UNED ETS de Ingeniería Informática

MÁSTER EN INGENIERÍA DE SISTEMAS Y CONTROL

Trabajo fin de Máster

Desarrollo y refinamiento de un prototipo de vestible para la adquisición de datos de presión plantar en evaluaciones "long-term"

Constantin Cosmin Dobrescu

Director: **José Antonio Cerrada Somolinos**
Codirector: **Iván González Díaz**

Curso 20-2021
Convocatoria Febrero

Máster en Ingeniería de Sistemas y Control

Trabajo fin de Máster

Desarrollo y refinamiento de un prototipo de vestible para la adquisición de datos de presión plantar en evaluaciones "long-term"

TFM Tipo B: Trabajo específico propuesto por el alumno.

Alumno: **Constantin Cosmin Dobrescu**

Director: **José Antonio Cerrada Somolinos**

Codirector: **Iván González Díaz**

Autorización

Autorizamos a la Universidad Complutense y a la UNED a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Firmado: José Antonio Cerrada

Dpto: Ingeniería de Software y Sistemas Informáticos

UNED

Firma del alumno

DOBRESCU --- CONSTANTIN COSMIN - X8810718Q

Firmado digitalmente por DOBRESCU --- CONSTANTIN COSMIN - X8810718Q
Fecha: 2021.02.11 20:14:15 +01'00'

Resumen

En este Trabajo Fin de Máster se plantea una serie de objetivos para poder desarrollar un prototipo de tecnología vestible en forma de plantilla sensorizada. Se realiza una nueva iteración sobre un prototipo inicial desarrollado en el marco de un Trabajo Fin de Grado. Este primer prototipo tenía como objetivo principal la captura de datos en tiempo real utilizando la tecnología "Bluetooth Low Energy". Disponía de 8 sensores de presión y una Unidad de Medición Inercial IMU que captura tanto la aceleración como la velocidad angular en los 3 ejes del espacio.

Con la experiencia adquirida en el desarrollo del primer prototipo, se ha llegado a la conclusión que es más conveniente orientar la plantilla a una nueva estrategia de captura de datos a largo plazo. Este cambio tiene dos motivos principales: mejorar la autonomía y aumentar la frecuencia de adquisición de datos. Realizar estas modificaciones implica rediseñar tanto el hardware como el software de la plantilla. En la parte hardware se han aumentado el número de sensores de presión de 12 sensores a un total de 32. Se añade un nuevo circuito integrado (IC) capaz de generar una marca temporal (RTC real-time clock), una memoria FLASH y nuevos componentes que gestionan el proceso de carga de la batería. El software se divide en dos partes, aquel que se ejecutara en la plantilla y el que se ejecutara en un dispositivo Android.

El objetivo principal de este tipo de tecnología vestible es ser utilizada para el análisis cuantitativo de la marcha humana. Para poder demostrar la utilidad de esta tecnología y justificar este desarrollo, en este trabajo se implementa un clasificador SVM para delimitar eventos de la marcha humana utilizando los datos de presión plantar y los generados por la unidad de medición inercial que dispone la plantilla.

Palabras Clave:

PSoC 4, IoT, Insole, Plantilla Sensorizada, sensores de presión, IMU, BLE, Bluetooth Low Energy, Memoria FLASH SPI, MOAA, clasificador SVM.

Lista de acrónimos

- **TFG**: Trabajo Fin de Grado
- **TFM**: Trabajo Fin de Máster
- **BLE**: Bluetooth Low Energy
- **L2CAP**: Logical Link Control and Adaptation Protocol
- **PSM**: Protocol/Service Multiplexers
- **MOAA**: Multiplexer and op-amp assisted approach
- **Op-Amp**: Operational Amplifier
- **BCD**: Binary-coded decimal
- **IMU**: Inertial Measurement Unit
- **RTC**: Real-Time Clock
- **API**: Application Programming Interface
- **USB**: Universal Serial Bus
- **MOSFET**: Metal-Oxide-Semiconductor Field-Effect Transistor
- **ADC**: Analog-to-Digital Conversion
- **DPS**: Degrees Per Second
- **SPI**: Serial Peripheral Interface
- **I2C**: Inter-Integrated Circuit
- **PSoC**: Programable System on Chip
- **SoC**: System on Chip
- **EVI**: External Event Interrupt
- **ECP**: Event Capture Enable
- **EIE**: External Event Interrupt Enable
- **EHL**: Event High/Low detection Select
- **EVF**: External Event Flag
- **DMA**: Direct Memory Access
- **HS**: Heel Strike
- **TO**: Toe Off
- **GAP**: Generic Access Profile
- **GATT**: Generic Attribute Profile
- **ATT**: Attribute Protocol
- **SM**: Security Manager

Índice

Lista de Figuras y Tablas:

1. Introducción

A medida que avanza la miniaturización de la tecnología de procesamiento, da paso a la creación de nuevos dispositivos que hace varias décadas serían impensables. Gracias a la mejora constante de la tecnología se han creado los dispositivos vestibles. Estos dispositivos electrónicos normalmente son pequeños y pueden ir en diversas partes del cuerpo dentro o incluso en el interior de la ropa/accesorio, por ejemplo: zapatos, gafas, anillos, guantes, relojes o pendientes entre otros.

En la última década la tecnología vestible está tomando un papel cada vez más importante dentro del ámbito sanitario. Este tipo de dispositivos tiene como objetivo la mejora de la salud y el bienestar general de las personas en todo el mundo. En este momento se están desarrollando multitud de dispositivos que podrían revolucionar la atención sanitaria de diferentes formas. Algunos ejemplos pueden ser los diferentes dispositivos de monitorización de la glucosa de manera no invasiva [1] o las capsulas médicas que ayudan a realizar una endoscopia no invasiva en pacientes [2] [3].

Gracias a los grandes avances en medicina en las últimas décadas, la esperanza de vida ha aumentado paulatinamente hasta llegar a 80 y 86 años en hombres y mujeres respectivamente [4]. Este envejecimiento de la población aumenta la necesidad de una mayor atención médico-sanitaria para tratar los diversos problemas derivados. Junto con el envejecimiento aumenta de las enfermedades crónicas, ya que, en 2005, todas las enfermedades crónicas representan el 72% del total de las enfermedades en la población mayor de 30 años [5].

Para hacer frente a los problemas actuales y futuros en el ámbito sanitario, se ha propuesto el modelo mHealth del inglés “mobile health”. Aunque no exista una definición estandarizada, se podría definir como el ejercicio de integración de la tecnología móvil y vestible, comunicación inalámbrica y los servicios en la nube para mejorar la investigación en el campo de la salud y atención sanitaria. Este modelo tiene el potencial de reducir el coste de la atención sanitaria y mejorar en gran medida el bienestar de las personas.

Los dispositivos englobados en el modelo mHealth pueden facilitar el seguimiento continuo de la salud tanto a nivel individual como a niveles de una mayor población; Fomentar el comportamiento saludable para prevenir futuros problemas; mejorar el conocimiento del personal médico sobre el paciente; reducir el número de visitas y así descongestionar los centros de salud [6]; o ofrecer soluciones a problemas de salud mucho más personalizadas utilizando la tecnología móvil que los usuarios ya dispone como los teléfonos inteligentes [7] [8] [9].

Uno de los dispositivos que lleva varios años en crecimiento en la tecnología vestible son las plantillas sensorizadas. En el mercado actual existen algunos dispositivos ya desarrollados pero muy pocos que se puedan comercializar y prácticamente ninguno que tenga un precio asequible de adquisición. Algunos ejemplos son las plantillas propuestas por la empresa MOTICON que ofrecen las plantillas entre 3500 y 4000€ dependiendo del plan elegido. Otros ejemplos son las plantillas fabricadas por Tekscan denominadas “F-SCAN”. Estas disponen de un gran número de sensores de presión en comparación con la anterior (hasta 960 sensores) que sólo tienen 13. Pero son mucho más intrusivas y el precio de la PCB sin electrónica ronda los 1000€ [10].

Las plantillas sensorizadas se pueden evaluar en función de las características de éstas en los siguientes aspectos:

- **Número de sensores de presión**: Dependiendo del uso que se les darán a las plantillas es tener más o menos sensores de presión.
- **Unidad de medición Inercial o IMU**: Esta nos puede ofrecer de 3 a 9 datos para la orientación dentro de un espacio tridimensional. Pueden proporcionar la variación en las 3 dimensiones espaciales de la aceleración, el giro aplicado y el campo magnético.
- **Nivel de intrusividad**: Hace referencia a cuanto de fácil es utilizar la plantilla para el usuario final, en este aspecto es muy importante si tiene la electrónica integrada o no.
- **Frecuencia de adquisición**: Al igual que el número de sensores es importante dependiendo del uso la frecuencia de adquisición de los datos.
- **Autonomía**: El tiempo que es capaz de funcionar de manera ininterrumpida la plantilla sin necesitar asistencia.
- **Precio**: Uno de los parámetros mas importante hoy en día de cualquier producto comercial.

Para el análisis espacio-temporal de la marcha humana existen multitud de sistemas desarrollados. Cada uno de ellos tiene su propia metodología, en el caso de Crea et al. [11] consiste en una plantilla de 64 sensores de presión con la electrónica externa a la plantilla. Esta se tiene que atar al calzado utilizado y transmite los datos inalámbricamente a un ordenador.

El prototipo propuesto por Bamberg et al. [12] se puede ver un cambio de filosofía donde no utilizan un excesivo número de sensores de presión resistivos (solo 4) pero añaden tres acelerómetros ortogonales, tres giroscopios ortogonales dos sensores de curvatura, dos sensores de presión piezoeléctricos.

Otros desarrollos importantes en el ámbito han sido los desarrollados por Lopez-Meyer [13]con una plantilla de cinco sensores y Schepers [14] que sensoriza un calzado con dos sensores de fuerza, uno en el talón y otro en la punta junto a dos sensores de inercia de seis ejes.

1.1. Descripción del proyecto

El TFM consistirá en la ampliación, mejora y refinamiento de un prototipo de vestible para la adquisición de datos de presión plantar en evaluaciones “long-term”. Este trabajo se fundamenta en un primer prototipo realizado por mí en el Trabajo Fin de Grado dentro del grupo de investigación MAmI de la Escuela Superior de Informática (UCLM) [15].

Este prototipo de vestible tiene 3 tipos de sensores: de presión; acelerómetro y giróscopo triaxiales. El corazón del prototipo vestible es un SoC[1] desarrollado por Cypress y destaca por su perfil de bajo consumo con una frecuencia máxima de funcionamiento de 48MHz ajustable. Dispone de un amplio número de puertos de entrada/salida y una excelente integración del protocolo Bluetooth Low Energy. Como se podrá ver más adelante todos los componentes auxiliares estarán conectados al SoC Cyble 222014-01.

[1] System on Chip

Antes de entrar en detalle de las modificaciones realizadas primero hay que ver desde dónde partimos. En la primera fase del desarrollo, el prototipo estaba compuesto 4 partes principales:

1. **PCB rígida**: Esta placa de circuito impreso es la encargada de integrar el SoC junto con los componentes auxiliares que este necesita. En la **Figura 1** y la **Figura 2** se puede ver el estado inicial del diseño físico del circuito que alberga al SoC y sus componentes auxiliares. En la **Figura 3** y **Figura 4** se pueden ver esos circuitos ya fabricados y montados.

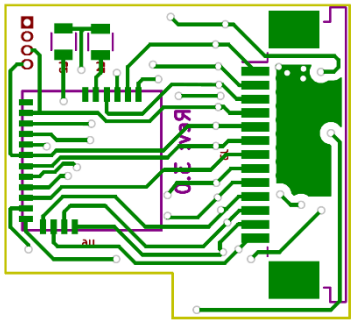


Figura 1: PCB capa inferior.

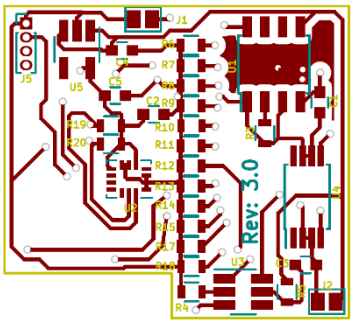


Figura 2: PCB capa superior.

Figura 4:Imagen real de la capa superior

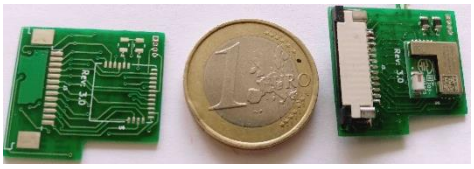


Figura 3: Imagen real de la capa inferior

2. **PCB flexible**: Es la encargada de tener una parte de los sensores de presión que tienen que estar distribuidos en toda la planta del pie. En la
3. **Figura** *5* se puede ver el diseño físico de la PCB. Se puede apreciar (donde está situado el logo "MAmI") una faja que es la que encaja con el conector FPC del circuito anterior y la disposición de una parte de los sensores de presión. Se especifica que es sólo una parte, debido a que la parte de la PCB flexible (por si sola) no puede detectar presión, más adelante se explica el funcionamiento de estos sensores. En la **Figura 6** se puede ver una foto del circuito real.

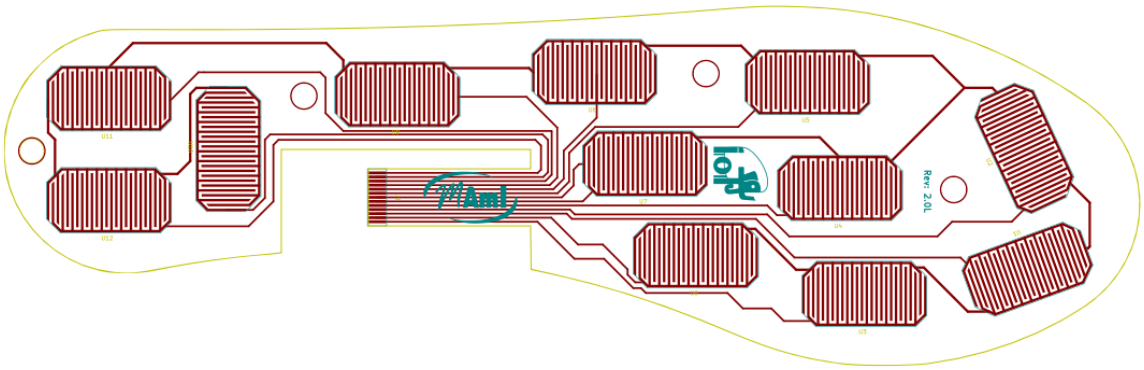


Figura 5 Diseño físico de la PCB

Figura 6: PCB's flexibles pie izquierdo y derecho

4. **Material resistivo**: Es el segundo componente de los sensores de presión, este material es capaz de cambiar su resistencia al paso de corriente dependiendo de la presión realizada. Este material junto con los patrones de pistas entrelazadas que se pueden ver en la
5. **Figura 5** y **Figura 6** crean un sensor de presión. El PSoC[2] junto a un divisor de voltaje, puede llegar a medir la variación en resistencia de este material. En la **Figura 7** se puede ver un esquema de este tipo de sensores. En este caso es un sensor comercial que sigue el mismo patrón descrito anteriormente, pistas entrelazadas con un material que cambia su resistencia en función de la presión aplicada. Además añade una capa de pegamento en medio para tenerlo todo unido.

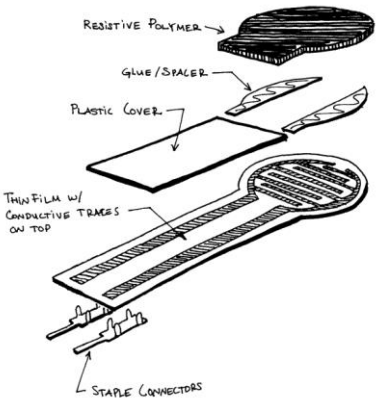


Figura 7: Sensor de presión comercial

6. **Envoltura a las PCB's**: Cada una de las partes anteriores se combinan entre sí mediante diferentes mecanismos. Esta última parte es la encargada de mantener alineadas la parte del material resistivo junto con las dos PCB's. Por otra parte, ofrece una protección a los circuitos debido al elevado peso humano que ejerce un gran estrés sobre estas. En la **Figura 8** se puede parte de la envoltura inferior de material flexible. Además, éste tiene 4 pilares situados en puntos estratégicos para mantener las PCB's centradas y alinear la posterior colocación del material resistivo. Esta envoltura está impresa en 3D con un material flexible.

[2] Programable System on Chip

Figura 8 Montaje parcial de los 4 componentes.

El prototipo presentado anteriormente no se puede utilizar para una monitorización a largo plazo. Este tiene una memoria interna muy limitada, 32KB de RAM y 256KB de FLASH, para almacenar los datos generados por los diferentes sensores. Además, la memoria FLASH interna solamente dispone de 100.000 ciclos de escritura, siendo muy pocos para una memoria tan pequeña. En este trabajo se va a mejorar el hardware general del prototipo y se le va a añadir 2 componentes extra. Uno de ellos es una memoria FLASH donde podrá guardar todos los datos y un chip RTC[3] para proporcionar la hora exacta en la que estos fueron adquiridos.

1.2. Objetivos

El **objetivo principal** del trabajo es adaptar el prototipo de la plantilla sensorizada, para una estrategia de adquisición de datos a largo plazo aumentando el número de sensores. Desarrollando tanto el hardware como el software necesario. Este objetivo principal se podría subdividir en:

- Ampliar a 32 los de sensores de presión de la plantilla.
- Añadir un sistema de carga/descarga, protección y regulación de la batería.
- Añadir un RTC "Reloj en tiempo real".
- Introducir un método que permita almacenar los datos generados.
- Realizar un diseño hardware modular para poder reconexiones componentes si es necesario.
- Mejorar la adquisición de datos de los sensores de presión evitando interferencias entre ellos.
- Mejorar el tiempo de adquisición de los datos generados por los sensores de presión.
- Mejorar la manera en la que se adquiere el nivel de la batería de la plantilla.
- Implementar un método de transferencia de datos masiva desde la plantilla a un dispositivo móvil por Bluetooth.
- Introducir la lógica de funcionamiento de los RTC al firmware de la plantilla.
- Realizar un primer análisis con un clasificador SVM de los datos generados.

[3] Real-Time Clock

2. Mejora del Hardware:

En el diseño de hardware primero se realiza la parte lógica del circuito y posteriormente la parte física. Esta metodología tiene varias ventajas, la primera es poder ignorar la distribución física de los diferentes componentes y preocuparse simplemente que las conexiones sean las adecuadas. Por otra parte, se puede tener varias versiones de un mismo diseño lógico, esto es importante en el desarrollo de prototipos. Al terminar el diseño lógico, se realiza una primera prueba con un circuito sin tomar las dimensiones adecuadas, centrándose sólo en que todos los componentes funcionen. Finalmente, una vez comprobado el diseño, se puede mandar el diseño a una empresa especializada que se encargara de fabricar la PCB según los estándares del mercado.

Esta iteración sobre el prototipo de plantilla sensorizada se centra más en probar el funcionamiento del diseño lógico (con memoria y RTC), permitiendo una gran flexibilidad a la hora de conexionar los componentes. Se añaden muchos más componentes y se hace un cambio de filosofía a la hora de leer los sensores resistivos. Por ello se ha decidido realizar un paso intermedio al producto final y crear una PCB rígida con todos los componentes que debe tener la plantilla, pero esta vez hacerlo en una PCB de desarrollo. Esto quiere decir que se intentara dejar lo más abierto posible los conexionados entre componentes.

2.1. Añadir la memoria Flash y el RTC.

La parte hardware de esta mejora no tiene mayor complicación que conectar de manera correcta cada uno de los chips. El único problema que se ha detectado en el SoC CYBLE 222014-01 son las limitaciones que tiene a la hora de realizar el conexionado. Como ocurre en todos los SoC's del mercado actual, en el documento de características se indican que todos los pines son de entrada/salida. Pero no cualquier pin puede utilizarse para comunicarse por los protocolos I2C[4] y SPI[5]. Este SoC hace parte de la familia de PSoC 4 BLE[6] orientado a un bajo consumo de energía. Por ello tiene muchos menos puertos utilizables y con ello muchas más limitaciones.

Dado que el anterior prototipo tenía una IMU que se comunicaba por el protocolo I2C, lo ideal sería añadir tanto el RTC como la memoria FLASH por el mismo protocolo. Una de las ventajas del protocolo I2C sobre el SPI es que utiliza solamente 2 cables/conexiones para comunicarse con el otro dispositivo. Por un cable se envía la señal de reloj y por el otro los datos. Para poder diferenciar entre uno u otro dispositivo, se utiliza un identificador único. Cuando el "Master", en nuestro caso el SoC, empieza a transmitir lo primero que transmite es la dirección/identificador del chip con el que se quiere comunicar. El inconveniente principal es que no se pueden alcanzar altas velocidades.

Utilizar este protocolo para la memoria FLASH no es posible por 2 razones: el puerto I2C es muy lento y la memoria FLASH de alta densidad no disponen de este protocolo. Si la memoria flash no se puede adaptar el protocolo I2C lo ideal sería que el resto de chips se adaptasen al

[4] Inter-Integrated Circuit
[5] Serial Peripheral Interface
[6] Bluetooth Low Energy

protocolo SPI y así no utilizar los dos protocolos. El objetivo principal de utilizar solamente un protocolo es reducir al máximo el número de puertos utilizados. En este punto entra en juego las limitaciones del propio SoC. Aunque la familia PSOC 4 permite hasta un número de 4 dispositivos SPI en el caso del CYBLE 222014-01 no es posible utilizar más de 1 sin sacrificar los puertos de depuración/programación software.

En teoría se podrían utilizar 2 dispositivos SPI si no se utilizase una función crucial que es la depuración y programación del SoC. Y aun así se tendría que seguir utilizando el protocolo I2C ya que disponemos de 3 dispositivos auxiliares. Esta limitación de número de dispositivos conectados al puerto SPI viene dada por varios factores. El primero de ellos es por la propia arquitectura de programación que ofrece la empresa CYPRESS. Ofrecen un programa llamado PSOC Creator donde parte del desarrollo software se realiza de manera gráfica, ver **Figura 9** , y parte de la manera tradicional con una programación estructurada en C.

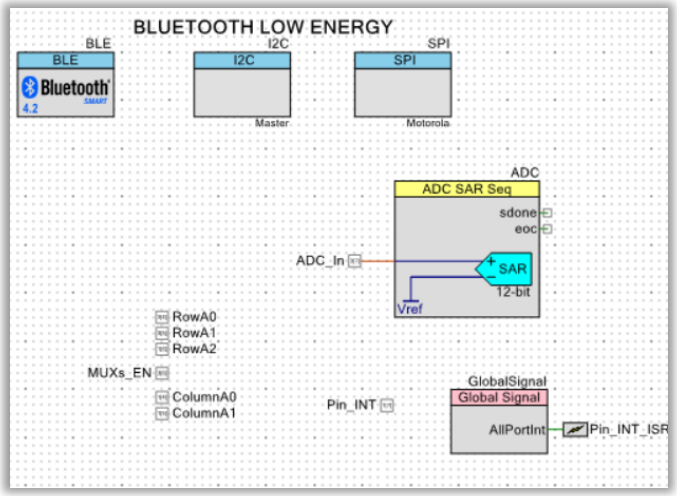


Figura 9 Ejemplo de componentes configurables en el entorno de desarrollo grafico

La parte de desarrollo gráfico utiliza una serie de bloques en los que se añaden diferentes componentes, como el protocolo SPI, I2C, BLE etc. En base a esos bloques genera una API[7] con las llamadas a distintas funciones de los diferentes protocolos. Dentro de este módulo grafico solo permite añadir 4 dispositivos en modo "Slave", ver **Figura 10**.

Aunque aparentemente el Software te permita elegir hasta 4 dispositivos a la hora de enrutar esas funcionalidades a pines físicos, no es posible. Toda esta complejidad viene dada por cómo funciona el protocolo SPI. Para un "*Master*" y un "*Slave*" es necesario 4 cables/conexiones. Una señal de reloj, un canal de transmisión de datos del "*Master*" al "*Slave*", otro viceversa y por último un tercero para seleccionar el "*Slave*" al que se le transmite los datos.

En teoría se podría utilizar un sólo puerto para seleccionar los distintos dispositivos "*Slave*", pero es un proceso mucho más complicado por el entorno de desarrollo proporcionado por Cypress.

