Universidad de la República
Facultad de Ingeniería

Sistema Autónomo y Remoto de Adquisición

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Francisco de Izaguirre, Maite Gil, Marco Rolón

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista.

Tutor

Dr. Ing. Nicolás Pérez ..... Universidad de la República
Dr. Ing. Pablo Monzón ..... Universidad de la República

Tribunal

Dr. Ing. Gabriel Usera ..... Universidad de la República
Dr. Ing. Leonardo Steinfeld ..... Universidad de la República
Ing. Florencia Blasina ..... Universidad de la República

Montevideo
jueves 27 febrero, 2020

Sistema Autónomo y Remoto de Adquisición, Francisco de Izaguirre, Maite Gil, Marco Rolón.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 213 páginas.
Compilada el jueves 27 febrero, 2020.
http://iie.fing.edu.uy/

La investigación científica original comienza en el preciso momento en que la sabiduría nos deja en la ignorancia.

Clemente Estable

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

A los integrantes de PestiBee, Alejandro Draper, Nicolás Obrusnik y Pablo Zinemanas, por sentar las bases de este proyecto.

A los integrantes de GIAQA. Por permitirnos interactuar con otra aplicación.

A José Ferrando y su familia, quienes junto a Franco Simini nos permitieron realizar las pruebas del equipo en sus colmenas.

A Daniel Ivanchuk, Gonzalo Zarazola, Emmanuel Dall'Oglio y Alejandro Romio por toda la ayuda y buena disposición.

A los docentes y compañeros que dieron una mano. En particular a Mauricio Gonzalez y Sebastián Fernandez por el apoyo brindado.

A todos los que año a año hacen posible Ingeniería deMuestra, sobre todo en el interior de nuestro país, gracias a ellos pudimos compartir lo que nos gusta hacer.

A la educación pública del Uruguay en toda su extensión. Especialmente a la Universidad de la República. No solo por la formación académica de gran calidad, sino también por la formación humana.

Al Instituto de Ingeniería Eléctrica que nos abrió las puertas y proporcionó los materiales que hicieron posible este trabajo, además de los conocimientos.

A nuestros tutores Nicolás Pérez y Pablo Monzón. Por ser nuestros referentes durante éste último año.

A nuestras familias, por el apoyo de ayer, hoy y mañana. En particular a Mirella, Rodrigo y Mariana.

A quienes compartieron y formaron parte de esta experiencia. A los que nos dieron para adelante.

De corazón, muchas gracias a todos.

SARA-Bee

Esta página ha sido intencionalmente dejada en blanco.

A mi familia por su apoyo incondicional y atento. En especial a Mariana, Inés, Cecilia y Sofía por la ayuda prestada en la decoración para Ingeniería de Muestra. A Mariana, gran sostén sobre todo cuando la noche es larga.
Francisco

A mis padres y a Cami por ser apoyo constante. A Ro, pilar fundamental. A mis amigas por tener siempre una palabra de aliento.
Maite

A mis padres y mis hermanos.
A Mire y a Noli.
Marco

A nuestras familias, nuestros tutores y amigos, todos los que nos han ayudado a construir nuestra colmena.
SARA-Bee

Esta página ha sido intencionalmente dejada en blanco.

Resumen

SARA-Bee es un sistema que se compone de un equipo modular de telemetría y un servidor. Este sistema es flexible y adaptable a múltiples problemas. Como aplicación particular se desarrollaron módulos para el monitoreo de colmenas de abejas.

El objetivo general del proyecto fue diseñar y construir un equipo cuyas características fundamentales fueran arquitectura flexible, autonomía energética y comunicación remota. Conformado por módulos independientes, el cual proporciona versatilidad y permite la re-utilización de gran parte del hardware para distintas aplicaciones.

El equipo se compone de tres tipos de módulos con funcionalidades específicas: adquisición, alimentación y principal, este último encargado de la comunicación y de la agenda. Cada módulo está basado en una placa de desarrollo de Arduino y un *shield* diseñado específicamente para el propósito de dicho módulo. El firmware de los módulos fue desarrollado en lenguaje C.

Para la comunicación entre módulos, se utilizó el bus I2C y se desarrolló un protocolo de comunicación en base a las características del bus y del equipo. Dicho protocolo contempla lecturas escalares y vectoriales en múltiples canales; y tiene una secuencia de inicialización en la cual el maestro escanea el bus consultando parámetros que caracterizan a cada uno de los módulos esclavos.

En cuanto a la comunicación con el servidor, se utilizó el protocolo MQTT a través de GPRS. Es posible configurar el equipo, solicitar medidas y la configuración actual por medio de clientes MQTT.

Se presentan dos bases de datos para el almacenamiento, MongoDB e InfluxDB. En la primera se almacenan los datos crudos tal cual los recibe el servidor. En InfluxDB se almacenan los valores propiamente dichos. Consecuentemente, se utilizan dos Web Services. Uno es Chronograf, un Web Service de InfluxDB que permite la presentación de los datos. El otro Web Service es MongoKU en el cual se visualiza los mensajes en crudo.

Se realizaron pruebas a nivel de laboratorio en cada de uno de los módulos de forma independiente, de todos los módulos en conjunto, y del equipo con el

servidor. El equipo y cada uno de sus módulos fueron diseñados para ser utilizados a la intemperie. El sistema completo se probó de forma continua en el techo de facultad durante una semana con resultado satisfactorio.

Prefacio

Se puede decir que esta documentación es el fruto del trabajo de un año. Sin embargo, estamos convencidos que es mucho más. Es la conjunción del trabajo de todos los años en la educación formal, en particular la que la Universidad de la República nos acobijó. Es las amistades conquistadas, horas de tertulias de discusión con los tutores, y amigos en general.

Lo iniciamos pensando que iba a ser un cierre para nuestra carrera de grado. Pero, sin embargo, hoy todo apunta a un nuevo inicio, de maestrías, de trabajo profesional.

Esperamos que quien lea esta documentación le podamos transmitir el regocijo que nos generó haber realizado este proyecto. Que pueda ver el crecimiento del grupo desde el inicio. Y sobre todo que lo pueda disfrutar y sacar tanto provecho como nosotros.

Cerramos este ciclo ilusionados y deseamos que este trabajo se continúe y sea el cimiento de futuros proyectos.

SARA-Bee

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Prefacio **IX**

1. Introducción **1**
1.1. Definición del problema 1
1.2. Objetivos 3
1.3. Alcance del proyecto 3
1.4. Antecedentes 4
1.5. Organización del documento 6

2. Arquitectura del sistema **9**
2.1. Introducción 9
2.2. Equipo 10
2.3. Servidor 12
2.4. ¿Por qué Arduino? 13
2.5. ¿Por qué I2C? 14
2.6. ¿Por qué GPRS? 15
2.7. ¿Por qué MQTT? 16

3. Protocolo de comunicación del bus **19**
3.1. Motivación 19
3.2. Estructura de los datos de los módulos 20
3.2.1. Dirección 21
3.2.2. Tipo de módulo 21
3.2.3. Canales 22
3.3. Estructura de los paquetes 22
3.3.1. Descripción 23
3.4. Comunicación entre módulos 25
3.4.1. Intercambio de paquetes 25
3.4.2. Inicialización y adquisición 26
3.5. Librería implementada 28

4. Módulo de adquisición de señales escalares **31**
4.1. Elección de Hardware de Desarrollo 33
4.2. Elección de componentes 34
4.2.1. Medida de humedad relativa y temperatura 34
4.2.2. Medida del peso 35

4.3. Arquitectura de Software 37
4.4. Diseño de hardware 39
4.4.1. Medida de humedad relativa y temperatura 39
4.4.2. Medida del peso 40
4.4.3. Diseño de shield 43
4.5. Pruebas y Validación de módulo 44
4.5.1. Sensor de humedad relativa y temperatura 44
4.5.2. Balanza 44
4.5.3. Comunicación 45
4.6. Trabajos a futuro 46

5. Módulo de adquisición de señales vectoriales 47
5.1. Elección de Hardware de Desarrollo 49
5.1.1. Arduino MKR Vidor 4000 50
5.2. Elección de componentes 52
5.2.1. Elección de los micrófonos 52
5.2.2. Módulo Micro SD 53
5.2.3. Convertidor de nivel de voltaje bidireccional 54
5.3. Arquitectura de Software 55
5.3.1. Microcontrolador SAMD21 55
5.3.2. FPGA Cyclone10 58
5.4. Diseño de Hardware 67
5.4.1. Diseño de placas adquisidoras 67
5.4.2. Diseño de PCB 72
5.5. Pruebas y Validación de módulo 74
5.5.1. Pruebas Hardware 74
5.6. Trabajos a futuro 81

6. Módulo principal 83
6.1. Elección de Hardware de Desarrollo 83
6.2. Elección de componentes 84
6.2.1. Módulo de almacenamiento microSD 84
6.2.2. Módulo de comunicación GPRS 84
6.3. Arquitectura de Software 86
6.4. Diseño de Hardware 92
6.4.1. Diseño de Shields 92
6.5. Pruebas y Validación de módulo 95
6.5.1. Hardware 95
6.5.2. Firmware 96
6.6. Trabajos a futuro 97

7. Módulo de alimentación 99
7.1. Elección de componentes 99
7.1.1. Controlador 99
7.1.2. Batería 102
7.1.3. Panel solar 103

7.2. Pruebas y Validación de módulo . . . 104
7.2.1. Consumo de los módulos . . . 104
7.2.2. Estimación de duración de batería . . . 106
7.2.3. Eficiencia . . . 106
7.3. Trabajos a futuro . . . 107

8. Servidor **109**
8.1. Sistema Operativo . . . 109
8.2. Comunicación . . . 109
8.3. Almacenamiento - Bases de datos . . . 111
8.4. Inferfaz Web . . . 112
8.4.1. MongoKU . . . 112
8.4.2. Grafana . . . 113
8.4.3. Chronograf . . . 113
8.5. Broker y cliente MQTT . . . 113
8.6. Cliente SARA-Bee . . . 113
8.7. Pruebas y Validación . . . 115
8.8. Trabajos a futuro . . . 117

9. Sistema completo **119**
9.1. Pruebas en campo . . . 119
9.2. Análisis holístico . . . 120
9.3. Trabajo a futuro . . . 121

10. Gestión de proyecto **123**
10.1. Análisis de Costos . . . 123
10.1.1. Costos de materiales . . . 123
10.1.2. Mano de obra . . . 127
10.1.3. Presupuesto total . . . 128
10.2. Análisis de Riesgos y Tiempos . . . 128

11. Conclusiones **131**
11.1. Conclusiones generales . . . 131
11.2. Evaluación de los resultados . . . 132
11.2.1. Plataforma de desarrollo . . . 132
11.2.2. Bus de datos y alimentación . . . 132
11.2.3. Comunicación remota . . . 133
11.2.4. Monitoreo del sistema . . . 133
11.3. Desafíos y aprendizaje . . . 133
11.3.1. Gestión de proyecto . . . 133
11.3.2. Desarrollo de firmware . . . 133
11.3.3. Diseño de hardware . . . 134
11.3.4. Compras en el exterior . . . 134
11.3.5. Documentación . . . 134

A. Arquitectura de Software **137**
A.1. Introducción . . . 137
A.2. Round-Robin (RR) . . . 137
A.3. Round-Robin con Interrupciones (RR-I) . . . 138
A.4. Encolado de Funciones (EdeF) . . . 139
A.5. Sistemas Operativos en Tiempo Real (RTOS) . . . 139

Apéndices **136**

B. Bus I2C **141**
B.1. Introducción . . . 141
B.2. Características eléctricas . . . 141
B.2.1. Condiciones de START y STOP . . . 142
B.2.2. Condición de START repetida . . . 142
B.2.3. Acknowledge (ACK) y Not Acknowledge (NACK) . . . 143
B.3. ADDRESS y bit de R/W . . . 143
B.3.1. Registros . . . 143
B.4. Operación general de I2C . . . 143
B.5. Lectura y escritura . . . 144

C. Acondicionamiento de señales **147**

D. FFT **149**

E. FPGA **153**

F. MQTT **155**
F.1. Introducción . . . 155
F.1.1. Origenes . . . 155
F.1.2. Beneficios . . . 156
F.1.3. Estructura de comunicación . . . 156
F.2. Clientes y Broker . . . 156
F.2.1. "Client" . . . 156
F.2.2. "Broker" . . . 157
F.3. "Topics" . . . 157
F.3.1. Wildcards . . . 158
F.4. Comunicación MQTT . . . 159
F.4.1. Mensaje del cliente para establecer conexión . . . 159
F.4.2. Rerpuesta del broker para establecer la conexión . . . 161
F.5. Publicaciones y Suscripciones . . . 162
F.5.1. Publicaciones . . . 162
F.5.2. Suscripción . . . 163
F.5.3. Des-Suscripción . . . 165
F.6. Calidad de servicio "QoS" . . . 166
F.6.1. QoS 0 - A lo sumo una vez . . . 166
F.6.2. QoS 1 - Por lo menos una vez . . . 166

F.6.3. QoS 2 - Exctamente una vez 167
F.7. Persistent Session 168
F.7.1. ¿Que se almacena en una sesión persistente? 168
F.7.2. ¿Como comenzar/terminar una sesion persistente? 168
F.7.3. ¿Puede el cliente saber si hay una sesió almacenada? 168
F.8. RetainedFLag 169

G. DHT22 **171**
G.1. Descripción 171
G.2. Características 171
G.3. Pines 172
G.4. Interfaz serial 172

H. HX711 **175**
H.1. Descripción 175
H.2. Características 176
H.3. Pines 176
H.4. Interfaz Serial 177

I. Contenido de la entrega **179**

Referencias **181**

Glosario **185**

Índice de tablas **189**

Índice de figuras **191**

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 1

Introducción

1.1. Definición del problema

SARA-Bee surge de la necesidad de tener un sistema de telemetría capaz de adquirir señales atmosféricas y agronómicas de forma remota, que cuente con autonomía energética para su compatibilidad en áreas rurales o sitios con difícil acceso eléctrico.

Con el avance de la tecnología, la gestión y monitoreo de equipos en diversas aplicaciones es cada vez más cercano, accesible y necesario. Tanto para mejorar la eficiencia de la producción, como para automatizar tareas que faciliten la labor en áreas de investigación y académicas. Se busca un sistema que facilite al consumidor final relevar los datos e información de forma amigable, remota, y en una plataforma de fácil acceso. Es de gran interés, en innumerables aplicaciones, obtener datos en tiempo real, para constatar el estado de funcionamiento del equipo, para así aplicar medidas preventivas y/o correctivas de forma inmediata, pudiendo evitar eventos más graves en el futuro. La sustentabilidad energética es otro pilar de gran relevancia, ya que ataca la seguridad energética, la equidad social, y la mitigación del impacto ambiental; se logra con el uso de fuentes renovables como la energía solar, eólica, geotérmica, hidroeléctrica, entre otras.

La aplicación particular que motiva a este proyecto es el monitoreo de colmenas de abejas, como indicador de residuos de pesticidas aplicados en su zona de influencia. El uso de pesticidas es común en la agroindustria con el fin de erradicar plagas, su uso extensivo en monocultivos busca potenciar el rendimiento de la producción, a costa de, entre otras, la salud de las colmenas de abejas; éstas se ven afectadas, ya sea a través de alimentos contaminados que recolectan e ingresan a la colmena, o bien por los residuos tóxicos que les queda en su cuerpo. Este impacto en la reducción de las poblaciones de abejas tendrá un efecto negativo sobre la producción de alimentos y la bio diversidad vegetal, ya que son responsables en cierta medida de polinizar un 75 % de los cultivos alimentarios a nivel mundial. [4]

Según los investigadores Nazly Yolieth Martin-Culma, Nelson Enrique Arenas-Suarez, "La alteración en las poblaciones de abejas es un claro daño colateral derivado del uso de pesticidas y, en consecuencia, podrían ser indicadores biológicos de su efecto nocivo a nivel ecológico y ambiental" [32]. Esto motivó a investigadores a formar parte de un proyecto interdisciplinario que nació en 2015, dirigido por el Dr. Ing. Nicolás Pérez y la Dra. en Química Silvina Niell, con integrantes del Polo Agroalimentario y Agroindustrial de Paysandú de la UdelaR en conjunto con INIA Estanzuela. Se planteó como objetivo utilizar la colmena de abejas para la detección de residuos contaminantes. La línea de investigación a seguir es la "caracterización acústica". El batido de las alas de estos seres vivos, generan vibraciones en el aire, que se resumen en un zumbido característico. La especie en la que se enfocaron los investigadores es *Apis Mellifera*, comúnmente llamada "abeja doméstica", y es la especie con mayor distribución a nivel mundial, con colonias en las que habitan entre 15.000 y 50.000 abejas. Se parte de la hipótesis, que con este número de abejas zumbando en simultáneo se puede realizar un reconocimiento de patrones y obtener información de la actividad de la colmena y su estado de salud, en particular reconocer si se encuentran en presencia de pesticidas, ya que su comportamiento se vería afectado por estos residuos contaminantes.

En 2015 el proyecto de fin de carrera *PestiBee* [1] propuso y plasmó una solución para el problema de la adquisición de las variables de las colmenas.

Mencionada esta aplicación, se debe destacar que este proyecto, *SARA-Bee*, resuelve un problema común para varias ramas de la Ingeniería, así fue que existieron otros propósitos que motivaron al proyecto, y en las que además *SARA-Bee* se vio directamente involucrado.

Un integrante de SARA-Bee, trabajó en conjunto con los tutores de este proyecto, los doctores Ing. Pablo Monzón e Ing. Nicolás Pérez, y el grupo GIAQA de Facultad de Química, llevando a cabo un proyecto que cumple con las mismas necesidades que *SARA-Bee* pero con enfoque en determinar la cantidad de fósforo y nitrógeno presente en el agua de cursos de los ríos. Este proyecto necesitaba una fuente de alimentación autónoma, adquirir muestras de agua, medir el nivel de PH y la cantidad de nitrógeno y fósforo de la misma, y transmitir los datos recolectados/procesados a un servidor, recibir los datos, almacenarlos y desplegarlos. Siendo las medidas la única diferencia entre ambos proyectos.

Considerando la existencia de otros casos que sólo difieran en las medidas de interés se busca un sistema modularizado que ataque este tipo de problema común en diversas aplicaciones, conformado por un sistema de adquisición y un servidor. Con módulos independientes que se encarguen de la adquisición de señales, de la gestión de alimentación del sistema a través de una fuente renovable, y una batería que se adapte a la necesidad del proyecto. Que cuente además con un módulo principal de comunicación que reciba las medidas obtenidas, las eleve a un servidor donde puedan ser desplegadas de forma amigable al usuario, y adaptable a sus

necesidades.

Se toma como objeto de estudio la colmena de abejas, su comportamiento y condiciones en las que se encuentra. Esta aplicación particular debe adquirir señales de audio, temperatura, humedad y peso. Con esta información se puede comparar cómo **debería** ser el comportamiento de la colmena (temperatura, humedad y peso), y cómo se **está** comportando en ese momento (audio).

1.2. Objetivos

El objetivo general de este proyecto es diseñar un sistema de módulos que permitan adquirir y procesar señales escalares (i.e: humedad, temperatura, peso), y vectoriales (i.e: audio), así como enviar información a un servidor y recibir órdenes remotas. El sistema debe ser energéticamente autónomo.

Cabe destacar que todos los módulos deben ser independientes entre sí. La finalidad del equipo a construir es que permita ser reutilizado y de propósito general.

El proyecto tiene como concepto principal ser modular, es decir que el equipo pueda descomponerse en módulos independientes y reemplazables, siendo únicamente indispensable el módulo principal y el de alimentación.

A su vez, el sistema debe contemplar una interfaz simple y bien documentada. Los módulos de adquisición y procesamiento deben poder ser reemplazables por cualquier módulo compatible con el equipo. Debe poder llevar adelante la adquisición y transmisión de datos para cualquier tipo de señal atmosférica/agronómica.

1.3. Alcance del proyecto

El alcance del proyecto incluyó los siguientes puntos:

- Comunicación local entre módulos (I2C) y comunicación remota entre módulo principal y servidor (GPRS).
- Almacenamiento total de los datos localmente (microSD) y parcial en servidor.
- Reacondicionamiento del módulo de alimentación desarrollado por PestiBee [1] o implementación de un nuevo módulo de alimentación.
- Adquisición, acondicionamiento y procesamiento (sólo audio) de las señales previo a la transmisión.
- Diseño e implementación de firmware y hardware; elección de gabinetes adecuados para cada módulo.

- Integración de los módulos del equipo y servidor.
- Pruebas a nivel de laboratorio.
- Documentación técnica e implementación de *test-points* [1] para cada módulo.
- Configuración de base de datos e interfaz web.
- Documentación global del proyecto.

El alcance no incluyó:

- Diseño de nuevo módulo de alimentación.
- El estudio de las señales de audio adquiridas.

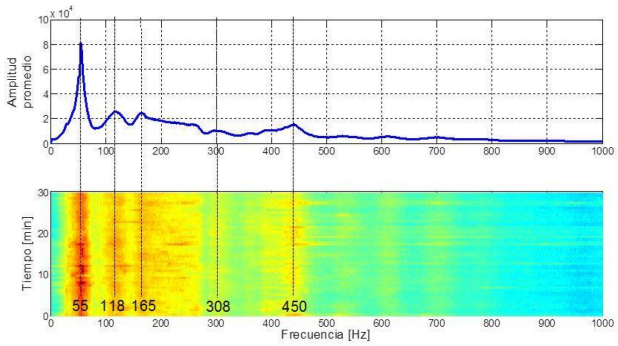
1.4. Antecedentes

A *SARA-Bee* le antecede el Proyecto de Fin de Carrera *PestiBee* [1], llevado a cabo en el 2015 por Alejandro Draper, Nicolás Obrusnik y Pablo Zinemanas, tutelados por Pablo Monzón y Nicolás Pérez. El proyecto tuvo como objetivo realizar el diseño de un sistema que permitiera la adquisición de variable ambientales y el audio de colmenas de abejas. El mismo debía tener autonomía eléctrica y ser capaz de trasmitir los datos a un servidor. *Pesti-Bee* relevó muestras de un canal de temperatura y 8 canales de audio, los cuales fueron pre-procesados en una placa Stellaris [21].
El proyecto *PestiBee* se basó en la investigación de dos trabajos previos que vale la pena mencionar.

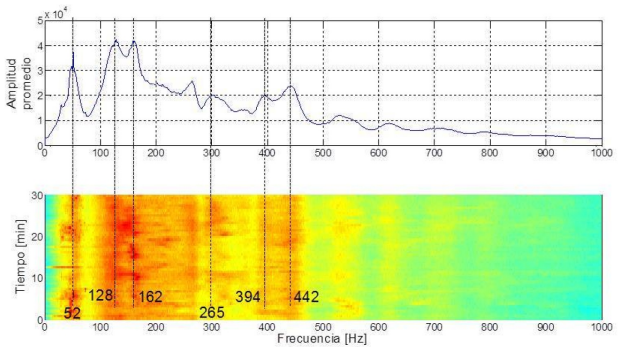
- "Plataforma de monitorización de colmenas basada en el análisis del sonido", un trabajo conjunto de la Universidad Europea de Madrid y el Ministerio de Medio Ambiente y Medio Rural Marino de España. [29]. Se construye un sistema para el monitoreo del sonido en las colmenas de abejas, creando una res de sensores inalámbricos; basado en SO Windows, micrófonos omnidireccionales, ADC de 8 bits, dos paneles solares, y la comunicación entre nodos a través del protocolo *Zigbee*.
- "Caracterización acústica de la colmena para detección temprana de contaminación por agroquímicos", CUP, Universidad de la República, Uruguay. Proyecto interdisciplinario ya mencionado, donde PestiBee propone desarrollar un prototipo [31]. Este proyecto realizó estudios para encontrar los picos de frecuencia relevante, y estudiaron la variación a lo largo del día. Tomaron muestras de 30 segundos, separadas entre sí por intervalos de 30 segundos, y realizaron el cálculo de la FFT a las 30 tramas que se obtienen en media hora, y luego promedian los módulos de las transformadas. Con estos datos

[1] Protocolo de pruebas de cada módulo para ser realizadas en cada uno de ellos de forma independiente.

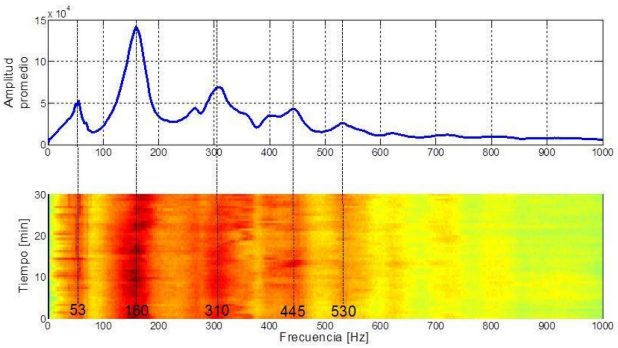
relevados, construyen el espectrograma, para observar cambios en el patrón de frecuencia en un día. En la figura 1.1 se pueden observar tres espectros y respectivos espectrogramas obtenidos para tres momentos distintos del día.



(a) Espectro y espectrograma del audio de la colmena de 9:15 am a 9:45 am



(b) Espectro y espectrograma del audio de la colmena de 12:30pm a 1:00pm



(c) Espectro y espectrograma del audio de la colmena de 4:30pm a 5:00pm

Figura 1.1: Espectros y espectrogramas del audio para tres momentos del día.

SARA-Bee se plantea realizar un sistema de monitoreo de propósito general, con un protocolo que permita la adhesión de otros módulos que adquieran nuevas

señales de interés; donde la modularidad sea protagonista, tanto en software como en hardware, y utilice una plataforma que facilite la re-utilización del sistema; tomando como aplicación particular, el objeto de estudio del *PestiBee*: el monitoreo de colmenas de abejas.

1.5. Organización del documento

El documento se encuentra organizado en capítulos y contiene anexado apéndices, a continuación se describe el contenido de los mismos.

Capítulos

En el capítulo 2 se describe la arquitectura del sistema: cómo se compone y qué función cumple cada una de sus partes. En el capítulo 3 se detalla el protocolo SSCMP desarrollado para la comunicación entre módulos sobre I2C. En los capítulos 4, 5, 6 y 7 se explica el diseño e implementación de los módulos de adquisición escalar, vectorial, principal y alimentación. En el capítulo 8 se describe la implementación del servidor. En el capítulo 9 se hace una reflexión del funcionamiento del sistema completo. En el capítulo 10 se hace un análisis sobre la gestión de proyecto realizada. Finalmente en el capítulo 11 se realizan las conclusiones finales del proyecto.

Apéndices

En el apéndice A se presenta de forma breve una introducción a cuatro arquitecturas típicas para sistemas embebidos: Round-Robin, Round-Robin con Interrupciones, encolado de funciones y Sistema Operativo en Tiempo Real (RTOS). En el apéndice B (vinculado al capítulo 3) se describen los aspectos más importantes del protocolo I2C. En el apéndice C (vinculado al capítulo 5) se muestra que el filtro utilizado en la etapa de filtrado del MASV es efectivamente un filtro Butterworth de segundo orden. En el apéndice D (vinculado al capítulo 5) se desarrolla el algoritmo de FFT, y la implementación radix-2 no recursiva implementada en el proyecto. El apéndice E (vinculado al capítulo 5) tiene como objetivo familiarizar al lector con la tecnología FPGA, su definición y etapas de diseño. El apéndice F (vinculado al capítulo 6) presenta aspectos básicos del protocolo tales como casos de usos, propiedades, composición de los mensajes, banderas importantes. En el apéndice G (vinculado al capítulo 4) se da una descripción del funcionamiento del sensor DHT22 de humedad relativa y temperatura. En el apéndice H (vinculado al capítulo 4) se brinda una descripción del módulo HX711 utilizado para la balanza. En I se explica el contenido de la entrega.

Marcas organizativas

Al final del documento se encuentran ubicadas las secciones complementarías: Referencias, donde se citan las fuentes bibliagráficas; Glosario, donde se explica la

nomenclatura utilizada y en qué páginas de la documentación se encuentran; y los Índices para las Figuras y las Tablas del documento.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 2

Arquitectura del sistema

En este capítulo se describe la arquitectura del sistema, cómo se compone y qué función cumple cada una de sus partes.

2.1. Introducción

SARA-Bee se compone sustancialmente de dos partes: el equipo y el servidor. El diagrama de la figura 2.1 se lee de izquierda a derecha. En la primera etapa se observa el ecosistema a ser monitoreado, junto a los sensores que lo caracterizan; y la energía solar ingresando al equipo por medio de los paneles. Luego, se puede observar el equipo en donde se lleva a cabo la gestión energética, la adquisición automatizada de muestras de forma periódica según el módulo, y la comunicación con el servidor por donde se envían las lecturas y se reciben parámetros. La información recibida por el mismo se puede acceder desde distintos entornos dependiendo de la necesidad del usuario. Los datos se pueden recibir crudos en un celular, y desde el servidor se pueden ver crudos o desplegados de forma gráfica en distintas configuraciones.

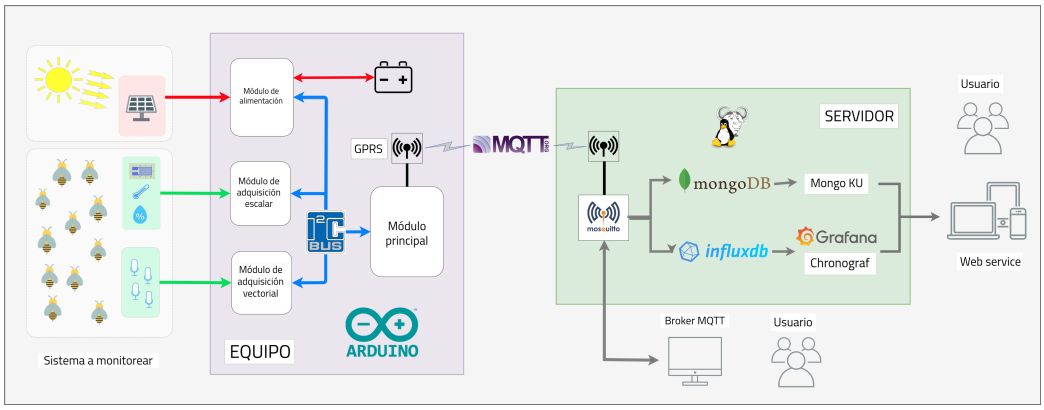


Figura 2.1: Arquitectura del sistema.

2.2. Equipo

El equipo se encarga de adquirir señales, pueden ser escalares o vectoriales. Una vez realizada la adquisición, procesa los datos obtenidos, los guarda localmente, y los envía (total o parcialmente) al servidor. Los parámetros de funcionamiento del equipo deben de poder modificarse local y remotamente.

El equipo está compuesto por varios módulos que se categorizan en tres tipos:

- Módulo principal
- Módulo de alimentación
- Módulo de adquisición
    - Multicanal
    - No Multicanal (secuencial)

A pesar de que, *a priori*, la definición de tipos de módulos predefinidos parece ser una limitante del equipo, de hecho le da al sistema la ventaja de tener generalidad y nos permite crear una interfaz simple e independiente de la implementación actual de cada módulo.

Si bien los módulos de adquisición y alimentación son similares, se realizó la distinción entre ellos para futuras implementaciones donde sea más riguroso el control del consumo energético.

Principal

Características generales:

- Maneja todos los aspectos de comunicación del equipo: entre módulos (I2C) y con el servidor (GPRS).
- Maneja la agenda del equipo.
- No realiza adquisición ni procesamiento.
- Es único y fundamental dentro del equipo.

Alimentación

Características generales:

- Gestiona y proporciona energía a todo el equipo.
- Reporta parámetros energéticos (i.e. voltaje de batería).
- Es único dentro del equipo.

En el proyecto este módulo está compuesto por una batería, un panel solar y un controlador.

Adquisición de señales

Características generales:

- Cuando es solicitado adquiere la señal escalar o vectorial en el canal indicado.
- Puede almacenar localmente las señales adquiridas, en caso de ser necesario.
- Puede procesar las señales adquiridas, esto es deseado en señales vectoriales de forma de comprimir información.
- Devuelve la lectura realizada, o una representación de la misma.

Los módulos realizados para el proyecto son dos. El **MASE** adquiere humedad, temperatura y peso. El **MASV** es el encargado de la adquisición y el preprocesamiento del audio.

Bus unificado

Para la conexión de los módulos se utilizará un bus de datos y alimentación unificado. Se definen cuatro pines para la alimentación: 0V, 3.3V, 5V, 12V, y dos para la comunicación: SDA y SCL. Se deja un pin libre para una eventual señal de control a definir.

El *pinout* del bus, que se muestra en la Tabla 2.1, no cumple ningún estándar pero sigue la convención de colores de las fuentes de alimentación ATX.

Pin	**Función**	**Color**
1	*12V*	*Amarillo*
2	*5V*	*Rojo*
3	*0V*	*Negro*
4	*SDA*	*Blanco*
5	*SCL*	*Violeta*
6	*3.3V*	*Naranja*
7	*No definido*	-

Tabla 2.1: Bus de datos y alimentación unificado.

Todos los módulos del bus deben ser compatibles a nivel de I2C en los siguientes puntos:

- Los esclavos deben tener direcciones distintas y el maestro no tener ninguna.
- El módulo principal debe tener un *pull-up* para las líneas SDA y SCL, en particular se utilizó un voltaje de $5V$ y un *pull-up* de $2{,}2k\Omega$.
- Velocidad de comunicación: $100kHz$.

- Deben comunicarse a través del protocolo SSCMP.

El conector utilizado es el GX16-7p que se muestra en la Figura 2.2, fue elegido por los tutores previo al inicio del proyecto por lo que estaba dentro de las especificaciones del mismo.

Figura 2.2: Conector GX16-7p utilizado para el bus unificado.

El chasis de cada módulo tiene dos conectores macho, cortocircuitados internamente, de forma de poder agregar módulos simplemente conectando cada uno con el siguiente a través de un conector hembra-hembra. Esto genera que el primer y el último módulo queden con un conector libre, que se conectan entre sí, brindándole redundancia y disminuyendo el tiempo de propagación entre los extremos del bus.

2.3. Servidor

El servidor se encarga de almacenar los datos recibidos y luego desplegarlos en una web. A través de éste se podrán modificar los parámentros del equipo.

En pos de cumplir con esta funcionalidad, en el servidor se instaló el broker y el cliente **Mosquitto** de MQTT para recibir los mensajes del módulo principal de *SARA-Bee* mediante suscripción a los topics (similar a canales). Se utilizó **MongoDB** e **InfluxDB** para almacenar los datos. Se conectó la base de datos **InfluxDB** al Web Service **Chronograf**, para poder presentar vía web la evolución temporal de los datos adquiridos y permitir un análisis posterior de los mismos.

La elección de Mosquitto como broker y cliente, y Chronograf como Web Services, viene principalmente dada por ser productos de software libre y gratuitos con un gran potencial, grandes comunidades y un extendido uso en proyectos similares.

La elección de InfluxDB como sistema de base de datos se sustenta por ser liviano, sencillo, rápido y seguro, actualmente es ampliamente utilizado para el monitoreo de sistemas IoT. La empresa InfluxData [16] desarrolladora de InfluxDB tiene una *suite* completa de backend y frontend. Chronograf es un Web Service perteneciente a esta *suite* por lo cual es específico de InfluxDB. Chronograf permite además de la visualización la gestión de la base de datos via web, y el uso de otros productos de la empresa, tales como Telegraf o Kapacitor.

La base de datos MongoDB almacena archivos JSON. En este proyecto permite el almacenamiento directo de los mensajes recibidos por el servidor.

2.4. ¿Por qué Arduino?

Arduino es una plataforma electrónica *open-source* de hardware y software. Nace en el año 2005 como un proyecto enfocado a estudiantes en el Instituto IVREA (IDII), en Ivrea (Italia). Arduino tiene sus orígenes en el proyecto Wiring.

La plataforma consta de dos elementos:

- IDE: entorno de desarrollo basado en el entorno *Processing* y en el lenguaje de programación Wiring.
- *Bootloader*: ejecutado por el microcontrolador cuando arranca y, entre otras cosas, permite prescindir de un programador.

Existen múltiples razones por las cuales Arduino es la plataforma ideal para desarrollar los módulos de nuestro proyecto. A continuación se enumeran las que se consideraron como más importantes.

Software

- Es de código abierto.
- Se encuentra a disposición un gran número de librerías.
- Existe una comunidad de desarrolladores activa que crece día a día.
- IDE compatible con los sistemas operativos más comunes.

Hardware

- Es de hardware abierto, lo que hace que sean muy económicas.
- Año a año crece la oferta de plataformas de hardware desarrollado por Arduino.
- Es de fácil acceso, se pueden comprar en cualquier tienda de electrónica.
- Existe una gran cantidad de módulos/*shields* compatibles con librerías desarrolladas.

Programador

Arduino no requiere programador. Este punto es muy importante en varios aspectos. En primer lugar, la hace muy flexible desde lo económico, ya que estos suelen ser costosos, y se hubiera necesitado más de uno durante el desarrollo del proyecto. Desde el punto de vista operativo es una gran ventaja, ya que se puede programar la placa directamente desde la PC a través de la IDE.

Visión a futuro

Una de las bases del proyecto es diseñar un equipo reutilizable y versátil, que se pueda adaptar fácilmente a las necesidades de nuevos clientes. Pensando a largo plazo, Arduino es la opción que mejor se adapta a este concepto, ya que suma adeptos día a día, y sigue incrementando su catálogo de productos.

2.5. ¿Por qué I2C?

I2C es un **requerimiento** de proyecto en pos de tener compatibilidad con otros proyectos del Departamento de Sistemas y Control. En adición a esto, existen otros motivos por los cuales I2C resulta ser la opción compatible con las características deseables del equipo.

Veamos algunas características de los buses comúnmente utilizados:

- SPI: Tiene como punto a favor que la mayoría de los microcontroladores tienen integrado módulos de harware compatibles con este protocolo, por lo tanto se puede controlar el bus directamente desde los pines del microcontrolador. Como punto en contra, necesita una línea para cada dispositivo, por lo cual no es escalable.

- Bus CAN o RS485: Ambos buses permiten la conexión de múltiples equipos con solo dos líneas al igual que I2C y tienen largo alcance a diferencia de este último. Como desventaja manejan niveles de voltaje distintos a CMOS, por lo que necesitan interfaces microcontrolador-bus implicando que cada módulo necesite hardware adicional.

- I2C: La mayoría de los microcontroladores tienen integrado I2C, lo que hace que se pueda controlar el bus directamente desde los pines del microcontrolador y no necesite hardware adicional más que una resistencia de *pull-up* para cada una de las líneas del bus. Necesita solo dos hilos para las señales y puede controlar hasta 112 esclavos. Esto resulta muy conveniente al desarrollar nuevos módulos y hace que el sistema sea más versátil, pudiendo cambiar fácilmente la topología del mismo, lo cual es una de las características requeridas del equipo. Cabe recordar también que se utilizará un bus de siete líneas para interconectar los módulos, de las cuales 4 ya están asignadas

para 3.3V, 5V, 12V y GND, por lo que restarían solamente 3. Una desventaja de I2C es su corto alcance (unos pocos metros), pero esto no será un inconveniente dado que los módulos serán ubicados en el mismo gabinete.

Cabe destacar que los módulos se van a implementar sobre Arduino, donde está desarrollada la librería *Wire*. Ésta, al ser de código abierto, y dado que Arduino dispone de una comunidad numerosa, se encuentra estable y bien documentada.

Visión a futuro

Por la razones mencionadas anteriormente, I2C es el protocolo que más se adapta a la filosofía del equipo. Si bien presenta limitaciones como el alcance o la cantidad de direcciones disponibles, se considera que es una opción conveniente dadas las posibles aplicaciones que se esperan del equipo. Si se presentaran otro tipo de aplicaciones que involucren por ejemplo, acceder a un sitio donde el equipo no quepa físicamente, o se deba abarcar distancias mayores a las que I2C permite, se sugiere agregar algún módulo inalámbrico que funcione en paralelo, y se comunique con un módulo base conectado al bus I2C. En esta línea comenzó a trabajar el proyecto de fin de carrera ReSeDEM de Gonzalo Zarazola, Matias Echeverría y Juan Martin Chiale quienes están desarrollando un módulo que actúa como maestro para una red LoRa *point to point* y como módulo de adquisición en el sistema *SARA-Bee*.

2.6. ¿Por qué GPRS?

GPRS es actualmente la red de telefonía más extendida [2] en el territorio nacional. Para este proyecto es primordial que se encuentre presente en la zona rural, y esta red lo está. Permite transmitir a velocidades máximas de 384kbps y soporta comunicaciones TCP/IP. Existen módulos económicos, de fácil acceso en el mercado que cuentan con una gran comunidad de soporte, como lo son por ejemplo, las series de productos SIM800 y SIM900 de SIMCom. Estos módulos tienen aplicaciones para TCP/IP, que es el protocolo sobre el que se basa el protocolo MQTT. A su vez, esta red consume menos energía que la comunicación en redes más veloces como lo es LTE.

Visión a futuro

La tecnología avanza, y en los próximos años GPRS será dado de baja en el territorio nacional. Esto podría resultar problemático para proyectos de telemetría, sin embargo, como el desarrollo de sistemas IoT es cada vez mayor, Antel se encuentra desplegando la red NB-IoT [7]. Actualmente esta red se encuentra únicamente en Montevideo y área Metropolitana [2]. Esta red promete tener un consumo más bajo, ancho de banda dinámico, y otros beneficios particulares para IoT [7].

Será de importancia hacer un seguimiento a los avances de estas tecnologías, en pos de evaluar una posible migración. Ésta podrá realizarse de forma independiente del resto del sistema gracias a la modularidad del mismo.

2.7. ¿Por qué MQTT?

MQTT es un protocolo liviano, eficiente en el uso del ancho de banda, y existen múltiples implementaciones en diversas plataformas y lenguajes. Estas cualidades lo ha hecho muy popular para aplicaciones de IoT. Más información se puede encontrar en el anexo F

En este proyecto MQTT simplifica la comunicación, ya que el módulo que envía la información al servidor solo debe establecer una conexión con el broker (cuando necesita publicar en los topics que se desea) y permite que *SARA-Bee* esté suscripta a topics por donde recibir actualización de las configuraciones de los módulos. En este protocolo los clientes sólo se comunican con el broker y no saben, ni es necesario que conozcan, si hay otros clientes que reciben la información que se publica. Esto asegura la escalabilidad del proyecto, haciendo posible que distintos clientes *SARA-Bee* puedan interactuar entre sí.

Los niveles de calidad de servicio que brinda MQTT (QoS de 0 a 2) ofrecen la libertad necesaria para manejar el envío de mensajes con la rigurosidad que sea necesaria, desde la calidad más baja, donde los mensajes que no se reciban en el momento pueden descartarse, hasta la máxima calidad, donde los mensajes deban llegar sí o sí y una única vez.

MQTT permite el uso de sesiones persistentes, en las cuales si un dispositivo que debe recibir un mensaje está desconectado, el mensaje se le entregará cuando se conecte la próxima vez. Esto, de la mano de la retención de mensajes y los testamentos, lo convierte en una herramienta muy potente. La retención de mensaje consiste en mensajes que se envían a los clientes cuando se suscriben en un tópico, y los testamentos son mensajes que se envían si un cliente falla en la conexión.

Por último, la estructura y la gestión de los topics es muy intuitiva, sencilla y versátil.

Visión a futuro

Haber utilizado MQTT pone al proyecto de cara al futuro, pues es uno de los protocolos por excelencia para IoT, ya que aparece en casi la totalidad de servidores en proyectos similares. Sin embargo, se deberá seguir de cerca la evolución de otras alternativas como **Cloud Pub/Sub** de Google, entre otros.

En el desarrollo del proyecto quedó en el debe una mejor utilización de las opciones de seguridad que permite MQTT, aspecto el cual se entiende que hay

que mejorar.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 3

Protocolo de comunicación del bus

El protocolo del bus I2C, también llamado SSCMP por sus siglas en inglés *SARA Simple Communication and Management Protocol*, es uno de los aspectos fundamentales del sistema. Le proporciona independencia con la aplicación implementada en el proyecto, y un alto nivel de flexibilidad y versatilidad al momento de agregar o quitar nuevos módulos.

3.1. Motivación

Un protocolo de comunicación es una serie de reglas que se definen con el objetivo de que dos (o más) dispositivos puedan comunicarse y entenderse.

La necesidad de crear un protocolo para el intercambio de datos entre los módulos nace durante el desarrollo del sistema de forma constructiva e iterativa. Toma como base las características y limitaciones del equipo y del bus I2C.

En la comunicación a través de I2C, se pueden diferenciar dos etapas distintas. La primera de ellas es maestro-transmisor/esclavo-receptor en donde no hay dificultades ya que el maestro controla la linea del reloj y por lo tanto controla el fin de cada paquete. En la segunda etapa, maestro-receptor/esclavo-transmisor, ambos deben saber de antemano la cantidad de datos que el esclavo va a transmitir. Esto le deja dos alternativas al desarrollador:

1. Usar paquetes de largo fijo. Tiene como ventaja ser la solución más simple. Como desventaja, limita la comunicación y por lo tanto las funcionalidades del equipo a la aplicación en particular.

2. Preguntar previamente el largo del paquete. Agrega un grado de complejidad al problema, sin embargo brinda al equipo gran versatilidad y escalabilidad.

Se optó por la segunda alternativa.

Podría ocurrir que el maestro envíe por error al esclavo dos preguntas **distintas** consecutivas y le solicite la respuesta inmediatamente. Si lo único que se

transmite son los datos, no hay forma de distinguir a qué pregunta corresponde la respuesta dada.

Cabe mencionar que no se encontraron protocolos suficientemente robustos para el intercambio de datos sobre I2C. Muchos de ellos, están definidos para algún sensor en particular. Son en la mayoría de los casos muy precarios, carecen de control de errores, son incompatibles entre ellos y no son escalables.

Por estos motivos, se decidió implementar un protocolo que consiste en el intercambio de paquetes con un formato dado y una secuencia de inicialización. Para el intercambio de paquetes, se define la forma para las preguntas y respuestas, y qué datos deben contener. La inicialización es llevada a cabo por el maestro, el cual descubre a los esclavos presentes en el bus. De esa forma, el maestro se independiza de saber cuántos y cómo son los esclavos, lo que hace posible cambiar la topología del bus sin modificar el firmware ni la configuración del maestro.

Las características que el protocolo desarrollado le proporciona al sistema son las siguientes:

- Comandos compactos y genéricos, predefinidos de forma de cubrir necesidades actuales y futuras. Esto posibilita agregar nuevas funcionalidades manteniendo las actuales.
- Escalabilidad en el envío de paquetes, permitiendo distintos tamaños, en particular en las lecturas de los canales.
- Cada módulo almacena los parámetros que lo caracterizan, de forma que es totalmente independiente del resto de los módulos, en particular del maestro.
- Independencia del tipo de dato adquirido, y de la arquitectura del microcontrolador. La única restricción es que sea representado en punto flotante de simple presición [1].
- Verificación de errores en los paquetes.

3.2. Estructura de los datos de los módulos

Cada módulo debe almacenar su configuración de forma de mantener total independencia del módulo principal, permitiendo que para agregar/quitar un módulo del sistema no requiera más que:

1. desconectar el módulo de alimentación del resto de los módulos,
2. conectar/desconectar el módulo a agregar/quitar,
3. conectar el módulo de alimentación nuevamente, verificando que el módulo principal se haya reiniciado.

[1] Compatible con el estándar IEEE 754 para aritmética en punto flotante.

Durante la secuencia de inicialización, cada módulo esclavo debe brindar al módulo principal los siguientes datos:

- Dirección.
- Tipo de módulo.
- Cantidad de canales.
- En cada canal:
  - Largo de la lectura.
  - Agenda.
  - Cantidad de parámetros y valor de dichos parámetros.

Así, el maestro tiene caracterizados a los módulos esclavos.

3.2.1. Dirección

La dirección corresponde al *address* I2C del bus que tendrá asignada el módulo. Debe ser fija y **única** en el bus. Este número, junto al número de canal, permitirá identificar la lectura en el servidor.

3.2.2. Tipo de módulo

Los módulos esclavos pueden tener las siguientes etiquetas:

- 0: Alimentación.
- Adquisición:
  - 1: Multicanal.
  - 2: No multicanal.

La diferenciación dentro de los esclavos entre alimentación y adquisición se realiza con el fin de asignar niveles de prioridad a dichos módulos, dándole más relevancia a los de alimentación; ya qué, dada la naturaleza del proyecto, el nivel de batería es más trascendente que la humedad relativa, por mencionar un ejemplo. Los módulos de **multicanal**, pueden tomar lecturas en simultáneo a todos los canales. En los **no multicanal**, se debe tomar la lectura de un canal a la vez. Esto puede surgir por limitaciones del microcontrolador, o por la misma naturaleza de las lecturas a realizar; por ejemplo, si se desea tomar una medida secuencial.

