SISTEMA DE MONITORITZACIÓ DE PACIENTS MITJANÇANT UNA XARXA DE SENSORS INALÀMBRICS

Autor: Ignacio López Chaveinte

TFM Enginyeria Informàtica – Sistemes encastats

Consultor: Antoni Morell

Data d'entrega: 13/06/2015

Al meu pare

Índex de continguts

Índex de taules i figures

1. Introducció

1.1. Justificació del projecte

Els sistemes encastats són dispositius de capacitat limitada, dedicats normalment a la realització d'una única aplicació amb una funcionalitat concreta. Es fan servir en molts contextos diferents. Darrerament s'està generalitzant l'ús dels anomenats *wearables*, que l'usuari porta sobre el seu cos per monitoritzar paràmetres com moviment, constants biològiques i d'altres, i que tenen múltiples aplicacions, de tipus mèdic, esportiu o simplement lúdic.

En aquest treball s'ha portat a terme el disseny i implementació d'un sensor de caigudes i monitor del ritme cardíac per al control dels pacients d'una planta hospitalària, mitjançant un sistema encastat portable.

Desenvolupar un sistema encastat permet assolir una visió molt completa del sistema a construir, des del hardware, el software, la interacció entre els dos, les comunicacions... Això és molt més difícil en aplicacions sobre sistemes de propòsit més general, on el desenvolupador treballa a un nivell d'abstracció més elevat. Aquesta visió completa del sistema ha estat el factor més important a l'hora de decidir-me a fer aquest projecte.

1.2. Descripció del projecte

En una planta d'hospital amb pacients ingressats, es vol detectar les caigudes que aquests puguin patir, per tal de proporcionar-los assistència immediata. Les caigudes més habituals i violentes que es poden produir en pacients ingressats són caigudes del llit o a la dutxa, que solen venir acompanyades d'una fase de caiguda lliure. A més, es vol poder monitoritzar el ritme cardíac dels pacients, per exemple per controlar el seu estat d'estrès. S'ha decidit que la millor forma de fer-ho és mitjançant una xarxa de sensors inalàmbrics (WSN).

Els sensors estaran acoblats amb una placa amb un microcontrolador, que els tractarà com a perifèrics d'entrada de dades sobre l'activitat cardíaca i l'acceleració dels pacients. El sistema també

estarà acoblat amb una placa de comunicacions wifi, que dotarà al sistema de connectivitat.

La taxa de batecs serà enviada a un servidor remot per mostrar-les al personal de l'hospital. Per la seva banda, l'acceleròmetre informarà al microcontrolador de si s'ha produït una caiguda (condició de 0g), i aquest enviarà una alarma al servidor, que mostrarà el missatge pertinent.

En resum, el projecte haurà de desenvolupar els següent elements:

- un sistema encastat per a l'adquisició de dades, amb
  - un placa amb microcontrolador.
  - un conjunt de sensors intel·ligents:
    - acceleròmetre.
    - sensor de batecs.
  - un mòdul per a la comunicació per ràdio entre sistema i el punt d'accés wifi.
- un servidor remot, que rebrà els diferents missatges que se li enviïn des dels nodes sensors.

A la figura 1 podem veure el diagrama general del sistema

1.3. Objectius del projecte

Els objectius que ha de complir el sistema per tal de satisfer les necessitats dels usuaris són:

- obtenir la taxa de batecs del pacient.
- detectar quan s'ha produït una caiguda.
- estimar la càrrega de la bateria a partir del seu voltatge de sortida.
- enviar les dades anteriors (ritme cardíac, alarma de caiguda i alarma de bateria baixa) al servidor remot a través d'una xarxa *wireless* 802.11 b/g. Les dades de ritme cardíac i de bateria baixa s'enviaran a intervals regulars prefixats i l'alarma de caiguda s'enviarà tan bon punt es produeixi.
- el servidor remot ha de mostrar les dades rebudes a l'usuari mitjançant una GUI.
- el sistema ha de detectar problemes de comunicació i ha de tractar de solucionar-los.

En cas que donés temps, es poden plantejar una sèrie d'objectius suplementaris, com ara:

- implementar varis modes de funcionament, a escollir per l'usuari des de la GUI:
  - detectar només batecs.
  - detectar només caigudes.
  - detectar batecs i caigudes.
- el servidor ha de detectar si una mota està operativa o no, i indicar-ho.

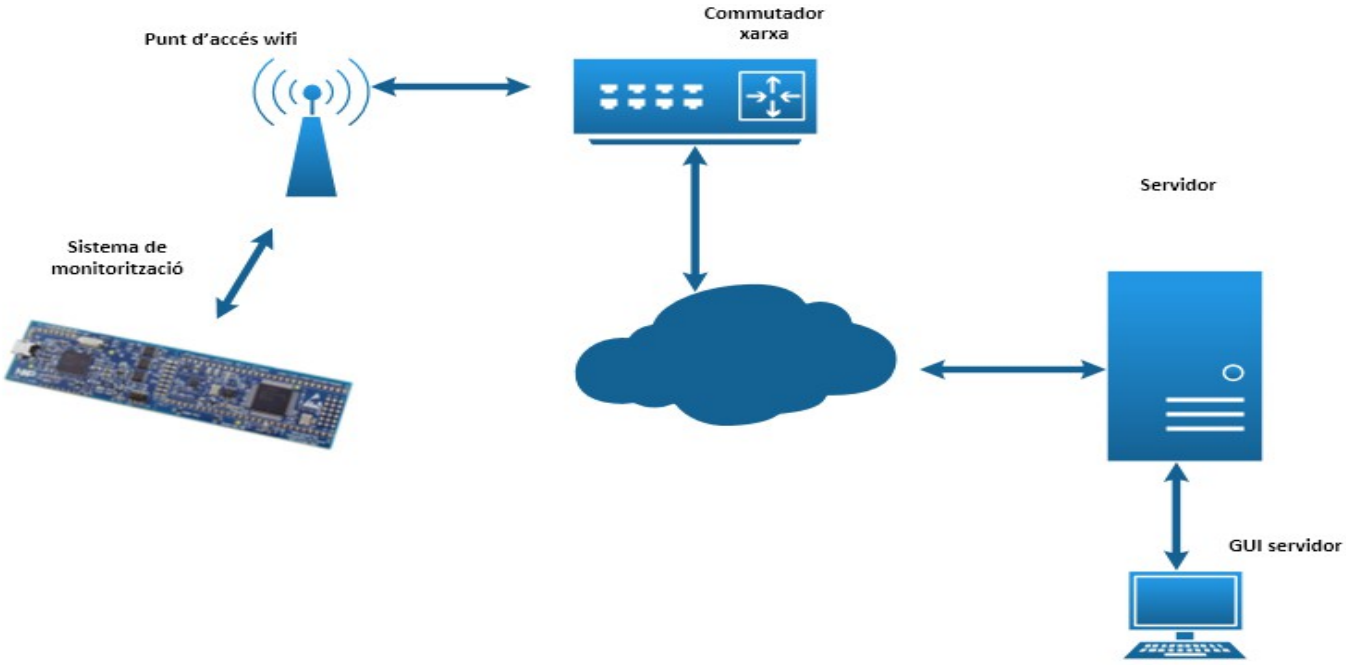


Figura 1: Esquema general del sistema

1.4. Línia de treball

La línia de treball consta de vàries fases, com es pot veure a la planificació del projecte (punt 1.5):

- **Documentació i proves** amb el *kit* de *hardware*, a partir de la *wiki* de l'assignatura i dels manuals recomanats.
- **Planificació** del projecte i presa de r**equisits**.
- **Disseny i implementació** de les les diferents funcionalitats:
  - sensor de caigudes.

- sensor de batecs.
- detecció de nivell de bateria.

Per provar les implementacions anteriors, farem un servidor de funcionalitat reduïda, que simplement mostri per consola els missatges rebuts. Quan totes les funcionalitats estiguin implementades, es combinaran en un únic sistema, creant i planificant les tasques necessàries, i es refinarà el servidor per dotar-lo de les funcionalitats gràfiques.

Com es veu, es va procedint d'una forma iterativa, afegint un requisit a implementar en cada iteració.

- Al final de cada iteració s'aniran fent els **tests** corresponents per comprovar que cada part funciona correctament, i en acabar es realitzarà un test de tot el sistema integrat.
- En paral·lel a tot l'anterior s'haurà de fer la **documentació** i redacció de la memòria.

1.5. Planificació del projecte

A l'hora de planificar el projecte, les fites han estat les diverses entregues planificades en forma de PACs:

- Decisió del projecte i comunicació al consultor. 4 de març
- PAC1 – Lliurament de la planificació del treball. 11de març
- PAC2 – Primer lliurament del projecte. 22 d'abril
- PAC3- Segon lliurament del projecte 27 maig
- Lliurament de la memòria final 13 juny
- Lliurament de la presentació i el codi 19 juny
- Inici del tribunal 22 juny
- Fi del tribunal 28 juny

A partir d'aquí, he definit les següents tasques de primer nivell, com mostra la Taula 1:

Nom de la tasca	Començament	Final
PAC1 - Decisió i planificació del projecte	25/febrer	11/març
PAC2 – Primer lliurament parcial de codi del projecte	12/març	22/abril
PAC3 – Segon lliurament parcial de codi del projecte	23/abril	27/maig
Lliurament de la memòria final	11/març	13/juny
Lliurament de la presentació i final del codi	14/juny	19/juny
Tribunal	22/juny	28/juny

Taula 1: Tasques de primer nivell

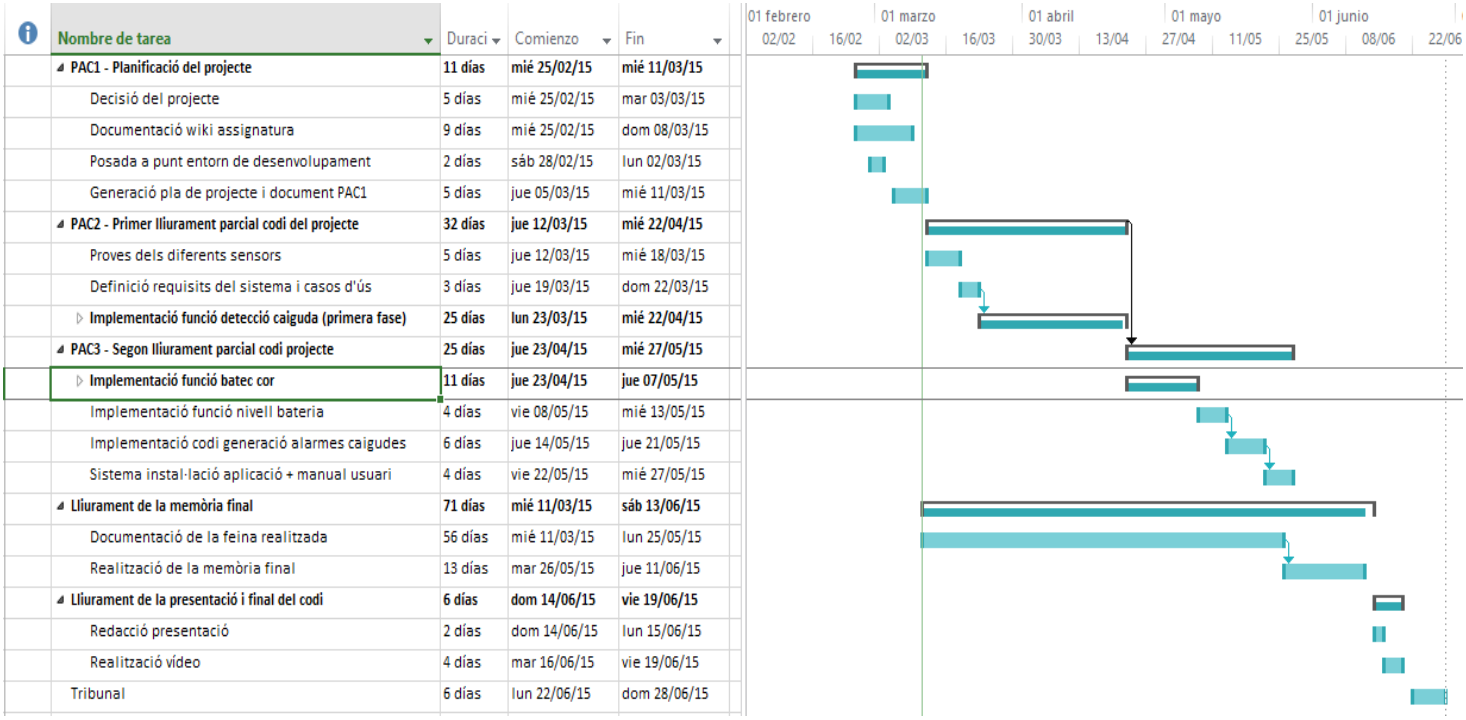
La figura 2 mostra el diagrama de Gantt inicial. A mesura que ha anat avançant el projecte, s'ha hagut de canviar alguns aspectes de la planificació:

- el codi de generació d'alarmes ha quedat ja pràcticament enllestit en la primera fase.
- s'ha allargat la implementació de la funció de detecció de batecs, degut al retard en la rebuda dels sensors i en que s'han provat un parell de sensors per tal d'escollir el més adient.
- per tant, s'ha retardat el començament de la funció de detecció del nivell de bateries.

Tenint en compte aquests canvis, la planificació de cadascuna de les activitats ha estat la següent:

PAC1 – Decisió i planificació del projecte (25/02 – 11/03)

- decisió del projecte (25/02-03/03)
- documentació sobre el *kit* de hardware al *wiki* de l'assignatura (25/02 – 08/03)



Nombre de tarea	Duraci	Comienzo	Fin
PAC1 - Planificació del projecte	**11 días**	**mié 25/02/15**	**mié 11/03/15**
Decisió del projecte	5 días	mié 25/02/15	mar 03/03/15
Documentació wiki assignatura	9 días	mié 25/02/15	dom 08/03/15
Posada a punt entorn de desenvolupament	2 días	sáb 28/02/15	lun 02/03/15
Generació pla de projecte i document PAC1	5 días	jue 05/03/15	mié 11/03/15
PAC2 - Primer lliurament parcial codi del projecte	**32 días**	**jue 12/03/15**	**mié 22/04/15**
Proves dels diferents sensors	5 días	jue 12/03/15	mié 18/03/15
Definició requisits del sistema i casos d'ús	3 días	jue 19/03/15	dom 22/03/15
Implementació funció detecció caiguda (primera fase)	**25 días**	**lun 23/03/15**	**mié 22/04/15**
PAC3 - Segon lliurament parcial codi projecte	**25 días**	**jue 23/04/15**	**mié 27/05/15**
Implementació funció batec cor	**11 días**	**jue 23/04/15**	**jue 07/05/15**
Implementació funció nivell bateria	4 días	vie 08/05/15	mié 13/05/15
Implementació codi generació alarmes caigudes	6 días	jue 14/05/15	jue 21/05/15
Sistema instal·lació aplicació + manual usuari	4 días	vie 22/05/15	mié 27/05/15
Lliurament de la memòria final	**71 días**	**mié 11/03/15**	**sáb 13/06/15**
Documentació de la feina realitzada	56 días	mié 11/03/15	lun 25/05/15
Realització de la memòria final	13 días	mar 26/05/15	jue 11/06/15
Lliurament de la presentació i final del codi	**6 días**	**dom 14/06/15**	**vie 19/06/15**
Redacció presentació	2 días	dom 14/06/15	lun 15/06/15
Realització vídeo	4 días	mar 16/06/15	vie 19/06/15
Tribunal	6 días	lun 22/06/15	dom 28/06/15

Figura 2: Diagrama de Gantt inicial

- posada a punt de l'entorn de desenvolupament i proves amb el mateix (28/02 – 02/03)
- generació del pla de projecte i resta de lliurables PAC1 (esquema memòria) (05/03 – 11/03)

PAC2 – Primer lliurament parcial (12/03 – 22/04)

- fer proves amb el diferents sensors / microcontrolador / ràdio (12/03 – 18/03)
- requisits del sistema (19/03 – 22/03)
- implementació de la funció de detecció de caiguda (23/03 – 22/04)
  - definició protocol comunicació sensor / servidor (23/03 – 27/03)
  - codi adquisició de dades acceleròmetre (28/03 – 02/04)
  - codi comunicacions amb servidor + codi servidor (recepció i presentació de dades) (03/04 – 17/04)
  - proves codi i integració (18/04 – 22/04)

PAC3 – Segon lliurament parcial (23/04 – 27/05)

- implementació funció batec cor (23/04 – 20/05)
    - codi adquisició de dades (23/04 – 10/05)
    - codi comunicació + servidor (11/05 – 18/05)
    - proves (19/05 – 20/05)
- implementació funció nivell de bateria (21/05 – 24/05)
- sistema instal·lació aplicació + manual d'usuari (25/05 – 28/05)

Lliurament de la memòria final (11/03 – 13/06)

- documentació de la feina realitzada (11/03 – 27/05)
- redacció de la memòria final (28/05 – 13/06)

Lliurament de la presentació i final del codi (14/06 – 19/06)

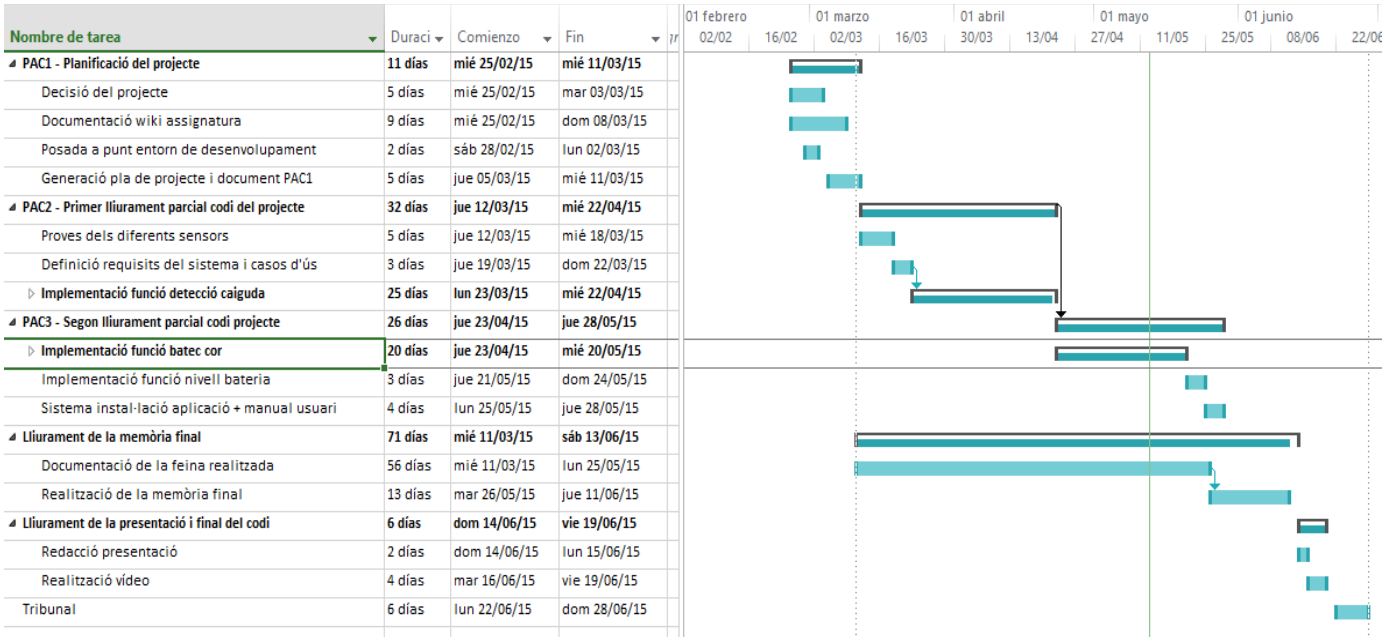
- redacció de la presentació (14/06 – 15/06)
- realització vídeo (16/06 – 19/06)

Tribunal (22/06 – 28/06)

Podem veure el diagrama de Gantt definitiu a la figura 3.

