Universidad Tecnológica Nacional

Proyecto Final

Datalogger Universal

Autores:

- *Cargnel, Pablo.*

Director:

- *Yarce, Gustavo.*

Proyecto final presentado para cumplimentar los requisitos académicos para acceder al título de Ingeniero Electrónico

en la

Facultad Regional Paraná

Agosto de 2024

Declaración de autoría:

Yo declaro que el Proyecto Final “Datalogger Universal” y el trabajo realizado son propios. Declaro:

- Este trabajo fue realizado en su totalidad, o principalmente, para acceder al título de grado de Ingeniero electrónico, en la Universidad Tecnológica Nacional, Regional Paraná.
- Se establece claramente que el desarrollo realizado y el informe que lo acompaña no han sido previamente utilizados para acceder a otro título de grado o pre-grado.
- Siempre que se ha utilizado trabajo de otros autores, el mismo ha sido correctamente citado. El resto del trabajo es de autoría propia.
- Se ha indicado y agradecido correctamente a todos aquellos que han colaborado con el presente trabajo.
- Cuando el trabajo forma parte de un trabajo de mayores dimensiones donde han participado otras personas, se ha indicado claramente el alcance del trabajo realizado.

Firmas:

-

Fecha: xx de xxxxxx de 202x

Agradecimientos:

En primer lugar, me gustaría agradecer a Gustavo Yarce por la ayuda y la predisposición en la elaboración de este proyecto final. En segundo lugar, agradecer a la Universidad Tecnológica Nacional y sus docentes, por brindarme una educación gratuita, laica, y de calidad durante todos estos años, permitiéndome desarrollar como Ingeniero y como persona, y a la empresa AutoSolve por brindarme su ayuda y darme un espacio para poder finalizar mi carrera. En tercer lugar, y no por eso menos importante, quería agradecer a mi familia que me apoyo todos estos años económica y emocionalmente, a mis amigos que sin su ayuda incondicional hubiera sido todo más difícil, y a mi pareja, que, en el tramo final, fue el soporte necesario para poder finalizar este proyecto.

Cargnel, Pablo.

Universidad Tecnológica Nacional

Abstract

Facultad Regional Paraná

Ingeniero en Electrónica

DATALOGGER UNIVERSAL

Cargnel, Pablo

Abstract:

This project presents the design and implementation of a universal datalogger with four channels capable of collecting data based on common electrical signals used in the field: 4-20 mA current loop, 0-20 mA current loop, 0-10 V voltage input, and constant voltage pulses. The device supports two operational modes: scheduled measurement intervals and pulse-based activation. In the scheduled mode, users can set the start time and duration of data logging sessions. In pulse-based activation, the device triggers data logging in response to external pulses. Data is stored on a microSD card and can be downloaded in a ".txt" format via a web server. Additionally, the web server allows configuration of operational modes and parameters that are essential for proper functioning.

The microcontroller used in this project is the ESP32, which communicates with an external Real-Time Clock Calendar (RTCC) via the I2C protocol to obtain accurate date and time information. Data is saved to a microSD card using the SPI protocol. Programming was done in C++ using the Arduino framework within Visual Studio Code. The web server interface was developed using HTML and JavaScript.

When configured in operational mode number 1, this device can collect data with a sampling frequency ranging from 1 to 60 seconds, for up to one hour. Equipped with an

external ADC ADS1115, the datalogger features a 16-bit resolution, capable of detecting changes as small as 0.125 mV per bit.

Keywords:

ADS1115, ESP32, data logging, webserver.

Resumen:

Este proyecto presenta el diseño e implementación de un datalogger universal con cuatro canales capaz de recolectar datos basados en señales eléctricas comunes utilizadas en el campo: lazo de corriente de 4-20 mA, lazo de corriente de 0-20 mA, entrada de voltaje de 0-10 V y pulsos de voltaje constante. El dispositivo soporta dos modos de operación: intervalos de medición programados y activación basada en pulsos. En el modo programado, los usuarios pueden configurar la hora de inicio y la duración del registro de datos. En la activación basada en pulsos, el dispositivo inicia el registro de datos en respuesta a pulsos externos. Los datos se almacenan en una tarjeta microSD y pueden descargarse en formato ".txt" a través de un servidor web. Además, el servidor web permite la configuración de modos de operación y parámetros esenciales para un funcionamiento adecuado.
El microcontrolador utilizado en este proyecto es el ESP32, que se comunica con un calendario de reloj en tiempo real (RTCC) externo a través del protocolo I2C para obtener información precisa de fecha y hora. Los datos se guardan en una tarjeta microSD utilizando el protocolo SPI. La programación se realizó en C++ utilizando el framework Arduino dentro de Visual Studio Code. La interfaz del servidor web fue desarrollada utilizando HTML y JavaScript.
Cuando se configura en el modo operativo número 1, este dispositivo puede recolectar datos con una frecuencia de muestreo que va desde 1 hasta 60 segundos, durante un máximo de una hora. Equipado con un ADC externo ADS1115, el datalogger cuenta con una resolución de 16 bits, capaz de detectar cambios tan pequeños como 0.125 mV por bit.

Palabras Clave:

ADS1115, ESP32, guardar datos, servidor web.

Índice:

Lista de Figuras:

Lista de Tablas

Lista de Abreviaciones y Símbolos

SD – Secure Digital
PLC – Programmable Logic Controller
USB – Universal Serial Bus
GSM – Global System for Mobile Communication
GPRS – General Packet Radio Service
HMI – Human Machine Interface
ADC – Analog-to-digital converter.
SPI –Serial peripheral interface.
I2C –Inter-integrated circuit.
CAN – Controller Area Network
PWM – Pulse Width Modulation
UART – Universal Asynchronous Receiver Transmitter.
RTC – Real time clock
IP – Internet Protocol
HTML – HyperText Markup Language
HTTP – Hypertext Transfer Protocol
CSV – Comma Separated Value
NTP –Network Time Protocol
EEPROM – Electrically Erasable Programmable Read-Only Memory

Dedicado a:

A mis padres Elizabeth y Norberto, y mis hermanos Cecilia y Juan Ignacio, cuya crianza, compañía y amor incondicional me llevaron al lugar en donde estoy.
A mis amigos, o la familia que elegí, que gracias a ellos la vida se hace más liviana y sencilla.
A mi pareja Daniela, que me brinda su apoyo y amor en los proyectos y objetivos que se plantean día a día.
A Nanci, por ser el apoyo que necesitaba en mi momento más difícil.
A mi abuela Elba, que su compañía en Paraná hizo que sea todo más fácil, y a mi abuela Gabi, gracias por el amor y el apoyo recibido durante todos estos años.
A todas las personas que se cruzaron en mi camino y me ayudaron a ser mejor persona y profesional.

Capítulo 1: Introducción

Un Datalogger o registrador de datos, es un dispositivo electrónico cuya función, como indica su nombre, es registrar y guardar datos. La importancia de registrar datos de sistemas proviene de la necesidad de conocer su comportamiento, ya sea para controlar, modificar, o simplemente obtener una base de datos de su funcionamiento, para su posterior procesamiento.

Por lo general los registradores de datos se dividen en 3 bloques:

1. Obtención de datos
2. Procesamiento de datos
3. Almacenamiento de datos

Según el Datalogger, la obtención de datos puede ser interna o externa al dispositivo, es decir, los sensores que recopilan la información vienen integrados o posee canales en donde se conectan los mismos, donde ingresa algún estándar de señal eléctrica de medición, que puede ser tensión, corriente o simplemente registrar pulsos digitales. Las variables más comunes que se miden en la industria son: temperatura, humedad, presión, caudal, nivel de líquidos o sólidos, etc.

El procesamiento de los datos viene dado por el conversor digital analógico que posea el dispositivo. Lo que define a un ADC es su resolución, y la velocidad a la que puede convertir los datos, es decir, la frecuencia de muestreo. Según la aplicación, se puede necesitar una precisión alta o baja de los datos sensados, esto también define el costo del dispositivo. Por lo general se utilizan microcontroladores que ya poseen ADC integrados, pero es normal utilizar integrados o módulos externos, cuyo único propósito es el de cumplir con la función de ADC, comunicando los datos a través de algún protocolo (I2C, Modbus, ethernet, etc).

Por último, para el almacenamiento de los datos se utiliza una memoria física, como puede ser una tarjeta SD o microSD. Se pueden utilizar, por ejemplo, discos duros, pero esto complejizaría y elevaría el costo del dispositivo innecesariamente. En los últimos años, gracias al avance de la tecnología, es común ver dataloggers que se conectan a internet y guardan la información en una base de datos en la nube y dentro de la misma se procesan y grafican los datos.

En el estudio de mercado realizado se pudo observar que existen distintos tipos de registradores de datos. Las características que los diferencian son: tipo de variable sensada, protocolo de comunicación utilizado, autonomía energética, portabilidad, capacidad de almacenamiento, frecuencia de muestreo, etc., pero todos tienen en común que los datos recopilados se guardan con la fecha y hora en que ocurrió la medición.

El proyecto sobre el cual está escrito el siguiente trabajo, consiste en el desarrollo, programación y elaboración de un sistema de recopilación, almacenamiento y procesamiento de datos, basado en las 4 señales eléctricas de medición más comunes utilizadas en la industria (de ahí el nombre “universal”), siendo estas: lazo de corriente de 4 – 20 [mA] y 0 – 20 [mA], señal de tensión continua en el rango de 0 – 10 [V] y pulsos de valor de tensión constantes (usados generalmente como totalizadores en controladores). Este dispositivo es configurado a través de WiFi, y guarda los datos en archivos de valores separados por coma dentro de una tarjeta microSD, para luego ser procesados.
La idea del desarrollo del proyecto surgió en el año 2020, por parte de la empresa de soluciones tecnológicas Autosolve, ante la necesidad de recopilar datos de un controlador de flujo de una planta potabilizadora de agua en la comuna de Humboldt. Cabe destacar que todo lo presentado en el siguiente trabajo fue cofinanciado por dicha empresa, y la programación de software, desarrollo de placas y armado final de producto fue realizado por quien suscribe.

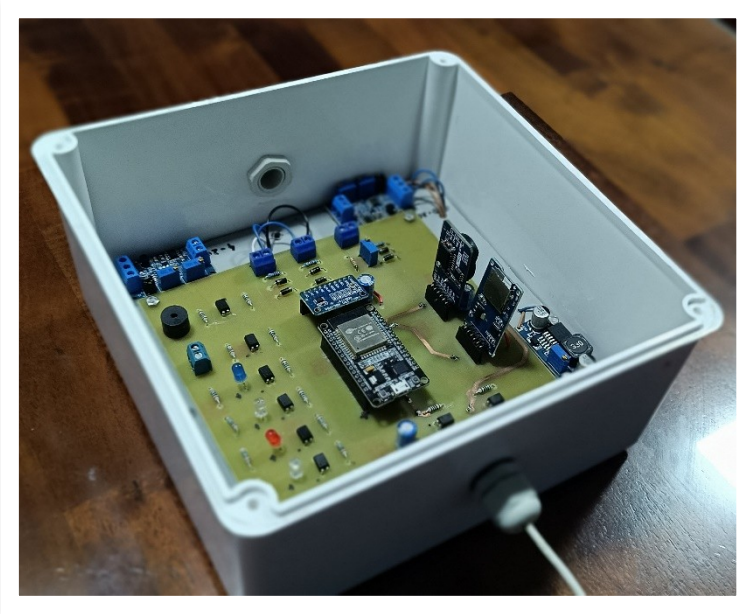

Figura 1 – “Datalogger Universal”

1.1 Fundamentación

A través del desarrollo de este proyecto, se quiere lograr la obtención de un producto versátil, que pueda responder eficientemente en distintos entornos, además de permitirle al usuario ciertas características y flexibilidades a la hora de plantear su sistema de obtención de datos. A través de su conexión inalámbrica, una vez instalado, puede ser configurado sin tener que quitarlo de su lugar, y sus múltiples entradas permiten flexibilidad dependiendo el tipo de sensor que se utiliza. Gracias a estas características, este producto podrá ser emplazado en casi cualquier sistema de sensado, y aun si el mismo cambiase, puede adaptarse debido a las características previamente mencionadas.

Capítulo 2: Desarrollo

2.1 Diagrama de bloques

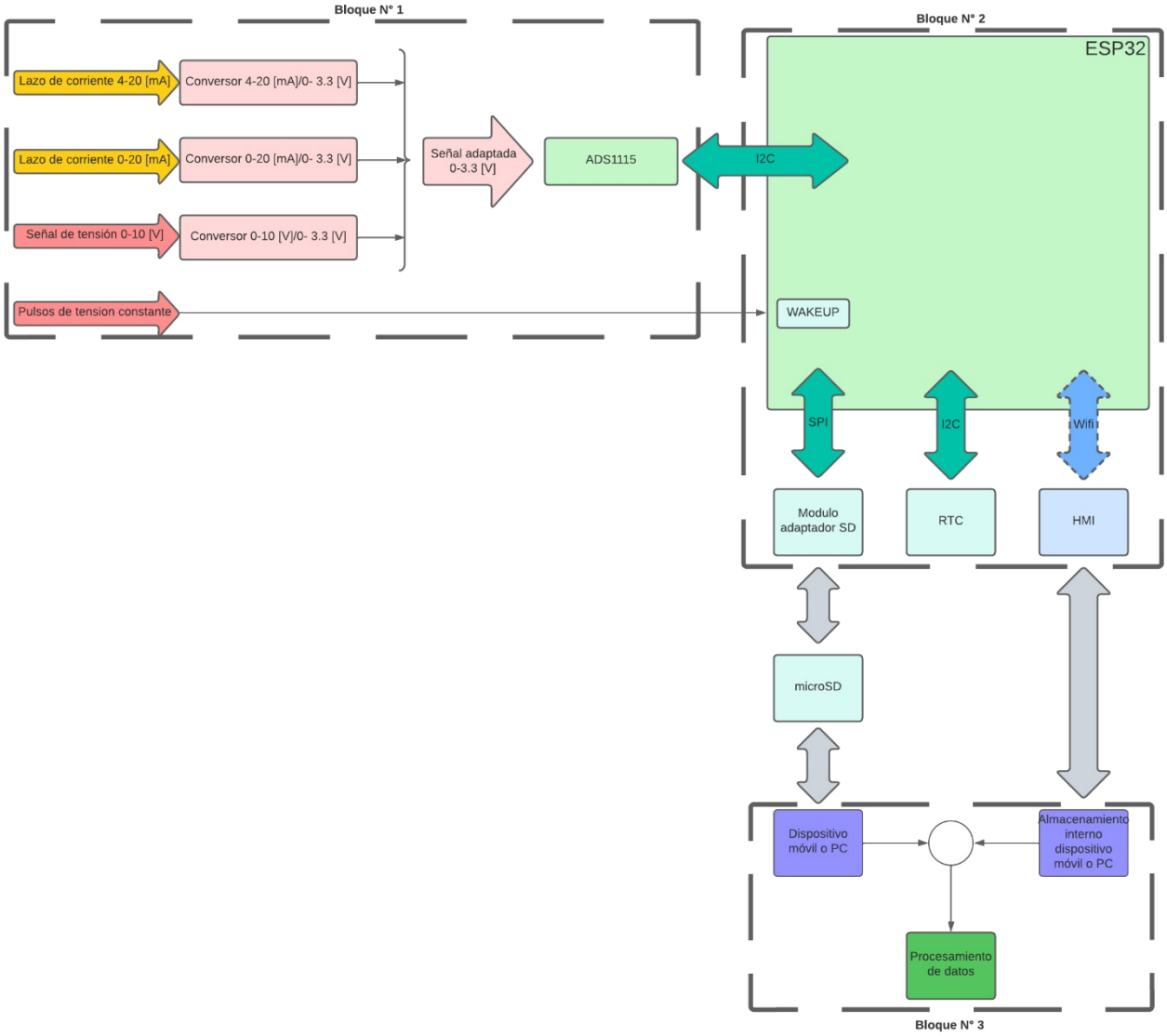

Figura 2 – "Diagrama de bloques"

El proyecto realizado se divide en tres grandes bloques, que a su vez están compuestos por bloques más pequeños, para facilitar su explicación. El bloque N°1 es el encargado de la adaptación de las señales provenientes de los sensores o controladores, a la tensión nominal con la cual trabaja el conversor analógico digital utilizado. El bloque N°2 es el encargado de recopilar y guardar los datos, y es el más extenso, ya que es en el cual se encuentra el microcontrolador elegido, junto con toda la lógica del programa, y a su vez la interfaz humano máquina (HMI), con la cual se puede configurar el dispositivo. Por último,

el bloque N°3, está compuesto por el tratamiento y procesamiento de los datos recopilados, para que el usuario pueda realizar un análisis con gráficas y tendencias. A continuación, se realizará un análisis exhaustivo de todos los bloques mencionados.

2.2 Bloque N°1: Obtención y adaptación de señales sensadas

2.2.1: Conversor de lazo de corriente a voltaje

A. Investigación de componentes

I. Análisis y selección

Como se mencionó previamente, se requiere de un adaptador de señales de lazo de corriente a la tensión con la que trabaja el ADC del módulo elegido (0 a 3.3 [V]). Se investigaron distintos circuitos, y por cuestiones de simplicidad y disponibilidad en el mercado, se optó por utilizar el módulo HW-685, el cual tiene una alta estabilidad, puede ser ajustado a lazos de corrientes de 0 – 20 [mA] y de 4 – 20 [mA], y su rango de tensión de salida convertida puede ser adaptable hasta 10 [V].

