UNIVERSIDAD POLITÉCNICA DE CARTAGENA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES

DEPARTAMENTO DE TECNOLOGIA ELECTRÓNICA

PROYECTO FIN DE CARRERA

INGENIERO TÉCNICO INDUSTRIAL EN ELECTRONICA INDUSTRIAL

"ARQUITECTURA DE DISEÑO DE ACCESORIOS ELECTRÓNICOS INALÁMBRICOS PARA DISPOSITIVOS MÓVILES INTELIGENTES"

DIRECTORES: D. Juan Antonio López Riquelme

D. Andrés Iborra García

AUTOR: Juan Manuel Molera Codina

OCTUBRE 2013

AGRADECIMIENTOS

- A Juan Antonio por su ayuda y preocupación constante, por estar siempre disponible y sobre todo por la paciencia.
- A Andrés Iborra por ofrecerme una salida en un momento difícil.
- A mi padre, por agobiarme tanto con que acabara de una vez.
- A mi madre, por comprender que su taxi particular no estaba disponible durante ciertos momentos del desarrollo de este proyecto, gracias por tu paciencia.
- A mi hermana por su ayuda prestada en estos últimos y duros momentos.
- A toda mi familia que siempre ha estado pendiente de mis estudios.
- A ti Eli, que siempre estás ahí y que cada día me das tu apoyo, tu comprensión y fuerza para seguir adelante.
- A ti, que aunque por desgracia no puedas estar en este momento, siempre te he tenido muy presente.

ÍNDICE

Capítulo 1

Introducción

1.1 Introducción

Un dispositivo móvil inteligente o *Smartphone* es un teléfono móvil construido sobre una plataforma informática móvil, con una mayor capacidad de almacenar datos y realizar actividades semejantes a una mini computadora y conectividad que un teléfono móvil convencional. Estas características hacen que cada vez el uso de técnicas tradicionales (llamadas de voz y envío de mensajes de texto, entre otras) en estos terminales sea poco utilizada, y que estén destinados principalmente a la navegación Web, consulta del e-mail, realización de conferencias de vídeo, etc.

Lo comentado anteriormente se traduce en que los *Smartphones* son sin ninguna duda los dispositivos más utilizados en la actualidad y que estén reemplazando a más de un accesorio de la vida cotidiana, tales como los despertadores y las cámaras digitales. Este uso intensivo de estos dispositivos hace que exista una elevada demanda de accesorios electrónicos por parte de los usuarios.

Por lo expuestos anteriormente, resulta de interés que existan arquitecturas hardware y software que permitan llevar a cabo el diseño de accesorios electrónicos (*Gadgets*) para *Smartphones* de una forma rápida y sencilla. Por ello, este proyecto fin de carrera surge con el objetivo principal de proponer una arquitectura hardware y software que permita realizar un prototipado rápido y sencillo de accesorios electrónicos para dispositivos móviles inteligentes basados en el sistema operativo Android.

1.2 Objetivos

El principal objetivo de este Proyecto Fin de Carrera es proponer una arquitectura hardware y software que permita realizar un prototipado rápido y sencillo de accesorios electrónicos para dispositivos móviles inteligentes basados en el sistema operativo Android. Para conseguir este objetivo será necesario llevar a cabo los siguientes sub-objetivos:

- Buscar y seleccionar los kits de desarrollo necesarios que permitan conseguir una arquitectura para diseñar *Gadgets* para dispositivos móviles inteligentes.
- Estudiar, tanto a nivel hardware como software, los kits de desarrollo seleccionados.
- Diseñar los componentes hardware necesarios para implementar la arquitectura.
- Diseñar todos los componentes software necesarios para implementar la aplicación que permite interactuar con la arquitectura propuesta, y que es compatible con el sistema operativo Android.
- Realizar las pruebas necesarias para validar la arquitectura propuesta.

1.3 Fases del Proyecto

A fin de cumplir el objetivo anteriormente mencionado, se realizará la selección de los kits de desarrollo que permitan diseñar un prototipo que se pueda conectar de forma inalámbrica con dispositivos móviles inteligentes, basados en el sistema operativo Android. Dicho prototipo permitirá la conexión de sensores analógicos y digitales.

Una vez que se haya seleccionado el kit de desarrollo, se llevaran a cabo las tareas de diseño e implementación, tanto a nivel de hardware como de software, para conseguir la implementación de una arquitectura que permita diseñar diferentes *Gadgets* para dispositivos móviles inteligentes. El dispositivo se comunicará con el dispositivo móvil inteligente mediante una interfaz inalámbrica, además de que permitirá la conexión de sensores analógicos y digitales. Además, será necesario desarrollar una aplicación que sea compatible con el sistema operativo Android. Dicha aplicación permitirá configurar la lectura de los sensores del *Gadget*, así como obtener las lecturas recibidas del mismo.

Para llevar a cabo las tareas y objetivos descritos se establecen las siguientes fases:

- Búsqueda y selección de los kits de desarrollo que permitan conseguir una arquitectura para diseñar *Gadgets* para dispositivos móviles inteligentes.

- Estudio, tanto a nivel hardware como software, los kits de desarrollo seleccionados.
- Diseño de los componentes hardware necesarios para implementar la arquitectura.
- Diseño de los componentes software necesarios para implementar la aplicación compatible con el sistema operativo Android.
- Pruebas y puesta a punto.
- Redacción del documento final de memoria de proyecto

1.4 Desarrollo de la Memoria

1.4.1 Capítulo 1 - Introducción

El Capítulo 1 presenta la motivación de este trabajo, además de enumerar los objetivos del mismo y describir la memoria presentada.

1.4.2 Capítulo 2. Estado del Arte

En el Capítulo 2 se desarrolla un estudio de los diferentes kits de desarrollo existentes en el mercado, así como de las posibilidades de conexión de un *Smarthpone* con dichos kits, además de estudiar las diferentes formas de diseñar un *Gadget*.

1.4.3 Capítulo 3. Descripción del Sistema

En el tercer Capítulo se describe en profundidad el hardware de la arquitectura propuesta para el desarrollo de este proyecto, así como los programas software para su manejo y programación.

1.4.4 Capítulo 4. Descripción del Hardware y Software Desarrollado

En el cuarto Capítulo se describe en profundidad las conexiones hardware realizadas, así como el software necesario para las distintas partes hardware que componen este proyecto.

1.4.5 Capítulo 6. Conclusiones y Trabajos Futuros

En este capítulo se presentan las conclusiones obtenidas tras el desarrollo de este Proyecto Fin de Carrera. Por otro lado se mostraran las líneas futuras que podrían seguir trabajos asociados a este proyecto.

Capítulo 2

Estado del Arte

2.1 Introducción

En este capítulo se va a realizar una revisión de los diferentes kits de desarrollo existentes en el mercado, los cuales están formados por el hardware necesario para realizar el diseño de un *Gadget* para un *Smartphone/Tablet*. Además, también será objeto de este capítulo la descripción de los módulos software de dichos kits.

Otros elementos tratados en el capítulo son los sistemas operativos (S.O) utilizados en los *Smartphones/Tablets*, además de los diferentes medios disponibles para la comunicación entre los kits de desarrollo y los dispositivos electrónicos anteriormente mencionados.

A continuación se hará una breve descripción de *Smartphone*, *Tablet* y *Gadget*.

2.1.1 ¿Qué es un Smartphone?

Se puede decir que un Smartphone (del inglés *smart*: inteligente y *phone*: teléfono), es un teléfono móvil que te permite llevar a cabo acciones propias de una PDA (*Personal Digital Assistant*), más allá de lo fuera de lo común en todos los móviles, es decir, llamadas de voz y SMS (*Short Message Service*).

El desarrollo de la tecnología de la microelectrónica y de las redes de comunicaciones es lo que ha hecho posible su aparición. La potencia de cálculo de un *Smartphone* es comparable a la de un ordenador de escritorio o portátil, además deben de ser capaces de ejecutar un sistema operativo móvil completo e identificable. Este sistema operativo para móviles debe tener su propia plataforma de desarrollo de aplicaciones, y permitir que éstas tengan una mejor integración con el *software* base y el *hardware* del teléfono.

Los primeros *Smartphones* combinaron funciones de PDA con cámara de fotos y navegador GPS, pero ahora incluyen conexión a Internet vía Wi-Fi o red móvil para navegar por la web, vídeo-llamadas, visionado de correo electrónico (E-Mail), reproductor multimedia, etc.

2.1.1.1 Primer Smartphone de la Historia

El primer *Smartphone* de la historia fue el *IBM Simon* [1]. Fabricado en 1992 y distribuido en EE.UU entre agosto de 1994 y febrero de 1995, tenía un precio de 899 dólares, con una interfaz de usuario ausente de botones físicos y basada totalmente en una pantalla táctil de tipo LCD monocromo.

Ilustración 2-1: *IBM Simon.*

Disponía principalmente de texto predictivo, agenda, funciones de SMS, correo electrónico, busca (*beeper*), fax y un módem para conexión a Internet; estas funciones eran más comunes de una PDA que de un móvil de la época. Además, mostraba un teclado *Qwerty* en pantalla desde el cual se podía introducir el texto estándar o predictivo.

Fue un teléfono móvil revolucionario, diseñado y construido por una unión empresarial entre la *International Business Machines* y la *BellSouth Cellular Corporation*.

El *IBM Simon* incluía muchas aplicaciones útiles, como una libreta de direcciones, calendario con citas, agenda, calculadora, reloj mundial, bloc de notas electrónico y anotaciones manuscritas a mano alzada. Sin embargo, a diferencia de otros dispositivos similares no necesitaba ser operado por un lápiz o *stylus*, y bastaba con presionar con un dedo para acceder a sus funcionalidades.

El sistema operativo usado por el *Simon* era *ROM-DOS*, que tenía compatibilidad con *MS-DOS* y con la arquitectura x86, soportaba formato de archivos *FAT32*, tenía un procesador con un ciclo de reloj de 16 MHz con registros de 16 bits, capacidad de 1 Mega de RAM y 1 Mega de almacenamiento, además de incluir un modem telefónico integrado.

El *Simon* contaba con una ranura PCMCIA, con la que se podía instalar nuevas funcionalidades a partir de programas de terceros. La experiencia del usuario final era bastante buena, ya que el teléfono procesaba la información de una manera ágil y fluida. Hoy en día, algunos de sus descendientes, dotados de recursos más superiores, no son capaces de presumir de lo mismo.

El primer teléfono al que, realmente, se le dio el nombre de *Smartphone* fue el *Ericsson GS88*, apodado "Pamela", que fue desarrollado en 1997 por la casa *Ericsson*. Disponía del sistema operativo de 16 bits GEOS de *GeoWorks*, que fue el mismo que se adoptó en los Nokia 9000/9110. Traía de serie correo electrónico POP3, SMS, reloj mundial y navegador entre otros. Se podía poner en modo de vuelo desactivando todas las comunicaciones inalámbricas. Tenía manos libres integrado, modem integrado, puerto de infrarrojos, conexión al PC por medio de RS-232 y teclado *Qwerty* físico.

2.1.2 ¿Qué es una Tablet?

Se entiende por *Tablet* como la forma y funcionalidad de un nuevo dispositivo que tiene unas prestaciones muy similares a las de un ordenador o computadora pero que se representa en una sola pieza, sin necesidad de teclado físico, con un diseño plano, fino y compacto, el cual contiene todos los componentes esenciales para su funcionamiento de forma autónoma. Todos estos elementos están integrados en una sola pieza aparente que está compuesta por pantalla táctil (sencilla o multi-táctil), CPU, puertos y conectores, unidades de almacenamiento, etc.

El término puede aplicarse a una variedad de formatos que difieren en la posición de la pantalla con respecto a un teclado. El formato estándar se llama **pizarra (*Slate*)** y carece de teclado integrado, aunque puede conectarse a uno inalámbrico (*Bluetooth*) o mediante un cable USB. Otro formato es el **portátil convertible**, que dispone de un teclado físico que gira sobre una bisagra o se desliza debajo de la pantalla. Un tercer formato, denominado **híbrido**, dispone de un teclado físico, pero puede separarse de él para comportarse como una pizarra. Por último, los **booklets** incluyen dos pantallas, al menos una de ellas táctil, mostrando en ella un teclado virtual.

Los dispositivos *Tablet* revolucionan el concepto de movilidad por ser fácilmente portables y permitir estar conectados a Internet de forma permanente y prácticamente en

cualquier lugar, además de permitir la ejecución de un sin fin de aplicaciones, tanto locales como remotas.

2.1.3 ¿Qué es un Gadget?

Un *Gadget* o dispositivo electrónico es un elemento que tiene un propósito y una función específica, generalmente de pequeñas proporciones, práctico y novedoso. Muchos *Gadgets* suelen tener un diseño más ingenioso que el de la tecnología corriente.

2.2 Las Tecnologías de la Información y las Comunicaciones

2.2.1 Introducción

Ya en la era de la información actual, se puede afirmar que se han creado las tecnologías de la información y comunicación (TIC), como son Internet, la informática y las telecomunicaciones, cuya materia prima es la información, y que se han establecido en casi todos los sectores de la sociedad contribuyendo a la mejora de la misma. Hoy día los ordenadores, las telecomunicaciones e Internet han llegado a casi todos los hogares y las empresas, pudiéndose decir que la humanidad depende casi totalmente de las TICs para desarrollar sus labores de trabajo y ocio.

El crecimiento desmedido de las TICs (como puede ser el caso de Internet, cuyo número de servidores se dobla cada año) no sólo es una ventaja para nuestra sociedad, sino que también es un inconveniente. El inconveniente es un alto consumo de energía eléctrica y la contaminación producida al fabricar los equipos que integran las TICs.

2.2.2 Los Centros de Procesos de Datos

Los Centros de Procesos de Datos (CPD) o *Data Centers* son un edificio, nave o sala de gran tamaño para mantener dentro una gran cantidad de material electrónico, generalmente servidores. Suelen pertenecer a grandes organizaciones, como pueden ser empresas, universidades, administraciones del estado, entidades bancarias, para que las mismas puedan tener acceso a la información que necesiten para su funcionamiento.

Un CPD de tamaño estándar ocupa unos 43.600 m^2 de espacio y puede llegar a consumir 43 MW al año. Debido al espectacular crecimiento de Internet y a la aparición de las redes sociales, junto con los servicios de Google, han creado la necesidad de ampliar enormemente el número de servidores requeridos para poner en funcionamiento los

servicios de Internet. Los servidores necesarios para ofrecer todas las páginas web y servicios de Internet a los que los usuarios acceden diariamente, residen en los CPD que consumen el 50 % de la energía en el funcionamiento del propio CPD y un 50 % en el enfriamiento del mismo. Debido precisamente al aumento del número de usuarios de servicios de Internet los CPDs necesitan incorporar más y más servidores para satisfacer las necesidades de la sociedad, creciendo en tamaño y en consumo energético.

2.2.3 Virtualización

La virtualización es la tecnología que permite que en un sistema *hardware* corran diferentes sistemas *software*. El *software* contiene en su capa de virtualización unos hipervisores que son los encargados de lograr que varios sistemas o aplicaciones sean capaces de gestionar los recursos *hardware* que tienen debajo de forma eficiente, para que el volumen de carga que le ofrecen esas máquinas virtuales puedan ser asumidas por el nodo virtualizado. Con esta técnica se está consiguiendo que por ejemplo en los Centros de Procesos de Datos funcionen varios servidores en una misma máquina fisica, con lo que se reduce la cantidad de *hardware* necesario, reduciendo espacio y lo que es más importante, minimizando el consumo eléctrico necesario para su funcionamiento y refrigeración, que se traduce en un ahorro económico y un menor porcentaje de CO_2 emitido a la atmósfera.

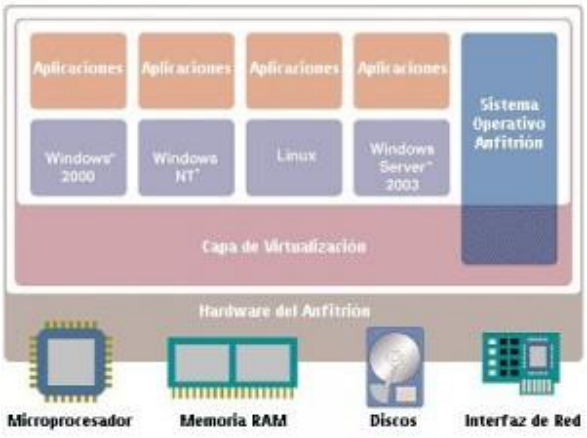


Ilustración 2-2: Esquema de Virtualización.

2.2.4 Computación en la Nube

A la computación en la nube se le conoce también como servicios en la nube, informática en la nube, nube de cómputo o nube de conceptos (*Cloud Computing*) y es un paradigma de reciente aparición que permite ofrecer a usuarios servicios de computación a través de Internet.

En la computación en la nube, todo lo que pueden ofrecer los sistemas informáticos se ofrece por servicio web. El servicio web es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de *software* desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. Con la computación en la nube, los usuarios

pueden acceder a cualquier servicio situado en la nube de Internet sin necesidad de tener conocimientos de los recursos que están utilizando.

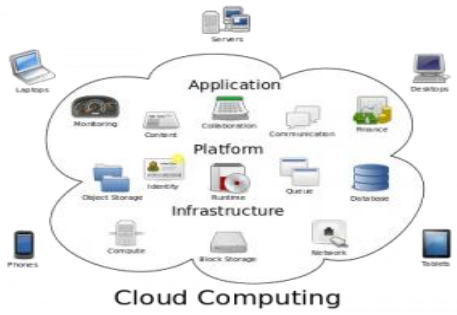


Ilustración 2-3: Esquema de computación en la nube.

La computación en la nube son servidores desde Internet encargados de atender las peticiones en cualquier momento. Se puede tener acceso a su información o servicio mediante una conexión a Internet desde cualquier dispositivo móvil o fijo ubicado en cualquier lugar. Sirven a sus usuarios desde varios proveedores de alojamiento repartidos frecuentemente también por todo el mundo.

Si las organizaciones utilizan la nube para el trabajo diario en sus oficinas, entonces necesitarán invertir menos dinero en servidores y grandes centros de datos propios, y se limitarán a utilizar los servicios que presta la nube. Si las organizaciones necesitan menos cantidad de *hardware* para su funcionamiento, entonces reducirán drásticamente su factura en electricidad y por consiguiente contribuirán a la mejora del medio ambiente emitiendo menos CO_2.

2.2.5 Computación en Clúster

Ilustración 2-4: Clúster de computadoras.

En la actualidad es común que se disponga de un conjunto de computadoras unidas por una red que son utilizadas para que trabajen en conjunto, compartiendo sus recursos (*hardware* y aplicaciones) y formando un clúster, con el objetivo de resolver un determinado problema matemático o científico que necesita una gran cantidad de ciclos de procesamiento. Algunas de las aplicaciones de la computación en clúster son para procesar algoritmos genéticos, simulación de líneas de fabricación, aplicaciones militares, bases de datos, inteligencia artificial, síntesis de imágenes, recuperación de imágenes por contenido, simulación de modelos del clima, análisis de

seísmos, algoritmos electromagnéticos, dinámica de fluidos, química cuántica, biomedicina, etc.

Es posible, aunque no necesariamente, que se utilice un sistema distribuido para resolver los problemas computacionales antes mencionados, de forma que las computadoras que trabajan conjuntamente para resolverlo estén conectadas, pero geográficamente dispersas las unas de las otras. Esta colaboración entre computadoras es transparente para el usuario que cree que se trata de un solo sistema.

La computación en clúster, ha resultado útil para resolver los problemas de gasto energético, puesto que son la base para implementar los llamados "servicios en la nube", que han sido descritos anteriormente.

2.2.6 Evolución de las Redes de Telefonía Móvil

La historia del teléfono móvil se remonta a los inicios de la Segunda Guerra Mundial, donde ya se veía que era necesaria la comunicación a distancia de un lugar a otro. Es por eso que la compañía Motorola creo un equipo llamado *Handie Talkie H12-16*, que es un equipo que permite el contacto con las tropas vía ondas de radio cuya, banda de frecuencias en ese tiempo no superaba los 600 KHz.

Comenzaron a perfeccionar y amoldar las características de este nuevo sistema revolucionario, ya que permitía comunicarse a distancia. Fue así que en los años 1980 se llegó a crear un equipo que ocupaba recursos similares a los *Handie Talkie* pero que iba destinado a personas que por lo general eran grandes empresarios y debían estar comunicados, momento en el que se crea el teléfono móvil y marca un hito en la historia de los componentes inalámbricos, ya que con este equipo se podía hablar a cualquier hora y en cualquier lugar.

2.2.6.1 Los inicios (0G): Los Pioneros

Los primeros sistemas de telefonía móvil civil empiezan a desarrollarse a partir de finales de los años 40 en los Estados Unidos. Eran sistemas de radios analógicas que utilizaba, en un primer momento, modulación en amplitud (AM) y posteriormente modulación en frecuencia (FM). Se popularizó el uso de sistemas FM gracias a una calidad superior de audio y una mejor resistencia a las interferencias. El servicio se daba en las bandas HF y VHF.

Los primeros equipos eran enormes y pesados, por lo que estaban destinados casi exclusivamente a su uso a bordo de vehículos. Generalmente se instalaba el equipo de radio en el maletero y se pasaba un cable con el teléfono hasta el salpicadero del coche.

Una de las compañías pioneras que se dedicaron a la explotación de este servicio fue la americana *Bell*. Su servicio móvil fue llamado *System Service*. No era un servicio popular porque era extremadamente caro, pero estuvo operando (con actualizaciones tecnológicas) desde 1946 hasta 1985.

2.2.6.2 Primera Generación (1G): Maduración de la Idea

En 1981 el fabricante *Ericsson* lanza el sistema NMT 450 (*Nordic Mobile Telephony* 450 MHz). Este sistema seguía utilizando canales de radio analógicos (frecuencias en torno a 450 MHz) con modulación en frecuencia (FM), y fue el primer sistema del mundo de telefonía móvil, tal como se entiende hasta hoy en día.

Los equipos de 1G pueden parecer algo aparatosos para los estándares actuales, pero fueron un gran avance para su época, ya que podían ser trasladados y utilizados por una única persona.

En 1986, *Ericsson* modernizó el sistema, llevándolo hasta el nivel NMT 900. Esta nueva versión funcionaba prácticamente igual que la anterior pero a frecuencias superiores (del orden de 900 MHz). Esto posibilitó dar servicio a un mayor número de usuarios y avanzar en la portabilidad de los terminales.

Además del sistema NMT, en los 80 se desarrollaron otros sistemas de telefonía móvil tales como AMPS (*Advanced Mobile Phone System*) en Estados Unidos y TACS (*Total Access Comunication System*). El sistema TACS se utilizó en España con el nombre comercial de *MoviLine* y estuvo en servicio hasta su extinción en 2003.

2.2.6.3 Segunda Generación (2G): Popularización

En la década de 1990 nace la segunda generación, que utiliza sistemas como GSM, IS-136, iDEN e IS-95. Las frecuencias utilizadas en Europa fueron de 900 y 1800 MHz.

El desarrollo de esta generación tiene como piedra angular la digitalización de las comunicaciones. Las comunicaciones digitales ofrecen una mejor calidad de voz que las analógicas, además se aumenta el nivel de seguridad y se simplifica la fabricación del terminal (con la reducción de costos que ello conlleva). En esta época nacen varios estándares de comunicaciones móviles: D-AMPS (EE.UU), *Personal Digital Cellular* (Japón), *cdmaOne* (EE.UU y Asia) y GSM.

Muchas operadoras telefónicas móviles implementaron acceso múltiple por división de tiempo (TDMA) y acceso múltiple por división de código (CDMA) sobre las redes AMPS existentes, convirtiéndolas así en redes D-AMPS. Esto trajo como ventaja para estas

empresas poder lograr una migración de señal analógica a señal digital sin tener que cambiar elementos como antenas, torres, cableado, etc. Inclusive, esta información digital se transmitía sobre los mismos canales (y por ende, frecuencias de radio) ya existentes y en uso por la red analógica. La gran diferencia es que con la tecnología digital se hizo posible hacer multiplexión, tal que en un canal antes destinado a transmitir una sola conversación a la vez se hizo posible transmitir varias conversaciones de manera simultánea, incrementando así la capacidad operativa y el número de usuarios que podían hacer uso de la red en una misma celda en un momento dado.

