



industriales
etsii

Escuela Técnica
Superior
de Ingeniería
Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Estudio y diseño de un sistema de monitorización del ritmo cardíaco basado en un PSoC y un dispositivo móvil inteligente

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y
AUTOMÁTICA



Universidad
Politécnica
de Cartagena

Autor: Elías Hernández Gómez
Director: Juan Antonio López Riquelme
Codirector: Nieves Pavón Pulido

Índice

1. CAPÍTULO 1: INTRODUCCIÓN.....	1
1.1. CONTEXTO	1
1.2. OBJETIVOS	2
1.3. DESARROLLO DE LA MEMORIA	2
1.3.1. CAPÍTULO 1 - INTRODUCCIÓN	2
1.3.2. CAPÍTULO 2 – ESTADO DEL ARTE	3
1.3.3. CAPÍTULO 3 – DESCRIPCIÓN DEL SISTEMA	3
1.3.4. CAPÍTULO 4 – DESCRIPCIÓN DEL HARDWARE Y SOFTWARE DESARROLLADO	3
1.3.5. CAPÍTULO 5 – CONCLUSIONES	3
2. CAPÍTULO 2: ESTADO DEL ARTE.....	5
2.1. INTRODUCCIÓN.....	5
2.2. MONITORIZACIÓN CARDÍACA.....	6
2.2.1. HISTORIA DE LA MONITORIZACIÓN.....	6
2.2.2. SISTEMAS DE MEDIDA DEL RITMO CARDÍACO	7
2.2.2.1. ELECTROCARDIOGRAMA	7
2.2.2.2. PHOTOPLETHYSMOGRAMA.....	8
2.3. ARQUITECTURAS PSOC COMERCIALES.....	9
2.3.1. PSOC 4 BLE.....	9
2.3.2. CSR1010	10
2.3.3. NRF8001	11
2.4. SISTEMAS DE COMUNICACIÓN	12
2.4.1. WIFI.....	12
2.4.1.1. HISTORIA DE WIFI.....	12
2.4.1.2. CARACTERÍSTICAS DE WIFI	13
2.4.2. BLUETOOTH.....	14
2.4.2.1. HISTORIA DE BLUETOOTH	14
2.4.2.2. CARACTERÍSTICA DE BLUETOOTH	15
2.4.2.3. BLUETOOTH LOW ENERGY	16
2.5. DISPOSITIVOS MÓVILES INTELIGENTES	18
2.5.1. HISTORIA DE LOS DISPOSITIVOS MÓVILES INTELIGENTES	19
2.6. SISTEMAS OPERATIVOS PARA DISPOSITIVOS MÓVILES INTELIGENTES	22
2.6.1. ANDROID.....	22
2.6.1.1. HISTORIA DE ANDROID.....	22
2.6.1.2. VERSIONES DE ANDROID	23
2.6.2. IOS.....	24
2.6.3. WINDOWS PHONE.....	25

2.7. CONCLUSIONES	25
3. CAPÍTULO 3: DESCRIPCIÓN DEL SISTEMA	27
3.1. INTRODUCCIÓN	27
3.2. DESCRIPCIÓN DE LOS ELEMENTOS DEL SISTEMA DE MEDIDA DEL RITMO CARDÍACO	28
3.3. DESCRIPCIÓN DEL KIT DE DESARROLLO CY8CKIT-042-BLE DE CYPRESS	29
3.3.1. INSTALACIÓN DE PSOC CREATOR.....	30
3.3.2. CREACIÓN DE UN PROYECTO EN PSOC CREATOR.....	31
3.3.3. ESTRUCTURA DE UN PROYECTO EN PSOC CREATOR.....	32
3.3.4. PROGRAMACIÓN DEL PSOC DESDE PSOC CREATOR.....	34
3.4. DESARROLLO DE APPS PARA ANDROID	35
3.4.1. INSTALACIÓN DEL JAVA SDK.....	35
3.4.2. INSTALACIÓN DE ANDROID STUDIO Y ANDROID SDK.....	36
3.4.3. CONFIGURACIÓN DEL DISPOSITIVO MÓVIL PARA DEPURAR APLICACIONES..	37
3.4.4. CREACIÓN DE UN PROYECTO EN ANDROID STUDIO.....	38
3.4.5. ESTRUCTURA DE UN PROYECTO ANDROID.....	40
3.4.6. INSTALAR LA APLICACIÓN EN UN DISPOSITIVO CON ANDROID.....	42
3.5. CONCLUSIONES	43
4. CAPÍTULO 4: DESCRIPCIÓN DEL HARDWARE Y SOFTWARE DESARROLLADO	45
4.1. INTRODUCCIÓN	45
4.2. DESCRIPCIÓN DEL SISTEMA DE MEDIDA DEL RITMO CARDÍACO	46
4.2.1. ETAPA DE EMISIÓN Y RECEPCIÓN DE LUZ.....	46
4.2.2. ETAPA DE FILTRADO Y AMPLIFICACIÓN.....	47
4.2.3. ETAPA DE CONVERSIÓN ANALÓGICA-DIGITAL.....	49
4.2.4. MONTAJE DEL SISTEMA DE MEDIDA.....	50
4.3. DESCRIPCIÓN DE LA ARQUITECTURA IMPLEMENTADA EN EL PSOC	52
4.3.1. CREACIÓN DEL PROYECTO EN PSOC CREATOR.....	53
4.3.2. CONFIGURACIÓN DE LOS COMPONENTES UTILIZADOS EN LA ETAPA DE FILTRADO.....	53
4.3.3. CONFIGURACIÓN DE LOS COMPONENTES UTILIZADOS PARA CALCULAR EL RITMO CARDÍACO.....	56
4.3.4. CONFIGURACIÓN DEL COMPONENTE UTILIZADO PARA REALIZAR LA COMUNICACIÓN POR BLUETOOTH LOW ENERGY.....	60
4.3.5. ESQUEMA FINAL Y LISTA DE PINES LUZ.....	67
4.4. DESCRIPCIÓN DE LA APP DESARROLLADA EN ANDROID	68
4.4.1. CREACIÓN DEL PROYECTO EN ANDROID STUDIO.....	69
4.4.2. ESTRUCTURA DE LA APLICACIÓN DE ANDROID.....	69

4.4.3. PROGRAMACIÓN DEL ARCHIVO MAINACTIVITY.JAVA	70
4.4.4. PROGRAMACIÓN DEL ARCHIVO BLUETOOTHLESERVICE.JAVA.....	74
4.4.5. PROGRAMACIÓN DEL ARCHIVO BASEDEDATOS.JAVA.....	77
4.4.6. PROGRAMACIÓN DEL ARCHIVO MYCONTENTPROVIDER.JAVA	78
4.4.7. PROGRAMACIÓN DEL ARCHIVO ACTIVITY_MAIN.XML	80
4.4.8. PROGRAMACIÓN DEL ARCHIVO ANDROIDMANIFEST.XML	83
4.5. CONCLUSIONES.....	84
5. CAPÍTULO 5: ANÁLISIS DE RESULTADOS Y CONCLUSIONES	85
5.1. PRUEBAS Y ANÁLISIS DE RESULTADOS.....	85
5.2. CONCLUSIONES Y TRABAJO FUTURO	87
6. REFERENCIAS.....	89

Capítulo 1

Introducción

1.1 Contexto

Con la irrupción de los *smartphones* y de las comunicaciones inalámbricas, se ha desarrollado un gran ecosistema con el que se puede desarrollar casi cualquier aplicación. Actualmente, se puede desarrollar un producto utilizando diferentes componentes electrónicos, dotarlo de inteligencia mediante un microcontrolador, y utilizando estándares como son WiFi y Bluetooth, comunicar el diseño con dispositivos compatibles. Por lo tanto, con la actual expansión de los *smartphones*, la compatibilidad de un producto está prácticamente asegurada con los millones de dispositivos móviles inteligentes que hay repartidos por el mundo.

Esta amplia versatilidad a la hora de desarrollar productos ha propiciado que en los últimos años haya crecido el interés por desarrollar todo tipo de sensores y actuadores, que ya no necesitan de un sistema propietario para poder ser manejados, sino que el uso de un Smartphone es suficiente para controlar el producto. Esta apertura de la industria está propiciando que cada vez hagan falta menos componentes y estándares propietarios en un mercado, que cada vez aboga más por la simplificación y la estandarización.

Uno de los sectores que más está apostando por esta filosofía es el de los *wearables* y dispositivos de *e-health*, pequeños dispositivos electrónicos que son capaces de monitorizar parámetros físicos del cuerpo, poniendo esta información al alcance de un dispositivo móvil inteligente. Entre estos dispositivos cabe destacar los medidores de ritmo cardíaco, los dispositivos que monitorizan el sueño, los que cuentan pisadas o incluso

(monitorizando a un *smartphone* en vez de a una persona) los que monitorizan la batería y el centro de notificaciones del dispositivo y avisan de cualquier alerta que acontezca.

Por ello, resulta de interés comprender la arquitectura y el proceso que envuelve el desarrollo y el diseño de uno de estos sensores, tanto desde el punto de vista de la electrónica que se debe desarrollar para construir el sensor, como el proceso que implica adquirir el dato y su posterior transmisión a uno de estos dispositivos móviles inteligentes, para poder ser controlado por el usuario.

1.2 Objetivos

Este trabajo tiene como objetivo diseñar un sistema que permita monitorizar el ritmo cardíaco, apoyándose en un circuito diseñado especialmente para medir y acondicionar dicha señal con un dispositivo de tipo PSoC, que también será usado para la recopilación de datos, y una aplicación de Android para monitorizarlos. Por tanto, se plantean los siguientes objetivos para realizar el sistema propuesto:

- Estudiar el sensor necesario para medir el pulso cardíaco, seleccionando los componentes adecuados y construyendo dicho sensor.
- Estudiar el diseño de aplicaciones basadas en un PSoC.
- Diseñar la electrónica de acondicionamiento necesaria para incorporar el sensor en el PSoC.
- Estudiar la arquitectura de Android para implementar una aplicación sobre un dispositivo móvil inteligente.
- Realizar las pruebas de funcionamiento necesarias para validar la arquitectura propuesta.

1.3 Desarrollo de la Memoria

1.3.1 Capítulo 1 - Introducción

Este capítulo presenta la motivación de este trabajo, además de enumerar los objetivos del mismo y describir la memoria presentada.

1.3.2 Capítulo 2. Estado del Arte

En el Capítulo 2 se exponen los distintos sistemas de monitorización cardíaca disponibles, se presentan algunas arquitecturas PSoC comerciales disponibles para implementar el dispositivo propuesto en este trabajo, además de estudiar los distintos dispositivos móviles inteligentes presentes en el mercado, utilizados para recibir los datos de monitorización cardíaca, así como de sus sistemas operativos.

Dicho capítulo concluye enumerando los componentes elegidos para realizar el proyecto.

1.3.3 Capítulo 3. Descripción del Sistema

En este capítulo se explican en detalle los elementos seleccionados para realizar el diseño planteado, explicando en su caso las herramientas de software necesarias para poder desarrollar y depurar el proyecto desarrollado.

1.3.4 Capítulo 4. Descripción del Hardware y Software Desarrollado

En el cuarto Capítulo se describe en profundidad el caso práctico desarrollado, tanto a nivel de la arquitectura hardware, como software finales implementadas.

1.3.5 Capítulo 5. Análisis de Resultados y Conclusiones

En el quinto Capítulo se explican las conclusiones y valoraciones obtenidas tras desarrollar este proyecto.

Capítulo 2

Estado del Arte

2.1 Introducción

Para llevar a cabo este proyecto, es necesario estudiar los distintos sistemas de monitorización cardíaca, para poner en contexto cuál es el sistema implementado finalmente en este Trabajo Fin de Grado. Para ello, se estudiarán algunos aspectos históricos sobre la monitorización cardíaca, así como comprender el origen de estas señales producidas por el corazón.

Posteriormente se compararán las diferentes alternativas en cuanto a chips programables disponibles en el mercado para poder procesar estas señales producidas por el corazón. En este punto se estudiarán las distintas especificaciones de cada producto para poder elegir la alternativa más adecuada.

Lo siguiente será decidir el sistema de comunicación empleado para transmitir las pulsaciones registradas hacia un dispositivo móvil inteligente. En esta parte, no sólo habrá que tener en cuenta las especificaciones del sistema de comunicación a utilizar, sino su compatibilidad, tanto con el chip programable, como con el dispositivo móvil al que enviarán las lecturas realizadas.

Por último, habrá que elegir el dispositivo móvil adecuado al que poder mandarle la información, por lo que se van a mostrar las principales alternativas que existen actualmente a la hora de adquirir un dispositivo móvil inteligente.

2.2 Monitorización cardíaca

La monitorización cardíaca consiste en medir los impulsos generados por el corazón para bombear la sangre del cuerpo con respecto al tiempo. Esto permite observar el comportamiento de cada corazón y poder evaluar las condiciones físicas de una persona, así como poder detectar posibles enfermedades cardíacas.

2.2.1 Historia de la monitorización

Aunque los primeros humanos serían conscientes de que los latidos del corazón variaban cuando realizaban actividades físicas, las primeras descripciones sobre el tema que se dejaron escritas provienen del griego Herófilo, que vivió del año 335 al 280 antes de Cristo [1].

Herófilo vivió la mayor parte de su vida en Alejandría, donde escribió multitud de trabajos en los que publicaba sus descubrimientos sobre anatomía. En ellos demuestra que las venas transportan sangre, y que había notables diferencias entre las venas y las arterias, las cuales tenían un pulso que seguía cierto ritmo. Otro griego llamado Rufo de Éfeso fue el primero en reconocer que el pulso estaba causado por la contracción y la relajación del corazón.

Uno de los antiguos científicos griegos que más aportó sobre el tema fue Galeno, el cual escribió unos 8 tratados, describiendo la utilidad de la medida del pulso para diagnosticar enfermedades. Sin embargo, con la mejora en la medida del tiempo lograda en el siglo XVIII, las técnicas para medir el pulso mejoraron. Es el caso de John Floyer, un físico inglés que inventó un aparato para medir el pulso basado en un reloj portable, mediante el cual tabuló el pulso y la respiración bajo una serie de condiciones.

Cuando la capacidad de medir el tiempo siguió mejorando, pronto se empezaron a describir las fluctuaciones periódicas del pulso arterial. En 1733, Stephen Hales se dio cuenta de que la presión arterial y el intervalo entre latido y latido variaban durante los ciclos de la respiración. En 1847, Carl Ludwig utilizó un aparato desarrollado por él mismo para registrar las oscilaciones periódicas que se producían durante la respiración en las ondas de la presión arterial. Se dio cuenta de que durante la inspiración el pulso aumentaba, y mientras se espiraba el pulso se relajaba.

A finales del siglo XIX y principios del siglo XX, Willem Einthoven utilizó galvanómetros para medir los cambios producidos en la actividad eléctrica del corazón. Esto llevó a que se desarrollara y se estandarizaran los electrocardiogramas (ECG), los cuales permitían evaluar los cambios de ritmo producidos entre cada latido del corazón. En la década de 1960, los electrocardiogramas se hicieron muy populares en los ambulatorios,

en los cuales se dejaban registrando la actividad de los pacientes durante varios días. Esto hizo que pronto se buscara la relación entre las variaciones del ritmo cardíaco y las enfermedades de los pacientes [2].

A principios de 1970, con la llegada del procesamiento de las señales digitales, se hizo posible utilizar nuevas técnicas para analizar las variaciones más pequeñas percibidas en el ritmo cardíaco, lo que ha permitido avanzar enormemente en este campo.

2.2.2 Sistemas de medida del ritmo cardíaco

Desde que se empezó a estudiar las variaciones del ritmo cardíaco, se han ido desarrollando diferentes teorías respecto al motivo por el que se producen estas variaciones. Hoy en día se sabe que los cambios producidos en la frecuencia cardíaca reflejan las complejas interacciones entre las fibras nerviosas parasimpáticas, las fibras nerviosas simpáticas y las células marcapasos localizadas en el nódulo sinoauricular.

Para medir estas variaciones en el ritmo cardíaco se han desarrollado una serie de técnicas:

2.2.2.1 Electrocardiograma

Cada vez que el corazón late, las corrientes eléctricas involucradas en el proceso son irradiadas por toda la cavidad torácica, por lo que colocando una serie de electrodos dispuestos sobre la piel en esta zona, se pueden captar estas corrientes [3].

Para realizar la prueba correctamente, es necesario no haber ingerido medicamentos que puedan interferir en los resultados obtenidos. Tampoco se recomienda hacer ejercicio ni haber tomado agua fría justo antes de realizar la prueba. Durante la realización de la prueba no se produce ningún tipo de daño en el paciente, ya que el electrocardiograma no envía ningún tipo de electricidad a su cuerpo.

El electrocardiograma se utiliza para medir cualquier tipo de afección del corazón, para conocer el ritmo con el que palpita el corazón con objeto de detectar posibles anomalías, para conocer los efectos que en el corazón producen ciertos medicamentos o incluso el marcapasos, en caso de haberlo. Por lo tanto, suele ser el primer examen que se le realiza para detectar algún tipo de cardiopatía.

Los resultados que se esperan de un electrocardiograma para poder considerar que un corazón está sano, son una frecuencia cardíaca de entre 60 y 100 latidos por minuto, con un ritmo cardíaco constante y uniforme, es decir, casi periódico. Aun así, es posible

que el electrocardiograma no detecte algunos problemas cardíacos, ya que este tipo de problemas no producen alteraciones en la gráfica generada por el electrocardiograma.

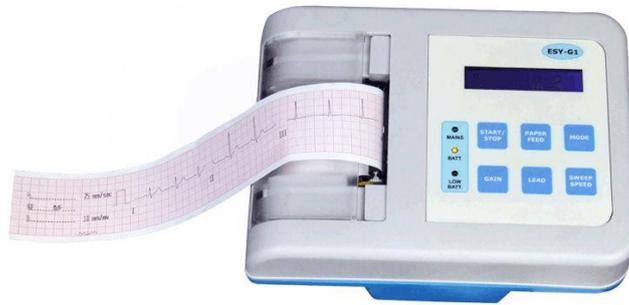


Ilustración 2-1. Electrocardiógrafo

2.2.2.2 Photoplethysmograma:

Cuando el corazón late, la sangre bombeada llega a los tejidos, produciéndose una variación en el volumen de los mismos. Por eso, si se detecta esta variación de volumen y se evalúa respecto al tiempo, se puede medir el ritmo cardíaco.

Para medir la variación del volumen de los tejidos, esta técnica se basa en la luz. Por ello, lo primero es elegir una zona del cuerpo que permita observar correctamente los tejidos cuando se le aplique luz, como es el lóbulo de la oreja o un dedo de la mano. A continuación hay que elegir uno de los siguientes métodos:

- **Reflexión:** Se coloca un transceptor junto a la piel de la zona elegida (ya sea un dedo o el lóbulo de la oreja). Entonces el transceptor emite un haz de luz, el cual penetra en la piel y rebota en el tejido, volviendo al transceptor con una cierta intensidad proporcional al volumen de sangre que haya en el tejido.
- **Transmisión:** Se coloca un emisor junto a la piel de la zona elegida (ya sea un dedo o el lóbulo de la oreja), y un receptor al otro lado del dedo o lóbulo de la oreja. Entonces el emisor emite un haz de luz, el cual penetra en la piel y atraviesa el tejido, llegando al receptor con una cierta intensidad proporcional al volumen de sangre que haya en el tejido.

Por tanto, cada vez que el corazón late, la sangre enviada por el mismo hacia los tejidos provocará que estos varíen su volumen, y que la luz que se ha hecho incidir sobre el tejido vea alterada su intensidad en función de la cantidad de sangre en la que deba

reflejarse o atravesar. Así, se puede detectar cada latido, así como generar una gráfica con las pulsaciones por minuto en tiempo real.

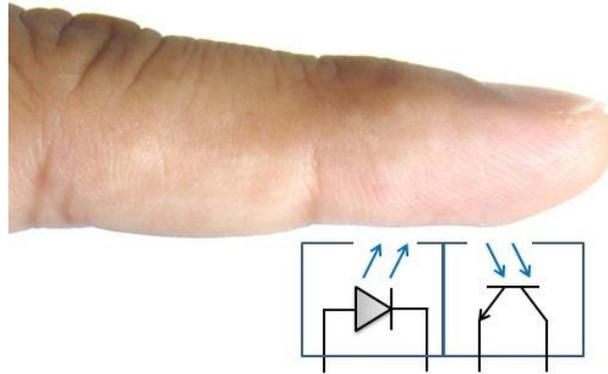


Ilustración 2-2. Esquema de funcionamiento de un photoplethysmógrafo

2.3 Arquitecturas PSoC comerciales

2.3.1 PSoC 4 BLE

Este dispositivo está desarrollado por la empresa Cypress [5]. Consta de una CPU con arquitectura ARM que funciona a 48 MHz. La memoria de programa es de unos 128 KB de tipo flash. Aunque pueda parecer que la potencia y la memoria son insignificantes al lado de los ordenadores actuales, hay que recordar que para el tipo de tarea que tiene que llevar a cabo el PSoC es más que suficiente. Además, al ser una velocidad de procesamiento tan “limitada”, se puede obtener un diseño final de bajo consumo.

Las dimensiones del encapsulado son 7 mm de largo, 7 mm de ancho y 0,6 mm de alto.

En cuanto a Bluetooth, el chip consta de un transceptor que transmite y recibe datos a 1 Mbps utilizando la banda de los 2,4 GHz, siendo compatible con la especificación 4.1 de Bluetooth. La capa de enlace proporciona protección AES de 128 bits, encriptación, y como novedades de Bluetooth 4.1, el chip es capaz de entrar en modo advertising en un bajo ciclo de trabajo. El controlador de banda base soporta, tanto modo maestro como modo esclavo, así como los protocolos L2CAP, ATT y SM. La API proporciona acceso a GATT, GAP y L2CAP. El PSoC soporta todos los perfiles del Bluetooth de baja energía definidos por el SIG.

En cuanto a los componentes físicos con los que cuenta el chip y que se pueden programar, cuenta con cuatro amplificadores operaciones en modo comparador, así como con un sensor de temperatura que se puede desactivar para ahorrar energía. A ello hay que

añadirle cuatro contadores, temporizadores y PWM para implementar la lógica necesaria o para controlar motores si fuera necesario. También cuenta con cuatro bloques digitales que permiten la conexión de periféricos para poder generar interrupciones en el chip.

No hay que olvidar los dos bloques dedicados a las comunicaciones serie, pudiendo implementar cada bloque los modos I²C, UART y SPI. Para complementar las comunicaciones, se incluyen 36 GPIO, que pueden programarse tanto para tareas analógicas como digitales.

En cuanto a las funciones especiales de este chip, en todos los pines soporta la conexión de una unidad LCD de 7 segmentos, así como de un detector capacitivo.

Este chip puede comprarse junto con un kit de desarrollo por sólo 49 €, por lo que es bastante asequible.



Ilustración 2-3. PSoC 4 BLE

2.3.2 CSR1010

Este chip cuenta con un procesador de 16 MHz con arquitectura RISC que ejecuta instrucciones de 16 bits. Su memoria se divide en 64 KB de memoria ROM para el almacenamiento permanente, y 64 KB de memoria RAM. Cabe destacar que es más que suficiente para las tareas que se llevarán a cabo en este trabajo.

Las dimensiones del encapsulado son 5 mm de largo, 5 mm de ancho y 0,6 mm de alto. Cuenta con 5 modos de funcionamiento, de los cuales 3 están dedicados al bajo consumo, siendo capaz de consumir unos 900 nA en el modo inactivo, por lo que es un chip muy adecuado, desde el punto de vista de la autonomía.

Tiene 12 líneas digitales bidireccionales que actúan tanto de entrada como de salida, pudiendo configurarse todas ellas con interrupciones. A su vez, estas líneas pueden

utilizarse para funciones alternativas, como son utilizarlas como interfaz SPI, UART, o para PWM. Además, el chip cuenta con 3 líneas de entrada o salida analógicas.

En cuanto a sensores incrustados en el chip, hay que decir que tiene embebido un sensor de temperatura con una precisión de ± 1 °C. Además, el chip proporciona interfaces serie de tipo UART, I²C y SPI, así como módulos PWM.

La conectividad Bluetooth utiliza la especificación 4.1. Puede operar, tanto en modo maestro como en modo esclavo, usando encriptación, GAP, L2CAP, y el resto de características del Bluetooth de baja energía, así como soportando la amplia gama de perfiles que permite la especificación Smart.

El chip puede adquirirse [6] junto con un kit de desarrollo por 99 €.



Ilustración 2-4. CSR1010

2.3.3 NRF8001

El NRF8001 incorpora un microprocesador de 16 MHz. Para el almacenamiento cuenta con una memoria no volátil para el almacenamiento permanente, y con una memoria de tipo RAM para el almacenamiento de datos volátiles, como pueden ser la información del perfil del cliente, o las direcciones de emparejamiento.