3.2.3. Canales

Cada canal equivale a una medida. Estas pueden ser lecturas escalares o vectoriales. Un mismo módulo puede tener varios canales con el mismo tipo de medida, por ejemplo cuatro canales de audio; o distintos canales, cada uno destinado a un tipo de medida, por ejemplo humedad, temperatura y peso. Los datos necesarios para caracterizar los canales son:

- Cantidad de canales del módulo.
- Largo de la lectura de cada canal. Esto es, la dimensión del vector adquirido.
- Agenda de cada canal en *ticks* [2]:
  - Período de adquisición.
  - Tiempo estimado de la medida.
- Parámetros de la adquisición de cada canal. Esto es muy general y pueden ser ganancias, cantidad de promedios, o frecuencia de corte de un filtro.

3.3. Estructura de los paquetes

La comunicación entre dispositivos está basada en paquetes de datos, ver figura 3.1, que tienen la siguiente estructura:

- **DRN**: Dirección.
- **CMD**: Comando:
  - **RGT**: Registro.
  - **CNL**: Canal.
- **CUT**: Carga útil:
  - **LRG**: Largo de datos.
  - **DTS**: Datos.
- **CSM**: Suma de verificación.

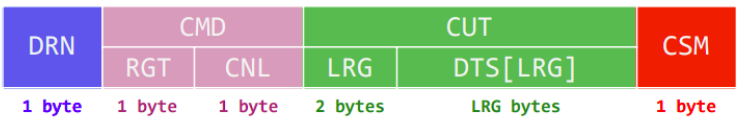


Figura 3.1: Estructura de los paquetes del protocolo.

Cada uno de esos campos se describe a continuación.

[2] Unidad temporal del funcionamiento del sistema.

3.3.1. Descripción

Dirección

La dirección es el *address* I2C del módulo. Tiene 1 byte de longitud y puede tomar valores entre 1 y 127. Se deja libre la 0 para futuras aplicaciones. Como convención, al módulo de alimentación se le asigna la dirección 1, el resto de las direcciones se dejan libres para los módulos de adquisición.

Comandos

El comando se compone de dos campos: Registro y Canal. Ambos tienen longitud de un byte y toman valores de 0 a 255. El Registro corresponde a la acción a realizar, siempre es un número par y la respuesta contiene el impar consecutivo.

Los registros se agrupan según su funcionalidad:

- En primer lugar los de configuración general del módulo son:
  - 0: Consultar tipo de módulo.
  - 2: Consultar cantidad de canales.
  - 4: Consultar largo de datos de cada canal.
  - 6: Consultar valores de la agenda del canal
  - 8: Actualizar valores de la agenda del canal
  - 10: Consultar cantidad de parámetros del canal.
  - 12: Consultar valores de los parámetros del canal.
  - 14: Actualizar valores de los parámetros del canal
- Seguido de los registros de adquisición:
  - 32: Lanzar lectura en canal N.
  - 34: Consultar estado de lectura en canal N.
  - 36: Obtener lectura en canal N.
- Luego los registros de gestión de energía:
  - 64: Ir a modo de bajo consumo.

En el caso de nuevas necesidades o agregar nuevos registros, se sugiere seguir esta estructura.

El Canal, si es distinto de cero, indica el número de canal al cual corresponde el registro; en el caso de ser cero, corresponde a un comando genérico. Cada canal representa una lectura, ya sea escalar o vectorial. Todo módulo esclavo contiene al menos un canal. Los posibles valores que toman corresponden a:

- $N > 0$: Número de canal.
- 0: Comando genérico, no dirigido a un canal específico.

Carga útil

La carga útil del paquete está conformada por los campos Largo y Datos. El primer campo se compone de dos bytes, puede tomar valores de 0 a 65535, e indica la longitud del campo Datos. Los datos del paquete dependen del módulo y del comando.

Para comandos con gran cantidad de datos (en el orden de 1k) se implementó el fraccionamiento de paquetes. Esto se debe a:

- Implementación en Arduino:
  - Las librerías usan buffers que no pueden ser mayores a 256.
  - Las librerías no usan memoria dinámica, sino que tienen todo el tiempo la RAM ocupada por los buffers de transmisión.
- La tasa de error y posibles fallas aumenta al tener paquetes muy largos.

La fragmentación debe ser definida previamente y que todos los módulos tengan configurado el mismo máximo de datos. Se recomienda fuertemente para paquetes largos.

Para esta aplicación se fragmentó el comando *Obtener lectura* a un largo de datos máximo de 128 bytes, que coincide con el largo máximo de datos que se obtiene en el sistema SARA-Bee. Cabe aclarar que para este caso no había necesidad de fraccionar pero se dejó implementado de forma de tener contemplada dicha necesidad en futuras aplicaciones.

Suma de verificación

La suma de verificación o *checksum* es el mecanismo del protocolo para definir la validez de un paquete. Se compone de un byte y corresponde a la suma de todos los bytes de todos los campos anteriores. El siguiente pseudocódigo ilustra este concepto:

```
dataSum = 0;
for(i=0; i<dataLenght; i++)
    dataSum += data[i];

checksum = address+register+channel+
    dataLenght[MSB]+dataLenght[LSB]+dataSum
```

Ejemplos

A continuación se muestran ejemplos para fijar mejor lo anteriormente descrito.

Analicemos la primer consulta que realiza el maestro a un módulo, por ejemplo a la dirección 1, *Consultar tipo de modulo*. La pregunta se compone de la dirección

I2C, en este caso 1, el canal es cero porque es una pregunta general y el largo de los datos es 0. El checksum es muy fácil de calcular para este caso y da 1. La respuesta tiene la misma dirección, el registro incrementado en 1, largo de datos igual a 1 y el valor del dato es 0, que se traduce a un módulo de alimentación. El checksum es 3.

Los paquetes son:

```
Pregunta:
    1-0-0-0-0-1

Respuesta:
    1-1-0-1-0-0-3
```

En lo anterior el guión separa los bytes y los números representan cada byte en decimal.

El segundo ejemplo es *Consultar agenda del canal*. La pregunta se compone de la dirección I2C, en este caso 1, el canal es el 1 y el largo de los datos es 0. El checksum es 8. La respuesta tiene la misma dirección, el registro incrementado en 1, largo de datos igual a 4 y el valor del datos son dos enteros, el primero es 6, que equivale el periodo en *ticks* y el segundo es el tiempo de medida, 1 *tick*. El checksum es 14.

Los paquetes son:

```
Pregunta:
    1-6-1-0-0-8

Respuesta:
    1-7-1-4-0-6-0-1-0-14
```

3.4. Comunicación entre módulos

3.4.1. Intercambio de paquetes

Existen dos formas de intercambiar información entre el maestro y el esclavo:

- El maestro desea transferirle datos al esclavo.
- El maestro desea que el esclavo le transfiera datos.

El primer intercambio es muy simple y necesita solo una etapa maestro-transmisor/ esclavo-receptor. El maestro solo debe armar y enviar un paquete de acuerdo al protocolo.

El segundo intercambio requiere de un paso adicional. Primero implica, al igual que el caso anterior, una etapa maestro-transmisor/esclavo-receptor, con un paquete compuesto por el registro, canal y datos necesarios para hacerle notar que a continuación va a realizarle una lectura y que debe tener listo un *buffer* con los datos solicitados. Estos datos pueden ser información del módulo, o una lectura de un canal. Luego realiza la etapa de maestro-receptor/esclavo-transmisor para obtener el paquete solicitado.

3.4.2. Inicialización y adquisición

Como se explicó anteriormente, el módulo principal tiene almacenado de forma volátil la información de los módulos en el sistema, por lo qué, al encenderse, debe escanear el bus y cargar los datos de los módulos presentes. Esta secuencia de preguntas y respuestas debe realizarse en el orden correcto, ya que el maestro debe ir caracterizando cada módulo comando a comando. El maestro debe sensar si hay un módulo en esa dirección y consultar de qué tipo es y cuántos canales tiene. Para esas preguntas, el maestro sabe qué cantidad de bytes debe esperar en la respuesta, por lo que tiene sentido que se realicen primero. En las preguntas siguientes, debe utilizar la información consultada previamente, en particular la cantidad de canales ya que debe recorrerlos de a uno y consultar: largo de lectura, agenda, cantidad de parámetros del canal y, en caso de haber parámetros, debe consultarlos.

La secuencia de inicialización tiene el siguiente orden:

1. Para cada módulo sensado:

    1.1. Consultar tipo de módulo.

    1.2. Consultar cantidad de canales.

2. Para cada canal del módulo:

    2.1. Consultar largo de datos.

    2.2. Consultar valores de la agenda.

    2.3. Consultar cantidad de parámetros.

    2.4. Consultar valores de los parámetros.

La Figura 3.2 ilustra más en detalle la primera y segunda consulta para un módulo de alimentación con la dirección 1 y dos canales. El resto de las consultas tienen un formato similar por lo que no aporta mostrar la secuencia completa.

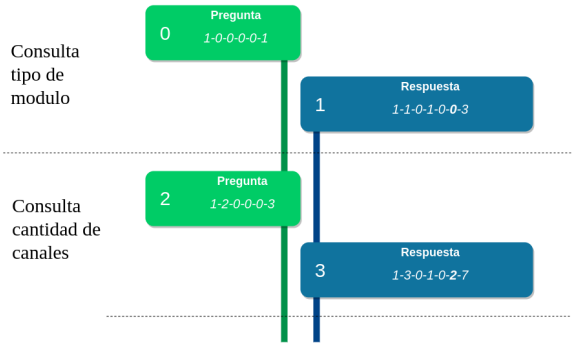


Figura 3.2: Ejemplo de inicialización del protocolo.

Luego de la inicialización el maestro tiene la información necesaria de cada módulo para comenzar la adquisición de datos en forma automatizada para todos los canales.

La secuencia de adquisición en cada canal se da de la siguiente manera:

- Esperar el período de adquisición.
- Enviar a adquirir.
- Esperar el tiempo de adquisición.
- Consultar estado de lectura.
- Si la lectura está pronta, solicitar lectura, de lo contrario esperar un tiempo determinado y volver a consultar.

En la figura 3.3 se ilustra gráficamente en el tiempo la secuencia anterior.

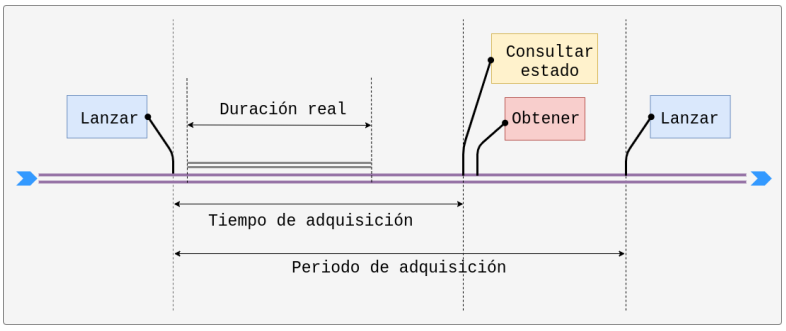


Figura 3.3: Diagrama de tiempos de la adquisición.

La figura 3.4 ejemplifica dicha secuencia para la adquisición de datos en el canal 2 en un módulo de alimentación de dirección 1. Luego de esperar el período de adquisición de dicho canal se lo envía a adquirir datos. Luego de que expire el tiempo de adquisición se le envía una consulta de lectura pronta, a la cual el módulo responde positivamente mediante un paquete con un dato de valor 1.

Finalmente, se solicita la lectura a través del comando *Obtener lectura*, el cual es respondido con paquete con largo de datos igual a 5. Los datos tienen la siguiente correspondencia:

- **0**: número de paquete, siempre es cero a excepción de los canales con largo de lectura mayor a 32 (128 bytes).
- **116-43-71-65**: valor de la lectura representado en punto flotante, en este caso: 12.4481[3].

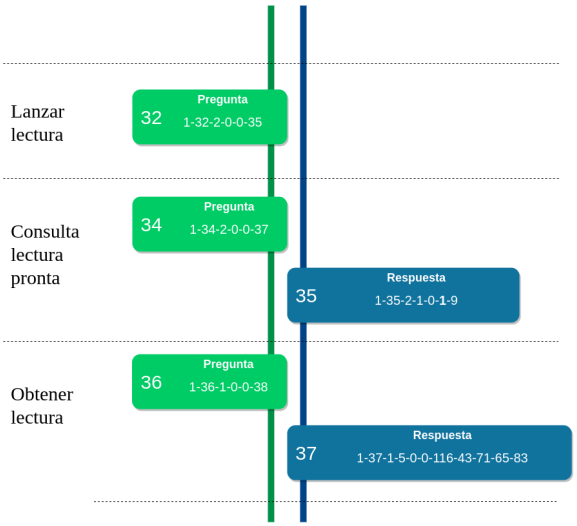


Figura 3.4: Ejemplo de adquisición del protocolo.

3.5. Librería implementada

Se implementó en Arduino una librería para SSCMP sobre la librería Wire [6], que funciona como una capa de abstracción de hardware. Esto permite que se pueda utilizar en cualquier placa de Arduino compatible con Wire.

La librería le provee al usuario una interfaz tanto para el maestro como para el esclavo.

En el módulo maestro se implementaron funciones para:

- Enviar paquete al esclavo.
- Solicitar y recibir paquete del esclavo.
- Verificar paquete.

En los módulos esclavo se implementaron funciones para:

[3]Sin unidades ya que es irrelevante para el sistema

- Recibir paquete del maestro.
- Enviar paquete al maestro cuando es solicitado.
- Verificar paquete.
- Procesar paquete recibido.

Al controlar la comunicación, el maestro controla también el comportamiento del esclavo, ya que éste solo actúa cuando se le indica. Una prueba de esto es que solo el esclavo procesa los paquetes recibidos. Los paquetes que recibe el maestro son respuestas de una pregunta previa.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

Módulo de adquisición de señales escalares

El módulo de adquisión de señales escalares, MASE, es el módulo más simple, se encarga de adquirir señales escalares y actúa como esclavo I2C en el equipo. Se encarga de adquirir magnitudes auxiliares al audio para poder caracterizar el estado de la colmena. Le proporciona al usuario un contexto de las condiciones ambientales a través de la temperatura y la humedad relativa. Brinda a su vez una idea del nivel de producción de miel de la colmena a través del peso. Cada medida se traduce como un canal del módulo con largo de lectura igual a uno.

La **humedad relativa** es la relación entre la cantidad de vapor de agua presente en el aire y la máxima cantidad que podría llegar a tener. Ésta puede tomar el valor máximo de 100 %HR que es cuando el ambiente se encuentra saturado de agua y el valor mínimo se da en los desiertos y es apenas inferior a 20 %HR.

En Uruguay la humedad relativa media anual oscila entre 70 %HR y 75 %HR, en todo el país [34]. El mes más húmedo es julio, con una media de 80 %HR, y el más seco es enero, con una media de 65 %HR. Es normal que durante el día la humedad relativa oscile entre 45 %HR poco después de mediodía, hasta valores superiores a 90 %HR en horas de la madrugada. Existen mapas con curvas de nivel para la humedad relativa en Uruguay en cada mes [19].

Las especificaciones para adquirir esta medida son:

- Valor mínimo: 20 %HR.
- Valor máximo: 100 %HR
- Incertidumbre: 5 %HR

La **temperatura** media anual en Uruguay es de 17,5 °C [34]. Las temperaturas extremas históricas van desde -11 °C, observada en Melo en un mes de Julio; hasta 44 °C, observada en Rivera en un mes de enero.

Mapas con curvas de nivel de las temperaturas medias en Uruguay en cada mes pueden ser encontrados en el sitio web de INIA [20] .

La especificaciones para la temperatura son:

- Valor mínimo: -15 ºC.
- Valor máximo: 45 ºC.
- Incertidumbre: 1 ºC.

Conociendo el **peso** de la colmena, se puede saber la cantidad de miel que ésta aloja. En particular, se puede relacionar la actividad recolectora de la colmena con la cantidad de abejas en la misma. Según la regla de Farrar, cuanto más aumenta la población de una colmena, mayor es la producción individual de cada abeja [45]. Esto se debe a que a medida que aumenta el número de abejas de una colmena, también aumenta la proporción de pecoreadoras, encargadas de la recolección de polen. Esto se traduce en la relación de la tabla 4.1.

Total de obreras	10000	20000	40000	60000
Porcentaje pecoreadoras	20 %	25 %	50 %	65 %
Peso de la población	1 kg	2 kg	4 kg	6 kg
Rendimiento de miel	1 kg	4 kg	16 kg	36 kg

Tabla 4.1: Regla de Farrar para la estimación del peso de la colmena.

Se asume una comunidad promedio de 60000 abejas. Aplicando la regla Farrar para este caso se obtiene 36kg de miel. Suponiendo un máximo de 10kg para el peso de la estructura de la colmena y las abejas se obtiene un total de 46kg. Con lo cual se considera razonable tomar como valor máximo de la balanza 50kg.

Las especificaciones para el diseño de la balanza son:

- Valor mínimo: $0kg$.
- Valor máximo: $50kg$.
- Incertidumbre: $0{,}5kg$.

Dado que el MASE será encargado de adquirir peso, humedad relativa y temperatura ambiente, tiene las siguientes características:

- Tipo de módulo: Adquisición multilectura.
- Dirección I2C: 2
- Cantidad de canales: 3
    - Humedad relativa en %HR.
    - Temperatura interna en ºC.
    - Peso en kg.

Cada canal tiene largo de lectura uno.

4.1. Elección de Hardware de Desarrollo

Como plataforma de desarrollo se eligió a la Arduino Uno, ver figura 4.1.

Las especificaciones técnicas de la placa son:

- Voltaje de operación: 5V
- Pines digitales IO: 14
- Pines digitales IO con PWM: 6
- Entradas analógicas: 6
- Memoria flash: 32 kB
- Memoria SRAM: 2 KB
- Memoria EEPROM: 1 KB

Se necesitaba un microcontrolador pequeño, ya que las restricciones para este módulo eran débiles. No era necesario realizar procesamiento complejo de datos, ni se requería una gran cantidad de memoria, o pines libres.

Es el modelo más simple, económico y popular de Arduino, por lo que se encuentra a un precio accesible en plaza y es suficiente para este desarrollo.

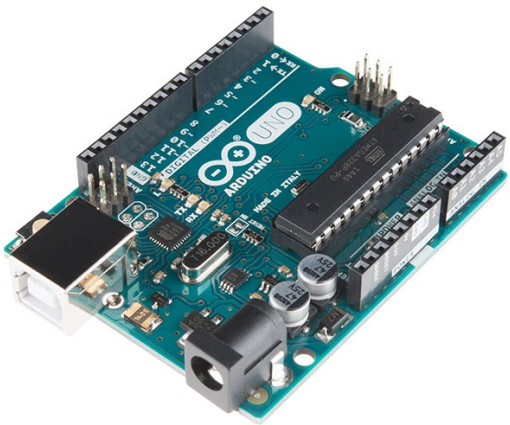


Figura 4.1: Arduino UNO utilizada para el desarrollo del MASE.

4.2. Elección de componentes

4.2.1. Medida de humedad relativa y temperatura

El circuito para la adquisición de temperatura y humedad se realizó en base al integrado DHT22, el cual se muestra en la figura 4.2.

Figura 4.2: Sensor de humedad relativa y temperatura DHT22.

Se eligió este componente debido a que adquiere dos de las tres medidas que debe realizar este módulo. No se necesita un circuito de adaptación para las lecturas. Existen librerías para la mayoría de las plataformas de Arduino y se puede conseguir en plaza a un precio accesible.

El sensor tiene las siguientes características proporcionadas por el fabricante:

- Rango de temperatura: De $-40°C$ a $125°C \pm 0{,}5°C$.
- Rango de humedad: De $0\,\%HR$ a $100\,\%HR \pm 5\,\%HR$.
- Frecuencia de muestreo: $0{,}5Hz$.
- Consumo: $2{,}5mA$.
- Voltaje de operación: De $3V$ a $5V$.

Por lo que efectivamente este sensor cumple los requerimientos para esta medida.

El sensor cuenta con cuatro pines:

1. **VCC**: Alimentación.
2. **Data Out**: bus bidireccional utilizado para transmitir los datos.
3. No conectado.
4. **GND** : Tierra.

Para un mayor nivel de detalle sobre el sensor y la interfaz con el microcontrolador referirse al Apéndice G.

4.2.2. Medida del peso

La implementación de la balanza fue realizada mediante dos etapas. La primera consistió en el sensado del peso a través de una configuración de celdas de carga, y la segunda la amplificación y conversión analógica digital.

Para el sensado del peso se utilizó las celdas de carga de la figura 4.3. Las celdas de carga equivalen eléctricamente a una resistencia variable. A su vez se puede dividir en una parte constante representada por un valor nominal dado, y una parte aditiva variable proporcional al valor de la fuerza impuesta sobre la celda en una dirección dada.

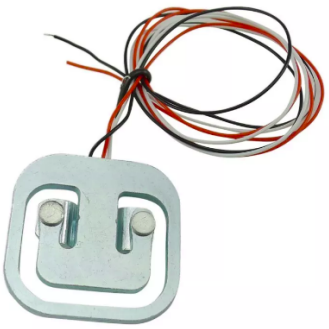


Figura 4.3: Celda de carga utilizada.

La celda utilizada se compone de dos galgas extensométricas. El circuito equivalente eléctrico de la celda de carga es el de la figura 4.4. Los extremos de la rama son los cables blanco (1) y negro (3) y el punto medio es el rojo (2). Se representa cada galga con una resistencia variable, con valores nominales idénticos. Dichas galgas fueron ensambladas mecánicamente de forma de que la primera (1-2) tiene un coeficiente positivo, y la segunda (2-3) tiene un coeficiente negativo.

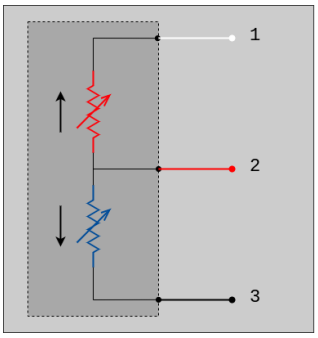


Figura 4.4: Equivalente eléctrico de la celda de carga.

La celda de carga tiene las siguientes características proporcionadas por el fabricante:

- Peso máximo: 50kg.
- Valor nominal de cada resistencia: $1000\Omega \pm 20\Omega$.
- Alimentación: De 0V a 10V.

La segunda etapa es de amplificación y conversión analógica-digital, es implementada en base al integrado HX711. Se utilizó el módulo de la figura 4.5.

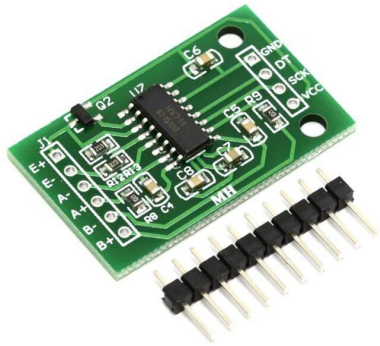


Figura 4.5: Módulo HX711

Los pines del módulo HX711 son:

- **E+**: Salida analógica. Alimentación analógica para el puente de Wheatstone.
- **E-**: Salida analógica. Tierra analógica para el puente de Wheatstone.
- **A-**: Entrada analógica. Entrada negativa de canal A.
- **A+**: Entrada analógica. Entrada positiva de canal A.
- **B-**: Entrada analógica. Entrada negativa de canal B.
- **B+**: Entrada analógica. Entrada positiva de canal B.
- **VCC**: Entrada digital. Alimentación proporcionada por el microcontrolador.
- **GND**: Entrada digital. Tierra digital proporcionada por el microcontrolador.
- **PDSCK**: Entrada digital. Control de bajo consumo y reloj serial.
- **DOUT**: Salida digital. Salida serial de datos.

En el Apéndice H se explica con mayor profundidad los aspectos relacionados a este integrado.

4.3. Arquitectura de Software

La Arquitectura elegida para el desarrollo del firmware es Round-Robin con interrupciones. La decisión fue en base a la naturaleza simple del problema a resolver. No se necesitaba una arquitectura de mayor complejidad. Las interrupciones son necesarias para las comunicaciones por I2C y UART. En el diagrama de la figura 4.6 se resume el funcionamiento del Round-Robin implementado.

Para I2C se desarrolló una librería que implementa el protocolo SSCMP. Cada vez que se recibe un paquete correcto, en la interrupción se levanta un flag de paquete recibido y se atiende esa solicitud en el loop principal. Luego se procesa dicho paquete de acuerdo al protocolo. En el diagrama esto se observa en **Procesar Paquete**.

Para la UART se adaptó una *shell*[1] para Arduino y se implementaron comandos con las mismas funcionalidades del protocolo. En este caso, la comunicacón es más simple porque no tiene las limitaciones de I2C. Cuando se recibe un nuevo comando, la interrupción levanta un flag de comando recibido. En el loop principal la shell se encarga de atender y procesar dicho comando, en el diagrama esto se ve como **Procesar Comando**. La UART se configuró a 115200bps.

Los comandos UART implementados son:

- "run": *Run*, lanza una lectura y devuelve el valor de ésta. A diferencia de I2C en este caso se necesita sólo este comando.
- "qry": *Query*, comando de solicitud de datos del módulo:
  1. Dirección.
  2. Tipo de módulo.
  3. Cantidad de canales.
  4. Largo de lectura de los canales
- "set": *Set*, comando para modificar datos del canal:
  1. Agenda.
  2. Parámetros.
- "get": *Get*, comando para solicitar datos del canal:
  1. Agenda.
  2. Parámetros.
- "lpm": *Low power mode*, envia el módulo a bajo consumo.

[1] Intérprete de comandos seriales, similar a una terminal de PC.

- "rst": *Reset*: comando para restaurar valores por defecto.

Por más detalle referirse al manual de usuario.

En el caso de que haya una solicitud de lectura nueva en un canal dado, ya sea por un comando UART o un paquete I2C, se levanta el flag correspondiente y se lo baja al iniciar la lectura en dicho canal. En el diagrama equivale a **Adquirir Lectura**.

En el caso de que la lectura sea realizada a través de I2C, al finalizar la adquisición se sube un flag de lectura pronta y se guarda la lectura en un buffer esperando a que el maestro la solicite. En el caso de que sea por UART, se devuelve el valor adquirido.

Si el sistema no tiene ninguna tarea pendiente, esto es: procesar algún comando UART o paquete I2C, o adquirir una nueva lectura, éste se va a **Bajo Consumo** donde apaga todos los módulos de hardware a excepción de I2C y UART.

La interfaz del sensor DHT22 y el integrado HX711 con el microcontrolador fue realizada con las librerías DHTlib [43] y HX711 [8] respectivamente, desarrolladas para Arduino.

Los módulos de software desarrollados son:

- **mase**: *main file* del proyecto.
- **configuracion**: Definiciones para la configuración general del módulo.
- **tipos**: Definiciones de tipos y constantes del módulo y los canales.
- **saraI2C**: Módulo del protocolo SSCMP
- **registros**: Funciones para procesar los paquetes del protocolo SSCMP.
- **saraSerial**: Módulo *shell* serial.
- **comandosSeriales**: Funciones de *callback* para los comandos de la shell serial.
- **sensores**: Módulo de interfaz con los sensores DHT22 y HX711.
- **bajoConsumo**: Módulo de gestión de consumo.

La descripción en detalle del código desarrollado para este módulo se encuentra en el repositorio de Gitlab del proyecto con su respectiva documentación Doxygen [36].

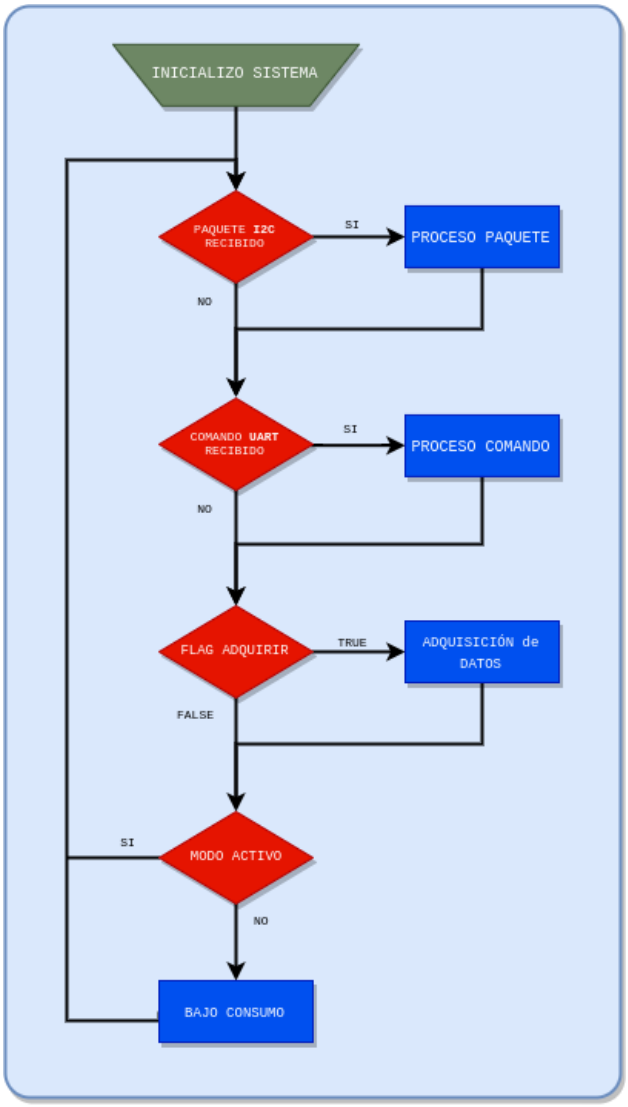


Figura 4.6: Diagrama del Round-Robin del MASE.

4.4. Diseño de hardware

4.4.1. Medida de humedad relativa y temperatura

El sensor DHT22 se encarga tanto de la adquisición como de la conversión analógica digital de las lecturas, por lo que no es necesario ninguna adaptación.

En la figura 4.7 se muestra una imagen del sensor implementado en base al DHT22 y el conector GX16-3p. El *pinout* es el siguiente:

1. **GND**.
2. **Data Out**.

3. **VCC**.

Por más información referirse al manual técnico.

Figura 4.7: Sensor de humedad relativa y temperatura DHT22.

4.4.2. Medida del peso

La primera etapa de la adquisición consistió en el sensado del peso a través de un puente de Wheatstone implementado por celdas de carga de 50kg.

La elección de la conexión y la configuración de las galgas se realizó en base a criterios eléctricos y mecánicos. A nivel eléctrico se desea:

- Optimizar la sensibilidad del puente de Wheatstone.
- Minimizar el consumo.

A nivel mecánico se desea:

- Las galgas deben funcionar como soporte, esto implica que debían estar distribuidas de forma de proporcionar estabilidad a la balanza.
- El peso de un objeto no puede depender de la posición en que sea ubicado en la balanza, es decir si está sobre un borde o en el centro la balanza debe medir lo mismo.

La primera configuración posible es la que se observa en la imagen a la izquierda en la figura 4.8. Esta configuración tiene la ventaja de utilizar solo una celda, y dos resistencias de valor nominar igual al de la galga. A nivel eléctrico la sensibilidad es la mínima posible. A nivel mecánico no es una buena configuración, ya que dificulta la estabilidad en el sistema sin modificar la medida, y la medida es dependiente de la posición del objeto.

La segunda configuración se observa en la imagen a la derecha de la figura 4.8. Se usan dos celdas conectadas por sus extremos opuestos: 1-3 y 3-1. En este caso

se duplica la sensibilidad del puente pero persiste el problema mecánico, por lo que se necesita una configuración más robusta.

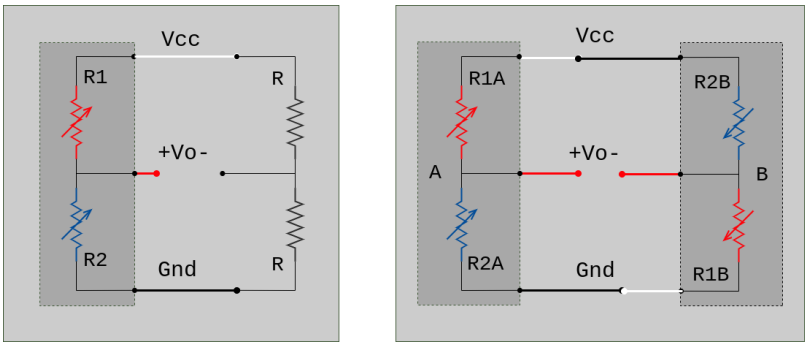


Figura 4.8: Esquema de conexión del puente usando una y dos celdas de carga.

Finalmente se propone la configuración de la figura 4.9. En ésta se utilizan cuatro galgas, conectando el extremo superior (1) al inferior (3) de la siguiente galga, y los centrales (2) son las entradas y salidas del puente.

Para la plataforma de la balanza se utilizó una placa de acrílico rectangular con las siguientes dimensiones:

- Espesor: 10mm.
- Largo: 50cm.
- Ancho: 30cm.

Se diseñó e imprimió en 3D un zócalo que se adhiere al acrílico y actúa como soporte para las galgas.

En esta configuración se cuatriplica la sensibilidad del puente de la configuración inicial. Se resuelve el problema mecánico al haber ubicado las galgas en la parte inferior de la placa, una en cada esquina. A su vez se compensan los efectos tanto del ruido como de la temperatura debido a la simetría del conexionado.

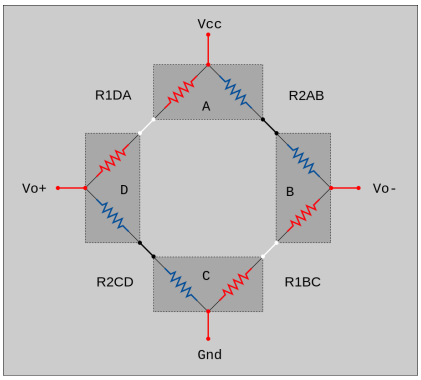


Figura 4.9: Esquema de conexión del puente usando cuatro galgas.

La segunda etapa es de amplificación y conversión analógica digital, es implementada por el módulo basado en el integrado HX711. En los pines E+ y E- se conectan Vcc y Gnd del puente respectivamente. En los pines A+ y A- el voltaje diferencial de los puntos medios de las ramas.

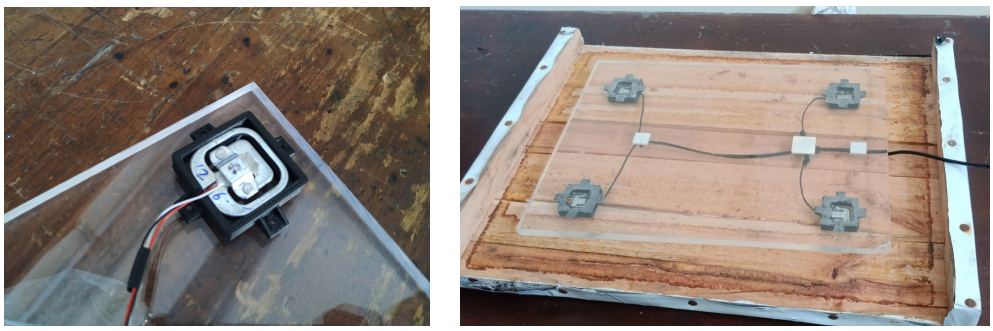
Dada la conexión realizada y configuración que utiliza el módulo adquirido se tienen las siguientes características:

- La frecuencia máxima de adquisición de datos es de 10 Hz.
- La alimentación proviene de los 0V y 5V provenientes del bus unificado.
- Se utiliza solamente el canal A.
- La ganancia utilizada para el canal A es de 128, que se configura serialmente durante las mediciones.
- El voltaje diferencial máximo a la entrada es de aproximadamente 39.0mV.

Finalmente, como se observa en la figura 4.9 se implementó la balanza con ésta configuración, utilizando un cable de cuatro hilos y un conector GX16-4p. El *pinout* es el siguiente:

1. **GND**.
2. **A+**.
3. **A-**.
4. **VCC**.

Por más información referirse al manual técnico.



(a) Galga y zócalo diseñado (b) Implementación final de la balanza.

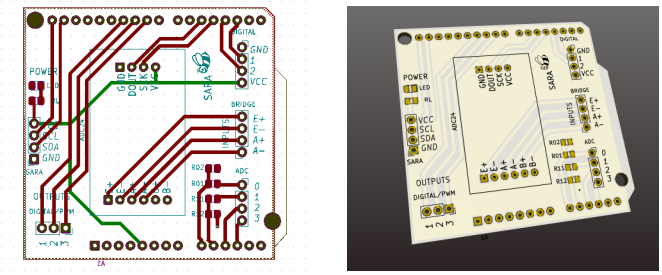
Figura 4.10: Imagen de la balanza implementada.

4.4.3. Diseño de shield

Para el diseño del PCB se utilizó la herramienta KiCad [24]. El shield diseñado para este módulo no contiene ningún componente discreto, simplemente tiene las pistas que une el microcontrolador al sensor DHT22 y al módulo del integrado HX711.

A su vez se agregaron conectores para la reutilización de este shield en el módulo de alimentación; tres conectores para salidas PWM para futuros usos y un LED para corroborar que la placa esté alimentada.

En la figura 4.11 se muestra el shield diseñado y el modelo 3D del mismo.

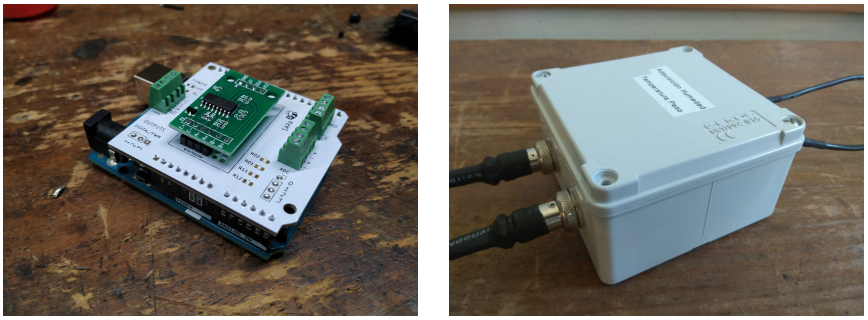


(a) Layout (b) Diseño 3D

Figura 4.11: Imagen del PCB del MASE.

En la imagen de la izquierda de la figura 4.12, se puede observar el módulo luego de ser fabricado y ensamblado conectado a la Arduino UNO.

En la imagen de la derecha de la figura 4.12, se observa el módulo completo ensamblado con dos conectores GX16-7p, cortocircuitados internamente, para el bus unificado. Además un conector GX16-4p para la balanza y un conector GX16-3p para el sensor de humedad y temperatura.



(a) Imagen del shield ensamblado del MASE (b) Implementación final del MASE

Figura 4.12: Imágenes del MASE implementado.

4.5. Pruebas y Validación de módulo

4.5.1. Sensor de humedad relativa y temperatura

Se debió integrar el sensor al módulo y corroborar que efectivamente las medidas fueran correctas.

Primero se realizó la prueba de comunicación con el microcontrolador, en base a la librería DHTLib para Arduino.

Luego se implementó un firmware básico en Arduino que realizara la adquisición de humedad relativa y temperatura cada un minuto y loggueara serialmente los datos. Se dejó por dos días y luego se contrastaron los datos adquiridos con medidas reales obtenidas en la web de Inumet.

Se obtuvieron resultados satisfactorios del sensor para ambas lecturas.

4.5.2. Balanza

Para el caso de la balanza las pruebas se realizaron en varias etapas, a medida que se iba diseñando y construyendo.

Se realizó una prueba de comunicación del módulo con el microcontrolador usando la librería HX711 para Arduino. En la cual se simuló el puente mediante potenciómetros. Luego se realizó la misma prueba de adquisición aplicando fuerzas sobre las galgas y observando su variación.

Se ensambló un prototipo inicial, a modo de prueba de concepto de la balanza. Se realizaron pruebas y una calibración del mismo. Se implementó un firmware en Arduino que realizara la adquisición periódicamente cada 10 segundos y loggueara serialmente la lectura. Durante la prueba se iba aumentando el peso y contrastando con una balanza comercial.

Las especificaciones de la balanza comercial son:

- Máximo peso: 150kg
- Resolución: 100g

Finalmente, se implementó la versión final de la balanza. Se realizaron las medidas de la tabla 4.2. Se puede notar que la balanza no mide pesos inferiores a 5kg. Debido a aspectos mecánicos, se necesita alrededor de 2kg para que se estabilice sobre la superficie, y en 5kg comienza a tener medidas válidas. En el intervalo 5kg a 30kg hay una relación lineal. Para valores cercanos a 35kg comienza a distorsionarse la medida. Ésto se debe a la deformación de los soportes impresos en 3D los cuales son de un material frágil no pensado para este tipo de aplicaciones.

Ésto se asentúa cada vez más con el peso. De no ser por este comportamiento la balanza podría llegar a 200kg debido a la configuración de las 4 galgas de 50kg.

Peso real (kg)	Peso medido (kg)
5.0	5.0
10.0	10.0
15.0	15.0
20.0	20.0
25.0	25.0
30.0	30.0
35.0	34.9
40.0	39.9
45.0	44.8
50.0	49.8

Tabla 4.2: Prueba de la balanza del MASE

Las especificaciones finales para el peso son:

- Valor mínimo: $5kg$.
- Valor máximo: $50kg$.
- Incertidumbre: $0{,}2kg$.

4.5.3. Comunicación

La comunicación para este módulo se divide en la comunicación I2C para poder transmitir de forma remota y el UART utilizado para la comunicación serial a nivel local.

Para la comunición con la UART se realizaron pruebas para todos los comandos. Esta funcionalidad facilitó el debugging en etapas posteriores.

Como primera instancia de prueba para la comunicación I2C se implementó un módulo simulador. Éste se configuró para adquirir cada canal cada 30 segundos y que se le solicitara la lectura 5 segundos después. Esta primera simulación fue exitosa.

Luego se procedió a probarlo junto al Módulo Principal, dichas pruebas se encuentran explicadas en el capítulo 6.

4.6. Trabajos a futuro

El objetivo del equipo construido es funcionar de forma continua por grandes períodos de tiempo en la intemperie.

Según se investigó uno de los problemas del sensor utilizado de humedad es que estos presentan sulfatamiento ante la exposición continua en ambientes donde la humedad es alta.

Dados los plazos del proyecto, no fue posible realizar los tests adecuados en el sensor de humedad y cuantificar de alguna forma este problema, por lo que se propone investigar sensores de humedad relativa para condiciones de humedad cercana a 100 %HR.

Por otra parte la balanza diseñada tiene varios aspectos a mejorar en pos de obtener un rango mayor, algunos de ellos son:

- Hacer los soportes con un material más resistente de forma que no se deforme y no interfiera en la medida.
- Agregar una base más robusta de forma de ser menos dependiente de la superficie en la cual está apoyada.
- Estudiar otras posibles configuraciones.
- Investigar celdas de carga de mejor calidad.

Una alternativa que se consideró durante el diseño es conectar la alimentación del puente a los 12V del bus en lugar a 5V. Esto aumentaría la sensibilidad por un factor de 2,4 veces. Finalmente se descartó porque incrementaba el consumo con ese mismo factor. En la aplicación no era necesaria mayor resolución, por lo cual la relación costo/beneficio resultó deficiente.

Capítulo 5

Módulo de adquisición de señales vectoriales

Este módulo se encarga de adquirir señales vectoriales y actúa como esclavo I2C en el equipo. La señal vectorial de interés es el audio, se utiliza como instrumento para el monitoreo de colmenas de abejas. El audio que se adquiere para esta aplicación tiene determinadas especificaciones que determina los parámetros de la adquisición y acondicionamiento previo.

Se cuenta con cuatro canales de audio, corresponden a cuatro módulos con micrófonos y etapas de amplificación y filtrado. Las señales se adquieren con micrófonos analógicos, pasan por dos etapas de amplificación, una de ganancia fija y otra de ganancia variable, y por una etapa de filtrado. El filtro tiene una frecuencia de corte determinada por el zumbido máximo de las abejas. Para comprender las decisiones tomadas, se explica a continuación las especificaciones de las señales de interés.

- La ganancia de la etapa de amplificación se decide según un estudio realizado por el proyecto que antecede, *Pestibee*. El estudio dice que estas señales apenas alcanzan los 200mV de voltaje de pico a pico, y se concluye que las señales rondan los 20mVpp, se puede observar en la figura 5.1. Para alcanzar la máxima excursión, se tiene en cuenta el voltaje de operación de la placa adquisidora, ésta opera a 3.3V. Se propone a priori una ganancia de 100V/V; y la posibilidad de tener otras ganancias distintas. Aquí nace la idea de definir dos etapas de amplificación, una con ganancia fija, y la otra con ganancia variable.

- Frecuencia de corte para la etapa de filtrado se definió en 1500Hz. Según investigadores, las abejas a estudiar baten sus alas hasta una frecuencia máxima de 700Hz, aunque en comportamientos atípicos pueden llegar a una frecuencia más elevada [23]. Se eligió extender el ancho de banda hasta 1500 Hz.

- Frecuencia de muestreo: 4kHz. La mínima frecuencia de muestreo aceptable para la reconstrucción de la señal, según el teorema de Nyquist, sería 3kHz. Se tiene en cuenta que para la adquisición de audio es recomendable considerar al menos un 10 % más, se eligió, por tanto, muestrear a 4kHz.

- Resolución en frecuencia de 2Hz. Se van a tomar ventanas de muestreo de 500ms. El tiempo entre ventanas es 0. Cada ventana es de 2048 muestras, las cuales se las procesará a través de la Transformada Rápida de Fourier. El número de ventanas es parametrizable, se eligió adquirir 32 ventanas de 500ms, o lo que es lo mismo, 16 segundos de muestreo. Las muestras que se van a transmitir al módulo principal son muestras representativas de la señal adquirida. Estas muestras van a formar un histograma de energía por bandas en el espectro. Para ello, se realiza primero un promedio de las muestras en frecuencia entre las 32 ventanas adquiridas. Luego, se toma el módulo de las ventanas promediadas y se realiza un promedio por bandas, en la porción del espectro de interés. Se divide al espectro en 32 bandas. Estos promedios por bandas son los que se envían al módulo principal, como valores representativos de la señal adquirida.

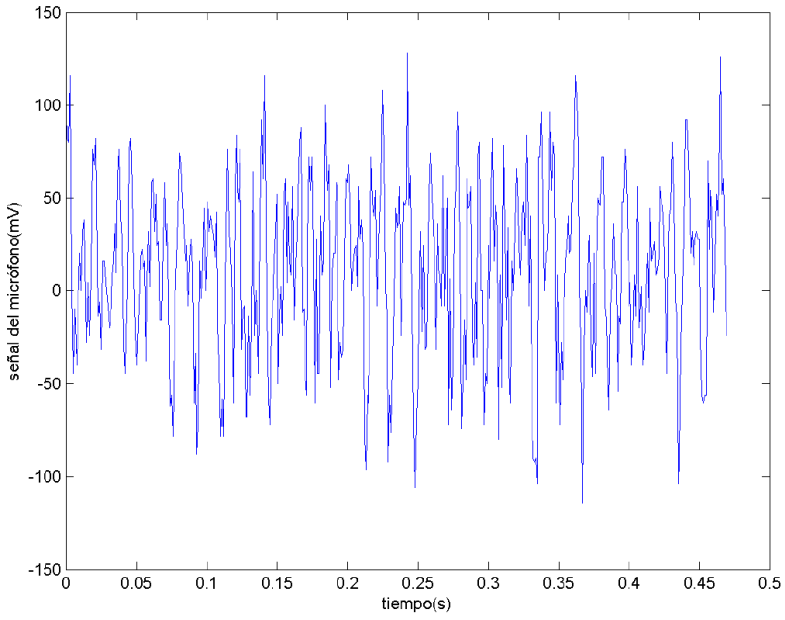


Figura 5.1: Señal de abejas obtenidas del proyecto PestiBee

En síntesis, las características de este módulo dentro del sistema son:

- Tipo de módulo: Adquisición no-multilectura.

- Dirección I2C: 3

- Cantidad de canales: 4

- Lectura del canal: Potencia del audio adquirido en 32 bandas hasta 4kHz en el respectivo micrófono.

