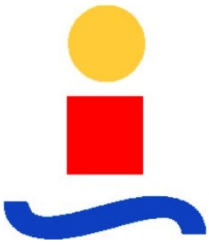


ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

GRADO EN INGENIERÍA DE LAS TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO DE UN DISPOSITIVO PARA LA DETECCIÓN DEL ESTRÉS A PARTIR DE LA SEÑAL DE FOTOPLETISMOGRAFÍA

AUTOR

David Martín Sánchez

TUTORA

María de Mar Elena Pérez

Índice

Índice de Figuras

Índice de Tablas

Glosario

AFE	Analog front end
CV	Coeficiente de varianza
ECG	Electrocardiograma, electrocardiografía
EMD	Descomposición empírica modal
HR	Ritmo cardíaco
HRV	Variabilidad de la frecuencia cardíaca
IMF	Función intrínseca modal
mRR	Media de los RR
pNN50	Porcentaje de RR consecutivos que discrepan en más de 50ms
PPG	Fotopletismograma, fotopletismografía
RMSSD	Cuadrado de la raíz media de la unión de los intervalos RR adyacentes
RR	Tiempo entre dos picos R consecutivos
SD1	Desviación estándar a través de la perpendicular de la línea de identidad de la gráfica de Poincaré
SD2	Desviación estándar a través de la línea de identidad de la gráfica de Poincaré
SDNN	Desviación estándar de los intervalos RR

Capítulo 1. Introducción

1.1 Antecedentes

Este documento forma parte del programa diseñado por la tutora de este trabajo para la detección del estrés, entendiéndose este último como una patología orgánica y no meramente psiquiátrica. En concreto, este trabajo toma el relevo de [1], donde se diseñó un algoritmo eficaz de detección del estrés.

El desencadenante de este programa fue la ascendente aparición durante los últimos años de estudios que relacionaban el estrés emocional o laboral con determinado impacto fisiológico, es decir, cierta característica perjudicial para la salud medible objetivamente. En estos estudios se demostró, por ejemplo, que los pacientes a los que se les había diagnosticado estrés eran más propicios a desarrollar enfermedades cardiovasculares. Este y otros ejemplos sirvieron para demostrar cómo el estado de ánimo influye en la salud cardiovascular, concluyendo que existe una estrecha relación entre uno y otro.

A partir de entonces, se comenzó a aplicar este nuevo enfoque a los estudios sobre enfermedades relacionadas con la regulación cardíaca. Los resultados arrojados confirmaban los estudios anteriores, puesto que se encontraba una mayor movilidad y mortalidad de enfermedades cardiovasculares en los pacientes que sufrían de depresión o ansiedad. Este hecho llevó a calificar el estrés de factor de riesgo para la salud, dejando atrás la tradicional interpretación de estado emocional.

En estos momentos, las líneas de investigación se centran en desarrollar métodos para diagnosticar el estrés. Uno de estos se basa en el análisis de la variabilidad de la frecuencia cardíaca (en adelante HRV, del inglés heart rate variability), que es el que se ha seguido en [1]. La HRV no es más que la capacidad que tiene el sistema cardiovascular para adaptar el ritmo cardíaco a las necesidades en cada momento determinadas por una actividad normal del individuo. El estrés, en este caso, se identifica como una variabilidad anómala de este ritmo respecto a un individuo sano, provocando trastornos en el funcionamiento de este sistema. Esto lleva a las enfermedades comentadas al comienzo, y de ahí la importancia de su diagnóstico.

Para obtener el valor de la HRV se suele usar un instrumento de electrocardiografía (ECG, del inglés electrocardiography o electrocardiogram) que mide la señal eléctrica de la actividad del corazón, de uso ampliamente extendido. Los métodos de extracción de este parámetro son varios y variados y depende de las características de la ocasión utilizar uno y otro. En [1] se ha trabajado con uno de estos métodos y se ha diseñado un algoritmo de detección de estrés para determinar el estado del individuo respecto a los valores que se extraigan de la señal de ECG.

Es decir, el trabajo precedente a este ha supuesto un punto de unión entre los métodos desarrollados para la extracción de la HRV y de los estudios de categorización del estrés. Esta técnica se explicará en parte en el capítulo siguiente para facilitar la comprensión de lo que se describe en este documento.

1.2 Objetivos y alcance

La propuesta y motivo de este trabajo fue el diseño de un dispositivo que implementase el algoritmo descrito en [1] para facilitar el diagnóstico de estrés. Además, se propuso modificar este algoritmo para llevar a cabo este diagnóstico sobre la señal de fotopletismografía (PPG, del inglés photoplethysmogram o photoplethysmography).

Al inicio de este trabajo se disponía del algoritmo de [1], codificado en un lenguaje de alto nivel, que toma una señal de ECG, extrae la HRV y calcula determinados parámetros, que se comentarán posteriormente, para determinar con sus valores el estado de estrés. Sin embargo, este proceso requería utilizar un ordenador ejecutando el algoritmo diseñado además de un bloque de adquisición de la señal previo. Todo esto se quería simplificar a un único dispositivo que trabajase autónomamente.

Un objetivo, por tanto, marcado para este trabajo es el de diseñar un prototipo, tanto a nivel hardware como a nivel software, de un dispositivo que diagnostique el estrés de manera cómoda y simple. Los requisitos de este dispositivo se enumeran en un capítulo posterior pero basta decir que debe estar orientado tanto para ser usado por un médico en consulta como por el propio paciente en otro lugar.

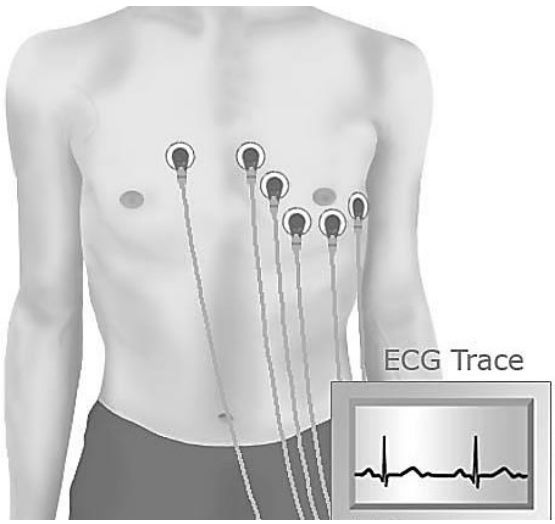


Figura 1: Sensores para la medición de ECG

Sin embargo, el principal inconveniente del método desarrollado es la adquisición de la señal. El dispositivo podría englobar todas las funcionalidades requeridas pero aun así no podría prescindir de los electrodos situados en determinados puntos del cuerpo del individuo para medir la señal de ECG. A parte de la incomodidad de operación con los electrodos, este método conlleva que el usuario debe disponer de ciertos conocimientos que no se buscan exigir con el diseño de este dispositivo. Es por esto por lo que se marcó un segundo objetivo a este trabajo: extraer la HRV con un método alternativo, es decir, mediante la PPG.

Figura 2: Sensor para la medición de PPG

La PPG es una técnica de medición óptica de los cambios volumétricos de la sangre en áreas periféricas como pueden ser las yemas de los dedos o los lóbulos de las orejas. Estos cambios volumétricos son los que se producen debidos a los latidos del corazón, irrigando la sangre a todas las zonas de nuestro cuerpo. Es una técnica no invasiva, de uso simple y muy útil que emplea un sistema de pulso oximetría, es decir, consistente en monitorizar la absorción de luz en estas zonas translúcidas. El sistema está compuesto típicamente por dos diodos LED, uno de longitud de frecuencia correspondiente al color rojo y otro correspondiente al infrarrojo, que emiten luz pulsátil intermitente e intercaladamente. En el otro extremo del sensor hay un fotodetector que detecta aquella luz que no ha sido absorbida. De este modo, se puede componer una señal con componente DC y AC de la que puede extraerse información como el pulso cardíaco, la saturación de oxígeno en la hemoglobina y otros parámetros como la respiración. En el siguiente capítulo se expondrán de manera más detallada estas características.

La proposición de emplear este método no fue casual y se debió a la manifestación de varios estudios de investigación de que algunos parámetros de las señales de ECG y PPG están correlados. Esto podría permitir en principio sustituir un método por otro, por lo que quedaba pendiente la corroboración de esta suposición.

1.3 Organización del documento

Este documento se ha redactado para exponer cómo ha sido el proceso de trabajo llevado a cabo para diseñar dicho dispositivo así como el estudio del nuevo método de detección. Se pretende que su exposición sea más explicativa que descriptiva, por lo que se dejan para anexos y bibliografía todas las características técnicas derivadas del diseño hardware y software del dispositivo.

En la primera parte, el Capítulo 2, se describen las señales de ECG y PPG, lo que representan y cuáles son sus parámetros clave. Además, se realiza una comparación entre ellas y se identifica la relación que hay entre ambas. La mayor parte del estudio sobre la extracción de la HRV a través de la señal de PPG ha sido realizada por el autor de este trabajo con anterioridad al mismo para otro proyecto, por lo que su aparición en este documento es meramente informativa. La parte original es la adaptación e integración en el algoritmo de detección de estrés.

La segunda parte que puede encontrarse en este documento, el Capítulo 3, recoge todo el proceso de diseño que se ha llevado a cabo tanto a nivel hardware como a nivel software para llegar al prototipo del dispositivo propuesto. En este capítulo se explican los pasos seguidos para llegar a la solución, dejando para el apartado de anexos la inclusión de los planos esquemáticos y del código programado.

La tercera parte de este documento, Capítulo 4, es la parte de resultados, que consiste en una discusión sobre el prototipo diseñado y en una propuesta previa a la fabricación de este prototipo para probar su correcto funcionamiento. Además, en esta parte se incluyen los resultados obtenidos al aplicar el algoritmo de detección de estrés de [1] a señales de PPG obtenidas de una base de datos.

Por último, en el Capítulo 5 se exponen las conclusiones alcanzadas a la finalización de este trabajo, así como la enumeración de futuras líneas de investigación y de desarrollo que se proponen para mejorar la técnica u optimizar el dispositivo.

Capítulo 2. Detección del estrés a través de la señal de PPG

2.1 Señal de ECG

La señal de ECG representa la actividad eléctrica recogida en el miocardio, que es el músculo encargado de la contracción cardíaca. Su forma es ya muy conocida y su uso es fundamental para la detección de distintas patologías derivadas del funcionamiento del sistema cardiovascular. En la bibliografía se referencian dos fuentes, [2] y [3], que se han utilizado para la comprensión del funcionamiento y las distintas características de una señal de ECG.

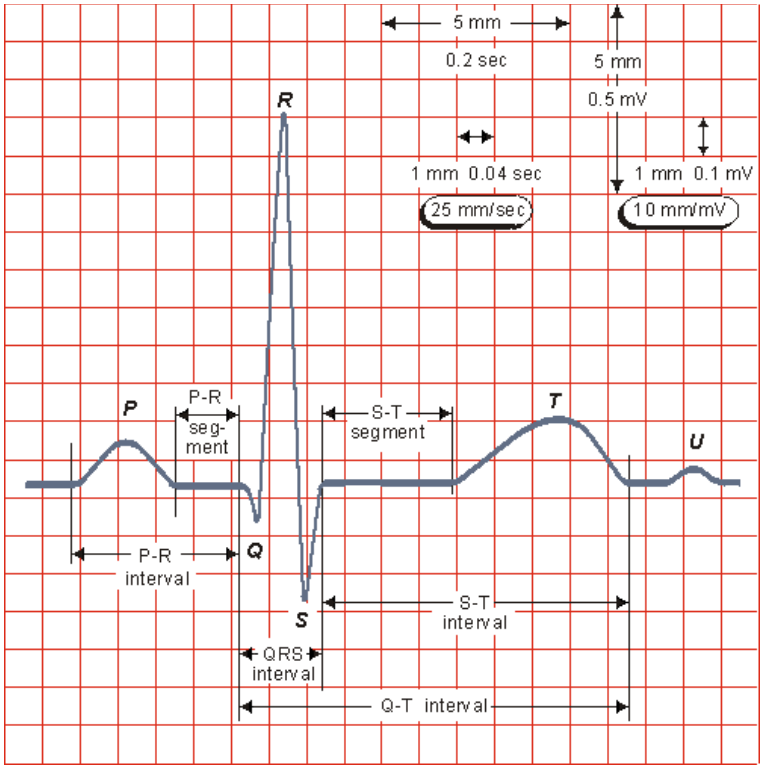


Figura 3: Ciclo completo del ECG normal

Como puede observarse en la Figura 1, la onda típica de ECG ha sido minuciosamente analizada y todas sus componentes estudiadas y descritas. Entre otros sucesos, en la señal de ECG pueden distinguirse las distintas contracciones de las fibras ventriculares. Estas contracciones son periódicas, con una frecuencia del orden del hercio, tomando como referencia el periodo de tiempo transcurrido entre un pico R y el inmediatamente posterior, conocido este valor como intervalo RR. Cuando el tiempo que transcurre entre una onda y otra, o el mismo intervalo RR, varía anómalamente quiere decir que en ese sistema cardiovascular existe determinada

enfermedad. En concreto, la detección del estrés toma como parámetro de medida para realizar el cálculo el intervalo RR.

Lo que se hizo en [1] fue emplear un detector de picos para extraer la información útil de la señal de ECG y a partir de ese vector de tiempo calcular el resto de parámetros. En este trabajo se lleva a cabo un procesamiento similar, sustituyendo la detección de picos R de la señal de ECG por la detección de picos sistólicos de la señal de PPG, como se verá más adelante. A partir de este nuevo vector de tiempo se calcula la misma serie de parámetros que para el caso anterior. A continuación se enumeran cuáles son estos y se describen brevemente (una información más detallada puede encontrarse en [1]).

2.1.1 mRR

Es la media aritmética del vector de tiempos RR:

$$mRR = \frac{1}{N}\sum_{i=1}^{N} RR_i$$

Con N la dimensión del vector.

2.1.2 SDNN

Es la desviación estándar del vector de tiempos RR:

$$SDNN = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(RR_i - mRR)}$$

2.1.3 RMSSD

Es la raíz cuadrada del valor medio de las diferencias al cuadrado de todos los intervalos RR sucesivos:

$$RMSSD = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N-1}(RR_{i+1} - RR_i)^2}$$

2.1.4 pNN50

Es el porcentaje de intervalos RR consecutivos que difieren entre sí en más de 50ms:

$$pNN50 = 100\frac{\sum_{i=1}^{N}\{|RR_{i+1} - RR_i| > 50ms\}}{N}$$

2.1.5 CV

Es una medida normalizada de la dispersión de la probabilidad de distribución:

$$CV = \frac{SDNN}{mRR}$$

2.1.6 SD1

Es la desviación estándar de la proyección de la gráfica de Poincaré a lo largo de la perpendicular de la línea de identidad. El punto de Poincaré es un método no lineal que consiste en la representación en una gráfica de todos los valores del vector de tiempos RR, donde un eje representa el componente RR_n y el otro eje el RR_{n+1}:

$$SD1 = \sqrt{Varianza\left(\frac{\overrightarrow{RR_\iota} - \overrightarrow{RR_{\iota+1}}}{\sqrt{2}}\right)}$$

2.1.7 SD2

Es la desviación estándar de la proyección de la gráfica de Poincaré a lo largo de la línea de identidad:

$$SD2 = \sqrt{Varianza\left(\frac{\overrightarrow{RR_\iota} + \overrightarrow{RR_{\iota+1}}}{\sqrt{2}}\right)}$$

Una vez calculados todos estos parámetros, el diagnóstico del estado de estrés se lleva a cabo mediante un umbral de decisión:

$$10{,}64 + 203{,}99 * SD1 - 108{,}74 * SD2 - 8{,}26 * entropía$$

Si el valor resultante es positivo, el individuo se detecta como estresado y si, al contrario, es negativo como no estresado.

2.2 Señal de PPG

La señal de PPG recoge las variaciones que se producen en el flujo o en el volumen sanguíneo que ocurre con cada contracción cardíaca. En los últimos años su uso se ha comenzado a extender debido a su fácil uso y su técnica no invasiva, lo que hace que el sensor de pulso oximetría se haya convertido en sustituto en muchas ocasiones de otro instrumental biomédico como puede ser el ECG.

Este desarrollo ha arrastrado consigo una amplia investigación de las propiedades de la señal extraída. A pesar de su aparente simplicidad, no sólo de su técnica sino también de la forma recogida en su onda, esta señal ha resultado ser muy compleja, pudiendo detectarse en ella multitud de actividades y respuestas como la frecuencia cardíaca, la respiración, saturación de

oxígeno, la viscosidad de la sangre, la presión arterial o incluso cambios de la postura del usuario. Como punto en contra se ha detectado que esta señal es muy sensible a los denominados artefactos, que son errores que se observan en la medida como resultado de la propia técnica de medida, del instrumental utilizado o del procesado necesario para extraer las características.

El ciclo completo de la señal de PPG de un paciente sano puede verse en la Figura 4. Una descripción detallada de sus características puede encontrarse en [12].

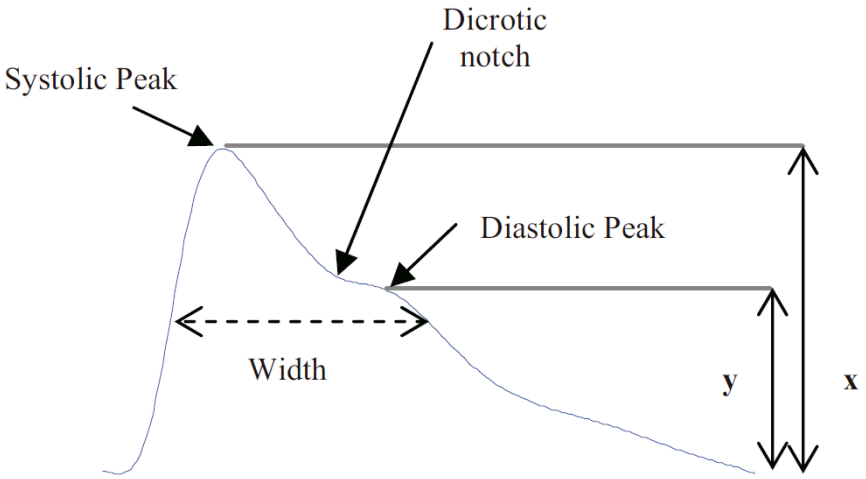


Figura 4: Ciclo completo de un PPG normal

Esta señal tiene una componente DC y una componente AC. Es en esta última donde se detectan las características de la señal, siendo el valor DC despreciado en la mayoría de las situaciones. A continuación se enumeran los principales parámetros de esta señal:

2.2.1 Amplitud sistólica

Es el nivel máximo de volumen que se detecta en el área sometida a medición correspondiente a la fase sistólica del corazón, donde se eyecta la sangre.

2.2.2 Amplitud diastólica

Tras el pico sistólico, el volumen comienza a decrecer pero se detecta un punto de inflexión tras el que se encuentra otro pico denominado diastólico, que está relacionado con la fase diastólica del sistema cardiovascular en la que los ventrículos se llenan de sangre. Este pico no siempre es visible al representar gráficamente la señal de PPG, quedando oculto por la bajada desde el pico sistólico. Este hecho no tiene relevancia en este trabajo, ya que no influye en la detección del estrés. Sin embargo, cuando sí es visible el pico diastólico, un detector de picos mal configurado podría confundir erróneamente el pico diastólico con el sistólico, resultando en una HR cercana al

doble y una HRV que no guarda relación a la que se obtiene sólo tomando los picos sistólicos. Esto se suele resolver incluyendo la condición de que para considerar un pico como sistólico su amplitud no debe ser menor que un tanto por ciento del mayor pico detectado.

2.2.3 Ancho del pulso

Es el tiempo que transcurre desde que el nivel traspasa el 50% del nivel del pico sistólico hasta que vuelve a traspasarlo cuando decrece la señal. Este valor está correlado con la resistencia del sistema vascular.

2.2.4 Intervalo pico-a-pico

Es el tiempo que transcurre entre dos picos sistólicos sucesivos. Este valor está correlado con el intervalo de tiempo RR extraído del ECG.

2.2.5 Intervalo de pulso

Es el tiempo que transcurre entre el nivel mínimo de volumen detectado en el ciclo de PPG y el siguiente. Este valor suele ser idéntico al del intervalo pico-a-pico y se prefiere su uso cuando los picos sistólicos son más difíciles de reconocer (por ejemplo, cuando la cima de la señal tiene una pendiente suave).

2.3 Relación entre ECG y PPG

Una vez estudiadas las características de ambas señales, se procede a relacionarlas para poder aplicar el método de detección del estrés sobre la señal de PPG. Como se comentó al inicio de este documento, se parte de la suposición de una correlación entre el intervalo RR y el intervalo peak-to-peak cuando ambas señales son registradas simultáneamente.

En la Figura 5 se comparan las señales extraídas mediante ECG y PPG en un mismo dominio de tiempo.

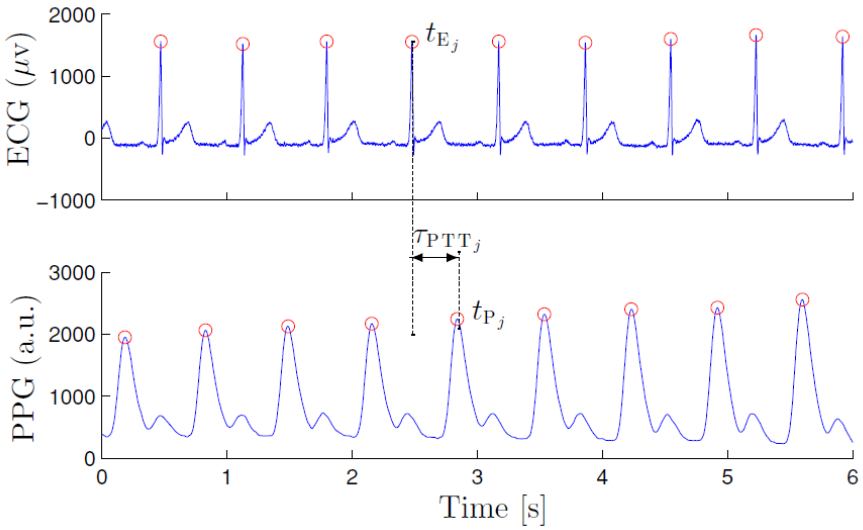


Figura 5: Comparación entre las señales de ECG y PPG

Como puede observarse en la figura anterior, entre el pico R de la señal de ECG y el pico sistólico de la señal de PPG existe un cierto retraso del orden de los centenares de milisegundos que se mantiene constante durante toda la medición y que no es más que el tiempo que transcurre desde que ocurre la sístole ventricular hasta que este evento se advierte en el área donde se encuentra el sensor de pulso oximetría. Pero más allá de este retraso no hay mayor discrepancia entre el periodo que transcurre entre dos picos R y dos picos sistólicos, pudiendo comprobarse como se hizo en [8] que la correlación entre ambos intervalos es cercana a 1. Esto quiere decir que *a priori* la extracción del vector de tiempos RR puede hacerse directamente midiendo el intervalo pico-a-pico de la señal de PPG.

Para completar el estudio de esta relación, se lleva a cabo una batería de pruebas para comprobar el correcto funcionamiento del algoritmo de detección de picos y para corroborar esta afirmación. Se adapta el código de detección de picos de [1] a la señal de PPG, teniendo en cuenta su composición e incluyendo la exclusión de picos diastólicos mencionada anteriormente. Este código puede encontrarse en el apartado de anexos, en Anexo III, Código del dispositivo, bajo el nombre de deteccionPicos.m. El cálculo de la HRV y la HR se lleva a cabo mediante las ecuaciones que las describen.

$$HRV_i = RR_i - RR_{i+1}$$

$$HR = \frac{60}{\frac{1}{N}\sum_{i=1}^{N} HRV_i}$$

Para la batería de pruebas se hizo uso de la base de datos MIMIC II, [39] y [40], una completa base de datos de señales fisiológicas de pacientes en la UCI de un hospital grabadas durante, en ocasiones, horas, que incluye registros de señales de ECG y PPG de un mismo paciente con el eje temporal compartido, lo que hace óptima su comparación. Como regla general, se ha tomado la derivada II de la señal de ECG para una comparación menos dispersa, aunque el algoritmo puede aplicarse a cualquier otra derivada. La selección de los registros sobre los que se han realizado las pruebas se ha hecho en base a tres parámetros.

1) Señales registradas. La primera característica que se tuvo en cuenta fue la selección de grabaciones en las que se midiese tanto la señal de ECG como la señal de PPG, así como su correcta alineación para poder comparar las características sucedidas simultáneamente en ambas señales, ya que no todos los registros de esta base de datos poseen ambas señales.
2) Número de muestras. Otro parámetro que se tuvo en cuenta fue el número de muestras del registro, ya que debía corresponderse con al menos un intervalo temporal de cinco minutos (ventana de tiempo estimada en [1] para una fiable detección del estrés). Las señales de esta base de datos están muestreadas a 125 Hz, por lo que el número de muestras no puede ser menor que 37500.
3) Artefactos. Los efectos producidos por la aparición de este efecto durante la medición de la señal se analizarán más adelante pero se adelanta que los registros que se seleccionaron se hicieron también en base a una mínima o nula aparición de estos eventos.

De este modo, se seleccionaron veinte registros de esta base de datos con los que se ha trabajado durante todo este trabajo, para comprobar la correlación entre ambas señales y el correcto funcionamiento de la detección de picos en esta primera parte y para entrenar el algoritmo de detección del estrés en el Capítulo 4.

El procedimiento de esta prueba consiste en tomar una ventana de cinco minutos y extraer los vectores de tiempo de los intervalos RR, para la señal de ECG, y de los intervalos pico-a-pico, para la señal de PPG. Se ha medido la HR para poder comparar fácilmente los resultados. En la Tabla 1 se muestra la frecuencia cardíaca medida independientemente en la señal de ECG y en la señal de PPG así como la diferencia que existe entre ellas. En la primera columna, el número de once dígitos separados por un guion identifica un registro de la base de datos. Las coordenadas indicadas entre paréntesis se refieren a en qué índice del vector se encuentran las señales de PPG y ECG (como se ha comentado, en cada registro se miden varias señales y cuando se lee uno de ellos hay que seleccionar previamente la señal en cuestión). Por último, los números que aparecen en la segunda fila indican la ventana temporal escogida en minutos.

Tabla 1: Comparación de los latidos por minutos medidos mediante ECG y PPG

Grabación	HR señal ECG	HR señal PPG	Diferencia
3000086_0003(2,4) 12:17	105	105	0
3000086_0006(2,4) 34:39	100	100	0
3000531_0001(3,4) 7:12	98	98	0
3000714_0004(6,3) 37:42	69	69	0
3000781_0002(2,4) 10:15	64	139	75
3000858_0008(3,4) 10:15	168	168	0
3000860_0004(3,4) 10:15	70	72	2
3000860_0007(3,4) 0:5	69	76	7
3001158_0005(3,4) 0:5	107	109	2
3001203_0007(3,4) 4:9	88	91	3
3001203_0010(3,4) 0:5	130	89	41
3001912_0005(6,3) 20:25	86	86	0
3001912_0007(6,3) 15:20	95	95	0
3001912_0012(6,3) 8:13	99	148	49
3002113_0004(6,3) 30:35	77	76	1
3002113_0007(6,3) 10:15	76	78	2

3002151_0005(6,3) 15:20	76	78	2
3002762_0004(2,5) 10:15	85	92	7
3003405_0003(6,3) 10:15	67	97	30
3100524_0004(6,3) 9:14	81	85	5

Como puede observarse, en la mayoría de los casos, la diferencia entre el HR medido de la señal de ECG y el medido de la señal de PPG es pequeña y admisible (el criterio de admisibilidad suele ser de un 5% del valor medido). Los casos en los que esta diferencia era considerable (en concreto, el registro 3000781_0002) se analizaron por separado y se visualizaron sus formas de onda. En ellos se observa que la señal grabada se había corrompido por la señal de respiración.

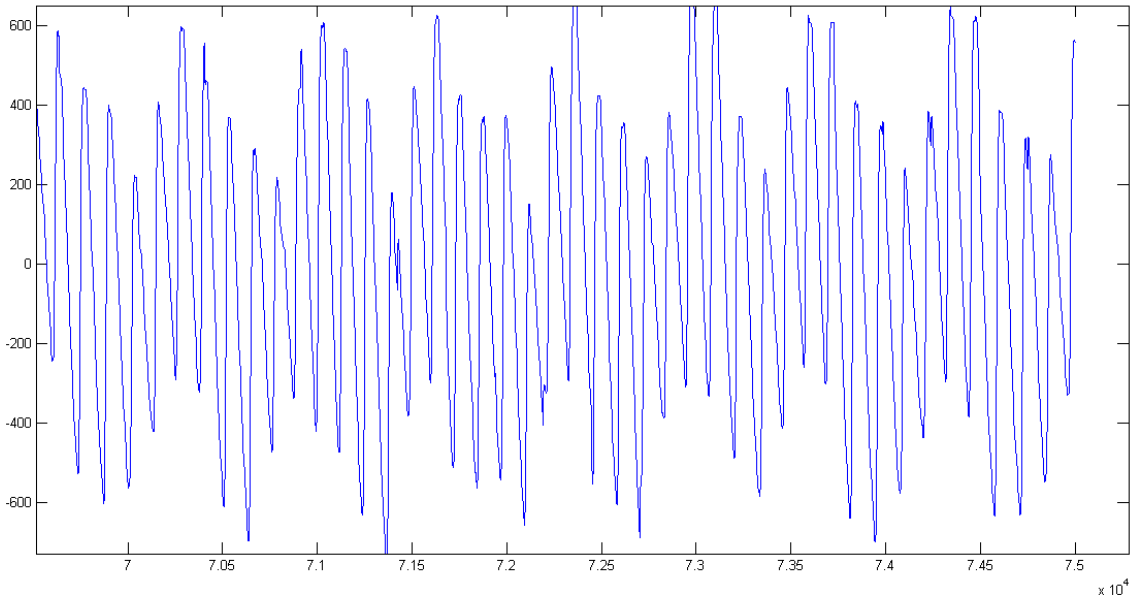


Figura 6: Efecto de la respiración en la señal de PPG

En la figura anterior se puede ver cómo la señal de respiración pulmonar, de frecuencia menor que la frecuencia cardíaca, modula la señal de PPG. Este no es un efecto común ya que la frecuencia pulmonar se suele eliminar para visualizar correctamente la señal de PPG, por lo que la discrepancia se atribuye a un mal tratamiento de la señal (en la etapa de ganancia o de filtrado).

Se concluye, por tanto, que la detección de picos mediante la señal de PPG es válida usando el algoritmo que se recoge en el tercer anexo. A continuación se discutirán las limitaciones que pueda tener esta técnica y en el Capítulo 4 se recogen los resultados que se obtienen para la detección del estrés.

2.4 Discusión sobre el método

Como se comentó previamente en este capítulo, en la descripción de la señal de PPG, esta señal se ve perjudicada por artefactos con frecuencia si no se guarda especial cuidado en su manipulación y medida, por lo que se dificulta su uso. Es por esto que una parte del estudio realizado sobre el método alternativo de detección del estrés se centra en saber cuándo este método es aplicable.

Durante los últimos años se han llevado a cabo varias investigaciones para conocer cómo afecta el estado del paciente a la medición de pulso oximetría. Estos estudios tenían como propósito analizar las alteraciones que se registraban en la HRV cuando un individuo pasaba del reposo a otro estado. En este otro estado, el sensor de pulso oximetría se mueve y la regulación cardíaca se altera desmesuradamente, lo que hace que surjan artefactos y se dificulte la detección de picos.

Estos estudios han demostrad que cuando el paciente se encuentra en posición supina, totalmente inmóvil y despierto, la correlación entre la variabilidad de la frecuencia cardíaca medida mediante el ECG y la extraída del PPG es prácticamente 1, como puede verse en [8]. Y esta correlación suele seguir siendo elevada cuando el paciente se encuentra sentado o cuando se le somete a la prueba conocida como mesa inclinada, en la que, estando inmóvil, se varía la posición de la cama donde se encuentra tumbado (lo que produce una variación en la medición). En el primer caso, [10], se detectó que las componentes de alta frecuencia perdían correlación, aunque esta permanecía en un rango calificable de aceptable. Es por tanto que se concluye que la medición puede llevarse a cabo tanto en posición supina (caso óptimo) como sentado, admitiendo un cierto movimiento.