En cuanto a las capacidades de comunicación Bluetooth, incorpora los protocolos y perfiles definidos en la especificación 4.0, como son el manejo de códigos de errores, el uso del protocolo ATT, los perfiles GAP, GATT, el administrador de seguridad, L2CAP y DTM.

Las dimensiones del encapsulado son 5 mm de largo, 5 mm de ancho y 0,85 mm de alto.

Entre los sensores internos del chip se encuentran el sensor de temperatura, y el monitor de batería, que indica la tensión VDD conectada a los pines de alimentación del chip.

Cabe destacar que el fabricante ha especificado una estimación de la batería usando este chip. Por lo que suponiendo que se utilice con un perfil de uso de un sensor de ritmo cardíaco, el tiempo estimado de duración de una batería de 220 mAh es de unos 8 años.

El chip se puede adquirir [7] junto con un kit de desarrollo por 107,09 €.



Ilustración 2-5. NRF8001

2.4 *Sistemas de comunicación*

2.4.1 *WiFi*

WiFi es una tecnología de comunicación inalámbrica ampliamente extendida en el mercado, destinada a conectar todo tipo de dispositivos electrónicos en una red local sin la necesidad de cables físicos.

2.4.1.1 *Historia de WiFi*

Sus orígenes se remontan a 1971, cuando un grupo de investigadores diseñaron la primera red de área local inalámbrica (WLAN), a la cual se la denominó ALOHAnet [8]. Esta red utilizaba ondas de radio UHF para realizar las conexiones entre los ordenadores situados en las distintas islas hawaianas.

En 1985, la comisión de las comunicaciones de Estados Unidos introdujo las bandas ISM, las cuales designaban las bandas de frecuencia de uso no comercial en las que se debía trabajar para poder realizar comunicaciones inalámbricas de tipo Industrial, Científico y Médico [9].

Ya en 1991, NCR y AT&T crearon las bases de lo que posteriormente sería el estándar 802.11, el cual define la normativa referente al funcionamiento de la capa física y la capa de enlace de datos en las redes WLAN. En aquella época, las velocidades de transmisión de datos eran muy inferiores a las utilizadas actualmente, pero gracias a una tecnología desarrollada por John O'Sullivan para una aplicación relacionada con la astrofísica, se consiguió aumentar la velocidad de las transmisiones [10].

Fue en 1997 cuando el IEEE por fin anunció oficialmente el estándar 802.11, en torno al cual se creó en 1999 una asociación sin ánimo de lucro llamada WECA, la cual fomentaba el desarrollo de dispositivos electrónicos compatibles con el estándar 802.11 del IEEE.

En el 2002, WECA pasó a denominarse WiFi Alliance, y desde entonces cualquier producto que quiera llevar el logo que indique que el producto ha pasado el proceso de certificado de la WiFi Alliance, debe cumplir el estándar 802.11 marcado por el IEEE, los estándares de seguridad WPA y WPA2, y el estándar de identificación EAP.

2.4.1.2 Características de WiFi

El estándar 802.11, en el que se basa la tecnología WiFi ha ido experimentando modificaciones con el paso de los años, ofreciendo nuevas características que se adecuan a las nuevas necesidades de los dispositivos que incorporan la tecnología WiFi, las principales versiones de este protocolo son:

- IEEE 802.11b: trabaja en la banda de los 2,4 GHz y alcanza una velocidad de 11 Mbps. El espectro electromagnético de este protocolo se divide en 14 canales que se superponen, dando opción a que convivan distintas redes WiFi en el mismo espacio.
- IEEE 802.11g: trabaja en la banda de los 2,4 GHz, alcanza una velocidad de 54 Mbps. El espectro electromagnético de este protocolo se divide en 14 canales que se superponen.
- IEEE 802.11n: trabaja en la banda de los 2,4 GHz y opcionalmente en la de 5 GHz, alcanza una velocidad de entre 54 Mbps y 600 Mbps. El espectro electromagnético de este protocolo se divide en 14 canales que se superponen.
- IEEE 802.11ac: trabaja en la banda de los 5 GHz, alcanza una velocidad de 1300 Mbps, aunque al trabajar en una frecuencia mayor, la señal pierde un

poco de alcance. El espectro electromagnético de este protocolo se divide en 23 canales que no se superponen.

Aunque el espectro electromagnético del WiFi se divide en varios canales, las distintas redes WiFi presentes en la misma zona pueden compartirlos sin interferirse entre ellas, ya que aunque por los canales haya más paquetes de datos, como cada red tiene su propio SSID (*Service Set Identifier*), el cual consiste en 32 bytes que identifican una determinada red, los dispositivos sólo tienen que ignorar los paquetes que no vayan dirigidos al SSID de la red a la que pertenecen.

En cuanto al alcance de la señal, un punto de acceso que cumpla con los estándares 802.11b y 802.11g puede alcanzar unos 100 metros de alcance, por lo que el WiFi se convierte en un buen sustituto de las comunicaciones cableadas en áreas locales.

La seguridad de los datos transmitidos inalámbricamente corre a cargo de los estándares de encriptación de datos WEP, WPA y WPA2. WPA utiliza el protocolo TKIP, el cual refuerza la seguridad proporcionada por WEP, aunque se ha comprobado que este protocolo también presenta vulnerabilidades, por lo que finalmente se recomienda utilizar WPA2, el cual utiliza AES (*Advanced Encryption Standard*) para proteger la red.

Por sus características, WiFi está presente en casi cualquier producto del mercado que necesite comunicarse inalámbricamente con otro dispositivo electrónico. Es el caso de los ordenadores de sobremesa, los portátiles, los smartphones, las tabletas, las Smart TV, las neveras, las bombillas inteligentes y todo tipo de productos que necesitan acceder a una red local para transmitir datos inalámbricamente. Es compatible con prácticamente todos los sistemas operativos actuales, tanto a nivel de escritorio, como son Windows, Mac OS X y Linux, como a nivel de dispositivos móviles como pueden ser Android, iOS y Windows Phone.

2.4.2 Bluetooth

La tecnología Bluetooth es un estándar que permite intercambiar datos de forma inalámbrica entre dos dispositivos situados a poca distancia entre sí. Es una tecnología muy expandida actualmente.

2.4.2.1 Historia de Bluetooth

En 1994, la empresa Ericsson decidió reemplazar las comunicaciones cableadas que utilizaban el estándar RS-232 por comunicaciones inalámbricas que utilizaran la radiofrecuencia para transmitir datos [11]. Otras compañías también habían pensado en la idea de realizar conexiones entre dispositivos de forma inalámbrica, pero pronto se dieron

cuenta de que si no se ponían de acuerdo en utilizar un protocolo de comunicaciones inalámbricas estandarizado, no sería posible que los dispositivos de una compañía se conectaran con los dispositivos de otra compañía que utilizara un estándar distinto [12].

Por lo que en 1998, Ericsson, Intel, Nokia, IBM y Toshiba crearon el Bluetooth SIG (*Bluetooth Special Interest Group*), una organización sin ánimo de lucro que se encarga de desarrollar el estándar Bluetooth, y de licenciar la tecnología Bluetooth y su marca a los fabricantes. Actualmente el Bluetooth SIG cuenta con más de 20000 miembros.

En cuanto al nombre de esta tecnología, proviene del rey escandinavo Harald Blåtand, el cual unió en un solo reino a las tribus de Dinamarca, Noruega y Suecia. En 1997, Jim Kardach, un trabajador de Intel, acuñó el nombre de Bluetooth inspirándose en la unificación que hizo este rey escandinavo, como metáfora de la unión inalámbrica entre dispositivos que posibilitaba esta nueva tecnología.

2.4.2.2 Características de Bluetooth

Bluetooth trabaja en una frecuencia situada entre los 2400 y los 2483,5 MHz de la banda ISM. Emplea una técnica de modulación en espectro ensanchado en la que la señal se emite sobre una serie de radiofrecuencias, sobre las cuales se va saltando de forma sincronizada con el transmisor. Esto evita que un receptor no autorizado sea capaz de escuchar la transmisión, ya que recibirá una señal ininteligible.

A lo largo de su historia, el estándar Bluetooth ha ido pasando por diferentes versiones, las cuales han ido dotando de mayor velocidad de transmisión y de nuevas características a los dispositivos que lo incorporaban. Las principales versiones son:

- Bluetooth 1.2: velocidad de transmisión de 1 Mbps.
- Bluetooth 2.0 + EDR: velocidad de transmisión de 1Mbps. Se introdujo el EDR (*Enhanced Data Rate*) para transmitir datos más rápidamente.
- Bluetooth 3.0 + HS: velocidad de transmisión de 24 Mbps y mejoras en la seguridad.
- Bluetooth 4.0: velocidad de transmisión de 24 Mbps. Introducción del bajo consumo.
- Bluetooth 4.1: es una modificación a nivel de software enfocada a mejorar el bajo consumo introducido en la especificación 4.0, por lo que no es necesario actualizar el hardware de los dispositivos que cumplan este estándar.

- Bluetooth 4.2: introduce características para interactuar con los dispositivos del IoT (*Internet of Things*).

Bluetooth permite establecer medidas de seguridad a la hora de realizar una conexión, requiriendo la introducción de una clave de seguridad para que los dos dispositivos se queden emparejados correctamente.

2.4.2.3 Bluetooth Low Energy

Es una tecnología perteneciente a la especificación 4.0 de Bluetooth, enfocada en el bajo consumo. Por ello, está pensada para utilizarse en dispositivos que ejecuten tareas muy concretas de recolección de datos, pudiendo entrar en un modo de muy bajo consumo entre cada conexión.

Debido a esto, con Bluetooth de baja energía se pierden ciertas posibilidades que se tenían con la especificación clásica, como transmitir audio o archivos de un tamaño considerable. No obstante, la ventaja obtenida con el aumento de la autonomía de la batería, introduce a los dispositivos que incorporen esta tecnología en un nuevo segmento repleto de posibilidades de sensorización que antes era imposible que existiese.

Historia de Bluetooth Low Energy

Sus orígenes se remontan al año 2001, cuando Nokia comenzó a investigar una tecnología inalámbrica basada en Bluetooth que consumiera menos energía. En 2004, Nokia publicó los resultados de su investigación, y a la tecnología que desarrolló le dio el nombre de *Bluetooth Low End Extension*.

En 2006, tras colaborar con la Unión Europea en el proyecto MIMOSA, la tecnología de Nokia se presentó con el nombre de Wibree. Posteriormente en 2007, el Bluetooth SIG acordó con Nokia la integración de Wibree en la próxima especificación del estándar Bluetooth como una tecnología de baja energía, conocida como Bluetooth Smart, hasta que finalmente en 2010 se terminó de integrar la tecnología Bluetooth Smart dentro de la pila de protocolos de la especificación 4.0 del Bluetooth.

Retrocompatibilidad

Al ser una especificación que introdujo un nuevo paradigma en el modo en el que la tecnología Bluetooth realizaba las comunicaciones, no es compatible con las anteriores versiones del bluetooth. En concreto, Bluetooth SIG ha definido una serie de distinciones en función del tipo de protocolo que emplean los dispositivos para saber los dispositivos con los que son compatibles:

- Bluetooth: son los dispositivos que utilizan la versión clásica del Bluetooth, por lo que sólo son compatibles con dispositivos que utilicen la versión clásica del protocolo.
- Bluetooth SMART: son los dispositivos que utilizan la nueva versión del protocolo, por lo que sólo son compatibles con dispositivos que soporten la nueva versión del protocolo.
- Bluetooth SMART READY: son dispositivos que soportan la versión clásica y la versión nueva del protocolo, por lo que son compatibles tanto con dispositivos que soporten ambas versiones del protocolo.

GAP

GAP significa *Generic Access Profile*, controla las conexiones y el modo *advertising* de los dispositivos. Para que pueda realizarse una conexión, los dos dispositivos a emparejar deben cumplir dos roles:

- Dispositivo Central: escanea los paquetes de *advertising* y responde al periférico para iniciar la conexión.
- Periférico: manda paquetes de *advertising* para que los dispositivos centrales puedan encontrarlo y conectarse a él.

Habitualmente, el dispositivo que ejerce el rol de periférico suele ser el que recolecta información mediante algún sensor, como es el caso de un ratón para controlar el movimiento del cursor en la pantalla. El elemento que ejerce el rol de dispositivo central suele ser un smartphone o un ordenador, ya que permiten que usando la interfaz gráfica, permiten configurar los parámetros y las acciones del dispositivo periférico.

GATT

GATT significa *Generic Attribute Profile*, es una especificación para enviar y recibir datos. Una vez que los dos dispositivos (el dispositivo central y el periférico) se han conectado entre sí, hay que establecer los roles de ambos a la hora de establecer la comunicación para intercambiar datos. Los dos roles disponibles son:

- Servidor: suele ser el dispositivo configurado como periférico, el cual recoge los datos de los sensores y se los envía al cliente. El servidor propone al cliente un intervalo de conexión, para que el cliente trate de conectarse cada vez que pase ese intervalo de conexión para ver si hay nuevos datos disponibles.

- Cliente: es el que realiza las peticiones al servidor para leer o para escribir datos en él.

Los datos a transmitir entre los dos dispositivos se organizan en la siguiente jerarquía:

- Perfiles: son un conjunto de servicios, los cuales agrupados cumplen una determinada función, por lo que se han estandarizado una serie de perfiles que pueden ser implementados por distintos fabricantes para que sus productos sean compatibles entre sí. Por ejemplo, se puede encontrar un perfil dedicado a medir la temperatura corporal, el cual ya ha quedado completamente definido por el estándar desarrollado por Bluetooth SIG, y así si distintos fabricantes implementan ese perfil, sus productos serán compatibles entre sí, sin necesidad de modificar ni readaptar el software de esos dispositivos.
- Servicios: contienen un conjunto de características.
- Características: son la unidad básica de la jerarquía de datos, contienen por fin el dato que queremos transmitir. Pueden ir acompañadas de una serie de descriptores, que se utilizan para por ejemplo definir la unidad utilizada para medir la característica, o para activar las notificaciones de esa característica, para que sea el servidor el que notifique al cliente cada vez que la característica haya cambiado, en vez de ser el cliente el que tenga que estar comprobando constantemente el valor de la característica.

Los perfiles y los servicios están identificados por un UUID, los cuales tienen una longitud de 16 bits para los servicios que han sido predefinidos por el Bluetooth SIG, y una longitud de 128 bits para los servicios personalizados. Esto permite que los fabricantes puedan crear sus propios servicios, expandiendo las posibilidades de los dispositivos.

2.5 Dispositivos móviles inteligentes

Hoy en día es habitual contar con uno o varios dispositivos denominados inteligentes (Smartphone). Son dispositivos que se conectan a Internet, y que son capaces de ejecutar tareas que hasta no hace mucho estaban reservadas para los ordenadores personales.

Su implantación ha supuesto un salto enorme en la forma de comunicarnos con el mundo, y han abierto un mar de nuevas posibilidades, que han permitido que mientras que, por ejemplo, antes hiciera falta varios aparatos distintos para llevar a cabo cada tarea, ahora sólo es necesario un único dispositivo.

2.5.1 Historia de los dispositivos móviles inteligentes

En 1992 IBM lanzó el Simon Personal Communicator [13]. La pantalla era de color blanco y negro, contaba con un elemento para realizar pulsaciones sobre la pantalla táctil, y se cargaba mediante una base de carga. Era capaz de enviar y de recibir emails y faxes. Su precio de venta era de unos 1099 \$.



Ilustración 2-6. IBM Simon

Nokia se introdujo en este segmento de dispositivos en 1996, lanzando el Nokia 9000 Communicator. Este dispositivo contaba con un teclado QWERTY y una pantalla en blanco y negro. Contaba con un procesador Intel de 24 MHz, y era capaz de enviar y recibir emails y faxes, de navegar por la web, y de procesar textos y hojas de cálculo.



Ilustración 2-7. Nokia 9000 Communicator

Aunque estas compañías habían fabricado dispositivos que se pueden considerar precursores de los dispositivos móviles inteligentes, el título de primer smartphone se lo lleva el GS 88, un dispositivo lanzado por Ericsson en 1997. Este dispositivo contaba con una pantalla táctil manejada mediante un *stylus*.

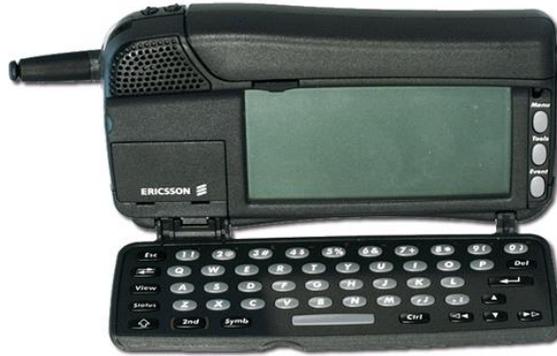


Ilustración 2-8. Ericsson GS 88

Llegados a este punto, otras compañías comenzaron a lanzar sus conceptos de smartphones, como Qualcomm con sus Palm. En 1999 Research In Motion (RIM) lanzó la BlackBerry 850.

Ya en el año 2000, los principales smartphones disponibles en el mercado contaban con Symbian como sistema operativo en el caso de Nokia, Palm OS en el caso de Qualcomm, BlackBerry OS en el caso de RIM, y Windows Mobile en el caso de Microsoft. Estos dispositivos contaban con conexión a Internet, podían enviar emails, navegar por la web y realizar algunas tareas necesarias en el mundo empresarial.

El diseño de estos dispositivos era muy variado, y cada compañía apostaba por su estilo propio, e incluso con varios diseños diferentes dentro de la misma compañía. Había dispositivos con teclado, sin teclado, con teclado deslizante, con pantallas giratorias, etc. Aun así, los smartphones que contaban con pantallas táctiles utilizaban una tecnología resistiva, por lo que hacía falta utilizar un *stylus* para moverse por el dispositivo, o realizar cierta presión con el dedo.

Aunque todos estos dispositivos eran considerados smartphones, su uso estaba enfocado principalmente al mercado empresarial. Pero con los años, Internet se fue popularizando en los hogares, y cada vez más gente empezaba a utilizar y a comprender las posibilidades de utilizar la red de redes, por lo que cuando se les presentara la posibilidad de utilizar un smartphone, le verían más utilidad de la que hasta ahora sólo se le encontraba en el sector empresarial.

Por eso, cuando en 2007 Apple lanzó el iPhone, lo hizo creando un dispositivo con una interfaz de usuario revolucionaria, la cual era muy sencilla de utilizar. El dispositivo contaba con una pantalla capacitiva, por lo que ya no hacía falta utilizar un *stylus* para manejarlo, y se alcanzaba una sensibilidad y una precisión nunca vistas a la hora de interactuar con la pantalla. Esto, unido a sus posibilidades multimedia, su capacidad de conectarse a Internet, y la introducción de una tienda de aplicaciones, que permitía ampliar

el ecosistema de aplicaciones del dispositivo, supuso un punto de ruptura en la historia de los dispositivos móviles inteligentes.



Ilustración 2-9. iPhone de primera generación

Tras el lanzamiento del iPhone, Microsoft continuó con Windows Mobile, y RIM seguía teniendo un número elevado de ventas de BlackBerry. Pero en 2008, un nuevo contendiente apareció en el mercado, Google lanzó Android, un sistema operativo basado en Linux, el cual fue integrado en los smartphones de diversos fabricantes.

Con el tiempo, la tendencia en el diseño de los smartphones ha seguido el camino iniciado por Apple, y la mayoría de los dispositivos móviles inteligentes han abandonado el uso del teclado físico, y optan por utilizar dispositivos rectangulares con pantallas táctiles cada vez mayores para manejar el dispositivo, a excepción de algunos botones para encender el dispositivo o para controlar el volumen.

Actualmente los smartphones se encuentran completamente integrados en nuestra sociedad, y se utilizan día a día por millones de personas. Por lo que una vez que han sido aceptados, han surgido otros dispositivos inteligentes con funciones similares a las de los smartphones, pero con formas distintas o usos más específicos. Es el caso de las tablets, las cuales emplean los mismos sistemas operativos que los smartphones, pero utilizan una pantalla de mayor tamaño, por lo que son ideales para consumir contenido multimedia, o para aprovechar el teclado en pantalla de mayor tamaño para redactar documentos de ofimática.

El último segmento importante a comentar es el de los gadgets, los cuales hacen referencia a cualquier tipo de dispositivo móvil inteligente, sin la necesidad de que sean teléfonos móviles. En esta categoría entran desde los wearables, hasta los dispositivos de e-health, así como los recientes dispositivos de tipo smartwatch. Son dispositivos que son

capaces de conectarse con su entorno y desarrollar alguna función, por lo que son perfectos para cubrir las necesidades presentes y futuras del mundo actual.

2.6 Sistemas Operativos para dispositivos móviles inteligentes

2.6.1 Android

Android es un sistema operativo creado en octubre de 2003 por Andy Rubin, Rich Miner, Nick Sears y Chris White. Está basado en el kernel Linux [14]. Inicialmente estaba destinado a las cámaras digitales, pero finalmente se enfocó para ser usado en smartphones, y poder así rivalizar con Symbian y con Windows Mobile.

Actualmente es el sistema operativo móvil más utilizado en el mundo, por lo que cuenta con un amplio ecosistema de aplicaciones y dispositivos compatibles con él. Se puede encontrar en smartphones, tablets, smartwatches e incluso en los automóviles.

Debido a esto, soporta una gran cantidad de sensores que se pueden encontrar en todo el abanico de dispositivos, como acelerómetros, giroscopios, chips de WiFi, Bluetooth, NFC, etc.

2.6.1.1 Historia de Android

En julio de 2005, Google adquirió Android por 50 millones de dólares, y se hizo cargo del desarrollo del mismo. El 5 de noviembre de 2007 se creó la Open Handset Alliance, un consorcio de compañías compuesto por Google, fabricantes como HTC, Sony y Samsung, operadoras como T-Mobile, y fabricantes de chips como Qualcomm y Texas Instruments, con el objetivo de desarrollar estándares abiertos para los dispositivos móviles.

El 22 de octubre de 2008 se lanzó al mercado el que oficialmente se considera el primer smartphone con Android, el HTC Dream, el cual comenzó la andadura de este sistema operativo. Debido a que Android tiene una licencia Apache, la cual es *open source*, los fabricantes pueden coger el código fuente y modificarlo a su gusto, por lo que en los comienzos de Android, cada fabricante modificaba la interfaz de usuario de acuerdo a su estilo. Esto era un factor diferenciador entre fabricantes, pero también se ha demostrado que debido a la falta de optimización, muchas veces suponía un lastre para el rendimiento de los dispositivos.



Ilustración 2-10. HTC Dream

En 2010, Google lanzó la línea de teléfonos Nexus, la cual fabricaba con el apoyo de un fabricante, para enseñar el camino que deberían seguir los fabricantes para ofrecer una experiencia de Android pura. Estos dispositivos ofrecían un rendimiento espectacular, y solían venir acompañados de la última versión de Android disponible.

2.6.1.2 Versiones de Android

Android ha ido recibiendo actualizaciones todos los años. El sistema de versiones utilizado por Google es un sistema muy utilizado en el ecosistema de Linux, que consiste en seguir un determinado patrón para nombrar a cada versión. En el caso de Android, cada versión nueva lleva el nombre de un dulce, y se va siguiendo un orden alfabético, llegando así a la siguiente lista de versiones:

- Android 1.5 Cupcake
- Android 1.6 Donut
- Android 2.0 – 2.1 Eclair
- Android 2.2 Froyo
- Android 2.3 Gingerbread
- Android 3.0 Honeycomb
- Android 4.0 Ice Cream Sandwich
- Android 4.1 – 4.3 Jelly Bean
- Android 4.4 KitKat
- Android 5.0 – 5.1 Lollipop

- Android 6.0 Marshmallow

Aunque el sistema ha mejorado mucho con el tiempo, como cada fabricante adapta la interfaz de Android, es habitual que los dispositivos se queden obsoletos al año y medio, ya que cada nueva versión de Android debe ser adaptada por el fabricante para los distintos modelos que posee, y posteriormente esta nueva versión puede ser modificada también por la operadora que ha suministrado el terminal al cliente, por lo que muchas veces suele transcurrir bastante tiempo desde que la nueva versión sale al mercado hasta que los usuarios de un dispositivo lo disfrutan. Esto ha llevado a la denominada “fragmentación”, la cual pese al excelente sistema en el que se ha convertido Android, supone uno de sus principales puntos negativos.

2.6.2 iOS

iOS es un sistema operativo móvil creado por Apple Inc. para ser utilizado en los dispositivos fabricados por la misma casa, como el iPhone y el iPad [15]. Debido a su desarrollo propietario y a su uso exclusivo en productos de Apple, no se ha extendido tanto como Android, pero aun así los dispositivos con iOS cuentan con una importante presencia en el mercado, y casi cualquier aplicación desarrollada para Android se encuentra también en iOS, por lo que se puede considerar una de las plataformas móviles más importantes.