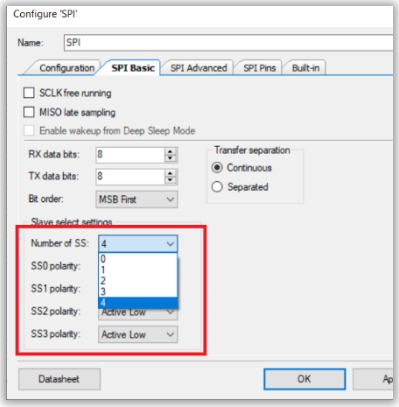


Figura 10 Interfaz para configurar protocolo SPI

[7] API: Application Programming Interfaces, Interfaz de Programación de Aplicaciones

Debido a esta serie de problemáticas se ha tomado la decisión de utilizar conjuntamente el protocolo SPI e I2C. El SPI se utilizará únicamente para la memoria FLASH y el I2C se utilizará para el RTC y la IMU.

Debido a los distintos problemas de consistencia que se han encontrado en el software PSoC Creator se ha decidido crear una PCB de desarrollo. Como anteriormente se ha mencionado la idea de una PCB de desarrollo es dejar lo mas abierto posible el conexionado entre los componetes para poder modificarlos según los requisitos del software.

Ademas de no permitir tantos dispositivos "Slave" como indica, los pines que se pueden dedicar a los protocolos I2C y SPI son muy estrictos. Para entender de que manera trabaja el PSOC Creator, en la **Figura 11** se puede ver la manera en la que se asignan los pines al chips fisico. El propio programa genera una serie de pines según los bloques que has añadido en el diseño grafico y tu tienes que asignarles un pin fisico. En la parte derecha se lista todos los pines que se han configurado de manera grafica y a que pin/puerto se va a asignar. Esta asignación se puede hacer automatica, o personalizada por el desarollador. En la parte izquierda se muestra la posición real que van a tener en el chip.

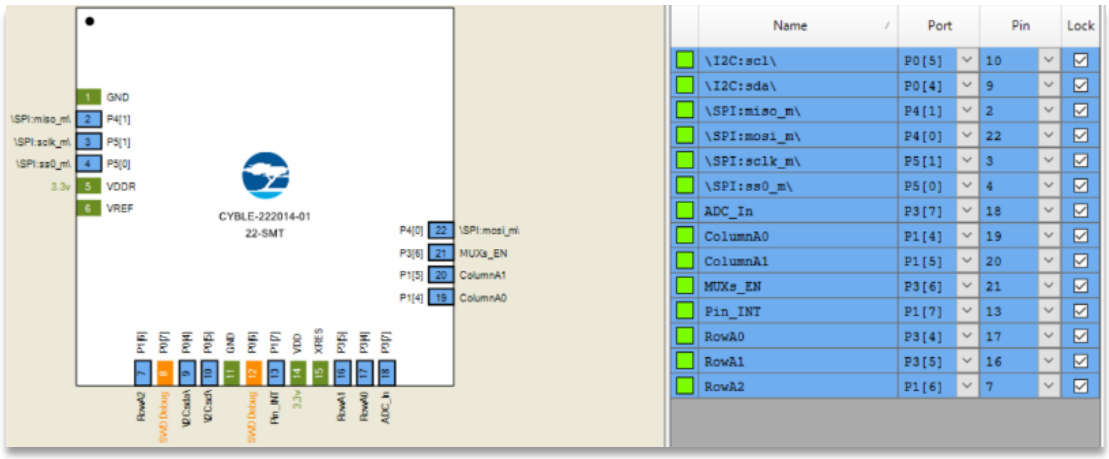


Figura 11 Asignación de pines en PSoC Creator

Una vez explicadas las limitaciones y motivado las decisiones tomadas, en la **Figura 12** se puede ver el diseño lógico del circuito que hace implica a la IMU (ISM330) la memoria FLASH SPI (MKDV2GIL) y el RTC (RV-8803).

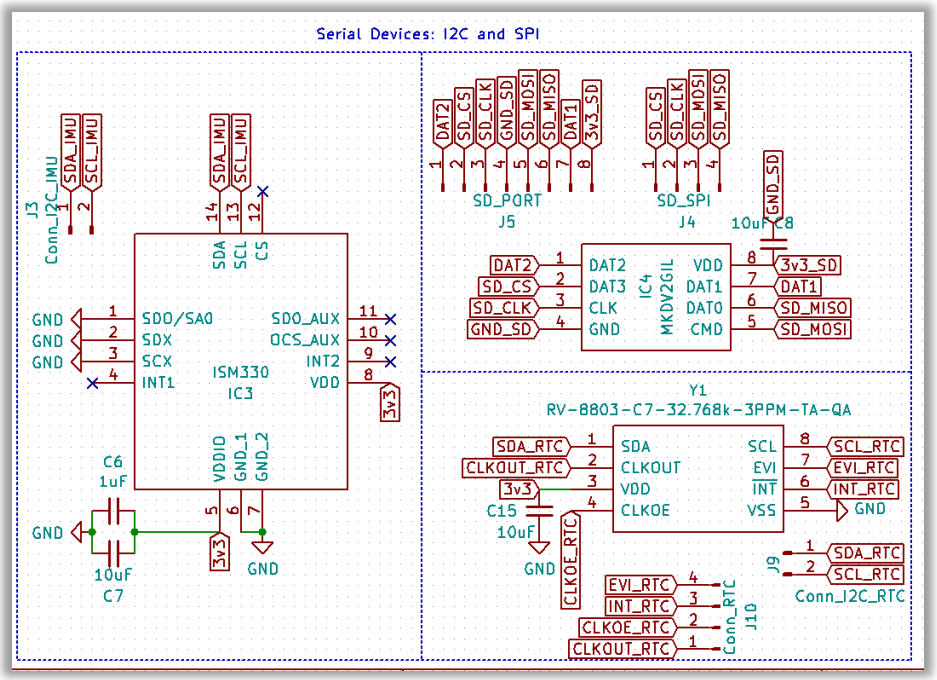


Figura 12 Diseño lógico de la comunicación con IMU, RTC y memoria FLASH

Dado que la IMU ya venía en el anterior prototipo simplemente se conecta las salidas del puerto I2C de la IMU al conector J3 para que posteriormente se pueda conectar al SoC según el enrutado en el Firmware.

El conexionado de la memoria Flash SPI tiene una particularidad. Se han añadido dos conectores, uno de ellos es para el protocolo SPI y el otro es para el protocolo SD standard. El conector SPI se utilizará para conectar con el SoC mientras que el conector SD estándar se utilizara para conectar la tarjeta a un puerto SD de un ordenador, pudiendo obtener los datos en bruto que contiene la memoria.

Por parte del RTC lo único destacable además del conector del protocolo I2C es el conector con las diferentes salidas que este proporciona. Dado que este RTC contiene un reloj interno de alta precisión es capaz de ofrecer una salida con un reloj de 32.768k por su salida "CLKOUT". Un pin de interrupción "INT" que conectaremos al SoC para despertarlo cuando necesitamos obtener datos de los sensores. Otro pin denominado "EVI[8]" que es un puerto de entrada que se utiliza para obtener una marca de tiempo. Por último, un pin "CLKOE" que habilita o no la salida "CLKOUT". En la parte de software se verá más en detalle cómo y para que se utilizan.

[8] External Event Interrupt

2.2. Aumento de sensores

En primera fase se procederá a ampliar el número de sensores de la plantilla de 12 a 32. En la **Figura 13** se pueden apreciar las conexiones de los sensores FSR (etiquetas FSR1 etc…) del prototipo anterior. A simple vista queda claro prácticamente todos los pines están ocupados con estas conexiones. Por tanto, hay que modificar sustancialmente la manera de conectar los 32 sensores. Una de las alternativas más baratas para conectar estos sensores es utilizar multiplexores analógico.

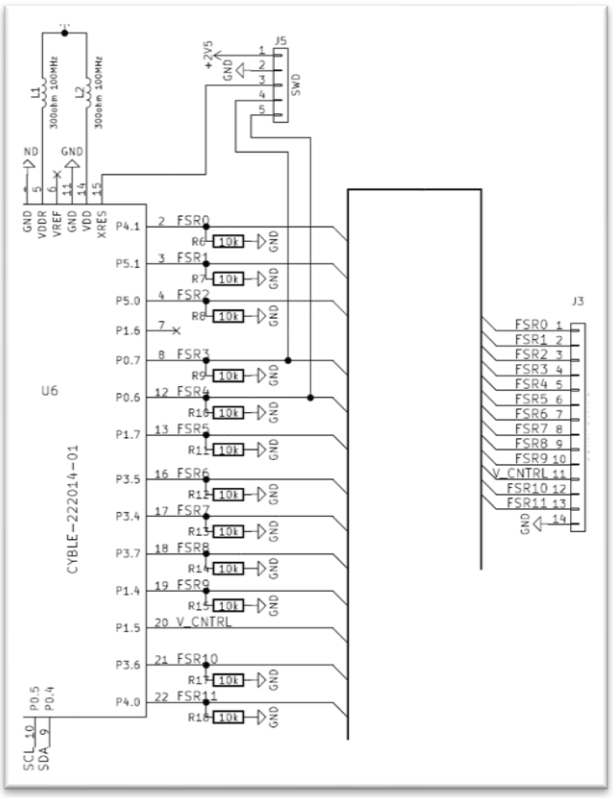


Figura 13 Diseño lógico PSoC 4 con 12 sensores.

Para entender de qué manera se realiza la lectura de un sensor de presión, en la **Figura 14** se puede ver un ejemplo básico de conexión para un sensor resistivo. El principio es sencillo, si el FSR varia su resistencia se producirá una caída de tensión, según el valor que tenga su resistencia interna. Con ello se conecta un pin del ADC[9], capaz de medir la variación de voltaje, en medio de las 2 resistencias. Esta es una manera muy sencilla de transformar una variación de resistencia en una variación de voltaje. Este sistema de dos resistencias se denomina un divisor de voltaje. Si se conoce el voltaje y una de las resistencias, se puede calcular el valor de la otra resistencia que es variable.

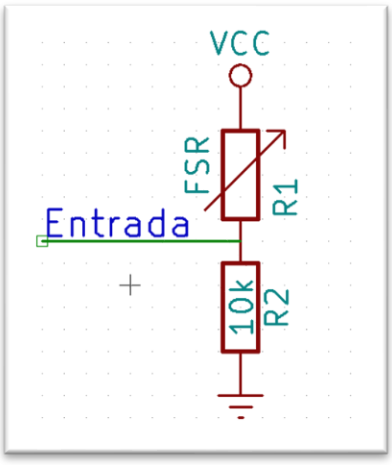


Figura 14 Conexión de un sensor Resistivo

La gran mayoría de ADC tienen un límite máximo en la impedancia de entrada. La impedancia de entrada hace referencia a la resistencia que hay entre la fuente y el canal de entrada del ADC.

[9] Analog-to-Digital Conversion

Por ejemplo, en el caso del ADS7953 la impedancia de entrada tendría que ser menor a 50ohm. Para entender porque esto es un problema, hay que analizar un modelo simple de un ADC:

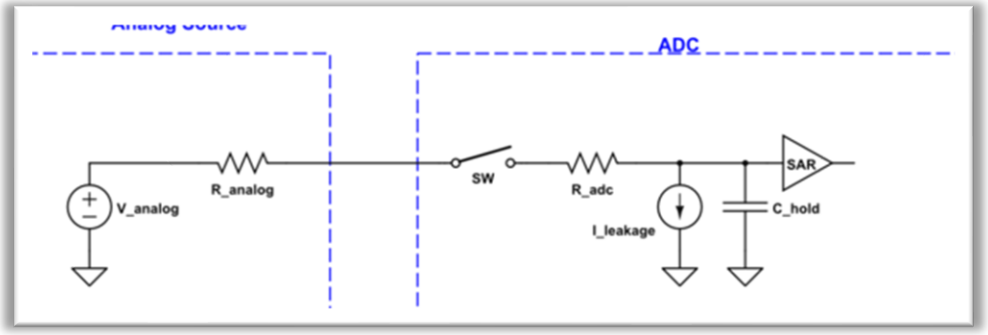


Figura 15 Modelo teórico del funcionamiento interno de un ADC

Lo que ocurre normalmente es que el conmutador "SW" se abre y se cierra en cada periodo de conversión. En este momento el condensador C_hold se tiene que cargar. Ignorando la fuga de corriente que se produce en el sistema, ya que es insignificante, tendríamos que el voltaje que va adquiriendo el condensador (V_{CHold}) viene dado por:

$$V_{C_{Hold}} = V_{Analog} * \left(1 - e^{-\frac{t}{\rho}}\right)$$

$$\rho = \left(R_{Analog} + R_{ADC}\right) * C_{Hold}$$

En los documentos técnicos de muchos ADC's se indica que el tiempo recomendado para que el condensador se cargarse es de 5 ρ. Si el tiempo de muestreo no es suficiente, el condensador no se cargará completamente pudiéndose quedar a mitad de su carga cuando el ADC convierta el valor a digital. Esto quiere decir que el ADC va a leer un valor erróneo debido a que el condensador interno no se ha cargado suficientemente.

Por la impedancia de entrada, la corriente que puede fluir de la fuente al ADC es limitada y por tanto requiere un tiempo determinado. Para solventar este problema hay que utilizar un amplificador operacional. No con el objetivo de aumentar la señal de entrada sino como un repetidor de la señal que le llega desde el divisor de voltaje. En este caso el amplificador operacional no tiene esta limitación de baja impedancia en su entrada. Por tanto, el op-amp[10] va a actuar como un simple repetidor, la señal que le entra la transmite a la salida actuando de fuente de alimentación sin impedancia.

Dado que se van a aumentar el número de sensores se necesita encontrar una solución a cómo multiplexar estos sin que se produzcan interferencias entre ellos. Existe mucha literatura al respecto, en [16] proponen un método de realizar lecturas de una matriz de sensores resistivos. El método propuesto obtiene ciertas ventajas sobre los otros métodos, pero sigue teniendo algún tipo de interferencias. Dado que no es extremadamente crucial disminuir la complejidad hardware se ha optado por una aproximación MOAA[11].

[10] Operational Amplifier
[11] MOAA=Multiplexer and op-amp assisted approach

Este método en dispositivos de muy bajo coste no es viable ya que necesita 3 multiplexores y espacio extra en la PCB. En el caso de la plantilla no añade una excesiva complejidad. Como se puede ver en la **Figura 16** Este esquema necesita tantas resistencias como filas tengamos y un multiplexor para todas las filas. Este se encargará de alimentar el op-amp que a su vez proporcionará la señal al ADC. Para las columnas es necesario tener 2 multiplexores, uno de ellos alimentara el divisor de voltaje y el otro se encargará que no existan interferencias con los otros sensores.

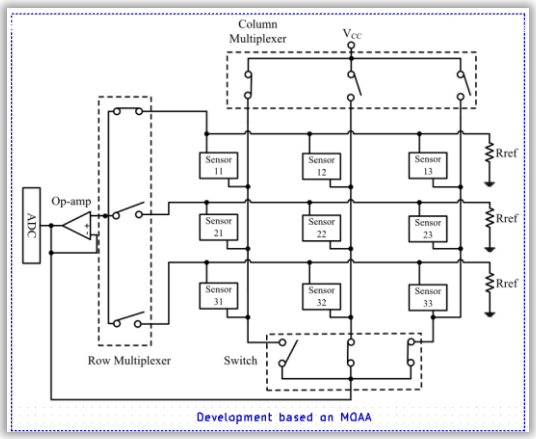


Figura 16 Lectura de sensores resistivos con MOAA

Para poder aplicar lo visto hasta ahora en el diseño lógico de la plantilla se ha decidido hacer en varios planos con conectores tanto para las columnas como para las filas. De esta manera se puede plantear otro esquema si fuera necesario. En la **Figura 17** se puede ver como quedarían los 32 sensores de la plantilla. Empezando por la esquina inferior derecha se empieza a asignar números según la fila y columna en la que se encuentre. Cabe mencionar que el conector J22 podría ser utilizado tanto por el multiplexor encargado del feedback como el que alimenta al divisor de voltaje. Estos trabajan de manera opuesta, cuando uno cierra una columna el otro la abre.

En la **Figura 18** Se puede ver la segunda parte del circuito donde intervienen los multiplexores. Dado que tenemos 4 columnas y 8 filas para un total de 32 sensores, en el conector J14 están conectadas todas las resistencias necesarias para el esquema MOAA. En ese mismo conector, está conectado el IC5 “TMUX1108” que se encarga de multiplexar las filas de la matriz de sensores y enviar los datos al op-amp que a su vez los envía al ADC. Por otro lado, Están los IC8 e IC 7 que son los multiplexores encargados de las columnas. Como se ha mencionado anteriormente estos comparten un mismo conector J16. Tanto J14 como J16 se colocarán al lado de J20 y J22 de la anterior figura para realizar una conexión rápida y permitir cualquier tipo de cambio futuro si fuera necesario. Para poder ver en más detalle el esquemático ver **Anexo 1** . Puede que resulte raro todas las conexiones realizadas con etiquetas, se recomienda al lector ver el Anexo XXX con una imagen de la PCB de desarrollo ya fabricada donde se puede ver cómo quedan estos conectores y se entienda mejor este conexionado.

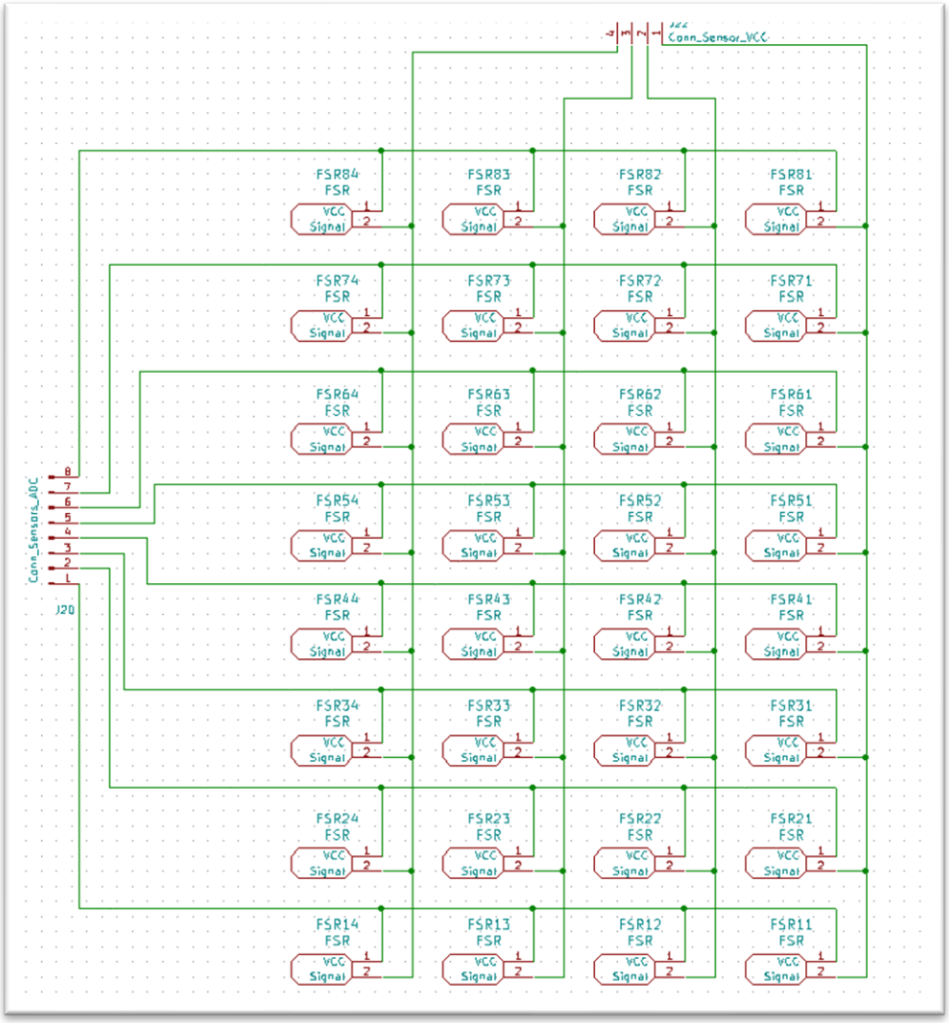


Figura 17 Primera parte de la conexión de 32 sensores resistivos con esquema MOAA

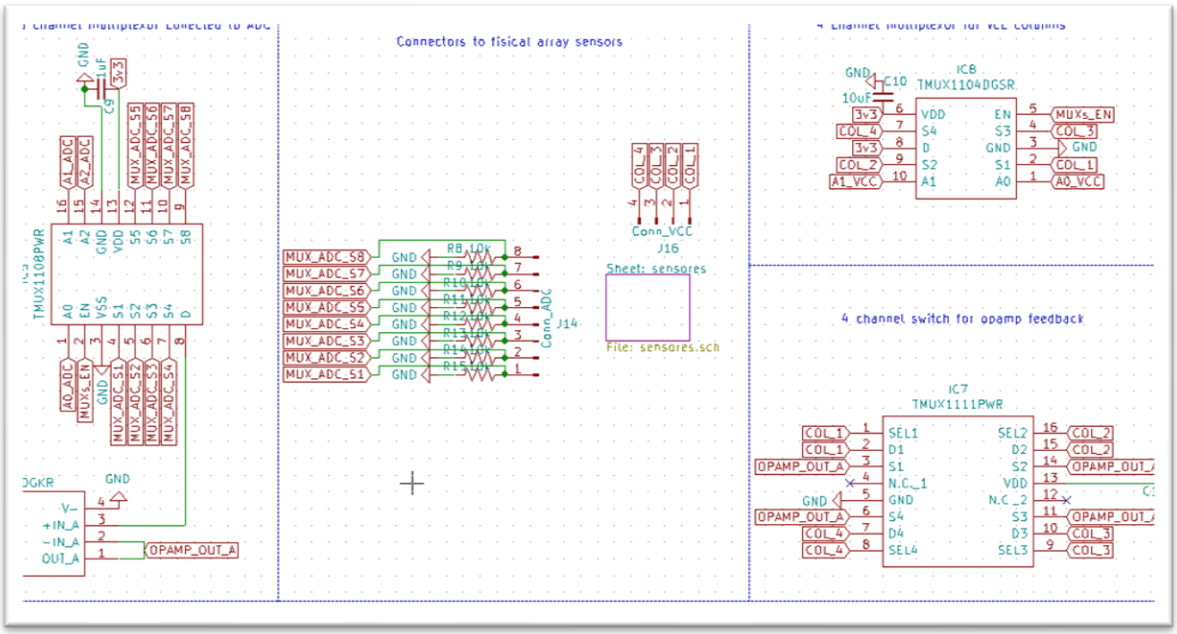


Figura 18 Segunda Parte de la conexión de 32 sensor con esquema MOAA

El multiplexor IC5 “TMUX1108” dispone de 3 pines (A0, A1 y A2) para poder codificiar las 8 columnas. El IC8 “TMUX1104” solamente dispone de 2 pines de selección (A0 y A1) para ir cambiando entre las diferentes columnas. En cambio el IC7 “TMUX1111” tiene otra filosofia a la hora de actuar sobre las columnas. Su estrategia es muy diferente a los dos anteriores. Este multiplexor encargado de la retroalimentación para evitar interferencias entre sensores tiene el circuito abierto en una columna mientras que las otras 3 estan cerradas. En lugar de utilizar otros dos pines de selección utiliza pines individuales para cada uno de sus canales. Esto significa que se podra utilizar la misma señal mandada por el IC8 para poder elegir cual de los canales se va a quedar abierto. De esta manera el IC7 e IC8 trabajan conjuntamente y evitamos conexiones innecesarias al SoC.

Dado que no se puede apreciar la configuración del *op-amp* en la **Figura 19** se puede un esquema más claro de cómo hay que configurar un *op-amp* para que actúe como repetidor de señal.

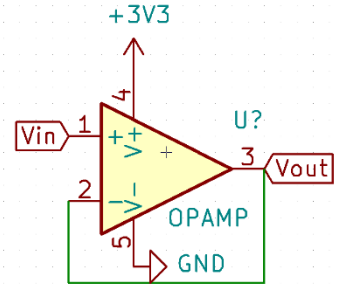


Figura 19 Amplificador Operacional como repetidor

2.3. Gestión de la batería

Uno de los grandes problemas del diseño anterior es la inexistencia de un protector de descarga de la batería. En este nuevo diseño se guardará el mismo chip encargado de la carga de la batería que es el “STC4054-4.2”. Este recibe 5V y carga la batería a una corriente constante, configurable con una resistencia en uno de sus pines.

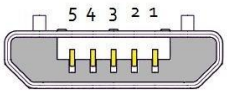


Figura 20 Configuración USB Micro-B

Además, se ha decidido que la carga se realice a través de un conector USB[12] y así poder programar el PSoC por el mismo puerto de carga. La programación se puede realizar con 4 pines, de los cuales 3 se conectan directamente y el otro es el GND. En la **Figura 20** el conector USB Micro-B dispone de 5 pines. El pin número 1 y el 5 se utilizan para GND y voltaje positivo respectivamente. Se utilizarán con el mismo objetivo en este diseño permitiendo que las plantillas se puedan cargar con cualquier cargador USB. Los pines intermedios 2,3 y 4 se utilizarán para la programación, teniendo que mapearlos con los pines XRES P0.6 y P0.7 respectivamente del PSoC.