1.7. Recursos emprats

Hem de tenir en compte que alguns dels elements hardware i software venien ja predeterminats pel projecte, així que podem considerar-los un requisit no funcional del mateix.



Nombre de tarea	Duraci	Comienzo	Fin
PAC1 - Planificació del projecte	**11 días**	**mié 25/02/15**	**mié 11/03/15**
Decisió del projecte	5 días	mié 25/02/15	mar 03/03/15
Documentació wiki assignatura	9 días	mié 25/02/15	dom 08/03/15
Posada a punt entorn de desenvolupament	2 días	sáb 28/02/15	lun 02/03/15
Generació pla de projecte i document PAC1	5 días	jue 05/03/15	mié 11/03/15
PAC2 - Primer lliurament parcial codi del projecte	**32 días**	**jue 12/03/15**	**mié 22/04/15**
Proves dels diferents sensors	5 días	jue 12/03/15	mié 18/03/15
Definició requisits del sistema i casos d'ús	3 días	jue 19/03/15	dom 22/03/15
Implementació funció detecció caiguda	**25 días**	**lun 23/03/15**	**mié 22/04/15**
PAC3 - Segon lliurament parcial codi projecte	**26 días**	**jue 23/04/15**	**jue 28/05/15**
Implementació funció batec cor	**20 días**	**jue 23/04/15**	**mié 20/05/15**
Implementació funció nivell bateria	3 días	jue 21/05/15	dom 24/05/15
Sistema instal·lació aplicació + manual usuari	4 días	lun 25/05/15	jue 28/05/15
Lliurament de la memòria final	**71 días**	**mié 11/03/15**	**sáb 13/06/15**
Documentació de la feina realitzada	56 días	mié 11/03/15	lun 25/05/15
Realització de la memòria final	13 días	mar 26/05/15	jue 11/06/15
Lliurament de la presentació i final del codi	**6 días**	**dom 14/06/15**	**vie 19/06/15**
Redacció presentació	2 días	dom 14/06/15	lun 15/06/15
Realització vídeo	4 días	mar 16/06/15	vie 19/06/15
Tribunal	6 días	lun 22/06/15	dom 28/06/15

Figura 3: Diagrama de Gantt definitiu

1.7.1. Recursos hardware

El projecte s'ha muntat partint del *kit* proporcionat UOC, compost pels següents dispositius:

- **LPCXpresso LPC1769**

  Aquesta placa[1] (veure figura 4) proporciona una plataforma de desenvolupament de baix consum. Algunes de les característiques de la placa són:

  - Microcontrolador ARM Cortex M-3, amb arquitectura Harvard i 3 busos separats per a instruccions, dades i perifèrics. La freqüència a la que opera és de fins a 120 MHz.
  - SRAM *on-chip* de 64 kB
  - Memòria *flash on-chip* de 512 kB.
  - Múltiples interfícies sèrie, entre les quals:
    - controlador USB 2.0.

1 Podem trobar un esquema complert de la placa a: http://www.embeddedartists.com/sites/default/files/docs/schematics/LPCXpressoLPC1769revB.pdf

- 4 UARTs.
- 2xCAN
- PWM
- RTC
- Ethernet

- 8 entrades analògiques (ADC) de 12 bits.
- 70 pins de propòsit general (GPIO).
- 4 modes de consum reduït
- Alimentació externa de 3.3 V.

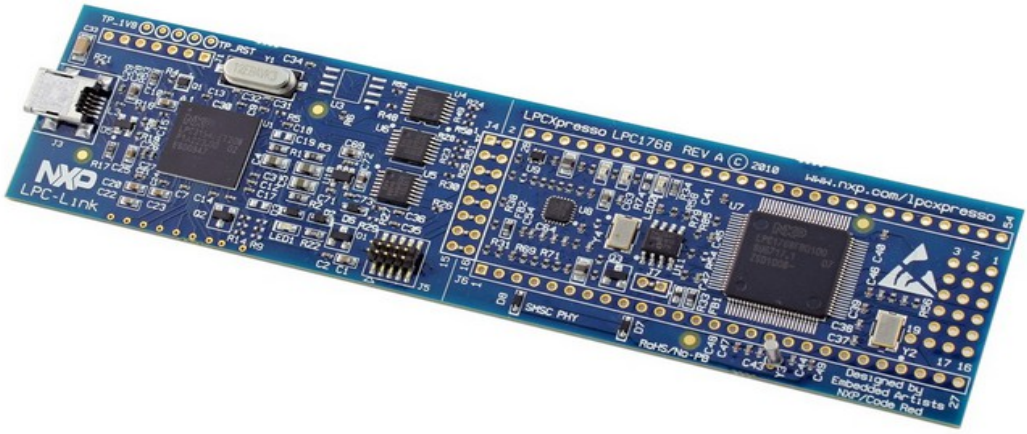


Figura 4: LPC1769

- **Mòdul de comunicacions Wifly RN-XV**

  Es un mòdul *wifi*[2] que permet connectar-nos a xarxes 802.11 b/g. Accepta comandes a través del seu port sèrie, i permet també obrir connexions amb servidors remots. Algunes de les seves característiques són:

  - Compatible amb 802.11 b/g.
  - UART amb taxa de dades de fins a 464Kbps.

2 Podem accedir al seu *datasheet* a l'adreça següent: http://wsn.uoc.edu/xvilajosana/arpa/WiFly-RN-XV-DS.pdf

- 10 GPIOs i 8 ADCs de 14 bits de resolució.
- Configuració via *Wifi* o UART.
- Compatible amb diversos modes d'autenticació *wifi*, com ara WEP, WPA i WPA2-PSK.
- Incorpora funcionalitat TCP/IP i UDP, i aplicacions que la fan servir, com ara DHCP, DNS, ARP, ICMP UDP, Telnet, FTP. També disposa de client HTTP.
- Alimentació de 3.3 V.

Figura 5: Wifly RN-XV

Un cop establerta una connexió amb un port TCP remot, reenvia les dades que rep pel port sèrie cap al servidor remot, i viceversa, de forma transparent per a l'usuari.

- **Adaptador USB-UART CP2102**

Es tracta d'un adaptador pont USB / UART (figura 6) que ens permet connectar un port sèrie de la LPCXpresso a un port USB del nostre ordinador, per poder visualitzar els missatges de *log* que genera el microcontrolador a través d'una aplicació de terminal sèrie. També ens ha permès configurar la *Wifly* a través del seu port sèrie.

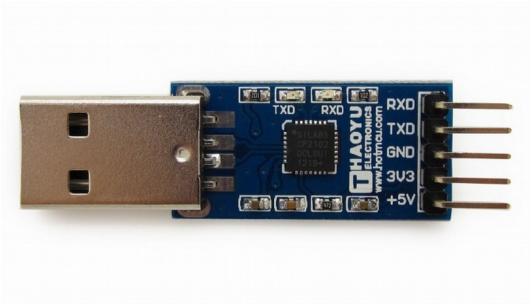


Figura 6: CP2102

- **Acceleròmetre MMA7361**

  És un acceleròmetre analògic de baix consum[3] (figura 7). Les seves característiques principals són:

  - Baix consum: 400 µA en mode normal i 3 µA en mode *sleep*.
  - Alimentació de 2.2 V – 3.6 V.
  - Temps de resposta de 0,5 ms.
  - Alta sensibilitat (800 mV/g en mode 1.5g), i dos rangs (±1,5 g i ±6 g).
  - 3 sortides analògiques que permeten mesurar l'acceleració en cadascun dels eixos.
  - Funcionalitat 0g, que genera un senyal de sortida alt quan el dispositiu es troba en caiguda lliure. La farem servir per a detectar caigudes.

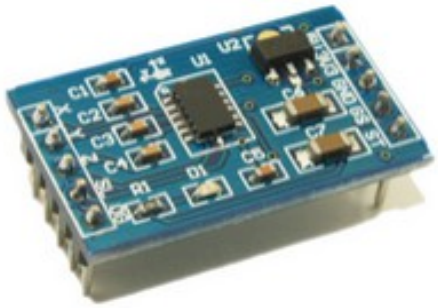


Figura 7: Acceleròmetre MMA7361

A part dels components anteriors, que venien inclosos al *kit*, hem fet servir els següents:

- **Pulse Sensor SEN-11574**

  Es tracta d'un fotopletismògraf, un aparell que genera un senyal analògic de sortida que mesura la quantitat de llum reflectida pels capil·lars perifèrics, per exemple de la punta del dit o del lòbul d'una orella. Aquesta reflexivitat depèn de la densitat de glòbuls vermells a la sang, que depèn a la seva vegada del moment del cicle cardíac en que ens trobem. Poden proporcionar molta informació sobre la condició cardíaca d'una persona, i sobre aspectes

3 A l'adreça següent podem consultar el *datasheet* del dispositu.

derivats, com el nivell d'estrès [5] i molts d'altres. Funciona a partir d'un LED emissor i un fotoreceptor, i disposa de només tres connexions, dues d'alimentació (GND i 3,3 V) i una tercera que és la sortida analògica[4] (veure figura 8).

- **Portàtil / ordinador de sobretaula**

  Per al desenvolupament i depuració del sistema encastat s'ha fet servir un ordinador portàtil amb Ubuntu 14.04. Per a la feina de documentació i per al servidor s'ha fet servir un ordinador de sobretaula amb Microsoft Windows 8.1.

- ***Protoboard***

  Placa de prototipatge (figura 9) que ens permet muntar els diferents components sense necessitat de soldar cables.

- **Material electrònic divers**

  - Cablejat de diferents tipus i alicates per treballar-hi
  - Resistències de diferents valors
  - Multímetre

Figura 8: sensor de batecs

4 Un esquema del dispositiu el tenim a: pulse-sensor.googlecode.com/files/PulseSensorAmpd%20-%20Schematic.pdf

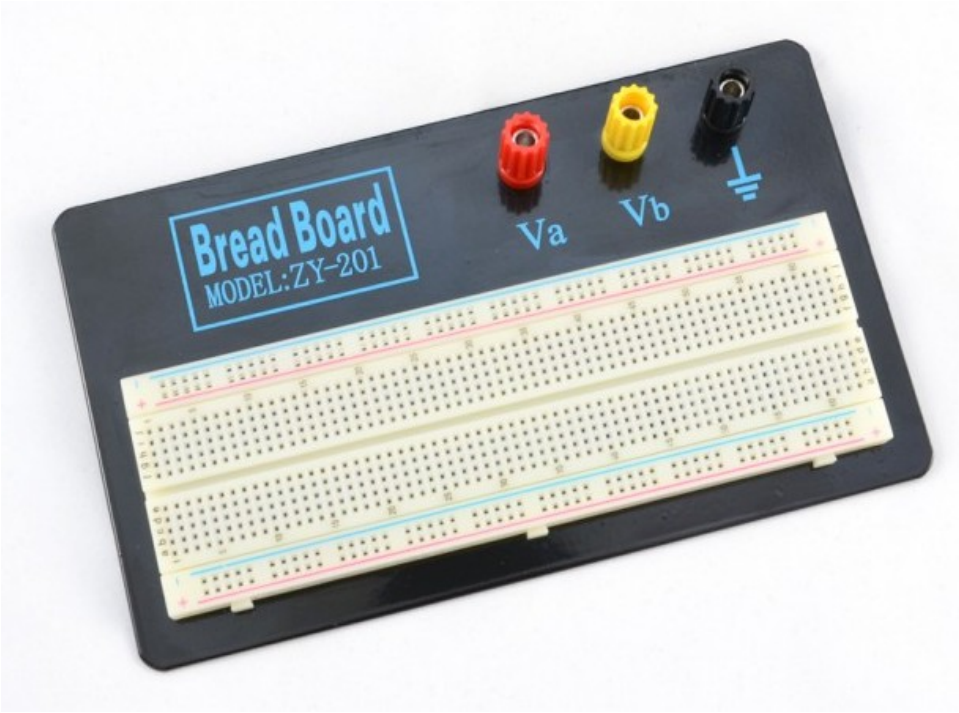


Figura 9: Protoboard

1.7.2 Recursos software

Els principals recursos software que s'han fet servir en el desenvolupament del projecte són:

- **IDE LPCXpresso**

  Entorn de desenvolupament integrat basat en Eclipse. Conté totes les eines necessàries per a desenvolupar aplicacions per a microcontroladors NXP LPC, i també un depurador compatible amb JTAG i SWD.

- **FreeRTOS**

  Sistema operatiu de temps real per a sistemes encastats, que proporciona tota una sèrie de mecanismes per la gestió i planificació de tasques, sincronització de les mateixes, accés concurrent a recursos compartits i intercanvi de missatges entre tasques.

- **Llibreries CMSISv2p00_LPC17xx**

CMSIS és l'acrònim de *Cortex Microcontroller Software Interface Standard*, i són les llibreries que permeten l'accés als diferents components del microcontrolador i perifèrics.

- **Llenguatge de programació C** per al sistema encastat.
- **Llenguatge de programació Java** per al servidor.
- Aplicació de **terminal sèrie CoolTerm**, per visualitzar els missatges de *debug*.
- Programari divers per a la documentació (LibreOffice **Writer**, Microsoft **Project, PowerPoint** i **Visio**).

1.8. Productes finals

Els productes finals que s'han obtingut són dos:

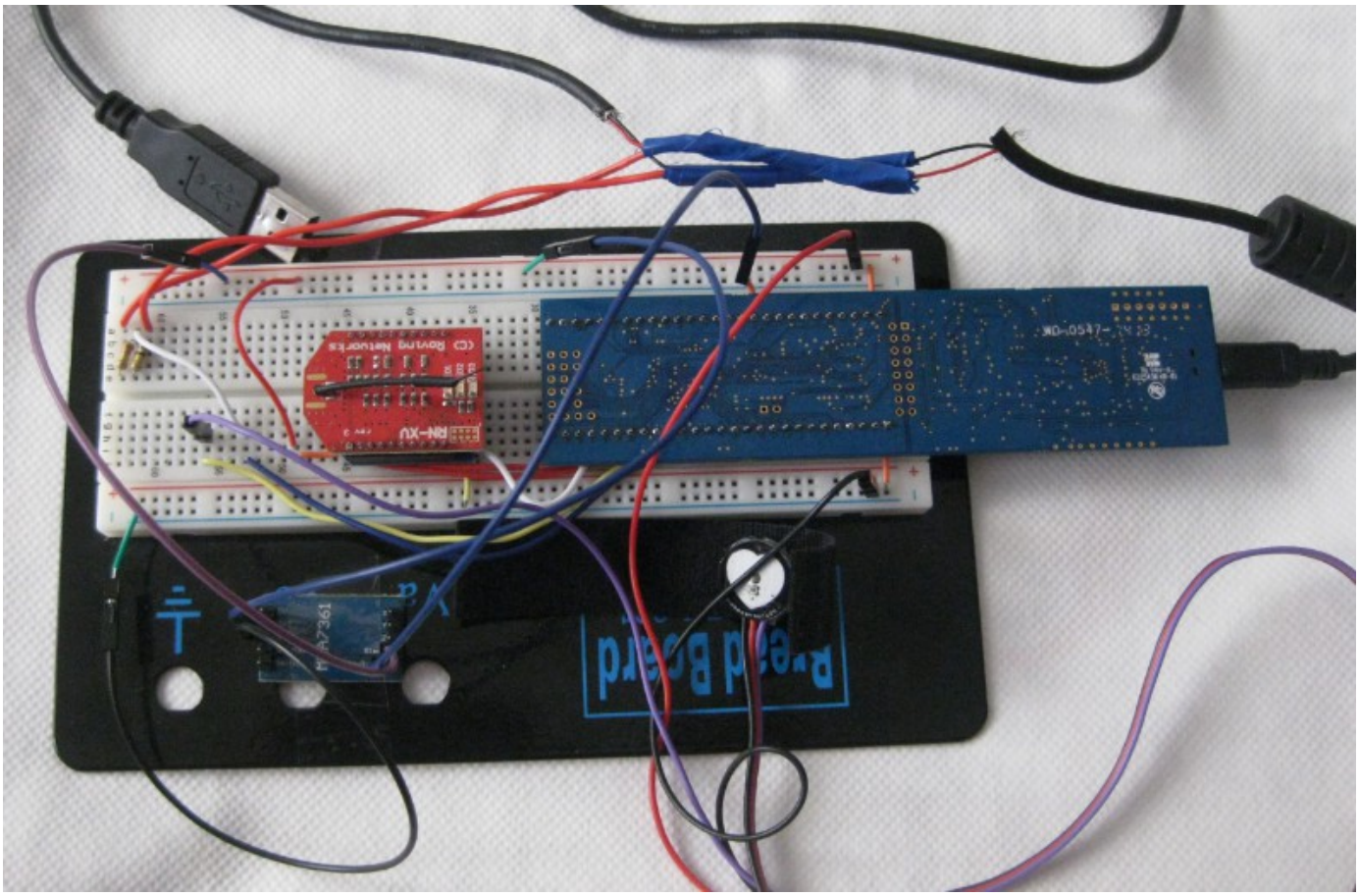


Figura 10: sistema encastat final

- Un sistema encastat de monitorització de pacients com el que mostra la figura 10, format per la LPC1769, el mòdul Wifly i els diferents sensors, muntats sobre la *protoboard*.
- Un servidor remot per a la recepció i visualització de les dades i alarmes generades.

2. Descripció del sistema

2.1. Requisits del sistema i casos d'ús

Els requisits del sistema són els que s'han definit a la secció d'objectius. En qualsevol cas, a la secció següent donarem una descripció més detallada del seu funcionament.

Hi haurà un cas d'ús que involucrarà l'usuari «personal de l'hospital»:

- Llegir avisos del sistema (bateria baixa, taxa de batecs, alarmes caiguda).

  Depenent de les circumstàncies, aquest cas d'ús pot incloure o desencadenar altres interaccions amb el sistema per part de l'usuari:

  - Eliminar de la GUI les dades corresponents a una placa amb la que s'ha perdut la connexió.
  - Resetejar una alarma de caiguda un cop aquesta ha estat tractada.

Més endavant s'explicarà amb més detall el funcionament de cadascuna d'aquestes tasques.

2.2. Descripció del sistema

El sistema està compost de dos grans blocs: el sistema encastat i el servidor, com mostra la figura 11, on podem observar la seva arquitectura.

A continuació descrivim breument la funcionalitat dels dos blocs principals i el seu protocol de comunicació.

2.2.1 Descripció sistema encastat

El sistema encastat consta dels següents elements:

- **Acceleròmetre**: la seva funció és detectar una condició de 0g i enviar un senyal a la placa,

posant un senyal lògic alt a un pin GPIO.

- **Sensor de polsos** (fotopletismògraf): mesura de forma contínua, amb medis òptics, el volum de sang als capil·lars perifèrics, i envia un senyal analògic a la placa a través d'una entrada ADC.
- **Wifly**: permet la comunicació entre la placa i el servidor extern. Es connecta a la placa a través d'una UART.
- Placa **LPC1769**: executa l'aplicació de monitorització.

En quant la placa rebi alimentació, s'iniciarà i executarà de forma seqüencial les següents accions:

- Configurar les UARTs de depuració i wifly.
- Configurar les entrades analògiques per al fotopletismògraf i la bateria.

