
# PFC Sistemes Encastats:

## Alarma per acceleració implementada sobre LPC1769 i controlada des de dispositius Android.

**TITULACIÓ: Enginyeria informàtica.**

**AUTOR:** Joaquín Ferrando Martínez.
**DIRECTOR:** Sebastià Cortes Herms.

DATA: 06 / 2014.

# 1. Resum.

En aquest document s'explicarà com s'ha realitzat el projecte de final de carrera (PFC) per al 2º cicle d'enginyeria informàtica.

Es tracta d'un PFC orientat a l'àrea dels sistemes encastats. Està format per un sistema d'alarma que és accionada pel moviment. Té capacitat per a poder comunicar-se amb un servidor propi, i aquest, amb dispositius Android, per tal d'informar de l'estat en què es troba.

El projecte està format per tres parts, cada una amb la seva tecnologia pròpia.

- Sistema encastat: FreeRTOS i C. Detecció de moviments i comunicació.
- Servidor propi: Java Estàndard. Comunicació extrem a extrem.
- Dispositiu Android: Java per a Android. Sistema de control de l'alarma.

La finalitat es crear un sistema robust, capaç de recuperar-se de fallades i a la vegada, pensat per dispositius limitats com pot ser un mòbil.

# Índex

# Índex de figures i taules.

# 2. Introducció.

Els sistemes encastats són dispositius pensats per realitzar algunes funcions molt concretes i normalment orientades a la computació en temps real. Funcionen com una sola peça on tots els elements estan units. No tenen una orientació pensada en l'expansió i la possibilitat de solucionar un gran nombre de necessitats, com sí tenen els ordinadors.

Els sistemes encastats estan vivint actualment un moment d'auge en la seva utilització.

Han aparegut molts dispositius denominats "wereables". Petits sistemes encastats, que permeten portar-los a sobre i ens donen informació sobre, per exemple, els nostres hàbits a l'hora de moure'ns al llarg del dia.

Un altre exemple que ha estès molt l'ús de microcontroladors i sistemes encastats, ha sigut el sistema Arduino. Es tracta d'un entorn pensat per a què el seu aprenentatge sigui ràpid i fàcil d'utilitzar, però a la vegada, manté la possibilitat de controlar tots els aspectes del microcontrolador a baix nivell, per a usuaris més avançats.

Els sistemes encastats, permeten realitzar equips de baix cost i consum, per realitzar tasques que no necessitin un gran poder de computació. És per aquest motiu que els trobem en tantes aplicacions.

## 2.1. Justificació.

Aquest projecte neix de la voluntat d'unir diferents tecnologies per tal de crear un sistema complet. Es tracta d'un sistema d'alarma pensat per ser controlat des de dispositius Android, de manera que es dóna una llibertat a l'usuari final, que no tindria amb altres sistemes.

La idea de programar sistemes encastats comença de l'anterior projecte de final de carrera, de la titulació d'ETIS a la URV, on es va realitzar el control d'un motor de CC a través de la implementació d'un PID.

Aquest tipus de sistemes permeten veure d'una manera més directa els efectes que produeix el codi que s'està programant, ja que donen una interacció física de manera més o menys senzilla.

## 2.2. Descripció.

Es tracta d'una alarma per a una porta, que és accionada per moviment. Un acceleròmetre està contínuament llegint dades sobre l'acceleració. Aquestes dades són enviades a un microcontrolador que s'encarrega de detectar canvis bruscos, que podrien significar que s'està intentant forçar lar porta.

La informació que processa el microcontrolador, és enviada a un servidor per tal que aquest pugui comunicar als dispositius destinataris la informació.

Els dispositius destinataris poden ser múltiples, l'únic requisit, és que treballin amb Android.

Els dispositius Android permetran a l'usuari final consultar en qualsevol moment l'estat en que es troba el sistema, així com rebre avisos d'alarma en cas que el microcontrolador deixi de funcionar, o enviï una alerta al detectar moviments bruscos.

## 2.3. Objectius.

Els objectius principals que s'han tingut en compte a l'hora de realitzar aquest projecte han sigut els següents:

- Evitar falses alarmes. El sistema ha de tenir una lògica que permeti evitar que nosaltres mateixos activem l'alarma de manera accidental al obrir o tancar la porta.
- Robustesa contra els talls elèctrics, de comunicació o destrucció del dispositiu. El sistema, de manera global, ha d'intentar solucionar de la millor manera tots els possibles inconvenients que impedeixin el seu funcionament normal.
- Aplicació Android de baix consum. L'aplicació en la part de l'usuari, ha de tindre un consum de dades i de bateria molt contingut, per tal d'evitar problemes.
- Transparència per a l'usuari. El sistema global ha d'intentar ser transparent a l'usuari, de manera que l'arquitectura utilitzada no introdueixi complicacions en el funcionament.

## 2.4. Planificació

Originalment es va realitzar una planificació aproximada per tal de repartir el temps disponible per realitzar el projecte.

Els punts que es presenten al diagrama de Gantt són els següents:

**Servidor Propi**

Programació Servidor: Programació de la lògica del servidor.
Comunicació Servidor-Android.
Comunicació Servidor-LPC1769.

**Android**
UI Android.
Funcionament aplicació.
Comunicació Android-Servidor.
Comunicació Android-LPC1769

**LPC1769**
Comunicació LPC1769-Servidor.
Comunicació LPC1769-Android
Detecció de moviments.

**Tests generals**: Tests de comunicació global i detecció de moviments.

**Documentació**: Memòria del projecte.

En la següent pàgina es mostra la planificació original. Aquesta planificació finalment no es va poder cumplir degut a la càrrega de treball d'una altre assignatura i a la feina del lloc de treball. És per això, que posteriorment es mostra la planificació final del projecte.

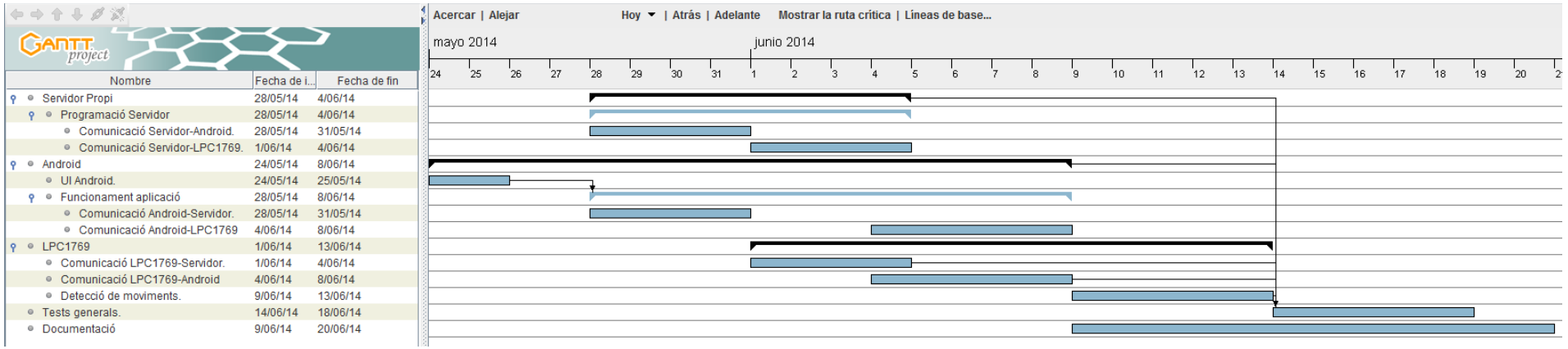


*Figura 1: Planificació original.*

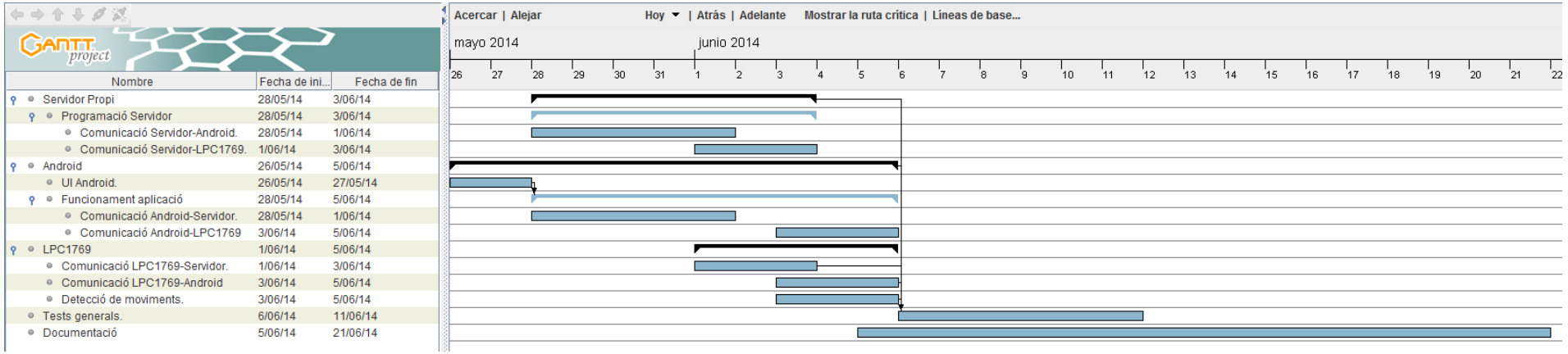


*Figura 2: Planificació final.*

Com es pot observar en les dues figures anteriors, la planificació es va haver de concentrar en la última setmana de Maig i la primera de Juny. Durant aquest període i sobretot la primera setmana de juny, es va dedicar al projecte el màxim d'hores possibles aprofitant una setmana de vacances.

Una vegada passada la primera setmana de Juny, es va dedicar el temps a fer comprovacions generals del comportament global i corregir els error que s'anaven trobant. A la vegada, s'ha anat redactant la documentació.

Si les possibilitats laborals ho haguessin permès, una distribució més equitativa del temps, com per exemple l'original, hagués facilitat la feina i evitat que moltes vegades s'hagués d'anar "contra rellotge" per tal d'acabar cada part del projecte.

# 3. Disseny funcional.

En el sistema que conforma aquest projecte, es troben els següents elements necessaris:

- Element encarregat de detectar moviments: Acceleròmetre.
- Element encarregat de tractar la informació dels moviments: LPC1769.
- Element encarregat de donar connectivitat: WiFly.
- Element encarregat de comunicar els 2 extrems: Servidor intermedi propi.
- Element encarregat de rebre avisos per notificar l'usuari: Android.

En la part referent a l'alarma, trobem els 3 primers elements. L'acceleròmetre, transmetrà la informació al LPC1769. Aquest, decidirà si els moviments que s'estan produint són susceptibles de disparar l'alarma. Per últim, el WiFly proporcionarà connectivitat a internet al LPC1769 per tal que aquest es pugui comunicar amb l'altre extrem.

La comunicació entre els 2 extrems (LPC1769 i Android) no es realitza directament, sinó que serà un servidor propi l'encarregat de controlar el pas de comandes.

Per últim, trobem el dispositiu Android, que serà l'encarregat d'informar a l'usuari de l'estat en què es troba l'alarma.

## 3.1 Funcionament alarma.

### 3.1.1 Lògica de l'alarma.

L'alarma haurà de disposar d'un sistema que permeti activar-la i desactivar-la. Un interruptor.

El funcionament de l'alarma ha de tenir en compte els següent punts:

- Activació i desactivació només quan no ha saltat l'alarma.
- Període de temps entre el moment que s'activa l'alarma, i el moment en que aquesta ja no permet cap moviment sense saltar. D'aquesta manera es podrà activar l'alarma i tancar la porta sense que el moviment de la pròpia porta al tancar-se, faci saltar l'alarma.
- Període de temps entre el moment en que l'alarma detecta moviment i el moment en que l'alarma salta. Aquest període de temps permetrà obrir la porta i desactivar l'alarma abans que salti.

A partir d'aquest punts, es pot veure que entre els diferents estats que tindrà la alarma, n'hi haurà d'haver dos on l'alarma esperarà un temps per passar al següent estat.

El funcionament és similar a les alarmes de les cases, on entre la detecció i l'activació es dóna un període per si es tracta de la pròpia persona de la casa.

Tenint en compte aquests detalls, a continuació es mostraran tots els estats que podrà tenir el sistema:

- **Inici**: L'LPC1769 s'inicia i estableix la connexió WIFI.
- **Sistema Iniciat**: L'LPC1769 ja es troba executant el codi del projecte i amb connexió a internet. Aquest estat és virtual, ja que qualsevol dels dos estats en que es pot trobar l'interruptor faran que es passi a un altre estat.
- **Alarma – Iniciant**: L'LPC1769 detecta que es vol activar la alarma a partir de l'interruptor, i comença un període d'espera de 10 segons abans d'activar-se. Permet que es pugui tancar la porta sense fer saltar la alarma. Durant aquest període, els moviments llegits per l'acceleròmetre són ignorats.
- **Alarma – Activada**: L'LPC1769 ja no ignora els moviments detectats per l'acceleròmetre, i en cas de detectar un moviment brusc, passarà al següent estat.
- **Alarma – Saltant**: L'LPC1769 ha detectat un moviment brusc i donarà un període de 10 segons abans de saltar. Permetrà que es pugui desactivar si la porta s'està obrint pel propietari.
- **Alarma – Saltada**: L'LPC1769 enviarà un missatge al dispositiu Android indicant que s'ha detectat una intrusió.
- **Alarma – Apagada**: L'interruptor de l'alarma es troba desactivat i l'LPC1769 ignorarà l'acceleròmetre i per tant, mai saltarà.

A continuació es mostra un diagrama on es poden veure els diferents estats i quina lògica segueixen:

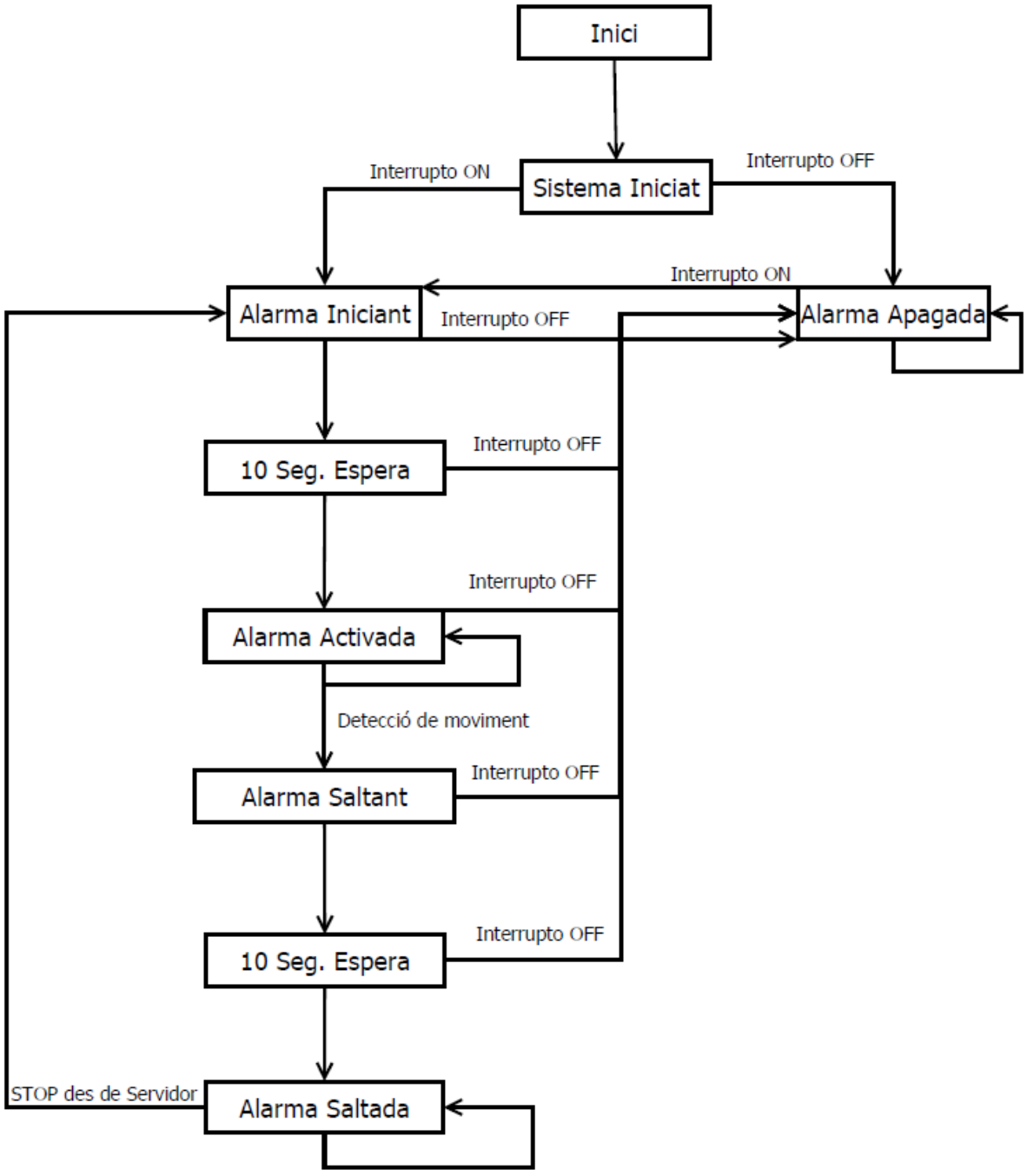


*Figura 3: Diagrama estat alarma*

Per tal de poder identificar de manera visual els estats en que es troba en cada moment l'alarma, es disposa de 3 LEDs; de color blau, vermell i verd respectivament.

