

INSTITUTO TECNOLÓGICO DE DURANGO



DEPARTAMENTO DE INGENIERIAS ELECTRICA Y ELECTRONICA

“Instrumentación y control de un sistema solar de generación de aire sintético para aplicaciones de secado de productos agroindustriales”

Reporte Final de Residencia Profesional

Empresa: DPTO. DE INGENIERIA SUSTENTABLE, CIMAV SEDE
DURANGO

Presenta:

Alan Yeray Olivas Díaz N.C: 17041626

17041626@itdurango.edu.mx

Carrera: Ingeniería Electrónica

Asesor Interno: Dra. Yolocuauhtli Salazar Muñoz

Asesor Externo: Dr. Erick César López Vidaña

Periodo de realización: Semestre enero – junio 2023

Carta Presentación

	INSTITUTO TECNOLÓGICO DE DURANGO	
Código: ITD-AC-PO-08-03	Formato de Carta de Agradecimiento Referencia a la Norma ISO 9001:2015 8.2.1, 8.2.2, 8.2.4, 8.5.1	Revisión: 0

Victoria de Durango, Dgo., 14/03/2023

DEPARTAMENTO: GESTIÓN TEC. Y VINC.
No. DE OFICIO: P.P.P. 2378/2023

Asunto: Presentación del estudiante
y agradecimiento

C. DR. IGNACIO RAMIRO MARTÍN DOMÍNGUEZ
JEFE DE DEPTO. DE INGENIERÍA SUSTENTABLE
CIMAV UNIDAD DURANGO
P R E S E N T E

El Instituto Tecnológico de Durango tiene a bien **presentar** a sus finas atenciones al (la) **C. OLIVAS DÍAZ ALAN YERAY**, con número de control **17041626**, de la carrera de **Ingeniería Electrónica**, quien desea desarrollar en ese organismo el proyecto de Residencias Profesionales, denominado "**Instrumentación y control de un sistema solar de generación de aire sintético para aplicaciones de secado de productos agroindustriales.**", cubriendo un total de 4 meses y 500 horas mínimo, en un periodo de cuatro a seis meses.

Es importante hacer de su conocimiento que nuestro(a) alumno(a) se encuentra inscrito(a) en esta institución educativa y cuenta con un seguro contra accidentes personales con la empresa THONA seguros, según póliza No. AP-TEC-056-03 y seguridad social IMSS, cuyo número es **26179973032**.

Asimismo, hacemos patente nuestro sincero **agradecimiento** por su buena disposición y colaboración para que nuestros alumnos, aún estando en proceso de formación, desarrollen un proyecto de trabajo profesional, donde puedan aplicar el conocimiento y el trabajo en el campo de acción en el que se desenvolverán como futuros profesionistas.

Al vernos favorecidos con su participación en nuestro objetivo, sólo nos resta manifestarle la seguridad de nuestra más atenta y distinguida consideración.

ATENTAMENTE
EXCELENCIA EN EDUCACIÓN TECNOLÓGICA®

M.C. CARLOS EDUARDO MERAZ CASTRO
JEFE DEL DÉPTO. DE GESTIÓN TECNOLÓGICA Y VINCULACIÓN
DEL INSTITUTO TECNOLÓGICO DE DURANGO

CEMC/CIBC



Carta Aceptacion



GOBIERNO DE
MÉXICO



CONACYT
Consejo Nacional de Ciencia y Tecnología



Coordinación de estudios de Posgrado

Oficio PO - 374/2023

Chihuahua, Chih, a 3 de febrero de 2023.

A quien corresponda
P r e s e n t e .

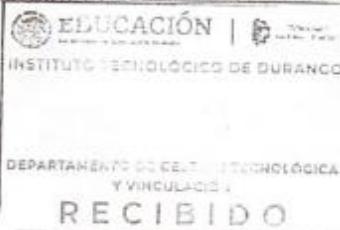
Por medio de la presente hago constar que el alumno **Alan Yeray Olivas Diaz**, perteneciente a **Instituto Tecnológico de Durango**, de la carrera de **Ing. Electrónica** y con No. de control **17041626** está aceptado en este Centro de Investigación para realizar **Residencias Profesionales**, en el departamento de **Departamento de Ingeniería Sustentable**, bajo la supervisión de **Erick César López Vidaña**, cubriendo un total de **500 Hrs.**, dentro del periodo comprendido del **30 de enero del 2023 al 9 de junio del 2023** del presente año en el siguiente horario de **9 a 17 h**, en este Centro de Investigación, desarrollando el siguiente proyecto:

Instrumentación y control de un sistema solar de generación de aire sintético para aplicaciones de secado de productos agroindustriales.

Se extiende la presente constancia en la ciudad de Chihuahua, Chihuahua el dia 3 del mes de febrero de 2023, para los fines legales a que haya lugar.

Atentamente

M.C. Jorge Alberto Escobedo Bretado
Coordinador Académico Unidad Durango



cimav
Centro de Investigación
en Materiales Avanzados, S.C.

**UNIDAD
DURANGO**



Miguel de Cervantes No. 120, Complejo Industrial Chihuahua, CP. 31136, Chihuahua, Chih., México.
Tel: (634) 439 1100 www.cimav.edu.mx

Carta Liberación



GOBIERNO DE
MÉXICO



CONAHCYT
CONSEJO NACIONAL DE HUMANIDADES
CIENCIAS Y TECNOLOGÍAS



Coordinación de estudios de Posgrado
Oficio PO - 1360/2023

Chihuahua, Chih, a 13 de julio de 2023.

M.C. Carlos Eduardo Meraz Castro
Jefe del departamento de Vinculación

P r e s e n t e .

Por medio de la presente hago constar que el alumno **Alan Yeray Olivas Díaz**, de la carrera de **Ing. Electrónica** perteneciente a **Instituto Tecnológico de Durango** y con número de control **17041626** realizó **Residencias Profesionales**, dentro del periodo comprendido del **30 de enero de 2023 al 9 de junio de 2023** cubriendo un total de **500** horas en este Centro de Investigación, desarrollando las siguientes actividades:

Instrumentación y control de un sistema solar de generación de aire sintético para aplicaciones de secado de productos agroindustriales.

Que nos reporta con resultados muy satisfactorios el asesor **Erick César López Vidaña**, una puntuación de **100**, por lo que no tenemos reserva alguna en felicitarlo por su excelente formación.

Se extiende la presente constancia en la ciudad de Chihuahua, Chihuahua el dia 13 del mes de julio de 2023, para los fines legales a que haya lugar.

A t e n t a m e n t e

M.C. Jorge Alberto Escobedo Bretado
Coordinador Académico Unidad Durango



Miguel de Cervantes No. 120, Complejo Industrial Chihuahua, CP. 31136, Chihuahua, Chih., México.
Tel: (614) 439 1100 www.cimav.edu.mx

V o. Bo

Erick César López Vidaña
Asesor Responsable
Centro de Investigación
en Materiales Avanzados, S.C.

**UNIDAD
DURANGO**



	INSTITUTO TECNOLÓGICO DE DURANGO	
Código: ITD-AC-PO-08-07	Formato de Evaluación de Reporte Final de Residencia Profesional Referencia a la Norma ISO 9001:2015 8.2.1, 8.2.2, 8.2.4, 8.5.1	Revisión: O

EVALUACIÓN DE REPORTE FINAL DE RESIDENCIA PROFESIONAL

Nombre del Residente: Alan Veray Olvera Diaz Número de control: 17041626
 Nombre del proyecto: Instrumentación y control de un sistema sobre de generación de arte sintético
 Programa Educativo: Ing. Electrónica para aplicaciones de sonido de productos digitales
 Periodo de realización de la Residencia Profesional: 30 de Enero del 2023 al 9 de Junio del 2023
 Calificación Final (promedio de ambas evaluaciones):

En qué medida el residente cumple con lo siguiente		Valor	Evaluación
Criterios a evaluar			
Evaluación por el asesor externo	Portada.	2	2
	Agradecimientos.	2	0
	Resumen.	2	1
	Índice.	2	2
	Introducción.	2	2
	Problemas a resolver, priorizándolos.	5	4
	Objetivos.	5	5
	Marco teórico (fundamentos teóricos)	10	8
	Procedimiento y descripción de las actividades realizadas.	5	5
	Resultados	45	45
	Conclusiones, recomendaciones y experiencia profesional adquirida.	15	13
	Competencias desarrolladas y/o aplicadas.	3	3
	Fuentes de información	2	1
	Calificación total		100 90

Observaciones:

Eziel C. López Vidriño

Centro de Investigación
en Materiales Avanzados, S. C. / Dependencia
Sello de la Institución

10 - Agosto 2023
Fecha de Evaluación

En qué medida el residente cumple con lo siguiente		Valor	Evaluación
Criterios a evaluar			
Evaluación por el asesor interno	Portada.	2	2
	Agradecimientos.	2	0
	Resumen.	2	1
	Índice.	2	2
	Introducción.	2	2
	Problemas a resolver, priorizándolos.	5	4
	Objetivos.	5	5
	Marco teórico (fundamentos teóricos)	10	8
	Procedimiento y descripción de las actividades realizadas.	5	5
	Resultados	45	45
	Conclusiones, recomendaciones y experiencia profesional adquirida.	15	13
	Competencias desarrolladas y/o aplicadas.	3	3
	Fuentes de información	2	1
	Calificación total		100 91

Observaciones:

Eziel C. López Vidriño



10 - 08 - 2023
Fecha de Evaluación

RESUMEN

El secado de alimentos representa uno de los procesos energéticos extensivos en su uso, donde aproximadamente el 30% de la producción mundial de energía es utilizada por el sector agrícola, donde una parte 3,62% se destina al secado de productos agrícolas. La práctica actual se basa en sistemas de calentamiento de aire con gas licuado de petróleo (gas L.P o LPG). Tecnologías como estas requieren un coste de inversión extra para lograr un aprovechamiento eficiente y lograr una rentabilidad relevante.

El objetivo es instrumentar un emulador de aire sintético que sea capaz de generar las condiciones características de una zona geográfica o localidad en específico.

Para poder realizarlo se debe de contar con el material necesario para poder emular las condiciones climáticas, como los sensores correspondientes y actuadores, donde serán instalados en el túnel de viento previamente construido en las instalaciones de CIMAV Unidad Durango.

CONTENIDO

RESUMEN	6
CONTENIDO	7
INDICE DE FIGURAS	9
1 INTRODUCCIÓN	11
1.1 DESCRIPCIÓN DE LA EMPRESA	11
1.1.1 MISIÓN.....	12
1.1.2 VISIÓN	12
1.2 PROBLEMA A RESOLVER	13
1.3 OBJETIVOS	15
1.3.1 OBJETIVO GENERAL	15
1.3.2 OBJETIVOS ESPECÍFICOS.....	15
1.4 JUSTIFICACIÓN	15
1.5 ALCANCES Y LIMITACIONES DEL PROYECTO	17
1.5.1 ALCANCES	17
1.5.2 LIMITACIONES	17
2 MARCO TEÓRICO	18
2.1 SECADO DE PRODUCTOS AGROIDUSTRIALES	18
2.2 PSICROMETRIA	19
2.3 TEORÍA SOBRE MICROCONTROLADORES	20
2.3.2 PROTOCOLOS DE COMUNICACIÓN PARA EL MICROCONTROLADOR.....	23
2.3.3 PLATAFORMAS DE DESARROLLO.....	24
2.4 TEORÍA SOBRE PAGINAS WEB	25
2.4.1 SERVIDOR WEB	25
2.4.2 LENGUAJES DE PROGRAMACION PARA WEB.....	26
2.4.3 TECNOLOGIAS PARA PROTOCOLOS WEB.....	26
3 PROCEDIMIENTO, DESCRIPCION DE ACTIVIDADES REALIZADAS	28
3.1 EQUIPOS	28
3.1.1 TUNEL DE VIENTO	28
3.1.2 SISTEMA DE CALENTAMIENTO AGUA/AIRE MULTIPROPÓSITO(SCAAM)	30
3.1.3 INSTRUMENTOS DE MEDICIÓN UTILIZADOS	32
3.1.4 SENsoRES UTILIZADOS	32
3.1.5 ACTUADORES UTILIZADOS	34
3.2 MATERIALES UTILIZADOS	35
3.3 METODOLOGIA	36
3.3.1 DISEÑO SENSOR DE TEMPERATURA Y HUMEDAD SHT40	36
3.3.2 CONSTRUCCIÓN DE LA PLACA DE CIRCUITO IMPRESO DEL MÓDULO DE TEMPERATURA Y HUMEDAD RELATIVA.....	38
3.3.3 INSTALACIÓN DEL VENTILADOR	48
3.3.4 SENSOR SUMERGIBLE DE TEMPERATURA DS18B20	71
3.3.5 PROGRAMACIÓN DE INTERFAZ DE USUARIO LOCAL	74
3.3.6 INSTALACIÓN DE TUBERÍAS	76

3.3.7	PAGINA WEB COMO INTERFAZ EN LA NUBE	77
4	RESULTADOS	83
4.1.1	SENSOR DE TEMPERATURA Y HUMEDAD RELATIVA SHT40.	83
4.1.2	SENSOR VELOCIDAD de viento FH400.....	84
4.1.3	VARIADOR DE FRECUENCIA, MOTOR TRIFASICO Y VENTILADOR.....	86
4.1.4	SENSOR DE TEMPERATURA DS18B20	88
4.1.5	IMPLEMENTACION DEL CIRCUITO GENERAL	89
4.1.6	INTERFAZ DE USUARIO EN LA NUBE	90
5	CONCLUSION	92
6	REFERENCIAS	93
7	ANEXOS	98
7.1.1	CODIGO LIBRERÍA SHT40.H.	98
7.1.2	CODIGO SHT40.CPP	99
7.1.3	CODIGO LIBRERÍA VARIADOR DE FRECUENCIA TECO_L510.H	102
7.1.4	CODIGO LIBRERÍA VARIADOR DE FRECUENCIA TECO_L510.H	103
7.1.5	CÓDIGO LIBRERIA FH400.H.....	107
7.1.6	CÓDIGO LIBRERIA FH400.CPP	108
7.1.7	CÓDIGO INTERFAZ DE USUARIO LOCAL.....	110
7.1.8	CODIGO INTERFAZ DE USUARIO EN LA NUBE BACK-END	127
7.1.9	CODIGO PARA INTERFAZ DE USARIO FRONT-END	146
7.1.10	CÓDIGO GRAFICAS SENsoRES EN MATLAB.....	164

INDICE DE FIGURAS

Figura 1: Vista satelital del Centro de Investigación CIMA (Latitud: 23,9929447, Longitud: -104.72651390902162).....	12
Figura 2: Paquete trama de datos	24
Figura 3: Túnel de viento en 3D	28
Figura 4: Medidas Túnel de Viento	29
Figura 5: Medidas patas del túnel del viento	30
Figura 6: Render SCAAM	31
Figura 7: Sensor FH400	33
Figura 8: Muestra Sensor IC Temperatura y Humedad Relativa.....	33
Figura 9: Sensor DS18B20	34
Figura 10: Diagrama del sensor de temperatura y humedad	37
Figura 11: Esquemático general para el microcontrolador Esp32 y el sensor de temperatura.....	37
Figura 12: Muestra de Ruteo de Sensor de Temperatura y Humedad Relativa	38
Figura 13: Configuración de impresión de circuito en Altium Designer	39
Figura 14: Selección de capas de interés.....	40
Figura 15: Resultado Final para Impresión.....	40
Figura 16: Resultado del método de planchado	42
Figura 17: Resultado Final Sensores de Temperatura y Humedad relativa.....	42
Figura 18: Vista por detrás sensor de temperatura y humedad relativa.....	43
Figura 19: Instalación de sensores en forma Daisy Chain	44
Figura 20: Ubicación de Sensores en el túnel de viento con su dirección	44
Figura 21: Vista previa de instalación de cable para sobre el túnel de viento para sensores.....	45
Figura 22: Muestra de trama recibida en la terminal de PlatformIO a través del puerto serial	46
Figura 23: Ecuaciones para conversión.....	46
Figura 24: Diagrama de Flujo para el sensor SHT40.	47
Figura 25: Código para librería	48
Figura 26: Motor trifásico.	49
Figura 27: Características Motor trifásico	49
Figura 28: Conexión tipo Estrella	50
Figura 29: Cableado de la conexión tipo estrella	50
Figura 30: Panel para programar el dimensionar el variador	51
Figura 31: Configuración 1	52
Figura 32: Configuración 2	52
Figura 33: Configuración 3	52
Figura 34: Configuración 4	53
Figura 35: Configuración 5	53
Figura 36: Configuración 6	53
Figura 37: Configuración 7	54
Figura 38: Configuración 9	54
Figura 39: Configuración 10	54
Figura 40: Configuración 11	55
Figura 41: Muestra de configuración del Variador	55
Figura 42: Características técnicas de comunicación	56
Figura 43: Características técnicas de comunicación	57
Figura 44: Diagrama de flujo programación de variador de frecuencia.....	59
Figura 45: Instrucción de inicialización de librería.	60
Figura 46: Analizador Lógico en UART Y PIN 18	61
Figura 47: RJ45.....	62
Figura 48: Conexión RS485	62
Figura 49: Circuito General	63
Figura 50: Ubicación F400.	64
Figura 51: Orientación FH400.....	65
Figura 52: Instalación FH400	65

Figura 53: Inicializa librería para el sensor FH400	66
Figura 54: Formula para conversión de velocidad de viento.....	67
Figura 55: Compensación	67
Figura 56: Conversión de Temperatura	68
Figura 57: Compensación de temperatura	68
Figura 58: Formula para conversión de humedad relativa	68
Figura 59: Diagrama de Flujo para programación para sensor FH400.....	69
Figura 60: Circuito general con sensor FH400.	70
Figura 61: Diagrama de Flujo para programación para sensor FH400.....	71
Figura 62: Circuito General con sensor de temperatura DS18B20	72
Figura 63: Instalación de sensor en tuberías para intercambiador de calor.....	73
Figura 64: Sensor de temperatura en termocupla.	73
Figura 65: Pantalla LCD como interfaz de usuario	74
Figura 66: Diagrama de flujo del funcionamiento de la interfaz local	75
Figura 67: Circuito para Interfaz de Usuario	76
Figura 68: Tuberías en intercambiador de calor	76
Figura 69: Ensamble de tuberías	77
Figura 70: Descripción de tareas encargadas a microcontroladores	78
Figura 71: Modulo microSD	78
Figura 72: Circuito modulo MicroSD conexión a microcontrolador.....	79
Figura 73: Archivos en la memoria Flash	80
Figura 74: Código para la interfaz de usuario en la nube	81
Figura 75: Diagrama de flujo Front-end para interfaz en la nube.....	82
Figura 76: Temperatura Sensor	83
Figura 77: Humedad Relativa Sensor	84
Figura 78: Velocidad del viento en sensor FH400 Fuente: propia.....	84
Figura 79: Instalación FH400	85
Figura 80: Instalación Parte Superior.	85
Figura 81: Instalación Parte Inferior	86
Figura 82: Ubicaciones de instalación	86
Figura 83: Ventilador ubicación anterior	87
Figura 84: Conexión de ventilador con túnel de viento.....	87
Figura 85: Ventilador ubicación actual	88
Figura 86: Grafica medición sensor DS18B20 Fuente: propia	88
Figura 87: Implementación y circuito	89
Figura 88: Vista previa de la interfaz desarrollada en HTML ,CSS Y JAVASCRIPT. Fuente: propia	90
Figura 89: Pagina Web Sinaloa	91

1 INTRODUCCIÓN

Actualmente se busca la formación de un colectivo de investigación e incidencia para generar soluciones aplicadas para atender, por un lado, la crisis de la disponibilidad del agua con calidad potable, así como de la transición energética en los diversos sectores sociales. El fomento del uso eficiente de la energía y la integración de tecnologías solares al sector productivo del país es importante pues permitirá ampliar el abanico de productos comercializables acondicionadores de aire, climatización de espacios y calor solar para procesos industriales en general y que generen un beneficio directo social y económico a las comunidades productivas.

1.1 DESCRIPCIÓN DE LA EMPRESA

El Centro de Materiales Avanzados (CIMAV), sede Durango, enfoca sus actividades al desarrollo de proyectos de investigación con criterios de excelencia en las áreas de *Energía y Medio Ambiente*, su objetivo es impulsar el desarrollo sustentable del país. Brinda soluciones tecnológicas y de innovación a los sectores productivo, académico y social mediante servicios, asesorías, consultas y generación de desarrollo tecnológico. Además, forma recursos humanos de alta especialidad para satisfacer la demanda académico-científica e industrial de México. Sus posgrados pertenecen al padrón del Programa Nacional de Posgrados de Calidad (PNPC):

- Maestría en Ciencia y Tecnología Ambiental
- Doctorado en Ciencia y Tecnología Ambiental (doble titulación)



Figura 1: Vista satelital del Centro de Investigación CIMAV (Latitud: 23,9929447, Longitud: -104.72651390902162).

Fuente: Google Maps

El área en la que se realizó el trabajo cursado durante la residencia, así como las pruebas y desarrollo del sistema, fue en el área de investigación de Energías Renovables.

1.1.1 MISIÓN

Realizar investigación científica, desarrollo tecnológico innovación y formación de recursos humanos con criterios de excelencia, en las áreas de Materiales, Energía y Medio Ambiente, para contribuir a impulsar el desarrollo sustentable del país.

1.1.2 VISIÓN

Ser un centro de clase mundial que eleve el nivel científico, tecnológico y de innovación del ámbito regional y nacional, en las áreas de Materiales, Energía y Medio Ambiente.

1.2 PROBLEMA A RESOLVER

Uno de los problemas que existen en México, que representa este proyecto, es de aproximadamente 12 millones de toneladas de alimentos cada año, casi 158 kg/persona por día y 56,000 toneladas en el país, según el Banco Mundial, equivalente al 34%. La producción está disponible para el consumo de personas. La capacidad de evaluar productos perdidos o desperdiciados en toda la cadena de suministro de alimentos contribuirá a lograr los objetivos de desarrollo sostenibles relacionados con los objetivos 12: Producción y consumo responsable y a lograr la meta Objetivo 12.1 de “reducir la mitad el desperdicio de alimentos per cápita mundial en la venta al por menor y a nivel de los consumidores y reducir las pérdidas de alimentos en las cadenas de producción y suministro, incluidas las pérdidas posteriores a la cosecha, hacia el 2030” (CIMAV Unidad Durango, 2023)

El proceso de secado es importante para la conservación de diversos alimentos agrícolas, de cría de animales y de peces, pero también es necesario en el procesamiento de diversos productos industriales, forestales y de otro tipo. (CIMAV Unidad Durango, 2023)

Actualmente, la comunidad de producción en la industria primaria deshidrata los productos mediante el secado manual, donde el calor ambiental retira lentamente la humedad de los alimentos y esta es eliminada gradualmente por el viento; sin embargo, este método tiene varias desventajas, a saber: el proceso en sí, como la contaminación del producto y la mala calidad sensorial. Por otro lado, existen tecnologías de secado de alimentos que utilizan combustibles fósiles como fuente de calor. Además de ser una tecnología relativamente costosa y de alto mantenimiento para las personas de estas comunidades marginadas, estos sistemas generan una huella de carbono en el proceso de fabricación. (CIMAV Unidad Durango, 2023)

El proceso de secado es importante para la conservación de diversos alimentos agrícolas, de cría de animales y de peces, pero también es necesario en el procesamiento de diversos productos industriales, forestales y de otro tipo (CIMAV Unidad Durango, 2023).

Actualmente, la comunidad de producción en la industria primaria deshidrata los productos mediante el secado manual, donde el calor ambiental retira lentamente la humedad de los alimentos y esta es eliminada gradualmente por el viento; sin embargo, este método tiene varias desventajas, a saber: el proceso en sí, como la contaminación del producto y la mala calidad sensorial. Por otro lado, existen tecnologías de secado de alimentos que utilizan combustibles fósiles como fuente de calor. Además de ser una tecnología relativamente costosa y de alto mantenimiento para las personas de estas comunidades marginadas, estos sistemas generan una huella de carbono en el proceso de fabricación (CIMAV Unidad Durango, 2023).