B. Propuestas de circuitos o esquemas.

I. Pruebas, experiencias.

Para poder verificar el correcto funcionamiento del conversor, se utilizó un Calibrador multifunción FLUKE 725 como fuente de corriente, junto con una fuente variable de tensión continua. El instrumento utilizado permite variar la corriente suministrada al orden de los microamperes, pero a fines prácticos se decidió incrementar la corriente de a un miliamperio, para poder visualizar la respuesta del conversor.

En primer lugar, se varió el potenciómetro de “cero” y se midió la salida hasta ajustar la misma al valor de 0 [V] para una corriente de entrada de 0 y 4 [mA] (dependiendo del lazo que se esté probando), y luego se ajustó el potenciómetro de valor máximo para una salida de 3,3 [V] a un valor de corriente de entrada de 20 [mA] (en ambos casos).

Luego, se comenzó a incrementar la corriente en el valor de 1 [mA], hasta llegar al máximo de escala de lazo, midiendo y anotando la tensión de salida, resultando las siguientes gráficas y tablas.

Lazo 4-20	
Iin [mA]	Vout [V]
4	0,001
5	0,207
6	0,413
7	0,619
8	0,826
9	1,032

10	1,238
11	1,445
12	1,651
13	1,857
14	2,064
15	2,27
16	2,476
17	2,682
18	2,889
19	3,095
20	3,302

Tabla 1 – "Iin vs Vout módulo HW-685 en lazo 4-20 [mA]"

Lazo 0-20	
Iin [mA]	Vout [V]
0	0,001
1	0,163
2	0,328
3	0,493
4	0,659
5	0,823
6	0,989
7	1,154
8	1,319
9	1,484
10	1,649
11	1,814
12	1,979
13	2,145
14	2,31
15	2,475
16	2,64
17	2,805
18	2,97
19	3,135
20	3,3

Tabla 2 – "Iin vs Vout módulo HW-685 en lazo 0-20 [mA]"

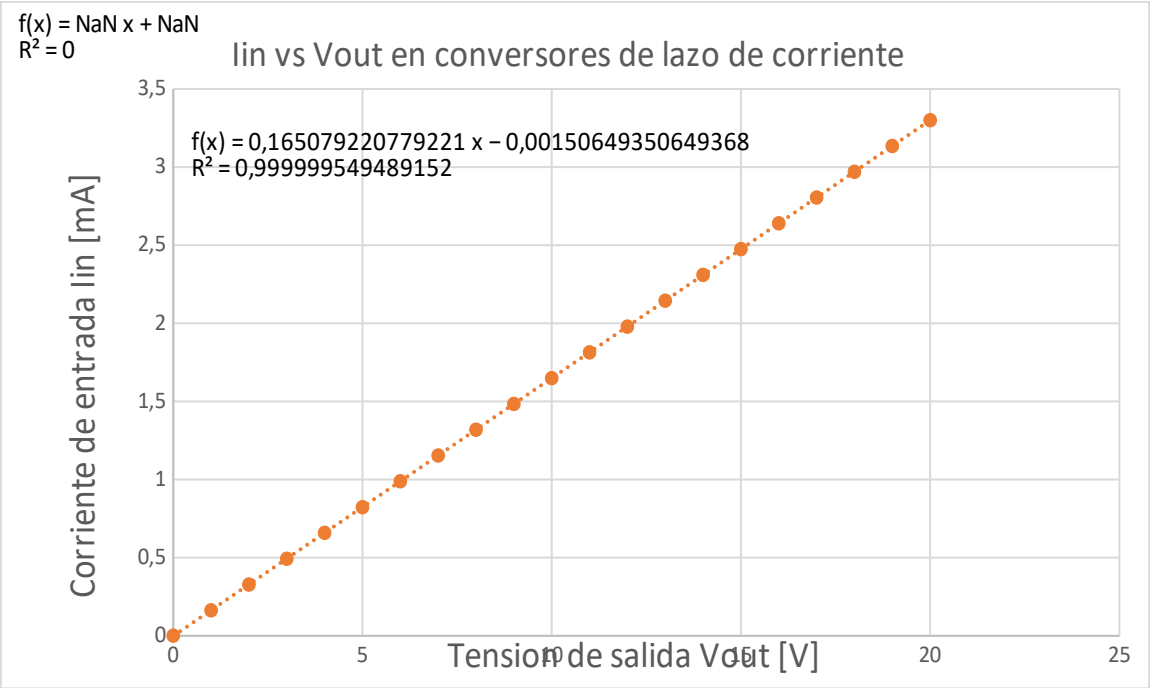

Figura 3 – "Gráfico Iin vs Vout de módulo HW-685 en lazo de corriente"

Como podemos ver en la gráfica anterior, visualmente la respuesta del conversor es lineal, y aplicando las fórmulas de Excel vemos también que el R^2 es igual a 1, por lo que se puede considerar matemáticamente lineal.

II. Problemas y soluciones implementadas

El primer problema encontrado fue que en estado de reposo (sin entrada de corriente), el circuito consume 120 [mA]. En este proyecto, se planteó a futuro reducir el consumo de corriente, por lo cual, como solución a este problema, se planteó un circuito para habilitar y deshabilitar el módulo mediante un pin digital, para que solamente esté activo en el momento de sensado del dispositivo.

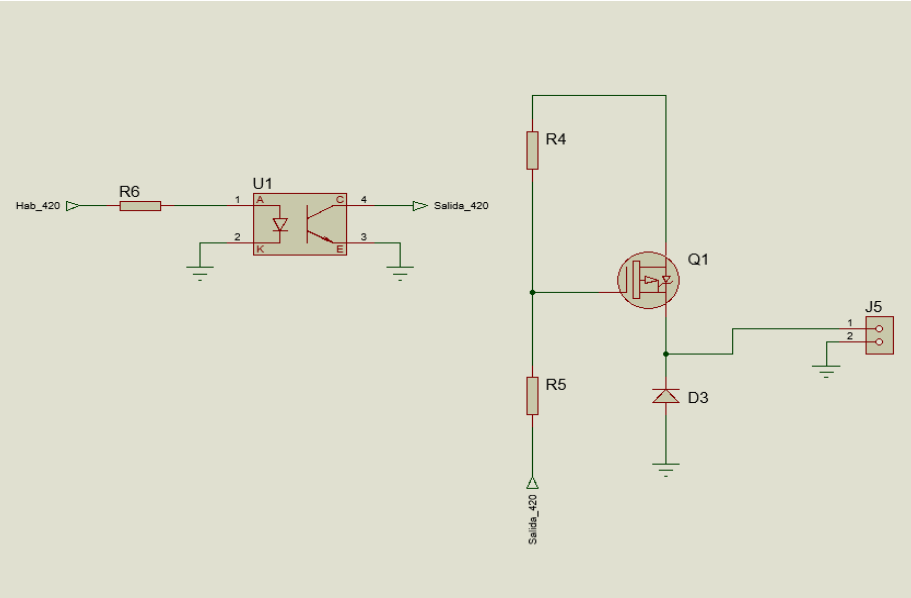

Figura 4 – "Circuito energizador de módulo HW-685"

En la lógica de inicialización y apagado del dispositivo, si el usuario configura un sensor de lazo de 4-20 [mA], se activa y desactiva esa entrada digital respectivamente, modo contrario siempre se mantiene apagada. En el diseño final de este prototipo no se incluyó este circuito.

C. Análisis del funcionamiento

Si bien no se pudo adquirir el circuito esquemático del módulo conversor, se explicará el principio de funcionamiento que se utiliza en el mismo, y se nombraran los aspectos de diseño que el fabricante agrega.

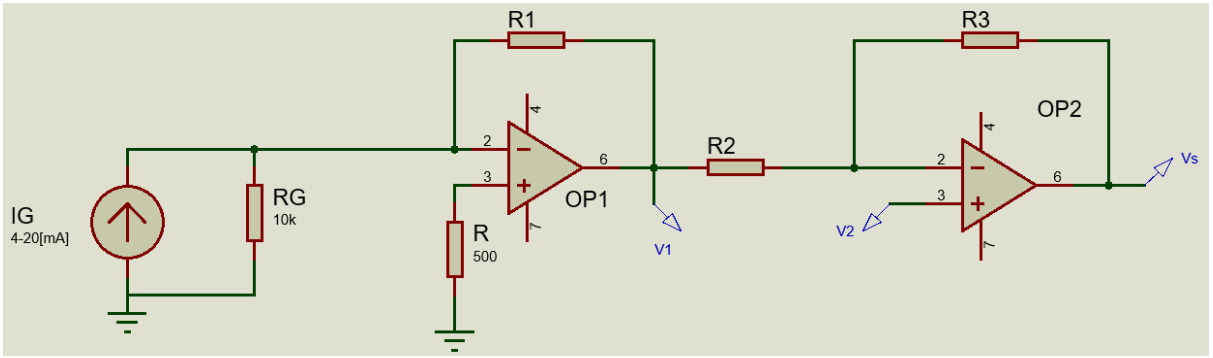

Figura 5 – "Circuito esquemático del módulo HW-685"

En la figura anterior se representa el sensor o transmisor con una fuente de corriente que varía en el rango 4-20 [mA], el cual proporciona una corriente proporcional a la magnitud física que está midiendo.

Se supone que se cumplen las condiciones de un amplificador operacional ideal, sabiendo que la ganancia tiende a infinito y la tensión entre las entradas es despreciable, se puede concluir que:

$$V2op1 = V3op1 = masa\ virtual$$

Sabiendo que la tensión en V2op1 está referenciada a una masa virtual, podemos concluir que la diferencia de potencial en la resistencia que modela el sensor (Rg) es cero, por lo tanto, no circulará corriente y:

$$I2op1 = Ig$$

La tensión de la resistencia V1 es:

$$V1 = -Ig * R1$$

Esta tensión es negativa ya que la corriente ingresa por el pin inversor del amplificador operacional. Se supone que la tensión Vs de salida tiene que variar entre 0 y 3.3 [V], por lo tanto, V1max tiene que valer 3.3 [V], cuando la entrada de corriente es de 20 [mA]:

$$R1 = \frac{V1max}{20[mA]} = \frac{3.3[V]}{20[mA]} = 165[ohms]$$

$$V1min = 165[ohms] * (-4[mA]) = -0.66[V]$$

$$V1max = -3.3[V]$$

Ahora se necesita adaptar el rango de tensiones de salida del primer operacional a la tensión requerida por el microcontrolador. Para esto se necesita un amplificador operacional en una configuración que adapte el rango de tensiones, como se ve en el circuito.

Se comienza planteando las condiciones de entrada y salida del circuito:

$$V1 = -0.66[V] \rightarrow Vs = 0[V]$$

$$V1 = -3.3[V] \rightarrow Vs = 3.3[V]$$

Sabiendo esto se haya la ecuación de la recta que tiene que respetar el OP2, para respetar el rango de tensiones de salida, resultando:

$$Vs = -0.825 - 1.25 * V1$$

Aplicando principio de superposición, anulando la entrada V2, queda el equivalente a un operacional en modo inversor:

$$Vs = \frac{-R3}{R2} * V1$$

Luego anulando V1, queda el equivalente a un operacional en modo no inversor:

$$Vs = \frac{R2+R3}{R2} * V2$$

Sumando ambas entradas resulta:

$$Vs = \frac{-R3}{R2} * V1 + \frac{R2+R3}{R2} * V2$$

Se analiza la ecuación de la recta con la obtenida previamente y se llega a las siguientes conclusiones:

$$1.25 = \frac{R2}{R3}; si\ R2 = 10[Kohms] \rightarrow R3 = 12.5[Kohms]$$

$$\frac{R2+R3}{R2} * V2 = -0.825 \rightarrow V2 = -0.366[V]$$

Utilizando esos valores de resistencias, se obtiene un circuito conversor de lazo de corriente 4-20 [mA] a un rango de tensiones de 0-3.3 [V].

Para poder usar este mismo circuito con una entrada de 0-20 [mA], se deben utilizar resistencias variables en lugar de los valores fijos que se calcularon, para poder variar el cero de entrada del mismo modo si se quiere utilizar una salida de 0 a 5 o 10 [V]. A continuación, se muestra una imagen del módulo utilizado, resaltando entradas, salidas y las resistencias variables antes mencionadas.

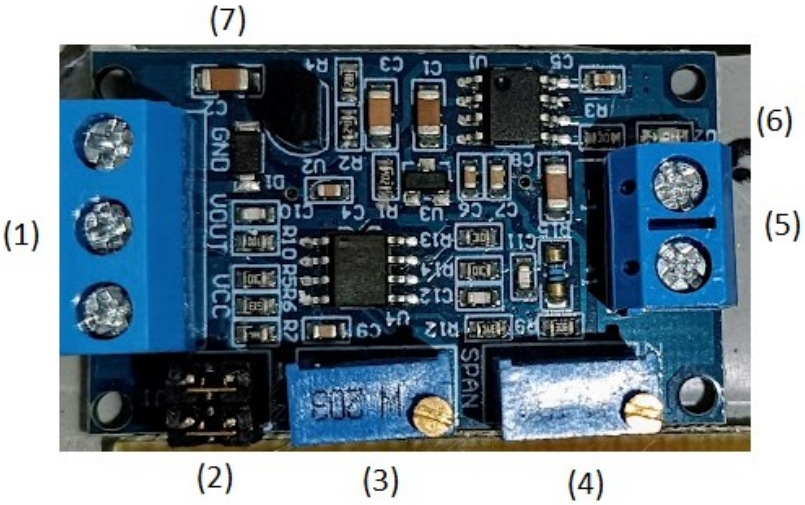

Figura 6 – "Módulo HW-685"

Referencias:

1. Bornera de alimentación y tensión de salida convertida. Los bornes superior e inferior corresponden a masa y tensión de entrada del módulo, y el del medio corresponde a la tensión de salida convertida.
2. Jumpers para cambiar el rango de tensión de salida. Dependiendo la combinación de estos jumpers y el lazo de corriente de entrada, se puede lograr una salida que varie en el rango de 0 a 10 [V].
3. Potenciómetro para ajuste fino del valor máximo de salida.
4. Potenciómetro para ajuste fino del valor mínimo de salida.
5. Bornera de ingreso de señal de lazo de corriente
6. Led indicador de tensión de entrada
7. Encuadrado en rojo muestra el diodo que se utiliza para proteger al circuito de tensiones inversas.

2.2.2: Adaptador de rango de tensión 0 - 10 [V] / 0 - 3.3 [V]

A. Investigación de componentes

I. Análisis y selección

En un primer lugar se planteó utilizar un adaptador de rango dinámico, utilizando un módulo Arduino o similar, pero se descartó debido a que se quiso reducir los costos de producto final. Luego se investigaron circuitos seguidores de tensión, para poder aislar la tensión de entrada a el ADC del microcontrolador, pero se decidió utilizar diodos schottky en antiparalelo para evitar que la tensión de entrada sobrepase los 3.3 [V].

B. Propuestas de circuitos o esquemas.

I. Pruebas, experiencias.

Al haber elegido el esquema de un divisor resistivo, se optó por elegir una resistencia fija, y un potenciómetro multivuelta para tener más precisión a la hora de fijar la tensión de salida.

II. Problemas y soluciones implementadas

No se encontraron mayores problemas para la implantación de este circuito.

C. Análisis del funcionamiento

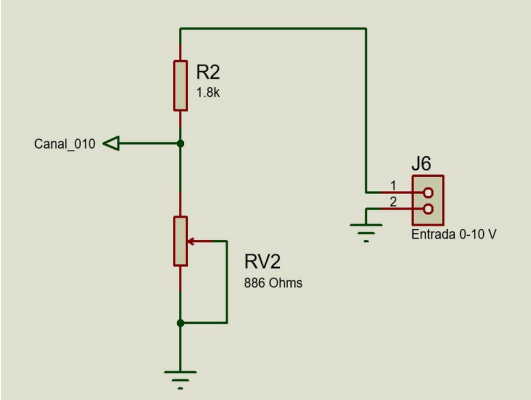

Figura 7 – "Adaptador de rango de tensión 0 - 10 [V] / 0 - 3.3 [V]"

El funcionamiento de un divisor resistivo es simple, la tensión de salida es la diferencia de potencial que se ve reflejada (en este caso), entre la resistencia RV2 y masa, y su valor numérico obtiene siguiendo la fórmula:

$$Vout = \frac{Vin * RV2}{RV2 + R2}$$

Como se mencionó previamente elegimos una resistencia fija, R2, y a partir de ella, calculamos el valor al que debemos fijar el potenciómetro para cumplir con el rango que se necesita en el presente trabajo, despejando RV2 de la fórmula anterior:

$$RV2 = \frac{Vout * R2}{V \in -V out} = 886 [ohms]$$

2.2.3: Pulsos de tensión constante

A. Investigación de componentes

I. Análisis y selección

La inclusión de este tipo de señal fue pensada en la primera prueba de campo del dispositivo, la cual luego será desarrollada profundamente más adelante en este informe, pero a modo de resumen, el sistema testeado era un controlador de flujo de agua de una central potabilizadora, y se necesitaba saber la cantidad de metros cúbicos de agua que circulaban en distintas franjas horarias del día. Para esto se utilizó una señal totalizadora proveniente del controlador, la cual era de tipo colector abierto (cero lógico o masa), pero dependiendo del tipo de controlador pueden ser señales de tensión constante (uno lógico). En este trabajo se planteó una señal de pulsos positivos, pero es fácilmente adaptable para pulsos de colector abierto, incluyendo dispositivos discretos o compuertas de negación lógica.

