CARRERA DE ESPECIALIZACIÓN EN SISTEMAS EMBEBIDOS

MEMORIA DEL TRABAJO FINAL

Firmware de dispositivo de adquisición de señales neurofisiológicas

Autor:
Ing. Leandro Ezequiel Arrieta

Director:
Dr. Ing. Diego Coulombie (UNLaM)

Jurados:
Mg. Ing. Eduardo Filomena (UNER)
Mg. Ing. Mara Fusco (UTN-FRH)
Mg. Ing. Pablo Slavkin (FIUBA)

Este trabajo fue realizado en la ciudad de Lomas de Zamora, entre junio de 2021 y agosto de 2022.

Resumen

La presente memoria describe el desarrollo del software de un dispositivo de adquisición de señales neurofisiológicas cuyo hardware fue desarrollado por el Grupo de Tecnología en Neuroseñales de la Universidad Nacional de La Matanza (GTN-UNLaM). Dentro de las características del dispositivo se destaca la posibilidad de adquirir señales de forma simultánea mediante 8 canales y la comunicación inalámbrica Bluetooth embebida en el propio dispositivo.

Para cumplimentar los objetivos se realizó un software embebido en C utilizando un sistema operativo en tiempo real y máquinas de estado finitas. Fueron fundamentales los conocimientos adquiridos en las materias programación de microcontroladores, testing de software en sistemas embebidos, sistemas operativos de tiempo real I y II y desarrollo de aplicaciones sobre sistemas operativos de propósito general. Los contenidos de estas materias se aplicaron en el software y en la herramienta de prueba desarrollada.

Agradecimientos

A completar más adelante

Índice general

Índice de figuras

Índice de tablas

Dedicado a mis hijos, Guillermina y Agustin

Capítulo 1

Introducción general

En este apartado se introducen los conceptos básicos sobre el dispositivo de adquisición de señales neurofisiológicas, sobre su estado del arte, sobre el marco normativo aplicado, y sobre la motivación, objetivos y alcance para llevar adelante este trabajo.

1.1. Dispositivo de adquisición de señales neurofisiológicas

El dispositivo de adquisición de señales neurofisiológicas (DASN) es un módulo de hardware y software embebido destinado a ser parte de un sistema electromédico. El desarrollo está enmarcado en una serie de proyectos de investigación del Grupo de Tecnología en Neuroseñales de la Universidad Nacional de La Matanza (GTN UNLAM) cuya finalidad es generar una plataforma de uso común para fabricantes locales, para investigación en las universidades y para el eventual desarrollo de nuevos productos y empresas de tecnología médica.

En la Figura 1.1 se puede ver un diagrama en bloques de un sistema electromédico que incorpora al módulo DASN. Forman parte también del sistema un estimulador que interactúa con el paciente y un equipo de registro donde se visualizan y guardan en formato digital los biopotenciales. Tanto el estimulador como el equipo de registro se comunican con el módulo DASN cuya función es adquirir señales eléctricas provenientes del sistema nervioso del paciente.

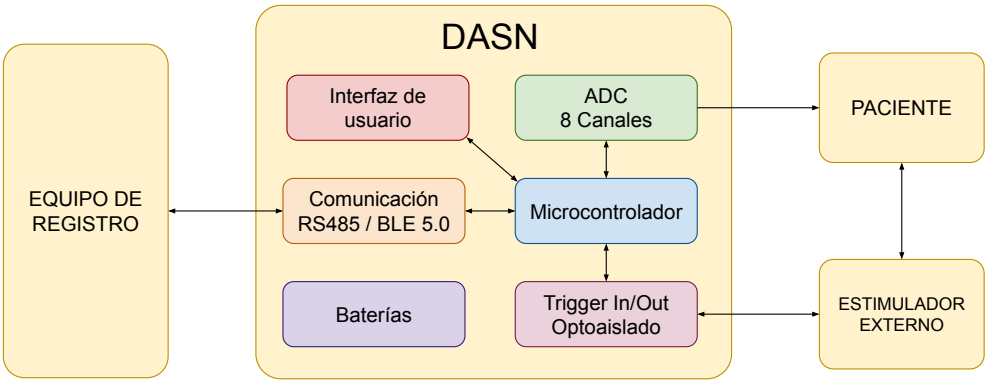

FIGURA 1.1. Diagrama en bloques del sistema.

El DASN tiene 8 canales de entrada, comunicación comunicación Bluetooth y RS485. Las señales adquiridas se pueden sincronizar con el estimulador externo, pudiendo funcionar como master o slave. Tiene la posibilidad de alimentarse por baterías y posee una interfaz de usuario para dar indicaciones de encendido, apagado y estado de funcionamiento.

1.2. Estado del arte

En particular para el subsector de la Neurofisiología, no existen placas comerciales que cumplan con las condiciones aplicables a un equipamiento médico de registro de señales neurofisiológicas. Si existen algunos ejemplos de placas de uso recreacional o didáctico, con un principio de funcionamiento similar pero destinado a interfaces cerebro-máquina (BCI). Ninguna de estas placas está destinada a la aplicación directa a la tecnología médica, ni posee certificaciones de calidad, ni garantiza características de funcionamiento esencial ni de seguridad básica.

1.3. Marco normativo

Las regulaciones de los dispositivos médicos tienen como objetivo proteger la salud de los pacientes, usuarios y terceros. Intentan asegurar la seguridad y eficacia de los productos disponibles en el mercado. Para cumplir con dichas regulaciones se debe actuar bajo las normas que son aceptadas por los organismos de control de los estados. Normas sobre sistemas de gestión de la calidad, sobre procesos y sobre productos alcanzan al diseño y la producción de equipamiento médico, incluido el software que lo conforma. La norma IEC 62304 [1] define las condiciones que debe cumplir el ciclo de vida del software; es decir, se trata de una norma de proceso. La IEC 60601-1 [2] (cuyo punto 14 se encarga del software del producto electro-médico o PEMS) y la IEC 82304-1 [3] son normas que definen las características que debe cumplir el software para que sea seguro y eficaz. La IEC 62304 no impone o prohíbe una metodología de desarrollo de software en particular, pero sí indica cuales son las características que debe tener el proceso de ciclo de vida elegido. Debe tratarse de un proceso controlado, que tenga en cuenta la verificación y validación del software durante el proceso de desarrollo, que provea evidencia documentada para demostrar el cumplimiento del proceso, que responda a una planificación, que tome en cuenta el análisis de los riesgos relacionados con el producto, entre otras consideraciones más. La norma no entra en detalle de cómo esas actividades deben ser ejecutadas, dejando al fabricante la libertad de crear sus propias prácticas para que sean coherentes con los principios regulatorios.