En el moment en que el sistema s'està iniciant, els 3 LEDs es troben encesos, a partir d'aquest moment, indiquen els següents estats:

- LED verd: Sempre està parpellejant, indica que l'LPC1769 està executant codi i per tant, està funcionant.
- LED blau: Indica si l'alarma està activada.
- LED vermell: Quan parpelleja, indica que el sistema es troba en un dels dos estats d'espera de 10 segons. Si està encès sense parpellejar, indica que s'alarma ha saltat.

La taula de possibles combinacions dels LEDs és la següent. No es mostra l'estat virtual "Sistema iniciat", ja que aquest no es veu representat pels LEDs.

| | Estats dels LEDS | | |
|---|---|---|---|
| Inici | | | |
| Alarma Iniciant | Parpellejant | | Parpellejant |
| Alarma Activada | Parpellejant | | |
| Alarma Saltant | Parpellejant | | Parpellejant |
| Alarma Saltada | Parpellejant | | |
| Alarma Apagada | Parpellejant | | |

*Taula 1: Estats dels LEDs*

Com es pot veure, els períodes d'espera de 10 segons es representen de la mateixa forma, tant si és per activar, com per fer salta l'alarma.

### 3.1.2 Comunicació alarma.

La alarma ha de poder comunicar-se amb l'usuari del dispositiu Android, per tal de mostrar l'estat en què es troba.

La comunicació es podria realitzar de manera directa, extrem a extrem. Aquesta configuració, però, presenta un problema. En cas que l'alarma sigui destruïda o desconnectada de l'electricitat, l'usuari final no tindria manera de saber-ho.

Una manera de solucionar el problema de detectar si l'alarma segueix funcionant, seria enviar missatges de manera periòdica al dispositiu Android. El que es coneix com a *keepalive.*

Implementar un *Keepalive,* implicaria que el dispositiu Android ha d'estar contínuament a l'espera de rebre missatges de l'alarma. Alhora, ha de comptar el temps que triguen per tal de considerar que l'alarma ha deixat de comunicar-se, si passa un cert temps sense rebre missatges. Aquest comportament podria provocar un consum de bateria i dades excessiu en el dispositiu Android.

Per tal de poder utilitzar la lògica de funcionament comentada, es decideix que la millor opció serà que existeixi un servidor intermedi que s'encarregui de controlar els 2 extrems de la comunicació, i només faci arribar els missatges a les 2 bandes, quan sigui realment necessari.

La utilització del servidor intermedi permetrà rellevar els aspectes de nivell més baix de la comunicació a aquest, de manera que els extrems s'abstrauran a un nivell més alt, aconseguint un menor acoblament.

El sistema de l'alarma, haurà d'enviar els següents tipus de missatges al servidor:

- *Keepalive:* Cada 2 segons, per indicar que segueix funcionant i amb connectivitat.
- Alarma saltada: En cas que salti l'alarma.

Podrà rebre els següents tipus de missatges del servidor:

- Resposta *Keepalive*: El servidor sempre respondrà als *keepalive* per tal d'indicar que s'han rebut correctament.
- Aturar alarma saltada: Haurà de passar de l'estat "Alarma saltada" a "Alarma iniciant"

La identificació dels tipus de missatge és farà a través dels següents ids que apareixen a la taula. L'id serà la dada que s'enviarà per identificar el tipus de missatge que s'està enviant.

| Direcció de la comunicació | ID missatge | Descripció |
|---|---|---|
| LPC -> Servidor | 10 | *Keepalive.* |
| LPC -> Servidor | 20 | Alarma saltada. |
| Servidor -> LPC | OK | *Keepalive rebut.* |
| Servidor -> LPC | STOP | Aturar alarma saltada. |

*Taula 2: Missatges alarma*

En cas que LPC1769 no rebi la resposta "OK" al enviar *Keepalive*, considerarà que el servidor intermedi s'ha desconnectat. En aquest moment, intentarà tornar a establir la comunicació de manera reiterada fins que torni a rebre respostes de "*Keepalive* rebut".

Com es pot comprovar, la importància de controlar en tot moment que la connexió està funcionant correctament és molt gran. En el moment en que el WiFly o el servidor perden la seva connectivitat a internet, el sistema queda desprotegit. Per tant, s'ha d'intentar fer tot el possible per evitar aquests casos.

## 3.2 Funcionament servidor intermedi.

La finalitat d'aquest servidor intermedi serà la de controlar la comunicació extrem a extrem entre l'LPC1769 i el dispositiu Android.

En l'apartat anterior ja s'han explicat els tipus de missatges que poden existir entre aquest servidor i l'LPC1769. La tecnologia sobre la que s'establirà la connexió (TCP, HTTP, UDP, etc..) s'explicarà en el disseny tècnic.

La comunicació entre el servidor i el dispositiu Android, haurà de tenir en compte les limitacions en quan a dades y bateria, que tenen els dispositius mòbils. Per això, en el disseny tècnic es discutiran les diferents alternatives i quina seria la opció més adient.

L'esquema general de la comunicació, es pot veure en la següent imatge.

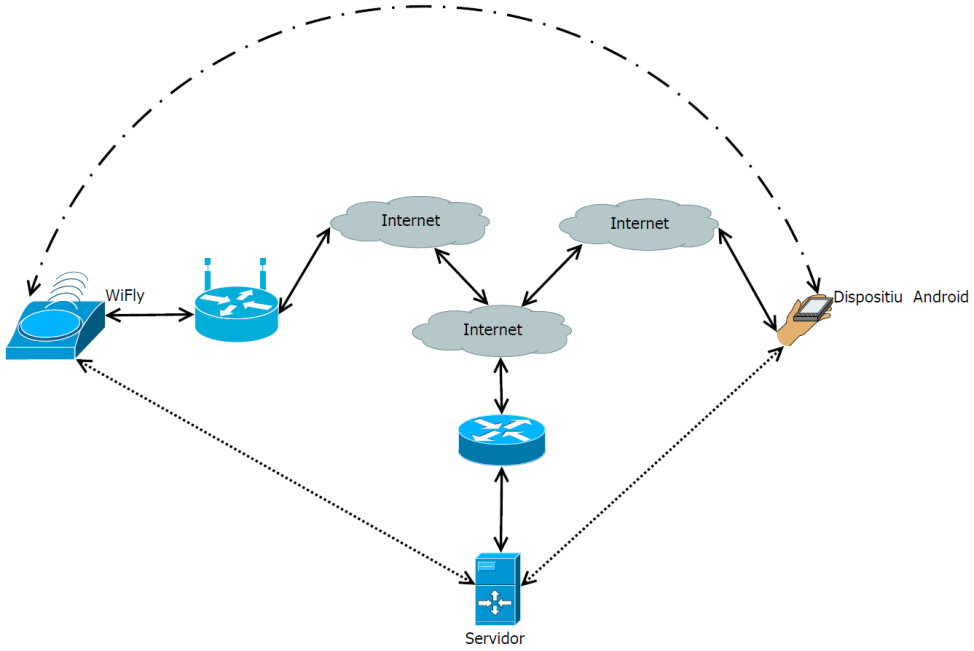


*Figura 4: Diagrama comunicació.*

Les línies contínues representen el camí real que pot seguir la comunicació. Les línies de punts, representen la comunicació des del punt de vista de la programació de la comunicació. Les línies amb ratlles i punts, representen el punt de vista de l'usuari final, on el servidor serà transparent i la comunicació serà extrem a extrem.

Com es pot veure en la imatge, poden arribar a existir tres connexions diferents a internet. El fet que el servidor i WiFly no es trobin a la mateixa xarxa, és degut a que el servidor no ha d'estar al mateix lloc físic on es troba l'alarma; si hi estigués, un tall en el

subministrament elèctric amb la finalitat de inutilitzar l'alarma, també apagaria el servidor. El qual, ja no podria avisar a l'usuari que el WiFly ha deixat de enviar senyals de *Keepalive.*

La configuració d'adreces que es tractarà en el disseny tècnic. S'haurà de tenir en compte que cap dels elements de la comunicació es trobarà a la mateixa xarxa i a més, les adreces IP poden variar, no es podrà fer un tractament estàtic d'aquestes.

L'exemple més clar d'aquesta casuística el trobem en el dispositiu mòbil. El qual, durant un mateix dia, pot passar d'estar connectat mitjançant WIFI, a estar-ho mitjançant internet mòbil. Això provocarà que tingui diferents adreces IP.

El servidor també haurà de ser capaç d'informar més d'un dispositiu Android. Haurà de contenir un llista de dispositius a notificar i en cas que l'alarma enviï una notificació, aquesta ha de ser rebuda per tota la llista de destinataris.

## 3.3 Funcionament dispositiu Android.

El dispositiu Android, serà l'encarregat de fer arribar a l'usuari tota la informació de l'estat del sistema.

Tot i que el servidor serà transparent a l'usuari, l'aplicació haurà de permetre comprovar-ne l'estat. Aquesta funcionalitat, té una finalitat més orientada a la depuració del projecte que no a l'usuari final. Per tant, en una versió final, es podria eliminar.

Les accions que ha d'implementar l'aplicació són:

- Enregistrar-se al servidor: El servidor contindrà una llista de dispositius a notificar en cas que l'alarma hagi saltat. Per tant, s'ha de proporcionar a aquests, un sistema de registrament.
- Conèixer l'estat en què es troba el servidor. Com ja s'ha explicat, aquesta funcionalitat té una finalitat de depuració. L'aplicació ha de tenir un botó que permeti consultar l'estat enviant un missatge al servidor.
- Notificar a l'usuari que l'alarma ha saltat. La notificació d'aquesta informació s'haurà de fer de manera sonora. Per tal que l'usuari se n'assabenti, el dispositiu haurà de sonar en forma d'alarma. Per tant, ha de ser independent de la resta de volums de so que el dispositiu té configurats en aquell moment.
- Conèixer l'estat en que es troba l'alarma. A través del botó de comprovació de la comunicació, el dispositiu mostrarà, a més de la informació del servidor, l'estat en que es troba la comunicació del WiFly i l'estat en què es troba l'alarma.
- Parar l'alarma. El dispositiu ha de ser capaç de parar l'alarma de manera remota quan aquesta hagi saltat. L'alarma s'ha de poder parar en qualsevol moment, no només en el moment que el dispositiu ha rebut la senyal.

Totes aquestes accions s'hauran de realitzar sempre tenint en compte les limitacions del dispositiu. Les decisions de quin mètode seguir per implementar de la manera més adequada cada funcionalitat, s'explicaran en el disseny tècnic.

# 4. Disseny

## 4.1. Disseny hardware.

### 4.1.1. LPC1769

Per realitzar aquest projecte es fa servir la placa LPCXpresso LPC1769 amb FreeRTOS. LPCXpresso és una plataforma de desenvolupament de baix cost i consum. Disposa del seu propi IDE de desenvolupament basat en Eclipse, i la seva programació es realitza sobre llenguatge C.

La programació del microcontrolador es fa a través del port USB del que disposa la placa, amb el qual, també es permet la depuració del codi de manera senzilla.

FreeRTOS, és un sistema operatiu en temps real orientat als sistemes encastats. Es troba disponible en múltiples models i arquitectures. Ofereix la possibilitat d'utilitzar fils d'execució, tasques, mutex, semàfors, cues, timers, etc...


*Figura 5: Placa LPC1769*

Les característiques generals de l'LPC1769 són:

- Microcontrolador: ARM Cortex M3.
- Voltatge d'operació: 5 V USB o 3.3 V per font externa.
- Pins E/S: 58
- Entrades analògiques: 8 amb resolució de 12 bits.
- Corrent DC per pins de E/S: 40 mA
- Memòria Flash: 512 KB
- RAM: 64 KB
- Velocitat de rellotge: 12 MHz
- Datasheet: http://www.nxp.com/documents/user_manual/UM10360.pdf

### 4.1.2. WiFly

WiFly és un adaptador WI-FI que permet la connexió a xarxes sense fil i la interacció amb connexions per internet de manera senzilla, a través de comandes simples al seu port sèrie.

Amb el kit rebut, també s'inclou un adaptador per tal de poder connectar el WiFly a una placa protoboard o qualsevol que utilitzi una separació 0.1" (2.54 mm) entre els pins.

WiFly utilitza un firmware de Roving que es troba en la versió 4.00 en els models sortits de fàbrica. Tot i això, la última versió disponible a la web és la 4.41. El mòdul donava alguns problemes amb la versió 4.00 a l'hora de treballar sobre sockets amb TCP. Per tal de solucionar-ho, es va actualitzar a 4.41 i els problemes van desaparèixer.


*Figura 6: WiFly*

Les característiques generals de WiFly són:

- Basat en XBee.
- Compatible amb el estàndards 802.11 b i g.
- Voltatge d'operació: 3.3 V.
- Baix consum: 38 mA.
- Entre les possibilitats TCP/IP inclou: DHCP, TCP, UDP, DNS, ARP, ICMP, Client HTTP i FTP
- Entrades analògiques: 3.
- Pins E/S: 8
- Datasheet: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Wireless/WiFi/WiFly-RN-XV-DS.pdf

### 4.1.3. MMA7361

Aquest integrat és un acceleròmetre de 3 eixos de baix consum. Permet la detecció de moviment així com la seva direcció, a través de la lectura dels seus ports analògics on es representa l'acceleració en forma de voltatge.

*Figura 7: MMA7361*

Les característiques generals del MMA7361 són:

- 3 eixos de mesurament.
- Baix consum: 400 uA.
- Voltatge d'operació: 2.2 V – 3.6 V (3.3 V recomanat).
- Temps de resposta de 0.5 ms.
- Datasheet: http://www.nxp.com/documents/user_manual/UM10360.pdf

### 4.1.4. Protoboard

Per tal de poder connectar tots els elements hardware d'una manera més neta que utilitzant únicament cables, s'ha fet servir una placa protoboard.

Aquesta placa connecta l'LPC1769 amb el WiFly i l'acceleròmetre. A més d'aquests elements, també s'utilitzen 3 LEDs que serviran per donar informació sobre l'estat de l'alarma, així com un interruptor per tal d'activar-la o desactivar-la.

Per facilitar la programació, s'ha inclòs un polsador que reinicia el WiFly, tot i que aquest pas es podria fer per software amb l'LPC1769.