B. Propuestas de circuitos o esquemas.

I. Pruebas, experiencias.

En el primer prototipo, no se contempló la aislación eléctrica entre la señal proveniente del controlador y el pin del ESP32, debido a que, en la teoría, las señales de colector abierto, cuando no están activadas están en un estado de alta impedancia, y cuando se activan su estado es de cero lógico o masa.

Para el prototipo final, era necesario proteger el microcontrolador frente a algún eventual cortocircuito o una sobretensión de entrada, es por eso por lo que se utilizó un optoacoplador.

La corriente máxima que admite la entrada del optoacoplador es de 50 [mA], y al utilizar una resistencia de 150 [ohms], la tensión máxima de entrada permitida es de 7.5 [V] aproximadamente.

II. Problemas y soluciones implementadas

No se encontraron mayores inconvenientes a la hora de implementar este circuito.

C. Análisis del funcionamiento

El circuito implementado fue el siguiente:

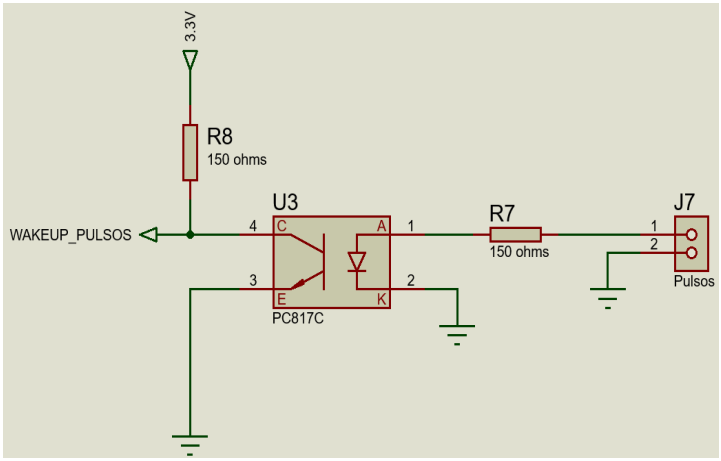

Figura 8 – "Circuito adaptador de pulsos de tensión constante"

La señal de entrada de pulsos de tensión constante provenientes del sensor, controlador, o dispositivo que se encuentre en el sistema donde se utilice el Datalogger Universal, ingresa por el ánodo del optoacoplador, excitando el fotodiodo, que trae como consecuencia la activación del fototransistor. La configuración que se encuentra a la salida es conocida como "pull-down", es decir, pone a masa la salida del transistor. Se eligió esta configuración, debido a que la señal de entrada al microcontrolador que detecta el pulso y hace que se despierte el mismo, está configurada para activarse mediante un cero lógico.

2.3 Bloque N°2: Almacenamiento de datos y configuración de dispositivo

2.3.1 Microcontrolador ESP32

A. Investigación de componentes

I. Análisis y selección

Para la elección del microcontrolador se realizó un análisis de las prestaciones que se requerían en el proyecto a realizar. Como ya se ha mencionado, la funcionalidad del dispositivo es recopilar datos en tiempo real, guardarlos con fecha y hora en una tarjeta microSD para luego procesarlos y graficarlos. La configuración del dispositivo se requería que fuese inalámbrica, ya que se plantea su uso en lugares remotos o de difícil acceso. La mayoría de los microcontroladores que se investigaron (Microchip, Atmel, STM32), poseen todas las características necesarias para la realización del dispositivo, a excepción de que ninguno tiene la comunicación Wifi o Bluetooth integrada. Si bien la mayoría cumple con los requerimientos mínimos para el desarrollo (conversor analógico digital, comunicación SPI, I2C, diversos pines digitales para su uso libre), el único que

posee todas las características necesarias es el ESP32, un microcontrolador desarrollado por Espressif.

Entre las múltiples ventajas que posee este microcontrolador, se nombran:

- Bajo costo. Su placa de desarrollo es fácil de conseguir y su precio es ligeramente más bajo que la competencia, teniendo en cuenta que los demás microcontroladores necesitan un módulo extra para su comunicación por Wifi o Bluetooth.
- Bajo consumo. En el modo Deep sleep, el consumo del microcontrolador puede ser reducido al orden de los microamperios.
- Además de incorporar la comunicación por Wifi y Bluetooth, posee también la capacidad de comunicarse mediante los protocolos SPI, I2C, CAN, Ethernet, y posee pines para ser utilizados como sensores capacitivos, entradas y salidas PWM, sensores de efecto Hall, etc.
- Puede ser programado tanto con las librerías de Arduino, como también MicroPython, lo cual brinda versatilidad a la hora de elegir el lenguaje de programación.

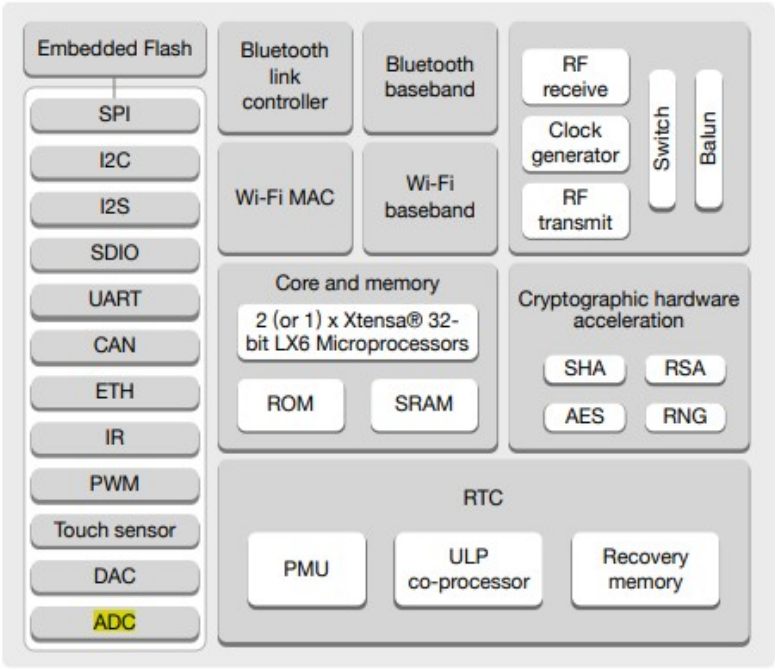

Figura 9 – "Diagrama de bloques del microcontrolador ESP32 ¨[1]"

B. Propuestas de circuitos o esquemas.

I. Pruebas, experiencias.

Se decidió utilizar un kit de desarrollo para la prueba y la implementación del proyecto. El diagrama de bloques de la placa es el siguiente:

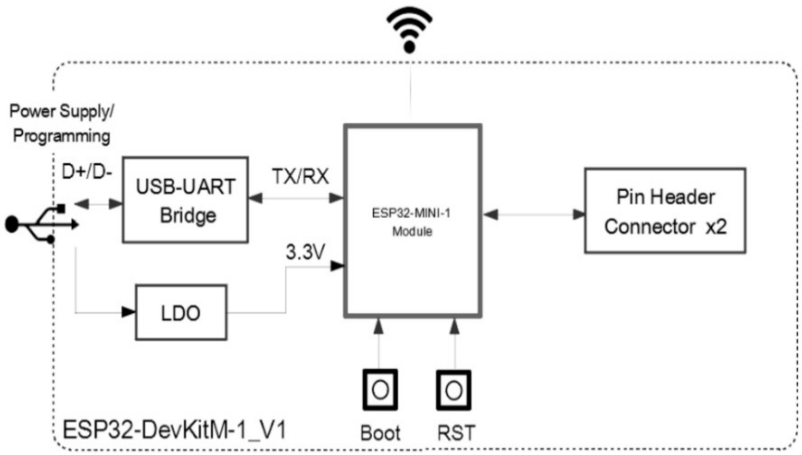

Figura 10 – "Diagrama de bloques de placa de desarrollo ESP32 [2]"

Como se muestra en la imagen, el kit está compuesto por el módulo ESP32 que contiene el microcontrolador, con la memoria interna de hasta 16 [MB] (dependiendo la versión) y un cristal de hasta 40 [MHz], una antena integrada, un puente USB-UART para la comunicación y grabación de programas, un regulador de tensión para abastecer al módulo, y dos hileras de pines macho que se conectan a los respectivos pines.

En este proyecto se empleó la versión del kit que posee 30 pines. Los utilizados para el funcionamiento serán descritos más adelante.
El regulador de tensión que se utiliza para suministrar la tensión de alimentación al ESP32 (3.3 [V]), es el AMS1117, que acorde con la datasheet, es un regulador de voltaje ajustable, diseñado para suministrar una corriente de 800 [mA] de salida, y operar con una diferencia de tensión entrada salida mínima de 1 [V].
El puente USB-UART, se implementa mediante el integrado CP2102, el cual es fabricado por Silicon Labs, que realiza la transferencia de datos de USB a UART para la grabación de programas y comunicación con la PC, sin la necesidad de excesivos componentes externos.

II. Problemas y soluciones implementadas

Durante el desarrollo del hardware y del software se presentaron diversos problemas. Se nombran los que se consideran de mayor importancia para el funcionamiento del dispositivo.

Investigando el ADC del ESP32 se encontraron diversas opiniones y ensayos, donde demostraban que el funcionamiento del ADC no era lineal, y que, además, en los rangos de tensiones de 0-0.1 [V] y 3.2-3.3 [V], el valor digital resultante no registraba cambios.

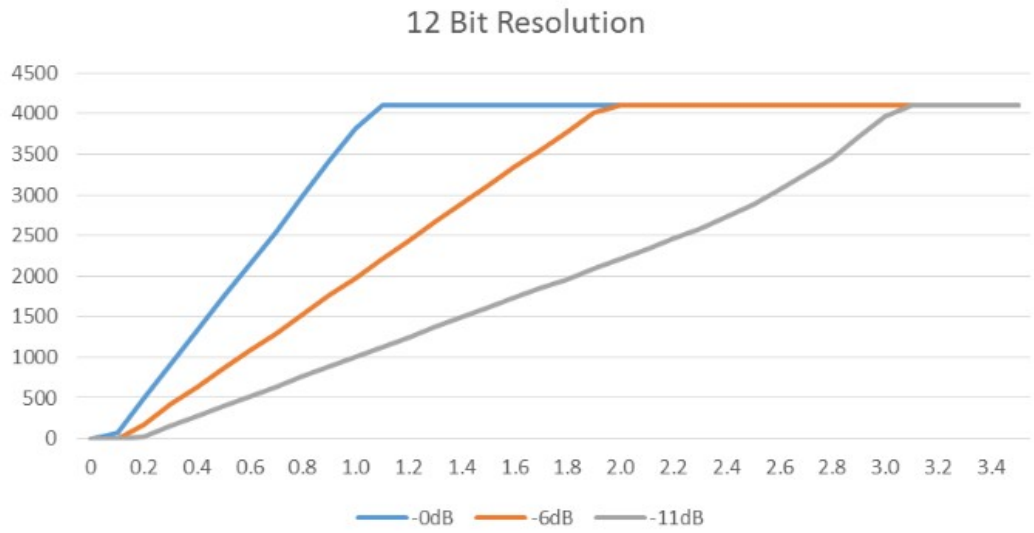

Figura 11 – "Test de linealidad del ADC [3]"

Como se ve en la anterior imagen, no existe linealidad en el conversor ADC del ESP32 en los rangos previamente nombrados. La imagen corresponde a un trabajo publicado, en el cual se concluyó que solo si se atenúa el canal del ADC, se podía obtener cierta linealidad, pero sin poder resolver el problema en los extremos de la recta.

Como primera solución a este problema, se pensó implementar una regresión lineal, para obtener un fórmula de corrección de los datos obtenidos. Cabe destacar que no todos los chips del ESP32 son iguales, por lo tanto, no todos los ADC serán iguales, y esta solución sólo serviría para el dispositivo desarrollado en el presente informe. Buscando una solución general para el problema mencionado anteriormente, se encontró un chip conocido como ADS1115, el cual es, como se describe en su datasheet, un conversor analógico digital de precisión, de 16 bit de resolución, y se comunica mediante el protocolo I2C. El mismo tiene 4 canales de entrada, y tiene integrada un PGA que permite lecturas de hasta ±0.265 [mV] (1 bit equivale a 0.0078125 [mV]). Este ADC no tiene un alto costo en el mercado, y su programación no posee demasiadas dificultades gracias a las librerías de Arduino, por lo que su implementación y adaptación al dispositivo fue sencilla.

Relacionado también con el ADC, se encontró información la cual decía que, si se implementan aplicaciones las cuales requerían el uso del ADC2 y del módulo Wifi, se generan conflictos internos en el ESP32, el cual resulta con una lectura errónea de los datos. Este problema también fue subsanado al utilizar un ADC externo.

Por último, también se experimentaron problemas utilizando las interrupciones del ESP32. Dependiendo del pin utilizado y cómo se activaban las mismas, se podía dar a lugar a interrupciones “falsas”. Esto trajo muchos problemas a la hora de implementar el modo de “wake up” para activar el servidor web y “sleep” para forzar el apagado del microcontrolador. Se encontraron diversas soluciones en los foros consultados, pero debido a que se utilizó el framework de Arduino para la programación, se escogió la que planteaba utilizar solo interrupciones activadas por nivel alto, y en determinados pines del microcontrolador.

C. Análisis del funcionamiento

Las funciones del ESP32 son, además de la comunicación con los periféricos implementados (RTC, ADS1115 y microSD), guardar la configuración de usuario mediante la utilización del servidor web y dependiendo de esta configuración, activarse y desactivarse para poder guardar los datos.

Existen tres modos de activación en el ESP32, por temporizador, por sensores táctiles, y por la utilización de señales externas al microcontrolador. Se utilizará el tercer modo mencionado, el cual tiene dos métodos, llamados ext0 y ext1. La diferencia entre ambos es que en “ext0” el micro se “despierta” por un único pin, y en “ext1” se “despierta” utilizando múltiples pines. Por una cuestión de diseño, se utilizaron ambos métodos.

Para el método ext0, se configuraron los pines que despertarán al microcontrolador para que recopile los datos del ADC (sea por un periodo de tiempo o solo por un pulso totalizador). Dentro de la lógica del software, está contemplado el hecho que solo uno de los dos puede estar configurado como fuente del método ext0. Estos se activan por un pulso bajo o un “0” lógico, por lo que se configuraron con la resistencia interna de “pull-up” que tiene el microcontrolador. Si el usuario configuró el dispositivo en modo de funcionamiento “1” (sensado en periodos de tiempo y una frecuencia de muestreo elegidas), el ESP32 se despierta a través de una señal proveniente del RTC, y si fue configurado en modo de funcionamiento “2”, esta señal es generada por una fuente externa.

El método ext1 puede ser utilizado de dos formas, para este trabajo se utilizó la que el microcontrolador se despierta si a cualquiera de los pines seleccionados se le asigna una señal de “1” lógico. Para discriminar el pin que fue activado, se utiliza una máscara de bits en hexadecimal, que luego, cuando el ESP32 se “despierta”, llama a una función que devuelve el pin que fue activado de la forma 2^{PIN}. Aplicando logaritmo, se puede obtener el número del pin, y luego decidir qué fragmento del código ejecutar. El dispositivo solo utiliza este método para “despertarse” y acceder mediante un servidor web, al HMI diseñado para su configuración. Para que el usuario pueda acceder a esta función cuando necesite, se colocó un pulsador conectado a un optoacoplador para proteger al ESP32. Una vez pulsado, genera el “1” lógico necesario para que el microcontrolador se

“despierte”. Si en un futuro se quieren añadir más pines, solo se debe modificar la máscara y agregar más pulsadores.

Durante la realización e implementación del dispositivo, se encontraron diversos problemas que podía tener el usuario a la hora de utilizarlo. Una de ellas fue que, en el caso de que el dispositivo fuese configurado en modo “1” se necesite modificar ciertos parámetros de configuración o agregar nuevos sensores, y el tiempo de sensado sea muy extenso para esperar que finalice. Es por eso por lo que se agregó un pulsador vinculado a un pin que genera una interrupción, la cual ejecuta una línea de código que hace que el ESP32 pase a modo desactivado o “sleep”. Una vez desactivado, se puede volver a despertar mediante el pulsador que activa el servidor web, previamente mencionado.