1.4. Motivación

La neurofisiología es una rama de las neurociencias, que se encarga del estudio funcional de la actividad bioeléctrica del sistema nervioso central, periférico y autonómico, mediante la utilización de equipos y técnicas de análisis avanzado, como la electroencefalografía (EEG), la electromiografía (EMG), los potenciales evocados (PE), la polisomnografía (PSG) y otras nuevas técnicas como el neuromonitoreo (NM) o la medición de profundidad de anestesia (MPA). La plataforma de adquisición de señales neurofisiológicas está pensada para cubrir las

necesidades de fabricantes de equipos del subsector de las neurociencias, para la investigación en las universidades y para el eventual desarrollo de nuevos productos y empresas de tecnología médica. En este marco nació la propuesta para dar vida al primer prototipo mediante la programación de su sistema embebido, con la motivación de aportar a la soberanía tecnológica y a la mejora del sistema de salud.

1.5. Objetivos y alcance

1.5.1. Objetivos del trabajo

El objetivo del presente trabajo final de carrera fue diseñar, implementar, ensayar y documentar la primera versión funcional del software embebido del DASN de acuerdo a los lineamientos impartidos en las distintas materias de la Especialización en Sistemas Embebidos de la Universidad de Buenos Aires.

1.5.2. Alcances del trabajo

El presente trabajo incluye:

- Diseño del software embebido del DASN.
- Diseño del protocolo de comunicación entre DASN y equipo de registro.
- Software de prueba para simular un equipo de registro y probar el DASN.

El presente trabajo no incluye:

- Diseño de hardware.
- Filtrado digital de las señales adquiridas.
- Validación de la norma ISO 62304.

Capítulo 2

Introducción específica

En este capítulo se hace mención a las partes constituyentes del dispositivo de adquisición de señales neurofisiológicas, se introduce a su hardware y al stack Bluetooth. Adicionalmente se muestran las herramientas de desarrollo utilizadas en el proyecto.

2.1. Hardware del DASN

El DASN está integrado en una sola placa y es un dispositivo portátil alimentado por baterías. Su arquitectura está basada en dos circuitos integrados principales, el microcontrolador CC2640R2 y *Analog front end* (AFE) ADS1299. Los AFE son dispositivos que incorporan la parte analógica para el acondicionamiento de la señal y el conversor analógico digital para poder digitalizar la señal. En la figura 2.1 se puede ver un diagrama en bloques del hardware.

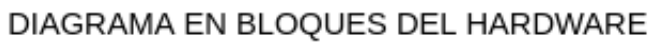

FIGURA 2.1. Diagrama en bloques del hardware.

2.1.1. Circuito de entrada

el circuito de entrada del DASN está basado en un circuito integrado ADS1299 [4], como se explicó en la sección 2.1 es un AFE. En particular el ADS1299 es un dispositivo de 8 canales, 24 bits por canal, conversión analógica digital sigma delta simultánea, con amplificadores de ganancia programable y un circuito multiplexor que permite conectar las entradas de múltiples formas que lo hacen ideal para el registro de biopotenciales y en particular de señales de EEG. En la figura 2.2 se puede ver el diagrama en bloques del ADS1299

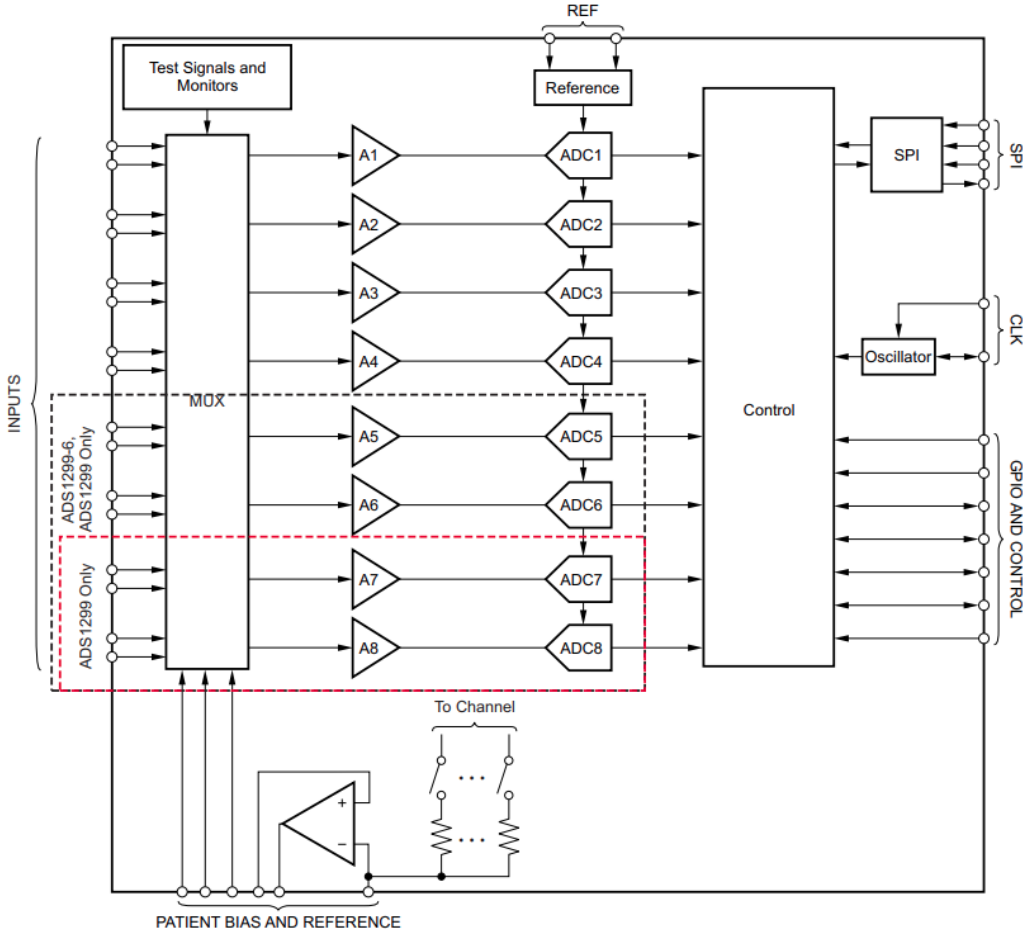

FIGURA 2.2. Diagrama en bloques del ADS1299 [1].

El ADS1299 posee un puerto de comunicación SPI. Adicionalmente se dispone de señales de control para poder configurar y controlar el dispositivo. El microcontrolador del DASN está conectado a todas estas interfaces de control del ADS1299.