Fue lanzado en 2007 para el iPhone, pero con su constante desarrollo se ha expandido a otros dispositivos como el iPad, el iPod y el Apple TV. A nivel interno comparte algunos frameworks con OS X, su hermano de escritorio, pero aunque cada sistema operativo ha seguido su desarrollo por su lado, en los últimos tiempos se está tendiendo a una cierta integración entre ambos.

A nivel de uso, su apariencia y comportamiento es similar al encontrado en Android, por lo que está pensado para ser utilizado en pantallas táctiles. Soporta diversos sensores como acelerómetros y giroscopios, así como un sensor de huellas dactilares, introducido en la última versión del sistema operativo.

Una de sus ventajas respecto a Android, es que Apple se encarga del desarrollo del sistema operativo (software) y del teléfono sobre el que se ejecuta (hardware), por lo que al controlar ambos componentes, se consigue una integración casi perfecta entre ambos, consiguiendo un gran nivel de fluidez en la interfaz gráfica sin necesidad de contar con un hardware muy potente.

2.6.3 Windows Phone

Windows Phone es un sistema operativo para dispositivos móviles desarrollado por Microsoft como reemplazo del antiguo Windows Mobile [16]. Mientras que Windows Mobile se centraba en el mercado empresarial, el nuevo Windows Phone se ha enfocado en un mercado más amplio, como han hecho iOS y Android.

La interfaz de usuario ha sido creada desde cero utilizando el nuevo lenguaje visual conocido como “Modern UI”, el cual se basa en la interacción del usuario por una serie de “Tiles”. El problema es que aunque la interfaz fuera novedosa y fácil de utilizar, Microsoft necesitaba un fabricante que estuviera dispuesto a llevar su sistema operativo. Este socio lo encontró en Nokia, la cual no había conseguido grandes resultados desde que comenzó el auge de los smartphones.

Por eso, en 2011 se anunciaron los primeros smartphones con el sistema operativo de Microsoft, el Lumia 800 y el Lumia 710. En 2013 Microsoft compró la división móvil de Nokia, pasando a producir ellos mismos el hardware. Aunque aun así, a día de hoy no han conseguido una gran cuota de mercado, habiendo llegado tarde a la lucha que ahora mismo mantienen Android e iOS por la cuota mercado.

2.7 Conclusiones

Tras haber presentado las distintas alternativas que se pueden encontrar en el mercado para llevar a cabo este Trabajo Fin de Grado, se ha hecho una selección con los elementos más adecuados para llevar a cabo este diseño:

- Photoplethysmograma por reflexión: ésta ha sido la técnica elegida para medir el ritmo cardíaco, debido al bajo coste que supone integrar en un circuito los sensores necesarios para construir este tipo de sensor, y a la relativa sencillez del funcionamiento del mismo.
- PSoC 4 BLE: este dispositivo será el encargado de procesar las señales producidas por el sensor y de transmitir las por Bluetooth a un dispositivo móvil inteligente. Es una alternativa adecuada debido al bajo coste del dispositivo, y al excelente soporte que ofrece Cypress Semiconductor, la empresa que lo desarrolla. Además, permite incorporar electrónica de acondicionamiento sobre el propio dispositivo.
- Bluetooth Low Energy: esta tecnología de comunicación encaja perfectamente con las necesidades de este proyecto, ya que permite la comunicación en distancias cortas para transmitir información entre el

PSoC y el dispositivo móvil inteligente, y proporciona un bajo consumo, que permite mantener al sistema el máximo tiempo posible activo sin la necesidad de reemplazar las baterías.

- Android: es la plataforma elegida para desarrollar la aplicación que permita al dispositivo móvil inteligente monitorizar los datos enviados por el PSoC. Su alta presencia en el mercado y su compatibilidad con millones de dispositivos, así como su naturaleza abierta y posibilidades de desarrollo, junto con una extensa documentación y ayuda proporcionada por la comunidad, han inclinado la balanza hacia otras alternativas descritas en este capítulo.

Capítulo 3

Descripción del Sistema

3.1 Introducción

En el capítulo anterior se han presentado las distintas alternativas que se pueden utilizar para llevar a cabo este proyecto, comentando los distintos sistemas de monitorización del ritmo cardíaco, los distintos kits de desarrollo disponibles en el mercado para digitalizar el ritmo cardíaco, los sistemas de comunicación inalámbrica disponibles para enviar los datos, y los diferentes sistemas operativos para dispositivos móviles, utilizados como interfaz del sistema.

En este capítulo se explican en detalle las alternativas elegidas para este trabajo, por lo que en primer lugar se explicarán los componentes seleccionados para realizar la sensorización. Después, se explicará cómo configurar el kit de desarrollo, así como el entorno de desarrollo de software utilizado para programar el PSoC. Por último, se explicará el proceso para desarrollar aplicaciones para dispositivos móviles basados en el sistema operativo Android.

3.2 Descripción de los elementos del sistema de medida del ritmo cardíaco

Dado que para medir el ritmo cardíaco se ha elegido un sistema de medida basado en un plethystógrafo, la electrónica a utilizar deberá ser acondicionada al tipo de señal proporcionada por el sensor seleccionado.

El componente principal de un plethysmógrafo es el transmisor de la señal luminosa y el receptor de la misma. Cuando el transmisor emita una señal luminosa, rebotará en el tejido sobre el que se proyecta, e incidirá sobre el receptor, el cual recogerá la intensidad de la misma. Como el receptor es un transistor, en función de la intensidad recibida se obtendrá una tensión distinta en el colector del mismo, por lo que en función de la cantidad de sangre que circule por el tejido sobre la que incide la luz, el receptor va a generar una señal que indicará las variaciones en el flujo sanguíneo del tejido que se está midiendo. Debido a ello, tanto el transmisor como el receptor deberán estar correctamente situados sobre el tejido de la persona para evitar posibles ruidos en la señal.

La señal medida mediante el receptor será una señal analógica, por lo que habrá que filtrarla adecuadamente para evitar los ruidos generados por los propios componentes electrónicos utilizados. Este filtrado va a consistir en una serie de filtros paso banda compuestos por resistencias, condensadores y amplificadores operacionales.

Una vez filtrada la señal, habrá que amplificarla a unos niveles adecuados para poder ser tratados por el PSoC, por lo que se utilizarán una serie de etapas de amplificación compuestas por amplificadores operacionales.

Tras haber obtenido una señal analógica con unos niveles adecuados y sin demasiado ruido, hay que digitalizar dicha señal para poder detectar los latidos del corazón. Por lo que mediante un amplificador operacional en modo comparador, se comparará la señal analógica, ya filtrada, con una tensión de referencia. Esta tensión de referencia se fijará en unos cuantos milivoltios por encima del cero. De este modo, pequeñas variaciones por encima de 0 V serán filtradas, a su vez que cada vez que se produzca una pulsación, el amplificador operacional generará una señal a nivel alto, y el resto del tiempo una señal a nivel bajo.

La señal descrita anteriormente será filtrada usando elementos del propio PSoC, así como transmitida al dispositivo móvil inteligente, usando el módulo de comunicación Bluetooth Low Energy que incorpora esta plataforma. De este modo se consigue un sistema que permite medir el pulso cardíaco.

Por tanto, usando componentes activos y pasivos se conseguirá una señal digital, de la que se podrá obtener fácilmente la medida de las pulsaciones por minuto.

Todo el proceso descrito anteriormente se puede resumir mediante el conjunto de etapas mostradas en la siguiente figura.

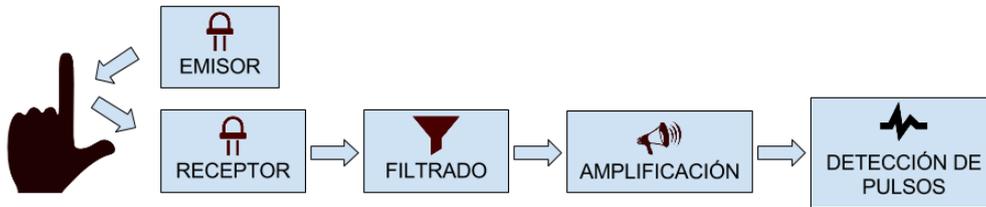


Ilustración 3-1. Flujograma de la electrónica necesaria para montar un Plethystógrafo

3.3 Descripción del kit de desarrollo CY8CKIT-042-BLE de Cypress

Este kit de desarrollo proporciona todos los elementos necesarios para comenzar a desarrollar proyectos propios que requieran de comunicación BLE, así como de una electrónica de acondicionamiento. El kit consta de:

- 1 placa base BLE Pioneer
- 1 módulo PRoC BLE
- 1 módulo PSoC 4 BLE
- 1 dongle USB CySmart
- 4 cables jumper
- 2 cables para el sensor de proximidad
- 1 pila de botón CR2032 de 3
- 1 cable USB de tipo Estándar-A a Mini-B
- Guía de inicio rápido

El kit viene con dos módulos en apariencia similares, un PRoC BLE y un PSoC 4 BLE. Aunque en principio puedan parecer semejantes, el PSoC 4 BLE incorpora más

funcionalidades que el PSoC, lo cual amplía las posibilidades de programación, por lo que para realizar este proyecto se ha utilizado el PSoC 4 BLE (de ahora en adelante simplemente PSoC).



Ilustración 3-2. Contenido del kit de desarrollo CY8CKIT-042-BLE

Como es de esperar, para probar algún diseño es necesario programar el PSoC. Para ello hay que conectar el PSoC con la placa base, ya que esta dispone de los pines e interfaces necesarias para hacer de intermediaria entre el PSoC y el ordenador. Posteriormente, es necesario conectar la placa base al ordenador mediante USB, y mediante un software que será descrito posteriormente se realizará la programación.

Una vez programado el PSoC, si se ha añadido algún tipo de funcionalidad Bluetooth, es posible probarla utilizando el dongle CySmart, el cual se conecta al ordenador mediante USB, y mediante un software que será descrito posteriormente, se podrá comprobar que las funcionalidades Bluetooth funcionan correctamente.

3.3.1 Instalación de PSoC Creator

Para poder programar en el PSoC, se usa el entorno de desarrollo que proporciona Cypress Semiconductor. Este software se encuentra disponible de forma gratuita para Windows en la página oficial de Cypress Semiconductor [17]. Actualmente la última versión disponible es la 3.2.

Una vez descargado, se puede instalar con el proceso y la configuración por defecto habitual en todos los programas instalados sobre el sistema operativo Windows.

Cuando el software se haya instalado, se puede ejecutar el entorno mediante el acceso directo “PSoC Creator 3.2”, que se encuentra dentro del menú inicio > todos los programas > Cypress.

Una vez abierto el entorno aparecerá una ventana parecida a la siguiente:

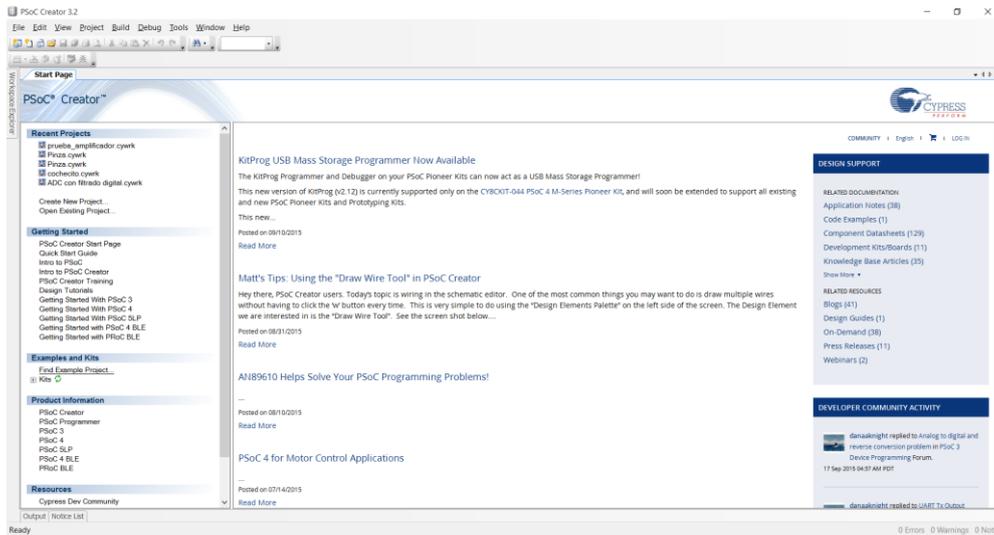


Ilustración 3-3. Ventana principal del software PSoC Creator

3.3.2 Creación de un proyecto en PSoC Creator

Para crear un nuevo proyecto hay que seleccionar la siguiente opción: File > New > Project.

En la ventana emergente hay que escoger PSoC 4100 BLE / PSoC 4200 BLE Design, darle un nombre al proyecto y pulsar sobre “OK”. A continuación se creará el proyecto, observándose la siguiente interfaz:

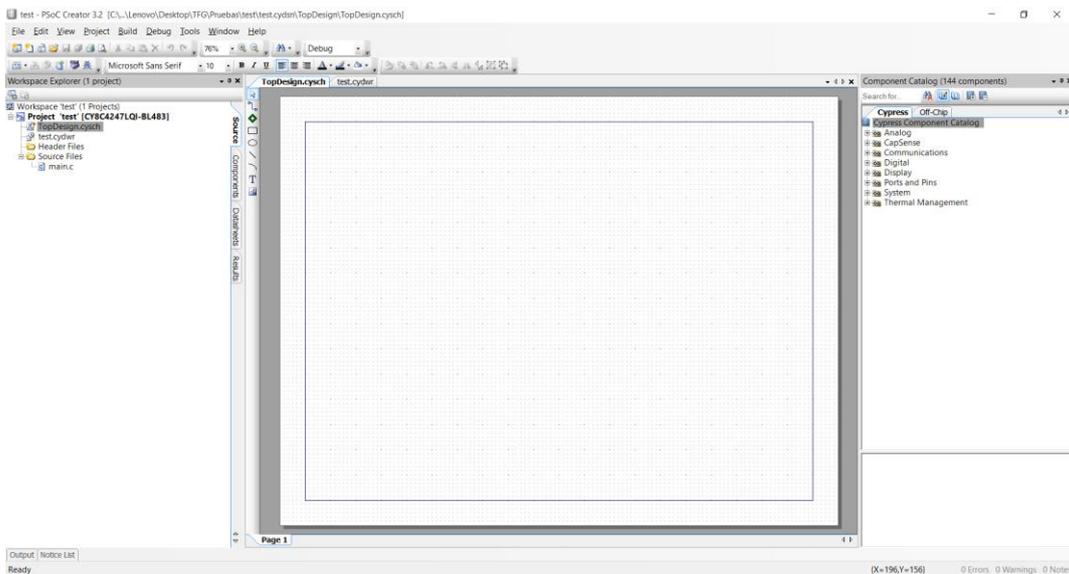


Ilustración 3-4. PSoC Creator mostrando el diseño esquemático de nuestro proyecto

3.3.3 Estructura de un proyecto en PSoC Creator

El proyecto consta de los siguientes elementos:

- Un archivo con extensión .cysch
- Un archivo con extensión .cydwr
- Un archivo con extensión .c

El archivo .cysch es el esquemático sobre el que se colocan los componentes del diseño. La librería del PSoC Creator incluye tanto componentes internos, que se pueden utilizar y configurar en el PSoC, como componentes externos al mismo. Aunque estos últimos componentes se conecten externamente al PSoC, la posibilidad de colocarlos en el esquemático permite conocer el circuito en su conjunto.

La ventana de la derecha llamada “Component Catalog”, permite buscar y seleccionar los componentes que se quieren añadir en el esquemático de la solución. Una vez seleccionado, se puede arrastrar hasta el lugar del esquemático deseado. Haciendo doble click sobre el componente, se abrirá una ventana con todos los parámetros que se pueden configurar de dicho módulo. Para conectar varios componentes sobre el esquemático, se puede pulsar la tecla “w” del teclado, y hacer click sobre los dos terminales de los dos componentes que hay que unir.

Una vez colocados en el esquemático todos los componentes necesarios y realizar la conexión de los mismos, se puede dar el caso de que sea necesario transmitir señales hacia o desde el PSoC. Por ello, desde el “Workspace Explorer” situado en la ventana de la izquierda, se puede hacer doble click sobre el archivo con extensión .cydwr para poder configurar correctamente las entradas y las salidas del PSoC.

Como se puede ver en la ilustración 3-5, en la ventana de la derecha aparecerán tantos componentes como pines de entrada y salida que se hayan configurado en el esquemático. Por lo que pulsando en la columna de “Port”, se mostrará una lista desplegable en la que se podrá elegir el puerto del PSoC al que conectar ese componente de entrada/salida.

Hay que tener en cuenta que aunque se puede mandar cualquier entrada y salida a cualquier puerto del PSoC, por la lógica interna del mismo hay determinados puertos que son más adecuados para determinados componentes internos del PSoC, por lo que conviene utilizarlos para esos componentes, ya que mejorará el rendimiento general del sistema. Esta asociación de componentes y de puertos se ve reflejada en la lista desplegable

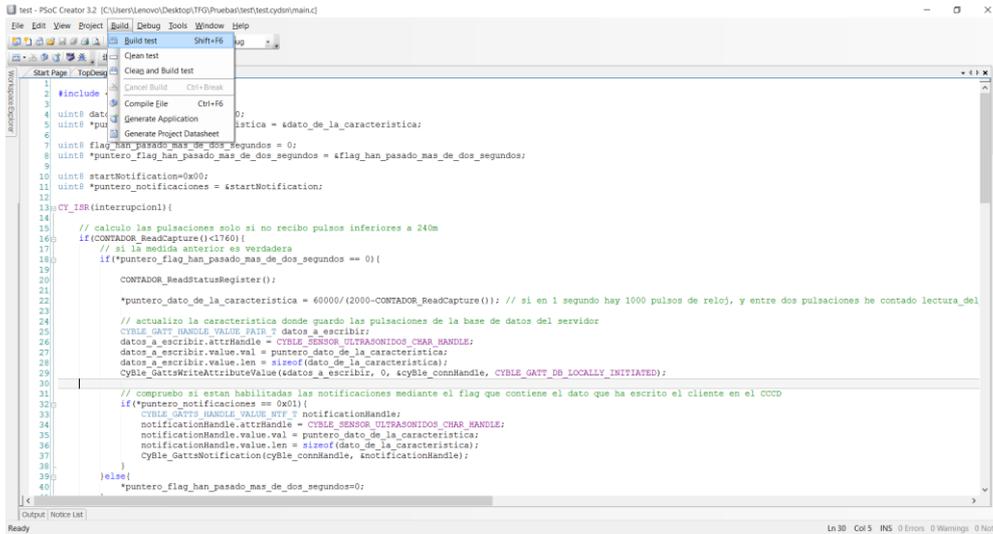


Ilustración 3-6. Generación del fichero project.h mediante el botón “Build test”

Finalmente, dentro del “main()” hay un bucle en el cual se debe añadir el código que se ejecutará indefinidamente en el PSoC.

3.3.4 Programación del PSoC desde PSoC Creator

Una vez que se haya terminado el programa, se puede compilar, así como cargarlo en el PSoC. Para ello, hay que conectar el PSoC utilizando el cable USB con el ordenador, y pulsar en el botón de la barra superior llamado “Program”, como se puede ver en la ilustración 3-7.

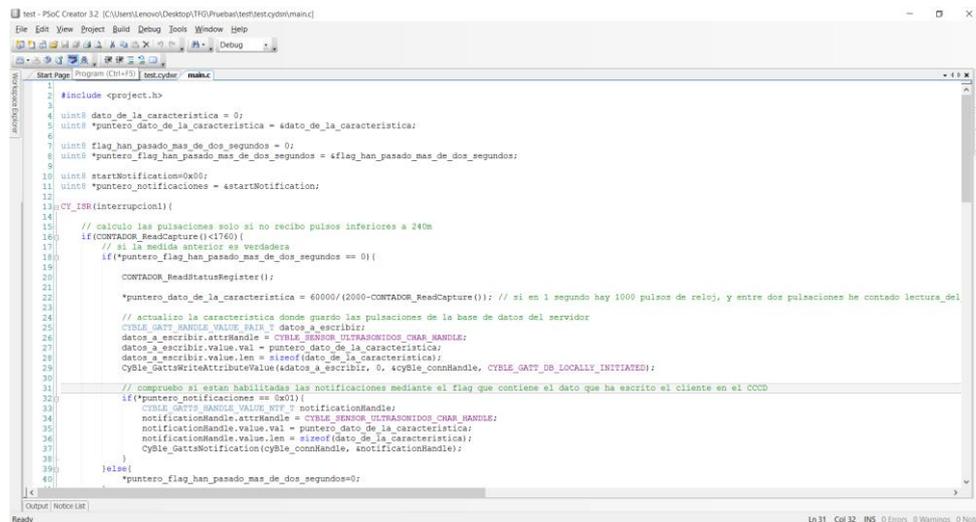


Ilustración 3-7. Programación del PSoC mediante el botón “Program”

En cuanto el dispositivo se haya programado, en la barra de estado inferior se mostrará “Ready”, y en la ventana “Output” situada también en la parte inferior se podrá leer algo parecido a esto “Device 'PSoC 4200 BLE CY8C4247LQ*-BLA83' was successfully

programmed at 10/04/2015 13:04:11". Por lo que el diseño se habrá cargado correctamente en el PSoC y será posible probarlo.

3.4 Desarrollo de Apps para Android

Dado que la aplicación para monitorizar el ritmo cardíaco va a ser utilizada en dispositivos con el sistema operativo Android, es necesario instalar una serie de utilidades de software necesarias para poder desarrollar aplicaciones para esta plataforma:

- Java SDK
- Android SDK
- Android Studio

3.4.1 Instalación del Java SDK

En primer lugar, para poder desarrollar aplicaciones en Android, es necesario tener instalado el kit de desarrollo de Java (*Java Development Kit*). Éste es necesario ya que el desarrollo de aplicaciones en Android implica el uso de archivos .java, y es necesario contar con un compilador que transforme los archivos .java en archivos .class de bytecode.

Por tanto, lo primero es descargar el JDK visitando la página web de Java [18], y a continuación pulsando junto a JDK, en "Download". Posteriormente, será necesario aceptar la licencia pulsando en "*Accept License Agreement*", y posteriormente seleccionar nuestro sistema operativo y su arquitectura. En este caso, se ha seleccionado Windows x64, cuyo paquete tiene el siguiente nombre `jdk-8u60-windows-x64.exe`, debido a que la versión más reciente disponible para Windows es la 8u60. Hay que tener en cuenta que para poder desarrollar aplicaciones para Android es necesario que la versión del JDK sea superior a la 6, y en el caso de querer desarrollar para Android 5.0 en adelante, la versión del JDK debe ser como mínimo la 7.

Una vez descargado, se puede instalar haciendo doble click sobre el archivo descargado, para posteriormente continuar con el asistente por defecto. Al finalizar la instalación, habrá que configurar las variables de entorno para que el siguiente paquete a instalar pueda encontrar el lugar donde se ha instalado Java. Para ello, hay que hacer click derecho sobre el icono de Equipo situado en el menú Inicio, seleccionar "Propiedades" > "Configuración avanzada del sistema" > "Variables de entorno..." y en el apartado de Variables del sistema pulsar sobre "Nueva..." y como nombre de la variable se puede utilizar "JAVA_HOME", y como valor "C:\Program Files\Java\jdk1.8.0_31".

Con esto se habrá quedado configurado el JDK para poder desarrollar aplicaciones que requieran el compilador de java.

3.4.2 Instalación de Android Studio y Android SDK

Los pasos para instalar el JDK han sido descritos en la sección anterior, pero aún son necesarios el entorno de desarrollo en el que poder programar, y el conjunto de APIs y herramientas que permiten desarrollar una aplicación.

Por ello, lo siguiente es instalar el entorno de desarrollo Android Studio, junto con las herramientas de desarrollo de Android (Android SDK). Google se encarga de proporcionar las dos herramientas empaquetadas, por lo que al ser la vía oficial para desarrollar aplicaciones para Android, se ha obviado la instalación de Android SDK y un entorno de desarrollo como Eclipse por separado, y se ha optado por la opción oficial de utilizar Android SDK junto con el IDE Android Studio [19].

Pulsar en “*DOWNLOAD ANDROID STUDIO FOR WINDOWS*”. Aceptar los términos y condiciones, y pulsar nuevamente en el botón situado abajo con el mismo texto que el botón de arriba. Esto hará que comience la descarga del paquete de software.

Al finalizar la descarga, se ejecuta el archivo descargado, y se instala siguiendo el asistente.