Uno de los problemas que pueden surgir al utilizar estos pines con otras funciones de las que fueron diseñadas es que el cargador pueda dejar inutilizado el microcontrolador por no cumplir

[12] Universal Serial Bus

los estándares o por alguna descarga. Los pines centrales (2,3 y 4) son especialmente sensibles ya que van conectados directamente al PSoC.

Para solucionar este problema se ha utilizado conmutadores analógicos. Se necesita 3 conmutadores para cada uno de los pines de programación. Para poder diferenciar si estamos cargando la batería o programando el PSOC se utiliza un cable especial de programación que tiene unido el pin 1 y 5, haciendo que el pin 1 que era 5V pase a ser GND.

En la **Figura 21** Conector USB y conmutadores de protección **Figura 21** se puede ver el diseño final del conector USB y los conmutadores. Tanto U3 U4 y U5 se encargan de proteger las líneas de programación del PSoC. Q2 será el *MOSFET*[13] encargado de entregar 3.3 voltios a la etiqueta ON_DEBUG cuando por el pin 1 tenga GND. Esto significa que cuando se necesite programar el SoC este MOSFET activara U3, U4 y U5 ya que su alimentación y activación están conectadas a estas etiquetas. En el caso del conmutador U2 que se ve en la imagen, al conectar un cargador (5V por pin 1 del conector) se encarga de redirigir los 5V a la entrada del regulador de 3.3V. De esta manera el SoC puede seguir funcionando aun cuando este cargando.

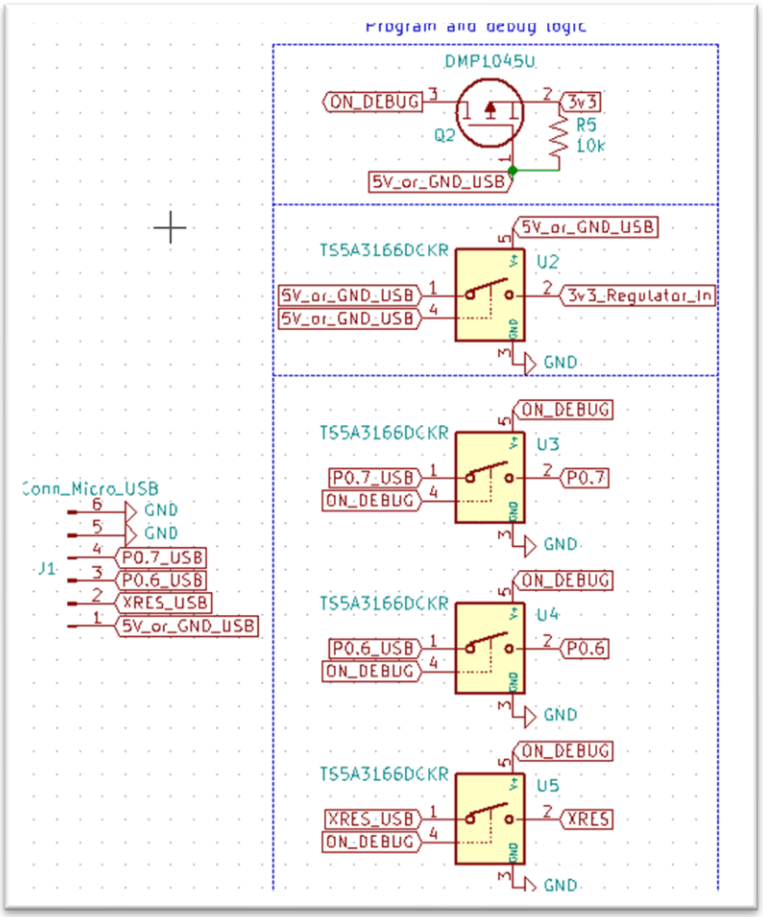


Figura 21 Conector USB y conmutadores de protección

[13] Metal-Oxide-Semiconductor Field-Effect Transistor

Una vez resuelto el puerto de carga hay que añadir un protector de descarga de la batería. El integrado elegido es el "AP9211SA", ver **Figura 22**. Este dejará pasar o no la corriente entre 2 de sus pines (S1 y S2), uno de ellos será el negativo de la batería y el otro el GND general del circuito. Cuando esté integrado detecte que la batería está por debajo de 3.4V, corta la conexión entre estos dos pines. Uno de los inconvenientes de este chip es el proceso de recuperación después de haber saltado la protección. Esta recuperación se producirá en 2 casos especiales: sus 2 pines de control (S1 y S2) se cortocircuitan intencionadamente; o el voltaje está por encima de 3.4V y no hay ningún consumo. Este último detalle de no tener ningún consumo hace que sea complicada la recuperación ya que siempre hay consumos residuales de los componentes.

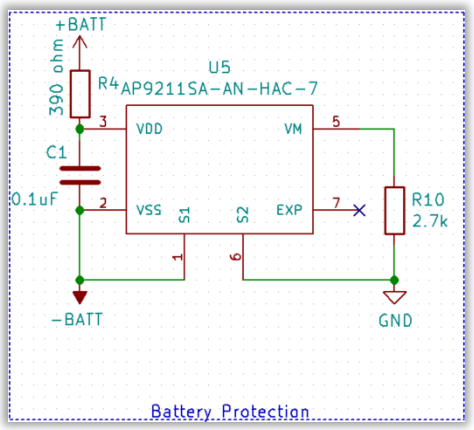


Figura 22: IC protector de batería

La solución encontrada para poder salir del modo de protección ha sido utilizando un MOSFET tipo P en el positivo de la batería, ver **Figura 23**. Su función es sencilla, si la carga (5V) no está presente, se activa dejando pasar la corriente entre la Fuente y el Drenador. Si se conecta el cargador corta la corriente entre Fuente y Drenador. Al ser el primer componente conectado al positivo de la batería y no consumir nada el chip "AP9211SA" podrá salir del modo protección si se alcanzan los 3.4V en la batería.

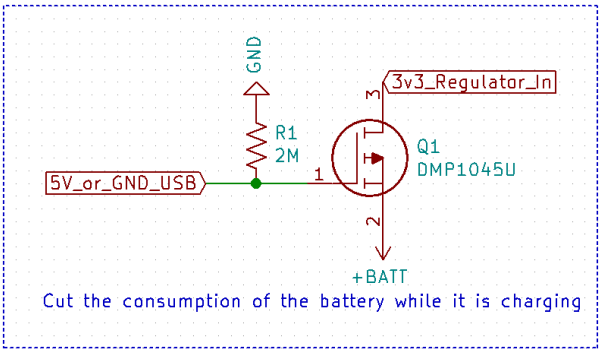


Figura 23: MOSFET de corte

El último paso para poder consumir energía de la batería es el regulador de voltaje. En la anterior versión se utilizaba un regulador lineal, el ADP150 de 3.3V. Los reguladores lineales tienen como ventaja que no tienen mucho ruido a la salida. Su principal inconveniente es la baja eficiencia de regulación. Al ser un dispositivo en el que se prioriza la duración de la batería es necesario utilizar los componentes más eficientes posibles. Para ello se ha decido cambiar de un regulador lineal a uno del mismo tipo que una fuente conmutada. El integrado elegido es el "ADP2108" de 3.3V que simplemente necesita 2 condensadores y una bobina. En la **Figura 24** se pueden ver estos componentes adicionales.

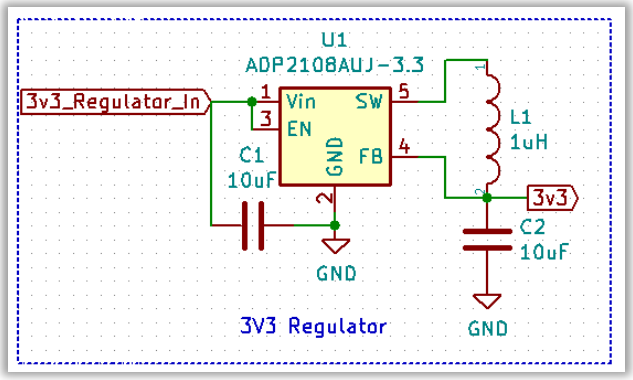


Figura 24: Regulador de voltaje, de 5V a 3.3V

Por último, se necesita un mecanismo de monitorización de la batería por parte del SoC. Se ha tomado la decisión de no utilizar el mismo ADC que se utiliza para los sensores de presión en combinación con la lectura de la batería. La razón es muy sencilla, la lectura de la batería se tiene que hacer en intervalos muy grandes de tiempo y desperdiciaríamos tiempo y energía capturando el voltaje de la batería cada vez que leemos los sensores de presión.

Existen muchas maneras de realizas esta tarea, cada una con ventajas y desventajas. En este trabajo se ha elegido utilizar un amplificador operacional en modo inversor. En primer lugar, ya disponemos del bus de datos I2C que se utiliza para el RTC y la IMU. En este caso se aprovecha y se utiliza el ADC MCP3021 que puede comunicarse por I2C sin problemas. Con ello evitamos ocupar más pines del SoC y no nos preocupa el trabajo que pueda añadirle ya que la lectura se realiza en periodos muy largos de tiempo.

Para poder entender el funcionamiento del esquema implementado se ha utilizado el programa LTspice. Este es un simulador de circuitos implementado por la empresa Analog's. Con este simulador podemos asegurarnos cómo funciona el esquema implementado. En la **Figura 25** está diseñado el circuito a simular.

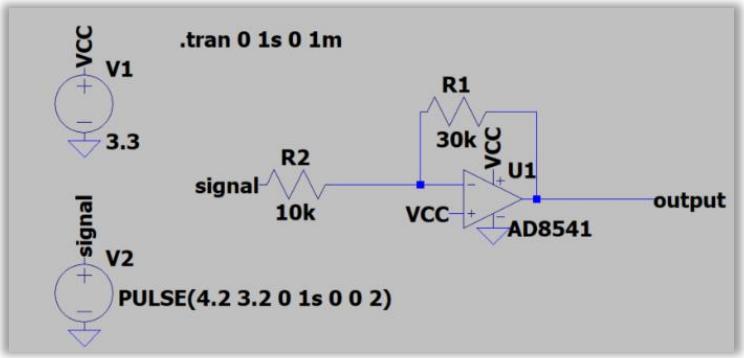


Figura 25 Diseño del circuito de amplificación en LTSpice

La fuente V1 proporciona 3.3V en la etiqueta VCC. La fuente V2 es la que simula el voltaje de una batería que empieza en 4.2V y baja a 3.2 V, este comportamiento lo envía a la etiqueta "signal".

El objetivo principal no es utilizar este *op-amp* para reducir la impedancia de entrada del ADC. Es para obtener más granularidad en la variación de la batería. En este caso estamos trabajando con un margen de 1V de variación. Dado que el *op-amp* también amplifica e invierte la señal,

conseguimos que 4.2V de la batería lleguen al ADC como 0.5V y cuando llegue a 3.3V la batería el ADC tenga 3.3V también. Amplificamos aproximadamente 3 veces la señal. En la **Figura 26** se muestra una gráfica con la señal de entrada y la señal de salida. Como se puede ver al final de la simulación la señal de salida se corta al final de la simulación. Esto se debe a que el *op-amp* no puede entrar más de la propia alimentación que se le está suministrando (incluso algo menos).

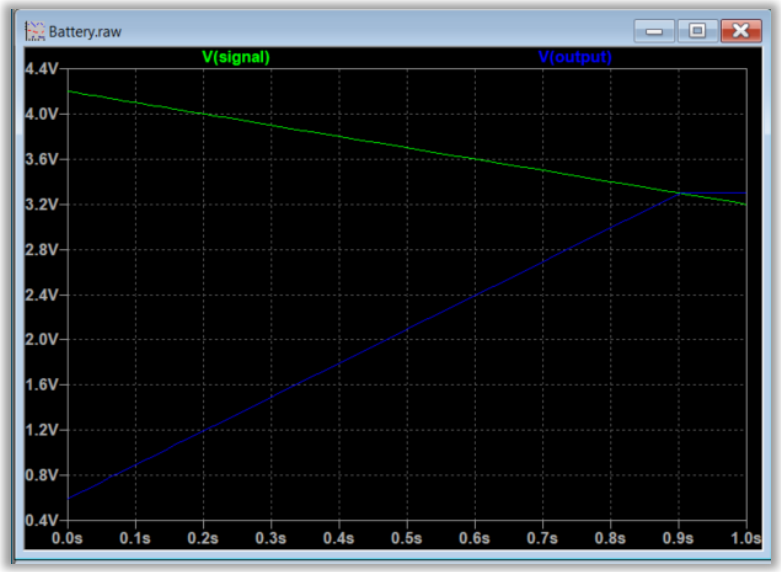


Figura 26 Simulación del circuito de la lectura del voltaje de la batería

Siempre hay que recordar que los amplificadores operacionales en modo inversor invierten la señal de entrada teniendo en cuenta la referencia que se le introduce en la entrada positiva.

Aunque se entiende el circuito mucho mejor con el ejemplo anterior, en la **Figura 27** está implementado en KiCad. Como nota aclaratoria, se han utilizado 3 resistencias de 10 Kohm para reducir los costes de producción. Añadir un componente diferente al circuito hace que sea más caro de fabricar. Como se ha mencionado anteriormente se aprovecha el encapsulado TLV2333 que contiene 2 *op-amp*. Uno se utiliza para la matriz de sensores de presión y el otro para la lectura del nivel de la batería.

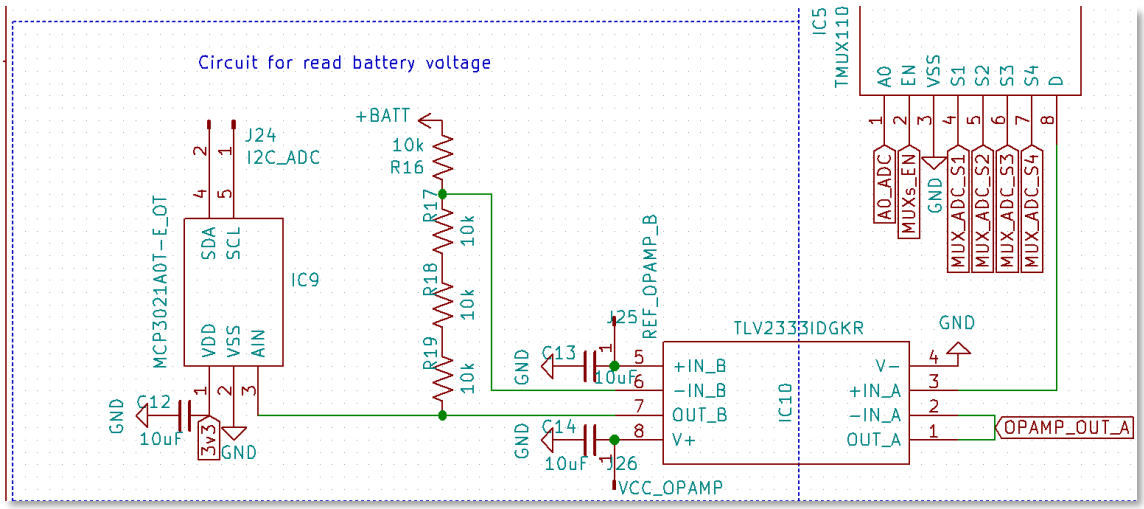


Figura 27 Implementación del op-amp en modo inversor en KiCad

2.4 Diseño físico

Aunque siempre se tiene que aislar el diseño lógico del físico en este prototipo se ha hecho una pequeña excepción. En este caso ser más permisivo con las conexiones entre los dispositivos ha influenciado en los diseños. Pero las modificaciones aportadas para la sección de diseño lógico son mínimas, con pequeños cambios se puede volver a un esquema más generalista.

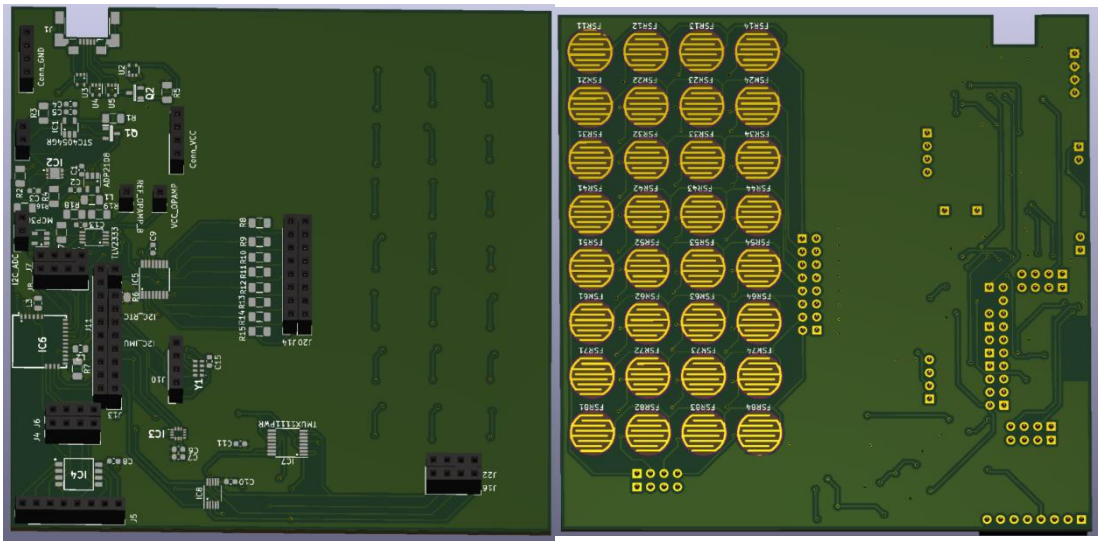


Figura 28 Renderizado del diseño físico con KiCad.

Las decisiones tomadas en el diseño físico del prototipo han sido guiadas por las restricciones puestas por los fabricantes de PCB's. Si en el diseño realizado se superan los 100x100mm el precio aumenta desproporcionadamente. Por tanto, el objetivo principal era diseñar el circuito lo más flexible posible dentro de este espacio. En la **Figura 28** se puede ver el renderizado del diseño final con el programa Kicad.

Una vez se realiza el diseño físico y el enrutado de todas las pistas hay que exportarlo en un formato especifico, este es el formato "gerber". Dentro de este fichero se incluye toda la información que el fabricante necesita para la fabricación. Uno de los fabricantes más conocidos a nivel mundial es "PCBWay", y se ha encargado un total de 5 placas (el mínimo permitido). La fabricación de estas PCB's es de solamente 5€, con 1€ por placa. En la **Figura 29** se puede ver una imagen de la PCB real.

En la **Figura 30** se puede ver la PCB con todos los componentes ya soldados. Se puede ver cómo se han conectado puentes en los conectores relacionados con los sensores. Este diseño está orientado a que si existe algún error se pueda tener alternativas de conexión sin tener que pedir otro prototipo. En la foto también se puede apreciar un adaptador USB a Serial. Este se utiliza para poder imprimir información en una consola en cualquier ordenador por parte del SoC y depurar el código.

La depuración también se puede realizar por el puerto USB tal y cómo se ha explicado anteriormente utilizando un dispositivo especial MiniProg3 ofrecido por Cypress, siendo ésta la única manera de programar el SoC. Al mismo tiempo ofrecer un sistema de depuración muy avanzado. Pero en ocasiones es conveniente disponer de un dispositivo de salida serial para no interrumpir el código. Los protocolos de comunicación inalámbrica tienen unos tiempos

determinados en los que se tiene que responder a las peticiones realizadas. Por ello, interrumpir el código para analizar el funcionamiento puede no ser conveniente.

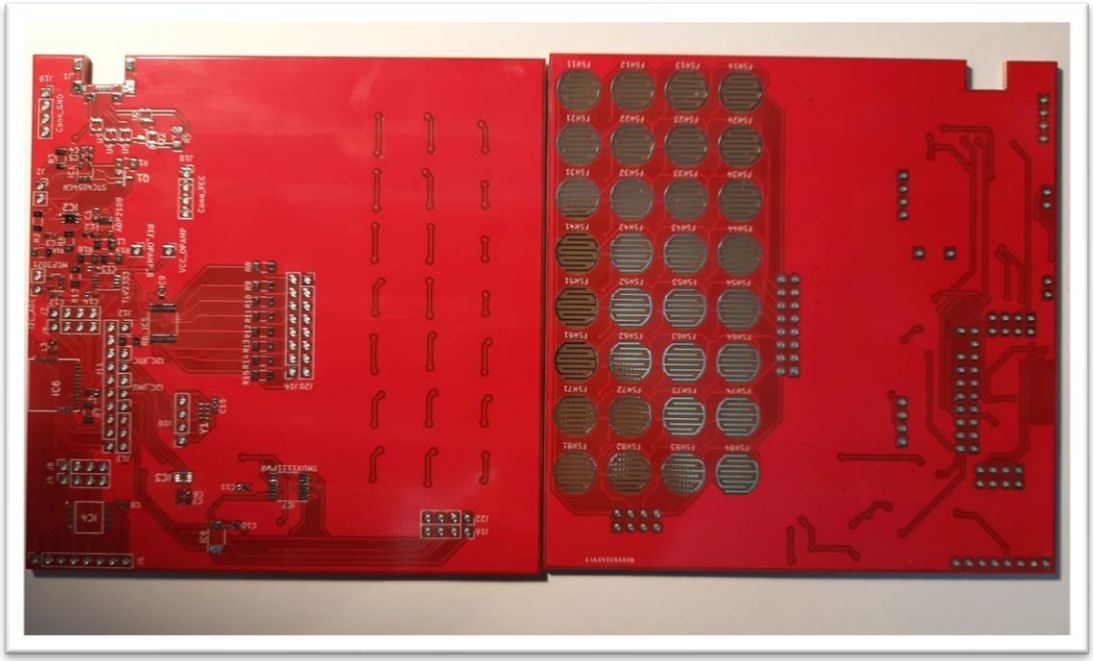


Figura 29 PCBs fabricadas en color Rojo

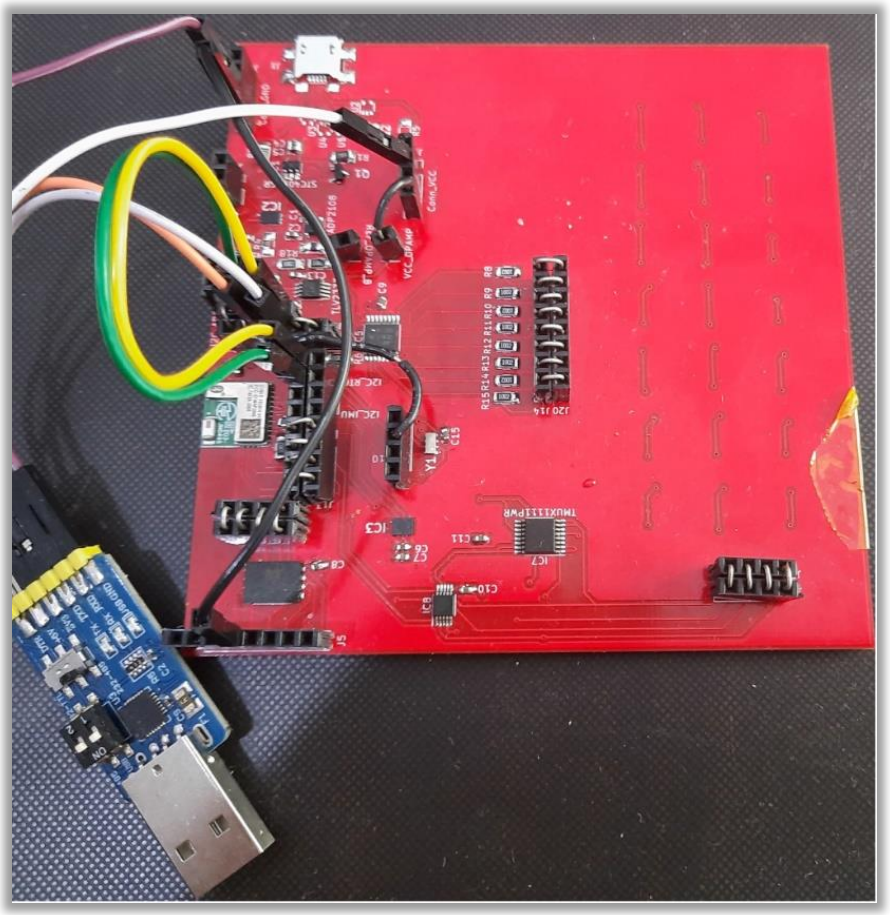


Figura 30 PCB con componentes soldados.