El estándar que ha universalizado la telefonía móvil ha sido GSM (*Global System for Mobile Communications*). Se trata de un estándar europeo nacido de los siguientes principios:

- Buena calidad de voz (gracias al procesado digital).
- Itinerancia (*Roaming*).
- Deseo de implantación internacional.
- Terminales realmente portátiles (de reducido peso y tamaño) a un precio asequible.
- Compatibilidad con la RDSI (Red Digital de Servicios Integrados).
- Instauración de un mercado competitivo con multitud de operadores y fabricantes.

GSM cumplió con todos sus objetivos, pero al cabo de un tiempo empezó a acercarse a la obsolescencia porque solo ofrecía un servicio de voz o datos a baja velocidad (9.6 Kb/s con WAP) y el mercado empezaba a requerir servicios multimedia que hacían necesario un aumento de la capacidad de transferencia de datos del sistema. Es en este momento cuando se empieza a gestar la idea de 3G, pero como la tecnología CDMA no estaba lo suficientemente madura en aquel momento se optó por dar un paso intermedio 2.5G.

2.2.6.4 Generación de Transición (2.5G)

Dado que la tecnología de 2G fue incrementada a 2.5G, se incluyeron nuevos servicios como EMS y MMS.

EMS es el servicio de mensajería mejorado, que permite la inclusión de melodías e iconos dentro del mensaje. En comparación con los SMSs, un EMS equivale a 3 ó 4 SMSs.

MMS (Sistema de Mensajería Multimedia) es un tipo de mensajes que se envían mediante GPRS y permite la inserción de imágenes, sonidos, videos y texto. Un MMS se envía en forma de diapositiva, en la cual cada plantilla sólo puede contener un archivo de cada tipo aceptado, es decir, sólo puede contener una imagen, un sonido y un texto en cada plantilla, si se desea agregar más de estos tendría que agregarse otra plantilla. Cabe mencionar que no es posible enviar un vídeo de más de 15 segundos de duración.

Para poder prestar estos nuevos servicios se hizo necesaria una mayor velocidad de transferencia de datos, que se hizo realidad con las tecnologías GPRS y EDGE.

GPRS (*General Packet Radio Service*) permite velocidades de datos desde 56 Kb/s hasta 114 Kb/s.

EDGE (*Enhanced Data rates for GSM Evolution*) permite velocidades de datos hasta 384 Kb/s.

Se empezó a tarificar por cantidad de datos enviados/recibidos, no por tiempo de conexión.

2.2.6.5 Tercera Generación (3G)

3G nace de la necesidad de aumentar la capacidad de transmisión de datos para poder ofrecer servicios como la conexión a Internet desde el móvil, la vídeo-conferencia, la televisión y la descarga de archivos. En este momento el desarrollo tecnológico ya posibilita un sistema totalmente nuevo: UMTS (*Universal Mobile Telecommunications System*).

UMTS utiliza la tecnología CDMA, lo cual le hace alcanzar velocidades realmente elevadas (de 144 Kb/s hasta 7,2 Mb/s, según las condiciones del terreno).

La tecnología HSDPA (*High Speed Downlink Packet Access*), tambien denominada 3.5G, 3G+ o turbo 3G, es la optimización de la tecnología espectral UMTS/WCDMA, incluida en las especificaciones de *3GPP release 5* y consiste en un nuevo canal compartido en el enlace descendente (*downlink*), que mejora significativamente la capacidad máxima de transferencia de información pudiéndose alcanzar tasas de bajada de hasta 14 Mbps (1,8, 3,6, 7,2 y 14,4 Mbps). Soporta tasas de *throughput* promedio cercanas a 1 Mbps. Actualmente, también está disponible la tecnología HSUPA (*High Speed Uplink Packet Access*), con velocidades de subida de hasta 5,8 Mbps, y HSPA+ con velocidades de hasta 84 Mbps de bajada y 22 Mbps en la subida.

El Sistema Universal de Telecomunicaciones Móviles (*Universal Mobile Telecommunications System* o UMTS) es una de las tecnologías usadas por los móviles de tercera generación, sucesora de GSM, debido a que la tecnología GSM propiamente dicha

no podía seguir un camino evolutivo para llegar a brindar servicios considerados de tercera generación.

UMTS ofrece los siguientes servicios:

- Facilidad de uso y bajos costes: UMTS proporciona servicios de uso fácil y adaptable para abordar las necesidades y preferencias de los usuarios, amplia gama de terminales para realizar un fácil acceso a los distintos servicios y bajo coste de los servicios para asegurar un mercado masivo, como el *roaming* internacional o la capacidad de ofrecer diferentes formas de tarificación.

- Nuevos y mejorados servicios: Servicios de voz, datos e información de alta calidad.

- Acceso rápido: La principal ventaja de UMTS sobre la segunda generación móvil (2G), es la capacidad de soportar altas velocidades de transmisión de datos de hasta 144 Kb/s sobre vehículos a gran velocidad, 384 Kb/s en espacios abiertos de extrarradios y 7,2 Mb/s con baja movilidad (interior de edificios). Esta capacidad sumada al soporte inherente del protocolo de Internet (IP), se combinan poderosamente para prestar servicios multimedia interactivos y nuevas aplicaciones de banda ancha, tales como servicios de vídeo telefonía y video conferencia y transmisión de audio y vídeo en tiempo real.

2.2.6.6 Cuarta Generación (4G): La Actualidad

La generación 4, o 4G es la evolución tecnológica que ofrece al usuario de telefonía móvil un mayor ancho de banda, que permite entre muchas otras cosas, la recepción de televisión en Alta Definición.

El sistema LTE (*Long Term Evolution*), es un nuevo estándar de la norma 3GPP. Lo novedoso de LTE es la interfaz radioeléctrica basada en OFDMA para el enlace descendente (DL) y SC-FDMA para el enlace ascendente (UL).

OFDMA (*Orthogonal Frecuency Division Multiple Access*), se utiliza para conseguir que un conjunto de usuarios de un sistema de telecomunicaciones puedan compartir el espectro de un cierto canal para aplicaciones de baja velocidad. El acceso múltiple se consigue dividiendo el canal en un conjunto de subportadoras (*subcarriers*) que se reparten en grupos en función de la necesidad de cada uno de los usuarios.

Para conseguir una mayor eficiencia, el sistema se realimenta con las condiciones del canal, adaptando continuamente el número de subportadoras asignadas al usuario en función de la velocidad que éste necesite y de las condiciones del canal. Si la asignación se hace rápidamente, se consigue cancelar de forma eficiente las interferencias co-canal y los desvanecimientos rápidos, proporcionado una mejor eficiencia espectral del sistema que OFDM.

El reciente aumento del uso de datos móviles y la aparición de nuevas aplicaciones y servicios como MMOG (Juegos Masivos Multijugador Online), televisión móvil, web 2.0, flujo de datos de contenidos, han sido las motivaciones por el que 3GPP desarrollase el proyecto LTE. Poco antes del año 2010, las redes UMTS llegan al 85% de los abonados de móviles. Es por eso que LTE 3GPP quiere garantizar la ventaja competitiva sobre otras tecnologías móviles. De esta manera, se diseña un sistema capaz de mejorar significativamente la experiencia del usuario con total movilidad, que utilice el protocolo de Internet (IP) para realizar cualquier tipo de tráfico de datos de extremo a extremo con una buena calidad de servicio (QoS) y, de igual forma el tráfico de voz, apoyado en Voz sobre IP (VoIP) que permite una mejor integración con otros servicios multimedia.

2.3 Arquitectura de los Smartphones

2.3.1 Introducción

La evolución del hardware de los *Smartphones* viene dada por la propia evolución en la fabricación de los circuitos integrados, y que viene siendo caracterizada por la Ley de Moore.

Las velocidades de procesamiento guardan una relación directa con el número de transistores incluidos sobre el chip, y cuanto más pequeño sea el transistor (proceso de producción menor), mayor cantidad de ellos podrá ser empaquetada dentro de un mismo chip.

2.3.2 Procesadores ARM

El diseño de los procesadores de los *Smartphones* está paralelizado con el desarrollo del concepto multi-núcleo y la disminución del proceso de fabricación en nanómetros, consiguiéndose que, a menor tamaño menor calor y menor consumo eléctrico, lo cual permite integrar un mayor número de ellos en el chip y se gana un mejor rendimiento.

El microprocesador es la parte más importante de cualquier equipo electrónico, y desde hace unos años la tendencia es duplicar, triplicar e incluso cuadriplicar el núcleo de

dicho microprocesador. Los sistemas operativos y el *software* que corre en ellos deberán de estar adaptados a esta tecnología para poder aprovecharla al máximo.

Arquitectura ARM: es una arquitectura RISC (*Reduced Instruction Set Computer*) de 32 bits desarrollada por *ARM Holding plc* (multinacional dedicada a los semiconductores y al desarrollo de *software*).

El 98 % de los microprocesadores utilizados en *Smartphones* hacen uso de la arquitectura de ARM. Diversas compañías (TI, *Qualcomm*, *Freescale*, *Samsung*, etc.) se encargan de plasmarlos en un chip, y los modifican en algunos aspectos para sacarle el máximo rendimiento y/o mejorar su consumo de energía y dedicarlos a un propósito específico.

2.3.3 System-on-chip (SoC, System-on-a-chip)

System-on-a-chip o SoC (también referido como *system-on-chip*, en español Sistema en un chip), describe la tendencia cada vez más frecuente, de usar tecnologías de fabricación que integran todos o gran parte de los componentes de un ordenador o cualquier otro sistema informático o electrónico en un único circuito integrado o chip. Este es un término de lo más común hoy día en los *Smartphones*, y por buenas razones, ya que el espacio en ellos es reducido.

Cuando se habla de los microprocesadores dentro de un *Smartphone* por lo general se refiere en realidad al SoC: una combinación que incluye cosas como el/los núcleos del procesador, el sistema de gráficos, memoria RAM y ROM, controladores de interfaz para USB, tecnología inalámbrica, reguladores de voltaje, etc. La idea SoC es que todos los componentes críticos de un dispositivo se encuentren en un área relativamente pequeña.

Compañías como *NVIDIA*, *Texas Instruments* y *Samsung* han entrado en la producción de los SoCs. Estas, y otras, toman (a través de licencias) la arquitectura ARM y su núcleo producido, y lo ponen dentro de sus chips en combinación con la GPU, memoria y todos los componentes que deseen.

2.4 Posibilidades de Kits de Desarrollo

2.4.1 Introducción

En este apartado se describirán brevemente los diferentes kits de desarrollo existentes en el mercado para la realización del diseño de un *Gadget*.

2.4.2 Electric Sheep

La placa *Electric Sheep* de *Sparkfun Electronics* [2] es una herramienta de desarrollo para crear aplicaciones y acessorios personalizados para Android. Esta placa se comunica con su dispositivo Android a través de USB mediante el aprovechamiento del protocolo "*Open Accessory*" de Android. Debido a la configuración del protocolo "*Open Accessory*", esta placa tiene que suministrar 500 mA al dispositivo Android a través de la conexión USB. Si se utiliza dispositivos periféricos se tiene que proporcionar más corriente a la entrada, soportando un máximo de 1,5 A.

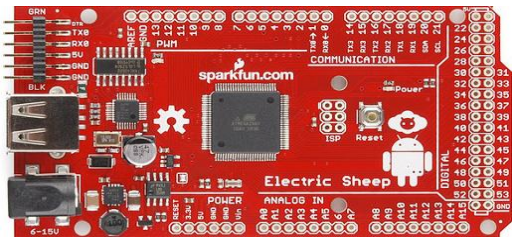


Ilustración 2-5: Placa de desarrollo *Electric Sheep.*

Las principales características de la placa de desarrollo *Electric Sheep* son las siguientes:

- Micro-controlador ATMEGA2560-16AU.
- 256 Kb Memoria Flash.
- 4 KB Memoria EEPROM.
- 8 KB Memoria RAM.
- Gran variedad de E/S analógicas, digitales, PWM, etc.
- Interfaz serie SPI Master/Slave.
- Voltaje de entrada: 6 – 15 V.
- Conector Jack para la alimentación.
- Conector USB.
- No posee módulo *Bluetooth* integrado.
- Dimensiones: 53,50 x 101,50 mm.

2.4.3 Microchip PIC24F Accessory Development Starter Kit

El kit de Microchip PIC24F [3] para el desarrollo de accesorios para Android es una placa utilizada para la evaluación y el desarrollo de accesorios electrónicos para el sistema

operativo Android de Google tanto para *Smartphones* como *Tablets*. El kit proporciona todas las herramientas y recursos necesarios para un inicio rápido en el desarrollo de accesorios para dispositivos Android. La plataforma proporciona una biblioteca para el acceso a los dispositivos Android a través del *framework* de las versiones Android 2.3.4, 3.1 y posterior.

Ilustración 2-6: Placa de desarrollo Microchip PIC24F.

Las principales características de la placa de desarrollo Microchip PIC24F son las siguientes:

- Microcontrolador PIC24F 16 – Bit.
- Dispone de botones, potenciómetros y Leds para la interfaz con el usuario.
- Conector Jack para la alimentación.
- Conector USB.
- No posee módulo *Bluetooth* integrado.

2.4.4 PIC32 Accessory Development Start Kit for Android

Este Kit es una 2º versión de Microchip para el desarrollo de aplicaciones para Android.

Ilustración 2-7: Placa de desarrollo Microchip PIC32

Las principales características de la placa de desarrollo Microchip PIC32 son las siguientes:

- Micro-controlador PIC32.
- Este kit posee una placa de expansión de entradas y salidas.
- Conector Jack para la alimentación.
- Conector USB.
- No posee módulo *Bluetooth* integrado.

2.4.5 Seeeduino ADK Main Board

La placa de desarrollo *Seeeduino ADK Main Board* [4] se comunica con su dispositivo Android a través de USB mediante el aprovechamiento del protocolo "*Open Accessory*" de Android. El *Seeeduino* es compatible con dispositivos Android v1.5 utilizando un micro-puente y con las v2.3.4 y superior con la API "*Open Accessory*" de Google.

Ilustración 2-8: Placa de desarrollo *Seeeduino*

Las principales características de la placa de desarrollo *Seeeduino* son las siguientes:

- Posee una enorme cantidad de entradas y salidas.
  - 56 I/O Digitales.
  - 16 Entradas Analógicas.
  - 14 Salidas PWM.
  - 4 UARTs.
- Voltaje de entrada: 6 – 18 V.
- Conector Jack para la alimentación.
- Conector USB.
- No posee módulo *Bluetooth* integrado.

2.4.6 PhoneDrone Board for Android

La placa *PhoneDrone* para Android permite conectar cualquier dispositivo Android (v2.3.4 y posterior) en el mundo de RC y UAVs. La placa tiene 8 canales de RC de entrada y salida, con la conversión PWM a PPM y el multiplexado entre RC y el control de Android.

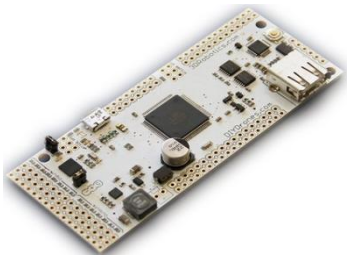


Ilustración 2-9: Placa de desarrollo *PhoneDrone.*

Las principales características de la placa de desarrollo *PhoneDrone* son las siguientes:

- Gran variedad de E/S analógicas, digitales, PWM, etc.
- Voltaje de entrada: 6 – 36 V.
- Conector USB.
- No posee módulo *Bluetooth* integrado.
- Dimensiones: 40,65 x 101,60 mm.

2.4.7 IOIO for Android

La placa IOIO [6] está diseñada especialmente para trabajar con dispositivos Android v1.5 y superior. La placa proporciona una sólida conexión a un dispositivo Android a través de una conexión USB o *Bluetooth,* y es totalmente controlable desde una aplicación Android usando una simple e intuitiva API sin necesidad de ningún tipo de programador externo.

Ilustración 2-10: Placa de desarrollo IOIO.

Las principales características de la placa de desarrollo IOIO son las siguientes:

- Microcontrolador PIC24F.

- Gran variedad de E/S analógicas, digitales, PWM, etc.
- Voltaje de entrada: 5 – 15 V.
- Conector USB.
- No posee módulo *Bluetooth* integrado, pero con un simple adaptador permite la conexión vía *Bluetooth*.

2.4.8 IOIO OTG

La placa de desarrollo IOIO OTG se diferencia de su antecesora en que la placa IOIO OTG implementa la norma USB On-The-Go, además de incluir un conector JST de 2 pines para la alimentación.

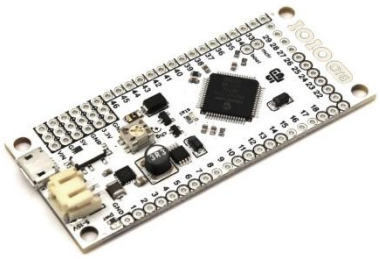


Ilustración 2-11: Placa IOIO OTG.

2.5 Posibilidades de Comunicación Smartphones

2.5.1 Introducción

Como se ha visto anteriormente, algunas de las placas de desarrollo permiten comunicación cableada (USB) e inalámbrica (*Bluetooth*). Por ello, en este apartado se describirán brevemente estos medios de comunicación.

2.5.2 USB

2.5.2.1 Definición

El Universal Serial Bus (USB) es un estándar industrial desarrollado a mediados de los años 1990 que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre ordenadores y periféricos y dispositivos electrónicos. La iniciativa del desarrollo partió de Intel, que creó el USB

Implementers Forum junto con IBM, *Northern Telecom*, *Compaq*, Microsoft, *Digital Equipment Corporation* y NEC. En 1996 se lanzó la primera especificación (USB 1.0), la cual no fue popular, hasta 1998 con (USB 1.1).

2.5.2.2 Usos

USB fue diseñado para estandarizar la conexión de periféricos, como ratones, teclados, memorias USB, joysticks, escáneres, cámaras digitales, teléfonos móviles, reproductores multimedia, impresoras, dispositivos multifuncionales, sistemas de adquisición de datos, módems, tarjetas de red, tarjetas de sonido, etc.

Su campo de aplicación se extiende en la actualidad a cualquier dispositivo electrónico o con componentes, se han implementado variaciones para su uso industrial e incluso militar. Pero en donde más se nota la influencia es en los teléfonos móviles (Europa ha creado una norma por la que todos los móviles deberán venir con un cargador microUSB).

2.5.2.3 Especificaciones

- Longitud: 5 metros máximo.
- Ancho: 11,5 mm (conector A), 8,45 mm (conector B).
- Alto: 4,5mm (conector A), 7,78 mm (conector B, antes de v3.0).
- Voltaje máximo: 5 V.
- Corriente máxima: 500 a 900 mA (depende de la versión).
- Cables: 4 hilos en par trenzado, 8 en USB 3.0.
- Pines: 4 (1 Alimentación, 2 Datos, 1 Masa).

Pin	Nombre	Color del cable	Descripción
1	VCC	Rojo	+5v
2	D−	Blanco	Data −
3	D+	Verde	Data +
4	GND	Negro	Masa

Tabla 2-1: Esquema Pines USB

2.5.2.4 Velocidades de Transmisión

Los dispositivos se clasifican en cuatro tipos según su velocidad de transferencia de datos:

- Baja Velocidad (1.0): Tasa de transferencia de hasta 1,5 Mbit/s (188 Kb/s). Utilizado en su mayor parte por dispositivos de interfaz humana como los teclados, ratones, cámaras web, etc.

- Velocidad Completa (1.1): Tasa de transferencia de hasta 12 Mbit/s. Ésta fue la más rápida antes de las especificaciones USB 2.0, y muchos dispositivos fabricados en la actualidad trabajan a esta velocidad. Estos dispositivos dividen el ancho de banda de la conexión USB entre ellos, basados en un algoritmo de impedancias LIFO.

- Alta Velocidad (2.0): Tasa de transferencia de hasta 480 Mbit/s, pero con una tasa real practica máxima de 280 Mbit/s. El cable USB 2.0 dispone de 4 líneas, un par para datos, y otro par de alimentación.

- Superalta Velocidad (3.0): Tasa de transferencia de hasta 4,8 Gbit/s. La velocidad del bus es 10 veces más rápida que la del USB 2.0, debido a que han incluido 5 contactos adicionales. Otra característica es su "regla de inteligencia", los dispositivos que se enchufan y después de un rato quedan en desuso, pasan inmediatamente a un estado de bajo consumo. A la vez, la intensidad de corriente se incrementa de 500 a los 900 mA, que sirve para abastecer a un teléfono móvil o un reproductor audiovisual portátil en menos tiempo.

- Futuro (3.1): Tasa de transferencia de hasta 10 Gbit/s. Se espera que para el año 2014 lleguen los primeros productos con esta tecnología.

Las señales del USB se transmiten en un cable de par trenzado con impedancia característica de 90 Ω ± 15 %, cuyos hilos se denominan D+ y D-. Éstos, colectivamente, utilizan señalización diferencial en *half dúplex,* excepto el USB 3.0 que utiliza un segundo par de hilos para realizar una comunicación en *full duplex*. La razón por la cual se realiza la comunicación en modo diferencial es simple, ya que reduce el efecto del ruido electromagnético en enlaces largos. Los niveles de transmisión de la señal varian de 0 a 0,3 V para el valor bajo (ceros) y de 2,8 a 3,6 V para el valor alto (unos) en las versiones 1.0 y 1,1, y en ± 400 mV en alta velocidad (2.0). En las primeras versiones, el apantallamiento de los cables no estaba conectados a masa, pero en el modo de alta velocidad se tiene una terminación de 45 Ω a masa o un diferencial de 90 Ω para acoplar la impedancia del cable.

2.5.3 USB On-The-Go

USB *On-The-Go*, también conocido por el acrónimo USB OTG, es una extensión de la norma USB 2.0 que permite a los dispositivos USB tener más flexibilidad en la gestión de la conexión USB. Permite que dispositivos como un reproductor de audio digital o teléfono móvil actúen como host, por lo que se les puede conectar una memoria USB, un ratón, un teclado, un disco duro, un modem, etc.

El estándar USB (USB 1.1/2.0) utiliza una arquitectura maestro/esclavo: un concentrador USB actúa como maestro USB, mientras un dispositivo USB actúa como esclavo. Sólo los concentradores USB pueden gestionar la configuración y la transferencia de datos durante la conexión. La norma USB OTG cambia esta situación. Los dispositivos compatibles con USB OTG son capaces de abrir una sesión, controlar la conexión e intercambiar las funciones maestro/dispositivo. Para ello, la norma USB OTG introduce dos nuevos protocolos: el protocolo SRP (*Session Request Protocol*: Protocolo de solicitud de sesión) y el protocolo HNP (*Host Negotiation Protocol*: Protocolo de Negociación de host).

2.5.4 Bluetooth

2.5.4.1 Definición

Se denomina Bluetooth al protocolo de comunicaciones diseñado especialmente para dispositivos de bajo consumo, que requieren corto alcance de emisión y basados en transceptores de bajo coste.

Bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN), que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz.

Los principales objetivos que se pretenden conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles y fijos.
- Eliminar los cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

Los dispositivos que con mayor frecuencia utilizan esta tecnología pertenecen a sectores de las telecomunicaciones y la informática personal, como PDA, teléfonos móviles, computadoras portátiles, ordenadores personales, impresoras o cámaras digitales.