D. Desarrollo del software o firmware

El programa está dividido en dos grandes funciones, la función setup y la función loop, como en la mayoría de los programas diseñados en Arduino. La función setup se ejecuta una única vez, y la función loop, como lo indica su nombre, es un bucle infinito. Como se verá a continuación, la mayor parte del programa está escrita en la parte de setup, y en el loop, solo se encuentran las variables vinculadas a las interrupciones por timer, que se utilizan para la frecuencia de muestreo y fin de tiempo de sensado.
Para poder analizar el software, se dividió el diagrama de flujo en varios bloques y se mencionaron las partes que se consideran más importantes para el funcionamiento del dispositivo.

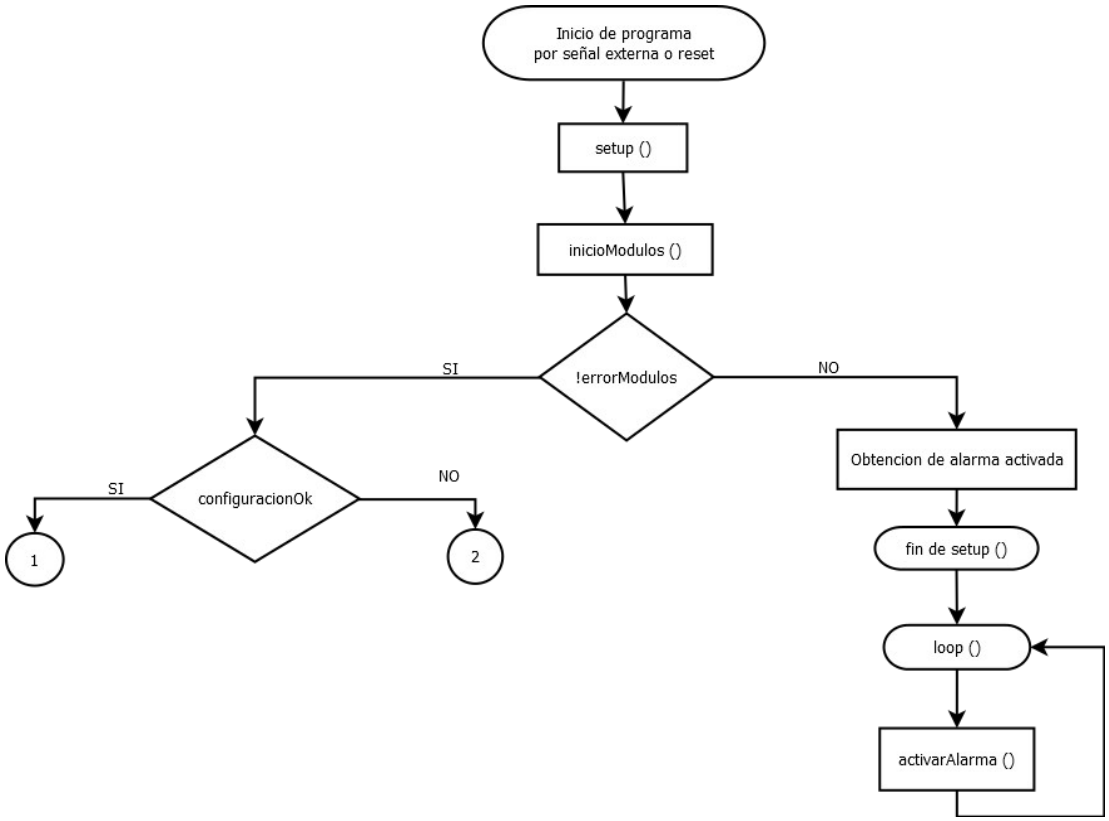

Figura 12 – "Diagrama de flujo Datalogger universal. Parte 1"

El microcontrolador comienza a ejecutar el programa cuando recibe una señal externa para activarse. Una vez recibida esta señal, se comienza a ejecutar la función setup(), donde primero se configuran los pines que se utilizan en el microcontrolador, así como las variables globales, y luego se ejecuta la función que inicia todos los periféricos utilizados. Cualquiera de los tres que se utilizan son fundamentales para el correcto funcionamiento del dispositivo, es por eso por lo que se incluyeron testigos visuales y sonoros, para avisarle al usuario que algo está fuera de lugar. Si no hay error se chequea la variable que guarda la información si el dispositivo está configurado para su uso o no.

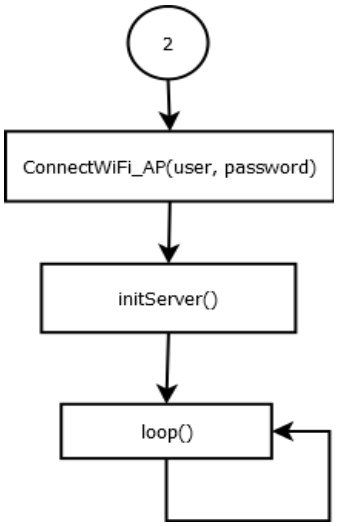

Figura 13 – "Diagrama de flujo Datalogger universal. Parte 2"

Si el dispositivo no está configurado, el programa ejecuta la parte de código que inicia el servidor web, en conjunto con la conexión a través de Wifi del dispositivo. En este caso, el ESP32 está configurado como punto de acceso, es decir, crea su propia red Wifi, para que los usuarios puedan conectarse. Esto quiere decir que, en principio no tiene acceso a internet. Para conectarse a esta red, hay que ingresar un usuario y contraseña predeterminada (luego podrá ser cambiada), y una vez ingresada a la red, se debe abrir un navegador, colocar la IP predeterminada del servidor web implementado, y comenzar a configurar el dispositivo.

Si el dispositivo fue configurado se ejecuta la función extraerConfig() y extrae toda la configuración necesaria para el funcionamiento del dispositivo, comenzando con el modo de operación. Si el modo de operación es 1, el microcontrolador se despierta según los horarios configurados por el usuario, mediante una señal de alarma del módulo RTC, y comienza a sensar por un determinado tiempo y una determinada frecuencia de muestreo, también configurados por el mismo. En cambio, si el modo es 2, el dispositivo se despierta cuando recibe un pulso en el pin del canal configurado para esta función, para guardar la fecha y hora en que ocurrió el evento, así como también si se desea, una variable analógica. Además de extraer el modo, se extraen la cantidad de sensores configurados, junto con los canales que se asignaron a cada uno.

Evaluando las distintas posibilidades de que ocurra algún error externo al diseño del dispositivo, se planteó la posibilidad de que el microcontrolador se reinicie por algún cambio en la tensión de entrada. Para esto se crea la variable "reset", la cual se define como una variable "RTC_DATA_ATTR". Este tipo de variables se guardan en la memoria del RTC interno del microcontrolador, manteniendo su valor si se modifica, incluso cuando el mismo se "duerme", pero volviendo a su valor de definición cuando se reinicia.

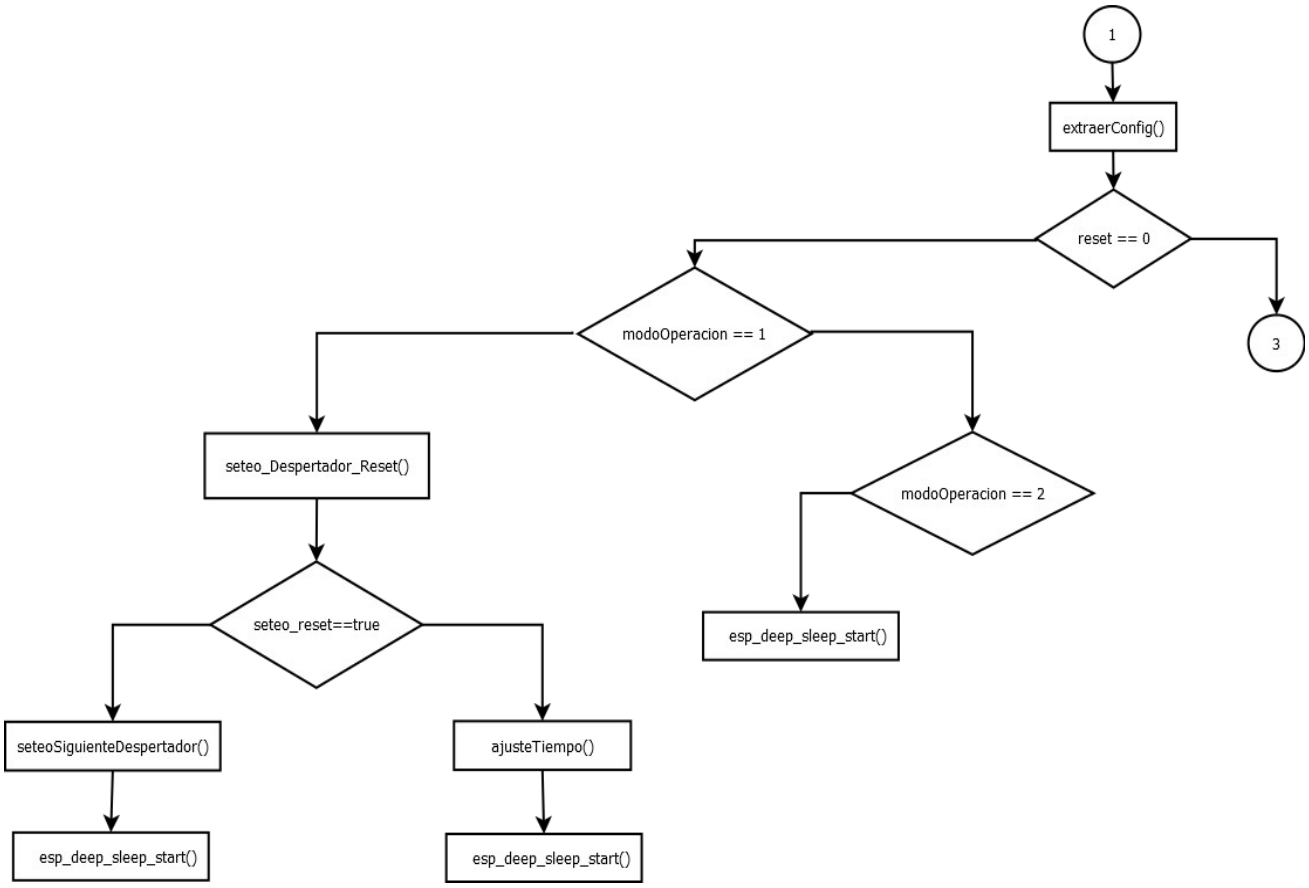

Figura 14 – "Diagrama de flujo Datalogger universal. Parte 3"

Si el microcontrolador se reinició y el modo de operación es 2, se modifica la variable reset y el microcontrolador vuelve a su estado de reposo y como se explicó previamente.

Si el modo de operación es 1, se debe verificar si en el momento que ocurre el reinicio, se encuentra dentro de un horario de sensado configurado por el usuario. Para esto, se ejecuta la función "seteo_Despertador_reset()", la cual devuelve un valor booleano. Si resultó verdadero, quiere decir que estamos fuera de un horario de sensado configurado, por lo que se setea el próximo despertador y se "duerme" el ESP32. Si resulto falso, quiere decir que el reinicio ocurrió mientras el dispositivo estaba recopilando datos, por lo tanto, se ejecutan una serie de funciones, que calculan el tiempo restante de sensado, y permiten que el ESP32 siga con su funcionamiento normal, hasta finalizar el tiempo de sensado, y se configura el próximo despertador, durmiendo el microcontrolador.

Si el dispositivo no se reinició, y tiene una configuración guardada, entonces el programa debe evaluar cuál fue la señal que hizo que se active. Para discriminar cuál fue el motivo, se realiza la instrucción "switch", y dependiendo del resultado, el programa puede realizar distintas acciones.

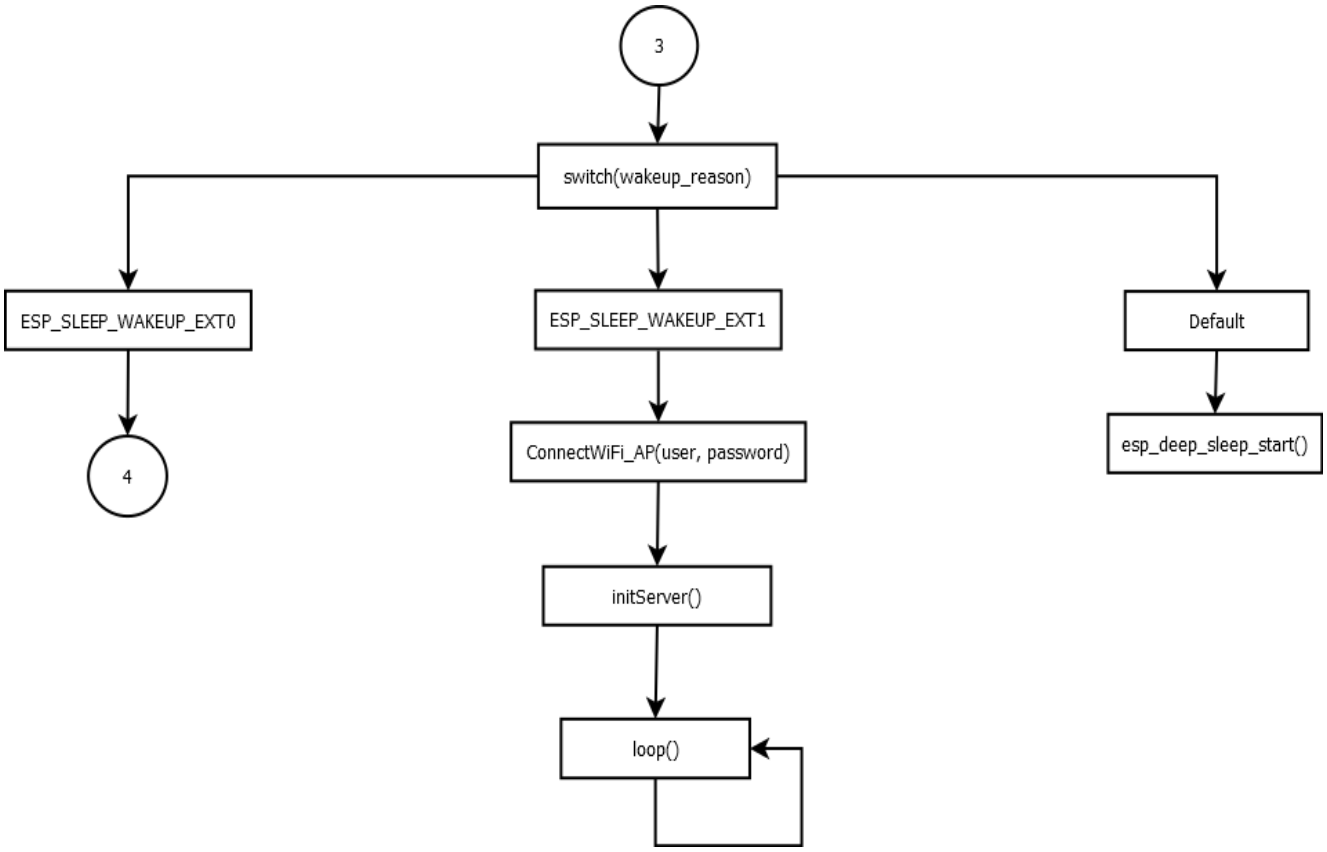

Figura 15 – "Diagrama de flujo Datalogger universal. Parte 4"

Si el usuario presiona el botón para activar el servidor web previamente mencionado, se ejecuta la opción ESP_SLEEP_WAKEUP_EXT1. Una vez finalizada la configuración, el dispositivo vuelve a dormirse hasta recibir una señal externa. Si se ejecuta la opción default, se considera que se activó por error y se apaga. En el caso que se ejecute la opción ESP_SLEEP_WAKEUP_EXT0, significa que la señal que activó el dispositivo fue para comenzar a recopilar datos.