3. Mejoras Firmware

En este punto es necesario explicar algunos conceptos básicos de la estrategia adoptada para ahorrar energía lo máximo posible. Primero se resumirá los estados de ahorro de energía por los que puede pasar el SoC y un esquema general en el que se puede ver el paso entre los diferentes estados:

- **Active**: En este estado el SoC tiene un alto consumo y este no es constante ya que depende en gran parte de la actividad de la CPU que puede variar. El consumo en este estado puede oscilar entre 1.3mA y 14mA.

- **Sleep**: En este estado se pone la CPU en espera, pero se dispone de todos los periféricos que dispone el SoC, como por ejemplo los relojes de alta frecuencia o ADC entre otros. El consumo en este estado oscila entre 1mA y 3mA.

- **Deep-Sleep**: Para reducir aún más el consumo en este estado además de la CPU se ponen en espera los componentes que más consumen. Algunos de estos son: los periféricos de alta velocidad como SPI o UART; los relojes de alta velocidad; o el ADC entre otros. El consumo en este estado es muy bajo llegando a oscilar entre 1.3µA y 15 µA . El bajo consumo de este estado provoca que el SoC tarde 25µs en pasar al estado activo.

En la **Figura 31** se puede ver el comportamiento cíclico del firmware en el **prototipo anterior**. En el bucle principal primero se ejecutan los eventos de bluetooth low energy (BLE), estos corresponden a eventos como conexiones/desconexiones, envió de datos etc. Una vez procesados estos eventos se comprueba si estamos conectados al dispositivo móvil para proceder con las tareas propias del dispositivo vestible: lectura de los FSR's, comunicación con la IMU y empaquetado de datos. Posteriormente se intenta poner el PSoC en sueño profundo, pero no siempre es posible, por ello se entra en una sección critica. Si se consigue entrar en sueño profundo se queda allí hasta que recibe una interrupción de los eventos BLE. Si no puede entrar en sueño profundo, intenta entrar en sueño ligero. Con ello se intenta dejar que los eventos BLE puedan acabar, y vuelva a entrar en sueño profundo. En la entrada y salida del sueño ligero se realizan algunas operaciones sobre unos relojes del sistema. Estos consumen bastante energía respecto al resto de procesamientos. Por ello se intercambia el ECO con el IMO que son dos relojes del sistema, uno de baja frecuencia y otro de alta frecuencia. Una vez realizado el cambio, se apaga el IMO para ahorrar energía.

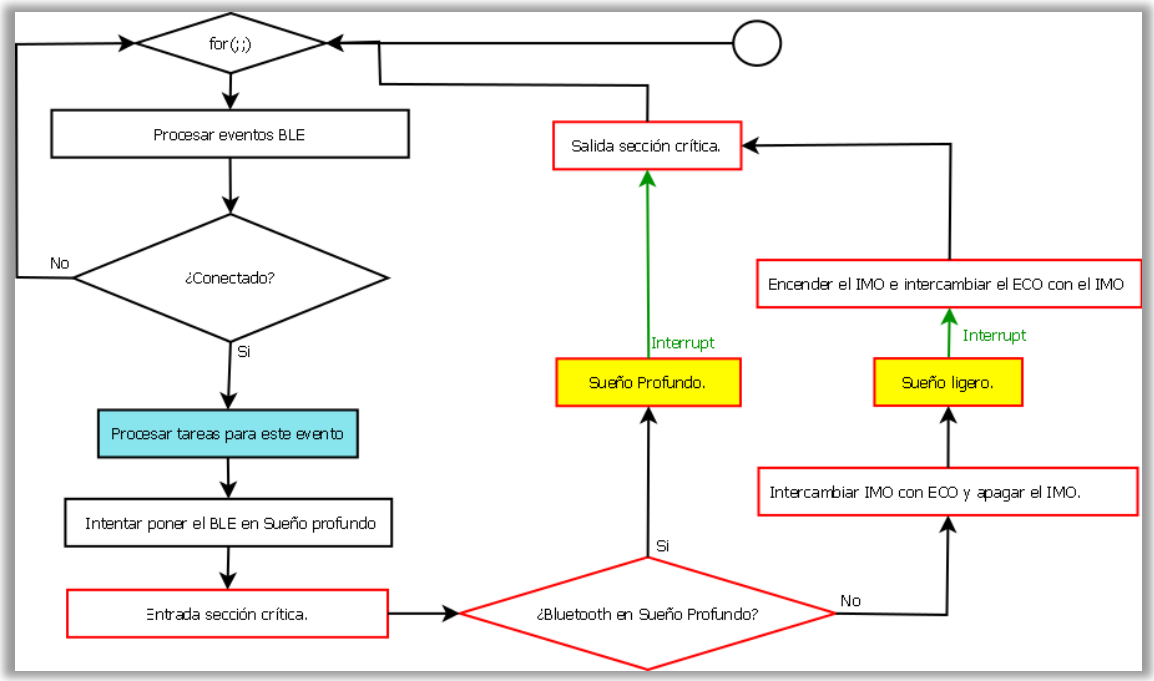


Figura 31 Comportamiento cíclico del Firmware (anterior prototipo)

Este esquema ha dado buenos resultados en el funcionamiento del anterior prototipo enviando datos en tiempo real. En la sección de "Procesar eventos BLE", **Figura 31** , es donde se realizan todos los envíos de datos. La idea principal es adquirir los datos y enviarlos al dispositivo móvil lo antes posible dando lugar a unas ventajas y unos inconvenientes. La principal ventaja es que el hardware y el software son relativamente sencillos. Como desventajas principales tiene la limitación de frecuencia de adquisición de datos y el consumo al tener que enviar datos a la misma tasa de adquisición.

Para entender porque existe una limitación de frecuencia lo más adecuado es ver como se organizan los datos en una comunicación con Bluetooth Low Energy. Los datos están organizados en Atributos, Características y Servicios. Como se puede ver en la **Figura 32** un Servicio contiene una o más Características y esta a su vez se compone de Atributos, este último es el que contiene los datos reales, mostrando como ejemplo la organización de un medidor de ritmo cardiaco. A su vez, este medidor puede hacer parte de un reloj inteligente que ofrece muchos otros datos, entre los cuales los pulsos del corazón. Por tanto, para separar las diversas funcionalidades que puede tener, ofrece como Servicio el ritmo cardiaco. Este a su vez puede tener varias características: dónde está situado el sensor o la característica que va a almacenar los datos. Esta última característica puede tener múltiples atributos, siendo este último paso donde realmente se almacenan los datos. Además del valor que uno obtiene de los atributos, se pueden configurar más opciones. Algunas de estas configuraciones son: la seguridad de ese dato, si es de lectura, escritura o incluso se puede configurar si vamos a notificar cuando el valor cambie.

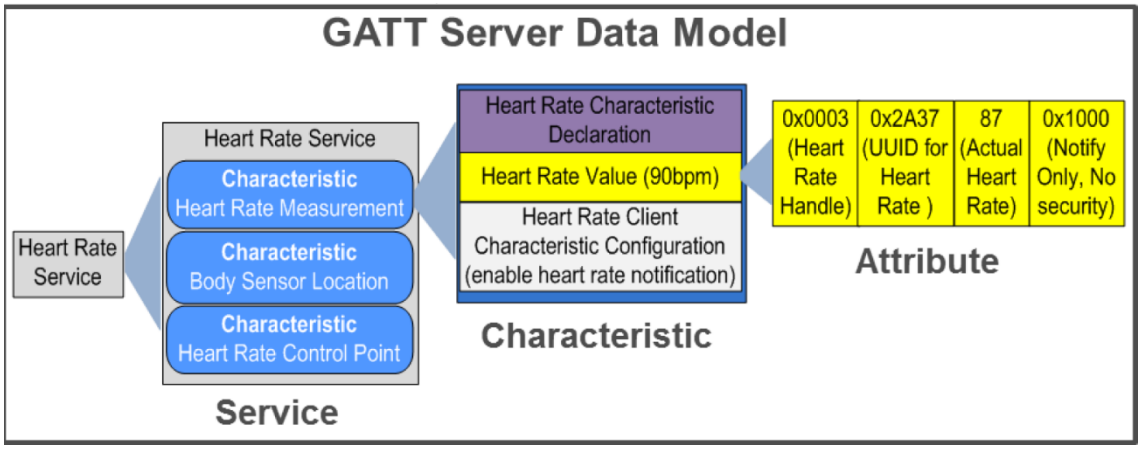


Figura 32 Esquema general protocolos GAPP[14] y GATT[15]

En este punto interviene una limitación del Sistema Operativo Android. Si se quiere utilizar BLE con este tipo de configuración, ninguna característica debe superar los 24 bytes de longitud. Por defecto 4 bytes los ocupa la cabecera y nos quedan útiles 20 bytes. Por tanto, se tienen que empaquetar 32 (sensores de presión) más 6 valores enteros de la unidad de medición inercial. Se utilizan enteros de 16 bits para almacenar todos los valores, por tanto, se va a necesitar 2 bytes por cada valor, que multiplicado por 38 valores tenemos un total de 76 bytes que enviar.

En principio no hay un límite de características que uno puede tener. La limitación viene a la hora de cuantas características simultaneas se pueden actualizar/enviar. En Android todas las actualizaciones de características se realizan en un intervalo de conexión. Dentro de este intervalo de conexión se pueden enviar varias características. En este caso se van a necesitar 4 características para empaquetar todos los datos, 76/20=3,8. En la literatura que uno puede encontrar por diversos foros se indica que tanto en IOS como en Android se dispone un máximo de 4 características simultaneas (4 características por intervalo). Pero las pruebas realizadas con el antiguo prototipo utilizando solamente 3 características han sido un fracaso, nunca se consigue enviar las 3 simultáneamente. Esto puede darse debido a la constante evolución de Android, ya que toda la información encontrada es escasa y algo desfasada. Hay algunos foros que mencionan incluso 6 características simultaneas, todo esto nos lleva a pensar que Android cada vez se pone más estricto. Esto se debe principalmente por el aumento constante de dispositivos que utilizan BLE y si Android permitiera un numero indefinido, probablemente habría dispositivos que nunca podrían actualizar sus datos debido al uso abusivo de otros.

Otra limitación impuesta por los sistemas Android es el número de intervalos. Dependiendo de cuantos intervalos de conexión exista se podrán enviar más o menos datos. En versiones de Android 5.1 se podría disponer de hasta 133 intervalos de conexión,133Hz. Esto quiere decir que podrías empaquetar varias características y enviarlas 133 veces por segundo. Actualmente Android ha bajado esta frecuencia a 89Hz.

Como se puede ver, existe una serie de limitaciones que se solucionaran en el capítulo 4 “Protocolos de comunicación”. La idea principal es eliminar la comunicación constante con el dispositivo móvil y guardar los datos en una memoria para posteriormente enviarlos cuando las plantillas están cargando.

[14] Generic Access Profile
[15] Generic Attribute Profile

En la **Figura 33** se representa el nuevo funcionamiento del Firmware. Dado que no se tiene que enviar constantemente datos al dispositivo móvil este estará más tiempo en sueño profundo y ahorrará más energía, alargando la vida útil del dispositivo.

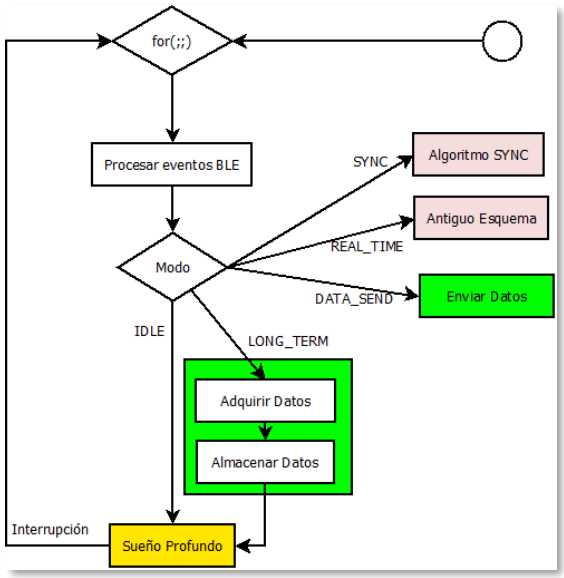


Figura 33 Estados dentro del Firmware de la Plantilla

Se sigue necesitando la presencia de comunicación BLE ya que el dispositivo tiene que ser capaz de cambiar entre los distintos modos. Dado que este prototipo no es una versión final se han dejado especificadas las partes que quedan por implementar y son necesarias para un mejor funcionamiento. Se ha definido 5 modos de operación:

- **IDLE**: El dispositivo está en modo ahorro de energía y no hace nada.
- **SYNC**: Este modo queda como futuro desarrollo. Consiste en implementar un método para calcular el posible desfase entre los relojes de la plantilla.
- **REAL_TIME**: Dado que existen muchos usos para la plantilla se tiene que poder cambiar a un modo en tiempo real. Se puede llegar a utilizar el antiguo esquema o incluso utilizar el presentado en el capítulo 4 “Protocolos de comunicación”.
- **LONG_TERM**: Consiste en adquirir todos los datos, guardarlos y entrar en sueño profundo.
- **DATA_SEND**: Cuando el dispositivo móvil decide que necesita los datos, cambia la plantilla en este modo y empieza la transferencia de datos almacenados.

Los siguientes subcapítulos tratan cada uno de los nuevos componentes en parte. Primero se tiene que entender su funcionamiento y como se tiene que adaptar al nuevo prototipo.

3.1. Funcionamiento RTC

El RTC RV8803 es un dispositivo que es capaz de proporcionar una marca de tiempo con mucha precisión. Para poder incorporar su funcionamiento al Firmware de las plantillas es necesario entender el funcionamiento de este. Como todos los dispositivos que utilizan el protocolo I2C este se configura a través de registros. Dependiendo del oscilador de cristal que utilice un RTC pueden tener más o menos deriva temporal. En el caso de las plantillas es importante entender que cada una de las plantillas dispondrá de un RTC. El dispositivo móvil mandara la hora exacta al mismo tiempo a las dos plantillas. Para no tener que definir la hora en periodos muy cortos de tiempo, es necesario que la deriva temporal de los dos RTC sea la menor posible. En la ficha técnica del RTC se dispone de la gráfica que representa la deriva temporal (corregida) en partes por millón respecto a la temperatura de trabajo, ver **Figura 34:**

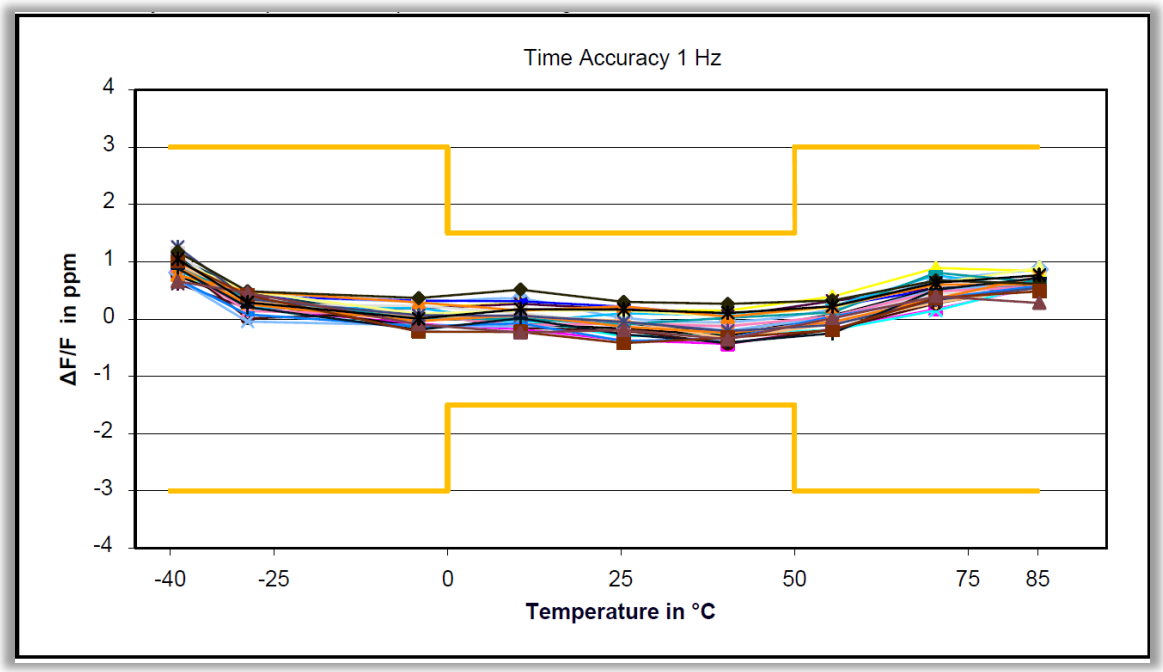


Figura 34 Deriva temporal corregida

Este RTC tiene un sistema automático que es capaz de reducir la deriva, como se puede ver en la anterior imagen. Esta deriva viene dada por la sensibilidad del oscilador de cristal que utiliza. En la **Figura 35** representa cómo varía la precisión del cristal respecto a la temperatura. En la parte superior, en rojo, se puede apreciar cómo se pierde precisión en los extremos de temperatura:

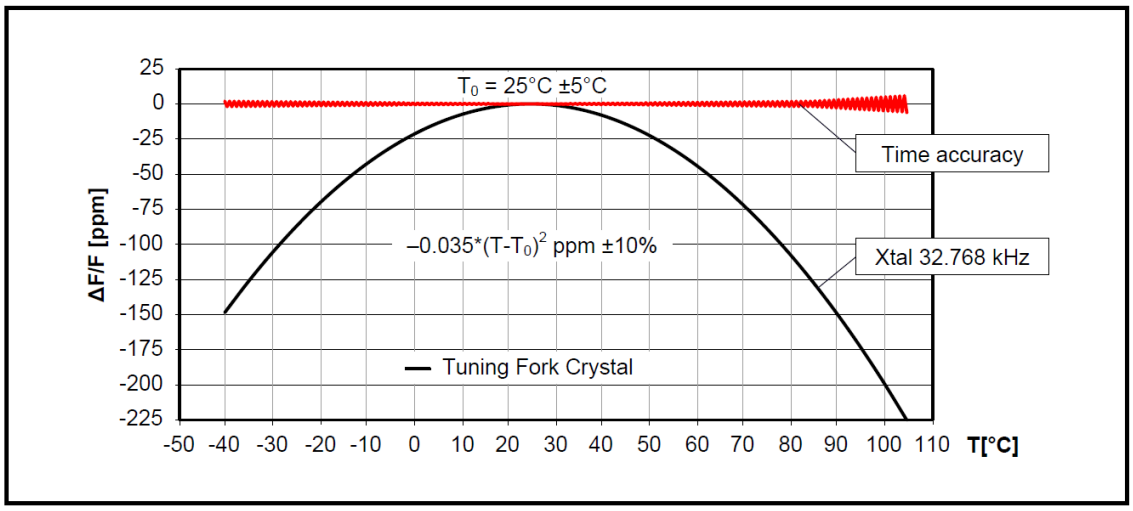


Figura 35 Sensibilidad a la temperatura del oscilador de cristal.

El RTC RV8803 trabaja internamente con el sistema de representación BCD[16]. Es una clase/tipo de codificación binaria de números decimales. Cada digito decimal es representado por 4 u 8 bits. En el caso del RV8803 utiliza 8 bits para representar estos números. Dependiendo del uso, cada fabricante utiliza esta codificación adaptada a sus necesidades. En este caso el RTC representa cada dato de una fecha con 8 bits: segundos, minutos, hora, día del mes, día de la semana, mes y el año.

Para hacerse una idea rápida de cómo se codifican los números en BCD en la **Tabla 1** se puede ver un ejemplo de 4 bits.

Tabla 1 Ejemplo representación BCD

Decimal:	0	1	2	3	4	5	6	7	8	9
BCD:	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Este tipo de configuración no está orientada a la eficiencia al a hora de representar los números. Pero es un sistema que la gran mayoría de RTC's utilizan debido al funcionamiento interno que tienen a la hora de contar el tiempo.

El RV8803 dispone de 3 categorías de registros:

- Registros básicos de tiempo y calendario desde 00h hasta 0Fh.
- Registros de configuración 1 desde 10h hasta 1Fh.
- Registros de configuración 2 desde 20h hasta 2Fh.

Toda la documentación relacionada con la organización interna del RTC se puede encontrar en su ficha técnica. Una definición rápida de estos registros se puede encontrar en el **Anexo 2**.

Se puede llegar a solucionar un mismo problema de múltiples formas utilizando los RTC. Como veremos más adelante el RV8803 tiene algunas funcionalidades incorporadas que te permiten

[16] Binary-coded decimal

tener una gran precisión temporal. Antes de entrar en detalle de cómo utilizar el RTC se describen los registros que representan el tiempo:

- 10h: Es el registro que nos dará las centésimas de segundo. Como tiene 1 bytes (8bits) este almacena los valores desde 00 hasta 99, utilizando el sistema BCD. Dando una granularidad de 10ms en la marca de tiempo. Es muy importante tener en cuenta que este registro es solo lectura. Para clarificar el funcionamiento de BCD de 8 bits este es un buen ejemplo. Dado que el numero a representar en binario, tiene dos dígitos, en decimal se utiliza 4 bits para cada uno, 8 en total. Si transformamos 99 decimal a BCD obtendremos 1001 1001.
- 00h: Representa los segundos y en este caso no es de solo lectura, se puede también escribir ya que es la manera de configurar la hora. Dado que solo necesitamos representar de 00 a 59 no se van a utilizar los 8 bits disponibles del registro. Se utilizarán 4 bits para representar el segundo digito y solo 3 para representar el primero.
- 01h Representa los minutos.
- 02h Representa las horas.
- 03h Representa el día de la semana. Este tiene una particularidad con respecto a las anteriores representaciones. Aunque sigue siento una especie de BCD no sigue el mismo formato anterior. Como sólo tenemos que representar 7 dígitos han decidido que cada bit desde el 0 al 6 representara un día. Si uno de ellos se pone a 1 significa que el día de la semana es esa posición de bit.
- 04h Representa el día del mes.
- 05h Representa el mes del año. Solo se utilizan 5 bits, del 0 al 4.
- 06h Representa el año. Solo es capaz de almacenar valores de 00 a 99.

Dado que el registro de centésimas de segundos no se puede escribir se tiene que dar prioridad al registro de los segundos y posteriormente al de los minutos. En el datasheet del RTC especifican claramente que hay que tener cuidado al leer estos registros. Si por ejemplo lees que estás en el segundo 59 puede darse el caso que al leer el registro de los minutos haya pasado ya al siguiente número. Por tanto, es recomendable realizar otra lectura y asegurar la integridad de los datos. Puede que sea la hora sea 07:30:59 y si en lugar de leer 30 leas 31 y pienses que son las 07:31:59.

Dado que no se puede escribir en el registro de las centésimas de segundos, se ofrecen la alternativas para empezar a contar desde 0. El RTC dispone de 2 bits para realizar sincronizaciones, el bit RESET y ERST del registro 2Fh “Event Control”.

- RESET: este bit se utiliza para congelar los registros. Si se pone a 1 ningún registro aumentara su tiempo. Se establece el tiempo que uno vea conveniente y al ponerlo a 0 el RTC empieza a aumentar los registros.
- ERST: Este bit reinicia el registro de las centésimas de segundo si está puesto a 1 y cuándo recibe un pulso en el pin EVI.

3.1.1. Eventos Externos

En la sección de mejoras hardware se especificó que el RTC dispone de un pin denominado “External Event Function (EVI)” que desempeña una funcionalidad muy interesante. Se podrá mandar un pulso a este pin para que realice una copia de los registros de las centésimas y los segundos. La idea principal de esta funcionalidad es lanzar un pulso a este pin sin tener que

realizar una lectura de registro por I2C. De esta manera tienes una marca temporal muy precisa. Una vez el SoC realice las tareas necesarias puede acceder los registros para saber exactamente cuándo se ha realizado la captura de datos.

Esta funcionalidad está controlada por los bits EIE[17] EHL[18], ECP[19] y EVF[20], de los registros 2Fh "Event Control" y 1Eh "Flag Register".

- Si establecemos EIE a 1 este se encargará de mandar una interrupción a través del pin INT del RTC.
- EHL si se establece a 0 considerara que ha ocurrido un evento en los flacos de bajada. Si se establece a 1 considerara un evento en los flancos de subida.
- ECP es el encargado de habilitar o no la captura de eventos a través del pin EVI. Si lo ponemos a 0 se desactiva esta característica y a 1 se habilita.
- EVF si habilitamos el bit EIE este generara un 1 en EVF para saber que la interrupción que se ha mandado por INT es por culpa de un evento externo y no es por otro motivo.