2.5.4.2 Usos

Los dispositivos que incorporan este protocolo pueden comunicarse entre ellos cuando se encuentran dentro de su alcance. Las comunicaciones se realizan por radiofrecuencia, de forma que los dispositivos no tienen que estar alineados y pueden incluso estar en habitaciones separadas si la potencia de transmisión es suficiente. Estos dispositivos se clasifican como "Clase 1", "Clase 2" o "Clase 3" en referencia a su potencia de transmisión, siendo totalmente compatibles los dispositivos de una clase con los de las otras.

Clase	Potencia máxima permitida (mW)	Potencia máxima permitida (dBm)	Alcance (aproximado)
Clase 1	100 mW	20 dBm	~100 metros
Clase 2	2.5 mW	4 dBm	~10 metros
Clase 3	1 mW	0 dBm	~1 metro

Tabla 2-2: Esquema Potencia Transmisión Dispositivos *Bluetooth.*

En la mayoría de los casos, la cobertura efectiva de un dispositivo de clase 2 se extiende cuando se conecta a un transceptor de clase 1. Esto es así gracias a la mayor sensibilidad y potencia de transmisión del dispositivo de clase 1, es decir, la mayor potencia de transmisión del dispositivo de clase 1 permite que la señal llegue con energía suficiente hasta el de clase 2. Por otra parte la mayor sensibilidad del dispositivo de clase 1 permite recibir la señal del otro pese a ser más débil.

Versión	Ancho de banda
Versión 1.2	1 Mbit/s
Versión 2.0 + EDR	3 Mbit/s
Versión 3.0 + HS	24 Mbit/s
Versión 4.0	24 Mbit/s

Tabla 2-3: Clasificación Dispositivos *Bluetooth* según su Ancho de Banda.

2.5.4.3 Información Técnica

La especificación de Bluetooth define un canal de comunicación a un máximo 720 Kbit/s (1 Mbit/s de capacidad bruta) con rango optimo de 10 m (opcionalmente 100 m con repetidores).

Opera en la frecuencia de radio de 2,4 a 2,48 GHz con amplio espectro y saltos de frecuencia con posibilidad de transmitir en *full dúplex,* con un máximo de 1600 saltos por segundo. Los saltos de frecuencia se dan entre un total de 79 frecuencias con intervalos de 1 MHz; esto permite dar seguridad y robustez.

La potencia de salida para transmitir a una distancia máximo de 10 metros es de 0 dBm (1 mW), mientras que la versión de largo alcance transmite entre 20 y 30 dBm (entre 100 mW y 1 W).

Para lograr alcanzar el objetivo de bajo consumo y bajo costo se ideó una solución que se puede implementar en un solo chip utilizando circuitos CMOS.

2.5.4.4 Arquitectura

El *hardware* que compone el dispositivo *Bluetooth* está compuesto por dos partes:

- Un dispositivo de radio, encargado de modular y transmitir la señal.
- Un controlador digital, compuesto por una CPU, un procesador de señales digitales llamado *Link Controller* y de las interfaces con el dispositivo anfitrión.

El *Link Controller* se encarga del procesamiento de la banda base y del manejo de los protocolos ARQ y FEC de la capa física; además, se encarga de las funciones de transferencia tanto asíncrona como síncrona, la codificación de audio y el cifrado de datos.

La CPU del dispositivo se encarga de las instrucciones relacionadas con *Bluetooth* en el dispositivo anfitrión, para así simplificar su operación. Para ello, sobre la CPU corre un *software* denominado *Link Manager* cuya función es la de comunicarse con otros dispositivos por medio del protocolo LMP.

Entre las tareas realizadas por el *Link Controller* y el *Link Manager*, destacan las siguientes:

- Envío y Recepción de Datos.
- Paginación y Peticiones.
- Establecimiento de conexiones.
- Autentificación.
- Negociación y establecimiento de tipos de enlace.

- Establecimiento del tipo de cuerpo de cada paquete.
- Establecer el dispositivo en modo *sniff* o *hold*.

2.5.4.5 Especificaciones

- Bluetooth v1.0 y v1.0b: Las versiones 1.0 y 1.0b tuvieron mucho problemas, y los fabricantes tenían dificultades para hacer sus productos interoperables. Las versiones 1.0 y 1.0b incluyen en hardware de forma obligatoria la dirección del dispositivo Bluetooth (BD_ADDR) en la transmisión (el anonimato se hace imposible a nivel de protocolo), lo que fue un gran revés para algunos servicios previstos para su uso en entornos Bluetooth.
- Bluetooth v1.1
    - Ratificado como estándar IEEE 802.15.1-2002.
    - Se corrigieron muchos errores en las especificaciones 1.0b.
    - Añadido soporte para canales no cifrados.
    - Indicador de señal recibida (RSSI).
- Bluetooth v1.2: Esta versión es compatible con USB 1.1.
    - Una conexión más rápida y *Discovery* (detección de otros dispositivos *Bluetooth*).
    - Salto de frecuencia adaptable de espectro ampliado (AFH), que mejora la resistencia a las interferencias de radiofrecuencia, evitando el uso de las frecuencias de lleno en la secuencia de saltos.
    - Mayor velocidad de transmisión en la práctica, de hasta 721 Kbit/s.
    - Conexiones Sincrónicas Extendidas (ESCO), que mejoran la calidad de voz en los enlaces de audio al permitir la retransmisión de paquetes corruptos, y, opcionalmente, puede aumentar la latencia de audio para proporcionar un mejor soporte para la transferencia de datos simultánea.
    - *Host Controller Interface* (HCI) el apoyo a tres hilos UART.
    - Ratificado como estándar IEEE 802.15.1-2005.

  - Introdujo el control de flujo y los modos de retransmisión L2CAP.

- Bluetooth v2.0 + EDR: La principal diferencia es la introducción de una velocidad de datos mejorada para acelerar la transferencia de datos. La tasa nominal de EDR es de 3 Mbit/a, aunque la tasa de transferencia de datos práctica es de 2,1 Mbit/s.

- Bluetooth v2.1 + EDR: Se mejora la experiencia de emparejamiento de dispositivos Bluetooth (*Secure Simple Pairing*).

- Bluetooth v3.0 + HS: Soporta velocidades de transferencia de datos teórica de hasta 24 Mbit/s, aunque no a través del enlace Bluetooth propiamente dicho. La conexión *Bluetooth* nativa se utiliza para la negociación y el establecimiento mientras que el tráfico de datos de alta velocidad se realiza mediante un enlace 802.11.

- Bluetooth v4.0: Se incluye *Bluetooth* clásico, *Bluetooth* de alta velocidad y protocolos de *Bluetooth* de baja energía. *Bluetooth* de alta velocidad se basa en Wi-Fi, y *Bluetooth* clásico se compone de protocolos *Bluetooth* heredados. *Bluetooth* baja energía (BLE) es un subconjunto de funcionalidades de *Bluetooth* v4.0 con una pila de protocolo completamente nuevo para la rápida acumulación de enlaces sencillos.

2.6 Posibilidades en el Diseño de un Gadget

Existen arquitecturas *hardware* y *software* que permiten desarrollar Gadgets sin que sea necesario invertir mucho tiempo en el diseño de todos los componentes *hardware* y *software* necesarios.

En particular, cuando el diseño del Gadget está enfocado a un dispositivo móvil inteligente basado en el sistema operativo Android, una alternativa sería utilizar uno de los kits de desarrollo vistos anteriormente, como por ejemplo la placa de desarrollo IOIO.

Se trata de un sistema de desarrollo que incluye un micro-controlador de Microchip y permite la conexión de un dongle *Bluetooth*, consiguiéndose así una comunicación inalámbrica bidireccional entre la placa de desarrollo IOIO y el dispositivo móvil inteligente.

El micro-controlador comentado no requiere de programación alguna por parte del desarrollador, ya que existe un *firmware* genérico que permite un control bastante extenso de la plataforma *hardware*.

En definitiva, el diseño del Gadget consiste en conectar los sensores y/o actuadores en puertos de la placa IOIO, así como realizar la programación de los componentes *software* necesarios para diseñar la aplicación para el dispositivo móvil inteligente.

Para ello, la placa IOIO proporciona una biblioteca que permite realizar tareas tales como las siguientes: configurar un pin de entrada o salida, fijar un pin de salida digital con los valores alto o bajo, etc.

En la Ilustración 2-12 se observa lo comentado anteriormente, un dispositivo móvil inteligente comunicándose a través de *Bluetooth* con la placa IOIO para actuar, en este ejemplo, sobre un LED.

Ilustración 2-12: Ejemplo de diseño de un *Gadget* basado en el kit IOIO.

Continuando con las posibilidades en el diseño de un Gadget, en algunas ocasiones la placa IOIO no será suficiente y será necesario utilizar un segundo micro-controlador conectado con dicha placa.

Un ejemplo de esto podría ser cuando el Gadget incorpora una unidad de almacenamiento permanente de tipo SD. En este caso la solución consiste en conectar la tarjeta SD con el segundo micro-controlador y conectar éste con la placa IOIO usando una interfaz digital estándar (I^2C, UART, etc.). Esta arquitectura de diseño de Gadget se muestra en la Ilustración 2-13.

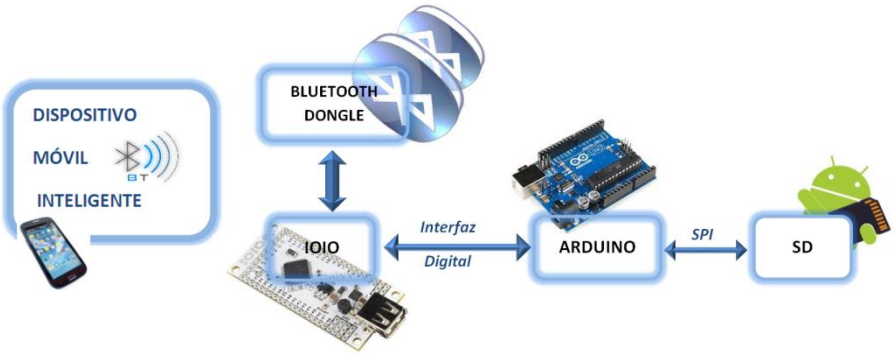


Ilustración 2-13: Arquitectura de diseño de un *Gagdet* basado en varios micro-controladores.

Como se ha comentado anteriormente, en el caso de que el diseño siga un flujo basado en la conexión de la placa IOIO con un segundo micro-controlador, será necesario seleccionar este último, siendo una solución óptima la arquitectura de *hardware* y *software* libre Arduino.

La arquitectura Arduino es de *hardware* libre porque te proporcionan todos los ficheros y diagramas esquemáticos para que el usuario pueda construirse la placa de Arduino, además el entorno de desarrollo puede descargarse libremente desde la página oficial de Arduino (*software* libre), existiendo una extensa comunidad que contribuye día a día a esta arquitectura.

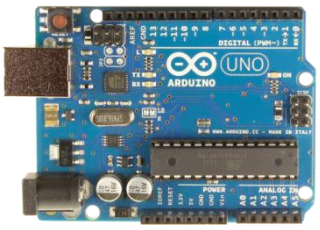


Ilustración 2-14: Placa Arduino UNO

Para esta arquitectura existen unas placas (Shields) también de *hardware* libre, que pueden ser conectadas encima de la placa Arduino extendiendo sus capacidades. Éstas Shields vienen con una biblioteca software con objeto de poder desarrollar un prototipo con poco tiempo de diseño invertido.

Ilustración 2-15: Arduino Ethernet Shield.

2.7 Sistemas Operativos para Smartphones

2.7.1 Android

Android [7] es un sistema operativo móvil basado en Linux, que junto con aplicaciones *middleware* (*software* que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones) está enfocado para ser utilizado en dispositivos móviles como *Smartphones*, *Tablets*, Google TV y otros dispositivos.

Fue desarrollado inicialmente por Android Inc., una firma comprada por Google en 2005. Pero en realidad es el principal producto de la *Open Handset Alliance*, un conglomerado de fabricantes y desarrolladores de *hardware*, *software* y operadores de servicios.

El anuncio del sistema Android se realizó el 5 de Noviembre de 2007 junto con la creación de la *Open Handset Alliance*. Google libero la mayoría del código de Android bajo la licencia Apache, una licencia libre y de código abierto.

El sistema operativo se halla en una zona de memoria de sólo lectura por dos motivos: para evitar que el usuario lo dañe sin querer y para que se sea fiel a las pequeñas modificaciones y aplicaciones integradas que los fabricantes suelen incluir en sus modelos.

Al ser un sistema operativo de código fuente abierto, Android permite toda clase de modificaciones. Además de las ROM oficiales, es muy habitual encontrar ROM hechas por grupos de voluntarios que toman el código base y le añaden o quitan características o interfaces de usuario. Incluso uno mismo puede crear la suya.

Para actualizar un móvil Android se tienen varias opciones, siempre dependiendo de la operadora y sobre todo del fabricante de nuestro dispositivo. Algunos permiten actualizar por medio de la conexión USB entre el móvil y el PC, y otros directamente en el dispositivo descargando un archivo a la microSD y encendiendo el móvil. Sin embargo, lo más normal es que se actualice un Android por OTA (*Over The Air*) o inalámbricamente, sin tener que conectar el móvil por cable ni para descargar un archivo ni para actualizar mediante un programa.

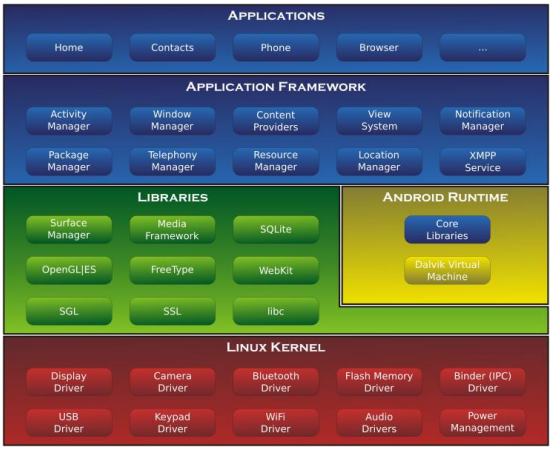


Ilustración 2-16: Arquitectura de Android.

La arquitectura de Android está distribuida en diferentes capas, las cuales se describen a continuación:

- *Applications* (Aplicaciones): Las aplicaciones bases incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.

- *Application Framework* (Marco de trabajo de aplicaciones): Los desarrolladores tienen acceso completo a los mismos APIs del *framework* usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del *framework*). Este mecanismo permite que los componentes sean reemplazados por el usuario.

- *Libraries* (Bibliotecas): Android incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del *framework* de aplicaciones de Android. Las bibliotecas escritas en lenguaje C/C++ incluyen un administrador de pantalla táctil (*surface manager*), un *framework Open Core* (para el aprovechamiento de las capacidades multimedia), una base de datos relacional *SQLite*, una API gráfica *OpenGL ES 2.0 3D*, un motor de renderizado *WebKit*, un motor gráfico SGL, el protocolo de comunicación segura SSL y una biblioteca estándar de C, llamada "*Bionic*" y desarrollada por Google específicamente para Android a partir de bibliotecas estándar "*libc*" de BSD.

- *Android Runtime* (Funcionalidad en tiempo de ejecución): Android incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual *Dalvik*. *Dalvik* ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. *Dalvik* ejecuta archivos en el formato *Dalvik Executable* (.dex), el cual está optimizado para memoria mínima.

- *Linux Kernel* (Núcleo Linux): Android dispone de un núcleo basado en *Linux* para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores, y también actúa como una capa de abstracción entre el *hardware* y el resto de la pila de *software*.

Los desarrolladores informaron de la dificultad de mantener aplicaciones para versiones diferentes de Android, debido a problemas de compatibilidad entre la versión 1.5 y 1.6, especialmente, por diferencias de resolución entre los distintos teléfonos Android.

Estos problemas se hicieron patentes durante el concurso ADC2. Posteriormente, el rápido aumento de modelos de teléfonos basados en Android con diferentes capacidades de *hardware* complicaba el desarrollo de aplicaciones para todos los modelos de teléfonos Android.

Plataforma	Nivel de API	%
4.x.x *Jelly Bean*	16	2,7%
4.0.x *Ice Cream Sandwich*	14-15	25,8%
3.x.x *Honeycomb*	12-13	1,8%
2.3.x *Gingerbread*	9-10	54,2%
2.2 *Froyo*	8	12%
2.1 *Eclair*	7	3,1%
1.6 *Donut*	4	0,3%
1.5 *Cupcake*	3	0,1%

Ilustración 2-17: Versiones Android.

Sin embargo, la situación parece haber mejorado "en parte", porque aún más del 54 % de teléfonos con Android usan versiones 2.3.X (salida en 2010). El principal problema de esto es a la hora de desarrollar aplicaciones nuevas con las APIs más recientes del SDK de Android, se tiene que usar una versión de API compatible con 2.3, así las nuevas funcionalidades de APIs superiores, como NFC a partir de la versión Android ICS 4.0, no se pueden usar si no se implementan a mano por no poder usar la API que hace uso de su funcionalidad.

2.7.2 iOS

iOS [8] (anteriormente denominado iPhone OS) es un sistema operativo móvil de Apple. Fue originalmente desarrollado para el *Smartphone* iPhone, siendo después usado en dispositivos como el iPod Touch, iPad y el Apple TV.

La interfaz de usuario de iOS está basada en el concepto de manipulación directa, usando gestos multitáctiles (*multitouch*). Los elementos de control son deslizadores, interruptores y botones. La respuesta a las órdenes del usuario es inmediata y provee de una interfaz fluida. La interacción con el sistema operativo incluye gestos como deslices, toques, pellizcos, los cuales tienen definiciones diferentes dependiendo del contexto de la interfaz.

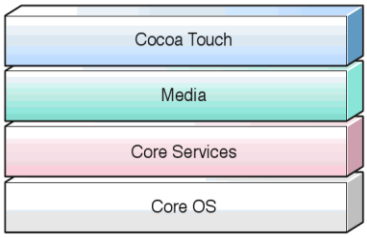


Ilustración 2-18: Cuatro capas de

iOS se deriva de Mac OS X, que a su vez está basado en Darwin BSD (plataforma de código abierto), que es un sistema operativo tipo Unix. iOS cuenta con cuatro capas de abstracción: la capa del núcleo del sistema operativo, la capa de "Servicios Personales", la capa de "Medios" y la capa de "*Cocoa Tocuh*". La versión actual del sistema operativo (iOS 6.0) ocupa más o menos 770 MB, variando por modelo.

2.7.3 Symbian OS

Symbian es un sistema operativo que fue producto de la alianza de varias empresas de telefonía móvil, entre las que se encuentran: Nokia, Sony Ericsson, Psion, Samsung, Siemens, Arima, Benq, Fujitsu, Lenovo, LG, Motorola, Mitsubishi Electric, Panasonic, Sharp, etc. Sus orígenes provienen de su antepasado EPOC32, utilizado en PDA´s.

El objetivo de Symbian fue crear un sistema operativo para terminales móviles que pudiera competir con el de Palm o el Windows Mobile 6.X de Microsoft y ahora con el SO Android de Google Inc, iOS de Apple Inc y BlackBerry OS de RIM.

En 2003 Motorola vendió el 13 % de su participación a Nokia, lo cual hizo que se quedara con el 32,3 % de la compañía. Más tarde, sin embargo, después de no tener el éxito esperado con sus terminales "*Linux-Like*", volvió al mundo de Symbian comprándole el 50 % de las participaciones a Sony Ericsson. El 24 de Junio de 2008, Nokia decidió comprar Symbian, adquiriendo el 52 % restante de las acciones de la compañía. El objetivo era establecer la Fundación Symbian y convertir este sistema operativo en una plataforma abierta. Entre 2009 y 2010 Nokia decide transferir el soporte y desarrollo del sistema operativo Symbian a la consultora *Accenture*, terminando la operación a finales de

Septiembre de 2011, cuando se terminó el desarrollo de la nueva versión Symbian Belle, convirtiéndose en la última versión de Symbian en la que Nokia participó de forma exclusiva. En Octubre de 2011 se confirma de forma oficial que Symbian tendrá soporte, sólo, hasta el año 2016, por no poder seguir siendo un competidor para la nueva versión de *Smartphones* con sistemas operativos de última generación como Androis, iOS o Windows Phone.

La interfaz gráfica por defecto es S60: La plataforma S60 es una plataforma para terminales móviles que utilicen el sistema operativo Symbian OS. Está desarrollada principalmente por Nokia y licenciada por ellos a otros fabricantes.

Symbian^3 se considera una nueva generación del sistema operativo Symbian y se usa en los *Smarpthones* de nueva generación de Nokia. Ésta tiene una gran compatibilidad de *hardware* y soporte para gráficos acelerados con la aceleración de *hardware* en 2D y 3D, soporte para HDMI, hasta 3 pantallas de inicio personalizables con *widgets*, mejoras estéticas notables gracias a la aceleración de gráficos y muchas mejoras generales en estabilidad, entre ellas la consistencia. Ésta es la primera versión de código abierto de Symbian, la cual se presentó un mes después de haber liberado el código fuente de todo el sistema.

2.7.4 Windows Mobile y Windows Phone

Windows Mobile [9] es un sistema operativo móvil de Microsoft, diseñado para su uso en *Smartphones* y otros dispositivos móviles.

Se basa en el núcleo del sistema operativo Windows CE y cuenta con un conjunto de aplicaciones básicas utilizando la API de Microsoft Windows. Está diseñado para ser similar, en su estética, a las versiones de escritorio de Windows. Además, existe una gran oferta de *software* de terceros disponibles para Windows Mobile.

Windows Phone es un sistema operativo móvil también desarrollado por Microsoft. Está pensado para el mercado de consumo generalista en lugar del mercado empresarial, por lo que carece de muchas funcionalidades que proporciona la versión anterior. Microsoft decidió no hacer compatible Windows Phone con Windows Mobile.

2.7.5 BlackBerry OS

BlackBerry Os es un sistema operativo móvil, de código cerrado, desarrollado por Research In Motion (RIM) para sus propios dispositivos BlackBerry.

El sistema permite multitarea y tiene soporte para diferentes métodos de entrada adoptados por RIM para su uso en computadoras de mano, particularmente la *trackwhell*, *trackball*, *touchpad* y pantallas táctiles.

El SO BlackBerry está claramente orientado a su uso profesional como gestor de correo electrónico y agenda. Desde la cuarta versión se puede sincronizar el dispositivo con el correo electrónico, el calendario, tareas, notas y contactos de *Microsoft Exchange Server*. Al igual que en el SO Symbian, desarrolladores independientes también pueden crear programas para BlackBerry pero en el caso de querer tener acceso a ciertas funcionalidades restringidas necesitan ser firmados digitalmente para poder ser asociados a una cuenta de desarrolladores de RIM.

Capítulo 3

Descripción del Sistema

3.1 Introducción

En el capítulo anterior se han estudiado algunos de los kits de desarrollo existentes en el mercado, así como las posibilidades de comunicación de éstos.

Dentro de las alternativas propuestas, se ha seleccionado proponer una arquitectura basada en el módulo IOIO en conjunto con la placa Arduino UNO.

Las ventajas de utilizar dichos elementos son que el *Gadget* se puede comunicar inalámbricamente mediante *Bluetooth* con el dispositivo móvil inteligente y que en caso de tener que conectar un sensor con una interfaz no compatible con el módulo IOIO, éste se podría conectar fácilmente con Arduino.

Mencionar que el Kit de Desarrollo IOIO OTG fue lanzado al mercado después de la realización de este Proyecto Fin de Carrera.

3.2 IOIO

En esta sección se va a describir tanto el *hardware* como el *software* de la placa IOIO, así como el entorno de desarrollo necesario para la realización de la aplicación Android.

