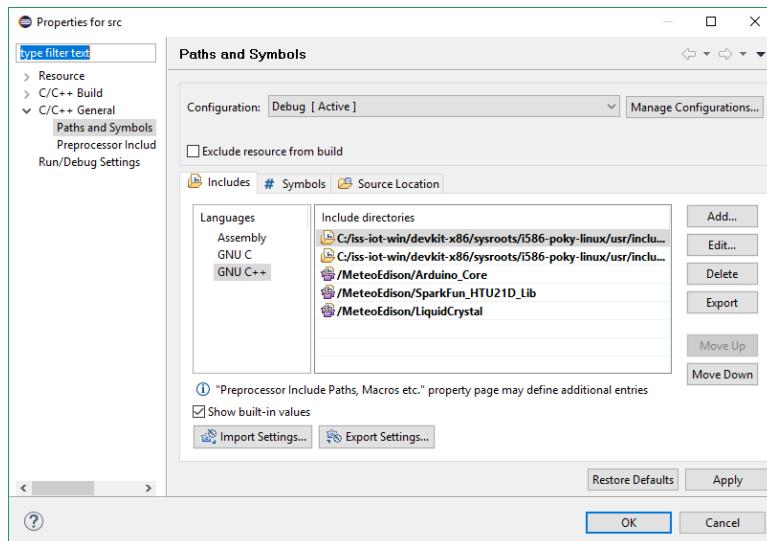


Fig. 58. Referencias del directorio *src*

El *núcleo Arduino* utiliza las librerías *math* y *pthread* y estas deben de referenciarse en nuestro proyecto, para ello:

1. Accedemos a las opciones del proyecto completo: *Project/Properties*.
2. En la sección *C/C++ General/Paths and symbols*, en la pestaña *Libraries* insertamos los *flags* del *linker* (enlazador) correspondientes a estas librerías: *-lm*, *-lpthread*; los caracteres “-l” no son necesarios (figura 59).

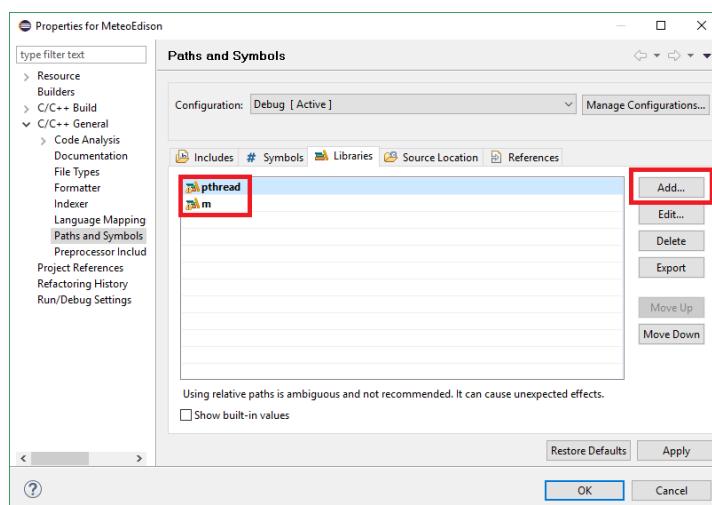


Fig. 59. Referencias generales de las librerías *math* y *pthread*

3. Presionamos “Apply” y cerramos las opciones.

Llegados a este punto ya tenemos todo lo necesario importado en nuestro proyecto, no obstante, al compilar, podemos apreciar ciertos errores en los códigos de los elementos importados, cosas que hay que corregir:

1. En el archivo cabecera de la librería correspondiente al sensor de humedad (HTU21D.h) añadimos la línea “**#define ARDUINO 168**” ya que el núcleo *Arduino* determina con dicho identificador la versión del IDE *Arduino* que se está utilizando y algunas librerías lo consultan para saber si incluir el archivo *WProgram.h* o *Arduino.h* (fichero que se renombra en versiones superiores a la 1.0). Definiendo dicha etiqueta con un valor de 168 decimos que nos encontramos en la versión *Arduino IDE 1.6.8* (figura 60).

```
HTU21D Humidity Sensor Library

#define ARDUINO 168

#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

#include <Wire.h>

#define HTU21D_ADDRESS 0x40 //Unshifted 7-bit I2C address for the sensor

#define TRIGGER_TEMP_MEASURE_HOLD 0xE3
#define TRIGGER_HUMD_MEASURE_HOLD 0xE5
#define TRIGGER_TEMP_MEASURE_NOHOLD 0xF3
#define TRIGGER_HUMD_MEASURE_NOHOLD 0xF5
#define WRITE_USER_REG 0xE6
#define READ_USER_REG 0xE7
#define SOFT_RESET 0xFE
```

Fig. 60. Definiendo la etiqueta ARDUINO en HTU21D.h

2. Excluimos la compilación del archivo *softwarepwm.cpp* del núcleo *Arduino*, ya que al haber importado de esta manera el núcleo, surge un conflicto entre ciertas definiciones de este archivo con el archivo *Tone.cpp*.
3. Analizando detalladamente la subida a la placa de un sketch *Arduino*, mediante su entorno de programación, podemos observar que el programa se lanza pasando como argumentos “*/sketch/sketch.elf /dev/pts/0*” mientras que el *Intel System Studio IoT Edition* exclusivamente usa el argumento “*/tmp/nombredelproyecto*”. El **primer argumento** es la **dirección/localización del programa** mientras que el **segundo argumento** es necesario para la **inicialización de la librería Serial** de manejo de la *UART*. Este segundo argumento, */dev/pts/0*, es la referencia al archivo que contiene el nombre del dispositivo correspondiente al puerto **COM**, por tanto, en el archivo *main.cpp* del núcleo *Arduino* aumentaremos el número de argumentos de entrada a dos, y

añadiremos el puerto serie correspondiente en el archivo de dispositivo apropiado:

```
// Incrementamos el número de argumentos a 2 y lo insertamos
argc = 2; argv[1] = "/dev/pts/0";

// Asociamos al dispositivo /dev/pts/0 el puerto serie USB
system("printf /dev/ttys0 1>/dev/pts/0");
```

- Otro aspecto a tener en cuenta es el hecho de que la placa Intel Edison presenta un servicio asociado a arrancar los Sketch Arduino en cuanto detecte que este se encuentra en la placa, por lo que, si alguna vez en el pasado hemos cargado un sketch Arduino a la placa, una vez reiniciada, éste sketch se ejecuta en segundo plano. Para evitar que se lance y utilice recursos hardware como la UART, los GPIO... debemos detener dicho servicio y de paso eliminamos el sketch (si existe), para ello, añadimos en el propio código de nuestra aplicación (tras las líneas insertadas anteriormente) lo siguiente:

```
// Detenemos el servicio de ejecución automática de los Sketch
// de Arduino
system("systemctl stop cloader");

// Eliminamos cualquier sketch Arduino existente en la placa
system("rm /sketches/sketch.elf");
```

Con los cambios anteriores, el archivo *main.cpp* presenta el código mostrado en la figura 61.

```
***** Global *****/
int main(int argc, char* argv[])
{
    char* platform_path = NULL;
    struct stat s;
    int err;

    /**
     * Modificamos los argumentos de entrada del programa para que corresponda con una subida del IDE Arduino
     */
    argc = 2; // Incrementamos el número de argumentos a 2
    argv[1] = "/dev/pts/0"; // Insertamos el segundo argumento
    system("printf /dev/ttys0 1>/dev/pts/0"); // Insertamos en el archivo /dev/pts/0 la referencia al puerto
    printf("\n");

    /**
     * Para asegurarnos que no se esté ejecutando un Sketch Arduino que previamente se haya cargado a la placa
     */
    system("systemctl stop cloader"); // Detenemos el servicio de ejecución automática de los Sketch de Arduino
    system("rm /sketches/sketch.elf"); // Eliminamos cualquier sketch Arduino existente en la placa
    /**
     */
```

Fig. 61. Modificaciones en *main.cpp*

Para finalizar con la preparación del entorno de desarrollo, necesitaríamos **añadir la librería UPM del sensor de presión MPL3115A2**, para este fin, se realizan los siguientes pasos:

1. Abrimos la vista de sensores soportados: *Intel(R) IoT/Sensor support view*.
2. Seleccionamos el sensor de presión atmosférica localizado en “**Atmospheric Pressure/mlp3115A2**” (figura 62).

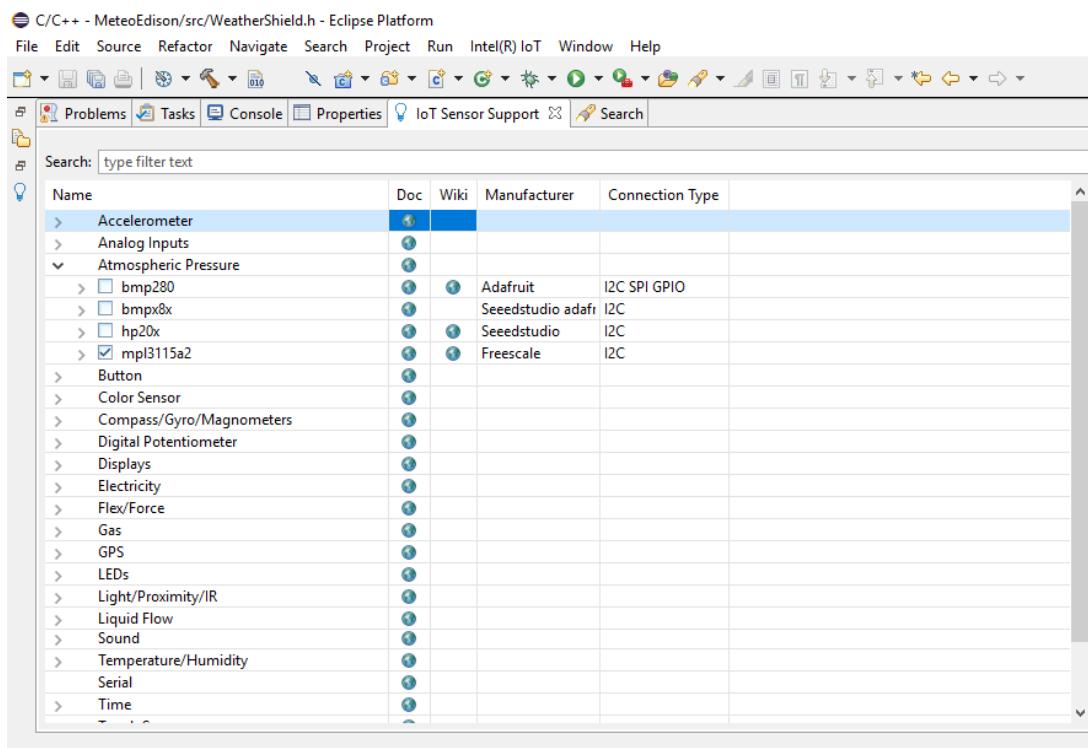


Fig. 62. Incluyendo librerías del repositorio de sensores UPM

4.2.3 Desarrollando la Inteligencia del Sistema METEO

En este apartado se explicará la **estructuración del código**, repasando los **elementos que componen al software** y se explicarán los **aspectos más significativos del código** desarrollado.

Ya que se desarrolla en el lenguaje C++ de medio-alto nivel, aprovechamos su capacidad de programación orientada a objetos para, a parte del uso de las librerías pre-implementadas, elaborar nuestras propias clases referentes a diversos elementos del sistema, de forma que nos facilitan la implementación y prueba de funcionamiento por separado de cada uno de estos elementos. Así, se han desarrollado clases para el **control del Weather Shield y del LCD**, para el **uso seguro del puerto serie** independientemente del proceso que lo utilice, y para la **gestión de creación y manejo de archivos de texto** plano.

De este modo, el proyecto presentará los siguientes archivos principales (excluyendo

claro está, las librerías externas que se utilizan):

- **MeteoEdison.cpp:** Archivo principal del programa. En él se encontrarán las funciones *setup()* y *loop()* pertenecientes a la capa de abstracción de Arduino (evidentemente el archivo inicial sería el *main.cpp* localizado en el núcleo de Arduino, pero ya que hacemos uso de la HAL Arduino, el papel de programa principal lo tomaría este archivo).
- **CSerialMutexed.h-CSerialMutexed.cpp:** Clase de inicialización y control del puerto serie de forma que su uso esté exento de problemas de acceso al recurso desde procesos diferentes. Para ello se recurre al uso de los elementos Mutex de los Threads POSIX (Pthreads), los cuales nos proporcionan exclusión mutua sobre el recurso compartido que en este caso presenta el puerto serie.
- **WeatherShield.h-WeatherShield.cpp:** Clase que engloba todo el control de los sensores del sistema, tanto de los presentes en el Shield ambiental, como del módulo de luz externo basado en fotorresistencia. A parte de inicializar cada uno de los sensores, esta clase nos permite realizar lecturas de los parámetros ambientales de forma simple, y asegura el uso del bus I2C mediante exclusión mutua (permite un acceso seguro al uso de los sensores desde cualquier proceso).
- **CLCD.h-CLCD.cpp:** Clase de inicialización y control de la pantalla LCD. Reduce y simplifica todo el uso del LCD a un solo método de escritura de texto en la línea del LCD que se indique.
- **CArchivoTexto.h-CArchivoTexto.cpp:** Clase que gestiona la creación de archivos de texto plano, así como el acceso, la utilización y el manejo de estos, tanto para escritura, como para lectura, consulta o eliminación. Además, la clase permite el envío TCP/IP de los archivos gracias a la implementación como cliente TCP.

Como se puede observar de las descripciones de estos archivos, los elementos hardware, como el conjunto de sensores y la pantalla LCD, son manejados en las clases *WeatherShield* y *CLCD* respectivamente; y hay que mencionar que, el último de los elementos hardware, el módulo de los pulsadores, es gestionado directamente en el fichero correspondiente al programa principal, el archivo *MeteoEdison.cpp*.

Otro aspecto a destacar es la relación existente entre todos estos archivos y los elementos software que implementan. El archivo *MeteoEdison.cpp* es el archivo principal que gobierna al resto de archivos, en él se crean y utilizan los objetos de las clases correspondientes a los elementos implementados en los otros archivos.

La estructura de las clases, así como la dependencia entre ellas, pueden ser observadas

en el siguiente esquema, correspondiente a la figura 63.

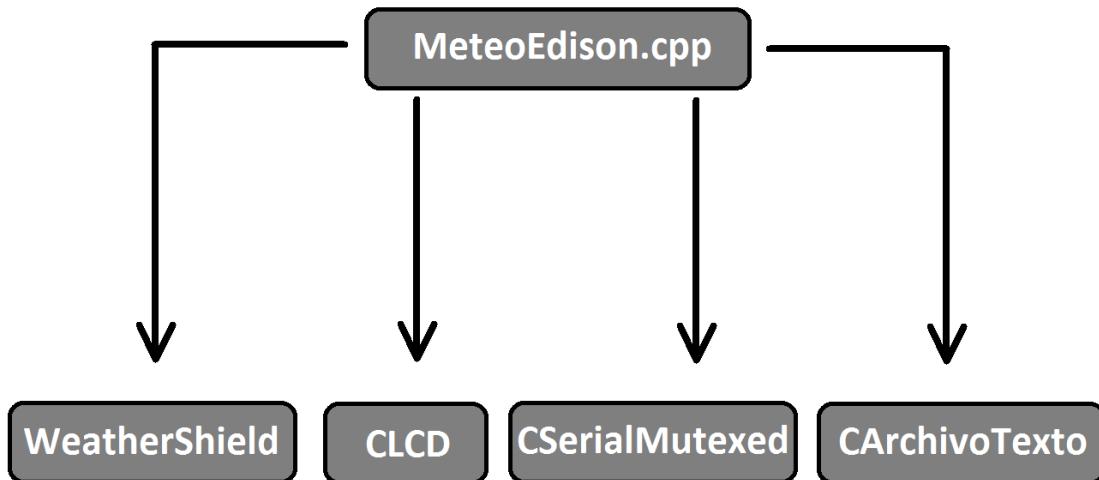


Fig. 63. Clases y dependencias del sistema

En la figura 64 se muestra de forma gráfica la secuencia de acciones que realiza el software y desglosaremos en detalle a continuación.

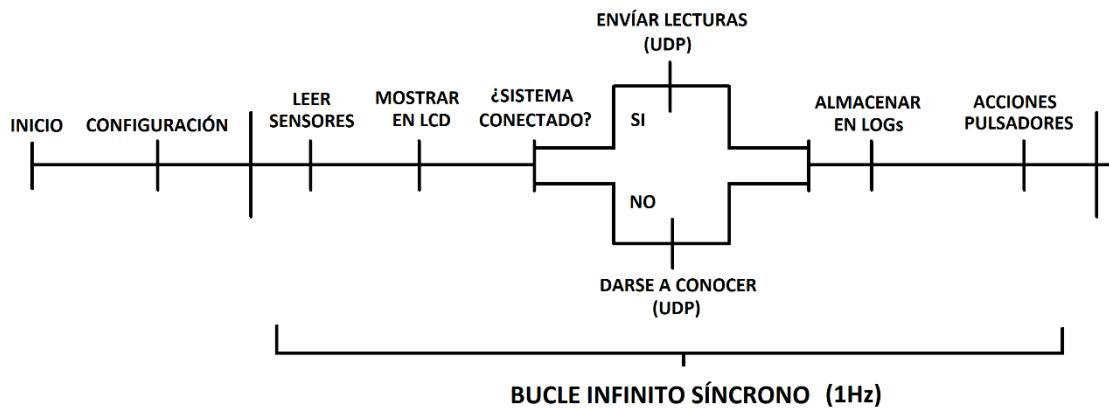


Fig. 64. Comportamiento global del sistema

Al inicializarse el programa, se configuran los periféricos de la Intel Edison, que se utilizarán; la pantalla LCD, los sensores del Weather Shield, los pines GPIO para la lectura analógica del sensor de luz, las entradas digitales del módulo de pulsadores y el puerto serie. Además, se configura la fecha del sistema a partir del reloj interno RTC que se encuentra en la placa y se abre un Socket de comunicaciones basado en datagramas para el envío y recepción de paquetes UDP en la red.