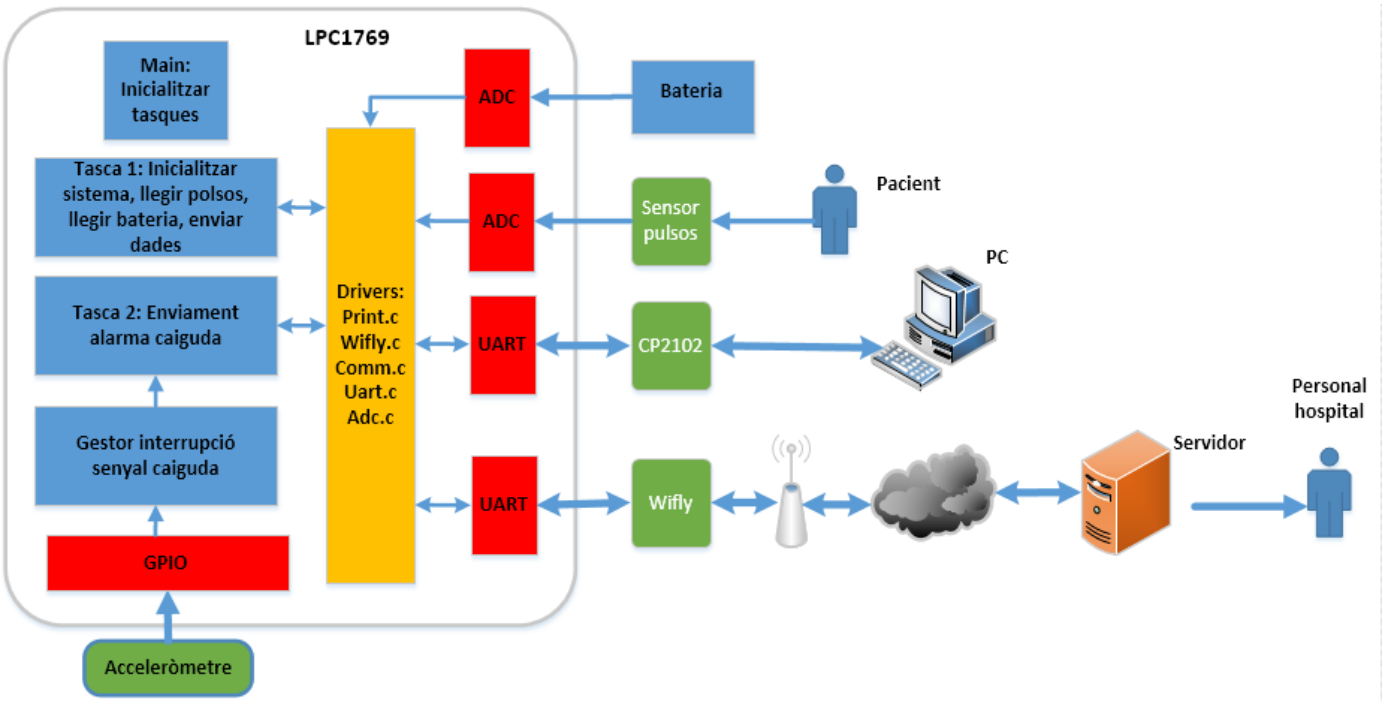


Figura 11: Arquitectura del sistema

- Connectar-se a la xarxa wifi.
- Obrir una connexió amb el servidor remot.

- Enviar un missatge d'identificació de la placa.
- Llegir dades del fotopletismògraf de forma contínua (cada 5 ms). La freqüència de batecs es calcularà per a grups de 10 batecs. En cas que passin 2,5 segons sense que s'hagi produït cap batec (normalment perquè l'usuari s'ha després del sensor), s'enviarà un senyal indicant aquest fet (és a dir, que la taca de batecs és de 0 batecs/minut), i es reiniciarà el recompte.
- Enviar la dada de taxa de batecs al servidor tan bon punt estigui disponible.
- Cada 10000 iteracions del bucle de lectura del fotopletismògraf, es mesurarà el voltatge de la bateria, i s'enviarà un missatge al servidor amb la informació de l'estat de les bateries.
- De forma asíncrona, en cas que l'acceleròmetre detecti una condició de caiguda, el gestor d'interrupció desbloquejarà la tasca d'alarma de caiguda, que enviarà un missatge al servidor i tornarà a quedar bloquejada.

Si la comunicació amb el servidor falli, cal assegurar que qualsevol missatge de caiguda que no s'hagi pogut entregar, sigui reenviat en recupera la connexió. Pels missatges de taxa de batecs, això no és important, perquè aquesta informació queda ràpidament desactualitzada. Tampoc pel que fa a l'estat de bateries, ja que al cap de poca estona es tornarà a mesurar i enviar i no és una informació crítica.

2.2.2. Comunicació amb el servidor

La comunicació entre la placa i el servidor serà bàsicament unidireccional, amb la placa iniciant tots els missatges. El servidor, però, respondrà a cada missatge amb un «OK», per tal que l'aplicació pugui determinar si la connexió resta oberta, i en cas contrari, reiniciar-la. Els diferents tipus de missatges són:

- **Alarma de caiguda (taula 2)**
- **Taxa de batecs (taula 3)**
- **Identificació de la placa (taula 4)**
- **Nivell de bateria (taula 5)**
- **Missatge rebut (taula 6)**

Format	Origen	Destinació	Descripció
«10»	Placa	Servidor	Informa que s'ha produït una caiguda.
Camp	**Descripció**		
1er (10)	Codi del missatge		

Taula 2: Missatge d'alarma de caiguda

Format	Origen	Destinació	Descripció
«20:N»	Placa	Servidor	Informa de la darrera taxa de batecs calculada
Camp	**Descripció**		
1er (20)	Codi del missatge.		
2on (N)	Taxa de batecs/min del darrer període.		

Taula 3: missatge de taxa de batecs

Format	Origen	Destinació	Descripció
«30:N»	Placa.	Servidor.	Identifica la placa que farà servir la connexió.
Camp	**Descripció**		
1er (30)	Codi del missatge.		
2on (N)	Identificador de la placa.		

Taula 4: missatge d'identificació de la placa

Format	Origen	Destinació	Descripció
«40:N»	Placa	Servidor	Informa del nivell de les bateries, en base al voltatge de sortida de la mateixa.
Camp	**Descripció**		
1er (40)	Codi del missatge		
2on (N)	Nivell de la bateria: • 0 baix • 1 normal		

Taula 5: Missatge de nivell de les bateries

Format	**Origen**	**Destinació**	**Descripció**
OK	Servidor	Placa	Informa que s'ha rebut el missatge enviat per la placa.
Camp	**Descripció**		
1er (OK)	Codi del missatge		

Taula 6: Missatge de confirmació

Només cal identificar la placa origen de la comunicació una vegada, amb el missatge d'identificació de placa (taula 6), ja que cada port de connexió amb el servidor el farà servir només una placa. Per tant, tornar a identificar-se en missatges subseqüents seria redundant.

Les freqüències aproximades a les quals s'intercanvien les podem veure a la taula 7. Podem deduir que el cas en que s'enviarien menys missatges és amb un pacient amb el sensor ben connectat i amb una baixa freqüència de polsos, i tot i així, l'interval màxim entre missatge seria d'uns 12-15 segons.

Tipus de missatge	**Freqüència estimada**
10	S'enviarà només quan es produeixi una caiguda.
20	Taxa de batecs: • en condicions normals: ◦ 1 missatge cada 12 segons aprox.(50 batecs / min). ◦ 1 missatge cada 4 segons aprox. (160 batecs / min). • si el sensor no detecta cap batec en un interval de 2,5 segons: ◦ 1 missatge cada 2,5 segons, indicant taxa 0 batecs / segon.
30	Cada vegada que s'inicií una nova connexió.
40	Cada 10000 iteracions del bucle principal de la tasca de mesura del fotopletismògraf. Com que es fa una mesura cada 5 ms: • cada 50 segons aproximadament.

Taula 7: Freqüències estimades dels missatges

2.2.3. Descripció del servidor

El servidor consta de dues parts:

- El servidor pròpiament dit, que rep les connexions de les plaques, i els seus missatges. Per a cada connexió, és crearà un fil d'execució per tractar les dades de la mateixa.
- Una GUI, per mostrar les dades enviades per cada placa:
  - cada placa amb una connexió activa mostrarà les seves dades en una zona de la GUI. Es

mostraran:

- taxa de batecs.
- senyal d'alarma, si s'ha produït una.
- indicació de bateria baixa, si és el cas.
- un botó per eliminar el senyal d'alarma. Només estarà actiu si s'està mostrant un missatge d'alarma.
- un botó per eliminar les dades d'una placa sense connexió. Només estarà actiu si el servidor considera que s'ha perdut la connexió amb la placa.

- com hem vist abans, una placa connectada hauria d'enviar missatges a un ritme no inferior a un cada 12-15 segons. En conseqüència, i sent una mica conservadors, el servidor considerarà que s'ha perdut la connexió si han passat més de 30 segons sense rebre cap missatge de la placa. Les dades de la placa no s'esborraran de la pantalla, ja que la seva informació pot continuar sent rellevant, per exemple si indica una caiguda que encara no ha estat advertida. Serà l'usuari qui decidirà quan eliminar aquesta informació, polsant el botó corresponent.

3. Disseny del sistema

3.1. Disseny hardware

En els apartats següents es descriuen les interconnexions entre els diferents elements hardware del sistema.

3.1.1. LPC - Wifly

Les connexions entre la LPC i la Wifly són les que es mostren a la figura 12. N'hi ha tres:

- Una connexió d'alimentació de la Wifly, mitjançant els pins GND i VOUT de la LPC1769.
- Una connexió entre la UART0 de la LPC1769 i la UART de la Wifly. La connexió entre el TX i el RX de les UARTs ha de ser creuada.
- Una connexió entre un pin GPIO de sortida de la LPC1769 (s'ha escollit el P0.22) i el pin de

RESET de la Wifly. L'aplicació d'un pols de sortida pel pin GPIO d'almenys 160 microsegons de duració permetrà resetejar la Wifly.

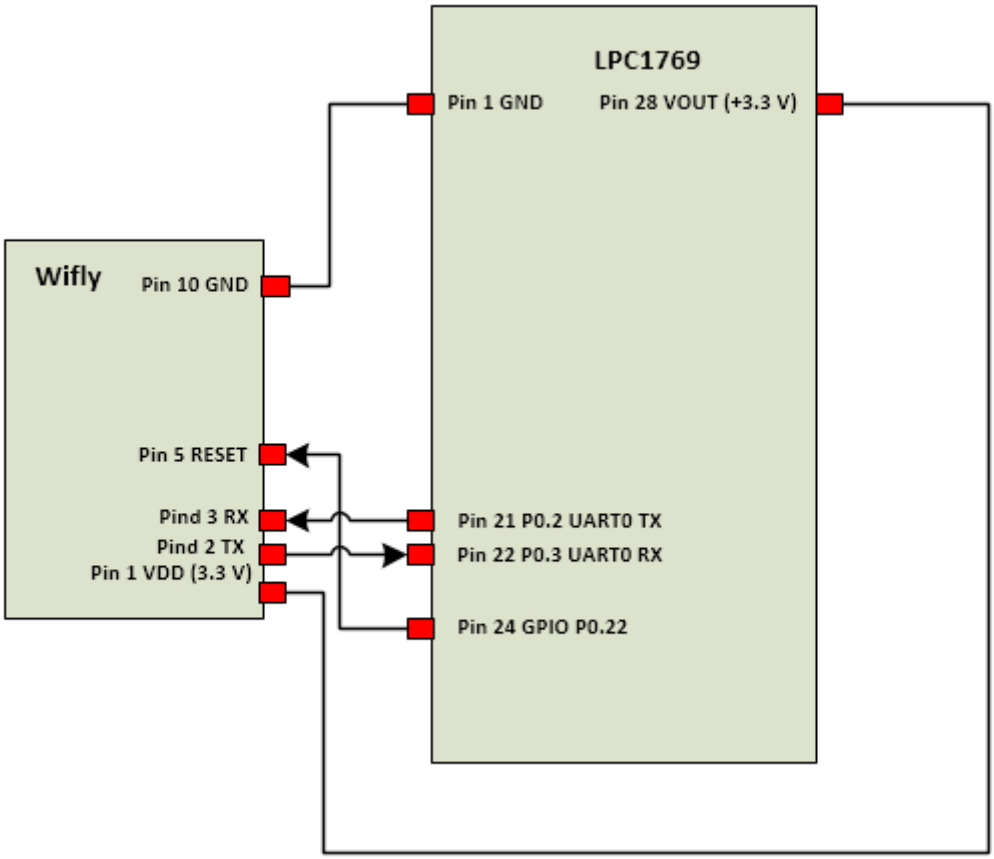


Figura 12: Connexions LPC1769-Wifly

3.1.2. LPC – MMA7361L (acceleròmetre)

A la figura 13 podem veure les connexions entre la LPC i l'acceleròmetre, que són:

- Una connexió d'alimentació de l'acceleròmetre, que es fa connectant-lo als pins GND i VOUT de la LPC1769, a través dels carrils d'alimentació de la *protoboard*.
- El pin Sleep permet posar l'acceleròmetre en mode *sleep* si el senyal està baix. Per tal que el xip operi en mode normal, connectarem el pin a un senyal lògic alt (3.3 V). Tant aquesta connexió com les d'alimentació es fan a través dels carrils laterals de la *protoboard*.
- El pin 0g donarà una sortida alta si l'acceleròmetre detecta que està en caiguda lliure. Volem detectar aquesta transició mitjançant un pin GPIO d'entrada (s'ha escollit el P0.4).

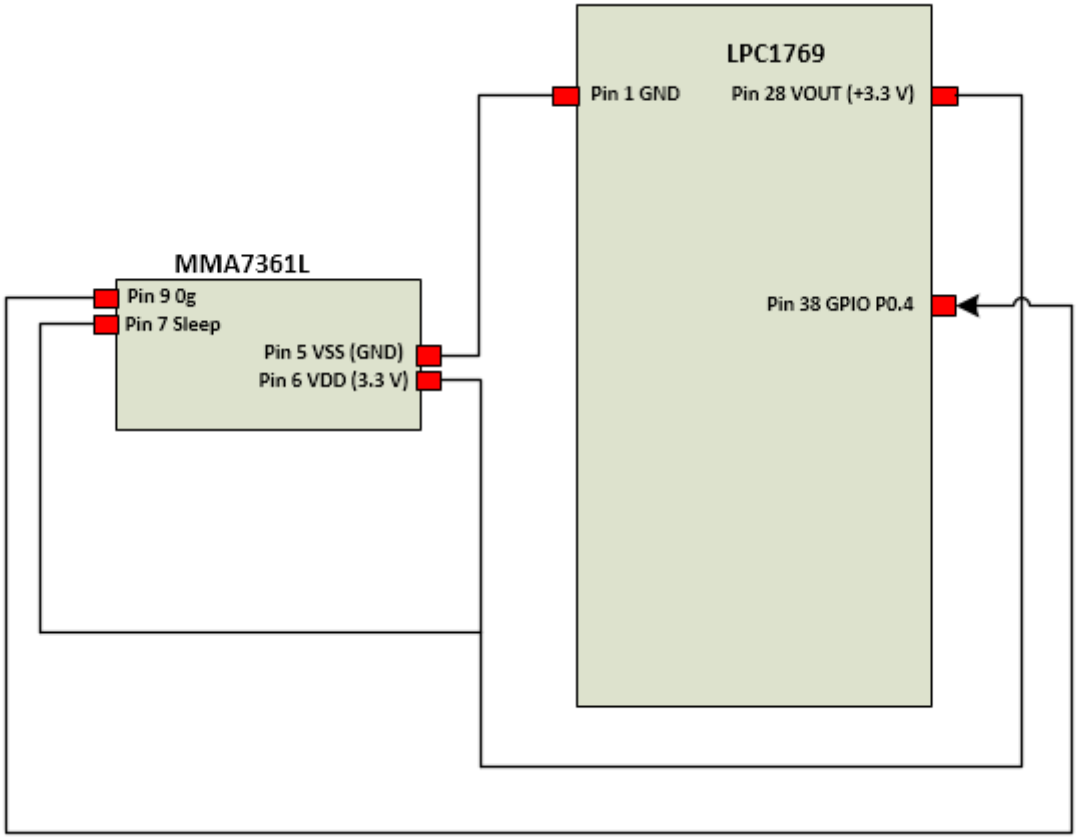


Figura 13: Connexions LPC1769-Acceleròmetre

3.1.3. LPC – CP2012

La figura 14 mostra les connexions entre la LPC i el mòdul UART-USB CP2012. Són:

- Una connexió de terra comuna entre els dos dispositius, per tal que els senyals a transmetre tinguin un punt de referència compartit. Aquesta connexió es fa a través del carril lateral corresponent de la *protoboard*.
- Una connexió entre la UART3 de la LPC1769 i la UART del mòdul. Com sempre, han d'estar connectats els pins TX d'una UART amb els RX de l'altra.

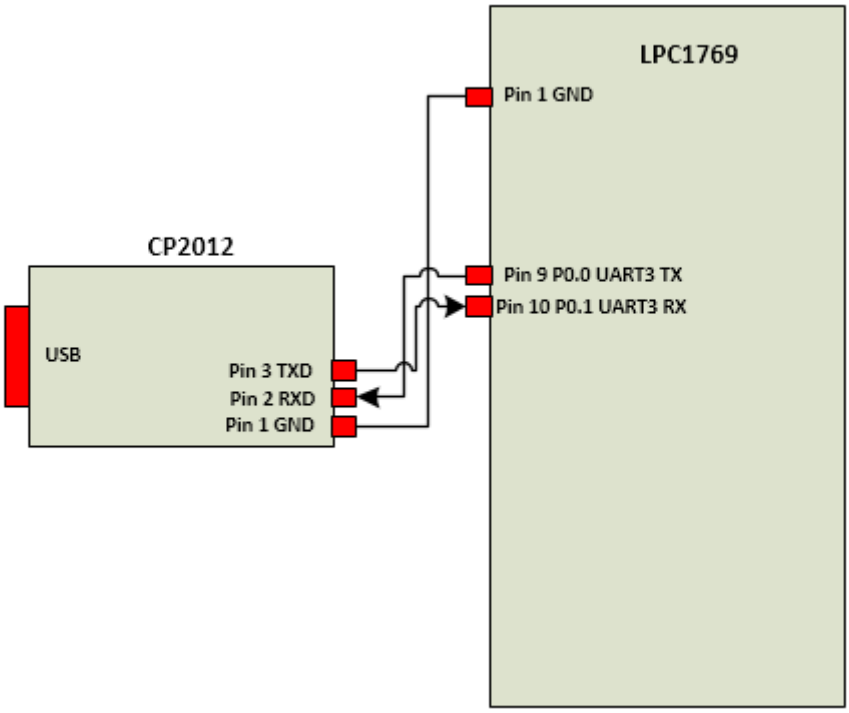


Figura 14: Connexions LPC1769-CP2102

3.1.4. LPC – Pulse Sensor SEN-11574

Les connexions entre la LPC i el sensor de polsos són les següents (figura 15):

- Una connexió d'alimentació pel sensor, que es fa connectant-lo als pins GND i VOUT de la LPC1769, a través dels carrils d'alimentació de la *protoboard*.
- Una connexió entre la sortida analògica del sensor i una de les entrades analògiques de la LPC1769. En aquest cas, s'ha escollit el quart canal de l'ADC (AD0.4).