L'explicació del significat dels LEDs s'ha fet a la part del disseny funcional. En aquest apartat, únicament s'explicarà com s'han connectat tots els elements així com els càlculs realitzats.

#### *4.1.4.1. LPC1769 – WiFly*

A la figura següent es pot veure de manera esquemàtica com s'ha realitzat la connexió.

El WiFly es comunica a través de l'UART 0 de l'LPC1769 i obté l'alimentació d'aquest utilitzant els carrils comuns de la placa protoboard, on s'han instal·lat VCC i GND del LPC1769 per alimentar totes les parts del circuit.

Els canals TX i RX de l'LPC1769 es connecten als canal RX i TX de WiFly respectivament. És a dir, els canals estan invertits. Això és així ja que un canal envia i l'altre rep, per tant, el que envia ha d'anar connectat al que rep i el que rep s'ha de connectar al que envia.

Es pot veure com s'ha instal·lat un polsador al pin de Reset del Wifly, que el connecta a terra en polsar-lo, provocant que es reinicii.

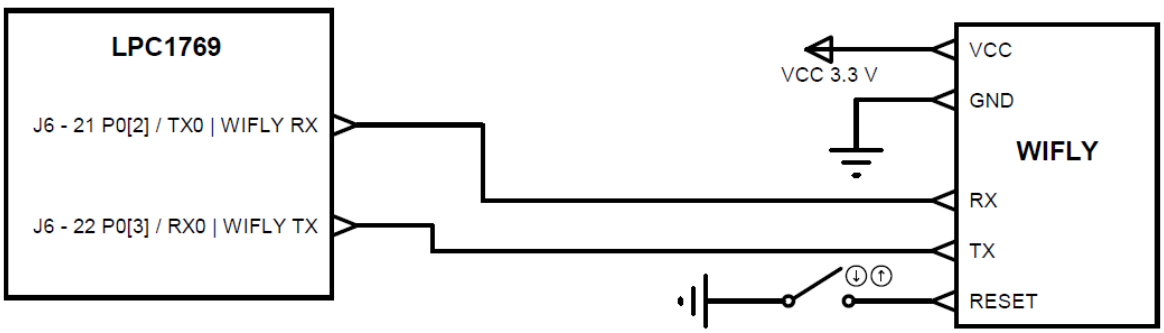


*Figura 8: Connexió LPC1769 - WiFly*

### *4.1.4.2. LPC1769 – MMA7361*

A la figura següent figura es pot veure com s'ha connectat l'acceleròmetre a l'LPC1769.

L'MMA7361 té 3 sortides analògiques (X,Y,Z), que donen un rang entre 0 i 3.3 V. Aquest valor, permet saber l'acceleració de cada eix.

Una vegada el MMA7361 està alimentat, és necessari que el pin SL estigui en estat High. Per fer-ho, s'ha connectat a 3.3V.

En aquest moment, el sensor ja genera dades a través dels voltatges analògics de les 3 sortides de que disposa.

Per llegir valors analògics, el LPC1769 disposa de 8 pins que poden ser configurats com a entrades analògiques. A la següent taula es mostren aquests pins:

| Port analògic | Pin corresponent | Utilitzat per UART | Bits per funcionar com AD |
|---|---|---|---|
| AD0.0 | P0.23 | -- | 01 |
| AD0.1 | P0.24 | -- | 01 |
| AD0.2 | P0.25 | TX UART3 | 01 |
| AD0.3 | P0.26 | RX UART3 | 01 |
| AD0.4 | P1.30 | -- | 11 |
| AD0.5 | P1.31 | -- | 11 |
| AD0.6 | P0.3 | RX UART0 | 10 |
| AD0.7 | P0.2 | TX UART0 | 10 |

A la tercera columna es pot veure que 4 d'aquests 8 pins, són utilitzats per les UART 0 i 3. Per tant, queden descartats ja que es necessiten per poder-se comunicar amb el Wifly i USB_CP2102 quan es necessitin els dos alhora.

Es trien els ports AD0.0, AD0.1 i AD0.4, ja que aquests no interfereixen amb el funcionament de les UART.

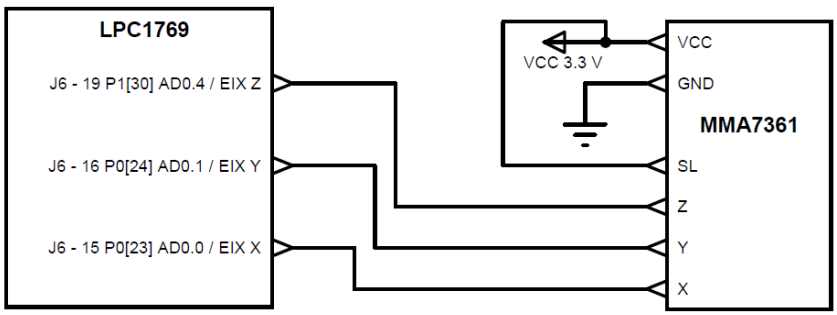


*Figura 9: Connexió LPC1769 – MMA7361*

### 4.1.4.3. LPC1769 – USB_CP2102

Per tal de poder depurar millor el codi i conèixer què està passant en cada moment al microcontrolador, es fa servir la UART3 per tal que enviï al port sèrie del CP2102 l'equivalent als "printf" que faríem servir en una aplicació en C de consola.

A la figura següent es mostra com s'han realitzat les connexions d'aquesta part. Al igual que passava amb el WiFly, la línies dels port sèrie van invertides per tal que es pugui produir la comunicació.

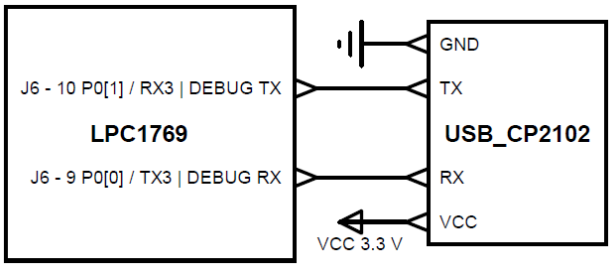


*Figura 10: Connexió LPC1769 – USB_CP2102*

### 4.1.4.4. LPC1769 – LED's

Per indicar l'estat en que es troba l'alarma, s'utilitzen un total de 3 LEDs. Es tracta de LEDs d'alta lluminositat, en color blau, vermell i verd respectivament.

L'alimentació requerida per aquests és la següent:

- Voltatges Funcionament en màxima lluminositat:
  - Blau: 3.2 – 3.4 V.
  - Verd: 3.2 – 3.4 V.
  - Vermell: 2.0 – 2.2V.
- Intensitat funcionament: Tots tres colors, 20 mA.

Com es pot veure, tant el LED blau com el verd podrien anar connectats directament a un GPIO del LPC1769, ja que aquests proporcionen 3.3 V. En canvi, el vermell necessitaria una resistència per evitar que es cremés.

La intensitat consumida està per sota del màxim de 40 mA que poden proporcionar el GPIO. Per tant, pel que fa al corrent, es podrien connectar directament.

La lluminositat d'aquests LED's és tan alta, que arriben a molestar a la vista si s'alimenten amb el màxim de voltatge i corrent que admeten. Per aquesta raó, es va decidir que tots funcionessin amb una resistència, per rebaixar així la seva lluminositat i evitar que molestessin al mirar-los directament.

Els valors de resistència triats no es van escollir fent càlculs, sinó que es van anar provant valors fins a trobar-ne un que reduís prou la lluminositat per tal que no molestés a la vista.

El valor escollit és de 470 Ω. Amb aquest, s'ha mesurat la intensitat amb un multímetre i posteriorment s'han calculat els voltatges utilitzant la llei d'Ohm.

LED Blau: 0.9 mA.

Voltatge consumit per la resistència: $\boldsymbol{Vr = 470 \cdot 0.9 \cdot 10^{-3} = 0.423\ V}$.

Voltatge consumit pel LED: $\boldsymbol{Vl = 3.3 - 0.423 = 2.877\ V}$.

LED Verd: 0.8 mA.

Voltatge consumit per la resistència $\boldsymbol{Vr = 470 \cdot 0.8 \cdot 10^{-3} = 0.376\ V}$.

Voltatge consumit pel LED: $\boldsymbol{Vl = 3.3 - 0.376 = 2.924\ V}$.

LED Vermell: 2.6 mA.

V. consumit per la resistència $\boldsymbol{Vr = 470 \cdot 2.6 \cdot 10^{-3} = 1.222\ V}$.

Voltatge consumit pel LED: $\boldsymbol{Vl = 3.3 - 1.222 = 2.078\ V}$.

Les intensitats han baixat considerablement així com la lluminositat dels LEDs.

Com s'ha explicat, la intensitat consumida pot ser proporcionada pels pins del microcontrolador sense necessitat de cap element extra. En cas que es vulgui connectar un element que consumeixi més de 40 mA, ja serà necessària la utilització de hardware per alimentar aquest element. Com es tracta d'una alarma, seria lògic que una de les sortides fes sonar algun element de potència. Per aquest motiu s'han utilitzat transistors.

Els transistors ens permeten governar altres elements de més potència utilitzant un voltatge i intensitat petits. Alternant entre el seu estat de saturació i tall, es poden fer servir com interruptors.

S'han triat uns transistors de baixa intensitat ja que la intensitat que consumeixen els LEDs es molt baixa després d'haver instal·lat les resistències. D'aquesta manera, si es volgués instal·lar un element de potència, es podria afegir un altre transistor en configuració Darlington o fins i tot un triac i alimentar així un element de potència connectat a la xarxa elèctrica de 230 V.

El transistor triat és de tipus PNP i és el model 2N3904. Es tracta d'un transistor generalista de baixa potència.

Les característiques més rellevants per poder realitzar els càlculs de la resistència que necessita a la base per poder saturar-se són:

- Transistor NPN 2N3904.
  - Voltatge Col·lector – Emissor màxim: 40 V.
  - β mínima @ 1 mA. = 70
  - Intensitat del col·lector màxima: 200 mA.
  - Voltatge Base – Emissor de saturació: 0.85 V

A continuació es calcularà la resistència necessària teòrica. Es faran servir els valors del LED vermell perquè és el que més intensitat consumeix.

$$Ib = \frac{Ic}{\beta} \qquad Ib = \frac{0.0026}{70} = 37.1\ \mu A.$$

A la base del transistor. Per tant, seria necessària una intensitat de $\mathbf{37.1}\ \mu\boldsymbol{A}$, que seran proporcionats per les sortides GPIO del microcontrolador.

$$Rb = \frac{Vin - Vbe}{Ib} \qquad Rb = \frac{3.3 - 0.85}{0.0000371} = 66037\ \Omega$$

La resistència necessària seria d'un 66 KΩ. Amb aquest valor seria suficient per poder fer funcionar els LED's, però si connectem un element que consumeixi més corrent, ja sigui un altre transistor per formar la configuració Darlington o un petit brunzidor, el transistor no entraria en saturació. Per aquest motiu, s'instal·la una residència de 1.5 KΩ que permetrà un rang més ampli de possibilitats.

Amb aquesta configuració, s'ha aconseguit que els 3 GPIO que ara alimenten els LEDs i que com a màxim podien proporcionar 40 mA, puguin proporcionar 200 mA cada un. A més, s'aïlla el pin del microcontrolador de l'element que es col connectar. D'aquesta manera, si es produeix una pujada de intensitat per un curtcircuit, es cremarà el transistor, però al pin del microcontrolador no li passarà res.

Per tal de connectar els transistors, s'han utilitzat 3 GPIO configurats com a sortida.

A continuació es mostra amb quins pins s'ha fet la connexió així com el muntatge dels transistors i resistències.

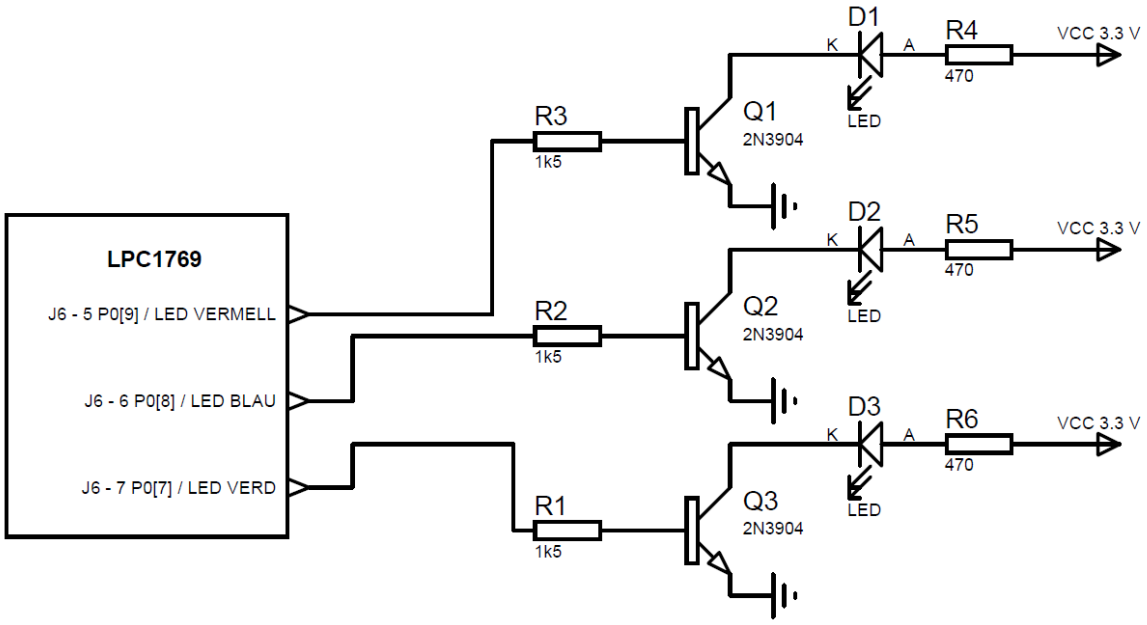


*Figura 11: Connexió LPC1769 – LED's*

### *4.1.4.5. LPC1769 – Interruptor Alarma.*

Per poder activar i desactivar l'alarma, es farà servir un interruptor de 2 posicions, que unit a un GPIO configurat com entrada, permetrà llegir l'estat en què es troba.

Els interruptors presenten problemes de rebots quan es canvia el seu estat. Aquests rebots, són produïts pel propi mecanisme intern de l'interruptor i provoquen un tren de polsos abans que s'estabilitzi el voltatge.

Per tal d'evitar lectures errònies de l'interruptor en tancar-lo o obrir-lo, s'ha instal·lat un condensador de 0.1 $\mu F$ en paral·lel, que absorbirà el tren de polsos evitant que aquest arribi al microcontrolador.

L'interruptor connecta el GPIO al terra del circuit, però quan es deixa l'interruptor obert, el pin del microcontrolador quedaria "a l'aire" provocant lectures errònies generades per els voltatges induïts de les múltiples ones de radio que ens rodegen.

Per evitar aquest efecte, s'ha instal·lat una resistència de Pull-Up, que connecta el pin al carril de 3.3 V quan l'interruptor queda obert. Així s'aconsegueix que en cap moment el pin quedi connectat a l'aire.

A continuació es mostra l'esquema de connexió de l'interruptor.

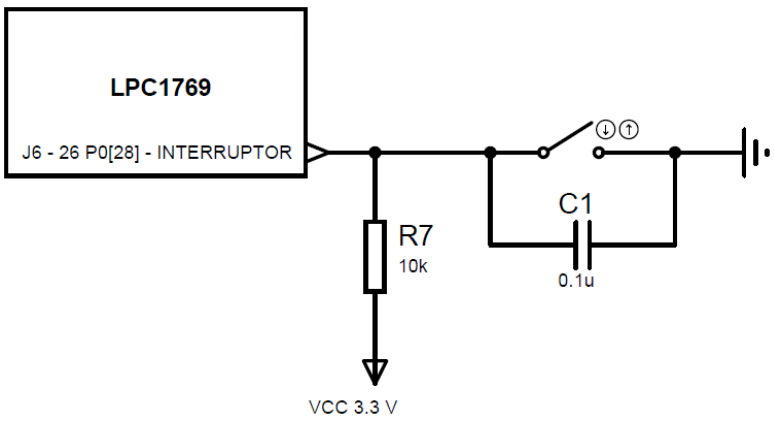


*Figura 12: Connexió LPC1769 – Interruptor Alarma*

### 4.1.4.6. Esquema global

Una vegada s'han explicat tots els elements que formen el circuit per separat, a continuació es mostra l'esquema global resultant.