El DASN posee un canal de potenciales evocados. Esta señal tiene una amplitud menor al ruido de entrada del ADS1299. Para poder medir este biopotencial el DASN incorpora un amplificador entre el paciente y el AFE para adaptar los

[1] Imagen tomada de la hoja de datos del ADS1299 [4].

niveles de la señal. El resto de los canales solo incorporan un filtro de radiofrecuencia pasivo.

Debido al amplificador extra que incorpora el canal de potenciales evocados, no se puede medir la impedancia de los electrodos de este canal con la señal de impedancia que genera el AFE ya que no se está en contacto con el paciente.

2.1.2. Microcontrolador y comunicación inalámbrica

El DASN posee un microcontrolador CC2640R2 con arquitectura Arm® Cortex®-M3. Este microcontrolador posee integrado un *tranceiver* de 2.4 GHz compatible con la tecnología Bluetooth de baja energía 5.1 y versiones anteriores del stack Bluetooth [5]. El hecho de tener este tipo de microcontrolador con el *tranceiver* integrado permite configurar el stack Bluetooth por completo, con lo que se logran mejores prestaciones en consumo y ancho de banda al poder elegir el protocolo de comunicación y el modo de trabajo. La placa del DASN posee una antena tipo F [6] integrada en el circuito impreso y también ofrece la posibilidad de conectar una antena externa.

2.1.3. Conceptos básicos del stack Bluetooth 5.0

El Bluetooth de baja energía, también conocido por sus siglas en inglés como BLE, se divide en distintas capas. La aplicación de usuario entra en contacto con las dos capas superiores del stack Bluetooth, la capa GAP y la capa GATT. A través de estas capas es que el software del DASN configura y maneja la comunicación BLE.

Capa GAP

La capa GAP del stack Bluetooth es responsable de las conexiones. De acuerdo a la definición de esta capa un dispositivo Bluetooth se puede encontrar en uno de los siguientes estados:

- *Standby*: el dispositivo se encuentra en el estado inactivo inicial al reiniciarse.
- *Advertising*: el dispositivo se anuncia con datos específicos que le permiten a cualquier dispositivo saber que es un dispositivo conectable. Este anuncio contiene la dirección del dispositivo y puede contener algunos datos adicionales, como el nombre del dispositivo.
- *Scanning*: se escuchan los mensajes enviados por algún *advertiser*.
- *Initiating*: se inicia la conexión, pero el *advertiser* es quien dependiendo los datos enviados por el *initiator* acepta o no la conexión.
- *Connected*: Cuando se establece una conexión entre ambos dispositivos.

A su vez, la capa GAP define cuatro roles para los dispositivos [ref]:

- *Broadcaster*: es un anunciante que no se puede conectar.
- *Observer*: busca anuncios pero no puede iniciar conexiones.
- *Peripheral*: es un anunciante que se puede conectar y funciona como *peripheral* en una conexión de una sola capa de enlace.

- *Central*: busca anuncios e inicia conexiones y funciona como central en una o varias conexiones.

Capa GATT

La capa GATT es usada por la aplicación para el envío y recepción de los datos. Los datos son pasados y almacenados desde la aplicación de usuario al *stack* Bluetooth en forma de características. La capa GATT define los siguientes roles para los dispositivos que se encuentran conectados:

- GATT *server*: Es el dispositivo que tiene el espacio de memoria para la información que va a ser leída o escrita por un GATT *client*.
- GATT *client*: Es el dispositivo que lee o escribe información de un GATT *server*.

Los roles de la capa GATT (*client* o *server*) son independientes de los roles de la capa GAP (*peripheral* o *central*). Un *peripheral* puede ser tanto un GATT *client* como un GATT *server*, y lo mismo ocurre con un *central*.

Es importante destacar que el protocolo de comunicación BLE funciona por conexiones. Cuando se tiene un nuevo dato no se transmite directamente sino que se deja lista para que el otro dispositivo que está conectado pueda ir a leerla.

GATT *characteristics* y *attributes*

Characteristics y *attributes* son dos términos muy usados cuando hablamos de la configuración Bluetooth. Las *characteristics* agrupan información llamada *attribute*. *Attribute* es la información que realmente se transfiere entre dispositivos. Las *characteristics* organizan y usan los *attributes* de distintas maneras, los pueden usar como campo de datos, propiedades y configuraciones.

GATT *services* y *profile*

Un GATT *service* es un conjunto de *characteristics*. Por ejemplo, los dispositivos para medir frecuencia cardíaca tienen un *service* de frecuencia cardíaca que contiene una *characteristic* para la medición de la frecuencia cardíaca y otra con la ubicación del dispositivo en el cuerpo, entre tantas otras. Se pueden agrupar varios *services* para formar un *profile*. Muchos *profile* solo implementan un *service*, por lo que los dos términos a veces se usan indistintamente.

2.1.4. Requerimientos

1. **Requerimientos de interfaces externas**

   1.1 El software deberá comunicarse con el equipo de registro utilizando una interfaz BLE 5.0.

   1.2 El software deberá comunicarse con el equipo de registro utilizando una interfaz RS485.

   1.3 El software deberá indicar mediante el led #1 que está transmitiendo las señales adquiridas de forma inalámbrica o cableada.

1.4 El software deberá indicar mediante el led #2 si está encendido o apagado. También deberá indicar con el mismo led #2 si entra en el modo pairing BLE.

1.5 El software deberá manejar una salida para sincronizar la adquisición con un estimulador externo (dispositivo externo como slave). La salida deberá poder configurarse entre normal bajo y normal alto. El pulso deberá poder configurarse entre 5 anchos de pulso diferentes (0,1 ms; 0,5 ms; 1 ms; 5 ms; 10 ms).

1.6 El software deberá manejar una entrada para sincronizar la adquisición con un estimulador externo (dispositivo externo como master). La detección deberá ser por flanco y se deberá poder configurar si el flanco es de subida o de bajada.

1.7 El software deberá manejar el pulsador que servirá para encender el dispositivo y para realizar el pairing BLE.

1.8 El software deberá generar la señal de impedancia para los canales de potenciales evocados.

1.9 El software deberá medir el estado de las baterías con el ADC interno del MCU.

2. **Requerimientos funcionales**

2.1 El software deberá configurar la frecuencia de muestreo de la señal adquirida entre 7 diferentes valores (250 Hz, 500 Hz, 1000 Hz, 2000 Hz, 4000 Hz, 8000 Hz y 16000 Hz).

2.2 El software deberá poder adquirir de 1 a 8 canales simultáneos.

2.3 Mediante comandos recibidos por BLE 5.0 o RS485 el software deberá poder iniciar y parar la adquisición.

2.4 El software deberá configurar la ganancia de amplificación de cada canal entre 7 diferentes valores (1, 2, 4, 6, 8, 12 y 24).