3.2.1 Hardware

3.2.1.1 Vista Conjunta

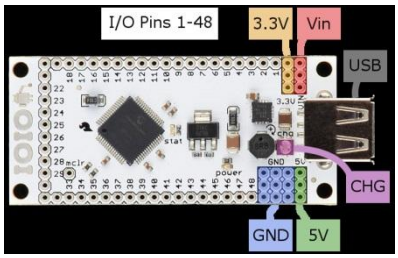


Ilustración 3-1: Vista Conjunta Placa IOIO

La placa IOIO contiene los siguientes componentes:

- Conector USB (tipo A): Se utiliza para conectar IOIO con el dispositivo Android en caso de comunicación cableada o con el módulo *Bluetooth* en caso de comunicación inalámbrica.
- 9 Pines GND: Conexión a tierra.
- 3 Pines VIN: Se utiliza para la alimentación de la placa. El voltaje suministrado debe ser entre 5 V – 15 V.
- 3 Pines 5 V: Se utiliza normalmente como salida de 5 V, cuando se alimenta la placa a través de los pines VIN, pero se puede utilizar como entrada de 5 V para alimentar a la placa en el caso en el que no esté alimentada a través de VIN.
- 3 Pines de 3,3 V: Salida de voltaje regulada a 3,3 V.
- 48 Pines E/S (numerado del 1-48): Pines de E/S de propósito general (más adelante se detallaran las funciones especiales que tienen algunos).
- LED de alimentación: Se ilumina cuando IOIO está correctamente alimentada.
- Pin MCLR: Se utiliza para la programación de un nuevo *firmware bootloader* en la placa IOIO.
- CHG o *Charge Current Trimmer*: Se utiliza para ajustar la cantidad de corriente suministrada en la línea VBUS del USB para el dispositivo

Android o para el dispositivo inalámbrico. Girando en dirección "+" se incrementa la intensidad.

3.2.1.2 Pines E/S

Los Pines de E/S son la esencia de IOIO. Se utilizan para conectar los circuitos externos utilizando diferentes interfaces. El denominador común de todos los pines es que pueden servir tanto para entradas como salidas digitales a 3,3 V.

Para mayor comodidad, las funciones comunes de los pines se representan gráficamente en la parte inferior de la placa, con una leyenda:

- Pines rodeados por un cuadrado se pueden utilizar como entradas analógicas (3,3 V).
- Pines rodeados por un círculo son tolerantes a 5 V, es decir, se puede utilizar como entradas de lógica a 5 V o salida de 5 V lógica, cuando se utiliza el modo de apertura con la ayuda de una resistencia de pull-up.
- Pines marcados con "P" se utilizan como entradas y salidas periféricas. Principalmente, esto incluye PWM, UART y SPI.
- Pines marcados con "Pi" se utilizan como entrada periférica (pero no de salida).
- Pines marcados con DAx y CLx se utilizan para TWI.

3.2.1.3 Tabla Funciones Pines

Leyenda:

- A/D: Pin que puede ser utilizado como entrada analógica.
- I²C: Pin que puede ser utilizado como I²C, DAx (Pin Data) y CLx (Pin Reloj), siendo "x" el número del módulo del I²C.
- PPSi: Pin que puede ser utilizado como entrada para periféricos reasignables (captura de entrada, UART, SPI).
- PPSo: Pin que puede ser utilizado como salida para periféricos reasignables (salida del comparador, UART, SPI).

- 5 V: Pin tolerante a 5 V, que se puede utilizar como entrada o salida a 5 V en el modo de apertura.

- Comp: Pin que puede ser utilizado como número de entrada del comparador.

- Prog: Pin que puede ser utilizado para ICSP, usa Vpp y/o C1/D1, C2/D2 o C3/D3, que son reloj y datos, respectivamente.

- Todos los pines pueden ser utilizados como entrada/salida digital (GPIO), además de soportar interrupciones.

IOIO pin	A/D	I²C	PPSi	PPSo	5V	Comp.	Prog.	PIC pin	PIC function
1								39	OSCI/CLKI/CN23/RC12
2								40	OSCO/CLKO/CN22/RC15
3			Y	Y	Y			42	RTCC/DMLN/RP2/CN53/RD8
4		DA0	Y	Y	Y			43	DPLN/SDA1/RP4/GD8/CN54/RD9
5		CL0	Y	Y	Y			44	SCL1/RP3/GD6/CN55/RD10
6			Y	Y	Y			45	RP12/GD7/CN56/RD11
7			Y	Y	Y			46	DMH/RP11/INT0/CN49/RD0
8						3D		47	SOSCI/C3IND/CN1/RC13
9			Y			3C		48	SOSCO/SCLKI/T1CK/C3INC/RPI37 /CN0/RC14
10			Y	Y	Y			49	VCPCON/RP24/GD9/VBUSCHG/CN50/RD1
11			Y	Y	Y			50	DPH/RP23/CN51/RD2
12			Y	Y	Y			51	RP22/GEN/CN52/RD3
13			Y	Y	Y			52	RP25/GCLK/CN13/RD4
14			Y	Y	Y			53	RP20/GPWR/CN14/RD5
15						3B		54	C3INB/CN15/RD6
16						3A		55	C3INA/SESSEND/CN16/RD7
17								58	GD10/VBUSST/VCMPST1/VBUSVLD /CN68/RF0
18					Y			59	GD11/VCMPST2/SESSVLD/CN69/RF1
19					Y			60	GD0/CN58/RE0
20					Y			61	GD1/CN59/RE1
21					Y			62	GD2/CN60/RE2
22					Y			63	GD3/CN61/RE3

Tabla 3-1: Tabla Funciones Pines

23					Y			64	HSYNC/CN62/RE4
24					Y			1	VSYNC/CN63/RE5
25		CL2			Y			2	GD12/SCL3/CN64/RE6
26		DA2			Y			3	GD13/SDA3/CN65/RE7
27			Y	Y		1D		4	C1IND/RP21/CN8/RG6
28			Y	Y		1C		5	C1INC/RP26/CN9/RG7
29			Y	Y		2D		6	C2IND/RP19/GD14/CN10/RG8
30			Y	Y		2C		8	C2INC/RP27/GD15/CN11/RG9
31	Y		Y	Y		1A	C3	11	PGEC3/AN5/C1INA/VBUSON /RP18/CN7/RB5
32	Y		Y	Y		1B	D3	12	PGED3/AN4/C1INB/USBOEN /RP28/CN6/RB4
33	Y					2A		13	AN3/C2INA/VPIO/CN5/RB3
34	Y		Y	Y		2B		14	AN2/C2INB/VMIO/RP13/CN4/RB2
35	Y (ref +)		Y	Y			C1	15	PGEC1/AN1/VREF-/RP1/CN3/RB1
36	Y (ref -)		Y	Y			D1	16	PGED1/AN0/VREF+/RP0/CN2/RB0
37	Y		Y	Y			C2	17	PGEC2/AN6/RP6/CN24/RB6
38	Y		Y	Y			D2	18	PGED2/AN7/RP7/RCV/CN25/RB7
39	Y		Y	Y				21	AN8/RP8/CN26/RB8
40	Y		Y	Y				22	AN9/RP9/CN27/RB9
41	Y							23	TMS/CVREF/AN10/CN28/RB10
42	Y							24	TDO/AN11/CN29/RB11
43	Y							27	TCK/AN12/CTEDG2/CN30/RB12
44	Y							28	TDI/AN13/CTEDG1/CN31/RB13
45	Y		Y	Y				29	AN14/CTPLS/RP14/CN32/RB14

Tabla 3-2: Tabla Funciones Pines (continuación)

46	Y		Y	Y				30	AN15/RP29/REFO/CN12/RB15
47		DA1	Y	Y	Y			31	SDA2/RP10/GD4/CN17/RF4
48		CL1	Y	Y	Y			32	SCL2/RP17/GD5/CN18/RF5
stat LED			Y	Y	Y			33	RP16/USBID/CN71/RF3
mclr							Vpp	7	MCLR

Tabla 3-3: Tabla Funciones Pines (continuación)

3.2.2 Fuente de Alimentación

3.2.2.1 Conexión Básica

La forma más común para alimentar a la placa IOIO es mediante la conexión de una fuente DC, en cualquier lugar entre 5 V – 15 V, con el lado "+" a uno de los tres pines marcados como "VIN" y el lado "+" a uno de los 9 pines marcados como "GND".

Tan pronto como se hace esto, el LED de encendido rojo debe encenderse. Otra forma más robusta es soldar un conector JST de montaje en superficie en la parte inferior de la placa, en los *pads* derecha debajo del conector USB.

3.2.2.2 Carga del Dispositivo Android desde IOIO

IOIO suministra 5 V para el dispositivo Android a través de la conexión USB (en la línea VBUS). Este suministro de 5 V permite la carga de la batería del dispositivo Android además de permitirle saber al dispositivo Android que un host USB está conectado. Con el potenciómetro se regula la corriente en la carga.

3.2.2.3 Limitación de Corriente de Carga

En aquellas aplicaciones en las que no se desee cargar el dispositivo, la corriente en la carga puede ser limitada. Una limitación de la corriente de carga, supone una disminución de la tensión en la línea VBUS del USB. De acuerdo con las especificaciones USB, la disminución de VBUS por debajo del umbral de 5 V es un voltaje erróneo. Esto no va a dañar el dispositivo Android, pero puede hacer que el dispositivo Android deje de detectar la conexión con la placa IOIO.

El punto en el que el dispositivo móvil inteligente deja de detectar la conexión con la placa IOIO varía según el dispositivo. La forma de ajustar la corriente en la carga necesaria para un dispositivo específico es ir regulándola con el potenciómetro.

3.2.2.4 Descripción de las Características de Potencia

IOIO tiene 2 reguladores de tensión en la placa:

- Un regulador de conmutación para entradas de 5 V – 15 V y para salidas de 5 V estables y que puede suministrar hasta 1,5 A.

- Un regulador lineal que se alimenta de la línea de 5 V, con salidas de 3,3 V estables y que puede suministrar hasta 800 mA.

Los 3 Pines de 5 V proporcionan 1 A de intensidad a los dispositivos externos, mientras que los pines de 3,3 V proporcionan 700 mA a los periféricos externos.

3.2.3 Bluetooth

3.2.3.1 Introducción

Como se mencionó en el capítulo anterior, la placa de desarrollo IOIO no posee un módulo integrado de *Bluetooth*, pero permite este tipo de comunicación conectando un módulo *Bluetooth*.

3.2.3.2 Módulos Bluetooth

A continuación se muestran los diferentes módulos *Bluetooth* que han sido probados y que funcionan con la placa IOIO.

Bluetooth Dongle Address And Name	Online Shop	Range [m]	Android Device Used For Testing	Android Version	IOIO Bootloader	IOIO Firmware	Comments
Miniature Bluetooth Dongle V2.0 Class I with 100m Range	maplin	54-58m	Nexus One, Nexus S, HTC One X	2.3.6, 4.0.3	3.03	3.23	ceramic antenna
Miniature Bluetooth Dongle V2.0	Seeedstudio	12m	Several	2.2.x,2.3.x 4.x	3.03	3.23	Magic Dongle
.... **00:15:83:3D:0A:57** .... Bluetooth 2.0 EDR Class 1 Vista-Ready USB Dongle 100m	DealExtreme	54-56m	Nexus One,Nexus S, Galaxy Nexus, HTC One S,HTC One X	2.3.6, 4.0.3	3.03	3.23	antenna is a fake
Ultra-Mini Bluetooth 3.0 USB Dongle	DealExtreme	50m	Nexus One, Nexus S, HTC One X	2.3.6, 4.0.3	3.03	3.23	nice signal quality
.... **00:15:83:15:A3:10** .... Mini Bluetooth 2.0 Dongle	DealExtreme	49m	Nexus One, Nexus S	2.3.6, 4.0.3	3.03	3.23	big latency at higher distance
Bluetooth USB Dongle (Windows 7 Compatible)	Inex	30m	LG-P350, Samsung Galaxy Gio, Google Nexus S, Sony Tablet S	2.3.4, 3.2.1, 4.0.4	3.03	3.11, 3.23	
.... **00:1F:81:00:08:30** .... Super Mini Bluetooth 2.0 Adapter Dongle	DealExtreme	12m	Nexus S	4.0.3	3.03	3.23	cheapest!
.... **00:1F:81:00:08:30** .... Super Mini Bluetooth 2.0 Adapter Dongle	DealExtreme	12m	Nexus S	4.0.3	3.03	3.23	

Tabla 3-4: Tabla Módulos *Bluetooth* Compatibles

Technaxx BT01 Bluetooth Dongle Class 1	Amazon	12m	Nexus S	4.0.3	3.03	3.23	
Mini Bluetooth 3.0 USB Dongle	DealExtreme	?	Xperia X8	2.3.7	3.01	3.11	
DYNAMODE MINI USB BLUETOOTH 100M BT-USB-M2	Ebay	?	?	?	?	?	
PPA USB Bluetooth Adapter v2.0 + EDR	FRYS	?	?	?	?	?	
Kinivo BTD-300 Bluetooth 3.0 USB adapter	Amazon	?	nexus 7, HTC One V	4.1.1,4.0.3	?	?	
.... **00:15:83:3D:0A:57** .... Generic USB 2.0 BT	Amazon	?	HTC Sensation 4G	4.0.3	3.01	3.11	Useful red LED
.... **00:15:83:3D:0A:57** .... Generic USB 2.0 BT	eBay	?	HTC One V	4.0.3	3.03	3.23	Useful red LED
Silver Hawk Bluetoothstick 1.2	N.A.	?	LG-P500, Acer A100	?	3.03	3.23	Very old part, eos
.... **00:1F:81:00:08:30** .... Mini Bluetooth 2.0 USB Dongle	Banggood	?	Nexus One	2.3.6	3.03	3.23	antenna is a fake
.... **00:15:83:3D:0A:57** .... Mini Bluetooth USB 2.0 Adapter V2.0 EDR Dongle	Banggood	?	Nexus One	2.3.6	3.03	3.23	
.... **00:11:67:D6:2F:49** .... MINI USB Bluetooth 3.0+ HS Compliant Dongle	Banggood	?	Nexus One	2.3.6	3.03	3.23	blue LED

Tabla 3-5: Tabla Módulos *Bluetooth* Compatibles (continuación)

.... **00:15:83:3D:0A:57** .... Bluetooth 2.0 USB Dongle	DealExtreme	?	Nexus One, HTC One X	2.3.6, 4.0.3	3.03	3.23	antenna is a fake
.... **00:15:83:0C:BF:EB** .... Magic Dongle	?	?	Nexus Galaxy, Nexus 7	4.1.1	3.03	3.23	used in Droidalyzer and IOIO Mint
.... **00:15:83:44:B9:E7** .... iConcepts v2.1 bluetooth adapter	Microcentet	?	Samsung Stealth V	2.36	3.03	3.23	
.... **00:11:67:D6:44:2B**	?	?	Droid 2 Global, Nexus 7	2.3.4, 4.1.1	3.01	3.23	
.... **00:1F:81:00:08:30** .... USB Bluetooth 2.0	DealExtreme	?	Nexus One	2.3.6	3.03	3.23	
.... **00:15:83:15:A3:10** .... Mini Mushroom Bluetooth Adapter	TrendTech	?	Galaxy Ace, Galaxy S III	2.3.3, 4.0.4	3.00	3.10	
.... **00:1F:81:00:08:30** .... Kingsmart BUD-07B (Bluetooth 2.0 + EDR Class 2)	Amazon	?	Galaxy S II Skyrocket	4.0.4	3.04	3.24	

Tabla 3-6: Tabla Módulos *Bluetooth* Compatibles (continuación)

3.2.4 Software

3.2.4.1 Introducción

La placa IOIO está basada en un micro-controlador de Microchip. Principalmente, dicha placa incluye una interfaz USB y tiene externalizados los pines del micro-controlador. En relación al desarrollo del *software*, se puede optar por dos alternativas.

La primera de ellas sería un diseño tradicional usando una herramienta de carga y depuración de Microchip.

La otra alternativa es posible gracias al desarrollo realizado por la comunidad IOIO. En dicha comunidad se puede descargar un *firmware* para el micro-controlador de la placa IOIO. Este *firmware* permite comunicar la placa IOIO a través de la interfaz USB (de forma cableada, así como inalámbrica conectando un módulo *Bluetooth*) con el dispositivo móvil inteligente. Además, proporcionan un conjunto de componentes *software* compatible con Android, los cuales permiten un control de bajo nivel de dicha placa. Así por ejemplo, desde Android se puede configurar un determinado pin digital de salida y poner dicho pin a valor alto o bajo. Naturalmente, esta segunda alternativa de diseño es la que permite diseñar prototipos de *Gadgets* de una forma más rápida y será la utilizada para implementar la arquitectura de diseño de accesorios electrónicos propuesta en este Proyecto Fin de Carrera.

3.2.4.2 Conceptos Previos

El corazón de la placa IOIO es un micro-controlador. Este micro-controlador ejecuta el código ("*firmware*") que da a IOIO su funcionalidad: el establecimiento de una conexión de datos con un dispositivo Android, el control de los pines, etc. Este código se almacena en la memoria Flash que existe en el interior del micro-controlador. El código *firmware* de IOIO consta de dos partes principales: gestor de arranque (*bootloader*) y la aplicación.

El gestor de arranque (*bootloader*) es el primer código que ejecuta IOIO cada vez que se reinicia. Lo que hace es establecer una conexión de datos con el dispositivo Android, y a continuación, utiliza esa conexión de datos para comprobar la existencia de nuevos códigos de aplicación en el dispositivo Android. Si lo encuentra, lo instala y lo ejecuta. Si no, se ejecuta cualquier aplicación que ya haya sido instalada.

El *firmware* de la aplicación proporciona a la placa IOIO su funcionalidad principal, se comunica con el IOIOLib, que se ejecuta en el dispositivo Android, ejecutándose posteriormente los comandos para el control de algún periférico, pines digitales, módulo UART etc.

IOIO viene pre-programado con un gestor de arranque y una aplicación.

3.2.4.3 IOIO Manager

IOIO Manager es una aplicación para Android que permite instalar un nuevo *firmware* en la placa IOIO. La aplicación es de código abierto y está disponible para descargarla de forma gratuita desde el *Play Store*.

Dicha aplicación permite gestionar los paquetes de imágenes en 2 bibliotecas: las imágenes de aplicaciones y las imágenes del gestor de arranque.

Al iniciar la aplicación IOIO Manager, la primera pantalla que aparece es la biblioteca de imágenes de las aplicaciones, tal como se muestra en la Ilustración 3-2.

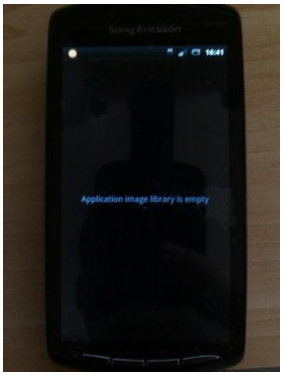


Ilustración 3-2: Pantalla Inicial IOIO Manager

Hay varias maneras de agregar elementos en esta biblioteca. Una de ellas consiste en escanear un código QR (previa instalación de la aplicación Barcode Scanner), seleccionando la tecla menú y pulsando en "Scan QR" para descargar las imágenes a través del código QR en la página de descargas oficiales de IOIO (https://github.com/ytai/ioio/wiki/Downloads).

En la Ilustración 3-3 se muestra un ejemplo de varios elementos agregados en la biblioteca de imágenes de aplicaciones.

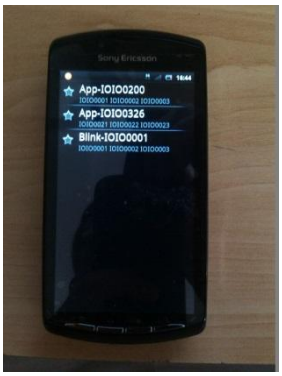


Ilustración 3-3: Biblioteca de Imágenes de Aplicaciones

En la Ilustración 3-4 se puede observar la pantalla de programación, que se accede a ella, seleccionando la tecla menú (en la pantalla principal) y posteriormente pulsando en "*programmer*". Para añadir elementos se ha de acceder previamente a la biblioteca de imágenes del gestor de arranque, a través de "*Select*…" (Situado en la esquina superior derecha de nuestro dispositivo Android) o seleccionando la tecla menú y pulsando en "Library".

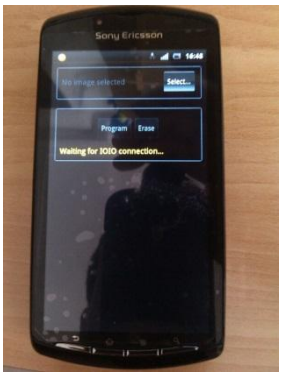


Ilustración 3-4: Pantalla de Programación IOIO Manager

Al igual que en la biblioteca anteriormente comentada, una de las maneras de añadir elementos a la biblioteca de imágenes del gestor de arranque es mediante el escaneo de un código QR, seleccionado la tecla menú y pulsando en "Scan QR".

En la Ilustración 3-5 se observa el modo de conexión para programar/actualizar el *firmware* de la placa IOIO. La imagen seleccionada (para actualizar el *firmware*) dentro de la biblioteca de imágenes del gestor de arranque, ha de ser compatible con nuestro *hardware*, por ejemplo, el micro-controlador PIC24FJ128DA206 requiere la instalación de la imagen SPRK0015. Sin embargo, el micro-controlador PIC24FJ256DA206 requiere la imagen SPRK0016.

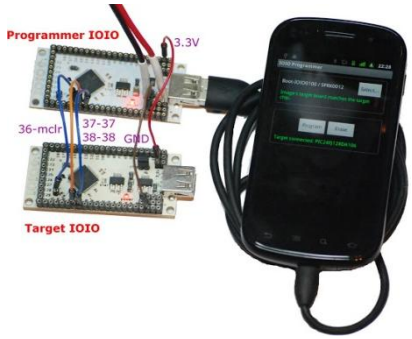


Ilustración 3-5: Conexión Programador

En caso de seleccionar una imagen errónea para el dispositivo móvil inteligente, aparecerá un error tal como se muestra en la Ilustración 3-6.

Ilustración 3-6: *Firmware* Incompatible *Hardware*

En la Ilustración 3-7 se puede observar la pantalla del programador con la selección de una imagen correcta. A continuación, es necesario pulsar sobre "*program*" para comenzar la actualización del *firmware*.

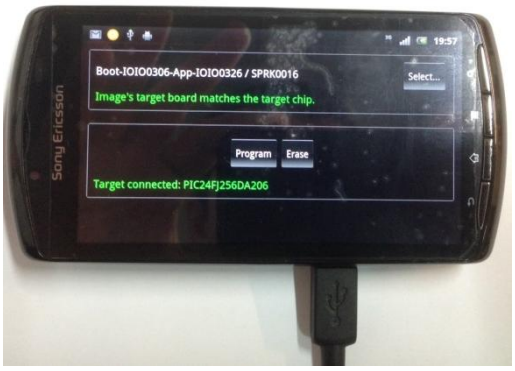


Ilustración 3-7: *Firmware* Compatible *Hardware*

En la Tabla 3-7 se muestra la conexión de los pines necesarios para programar la placa IOIO.

Programmer	Target
38	38
37	37
36	mclr (the isolated pin near pins 33 and 29)
3.3V	3.3V
GND	GND

Tabla 3-7: Conexión Pines Programar IOIO

3.2.4.4 IOIO Hardware Tester

IOIO *Hardware Tester* (ver Ilustración 3-8) es una aplicación para Android. La aplicación está disponible para descargarla de forma gratuita desde el *Play Store*.

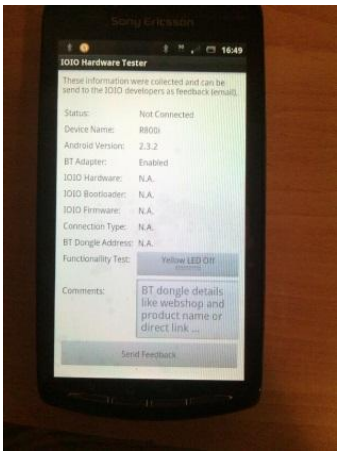


Ilustración 3-8: Aplicación IOIO *Hardware Tester*

Su funcionalidad es informar si el dispositivo móvil inteligente funciona correctamente con la placa IOIO, proporcionando la siguiente información:

- Estado de Conexión.
- Nombre Dispositivo Móvil Inteligente.
- Versión Android de dicho dispositivo.
- Existencia de un módulo *Bluetooth*.
- Versión IOIO *Hardware*.
- Versión IOIO *Bootloader*.