A continuación, el sistema comienza un proceso de operación síncrono, por el que,

cada segundo, se realizan **lecturas de todos los sensores**, estas lecturas son procesadas y mostradas por la **pantalla LCD**, se determina si el **sistema** está conectado o no con algún sistema de monitorización (aplicación UWP), en caso de estar conectado, se sacan las lecturas de los sensores por el puerto serie y se envían en un paquete UDP, si no se encuentra conectado, se envía un paquete UDP con un identificador del sistema y la dirección IP del mismo (este es un paquete Broadcast para dar a conocer al sistema en la red); tras la lectura y el posible envío de los datos, estos son almacenados en **archivos de registro o Log** según el paso del tiempo, habrá un Log del último minuto, un Log de la última hora y un log de las últimas 24 horas (Todos ellos con 60 muestras cada uno, por lo que, cuando se adquieran 60 muestras correspondientes a cada segundo, estas se promedien e inserten su valor medio en el log de minutos como una única muestra; al igual pasaría con las muestras de minutos y horas), de modo que cada muestra irá incrementando los Logs. Además de esto, **se gestiona el estado de los pulsadores** para determinar si se ha presionado alguno y se debe realizar alguna operación adicional asociado a ello.

De forma asíncrona, el sistema se encuentra a la espera de recibir algún paquete UDP, cuando reciba alguno, se encargará de gestionarlo como se requiera. Estos paquetes, serán enviados por el sistema de monitorización (aplicación UWP) y consistirán en una serie de mensajes que albergan **comandos de petición** hacia el sistema, tales como **comandos para indicar la conexión establecida** entre el sistema de monitorización y el dispositivo o la desconexión entre estos y **comandos para solicitar el envío de los archivos Logs**.

Con lo anterior, podemos hacernos una idea global del comportamiento del sistema, más adelante entraremos en detalle, pero primero especificaremos tanto la funcionalidad de los pulsadores, como la forma de representar los datos en el LCD.



Fig. 65. Representación de datos por el LCD

El LCD tiene dos líneas de 16 caracteres representables en cada una, de modo que, representaremos en la **primera línea la fecha del sistema** y dejaremos la **segunda** para mostrar **los valores de los parámetros ambientales**. En la figura 65 se muestra un ejemplo donde se puede apreciar la fecha, la temperatura y la humedad en el display. Ya que tenemos 4 parámetros, temperatura, humedad, presión y luz; con sus valores decimales cada uno de ellos, no es de extrañar que no quepan en una sola línea del LCD, por ello, se recurre a representar exclusivamente dos de los parámetros en la línea y, habrá que buscar alguna forma para poder comutar esta dupla representada por el otro par de parámetros, ahí es donde entran en juego los pulsadores del sistema.

El módulo de pulsadores se compone de **tres botones**, de entre los cuales, **uno de ellos permitirá** al usuario/operario **comutar los parámetros ambientales que se muestran en el LCD**, **otro de los pulsadores permitirá desconectar manualmente** al dispositivo del sistema de monitorización y **el último eliminará los archivos Logs** internos del dispositivo, reseteando así el historial de muestras adquiridas.

Ahora que tenemos una idea general del funcionamiento del sistema y volviendo al código desarrollado, podemos entrar más en detalle a especificar la funcionalidad de cada elemento software implementado.

WeaherShield:

La clase *WeatherShield* permite, como ya hemos dicho, controlar todos los sensores que participan en nuestro sistema, y acceder a ellos de forma mediante exclusión mutua del bus I2C.

Al **crear e iniciar** un objeto de esta clase, se **inicializan** los sensores de humedad **HTU21D** (que utiliza la librería *Arduino* de *Sparkfun*) y presión **MPL3115A2** (que usa la librería UPM correspondiente), así como el **pin de entrada analógica** (en la que se utiliza la librería MRAA) para leer los valores de luminosidad del sensor de luz basado en fotorresistencia.

Con el objeto creado e inicializado, se pueden **realizar lecturas de los distintos sensores**. Se implementan métodos para **leer cada sensor por separado** y devolver el valor leído, y un método global para **realizar una lectura de todos juntos**, mediante el método *leer()*. **Las lecturas devuelven unas estructuras de datos** internas a la clase que contienen los parámetros correspondientes.

Los parámetros ambientales leídos desde esta clase se basan en la **temperatura, la humedad, la presión y la luminosidad ambiental**. No obstante, **el parámetro de presión nos permite además determinar la altitud respecto al nivel del mar** a la que nos encontramos. No es un valor exacto, pues se obtiene a partir de la relación que existe entre la presión atmosférica base a nivel del mar (100KPa = 0m de altitud) respecto a la temperatura, pero si lo suficientemente aproximado. Por tanto, la altitud también será tratada en nuestro sistema como un parámetro de lectura más.

Cabe mencionar, que el parámetro de temperatura se puede obtener tanto desde el sensor de humedad, como desde el sensor de presión. En vista a las pruebas realizadas en el desarrollo se determina que, aunque los valores obtenidos de ambos sensores son prácticamente iguales, el sensor de presión presenta más estabilidad en la lectura, por lo que, **en el sistema, la lectura del parámetro de temperatura se realizará a partir del sensor de presión MPL3115A2.**

En nuestro sistema, se precisa conocer los valores de todos los sensores, por lo que se **utiliza el método de lectura global** de los sensores, leer(), y se recogen los valores devueltos en la estructura de datos **datos_weathershield**.

CLCD:

Poco más se puede añadir a la clase del LCD, en ella se utiliza la librería *Arduino LiquidCrystal* para inicializar la comunicación paralela con la pantalla LCD, y se implementa un método que permite insertar el texto proporcionado por el usuario, en la línea fijada.

En esencia, lo que este método de escritura hace, es posicionar el cursor del LCD en el primer carácter de la línea que especifica el usuario y sobrescribe el contenido de la línea con el texto proporcionado.

CSerialMutexed:

Esta clase utiliza la librería **Serial** y permite que se utilice el puerto serie de manera segura desde cualquier proceso de nuestro programa.

La clase implementa métodos semejantes a los que se encuentran en la librería Serial, pero presentan *Mutex* para la exclusión mutua del puerto serie cuando se realiza alguna llamada a este.

CArchivoTexto:

Con esta clase se pueden crear archivos de texto plano y utilizarlos de forma simple dadas nuestras necesidades.

Esta clase **se desarrolló para satisfacer la necesidad de controlar los 3 archivos de Logs** que almacenan las muestras leídas de los sensores. Entre las características principales, la clase facilita esta tarea permitiendo **insertar líneas de texto en la última posición del archivo**, así como **eliminar la primera línea y reposicionar todo el texto** de forma que al incluir una muestra cuando se ha alcanzado el número máximo de estas (60 para los Logs correspondientes al último minuto y última hora; y 24 para el correspondiente a las últimas 24 horas) se realice de forma eficiente. Además, la clase implementa funciones para determinar el número de líneas escritas en el archivo, así

como mostrar todo su contenido.

Por tanto, **la clase implementa, el promediado de las muestras** almacenadas en los archivos, útil para guardar las muestras correspondientes al promedio del último minuto (60 muestras de cada segundo) en el archivo de última hora (nueva muestra de minuto correspondiente al promedio de las 60 muestras de los segundos) y de esté con respecto al archivo de últimas 24 horas respectivamente.

El promediado consiste en la lectura de cada línea de texto del archivo para realizar la media aritmética de cada valor de parámetro ambiental (temperatura, humedad, presión, altitud y luz) y retornar una nueva línea de texto correspondiente a este promedio.

Por último, se implementa un **método para enviar los archivos por flujo TCP/IP** a la dirección que se le indique, para ello se abre un nuevo socket, esta vez TCP, y se realiza la conexión de este con el sistema de monitorización.

Programa Principal (MeteoEdison):

El programa principal define el comportamiento global del sistema que vimos anteriormente (figura 63) y es el que hace uso del resto de elementos software que acabamos de detallar.

Para comenzar, el programa establece las definiciones generales, las variables globales y locales, y los objetos de las clases (el de los sensores, el puerto serie, el LCD y los 3 de cada archivo de Log):

```
WeatherShield Sensores;
C_SerialMutexed SerialMutexed;
C_LCD LCD;
CArchivoTexto Log_min((char*)"log_min.log");
CArchivoTexto Log_hor((char*)"log_hora.log");
CArchivoTexto Log_dia((char*)"log_dia.log");
```

Tras ello, se inician y configuran todos los elementos necesarios y **se crean cuatro hilos de ejecución, hebras:**

```
pthread_create(&H_principal, NULL, P_principal, NULL);
pthread_create(&H_tiempo_transc, NULL, P_tiempo_transc, NULL);
pthread_create(&H_ctrlPulsadores, NULL, P_ctrlPulsadores, NULL);
pthread_create(&H_recepcion_udp, NULL, P_recepcion_udp, NULL);
```

La primera de ellas es la **hebra principal** y será en **donde se lleve a cabo todo el proceso principal de la funcionalidad del sistema**; se utiliza una hebra, en vez de utilizar el propio hilo de ejecución inicial (con la función *loop()*) para evitar posibles problemas de prioridades, por tanto, el hilo de ejecución inicial, tras crear las hebras, se detiene indefinidamente.

La **segunda hebra** se encargará de **determinar el transcurso del tiempo**, y será la

responsable de indicar, mediante la activación de un semáforo, el **transcurso de cada segundo**. La hebra principal se mantendrá a la espera de que el semáforo correspondiente al transcurso de un segundo sea activado por esta segunda hebra y, esto, **establecerá la sincronía de nuestro proceso principal**.

La **tercera hebra** se corresponde simplemente con el proceso (asíncrono) de **detección del estado de los pulsadores** (controla el módulo de los pulsadores). Determina el estado de los pulsadores mediante variables compartidas y, la hebra principal consulta dichas variables para determinar si debe realizar alguna acción.

La **última de las hebras** se encarga de **gestionar la recepción de los mensajes UDP** recibidos a través del socket abierto (puerto 3333). La hebra **determina la conexión y/o desconexión del dispositivo** con el sistema de monitorización **y la petición de enviar los Logs**.

Entrando más en detalle en los mensajes UDP con las peticiones realizadas desde el sistema de monitorización, cabe destacar que, **al recibir una petición de conexión**, el dispositivo recibe en el mensaje el comando de conexión y la fecha del equipo donde se encuentra el sistema de monitorización, de modo que **el dispositivo actualiza tanto la hora de su sistema, como la hora del RTC interno** (el dispositivo se pone en hora cada vez que un sistema de monitorización se conecta a él).

Por otro lado, cuando el sistema de monitorización envía una **petición de envío de los archivos Logs**, el dispositivo **crea una hebra encargada de realizar dicho envío** (en este caso envío TCP/IP por el puerto 6666).

Para terminar con las características más destacables del software desarrollado, **comentaremos a grandes rasgos el formato de los mensajes que se envían por red**. Así, cuando se realiza una lectura de los sensores, los valores leídos se integran en una **cadena de caracteres** con el siguiente formato:

```
"fecha=%s,temp=% .2f,hum=% .2f,pres=% .2f,alti=% .2f,luz=% .2f;\n"
```

Un ejemplo de mensaje sería el siguiente:

```
"fecha=27/10/2016-17:29:19,temp=18.53,hum=60.00,pres=102.25,  
alti=8.00,luz=75.75;\n"
```

Esta cadena es la que se almacena en cada línea de los archivos de Log y, también, es la que se envía a través de UDP cuando la conexión está establecida. De modo que el sistema de monitorización se encargará de recibirla y extraer los valores de cada parámetro.

Por otro lado, los **mensajes enviados, desde el sistema de monitorización, para realizar las peticiones de conexión/desconexión y exportar los archivos de log**, siguen el formato:

Mensaje de conexión -> "Codigo:%d;Fecha:%s"

Mensaje de desconexión o exportación de Logs -> "Codigo:%d"

Como se puede apreciar el comando de conexión lleva el parámetro de fecha del sistema de monitorización, para actualizar la fecha del dispositivo, tal y como se ha dicho anteriormente.

Por último, el **mensaje que publicita el dispositivo a la red** cuando no se encuentra conectado muestra un valor numérico correspondiente a su identificador y la dirección IP local en la que se encuentra (esto se ha desarrollado pensando en una posible ampliación del sistema en el futuro, en el que se utilicen más de un dispositivo sensor, y se puedan identificar para realizar su conexión desde el sistema de monitorización). El formato de este mensaje utilizado para darse a conocer presenta el siguiente formato:

"Dispositivo=%d;Direccion=%s;\n"

Un ejemplo de mensaje sería el siguiente:

"Dispositivo=0023;Direccion=192.168.1.131;\n"

Otros Aspectos:

La inclusión del núcleo Arduino a un proyecto del entorno *Intel System Studio IoT Edition*, provoca que haya ciertos aspectos considerables en el proceso de desarrollo.

En un proyecto donde no se utilice la capa de abstracción hardware de Arduino, tenemos que el método por defecto para mostrar datos por la consola utiliza la salida por pantalla **stdout**, pero si nos fijamos en nuestro proyecto donde se usa Arduino, en el archivo main.cpp **se re-direccionan** las salidas de tanto **stdout** como de **stderr** (errores) **a unos archivos de Log (localizados en "/tmp/log.txt" y "/tmp/log_er.txt") implícitos en el sistema**, luego **estas llamadas no son posibles para visualizar datos por la consola** (lo que escribiríamos por **stdout**, por ejemplo, se insertaría en el archivo de log en cuestión, y no se mostraría en la consola). Para mostrar datos por la consola, se puede recurrir a escribir por el puerto serie.

En la *Intel Edison* el **reloj del sistema es sincronizado** con los servidores de google a través de un proceso que se ejecuta en segundo plano (demonio) siempre que se tiene conexión a internet. Se puede ver ejecutando "systemctl status systemd-timesyncd" o "timedatectl status". Esto provoca que no podamos actualizar la hora del dispositivo según la hora del sistema de monitorización, por lo que, procedemos a desactivarlo en el código, para ello, se realiza la llamada de sistema siguiente en la función setup:

```
system("timedatectl set-ntp false");
```

Otra característica importante de mención es que, **al utilizar el núcleo Arduino, se inicializan y configuran en cierta medida las UARTs** asociadas a los puertos serie y, por ende, **podrían experimentarse incompatibilidades o problemas al intentar**

utilizar algún puerto serie mediante la librería MRAA. Para evitar esto, se debe utilizar exclusivamente las librerías Serial que otorgan la HAL de Arduino.

5 Programa de Monitorización y Control (MeteoPanel)

El último elemento a desarrollar, que complementa al dispositivo que realiza las lecturas de los parámetros ambientales y las envía por red, y completa la plataforma METEO, es el **sistema de monitorización**, una aplicación que brinda acceso a la información proporcionada por el dispositivo, así como al control de este, desde un ordenador y un entorno gráfico amigable, todo ello gracias a la **Universal Windows Platform (UWP)**.

Para simplificar, se puede definir al sistema de monitorización como una aplicación universal (desarrollada con UWP) que permite al usuario conectarse con los dispositivos de la plataforma METEO y ver gráficamente los valores de los parámetros ambientales recogidos por este.

A continuación, **se repasará** tanto el proceso de **preparación del entorno de desarrollo** utilizado para la elaboración de aplicaciones universales, como la **funcionalidad de la aplicación**, el proceso de **implementación** y los **aspectos más característicos** del desarrollo.

5.1 Preparando el Equipo para Desarrollar Aplicaciones Universales (UWP)

El **desarrollo de aplicaciones universales** utiliza el entorno de desarrollo integrado **Visual Studio**, de Microsoft (evidentemente el desarrollo mediante UWP, perteneciente a Microsoft, es facilitada a través del entorno predilecto de éste).

Como se comentó anteriormente, **existen varios lenguajes de programación usables en el desarrollo UWP** (*C#, C++, Visual Basic y JavaScript*), de entre ellos, para este proyecto, **se escoge el lenguaje C#**, orientado a objetos, simple, moderno y de propósito general.

Para comenzar con el desarrollo, accedemos a la web de Visual Studio (<https://www.visualstudio.com/es/vs/>) y **descargamos el entorno** (figura 66).



Fig. 66. Descargando el IDE Visual Studio

Una vez descargado, lo **instalamos** y, tras completarse todo el proceso, **iniciamos el programa**, tal como se muestra en la figura 67.