5.1. Elección de Hardware de Desarrollo

En el inicio del proyecto iban a desarrollarse dos módulos distintos, uno que adquiriera audio, y uno que lo procesara. En ésta lógica se consideró que la placa adquisidora fuese una Arduino UNO. Pero sus escasos 2kb de SRAM y un ADC con 10 bits de resolución, no lo hacían un buen candidato para adquisición del audio. Por lo cual se consideraron las siguientes alternativas: Arduino MEGA, Raspberry PI y Arduino MKR Vidor 4000. Las dos primeras placas de desarrollo mencionadas requerirían un ADC externo. En todas las opciones se era capaz de adquirir y procesar en el mismo hardware, por lo cual se decidió unificar las tareas en un único módulo. Fue durante el proceso de la selección del hardware que se conoció por parte del grupo la existencia de la placa Arduino MKR Vidor 4000. Se estudiaron las placas candidatas, el análisis de estas se puede observar en la tabla 5.1. Las características principales tenidas en cuenta son el consumo (en reposo), resolución del ADC y memoria.

Luego de evaluar las ventajas y desventajas de cada placa, y teniendo presente las especificaciones de diseño, se decidió finalmente volcarse por la Arduino MKR Vidor 4000, ya que sus características cumplen con la necesidad del proyecto. Si bien la Raspberry Zero es considerablemente mejor en memoria, esta placa no tiene un ADC interno. Además, la Arduino elegida tiene un plus, y es que incluye un FPGA, donde se realizaría el procesamiento de audio. La FPGA es básicamente una matriz programable de puertas lógicas. Esta tecnología se desarrolla en mayor profundidad en el apéndice E.

Crear un módulo de adquisición y procesamiento de señales en la Raspberry Pi Zero queda por fuera del alcance del proyecto, pero se sugerirá como mejoras a futuro.

Placas	**Procesador**	**VCC**	**RAM**	**Flash**	**CLK**	**ADC**	**Comunicación**	**Puertos analógicos**	**Consumo**
Arduino Uno	ATmega328P	5V	2KB	32KB	16MHz	10 bits	UART, SPI, I2C	6	46mA
Arduino MEGA	ATmega2560	5V	8KB	256KB	16MHz	12 bits	4 UARTs	16	93mA
Arduino	ATSAMD21	3.3V	32KB	256KB	48MHz	8/10/12	UART, SPI, I2C	7	100mA
MKRVidor4000	Cyclone 10CL016	3.3V	8MB	2MB	48-200MHz		UART, SPI, I2C		
Raspberry Pi Zero	BCM2835	5V	512MB	uSD-OS	1.2-1.4GHz	No	GPIO, SPI	3 GPIO	80mA

Tabla 5.1: Comparación de plataformas para MASV

5.1.1. Arduino MKR Vidor 4000

Lo primero a conocer de esta placa, es que tiene un FPGA Intel Cyclone 10 [1], y un microprocesador Atmel [2]. Se observa en la figura 5.2.

Microcontrolador

El microcontrolador es ATSAMD21 (procesador Arm Cortex-M0+), las especificaciones que vale la pena mencionar son:

Figura 5.2: Arduino MKR Vidor 4000

- Voltaje de alimentación: 5V
- Voltaje de operación del circuito: 3.3V
- ADC de 12 bits
- 22 pines digitales, 7 analógicos
- Memoria flash: 256kB
- SRAM: 32KB
- Frecuencia de reloj: 48MHz
- Comunicación I2C y SPI

[1] FPGA Intel 10CL016YU256C8G
[2] Microcontrolador Intel SAMD21G18A-U

FPGA

El FPGA es de Intel, Cyclone 10CL016, se conecta con el microcontrolador y con pines al mundo. El FPGA cuenta con las siguientes especificaciones:

- Voltaje de operación 3.3V
- Flash memory 2MB
- SDRAM 8MB
- Reloj 48MHz, hasta 200MHz
- 15K elementos lógicos programables
- 510K bits de memoria embebida (56 bloques de 9K)

¿Cómo se programa la FPGA?

La programación de las FPGA de Intel se realiza a través de Quartus. Quartus es un Software creado por Altera, para el análisis y la síntesis de diseños realizados. Se programa en un lenguaje de descripción de Hardware ó HDL. El código se programó en VHDL. Resta aclarar que para la programación de esta FPGA, la versión compatible del software es Quartus Prime Lite Edition, libre; o Quartus Prime Standard Edition, pago. Se utilizó la versión libre de Quartus.

La FPGA sobre la Arduino MKR Vidor 4000 se puede programar de tres maneras:

1. Flasheando la imagen con el código Arduino, utilizando un archivo .ttf.
2. Programando la imagen en RAM mediante USB Blaster.
3. Programando la imagen en RAM a través del emulador USB Blaster vía el microcontrolador.

Se eligió la primer opción: a través del archivo con extensión .ttf, *Tabular Text File*. Es un archivo en ASCII que contiene datos de configuración del programa, equivalente a un *Raw Binary File* en decimal. Este archivo .ttf se genera al compilar el código en Quartus. Luego, se debe invertir este archivo *byte a byte*, ya que así lo requiere el FPGA. Esta conversión se hace a través de un script de Java: *ReverseByte* [9]. Por último, se debe agregar este archivo creado como un header (".h") en el Sketch de Arduino. El microcontrolador carga este archivo en la FPGA, logrando la programación del circuito deseado.

Vale la pena mencionar que las otras opciones de programación para la FPGA no eran posibles o viables para el proyecto. Programar la imagen en RAM no es funcional para esta aplicación, ya que se debería cargar el programa cada vez que se encienda el sistema. Además, para conectarse mediante el USB Blaster, se debe tener acceso a los pines *JTAG, Joint Test Action Group*, y éstos no están

accesibles en la placa; se debería localizarlos, soldar cables para tener acceso, y adaptarlos al conector USB del USB-Blaster. Tarea tediosa, qué, además tiene otro inconveniente, aún logrando este cometido, para conectar la FPGA de esta forma, el microcontrolador debe estar en modo *Bootloader* (o de arranque), lo que imposibilita que el microcontrolador y la FPGA funcionen en forma simultánea.

5.2. Elección de componentes

5.2.1. Elección de los micrófonos

Los transductores elegidos son los micrófonos *electret*, y son una variable de los micrófonos condensador, se observan en la figura 5.3.

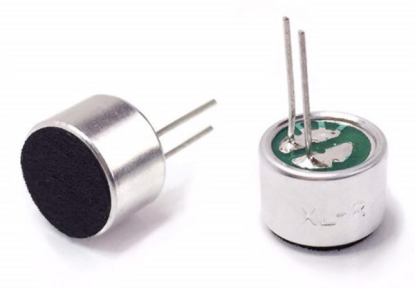


Figura 5.3: Micrófono electret

El campo de polarización requerido por los micrófonos se crea por un *electret*. Éste es un material dieléctrico que es capaz de sostener una polarización eléctrica permanente, similar a la polarización magnética de un imán. Por lo cual el diafragma se encuentra siempre polarizado, sin requerir de una tensión de polarización.

Los capacitores electret presentan las siguientes ventajas:

- No necesita tensión de polarización
- Construcción robusta
- Alta capacitancia, alrededor de los 500pF
- Bajo costo
- Baja sensibilidad a la humedad, y cambios de temperatura, por lo que pueden emplearse en ambientes exteriores.
- Tienen respuesta en frecuencia aceptable para el rango de frecuencias entre 50 Hz y 15000 Hz.
- Sensibilidad entre -50dB y -70dB.

Los micrófonos electret de mayor calidad entre los que se consiguen en plaza, incorporan un preamplificador de transistor de efecto de campo (FET), conectado como Common Source, para adaptar su impedancia alta, y aumentar la señal. Por lo que estos sí requieren alimentación, entre 2V y 12V. La configuración para el micrófono es la que se observa en la figura 5.4.

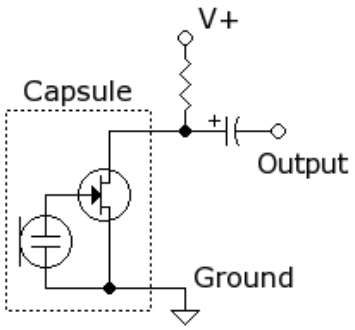


Figura 5.4: Conexión micrófono electret

La elección de los valores de los componentes no es crítica.

- Condensador entre 220nF y 220μ F. Se eligió un condensador de 100μ F.
- Resistencia: la impedancia de salida del micrófono será igual a esta resistencia, e interesa que sea baja. Se recomienda 470Ω para una alimentación de 5V.

Estos micrófonos tienen como desventajas:

- Saturan a alto volumen. El pre-amplificador interno satura cuando recibe una presión sonora alta.
- Baja respuesta en alta frecuencia.

Este último punto no afecta en las decisiones del proyecto, ya que la señal de interés no tiene componente en altas frecuencias.

5.2.2. Módulo Micro SD

Para el almacenamiento local de las muestras se agrega un módulo micro SD, ya que la placa no brinda la suficiente memoria para almacenar el audio adquirido. Este módulo se consigue fácilmente en tiendas electrónicas, y es de bajo costo, se lo observa en la figura 5.5. Se comunica mediante SPI. Cuenta con un circuito de conversión de voltaje para comunicarse tanto a 3.3V como a 5V, lo que facilita la adaptación a las distintas placas de nuestro proyecto. En particular se lo utiliza en el Módulo Principal (Arduino Mega - voltaje de operación de 5V), y en el Módulo presente de este capítulo (voltaje de operación de 3.3V). Puede ser alimentado hasta 5V debido a su regulador de voltaje incluido.

Los pines de este módulo son:

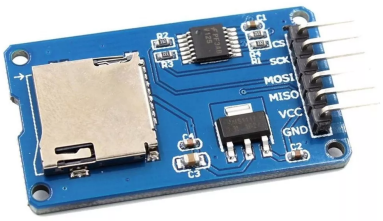


Figura 5.5: Módulo Micro SD para almacenamiento de datos local

* **CS** : *Chip Select*, habilita la comunicación con el esclavo SPI .
* **SCK** : *Serial Clk*, para la sincronización.
* **MOSI** : *Master Output, Slave Input.*
* **MISO** : *Master Input, Slave Output.*
* **VCC** : Alimentación hasta 5V.
* **GND** : Tierra, o voltaje de referencia.

5.2.3. Convertidor de nivel de voltaje bidireccional

Para la comunicación I2C de este módulo esclavo con el módulo principal, se conectó en los hilos de comunicación un convertidor de nivel de voltaje bidireccional, debido a la diferencia en los voltajes de operación. Se consiguió un módulo en plaza, modelo MOD-CXI2C, de baja costo, con 4 canales bidireccionales de los cuales se utilizaron solamente 2. El módulo se puede observar en la figura 5.6. Los

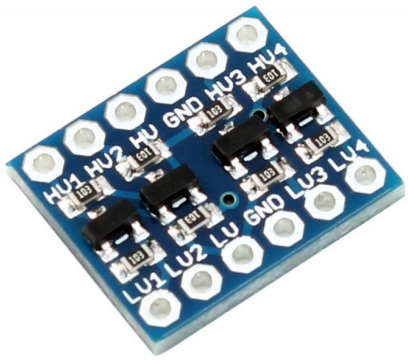


Figura 5.6: Convertidor de nivel de voltaje, para la comunicación I2C

pines a utilizar de este módulo son:

* HV1 : SDA de I2C, proveniente del módulo principal.
* HV2 : SCK de I2C, proveniente del módulo principal.

* HV : Alimentación 5V.
* GND : Ambas tierras deben estar al mismo potencial.
* LV1 : SDA de I2C, proveniente del módulo de adquisición de señales.
* LV2 : SCK de I2C, proveniente del módulo de adquisición de señales.
* LV : Alimentación 3.3V.

5.3. Arquitectura de Software

En este módulo se implementó la comunicación con el módulo principal, la adquisición de 4 canales de audio, las operaciones con las muestras procesadas en el microcontrolador, y el almacenamiento en el módulo micro SD de las muestras crudas y las procesadas. El procesamiento de las señales se realizó en la FPGA, aplicando FFT.

El sistema diseñado envía m muestras de una señal en el tiempo a la FPGA. Estas muestras tienen resolución *bit_depth*. La FPGA calcula su FFT y las devuelve al microcontrolador. La cantidad de muestras es parametrizable. Para la comunicación micro-FPGA, se utiliza un puerto SPI, de tamaño de palabra enviada y recibida *N_SPI*, también parametrizable. Entre el micro y la FPGA hay flujo de datos a procesar, datos procesados y comandos de instrucciones. La FPGA actúa como esclavo, es decir, espera recibir instrucciones desde un sistema maestro: el microcontrolador. La transmisión de datos SPI es de a 8 bits por limitaciones en la implementación de la librería SPI en el micro. Las especificaciones de diseño definen que las muestras a enviar son de 16 bits, por lo tanto se realiza el doble de transmisiones respecto a la cantidad muestras.

Cuando el microcontrolador recibe la ventana de muestras procesadas de la FPGA, la agrega a un promedio que realiza de todas las ventanas adquiridas. Al alcanzar las 64 ventanas de 500ms, se calcula un promedio por bandas. Estas bandas conforman la sección del espectro de interés. Este último promedio por bandas es el que se envía al módulo principal.

La arquitectura de este módulo se observa en la figura 5.7.

5.3.1. Microcontrolador SAMD21

La arquitectura elegida para este módulo es Round-Robin con interrupciones, con una máquina de estados para la adquisición y procesamiento de los datos. Se tomó como base el firmware desarrollado para el MASE, y se añadió la máquina de estados que se encarga de adquirir, procesar y guardar las muestras y los datos procesados. Las interrupciones son necesarias para la comunicación por I2C.

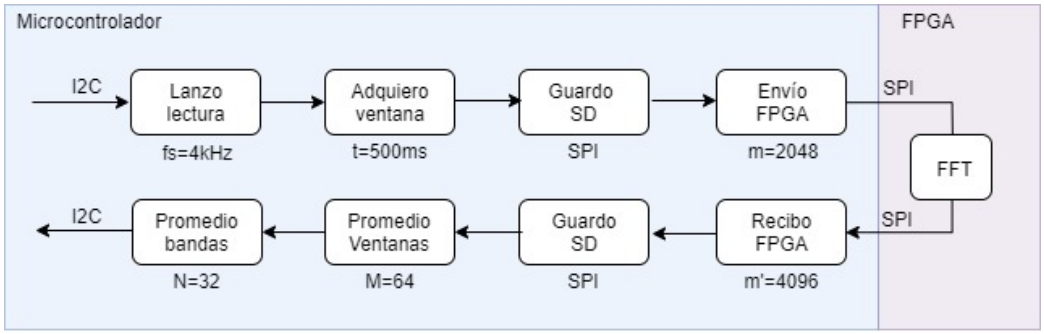


Figura 5.7: Diagrama de bloques Módulo Adquisición Señales Vectoriales

Para I2C se utilizó la librería implementada en Arduino para el protocolo al igual que el MASE.

Cuando se recibe un paquete correcto en la interrupción, se levanta un flag de paquete recibido. Se atiende esa solicitud en el loop principal, y se procesa dicho paquete de acuerdo al protocolo. En el diagrama se menciona como **Procesar Paquete**.

En el caso de que haya una solicitud de lectura nueva en un canal dado, se levanta el respectivo flag. Luego se atiende dicho flag en el loop principal y se entra en el bloque de **Adquirir Lectura**. Este bloque es un bucle que se itera *Cantidad de ventanas* veces y tiene los siguientes estados:

- Adquirir ventana: adquiere 2048 muestras de audio a 4kHz.
- Guardo lecturas en la microSD.
- Envío lecturas hacia la FPGA.
- Recibo coeficientes desde FPGA.
- Guardo coeficientes en la microSD.
- Calculo el módulo y promedio con ventanas anteriores.

Finalmente, al salir del bucle se ejecuta un nuevo promedio de los módulos comprimiendo en 32 bandas la FFT. Dichas bandas se almacenan en un *buffer* hasta que el maestro solicita la lectura.

Cuando el módulo no está activo, esto es cuando no recibió un paquete válido ni tiene ninguna adquisición pendiente, va a modo de **Bajo consumo**

En el diagrama de la Figura 5.8 se muestra el Round-Robin implementado.

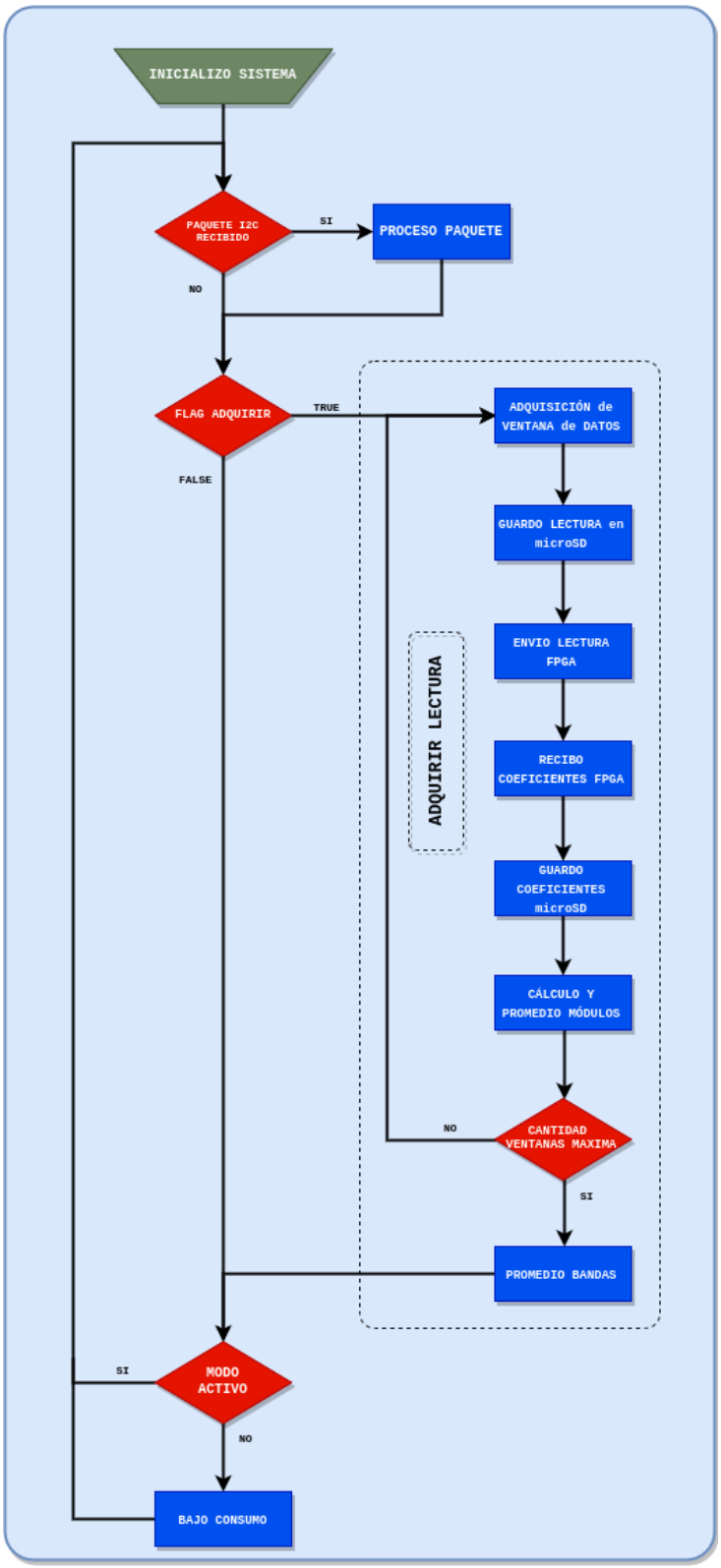


Figura 5.8: Diagrama del Round-Robin del módulo de adquisición de señales vectoriales.

5.3.2. FPGA Cyclone10

El microcontrolador adquiere las señales de audio a una tasa de muestreo de 4kHz, en ventanas de 500ms, obteniéndose 2048 muestras para procesar que ingresan a la FPGA. La razón de que la comunicación entre el microcontrolador y la FPGA sea SPI, es que un integrante de SARA trabajó sobre esta transmisión de datos en el proyecto del curso de *Diseño lógico II*.

El procesamiento de audio consiste en realizar la Transformada Rápida de Fourier, para estudiar los componentes en frecuencia de las señales de interés. Por mayor información acerca de la FFT acudir al apéndice D.

Para el procesamiento de las muestras se eligió trabajar con *cores* abiertos, obtenidos desde *OpenCores*. Estos son bloques de códigos abiertos con funcionalidades específicas disponibles para la programación de las FPGAs. Se decidió implementar un canal de cálculo de FFT, lo que permite el procesamiento de un canal a la vez. Los cores abiertos utilizados son:

- SPI: "SPI Master/Slave Interface" [12], código en VHDL, core esclavo que responde al SPI maestro implementado en el microcontrolador. Se eligió utilizar el modo 0, que corresponde a CPOL=0, CPHA=0. *CPOL, Clock polarization*, y *CPHA, Clock phase*.

- FFT: "Parametrized FFT engine" [46], trabaja en "pipeline mode", esto significa que a medida que ingresan las muestras al core FFT, luego de un tiempo de latencia determinado, el core va devolviendo las muestras procesadas por su salida.

Los pines que interconectan al microcontrolador con la FPGA son:

* clk: Reloj del sistema.

* reset: Reset del sistema.

* spi_MOSI: Master output Slave input, pin que recibe los datos seriales enviados por el Master.

* spi_MISO: Master input, Slave output, pin a través del cual se envían los datos seriales hacia el Master.

* spi_ssel: Slave select, activo por nivel bajo, el habilita el esclavo a quien quiere dirigir los datos en un sistema multi Slave.

* spi_clk: Reloj serial SPI.

La asignación de estos pines en la placa se presentan en la tabla 5.2.

Nombre	I/O	Cyclone10	Puerto SAMD	#define SAMD
clk	entrada	PIN_E2	PA27	30
reset	entrada	PIN_G1	PA22	0
spi_miso	salida	PIN_C16	PA19	10
spi_mosi	entrada	PIN_F16	PA16	8
spi_sck	entrada	PIN_F15	PA17	9
spi_ss	entrada	PIN_G15	PA21	7

Tabla 5.2: Pines interconexión microcontrolador - FPGA

Estructura del sistema diseñado

El diseño en la FPGA se conforma por los bloques que se mencionan a continuación. Se puede observar la estructura de los bloques diseñados en la figura 5.9.

- SPI slave
- Intérprete de comandos
- FFT driver lectura
- FFT módulo
- FFT driver escritura

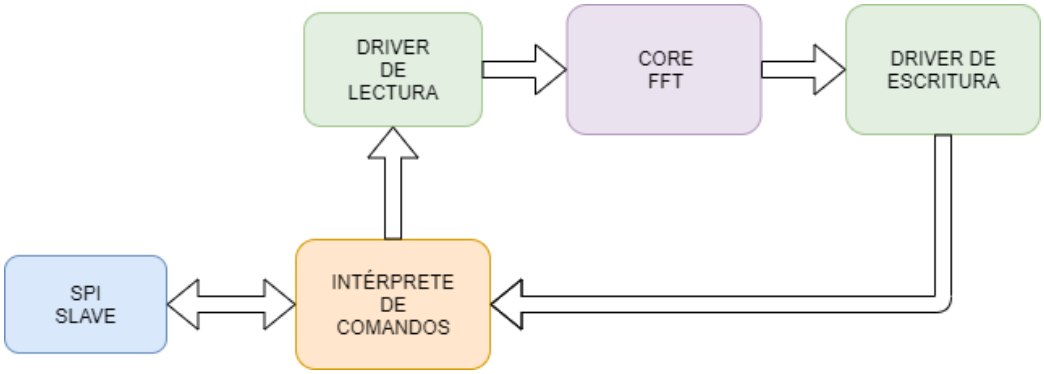


Figura 5.9: Diagrama de bloques del código en VHDL

El microcontrolador se comunica con la FPGA a través del bloque *SPI_slave*, éste se comunica a su vez de forma bidireccional con el bloque *Interprete de Comandos*. Este último bloque es el encargado de recibir e interpretar comandos, y desencadenar 3 posibles acciones: la escritura de muestras que serán enviadas al core FFT, la lectura de las muestras ya procesadas, o el envío de un byte que brinda información acerca del estado en el que se encuentra la FPGA. La transmisión de SPI es en paquetes de a 8 bits; las muestras adquiridas de 16 bits, por lo tanto, previo a ingresar al core FFT, los bytes recibidos son concatenadas por el Driver

Acción	Byte de comando
Escritura	0xA0
Lectura	0xAA
Status	0x55

Tabla 5.3: Comandos del protocolo de comunicación microcontrolador - FPGA

de Escritura, e ingresan al core las muestras de 16 bits. Lo mismo ocurre una vez finalizado el procesamiento de FFT. El Driver de Lectura recibe las muestras procesadas de 16 bits, y las divide en 2 bytes para poder ser transmitidas por SPI hacia el microcontrolador.

El diseño es paramétrico en la cantidad de muestras a procesar. Se definieron las siguientes especificaciones:

* 1 canal.

* Número de muestras 2048.

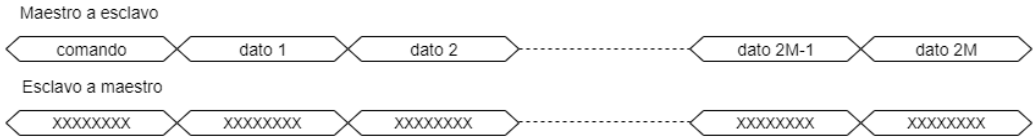
* Profundidad de muestras 16 bits.

Protocolo de comunicación Micro-FPGA

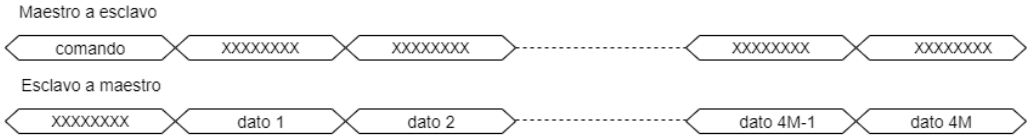
El microcontrolador y la FPGA se comunican bajo un protocolo implementado en base a comandos de escritura, lectura, y consulta del estado de la FPGA. El maestro envía estos comandos a la FPGA, y ésta los interpreta para tomar las distintas acciones. En la tabla 5.3 se observan los tres comandos posibles. En la figura 5.10 se observa el protocolo de comandos que manejan el maestro (microcontrolador) y el esclavo (FPGA).

Byte status	Significado
Bit 0	En 1: sistema pronto para enviar muestras procesadas
Bit 1	En 1: sistema disponible para recibir nuevas muestras
bit2-bit7	Sin uso, valor 0 por defecto

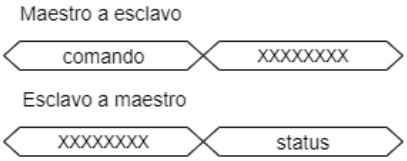
Tabla 5.4: Significado del byte de status



(a) Comando de escritura Master - Slave



(b) Comando de lectura Master - Slave



(c) Comando de status Master - Slave

Figura 5.10: Protocolo de comunicación microcontrolador - FFT

Cuando el maestro envía un comando de status, el sistema responde en el byte siguiente. En la tabla 5.4 se detalla el significado de cada bit del byte de status.

En el ciclo de escritura (muestras nuevas a la FPGA), el sistema queda a la espera de recibir 2*M bytes a ser procesados, o lo que es lo mismo M muestras, ya que cada muestra ocupa 2 bytes. En el ciclo de lectura (muestras procesadas), el maestro envía el comando de lectura, y se queda la espera de que el esclavo le envíe 4*M bytes, ya que la FFT trabaja con muestras complejas, cada muestra con su parte real (16 bits) y parte imaginaria (16 bits).

El orden de los datos definido es el siguiente: para el ciclo de escritura, deben enviarse las muestras en orden creciente. Deben enviarse el byte menos significativo y el mas significativo de una misma muestra de manera consecutiva, comenzando por el byte menos significativo. Para el ciclo de lectura, el orden es similar, se reciben las muestras en orden creciente. Se recibe primero el byte menos significativo de la parte real, luego el más significativo de la parte real, luego el byte menos significativo de la parte imaginaria y por último el byte más significativo

de la parte imaginaria. La estructura descrita se recibe de manera consecutiva. Vale la pena aclarar qué, si el sistema recibiera un comando inválido, distinto al de escritura, lectura, o status, el sistema lo ignora, y se queda a la espera de un comando válido. En la figura 5.11 se detalla lo anteriormente descrito.

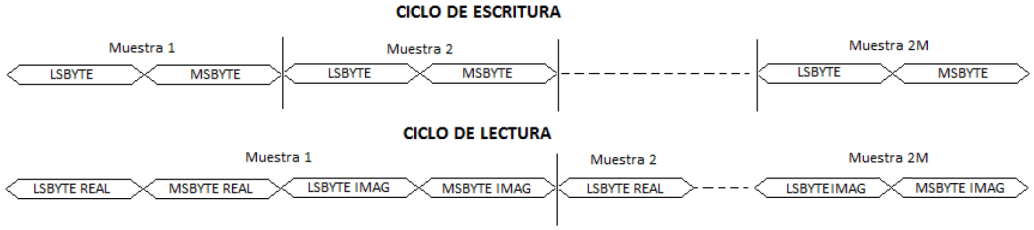


Figura 5.11: Orden de bytes para ciclos de lectura y escritura

Se plantea a continuación un breve desarrollo de los bloques implementados en el diseño.

SPI slave

Este bloque es un core implementado como esclavo de SPI, es el único que tiene entradas y salidas al microcontrolador. Estas E/S se resumen en cuatro señales que definen la comunicación SPI: el CLK, el selector de esclavo, y los canales de transmisión (MISO) y recepción (MOSI) desde el maestro.

- MISO: Master Input, Slave Output
- MOSI: Master Output, Slave Input
- SCK: CLK de la comunicación, en cada flanco se lee o escribe un bit
- SS: Slave Select, nivel bajo indica al esclavo que se habilita la comunicación

Este bloque cuenta además con las siguientes señales:

- clk_i: clk interno del FPGA
- di_i[N-1..0]: entrada de datos, a ser enviados al maestro.
- wren_i: señal que el sistema utiliza para indicar al bloque SPI que hay datos válidos para ser transmitidos. Se mantiene en alto por dos flancos de reloj, cuando se ingresa un nuevo byte en la entrada di_i.
- di_req_o: salida del bloque SPI. Cuando el esclavo transmite al maestro, antes del envío del último bit, solicita al sistema mediante esta señal, que se le ingrese un nuevo byte (en di_i) para ser transmitido.
- wr_ack_o: reconocimiento al sistema de que el bloque SPI recibió el próximo byte a ser transmitido.

- do_valid_o: salida del bloque SPI. Indica al sistema que hay datos válidos recibidos desde el maestro; ésta señal se levanta por dos flancos, luego del tercer flanco transcurrido que se recibió el byte.
- do_o[N-1..0]: salida de datos recibidos desde el maestro.

En resumen, el bloque SPI es el encargado de recibir y transmitir bytes que viajan por MOSI y MISO respectivamente. SPI envía de a 1 bit, cada 8 bits recibidos por el FPGA, el bloque SPI escribe en su salida *do_o* el byte recibido, y levanta la señal *do_valid* para informarle al sistema, que hay un dato nuevo disponible. La transmisión de datos hacia el maestro - microcontrolador, se realiza de forma similar. El sistema escribe en la entrada *di_i* del bloque SPI el byte a ser transmitido, y levanta la entrada *wren_i* para informarle al bloque SPI que el dato en *di_i* es válido; luego el bloque descompone este byte recibido y lo envía bit a bit por la señal MISO. El diagrama de tiempos de este bloque lo observamos en 5.12.

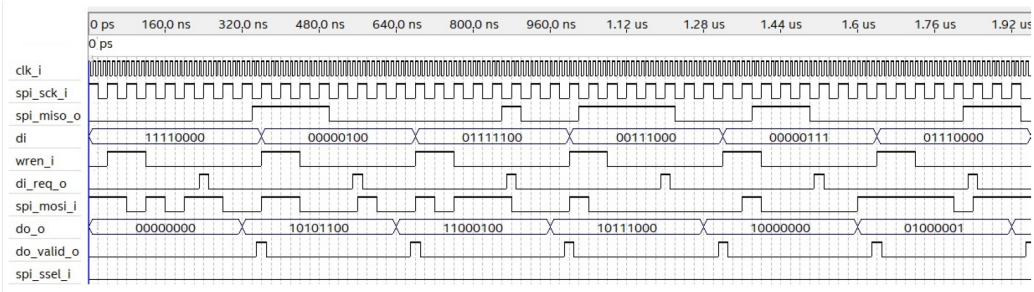


Figura 5.12: Diagrama de tiempos del core SPI - simulación desde Quartus

Intérprete de comandos

Este bloque es el nexo entre la comunicación con el maestro, y el procesamiento propio de la FFT. Se comunica con el bloque SPI, el driver de lectura, y el driver de escritura. Este bloque es una gran máquina de estados, formada por sub-máquinas de estado. En la figura 5.13 se puede observar la máquina de estados principal.

FFT driver escritura

Las palabras recibidas por SPI son de largo N_spi (8 bits en particular), y el IP Core de FFT recibe las muestras de tamaño bit_depth (se definió 16 bits). Por lo tanto una de las funciones que cumple el driver de escritura es, a medida que se reciben 2 bytes consecutivos, los concatena y almacena en memoria DPRAM las muestras. El driver sensa el estado del core FFT a través de la señal *busy*, y cuando se encuentra disponible, el driver le indica al core a través de la señal *start* que puede leer las muestras a ser procesadas. El Driver escribe las muestras en direcciones de memoria que deberá indicarle al Core FFT de dónde leerlas. Las direcciones las indica en su salida *address_in*.

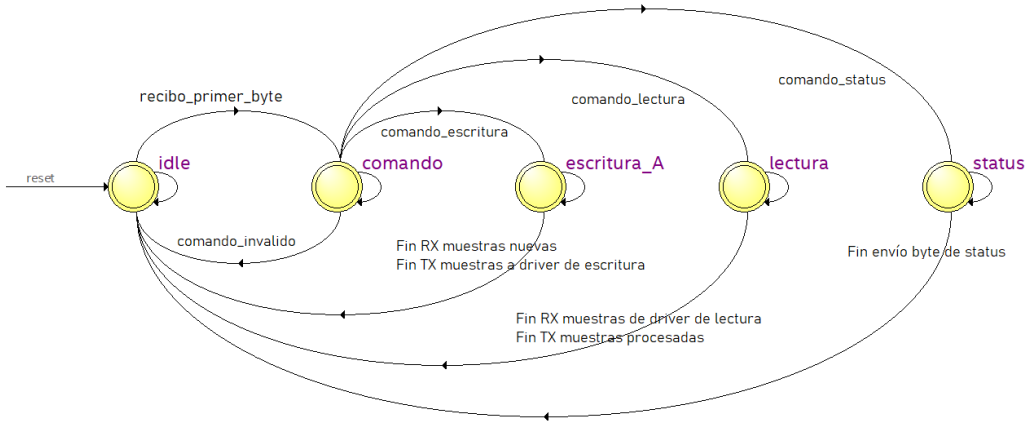


Figura 5.13: Diagrama de estados del bloque Intérprete de comandos

FFT

El Core FFT elegido es parametrizable en el número de muestras a procesar (potencia de 2), y en el formato de datos que ingresan. Este core puede ser implementado de dos maneras, haciendo uso de un solo bloque Butterfly, o utilizando múltiples bloques. Éste último es considerablemente más rápido, a costas de un alto consumo de recursos. Se desarrolla este punto en el apéndice D. Se decidió la implementación de menor consumo de memoria, utilizando un solo bloque butterfly. El tiempo de procesamiento está definido por la latencia del bloque Butterfly, y está definido como una constante BTFLY_LATENCY del archivo fft_engine.vhd. El bloque funciona en un modo llamado "pipeline mode", que significa que en cada flanco de reloj es capaz de leer una muestra; y luego del tiempo de latencia, puede escribir en cada flanco una muestra procesada. Al utilizar un solo bloque de cálculo de Butterfly, se utilizan dos DPRAM para acelerar el procesamiento. DPRAM: Dual-ported RAM; es un tipo de acceso de memoria que permite escritura y lectura de forma simultánea. Lo que permite que el Core FFT comience a leer las muestras a procesar de las direcciones de memoria, mientras el driver de escritura aún sigue escribiendo nuevas muestras; ídem con el driver de lectura, se pueden ir leyendo las muestras procesadas mientras el Core aún escribe el total de las muestras.

FFT driver lectura

De forma análoga que el driver de escritura, este driver de lectura se encarga de segmentar las muestras de tamaño bit_depth que el IP Core devuelve. El driver tiene una salida hacia el core, donde indica las direcciones de memoria donde almacenar las muestras procesadas. El driver espera a que el IP Core le indique a través de las señal *ready* que puede leer las muestras de dichos lugares de memoria. Luego, este driver espera por la solicitud del Intérprete de Comandos, para enviarle las muestras procesadas. A través de una salida *data_rdy* indicá que hay un dato válido en su salida *data_p*. Una entrada *read* indica al driver que el Intérprete ya

leyó el byte anterior, y está disponible para recibir uno nuevo.

Simulación del sistema diseñado

Para constatar que el sistema diseñado esté efectivamente trabajando según lo deseado, se realizaron simulaciones de todos los bloques de forma independiente, utilizando las herramientas de Quartus. Se realizó finalmente una simulación del sistema completo, en la imagen 5.14 se muestra un ciclo de funcionamiento. Este ciclo se compone por una consulta de estado, una solicitud de escritura, la transmisión de las muestras, nuevamente una consulta de estado para sensar si están prontas las muestras, y por último la solicitud de lectura de las muestras procesadas.

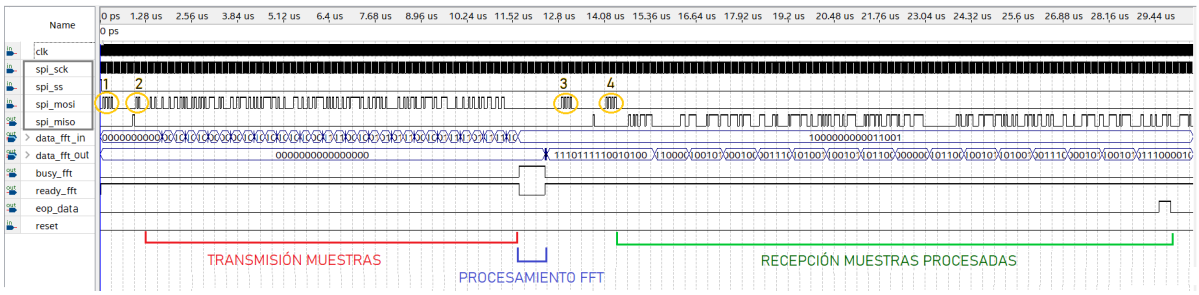


Figura 5.14: Simulación del código de la FPGA en Quartus para 16 muestras

En la simulación de la imagen 5.14 se presentan las 4 entradas y salidas principales, recuadradas en gris: *spi_sck*, *spi_ss*, *spi_mosi*, y *spi_miso*. Estas señales, junto con el *reset* y el *clk* son las que se conectan al microcontrolador. El resto de las señales se despliegan para mostrar el funcionamiento del sistema. Estas señales son:

- data_fft_in: data que ingresa al core FFT
- data_fft_out: data que sale del core FFT
- busy_fft: core FFT ocupado
- ready_fft: core FFT listo
- eop_data: señal que indica el fin de las muestras procesadas

El reloj del sistema se setea en 20ns, y el reloj de la transmisión de spi en 40ns. Se toma una ventana para simular de $35\mu s$. En *spi_mosi* se ingresan los datos de entrada, los datos que mandaría el microcontrolador. En *spi_miso* se muestran los datos de salida del FPGA, datos que recibiría el microcontrolador. La señal *spi_ss* es activa por nivel bajo, y es la señal que habilita al esclavo, por lo cual se la mantiene en 0. Las señales *busy_fft* y *ready_fft* indican el estado del core FFT, busy_fft se levanta cuando el core procesa las muestras. Las señales *data_fft_in* y

data_fft_out indican las muestras de 16 bits de entrada y de salida del Core FFT respectivamente.

Se realiza la simulación para 16 muestras para poder mostrar el ciclo entero. 16 muestras implican 32 bytes de transmisión, y 64 bytes de recepción de las muestras procesadas, ya que devuelve la parte real y la parte imaginaria de cada muestra. Se envía una secuencia de 4 comandos que se enumeran en la imagen:

1. Status: 0x55

2. Escritura 0xA0

3. Status 0x55

4. Lectura 0xAA

El ciclo comienza con una solicitud acerca del estado del sistema (comando Status). Cuando la FPGA se encuentra disponible para recibir muestras, devuelve 0b00000010. El maestro envía un comando de escritura, y a continuación las 16 muestras (32 bytes) a ser procesados. Cuando el core FFT levanta la señal de *busy_fft* significa que se comenzó el procesamiento, cuando finaliza sube *ready_fft*, y baja $busy_fft$. Se envía a continuación un comando de status, el sistema devuelve 0b00000001, indicando que tiene muestras procesadas prontas para ser enviadas. Para finalizar el ciclo, se envía un comando de lectura, y el sistema escribe en *spi_miso* los 64 bytes correspondientes a las 16 muestras procesadas.

Se menciona como conclusiones de diseño que el tiempo de procesamiento de la FFT es considerablemente más chico que el tiempo de transmisión de datos por SPI. El tiempo de procesamiento para 16 muestras es de $0,72\mu$ segundos, un 6 % del tiempo de transmisión de muestras nuevas, y un 3 % del tiempo de transmisión de las muestras procesadas.

Para verificar que el circuito construido opere de forma correcta en el Hardware en el que se va a implementar, se realizó un chequeo de la frecuencia de funcionamiento. Esto se hace a través de la función de Quartus "Timing Analyzer",que indica la frecuencia máxima de operación. Esto se observa en la tabla 5.5. El reloj de la FPGA es inyectado por el microcontrolador a través del *PIN_E2*, con su misma frecuencia de funcionamiento: 49MHz. El reloj de la comunicación de SPI, lo setea el maestro al establecer el canal de SPI a través del pin *SCK*, frecuencia que se definió en 4MHz. Ambas frecuencias cumplen con estas restricciones. Se puede asegurar que el circuito diseñado funcione correctamente en la FPGA Cyclone 10.

Reloj	Frecuencia máxima
clk_{FPGA}	149.97 MHz
clk_{SPI}	197,78 MHz

Tabla 5.5: Frecuencias máximas de funcionamiento en sistema VHDL

5.4. Diseño de Hardware

5.4.1. Diseño de placas adquisidoras

Elección de los amplificadores

Para la elección de los amplificadores se debe tener en cuenta, en primer lugar, que funcionen correctamente con la alimentación que especifica el proyecto: de 0 a 5V. Se estudiaron los amplificadores operacionales más utilizados hoy en día. Se eligieron los amplificadores LM324 de Texas Instruments, los cuales a pesar de tener muchos años en el mundo de la electrónica continúan siendo útiles en múltiples aplicaciones, son además de bajo costo, y se encuentran accesible en plaza. Este circuito integrado implementa cuatro amplificadores operacionales, y cumple con las especificaciones de diseño. Sus principales características son:

- Rango de voltaje de alimentación simple: 3VDC - 30VDC
- Ancho de banda: 1MHz
- Ganancia de voltaje DC: 100dB
- Baja corriente de polarización de entrada: 45nADC
- Baja tensión de offset de entrada: 2mVDC. I_{offset}: 5nADC
- Rango voltaje de salida: 0VDC - VCC
- Consumo de corriente: 700uA

Punto de operación

El punto de operación del circuito se fija teniendo en cuenta el voltaje de referencia que tiene el ADC de la Arduino MKR Vidor 4000. El voltaje de operación del microcontrolador de la placa en cuestión es 3.3V, por lo tanto se fija el punto de operación en 1.5V, y así se obtiene la máxima excursión posible:

Este voltaje de operación se lleva a cabo mediante un divisor resistivo, con salida al borne no inversor de los amplificadores de las etapas de ganancia. Se observará en las figuras de los circuitos a continuación. Las resistencias diseñadas para este punto son: $R_{po1} = 15k\Omega$ y $R_{po2} = 6{,}8k\Omega$, y el voltaje de alimentación es de 5V. Se define el punto de operación:

$$V_{OP} = \frac{R_{po2}}{R_{po1}+R_{po2}}.VCC = 1{,}56V$$

Etapa amplificadora

La etapa de amplificación consta de dos etapas amplificadoras, una etapa de ganancia fija 10V/V, y la otra etapa de ganancia variable. Para esto se implementó la ganancia variable mediante un Dip switch como el que se observa en la figura 5.15. Definiendo una ganancia total del circuito de 82V/V, 100V/V, 150V/V, y 200V/V. Las ganancias se eligieron según el estudio en abejas realizado por el proyecto que antecede, PestiBee. El circuito que se implementó para esta etapa de amplificación con una etapa de ganancia fija, y otra variable, se observa en la figura 5.16.

Figura 5.15: Dip switch para modificar ganancias variables

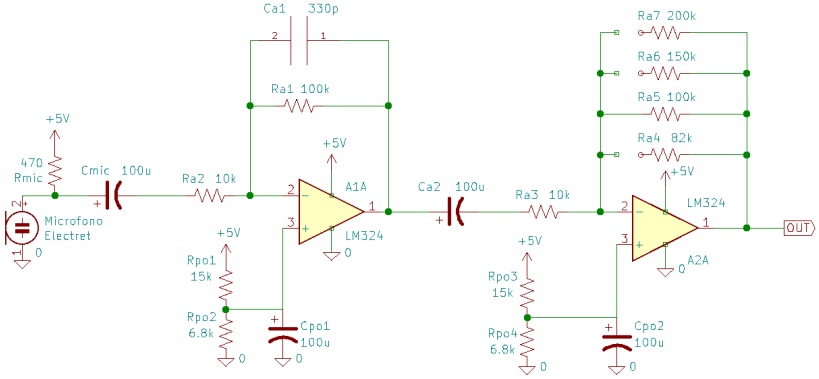


Figura 5.16: Etapa de amplificación, ganancia fija, y variable

Ambas etapas de amplificación se realizaron mediante el operacional LM324 conectado como inversor, la ganancia quedó determinada por: $G = -R_{a1}/R_{a2}$, siendo Ra1 la resistencia en la re-alimentación, y Ra2 la resistencia en la entrada inversora.

En la etapa de amplificación con ganancia fija, se agregó un capacitor, en paralelo a la resistencia de la realimentación del amplificador, definiendo un filtro pasabajos de primer orden con frecuencia de corte 1500Hz. Esto se realizó para mejorar la calidad del filtro Butterworth que se implementa luego de la etapa de amplificación. La implementación del capacitor, sumada al efecto del filtro de segundo orden en la etapa siguiente, implica un filtro con una caída final de -60dB, en la frecuencia deseada 1500Hz.

Etapa de filtrado

Se mencionó al comienzo de este capítulo que las frecuencias de las señales de interés son hasta 700Hz dado que las abejas en cuestión no suelen emitir sonidos en frecuencias superiores, en condiciones normales. Para evitar perdida de información, se decide filtrar en 1500Hz. Se estudiaron cuatro filtros posibles:

- Butterworth
- Tchebychev
- Bessel
- Linkwitz-Riley

El filtro Butterworth optimiza la respuesta plana en la banda pasante, y tiene una caída de 20n dB por década, siendo n el orden del filtro. Tchebychev tiene la caída más brusca de todos, pero tiene un pico cerca de la frecuencia de corte, por lo que se descarta. El filtro de Bessel no cae tan rápido, pero es el que brinda menor errores de fase; esto último no genera una gran ventaja. Por último, el filtro de Linkwitz-Riley tiene el punto de corte en -6dB, y se construye a partir de dos filtros Butterworth. Si bien este último filtro se utiliza generalmente en aplicaciones de audio, se consideró que con un filtro Butterworth ya es adecuado para la aplicación que presenta SARA-Bee.

Se eligió el filtro pasabajos Butterworth, ya que genera la respuesta más plana posible en la banda pasante. La salida se mantiene constante hasta los 1500Hz, luego cae a razón de 40dB por década, por ser filtro de segundo orden. En el apéndice C se desarrolla los cálculos que justifican que el diseño realizado es efectivamente un filtro Butterworth de segundo orden. El circuito diseñado se puede observar en la figura 5.17.