2.5 El software deberá configurar el ADC para usar una tipología de entrada diferencial o referencial.

2.6 El software deberá medir la impedancia de los electrodos con una señal de medición de impedancia cuya frecuencia deberá ser de 7,8 Hz o 31,2 Hz.

2.7 El software deberá seleccionar a qué electrodos le inyecta la señal de medición de impedancia.

2.8 El software deberá seleccionar para cada electrodo si usa la señal de impedancia generada por el ADC o la generada por el MCU.

2.9 El software deberá enviar la siguiente información de autodiagnóstico: temperatura del ADC, valor de las tensiones del ADC y frecuencia del clock del ADC.

2.10 El software deberá prender y apagar el equipo con una pulsación corta del botón, pulsación menor a 1 segundo.

2.11 El software deberá entrar en el modo de apareo de la comunicación BLE con una pulsación larga del botón, pulsación mayor a 4 segundos.

2.12 Luego de 1 minuto de inactividad, el software deberá entrar a un modo de bajo consumo de energía. Para esto deberá apagar el ADC, el transceiver RS485 y la alimentación de todos los periféricos externos al MCU.

2.13 El software deberá manejar una comunicación RS485 hasta 3 MBd.

2.14 El software deberá poder recibir comandos mientras está transmitiendo las señales adquiridas.

3. **Requerimientos de documentación**

3.1 La documentación debe cumplir los requisitos de la norma ISO 62304.

3.2 Toda la documentación del proyecto se almacenará bajo un sistema de control de versiones GIT.

3.3 Toda la documentación del código se realizará utilizando la herramienta Doxygen.

3.4 Se deberá realizar un informe de avance del proyecto en el séptimo mes de trabajo

3.5 Se deberá realizar la memoria técnica del trabajo final con la plantilla elaborada por la cátedra de gestión de proyecto.

2.1.5. Herramientas de desarrollo

Para el desarrollo del firmware se utilizó el entorno de desarrollo integrado (IDE por sus siglas en inglés de Integrated Development Environment) Code Composer Studio (CCS) version: 10.4.0.00006 y el kit de desarrollo de software (SDK por sus siglas en inglés de Software Development Kit) SimpleLink CC2640R2 SDK: 5.10.00.02 ambos de Texas Instrument.

Como plataforma de hardware se dispuso de 2 placas de desarrollo LAUNCHXL-CC2640R2 [7]. En la figura 2.3 podemos ver una de estas placas. Estas placas fueron de mucha utilidad para comenzar a familiarizarse con el dispositivo y las herramientas.

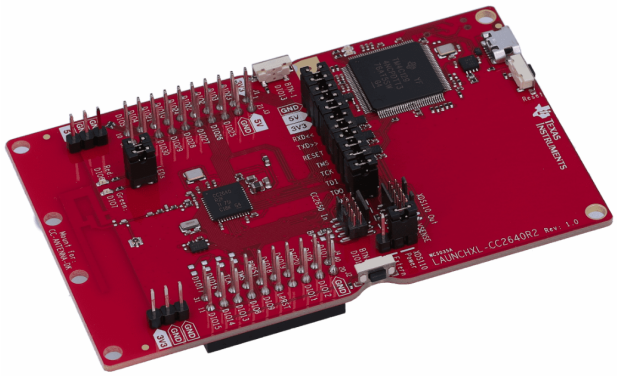

FIGURA 2.3. Kit de desarrollo launchpad CC2640R2.

Capítulo 3

Diseño e implementación

En este capítulo se describe la arquitectura de la aplicación, se muestra cómo interaccionan los distintos procesos entre sí y se explican las estrategias utilizadas para cumplir los requerimientos.

3.1. Arquitectura del sistema

La arquitectura esta basada en el sistema operativo TI-RTOS (*Texas Instrument Real Time Opertation System*) propiedad de texas instrument. Esta decisión se tomó ya que el SDK provee una API para comunicarse con el stack Bluetooth basada en este sistema operativo. Si bien creando una capa de abstracción de hardware (HAL) es posible utilizar otros sistemas operativos más generales como el freeRTOS (utilizado en las materias de sistemas operativos en tiempo real I y II), esta tarea no fue contemplada en la planificación ya que no era un requisito para este trabajo. Es una buena práctica no quedar fuertemente ligado a la plataforma de hardware utilizada pero en este caso el tiempo de desarrollo requerido para no utilizar el TI-RTOS no justifica los beneficios.

El stack Bluetooth se encuentra en una biblioteca la cual Texas Instrument no provee su código fuente por temas de política de la compañía. Como se ha dicho previamente provee una API para poder comunicarse con el stack Bluetooth. El stack requiere funcionar en la tarea del sistema operativo de mayor prioridad. Para el correcto funcionamiento del sistema es muy importante que ninguna tarea que hagamos sea bloqueante ya que esto provocaría fallos en la comunicación Bluetooth.

Se planteó una arquitectura modular, donde cada módulo encapsule sus funciones. En la figura 3.1 podemos ver un diagrama en bloques general donde cada bloque representa un módulo de software.

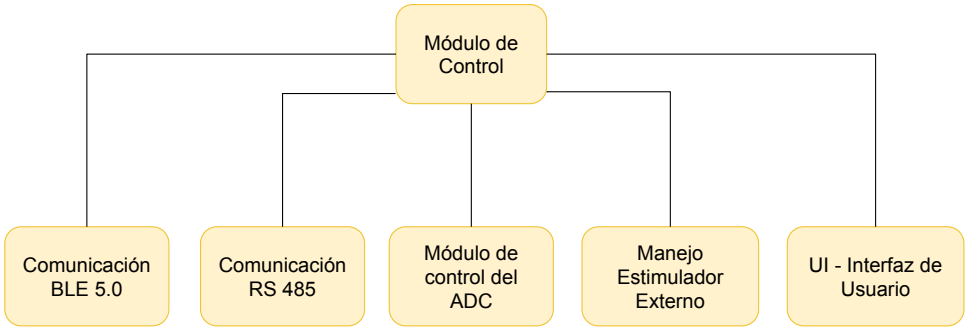

FIGURA 3.1. Diagrama en bloques de la arquitectura de software.

3.2. Módulo de control

Este módulo es el módulo principal, esta desarrollado en los archivos DASN_app.c y DASN_app.h. Corre en una tarea del sistema operativo y es el encargado de inicializar el hardware y los distintos módulos. Una vez inicializado queda a la espera de recibir eventos a través de la API Event_pend(), la cual es una función no bloqueante. Se lee el evento recibido y se ejecuta la tarea correspondiente.