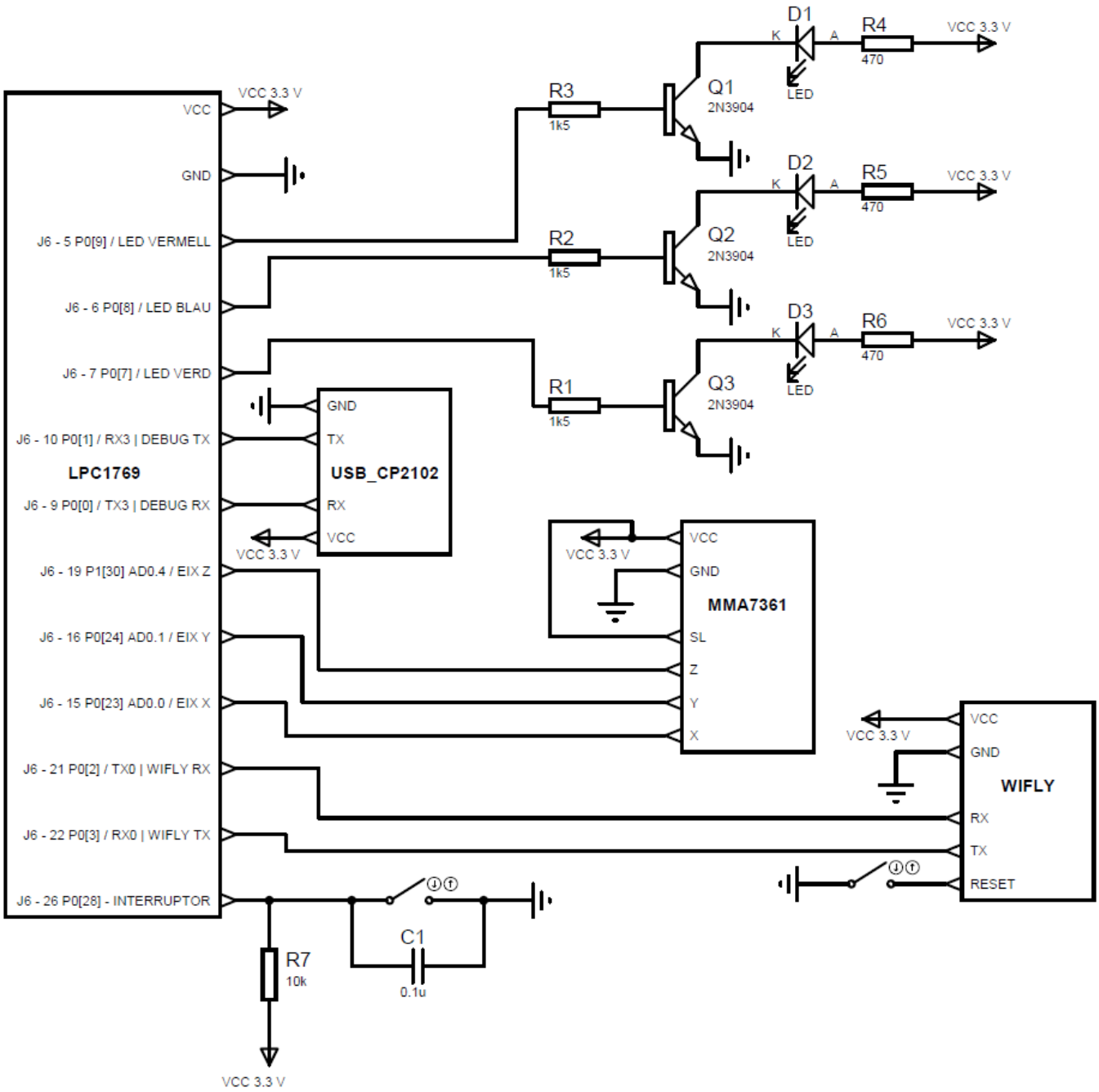


*Figura 13: Esquema global del circuit*

### *4.1.4.7. Construcció de l'esquema sobre la placa protoboard*

Per últim, es mostra una imatge on es pot veure com ha quedat tot muntat a la placa.

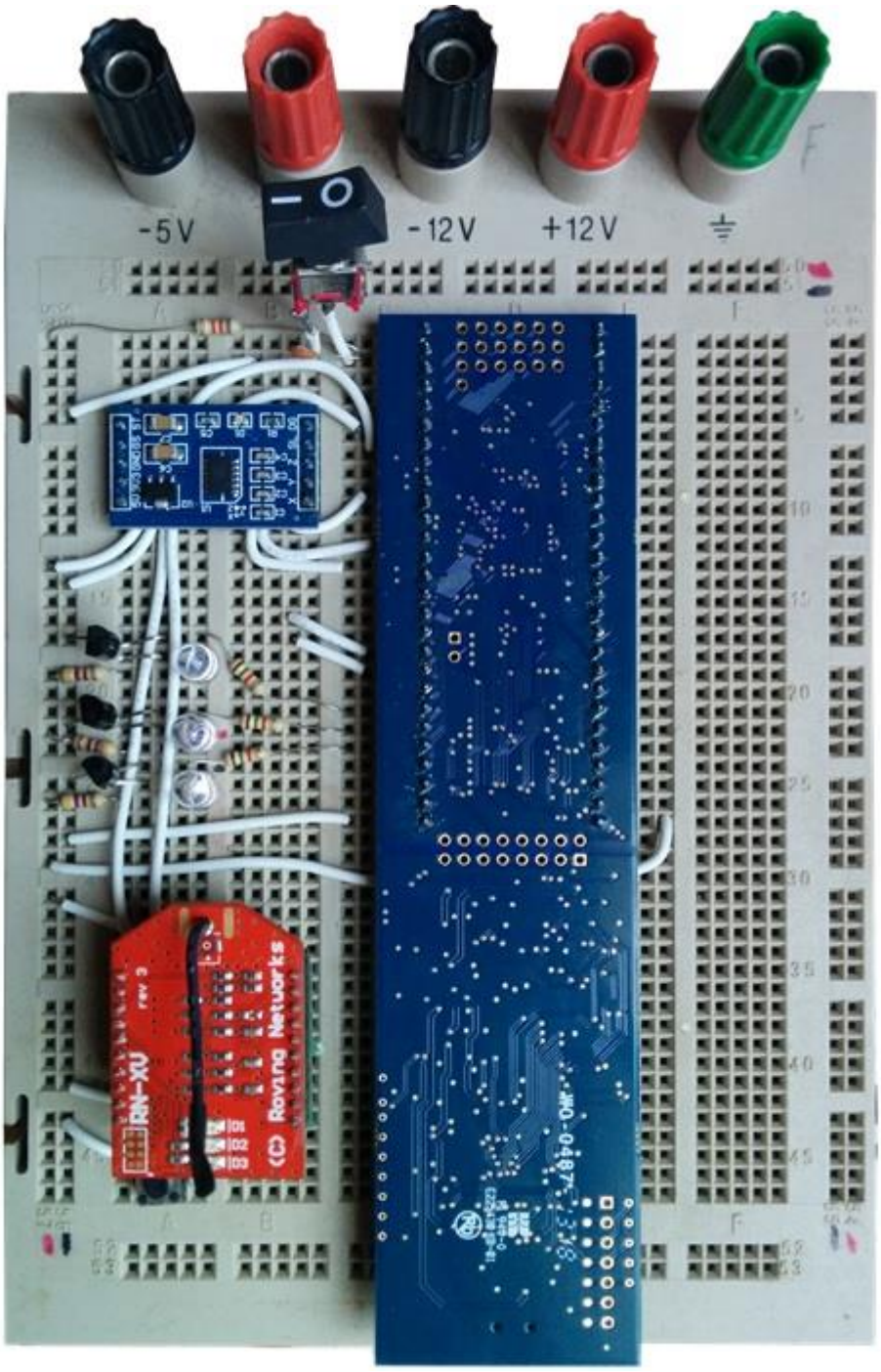


*Figura 14: Circuit muntat sobre protoboard*

## 4.2. Disseny tècnic.

En aquest apartat s'explicarà a un nivell més baix, com s'ha realitzat la implementació de cada una de les parts que conformen el sistema de la alarma.

### 4.2.1. LPC1769.

La placa LPC1769 proporcionada per fer el projecte funciona sobre FreeRTOS. Aquest petit kernel, permet la utilització de tècniques de programació en paral·lel sense tenir que controlar canvis de context, quantums, etc...

Les connexions hardware que s'han realitzat entre la LPC1769, els LED, l'acceleròmetre i WiFly ja s'han explicat en l'apartat anterior. En aquest apartat, es detallarà com s'ha programat la placa per tal de poder utilitzar aquest hardware extern.

#### *4.2.1.1 LPC1769 – Lectura i conversió a digital de valors analògics.*

Per poder llegir els voltatges generats per l'acceleròmetre, s'utilitzaran els ports ADC que proporciona la placa.

Els ports ADC permeten transformar un voltatge en una representació numèrica d'aquest. En el cas del LPC1769, treballa a 3.3V i els ports son de 12 bits.

Amb 12 bits podem representar 4096 valors diferents. Per tant, s'obtindrà un valor entre 0 i 4095 que correspondrà al rang de voltatge 0V – 3.3V.

LPC1769 disposa de 8 pins que poden ser configurats per treballar com a ADC. A la següent taula es mostren aquests pins:

| Port analògic | Pin corresponent | Utilitzat per UART | Bits per funcionar com ADC |
|---|---|---|---|
| AD0.0 | P0.23 | -- | 01 |
| AD0.1 | P0.24 | -- | 01 |
| AD0.2 | P0.25 | TX UART3 | 01 |
| AD0.3 | P0.26 | RX UART3 | 01 |
| AD0.4 | P1.30 | -- | 11 |
| AD0.5 | P1.31 | -- | 11 |
| AD0.6 | P0.3 | RX UART0 | 10 |
| AD0.7 | P0.2 | TX UART0 | 10 |

**Taula 3: Ports ADC LPC1769**

Com es pot veure, els pins poden tenir diferents funcions segons com estan configurats. Aquesta configuració, es realitza al registre PINSEL. En aquest cas, es trien els ports AD0.0, AD0.1 i AD0.4 ja que aquests no interfereixen amb el funcionament de les UART.

Per tal de configurar els port, el valor que s'ha assignat a la taula correspon al PINSEL1 per els ports AD0.0 i AD0.1. Per a AD0.4, el registre que cal configurar és PINSEL3.

Aquesta informació es troba a la següent imatge extreta del *datasheet* del LPC1769.

S'han ressaltat en vermell els ports utilitzats.

| PINSEL1 | Pin name | Function when 00 | Function when 01 | Function when 10 | Function when 11 | Reset value |
|---|---|---|---|---|---|---|
| 1:0 | P0.16 | GPIO Port 0.16 | RXD1 | SSEL0 | SSEL | 00 |
| 3:2 | P0.17 | GPIO Port 0.17 | CTS1 | MISO0 | MISO | 00 |
| 5:4 | P0.18 | GPIO Port 0.18 | DCD1 | MOSI0 | MOSI | 00 |
| 7:6 | P0.19[1] | GPIO Port 0.19 | DSR1 | Reserved | SDA1 | 00 |
| 9:8 | P0.20[1] | GPIO Port 0.20 | DTR1 | Reserved | SCL1 | 00 |
| 11:10 | P0.21[1] | GPIO Port 0.21 | RI1 | Reserved | RD1 | 00 |
| 13:12 | P0.22 | GPIO Port 0.22 | RTS1 | Reserved | TD1 | 00 |
| 15:14 | P0.23[1] | GPIO Port 0.23 | AD0.0 | I2SRX_CLK | CAP3.0 | 00 |
| 17:16 | P0.24[1] | GPIO Port 0.24 | AD0.1 | I2SRX_WS | CAP3.1 | 00 |
| 19:18 | P0.25 | GPIO Port 0.25 | AD0.2 | I2SRX_SDA | TXD3 | 00 |

| PINSEL3 | Pin name | Function when 00 | Function when 01 | Function when 10 | Function when 11 | Reset value |
|---|---|---|---|---|---|---|
| 11:10 | P1.21[1] | GPIO Port 1.21 | $\overline{\text{MCABORT}}$ | PWM1.3 | SSEL0 | 00 |
| 13:12 | P1.22 | GPIO Port 1.22 | MCOB0 | USB_PWRD | MAT1.0 | 00 |
| 15:14 | P1.23 | GPIO Port 1.23 | MCI1 | PWM1.4 | MISO0 | 00 |
| 17:16 | P1.24 | GPIO Port 1.24 | MCI2 | PWM1.5 | MOSI0 | 00 |
| 19:18 | P1.25 | GPIO Port 1.25 | MCOA1 | Reserved | MAT1.1 | 00 |
| 21:20 | P1.26 | GPIO Port 1.26 | MCOB1 | PWM1.6 | CAP0.0 | 00 |
| 23:22 | P1.27[1] | GPIO Port 1.27 | CLKOUT | $\overline{\text{USB\_OVRCR}}$ | CAP0.1 | 00 |
| 25:24 | P1.28 | GPIO Port 1.28 | MCOA2 | PCAP1.0 | MAT0.0 | 00 |
| 27:26 | P1.29 | GPIO Port 1.29 | MCOB2 | PCAP1.1 | MAT0.1 | 00 |
| 29:28 | P1.30 | GPIO Port 1.30 | Reserved | $V_{BUS}$ | AD0.4 | 00 |

**Taula 4: Registre PINSEL 1 i 3**

Per tal que el microcontrolador activi les entrades ADC, a més del registre PINSEL, també cal configurar-ne d'altres.

El registre PCONP és l'encarregat d'activar i desactivar funcions. Per tant, caldrà activar el convertidor analògic/digital indicant-ho en aquest registre.

El *datasheet* indica que, per activar les funcions que controla PCONP, cal posar a 1 el bit que li correspon. Segons la taula que es mostra a continuació, caldrà posar a 1 el bit 12 del registre per activar l'ADC.

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 0 | - | Reserved. | NA |
| 1 | PCTIM0 | Timer/Counter 0 power/clock control bit. | 1 |
| 2 | PCTIM1 | Timer/Counter 1 power/clock control bit. | 1 |
| 3 | PCUART0 | UART0 power/clock control bit. | 1 |
| 4 | PCUART1 | UART1 power/clock control bit. | 1 |
| 5 | - | Reserved. | NA |
| 6 | PCPWM1 | PWM1 power/clock control bit. | 1 |
| 7 | PCI2C0 | The $I^2C0$ interface power/clock control bit. | 1 |
| 8 | PCSPI | The SPI interface power/clock control bit. | 1 |
| 9 | PCRTC | The RTC power/clock control bit. | 1 |
| 10 | PCSSP1 | The SSP 1 interface power/clock control bit. | 1 |
| 11 | - | Reserved. | NA |
| 12 | PCADC | A/D converter (ADC) power/clock control bit. | 0 |

Taula 5: Registre PCONP

Cal indicar al microcontrolador la font del rellotge que utilitzarà l'ADC per funcionar. Com que no es necessita una gran velocitat, es tria la divisió / 8. Es posa el bit 25 a 1 i el bit 24 a 1.