El diseño de esta etapa fue iterativo, y se describe a continuación. La frecuencia de corte es definida por: $w_c = \frac{\sqrt{2}}{RC} \Rightarrow f_c = \frac{\sqrt{2}}{2\pi RC}$ A priori, se definen las resistencias $R_{f1} = R_{f2} = 10k\Omega$, y los capacitores $C_{f2} = \frac{C_{f1}}{2} = 15nF$. En plaza se consiguen estos valores, salvo por C_{f2}, el valor próximo es 8.2 nF. Estos valores definen una frecuencia de corte de $f_c = 1500{,}527Hz$. Sin embargo, como se mencionó, en la etapa amplificadora se agregó un filtro de primer orden con el fin de generar un filtro de mayor calidad, con una caída total de 60dB por década. Este filtro de primer orden genera que la frecuencia de corte total del sistema quede por debajo

de los 1500 Hz. Se resolvió aumentar la frecuencia de corte del filtro Butterworth para que la frecuencia de corte final sea la deseada. Se decide iterar los parámetros del circuito diseñado, y se modifica los valores de las resistencias a: $R_{f1} = R_{f2} = 8{,}2k\Omega$. Este cambio determina una frecuencia de corte para el filtro de primer orden de $f_c = 1875{,}659Hz$, obteniendo una frecuencia de corte final de $1500Hz$.

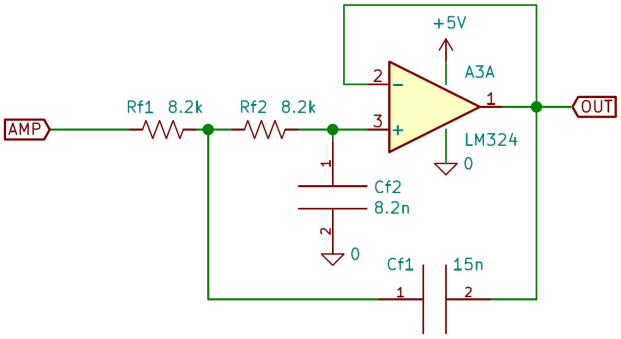


Figura 5.17: Etapa de filtrado, Butterworth de segundo orden

Circuito final

El circuito de acondicionamiento resultante de las señales analógicas se muestra en la figura 5.18. Consta de dos etapas de amplificación y una etapa de filtrado.

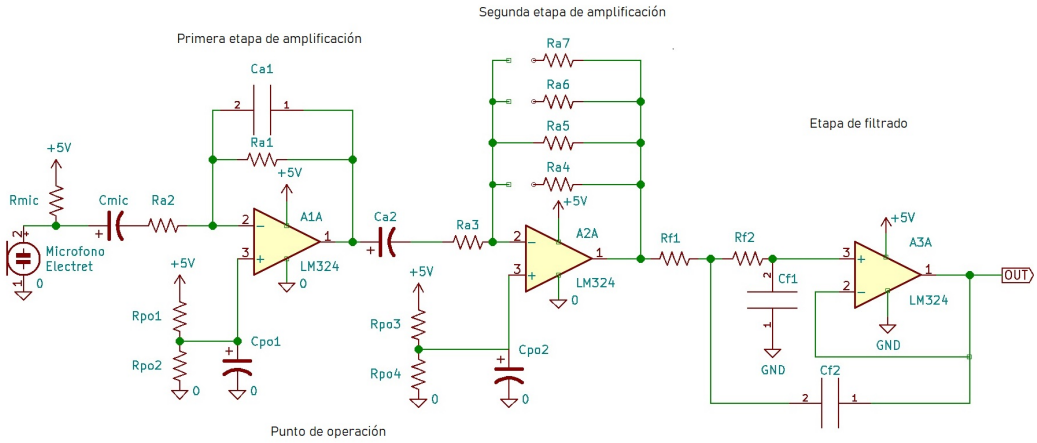


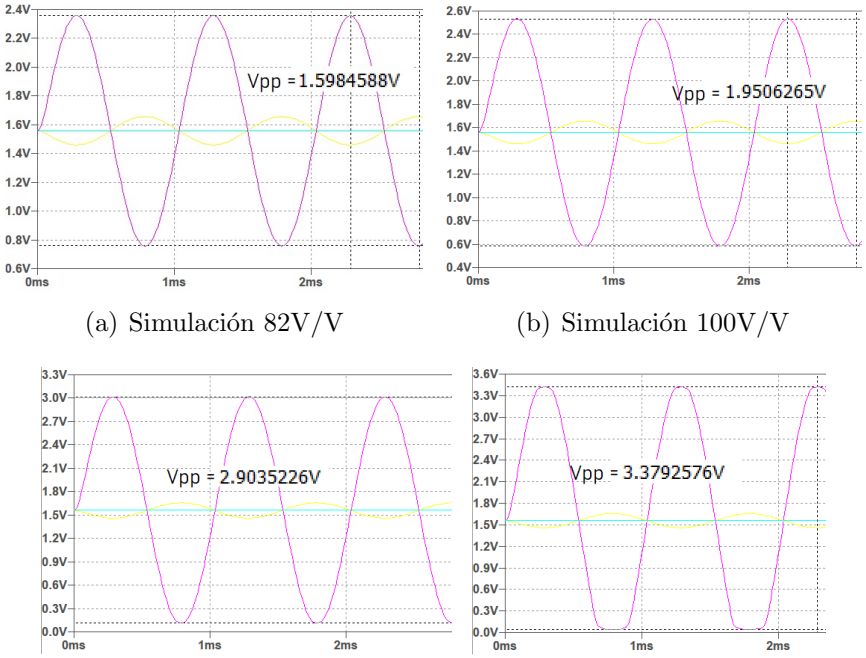
Figura 5.18: Circuito final de acondicionamiento de señales vectoriales

Simulaciones

En primer lugar se verifica el funcionamiento de la etapa de amplificación, de ganancia fija 10V/V. Las ganancias variables se modifican mediante un DIP Switch. Se simula con una señal sinusoidal de entrada de 10mV pico. En las gráficas de la figura 5.19 se observan las señales de salida: en amarillo se observa la

amplificación con ganancia 10V/V, en rosado la ganancia variable, y en celeste se observa el punto de operación.

Se obtuvieron los resultados de la simulación a través del programa LTSpice XVII. Se releva la primera etapa de amplificación de 10 V/V fija, y se obtiene una salida de 194,4 mV, lo que indica una ganancia de $G_{fija} = 9{,}7V$. Para las ganancias variables se relevan los valores de la tabla 5.6. Se menciona que el amplificador x200 implica una salida saturada para la señal de prueba de amplitud pico 10mV, y puede observarse en la figura 5.19(d), por ello la ganancia simulada no se asemeja al valor deseado.



(a) Simulación 82V/V (b) Simulación 100V/V

(c) Simulación 150V/V (d) Simulación 200 V/V

Figura 5.19: Simulación de etapa amplificadora

$G_{ideal} V/V$	$G_{simulada} V/V$
82	79,86
100	97,52
150	145,23
200	168,97

Tabla 5.6: Simulación etapa amplificadora

Se simula el circuito estudiando su respuesta en frecuencia, y se obtiene un buen comportamiento según el diseño, con una frecuencia de corte de 1556.4Hz. Se observa en la figura 5.20 el módulo y fase de la respuesta en frecuencia.

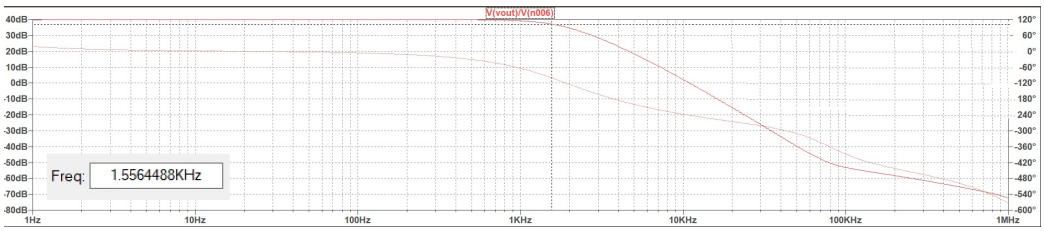


Figura 5.20: Simulación respuesta en frecuencia

5.4.2. Diseño de PCB

Se utilizó el software KiCad para la creación de los archivos necesarios para la impresión del hardware descrito en este capítulo. El PCB principal en este módulo es el acondicionamiento de las señales de audio. Además de esta placa, se desarrollaron Shields; uno para adaptar las 4 placas de acondicionamiento de audio; y otro Shield donde se realizan las conexiones para la placa Arduino y el módulo micro SD, y conversor de nivel de voltaje. Las placas se envían a China a ser fabricadas, a través de *PCBWay*; y todos los componentes fueron soldados y ensamblados por los autores del proyecto.

Acondicionamiento de señales de audio

Esta placa implementa el circuito diseñado de amplificación y filtrado. Los componentes son SMD, y el DIP Switch corresponde a la ganancia variable, con valores ya mencionados. El diseño se realiza con 4 capas de cobre, esto le brindó mayor complejidad, pero permitió realizar el diseño en un tamaño muy reducido: 3.5xmx3.5cm. Se observa en la figura 5.21 el diseño del PCB, y su modelo 3D. Las capas externas se utilizaron para las pistas, en la figura se ven en rojo las pistas de la capa superior, y en amarillo las pistas de la capa inferior. Las 2 capas internas de cobre se utilizan como planos de Vcc y de tierra.

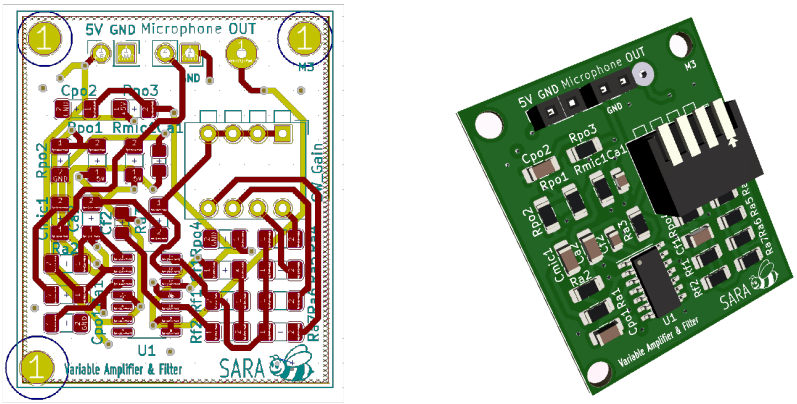


Figura 5.21: PCB y modelo 3D para el acondicionamiento de señales de audio

Shield placas de audio

Este Shield cumple la función de agrupar los 4 módulos de acondicionamiento de las señales de audio, para los 4 canales posibles. Tiene la entrada de alimentación 5V, y GND; las 4 entradas de los micrófonos, y las 4 salidas del audio acondicionado hacia la Arduino. Se observa en la figura 5.22 el diseño del PCB, y su modelo 3D.

Existieron dos instancias de fabricación de PCBs, en la primera fabricación, la comunicación I2C entre el MASV y el módulo principal no había sido aún contemplada, y por lo tanto no se había considerado el conversor de nivel de voltaje bidireccional. Los diseñados que se presentaron anteriormente, son los diseños finales de este Shield.

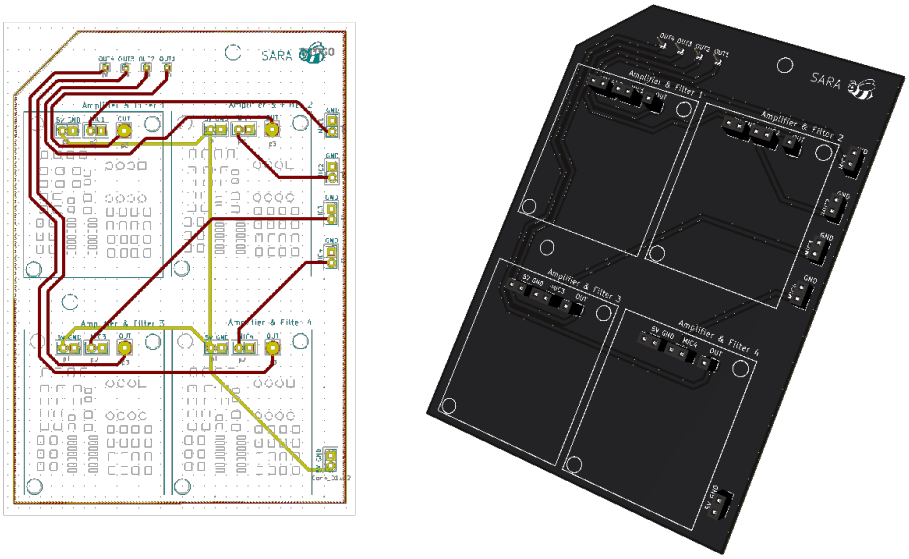


Figura 5.22: PCB y modelo 3D del Shield para los 4 módulos de audio

Shield Arduino MKR Vidor 4000 más componentes

Esta Shield cumple la función de tener un módulo más limpio de cables, y de agrupar a la placa principal los componentes que utiliza: módulo micro SD y conversor de nivel de voltaje bidireccional para la comunicación con I2C. Se observa en las figuras 5.23 el diseño del PCB, y su modelo 3D.

Se soldaron los componentes, y se ensamblaron las placas. Las placas resultantes se pueden observar en la figura 5.24.

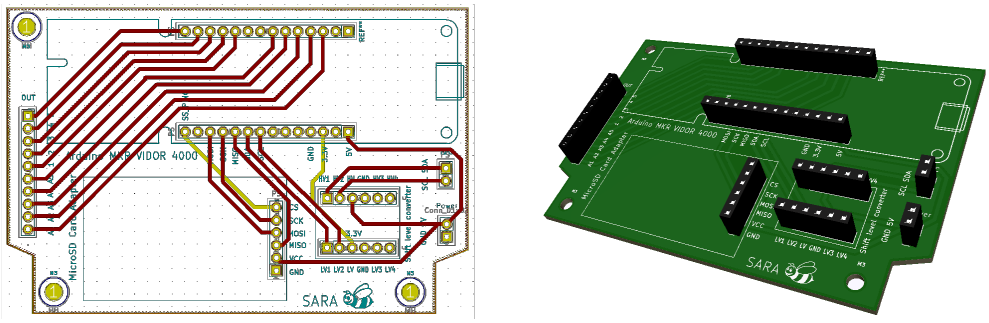
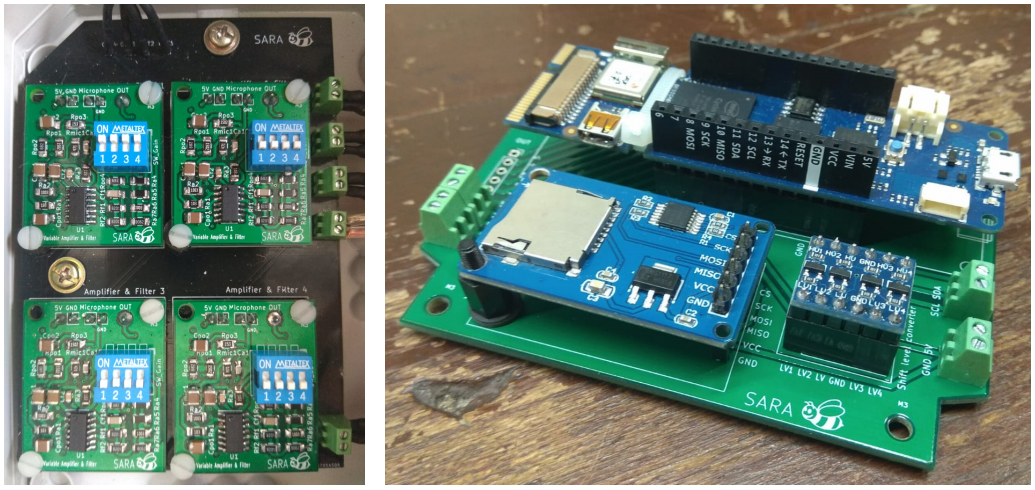


Figura 5.23: PCB y modelo 3D del Shield de Arduino MKR Vidor 4000 más componentes



(a) Shield más placas acondicionamiento

(b) Shield Arduino más componentes

Figura 5.24: MASV - Placas fabricadas

5.5. Pruebas y Validación de módulo

5.5.1. Pruebas Hardware

Luego de realizado el diseño del hardware, se prueba el circuito en protoboard con componentes obtenidos en plaza, y se verifica un comportamiento deseado. Luego de la fabricación de placas PCB en el exterior, se ensamblan los componentes, se prueba el nuevo hardware y se verifica nuevamente que cumpla las especificaciones de diseño. Se muestran los resultados obtenidos con las placas finales. Para estas pruebas, se solicitó a docentes del IIE, el préstamo de la placa Analog Discovery 2 de Digilent, que incluye osciloscopio, generador de ondas y fuente. Se utiliza el software *digilent.waveform* en la PC para la observar y registrar los resultados obtenidos.

Las pruebas fueron realizados con una entrada de las siguientes especificaciones:

- Sinusoidal
- Amplitud pico 10mV
- Offset 0V
- Frecuencia 1kHz

En el canal 1, color azul o número II de referencia en la imagen, se va a observar la entrada; en el canal 2 la salida, color rojo o número I. Las escalas del osciloscopio se pueden ver en la tabla 5.7.

	Escala
Voltaje CH1	50 mV/div
Voltaje CH2	1 V/div
Temporal	5 ms/div

Tabla 5.7: Escalas del osciloscopio para pruebas en las placas finales

Amplificación

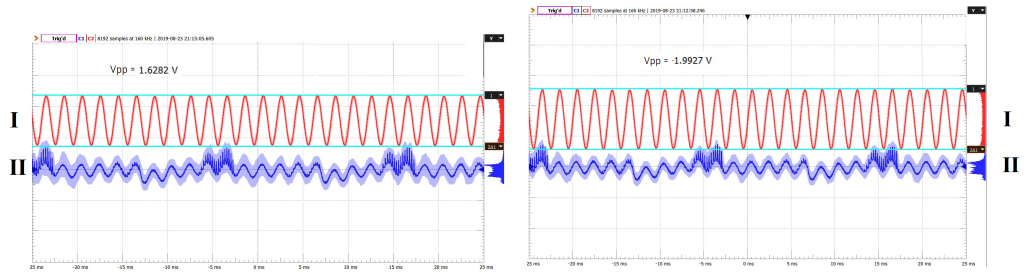
Se prueba la etapa amplificadora para las 4 ganancias, ajustando el DIP Switch. Se registra la salida para los 4 amplificadores posibles: 82v/v, 100v/v, 150v/, y 200v/v. En la figura 5.25 se ve lo obtenido para cada ganancia.

Para constatar que efectivamente funcione según lo deseado, se realiza un estudio comparativo de los valores teóricos, simulados, y real. Se muestran estos resultados en la tabla 5.8.

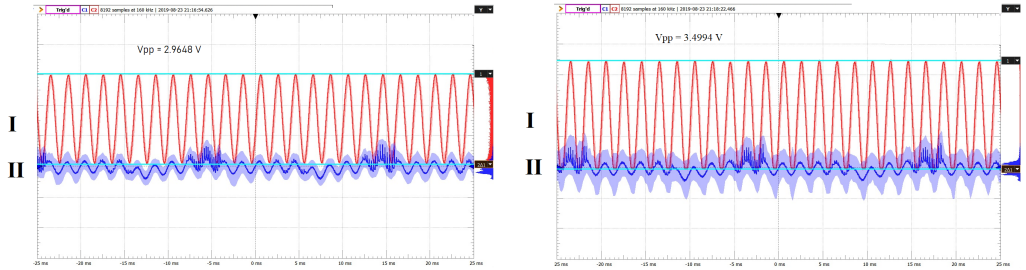
$G_{ideal}(V/V)$	$G_{simulada}(V/V)$	$G_{real}(V/V)$	$\Delta G_{real}(V/V)$
82	79,86	81,41	0,59
100	97,52	99,64	0,36
150	145,23	148,24	1,76
200	168,97	174,97	NC

Tabla 5.8: Resultados etapa amplificadora

Lo primero a mencionar es qué, con una entrada de 10mV de pico, el amplificador x200 implica una salida saturada, ya que el punto de operación es de 1.5V. Por lo tanto, este no es un resultado que permita sacar conclusiones. Se observa que la ganancia simulada es más lejana que la ganancia obtenida en el circuito construido; esto se puede deber a qué las librerías que se utilizan para la simulación del amplificador LM324 se construye teniendo en cuenta variables que no corresponden en la placa construida. El hecho de haber enviado a fabricar la placa, quita ruido e inestabilidades que puede generar el circuito realizado mediante otros métodos. Finalmente, se resalta que los resultados obtenidos en el Hardware real que se va a implementar, son satisfactorios, y cercanos a la ganancia ideal diseñada.



(a) Amplificador 82V/V (b) Amplificador 100V/V



(c) Amplificador 150V/V (d) Amplificador 200 V/V

Figura 5.25: Etapa amplificadora

Filtrado

Se utiliza nuevamente la placa Analog Discovery 2. Se inyecta la entrada mencionada, y se lee la respuesta en frecuencia del sistema. Se mide la caída 3dB utilizando dos cursores, y se obtiene una frecuencia de corte de 1607,9 Hz; se puede observar el resultado en la figura 5.27. En la figura 5.26 se observa el módulo de la respuesta en frecuencia del sistema.

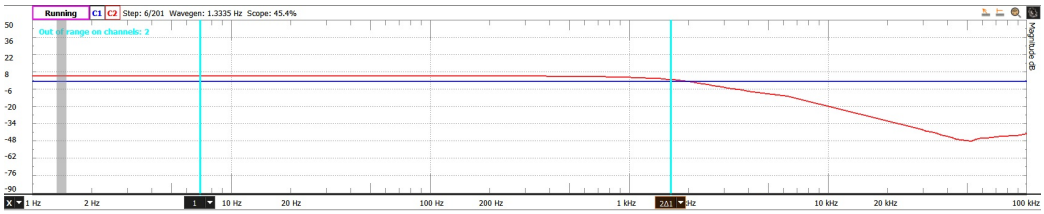


Figura 5.26: Respuesta en frecuencia del circuito real

Normal Delta Clear Show

	Position	Ref	Δ X	decade	C1	C1 ΔY	C2	C2 ΔY
1	6.9266 Hz	none			0.0158 dB		4.637 dB	
2	1.6148 kHz	1	1.6079 kHz	2.3676	0.0147 dB	-0.0011 dB	1.614 dB	-3.0231 dB

Figura 5.27: Frecuencia de corte real

Nuevamente hacemos un análisis de los datos reales, simulados y reales. Se observan en la tabla 5.9

$fc_{ideal}[Hz]$	$fc_{simulada}[Hz]$	$fc_{real}[Hz]$
1500	1556,4	1607,9

Tabla 5.9: Validación: frecuencia de corte

Se observa que la frecuencia de corte real es cercana a la ideal y a la simulada. Teniendo en cuenta que el valor de los componentes no son exactos, esto genera una leve diferencia, y se considera aceptable para las especificaciones diseñadas.

Punto de operación

De forma análoga se comparan los valores del punto de operación. Aquí no hay prácticamente variables que influyan en esta medida, se observa en la tabla 5.10. El punto de operación real se midió con un osciloscopio Hantek portátil con 3 cifras decimales de resolución.

$PO_ideal[Hz]$	$PO_simulada[Hz]$	$PO_real[Hz]$
1,560	1,560	1,559

Tabla 5.10: Validación: punto de operación

Comunicación I2C maestro - esclavo

Aquí se mencionan las pruebas realizadas para verificar el funcionamiento del conversor de nivel de voltaje bidireccional, utilizado para la comunicación I2C entre el módulo principal y este módulo de adquisición de señales vectoriales. Se enviaron números consecutivos desde una Arduino a otra, y viceversa, y se verificó en la ventana del puerto serial que se recibieran todos los valores enviados. Se pinchó además la señal de datos de I2C *SDA* y se observaron las conversiones de voltaje correctamente, se puede ver en la figura 5.28 lo verificado en el osciloscopio: en azul se despliega la señal de nivel 5V, y en verde 3.3V.

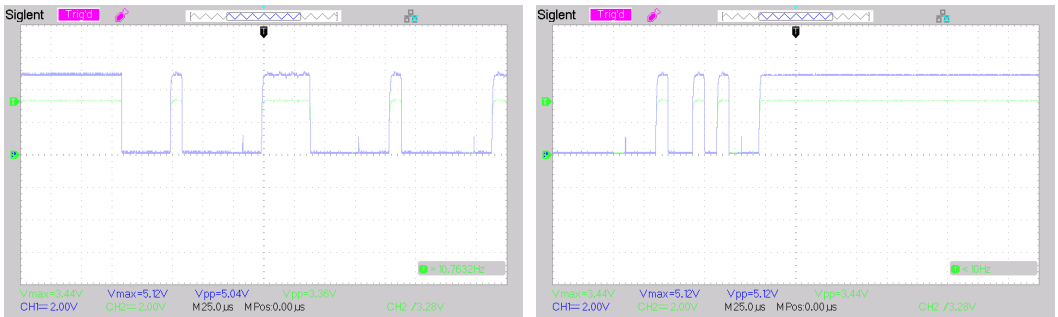


Figura 5.28: Señal SDA a través del convertidor de nivel de voltaje para la comunicación I2C

Procesamiento en FPGA utilizando la Transformada rápida de Fourier

Para probar que el código encargado del procesamiento en la FPGA se haya cargado correctamente en la Arduino MKR Vidor 4000, y que funcione adecuadamente, se utilizó un programa de prueba en Octave que calcula y devuelve la FFT, donde se eligió una señal periódica arbitraria.

A través de la IDE de Arduino se imprimen las muestras procesadas por la FPGA y se comparan los bytes devueltos (signed) con los valores decimales que devuelve el programa de Octave. Se utiliza un conversor Online para comparar los resultados, y se verifica que lo devuelto por la FPGA es el resultado deseado. Los valores obtenidos se observan en la tabla 5.11.

Se advierte en la documentación del Core FFT, que los bits menos significativos de las muestras devueltas por la FPGA pueden diferir del valor de la transformada real. Esta diferencia se observa en la tabla [46] en los bits resaltados en negrita.

Muestras	Frec	Re_{FFT} Octave	Im_{FFT} Octave	Re_{FFT} FPGA	Im_{FFT} FPGA
1712	0	7067	0	0001101110011011	0
5254	$\pi/4$	-1785	1799	1111100100000110	0000011100000111
8852	$\pi/2$	-1785	0	1111100100000110	0
12450	π	-1785	-1799	1111100100000110	1111100111111000

Tabla 5.11: Verificación del diseño en FPGA con muestras periódicas

Frecuencia de muestreo

Se verificó que el muestreo de las señales de interés se realice adecuadamente. La frecuencia de muestreo es de 4 kHz, ya que la frecuencia de corte en el circuito de acondicionamiento es de 1500Hz. Se configuró el Firmware de forma de adquirir muestras a la frecuencia deseada. Se reconstruye la señal en Matlab, a partir de las muestras obtenidas. La señal de entrada es una sinusoide de 700Hz, y se verifica que se puede reconstruir correctamente. Se observa en la figura 5.29.

Prueba del sistema completo

Finalmente, se realizaron pruebas del sistema en su totalidad. Se inyectó una señal sinusoidal, que pasa por el circuito de acondicionamiento de señales, ingresa a la Arduino MKR Vidor 4000 donde es adquirida a 2kHz, y es procesada en la FPGA. La señal de entrada es la siguiente:

- Sinusoidal
- Amplitud pico 10mV
- Offset 0V

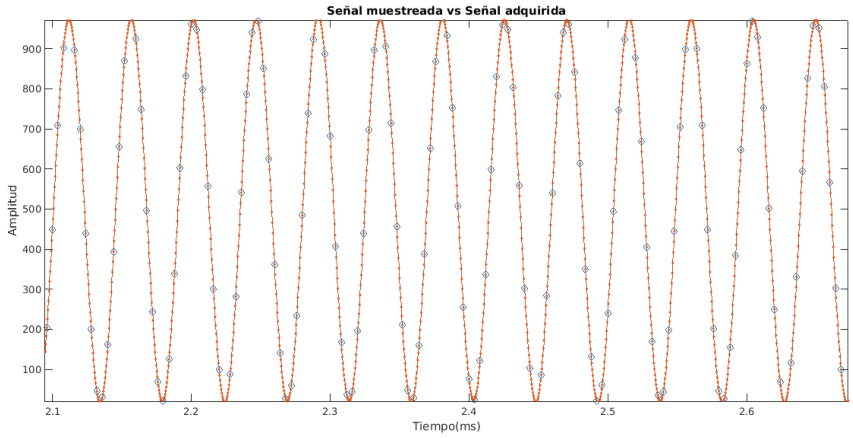
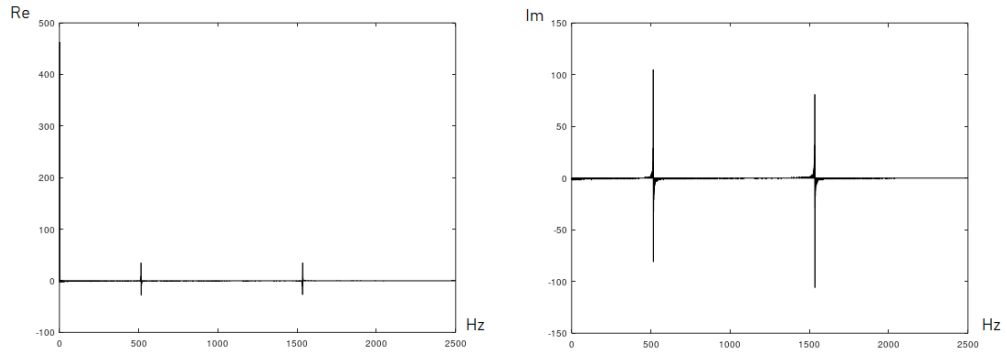


Figura 5.29: Señal reconstruida a una frecuencia de muestreo de 4kHz

- Frecuencia 500Hz

Las muestras procesadas son recibidas por el microcontrolador. Se adquieren 32 ventanas de 500ms, el micro recibe las muestras promediadas, y realiza un promedio de las 32 ventanas. Este promedio se imprime en pantalla para luego ser graficado en Octave; con el fin de constatar el correcto funcionamiento del sistema. El promedio de las muestras se ven desplegadas en la figura 5.30. En 5.30(a) se muestra la parte real del promedio, con una componente principal en 0 Hz, correspondiente a la continua del seno, y otra componente de menor amplitud en 500Hz, correspondiente a la frecuencia $\pi/4$, ya que la frecuencia de muestreo 2kHz corresponde a 2π. La componente que se observa en la frecuencia de 1500Hz nace por la periodicidad del espectro, y la simetría par en la magnitud; por lo tanto, la componente que se observa en 1500Hz, corresponde a la componente "negativa" del espectro espejado centrado en 2kHz. Este asunto se resuelve aplicando un filtro por software, eliminando las frecuencias mayores a la frecuencia de corte del sistema: 1500Hz. La parte imaginaria del promedio se observa en la figura 5.30(b) donde aparece una componente en 500Hz, y de forma análoga, por la simetría de la magnitud, aparece una componente en 1500Hz.

Estas gráficas desplegadas anteriormente fueron de utilidad para constatar el correcto funcionamiento del sistema, pero no son las muestras que se enviarán al módulo principal para verse desplegados luego en el servidor. En el microcontrolador se realizó una operación más. Se calcula el módulo del promedio de las 32 ventanas, y se vuelve a promediar, ahora por bandas. Se tienen 1024 coeficientes promediados, y se toman 32 muestras por banda, lo que resume a un total de 32 bandas que se envían al módulo principal. El módulo del promedio por bandas se observa en la figura 5.31.



(a) (b)

Figura 5.30: Promedio muestras procesadas por FFT de un seno de 500Hz: (a) Parte Real del promedio; (b) Parte Imaginaria del promedio;

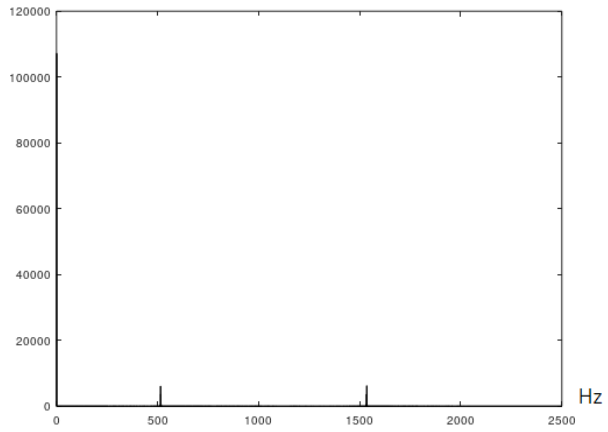
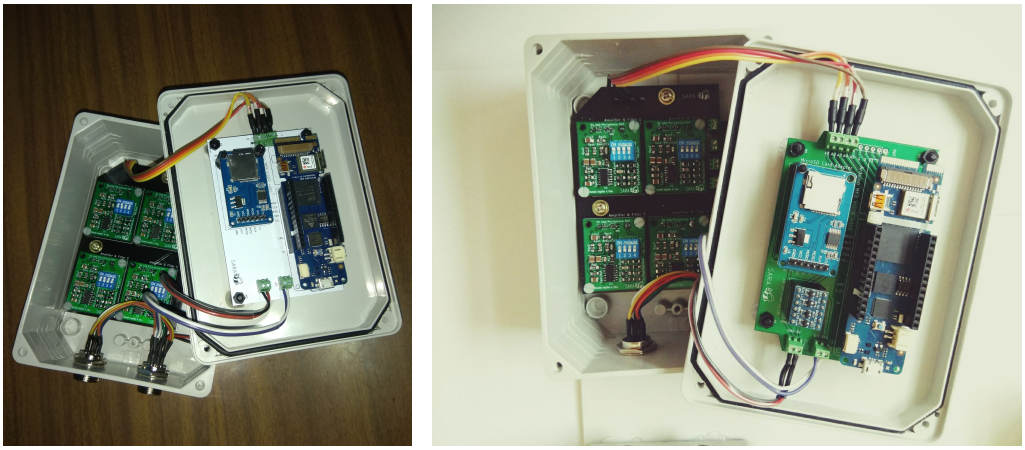


Figura 5.31: Promedio del módulo por bandas para un seno de 500Hz

Módulo final

El módulo diseñado se adapta a una caja estanca de 15cmx11cm, con conexiones exteriores al BUS de alimentación e I2C, y a los 4 canales de audio. En la figura 5.32 se muestra la primera versión del módulo MASV, y la versión final. La diferencia entre estas dos versiones es el Shield principal donde se ubica la Arduino y el módulo SD. En la segunda versión se agrega el conversor de nivel de voltaje. El módulo desde una vista exterior se observa en la figura 5.33.



(a) Primera versión (b) Versión final

Figura 5.32: MASV

Figura 5.33: MASV en su caja estanca con los 4 canales de audio y conectores del BUS

5.6. Trabajos a futuro

Raspberry Pi

En el transcurso del proyecto, surgió la idea de hacer en paralelo dos módulos: el módulo con la Arduino MKR Vidor 4000 y otro módulo de adquisición de señales vectoriales utilizando como plataforma Raspberry Pi, en particular utilizando una placa Raspberry Pi Zero. Esta placa no tiene un ADC incorporado, por lo que se considera incluir micrófonos digitales, con un ADC interno. Los micrófonos que podrían adaptarse a ese sistema son los trabajados en la tesis "UrbanEar: Monitoreo sonoro urbano de bajo costo" [26]. La razón principal de utilizar una placa Raspberry Pi es su amplia capacidad de memoria y posibilidad de desarrollar un procesamiento más profundo. Se podrían tomar ventanas de muestreo más amplias. La FFT se implementaría en software, y con las muestras procesadas se

podría hacer estudios estadísticos, u obtener conclusiones del resultado de forma local.

Micrófonos de mayor sensibilidad

En cuanto a los transductores utilizados en este módulo: micrófonos electret, se podrían realizar grandes mejoras. Estos micrófonos fueron seleccionados, principalmente, porque son accesibles en plaza, tienen bajo costo y se adaptaban a las necesidades del proyecto. Sin embargo, se recomienda hacer un estudio de mayor profundidad de estos micrófonos seleccionados, y realizar pruebas que verifiquen su respuesta para el rango de frecuencia de funcionamiento. Para aplicaciones donde se desee obtener señales sonoras más débiles, se debería abrir el abanico posible de micrófonos y sustituirlos por otros de mayor sensibilidad.

Estudio de señales a frecuencia de muestreo mayores

El módulo tal como se llevó a cabo, con la Arduino MKR Vidor 4000, tenía a priori una limitante en la frecuencia de muestreo. Se trabajó para alcanzar una frecuencia de muestreo hasta de 40MHz, y puede ser de gran interés verificar la adquisición para frecuencias altas, para la re-utilización del módulo en otras aplicaciones.

Ganancia variable controlada por Software

Una mejora en la etapa amplificadora variable es la automatización del manejo de la ganancia variable. La ganancia se podría controlar a través de comandos que envíe el usuario por el servidor. Se debería quitar el DIP Switch hoy implementado, y utilizar llaves analógicas: o bien un integrado (ie TC4066BP), o construirlo con transistores; y que la señal de control sea una salida digital de la placa Arduino.

Comunicación micro-FPGA en paralelo

La implementación para la comunicación entre el microcontrolador y la FPGA fue realizada por SPI, y se vio que enlentece considerablemente el procesamiento de muestras. Si bien, el tiempo de procesamiento para nuestro proyecto no es crítico, pueden existir otras aplicaciones donde sí lo sea, por ello se sugiere utilizar una interfaz de comunicación paralela, así alcanzar una velocidad mayor en la transmisión y recepción de muestras. Esto es realizable, ya que existen pines suficientes, sin uso, entre el microcontrolador y la FPGA.

Capítulo 6

Módulo principal

El módulo principal es el encargado de gestionar el sistema, los tiempos, alertas de pérdida de conexión con los módulos, el almacenamiento local de los datos obtenidos, la transmisión de datos al servidor, y la recepción remota de comandos.

6.1. Elección de Hardware de Desarrollo

Se buscó que fuera accesible económica y técnicamente, por lo que se consideraron dos plataformas posibles: *Raspberry* y *Arduino*.

En lo que refiere a Raspberry el dispositivo considerado fue Raspberry PI Zero. Ésta es la placa con menos atributos de la empresa. Mientras que de la compañía Arduino se tuvieron en cuenta los modelos UNO, MEGA y DUE. En la tabla 6.1 se presentan las características estudiadas para definir la elección del hardware.

Marca	**Raspberry Pi**	**Arduino**		
Modelo	Zero - Zero w	Uno	Mega	DUE
Memoria SRAM	512 MB	2kB	8kB	96kB
Velocidad del reloj	1GHz	16MHz	16MHz	84Mhz
Sistema Operativo	Si	No	No	No
Precio Tienda Oficial (U$S)	5 - 10	22	38.5	35,5
Precio Local ($)	No hay - 2015	282 ∽ 844	685 ∽ 1724	3861
Consumo en corriente (mA)	170-175 [44]	46	93	75

Tabla 6.1: Tabla de comparación de Hardware Modulo Principal

En la tabla 6.1 Arduino UNO y MEGA presentan margenes amplios de precio. Esto se debe a la existencia en el mercado local tanto de imitaciones como de placas originales.

Se decidió elegir en base al precio y la disponibilidad del mercado local. Dentro de éstas el consumo energético de Raspberry PI Zero W es sensiblemente superior

a las placas Arduino. Por ello se evaluó que de las placas del mercado era mejor utilizar la gama que ofrece Arduino.

Dentro de la gama Arduino se descartó DUE debido al costo. La Arduino MEGA fue la elegida ya que ofrece mayor memoria SRAM respecto a la UNO, esto permite mayor libertad para desarrollar un sistema robusto.

6.2. Elección de componentes

Para las tareas de almacenamiento local y comunicación remota se requirió de hardware específico.

El almacenamiento local se realizó en tarjetas microSD. Estas se comunican con la Arduino mediante el protocolo SPI. Hay librerías desarrolladas para Arduino que permiten utilizar de manera sencilla esos módulos.

Para solucionar la comunicación remota, se requirió de un módulo que permitiera la conexión a internet a través de las redes de telefonía. Con este fin se eligió GPRS como tecnología de red y el modulo SIM800 como módem con base en los siguientes motivos:

- En este momento es la red telefónica más extendida [2], incluso en lugares de difícil acceso.
- Dentro de las tecnologías que se encuentran desplegadas en todo el territorio nacional es la de menor consumo energético.
- Se cuenta con módulos robustos y de bajo precio.
- Existen librerías disponibles para utilizarlo como módem.

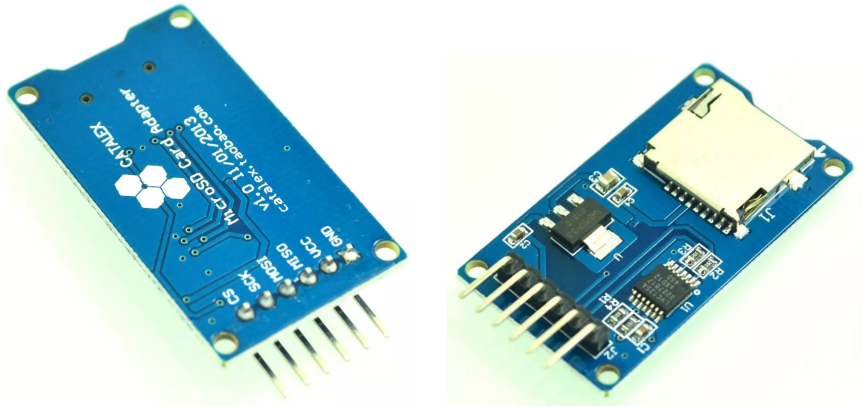
6.2.1. Módulo de almacenamiento microSD

El módulo de tarjetas de memoria que se utilizó se puede visualizar en la Figura 6.1.

El manejo de este módulo se realizó a través de la librería **SD** de Arduino [5]. La cual está basada en la librería **sdfatlib** [17]. Con ella se puede manejar tarjetas con formato FAT16 y FAT32.

6.2.2. Módulo de comunicación GPRS

El otro submódulo que contiene el Módulo Principal es un módem GPRS. Es a través de éste que se establece la conexión a internet.



(a) Parte trasera de la lectora SD (b) Parte delantera de la lectora SD

Figura 6.1: Módulo de lectura/escritura SD

En la figura 6.2 se muestra una imagen del módulo SIM800 y el detalle de sus pines. El Arduino se comunica con el módulo mediante comandos AT, estos son transmitidos a través de una interfaz serial.

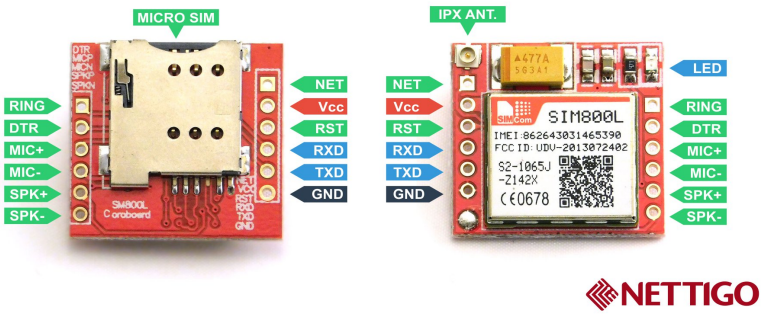


Figura 6.2: Módulo SIM800 - GPRS

El uso del módulo SIM800 está ampliamente expandido en el mundo de la tecnologías IoT debido a su robustez, su bajo costo, una amplia cartilla de comandos AT, entre los cuales están implementados los protocolos TCP/IP y HTTP. Hay múltiples implementaciones del protocolo MQTT que utilizan este módulo, y se encuentran con desarrollo para Arduino.

Se analizaron varias opciones de librerías disponibles. La librería **SIM800_MQTT** [14] resuelve tanto la comunicación con el SIM800 como el protocolo MQTT, sin embargo no cuenta con mantenimiento desde el año 2015, y es el motivo por el cual fue dejada de lado. Cabe destacar que hay una actividad importante de usuarios en el foro de GitHub de este proyecto.

Otra librería fue **Mqtt-and-Sim800-900** [25]. En este repositorio se cuenta con una solución para MQTT con SIM800 y SIM900. La implementación es basada

en librerías de terceros, las cuales han avanzado y los autores no han acompasado estos cambios. Por lo cual se descartó utilizar esta solución.

Finalmente se optó por utilizar en conjunto las librerías **PubSubClient** [33] y **TinyGSM** [38]. TinyGSM es desarrollada para manejar como módems a módulos de redes telefónicas. Una lista completa se puede encontrar en la web del proyecto. En particular, permite utilizar como módem distintas versiones de SIM800 y SIM900. Dentro de los ejemplos de uso se encuentra el uso como cliente MQTT.

Para la comunicación MQTT se utilizó la librería PubSubClient, ya que era empleada en la librería Mqtt-and-Sim800-900 y sugerida en TinyGSM .

Ambos proyectos están activos tanto en desarrollo como en comunidad.

6.3. Arquitectura de Software

Se estudiaron pros y contras de las siguientes arquitecturas de software:

- Round Robin (RR)
- Round Robin con Interrupciones (RR-I)
- Encolado de Funciones (EF)
- Sistema Operativo para Tiempo Real (RTOS)

Un desarrollo más profundo se encuentra en el apéndice A.

Inicialmente se seleccionó la arquitectura Encolado de Funciones, ya que permite asignar y manejar prioridades en las tareas de forma más efectiva que las variantes de Round Robin. Permite estructurar agendas y requiere menos procesamiento que un RTOS.

Esta implementación no encolaba únicamente punteros a funciones. Se encolaban estructuras que tenían el puntero a la función, y también una lista de parámetros necesarios por éstas.

Este diseño desembocó en una lógica muy compleja, derivando en la investigación del uso de RTOS.

En esta nueva línea de trabajo, se estudió la librería FreeRTOS [15]. Migrar a esta esquema permitía un mejor manejo de los tiempos de respuesta, así como del manejo de prioridades. Prontamente este camino también fue abandonado, puesto que el uso en distintas partes del código de la función **delay** de Arduino generaba grandes problemas. Si bien se eliminó el uso de dicha función en todo lo implementado por los autores, aún se utilizaban en las librerías de terceros. Por lo cual finalmente, en común acuerdo con tutores, se decidió migrar a una arquitectura tipo Round-Robin, con el flujo que se observa en la Figura 6.3.

Este flujo se definió dejando de lado las prioridades, y respondiendo de forma exacta a los tiempos, de forma de que se mejoró el control sobre el sistema. Bajo esta configuración también se logró informar sobre conexiones y desconexiones entre el principal y los módulos detectados.

En una versión previa, los mensajes se almacenaban en una cola, y se enviaban cuando ya no se debían tomar más medidas o cuando se detectaba que no se tenía memoria suficiente para almacenar el dato que se debía obtener a continuación. Este enfoque se abandonó debido a la complejidad requerida para un correcto manejo de la memoria.

En la figura 6.3 se observa el flujo que ocurre en el dispositivo, el cual se puede dividir en tres grandes bloques. El bloque *inicialización*, se puede ver en la figura 6.3(b). Los otros dos ocurren en un bucle infinito. En la función principal se ejecutan las acciones, su flujo se observa en la figura 6.3(c), y la función de bajo consumo. Para salir de bajo consumo hay un timer que despierta al equipo cada 5 segundos, en ese momento se realiza una verificación de los tiempos de las agendas de los canales de los módulos. Si hay un módulo al que se debe solicitar una medida o solicitar respuesta, se ejecuta la función principal.

Durante la inicialización se realiza un escaneo de los módulos que están conectados al bus. El flujo se puede observar en la figura 6.4.

Tras esta solicitud de datos, el módulo principal, maestro del bus, tiene armado un perfil de cada módulo esclavo. Tiene una agenda que cumplir y la información necesaria para poder intercambiar información por I2C utilizando el protocolo SSCMP.

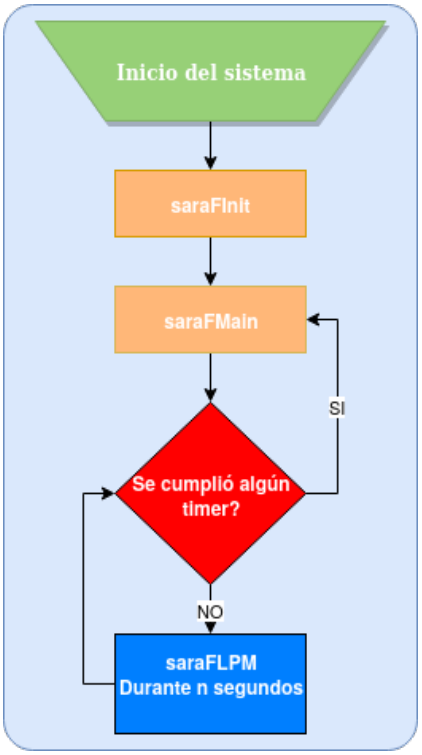
Luego de adquirir los datos de los módulos esclavos, se publica dicha información en el topic[1] <proyecto><dispositivo> / <propiedades> del broker MQTT.