Una vez instalado, es necesario instalar una serie de paquetes que permiten desarrollar aplicaciones de Android. Para ello, una vez abierto Android Studio, hay que pulsar sobre en el icono “*SDK Manager*” situado en la barra superior [20]. Se abrirá el SDK Manager, con el cual se puede administrar las herramientas de desarrollo disponibles. Como mínimo hay que seleccionar las siguientes herramientas:

- Android SDK Tools
- Android SDK Platform-tools
- Android SDK Build-tools
- Android Support Repository
- Android Support Library
- Android 4.4 (API19) SDK Platform

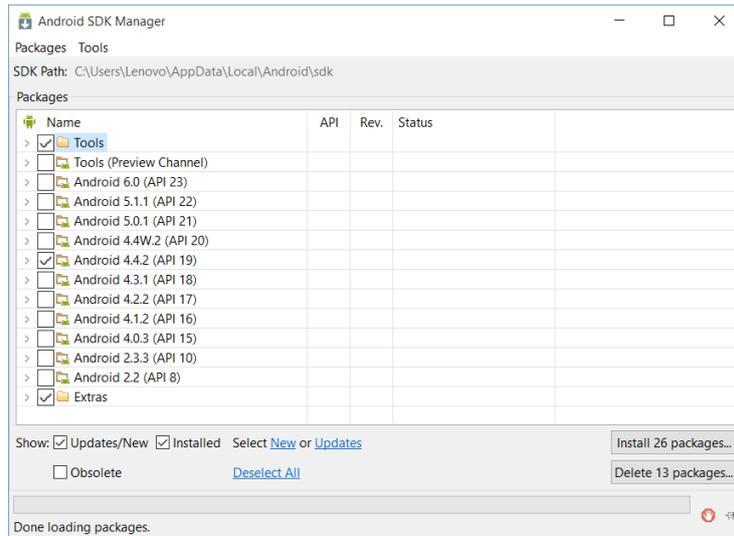


Ilustración 3-8. Android SDK Manager

Esto proporciona acceso a las librerías proporcionadas por la API seleccionadas (en este caso la API19), así como a las herramientas de compilación necesarias para generar el archivo .apk (que instalaremos en el dispositivo móvil una vez compilemos la aplicación). Tras haber seleccionado estos elementos, se pulsa en “Instalar paquetes”, se aceptan la licencia y se pulsa nuevamente en “Instalar”. Tras esto, ya estarán las herramientas de desarrollo listas para crear aplicaciones para Android.

Aunque ya se puedan desarrollar aplicaciones, es necesario probarlas para encontrar posibles fallos, por lo que será necesario un dispositivo que ejecute Android. Concretamente, Android Studio permite configurar un dispositivo virtual (AVD), pero dado que se dispone de un dispositivo móvil para realizar el proyecto, se realizará la configuración del mismo para poder depurar las aplicaciones directamente sobre él.

3.4.3 Configuración del dispositivo móvil para depurar aplicaciones

Para poder depurar aplicaciones en el dispositivo móvil, es necesario realizar una pequeña configuración en el dispositivo. Cualquier dispositivo con Android 4.2 en adelante tiene las opciones de desarrollo ocultas, por lo que lo primero es habilitar las opciones de desarrollo. Para ello, hay que abrir los ajustes del dispositivo, y pulsar en “Información del teléfono”. Ahora se pulsan 7 veces seguidas en “Número de compilación” hasta que aparezca un mensaje indicando que se han activado las opciones de desarrollo.

Tras activarlas y volviendo a los Ajustes del dispositivo, ya se puede pulsar en la nueva opción “Opciones de desarrollo”. Una vez dentro, se pulsa en “Depuración USB”, y con eso ya estaría configurado el dispositivo. Ahora sólo queda asegurar de que están

instalados y actualizados los drivers de nuestro dispositivo móvil en el ordenador, para que el ordenador pueda reconocer correctamente el dispositivo cuando se conecte mediante un cable USB.



Ilustración 3-9. Dispositivo con Android mostrando las opciones de desarrollo

3.4.4 Creación de un proyecto en Android Studio

Al abrir Android Studio por primera vez, si no se ha creado ningún proyecto se abrirá una ventana de bienvenida [21]. Aquí se debe seleccionar “*Start a new Android Studio Project*”. A continuación, hay que escribir un nombre a la aplicación, así como indicar el dominio de la compañía. Por último, se elige el lugar del disco duro en el que guardar el proyecto.

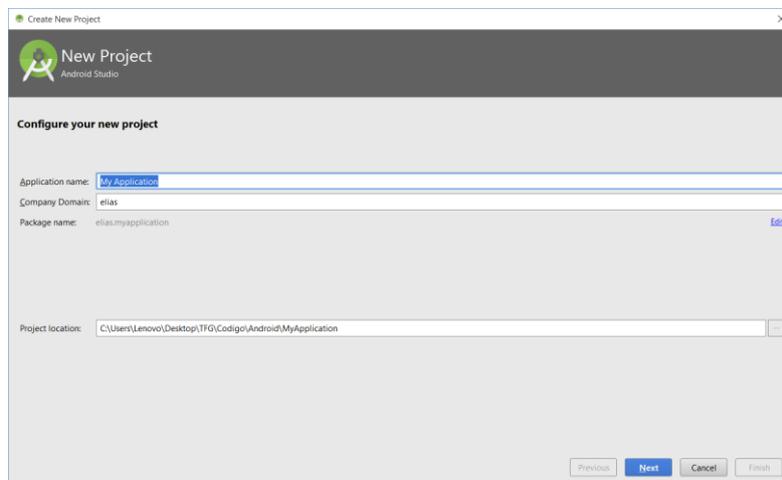


Ilustración 3-10. Ventana de creación de un nuevo proyecto en Android Studio

Lo siguiente es establecer los dispositivos para los que funcionará la aplicación, ya que Android puede ejecutarse en una gran variedad de dispositivos, ya sean *smartphones*, *tablets*, *TV*, *wearables* o automóviles. Una vez elegido el dispositivo, lo siguiente es establecer cuál será la versión mínima del SDK. La versión del SDK determinará la versión mínima de Android que debe tener instalado el dispositivo para poder ejecutar la aplicación.

Hay que tener en cuenta que aunque utilizar la versión más reciente del SDK (y por tanto de la API) proporciona acceso a las últimas características y mejoras introducidas en Android, debido a la fragmentación de Android se corre el riesgo de que una aplicación sea incompatible con la mayoría de los dispositivos existentes en el mercado, ya que estos suelen utilizar versiones de Android más antiguas, que apenas reciben un año de soporte por parte del fabricante, por lo que enseguida se quedan sin actualizaciones.

Por ello, lo ideal es elegir la primera versión del SDK en la que se introdujeron las características de la API que se quieren utilizar en una aplicación, ya que así se conseguirá que una aplicación sea capaz de acceder a los recursos de la API que necesita para llevar a cabo el proyecto que se tiene en mente, y a la vez se conseguirá que la aplicación sea compatible con el mayor número de dispositivos posible, aunque haya que renunciar a las últimas funciones y mejoras de Android.

Tras haber elegido la versión mínima del SDK, hay que añadir una actividad al proyecto. En este caso, lo normal es comenzar con una “*Blank Activity*”, la cual se explicará más adelante. Por defecto, la actividad creada se llamará “*MyActivity*”, por lo que se puede aceptar la configuración por defecto y pulsar en el botón “Finish”, con lo que se habrá creado el proyecto.

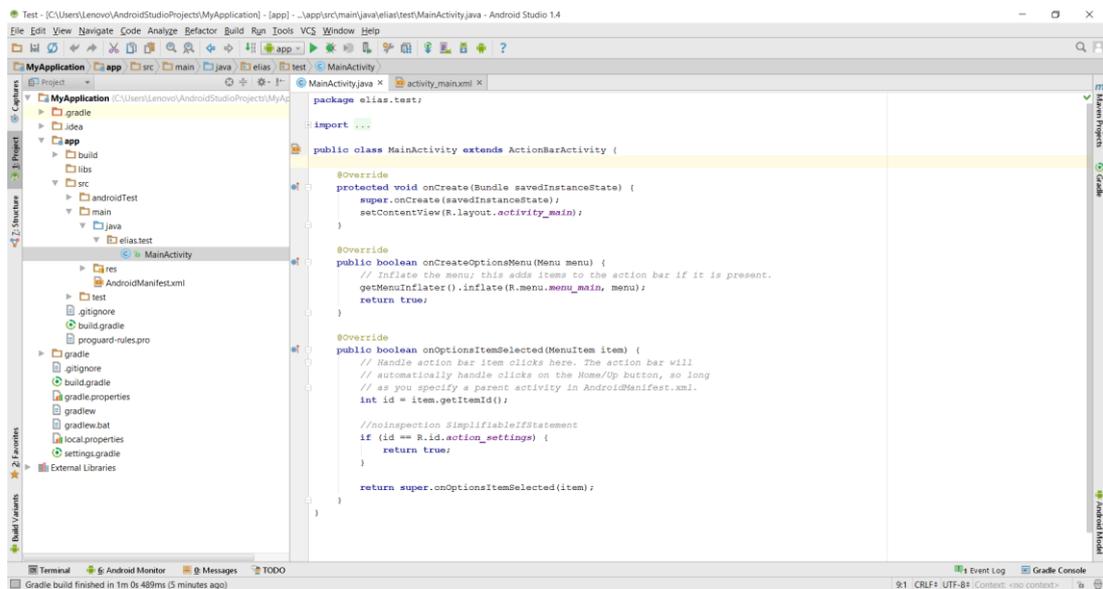


Ilustración 3-11. Android Studio mostrando el fichero MainActivity.java

Una vez preparado el entorno de desarrollo y creado un nuevo proyecto, lo siguiente es tener claros algunos conceptos sobre la estructura de un proyecto Android, es decir, los elementos que lo componen.

3.4.5 Estructura de un proyecto Android

Para poder hacerse una mejor idea de la estructura que tiene el proyecto es necesario realizar una modificación en Android Studio. Dicho cambio implica pulsar en la lista desplegable situada en la parte superior izquierda, y cambiar de “Android” a “Project”. En la nueva jerarquía lo primero que aparece es una carpeta con el nombre del proyecto, y otras carpetas que representan los distintos módulos de la aplicación [22].

Normalmente, sólo se utilizará el módulo “app”, el cual incluye dos carpetas fundamentales:

- /app/src/main/java
- /app/src/main/res

La carpeta “java” contiene archivos con el código fuente de la aplicación, sin incluir los archivos que definen la interfaz de usuario [23]. Por lo tanto, cualquier archivo .java estará aquí.

Cuando se inicie una aplicación de Android, el primer fichero que se ejecutará será el MainActivity.java, por lo que se debe configurar adecuadamente. Dado que la clase *MainActivity* se extiende de una clase *Activity*, su ciclo de vida es el mismo que el de cualquier Activity en Android (ver Figura 3-12).

El primer método que se ejecuta cuando se crea una *Activity* es el método “onCreate()”, por lo que en este método se debe establecer el archivo .xml del que se obtendrá la apariencia de nuestra Activity, así como obtener la referencia a los distintos elementos que se hayan situado en la pantalla.

A su vez, los métodos “onStart()”, “onResume()”, “onPause()”, “onStop()” y “onDestroy” permiten realizar distintas acciones en función de los procesos por los que vaya pasando una Activity, ya que puede ser que se cambie de una Activity a otra, que se cierre la aplicación, o que posteriormente se vuelva a la misma Activity. Por lo tanto, con los métodos listados anteriormente, se pueden procesar adecuadamente todo este tipo de situaciones.

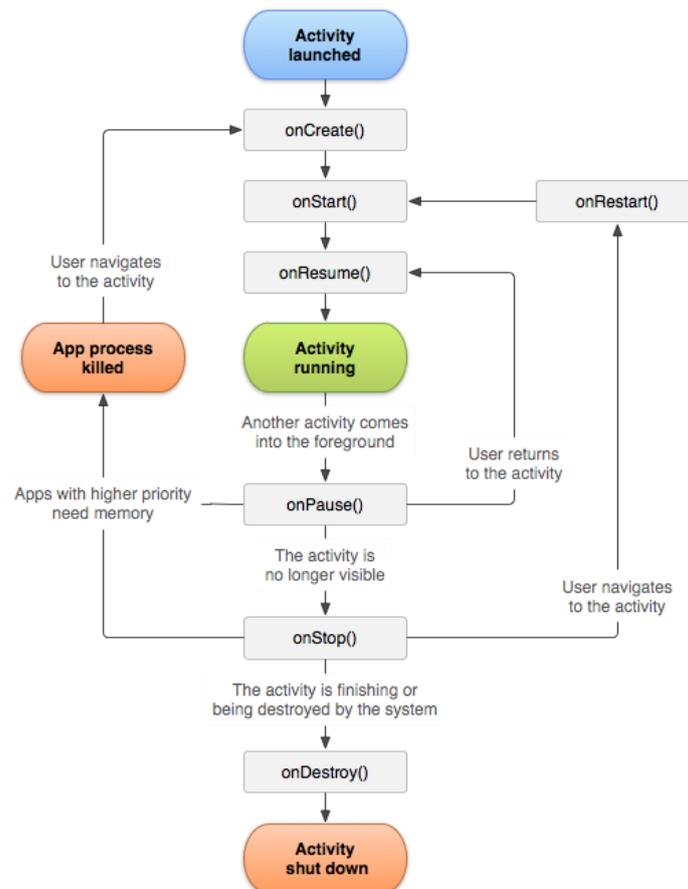


Ilustración 3-12. Ciclo de vida de una Activity en Android

La carpeta “res” contiene archivos que definen la interfaz de usuario. Por lo tanto, cualquier archivo como imágenes, iconos, sonidos, strings, layouts se incluyen en esta carpeta.

En un layout se pueden definir mediante el lenguaje de marcado XML los distintos elementos que compondrán la interfaz, ya sean cuadros de texto, botones, listas desplegables, menús, e incluso se pueden crear controles propios para la interfaz.

Dentro de un layout, hay que definir la disposición de los elementos dentro de la interfaz. Lo más habitual para diseñar interfaces sencillas es utilizar un *LinearLayout* o un *RelativeLayout*. Mediante *LinearLayout* se sitúan los elementos uno a continuación del otro, ya sea en horizontal o en vertical, y mediante *RelativeLayout* se distribuyen los elementos unos respecto de otros. Por lo que elegir entre uno u otro será elección del desarrollador.

Una vez definida la distribución de los elementos, los nuevos elementos que se incluyan tendrán una serie de propiedades que se podrán modificar mediante etiquetas, las cuales modificarán su tamaño, forma, distribución, márgenes respecto a otros elementos, colores, entre otros.

Volviendo a la estructura del proyecto, hay dos archivos muy importantes situados fuera de la carpeta java y res:

- /app/src/main/AndroidManifest.xml
- /app/build.gradle

El archivo *AndroidManifest.xml* define los aspectos fundamentales de la aplicación, como son el nombre de la aplicación, su icono, las distintas pantallas y servicios que tiene la aplicación, y los permisos que requiere la aplicación para funcionar.

El archivo *build.gradle* contiene la información necesaria para poder compilar la aplicación de Android, como la versión del SDK utilizada para realizar la compilación, el ID de la aplicación y la versión mínima del SDK. Hay un *build.gradle* para cada módulo del proyecto, así como un *build.gradle* para todo el proyecto.

3.4.6 Instalar la aplicación en un dispositivo con Android

Una vez creada la aplicación, se puede realizar la compilación e instalación en un dispositivo con Android para probarla y detectar posibles fallos.

Para ello, lo primero es haber configurado correctamente el dispositivo móvil con Android para poder depurar aplicaciones mediante USB, y conectarlo mediante el cable USB al ordenador. A continuación, se pulsa sobre el botón de la barra superior “Run *app*”, tras lo cual Android Studio comenzará a compilar el proyecto, hasta que muestre una ventana como la de la ilustración 3-13. En esta ventana se puede elegir entre ejecutar la aplicación en un dispositivo virtual, o instalarla en un dispositivo real. Por lo que, en este trabajo se elegirá el dispositivo que aparece en la lista.

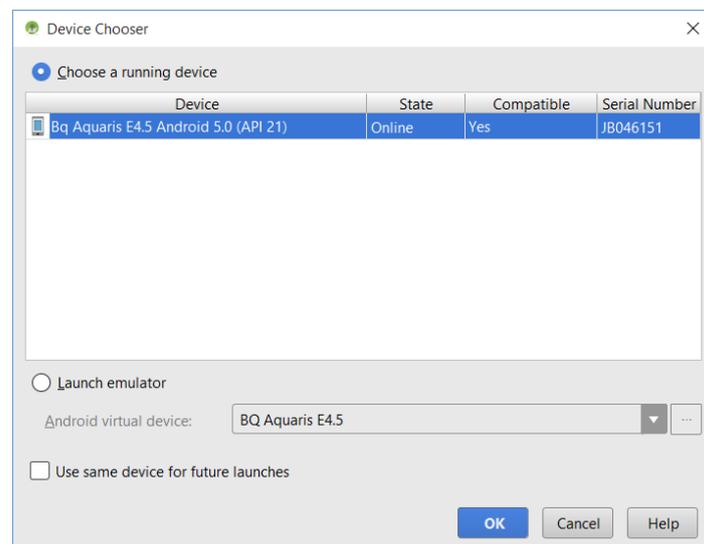


Ilustración 3-13. Ventana para elegir el dispositivo con Android en el que instalar la aplicación

Cuando la aplicación se haya instalado y se esté en ejecución, en la parte inferior de Android Studio se podrá hacer uso de la pestaña llamada “*Android Monitor*”, dentro de la cual se seleccionará la pestaña “*logcat*” y visualizar en tiempo real la información relativa al dispositivo Android, por lo que si la aplicación experimenta algún problema, se verá reflejado en este lugar y será posible detectar el fallo.

3.5 Conclusiones

Tras haber descrito las distintas herramientas utilizadas para desarrollar este trabajo, en esta sección se van a recapitular los distintos elementos que son necesarios para llevar a cabo la construcción y el diseño de este Trabajo Fin de Grado.

En primer lugar, para la parte de la sensorización, se utilizará un transceptor que emita una señal luminosa y que sea capaz de recoger la señal reflejada por el tejido. Esta señal deberá ser acondicionada antes de ser procesada, por lo que será filtrada y amplificada mediante una serie de filtros compuestos por resistencias, condensadores y una serie de amplificadores operacionales. Toda esta parte de sensorización será construida con ayuda del PSoC, ya que incorpora algunas funciones que evitarán utilizar algunos componentes electrónicos.

Una vez obtenida la señal analógica sin ruido, ésta será introducida en un amplificador operacional en modo comparador, y mediante una tensión de referencia ligeramente superior a 0 V, se generará una serie de señales de nivel bajo cuando no haya ningún latido del corazón, y de nivel alto cuando haya algún latido del corazón.

Esta señal “digital” de ceros y unos será utilizada en el PSoC para poder calcular las pulsaciones por minuto. Para programar el PSoC se utilizará el entorno de desarrollo PSoC Creator, con el que se realizará toda la parte relativa al procesado de la señal y a la transmisión mediante Bluetooth.

La señal enviada por Bluetooth será recibida por un dispositivo móvil inteligente, basado en el sistema operativo Android. Dicho dispositivo ejecutará una aplicación realizada mediante Android Studio, que permitirá recibir la señal mediante Bluetooth y mostrarla en tiempo real.

Capítulo 4

Descripción del Hardware y Software desarrollado

4.1 Introducción

Ya se ha hablado sobre los elementos de los que va a estar compuesto el proyecto, así como de los entornos de desarrollo necesarios para programar algunas partes del mismo.

Ahora se va a explicar la programación y la configuración de todos los elementos que integran el proyecto, comenzando por la parte de la sensorización, en la que se explicará cómo se ha calculado el valor de las resistencias y de los condensadores que se han utilizado, así como los parámetros de los filtros y las etapas de amplificación de la señal.

Después se describirá el programa implementado en el PSoC, explicando toda la lógica que se ha tenido que desarrollar para calcular las pulsaciones por minuto, así como la configuración que se ha tenido que hacer para poder tener funcionando correctamente el bluetooth de baja energía del PSoC.

Por último, se explicará cómo se ha desarrollado la aplicación para Android, mostrando el funcionamiento que se ha implementado, así como los distintos métodos que se han programado para poder tratar correctamente los datos recibidos por Bluetooth.

4.2 Descripción del sistema de medida del ritmo cardíaco

El sistema de medida está formado por varios componentes electrónicos, por lo se va a dividir el sistema en varias etapas en función de las tareas que se realicen en cada una.

4.2.1 Etapa de emisión y recepción de luz

Para comenzar a montar el sistema de medida, en primer lugar se ha utilizado un foto reflector, modelo RPR-220. Este componente está formado por un diodo que emite luz infrarroja, y por un fototransistor que detecta señales de una longitud de onda de hasta 800 nm [25].

Este componente se va a encargar de emitir una luz infrarroja, que se reflejará sobre el tejido contra el que la proyectemos, y a su regreso sobre el componente incidirá sobre el fototransistor, generando así una señal que se podrá medir. Esta señal variará en función de la luz reflejada en el fototransistor, por lo que cada vez que se produzca un latido del corazón, el nivel de sangre del tejido variará, y la luz reflejada por el tejido será distinta, por lo que al incidir sobre el fototransistor se podrá medir esta diferencia.

El RPR-220 consta de 4 pines, los cuales se conectarán de la siguiente forma:

- Cátodo del diodo emisor de luz: a masa (0 V).
- Emisor del fototransistor: a masa (0 V).
- Ánodo del diodo emisor de luz: a una resistencia de 120 Ω , para que al conectar la otra patilla de la resistencia con $V_{cc} = 3,3$ V se genere una intensidad directa por el diodo de unos 27 mA.
- Cátodo del fototransistor: a una resistencia de 10 k Ω , para que al conectar la otra patilla de la resistencia a $V_{cc} = 3,3$ V actúe como resistencia de *pull-up* y se evite así el posible ruido que pueda alterar la salida del fototransistor.

Por lo tanto, esta primera etapa sólo requiere dos resistencias, un fototransistor, la masa y la alimentación a 3,3 V.

El circuito a construir será el mostrado en la ilustración 4-1.

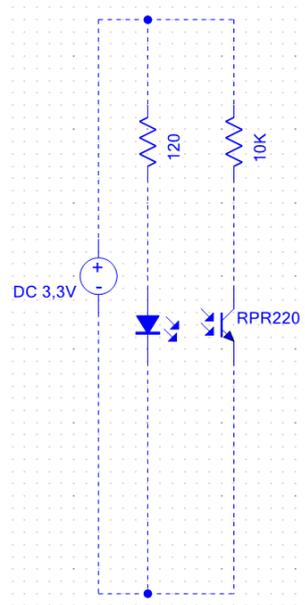


Ilustración 4-1. Etapa de emisión y recepción de luz

4.2.2 Etapa de filtrado y amplificación

Una vez que se ha generado una señal que varía con las pulsaciones recibidas, hay que filtrar la señal para evitar los posibles ruidos derivados del sensor, y amplificar la señal para que sea más fácil trabajar con ella.

Para filtrar la señal se van a utilizar dos filtros. El primero será un filtro paso alto pasivo, el cual dejará pasar las frecuencias que se encuentren por encima de 0,7 Hz. El segundo filtro será un filtro paso bajo activo, el cual dejará pasar las frecuencias situadas por debajo de 2,34 Hz. Se ha elegido un filtro activo porque se quiere amplificar la señal, y por lo tanto si sólo se hubiera utilizado un filtro pasivo no se podría haber tenido una ganancia superior a 1.

En la realización de esta etapa se utilizarán resistencias y condensadores, pero se van a utilizar los amplificadores operacionales que vienen integrados en el PSoC, por lo que así se podrán ahorrar componentes. Para conectar las señales al amplificador operacional interno del PSoC, se deberán utilizar una serie de pines situados en el PSoC, los cuales se explicará posteriormente cómo configurarlos [4].

Comenzando por el filtro paso alto (ver Ilustración 4-2), la señal generada por el fototransistor se introducirá en el condensador del filtro paso alto, el cual tendrá una frecuencia de corte de 0,7 Hz, por lo que fijando el valor del condensador en 4,7 μF y utilizando la siguiente ecuación:

$$\text{Frecuencia de corte} = \frac{1}{2 \cdot \pi \cdot R \cdot C}$$

Se despeja la ecuación y se obtiene que la resistencia de este filtro debe tener un valor de 47 kΩ, por lo que ya se tendrían calculados los parámetros del primer filtro. Ahora se coge la salida del condensador y se conecta en la entrada no inversora del amplificador operacional del filtro paso bajo.

Para el filtro activo paso bajo, como la frecuencia de corte será de 2,34 Hz, una vez fijado el valor del condensador en 100 nF y utilizando la ecuación anterior, se obtiene que la resistencia conectada a la realimentación deberá tener un valor de 680 kΩ.

Para calcular el valor de la resistencia conectada a la patilla inversora, se debe utilizar la fórmula para calcular la ganancia:

$$\text{Ganancia} = 1 + \frac{\text{Resistencia realimentación}}{\text{Resistencia inversora}}$$

Asumiendo que se quiere una ganancia de 10, la resistencia de la patilla inversora tendrá un valor de 68 kΩ.

Con esto se habría construido un filtro paso bajo y paso alto de primer orden, pero para mejorar las características del circuito, se va a conectar la salida del amplificador operacional a una nueva etapa de filtrado y amplificación compuesta nuevamente por otro filtro pasivo paso alto con las mismas características que el que se acaba de construir, y otro filtro activo paso bajo con las mismas características que el que se acaba de construir.