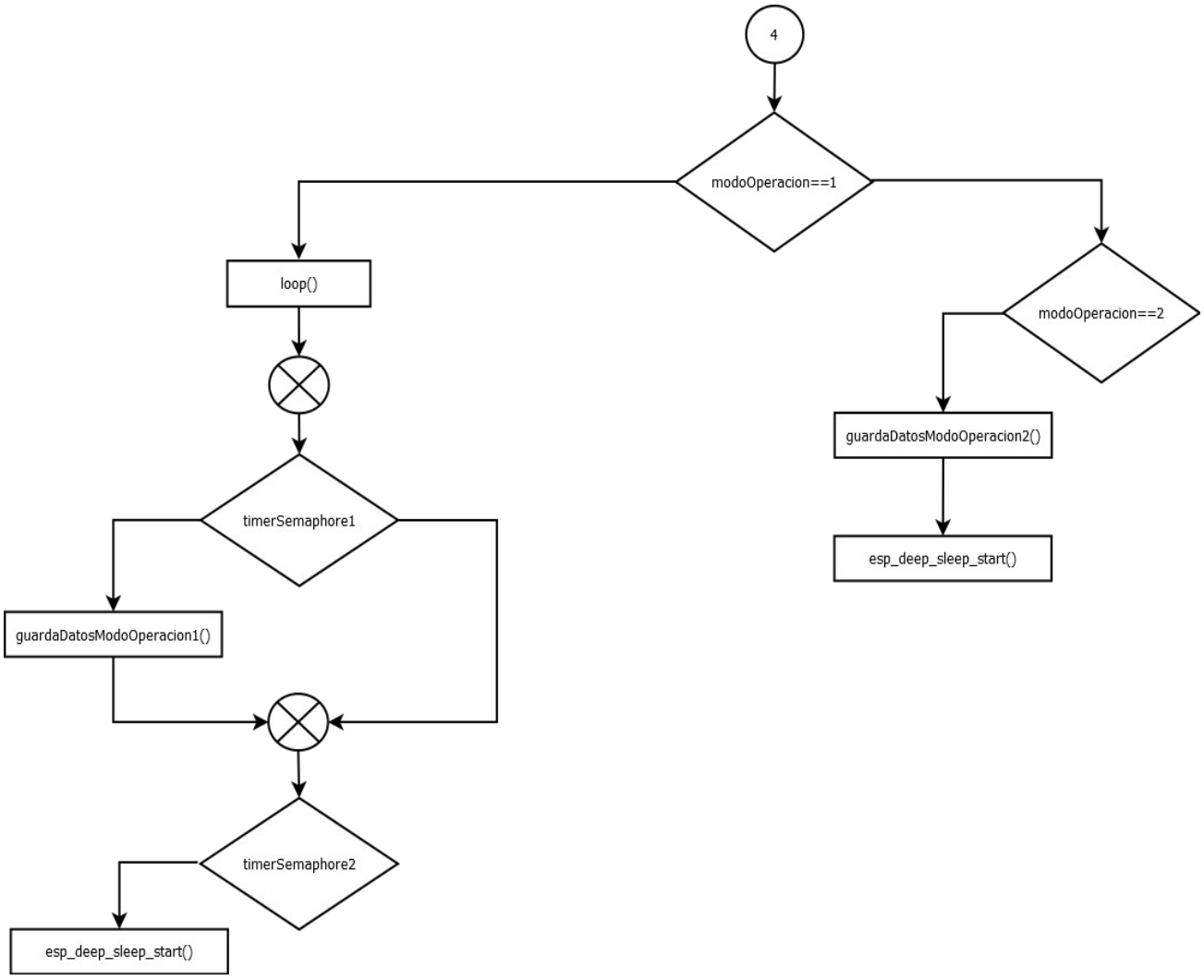

Figura 16 – "Diagrama de flujo Datalogger universal. Parte 5"

Se realizan las comparaciones necesarias para identificar qué modo de operación fue configurado por el usuario, y partir de eso se guardan los datos del mismo modo que se explicó en la parte anterior, con la única diferencia que el tiempo de sensado no se tiene que ajustar ya que no hubo un reinicio.

2.3.4 HMI. Diseño e implementación.

A. Investigación de componentes

I. Análisis y selección

En un principio, la comunicación entre el usuario y el dispositivo diseñado en este informe se planteó que se iba a realizar en dos partes, la primera, por bluetooth, para acceder a la

configuración del microcontrolador, y la segunda por wifi, para acceder a los datos recopilados. Finalmente, la opción de comunicarse por bluetooth fue descartada, ya que además de que se requería más tiempo para interiorizarse en los lenguajes de programación requeridos, el consumo de energía eléctrica iba a ser mayor y el diseño final del dispositivo se iba a complejizar innecesariamente.
Como solución se planteó utilizar solo la comunicación por wifi, programando y diseñando un servidor web que aloje el HMI. Las ventajas de utilizar un servidor web se mencionan en los siguientes ítems:

- Puede funcionar para cualquier sistema operativo que tenga un explorador de internet (Android, IOS, Linux, Windows, etc)
- Menor tiempo de desarrollo que una aplicación solo para Android, debido a que los lenguajes de programación HTML y JavaScript similares a los aprendidos en la carrera.
- La comunicación por wifi soporta mayor distancia entre dispositivos que por bluetooth.
- No se necesita una conexión a internet, ya que el microcontrolador puede emitir la señal y comunicarse con cualquier dispositivo.

B. Análisis del funcionamiento

En primera instancia, el usuario debe poseer un dispositivo que sea compatible para conectarse por wifi (Smartphone, Notebook, Netbook, etc.). Luego se debe activar la señal de wifi del Datalogger, el cual como ya se ha mencionado, tiene un pulsador para esta función.

El ESP32, se puede configurar de dos modos, el modo “Station” y el modo “Access Point”. En el modo “Station”, se puede conectar a un access point o punto de acceso, como puede ser un router, y si ese router tiene acceso a internet, el ESP32 lo tendrá también. En ese caso el ESP32 se comporta como un “cliente”, puede hacer peticiones y controlar otros dispositivos conectados a la red, y viceversa. En el modo “Access point” el ESP32 se comporta como un router, y permite que distintos dispositivos se conecten a él. En este caso, no existe conexión a internet. Para el dispositivo diseñado en este trabajo final, se eligió el modo “Access point”, debido a que el mismo está pensado para ser utilizado en zonas de difícil acceso, donde no haya conexión a internet. Entonces, el usuario se conecta mediante su dispositivo al ESP32, ingresando a la parte de configuración de wifi y buscando el nombre del dispositivo en las redes disponibles. Por cuestiones de seguridad la red es privada y solo se puede acceder ingresando una contraseña.

Por defecto, el dispositivo viene configurado con el nombre de red Datalogger_Universal, y la contraseña AutoSolve. Desde el HMI, el usuario puede cambiar estos valores, para identificarlo como desee. Dentro del ESP32 se encuentra escrito el código que genera el servidor web donde se aloja el HMI para configurar los modos de funcionamiento del dispositivo. Para ingresar al servidor web debemos escribir en el navegador web que deseemos, la IP asignada al mismo. En este caso es la siguiente: 192.168.1.17.

El servidor web generado, es de tipo asíncrono, es decir, que permite y puede manejar más de una conexión a la vez. Se eligió este tipo de servidor no solo por esta función (si bien se ensayó y funciona correctamente, sin generar errores), sino porque se evita escribir código extra en el bucle “loop()”, haciendo más eficiente el programa. El usuario y el ESP32 se comunican mediante el protocolo “HTTP” (protocolo de transferencia de hipertexto), con el cual funcionan todas las páginas web que se alojan en internet. Este protocolo sigue el esquema de “petición-respuesta”, mediante los métodos GET y POST para las peticiones donde se aloja el servidor (en este caso el ESP32), y los códigos de respuesta para las respuestas al servidor (Ej.: 200 solicitud exitosa, 404 cuando no se halló lo que el contenido solicitado, etc.).

Una vez el usuario se conecta con el ESP32, y coloca la IP del servidor en el navegador web, se accede al HMI, para poder configurar el dispositivo. Hay que destacar que la HMI se adapta según el tamaño de pantalla que posea el usuario, como se ve en las siguientes imágenes.

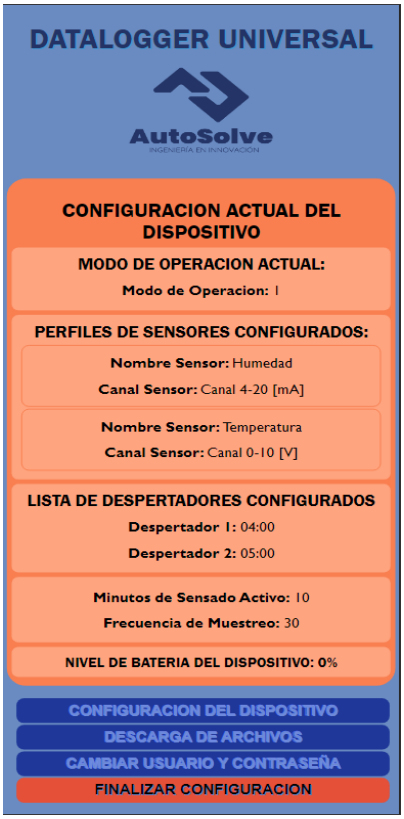

Figura 17 – “Diseño HMI Vista Samsung Galaxy A51/71”

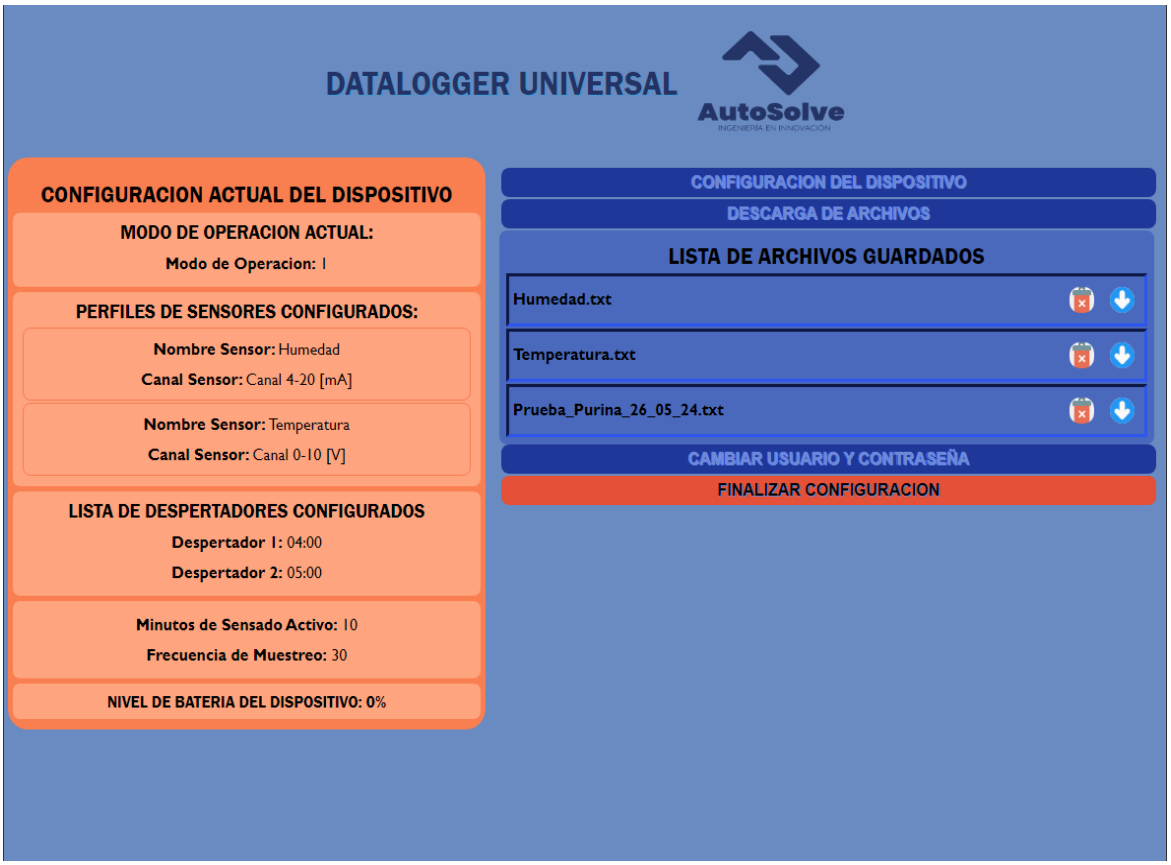

Figura 18 – "Diseño HMI Vista Ipad air"

La pantalla principal del HMI se divide en dos grandes bloques, el bloque de la izquierda (color naranja), el cual, como indica su nombre, es el encargado de mostrar la configuración actual del dispositivo. El bloque de la derecha (a su vez subdividido en tres pestañas) es el encargado de modificar la configuración del dispositivo, mostrar y permitir la descarga de los archivos de los datos sensados, y cambiar el nombre de usuario y contraseña, como se mencionó previamente.

Cada modificación que se realice en el HMI, genera un método de petición o "request" por parte del navegador web, gracias a la programación realizada mediante JavaScript. Existen varios métodos, pero como se explicó antes, en el presente trabajo se utilizaron dos, "GET" y "POST". La diferencia entre ambos es que cuando el navegador ejecuta el método GET, solo toma información del ESP32 o del HMI para mostrarla, y el método POST, además de poder tomar la información, la modifica dentro del ESP32 y el HMI.

Para elegir el modo de funcionamiento, se debe hacer clic en el selector y se despliegan las dos opciones que se encuentran en el dispositivo. Se colocó entre paréntesis la descripción de cada modo, evitando así posibles confusiones del usuario.

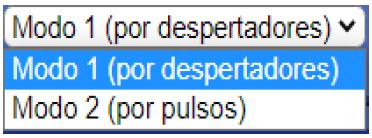

Figura 19 – "HMI: Selección de Modos de operación"

Por una cuestión de diseño, cada vez que en el HMI se cambia el modo de funcionamiento, se elimina cualquier configuración que haya sido guardada, y se refresca la pestaña, para mostrar los elementos correspondientes. Para advertir al usuario, se muestra una ventana de alerta, donde el mismo puede elegir la acción a realizar.

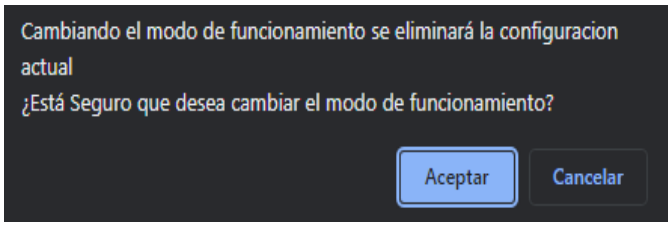

Figura 20 – "HMI: Aviso de cambio de modo"

En el modo de operación 1, se debe configurar como mínimo una entrada de señal de un sensor para guardar sus datos. Con el signo "+" se agrega un perfil de sensor, que contiene un área de texto para colocar el nombre del sensor, un selector para elegir el canal de los datos sensados, y dos botones, uno para eliminar el sensor guardado, y el otro para guardarlo.

Figura 21 – "HMI: designación de sensores y canales en modo 1"

Solo se pueden agregar hasta 3 perfiles de sensores, debido al diseño del dispositivo. En caso de que se presione el botón para agregar más de tres veces, se mostrará un mensaje de error.

Una vez que se coloca el nombre del sensor y se elige un canal, se debe presionar el botón de guardar, el cual está representado por la imagen de color verde con una tilde en el medio, y mediante un conjunto de funciones programadas en JavaScript, se verifican ciertos requisitos para que el sensor pueda ser guardado en la configuración del dispositivo. Primero se verifica si el nombre no contiene ningún carácter especial o algún espacio en su nombre, debido a las reglas de la estructura del sistema de archivos que se utiliza en la mayoría de los sistemas operativos. De haber algún error, el HMI muestra los siguientes mensajes, y no permite guardar el perfil del sensor.

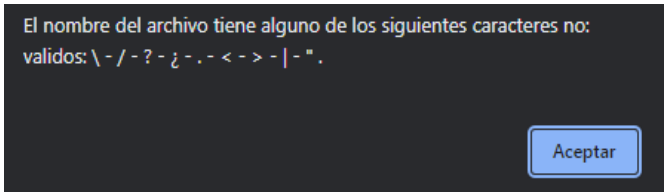

Figura 22 – "HMI: Aviso de caracteres no validos"

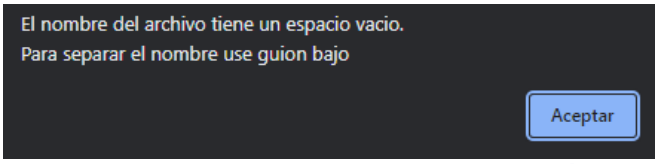

Figura 23 – "HMI: Aviso de nombre erróneo"

Si no existe ningún error, el HMI realiza un POST al ESP32 para verificar si existe algún sensor con el mismo nombre (dentro del ESP32, se ingresa al sistema de almacenamiento y se compara con los nombres de los archivos existentes). Si se encuentra alguna coincidencia, se da aviso al usuario para que decida si quiere continuar guardando los datos en el mismo archivo o quiere cambiar el nombre.

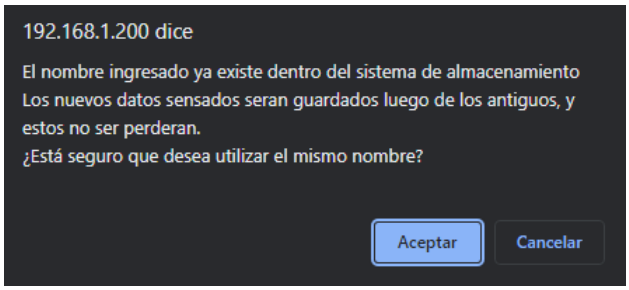

Figura 24 – "HMI: "Aviso de nombre de perfil de sensor existente"

De dar una respuesta afirmativa, ahora se debe comparar el nombre y el canal del sensor configurado, con los perfiles de los sensores que se encuentran "activos" o ya configurados. Por una cuestión de lógica de programa no puede haber dos perfiles de

sensores activos con el mismo nombre y/o el mismo canal. Si existiese algún sensor previamente configurado con el mismo nombre o canal, el HMI le avisa al usuario del error. Si no existe ninguno de los errores mencionados anteriormente, el ESP32 guarda el perfil del sensor dentro de su configuración, y el HMI genera un mensaje confirmando que el perfil del sensor fue guardado, y actualiza la información correspondiente.

Luego de haber guardado los nombres y canales de los sensores, en el modo 1, se debe configurar el horario en el cual se quiere que el dispositivo recopile los datos (horario despertadores) y el tiempo que se requiere que el mismo esté activo (minutos de sensado activo).