Posterior a la inicialización, y mientras no se esté en bajo consumo, se ejecuta la función principal. Cuando se detecta que hay una acción que tomar en base a la agenda, se utiliza la función **saraFMedirCanal**, cuyo flujo se observa en la figura 6.5. En esta función se instancian otras, dentro de ellas se encuentra la función en la cual se solicita la respuesta a una medida. Esta función se llama **saraSolicitarRespuesta**. La misma establece una comunicación I2C; una vez que recibe la medida, arma un mensaje, lo guarda en la tarjeta SD, y posteriormente lo publica vía MQTT.

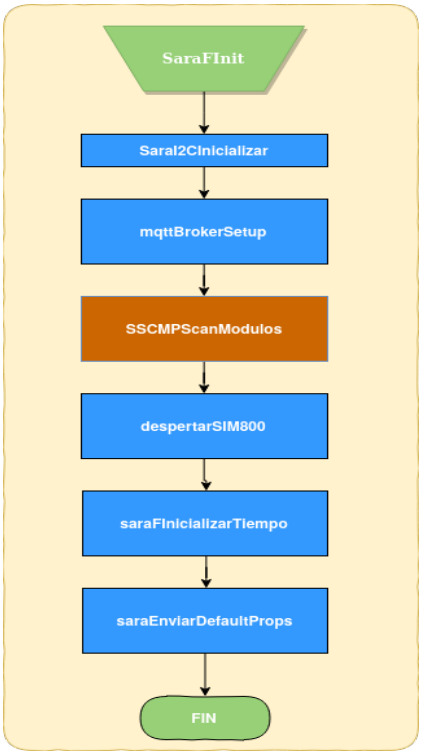
En la figura 6.6 se pueden ver los flujos de *saraSolicitarRespuesta* figura 6.6(a) y *saraFGuardarSD* figura 6.6(b).

En base a los diagramas anteriores se puede tener una idea cabal de la lógica

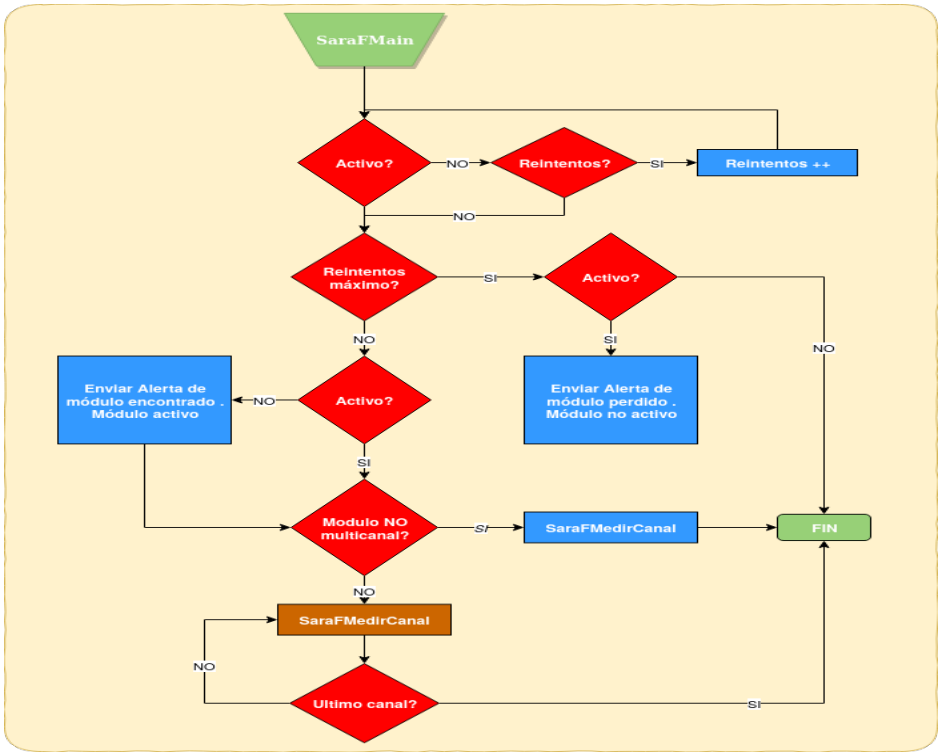
[1] En el anexo F se puede obtener más información al respecto.



(a) Flujo general del módulo



(b) Flujo de inicialización



(c) Flujo normal del equipo

Figura 6.3: Flujo del módulo principal

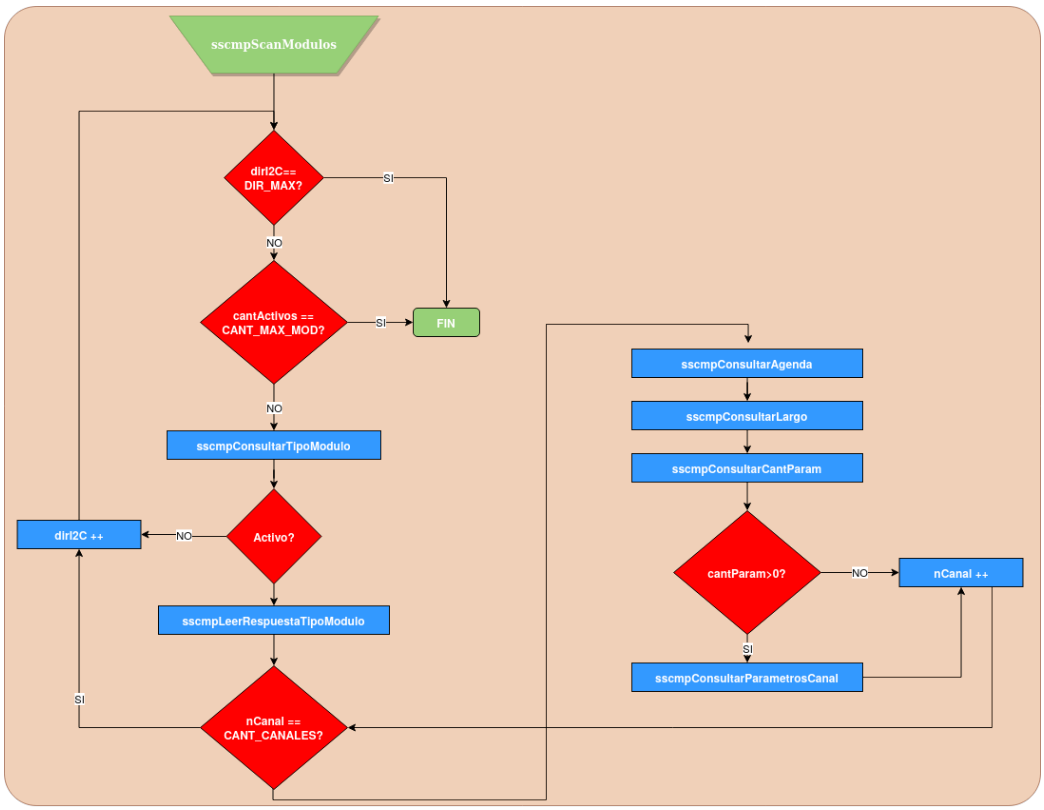


Figura 6.4: Flujo del escaneo de módulos esclavos

del módulo principal.

Finalmente en la figura 6.7 se puede ver la función de *callback* de MQTT. En MQTT una función de *callback* es aquella que se ejecuta cuando un mensaje es recibido en alguno de los topics a los que se está suscrito. En este proyecto sólo se reciben mensajes en el topic **SARA/SARABEE/cmd**. Se pueden ver de la siguiente forma <**proyecto**>/<**dispositivo**>/**cmd**. Esta función es requerida por todos los clientes de MQTT y es específica de la solución particular. En el caso de *SARA-Bee* se desea procesar el mensaje recibido y tomar acciones. Los comandos que se pueden recibir a través de MQTT y las acciones que estos ejecutan se pueden ver en la tabla 6.2.

Los comandos recibidos pueden utilizar como separador los siguientes símbolos “,”, “;”, “@”, “ ”, “/” y “-”. Los comandos pueden llegar en mayúscula, en minúscula o con letra capital. Se debe tener en cuenta que se debe utilizar un carácter de fin de línea. En el caso de *SARA-Bee* se decidió utilizar la letra **b**.

Se debe tener en cuenta que al enviar el comando GET con la dirección **0**, se realiza una solicitud con las configuraciones de todos los módulos.

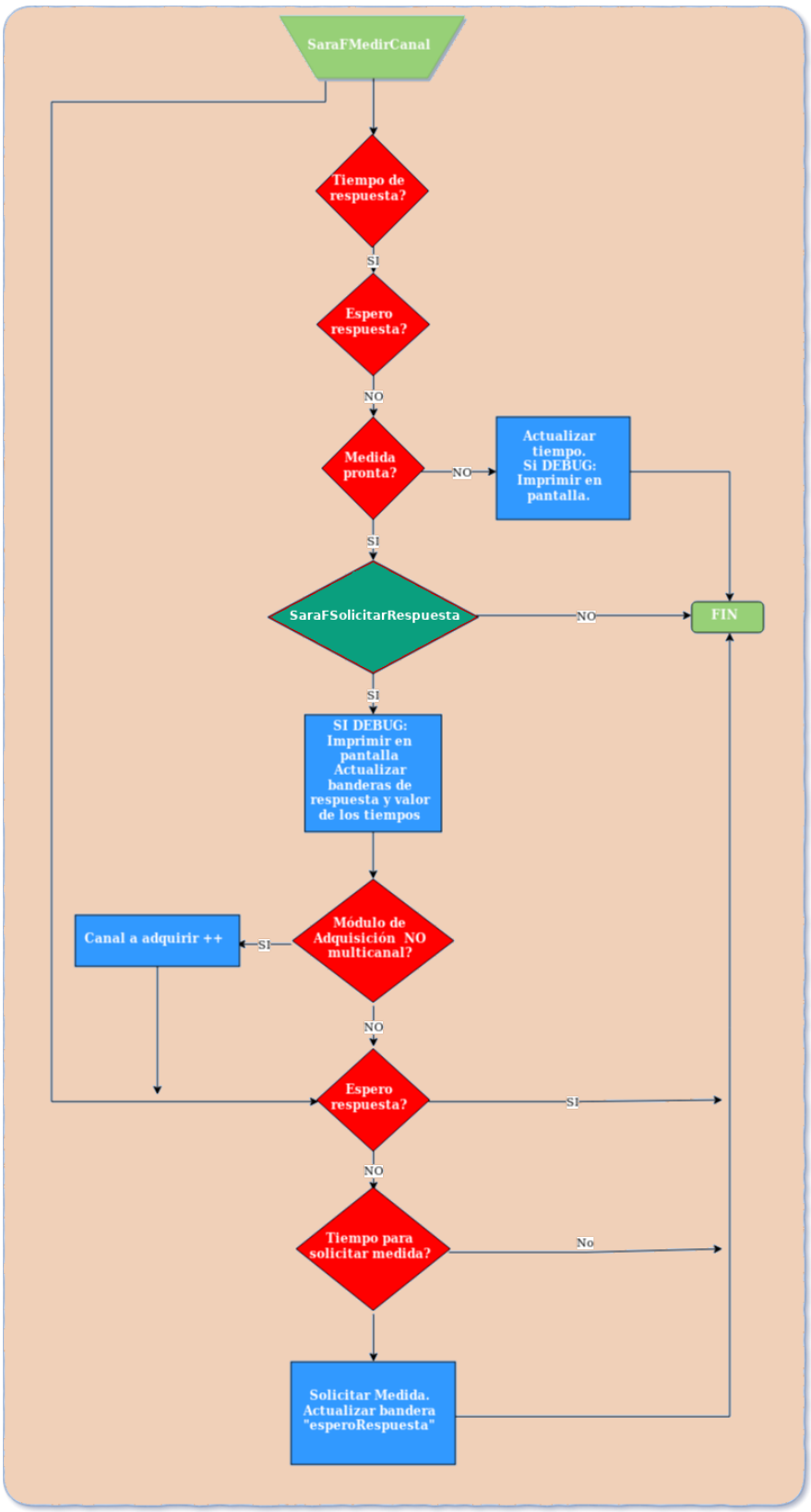
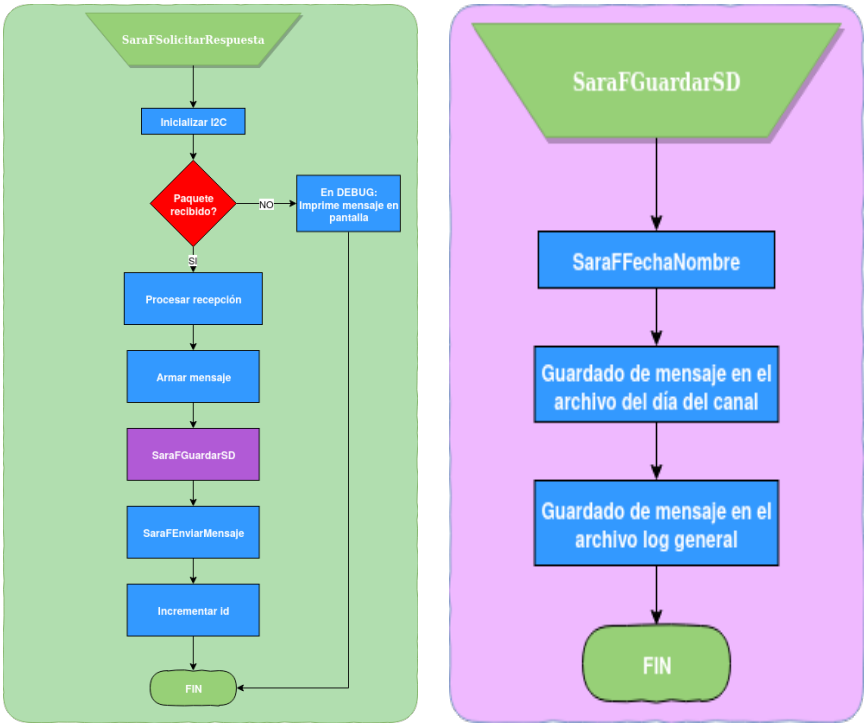


Figura 6.5: Flujo de la función utilizada para adquirir medidas en un canal



(a) Flujo al solicitar respuesta (b) Flujo para guardado en SD

Figura 6.6: Diagrama de flujo en la recepción de medidas

Acción	Descripción de la acción	Comando
RUN	Solicitar que se adquiera una medida al momento de recibir el mensaje	run-<nModulo>-<nCanal>b
SET	Solicitar cambios en parámetros o agenda	set-<nModulo>-<nCanal>-<valPar1>-<valPar2>pb set-<nModulo>-<nCanal>-<agenda1>-<agenda2>ab
GET	Solicitar propiedades o agende un módulo	get-<nModulo>b

Tabla 6.2: Comandos recibidos por MQTT

En caso que se desee cambiar los tiempos de la agenda, por ejemplo si se quisiera adquirir el canal 2 cada 10 segundos y esperar la respuesta 4 segundos, se solicita un cambio de agenda de la siguiente forma: **SET-2-2-10-4ab**. El procedimiento para realizar un cambio en los parámetros de un canal es análogo al cambio de los tiempos de la agenda. Por ejemplo, suponiendo que la medida de temperatura tuviese dos parámetros configurables, y este fuera un valor entero entre 0 y 5, para solicitar el cambio de parámetro, se debe saber qué:

- el módulo de adquisición auxiliar MASE tiene la dirección I2C 2
- el canal de temperatura es el 1

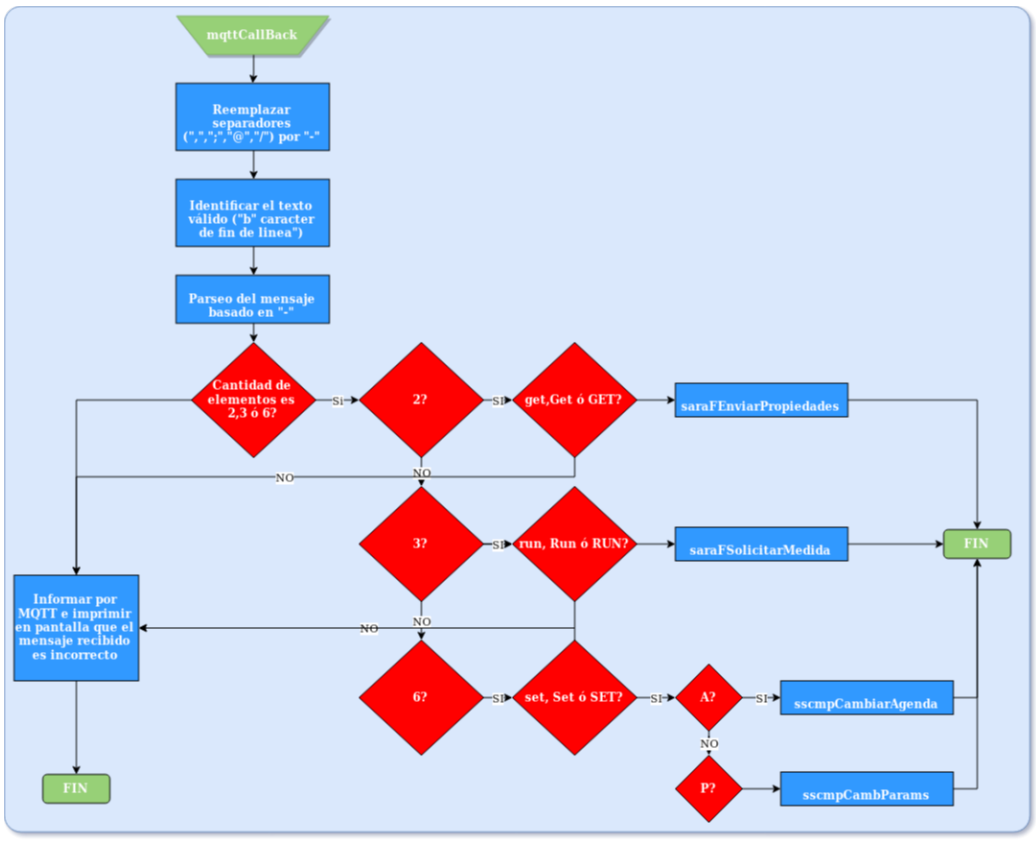


Figura 6.7: Flujo dentro de MQTT callback

por lo tanto, solicitar el cambio se realiza con el texto **set-2-3-1-2pb**. En el caso de *SARA-Bee* este mensaje será rechazado pues el módulo MASE no tiene parámetros.

6.4. Diseño de Hardware

El diseño de hardware para este módulo es la implementación de un *shield* para Arduino MEGA el cual abarca aspectos de comunicación y almacenamiento. No tiene hardware de adquisición ni de adaptación de señales.

6.4.1. Diseño de Shields

Una vez elegida Arduino MEGA como placa de desarrollo para el módulo principal, se diseñó un *shield* para dicha placa. Los motivos para diseñarlo a medida fueron los siguientes:

- No había shields de Arduino para el SIM800, sí para el SIM900 que es una versión anterior.

- Se requería un regulador de 5V a 4V con *enable* para apagar por completo el módulo GPRS para ahorrar energía.
- Se requería un módulo microSD.

Los requerimientos del *shield* son:

- Bornera con 5V, SCL (I2C), DATA(I2C), GND.
- *Pull-up* en las líneas de I2C.
- Huella para el módulo SIM800.
  - Circuito de regulación de voltaje, de 5V a 4V para el SIM800.
  - Conexión de un pin digital de salida del microcontrolador con *enable* del circuito de regulación.
  - Conexión puerto serie Serial1 de Arduino al puerto serie del SIM800.
- Huella para tarjeta microSD.
  - Conexión alimentación.
  - Conexión SPI del módulo con Arduino.

El shield se diseñó mediante la herramienta KiCad y se utilizó la tecnología SMD para los componentes utilizados.

Las resistencias de *pull-up* elegidas son de $2{,}2k\Omega$, y se eligieron en base a pruebas sobre I2C. La Arduino MEGA tiene incorporada resistencias de *pull-up* de $10k\Omega$ pero no fueron suficientes. Un análisis más profundo en este aspecto se realiza en la sección Pruebas.

En la figura 6.8 se muestra el circuito utilizado para la regulación de 5V a 4V para el SIM800. Dicho circuito fue obtenido de una *Application Note* para el diseño de hardware [13], donde también se recomendó usar un diodo Zener y capacitores extra para la sobretensión y el ruido/interferencia. Éstos se agregaron al PCB pero en la práctica no se utilizaron.

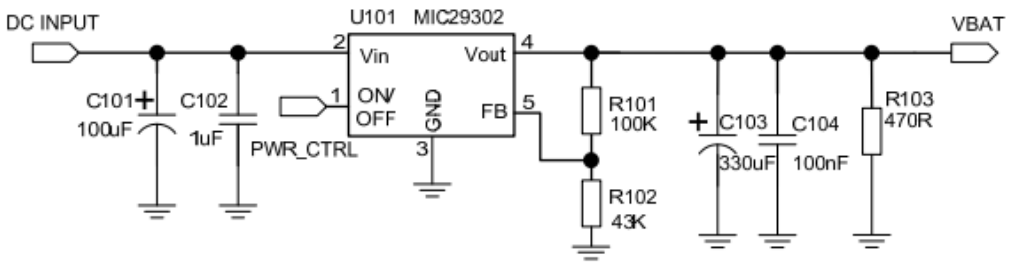


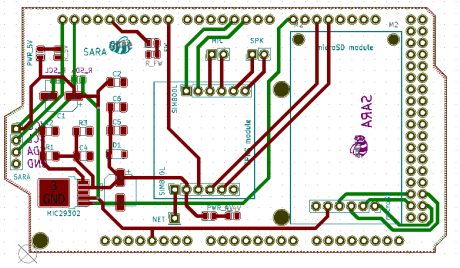
Figura 6.8: Circuito de regulación de voltaje para el SIM800

Se agregaron LEDs de *power* opcionales para 4V y 5V, y el LED en el pin 13 de Arduino.

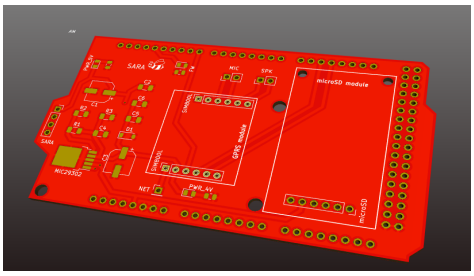
En la figura 6.9 se observa el PCB diseñado. En la subfigura 6.9(a) se observa el diseño propiamente dicho. El software utilizado, KiCad, permite obtener representaciones 3D las cuales permiten tener una clara idea de como será la placa fabricada, esto se puede ver en la subfigura 6.9(b).

Finalmente en la subfigura 6.9(c) se presenta el *shield* integrado en el sistema con los componentes soldados e incorporados.

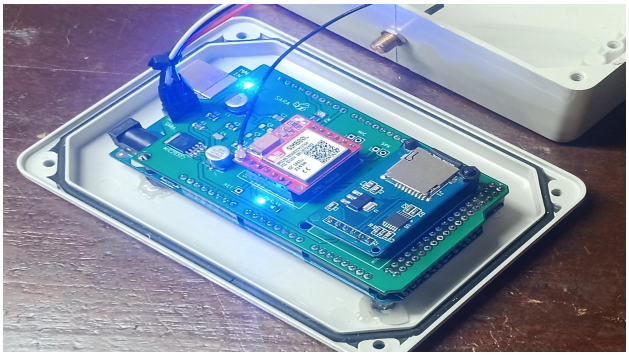
El módulo SIM800 y la lectora de SD no van soldados sobre el *shield*, sino que se sueldan zócalos, donde van encastrados y sostenidos por tornillos.



(a) Layout del PCB



(b) Captura 3D la parte anterior



(c) Fotografía del shield final ensamblado

Figura 6.9: Diseño y ensamble de shield para módulo principal

El diseño y desarrollo de este *shield* permitió mejorar el *testing* de los problemas del firmware pues solucionó problemas de alimentación que ocurrían con anterioridad.

Además permite brindar, a quien desee, una solución definida y segura.

6.5. Pruebas y Validación de módulo

Para la validación del módulo se realizaron pruebas a nivel de hardware y firmware.

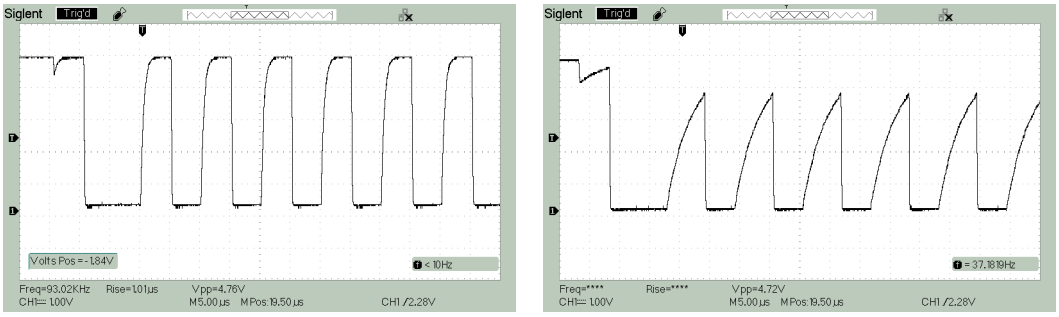
6.5.1. Hardware

Las pruebas de hardware en primera instancia fueron para evaluar el *shield* y fueron las siguientes:

- En I2C:
  - Niveles de voltaje.
  - Velocidad.
- Para el SIM800:
  - Regulación de voltaje.
  - *Enable*.

La velocidad máxima está limitada por el el cableado del bus, lo deseable era que esta fuera mayor o igual a $100kHz$. Cuanto mayor es la velocidad, puede decrecer la distancia total del bus. Disminuyendo el valor del *pull-up* se puede aumentar la distancia, esto trae un aumento de consumo y a su vez provoca que los niveles de voltaje difieran de lo ideal. Las pruebas se realizaron con todos los módulos conectados, por lo que la elección de dicha resistencia está atada a la topología del bus.

En la Figura 6.10 se observa el caso del *pull-up* $10k\Omega$ de la Arduino MEGA, esto no es suficiente, ya que el voltaje que alcanza es $3{,}68V$ cuando la señal sube a $5V$. Se probaron distintos valores hasta llegar al valor óptimo de $2{,}2k\Omega$, este valor genera un buen tiempo de subida/bajada de la señal con los niveles de voltaje de las líneas y el consumo.



(a) Pull-up 2k

(b) Pull-up 10k

Figura 6.10: Elección de la resistencia de *pull-up* para las lineas I2C.

La regulación del voltaje fue exitosa, en la tabla 6.3 se muestran el valor relevado y medido.

Teórico (V)	Medido (V)
4.10	4.12

Tabla 6.3: Voltaje teórico y obtenido para el regulador lineal.

Por otro lado, el *enable* funcionó correctamente, habilitando y deshabilitando el regulador.

6.5.2. Firmware

A nivel de firmware se evaluó:

1. Envío de mensajes MQTT.
2. Escaneo del bus. Figura 6.11(a).
3. Envío de medidas adquiridas. Figura 6.11(b) y figura 6.11(c).
4. Correcto manejo de la agenda.
5. Detección de módulos desconectados. Figura 6.11(d)
6. Detección de módulo reconectado. Figura 6.11(d)
7. Recepción de mensajes desde MQTT.
8. Cambio de parámetro vía MQTT
9. Solicitud de medida vía MQTT
10. Solicitud de parámetros vía MQTT

El envío de mensajes MQTT se probó publicando un mensaje en el broker al momento de establecer la conexión, y publicando un mensaje previo a realizar la desconexión. La figura 6.11 muestra los mensajes recibidos para los pruebas mencionadas anteriormente.

La subfigura MQTT 6.11(b) muestra mensajes de medidas recibidas de los módulos 1 - alimentación, y 2 - MASE. Y la subfigura 6.11(c) enseña una medida realizada por el módulo MASV para la adquisición de audio. En este caso, se observa en el campo *data* un vector con los datos obtenidos.

En la subfigura 6.11(a) se observa un mensaje donde se listan las propiedades de un módulo. Estas propiedades coinciden con las propiedades del MASE, módulo

que estaba siendo utilizado en ese momento.

La subfigura 6.11(d) muestra los mensajes recibidos cuando el módulo principal pierde conexión con un módulo auxiliar, y cuando reencuentra a dicho módulo.

La comprobación del ítem 4 es engorrosa puesto que se debe comprobar que el dispositivo respeta los tiempos de la agenda. Para comprobar la agenda también es necesario comprobar que se respeta la secuencialidad de los módulos no-multicanal. Para las pruebas del módulo MASV, se solicitó que cada canal de audio tuviese una diferencia de 10 minutos entre adquisición. Se solicitaba la medida del canal 1, y se esperaba 2 minutos para asegurarse que la medida estuviera pronta. A partir de ese momento, se esperaba 10 minutos más para solicitar el inicio de medida en el canal de audio siguiente. Además se debió comprobar que la respuesta fuera circular, es decir que una vez adquirido el canal 4 de audio, el siguiente fuera el canal 1.
Las pruebas se realizaron con los módulos contemplados para emular el PestiBee, este *device* se denominó SARASOL. Se utilizó otro *device* únicamente para pruebas y muestras denominado SARAPRU.

6.6. Trabajos a futuro

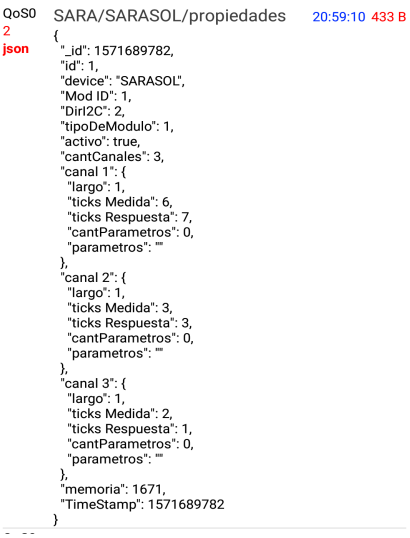
El módulo principal permite incursionar hacia distintos desafíos. Quedó por el camino lograr trabajar con *FreeRTOS*, de forma de retomar la línea de manejar prioridades y repuesta inmediata a ciertos eventos.

A su vez, el módulo principal podría ser implementado en otro hardware. En particular se puede migrar hacia el uso de *Raspberry Pi*, en cuyo caso se puede disponer de un sistema operativo, y el uso de una amplia variedad de lenguajes. Su podría cambiar el enfoque y realizar determinados procesamientos dentro de este módulo, como mejorar la calidad de audio.

Una tarea interesante sería implementar sistemas de mejoras a los tiempos. Utilizar como base los tiempos de agenda planteados por el usuario, pero insertando la búsqueda de un óptimo. Un ejemplo es aplicar un símil de TCP Reno, donde se hagan las consultas y los pedidos de respuesta, y si falla, la próxima vez no se utilice el tiempo de la agenda, sino el doble.

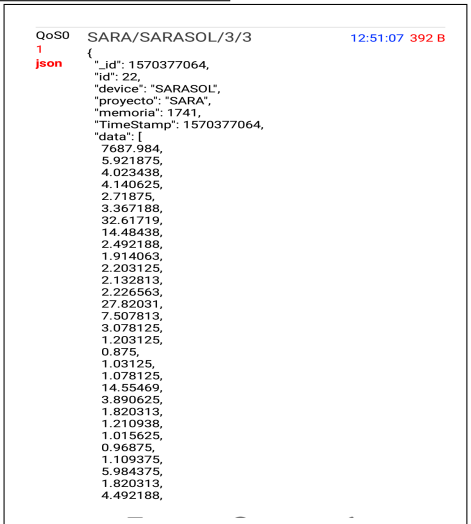
Se sugiere realizar mejoras en el manejo de alertas, del consumo y gestión de la energía, de forma de obtener información del estado de la batería y la carga, y con ello entrenar algoritmos como puede ser MPPT.

En cuanto a MQTT, migrar a una plataforma como Raspberry PI permitiría utilizar mensajes cifrados mejorando la seguridad del sistema.

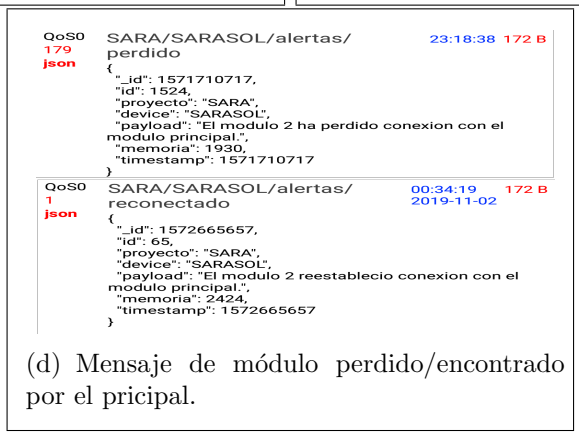


(a) Mensajes de establecimiento de conexión e informe de desconexión de MQTT.

(b) Mensajes de medidas recibidas del módulo 2, MASE.



(c) Mensaje de vector de medida recibida del módulo 3, MASV.



(d) Mensaje de módulo perdido/encontrado por el pricipal.

Figura 6.11: Pruebas de validación para el módulo principal

Capítulo 7

Módulo de alimentación

El módulo de alimentación es el encargado de brindar energía al equipo y actúa en el sistema como un bloque esclavo I2C. Está compuesto por un panel solar, una batería y un controlador.

A diferencia del resto de los módulos no hay secciones de *Elección de Hardware de Desarrollo*, *Arquitectura de Software* y *Diseño de Hardware*, dado que gracias a la generalidad del sistema, el problema a resolver es idéntico al del MASE. Por lo que se reutilizó la misma plataforma de hardware y el *shield*, y se hicieron ajustes mínimos en el firmware.

Características del módulo:

- Tipo de módulo: Alimentación.
- Dirección I2C: 1
- Cantidad de canales: 2
    - Voltaje de la batería en Volts (V).
    - Temperatura interna en grados Celsius (ºC).

Cada canal tiene largo de lectura uno.

7.1. Elección de componentes

7.1.1. Controlador

Como primer opción para el controlador se evaluó reacondicionar el hardware del *PestiBee* destinado a este fin. Como segunda opción se analizó comprar en plaza o importar una placa que resuelva dicho problema.

Placa de Alimentación de PestiBee

En primer lugar se estudió a nivel teórico que este módulo fuera independiente del *PestiBee* y compatible con nuestro sistema. La conclusión fue que efectivamente todas las conexiones con el *PestiBee* eran de alimentación o señales de salida que indicaban el estado del panel solar y el estado de carga de batería. Por lo que no sólo es compatible sino que es precisamente lo que se necesita.

En líneas generales el módulo consta de dos etapas:

- Gestión de carga de batería a través de panel solar.
- Conversión DC-DC de 12V a 5V.

El siguiente paso fue realizar pruebas para determinar en qué estado se encontraba el módulo. Cabe recordar que estaba sin uso desde hace aproximadamente tres años, cuando culminó el proyecto.

En las pruebas se determinó que la segunda etapa funcionaba correctamente, logrando reducir los $14V$ de la batería a 5 V. A diferencia de la primera etapa que no tenía un buen funcionamiento, ésta se encargaba de aplicar el método de control MPPT e informar mediante cuatro pines de control el estado de la carga de batería a través del panel solar.

Se dedujo que la causa principal del mal funcionamiento de la primera etapa era el integrado encargado del MPPT, por lo cual se realizó la compra de dicho integrado en el exterior. Para aprovechar la importación de componentes se realizó un estudio de placas similares características y se efectuó la compra de una de ellas para evitar riesgos y tener una alternativa.

Una vez arribados los componentes se prosiguió a remplazar el integrado del módulo. Se comprobó que la placa funciona de acuerdo a la documentación del *PestiBee*.

En común acuerdo con los tutores, se decidió utilizar la placa adquirida independientemente de si se lograba reparar el otro controlador.

Placa de alimentación importada

La placa adquirida es la DFR0580 [10], ver Figura 7.1. Es un módulo de gestión de energía solar de alta eficiencia y potencia media.

Características proporcionadas por el fabricante:

- Diseñado para paneles solares estándar de 18V y batería de plomo-ácido de 12V.
- Algoritmo MPPT (para 18V), maximizando la conversión de energía solar.

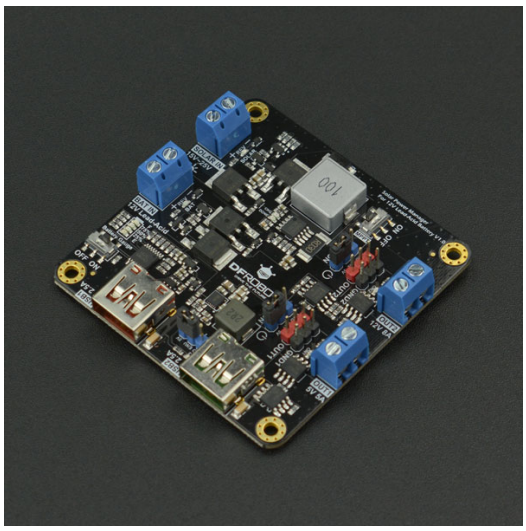


Figura 7.1: Placa de gestión de panel solar y batería del MA.

- Salidas de 12V-8A y 5V-5A.
- Protecciones para la batería.
- Indicadores de estado de carga.

Especificaciones proporcionadas por el fabricante:

- Rango de voltaje del panel solar: 15V - 25V
- Voltaje de la batería: 12V
- Corriente de carga máxima: 4A
- Voltaje de flotación: 13.55V
- Voltaje de estado constante: 14.8V
- Voltaje de liberación de protección contra sobredescargas: 11.8V
- Voltaje de protección contra sobredescargas: 10.8V
- Corriente de operación $<$ 4mA

La placa adopta un modo de tres etapas de carga para baterías de plomo ácido:

- Carga por goteo (baja corriente): cuando el voltaje de la batería se encuentra por debajo del umbral de los 11.1V, la batería se carga con una corriente inferior a 700mA hasta superar dicho potencial.
- Carga a corriente constante: cuando el voltaje de la batería se encuentra por encima de los 11.1V, la batería se carga a corriente constante de hasta 4A.

- Carga a voltaje constante: cuando el voltaje de la batería se encuentra cercano al voltaje de corte 14.8V, la batería se carga a voltaje constante, y la corriente comienza a disminuir gradualmente.

Además el módulo permite mantener la carga en flotación, para lo cual tiene programado el siguiente ciclo:

- Carga en flotación: cuando la batería se encuentra inactiva, es decir, no se registran ciclos de carga, el módulo se encarga de mantener la batería en 13.55V, su voltaje de flotación.

- Carga automática: si la batería se encontraba en flotación, pero se registra que el voltaje de la misma decrece hasta 12.42V, se inicia un ciclo de carga en tres etapas.

El circuito de protección de la placa le brinda mayor robustez y seguridad al hardware. La protección para la batería funciona de forma tal qué, cuando el voltaje cae por debajo de 10.8V, el circuito desconecta la batería para evitar que se siga descargando y generando desgaste. Cuando la batería se encuentra en un ciclo de carga, al alcanzar el voltaje de liberación de protección contra sobredescargas: 11.8V, el circuito habilita nuevamente la protección.

7.1.2. Batería

El controlador es compatible con baterías de plomo ácido; como pueden ser: baterías de gel, AGM (Absorbent Glass Mat battery), de plomo ácido sellada, de plomo ácido abierta, entre otras.

Las dos características a tener en cuenta para la elección de la batería son:

* Rango de voltaje de flotación: debe incluir 13.55V

* Rango de voltaje del ciclo de uso: debe incluir 14.8V

Se eligió una batería que cumple con estas condiciones. Se consiguió en plaza una batería de plomo sellada Ultracell UL4-12, ver Figura 7.2.

Tiene las siguientes características:

- Voltaje nominal: 12V

- Ampere/hora: 4AH en 20 horas

- Uso *standby*: 13.5V - 13.8V

- Ciclo de uso: 14.4V - 15.0V

Figura 7.2: Batería del MA.

La eficiencia de este tipo de baterías es de 85 % [28].

7.1.3. Panel solar

Para la alimentación solar se utiliza una configuración de dos paneles solares conectados en paralelo. Los paneles solares utilizados, ver Figura 7.3 son heredados del PestiBee, por lo que su dimensionado se encuentra en la respectiva documentación [1].

Esta elección se hizo en base a que la aplicación de ambos proyectos era la misma y con un consumo similar.

Las características del panel solar son:

- Potencia máxima: 10W
- Voltaje de máxima potencia: 17.6V
- Voltaje de circuito abierto: 21.6V
- Corriente de máxima potencia: 0.58A
- Corriente de cortocircuito: 0.63A

Se obtiene un equivalente de un panel de 20W de potencia y 1.16A máximos de alimentación.

Figura 7.3: Panel solar del MA

7.2. Pruebas y Validación de módulo

7.2.1. Consumo de los módulos

Los consumos de los módulos se midieron a 5V.

MASE

El consumo de este módulo es de 25mA. El consumo del módulo completo se explica:

- Arduino UNO.
- Las celdas de carga debido a su resistencia equivalente.
- El sensor DHT22.
- El módulo HX711.

MASV

El consumo de este módulo es de 50mA. El consumo del módulo completo se explica:

- Arduino MKR Vidor 4000: microcontrolador y FPGA.
- Placas de amplificación y filtrado.
- *Shift level converter*.
- Módulo microSD.

MP

El consumo de este módulo es de 70mA en promedio. El consumo del módulo completo se explica:

- Arduino MEGA.
- Conversor DC-DC lineal.
- Módulo microSD.
- Módulo GPRS.

Este módulo tiene picos de consumo elevados que rondan los 2A al transmitir a través de GPRS. El consumo promedio es alrededor de 130mA.

MA

El consumo del Arduino y el *shield* de este módulo es igual al del MASE. Se debe adicionar el consumo del controlador, con lo cual el consumo total obteniendo es 30mA.

Equipo

De la suma de lo anterior se obtiene la potencia por separado, ver tabla 7.1, con cada módulo alimentado a 5V.

Corriente (mA)	Voltaje (V)	Potencia (W)
235	5.10	1.20

Tabla 7.1: Consumo del equipo en 5V.

Por otro lado se relevó la potencia del equipo completo, ver tabla 7.2, desde la alimentación de 12V proveniente de la batería.

Corriente (mA)	Voltaje (V)	Potencia (W)
95.0	12.6	1.20

Tabla 7.2: Consumo del equipo en 12V.

En la figura 7.4 se muestra el consumo en tiempo real, se puede ver que el consumo del equipo es constante con picos impuestos por la comunicación GPRS.

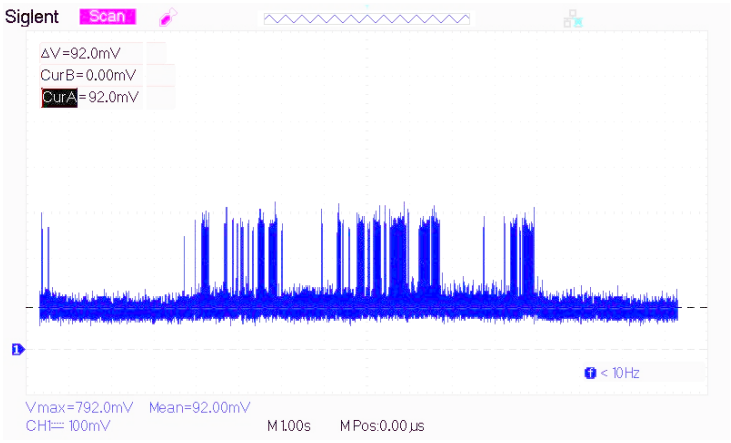


Figura 7.4: Consumo en tiempo real del equipo en 12V

7.2.2. Estimación de duración de batería

Se toma el peor escenario para la estimación de la batería. Se suman los consumos de los módulos y se llega a un total de 42.2 horas. Esto es un cálculo ideal. Si se estudia la limitación real de la batería según los datos del fabricante, se obtiene que es capaz de dar la corriente deseada por un período de 12 horas, teniendo en cuenta el voltaje de la batería a la cual se realizó el estudio. Se observa este resultado en la tabla 7.3. Estas pruebas se hicieron a 12.6V con una resistencia de 0.5Ω. Para este cálculo se toma como hipótesis que el panel no le está proporcionado energía.

En la práctica este valor es menor debido a que el sistema de protección del controlador corta la alimentación proporcionada la batería cuando esta cruza del umbral de 11.8V. A nivel de laboratorio se observó que esto reduce un 70 % la duración real.

Corriente (A)	Cantidad de carga (A/h)	Duración (h)
0.095	4.0	12

Tabla 7.3: Estimación de duración de batería.

7.2.3. Eficiencia

Se calculó la eficiencia energética del módulo, el cálculo de la misma se realiza mediante el producto de las eficiencias parciales.

Primero se presenta la salida de 12V, ver Tabla 7.4, calculada a partir de la eficiencia de la batería y del controlador. Luego en la Tabla 7.5, se muestra la eficiencia conjunta para la salida de 5V. La cual es equivalente a la de 12V agregando la etapa de regulación 12V a 5V.

Batería	85 %
Controlador	94 %
Conjunta	79.9 %

Tabla 7.4: Eficiencia en salida de 12V.

Batería	85 %
Controlador	94 %
Regulador 12V-5V	98 %
Conjunta	78.3 %

Tabla 7.5: Eficiencia en salida de 5V.

7.3. Trabajos a futuro

Como mejoras al funcionamiento de este módulo se sugiere trabajar en los siguientes puntos:

- Optimizar el consumo del equipo, ya que fue uno de los puntos débiles del proyecto. Si bien esto es ajeno al módulo, es una mejora al equipo y repercute de forma directa en este módulo.
- Medir en campo:
  - Voltaje del panel.
  - Corrientes del panel, batería y salidas de 12V/5V.
- Migrar a baterías de litio. Estas almacenan hasta tres veces más de energía que las de plomo-ácido, por lo que se puede reducir el tamaño. Por otro lado se cargan más rápidamente y tienen mayor vida útil.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 8

Servidor

El servidor es el encargado de recibir la información que envíe el módulo principal, almacenarla y desplegarla en una interfaz web. También debe permitir enviar comandos al módulo principal para configurar los módulos auxiliares, forzar medidas, u obtener la configuración actual.

8.1. Sistema Operativo

Se buscó que el sistema sea libre y replicable. Por lo cual el sistema operativo que se utilizó para tales fines es Linux. Dentro de la amplia gama de distribuciones que Linux ofrece, se ha decidido utilizar la Fedora Server.

Se buscó que los repositorios incluidos en la distribución sean de software exclusivamente libres y gratuitos, de forma que posea una gran comunidad que permita disipar las dudas que se encuentre. Y que lo económico no sea impedimento para la implementación. Se fue consciente que cuando se menciona Linux provoca cierto rechazo, por lo cual también se buscó que fuera una distribución conocida.

Todo esto se encontró reflejado en Fedora. Sin embargo, todo el software utilizado en el servidor puede ser montado sobre cualquier distribución de Linux, o sobre cualquier máquina con sistema operativo Windows o MacOS.

8.2. Comunicación

Uno de los grandes desafíos que se enfrentó al realizar este proyecto, fue encontrar una solución al problema de establecer una comunicación en la que el usuario envíe comandos a los dispositivos.

Este problema surge principalmente porque los teléfonos celulares cambian su IP cada vez que se conectan a la red. Por lo tanto, es imposible que el servidor inicie una comunicación por motus propio con los dispositivos.

En la búsqueda para solucionar este problema se consideró utilizar HTTP, o FTP. Donde el dispositivo consultara mediante polling una URL, y que si ésta tuviese información entonces la envíe al dispositivo.

Mientras se buscaba la solución se encontró el protocolo MQTT. Éste es ampliamente utilizado en soluciones de IoT.

La mayor particularidad de este protocolo orientado a conexión, es que implementa una lógica de suscripción/publicación. De esta manera permite desacoplar lo que ocurre en el servidor, de lo que ocurre del lado de los dispositivos alojados en campo. En particular permite independizarse de tener que saber quién va a recibir los mensajes.

Esto permite que el dispositivo remoto no necesite información acerca de los receptores de sus mensajes, ni siquiera cuántos reciben sus mensajes. De igual forma permite que quien va a comunicarse con el dispositivo no necesite saber su dirección.

Al ser un protocolo orientado a conexión, permite enviar mensajes desde el servidor hacia los dispositivos cuando estos últimos establecen una conexión con el primero.