Esto dará una ganancia en cascada de 100, y mejorará el filtrado de la señal.

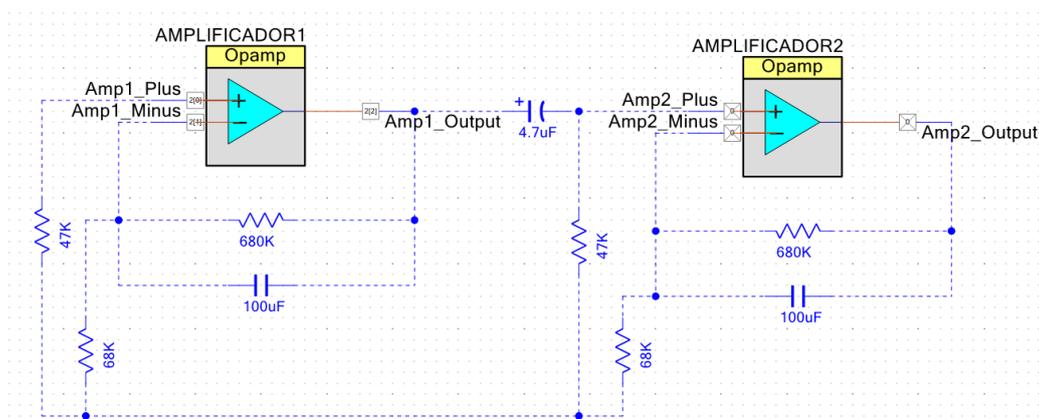


Ilustración 4-2. Etapa de filtrado y amplificación

4.2.3 Etapa de conversión analógica-digital

Por la salida del amplificador operacional de la última etapa de filtrado se ha obtenido una señal analógica que refleja con bastante precisión los latidos del corazón, pero para poder realizar algún tipo de lógica con un microcontrolador como es el PSoC, hay que generar una señal digital, a partir de esta señal analógica, que se pueda introducir en los componentes digitales del PSoC.

Para generar esta señal digital, se va a conectar la señal analógica en un amplificador operacional en modo comparador, y mediante una tensión de referencia, el amplificador dará un nivel alto cuando la señal esté por encima de la tensión de referencia, y un nivel bajo cuando la señal esté por debajo de la tensión de referencia.

En esta etapa se va a utilizar una resistencia, pero el amplificador operacional que hace falta se va a coger nuevamente del interior del PSoC.

Lo primero será conectar la señal analógica a la entrada no inversora del amplificador operacional. Para generar la tensión de referencia de 10 mV, se aprovechará que el PSoC puede proporcionar una determinada corriente configurable por uno de sus pines, y se conectará ese pin a una resistencia de 391 Ω , por la que se hará pasar 250,8 μA . Esto generará los 10 mV que hacen falta, y se conectarán a la entrada inversora del amplificador operacional.

La salida del amplificador operacional en modo comparador dará una señal digital, por lo que se podrá introducir la señal en el PSoC, y utilizarla como una señal digital, que ahora sí se podrá aprovechar.

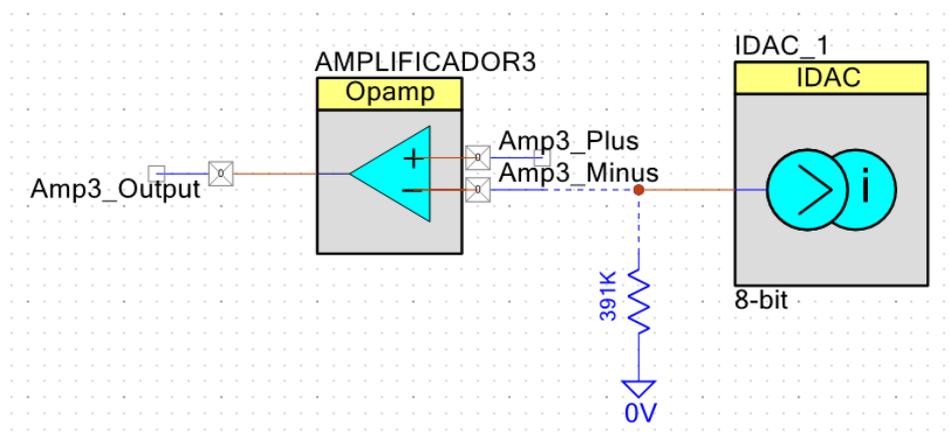


Ilustración 4-3. Etapa de conversión analógica-digital

Por lo tanto, una vez que ya se han descrito todas las etapas del sistema de medida del ritmo cardíaco, en la ilustración 4-4 se va a mostrar una visión general de cómo estarían distribuidos los componentes.

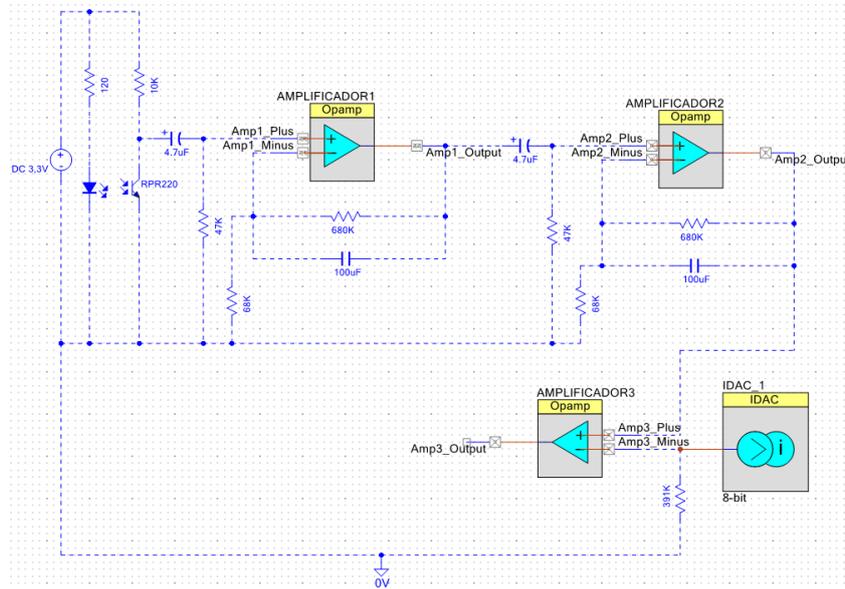


Ilustración 4-4. Esquemático del sensor de medida del ritmo cardíaco

4.2.4 Montaje del sistema de medida

Durante el desarrollo de este proyecto se ha estado trabajando con una protoboard, la cual aporta flexibilidad y permite sustituir rápidamente unos componentes por otros. Esto facilita el desarrollo y la corrección de errores mientras se está diseñando el sistema de medida, pero no es un formato adecuado para la presentación final del proyecto, ya que la gran cantidad de cables dificulta la utilización del sensor.

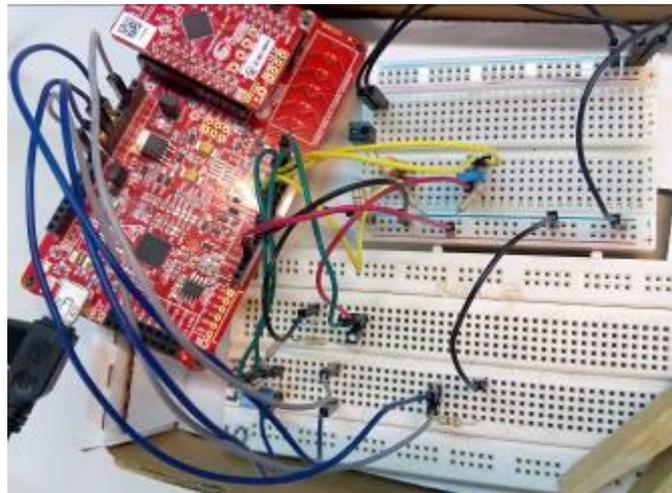


Ilustración 4-5. Montaje del sensor de medida en una protoboard

Por ello, para el montaje final se ha utilizado un *shield* compatible con la distribución de pines que lleva la placa de desarrollo del PSoC.

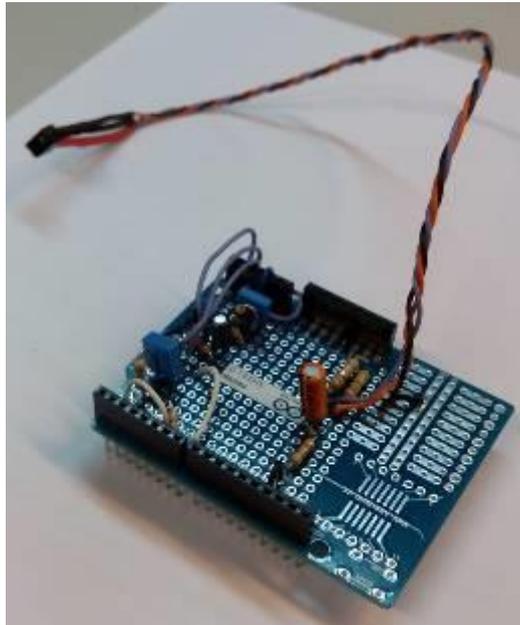


Ilustración 4-6. Montaje final del *shield* con los componentes soldados

Sobre el *shield* se han soldado los mismos componentes utilizados en la protoboard, y se ha utilizado un cable más largo para las conexiones del fototransistor sobre el que se pondrá el dedo para medir las pulsaciones. Por último, se ha utilizado tubo termoretráctil en los extremos de las patillas del fototransistor para proteger las conexiones eléctricas entre el fototransistor y el *shield*.

Una vez terminado el proceso de montaje sobre el *shield*, ya se puede acoplar sobre el PSoC, siendo el resultado final el mostrado en las ilustraciones 4-7, 4-8 y 4-9.

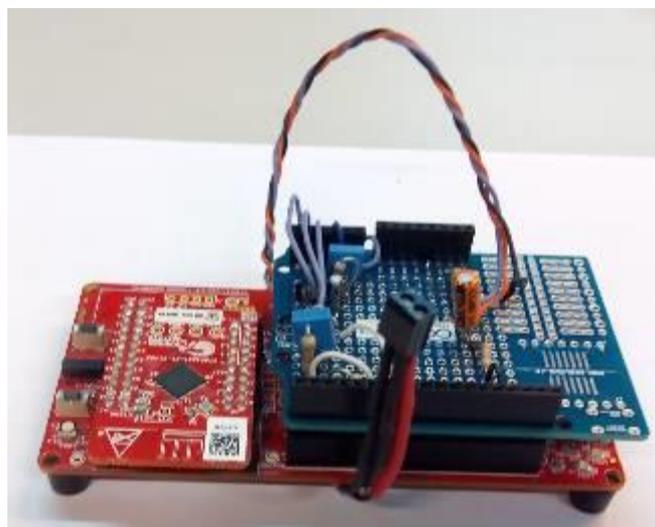


Ilustración 4-7. Diseño final del shield conectado al PSoC

Como se puede ver, se ha obtenido un diseño final más reducido respecto al prototipo construido sobre la protobard, y debido a la longitud del cable conectado al fototransistor, se ha ganado movilidad para poder realizar medidas en una mayor cantidad de zonas de las que permitía el prototipo inicial.



Ilustración 4-8. Diseño final del shield conectado al PSoC

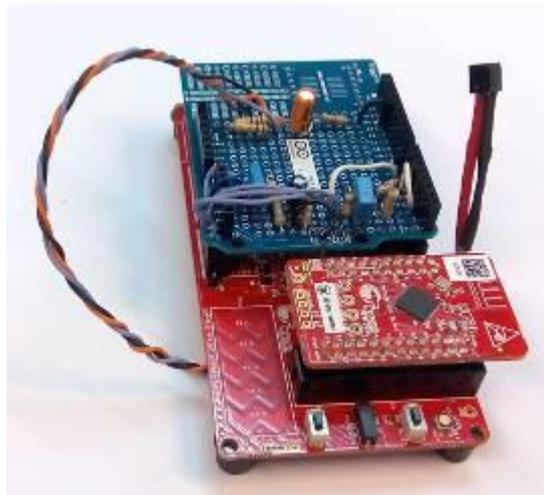


Ilustración 4-9. Diseño final del shield conectado al PSoC

4.3 Descripción de la arquitectura implementada en el PSoC

El PSoC va a ser la pieza central del proyecto, ya que va a ser el encargado de recibir la señal del pulso, procesarla para obtener las pulsaciones por minuto y enviarla por Bluetooth a un dispositivo. Además, aprovechando sus componentes internos se va a introducir en la etapa de la sensorización, para poder fabricar los filtros activos de paso

bajo que hacen falta, así podrán ahorrarse componentes a la hora de comprar todo lo necesario.

4.3.1 Creación del proyecto en PSoC Creator

En primer lugar se creará un nuevo proyecto como ya se ha visto en el capítulo anterior, mediante el menú de la barra superior “File > New > Project”, se elige la plantilla que viene por defecto llamada “PSoC 4100 BLE / PSoC 4200 BLE Design”, se le da un nombre al proyecto (en este caso se ha elegido “Pulsaciones”), se elige la ruta donde se guardará, y se hace click en “OK”.

Con esto, se creará un nuevo proyecto, y en la ventana del PSoC Creator se abrirá el archivo “TopDesign.cysch”, el cual será el esquemático sobre el que se colocarán los componentes.

4.3.2 Configuración de los componentes utilizados en la etapa de filtrado

Lo primero será añadir los dos amplificadores operacionales que se van a utilizar en los filtros activos. Para ello, en la ventana de la derecha llamada “Component Catalog” se introduce “opamp” en el cuadro de búsqueda, y se arrastra el componente llamado “Opamp [v1.10]” al esquemático. Para añadir el segundo amplificador operacional se repite el mismo proceso.

Una vez situados en el esquemático, se hace doble click sobre uno de ellos, y se configura de la siguiente manera:

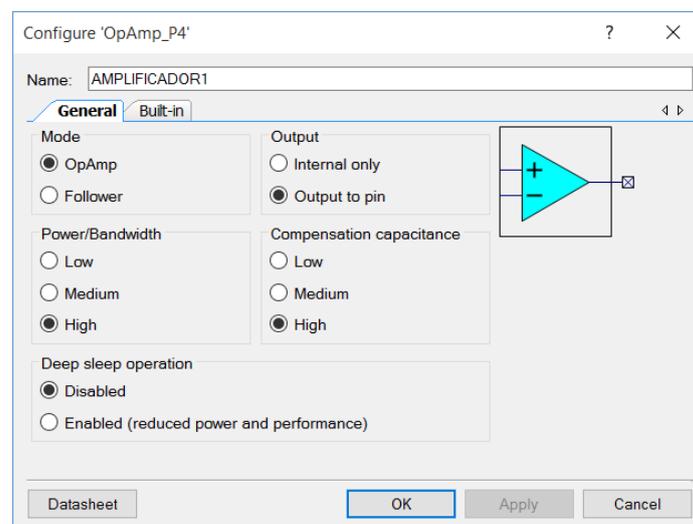


Ilustración 4-10. Ventana de configuración del amplificador operacional de la etapa de filtrado

Como se puede ver, lo primero ha sido darle un nombre al componente, y configurar el amplificador en modo “OpAmp”. Se le ha indicado que utilice toda la potencia que necesite para funcionar correctamente, y se le ha especificado que la salida mande a un pin. Tras repetir esta configuración con el segundo amplificador operacional, se deberán añadir los pines necesarios para poder mandar señales desde o hacia el PSoC.

Para ello hay que dirigirse al cuadro de búsqueda del “Component Catalog”, e introducir “analog pin”, se selecciona “Analog pin [v2.20]” y se arrastra al esquemático. Se repite este proceso otras cinco veces, ya que se debe contar con un total de seis pines analógicos para poder introducir y sacar señales de los dos amplificadores operacionales.

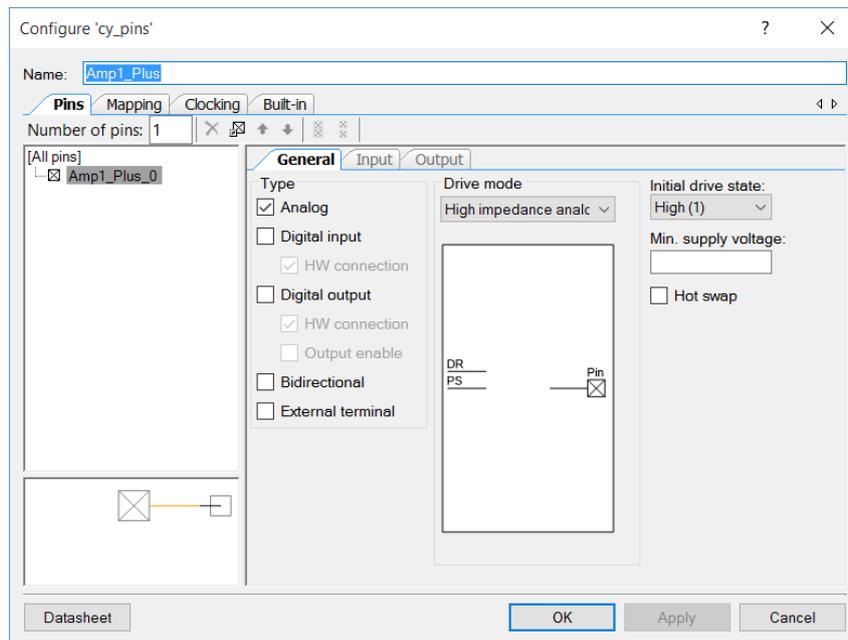


Ilustración 4-11. Ventana de configuración de los pines de la etapa de filtrado

Los seis pines se deben configurar como se muestra en la ilustración 4-11, ya que todos son pines analógicos, y su finalidad es proporcionar dos entradas y una salida para cada amplificador operacional. Una vez configurados, hay que conectarlos con los amplificadores, para ello se pulsa la tecla “w” y pulsando en la punta del pin y en la punta del amplificador, quedando el esquemático de la siguiente forma:

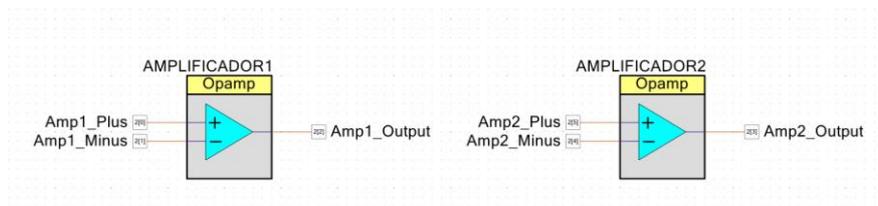


Ilustración 4-12. Esquemático de la etapa de filtrado

A continuación, hay que dirigirse a la ventana izquierda llamada “Workspace Explorer”, y dentro del proyecto se encontrará un archivo llamado “Pulsaciones.cydwr”, se hace doble click sobre él, y en la ventana de la derecha se asigna el puerto al que se quieren conectar los pines analógicos que se han colocado en el esquemático. En este caso, y debido al funcionamiento interno del PSoC, se ha optado por utilizar los puertos que la documentación del PSoC consideraba más adecuados para utilizar los amplificadores operacionales. No se recomienda utilizar otros, ya que durante la fase de pruebas ha habido algunos problemas cuando no se usaban los puertos estipulados en la documentación. Por ello, lo adecuado será utilizar la siguiente asignación:

- Amplificador 1 – Entrada no inversora: P2[0]
- Amplificador 1 – Entrada inversora: P2[1]
- Amplificador 1 – Salida: P2[2]
- Amplificador 2 – Entrada no inversora: P2[5]
- Amplificador 2 – Entrada inversora: P2[4]
- Amplificador 2 – Salida: P2[3]

Alias	Name	Port	Pin	Lock
Amp1_Minus		P2[1] OA0:vminus, SCB0:spi_select[2]	38	<input checked="" type="checkbox"/>
Amp1_Output		P2[2] OA0:vout_10x, SRSS:wakeup, SCB0:spi_select[3]	39	<input checked="" type="checkbox"/>
Amp1_Plus		P2[0] OA0:vplus, SCB0:spi_select[1]	37	<input checked="" type="checkbox"/>
Amp2_Minus		P2[4] OA1:vminus	41	<input checked="" type="checkbox"/>
Amp2_Output		P2[3] OA1:vout_10x, SRSS:ext_clk_1f	40	<input checked="" type="checkbox"/>
Amp2_Plus		P2[5] OA1:vplus	42	<input checked="" type="checkbox"/>

Ilustración 4-13. Asignación de Puertos a los pines analógicos de la etapa de filtrado

Tras haber asignado los puertos a los pines correspondientes, ya se pueden añadir las primeras líneas de código en el fichero “main.c”, pero antes es conveniente compilar lo que se lleva hecho, para que PSoC Creator genere el fichero “project.h”, y desde el “main.c” se puedan utilizar los componentes que se han añadido en el esquemático. Para ello, en el menú superior se hace click en “Build > Build Pulsaciones”.

Si no ha habido ningún fallo en el esquemático, la compilación terminará y se podrá editar el fichero “main.c”. En este caso, como sólo se han añadido dos amplificadores operacionales, para inicializarlos cuando se encienda el PSoC sólo se deben añadir estas dos líneas dentro de la función “main()”:

```
AMPLIFICADOR1_Start();
```

```
AMPLIFICADOR2_Start();
```

4.3.3 Configuración de los componentes utilizados para calcular el ritmo cardíaco

Una vez terminada la etapa de filtrado y de amplificación, se habrá logrado generar una señal analógica que reflejará los latidos del corazón, pero para el PSoC esta señal no es válida. El PSoC necesita una señal digital con la que poder trabajar.

Por ello, se ha utilizado otro amplificador operacional pero esta vez en modo comparador. Por la entrada no inversora se va a introducir la señal analógica que contiene el pulso, y por la entrada inversora una tensión de referencia para poder descartar el ligero ruido que haya por encima de 0 V. Por tanto, cada vez que la entrada no inversora sea superior a la entrada inversora, la salida de este amplificador operacional estará a nivel alto, lo cual será un “1 digital”. Cuando la entrada no inversora sea inferior a la tensión de referencia que se ha introducido en la entrada inversora, la salida del amplificador estará a nivel bajo, lo cual será un “0 digital”. Esto hará que la señal analógica con los latidos del corazón se convierta en una señal digital de ceros y unos que el PSoC podrá procesar.

Lo primero será situar en el “TopDesign.cysch” el amplificador como ya se ha hecho antes, buscando “opamp” en el “Component Catalog”, y arrastrándolo sobre el esquemático. La configuración es la misma que se ha realizado con los dos amplificadores operacionales de la etapa de filtrado, solamente se le ha nombrado como “AMPLIFICADOR3”. Lo siguiente es añadir tres pines analógicos, iguales que los añadidos anteriormente, para ello se busca “analog pin” y se arrastra el componente en el esquemático. Su configuración es idéntica a la ya mostrada para los otros pines analógicos.

En el fichero “Pusaciones.cydwr” hay que asignar los puertos a los pines del amplificador operacional, quedando así:

- Amplificador 3 – Entrada no inversora: P1[0]
- Amplificador 3 – Entrada inversora: P1[1]
- Amplificador 3 – Salida: P1[2]

Se compila mediante “Build > Build Pusaciones”, y en el “main.c” se añade justo debajo de donde se han inicializado los otros dos amplificadores la siguiente línea:

```
AMPLIFICADOR3_Start();
```

La tensión de referencia que se va a introducir por la entrada inversora del amplificador operacional va a ser de 10 mV, por lo que aprovechando que el PSoC puede generar una intensidad, se va a sacar una intensidad de 250 μA por un pin analógico, y mediante una resistencia de 391 Ω se obtendrán los 10 mV, que se podrán introducir por un pin analógico en la entrada inversora del amplificador operacional.

Para ello se busca en el “Component Catalog” el siguiente componente “dac”, y se arrastra el componente llamado Current DAC (7 or 8-bit) [v1.10] al esquemático. Se hace doble click sobre él, como nombre se asigna “IDAC”, en “Polarity” se elige “Positive” para que actúe como fuente, en “Range” se elige de 0 a 306 μA , y en “Value” se pone “250,8”.