3.1.2. Interrupción periódica

Otra funcionalidad que ofrece el RTC es la denominada "Periodic countdown timer interrupt function". Esta funcionalidad permite que el RTC genere una interrupción por el pin INT cada cierto intervalo. Este intervalo se puede configurar según los registros Timer0 , Timer1 y los bits 1 y 0 del registro "*Extension Register*". Este último registro ofrece dos bits en los que se puede elegir la fuente de reloj que se quiere utilizar. Se puede elegir fuente las siguientes frecuencias: 1/60, 1, 64, 4096 Hz. Si establecemos por ejemplo 4096 Hz como fuente de reloj y querremos establecer 50 Hz en la interrupción se tiene que calcular los valores a copiar en Timer0 y Timer1. En este caso vendrá dado por: x=4096/freq => x=4096/50 =~82. Como los dos registros Timer0 y Timer1 actúan como uno solo de 16bits hay que escribir el valor 82 en una variable de 16 bits y escribirla en esos registros como si fueran una variable de 16 bits. En la Tabla 2 se muestra unos ejemplos de periodos de interrupción según la configuración elegida en los registros "*timer*" y según la fuente de reloj elegida:

Tabla 2 Periodos de interrupción

Timer Value (0Bh, 1Bh), (0Ch, 1Ch)	Countdown Period			
	TD = 00 (4096 Hz)	TD = 01 (64 Hz)	TD = 10 (1 Hz)	TD = 11 (1/60 Hz))
0	-	-	-	-
1	244.14 µs	15.625 ms	1 s	1 min
2	488.28 µs	31.25 ms	2 s	2 min
:	:	:	:	:
41	10.010 ms	640.63 ms	41 s	41 min
205	50.049 ms	3.203 s	205 s	205 min
410	100.10 ms	6.406 s	410 s	410 min
2048	500.00 ms	32.000 s	2048 s	2048 min
:	:	:	:	:
4095 (FFFh)	0.9998 s	63.984 s	4095 s	4095 min

[17] External Event Interrupt Enable
[18] Event High/Low detection Select
[19] Event Capture Enable
[20] External Event Flag

Estos dos registros están compuestos por:

- Timer counter 0 Register: lower 8 bits
- Timer counter 1 Register: upper 4 bits

Los bits necesarios para gestionar y/o configurar esta función son:

- Flag register:
  - Bit TF: Se pondrá a 1 siempre que se produzca una interrupción proveniente de “Periodic Countdown Timer”
- Extension register:
  - Bit TE: Habilita la función “Periodic Countdown Timer”
  - Bits TD: Encargados de seleccionar el reloj fuente del contador, se puede elegir entre: 4096, 64, 1, 1/60 Hz.
- Control Register:
  - Bit TIE: Con este bit se puede configurar si se quiere que el RTC genere una interrupción con el contador o no.

3.1.3. Obtener una marca temporal

Para disponer de una marca temporal en la adquisición de datos, se puede utilizar la manera tradicional, utilizando el pin EVI para guardar una marca temporal en los registros internos del RTC. El problema de esta función es que solamente se dispone de una precisión de 10ms. Esto te limita a una frecuencia máxima de 100Hz y además tienes un error máximo de 10ms. Para el funcionamiento de un microcontrolador, 10 ms es un error significante.

Dados los inconvenientes mencionados anteriormente, se ha decidido que lo más adecuado es utilizar otra estrategia para tener una marca temporal. El SoC tiene un sistema de gestión de energía gobernado por interrupciones, y es mucho más adecuado utilizar la funcionalidad de interrupción periódica del RTC. Consiste en asumir que una vez se despierte el SoC del sueño profundo, debido a la interrupción del RTC, incremente un contador. Dado que los registros de tiempo se configuran en base a la precisión del reloj interno, las interrupciones tendrán la misma precisión.

Una vez que se establezca la hora dentro del RTC, se configuren todos los registros y se empiece la funcionalidad de interrupción periódica, el SoC se despertará en periodos de tiempo conocidos. Con establecer un contador interno es más que suficiente para saber exactamente cuándo se han adquirido los datos. Esta estrategia se puede combinar con la lectura de los registros de tiempo del RTC. Aun teniendo un contador interno se puede hacer una lectura de los registros y acotar mucho más el error cometido.

Con esta nueva estrategia y sabiendo que podemos realizar interrupciones en intervalos de 4096 Hz hasta 1/60 Hz se puede limitar el error cometido. Lo más intuitivo es que si es necesario adquirir datos a una frecuencia de por ejemplo 100Hz se establezcan las interrupciones a 100Hz. Pero si se requiere minimizar el error se puede establecer una frecuencia de interrupción mayor y no adquirir datos en todos los períodos. En este punto se sacrifica eficiencia energética debido a que se despierta al procesador simplemente para aumentar una variable. En cambio, tenemos más granularidad a la hora de medir el tiempo. Si por ejemplo querremos adquirir datos a 100Hz y realizamos interrupciones cada 300Hz tendríamos una granularidad de aproximadamente 3.3ms. Para no complicar en exceso el prototipo en este nivel no se van a tomar medidas

respecto a la ineficiencia de despertar el procesador para aumentar una sola variable. Como futuro trabajo, se puede intentar solucionar este problema con un chip externo que haga de contador.

Como nota final respecto a las interrupciones hay que recordar, que las fuentes de interrupción adicionales al RTC, son las producidas por la pila del protocolo BLE. Por tanto, el Firmware tendrá que diferenciar en todo momento por culpa de que interrupción se ha despertado y actuar en consecuencia.

3.1.4. Configuración RTC en PSoC Creator

Una vez entendido el funcionamiento interno y de las alternativas que se tiene, hay que incorporarlo al Firmware. Dado que el protocolo I2C es sencillo, una vez se entiende que registros hay que modificar, se puede escribir una librería en C. Como el objetivo de este trabajo no es realizar esta tarea, que es sencilla, pero que llevaría mucho tiempo, se ha decidido realizar una adaptación de una librería desarrollada por SparkFun [17]. Esta está escrita para el entorno de desarrollo Android, en C++. Para poder adaptarla al PSoC Creator se ha tenido que adaptar a C transformado la orientación a objetos a una programación estructurada.

Esta librería ofrece todas las funciones necesarias para configurar el comportamiento descrito en las secciones anteriores:

```
//Starts and stops our countdown timer
void RTCsetCountdownTimerEnable(uint8_t timerState);
void RTCsetCountdownTimerClockTicks(uint16_t clockTicks);
void RTCsetCountdownTimerFrequency(uint8_t countdownTimerFrequency);
//Enables a given interrupt within Interrupt Enable register
void RTCenableHardwareInterrupt(uint8_t source);
void RTCdisableAllInterrupts();
uint8_t RTCgetInterruptFlag(uint8_t flagToGet);
void RTCclearInterruptFlag(uint8_t flagToClear);
void setTime(uint8_t * time, uint8_t len);
//Update the local array with the RTC registers
void updateTime();
//Values in RTC are stored in Binary Coded Decimal.
// These functions convert to/from Decimal
uint8_t BCDtoDEC(uint8_t val);
uint8_t DECtoBCD(uint8_t val);
```

Además de la librería que nos permite configurar el RTC hay que implementar la lógica de interrupción. El RTC se conectará a un pin del SoC y realizará interrupciones periódicas según lo indicado. En la **Figura 36** se añade un Pin de entrada donde se conectará con el RTC y un bloque que genera unas funciones dentro del SoC para poder despertar de sueño profundo con esta interrupción

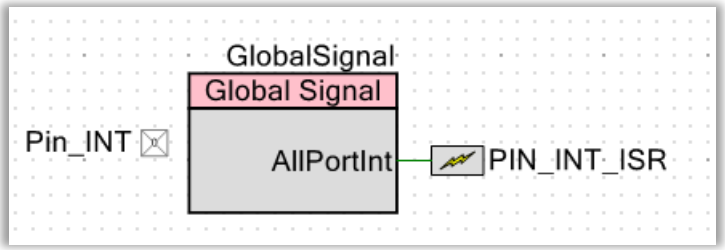


Figura 36 Configuración interrupción PSoC Creator

Posteriormente en el código, simplemente hay que crear una función que se ejecutara cada vez que se recibe una interrupción:

```
CY_ISR (Pin_INT_Handler){
      contador++;
}
```

3.2. Lectura de los 32 sensores de presión

La técnica MOAA dispone de 3 multiplexores que tienen que actuar conjuntamente para poder realizar lecturas validas de los 32 sensores, ver **Figura 37** Gracias a la elección del multiplexor de retroalimentación (multiplexor inferior en **Figura 37**) se ha conseguido eliminar sus las líneas de control. Actuando solamente en el multiplexor de alimentación (multiplexor superior en **Figura 37**) se consigue controlar el inferior. Este se encarga de retroalimentar el array de sensores para que no existan interferencias entre ellos.

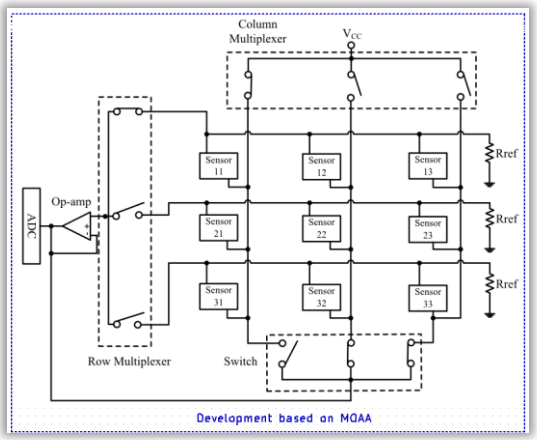


Figura 37 Configuración MOAA

Los dos multiplexores tienen que trabajar simultáneamente para seleccionar adecuadamente la columna deseada. Una vez conseguido seleccionar la columna de sensores que se quiere leer, se indica la fila con el ultimo multiplexor. Este último a diferencia del anterior (2 pines) dispone de 3 pines (bits) de selección para poder elegir entre las 8 filas.

Dentro del entorno de desarrollo PSoC Creator simplemente se tienen que añadir el ADC y los pines de salida que controlan los multiplexores. En la **Figura 38** se puede ver de qué manera permite añadir estos componentes al entorno de desarrollo.

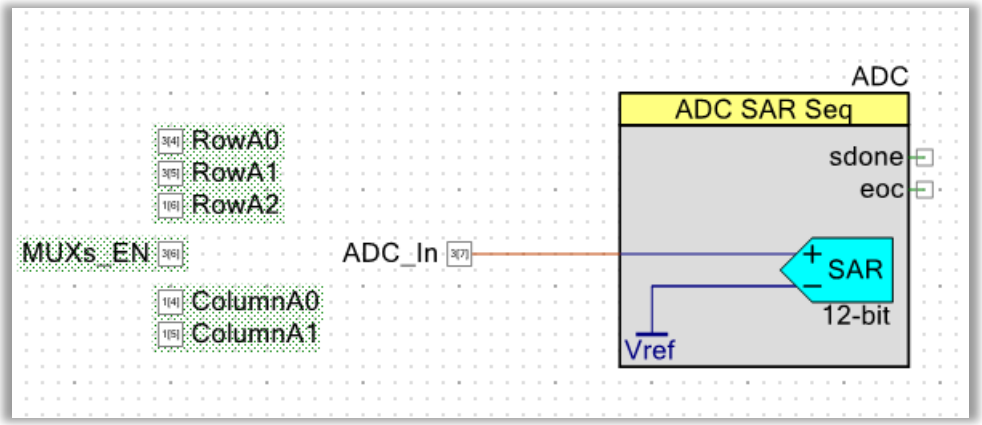


Figura 38 PSoC Creator, ADC y pines de contorl

Todos los pines funcionan de forma binaria (digital), alto o bajo y se puede ver las posibles configuraciones en la **Figura 39**. Todos están configurados como pines digitales de salida, de esta manera el SoC podrá a 3.3V el pin o a 0V.

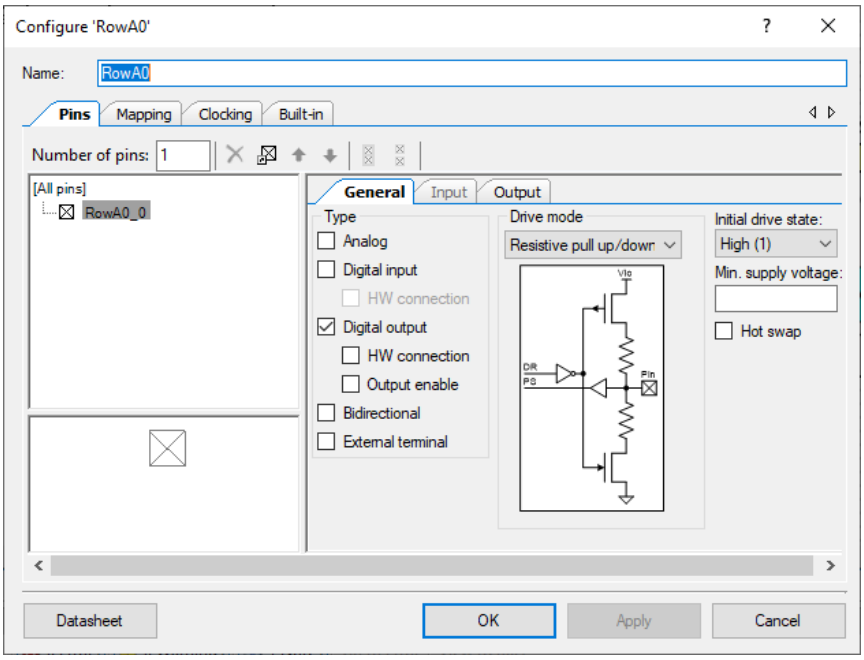


Figura 39 Configurar un pin en PSoC creator

La ventana de configuración del ADC se puede ver en la **Figura 40**, donde se encuentran todos los parámetros necesarios para su funcionamiento. Algunos de los más importantes es elegir el valor SPS que indica a cuantas muestras por segundo va a trabajar el ADC (velocidad de funcionamiento). Es importante seleccionar también el Vref que es el voltaje de referencia, el valor leído será comparado con este para poder dar uno u otro valor. También se puede elegir si querremos que el ADC realice una media de los valores leídos, con esto se puede obtener un valor mucho más estable y que reduce el ruido de la señal.

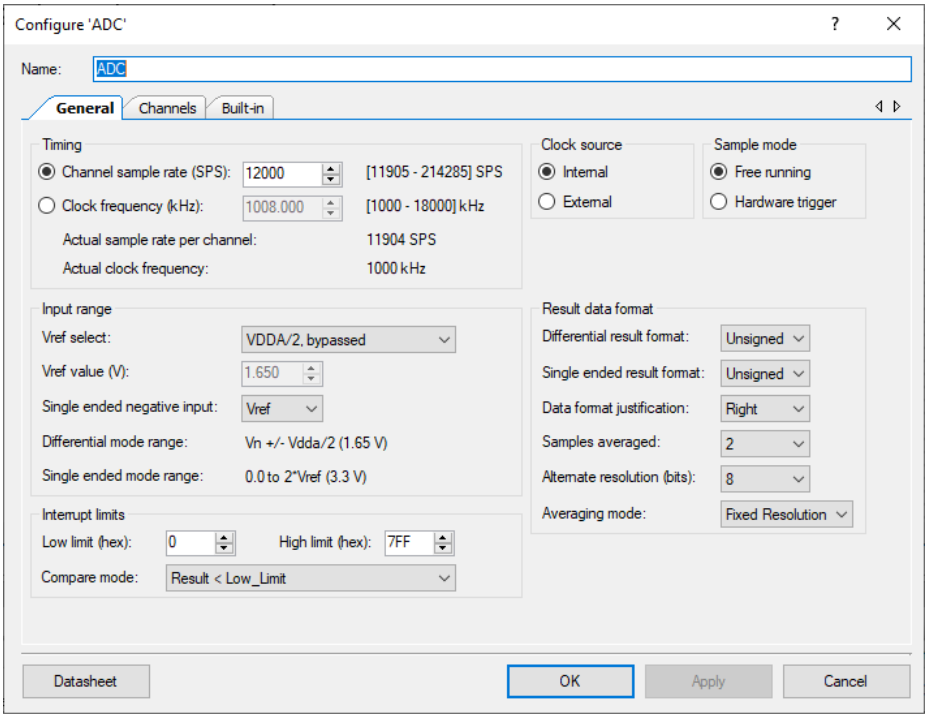


Figura 40 Configurar ADC en PSoC Creator

En el siguiente trozo de código se puede ver el proceso interno que tiene que seguir el firmware para poder leer uno a uno los sensores:

```
for(int j=0,k=0;j<4;j++){
      ColumnA0_Write((j & 0b10)>>1);
      ColumnA1_Write( j & 0b01 );
      for(int i=0;i<8;i++,k++){
            RowA0_Write((i & 0b100)>>2);
            RowA1_Write((i & 0b010)>>1);
            RowA2_Write( i & 0b001 );
            ADC_IsEndConversion(ADC_WAIT_FOR_RESULT);
            Fsrs[k]=ADC_GetResult16(0);
      }
}
```

Al disponer de 4 columnas y 8 filas se realizan dos bucles “for” anidados. En el caso de las columnas se simula con operaciones básicas AND y desplazamientos el incremento de una variable de 2 bits. El SoC ofrece la función X_Write() donde X es el pin al que querremos poner a alto 3.3V o a bajo 0V. Una vez se consigue seleccionar el sensor deseado le indicamos al ADC que vamos a esperar a que termine de adquirir el dato, así evitamos leer registros incompletos. Por último, realizamos una lectura del dato con la función “ADC_GetResult16(0)”. Dado que el SoC dispone de un multiplexor interno permitiendo añadir más canales hay que seleccionar el 0 siendo este el único en la modalidad elegida.

3.3. Funcionamiento de la Memoria FLASH

En la sección 2.1 “Añadir la memoria Flash y el RTC. se mostró las conexiones necesarias para poder utilizar la memoria Flash. En esta sección se tratará su funcionamiento y como adaptarla al firmware. Aunque esta memoria venga en formato SMD (Tecnología de montaje superficial) tiene el mismo funcionamiento que una tarjeta microSD tradicional. Este tipo de tarjetas permiten disponen de 2 protocolos de comunicación, SPI y bus SD. La gran diferencia entre los dos protocolos es el bus de datos, en SPI solo se utiliza un bus de 1 bit y en SD un bus de 4 bits.

Antes de decantarse por este tipo de almacenamiento se realizaron varias pruebas de si este tipo de chips funcionaban con el SoC elegido y se consiguieron buenos resultados iniciales. En este prototipo se intenta realizar una prueba de concepto para evaluar la viabilidad de esta memoria junto al SoC.

Para poder gestionar de manera adecuada una memoria es necesario implementar un sistema de archivos/ficheros. Estos sistemas estructuran y organizan la escritura, búsqueda, lectura, almacenamiento, edición y eliminación de archivos de una manera correcta. En ciertas ocasiones las memorias pueden tener fallos y es importante que el sistema de ficheros pueda detectar estos fallos. Hoy en día existen multitud de sistemas de archivos, algunos de los más populares son: NTFS, EXT, HFS, APFS, HPFS, FAT entre otros.

Dado que es un trabajo muy arduo diseñar e implementar un sistema de ficheros, y se saldría del alcance de este trabajo, se va a utilizar uno de los sistemas más sencillo, el FAT32. Aún siendo el sistema de ficheros más sencillo y liviano sigue siendo una tarea complicada para un procesador que trabaja a 48MHz. Existen múltiples desarrollos de este sistema de ficheros para

adaptarlos a diferentes microcontroladores y uno de ellos es el FatFs desarrollado por ChaN [18]. Este proporciona toda una serie de ficheros que hacen posible el funcionamiento del sistema de archivos. El desarrollo de esta librería se ha pensado especialmente para ser adaptada por diferentes personas en diferentes microcontroladores. Simplemente hay que implementar una serie de funciones dependientes de la plataforma utilizada, en este caso SoC.

Esta librería es bastante modular y te permite limitarla lo máximo posibles, si uno solamente quiere leer de la memoria se puede configurar para que se añadan solamente las funciones relacionadas con esa tarea. Esto es algo interesante para microcontroladores con poca memoria de programación. Además de gestionar ficheros, tiene todas las funcionalidades que se necesitan para formatear completamente la memoria. Permite la utilización de múltiples volúmenes y todo lo que uno podría tener en un Sistema Operativo con respecto a un sistema de ficheros. Aunque se ha mencionado que se utilizara FAT32 esta librería es capaz de trabajar con FAT, FAT16, FAT32 y exFAT.

Muchas de las funciones que ofrece son muy parecidas a las que se utilizan en programas que se ejecutan en un Sistema Operativo. Por ejemplo, para formatear la tarjeta de memoria con FAT32 y 512 de tamaño de bloque, hay que ejecutar:

```
FATFS fatFs;
BYTE work[FF_MAX_SS];
MKFS_PARM opt;
opt.fmt=FM_FAT32;
opt.au_size=512;
UART_UartPutString("\r\n  Formating with mkfs option SFD.... ");
resultF =f_mkfs("", &opt, work, sizeof work);
```

Se elige el sistema de ficheros y el tamaño de bloques de la tarjeta para posteriormente formatear con f_mkfs. El procedimiento para poder escribir en un fichero es algo más complejo y se puede realizar de varias maneras. En todo caso, se tiene que seguir el siguiente orden: montar la “memoria”, abrir el fichero, escribir dentro del fichero y por último cerrarlo. En el siguiente código se han eliminado el código de control/depuración para que se vea de manera sencilla el procedimiento:

```
resultF = f_mount(&fatFs, "", 1);
resultF = f_open(&fileO, "FileName.txt",FA_WRITE | FA_OPEN_ALWAYS);
resultF = f_write(&fileO,&DataToSave, sizeof DataToSave,
&writtenBytes);
resultF = f_close(&fileO);
```

Esta escritura de ficheros se tiene que realizar una vez hemos adquirido todos los datos, tanto de los sensores de presión como de la IMU. Para el análisis de la marcha humana, normalmente se necesitan frecuencias de más de 50 Hz. Dado que no es viable escribir 76 bytes en memoria cada vez que se obtienen, se guardara en memoria RAM para escribir bloques de datos más grandes. Esto es debido a que se invierte más energía en el procedimiento de montaje, apertura y cierre del fichero que la propia escritura.

Una de las tareas más pesadas dentro de esta librería es la apertura del fichero. En las pruebas realizadas en este prototipo se ha observado que es un gran cuello de botella. Una primera solución podría ser tener abierto el fichero durante un periodo largo de tiempo e ir escribiendo los nuevos datos. FatFs dispone de una función llamada “f_sync” que se encarga de actualizar los datos de control relacionados con un fichero abierto. Realiza el mismo trabajo que la función

“f_close” de cierre, pero permite seguir escribiendo. Esto se realiza en sistemas de registro de datos “*data loggers*” donde un fichero se puede tener abierto durante mucho tiempo.

Lo más importante para este prototipo es tener un bajo consumo de energía. Como se especificó en la sección **2.1** “Añadir la memoria Flash y el RTC.” el protocolo SPI que se va a utilizar para comunicar con el chip de memoria utiliza 4 cables/conexiones. Uno de estos el “CLK” que es la señal de reloj con la que funcionara la memoria. La gran mayoría de tarjetas de memoria están configuradas para entrar en modo ahorro de energía si no reciben ningún señal de reloj. Esto se ha comprobado en este prototipo utilizando un osciloscopio y una resistencia de 1ohm en serie con la alimentación del chip. Si se conecta una resistencia de 1 ohm en serie con una carga se producirá una caída proporcional de voltaje en la resistencia con la corriente que la atraviesa. Aplicando la ley de ohm y teniendo una resistencia conocida se tiene que el consumo en Amperios de una carga es igual al voltaje dividido entre la resistencia. Al utilizar una resistencia de 1 ohm la corriente que la atraviesa es igual al voltaje detectado.

Aplicando el método anterior se ha notado una peculiaridad en el comportamiento de la memoria en conjunto con la librería FatFs. Siempre que el microcontrolador ejecuta alguna función de la librería se produce un intercambio de datos entre los dos dispositivos. Por ejemplo, montar la memoria lleva un periodo de tiempo determinado en el que se recibe señal de reloj y se produce un consumo energético por parte de la memoria. En cambio, al utilizar la función “f_open” la tarjeta no puede pasar a un estado de bajo consumo. Esta transición solamente se produce al realizar el “f_close”.