3.1.5. LPC – Sensor nivell bateria

El nivell de càrrega de la bateria l'estimarem a partir de la tensió de la mateixa. Com la tensió de la bateria és de 5 V, i més de 3,3 V saturaria les entrades analògiques de la LPC1769, haurem de rebaixar aquesta tensió mitjançant un divisor de tensió. La figura 16 mostra l'esquema.

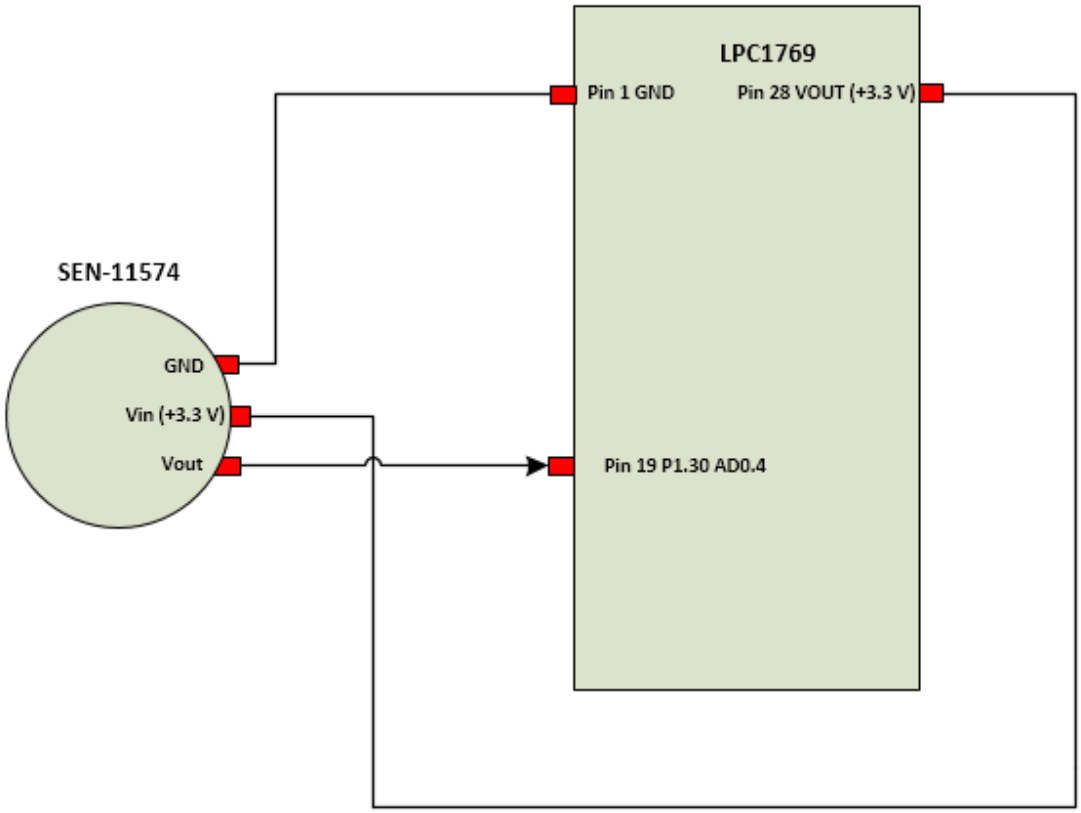


Figura 15: Connexions LPC1769-Sensor de batecs

Veiem que no estan representades les connexions d'alimentació entre la bateria o font d'alimentació i la placa, ja que hem alimentat la placa via USB. Per tal de fer el sensor de bateria hem fet una derivació del cable USB d'alimentació cap a l'entrada analògica AD0.1.

Pel que fa les resistències, fem servir valors grans per reduir la intensitat que hi circula i, per tant, el consum. Els valors són:

- R_1 – 330 KΩ
- R_2 – 100 KΩ

La tensió al punt de mesura del divisor de tensió, quan la bateria està totalment carregada (o alimentem el sistema a través d'un port USB d'un ordinador) serà, aplicant la llei d'Ohm: $V_d = 5 * R_2 / (R_1 + R_2)$. És a dir, **$V_d = 1{,}16$ V**

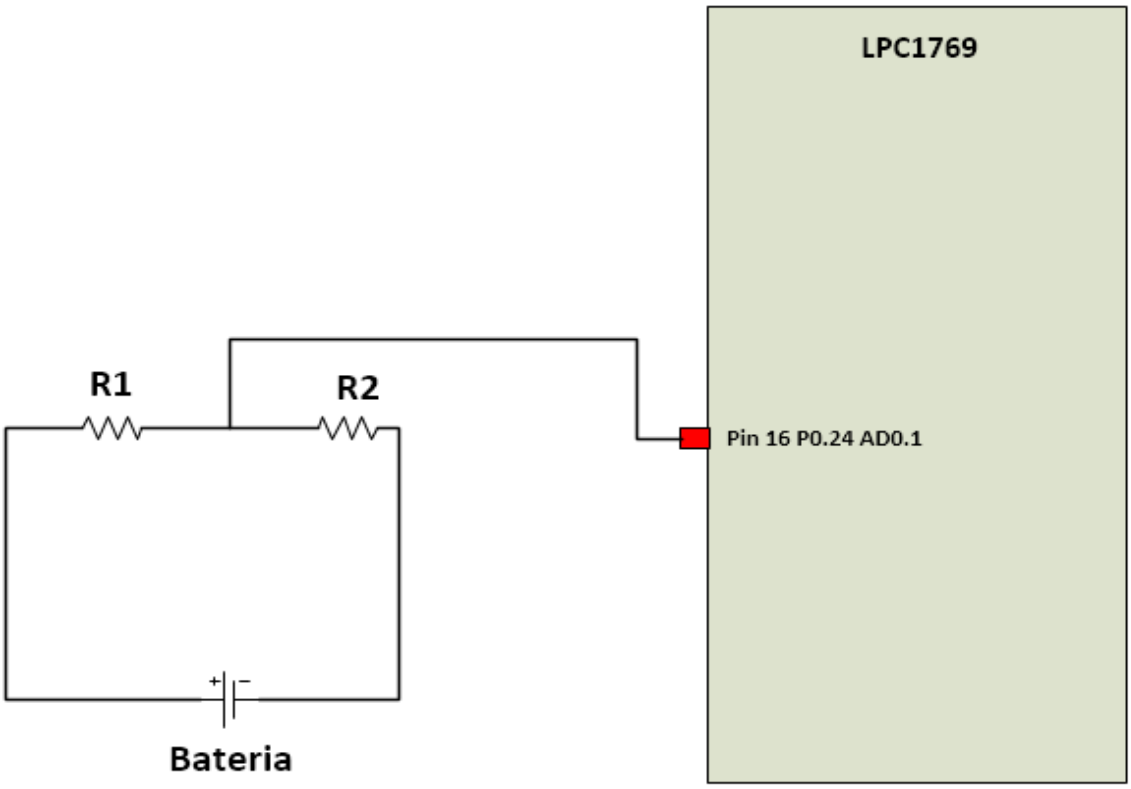


Figura 16: Connexions LPC1769-Sensor voltatge bateria

3.2. Disseny de l'aplicació

La figura 17 mostra els diferents mòduls de que consta l'aplicació. Hi ha un mòdul principal, format per les tasques de l'aplicació principal i el gestor d'interrupcions, i una sèrie de llibreries o *drivers* que controlen l'accés als diferents perifèrics.

La descripció detallada de cadascun dels mòduls es farà més endavant. Pel que fa a les tasques de l'aplicació principal, són:

- **vTaskHeartRate**: s'encarrega d'inicialitzar la placa, detectar polsos, mesurar el voltatge de la bateria, i enviar els missatges corresponents al servidor.
- **vTaskFall**: s'encarrega d'enviar un missatge d'alarma de caiguda al servidor.

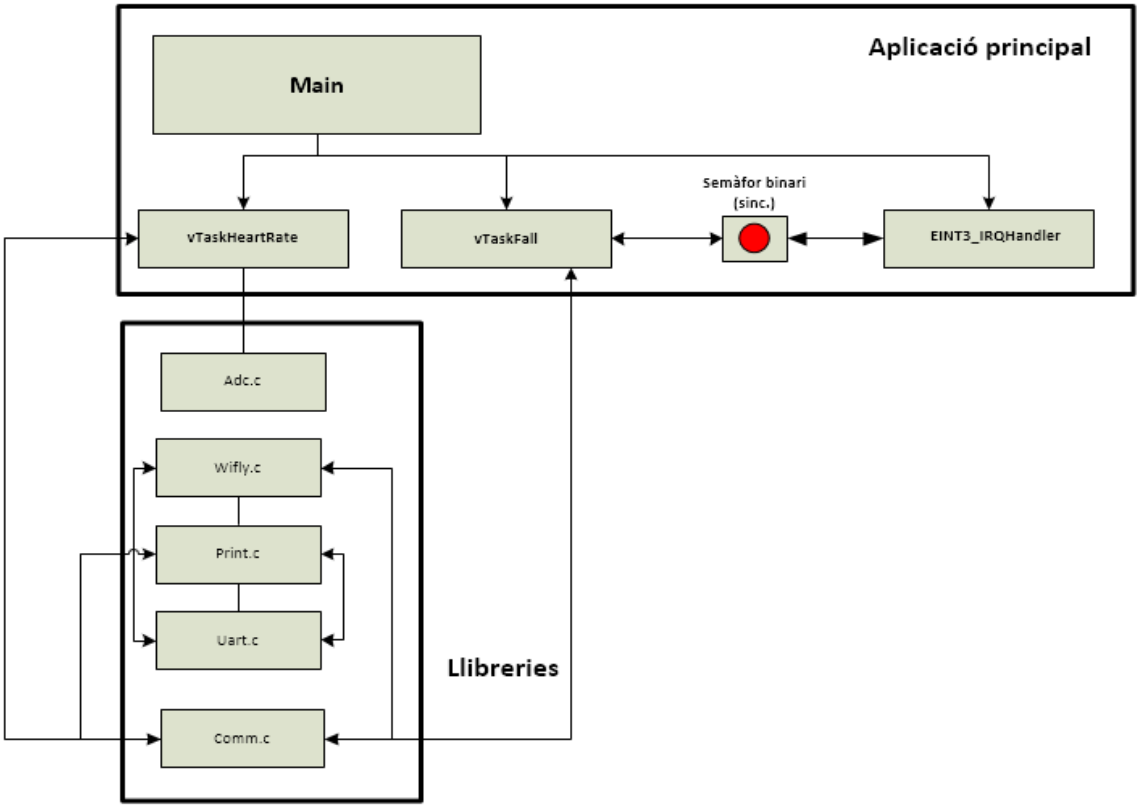


Figura 17: Mòduls del sistema

3.3. Disseny del servidor

El diagrama de classes del servidor el podem a la figura 18. Les classes de que consta són les següents:

- **Server**: classe principal de l'aplicació, on està el mètode *main()*. Conté també el *socket* del servidor, que accepta les connexions per part dels dispositius de monitorització de pacients.
- **ConBoard**: classe interna de Server. Hi ha una per cada connexió que s'estableix amb un dispositiu de monitorització. Hereta de la classe *Thread*, i per tant, els objectes d'aquesta classe s'executen concurrentment. Conté el *socket* de comunicació amb el client i les dades d'estat de la comunicació (si la connexió és vàlida, si es pot eliminar...).
- **BoardPanel**: és un panell amb els diferents elements de visualització de dades i control per part de l'usuari. Hi ha un per cada placa connectada.
- **ServerGUI**: és l'entorn gràfic del servidor. És una finestra que agrega els diferents panells

de cadascuna de les plaques.

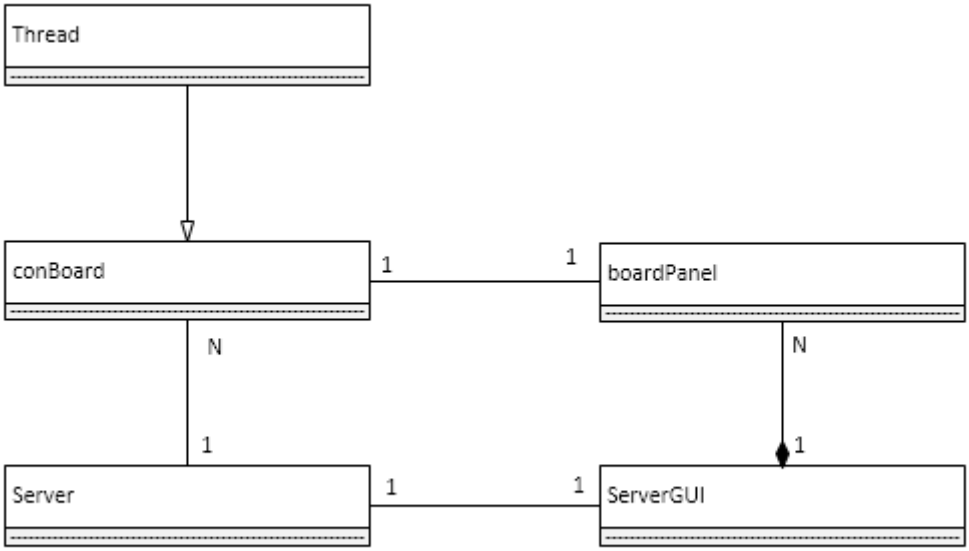


Figura 18: Diagrama de classes del servidor

3.4. Disseny de la interfície d'usuari

A la figura 19 podem veure un esquema de la interfície d'usuari.

Hi ha un panell per a cada una de les plaques que estan o han estat connectades al servidor i que l'usuari no ha eliminat explícitament:

- Un sensor 1, que en aquest moment mostra una taxa de batecs de 62 i una alarma de caiguda. El botó «reset» s'ha habilitat per tal que l'usuari elimini manualment l'alarma quan aquesta ja ha estat tractada. Aquest botó roman desactivat si no hi ha cap alarma.
- Un sensor 2, sense connexió amb el servidor. En el moment que la va perdre, mostrava una taxa de batecs de 0, probablement perquè l'usuari no portava el sensor acoblat. Es mostra que la bateria estava baixa (de fet, podria ser que l'hagués perdut perquè s'hagués acabat la bateria). El botó «eliminar» està habilitat, per tal que l'usuari esborri el panell si ho considera oportú. Aquest botó estarà deshabilitat si la connexió roman activa.

Figura 19: Interfície d'usuari

Cada cop que una placa es connecta al servidor es crea un nou panell per representar les seves dades. Si la placa ja té un panell visible d'una connexió anterior, simplement s'elimina el panell antic (mantenint si s'escau el missatge d'alarma de caiguda) i se substitueix pel nou panell.

4. Disseny funcional del sistema

En aquest apartat mostrarem el diagrama de flux de l'aplicació principal, a diversos nivells de detall, i del servidor. També donarem explicacions sobre els aspectes de la implementació de l'aplicació que ho requereixin.

4.1 Aplicació principal

A la figura 20 podem veure, a un primer nivell, el diagrama de flux de l'aplicació principal. Es creen dues tasques, fent servir els serveis que ens proporciona FreeRTOS:

- *vTaskHeartRate*: inicialitza el hardware (UARTs, GPIO i ADC), recull les dades del fotopletismògraf i de la bateria i envia els missatges corresponents al servidor.
- *vTaskFall*: envia una alarma de caiguda quan es detecta un senyal de 0g de l'acceleròmetre.

També s'habilitarà el *handler* d'interrupcions EINT3, que s'ocupa de les interrupcions als GPIO.

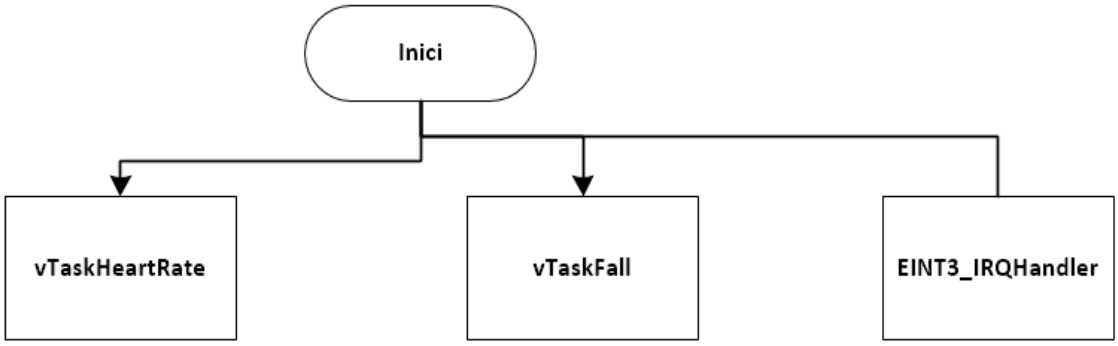


Figura 20: Tasques de l'aplicació principal

Finalment, s'invocarà el planificador de FreeRTOS per tal que s'executin les tasques de forma concurrent, segons les seves prioritats.

Com que el tractament i enviament del senyal d'alarma de caiguda és prioritari, s'ha programat per a la tasca *vTaskFall* una prioritat superior (3) a la de *vTaskHeartRate* (1). En cas que *vTaskFall* no es bloquegés mai, això faria que *vTaskHeartRate* no s'executés. A la pràctica, però, *vTaskFall* restarà bloquejada la major part del temps, excepte quan es produeixi una interrupció per part de l'acceleròmetre. En aquell moment, es desbloquejarà, enviarà l'alarma i es tornarà a bloquejar.

La recollida de dades de la bateria, i l'enviament de dades, es podrien haver implementat també en tasques diferents. S'ha considerat que això complicava el disseny de l'aplicació i era innecessari, ja que no hi ha requisits temporals de planificació massa estrictes pel que fa a la recollida de dades de

batecs, de bateria i d'enviament de missatges. Aquestes funcions es poden fer sense cap problema de forma seqüencial (com es pot veure als diagrames de flux) i, per tant, en una sola tasca.

4.1.1. Diagrama de flux vTaskHeartRate

La figura 21 mostra el diagrama de flux de *vTaskHeartRate*. Les tasques es fan de forma seqüencial, fin a arribar a «Comptar taxa de batecs o bateria», que en realitat és un bucle que s'executa de forma contínua, com es veurà més endavant. Els seus passos són:

4.1.1.1. Inicialització UARTS

Hem fet servir dues de les quatre UARTs de que disposa la LPC1769, una per a la Wifly i l'altra pel CP2102, com hem vist anteriorment. Les funcions d'inicialització de les dues UARTs es troben a les llibreries *uart.c*, *wifly.c* i *print.c*, que ja venien donades i que comentaré breument en un apartat posterior.

4.1.1.2. Connectar el servidor

Podem veure l'esquema d'aquesta subtasca a la figura 22. Els seus passos són

Resetejar wifly

Connectar amb la xarxa wifi

Un cop inicialitzada la Wifly, s'efectua la connexió amb el punt d'accés wifi. L'encarregat de fer-ho és la funció *joinSSID()* del mòdul *comm.c*, que a la seva vegada crida a *Wifly_JoinSSID()*, del mòdul *wifly.c*.

Obrir connexió amb el servidor

Podem veure l'esquema d'aquesta subtasca a la figura 22. Es crida la funció *openConn()*, de *comm.c* que executa *Wifly_Open()*, del mòdul *wifly.c*. Aquesta és l'encarregada d'obrir una connexió TCP amb el servidor remot. En cas que la connexió no s'estableixi de forma satisfactòria, s'intentarà fins a 3 cops. Si encara no s'ha aconseguit, es torna a iniciar el procés des del punt on es reseteja la Wifly. Tot el procés es repeteix fins a establir connexió.