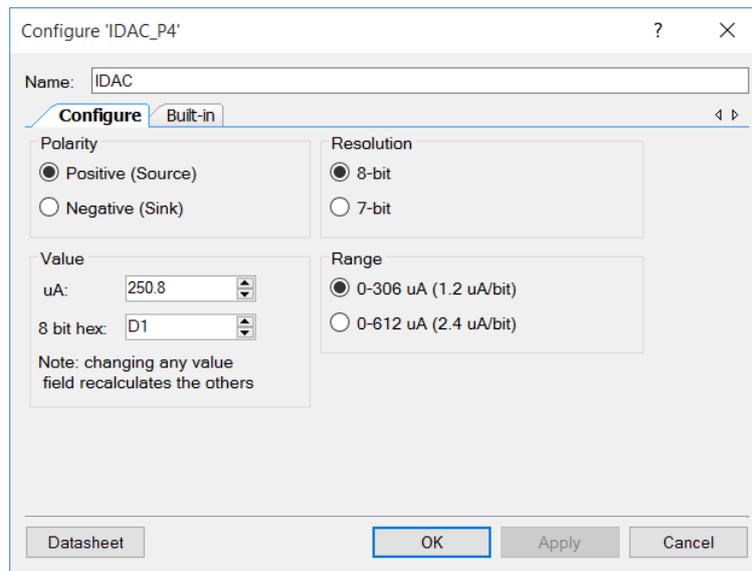


Ilustración 4-14. Ventana de configuración del IDAC

Para poder sacar la corriente del IDAC fuera del PSoC se tiene que mandar a un puerto, así que nuevamente se arrastra otro pin analógico al esquemático, y se conecta con el “IDAC”. Esta vez el puerto al que se le asignará el pin analógico que se ha conectado al IDAC es el P0[5].

Tras compilar con “Build > Build Pulsaciones”, en el “main.c”, debajo de donde se ha inicializado el “AMPLIFICADOR3” se debe añadir lo siguiente para inicializar el IDAC:

```
IDAC_Start();
```

Ahora ya se tiene al amplificador comparando correctamente las dos señales, y generando una salida digital. El problema es que la salida del amplificador es analógica, y el siguiente componente que se va a utilizar espera una entrada digital, por lo que hay que hacer el “truco” de sacar la salida del amplificador a un pin analógico (esto ya se ha hecho), y puentear ese pin a un nuevo pin del PSoC, el cual será definido como un pin de entrada digital. Para ello hay que irse al “Component Catalog”, buscar “digital input pin”, y arrastrar

el componente llamado “Digital Input Pin [v2.10]” al esquemático. Tras eso hay que dirigirse a “Pulsaciones.cydwr” y asignarle el puerto P1[6]. Por este pin se recibirá un “1” cada vez que se detecte una pulsación, y un “0” el resto del tiempo.

Con esto ya se habrían definido todas las entradas y las salidas físicas del PSoC, así que el resto de componentes que hay que añadir y configurar son todos componentes digitales internos del PSoC.

Ahora hay que añadir el componente que va a permitir realizar la lógica para medir las pulsaciones por minuto. Hay que añadir un temporizador, el cual tendrá dos entradas. En la primera entrada se conectará una señal de reloj, y en la otra entrada se conectará el pin digital que se acaba de añadir. El temporizador contará los pulsos de reloj que ha recibido entre dos “1” que reciba por la entrada del pin digital.

A la salida del temporizador se conectarán dos interrupciones. La primera se activará con cada “1” recibido por la entrada del pin digital. La otra se activará cuando hayan pasado más de dos segundos sin recibir un “1”, para evitar falsas medidas.

Para comenzar a añadir los componentes, hay que irse a “Component Catalog” y buscar “timer”, arrastrar el componente llamado “Timer [v2.70]” al esquemático, hacer doble click y configurarlo de la siguiente manera:

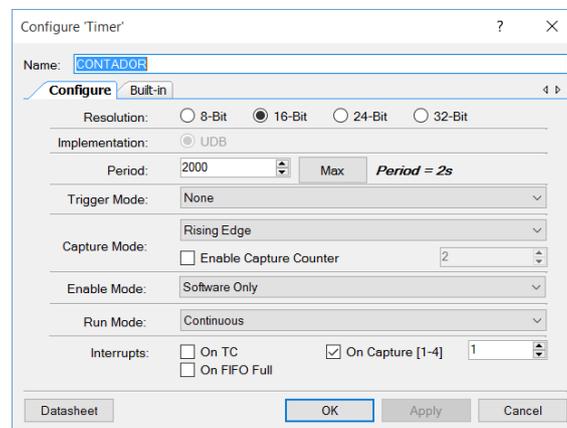


Ilustración 4-15. Ventana de configuración del Timer

Se establece una resolución de 16 bits, para poder contar más tiempo. Se le pone como nombre “CONTADOR”. En “Period” se establece 2000, para que el registro máximo de tiempo no exceda de los 2 segundos. Por último se habilitan las interrupciones cuando se produzca “On Capture [1-4]”. Se pulsa en “OK”, y se conecta el pin digital que se había añadido a la entrada del Timer llamada “capture”.

Se compila mediante “Build > Build Pulsaciones” y en el “main.c”, debajo de donde se había inicializado el IDAC se añade lo siguiente:

CONTADOR_Start();

Ahora se busca en el “Component Catalog” un “clock”, y se arrastra el componente “Clock [v2.20]” al esquemático. Se conecta el reloj a la entrada del Timer llamada “clock”. Se hace doble click y se configura del siguiente modo, llamándolo “Clock” y estableciendo la frecuencia en 1 kHz:

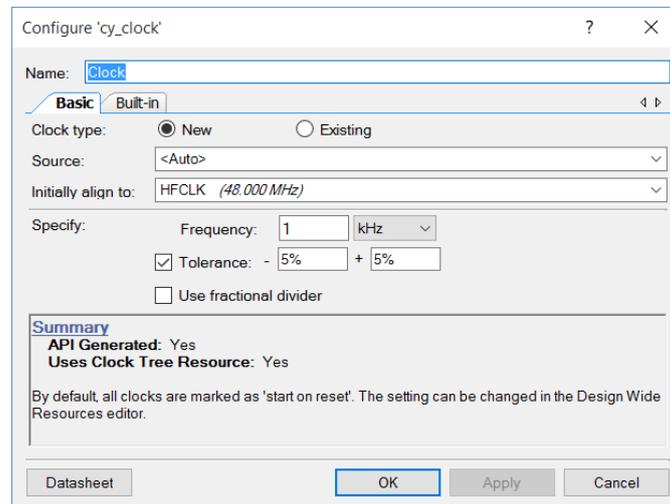


Ilustración 4-16. Ventana de configuración del Clock

Se compila mediante “Build > Build Pulsaciones” y en el “main.c”, debajo de donde se había inicializado el Timer se añade lo siguiente:

Clock_Start();

Ahora se añade un “0” lógico al esquemático, buscando en el “Component Catalog” “logic low”, y arrastrando el componente “Logic Low ‘0’ [v1.0]” al esquemático. Este componente no precisa configuración, así que directamente se conecta a la entrada del Timer llamada “reset”.

Por último se añaden las dos interrupciones. Buscando en el “Component Catalog” “interrupt”, y arrastrando el componente “Interrupt [v1.70]” al esquemático. Se hace doble click, y simplemente se le cambia el nombre por “PULSE_CUNT”. Ahora se conecta a la salida del Timer llamada “interrupt”.

La segunda interrupción se añade igual, pero esta vez como nombre se le pone “NO_HAY_DEDO”, y se conecta a la salida del Timer llamada “tc”.

Se compila mediante “Build > Build Pulsaciones” y en el “main.c”, debajo de donde se había inicializado el Clock se añade lo siguiente:

CyGlobalIntEnable;

```
PULSE_COUNT_StartEx(&dedo_detectado);
```

```
NO_HAY_DEDO_StartEx(&no_se_detecta_dedo);
```

La primera línea habilita las interrupciones globales, necesarias para poder recibir interrupciones. Las otras dos líneas hacen referencia a dos funciones, `CY_ISR(dedo_detectado)` y `CY_ISR(no_se_detecta_dedo)`, las cuales se encargarán de manejar las interrupciones para poder calcular y enviar correctamente las pulsaciones por Bluetooth. Estas funciones se explicarán tras describir el último componente que se añadirá al esquemático, el componente *Bluetooth Low Energy*.

Por lo tanto, el esquema que se tendría para esta parte es el siguiente:

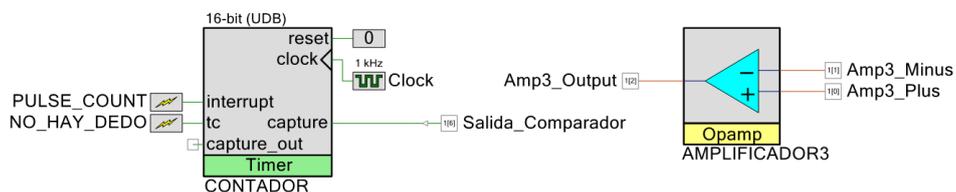


Ilustración 4-17. Esquemático de la etapa para calcular el ritmo cardíaco

4.3.4 Configuración del componente utilizado para realizar la comunicación por Bluetooth Low Energy

Ahora que ya se han configurado todos los componentes que se encargan de procesar las señales que contienen el pulso, se va a añadir el componente que se encarga de gestionar las comunicaciones del PSoC.

Hay que irse a “Component Catalog” y buscar “bluetooth”. Se arrastra el componente “Bluetooth Low Energy (BLE) [v2.10]” al esquemático, y se hace doble click para configurarlo.

En la ventana que se abre, se elige “Profile Collection” para que el PSoC pueda utilizar alguno de los perfiles que permite el estándar Bluetooth. En este caso, aunque se cuenta con bastantes perfiles, se va a crear uno personalizado, por lo que dentro de “Profile configuration”, en “Profile” se elige “Custom”. Dado que el PSoC va a tener el dato con las pulsaciones registradas, va a hacer de servidor, por lo que en “Profile role” se elige “Server (GATT Server)”, y por último, dado que se quiere que el PSoC sea un dispositivo periférico, y que el rol central lo lleve el dispositivo móvil, en “GAP role” se elige “Peripheral”.

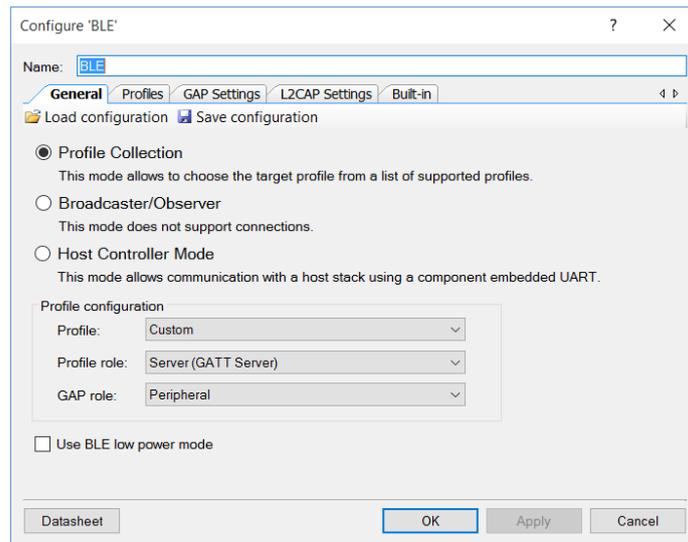


Ilustración 4-18. Ventana de configuración General del BLE

Ahora hay que irse a la pestaña de “GAP Settings”, y en “Device name:” poner “Sensor de ritmo cardiaco”, este será el nombre del dispositivo. Ahora hay que irse a la subpestaña “Advertisement packet”, y seleccionar “Local Name”, para que cuando se escanee al PSoC desde el dispositivo móvil, en el paquete de advertising el PSoC mande el nombre que se le ha puesto antes en “Device name”.

En la subpestaña “Advertisement settings” se podrá configurar el tiempo que durará el modo advertising, y en la subpestaña “Peripheral preferred connection parameters” se podrá configurar el tiempo mínimo y máximo entre conexión y conexión con el dispositivo móvil. Así como el timeout antes de dar por terminada la conexión.

Ahora hay que dirigirse a la subpestaña “Security”, y en “Security level” establecer “No Security (No authentication, No encryption)”. En “I/O capabilities” se pone “No Input No Output”, y en “Bonding requirement” se selecciona “No Bonding”, ya que como se está haciendo un sensor de ritmo cardiaco, se quiere crear un dispositivo al que sea muy sencillo conectarse y se ha optado por eliminar las medidas de autenticación para acelerar el proceso de conexión.

Por último, hay que dirigirse a la pestaña “Profiles”, en la cual se configurarán los servicios y características del perfil Bluetooth. Como se va a crear un perfil personalizado, hay que hacer click derecho sobre “Custom Service”, pulsar sobre “Rename” y poner “Pulsaciones”. Este será el nombre del servicio personalizado. En el campo de la derecha donde pone UUID se pondrá la UUID personalizada que se quiera para identificar al servicio, en este caso y por simplicidad se ha optado por una UUID de 128 bits con un valor de “00000001-0001-0001-0001-000000000001”.

Hay que repetir el proceso con la característica, haciendo click derecho sobre “Custom Characteristic”, pulsando sobre “Rename” y poniendo “Pulsaciones_por_minuto”. Este será el nombre de la característica personalizada, la cual contendrá el dato con las pulsaciones por minuto. En el campo de la derecha donde pone UUID se pondrá la UUID personalizada que se quiera para identificar a la característica, en este caso y por simplicidad se ha optado por una UUID de 128 bits con un valor de “00000002-0002-0002-0002-000000000002”.

En “New field” se podrá elegir el tipo de dato que almacenará la característica, así como su longitud y su valor. En este caso, se ha dejado el tipo de dato en “uint8”, y el valor por defecto se ha establecido en 0.

Abajo en las propiedades de la característica, se ha marcado “Read” y “Notify”, para que cada vez que la característica reciba un cambio, el PSoC pueda notificar el nuevo valor de la característica al dispositivo móvil, en vez de que el dispositivo móvil tenga que estar preguntando periódicamente por el valor de la característica para poder actualizarlo.

Al marcar “Notify”, se creará un nuevo descriptor llamado “Client Characteristic Configuration”, el cual debe ser escrito por el dispositivo móvil para habilitar las notificaciones en el PSoC.

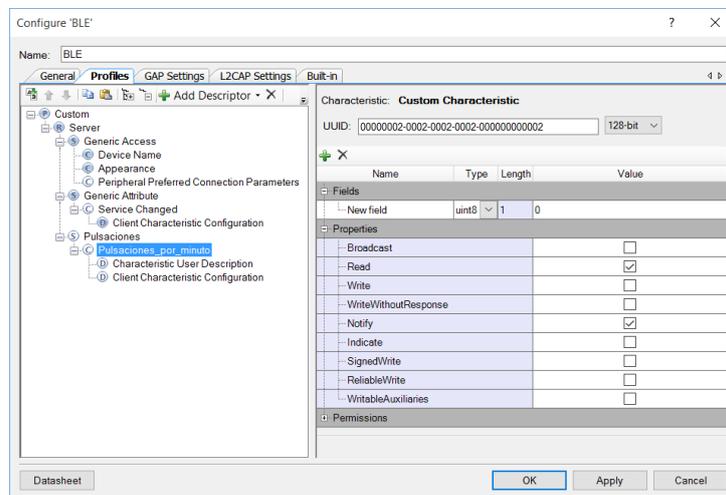


Ilustración 4-19. Ventana de configuración del perfil de bluetooth

Se pulsa “OK”, y el componente BLE ya estará correctamente configurado. Ahora hay que pulsar en “Build > Build Pulsaciones”, y si no hay ningún error de compilación ya se está en condiciones de irse al “main.c” a configurar el comportamiento del módulo BLE, así como a configurar el comportamiento de las interrupciones del Timer.

Lo primero que se debe hacer en el “main.c” será añadir debajo de las dos interrupciones que se habían inicializado lo siguiente:

CyBle_Start(StackEventHandler);

Esta función iniciará el módulo Bluetooth del PSoC, y dejará registrada la función “StackEventHandler”, para que cada vez que ocurra un evento relacionado con Bluetooth, se ejecute uno de los casos que se han definido dentro de la función.

Esta función recibe como primer parámetro una variable de tipo “uint32” llamada “eventCode”, el cual se analizará mediante una sentencia “switch” para evaluar el tipo de evento que se acaba de producir. Como segundo parámetro recibe un dato de tipo “void” llamado “*eventParam”, el cual contiene un paquete de datos con toda la información que el dispositivo móvil intenta escribir en el PSoC, ya que aparte del valor que el dispositivo móvil quiere escribir en el PSoC, en el paquete también se encuentran datos referentes a la característica en la que se quiere escribir el dato. En este proyecto se han distinguido dos casos fundamentales, los cuales dan la funcionalidad básica para que el PSoC pueda realizar las tareas que debe:

- **case CYBLE_EVT_STACK_ON:** este evento indica que el stack del Bluetooth se ha iniciado correctamente, por lo que se puede aprovechar para iniciar el modo advertising, en caso de que se quiera que el dispositivo sea visible inmediatamente. El PSoC cuenta con dos modos de advertising, el modo rápido y el modo lento. Normalmente se comienza con el modo rápido para que sea muy fácil encontrar al PSoC, y si en el tiempo que dura este modo no se realiza ninguna conexión, se pasa al modo lento, el cual también permite la conexión, pero aumenta los tiempos entre los paquetes de advertising. Por lo tanto, el modo rápido consume más energía que el modo lento. En este caso sólo se va a utilizar el modo rápido, ya que se quiere que el PSoC se pueda encontrar rápidamente. Para ello se utiliza la siguiente función:

CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);

- **case CYBLE_EVT_GATTS_WRITE_REQ:** este caso indica que se acaba de recibir una petición de escritura por parte del dispositivo que se ha conectado por Bluetooth al PSoC. En este proyecto, la única petición de escritura que se puede recibir, será la que escriba en el descriptor que habilita las notificaciones de la característica en la que se almacena el valor de las pulsaciones por minuto. Por lo tanto, lo primero que habrá que hacer es crear una variable en la que se almacenará el paquete de datos que se ha recibido por parte del dispositivo móvil:

CYBLE_GATTS_WRITE_REQ_PARAMT *wrReqParam;

Ahora se almacena en ella el paquete de datos recibido:

```
wrReqParam = (CYBLE_GATTS_WRITE_REQ_PARAM_t *) eventParam;
```

Una vez guardado el paquete de datos, se puede proceder a comprobar si el dispositivo móvil quiere escribir en el descriptor que habilita las notificaciones, para ello se comprueba si el “handle” del paquete de datos recibido coincide con el “handle” del descriptor, ya que todos los servicios, características y descriptores tienen un handle asociado:

```
if(CYBLE_PULSACIONES_PULSACIONES_POR_MINUTO_CLIENT_CHARACTERISTIC_CONFIGURATION_DESC_HANDLE == wrReqParam->handleValPair.attrHandle)
```

Como se puede ver, se ha accedido a la variable en la que se ha guardado el paquete de datos recibido, y se ha accedido al valor del handle al que va referido el paquete de datos. Por lo tanto, si efectivamente el dispositivo móvil quiere escribir en el descriptor, querrá habilitar o deshabilitar las notificaciones, por lo que se modifica una variable que se ha creado a modo de flag para poder conocer si las notificaciones están activadas o desactivadas.

```
*puntero_notificaciones = wrReqParam->handleValPair.value.val[0x00];
```

Lo que se hace es almacenar en el flag el valor que ha escrito el usuario, ya que si quiere habilitar las notificaciones habrá mandado un 1, y si quiere desactivarlas habrá mandado un 0, por lo que así ya se puede conocer el estado que se le asignará al flag.

Ahora se modifica por fin el valor del CCCD almacenado en el servidor GATT del PSoC, para que se quede reflejado que se han habilitado las notificaciones

```
CyBle_GattsWriteAttributeValue(&wrReqParam->handleValPair, 0, &cyBle_connHandle, CYBLE_GATT_DB_LOCALLY_INITIATED);
```

Lo primero ha sido llamar a la función que se encarga de escribir los valores en el servidor GATT. Los dos parámetros fundamentales que se le han pasado a esta función han sido en primer lugar el valor que ha escrito el dispositivo móvil, y en segundo lugar el handle que se encarga de manejar la conexión Bluetooth.

Por último se llama a una función que va a generar una respuesta para el dispositivo móvil que ha intentado escribir en el PSoC:

```
CyBle_GattsWriteRsp(cyBle_connHandle);
```

Una vez descritos los distintos casos que se van a tratar cuando se reciban eventos de Bluetooth, hay que añadir una función más en el bucle que se ejecuta indefinidamente en la función “main”. Dado que este bucle se va a ejecutar de forma indefinida, mediante esta función se va a hacer que se procesen todos los eventos de Bluetooth que estén pendientes de procesar:

CyBle_ProcessEvents();

Ahora se va a explicar lo que ocurre en las dos interrupciones que se han definido anteriormente.

En la primera, tratada en “CY_ISR(dedo_detectado)”, se va a entrar cada vez que se reciba una pulsación. Lo primero que se hará será resetear el flag que indica que han pasado más de dos segundos. Este flag se utiliza para que cuando pasen más de dos segundos el PSoC indique que hay 0 pulsaciones por minuto, pero que si vuelven a pasar otros dos segundos sin recibir ninguna pulsación, el PSoC no vuelva a enviar otra vez que hay 0 pulsaciones por minuto.

A continuación, se “limpia” la interrupción. Para ello, hay que leer un registro del “Timer”, el cual desactiva la interrupción en cuanto es leído:

CONTADOR_ReadStatusRegister();

Lo siguiente es calcular las pulsaciones por minuto. Para ello, hay que leer el valor almacenado en el registro tras recibir un flanco de subida en la entrada “Capture”. Dado que se ha establecido que el periodo del contador tendrá un valor de 2000, y el registro va contando desde 2000 hasta 0, cuando se lea el valor del registro se deberá restarle a 2000 el valor del registro, con lo cual se obtendrán los pulsos de reloj que se han recibido por la entrada “Clock”.

Para calcular las pulsaciones por minuto, como se sabe que en un segundo el reloj va a introducir 1000 pulsos (el reloj funciona a 1 kHz), sabiendo cuantos pulsos de reloj se han recibido entre dos latidos del corazón, se puede calcular el tiempo transcurrido entre los dos pulsos, por lo que si ahora se divide 60000 entre 2000-Pulsos_de_reloj se obtendrán las pulsaciones por minuto. Se ha utilizado 60000 en vez de 60 segundos porque el periodo del temporizador lee 1000 pulsos por segundo, en vez de 1 pulso por segundo.

Ahora se comprueba si las pulsaciones por minuto se encuentran en un rango aceptable, para evitar ciertas medidas falsas introducidas por el sensor. Se ha considerado como aceptable un valor mínimo de 30 pulsaciones por minuto, y un valor máximo de 210 pulsaciones por minuto.

Si las pulsaciones están dentro del rango, se podrá proceder a almacenarlas en la base de datos GATT. Para ello hay que crear una variable (`datos_a_escribir`) de tipo “CYBLE_GATT_HANDLE_VALUE_PAIR_T”, ya que esta vez no se cuenta con un dato escrito por un dispositivo móvil, sino que hay que generar un dato que se pueda escribir en el servidor GATT del PSoC.

Lo siguiente es asociarle a esta variable todos los datos referentes a la característica en la que se quiere escribir el valor de las pulsaciones, como son el “handle” de la característica (`datos_a_escribir.attrHandle = CYBLE_PULSACIONES_POR_MINUTO_CHAR_HANDLE`), el valor de la misma (`datos_a_escribir.value.val`), y la longitud del tipo de dato que se va a almacenar (`datos_a_escribir.value.len`).

Ahora se escribe por fin en la base de datos GATT mediante esta función, la cual es idéntica a la vista anteriormente para escribir un valor en una base de datos GATT:

```
CyBle_GattsWriteAttributeValue(&datos_a_escribir, 0,
&cyble_connHandle, CYBLE_GATT_DB_LOCALLY_INITIATED);
```

Ya se ha escrito el valor de las pulsaciones en la base de datos GATT del PSoC, pero aunque si un dispositivo quisiera leer la característica podría hacerlo y ver las pulsaciones actualizadas, se quiere comprobar si el flag de las notificaciones está habilitado para poder notificar al dispositivo móvil de que hay nuevos datos.

Para ello simplemente hay que comprobar si el flag que se ha modificado antes al recibir la petición de escritura en el descriptor de las notificaciones se encuentra a 1. Si es así, se repiten unos pasos muy similares a los descritos arriba para escribir en la base de datos GATT.

En primer lugar, se crea una variable (`notificationHandle`) de tipo “CYBLE_GATTS_HANDLE_VALUE_NTF_T”, y se almacena en ella el handle de la característica que se quiere notificar al dispositivo (`notificationHandle.attrHandle = CYBLE_PULSACIONES_POR_MINUTO_CHAR_HANDLE`), el valor de la característica (`notificationHandle.value.val`) y la longitud del tipo de dato que se va a notificar (`notificationHandle.value.len`).

Una vez que se tiene listo el tipo de dato que se va a notificar, se puede llamar a la función que se va a encargar de realizar la notificación, a la cual sólo se le pasa el handle de la conexión Bluetooth, y la variable que se ha creado para almacenar el dato de la notificación:

```
CyBle_GattsNotification(cyBle_connHandle, &notificationHandle);
```

Tras haber notificado el dato al dispositivo móvil (y si no se hubiera hecho también), se debe resetear la cuenta del Timer, para que pueda volver a comenzar desde 1999 hasta 0 (2000 valores):

```
CONTADOR_WriteCounter(1999);
```

En cuanto a la otra interrupción, se ha implementado en “CY_ISR(no_se_detecta_dedo)”. Esta interrupción se activa cuando el contador del Timer llega a 0, lo cual indica que han pasado 2 segundos sin recibir ninguna pulsación por la entrada “Capture”, por lo que se debe actualizar el dato de las pulsaciones por minuto a 0.