Una alternativa para poder aprovechar el estado de bajo consumo de la memoria es abrir el fichero, escribir y cerrarlo inmediatamente. Pero por culpa de la alta complejidad del sistema de ficheros se emplea más tiempo en procesos de control que en la propia escritura de datos. Anteriormente se ha mencionado que es más recomendable realizar múltiples adquisiciones de datos guardándolos en memoria RAM y posteriormente escribirlos en la memoria Flash. La memoria RAM del PSoC 222014-01 es de 32 Kbytes. En una estimación del entorno de desarrollo PsoC Creator se ha calculado que el firmware actual ocupa alrededor del 40%. Para tener un margen de error para posibles mejoras se ha decidido no utilizar más de 15Kbytes. Dado que se generan 76bytes por cada adquisición tenemos un total de 197 bloques, 15000/76=197,37. Dependiendo de la frecuencia de adquisición de datos elegida tenemos que escribir en memoria cada ciertos segundos. Si se configura a 100 Hz, se tiene que escribir en la memoria SD cada 2 segundos aproximadamente.

Debido a la implementación de esta librería, del sistema FAT32 y al propio funcionamiento del sistema de ficheros, es más costoso el proceso de control de los ficheros que la propia escritura. Cuando la librería utiliza la función “f_open” se le puede indicar los permisos y opciones con los que se va a realizar la apertura. Una de las opciones es colocar el puntero de escritura/lectura del fichero al final del documento. De esta manera se puede seguir escribiendo sin borrar los datos anteriores. Para poder apuntar al final del fichero el sistema tiene que recorrer una serie de punteros para poder llegar hasta el final del documento. Cuanto más grande sea el documento abierto más tiempo va a tardar en colocar el puntero al final del archivo. Este comportamiento se ha observado utilizando el osciloscopio. Cuantos más datos se escriben en un documento, más tardaba en realizar la escritura. Otra prueba realizada ha sido dividir la escritura de datos en múltiples ficheros. Pero a partir de cierto número de ficheros empieza a ocurrir el mismo problema.

Este comportamiento no se puede apreciar en sistemas con un Sistema Operativo debido a las altas frecuencias de funcionamiento. Además, cabe recordar que el protocolo SPI limita la comunicación a un solo bit en el bus de datos. Configurando el SoC a 48 MHz el entorno de desarrollo nos indica que se pueden alcanzar una tasa máxima de 375Kbytes/s.

Dado el pésimo rendimiento que se la podido extraer a la memoria, en esta primera aproximación, se ha decidido que hay que descartar el uso del sistema de ficheros FAT32 con la librería FatFs. Como trabajo futuro se va a necesitar implementar una librería lo más liviana posible para la escritura de datos en la tarjeta de memoria. Una ventaja de este tarea va a ser realizar un diseño pensando en la utilización de los canales DMA[21] . Este es un mecanismo que tienen una gran mayoría de dispositivos para transferir datos directamente a memoria sin involucrar al microcontrolador. Dado la alta complejidad de esta tarea y el limitado alcance de este TFM se ha decidido implementar en la siguiente versión del prototipo.

Este método solo es viable en un entorno en el que el ahorro de energía no es crucial, o no se realicen tantas escrituras.

[21] Direct Memory Access

4. Protocolos de comunicación

En la introducción del capítulo 3 Mejoras Firmware se ha explicado las diferentes limitaciones a la hora de transferir datos utilizando la manera tradicional del bluetooth low energy. Estas limitación son muy específicas de la plataforma Android. Si la comunicación se produciría entre dos dispositivos de la empresa CYPRESS estas limitaciones no existirían. Android aplica estas limitaciones para asegurarse que ningún dispositivo abusa de la comunicación BLE.

Hasta este punto siempre se ha mencionado la comunicación BLE como un genérico. El estándar BLE utiliza una pila de protocolos, **Figura 41**, para poder llevar a cabo la tarea de comunicaciones. Al igual que en Redes de ordenadores en este caso existe una pila del estándar BLE. Dado que este tema no se desarrolla en este trabajo se ha decidido no entrar en mayor detalle de las capas implicadas en la comunicación estándar.

Dadas estas limitaciones, los desarrolladores de Android han decidido dar soporte nativo, al protocolo L2CAP[22] de la pila de protocolos BLE a partir de la versión 10 o también denominada "Q".

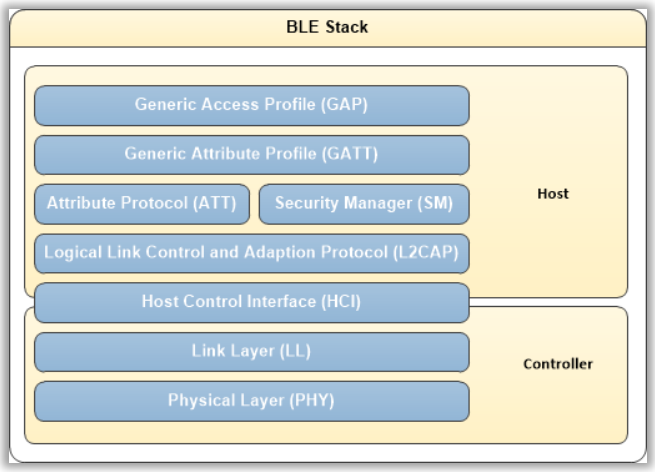


Figura 41 Pila de protocolos Bluetooth

Utilizando directamente L2Cap se prescinde de los protocolos GAP, GATT, ATT[23] y SM[24] de la pila. Una de las ventajas principales de esto es la eliminación de muchos datos de control entre los dos dispositivos. Además, parte de las limitaciones que se tenían anteriormente desaparecen y se podría llegar a velocidades de transferencia mucho más altas en comparación con el anterior método. A cambio vuelve a aparecer un problema clásico a lo largo del desarrollo de este prototipo. Al ser una funcionalidad nueva dentro de Android existe poca documentación/ejemplos de cómo utilizar este método de transferencia de datos. Únicamente existe la documentación que aporta Android al introducir estas nuevas funciones. Aun así, se ha conseguido buenos resultados utilizando este método. Las únicas limitaciones que impone

[22] Logical Link Control and Adaption
[23] Attribute Protocol
[24] Security Manager

Android es el tamaño de bytes que se pueden enviar por un solo envió. Pero no establece ningún límite del número de envíos por segundo.

El prototipo anterior estaba orientado a ofrecer datos en tiempo real al dispositivo Android. El problema principal de esta estrategia es que la conexión constante a través del protocolo BLE consume mucha energía. Además, no en todos los tipos de uso del prototipo es necesario tener los datos en tiempo real. Por estos motivos se ha decidido orientar la captura de datos a una estrategia a largo plazo. En la sección de mejora Hardware se ha añadido una memoria flash donde se guardarán todos los datos de la plantilla.

Aunque se puede seguir la misma estrategia, utilizando la comunicación estándar BLE no es aconsejable debido a las limitaciones que este tiene.

Para poder implementar el protocolo L2CAP entre el PSOC 4 BLE es necesario que el dispositivo móvil disponga del protocolo Bluetooth Low Energy. Si el dispositivo tiene BLE y además dispone de una versión Android 10 o superior se podrá utilizar este método de transferencia.

Irónicamente para poder establecer una enlace utilizando directamente el protocolo L2CAP se va a tener que utilizar también el método tradicional BLE. En la **Figura 42** se hace un resumen del flujo de datos entre el dispositivo Android y el Firmware de la Plantilla. La nueva función implementada por Android es "listenUsingInsecureL2capChannel()" que se invoca directamente del adaptador bluetooth. Este devuelve un objeto de tipo ServerSocket creando así un canal orientado a la conexión. Pero debido al funcionamiento interno de Android, éste no permite configurar de manera explícita el valor PSM[25]. Este valor PSM identifica el protocolo utilizado para comunicarse a través de un canal L2CAP. Android asigna dinámicamente este valor, por tanto, una vez creado este ServerSocket y obtenido el valor PSM se le tiene que comunicar a la plantilla. Aunque este procedimiento complica la comunicación, solamente se tiene que hacer establecer una comunicación L2CAP.

[25] Protocol/service multiplexer

Una vez informada la plantilla del PSM el dispositivo móvil se pone a la escucha en el objeto ServerSocket con el método ".accept(timeout)". En este momento la plantilla se conecta a este socket y el método devuelve otro objeto del tipo BluetoothSocket. Una vez obtenido este objeto se puede cerrar el ServerSocket,".close()" , ya que no se esperan más conexiones. Este tipo de comportamiento es bastante parecido a los que se puede utilizar para establecer una comunicación por una red TCP/IP.

Dentro del BluetoothSocket existe un objeto del tipo InputStream. Es aquí donde se van a ir recibiendo todos los datos que se envían por la plantilla. Este esquema funcionada como un clásico Productor/Consumidor. Con la ventaja que el Productor es mucho más lento en el procesamiento que el dispositivo Android.

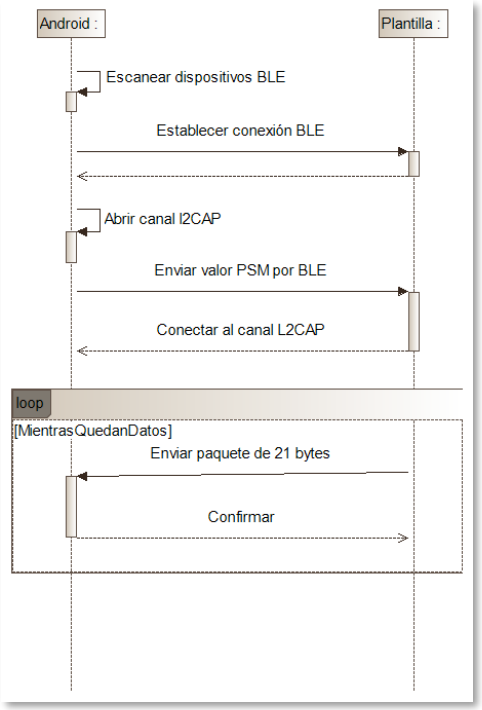


Figura 42 Flujo de datos en la comunicación.

Este InputStream dispone de dos métodos fundamentales, ".available()" y ".read()". El primero nos va a indicar si existen datos disponibles y el segundo es un método bloqueante que devuelve el un array de bytes proveniente del otro dispositivo. Dado que el dispositivo Android es mucho más rápido es recomendable optar por acumular una cantidad de datos importante y realizar la lectura en ese instante.

Para comprobar el funcionamiento de este método en el SoC se ha implementado lo básico para probar el concepto. El Firmware en este caso tiene un bucle infinito en el que comprueba 3 estados en los que puede estar: PSM registrado, PSM no registrado y canal de comunicación creado. En un principio se inicia en el estado en el que no dispone del valor PSM del Android. Una vez que este le comunica el valor del PSM puede establecer el canal de comunicación. Como previamente el dispositivo Android ha sido el que ha iniciado la conexión BLE, el Firmware ya conoce también la dirección física de este, por tanto, se tiene todo lo necesario para iniciar una comunicación utilizando L2CAP.

Cuando se ha iniciado la transferencia de datos, el SoC empieza a enviar los datos con un máximo de 21 bytes por paquete. Para asegurarse que el dato enviado ha sido transferido, lo ideal es esperar a que el móvil confirme la llegada. Una vez que tengamos la confirmación se procede a enviar el siguiente paquete. Este comportamiento cíclico se puede repetir hasta que se termine de enviar todos los datos.

Como anteriormente se ha mencionado existe muy poca documentación respecto a la utilización de este protocolo. Este es un claro ejemplo como un dispositivo puede abusar del estándar BLE para transferir datos de manera masiva. Aparentemente el Sistema Android interviene de vez en cuando en esta transferencia. Para comprobar las velocidades a las que se puede llegar utilizando este protocolo se ha dejado el móvil y el SoC transferir de manera indeterminada datos. Haciendo una media se ha calculado una velocidad de 18Kbytes/s. El potencial de transferencia es mucho mayor, pero Android interviene y limita las transferencias. Se ha comprobado que puede haber picos de más de 32Kbytes/s y en ocasiones la velocidad puede bajar hasta 4Kbytes. El porqué de este comportamiento se sigue sin poder encontrar ninguna explicación.

Otro problema encontrado a la hora de realizar transferencias durante un largo periodo de tiempo es que en ciertas ocasiones la transferencia se para. No sería un grave problema si no fuera porque provoca un error general en el Bluetooth del dispositivo Android. Si este error se produce directamente ninguna aplicación puede utilizar el Bluetooth. Este comportamiento es muy probable que sea un error “BUG” del Sistema Operativo. Debería de poder recuperarse, y volver a configurar él envió de datos para una nueva transferencia. Utilizando el debugger del Andriod Studio se ha observado que en estos periodos de bloqueo en el que deja inutilizado completamente el bluetooth el InputStream está siempre lleno y desbordado. Después de varios intentos sin poder arreglar el problema en el lado del móvil Andorid se ha decidido probar varias estrategias en el SoC. La que ha dado una buena respuesta es añadir un pequeño delay de 1 milisegundo cada cierto número de paquetes. Después de mucho procedimiento de prueba y error se añade una espera en el SoC de 1ms cada 400 paquetes enviados.

Pese a las complicaciones mencionadas se ha conseguido establecer un enlace de comunicación plantilla/móvil con una buena tasa de transferencia de 18Kbytes/s de media. Con esta velocidad se podría llegar a transferir 256Mbytes en unas 4 horas. Dado que el objetivo es realizar estas transferencias mientras la plantilla está cargando no nos preocupa el consumo de energía del SoC.

Con el anterior método se podría alcanzar una velocidad máxima, teórica de 3,56Kbytes/s. Con un máximo de 89Hz y 2 características por conexión y 20 bytes cada característica se obtiene los 3,56Kbytes/s.

5. Análisis de datos de presión

El desarrollo de este tipo de tecnología vestible, relativamente nuevos en el mercado (sin referencias), es complicada por los imprevistos que pueden aparecer. Existen muchas técnicas industriales para acotar estos problemas lo máximo posible e intentar evitarlos. Una manera alternativa de ir superando obstáculos es el diseño de prototipos, siempre cuando estos resulten baratos. De esta manera se ahorra un esfuerzo enorme en el estudio en detalle de todos los documentos implicados en un cada parte del vestible. Si se tuviera que hacer un estudio para limitar los posibles errores, probablemente se tendría que haber invertido todo el tiempo hasta este último prototipo, en leer miles de páginas. Cada chip, cada sensor y cada técnica de análisis es un mundo y existe una extensa documentación al respecto.

Hoy en día se dispone de técnicas de prototipado muy rápidas. Con respecto a la creación de material se pueden utilizar las impresoras 3D para crear recubrimientos o cualquier objeto en tridimensional. Con relación a los circuitos impresos se dispone de multitud de páginas web que ofrecen este servicio por un coste muy bajo. Disponiendo de las herramientas y conocimientos necesarios hoy en día es asumible el desarrollo de productos en base a multitud de prototipos. Otro aspecto importante, que no está bajo el control del desarrollador, es la falta de documentación o la incompletitud de esta. Muchas veces se asumen funcionalidades o características que el producto final no tiene. Para entender a qué se hace referencia, en la parte hardware se han explicado algunas problemáticas que se ha tenido a la hora de conectar la memoria FLASH. El SoC no puede admitir más de 3 dispositivos en modo "*Slave*" aunque en la documentación especifica hasta 4. No todos los puertos E/S se pueden utilizar para el ADC o para la comunicación I2C. Y muchas veces esta incompletitud de documentación puede causar grandes costes en el desarrollo de prototipos. En ocasiones es conveniente realizar una primera aproximación y poder realizar un plan más avanzado y completo para los siguientes prototipos.

Dado el conocimiento adquirido en el primer prototipo, y la necesidad de explorar cada técnica propuesta, se ha creado un prototipo intermedio entre el prototipo inicial (presentado en el TFG) y el prototipo presentado en este TFM. La idea principal era poder avanzar en el análisis de los datos al mismo tiempo que en el prototipado hardware. Dado que el TFM tiene un alto contenido en diseño hardware se ha decidido incorporar, el análisis de la marcha humana. Este prototipo intermedio incorpora solamente el aumento de sensores hasta 32. Con este prototipo se pueden descubrir posibles inconvenientes que de otra manera no se podría descubrir.

5.1. Descripción del problema:

Se dispone de un prototipo de plantillas sensorizadas, ver **Figura 43**, que contiene 32 sensores de presión y una Unidad de Medición Inercial (IMU). Los sensores de presión están distribuidos a lo largo de toda la superficie de la plantilla menos en la zona de los circuitos. Tanto la IMU como el SoC están situados en el hueco del pie donde menos presión se realiza. Cada uno de los sensores de presión puede oscilar desde 0 a 4096 dependiendo que fuerza se está ejerciendo sobre este. La IMU es de 6 ejes, esto quiere decir que mide las aceleraciones y giros en los ejes X, Y, Z.

Uno de los campos de investigación que se verán beneficiados con este proyecto es el análisis cuantitativo de la marcha, a partir de ahora QGA, permite demarcar con precisión eventos como el golpe de talón (heel-strike HS) o el levantamiento de puntera (toe-of TO), así como otras fases del ciclo de la marcha humana, ver **Figura 44**.

Figura 43. Prototipo de plantilla sensorizada

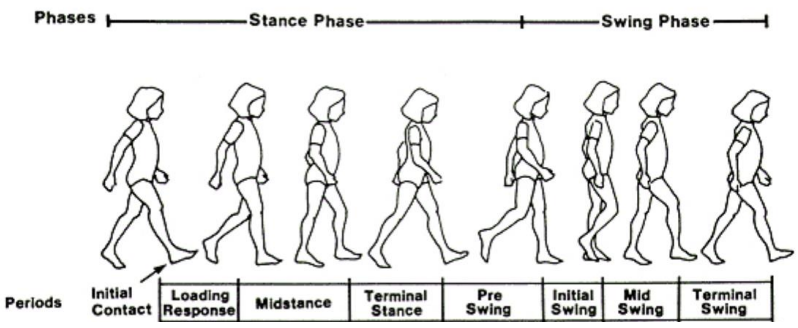


Figura 44. Fases del ciclo de la marcha humana

Esta capitulo se centra en el análisis de la marcha humana normal (sin ninguna patología). Por tanto, se puede ver en la **Figura 45** cual es el patrón de una marcha humana normal basado en los sensores de presión que dispone una de las dos plantillas.

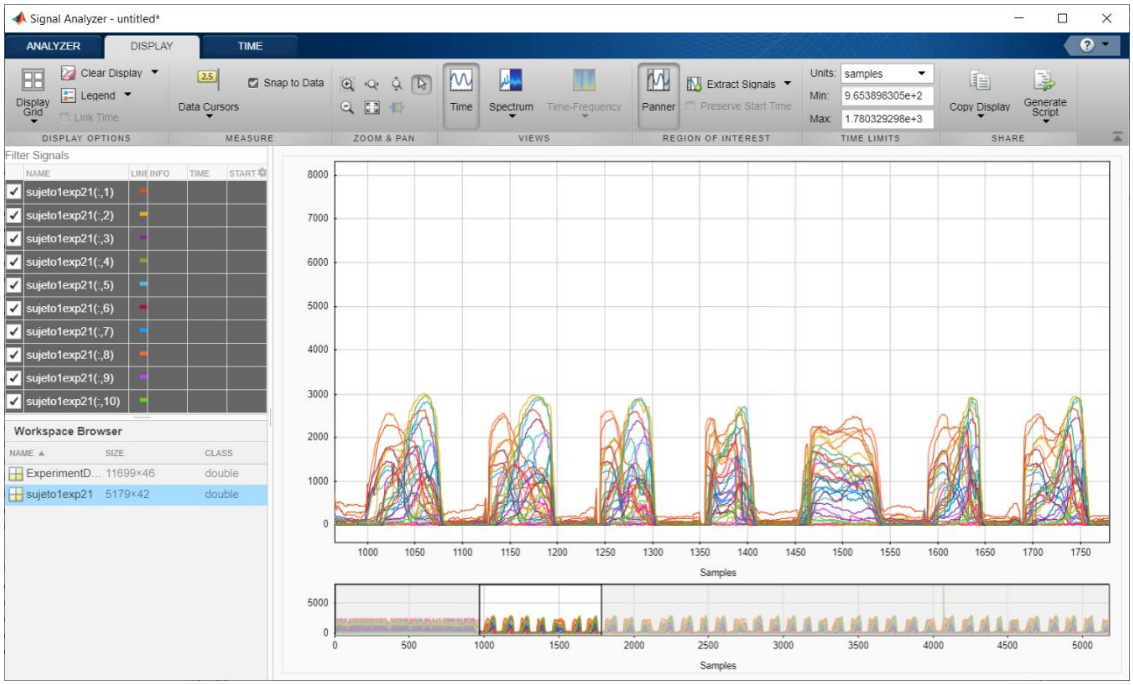


Figura 45, señales generadas por los sensores de presión.

Para el primer análisis de los datos se ha utilizado "*Signal Analyzer*" de Matlab. Se puede diferenciar claramente una fase en la que los sensores están activos y una en la que no.

Por otra parte, también se puede apreciar en el análisis que hay valores que exceden los valores normales de los sensores, esto se puede apreciar en los valores alrededor de 4100 (**Figura 45)** en la ventana baja. Se puede apreciar unas línea haciendo que los demás datos estén a una escala que no corresponde.

En las **Figura 46 y Figura 47** se pueden ver los datos generados por la IMU. La IMU se ha configurado para un máximo de +-8g's para la aceleración y 1000dps[26] (degree per second) para el giroscopio. Dada esta configuración y sabiendo que tenemos un registro de 16 bits para los datos, tendremos unos valores entre – 32768 y +32768 . Para saber exactamente la fuerza"g" o los "dps" se tiene que multiplicar por 8/32768 y 1000/32768 respectivamente.

[26] Degrees Per Second

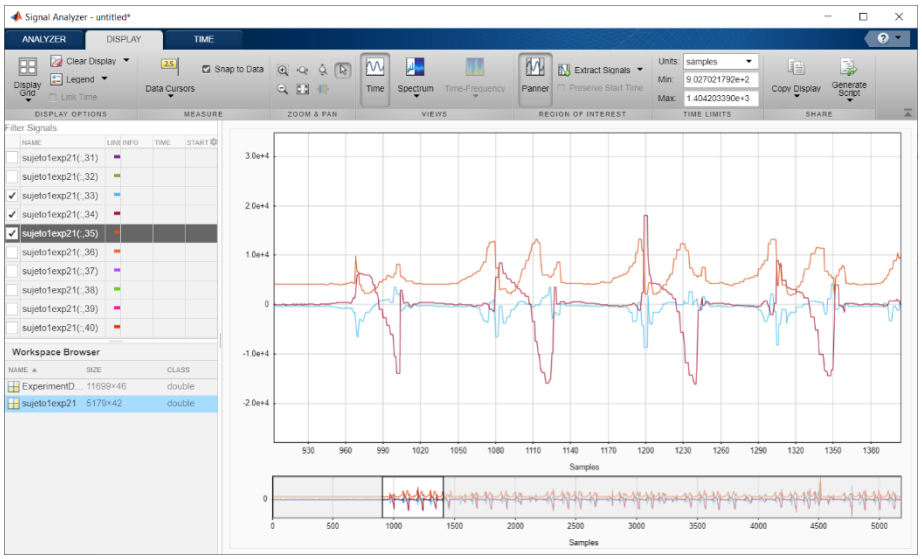


Figura 46. Datos generados por la IMU: Aceleraciones.

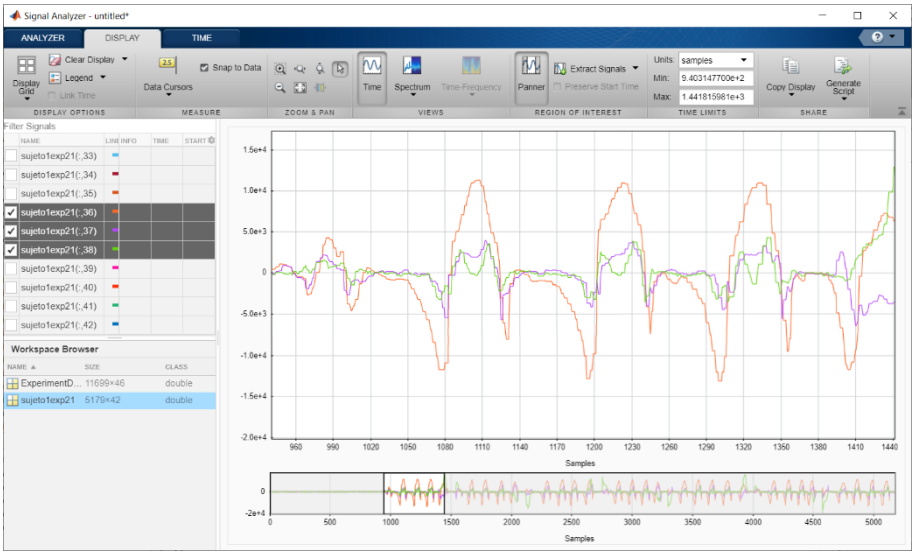


Figura 47 Datos generados por la IMU: Giroscopio.