Para entender mejor pongamos un ejemplo. Supongamos que hay un periodista en un medio de prensa escrita. Por lo tanto la función este periodista es escribir la noticia, y no es de su menester saber ni cuántos suscriptores tiene el medio ese día, ni dónde están las casas de esos suscriptores. Su función luego de escribir la noticia es entregarla al editor, quien recopila las noticias, y las envía a la imprenta. Luego de la imprenta, se entregan los impresos a un sistema de mensajería quienes son, ahora si, los que necesitan saber la dirección de aquellos que deben recibir el material. El periodista le pueden entregar su noticia a más de un editor, y todos los editores utilizan el mismo sistema de mensajería.

En esta analogía, quien publica MQTT es el periodista, mientras que todo el resto del proceso lo hace el broker de MQTT (servidor), los editores serían los topics. Los suscriptores de MQTT son representados por los suscriptores del diario. Las analogías no son perfectas, una gran diferencia con la analogía planteada es que eventualmente todos los suscriptores también pueden publicar.

Se debe destacar que una amplia cantidad de servidores diseñados para IoT manejan MQTT como protocolo de comunicación de datos.

Para profundizar acerca de MQTT se sugiere leer el apéndice F.

Cabe en este momento preguntarse si se hubiese podido hacer lo mismo con HTTP. La respuesta es sí. O al menos la idea general. Sin embargo, de utilizar

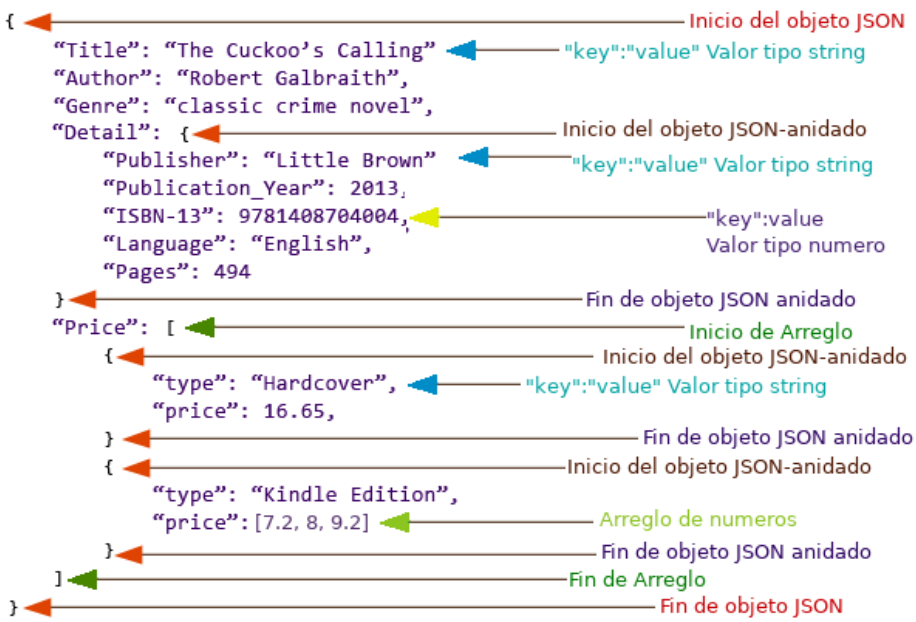


Figura 8.1: Ejemplo de la estructura de JSON

HTTP, hay que diseñar e implementar un intercambio de mensajes y una lógica, que en caso de utilizar MQTT ya está resuelto de manera eficiente.

8.3. Almacenamiento - Bases de datos

Como se mencionó en la sección anterior, existe una amplia cantidad de soluciones de servidores para proyectos de IoT. Es la norma, que éstos reciban o sugieran que los datos enviados sean empaquetados en un JSON. Se decidió emular esta estrategia. De esta forma se facilita la compatibilidad con los sistemas ya existentes.

Una decisión que surge naturalmente es utilizar una base de datos que soporte estos archivos. En este sentido se dispone de MongoDB, una base de datos no estructurada para el almacenamiento de JSON.

Al transmitir la información como JSON, se utiliza una estructura de diccionario <**key**>:<**valor**>. El valor puede ser otro diccionario, un vector, o un JSON anidado. Un ejemplo de JSON se puede ver en la figura 8.1.

Dentro de la versatilidad de esta base de datos se encuentra que no se requiere estructura previa. El JSON recibido en cada mensaje puede contener tanto campos distintos, como distinta cantidad de campos.

Los servidores existentes que exigen que los datos sean enviados como JSON,

es común que exijan que las medidas sean enviadas en un JSON anidado, cuyo valor de **key** sea **data**. Un ejemplo de JSON anidado es el campo *Detail* en la figura 8.1. Como se buscó tener compatibilidad con estos servidores, y siguiendo la lógica planteada previamente, se decidió utilizar esa misma estructura.

Tener este campo facilita el procesamiento del mensaje, ya que permite saber exactamente donde está la carga útil del mensaje. En este caso se entiende que la carga útil son los datos adquiridos.

A su vez es importante poder tener referencias temporales, por lo cual un key del JSON se denominó **timestamp**, y su valor es el **Epoch** [30] del momento en que el módulo principal recibió la medida.

MongoDB tiene un campo especial llamado **_ID** con el cual ordena la base de datos. Si este campo no es detallado, cuando se quiera guardar el mensaje, el sistema lo auto-completará. Se tomó la decisión de que este campo se complete con el mismo valor que **timestamp**.

En esta base de datos se almacenan todos los mensajes que llegan desde los dispositivos. Los mensajes pueden ser: mensajes con medidas, mensajes con alertas, mensajes de confirmación en los cambios de configuración, etc.

Para la representación en gráficas se decidió que lo mejor es utilizar una base de datos específica para series temporales. Dentro de estas bases de datos, *InfluxDB* es una de las soluciones más utilizadas, puesto que está optimizada para este uso. Esta base de datos es de código abierto y desarrollada por *InfluxData* en el lenguaje *Go*.

8.4. Inferfaz Web

Para la interfaz web se presentan tres Web Services de los cuales se eligieron dos. Todos tienen características complementarias.

8.4.1. MongoKU

Este Web Service permite desplegar en un navegador las bases de datos MongoDB, lo cual es de utilidad para poder ver los mensajes recibidos. No solo los que contienen medidas, sino también las alertas y las propiedades. Se puede navegar dentro de este Web Service buscando los mensajes que contengan cierta key, o cierto <key>:<value>. En particular, se puede filtrar si es una alerta, un cierto timestamp, o un _ID (en nuestro caso lo mismo que filtrar por timestamp).

8.4.2. Grafana

Grafana cuenta con soporte de integración para un amplio abanico de bases de datos y aplicaciones. Presenta una interfaz gráfica limpia y sencilla, que permite múltiples formas de representar los datos, e incluso permite tomar acciones con ellos.

Grafana admite más bases de datos que InfluxDB, por ejemplo SQL o PosgresQL. También se pueden conectar directamente algunos recolectores como es el caso de Telegraf o Graphite.

Es posible exportar los datos a un CSV, realizar scripts para tomar acciones o analizar los datos, entre otras opciones.

8.4.3. Chronograf

Chronograf es de la empresa InfluxData, la cual desarrolló InfluxDB. Es software libre, y permite gestionar las bases de datos de este tipo desde el mismo Web Service. La contra es que es compatible únicamente con bases de datos InfluxDB. Al igual que Grafana, permite distintas representaciones gráficas de los datos, y exportarlos a un CSV, además de tomar acciones con ellos.

Este Web Service se puede utilizar en conjunto con el resto de los productos de la compañía, tales como Telegraf para la centralización de los datos, o Kapacitor para el análisis. Éste es el web service seleccionado para la representación gráfica de los datos, en pos de brindar un único entorno de trabajo para el usuario.

8.5. Broker y cliente MQTT

En este servidor se instaló el broker para el protocolo MQTT, el cual gestiona los mensajes que se reciben desde el módulo principal y desde el cliente que utilice el usuario para enviar comandos. Los entregará a los clientes que estén suscriptos a los topics correspondientes. También se instaló un cliente, para poder tomar acciones desde el servidor y enviar mensajes al módulo principal. En ambos casos, el software que se utiliza es desarrollado por *Eclipse* y se llama *Mosquitto*. Esta opción es elegida por ser software liviano y libre.

8.6. Cliente SARA-Bee

Una decisión muy importante que se tomó fue que a nivel de campo los módulos no tuvieran conocimiento de la magnitud adquirida. Dejando el etiquetado de las magnitudes en mano de quien diseña el servidor.

Todos los lenguajes utilizados hoy en día tienen una librería que implementa un cliente de MQTT. La empresa Eclipse, responsable de la liberación de este protocolo, ha desarrollado una familia de clientes llamados **PAHO** [27]. Estos clientes

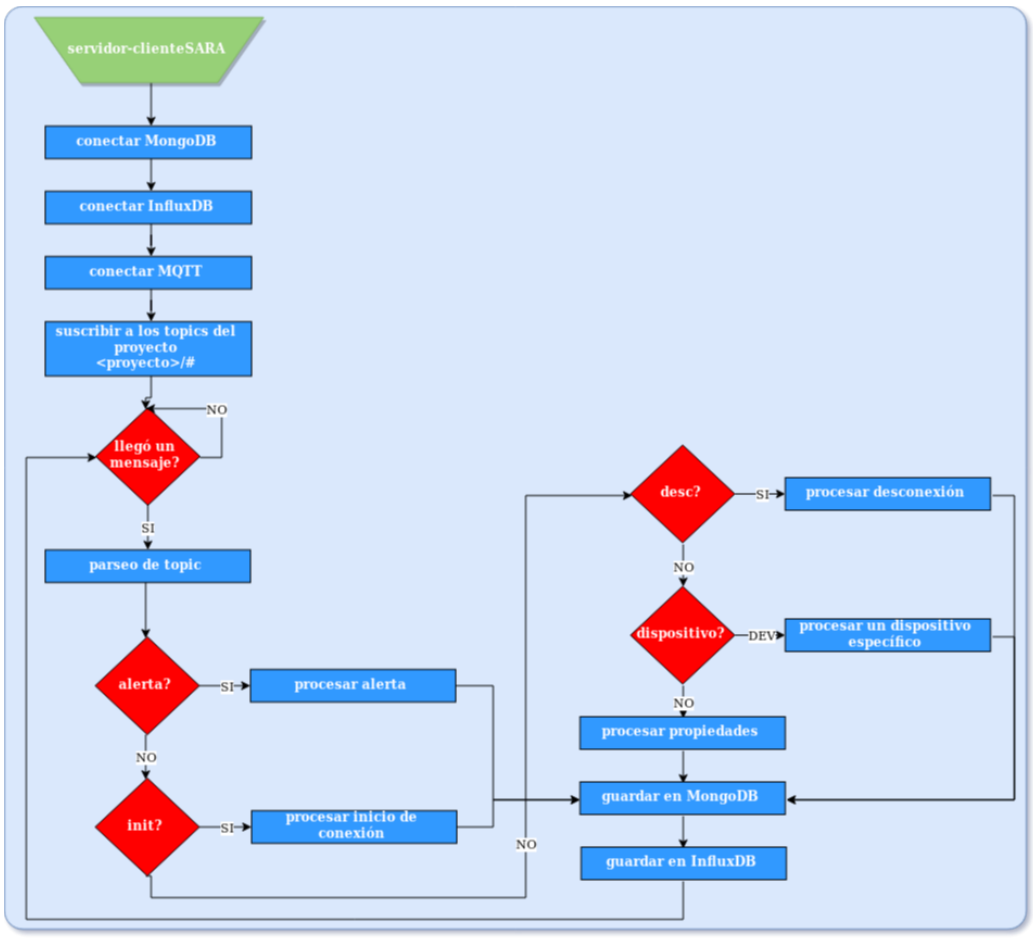


Figura 8.2: Flujo del cliente SARA

.....

Se decidió crear un servicio de Systemd para Linux que es un script de Python al inicio del sistema.

La figura 8.2 muestra el diagrama de flujo para este script.

Como se observa en la figura 8.2, primero se establece una conexión con la base de datos MongoDB, la base de datos InfluxDB, y se suscribe a todos los topics del proyecto seleccionado mediante el uso de la librería PAHO para Python. En esta documentación, el nombre del proyecto es SARA y el dispositivo a modo de ejemplo es SARABEE.

Cada vez que llega un mensaje en un canal de SARABEE primero se estudia el topic en el cual se recibió el mensaje. Se distingue si es un mensaje con las propiedades del sistema, o si es otro tipo de mensaje. Si es un mensaje de propiedades se guarda el mensaje en la base de datos MongoDB en una colección especifica para

las propiedades, y se espera al próximo mensaje.

Si no es un mensaje de propiedades, es de interés saber si es alguno de los siguientes mensajes:

- Mensaje de conexión establecida
- Mensaje de desconexión
- Mensaje con una alerta del sistema
- Mensaje con una medida

Los mensajes recibidos de estas cuatro categorías se almacenan en la misma colección de MongoDB. Los primeros tres casos son mensajes del sistema. Son de gran utilidad si se desea mostrar en los Web Services el estado de las conexiones, por ejemplo mediante un código de colores. De momento, únicamente se almacenan en la base de datos MongoDB.

En caso que sea una medida, utilizando el topic se obtiene el número de módulo, y el número de canal. Con esto se puede etiquetar las medidas. Las bases de datos InfluxDB reciben un JSON, el cual tiene varios JSONs anidados. Hay dos que nos son de interés. Uno es *tags*, donde se indica el nombre del dispositivo, y otro JSON anidado cuya key es *measurements* donde se indica <nombre de la medida>:<medida recibida>. De esta forma, en los Web Services se filtra primero por nombre del dispositivo y luego por el nombre de la medida.

8.7. Pruebas y Validación

Lo primero que se comprobó fue el éxito de la instalación del broker Mosquitto de MQTT. Para ello se instaló el broker y el cliente de Mosquitto desde el repositorio de Fedora. Se suscribió localmente desde el cliente a un topic, y también localmente se realizaron publicaciones un el topic seleccionado. Al realizar esto, en la pantalla de la computadora se podía observar los mensajes recibidos.

Se realizó el mismo procedimiento con distintos topics. Luego se utilizaron distintas computadoras dentro de la red, una con un cliente suscrito y otra con un cliente publicando. Posteriormente se instaló un cliente de DDNS que permitió conexiones desde el exterior de la red realizando las mismas pruebas. Desde este momento y en adelante el cliente utilizado para suscribirse fue un smartphone Android.

Una vez verificado que era posible publicar y suscribirse en el broker, se realizaron pruebas donde se verificó que era posible la suscripción y la publicación desde módulos principales. Verificar que el módulo principal se suscribía fue explicado en el capítulo 6, dedicado a dicho módulo.

Para validar la escritura en las bases de datos InfluxDB y MongoDB se utilizó la terminal de Linux empleando los comandos correspondientes a la escritura, y luego utilizando los clientes de terminal que brindan ambas soluciones, para verificar la lectura.

Para probar el correcto funcionamiento del script en su completitud, primero se comprobó que el cliente de la librería PAHO de MQTT para Python se conectó y recibió mensajes del broker. Para ello simplemente se ejecutó el segmento de código de la suscripción a topics, y en la función que se ejecuta cuando recibe un mensaje lo único que se hizo fue imprimir en pantalla el nombre del topic en cual se recibió, y el contenido del mensaje. Los mensajes fueron desde clientes de MQTT alojados en otras computadoras o smartphones.

Una vez verificado esto, se procedió a a verificar que se podía escribir en la base de datos MongoDB. Para ello cada vez que se recibía un mensaje se lo insertaba en la base de datos. Para verificar que esto efectivamente se realizaba, se accedió a la base de datos desde la terminal y se vieron los mensajes.

A continuación se hizo lo propio con la base de datos InfluxDB. Para ellos se siguió el mismo sistema. Los mensajes recibidos se parsearon y se insertaron en la base de datos. Al igual que en MongoDB, la verificación se realizó desde la terminal de Linux.

Una vez validados estos aspectos, se realizó una verificación del script completo siguiendo la misma lógica.

En este punto se tenía certeza del correcto funcionamiento del script y se pasó a confirmar el uso de los Web Services. Estos fueron instalados en la configuración por defecto. Se habilitaron las políticas de seguridad necesarias para aceptar conexiones en los puertos requeridos. Posteriormente la verificación del uso se realizó simplemente primero verificando que las bases de datos detectadas y los campos eran correctos, y posteriormente que los datos desplegados también lo fueran.

Un ejemplo de la visualización del Web Service Chronograf se puede observar en la figura 8.3, es una captura de un dashboard creado donde se observa el monitoreo de peso, humedad, temperatura y el voltaje en la batería. En la figura8.4 se observan las bases de datos InfluxDB disponibles, las cuales pueden ser seleccionadas para desplegar en gráficas, tablas u otras representaciones. Una vez seleccionada la base de datos, también se puede observar como generar la consulta a la base de datos de forma gráfica.

El servidor se desarrolló en etapas y en paralelo al resto de los módulos del sistema siendo finalizado previo al resto. Por lo cual finalizada esta etapa, se continuó recibiendo mensajes y desplegándolos durante el desarrollo de los módulos, principalmente durante las pruebas en campo. En el tiempo remitido se encontra-

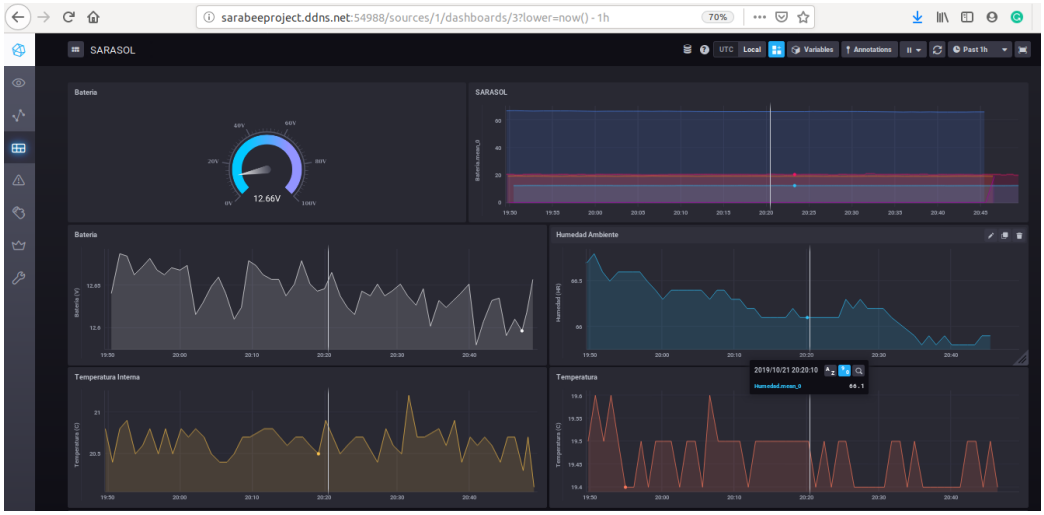


Figura 8.3: Dashboard de representaciones en Chronograf

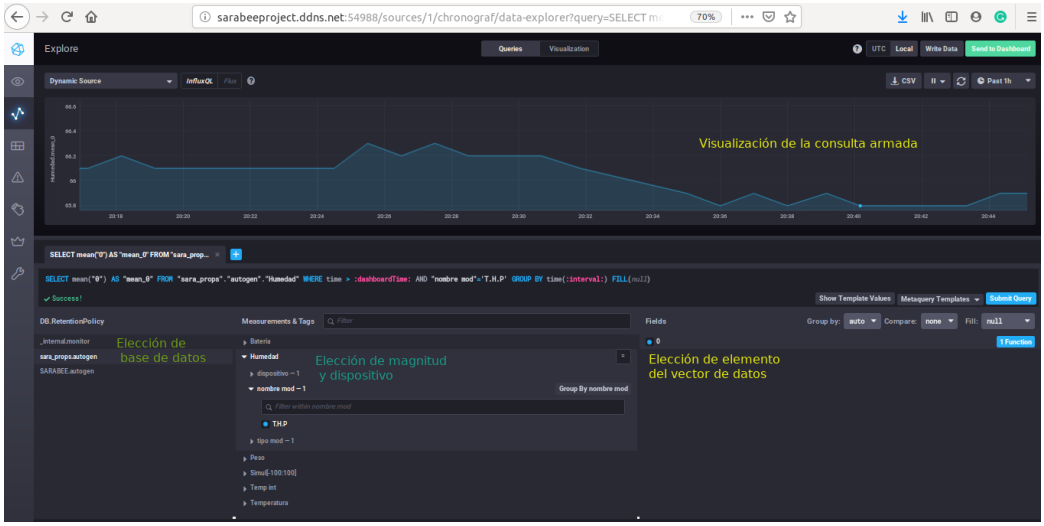


Figura 8.4: Selección de la base de datos y armado de la consulta en Chronograf

ron y corrigieron pequeños errores que en ningún momento afectaron directamente al funcionamiento ni al desempeño.

8.8. Trabajos a futuro

Una línea a profundizar es el trabajo con los datos adquiridos, y desarrollar acciones ante las alertas o datos corruptos.

A lo largo del proyecto se encontraron distintas alternativas para solucionar el problema y hubieron particularmente dos software que quedaron postergados para su estudio e implementación. Estos son Telegraf y Kapacitor. Ambos son de la *suite* de InfluxDB. Se entiende que sería bueno su implementación en el futuro

cercano ya que permitirían contar con un sistema receptor con productos que están pensados para trabajar juntos, mejorando así la eficiencia. Sumado a esto, utilizar estas soluciones de software libre permite disminuir posibles errores y contar con la comunidad que tienen detrás.

En líneas generales se entiende que este producto podría ser ofrecido como terminado, utilizando empaquetados como Doker, donde al usuario final se le entregase una imagen con todo el software instalado y configurado, donde únicamente realice los cambios necesarios para su aplicación particular en el cliente SARA-Bee.

A su vez se entiende que faltó proponer soluciones para el cliente SARA-Bee en otros lenguajes, como por ejemplo Octave.

Capítulo 9

Sistema completo

En este capítulo se realiza un análisis y reflexión del sistema completo.

9.1. Pruebas en campo

Existieron dos instancias en donde se llevaron a cabo pruebas de campo. La primera prueba se realizó en la azotea de Facultad, donde se dejó SARABEE una semana relevando datos. Se constataron los ciclos de carga durante el día, y de descarga de la batería en la noche. La energía absorbida por los paneles fue tal que evitó que el equipo se apagara en ausencia del sol. El equipo se encontró transmitiendo de forma continua al servidor. Se relevaron las medidas de humedad, temperatura interna y externa, y voltaje de batería. Día a día se constató que las medidas reportadas fueran las reales.

La segunda prueba de campo se realizó en Villa del Rosario, en el departamento de Lavalleja, a 23km de Minas. En el sitio habita el apicultor José Ferrando, el cual brindó su conocimiento y una de sus colmenas en donde instalar el equipo. . Debido a la lejanía, y a la imposibilidad de dejar el equipo *in situ*, se relevaron las medidas durante unas pocas horas. Este tiempo alcanzó para constatar los valores obtenidos, se observan en la tabla 9.1. En la figura 9.1 se presenta material fotográfico de ésta actividad

En esta etapa de pruebas se destaca la exitosa comunicación del equipo con el servidor. En el sitio donde se relevaron los datos el nivel de señal para tecnologías 3G y 4G era nulo. Sin embargo, se encontró señal GPRS, lo cual permitió establecer una conexión. Desde la casa del productor se visualizaron los datos adquiridos en tiempo real.

La temperatura del MA fue relevada por el sensor ubicado dentro de la caja estanca, donde se encontraba recibiendo el sol directo, y se observó el aumento de esta medida con el transcurro de los minutos. El voltaje de la batería se encontró en carga máxima, ya que los paneles estaban recibiendo la suficiente energía solar.

Magnitud	Valor promedio relevado
Temperatura MA [°C]	38.5
Temperatura colmena [°C]	36.0
Humedad relativa [%HR]	50.0
Peso colmena [kg]	41.5
Voltaje batería [V]	13.8

Tabla 9.1: Datos relevados en colmena de abeja

La colmena de estudio estaba conformada por entre 30.000 y 40.000 individuos, por lo que el apicultor constató que el valor del peso adquirido es coherente.

Se colocaron tres micrófonos dentro de la colmena de abejas, y se obtuvieron medidas cada 10 minutos. Los datos crudos se almacenaron localmente en la tarjeta SD del MASV. Se procesaron estos datos, y se encontró que la señal de interés era muy débil. Se percibió mucho ruido, lo que hizo difícil identificar el batido de las abejas. Esto se cree que fue debido a las siguientes variables. En primer lugar, existió una mala colocación de los micrófonos, no se consideró que pudiera quedar bloqueada la membrana del micrófono con los propios paneles de la colmena. La amplificación se encontraba seteada en 100V/V, y se concluyó que en estas condiciones se debería configurar una ganancia mayor para fortalecer la señal de estudio. Por último, para esta instancia de pruebas, se fabricaron cables más extensos para los micrófonos, con el fin de abarcar una distancia mayor. Esto último sumado a las interconexiones de la caja estanca exterior, y las interconexiones internas en el MASV generó aún más ruido de lo esperado.

9.2. Análisis holístico

Desde el inicio del proyecto, la visión de los autores con respecto al equipo era: adquirir, procesar y transmitir cualquier tipo de medida, sin importar su naturaleza. Esto se trasladó al diseño e implementación, tomándolo como un concepto y no como una aplicación particular.

El proyecto pretende sentar una base para crear un sistema que provea soluciones para diversos problemas con características similares: adquisición de datos *in situ* con eventual procesamiento; autonomía energética; transmisión remota de datos, y almacenamiento e interfaz gráfica en servidor.

La arquitectura elegida permite equilibrar la complejidad de funcionamiento, fraccionando las tareas en los distintos módulos.

El diseño de la arquitectura tuvo como pilares los siguientes puntos:

Figura 9.1: Imágenes de la prueba del equipo en campo.

1. **Modularidad**: esto no solo implica dividir el equipo en módulos, sino que estos deben ser totalmente independientes.

2. **Transparencia**: el módulo principal es independiente de la cantidad de módulos, de la cantidad de canales, del largo de cada canal y de la medida en particular. Para la medida sólo se requiere que pueda ser representada en punto flotante. Cada módulo esclavo almacena información que lo caracteriza pero únicamente el módulo maestro es el que sabe gestionarla, esto se traduce a versatilidad y escalabilidad. Le permite al sistema medir en distintos ecosistemas cualquier variable, escalar o vectorial, en un extremo y asignarle una etiqueta que lo caracterice en el otro extremo, el servidor.

3. **Reciclable y de bajo costo**: Gran parte de las decisiones se centraron en este punto. Como proyecto de grado, además de nuestro aprendizaje, se buscó implementar un sistema con potencial para ser reutilizable por los tutores, futuros estudiantes o quien guste.

9.3. Trabajo a futuro

Evaluando el equipo de forma global.
Con respecto al protocolo:

- Agregar a los canales *tags* en lenguaje natural.

- Implementar comandos comandos específicos para el manejo de actuadores, tales como motores.

Con respecto al equipo:

- Sistema de seguimiento de luz en los paneles solares, para que se dispongan siempre en posición óptima respecto al sol, y recibiendo la máxima energía posible. Se debe evaluar si trae un beneficio significativo realizar esta tarea.
- Mejoras de seguridad en la comunicación sobre MQTT. Requiere un hardware más potente, pero sin salir de las soluciones de IoT, como por ejemplo Raspberry PI.
- Integrar algoritmos de predicción para la gestión de la energía y el manejo de la agenda.
- Permitir agenda con fecha y hora.

Con respecto al servidor:

- Integrar herramientas de gestión de eventos.
- Proponer una solución tipo Docker[1] [11].
- Integrar el envío de mensajes desde la misma interfaz web.

[1]Docker proporciona una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.

Capítulo 10

Gestión de proyecto

En este capítulo se realiza un análisis y reflexión breve de la gestión del proyecto. En particular se hace hincapié en los costos, riesgos y tiempos de ejecución; temas analizados previo al inicio del proyecto.

10.1. Análisis de Costos

Para realizar la estimación inicial del presupuesto total del proyecto se separaron por un lado los costos debido a materiales, y por otro el costo que implicaría las horas/estudiante, es decir la mano de obra del proyecto. Los ingresos del proyecto, fueron del Departamento de Sistemas y Control, del Instituto de Ingeniería Eléctrica de la Universidad de la República (UdelaR).

10.1.1. Costos de materiales

Los costos de materiales se presentan para cada módulo. Y en el entendido de la producción a pequeña escala.

MA

En la tabla 10.1 se despliegan los gastos para los materiales necesarios para este módulo.

Item	**Costo (U$S)**	**Unidades**	**Total (U$S)**
Arduino UNO	15	1	15
Fabricación *shield*	5	1	5
Controlador	25	1	25
Componentes	5	1	5
Sensor DHT11	3	1	3
Cables y conectores	5	4	20
Caja estanca	6	1	6
Total (U$S)			79

Tabla 10.1: Costos de Materiales del MA

Los componentes son:

- Resistencias SMD.
- Pin headers hembra y macho
- Conectores para el shield, arandelas, tuercas y tornillos.

MASE

En la tabla 10.2 se despliegan los gastos para los materiales necesarios para este módulo.

Item	**Costo (U$S)**	**Unidades**	**Total (U$S)**
Arduino UNO	15	1	15
Fabricación *shield*	5	1	5
Componentes	5	1	5
Sensor DHT22	20	1	20
Módulo HX711	3	1	3
Celdas de carga	3	4	12
Cables y conectores	5	4	20
Caja estanca	6	1	6
Acrílico de la balanza	30	0	0
Total (U$S)			86

Tabla 10.2: Costos de Materiales del MASE

Los componentes son:

- Pin headers hembra y macho
- Conectores para el shield, arandelas, tuercas y tornillos.

MASV

En la tabla 10.3 se despliegan los gastos para los materiales necesarios para este módulo.

Item	**Costo (U$S)**	**Unidades**	**Total (U$S)**
Arduino MKR Vidor 4000	75	1	75
Fabricación *shield*	115	1	115
Componentes	50	1	50
Micrófonos	1	4	4
Conectores para micrófono	1	4	4
Módulo microSD	4	1	4
Shift Level Converter	12	1	12
Cables y conectores exterior	5	4	20
Caja estanca	8	1	8
Total (U$S)			292

Tabla 10.3: Costos de Materiales del MASV

El costo de la fabricación en este módulo fue mayor debido a que para fabricar un shield son necesarios más PCBs, el tamaño de uno de ellos era el doble que los de los otros módulos y el PCB de los filtros de adquisición tenía 4 capas.

Los componentes son:

- Componentes SMD: resistencias, capacitores y amplificadores diferenciales.
- DIP switches.
- Pin headers hembra y macho
- Conectores para el shield, arandelas, tuercas y tornillos.

MP

En la tabla 10.4 se despliegan los gastos para los materiales necesarios para este módulo.

Item	**Costo (U$S)**	**Unidades**	**Total (U$S)**
Arduino MEGA	30	1	30
Fabricación *shield*	15	2	30
Componentes	30	1	30
Módulo SIM800	12	1	12
Módulo microSD	12	1	12
Cables y conectores	5	2	10
Caja estanca	7	1	7
Total (U$S)			121

Tabla 10.4: Costos de Materiales del MP

Los componentes son:

- Componentes SMD: resistencias, capacitores y reguladores lineales.
- DIP *switches*.
- *Pin headers* hembra y macho.
- Conectores para el *shield*, arandelas, tuercas y tornillos.

Este fue el único módulo para el cual hubo una etapa de refabricación de PCB, por lo que en cantidad se cuantifica doble.

Gastos adicionales

Durante el desarrollo surgieron gastos que no estaban contemplados en la planificación, éstos fueron parte del desarrollo del prototipo del proyecto por lo que se los agrega. Éstos son:

- Componentes dañados.
- Componentes no utilizados por cambio en el diseño.
- Materiales para la soldadura: Estaño, flux, malla de desoldadura, etc.

Se hace una estimación *grosso modo* de U$S 100 para estos gastos.

Equipo

Item	Costo (U$S)	Unidades	Total (U$S) (Proyecto)	Total (U$S) (Equipo nuevo)
MASE	86	1	86	86
MASV	300	1	300	300
MP	121	1	121	121
MA	79	1	79	79
Batería	30	1	0	30
Paneles solares	60	2	0	120
Reparación MA de PestiBee	30	1	30	30
Gastos adicionales	100	1	100	100
Cables	15	1	15	15
Conectores exteriores	5	12	60	60
Caja estanca exterior	100	1	0	100
Total (U$S)			791	1041

Tabla 10.5: Costos de Materiales del equipo

Los materiales cuyo total se presenta como cero en *Total (Proyecto)* son materiales necesarios para el desarrollo del prototipo del equipo pero fueron que donados o heredados de proyectos anteriores, en su mayoría provenientes del PestiBee.

La estimación inicial del costo fue de U$S 375, poco menos de la mitad de lo que realmente se gastó. Los motivos de esa enorme diferencia se deben a la inexperiencia de los integrantes a presupuestar proyectos de esta magnitud.

Los errores más grandes en la estimación fueron:

- No se tuvieron en cuenta los conectores, que cada uno de ellos vale U$S 5 y su costo fue superior a los U$S 100
- No se tuvo en cuenta los gastos adicionales que sumó U$S 100 a los gastos.
- Para el MASV se eligió una plataforma de mayor valor y se realizaron varios PCB incluyendo uno de cuatro capas.

10.1.2. Mano de obra

La mano de obra no es remunerada debido a que el proyecto es con fines educativos. De todas formas, se realiza una estimación previa y se contabilizan las horas totales dedicadas para tener una idea de la magnitud que ocupa dentro de un presupuesto.

Estimación Inicial

La estimación del costo de las horas estudiante se realizó mediante un cálculo aproximado, teniendo en cuenta la remuneración salarial de cada de uno de los integrantes del grupo. La cifra que se obtuvo fue U$S 10 por hora. La cantidad de horas estimada para el proyecto es de 1560 horas a ejecutar entre los todos los integrantes del grupo, lo que estima un total de U$S15660 que equivale a $ 501200 utilizando la cotización a $ 32[1] para el dolar americano.

[1]Precio aproximado de compra del dolar americano en Agosto 2018

Cantidad Final

Las horas dedicadas al proyecto fueron contabilizadas de forma individual y se obtuvo una cantidad total de 2595 horas entre los tres integrantes.

Usando el mismo valor de costo para las horas que al inicio (U$S 10) da un total de U$S25950 lo cual equivale a un total de $ 960150 utilizando una cotización de $37[2] para el dolar americano.

10.1.3. Presupuesto total

Finalmente, el presupuesto total se calcula en la tabla 10.6, siendo éste la suma de los materiales y la mano de obra.

Item	**Total (U$S)**
Materiales	791
Mano de obra	25950
Total (U$S)	**26741**

Tabla 10.6: Presupuesto total

La observación más notoria es la relación entre el costo de los materiales y la mano de obra.

10.2. Análisis de Riesgos y Tiempos

Al iniciar el proyecto se realizó una estimación de los posibles riesgos que podrían desencadenar el fracaso parcial o total del mismo. Los cinco mayores riesgos estimados inicialmente fueron:

1. Daño en el equipo o robo al dejarlo a la intemperie.

2. Estimación errónea de la capacidad del módulo de alimentación.

3. Módulo principal no logra comunicarse con demás módulos.

4. Módulo principal no logra comunicarse con servidor.

5. Placa con errores de diseño y necesidad de re-fabricar.

En la Tabla 10.7 se muestra la evaluación realizada para dichos riesgos.

[2]Precio aproximado de compra del dolar americano en Noviembre 2019

Riesgo	Probabilidad de ocurrencia	Nivel de impacto
1	Poco probable	Extremo
2	Poco probable	Extremo
3	Poco probable	Medio
4	Poco probable	Medio
5	Poco probable	Bajo

Tabla 10.7: Evaluación de 5 riesgos principales.

Si bien hubo necesidad de solicitud de prórroga, durante el desarrollo el equipo no se vió afectado por ninguno de estos riesgos. Para el riesgo 1 se tomaron como precauciones dejar el equipo en lugares con acceso restringido: Laboratorio de Proyecto inicialmente y techo de facultad durante las pruebas.

Para el riesgo 2 se hizo una sobreestimación de la potencia necesaria del módulo de alimentación.

Para los riesgos 3 y 4 se realizaron pruebas exhaustivas y se lograron resultados satisfactorios en cuanto a comunicación.

Para el riesgo 5, si bien hubo una re-fabricación de la placa del módulo principal, estaba fuera del alcance y la motivación era dejar como legado este módulo libre de errores para futuros proyectos.

Se debe considerar que estaba latente un sexto riesgo, para el cual se tomaron medidas preventivas desde el primer momento. El riesgo en cuestión concierne al módulo de alimentación. La mejor opción era poder recuperar el módulo fabricado por el proyecto Pestibee. Dado que éste módulo fallaba, se importaron los componentes necesarios para su reparación y un módulo que lo supliera. Fue gracias a la acertada decisión de adquirir éste segundo que se desarrolló el proyecto con tranquilidad y finalmente se reparó el módulo de Pestibee.

Los mótivos que generaron retrasos en las tareas se debieron a subestimar el nivel de impacto en ciertas actividades. En particular las siguientes generaron retrasos considerables en el avance del proyecto:

- Elección de plataforma de desarrollo para el MASV: Inicialmente se consideró Raspberry Pi pero el grupo optó finalmente por una plataforma con FPGA, lo cual desencadenó en una serie de dificultades no previstas que llevaron a retrasos en las tareas de uno de los integrantes obligando a realizar cambios grandes en el cronograma.

- Elección de la arquitectura de firmware del MP: como se explica en dicha sección inicialmente se tenía previsto usar RTOS y se comenzó a desarrollar software para esa arquitectura. Por problemas con librerías no se pudo implementar correctamente y se tuvó que reevaluar y migrar a una arquitectura más simple, lo cual conllevó un proceso lento no considerado al principio.

La participación en *Ingeniería de Muestra* fue otra actividad que se inetrgó en el cronograma del proyecto en la última etapa. Si bien no fue motivo de retrasos, no había sido considerada inicialmente y no se había dimensionado el esfuerzo y tiempo que implica participar de esta actividad hasta algunas semanas antes.

El grupo participó de la intancia realizada en Paysandú los días 13 y 14 de Septiembre del 2019 , y también de la instancia realizada en Montevideo del 10 al 12 de Octubre del mismo año, dedicándole una semana de preparación a cada una de ellas.

En la Figura 10.1 se muestra la dedicación mes a mes durante el proyecto de los tres integrantes.

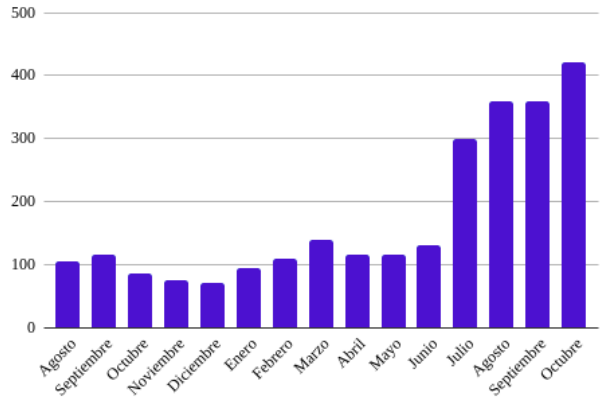


Figura 10.1: Dedicación mensual grupal durante el proyecto.

Capítulo 11

Conclusiones

En este capítulo se realiza una reflexión y evaluación del proyecto realizado y se sugieren futuras correcciones o aplicaciones del sistema.

11.1. Conclusiones generales

En líneas generales, el primer aspecto a remarcar es que se alcanzaron los objetivos planteados al inicio.

Se diseñó un sistema que acopla un equipo de adquisición modular con un servidor. No solo se logró que dicho sistema funcione correctamente, sino que alcance las características de diseño planteadas.

Para el equipo:

- Compuesto por módulos independientes que se comunican a través de un protocolo *ad hoc*.
- Permite adquirir señales escalares, en particular humedad relativa, temperatura y peso.
- Permite adquirir y procesar señales vectoriales, en particular audio.
- La adquisición se puede solicitar por UART conectando el módulo via USB a un PC.
- Almacenamiento local de los datos adquiridos y procesados.
- Autonomía energética.
- Comunicación con el servidor en ambos sentidos: envía lecturas y alertas y recibe parámetros de su configuración.

Para el servidor:

- Centralizador de mensajes.

- Almacenamiento de todos los mensajes enviados por el equipo
- Almacenamiento en series temporales
- Acceso vía Web Services a los datos almacenados
- 100 % Software Libre

El desarrollo del proyecto implicó integrar múltiples disciplinas dentro la Ingeniería Eléctrica. Desde potencia en el manejo del panel solar, electrónica analógica en el diseño de circuitos y *shields*, pasando por la electrónica digital en la programación del firmware; hasta telecomunicaciones para lograr la transmisión de datos al servidor.

El proyecto no solo sirve como prueba de concepto, sino que sentó las bases para seguir trabajando y alcanzar un producto final.

11.2. Evaluación de los resultados

11.2.1. Plataforma de desarrollo

De las múltiples plataformas adaptables a las características del proyecto, se optó por trabajar con la plataforma de Arduino. Constatamos que fue una buena elección por contar con una gran comunidad de desarrolladores, por ser económicas, de código abierto, por contar con librerías que faciliten algunos puntos, y por ser fácilmente programable a través de la IDE desde cualquier sistema operativo. Se decidió incluir en el proyecto una tecnología nueva, la última placa liberada por Arduino, que incluye un FPGA. Éste fue un desafío grande, que si bien sumó problemas críticos durante el transcurso del proyecto, se encontraron soluciones y se concluye que dejó aprendizajes valiosos en cómo pararse frente a una tecnología nueva, con poca documentación, y prácticamente nula repercusión en la comunidad de desarrolladores.

11.2.2. Bus de datos y alimentación

El bus implementado fue un requerimiento del proyecto, que además de satisfacer las especificaciones de SARA-Bee, se considera de propósito amplio. Las líneas de alimentación de 12V y 3.3V si bien se pueden prescindir en nuestro proyecto, pueden ser de gran utilidad en otras aplicaciones. De los 7 pines que conforman el Bus, hay un pin libre, que se sugiere podría utilizarse como segunda tierra, y generar circuitos con tierras aisladas. Finalmente, se destaca el protocolo creado, que cumple con las especificaciones de diseño planteadas. Las principales ventajas del protocolo: evitar la manipulación del módulo principal al conectar/desconectar módulos esclavos (debido a la inicialización diseñada), y posibilitar tener cuantos módulos esclavos le permita I2C (127). Para concluir, se alcanza un protocolo de comunicación a través del bus implementado que cumple con los objetivos planteados.

11.2.3. Comunicación remota

GPRS resultó ser una tecnología que se adaptó correctamente a los requerimientos del proyecto. En el mercado local nunca se encontró falta del hardware requerido .
Utilizar correctamente el módulo llevó una curva de aprendizaje, empezando por el entendimiento de los comandos AT involucrados, pasando por la búsqueda de una librería completa y funcional, y solucionando problemas de alimentación. Una vez superada las dificultades de la alimentación no se encontró dificultades para la conexión a la red en los lugares de prueba.

11.2.4. Monitoreo del sistema

Para monitorear el sistema se diseñó un servidor basado en software libre, sobre el cual corren Web Services para permitir al usuario visualizar la información recibida.
No se constataron problemas en la visualización y la creación de consultas a las bases de datos, así como el manejo de las mismas resultó satisfactorio.
Queda aún por delante la integración con herramientas de análisis de datos, en particular con Kapacitor de forma de hacer más efectiva la detección de funcionamiento incorrecto del equipo o de actividad anormal del equipo en campo.

11.3. Desafíos y aprendizaje

11.3.1. Gestión de proyecto

En primer lugar, cabe destacar la experiencia y aprendizaje que brinda la auto gestión del proyecto. Esto, por contrapartida, lleva a qué por ejemplo, existan grandes márgenes de error entre el tiempo y costos estimados, con los datos reales. Este proyecto se nos presentó sin muchas limitantes, dejando a los autores la libertad de tomar el camino de mayor interés. Esto implicó tomar decisiones bajo un gran abanico de posibilidades, como lo fue elegir la plataforma, las diversas placas posibles, los componentes que acompañan cada placa (módulo micro SD, módulo GPRS, conversores de nivel de voltaje, micrófonos, sensores de humedad y temperatura, step-down, conversor DC-DC, etc.), la elección de qué arquitectura de software llevar a cabo, y cómo implementarla, entre otras. Todas estas decisiones, algunas más sencillas, otras de mayor complejidad, implicaron un insumo de tiempo que no fue previsto en la planificación inicial del proyecto, y con las que se tuvo que ir trabajando para lograr el alcance propuesto del proyecto. Hoy se pueden sacar conclusiones y aprendizajes de cómo pararse frente a la planificación de un proyecto de esta magnitud.

11.3.2. Desarrollo de firmware

La elección de la plataforma significó que la programación se realice en código C a través de la IDE de Arduino. Si bien mencionamos sus ventajas, cabe destacar

que la depuración del código en Arduino puede volverse muy tediosa. La complejidad del código implementado para cumplir con las especificaciones diseñadas del equipo, implicó una depuración difícil de llevar a cabo, y destacó el ingenio para corregir los bugs y así alcanzar la funcionalidad deseada en el código .

11.3.3. Diseño de hardware

Llevar adelante un proyecto que implique un fuerte pilar en Hardware, no es algo que se adquiere en el paso por Facultad, y por lo tanto, aparece una innumerable cantidad de situaciones desconocidas, que brindan conocimientos y enseñanzas en este nuevo campo. Para la fabricación de los PCB se utilizó el software libre *Kicad*, similar al *Eagle* que se emplea en los cursos de Electrónica, e implicó involucrarse con gran profundidad en esta herramienta para lograr los diseños deseados. Los *shields* fabricados, si bien tienen meramente un circuito sencillo, conllevan la creación de las huellas de los componentes en el Software, que es un trabajo que requiere paciencia y tiempo. La placa de acondicionamiento para las señales del MASV es de 4 capas, éste fue un trabajo complejo que consumió gran insumo de tiempo.

Por otro lado, el ensamblado de los módulos, cada uno en una caja estanca de acuerdo a su tamaño, con las perforaciones para los conectores de chasis, el propio armado de los conectores y de los cables del bus, la fijación de las placas dentro de las cajas, fue un trabajo de ingenio y práctica del que, sin duda, se obtuvieron aprendizajes y experiencia.

11.3.4. Compras en el exterior

Tanto la fabricación de los PCB, como la compra de muchos de los componentes, se realizaron en el exterior. Esto es un punto no menor, ya qué se suman variables que pueden ser perjudiciales para un proyecto como éste, donde hay que cumplir sí o sí los plazos de tiempo estimados. Las compras en el exterior implican conocer la regulación de la Aduana, los trámites que se deben realizar previo al arribo del paquete, el precio del flete, y el precio de piso en caso de quedar retenido en Aduana. Implica también el estudio profundo previo a la compra, ya qué, si lo que recibo no es en verdad lo que necesitaba, es un gasto de dinero y tiempo que no se recupera, y que seguramente perjudique a la ejecución del proyecto. Particularmente, se presentaron complicaciones que se supieron corregir a tiempo, y etiquetamos la compra en el exterior como una experiencia positiva.

11.3.5. Documentación

Parte de los objetivos del proyecto, era crear un sistema re-utilizable y aplicable en otros objetos de estudio. Por ello, se creó un Manual de Usuario y un

Manual Técnico[1]. El Manual de Usuario indica los pasos a seguir para poder utilizar el equipo, qué conexiones se deben hacer, y una explicación conceptual de cada módulo. El Manual técnico tiene el objetivo de presentar los test a realizar en caso de algún mal funcionamiento, indica las especificaciones de componentes, y cómo depurar al sistema en forma general. Por otra parte, los códigos desarrollados fueron documentados en Doxygen. Todo eso sumado a la Memoria de Proyecto hace de la Documentación un hito en sí dentro del proyecto y una práctica útil y enriquecedora.

[1]Documentación y manuales extras, junto con el código fuente se encuentran en `https://gitlab.fing.edu.uy/sarabee/publico`

Esta página ha sido intencionalmente dejada en blanco.

Apéndice A

Arquitectura de Software

Este capítulo presenta de forma breve una introducción a cuatro [39] arquitecturas típicas para sistemas embebidos: Round-Robin, Round-Robin con Interrupciones, encolado de funciones y Sistema Operativo en Tiempo Real (RTOS)."

A.1. Introducción

Los sistemas embebidos, son sistemas de bajos recursos con escasa interación con el humano y de rápida reacción. Partiendo de éste entendido, cuanto más simple sea la arquitectura mejor. Sin embargo hay otros factores a tener en cuenta como por ejemplo, la escalabilidad del producto y cuanto control se requiere tener sobre éste.