En esta interrupción se ha añadido un flag para que si no se reciben nuevas pulsaciones no se esté continuamente mandando un 0 al dispositivo móvil.

En cuanto al funcionamiento para actualizar la característica de la base de datos GATT del PSoC y para notificar al dispositivo móvil, las funciones y los pasos a utilizar son los mismos, lo único que en vez de tener que calcular las pulsaciones por minuto leyendo el valor del registro del Timer, directamente se puede establecer el valor de las pulsaciones por minuto a 0.

4.3.5 Esquema final y lista de pines

Por último, una vez que se han colocado todos los componentes y que se ha realizado toda la configuración y la programación correspondiente, se va a mostrar el esquema final utilizado en el PSoC Creator, así como la lista de pines usada y el puerto al que se han asociado.

Un aspecto no comentado todavía, es que por motivos de depuración, durante el desarrollo de este proyecto se ha utilizado un pin extra llamado “LED_Pulsaciones” conectado a un LED interno del PSoC mediante el puerto P2[6], el cual se ha conectado en la entrada “Capture” del “Timer”. Así, cada vez que se detecta una pulsación, el LED se enciende y se puede observar como brilla mirando la placa de desarrollo sobre la que está situado el PSoC.

Esto permite conocer si la señal digital que se obtiene del amplificador operacional en modo comparador está funcionando correctamente, aun sin tener desarrollada o funcionando correctamente la aplicación de Android. Por ello, y aunque no es un objetivo del proyecto, el LED se ha dejado en el proyecto final, para que cualquiera interesado en

reproducir los pasos seguidos en este proyecto pueda contar con una herramienta más que le ayude en el desarrollo del mismo.

Alias	Name	Port	Pin	Lock
Amp1_Minus		P2[1] OA0:vminus, SCB0:spi_select[2]	38	<input checked="" type="checkbox"/>
Amp1_Output		P2[2] OA0:vout_10x, SRSS:wakeup, SCB0:spi_select[3]	39	<input checked="" type="checkbox"/>
Amp1_Plus		P2[0] OA0:vplus, SCB0:spi_select[1]	37	<input checked="" type="checkbox"/>
Amp2_Minus		P2[4] OA1:vminus	41	<input checked="" type="checkbox"/>
Amp2_Output		P2[3] OA1:vout_10x, SRSS:ext_clk_lf	40	<input checked="" type="checkbox"/>
Amp2_Plus		P2[5] OA1:vplus	42	<input checked="" type="checkbox"/>
Amp3_Minus		P1[1] OA2:vminus, TCFWM0:line_out_compl, LFCOMP:comp[1], SCB1:spi_select[1]	29	<input checked="" type="checkbox"/>
Amp3_Output		P1[2] OA2:vout_10x, TCFWM1:line_out, SCB1:spi_select[2]	30	<input checked="" type="checkbox"/>
Amp3_Plus		P1[0] OA2:vplus, TCFWM0:line_out, LFCOMP:comp[0], SRSS:ext_clk_lf	28	<input checked="" type="checkbox"/>
LED_Pulsaciones		P2[6] OA0:vplus_alt	43	<input checked="" type="checkbox"/>
Salida_Comparador		P1[6] OA2:vplus_alt, TCFWM3:line_out, SCB0:uart_rts, SCB0:spi_select[0]	34	<input checked="" type="checkbox"/>
Salida_IDAC		P0[5] LFCOMP:in_n[1], TCFWM1:line_out_compl, SCB0:uart_tx, SCB0:i2c_scl, SCB0:spi_miso	25	<input checked="" type="checkbox"/>

Ilustración 4-20. Tabla con la asignación de puertos a los pines del esquemático

En el esquemático se han dejado los comentarios que se han considerado oportunos para la comprensión del mismo.

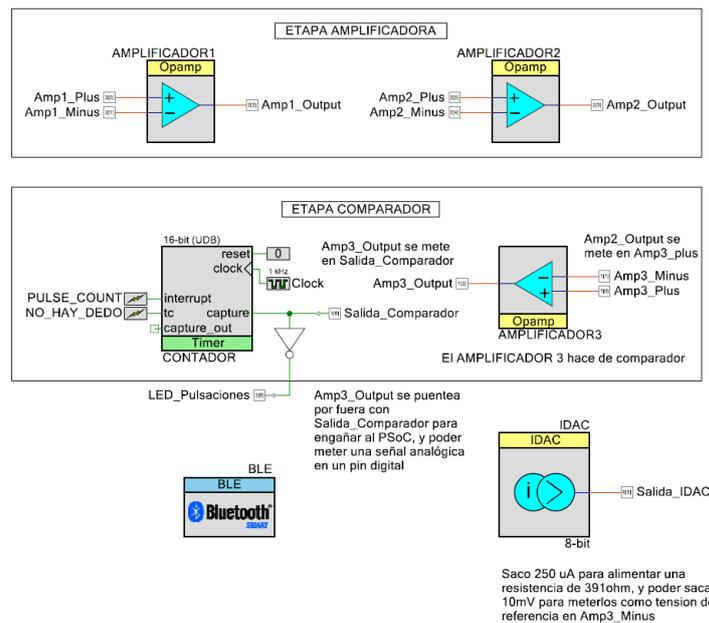


Ilustración 4-21. Esquemático final utilizado en el PSoc Creator

4.4 Descripción de la App desarrollada en Android

Tras haber montado el sistema de medida y haber sido capaces de programar el PSoc para que obtenga las pulsaciones por minuto y las envíe por Bluetooth, hay que crear la aplicación para Android que permita recibir las pulsaciones del PSoc por Bluetooth y mostrarlas en tiempo real.

4.4.1 Creación del proyecto en Android Studio

Lo primero será crear un nuevo proyecto siguiendo los pasos explicados en el capítulo anterior. Para ello, tras abrir Android Studio, se pulsa en “Start a new Android Studio project”. En la primera ventana se establece el nombre de la aplicación. En este proyecto se ha utilizado como nombre “Pulsaciones”. Se elige la carpeta donde guardar el proyecto y se pulsa en “Next”.

Ahora se tiene que elegir la plataforma o plataformas en las que se ejecutará la aplicación. En este proyecto se ha elegido “Phone and Tablet”, y la versión mínima del SDK se ha establecido en la API 19, correspondiente a Android 4.4 KitKat, dado que esta versión soporta el uso del Bluetooth Low Energy. Una vez elegida se hace click sobre “Next”.

La siguiente pantalla permite elegir el tipo de Activity con el que comenzará la aplicación. En este proyecto se ha elegido la “Blank Activity”. Por lo que se selecciona y se hace click en “Next”.

Se llegará a la última pantalla, en la que se debe dar un nombre a la actividad. En este proyecto se ha dejado el nombre que viene por defecto, “MainActivity”. Se hace click en “Finish”, y comenzará la creación del proyecto y su posterior compilación para verificar que todos los archivos se han creado correctamente.

Cuando la creación del proyecto haya terminado, en la ventana principal de Android Studio se habrá abierto el fichero “MainActivity.java”.

4.4.2 Estructura de la aplicación de Android

Con el proyecto ya creado, se van a comentar los distintos ficheros de los que va a estar formado este proyecto, ya que van a ser los que definan la interfaz de la aplicación y las tareas que va a llevar a cabo.

El proyecto se ha dividido en cuatro archivos “.java”:

- MainActivity.java: este archivo contiene la actividad principal, y es el primero que se ejecuta cuando se abre la aplicación. Se va a encargar de mostrar la interfaz de usuario, lanzar el servicio del Bluetooth, lanzar la orden de conectarse al PSoC, actualizar las pulsaciones por minuto en la interfaz gráfica cuando las reciba del servicio de Bluetooth, y guardar el valor de las pulsaciones en una base de datos.
- BluetoothLeService.java: este archivo contiene la definición del servicio que se va a ejecutar de forma paralela a la actividad principal [24]. Este servicio es el que

realmente va a realizar todas las operaciones con Bluetooth. Se va a encargar de iniciar la conexión con el PSoC, de descubrir sus servicios y características, de habilitar las notificaciones y de recibir el valor de las pulsaciones que envíe el PSoC. Este servicio será el intermediario a la hora de transferir datos y gestionar la conexión entre la actividad principal y el PSoC.

- `BaseDeDatos.java`: en este archivo se va a definir la base de datos SQL, indicando el nombre que tendrá la nueva tabla que se cree en la base de datos, así como el tipo de datos que debe almacenar la tabla. Esta base de datos se utilizará para almacenar un histórico con las pulsaciones por minuto recibidas por la aplicación.
- `MyContentProvider.java`: este archivo va a hacer de intermediario entre la base de datos y la actividad principal. Va a crear un content provider que se deberá utilizar para introducir nuevos datos en la base de datos. Este content provider también proveerá métodos para poder acceder a los datos desde una aplicación externa a ésta.

Junto con estos archivos, se contará con un “`activity_main.xml`” en el que se podrá diseñar la interfaz de la aplicación, y con un “`AndroidManifest.xml`”, en el que se podrá asegurar que las actividades, servicios y content providers se tendrán en cuenta de forma correcta a la hora de compilar la aplicación.

4.4.3 Programación del archivo `MainActivity.java`

En este archivo se va a comenzar creando una clase llamada “`MainActivity`”, que extiende de la clase “`ActionBarActivity`”.

Dentro de esta clase, lo primero será definir las variables que se van a utilizar en la actividad. Las dos primeras se llaman “`Pulsaciones`” y “`Calidad`”, y serán de tipo “`TextView`”. Van a permitir modificar posteriormente los valores de las pulsaciones por minuto en la interfaz de la aplicación.

La tercera variable es el objeto “`mBluetoothAdapter`”. Este objeto es de tipo “`BluetoothAdapter`”, y va a permitir que la aplicación pueda utilizar el Bluetooth del dispositivo. A su vez, se ha creado de tipo “`public static`” para que pueda ser utilizada desde “`BluetoothLeService.java`” sin necesidad de crear un objeto.

La cuarta variable es el objeto “`mBluetoothLeService`”. Este objeto es de tipo “`BluetoothLeService`” (el servicio que se va a ejecutar en segundo plano), por lo que se utilizará para ejecutar métodos definidos en “`BluetoothLeService.java`”, por lo tanto se

utilizará cuando se quiera realizar una comunicación con el servicio de Bluetooth (que va a estar ejecutándose en segundo plano) para realizar una conexión con el PSoC.

La última variable es el objeto “mServiceConnection”. Este objeto es de tipo “ServiceConnection”, y se va a utilizar para manejar los eventos producidos cuando el servicio se cree o se destruya.

El primer método que hay que tratar es el método “onCreate”, el cual se ejecuta cuando se abra la aplicación y se lance la actividad principal. Dentro de este método se establecerá en primer lugar la apariencia de la aplicación, usando el archivo activity_main.xml para cargar los datos de la interfaz mediante el siguiente comando:

```
setContentView(R.layout.activity_main);
```

En cuanto se cree la interfaz, lo siguiente es obtener la referencia a los dos “TextView” que se han creado en la interfaz, esto va a permitir modificarlos posteriormente. La referencia se obtiene mediante este código:

```
Pulsaciones = (TextView)findViewById(R.id.pulsaciones);
```

```
Calidad = (TextView)findViewById(R.id.calidad);
```

Donde “pulsaciones” y “calidad” es el ID que se le ha dado a estos “TextView” en el archivo “activity_main.xml”.

Una vez que se ha tratado lo referente a la interfaz gráfica de la actividad, ya se puede iniciar el adaptador Bluetooth mediante:

```
final BluetoothManager bluetoothManager = (BluetoothManager)  
getSystemService(Context.BLUETOOTH_SERVICE);
```

```
mBluetoothAdapter = bluetoothManager.getAdapter();
```

Una vez obtenido al adaptador, se verifica si el Bluetooth está encendido comprobando si `mBluetoothAdapter.isEnabled()`

Si el Bluetooth se encuentra desactivado, se lanza un “intent” para que el sistema Android solicite habilitar el Bluetooth:

```
Intent enableBtIntent = new  
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
```

Lo último que hay que hacer dentro del método “onCreate” es crear un “intent” con el que se podrá lanzar el servicio de Bluetooth. A este intent habrá que indicarle la

actividad en la que se está y la clase en la que hay que basarse para crear el servicio. Por lo que se utilizará este código:

```
Intent gattServiceIntent = new Intent(this,  
BluetoothLeService.class);
```

Para lanzar el servicio, se utiliza “bindService”, ya que se quiere enlazar el servicio con esta actividad para poder intercambiar datos entre ellas. Además, esto hará que cuando desaparezca la actividad principal, el servicio también desaparezca, y no se quede en segundo plano. Como parámetro se ha pasado “mServiceConnection”, ya que se va a utilizar para poder enterarse de que el servicio se ha creado o se ha cerrado. El comando es el siguiente:

```
bindService(gattServiceIntent, mServiceConnection,  
BIND_AUTO_CREATE);
```

Una vez que el servicio se haya creado, el “mServiceConnection” llamará al método “onServiceConnected”, mediante el cual se obtendrá primero un objeto del servicio que se acaba de lanzar:

```
mBluetoothLeService = ((BluetoothLeService.LocalBinder)  
service).getService();
```

Tras haber obtenido el objeto del servicio, se llama a su método “connect” para conectarse al PSoC, para lo cual se le pasa como parámetro la dirección MAC del PSoC:

```
mBluetoothLeService.connect("00:A0:50:0F:13:1C");
```

Con esto el resto de la ejecución de la aplicación correría a cargo del servicio de Bluetooth que se ha lanzado, por lo que en la actividad principal sólo hay que quedarse a la espera de que el servicio de Bluetooth informe con alguna novedad sobre el estado de la conexión o mande algún dato.

El servicio de Bluetooth informará de estos eventos utilizando un “Broadcast Update”, por lo que la actividad principal deberá quedarse a la espera de recibir uno de estos mensajes mediante la creación de un “Intent Filter”.

Por lo tanto se definirá un intent filter llamado “intentFilter”, al que se le añadirán las “acciones” a las que debe prestarles atención. Para añadir una acción se utiliza:

```
intentFilter.addAction(BluetoothLeService.ACTION_GATT_CONNECTED)
```

Esto hará que el “intent filter” le preste atención a un “Broadcast Update” enviado por el servicio Bluetooth en el que se indique que el dispositivo móvil se acaba de conectar al PSoC.

Tras haber creado el filtro para prestarle atención solamente a los “Broadcast Update” que se hayan definido, se deberá crear un “BroadcastReceiver” llamado “mGattUpdateReceiver” en el cual se realizará una acción en función del “Broadcast Update” que se haya recibido.

Antes de definir las acciones que realizará el “BroadcastReceiver”, se deberá asociar el “Intent Filter” al “BroadcastReceiver. Esto se hará en un método llamado “onResume()”, el cual se ejecuta después del “onCreate()”, y justo antes de que la actividad comience a ejecutarse. Habrá que poner esto:

```
registerReceiver(mGattUpdateReceiver,  
makeGattUpdateIntentFilter());
```

Una vez que se haya asociado el “Intent Filter” con el “BroadcastReceiver”, hay que comentar lo que ocurrirá cuando se reciba uno de estos broadcast.

Dentro del “mGattUpdateReceiver” hay un método llamado “onReceive”, el cual se ejecuta cuando el “Intent Filter” detecta una acción de las que se le han asociado.

Mediante “intent.getAction()” se va comprobando en un bucle “if” el motivo por el que se ha recibido el “BroadcastReceiver”. La acción que interesa es “ACTION_DATA_AVAILABLE”, la cual indica que se acaba de recibir un dato del PSoC.

Cuando se reciba un dato del PSoC, será porque se acaba de recibir una pulsación, por lo que se almacenará el dato en una variable mediante el siguiente código:

```
final int dato_a_actualizar =  
intent.getIntExtra(BluetoothLeService.EXTRA_DATA, 0);
```

Como se puede ver, se tiene que extraer el dato que hay adjunto al intent. Una vez que se tiene el dato con las pulsaciones, hay que proceder a actualizar el “TextView” de la interfaz que se encarga de mostrar las pulsaciones por minuto. Para ello, mediante “runOnUiThread” se ejecuta lo siguiente:

```
Pulsaciones.setText(String.valueOf(dato_a_actualizar));
```

Hay que tener en cuenta que como el dato recibido es un “int”, y se quiere establecer el texto de un “Text View”, mediante el método “String.valueOf()” se puede extraer una cadena de texto de la variable de tipo “int”.

También se van a clasificar las pulsaciones recibidas en una serie de rangos, para que en función del valor de las pulsaciones, se actualice el “TextView” que indica el estado del ritmo cardíaco:

```
Calidad.setText("EXCELENTE");  
Calidad.setTextColor(Color.parseColor("#006633"));
```

Una vez que se ha actualizado la interfaz gráfica mostrando el valor más reciente de las pulsaciones, se va a guardar el dato de las pulsaciones en una base de datos, por lo que hará falta utilizar el “Content Provider” que se explicará más adelante.

Lo primero será crear un objeto de tipo “ContentValues” llamado “values”, al cual se le pasará como parámetros el nombre de la columna de la tabla de la base de datos en la que se quiere introducir el valor, y el dato que se quiere almacenar.

Después, se creará un objeto de tipo “ContentResolver” llamado “cr”, con el que se podrá llamar al método “insert” del “ContentProvider” que se explicará más adelante, al que se le pasará como parámetros la “URI” del “ContentProvider”, y el objeto “values” que se ha creado antes.

4.4.4 Programación del archivo BluetoothLeService.java

En este archivo lo primero será crear una clase llamada “BluetoothLeService”, la cual se extiende de “Service”, por lo tanto será una clase destinada a definir el servicio que se ha lanzado para controlar el Bluetooth. Lo siguiente será definir las variables que se van a utilizar en este fichero.

El primer objeto se llamará “mBinder” y será de tipo “IBinder”, el cual se utilizará cuando se cree o se destruya el servicio.

En segundo lugar se creará un objeto de tipo “BluetoothGatt” que se ha llamado “mBluetoothGatt”. Este objeto se usará para realizar acciones concretas sobre el PSoC, como puede ser mandarle una orden para conectarse a él, o mandarle una petición de escritura o de lectura en una característica.

El siguiente objeto se va a utilizar solamente una vez, para poder activar las notificaciones en el PSoC. Se llama “característica_pulsaciones” y es de tipo “BluetoothGattCharacteristic”. Junto a este objeto se definirán otros tres, llamados

“PULSACIONES_SERVICE”, “PULSACIONES_CHARACTERISTIC” y “DESCRIPTOR”, los cuales contendrán el UUID del servicio, característica y descriptor (respectivamente) que se han creado en el PSoC para almacenar el valor de las pulsaciones.

Por último, se definirán cinco cadenas de texto que contendrán el mensaje que el “BroadcastUpdate” de este servicio enviará, para que el “BroadcastReceiver” de la actividad principal pueda ir enterándose de los eventos que se van produciendo en lo que a la conexión Bluetooth con el PSoC se refiere. Las cadenas de texto que se han definido han sido “ACTION_GATT_CONNECTED” para cuando la aplicación se conecte por Bluetooth con el PSoC, “ACTION_GATT_DISCONNECTED” para cuando la aplicación se desconecte del PSoC, “ACTION_GATT_SERVICES_DISCOVERED” para cuando la aplicación haya terminado de escanear todos los servicios del PSoC, “ACTION_DATA_AVAILABLE” para cuando el PSoC envíe el valor de alguna característica, “EXTRA_DATA” para incluir la característica que se haya recibido del PSoC en el “intent” que se enviará a la actividad principal.

Tras definir las variables, se va a comentar los métodos creados en este archivo. Los dos primeros se van utilizar cuando se cree o se destruya el servicio, son “onBind” y “onUnbind”. “onBind” devolverá el objeto “mBinder”, el cual se usará en la actividad principal para poder comunicarse con el servicio. En cuanto a “onUnbind”, llamará a un método llamado “close()” que cerrará correctamente la conexión Bluetooth con el PSoC.

El siguiente método de interés es el método “connect(final String address)”, el cual es llamado desde la actividad principal cuando esta recibe el aviso de que el servicio se ha creado. En el método “connect” se va a crear un “device” de tipo “BluetoothDevice”, al que se le pasará la dirección MAC enviada desde la actividad principal, la cual se usará con el objeto “mBluetoothAdapter” que se ha creado en la actividad principal. Esto generará un objeto que contendrá toda la información necesaria para iniciar la conexión con el PSoC. Ahora sólo falta utilizar el objeto “mBluetoothGatt” para administrar las acciones que se quieren realizar sobre el PSoC, por lo que se utilizará el siguiente código:

```
mBluetoothGatt = device.connectGatt(this, false, mGattCallback);
```

Como se puede ver, el método “connectGatt” lanza un objeto llamado “mGattCallback” de la clase “BluetoothGattCallback”, la cual es una “callback function” que lanzará uno de sus métodos cuando se produzca algún evento en la conexión Bluetooth con el PSoC.

La última parte del archivo que queda por tratar es lo que ocurre en el objeto “mGattCallBack” cuando recibe algún evento del Bluetooth del dispositivo.

El primer método se llama “onConnectionStateChange”, y la aplicación entra en él cuando el estado de la conexión Bluetooth ha cambiado, por ello se deberá comprobar si el parámetro “newState” indica que el dispositivo móvil se acaba de conectar por Bluetooth al PSoC (BluetoothProfile.STATE_CONNECTED) o que el dispositivo móvil se acaba de desconectar (BluetoothProfile.STATE_DISCONNECTED). En caso de que el dispositivo móvil se acabe de conectar, se iniciará el proceso de descubrir los servicios del PSoC llamando al método “discoverServices()” del objeto “mBluetoothGatt” que se ha dicho que se utilizaría para ejecutar comandos de Bluetooth.

Cuando haya terminado el proceso de descubrir los servicios del PSoC, la callback function ejecutará automáticamente el método “onServicesDiscovered”. Indica que ya se ha terminado el proceso de descubrir los servicios, por lo que ya se pueden activar las notificaciones en el PSoC para que éste pueda mandar el valor actualizado de las pulsaciones cuando se produzca alguna.

Hay que tener en cuenta que el motivo por el que se ha realizado un escaneo antes de activar las notificaciones, es porque aunque se conozcan el UUID del servicio y de la característica a la que se quiere activarle las notificaciones, el stack Bluetooth de Android no puede crear directamente un objeto de tipo descriptor con los UUID que se conocen y escribir en el PSoC. Antes de realizar cualquier operación sobre un atributo del servidor (en este caso el PSoC), el dispositivo con Android necesita haber realizado un escaneo para haber indexado los manejadores o handles de todos los servicios del PSoC.

Por lo tanto, ahora ya se puede construir el objeto “característica_pulsaciones” con el UUID del servicio y de la característica que contiene el dato con las pulsaciones mediante:

```
característica_pulsaciones =  
mBluetoothGatt.getService(PULSACIONES_SERVICE).getCharacteristic  
(PULSACIONES_CHARACTERISTIC);
```

Ahora se completa el objeto “descriptor” creado anteriormente con los datos de la característica a la que se le quieren activar las notificaciones, ya que cada característica tiene su propio descriptor. Y se lanza el comando para escribir en el descriptor, utilizando para ello el objeto “mBluetoothGatt”, el cual se encarga de realizar directamente operaciones con el dispositivo Bluetooth al que el dispositivo móvil se haya conectado. Los fragmentos de código son los siguientes:

```
mBluetoothGatt.setCharacteristicNotification(característica_pulsaciones, true);
```

```
BluetoothGattDescriptor descriptor =  
caracteristica_pulsaciones.getDescriptor(DESCRIPTOR);  
descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_  
VALUE);  
  
mBluetoothGatt.writeDescriptor(descriptor);
```

El último método de la “callback function” al que hay que prestarle atención es a “onCharacteristicChanged”. Este método se lanzará cuando una característica del PSoC haya cambiado (en este caso la característica con el valor de las Pulsaciones), por lo que se deberá lanzar un “BroadcastUpdate” indicando que hay nueva información disponible, así como adjuntar el dato recibido.

Esto se hace en primer lugar creando un “intent” que contendrá la acción “ACTION_DATA_AVAILABLE”, para que el “intent filter” de la actividad principal pueda prestarle atención, dado que es uno de los filtros que se ha definido en el “intent filter” de la actividad principal.