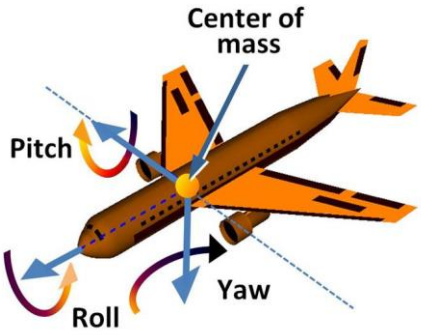


Figura 48 Rotaciones Yaw Pitch Roll.

Además, en el preprocesamiento se han generado otros datos adicionales. Los datos generados por la IMU son difíciles de interpretar visualmente, en su formato en bruto, por tanto, se ha aplicado un algoritmo de fusión de datos inerciales llamado Mahony. A partir de esta fusión se ha generado los ángulos que se utilizan normalmente en la aviación: Yaw[27], Pitch[28] y Roll[29]. Estos ángulos se pueden entender de manera más sencilla viendo la **Figura 48**.

[27] Dirección.
[28] La elevación.
[29] El ángulo de alabeo.

En la **Figura 49** se pueden ver estos datos en “Signal Analyzer”.

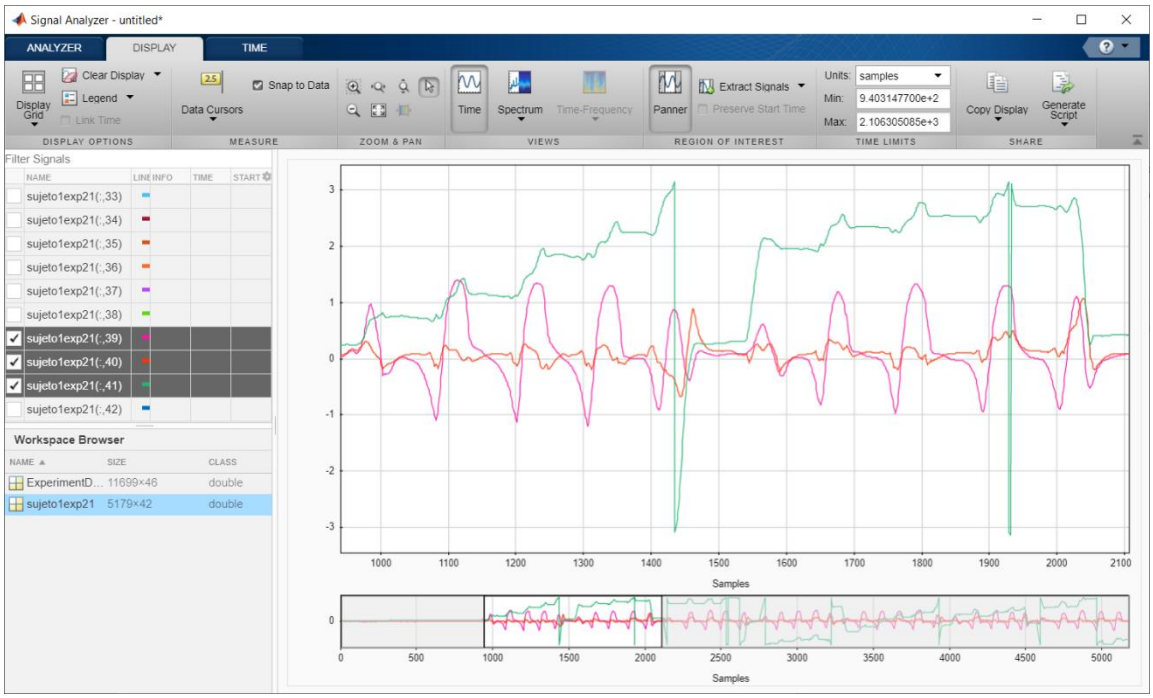


Figura 49. Algoritmo Mahony.

Por último, se dispone de la marcación de los eventos golpe de talón HS y levantamiento de puntera TO. Para entender de manera más sencilla la distribución de los datos y cuando ocurren estos eventos, en la **Figura 50** se puede apreciar con una línea discontinua cuando ocurren estos eventos. También se ha dejado algunos de los sensores de presión activos para entender el contexto y se han estandarizado los datos.

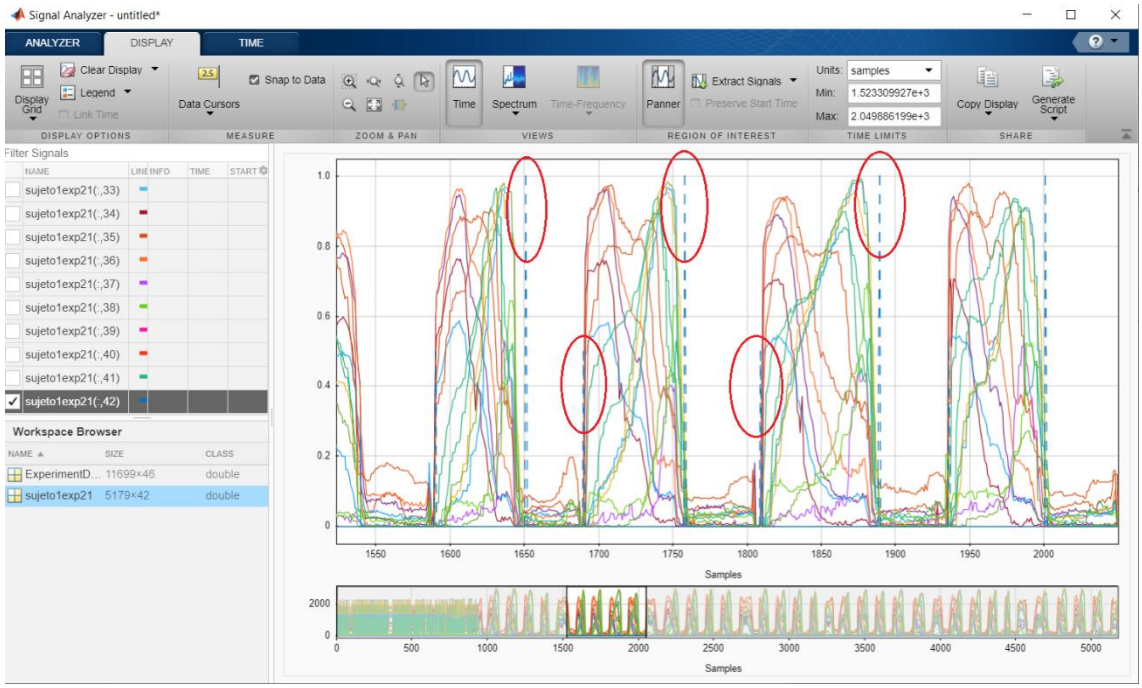


Figura 50 Demarcación de Eventos HS y TO.

Para poder analizar la marcha de una persona lo primordial es poder determinar estos dos Eventos TO y HS. A partir de estos se puede intentar demarcar otros eventos. Para este trabajo se dispone de 5 experimentos de 3 personas distintas.

5.2. Utilizando clasificadores SVM

En la **Figura 50** en la parte baja se puede apreciar la ventana temporal y los datos en miniatura. Todo experimento empieza con el sujeto de pie, por tanto, esos segundos hasta que empieza a andar se han eliminado. Además, se han juntado todos los datos de todos los sujetos y se han eliminado los intervalos de sensores de presión activos donde no había una demarcación de los eventos TO y HS. En otras palabras, se han eliminado porciones de las señales donde los sensores de presión estaban activos, pero no había una clasificación de eventos en esas regiones. Esto es debido a la manera en la que se ha realizado el experimento, ya que los sujetos de prueba podrían dar aproximadamente 4 o 5 pasos con un solo pie y luego tenían que darse la vuelta. Esto implica que en la zona en la que el sujeto gira para cambiar de dirección 180º no se incluye en el experimento.

El principal problema para no poder aplicar directamente un clasificador supervisado es que se dispone de aproximadamente 11.000 datos por cada sensor de presión (32), 6 para la IMU y 3 columnas más para los ángulos YAW Pitch y Roll. Con una matriz de ~11.000x42. De estos 300 están clasificados como eventos TO y HS. Hay un claro desequilibrio si se quiere dividir en 3 clases: 0 cuando el pie no está apoyado, 1 cuando se produce el evento HS y 2 para TO.

Se puede aplicar 2 estrategias diferentes:

- Realizar un "*clustering*" sobre los datos que no están clasificados como eventos HS y TO y obtener los 150 datos más significativos de cada clúster. Esto quiere decir que se realiza un clúster indicando que se quiere obtener 150 clases. Posteriormente se obtiene la fila que está más cercana al centroide del "*cluster*". Con esto se puede considerar que se obtendrá un equilibrio entre las clases.
- Utilizar una matriz de costes indicando la penalización por equivocarse en cada una de las clasificaciones.

Si se pone en práctica la primera idea y se aplica con un clasificador multiclase SVM obtendremos la matriz de confusión de la **Figura 51**.

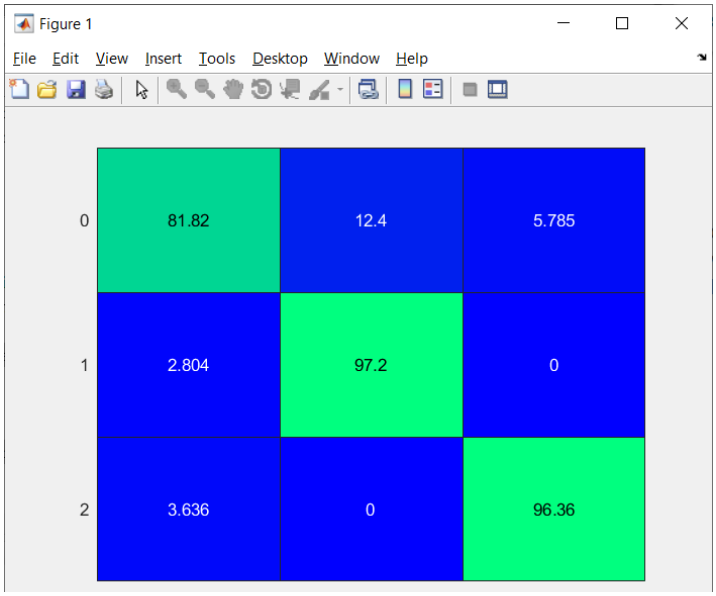


Figura 51 Matriz de confusión con clustering kmeans.

A primera vista puede parecer un buen clasificador vistos los resultados. El problema es que, al probar el clasificador con otros datos, diferentes de los que ha sido entrenado, se equivoca muchas veces clasificando la clase 0 como 1. Puede que este clasificador sea una buena opción cuando se tenga una buena base de experimento mucho más grande. Dándole la oportunidad de poder entender cuál es la diferencia entre las clases 1,2 y la 0. En la **Figura 51** se puede ver como el clasificador falla mucho en la clase 1 y la 0. En la

Figura 52 se puede ver como muchos de los datos entre zancadas son clasificados como clase 2. Y hay que recordar que el evento TO es muy puntual, no puede durar tanto. El error que se produce en la clasificación de la clase 1 es aceptable ya que por ejemplo en este caso se ha observado un error máximo de 8 muestras. Teniendo en cuenta que se han tomado muestras a una frecuencia de 90Hz, pudiendo tener caídas hasta 80Hz estaríamos herrando entorno a los 100 ms.

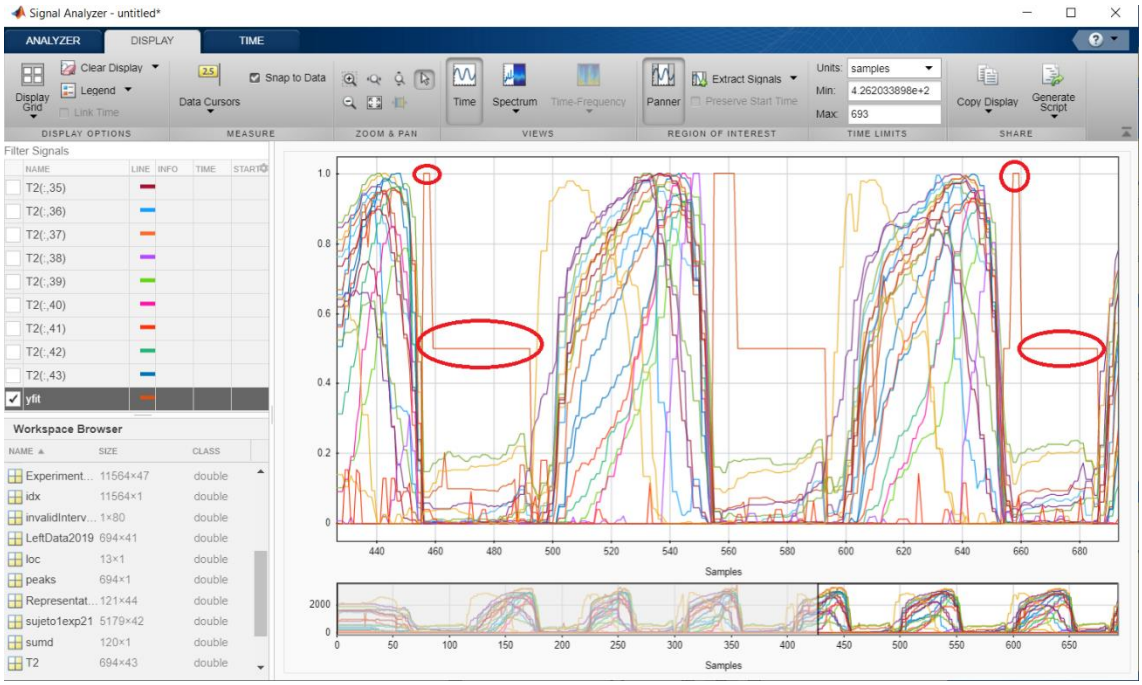


Figura 52 Test sobre nuevos datos.

Las principales características de este clasificador utilizando Matlab han sido:

- Validación cruzada de K=50 iteraciones.
- La función del "*kernel*" de tipo gaussiano.
- Escala del "*kernel*" =6.1.
- Estandarización de todos los datos.

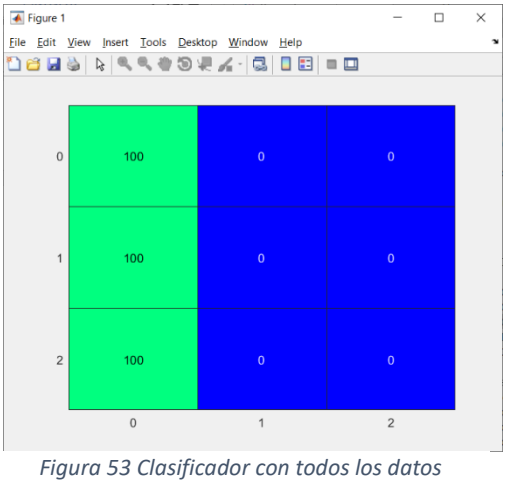


Figura 53 Clasificador con todos los datos

Llegado a este paso se ha realizado la prueba de entrenar el clasificador SVM directamente sin ningún tipo de estrategia, ver **Figura 53**. El resultado de esta prueba es simplemente desastroso. Como los datos que representan a los eventos son tan pocos el clasificador decide clasificar todo como 0 y al representar tan poco porcentaje del total decide que es la mejor estrategia. Y efectivamente en porcentajes es así, pero no es lo que se pretende en este trabajo.

Esta matriz de confusión es útil para saber que matriz de costes tenemos que indicarle al clasificador. Después de múltiples pruebas respecto a qué pesos asignarle se ha decidido que lo mejor es:

$$C = \begin{bmatrix} 0 & 1 & 1 \\ 100 & 0 & 1 \\ 100 & 1 & 0 \end{bmatrix}$$

Esto quiere decir que vamos a penalizar 100 veces más de lo normal cuando el clasificador prediga un 0 y la clase verdadera sea un 1. Lo mismo para cuando prediga un 0 y sea un 2. Con esta estrategia, vemos que es bastante más equilibrada, ver **Figura 54**. Pero si ponemos a prueba este clasificador, su capacidad de clasificación real con nuevos datos es muy pobre. En los círculos rojos de la **Figura 55** se puede ver que de 6 pasos solo ha clasificado 2 datos como evento TO. Lo que indica claramente que se necesitan más datos de entrenamiento para poder utilizar esta estrategia.

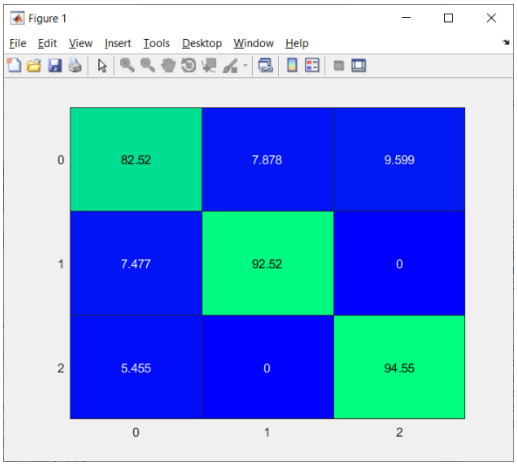


Figura 54 Matriz con función de coste

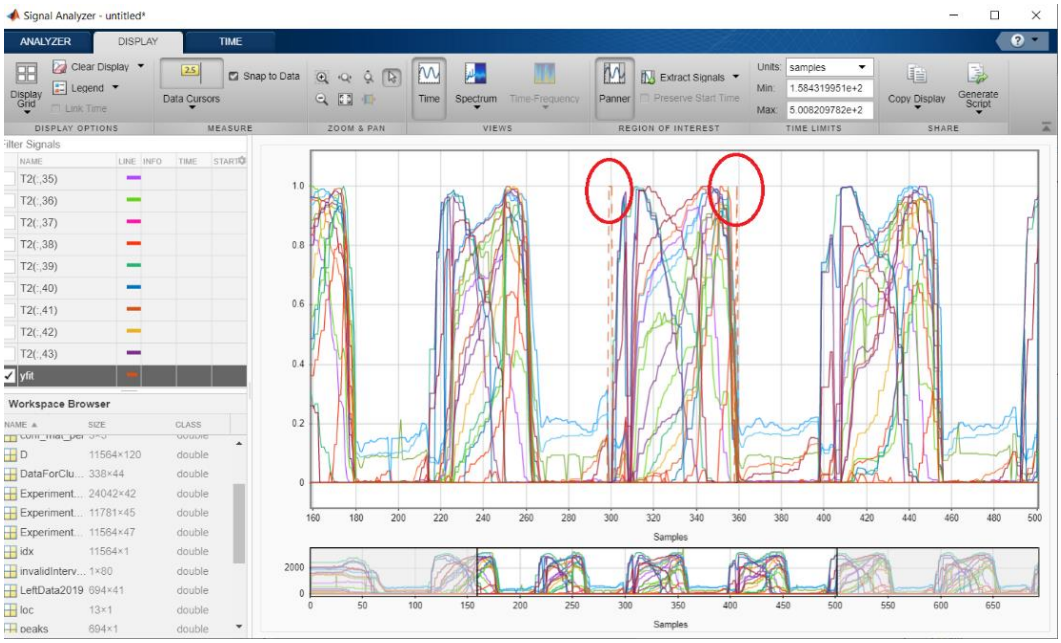


Figura 55 Clasificador con pesos

Estas dos estrategias son una buena opción cuando se tenga muchos más datos de clasificación y puede que sea mucho más efectivo a la hora de tener que introducir datos de personas con algún tipo de patología en la marcha.

Existe otra estrategia que implica realizar un tipo de trampa, pero siendo mucho más efectivo dado el problema actual. En lugar tener 3 clases o incluso 4 se ha observado en los distintos experimentos que siempre los eventos HS y TO van en orden. También están separados en el tiempo por la activación de los 32 sensores de presión. Teniendo esto en cuenta se puede considerar 2 clases:

- Clase 0, como la clase donde los sensores de presión no están activos y el pie humano está en el aire.
- Clase 3, como la clase que demarca el inicio de HS y el final de TO. Básicamente el intervalo en el que los sensores de presión están activos.

Nota: se ha utilizado la clase 3 debido a que hubo un intento de realizar un clasificador con 4 clases, indicando la clase 3 como la clase que hay entre la 1 y la 2. Esto facilita la utilización de código en Matlab ya que sigue siendo compatible con las distintas versiones. Pero al final siguen siendo solamente 2 clases identificadas con un 0 y un 3.

En base a esta nueva estrategia se ha entrenado otro clasificador con los resultados de la **Figura 56**. Como se puede ver los datos son muy prometedores. En parte se le ha facilitado "la vida" al clasificador ya que los clasificadores SVM por definición solo se pueden aplicar para 2 clases. Si se quiere hacer con múltiples clases se tienen que realizar comparaciones entre clases, lo que aumenta mucho el tiempo de clasificación y más aun dándole una función de costes para compensar el desajuste en los datos.

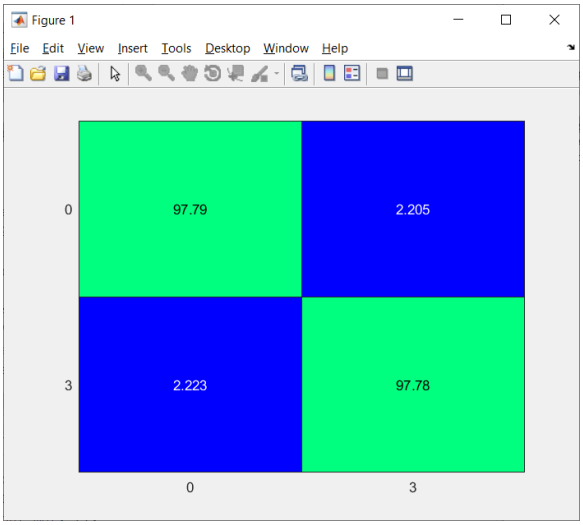


Figura 56. Clasificador con 2 clases

Estos resultados se han obtenido con las siguientes configuraciones:

- Kernel de tipo polinomial.
- Polinomios de orden 3.
- Matriz de coste: C = [0 1; 1.5 0];
- Estandarización de los datos.

Se ha aplicado una matriz de costes para equilibrar la matriz y sea muy parecida en los cuatro cuadrantes. En la **Figura 57** se puede ver el resultado de la clasificación, las líneas discontinuas naranjas indican un cambio de 0 a 3. Por otra parte, en la **Figura 58** se muestra el resultado de la clasificación y el máximo de cada serie de los sensores de presión. Así se puede ver mucho más claro cuáles son los resultados.

Sabiendo como se ha entrenado a este clasificador, podemos decir con mucha precisión, más del 97%, que cuando se produce un cambio de una clase 0 a clase 3 se ha producido un evento HS y cuando se realiza una transición de la clase 3 a la 0 es un evento TO. Teniendo en cuenta que se ha aplicado a datos de los cuales no se han utilizado en el entrenamiento el resultado es más que aceptable. Se puede apreciar que alrededor del número 300 (eje x) se produce un fallo de clasificación. A la hora de utilizar este clasificador se puede tomar una estrategia de tener en cuenta que hay bastante tiempo entre eventos, asignándole una probabilidad baja de ser una buena clasificación. Dicho de otra manera, en ningún caso un Evento TO es seguido por un evento HS en un periodo tan corto de tiempo. En un futuro post procesamiento se puede indicar unos márgenes aceptables y limpiar aún más la clasificación.