Por lo tanto, es necesario utilizar la energía solar térmica para el secado de productos agrícolas como una alternativa técnicamente factible, económicamente viable y de bajo impacto ambiental, ya que evita la generación de grandes cantidades de gases de efecto invernadero y contaminantes. Por su naturaleza, el flujo de energía radiante que utilizan las diferentes tecnologías solares varía con el tiempo; por otro lado, la climatología juega un papel crucial en el comportamiento térmico de las tecnologías utilizadas en el desarrollo de cualquier proyecto sostenible que pretenda garantizar su viabilidad (CIMAV Unidad Durango, 2023).

Al diseñar varios sistemas de calefacción solar con la ayuda del modelado y la simulación, es una ventaja que la optimización técnica y económica se realice antes de la compra e instalación del equipo, lo que garantiza un funcionamiento óptimo del sistema diseñado. (CIMAV Unidad Durango, 2023)

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

Diseñar un sistema de instrumentación para un emulador de aire sintético que permita generar aire con condiciones psicrométricas variables.

1.3.2 OBJETIVOS ESPECÍFICOS

- Diseñar un sistema de medición de temperatura y humedad relativa para distintos puntos en túnel de viento para el emulador de aire sintético.
- Diseñar un sistema que emule distintos niveles de flujo de aire dentro del emulador de aire sintético.
- Integrar el sistema de instrumentación y control de las variables temperatura y flujo de aire en un microcontrolador de 32 bits.
- Diseñar una interfaz gráfica en la nube para monitorización y control para el emulador de aire sintético.
- Realizar pruebas de funcionamiento del sistema de instrumentación.

1.4 JUSTIFICACIÓN

De acuerdo con la “Agencia Internacional de Energías Renovables” (IRENA), México se encuentra entre 15 países más importantes para la generación de energía solar, además del Banco Mundial que ubicó a México entre los siete países con mejores condiciones para el desarrollo de proyectos con este tipo de energía. De acuerdo con la Secretaría de Recurso Humanos y Medio Ambiente (SRNyMA, 2020), Durango presenta un gran potencial de energía solar, teniendo un promedio de 5.7 kWh/m²/día, siendo un estado con gran cantidad de recurso solar para desarrollar proyectos de deshidratación solar. (CIMAV Unidad Durango, 2023)

Un análisis elaborado por “MarketDataMéxico” en 2019 detalla que alrededor de 270 negocios en México están dentro de la actividad de deshidratación de frutas y

verduras, con ingresos anuales estimados en MXN \$2,000 millones. La mayor cantidad de ingresos anuales promedio se generan en Baja California, con MXN \$290 millones, en está Jalisco con MXN \$250 millones, y, en tercero, Nayarit, con MXN \$240 millones, siendo estos también estados en donde, tanto la radiación solar como las temperaturas del lugar son altas y con potencial para el deshidratado de productos agroindustriales. (CIMAV Unidad Durango, 2023)

El proyecto tiene el objetivo de diseñar, construir e instrumentar un emulador de aire con un amplio rango de condiciones psicrométricas preestablecidas, que sea capaz de generar las condiciones características de una zona geográfica o localidad en específico con el propósito de validar experimentalmente los procesos de secado de una gran variedad de productos. (CIMAV Unidad Durango, 2023)

Los sistemas de secado utilizan el aire atmosférico que los rodea para acondicionarlo y forzarlo a fluir sobre los productos a ser secados, con el objeto de promover la transferencia de la humedad del producto hacia la corriente de aire, y conseguir con ello la deshidratación del producto. Debido a que en diferentes regiones geográficas el aire atmosférico tiene diferentes temperaturas y humedades, el desempeño de las secadoras puede ser significativamente diferente en cada lugar geográfico en donde opere, y consecuentemente lograr o no su función de secado adecuadamente, afectando con ello la rentabilidad alcanzada. Por ello el emulador climático servirá para permitir validar experimentalmente diversos sistemas de secado destinados a diversas regiones geográficas y productos, operando desde la ciudad de Durango mediante la emulación de las condiciones climáticas del lugar de destino final. Los datos que se generen en las corridas experimentales servirán para validar la metodología de diseño por simulación, y establecer una gama de secadores solares adecuados a productos, regiones y capacidades, que pueden deshidratar de manera eficiente los materiales de interés sin comprometer la calidad y estabilidad de los productos, y de forma rentable. (CIMAV Unidad Durango, 2023)

1.5 ALCANCES Y LIMITACIONES DEL PROYECTO

1.5.1 ALCANCES

Este proyecto busca generar un sistema experimental para los siguientes objetivos principales:

Completar la etapa de construcción e instrumentación de un emulador de aire con un amplio rango de condiciones psicrométricas preestablecidas, que sea capaz de emular las condiciones climáticas características de una zona geográfica o localidad en específico con el propósito de validar experimentalmente, desde Durango, el desempeño de prototipos de secadoras de productos agrícolas, que van a operar finalmente en diferentes regiones del país, con otras condiciones de temperatura y humedad relativa.

El emulador climático servirá para permitir validar experimentalmente diversos sistemas de secado destinados a diversas regiones geográficas y productos, operando desde la ciudad de Durango mediante la emulación de las condiciones climáticas del lugar de destino final.

1.5.2 LIMITACIONES

La principal limitación para la realización del proyecto es el tiempo disponible durante las estadías de la residencia profesional que impiden concluir las etapas del proyecto definidas.

Otra limitación es el material faltante necesario para la realización de las etapas del proyecto que permiten que el proyecto pueda cumplir su cometido.

2 MARCO TEÓRICO

2.1 SECADO DE PRODUCTOS AGROINDUSTRIALES

El Dr. José Octavio menciona en su artículo donde es autor de “Secado en la industria de los alimentos” (Dr. Rodiles López, 2020). El secado es una de las técnicas de conservación de alimentos más antiguas e implica eliminar la humedad de los alimentos. Hace años, cuando los suministros escaseaban, los alimentos se dejaban secar naturalmente al sol, y este sistema todavía se usa en la actualidad.

El secado al sol, aunque barato, tiene varias desventajas: requiere mucho espacio dependiendo de las condiciones ambientales, hay polvo, insectos, roedores, etc. contaminación, y sucede lentamente.

La deshidratación industrial utiliza una variedad de diferentes tipos de equipos, pero todos ellos se basan en la evaporación de agua a altas temperaturas. En general, estos productos tienen un contenido de humedad final inferior al 5%. También incluye evaporación industrial, eliminación de agua de productos alimenticios líquidos. La deshidratación reduce el peso y el volumen de los alimentos, lo que afecta directamente los costos de procesamiento.

La deshidratación industrial depende de varios factores, los más importantes son:

- **Sistema de calefacción:**

La transferencia de calor puede ocurrir por convección, conducción o radiación, a veces en combinación.

- **Área de superficie:**

Es mejor dividir los alimentos en trozos pequeños o capas delgadas para aumentar la transferencia de calor y reducir el tiempo de secado, ya que el calor se transfiere más fácilmente a los objetos pequeños que a los grandes.

- **Presión:**

Si se utiliza vacío en el sistema, se reduce el punto de ebullición del agua. Por ejemplo, el agua hierva a 1000 grados centígrados a presión atmosférica

(760 mmHg), pero a 50 mmHg hierve a 380 grados centígrados. Cuanto menor sea la temperatura, menor será el daño, pero mayor será el costo operativo.

- **Tiempo:**

Depende directamente de la temperatura y la presión del sistema. El procesamiento prolongado puede destruir las propiedades organolépticas y nutricionales de los alimentos. El uso de altas temperaturas y cortos períodos de tiempo causa menos daño a los alimentos que la exposición prolongada a bajas temperaturas.

- **Tipo de alimentos:**

Cada alimento tiene su propio comportamiento durante el secado. Dos alimentos pueden tener el mismo contenido de humedad inicial, pero sus otros ingredientes son bastante diferentes. Por lo tanto, tienen diferentes propiedades cuando se secan.

2.2 PSICROMETRIA

La psicrometría se define como la medida de la humedad del aire. Técnicamente hablando, es la ciencia que se ocupa de las propiedades termodinámicas del aire húmedo y los efectos de la humedad sobre los materiales y el confort humano. (Juan, Danahé San, 2015)

IMPORTANCIA EN EL SECADO

En el proceso de secado, la medición psicométrica es importante porque permite conocer las condiciones del aire utilizado para secar el material, lo que a su vez afecta la eficiencia del proceso de secado. (S&P, 2020)

HUMEDAD RELATIVA

Es la relación entre la cantidad de vapor de agua presente en el aire y la cantidad máxima que puede contener a una temperatura y presión determinadas. La humedad relativa afecta la velocidad de secado del material. (José Antonio Marques Pereira, 1991)

PUNTO DE ROCÍO

Esta es la temperatura a la que el aire se satura de vapor de agua y comienza a condensarse. El punto de rocío es importante en el proceso de secado porque indica la temperatura mínima a la cual se puede enfriar el aire sin que se forme condensación que pueda afectar la calidad del material secado. (José Antonio Marques Pereira, 1991)

TEMPERATURA

Cuanto mayor sea la temperatura, más rápido se eliminará el agua. Sin embargo, también aumenta el potencial de degradación del valor nutricional y las propiedades organolépticas de los alimentos, y se necesita investigación para determinar la temperatura óptima de secado, el daño mínimo a los alimentos y la deshidratación máxima (Dr. Rodiles López, 2020).

2.3 TEORÍA SOBRE MICROCONTROLADORES

2.3.1.1 ARQUITECTURA DEL MICROCONTROLADOR ESP32 UTILIZADO

Se basan en un microprocesador Tensilica Xtensa LX6 de uno o dos núcleos con una frecuencia de operación de hasta 240 MHz (Carmenate José Guerra, 2021).

2.3.1.2 PERIFÉRICOS UTILIZADOS EN EL MICROCONTROLADOR

CONVERTIDOR DIGITAL A ANALÓGICO (DAC)

El ESP32 tiene dos canales DAC de 8 bits que convierten señales digitales en señales de salida de voltaje analógico. Los canales ESP32 DAC son útiles para generar señales analógicas, como las que se utilizan para la reproducción de audio. (Teja Ravi, 2021).

CONVERTIDOR ANALÓGICO A DIGITAL (ADC)

El ESP32 es un microcontrolador con dos convertidores de analógico a digital (ADC) de 12 bits que son útiles para medir señales analógicas como la temperatura, la presión y la intensidad de la luz. El ESP32 ADC tiene una resolución de 12 bits, lo que significa que puede detectar 4096 niveles discretos de señales analógicas (Teja Ravi, 2021).

Además, el rango de tensión que puede medir es de 0 a 3,3 voltios, lo que corresponde a una resolución de 0,0008 voltios (0,8 mV) por unidad. El ESP32 ADC se

puede configurar mediante programación para establecer la resolución y el rango del canal. Sin embargo, el ESP32 ADC tiene algunas limitaciones, como su comportamiento no lineal y el hecho de que el pin ADC2 no se puede usar cuando el Wi-Fi está activado (Teja Ravi, 2021).

RELOJ DE TIEMPO REAL (RTC)

El ESP32 cuenta con un RTC interno que permite mantener la fecha y hora en la placa incluso después de apagarla. El RTC del ESP32 puede ser sincronizado con un servidor NTP y utiliza un cristal integrado de 32.768 kHz como fuente de reloj (Asanza Victor, 2021).

ETHERNET EN EL MICROCONTROLADOR

Dispone de conexión Ethernet a través de un módulo externo que se puede conectar a la placa. El bloque Ethernet está conectado a través de pines SPI en la placa y puede configurarse mediante programación para establecer la dirección IP, la máscara de subred y la puerta de enlace. Además, el ESP32 tiene conectividad Wi-Fi y Bluetooth incorporada, lo que lo hace ideal para aplicaciones de Internet de las cosas (IoT). La conexión Wi-Fi ESP32 es compatible con el estándar 802.11 b/g/n en la banda de 2,4 GHz con una velocidad de hasta 150 Mbit/s. La conexión Bluetooth ESP32 es compatible con los estándares clásicos de Bluetooth y BLE (Circuitschool, 2022).

TRANSMISOR RECEPTOR ASINCRONO UNIVERSAL (UART)

UART significa Universal Asynchronous Receiver/Transmitter, que define un protocolo o conjunto de reglas para intercambiar datos en serie entre dos dispositivos. La comunicación UART puede ser simple (los datos se envían en una sola dirección), semidúplex (cada lado transmite, pero solo uno a la vez) o dúplex completo (ambos lados pueden transmitir al mismo tiempo) (Schwarz Rohde, 2023).

El ESP32 tiene varios puertos UART que permiten la comunicación serial con otros dispositivos. El puerto UART0 está reservado para el terminal, el puerto UART1 comparte los mismos pines con la memoria flash y el puerto UART2 se recomienda principalmente para la comunicación en serie con otros dispositivos.

El ESP32 tiene una biblioteca UART que le permite configurar el puerto UART y establecer la velocidad en baudios, la cantidad de bits de datos, la paridad y el bit de parada (Espressif Systems (Shanghai) Co., Ltd., 2016).

PROTOCOLO CIRCUITO INTER-INTEGRADO (I2C)

El bus I2C usa dos cables designados como línea de datos en serie, o SDA, y línea de reloj en serie, o SCL. SDA y SCL son líneas bidireccionales abiertas de drenaje/colector y están conectadas al bus de energía positiva a través de una fuente de corriente o una resistencia de arrastre. Se pueden conectar múltiples dispositivos al bus, con el número máximo limitado por la capacidad del bus. Un dispositivo maestro controla el bus y cada dispositivo del bus tiene una dirección única. El dispositivo maestro puede transmitir y recibir datos a través del bus. I2C admite la operación multimaestro con detección de colisiones y arbitraje para evitar que dos o más dispositivos maestros inicien transferencias de datos al mismo tiempo (Pini Art, 2020).

El ESP32 cuenta con dos puertos I2C que permiten la comunicación con dispositivos externos a través del protocolo I2C. El ESP32 cuenta con una librería I2C que permite configurar los puertos I2C y establecer la velocidad de transmisión y la dirección del dispositivo (RandomNerd, 2023).

BUS DE INTERFAZ DE PERIFÉRICOS (SPI)

Es un bus serie síncrono, es decir. utiliza una línea de reloj para sincronizar las transferencias de datos entre dispositivos. Estas transferencias ocurren en dos líneas, una para transferencias de maestro a maestro y otra para transferencias de esclavo a maestro. Para iniciar o detener una transferencia, el bus incluye una línea de selección de chip que le permite identificar el dispositivo con el que desea intercambiar información (Espressif Systems (Shanghai) Co., Ltd, 2016).

Es un bus de cuatro hilos. Es un protocolo maestro-esclavo que permite controlar múltiples periféricos a través de un solo bus. Se considera un sistema host único, es decir. un dispositivo es responsable de generar el reloj para la transferencia sincrónica e indicar el inicio y el final de la transferencia a los dispositivos aguas abajo

a través de las líneas de selección de chip. Esto permite que varios sistemas esclavos se conecten al mismo bus para que el maestro pueda intercambiar información con todos los sistemas esclavos usando el mismo reloj y líneas de datos. El esclavo configura su salida de datos a una alta impedancia para permitir que otros esclavos accedan al bus sin distorsionar el mensaje (Espressif Systems (Shanghai) Co., Ltd, 2016).

La comunicación es dúplex completo, lo que significa que se permite enviar y recibir información al mismo tiempo, lo que permite que el dispositivo actúe como emisor y receptor de información (Espressif Systems (Shanghai) Co., Ltd, 2016).

ESP32 tiene cuatro periféricos SPI, a saber, SPI0, SPI1, HSPI y VSPI. Los dispositivos SPI0 y SPI1 están dedicados a la memoria caché flash utilizada por el ESP32 para mapear la memoria SPI interna, por lo que no se recomienda usarlos para otras tareas. Los dispositivos HSPI y VSPI se pueden utilizar para comunicarse con otros dispositivos mediante el protocolo SPI. Cada bus HSPI y VSPI puede controlar hasta tres periféricos SPI. El ESP32 tiene una biblioteca SPI que le permite configurar el dispositivo SPI y establecer la velocidad en baudios, el modo de transferencia, el bit de comando y la dirección del dispositivo (Espressif Systems (Shanghai) Co., Ltd, 2016).

2.3.2 PROTOCOLOS DE COMUNICACIÓN PARA EL MICROCONTROLADOR

RS485

Es un estándar de comunicación ampliamente utilizado en aplicaciones de control y adquisición de datos. Una de sus principales ventajas es que permite incluir múltiples dispositivos RS485 en un mismo bus, permitiendo interconectar múltiples nodos. Su función principal es transmitir señales a través de dos cables. Uno de los cables lleva la señal original y el otro lleva su copia invertida. Este método de transmisión es altamente inmune a la interferencia de modo común. Los cables de par trenzado utilizados como líneas de transmisión pueden ser blindados o no blindados.

Una red de comunicación creada utilizando la interfaz RS-485 consta de transceptores conectados por medio de un par trenzado (dos pares trenzados). La función principal de la interfaz RS-485 es la transmisión de datos diferencial (equilibrada). Esto significa que la señal se envía a través de dos cables. De esta forma, un hilo del par lleva la señal original y el otro la copia inversa (Wies Olga, 2021).

MODBUS RTU

Modbus es un protocolo de comunicaciones basado en una arquitectura maestro/esclavo o cliente/servidor desarrollado por Modicon en 1979 para su línea de controladores lógicos programables (PLC). Es el protocolo más utilizado para conectar dispositivos electrónicos industriales. El protocolo Modbus le permite controlar una red de dispositivos, por ejemplo, medidores de temperatura y humedad pueden transmitir los resultados a una computadora. Modbus también se usa para conectar computadoras de monitoreo a unidades remotas (RTU) en sistemas de adquisición de datos de control de supervisión (SCADA). Existen versiones del protocolo Modbus (Modbus/TCP) para redes serie y Ethernet (Logicbus, 2023).

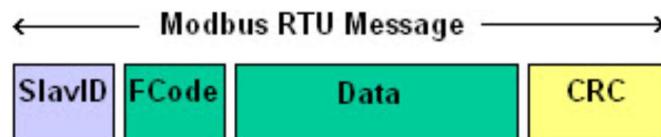


Figura 2: Paquete trama de datos

Fuente: <https://www.logicbus.com.mx/Modbus.php>

2.3.3 PLATAFORMAS DE DESARROLLO

PLATFORMIO EN VS CODE

Visual Studio Code (VS Code) es un editor de código fuente desarrollado por Microsoft. Es un software multiplataforma gratuito disponible para Windows, GNU/Linux y macOS. VS-Code se integra bien con Git, admite la depuración de código y tiene muchas extensiones (Flores Frankier, 2022).

Un entorno de desarrollo integrado (IDE) para la programación de sistemas integrados es PlatformIO. Ofrece un entorno unificado para crear aplicaciones para muchas plataformas de hardware, como Arduino, ESP32, Raspberry Pi, etc. PlatformIO

proporciona un conjunto de herramientas y bibliotecas para el desarrollo de proyectos integrados que se basan en el lenguaje de programación C/C. Ofrece la capacidad de crear y depurar código, administrar bibliotecas, compilar y cargar software en dispositivos de hardware y tiene una interfaz gráfica de usuario fácil de usar (Córdova Diego Hinojosa, 2021).

LENGUAJE DE PROGRAMACIÓN C++

El lenguaje de programación es una herramienta que se puede utilizar para dar instrucciones a una computadora y ejecutar algoritmos en los campos matemático, estadístico, contable, técnico, gráfico, etc. para resolver problemas de carácter. La capacidad que proporciona un lenguaje para implementar algoritmos y sistemas es una función su gramática; que incluye tres aspectos principales: vocabulario, sintaxis y semántica (Menchaca García, 2020).

ALTIUM DESIGNER

Altium Designer es un conjunto de programas para el diseño electrónico en todas sus fases y para todas las disciplinas, ya sean esquemas, simulación, diseño de circuitos impresos, implementación de FPGA, o desarrollo de código para microprocesadores (Qué es Altium Designer, 2008).

MATLAB

MATLAB es un gran programa informático técnico y científico. Para algunas operaciones, es muy rápido si la función se puede ejecutar en código nativo con el tamaño más adecuado para aprovechar sus capacidades de vectorización. En otras aplicaciones, es mucho más lento que el código equivalente escrito en C/C++ o Fortran (García J. - Rodríguez, 2020).

2.4 TEORÍA SOBRE PAGINAS WEB

2.4.1 SERVIDOR WEB

Un servidor web es un software que forma parte de un servidor cuyo trabajo principal es devolver información (páginas) cuando un usuario lo solicita. En otras

palabras, el software permite que los usuarios que deseen ver páginas web en un navegador lo hagan (Webempresa, 2023).

2.4.2 LENGUAJES DE PROGRAMACION PARA WEB

LENGUAJE DE MARCADO DE HIPERTEXTO: HTML

HTML significa Lenguaje de marcado de hipertexto y, en resumen, es el lenguaje que utilizan los navegadores para presentar información en la World Wide Web. Por lo tanto, los navegadores de Internet son los encargados de interpretar los códigos HTML. A diferencia de los lenguajes de programación, todos los errores en el código HTML no recibes un mensaje, simplemente no funciona. Es un lenguaje muy simple porque usa tokens o etiquetas que consisten en texto. Está encerrado por un par de corchetes angulares (< y >) (Pére Rodríguez, 2018).

HOJAS DE ESTILO:CSS

Las llamadas Hojas de Estilo en Cascada, CSS o Cascading Style Sheets abrieron nuevas posibilidades para los diseñadores de páginas web. La idea detrás del desarrollo de CSS es separar la estructura de un documento de su presentación o apariencia, como una combinación de XHTML y CSS (Orós Cabello, 2014).

JAVASCRIPT

JavaScript fue diseñado por Netscape para integrarse con HTML, lo que permite la creación de páginas interactivas sin necesidad de scripts CGI o Java (Orós Cabello, 2014).

2.4.3 TECNOLOGIAS PARA PROTOCOLOS WEB

PROTOCOLO DE TRANSFERENCIA DE HIPER TEXTOS (HTTP)

Es el Protocolo de transferencia de información de la World Wide Web, un código diseñado para permitir que la computadora solicitante y la computadora que contiene la información solicitada "hablen" el mismo idioma mientras transmiten información a través de una red. HTTP establece estándares para la sintaxis y la semántica informática (forma y significado) que se utilizan para establecer la comunicación entre los distintos elementos (servidores, clientes, proxies) que componen la arquitectura de la red (Editorial Etecé, 2023).

ASÍNCRONO JAVASCRIPT Y XML (AJAX)

Una técnica diseñada para evitar el retraso inherente a las solicitudes y respuestas del servidor mediante la transferencia de datos en segundo plano mediante un protocolo diseñado para transferir pequeños paquetes de datos rápidamente. Con Ajax, puedes enviar una solicitud al servidor y recibir una respuesta en segundo plano (sin recargar toda la página web), y usar esos datos para cambiar el contenido de la página usando JavaScript, creando una dinámica y rapidez (Krall César, 2023).

3 PROCEDIMIENTO, DESCRIPCION DE ACTIVIDADES REALIZADAS

A continuación, se muestra los equipos utilizados durante la estadía de la residencia profesional para después pasar al material y a la metodología realizada.

3.1 EQUIPOS

3.1.1 TUNEL DE VIENTO

Las siguientes imágenes muestran el diseño del túnel y sus medidas donde se realizó la instrumentación de sensores y actuadores.