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 1:0 | PCLK_WDT | Peripheral clock selection for WDT. | 00 |
| 3:2 | PCLK_TIMER0 | Peripheral clock selection for TIMER0. | 00 |
| 5:4 | PCLK_TIMER1 | Peripheral clock selection for TIMER1. | 00 |
| 7:6 | PCLK_UART0 | Peripheral clock selection for UART0. | 00 |
| 9:8 | PCLK_UART1 | Peripheral clock selection for UART1. | 00 |
| 11:10 | - | Reserved. | NA |
| 13:12 | PCLK_PWM1 | Peripheral clock selection for PWM1. | 00 |
| 15:14 | PCLK_I2C0 | Peripheral clock selection for I2C0. | 00 |
| 17:16 | PCLK_SPI | Peripheral clock selection for SPI. | 00 |
| 19:18 | - | Reserved. | NA |
| 21:20 | PCLK_SSP1 | Peripheral clock selection for SSP1. | 00 |
| 23:22 | PCLK_DAC | Peripheral clock selection for DAC. | 00 |
| 25:24 | PCLK_ADC | Peripheral clock selection for ADC. | 00 |

| PCLKSEL0 and PCLKSEL1 individual peripheral's clock select options | Function | Reset value |
|---|---|---|
| 00 | PCLK_peripheral = CCLK/4 | 00 |
| 01 | PCLK_peripheral = CCLK | |
| 10 | PCLK_peripheral = CCLK/2 | |
| 11 | PCLK_peripheral = CCLK/8, except for CAN1, CAN2, and CAN filtering when "11" selects = CCLK/6. | |

Taula 6: Registre PCLKSEL1

Per acabar amb la configuració de l'ADC, cal configurar el registre ADCR. En aquest podem seleccionar el divisor aplicat al senyal de rellotge que arriba al perifèric. És un número de 8 bits que serà el divisor aplicat. Al valor assignat se li suma 1 per evitar la divisió entre 0. Se li assigna el valor 1.

El bit BURST (16) controla el procés de conversió A/D. Per poder treballar amb més d'un pin, com serà el nostre cas, cal que sigui configurat amb un 1.

Per tal d'activar la conversió A/D cal posar un 1 al bit 21.

Cal indicar quins dels 8 pins del ADC es volen llegir. Això es fa amb els bits 0 a 7 d'aquest registre. Com que s'estaran llegint AD0.0, AD0.1 i AD0.4 caldrà a posar els bits 0, 1 i 4 a 1.

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 7:0 | SEL | | Selects which of the AD0.7:0 pins is (are) to be sampled and converted. For AD0, bit 0 selects Pin AD0.0, and bit 7 selects pin AD0.7. In software-controlled mode, only one of these bits should be 1. In hardware scan mode, any value containing 1 to 8 ones is allowed. All zeroes is equivalent to 0x01. | 0x01 |
| 15:8 | CLKDIV | | The APB clock (PCLK_ADC0) is divided by (this value plus one) to produce the clock for the A/D converter, which should be less than or equal to 13 MHz. Typically, software should program the smallest value in this field that yields a clock of 13 MHz or slightly less, but in certain cases (such as a high-impedance analog source) a slower clock may be desirable. | 0 |
| 16 | BURST | 1 | The AD converter does repeated conversions at up to 200 kHz, scanning (if necessary) through the pins selected by bits set to ones in the SEL field. The first conversion after the start corresponds to the least-significant 1 in the SEL field, then higher numbered 1-bits (pins) if applicable. Repeated conversions can be terminated by clearing this bit, but the conversion that's in progress when this bit is cleared will be completed.<br>**Remark:** START bits must be 000 when BURST = 1 or conversions will not start. If BURST is set to 1, the ADGINTEN bit in the AD0INTEN register (Table 534) must be set to 0. | 0 |
| | | 0 | Conversions are software controlled and require 65 clocks. | |
| 20:17 | - | | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 21 | PDN | 1 | The A/D converter is operational. | 0 |
| | | 0 | The A/D converter is in power-down mode. | |

Taula 7: Registre ADCR

Una vegada està l'ADC configurat, ja es podran llegir els valors que generarà. Per tal de fer-ho cal accedir al registre ADDR.

El registre ADDR guarda els valors de les conversions, a més d'indicar quan s'han acabat. El bit 31 està a 1 quan ha acabat la conversió. El resultat es guarda en els 12 bits que van del 4 al 15 d'aquest registre.

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 3:0 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 15:4 | RESULT | When DONE is 1, this field contains a binary fraction representing the voltage on the AD0[n] pin, as it falls within the range of $V_{REFP}$ to $V_{REFN}$. Zero in the field indicates that the voltage on the input pin was less than, equal to, or close to that on $V_{REFN}$, while 0xFFF indicates that the voltage on the input was close to, equal to, or greater than that on $V_{REFP}$. | NA |
| 29:16 | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |
| 30 | OVERRUN | This bit is 1 in burst mode if the results of one or more conversions was (were) lost and overwritten before the conversion that produced the result in the RESULT bits.This bit is cleared by reading this register. | |
| 31 | DONE | This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read. | NA |

**Taula 8: Registre ADDR**

### 4.2.1.2 LPC1769 – Configuració de sortides digitals.

L'LPC1769 controlarà 3 LEDs. Per tant, s'hauran de configurar 3 pins com a sortides digitals.

Amb el registre PINSEL configurarem els pins per a què treballin com a GPIO. El PINSEL corresponent als pins dels LEDs P0_07, P0_08 i P0_09 és el PINSEL 0. Segons la següent taula, caldrà configurar els bits de cada pin a 00.

| PINSEL0 | Pin name | Function when 00 | Function when 01 | Function when 10 | Function when 11 | Reset value |
|---|---|---|---|---|---|---|
| 1:0 | P0.0 | GPIO Port 0.0 | RD1 | TXD3 | SDA1 | 00 |
| 3:2 | P0.1 | GPIO Port 0.1 | TD1 | RXD3 | SCL1 | 00 |
| 5:4 | P0.2 | GPIO Port 0.2 | TXD0 | AD0.7 | Reserved | 00 |
| 7:6 | P0.3 | GPIO Port 0.3 | RXD0 | AD0.6 | Reserved | 00 |
| 9:8 | P0.4[1] | GPIO Port 0.4 | I2SRX_CLK | RD2 | CAP2.0 | 00 |
| 11:10 | P0.5[1] | GPIO Port 0.5 | I2SRX_WS | TD2 | CAP2.1 | 00 |
| 13:12 | P0.6 | GPIO Port 0.6 | I2SRX_SDA | SSEL1 | MAT2.0 | 00 |
| 15:14 | P0.7 | GPIO Port 0.7 | I2STX_CLK | SCK1 | MAT2.1 | 00 |
| 17:16 | P0.8 | GPIO Port 0.8 | I2STX_WS | MISO1 | MAT2.2 | 00 |
| 19:18 | P0.9 | GPIO Port 0.9 | I2STX_SDA | MOSI1 | MAT2.3 | 00 |

Taula 9: Registre PINSEL 0

En el registre PCONP caldrà activar els GPIO. Aquest registre ja s'ha vist anteriorment quan s'han configurat l'ADC. En aquest cas, és el bit 15 el que controla l'activació dels GPIO. S'haurà de posar a 1.

| | | | |
|---|---|---|---|
| 15 | PCGPIO | Power/clock control bit for IOCON, GPIO, and GPIO interrupts. | 1 |

Taula 10: Registre PCONP bit 15

Una vegada s'han activat els GPIO, cal indicar la direcció en que treballaran. Aquesta pot ser entrada o sortida. En el nostre cas, ha de ser sortida. El registre que controla la direcció és FIODIR. Serà necessari posar a 1 els bits 7, 8 i 9 d'aquest registre, que són els corresponents als pins dels LEDs.

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | FIO0DIR<br>FIO1DIR<br>FIO2DIR<br>FIO3DIR<br>FIO4DIR | | Fast GPIO Direction PORTx control bits. Bit 0 in FIOxDIR controls pin Px.0, bit 31 in FIOxDIR controls pin Px.31. | 0x0 |
| | | 0 | Controlled pin is input. | |
| | | 1 | Controlled pin is output. | |

Taula 11: Registre FIODIR

Per tal de canviar d'estat el pin s'haurà d'utilitzar el registre FIOSET per posar l'estat HIGH i FIOCLR per posar l'estat LOW. En els 2 casos el canvi es realitza posant a 1 el bit corresponent al pin. En aquest cas, seran els bits 7, 8 i 9.

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | FIO0SET<br>FIO1SET<br>FIO2SET<br>FIO3SET<br>FIO4SET | | Fast GPIO output value Set bits. Bit 0 in FIOxSET controls pin Px.0, bit 31 in FIOxSET controls pin Px.31. | 0x0 |
| | | 0 | Controlled pin output is unchanged. | |
| | | 1 | Controlled pin output is set to HIGH. | |
| **Bit** | **Symbol** | **Value** | **Description** | **Reset value** |
| 31:0 | FIO0CLR<br>FIO1CLR<br>FIO2CLR<br>FIO3CLR<br>FIO4CLR | | Fast GPIO output value Clear bits. Bit 0 in FIOxCLR controls pin Px.0, bit 31 controls pin Px.31. | 0x0 |
| | | 0 | Controlled pin output is unchanged. | |
| | | 1 | Controlled pin output is set to LOW. | |

Taula 12: Registres FIOSET i FIOCLR

#### *4.2.1.3 LPC1769 – Configuració entrades digitals.*

Per poder llegir l'interruptor, farà falta configurar un dels GPIO com a entrada. La configuració del pin com a GPIO serà igual que en el cas de les sortides, l'única diferència, és que ara és el pin P0_28, que correspon als bit 24 i 25 del PINSEL1. S'haurà de posar a 00 que és la configuració GPIO.

Per tal de configurar el GPIO com a entrada, no caldrà modificar el registre FIODIR, ja que aquest per defecte, configura els pins com a entrades.

Una vegada configurat el pin, ja es podrà llegir. Per fer-ho s'haurà de consultar el registre FIOPIN en el bit corresponent al pin. En aquest cas, serà el bit 28.

| Bit | Symbol | Value | Description | Reset value |
|---|---|---|---|---|
| 31:0 | FIO0VAL<br>FIO1VAL<br>FIO2VAL<br>FIO3VAL<br>FIO4VAL | | Fast GPIO output value bits. Bit 0 corresponds to pin Px.0, bit 31 corresponds to pin Px.31. Only bits also set to 0 in the FIOxMASK register are affected by a write or show the pin's actual logic state. | 0x0 |
| | | 0 | Reading a 0 indicates that the port pin's current state is LOW. Writing a 0 sets the output register value to LOW. | |
| | | 1 | Reading a 1 indicates that the port pin's current state is HIGH. Writing a 1 sets the output register value to HIGH. | |

**Taula 13: Registres FIOPIN**

#### *4.2.1.4 LPC1769 – UARTS i control WiFly.*

En aquest projecte es faran servir 2 UART del LPC1769. Un d'ells serà per controlar el WiFly, i l'altre només tindrà propòsit de depuració.

L'UART permet transmetre i rebre dades en comunicacions en sèrie. S'encarrega de transmetre seqüències de bits que representen dades. En la comunicació entre dos UART. La emissora transforma les dades i les envia en forma de bits. La receptora, s'encarrega de reconstruir les dades originals a partir dels bits rebuts per tal de passar-les al sistema i que puguin ser utilitzades.

La velocitat de transmissió utilitzada serà de 9600 bauds, ja que és la que té configurada WiFly per defecte. La configuració utilitzada serà 8N1.

La connexió física entre els UART s'explica en el disseny hardware.

No s'entrarà en detall de la configuració de registres per a utilitzar els UART. Aquesta configuració ja venia donada en les llibreries. Per tant, només en comentaran les funcions més utilitzades.

- UART_Init. Inicialitza la UART
  - Configura el PINSEL i registres del UART a 8N1. Estableix el *baudrate* i activa les interrupcions.
- UART_Close. Tanca la UART
  - Desactiva les interrupcions i reconfigura el PINSEL.
- UART_Write. Escriu un caràcter a l'UART.
  - Modificar el registre THR amb el caràcter a enviar.
- UART_Send. Escriu un una cadena de caràcters per l'UART.
  - Utilitza UART_Write per enviar cada un dels caràcters.
- UART_Flush. Neteja el buffer de lectura de l'UART.
  - Escriu 0s a la direcció de memòria del buffer.
- UART_Read. Llegeix els caràcters que ha rebut l'UART.
  - Desactiva les interrupcions i llegeix la zona de memòria del buffer per recuperar els caràcters que s'han rebut.

#### *4.2.1.5 LPC1769 – Codi Alarma i FreeRTOS.*

El kernel que executa l'LPC1769, ens facilita la tasca de programar el sistema de manera que realitzi diferents accions de manera concurrent. En el desenvolupament d'aquest projecte s'han utilitzat bàsicament tasques i cues.

Una tasca en FreeRTOS és un fragment de codi que s'executa de manera indefinida amb una prioritat determinada. El control de les diferents tasques i la decisió de quina s'ha d'executar en cada moment, és portat a terme per un planificador.

La naturalesa concurrent de les tasques, provoca que es necessiti un sistema de pas de variables segur entre aquestes. Si dos tasques treballen amb els mateixos valors, s'ha d'evitar que aquests es modifiquin quan s'estan llegint. FreeRTOS proporciona les cues per poder manegar aquestes situacions.

Les cues ens permeten el pas de variables entre tasques de manera segura. Una tasca pot inserir valors a una cua per tal que una altre els llegeixi. Es permet programar un temps màxim d'espera, per inserir valors si les tasques s'executen amb velocitats diferents.

La protecció de zones de codi crítiques es pot realitzar mitjançant la utilització de mutex. Un exemple d'aquest cas es troba l'accés a WiFly. L'adaptador WIFI és un dispositiu no compartible. Per tant, cada vegada que s'està utilitzant, s'ha de bloquejar per evitar que altre fils d'execució intentin accedir-hi.

Amb aquesta petita introducció sobre com funcionen els elements de FreeRTOS utilitzats, s'explicarà el disseny del codi de l'LPC1769 implementat. Una part important consistia en la programació dels registres, com que aquesta part ja s'ha explicat anteriorment, en aquesta secció ja no se'n parlarà.

En el codi de l'arxiu principal Main.c hi trobem bàsicament dues tasques que s'encarreguen de dur a terme les accions següents:

- **Task1:** Les funcions d'aquesta tasca són:
  - Llegir l'acceleròmetre cada 100ms.
  - Connectar-se a la xarxa wifi.
  - Comunicar-se amb el servidor propi.
  - Fer parpellejar el LED vermell en el períodes de 10 segons.
  - Fer parpellejar el LED verd cada 2 segons.
  - Enviar els valors de l'acceleròmetre a la tasca TaskConvert.
  - Controla el LED vermell quan la tasca TaskConvert fa saltar l'alarma.

Aquesta tasca s'executa cada 100ms. La raó d'aquest temps, recau en la necessitat de detectar moviments bruscos que es poden produir en un període de temps molt curt. Aquesta tasca, també s'encarrega de comunicar-se amb el servidor propi. Per això, s'afegeix un control per tal que només s'hi comuniqui cada 2 segons per evitar així saturar la comunicació.

L'altre tasca que es troba en el codi s'encarrega de:

- **TaskConvert:** Les funcions que realitza són:
  - Llegir els valors de la cua enviats per la tasca anterior.
  - Convertir els valors d'acceleració a percentatges.
  - Si el canvi d'acceleració és major al 10% fer saltar l'alarma.
  - Llegir el valor de l'interruptor.
  - Controlar el LED blau segons l'estat de l'interruptor.

El pas dels valors llegits per l'acceleròmetre cap a la tasca de conversió, es realitza utilitzant un total de 3 cues. Cada cua correspon a un dels eixos de l'acceleròmetre.

S'han afegit funcions per tal d'encendre, apagar i fer parpellejar els LEDs de manera senzilla (funcions switchOnLed, switchOffLed i toggleLed respectivament). Per llegir el valor de l'interruptor, s'ha afegit la funció readSwitch.