- Versión IOIO *Firmware*.
- Tipo de Conexión.
- Dirección del módulo *Bluetooth*.
- Test de Funcionalidad.

3.2.5 Entorno de Desarrollo

El desarrollo de programas para Android se hace habitualmente con el lenguaje de programación Java y el conjunto de herramientas de desarrollo (SDK, *Software Development Kit*).

El SDK de Android incluye un conjunto de herramientas de desarrollo. Comprende un depurador de código, biblioteca, un simulador de teléfono basado en QEMU, documentación, ejemplos de código y tutoriales.

La plataforma integral de desarrollo (IDE, *Integrated Development Environment*) soportada oficialmente es Eclipse junto con el complemento ADT (*Android Development Tools Plugin*). Eclipse es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido".

Las actualizaciones del SDK están coordinadas con el desarrollo general de Android. El SDK soporta también versiones antiguas de Android, por si los programadores necesitan instalar aplicaciones en dispositivos ya obsoletos o más antiguos. Las herramientas de desarrollo son componentes descargables, de modo que una vez instalada la última versión, pueden instalarse versiones anteriores y hacer pruebas de compatibilidad.

Una aplicación Android está compuesta por un conjunto de ficheros empaquetados en formato ".apk".

3.2.5.1 Instalación del Entorno de Desarrollo

Descargar el paquete ADT en la página oficial de Android *Developer* (http://developer.android.com/sdk/index.html), el paquete incluye todo lo necesario para comenzar a desarrollar aplicaciones para Android.

Descomprimir el archivo en el lugar donde se quiera tener instalado.

Ejecutar el archivo "*eclipse.exe*" para lanzar la aplicación. La primera vez que arranque, solicitará que se le indique un directorio de trabajo donde se guardaran los proyectos (marcando en "*Use this as the default and do not ask again*" no preguntará de nuevo por el directorio de trabajo).

3.2.5.2 Instalación del Plugin ADT

Una vez que aparece en la pantalla principal de Eclipse, se selecciona en "*Help*" > "*Install New Software*", tal como se muestra en la Ilustración 3-9.

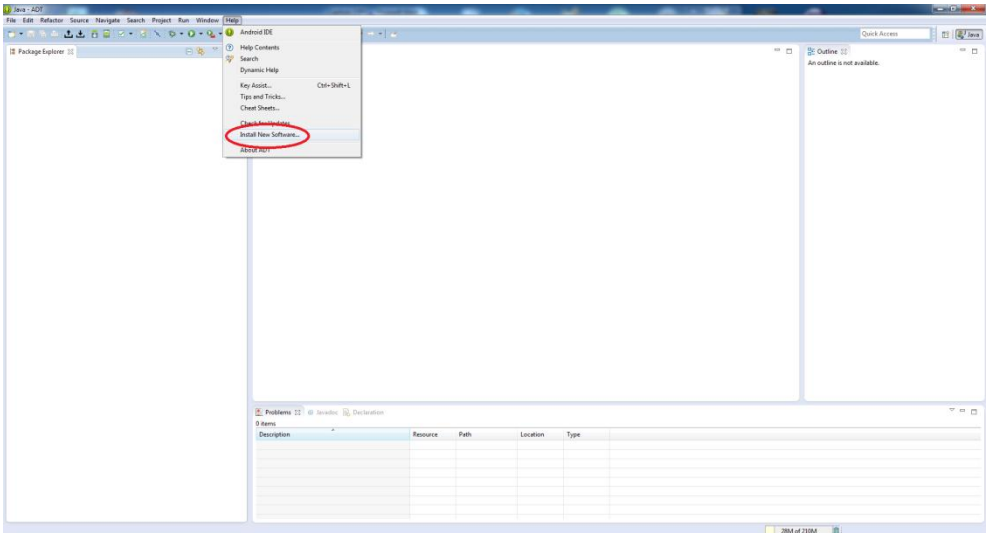


Ilustración 3-9: Instalación Plugin ADT

En esta pantalla, es necesario hacer click en "*Add*" (situado en la esquina superior derecha), y rellenamos los campos con los datos "ADT Plugin" para el nombre y la URL de la localización "https://dl-ssl.google.com/android/eclipse/", tal como se muestra en la Ilustración 3-10.

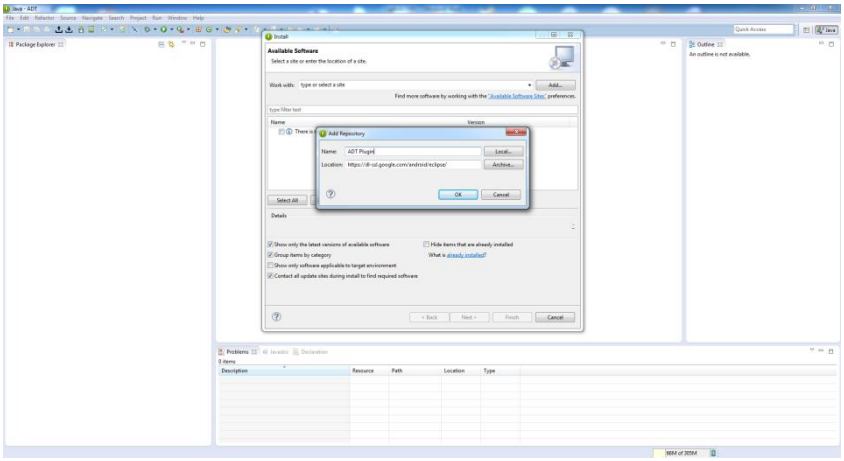


Ilustración 3-10: Instalación Plugin ADT (continuación)

Se seleccionan las herramientas de desarrollo, se aceptan los acuerdos de licencias, etc. Finalmente, una vez terminada la instalación deberá reiniciarse Eclipse.

En las últimas versiones la descarga del paquete ADT, ya incluye la instalación del plugin ADT en Eclipse.

3.2.5.3 Instalación del SDK

En la pantalla principal de Eclipse, se selecciona "*Window*" > "*Preferences*", tal como se muestra en la Ilustración 3-11.

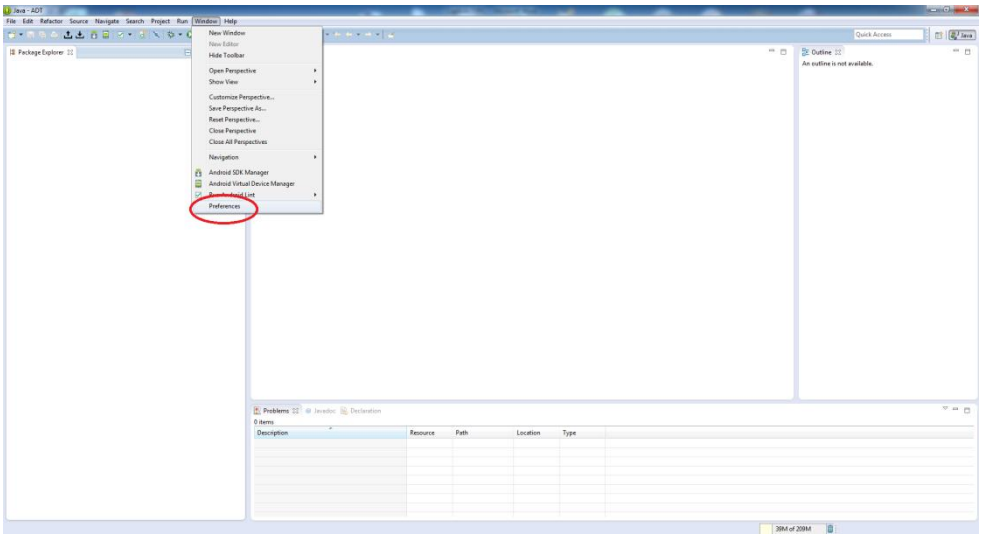


Ilustración 3-11: Instalación SDK

A continuación se selecciona la sección de Android y se busca la localización del SDK, tal como se muestra en la Ilustración 3-12.

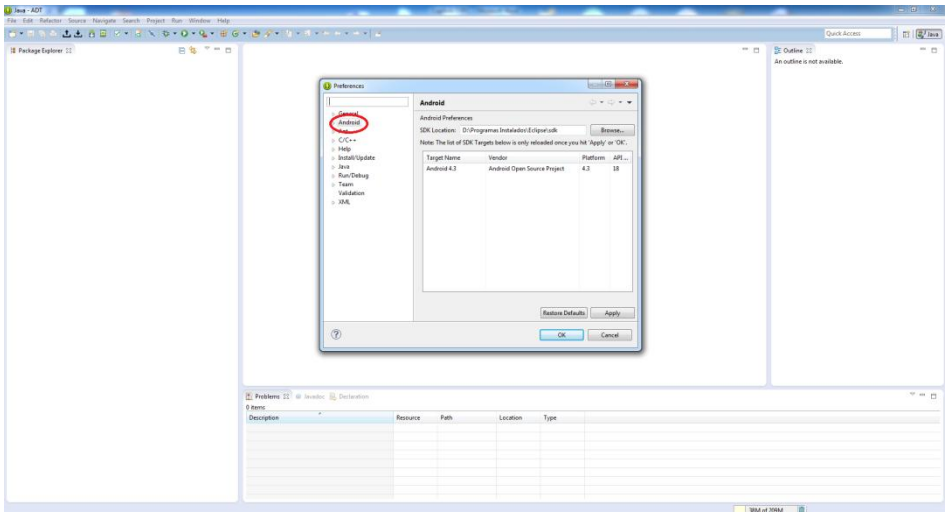


Ilustración 3-12: Instalación SDK (continuación)

Al igual que lo ocurrido en la instalación del plugin ADT, el paquete ADT oficial ya incluye el proceso descrito en este apartado, además de incluir la instalación mínima del SDK necesaria para comenzar a desarrollar aplicaciones en Android.

En la Ilustración 3-13, se muestra una de las formas de acceder al administrador del SDK ("*Window*" > "*Android SDK Manager*").

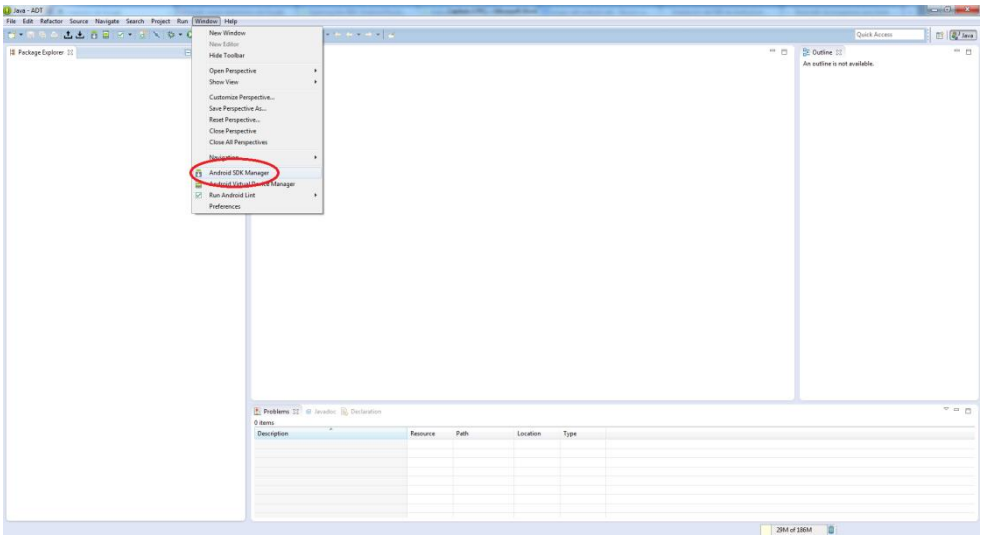


Ilustración 3-13: Acceso Android SDK Manager

El SDK de Android separa herramientas, plataformas y otros componentes en paquetes que se pueden descargar a través del administrador del SDK, tal como se muestra en la Ilustración 3-14.

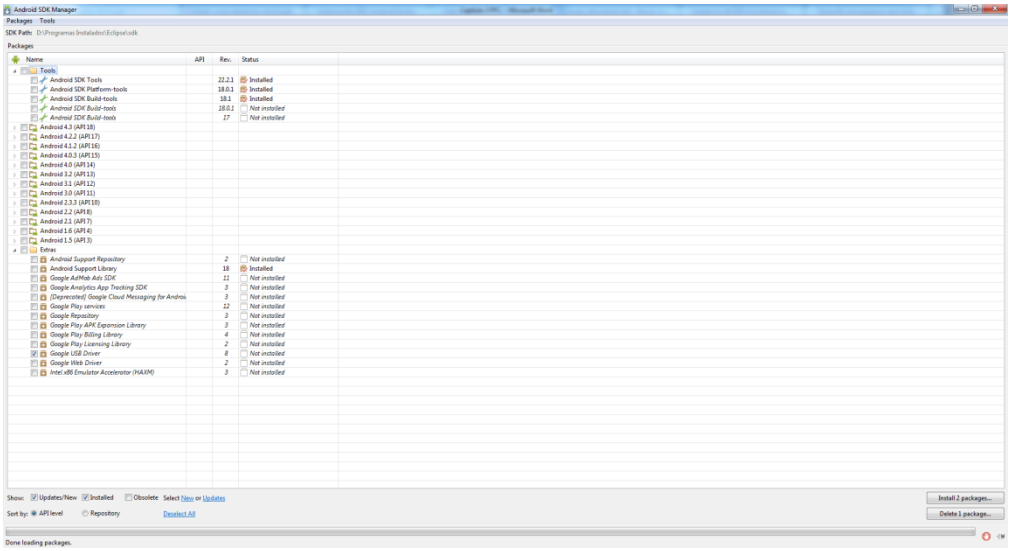


Ilustración 3-14: Android SDK Manager

3.2.5.4 Creación Proyecto IOIO

La comunidad IOIO proporciona una serie de bibliotecas (IOIOLib, IOIOLibBT y IOIOLibAccessory) que permiten el control de la placa IOIO por parte de nuestra aplicación Android. Las bibliotecas exponen un conjunto de interfaces Java, que abarcan las diversas características de la placa IOIO. Al crear la aplicación, IOIOLib se empaqueta en nuestro archivo ".apk", dando independencia a nuestra aplicación que no requiere ninguna instalación adicional.

IOIOLib es la biblioteca principal que se utiliza para la creación de la interfaz entre nuestra aplicación Android y la placa IOIO. IOIOLibBT y IOIOLibAccessory son bibliotecas adicionales para la comunicación inalámbrica Bluetooth y el protocolo de "*Open Accessory*" de Android, respectivamente.

Dichas bibliotecas se encuentran separadas ya que cada una de ellas requiere una versión de Android distinta, IOIOLib funciona en cualquier versión de Android (v1.5 en adelante), IOIOLibBT se ha introducido para v2.x de Android en adelante y el protocolo "*Open Accessory*" está disponible en determinados dispositivos con versión de Android 2.3.4 y dispositivos posteriores.

Además del *software*, la descarga incluye alguna aplicación a modo de ejemplo.

Las bibliotecas de IOIO han de ser importadas a nuestro proyecto para poder utilizarlas en nuestra aplicación Android. Para ello, una vez descargadas (https://github.com/ytai/ioio/wiki/Downloads), seleccionamos en nuestro entorno de desarrollo Eclipse en "*File*" > "*Import*", tal como se muestra en la Ilustración 3-15.

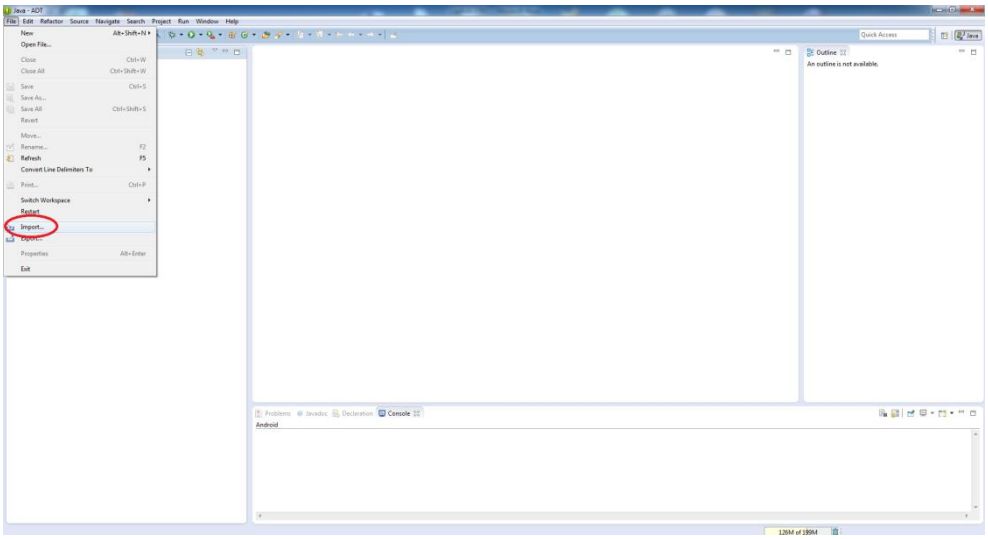


Ilustración 3-15: 1º Parte Importación Bibliotecas IOIO

A continuación, seleccionamos en "*General*" > "*Existing Projects into Workspace*" > "*Next*", tal como se muestra en la Ilustración 3-16.

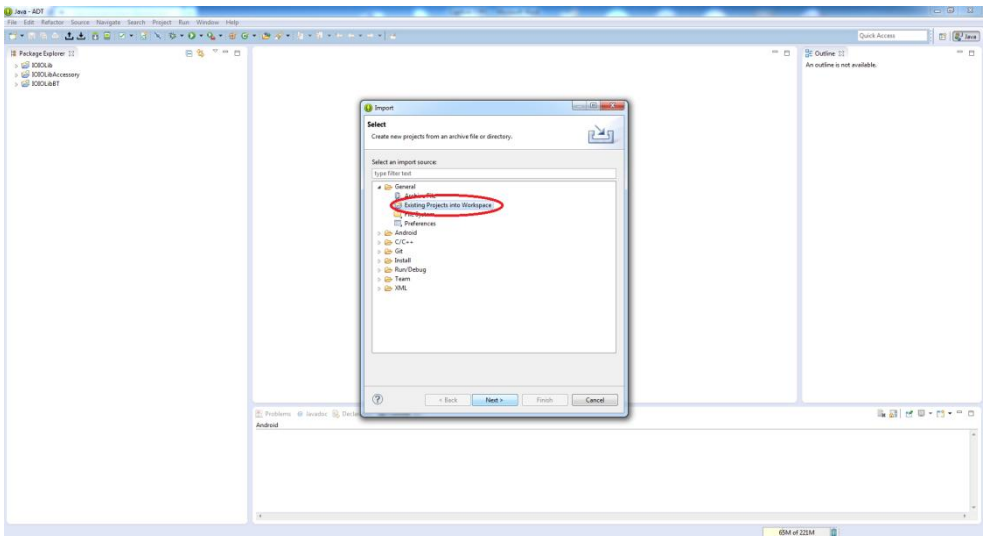


Ilustración 3-16: 2º Parte Importación Bibliotecas IOIO

Seguidamente, seleccionamos el directorio donde tenemos descargadas las librerías de IOIO y los proyectos (que en este caso son las librerías) que deseamos incluir, y por último pulsamos en "*Finish*".

Cabe mencionar que a la hora de importar las librerías de IOIO a Eclipse, se pueden producir diversos errores, dichos errores suelen estar descritos en la Wiki de IOIO (https://github.com/ytai/ioio/wiki/Eclipse-Troubleshooting), así como la solución a los mismos.

Una vez importadas las librerías IOIO y creado nuestro proyecto de aplicaciones Android (en blanco), podemos comenzar con la programación de nuestra aplicación. El punto de partida es crear una instancia con la interfaz IOIO, para ello hay 2 caminos posibles, una configuración "manual" con las funciones que se proporcionan en la Wiki de IOIO o utilizar un marco de trabajo proporcionado por la comunidad IOIO.

La recomendación por parte de la comunidad IOIO es utilizar el marco de trabajo, ya que en él, se preveen ciertas complejidades que en la configuración "manual" no se tienen en cuenta, favoreciendo el correcto funcionamiento entre la aplicación Android y la placa IOIO.

A continuación se muestra el marco de trabajo recomendado para la realización de nuestra aplicación Android:

```
public class MainActivity extends IOIOActivity
{

	@Override
	protected void onCreate(Bundle savedInstanceState)
	{
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);
	}

	class Looper extends BaseIOIOLooper
	{
		@Override
		protected void setup() throws ConnectionLostException
		{
		}

		@Override
		public void loop() throws ConnectionLostException
		{
		}
	}

	@Override
	protected IOIOLooper createIOIOLooper()
	{
		return new Looper();
	}

}
```

Además de esto, como mínimo en nuestra aplicación Android debe declararse el permiso "*android.permission.INTERNET*". Esto se configura abriendo el archivo "*AndroidManifest.xml*", pestaña "*Permissions*" > "*Add*" > "*Uses Permission*" > y seleccionar en nombre "*android.permission.INTERNET*", tal como se muestra en la Ilustración 3-17.

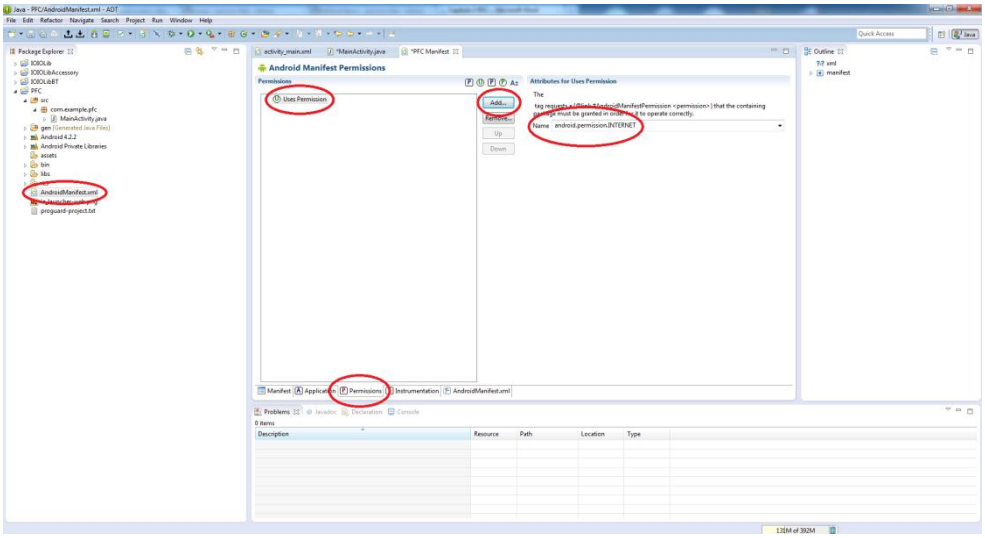


Ilustración 3-17: Declaración Permisos Android

3.3 Arduino

En esta sección se va a describir tanto el *hardware* como el *software* de Arduino UNO [10], así como el entorno de desarrollo necesario para la realización de un proyecto con Arduino.

Arduino es una plataforma de desarrollo de computación física (*physical computing*) de código abierto, basada en una placa con un sencillo micro-controlador y un entorno de desarrollo para crear *software* (programas) para la placa.

Arduino se usa para crear objetos interactivos, leyendo datos de una gran variedad de interruptores y sensores y controlar multitud de luces, motores y otros actuadores físicos. Los proyectos de Arduino pueden ser autónomos o comunicarse con un programa (*software*) que se ejecuta en un ordenador.

El lenguaje de programación de Arduino es una implementación de *Wiring*, una plataforma de computación física parecida, que a su vez se basa en *Processing*, un entorno de programación multimedia.