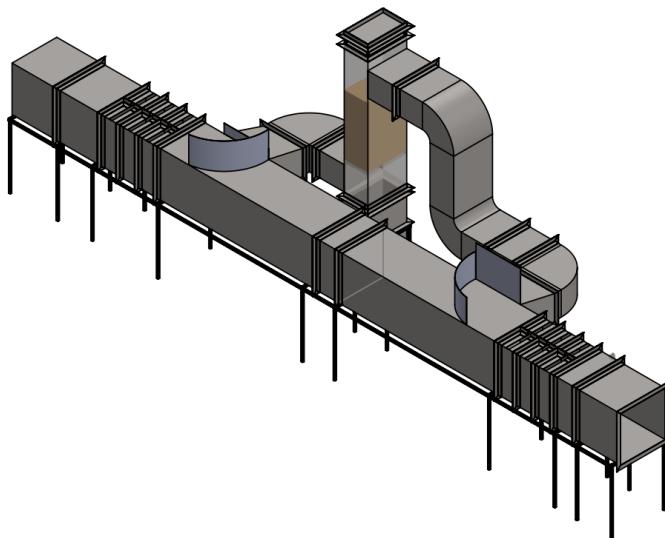
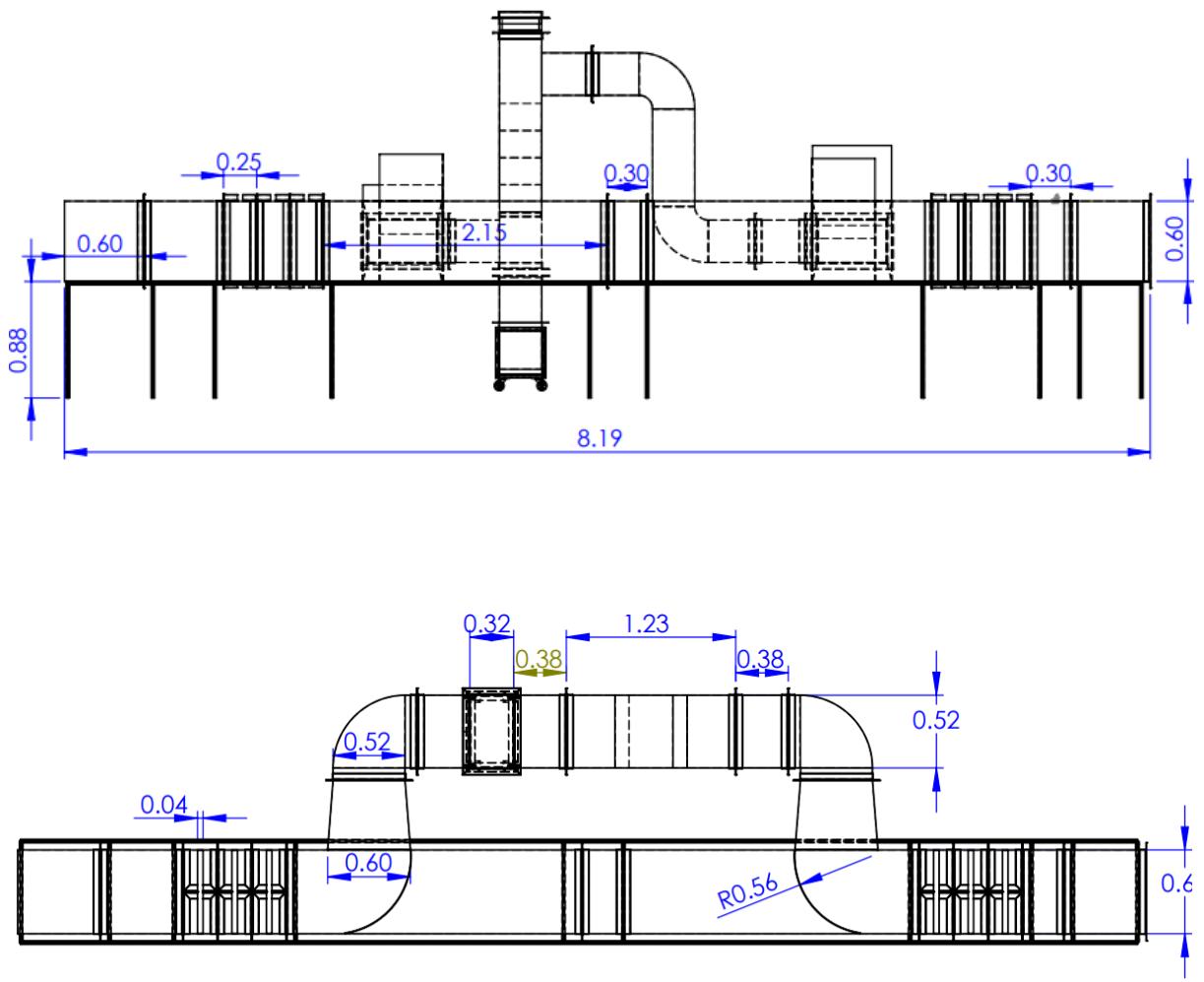


Figura 3: Túnel de viento en 3D

Fuente: Reporte Técnico Desalinización solar de agua para consumo humano a través del proceso de humidificación deshumidificación. CIMAV Durango.



Vista superior

Figura 4: Medidas Túnel de Viento

Fuente: Reporte Técnico Desalinización solar de agua para consumo humano a través del proceso de humidificación – deshumidificación. CIMAV Durango.

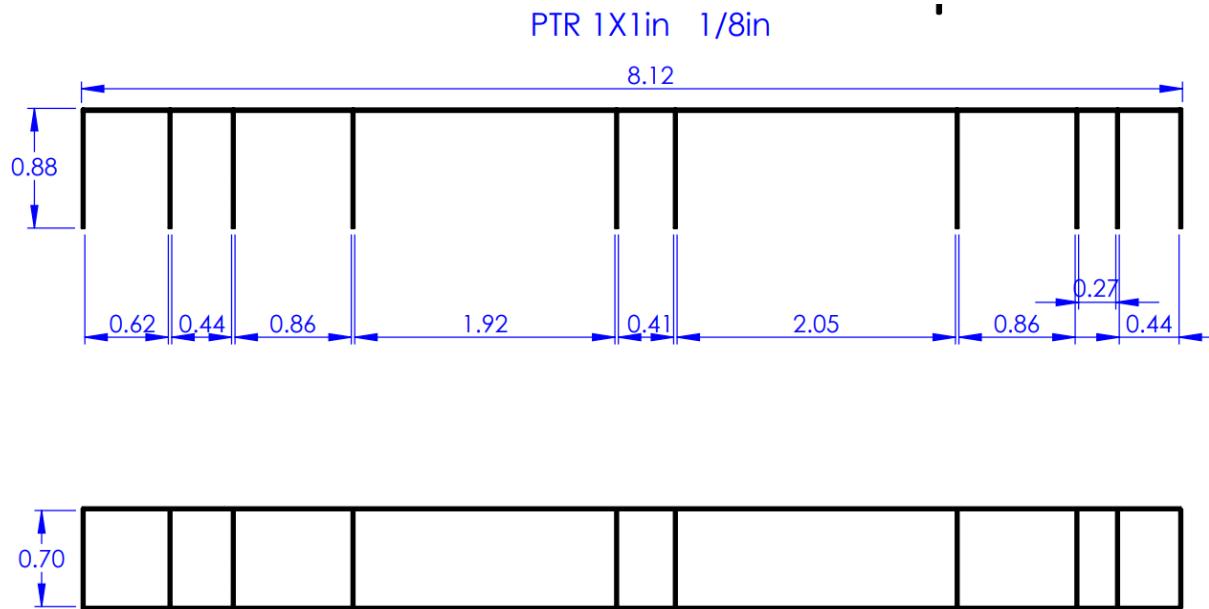


Figura 5: Medidas patas del túnel del viento

Fuente: Reporte Técnico Desalinización solar de agua para consumo humano a través del proceso de humidificación – deshumidificación. CIMAV Durango.

3.1.2 SISTEMA DE CALENTAMIENTO AGUA/AIRE MULTIPROPÓSITO(SCAAM)

Sistema de calentamiento agua/aire multipropósito (SCAAM) tiene como finalidad la caracterización térmica de dispositivos, así como de procesos térmicos. Este consta de un conjunto de dispositivos que pueden ser interconectados entre sí y que tienen el propósito de formar una plataforma de pruebas versátil, para la caracterización y validación de equipos termo-hidráulicos tales como: colectores solares, intercambiadores de calor, prototipos de captación solar, sistemas de almacenamiento térmico, entre otros. Así mismo tiene la capacidad de soportar pruebas experimentales instrumentadas para prototipos de procesos industriales que requieran una caracterización energética, para procesos de calentamiento como: deshidratación, pasteurización, desalinización, esterilización entre otros.

El SCAAM permite movilidad de sus componentes, así como diferentes configuraciones hidráulicas mediante la versatilidad de su diseño especial para la variación del número y tipo de colectores solares. Es posible realizar múltiples arreglos

como, por ejemplo: conexión serie/paralelo, volumen variable de almacenamiento, flujos máicos de agua o aire, fuente y capacidad de energía auxiliar, entre otros (Durango, 2023).

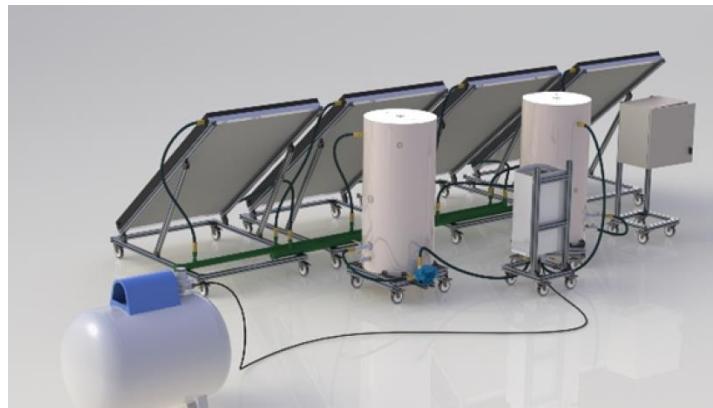


Figura 6: Render SCAAM

Fuente: CIMAV Durango

ELEMENTOS QUE CONFORMAN EL SCAAM:

COLECTORES SOLARES DE PLACA PLANA

Un colector solar térmico es una parte integral de un colector solar que recoge la radiación solar y la convierte en calor. Este tipo de panel solar también se conoce como colector térmico o panel solar térmico (Planas Oriol, 2015).

TERMOTANQUES

Un tanque de agua a temperatura constante es un calentador de agua que calienta una cierta cantidad de agua transfiriendo calor del agua al tanque de agua para que los usuarios puedan usar agua caliente (R. José Luis, 2023).

CALDERA AUXILIAR BOSCH DE 52 KW

Una caldera de condensación es una caldera de calefacción a gas de alto rendimiento basada en el aprovechamiento del calor latente de condensación en los gases de combustión (Arnabat Idoia, 2020)

ADQUISIDOR DE DATOS HOBO

HOBO RX3000 brinda acceso instantáneo a datos ambientales específicos del sitio a través de Internet en cualquier momento y en cualquier lugar. Esta estación de trabajo configurable combina la versatilidad de los sistemas más costosos con la calidad del sensor, una pantalla LCD integrada y una conveniente funcionalidad plug-and-play (HOBO, 2023).

PANEL FOTOVOLTAICO

Los módulos fotoeléctricos, también conocidos como paneles solares o paneles solares, son equipos que perciben la energía solar para comenzar y transformarse en energía sostenible. Por lo general, el recubrimiento de materiales semiconductores suele ser silicio. Los elementos básicos de cada panel solar son sensibles a la luz y generan electricidad con la luz solar. Esto se debe a fenómenos físicos llamados efectos fotoeléctricos (Enel Spa, 2023).

3.1.3 INSTRUMENTOS DE MEDICION UTILIZADOS

- Anemómetro
- Termómetro
- Osciloscopio
- Multímetro
- Analizador Lógico

3.1.4 SENsoRES UTILIZADOS

SENSOR ANEMOMETRO FH400

DegreeC FH400 es un sensor de velocidad del viento, temperatura y humedad versátil y resistente de alto rendimiento con salida de comunicación analógica. FH400 es adecuado para aplicaciones exigentes. La serie FH400 se puede configurar de

forma personalizada con diferentes rangos de velocidad, longitudes mecánicas y métodos de comunicación de salida. La salida de voltaje se puede configurar en 0-5V o 0-10V (DEGREE CONTROLS, 2022).



Figura 7: Sensor FH400

Fuente: FH400 Air Velocity, Temperature, and Humidity Sensor User Guide

SENSOR DE TEMPERATURA Y HUMEDAD SHT40

Para la medición de humedad y temperatura del aire se empleó los circuitos integrados SHT40 del fabricante Sensirion. A continuación, se muestra una imagen ilustrativa del sensor.

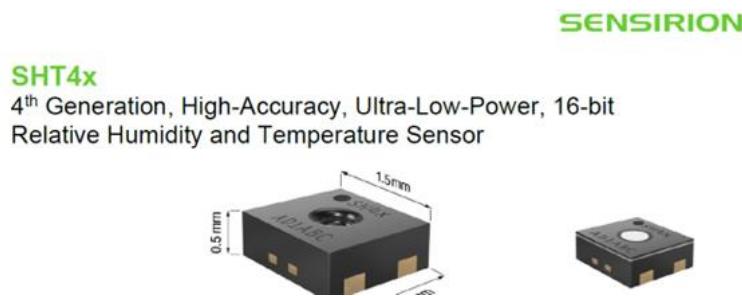


Figura 8: Muestra Sensor IC Temperatura y Humedad Relativa

Fuente: <https://sensirion.com/products/catalog/SHT40>

SHT4x es una plataforma de sensores digitales para medir la temperatura y la humedad relativa en diferentes clases de precisión. Su interfaz I2C proporciona una variedad de direcciones preestablecidas I2C mientras mantiene un consumo de energía ultra bajo. Los calentadores internos están disponibles con potencia ajustable para tres

niveles de calentamiento, lo que permite que el sensor funcione en condiciones adversas. El paquete de cuatro pines sin plomo de dos planos es adecuado para la tecnología de montaje en superficie (SMT) y viene con una película de PTFE patentada o una tapa protectora extraíble opcional. Los certificados de calibración específicos del sensor se proporcionan de acuerdo con ISO17025 y pueden identificarse mediante un número de serie único (Sensirion, 2023).

SENSOR DE TEMPERATURA DS18B20 SUMERGIBLE

El DS18B20 es un sensor de temperatura fabricado por Maxim Integrated. Proporciona salida a través de un bus de comunicación digital. Tiene un amplio rango de medición de -55 °C a 125 °C y una precisión superior a ±0,5 °C de -10 °C a 85 °C. El DS18B20 utiliza el bus de comunicaciones patentado de Maxim Integrated llamado 1-Wire. La principal ventaja de un bus de 1 hilo es que requiere un solo conductor (sin incluir el conductor de tierra) para la comunicación. Estos dispositivos se pueden alimentar directamente desde las líneas de datos o mediante una línea adicional de 3,0 a 5,5 V (Llamas Luis, 2016).



Figura 9: Sensor DS18B20

Fuente: <https://www.luisllamas.es/temperatura-liquidos-arduino-ds18b20/>

3.1.5 ACTUADORES UTILIZADOS

VARIADOR DE FRECUENCIA

Los variadores o convertidores de frecuencia son sistemas que se encuentran entre la fuente de alimentación eléctrica y los motores eléctricos. Sirven para regular la velocidad de giro de los motores de corriente alterna (AC). Regulando la frecuencia de la electricidad que recibe el motor, el variador de frecuencia consigue ofrecer a este motor la electricidad demandada, evitando así la pérdida de energía, o lo que es lo

mismo, optimizando el consumo. El Variador trifásico que utilice para el desarrollo del proyecto es el TECO L510.

MOTOR TRIFASICO

El motor trifásico que se utilizo tiene las siguientes especificaciones:

- 1750 Rpm.
- 60 Hz.
- Fp: 66.8
- Eficiencia nominal:77.
- Corriente eléctrica de trabajo 2,70 Amperios.
- Cp:0,75
- Tipo de conexión estrella.
- Kw: 0,56

VENTILADOR

- Motor: $\frac{3}{4}$ hp
- Flujo de aire: 1500 cfm
- Presión estática: 1.5 in. Hg
- Impulsor: álabes rectos atrasados
- Tamaños de conducto disponibles: 4", 5", 6", 8" y 10"
- Voltaje: 120V
- Peso: 25 libras

3.2 MATERIALES UTILIZADOS

- Tuberías galvanizadas
- Acido férrico
- Cable UTP trenzado
- Cautín
- Fuente de voltaje
- Estaño para soldar
- Pinzas de corte
- Cinta métrica
- Papel térmico
- Cinta aislante
- Placas fenólicas
- Plancha
- Cuter

3.3 METODOLOGIA

3.3.1 DISEÑO SENSOR DE TEMPERATURA Y HUMEDAD SHT40

Se construyo el sistema de medición de temperatura y humedad relativa utilizando el diseño propio que utilizaban los investigadores en CIMAV. Se utilizan el sensor SHT40 para censar la temperatura y humedad relativa ambiental. Los datos que se adquieren se envían al microcontrolador de 8 bits Attiny202. El microcontrolador se encarga de hacer dos cosas: cambiar el ID del sensor SHT40 para poder utilizarlo posteriormente y enviar los datos censados a través del puerto serial UART del microcontrolador Attiny202.

Los datos enviados a través puerto serial son recibidos por el circuito integrado MAX485, quien es el encargado de convertir la señal recibida en RS485 para poder mandarla a la distancia necesaria para luego lograr ser recibida por el microcontrolador ESP32 quien a su vez cuenta con otro MAX485 para recibir y tratar la señal enviada por el MAX485 del sensor de temperatura, ver Figura (12).

El circuito integrado MAX485 del lado del microcontrolador ESP32 convierte la señal RS485 a la señal tipo UART que se encuentra en los pines GPIO16 y GPIO17 del microcontrolador. Se utilizo el UART2 del microcontrolador ESP32 para enviar información y el UART0 se utilizó para realizar depuración en el código con el objetivo de encontrar y solucionar error en el código escrito. De haber utilizado un solo UART los datos enviados y recibidos por el puerto serial habrían ocasionado problemas de recepción.

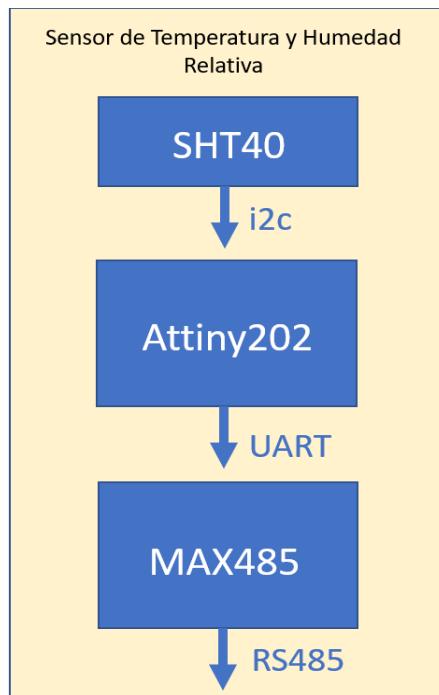


Figura 10: Diagrama del sensor de temperatura y humedad

Fuente propia

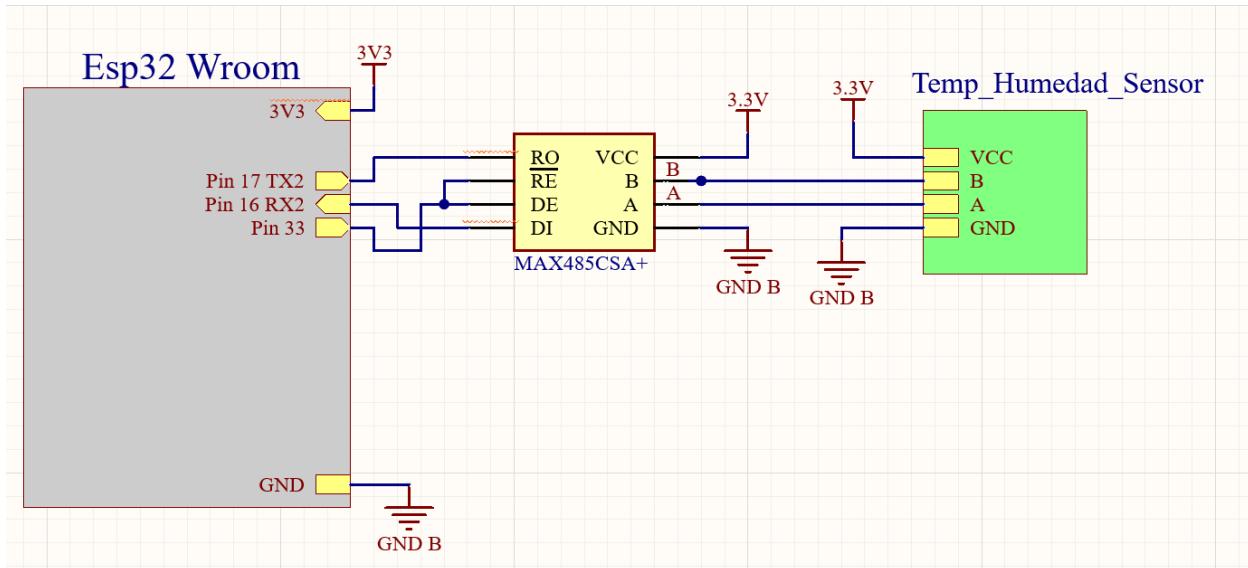


Figura 11: Esquemático general para el microcontrolador Esp32 y el sensor de temperatura.

Fuente Propria.

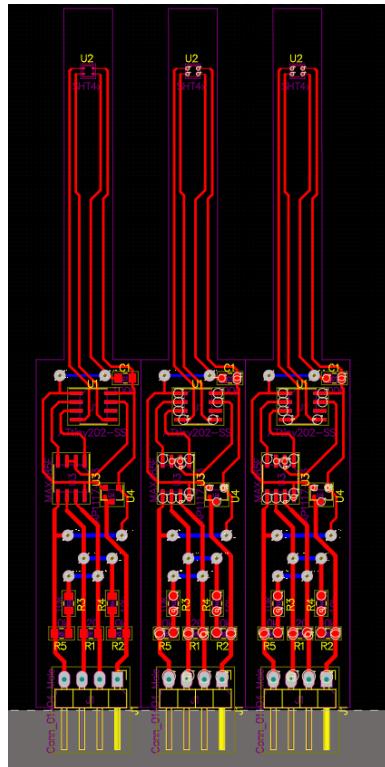


Figura 12: Muestra de Ruteo de Sensor de Temperatura y Humedad Relativa

Fuente: propia

3.3.2 CONSTRUCCIÓN DE LA PLACA DE CIRCUITO IMPRESO DEL MÓDULO DE *TEMPERATURA Y HUMEDAD RELATIVA*

Se utilizó el método de planchado para circuitos eléctricos. Se utilizó la capa superior del circuito para ser impreso. La capa inferior no se utilizó por razones de la tabla fenólica no contaba con cobre recubierto en ambos lados. Los materiales para realizar el método de planchado:

- Tabla Fenólica
- Cloruro Férrico
- Agua
- Plancha
- Hoja de transferencia térmica
- Impresora Láser o fotocopiadora
- Recipiente

Se ajustó la escala del diseño para montar los componentes en nuestro PCB, y que las medidas coincidan. Se especifica que debe ser a blanco y negro la impresión seleccionando la opción “Mono”, se continua con la opción Carta (Letter) y la escala debe ser colocada en 106%. Se continua en la siguiente pestaña llamada “Pages” donde se debe seleccionar la única capa de interés que es la capa superior llamada “F.Cu”.