Sin embargo, la correlación de la HRV decrece hasta hacerse inadmisible cuando el paciente se encuentra caminando, corriendo o haciendo ejercicio. Esto hace que el método para medir la HRV mediante PPG no pueda reemplazar al electrocardiógrafo cuando el paciente no esté tumbado o sentado. No obstante, el ritmo cardíaco medido a través del PPG y del ECG sí permanece altamente correlado cuando el paciente está estático o se está moviendo a excepción de cuando este se encontraba andando, mejorándose la correlación cuando se inmoviliza el sensor.

También se llevó a cabo un análisis sobre cómo afecta a la medición el estado previo del paciente, es decir, cómo difería el resultado si antes de someterse a la prueba este había estado haciendo ejercicio. El informe [9] llevó a cabo un experimento en el que se analizó las formas de onda de la PPG inmediatamente después de realizar ejercicio y veinte minutos más tarde de este. En concreto, se centraron en la anchura del pulso al 90, 70, 50, 30 y 10% de la amplitud máxima del pico sistólico y vieron que estas variaban en ambos casos estudiados. Sus resultados puede que no sean de aplicación directa a este trabajo pero sí sirvieron para determinar que pasados los veinte minutos la forma de onda se había recuperado casi por completo del esfuerzo al que había sido sometido el paciente, por lo que se considera que transcurrido ese tiempo de una actividad física se puede llevar a cabo la medición del estrés.

2.5 Detección del estrés a través de la señal de PPG

En la Figura 7 puede verse un esquema que resume las etapas que componen la adaptación de la señal extraída del sensor de pulso oximetría para poder aplicar la señal al algoritmo de detección del estrés.

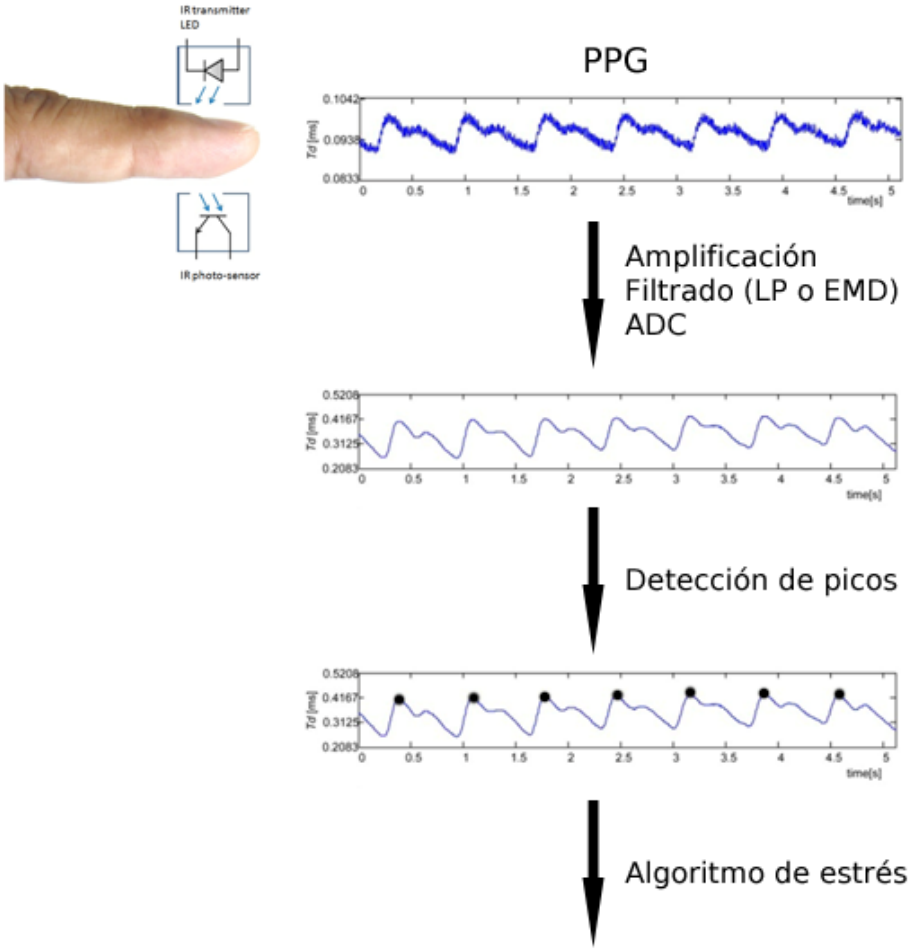


Figura 7: Procesamiento de la señal de PPG

Hasta ahora se ha descrito ese proceso al completo a excepción de la segunda etapa de acondicionamiento de la señal, que será el tema principal de este punto.

Como en la mayoría de la adquisición de datos, es necesaria una etapa de amplificación, otra de filtrado y una que convierta el valor medido a digital. En esta conversión radica la precisión del instrumento y de la amplificación depende la inmunidad ante efectos como el mencionado anteriormente, donde la frecuencia pulmonar modulaba la señal y dificultaba la detección de picos. Con el filtrado se elimina todo el ruido de interferencia que afecta a la señal y se puede llevar a cabo con un filtro paso bajo tradicional. Pero algunos estudios han demostrado que la aplicación de una nueva técnica mejora esta adquisición.

Gran parte de los artefactos que afectan a la señal se deben al instrumento de medida. Esto ocurre por el simple movimiento del sensor, ya que, al tratarse de una pinza no representa un anclaje perfecto y el dedo puede encontrar cierta holgura en su interior y hacer que el trayecto que recorre la luz emitida por los diodos LED no sea el mismo en cada instante. Experimentos como el llevado a cabo en [9] apuntan a que este es uno de los motivos por los que ocurren los artefactos aunque una mejor fijación del sensor no siempre implica una notable mejora de la señal medida o no siempre es posible.

Es por ello por lo que hay que recurrir a otros métodos para que la medida esté aislada de todos los efectos negativos que puedan afectarla. Uno de los que se ha demostrado más efectivo es la descomposición empírica modal (en adelante EMD, del inglés empirical mode decomposition), [5], [6] y [7]. Esta técnica matemática consiste en descomponer la señal medida en componentes denominadas funciones intrínsecas modales (en adelante IMF, del inglés intrinsic mode function):

$$x(t) = \sum_{i=1}^{N} IMF_i + r_N$$

La anterior definición quiere decir que la señal de PPG se puede descomponer en un número N componentes más simples y un residuo. Las IMF deben cumplir dos condiciones:

1) El número de máximos y de cruces por cero debe ser igual o diferir en sólo una unidad.
2) La media entre la envolvente de los máximos y la envolvente de los mínimos es 0.

Mediante este método se consigue extraer la información fundamental de la señal, eliminando la componente DC y el ruido, minimizando el efecto de los artefactos ya que en muchas ocasiones es capaz de reconstruir la señal.

En número de componentes N es arbitrario en cada situación. En concreto, en los estudios de PPG, el menor número de componentes que han sido necesarias para extraer la máxima

información de la señal, sin afectar a la correlación, es de 6, valor que se tomará también en este trabajo.

Por lo tanto, a la señal medida por el sensor de pulso oximetría se le aplicará esta descomposición modal antes de someterla a la detección de picos. No obstante, en las pruebas descritas en este documento se ha prescindido de este paso ya que las señales ofrecidas por la base de datos MIMIC II han sido previamente filtradas y la EMD no tiene ningún efecto (las diferencias son mínimas), por lo que no se aprecia mejora en la medición de HR.

Capítulo 3. Diseño de la solución

3.1 Diseño funcional

Una vez probada la correlación entre la HRV medida mediante el ECG y la medida con la señal de PPG, se procede a diseñar el dispositivo que integrará esta medición para llevar a cabo la detección del estado de estrés.

En primer lugar, se recopilan los requisitos que se impusieron al proponer este trabajo. Estos son que el dispositivo debe ser portable y autónomo, de tal modo que pueda ser usado tanto por un médico en consulta como por un usuario en su casa. Es por esto que los dos principales requisitos a tener en cuenta serán:

1- Tamaño reducido, que permita que el dispositivo sea lo suficientemente pequeño como para hacerlo portable.
2- Consumo de energía mínimo, para evitar una descarga muy rápida de las baterías que lo alimentan para hacerlo autónomo.

Ambos puntos anteriores implican a su vez otros requisitos que el dispositivo deberá cumplir:

1- Deberá soportar un margen de temperatura acorde al rango comercial (de 0 a 70ºC).
2- Deberá ser robusto mecánicamente. Como no deberá sufrir altas temperaturas ni vibraciones, la robustez que se perseguirá será la de la electrónica de consumo habitual.
3- Deberá tener en cuenta la inmunidad electromagnética y electroestática que permita un funcionamiento normal.

Además, el diseño del dispositivo deberá tener en cuenta el coste que este tendrá, para poder rivalizar con los dispositivos que actualmente pueda haber en el mercado. Hoy en día, el método de detección del estrés que se emplea parte de la señal de electrocardiograma, por lo que el coste de referencia será el de un electrocardiógrafo.

De este modo, una vez que están definidos los requisitos del dispositivo, se comienza su diseño hardware, tomando como punto de partida el esquema típico de un sistema embebido:

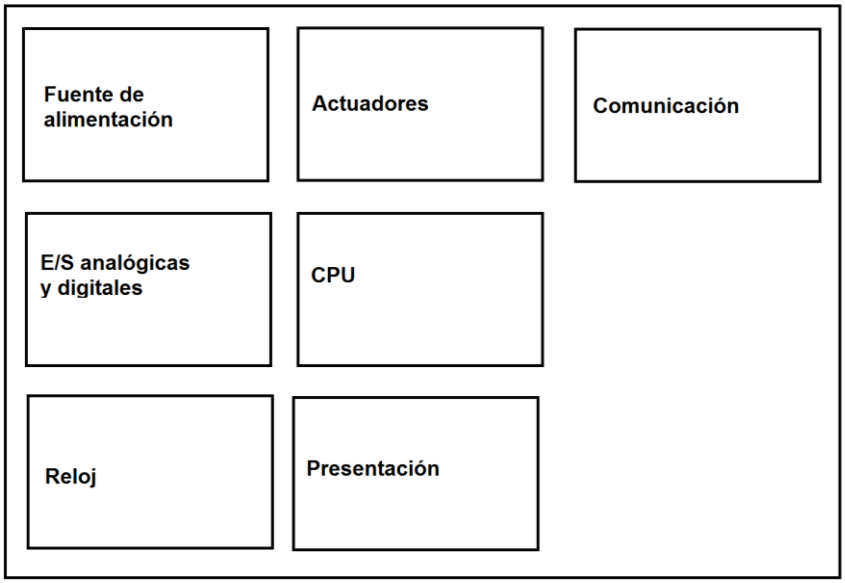


Figura 8: Esquema funcional de un dispositivo embebido

A continuación se detalla una discusión sobre los puntos que se han tenido en cuenta para la definición de cada bloque.

3.1.1 E/S analógicas y digitales

En este bloque se tendrá el sensor SpO2, que consta de dos diodos LED (rojo e infrarrojo) y de un fotodetector, que mide la saturación de oxígeno en sangre.

3.1.2 Presentación

Para aportar mayor autonomía al dispositivo, se plantea la incorporación de una pantalla para representar las medidas de frecuencia cardíaca, nivel de oxigenación de la sangre y el nivel de estrés.

3.1.3 Actuadores

El dispositivo dispondrá de un par de botones para que el usuario pueda moverse por un menú de opciones.

3.1.4 Comunicación

Para ampliar las capacidades del dispositivo, se implementan en el diseño dos interfaces de comunicación: una alámbrica (USB) y otra inalámbrica (Bluetooth), para poder intercambiar los datos registrados y calculados con otros dispositivos que el paciente o el médico pudiesen usar.

3.1.5 Fuente de alimentación

Como fuente de alimentación se dispondrá de una batería que permita el funcionamiento de todo el sistema. Esta podrá cargarse a través de la interfaz USB.

3.1.6 CPU

Se selecciona en un primer momento como unidad de procesamiento un microcontrolador. La elección de este se justificará en los siguientes apartados.

3.2 Diseño Hardware

El diseño hardware de este proyecto se ha dividido en ocho bloques que representan cada uno de los bloques principales que se han identificado.

3.2.1 Diseño esquemático

3.2.1.1 Analog front end (AFE)

Este primer apartado fue el primero en ser acometido en la realización este trabajo. Se trata de la etapa encargada de medir la señal de saturación de oxígeno y enviarla al microcontrolador para su posterior procesado. Las señales en medicina tienen la peculiaridad de ser pequeñas y de estar alteradas, a veces en gran medida, por el ruido. Por eso, la amplificación, el muestreo y la conversión analógica-digital deben hacerse con especial cuidado para mantener la integridad de la información. El dispositivo a usar debe asegurar un bajo nivel de ruido en cada una de sus etapas así como un adecuado tratamiento de estas señales biomédicas.

Desde un principio, se partió del conjunto de soluciones que la empresa Texas Instruments ofrece [11] en general para el desarrollo de dispositivos médicos y en particular para los de pulso oximetría. El circuito integrado seleccionado para este bloque es el AFE4490 [12], una etapa analógica que integra el amplificador de transimpedancia, el filtro, el conversor analógico-digital y el controlador de los leds emisores, especialmente diseñado para esta aplicación. La comunicación con el microcontrolador se lleva a cabo a través del puerto SPI.

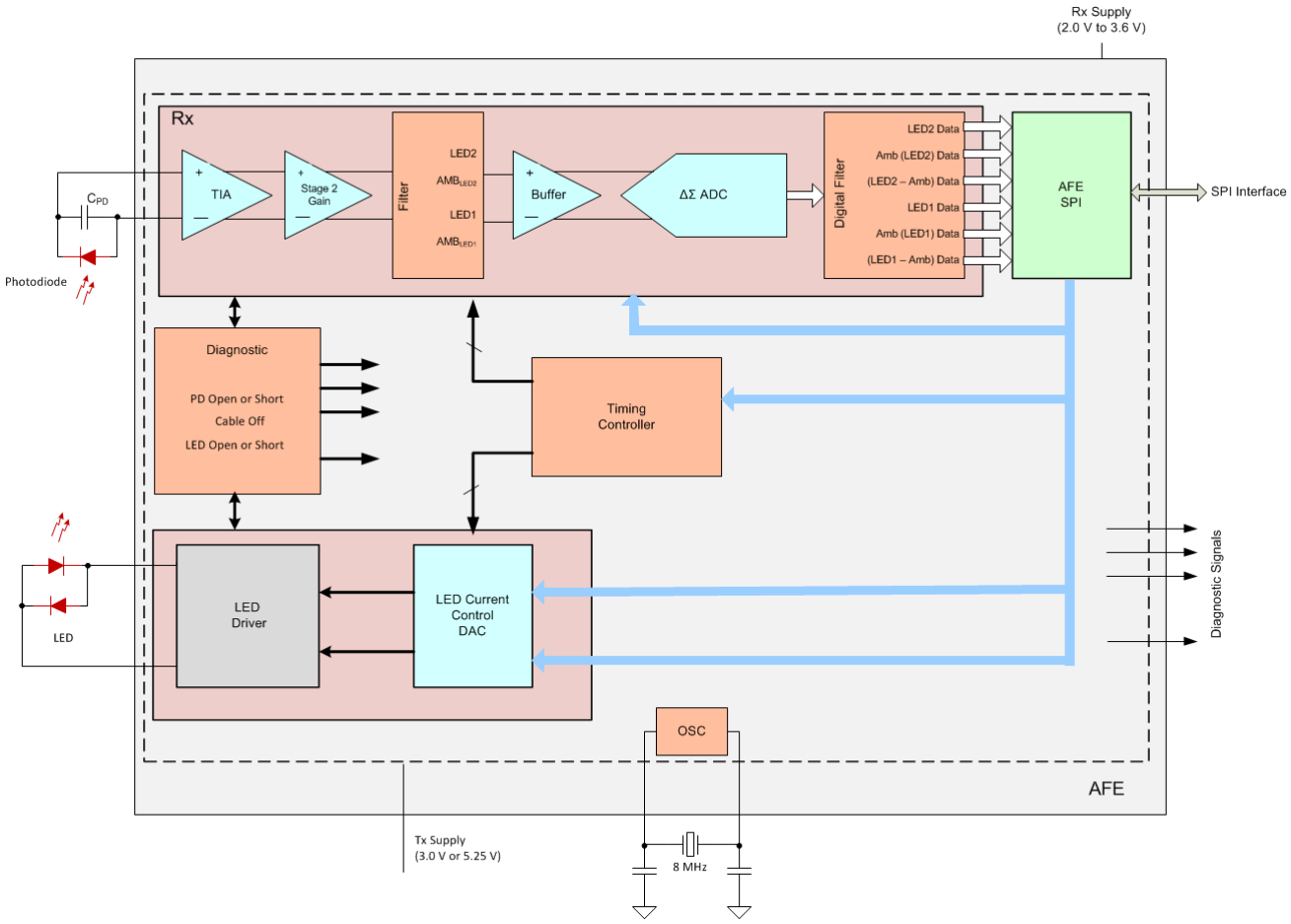


Figura 9: Analog front end AFE4490

Para la medición de la absorción de la luz roja por la oxigenación de la sangre, se empleará un sensor comercial [41], cuyo conector al dispositivo será un puerto DB9. Este sensor consta de dos diodos LED emisores, uno de longitud de onda que se corresponde con el color rojo y otro con el infrarrojo, conectados inversamente, de modo que el pulso de luz emitido será intercaladamente de uno u otro diodo. En el otro extremo del sensor se encuentra un fotodetector que recibirá la luz que no haya sido absorbida por el tejido, componiendo la señal de PPG de la cual se podrá extraer el nivel de oxigenación, el flujo de sangre y la frecuencia cardíaca. Existen otros sensores disponibles en el mercado para realizar esta medición en otras áreas periféricas como el lóbulo de la oreja o la frente pero en este trabajo se ha optado por el sensor de dedo, debido al carácter de prototipo de este diseño ya que este tipo de sensores son reutilizables.

En el Anexo I, en el Plano 1, se puede ver con detalle cómo ha sido la configuración de este dispositivo. Para la elección de los componentes se ha tenido presente su hoja de características. Cabe destacar el componente que se encuentra entre la entrada y salida del sensor DB9 y el circuito integrado denominado *voltage clamper*.

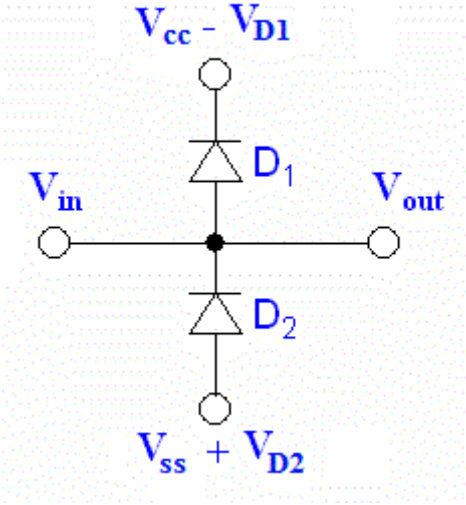


Figura 10: Voltage clamper

Esta etapa ha sido configurada para que los pulsos emitidos y recibidos se encuentren siempre comprendidos en un rango fijo de tensión para que su valor sea conocido en todo momento. Esta técnica se ha tomado siguiendo las pautas de dos modelos de referencia que pone a disposición Texas Instruments, [14] y [15], donde el *clamp* de las señales se hace entre su alimentación y su tierra.

Según [13], para minimizar el ruido (uno de los factores más importantes a tener en cuenta en este dispositivo) las tierras analógicas y digitales deben separarse. A la tierra analógica del sistema se conectarán las tierras del conversor analógico-digital (RX_ANA_GND y RX_DIG_GND en el Anexo I, Plano 1) y el resto a la tierra digital del sistema. No debe confundirse RX_ANA_GND con la etapa analógica y a RX_DIG_GND con la etapa digital. En realidad, corresponden a las referencias analógica y digital del conversor interno del AFE4490 y deben cortocircuitarse externamente. El motivo por el que no se hace esto internamente es porque las vías del circuito integrado no son capaces de soportar las altas corrientes.

La alimentación de la parte digital será, como recomienda su hoja de características, de 5V mientras que la de la parte analógica será de 3V. La tierra de ambas partes no podrá diferir en más de 0.3V, por lo que entre ambas deberá conectarse una resistencia o un jumper.

3.2.1.2 Pantalla

La funcionalidad para la que se ideó la incorporación de una pantalla es representar la información útil al usuario y hacerlo así independiente de cualquier otro equipo para su uso.

Los requisitos que debe tener una pantalla para esta aplicación no son muy exigentes, siendo válidas una amplia gama de estas. En este trabajo se ha decidido usar una pantalla

alfanumérica de 16x2 celdas de tecnología HD44780 por tratarse de una pantalla ampliamente extendida en el mercado y de bajo precio, lo que ofrece una extensa comunidad de ayuda para el soporte técnico, además de por haber trabajado anteriormente con ella el autor del trabajo, lo que reduciría el tiempo empleado en la fase de programación.

Para saber si la pantalla era apta para esta aplicación, se realizó una prueba en la que se representaron todos los posibles datos que en un futuro se mostrarían al usuario.

Figura 11: Pruebas de impresión en pantalla

En el Anexo I, en el Plano 2, se puede ver con detalle cómo ha sido la configuración de este bloque. Para la elección de los componentes se ha tenido presente su hoja de características [16]. La alimentación determinada ha sido de 5V (conectada a LED_DRV_SUP en Anexo I, Plano 2).

Para usar un menor número de pines del microprocesador, facilitando así su posterior diseño a nivel layout, la información de la pantalla se transmitirá en modo 4 bits, es decir, por nibbles. Cada vez que se quiera imprimir algo por la pantalla, serán necesarias dos instrucciones en las que se enviarán 4 bits de los 8 necesarios. Esto, aunque reduce la velocidad de

comunicación, permite disminuir a la mitad el número de pines necesarios en el microcontrolador, algo a tener en cuenta en dispositivos donde se quiere reducir el tamaño, ya que permite emplear microcontroladores LPC (low pin count).

3.2.1.3 USB

El microcontrolador elegido para este dispositivo integra una interfaz de comunicación USB (capa física). Este no fue un factor tenido en cuenta para la elección del microcontrolador, pues de no disponer de esta interfaz, podría haberse utilizado un componente que actuase de puente entre la UART del microcontrolador y el puerto USB de forma cómoda.

No obstante, al estar integrada esta interfaz, en el diseño de este bloque tan sólo es necesaria la consideración de una protección contra las posibles descargas electroestáticas procedentes del contacto del usuario con el conector. Como puede verse en [19], los componentes protectores de descargas electroestáticas son numerosos y los criterios de elección han sido el tamaño y el nivel de protección. En el Anexo I, en el Plano 3, se puede ver con detalle cómo ha sido la configuración de este bloque. Para la conexión del protector TPD4E004 [18] se ha tenido presente su hoja de características.

El conector USB elegido ha sido el tipo mini-B [17], teniendo en cuenta el requisito de hacer un dispositivo pequeño y compacto. La carcasa de este conector es conectada a la tierra del dispositivo a través de una resistencia.

3.2.1.4 Bluetooth

Para la etapa de Bluetooth se ha seleccionado el componente integrado que ofrece Texas Instruments LMX9838 [21]. La justificación de esta elección se expone a continuación.

La idea de partida era crear una etapa que pudiese comunicarse con otros dispositivos haciendo uso de la tecnología inalámbrica Bluetooth [20], existente en gran número de dispositivos móviles y ordenadores. Al tratarse de un dispositivo portable y autónomo, el consumo de batería de cada componente estuvo en todo momento analizado durante el diseño hardware. Es por eso que en un primer momento se planteó el uso de la variante Bluetooth denominada Bluetooth Low Energy (conocido como BLE), un protocolo de menor consumo ideado para dispositivos como el que se estaba diseñando. Sin embargo, a la hora de elegir el componente, apareció el inconveniente de que la disponibilidad en el mercado no era como la del protocolo Bluetooth convencional. Los distintos módulos de BLE que se estudiaron implicaban un mayor número de componentes adicionales, así como el diseño de la antena de transmisión, complicando el planteamiento del componente, por lo que el uso de esta variante del protocolo tuvo que ser desechada.

Tras ese estudio, se determinó que el componente que se emplearía para la comunicación inalámbrica sería el módulo mencionado con anterioridad, que incorpora todo lo necesario para realizar esta comunicación, simplificando su diseño, a excepción de algunos componentes pasivos.

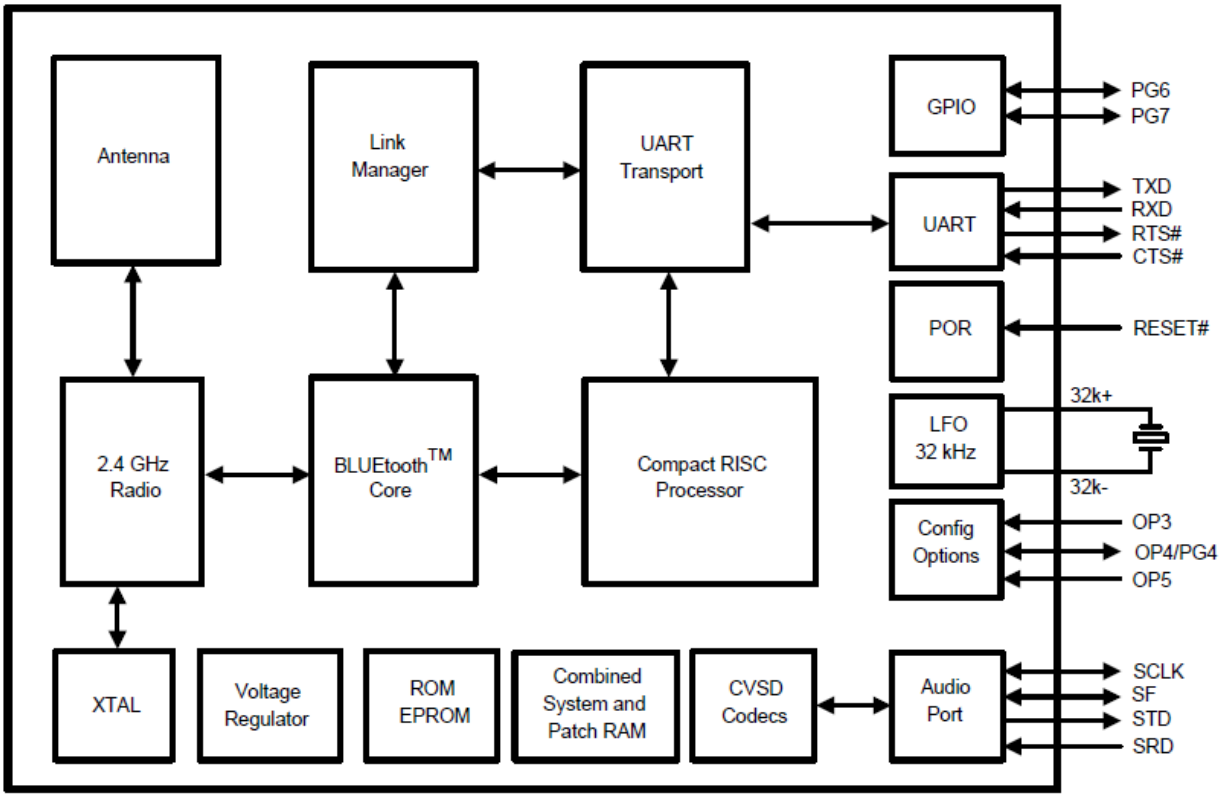


Figura 12: Diagrama de bloques del módulo Bluetooth LMX9838

Aunque esto implica renunciar a un consumo reducido cada vez que se transmitiese la información por Bluetooth, se ha decidido incorporar este componente favoreciendo otros dos de los requisitos: bajo coste y reducido tamaño.

En el Anexo I, en el Plano 4, se puede ver con detalle cómo ha sido la configuración de este componente. Para la elección de los componentes se ha tenido presente su hoja de características y del modelo de referencia [22]. Para minimizar el consumo, se ha incorporado un reloj de 32 KHz como recomienda el fabricante, lo que posibilita un funcionamiento óptimo del módulo. La tensión de alimentación será de 3V (conectado al pin MCU_DVCC, en el Anexo I, Plano 3), ya que este valor está dentro del rango admisible por el módulo de Bluetooth y permite simplificar la etapa de reguladores de tensión.

La velocidad de transmisión de la UART se deja a elección, sustituyendo las conexiones físicas por jumpers que permiten decidir si esta será de 9.6kbps, 115.2kbps o 921.6kbps. El motivo

de esta decisión es aumentar el margen de maniobra en la fase de desarrollo software de este trabajo, para la que se deja la tarea de elegir esta tasa según aporte mejores prestaciones.

Añadir que en este bloque se incorporan dos LED para informar al usuario del estado de la comunicación inalámbrica (LED3 encendido cuando haya conexión y LED4 parpadeando mientras se encuentre transmitiendo, en Anexo I, Plano 3).

3.2.1.5 Batería

La alimentación del sistema procederá de una batería de Ion-Litio de 3.7V, 1.5A y una capacidad de entre 300 y 6000mAh. Se ha elegido esta tras seleccionar el cargador y el controlador de carga.

Para aumentar el tiempo de vida del dispositivo que se diseña en este trabajo o prescindir de continuas reposiciones de la batería, se determinó que esta podría cargarse cuando el dispositivo se conectase mediante USB al ordenador del usuario. Para ello se analizaron las distintas posibilidades que existían en el mercado [23] y se determinó que este componente sería el BQ24072 [24], un cargador de baterías especialmente diseñado para la tecnología Ion-Litio y USB-friendly, lo que lo hace idóneo para el propósito que se tiene. Este componente permite alimentar el dispositivo independientemente de la gestión de la carga de la batería. Esto implica que no será necesaria una carga de la batería con el dispositivo en reposo sino que puede aprovecharse la conexión de este con el ordenador del usuario mientras se transmite la medición para cargar la batería. Aun así, el método de carga que permite es tanto un puerto USB como un adaptador de corriente alterna.

En el Anexo I, en el Plano 5, se puede ver con detalle cómo ha sido la configuración de este componente. Para la elección de los componentes se ha tenido presente su hoja de características. Entre las decisiones que se han tomado cabe destacar la de no limitar la corriente de entrada (máxima corriente). Los cálculos de algunos de los valores de los componentes de este bloque se han guiado de [26]. También añadir que la funcionalidad de *termination charge* se ha deshabilitado. El fabricante ofrece la posibilidad de desconectar el cargador una vez se ha alcanzado el 100% de su capacidad, sin embargo, experimentalmente se ha demostrado [27] que cuando se carga la batería a la vez que hay conectada una carga a la salida (como es el caso de este dispositivo, en el que se pretende que realice mediciones mientras se carga) se generan picos de tensión que el cargador puede detectar erróneamente como un estado de carga completado (la decisión se basa en el nivel de intensidad). Esto produce que la batería no llegue a cargarse por completo, por lo que se ha decidido deshabilitar esta funcionalidad y no interrumpir la carga.

Para este componente, además, se han incluido dos diodos LED indicadores para informar al usuario del estado de la batería (LED1 encendido para indicar que la batería es apta, en Anexo I,

Plano 5) y del estado de la carga (LED2 encendido para indicar que la batería se está cargando, en Anexo I, Plano 5).

Aun habiendo diseñado un sistema autónomo, con batería incluida, y un cargador para aumentar su tiempo de autonomía o el periodo de reposición, se hace imprescindible el uso de un controlador de carga para que el microcontrolador conozca en todo momento el estado de la batería. De este modo, se podrá informar al usuario de cuándo necesita poner en carga el dispositivo y determinar si será posible o no una nueva medición con el nivel de carga actual. Para ello se hizo, como en los casos anteriores, un análisis de las soluciones que ofrecían los fabricantes y se seleccionó el componente BQ27200 [25] de Texas Instruments. El motivo por el que se selecciona nuevamente de este fabricante es para asegurar un correcto funcionamiento del conjunto de los dispositivos, tomando como criterio la interoperabilidad entre integrados en este proyecto de prototipado por encima de unas características óptimas que podrían haberse encontrado en algún componente de otro fabricante.