Es por ello que a continuación se presentarán cuatro arquitecturas distintas, ordenadas de la más simple a la más compleja.

A.2. Round-Robin (RR)

Round-Robin es sin lugar a dudas la arquitectura más simple a intuitiva para trabajar en un sistema embebido. Esta arquitectura consiste en un bucle principal, donde se verifica si hay algún dispositivo que requiere de atención. Ésto presenta ventajas tales como:no se utilizan interrupciones y por lo tanto tampoco se comparten datos entre ISR[1] y el bucle principal, no hay latencia de interrupciones. Por lo cual es un sistema muy simple y efectivo si no se tienen tiempos límites para atender dispositivos, o no se requiere asignar prioridades a estos, en tal caso una solución posible es consultar al dispositivo varias veces en el bucle. Ésto puede llegar a se muy complicado si la verificación es muy reiterativa o ésta actividad hay que realizarla con más de un dispositivo.

En resumen, esta arquitectura tiene como ventaja la simpleza. Pero puede presentar múltiples problemas, se listan algunos:

[1]Rutinas de Asistencia a las Interrupciones

- Si cualquier dispositivo requiere atención antes de que le toque su turno en el bucle principal no hay un mecanismo para hacer que éste sea atendido.
- Quizá no haya problema con el tiempo para que los dispositivos sean atendidos, pero aún así, puede que hayan dispositivos que ocupen durante mucho tiempo al procesador y estén trancando de atender otros dispositivos menos exigente de tiempo.
- Y aún si fuese posible lograr que el bucle principal fuese tan rápido que satisfaga lo anterior, agregar cualquier dispositivo podría provocar que ya no lo sea.

A.3. Round-Robin con Interrupciones (RR-I)

La arquitectura conocida como Round-Robin con Interrupciones tiene en su bucle principal banderas, que de estar activas ejecuta el código correpondiente al dispositivo de esa bandera. Funcionando de ésta forma igual que Round-Robin. Sin embargo tiene atención a interrupciones, que es donde ocurre lo importante. En las ISR primero se ejecuta el código impresindible para el correcto funcionamiento del dispositivo y finalmente, antes de retornar a al a tarea que se estaba realizando antes de la interrupción, se enciende la flag del dispositivo.

Lo anterior muestra que RR-I permite mejorar levemente el control que se tiene al establecer prioridades. Esto se debe a la ejecución de las ya mencionadas ISR, pues al ocurrir una interrupción el hardware envía una señal de interrupción la cual causa que el microcontrolador pare lo que está haciendo y ejecute la rutina asociada.

Por lo tanto, efectivamente, el código que se coloca en la rutina tiene una prioridad superior al resto del código que se ejecuta; a su vez es posible asignar prioridades a las interrupciones [40].

Hablemos ahora de los inconvenientes que puede introducir RR-I en nuestro sistema. Ésta arquitectura puede introducir el problema de los datos compartidos [41], un problema que ocurre al compartir información entre la ISR y el código principal, (por ejemplo al cambiar el valor de la bandera; o al poner en una cola los valores que obtuvo un sensor). Este problema es muy difícil de detectar, pues no ocurre en cada ejecución, incluso podría no presentarse nunca.

Al igual que RR, RR-I tiene el inconveniente que el mayor tiempo de espera de una tarea es que su bandera se haya prendido en el mismo instante en que el bucle principal acabase de consultarla y ésta estuviese con valor false, y a su vez todas las demás banderas estuviesen fuesen puestas en true a partir de ese momento. Lo cual implicaría que el tiempo fuese la suma del tiempo de ejecución de todas las otras tareas más el tiempo que implica atender todas las otras ISR. De la misma manera que en en RR, se puede implementar, en caso que sea necesario, que alguna bandera sea consultada más de una vez en una misma iteración del bulce principal, heredando los mismos comentarios que en el caso de RR.

Siendo breve, Round-Robin con Interrupciones es una arquitectura sencilla y potente siempre que pueda solucionarse el problema de datos compartidos, que no

sea necesario asignar de forma rigurosa tiempos límites antes de la ejecución de una tarea (o bien esto no genera un bucle principal intrincado), y que no haya necesidad de agregar o quitar la gestión de dispositivos.

A.4. Encolado de Funciones (EdeF)

Esta arquitectura difiere de las anteriores, primero porque lo que se realiza en el bucle principal es la ejecución de funciones acumuladas en una cola, las cuales son colocadas allí como punteros a funciones a través de por las ISR. Y la otra diferencia es que las ISR en lugar de cambiar de estado una bandera, lo que hace es encolar la función respectiva con los parámetros necesarios. Como el encargado de colocar la función en la cola es la ISR, ésta también es capaz de ordenar dicha cola, logrando de ésta forma asignar prioridad a las tareas además de a las interrupciones. El mayor problema que tiene ésta arquitectura es que una tarea con baja prioridad podría no ejecutarse nunca en caso de que siempre se encole una tarea de mayor prioridad antes de que ésta sea ejecutada, sin embargo una tarea de alta prioridad el peor caso que se puede encontrar es que en el momento de ocurrir su interrupción se haya comenzado a ejecutar la tarea de mayor tiempo. Y por lo tanto el tiempo máximo que deberá esperar es: el tiempo de la tarea que consume más tiempo, más el tiempo que lleva ejecutar su propia ISR. Solucionar éste problema podría conllevar que las tareas que consumen mayor tiempo se seccionen, y cada sección encole a la siguiente, pudiendo elevar mucho la complejidad del sistema.

En concreto, ésta es una arquitectura más compleja que las dos anteriores. La cual permite tener control sobre la prioridad de las tareas además de las interrupciones. También permite mayor escalabilidad pues agregar un nuevo dispositivo no es más que agregar la ISR correspondiente que encole la tarea a desarrollar y gestione su prioridad. Es una arquitectura que mejora el tiempo de respuesta a las tareas de alta prioridad, pero que puede perjudicar a las tareas de baja prioridad, logrando incluso que nunca se ejecuten.

A.5. Sistemas Operativos en Tiempo Real (RTOS)

Ésta es la arquitectura más sofisticada que se introducirá. Al igual que en las arquitecturas previamente mencionadas, en RTOS se atiende primero a las rutinas de las interrupciones más urgentes. Estas "firman" que hay tareas para realizar.

RTOS se diferencia de EdeF y de RR-I en que la "firma"que se realiza entre la ISR y la tarea, la realiza el RTOS mismo, y esto logra que no hay datos compartidos. Tampoco hay un bucle que decide que tarea se ejecuta a continuación, sino que es un mecanismo interno del RTOS quien lo decide, ya que sabe que taream sub tarea y sub rutina tiene mayor urgencia. Estas dos diferencias son principalmente convenientes al momento de programar, pero además existe otra. Esta diferencia es sustancial: el RTOS puede suspender cualquier tarea en cualquier parte de la ejecución, atender una tarea de mayor importancia, y retornar al lugar de la tarea

anterior que estaba ejecutando, por lo tanto el tiempo de espera para la tarea de mayor importancia es cero.

En síntesis, en RTOS la agenda de respuestas del sistema es muy estable incluso si hay cambios en el código, a diferencia del caso de RR, RR-I y EdeF donde ésta depende del largo de todos los tasks (incluso de los menos prioritarios), y de los cambios que puedan ocurrir en estos (o bien si se agrean otros, o quitan los existentes). No se debe perder de vista que el RTOS en si mismo consume tiempo de procesamiento en sus tareas internas.

Apéndice B

Bus I2C

Este capítulo presenta una introducción a los aspectos básicos del bus I2C [22].

B.1. Introducción

El bus I2C(del inglés Inter-Integrated Circuit), también conocido como bus I^2C, es un bus serie de datos desarrollado en 1982 por Philips Semiconductors. Por motivos de licencia Atmel, fabricante de los microcontroladores utilizados por Arduino, introdujo la designación TWI (del inglés Two Wire Interface). Desde el punto de vista técnico TWI e I2C son idénticos por lo que nos referiremos a ellos de forma indistinta. El sistema de comunicación es entre maestros y esclavos, en la topología puede existir más de un maestro, pero generalmente se usa un solo maestro y varios esclavos. Todos los esclavos tienen una dirección asignada.

B.2. Características eléctricas

I2C se compone de cuatro líneas, dos líneas de alimentación y dos de señal:

- VDD
- GND
- El reloj, denotado como SCK (Serial Clock)
- La línea de datos, denotado como SDA (Serial Data).

El Bus I2C trabaja con lógica positiva, esto quiere decir que un nivel alto en la línea de datos corresponde a un 1 lógico, el nivel bajo a un 0.

Las líneas de señal necesitan un pull-up de resistencias hacia VDD. Los dispositivos conectados al bus se configuran como open-drain (open-collector en bipolares), ver Figura B.1 lo cual en combinación con las resistencias pull-up, crea un circuito wired-AND. El nivel alto debe ser de al menos 0,7 x VDD y el nivel bajo no debe ser más de 0,3 x VDD.

La velocidad estándar del bus es 100kb/s, existen mayores pero no serán utilizadas.

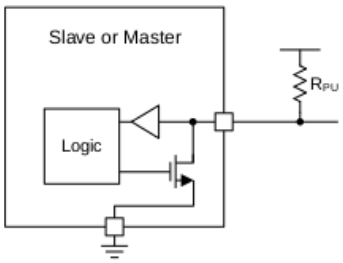


Figura B.1: Estructura open-drain

B.2.1. Condiciones de START y STOP

Toda transmisión de datos entre dispositivos comienza con una condición de START y finaliza con la condición de STOP. En la secuencia de START la línea SCL se mantiene constante en 1 y SDA baja a 0. En la secuencia de STOP la línea SCL también se mantiene constante en 1 pero SDA sube a 1. En la Figura B.2 se puede observar esto, la transferencia de datos se da siempre entre estas dos secuencias. Una particularidad es que la línea SDA siempre debe mantenerse constante cuando SCL está en 1 a excepción de estos dos casos.

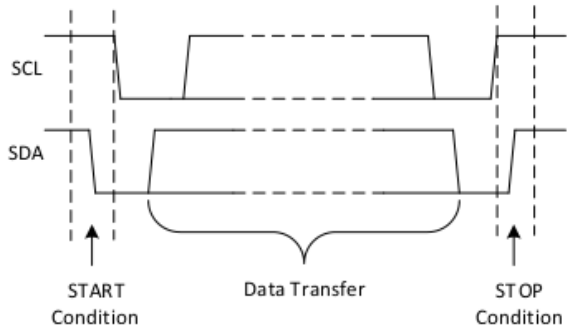


Figura B.2: Condiciones de START y STOP

B.2.2. Condición de START repetida

La condición de START repetida es similar a la condición de START, pero difiere de dicha condición porque ocurre antes de una condición de STOP, cuando el bus está activo.

B.2.3. Acknowledge (ACK) y Not Acknowledge (NACK)

Cada byte de datos enviado por el transmisor, incluido el byte de dirección, es seguido por un bit (SDA en 0) ACK del receptor. El bit ACK permite al receptor comunicar al transmisor que el byte se recibió correctamente y otro byte puede ser enviado. Antes de que el receptor pueda enviar un ACK, el transmisor debe liberar la línea SDA. El NACK es cuando el receptor no pone SDA a 0 y la linea permanece libre durante ese bit. El NACK significa una de las siguientes condiciones:

- El receptor no puede recibir o transmitir porque está realizando alguna función en tiempo real y no está listo para iniciar la comunicación con el maestro.
- Durante la transferencia, el receptor obtiene datos o comandos que no entiende.
- Durante la transferencia, el receptor no puede recibir más bytes de datos.
- Un maestro-receptor ha terminado de leer los datos e indica esto al esclavo a través de un NACK.

B.3. ADDRESS y bit de R/W

Luego de enviar la condición de STOP el maestro inmediatamente 7 bits que equivalen el ADDRESS del dispositivo con el que quiere entablar la comunicación. Luego envía un octavo bit (R/W) indicando si es escritura (bit en 1) o lectura (bit en 0). Si el esclavo detecta debe responder con un ACK. Los 7 bits de la ADDRESS equivale a un total de 128 direcciones. A su vez existen direcciones reservadas lo cual limita a 112 direcciones, esto es un máximo de 112 dispositivos. Se definiran direcciones para los modulos que seran unicos y dividirá en rangos preasignados de forma de detectar el tipo de módulo a partir de su dirección.

B.3.1. Registros

El registro es el siguiente byte enviado por el maestro luego del ACK de la dirección y el bit R/W. Dicho byte indica en que registro el maestro va a leer o escribir sus datos. No necesariamente tiene que ser un registro físico sino que puede ser una directiva para el esclavo.

B.4. Operación general de I2C

Existen dos tipos de procedimientos en la comunicación a través de I2C:

1. Maestro-transmisor y esclavo-receptor
2. Maestro-receptor y esclavo-transmisor

La secuencia de transmisión de maestro-transmisor y esclavo-receptor es la siguiente:

1. Maestro-transmisor envía la condición de START y direcciona al esclavo-receptor
2. Maestro-transmisor envía datos al esclavo-receptor
3. Maestro-transmisor termina la transferencia con una condición de STOP

La secuencia de transmisión de maestro-receptor y esclavo-transmisor es la siguiente:

1. Maestro-receptor envía la condición de START y direcciona el esclavo-transmisor
2. Maestro-receptor envía el registro requerido a esclavo-receptor
3. Maestro-receptor recibe datos desde esclavo-receptor
4. Maestro-receptor termina la transferencia con una condición de STOP

B.5. Lectura y escritura

Para escribir en el bus, el maestro envia una condición de STOP en el bus con la dirección del esclavo, así como el último bit (el bit R/W) en 0, lo que significa una escritura. Después de que el esclavo envía el bit de ACK, el maestro envía el registro en el que desea escribir. El esclavo le envía un ACK de nuevo, avisándole al maestro que está listo. Después de esto, el maestro comienza a enviar los datos de registro al esclavo, hasta que el maestro haya enviado todos los datos, y el maestro termina la transmisión con una condición de STOP. En la Figura B.3 se puede ver con más detalle esta secuencia.

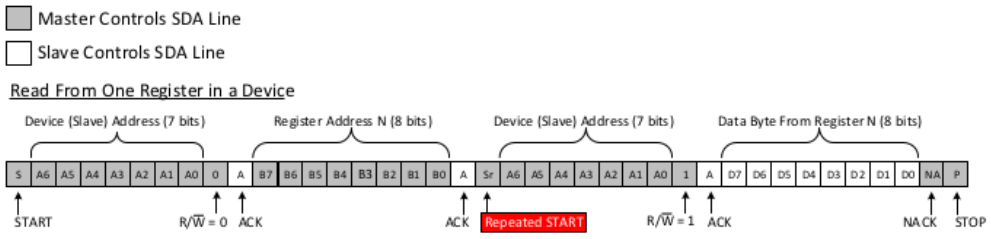


Figura B.3: Maestro transmisor - Esclavo receptor

La lectura de un esclavo es muy similar a escribir, pero con algunos pasos adicionales. Para poder leer desde un esclavo, el maestro debe primero indicar al esclavo qué registro desea leer. Esto se hace cuando el maestro comienza la transmisión de manera similar a la escritura, enviando la dirección con el bit R/W igual a 0, que significa una escritura, seguido de la dirección de registro de la que desea leer. Una vez que el esclavo reconoce esta dirección de registro, el maestro

envía una condición de START repetida, seguida de la dirección del esclavo con el bit R/W establecido en 1 para una lectura. Esta vez, el esclavo reconoce con un ACK la solicitud de lectura y el maestro libera el bus SDA, pero continua suministrando el reloj al esclavo en SCL. Durante esta parte de la transacción, el maestro se convierte en maestro-receptor, y el esclavo se convierte en esclavo-transmisor. Al final de cada byte de datos, el maestro envía un ACK al esclavo, informándole al esclavo que está listo para más datos. Una vez que el maestro ha recibido el número de bytes que espera, envía un NACK, indicando al esclavo que detenga las comunicaciones y libere el bus. El maestro culmina la comunicación con la condición de STOP. En la Figura B.4 se puede ver con más detalle esta secuencia.

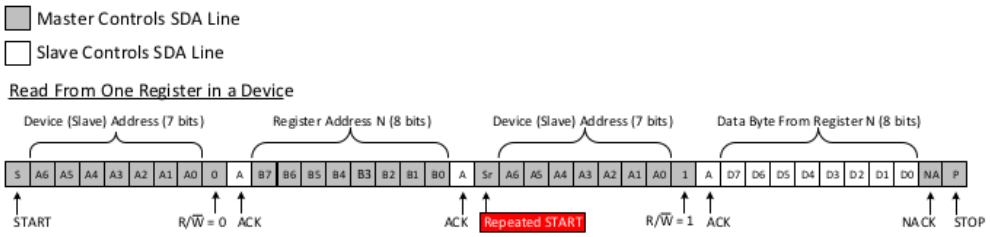


Figura B.4: Maestro receptor - Esclavo transmisor

Esta página ha sido intencionalmente dejada en blanco.

Apéndice C

Acondicionamiento de señales

Este anexo cumple la función de desplegar los cálculos realizados para el acondicionamiento de señales, en particular la etapa amplificadora. El filtro diseñado es un filtro Butterworth de segundo orden, el circuito se observa en la figura C.1.

El filtro Butterworth tiene una caída de 20n dB por década, siendo n el orden del filtro. Se observa en la figura C.2 el filtro para orden 2, 3 y 4. En el proyecto se decidió implementar un filtro de segundo orden, alcanzando una caída de 40dB por década.

Se toma como hipótesis que el amplificador operacional es ideal, entonces $V_- = V_o$. Para hallar la transferencia del circuito, se aplica el método de nodos de la ley de Kirchhoff, en el nodo N:

$$\frac{V_{in}-V}{R_{f1}} = (V - V_o)C_{f1}s + \frac{(V-V_o)}{R_{f2}}$$

Se eligió la relación: $R_{f1} = R_{f2} = R$ y $C_{f2} = \frac{C_{f1}}{2} = \frac{C}{2}$

Se relacionan los voltajes V con V_o a través de un divisor de tensión:

$$V_o(s) = \frac{\frac{V(s)}{C_{f2}s}}{R_{f2}+\frac{1}{C_{f2}s}} = \frac{2}{RCs+2}.V$$

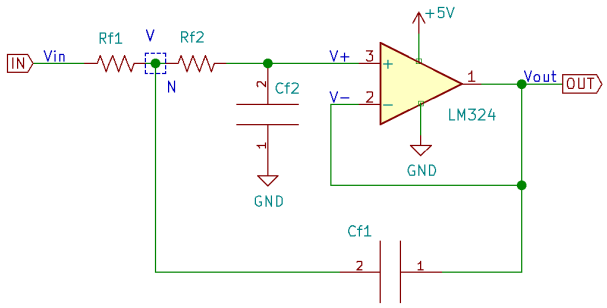


Figura C.1: Filtro Butterworth de segundo orden

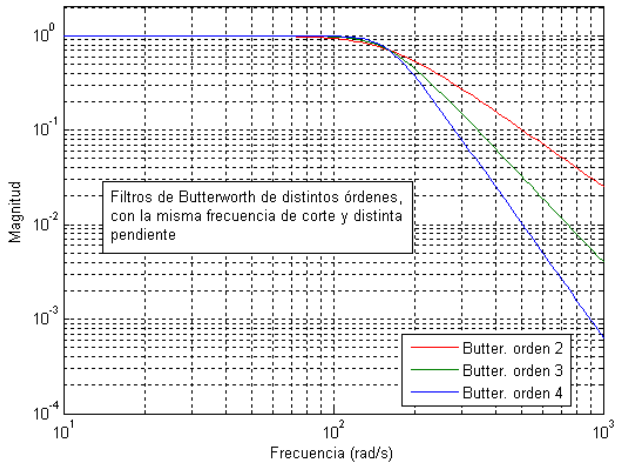


Figura C.2: Órdenes filtro Butterworth

Despejando V: $V(s) = \frac{RCs+2}{2}.V_o(s)$, se sustituye esta relación en la ecuación de nodos, y operando se obtiene la función de transferencia:

$$H(s) = \frac{V_o(s)}{V_{in}(s)} = \frac{1}{1+RCs+\frac{(RC)^2}{2}s^2}$$

Se escribe la transferencia en frecuencia, s=jw:

$H(jw) = \frac{1}{1+RCjw-\frac{(RC)^2}{2}w^2}$. Se define la frecuencia w_o, como $w_o = \frac{1}{RC}$

Se calcula el módulo de la transferencia al cuadrado en función de w_o:

$$\|H\|^2 = \frac{1}{(1-\frac{w^2}{2w_o})^2+\frac{w^2}{w_o}}$$

Operando:

$$\|H(jw)\|^2 = \frac{1}{1+\frac{w^4}{4w_o^4}}$$

El módulo al cuadrado de un filtro Butterworth de orden N es:

$$\|H(jw)\|^2 = \frac{1}{1+(\frac{w}{w_c})^{2N}}$$

Se verifica que la implementación del filtro es un filtro Butterworth de segundo orden, con una frecuencia de corte: $w_c = \frac{\sqrt{2}}{RC}$.

Apéndice D

FFT

Este anexo tiene el fin de desarrollar el algoritmo de la Transformada Rápida de Fourier, es fuertemente basada en el apéndice que utiliza PestiBee [1]. Y por más información se puede consultar la bibliografía utilizada [18].

Se le llama Transformada Rápida de Fourier (FFT) a un algoritmo computacional para calcular de forma rápida una Transformada Discreta de Fourier (DFT). Esta se define como una serie de muestras equiespaciadas de la Transformada de Fourier de la señal. De esta forma, una DFT de n puntos sería idéntica a tomar n muestras de la transformada de Fourier de la señal, separadas entre sí: $wk = 2kN$. Así, la DFT de una serie finita de longitud N es:

$X[k] = \sum_{n=0}^{N-1} x[n] W_{N_{kn}},\ k = 0, 1, ..., N-1$

Siendo $W_{N_k} = e^{\frac{-j2\pi k}{N}}$

Calcular la DFT puede llevar una gran cantidad de operaciones, calcular la DFT de 2048 muestras requiere 4.194.304 multiplicaciones de valores complejos y una cantidad similar de sumas complejas. Para el caso de una FFT la cantidad de operaciones rondan las 22.528, lo que muestra la utilidad de implementar estos algoritmos.

Este algoritmo se lleva a cabo en la FPGA de la placa Arduino MKR Vidor 4000, y se utiliza un core abierto obtenido desde *OpenCores* [46], el cual realiza las operaciones utilizando un bloque *Butterfly*, su nombre proviene de la forma del diagrama de flujo de datos, e implementa radix-2 no recursivo de decimación en el tiempo.

Se desarrolla brevemente cómo se implementarían estos cálculos de la FFT. Se divide el cálculo de la DFT en pequeñas transformadas. Aprovechando la simetría y periodicidad del número complejo WN, se calculan 1024 DFT de muestras de dos datos, y luego se recombinan para obtener la DFT de 2048 muestras.

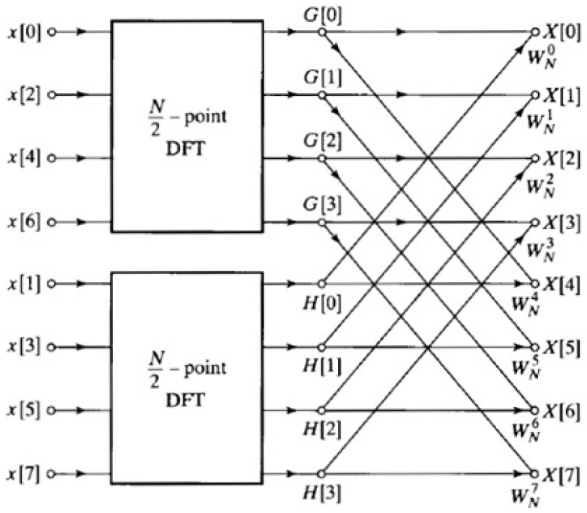


Figura D.1: Método radix 2 para ocho muestras

Este método parte de una cantidad de datos N, con N potencia de 2. Así, para el cálculo de la DFT se separan las N muestras en dos tiras de N=2: por un lado los puntos de índice par y por otro lado los impares. De esta forma se tiene:

$$X[k] = \sum_{m=0}^{N-1} x[2m] W_M^{mk} + \sum_{m=0}^{N-1} x[2m+1] W_M^{mk}$$

Es decir, se tienen ahora dos sub-FFT, de N/2 puntos cada una. Como N es potencia de 2, N/2 es par, y por tanto cada una de estas sub-FFT se puede subdividir en otras dos. A cada subdivisión se le considera como una etapa. Este razonamiento se aplica hasta tener que calcular la DFT de una pareja de puntos, que se resuelve rápidamente sustituyendo N=2 en la primera ecuación de X[k]. En la figura D.1 se muestra un diagrama que representa el método para ocho datos.

La consideración más importante para este tipo de algoritmos es si se realiza de forma recursiva o *in place* (no recursiva). Si la implementación es recursiva, y el algoritmo es óptimo, necesitan dos registros del tamaño de las muestras de valores complejos, uno que tiene los valores de la etapa actual y otro para almacenar los de la etapa siguiente. En el peor caso, por cada etapa se crea un registro del tamaño de las muestras.

Este método puede ser más rápido que el no recursivo, pero tiene la desventaja de que requiere más memoria RAM. En el caso de 2048 muestras, el método recursivo es de 11 etapas. Cada registro tiene 2048 valores reales y la misma cantidad de complejos. Cada valor ocupa 32 bits, por lo que cada registro ocupa en memoria 16kB. Al tener dos registros se alcanza el valor de la memoria RAM total (32kB), y entonces es imposible que estos algoritmos funcionen con el microcontrolador utilizado.

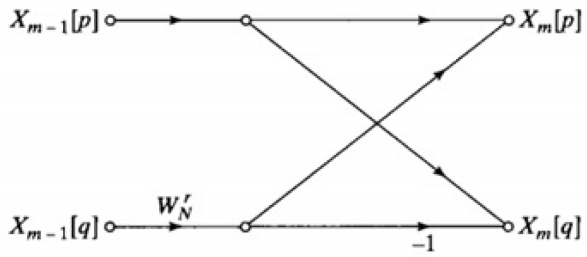


Figura D.2: Algoritmo no recursivo

En cambio los algoritmos de cálculos *in place* o no recursivos, utilizan solo un registro para guardar los valores de las etapas y realizar los cálculos. La forma en que este tipo de algoritmos funcionan es mediante un reordenamiento del registro que guarda o accede a los datos. Se hace de tal manera que el cálculo de los valores *p-ésimo* y *q-ésimo* de la etapa m, solo requiere los mismos valores de la etapa m-1. Esta implementación del algoritmo puede observarse en la figura D.2

Resolución

Como se ve en la ecuación B.1, la DFT tiene tantas componentes de frecuencia como muestras tiene la señal. Por lo tanto, la resolución de la DFT está dada por: $\Delta f = \frac{f_s}{N}$

Donde N es la cantidad de muestras y fs es la frecuencia de muestreo. La ventana de la señal tiene duración:

$$T = TsN = \frac{N}{f_s}$$

Entonces, despejando fs de la ecuación de resolución y sustituyendo en la ecuación anterior se obtiene que:

$$T = \frac{1}{\Delta f}$$

Por lo tanto, para obtener una resolución en frecuencia de 2Hz se toman ventanas de 500ms.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice E

FPGA

Este anexo tiene el fin de familiarizarse con las FPGA. Por más información acudir a la referencia [35]. Su nombre viene de un acrónimo inglés: *Field Programmable Gate Array*, que significa matriz de puertas reprogramable. Como primer concepto,las FPGA permiten describir un circuito digiral usando un lenguaje específico, lo más comunes son VHDL y Verilog, VHDL es utilizado en el proyecto, y cuando este código se carga en el integrado, el circuito es creado físicamente allí.

Las FPGA se componen de puertas lógicas biestables, puertos de entrada y salida. Los componentes se encuentran sin conectar, hasta que se les carga un *bitstream*, este es un archivo generado a partir de la descripción del circuito.

Se crearon en 1984 por Ross Freeman y Bernard Von Der Schmitt, confundadores de Xilinx, surgen como una evolución de los CPLD (Complex Programmable Logic Device). En particular, las FPGA son el resultado de dos teconologías, por un lado los dispositivos lógicos programables PLD (Programmable Logic Devices), y potr otro los circuitos integrados de aplicación específica ASIC (Application Specific Integrate Circuit).

Las FPGA son dispositivos semiconductores que se basan en una matriz de bloques lógicos configurables (CLB) conectados a través de interconexiones programables. Se pueden reprogramar para la aplicación deseada después de la fabricación. Esta es la principal característica que distingue las FPGA a los circuitos integrados específicos de aplicación (ASIC), que se fabrican a medida para tareas de diseño específicas.

Las FPGA tienen celdas que se pueden configurar como memoria (FLIP-FLOP tipo D), como multiplexor o con una función lógica tipo AND, OR, XOR. El programador debe describir el hardware que desee en la FPGA, es decir la función lógica que debe realizar cada CLB e interconectarlos.

El flujo de trabajo de las FPGA se observa en la figura E.1. En primer lugar, se debe diseñar el circuito, una vez definido lo que se desea crear, se pasa a escribir

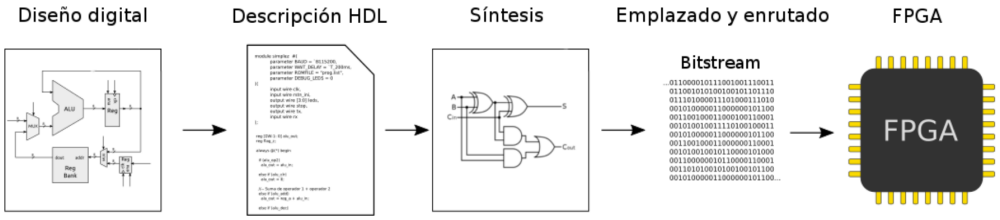


Figura E.1: Etapas de diseño para una FPGA

el código en lenguaje de descripción: en VHDL o Verilog. La tercera etapa es la síntesis del circuito y es una de las claves del proceso, aquí se escogen los elementos a usar y su interconexión en función de los archivos de descripción. Hasta aquí los pasos anterior son los mismos para todas las familias de las FPGAs. Sin embargo, la última etapa, conocida como emplazado y enrutado, es específica para cada familia. En esta etapa se crea el bitstream, archivo que describe la configuración interna necesaria para crear lo que haya sido previamente descrito y sintetizado.

Apéndice F

MQTT

MQTT es un protocolo de comunicación liviano, de simple implementación, eficiente en el consumo de ancho de banda y con control de calidad de servicio. El mismo fue desarollado en 1999 por Andy Stanford y Arlen Nipper, el primero pertenecía a "IBM" y el segundo a "Arcom", la actual "Cirrus Link". En ese entonces estaban trabajando en la serie de productos "MQ" de "IBM" y de allí el nombre (MQ Telemetry Transport). En la actualidad es ampliamente utilizado en proyectos de sistemas embebidos, particularmente en el mundo de IoT.

F.1. Introducción

MQTT es un protocolo de comunicación desarrollado originalmente por IBM, el mismo es liviano, eficiente y sencillo, actualmente es ampliamente utilizado para solucionar la conexión de dispositivos IoT. Parte de la simpleza reside en el sistema de publicación y suscripción, donde las conexiones de los clientes ocurre solo con el *broker* y no con los otros clientes. Esto permite, entre otras cosas, que el cliente que hace una publicación no precisa conocer el estado de quien lo va a recibir, incluso éste último puede no existir. A continuación se presenta un capítulo para entender MQTT con mayor profundidad. Este capítulo está fuertemente basado en las 10 entradas del blog de HiveMQ tituladas MQTT Essentials [42].

F.1.1. Origenes

Desde 1993 la compañía "IBM" se encontraba trabajando en una serie de productos para *middleware* los MQ. En 1999 Andy Stanford y Arlen Nipper desarrollan un protocolo de comunicación para estos equipos al que llaman MQTT (MQ Telemetry Transport). Estos equipos tenían bajos recursos y un uso particular de ellos fue transmitir información de las plataformas petroleras vía satélite. Para ello el protocolo debía tener una implementación simple, ser liviano, eficiente en el consumo de ancho de banda, poder interactuar con distintas bases de datos sin importar cuales sean, y además era necesario que tuviera control sobre la calidad de servicio (QoS) y que tuviera control sobre continuidad de la comunicación. En

2011 Eclipse solicitó que se le fuera brindado éste protocolo para su plataforma "Paho". Desde allí se presentó ante OASIS en 2013 para que fuera estandarizado. Actualmente el protocolo se encuentra en su versión 5, siendo la versión 3.1.1 la mayormente utilizada.

F.1.2. Beneficios

Los ya nombrados:

- simple de implementar
- distintos niveles de calidad de servicio (QoS=0-Qos=2)
- liviano
- eficiente en el uso del ancho de banda
- independiente de las bases de datos que se utilice (*"data agnostic"*)
- control sobre la continuidad de la comunicación

Pero también trae otros beneficios intrínsecos a la implementación mediante suscriptores y publicadores:

- desacopla la comunicación en el tiempo
- desacopla la comunicación en el espacio
- no es necesario el sincronismo
- los clientes no necesitan información sobre la existencia de otros clientes

F.1.3. Estructura de comunicación

El sistema se compone de "N" **clientes**, un **broker** y M **topics**. Los clientes sólo se comunican con el broker y lo hacen a través de los topics. Las acciones que pueden realizar los clientes son o bien suscribirse a los topics o bien publicar en el topic. De esta forma un cliente le comunica al broker que publica en un topic, y el broker le envía a el mensaje a todos los suscriptores. Cuando se dijo que MQTT desacopla en tiempo y espacio y que no es necesario el sincronismo, es debido a ésta metodología. Ya que no es necesario que el cliente suscriptor esté conectado a la misma vez que el cliente que publicó.

F.2. Clientes y Broker

F.2.1. "Client"

Tanto los suscriptores como los publicadores son clientes MQTT, por lo que estas funcionalidades deben estar implementadas en los clientes. Hoy en día hay

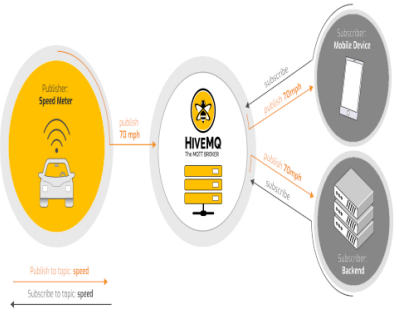


Figura F.1: Ejemplo del esquema de publicación y suscripción

librerías que implementan MQTT en una amplia gama de lenguajes, por ejemplo los hay en Android, **Arduino**, C, C++, C#, Elixir, GO, Haskel, iOS, JAVA, JavaScript y .NET entre otros. [1].

F.2.2. "Broker"

Se puede decir que el broker es el corazón mismo de este protocolo basado en publicaciones y suscripciones. éste es el responsable de recibir, filtrar y redirigir hacia los suscriptores los mensajes. Es el responsable de la autentificación y autorización de los clientes, y por lo tanto también almacena la información de las sesiones persistentes de los clientes. Así es entonces que el broker es la parte central del sistema, por ello es deseable que éste sea escalable, integrable con sistemas de backend, fácil de monitorear y por supuesto robusto.

F.3. "Topics"

Los topics o tópicos, se pueden ver como canales que son gestionados por el broker y utilizados por los clientes, cuyo nombre es una palabra con caracteres UTF-8, sensible a mayúsculas/minúsculas y de largo al menos un carácter, cabe resaltar que acepta espacios como carácter aunque no es recomendado. Cuando un cliente quiere comunicar algo lo publica en el topic correspondiente, el broker lo escucha y se lo transmite a todos los clientes que estén suscriptos a ese topic. Los topics no hay que crearlos previamente, comienzan a existir cuando un cliente o bien publica o bien se suscribe a un topic. Un topic puede consistir de varios niveles de topics separados por una barra ("/"). Ejemplos de esto son los siguientes

- casa

[1]En el siguiente link se pueden encontrar implementaciones de MQTT para distintaas plataformas y lenguajes `https:\/\/github.com\/mqtt\/mqtt.github.io\/wiki\/libraries`

- casa/cocina
- casa/cocina/luz
- casa/cocina/heladera
- casa/dormitorio/luz
- casa/dormitorio/temperatura ambiente
- SARA-Bee/modulo_de_alimentacion

F.3.1. Wildcards

Un cliente, puede suscribirse a un tópico específico, o puede suscribirse a varios en simultaneo mediante el uso de **Wildcards**. Lo que no podrá hacer es publicar simultaneamente mediante el uso de éstas.

Un nivel "+"

éste simbolo lo que permite es incluir todo un nivel de tópicos. Por ejemplo si se suscribe a CASA/+/LUZ se estará suscribiendo a CASA/COCINA/LUZ y a CASA/DORMITORIO/LUZ.

Multiples niveles " # "

Este carácter derá ser utilizado como último carácter, ya que al emplearlo lo que se está haciendo es suscribirse a todos los tópicos que se encuentran a continuación. al utilizar CASA/COCINA/# se está suscribiendo a

- casa/cocina/luz
- casa/cocina/heladera

y al utilizar CASA/# la suscripción es realizada para:

- casa/cocina
- casa/cocina/luz
- casa/cocina/heladera
- casa/dormitorio
- casa/dormitorio/luz
- casa/dormitorio/temperatura

Si la suscripción fuese a # el cliente va a recibir todas las publicaciones que se realicen en el broker.

Tópicos especiales "$"

Hay una única restricción para nombrar los topics de MQTT, y es que no pueden comenzar con el signo $, ya que los tópicos que comienzan con él son para estadísticas internas del broker y por lo tanto utilizando Wildcards tampoco se realiza una suscripción a estos y los clientes no tienen permitido publicar en estos topics. El uso de estos topics no está estandarizado y depende de la implementación del broker, un ejemplo habitual es el siguiente.

- $SYS/broker/clients/connected
- $SYS/broker/clients/disconnected
- $SYS/broker/clients/total
- $SYS/broker/messages/sent
- $SYS/broker/uptime

F.4. Comunicación MQTT

Es un protocolo que va sobre TCP/IP, y por lo tanto los clientes como el broker tienen que trabajar con TCP stack.

Las conexiones son simepre entre los clientes y el broker como se puede ver en la Figura F.2.

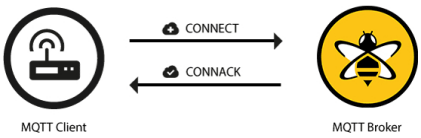


Figura F.2: Flujo de conexión MQTT

Estas conexiones comienzan con el cliente enviando al broker el mensaje **CONNECT**, y éste contesta con el mensaje **CONNACK** y el *status code* correspondiente. Una vez que la comunicación queda establecida, el broker la mantiene hasta que el cliente envía le mensaje de desconexión, o bien detecta que la conexión falló. A continuación se describe el mensaje y sus campos a nivel de usuario.

F.4.1. Mensaje del cliente para establecer conexión

El mensaje de conexión es como el representado en la Figura F.3. Este mensaje es enviado por el cliente al broker. En busca de evitar clientes maliciosos que puedan enlentecer al broker, si el mensaje no corresponde con la especificación de MQTT o el cliente demora demasiado entre que abrió el socket y envió el mensaje el broker cierra la conexión.

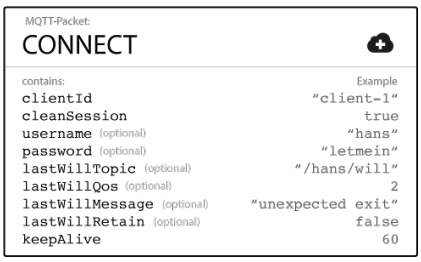


Figura F.3: Mensaje del cliente para conexión MQTT "CONNECT"

Identificador de cliente "ClientID"

Este identificador es único para cada cliente que se conecta al borker y es utilizado por éste no solo para identificar al cliente, sino también para saber su estado. Desde MQTT 3.1.1, si el cliente no necesita que el broker retenga su estado puede envíar vacío este campo.[2]

"clean session"

Este campo es una bandera, la cual le indica al broker si el cliente desea establecer una sesión persistente o no, en el primer caso deberá seterase como "False" y en el segundo como "True". En una sesión persistente, el broker almacenará las suscripciones del cliente, y si la suscripción tiene calidad de servicio (QoS) nayormayor a 0 (1-2) también almacenará los mensajes que deba entregarle a éste. En caso que la sesión sea no persistente (Clean sesion = true) entonces no almacenará no almacenará nada respecto del cliente y además borrará toda la información de sesiones persistente anteriores.

Usuario y contraseña ("username" / "password")

El cliente puede enviar usuario y contraseña para autenticarse y conseguir autorizaciones. Si no se utiliza ninguna encriptación tanto el usuario como la contraseña serán enviados en texto plano. Hay brokers que permiten autenticación SSL y por lo tanto no necesitan de usuario y contraseña.

último de deseo ("lastWillTopic"/"lastWillQoS"/"lastWillMessage")

Si la conexión del cliente con el broker falla, el broker publicará el mensaje que se encuentra en el campo "lasWillMessage" en el topic indicado en el campo "lastWillTopic" y tendrá la calidad de servicio indicada en "lastWillQoS". Todo esto está bajo una funcionalidad de MQTT llamada "Last Will and Testament (LWT)".

[2]Si el ClientID es envíado vacío, entonces la sesión no tendrá estados, y por lo tanto la bandera "clean sesion" deberá estar seteada como "true" o el mensaje será rechazado por el broker

"keepAlive"

El campo "KeepAlive" indica el tiempo en segundos tras el cual si no hubo interación entre broker y cliente, ambos asumirán que la comunicación falló. Mientras el cliente no tenga necesidad de publicar en el borker, deberá enviarle un paquete de PING y el mismo deberá ser respondido por el broker.

F.4.2. Rerpuesta del broker para establecer la conexión

Cuando el broker recibe una mensaje "CONNECT" deber responder con un el mensaje "CONNACK", este mensaje contiene dos entradas como se puede ver en la FiguraF.4:

- Bandera se sesión presente "sessionPresent"
- Código de retorno "returnCode"

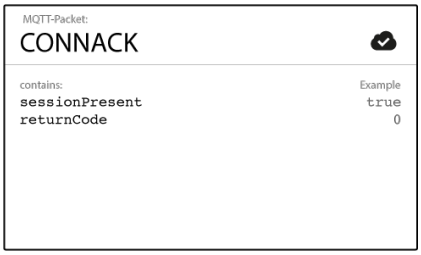


Figura F.4: CONNACK - Respuesta del broker para el mensaje CONNECT

a continuación se verán los campos del mensaje.

Bandera de sesión presente "sessionPresent"

Esta bandera le indica al cliente que el broker ya tiene una **sesión persistente** con ese cliente de una interación previa, sessionPresent=True si hay información previa, sessionPresent=False si no la hay. El caso sessionPresent=True se puede dar solo si en el paquete "CONNECT" la bandera "cleanSession" está seteada como False, ya que en caso de ser True el broker habrá desechado toda información almacenada sobre el cliente. La información de esta bandera, usualmente es usada para saber si es necesario suscribirse a topics.

Código de retorno

Esta entrada del mensaje "CONNACK" retorna en el campo "returnCode" un código que le dice al cliente si la conexión fue exitosa. Los valores que puede tomar "returnCode" son los que se muestran en la TablaF.1.

se puede enconrar más detalle en la documentación de especificidad de MQTT [3]

[3] `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718035`

Código de retorno	Significado del Código
0	conexión establecida
1	conexión rechazada, versión inaceptable del protocolo
2	conexión rechazada, rechazo del identificador
3	conexión rechazada, servidor no disponible
4	conexión rechazada, usuario o contraseña equivocada
5	conexión rechazada, sin autorización

Tabla F.1: Tabla de códigos de retorno de CONNACK

F.5. Publicaciones y Suscripciones

El sistema de publicaciones y suscripciones de MQTT es parte fundamental en su sencillez y de la explicación de porqué es un protocolo tan usado para IoT. Es lo que permite que mediante el broker, un cliente pueda interactuar con otro, o con un servidor, o con una aplicación sin necesidad de hacerlo directamente. Veamos entonces como aplican estos dos conceptos.

F.5.1. Publicaciones

El cliente de MQTT puede publicar en el broker ni bien se conecta con él. Cada mensaje publicado debe tener el nombre del tópico al cual el broker deberá redirigirlo para que lo reciban los clientes interesados, es usual que los mensajes tengan un *payload* que contiene la información a transmitir en forma de bytes. La estructura del payload va especificado según el caso de uso del cliente que publica, puede estar estucturado como binario, texto plano, XML, JSON entre otros. A continuación se estudia el mensaje **"PUBLISH"** que se muestra en la FiguraF.5

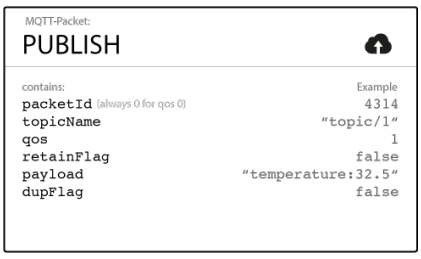


Figura F.5: Paquete de publicación

Identificador del paquete - "packetId"

Este identificador indentifica de forma única a cada mensaje entre borker y cliente, este campo solo es importante si utiliza QoS mayor a 0, en cuyo caso tendrá un valor también distinto de 0.

Nombre del tópico - "topicName"

Nombre completo del topic en el cual se quiere publicar por ejemplo CASA/COCINA/TEMPERATURA o CASA/COCINA como se vió en la sección F.3

Clidad de Servicio - "QoS"

Valor del nivel de QoS que se espera tener para alcanzar el destino, en este caso el broker.

Bandera de retención - "retainedFlag"

Indica si el mensaje debe ser retenido como el último mensaje válido hasta que se publique otro. éste mensaje será enviado al cliente cuando se suscriba al topic.

"payload"

Es el mensaje en sí mismo, puede ser imágenes, texto, JSON, información encriptada, o cualquier cosa que sea enviada en binario.

"dupFLag"

Esta bandera es relevante si QoS es mayor que 0, ya que indica si el mensaje es un duplicado. Esto es, que es un reenvío del mensaje porque el receptor no confirmo que lo recibió. Cuando el mensaje es enviado por el un cliente, el broker lee el mensaje, si corresponde envía la confirmación de que lo recibió, y lo procesa (estoy incluye que se lo envía los clientes suscriptos al topic). El cliente que publica, solo se tiene que preocupar de que éste llege al broker, una vez que el broker le confirma la recepción pasa a ser responsabilidad de éste. La Figura F.6 muestra el esquema de éste flujo.

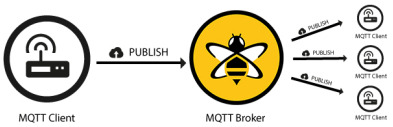


Figura F.6: Flujo de una publicación

F.5.2. Suscripción

Para recibir los mensajes que se publican en un topic los clientes deben suscribirse a ese tópico. Este flujo se puede ver representado en la Figura F.7 Para ello debe enviar el mensaje de Suscripción que lo podemos ver en la Figura F.8, el broker deberá responder con un mensaje "SUBACK" como el de la Figura F.9 los campos de ambos paquetes se detallan a continuación.

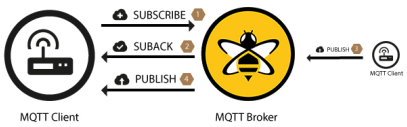


Figura F.7: Flujo de paquetes al realizar una suscripción

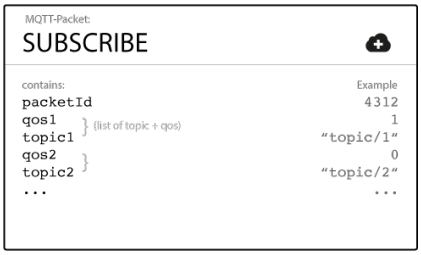


Figura F.8: Paquete de suscripción a tópico

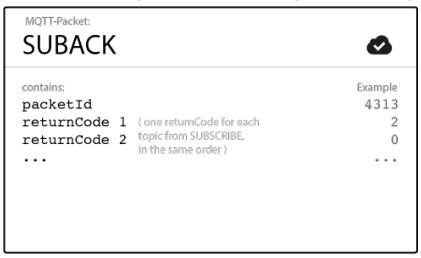


Figura F.9: Paquete de confirmación de suscripción

Identificador de paquete - "packetID"

Este campo de ambos paquetes, tanto en SUBSCRIBE como en SUBACK, únicamente identifica el paquete en flujo entre cliente y broker, son las librerías del cliente y del broker las responsables de configurar internamente este valor.

Topicos y Calidad de servicio

Como en un único mensaje se pueden suscribir a más de un tópico, se debe incluir para cada suscripción el campo "topicN" y "QosN". En caso de que se esté sobre-suscribiendo a un tópico, prevalecerá la suscripción que tenga el valor mayor de QoS.