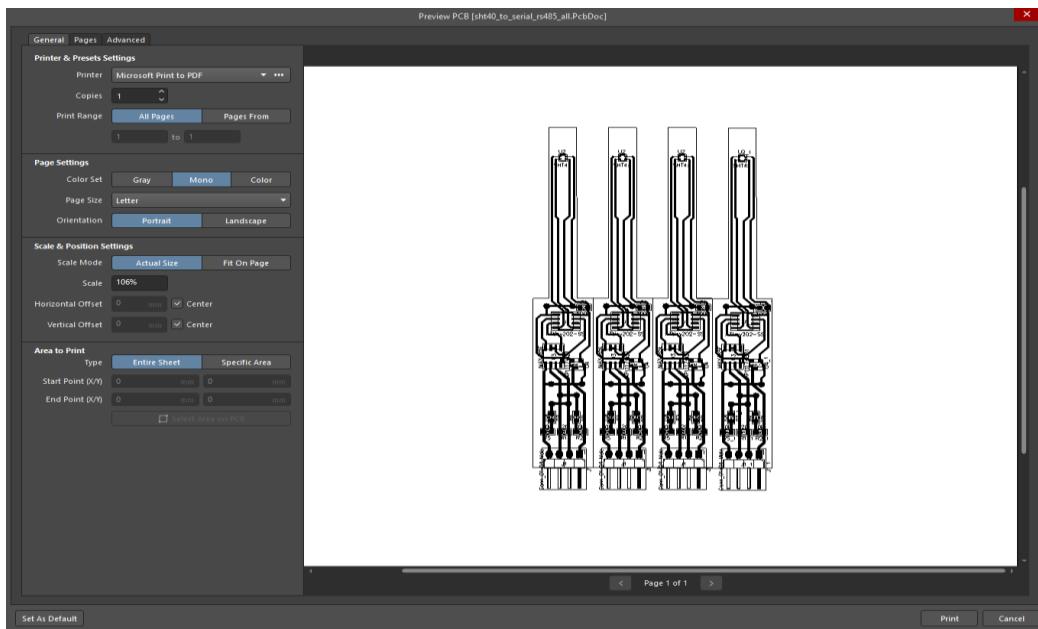


Figura 13: Configuración de impresión de circuito en Altium Designer

Fuente Propia.

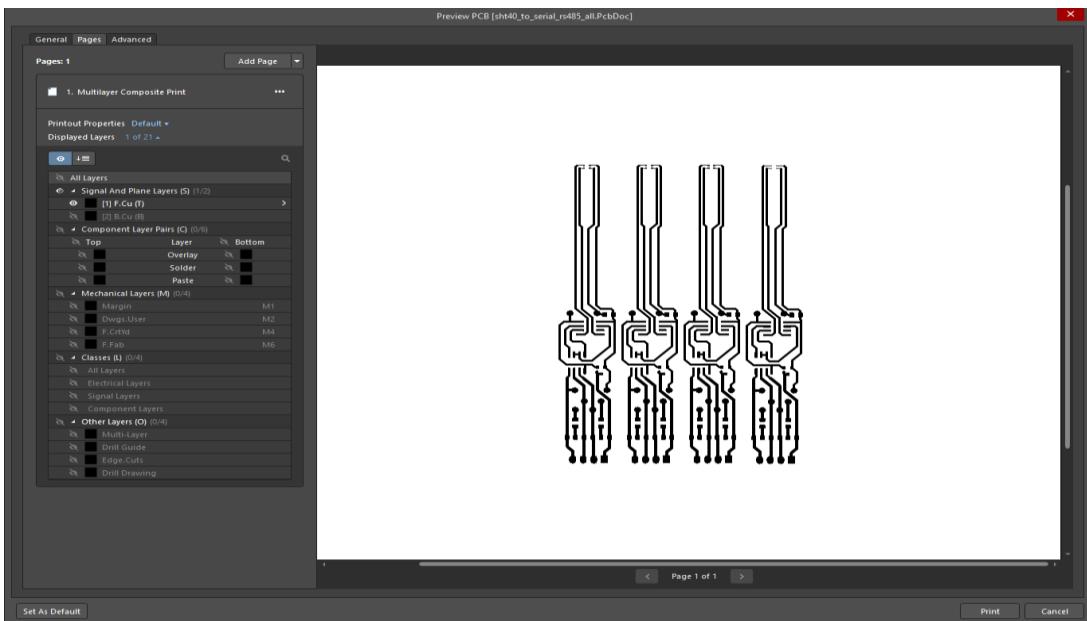


Figura 14: Selección de capas de interés

Fuente: Propia

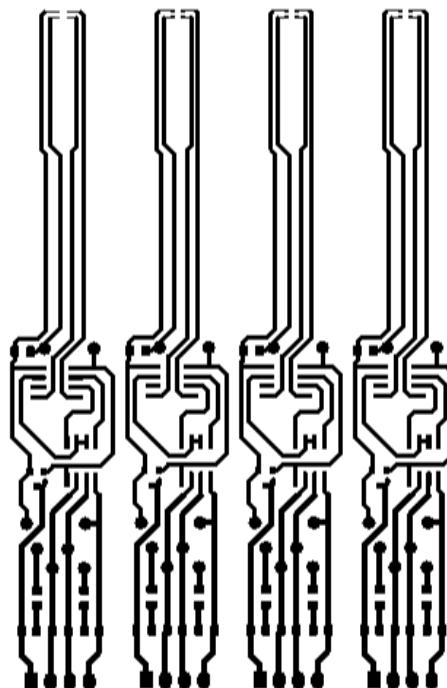


Figura 15: Resultado Final para Impresión

Fuente propia

El siguiente paso fue la transferencia a la tabla fenólica. Primero se limpió la tabla fenólica con agua y un estropajo para dejarla lo suficientemente limpia para utilizarla. Se coloco la hoja de transferencia donde del lado de la impresión está mirando a cara con cobre de la tabla fenólica, se sujetó con cinta adhesiva para sujetarla y se comenzó a pasar la plancha caliente para lograr la transferencia de la tinta a la tabla fenólica. Se debe pasar la plancha alrededor de la tabla durante diez minutos aproximadamente.

Al terminar se sometió la tabla fenólica a un chorro de agua durante unos instantes para lograr un choque térmico con el motivo de lograr una mejor adherencia de la tinta a la tabla fenólica. El siguiente paso fue retirar la hoja de transferencia terminada de la tabla fenólica con delicadeza. Se comprueba que todo el circuito fue transferido correctamente, en caso contrario se puede solucionar dibujando las pistas que no se transfirieron con un plumón negro permanente sobre la tabla fenólica.

Se sumergió la tabla fenólica en ácido férreo con el circuito adherido a ella en el recipiente adecuado para contenerlo sin derramar líquido. Se debe de mover el recipiente de lado a lado para que el ácido haga su trabajo de retirar el cobre que no está cubierto por las pistas del circuito.

Al no tener cobre visible en la tabla fenólica es el indicativo que está listo y se puede retirar la tabla del recipiente. Se limpia la tabla fenólica con agua y un estropajo para quitar la tinta que recubre las pistas de cobre. Se cortó la tabla en el tamaño necesario previsto con una seguita y se utilizó una pulidora para lograr la forma final para el sensor.



Figura 16: Resultado del método de planchado

Fuente propia



Figura 17: Resultado Final Sensores de Temperatura y Humedad relativa

Fuente: Propia



Figura 18: Vista por detrás sensor de temperatura y humedad relativa

Fuente: Propia

PROGRAMACIÓN DEL CIRCUITO INTEGRADO SHT40

Se programo el microcontrolador Attiny202 en lenguaje C donde se asignó un nuevo ID o dirección para el sensor y fue programado por uno de los investigadores quien autor del código integración del circuito integrado SHT40 con el Attiny202.

Se construyo siete sensores de temperatura y humedad donde para el primer sensor se asignó la dirección en hexadecimal de 0x31 y para los siguientes sensores se les asigno la dirección de 0x32, 0x33, 0x33, 0x34, 0x35, 0x36, 0x37.

CABLEADO PARA SENsoRES DE TEMPERATURA Y HUMEDAD

El cableado utilizado para lograr la conexión de sensores al Esp32 es el cable UTP de tamaño 22 AWG ya que es el material que ya se tenía comprado, no es el recomendado para aplicaciones de RS485. Se utilizo alrededor de 20 metros de cable.

INSTALACIÓN Y UBICACIÓN DE SENsoRES

Se colocaron cinco sensores en los siguientes puntos mostrados en la imagen y utilizando el concepto de implementación en Daisy Chain para RS485. Los otros sensores faltantes se plateo usarlos de repuesto de llegar a ser necesarios.

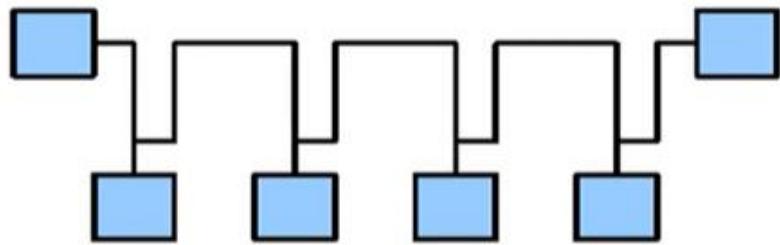


Figura 19: Instalación de sensores en forma Daisy Chain

Fuente: https://www.youtube.com/watch?v=3k7cEBm0CrA&ab_channel=TexasInstruments

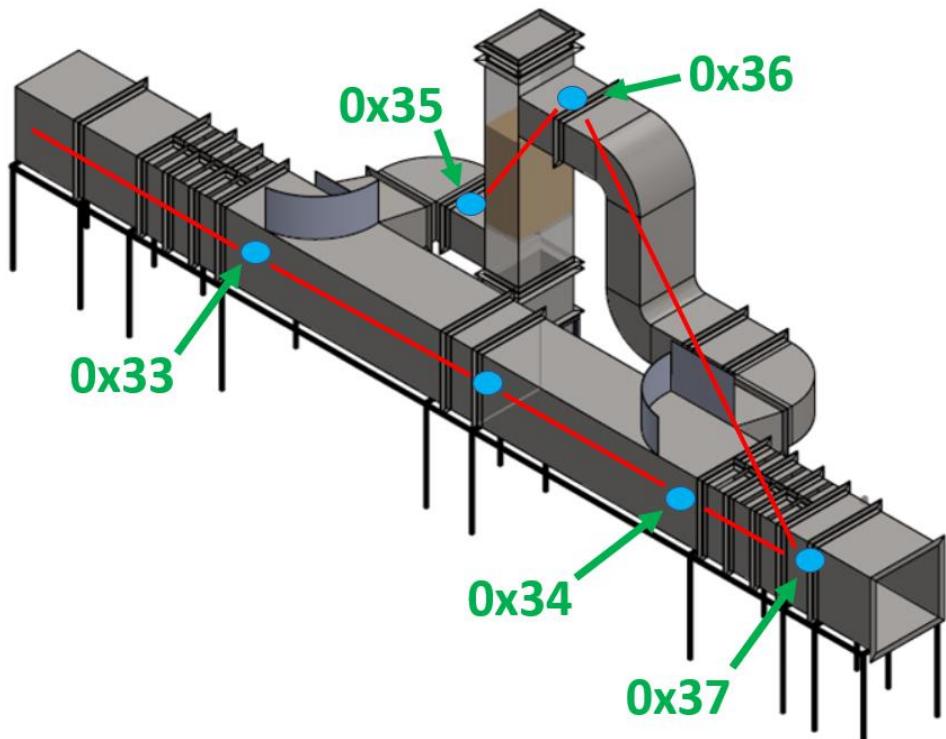


Figura 20: Ubicación de Sensores en el túnel de viento con su dirección

Fuente Propia



Figura 21: Vista previa de instalación de cable para sobre el túnel de viento para sensores

Fuente propia

PROGRAMACIÓN DEL SENSOR DE TEMPERATURA Y HUMEDAD RELATIVA

Para la programación del sensor de temperatura y humedad se desarrolló una lógica de programación propia para adquirir los datos de censado enviados por los sensores. Para lograrlo es necesario entender que es lo que envía el sensor (trama de datos) y para ello se debe de leer la hoja técnica de datos que proporciona el fabricante del (IC) circuito integrado SHT40.

Desde la Esp32 se debe enviar los siguientes bytes expresados en hexadecimal “0x66,0x66,0x66” para decirles a los sensores que se desocupen de cualquier cosa que estén haciendo y se continúa enviado “0x51, dirección del sensor,0xF1” para decirle al sensor correspondiente que envíen la medición de la temperatura y humedad. Se debe tener en cuenta que no se pueden pedir la medición a varios sensores de forma simultánea ya que chocaría con los demás datos que viajan en el mismo bus de datos, hay que tomar en cuenta que el bus de half-duplex, esto quiere decir que no se puede enviar y recibir simultáneamente. Por esta razón de debe enviar y recibir datos de los sensores secuencialmente. Dicho de otra manera; se solicita la medición a un único sensor y se espera un tiempo para su recepción y se continua secuencialmente con los demás sensores faltantes. La trama de datos que se recibe se muestra en la figura 22.

```

ffffU2 Dirección del sensor: 0x32
Trama Recibida: 6E FE 24 48 1E 63
La temperatura es: 30.87°C
La humedad es: 29.21 %RH

```

Figura 22: Muestra de trama recibida en la terminal de PlatformIO a través del puerto serial

Fuente Propia

Recibimos: “6E FE 24 48 1E 64” expresados en hexadecimal, donde los dos primeros bytes (0x63 y 0xF3) es el valor de la temperatura y el tercer byte recibido (0x24) es el valor CRC de ocho bits. Los siguientes datos en hexadecimal es el mismo caso, pero ahora para la humedad relativa (0x48 y 0x1E) donde el CRC8 es de 0x63. Para comprobar si la medición se recibo, se hizo la comprobación y es simplemente sumar el primer byte con el segundo y debe de dar como resultado el tercer byte.

Una vez con los datos obtenidos podemos hacer la conversión necesaria. La hoja de datos nos proporciona la formulas necesarias para hacer la conversión para la temperatura y la humedad relativa.

$$RH = \left(-6 + 125 \cdot \frac{S_{RH}}{2^{16} - 1} \right) \%RH$$

$$T = \left(-45 + 175 \cdot \frac{S_T}{2^{16} - 1} \right) ^\circ C$$

$$T = \left(-49 + 315 \cdot \frac{S_T}{2^{16} - 1} \right) ^\circ F$$

Figura 23: Ecuaciones para conversión

Fuente: <https://sensirion.com/products/catalog/SHT40>

En el código La función que envía los datos que incluye el entorno de desarrollo Arduino es una instrucción llamada “Serial.write()” es el encargado de mandar datos por el puerto serial. Esta función por default (9600 baudios, 1 bit de inicio, 1 bit de parada, sin bit de paridad). No hay que hacer nada para tener esa configuración, pero en nuestro caso utilizamos un segundo puerto serial, el UART2. Para utilizar el UART2 debemos invocar una librería propia del Esp32 donde la instrucción serial “#include

<HardwareSerial.h>, con esta instrucción declarada ahora podemos un segundo UART y usar la función “Serial2.write()”. Podemos utilizar el UART1 depurar el programa y el UART2 para enviar y recibir datos. El siguiente diagrama muestra la lógica de programación del código para la adquisición de datos del sensor de temperatura y humedad.

DIAGRAMA DE FLUJO

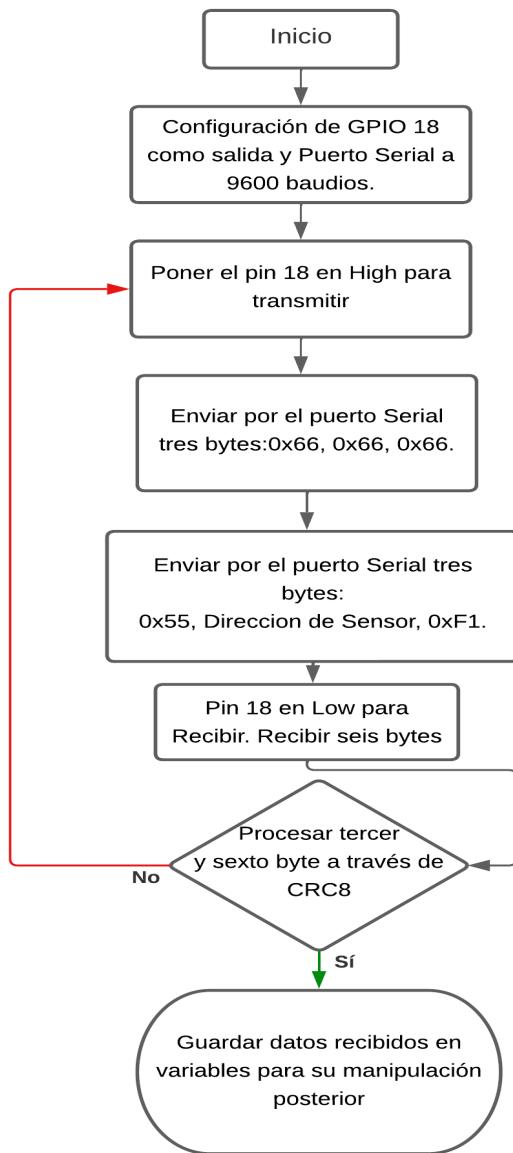


Figura 24: Diagrama de Flujo para el sensor SHT40.

Fuente propia.

El código se implementó en librería para poder integrarlo de forma simple a programas futuros en el proyecto y también para facilitar encontrar posibles errores. En la librería llamada “SHT40” se encuentra una única función que puede utilizar el usuario para obtener la temperatura y humedad de cualquier sensor independientemente de la cantidad de sensores que tenga. Se debe incluir la librería como se muestra en la figura 25.

```
#include <SHT40.h>
SHT40 sht40;
```

Figura 25: Código para librería

Fuente propia

La función para trabajar es: sendGetSerial(dirección del sensor). Además de agregar sht40.setup(); para hacer funcionar la librería correctamente, los datos que se reciben se deben de guardar en alguna variable tipo float. El código se puede encontrar en la sección de Anexos.

3.3.3 *INSTALACIÓN DEL VENTILADOR*

Se necesitaba generar corrientes de aire para poder emular las condiciones climáticas necesarias con respecto a la humedad, para ello se requiere un ventilador con las características necesarias.

El ventilador cuenta con un motor trifásico para mover las aspas y generar corrientes de aire. El ventilador tiene la característica que las corrientes de aire que genera no son uniformes y para solucionarlo se planeó instalar un difusor de inyección a la entrada del túnel de viento al que se conecta el ventilador con el objetivo de lograr uniformidad en las corrientes de aire que se generen.



Figura 26: Motor trifásico.

Fuente: Propia

3.3.3.1 CONEXIÓN DEL MOTOR DEL VENTILADOR Y VARIADOR



a las capacidades del motor al que está conectado. El variador de frecuencia es útil no solo para control, también para protección.

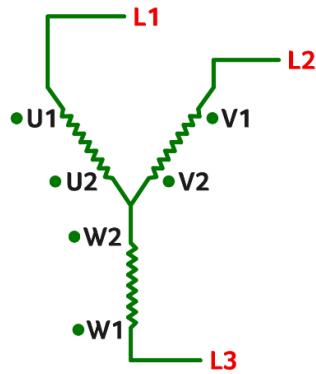


Figura 28: Conexión tipo Estrella

Fuente: <https://intensity.mx/es/blog/arranque-estrella-delta-en-compresores-para-aire-acondicionado>



Figura 29: Cableado de la conexión tipo estrella

Fuente: Propia

3.3.3.2 DIMENSIONAMIENTO DEL MOTOR AL VARIADOR DE FRECUENCIA

El motor tiene sus características y esas características se programaron dentro del variador de frecuencia para que el variador de frecuencia las tome en cuenta y pueda trabajar con el motor trifásico de la forma más optima posible.

Para realizar el dimensionamiento se utilizó el manual del variador para programar con base a los comandos que se declaran en el manual de usuario del variador. Se debe de presionar el botón “Mode” durante dos segundos y con ellos introducir el número de comando que tendrá asignado una función específica con posibilidad de ser modificada. Para introducir el comando se utilizar las flechas de arriba y abajo para incrementar el número de comando. Se continua presionando el botón “ENT/ <” para moverte de a la siguiente columna para anotar el siguiente número de comando. Al terminar de anotar el comando debe mantener presionado el mismo botón anterior (ENT/ <) para entrar a la función y escribir el valor deseado con su respectiva funcionalidad. Al escribirla de nuevo presionas el mismo botón anterior (ENT/ <) para confirmar los cambios. A continuación, se muestran los cambios hechos con sus respectivos comandos.



Figura 30: Panel para programar el dimensionar el variador

Fuente: Manual de usuario Teco L510

00-Basic parameter group	
00-00	Control mode
Range	【0】 : V/F mode 【1】 : SLV mode

Select control mode in parameter 00-00 best suitable for the application.

Default control mode is V/F.

Figura 31: Configuración 1

Fuente: Manual de usuario Teco L510

Se selecciono en esta configuración el modo de actuar el variador. Se escogió el modo (1) “SLV Sensorless Vector” el cual es usado para propósitos generales que requieran alta precisión, control de velocidad.

00-02	Main Run Command Source selection
00-03	Alternative Run Command Source selection
Range	【0】 : Keypad 【1】 : External Run/Stop Control 【2】 : Communication

Figura 32: Configuración 2

Fuente: Manual de usuario Teco L510

Para el comando 00-02 se programó con la opción (2) para utilizarla con RS485 conectado junto el microcontrolador Esp32. El siguiente comando fue 00-03 para escoger control alternativo que fue la opción (0).

00-05	Main Frequency Command Source Selection
00-06	Alternative Frequency Command Source Selection
Range	【0】 :UP/DOWN of Keypad 【1】 :Potentiometer on Keypad 【2】 :External AVI Analog Signal Input 【3】 :External ACI Analog Signal Input 【4】 :External Up/Down Frequency Control 【5】 :Communication setting Frequency 【6】 :PID Output frequency

Figura 33: Configuración 3

Fuente: Manual de usuario Teco L510

Para el comando 00-05 se escogió la opción (5) para utilizarla con el microcontrolador Esp32 y mandar valor de la frecuencia deseada y para el comando 00-06 se seleccionado la opción (1) que es el potenciómetro que tiene integrado el variador de frecuencia.

00-07	Main and Alternative Frequency Command Modes
Range	【0】 :Main or Alternative Frequency. 【1】 :Main frequency + Alternative Frequency

Figura 34: Configuración 4

Fuente: Manual de usuario Teco L510

Para el comando 00-07 se escogió la opción (1) con el propósito de seguridad. Actuar si algo sale mal.

00-09	Frequency Command save on power down (Communication mode)
Range	【0】 :Disable 【1】 :Enable

Figura 35: Configuración 5

Fuente: Manual de usuario Teco L510

Para el comando 00-09 se selección la opción (1) para que recuerde el variador cual fue el ultimo valor de frecuencia se le envió.

00-12	Frequency Upper limit
Range	【0.01~599.00】 Hz
00-13	Frequency Lower limit
Range	【0.00~649.99】 Hz

Figura 36: Configuración 6

Fuente: Manual de usuario Teco L510

Para el comando 00-12 se seleccionó la frecuencia límite superior para de 60 Hz para trabajar ya que es la frecuencia especificada del motor trifásico. Para el comando 00-13 se seleccionó el valor de 0 Hz.

02-05	Motor Rated Power
Range	【0~100.0】 kW
02-06	Motor Rated Frequency
Range	【0~599.0】 Hz

Figura 37: Configuración 7.

Fuente: Manual de usuario Teco L510

Para el comando 02-05 se introducción el valor del motor trifásico que fue de 0.56 Kw y para el comando 02-06 fue de 60 Hz.

08-04	Over voltage Prevention Level during Run Mode
Range	230: 【350~390】 VDC
	460: 【700~780】 VDC

Figura 37.5: Configuración 8

Fuente: Manual de usuario Teco L510

Para el comando 08-04 se introdujo el valor de 350 volts para la opción de 230 volts. Es el valor de sobre voltaje mínimo permitido y nos sirve de protección para el motor. En caso de llegar a esos voltajes el variador actuara y cortara la corriente eléctrica al motor para protegerlo.

09-00	Assigned Communication Station Number
Range	【1 ~ 32】

Figura 38: Configuración 9

Fuente: Manual de usuario Teco L510

Para el comando 09-00 le asignamos el valor de 1 para decirle que el variador tiene la dirección #1.

09-01	RTU code /ASCII code Selection
Range	【0】 :RTU 【1】 :ASCII 【2】 :BACnet

Figura 39: Configuración 10

Fuente: Manual de usuario Teco L510

Para el comando 09-01 se seleccionó la opción (0) para indicarle que queremos utilizar Modbus RTU lo que nos permitirá mandarle instrucciones de manera fácil por RS485 y el microcontrolador ESP32. Para el comando 09-01 se seleccionó la opción (1) de 9600 baudios ya que es la velocidad con la que se trabajan los sensores SHT40.

09-02	Baud Rate Setting (bps)
Range	【0】 : 4800
	【1】 : 9600
	【2】 : 19200
	【3】 : 38400

Figura 40: Configuración 11

Fuente: Manual de usuario Teco L510

El variador de frecuencia tiene muchos más comandos y configuración, pero no fueron necesarios de ajustar o cambiar ya que el trabajo que realizar el variador por defecto es lo suficientemente bueno.