Algunas de las ventajas de Arduino respecto a otros sistemas son las siguientes:

- Asequible: Las placas Arduino son más asequibles comparadas con otras plataformas de micro-controladores.
- Multi-Plataforma: El software de Arduino funciona en los sistemas operativos Windows, Macintosh OSX y Linux.
- Entorno de programación simple y directo: El entorno de programación de Arduino es fácil de usar para principiantes y lo suficientemente flexible para los usuarios avanzados.
- *Software* ampliable y de código abierto: El *software* Arduino está publicado bajo una licencia libre y preparado para ser ampliado por programadores experimentados. El lenguaje puede ampliarse a través de librerías de C++.
- *Hardware* ampliable y de código abierto: Arduino está basado en los micro-controladores ATMEGA168, ATMEGA328 y ATMEGA1280. Los planos de los módulos están publicados bajo licencia *Creative Commons*, por lo que los diseñadores de circuitos con experiencia pueden hacer su propia versión del módulo, ampliándolo u optimizándolo.

3.3.1 Hardware Arduino UNO

3.3.1.1 Vista Conjunta

Ilustración 3-18: Vista Conjunta Arduino UNO

La placa de Arduino UNO está basada en el micro-controlador ATMEGA328. Tiene 14 pines digitales de entrada/salida (de los cuales 6 se pueden utilizar como salidas PWM), 6 entradas analógicas, una conexión USB utilizada para subir programas a la placa y para la comunicación serie entre la placa y el ordenador (puede utilizarse como alimentación a la placa), un conector Jack de alimentación externa, un programador serie en circuito "*In-circuit Serial Programmer*" o "ICSP" y un botón de reinicio.

Incluye 2 pines (SDA y SCL) localizados cerca del pin AREF (terminal de referencia analógica), internamente un selector de alimentación entre el USB y el conector de alimentación externa. El pin IOREF permite que los *Shields* se adapten al voltaje empleado por la placa y el pin reset utilizado para añadir un botón de reset a los *Shields* que bloquean el de la placa principal.

3.3.1.2 Resumen

Micro-controlador	ATMEGA328
Voltaje Funcionamiento	5 V
Voltaje Entrada (Recomendado)	7 – 12 V
Voltaje Entrada (Límites)	6 – 20 V
Pines E/S Digitales	14 (6 proporcionan salida PWM)
Pines Entrada Analógica	6
Corriente Pines E/S	40 mA
Corriente Pin 3.3 V	50 mA

Memoria Flash	32 KB (0.5 KB para el *bootloader*)
SRAM	2 KB
EEPROM	1 KB
Velocidad Reloj	16 MHz

3.3.1.3 Memoria

El ATMEGA328 tiene 32 KB de Memoria Flash para almacenar código (de los cuales 0.5 KB se usan para el *bootloader*). Tiene 2 KB de SRAM y 1 KB de EEPROM (que puede ser leída y escrita con la librería EEPROM).

3.3.1.4 Funciones Pines E/S

Cada uno de los 14 pines digitales de Arduino UNO puede ser utilizados como entrada o salida, utilizando las funciones "*digitalRead()*", "*digitalWrite()*" y "*pinMode()*". Funcionan a 5 V. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia pull-up interna (desconectada por defecto) de 20 – 50 KOhm. Además, algunos pines tienen funciones especiales:

Serie: Pines 0 (RX) y 1 (TX). Se utiliza para recibir (RX) y de transmisión (TX) de datos en serie TTL.

Interrupciones Externas: Pines 2 y 3. Estos pines pueden ser configurados para activar una interrupción en un valor bajo, un flanco ascendente o descendente, o un cambio de valor.

PWM: Pines 3, 5, 6, 9, 10 y 11. Proporcionan salida PWM de 8 bits.

SPI: Pines 10 (SS), 11 (MOSI), 12 (MISO) y 13 (SCK). Estos pines soportan la comunicación SPI usando la librería SPI.

Arduino UNO tiene 6 entradas analógicas, numeradas A0 – A5, cada una de las cuales proporcionan 10 bits de resolución, es decir, 1024 valores diferentes.

TWI: Pines A4 o SDA y pines A5 o SCL. Soporta comunicación TWI usando la librería Wire.

AREF: Voltaje de referencia para entradas analógicas.

Reset: Pin para resetear el micro-controlador, normalmente se utiliza para agregar un botón de reinicio para los *Shields* que bloquean la que está en la placa principal.

3.3.1.5 Comunicación

El Arduino UNO tiene un número de infraestructuras para comunicarse con un ordenador, otro Arduino, u otros micro-controladores. El ATMEGA328 provee comunicación serie UART TTL (5 V), la cual está disponible en los pines digitales 0 (Rx) y 1 (Tx). Un FTDI FT232RL en la placa canaliza esta comunicación serie al USB y los drivers FTDI (incluidos con el *software* Arduino) proporcionan un puerto de comunicación virtual al *software* del ordenador. El *software* Arduino incluye un monitor serie que permite a datos de texto simple ser enviados a y desde la placa Arduino.

El ATMEGA328 tambien soporta comunicación I2C (TWI) y SPI.

3.3.1.6 Programación

El Arduino UNO puede ser programado con el *software* Arduino vía USB.

El ATMEGA328 del Arduino UNO viene con un *bootloader* pregrabado que te permite subirle nuevo código sin usar un programador *hardware* externo. Se comunica usando el protocolo original STK500. Es algo no habitual en otros tipos de micro-controladores que generalmente requieren un programador externo.

Tambien puedes saltar el *bootloader* y programar el ATMEGA328 a través de la cabecera ICSP (*In-Circuit Serial Programming*).

3.3.1.7 Protección de Sobrecarga del USB

El Arduino UNO tiene un fusible reseteable que protege los puertos USB del ordenador de cortes y sobrecargas. Aunque la mayoría de los ordenadores proporcionan su propia protección interna, el fusible proporciona una capa de protección extra. Si se aplican más de 500 mA al puerto USB, el fusible automáticamente romperá la conexión hasta que el corte o la sobrecargar sean eliminados.

3.3.2 Alimentación

El Arduino UNO puede ser alimentado a través de la conexión USB o con una fuente de alimentación externa, la selección se hace de manera automática internamente.

La placa puede funcionar con un suministro externo de 6 - 20 V. Si se proporcionan menos de 7 V, el pin de 5 V puede suministrar menos de 5 V y la placa puede ser inestable. Si se alimenta con más de 12 V, el regulador de voltaje se puede sobrecalentar y dañar la placa. El rango recomendado es de 7 V – 12 V.

Vin: Se utiliza para la alimentación de la placa. El voltaje suministrado debe ser entre 7 V – 12 V. En caso de alimentar a la placa mediante el conector de alimentación externo se puede utilizar este pin como salida.

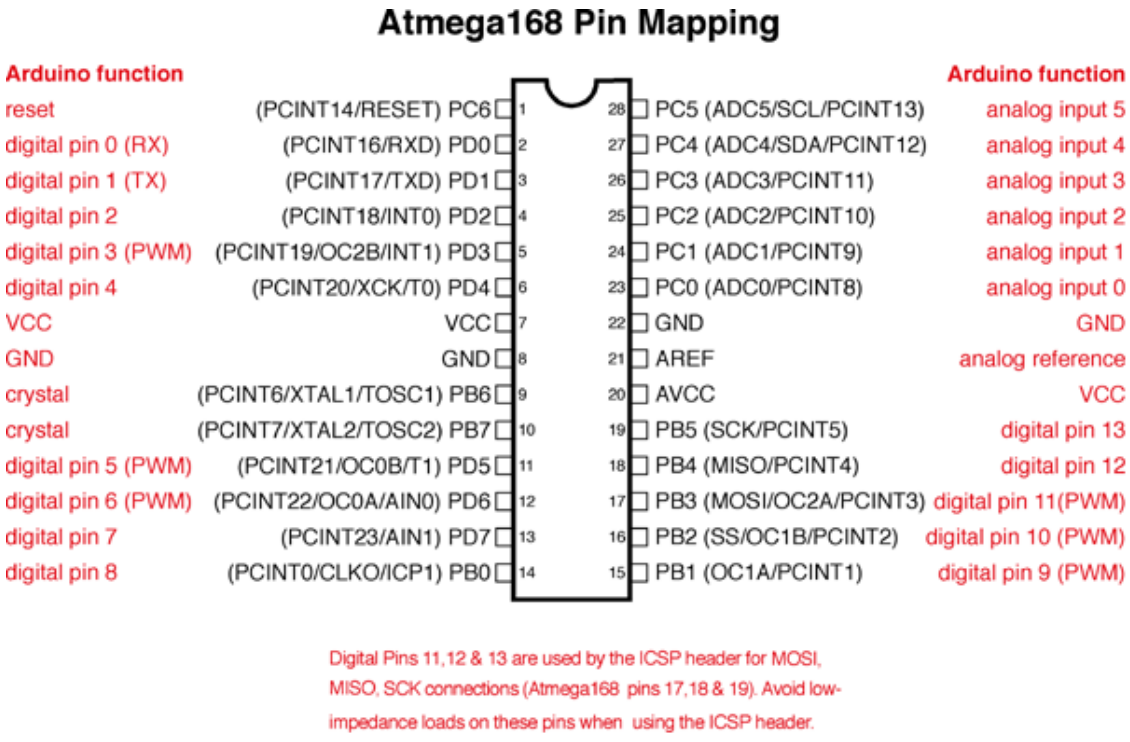
5 V: Salida a 5 V. No se recomienda alimentar la placa a través de este pin, ya que no pasa por el regulador y puede dañar la placa.

3.3 V: Salida a 3.3 V, 50 mA de corriente máxima.

GND: Pines de tierra.

IOREF: Este pin de la placa Arduino UNO proporciona la referencia de tensión con la que opera el micro-controlador a los *Shields*.

3.3.3 Mapeado entre ATMEGA168/328 y Arduino



3.3.4 Software

3.3.4.1 Introducción

El entorno de código abierto Arduino hace fácil escribir código y cargarlo a la placa. Es un entorno multiplataforma, pudiendo ser utilizado en Windows, Macintosh y Linux.

El entorno de desarrollo (IDE, *Integrated Development Environment*) está escrito en Java y basado en *Processing*, *avr-gcc* y otros programas también de código abierto.

Es un entorno fácil de usar, es simple y directo, dando flexibilidad tanto a usuarios principiantes como avanzados.

La programación de la placa de Arduino se hace vía USB, contando con una amplia y activa comunidad de usuarios.

Se puede descargar libremente desde la página oficial de Arduino (http://arduino.cc/en/Main/Software) y no requiere instalación.

3.3.4.2 Bibliotecas

Arduino ofrece una serie de bibliotecas "estándar" basadas en C/C++ que se pueden importar al Sketck, las bibliotecas "estándar" son las siguientes:

- EEPROM: Para leer y escribir en memorias "permanentes".
- Ethernet: Para conectar a internet usando el Ethernet Shield.
- Firmata: Para comunicarse con aplicaciones en la computadora usando un protocolo estándar Serial.
- LiquidCrystal: Para controlar Displays de cristal líquido (LCD).
- Servo: Para controlar servomotores.
- SoftwareSerial: Para la comunicación serial de cualquier pin digital.
- Stepper: Para controlar motores paso a paso.
- Wire: Interfaz de dos cables (TWI/I2C), para enviar y recibir datos a través de una red de dispositivos y sensores.

3.3.4.3 Diseño Sketch

El diseño de un Sketch para Arduino no requiere de avanzados conocimientos en electrónica y en programación. Así, por ejemplo, para configurar una línea digital como salida se puede utilizar la siguiente función "pinMode(Pin13, OUTPUT)". De mismo modo, para escribir un valor alto o bajo en dicho pin se utiliza la siguiente función "digitalWrite(Pin13, HIGH o LOW)".Ademas, para realizar tareas más complejas, tales como gestionar una comunicación por UART, se pueden utilizar las siguientes funciones "Serial.begin(speed)", "Serial.read()", "Serial.available()", etc…

La estructura básica del lenguaje de programación Arduino es bastante simple y se organiza en al menos dos partes o funciones que encierran bloques de declaraciones.

```
void setup()
{
    Statements;
}
void loop()
{
    Statements;
}
```

Ambas funciones son requeridas para que el programa funcione.

- setup(): La función "setup" debería contener la declaración de cualquier variable al comienzo del programa. Es la primera función a ejecutar en el programa, es ejecutada una vez y usada por ejemplo para inicializar las comunicaciones serie.

- loop(): La función "loop" se ejecuta a continuación e incluye el código que se ejecuta continuamente leyendo entradas, activando salidas, etc. Esta función es el núcleo de todos los programas Arduino y hace la mayor parte del trabajo.

3.3.5 Entorno de Desarrollo

Como ya se ha mencionado anteriormente, el entorno de desarrollo de Arduino es gratis y no requiere instalación (http://arduino.cc/en/Main/Software).

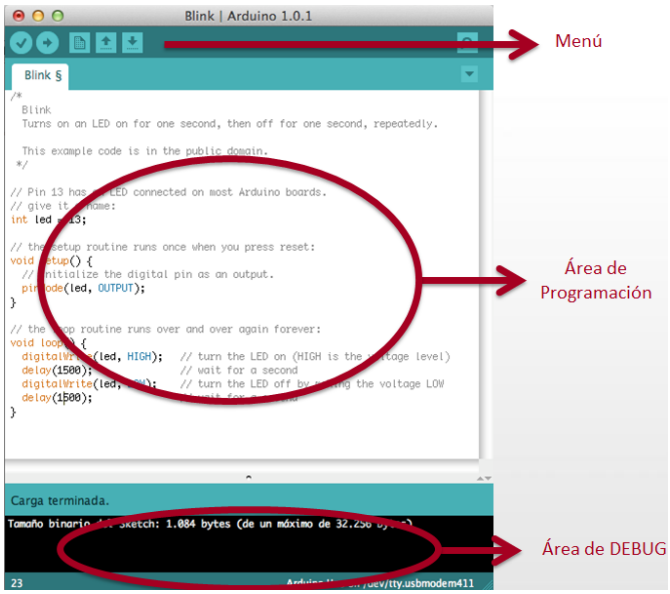


Ilustración 3-19: Entorno de Desarrollo Arduino

En el siguiente enlace se encuentra toda la información necesaria para la instalación del software en la plataforma Windows, Mac o Linux (http://arduino.cc/es/Guide/HomePage).

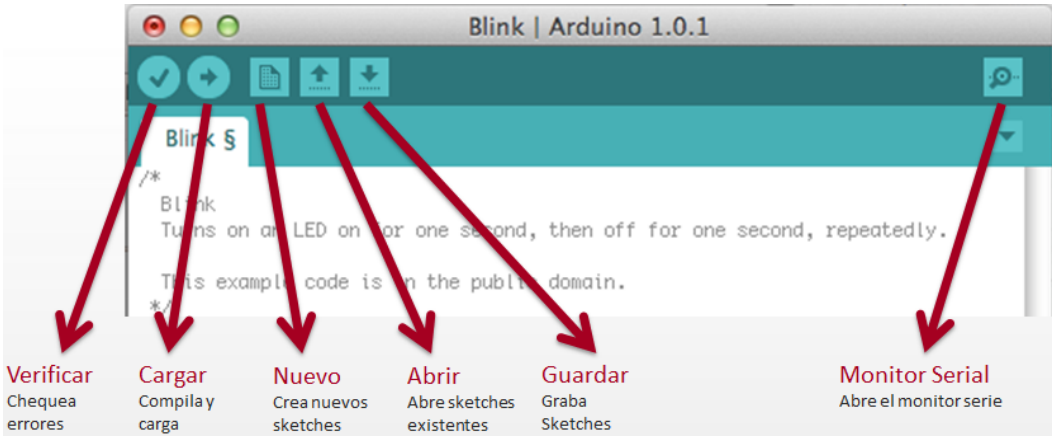


Ilustración 3-20: Menú Entorno de Desarrollo

El software desarrollado con Arduino se conoce como sketches. Los sketches se escriben con un editor de texto y son guardados con la extensión ".ino".

En la Ilustración 3-21, se muestra como seleccionar el puerto.

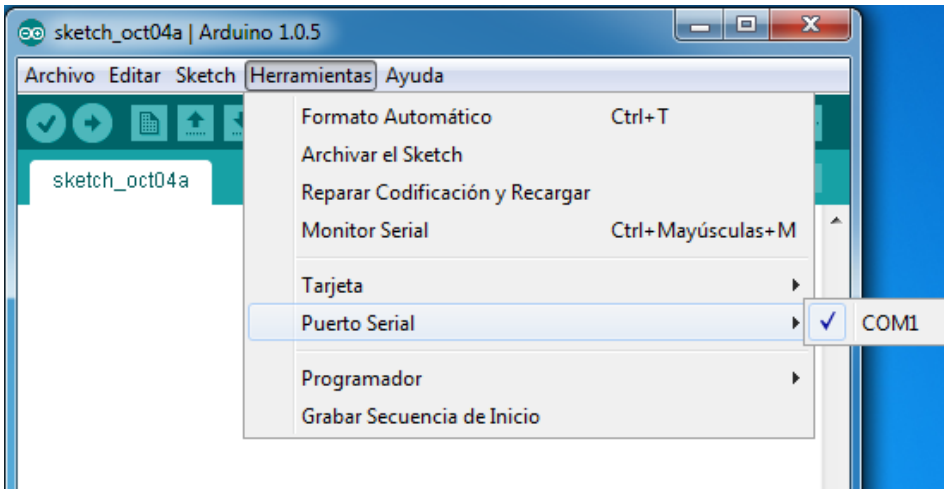


Ilustración 3-21: Menú de selección de puerto del Entorno Arduino

En la Ilustración 3-22, se muestra como seleccionar la placa de Arduino con la que estemos trabajando.

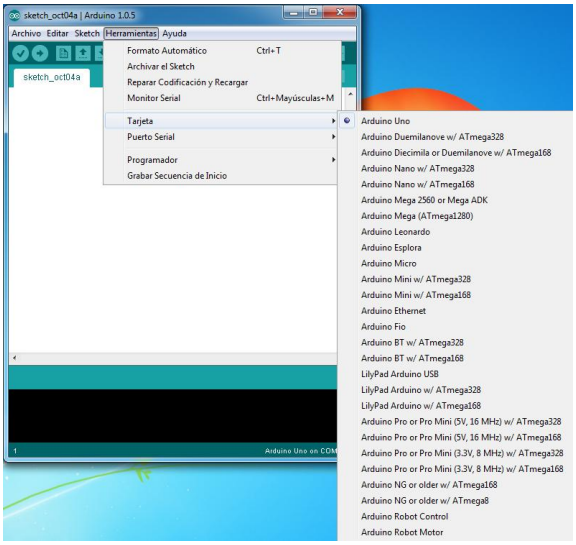


Ilustración 3-22: Menú de selección de placa del Entorno Arduino

3.3.6 Shields

Las Shields son placas que pueden ser conectadas encima de la placa Arduino extendiendo sus capacidades. Para ello los pines de sus puertos guardan una disposición de compatibilidad.

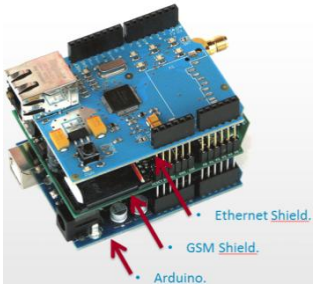


Ilustración 3-23: Arduino & Shields

Las diferentes Shields siguen la misma filosofía que el conjunto original: son fáciles de montar y baratas de producir.

Existe una gran variedad de Shields con diversa funcionalidad: control de motores, comunicaciones, prototipado rápido, etc…Ademas, de incluir una biblioteca software con objeto de poder desarrollar un prototipo con poco tiempo de diseño invertido.

Capítulo 4

Descripción del *Hardware* y *Software* Desarrollado

4.1 Introducción

En el capítulo anterior se describió tanto el *hardware* como el *software* necesario para la realización de este Proyecto Fin de Carrera.

En este capítulo se detallará la implementación del *hardware* y el *software* desarrollado para culminar la Arquitectura y Diseño de Accesorios Electrónicos Inalámbricos para Dispositivos Móviles Inteligentes.

4.2 Hardware

En este apartado se describirán las conexiones realizadas entre los distintos elementos *hardware* que componen el *Gadget*. La alimentación del *Gadget* se realiza mediante un transformador con conector *jack* de 12 V a la placa Arduino UNO, a través del pin 5 V de Arduino se alimenta tanto el sensor de temperatura como la placa IOIO y a través del pin 3,3 V se alimenta la tarjeta SD.

4.2.1 Conexión Arduino UNO / SD

En la Ilustración 4-1 se muestra la conexión realizada entre Arduino UNO y la tarjeta SD. Como se observa ha sido necesario la creación de una tarjeta con el zócalo de la SD, las resistencias necesarias para el divisor de tensión por incompatibilidad en el voltaje entre Arduino UNO y la SD, las salidas de Arduino UNO son de 5 V y la SD requiere 3 V, y el conector que permite la conexión entre Arduino UNO y la placa SD usando cables especiales.

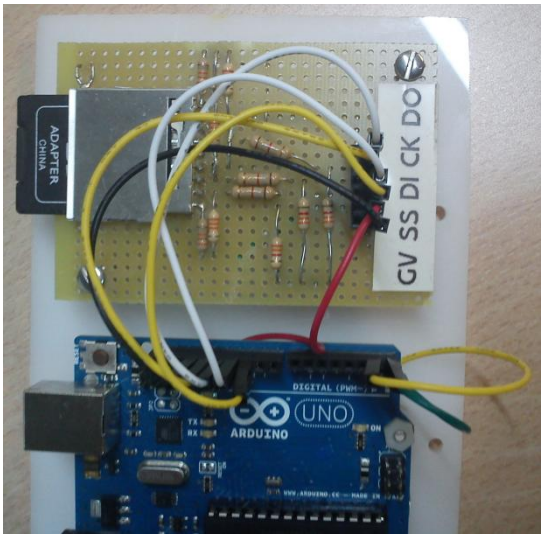


Ilustración 4-1: Conexión Arduino UNO / SD

Esta conexión se realiza mediante interfaz periférico serie (SPI), es un protocolo de datos serie síncrono utilizado por micro-controladores para comunicarse con uno o varios dispositivos periféricos de una forma rápida y a través de una distancia corta.

En la Ilustración 4-2 se muestra el esquemático de la interfaz SPI entre la tarjeta SD y el micro-controlador.

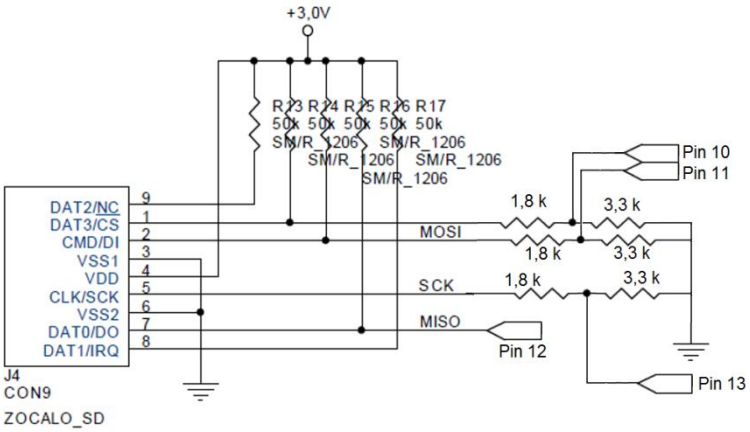


Ilustración 4-2: Esquemático de la interfaz SPI entre la tarjeta SD y el micro-controlador

MISO (*Master In Slave OUT*): Línea del esclavo para enviar datos al maestro.

MOSI (*Master In Slave In*): Línea del maestro para enviar datos a los periféricos.

SCK (*Serial Clock*): Los pulsos de reloj generados por el maestro para sincronizar la transmisión de datos.

SS (*Slave Select Pin*): Línea utilizada por el maestro para habilitar o inhabilitar la comunicación.

En la Ilustración 4-3 se muestran los pines utilizados para dicha conexión.