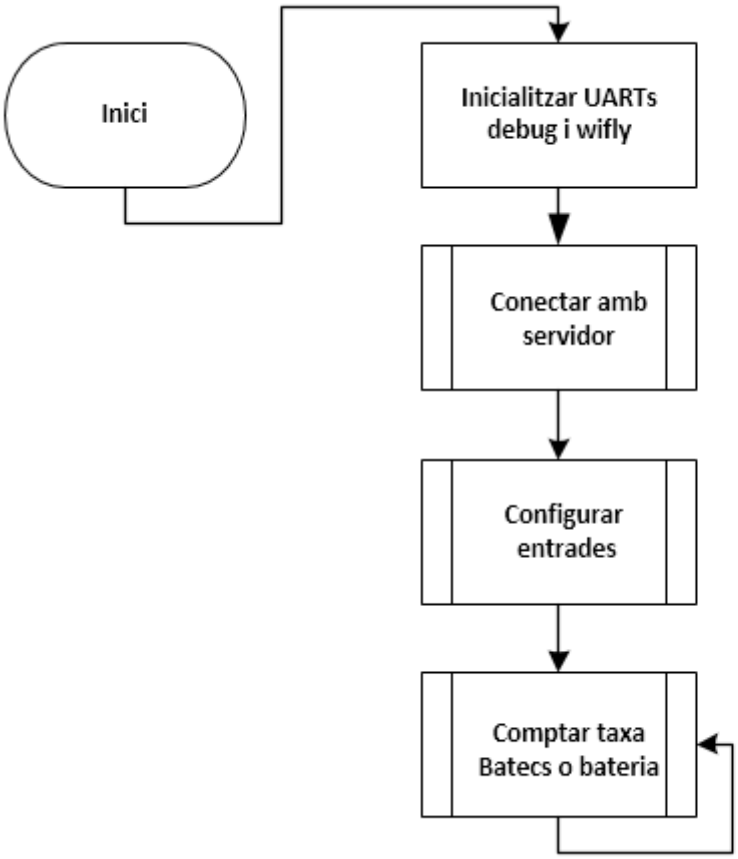


Figura 21: Diagrama de flux de vTaskHeartRate

4.1.1.3. Configurar entrades

La subtasca «configurar Entrades» consta de tres passos estrictament seqüencials:

- configurar el conversor analògic / digital.
- configurar el GPIO d'entrada.
- configurar la interrupció del GPIO.

El conversor analògic digital transforma els voltatges analògics de les seves entrades, en el rang de 0 a 3,3 V, a senyals digitals de 12 bits. Cada interval d'un bit representa per tant un increment del voltatge de 3.3 V / 4096 = 0,0008 V aproximadament.

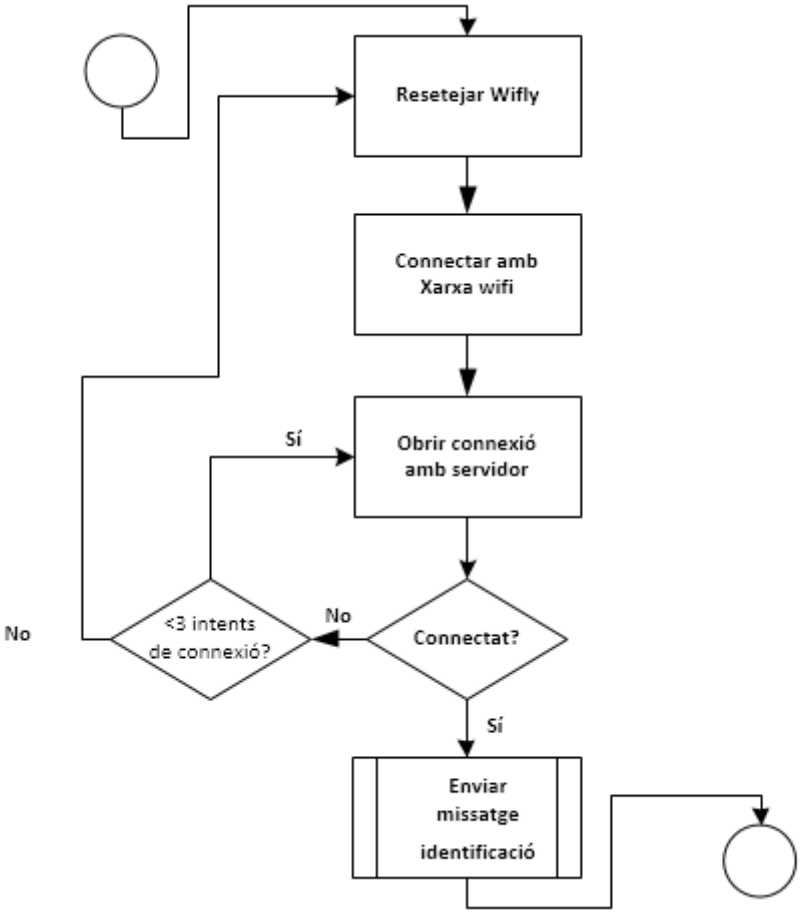


Figura 22: Connectar amb el servidor

Els passos per a configurar l'entrada de l'ADC són els següents:

- Assignar al pin seleccionat la funció d'entrada analògica. En general, un pin determinat pot realitzar vàries funcions, depenent del valor dels bits que configuren el pin al registre PINSEL adequat. Com podem veure a la figura següent del manual de la placa, les configuracions necessàries pel nostre cas són:
  - per fer funcionar P0.24 com a AD0.1, els pins 117:16 de PINSEL1 han de prendre el valor 01.
  - per fer funcionar P1.30 com a AD0.4, s'han de posar els pins 29:28 de PINSEL3 a 11.
- A continuació, caldrà activar la funció ADC activant el bit 12 del registre PCONP, que és

l'encarregat d'habilitar i deshabilitar diverses funcions (timers, PWM, ADC...). El valor d'aquest bit s'haurà de posar a 1.

- Seguidament, s'escull l'entrada de rellotge que farem servir. Escollim un divisor de 8 (PCLK_peripheral / 8), posant a 11 els bits 24 i 25 de PCLKSEL1.
- Finalment, cal configurar alguns aspectes al registre ADCR:
  - magnitud per la qual dividim el senyal del rellotge al ritme del qual treballa el conversor. Li assignarem un 1, amb el que el divisor serà igual a 2. Ve controlar pels bits 15:8 de ADCR.
  - activar la conversió ADC posant a 1 el bit 21 de ADCR.
  - indicar els pins que es llegiran. Es configuren a partir dels bits 0 a 7 de ADCR. Com que volem activar el 4 i el 1, posarem els bit 4 i 1 a 1.

Configurar GPIO entrada

El pin P0.4 l'hem de configurar com a GPIO d'entrada, per tal de poder llegir el senyal de 0g de l'acceleròmetre. Els passos a realitzar són:

- configurar el pin P0.4 com a entrada GPIO. El mode funcionament d'aquest pin ve donat pel valor dels pins 9:8 del registre PINSEL0. Haurem d'assignar un valor de 00.
- activar els GPIO. Es fa posant a 1 el bit 15 de PCONP.
- indicar que el pin funcionarà com a entrada de dades. La direcció del pin Px.N es configura mitjançant el bit N del registre FIOxDIR. Si volem configurar-lo com a entrada, haurem de posar el valor a 0. Per tant, el bit 4 de FIO0DIR haurà de posar-se a 0.

4.1.1.4. Comptar taxa batecs i nivell de bateria

És pot considerar la part més important del sistema, on es prenen i es processen les dades. El sistema compta batecs executant un bucle continu en que va mesurant el senyal analògic del fotopletismògraf. Mentre executa aquest bucle, va oscil·lant entre dos estats, i es comportarà de forma diferents segons si estigui en un o en l'altre. La figura 23 mostra el diagrama d'estats i quins esdeveniments provoquen la transició d'un a l'altre.

El sentit d'aquests dos estats s'explica amb més profunditat a l'apartat **Implementació de**

l'algoritme comprador de batecs, en aquesta mateixa secció. La idea és que, un cop hem detectat un pic de durada suficient, i per tant un batec, durant 250 ms ignorarem els senyals del sensor per tal d'assegurar-nos que quan tornem a intentar detectar un batec el pic ja ha passat, i d'aquesta forma no comptem més d'un batec per cada pic.

Hem de tenir també en compte a més que la detecció del senyal de nivell de voltatge de la bateria s'efectua en el mateix bucle, però amb una freqüència molt menor (només una de cada 10000 iteracions del bucle).

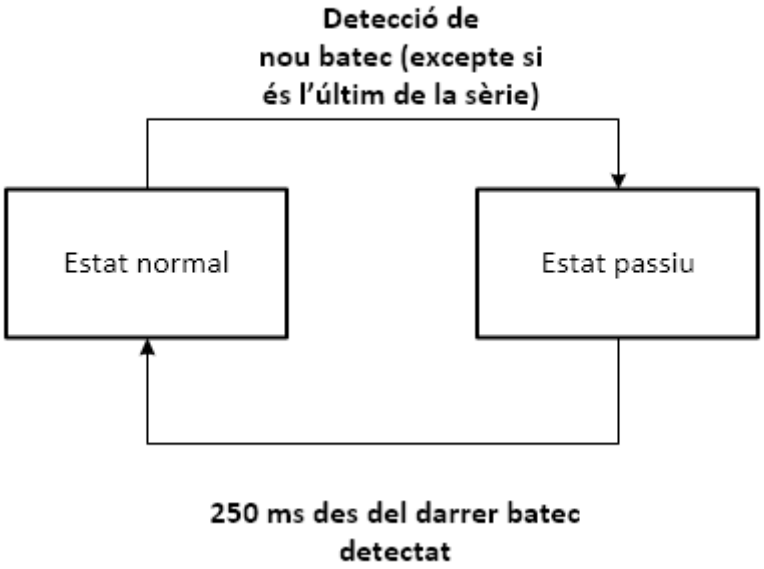


Figura 23: Estats del sistema durant l'adquisició i tractament de les dades

El diagrama de flux del sistema té dues versions, depenent de en quin dels dos estats es troba el sistema:.

- **Estat normal (figura 24)**
- **Estat passiu (figura 25)**

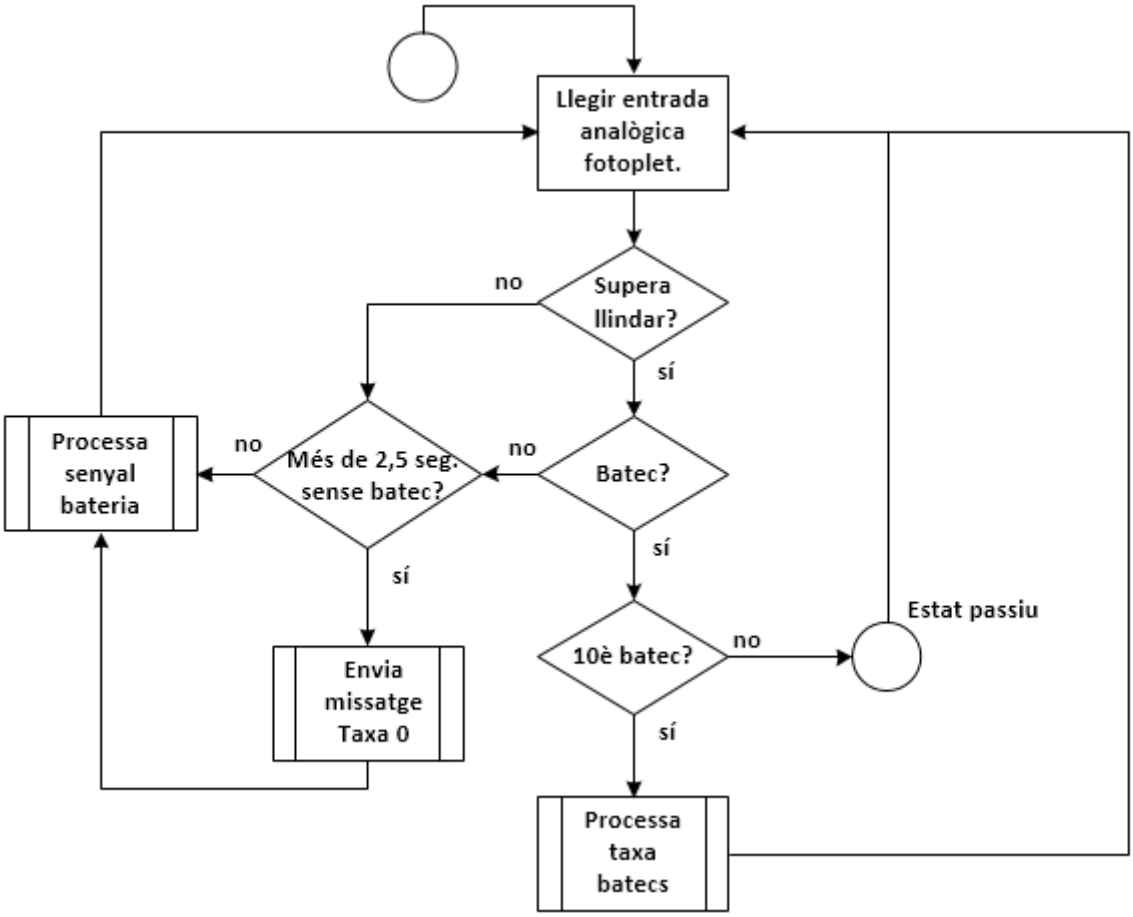


Figura 24: Comptador batecs i nivell de bateria - Estat normal

Podem veure que hi ha dos subprocessos per desenvolupar als diagrames anteriors. Són:

- **processar senyal bateria (figura 26)**.
- **processar taxa de batecs (figura 27).**
- **enviar missatge**

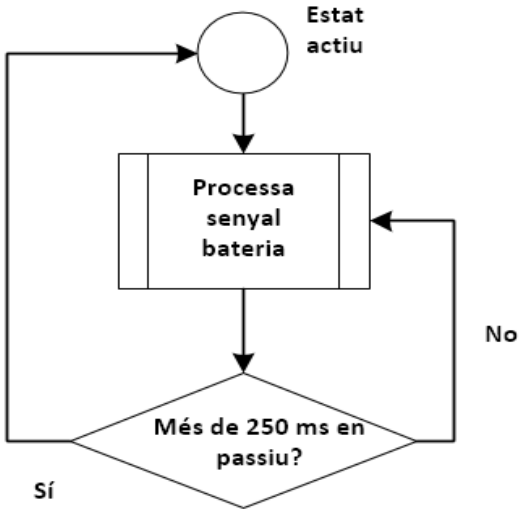


Figura 25: Comptador batecs i nivell de bateria - Estat passiu

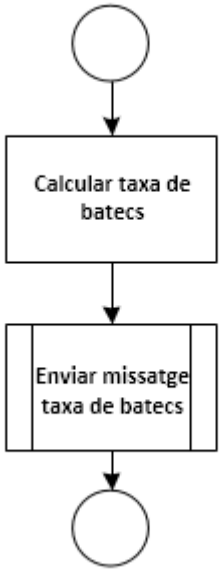


Figura 26: Processar taxa de batecs

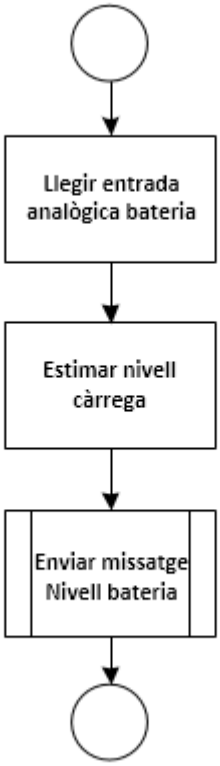


Figura 2: Processar senyal bateria

El diagrama d'enviar missatge el veurem en un apartat posterior. A continuació, a l'apartat **Implementació algoritme comprador de batecs** explicarem pas a pas aquests diagrames.

Implementació algoritme comprador de batecs

Per entendre el funcionament de l'algoritme comptador de batecs, el primer que haurem de tenir en compte és quin és el patró a esperar del senyal d'entrada del fotopletismògraf [9] (figura 28).

És un patró regular amb un pic principal i un pic secundari, menor, per cada batec. El patró real que hem trobat en el nostre cas és molt similar, tot i que apareixen alguns pics anòmals, molts estrets, la causa dels quals no està clara.

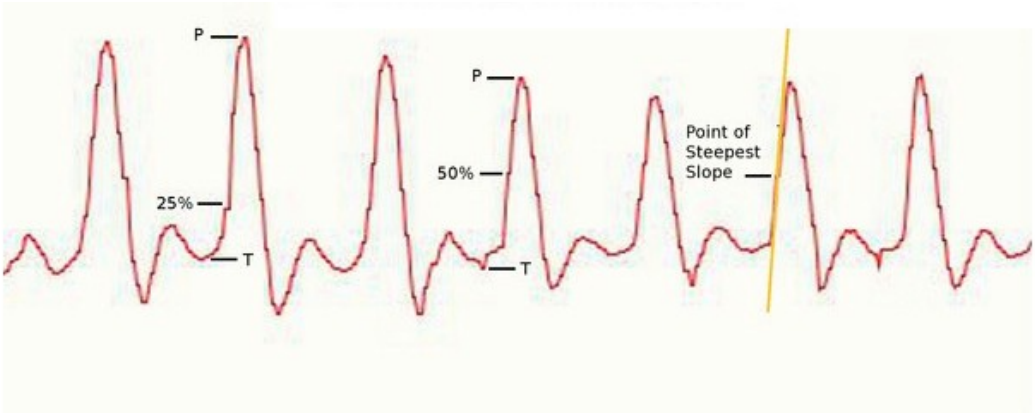


Figura 28: Patró de sortida esperable

A la figura 29 podem observar una sèrie de mesures, efectuades a intervals de 5 ms:

- hi ha 11 batecs clarament delimitats.
- hi ha tres pics anòmals (dos dels quals estan molt junts). Els tres es produeixen en el mateix punt del cicle cardíac, en el màxim secundari que apareix després del batec, anomenat tècnicament elevació dicròtica (*dicrotic notch*). En altres gràfiques he confirmat que una gran part dels pics anòmals se situa en aquestes elevacions dicròtiques, però no tots.
- podem apreciar la presencia de soroll, especialment visible a les parts de la gràfica que no són el pic principal. Tot i això, no s'arriba a distorsionar la mateixa, de forma que els pics són molt clarament identificables.

Considerarem qualsevol senyal a l'entrada analògica que passi de 3000 com a susceptible de formar part d'un pic. Estudiant les dades de la mostra de la figura 29, i d'altres que s'han fet, s'ha observat que, per a 5ms de separació entre mostres:

- els pics que representen batecs tenen unes 20-32 mostres amb un valor superior a 3000 a l'entrada de l'ADC (uns 100-164 ms). Com que això s'ha calculat per a una taxa d'uns 55-60 batecs per minut, es dedueix que, a taxes molt elevades d'uns 180 batecs per minut, el senyal de la mostra durant un pic estarà per sobre de 3000 almenys durant 6 mostres (30 ms).
- els pics «anòmals» no passen mai de 3000 durant més de una mostra, i el seu valor sol ser (encara que no sempre) el màxim que dona l'entrada analògica (4095). Això s'ha confirmat en vàries sèries de mostres.

- el senyal no passa mai de 3000 en qualsevol altre cas.