Figura 41: Muestra de configuración del Variador

Fuente: Propia

PROGRAMACIÓN DEL VARIADOR DE FRECUENCIA EN LENGUAJE C++

El variador admite varias formas de control. Puede ser controlador por lazo de corriente 4-20mA, 0-10v, un potenciómetro o por protocolos de comunicaciones como Modbus y BACnet. Se optó por utilizar el protocolo Modbus para evitar utilizar hardware extra.

El microcontrolador esp32 es el encargado de enviar las instrucciones necesarias bajo el protocolo escogido para variador de frecuencia bajo la capa física RS485, se ocupó el mismo material que se usó para lograr comunicación para los sensores de temperatura y humedad.

El variador no cuenta con una librería de código propia, fue necesario realizar una desde cero y para ello se utilizó el manual de usuario del variador de frecuencia donde contaba con poca documentación.

El manual de usuario del variador de frecuencia nos dice como está configurado la comunicación y la que se maneja normalmente en todos los dispositivos. Debemos asegurarnos en nuestro microcontrolador tengas esta misma configuración. La función que incluye el entorno de desarrollo Arduino incluye una instrucción llamada “Serial.write()” es el encargado de mandar datos por el puerto serial.

Esta función por default está configurada de esta manera. No hay que hacer nada, pero en nuestro caso utilizamos el segundo puerto serial, el UART2. Para utilizar el UART2 debemos invocar una librería propia del Esp32 donde la instrucción serial “#include <HardwareSerial.h>”, con esta instrucción declarada ahora podemos usar un segundo UART y usar la función “Serial2.write(). Podemos utilizar el UART1 depurar el programa y el UART2 para enviar y recibir datos. El manual muestra un ejemplo de cómo debe ser la trama de datos que se debe enviar al variador (figura 43).

Default Communication Setting is: Address “1”, 9600 Bits/sec, 1 Start Bit, 1 Stop Bit, and No Parity

Figura 42: Características técnicas de comunicación

Fuente: Manual de usuario Teco L510

Frequency Reference Command: 60.00 Hz (Inverter Node Address: 01)

Command String (hexadecimal): 01 06 25 02 17 70 2D 12

To set the frequency reference to 60.00, a value of '6000' (1770h) has to be send to the inverter

Note: The last 2 bytes of the command strings consist of a CRC16 checksum.

Figura 43: Características técnicas de comunicación

Fuente: Manual de usuario Teco L510

Se armo un paquete de datos y eso será la trama de datos. La trama de datos sigue el protocolo Modbus donde el primer byte (0x01) significa la dirección del esclavo ya que Modbus funciona con maestro y esclavo donde nuestro microcontrolador es el maestro y el variador es esclavo. Entonces se enviar una petición de comando de referencia de frecuencia al esclavo con la dirección 0x01 que es el variador de frecuencia. El segundo byte es 0x06 que es el código de función es básicamente la dirección en memoria del variador donde está el registro al que queremos para luego acceder con los siguientes bytes que enviamos que son 0x25 y 0x02 para cambiar el valor de referencia de frecuencia que deseamos, en este caso es 60 Hz entonces se debe enviar los bytes de 60 Hz en hexadecimal 0x17 y 0x70. Los últimos son los dos bytes de CRC de 16 bits. En este caso el CRC16 es más complejo y debe calcular cada vez que enviamos un valor de frecuencia diferente.

Para otros envíos de instrucciones el manual de usuario nos proporciona la trama de datos completa con el CRC16 calculado para instrucciones como encender y apagar el variador.

El algoritmo para el CRC16 es el siguiente:

1. Carga un registro de 16 bits con FFFF en hexadecimal (todos unos). Llama a este registro CRC.

2. Realiza una operación O exclusiva (XOR) entre el primer byte de 8 bits del mensaje y el byte de menor orden del registro CRC de 16 bits, y coloca el resultado en el registro CRC.
3. Desplaza el registro CRC un bit hacia la derecha (hacia el LSB), rellenando con ceros el MSB. Extrae y examina el LSB.
4. (Si el LSB es 0): Repite los pasos (3) (otro desplazamiento). (Si el LSB es 1): Realiza una operación XOR entre el registro CRC y el valor del polinomio A001 en hexadecimal (1010 0000 0000 0001), y coloca el resultado en el registro CRC.
5. Repite los pasos (3) y (4) hasta que se hayan realizado 8 desplazamientos. Al finalizar, se habrá procesado un byte completo de 8 bits.
6. Repite los pasos (2) a (5) para el siguiente byte de 8 bits del mensaje. Continúa haciendo esto hasta que se hayan procesado todos los bytes. El contenido final en el registro CRC es el valor del CRC. Al enviar el valor del CRC, el byte de menor orden debe ser enviado primero, seguido del byte de mayor orden. Por ejemplo, si el valor del CRC es 1241 en hexadecimal, el byte de mayor orden debe ser 41 en hexadecimal y el byte de menor orden debe ser 12 en hexadecimal.

Cuando se envía el paquete de datos armado y verificado con el CRC16 al variador, el protocolo Modbus especifica que se reenvía el mismo paquete de datos que nosotros enviamos como comprobante que el paquete de datos que enviamos fue correcto, en el caso del variador de frecuencia es al revés. Si no recibía nada de parte del variador al enviar el paquete de datos significa que el variador recibió satisfactoriamente los datos, pero si llega una trama de datos de parte del variador significa que existe un error en alguna parte.

DIAGRAMA DE FLUJO

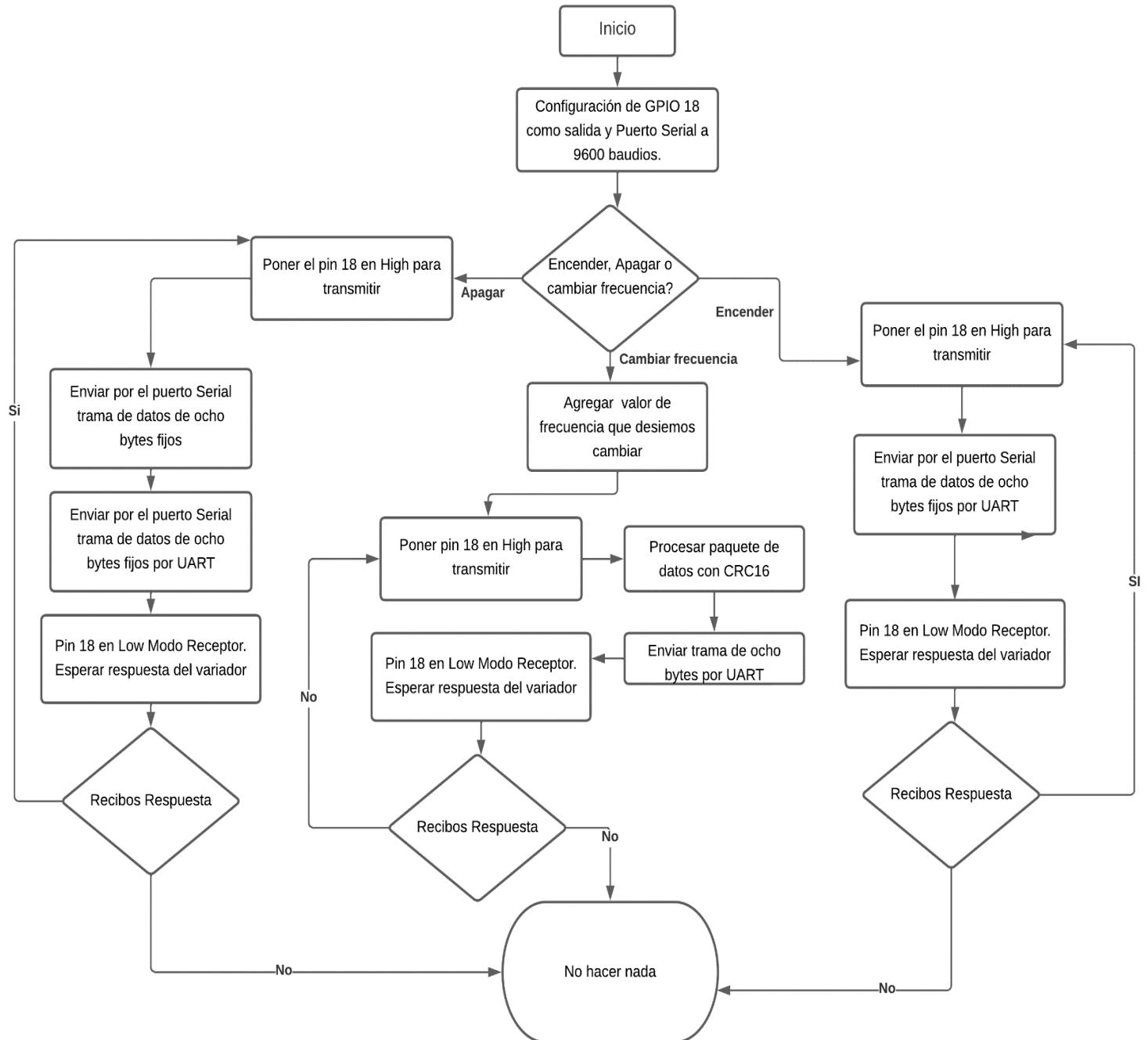


Figura 44: Diagrama de flujo programación de variador de frecuencia

Fuente: propia

Al igual que el código para el sensor Sht40, el código realizado para el variador se hizo librería para integrarla a futuros programas con facilidad. Donde se inicializar como se muestra en la figura 45.

```
#include <TECO_L510.h> // libreria propia para el variador de frecuencia TECO L510
TECO_L510 teco;
```

Figura 45: Instrucción de inicialización de librería.

Fuente: Propia

Las funciones que se pueden usar de la librería son: setup(): para inicializar todo lo necesario para hacer funcionar la librería, runVariador(): para encender variador, stopVariador(): parar el variador y set_freq_variador(frecuencia): para hacerle saber al variador a que frecuencia queremos hacer girar el ventilador. Un detalle importante es que se debe enviar el valor de la frecuencia antes de encender el ventilador. Entonces primero enviamos la función set_freq_variador() esperamos 40ms entre instrucción y luego enviamos la función runVariador().

Detalles que se deben tomar en cuenta es que no se debe enviar instrucciones al variador de forma secuencia e inmediata, debe haber un tiempo de espera de 40ms entre cada instrucción ya que es el tiempo de respuesta del variador. Al no respetar esto el variador no tomara en cuenta las demás instrucciones porque no es capas de escucharlas o procesarlas a tiempo. Otro de detalle es que el microcontrolador es demasiado rápido provocando que al momento de enviar la trama de datos y cambiar el pin de estado alto a estado bajo, lo hace mucho antes de lo debido provocando que corte la trama de datos. Para solucionar esto se debe incluir alguna instrucción que detenga el programa temporalmente. Para comprobar esto se utilizó un analizador lógico y software Logic 2; donde nos arroja los siguientes resultados que se muestran en la figura 46.



Figura 46: Analizador Lógico en UART Y PIN 18

Fuente: Propia

Donde en la fila con nombre Channel 0 se analiza en el pin TX del UART se puede toda la trama de datos y en la fila de abajo con nombre de Channel 1 se analiza el pin 18 que controla el modo transmisor y receptor del max485, se puede observar que pasa de transmisor a receptor a mitad del envío de la trama de datos. Para solucionar esto se debe retrasar esta acción y se logró utilizando la instrucción “vTaskDelay(pdMS_TO_TICKS(80))” de la librería FreeRTOS que hace lo mismo que la clásica función “Delay(80)” pero sin bloquear por completo el microcontrolador. Se agrega el valor de 80ms porque es el tiempo mínimo necesario para que se envíen todos los datos.

[CONEXIÓN DEL VARIADOR DE FRECUENCIA AL MICROCONTROLADOR ESP32](#)

Para la conexión del variador al microcontrolador se utiliza RS485 donde se utiliza el conector RJ45. El manual del variador de frecuencia muestra los cables que se utilizan dentro del RJ45 para enviar y recibir datos.



Figura 47: RJ45

Fuente: <https://uelectronics.com/producto/conector-rj45-utp-cat-5/>

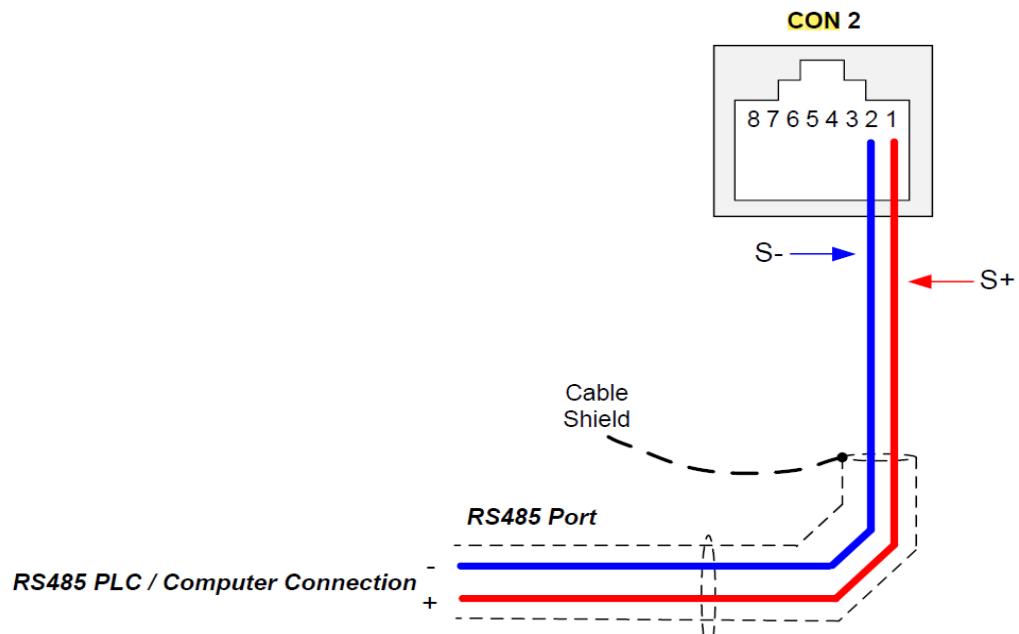


Figura 48: Conexión RS485

Fuente: Manual de usuario Teco L510

CIRCUITO CON VARIADOR DE FRECUENCIA

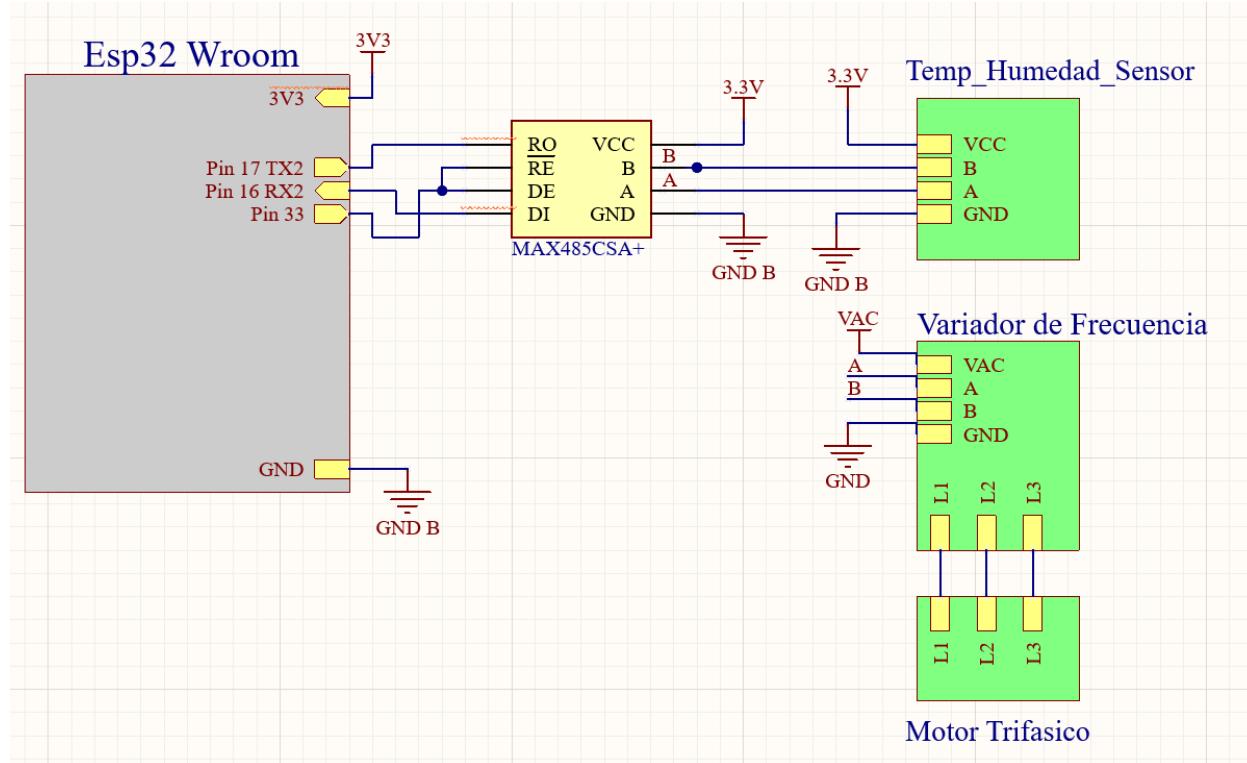


Figura 49: Circuito General

Fuente: Propia

El esquemático del circuito de la figura 49 muestra dos nuevos recuadros que representan el variador de frecuencia y el motor trifásico. El recuadro verde con nombre “Variador de Frecuencia tiene las etiquetas “A y B” a la salida de las entradas “A y B”, se muestra de esta manera para indicar que existe una conexión con todas las etiquetas que tengan “A y B” al realizarlo de esta manera se simplifica el uso de conexiones entre componentes logrando una fácil interpretación del cableado existente.

3.3.3.3 SENSOR DE VIENTO FH400

El Sensor FH400 funciona con un rango de voltaje de 18 volts a 29 volts. Se trabaja con 20 volts. Tiende a consumir 15mA nominales, tiene un tiempo de respuesta de 400ms, hace mediciones de temperatura desde los -40C hasta 105C, Humedad Relativa de 5 a 95% y velocidad de viento desde 0.15 m/s hasta 10 m/s, Arroja valores

analógicos de 0-5 volts, comienza a operar después de los 10 segundos. Para polarizar el sensor se utilizó una fuente de alimentación de laboratorio marca GWINSTEK modelo GPC-30300.

El sensor es bastante sensible a perturbación externas como puede ser el movimiento y temperatura ambiental. La señal analógica en milivolts aumenta de magnitud dependiendo de cuanto voltaje se le suministre. Para leer la señal entregada por el sensor, se utilizó un filtro de media móvil para reducir el “ruido” utilizando entre 50 a 100 muestras.

INSTALACIÓN Y UBICACIÓN DEL SENSOR FH400

El sensor se instaló en la primera sección del túnel de viento, en la parte baja del túnel de viento, justo antes de la ubicación de los intercambiadores de calor.

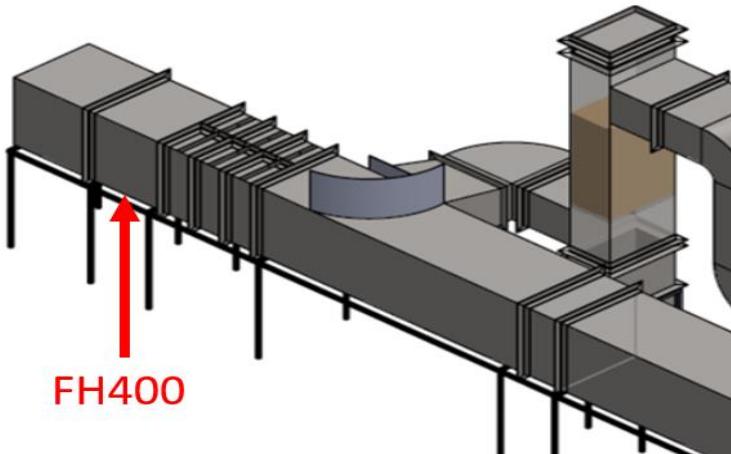


Figura 50: Ubicación F400.

Fuente: Propia

El sensor FH400 se instaló tomando en cuenta la orientación que señala el propio sensor. Cuenta una flecha marcada en el cabezal que debe estar apuntando hacia donde fluye el aire. Para lograr hacer la instalación se realizó un agujero en la parte inferior del túnel un taladro y utilizando diferentes grosores de broca hasta llegar al diámetro suficiente para introducir el sensor y fijar su posición con cinta de aislar y la sujeción de rosca que cuenta el propio sensor.



Figura 51: Orientación FH400

Fuente: Propia



Figura 52: Instalación FH400

Fuente: Propia

PROGRAMACIÓN DEL SENSOR FH400 EN C++

Como en los códigos anteriores se realizó una librería para su posterior integración en la programación futura. Donde se inicializa como se muestra en la figura 53.

```
#include <FH400.h> // libreria propia para el sensor FH400  
FH400 fh400;
```

Figura 53: Inicializa librería para el sensor FH400

Fuente: Propia

La librería cuenta con las funciones `setup()`: que configura lo necesario para hacer funcionar la librería, que es básicamente activar los pines necesarios y declararlos como entradas, además de activar los puertos como ADC (convertidor analógico digital) para leer la señal analógica del sensor FH400. En librería se pueden utilizar tres funciones más donde cada una ella entrega el valor censado. La primera es `conversión_velocidad()` que entrega un valor entero con el valor de la velocidad del viento.

Lo que hay dentro de esa función es que los valores que arroja el sensor son analógicos por ello debemos leerlo con el ADC de 12 bits de resolución del microcontrolador esp32 con voltaje de referencia 3.3 volts. El sensor FH400 arroja una señal de medición de voltajes de 0 a 10 volts por lo que no podremos medir valores superiores al voltaje de referencia del esp32.

Para la aplicación requerida que se ocupó fue medir el viento que puede mover ventilador conectado a motor trifásico de hasta 60 Hertz. El sensor pudo entregar una medición pico de 2.20 volts al momento de medir la velocidad de viento entregada por el ventilador a su máxima potencia. Esto significa que el valor máximo de medición que se puede leer está dentro del rango de medición permitido por el microcontrolador sin que pueda ser dañado.

A continuación, se explica el desarrollo del algoritmo desarrollado para leer las señales analógicas del sensor: Para hacer la conversión necesaria de la lectura de la velocidad según la hoja de datos del sensor se debe realizar la formula (figura 54).

$$\text{Velocity} = \frac{\text{Voutput} * \text{Velocity High Range}}{\text{Vout Maximum}}$$

Figura 54: Formula para conversión de velocidad de viento.

Fuente: <https://www.degrec.com/products/embedded-airflow-sensors-switches/fh-series-probe-air-velocity-temperature-and-humidity-sensors/>

- Donde Voutput = Lectura del ADC.
- Velocity High Range = es 10160 mm/s.
- Vout Máxima = 5 volts.

Lo siguiente que se hizo en la función es conseguir 20 muestras con el ADC y promediarlas, lo que estamos haciendo es hacer un filtro con media móvil, esto con el objetivo de eliminar el “ruido” que otorga el sensor a sus lecturas debido a su sensibilidad al movimiento y temperatura. Además de esto las mediciones que se realizan se debe de incluir una compensación de medición según lo menciona la hoja de datos del sensor.