A continuación se creará una variable llamada “data” de tipo “byte[]” para poder almacenar el dato que ha enviado el PSoC por Bluetooth. Lo siguiente será convertir ese byte de datos a una variable de tipo “int”, para que las pulsaciones puedan ser leídas en formato decimal fácilmente. El inconveniente aquí, es que el PSoC ha mandado el valor de las pulsaciones en un byte sin signo, pero en java los byte tienen signo, así que a la hora de almacenar el valor del byte dentro del “int” hay que utilizar lo siguiente:

```
int value = (data[0] & 0xFF);
```

Por último se adjunta el “value” en el “intent” mediante el método “putExtra”, el cual sigue un esquema de “Clave – Valor”, por lo que a la información almacenada en el intent se le asigna como clave “EXTRA_DATA”, y finalmente se lanza el intent con:

```
sendBroadcast(intent);
```

4.4.5 Programación del archivo BaseDeDatos.java

En este archivo se ha definido la base de datos que se va a utilizar para almacenar el valor de todas las pulsaciones que reciba la aplicación mediante Bluetooth.

Lo primero es crear una clase llamada “BaseDeDatos” que extienda de “SQLiteOpenHelper”. Esta clase contará con dos métodos fundamentales, “onCreate” y “onUpgrade”.

El método “onCreate” se deberá utilizar para lanzar el comando que cree la base de datos. Este comando creará una tabla llamada “Sensor” dentro de la base de datos. Esta tabla estará compuesta por 3 columnas. La primera será un ID, el cual se asignará y se incrementará automáticamente a cada nueva fila de la tabla. La segunda columna contendrá un timestamp con la fecha y la hora en la que se ha introducido el dato, y la última columna contendrá el dato con el valor de las pulsaciones recibidas. El comando por tanto es el siguiente:

```
db.execSQL("CREATE TABLE Sensor (_id INTEGER PRIMARY KEY
AUTOINCREMENT, created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
dato INT)");
```

El otro método es el llamado “onUpgrade”, el cual se utiliza solamente cuando se quiere realizar alguna modificación en la estructura de la tabla, como puede ser añadir alguna columna o cambiar algún tipo de dato. En este método se debe definir lo que se hará con la tabla anterior, y lanzar el comando que cree la nueva tabla. Por lo tanto, es un método utilizado para actualizar la tabla a una nueva versión.

4.4.6 Programación del archivo MyContentProvider.java

En este archivo se definirá el “Content Provider” que se usará como intermediario entre la base de datos y la aplicación, ya que su uso permitirá recuperar e introducir nuevos datos desde cualquier aplicación que utilice la interfaz proporcionada por el “Content Provider”.

Lo primero será crear una clase llamada “MyContentProvider” que extienda de la clase “ContentProvider”. Dentro se definirá la “URI” del content provider, la cual debe ser utilizada para poder acceder a la base de datos desde cualquier aplicación externa a este proyecto. La URI y su definición es la siguiente:

```
private static final String uri =
"content://elias.pulsaciones/pulsaciones";

public static final Uri CONTENT_URI = Uri.parse(uri);
```

Junto a esta URI se creará un objeto llamado “Sensor” que extienda de la clase “BaseColumns” donde se definirán dos variables para poder acceder a las columnas de la tabla de la base de datos, son “COL_TIMESTAMP” y “COL_PULSACIONES”. La columna “ID” no hace falta crearla, ya que va implícita en la definición de la clase “BaseColumns”.

También se creará un objeto llamado “usdbh” del tipo “BaseDeDatos”, así como un “UriMatcher” para que el content provider pueda interpretar la URI que reciba de la aplicación cuando sea utilizado para acceder a la base de datos.

En cuanto a los métodos con los que contará esta clase, el primero es el “onCreate()”, cuya misión es crear la base de datos, la cual se quedará iniciada en el objeto “usdbh”:

```
usdbh = new BaseDeDatos(getContext(), "BaseDeDatos", null, 1);
```

Para insertar datos en la tabla de la base de datos se utilizará el método “insert”, el cual obtiene la base de datos en modo escritura, inserta el valor en la base de datos, obtiene el ID del elemento que ha insertado mediante “db.insert”, y genera la URI del elemento que acaba de insertar.

Para leer datos de la tabla de la base de datos se utilizará el método “query”, el cual comprobará en primer lugar si la URI que se quiere leer hace referencia a una entrada de la tabla o a la tabla entera. En función de eso actualizará el valor de una variable que se introducirá en la función “db.query” que se utilizará para realizar la consulta en la base de datos.

Para eliminar o para actualizar elementos de la base de datos se utiliza el método “delete” y “update”, respectivamente. La estructura de estos métodos es similar a la utilizada en el método “query”, ya que lo primero que se deberá hacer es comprobar si la URI recibida se refiere a la tabla o a un elemento concreto de la tabla, para posteriormente lanzar el comando “db.delete” o “db.update”.

El último método de esta clase es el “getType”, el cual se utiliza para poder identificar el tipo de datos que devuelve el content provider. Para ello, mediante el método “uriMatcher” se identificará si la URI hace referencia a la tabla de la base de datos o a un elemento de la tabla, y en función de eso devolverá un String que representará el MIME Type de una lista de registros (si la URI hace referencia a la tabla), o el MIME Type de un único registro (si la URI hace referencia a un elemento de la tabla). Los String utilizados han sido:

```
"vnd.android.cursor.dir/vnd.elias.pulsaciones";
```

Para la tabla entera, y:

```
"vnd.android.cursor.item/vnd.elias.pulsaciones";
```

Para un solo elemento de la tabla.

4.4.7 Programación del archivo `activity_main.xml`

En este archivo se ha definido la interfaz gráfica de la aplicación. Dado que el lenguaje utilizado es XML, mediante una serie de etiquetas se van definiendo los distintos elementos que componen la interfaz.

En primer lugar hay que elegir como se van a disponer los elementos en la pantalla. En este proyecto se ha optado por un layout de tipo relativo, un “RelativeLayout”. Este tipo de layout distribuye los elementos mediante una serie de relaciones entre ellos, por lo que en vez de definir posiciones absolutas en la pantalla, se colocan una serie de elementos que se podrían considerar como los pilares de la interfaz, y el resto de elementos se van colocando en una posición relativa a estos, ya sea debajo, a la izquierda, o a una distancia determinada de ellos.

Como el objetivo de la interfaz gráfica es mostrar las pulsaciones por minuto con la mayor simpleza y rapidez posible, se ha optado por prescindir de botones para escanear dispositivos bluetooth, ni para gestionar la conexión. Así que mediante la lógica ya explicada que sigue la aplicación, unos segundos después de haber abierto la aplicación, el dispositivo con Android ya se habrá conectado por Bluetooth al PSoC y habrá activado las notificaciones, por lo que automáticamente se debería empezar a recibir en tiempo real las pulsaciones por minuto enviadas por el PSoC.

Así que en la interfaz tan sólo se utilizarán cuatro “TextView”, los cuales son capaces de mostrar un texto en la interfaz. Se han usado cuatro porque dos de ellos se van a utilizar a modo de texto que se quedará permanentemente indicando la unidad del dato que se está recibiendo, y los otros dos “TextView” se irán actualizando con el dato recibido, uno mostrará las pulsaciones por minuto, y el otro si el ritmo cardíaco es bueno o regular.

Por lo tanto, tras haberlos creado, hay que asignarles un “id”. Este “id” va a permitir que se puedan situar unos “TextView” en una cierta posición respecto a los otros “TextView”, además de utilizarse para que puedan ser identificados en los archivos .java para poder ser actualizados cuando se reciba un nuevo valor.

El resto de propiedades de los “TextView” son meramente estéticas, ya sea para definir el tamaño del texto, o para definir su color. Las más destacables son:

```
android:layout_width="wrap_content"
```

Mediante “wrap_content” se indica que el “TextView” debe “envolver” al texto que contenga. Esto hará que las dimensiones dependan del contenido del “TextView”.

```
android:textSize="20sp"
```

```
android:layout_marginTop="100dp"
```

En cuanto a “textSize” y “layout_marginTop”, lo destacable son las unidades que se han utilizado, ya que en vez de utilizar píxeles, se ha utilizado “sp” para el texto y “dp” para los márgenes. Estas unidades hacen que el tamaño no venga fijado por una cantidad determinada de píxeles, sino que vayan en función de la resolución y la densidad de píxeles que tenga cada pantalla.

Son por tanto una unidad relativa, que permite que los elementos se adapten correctamente a diferentes pantallas, ya que si se utilizaran directamente los píxeles, una pantalla con mucha resolución tendría una distancia en píxeles distinta de una pantalla que tenga menos resolución y para la que por tanto los mismos píxeles ocuparían un mayor espacio de la pantalla.

Tras definir la interfaz, al abrir la aplicación por defecto se solicitará que se active el Bluetooth si no estaba activado:

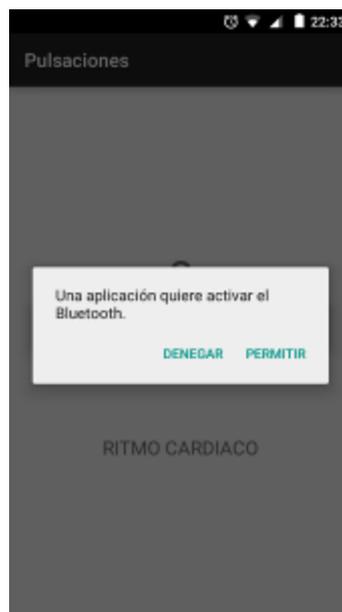


Ilustración 4-22. Aplicación solicitando que se active el Bluetooth

Tras darle a “PERMITIR”, la interfaz principal mostrará su apariencia por defecto, mostrando un valor de 0 “PULSACIONES POR MINUTO”:



Ilustración 4-23. Aplicación mostrando su apariencia por defecto

Si se ha conseguido conectar el dispositivo móvil al PSoC mediante Bluetooth, la aplicación comenzará a recibir los valores de las pulsaciones, mostrándolos así:

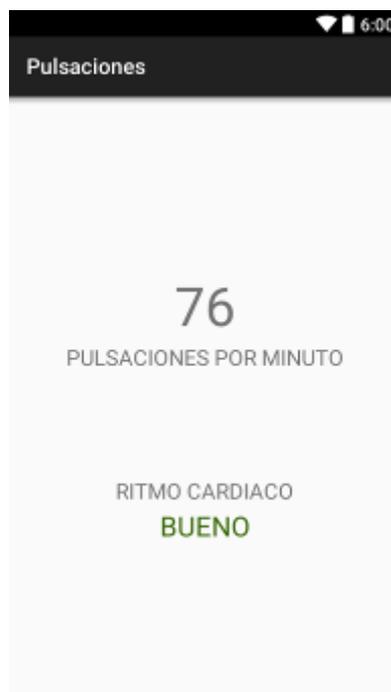


Ilustración 4-24. Aplicación conectada al PSoC mostrando las pulsaciones en tiempo real

4.4.8 Programación del archivo *AndroidManifest.xml*

Se trata del último archivo que hay que verificar antes de compilar y ejecutar la aplicación, el cual también se edita mediante etiquetas XML, al igual que el fichero `activity_main.xml`, descrito anteriormente.

Lo primero que hay que hacer es añadir los permisos que va a necesitar la aplicación para poder llevar a cabo los objetivos que se han propuesto. Los permisos son los siguientes:

```
<uses-permission android:name="android.permission.BLUETOOTH" />

<uses-permission
android:name="android.permission.BLUETOOTH_ADMIN" />

<permission android:name="elias.pulsaciones.READ_DATABASE"
android:protectionLevel="normal" />

<permission android:name="elias.pulsaciones.WRITE_DATABASE"
android:protectionLevel="normal" />
```

El permiso “`android.permission.BLUETOOTH`” se va a utilizar para que la aplicación pueda realizar comunicaciones utilizando el Bluetooth, ya sea iniciar una conexión, aceptar una conexión o transferir datos.

El permiso “`android.permission.BLUETOOTH_ADMIN`” se va a utilizar para poder realizar acciones más avanzadas con el Bluetooth, como escanear dispositivos o cambiar los ajustes del Bluetooth.

Los permisos “`elias.pulsaciones.READ_DATABASE`” y “`elias.pulsaciones.WRITE_DATABASE`” se utilizan para que el content provider pueda manipular correctamente la base de datos.

El resto del archivo está compuesto por etiquetas que expresan la estructura que se ha creado para la aplicación. Se encuentra una “`<activity>`”, la cual es la actividad principal. Dentro de esta actividad hay un “`<intent-filter>`” para poder recibir los eventos del “Broadcast Update”. Después se encuentra un “`<service>`”, el cual hace referencia al servicio que se ha creado para manejar los eventos del Bluetooth. Por último, se encuentra una referencia al “`<provider>`” que se ha creado, el cual incorpora una serie de permisos de lectura y escritura, así como una etiqueta en la que se define la “`authoritie`” del content provider, para que pueda ser accedido por otras aplicaciones.

4.5 Conclusiones

En este capítulo se ha explicado todo lo referente a la creación del proyecto. Se ha comenzado explicando el proceso para construir el sensor de medida, justificando los valores utilizados en los componentes, así como su disposición en el esquema del que forman parte para poder medir las pulsaciones.

Después se ha explicado cómo se ha programado el PSoC para que fuera capaz de formar parte del sistema de medición, así como la configuración realizada para poder calcular las pulsaciones por minuto a partir de la señal que se introducía en él. La parte del PSoC se ha completado explicando la configuración y programación del Bluetooth incorporado en el mismo, para poder transmitir las pulsaciones en tiempo real a un dispositivo móvil.

Para terminar, se ha dado un repaso a todo el ciclo de trabajo que debía seguir la aplicación de Android para poder conectarse al PSoC y recibir las pulsaciones por minuto en tiempo real, así como la forma en la que se ha programado la interfaz y la programación utilizada para poder almacenar las pulsaciones recibidas en una base de datos, la cual se ha dejado correctamente configurada para que pueda ser abierta desde una aplicación externa a ésta.

Capítulo 5

Análisis de Resultados y Conclusiones

5.1 Pruebas y análisis de resultados

Tras haber terminado la fase de desarrollo del proyecto, se debe comprobar que funciona correctamente. Dado que el sistema se compone de tres elementos fundamentales, que son el sistema de medida, el PSoC, y la aplicación para Android, se deberá comprobar que los tres componentes están funcionando correctamente.

En el caso del sensor de medida, hay que asegurarse de que el shield se encuentra correctamente acoplado a la placa de desarrollo del PSoC, y prestar especial atención al fototransistor.

Durante la realización de las pruebas de funcionamiento, el fototransistor ha supuesto el principal foco de los problemas, ya que debido al funcionamiento del mismo, si se quieren obtener resultados precisos, se debe mantener la superficie del cuerpo que esté en contacto con el sensor en una posición lo más inmóvil que sea posible respecto del sensor.

Dado que el sensor detecta las variaciones en la luz reflejada en el tejido del cuerpo, cualquier mínimo movimiento puede provocar falsos positivos que distorsionarían la señal que se quiere obtener para detectar los latidos del corazón. Por lo tanto, se recomienda colocar el sensor en una pinza y sujetar el dedo con la misma, para que durante toda la fase de medición se puedan evitar este tipo de errores en la medida de lo posible.

Otro de los componentes que han presentado cierta complicación durante la fase de pruebas han sido los amplificadores operacionales del PSoC, ya que pese a poder ser utilizados desde cualquier puerto con los que cuenta el PSoC, durante las pruebas se observaba su salida con el osciloscopio, y no terminaban de funcionar como se esperaba. Por ello, tras varias pruebas se consiguió utilizar los pines del PSoC más adecuados para que los amplificadores operacionales funcionaran correctamente.

Hay que tener en cuenta que durante las pruebas el PSoC se ha alimentado mediante la conexión USB con el ordenador. Si las pruebas se hubieran realizado con algún tipo de pila o de batería, habría que asegurarse de que la tensión suministrada es la correcta, ya que por debajo de un cierto nivel de tensión los componentes internos del PSoC como pueden ser por ejemplo la radio podrían ver alterado su funcionamiento, y no funcionar completamente con el rendimiento o la funcionalidad con la que se espera que funcionen.

En cuanto al dispositivo móvil con Android, para poder realizar las pruebas, debe ser un dispositivo móvil compatible con la versión 4.0 de Bluetooth, y durante las pruebas debe tener activado el Bluetooth.

Una vez que se tienen claras las recomendaciones sobre el uso de los elementos de este proyecto, el procedimiento para realizar las pruebas es el siguiente:

- En primer lugar, se deberá colocar el sensor sobre la zona del cuerpo en la que se pretenda medir el ritmo cardiaco. Se recomienda que el sensor se coloque en una posición en la que se mueva lo menos posible, y que la zona del cuerpo expuesta al sensor se mantenga lo más inmóvil posible respecto a éste.
- Lo siguiente es alimentar al PSoC. En estas pruebas se ha alimentado conectando la placa de desarrollo mediante el puerto USB a un ordenador. Tras unos dos segundos, el PSoC se habrá iniciado correctamente y habrá comenzado a emitir paquetes de “advertising”, para que un dispositivo lo encuentre mediante Bluetooth y se conecte a él.
- A continuación se ejecuta la aplicación de Android, la cual se conectará automáticamente con el PSoC y le pedirá que le mande las pulsaciones que el PSoC detecte, todo esto sin necesidad de intervención por parte del usuario.
- Por último, si el sensor se ha colocado correctamente, la aplicación de Android mostrará en tiempo real las pulsaciones por minuto. Si el sensor se retira del tejido sobre el que está situado, o si no se encuentra bien colocado y pasan más de 2 segundos sin recibir una pulsación, la aplicación de Android mostrará

automáticamente un valor de 0 pulsaciones por minuto, para que el usuario sepa que el sensor no está detectando ninguna pulsación.

Para la realización de las pruebas de este proyecto, el sensor se ha situado en el extremo del dedo índice de la mano derecha, y ha habido momentos en los que ha sido bastante difícil encontrar la posición adecuada en la que se pudiera detectar el ritmo cardíaco.

Sin embargo, en los momentos en los que el sensor detectaba correctamente las pulsaciones, se ha comparado el valor medido con el valor generado por un pulsómetro que obtiene las pulsaciones mediante una banda pectoral. Los resultados obtenidos han dejado en el caso más favorable una diferencia con el valor obtenido por la banda pectoral de unas 2 pulsaciones por minuto, y en el caso más desfavorable una diferencia de 6 pulsaciones por minuto.

Los resultados indican que este sistema de medida no aporta un nivel de precisión del 99%, pero dejan claro que el sistema es capaz de detectar las pulsaciones y que el dispositivo móvil es capaz de mostrarlas con una cierta precisión.

5.2 Conclusiones y trabajo futuro

Tras haber concluido este proyecto, hay que valorar el trabajo realizado con respecto a los objetivos marcados al comienzo del mismo. En primer lugar hay que valorar el proceso de selección de componentes. Primero hubo que elegir el método para medir las pulsaciones, siendo la Fotopleletismografía el método seleccionado. Este método conllevaba la utilización de una cierta electrónica que permitiera acondicionar la señal obtenida para poder ser utilizada y procesada. Para ello hubo que utilizar etapas de filtrado y de amplificación.

Después fue necesario utilizar sistema programable que permitiera procesar la señal que contenía las pulsaciones para poder extraer información de ella. Para este cometido hubo que elegir una arquitectura PSoC que realizara esta tarea, siendo el PSoC 4 BLE el sistema elegido. Esto conllevó un periodo para estudiar y comprender el funcionamiento de las herramientas que permitían programar el PSoC, así como el estudio del protocolo Bluetooth Low Energy para que el PSoC fuera capaz de transmitir el valor de las pulsaciones a un dispositivo externo.

Tras haber solucionado la parte de capturar y procesar las pulsaciones, hubo que elegir una plataforma en la que poder mostrarlas, siendo Android la plataforma de software ganadora, debido al gran ecosistema de dispositivos con los que cuenta. Esto supuso un periodo para estudiar el desarrollo de aplicaciones para Android, y la necesidad de hacer

uso de la API que permitiera utilizar las funciones de Bluetooth Low Energy del dispositivo móvil sobre el que se ejecutara la aplicación.

Por último, tras haber comprobado que todos los elementos del proyecto funcionaban correctamente, se integró la electrónica de acondicionamiento de señal en un shield que pudiera ser fácilmente integrado sobre un dispositivo compatible con el mismo, y se pulió la interfaz de la aplicación de Android para que el usuario tuviera que intervenir lo menos posible en el proceso y pudiera observar las pulsaciones por minuto rápidamente.

Por lo tanto los objetivos propuestos al comienzo del proyecto se han ido cumpliendo conforme se iban ensamblando los distintos elementos del sistema y se comprobaba que lo estudiado y desarrollado funcionaba. No obstante, hay que tener en cuenta que se dispone de un margen de mejora para mejorar la precisión de la medición del ritmo cardíaco. Los puntos en los que se podría trabajar en un futuro son:

- Utilizar componentes electrónicos que cuenten con unas mejores tolerancias, para poder ajustar mejor la ganancia y la frecuencia de corte de los filtros.
- Experimentar con otros sensores distintos al fototransistor de infrarrojos, buscando el más adecuado y el más preciso para este tipo de aplicación.
- Construir un sistema que permita fijar el fototransistor que realiza la medición a la piel, para evitar el error causado por los movimientos involuntarios.
- Utilizar amplificadores operacionales externos y de mayor precisión que los utilizados internamente por el PSoC, para evitar la latencia y los errores producidos al tener que procesar la señal a través del PSoC.
- Aumentar la frecuencia de reloj que se introduce en el “Timer” del PSoC, para que al calcular las pulsaciones por minuto se tenga una variable en el registro que cuente con mayor resolución que la utilizada hasta ahora, esto permitirá reducir los errores al redondear las pulsaciones por minuto a un valor entero.
- Buscar un nuevo algoritmo para el PSoC que permita mejorar la precisión en el momento de calcular las pulsaciones por minuto.
- Mejorar la aplicación de Android, añadiendo una pantalla para poder visualizar un registro histórico de las pulsaciones almacenadas en la base de datos que se ha creado.

Referencias

- [1] George E. Billman, “Heart Rate Variability – A Historical Perspective” (2011) <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3225923>
- [2] “Heart Rate Variability” https://en.wikipedia.org/wiki/Heart_rate_variability
- [3] “Electrocardiograma” (2014) <https://www.nlm.nih.gov/medlineplus/spanish/ency/article/003868.htm>
- [4] Asha Ganesan & Ananda Ganesh, “Product how-to: Heart rate monitor using a programable SoC” (2013) <http://www.edn.com/design/medical/4421721/Product-how-to--Heart-rate-monitor-using-a-programmable-SoC>
- [5] “CY8CKIT-042-BLE Bluetooth Low Energy (BLE) Pioneer Kit” <http://www.cypress.com/?rID=102636>
- [6] “CSR10X0 Starter Development Kit” <http://www.broadband.se/en/shop/bluetooth/bluetooth-smart/devkit-bluetooth-smart/csr10x0-starter-development-kit>
- [7] “NRF8001-DK” <https://www.rutronik24.com/product/nordic/nrf8001-dk/1323.html>
- [8] “WiFi: historia, evolución, aplicaciones, desarrollos...” (2010) <http://www.wificlub.org/featured/wifi-historia-evolucion-aplicaciones-desarrollos/>
- [9] “A brief history of Wi-Fi” (2004) <http://www.economist.com/node/2724397>
- [10] “Wi-Fi” <https://en.wikipedia.org/wiki/Wi-Fi>
- [11] “A short history of Bluetooth” (2014) <https://www.nordicsemi.com/eng/News/ULP-Wireless-Update/A-short-history-of-Bluetooth>
- [12] “Bluetooth” <https://en.wikipedia.org/wiki/Bluetooth>
- [13] Taylor Martin, “The evolution of the smartphone” (2014) <http://pocketnow.com/2014/07/28/the-evolution-of-the-smartphone>

- [14] “Android (operating system)”
[https://en.wikipedia.org/wiki/Android_\(operating_system\)](https://en.wikipedia.org/wiki/Android_(operating_system))
- [15] “iOS” <https://en.wikipedia.org/wiki/IOS>
- [16] “Windows Phone” https://en.wikipedia.org/wiki/Windows_Phone
- [17] “PSoC Creator Integrated Design Environment (IDE)”
<http://www.cypress.com/products/psoc-creator>
- [18] “Java SE, Downloads”
<http://www.oracle.com/technetwork/es/java/javase/downloads/index.html>
- [19] “Download Android Studio and SDK Tools”
<https://developer.android.com/sdk/index.html>
- [20] “Adding SDK Packages” <https://developer.android.com/sdk/installing/adding-packages.html>
- [21] “Creating a Project with Android Studio”
<https://developer.android.com/training/basics/firstapp/creating-project.html#Studio>
- [22] Salvador Gómez Oliver “Estructura de un proyecto Android (Android Studio)” (2014)
<http://www.sgoliver.net/blog/estructura-de-un-proyecto-android-android-studio/>
- [23] Salvador Gómez Oliver “Componentes de una aplicación Android” (2010)
<http://www.sgoliver.net/blog/componentes-de-una-aplicacion-android/>
- [24] “Android Developers: Bluetooth LE Package Summary”
<https://developer.android.com/reference/android/bluetooth/le/package-summary.html>
- [25] “RPR-220”
http://rohms.rohm.com/en/products/databook/datasheet/opto/optical_sensor/photosensor/rpr-220.pdf