3.3. Módulo BLE

Como se explicó previamente en la sección 3.1, a través de una API vamos a hacer uso del stack Bluetooth. A continuación se detallan las configuraciones principales que se llevaron a cabo para el desarrollo del DASN.

3.3.1. Configuración capa GAP

Como vimos en la sección 2.1.3 se le debe asignar un rol al DASN dentro de la capa GAP. El DASN por ser un dispositivo portable necesita anunciarse al resto de los dispositivos y entablar una conexión para transmitir los datos adquiridos. El rol que cumple con estos requisitos es el de un *peripheral*. Para configurar al dispositivo como un *peripheral* en la inicialización del módulo BLE debemos hacer uso de la siguiente API:

- GAP_DeviceInit() con el parámetro GAP_PROFILE_PERIPHERAL.

3.3.2. Parámetros de conexión

En esta sección se describen los parámetros de la conexión. Los parámetros de la conexión son enviados por el dispositivo iniciador de la misma, en nuestro caso el equipo que inicia la conexión siempre es el equipo de registro. Para asegurar un correcto funcionamiento no se puede dejar a elección del equipo de registro los parámetros de conexión. El DASN hace *streaming* de datos, lo que requiere un ajuste exacto de los tiempos. Debe ser el propio DASN quien elija los parámetros de conexión, y esto está contemplado por el stack Bluetooth. Luego de establecerse la conexión el DASN solicita al otro dispositivo cambiar los parámetros de conexión. En el código 3.1 se muestran los parámetros y la API para actualizar los parámetros en el *stack* Bluetooth.

```
static void DASN_sendParamUpdate(uint16_t connHandle)
{
    gapUpdateLinkParamReq_t req;

    req.connectionHandle = connHandle;
    req.connLatency = SLAVE_LATENCY;
    req.connTimeout = CONN_TIMEOUT;
    req.intervalMin = MIN_CONN_INTERVAL;
    req.intervalMax = MAX_CONN_INTERVAL;

    // Send parameter update
    bStatus_t status = GAP_UpdateLinkParamReq(&req);
}
```

CÓDIGO 3.1. Parámetros de conexión

A continuación se describen los parámetros y en la tabla 3.1 se muestran los valores asignados a cada uno de ellos:

- *Connection interval*: es el tiempo entre las conexiones para enviar datos. Este parámetro tiene unidades de 1,25 ms. Acepta valores entre 6 (7,5 ms) y 3200 (4 s).
- *Peripheral latency*: representa el número máximo de eventos de conexión que un dispositivo se puede saltear. Sirve para minimizar el consumo si no se tiene nueva información a enviar, pero del otro lado de la conexión se tarda más tiempo en detectar un error ya que no se sabe si el paquete se perdió o no fue enviado hasta que se cumpla el tiempo de latencia. Acepta valores entre 0 y 499.
- *Supervision time-out*: Es el tiempo máximo entre dos eventos de conexión. Si se cumple este tiempo sin que ocurra un evento de conexión, el dispositivo vuelve a un estado desconectado. Este parámetro tiene unidades de 10ms. Acepta valores entre 10 (100 ms) y 3200 (32 s).

TABLA 3.1. Parámetros de conexión configurados

Parámetro	**Valor**	**Observaciones**
MIN_CONN_INTERVAL	6	Se elije el mínimo valor para maximizar el ancho de banda
MAX_CONN_INTERVAL	6	Se hace igual a MIN_CONN_INTERVAL para asegurar el uso de este valor y que no se negocie otro con el *client*
SLAVE_LATENCY	0	No se permite saltear ninguna conexión.
CONN_TIMEOUT	200	En unidades de 10 ms da 2 s.

3.3.3. Configuración capa GATT

Un dispositivo en la capa GATT puede funcionar como *client* o *server*. Al inicializar el DASN se configura en el stack como *server* ya que dispondrá de las *characteristics* a ser leídas o escritas por el equipo de registro.

Con la API GGS_SetParameter() y los parámetros correspondientes se configuró el nombre del dispositivo en el servicio GAP GATT. El código 3.2 es un extracto del código de inicialización donde vemos como se configura el nombre del dispositivo.

```
1 //Largo del atributo nombre del dispositivo
2 #define GAP_DEVICE_NAME_LEN    12
3
4 //Nombre del dispositivo
5 static uint8_t attDeviceName[GAP_DEVICE_NAME_LEN] = "DASN V.1.0.0";
6
7 //GGS_DEVICE_NAME_ATT indica que vamos a setear el nombre del
      dispositivo
8
9 GGS_SetParameter(GGS_DEVICE_NAME_ATT,GAP_DEVICE_NAME_LEN, attDeviceName);
```

CÓDIGO 3.2. Configuración nombre del dispositivo

3.3.4. *Services* Bluetooth configurados

El DASN se desarrolló con un único *service*. Se le dió el nombre DATA_SERVICE el cual se compone de tres *characteristics*

- DS_CMD_RCV: usada para recibir comandos en el DASN.
- DS_CMD_SND: usada para enviar comandos desde el DASN.
- DS_STREAM: usada para enviar el paquete de datos desde el DASN.

Cada *service* y *characteristic* requiere un número de identificación (UUID). En el código 3.3 se puede ver un extracto del archivo *data_service.h* que muestra la configuración de los UUID para el *service* del DASN.

```
65 // Service UUID
66 #define DATA_SERVICE_SERV_UUID 0x1130
67
68 // Cmd rcv Characteristic defines
69 #define DS_CMD_RCV_ID                  0
70 #define DS_CMD_RCV_UUID                0x1131
71 #define DS_CMD_RCV_LEN                 1
72 #define DS_CMD_RCV_LEN_MIN             0
73
74 // Cmd snd Characteristic defines
75 #define DS_CMD_SND_ID                  1
76 #define DS_CMD_SND_UUID                0x1132
77 #define DS_CMD_SND_LEN                 1
78 #define DS_CMD_SND_LEN_MIN             0
79
80 // Stream Characteristic defines
81 #define DS_STREAM_ID                   2
82 #define DS_STREAM_UUID                 0x1133
83 #define DS_STREAM_LEN                  27
84 #define DS_STREAM_LEN_MIN              0
```