El BQ27200 ha sido seleccionado por funcionar con baterías de Ion-Litio y por tratarse de un módulo autónomo e integrado en un único componente. Se conecta directamente a los terminales de la batería para conocer con precisión el estado de esta y permite una compensación por temperatura (en caso de disponer la batería de un terminal termistor). La información que obtiene (tiempo para descarga, corriente, capacidad, estado de la carga, temperatura,…) se transmite al microcontrolador mediante la interfaz I2C. En el Anexo I, en el Plano 5, se puede ver con detalle cómo ha sido la configuración de este componente. Para la elección de los componentes se ha tenido presente su hoja de características. La etapa que puede verse en los puertos SCL y SDA pertenecientes a la interfaz I2C está configurada para que actúe como protección contra descargas electroestáticas e impedir que estas dañen el microcontrolador.

3.2.1.6 Memoria

Como se introdujo al comienzo, se plantea la posibilidad de almacenar toda la medición de PPG en una memoria y que esta sea una tarjeta SD. Concretamente, se trataría del tipo micro-SD, una tarjeta de memoria ampliamente extendida y compatible con multitud de dispositivos, posibilitando también el traspaso de la información de un nuevo modo, es decir, directamente insertando la tarjeta en otro dispositivo.

Para este bloque, como en el caso de la interfaz USB, tan sólo es necesario el conector adecuado para este tipo de memorias [28]. Nuevamente, se incorpora un protector contra descargas electroestáticas procedentes del contacto entre el usuario y el conector aunque en este caso se ha utilizado el componente TPD8E003 [29]. El motivo de la elección de este protector y no el mismo que en el caso del USB se encuentra en la fase del diseño del layout de este dispositivo,

ya que no era posible el rutado del TPD4E004. El TPD8E003, de similar composición, sí permitía una cómoda conexión entre las distintas partes, reduciendo la longitud de las pistas y el número de vías.

En el Anexo I, en el Plano 6, se puede ver con detalle cómo ha sido la configuración de este bloque. Para el conexionado con el resto del sistema se ha tenido presente la hoja de características del conector y del protector. La comunicación entre la memoria y el microcontrolador se realiza mediante la interfaz SPI en vez de uno de los modos que ofrece la tecnología SD, para facilitar su programación, por lo que será necesario disponer de una tarjeta que soporte este protocolo. Para alimentar de este bloque se ha tomado la tensión de 3V (conectado a MCU_DVCC en Anexo I, Plano 6).

3.2.1.7 Reguladores de tensión

En este bloque se ha agrupado la etapa encargada de suministrar el nivel de alimentación requerido por cada componente del dispositivo.

Tabla 2: Niveles de tensión por componente

Componente	Tensión
AFE – Parte analógica	3V
AFE – Parte digital	5V
Pantalla LCD	5V
Módulo Bluetooth	3V
Memoria SD	3V
Protector USB	3V
Microcontrolador	3V

En la Tabla 2 se agrupan los componentes según su tensión de alimentación. Como puede observarse, durante la etapa de diseño a nivel hardware se ha tenido en cuenta estos valores para simplificar este bloque de reguladores de tensión. Por este motivo, el resultado es que el sistema dispone de sólo dos valores distintos de alimentación.

Este bloque de reguladores de tensión se puede dividir a su vez en dos etapas. Una primera etapa que es necesaria para convertir la tensión de alimentación que aporta la batería (entre 3.7 y 4.4V) a una más alta de la que podamos obtener los niveles necesarios en la segunda etapa (los 3 y 5V).

Por eso, en primer lugar se tiene el componente TPS61093 [31], un elevador de tensión de bajo nivel de entrada. Entre las características que han hecho que sea este el integrado

seleccionado [30] destaca un desacoplo entre la alimentación a la entrada y la tensión regulada a la salida, un diseño pensado para trabajar con baterías de Ion-Litio, un bajo consumo y un tamaño reducido. Otra de las características fundamentales fue su bajo ruido, diseñado especialmente para funcionar en dispositivos médicos.

En el Anexo I, en el Plano 7, se puede ver con detalle cómo ha sido la configuración de este componente. Para la elección de los componentes se ha tenido presente su hoja de características. En este circuito integrado se selecciona el nivel de tensión a la salida modificando el valor de dos resistencias (las conectadas entre OUT y FB y entre FB y GND en Anexo I, Plano 7). Sus valores se han elegido para tener un nivel a la salida de 8.97V, siguiendo las recomendaciones de [14].

La segunda etapa de este bloque tiene como entrada esta tensión de 8.97V y aporta dos salidas de 5 y 3V respectivamente.

Para conseguir el nivel de 5V (LED_DRV_SUP y TX_CTR_SUP en Anexo I, Plano 7), es decir, la alimentación de la etapa digital del AFE así como de la pantalla, se ha utilizado el regulador de tensión de bajo nivel de ruido LP3878 [32]. A pesar de existir en el mercado [30] reguladores que aportasen un nivel de tensión de 5V con un menor nivel de ruido, se ha sacrificado en un grado esta característica para conseguir un componente capaz de proporcionar una mayor corriente, necesaria para el correcto funcionamiento de los dos componentes ya mencionados. En concreto, el AFE requiere una corriente de hasta 200mA y la pantalla funciona con una corriente de 120mA. El LP3878 es capaz de suministrar una corriente de 800mA a su salida manteniendo una tensión constante y precisa a la salida.

En el Anexo I, en el Plano 7, se puede ver con detalle cómo ha sido la configuración de este componente. Para la elección de los componentes se ha tenido presente su hoja de características. El nivel de tensión se determina mediante el valor de dos resistencias y un condensador (conectados entre OUT y ADJ y entre ADJ y GND en Anexo I, Plano 7).

El nivel de 3V (RX_ANA_SUP, RX_DIG_SUP, MCU_AVCC y MCU_DVCC en Anexo I, Plano 7) que alimenta el resto de componentes se ha obtenido utilizando el módulo regulador de tensión de bajo ruido TPS7A4901 [33], de la misma familia que el convertidor de baja entrada. Se ha hecho así para optimizar el funcionamiento en conjunto de ambos componentes y teniendo en cuenta las prestaciones que ofrece contra el ruido. Para el dispositivo, se ha decidido utilizar dos componentes del mismo modelo en vez de sólo uno para poder suministrar de manera independiente la alimentación del AFE y la del resto de componentes, desacoplando así la etapa de adquisición de la señal de la de procesado y comunicación. Otro motivo para utilizar dos componentes integrados en vez de uno ha sido que el nivel máximo de corriente a la salida capaz de aportar es de 150mA, una corriente insuficiente si tuviese que alimentar a todo el bloque de 3V del dispositivo, impidiendo un correcto funcionamiento del sistema.

En el Anexo I, en el Plano 7, se puede ver con detalle cómo ha sido la configuración de este componente. Para la elección de los componentes se ha tenido presente su hoja de características. El método de configuración de la tensión a la salida es la misma que en los dos casos anteriores; un condensador y dos resistencias (conectados entre OUT y FB y entre FB y GND en Anexo I, Plano 7).

Las inductancias que se conectan a las salidas de los reguladores de tensión se añaden para minimizar el ruido que generan los reguladores de tensión y para aislar también la alimentación digital de la analógica (del mismo modo que se ha separado la tierra digital de la tierra analógica).

3.2.1.8 Microcontrolador

La elección del microcontrolador fue la última decisión que se tomó en el diseño hardware del dispositivo. Muchos de sus requisitos estaban determinados desde el principio aunque no todos. Se realizó un análisis para determinar cuál sería el más apropiado y se estudiaron las distintas alternativas existentes en el mercado. Estos fueron unos de los puntos que se tuvieron en cuenta a la hora de realizar este análisis:

1) Herramientas de desarrollo: la fase de programación es una parte importante de este trabajo por lo que se buscó un microcontrolador cuya herramienta de desarrollo estuviese a disposición del autor de este documento o que hubiese sido utilizada por el mismo con anterioridad para que el prototipado fuese más cómodo.
2) Base de conocimientos: debido al gran número de microcontroladores existentes en el mercado, una MCU utilizada con anterioridad por el autor del trabajo sería más oportuna para su uso en el dispositivo, o al menos se buscaría que ese microcontrolador estuviese extendido en el mercado, para poder contar con un gran número de aplicaciones prácticas reales, con información abundante sobre su uso y características y una base de datos donde se recopilase información sobre este.
3) Precio y disponibilidad: debido a que uno de los requisitos era la portabilidad y un posible uso extendido entre los usuarios, el costo de este componente, así como el del resto de componentes del dispositivo, era otro de los factores determinantes.
4) Familia versátil y desarrollo: como se ha mencionado, desde un principio no se partió de un microcontrolador específico, sino de unos requerimientos. Es por eso que una familia de microcontroladores sería la mejor opción en este caso, para conocer algunos rasgos característicos a tener en cuenta durante la fase de diseño hardware y luego poder cómodamente seleccionar el más adecuado. Además, una familia de microcontroladores permite en un futuro reemplazar el microcontrolador por otro más actual o de mejores

prestaciones sin implicar un gran número de cambios en el resto del sistema (se recuerda que este dispositivo tiene carácter de prototipo y es susceptible a cambios en un futuro).

5) Cantidad de memoria y periféricos: este punto es el que en un principio no permitió seleccionar una solución determinada. El número de periféricos o la comunicación con estos era desconocida *a priori*, así como la capacidad de memoria necesaria.
6) Solidez del fabricante y posibilidades de segundas fuentes.

Teniendo en cuenta estos seis puntos, se seleccionó la familia MSP430 del fabricante Texas Instruments [34] como solución del diseño, una conocida familia de microcontroladores de bajo consumo, orientado a este tipo de aplicaciones. Dentro de esta familia se encontraron varios componentes diseñados especialmente para ser integrados en dispositivos médicos y, más concretamente, en pulso oxímetros. Además, uno de los microcontroladores que componen esta familia había sido utilizado por el autor del proyecto en otra ocasión. De igual modo, el entorno de desarrollo Code Composer Studio ya había sido utilizado para la programación de aplicaciones en circuitos de este fabricante.

Sin embargo, en un principio no se podían determinar detalladamente los requisitos que eran necesarios cumplir por parte de este microcontrolador, ya que se desconocían de partida los requerimientos que exigirían el resto de componentes del sistema. Es por esto que, una vez llegados al final del diseño hardware, se recopilaron los requisitos necesarios para que los periféricos seleccionados funcionasen.

Uno de los puntos claves en esta decisión fue la interfaz de comunicación que posibilitaba el traspaso de información entre la MCU y los periféricos. Se determinó que era necesaria una interfaz I2C que permitiese conocer al microcontrolador el estado de la batería del que informaba el componente BQ27200.

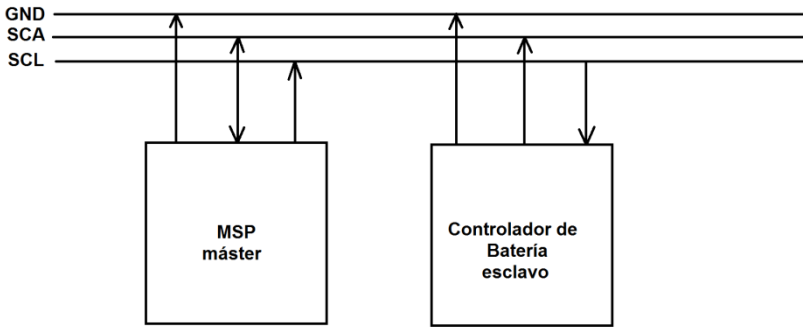


Figura 13: Interfaz I2C

Por otro lado, era necesaria una interfaz SPI para conectar el MSP con el AFE y con la tarjeta de memoria SD.

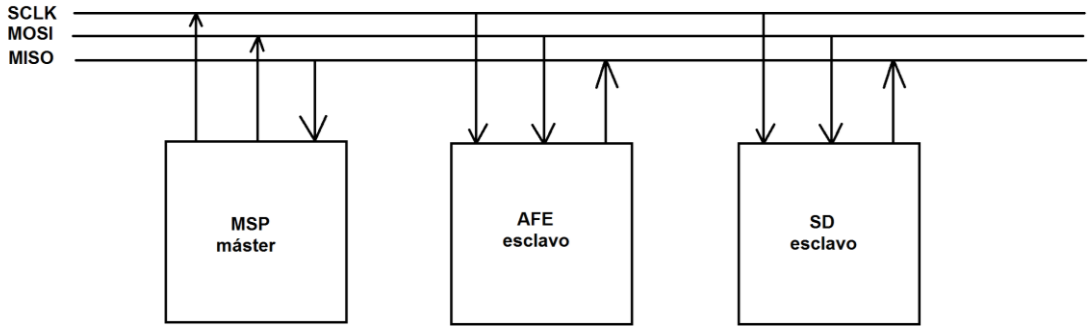


Figura 14: Interfaz SPI

Y, por último, el sistema requería una interfaz de comunicación serie (UART) para intercambiar información con el módulo de Bluetooth y con el puerto USB. Como se mencionó en el bloque USB, el microcontrolador seleccionado integra la función de comunicación USB, por lo que el único componente que requiere UART es el Bluetooth.

Para programar el microcontrolador, se analizaron las diferentes opciones existentes. Lo más común era utilizar el puerto JTAG al que se accedía a través de un programador conectado al PC de desarrollo. Sin embargo, existe otra opción denominada Spy-by-wire, que consiste en el uso de únicamente dos hilos para esta programación. El desarrollo, inevitablemente, será más lento de este modo, pero su uso evita tener que disponer de la herramienta programadora.

Una vez identificadas todas las especificaciones del microcontrolador, se procedió a su elección, siendo el MSP430F5529 el elegido [35], un microcontrolador que cumple todos los requisitos y con puertos de propósito general suficientes para controlar todo el sistema.

Tras el estudio y de estos bloques y la elección de sus componentes y conexiones, llegados a este punto del trabajo se obtiene un diseño a nivel esquemático completo del dispositivo. Como entregables de esta fase se consideran un listado de todos los nodos del dispositivo, que es una descripción detallada y simplificada de todas las conexiones del sistema, así como un listado de materiales que se detalla en el Anexo II. Ambos documentos serán el punto de partida para la siguiente fase, el diseño layout.

3.2.2 Diseño layout

Una vez que se ha definido exactamente el dispositivo a nivel hardware, identificando los componentes, conexiones y características técnicas, se procede a diseñar el circuito PCB que se fabricará posteriormente.

Para ello, se hace un análisis previo sobre las técnicas de fabricación que se usarán para construir el dispositivo así como un estudio del reparto del espacio en la placa. Se determina que la placa PCB tendrá dos caras a las que se conectaran los componentes, preferentemente SMD, para cumplir con el requerimiento de tamaño reducido y compacto.

Durante esta fase se ha tenido como referencia bibliográfica el libro del autor Gerald L. Ginsberg [36].

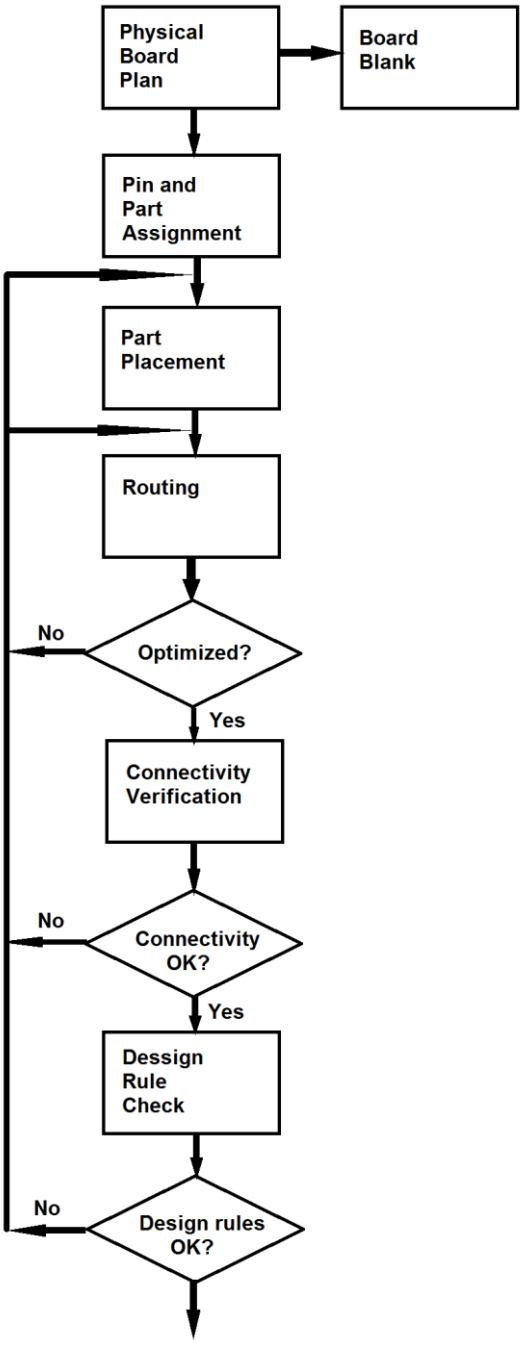


Figura 15: Diagrama de diseño del PCB

Siguiendo su metodología, se planteó un primer modelo del posicionamiento en la placa, intentando separar los componentes de mayor nivel de ruido de aquellos más vulnerables, así como orientar estos de modo que su rutado fuese más simple para optimizar las longitudes de las pistas.

Así, por ejemplo, la etapa de comunicación se sitúa en el extremo opuesto a la etapa de adquisición de la señal del sensor de pulso oximetría. El conector de la pantalla y los botones de actuación del usuario se han dispuesto teniendo en cuenta el posible aspecto final del dispositivo. La etapa de alimentación y regulación de tensión se ha agrupado y se ha intentado situar en un espacio separado del resto de señales.

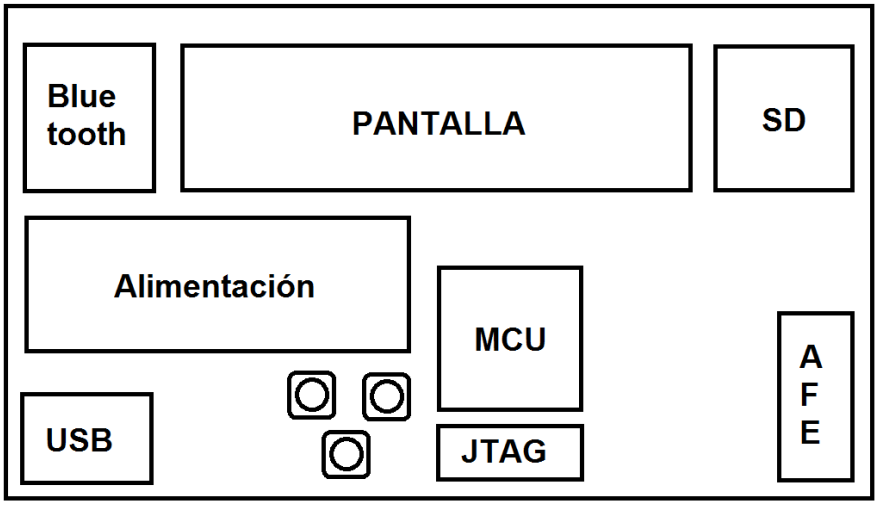


Figura 16: Esquema del diseño de la huella PCB

Tras el planteamiento inicial, se procedió a la orientación de los componentes para un rutado más directo y evitar en gran medida los rodeos y el uso de vías. Tras esto, se ajustó el planteamiento inicial y se procedió a la colocación de cada huella de los componentes en el lugar asignado.

Para el rutado de las conexiones, se tomó el criterio de dibujar primero las pistas que portarían las señales del sistema y dejar para el final el rutado de la alimentación. Esto es así porque se considera que un correcto funcionamiento del dispositivo depende en gran medida del tratamiento que se haga con las señales que se manejan. Como se ha comentado anteriormente, los instrumentos que trabajan con señales biomédicas están diseñados para ser robustos al ruido que puede llegar a destruir estas pequeñas señales. Es por eso que las primeras pistas en ser dibujadas fueron las que conectaban el conector DB9 del sensor de pulso oximetría con el componente AFE y las que comunicaban a este con el microcontrolador.

Posteriormente se trazaron las vías que comunican el microcontrolador y la memoria, donde se almacenará la medición para su posterior procesado; la interfaz UART del Bluetooth y las pistas que conectan el puerto USB.

Siguiendo las recomendaciones del fabricante de los componentes, las vías INN e INP (en Anexo I, Plano 1), que reciben la información recogida por el fotodetector del sensor, se trazan paralelas, así como TXN y TXP (en Anexo I, Plano 1), que envían los pulsos a los diodos LED rojo e infrarrojo necesarios para componer la gráfica de PPG, y DM y DP (en Anexo I, Plano 3), que son las conexiones de la comunicación diferencial USB. No obstante, este criterio para rutar paralelas las pistas que transportan información entre los mismos componentes se ha intentado mantener como regla para el rutado de todo el dispositivo, siendo este caso el de la interfaz I2C y SPI entre otras.

Después se procedió al rutado de la etapa de alimentación. Durante la fase de diseño físico se ha intentado posicionar la mayoría de los componentes en la cara superior, minimizando el uso de la inferior y relegándola a un uso para solventar conflictos irresolubles a una cara o simplificable a dos caras, para evitar el uso de las vías y el paso de las señales de una cara a otra, siguiendo las recomendaciones que hacen los autores que han escrito al respecto. Sin embargo, en este bloque de alimentación se han tenido que posicionar los reguladores de tensión en la capa inferior para no interferir con las vías de señal y para intentar que las vías de alimentación fuesen lo más directas posibles y tan sólo cruzasen la placa de una cara a otra por una única vía.

Por último, se procedió a la colocación del resto de componentes que forman el sistema (resistencias, condensadores, inductancias,…). El tamaño de estas se ha elegido por disponibilidad, siendo el único criterio a tener en cuenta el tamaño, intentando que fuese el menor posible. Se intentó que su colocación no perjudicara a los caminos dibujados hasta el momento y tan sólo fueron necesarios unos cambios bien en estos o en el esquemático. También fueron hechos varios cambios en la asignación de los pines de propósito general del microcontrolador, para reducir la longitud de las pistas y minimizar el uso de vías para conectar los equipos. En el Anexo I, Plano 9 y Plano 10 puede verse con más detalle del dibujo final de las huellas de los componentes en la cara superior e inferior respectivamente.

En esta fase se tuvieron en cuenta las especificaciones de los fabricantes, que recomiendan situar todos los componentes que acompañen al mismo integrado en la misma cara y situar los condensadores de desacoplo lo más cerca posible de los nodos que se quieren proteger. No se colocó ningún test point ni signal point para comprobar el correcto funcionamiento del PCB a nivel físico ya que no se prevé proteger las caras con plástico y los pads, las vías y las pistas podrán actuar directamente como puntos de comprobación.

Una vez finalizado el rutado de las huellas de los componentes, se procedió a una inspección y optimización. Se llevaron a cabo varios cambios, donde algunas huellas se

desplazaron, sin afectar a su rutado, para permitir simplificar alguna conexión, otras pistas se consiguieron trazar por completo en la misma cara e incluso se consiguió reducir el tamaño final del dispositivo en un par de centímetros.

Para dar por finalizada esta fase de diseño a nivel físico, se procedió a una comprobación del conexionado, tomando como referente el listado de nodos y componentes que se obtuvo en la fase anterior de diseño a nivel esquemático. Se procedió a la corrección de los pequeños errores que aparecieron y se sometió el diseño a un DRC en el que se evaluaran las reglas de diseño ajustadas a las especificaciones de las máquinas que fabricarían la PCB posteriormente.

3.3 Diseño Software

Una vez conocido físicamente el dispositivo, se procede a programar sus funcionalidades. Este proceso se lleva a cabo mediante lenguaje de alto nivel (C), además del ensamblador imprescindible de cada componente.

Nuevamente se ha dividido el trabajo en varios bloques (esta vez siete) para facilitar su programación. El método de operación que se ha seguido ha sido el de buscar librerías para los distintos componentes, adaptarlas al dispositivo y finalmente crear el programa principal que se ejecute en el microcontrolador.

3.3.1 Diseño de los diagramas de flujo

La primera fase de la etapa de diseño software ha consistido en una descripción a modo de diagramas de flujo de cada uno de los bloques en los que se ha dividido el diseño. Con esto se pretende describir funcionalmente cada componente, haciéndolo independiente del lenguaje de programación usado e incluso del componente hardware elegido, por si en un futuro decidiese cambiarse este por otro.

3.3.1.1 Proceso de impresión de mensajes por pantalla

El diagrama de flujo que rige el funcionamiento de la pantalla representa un claro resumen del funcionamiento del dispositivo, por eso su explicación se pone en primer lugar. En el trabajo también se acometió en primer lugar ya que aporta la estructura básica del programa a partir de la cual se puede ir completando cada bloque de programación (desarrollando el contenido de cada estado).

El diagrama de flujo diseñado para la posterior programación del bloque de representación visual se muestra en la Figura 17 y su explicación se detalla a continuación.

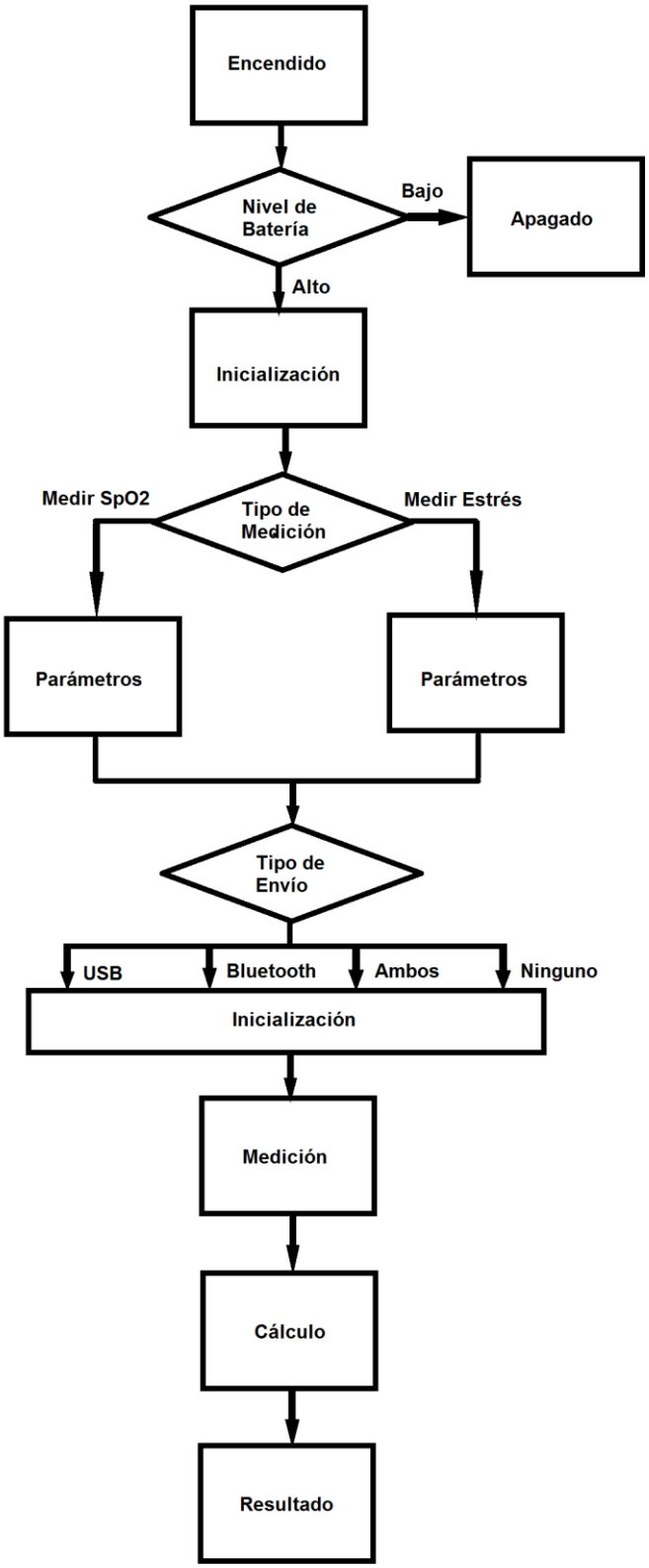


Figura 17: Diagrama de flujo de la pantalla

Al encender el dispositivo, se comprobará el estado de la batería para saber si esta permitirá realizar una medición o no. Si el nivel de carga es insuficiente, se mostrará el mensaje que aparece en la Figura 18:

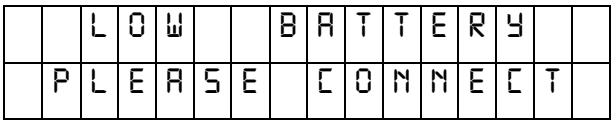


Figura 18: Mensaje de batería baja

Si el nivel es adecuado, el microcontrolador inicializará todos los periféricos:

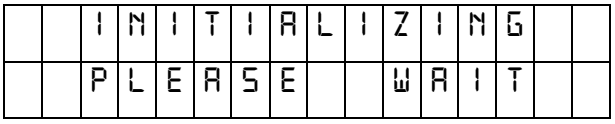


Figura 19: Mensaje de inicialización

Una vez se haya inicializado correctamente el dispositivo, por la pantalla se representará un menú de opciones. En primer lugar, dará a elegir entre medir el nivel de SpO2, como un pulso oxímetro común, o detectar el estrés, que es el propósito de este trabajo.

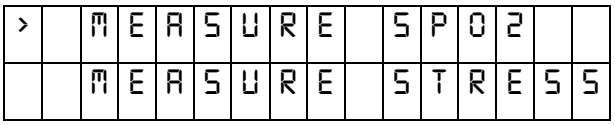


Figura 20: Mensaje de selección de la medición

La elección de una u otra opción se realizará mediante los tres botones de los que dispone el dispositivo. Con los botones de la derecha y de la izquierda (conectados a SE y NO respectivamente en Anexo I, Plano 8) se mueve la flecha que aparece en la pantalla arriba y abajo, modificando el valor de la variable *measure* (Anexo III, Código del dispositivo), para poder posteriormente ejecutar una rutina u otra. Cuando se pulsa el botón central (conectado a OK en Anexo I, Plano 8), se confirma la selección y se pasa al siguiente menú.

El siguiente menú sirve para seleccionar alguna interfaz de transmisión de datos. El dispositivo cuenta con una interfaz USB y una inalámbrica (Bluetooth). En este paso se puede decidir si transmitir la información por alguna de estas dos interfaces, por ambas o preferir representar simplemente los resultados en la pantalla.

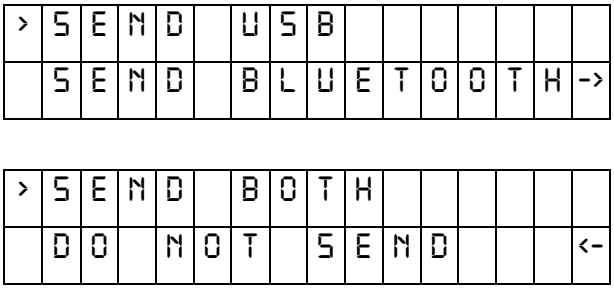


Figura 21: Mensajes de selección de las interfaces de comunicación

Una vez se ha decidido qué medición realizar y por qué medio transmitir esta información, se inicializará la medición de la señal de PPG:

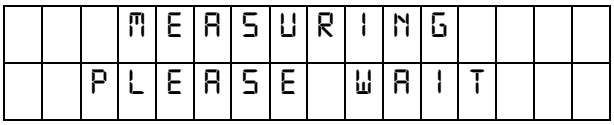


Figura 22: Mensaje de medición

Esta pantalla permanecerá fija durante la medición. La duración de esta medición dependerá de si se ha seleccionado SpO2, en cuyo caso será inferior a 20 segundos, o Stress, en cuyo caso será de 5 o 10 minutos. La duración de la rutina de medición, controlada por un timer, se define cuando se determina qué tipo de medición se va a llevar a cabo.