Pel que fa a la llibreria de WiFly proporcionada, s'hi ha afegit dues funcions per facilitar la comunicació amb el servidor propi:

- WiflyOpen: Estableix una connexió amb el socket TCP del servidor propi. No es tancarà la comunicació si no es produeix cap error.
- Wifly_Send_And_Read_Response: Envia un missatge al servidor propi i en llegeix la resposta. La lectura de la resposta és important, ja que serà en ella on arribaran les instruccions que té el servidor propi per a que s'executin en l'LPC1769, com per exemple, parar l'alarma saltada.

### 4.2.2. Servidor Propi.

El servidor propi té la finalitat de comunicar l'LPC1769 amb els dispositius Android de l'usuari final.

Es va optar per un servidor programat en Java per tal de poder controlar les comunicacions de manera més precisa.

S'explicarà el mètode de comunicació que s'ha triat en cada cas i el disseny de les classes principals del servidor.

Per tal d'evitar que la comunicació de l'LPC1769 i Android amb el servidor, depengués de les IP que aquest té en cada moment, es va optar per utilitzar un sistema DNS gratuït.

El servei triat pertany a la pàgina http://www.noip.com/ i va permetre utilitzar el nom embeddedpfc.noip.me per obtenir l'adreça IP del servidor a cada moment.

#### *4.2.2.1. Comunicació LPC1769 - Servidor.*

Aquesta comunicació serà constant. Cada 2 segons el WiFly enviarà un *Keepalive.*

La primera opció va ser utilitzar el protocol HTTP. Al tractar-se d'una comunicació constant on no era necessari crear noves connexions de manera repetida, es va decidir que aquesta no era la millor opció. A més, s'afegia la necessitat d'enviar capçaleres HTTP que no aporten cap benefici a la finalitat d'aquesta comunicació.

Posteriorment, es va pensar en la possibilitat d'utilitzar un socket. És vol que els senyals *Keepalive* arribin al servidor i les respostes tornin correctament al WiFly. TCP assegura l'entrega dels missatges. Per tant, era una millor opció que UDP.

Aquest és el funcionament que es volia, és mantenir un socket TCP obert en el servidor a espera de connexions i una vegada arriba una, no tancar-lo.

Si el WiFly deixa d'enviar missatges, es necessari que el servidor ho detecti, tanqui el socket, i el torni obrir a l'espera que WiFly es torni a comunicar. És per això, que es va decidir establir un *TimeOut* de 10 segons.

El servidor TCP es troba implementat a la classe *ServidorTCP*.

#### 4.2.2.1.1 Classe ServidorTCP.

Aquesta classe s'executa sobre un fil per tal de no bloquejar l'aplicació. S'han creat 5 fils d'execució en un *pool*, per evitar problemes de memòria en la creació de nous fils, quan el WiFly deixa de respondre.

La classe, s'encarrega de crear un socket TCP en el port 5000 i resta a l'espera de missatges. Una vegada rep un missatge, comprova el tipus (*keepAlive o* Alarma) i actualitza les variables globals que guarden l'estat del LPC1769 en el servidor.

En el cas de passar 10 segons sense rebre missatges, tanca el socket per tornar-lo a obrir a l'espera de noves connexions.

### *4.2.2.2. Comunicació Servidor - Android.*

En la comunicació amb Android, s'ha d'intentar evitar que aquesta produeixi que el dispositiu no entri en estat de baix consum. A més, ha de ser una comunicació on no sigui necessari estar contínuament consultant informació per saber si alguna cosa ha canviat. És el que es coneix com *polling.*

La millor opció que es troba és la implementació d'un sistema *push* que sigui capaç d'enviar notificacions. El sistema de notificacions d'Android permet comunicar esdeveniments a les aplicacions sense que aquestes s'hagin d'encarregar de la pròpia comunicació en sí.

Google ofereix les llibreries necessàries per tal de poder generar aquest tipus de notificacions. Per poder utilitzar el seu sistema, cal registrar-se i crear un Google API Project. Se'ns proporcionaran unes claus que seran necessàries per poder utilitzar el seu servei.

Per tal de poder-se comunicar amb un dispositiu concret, és necessari disposar d'un identificador únic d'aquest. Aquest identificador no és la seva IP, sinó una cadena de caràcters generada per Google.

El funcionament general per tal de poder enviar notificacions a un dispositiu concret és el següent.

1. El dispositiu es registra als servidor de Google demanant un identificador.
2. Els servidors de Google registren el dispositiu i li proporcionen l'identificador.
3. El dispositiu envia al servidor propi el seu identificador.

Amb aquest procés, el servidor ja té l'identificador del dispositiu. Quan sigui necessari enviar una notificació a aquest, el servidor recuperarà l'identificador que ha emmagatzemat i farà una petició als servidors de Google, juntament amb el missatge a enviar i l'identificador del destinatari.

Com es veu, a part de la comunicació amb notificacions, serà necessari un sistema que permeti al dispositiu comunicar al servidor el seu identificador personal, la primera vegada que es registra.

La solució més simple en aquest cas és implementar un servidor HTTP, ja que es tracta d'una comunicació de curta durada i que es produirà molt poc sovint.

Aprofitant que s'ha de crear un servidor HTTP per a què el dispositiu pugui comunicar el seu identificador, també es farà servir per consultar l'estat de l'alarma de manera activa. És a dir, les notificacions, serviran per comunicar al dispositiu que l'alarma ha saltat, evitant que s'hagi d'estar consultant contínuament l'estat, però a la vegada, el dispositiu proporcionarà un mètode a l'usuari per a que, en cas de necessitar-ho, pugui consultar l'estat quan vulgui.

Per tal de poder implementar la comunicació amb el dispositiu, es necessita el següent:

- Servidor HTTP. Permet al dispositiu enviar el seu identificador i consultar l'estat de l'alarma de manera activa.
- Objecte per representar les dades de registre del dispositiu.
- Sistema de persistència de dades per guardar els identificadors dels dispositius a disc.
- Classe capaç de generar les peticions als servidors de Google per tal que aquest enviïn les notificacions.

#### 4.2.2.2.1 Classe HTTPServer.

Classe encarregada de crear el servidor HTTP i atendre les peticions que arribin.

El tipus de petició es distingeix segons els paràmetres que arriben a la pròpia URL. Sempre han de començar amb la paraula "test" i posteriorment, la resta de valors. Les possibilitats acceptades són:

- Paràmetre identificador del dispositiu.
  - El seu valor identifica el dispositiu que s'està comunicant, en aquest cas sempre serà "Android" per identificar que es tracta del dispositiu.
- Paràmetre identificador de l'acció a realitzar:
  - El seu valor indica l'acció que ha de realitzar el servidor. Els seus valors poden ser:
    - "register": Indica que és una petició per registrar un dispositiu.
    - "STOP": Indica que es vol aturar l'alarma si ha saltat.
    - "testNotification": Indica que es vol rebre informació de l'estat de l'alarma. Es tracta de la crida activa per conèixer l'estat del sistema.
- Paràmetre de registre de dispositiu.
  - Serà del següent tipus: "id=identificador".
    - "id": Indica que es tracta d'una petició de registre.
    - "identificador": Conté el número de registre que identifica el dispositiu.

Per exemple, els següents missatge són acceptats:

Host:port/test?android =register&id=IdentificadorDispositiu -> Demana el registrament del dispositiu.

Host:port/test?android=testNotification -> Demana, de manera activa, l'estat del sistema.

Host:port/test?android=STOP -> Demana que es pari l'alarma si ha saltat.

Aquesta classe també genera les respostes a aquestes peticions. Només la petició de dades activa generarà una resposta amb dades.

La resposta a una petició de dades seguirà la següent estructura:

- Identificador de la connexió de WiFly (wiflyconnection).

    - El seu valor serà un String indicant si el WiFly té connexió al servidor propi.
- Identificador de l'estat de l'alarma (wiflystatus).
    - El seu valor serà un String indicant l'estat de l'alarma.

Un exemple de resposta podria ser el següent.

wiflyconnection=Connected&wiflystatus=OK.

#### 4.2.2.2.2 Classe GCMData.

Aquesta classe és l'objecte on es guardaran les dades dels diferents dispositius a notificar. Es guardaran les seves ids i és l'objecte que es farà servir per la persistència de les dades.

#### 4.2.2.2.3 Classe Persistance.

Per tal de guardar els identificadors dels dispositius a notificar, entre les diferents execucions del servidor, es decideix que és necessari crear un sistema de persistència per a les dades de GCMData.

Es tracta de dades que ocupen molt poc. Utilitzar una base de dades comercial seria excessiu per la funcionalitat que es requereix en aquest cas, on les dades ocuparan pocs bytes i seran totes del mateix tipus.

L'opció escollida és serialitzar l'objecte GCMData i guardar-lo directament a disc.

La classe Persistance és l'encarregada d'aquesta tasca: Comprova l'existència prèvia del fitxer amb les dades, permet escriure i llegir l'objecte guardat en disc i, en cas que sigui la primera vegada que s'executa, genera el fitxer per poder guardar les dades.

#### 4.2.2.2.4 Classe NotificacionSender.

La classe NotificationSender és l'encarregada d'enviar notificacions als dispositius.

Proporciona un mètode on es rep el missatge a enviar. Envia les peticions a través de les llibreries de Google, juntament amb la llista de destinataris que obté a partir de Persistance.

Els servidors de Google seran els encarregats de fer arribar la notificació. Per tant, si algun del destinataris en aquell moment no té connexió a internet, la notificació romandrà als servidors de Google fins que pugui ser enviada.

### *4.2.2.3. Altres classes del servidor.*

Una vegada explicades les classes referents a la comunicació, tant en una direcció com en l'altre, s'explicaran la resta de classes que conformen el servidor propi.

#### 4.2.2.3.1 Classe Main.

És la classe principal. Conté la interfície gràfica així com les inicialitzacions necessàries per fer funcionar l'aplicació.

L'aspecte gràfic de l'aplicació és el següent.

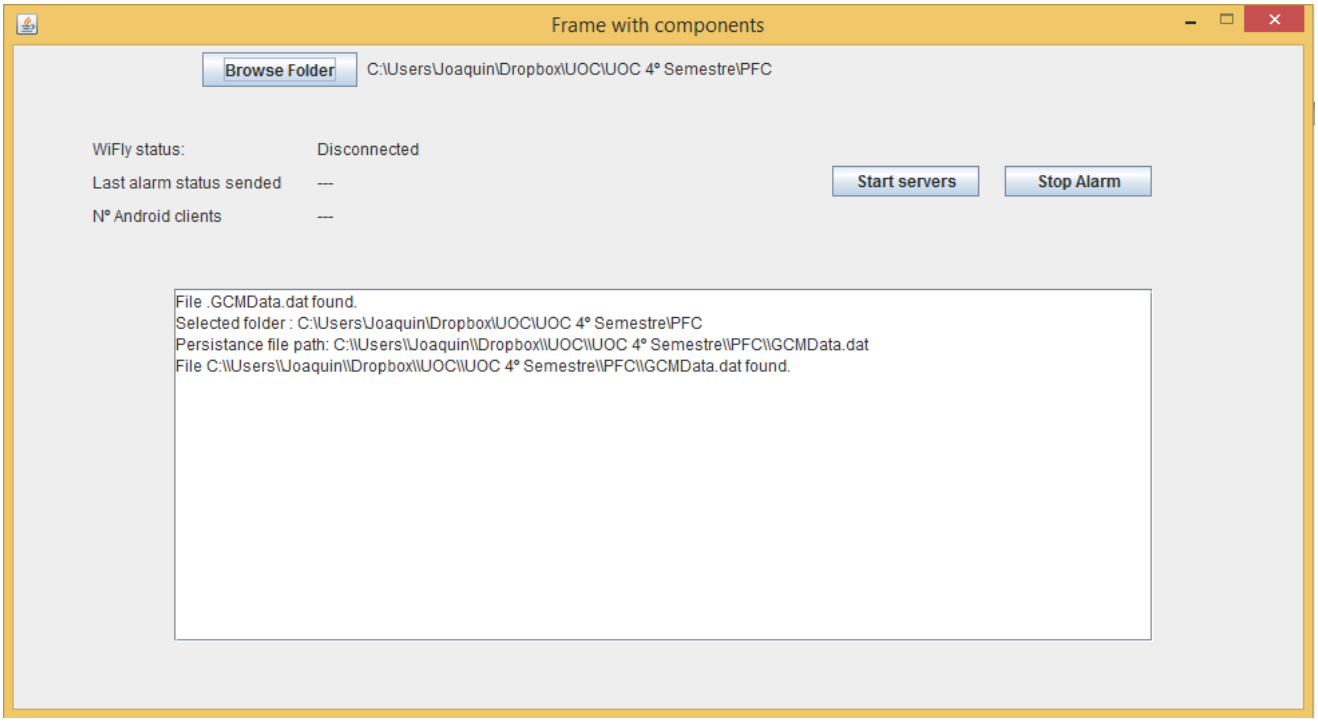


*Figura 14: UI Servidor propi*

La funció dels diferents botons és:

- Start servers: Inicia els servidors TCP i HTTP.
- Stop Alarm: Para l'alarma quan aquesta ha saltat.
- Browse Folder: Permet triar la ruta on es guardarà el fitxer de persistència.

Les etiquetes proporcionen la següent informació:

- WiFly status: Indica l'estat de la connexió a amb WiFly.
- Last alarm status sended: Indica l'últim estat que s'ha rebut de l'alarma.
- Nº Android clients: Indica el nombre total de dispositius registrats per rebre notificacions.

El quadre de text de la part inferior serveix com una consola de depuració.

#### 4.2.2.3.2 Classes StartServersActionListener i StopServerActionListener.

Aquestes dues classes contenen les accions que es realitzen al fer click sobre els botons "Start servers" i "Stop Alarm" respectivament.

#### 4.2.2.3.3 Classe ComVars.

Aquesta classe conté les variables de l'estat de l'alarma. Poden ser llegides per l'aplicació Android de manera activa a través de l'enviament de peticions.

Conté informació sobre l'estat de la connexió de l'alarma, l'últim missatge d'estat que ha enviat i si s'ha d'enviar un missatge de para l'alarma o si ha saltat.

L'accés a aquestes variables està protegit (*thread safe*) ja que diferents threads poden intentar accedir-hi de manera concurrent alhora.

#### 4.2.2.3.4. Classe FolderSelect.

Conté l'explorador de fitxer que permet seleccionar la ruta del fitxer de persistència. Només permet seleccionar directoris i no fitxers. Això és així perquè el fitxer de dades sempre tindrà el mateix nom.

Si no es tria cap ruta, per defecte és la mateixa on es troba el fitxer .jar de l'aplicació.

#### 4.2.2.3.5. Classe PFCConstants.

En aquesta classe hi ha totes les constants de l'aplicació. Els valors utilitzats que no variaran.

#### 4.2.2.3.6. Classe RefreshUI.

A RefreshUI es troba el fil d'execució encarregat d'actualitzar la informació que es veu en la interfície gràfica. Per tal d'escriure en la pantalla, els mètodes utilitzats també són *thread safe* per evitar que 2 fils intentin escriure alhora.

### 4.2.3. Aplicació Android

L'aplicació Android té la finalitat de proporcionar una interfície a l'usuari final per a poder comunicar-se amb l'alarma, encara que no estigui a prop d'ella.

Actualment Android té el 90% de la quota de mercat de telèfons intel·ligents a Espanya. Per aquest motiu, es va triar Android, ja que un gran percentatge de la població disposa d'un dispositiu que pot ser utilitzat per controlar l'alarma i que casi sempre porten a sobre.

L'aplicació permet consultar l'estat de l'alarma i el servidor sempre que es vulgui. A més, pot rebre notificacions del servidor comunicant esdeveniments.

En cas que l'alarma salti, l'aplicació farà sonar i vibrar el telèfon, com si es tractés d'una alarma del dispositiu, per d'assegurar que l'usuari se n'assabenta.