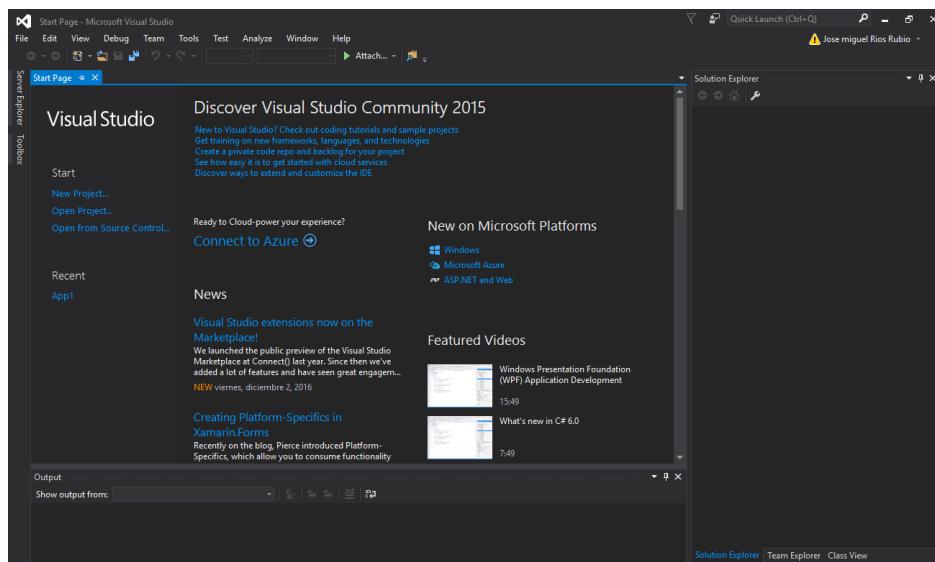


Fig. 67. Ventana de inicio de Visual Studio

A continuación, procedemos a **crear nuestro proyecto** basado en *UWP*. Para ello, abrimos la ventana de creación de proyecto nuevo seleccionando “*File/New/Project...*”. En esta ventana crearemos un proyecto a partir de la plantilla de aplicación universal en blanco, localizada en “*Templates/Visual C#/Windows/Universal*” y le damos nombre al proyecto, en este caso, como se puede apreciar en la figura 68, **MeteoPanel**.

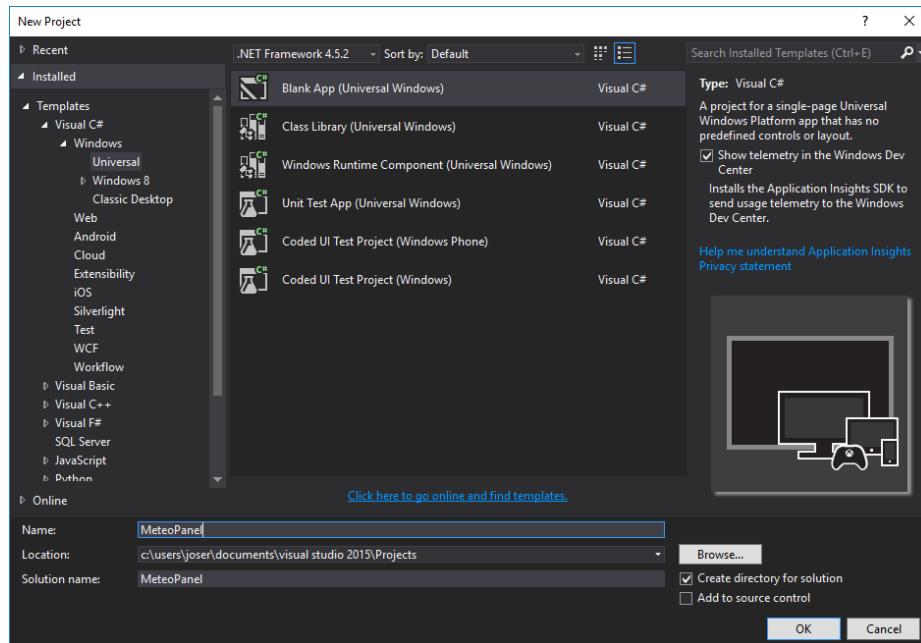


Fig. 68. Creación de proyecto universal en Visual Studio

Con esto estaríamos listo para desarrollar la aplicación, no obstante, en la fecha en la que se realiza este proyecto, **no existe una herramienta “oficial”**, dentro de la API que proporciona la UWP de Microsoft, **con la que representar gráficas dinámicas en aplicaciones universales**, las cuales utilizaremos para mostrar los valores de los parámetros ambientales. Por este motivo, **se recurre al recurso de un tercero**, un subconjunto de un **ToolKit** que incluye controles XAML para representar datos de forma visual (gráficas), el **WinRTXamlToolkit.Controls.DataVisualization.UWP** [18]. Este *Toolkit* no pertenece a Microsoft, pero es desarrollado, por cuenta ajena, por varios de sus empleados, los cuales trabajaron en el equipo de desarrollo del *Toolkit* “oficial” **Silverlight** (que en primera instancia buscaba dar soporte a contenidos multimedia en el desarrollo de aplicaciones web) y, es por esto mismo, que este recurso se basa en Silverlight y se constituye como “port” de este (se ha portado parte de su base para darle compatibilidad UWP).

Para incluir el *Toolkit*, se recurre a la herramienta de manejado de paquetes que proporciona Visual Studio: **NuGet Package Manager**. Esta herramienta brinda al desarrollador un acceso simple y rápido al repositorio de paquetes, libre y abierto, llamado NuGet, además, de permitir importar los paquetes que se requieran, en el proyecto/solución actual.

Procedemos a abrir esta herramienta de Visual Studio mediante “Tools/NuGet Packages Manager/Manage NuGet Packages for Solution...”, una vez en ella, seleccionamos la pestaña de búsqueda “Browse” y buscamos e instalamos el paquete correspondiente al *Toolkit* (buscamos *WinRTXamlToolkit.Controls.DataVisualization*):



WinRTXamlToolkit.Controls.DataVisualization.UWP. Este paso es representado en la siguiente imagen, la figura 69.

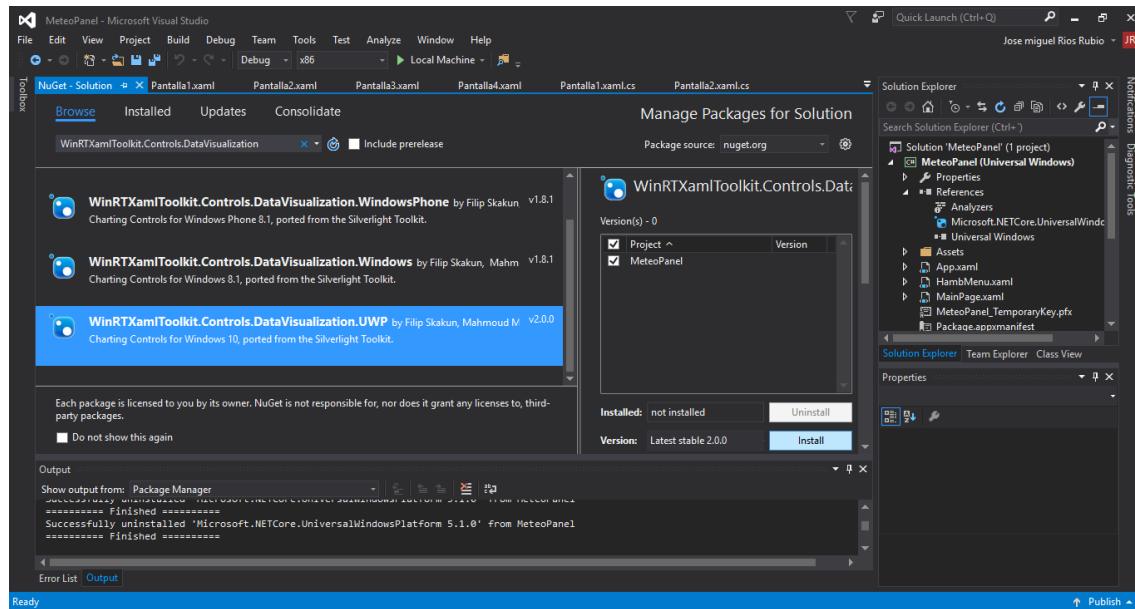


Fig. 69. Incluyendo controles de visualización de datos en el proyecto universal

Una vez incluida, ya tendremos las herramientas necesarias para incluir gráficas en nuestra aplicación universal.

5.2 Desarrollando la Aplicación Universal

Comencemos por un repaso global de las características funcionales que tendrá la aplicación universal.

Según las **especificaciones** dadas, el sistema de monitorización debe de presentar **dos métodos** principales con los que el usuario/operario puede **acceder a la información** de los parámetros ambientales, el primero de ellos es una **representación de los valores ambientales a tiempo real**, mientras que el segundo de ellos permite **mostrar un registro o histórico de los valores**.

Para mostrar los valores **a tiempo real**, en nuestra aplicación universal, representaremos en distintas **columnas gráficas** a cada parámetro ambiental (temperatura, humedad, luz, presión y altitud) y mostraremos los valores cada muestra en ellos de forma automática. Por otro lado, para **representar los históricos** utilizaremos una única **gráfica lineal de dos ejes** y, presentaremos diversos **controles** al usuario para que este le **indique al sistema que parámetro y periodo temporal representar** en la gráfica.

Uno de los aspectos a tener en cuenta, visto lo anterior, es que, **todas esas gráficas**

no se pueden representar de forma óptima para su visualización en una única ventana, por lo que se recurrirá al uso de algún elemento que permita cambiar los elementos de dicha ventana y, así, **utilizar varias pantallas virtuales para cada tipo de representación**, en una pantalla se mostrarán las gráficas a tiempo real y en otra la gráfica del histórico.

Y ya que se hace uso de varias pantallas, **optamos por subdividir toda la funcionalidad del sistema de monitorización en pantallas virtuales**. De este modo, **se establecen 5 pantallas** distintas, una para el **control de la conexión/desconexión** con los dispositivos, dos para las **representaciones de los datos a tiempo real y registro**, una pantalla para los **ajustes del sistema** y una última de **información sobre la plataforma METEO**.

La **implementación de la funcionalidad de múltiples pantallas** se puede conseguir de diversas formas, no obstante, para aprovechar las nuevas características de *UWP*, usaremos lo que se denomina “**Hamburguer Menu**”, un menú basado en el control *XAML Splitview*. Este tipo de menú es típico y reconocible por ser utilizado en las aplicaciones de *Smartphones* y *Tablets* y permite abrirse y compactarse a voluntad del usuario. Con este menú, navegaremos entre las distintas pantallas implementadas. En la figura 70 se representa los dos estados (abierto y compactado) del menú.

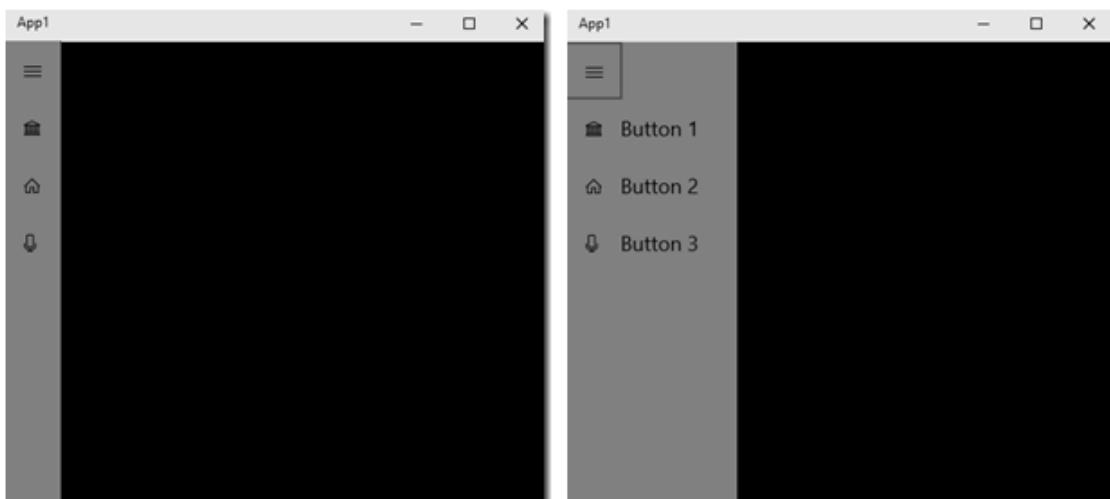


Fig. 70. Incluyendo controles de visualización de datos en el proyecto universal

Para comprender el funcionamiento genérico de la aplicación, recorreremos el caso de uso estándar del mismo por parte de un usuario.

Cuando el usuario lanza el sistema de monitorización en su equipo, se le mostrará la **pantalla de Conexión/Desconexión**, (en el proyecto *UWP Pantalla1*), con la que se puede detectar los dispositivos METEO que se encuentran activos (transmitiendo su paquete de reconocimiento) en la red, mostrando su identificador y su dirección de red, de modo que el usuario puede seleccionar uno de los dispositivos, y conectarse a él, tal

y como puede apreciarse en la figura 71.



Fig. 71. Aplicación universal. Pantalla de conexión

Una vez conectado, el usuario puede pasar, mediante el *Hamburguer Menu*, a la **pantalla de monitorización de datos a tiempo real** (en el proyecto, *Pantalla2*), y visualizar en las gráficas los valores de los parámetros ambientales, que va leyendo el dispositivo a cada segundo, como se observa en la figura 72.

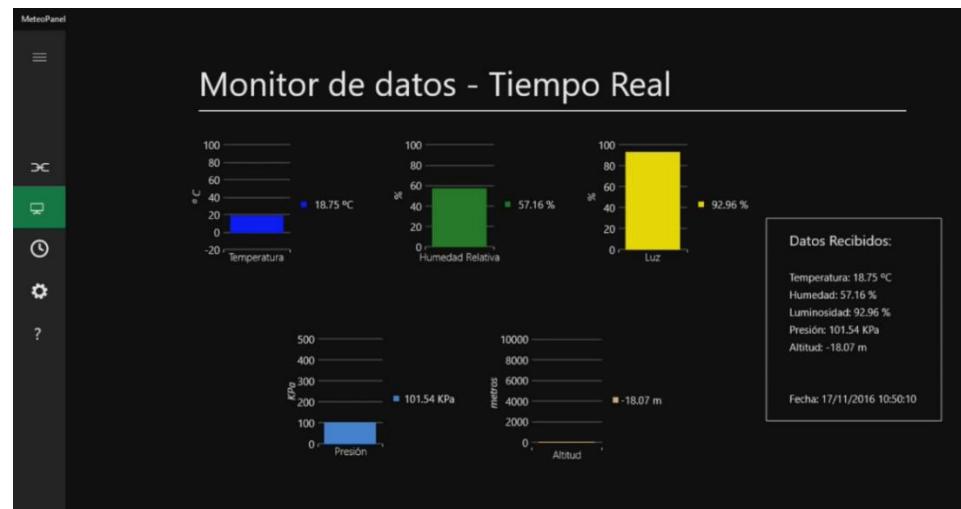


Fig. 72. Aplicación universal. Pantalla de monitorización a tiempo real

En el caso de que el usuario requiera examinar el registro de valores de algún parámetro, obtenidos por el dispositivo a lo largo de un periodo temporal, deberá pasar a la **pantalla de registro histórico** (en el proyecto, *Pantalla3*). En ella, podrá seleccionar el parámetro ambiental (temperatura, humedad, presión, altitud o luz) y el rango de tiempo (último minuto, última hora o últimas 24 horas) y ver la gráfica lineal

correspondiente a dicho periodo. Además, esta pantalla presenta un botón para exportar los archivos de Log (usados para la representación anterior) localizados en el dispositivo. La figura 73 muestra la pantalla de registro histórico representando las muestras correspondientes al Log del último minuto.

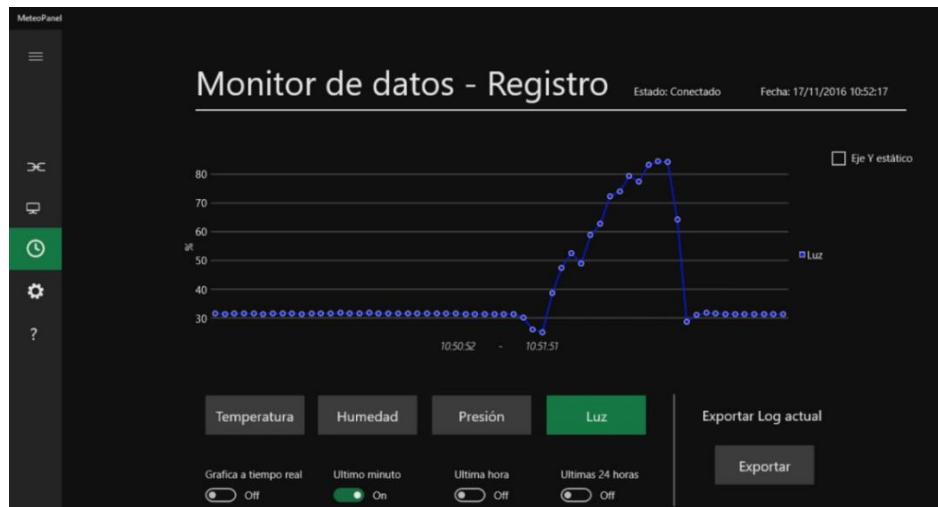


Fig. 73. Aplicación universal. Pantalla de registro histórico

Si el usuario que utiliza el sistema necesita un control avanzado sobre las comunicaciones que se llevan a cabo entre el sistema de monitorización y el dispositivo, puede acceder a la **pantalla de ajustes** (en el proyecto, *Pantalla4*) donde, entre otras cosas, se pueden configurar los puertos UDP y TCP utilizados por el sistema de monitorización para la comunicación de datos, por defecto 3333 y 6666 respectivamente; además de especificar el adaptador de red a utilizar, por defecto seleccionada la tarjeta wifi, como puede verse en la figura 74.