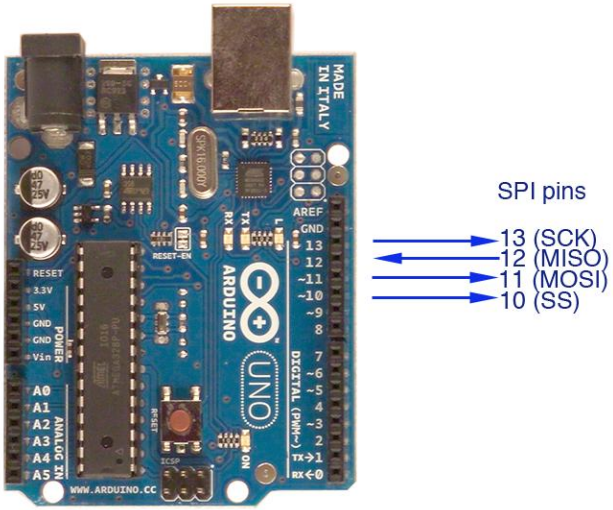


Ilustración 4-3: Pines SPI Arduino UNO

4.2.2 Conexión Arduino UNO / Sensor

Además de las conexiones realizadas entre Arduino UNO y la SD descritas anteriormente, Arduino UNO tiene conectado un sensor de temperatura, en concreto el modelo MCP9700 de Microchip, tal como se muestra en la Ilustración 4-4.

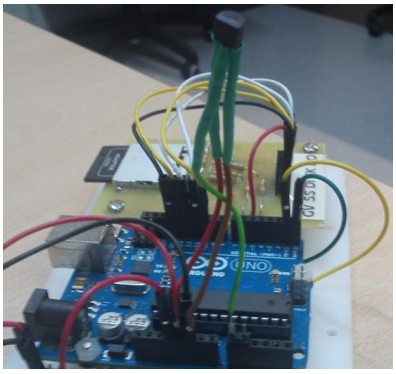


Ilustración 4-4: Conexión Arduino UNO / Sensor MCP9700

En la Ilustración 4-5 se muestra una imagen de dicho sensor. El rango de voltaje de alimentación del sensor es de 2,3 – 5,5 V, proporcionando una salida de 10 mV / ºC. Las patillas del sensor corresponden a la alimentación (patilla 1), salida del sensor (patilla 2) y masa (patilla 3).

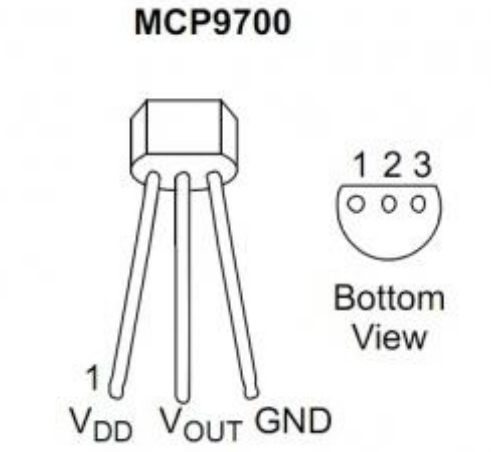


Ilustración 4-5: Sensor MCP9700 Microchip

Para la alimentación del sensor se utiliza el pin de alimentación de 5 V de Arduino UNO y para la recepción de los datos del sensor, una de las seis entradas analógicas disponibles en Arduino UNO, en este caso la entrada A0.

4.2.3 Conexión Arduino UNO / IOIO

Existen diferentes métodos para la comunicación entre Arduino UNO / IOIO, uno de los métodos disponibles y el utilizado en este caso, es mediante UART.

UART son las siglas de "*Universal Asynchronous Receiver-Transmitter*". Es una interfaz digital de comunicación serie asíncrona, además de las líneas de control, consta principalmente de los pines Tx y Rx, dichos pines permiten transmitir y recibir datos respectivamente. Al disponer de dos líneas de comunicación, estas son de tipo *full dúplex*, es decir, se puede enviar y recibir datos al mismo tiempo.

La particularidad de este tipo de conexión es que en ambos dispositivos se debe mantener el mismo tipo de configuración en cuanto a bits de inicio, bits de parada, velocidad y paridad.

La UART toma bytes de datos y transmite los bits individuales de forma secuencial, en el destino, un segundo UART reensambla los bits en bytes completos.

En la Tabla 4-1 se muestra este tipo de conexión, en este caso la configuración de la UART utilizada es de 1 bit de inicio, 1 bit de parada, 9600 bps de velocidad y sin paridad.

PIN IOIO	PIN ARDUINO	DESCRIPCIÓN
4	0	Salida IOIO (Tx)/ Entrada Arduino (Rx)
3	1	Entrada IOIO (Rx)/ Salida Arduino (Tx)

Tabla 4-1: Conexión UART entre Arduino UNO / IOIO

Al contrario que en Arduino UNO, IOIO permite configurar varias UART. Como se ha comentado anteriormente, en este caso solo se ha configurado una UART en los pines 3 y 4.

4.2.4 Conexión IOIO / Dispositivo Móvil Inteligente

IOIO permite dos tipos de conexión, cableada e inalámbrica (*Bluetooth*). Como se comentó en el capítulo 2 aunque IOIO no integra un módulo *Bluetooth*, permite este tipo de conexión mediante un *dongle* Bluetooth en el *host* USB, tal como se muestra en la Ilustración 4-6.

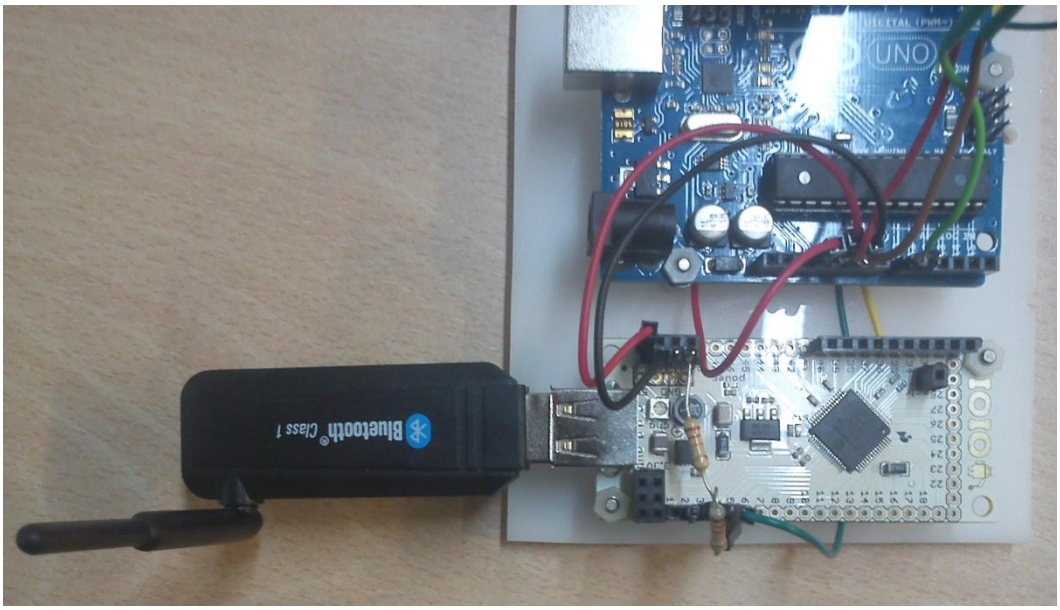


Ilustración 4-6: Conexión IOIO / Dispositivo Móvil Inteligente (*Bluetooth*)

Cabe destacar que IOIO incluye un regulador de corriente en la carga y que es necesario que ofrezca la alimentación necesaria para el correcto funcionamiento del módulo Bluetooth, tal como se muestra en la Ilustración 4-7.

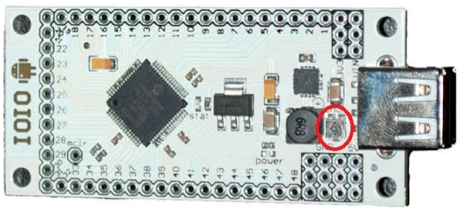


Ilustración 4-7: Regulador Corriente IOIO

4.2.5 Vista Conjunta

El esquema de conexión de todos los elementos descritos anteriormente de forma individual, se muestra en la Ilustración 4-8.

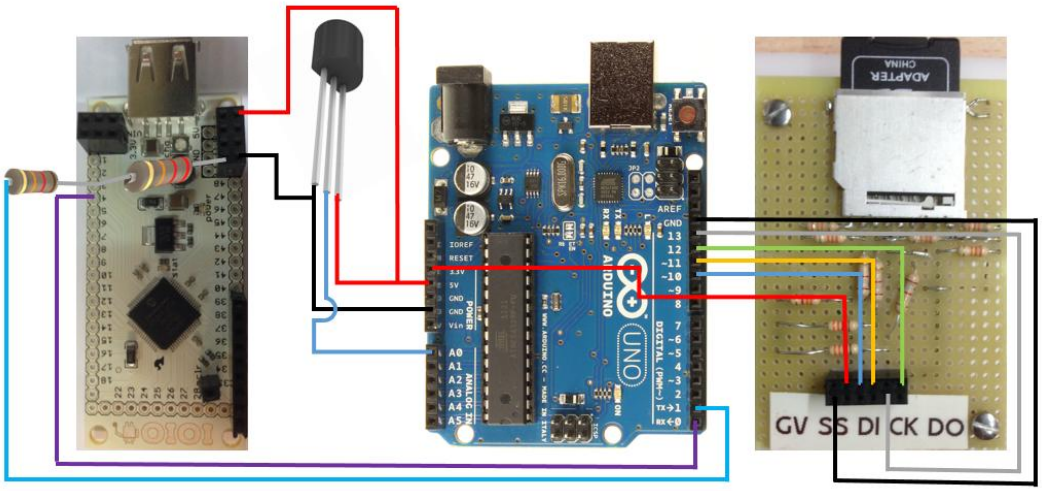


Ilustración 4-8: Esquema de Conexión Completo del *Gadget* basado en la placa IOIO

La vista del sistema para diseñar accesorios electrónicos inalámbricos para dispositivos móviles inteligentes es la mostrada en la Ilustración 4-9, actualmente dicha arquitectura solo tiene conectado un sensor de temperatura tal como se ha comentado anteriormente. Resultaría sencillo añadir la conexión de sensores y actuadores a dicha arquitectura.

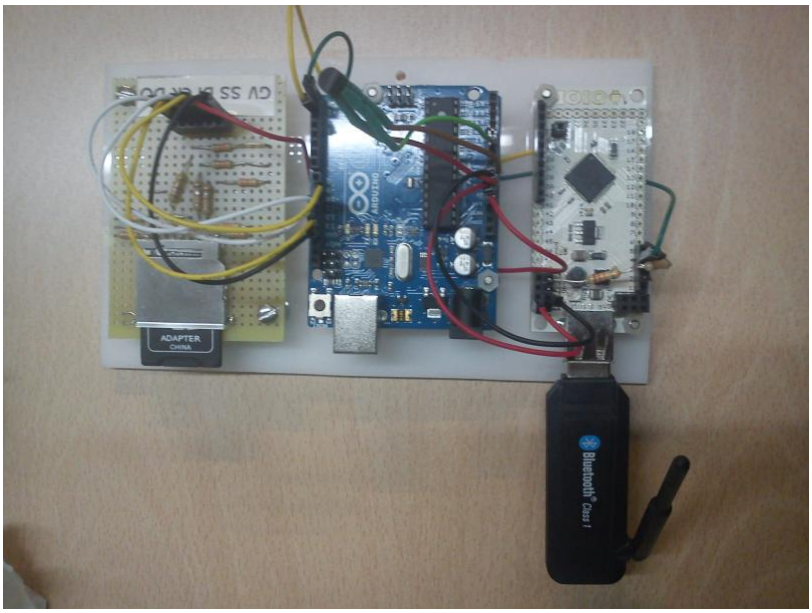


Ilustración 4-9: Vista Conjunta Sistema

4.3 Software

En este apartado se describirán el Sketch Arduino y la aplicación Android realizada, así como su código fuente.

4.3.1 Sketch Arduino

Como se comentó en el capítulo 3, la estructura básica de un Sketch de Arduino se basa en al menos 2 funciones setup() y loop().

En el Listado 4-1 se muestra la inicialización del Sketch que se ha desarrollado para la placa Arduino UNO.

```
void setup(){
  Serial.begin(9600);
  previousMillis = millis();
  pinMode(10, OUTPUT);

  if (!SD.begin(10)) {
    return;}}
```

Listado 4-1: Inicialización Sketch Arduino

Dicha inicialización consiste en habilitar la UART a 9600 bps, el programa requiere de temporización y para ello se ha utilizado la función "millis()" que devuelve el tiempo transcurrido en milisegundos desde que se arrancó la placa Arduino UNO con el programa actual, por ello se ha inicializado la variable "previousMillis" con dicho valor, también se ha establecido el pin digital 10 como salida (necesario para habilitar la tarjeta SD) e inicializado la interfaz SPI entre dicha tarjeta y Arduino UNO.

Para la UART se ha habilitado el uso de interrupciones por hardware, en el Listado 4-2 se muestra dicha estructura.

```
void serialEvent(){
  if (!block){
    char ch;
    while (Serial.available()){
      delay(2);
      ch = Serial.read();
      if (ch == ';'){
        startCharacter = true;
        bytesReceived = 0;}

      if ((ch != '\n') && startCharacter){
        dataArray[bytesReceived] = ch;
        bytesReceived++;
        continue;}
      dataArray[bytesReceived] = '\0';
      break;
    }
    if((bytesReceived > 0) && (dataArray[bytesReceived] == '\0')) dataReceived = true;
  }
}
```

Listado 4-2: Estructura Interrupciones *Hardware* UART

Para gestionar la sección de datos de la UART se ha utilizado la función "serialEvent()", dicha función se ejecuta cuando se recibe un byte por la UART, almacenando dicho byte en la variable de tipo char denominada "ch". Si el inicio de dicha trama corresponde al establecido (';'), dicho byte y los sucesivos se almacenan en otra variable de tipo char denominada "dataArray" hasta recibir el fin de trama (carácter retorno de carro). Una vez finalizado el proceso habilitar una variable de tipo booleano llamada "dataReceived".

En función de la trama recibida, posteriormente se ejecutara dentro del loop() unas acciones u otras.

Las acciones existentes en el loop() son las siguientes: Solicitud por parte de Android a Arduino sobre la fecha y hora actual de Arduino, sincronización del dispositivo Arduino con Android (creándose un fichero para la lectura de datos del sensor en la SD), solicitud por parte de Android a Arduino sobre el tiempo de temporización del sensor, establecimiento del tiempo de temporización del sensor y respuesta a la petición de información de la SD por parte de Android.

A modo de ejemplo se explica en detalle la acción de sincronización del dispositivo Arduino con Android.

```
if (dataReceived && (strstr(dataArray, ";TT"))){
    dataReceived = false;
    block = true;

    for (unsigned int i = 0; i < 19; i++){
      reply[i] =  dataArray[i+3];}
    Serial.write(reply);

    readingDay = ((dataArray[3] - 48) * 10) + (dataArray[4] - 48);
    readingMonth = ((dataArray[6] - 48) * 10) + (dataArray[7] - 48);
    readingYear  =  ((dataArray[9]  -  48)  *  1000)  +  ((dataArray[10]  -  48)  *  100)  +
((dataArray[11] - 48) * 10) + (dataArray[12] - 48);
    readingHour = ((dataArray[14] - 48) * 10) + (dataArray[15] - 48);
    readingMinute = ((dataArray[17] - 48) * 10) + (dataArray[18] - 48);
    readingSecond = ((dataArray[20] - 48) * 10) + (dataArray[21] - 48);

setTime(readingHour,readingMinute,readingSecond,readingDay,readingMonth,readingYear);

    fileName[0] = dataArray[3];
    fileName[1] = dataArray[4];
    fileName[2] = dataArray[6];
    fileName[3] = dataArray[7];
    fileName[4] = dataArray[14];
    fileName[5] = dataArray[15];
    fileName[6] = dataArray[17];
    fileName[7] = dataArray[18];
    fileName[8] = '.';
    fileName[9] = 't';
    fileName[10] = 'x';
    fileName[11] = 't';
    fileName[12] = '\0';
    dateSet = true;
    memset ( (char*)dataArray, 0, 30 );
    block = false;}
```

Listado 4-3: Acción Sincronización Dispositivo Arduino con Android

Como se ha comentado anteriormente, en función de la trama recibida se ejecuta una u otra acción, en este caso, para la sincronización del dispositivo Arduino con Android ha de recibirse la trama ";TT", más la validación por parte de la UART de que se ha recibido la trama entera (variable booleana *dataReceived*).

Una vez se cumplen las dos acciones, se ejecuta dicha acción. En primer lugar se cambia el valor de la variable booleana "*dataReceived*" para que no se vuelva a ejecutar ninguna acción hasta recibir una trama nueva y el valor de la variable booleana "*block*" para bloquear las interrupciones en la UART.

A continuación se agrega la información recibida en la trama en la variable de tipo char "*reply*", agregando solo los datos necesarios y no la trama entera. Una vez concluido este proceso se envía dicho valor a través de la UART para la confirmación con Android de que los valores recibidos son los correctos.

Para establecer el tiempo en Arduino, se utiliza la función "*setTime(hr,min,sec,day,month,yr)*", para ello es necesario pasarle los datos como se ve en la función hora, minuto, segundo, día, mes y año. Por eso, previamente, se han obtenido dichos datos.

Esta acción, además de la sincronización de Arduino, crea un fichero en la SD para la escritura de los valores del sensor. Para ello utiliza tanto la información del día, mes, hora y minutos del tiempo recibido en Arduino para nombrar ese fichero, creándose un fichero cada vez que se sincronice Arduino con Android.

Para iniciar la lectura del sensor, es obligatorio la sincronización de Arduino, por eso, posteriormente se habilita la lectura del sensor mediante la variable booleana "*dateSet*".

Por último, esta acción limpia el contenido de la variable "*dataArray*" para que no queden residuos en ella y afecte a las futuras tramas recibidas y desbloquea mediante la variable "*block*" las interrupciones hardware en la UART.

Además de las acciones especificadas arriba, en el loop() se comprueba periódicamente el tiempo transcurrido para que en el caso de que cumpla con las especificaciones seleccionadas por parte del usuario, ejecute la lectura del sensor y posteriormente el grabado de dicho valor en la SD, tal como se muestra en el Listado 4-4.

```
currentMillis = millis();
  if ((currentMillis - previousMillis) >= INTERVAL)
  {
    previousMillis = currentMillis;
    minutes++;
  }


  if ((minutes >= readingMinutes) || firstReading)
  {
    minutes = 0;
    if (firstReading) firstReading = false;

    if (dateSet)
    {
      sensorReading = analogRead(A0);
      time_t t = now();

      Number2DecimalAscii((char)day(t), horafecha, 2, 1);
      horafecha[2] = '/';
      Number2DecimalAscii((char)month(t), horafecha, 2, 4);
      horafecha[5] = '/';
      Number2DecimalAscii((unsigned int)year(t), horafecha, 4, 9);
      horafecha[10] = ' ';
      Number2DecimalAscii((char)hour(t), horafecha, 2, 12);
      horafecha[13] = ':';
      Number2DecimalAscii((char)minute(t), horafecha, 2, 15);
      horafecha[16] = ':';
      Number2DecimalAscii((char)second(t), horafecha, 2, 18);
      horafecha[19] = '\0';

      Number2DecimalAscii(sensorReading, sensor, 4, 3);
      sensor[4] = '\0';

      entry = SD.open(fileName, FILE_WRITE);

      if (entry)
      {
        entry.print((char*)horafecha);
        entry.print(";");
        entry.println((char*)sensor);
        entry.close();
      }
    }
  }
```

Listado 4-4: Temporización, Lectura Sensor y Grabado SD

Por último, en el Listado 4-5 se muestra la estructura utilizada para la lectura de datos de la SD.

```
void printDirectory(File dir, int numTabs){
  while(true){
     entry =  dir.openNextFile();
     delay(500);
     if (! entry) {break;}
     if (entry.isDirectory()) {}
     else {if (entry){
         char anterior = 0;
         String linea = "";
         while (entry.available()){
           char actual = entry.read();
           linea += actual;
           if ((actual == 10) && (anterior == 13)){
             procesarlinea(linea);
             linea = "";}
           anterior = actual;}} }
     entry.close();
     delay(500);}}
```

Listado 4-5: Estructura Lectura SD

Esta estructura, recorre línea a línea y fichero a fichero la SD. Una vez se detecta que se ha obtenido una línea (mediante la variable "actual" y "anterior") se procesa dicha línea.

En dicho proceso, que se muestra en el Listado 4-6, se comprueba que la línea esté dentro de los márgenes especificados por el usuario, en el caso correcto, se envía dicha línea a través de la UART.

```
void procesarlinea(String linea2)
{
 daySD = ((linea2.charAt(0) - 48) * 10) + (linea2.charAt(1) - 48);
 monthSD = ((linea2.charAt(3) - 48) * 10) + (linea2.charAt(4) - 48);
 yearSD = ((linea2.charAt(6) - 48) * 1000) + ((linea2.charAt(7) - 48) * 100) +
((linea2.charAt(8) - 48) * 10) + (linea2.charAt(9) - 48);
 hourSD = ((linea2.charAt(11) - 48) * 10) + (linea2.charAt(12) - 48);
 minuteSD = ((linea2.charAt(14) - 48) * 10) + (linea2.charAt(15) - 48);

 if ((yearFirst == yearSecond) && (yearFirst == yearSD))
 {
       …
               if ((minuteFirst <= minuteSD) && (minuteSecond >= minuteSD))
               {
                 Serial.print(linea2);
               }
       …
}
```

Listado 4-6: Comprobación Final Línea

La arquitectura planteada permite al usuario elegir un rango de fechas para la solicitud de información de la SD, siendo necesario y obligatorio la especificación tanto del límite inferior como superior de dicho rango.

4.3.2 Aplicación Android

La aplicación Android consta de 5 vistas, que se detallarán a continuación. En esta aplicación se ha implementado el componente "ViewFlipper" que permite cambiar de vista deslizando el dedo por la pantalla, en el caso de que no se desee usar este componente, en cada vista se integran los controles necesarios tanto para avanzar como retroceder de vista.

Ilustración 4-10: 1ª Vista Aplicación Android

La primera vista, denominada "SÍNCRONIZACIÓN", integra dos de las acciones descritas anteriormente, en concreto, la sincronización de Arduino UNO con Android y la solicitud por parte de Android de la fecha y hora actual de Arduino UNO.

Como se observa en la Ilustración 4-10 tenemos 2 botones denominados "SINCRONIZAR" y "SOLICITAR FECHA ARDUINO" para las acciones comentadas anteriormente, además esta primera vista incluye 2 controles para visualizar texto, tanto para mostrar la información enviada como la recibida.

La adquisición de la fecha del dispositivo Android se hace automáticamente tomando los valores internos de dicho dispositivo.

Esta vista incluye un botón denominado "RESETEAR", cuya función es borrar el contenido de los controles para visualizar texto.

En la Ilustración 4-11 se muestra la segunda vista.

Ilustración 4-11: 2ª Vista Aplicación Android

En la segunda vista, denominada "TEMPORIZACIÓN", se integran las siguientes acciones, establecimiento del tiempo de temporización del sensor y la solicitud por parte de Android del tiempo establecido para la temporización del sensor.

Tal como se observa en la Ilustración 4-11, tenemos 2 botones denominados "TEMPORIZAR" Y "SOLICITAR MINUTOS ARDUINO" para las acciones comentadas anteriormente, además esta primera vista incluye 2 controles para visualizar texto, tanto para mostrar la información enviada como la recibida.

En esta vista se incluye un control para introducir datos, necesario para una de las acciones descritas anteriormente, este control tiene por defecto un filtro que impide

introducir cualquier carácter que no sea numérico. También se especifica que los datos introducidos deben estar en formato de 2 dígitos, desde 01 a 99.