CÓDIGO 3.3. Configuración UUID

3.3.5. Tabla de *attributes*

Los *attributes* se deben declarar en una estructura que Texas llama tabla de *attributes*, en el código 3.4 podemos ver un extracto del archivo *data_service.c* en donde

vemos la declaración del *service* y de las *characteristics*. Podemos destacar que la *characteristic* DS_CMD_RCV tiene permiso de escritura (GATT_PERMIT_WRITE) ya que el dispositivo *client* tiene que poder escribir en esta *characteristic* el comando a enviar. En cambio, las *characteristics* DS_CMD_SND y DS_STREAM tienen solo permiso de lectura (GATT_PERMIT_READ).

```
153 static gattAttribute_t Data_ServiceAttrTbl[] =
154 {
155     // Data_Service Service Declaration
156     {
157         { ATT_BT_UUID_SIZE, primaryServiceUUID },
158         GATT_PERMIT_READ,
159         0,
160         (uint8_t *)&DataServiceDecl
161     },
162     // Cmd Rcv Characteristic Declaration
163     {
164         { ATT_BT_UUID_SIZE, characterUUID },
165         GATT_PERMIT_READ,
166         0,
167         &ds_CmdRcvProps
168     },
169     // Cmd Rcv Characteristic Value
170     {
171         { ATT_UUID_SIZE, ds_CmdRcvUUID },
172         GATT_PERMIT_WRITE,
173         0,
174         ds_CmdRcvVal
175     },
176     // Cmd Snd Characteristic Declaration
177     {
178         { ATT_BT_UUID_SIZE, characterUUID },
179         GATT_PERMIT_READ,
180         0,
181         &ds_CmdSndProps
182     },
183     // Cmd Snd Characteristic Value
184     {
185         { ATT_UUID_SIZE, ds_CmdSndUUID },
186         GATT_PERMIT_READ,
187         0,
188         ds_CmdSndVal
189     },
190     // CmdSnd CCCD
191     {
192         { ATT_BT_UUID_SIZE, clientCharCfgUUID },
193         GATT_PERMIT_READ | GATT_PERMIT_WRITE,
194         0,
195         (uint8_t *)&ds_CmdSndConfig
196     },
197     // Stream Characteristic Declaration
198     {
199         { ATT_BT_UUID_SIZE, characterUUID },
200         GATT_PERMIT_READ,
201         0,
202         &ds_StreamProps
203     },
204     // Stream Characteristic Value
205     {
206         { ATT_UUID_SIZE, ds_StreamUUID },
207         GATT_PERMIT_READ | GATT_PERMIT_WRITE,
208         0,
```

```
209             ds_StreamVal
210         },
211         // Stream CCCD
212         {
213             { ATT_BT_UUID_SIZE, clientCharCfgUUID },
214             GATT_PERMIT_READ | GATT_PERMIT_WRITE,
215             0,
216             (uint8_t *)&ds_StreamConfig
217         },
218 };
```

CÓDIGO 3.4. Tabla de *attributes*

Otro item a destacar de la tabla de *attributes* es que tanto el DS_CMD_SND y DS_STREAM tienen un *attribute* CCCD (de sus siglas en ingles *Client Characteristic Configuration Descriptor*). Este *attribute* sirve para que la *characteristic* se pueda configurar para que el GATT *server* envíe una notificación al GATT *client* cuando haya cambiado el valor de una *characteristic*. Esta es una configuración muy importante ya que determina la forma de funcionamiento del DASN. El DASN cuando tiene un nuevo dato para enviarle al equipo de registro escribe la *characteristic*, ya sea el dato adquirido o un comando de respuesta, con el valor correspondiente y gracias a esta configuración es que el equipo de registro se entera que hay un nuevo dato.

3.4. Módulo de adquisición

Este módulo es el encargado de manejar la comunicación SPI con el ADS1299. Corre en una tarea independiente del sistema operativo y se comunica con el módulo de control a través de una cola para pasar los datos adquiridos.

3.4.1. Máquina de estados

En la figura 3.2 se puede ver el diagrama en bloques de la máquina de estados del módulo.

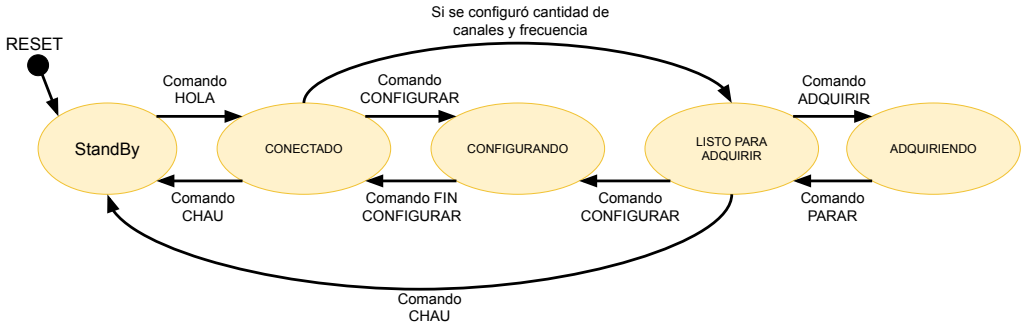

FIGURA 3.2. Diagrama en bloques de la máquina de estados del modulo de adquisición.

A continuación se describen los estados:

- STANDBY: estado inicial al reset.
- CONECTADO: se encienden las fuentes de alimentación y queda a la espera de estar configurado para poder pasar al estado LISTO PARA ADQUIRIR.
- CONFIGURANDO: se configura frecuencia de muestreo, canales a adquirir y señal de impedancia.
- LISTO PARA ADQUIRIR: se encuentra configurado y listo para adquirir.
- ADQUIRIENDO: lee los datos del ADS1299 por SPI y los envía a través de una cola de mensajes al módulo de control.

3.4.2. Protocolo de comunicación

La comunicación entre el *server* y el *client* se realiza leyendo y escribiendo en las *characteristics* DS_CMD_SND y DS_CMD_RCV. En la tabla 3.2 se puede ver la lista de comandos con su descripción.

TABLA 3.2. Tabla de comandos

Nombre	Valor	Descripción
HOLA	0x01	Inicia la comunicación
OK	0x02	Comando ejecutado
NO_OK	0x03	No se pudo ejecutar el comando
CHAU	0x04	Termina la comunicación
CONFIG_INICIAR	0x05	Inicia configuración
CONFIG_TERMINAR	0x06	Termina configuración
CONFIG_CH_ALL_ON	0x07	Habilita todos los canales
CONFIG_CH_ALL_OFF	0x08	Deshabilita todos los canales
CONFIG_CH1_ON	0x09	Canal 1 habilitado
CONFIG_CH2_ON	0x0A	Canal 2 habilitado
CONFIG_CH3_ON	0x0B	Canal 3 habilitado
CONFIG_CH4_ON	0x0C	Canal 4 habilitado
CONFIG_CH5_ON	0x0D	Canal 5 habilitado
CONFIG_CH6_ON	0x0E	Canal 6 habilitado
CONFIG_CH7_ON	0x0F	Canal 7 habilitado
CONFIG_CH8_ON	0x10	Canal 8 habilitado
CONFIG_FREC_1	0x11	Configura ADC con frecuencia 1 (16 kHz)
CONFIG_FREC_2	0x12	Configura ADC con frecuencia 2 (8 kHz)
CONFIG_FREC_3	0x13	Configura ADC con frecuencia 3 (4 kHz)
CONFIG_FREC_4	0x14	Configura ADC con frecuencia 4 (2 kHz)
CONFIG_FREC_5	0x15	Configura ADC con frecuencia 5 (1 kHz)
CONFIG_FREC_6	0x16	Configura ADC con frecuencia 6 (500 Hz)
CONFIG_FREC_7	0x17	Configura ADC con frecuencia 7 (250 Hz)
ADQUIRIR	0x18	Inicia la adquisición
PARAR	0x19	Para la adquisición
LEER_ESTADO	0x1A	Pregunta en que estado se encuentra
WAKE_UP	0x1B	Actualiza la máquina de estados
ZSIGNAL_31_2	0x1C	Habilita señal de impedancia de 31,2 Hz