Air Velocity Range	Air Velocity Accuracy*
0.15 to 1.0 m/s (30 to 200 fpm)	± (1% of reading + 0.05 m/s [10 fpm])
0.5 to 10 m/s (100 to 2,000 fpm)	± (4% of reading + 0.10 m/s [20 fpm])
1.0 to 20 m/s (200 to 4,000 fpm)	± (5% of reading + 0.15 m/s [30 fpm])

Figura 55: Compensación

Fuente: <https://www.degrec.com/products/embedded-airflow-sensors-switches/fh-series-probe-air-velocity-temperature-and-humidity-sensors/>

Para sensor que el cuento es la segunda opción que entrega mediciones de 0.5 a 10 m/s. Se le debe agregar en código el 4% de la lectura más el 0.10 m/s. La siguiente función es conversión_temp(): esta función utiliza una formula casi igual al de la velocidad de temperatura y también tiene declarado un filtro de media móvil con 50 a 100 muestras. Se incluyó una compensación de medición que depende de la velocidad del viento.

$$\text{Temperature } (\text{°C}) = \frac{\text{Tout} * \text{TemperatureRange}(\text{°C})}{\text{ToutMaximum(VDC)}} = \frac{\text{Tout} * 100}{\text{ToutMaximum(VDC)}}$$

Figura 56: Conversión de Temperatura

Fuente: <https://www.degrec.com/products/embedded-airflow-sensors-switches/fh-series-probe-air-velocity-temperature-and-humidity-sensors/>

- Donde: Tout = Lectura del ADC
- TemperatureRange = 100 C y ToutMaximun = 10 volts.

$$\begin{aligned} \text{at velocities} > 0.5 \text{ m/s [100 fpm]} &= \pm 1 \text{ °C [1.8 °F]} \\ \text{at velocities} < 0.5 \text{ m/s [100 fpm]} &= \pm 2 \text{ °C [3.6 °F]} \end{aligned}$$

Figura 57: Compensación de temperatura

Fuente: <https://www.degrec.com/products/embedded-airflow-sensors-switches/fh-series-probe-air-velocity-temperature-and-humidity-sensors/>

Para la última función que es conversion_hum(): que es para obtener la humedad y en este caso no se requiere compensación alguna pero si se incluye un filtro de media móvil para 20 muestras.

$$\text{Humidity } (\%) = \frac{\text{Hout} * \text{Humidity Range High } (\%)}{\text{Hout Maximum } (\text{VDC})} = \frac{\text{Hout} * 100}{\text{Hout Maximum } (\text{VDC})}$$

Figura 58: Formula para conversión de humedad relativa

Fuente: <https://www.degrec.com/products/embedded-airflow-sensors-switches/fh-series-probe-air-velocity-temperature-and-humidity-sensors/>

DIAGRAMA DE FLUJO

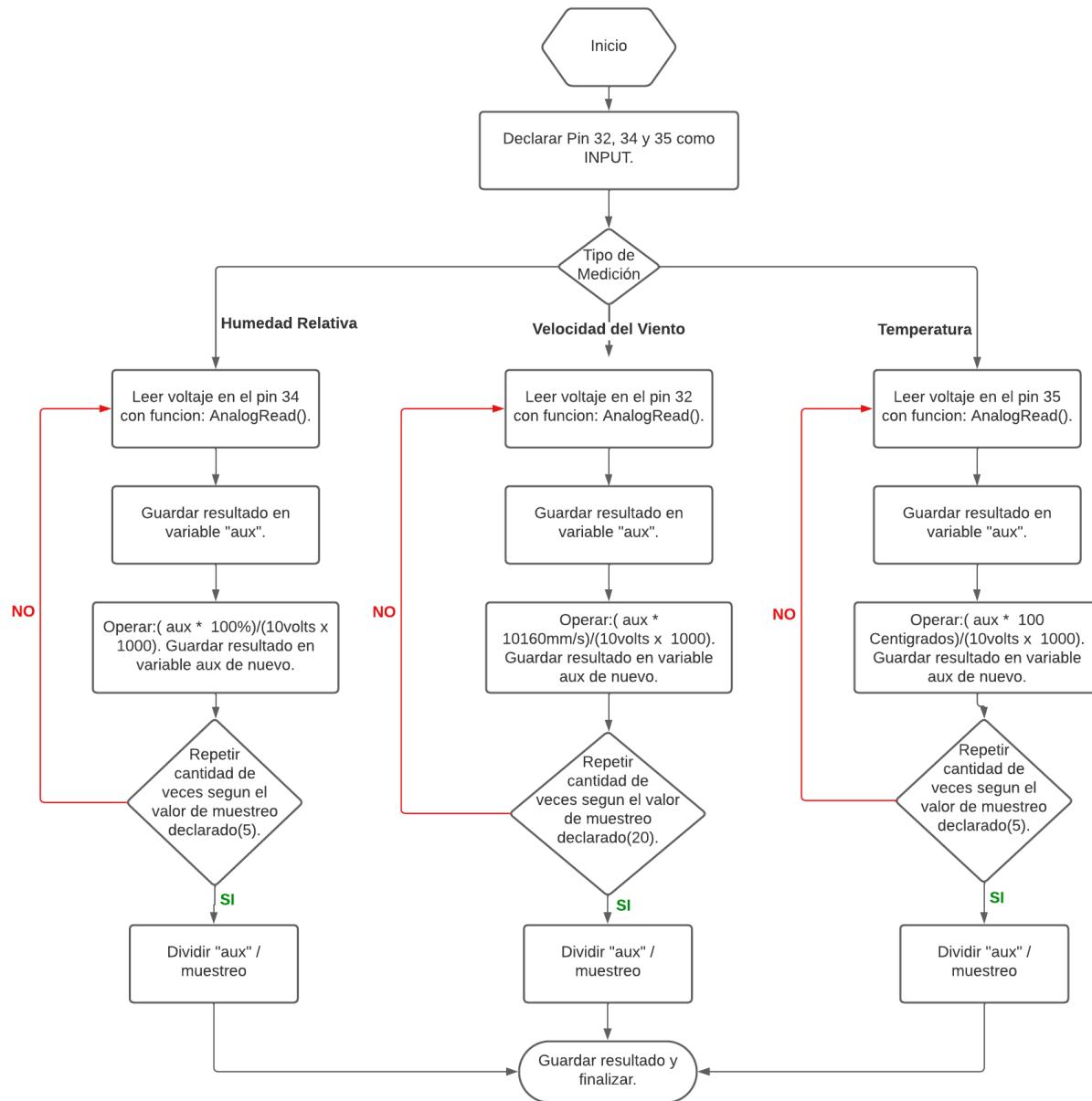


Figura 59: Diagrama de Flujo para programación para sensor FH400.

Fuente: Propia.

CIRCUITO CON SENSOR DE VELOCIDAD DE VIENTO FH400

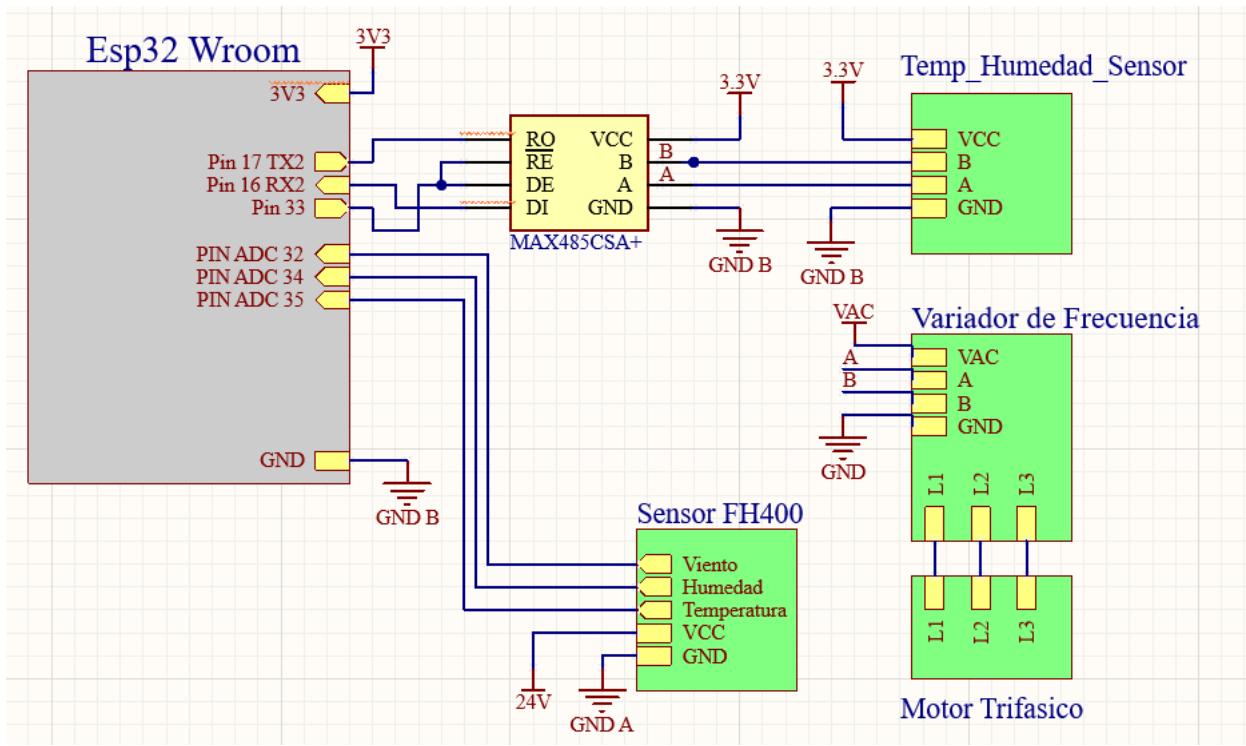


Figura 60: Circuito general con sensor FH400.

Fuente: Propia.

El Recuadro Verde de la figura 60 representa al sensor FH400 con sus respectivos puertos (salidas o entradas) del sensor. Se conectan directamente al microcontrolador para procesar las mediciones entregadas por el sensor.

El diagrama se especifica “GND A” indicando que habrá un aislamiento de “tierras” entre “GND A” y “GND B”, donde GND A será para la electrónica que entregue señales analógicas y GND B será para señales digitales. Separarlas evitara ruido eléctrico que pueda ocasionar inestabilidad en el sistema final (Microcontrolador). La excepción en el diagrama existe “GND” y “VAC” quienes no tienen conexión alguna con el circuito del microcontrolador.

3.3.4 SENSOR SUMERGIBLE DE TEMPERATURA DS18B20

PROGRAMACIÓN EN EL LENGUAJE C++

El sensor ya cuenta con sus librerías desarrolladas. Utiliza dos librerías donde requiere tres funciones en el código para lograr obtener las mediciones que se requieran del sensor. El sensor se trabajó con la resolución de medición por defecto de 12 bits y únicamente se ocupó un solo sensor.

DIAGRAMA DE FLUJO

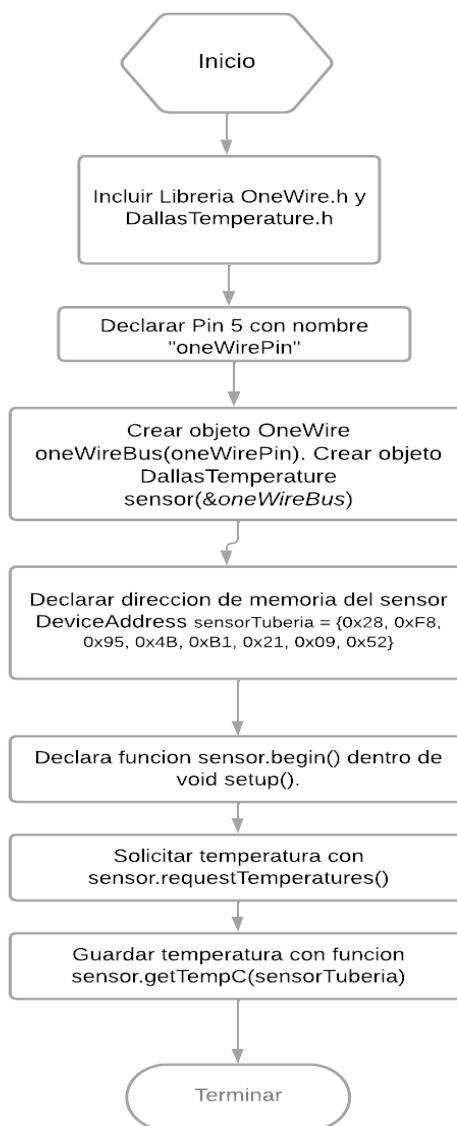


Figura 61: Diagrama de Flujo para programación para sensor FH400.

Fuente: Propia.

CIRCUITO ELÉCTRICO DEL SENSOR DE TEMPERATURA DS18B20

El sensor tiene conexión con el microcontrolador "B" en el pin digital #5 donde en el esquemático se muestra que requiere una resistencia pull-up de 4.7k ohms para funcionar. La magnitud de la resistencia varía dependiendo del largo del cable utilizado. En este caso de utiluso un cable de alrededor de dos metros de longitud.

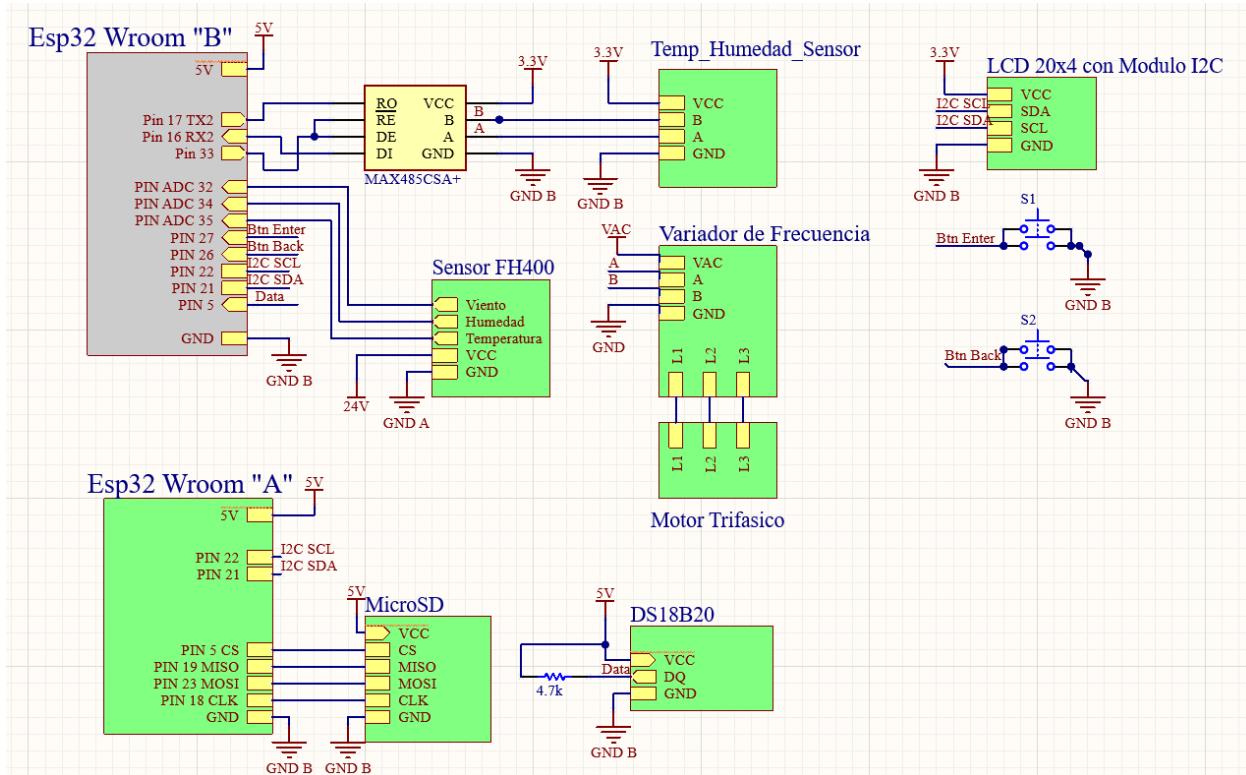


Figura 62: Circuito General con sensor de temperatura DS18B20

Fuente: Propia

INSTALACION DEL SENSOR DS18B20

Se utilizo una termocupla para el sensor de temperatura con el objetivo de poder acoplarlo a la tubería y medir la temperatura de líquido o gas que se ingrese esta vía. Fue instalado en las tuberías de los intercambiadores de calor más cercanos al ventilador y variador de frecuencia.

DS18B20

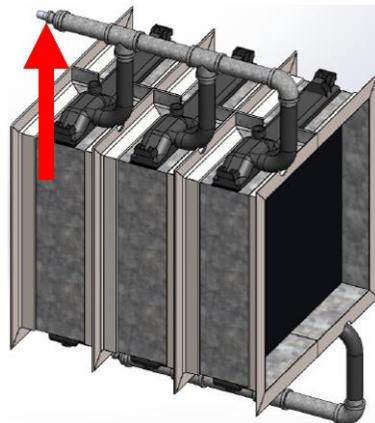


Figura 63: Instalación de sensor en tuberías para intercambiador de calor

Fuente: Diseño y construcción de un sistema de acondicionamiento de aire ambiental, para la validación experimental de sistemas sustentables de para productos diverso.



Figura 64: Sensor de temperatura en termocupla.

Fuente: propia.

3.3.5 PROGRAMACIÓN DE INTERFAZ DE USUARIO LOCAL

Para lograr utilizar los sensores y actuadores para la emulación de aire sintético es necesario contar con una interfaz de usuario para mostrar las mediciones que se entregan. Se utilizó una pantalla LCD de 20x4, dos botones simples normalmente abiertos.

La pantalla LCD 20x40 fue inicializada como esclavo y el microcontrolador como maestro para lograr las condiciones necesarias que requieren ser declaradas por el protocolo I2C para establecer comunicaciones entre dispositivos. En esta interfaz de usuario local se integraron las librerías correspondientes de los sensores utilizados y actuadores ocupados donde ahora se suman las librerías de la pantalla LCD.



Figura 65: Pantalla LCD como interfaz de usuario

Fuente: Propia.

El código realizado para la interfaz local es donde se integró todos los códigos de programación de los sensores y actuadores vistos anteriormente(librerías). El código de se ejecuta en el microcontrolador esp32 “B” encargado de la obtención de las señales adquiridas por los sensores y su respectivo procesado, además de mostrar al usuario los valores censados en tiempo real.

DIAGRAMA DE FLUJO

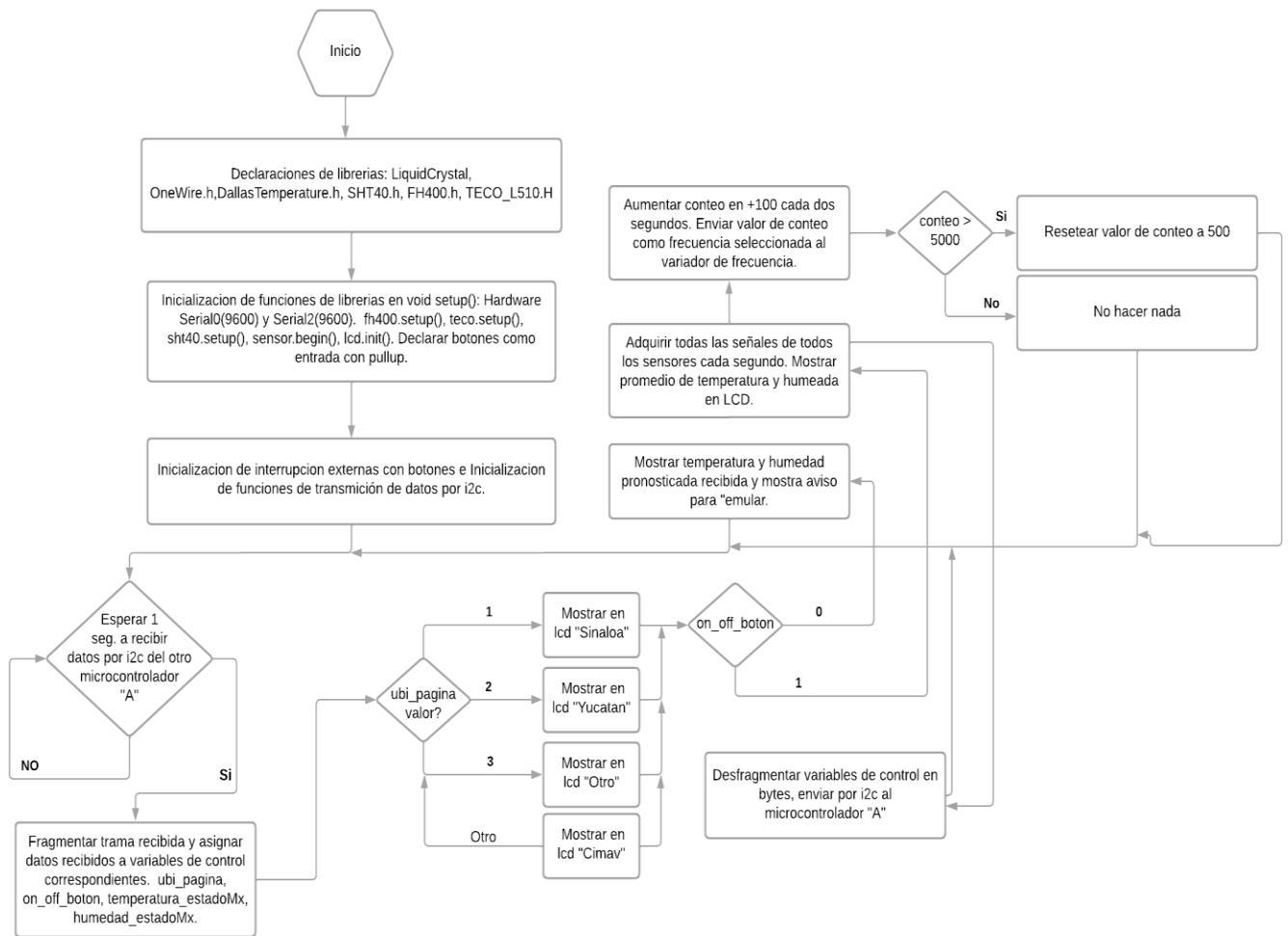


Figura 66: Diagrama de flujo del funcionamiento de la interfaz local

Fuente: Propia

CIRCUITO INTERFAZ DE USUARIO LOCAL

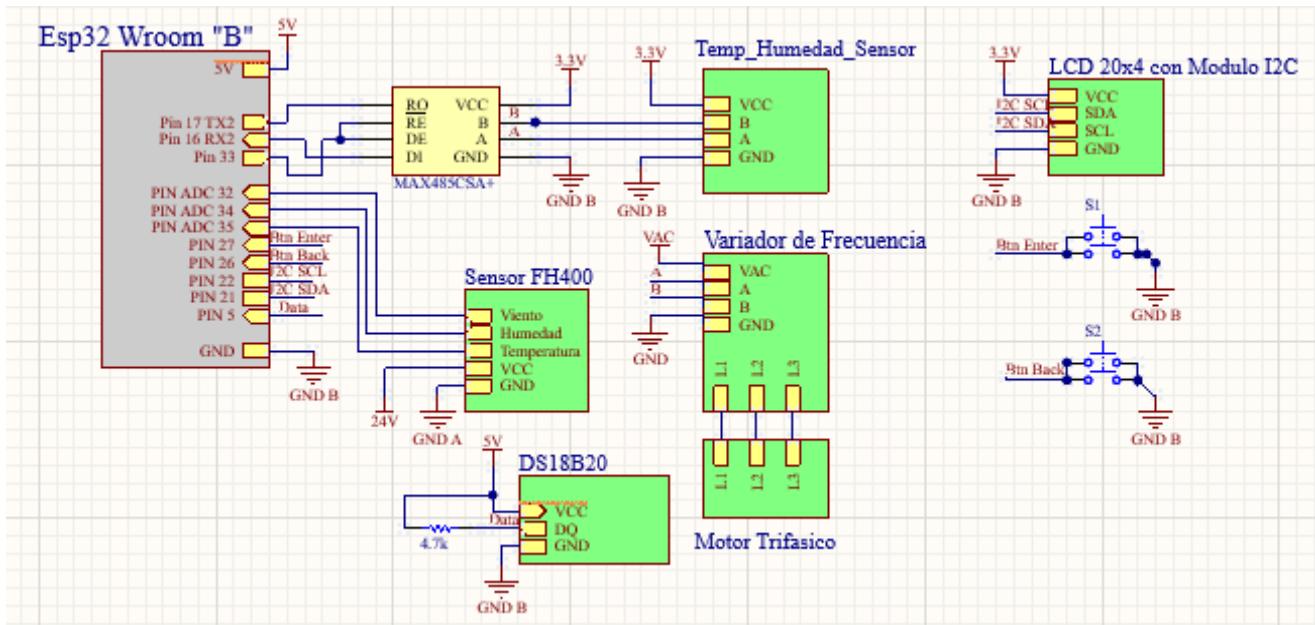


Figura 67: Circuito para Interfaz de Usuario

Fuente: Propia

3.3.6 INSTALACIÓN DE TUBERÍAS

El túnel de viento se le debe de inyectar agua a temperaturas altas para lograr la humedad necesaria, esto en los primeros intercambiadores y para ello deben de pasar atreves de tuberías y mangueras que logran soportar temperaturas altas. Ahora se muestra una imagen con un render en 3D de las tuberías a colocadas.