Código de retorno - " returnCode N"

Estos códigos de retorno pertenecen únicamente al paquete SUBACK, y es enviado como repuesta para cada par topic/QoS que fue recibido en el paquete SUBSCRIBE. Esto es que habrán tantos returnCode como suscripciones a tópicos, y los valores posibles son los de la Tabla F.2.

Código de retorno	Significado del código
0	Exitoso - Máximo QoS=0
1	Exitoso - Máximo QoS=1
2	Exitoso - Máximo QoS=2
128	Exitoso - Falló

Tabla F.2: Códigos de retorno en SUBACK

F.5.3. Des-Suscripción

Es la contracara a las suscripción y lo que hace es des-suscribir al cliente del topic. EL mensaje que se debe enviar se llama **"UNSUBSCRIBE"** y su estructura es muy similar a la del mensaje SUBSCRIBE ya que lo compone el identificador de paquete y los tópicos a los cuales se a borrar la susbripción, un ejemplo presenta en la Figura F.10. Para completar la des-suscripción el broker debe contestar con un mensaje llamado **"UNSUBACK"** y es similar al de la Figura F.11

Figura F.10: Paquete de dessuscripción a tópico

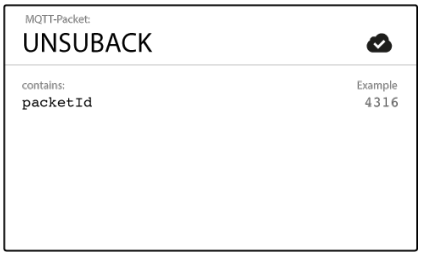


Figura F.11: Paquete de confirmación de dessuscripción

Como se observa en dichas figuras, estos mensajes son mas cortos y simples que los utilizados en la suscripción, sin embargo los campos ya son conocidos. El mensaje UNSUBSCRIBE enviado por el cliente debe contener el identificador de paquete para que el broker pueda responder en el mensaje UNSUBACK que el la petición se concretó correctamente. Y además en el mensaje UNSUBSCRIBE se envía a que tópicos se está desusccribiendo. La simpleza de el flujo de esta comunicación se puede observar en la Figura F.12

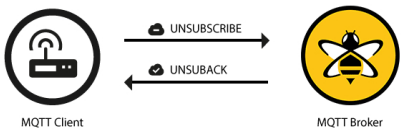


Figura F.12: Flujo de paquetes al realizar una dessuscripción

F.6. Calidad de servicio "QoS"

La calidad de servicio es un acuerdo entre quien envía el mensaje y quien lo recibe, con el cual definen con que garantía ca a ser entregado el mensaje. Hay tres niveles:

- 0 - A lo sumo una vez
- 1 - Por lo menos una vez (puede llegar duplicado)
- 2 - Exactamente una vez

Cuando se habla de entregar mensajes hay dos direcciones posibles, la publicación de un cliente al broker y la entrega de ese mensaje del broker a los clientes suscriptos al topic. El cliente que publica en el broker define un QoS en el cuerpo del mensaje. Y el broker lo va a entregar a los suscriptores con el QoS que ellos definieron en la suscripción.

F.6.1. QoS 0 - A lo sumo una vez

En este caso no hay garantía de que el mensaje sea entregado, el mensaje no va a ser ni almacenado ni retransmitido por quien lo envía, ya sea el broker o un cliente. A este nivel se lo llama **"fire and forget"** algo así como enviar y olvidar, brinda las mismas garantías que el protocolo TCP sobre el que corre MQTT. Este flujo se observa en la Figura F.13

Figura F.13: Flujo para establecer conexión con QoS 0

F.6.2. QoS 1 - Por lo menos una vez

Este QoS garantiza que el mensaje es recibido por lo menos una vez. Quien envía el mensaje lo almacena hasta que el receptor le contesta con el mensaje PUBACK con el ID del mensaje envíado. Bajo éste regimen los mensajes puede llegar diplicados si el receptor no responde a tiempo y el mensaje es reenviado. A

partir de que el mensaje debe ser reenviado, en el mensaje se pone una bandera llamada **DUP** que indica que es un duplicado. En QoS 1 esta bandera no es de interés. El intercambio de paquetes que representado en la Figura F.14

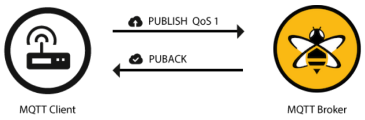


Figura F.14: Flujo para establecer conexión con QoS 1

F.6.3. QoS 2 - Exctamente una vez

Es el nivel mas alto de para garantizar la entrega de mensajes, así como el más lento. Para garantizar que el mensaje fue entregado una única vez se realizar un *handshake* de cuatro etapas entre emisor y receptor. Ambos utilizan el identificador del paquete PUBLISH para coordinar la entrega. Cuando el receptor recibe un PUBLISH que tiene QoS 2, lo procesa y envía un mensaje **PUBREC** que indica que lo recibió. Si el emisor no recibe el PUBREC renvíá el PUBLISH pero esta vez con la bandera DUP seteada como verdadera hasta que reciba el PUBREC. Una vez que lo recibe al PUBREC lo almacena y descarta el mensaje PUBLISH, además envía con un mensaje **PUBREL**. Cuando el receptor obtiene el paquete PUBREL puede descartar todos los estados almacenados y responde finalmente con un mensaje **PUBCOMP**. Una vez que el emisor recibe el PUBCOMP el identificador del paquete puede ser reutilizado. Los paquetes utilizados en esta transacción así como un esquema del flujo se pueden ver en la Figura F.15.

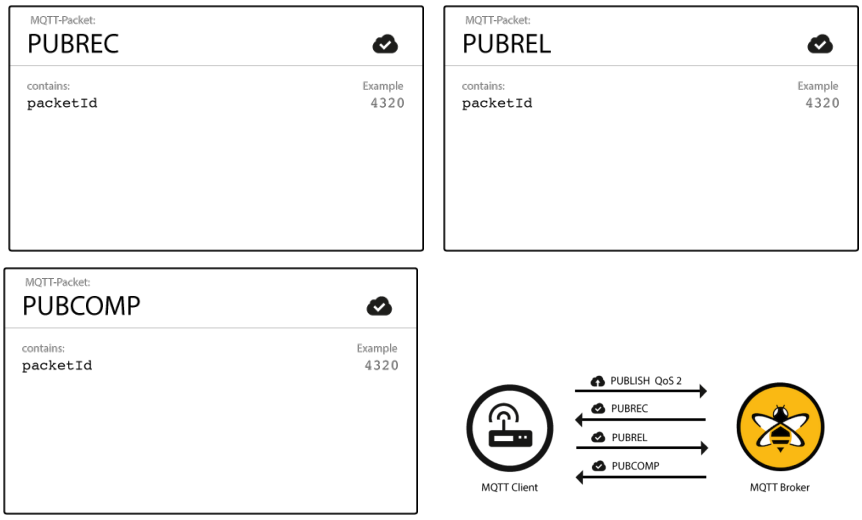


Figura F.15: Paquetes para el handshake de QoS 2

F.7. Persistent Session

Para recibir un mensaje desede el broker, el cliente se debe conectar y suscribirse a los topics de interés. ¿Que ocurre si la conexión entre el cliente y el broker es interrumpida?

Si la sesión es no persistente, el broker no guardará los mensajes recibidos en el topic, por lo cual el cliente no recibirá esos mensajes.

En cambio si la sesión es persistente el broker almacenará los mensajes recibidos, y se los entregará al cliente cuando este se conecte. La persistencia no es únicamente respecto al encolado de los mensajes. Al ser una sesión persistente el broker almacena los topics a los cuales se suscribió el dispositivo. Esto es útil en equipos de bajos recursos. Ya que se evita el intercambio de paquetes necesario para la suscripción.

F.7.1. ¿Que se almacena en una sesión persistente?

En las sesiones persistentes el broker almacena

- Existencia de sesión. Aunque no tenga suscripciones.
- Todas las suscripciones del cliente
- Todos los mensajes perdidos con QoS 1 o 2.[4]
- Todos los mensajes QoS2 que no recibieron confirmación.

F.7.2. ¿Como comenzar/terminar una sesion persistente?

En el momento que el cliente se conecta con el broker puede solicitar una sesión persistente mediante el uso de la bandera *cleanSession*. Cuando la bandera es *true*, el cliente no está solicitando una sesión persistente. Por el contrario se está solicitando una sesión limpia. De haber habido previamente una sesión persistente la información de ésta será desechada.

Si la bandera está configurada como *false* el broker entiende que se desea una sesión persistente. Y se mantendrá como sesión persistente hasta que el cliente solicite una sesión limpia.

F.7.3. ¿Puede el cliente saber si hay una sesió almacenada?

A partir de versión 3.1.1 de MQTT el mensaje *CONNACK* contiene una bandera que indica si existe una sesión persistente.

De manera similar al broker el cliente deberá almacenar la siguiente información acerca la sesión:

- Todos los mensajes con QoS1 y QoS2 que no hayan sido confirmados por el broker.

[4]En Mosquitto se puede configurar para que también almacene los mensajes con QoS0.

- Todos los mensajes QoS2 que no han sido completamente confirmados por el broker.

F.8. RetainedFLag

En MQTT el cliente publica un mensaje pero no hay garantía de que ese mensaje llegue a otro cliente que esté suscrito. Sólo se está seguro de que el mensaje fue entregado al broker.The publishing client can only make sure that the message gets delivered safely to the broker. Entre mensajes publicados en un mismo topic puede pasar, segundos, minutos, horas, días o más aún. Por lo tanto un cliente podría demorar en recibir un mensaje en el topic que se suscribió. Para evitar esto, y que el cliente pueda estar seguro de haberse suscrito al topic deseado. O bien para enviar un mensaje de bienvenida, exiten los **mensajes retenidos** ó *retained message*.

Estos son mensajes normales de MQTT pero con la bandera *retainedFlag* indicada como *true*. El broker almacenará el último *retained message* recibido y lo entregará a cada cliente que se suscriba al topic. El broker sólo almacena un mensaje retenido por topic.

En caso de que el cliente utilice *wildcards* al momento de la suscripción, recibirá los mensajes retenidos de todos los canales que tengan uno. El broker envía el mensaje con la bandera *retained flag* indicando *true* si es un mensaje retenido ó indicando *false* en caso que no lo sea. Por lo cual es posible identificar desde el cliente si es un mensaje retenido o no.

Es importante entender que los mensajes retenidos no están relacionados con las sesiones persistentes. En el primer caso son mensajes que quedan almacenados en el broker y se envían en cada cliente que se suscribe al topic y la única forma de borrarlo es enviando un mensaje vacío marcado como retenido. Mientras que el segundo es una sesión de todos los topics y mensajes del cliente en cuestión.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice G

DHT22

El DHT22 [3] es un sensor humedad relativa y temperatura. En esta sección se hace una descripción breve de funcionamiento y sus pines.

G.1. Descripción

El sensor DHT22, que se muestra en la figura G.1, consiste en tres partes: un sensor de humedad relativa, un sensor de temperatura NTC (termistor) y un circuito integrado que se encargado de la interfaz digital.

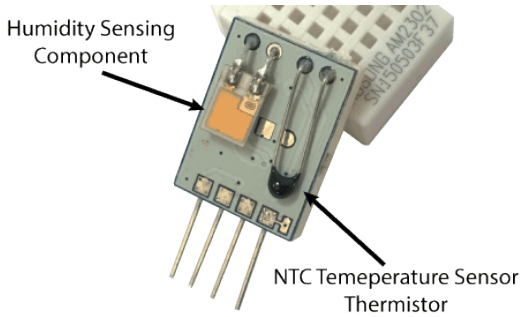


Figura G.1: Sensor DHT22 por dentro.

G.2. Características

Características del sensor:

- Rango de temperatura: De $-40 \circ C$ a $125 \circ C \pm 0{,}5 \circ C$.
- Rango de humedad: De $0\,\%HR$ a $100\,\%HR \pm 5\,\%HR$.
- Frecuencia de muestreo: $0{,}5Hz$.
- Consumo: $2{,}5mA$.
- Voltaje de operación: De $3V$ a $5V$.

G.3. Pines

El sensor tiene cuatro pines:

1. VCC: Alimentación.
2. Data Out: bus bidireccional utilizado para transmitir los datos.
3. No conectado
4. GND : Tierra.

G.4. Interfaz serial

La comunicación entre el microcontrolador y el sensor se da únicamente a través del pin Data Out que funciona como bus bidireccional. En la figura G.2 se muestra el proceso de transmisión completo del DHT22.

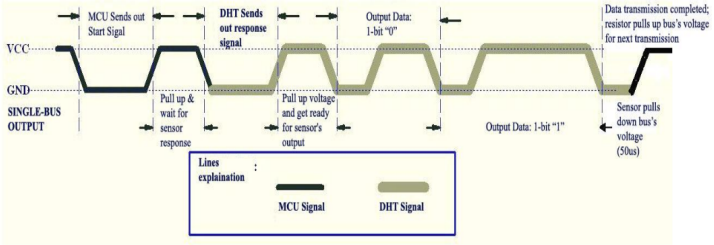


Figura G.2: Proceso de transmisión completo del DHT22.

Inicialmente el bus está en 1, que es su estado de reposo y se encuentra libre de datos. El sensor se encuentra en modo de bajo consumo. El microcontrolador inicia la comunicación poniendo a 0 el bus y luego volviendo a 1 y espera a que el sensor complete la señal de inicio. El sensor pasa a modo normal y responde al microcontrolador poniendo a 0 y luego a 1 nuevamente. La figura G.3 muestra esto con más detalle.

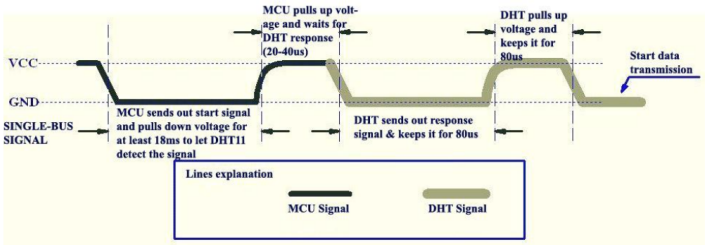


Figura G.3: Secuencia de inicio de transmisión del DHT22.

De esta forma el bus está listo para la transmisión de datos. El sensor procede a enviar 40 bits de la siguiente forma:

- Transmisión de un 0: 50μs el bus en 0 y 26μs-28μs el bus en 1.
- Transmisión de un 1: 50μs el bus en 0μs y 70μs el bus en 1.

En las figuras G.4 y G.5 se puede observar el diagrama de tiempos para cada caso.

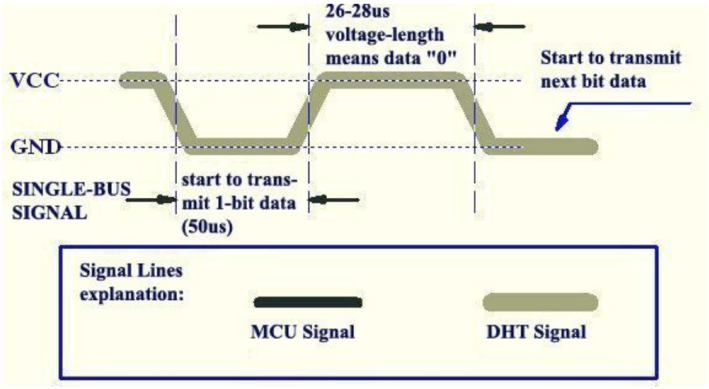


Figura G.4: Transmisión de un cero del DHT22.

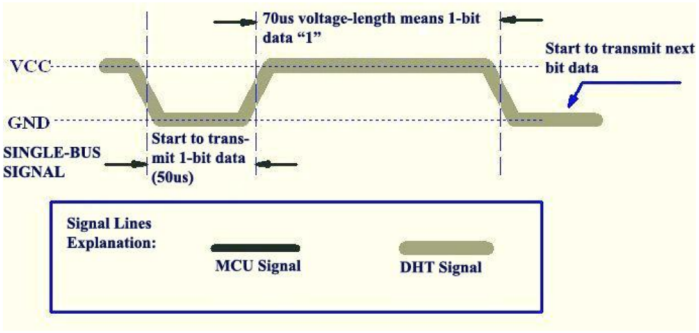


Figura G.5: Transmisión de un uno del DHT22.

Si durante la transmisión el bus está siempre en 1 quiere decir que hay un error.

El proceso completo de transmisión demoran un par de milisegundos pero se debe esperar al menos un segundo antes de comenzar de nuevo para que el sensor se estabilice.

Una vez transmitidos los datos el sensor vuelve al modo de bajo consumo, sin importar si el microcontrolador leyó los datos, y espera a que el microcontrolador inicie la secuencia de transmisión.

Los datos constan de partes decimales e enteras. Una transmisión de datos completa es de cinco bytes (40 bits), y el sensor envía primero el bit más significativo.

El formato de datos enviados es el siguiente:

1. Parte entera de la humedad relativa.
2. Parte decimal de la humedad relativa.
3. Parte entera de la temperatura.
4. Parte decimal de la temperatura.
5. Control: suma de los bytes anteriores.

Apéndice H

HX711

El HX711 [37] es un circuito integrado que tiene como funcionalidad la amplificación y conversión analógica-digital de señales escalares, en particular se utilizará en el circuito que sensará el peso. En esta sección se hace una descripción breve de funcionamiento y sus pines.

H.1. Descripción

Este integrado es un ADC de 24 bits diseñado para la confección de balanzas a partir de galgas extensiométricas configuradas como un puente de Wheatstone. Está compuesto por dos canales diferenciales, de bajo ruido, uno con dos ganancias programables y uno con ganancia fija. No requiere de programación y se puede controlar directamente desde los pines. Tiene un protocolo propio para la transmisión de datos.

Internamente se compone de los siguientes bloques:

- Multiplexor para los canales analógicos de entrada.
- Amplificador de ganancia programable.
- Conversor analógico-digital Sigma-Delta de 24 bits.
- Interfaz digital de salida.
- Regulador.
- Oscilador.

En la figura H.1 se puede ver como se organizan dichos bloques con el integrado configurado con una aplicación típica.

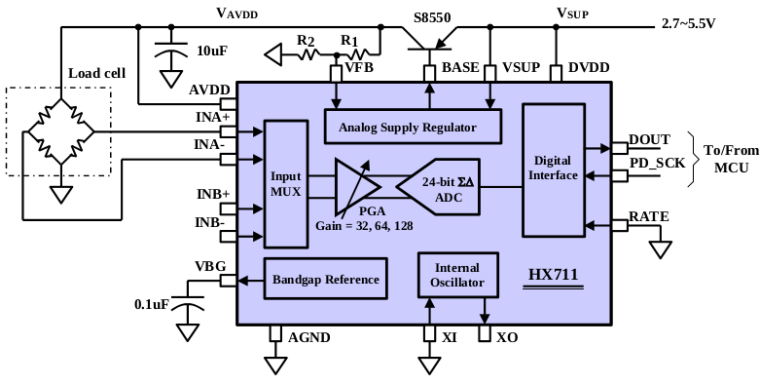


Figura H.1: Diagrama de bloques del HX711

H.2. Características

El circuito integrado presenta las siguientes características:

- Dos canales seleccionables, con ganancias programables (32-64-128).
- Frecuencia de transmisión de datos seleccionable (10sps-80sps)
- El consumo nominal es de 1.5 mA (máximo) en operación normal y 1 uA (máximo) en bajo consumo.
- Tiene integrado un regulador y y un oscilador por lo que elimina la necesidad de un reguladores y osciladores externos.
- El rango del voltaje de operación es de 2.6V - 5.5V.

H.3. Pines

El integrado cuenta con 16 pines:

1. VSUP: Voltaje alimentación del regulador.
2. BASE: Salida analógica. No conectada cuando no se utiliza.
3. AVDD: Alimentación analógica.
4. VFB: Entrada analógica. Entrada de control del regulador. Conectada a GND cuando no se utiliza.
5. AGND: Tierra analógica.
6. VBG: Entrada analógica. Referencia de salida de bypass.
7. INA-: Entrada analógica. Entrada negativa de canal A.
8. INA+: Entrada analógica. Entrada positiva de canal A.

9. INB-: Entrada analógica. Entrada negativa de canal B.
10. INB+: Entrada analógica. Entrada positiva de canal B.
11. PDSCK: Entrada digital. Control de bajo consumo y reloj serial.
12. DOUT: Salida digital. Salida serial de datos.
13. XO: Entrada/Salida digital. Conectada al oscilador externo. No conectada cuando no se utiliza.
14. XI: Entrada digital.
15. RATE: Entrada digital. Control de frecuencia de salida de datos. Usando el oscilaodr interno: 0 lógico para frecuencia de 10Hz y 1 lógico para 80Hz. Usando un oscilador externo se puede aumentar dichas frecuencias.
16. DVDD: Alimentación digital.

En la figura H.2 se puede ver el integrado y la disposición de sus pines.

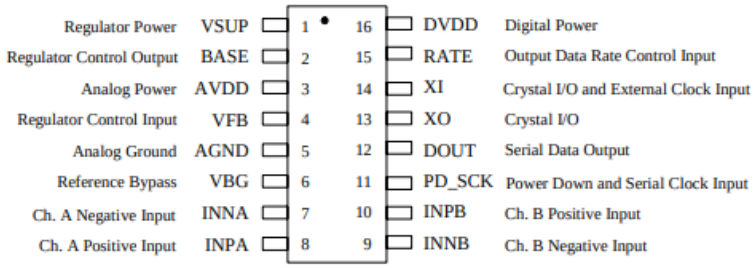


Figura H.2: Integrado HX711

H.4. Interfaz Serial

Para la comunicación serial son utilizados los pines PDSCK y DOUT. La transmisión se da mediante la siguiente secuencia:

Cuando los datos no están prontos la salida DOUT se mantiene en 1. PDSCK debe estar en 0.

Cuando DOUT baja significa que los datos están prontos para transmitirse.

Aplicando 24 pulsos positivos en PDSCK se transmiten los 24 bits desde el más significativo al menos significativo. Luego de esa secuencia de pulsos se transmiten entre 1 y 3 pulsos más que le indican la ganancia de la próxima medida como se indica en la tabla H.1.

La secuencia completa se puede apreciar en la figura H.3.

Si PDSCK se sube a 1 y se mantiene por más de 60us el integrado pasa al modo de bajo consumo hasta que PDSCK baje a cero y retorna al modo normal como se puede observar en la figura H.4.

Pulsos	Canal	Ganancia
1	A	128
2	B	32
3	A	64

Tabla H.1: Codificación de la ganancia de los canales mediante pulsos en el HX711.

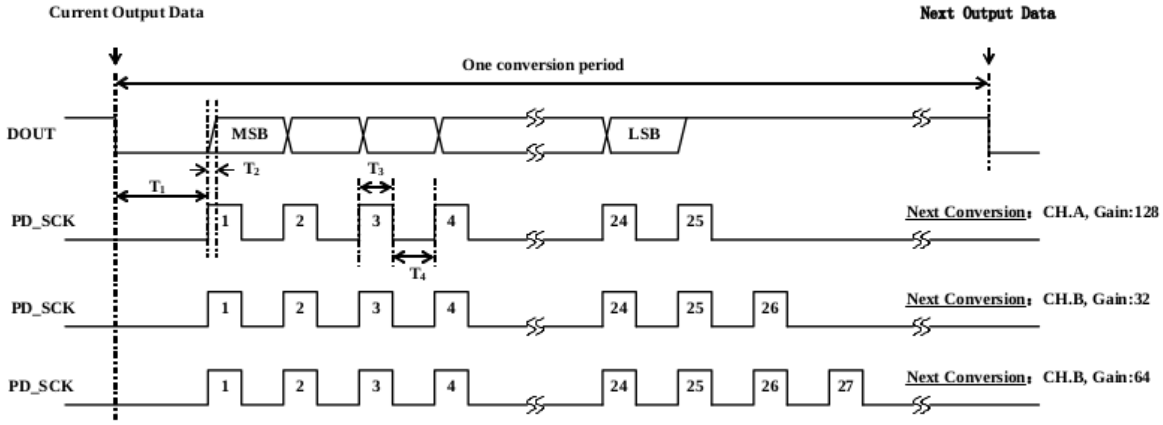


Figura H.3: Diagrama de tiempos de interfaz serial del HX711.

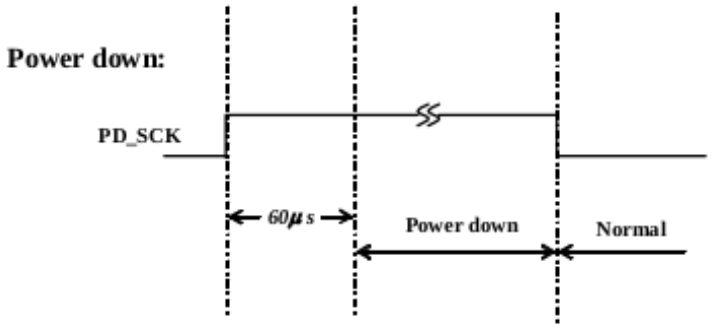


Figura H.4: Bajo consumo del HX711

Apéndice I

Contenido de la entrega

El contenido está seccionado en:

- **Documentación**: Documentación del proyecto.
- **Manuales**: Manuales de usuario y técnico del equipo.
- **Esquemáticos**: Esquemáticos y PCB de los shields circuitos.
- **Códigos**: Archivos fuentes del firmware de los módulos y *scripts* del servidor

En la figura I.1 se muestra la estructura de la entrega realizada.

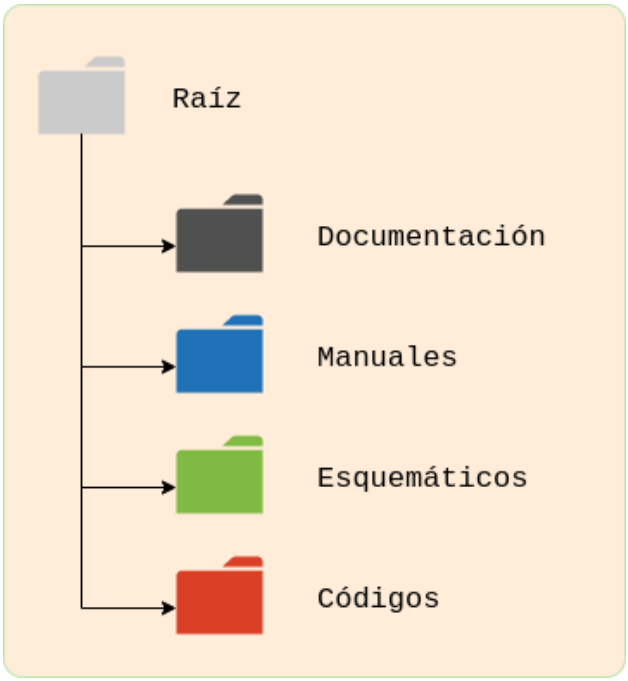


Figura I.1: Estructura del contenido de la entrega

El código desarrollado se encuentra también en el repositorio del proyecto [36].

Esta página ha sido intencionalmente dejada en blanco.

Referencias

[1] Nicolás Obrusnik y Pablo Zinemanas Alejandro Draper. *Pestibee*. Facultad de Ingeniería UDELAR, Montevideo, Uruguay, 2015.

[2] ANTEL. Cobertura movil antel. http://www.antel.com.uy/principal?p_p_id=101&p_p_lifecycle=0&p_p_state=maximized&p_p_mode=view&_101_struts_action=%2Fasset_publisher%2Fview_content&_101_assetEntryId=276464&_101_type=content&_101_urlTitle=cobertura-movil-lte-3-7g-gsm&inheritRedirect=false.

[3] Ltd Aosong Electronics Co. Dht22 datasheet. https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf.

[4] Lara Aratani. *This article is more than 2 months old Pesticide widely used in US particularly harmful to bees*. The Guardian, New York, USA, 6 Agosto 2019.

[5] Arduino. Sd. https://www.arduino.cc/en/Reference/SD.

[6] Arduino. Wire. https://www.arduino.cc/en/Reference/Wire.

[7] Diego Belotti. Fiware - estrategia antel en iot. http://veramas.com.uy/veramas/vod/38869/973.

[8] Bodge. Arduino library for hx711. https://github.com/bogde/HX711.

[9] Philippe Boudot. Mkr vidor 4000 : Programmation du fpga. https://systemes-embarques.fr/wp/archives/mkr-vidor-4000-programmation-du-fpga-partie-1/.

[10] DFROBOT. Solar power manager for 12v lead-acid battery. https://www.dfrobot.com/product-1795.html.

[11] Docker. Enterprice container platform — docker. https://www.influxdata.com/.

[12] Jonny Doin. Spi master/slave interface. https://opencores.org/projects/spi_master_slave.

[13] Elecrow. Sim800c hardware design. https://www.elecrow.com/download/SIM800C_Hardware_Design_V1.02.pdf.

[14] Elementzone. Sim800_mqtt. https://github.com/elementzonline/SIM800_MQTT.

[15] FreeRTOS. The freertos^{TM} kernel. https://freertos.org/.

[16] FreeRTOS. Influxdb: Purpose-built open source time series database — influxdata. https://www.docker.com/.

[17] William Greiman. Arduino fat16/fat32 library. https://github.com/greiman/SdFat.

[18] Prentice Hall. *Oppenheim, Discrete-Time Signal Processing*. 1999.

[19] INIA. Humedad relativa media mensual (%). http://www.inia.uy/gras/Clima/Caracterizaci%C3%B3n-agroclim%C3%A1tica/Humedad-relativa.

[20] INIA. Temperatura del aire. http://www.inia.uy/gras/Clima/Caracterizaci%C3%B3n-agroclim%C3%A1tica/Temperatura-del-aire.

[21] Texas Instruments. Stellaris - lm4f232 evaluation board - user's manual. http://www.ti.com/lit/ug/spmu272/spmu272.pdf.

[22] Texas Instruments. Understanding the i2c bus. Technical report, 2015.

[23] D. S. Jacob S. Ishay. *The sounds of honey bees and social wasps are always composed of a uniform frequency*. Acoustical Society of America, 1982.

[24] KiCad. Kicad. http://www.kicad-pcb.org.

[25] Gabriel Lavoura. Mqtt-and-sim800-900. https://github.com/Gabriellavoura/Mqtt-and-Sim800-900.

[26] Andrés Corchs y Hernán Pereyra Leopoldo Agorio. *Urbanear : Monitoreo sonoro urbano de bajo costo*. Facultad de Ingeniería UDELAR, Montevideo, Uruguay, 2019.

[27] Roger Light. Mqtt version 3.1.1 client class. https://pypi.org/project/paho-mqtt/.

[28] D. Linden and T. B. Reddy. *Handbook of Batteries*. McGraw-Hill, 3rd edition, 2001.

[29] J. L. M. D. Atauri Mezquida. *Platform for bee-hives monitoring based on sound analysis*. Spanish Journal of Agricultural Research, 2009.

[30] misja.com. Convertir epoch - tiempo unix. https://espanol.epochconverter.com/.

[31] F. Jesús C. Pérez L. C. Letelier Y. Mendoza N. Pérez, S. Niell and S. Díaz-Cetti. *Caracterización acústica de la colmena para la detección temprana de contaminación por pesticidas*. Facultad de Ingeniería UDELAR, Montevideo, Uruguay, 2014.

[32] Nelson Enrique Arenas-Suárez Nazly Yolieth Martin-Culma. *Daño colateral en abejas por la exposición a pesticidas de uso agrícola*. Universidad de Cundinamarca, Fusagasugá - Colombia, Noviembre 2017.

[33] Nick O'Leary. Pubsubclient. `https://github.com/knolleary/pubsubclient`.

[34] RAU. Clima del uruguay. `https://www.rau.edu.uy/uruguay/geografia/Uy_c-info.htm`.

[35] Gaye Lightbody Ying Yi Roger Woods, John McAllister. *FPGA-based Implementation of Signal Processing Systems*. Wiley, November 24, 2008.

[36] Sara-Bee. Repositorio del proyecto. `https://gitlab.fing.edu.uy/sarabee/publico`.

[37] AVIA Semiconductor. Hx711 datasheet. `https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf`.

[38] Volodymyr Shymanskyy. Tinygsm. `https://github.com/vshymanskyy/TinyGSM`.

[39] David E. Simon. *An Embedded Software Primer*, pages 115–132. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1999.

[40] David E. Simon. *An Embedded Software Primer*, pages 81–111. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1999.

[41] David E. Simon. *An Embedded Software Primer*, pages 92–103. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1999.

[42] The HiveMQ Team. Mqtt essentials. `https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/`, 2015.

[43] Rob Tillaart. Arduino library for dht22. `https://github.com/RobTillaart/Arduino/tree/master/libraries/DHTlib`.

[44] Rob Tillaart. Energia consumida por raspeberry pi. `https://www.redeszone.net/2017/03/02/energia-consumida-raspberry-pi/`.

[45] Wikipedia. Regla de farrar. `https://es.wikipedia.org/wiki/Regla_de_Farrar`.

[46] Wojciech Zabolotny. Parametrized fft engine. `https://opencores.org/projects/versatile_fft`.

Esta página ha sido intencionalmente dejada en blanco.

Glosario

ADC *Analog to Digital Conversor*, Conversor Analógico Digital. 4, 49, 50, 67, 81, 175

AT Estándar abierto de comandos para configurar. 85

ATX *Advanced Technology eXtended*, estándar para fuentes de alimentación de PC. 11

backend Aplicaciones que no interactúan directamente con el usuario . 13

broker Intermediario. Recibe y gestiona la entrea de mensajes. xiii, xiv, 12, 16, 87, 96, 110, 113, 115, 116, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 168, 169, 188, 193

CAN *Controller Area Network* Bus CAN. 14

CMOS *Complementary metal-oxide-semiconductor* Semiconductor complementario de óxido metálico. 14

dashboard Del inglés tablero. Se utiliza como término para definir el área donde se despliega información relevante en una interfaz gráfica. Por ejemplo un tablero donde se despliegan gráficas o tablas relevantes. 116, 117, 192

DDNS *Dynamic DNS*, DNS dinámico. 115

DNS *Domain Name System*, Sistema de nombres de dominio. 185

EEPROM *Electrically Erasable Programmable Read-Only Memory.* Tipo de memoria ROM que puede ser programada, borrada y reprogramada eléctricamente. 33

Epoch Cantidad de segundos transcurridos desde el 1/1/1970 00:00:00Z [30]. 112

FAT *File Alocation Table.* Arquitectura para el sistema de archivos. 84

FFT *Fast Fourier Transfrom* Transformada rápida de Fourier. 4, 6, 55, 56, 58, 59, 60, 61, 63, 64, 78, 80, 81, 191

FPGA *Field Programmable Gate Array*, es hardware programable a través de compuertas lógicas. xii, xiv, 6, 49, 50, 51, 52, 55, 58, 59, 60, 61, 62, 63, 66, 78, 82, 104, 153, 154, 189, 193

frontend Aplicaciones que permiten la interacción con el usuario. 13

GND *Ground*. Refencia eléctrica para 0V, usualmente denominada *tierra*. 15

GPRS *General Packet Radio Service*, servicio general de paquetes vía radio. vii, xi, xii, 3, 10, 15, 84, 85, 93, 105, 119, 133, 192

HDL *Hardware Description Language*. 51, 188

HTTP *Hypertext Transfer Protocol*. Protocolo de trasferencia de hipertexto utilizado para comunicaciones de internet. 85

I2C *Inter-Integrated Circuit*, es un bus serie de datos, utilizada para la intercomunicación entre módulos. vii, xi, xiv, 3, 6, 10, 11, 14, 15, 19, 20, 21, 23, 25, 32, 37, 38, 45, 47, 48, 50, 54, 55, 56, 73, 77, 87, 91, 93, 95, 99, 141, 142, 143, 144, 191, 192

IDE *Integrated Development Environment*. Entorno de desarrollo integrado. 13, 14, 132

InfluxDB Base de datos optimizada para el almacenamiento y manejo de series temporales. vii, 12, 13, 112, 113, 114, 115, 116

INIA Instituto Nacional de Investigación Agropecuaria. 32

IO *Input / Output*, Entrada / Salida. 33

IoT *Internet of Things*. Internet de las cosas, concepto que se refiere a una interconexión digital de objetos cotidianos con internet. 13, 15, 16, 85, 111, 122

IP *Internet Protocol*, protocolo de Internet. 15, 85, 159

IP Core *Intellectual Property Core*. Core bajo propiedad intelectual. 63, 64

ISR *Interrupt Service Routine*, Rutina de Atención a Interrupciones. 137, 138, 139

JSON *JavaScript Object Notation*. Formato de archivo estandar abierto open-standard usado en texto leíble por humanos para la transmisión de objectos que consisten en parejas de atributo-valor o arreglos de datos.. 13, 111, 112, 162, 163, 192

LED *Light-emitting diode*, Diodo emisor de luz. 43

LoRa *Long Range*. Tecnología utlizada para redes de sensores inalámbricos. 15

LTE *Long term evolution*, estándar para comunicaciones inalámbricas. 15

MA *Módulo de Alimentación* . 105, 119, 124, 127, 189

MASE *Módulo de Adquisición de Señales Escalares* . 11, 31, 32, 43, 55, 56, 91, 92, 96, 98, 99, 104, 105, 124, 127, 189, 191

MASV *Módulo de Adquisición de Señales Vectoriales* . 6, 11, 49, 73, 74, 80, 81, 96, 97, 98, 104, 120, 125, 127, 129, 134, 189

microSD Formato para tarjetas de memoria flash para el almacenamiento en dispositivos electrónicos. XII, 84, 93, 105

MongoDB Base de datos para el almacenamiento de JSONs. VII, 12, 13, 111, 112, 114, 115, 116

MP *Módulo Principal* . 105, 126, 127, 129, 189

MPPT *Maximum Power Point Tracking*, seguimiento del punto de máxima potencia. 97, 100

MQTT *Message Queue Telemetry Transport*. Protocolo ultraliviano de transporte de mensajes del tipo publicación/subscripción. VII, XI, XIII, XIV, 12, 15, 16, 17, 85, 86, 87, 89, 91, 92, 96, 97, 98, 110, 111, 113, 115, 116, 122, 155, 156, 157, 158, 159, 160, 161, 162, 164, 166, 168, 169, 187, 189, 192, 193

NB-IoT *Narrow Band IoT*, es un estándar LPWAN. 15

PAHO Nombre del proyecto llevado adelante por la compañía Eclipse, mediante el cual se libreró el protoclo MQTT. En Maorí significa *broadcast*. 113, 114, 116

PCB *Printed Circuit Board*, Placa de circuito impresa. XII, 43, 72, 73, 74, 94, 125, 126, 127, 179, 191, 192

PWM *Pulse-width modulation*. Modulación por ancho de pulso. 33, 43

QoS *Quality of Service*, Calidad de servicio. 16, 168, 169

RAM *Random Access Memory*. Memoria de acceso aleatorio. 24

RS485 Bus RS485 . 14

RTOS *Real Time Operating System*, Sistema Operativo en Tiempo Real, es una arquitectura para sistemas embebidos. 6, 86, 129

SMD *surface-mount device*. Tecnología de Montaje Superficial, . 72, 93

SPI *Serial Peripheral Interface*: Interfaz de periféricos serial. 14, 50, 53, 54, 55, 58, 59, 60, 62, 63, 66, 82, 84, 93, 191

SRAM *Static Random Access Memory*. Memoria RAM estática, que no necesita circuito de refresco. 33, 49, 50, 84

SSCMP *SARA Simple Communication and Management Protocol*, Protocolo de comunicación y manejo desarollado por SARA. 6, 12, 19, 28, 37, 38, 87

Systemd Suite de software que provee servicios para el funcionamiento de un sistema operativo Linux. 114

TCP *Transmission Control Protocol*, Protocolo de Control de Transmisión. 15, 85, 97, 159, 166

topic Canales gestionados por el broker y utilizados por los clientes, . XIV, 12, 16, 87, 89, 110, 113, 114, 115, 116, 156, 157, 159, 160, 161, 163, 164, 165, 166, 168, 169

UART *Universal Asynchronous Receiver-Transmitter*, Transmisor-Receptor Asíncrono Universal, es el dispositivo que controla los puertos y dispositivos serie. 37, 38, 45, 131

VHDL Concatenación de VHSIC y HDL. 51, 58, 59, 153, 191

VHSIC *Very High Speed Integrated Circuit*. 188

Web Service Servicios instalados en un sistema operativo que permiten ser accedidos desde un navegado web. VII, 12, 13, 112, 113, 115, 116, 132, 133

Wiring Plataforma de prototipado electrónico de fuente abierta. 13

Zigbee Protocolo de comunicación de alto nivel, utilizado en redes de sensores inalámbricos. 4

Índice de tablas

2.1. Bus de datos y alimentación unificado. 11

4.1. Regla de Farrar para la estimación del peso de la colmena. 32
4.2. Prueba de la balanza del MASE 45

5.1. Comparación de plataformas para MASV 49
5.2. Pines interconexión microcontrolador - FPGA 59
5.3. Comandos del protocolo de comunicación microcontrolador - FPGA 60
5.4. Significado del byte de status 61
5.5. Frecuencias máximas de funcionamiento en sistema VHDL 67
5.6. Simulación etapa amplificadora 71
5.7. Escalas del osciloscopio para pruebas en las placas finales 75
5.8. Resultados etapa amplificadora 75
5.9. Validación: frecuencia de corte 77
5.10. Validación: punto de operación 77
5.11. Verificación del diseño en FPGA con muestras periódicas 78

6.1. Tabla de comparación de Hardware Modulo Principal 83
6.2. Comandos recibidos por MQTT 91
6.3. Voltaje teórico y obtenido para el regulador lineal. 96

7.1. Consumo del equipo en 5V. 105
7.2. Consumo del equipo en 12V. 105
7.3. Estimación de duración de batería. 106
7.4. Eficiencia en salida de 12V. 107
7.5. Eficiencia en salida de 5V. 107

9.1. Datos relevados en colmena de abeja 120

10.1. Costos de Materiales del MA 124
10.2. Costos de Materiales del MASE 124
10.3. Costos de Materiales del MASV 125
10.4. Costos de Materiales del MP 126
10.5. Costos de Materiales del equipo 127
10.6. Presupuesto total 128
10.7. Evaluación de 5 riesgos principales. 129

Índice de tablas

F.1. Tabla de códigos de retorno de CONNACK 162
F.2. Códigos de retorno en SUBACK 165

H.1. Codificación de la ganancia de los canales mediante pulsos en el HX711. 178

Índice de figuras

1.1. Espectros y espectrogramas del audio para tres momentos del día. 5

2.1. Arquitectura del sistema. 9
2.2. Conector GX16-7p utilizado para el bus unificado. 12

3.1. Estructura de los paquetes del protocolo. 22
3.2. Ejemplo de inicialización del protocolo. 27
3.3. Diagrama de tiempos de la adquisición. 27
3.4. Ejemplo de adquisición del protocolo. 28

4.1. Arduino UNO utilizada para el desarrollo del MASE. 33
4.2. Sensor de humedad relativa y temperatura DHT22. 34
4.3. Celda de carga utilizada. 35
4.4. Equivalente eléctrico de la celda de carga. 35
4.5. Módulo HX711 36
4.6. Diagrama del Round-Robin del MASE. 39
4.7. Sensor de humedad relativa y temperatura DHT22. 40
4.8. Esquema de conexión del puente usando una y dos celdas de carga. 41
4.9. Esquema de conexión del puente usando cuatro galgas. 41
4.10. Imagen de la balanza implementada. 42
4.11. Imagen del PCB del MASE. 43
4.12. Imágenes del MASE implementado. 43

5.1. Señal de abejas obtenidas del proyecto PestiBee 48
5.2. Arduino MKR Vidor 4000 50
5.3. Micrófono electret 52
5.4. Conexión micrófono electret 53
5.5. Módulo Micro SD para almacenamiento de datos local 54
5.6. Convertidor de nivel de voltaje, para la comunicación I2C 54
5.7. Diagrama de bloques Módulo Adquisición Señales Vectoriales . . . 56
5.8. Diagrama del Round-Robin del módulo de adquisición de señales vectoriales. 57
5.9. Diagrama de bloques del código en VHDL 59
5.10. Protocolo de comunicación microcontrolador - FFT 61
5.11. Orden de bytes para ciclos de lectura y escritura 62
5.12. Diagrama de tiempos del core SPI - simulación desde Quartus . . . 63

5.13. Diagrama de estados del bloque Intérprete de comandos 64
5.14. Simulación del código de la FPGA en Quartus para 16 muestras . 65
5.15. Dip switch para modificar ganancias variables 68
5.16. Etapa de amplificación, ganancia fija, y variable 68
5.17. Etapa de filtrado, Butterworth de segundo orden 70
5.18. Circuito final de acondicionamiento de señales vectoriales 70
5.19. A set of four subfigures. 71
5.20. Simulación respuesta en frecuencia 72
5.21. PCB y modelo 3D para el acondicionamiento de señales de audio . 72
5.22. PCB y modelo 3D del Shield para los 4 módulos de audio 73
5.23. PCB y modelo 3D del Shield de Arduino MKR Vidor 4000 más componentes 74
5.26. Respuesta en frecuencia del circuito real 76
5.27. Frecuencia de corte real 76
5.28. Señal SDA a través del convertidor de nivel de voltaje para la comunicación I2C 77
5.29. Señal reconstruida a una frecuencia de muestreo de 4kHz 79
5.30. A set of four subfigures. 80
5.31. Promedio del módulo por bandas para un seno de 500Hz 80
5.33. MASV en su caja estanca con los 4 canales de audio y conectores del BUS 81

6.1. Módulo de lectura/escritura SD 85
6.2. Módulo SIM800 - GPRS 85
6.3. Flujo del módulo principal 88
6.4. Flujo del escaneo de módulos esclavos 89
6.5. Flujo de la función utilizada para adquirir medidas en un canal . . 90
6.6. Diagrama de flujo en la recepción de medidas 91
6.7. Flujo dentro de MQTT callback 92
6.8. Circuito de regulación de voltaje para el SIM800 93
6.9. Diseño y ensamble de shield para módulo principal 94
6.10. Elección de la resistencia de *pull-up* para las lineas I2C. 95
6.11. Pruebas de validación para el módulo principal 98

7.1. Placa de gestión de panel solar y batería del MA. 101
7.2. Batería del MA. 103
7.3. Panel solar del MA 104
7.4. Consumo en tiempo real del equipo en 12V 106

8.1. Ejemplo de la estructura de JSON 111
8.2. Flujo del cliente SARA 114
8.3. Dashboard de representaciones en Chronograf 117
8.4. Selección de la base de datos y armado de la consulta en Chronograf 117

9.1. Imágenes de la prueba del equipo en campo. 121

10.1. Dedicación mensual grupal durante el proyecto. 130

B.1. Estructura open-drain 142
B.2. Condiciones de START y STOP 142
B.3. Maestro transmisor - Esclavo receptor 144
B.4. Maestro receptor - Esclavo transmisor 145

C.1. Filtro Butterworth de segundo orden 147
C.2. Órdenes filtro Butterworth 148

D.1. Método radix 2 para ocho muestras 150
D.2. Algoritmo no recursivo 151

E.1. Etapas de diseño para una FPGA 154

F.1. Ejemplo del esquema de publicación y suscripción 157
F.2. Flujo de conexión MQTT 159
F.3. Mensaje del cliente para conexión MQTT "CONNECT" 160
F.4. CONNACK - Respuesta del broker para el mensaje CONNECT . 161
F.5. Paquete de publicación 162
F.6. Flujo de una publicación 163
F.7. Flujo de paquetes al realizar una suscripción 164
F.8. Paquete de suscripción a tópico 164
F.9. Paquete de confirmación de suscripción 164
F.10. Paquete de dessuscripción a tópico 165
F.11. Paquete de confirmación de dessuscripción 165
F.12. Flujo de paquetes al realizar una dessuscripción 166
F.13. Flujo para establecer conexión con QoS 0 166
F.14. Flujo para establecer conexión con QoS 1 167
F.15. Paquetes para el handshake de QoS 2 167

G.1. Sensor DHT22 por dentro. 171
G.2. Proceso de transmisión completo del DHT22. 172
G.3. Secuencia de inicio de transmisión del DHT22. 172
G.4. Transmisión de un cero del DHT22. 173
G.5. Transmisión de un uno del DHT22. 173

H.1. Diagrama de bloques del HX711 176
H.2. Integrado HX711 177
H.3. Diagrama de tiempos de interfaz serial del HX711. 178
H.4. Bajo consumo del HX711 178

I.1. Estructura del contenido de la entrega 179

Esta es la última página.
Compilado el jueves 27 febrero, 2020.
http://iie.fing.edu.uy/