Cuando finalice la medición, el microcontrolador se encargará de calcular los parámetros necesarios. Mientras los calcula, en la pantalla se representará:

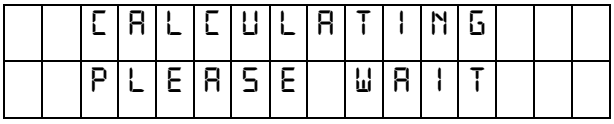
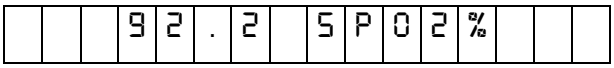


Figura 23: Mensaje de cálculo

Y una vez terminado el cálculo, se representará el resultado dependiendo de si se ha seleccionado medición del SpO2 o del estrés:



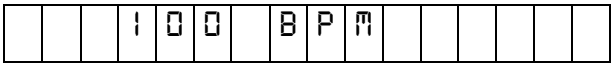


Figura 24: Mensaje de resultados de SpO2

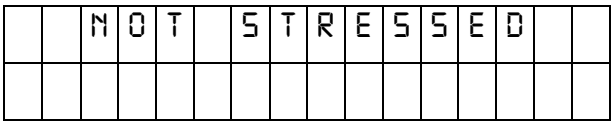


Figura 25: Mensaje de resultado de estrés

3.3.1.2 Proceso de medición del nivel de batería

El microcontrolador MSP430F5529 utiliza su interfaz I2C para comunicarse con el módulo BQ27200, encargado de monitorizar la batería que alimenta el dispositivo. En la Figura 26 se muestra como es esta comunicación (este diagrama de bloques representa el funcionamiento interno del estado inmediatamente posterior al encendido del dispositivo, en el diagrama de flujo de la pantalla).

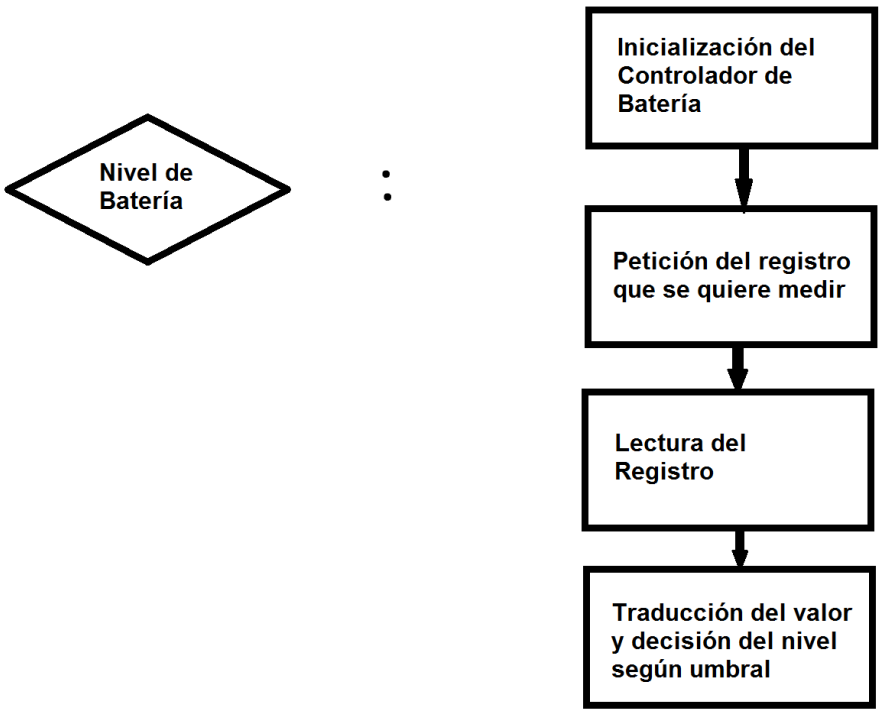


Figura 26: Diagrama de flujo de la batería

La lectura del nivel de la batería será el primer proceso llevado a cabo tras el encendido del dispositivo para ver si es posible un nuevo experimento. Para realizar esta elección, se plantea adquirir el valor del registro RSOC [25], que proporciona el valor relativo del estado de carga en tanto por ciento (al especificar relativo se refiere a respecto al último valor registrado de batería descargada). Este es el método que se escoge aunque igualmente válido habría sido otro en el que se determinara mediante el tiempo-para-vacío, el nivel de corriente u otro de los registros disponibles en la memoria EPROM del módulo BQ27200.

Tras inicializar la interfaz I2C del MSP430F5529, el microcontrolador enviará la petición de lectura del registro especificado al módulo de gestión de la batería. La respuesta de este será un byte de información en el que se especifica este valor.

El valor que esté almacenado en el registro RSOC será ponderado para, a partir de él, determinar si es posible o no llevar a cabo una nueva medición. Para ello, se debe estudiar *a posteriori* cuánto porcentaje de batería consume una medición (de estrés, ya que es la que mayor consumo exige). De este modo, el método de decisión consiste en comparar el valor de la batería con este umbral mínimo y, si es mayor, se podrá llevar a cabo la medición. En caso contrario, se notifica por pantalla de que no es posible continuar hasta que se someta la batería a una nueva carga.

3.3.1.3 Proceso de envío de mensajes mediante Bluetooth

Como se explicó en el apartado de diseño hardware, para la comunicación Bluetooth se eligió el módulo LMX9838 que integra todos los componentes necesarios para una comunicación de este tipo. La conexión con su homólogo, al que enviará la información, la realiza automáticamente este módulo, sin necesidad de que el microcontrolador actúe para configurar o supervisar. La elección de este módulo ha conllevado una simple y cómoda programación ya que todos los datos que este reciba por la interfaz serie serán automáticamente empaquetados en la torre de protocolos Bluetooth y enviados al destinatario.

En la siguiente figura se detalla el funcionamiento de este bloque:

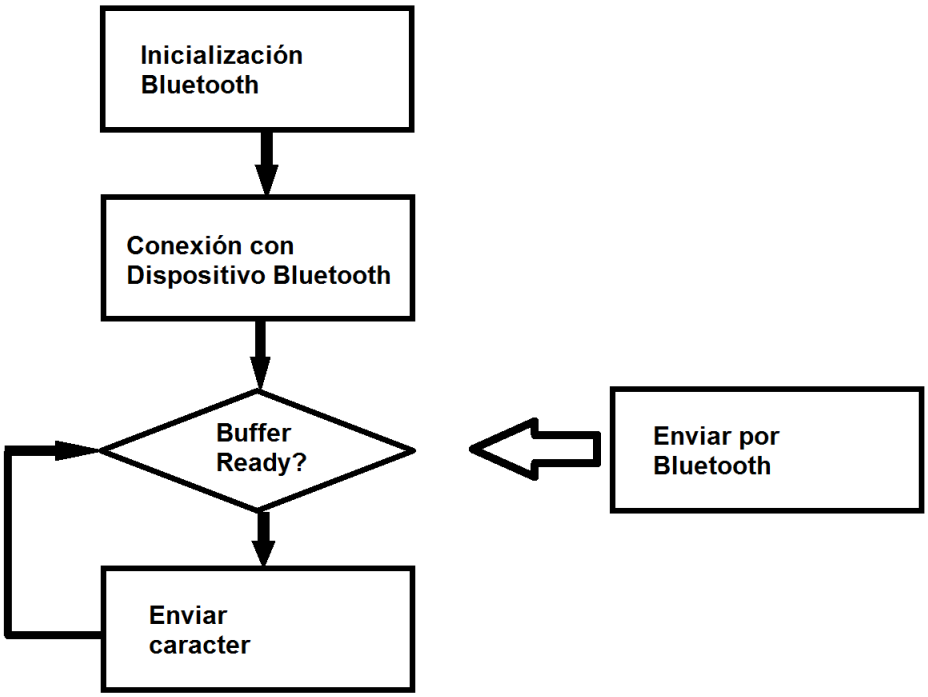


Figura 27: Diagrama de flujo del Bluetooth

En primer lugar, se inicializa la interfaz UART del dispositivo para llevar a cabo la comunicación por Bluetooth. El siguiente paso corresponde al turno de conexión del módulo Bluetooth con el dispositivo al que se va a enviar la información. Este paso no tiene correspondencia en este trabajo dado que esta conexión la lleva a cabo automáticamente el módulo LMX9838. A continuación, el módulo (o la interfaz UART) se queda disponible para un posible envío. Cuando desde el programa principal se invoque la función para enviar datos, se almacenarán en el buffer interno del módulo y este lo transmitirá cuando esté preparado, poniéndose a continuación a la espera de un próximo envío.

La cuestión por resolver en este bloque es la velocidad de transmisión y la información transmitida en cada envío. La tasa debe elegirse experimentalmente, según los resultados que dé uno u otro tipo (en general, se optará por la tasa más alta). Los datos enviados en cada transmisión se plantean en este trabajo del tamaño del registro del ADC del AFE (64 bytes cada registro). Esto es así para que en cada envío se pueda informar íntegramente del nivel DC del LED rojo, AC, del DC del LED infrarrojo y el AC. No obstante, aunque esté así configurado, en este trabajo no se transmite esta información mediante Bluetooth dado que no se dispone aún de una aplicación en el otro extremo para interpretar o representar estos datos y sólo contribuiría a un mayor consumo de la batería.

3.3.1.4 Proceso de envío de datos mediante USB

La comunicación vía puerto USB es el otro tipo de comunicación que se ha diseñado para el dispositivo. El microcontrolador, como ya se ha mencionado, integra esta interfaz, por lo que no precisa más diseño que el hardware.

El diagrama de flujo diseñado para este bloque se muestra en la Figura 25:

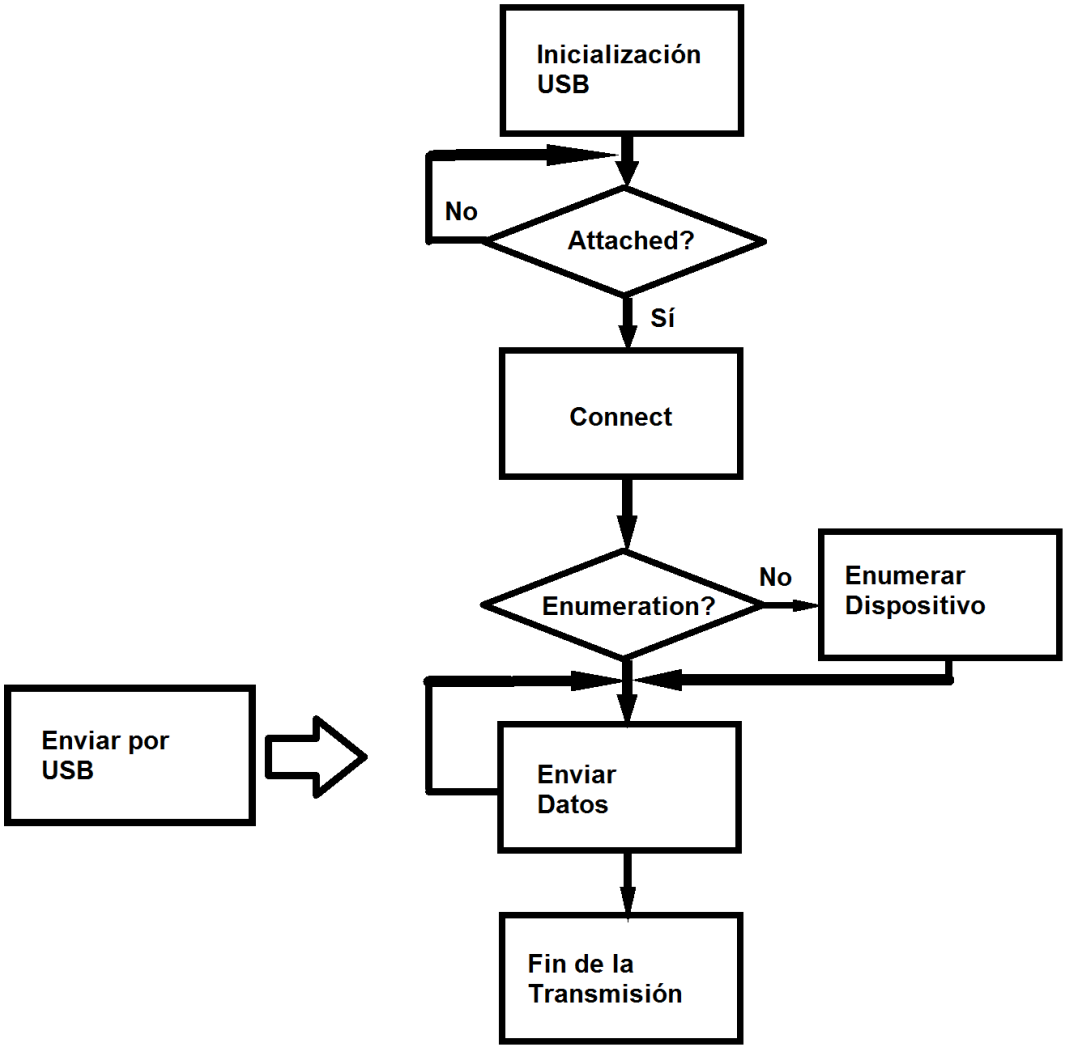


Figura 28: Diagrama de flujo del USB

Este diagrama es más complejo que el de la comunicación Bluetooth ya que no se trata de un módulo autónomo. En este caso, a parte de la inicialización de la interfaz, es necesario un control para comprobar si el dispositivo ha sido conectado en el otro extremo y, en caso afirmativo, comenzar la enumeración. Tras esto, el envío de datos sucede con normalidad.

La peculiaridad de este bloque, es que la información puede viajar en los dos sentidos. Concretamente, esto quiere decir que el integrado AFE4490 ofrece la posibilidad de modificar el valor de algunos de sus parámetros internos (valores de resistencias y condensadores en el amplificador de transimpedancia, por ejemplo) durante la ejecución. Esto se lleva a cabo mediante la interfaz SPI y los hilos de propósito general que comunica el AFE con el microcontrolador, quien puede retrasmitir la información que le envíe directamente el ordenador al que está conectado. Esta funcionalidad, en cualquier caso, es una herramienta más que se utiliza durante la fase de programación para depurar el programa pero nunca una opción que se deja abierta en el producto final, por lo que se excluye del diagrama de flujo que contempla sólo lo que sería la ejecución normal del programa.

3.3.1.5 Proceso de almacenamiento y lectura de datos en memoria SD

Para el diseño de la comunicación con la tarjeta de memoria microSD incorporada en el dispositivo se ha tenido en cuenta la información recopilada sobre este tipo de memorias y su programación en [45] y [47]. Las memorias SD implementan tres protocolos distintos para la lectura y escritura de sus datos: el modo SPI, el modo SD de 1 bit y el modo SD de 4 bits. En este trabajo se ha decidido usar el protocolo SPI ya que, a pesar de ser más lento que los otros dos modos, es un protocolo muy extendido con una eficiente implementación en la familia de protocolos MSP430, permitiendo una completa funcionalidad de la tarjeta SD (esto conlleva, además, que la tarjeta insertada debe implementar internamente este protocolo).

Dado que una de las características de esta tecnología es que la tarjeta puede ser extraída e insertada en otro dispositivo distinto, se planteó la posibilidad de almacenar los datos registrados por el sensor de pulso oximetría de tal modo que posteriormente, desde un ordenador, se pudiese acceder a toda esta grabación. Es por esto que en este trabajo se ha decidido emplear el sistema de ficheros FAT32 para la localización estandarizada de ficheros.

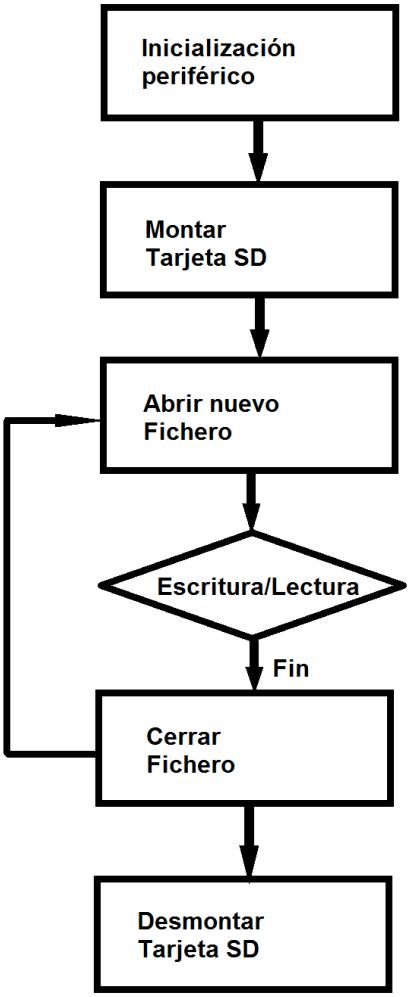


Figura 29: Diagrama de flujo de la memoria

Tras inicializar la interfaz y montar la tarjeta microSD insertada, el programa principal crea un fichero en esta tarjeta en el que almacenará a continuación los datos que se vayan leyendo del módulo AFE. No se lleva a cabo una discriminación de los datos previa ni una reordenación, de tal modo que la lectura posterior de este fichero sea una representación fiel de las lecturas que se han hecho del sensor.

Como se ha comentado, este almacenamiento se aprovecha para poder pasar la información a otro dispositivo de manera directa. Sin embargo, el propósito principal de este almacenamiento es poder detectar luego los picos de la señal de PPG y calcular los parámetros, dado que la memoria RAM del microcontrolador no es suficiente. Por ello, durante el estado de "Medición", el microcontrolador estará continuamente invocando la función para almacenar bytes de información en la memoria sobre FAT32 y, durante el estado de "Cálculo", leerá esos datos y escribirá los resultados. Estos resultados no se plantean almacenar en este diseño en la tarjeta

aunque de querer hacerse valdría con invocar nuevamente la función de escribir en la memoria o, si se prefiere, crear un nuevo fichero. Cabe resaltar que es importante cerrar el fichero una vez se ha completado la medición, para poder ser leído correctamente en otro dispositivo, así como desmontar la tarjeta para evitar posibles errores.

3.3.1.6 Proceso de manejo del analog front end

El módulo AFE4490 trabajará de forma autónoma controlando los diodos LED rojo e infrarrojo, midiendo la señal recibida en el fotodetector y realizando la conversión analógico-digital. Cada vez que se complete una conversión, se indica como una interrupción por el pin ADC_RDY en Anexo I, Plano 1. En este momento, el microcontrolador accederá a los registros donde se ha almacenado la información relativa a la señal de PPG y habilitará una vez más la conversión. Este acceso se realiza mediante la interfaz SPI, que es controlada en el programa principal. El proceso relativo a este bloque es el que se muestra a continuación:

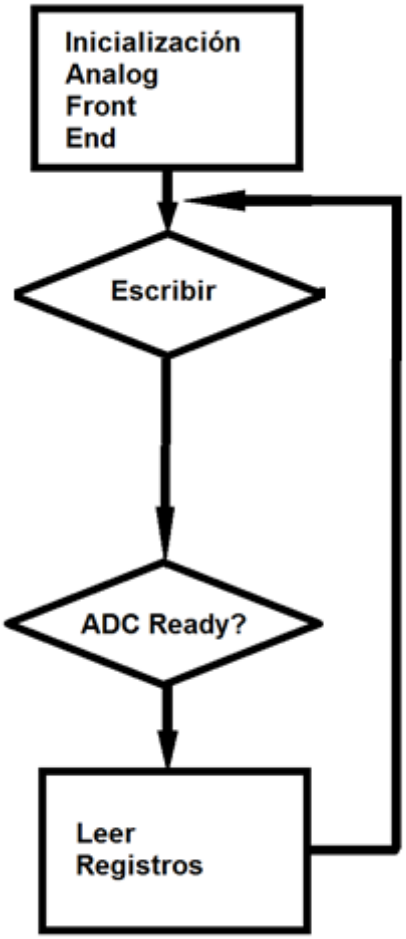


Figura 30: Diagrama de flujo del analog front end

El módulo AFE trabaja autónomamente transmitiendo los pulsos y leyendo la señal recibida en el fotodetector, con su posterior amplificación, filtrado y conversión digital. La comunicación que mantiene con el microcontrolador es para enviar la información almacenada en los registros de la memoria interna así como para modificar el valor de algunos de estos en sentido inverso. En este diagrama no se incluyen los estados relativos a la programación del valor de los registros del AFE4490. Como se ha comentado anteriormente, estos se pueden modificar mediante esta interfaz SPI a través de un ordenador que modifique los valores. Esto es así para determinar los valores óptimos que permitan la mejor medición. Esto se lleva a cabo como fase de depurado y, una vez determinados los valores de los registros, estos se guardan definitivamente y no vuelve a usarse esta funcionalidad.

3.3.1.7 Diagrama de flujo completo del dispositivo

El diagrama de flujo que representa el funcionamiento del dispositivo es similar al representado en la Figura 17.

Al inicializar el dispositivo, se lleva a cabo la primera inicialización de periféricos, en la que se configura la comunicación con la pantalla y con el controlador de batería. Esto es para invocar a la función que mide el nivel de batería y poder indicar por pantalla si es suficiente como para llevar a cabo una nueva medición. Si es correcto, se procede con la segunda inicialización, que corresponde a la configuración fundamental y común a ambos tiempos de mediciones (estrés y SpO2): módulo AFE, memoria SD, botones, timers y relojes.

La primera opción que se da a elegir en el menú es la de selección del tipo de medición: estrés o SpO2. Con los botones laterales se selecciona uno u otro y con el botón central se confirma la elección. Cuando se pulsa este último es cuando se determina en el programa principal que el tipo elegido es el tipo de medición que se debe llevar a cabo y se configura la duración del timer para que el tiempo de medida sea de 20 segundos, para SpO2, o de 5 minutos, para estrés.

La segunda opción que se da a elegir en el menú es la selección de la vía de comunicación que se prefiere para transmitir la información. Esta puede ser USB, Bluetooth, ambas simultáneamente o ninguna de ellas, en cuyo caso el resultado sólo se notifica por pantalla y no se podrá visualizar la señal de PPG. Como en el caso anterior, con los botones se selecciona una opción y, al confirmar, se procede con la tercera inicialización, es decir, la de la o las interfaces que se hayan escogido.

Una vez se ha completado esta inicialización, el dispositivo comienza a leer la señal medida por el sensor. Los registros en cuestión son del 42 al 47 en el AFE4490, que corresponden cada uno de ellos a lectura del LED rojo, lectura de ambiente, lectura de LED infrarrojo, lectura del

ambiente, lectura del ambiente mientras el pulso de LED rojo (DC) y lectura del ambiente mientras el pulso del LED infrarrojo. Estos valores se almacenan en un buffer que se prepara para ser enviado a la tarjeta de memoria. Posteriormente, ese buffer se configura para transmitirlo por USB en caso de haber seleccionado la opción de transmitir por esta vía. Además, en este proceso de medición es donde se lleva a cabo el control de la interfaz USB, para la conexión, desconexión y enumeración del dispositivo.

Una vez expire el timer, se detendrá la medición del AFE se comenzará el cálculo del resultado, dependiendo de si se ha elegido SpO2 o estrés. En el primer caso, se implementa la siguiente función, [48] y [49], para obtener el valor de la saturación en sangre:

$$SpO2 = K\frac{{RED_{RMS}}/{RED_{MEAN}}}{{IR_{RMS}}/{IR_{MEAN}}}$$

El valor medio y RMS se calcula directamente sobre los valores de los registros leídos. El parámetro K es una constante cuyo valor ronda las 98.56 unidades pero que debe ser calibrado para cada dispositivo.

En el caso de tener que medir el estado de estrés, se ejecuta un programa que replica el algoritmo utilizado en [1].

3.3.2 Codificación

Una vez diseñada la estructura software, se pasó a su codificación tanto a nivel alto como a ensamblador. Como lenguaje de alto nivel se ha usado C.

3.3.2.1 Programación de la pantalla

La librería para trabajar con una pantalla LCD 16x2 de tecnología HD44780 se ha tomado de la fuente [42] y se ha modificado para adaptarlo a los requerimientos de este trabajo y al dispositivo físico diseñado. Está compuesto por la librería denominada lcd16.h, cuyas funciones están codificadas en lcd16.c. Este código se encuentra en el Anexo III, Código del dispositivo.

- void lcdinit(void): configura los puertos GPIO del microcontrolador que se han conectado a la pantalla LCD 16x2 del dispositivo. Esta función es llamada una única vez al comienzo del programa.
- void integerToLcd(int integer): imprime en la pantalla un número (decenas y unidades) almacenado en la variable *integer*.
- void integerToLcd3(int integer): tiene la misma funcionalidad que la función anterior con la diferencia de que imprime las centenas, decenas y unidades de ese número.

- void lcdData(unsigned char l): imprime en la pantalla el carácter *s*.
- void prints(char *s): orden llamada desde la función principal para imprimir una cadena de caracteres.
- void gotoXy(unsigned char x,unsigned char y): función que permite seleccionar la celda de la pantalla en la que se desea imprimir un carácter o desde la que se quiere empezar a imprimir una cadena.
- void waitlcd(unsigned int x): espera el número de milisegundos especificado en la variable *x*.

3.3.2.2 Programación de la medición del nivel de batería

La librería necesaria para la comunicación por I2C se ha tomado de la fuente [43] y se ha modificado para adaptarlo a los requerimientos de este trabajo y al dispositivo físico diseñado. Consiste en una única función cuyos ficheros de cabecera y de código son, respectivamente, fuel_gauge.h y fuel_gauge.c. Este código se encuentra en el Anexo III, Código del dispositivo.

- int readBattery(void): función que lee el registro RSOC y determina si se puede llevar a cabo o no una nueva medición.

3.3.2.3 Programación del módulo Bluetooth

La librería necesaria para la comunicación por serie UART se ha tomado de la fuente [43], como en el caso anterior, y se ha modificado para adaptarlo a los requerimientos de este trabajo y al dispositivo físico diseñado. Está compuesta por dos funciones cuyos ficheros de cabecera y de código son, respectivamente, bluetooth.h y bluetooth.c. Este código se encuentra en el Anexo III, Código del dispositivo.

- void initBluetooth(): esta función es llamada una única vez o ninguna en la ejecución del programa principal cuando se selecciona el método de transmisión de datos para inicializar la interfaz UART.
- void sendBluetooth(unsigned char *s): función que envía al módulo LMX9838 un byte de información.

3.3.2.4 Programación de la comunicación USB

El código fuente empleado para la inicialización, enumeración y transmisión de información a través de esta interfaz se ha tomado de la fuente [44] y no han sido necesarias modificaciones para implementarlo dentro de este trabajo. Es por este motivo que no se incluye en el Anexo III. Tan sólo se comenta cómo se trabaja con estas librerías en el programa principal. Las funciones que se han utilizado son las siguientes:

- BYTE USB_init(void): inicializa la interfaz física USB.
- BYTE USB_setEnabledEvents(WORD events): habilita o deshabilita los eventos USB indicados como parámetros.
- BYTE USB_connectionInfo(void): indica el estado de la conexión USB.
- BYTE USB_handleVbusOnEvent(void): si se ejecuta esta función, indica que se ha aplicado una tensión válida al pin VBUS, por lo que mantiene despierto al microcontrolador.
- BYTE cdcSendDataInBackground(BYTE* dataBuf, WORD size, BYTE intfNum, ULONG ulTimeout): envía información por USB en segundo plano.
- BYTE USB_connectionState(void): devuelve el estado del USB (no confundir con connectionInfo; uno se refiere al estado físico y este al estado en un diagrama de estados).
- WORD cdcReceiveDataInBuffer(BYTE *, WORD, BYTE): recibe la información que se envía desde el otro extremo para configurar el analog front end.

3.3.2.5 Programación del manejo de la memoria SD

Las librerías utilizadas para el almacenamiento y lectura de datos de la tarjeta de memoria microSD se han tomado de la fuente [46] y se ha modificado para adaptarlo a los requerimientos de este trabajo y al dispositivo físico diseñado. Consiste de ocho ficheros, cinco de cabecera (ff.h, ffconf.h, HAL_SDCard.h, integer.h y mmc.h) y tres de código fuente (ff.c, HAL_SDCard.c y mmc.c), de los cuáles sólo ha sido necesario modificar HAL_SDCard.h cuyo código se encuentra en el Anexo III, Código del dispositivo. Las funciones principales usadas por el programa principal son:

- void fat_init(void): inicializa la tarjeta SD.
- FRESULT f_mount(BYTE, FATFS*): monta o desmonta el dispositivo lógico.
- FRESULT f_open(FIL*, const TCHAR*, BYTE): abre o crea un fichero.
- FRESULT f_read(FIL*, void*, UINT, UINT*): lee datos de un fichero.
- FRESULT f_write(FIL*, const void*, UINT, UINT*): escribe datos en un fichero.
- FRESULT f_close(FIL*): cierra el fichero que se encuentra abierto.

3.3.2.6 Programación del analog front end

El código fuente empleado para la inicialización, enumeración y transmisión de información a través de esta interfaz se ha tomado de la fuente [44]. Consta de un fichero de cabecera (AFE44x0.h) y uno con el código fuente (AFE44x0.c), de los cuáles sólo ha sido necesaria la modificación del fichero de cabecera, que se incluye en el Anexo III, Código del dispositivo. A continuación se detalla cómo se trabaja con estas librerías en el programa principal. Las funciones que se han utilizado son las siguientes:

- void AFE44xx_PowerOn_Init(void): inicialización del módulo AFE.

- void Disable_AFE44xx_DRDY_Interrupt(void): desactiva la interrupción que indica que la conversión analógico-digital de la medición se ha completado.
- unsigned long AFE44xx_Reg_Read(unsigned char Reg_address): lee un byte de información del módulo AFE4490. La dirección del registro que se quiere leer se pasa como parámetro y este será, por lo general, el que se corresponda con el valor convertido del pulso rojo, infrarrojo y del valor ambiente entre estos.
- void AFE44xx_Reg_Write(unsigned char Reg_address, unsigned char Reg_data): almacena en el registro indicado el byte pasado como parámetro. Esta función sólo se utiliza en la fase de programación para definir los parámetros del módulo AFE4490.

3.3.2.7 Programación conjunta del dispositivo

Para completar el funcionamiento diseñado para este dispositivo ha sido necesaria, además, la creación de dos fichero de cabecera y dos ficheros con código fuente.

Los ficheros button.h y button.c se encargan de configurar los tres pines del microcontrolador conectados a los botones (SE, NO, y OK en Anexo I, Plano 8) como entradas digitales y de leer sus correspondientes valores. Consisten de dos funciones:

- void Init_Buttons(void): inicializa los botones como entradas digitales.
- int readButton(int button): lee el valor (alto o bajo) del botón que se indica como parámetro. El valor 1 corresponde al botón NO, el 2 al SE y el 3 al OK.

Por otro lado, los ficheros math_functions.h y math_functions.c codifican el algoritmo de [1] y la detección de picos sistólicos en la señal de PPG almacenada en la memoria. Contiene las siguientes funciones:

- unsigned long *peakDetect(FIL *fp, UINT btr): detecta los picos de la señal almacenada en el fichero que se pasa por parámetro.
- unsigned long mean(unsigned long *s, UINT btr): calcula la media temporal de los valores de los picos.
- unsigned long std(unsigned long *s, UINT btr): calcula la desviación estándar.
- unsigned long rmssd(unsigned long *s, UINT btr): calcula el valor de la RMSSD.
- int pnn50(unsigned long *s, UINT btr): calcula el porcentaje de intervalos consecutivos que difieren en más de 50 ms.
- unsigned long sd1(unsigned long *s, UINT btr): calcula el parámetro SD1.
- unsigned long sd2(unsigned long *s, UINT btr): calcula el parámetro SD2.
- unsigned long Apen(unsigned long r, unsigned long *s, UINT btr): calcula la entropía de los valores.

Los cuatro ficheros se incluyen en el Anexo III, código del dispositivo.

Tras esto, se procedió a codificar el programa principal, nombrado main.c en el Anexo III, código del dispositivo, y que implementa el diagrama de flujo mencionado en el punto anterior. Dentro de este fichero se han creado las funciones que engloban el cálculo de la saturación de la sangre y del nivel de estrés:

- void measureSpO2(void): calcula el valor del SpO2 así como del pulso y lo representa por pantalla.
- void measureStress(void): detecta los picos sistólicos y ejecuta sobre ellos el algoritmo de detección de estrés, representando el resultado por pantalla.

Durante esta fase de programación, se detectaron algunos conflictos de cara a la integración del programa en el dispositivo hardware diseñado. Para su correcto funcionamiento, se procedió a su rectificación. Estos son, por ejemplo, la modificación de la asignación de un puerto del microcontrolador por no disponer de entrada de interrupciones, la incorporación de resistencias pull-up el los hilos que comunican al microcontrolador con la memoria y la configuración del puerto PUR (ver Anexo I, Plano 8) para permitir la correcta enumeración del dispositivo.