Com s'ha explicat en el servidor propi, el sistema de comunicació en la direcció Servidor-> Android es farà a través de notificacions. El sistema en sentit contrari, Android -> Servidor, es farà sobre HTTP.

#### *4.2.3.1. Comunicació Servidor – Android.*

Per utilitzar el sistema de notificacions, caldrà que el dispositiu es registri als serveis de Google per obtenir el seu identificador únic, que després, el servidor propi utilitzarà per enviar-li notificacions.

Del procés de registrament se n'encarrega la classe RegisterGCM, on gran part del codi es proporciona com a exemple en la documentació que ofereix Google sobre el sistema de notificacions.

#### 4.2.3.1.1. Classe RegisterGCM.

Aquesta classe s'encarrega de registrar el dispositiu per poder rebre l'identificador únic i posteriorment, l'envia al servidor propi.

Per utilitzar notificacions, cal que el dispositiu disposi dels *PlayServices* instal·lats. Aquesta comprovació és el primer que realitza aquesta classe.

Una vegada s'ha comprovat que el dispositiu podrà treballar amb notificacions, s'intenta recuperar l'identificador, si aquest ha estat guardat per una execució de l'aplicació anterior. Si no es troba un registre previ, comença el procés. En aquest, s'han de proporcionar les claus que es donen quan ens registrem per poder utilitzar la API.

S'envia la petició de registre en un fil d'execució, per tal d'evitar bloquejar l'aplicació.

Quan es rep la resposta amb l'identificador, aquest es guarda de manera local per tal d'evitar que en futures execucions s'hagi de tornar a demanar.

Per últim, s'envia al servidor propi l'identificador obtingut a través d'una petició HTTP, com ja s'ha explicat en l'apartat de comunicació del servidor propi.

#### 4.2.3.1.2. Classe HTTPClient.

Aquesta classe implementa el client HTTP encarregat d'enviar informació al servidor.

El tipus d'informació s'ha de muntar en els paràmetres per tal que el servidor sàpiga com llegir-la. En l'explicació del servidor, ja s'han indicat quins paràmetres poden haver, però per tal de recordar-los, es mostren a continuació.

- Paràmetre identificador del dispositiu.
  - El seu valor identificarà el dispositiu que s'està comunicant, en aquest cas sempre serà "android" per identificar que es tracta del dispositiu.
- Paràmetre identificador de l'acció a realitzar:
  - El seu valor indicarà l'acció que ha de realitzar el servidor. Els seus valors poden ser:
    - "register": Indica que és una petició per registrar un dispositiu.
    - "STOP": Indica que es vol aturar l'alarma si ha saltat.
    - "testNotification": Indica que es vol rebre informació de l'estat de l'alarma. Es tracta de la crida activa per conèixer l'estat del sistema.
- Paràmetre de registre de dispositiu.
  - Serà del següent tipus: "id=identificador".
    - "id": Indica que es tracte d'una petició de registre.
    - "identificador": Conté el número de registre que identifica el dispositiu.

La connexió té configurat un *timeout* de 3 segons, on, si no es rep resposta del servidor, es considerarà que aquest ha perdut la connectivitat o ha deixat de respondre.

#### 4.2.3.1.3. Classe ParametersBuilder.

Aquesta classe servirà per generar el paràmetres que s'envien a les peticions HTTP, d'una manera més senzilla.

Conté 3 mètodes, que ja tenen predefinits els paràmetres, per als 3 tipus de missatges que es poden enviar al servidor propi (registrar id, parar alarma i consulta activa d'informació).

#### 4.2.3.1.4. Classe SendAsyncAction.

Si es realitzessin les peticions HTTP sobre el fil principal de l'aplicació, aquesta quedaria bloquejada esperant la resposta.

Per evitar aquest comportament es fa servir la classe SendAsyncAction.

Aquesta classe crea tasques que s'executen en segon pla per tal d'enviar les peticions HTTP sense bloquejar l'aplicació.

Una vegada ha enviat la petició al servidor propi, també s'encarrega de mantenir-se a l'espera de la resposta i de tractar la informació rebuda, per actualitzar la interfície gràfica de l'aplicació amb les noves dades de l'estat de l'alarma i el servidor que s'han rebut.

#### 4.2.3.1.5 Classes NotificationReceiver i NotificationHandler

Aquestes dues classes s'encarreguen de rebre i tractar les notificacions que arriben a l'aplicació.

NotificationReceiver s'encarrega de despertar el dispositiu del seu estat de baix consum, en cas que s'hi trobi, i de passar la notificació a NotificationHandler per a què la tracti.

NotificationHandler filtre el tipus de missatge rebut, només en el cas que sigui una notificació d'alarma, farà que el dispositiu comenci a sonar de manera continuada.

Una vegada s'ha obtingut el tipus de missatge, es recuperarà la cadena de text que conté i es generarà una notificació en el sistema, on es mostrarà la informació rebuda.

Totes les notificacions rebudes recuperaran el missatge i el mostraran a l'usuari en una notificació d'Android.

L'aspecte de les notificacions generades en el sistema és el següent.

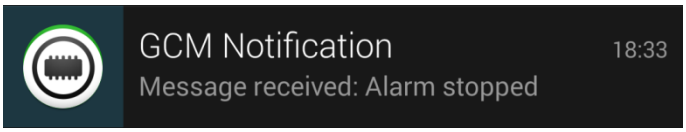


*Figura 15: Exemple notificació generada.*

#### *4.2.3.2. Alarma Android.*

Quan es rep una notificació del servidor propi, indicant que l'alarma ha saltat, s'ha d'intentar notificar a l'usuari de manera que no confongui amb una notificació d'un altre aplicació, de la qual, potser no faria cas en aquell moment.

Per tal de cridar l'atenció de l'usuari, i només en els casos en que l'alarma ha saltat, el dispositiu vibrarà i sonarà independentment de la configuració de so de telèfon que es tingui configurada en aquell moment.

L'aplicació utilitzarà el volum i so que té l'alarma del propi dispositiu assignats. Encara que el dispositiu es trobi en silenci pel que fa a la configuració del so del timbre del telèfon.

##### 4.2.3.2.1. Classe AlarmScheduler.

Quan la classe que tracta les notificacions en rep una amb el senyal d'alarma, cridarà a AlarmScheduler per tal de generar una alarma al propi dispositiu.

AlarmScheduler s'encarrega d'utilitzar el sistema de programació de tasques d'Android per tal de programar-ne una que soni de manera casi immediata, amb només 500 ms de retràs entre que es rep i aquesta sona.

Per encendre la pantalla per mostrar un missatge, vibrar i sonar; es crida a la classe AlarmActivity.

#### 4.2.3.2.2. Classe AlarmActivity.

Aquesta classe conté els elements visuals que es mostraran per pantalla al rebre l'alarma. Posteriorment cridarà a AlarmAlert amb la finalitat que el dispositiu comenci a sonar.

El missatge que es mostra per pantalla és el següent.

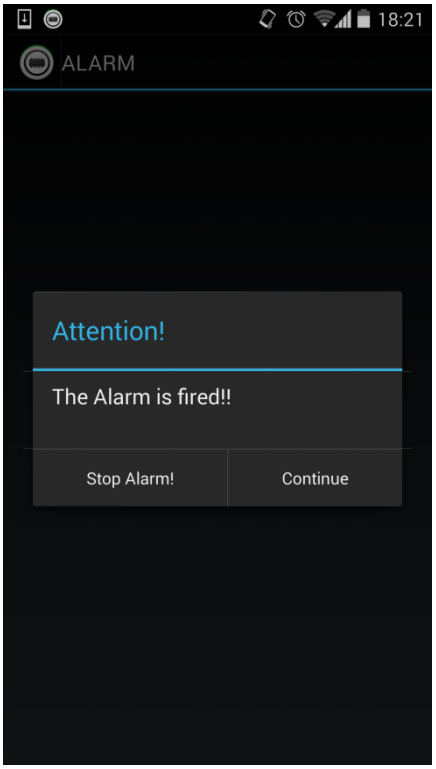


*Figura 16: Missatge Alarma.*

#### 4.2.3.2.3. Classe AlarmAlert.

Classe encarregada d'engegar la pantalla del dispositiu, encara que aquest estigui bloquejat i amb contrasenya. També el farà sonar i vibrar, si disposa de motor de vibració.

Mostra un missatge per pantalla on avisa que l'alarma ha saltat, i dona la possibilitat de triar entre continuar cap a l'aplicació o parar l'alarma des del propi missatge.

Utilitza una instància del reproductor de música per fer sonar l'alarma. Per tal de vibrar, fa una crida al servei que permet aquesta acció i el manté durant 2 segons.

Si l'usuari tria qualsevol de les dues opcions, l'alarma deixarà de sonar, ja que ha rebut el missatge.

#### *4.2.3.3. Altres classes.*

A continuació s'explicaran la resta de classe que formen l'aplicació.

##### 4.2.3.3.1. Classe PFCConstants

Conté les constants de l'aplicació. Identificadors de missatges, missatges per mostrar a l'usuari, etc.. En general, tots aquells valors que no varien.

##### 4.2.3.3.2. Classe MainActivity

Aquesta classe inicialitza l'aplicació i tots els objectes necessaris, així com les seves variables, perquè puguin funcionar correctament.

Estableix les accions dels botons de pantalla i mostra la interfície gràfica de l'aplicació.

La interfície gràfica es troba definida al fitxer debug.xml dins el directori /res/layout del projecte Android de l'aplicació.

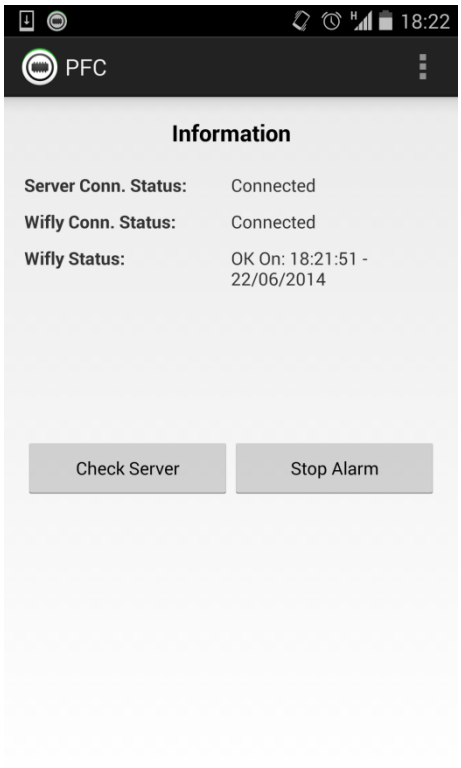


*Figura 17: Interfície gràfica aplicació Android..*

#### *4.2.3.4. Permisos de l'aplicació.*

A continuació es detallaran els permisos utilitzats per l'aplicació. Aquests estan definits al fitxer AndroidManifest.xml a l'arrel del projecte.

- Buscar comptes en el dispositiu: Per tal de comprovar si el dispositiu pot rebre notificacions i guardar el seu identificador.
- Accés complet a la xarxa: Proporciona accés a internet a l'aplicació per poder-se comunicar amb la resta del sistema.
- Rebre dades d'internet: Permet rebre notificacions.
- Controlar la vibració: Permet fer vibrar el dispositiu al rebre un senyal d'alarma.
- Impedir que el telèfon entri en mode suspensió. Permet "despertar" en dispositiu si es troba bloquejat per tal de mostrar el missatge d'alarma i sonar.

# 5. Valoració econòmica.

Es farà una valoració econòmica per sobre, suposant que el servidor propi és un ordinador de la nostra propietat. Com es comentarà a la part de possibles millores. Adaptar l'aplicació per treballar en servidors web permetria reduir els costos.

S'inclourà el cost d'un dispositiu Android de gama mitja-baixa, ja que els requisits de l'aplicació són mínims.

| Concepte | Quantitat | Preu unitari | Preu total |
|---|---|---|---|
| LPC1769 | 1 | 10 € | 10 € |
| WiFly | 1 | 30 € | 30 € |
| MMA7361 | 1 | 10 € | 10 € |
| Servidor | 1 | 300 € | 300 € |
| LEDs, resistències, etc… | 1 | 2 € | 2 € |
| Dispositiu Android | 1 | 200 € | 200 € |
| Desenvolupament | 180 hores | 10 € | 1.800 € |
| Total | | | 2.352 € |

Taula 14: Registres FIOPIN

El cost inicial és alt. Adaptant l'aplicació per funcionar en servidors web, es podria reduir el cost de manera important. També cal tenir en compte que si el sistema es comercialitzés, els costos de desenvolupament es repartirien.

# 6. Conclusions.

El funcionament final del sistema complet ha sigut l'esperat. S'han pogut implementar totes les parts i el seu funcionament és el desitjat. Tot i això, han quedat alguns aspectes per polir.

## 6.1. Avaluació.

La realització d'aquest projecte m'ha permès aprendre com funcionen els sistemes en temps real en els sistemes encastats.

He pogut entendre millor com programar microcontroladors, i la importància de llegir els seus *datasheets* per tal d'entendre com funcionen i saber com configurar-los, per poder programar sobre ells correctament.

Les implicacions de comunicació del projecte, m'han permès treballar amb la generació de notificacions a Android, ús de clients i servidors HTTP simples.

En general, el projecte m'ha fet aprendre molts aspectes nous dels sistemes encastats i també patir moltes hores per poder treure la feina endavant.

## 6.2. Possibles millores.

Principalment crec que hi hauria 2 millores que són destacables.

1. Implementar tot el sistema sobre un servidor web enlloc d'un servidor propi:
   Tècnicament des del meu punt de vista, no aportaria cap benefici. La millora vindria donada per la facilitat d'instal·lar una aplicació web en un servidor i tindre-la funcional per un cost anual molt petit.
   Per poder instal·lar l'aplicació Java actual, s'hauria d'utilitzar un servidor dedicat.
2. Instal·lació d'un sistema de contrasenya per desactivar l'alarma. Actualment l'alarma es pot desactivar (sempre i quan no hagi saltat) amb la desactivació de l'interruptor.
   La utilització d'una botonera per poder introduir una contrasenya, afegiria protecció extra al sistema.

# 7. Bibliografia.

Notificacions: http://developer.android.com/guide/topics/ui/notifiers/notifications.html

Servei GCM: http://developer.android.com/google/gcm/gs.html

Alarma Android: http://developer.android.com/reference/android/app/AlarmManager.html

*Datasheet* LPC1769: http://www.nxp.com/documents/user_manual/UM10360.pdf

Comandes WiFly: http://ww1.microchip.com/downloads/en/DeviceDoc/50002230A.pdf

*Datacheet* Acceleròmetre:
http://www.apexelectrix.com/PDFs/MMA7361/MMA7361_module_datasheet.pdf

NXP Knowledgebase: http://knowledgebase.nxp.com

Wiki sistemes encastats:
http://cv.uoc.edu/webapps/xwiki/wiki/matembeddedsystemslabhome/view/Material/IniciCortexM3/

Threads Java: http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html

# 8. Annex.

S'explicaran els diversos passos que es segueixen per iniciar tot el sistema.

## 8.1 Instal·lació servidor propi.

Per instal·lar el servidor propi només caldrà assegurar-se que l'ordinador sobre el que treballem té instal·lat Java.

Per executar el servidor, cal fer doble click sobre el fitxer ServidorPropi.jar i aquest s'obrirà. Una vegada obert, farem click a "Start servers" i ja tindrem el servidor funcionant.

## 8.2 Instal·lació de l'aplicació Android.

Per instal·lar aquesta aplicació haurem d'habilitar els permisos d'execució d'aplicacions de fora de Play Store. Això es pot fer als ajustos de seguretat del dispositiu.

Amb l'execució d'aplicacions externes activada, només caldrà enviar l'arxiu PFC.apk al dispositiu (per exemple per e-mail). Des del dispositiu, tocant el fitxer, apareixerà el diàleg d'instal·lació on es mostren els permisos.