Figura 25 – "HMI: Selección de minutos de sensado y despertadores en modo 1"

Para que un horario de despertador se guarde correctamente, debe haber valor válido de minutos de sensados activo (rango de 1 a 59 min), y haber seleccionado correctamente un horario dentro del elemento correspondiente. De modo contrario aparecerá un mensaje de error.

Otro error contemplado que puede ocurrir cuando se configuran los despertadores, es que el usuario elija un despertador que se encuentra dentro del tiempo del despertador anterior sumado a los minutos activos de sensado. En este caso aparece un mensaje diferente, donde se avisa al usuario del error previamente explicado.

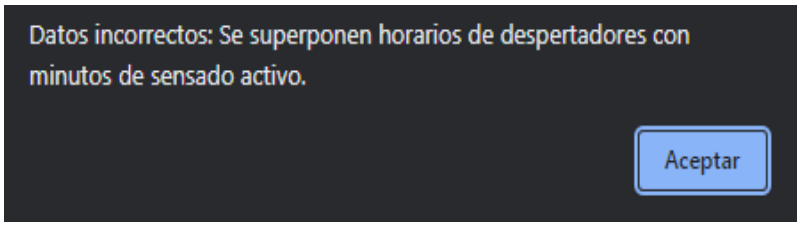

Figura 26 – "HMI: Aviso de error en horarios de despertadores"

Para simplificar y evitar que el usuario cometa errores con respecto a lo mencionado anteriormente, una vez que se guarda el primer despertador, no se pueden cambiar los minutos de sensado activos, a menos que se eliminen todos los despertadores creados, para comenzar nuevamente el proceso.

Finalizada la configuración de horarios de despertadores, se debe elegir la frecuencia de muestreo de datos. Debido a que el dispositivo se diseñó para procesos relativamente lentos, el rango de frecuencia de muestreo elegido es de 1 a 59 segundos. También se eligió este rango, teniendo en cuenta que el mínimo de minutos activo es de 1, por lo tanto, no sería coherente que hubiera una frecuencia de muestreo mayor a 60 segundos.

Figura 27 – "HMI: Selección de frecuencia de muestreo"

Para finalizar la configuración tanto en modo 1 como en modo 2, se debe presionar el botón que tiene como nombre "Finalizar configuración". Una vez presionado el botón, el HMI alerta al usuario que el dispositivo pasará a modo sensado, preguntándole si es lo que desea hacer. Si el usuario desea finalizar la configuración, el dispositivo evalúa si se configuraron todos los elementos necesarios para que funcione en modo 1. Si falta alguno de ellos se muestra al usuario el siguiente mensaje.

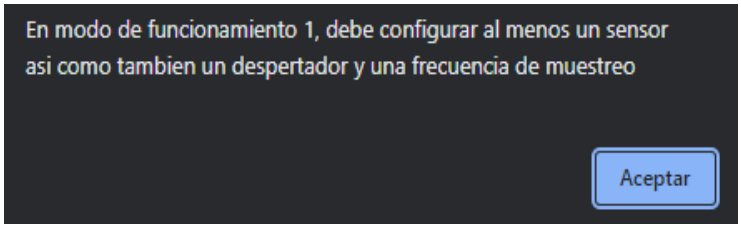

Figura 28 – "HMI: Aviso de falta de parámetros de configuración"

Finalizada la configuración, el HMI se bloquea, mostrándose de la siguiente manera, y se avisa al usuario, la manera de volver a acceder a la configuración del dispositivo.

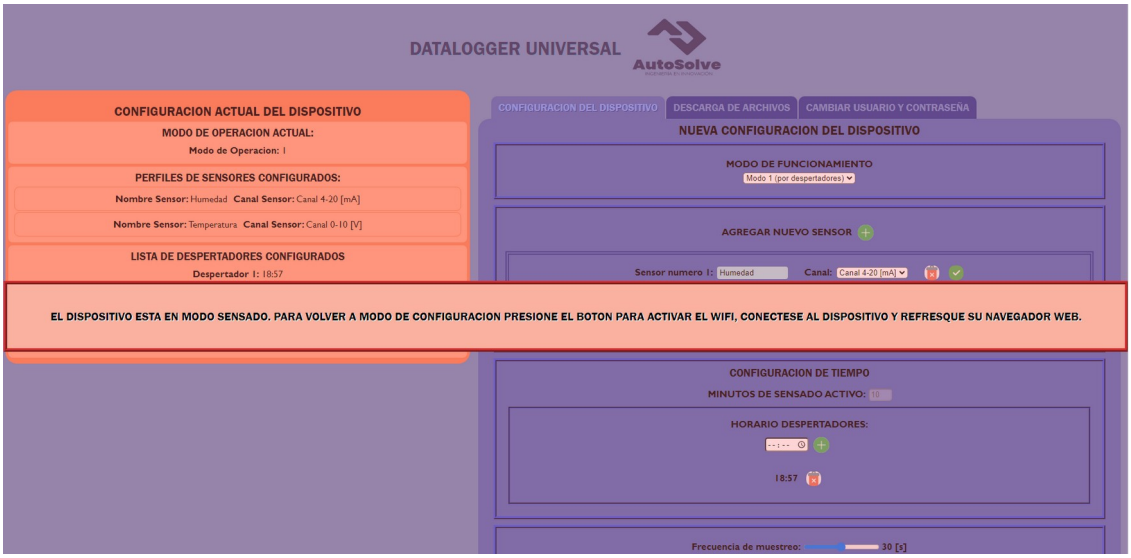

Figura 29 – "HMI: Pantalla de bloqueo luego de finalizar configuración"

Para el modo de funcionamiento 2, el HMI se muestra de la siguiente manera:

Figura 30 – "HMI: Selección de sensores en modo de operación 2"

A diferencia del modo de funcionamiento anterior, no se tienen que configurar despertadores ni frecuencia de muestreo, ya que el microcontrolador se despierta detectando un pulso en el canal correspondiente para esta función. Para que el horario y la fecha en el que ocurrió el evento sean guardados, se debe configurar un perfil de sensor por pulsos, colocando un nombre y presionando el botón de guardar. A su vez, se brinda la opción de configurar tres sensores extras, conectados a cada canal existente en el dispositivo, por si el usuario quiere registrar algún valor en el momento que se produjo el pulso.

De manera análoga con el modo de funcionamiento 1, cuando se presiona el botón de finalizar configuración, aparece un mensaje de error si el usuario no guardó un perfil de sensor por pulsos. De haber configurado uno, el HMI muestra el mismo mensaje, avisando al usuario que el dispositivo está en modo de sensado, indicando los pasos para volver a acceder al HMI.

Para poder descargar los archivos que contienen los datos guardados, se tiene que acceder a la segunda pestaña del módulo principal, en donde se muestra una lista de los archivos presentes en la tarjeta microSD.

Figura 31 – "HMI: Pestaña de descarga de archivos"

Presionando el botón de eliminar, aparecerá un mensaje advirtiendo al usuario que, una vez eliminado el archivo, no se podrá recuperar. Si se presiona el botón descargar, aparecerá la ventana del sistema operativo que se esté utilizando, para guardar el archivo en su sistema de almacenamiento. Si el archivo tiene un tamaño considerable, aparecerá una animación en el HMI indicando que se está descargando un archivo, y a su vez bloqueando el HMI para que el usuario no pueda realizar ninguna modificación. Una vez finalizada la descarga, la animación desaparece volviendo a mostrar la pantalla principal del HMI y permitiendo al usuario continuar con la configuración.

Finalmente, en la tercera y última pestaña, se muestra el módulo para cambiar el usuario y la contraseña. El usuario en este caso es el nombre que tendrá la red para conectarnos al dispositivo, y la contraseña es la utilizada para entrar en esa red.

Figura 32 – "HMI: Pestaña de cambio de usuario y contraseña"

Por seguridad, se recomienda al usuario utilizar en su contraseña al menos 8 caracteres, y elegir algún carácter especial.

2.3.2 Módulo microSD

A. Investigación de componentes

I. Análisis y selección

Para guardar los datos recopilados por las entradas analógicas del dispositivo, se necesita un sistema de almacenamiento, así como también un dispositivo donde se guarden los mismos. Debido a que los datos se guardan en una línea de texto en un archivo con extensión ".txt", los mismos no ocupan un espacio considerable. Sabiendo esto, se eligió como dispositivo de almacenamiento una tarjeta microSD de 64 [GB], para que, en términos prácticos, el espacio de almacenamiento no se termine "nunca".

Una vez elegido el dispositivo de almacenamiento, se investigaron distintos sistemas de almacenamiento, y para simplificar el diseño y disminuir el tiempo de ejecución del proyecto, se decidió utilizar un módulo comercial, el cual ya tiene todos los elementos necesarios para acoplarse a cualquier microcontrolador, y se comunica mediante el protocolo SPI.

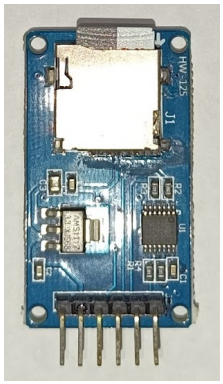

Figura 33 – "Modulo microSD"

B. Propuestas de circuitos o esquemas.

I. Pruebas, experiencias.

El módulo lector de memorias microSD, tiene 6 pines, de los cuales 4 están conectados al ESP32, para poder comunicarse mediante el protocolo SPI.
A continuación, se nombran los mismos:

Módulo Micro SD	ESP32
Vcc	5 [V]
CS (chip select)	GPIO 5
MOSI (master output slave input)	GPIO 23
CLK (clock)	GPIO 18
MISO (master input slave output)	GPIO 19
GND	GND

Tabla 3 – "Pinout Modulo microSD"

Para probar el módulo, se conectó el kit de desarrollo del ESP32 junto con el módulo en una placa de prueba, y se descargó al mismo un ejemplo de programa que está incluido en la librería de Arduino, qué es la encargada de todas las acciones que se realizan con el módulo, así como también la escritura dentro de la tarjeta microSD. Se utilizó una tarjeta 64 [GB], que luego, para verificar que todo funcione correctamente, se vinculó a una computadora y se revisó su contenido, comparándolo con el que debería haber escrito el programa.

II. Problemas y soluciones implementadas

El primer problema que se encontró fue que los datos no se guardaban correctamente en la tarjeta. Utilizando el monitor serial del editor de código que se utilizó para este proyecto, se observó que el módulo no se inicializó correctamente. El problema estaba en la tensión

de entrada del módulo, ya que obligatoriamente debe ser de 5 [V]. Una vez corregido esto, se inicializó correctamente y guardó los datos sin corromperse.

Teniendo en cuenta que a futuro el lector de tarjetas puede volver a fallar por un problema de alimentación, y siendo un punto crítico del Datalogger (ya que si el mismo no puede guardar los datos es obsoleto), se agregó un led de error para que el usuario pueda tener un indicador visual si el mismo llegase a fallar, y pueda dar aviso para realizar el mantenimiento correspondiente, así como también una alarma de tipo buzzer.

C. Análisis del funcionamiento

Este módulo, como se explica en la parte del desarrollo de software, es inicializado por el microcontrolador en el comienzo del programa. En la inicialización se crean los directorios que contendrán los archivos que guardan los datos sensados, así como también el archivo de registro de eventos del microcontrolador (aquí se guardan eventos como, por ejemplo, cuando el microcontrolador se “despierta” para sensar datos, o cuando existen reinicios inesperados, etc.).

Una vez inicializado el módulo y creado los directorios, si existe un perfil de sensor configurado, se crea el archivo “.txt” que guardará los datos recopilados por el dispositivo. Este archivo, para su posterior procesamiento en Excel es del tipo CSV, y guarda los datos de la siguiente manera:

“Año, Mes, Dia, Hora, Minuto, Segundo, Valor ADC, Valor Convertido”

El valor convertido, es el valor del conversor analógico digital, transformado a una escala de 0 a 3.3 [V]. Este valor es necesario para luego adecuarlo a la variable original medida, para graficar los datos obtenidos.

Luego, mediante el HMI, podemos acceder a los directorios antes mencionados, y descargar estos archivos a nuestro smartphone o computadora, para su posterior procesamiento.

2.3.3 RTC: Real Time Clock

A. Investigación de componentes

I. Análisis y selección

Para el presente proyecto, se necesita llevar un registro del tiempo real, para que el usuario sepa en el momento que se recopilaron los datos, y realice los análisis correspondientes dependiendo el proceso sobre el cual se están obteniendo los datos.

En un principio se intentó utilizar el RTC interno del ESP32, pero esta alternativa se descartó debido a que la deriva de tiempo de este puede variar hasta 5 segundos por día, además de que se necesita otra versión de kit de desarrollo el cual tiene un pin para colocar la batería que mantiene energizado el módulo RTC.

Como segunda opción, existe la posibilidad de que el ESP32 se conecte a internet, y obtenga la fecha y hora actual de un servidor NTP, que es un protocolo de red. Esta opción es útil en lugares donde se tiene acceso a internet, pero este proyecto está planteado para que pueda ser utilizado en lugares de difícil acceso, por lo que mayoritariamente nos encontraremos sin acceso a internet o con señales muy débiles de WiFi.

Finalmente se decidió elegir la opción más viable para este proyecto, en cuestión de precisión y facilidad de uso, el módulo que contiene un integrado DS3231, el cual será desarrollado en el siguiente punto.

B. Propuestas de circuitos o esquemas.

I. Pruebas, experiencias.

El DS3231, de acuerdo con su hoja de datos, es un reloj de tiempo real, extremadamente preciso y de bajo costo, que se programa y comunica mediante el protocolo I2C y tiene integrado en su chip un cristal oscilador compensado en temperatura.

El circuito típico de aplicación es el siguiente:

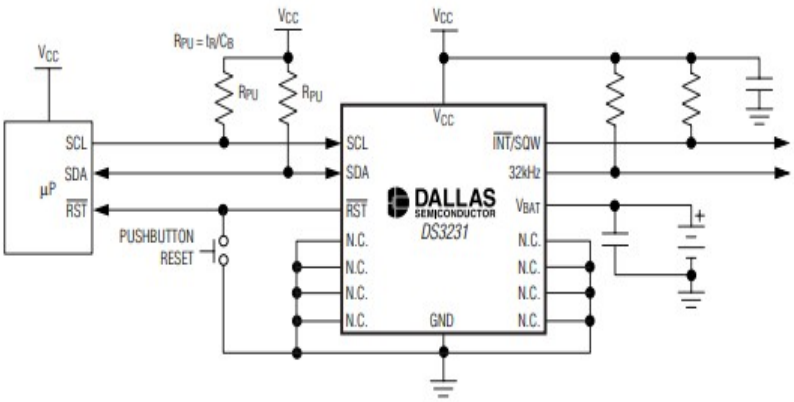

Figura 34 – "Circuito típico de aplicación DS3231 [4]"

Los pines SCL y SDA son los que se utilizan para comunicarse con el microcontrolador mediante el protocolo I2C. La dirección de este puede ser cambiada manipulando estos pines, pero por defecto es 0x68. Luego para mantener el tiempo actual, posee un pin Vbat, donde se coloca una batería o pila, que mantiene energizado el módulo, y por lo tanto su configuración. El pin INT/SQW (interrupción y onda cuadrada), como lo dice su

nombre, es el que se utiliza para generar interrupciones externas al dispositivo y también una onda cuadrada con frecuencia ajustable. Finalmente, el pin 32kHz, genera una onda sinusoidal de dicha frecuencia, para aplicaciones que requieran una fuente de reloj.

Para probar el módulo, se realizó un programa con el kit de desarrollo del ESP32 en el cual, utilizando la librería de Arduino, se configuraba la hora actual del dispositivo y la mostraba en pantalla.

Asimismo, se colocó una pila CR2032 en el zócalo que posee el módulo, el cual está conectado al pin Vbat mencionado previamente. Luego, quitando la alimentación de 5 [V] del módulo y volviendo a conectar, se comunicó nuevamente con el microcontrolador, y se verificó que la hora siga siendo la actual, y que el dispositivo no haya perdido su configuración.

II. Problemas y soluciones implementadas

Uno de los problemas encontrados fue el de ajustar la hora precisamente dentro del RTC. Se probaron distintos programas que venían dentro de la librería, pero la hora siempre quedaba con un error de 14 segundos. Se decidió fijar la hora mediante una línea de código (es decir, manualmente), calculando el tiempo de compilado del programa en Arduino, quedando como resultado el horario correcto.

C. Análisis del funcionamiento

Como parte del Datalogger, el módulo RTC cumple dos funciones fundamentales para su funcionamiento. La primera es suministrar la hora exacta a la cual se están obteniendo los datos, para guardarlos dentro del archivo en la tarjeta microSD.

Previamente a su montaje, debemos configurar la hora, tal como se explicó en el punto B.1. Gracias a la pila que se coloca en el módulo RTC, el cristal del DS3231 continúa oscilando, y mediante un contador, se compara la cantidad de oscilaciones del cristal, con un valor de referencia, el cual se encuentra guardado en la EEPROM. Este valor de referencia es el valor de la fecha y hora actual que se guardó por primera vez, actualizado gracias a las técnicas de compensación de temperatura que posee el DS3231. Dentro del programa, se verifica la correcta inicialización del RTC, y luego se realizan pedidos de hora actual mediante el protocolo I2C, mientras el dispositivo esté sensando y guardando datos.