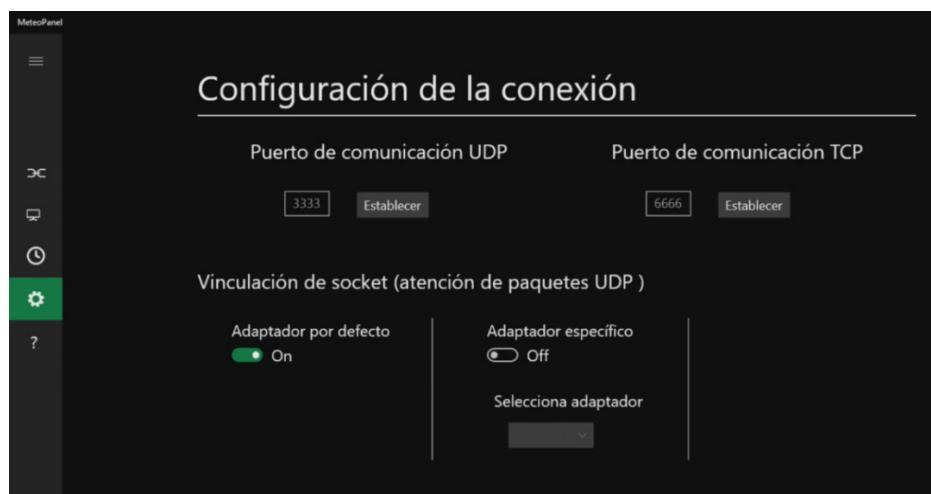


Fig. 74. Aplicación universal. Pantalla de ajustes

La última pantalla, la **pantalla de información** (en el proyecto, *Pantalla5*) es, como su propio nombre indica, una pantalla informativa que contiene exclusivamente información sobre el sistema METEO y el proyecto. Esta pantalla se representa en la figura 75.

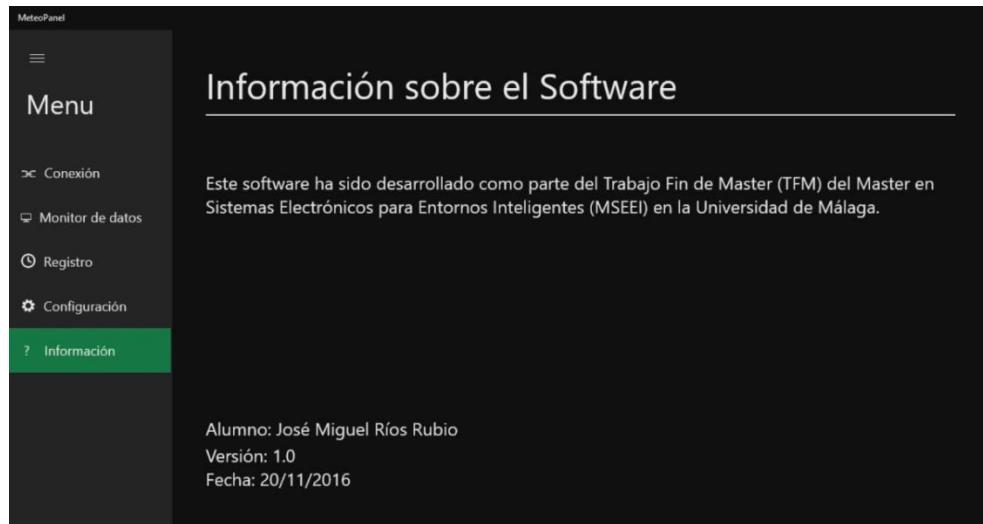


Fig. 75. Aplicación universal. Pantalla de información

Por último, cuando el usuario decida liberar la conexión con el dispositivo al que se encuentra conectado, deberá de acceder nuevamente a la pantalla inicial de conexión/desconexión, y proceder a la desconexión.

Ahora que tenemos una idea global de la funcionalidad, pasaremos a **comentar las características y aspectos más destacables de la implementación**, para ello, recorremos las distintas pantallas implementadas e iremos comentando la funcionalidad de cada una de ellas, pero, antes de eso, hay que comentar la estructuración del código.

El desarrollo del proyecto de aplicación universal en el entorno Visual Studio presenta un conjunto de archivos que hacen referencia tanto a las opciones del proyecto, como a los propios archivos de nuestra aplicación. Todos estos archivos, así como la estructura del proyecto, se muestra en la figura 76.

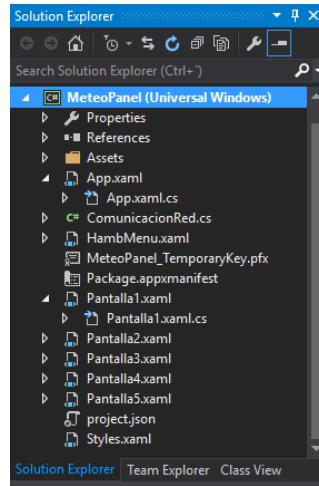


Fig. 76. Explorador de soluciones. Estructura del proyecto

Hay que destacar que cada uno de los archivos que presenta un entorno gráfico se componen de un conjunto de dos ficheros, uno de formato **xaml**, que modela la interfaz de usuario, y otro de formato **cs** (C#), correspondiente al código que habrá detrás de los elementos gráficos de dicha interfaz.

A continuación, se especifica la funcionalidad de los archivos más destacables para el desarrollo de nuestra aplicación:

App.xaml/xaml.cs:

Estos archivos componen el **elemento principal de la aplicación**, el cual es ejecutado por defecto al comenzar el programa. Internamente se establecen parámetros de configuración de la funcionalidad de la aplicación y, en nuestro proyecto, este archivo **se encarga de ejecutar la pantalla “HambMenu”**:

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    .
    .
    .
    if (e.PrelaunchActivated == false)
    {
        if (rootFrame.Content == null)
        {
            rootFrame.Navigate(typeof(HambMenu), e.Arguments);
        }
        Window.Current.Activate();
    }
}
```

ComunicacionRed.cs:

Archivo auxiliar en el que se **implementan los aspectos de comunicación de Red** a través de una clase denominada *ComunicacionRed*. Lo denotamos por “auxiliar” ya que, por sí solo, no realiza ninguna acción, **su contenido es utilizado desde el resto de**

archivos de nuestro proyecto. Este archivo **contiene las listas** (objetos) que contendrán los valores **de los parámetros ambientales** que se reciban, así como una **estructura de datos** para almacenar los valores recibidos a tiempo real.

En este archivo **se implementan** métodos para la **apertura de los sockets** (UDP y TCP), métodos para **establecer la conexión y/o desconexión** del sistema de monitorización con el dispositivo, una tarea para el **envío de mensajes UDP** y eventos asíncronos de **recepción de mensajes tanto UDP como TCP** y métodos para el **almacenamiento de los datos recibidos**.

De lo anterior **lo más destacable es la recepción de los datos**. En estas funciones, que se ejecutan de forma automática cuando se recibe algún paquete UDP/TCP (pues son los manejadores de estas recepciones, asociadas cuando se abrieron los sockets), hay que destacar que realizan todo el proceso de descomposición del mensaje recibido y el almacenamiento de los valores extraídos.

Cuando llega un paquete UDP, se extrae el mensaje asociado y se determina si es un mensaje de publicitación del dispositivo, o un mensaje de datos ambientales:

```
if ((mensajeRecibido.Contains("Dispositivo")) && (mensajeRecibido.Contains("Direccion")))
{
    ...
}
else if (mensajeRecibido.Contains("fecha") && (mensajeRecibido.Contains("temp")) && ...
{
    ...
}
```

Una vez determinado si el tipo de mensaje se corresponde con uno de datos, **se extraen los valores de estos mediante la subdivisión de la cadena de caracteres**, este es el motivo de utilizar el carácter ';' como separador de parámetros en el mensaje enviado. Una vez hecha esta subdivisión, **se extraen los valores de cada parámetro a partir de conversiones de texto al formato adecuado** (para los parámetros ambientales, variables de punto flotante) y **se guardan en la estructura de datos correspondiente y en las listas de registro**:

```
String[] substrings = mensajeRecibido.Split(';');
foreach (var substring in substrings)
{
    if (substring.Contains("fecha"))
    {
        String[] substrings2 = substring.Split('=');
        datos_recibidos.dia = substrings2[1].Substring(0, 10);
        datos_recibidos.hora = substrings2[1].Substring(11, 8);
    }
    else if (substring.Contains("temp"))
    {
        String[] substrings2 = substring.Split('=');
        datos_recibidos.temp = float.Parse(substrings2[1]);
    }
    else if (substring.Contains("hum"))
    {
        String[] substrings2 = substring.Split('=');
        datos_recibidos.hum = float.Parse(substrings2[1]);
    }
    ...
}
```

Sobre la recepción de mensaje por TCP cabe destacar que este es consecuencia del envío previo de un mensaje UDP hacia el dispositivo con un mensaje de petición de exportar los archivos Logs. Cuando se recibe un mensaje TCP, en primera instancia se extrae el nombre del archivo a exportar y el número de líneas de datos de dicho Log, tras esto, el resto del mensaje se compondrá el contenido del Log.

HambMenu.xaml/xaml.cs:

Este elemento se corresponde con la ventana principal en la que **se implementa la “vista dividida”** (*SplitView*) con el fin de desarrollar el ***Hamburguer Menu*** que dota al sistema de la capacidad de movernos entre las diversas pantallas que contiene la aplicación.

En el archivo HambMenu.xaml se insertan y caracterizan visualmente los controles XAML para la vista dividida y el menú de navegación entre pantallas, mientras que en el archivo HambMenu.cs se implementa el comportamiento que tendrá el menú, tanto los cambios visibles sobre este al abrirlo o compactarlo, como la acción asociada a la pulsación de cada botón presente en el menú (en este caso el cambio entre pantallas).

Pantalla1.xaml/xaml.cs:

Este elemento se corresponde con la **pantalla de conexión/desconexión**.

En el archivo **Pantalla1.xaml** se modela la apariencia de la pantalla, con sus controles correspondientes. Estos controles son **dos botones, uno de conexión y otro de desconexión**, una **línea de texto sobre el estado de la conexión** (muestra al usuario si el sistema está conectado o no) y una **ListBox que muestra los dispositivos que se encuentren transmitiendo en la red**, permitiendo seleccionarlos.

Sobre el código de esta pantalla, el archivo **Pantalla1.cs**, hay que comentar la **apertura de los sockets** asociados a la comunicación de red y la **inicialización de las listas** que serán usadas **para almacenar** los valores de los **registros históricos** de tiempo que se consulten al dispositivo (ambas cosas haciendo uso de la clase implementada en el archivo *ComunicacionRed.cs*).

```
ComunicacionRed.poblar listas();
ComunicacionRed.abrirPuerto udp();
ComunicacionRed.abrirPuerto_tcp();
```

Otros aspectos destacables del código son los **métodos correspondientes a los botones de conexión y desconexión**, los cuales simplemente **crean tareas** que **hacen la llamada** correspondiente al método de **conexión/desconexión** implementado en la clase *ComunicacionRed* y **esperan a que se realice la operación**.

Por último, **se crea un Timer** encargados de ejecutar una tarea de forma síncrona, cada segundo, que se encarga de **actualizar la lista de dispositivos detectados**. De forma complementaria, **se crea otro Timer** con su tarea correspondiente que, en este caso, se ejecuta cada 20 segundos y **limpia la lista con el fin de eliminar posibles**

dispositivos que han dejado de existir en la red (ya no transmiten su paquete de reconocimiento).

Pantalla2.xaml/xaml.cs:

Estos archivos comprenden a la **pantalla de monitorización de datos a tiempo real**.

El **aspecto visual** de la pantalla recoge **6 gráfica de una sola columna cada una**, las cuales **representan a cada uno de los parámetros ambientales** y, muestran el valor de ellos a tiempo real. Además, se incluye **una tabla que representa los mismos datos recibidos, pero en modo texto**, el campo fecha permite comprobar la correcta recepción de datos.

Como **resumen del código** de esta pantalla, hay que destacar la **inicialización de las gráficas** y la **creación de un Timer** encargado de ejecutar una tarea que **actualiza o refresca (redibuja) los valores recibidos de cada gráfica**.

Pantalla3.xaml/xaml.cs:

Este par de archivos conforman la **pantalla que representa los registros históricos** de los parámetros ambientales.

Gráficamente, esta pantalla contiene una **gráfica lineal** de dos ejes en la que se muestra el **rango de valores que toma un parámetro ambiental**, dentro del **periodo de tiempo** seleccionado mediante los botones y *switches* inferiores.

El mayor contenido de **código** de esta pantalla está asociado a los **eventos de pulsación de los controles (botones y switches)** los cuales son simples pero abundantes ya que se implementa la gestión de pulsación de uno de ellos y la desactivación de los otros en función del botón/*switch* presionado (para conseguir que cuando el usuario seleccione, por ejemplo, el parámetro humedad encontrándose con el botón de temperatura activo, este último se desactive automáticamente; al igual con los *switches* y los rangos de tiempo a representar).

Lo más destacado es la **implementación de una clase CGrafica** para controlar de forma cómoda la gráfica lineal, se implementan métodos para el dibujado, cambiar el rango de los ejes, la ocultación o visibilidad de estos y establecer el color de la línea.

Con la clase anterior, se puede interaccionar de forma simple con la gráfica y, para ello, **se crea un Timer** que ejecuta la función de **redibujado de los valores de la gráfica**. Este redibujado tomará los valores del parámetro que esté seleccionado en los botones inferiores, además del rango establecido con los *Switches*.

Lo último destacable de esta pantalla, es su **capacidad de exportar los Logs**, para ello, se presenta un botón de exportar que cuando el usuario lo presiona se encarga de lanzar el llamado *PickSaveFile* que permite a las aplicaciones UWP guardar un archivo. La creación de este nuevo archivo “exportado” se implementa creando un archivo vacío

y reemplazando el mismo a partir de los Logs exportados que se encuentran en una carpeta del sistema (los Logs se exportan cada vez que se selecciona una nueva combinación de parámetro y rango temporal a visualizar en esta gráfica, con los controles inferiores de la pantalla).

Pantalla4.xaml/xaml.cs:

Esta pantalla muestra los **ajustes o configuraciones** que se pueden establecer en relación a la **comunicación de red**.

La **interfaz de usuario** de esta pantalla permite **establecer los puertos** para las comunicaciones UDP y TCP (que por defecto se presentan como 3333 y 6666 respectivamente). Para ello se deben de introducir el nuevo número de puerto y seguido a eso presionar el botón de establecer correspondiente. En la parte inferior, se puede **seleccionar el adaptador de red a utilizar** si el por defecto (tarjeta inalámbrica) o establecer uno de forma manual, esto se selecciona con los *Switches* y, en el caso de querer seleccionar uno manualmente, se utiliza el *Combobox*.

Al establecer un nuevo número de puerto lo que se hace en el **código** simplemente es **cambiar el valor de la variable**, interna a la clase *ComunicacionRed*, correspondiente.

Otro aspecto destacable de código es la **implementación de una clase auxiliar LocalHostItem**, que **permite determinar los adaptadores de red disponibles en el equipo**. Esta clase es usada para representar en el ComboBox cada uno de los adaptadores y, en el momento en el que se selecciona de forma manual alguno, se establece dicho adaptador en la variable correspondiente, externa a esta pantalla.

Pantalla5.xaml/xaml.cs:

En esta última pantalla, como ya se comentó, se muestra información relacionada con el presente proyecto METEO, por lo que **se compone exclusivamente de texto informativo** (cosas tales como el contexto de esta aplicación universal, el nombre del desarrollador, la versión, la fecha de desarrollo y la finalidad del proyecto) y no hay ningún otro elemento de interacción posible para el usuario.

Otros Aspectos:

Hay que comentar que, para poder navegar mediante las pantallas, por defecto los datos de éstas se crean y se destruyen conforme se cambia de una a otra pantalla, para evitar esto y conseguir que las cosas permanezcan (y evitar también que deban ser recargadas cada vez que se accede a una de las pantallas) se recurre al método de páginas cacheadas. Para establecer que una página debe ser almacenada en cache y no eliminada, se debe activar la propiedad “**NavigationCacheMode**” del archivo XAML de cada página. Por ejemplo, para la Pantalla1:



```
<Page
    x:Class="MeteoPanel.Pantalla1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MeteoPanel"
    xmlns:d="http://schemas.microsoft.com/expressionblend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    NavigationCacheMode="Enabled">
```

Las aplicaciones universales buscan dotar de una **mayor seguridad en los dispositivos**, y para ello, ha adoptado un sistema basado en **permisos** (muy típicos de sistemas como *Android*). Por este motivo, para que nuestra aplicación pueda ser capaz de utilizar los servicios de red, para la comunicación UDP y TCP mediante Sockets, se deben de activar dichos permisos, los cuales serán notificados al usuario cuando vaya a realizar la instalación de la aplicación.

A continuación, procedemos a activarlos, debemos de acceder al archivo del proyecto **Package.appxmanifest**, una vez dentro, en la pestaña **Capabilities**, se deben de activar las capacidades de Internet y de redes privadas: **Internet (Client and Server)** y **Private Networks (Client and Server)**, como puede observarse en la figura 77.

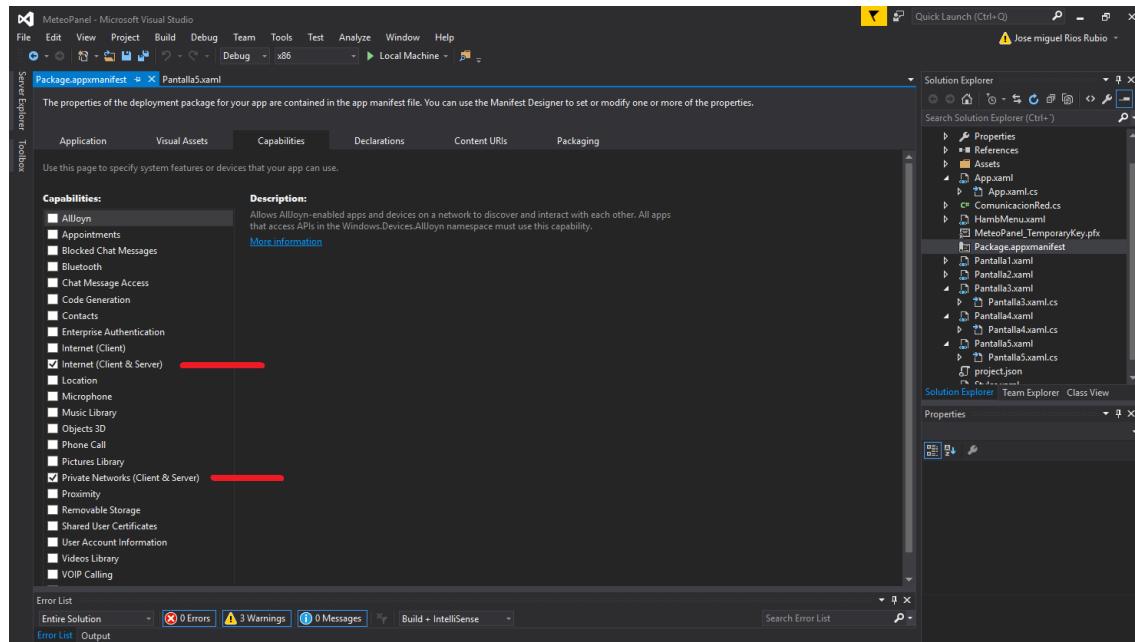


Fig. 77. Propiedades de la aplicación (archivo *Manifest*). Activando servicios de red

REFERENCIAS

- [1] Intel Corporation, «Intel Edison Board Product Brief,» 2014. [En línea]. Available: http://www.intel.com/content/dam/support/us/en/documents/edison/sb/edison_pb_331179002.pdf. [Último acceso: 2016].
- [2] Intel Corporation, «Intel Edison Arduino Breakout Board Hardware Guide,» 2014. [En línea]. Available: http://www.intel.com/content/dam/support/us/en/documents/edison/sb/edisonarduino_hg_331191007.pdf. [Último acceso: 2016].
- [3] Intel Corporation, «Intel Development IDEs,» [En línea]. Available: <https://software.intel.com/en-us/iot/tools-ide/ide>. [Último acceso: 2016].
- [4] Linux Foundation, «Yocto Project,» [En línea]. Available: <https://www.yoctoproject.org/>. [Último acceso: 2016].
- [5] NXP Semiconductors, «Datasheet MPL3115A2,» [En línea]. Available: <http://www.nxp.com/assets/documents/data/en/data-sheets/MPL3115A2.pdf>. [Último acceso: 2016].
- [6] Measurement Specialties, «Datasheet HTU21D,» [En línea]. Available: <http://www.datasheet.es/PDF/779951/HTU21D-pdf.html>. [Último acceso: 2016].
- [7] Everlight, «Datasheet ALS-PT19,» [En línea]. Available: <https://cdn-shop.adafruit.com/product-files/2748/2748+datasheet.pdf>.
- [8] Topway, «Datasheet LCD Topway LMB162H,» [En línea]. Available: <http://topwaydisplays.eu/documentos/004571-LNK04448.pdf>. [Último acceso: 2016].
- [9] Microsoft, «Conoce la Universal Windows Platform (UWP),» 11 5 2015. [En línea]. Available: <https://blogs.msdn.microsoft.com/esmsdn/2015/05/11/conoce-la-universal-windows-platform-uwp/>. [Último acceso: 2016].
- [10] Sparkfun , «Esquemático Wheather Shield,» [En línea]. Available: <http://cdn.sparkfun.com/datasheets/Dev/Arduino/Shields/Weather%20Shield.pdf>. [Último acceso: 2016].
- [11] Arduino CC, «Esquemático Arduino UNO Rev. 3,» [En línea]. Available: https://www.arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf. [Último acceso: 2016].

- [12] Intel Corporation, «Esquemático Intel Galileo Gen. 2,» [En línea]. Available: <http://www.intel.es/content/www/es/es/embedded/products/galileo/galileo-g2-schematic.html>. [Último acceso: 2016].
- [13] ON Semiconductor, «Datasheet NCP1117,» [En línea]. Available: http://www.onsemi.com/pub_link/Collateral/NCP1117-D.PDF. [Último acceso: 2016].
- [14] Intel Corporation, «Esquemático Intel Edison Arduino Breakout Board,» [En línea]. Available: http://www.intel.com/content/dam/support/us/en/documents/edison/sb/edison_arduino_hvm_8_26.pdf. [Último acceso: 2016].
- [15] Sunrom Technologies, «Datasheet Fotorresistencia Sunrom 3190,» [En línea]. Available: http://igem.org/wiki/images/1/1a/File-T--Technion_Israel-Hardwarespecsldr.pdf. [Último acceso: 2016].
- [16] Intel Corporation, «Librería MRAA,» [En línea]. Available: <https://github.com/intel-iot-devkit/mraa>. [Último acceso: 2016].
- [17] Intel Corporation, «Repositorio UPM,» [En línea]. Available: <https://github.com/intel-iot-devkit/upm>. [Último acceso: 2016].
- [18] M. M. (. d. e. S. T. Filip Skakun, «Github del proyecto WinRT XAML Toolkit,» [En línea]. Available: <https://github.com/xyzzer/WinRTXamlToolkit>. [Último acceso: 2016].

Pruebas

Descripción de las pruebas de verificación del funcionamiento individual de los subsistemas que forman la arquitectura y/o su correcta comunicación, así como de las pruebas de validación de los requisitos del sistema completo.

TÍTULO DOCUMENTO	Documento de pruebas de verificación y validación
FECHA DE ENTREGA	12/12/2016
AUTOR/ES	José Miguel Ríos Rubio

COMENTARIOS	VERSIÓN	FECHA
Versión Inicial	1.0	05/12/2016

ÍNDICE

1. Introducción
2. Pruebas de Verificación del Sistema
 - 2.1. Verificación de la Comunicación Serie
 - 2.2. Verificación de la Conexión/Desconexión
 - 2.3. Verificación de la Funcionalidad de la Pantalla LCD
 - 2.4. Verificación de la Funcionalidad de los Pulsadores
 - 2.5. Verificación de la lectura de parámetros ambientales
 - 2.6. Verificación de los registros de lecturas realizadas (Logs)
 - 2.7. Verificación de las comunicaciones por red
 - 2.8. Verificación de la aplicación de monitorización

METEO: Estación de monitorización meteorológica



1. Introducción

Con todo el sistema implementado, tanto el dispositivo hardware que realiza las mediciones, como la aplicación software que permite visualizarlas procedemos a validar su correcto funcionamiento y corroborar el cumplimiento de las especificaciones impuestas al proyecto.

2. Pruebas de Verificación del Sistema

Tanto para comprobar el correcto funcionamiento del dispositivo Hardware, como del sistema de monitorización (aplicación), se requiere el uso de ambos, ya que se encuentran intrínsecamente ligados y se comunican entre sí. Así, se verifica cada elemento del sistema, al mismo tiempo que se consigue validar el sistema completo.

De esta forma, se procede a **conectar el dispositivo hardware (Intel Edison) a un ordenador** a través de **dos conexiones USB**, una de ellas **para la alimentación del dispositivo** (consiguiendo que el sistema arranque), y la otra para la **comunicación del puerto serie**.

Una vez que el dispositivo haya arrancado, **ejecutamos en el ordenador el sistema de monitorización** realizado, la aplicación universal. El sistema de monitorización mostrará la pantalla de conexión/desconexión de dispositivos y, en ella, no aparecerá ningún dispositivo al que poder conectarse pues, aunque el dispositivo este alimentado, aún no hemos ejecutado su programa (y por ende no está transmitiendo nada en la red).

Procedemos ahora a **lanzar el programa desarrollado para el dispositivo** o, en su defecto, cargarlo en la placa a través del entorno de desarrollo (*Intel System Studio IoT Edition*).

2.1. Verificación de la Comunicación Serie

Lo primero que verificaremos será la **comunicación por el puerto serie**, para ello, **abrimos** algún cliente que permita abrir un terminal de comunicación con el puerto COM (en Windows) correspondiente al dispositivo, en este caso, usaremos **Putty**.

La **configuración para la conexión** se basa en **seleccionar comunicación serie**, establecer el **número de puerto COM** (que se puede consultar en el administrador de dispositivos de Windows) y **fijar la velocidad** de dicha comunicación, en este caso **115200 baudios**, tal como se muestra en la figura 1.

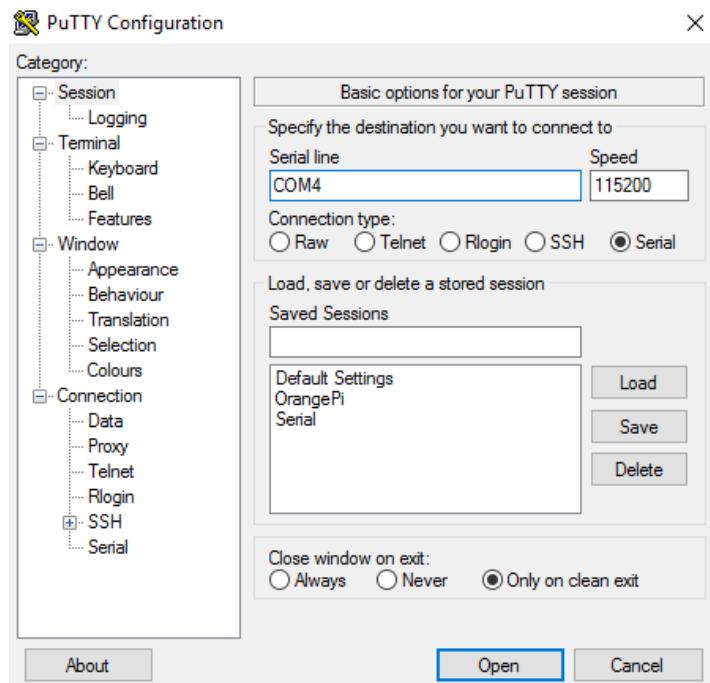


Fig. 1. Putty. Configuración de comunicación serie

Una vez abierto, podemos comprobar que se envían los mensajes de forma correcta por el puerto serie, el dispositivo notifica los envíos de los paquetes de identificación que se están enviando por red, el resultado de esto se muestra en la figura 2.

The screenshot shows the PuTTY terminal window titled 'COM4 - PuTTY'. The window displays a series of identical messages being sent over the serial port:

```

root@intel-edison:~#
root@intel-edison:~# /dev/ttyGS0Weather Shield
Dispositivo=1234;Direccion=192.168.1.128;

Dispositivo=1234;Direccion=192.168.1.128;

Dispositivo=1234;Direccion=192.168.1.128;

Dispositivo=1234;Direccion=192.168.1.128;

Dispositivo=1234;Direccion=192.168.1.128;

Dispositivo=1234;Direccion=192.168.1.128;

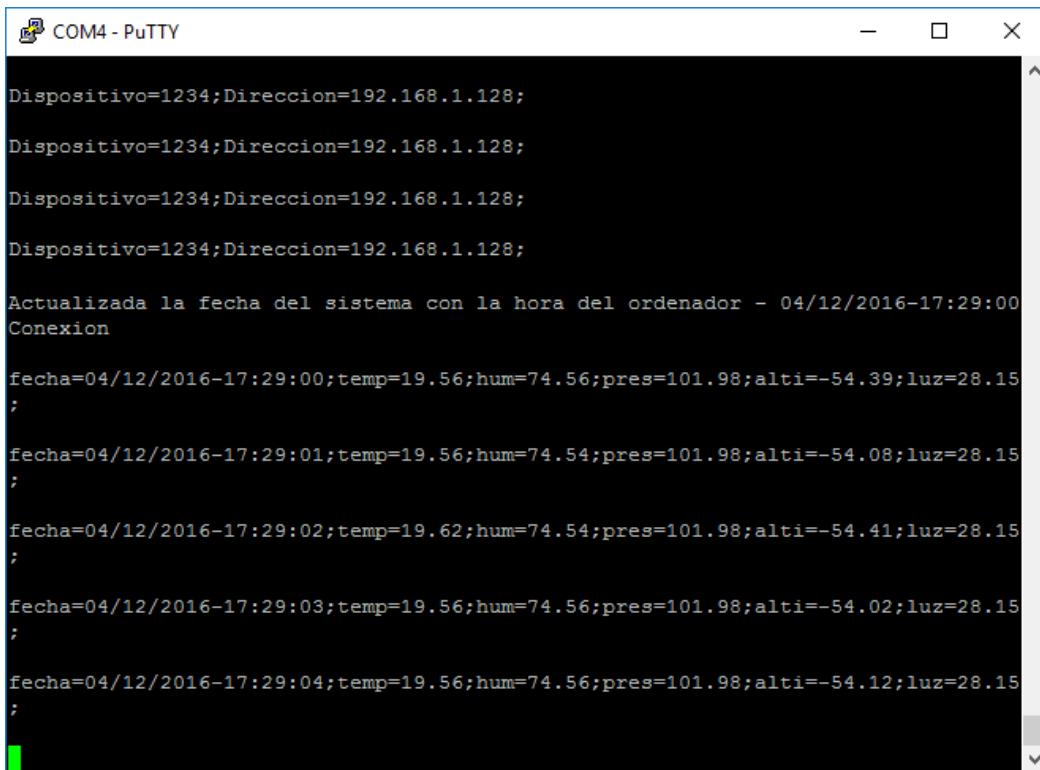
Dispositivo=1234;Direccion=192.168.1.128;

```

Fig. 2. Putty. Termina serie

2.2. Verificación de la Conexión/Desconexión

A continuación, comprobaremos la conexión del sistema de monitorización con el dispositivo. En primer lugar, con la aplicación de monitorización, se corrobora que el dispositivo está enviando los paquetes de reconocimiento para darse a conocer en la red, pues aparecen en la lista de dispositivos. Seleccionando al dispositivo de la lista y presionando el botón de conectar, podemos observar que el sistema se conecta; entre las cosas a tener en cuenta para esto, hay que destacar que por el puerto serie se notifica la conexión, y se aprecia que deja de enviar los paquetes de identificación y comienza a enviar paquetes de datos, como se puede ver en la figura 3; del mismo modo, tal y como aparece en la figura 4, el propio sistema de monitorización nos muestra, con el texto de estado inferior al botón de conectar, que se ha establecido exitosamente la conexión.



The screenshot shows a terminal window titled "COM4 - PuTTY". The window displays a series of text messages indicating a successful serial connection. The messages include device identification, system time synchronization, and periodic data transmission. The data transmitted includes temperature, humidity, pressure, altitude, and light levels.