Capítulo 4. Resultados

4.1 Resultados del dispositivo

El resultado de este trabajo es el diseño de un prototipo para un dispositivo electrónico de detección del estrés. Este diseño se ha logrado llevar a cabo de acuerdo a los requisitos y especificaciones que se plantearon al comienzo. El presupuesto estimado para su fabricación es de 66.45€, cantidad que se detalla en el Anexo II, Lista de materiales. En las siguientes dos figuras se muestra una simulación de su aspecto final.

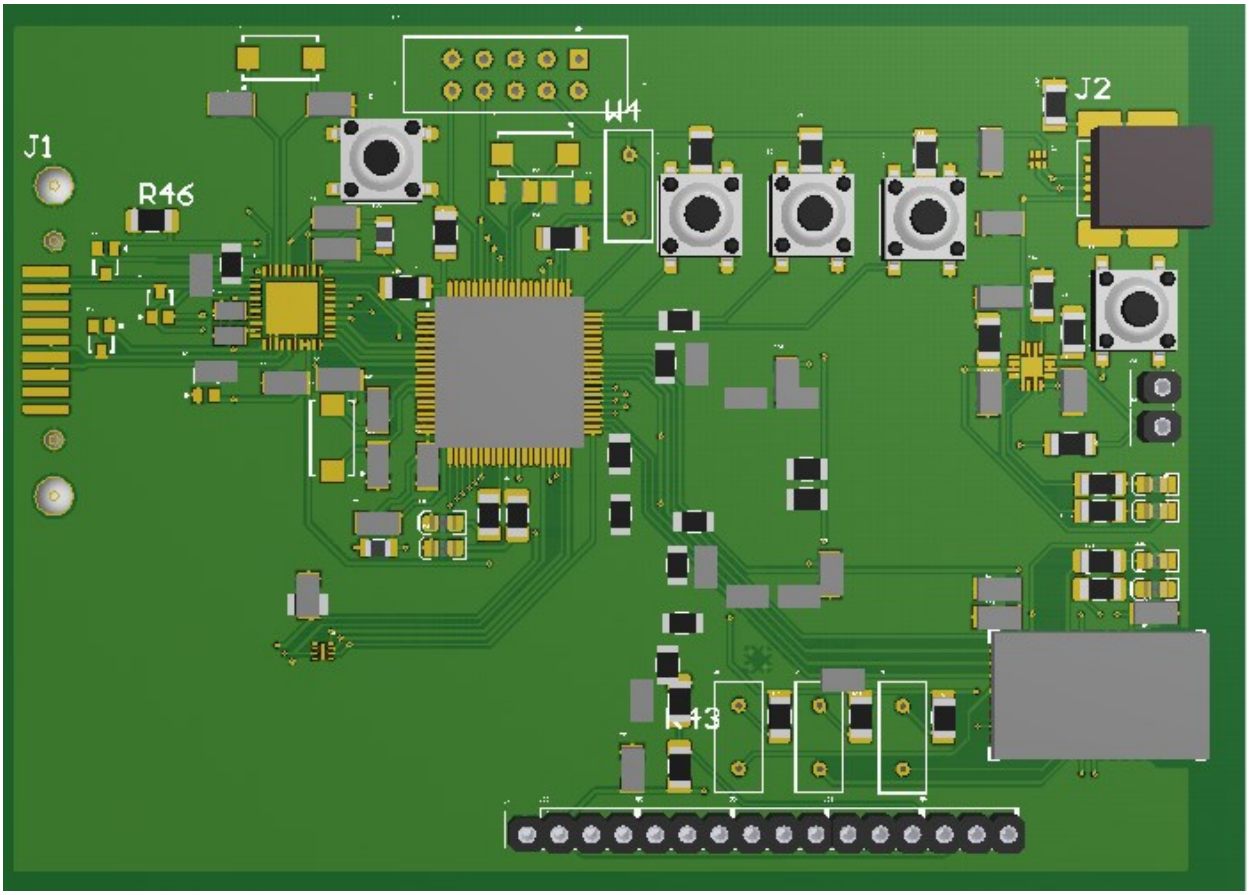


Figura 31: Representación virtual del dispositivo diseñado - Cada superior

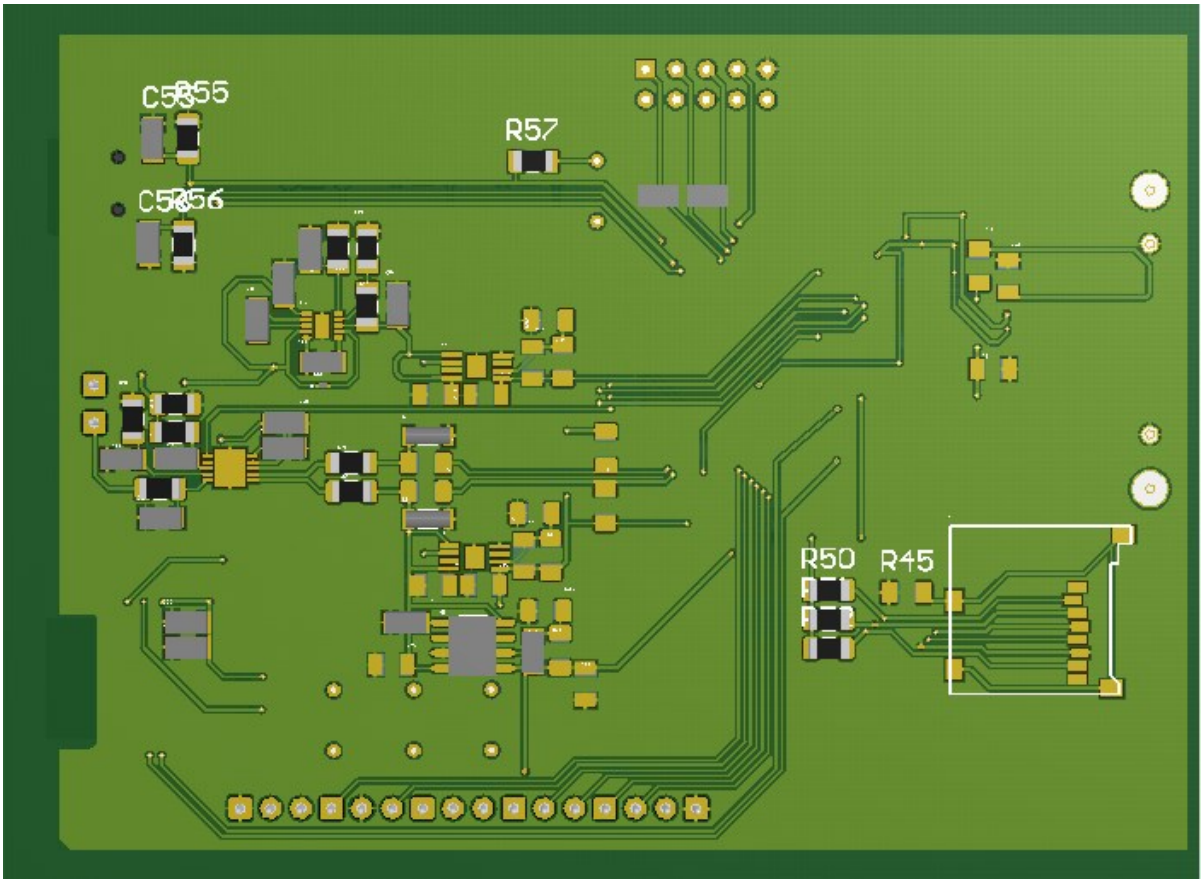


Figura 32: Representación virtual del dispositivo diseñado - Cara posterior

Debido a las dificultades técnicas de fabricación de un diseño ASIC no se ha podido fabricar dicho dispositivo para su prueba. Sin embargo, a continuación se propone una alternativa para su futura implementación que sirva como paso previo a la fabricación del prototipo.

Este diseño alternativo se basa en una estructura modular, esto es, en dividir los bloques diseñados en este trabajo en módulos individuales, de características similares, ya ensamblados. De esta forma, se elimina casi por completo el problema de fabricación, que es el principal impedimento a la hora de llevar a la práctica este diseño. El tamaño, irremediablemente, aumenta, aunque el precio no necesariamente lo hace. El consumo de este diseño modular no será óptimo ya que se trata de una solución adaptada y pierde los beneficios de una específica.

Como módulo principal de este nuevo diseño se propone una de las múltiples placas de evaluación que hay en el mercado, como puede ser la STM32F407Discovery o la MSP-430F5529 Launchpad, que incorporan microprocesadores con capacidad suficiente para el procesamiento que exige el programa desarrollado, así como suficientes puertos que permiten la comunicación con periféricos mediante protocolo SPI, I2C, UART así como entradas y salidas de propósito general.

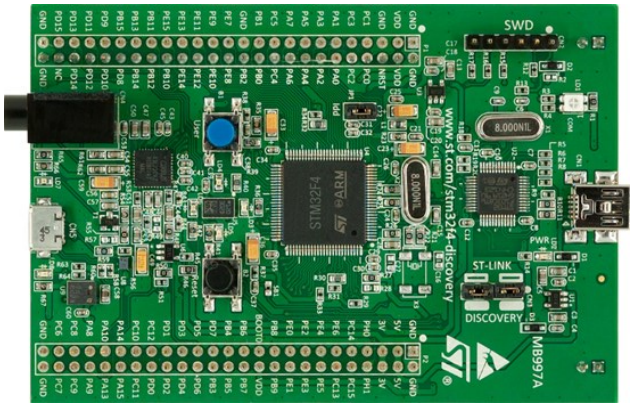


Figura 33: Módulo STM32F4Discovery

Para la comunicación inalámbrica pueden encontrarse módulos como el HC-05, que incorpora todo lo necesario para la comunicación en un mismo componente, siendo necesario tan sólo el conexionado del módulo con la placa de evaluación elegida. De este mismo modo, también se pueden encontrar módulos para conectar tarjetas de memoria SD, necesaria para almacenar la grabación de PPG en caso de no disponer de capacidad suficiente el microcontrolador, y módulos puente que permiten que una de las interfaces UART de la placa de evaluación se comunique con el puerto USB del PC al que esté conectado el dispositivo. Además, en caso de no disponer de suficientes pines para interconectar todos los módulos, se propone otro que traduzca la comunicación con la pantalla al protocolo I2C, reduciendo en gran medida de este modo el uso de estos.

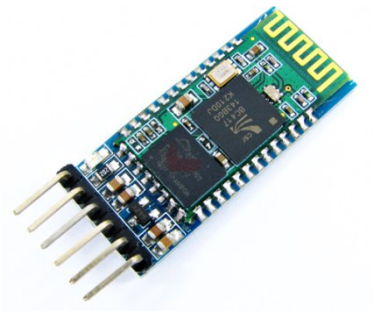


Figura 34: Módulo HC-05

Sin embargo, el AFE no es posible encontrarlo como solución modular. Este bloque quedaría como el único por fabricar. En el Anexo I, Planos del dispositivo, se añaden dos planos (Plano 11 y Plano 12) que corresponden al diseño esquemático y layout de lo que sería el módulo de adquisición de la señal, comunicándose con la placa de evaluación a través de la interfaz SPI.

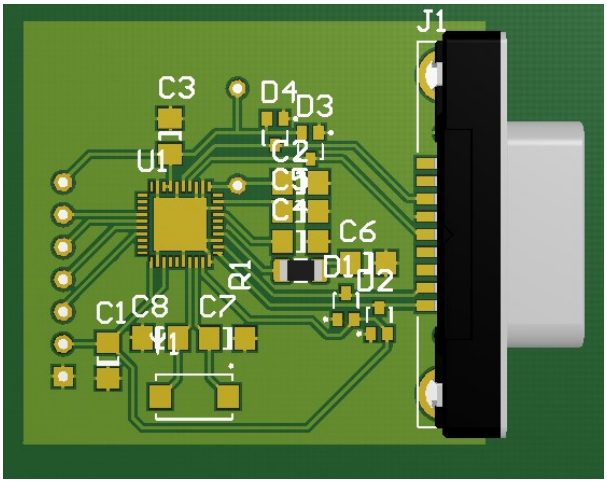


Figura 35: Módulo AFE

4.2 Resultados del programa

El programa codificado cumple los requisitos planteados partiendo del diseño hardware a excepción de parte de la comunicación con los dispositivos inalámbricos. Aunque se tienen librerías para llevar a cabo esta comunicación, no se dispone de un programa que se ejecute en otro dispositivo que se encargue de interpretar la información recibida. Sin embargo, debido a la imposibilidad de fabricación del dispositivo físico, no ha sido posible el depurado del programa final.

En caso de optarse por implementar la solución de manera modular, como se ha descrito anteriormente, el código proporcionado en este trabajo variaría irremediablemente si cambiase el microcontrolador. No obstante, seguirían siendo igual de válidos los diagramas de bloque en los que se describe el funcionamiento de este.

4.3 Resultados del algoritmo de detección del estrés

Del mismo modo descrito en el Capítulo 2, se procedió a someter un conjunto de registros de la base de datos MIMIC II al algoritmo de detección del estrés diseñado en [1].

Sin embargo, se observó que los resultados diferían notablemente cuando se efectuaba la misma prueba sobre la señal de ECG a cuando se hacía sobre la de PPG. En concreto, para la señal de PPG, en todos los casos el diagnóstico era el mismo: estresado. Esto llevó a pensar que se trataba a una falta de muestras, por lo que se procedió a aumentar la ventana de medida de cinco a diez minutos. En la siguiente tabla pueden verse los resultados que se obtuvieron en cada caso para cada uno de los veinte registros.

Tabla 3: Comparación de la detección del estrés mediante la señal de ECG y la de PPG

Grabación	Señal ECG	Señal PPG

3000086_0003(2,4)	No estresado	No estresado
3000086_0006(2,4)	No estresado	Sí estresado
3000531_0001(3,4)	No estresado	No estresado
3000714_0004(6,3)	No estresado	Sí estresado
3000781_0002(2,4)	No estresado	No estresado
3000858_0008(3,4)	Sí estresado	Sí estresado
3000860_0004(3,4)	Sí estresado	Sí estresado
3000860_0007(3,4)	No estresado	Sí estresado
3001158_0005(3,4)	No estresado	Sí estresado
3001203_0007(3,4)	No estresado	Sí estresado
3001203_0010(3,4)	No estresado	Sí estresado
3001912_0005(6,3)	Sí estresado	Sí estresado
3001912_0007(6,3)	Sí estresado	Sí estresado
3001912_0012(6,3)	No estresado	Sí estresado
3002113_0004(6,3)	No estresado	Sí estresado
3002113_0007(6,3)	No estresado	Sí estresado
3002151_0005(6,3)	Sí estresado	Sí estresado
3002762_0004(2,5)	Sí estresado	Sí estresado
3003405_0003(6,3)	No estresado	No estresado
3100524_0004(6,3)	No estresado	No estresado

Como puede observarse, hay nueve discrepancias, lo que sitúa la tasa de error en el 45%. Este valor es inadmisible, por lo que se procedió a analizar con detenimiento qué estaba ocurriendo.

Se analizó visualmente cómo eran señales de ECG y PPG en esta nueva ventana de tiempo así como los parámetros individuales descritos en el Capítulo 2 para identificar dónde erradicaba esta diferencia. Se observó que la diferencia radicaba en la entropía asociada a la señal. Todos los demás parámetros coincidían en valor con los de la señal de ECG. La diferencia de entropía viene causada por las características intrínsecas de la señal de PPG. Como se comentó anteriormente, es una señal muy simple pero que se ve afectada por multitud de fenómenos biológicos así como por el método de medición. La reducción de artefactos por sí sola no logra regular el valor de entropía lo suficiente como para que se asemeje al que corresponde a la señal de ECG. Es por ello por lo que se determinó que el método de evaluación del estrés y su umbral de decisión no eran válidos para la detección del estrés mediante PPG. Se procedió a determinar un nuevo método y un nuevo umbral que replicara el diagnóstico con esta señal.

En la siguiente tabla puede verse la entropía asociada a la señal ECG y PPG por cada registro, así como el diagnóstico de estrés (sin modificar aún el umbral).

Tabla 4: Comparación de entropías entre ECG y PPG

ECG		**PPG**	
Entropía	**Estrés**	**Entropía**	**Estrés**
1,741954	No	2,144337	No
1,520125	No	0,332977	Sí
1,830755	No	2,023637	No
1,351868	No	1,159563	Sí
1,918812	No	1,712511	No
1,187171	Sí	0,469826	Sí
0,293499	Sí	0,487748	Sí
1,322404	No	0,603394	Sí
1,312187	No	0,261913	Sí
1,626028	No	0,298502	Sí
1,733908	No	0,619466	Sí
0,015083	Sí	1,126206	Sí
0,849046	Sí	1,023353	Sí
1,791694	No	0,723309	Sí
1,806309	No	0,26115	Sí
1,623835	No	0,564987	Sí
0,810773	Sí	0,84179	Sí
0,630986	Sí	0,673192	Sí
1,855922	No	2,16059	No
2,400331	No	2,224866	No

Como puede observarse, la entropía asociada a los individuos estresados según la señal de ECG se encuentra dentro de una región acotada. Se propuso por este motivo una nueva detección del estrés dependiendo de si la entropía se encontraba dentro de esta región o no. El diagnóstico sería estresado cuando la entropía estuviese dentro del rango que va de 0.45 a 1.7 y si estuviese fuera de ese entorno, el individuo se detectaría como no estresado. Haciendo esto, el resultado varía como se puede ver en la siguiente tabla:

Tabla 5: Detección del estrés con nuevo método

ECG	PPG
No	No
No	No
No	No
No	Sí
No	No
Sí	Sí
Sí	Sí
No	Sí
No	No
No	No
No	Sí
Sí	Sí
Sí	Sí
No	Sí
No	No
No	Sí
Sí	Sí
Sí	Sí
No	No
No	No

Este nuevo método y este nuevo umbral reducen las discrepancias a 5, por lo que el error se sitúa en un 25%. En concreto, cabe destacar que el error del algoritmo está en diagnosticar sí estresado cuando mediante la señal de ECG se ha detectado lo contrario (en ningún caso se ha diagnosticado no estresado cuando se ha detectado estresado en la señal de ECG).

Este error es el mínimo que se ha logrado conseguir analizando la entropía de las señales. Una vez entrenado el nuevo algoritmo con los veinte registros indicados se procedió a seleccionar diez nuevos para comprobar su respuesta. En la Tabla 6 se muestran los resultados.

Tabla 6: Comparación del test de estrés con nuevos registros

Registro	Estrés ECG	Estrés PPG

3004110_0008(2,4) 10:20	Sí	Sí
3004110_0014(2,4) 20:30	No	Sí
3004110_0017(2,4) 10:20	No	Sí
3006137_0001(2,4) 20:30	No	No
3006137_0004(3,4) 20:30	No	No
3006137_0015(3,4) 20:30	No	No
3200059_0006(6,5) 30:40	Sí	Sí
3200059_0008(6,5) 30:40	Sí	Sí
3200059_0010(6,5) 20:30	Sí	Sí
3200359_0008(4,5) 30:40	Sí	Sí

Como se observa en los resultados, el error es similar al obtenido mientras se entrenaba el algoritmo con los otros registros, habiendo diferido en dos casos el resultado entre ECG y PPG y siendo esta diferencia debida a diagnosticar “sí estresado” cuando en la señal de ECG se detectaba como “no estresado”.

Capítulo 5. Conclusiones

5.1 Conclusiones

El uso de la señal de PPG para la detección del nivel de estrés se ha demostrado admisible con este trabajo en una serie de situaciones. De este modo, la ECG puede ser sustituida por un método más cómodo y simple de usar, reduciendo el coste de la técnica y ampliando el rango de usuarios que puedan hacer uso de esta. Sin embargo, estas ventajas implican que hay algunos inconvenientes. En concreto, los artefactos inherentes al uso de la fotopletismografía reducen el rango de aplicación de este método y dificulta la detección allí donde la electrocardiografía no encontraba problema. Dependerá de cada caso el uso de una técnica u otra aunque mientras se preste especial cuidado en la adquisición de la señal y la postura del individuo sea supina o sentada, este método de PPG es fiable.

Se ha conseguido diseñar un dispositivo que lleve a cabo la detección del estrés, tanto a nivel hardware como software. Los requisitos que se impusieron para este dispositivo se han podido cumplir aunque las dificultades técnicas que implica su fabricación han impedido que este prototipo haya sido llevado a la práctica. Se ha propuesto, no obstante, una alternativa más factible a la hora de fabricar que se considera un buen punto de partida para avanzar en el prototipado de este dispositivo, sobre el que probar el programa así como modificaciones en el algoritmo de detección de estrés.

5.2 Línea de futuros desarrollos

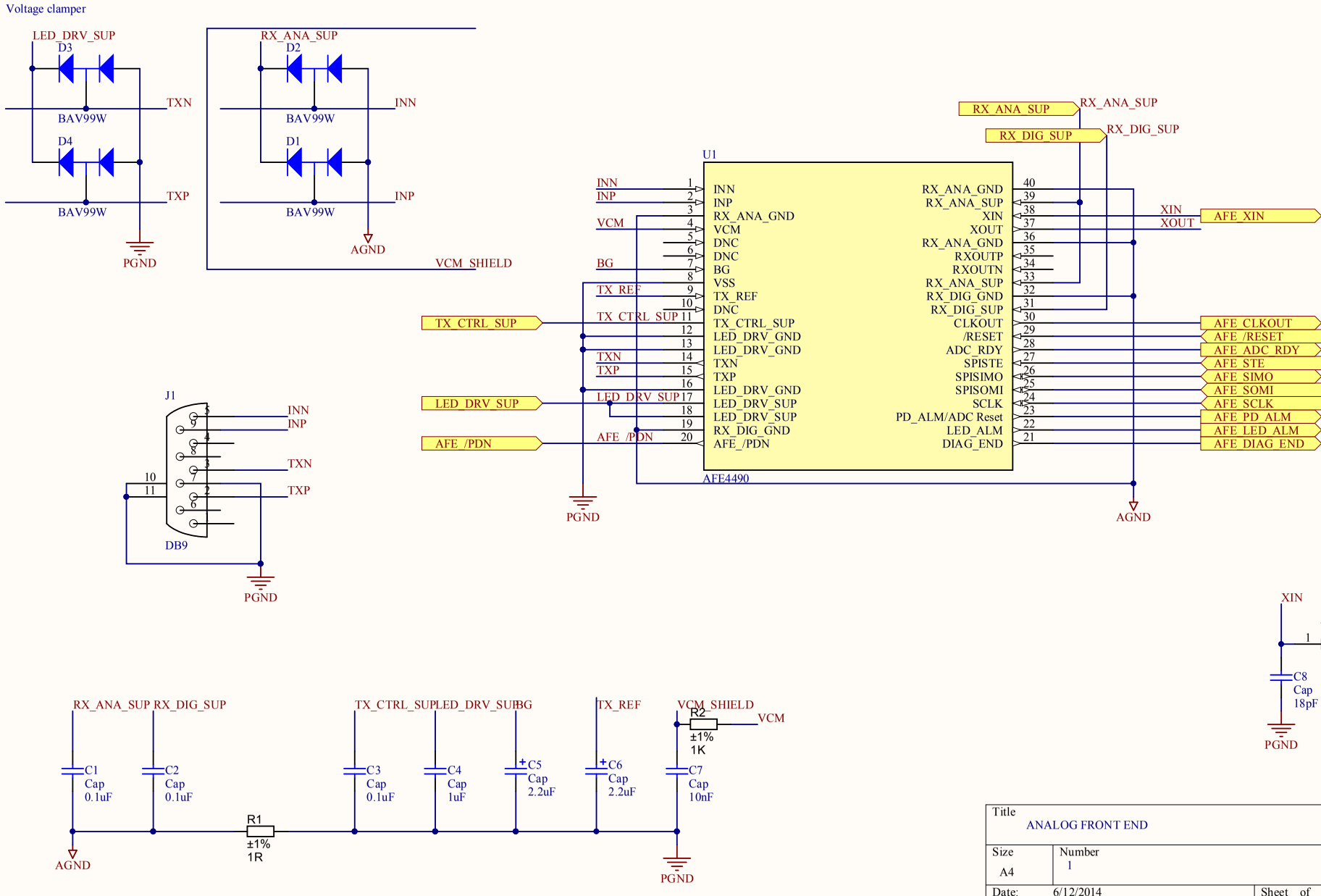
El diseño de un dispositivo abarca un amplio rango de características y de distintas decisiones. El diseño plasmado en este documento responde a las especificaciones propuestas al comienzo de este trabajo aunque no es la única solución a la que se podría haber llegado. Es por esto que el diseño actual es susceptible a ser modificado bien para ser mejorado o para añadir nuevas funcionalidades. Su desarrollo puede tratarse de una línea futura continua aunque en este punto se quiere señalar a otros puntos:

1) Una de las características principales del dispositivo es su autonomía. Internamente podría sufrir diversas modificaciones aunque siempre mantendrían la funcionalidad original que es adquirir la señal de PPG del individuo y diagnosticar el nivel de estrés. Un punto a desarrollar es la interacción de este dispositivo con otros. Dispone de una interfaz inalámbrica, otra alámbrica así como de una pantalla para visualizar directamente los resultados. Se plantea a partir de aquí proceder a desarrollar las

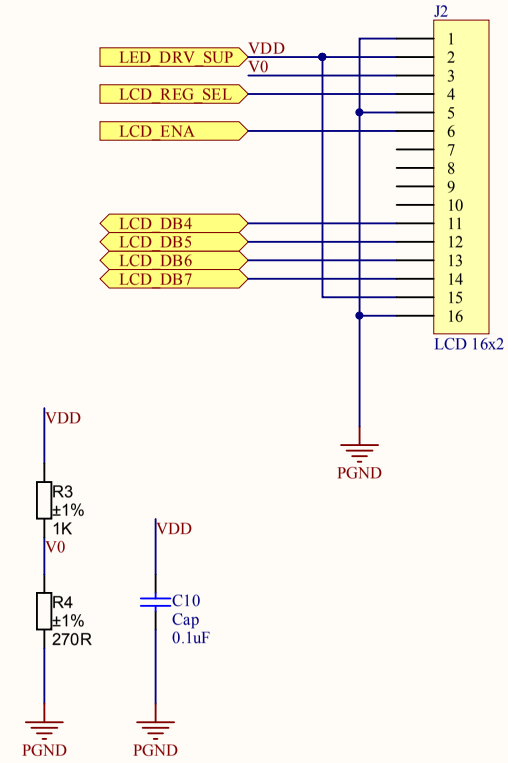
plataformas que se ejecuten en los otros extremos para representar la información y hacer uso de los datos proporcionados.

2) Del punto anterior se desprende otra rama en cuando interoperabilidad. Para poder trabajar con otros dispositivos, necesita un protocolo común de comunicación como puede ser USB o Bluetooth. Pero, además, para poder trabajar con otros dispositivos, necesita cumplir una serie de estándares que certifiquen un lenguaje común que permita que puedan funcionar entre sí. En concreto, los dispositivos de pulso oximetría (dispositivo más cercano al diseñado en este trabajo) deben cumplir la norma ISO13485, que indica entre otras cosas, qué parámetros se deben transmitir y en qué formato.
3) El diagnóstico de estrés se ha probado posible aunque complicado. Este trabajo se ha atenido a replicar en la medida de lo posible la respuesta del algoritmo para la señal de ECG en la señal de PPG. Se hace imprescindible, no obstante, un estudio más en profundidad para definir los parámetros que describen el estrés y poder definir un método y un umbral más preciso.

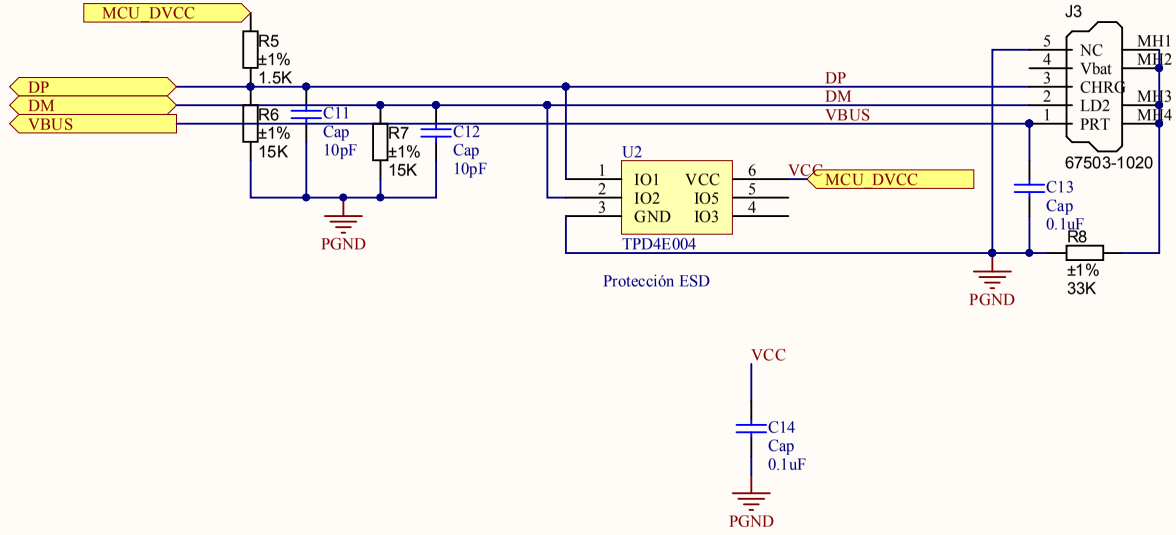
Anexo I. Planos del dispositivo



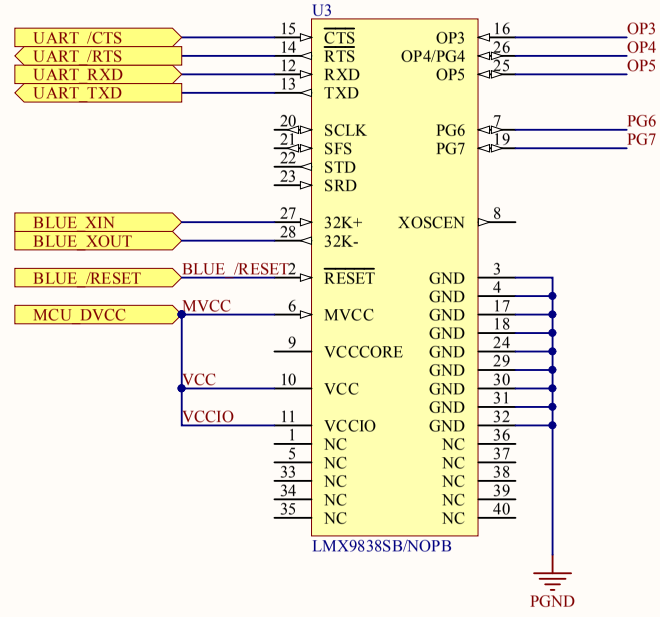
Title: ANALOG FRONT END		
Size: A4	Number: 1	Revision: 6
Date: 6/12/2014	Sheet of	
File: C:\Users\..\AFE.SchDoc	Drawn By: DAVID MARTÍN SÁNCHEZ	

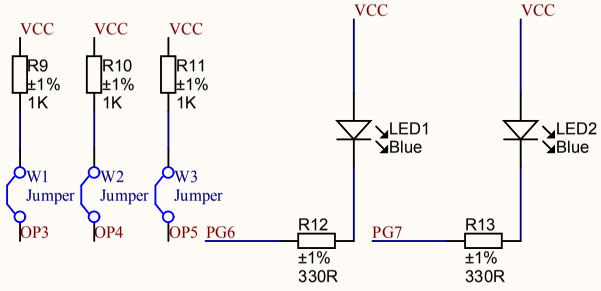


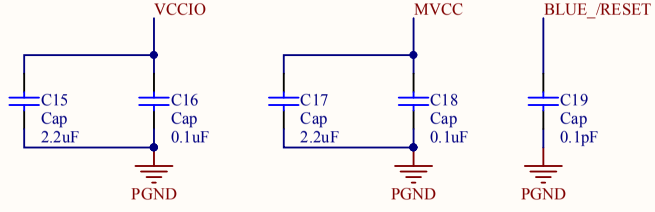
Title: PANTALLA LCD		
Size: A4	Number: 2	Revision: 6
Date: 6/12/2014	Sheet of	
File: C:\Users\..\LCD.SchDoc	Drawn By: DAVID MARTÍN SÁNCHEZ	