La segunda función que cumple el módulo es la de “despertar” al microcontrolador, en el horario que previamente haya configurado el usuario. Esto se realiza mediante el pin INT/SQW. Internamente, el DS3231 tiene dos registros los cuales pueden guardar dos alarmas respectivamente. Mediante la librería de Arduino, se accede a uno de estos registros, y según la lógica del programa principal, se establece la hora a la cual se tiene que “despertar” el ESP32 para comenzar a sensar datos. Este valor se compara con la

hora actual dentro del DS3231, y cuando los dos valores son iguales, se genera un pulso de colector abierto en el pin previamente mencionado, el cual está vinculado a uno de los pines de “wake-up” del ESP32, provocando así que el dispositivo se encienda y comience a ejecutar su programa. Como ya se explicó previamente, este valor queda guardado y no se pierde, ante eventuales fallas en la alimentación del dispositivo.

2.4 Diseño Completo

A continuación, se muestra una imagen del diseño del circuito esquemático final implementado:

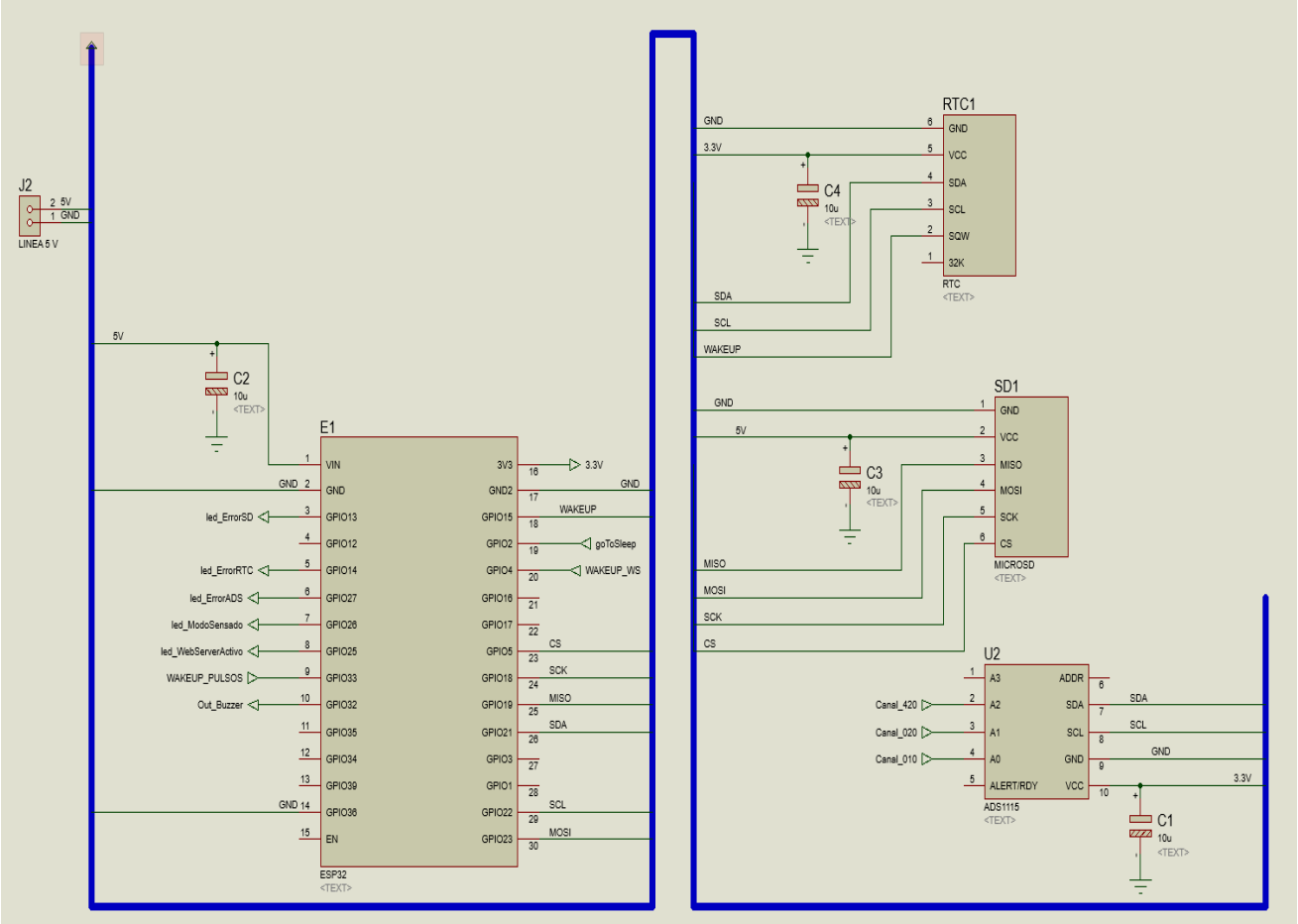

Figura 35 – “Circuito esquemático de ESP32 y periféricos: parte 1”

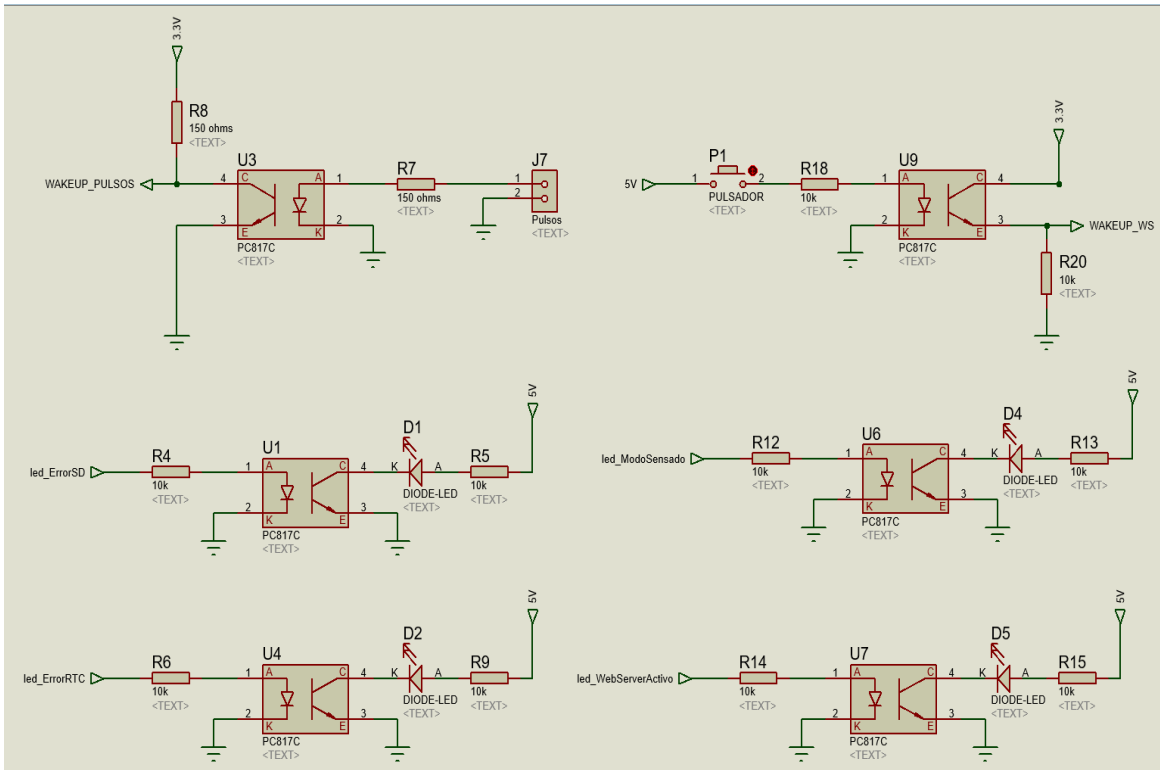

Figura 36 – "Circuito esquemático de ESP32 y periféricos: parte 2"

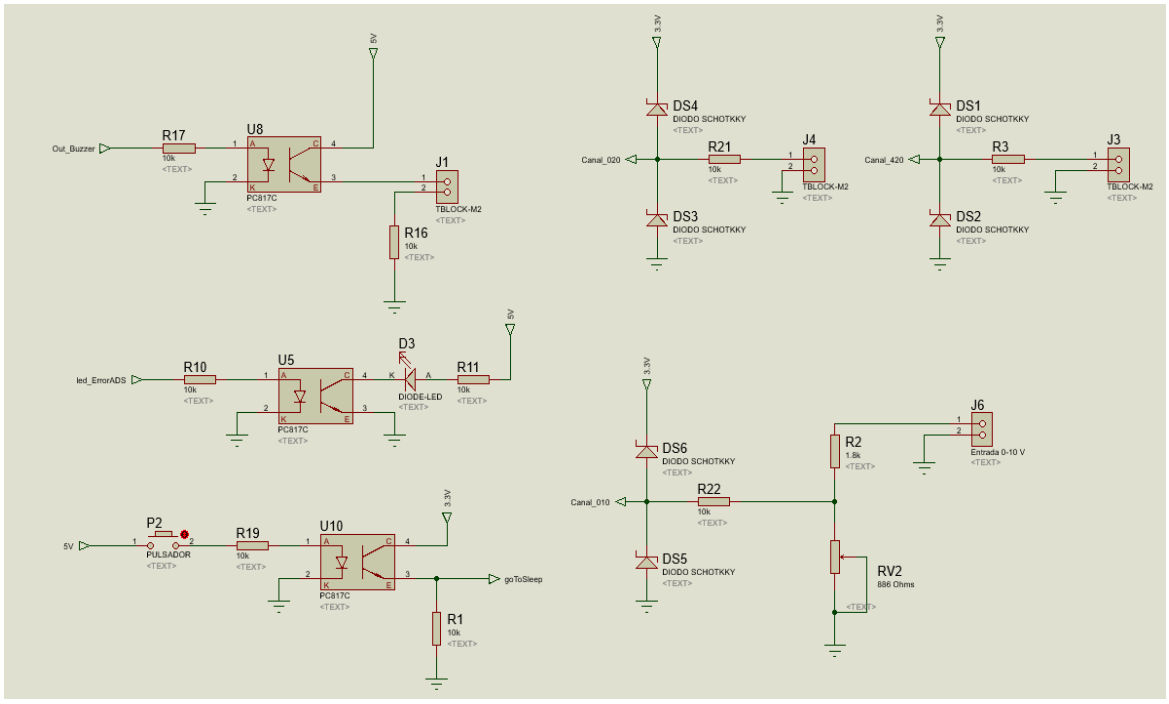

Figura 37 – "Circuito esquemático de ESP32 y periféricos: parte 3"

Los módulos de ESP32, y sus periféricos tuvieron que ser diseñados manualmente, ya que el programa utilizado no los contenía. El diseño impreso en la placa de cobre y luego soldado fue el siguiente:

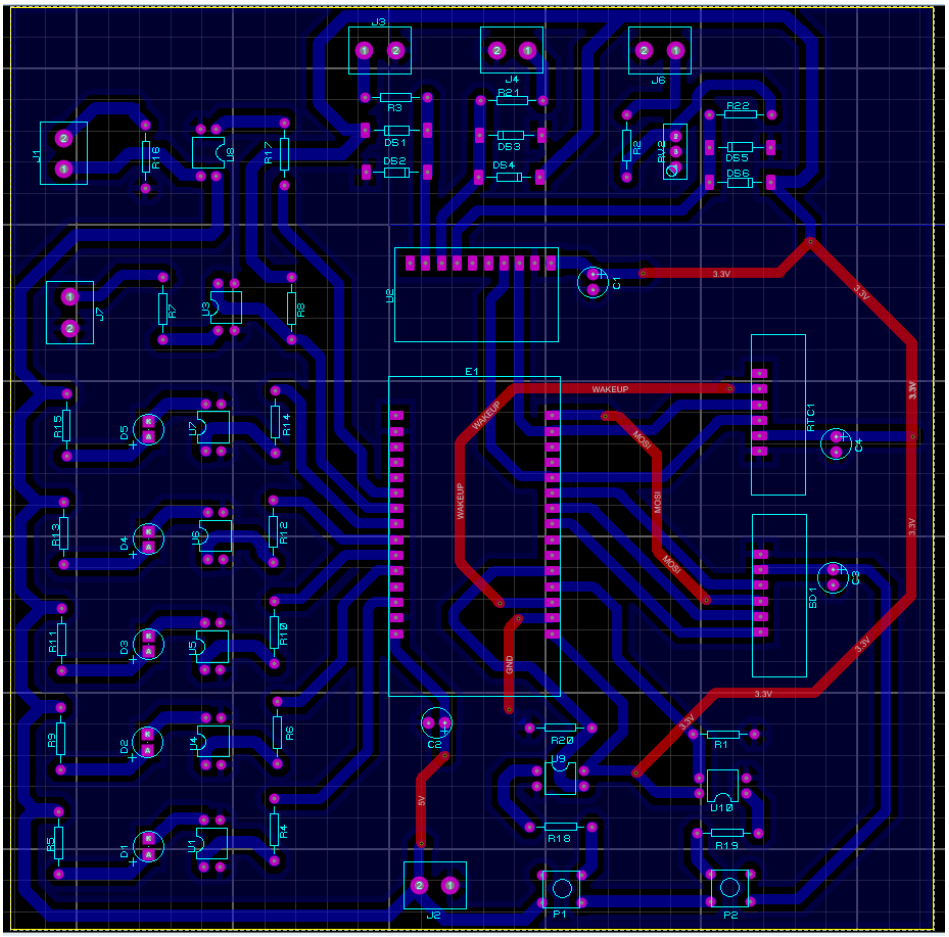

Figura 38 – "Circuito impreso final"

Vale destacar que, si bien se diseñó como una placa doble faz, para ahorrar gastos se podría haber colocado cables para reemplazar las pistas.

Luego de realizar el proceso de soldado de componentes, se buscó una caja de dimensiones acorde a lo que se necesitaba, y se modificó para montar la placa y sus periféricos, quedando como resultado lo siguiente:

Figura 39 – "Diseño final de dispositivo 1"

Figura 40 – "Diseño final de dispositivo 2"

El presente dispositivo es capaz de leer y guardar datos de tres tipos de señales eléctricas, como ya se ha mencionado previamente, lazo de corriente 4-20 [mA], 0-20 [mA] y lazo de tensión 0-10 [V], a una frecuencia de muestreo que varía de 1 a 59 segundos (procesos lentos). Posee también un agregado que detecta señales

totalizadoras, para poder guardar, por ejemplo, una señal de un caudalímetro, cuando por el mismo circula cierta cantidad de volumen de líquido. El ADC utilizado tiene una ganancia programable, la cual se ajustó a la tensión utilizada por los adaptadores de señal, quedando como resultado la equivalencia de 1 bit = 0.125 [mV]. La tensión de alimentación del dispositivo es de 5 [V], y posee una ficha USB macho, para poder conectar a cualquier fuente de tensión, como, por ejemplo, la que se utiliza para cargar las baterías de los celulares.

Capítulo 3: Resultados

En este apartado, se decidió mostrar las pruebas realizadas en campo del dispositivo armado en el presente proyecto. En una etapa temprana del desarrollo, se tuvo la posibilidad de asistir a la planta potabilizadora de la comuna de Humboldt, la cual tenía un controlador de flujo que detectaba el caudal instantáneo de agua que pasaba por la planta. En esta etapa se probó la señal totalizadora proveniente del controlador, que genera un pulso cada metro cúbico de agua que pasaba por el mismo. Esta señal se conectó a un prototipo del proyecto, que despertaba el microcontrolador y guardaba la fecha y horario exacto en que ocurrió el pulso.

Figura 41 – "Primer prototipo en prueba de campo"

Luego de recopilar datos por 3 días, se obtuvo un archivo de extensión ".txt" cuyos datos fueron procesados con Excel, para obtener gráficas que puedan visualizar el consumo de agua de la comuna en diversas franjas horarias.