ZSIGNAL_7_8	0x1D	Habilita señal de impedancia de 7,8 Hz
ZSIGNAL_OFF	0x1E	Deshabilita señal de impedancia
RESET	0x1F	Reinicia el dispositivo

Los datos adquiridos se transmiten con la *characteristic* DS_STREAM. Tiene un tamaño de 27 bytes, los primeros 3 bytes corresponden a un canal *status* y a continuación los 8 canales con 3 bytes por canal. En el canal de *status* se tiene la información de si algún electrodo se desconectó y también del estad de los cuatro GPIOs que dispone el ADS1299.

3.4.3. Configuración del puerto SPI

El puerto SPI se configura en modo *callback*, en este modo la comunicación no es bloqueante. Al momento de inicializar el puerto SPI se carga con un puntero a la función de *callback* que se ejecutará al terminarse una transacción de lectura o escritura del puerto. La tabla 3.3 muestra la configuración del puerto.

TABLA 3.3. Configuración SPI

Parámetro	Valor	Descripción
bitRate	4000000	Frecuencia del clock del puerto, ver 3.4.3
dataSize	8	Cantidad de bits por palabra
frameFormat	SPI_POL0_PHA1	Acorde a hoja de datos del ADS1299 [4]
mode	SPI_MASTER	El CC2640R2 maneja el clock
transferMode	SPI_MODE_CALLBACK	Modo *callback*
transferCallbackFxn	transferCallback	Puntero a la función de *callback*

Frecuencia del clock

En la figura 3.3 se muestra la trama de datos de salida del ADS1299. La trama de datos siempre es de 216 bits, no importa si hay canales apagados o no. Esto es así ya que los canales apagados se leen en cero. Considerando que la mayor frecuencia de muestreo es 16 kHz, la frecuencia mínima del clock del SPI debe ser:

$$ClockSPI > 216 * 16kHz = 3,456MHz \tag{3.1}$$

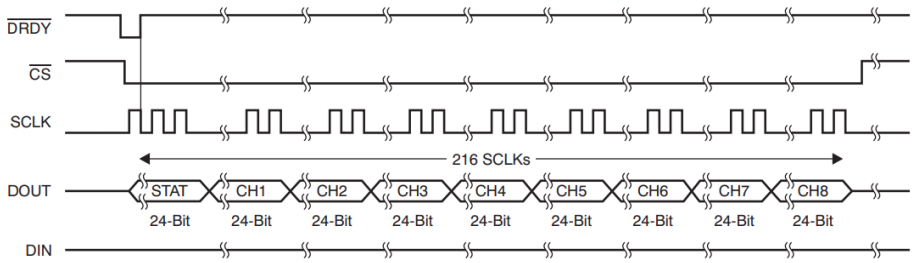

FIGURA 3.3. SPI BUS *data output* [1].

Como se puede ver en la la tabla 3.3 se elige una frecuencia de 4 MHz

[1]Imagen tomada de la hoja de datos del ADS1299 [4].

3.4.4. Configuración ADS1299

En la tabla 3.4 se muestran los comandos que se le pueden enviar al ADS1299.

TABLA 3.4. Comandos ADS1299

Comando	Valor	Descripción
WAKEUP_CMD	0x02	Sale del modo *stand by*.
STANDBY_CMD	0x04	Entra en el modo *stand by*.
RESET_CMD	0x06	Reset del dispositivo.
START_CMD	0x08	Comienzo y sincronización de la conversión.
STOP_CMD	0x0A	Detiene la conversión.
RDATAC_CMD	0x10	Habilita el modo de lectura continua de datos.
SDATAC_CMD	0x11	Detiene la lectura continua de datos.
RDATA_CMD	0x12	Lee un *stream* de datos.
RREG _CMD	0x2x	Lee n registros comenzando por la dirección especificada en los 5 bits menos significativos del comando. La cantidad n de registros a leer se especifica en el siguiente byte enviado.
WREG_CMD	0x4x	Escribe n registros, registro y cantidad idem que RREG_CMD.

Para configurar y utilizar el ADS1299 se crearon funciones para este fin. En el código 3.5 vemos la función para escribir en un registro interno que es parte del archivo *DASN_ADS1299.c*. Esta función recibe como parámetro el registro y el valor a escribir y hace uso de los comandos mostrados en la tabla 3.4.

```
524 /**
525  * @brief   Escribe en un registro interno del ads1299
526  *
527  * @param   registro    : Registro a escribir del tipo registers_t.
528  * @param   data        : Palabra a escribir en el registro
529  */
530 void ads1299_writeRegister(registers_t registro, uint8_t data)
531 {
532     // Se configura la transaccion
533     ads1299transaction.count = 5;
534     ads1299transaction.txBuf = txbuf;
535     ads1299transaction.rxBuf = rxbuf;
536     *(transaction_t*)(ads1299transaction.arg) = WRITE_REG;
537 
538     ((uint8_t*)ads1299transaction.txBuf)[0] = SDATAC_CMD;
    ///< Necesario si esta en RDATAC mode
539     ((uint8_t*)ads1299transaction.txBuf)[1] = (WREG_CMD | registro);
    ///< Se hace la OR para tener el valor del comando compuesto
540     ((uint8_t*)ads1299transaction.txBuf)[2] = NULL_CMD;
    ///< Indica que se lee un solo registro
541     ((uint8_t*)ads1299transaction.txBuf)[3] = data;
542 
543     ((uint8_t*)ads1299transaction.txBuf)[4] = NULL_CMD;
544 
545     //inicio transferencia por SPI
546     SPI_transfer(ads1299handle, &ads1299transaction);
547 
548 }
```

CÓDIGO 3.5. Función para escribir un registro del ADS1299.