Una vegada l'aplicació està instal·lada, ja es pot obrir per tal que es registri al servidor.

## 8.3 Compilació LPC1769.

Per poder compilar l'aplicació i enviar-la al LPC1769, primer de tot, caldrà copiar les carpetes que componen el codi del LPC1769 al Workspace que tinguem configurat a l'IDE de desenvolupament LPCXpresso.

Obrim l'IDE i seleccionem la carpeta del Workspace. Una vegada tenim LPCXpresso obert, s'haurà d'importar el projecte. Per fer això, anem a la pestanya File->Import.

S'obrirà el quadre de diàleg que es mostra a continuació on seleccionarem General -> Existing Projects into Wokspace

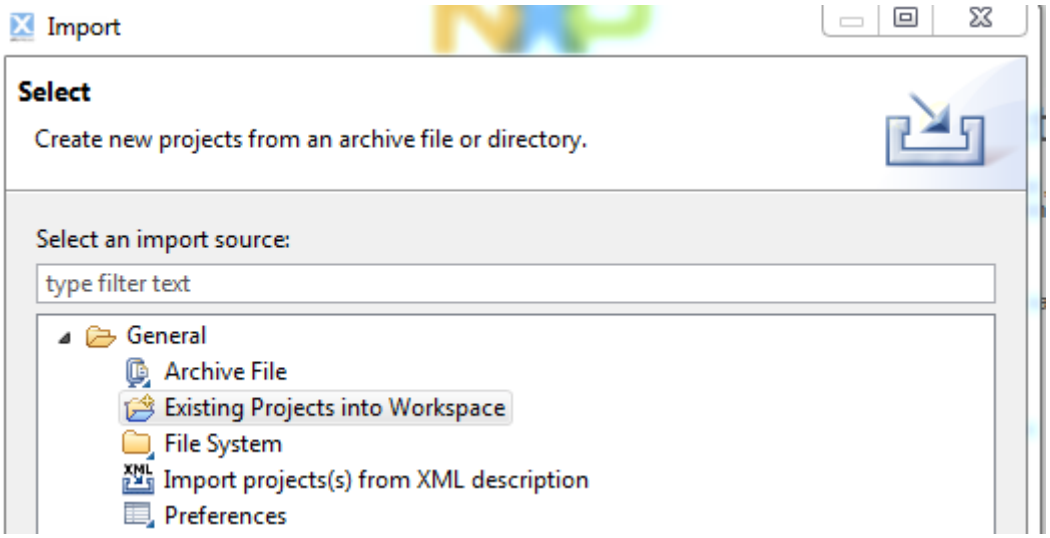


*Figura 18: Compilar codi LPC1769 - 1*

En la següent finestra seleccionarem "Browse" i marcarem la carpeta del Workspace.

A continuació, en l'apartat Projects de la finestra, ens apareixeran els 4 elements que conformen el codi. Els haurem de marcar tots.

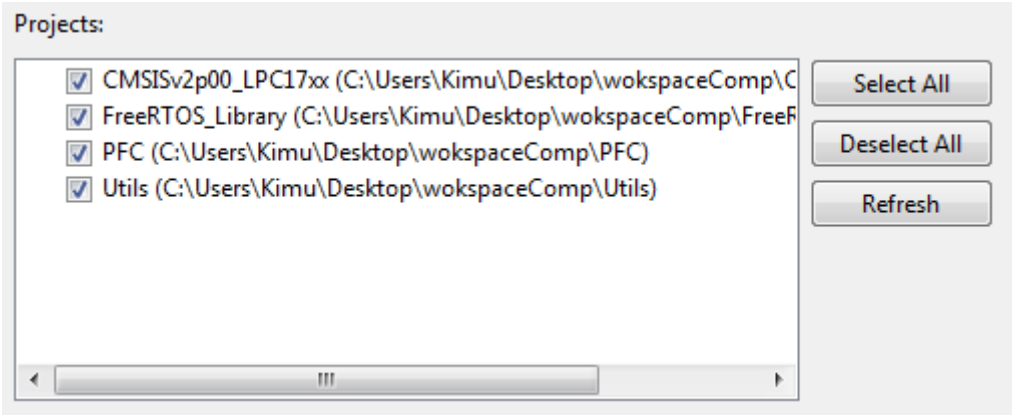


*Figura 19: Compilar codi LPC1769 - 2*

Una vegada macats, es pot fer click a Finish. Al Project Explorer, ens apareixeran els 4 elements de la figura:

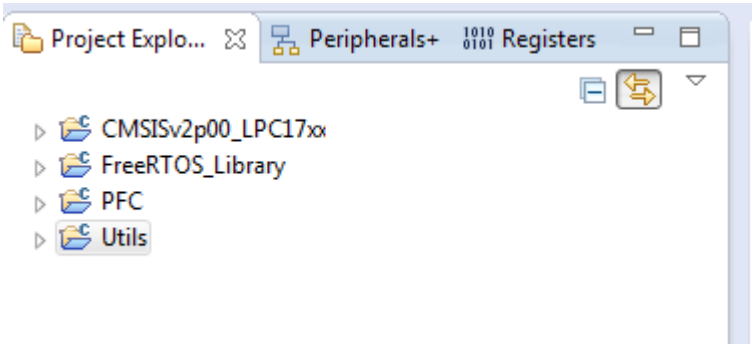


*Figura 20: Compilar codi LPC1769 - 3*

Una vegada tenim el projecte importat, l'haurem de compilar.

Per compilar el projecte seleccionarem cada un dels elements, deixant PFC per l'últim, i farem click a la icona de compilar (Martell en la imatge).

*Figura 21: Compilar codi LPC1769 - 4*

Quan ja tenim tot compilat, podem carregar el codi al LPC1769 a través de la icona de debug, que a més, una vegada carregat el codi, ens permetrà depurar-lo si ho desitgem (Escarabat en la imatge).

*Figura 22: Compilar codi LPC1769 - 5*

A partir d'aquest moment el codi ja està al LPC1769 i per tant ja pot funcionar sense necessitat d'estar connectat a un ordinador. Només necessita estar alimentat.

## 8.4 Compilació aplicació Android.

Per poder compilar l'aplicació Android, serà necessari importar el projecte a Eclipse juntament amb les llibreries per les notificacions.

L'Eclipse sobre el que s'importi el projecte ha de tenir instal·lat l'SDK d'Android prèviament.

Per importar el projecte, fem click a "File->Import". Seleccionem "Existing Android Code Into Workspace".

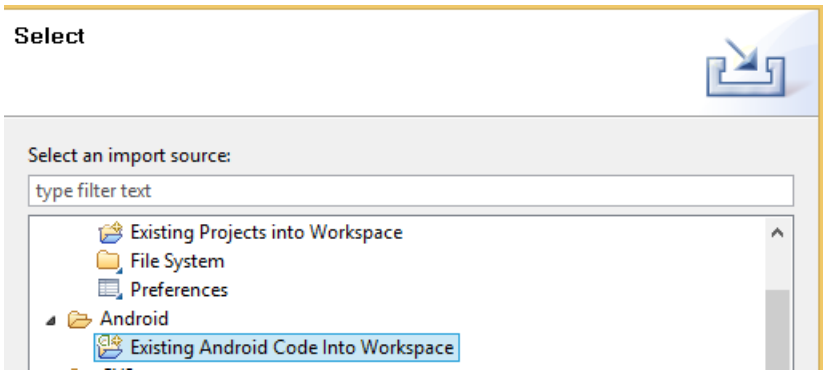


*Figura 23: Compilar aplicació Android - 1*

A la següent finestra, fent click a "Browse," seleccionem la carpeta on hi ha l'aplicació i la llibreria. En la secció "Projects" apareixeran 2 elements, que caldrà seleccionar. Una vegada seleccionats, fem click a "Finish".

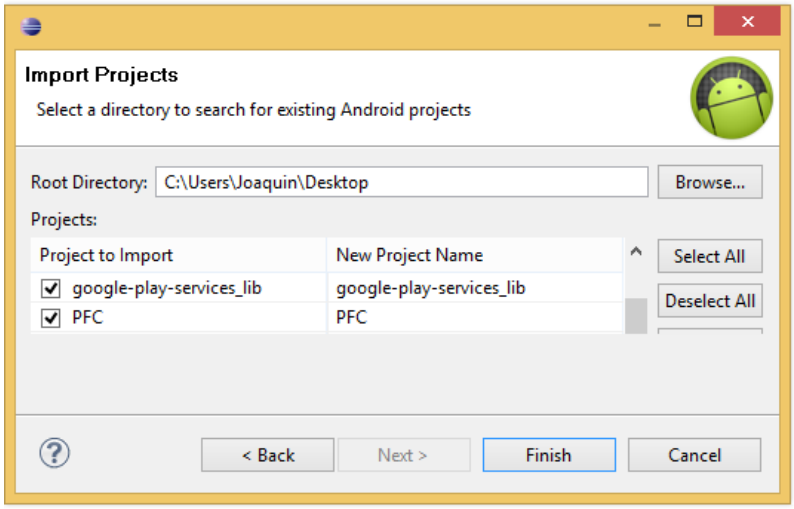


*Figura 24: Compilar aplicació Android - 2*

En el "Package Explorer", ara apareixeran els dos projectes, però un d'ells tindrà errors per falta de llibreries.

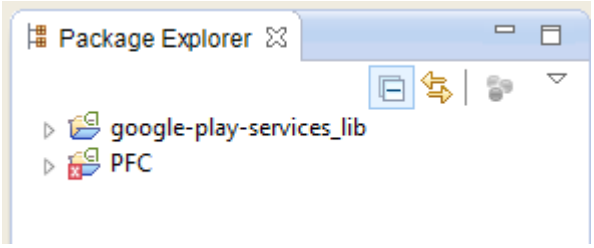


*Figura 25: Compilar aplicació Android - 3*

Serà necessari afegir la ruta de les llibreries a la compilació Android. Per fer això, fem click amb el botó dret al projecte PFC, i seleccionem "Properties". A la finestra que s'obre, anem a Android, i en l'apartat "Library", apareixerà una llibreria amb una creu. Fem click sobre la llibreria i després sobre "Remove". A continuació, fem click a "Add" i a la finestra que s'obre seleccionem el projecte de la llibreria. Fem click a "OK" a les "Properties" i ara ja apareixerà el projecte al "Package Explorer" sense cap error.

Una vegada acabat aquest procés, ja es pot compilar i executar l'aplicació fent click a "Run".

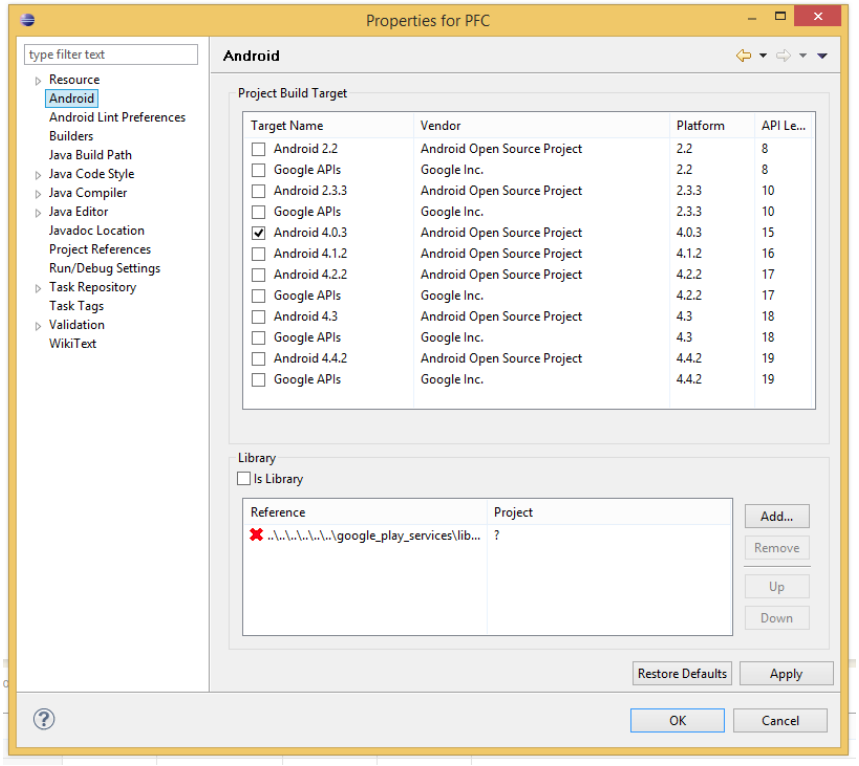


*Figura 26: Compilar aplicació Android - 4*

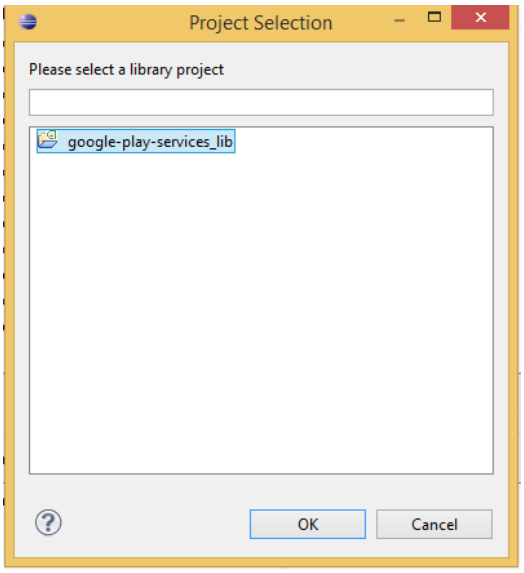


*Figura 27: Compilar aplicació Android - 5*

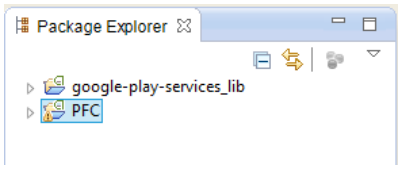


*Figura 28: Compilar aplicació Android - 5*

## 8.5 Compilació Servidor propi.

Per poder compilar el servidor propi, seguirem el mateix procés d'importació a Eclipse que hem utilitzat fins ara.

Anem a "File -> Import" i seleccionem "Existing Projects into Workspace". A la finestra que aparegui, fent click a "Browse", busquem el directori on es troba el projecte. A la llista de projects, seleccionem el projecte que ha aparegut i fem click a "finish". El projecte apareixerà al "Package Explorer" i ja el podrem executar fent click a "Run".

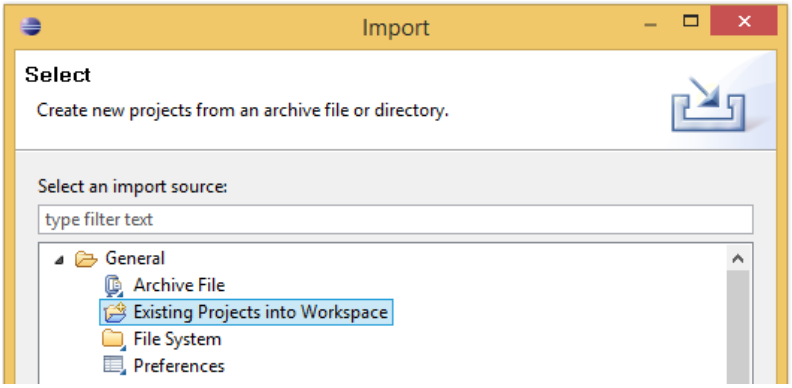


*Figura 29: Compilar Servidor propi – 1*

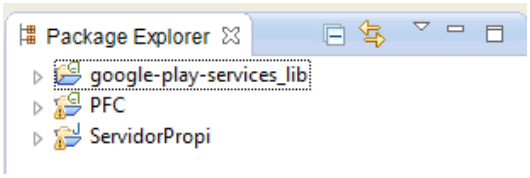


*Figura 30: Compilar Servidor propi – 2*