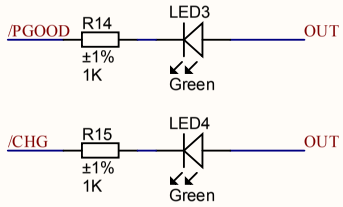
Title: USB		
Size: A4	Number: 3	Revision: 6
Date: 6/12/2014	Sheet of	
File: C:\Users\..\USB.SchDoc	Drawn By: DAVID MARTÍN SÁNCHEZ	



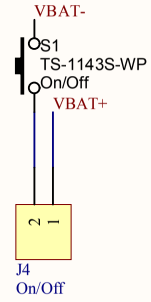




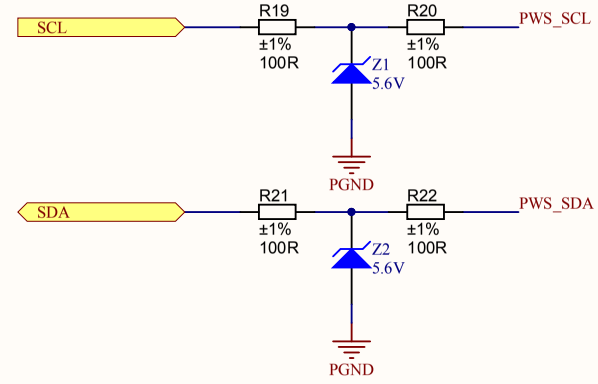
Title: BLUETOOTH		
Size: A4	Number: 4	Revision: 6
Date: 6/12/2014	Sheet of	
File: C:\Users\..\BLUE.SchDoc	Drawn By: DAVID MARTÍN SÁNCHEZ	



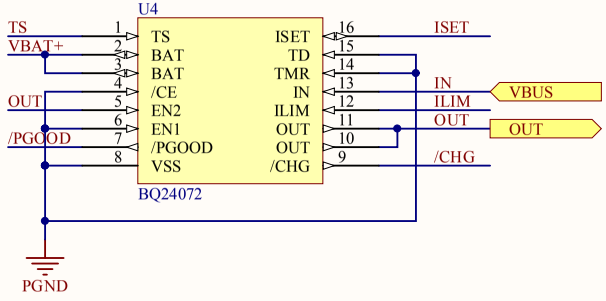
Conector de batería



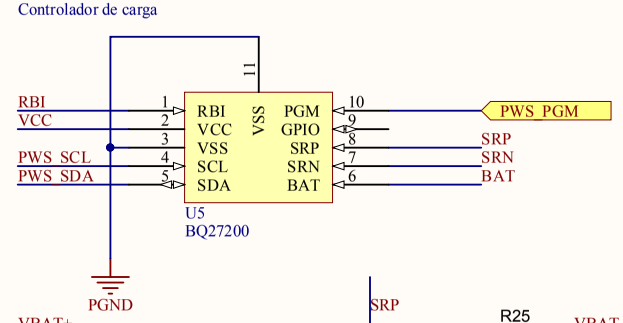
Protección ESD

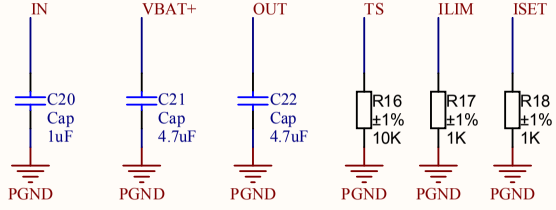


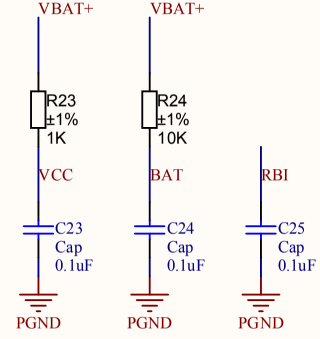
Cargador de batería

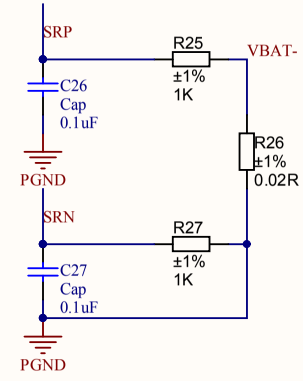


Controlador de carga

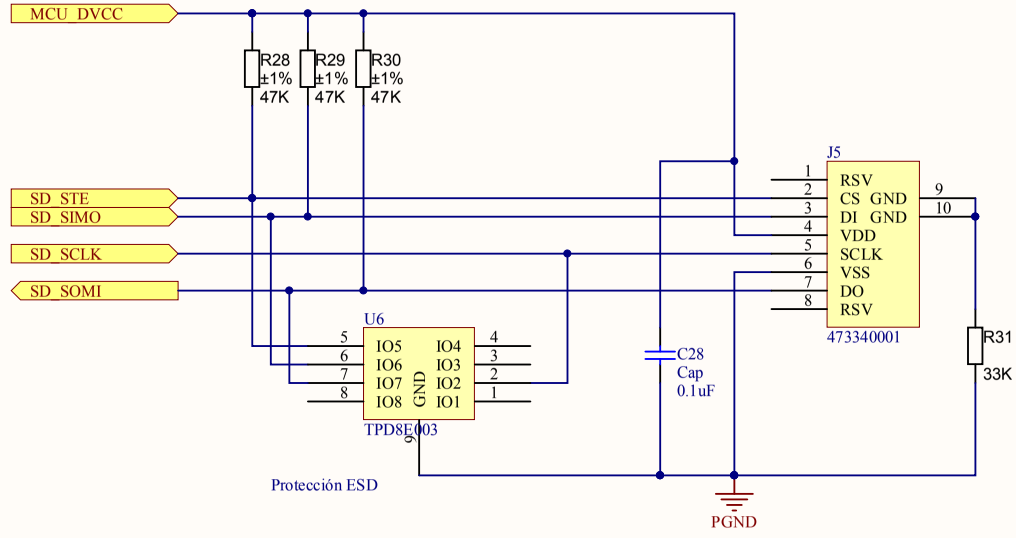




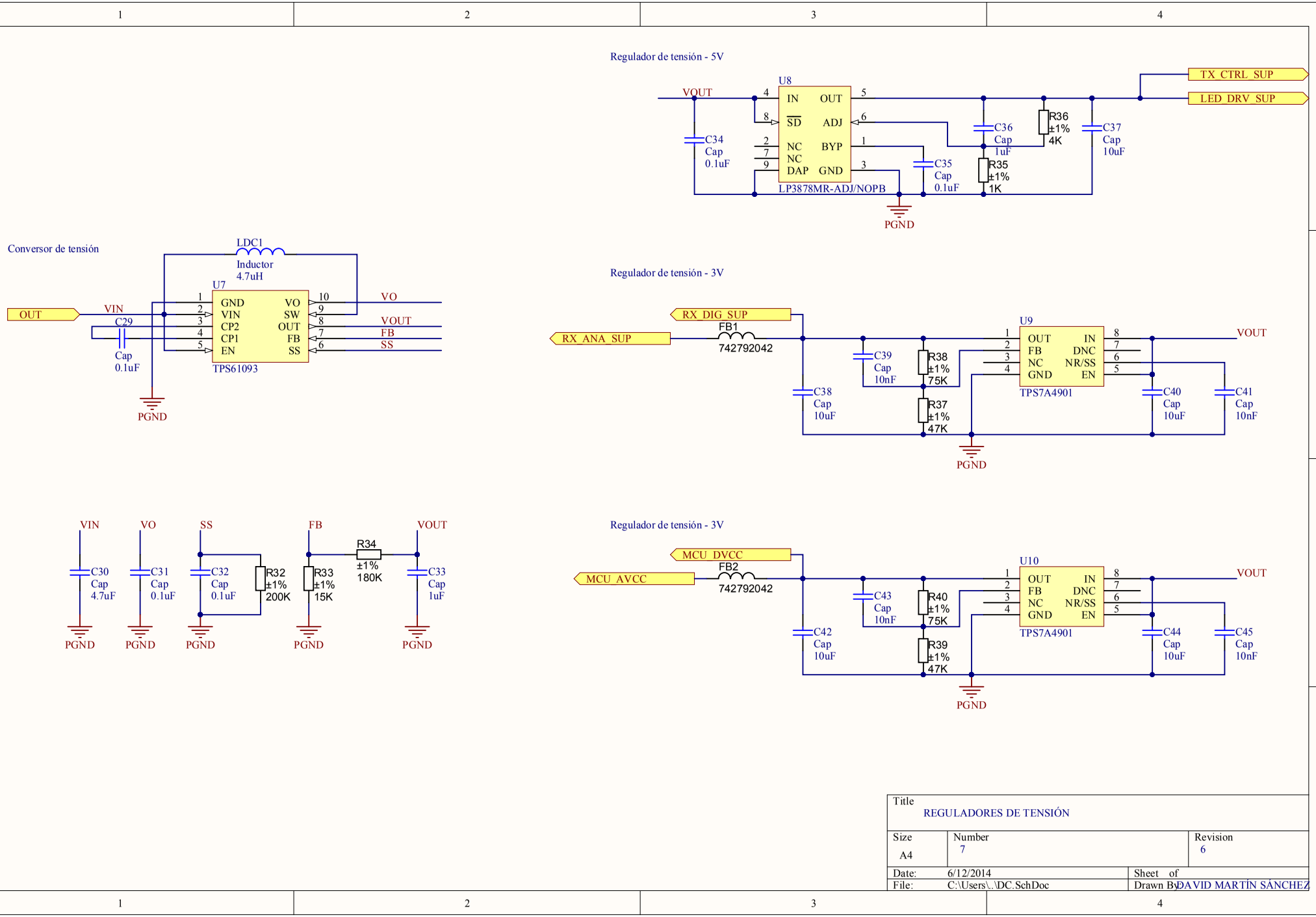


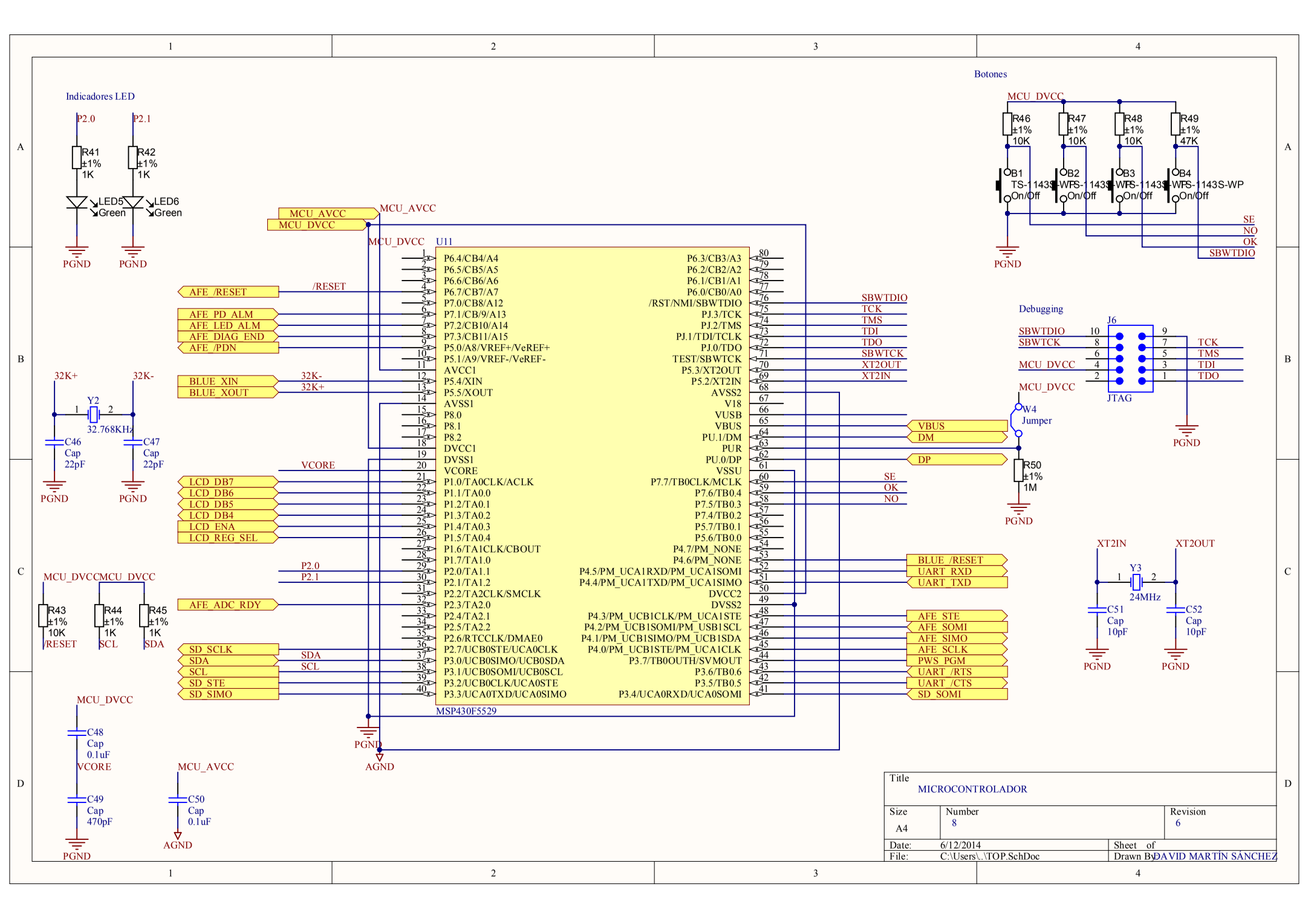


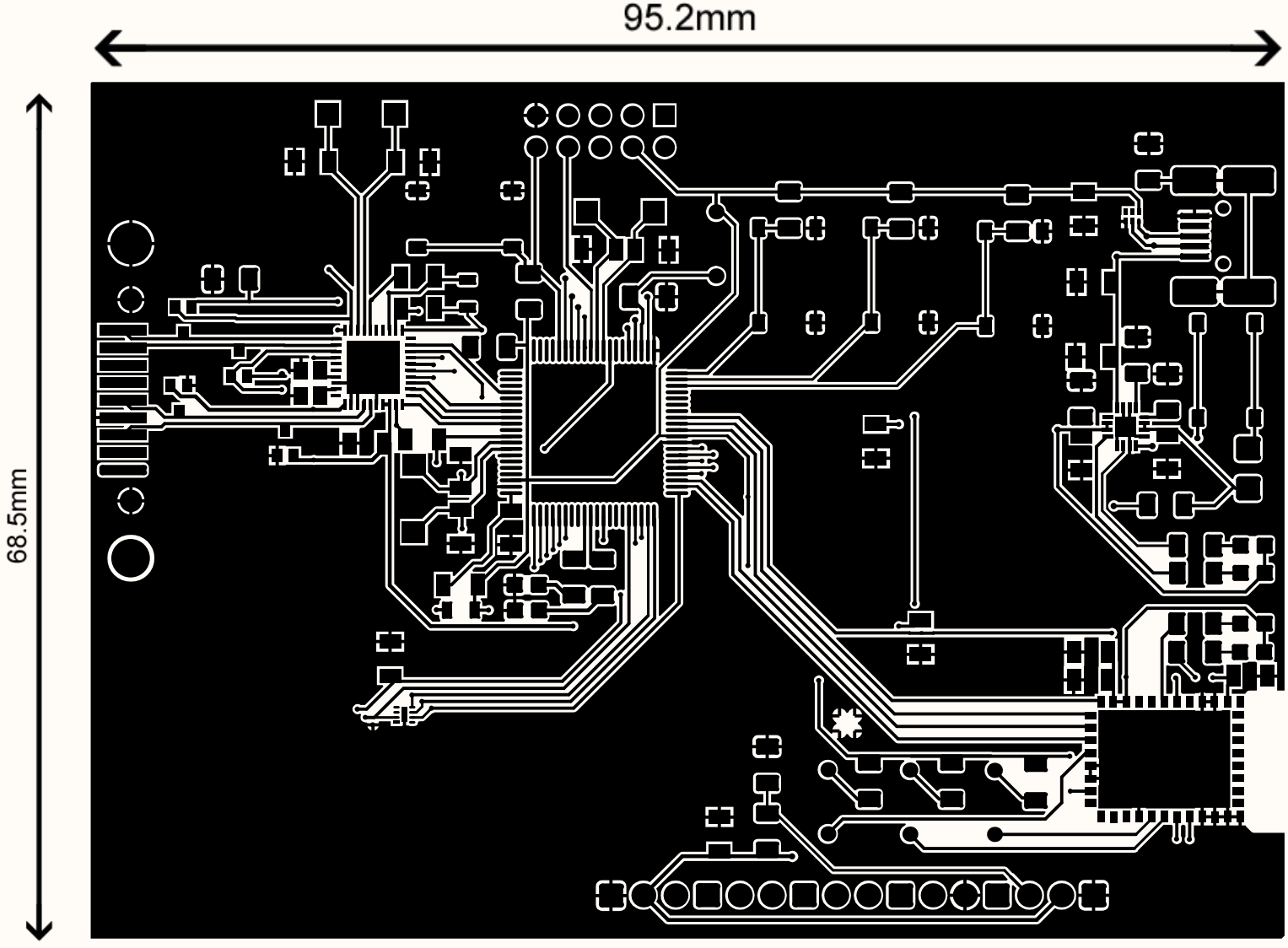
Title: FUENTE DE ALIMENTACIÓN		
Size: A4	Number: 5	Revision: 6
Date: 6/12/2016		Sheet of
File: C:\Users\..\PWS.SchDoc		Drawn By: DAVID MARTÍN SÁNCHEZ



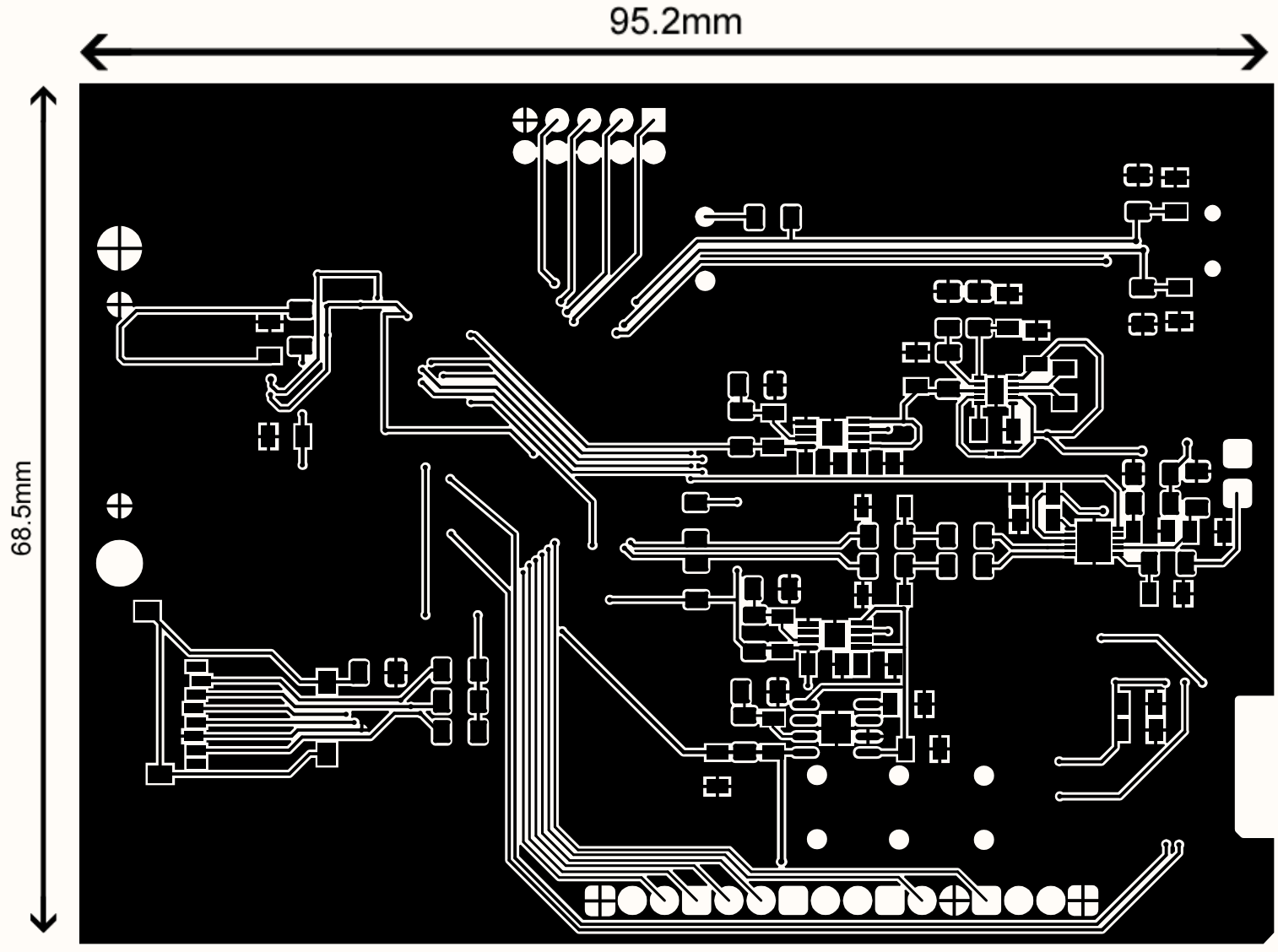
Title: MEMORIA		
Size: A4	Number: 6	Revision: 6
Date: 6/12/2014	Sheet of	
File: C:\Users\..\SD.SchDoc	Drawn By: DAVID MARTÍN SÁNCHEZ	


Regulador de tensión - 5V
U8
VOUT
IN
OUT
SD
ADJ
NC
BYP
NC
DAP
GND
LP3878MR-ADJ/NOPB
C34
Cap
0.1uF
C35
Cap
0.1uF
C36
Cap
1uF
R35
±1%
1K
R36
±1%
4K
C37
Cap
10uF
TX_CTRL_SUP
LED_DRV_SUP
PGND
Conversor de tensión
LDC1
Inductor
4.7uH
U7
GND
VIN
CP2
CP1
EN
VO
SW
OUT
FB
SS
OUT
VIN
C29
Cap
0.1uF
VO
VOUT
FB
SS
TPS61093
PGND
Regulador de tensión - 3V
RX_DIG_SUP
RX_ANA_SUP
FB1
742792042
U9
OUT
FB
NC
GND
IN
DNC
NR/SS
EN
TPS7A4901
C38
Cap
10uF
C39
Cap
10nF
R38
±1%
75K
R37
±1%
47K
C40
Cap
10uF
C41
Cap
10nF
VOUT
PGND
VIN
VO
SS
FB
VOUT
C30
Cap
4.7uF
C31
Cap
0.1uF
C32
Cap
0.1uF
R32
±1%
200K
R33
±1%
15K
R34
±1%
180K
C33
Cap
1uF
PGND
Regulador de tensión - 3V
MCU_DVCC
MCU_AVCC
FB2
742792042
U10
OUT
FB
NC
GND
IN
DNC
NR/SS
EN
TPS7A4901
C42
Cap
10uF
C43
Cap
10nF
R40
±1%
75K
R39
±1%
47K
C44
Cap
10uF
C45
Cap
10nF
VOUT
PGND
Title
REGULADORES DE TENSIÓN
Size
A4
Number
7
Revision
6
Date:
6/12/2014
Sheet of
File:
C:\Users\..\DC.SchDoc
Drawn By:
DAVID MARTÍN SÁNCHEZ

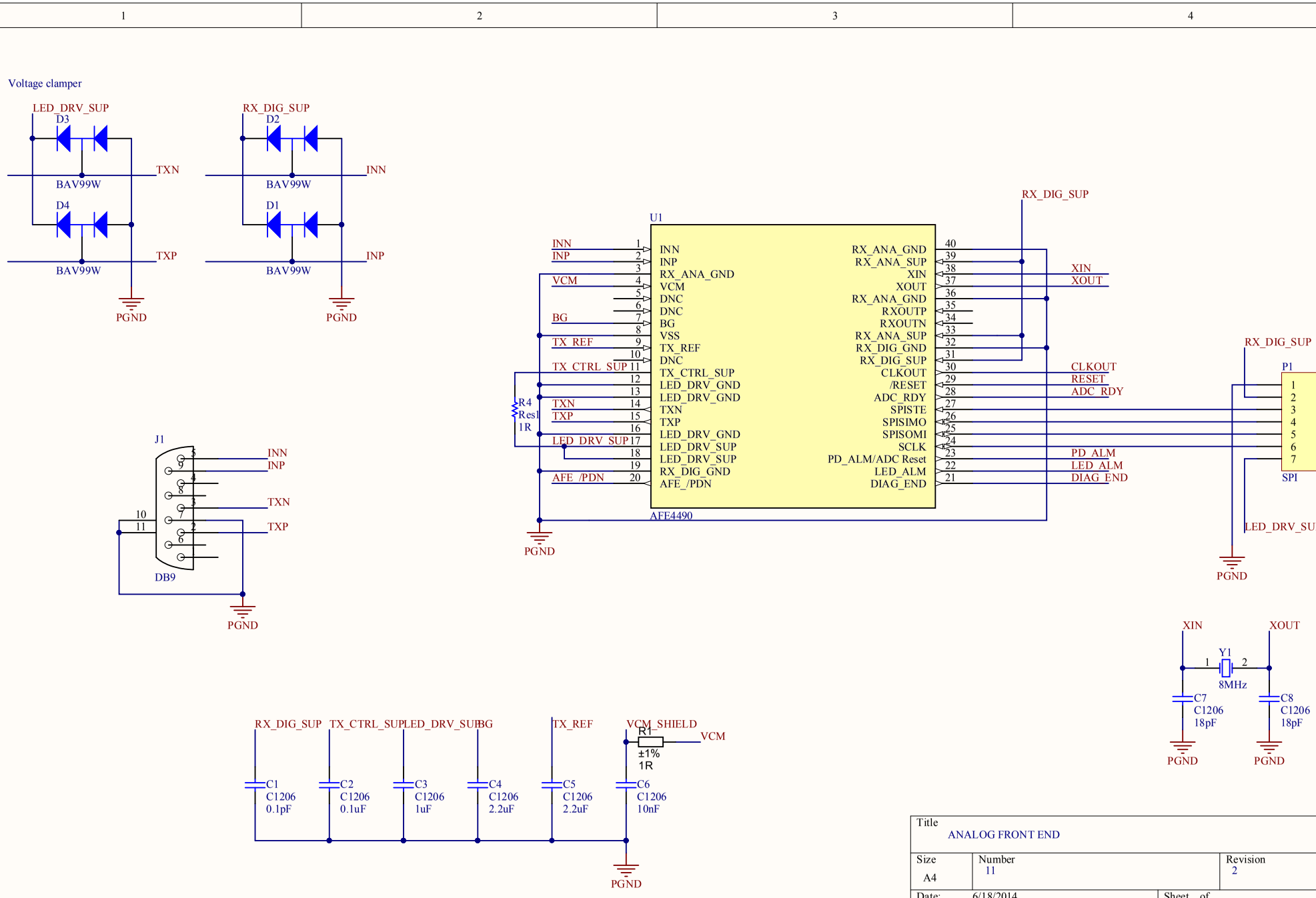

Indicadores LED
P2.0
P2.1
R41 ±1% 1K
R42 ±1% 1K
LED5 Green
LED6 Green
PGND
Botones
MCU_DVCC
R46 ±1% 10K
R47 ±1% 10K
R48 ±1% 10K
R49 ±1% 47K
B1 TS-1143S-WP On/Off
B2 TS-1143S-WP On/Off
B3 TS-1143S-WP On/Off
B4 TS-1143S-WP On/Off
SE
NO
OK
SBWTDIO
MCU_AVCC
MCU_DVCC
U11
P6.4/CB4/A4
P6.5/CB5/A5
P6.6/CB6/A6
P6.7/CB7/A7
P7.0/CB8/A12
P7.1/CB/9/A13
P7.2/CB10/A14
P7.3/CB11/A15
P5.0/A8/VREF+/VeREF+
P5.1/A9/VREF-/VeREF-
AVCC1
P5.4/XIN
P5.5/XOUT
AVSS1
P8.0
P8.1
P8.2
DVCC1
DVSS1
VCORE
P1.0/TA0CLK/ACLK
P1.1/TA0.0
P1.2/TA0.1
P1.3/TA0.2
P1.4/TA0.3
P1.5/TA0.4
P1.6/TA1CLK/CBOUT
P1.7/TA1.0
P2.0/TA1.1
P2.1/TA1.2
P2.2/TA2CLK/SMCLK
P2.3/TA2.0
P2.4/TA2.1
P2.5/TA2.2
P2.6/RTCCLK/DMAE0
P2.7/UCB0STE/UCA0CLK
P3.0/UCB0SIMO/UCB0SDA
P3.1/UCB0SOMI/UCB0SCL
P3.2/UCB0CLK/UCA0STE
P3.3/UCA0TXD/UCA0SIMO
P6.3/CB3/A3
P6.2/CB2/A2
P6.1/CB1/A1
P6.0/CB0/A0
/RST/NMI/SBWTDIO
PJ.3/TCK
PJ.2/TMS
PJ.1/TDI/TCLK
PJ.0/TDO
TEST/SBWTCK
P5.3/XT2OUT
P5.2/XT2IN
AVSS2
V18
VUSB
VBUS
PU.1/DM
PUR
PU.0/DP
VSSU
P7.7/TB0CLK/MCLK
P7.6/TB0.4
P7.5/TB0.3
P7.4/TB0.2
P5.7/TB0.1
P5.6/TB0.0
P4.7/PM_NONE
P4.6/PM_NONE
P4.5/PM_UCA1RXD/PM_UCA1SOMI
P4.4/PM_UCA1TXD/PM_UCA1SIMO
DVCC2
DVSS2
P4.3/PM_UCB1CLK/PM_UCA1STE
P4.2/PM_UCB1SOMI/PM_USB1SCL
P4.1/PM_UCB1SIMO/PM_UCB1SDA
P4.0/PM_UCB1STE/PM_UCA1CLK
P3.7/TB0OUTH/SVMOUT
P3.6/TB0.6
P3.5/TB0.5
P3.4/UCA0RXD/UCA0SOMI
MSP430F5529
AFE_/RESET
/RESET
AFE_PD_ALM
AFE_LED_ALM
AFE_DIAG_END
AFE_/PDN
BLUE_XIN
BLUE_XOUT
32K-
32K+
VCORE
LCD_DB7
LCD_DB6
LCD_DB5
LCD_DB4
LCD_ENA
LCD_REG_SEL
P2.0
P2.1
AFE_ADC_RDY
SD_SCLK
SDA
SCL
SD_STE
SD_SIMO
SDA
SCL
SBWTDIO
TCK
TMS
TDI
TDO
SBWTCK
XT2OUT
XT2IN
VBUS
DM
DP
SE
OK
NO
BLUE_/RESET
UART_RXD
UART_TXD
AFE_STE
AFE_SOMI
AFE_SIMO
AFE_SCLK
PWS_PGM
UART_/RTS
UART_/CTS
SD_SOMI
Debugging
J6
JTAG
SBWTDIO
SBWTCK
MCU_DVCC
TCK
TMS
TDI
TDO
W4 Jumper
R50 ±1% 1M
32K+
32K-
Y2
32.768KHz
C46 Cap 22pF
C47 Cap 22pF
MCU_DVCC
R43 ±1% 10K /RESET
R44 ±1% 1K SCL
R45 ±1% 1K SDA
XT2IN
XT2OUT
Y3
24MHz
C51 Cap 10pF
C52 Cap 10pF
MCU_DVCC
C48 Cap 0.1uF
VCORE
C49 Cap 470pF
MCU_AVCC
C50 Cap 0.1uF
PGND
AGND
Title
MICROCONTROLADOR
Size
A4
Number
8
Revision
6
Date:
6/12/2014
File:
C:\Users\..\TOP.SchDoc
Sheet of
Drawn By:
DAVID MARTÍN SÁNCHEZ

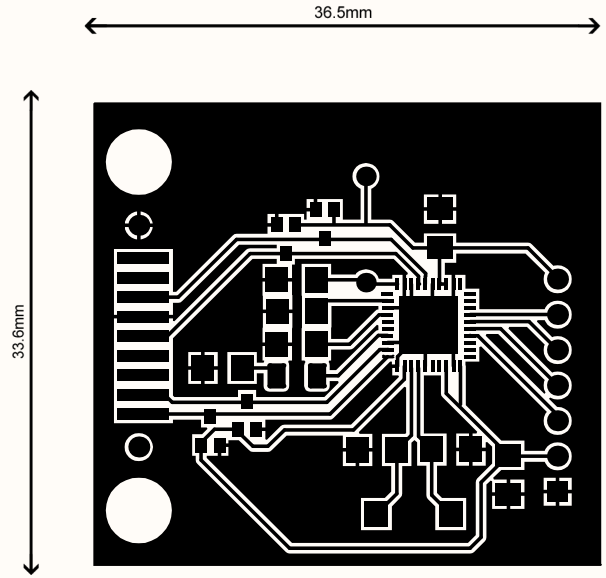


Title: LAYOUT TOP		
Size: A4	Number: 9	Revision: 6
Date: 6/12/2014		Sheet of
File: C:\Users\..\LCD.SchDoc		Drawn By: DAVID MARTÍN SÁNCHEZ



Title: LAYOUT BOTTOM		
Size: A4	Number: 10	Revision: 6
Date: 6/12/2014		Sheet of
File: C:\Users\..\LCD.SchDoc		Drawn By: DAVID MARTÍN SÁNCHEZ


Voltage clamper
LED_DRV_SUP
D3
BAV99W
TXN
D4
BAV99W
TXP
PGND
RX_DIG_SUP
D2
BAV99W
INN
D1
BAV99W
INP
PGND
U1
INN
INP
RX_ANA_GND
VCM
DNC
DNC
BG
VSS
TX_REF
DNC
TX_CTRL_SUP
LED_DRV_GND
LED_DRV_GND
TXN
TXP
LED_DRV_GND
LED_DRV_SUP
LED_DRV_SUP
RX_DIG_GND
AFE_/PDN
RX_ANA_GND
RX_ANA_SUP
XIN
XOUT
RX_ANA_GND
RXOUTP
RXOUTN
RX_ANA_SUP
RX_DIG_GND
RX_DIG_SUP
CLKOUT
/RESET
ADC_RDY
SPISTE
SPISIMO
SPISOMI
SCLK
PD_ALM/ADC Reset
LED_ALM
DIAG_END
AFE4490
RX_DIG_SUP
XIN
XOUT
CLKOUT
RESET
ADC_RDY
PD_ALM
LED_ALM
DIAG_END
R4
Resl
1R
PGND
RX_DIG_SUP
P1
SPI
LED_DRV_SUP
PGND
J1
INN
INP
TXN
TXP
DB9
PGND
XIN
XOUT
Y1
8MHz
C7
C1206
18pF
C8
C1206
18pF
PGND
PGND
RX_DIG_SUP
TX_CTRL_SUP
LED_DRV_SUP
BG
TX_REF
VCM_SHIELD
R1
±1%
1R
VCM
C1
C1206
0.1pF
C2
C1206
0.1uF
C3
C1206
1uF
C4
C1206
2.2uF
C5
C1206
2.2uF
C6
C1206
10nF
PGND
Title
ANALOG FRONT END
Size
A4
Number
11
Revision
2
Date:
6/18/2014
Sheet of
File:
C:\Users\..\AFE.SchDoc
Drawn By:
DAVID MARTÍN SÁNCHEZ



Title: LAYOUT MÓDULO AFE		
Size: A4	Number: 12	Revision: 2
Date: 6/12/2014		Sheet of
File: C:\Users\..\LCD.SchDoc		Drawn By: DAVID MARTÍN SÁNCHEZ

Anexo II. Listado de materiales

A continuación se adjunta una tabla con todos los componentes seleccionados para el dispositivo diseñado en este trabajo, en la que se desglosa además su modelo y encapsulado, así como el precio, procediendo al final de la tabla a mostrar el presupuesto total del dispositivo.