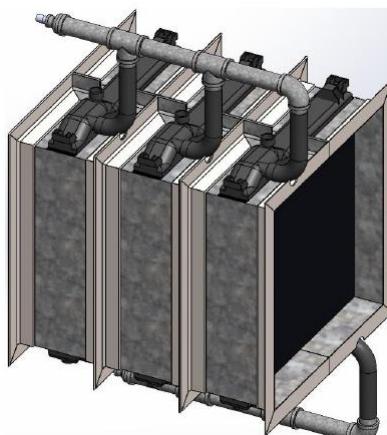


Figura 68: Tuberías en intercambiador de calor

Fuente: Diseño y construcción de un sistema de acondicionamiento de aire ambiental, para la validación experimental de sistemas sustentables de para productos diverso

El material de las tuberías utilizadas fue de cobre y acero galvanizado con medidas de 3/4 de pulgadas y 1/2 de pulgadas. Se instalaron cuatro arreglos de tuberías en las siguientes secciones:



Figura 69: Ensamble de tuberías

Fuente: Propia

3.3.7 PAGINA WEB COMO INTERFAZ EN LA NUBE

Se diseño una página web alojada en un servidor dentro del microcontrolador Esp32 con el propósito de mostrar las variables climáticas pronosticadas extraídas desde un software especializado llamado “Meteonorm”. También para mostrar las variables climáticas emuladas con su respectivo sistema de control. La interfaz web tiene la capacidad de seleccionar cualquier estado de México para emular siempre que esté disponible dentro de la interfaz y se cuente con la base de datos de las variables climáticas a emular.

Para el desarrollo de este mismo, se implementó otro microcontrolador extra del mismo tipo (ESP32), únicamente para poder dedicarse a la ejecución del servidor Web y el otro microcontrolador a la obtención y control de variables censadas. Ambos microcontroladores tienen comunicación entre ellos para intercambiar información necesaria.

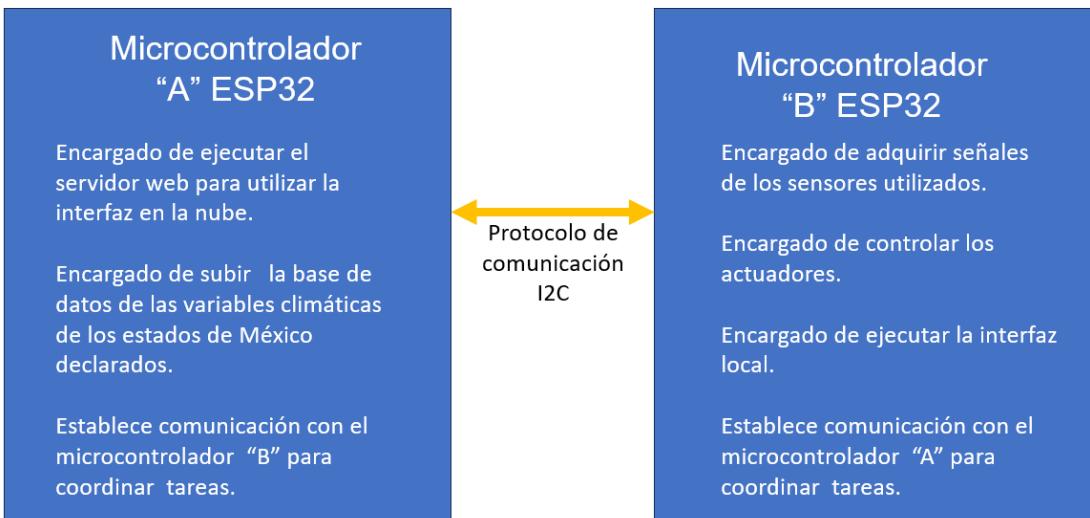


Figura 70: Descripción de tareas encargadas a microcontroladores

Fuente: Propia.

El microcontrolador “A” es el encargado de ejecutar la interfaz en la nube a través del servidor web donde muestra la temperatura y la humedad del estado de México seleccionado. Estas variables climáticas fueron pronosticadas con un software especializado llamado “Meteonorm”. Para lograr lo anterior se utiliza una base de datos no relacional guardados en un archivo con extensión .csv que se almacena en una tarjeta microSD y es leída por el microcontrolador cuando inicia el servidor.

El microcontrolador utiliza un módulo microSD para poder leer los datos de este. Se utilizó el protocolo de comunicación SPI para establecer comunicaciones entre el módulo y el microcontrolador. Con esto fue posible leer y escribir cualquier archivo que se requiera en el momento necesario.

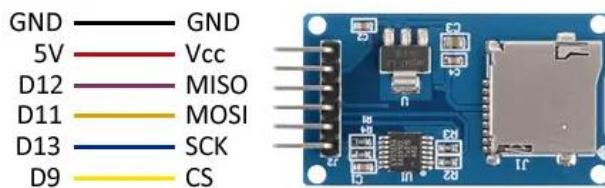


Figura 71: Modulo microSD

Fuente: <https://www.luisllamas.es/tarjeta-micro-sd-arduino/>

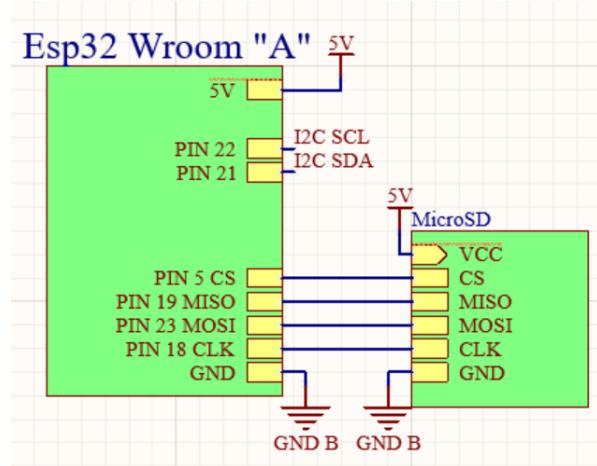


Figura 72: Circuito modulo MicroSD conexión a microcontrolador

Fuente Propia.

El microcontrolador “A” tiene comunicación con el microcontrolador “B” para coordinar con las tareas designadas. El microcontrolador “A” es declarado como esclavo y el “B” como maestro según el protocolo I2C. Es importante recalcar que el microcontrolador “B” ya fue declarado previamente como maestro al estar utilizando la pantalla LCD 20x4 que utiliza comunicación I2C.

La interfaz web se desarrolló utilizando lenguaje C++ para la lógica del servidor web alojado en el microcontrolador y se utilizó lenguajes HTML, CSS y JAVASCRIPT para la interfaz gráfica en la nube donde se ejecuta del lado del usuario. Con el objetivo de tener un sencillas y facilidad visual para el usuario final que analizara el controlador el circuito final.

Se desarrollaron tres páginas web que integran la interfaz de usuario en la nube para seleccionar el estado de México a emular y mostrar sus respectivas variables climáticas. Cada página requiere código de programación extra que entrega la lógica para funcionar, así como el estilo que otorga una vista interesante a la vista.

Para el algoritmo para crear un servidor web con los protocolos HTTP, TCP e IP fueron simplificados por la librería utilizada llamada ESPAsyncWebServer, que permite crear fácilmente un servidor, así como subir todos los archivos que se quieran mostrar en la página web o interfaz mostrada. El servidor es asíncrono lo que significa que cada

conexión de usuario nueva lo tratará aislado de los otros usuarios. El servidor utiliza el puerto 80 para conectividad.

La librería utiliza SPIFFS quien es el encargado de guardar todos los archivos de cualquier tipo en la memoria flash del microcontrolador de 4MB. Ahí se guardan 14 archivos de distintos formatos programados para hacer funcionar la interfaz en la nube.

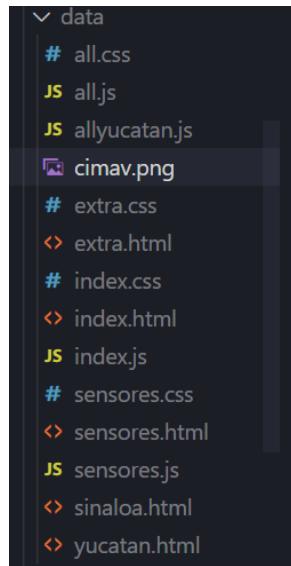


Figura 73: Archivos en la memoria Flash

Fuente: propia.

Se decidió utilizar un microcontrolador dedicado para la interfaz de usuario en la nube para la cantidad de procesado que requería para mostrar el resultado del control de los sensores y actuadores. Otra razón más fue por la cantidad de memoria flash limitada que se tendría al integrar los dos códigos de programación en un solo microcontrolador, habría faltado memoria flash al almacenar cualquier cambio o avance en futuros cambios en la programación.

DIAGRAMA DE FLUJO DE BACK-END EN SERVIDOR WEB

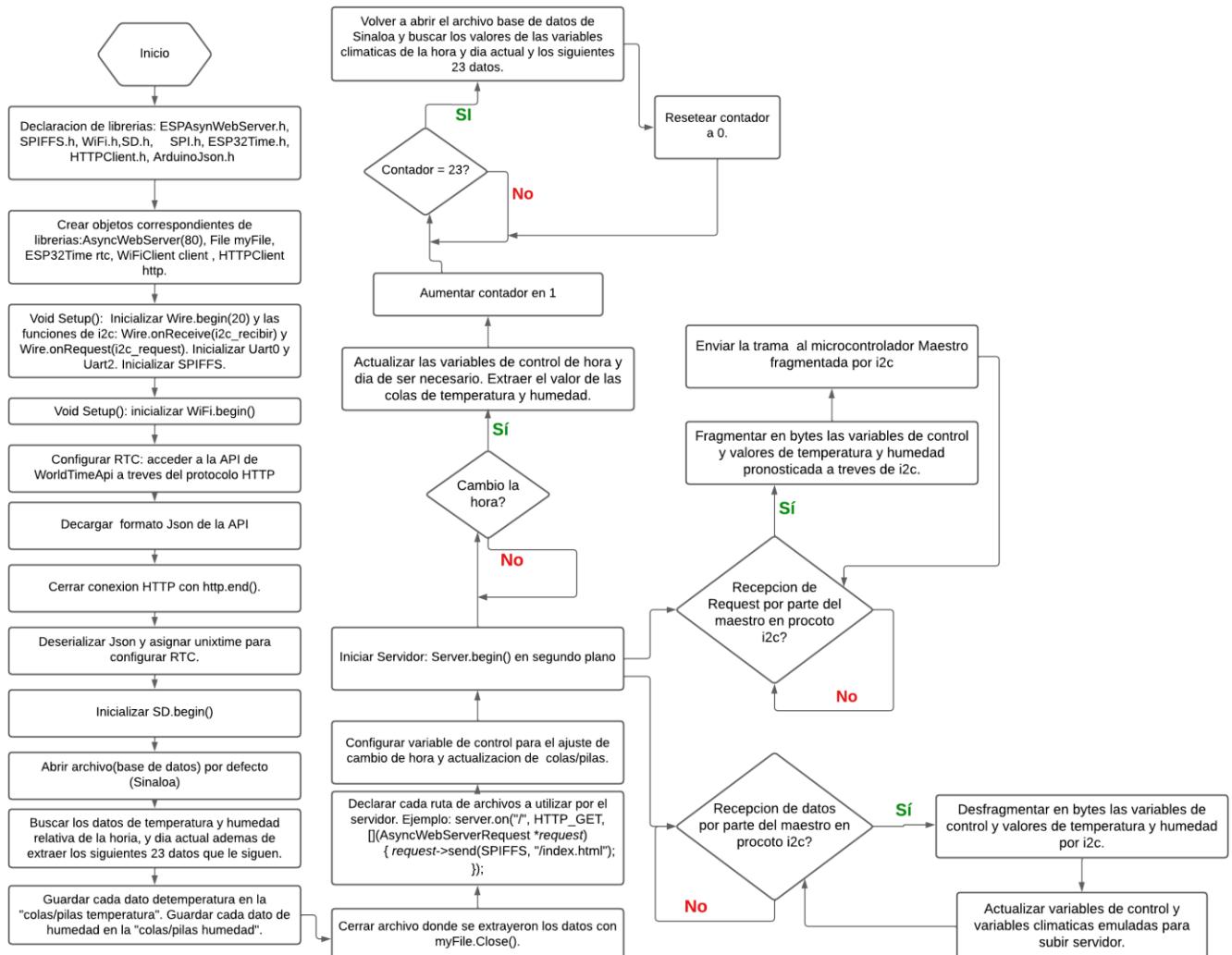


Figura 74: Código para la interfaz de usuario en la nube

Fuente: Propia

DIAGRAMA DE FLUJO DE FRONT-END EN SERVIDOR WEB

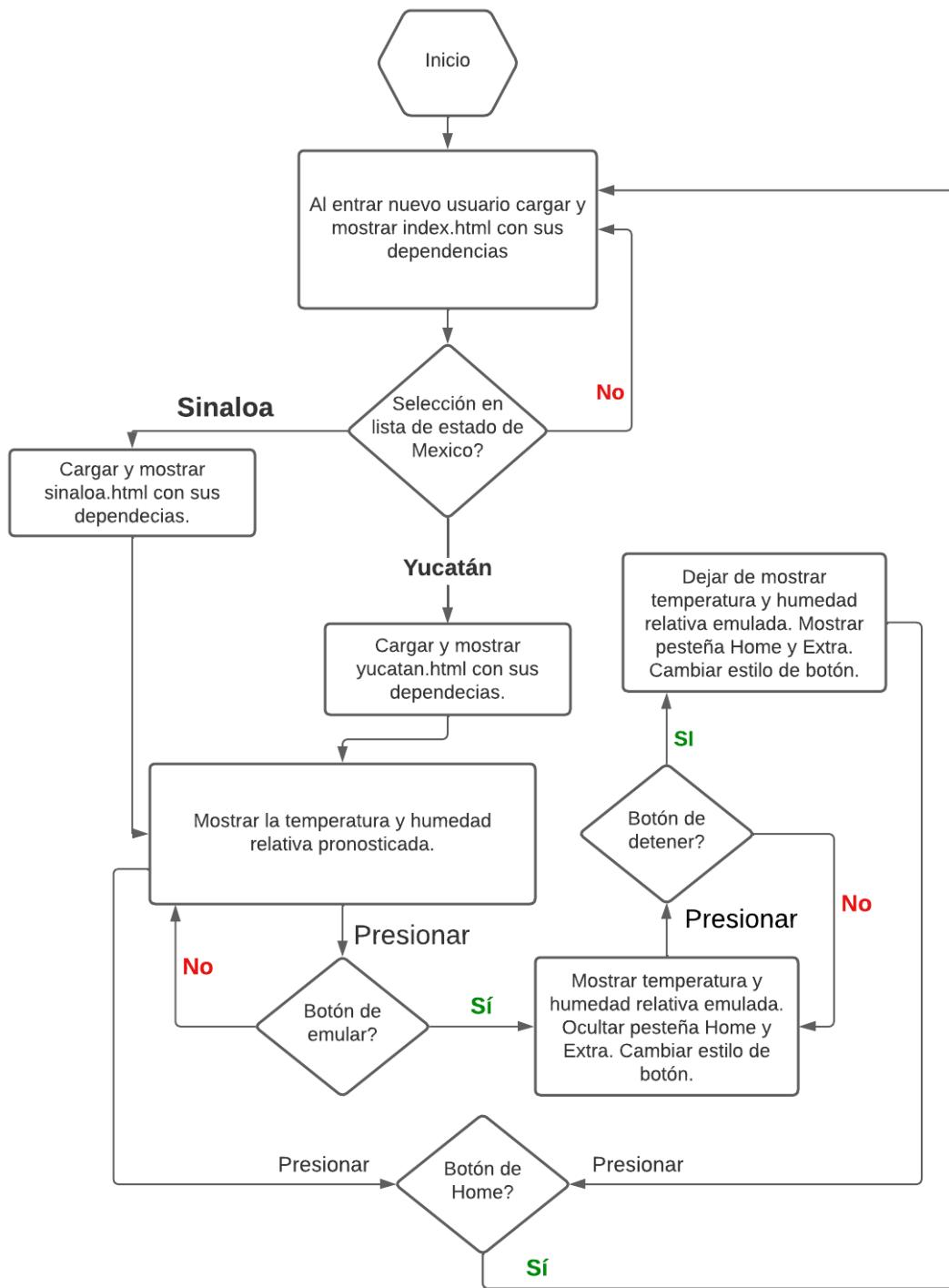


Figura 75: Diagrama de flujo Front-end para interfaz en la nube.

Fuente: Propia

4 RESULTADOS

4.1.1 SENSOR DE TEMPERATURA Y HUMEDAD RELATIVA SHT40.

Para mostrar las mediciones del sensor se utilizó el software de Matlab para graficar los datos censados. Se realizó la gráfica para un solo sensor. Los datos se entregaban a Matlab por el puerto serial para graficar en tiempo real a través del App Designer de Matlab.

En la figura 76 se muestra como sube la temperatura gradualmente según pasa el tiempo. Esto debe a que el sensor fue sometido al lado de una herramienta de soldadura por plataforma caliente donde este incremente su temperatura gradualmente hasta alcanzar el valor establecido. Como se muestra en la figura 76 mide a la par que la soldadura por plataforma caliente incrementa.

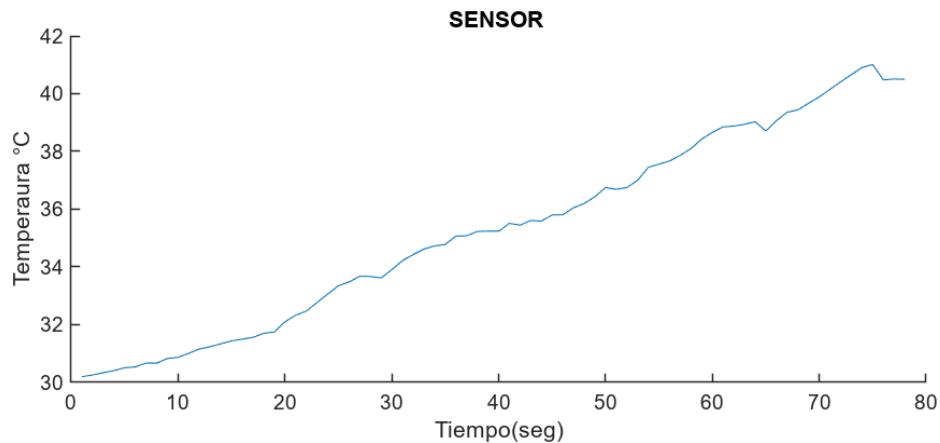


Figura 76: Temperatura Sensor

Fuente: Propia

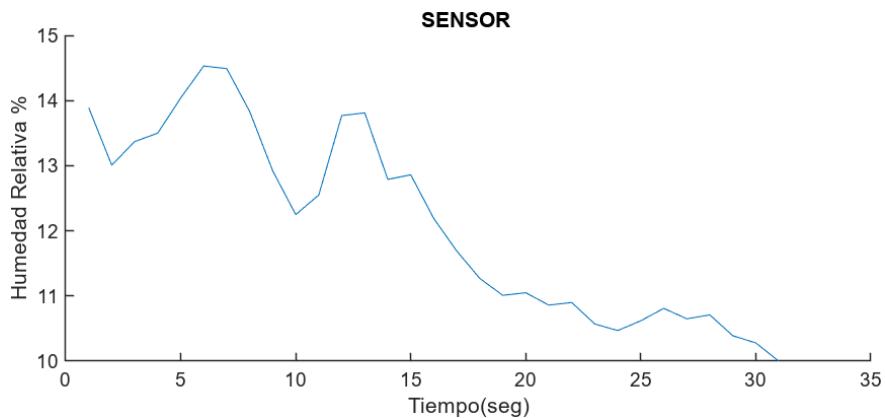


Figura 77: Humedad Relativa Sensor

Fuente: Propia

La figura 77 muestra como decae la humedad debido a que fue sometido con un rociador con agua durante unos segundos para aumentar la humedad.

4.1.2 SENSOR VELOCIDAD DE VIENTO FH400

El sensor cuenta con la capacidad de medir temperatura, humedad relativa y velocidad del viento. Únicamente se utilizó la función de medir la velocidad de viento en el código de programación. Se realizaron las pruebas con el ventilador de una fuente de alimentación de laboratorio, donde se acercaba y alejaba el sensor del ventilador, esto se puede apreciar en la figura 78 donde se muestra con sube y baja la curva rápidamente.

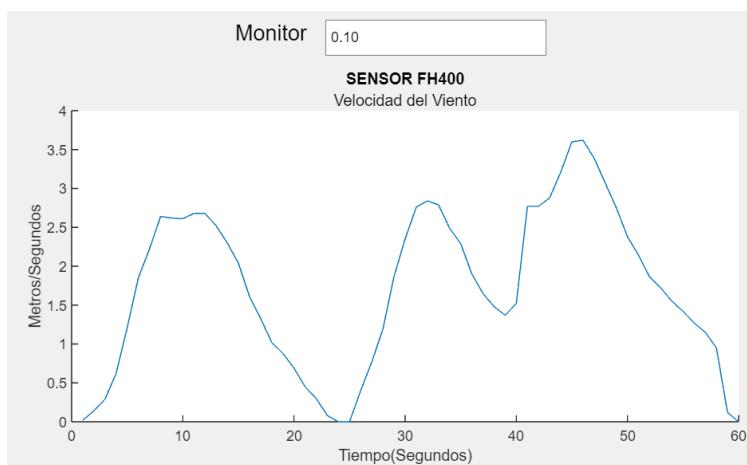


Figura 78: Velocidad del viento en sensor FH400 Fuente: propia



Figura 79: Instalación FH400

Fuente: Propia

Vista final del arreglo de tuberías de la parte superior e inferior del túnel de viento conectadas a sus intercambiadores de calor.



Figura 80: Instalación Parte Superior.

Fuente: Propia



Figura 81: Instalación Parte Inferior

Fuente: Propia

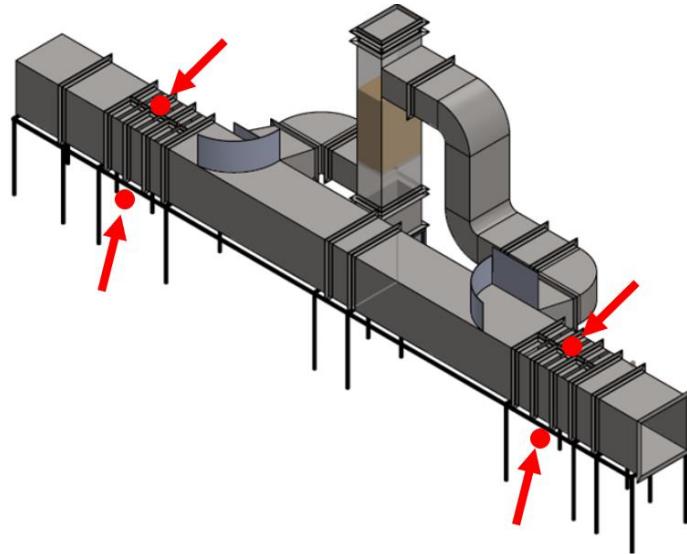


Figura 82: Ubicaciones de instalación

Fuente: Propia

4.1.3 VARIADOR DE FRECUENCIA, MOTOR TRIFASICO Y VENTILADOR

El control para el variador de frecuencia puede controlar la velocidad del motor trifásico casi en su totalidad. El variador de frecuencia puede reproducir una señal mínima para el motor de 5hz y la una señal máxima de 60hz como especifica el motor trifásico.