```
Dispositivo=1234;Direccion=192.168.1.128;
Dispositivo=1234;Direccion=192.168.1.128;
Dispositivo=1234;Direccion=192.168.1.128;
Dispositivo=1234;Direccion=192.168.1.128;

Actualizada la fecha del sistema con la hora del ordenador - 04/12/2016-17:29:00
Conexion

fecha=04/12/2016-17:29:00,temp=19.56,hum=74.56,pres=101.98,alti=-54.39;luz=28.15
;

fecha=04/12/2016-17:29:01,temp=19.56,hum=74.54,pres=101.98,alti=-54.08;luz=28.15
;

fecha=04/12/2016-17:29:02,temp=19.62,hum=74.54,pres=101.98,alti=-54.41;luz=28.15
;

fecha=04/12/2016-17:29:03,temp=19.56,hum=74.56,pres=101.98,alti=-54.02;luz=28.15
;

fecha=04/12/2016-17:29:04,temp=19.56,hum=74.56,pres=101.98,alti=-54.12;luz=28.15
;
```

Fig. 3. Puerto serie. Conexión establecida



Fig. 4. Sistema de monitorización. Conexión establecida

Si ahora presionamos el botón de desconectar, se verifica que se realiza la desconexión correspondiente y es notificada, al igual que en la conexión, tanto por el puerto serie (y que vuelve a enviar paquetes de reconocimiento) como por la aplicación de monitorización. En la siguiente imagen, la figura 5, se muestra como la aplicación presenta el mensaje de estado.



Fig. 5. Sistema de monitorización. Conexión establecida

2.3. Verificación de la Funcionalidad de la Pantalla LCD



Fig. 6. LCD en funcionamiento

La verificación de la funcionalidad de la pantalla LCD es directa desde el primer momento en el que arranca el programa. Como se ve en la figura 6, una vez iniciado el sistema, en la pantalla LCD se muestra el texto correspondiente a la fecha del sistema y los parámetros ambientales que son leídos.

2.4. Verificación de la Funcionalidad de los Pulsadores

Para determinar el correcto funcionamiento de los pulsadores, comprobamos que la pulsación de cada uno de ellos es detectada por el sistema y cumple con su función correspondiente. Así, verificamos la funcionalidad de cada botón:

- **Pulsador de comutación de parámetros ambientales representados en el LCD:** Para verificar su funcionalidad basta con apreciar que en el LCD se muestran, por defecto, los parámetros de temperatura y humedad y que, al dejar presionado este pulsador, estos dos parámetros conmutan por los de presión y luz.
- **Pulsador de Desconexión:** Si el dispositivo se encuentra conectado, observando el puerto serie podemos apreciar como al presionar este botón se realiza la desconexión correspondiente del dispositivo con el sistema de monitorización.
- **Pulsador de eliminación de Logs:** Al igual que con el pulsador anterior, observando el puerto serie, podemos verificar que al presionar este botón el dispositivo procede a la eliminación de los archivos de Log internos, como puede verse en la figura 7. Además, si detenemos el programa y consultamos a mano

(mediante el comando `ls`) los archivos existentes podemos corroborar que efectivamente se han eliminado.

```
COM4 - PuTTY
Dispositivo=1234;Direccion=192.168.1.128;
Dispositivo=1234;Direccion=192.168.1.128;
Dispositivo=1234;Direccion=192.168.1.128;
Eliminando Logs...
Logs eliminados
Dispositivo=1234;Direccion=192.168.1.128;
Dispositivo=1234;Direccion=192.168.1.128;
Dispositivo=1234;Direccion=192.168.1.128;
Dispositivo=1234;Direccion=192.168.1.128;
```

Fig. 7. LCD en funcionamiento

2.5. Verificación de la lectura de parámetros ambientales

Una vez que el dispositivo se encuentra en funcionamiento, se puede verificar que se realizan lecturas correctas de cada uno de los parámetros ambientales. Esto puede verse a partir de la pantalla LCD o mediante la pantalla de la aplicación de monitorización, como se muestra en la siguiente imagen, la figura 8.

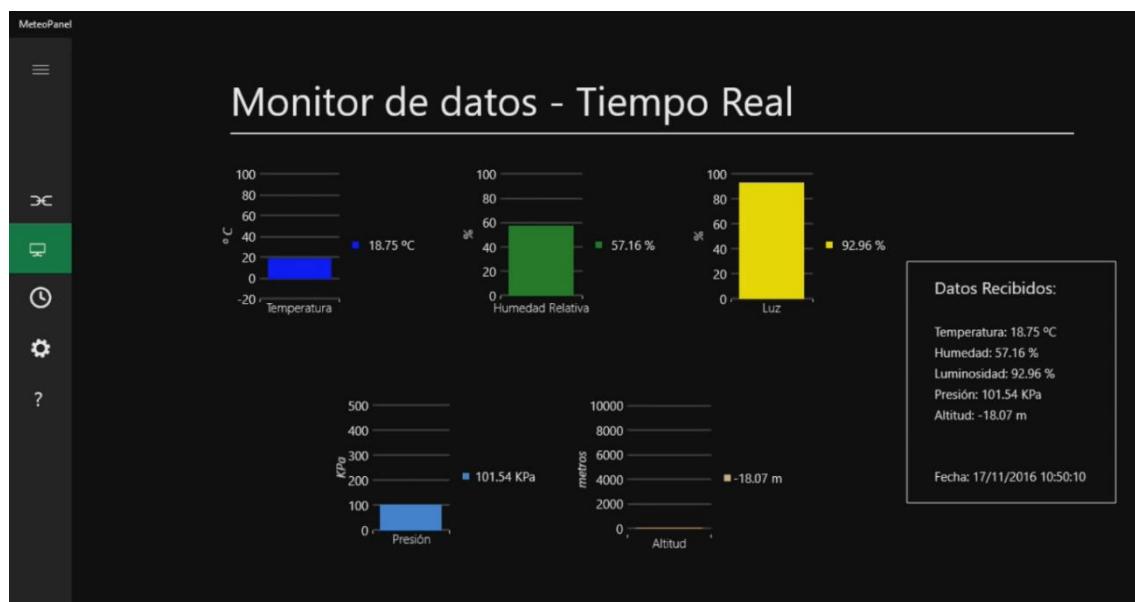


Fig. 8. Sistema de monitorización. Conexión establecida

Para **determinar si los valores de los parámetros ambientales leídos son correctos**, se recurre a **comprobar que estos se corresponden con la realidad**.

De esta forma, para el parámetro de **temperatura**, se comprueba que los valores leídos son adecuados, **comparando estas medidas con las proporcionadas por otros sistemas externos** (como el valor determinado por un sensor de temperatura con LCD). Además, durante todo el proceso de implementación del sistema se pudo comprobar que el valor de temperatura variaba de forma adecuada según el día y la hora.

Los valores del parámetro de **humedad relativa** son comprobados haciendo uso de una **aplicación de Smartphone** que recopila información ambiental de la plataforma **weather.com**; una captura de pantalla de esta aplicación es mostrada en la figura 9. Durante varios momentos del día comprobamos que los valores de humedad que muestra la aplicación y el valor determinado por nuestro sistema son cercanos y concluimos así que las lecturas obtenidas son correctas. Además, se puede observar la **variación de los valores** en la gráfica de la aplicación de monitorización, **cuando soplamos sobre el sensor**.



Fig. 9. Comparando la humedad relativa mediante App Android

Como bien se ha dicho, los **parámetros de presión atmosférica y altitud respecto al nivel del mar** están ligados (la altitud se calcula a partir del valor de presión) y, **esta prueba de verificación se basa en la posición geográfica en la que nos encontramos**. La prueba se realizó en un lugar cercano a la costa que se encuentra a unos escasos 200 metros, en línea recta, del mar por lo que se podría considerar que nos encontramos a nivel del mar, correspondiente a unos **100KPa y 0m de altitud**; los

errores cometidos por el sensor junto a la condición atmosférica actual juegan su papel y podemos apreciar un valor de presión atmosférica de 101.54KPa cercano a los 100KPa teóricos y la altitud correspondiente (a dicha presión) de $-18.07m$.

Los valores que se obtienen desde la **fotorresistencia** se corresponden al valor porcentual de la luminosidad ambiental. Para asegurarnos la correcta funcionalidad de la lectura del mismo, se procede a **iluminar, en mayor o menor medida, la fotorresistencia mediante una linterna**, de forma que se visualiza en la gráfica correspondiente estas variaciones del nivel de luminosidad. Así mismo en el caso de **tapar con la mano el sensor**, se puede apreciar como disminuye hasta casi 0 el valor de luz.

2.6. Verificación de los registros de lecturas realizadas (Logs)

Las lecturas de los sensores se van almacenando en los *Logs* internos del dispositivo. La pantalla de registro histórico de la aplicación de monitorización hace uso de los *Logs* y, por tanto, se puede determinar que, de manera general, estos son creados y se almacenan los valores en ellos de manera correcta. Al representar algún intervalo temporal se puede verificar que las muestras son variables y el número de ellas se corresponde con el intervalo fijado. El resultado de esta prueba se muestra en la figura 10.

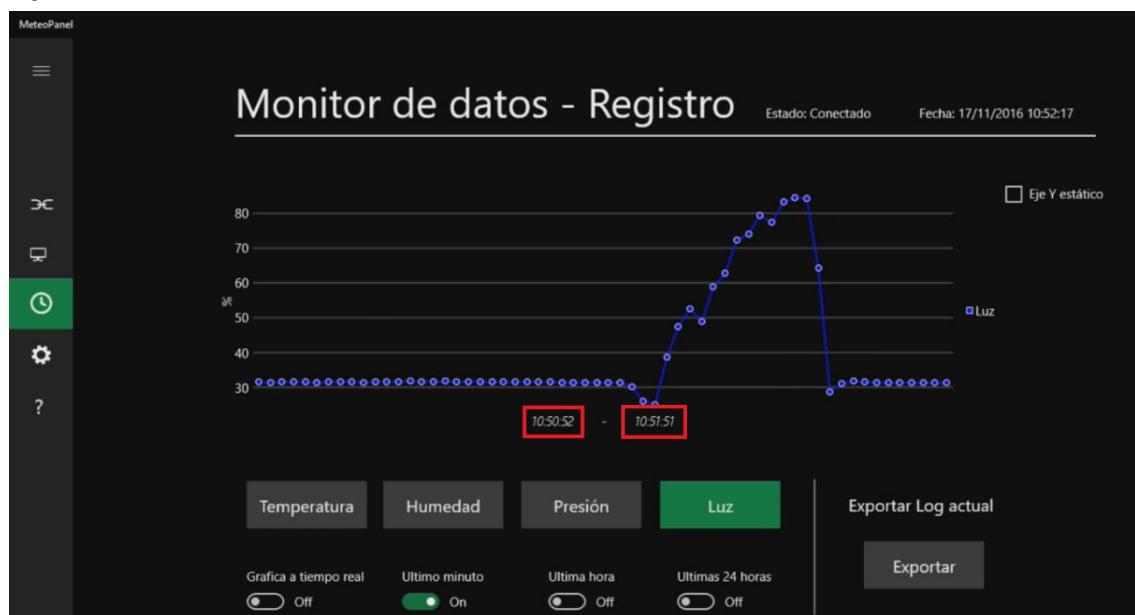
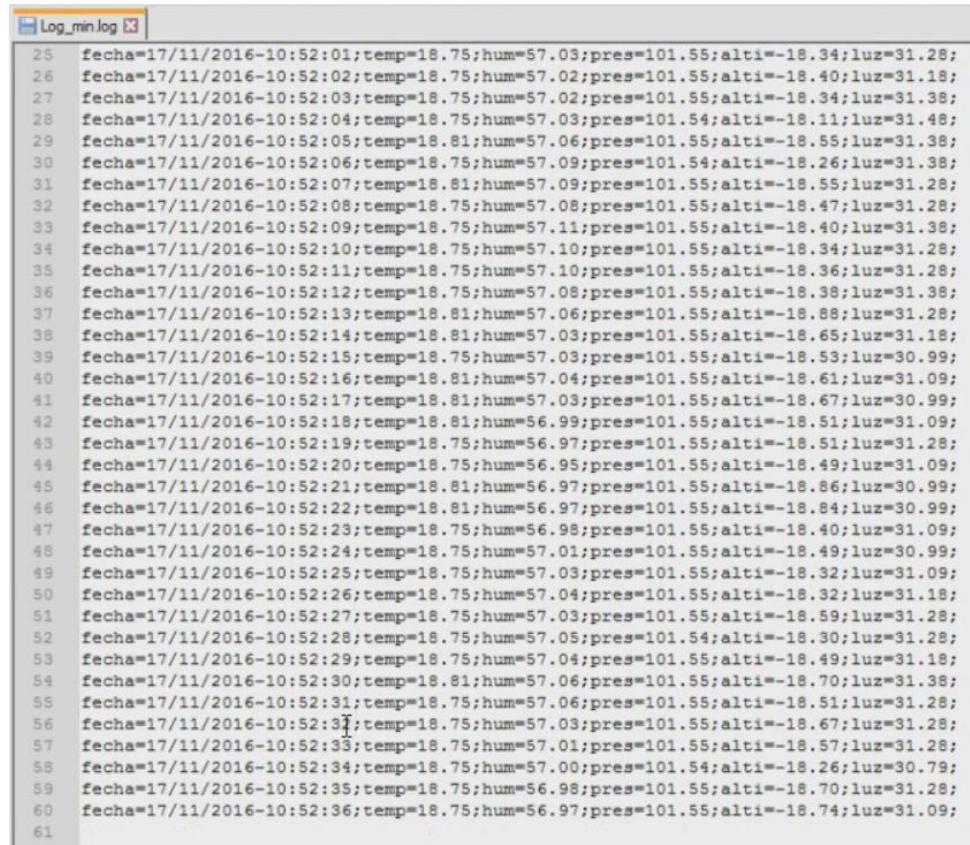


Fig. 10. Muestras almacenadas en el Log correspondiente al último minuto

Del mismo modo, la funcionalidad de exportar los *Logs* del sistema se puede verificar al presionar el botón correspondiente y, como se muestra en la figura 11, consultando el contenido del fichero exportado, se puede apreciar el correcto contenido de estos (con todas sus muestras).