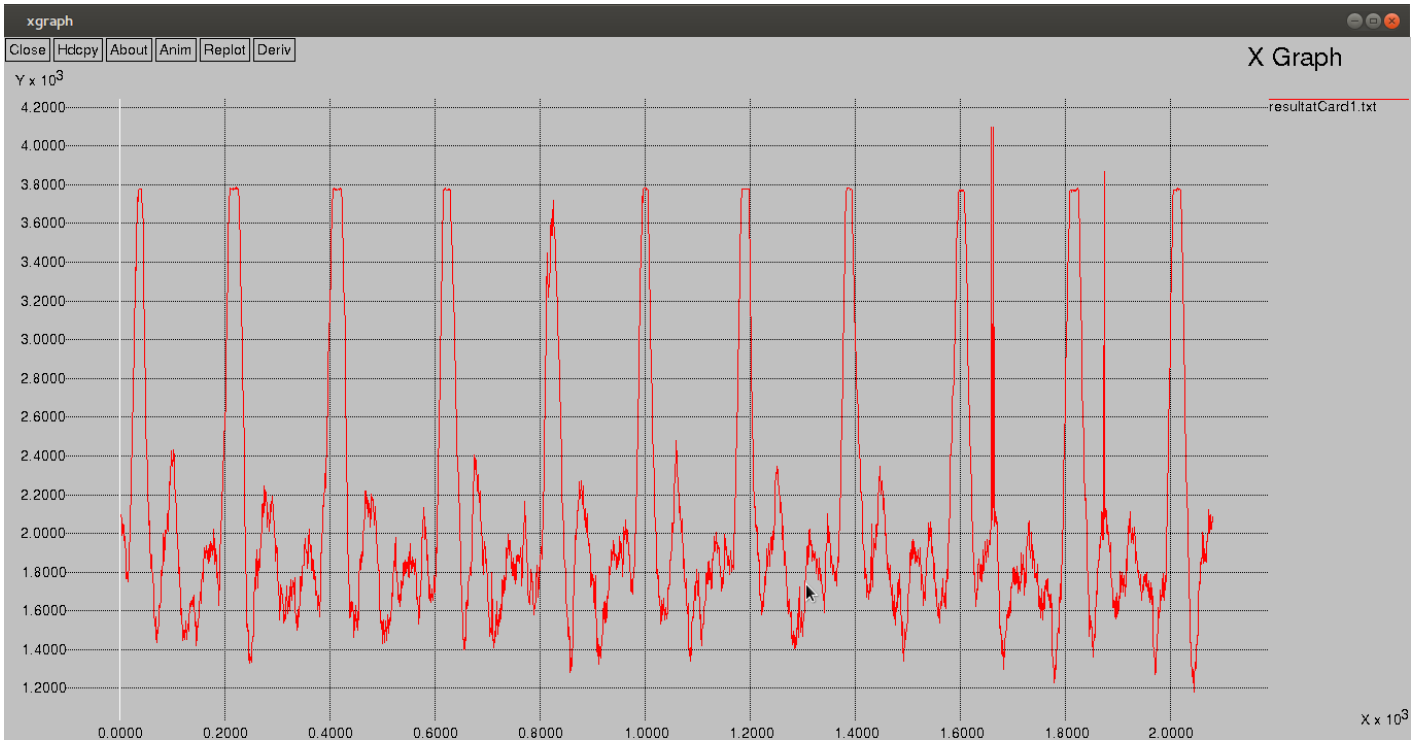


Figura 29: Exemple de patró de sortida real del fotopletismògraf

Tenint tot això en compte, es pot entendre millor el funcionament de l'algorisme comptador de batecs i nivell de bateria, del qual resumim els seus passos principals (veure figures 24 i 25):

- en primer lloc, tenim un bucle principal, tant en l'estat normal com en el passiu, que s'executa a un ritme regular de 5 ms. Executarem la funció de l'API de FreeRTOS *vTaskDelayUntil()* al principi del bucle per assegurar-nos d'això. Així, podem bloquejar la funció assegurant-nos que es desbloqueja a intervals regulars de 5ms.[5]

- a continuació, si estem en estat normal, fem la lectura de l'entrada digital AD0.4.

- si l'entrada val més de 3000, és possible que ens trobem davant un batec. Comprovarem que el valor es manté superior a 3000 durant 5 iteracions del bucle, i si és així incrementem el nombre de batecs registrats. Llavors, entrem a l'estat passiu, durant el qual continuarem fent iteracions del bucle, però durant 250 ms no llegirem mesures del fotopletismògraf, per donar temps a que passi el pic, i no comptar varis batecs en l'espai que correspon a un sol pic.

5 Veurem una mica més endavant que passa si el processament entre dues crides a *vTaskDelayUntil()* supera els 5ms.

- en cas que l'entrada no es mantingui a més de 3000 durant 5 mostres, descartarem el possible batec com a pic «anòmal».
- repetim el bucle anterior fins comptar 10 batecs i es processa la taxa de batecs (figura 27):
  - calculant la taxa de batecs dividint el nombre de batecs per l'interval temporal total.
  - enviant un missatge al servidor amb aquesta informació.

A més, en l'interior del bucle es realitzen dues tasques més si estem en estat normal (figura 24), i només una si estem en estat passiu (figura 25).

- en estat normal:
  - si la darrera mesura indica que no hi ha batec, abans de fer la nova iteració comprovarem que no han passat més de 2,5 segons des del darrer batec. Si és així, enviem un missatge de 0 batecs al servidor.
  - cada 10000 iteracions del bucle, fem una mesura del nivell de la bateria i enviem el missatge al servidor.
- en estat passiu, fem iteracions sense fer mesures de batecs durant 250 ms. Si mentre estem en aquest estat es compleixen 10000 iteracions del bucle, processem el senyal de bateria exactament igual que com si estiguéssim en l'estat actiu.

Càlcul taxa de batecs

El càlcul de la taxa de batecs es fa de la següent forma:

- després de detectar el primer batec de la sèrie, s'inicialitza la variable *timeTot,* que portarà el compte del temps de l'interval de 10 batecs.
- *timeTot* s'actualitzarà en cada iteració del bucle, aprofitant que sabem que es fan a intervals de 5ms. Hi ha dos situacions en que no podem garantir, però, que la iteració tardi menys de 5ms:
  - quan en mig de la sèrie de 10 batecs hem d'enviar un missatge de 0 batecs perquè han passat més de 2,5 segons sense batec.
  - quan hem d'enviar un missatge de nivell de bateria.

En qualsevol d'aquests dos casos, reiniciarem el recompte de la sèrie de 10 batecs, el temps total, i la variable de sincronització per a *vTaskDelayUntil()*.

- quan s'hagin detectat els 10 batecs, tindrem que el temps mig de l'interval interbatec serà:

  $$\mathbf{t_{interbatec} = timeTot / 9}$$

  ja que hem mesurat 9 intervals. La taxa de batecs per minut serà per tant:

  $$\mathbf{taxa\ de\ batecs\ (\ batecs\ /\ min) = (60000)\ /\ t_{interbatec} = 540000\ /\ timeTot}$$

Càlcul nivell de bateria

Per fer el càlcul del nivell de bateria, tenim definides dues constants al fitxer *configurations.h*:

- ***PERCENTAGE_BATT_TRESHOLD***

  Indica la fracció del voltatge màxim teòric del divisor de tensió per sota del qual es considerarà que la bateria està en un nivell baix. En el nostre cas valdrà «0.75». L'elecció que hem fet és fins a cert punt convencional. Una elecció més acurada exigiria un estudi de les corbes de descàrrega típiques dels diversos tipus de bateria, i escollir el valor anterior a partir d'aquestes corbes.

- ***MAX_SIGNAL***

  Ens indica el senyal digital de sortida del canal AD0.1 quan la tensió d'entrada és la màxima teòricament possible. Com hem vist a la secció **3.1.4**., la tensió de sortida del divisor de tensió quan la de la bateria és de **5V** serà de $\mathbf{V_d = 1{,}16\ V}$. Llavors, la sortida del conversor AD0.1 serà de (1,16 V / 3,3 V) * 4095 = **1440**, valor que hem assignat a MAX_SIGNAL.

Així doncs, l'única cosa que haurem de fer és comparar el senyal analògic que ens dona la bateria a través de l'entrada AD0.1, amb el senyal llindar que indica el límit entre nivell de bateria alt i baix. Aquest senyal llindar serà:

PERCENTAGE_BATT_TRESHOLD * MAX_SIGNAL

4.1.2. Diagrama de flux vTaskFall i EINT3_IRQHandler()

La figura 30 mostra el diagrama de flux de la tasca *vTaskFall* i del gestor d'interrupcions *EINT3_IRQHandler()*.

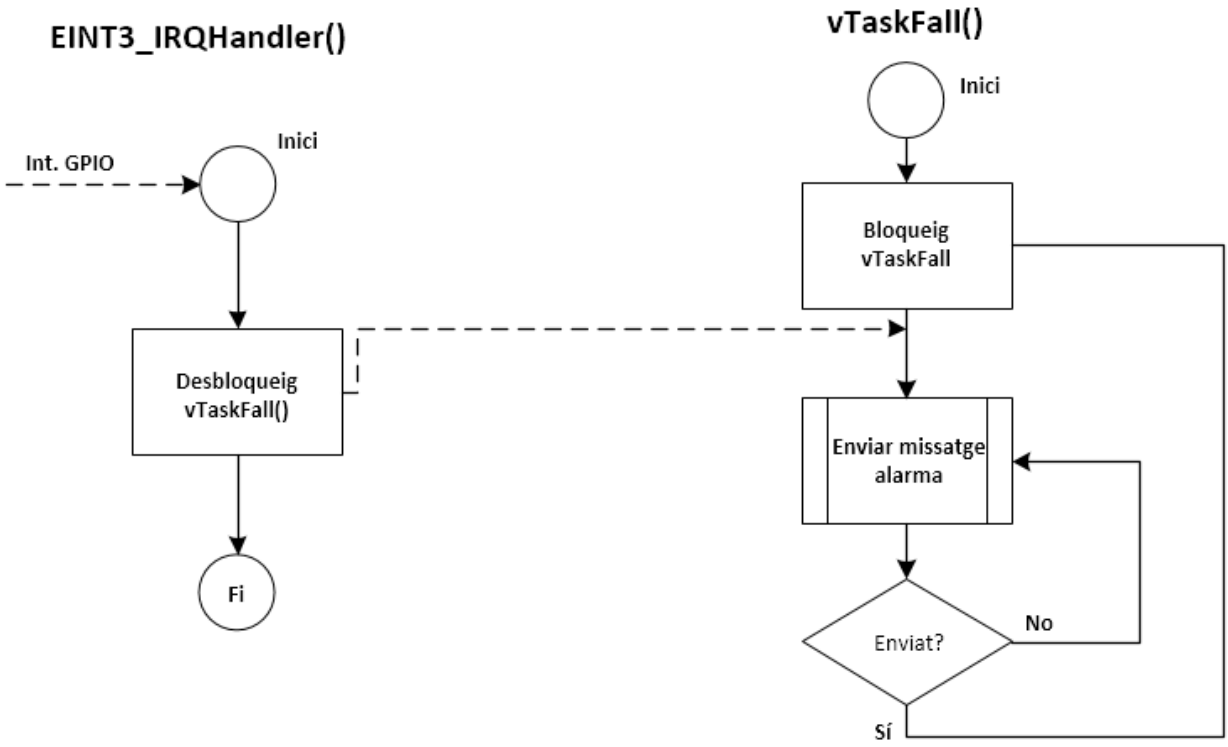


Figura 30: Diagrama gestor de caigudes

Implementació del gestor d'interrupcions i sincronització amb la tasca vTaskFall()

Recordem que ja tindrem configurat el pin P0.4 com a GPIO d'entrada (veure **Configurar GPIO entrada**, apartat 4.1.1.3.**).** Un cop fet això, i a la mateix funció (per tant *vHeartBeatRate*), haurem de [10]:

- indicar al controlador d'interrupcions (NVIC) quins pins i quin tipus de transicions (alt o baix) dispararan interrupcions. El registre que habilita interrupcions en el flanc de pujada és *IO0IntEnR*. Com que l'acceleròmetre indica una condició de 0g posant el senyal a 1, posarem a 1 el 4rt bit de *IO0IntEnR*, que correspon a habilitar la interrupció en el flanc de pujada de P0.4.

- indicar al controlador d'interrupcions que s'ha d'habilitar el gestor d'interrupcions que s'encarrega de la interrupció configurada en el punt anterior. Ho farem amb:

  NVIC_EnableIRQ(EINT3_IRQn)

on *EINT3_IRQn* és una macro amb el valor de 21 (totes les interrupcions sobre els pins GPIO es gestionen via la interrupció 21).

- definir *vTaskFall()*. És una funció que s'executa amb alta prioritat (superior a *vTaskHeartRate*), i que fa servir un semàfor binari per sincronitzar-se amb les interrupcions generades per l'acceleròmetre.

  El semàfor serà una variable global a nivell de mòdul, a la que accediran *vTaskFall()* i *EINT3_IRQn(),* de tipus *xSemaphoreHandle.* La creem amb la funció de la API de FreeRTOS *vSemaphoreCreateBinary().* L'accés al semàfor binari es fa a través de dues funcions de la mateixa API:

  - *xSemaphoreTake()* intenta agafar el semàfor. En crear-se, el semàfor binari (que és una cua de mida 1) està ple[6]. Per tant, en executar per primer cop *vTaskFall()*, haurem de cridar la funció dues vegades perquè es bloquegi. Quan el gestor d'interrupcions torni a donar el semàfor, i *vTask()* es desbloquegi, bastarà que *vTaskFall()* cridi *xSemaphoreTake()* una sola vegada per què torni a bloquejar-se.
  - *xSemaphoreGiveFromISR()* «dona» el semàfor. Llavors est disponible per a qualsevol tasca que estigui esperant-lo mitjançant una crida a *xSemaphoreTake()*, o que cridi aquesta funció en el futur. Aquesta versió de la funció *xSemaphoreGive()* és la que s'ha de fer servir si la cridem des d'una rutina de gestió d'interrupcions, com és el cas.

  El primer que fa *vFallTask()* (veure Figura 30) és intentar agafar el semàfor (dues vegades en la primera iteració, després bastarà amb una), i bloquejar-se fins que el gestor d'interrupció el «doni». Quan això passi, en produir-se una interrupció 0g, *vFallTask()* s'executarà de forma immediats degut a la seva alta prioritat i enviarà un missatge d'alarma de caiguda. Després es bloquejarà en intentar prendre el semàfor de nou.

- Finalment, definim el gestor d'interrupció *EINT3_IRQHandler(),* que haurà d'efectuar dues tasques:

  - desbloquejar a *vTaskFall()* donant el semàfor.

6 Aquest comportament no és el que està documentat als manuals oficials de FreeRTOS i a la seva plana web, que indiquen, erròniament, que per defecte el semàfor binari es crea buit.

- indicar que la interrupció ja ha estat tractada i resetejar el flag que indica que s'ha produït la interrupció. Això es fa posant a 1 el bit corresponent de *IO0IntClr* (en aquest cas, el 4).

4.1.3. Diagrama de flux de la subtasca d'enviament de missatges al servidor

En els diagrames anteriors hem vist de forma repetida la subtasca d'enviament de missatge al servidor. Aquesta subtasca no inclou només l'enviament de missatge, sinó també la recepció de la resposta per part del servidor, que sempre serà un missatge «OK» i, en cas, que aquesta no es rebi, la reinicialització de la connexió. El seu diagrama de flux el podem veure a la figura 31.

La subtasca iniciar connexió és la que s'ha descrit a la secció **4.1.1.1**, i no acaba fins que la connexió s'ha establert.

4.2. Servidor

Passem a descriure breument el funcionament del servidor. Com hem dit anteriorment, consta de quatre classes:

- **Server**

  És la classe principal de l'aplicació, ja que conté el *main()*, té com a components principals:

  - un ArrayList d'elements ConBoard, representant cadascun una connexió a una placa.
  - un ServerGUI, que és la interfície gràfica del servidor.

  Els mètodes principals que conté són.

  - *main()*, que crea un objecte Server i l'arrenca.
  - *start()*, és el mètode que posa en marxa el Server. Crea un ServerSocket que accepta connexions de forma indefinida. Quan s'estableix una connexió, es crea un nou ConBoard i se li passa el *socket* amb la connexió a la placa.

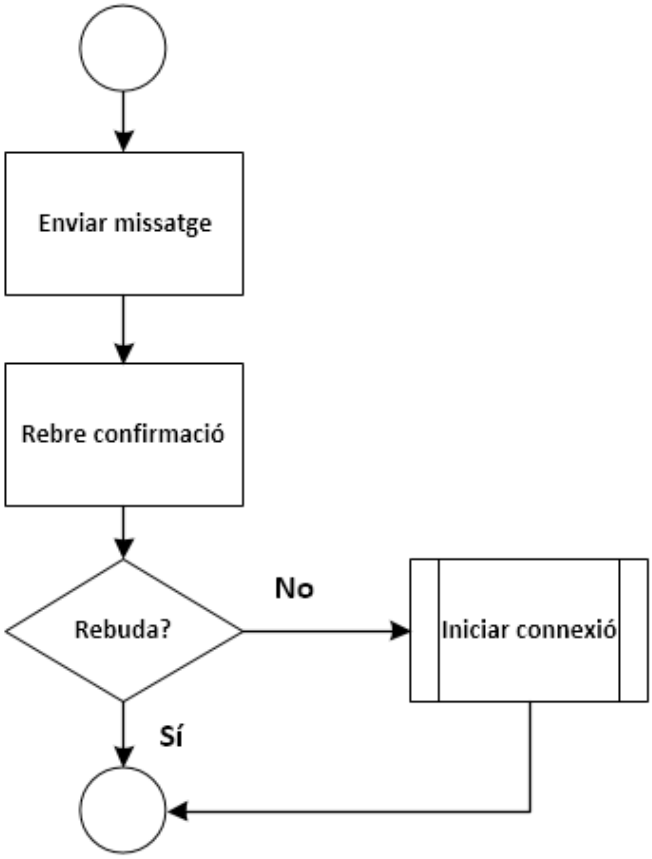


Figura 31. Enviar missatge

- **ConBoard**

  Com ja hem dit, representa una connexió amb una placa. Hereta de la classe Thread, així que cada connexió s'executa en un fil diferent. Els principals components que conté són:

  - un Socket de connexió amb una placa. Té un *timeout* de 30 seg, després dels quals el servidor considerarà la placa com a desconnectada.
  - un BoardPanel, que mostra a la GUI la informació que envia la placa, i el seu estat.
  - vàries variables (*connected*, *disposable*) que mostren l'estat de la connexió i si l'usuari ha marcat el ConBoard per ser eliminat o no (en polsar el botó «Eliminar»).
  - el mètode *run()* és el mètode principal de ConBoard. Conté un bucle que es va executant de forma contínua mentre l'usuari no marqui l'objecte per eliminar. El bucle executa una sèrie de d'accions depenent de si la connexió és activa o no:

- si és activa, rep els missatges a través del *socket* i selecciona amb un *switch* les accions a executar. La primera sempre serà respondre a la placa amb un missatge d'OK.
    - Codi 10 (alarma): mostra el missatge d'alarma al BoardPanel i activa el botó de Reset.
    - Codi 20 (taxa de batecs): actualitza la taxa de batecs al BoardPanel.
    - Codi 30 (identificació de placa): és el missatge que es rep immediatament després de que una placa es connecti. Primer, comprova que no hi hagi ja un ConBoard per a una placa amb el mateix id. Si existeix, es tractarà d'una placa que s'ha desconnectat, no s'ha eliminat el ConBoard corresponent, i ara es torna a connectar. En aquest cas, s'elimina el ConBoard antic, i el panell del nou ConBoard substitueix al de l'antic. També ens hem d'assegurar que si es mostrava una alarma de caiguda en l'antic ConBoard, també el nou el mostri.
    - Codi 40 (nivell de bateria): si indica un nivell de bateria baix, es mostrarà al BoardPanel de la placa. En cas contrari, no es mostrarà cap missatge.