Componente	Descripción	Valor/modelo	Encapsu-lado	Ud.	Precio ud.	Precio total
U1	Analog front end	AFE4490		1	7,95	7,95
U2	Protección EMD	TPD4E004		1	0,22	0,22
U3	Módulo Bluetooth	LMX9838		1	16,97	16,97
U4	Cargador de batería	BQ24072		1	1,15	1,15
U5	Controlador de carga	BQ27200		1	1,49	1,49
U6	Protección EMD	TPD8E003		1	0,18	0,18
U7	Convertidor de tensión	TPS61093		1	1,38	1,38
U8	Regulador de tensión	LP3878		1	0,96	0,96
U9, U10	Regulador de tensión	TPS7A4901		2	1,1	2,2
U11	Microcontrolador	MSP430F5529		1	3,58	3,58
D1, D2, D3, D4	Array de diodos de alta capacidad de conmutación	BAV99W	SOT-323	4	0,035	0,14
J1	Conector SpO2	DB9		1	0,782	0,782
J2	Array de pines hembra	16x1		1	0,012	0,012
J3	Conector USB mini-B	548190572		1	0,506	0,506
J4	Array de pines macho	2x1		1	0,08	0,08
J5	Conector micro SD	473340001		1	0,42	0,42
J6	Array de pines macho	5x2		1	0,01	0,01
C1, C2, C3, C20, C33, C36	Condensador	1 uF	1206	6	0,005	0,03
C4, C10, C13, C14, C16, C18, C23, C24, C25, C26, C27, C28, C29, C31, C32, C34, C35, C48, C50	Condensador	0,1 uF	1206	19	0,005	0,095
C5, C6, C15, C17	Condensador	2,2 uF	805	4	0,005	0,02
C7, C39, C41, C43, C45	Condensador	10 nF	1206	5	0,005	0,025
C8, C9	Condensador	18 nF	1206	2	0,005	0,01
C11, C12, C51, C52	Condensador	10 pF	1206	4	0,005	0,02
C19	Condensador	0,1 pF	1206	1	0,005	0,005
C21, C22, C30	Condensador	4,7 uF	1206	3	0,005	0,015
C37, C38, C40, C42, C44	Condensador	10 uF	1206	5	0,005	0,025

C46, C47	Condensador	22 pF	1206	2	0,005	0,01
C49	Condensador	470 pF	1206	1	0,005	0,005
R1	Resistencia	1	1206	1	0,001	0,001
R2, R3, R9, R10, R11, R14, R15, R17, R18, R23, R25, R27, R35, R41, R42, R44, R45	Resistencia	1K	1206	17	0,001	0,017
R4	Resistencia	270	1206	1	0,001	0,001
R5	Resistencia	1,5K	1206	1	0,001	0,001
R6, R7, R33	Resistencia	15K	1206	3	0,001	0,003
R8, R31	Resistencia	33K	1206	2	0,001	0,002
R12, R13	Resistencia	330	1206	2	0,001	0,002
R16, R24, R43, R46, R47, R48	Resistencia	10K	1206	6	0,001	0,006
R19, R20, R21, R22	Resistencia	100	1206	4	0,001	0,004
R26	Resistencia	0,02	1206	1	0,001	0,001
R28, R29, R30, R37, R39, R49	Resistencia	47K	1206	6	0,001	0,006
R32	Resistencia	200K	1206	1	0,001	0,001
R34	Resistencia	180K	1206	1	0,001	0,001
R36	Resistencia	4K	1206	1	0,001	0,001
R50	Resistencia	1M	1206	1	0,001	0,001
R38, R40	Resistencia	75K	1206	2	0,001	0,002
LED1, LED2	Diodo LED	Azul	805	2	0,058	0,116
LED3, LED4	Diodo LED	Verde	805	2	0,058	0,116
LED5, LED6	Diodo LED	Amarillo	805	2	0,058	0,116
W1, W2, W3, W4	Jumper	2x1		4	0,08	0,32
Z1, Z2	Diodo Zener	5,6 V	SOD-323	2	0,014	0,028
Y1	Cristal oscilador	8 MHz		1	1,65	1,65
Y2	Cristal oscilador	32,768 KHz		1	1,3	1,3
Y3	Cristal oscilador	24 MHz		1	1,61	1,61
S1	Interruptor	6,5x6,5mm		1	0,63	0,63
B1, B2, B3, B4	Pulsador	6,5x6,5mm		4	0,02	0,08
FB1, FB2	Ferrite Bead			2	0,2	0,4

LDC1	Inductancia	4,7 uH	603	1	0,012	0,012
					TOTAL componentes	44,718
					PCB dos caras	4,786
					Sensor SpO2	12,5
					Tarjeta memoria micro SD	4,45
					TOTAL dispositivo	66,454

Anexo III. Código del dispositivo

A continuación se adjunta el código de los principales ficheros utilizados para la ejecución del programa principal del dispositivo.

deteccionPicos.m

Esta función, programada en el lenguaje de alto nivel que utiliza el programa Matlab, detecta los picos de una señal que se pasa como parámetro cuando se llama a esta función (data) y devuelve el vector que recoge la HRV.

```
function hrv=deteccionPicos(data)
samp_freq=125; %Frecuencia de muestreo de la base de datos MIMIC II
% En primer lugar, se determina la longitud del vector 'data' y se
% crea el eje temporal
[a b] = size(data);
if(a>b)
 len =a;
end
if(b>a)
 len =b;
end
if (a | b == 1);
  tt = 1/samp_freq:1/samp_freq:ceil(len/samp_freq);
  t = tt(1:len); % t será el eje temporal
end
% Se construye un vector compuesto por la diferencia entre el valor
% en el índice i de 'data' y el que hay en i+1 y se detecta qué
% índice corresponde a un máximo según la regla que se indica
A=diff(data);
A1=A(2:end);
A2=A(1:end-1);
imx=find(A1.*A2<=0 & A1-A2<0 & A2>0);
% Una vez obtenidos los índices correspondientes a los picos de la
% señal, se procede a seleccionar aquellos que son picos sistólicos
n=1;
m=1;
for k=1:imx(end)
   if k==imx(n)
       if data(k)>0.4*max(data(1:30))
       detec(m)=data(k);
       detect(m)=imx(n);
       m=m+1;
       end
       n=n+1;
   end
end
P_t=t(detect);
hrv=diff(P_t);
end
```

lcd16.h

Fichero de cabecera de las funciones de la pantalla.

```
/*
 * lcd16.h
 *
 *  Creado:  Dec 1, 2011 by Gauray www.circuitvalley.com
 *  Modificado: May 18,2014 by David Martín Sánchez
 *
 */

#ifndef LCD16_H_
#define LCD16_H_

#include <msp430f5529.h>
#define  EN BIT4
#define  RS BIT5

void waitlcd(unsigned int x);
void lcdinit(void);
void integerToLcd(int integer);
void integerToLcd3(int integer);
void lcdData(unsigned char l);
void prints(char *s);
void gotoXy(unsigned char  x,unsigned char y);
#endif /* LCD16_H_ */
```

lcd16.c

Código fuente de la pantalla.

```
/*
 * lcd16.c
 *
 *  Creado:  Dec 1, 2011 by Gauray www.circuitvalley.com
 *  Modificado: May 18,2014 by David Martín Sánchez
 *
 */
#include "lcd16.h"

void lcdcmd(unsigned char Data)
{
P1OUT &= ~RS; //because sending command
P1OUT &= ~EN;
P1OUT &= 0xF0;
P1OUT |= ((Data >> 4) & 0x0F);
P1OUT |= EN;
waitlcd(2);
P1OUT &= ~EN;
P1OUT &= 0xF0;
P1OUT |= (Data & 0x0F);
P1OUT |= EN;
waitlcd(2);
P1OUT &= ~EN;
}
void lcdData(unsigned char l)
{
```

```
P1OUT |= RS;  //because sending data
P1OUT &= ~EN;
P1OUT &= 0xF0;
P1OUT |= ((l >> 4) & 0x0F);
P1OUT |= EN;
waitlcd(2);
P1OUT &= ~EN;
P1OUT &= 0xF0;
P1OUT |= (l & 0x0F);
P1OUT |= EN;
waitlcd(2);
P1OUT &= ~EN;
}
void lcdinit(void)
{
P1OUT &= ~RS;
P1OUT &= ~EN;
P1OUT |= 0x3;
waitlcd(40);
P1OUT |= EN;
P1OUT &= ~EN;
waitlcd(5);
P1OUT |= EN;
P1OUT &= ~EN;
waitlcd(5);
P1OUT |= EN;
P1OUT &= ~EN;
waitlcd(2);
P1OUT &= 0xF2;
P1OUT |= EN;
P1OUT &= ~EN;
lcdcmd(0x28);
waitlcd(250);
lcdcmd(0x0E);
waitlcd(250);
lcdcmd(0x01);
waitlcd(250);
lcdcmd(0x06);
waitlcd(250);
lcdcmd(0x80);
waitlcd(250);
}
void prints(char *s)
{
while (*s)
{
 lcdData(*s);
 s++;
}
}
void gotoXy(unsigned char  x,unsigned char y)
{
 if(x<40)
 {
  if(y) x |= 0x40;
  x |=0x80;
  lcdcmd(x);
  }
```

```
}
void integerToLcd(int integer)
{
unsigned char tens,ones;
tens=(integer%100)/10;
lcdData( tens + 0x30);
ones=integer%10;
lcdData( ones + 0x30);
}
void integerToLcd3(int integer )
{
unsigned char thousands,hundreds,tens,ones;
thousands = integer / 1000;
hundreds = ((integer - thousands*1000)-1) / 100;
lcdData( hundreds + 0x30);
tens=(integer%100)/10;
lcdData( tens + 0x30);
ones=integer%10;
lcdData( ones + 0x30);
}
```

fuel_gauge.h

Fichero de cabecera de la función que controla el nivel de la batería.

```
/*
 * fuel_gauge.h
 *
 *  Creado:  Apr 1, 2009 by Bhargavi Nisarga, Texas Instruments
 *  Modificado: May 20,2014 by David Martín Sánchez
 *
 */

#ifndef FUEL_GAUGE_H_
#define FUEL_GAUGE_H_

int readBattery(void);
#endif /* FUEL_GAUGE_H_ */
```

fuel_gauge.c

Código fuente de la batería.

```
/*
 * fuel_gauge.c
 *
 *  Creado:  Apr 1, 2009 by Bhargavi Nisarga, Texas Instruments
 *  Modificado: May 20,2014 by David Martín Sánchez
 *
 */

#include <msp430f5529.h>
#include "fuel_gauge.h"
#include "lcd16.h"

int readBattery(void){
      unsigned char RXData;
```

```
      unsigned char reg_addr=0x0b;
      int level;

      P3SEL |= 0x03;
      UCB0CTL1 |= UCSWRST;
      UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC;
      UCB0CTL1 = UCSSEL_2 + UCSWRST;
      UCB0BR0 = 12;
      UCB0BR1 = 0;
      UCB0I2CSA = 0x48;
      UCB0CTL1 &= ~UCSWRST;
      UCB0IE |= UCRXIE;
      UCB0CTL1 |= UCTR + UCTXSTT;
      UCB0TXBUF=reg_addr;
      while (UCB0CTL1 & UCTXSTT);
    UCB0CTL1 &= ~UCTR;
      __delay_cycles(30);
      UCB0CTL1 |= UCTXSTT;
      while (UCB0CTL1 & UCTXSTT);
      while (!(UCB0IFG & UCTXIFG));
      RXData = UCB0RXBUF;
    UCB0CTL1 |= UCTXSTP;
      while (UCB0CTL1 & UCTXSTP);
      if(RXData > 0x0F)
            level=1;
      else
            level=0;

      return level;
}
```

bluetooth.h

Fichero de cabecera de las funciones para transmitir mediante Bluetooth.

```
/*
 * bluetooth.h
 *
 *  Creado:  Apr 1, 2009 by Bhargavi Nisarga, Texas Intruments
 *  Modificado: May 20,2014 by David Martín Sánchez
 *
 */

#ifndef BLUETOOTH_H_
#define BLUETOOTH_H_

void initBluetooth();
void sendBluetooth(unsigned char *s);

#endif /* BLUETOOTH_H_ */
```

bluetooth.c

Código fuente del Bluetooth.

```
/*
 * bluetooth.c
 *
 *  Creado:  Apr 1, 2009 by Bhargavi Nisarga, Texas Intruments
 *  Modificado: May 20,2014 by David Martín Sánchez
 *
 */

#include "msp430f5529.h"
#include "bluetooth.h"
#include "lcd16.h"

Static unsigned char *data;

void initBluetooth(){
       P4SEL = BIT4+BIT5;
       UCA1CTL1 |= UCSWRST;
       UCA1CTL1 |= UCSSEL_1;
       UCA1BR0 = 0x03;
       UCA1BR1 = 0x00;
       UCA1MCTL |= UCBRS_6+UCBRF_0;
       UCA1CTL1 &= ~UCSWRST;
       UCA1IE |= UCRXIE;
       gotoXy(0,0);
       prints(" BLUETOOTH INIT ");
       gotoXy(0,1);
       prints("  WAS CORRECT!  ");
}

void sendBluetooth(unsigned char *s){
       data=s;
       __bis_SR_register(LPM4_bits + GIE);
       __no_operation();
}

#pragma vector=USCI_A1_VECTOR
__interrupt void USCI_A1_ISR(void)
{
  switch(__even_in_range(UCA1IV,5))
  {
  case 0:break;
  case 2:
    while (!(UCA1IFG&UCTXIFG));
    UCA1TXBUF = *data;
    break;
  case 4:break;
  default: break;
  }
}
```

HAL_SDCard.c

Driver de la tarjeta microSD.

```
/*
 * HAL_SDCard.c
 *
 *  Creado:  Copyright (C) 2010, Texas Instruments Incorporated.
```

```
 * Modificado: May 22, 2014 by David Martín Sánchez
 *
 */

#include "msp430.h"
#include "HAL_SDCard.h"
#define SPI_SIMO        BIT3
#define SPI_SOMI        BIT4
#define SPI_CLK         BIT7
#define SD_CS           BIT2
#define SPI_SEL         P3SEL
#define SPI_DIR         P3DIR
#define SPI_OUT         P3OUT
#define SPI_REN         P3REN
#define SD_CS_SEL       P3SEL
#define SD_CS_OUT       P3OUT
#define SD_CS_DIR       P3DIR
#define SD_CLK_SEL        P2SEL
#define SD_CLK_DIR        P2DIR

void SDCard_init(void)
{
    SPI_SEL |= SPI_SOMI + SPI_SIMO;
    SPI_DIR |= SPI_SIMO;
    SPI_REN |= SPI_SOMI;
    SPI_OUT |= SPI_SOMI;
    SD_CLK_SEL |= SPI_CLK;
    SD_CLK_DIR |= SPI_CLK;
    SD_CS_SEL &= ~SD_CS;
    SD_CS_OUT |= SD_CS;
    SD_CS_DIR |= SD_CS;
    UCB0CTL1 |= UCSWRST;
    UCB0CTL0 = UCCKPL + UCMSB + UCMST + UCMODE_0 + UCSYNC;
    UCB0CTL1 = UCSWRST + UCSSEL_2;
    UCB0BR0 = 63;
    UCB0BR1 = 0;
    UCB0CTL1 &= ~UCSWRST;
    UCB0IFG &= ~UCRXIFG;
}

void SDCard_fastMode(void)
{
    UCB0CTL1 |= UCSWRST;
    UCB0BR0 = 2;
    UCB0BR1 = 0;
    UCB0CTL1 &= ~UCSWRST;
}

void SDCard_readFrame(uint8_t *pBuffer, uint16_t size)
{
    uint16_t gie = __get_SR_register() & GIE;
    __disable_interrupt();
    UCB0IFG &= ~UCRXIFG;
    while (size--){
        while (!(UCB0IFG & UCTXIFG)) ;
        UCB0TXBUF = 0xff;
        while (!(UCB0IFG & UCRXIFG)) ;
        *pBuffer++ = UCB0RXBUF;
```

```
    }

    __bis_SR_register(gie);
}

void SDCard_sendFrame(uint8_t *pBuffer, uint16_t size)
{
    uint16_t gie = __get_SR_register() & GIE;
    __disable_interrupt();
    while (size--){
        while (!(UCB0IFG & UCTXIFG)) ;
        UCB0TXBUF = *pBuffer++;
    }
    while (UCB0STAT & UCBUSY) ;
    UCB0RXBUF;
    __bis_SR_register(gie);
}

void SDCard_setCSHigh(void)
{
    SD_CS_OUT |= SD_CS;
}

void SDCard_setCSLow(void)
{
    SD_CS_OUT &= ~SD_CS;
}
```

AFE44x0.h

Librería para el manejo del módulo AFE4490.

```
/*
 *  maths_functions.c
 *
 *  Creado:  (C) Texas Instruments Inc., 2013 by Praveel Aroul
 *  Modificado: May 10, 2014 by David Martín Sánchez
 */

#ifndef AFE44x0_H_
#define AFE44x0_H_

#define AFE_RESETZ      BIT7
#define AFE_PDNZ        BIT0
#define AFE_ADC_DRDY    BIT3
#define AFE_PD_ALM      BIT1
#define AFE_LED_ALM     BIT2
#define AFE_DIAG_END    BIT3

struct AFE44xx_state{
      unsigned char state;
      unsigned char SamplingRate;
      unsigned char command;
};

typedef enum stECG_RECORDER_STATE {

      IDLE_STATE =0,
```

```
      DATA_STREAMING_STATE,
      ACQUIRE_DATA_STATE,
      ECG_DOWNLOAD_STATE,
      ECG_RECORDING_STATE
}ECG_RECORDER_STATE;

void Init_AFE44xx_Resource(void);
void AFE44xx_Default_Reg_Init(void);
void AFE44xx_Reg_Write(unsigned char Reg_address, unsigned long Reg_data);
unsigned long AFE44xx_Reg_Read(unsigned char Reg_address);
void Init_AFE44xx_DRDY_Interrupt (void);
void Enable_AFE44xx_DRDY_Interrupt (void);
void Disable_AFE44xx_DRDY_Interrupt (void);
void Set_GPIO(void);
void Set_UCB1_SPI(void);
void AFE44xx_Read_All_Regs(unsigned long AFE44xxeg_buf[]);
void AFE44xx_PowerOn_Init(void);
void AFE44xx_Parse_data_packet(void);
void ADS1292x_Parse_data_packet(void);
void Set_Device_out_bytes(void);

#endif /*AFE44x0_H_*/
```

maths_functions.h

Fichero de cabecera de las funciones utilizadas por el programa principal para detectar el nivel de estrés y calcular la saturación en sangre y los latidos por minuto.

```
/*
 *  maths_functions.c
 *
 *  Creado:  May 18, 2014 by David Martín Sánchez
 *
 */
#include "msp430f5529.h"
#include "../microSD/integer.h"
#include "../microSD/ff.h"

unsigned long *peakDetect(FIL * fp, UINT btr);
unsigned long mean(unsigned long * s, UINT btr);
unsigned long std(unsigned long * s, UINT btr);
unsigned long rmssd(unsigned long * s, UINT btr);
int pnn50(unsigned long * s, UINT btr);
unsigned long sd1(unsigned long * s, UINT btr);
unsigned long sd2(unsigned long * s, UINT btr);
unsigned long Apen(unsigned long r, unsigned long * s, UINT btr);
```

maths_functions.c

Código fuente para el cálculo de estrés y saturación en sangre.

```
/*
 *  maths_functions.c
 *
 *  Created on:  May 18, 201a by David Martín Sánchez
 *
```

```
 */
#include "msp430f5529.h"
#include "maths_functions.h"
#include "microSD/ff.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "../microSD/integer.h"
#include "../microSD/ff.h"

unsigned long *peakDetect(FIL * fp, UINT btr){

	unsigned long *s=malloc(sizeof(unsigned int)* btr);
	unsigned long *p=malloc(sizeof(unsigned int)* btr);
	unsigned long *hrv=malloc(sizeof(unsigned int)* btr);
	int long *a=malloc(sizeof(unsigned int)* btr);
	float *tt=malloc(sizeof(float)* btr);
	unsigned long e[5];
	unsigned int i, fs, j;
	UINT bytesRead;

	fs=125;

	for(i=0;i<btr;i++){
		f_read(fp, &s[i], 8, &bytesRead);

		f_read(fp, &e, 5*8, &bytesRead);
	}

	for(i=1;i<=ceil(btr/fs);i++){
		tt[i]=i/fs;
	}

	for(i=1, a[0]=s[0];i<btr;i++){
		a[i]=s[i]-s[i-1];
	}

	for(i=0, j=0;i<btr;i++){
		if(a[i+1]*a[i]<=0 && a[i+1]-a[i]<0 && a[i]>0){
			if(s[i]>0){
				p[j]=i;
				j++;
			}
		}
	}

	for(i=1, hrv[0]=p[0];i<btr;i++){
		hrv[i]=p[i]-p[i-1];
	}
	f_close(&file);
	f_mount(0,0);

	return hrv;
}

unsigned long mean(unsigned long * s, UINT btr){
	int mean, i;
```

```
	for(i=0, mean=0;i<btr;i++){
		mean+=(s[i]/btr);
	}

	return mean;
}

unsigned long std(unsigned long * s, UINT btr){
	unsigned long std;
	unsigned long * sq=malloc(sizeof(unsigned int)* btr);
	int i;

	for(i=0;i<btr;i++){
		sq[i]=s[i]^2;
	}

	std=sqrt(mean(sq, btr)-mean(s,btr)^2);

	return std;
}

unsigned long rmssd(unsigned long * s, UINT btr){
	unsigned long rmssd;
	int i;

	for(i=0, rmssd=0;i<btr-1;i++){
		rmssd+=((s[i+1]-s[i])^2)/(btr-1);
	}

	rmssd=sqrt(rmssd);

	return rmssd;
}

int pnn50(unsigned long * s, UINT btr){
	int i, pnn50;

	for(i=0,pnn50=0;i<btr;i++){
		if(s[i+1]-s[i]>50)
			pnn50++;
	}

	pnn50*=(100/btr);

	return pnn50;
}

unsigned long sd1(unsigned long * s, UINT btr){
	unsigned long sd1, p;
	unsigned long *x=malloc(sizeof(unsigned int)* btr);
	int i;

	for(i=0;i<btr;i++){
		x[i]=(s[i]-s[i+1])/sqrt(2);
	}

	for(i=0,p=0;i<btr;i++){
		p+=(x[i]^2)/btr;
```

```
	}

	sd1=sqrt(p-mean(x, btr)^2);

	return sd1;
}

unsigned long sd2(unsigned long * s, UINT btr){
	unsigned long sd2, p;
	unsigned long *x=malloc(sizeof(unsigned int)* btr);
	int i;

	for(i=0;i<btr;i++){
		x[i]=(s[i]+s[i+1])/sqrt(2);
	}

	for(i=0,p=0;i<btr;i++){
		p+=(x[i]^2)/btr;
	}

	sd2=sqrt(p-mean(x, btr)^2);

	return sd2;
}

unsigned long Apen(unsigned long r, unsigned long * s, UINT btr){
	unsigned long entropia,phi[2];
	int i,j,k,m;

	for(j=0;j<=2;j++){
		m=2+j;
		unsigned long *ci=malloc(sizeof(unsigned int)* (btr-m+1));
		unsigned long *dataMat=malloc(sizeof(unsigned int)* (btr-1));

		for(i=0;i<btr-m+1;i++){
			for(k=0;k<btr-1;k++){
				if(dataMat[k]=s[k]-s[i]<0)
					dataMat[k]*=(-1);
				if(dataMat[k]<r)
					ci[i]++;
			}
			ci[i]=ci[i]/(btr-m+1);
		}
		for(i=0;i<btr-m+1;i++){
			phi[j]+=log(ci[i])/(btr-m+1);
		}
	}

	entropia=phi[0]-phi[1];

	return entropia;
}
```

button.h

Fichero de cabecera con las funciones para manejar los botones del dispositivo.

```
/*
```

```
 *  button.h
 *
 *  Creado:  May 26, 2014 by David Martín Sánchez
 *
 */

#ifndef BUTTON_H_
#define BUTTON_H_

#include <msp430f5529.h>
#define  SE BIT7
#define  OK BIT6
#define  NO BIT5

void Init_Buttons(void);
int readButton(int button);

#endif /* BUTTON_H_ */
```

button.c

Código fuente para los botones pulsadores.

```
/*
 *  button.c
 *
 *  Creado:  May 26, 2014 by David Martín Sánchez
 *
 */
#include "msp430f5529.h"
#include "button.h"

void Init_Buttons(void){
	P7DIR &= ~(SE+OK+NO);
	P7REN |= SE+OK+NO;
	P7OUT |= SE+OK+NO;
}

int readButton(int button){
	int estado = 0;

	switch(button){
	case 1:
		estado=P7IN & NO;
		break;
	case 2:
		estado=P7IN & SE;
		break;
	case 3:
		estado=P7IN & OK;
		break;
	default:;
	}

	return estado;
}
```