Finalmente, se incluye un botón denominado "RESETEAR", cuya función es idéntica al descrito en la vista anterior.

En la Ilustración 4-12 se muestra la tercera vista.

Ilustración 4-12: 3ª Vista Aplicación Android

La tercera vista, denominada "SOLICITUD DE INFORMACIÓN", integra la acción de petición de información de la SD por parte de Android.

Tal como se comentó anteriormente, esta arquitectura permite al usuario elegir un rango de fechas para solicitar dicha información. Como se observa en la Ilustración 4-12, la vista dispone de 2 controles de texto ("FECHA INICIAL" y "FECHA FINAL"), y estos a su vez de 2 botones ("FECHA" y "HORA") asociados a los controles anteriores, para ingresar tanto la fecha como la hora en la que se desea realizar el escaneo de información en la SD.

Para ingresar la información necesaria, se ha implementado en los botones comentados anteriormente, la funciones "*DatePicker*" y "*TimePicker*", tal como se muestra en la Ilustración 4-13.

Ilustración 4-13: Vista Función *DatePicker*

El botón denominado "SOLICITAR INFORMACIÓN", es el responsable de llevar a cabo la acción comentada anteriormente, y su función estará disponible bajo la condición de que se hayan ingresado los datos correctamente.

Posteriormente, se incluyen 3 controles de texto, para visualizar la cantidad de datos recibidos, indicando cuales han sido recibidos correctamente y cuales erróneamente si surgiese algún problema durante la comunicación.

Finalmente, se incluye un botón denominado "RESETEAR", con la función descrita en las vistas anteriores.

En la Ilustración 4-14 se muestra la cuarta vista.

Ilustración 4-14: 4ª Vista Aplicación Android

La función de la cuarta vista es graficar los datos procesados en Arduino UNO, para ello se procede a la lectura de un fichero creado en la SD de Android donde se hayan guardados los datos recibidos de Arduino UNO.

En la Ilustración 4-15 se muestra un ejemplo de este proceso. La gráfica implementada contiene la particularidad de permitir el desplazamiento vertical y horizontal, dispone de controles de zoom y para hacer más asequible la lectura de los valores representados, al pulsar sobre uno de ellos muestra un "*Toast*" (mensaje emergente) con el valor elegido y la fecha de adquisición de dicho valor.

Ilustración 4-15: Ejemplo Gráfica

En la Ilustración 4-16 se muestra la quinta vista.

Ilustración 4-16: 5ª Vista Aplicación Android

En esta vista se ha implementado la función "*scroll*" para visualizar los datos procesados en Arduino UNO en forma de texto, esta función permite el desplazamiento vertical de los datos guardados en el fichero creado en la SD de Android.

Puesto que los convertidores ADC de Arduino UNO tienen una resolución de 10 bits, los valores analógicos que muestrean tienen una representación digital comprendida entre los valores 0 – 1023, por ello se incluye un botón denominado "Convertir a ºC", cuya función es convertir el valor en bruto del ADC de Arduino (mediante el control para introducir datos) en ºC y se representa mediante un "*Toast*".

4.3.3 Código Fuente Android

En esta sección se van a describir los componentes *software* que se han diseñado para la aplicación Android. En este caso solo ha sido necesario implementar una actividad para las cinco vistas que componen la aplicación.

La descripción de dicha actividad se realizará enumerando todas las funcionalidades de la aplicación Android, y se describirá en detalle una de ellas.

Las funcionalidades de la aplicación Android son las siguientes:

- Sincronización del dispositivo Arduino UNO con Android.
- Solicitud por parte de Android a Arduino UNO sobre la fecha y hora actual de Arduino UNO.
- Establecimiento del tiempo de temporización del sensor.
- Solicitud por parte de Android a Arduino UNO sobre el tiempo de temporización del sensor.
- Petición de información de la SD.
- Visualización gráfica de los datos procesados.
- Visualización a modo de texto de los datos procesados.

A continuación, se describirá la estructura de la aplicación Android, así como algunos de los métodos utilizados en la aplicación. Finalmente como se ha comentado, se describirá una de las funcionalidades en detalle.

En el Listado 4-7 se muestra la estructura de la aplicación Android.

```
public class MainActivity extends IOIOActivity
{
        @Override
        protected void onCreate(Bundle savedInstanceState)
        {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        }

        class Looper extends BaseIOIOLooper
        {

        @Override
        protected void setup() throws ConnectionLostException
        {
        }
        @Override
        public void loop() throws ConnectionLostException
        {
        }
        }

        @Override
        protected IOIOLooper createIOIOLooper()
        {
        return new Looper();
        }
}
```

Listado 4-7: Estructura Aplicación Android

La clase creada por defecto para una aplicación Android extiende de un "*Activity*" (Actividad), en nuestro caso extiende de "*IOIOActivity*" para utilizar el marco de trabajo recomendado por la comunidad IOIO.

A continuación se llama al método "*onCreate*", esto se hace así porque cuando una actividad se crea, se puede querer crear con información que tenía previamente. Esta información se recibe en el objeto "*savedInstanceState*". También se puede observar que se llama al método "*setContentView*" (que solo se puede llamar desde "*onCreate*") que se utiliza para fijar la vista de la actividad, que es inmutable. Se utiliza normalmente para inicializar nuestra GUI (Interfaz Gráfica de Usuario).

En el Listado 4-8 se muestra la clase "*Looper*" de nuestra aplicación.

```
class Looper extends BaseIOIOLooper
{
        private Uart uart;
        private InputStream in;
        private OutputStream out;
…
```

Listado 4-8: Clase *Looper*

La clase "*Looper*" es el hilo en el que se ejecuta toda la actividad IOIO. Una vez establecida la conexión con IOIO se llama al método "*setup()*" y posteriormente a "*loop()*" que se ejecuta periódicamente mientras haya establecido una conexión con IOIO. Incluye atributos privados que permiten gestionar las comunicaciones con IOIO.

En el Listado 4-9 se muestra el método "*setup()*" de nuestra aplicación.

```
…
@Override
protected void setup() throws ConnectionLostException{
        uart = ioio_.openUart(3, 4, 9600, Uart.Parity.NONE, Uart.StopBits.ONE);
        in = uart.getInputStream();
        out = uart.getOutputStream();
}
…
```

Listado 4-9: Método *setup()*

Como se ha comentado anteriormente, este método se llama cuando se haya establecido una conexión con IOIO, por lo general, se utiliza para abrir pines y como en este caso para la configuración de la UART y obtener los *streams* de entrada / salida asociados a la UART.

En el Listado 4-10 se muestra el método "*loop()*" de nuestra aplicación.

```
…
@Override
public void loop() throws ConnectionLostException{
…
```

Listado 4-10: Método *loop()*

Este método es llamado posteriormente al método "*setup()*" y se llama periódicamente mientras haya una conexión establecida con IOIO. Dentro de él se implementará toda la lógica de los controles disponibles en nuestra aplicación.

En el Listado 4-11 se muestra el método para crear nuestro hilo IOIO.

```
…
@Override
protected IOIOLooper createIOIOLooper(){
        return new Looper();
}
…
```

Listado 4-11: Método Hilo IOIO

Como se ha comentado anteriormente, una actividad solo puede contener una vista, el método "*viewFlipper*" permite agregar vistas a la actividad. Para ello se debe modificar el archivo ".xml" (vista de nuestra actividad) agregando el método "*viewFlipper*" y los distintos archivos ".xml" que corresponden a cada vista agregada, tal como se muestra en el Listado 4-12.

```
<ViewFlipper
…
        <include layout = "@layout/pagina_uno"/>
        <include layout = "@layout/pagina_dos"/>
        <include layout = "@layout/pagina_tres"/>
…
</ViewFlipper>
```

Listado 4-12: Método *viewFlipper* en la Vista de la Actividad

En el Listado 4-13 se observa la implementación del método "*viewFlipper*" en nuestra actividad.

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main)

       …

       viewFlipper.setOnTouchListener(new ListenerTouchViewFlipper());
}

private class ListenerTouchViewFlipper implements View.OnTouchListener
{
@Override
public boolean onTouch(View v, MotionEvent event)
{
        switch (event.getAction())
        {
                case MotionEvent.ACTION_DOWN:
                distance_x = event.getX();
                return true;
                case MotionEvent.ACTION_UP:
                float distance = distance_x - event.getX();
                if (distance > 100.0)
                {
                        if (vista_actual == max_vistas)
                        {
                                return false;
                        }
                        else
                        {
                        vista_actual ++;
                        viewFlipper.setInAnimation(inFromRightAnimation());
                        viewFlipper.setOutAnimation(outToLeftAnimation());
                        viewFlipper.showNext();
                        }
                }
                if (distance < -100.0)
                {
                        …
                }
…
        }
        private Animation inFromRightAnimation()
        {
                Animation inFromRight = new TranslateAnimation(
                                Animation.RELATIVE_TO_PARENT, +1.0f,
                                Animation.RELATIVE_TO_PARENT, 0.0f,
                                Animation.RELATIVE_TO_PARENT, 0.0f,
                                Animation.RELATIVE_TO_PARENT, 0.0f);

                inFromRight.setDuration(500);
                inFromRight.setInterpolator(new AccelerateInterpolator());

                return inFromRight;
        }
…
```

Listado 4-13: Método *viewFlipper* en la Actividad

El método calcula la distancia y la dirección recorrida por el puntero a través de la pantalla, en el caso de que se cumpla alguna de las condiciones implementadas ejecuta el desplazamiento de la vista. Cuanto menor sea el número establecido como condición para el desplazamiento de la vista mayor será la sensibilidad de este método.

También se agregan unas animaciones para el desplazamiento de la vista, haciendo este movimiento más suave.

Cabe mencionar, que el método "*viewFlipper*" es cíclico, en este caso se ha querido suprimir esta característica, para ello utilizando unas determinadas variables se localiza la posición actual de la vista, dichas variables se incrementan o decrecen en cada desplazamiento de vista, tal como se muestra en el Listado 4-14. Mediante estas variables y un conjunto de sentencias "*if*" se consigue eliminar dicha características.

```
private void ShowviewNext()
{
        if (vista_actual == max_vistas)
        {
                return;
        }

        else
        {
                vista_actual ++;
                viewFlipper.setInAnimation(inFromRightAnimation());
                viewFlipper.setOutAnimation(outToLeftAnimation());
                viewFlipper.showNext();
        }
}

private void ShowviewPrevious()
{
        if (vista_actual == 0)
        {
                return;
        }

        else
        {
                vista_actual --;
                viewFlipper.setInAnimation(inFromLeftAnimation());
                viewFlipper.setOutAnimation(outToRightAnimation());
                viewFlipper.showPrevious();
        }
}
```

Listado 4-14: Método Control Vista

En el Listado 4-15 se muestra una de las formas de declarar variables y controles para nuestra aplicación Android.

```
public class MainActivity extends IOIOActivity
{
      …
      private EditText editTemporize;
      …
      private TextView warning;
      …
      private Button nextView;
      …
      public int max_vistas = 4;
      …
      byte [] dR = {';', 'D', 'T', '\n'
      …
```

Listado 4-15: Ejemplo de Declaración de Variables y Controles

Una de las acciones llevadas a cabo dentro del "*onCreate*" es enlazar los controles que forman nuestra vista con los controles de nuestra actividad, para poder darle funcionalidad posteriormente a ese control. En el Listado 4-15 se muestra esta acción.

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
	super.onCreate(savedInstanceState);
	setContentView(R.layout.activity_main);

	…

	nextView = (Button) findViewById(R.id.nextView);

	…

	warning = (TextView) findViewById(R.id.warning);

	…
```

Listado 4-16: Asignación de Controles

Otro de los métodos utilizados en la actividad es el "*DatePicker*" y "*Timepicker*", estos widgets permiten seleccionar una fecha y hora respectivamente de una forma cómoda y con una interfaz más familiarizada para el usuario.

En el Listado 4-17 se muestra la parte donde iniciar los métodos comentados anteriormente. Al pulsar uno de los botones configurados para estos métodos se llamará a "*showDialog()*" pasándole un identificador para identificar que "*DatePicker*" o "*TimePicker*" se va a utilizar.

```
…
date.setOnClickListener(new OnClickListener()
{
	@Override
	public void onClick(View v)
	{
		showDialog(DATE_DIALOG_ID);
	}
});
time.setOnClickListener(new OnClickListener()
{
	@Override
	public void onClick(View arg0)
	{
		showDialog(TIME_DIALOG_ID);
	}
});
date2.setOnClickListener(new OnClickListener()
{
	@Override
	public void onClick(View v)
	{
		showDialog(DATE_DIALOG_ID_2);
	}
});
…
```

Listado 4-17: Métodos *DatePicker* y *TimePicker*

En el Listado 4-18 se muestra el método ("*onCreateDialog()*") que se ejecutará posteriormente a "*showDialog()*". En este método, se guardarán los datos de fecha u hora actuales según corresponda y se abrirá un "*DatePickerDialog*" o "*TimePickerDialog*".

```
protected Dialog onCreateDialog(int id)
{
        switch (id)
        {
                case DATE_DIALOG_ID:
                        final Calendar calendar = Calendar.getInstance();

                        firstYear = calendar.get(Calendar.YEAR);
                        firstMonth = calendar.get(Calendar.MONTH);
                        firstDay = calendar.get(Calendar.DAY_OF_MONTH);

        return new DatePickerDialog(this, datePicker, firstYear, firstMonth, firstDay);

                case TIME_DIALOG_ID:
                        final Calendar calendar2 = Calendar.getInstance();

                        firstHour = calendar2.get(Calendar.HOUR_OF_DAY);
                        firstMinute = calendar2.get(Calendar.MINUTE);

        return new TimePickerDialog(this, timePicker, firstHour, firstMinute, true);

        }
                …

        return null;
}
```

Listado 4-18: Métodos *DatePicker* y *TimePicker* (continuación)

El "*DatePickerDialog*" o "*TimePickerDialog*" recibe los parámetros de fecha y hora guardados anteriormente, y en el caso en el que el usuario pulse el botón "*set*" del *widget* "*DatePicker*" o "*TimePicker*" actualiza las variables que contienen la fecha u hora según corresponda y ejecuta una función para actualizar los controles para visualizar texto, tal como se muestra en el Listado 4-19.

```
private DatePickerDialog.OnDateSetListener datePicker = new
DatePickerDialog.OnDateSetListener(){
@Override
public void onDateSet(DatePicker view, int year, int monthOfYear, int dayOfMonth)
{
        cntrl = 0;
        firstYear = year;
        firstMonth = monthOfYear;
        firstDay = dayOfMonth;
        cntrl = 1;
        updateDisplay();}};

private void updateDisplay()
{
        if (cntrl == 1)
        {
                if (cntrl2 == 1)
                {
                        fechaPrimera = "";
                        firstDate2.setText("Fecha Inicial: " + dec(firstDay) + "/" +
dec(firstMonth + 1) + "/" + firstYear + " " + dec(firstHour) + ":" + dec(firstMinute));
                        fechaPrimera = dec(firstDay) + "/" + dec(firstMonth + 1) + "/" +
firstYear + " " + dec(firstHour) + ":" + dec(firstMinute);
                        }
                        …
}
```

Listado 4-19: Métodos *DatePicker* y *TimePicker* (continuación)

A modo de ejemplo se explica en detalle la funcionalidad de sincronización del dispositivo Arduino UNO con Android.

```
synchronize.setOnClickListener(new OnClickListener()
{
        @Override
        public void onClick(View v)
        {
                try
                {
                        synchronize.setEnabled(false);
        SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
                        String date = dateFormat.format(Calendar.getInstance().getTime());

                        for (int i = 0; i < 19; i++)
                        {
                                syn [i+3] = (byte) date.charAt(i);
                        }

                        out.write(syn, 0, 23);
                        out.flush();
                        Thread.sleep(500);
                        textSynchronize.setText("Fecha Enviada: " + date);
                        new synchronize().execute(date);
                }

                catch (IOException e)
                {
                        e.printStackTrace();
                }

                catch (InterruptedException e)
                {
                        e.printStackTrace();
                }
        }
});
```

Listado 4-20: Funcionalidad Sincronización de Arduino UNO con Android

Como se ha comentado anteriormente, una vez establecida una conexión con IOIO, se llamará periódicamente al método "*loop()*", dentro de este método, se encuentra la implementación del control correspondiente a la funcionalidad comentada anteriormente, tal como se muestra en la Ilustración 4-20.

En primer lugar se deshabilita el control para evitar que se pueda pulsar sobre él una segunda vez sin que haya terminado el proceso, mediante la instrucción "*synchronize.setEnabled(false)*".

A continuación se declara una variable de tipo "*SimpleDateFormat*" denominada "*dateFormat*" con el formato en el que queremos obtener la fecha y hora del dispositivo Android, y en la variable de tipo string "*date*" se guarda la fecha y hora del dispositivo Android utilizando el formato comentado anteriormente.

Una vez obtenida dicha información, ésta se va agregando al *array* denominado "*syn*" mediante un bucle "*for*" a partir de la posición tres del *array*, puesto que los tres primeros *byte* del *array* están reservados como cabecera de la trama.

Cuando el *array* contenga toda la información necesaria se mandará a través de la UART mediante el stream de salida denominado "*out*" y se mostrará a través del control para visualizar texto la información enviada.

Si Arduino tarda en responder a una de las peticiones de Android, la aplicación podría bloquearse, la solución a este problema pasa por crear un hilo. De esta forma, nuestra aplicación sigue ejecutándose sin problemas y simultáneamente nuestro hilo se encuentra a la espera de recibir la información correspondiente de Arduino, tal como se muestra en el Listado 4-21.

```
private class synchronize extends AsyncTask <String, Void, String>{
        protected String doInBackground(String... params){
                long start = System.currentTimeMillis();
                String concatenate = "";
                boolean fin = true;
                int control = 0;
                while (fin){
                        long end = System.currentTimeMillis();
                        if ((end - start) > 30000)
                        {
                                fin = false;
                        }
                        else
                        {
                                try
                                {
                                        int availableBytes = in.available();
                                        while ((availableBytes > 0) && (fin))
                                        {
                                                control = 1;
                                                char Rx = (char) in.read();
                                                concatenate += Rx;
                                                availableBytes --;
                                        }
                                        if (control == 1)
                                        {
                                                if (concatenate.equals(params[0]))
                                                {
                                                        fin = false;
                                                        return "OK";
                                                }
                                                else
                                                {
                                                        fin = false;
                                                        return "MAL";
                                                }
                                        }
                                }
                                catch (IOException e) {e.printStackTrace();}
                        }
                }

                return "MAL";
        }
        protected void onPostExecute(String result){
                if (result == "OK"){
                        Toast.makeText(MainActivity.this, "Dato Recibido Correctamente",
Toast.LENGTH_SHORT).show();
                        synchronize.setEnabled(true);
                }
                if (result == "MAL"){
                        Toast.makeText(MainActivity.this, "Error Recepción De Datos",
Toast.LENGTH_SHORT).show();
                        synchronize.setEnabled(true);}
        }
}
```

Listado 4-21: Hilo Funcionalidad Sincronización

En este caso, el hilo se ha creado para comprobar que los datos recibidos en Arduino coinciden con los enviados por parte de Android, por ello al ejecutar el hilo se le pasa como parámetro la variable "*date*" que contiene la información enviada a Arduino.

El hilo no se ejecuta periódicamente, por ello se implementa un bucle "*while*" que permite periódicamente comprobar si ha llegado información a través de la UART. En caso de fallo en la comunicación entre Android y Arduino en el hilo se implementa una lógica basada en tiempo para que transcurridos más de 30 segundos el bucle "*while*" deje de actuar y se finalice la ejecución del hilo.

Los datos recibidos a través de la UART se van concatenando en una variable de tipo *string* denominada "*concatenate*", una vez se han recibido todos los datos, se comparan ambas variables y en función del resultado, el retorno obtiene un valor.

Finalizando el proceso, una vez recibido el valor de retorno, se muestra su resultado a través de un "*Toast*" y por último se habilita el control que permite de nuevo la sincronización de Arduino con Android.

Capítulo 5

Conclusiones y trabajos futuros

5.1 Conclusiones

En este Proyecto Fin de Carrera se ha implementado una arquitectura hardware y software que permite llevar a cabo el diseño de accesorios electrónicos (*Gadgets*) para *Smartphones* de una forma rápida y sencilla. En concreto, dicha arquitectura permite realizar un prototipado rápido y sencillo de accesorios electrónicos para dispositivos móviles inteligentes basados en el sistema operativo Android.

La arquitectura hardware está basada en las placas de desarrollo IOIO y Arduino UNO. Además, se ha diseñado una placa que está conectada a la placa Arduino UNO y permite la conexión de una tarjeta SD, así como de otras de menor tamaño con el correspondiente adaptador.

La arquitectura software está diseñada sobre el conjunto de componentes software, compatibles con el sistema operativo Android, diseñados por la comunidad IOIO. Además, también se ha diseñado en Sketch de la placa Arduino UNO necesario para el correcto funcionamiento de la arquitectura.

Se ha validado la arquitectura hardware y software propuesta mediante la conexión de un sensor de temperatura analógico comercializado por Microchip. Sin embargo, debido a que ambas arquitecturas se han diseñado teniendo en cuenta que resulte sencillo la conexión de otros sensores y/o actuadores, sería sencillo incluir otros elementos sensores y actuadores en la arquitectura.

5.2 Trabajos futuros

Actualmente la arquitectura propuesta en este Proyecto Fin de Carrera sólo es compatible con dispositivos móviles inteligentes que ejecutan el sistema operativo Android. En el futuro sería interesante conseguir que la arquitectura fuese compatible con dispositivos móviles inteligentes basados en otros sistemas operativos.

La arquitectura se ha validado usando un único sensor de temperatura. No obstante, se ha diseñado teniendo en cuenta que resulte sencillo incorporar otros elementos sensores y/o actuadores. Por ello, otro trabajo futuro a realizar sería incorporar otros elementos sensores y/o actuadores, con objeto de demostrar la facilidad de incorporar nuevos elementos.

El sistema de alimentación de la arquitectura está basado en un adaptador a la red de 230 VAC o en conectar la placa de Arduino UNO a un puerto USB. Por este motivo, una tarea pendiente sería el diseño de un sistema de alimentación para la arquitectura. En concreto sería adecuado seleccionar una batería óptima, así como un sistema de carga adecuado. Dicho sistema de carga podría estar basado en el uso de energías renovables. De este modo, los *Gadgets* diseñados con la arquitectura podrían ser utilizados en entornos exteriores.

El diseño hardware del *Gadget* está basado en la conexión de todas las placas de desarrollo seleccionadas, así como la placa de expansión para conectar la SD con Arduino UNO. Como trabajo futuro resultaría también interesante llevar a cabo la miniaturización del *Gadget* mediante el diseño electrónico de todo el conjunto en una sola PCB.

BIBLIOGRAFÍA

[1] Documentación sobre dispositivos móviles inteligentes. Disponible online en: http://histinf.blogs.upv.es/2012/12/03/smartphones/

[2] Placa Desarrollo Electric Sheep . Disponible online en: https://www.sparkfun.com/products/retired/10745

[3] Placa de Desarrollo Microchip. Disponible online en: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en553673

[4] Placa de Desarrollo Seeduino. Disponible online en: http://www.seeedstudio.com/depot/seeeduino-adk-main-board-p-846.html

[5] PhoneDrone Board for Android. Disponible online en: https://store.3drobotics.com/products/phonedrone-board-for-android

[6] Placa de Desarrollo IOIO para Android. Disponible online en: https://github.com/ytai/ioio/wiki

[7] Sistema Operativo Android. Documentación online en: http://developer.android.com/index.html

[8] Sistema Operativo iOS. Documentacion online en: https://developer.apple.com/

[9] Sistema Operativo Windows. Documentación online en: http://developer.windowsphone.com/en-us

[10] Placa de Desarrollo Arduino. Documentación online en: http://www.arduino.cc/