- **BoardPanel**

  Representa el panell d'informació per a una placa, i conté per tant les diferents etiquetes i botons. Hereta de la classe JPanel i implementa la interfície ActionListener per tal de poder implementar les accions a realitzar en polsar un dels botons que conté, que són:

  - Reset: Només habilitat quan es mostra un missatge d'alarma. Polsant-lo, s'elimina el missatge i es deshabilita el propi botó.
  - Remove: Només habilitat quan s'ha desconnectat la placa (és a dir, quan s'ha exhaurit el *timeout* del *socket* del ConBoard corresponent). Polsar aquest botó posa a *true* la variable *disposable* del ConBoard corresponent, amb el que se li indica que surti del bucli infinit de recepció de missatges i que acabi el seu fil d'execució.

- **ServerGUI**

  Hereta de la classe JFrame, i és la GUI del sistema. Conté un ArrayList amb els diferents BoardPanels existents, que es visualitzen verticalment a la finestra.

5. Descripció dels diferents drivers

A continuació descriuré breument els diferents *drivers* que he fet servir i les seves funcions principals.

printf.c

Conté funcions per inicialitzar i enviar dades a una UART en formats diferents. Es tracta d'un *wrapper* per a l'accés segur a les funcions del driver *uart.c*, on es fan aquestes tasques a baix nivell.

- **uint32_t Print_Init(uint8_t PortNum, uint32_t BaudRate)**

  Inicialitza la UART **PortNum**, amb una taxa de dades de **BaudRate.** Retorna TRUE o FALSE segons si **PortNum** és una UART vàlida o no. Crida a **UART_Init()**.

- **void Lock_Print()**

  «Pren» un semàfor per bloquejar l'accés concurrent a una UART.

- **void UnLock_Print()**

  «Torna» un semàfor per tal de desbloquejar l'accés a una UART.

- **void Print_Text (uint8_t PortNum, char *Text)**

  Transmet el text al que apunta **Text** per la UART **PortNum.** Crida a **UART_Send()**.

- **void Print_Flush (uint8_t PortNum)**

  Esborra el buffer que fa servir la UART **PortNum.** Crida a **UART_Flush()**.

- **uint32_t CheckRead (uint8_t PortNum, char *Text, char *Value, uint32_t Time)**

  Llegeix una cadena per la UART **PortNum** durant **Time** ms. Emmagatzema la cadena a la zona de memòria apuntada per **Text**. Retorna TRUE o FALSE depenent de si la cadena llegida és o no igual a la cadena apuntada per **Value.** Crida a **UART_Read()**.

- **void Print_TT_CRLF (uint8_t PortNum, char *Text, char *Word)**

  Envia a la UART **PortNum** un text compost per les dues cadenes apuntades per **Text** i **Word**, més un caràcter de fi de línia. Crida a **UART_Send()**.

- **void Print_T_CRLF (uint8_t PortNum, char *Text)**

  Envia a la UART **PortNum** un text compost la cadena apuntada per **Text**, més un caràcter de fi de línia. Crida a **UART_Send()**.

uart.c

Conté les funcions que permeten manipular i configurar les UARTs a baix nivell.

- **uint32_t UART_Init (uint8_t PortNum, uint32_t Baudrate)**

  Inicialitza tots els paràmetres necessaris per a fer servir la UART **PortNum** a una taxa de transferència de **BaudRate.** Retorna TRUE si **PortNum** és un número de UART vàlida, i FALSE en cas contrari.

- **void UART_Flush (uint8_t PortNum)**

  Posa a 0 la zona de memòria que fa servir com a buffer la UART **PortNum**.

- **void UART_Send (uint8_t PortNum, char *BufferPtr)**

  Envia la cadena apuntada per **BufferPtr** a la UART **PortNum**. Ho fa byte a byte fent servir **UART_Write()**.

- **void UART_Write (uint8_t PortNum, uint8_t send)**

  Envia el byte **send** a la UART **PortNum**, escrivint el byte al registre de la UART i esperant a que tot el byte s'hagi transmès.

wifly.c

Implementa les principals accions a realitzar sobre la Wifly. Per a les accions que impliquen actuar sobre el port sèrie, es fan servir les funcions contingudes a **printf.c** i/o **uart.c**.

- **uint32_t Wifly_Init (uint8_t PortNum, uint32_t Baudrate)**

  Configura la UART **PortNum**, que serà utilitzada per la Wifly, amb la velocitat de transferència **Baudrate**, i reseteja la Wifly. Crida, entre d'altres, a **Print_Init()**. Retorna TRUE o FALSE segons si **PortNum** és una UART vàlida o no.

- **void Lock_Wifly()**

«Pren» un semàfor per tal de bloquejar l'accés concurrent a la UART de la Wifly.

- **void UnLockWifly()**

  «Retorna» un semàfor per tal de desbloquejar l'accés concurrent a la UART de la Wifly.

- **void Wifly_Reset()**

  Reseteja per software la *wifly*, activant el pin GPIO connectat a la seva entrada RESET.

- **uint32_t Wifly_JoinSSID (uint8_t PortNum, char * ssid, char * passwd, uint8_t Mode)**

  Connecta la *wifly* associada a **PortNum** a la xarxa wifi **ssid**, fent servir el mode **Mode** i la contrasenya **passwd**. Si el missatge de resposta és l'esperat, retorna **TRUE,** si no, **FALSE**. Crida, entre d'altres, a **Print_TT_CRLF()**.

- **uint32_t Wifly_CMD (uint8_t PortNum)**

  Envia la cadena «$$$» a la Wifly connectada a la UART **PortNum**, per tal de posar-la en mode comanda. Si la resposta és l'esperada retorna **TRUE**, si no, **FALSE**. Crida, entre d'altres, a **Print_Text()**, ja que «$$$» no ha finalitzar amb caràcter de nova línia.

- **char *Wifly_Set_TT_Check (uint8_t PortNum, char *Command, char *Value, char *Check, uint32_t Time)**

  Envia la comanda apuntada per **Command** a la UART **PortNum**. Espera resposta durant **Time** ms. Si la resposta, emmagatzemada al buffer **Value**, és igual a la cadena apuntada per **Check**, retorna **TRUE**. En cas contrari, retorna **FALSE**. Crida, entre d'altres, a **Check_Read()**.

- **uint32_t Wifly_Open (uint8_t PortNum, char *Address)**

  Envia la comanda «open» amb l'adreça i port emmagatzemada al buffer **Address** a la UART **PortNum**.Això permet obrir un port TCP a l'adreça i port indicats. Retorna **TRUE** si la UART retorna la resposta esperada, i **FALSE** en cas contrari.

adc.c

Conté les funcions que permeten configurar un canal ADC de la LPC1769.

- **void adcInit()**

Inicialitza el conversor analògic digital (veure **Configurar entrada analògica**, **4.1.1.3.**)

- **void ADC_Channel(int ch)**

  Habilita el canal analògic **ch**, (veure **Configurar entrada analògica**, **4.1.1.3.**).

comm.c

Proporciona una capa intermèdia entre els drivers *print.c* i *wifly.c,* i l'aplicació principal, a més proporcionar funcions per a la gestió dels missatges.

- **void debugInit()**

  Inicialitza la UART de debug i mostra un missatge amb el resultat. Crida a **Print_Init()**.

- **void wiflyInit()**

  Inicialitza la UART de la wifly i mostra un missatge amb el resultat. Crida a **Wifly_Init()**.

- **void joinSSID()**

  Permet unir-se a una xarxa wifi. Crida a **Wifly_JoinSSID()**.

- **uint8_t OpenConn()**

  Construeix la URL del servidor, i obre una connexió. Crida a **Wifly_open()**, i retorna TRUE o FALSE segons si ha tingut èxit o no.

- **void connInit()**

  Correspon al diagrama de flux de la figura 22. Reseteja la wifly i segueix tots els passos per tenir una connexió operativa amb el servidor. No retorna fins que aquesta no s'ha establert.

- **void stablishConn()**

  Crida a **connInit()** per establir la connexió i envia el missatge d'identificació. Repeteix el procés fins a rebre la resposta que s'espera del servidor.

- **uint8_t sendMessage(uint_8 type, uint8_t param)**

  Envia un missatge de tipus **type**. Retorna TRUE o FALSE segons si el servidor ha respòs amb OK o no.

- **void sendMessageCheckConn(uint_8 type, uint8_t param)**

  Envia un missatge de tipus **type**, fent servir **sendMessage()**. Si el servidor no dona resposta, reinicia la connexió cridant a **stablishConn()**.

6. Valoració econòmica del projecte

La valoració econòmica del desenvolupament i implantació d'una sola placa de monitorització es mostra a la taula 8.

El preu total de desenvolupament i implantació d'una sola placa seria de 3945 € aproximadament. La major part del cost es degut a les hores de desenvolupament, i seria per tant un cost fix que no augmentaria en crear més plaques (o augmentaria poc).

El preu és elevat però pot ser assumible si desenvolupem un nombre suficient de sistemes, tenint en compte a més que els costos dels components probablement baixarien bastant en fabricar quantitats elevades del nostre producte.

Veiem un cas hipotètic: si subministrem el nostre producte a 25 hospitals, cadascun amb 200 llits de mitjana, i suposant que s'adquireix una placa per cada cinc llits, això representarien unes 1000 plaques, amb un cost total de 348.600 €[7], i un cost unitari de 348 € aproximadament.

Hem de tenir en compte, a més, que estem parlant d'un sistema amb un ús mèdic (i que de fet podria adaptar-se fàcilment per a altres aplicacions com la seguretat en el treball). Per tant, no es tracta simplement d'un objecte de consum, sinó que, sempre que demostri el seu impacte real en la millora de la salut pública (o la seguretat en el treball) o de la qualitat assistencial, pot esdevenir inclús d'ús més o menys obligat per part dels establiments sanitaris (o les empreses). Seguint amb el nostre escenari, si cada sensor tingués una vida operativa de 2 anys, suposant que passen 120 pacients a l'any per llit (1 cada tres dies) i que només 1 de cada 10 necessités portar el nostre sensor, tindríem 60.000 pacients per any que farien servir el sensor, i 120.000 durant els 3 anys. Això representa 120 pacients per sensor, i un cost unitari de 2,9 € per pacient, o de 1,45 € per pacient i dia d'hospitalització.

Sigui com sigui que avaluem la millora en seguretat, benestar i qualitat assistencial a cadascun dels

7 Evidentment és només una estimació ja que, si bé no hem inclòs els costos de fabricació referents a mà d'obra, lloguers, etc , tampoc hem tingut en compte que el cost unitari dels materials serà inferior al calculat abans, degut al volum de producció.

pacients per l'ús del sensor, serà probablement superior a aquests 1,45 € per pacient i dia.

Material	Preu unitari	Quantitat	Preu total
LPC1769	20 €	1	20 €
Wifly RN-XV	32 €	1	32 €
Adaptador UART-USB CP2102 + cables connexió	4 €	1	4 €
Acceleròmetre MMA7361	9,33 €	1	9,33 €
Sensor de polsos SEN-11574	29,75 €	1	29,75 €
Servidor	250 €	1	250 €
Hores de desenvolupament	12 € / h	300 h	3.600 €

Taula 8: Estimació del cost del projecte

També s'ha de tenir en compte que, amb algunes petites adaptacions, es podria fer servir el sistema per a determinades activitat laborals, on també la detecció de caigudes i/o la detecció d'alts nivells d'estrès (per exemple, en bombers, treballadors en ambients perillosos o en alçada, etc) és important. De nou, un sistema com el nostre que certifiqués la seva utilitat en la reducció del risc en aquests àmbits podria inclús esdevenir obligatori o en tot cas, tenir una molt bona sortida.

7. Conclusions i possibles millores a implementar

Les possibles millores a implementar en el projecte actual són moltes, com per exemple:

- Afegir funcionalitat de servidor web al servidor del sistema, per tal que es pugui accedir a les dades de les plaques des de qualsevol navegador.
- Implementar un *watchdog* per tal que el sistema es reinicialitzi si per qualsevol motiu deixa de respondre o es queda «penjat».
- Permetre a l'usuari configurar el mode d'operació del sistema des de la GUI del servidor. Com he indicat a la secció d'Objectius, aquests modes d'operació podrien ser:
  - Detecció de batecs + sensor de caigudes.
  - Només detecció de batecs.

- Només sensor de caigudes.

- Millorar el sistema d'alerta, de forma que s'enviï un SMS o algun missatge per algun servei de missatgeria al personal de l'hospital en cas que es produeixi alguna situació que requereixi intervenció immediata.

- Millorar l'algorisme de detecció de batecs de forma que pugui distingir entre pols molt baix o absència del mateix, i sensor desconnectat.

Com a conclusió del projecte, considero l'experiència en general com a molt positiva (i esgotadora!), ja que m'ha permès obtenir una visió bastant general del funcionament d'un sistema encastat en tots els seus nivells, i per extensió, de qualsevol sistema programable. Es tracta, d'un camp amb múltiples aplicacions, i que dona molt espai per a la inventiva i la creativitat de cadascú.

Respecte del resultat final, crec que és bastat satisfactori. Els objectius inicials plantejats s'han acomplert, i s'ha implementat un objectiu suplementari, com és el de fer que el servidor detectés i gestionés de forma adequada les pèrdues de connexió.

8. Bibliografia i recursos web[8]

8.1. Bibliografia

1. Barry, Richard. ***Using the FreeRTOS Real Time Kernel*** (2009). FreeRTOS.

2. Freescale Semiconductor. ***Technical Data. ±1.5g, ±6g Three Axis Low-g Micromachined Accelerometer.*** Abril 2008. Disponible a: http://www.freescale.com/files/sensors/doc/data_sheet/MMA7361L.pdf

3. NXP. ***LPC176x/5x User manual.*** Revisió 3.1. Abril 2014. Disponible a http://www.nxp.com/documents/user_manual/UM10360.pdf

4. Roving Networks. ***Wifly Command Reference, Advanced Features & Applications User's Guide.*** Abril 2013. Disponible a http://ww1.microchip.com/downloads/en/DeviceDoc/rn-wiflycr-ug-v1.2r.pdf

5. Stefan Hey, et al. 2009. ***Continuous Noninvasive Pulse Transit Time Measurement for***

8 Tots els enllaços han estat comprovats amb data 12/06/2015

Psycho-physiological Stress Monitoring. In *Proceedings of the 2009 International Conference on eHealth, Telemedicine, and Social Medicine* (ETELEMED '09). IEEE Computer Society, Washington, DC, USA, 113-116.

8.2. Recursos web

6. LPCXpresso Forum: http://www.lpcware.com/forums/lpcxpresso/lpcxpresso-forum

7. Oracle Java Documentation. ***Trail: Creating a GUI with JFC/Swing***: https://docs.oracle.com/javase/tutorial/uiswing/index.html

8. Oracle Java Documentation. ***Trail: Custom Networking:*** https://docs.oracle.com/javase/tutorial/networking/index.html

9. ***Pulse Sensor Amped Technical Article***. http://pulsesensor.com/pages/pulse-sensor-amped-arduino-v1dot1

10. Ramesh, Rohit. ***Introduction To Embedded Systems Development.*** http://www.cs.umd.edu/~rohit/ESDbook.pdf

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

ANNEXOS

1. Codi del projecte

En aquest apartat incloem el codi del projecte. En primer lloc hi haurà el codi del sistema encastat. Per qüestions d'espai, no s'inclouran tots el drivers, sinó només les que he desenvolupat jo, és a dir `adc.c` i `comm,c`, tot i que a `wifly.c` també he fet alguna modificació menor. Tampoc s'inclouen els fitxers de capçalera.

1.1. Sistema encastat

Fitxer Main.c

```
#include "LPC17xx.h"
#include "FreeRTOS.h"
#include "task.h"
```

```
#include "semphr.h"
#include "queue.h"
#include "type.h"
#include "printf.h"
#include "wifly.h"
#include "adc.h"
#include "comm.h"
#include "config.h"
#include "stdio.h"
#include "string.h"
#include "configurations.h"

// Task declaration
static void vTaskHeartRate( void *pvParameters );
static void vTaskFall( void *pvParameters );
void systemInit(void);
void readBattery(void);

/* Declare a binary semaphore for fall task synchronization */
static xSemaphoreHandle xSemFall =NULL;

/
*********************************************************************
 ** Function name:         EINT3_IRQHandler
 **
 ** Description:      External Interrupt 3 Handler
 **
 ** Parameters:            None
 ** Returned value:        None
*********************************************************************/
void EINT3_IRQHandler()
{
      static portBASE_TYPE xHigherPriorityTaskWoken;
      xSemaphoreGiveFromISR(xSemFall,&xHigherPriorityTaskWoken);
      LPC_GPIOINT->IO0IntClr |= (1 << 4);        //Clear interrupt on
                                                 //P0[4]
}

/
*********************************************************************
 ** Function name:         main
 **
 ** Description:      main function
 **
 ** Parameters:            None
 ** Returned value:        int
*********************************************************************/
int main( void )
{
      /* Create binary semaphore for fall task sync */
      vSemaphoreCreateBinary(xSemFall);

      /* Only create the tasks if the semaphore was created successfully.
*/
```

```
	if (xSemFall != NULL)
	{
	/* Task Creation */
	xTaskCreate( vTaskHeartRate, (char *)"HeartRate", 240, NULL, 1,
NULL);
	xTaskCreate( vTaskFall, (char *)"FallDetection", 240, NULL, 3,
NULL);
		/* Start the tasks running. */
		vTaskStartScheduler();
	}

	/* If all is well we will never reach here as the scheduler will
now be running.  If we do reach here then it is likely that there was
insufficient heap available for the idle task to be created. */
	for( ;; );
	return 0;
}

/
*****************************************************************************
 ** Task name:		vTaskHeartRate
 **
 ** Description:	initializes peripherals
 **			monitors heart rate and sends to server
 **			monitors battery and sends to server
 **
 ** Parameters:		pointer to whatever structure desired (not
used)
 ** Returned value:	none

*****************************************************************************/
static void vTaskHeartRate( void *pvParameters )
{
	uint16_t beatSignal=0;		//photopletysmograph signal level

	uint16_t timeTot=0;		//accumulated time (10 beats)
	uint16_t timeLastBeat=0;
	uint8_t countTreshold=0;	//signal>3000 counter
	uint8_t numBeats=0;
	uint8_t state=1;		//counting state	(0= ignore
signals ; 1 = don't ignore them)
	uint32_t countIt=0;  //counter of iterations. When 10000 we'll read
battery level
	portTickType xLastWakeTime;

	systemInit();

	//time counter initialization
	xLastWakeTime = xTaskGetTickCount();

	for( ;; )
	{
		vTaskDelayUntil(&xLastWakeTime, 5/portTICK_RATE_MS);  //heart
reading interval is 5 ms
```

```
            timeTot=timeTot+5;
            timeLastBeat=timeLastBeat+5;
            countIt++;

            /* Read value in AD0.4*/
            while((LPC_ADC->ADDR4 >> 31) && 1){} //Wait until AD0.4
conversion
            beatSignal = (LPC_ADC->ADDR4>>4) & 0xFFF; //Read value

            //if we count a signal>3000 five consecutive times, in normal
state, we have a beat
            if (beatSignal>BEAT_SIGN_TRESHOLD)
            {
                  countTreshold++;

                  if (countTreshold==5 && state)
                  {
                        numBeats++;
                        /*start counting in the first beat*/
                        if (numBeats==1)
                        {
                              timeTot=0;
                        }
                        //Calculate beat rate if we have 10 beats
                        else if (numBeats==10)
                        {
                              sendMessageCheckConn(20,540000/timeTot);

                              //before starting new round of 10 beat count,
we reset times
                              xLastWakeTime = xTaskGetTickCount();
                              numBeats=0;
                        }
                        countTreshold=0;
                        timeLastBeat=0;
                        state=0;   //after counting a beat, we will ignore
signals for a while
                  }
            }
            else
            {
                  countTreshold=0;
            }
            //250 ms after the last Beat, we stop ignoring signals
            if (timeLastBeat==NON_COUNT_INTERVAL)
                  state=1;
            //if 2,5 seconds since last beat, send a message to the server
with 0 beat/sec rate.
            if (timeLastBeat==ZERO_BEAT_INTERVAL)
            {
                  sendMessageCheckConn(20,0);

                  //before starting new round of 10 beat count, we reset
```

```
times
                numBeats=0;
                timeLastBeat=0;
                xLastWakeTime = xTaskGetTickCount();
          }
          //every 10000 signals, read battery
          if (countIt==BATTERY_INTERVAL)
          {
                countIt=0;

                readBattery();

                //before starting new round of 10 beat count, we reset
times
                xLastWakeTime = xTaskGetTickCount();
                timeLastBeat=0;
                numBeats=0;
          }
     }
}

/
*****************************************************************************
 ** Task name:             SystemInit
 **
 ** Description:      Initializes required hardware
 **
 ** Parameters:            none
 ** Returned value:        none

*****************************************************************************/
void systemInit(void){

     debugInit();
     wiflyInit();

     //joinSSID();
     stablishConn();

     adcInit();      //Initializes ADC for 0g and battery level
detection
     adcChannel(4);  //Set channel 4 for AD0.4
     adcChannel(1); //Set channel 1 for AD0.1

     //Enables P0.4 as input GPIO pin
     LPC_PINCON->PINSEL0 &=~((1<<9)|(1<<8));
     LPC_GPIO0->FIODIR &= ~(1<<4);

     LPC_GPIOINT->IO0IntEnR |= (1<<4);     //Initialize interruption for
P0.4 in rising flank
     NVIC_EnableIRQ(EINT3_IRQn);               //Enable interrupt
number 3
}
```

```
/
*********************************************************************
 ** Task name:             readBattery
 **
 ** Description:      read and sends battery level
 **
 ** Parameters:            none
 ** Returned value:        none

*********************************************************************/
void readBattery(void){

      uint16_t batterySignal=0;
      uint8_t batteryState=0;

      //read battery level
      while((LPC_ADC->ADDR1 >> 31) && 1){}
      batterySignal = (LPC_ADC->ADDR1>>4) & 0xFFF;

      if (batterySignal<(PERCENTAGE_BATT_TRESHOLD*MAX_SIGNAL))
      {
            batteryState=0;
      }
      else if (batterySignal>=(PERCENTAGE_BATT_TRESHOLD*MAX_SIGNAL))
      {
            batteryState=1;
      }
      sendMessageCheckConn(40,batteryState);
}

/
*********************************************************************
 ** Task name:             vTaskFall
 **
 ** Description:      handle fall condition.
 **
 ** Parameters:            pointer to whatever structure desired
 ** Returned value:        none

*********************************************************************/
static void vTaskFall( void *pvParameters )
{
      xSemaphoreTake(xSemFall,portMAX_DELAY);
      for (;;)
      {
            xSemaphoreTake(xSemFall,portMAX_DELAY);
            while (!sendMessage(10,0)) //send fall alarm message
            {
                  stablishConn();
            }
      }
}
```