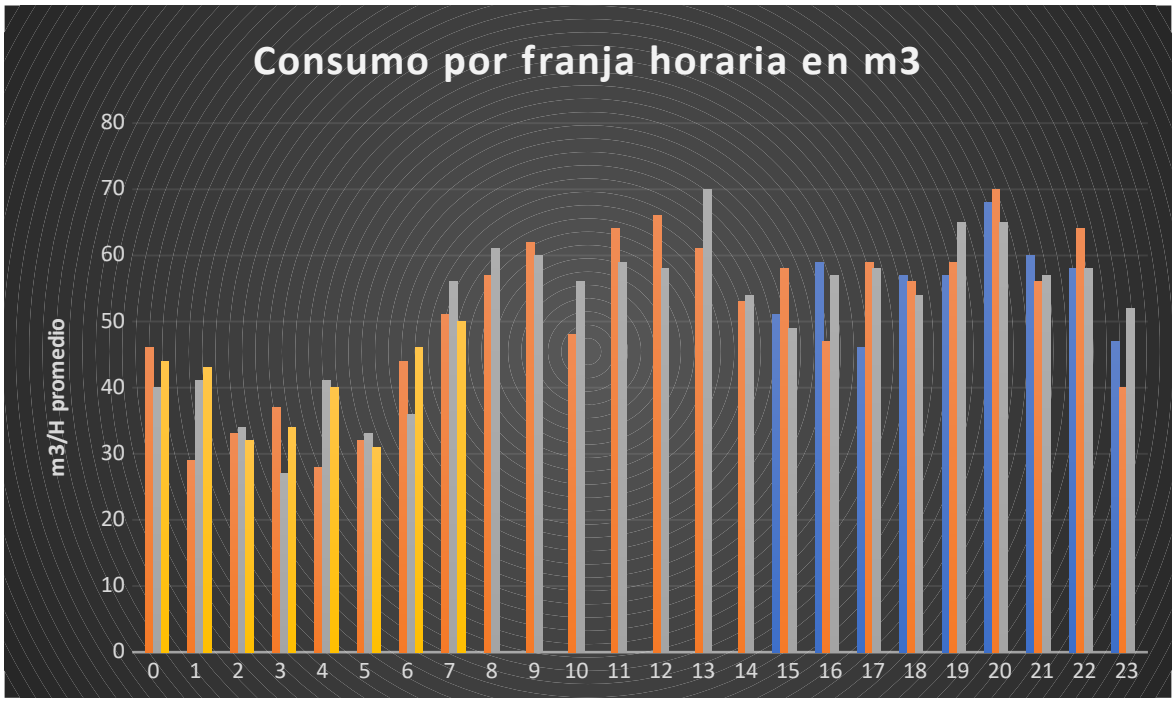

Figura 42 – "Grafico de datos obtenidos en primera prueba de campo planta potabilizadora Humboldt"

Una vez terminado el prototipo final, se pudo realizar otra prueba en campo, dentro de la fábrica Nestlé Purina, perteneciente a Nestlé Argentina S.A, lugar donde el que suscribe desarrolla tareas de mantenimiento eléctrico.
En esta prueba se utilizaron los siguientes materiales:

- Horno seco calibrador FLUKE 9100S
- Transductor de temperatura IFM TR2432
- PT1000 IFM TT1250
- Calibrador multifunción FLUKE 725
- Tester y fuente de 24 [V]

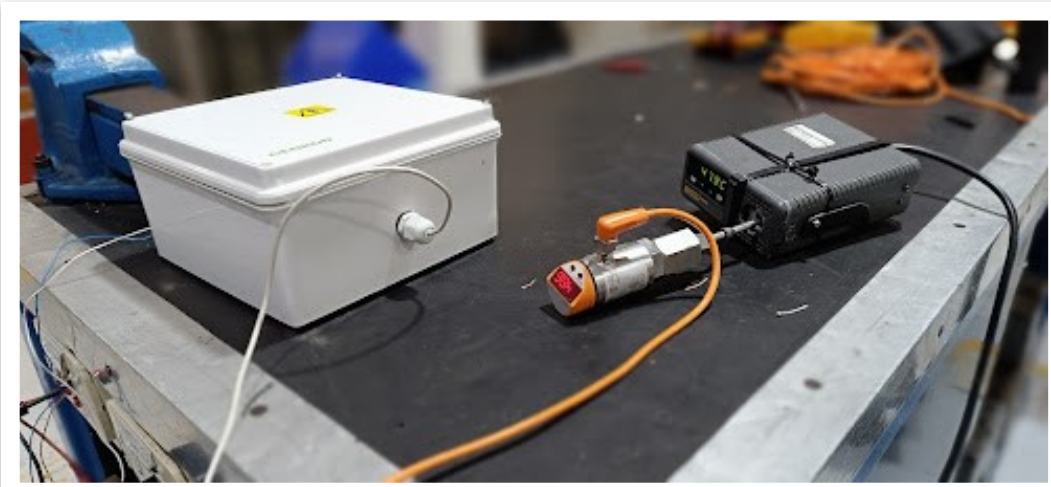

Figura 43 – "Diseño final en prueba de campo en planta Nestlé Purina"

La prueba consistió en, colocar el horno calibrador en 40 [°C], fijar un horario de despertador cercano al actual en ese momento, y luego variar la temperatura del horno a 80 [°C] e ir recopilando los datos a una frecuencia de muestreo de 1 segundo por muestra en el canal de lazo de corriente de 4-20 [mA]. Este proceso se realizó también a la inversa.

Para poder verificar las mediciones en todo el rango del lazo de corriente, se modificó el escalado del transductor de temperatura, para que 4 [mA] sea igual a 40 [°C] y 20 [mA] sea 80 [°C]. Las gráficas de los datos procesados se ven a continuación:

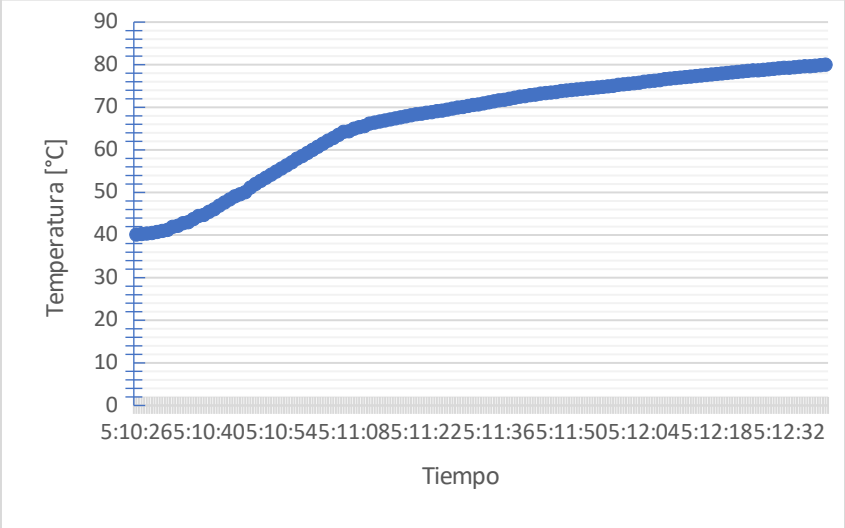

Figura 44 – "Grafico de temperatura de calentamiento de 40 a 80 °[C]"

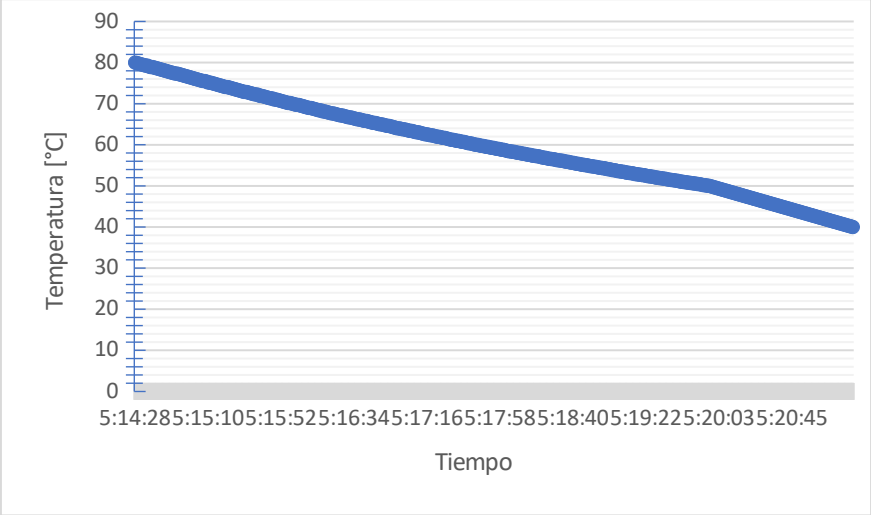

Figura 45 – "Grafico de temperatura de enfriamiento de 80 a 40 [°C]

En los gráficos generados se puede visualizar como es el sistema de calentamiento y enfriamiento del horno de calibración. Como se ve en la gráfica de calentamiento, el sistema de control del horno alcanza de manera más veloz el set point de temperatura (aproximadamente 3 min), y en el proceso de enfriamiento, es más lento (aproximadamente 6 min).

Capítulo 4: Análisis de Costos

4.1 Introducción

Para realizar el análisis de costos de materiales utilizados en el presente trabajo, se decidió utilizar precios en dólares de los materiales debido a que este proyecto comenzó en 2020, y como es de público conocimiento, los precios en moneda argentina en ese momento eran significativamente más bajos.

4.2 Costos del proyecto

A continuación, se muestran los costos de los materiales utilizados para la realización del presente proyecto.

Material	Costo (USD)	Observaciones
Placa de desarrollo ESP32	$2.2	A cargo de AutoSolve
Modulo RTC DS3231	$1.86	A cargo de AutoSolve
Módulo microSD	$0.42	A cargo de AutoSolve
ADC ADS1115	$2.31	A cargo del que suscribe
Modulo Step Up XL XL6009	$0.43	A cargo de AutoSolve
Caja estanca 20x20x10 cm	$11	A cargo del que suscribe
Componentes misceláneos	$15	A cargo del que suscribe

Total	$33.22

Tabla 4 – "Costos de materiales para la realización del proyecto"

Para obtener el costo total del trabajo realizado, también se debe tener en cuenta las horas hombre dedicadas en el diseño, posterior montaje y pruebas del dispositivo. Para poder calcular el monto que se le debería pagar a un ingeniero para poder realizar el trabajo, se llegó a un tiempo estimativo de finalización de este. Entonces, si la persona que lo ejecuta le dedica un total de 8 horas diarias, 5 días a la semana, este proyecto podría terminarse en un total de 4 meses. El monto final que se le debería pagar a la persona encargada es de $2.400.000 pesos argentinos ($600.000 pesos por mes).

4.3 Plan de venta y amortización de la inversión

Para poder amortizar la inversión realizada en el desarrollo del dispositivo, se planea ofrecer el mismo como un servicio en conjunto con las obras que realiza la consultora AutoSolve.

Debido a que cierto porcentaje de los clientes de la empresa son el público objetivo de este dispositivo, el primer paso de búsqueda y contacto no sería de gran dificultad. La mayoría de las comunas donde se han instalado sistemas de control para sus plantas potabilizadoras necesitan un control sobre el flujo de agua que se consume, por lo que se ofrecería como un servicio extra, alquilando el dispositivo para su uso. Una vez que el usuario obtenga los datos y pueda observar la funcionalidad del dispositivo, se negociaría la venta y posterior servicio de soporte técnico. A su vez, si el usuario no posee la capacidad o los conocimientos, se puede ofrecer el procesamiento e interpretación de los datos obtenidos, como otro servicio extra fuera de lo que es el dispositivo.

Otra manera de generar visualización sobre el dispositivo es presentarlo en diversos congresos y simposios, operando de la misma manera, primero ofreciendo su alquiler y luego su posterior venta.

Capítulo 5: Discusión y Conclusión.

El dispositivo realizado en el presente proyecto final cumplió con las expectativas planteadas por la empresa AutoSolve, las cuales eran diseñar un recopilador de datos confiable, de múltiples entradas, y que sea configurable desde un servidor web. Lo que hace diferente a este dispositivo de los que se encuentran en el mercado, es que la gran mayoría de estos están abocados a una sola señal eléctrica y, además, muy pocos o casi ninguno tienen la opción de ser configurados para sensar en ciertos horarios, ni tampoco la descarga de los archivos en tiempo real, sin tener que desmontar el dispositivo de su entorno.

Si bien este dispositivo se pensó para procesos lentos, se puede adaptar a las condiciones del cliente. La frecuencia de muestreo puede ser modificada para llegar a valores mucho más pequeños, así como también el tiempo de sensado puede ser aumentado.

Para la realización de este proyecto se encontraron con varias dificultades, la gran mayoría relacionadas al microcontrolador elegido. Si bien el ESP32 tiene muchas prestaciones, y hoy en día es muy utilizado en aplicaciones de IOT, se debe conocer a fondo los requerimientos necesarios para que funcione correctamente, lo cual se toma como aprendizaje enriquecedor para futuros diseños. No solo se aprendió sobre este microcontrolador, sino que también se programó en un lenguaje nuevo (HTML y JavaScript), el cual no se dicta en la carrera de Ingeniería Electrónica, y fue muy útil para la elaboración del HMI.

Como mejoras a futuro, se debería rediseñar el proyecto para utilizar el ESP32 fuera de su placa de desarrollo, así como también utilizar componentes más pequeños para optimizar espacio y así reducir costos de diseño. Dentro del código implementado, se hicieron mejoras para reducir el consumo eléctrico del dispositivo, por si en algún momento se plantea el uso de este alimentado mediante baterías. En la parte de software, se podría agregar en el servidor web, o en una aplicación aparte, un graficador de datos para darle autonomía al usuario, y que pueda ver los datos en tiempo real.

Este diseño se puede adaptar a las necesidades del usuario, las características de este (frecuencia de muestreo, tiempo de sensado, cantidad de canales, etc.) fueron fijadas para darle un marco al proyecto que se presenta en este informe.

Para realizar una comparación con los dataloggers encontrados en el estudio de mercado, se decidió tomar como parámetro la única empresa argentina que se pudo encontrar, que desarrolla y fabrica sus propios dataloggers, Alfa instrumentos. Esta empresa, posee una gama de dataloggers que tienen distintas características. A continuación, se realiza una comparativa con respecto a 3 productos fabricados por la empresa nombrada.

Logger - DA	
Ventajas	Desventajas
Menor dimensión (70x60x35 mm)	Memoria de 32000 registros
Alimentación por batería 3.6 V	ADC de resolución de 14 bits
Entrada para PT100	Dos canales de medición

Tabla 5 – "Ventajas y desventajas de Logger – DA"

El Logger-DA tiene unas dimensiones más pequeñas y cuenta con alimentación a batería con una autonomía estimada de 6 meses. Como desventajas, solo tiene una memoria de 32000 registros (en el caso del proyecto desarrollado es prácticamente infinito el almacenamiento de la tarjeta microSD), así como también su ADC es de 14 bits, mientras que el ADS1115 posee 16 bits de resolución. Por último, su frecuencia de muestreo es programable a partir de 1 [s], al igual que lo desarrollado en este trabajo final.

Logger DLW-V005-TH	
Ventajas	Desventajas
Sensores de temperatura y humedad integrados	1 solo canal además de los sensores integrados
Visualización de datos en servidor web	Se necesitan 24 V de tensión de entrada
Salida digital (opcional)	Rangos de temperatura y humedad acotados

Tabla 6 – "Ventajas y desventajas de Logger DLW-V005-TH"

El Logger DLW-V005-TH posee sensores integrados para guardar datos de temperatura y humedad, y a su vez pueden ser observados en tiempo real dentro de un servidor web. También posee una salida digital configurable, por ejemplo, para utilizarla como alarma. Como desventajas, solo tiene un canal analógico para otro tipo de sensor, además de que se necesita una fuente de 24 [V] para utilizarlo, mientras que para energizar el Datalogger Universal solo hace falta una fuente de 5 [V], la cual puede ser fácilmente implementada con un cargador de smartphone.

Logger DLW-V005-QT

Ventajas	Desventajas
Sensor de caudal integrado	Resolución de 12 bits
1 salida digital y 1 analógica (opcional)	1 canal de entrada analógico y 1 digital
Visualización de caudal y volumen en servidor web	
Frecuencia de muestreo de 1000 muestras/segundos	
Soporta comunicación Modbus	

Tabla 7– "Ventajas y desventajas de Logger DLW-V005-QT"

Por último, el Logger DLW-V005-QT, el mas completo de la línea de dataloggers, posee funcionalidades extra, como salidas digitales y analógicas, una frecuencia de muestreo mucha mas alta, y se puede comunicar a través del protocolo modbus, y como desventajas tiene menos entradas con menos resolución.

Se puede concluir que el prototipo desarrollado en este trabajo puede competir con los productos existentes en el mercado, pero también vale destacar que su elección, depende la aplicación y lo que precise el usuario. También es cierto que al utilizar el microcontrolador ESP32, muchas de las funcionalidades que tienen los otros dataloggers se podrían implementar a futuro, como por ejemplo la comunicación por modbus, salidas digitales o analógicas, y aumentar la frecuencia de muestreo de datos.

Capítulo 6: Literatura Citada.

- [1] Espressif, “ESP32 Series”, ESP32 datasheet, versión 4.6 (2024).
- [2] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/user-guide-devkitm-1.html.
- [3] Sharp, A., & Vagapov, Y. (2017). Comparative analysis and practical implementation of the ESP32 microcontroller module for the Internet of Things.
- [4] Dallas Semiconductor, “Extremely Accurate I2C- Integrated RTC/TCXO/Crystal”, DS3231 datasheet, (2013).
- Texas Instruments, “ADS111x Ultra-Small, Low-Power, I 2C-Compatible, 860-SPS, 16-Bit ADCs With Internal Reference, Oscillator, and Programmable Comparator”, ADS1115 datasheet, (2009).
- Coughlin, R. F., Driscoll, F. F., & Flores, G. A. (1999). Amplificadores operacionales y circuitos integrados lineales (Vol. 5). Prentice Hall.
- http://www.alfainstrumentos.com/Dataloggers%20Linea%20Agron.html
- https://developer.mozilla.org/es/docs/Web/JavaScript