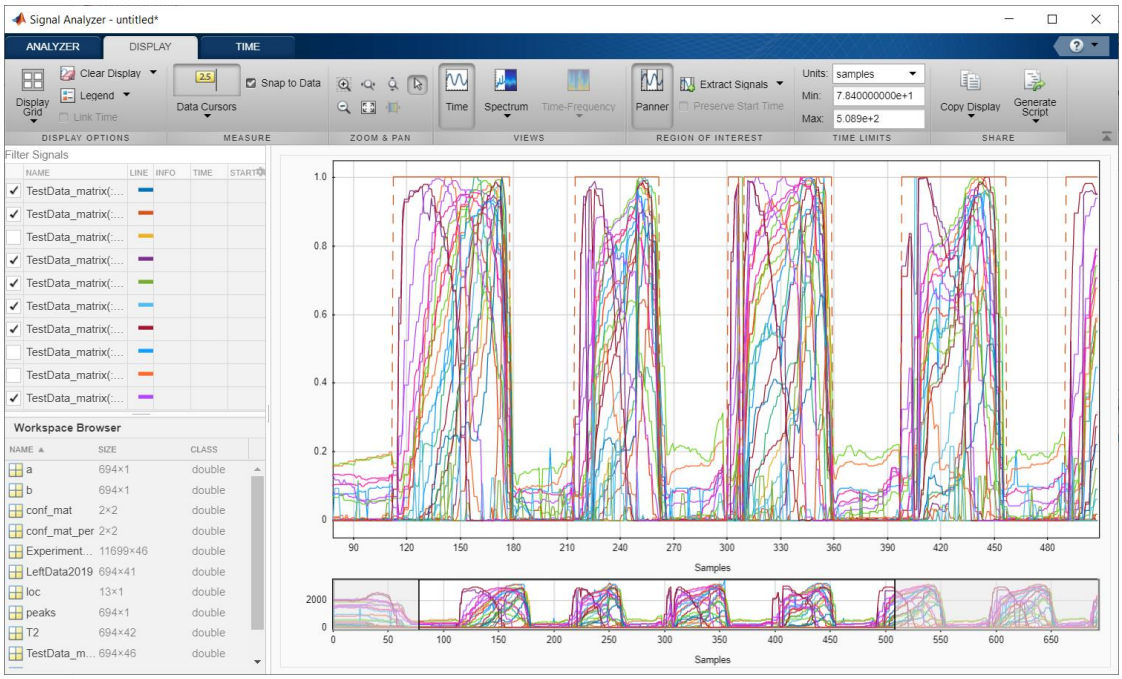


Figura 57 Test del clasificador con 2 clases

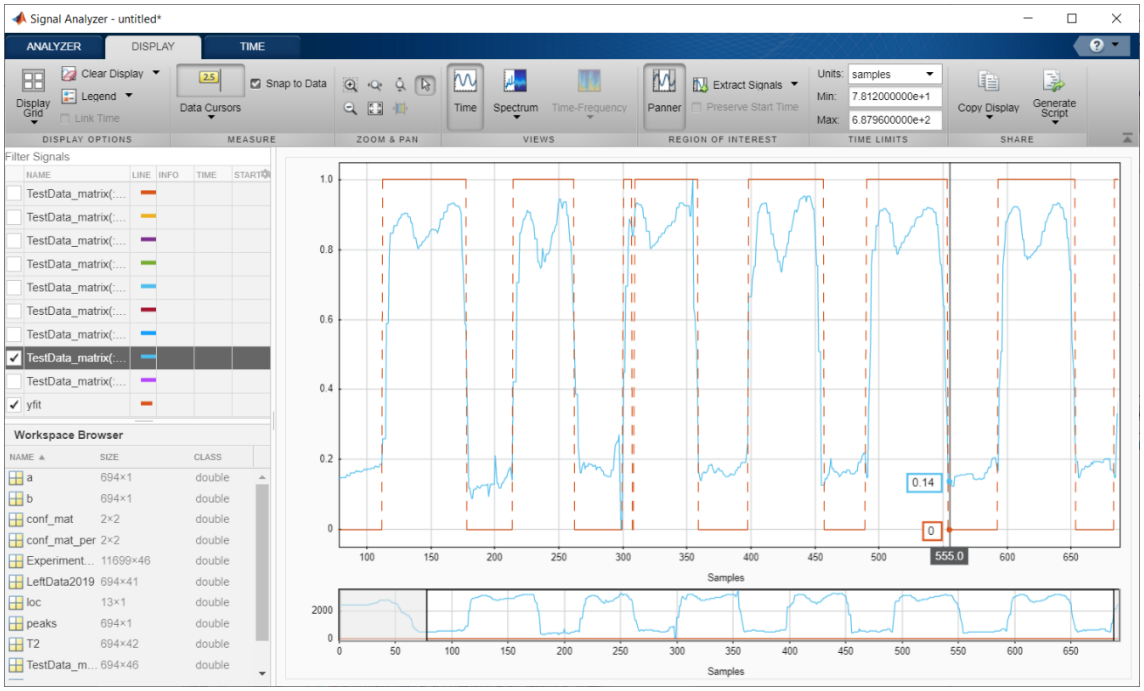


Figura 58 Clasificación con 2 clases sin muchas señales.

6. Conclusiones y trabajos futuros

En este capítulo se explicará de qué manera se han cumplido cada uno de los sub-objetivos propuestos. Por último, se expondrá una serie de trabajos futuros para poder seguir mejorando el prototipo de plantilla sensorizada y así poder llegar a un producto comercial.

El objetivo principal se ha cumplido: Adaptar el prototipo de la plantilla sensorizada, para una estrategia de adquisición de datos a largo plazo aumentando el número de sensores.

Para poder cumplir el objetivo principal se han definido una serie de sub-objetivos que se han completado en gran parte:

1. Ampliar a 32 los de sensores de presión de la plantilla.
2. Añadir un sistema de carga/descarga, protección y regulación de la batería.
3. Añadir un RTC "Reloj en tiempo real".
4. Introducir un método que permita almacenar los datos generados.
5. Realizar un diseño hardware modular para poder reconexiones componentes si es necesario.
6. Mejorar la adquisición de datos de los sensores de presión evitando interferencias entre ellos.
7. Mejorar el tiempo de adquisición de los datos generados por los sensores de presión.
8. Mejorar la manera en la que se adquiere el nivel de la batería de la plantilla.
9. Implementar un método de transferencia de datos masiva desde la plantilla a un dispositivo móvil por Bluetooth.
10. Introducir la lógica de funcionamiento de los RTC al firmware de la plantilla.
11. Realizar un primer análisis con un clasificador SVM de los datos generados.

Sub-objetivos cumplidos:

1. La **sección 2.2** "Aumento de sensores" se explica todas las modificaciones realizadas para poder permitir el aumento de sensores. En la **sección 3.2** "Lectura de los 32 sensores de presión" se explica las modificaciones software necesarias para su funcionamiento.
2. La **sección 2.3** "Gestión de la batería" se realizan múltiples cambios en el prototipo anterior tanto para poder gestionar mejor la batería como para poder depurar el SoC por el puerto USB.
3. La **sección 2.1** "Añadir la memoria Flash y el RTC. Explica que problemáticas se han encontrado al añadir el RTC y en la **sección 3.1** "Funcionamiento RTC" se explica el funcionamiento interno de este.
4. Este objetivo se cumple parcialmente dado que el funcionamiento de la memoria FLASH no se puede adaptar al software de la plantilla. La **sección 2.1** "Añadir la memoria Flash y el RTC" Explica las conexiones necesarias para añadir una memoria FLASH por SPI al SoC y en la **sección 3.3** "Funcionamiento de la Memoria FLASH".
5. Todo el **Capítulo 2** "Mejora del Hardware: es lo más permisivo posible. Cualquier error de conexión se puede corregir debido a que ningún componente está ligado a una conexión directa con otro. Todos se conectan a un conector el cual tiene su conector espejo en el dispositivo a conectar.

6. El propio diseño explicado en la **sección 2.2** “Aumento de sensores” hace que no exista ningún tipo de interferencia entre los distintos sensores.
7. Debido a la incorporación de un op-amp en el esquema de adquisición en la **sección 2.2** “Aumento de sensores” se mejora el tiempo de adquisición del valor de cada sensor.
8. La **sección 2.3** “Gestión de la batería” se muestra un método de lectura de la batería utilizando un op-amp en modo inversor y amplificador utilizando un ADC externo.
9. El **Capítulo 4** “Protocolos de comunicación” ofrece una alternativa de comunicación para poder transferir datos de manera masiva desde la plantilla al dispositivo móvil.
10. La **sección 3.1** “Funcionamiento RTC” explica la lógica de funcionamiento y como hay que introducir la lógica de funcionamiento al software de la plantilla.
11. El **Capítulo 5** “Análisis de datos de presión” realiza un primer análisis utilizando clasificadores SVM para poder interpretar los datos generados.

6.1. Trabajos Futuros

Las nuevas tareas descubiertas en esta iteración que se deben abordar en el siguiente prototipo son:

1. Explorar la posibilidad de añadir una memoria externa sin ningún tipo de gestión. Conectarla directamente al SoC para poder trabajar a un muy bajo nivel. Un buen ejemplo de inicio son las memorias FLASH NAND de la empresa Micron con hasta 8 bits de bus de datos, estos ofrecen librerías para poder utilizar estas memorias. Añadir un chip externo para aumentar la salida GPIO.
2. Explorar las diversas maneras de combinar los circuitos integrados con la distribución de los sensores en la suela de la pierna.
3. Es importante que el reloj de las dos plantillas estén lo más sincronizados posible para poder analizar las diferentes zancadas entre las dos piernas. Por ello se tiene que investigar más la manera de minimizar el error temporal entre los dos relojes.
4. Aunque el protocolo L2CAP se ha utilizado para enviar datos almacenados en memoria, este se podría utilizar también para una conexión en tiempo real entre el dispositivo móvil y las plantillas. Sería recomendable realizar un modo de funcionamiento de tiempo real utilizando L2CAP.

Bibliografía

[1] J. Kim, A. S. Campbell y J. Wang, «Wearable non-invasive epidermal glucose sensors: A review,» *Talanta,* vol. 177, p. 163–170, 1 2018.

[2] X. Wu, W. Hou, C. Peng, X. Zheng, X. Fang y J. He, «Wearable magnetic locating and tracking system for MEMS medical capsule,» *Sensors and Actuators A: Physical,* vol. 141, p. 432–439, 2 2008.

[3] Y.-L. Zheng, X.-R. Ding, C. C. Y. Poon, B. P. L. Lo, H. Zhang, X.-L. Zhou, G.-Z. Yang, N. Zhao y Y.-T. Zhang, «Unobtrusive Sensing and Wearable Devices for Health Informatics,» *IEEE Transactions on Biomedical Engineering,* vol. 61, p. 1538–1554, 5 2014.

[4] WHO, «World Health Organization,» [En línea]. Available: https://apps.who.int/gho/data/node.main.688?lang=en. [Último acceso: 5 1 2021].

[5] K. Strong, C. Mathers, S. Leeder y R. Beaglehole, «Preventing chronic diseases: how many lives can we save?,» *The Lancet,* vol. 366, p. 1578–1582, 10 2005.

[6] S. Kumar, W. J. Nilsen, A. Abernethy, A. Atienza, K. Patrick, M. Pavel, W. T. Riley, A. Shar, B. Spring, D. Spruijt-Metz, D. Hedeker, V. Honavar, R. Kravitz, R. C. Lefebvre, D. C. Mohr, S. A. Murphy, C. Quinn, V. Shusterman y D. Swendeman, «Mobile Health Technology Evaluation,» *American Journal of Preventive Medicine,* vol. 45, p. 228–236, 8 2013.

[7] S. Krishna, S. A. Boren y E. A. Balas, «Healthcare via Cell Phones: A Systematic Review,» *Telemedicine and e-Health,* vol. 15, p. 231–240, 4 2009.

[8] K. Patrick, W. G. Griswold, F. Raab y S. S. Intille, «Health and the Mobile Phone,» *American Journal of Preventive Medicine,* vol. 35, p. 177–181, 8 2008.

[9] W. T. Riley, D. E. Rivera, A. A. Atienza, W. Nilsen, S. M. Allison y R. Mermelstein, «Health behavior models in the age of mobile interventions: are our theories up to the task?,» *Translational Behavioral Medicine,* vol. 1, p. 53–71, 2 2011.

[10] «Biosensemedical Medical,» [En línea]. Available: https://biosensemedical.com/product/tekscan-f-scan-versatek-sensor/ . [Último acceso: 08 01 2021].

[11] S. Crea, M. Donati, S. D. Rossi, C. Oddo y N. Vitiello, «A Wireless Flexible Sensorized Insole for Gait Analysis,» *Sensors,* vol. 14, p. 1073–1093, 1 2014.

[12] S. Bamberg, A. Y. Benbasat, D. M. Scarborough, D. E. Krebs y J. A. Paradiso, «Gait Analysis Using a Shoe-Integrated Wireless Sensor System,» *IEEE Transactions on Information Technology in Biomedicine,* vol. 12, p. 413–423, 7 2008.

[13] P. Lopez-Meyer, G. D. Fulk y E. S. Sazonov, «Automatic Detection of Temporal Gait Parameters in Poststroke Individuals,» *IEEE Transactions on Information Technology in Biomedicine,* vol. 15, p. 594–601, 7 2011.

[14] H. M. Schepers, E. H. F. van Asseldonk, C. T. M. Baten y P. H. Veltink, «Ambulatory estimation of foot placement during walking using inertial sensors,» *Journal of Biomechanics,* vol. 43, p. 3138–3143, 12 2010.

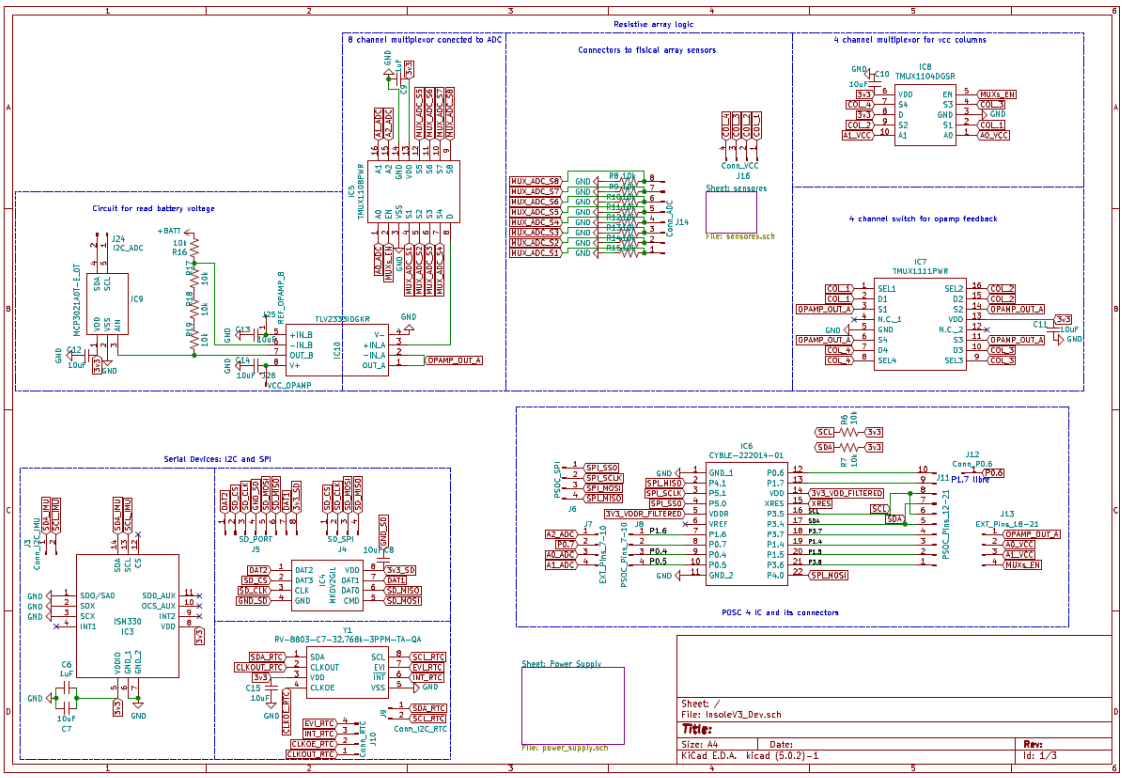
[15] D. C. Constantin, «UCLM,» [En línea]. Available: http://hdl.handle.net/10578/19071. [Último acceso: 20 01 2021].

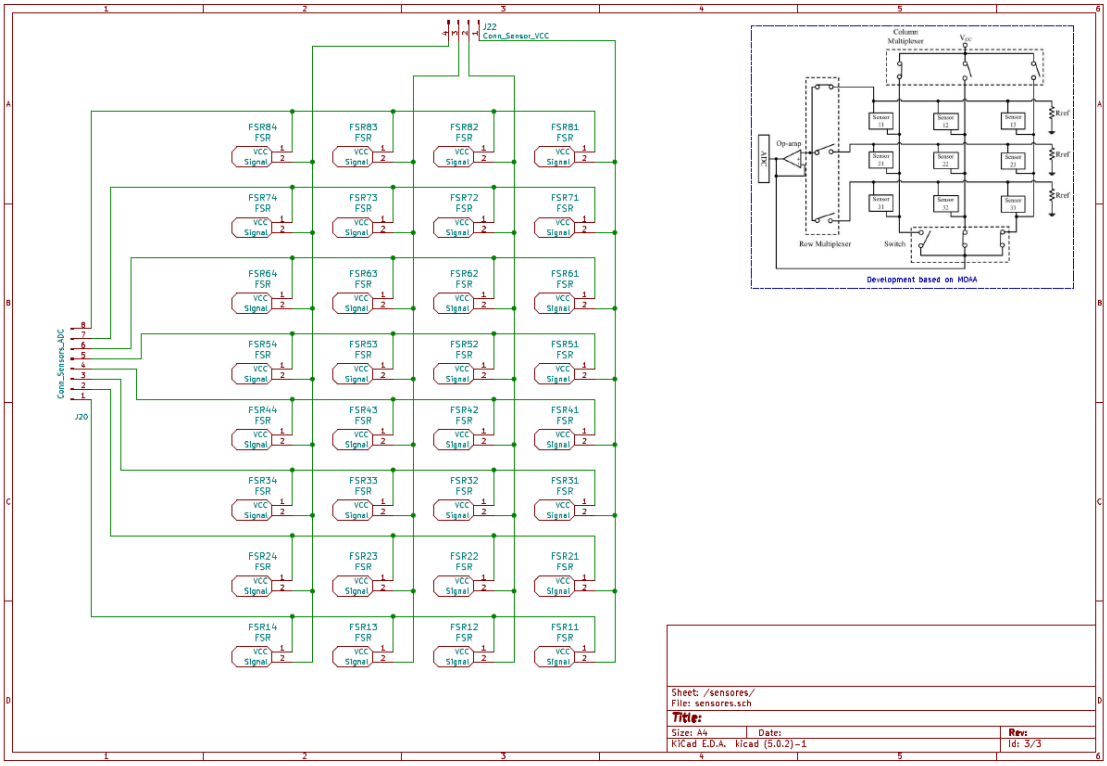
[16] L. Shu, X. Tao y D. D. Feng, «A New Approach for Readout of Resistive Sensor Arrays for Wearable Electronic Applications,» *IEEE Sensors Journal,* vol. 15, p. 442–452, 1 2015.

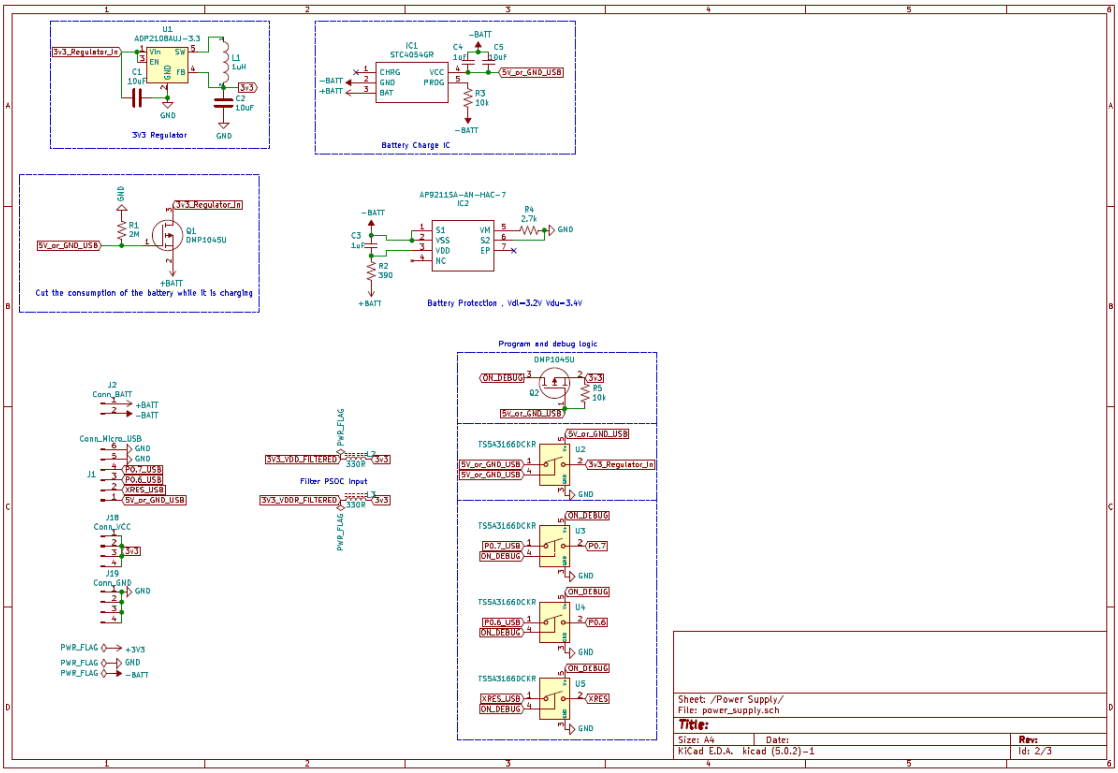
[17] A. England, «GitHub,» [En línea]. Available: https://github.com/sparkfun/SparkFun_RV-8803_Arduino_Library. [Último acceso: 1 2 2021].

[18] ChaN, «Elm-ChaN,» [En línea]. Available: http://elm-chan.org/fsw/ff/00index_e.html. [Último acceso: 28 01 2021].

Anexo 1 Diseño lógico PCB






3V3 Regulator
Battery Charge IC
Cut the consumption of the battery while it is charging
Battery Protection , Vdl=3.2V Vdu=3.4V
Program and debug logic
Filter PSOC Input
Sheet: /Power Supply/
File: power_supply.sch
Title:
Size: A4
Date:
KiCad E.D.A. kicad (5.0.2)-1
Rev:
Id: 2/3

Anexo 2: Organización de los registros del RTC

Register Definitions, Address 00h to 0Fh (Basic time and calendar register):

Address	Function	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
00h	Seconds	○	40	20	10	8	4	2	1
01h	Minutes	○	40	20	10	8	4	2	1
02h	Hours	○	○	20	10	8	4	2	1
03h	Weekday	○	6	5	4	3	2	1	0
04h	Date	○	○	20	10	8	4	2	1
05h	Month	○	○	○	10	8	4	2	1
06h	Year	80	40	20	10	8	4	2	1
07h	RAM	RAM data							
08h	Minutes Alarm	AE_M	40	20	10	8	4	2	1
09h	Hours Alarm	AE_H	GP0	20	10	8	4	2	1
0Ah	Weekday Alarm	AE_WD	6	5	4	3	2	1	0
	Date Alarm		GP1	20	10	8	4	2	1
0Bh	Timer Counter 0	128	64	32	16	8	4	2	1
0Ch	Timer Counter 1	GP5	GP4	GP3	GP2	2048	1024	512	256
0Dh	Extension Register	TEST	WADA	USEL	TE	FD		TD	
0Eh	Flag Register	○	○	UF	TF	AF	EVF	V2F	V1F
0Fh	Control Register	RESERVED		UIE	TIE	AIE	EIE	○	RESET
○ Read only. Always 0.									

Register Definitions, Address 10h to 1Fh (Extension register ①):

Address	Function	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
10h	100^{th} Seconds (Read Only)	80	40	20	10	8	4	2	1
11h	Seconds	○	40	20	10	8	4	2	1
12h	Minutes	○	40	20	10	8	4	2	1
13h	Hours	○	○	20	10	8	4	2	1
14h	Weekday	○	6	5	4	3	2	1	0
15h	Date	○	○	20	10	8	4	2	1
16h	Month	○	○	○	10	8	4	2	1
17h	Year	80	40	20	10	8	4	2	1
18h	Minutes Alarm	AE_M	40	20	10	8	4	2	1
19h	Hours Alarm	AE_H	GP0	20	10	8	4	2	1
1Ah	Weekday Alarm	AE_WD	6	5	4	3	2	1	0
	Date Alarm		GP1	20	10	8	4	2	1
1Bh	Timer Counter 0	128	64	32	16	8	4	2	1
1Ch	Timer Counter 1	GP5	GP4	GP3	GP2	2048	1024	512	256
1Dh	Extension Register	TEST	WADA	USEL	TE	FD		TD	
1Eh	Flag Register	○	○	UF	TF	AF	EVF	V2F	V1F
1Fh	Control Register	RESERVED		UIE	TIE	AIE	EIE	○	RESET
○ Read only. Always 0.									

Register Definitions, Address 20h to 2Fh (Extension register ②):

Address	Function	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
20h	100^{th} Seconds CP (Read Only)	80	40	20	10	8	4	2	1
21h	Seconds CP (Read Only)	○	40	20	10	8	4	2	1
2Ch	Offset	○	○	OFFSET					
2Fh	Event Control	ECP	EHL	ET		○	○	○	ERST
○ Read only. Always 0.									