Fitxer comm.c

```
#include "LPC17xx.h"
#include "FreeRTOS.h"
#include "uart.h"
#include "printf.h"
#include "wifly.h"
#include "config.h"
#include "configurations.h"
#include "comm.h"
#include <stdio.h>
#include <string.h>
static char *ssid = NETWORK_SSID;
static char *passwd = PASSWD;
static char *serverAdd = SERVER_ADDRESS;
static char *serverPort = SERVER_PORT;

/
*********************************************************************
 ** Function name:         debugInit
 **
 ** Description:      initializes debug UART
 **
 ** Parameters:            none
 ** Returned value:        none
*********************************************************************/

void debugInit()
{
      if(Print_Init(UART_DBG, BR_96))
        Print_Text(UART_DBG, (char *)"Debug port initialized: OK\n");
}

/
*********************************************************************
 ** Function name:         wiflyInit
 **
 ** Description:      initializes wifly UART
 **
 ** Parameters:            none
 ** Returned value:        none

*********************************************************************/
void wiflyInit()
{
      if(Wifly_Init(UART_WF, BR_96))
            Print_Text(UART_DBG, (char *)"Wifly port initialized: OK\n");
      else
            Print_Text(UART_DBG, (char *)"Wifly port initialized:
ERROR\n");
}

/*********************************************************************
```

```
 ** Function name:         joinSSID
 **
 ** Description:      joins wifi network
 **
 ** Parameters:            none
 ** Returned value:        none

*********************************************************************/
void joinSSID()
{
      if(Wifly_JoinSSID(UART_WF, ssid, passwd, WPA1))
            Print_Text(UART_DBG, (char *)"Connected to wifi AP: OK\n");
      else
            Print_Text(UART_DBG, (char *)"Connected to wifi AP: ERROR\n");

}

/
**********************************************************************
 ** Function name:         openConn
 **
 ** Description:      opens remote connection
 **
 ** Parameters:            none
 ** Returned value:        none

*********************************************************************/
uint8_t openConn()
{
      char addPort[20];          //address + port
      uint32_t success;

      strcpy(addPort,serverAdd);
      strcat(addPort," ");
      strcat(addPort,serverPort);

      success = Wifly_Open(UART_WF, addPort);
      if (!success)
            Print_Text(UART_DBG, (char *)"Connected to server: ERROR\n");
      else
            Print_Text(UART_DBG, (char *)"Connected to server: OK\n");

      return(success);
}

/
**********************************************************************
 ** Function name:         connInit
 **
 ** Description:      joins wifi network + open remote connection
 **
 ** Parameters:            none
 ** Returned value:        none
```

```
*********************************************************************/
void connInit()
{
      uint8_t tries = 0;
      uint8_t success=FALSE;
      char message[10];          //message to server

      while(!success)
      {
            Wifly_Reset();
            Delay_Ms(DELAY, 500); // min 160us

            joinSSID();
            while (!success && tries < 4)
            {
                  success =  openConn();
                  tries++;
            }
            tries=0;
      }
      sprintf(message, "");
      Print_T_CRLF(UART_WF,message);
}

/
*********************************************************************
 ** Function name:         stablishConn
 **
 ** Description:      initializes connection & identifies board until
success
 **
 ** Parameters:            none
 ** Returned value:        none

*********************************************************************/
void stablishConn(void)
{
      do
      {
            connInit();
      }
      while (!sendMessage(30,0)); //send identification message
}

/
*********************************************************************
 ** Task name:        sendMessage
 **
 ** Description:      sends a message of the indicated type
 **
 ** Parameters:            type of message
 **                        second parameter depends on type of message
 **                              - beat rate
 **                              - battery status
```

```
 **                                - identification
 **                                - (ignored if not relevant for current
message)
 ** Returned value:        status

*************************************************************************/
uint8_t sendMessage(uint8_t type, uint8_t param)
{
      char * ans;
      char message[20];               //message to server
      uint8_t status = TRUE;

      switch (type){
            /* beat rate message - 20
             * param = beat rate
             */
            case 20:
                  sprintf(message, "%d:%3d", type, param);
                  break;
            /* board identification message - 30
             * param : ignored
             */
            case 30:
                  sprintf(message, "%d:%3d", type, BOARD_NUM);
                  break;
            /* battery level message - 40
             * param = battery status
             */
            case 40:
                  sprintf(message, "%d:%3d", type, param);
                  break;
            /*
             * Other message types
             * 10 - Fall alarm
             * param : ignored
             */
            default:
                  sprintf(message, "%d", type);
                  break;
      }

      ans = Wifly_Send_And_Read_Response(UART_WF, message, 2000);

      if (strstr(ans, (char*)"OK")==NULL)
      {
            status=FALSE;
      }
      return status;
}

/
*************************************************************************
 ** Task name:        sendMessageCheckConn
 **
```

```
 ** Description:      sends a message and checks connection.
Reestablishes connection if necessary
 **
 ** Parameters:            type of message
 **                        second parameter depends on type of message
 **                              - beat rate
 **                              - battery status
 **                              - identification
 **                              - (ignored if not relevant for current
message)
 ** Returned value:        none

*********************************************************************/
void sendMessageCheckConn(uint8_t type, uint8_t param)
{
      if (!sendMessage(type,param))
      {
            stablishConn();
      }
}
```

Fitxer adc.c

```
/
*********************************************************************
 ** Task name:        adcInit()
 **
 ** Description:      initializes ADC
 **
 ** Parameters:            none
 ** Returned value:        none

*********************************************************************/

void adcInit(){
   LPC_SC->PCONP |= (1 << 12); //Start ADC

   LPC_SC->PCLKSEL0 |= (1<<24); //ADC clock
   LPC_SC->PCLKSEL0 |= (1<<25);
   LPC_ADC->ADCR |= ( 1 << 8 ); // PCLK scaling

   LPC_PINCON->PINSEL1 |=(1<<16);//p0.24 as AD0.1

   LPC_PINCON->PINSEL3 |=(1<<28);//p1.30 as AD0.4
   LPC_PINCON->PINSEL3 |=(1<<29);//

   LPC_PINCON->PINMODE1 |=(1<<15);  //desactivate pull-up & pull-down
   LPC_ADC->ADCR |=(1<<21); //ADC on
   LPC_ADC->ADCR |=(1<<16); //burst mode
}

/*********************************************************************
 ** Task name:        adcChannel(int ch)
```

```
 **
 ** Description:      sets active ADC channel
 **
 ** Parameters:            int ch          active channel
 ** Returned value:        none

*********************************************************************/

void adcChannel(int ch){  //Conversion channel
    LPC_ADC->ADCR   |= (1<<ch);
}
```

1.2. Servidor

Fitxer Server.java

```
*
 * Class: Server.
 *
 * Contains main function.
 */
public class Server
{
      private ArrayList<conBoard> cBoard;        //list of connected
boards
      private int port;                                 //server port
      private ServerGUI sg;                       //server GUI
      private boolean cont;
      private ServerSocket serverSocket;

      /*
       * Server constructor
       */
      public Server(int port)
      {
            this.port=port;
            this.sg=new ServerGUI(this);
            cBoard= new ArrayList<conBoard>();
            cont=true;
      }

      /*
       * Method: start
       *
       * Listens for connections
       * Creates a new conBoard for every new connection
       */
      public void start()
      {
            try
            {
                  serverSocket = new ServerSocket(port);
```

```
            while(cont)
            {
                  Socket socket = serverSocket.accept();
                  conBoard m = new conBoard(socket);
                  cBoard.add(m);
                  m.start();
            }
      }
      catch (IOException e)
      {
            String msg = e +"\n";
            System.out.println(msg);
      }
}

/*
 * Method: stop
 *
 * closes Server Socket
 */
public void stop()
{
      try {
            serverSocket.close();
      } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
      }
}

/*
 * Method: setDisposable(String id)
 *
 * conBoard id will be signaled as prepared to finish
 */
public void setDisposable(String id)
{
      for(conBoard c: cBoard)
      {
            if (c.id.equals(id))
            {
                  c.disposable=true;
                  break;
            }
      }
}

/*
 * main
 *
 * Creates a Server object and runs it
 */
public static void main(String[] args)
```

```
	{
		Server server = new Server(5432);
		server.start();
	}

	/*
	 * Inner class conBoard
	 *
	 * One is created for every connected board.
	 * Manages the connection with the board & prints info at
	 * the corresponding panel of the GUI.
	 */
	class conBoard extends Thread {
		private Socket socket;
		private String id;
		private boardPanel panel;
		private BufferedReader inFromClient;
		private DataOutputStream outToClient;
		private boolean connected=true;
		private boolean disposable=false;

		conBoard(Socket socket)
		{
			this.socket = socket;
			this.panel = new boardPanel(new FlowLayout(),sg);
			this.id="NNNNN";

			//we establish a 30 sec timeout on the socket
			try {
				this.socket.setSoTimeout(30000);
			} catch (SocketException e1) {
				// TODO Auto-generated catch block
				e1.printStackTrace();
			}

			try
			{
				inFromClient = new BufferedReader (new
InputStreamReader(
					this.socket.getInputStream()));
				outToClient = new
DataOutputStream(this.socket.getOutputStream());
			}
			catch (IOException e) {
				e.printStackTrace();
			}
		}

		/*
		 * Method: run()
		 *
		 * reads and responds to board messages
		 *
		 */
```

```
            public void run()
            {
                  String clientMessage;
                  while (!disposable)
                  {
                        if(connected)
                        {
                              try
                              {
                                    clientMessage = inFromClient.readLine();
                                    System.out.println(clientMessage);
                              }
                              catch (IOException e)
                              {
                                    //if timeout expires, board is
disconnected
                                    connected=false;
                                    clientMessage = "ERROR";
                              }
                              /* make the action according to board message
*/
                              switch(clientMessage.substring(0,2)) {
                              /*
                               * 10: fall
                               * 20: beat rate
                               * 30: board identification
                               * 40: battery level
                               */
                              case "10":
                                    answer();
                                    this.panel.setAlarm(true);
                                    this.panel.setReset(true);
                                    break;
                              case "20":
                                    answer();

      this.panel.setlFreqValue(clientMessage.substring(3,6));
                                    break;
                              case "30":
                                    answer();
                                    //mirar a veure si ja hi ha un board amb
aquest id
                                    for(conBoard c: cBoard)
                                    {
                                          if
(c.id.equals(clientMessage.substring(3,4)))
                                          {
                                                sg.removePanel(c.panel);
                                                cBoard.remove(c);
                                                c.connected=false;
                                                c.disposable=true;
                                                break;
                                          }
                                    }
```

```
                                    this.id=clientMessage.substring(3,5);
                                    this.panel.setId(this.id);
                                    sg.addPanel(this.panel);
                                    sg.revalidate();
                                    sg.repaint();
                                    break;
                              case "40":
                                    answer();
                                    if
(clientMessage.substring(3,4).equals("0"))
                                          this.panel.setBatteryLevel(0);
                                    else if
(clientMessage.substring(3,4).equals("0"))
                                          this.panel.setBatteryLevel(1);
                                    break;
                              }
                        }
                        else
                        {
                              this.panel.setlDiscon(true);
                              this.panel.setRemove(true);

                              while(true)
                              {
                                    try {
                                          Thread.sleep(100);
                                    } catch (InterruptedException e) {
                                          // TODO Auto-generated catch block
                                          e.printStackTrace();
                                    }
                                    if (disposable)
                                    {
                                          sg.removePanel(this.panel);
                                          sg.revalidate();
                                          sg.repaint();
                                          cBoard.remove(this);
                                          break;
                                    }
                              }
                        }
                  }
            }

            /*
             * Method: answer()
             *
             * answer with an OK
             *
             */
            private void answer(){
                  try
                  {
                        outToClient.writeBytes("OK\n");
                  }
```

```
                catch(IOException e)
                {
                      // TODO Auto-generated catch block
                      e.printStackTrace();
                }
          }
      }
}
```

Fitxer BoardPanel.java

```
public class boardPanel extends JPanel implements ActionListener{

      private JLabel lFreqLabel, lFreqValue, lAlarm, lId, lIdValue,
lDiscon, lBatteryLevel;
      private String id;
      private JButton bRemove, bReset;
      ServerGUI sGUI;

      public boardPanel(FlowLayout flowLayout, ServerGUI sGUI) {

      this.setComponentOrientation(ComponentOrientation.LEFT_TO_RIGHT);
            setLayout(null);
            this.sGUI=sGUI;

            lId = new JLabel("Placa num ");
            lIdValue = new JLabel("");
            lFreqLabel = new JLabel("Freqüència de batecs");
            lFreqValue = new JLabel("");
            lAlarm = new JLabel("CAIGUDA!!");
            lDiscon = new JLabel("PLACA DESCONNECTADA!!");
            lBatteryLevel = new JLabel("BATERIA BAIXA");

            bRemove = new JButton("Elimina");
            bReset = new JButton("Reset");

            bRemove.addActionListener(this);
            bReset.addActionListener(this);

            lId.setBounds(250,10, 120, 10);
            add(lId);
            lIdValue.setBounds(320,10, 120, 10);
            add(lIdValue);

            lFreqLabel.setBounds(50,40, 150, 20);
            add(lFreqLabel);
            lFreqValue.setBounds(180,38, 150, 20);
            lFreqValue.setFont(new Font("Times New Roman", Font.BOLD,
20));
            add(lFreqValue);

            lAlarm.setBounds(250,40, 150, 20);
            lAlarm.setForeground(Color.RED);
```

```
            lAlarm.setOpaque(true);
            lAlarm.setFont(new Font("Courier New", Font.ITALIC, 25));
            lAlarm.setVisible(false);
            add(lAlarm);

            lBatteryLevel.setBounds(350,40, 150, 20);
            lBatteryLevel.setForeground(Color.RED);
            lBatteryLevel.setOpaque(true);
            lBatteryLevel.setFont(new Font("Courier New", Font.ITALIC,
25));
            lBatteryLevel.setVisible(false);
            add(lBatteryLevel);

            bReset.setBounds(250,65, 90, 20);
            add(bReset);
            bReset.setEnabled(false);

            lDiscon.setBounds(250,90, 300, 20);
            lDiscon.setFont(new Font("Courier New", Font.BOLD, 20));
            lDiscon.setVisible(false);
            add(lDiscon);

            bRemove.setBounds(250,115, 90, 20);
            add(bRemove);
            bRemove.setEnabled(false);

            setBorder(BorderFactory.createLineBorder(Color.black));

      }
      public void setId(String id)
      {
            this.lIdValue.setText(id);
            this.id = new String(id);
      }
      public void setAlarm(boolean cond)
      {
            this.lAlarm.setVisible(cond);
      }
      public void setlFreqValue(String freq)
      {
            this.lFreqValue.setText(freq);
      }
      public void setlDiscon(boolean cond)
      {
            this.lDiscon.setVisible(true);
      }
      public void setRemove(boolean set)
      {
            this.bRemove.setEnabled(set);
      }
      public void setReset(boolean set)
      {
            this.bReset.setEnabled(set);
      }
```

```
	public void setBatteryLevel(int level)
	{
		if (level==0)
		{
			this.lBatteryLevel.setVisible(false);
		}
		else if (level==1)
		{
			this.lBatteryLevel.setVisible(true);
		}
	}

	@Override
	public void actionPerformed(ActionEvent arg0) {
		if (arg0.getSource()==bRemove)
		{
			this.sGUI.setDisposable(id);
			this.bRemove.setEnabled(false);
		}
		else
		{
			this.lAlarm.setVisible(false);
			this.bReset.setEnabled(false);
		}
	}

}
```

Fitxer ServerGUI.java

```
public class ServerGUI extends JFrame {

	private ArrayList<boardPanel> pMotes;
	Container cp;
	Server server;

	ServerGUI(Server server){
		super("Monitorització pacients");
		pMotes = new ArrayList<boardPanel>();
		setSize(600,400);
		cp = getContentPane();
		cp.setLayout(new BoxLayout(cp, BoxLayout.Y_AXIS));
		this.server=server;
		setVisible(true);

	}

	public void addPanel(boardPanel panel){
	  cp.add(panel);
	  pMotes.add(panel);
	}
```

```
	public void removePanel(boardPanel panel){
		cp.remove(panel);
		pMotes.remove(panel);
	}

	public void setDisposable(String id)
	{
		server.setDisposable(id);
	}
}
```