```

Log_min.log
25 fecha=17/11/2016-10:52:01;temp=18.75;hum=57.03;pres=101.55;alti=-18.34;luz=31.28;
26 fecha=17/11/2016-10:52:02;temp=18.75;hum=57.02;pres=101.55;alti=-18.40;luz=31.18;
27 fecha=17/11/2016-10:52:03;temp=18.75;hum=57.02;pres=101.55;alti=-18.34;luz=31.38;
28 fecha=17/11/2016-10:52:04;temp=18.75;hum=57.03;pres=101.54;alti=-18.11;luz=31.48;
29 fecha=17/11/2016-10:52:05;temp=18.81;hum=57.06;pres=101.55;alti=-18.55;luz=31.38;
30 fecha=17/11/2016-10:52:06;temp=18.75;hum=57.09;pres=101.54;alti=-18.26;luz=31.38;
31 fecha=17/11/2016-10:52:07;temp=18.81;hum=57.09;pres=101.55;alti=-18.55;luz=31.28;
32 fecha=17/11/2016-10:52:08;temp=18.75;hum=57.08;pres=101.55;alti=-18.47;luz=31.28;
33 fecha=17/11/2016-10:52:09;temp=18.75;hum=57.11;pres=101.55;alti=-18.40;luz=31.38;
34 fecha=17/11/2016-10:52:10;temp=18.75;hum=57.10;pres=101.55;alti=-18.34;luz=31.28;
35 fecha=17/11/2016-10:52:11;temp=18.75;hum=57.10;pres=101.55;alti=-18.36;luz=31.28;
36 fecha=17/11/2016-10:52:12;temp=18.75;hum=57.08;pres=101.55;alti=-18.38;luz=31.38;
37 fecha=17/11/2016-10:52:13;temp=18.81;hum=57.06;pres=101.55;alti=-18.88;luz=31.28;
38 fecha=17/11/2016-10:52:14;temp=18.81;hum=57.03;pres=101.55;alti=-18.65;luz=31.18;
39 fecha=17/11/2016-10:52:15;temp=18.75;hum=57.03;pres=101.55;alti=-18.53;luz=30.99;
40 fecha=17/11/2016-10:52:16;temp=18.81;hum=57.04;pres=101.55;alti=-18.61;luz=31.09;
41 fecha=17/11/2016-10:52:17;temp=18.81;hum=57.03;pres=101.55;alti=-18.67;luz=30.99;
42 fecha=17/11/2016-10:52:18;temp=18.81;hum=56.99;pres=101.55;alti=-18.51;luz=31.09;
43 fecha=17/11/2016-10:52:19;temp=18.75;hum=56.97;pres=101.55;alti=-18.51;luz=31.28;
44 fecha=17/11/2016-10:52:20;temp=18.75;hum=56.95;pres=101.55;alti=-18.49;luz=31.09;
45 fecha=17/11/2016-10:52:21;temp=18.81;hum=56.97;pres=101.55;alti=-18.86;luz=30.99;
46 fecha=17/11/2016-10:52:22;temp=18.81;hum=56.97;pres=101.55;alti=-18.84;luz=30.99;
47 fecha=17/11/2016-10:52:23;temp=18.75;hum=56.98;pres=101.55;alti=-18.40;luz=31.09;
48 fecha=17/11/2016-10:52:24;temp=18.75;hum=57.01;pres=101.55;alti=-18.49;luz=30.99;
49 fecha=17/11/2016-10:52:25;temp=18.75;hum=57.03;pres=101.55;alti=-18.32;luz=31.09;
50 fecha=17/11/2016-10:52:26;temp=18.75;hum=57.04;pres=101.55;alti=-18.32;luz=31.18;
51 fecha=17/11/2016-10:52:27;temp=18.75;hum=57.03;pres=101.55;alti=-18.59;luz=31.28;
52 fecha=17/11/2016-10:52:28;temp=18.75;hum=57.05;pres=101.54;alti=-18.30;luz=31.28;
53 fecha=17/11/2016-10:52:29;temp=18.75;hum=57.04;pres=101.55;alti=-18.49;luz=31.18;
54 fecha=17/11/2016-10:52:30;temp=18.81;hum=57.06;pres=101.55;alti=-18.70;luz=31.38;
55 fecha=17/11/2016-10:52:31;temp=18.75;hum=57.06;pres=101.55;alti=-18.51;luz=31.28;
56 fecha=17/11/2016-10:52:32;temp=18.75;hum=57.03;pres=101.55;alti=-18.67;luz=31.28;
57 fecha=17/11/2016-10:52:33;temp=18.75;hum=57.01;pres=101.55;alti=-18.57;luz=31.28;
58 fecha=17/11/2016-10:52:34;temp=18.75;hum=57.00;pres=101.54;alti=-18.26;luz=30.79;
59 fecha=17/11/2016-10:52:35;temp=18.75;hum=56.98;pres=101.55;alti=-18.70;luz=31.28;
60 fecha=17/11/2016-10:52:36;temp=18.75;hum=56.97;pres=101.55;alti=-18.74;luz=31.09;
61

```

Fig. 11. Contenido del Log correspondiente al último minuto

2.7. Verificación de las comunicaciones por red

Evidentemente, vistas las pruebas anteriores, se puede asegurar que la comunicación de red es satisfactoria y su verificación se basa en la correcta transmisión/recepción de mensajes por parte del dispositivo y del sistema de monitorización (tanto con el protocolo UDP, como el TCP) que permiten al conjunto global del sistema cumplir con su funcionalidad principal.

2.8. Verificación de la aplicación de monitorización

La gran mayoría de los aspectos de la aplicación de monitorización pueden confirmarse repasando las pruebas anteriores: la conexión/desconexión con el dispositivo, la recepción/envío de mensajes por red y la representación de los datos en tiempo real y registrados.

METEO: Estación de monitorización meteorológica



Conclusiones y Posibles Mejoras

Documentación relacionada con el cierre del proyecto, incluyendo un estudio de las conclusiones obtenidas y manual de usuario de instalación si procede.

TÍTULO DOCUMENTO	Documento sobre conclusiones y posibles mejoras
FECHA DE ENTREGA	12/12/2016
AUTOR/ES	José Miguel Ríos Rubio

COMENTARIOS	VERSIÓN	FECHA
Versión Inicial	1.0	04/12/2016

ÍNDICE

1. Conclusiones Generales
2. Mejoras y Líneas Futuras

METEO: Estación de monitorización meteorológica



1. Conclusiones Generales

El resultado del proyecto ha sido satisfactorio en todos los sentidos, se han adquirido nuevos conocimientos y asentado las bases teóricas de otros.

Entre las cosas más características aprendidas se encuentran:

- El uso de las plataformas Intel Galileo/Edison utilizadas y las posibilidades de desarrollo que ofrecen.
- El desarrollo de Aplicaciones Universales mediante la nueva plataforma UWP de Microsoft.
- Comunicación por Red (protocolos UDP y TCP/IP).
- Programación de alto nivel mediante C#.

Además, tras la elaboración de este proyecto podemos sacar unas propias conclusiones al respecto de tanto el Hardware utilizado, como del software de desarrollo.

Las plataformas Intel Galileo/Edison son sistemas que cumplen con la función para la que se desarrollaron: permitir un **desarrollo rápido de productos IoT a medida**. No obstante, **su alto precio no compensa** para las prestaciones que ofrece y, ni siquiera en aquello para lo que se implementa se encuentra un gran soporte por parte de Intel, además de que **se reduce el soporte de estos productos para sacar nuevos dispositivos** de esta gama, como la nueva *Intel Joule*. La **compatibilidad** que debería presentar la *Intel Edison con la plataforma Arduino* presenta varios vacíos que no **están contemplados** (como la conexión de más de 2 dispositivos en un mismo bus I2C) y que, quedarán sin solución por parte de Intel.

La Plataforma Universal de Windows (UWP) busca unificar el desarrollo de todo tipo de software de modo que se comparta una misma base de código para la gran variedad de dispositivos que nos rodea. Aunque a mi parecer es una solución muy adecuada para conseguir estandarizar el desarrollo software, la plataforma no ha recibido buenas críticas, la mayoría de ellas basadas en que Microsoft busca imponer su solución sobre el resto de alternativas y monopolizar todo el desarrollo software, el aspecto más destacable de esta crítica es la plataforma de distribución de estas aplicaciones, pues Microsoft centraliza la distribución de estas exclusivamente desde su Tienda de Aplicaciones. Esta dura crítica es, junto a la escasa publicidad que se le ha dado a la plataforma, lo que en mi opinión hace que la UWP no despegue.

2. Mejoras y Líneas Futuras

El sistema desarrollado presenta bastantes líneas de ampliación y mejoras futuras que se pueden adoptar.

En una gran medida, el sistema desarrollado se caracteriza por estar hecho a medida y, como tal, presenta muchos aspectos desarrollados pensando exclusivamente en el entorno controlado de uso. Por ejemplo, la comunicación de red está centrada en el uso de una red local y, aunque algunos aspectos se han desarrollado con el fin de que en el futuro se pueda extender la funcionalidad de, por ejemplo, el número de dispositivos METEO que se encuentren en la red, actualmente el sistema solo soporta a un único dispositivo. Otro aspecto destacable es el hecho de que, si un sistema de monitorización se conecta a un dispositivo, este deja de estar accesible en la red, por lo que solo un cliente se puede conectar al mismo tiempo.

A pesar de todo esto, hay bastantes mejoras que se pueden implantar en el sistema, por ejemplo, los siguientes:

- Convertir el sistema en escalable y permitir el uso de más dispositivos en la red.
- Incrementar el número de parámetros ambientales a sensar.
- Hacer más eficiente la comunicación de red (mejorar el formato de los mensajes que se envían, codificar dichos mensajes...).
- Hacer accesible los datos adquiridos a Internet, de modo que se envíen y registren en alguna plataforma remota de *IoT* (como puede ser *Thingspeak*).
- Buscar una solución de menor consumo e implantar dispositivos a baterías.
- Desarrollar la carcasa del sistema y convertir el prototipo en un producto final.
- Implementar una interfaz web que cumpla la función de sistema de monitorización y permita acceder a los datos a través de un navegador.
- Incrementar el número de ajustes de configuración del sistema.
- Permitir exportar Logs independientes de cada parámetro ambiental, y no solo un Log global que lo contiene todo.

Anexo

Problemas en el Proceso de Implementación

Documentación relacionada con los problemas que han ido surgiendo durante el desarrollo del proyecto, sobre todo en la etapa de implementación del sistema.

TÍTULO DOCUMENTO	Anexo sobre los problemas surgidos durante el proceso de implementación
FECHA DE ENTREGA	12/12/2016
AUTOR/ES	José Miguel Ríos Rubio

COMENTARIOS	VERSIÓN	FECHA
Versión Inicial	1.0	01/12/2016

ÍNDICE

1. Introducción
2. El Hardware Inicial del Sistema
 - 2.1. Diferencias entre Intel Galileo e Intel Edison
 - 2.2. Shield LCD I2C
 - 2.3. El Problema con la Intel Galileo y sus Consecuencias
 - 2.4. Restaurando el Firmware de la Intel Galileo

METEO: Estación de monitorización meteorológica



1. Introducción

Cuando se comenzó el proyecto estaba fijado el uso de la plataforma Intel Galileo como elemento inteligente del sistema METEO, no obstante, ésta dejó de funcionar y se tuvo que migrar a la plataforma Intel Edison para finalizar el proyecto. Esta migración llevó consigo diversos cambios, tanto en el hardware del sistema, como en el software y los ajustes de los entornos de desarrollo utilizados.

En este anexo se recorre de forma rápida el desarrollo inicial que se hizo del sistema, con los elementos iniciales, los problemas que aparecieron y los cambios que se realizaron para superar estos contratiempos.

2. El Hardware Inicial del Sistema

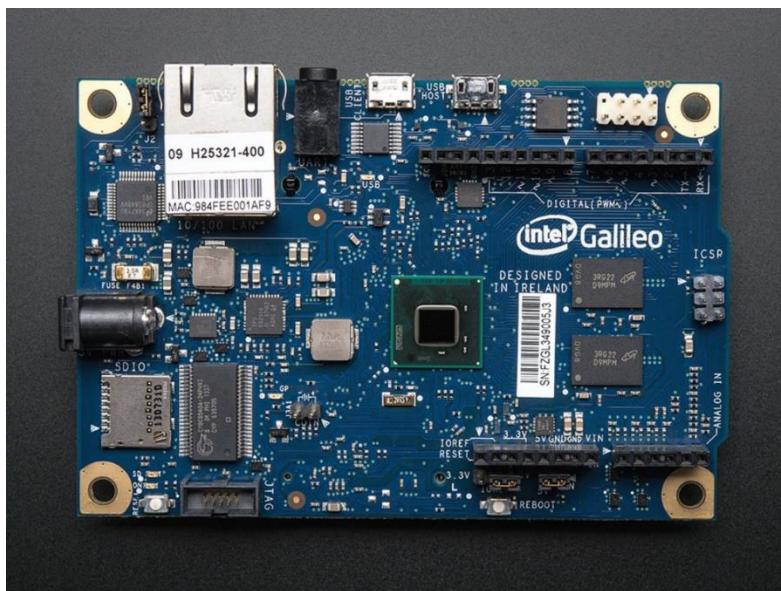


Fig. 1. Intel Galileo Gen 2 [1]

El proyecto comenzó con la placa **Intel Galileo Gen 2** (segunda generación), mostrada en la figura 1 [1], hermana menor de la Intel Edison. **Sin entrar en detalle sobre sus características, mencionaremos aquellas que la diferencian de la Intel Edison.**

2.1. Diferencias entre Intel Galileo e Intel Edison

La Intel Galileo **no presenta wifi integrado** (si se requiere de conexión wifi se debe de hacer uso de una tarjeta inalámbrica externa) sino que **presenta un puerto Ethernet**, el cual debe de utilizarse para conectar la placa a un *Router*, esto debido a que la programación de la misma se realiza a través de la red local.

La alimentación de la placa no puede ser por USB, como en la Edison, la Intel Galileo precisa de una **alimentación forzada a través del conector DC**, haciendo uso de una fuente de alimentación externa.

Otro aspecto destacable es que, en la Intel Galileo, **si se quiere afrontar el desarrollo avanzado** del sistema mediante el entorno *Intel System Studio IoT Edition*, **se debe de descargar una imagen de Linux más completa, y esta debe de grabarse en una tarjeta micro-sd que se inserta en la placa**.

Por último, Intel no ofrece una herramienta de configuración inicial de la placa para instalar los drivers, *flashear firmwares* o configurar el SSH, como la que existe para la Intel Edison, por ello, todos estos aspectos se deben de realizar de manera manual.

2.2. *Shield LCD I2C*



Fig. 2. Adafruit RGB LCD Shield

A parte de estos inconvenientes, los cuales provocan que la configuración inicial de la placa para poder empezar a desarrollar con ella sea más lenta que con la Intel Edison, la Intel Galileo presenta la ventaja de que **la compatibilidad hardware con la plataforma Arduino está mejor implementada**, esto es que, **no existe la incompatibilidad del pin V_{IN}**, que en la Intel Edison impedía conectar directamente la *Weather Shield* a la placa y nos obligaba a modificar las conexiones y tener que usar cables auxiliares, **ni el problema de conexión de componentes I2C a un mismo bus** o, al menos, no con el límite máximo de dos elementos, como pasaba con la Intel Edison.

Esto significa que, con la Intel Galileo, se podía utilizar un *Shield* específico con un LCD el cual presenta un controlador I2C (*RGB LCD Shield*, de *Adafruit*) que permite un

control simple del LCD a través de esta interfaz (sin la necesidad de la gran cantidad de cables que se utilizan en la interfaz paralela). Así, la conexión hardware del *Weather Shield* y este *Shield LCD* con la Intel Galileo permite una integración directa sin necesidad de cables (gracias a la disposición de pines adaptados a los *Shields Arduino*). En la figura 2 se muestra el *RGB LCD Shield* utilizado.

2.3. El Problema con la Intel Galileo y sus Consecuencias

El proceso de implementación del sistema mediante la Intel Galileo alcanzó el desarrollo completo del hardware y el software del dispositivo, y de una parte básica de la aplicación de monitorización (conexión y monitorización a tiempo real).

El problema surgió de la noche a la mañana, sin modificaciones hardware aparentes (aunque no completamente descartable que se pudiera producir algún cortocircuito inesperado mediante la manipulación de la placa cuando se guardaba y se extraía de la caja en la que se almacenaba), **una vez que se alimentó la placa notamos que no se podía establecer la conexión de red** para poder cargar el software que estábamos desarrollando, tras varios minutos de pruebas **pudimos confirmar que la placa no era detectada por el Router** (lo que quiere decir que hay un problema exclusivamente en ella) y, por tanto, procedímos a apagarla cuando **nos dimos cuenta que uno de los circuitos integrados de la placa estaba más caliente de lo normal**. Después de desconectar la alimentación y volver a conectarla apreciamos que no solo se calentaba el circuito, sino que salía un poco de humo de él, por lo que volvimos a desconectar la placa y procedimos a investigar al respecto.

La temperatura hizo que la serigrafía que se encontraba impresa sobre este integrado no fuera legible, por lo que no podíamos saber a qué se correspondía tal circuito, e Intel tampoco ofrece información al respecto. **Analizando el esquemático de la Intel Galileo, pudimos determinar que el circuito era el encargado de multiplexar las conexiones GPIO de la interfaz Arduino** de modo que establece si las señales son a 3.3V o a 5V, la señal de control del multiplexor se corresponde con el **jumper IOREF**, localizable en la figura 3 [20], de la placa y este **se encuentra en la configuración para los 5V**.

A continuación, **procedimos a cambiar dicho jumper de modo que se establecieran las señales de los GPIO a 3.3V**, hecho esto volvimos a arrancar la placa. El resultado fue que este circuito integrado no se calentara, sin embargo, **la placa seguía sin establecer conexión de red**.

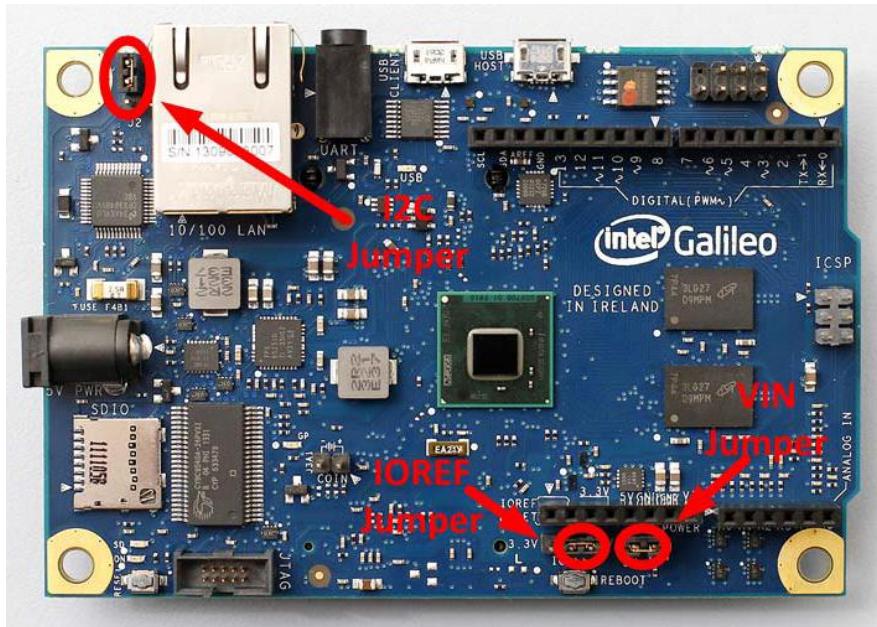


Fig. 3. Disposición de Jumpers de la placa Intel Galileo [20]

Cabe mencionar que, desde un principio, el SOC de la Intel Galileo tomaba una temperatura bastante alta, y aunque no se llegó a comprobar la temperatura exacta a través del software (para verificar si estaba en un rango seguro), era claramente apreciable cuando físicamente se manejaba la placa.

2.4. Restaurando el Firmware de la Intel Galileo

Puede ser que, en algún momento, debido a una mala conexión, un uso excesivo, un bajo voltaje de alimentación, una alta temperatura o cualquier otro factor; se corrompa el firmware de nuestra placa y nos la deje en un estado de “brick”, sin posibilidad de utilizarla, ni para cargarle un nuevo programa ni tan siquiera para acceder a ella mediante el puerto USB o el ethernet.

En nuestro caso habíamos conseguido que el integrado no se calentara, pero **el sistema no respondía como debía, la única solución posible era acceder a él mediante el puerto serie** asociado a la UART y comprobar si toda la placa se había estropeado, si había sido solo ese multiplexor y había afectado a algún circuito correspondiente a la conexión Ethernet, o simplemente que el firmware se ha corrompido. Para ello, necesitamos conectarnos a los pines correspondientes mediante un conversor Serie-USB, como se puede observar en la figura 4.

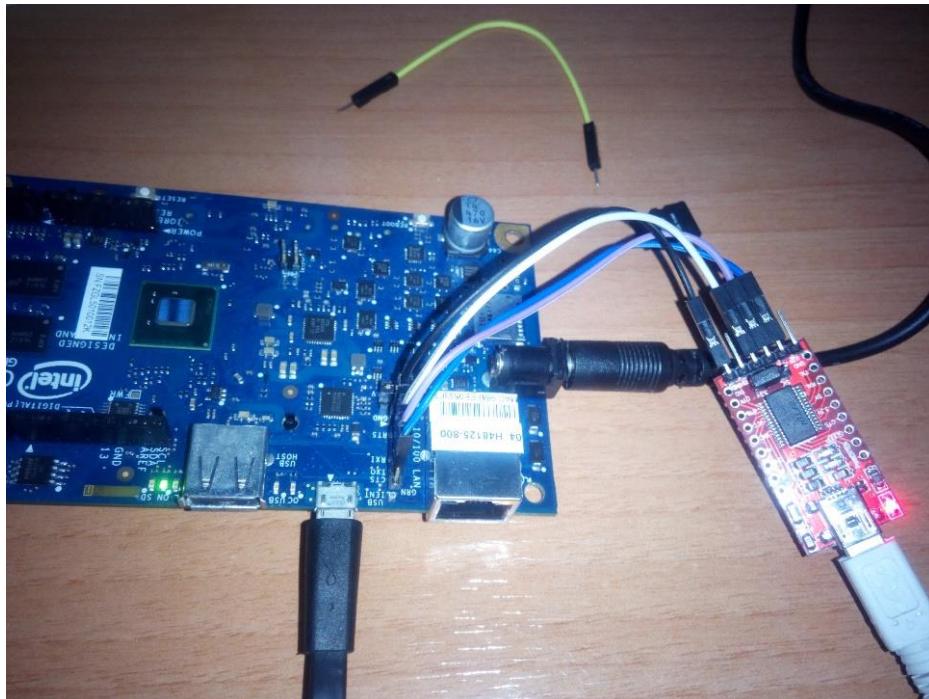


Fig. 4. Localización de los pines del puerto serie en la Intel Galileo

Primeramente, desconectamos las posibles conexiones del USB y alimentación de la placa, y extraemos la tarjeta SD. A continuación, **conectamos exclusivamente el cable de alimentación de la Intel Galileo**. Una vez encendida, **realizamos las conexiones del módulo serie-USB a los pines correspondientes de la placa**, tal y como se muestra en la figura 5.

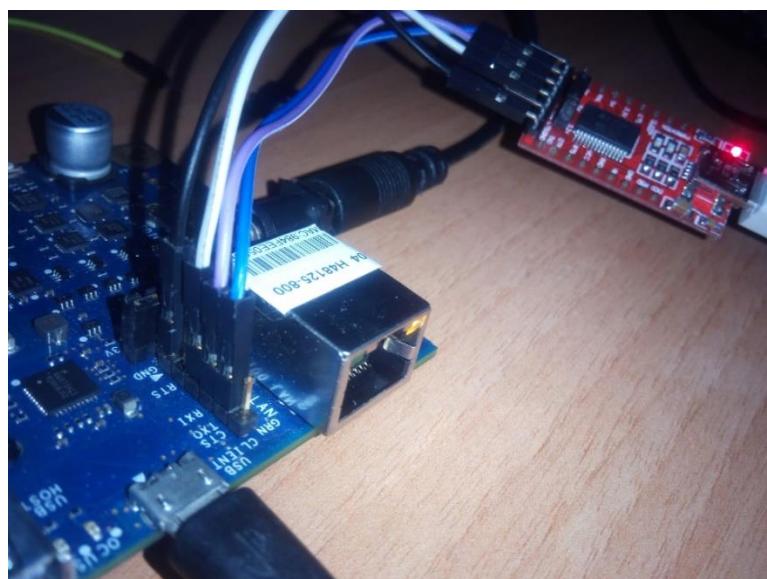


Fig. 5. Conexión al puerto serie de la Intel Galileo mediante un módulo usb-serie

Una vez conectado, **abrimos el programa Putty y nos conectamos al puerto COM** correspondiente, en este caso el **COM4**, a una velocidad de **115200 baudios**, como se puede ver en la figura 6.

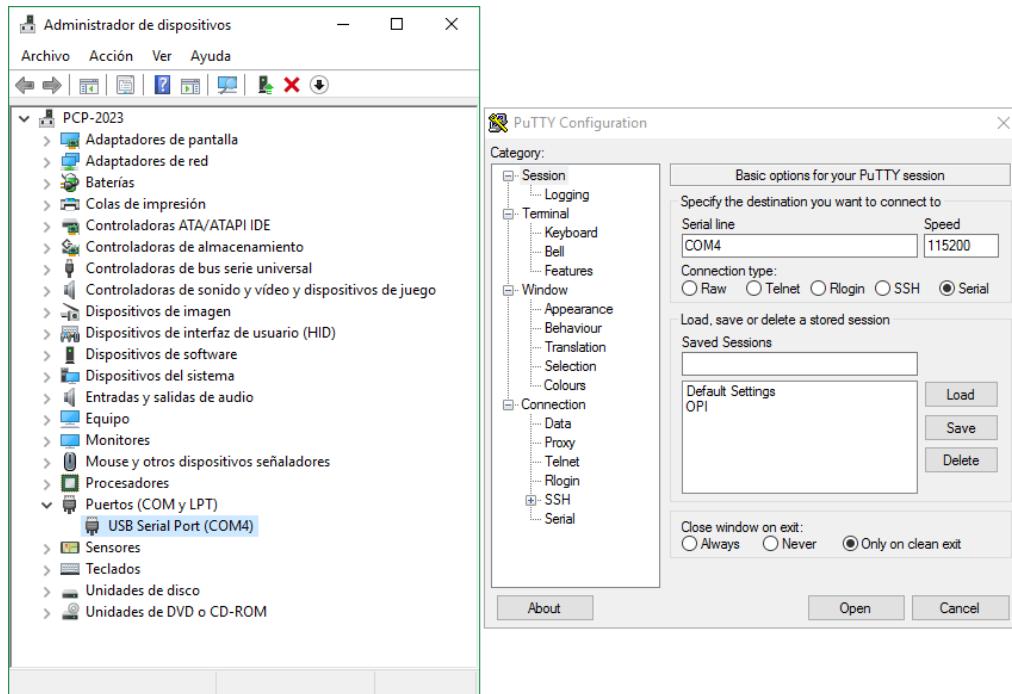


Fig. 6. Configuración de conexión serie en Putty

Al abrirse el terminal de conexión de Putty y presionar la tecla ENTER **podemos confirmar que el sistema de la placa falla al intentar arrancar debido a que el firmware se encuentra en un estado corrupto**: ASSERT_EFI_ERROR, tal y como se muestra en la figura 7.