Luego del reset el módulo de adquisición inicia en el estado STANDBY, en este estado se configuran los registros CONFIG3, CH1SET al CH8SET.

En el código 3.6 podemos ver el llamado a la función ads1299_writeRegister para escribir el registro CONFIG3 y los registros correspondientes a los 8 canales, en donde podemos ver que el dato a escribir está formado por la suma de *defines*, estos definidos en el archivo *DASN_ADS1299.h*.

```
198  ads1299_writeRegister(CONFIG3 ,
199          ADS1299_CONFIG3_INTERNAL_REF_BUF_EN |
200          ADS1299_CONFIG3_BIASREF_SIGNAL_INTERNALLY |
201          ADS1299_CONFIG3_BIAS_BUFFER_ENABLE
202          );
203
204  reg = CH1SET;
205  for(uint8_t i=0; i<8 ; i++)
206    {
207    ads1299_writeRegister(reg,
208             ADS1299_CH_N_SET_SETUP_NO |
209             ADS1299_CH_N_PGA_GAIN_1 |
210             ADS1299_CH_N_SRB2_OPEN |
211             ADS1299_CH_N_NORMAL_ELECTRODE_INPUT
212                      );
213    reg++;
214    }
```

CÓDIGO 3.6. Configuración de CONFIG3.

El procedimiento para escribir en los registros internos del ADS1299 es siempre el mismo. Por ejemplo la señal de impedancia se configura escribiendo en el registro LOFF cuando se recibe el comando ZSIGNAL_31_2, ZSIGNAL_7_8 o ZSIGNAL_OFF (ver lista de comandos en tabla 3.2) siempre y cuando el módulo se encuentre en el estado CONFIGURANDO, de lo contrario el pedido será rechazadp.

3.4.5. Lectura de datos del ADS1299

Para adquirir señales el ADS1299 se configura en el modo de adquisición continua. Cuando hay un dato listo para ser leído el ADS1299 lo indica con un cambio de estado en el pin DRDY. Este pin desde el microcontrolador se lee por interrupción. Cuando se dispara la interrupción de DRDY, la rutina que atiende la interrupción lo que hace es iniciar la lectura de datos por SPI y libera la CPU ya que como se vio en la subsección 3.4.3 el puerto SPI trabaja en modo *callback*. Cuando los datos leídos por el SPI están listos, se ejecuta la función de *callback*. Dentro de esta función lo que se hace es enviar un evento al módulo de control y a través de una cola de mensajes los datos leídos del ADS1299.

3.5. Módulo interfaz de ususario

Este módulo está desarrollado en los archivos *DASN_UI.c* y *DASN_UI.h*. Es el encargado de manejar el pulsador y los dos leds que dispone la placa. Se decidió que sea un módulo independiente del módulo de control para que tenga más flexibilidad y si en futuros diseños se decide cambiar el hardware, esta arquitectura de software permite que sea escalable a más leds, pulsadores u otra manera de entregarle la información al usuario.

Cuando se quiere tomar alguna acción con los leds simplemente el módulo de control envía un evento al módulo interfaz de usuario indicando que acción se va a tomar.

3.5.1. Manejo de los leds

Para el manejo de los leds se hizo uso de los *clocks* que ofrece el sistema operativo. A estos *clocks* se les puede configurar un tiempo que al cumplirse hace que se ejecute una función de *callback*. En esta función se prende, apaga o cambia de estado el led según haya sido solicitado por el módulo de control. Esta configuración permite manejar los leds de forma totalmente desatendida sin casi ocupar tiempo de CPU.

3.5.2. Manejo del pulsador

El pulsado se maneja por interrupción y se aplicó una rutina antirrebote para evitar falsas pulsaciones. Esta rutina se implementó utilizando un *clock* del sistema operativo al que se le configuró un tiempo de 50 ms.

Cuando se presiona el pulsador y se ejecuta la interrupción por cambio de estado en el PIN del microcontrolador, lo que hace esta rutina es deshabilitar la interrupción del PIN para que el ruido de contacto del pulsador no genere más llamados a esta rutina. A su vez enciende el *clock* configurado en 50 ms y libera la CPU para que se puedan seguir haciendo otras tareas. Cuando se cumplen los 50 ms y se ejecuta la función de *callback* del *clock*, esta función chequea el estado del PIN del pulsador y determina si efectivamente se presionó o no. En el caso que se detecte que se presionó el pulsador se envía un evento al módulo de control, de lo contrario no se hace nada. En ambos casos se vuelve a habilitar la interrupción por cambio de estado del pulsador para dejar todo listo para un nuevo ciclo.

Capítulo 4

Ensayos y resultados

En este capítulo se describen los ensayos realizados y los resultados obtenidos. Adicionalmente se muestra el banco de prueba y herramientas utilizadas.

4.1. Banco de pruebas

4.1.1. Software utilizado

4.1.2. Hardware utilizado

4.2. Test Unitarios

4.3. Ensayos sobre la comunicación inalámbrica

4.3.1. Verificación de servicios BLE

4.3.2. Verificación de alcance

4.4. Medicion de impedancia

4.5. Adquisición de señal

Capítulo 5

Conclusiones

5.1. Trabajo realizado

5.2. Trabajo futuro

Bibliografía

[1] *IEC 62304:2006 Medical device software Software life cycle processes*.

[2] *IEC 60601-1:2005+AMD1:2012+AMD2:2020 CSV Consolidated version Medical electrical equipment. Part 1: General requirements for basic safety and essential performance*.

[3] *IEC 82304-1:2016 Health software Part 1: General requirements for product safety*.

[4] *ADS1299-x Low-Noise, 4-, 6-, 8-Channel, 24-Bit, Analog-to-Digital Converter for EEG and Biopotential Measurements*. URL: https://www.ti.com/lit/ds/symlink/ads1299.pdf.

[5] *BLE 5.0 Stack*. Texas instrument. URL: https://dev.ti.com/tirex/explore/content/simplelink_cc2640r2_sdk_5_10_00_02/docs/ble5stack/ble_user_guide/html/ble-stack-5.x/overview-cc2640.html.

[6] *2.4-GHz Inverted F Antenna*. Texas instrument. URL: https://www.ti.com/lit/an/swru120d/swru120d.pdf?ts=1661942737065&ref_url=https%253A%252F%252Fwww.google.com%252F.

[7] *LAUNCHXL-CC2640R2 SimpleLink™ Bluetooth® Low Energy CC2640R2 wireless MCU LaunchPad™ development kit*. URL: https://www.ti.com/tool/LAUNCHXL-CC2640R2?keyMatch=CC2640R2%20LAUNCHPAD.