main.c

Programa principal.

```
/*
 * ======== main.c ========
 * Stress detection
 * ------------------------------------------------------------------------*/

#include <intrinsics.h>
#include <string.h>
#include <msp430f5529.h>
#include "USB/USB_config/descriptors.h"
#include "USB/USB_API/USB_Common/device.h"
#include "USB/USB_API/USB_Common/types.h"
#include "USB/USB_API/USB_Common/usb.h"
#include "F5xx_F6xx_Core_Lib/HAL_UCS.h"
#include "F5xx_F6xx_Core_Lib/HAL_PMM.h"
#include "USB/USB_API/USB_CDC_API/UsbCdc.h"
#include "USB/USB_User/usbConstructs.h"
#include "AFE44x0_MSP430F5529_APP/AFE44x0.h"
#include "AFE44x0_FWVersion.h"
#include "AFE44x0_main.h"
#include "lcd16.h"
#include "maths_functions.h"
#include "fuel_gauge.h"
#include "bluetooth.h"
#include "microSD/ff.h"
#include "microSD/diskio.h"
#include "button.h"

#define MAX_STR_LENGTH 64

//============================================================================

//Function declarations
VOID Init_Ports (VOID);
VOID Init_Clock (VOID);
VOID Init_TimerA1 (VOID);
VOID Init_TimerB(VOID);
BYTE retInString (char* string);
void initCLK(void);
void SetVcoreUp (unsigned int level);
FRESULT WriteFile(char*, char*, WORD);
void fat_init(void);
VOID measureSpO2(VOID);
VOID measureStress(VOID);
unsigned char ascii2uint8 (unsigned char asciiVal);

//Global flags set by events
volatile BYTE bCDCDataReceived_event = FALSE;

//Global variable declarations
char wholeString[MAX_STR_LENGTH] = "";
unsigned long AFE44xx_SPO2_Data_buf[6];
unsigned char txString[MAX_STR_LENGTH] = "";
char sendDataFlag = 0;
char readDataFlag = 0;
unsigned long AFE44xxRegArr[49];
```

```
unsigned char AFE44xxRegAddr[49];
unsigned char AFE44xxAddr;
unsigned long AFE44xxRegVal;
unsigned int totalCount;
unsigned int sampleCount;
FIL file;                                                   /* Opened file object */
FATFS fatfs;                                                /* File system object */
DIRS dir;                                                   /* Directory object   */
FRESULT errCode;                                            /* Error code object  */
FRESULT res;                                                /* Result object      */
UINT bytesRead;                                             /* Bytes read object  */
UINT read;                                                  /* Read bytes object  */
unsigned char MST_Data,SLV_Data;
BYTE buffer[32];
int result=1;
static int seconds;
int n=0;

/*
 * ======== main ========
 */
VOID main (VOID)
{
    //
    // Local variable declaration
    //
    unsigned char Reg_Init_i;
    unsigned int bytesRead;
    unsigned int bytesWritten;
    int level=1;                //Level of battery (0 low, 1 correct)
    int measure=0;              //Measure type (0 Sp02, 1 Stress)
    int buttonOK=0;             //Button OK on device
    int count=0;                //Time recording PPG signal
    int transmit=0;             //Communication via (0 USB, 1 Bluetooth, 2 both, 3 none)
    int flag=0;

    WDTCTL = WDTPW + WDTHOLD;                               // Stop watchdog timer

    //
    // First initialization: screen and watch dog
    //
    P1DIR = 0xFF;
    P1OUT = 0x00;
    lcdinit();

    //
    // Battery level measure
    //
    level=readBattery();
    if(level == 0){                 //If it's low, program ends
      gotoXy(0,0);
      prints("  LOW  BATTERY  ");
      gotoXy(0,1);
      prints(" PLEASE CONNECT ");
    }
    else{                           //If it's correct, device initialization starts
      gotoXy(0,0);
      prints("  INITIALIZING  ");
```

```
gotoXy(0,1);
prints("  PLEASE  WAIT  ");

//
// Second initialization: analog front end, buttons, microSD, timers and clocks
//
Init_Ports();                       //Do this first  because clocks do change ports
SetVCore(3);
AFE44xx_PowerOn_Init();
fat_init();
Init_Buttons();
Init_TimerA1();
Init_TimerB();
Init_Clock();                     //Init clocks AFE
initCLK();                         //Init microSD card clock

  //
  // Measure type selection
  //
  gotoXy(0,0);
  prints(">MEASURE SPO2   ");
  gotoXy(0,1);
  prints(" MEASURE STRESS ");
  while(buttonOK==0){
       if(measure==0){
             gotoXy(0,0);
             prints(">");
             gotoXy(0,1);
             prints(" ");
       }
       else{
             gotoXy(0,0);
             prints(" ");
             gotoXy(0,1);
             prints(">");
       }
       flag=0;
       while(flag==0){
             flag=1;
             if(readButton(1)){
                   measure=0;
             }
             else if(readButton(2)){
                   measure=1;
             }
             else if(readButton(3)){
                   buttonOK=1;
             }
             else
                   flag=0;
       }
  }

  //
  // Measure length selection
  //
  if(measure==0)                //If SpO2, 20 seconds window's necessary
       count=20;
```

```
else                              //If Stress, 5 minutes window's necessary
     count=300;

//
// Communication via selection
//
buttonOK=0;
gotoXy(0,0);
prints(">SEND USB      ");
gotoXy(0,1);
prints(" SEND BLUETOOTH ");
gotoXy(15,1);
lcdData(0x7e);                    //Special character: ->
while(buttonOK==0){
     if(transmit==0){
            gotoXy(0,0);
            prints(">");
            gotoXy(0,1);
            prints(" ");
     }
     else if(transmit==1){
            gotoXy(0,0);
            prints(" ");
            gotoXy(0,1);
            prints(">");
     }
     else if(transmit==2){
            gotoXy(0,0);
            prints(">SENT BOTH      ");
            gotoXy(0,1);
            prints(" DON'T SEND     ");
            gotoXy(15,1);
            lcdData(0x7f);      //Special character: <-
     }
     else if(transmit==3){
            gotoXy(0,0);
            prints(" ");
            gotoXy(0,1);
            prints(">");
     }
     flag=0;
     while(flag==0){
            flag=1;
            if(readButton(1)){
                  if(transmit!=0)
                        transmit--;
            }
            else if(readButton(2)){
                  if(transmit<3)
                        transmit++;
            }
            else if(readButton(3)){
                  buttonOK=1;
            }
          else
            flag=0;
     }
}
```

```
        //
        // Third initialization: USB, Bluetooth, both or neither
        //
        if(transmit==0){
             USB_init();
             USB_setEnabledEvents(kUSB_VbusOnEvent + kUSB_VbusOffEvent +
kUSB_receiveCompletedEvent + kUSB_dataReceivedEvent + kUSB_UsbSuspendEvent +
kUSB_UsbResumeEvent + kUSB_UsbResetEvent); //Enable various USB event handling routines
             if (USB_connectionInfo() & kUSB_vbusPresent){                    //See if
we're already attached physically to USB, and if so, connect to it
                   USB_handleVbusOnEvent();
             }
        }
        else if(transmit==1)
             initBluetooth();
        else if(transmit==2){
             USB_init();
             USB_setEnabledEvents(kUSB_VbusOnEvent + kUSB_VbusOffEvent +
kUSB_receiveCompletedEvent + kUSB_dataReceivedEvent + kUSB_UsbSuspendEvent +
kUSB_UsbResumeEvent + kUSB_UsbResetEvent);
             if (USB_connectionInfo() & kUSB_vbusPresent){
                   USB_handleVbusOnEvent();
             }
             initBluetooth();
        }

        //
        // Measurement starts
        //
        gotoXy(0,0);
        prints("   MEASURING    ");
        gotoXy(0,1);
        prints("  PLEASE WAIT   ");
        __enable_interrupt();                 //Enable interrupts globally
        __bis_SR_register(LPM3_bits + GIE);  // Enter LPM3, enable interrupts
        __no_operation();                     // For debugger

        seconds=0;
        while (seconds<count)
        {
             n++;
             if (readDataFlag)
             {
             readDataFlag = 0;
             sampleCount++;
             if (sampleCount == totalCount)
             {
                   sampleCount = 0;
                   totalCount = 1;
                   Disable_AFE44xx_DRDY_Interrupt();
                   //P2OUT &= ~BIT0;   //Turn off LED P5.0 (Green)
             }
             AFE44xx_SPO2_Data_buf[0] = AFE44xx_Reg_Read(42);  //read RED Data
             AFE44xx_SPO2_Data_buf[1] = AFE44xx_Reg_Read(43);  //read Ambient data
             AFE44xx_SPO2_Data_buf[2] = AFE44xx_Reg_Read(44);  //read IR Data
             AFE44xx_SPO2_Data_buf[3] = AFE44xx_Reg_Read(45);  //read Ambient Data
             AFE44xx_SPO2_Data_buf[4] = AFE44xx_Reg_Read(46);  //read RED - Ambient Data
```

```
AFE44xx_SPO2_Data_buf[5] = AFE44xx_Reg_Read(47);  //read IR - Ambient Data
sendDataFlag = 1;
}
if (sendDataFlag)
{
sendDataFlag = 0;
f_write(&file, &AFE44xx_SPO2_Data_buf, 48, &bytesWritten);
if(transmit == 0 || transmit == 2){
txString[0]=(unsigned char) START_READ_ADC_REG_CMD;
txString[1]=(unsigned char) SOT;
txString[2]=(unsigned char)(AFE44xx_SPO2_Data_buf[0] & 0x000000FF);
txString[3]=(unsigned char)((AFE44xx_SPO2_Data_buf[0] & 0x0000FF00) >> 8);
txString[4]=(unsigned char)((AFE44xx_SPO2_Data_buf[0] & 0x00FF0000) >> 16);
txString[5]=(unsigned char)(AFE44xx_SPO2_Data_buf[1] & 0x000000FF);
txString[6]=(unsigned char)((AFE44xx_SPO2_Data_buf[1] & 0x0000FF00) >> 8);
txString[7]=(unsigned char)((AFE44xx_SPO2_Data_buf[1] & 0x00FF0000) >> 16);
txString[8]=(unsigned char)(AFE44xx_SPO2_Data_buf[2] & 0x000000FF);
txString[9]=(unsigned char)((AFE44xx_SPO2_Data_buf[2] & 0x0000FF00) >> 8);
txString[10]=(unsigned char)((AFE44xx_SPO2_Data_buf[2] & 0x00FF0000)>> 16);
txString[11]=(unsigned char)(AFE44xx_SPO2_Data_buf[3] & 0x000000FF);
txString[12]=(unsigned char)((AFE44xx_SPO2_Data_buf[3] & 0x0000FF00)>> 8);
txString[13]=(unsigned char)((AFE44xx_SPO2_Data_buf[3] & 0x00FF0000)>> 16);
txString[14]=(unsigned char)(AFE44xx_SPO2_Data_buf[4] & 0x000000FF);
txString[15]=(unsigned char)((AFE44xx_SPO2_Data_buf[4] & 0x0000FF00)>> 8);
txString[16]=(unsigned char)((AFE44xx_SPO2_Data_buf[4] & 0x00FF0000)>> 16);
txString[17]=(unsigned char)(AFE44xx_SPO2_Data_buf[5] & 0x000000FF);
txString[18]=(unsigned char)((AFE44xx_SPO2_Data_buf[5] & 0x0000FF00)>> 8);
txString[19]=(unsigned char)((AFE44xx_SPO2_Data_buf[5] & 0x00FF0000)>> 16);
txString[20]=(unsigned char) EOT;
txString[21]=(unsigned char) CR;
// Send the response over USB
cdcSendDataInBackground((BYTE*)txString,22,CDC0_INTFNUM,0);
}
}
if(transmit==0 || transmit==2){
BYTE i;      //Check the USB state and directly main loop accordingly
switch (USB_connectionState())
{
case ST_USB_DISCONNECTED:
//__bis_SR_register(LPM3_bits + GIE);
//Enter LPM3 w/ interrupts enabled
_NOP();                                     //For Debugger
break;
case ST_USB_CONNECTED_NO_ENUM:
break;
case ST_ENUM_ACTIVE:
 //__bis_SR_register(LPM0_bits + GIE);
//Enter LPM0 (can't do LPM3 when active)
_NOP();                                     //For Debugger
//Exit LPM on USB receive and perform a receive operation
if (bCDCDataReceived_event){
//Some data is in the buffer; begin receiving a command
P2OUT |= BIT3;                               //Turn on LED P5.1 (Blue)
char pieceOfString[MAX_STR_LENGTH] = "";
//Holds the new addition to the string
//char outString[MAX_STR_LENGTH] = "";
//Holds the outgoing string
//Add bytes in USB buffer to theCommand
```

```
            cdcReceiveDataInBuffer((BYTE*)pieceOfString, MAX_STR_LENGTH, CDC0_INTFNUM);
            //Get the next piece of the string
            strcat(wholeString,pieceOfString);
            //cdcSendDataInBackground((BYTE*)pieceOfString,
            //strlen(pieceOfString),CDC0_INTFNUM,0);
            //Echoes back the characters received (needed for Hyperterm)
            if (retInString(wholeString)){         //Has the user pressed return yet?
            if (wholeString[0] == WRITE_REG_CMD) // AFE44xx Write Operation
            {
            AFE44xxAddr = (ascii2uint8 (wholeString[1]) << 4) | ascii2uint8
(wholeString[2]);
            unsigned long AFE44xxRegData[3];
            AFE44xxRegData[0] = (ascii2uint8 (wholeString[3]) << 4) | ascii2uint8
(wholeString[4]);
            AFE44xxRegData[1] = (ascii2uint8 (wholeString[5]) << 4) | ascii2uint8
(wholeString[6]);
            AFE44xxRegData[2] = (ascii2uint8 (wholeString[7]) << 4) | ascii2uint8
(wholeString[8]);
            AFE44xxRegVal = (AFE44xxRegData[0]<<16)| (AFE44xxRegData[1]<<8) |
(AFE44xxRegData[2]);
            AFE44xx_Reg_Write(AFE44xxAddr, AFE44xxRegVal);
            }
            else if (wholeString[0] == READ_REG_CMD) // AFE44xx Read Operation
            {
            AFE44xxAddr = (ascii2uint8 (wholeString[1]) << 4) | ascii2uint8
(wholeString[2]);
            AFE44xxRegVal = AFE44xx_Reg_Read(AFE44xxAddr);
            txString[0] = (unsigned char) READ_REG_CMD;
            txString[1] = (unsigned char) SOT;
            txString[2] = (unsigned char)(AFE44xxRegVal & 0x000000FF);
            txString[3] = (unsigned char)((AFE44xxRegVal & 0x0000FF00) >> 8);
            txString[4] = (unsigned char)((AFE44xxRegVal & 0x00FF0000) >> 16);
            txString[5] = (unsigned char) EOT;
            txString[6] = (unsigned char) CR;
            // Send the response over USB
            cdcSendDataInBackground((BYTE*)txString,7,CDC0_INTFNUM,0);
            }
            else if (wholeString[0] == START_READ_ADC_REG_CMD)
            // Start AFE44x0 ADC Reg Read Command
            {
            totalCount = 1;
            sampleCount = 0;
            for (i = 0; i < ((ascii2uint8 (wholeString[2]) << 4) | ascii2uint8
(wholeString[3])); i++)
            {
            totalCount *= 2;
            }
            Enable_AFE44xx_DRDY_Interrupt();             // Enable DRDY interrupt
            readDataFlag = 0;
            sendDataFlag = 0;
            AFE44xx_SPO2_Data_buf[0] = 0;
            AFE44xx_SPO2_Data_buf[1] = 0;
            AFE44xx_SPO2_Data_buf[2] = 0;
            AFE44xx_SPO2_Data_buf[3] = 0;
            AFE44xx_SPO2_Data_buf[4] = 0;
            AFE44xx_SPO2_Data_buf[5] = 0;
            }
            else if (wholeString[0] == STOP_READ_ADC_REG_CMD)
```

```
// Stop AFE44x0 ADC Reg Read Command
{
sampleCount = 0;
totalCount = 1;
Disable_AFE44xx_DRDY_Interrupt();
readDataFlag = 0;
sendDataFlag = 0;
//P2OUT &= ~BIT0;                                 //Turn off LED P5.0 (Green)
}
else if (wholeString[0] == FW_UPGRADE_CMD)     // Firmware Upgrade Command
{
USB_disable();                                  // Disable CDC connection
TA1CTL &= ~MC_1;                                // Turn off Timer
Disable_AFE44xx_DRDY_Interrupt();               // Disable interrupt
readDataFlag = 0;                               // Disable Read Data flag
sendDataFlag = 0;                               // Disable send Data flag
__delay_cycles(200);                            // Delay
((void (*)())0x1000)();                         // Go to BSL
}
else if (wholeString[0] == DEV_ID_CMD)         // Dev ID Command
{
txString[0] = (unsigned char)DEV_ID_CMD;
txString[1] = (unsigned char)SOT;
txString[2] = 0x34;
txString[3] = 0x34;
txString[4] = 0x39;
txString[5] = 0x30;
txString[6] = (unsigned char)EOT;
txString[7] = (unsigned char)CR;
cdcSendDataInBackground((BYTE*)txString,8,CDC0_INTFNUM,0);
// Send the response over USB
}
else if (wholeString[0] == FW_VERSION_CMD) // FW Version Command
{
txString[0] = (unsigned char)FW_VERSION_CMD;
txString[1] = (unsigned char)SOT;
txString[2] = (unsigned char)AFE44x0_Major_Number;
txString[3] = (unsigned char)AFE44x0_Minor_Number;
txString[4] = (unsigned char)EOT;
txString[5] = (unsigned char)CR;
cdcSendDataInBackground((BYTE*)txString,6,CDC0_INTFNUM,0);
// Send the response over USB
}
for (i = 0; i < MAX_STR_LENGTH; i++){
//Clear the string in preparation for the next one
wholeString[i] = 0x00;
}
P2OUT &= ~BIT3;                                 //Turn off LED P5.1 (Blue)
}
bCDCDataReceived_event = FALSE;
}
break;
case ST_ENUM_SUSPENDED:
P2OUT &= ~BIT0;                                 //When suspended, turn off LED
//__bis_SR_register(LPM3_bits + GIE);
//Enter LPM3 w/ interrupts
_NOP();
break;
```

```
            case ST_ENUM_IN_PROGRESS:
            break;
            case ST_NOENUM_SUSPENDED:
            P5OUT &= ~BIT0;
            //__bis_SR_register(LPM3_bits + GIE);
            _NOP();
            break;
            case ST_ERROR:
            _NOP();
            break;
            default:;
            } //switch
            }//if USB
        }//while(seconds<count)

        //
        // Calculates result
        //
        gotoXy(0,0);
        prints("  CALCULATING   ");
        gotoXy(0,1);
        prints("  PLEASE WAIT   ");
        if(measure==0)
            measureSpO2();
        else
            measureStress();

    } //else
} //main()

/*
 * ======== Init_Clock ========
 */
VOID Init_Clock (VOID)
{
    //Initialization of clock module
    if (USB_PLL_XT == 2){
            #if defined (__MSP430F552x) || defined (__MSP430F550x)
                  P5SEL |= 0x0C;                            //enable XT2 pins for F5529
            #elif defined (__MSP430F563x_F663x)
                  P7SEL |= 0x0C;
            #endif

        //use REFO for FLL and ACLK
        UCSCTL3 = (UCSCTL3 & ~(SELREF_7)) | (SELREF__REFOCLK);
        UCSCTL4 = (UCSCTL4 & ~(SELA_7)) | (SELA__REFOCLK);

        //MCLK will be driven by the FLL (not by XT2), referenced to the REFO
        Init_FLL_Settle(USB_MCLK_FREQ / 1000, USB_MCLK_FREQ / 32768);
       //Start the FLL, at the freq indicated by the config
       //constant USB_MCLK_FREQ
        XT2_Start(XT2DRIVE_2);                              //Start the "USB crystal"
    }
      else {
            #if defined (__MSP430F552x) || defined (__MSP430F550x)
                  P5SEL |= 0x10;                                        //enable XT1 pins
            #endif
        //Use the REFO oscillator to source the FLL and ACLK
```

```
        UCSCTL3 = SELREF__REFOCLK;
        UCSCTL4 = (UCSCTL4 & ~(SELA_7)) | (SELA__REFOCLK);

        //MCLK will be driven by the FLL (not by XT2), referenced to the REFO
        Init_FLL_Settle(USB_MCLK_FREQ / 1000, USB_MCLK_FREQ / 32768); //set FLL (DCOCLK)

        XT1_Start(XT1DRIVE_0);                          //Start the "USB crystal"
    }
}

/*
 * ======== Init_Ports ========
 */
VOID Init_Ports (VOID)
{
    //Initialization of ports (all unused pins as outputs with low-level
    P1OUT = 0xFF;
    P1DIR = 0xFF;
    P2OUT = 0xFF;
    P2DIR = 0xFF;
    P3OUT = 0x00;
    P3DIR = 0xFF;
    P4OUT = 0x00;
    P4DIR = 0xFF;
    P5OUT = 0x00;
    P5DIR = 0xFF;
    P6OUT = 0x00;
    P6DIR = 0xFF;
    P7OUT = 0x00;
    P7DIR = 0xFF;
    P8OUT = 0x00;
    P8DIR = 0xFF;
}

/*
 * ======== UNMI_ISR ========
 */
#pragma vector = UNMI_VECTOR
__interrupt VOID UNMI_ISR (VOID)
{
    switch (__even_in_range(SYSUNIV, SYSUNIV_BUSIFG))
    {
        case SYSUNIV_NONE:
            __no_operation();
            break;
        case SYSUNIV_NMIIFG:
            __no_operation();
            break;
        case SYSUNIV_OFIFG:
            UCSCTL7 &= ~(DCOFFG + XT1LFOFFG + XT2OFFG); //Clear OSC flaut Flags fault
flags
            SFRIFG1 &= ~OFIFG;                          //Clear OFIFG fault flag
            break;
        case SYSUNIV_ACCVIFG:
            __no_operation();
            break;
        case SYSUNIV_BUSIFG:
```

```
            //If bus error occured - the cleaning of flag and re-initializing of
                                                                //USB is required.
            SYSBERRIV = 0;                                      //clear bus error flag
            USB_disable();                                      //Disable
    }
}

/*
 * ======== Init_TimerA1 ========
 */
VOID Init_TimerA1 (VOID)
{
    //TA1CCTL0 = CCIE;                                  //CCR0 interrupt enabled
    //TA1CTL = TASSEL_1 + TACLR;                        //ACLK, clear TAR
}

/*
 * ======== Init_TimerB ========
 */
VOID Init_TimerB (VOID)
{
      TBCCTL0 = CCIE;                                   // TRCCR0 interrupt enabled
      TBCCR0 = 16384-1;
      TBCTL = TBSSEL_1 + MC_1 + TBCLR;                  // ACLK, upmode, clear TBR
}

/*
 * ======== retInString ========
 */
//This function returns true if there's an 0x0D character in the string; and if so,
//it trims the 0x0D and anything that had followed it.
BYTE retInString (char* string)
{
    BYTE retPos = 0,i,len;
    char tempStr[MAX_STR_LENGTH] = "";

    strncpy(tempStr,string,strlen(string));             //Make a copy of the string
    len = strlen(tempStr);
    while ((tempStr[retPos] != 0x0A) && (tempStr[retPos] != 0x0D) && (retPos++ < len)) ;
    //Find 0x0D; if not found, retPos ends up at len

    if ((retPos < len) && (tempStr[retPos] == 0x0D)){   //If 0x0D was actually found...
        for (i = 0; i < MAX_STR_LENGTH; i++){           //Empty the buffer
            string[i] = 0x00;
        }
        strncpy(string,tempStr,retPos);//...trim the input string to just before 0x0D
        return ( TRUE) ;               //...and tell the calling function that we did so
    } else if ((retPos < len) && (tempStr[retPos] == 0x0A)){
      //If 0x0D was actually found...
        for (i = 0; i < MAX_STR_LENGTH; i++){                   //Empty the buffer
            string[i] = 0x00;
        }
        strncpy(string,tempStr,retPos);//...trim the input string to just before 0x0D
        return ( TRUE) ;               //...and tell the calling function that we did so
    } else if (tempStr[retPos] == 0x0D){
        for (i = 0; i < MAX_STR_LENGTH; i++){                   //Empty the buffer
            string[i] = 0x00;
        }
```

```
        strncpy(string,tempStr,retPos);//...trim the input string to just before 0x0D
        return ( TRUE) ;               //...and tell the calling function that we did so
    } else if (retPos < len){
        for (i = 0; i < MAX_STR_LENGTH; i++){                  //Empty the buffer
            string[i] = 0x00;
        }
        strncpy(string,tempStr,retPos);//...trim the input string to just before 0x0D
        return ( TRUE) ;               //...and tell the calling function that we did so
    }

    return ( FALSE) ;                  //Otherwise, it wasn't found
}

/*
 * ======== measureSpO2 ========
 */
VOID measureSpO2(VOID){

      unsigned long redMean=0;
      unsigned long irMean=0;
      unsigned long redRMS=0;
      unsigned long irRMS=0;
      int m, SPO2, bpm;
      uint16_t integer, fraction;

      for(m=0;m<n;m++){
            f_read(&file, &AFE44xx_SPO2_Data_buf, 48, &bytesRead);
            redMean+=AFE44xx_SPO2_Data_buf[0]/n;
            irMean+=AFE44xx_SPO2_Data_buf[2]/n;
            redRMS+=(AFE44xx_SPO2_Data_buf[0]^2)/n;
            irRMS+=(AFE44xx_SPO2_Data_buf[2]^2)/n;
      }

      redRMS=sqrt(redRMS);
      irRMS=sqrt(irRMS);

      SPO2=100-25*((redRMS/redMean)/(irRMS/irMean));
      bpm=60/mean(peakDetect(&file,n),bytesRead);

      integer=SPO2 >> 8;
      //Descompone el valor de SpO2 en parte entera y decimal para representarlo por
      pantalla
      fraction=((SPO2 >> 4)%16)*625;
      gotoXy(0,0);
      prints("  ");
      gotoXy(2,0);
      integerToLcd(integer);
      gotoXy(4,0);
      prints(".");
      gotoXy(5,0);
      integerToLcd(fraction);
      gotoXy(6,0);
      prints(" SpO2 %  ");
      integer=bpm >> 8;
      //Describe los latidos por minuto como un número entero
      gotoXy(0,1);
      prints("  ");
      gotoXy(2,0);
```

```
	integerToLcd3(integer);
	gotoXy(6,0);
	prints(" bpm       ");
}

/*
 * ======= measureStress =======
 */
VOID measureStress(VOID){

	int stressed=0;
	unsigned long SDNN, entropia;
	unsigned long * hrv;

	hrv=peakDetect(&file,n);

	SDNN=std(hrv,n);
	entropia=Apen(0.2*SDNN,hrv,n);

	if(entropia>0.45 || entropia<1.7)
		stressed=1;

	if(stressed==1){
		gotoXy(0,0);
		prints("    STRESSED    ");
		gotoXy(1,0);
		prints("                ");
	}
	else{
		gotoXy(0,0);
		prints("  NOT STRESSED  ");
		gotoXy(1,0);
		prints("                ");
	}
}

/*
 * ======== TIMER1_A0_ISR ========
 */
#pragma vector=TIMER1_A0_VECTOR
__interrupt void TIMER1_A0_ISR (void)
{
}

/*
 * ======== TIMER1_B1_ISR ========
 */
#pragma vector=TIMERB0_VECTOR
__interrupt void TIMERB1_ISR(void)
{
    seconds = seconds+1;;                          // Increments seconds value
}

// Port 2 interrupt service routine
#pragma vector=PORT2_VECTOR  //DRDY interrupt
__interrupt void Port_2(void)
{
  switch (P2IV)
```

```
  {
  case P2IV_P2IFG0:
    P2IFG &= ~BIT3;                         // Clear P2.3 IFG i.e Data RDY interrupt status
    //P2OUT |= BIT0;                        //Turn on LED P5.0 (Green)
    readDataFlag = 1;                       // Set Flag to read AFE44x0 ADC REG data
    break;

  case P2IV_NONE:
    break;
  }
}

unsigned char ascii2uint8 (unsigned char asciiVal)
{
      unsigned char uint8Val;

      if (asciiVal > 0x60 && asciiVal < 0x67)           // 'a' to 'f'
            uint8Val = ((asciiVal - 0x61) + 0x0A);
      else if (asciiVal > 0x40 && asciiVal < 0x47)      // 'A' to 'F'
            uint8Val = ((asciiVal - 0x41) + 0x0A);
      else if (asciiVal > 0x2F && asciiVal < 0x3A)      // '0' to '9'
            uint8Val = (asciiVal - 0x30);
      else                                              // others
            uint8Val = 0;

      return uint8Val;
}

void fat_init(void){
    errCode = -1;

    while (errCode != FR_OK){     //go until f_open returns FR_OK (function successful)
        errCode = f_mount(0, &fatfs);                       //mount drive number 0
        errCode = f_opendir(&dir, "/");                             //root directory

        errCode = f_open(&file, "/teste.txt", FA_CREATE_ALWAYS | FA_WRITE);
        if(errCode != FR_OK)
            result=0;                                       //used as a debugging flag
    }
}

void initCLK(void){
  volatile unsigned int i;

  // Increase Vcore setting to level3 to support fsystem=25MHz
  // NOTE: Change core voltage one level at a time..
  SetVcoreUp (0x01);
  SetVcoreUp (0x02);
  SetVcoreUp (0x03);

  UCSCTL3 = SELREF_2;                       // Set DCO FLL reference = REFO
  UCSCTL4 |= SELA_2;                        // Set ACLK = REFO

  __bis_SR_register(SCG0);                  // Disable the FLL control loop
  UCSCTL0 = 0x0000;                         // Set lowest possible DCOx, MODx
  UCSCTL1 = DCORSEL_7;                      // Select DCO range 50MHz operation
  UCSCTL2 = FLLD_1 + 762;                   // Set DCO Multiplier for 25MHz
                                            // (N + 1) * FLLRef = Fdco
```

```
                                          // (762 + 1) * 32768 = 25MHz
                                          // Set FLL Div = fDCOCLK/2
  __bic_SR_register(SCG0);                // Enable the FLL control loop

  // Worst-case settling time for the DCO when the DCO range bits have been
  // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
  // UG for optimization.
  // 32 x 32 x 25 MHz / 32,768 Hz ~ 780k MCLK cycles for DCO to settle
  __delay_cycles(782000);

  // Loop until XT1,XT2 & DCO stabilizes - In this case only DCO has to stabilize
  do
  {
    UCSCTL7 &= ~(XT2OFFG + XT1LFOFFG + DCOFFG);
                                          // Clear XT2,XT1,DCO fault flags
    SFRIFG1 &= ~OFIFG;                    // Clear fault flags
  }while (SFRIFG1&OFIFG);                 // Test oscillator fault flag
}

void SetVcoreUp (unsigned int level)
{
  // Open PMM registers for write
  PMMCTL0_H = PMMPW_H;
  // Set SVS/SVM high side new level
  SVSMHCTL = SVSHE + SVSHRVL0 * level + SVMHE + SVSMHRRL0 * level;
  // Set SVM low side to new level
  SVSMLCTL = SVSLE + SVMLE + SVSMLRRL0 * level;
  // Wait till SVM is settled
  while ((PMMIFG & SVSMLDLYIFG) == 0);
  // Clear already set flags
  PMMIFG &= ~(SVMLVLRIFG + SVMLIFG);
  // Set VCore to new level
  PMMCTL0_L = PMMCOREV0 * level;
  // Wait till new level reached
  if ((PMMIFG & SVMLIFG))
    while ((PMMIFG & SVMLVLRIFG) == 0);
  // Set SVS/SVM low side to new level
  SVSMLCTL = SVSLE + SVSLRVL0 * level + SVMLE + SVSMLRRL0 * level;
  // Lock PMM registers for write access
  PMMCTL0_H = 0x00;
}
```

Bibliografía

[1] Diseño del algoritmo para la detección del estrés a partir del electrocardiograma. Carlos Chilla García.

[2] Bioelectrónica, señales bioeléctricas. José M. Ferrero Corral.

[3] Rapid interpretation of EKG's. Dale Dubin.

[4] On the analysis of fingertip photoplethysmogram signals. Mohamed Elgendi.

[5] Heart rate extraction from the photoplethysmogram. Tsu-Hsun Fu, Shing-Hong Liu, Kuo-Tai Tang.

[6] Can photoplethysmography variability serve as an alternative approach to obtein heart rate variability information. Sheng Lu, He Zhao, Kihwan Ju, Kunsoo Shin, Myoungho Lee, Kirk Shelley, Ki H. Chon.

[7] Artifact reduction based on empirical mode decomposition (EMD). Qian Wang, Ping Yang, Yuanting Zhang.

[8] Photoplethysmography pulse rate variability as a surrogate measurement of heart rate variability during non-stationary conditions. E. Gil, M. Orini, R. Bailón, J.M. Vergara, L. Mainardi, P. Laguna.

[9] Changes in the photoplethysmogram waveform after exercise. C.C.Y. Poon, X.F. Teng, Y.M. Wong, C. Zhang, Y.T. Zhang.

[10] Interchangeablity between heart rate and photoplethysmography variabilities during sympathetic stimulations. K. Charlot, J. Cornolo, J.V. Brugniaux, J.P. Richalet, A. Pichon.

[11] Health care guide. Texas Instruments.

[12] AFE4490. Texas Instruments.

[13] Staying well grounded. Hank Zumbahlen.

[14] AFE4400 and AFE4490 development guide. Texas Instruments.

[15] SpO pulse ox wrist reference design. Texas Instrumens.

[16] LCD FDCC1602B. Fordata Electronic.

[17] USB type mini-B 548190572. Molex.

[18] TPD4E004. Texas Instruments.

[19] System level ESD/EMI protection guide. Texas Instruments.

[20] SimpleLink Bluetooth CC256x devices. Texas Instruments.

[21] LMX9838. Texas Instruments.

[22] LMX9838 dongle schematic. Texas Instruments.

[23] Battery management solutions. Texas Instruments.

[24] BQ24072. Texas Instruments.

[25] BQ27200. Texas Instruments.

[26] Power-path Li-Ion charger section. Texas Instruments.

[27] Battery charger termination issues with system load applied across battery while charging. Texas Instruments.

[28] Micro SD 0473090265. Molex.

[29] TPD8E003. Texas Instruments.

[30] Low dropout regulators, quick reference guide. Texas Instruments.

[31] TPS61093. Texas Instruments.

[32] LP3878-ADJ. Texas Instruments.

[33] TPS7A4901. Texas Instruments.

[34] MSP ultra-low-power microcontrollers.Texas Instruments.

[35] MSP430F5529. Texas Instruments.

[36] Printed circuits design featuring computer-aided technologies. Gerald L. Ginsberg.

[37] MSP430 workshop, student guide. Texas Instruments.

[38] International standard 11073-10404. ISO/IEEE.

[39] Multiparameter intelligent monitoring in intensive care II (MIMIC-II), a public-access ICU database. M. Saeed, M. Villarroel, A.T. Reisner, G. Clifford, L. Lehman, G.B. Moody, T. Heldt, T.H. Kyaw, B. E. Moody, R.G. Mark.

[40] PhysioBank, PhysioToolkit and PhysioNet, components of a new research resource for complex physiologic signals. Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PCh, Mark RG, Mietus JE, Moody GB, Peng C-K, Stanley HE.

[41] Nellcor DS-100A. Covidien.

[42] Gauray. www.circuitvalley.com

[43] Bhargavi Nisarga. Texas Instruments.

[44] Praveel Aroul. Texas Instruments.

[45] Application note: Secure Digital card interface for MSP430. F. Foust.

[46] Lee R Wallace. University of Florida.

[47] Interfacing a MSP430 with an SD Card. University of Florida.

[48] Calculation of SpO2 using PPG signals. Praveen Aroul.

[49] A real time analysis of PPG signal for measurement of SpO2 and pulse rate. Sangeeta Bagha, Laxmi Shaw.