```

COM4 - PuTTY
New boot attempt selected.....
ASSERT_EFI_ERROR (Status = Aborted)
*****
*****      ERROR: System boot failure!!!!!!
***** - Press 'R' if you wish to force system recovery
*****      (connect USB key with recovery module first)
***** - Press any other key to attempt another boot
*****

```

Fig. 7. Error de arranque. Firmware corrupto

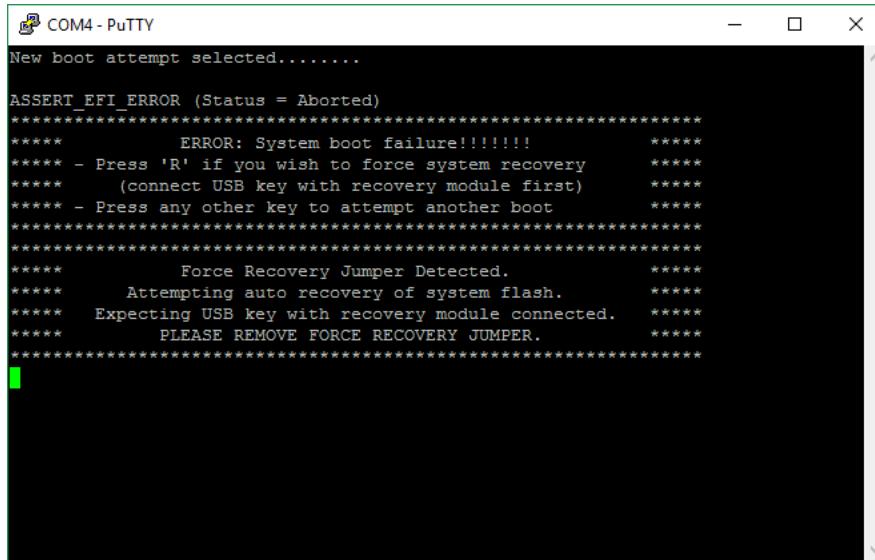
Para arreglar esto **se recurre al modo de recuperación** o “recovery”, para ello, en la placa Galileo Gen 2, y sin cerrar en ningún momento el terminal de Putty, **deben de seguirse los siguientes pasos:**

1. Descargar el archivo de recuperación FVMAIN.fv de la placa Galileo Gen 2:
<https://communities.intel.com/servlet/JiveServlet/download/245873-80879/FVMAIN.fv.zip>
2. Grabar el archivo en la raíz de un disco USB (pendrive) de formato FAT32.
3. Insertar el disco USB (pendrive) en el puerto USB de la Intel Galileo.
4. Quitar el cable de alimentación de la Intel Galileo.
5. Cortocircuitar a GND la resistencia mostrada en la siguiente imagen (figura 8):



Fig. 8. Cortocircuitando GND sobre la resistencia de recuperación

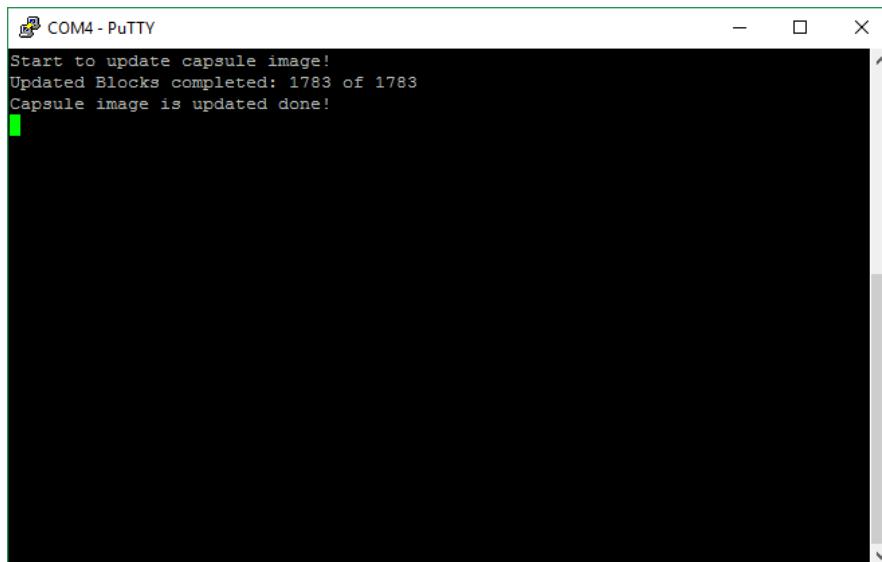
6. Volver a conectar el cable de alimentación de la Intel Galileo.
7. Esperar a que en el terminal se muestre que el sistema ha entrado en el modo de recuperación y que se retire la conexión a GND de la resistencia (figura 9).



```
COM4 - PuTTY
New boot attempt selected.....
ASSERT_EFI_ERROR (Status = Aborted)
*****
      ERROR: System boot failure!!!!!
*****
- Press 'R' if you wish to force system recovery
      (connect USB key with recovery module first)
*****
- Press any other key to attempt another boot
*****
*****
      Force Recovery Jumper Detected.
*****
      Attempting auto recovery of system flash.
*****
      Expecting USB key with recovery module connected.
*****
      PLEASE REMOVE FORCE RECOVERY JUMPER.
*****
```

Fig. 9. Entrando en el modo de recuperación

8. Rápidamente retirar el cortocircuito a GND de la resistencia y esperar a que termine el proceso de recuperación. El proceso puede tardar hasta 5 minutos (figura 10).



```
COM4 - PuTTY
Start to update capsule image!
Updated Blocks completed: 1783 of 1783
Capsule image is updated done!
```

Fig. 10. Proceso de recuperación finalizado

Completado el proceso de recuperación, ahora falta actualizar el firmware que hemos cargado (1.0.1) a la última versión (1.0.4), para ello utilizaremos la herramienta de actualización proporcionada por Intel; hay que seguir los siguientes pasos:

1. Descargamos la herramienta de actualización de firmware:
<https://downloadcenter.intel.com/download/24748/Intel-Galileo-Firmware-Updater-and-Drivers-1-0-4>
2. Conectamos la alimentación de la placa.
3. Conectamos por USB la placa al ordenador.
4. Abrimos la herramienta de actualización descargada (*firmware updater*).
5. Seleccionamos el puerto COM adecuado y presionamos el botón de actualizar firmware (figura 11).

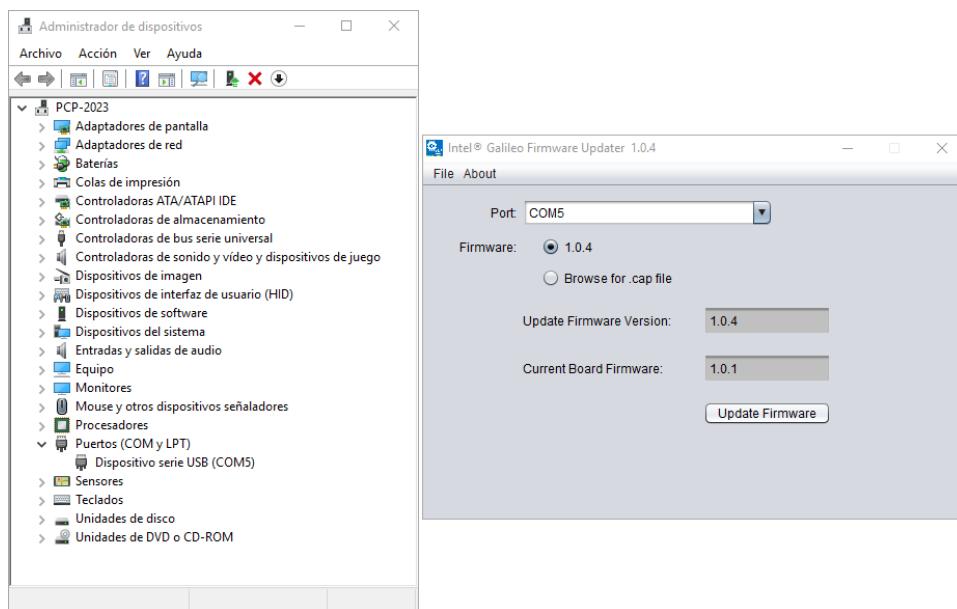


Fig. 11. Configurando el programa para actualizar firmware en Intel Galileo

6. Aceptamos las dos advertencias que nos muestra sobre mantener la alimentación de la placa y esperamos a que se complete el proceso, el cual puede tardar hasta 5 minutos (figura 12).

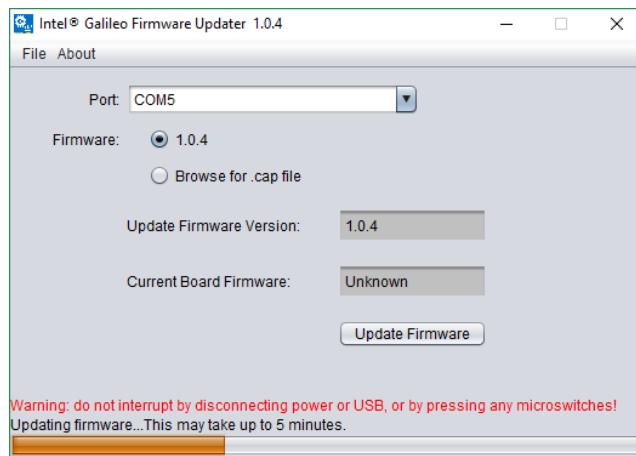


Fig. 12. Actualizando el firmware de la Intel Galileo

Una vez **completado el proceso** tenemos el firmware completamente recuperado y podemos acceder por red a la placa y volver a cargar programas en ella, no obstante, corroboramos que el circuito integrado que multiplexa las señales GPIO se ha dañado y ahora, la placa Intel Galileo no puede hacer uso de ella.

El resultado final es que, aunque el sistema vuelve a estar operativo, cuando se intenta acceder a algún elemento del GPIO el sistema, éste no lo detecta. Haciendo uso de la HAL de Arduino no se nos notifica de esto, pero cuando se utiliza la librería MRAA, ésta si es capaz de determinar que el GPIO no es reconocible por el SOC. Por tanto, el sistema en este estado no nos vale para nuestra aplicación y **por ello se recurrió a la Intel Edison**.

Referencias

- [1] Electronilab, «Electronilab,» [En línea]. Available: <http://electronilab.co/tienda/intel-galileo-board-de-desarrollo-arduino/>. [Último acceso: 2016].
- [2] Intel Corporation, «Guía de usuario de Intel Galileo,» [En línea]. Available: http://download.intel.com/support/galileo/sb/galileo_boarduserguide_330237_001.pdf. [Último acceso: 2016].