Para poder utilizar el ventilador primero se desmontó la base y se colocó al ducto de aire. Se hicieron varias modificaciones por ejemplo cortar 4mm las patas de la base que sostiene el ventilador y se cortó los acoplos que permiten encajar el ventilador al ducto.



Figura 83: Ventilador ubicación anterior

Fuente: Propia



Figura 84: Conexión de ventilador con túnel de viento

Fuente: Propia



Figura 85: Ventilador ubicación actual

Fuente: Propia

4.1.4 SENSOR DE TEMPERATURA DS18B20

El sensor al utilizar una resolución de medición de 12 bits tiene un tiempo de respuesta de 720 milisegundos. El sensor de temperatura dentro del rango de 10°C a 85°C puede llegar a tener un error de medición de hasta 0.05%.

El sensor fue sometido dentro de un vaso de agua “fría” y dentro de los 20 segundo que se muestran en la figura 86, los segundos siguientes el sensor fue sometido a la medición a lado de una cafetera donde se muestra como incrementa la temperatura gradualmente.

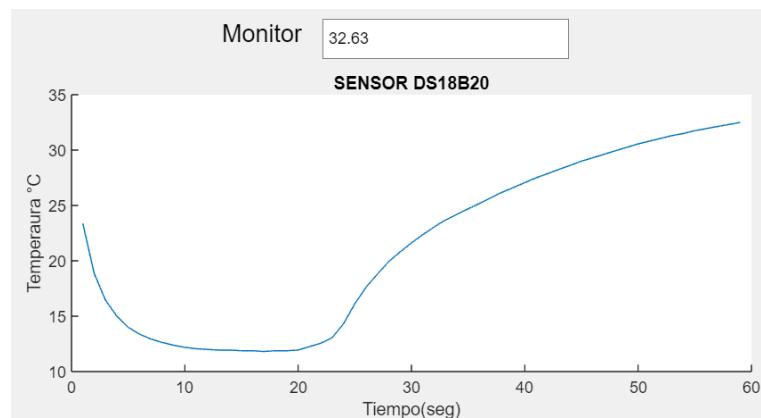


Figura 86: Grafica medición sensor DS18B20 Fuente: propia

4.1.5 IMPLEMENTACION DEL CIRCUITO GENERAL

Es el resultado de la integración de todos los sensores, módulos y conexión necesarias que hizo para hacer funcionar la programación.

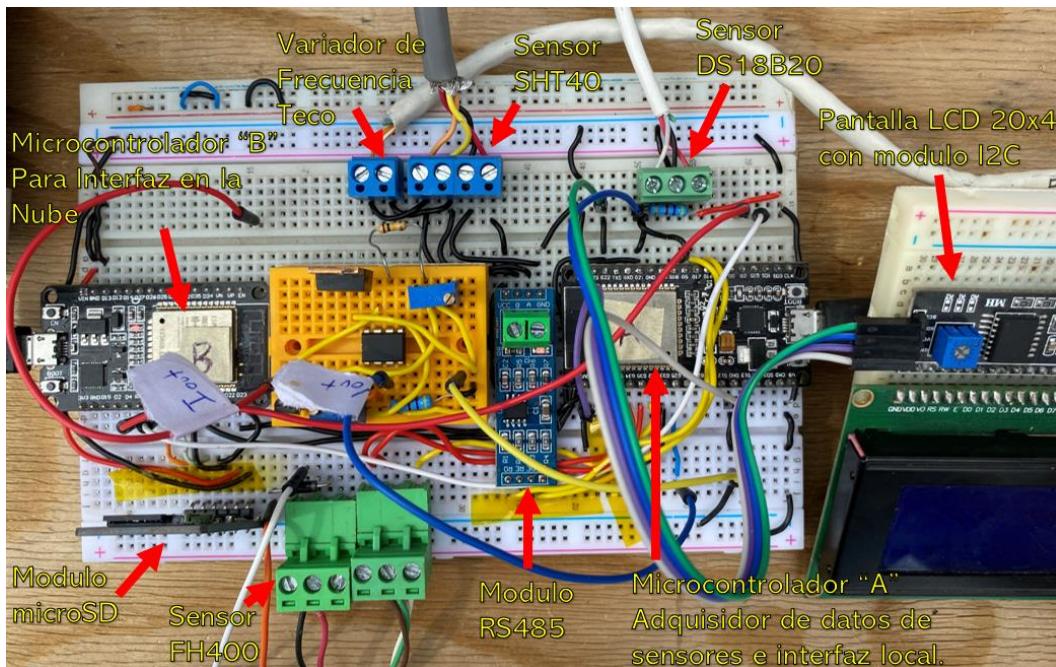
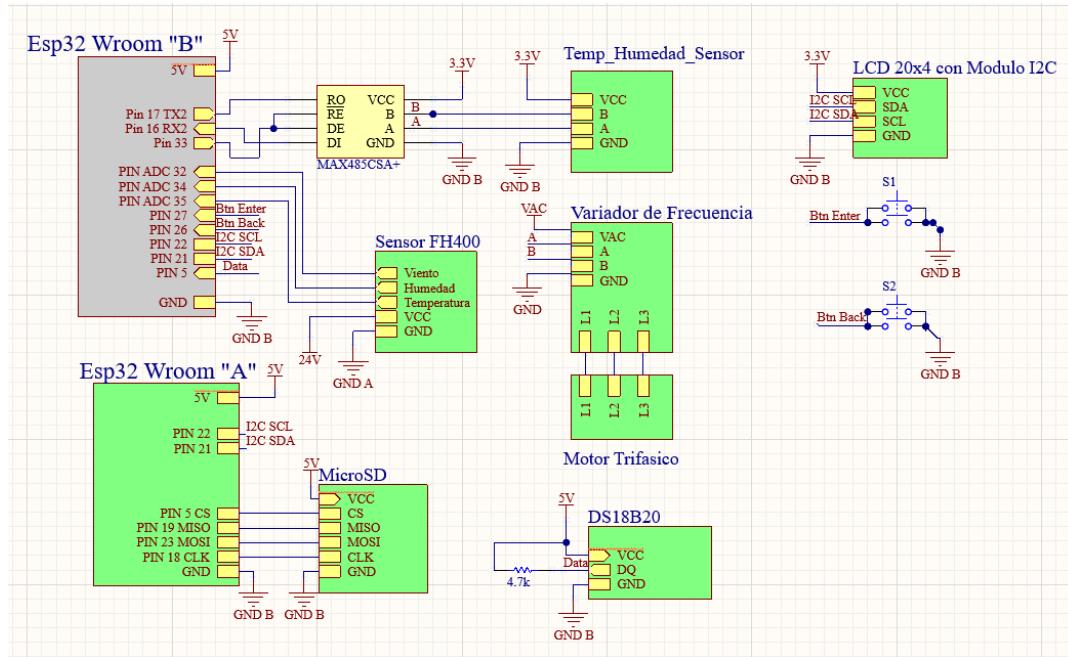


Figura 87: Implementación y circuito

Fuente: propia

4.1.6 INTERFAZ DE USUARIO EN LA NUBE

La interfaz corre en la red Wifi de edificio, estoy quiere decir con solo se puede acceder a la interfaz al estar conectado a la misma red wifi en el que microcontrolador esté conectado. La ruta o enlace para acceder es IP dinámica otorgada por el router ocupado y para llegar a conocerla se debe acceder al puerto serial del microcontrolador al estar conectado al pc únicamente una vez. Las siguientes ocasiones ya no será necesario realizar el procedimiento anterior ya que mantendrá la misma IP.

The figure consists of two vertically stacked screenshots of a web-based user interface. Both screenshots feature a header with the 'cimav' logo on the left and a blue 'Extra' link on the right. The main title in both cases is: **Sistema Solar de generación de aire sintético para aplicaciones de secado de productos agroindustriales**. Below the title, there is a section labeled **Opciones a emular:** which contains a dropdown menu. In the first screenshot, the dropdown menu has a single option: **Selecciona una opción**. Below the dropdown is a black rectangular button with the word **Mostrar** in white. In the second screenshot, the dropdown menu is open, showing three options: **Selecciona una opción**, **Sinaloa**, and **Yucatán**.

Figura 88: Vista previa de la interfaz desarrollada en HTML, CSS Y JAVASCRIPT. Fuente: propia



[Home](#) [Extra](#)

Sinaloa a Emular Escuinapa

Datos Pronosticados:

Clima: T: °C HR: %

Fuente: Meteonorm

Datos Emulados:

Clima: T: °C HR: %

Presionar el botón para emular:

Comenzar

Figura 89: Pagina Web Sinaloa

Fuente: propia

5 CONCLUSION

Los resultados del diseño de instrumentación se cumplieron como el objetivo general, pero hubo objetivos específicos donde no se pudo cumplir en su totalidad.

Los sensores y actuadores se instalaron y probaron demostrando un correcto funcionamiento actuando en conjunto. El diseño de un sistema de control digital para generar el nivel de variables climáticas deseadas no se alcanzó a completar por falta de tiempo durante el periodo de tiempo permitido durante la residencia profesional además de material faltante que impidió el avance del proyecto como se esperaba.

Se desarrollaron interfaces de usuario para utilizar los sensores, actuadores y utilizar el sistema de control digital como parte del producto final a entregar, pero a causa de falta de tiempo en el periodo de la residencia no se concluyó satisfactoriamente.

Como experiencia profesional adquirida vale la pena mencionar lo referente a sistemas embebidos como lo son microcontroladores, donde gran parte de proyecto se trabajó con ellos y vivió como desarrollar un flujo de trabajo y eficiente para lograr completar los objetivos relacionados con la programación y sistemas embebidos. Así como aumentar todo el conocimiento referente de sensores y actuadores utilizados en el proyecto.

6 REFERENCIAS

Arnabat Idoia. (13 de Noviembre de 2020). *Calderas de condensación ¿Cómo funcionan y qué ventajas tienen?* Obtenido de CALORYFRIO: <https://www.caloryfrio.com/calefaccion/calderas/funcionamiento-calderas-de-condensacion.html>

Asanza Victor. (Agosto de 2021). *REAL-TIME CLOCK RTC INTERNO*. Obtenido de vasanza.blogspot: <https://vasanza.blogspot.com/2021/08/esp32-real-time-clock-rtc-interno.html?m=1>

AWS. (Enero de 2023). *¿Qué es JavaScript?* Obtenido de aws: <https://aws.amazon.com/es/what-is/javascript/>

Captura-el. (Junio de 2008). *Qué es Altium Designer.* Obtenido de Redeweb: https://www.redeweb.com/_txt/643/26.pdf

Carmenate José Guerra. (2021). *ESP32 Wifi y Bluetooth en un solo chip*. Obtenido de programarfacil: <https://programarfacil.com/esp8266/esp32/>

CIMAV Unidad Durango. (2023). *Diseño y construcción de un sistema de acondicionamiento de aire ambiental, para la validación experimental de sistemas sustentables de para productos diversos*. Durango: Cimav. Recuperado el 6 de 6 de 2023

Circuitschool. (11 de Enero de 2022). *What is ESP32, how it works and what you can do with ESP32?* Obtenido de Circuitschool: https://www.tutorialspoint.com/esp32_for_iot/esp32_for_iot_introduction.htm

Componentes Electrónicas,. (2023). *¿Qué es MATLAB?* Obtenido de Componentes Electrónicas: <https://www.compelect.com.co/que-es-matlab/>

Córdova Diego Hinojosa. (21 de Octubre de 2021). *PlatformIO: Introducción*. Obtenido de linkedin: <https://www.linkedin.com/pulse/platformio-introducci%C3%B3n-diego-hinojosa-c%C3%B3rdova/?originalSubdomain=es>

DEGREE CONTROLS, I. (2022). *FH-Series Probe Style with Humidity*. Obtenido de degreeC: <https://www.degrec.com/products/embedded-airflow-sensors-switches/fh-series-probe-air-velocity-temperature-and-humidity-sensors/>

Diego Santos. (Enero de 2023). *Introducción al CSS: qué es, para qué sirve y otras 10 preguntas frecuentes*. Obtenido de Hubspot: <https://blog.hubspot.es/website/que-es-css>

Editorial Etecé. (2023). *HTTP*. Obtenido de concepto: <https://concepto.de/http/>

Enel Spa. (2023). *Módulo fotovoltaico*. Recuperado el 6 de 6 de 2023, de enel Green Power: <https://www.enelgreenpower.com/es/learning-hub/energias-renovables/energia-solar/modulo-fotovoltaico>

Espressif Systems (Shanghai) Co., Ltd. (Junio de 2016). *SPI Master Driver*. Obtenido de docs espressif: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/spi_master.html

Espressif Systems (Shanghai) Co., Ltd. (2016). *Universal Asynchronous Receiver/Transmitter (UART)*. Obtenido de espressif: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/uart.html>

Espressif Systems. (s.f.). *ESP32 WROOM 32 Datasheet v3.4*. Obtenido de Espressif: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

Flores Frankier. (22 de Julio de 2022). *Qué es Visual Studio Code y qué ventajas ofrece*. Obtenido de OpenWebinars: <https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/>

García J. - Rodríguez, J. (2020). *MATLAB: guía de aprendizaje*. Jorge Sarmiento Editor - Universitas. Recuperado el 6 de Junio de 2023, de Editor - Universitas. <https://elibro.net/es/lc/itdurango/titulos/181970>

HOBO. (2023). *onsetcomp*. Recuperado el 5 de 6 de 2023, de Estación de monitoreo remoto RX3000: <https://www.onsetcomp.com/products/data-loggers/rx3000>

IDEAM. (Marzo de 2023). *RADIACIÓN SOLAR*. Obtenido de IDEAM: <http://www.ideam.gov.co/web/tiempo-y-clima/radiacion-solar-ultravioleta>

Instituto Geológico y Minero de España. (2023). *¿Qué es una bomba de agua?* Recuperado el 4 de 6 de 2023, de igme: https://www.igme.es/ZonalInfantil/MateDivul/guia_didactica/pdf_carteles/carte14/CARTEL%204_4-4.pdf

José Antonio Marques Pereira, D. M. (1991). *Principios de secado de granos psicometria higroscoopia*. Santiago, Chile.

Juan, Danahé San. (14 de Mayo de 2015). *¿Para qué sirve la psicrometría en HVAC?* Obtenido de 0grados: <https://0grados.com/para-que-sirve-la-psicrometria-en-hvac/>

Kinsta. (19 de Diciembre de 2022). *¿Qué Es el Web Scraping? Cómo Extraer Legalmente el Contenido de la Web*. Obtenido de Kinsta: <https://kinsta.com/es/base-de-conocimiento/que-es-web-scraping/>

Krall César. (Junio de 2023). *¿Qué es y para qué sirve Ajax? Ventajas e inconvenientes. JavaScript asíncrono, XML y JSON.* Obtenido de aprenderaprogramar: https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=882:que-es-y-para-que-sirve-ajax-ventajas-e-inconvenientes-javascript-asincrono-xml-y-json-cu01193e&catid=78&Itemid=206

Llamas Luis. (27 de Junio de 2016). *Medir temperatura de líquidos y gases con Arduino y DS18B20*. Obtenido de LuisLlamas: <https://www.luisllamas.es/temperatura-liquidos-arduino-ds18b20/>

Logicbus. (2023). *¿Qué es Modbus?* Obtenido de Logicbus: <https://www.logicbus.com.mx/Modbus.php>

López Quijado, J. (2014). *Domine PHP y MySQL* (2a. ed.). RA-MA Editorial. Recuperado el 8 de 06 de 2023, de <https://elibro.net/es/lc/itduran/go/titulos/106410>

Menchaca García, F. R. (2020). *Fundamentos de programación en Lenguaje C*. Instituto Politécnico Nacional. Obtenido de https://www2.eii.uva.es/fund_inf/cpp/temas/1_introduccion/introduccion.html

National Instrument. (Junio de 2023). *¿Qué es Multisim?* Obtenido de Ni: <https://www.ni.com/es-mx/shop/electronic-test-instrumentation/application-software-for-electronic-test-and-instrumentation-category/what-is-multisim.html>

ORACLE. (Junio de 2022). *¿Qué es el IoT?* Obtenido de ORACLE: <https://www.oracle.com/mx/internet-of-things/what-is-iot/>

Orós Cabello, J. C. (2014). *Diseño de páginas Web con XHTML, JavaScript y CSS*. Recuperado el 6 de Junio de 2023, de elibro: <https://elibro.net/es/lc/itduran/go/titulos/106414>

Pérez Rodríguez, M. D. (2018). *HTML 4.0* (2a. ed). Recuperado el 6 de JUNIO de 2023, de elibro: <https://elibro.net/es/lc/itduran/go/titulos/225267>

Pini Art. (7 de Abril de 2020). *Por qué el Bus de circuitos inter-integrados (I2C) hace que la conexión de los CI sea tan fácil y cómo utilizarlo.* Recuperado el 5 de Junio de 2023, de DigiKey: <https://www.digikey.com.mx/es/articles/why-the-inter-integrated-circuit-bus-makes-connecting-ics-so-easy>

Planas Oriol. (28 de Septiembre de 2015). *¿Qué es un colector solar térmico? Características y tipos.* Obtenido de solar-energia.net: <https://solar-energia.net/energia-solar-termica/componentes/colector-solar-termico>

PR Electronics. (2023). *Los fundamentos de los lazos de corriente de 4...20 mA.* Obtenido de preelectronics: <https://www.preelectronics.com/es/los-fundamentos-de-los-lazos-de-corriente-de-4-20-ma/#:~:text=La%20corriente%20de%204...,valores%20del%20proceso%20al%20controlador.>

R. José Luis. (2023). *Cómo funciona un termotanque.* Recuperado el 5 de 6 de 2023, de Como funciona: <https://como-funciona.co/un-termotanque/>

RandomNerd. (2023). *ESP32 I2C Communication: Set Pins, Multiple Bus Interfaces and Peripherals (Arduino IDE).* Obtenido de RandomNerdTutorials: <https://randomnerdtutorials.com/esp32-i2c-communication-arduino-ide/>

Rodiles López, D. J. (09 de Septiembre de 2020). *Secado en la industria de alimentos.* Obtenido de tecnoAgro: <https://tecnoagro.com.mx/no.-143/secado-en-la-industria-de-alimentos>

S&P. (2 de Noviembre de 2020). *solerpalau.* Obtenido de ¿Qué es el diagrama psicrométrico?: <https://www.solerpalau.com/es-es/blog/diagrama-psicrometrico/>

Schwarz Rohde. (12 de Mayo de 2023). *Entendiendo el UART.* Obtenido de Rohde Schwarz: https://www.rohde-schwarz.com/lat/productos/prueba-y-medicion/essentials-test-equipment/digital-oscilloscopes/entendiendo-el-uart_254524.html

Sensirion. (Febrero de 2023). *SHT40.* Obtenido de SENSIRION: <https://sensirion.com/products/catalog/SHT40>

SolerPalau. (7 de Enero de 2020). *¿Qué es y para qué sirve un variador de frecuencia?* Obtenido de SolerPalau: <https://www.solerpalau.com/es-es/blog/variador-de-frecuencia/>

Teja Ravi. (12 de Marzo de 2021). *ESP32 ADC Tutorial | How to use ADC in ESP32?* Obtenido de ElectronicsHub: <https://www.electronicshub.org/esp32-adc-tutorial/>

Webempresa. (Junio de 2023). *¿Qué es un servidor Web y para qué sirve?* Obtenido de Webempresa: <https://www.webempresa.com/hosting/que-es-servidor-web.html>

Wies Olga. (20 de Octubre de 2021). *Guía de la comunicación RS485.* Obtenido de Electronic Team, Inc.: <https://www.elitima.com/es/article/rs485-communication-guide/>

Wikipedia. (10 de Noviembre de 2022). *Verificación de redundancia cíclica*. Obtenido de Wikipedia:

https://es.wikipedia.org/wiki/Verificaci%C3%B3n_de_redundancia_c%C3%ADclica

7 ANEXOS

7.1.1 CODIGO LIBRERÍA SHT40.H.

```
1. #ifndef _SHT40_H
2. #define _SHT40_H
3. #include <Arduino.h>
4.
5. class SHT40
6. {
7.
8. public:
9.     const uint8_t adrSensor_x33 = 0x33;
10.    const uint8_t adrSensor_x34 = 0x34;
11.    const uint8_t adrSensor_x37 = 0x37;
12.    const uint8_t adrSensor_x36 = 0x36;
13.    const uint8_t adrSensor_x35 = 0x35;
14.    const uint8_t adrSensor_x31 = 0x31;
15.    const uint8_t adrSensor_x32 = 0x32;
16.
17.    struct Sensado
18.    { // estructura para almacenar los valores de temperatura y
humedad
19.        float FT;
20.        float FRH;
21.    };
22.
23.    struct Sensado dataSensor31 = {0, 0}, dataSensor32 = {0, 0},
dataSensor33 = {0, 0}, dataSensor34 = {0, 0}, dataSensor35 = {0, 0},
dataSensor36 = {0, 0}, dataSensor37 = {0, 0};
24.
25.    SHT40(); // constructor
26.    ~SHT40(); // destructor
27.    void setup(uint8_t Pinn);
28.    void ResetBus();
29.    void SendGetSerial(uint8_t Nsensor); // retorna dos valores
30.
31.    uint8_t getDataSensores(uint8_t Nsensor);
32.    float FTx = 0, FRHx = 0;
33.    uint8_t _EnTxPinn; // pin de habilitacion de transmision
34.
35. private:
36.    void LeerSerial(uint8_t Nsensor);
37.    void Formulas();
38.    void impresion_trama();
39.    void CRC();
40.    unsigned char crcF, crcF2;
41.
42. protected:
43.    int16_t Temp1 = 0, Hum1 = 0, Temp2 = 0, Hum2 = 0, HCRC = 0,
TCRC = 0;
44.    uint8_t Nvar;
45.    float ST = 0, SRH = 0;
46.    byte buffer[6]; // buffer para almacenar los datos recibidos
47. }
```

```
48.  
49.     #endif // _SHT40_H
```

7.1.2 CODIGO SHT40.CPP

```
1. #include <SHT40.h>  
2. #include <freertos/task.h>  
3. #include <HardwareSerial.h>  
4.  
5.  
6. SHT40::SHT40() // de la clase sht40 existe un metodo SHT40  
7. {  
8. }  
9.  
10.    SHT40::~SHT40() // Destructor  
11.    {  
12.    }  
13.  
14.    void SHT40::setup(uint8_t Pinn) // configuracion EnTxPin  
15.    {  
16.        _EnTxPinn = Pinn;  
17.        pinMode(Pinn, OUTPUT); //esta instruccion tambien esta  
    inicializada en la libreria TECO_L510.h  
18.    }  
19.  
20.  
21.    void SHT40::SendGetSerial(uint8_t Nsensor) //Retornas Temp y Hum  
22.    { // Valor en Hexadecimal  
23.        digitalWrite(_EnTxPinn, HIGH);  
24.        ResetBus();  
25.        vTaskDelay(pdMS_TO_TICKS(50));  
26.        Serial2.write(0x55);  
27.        Serial2.write(Nsensor); // Nsensor 0x32, 0x33, 0x34, 0x35  
28.        Serial2.write(0xF1);  
29.        Serial2.flush(); // esperamos a que se envien los datos  
30.        Serial.print("Sht40: Addr. Sensor: 0x");  
31.        Serial.println(Nsensor, HEX);  
32.        LeerSerial(Nsensor);  
33.        getDataSensores(Nsensor);  
34.  
35.    }  
36.  
37.  
38.    void SHT40::LeerSerial(uint8_t Nsensor)  
39.    {  
40.        bool TrueCRC1 = false;  
41.        bool TrueCRC2 = false;  
42.        digitalWrite(_EnTxPinn, LOW); // RS485 como receptor  
43.        vTaskDelay(pdMS_TO_TICKS(25));  
44.  
45.        if (Serial2.available() > 0)  
46.        {  
47.            for (int i = 0; i < 6; i++)  
48.            {  
49.                buffer[i] = Serial2.read();  
50.                vTaskDelay(pdMS_TO_TICKS(25));
```

```

51.        }
52.        //impresion_trama();
53.        TCRC = buffer[2];
54.        HCRC = buffer[5];
55.        CRC();
56.        if (TCRC == crcF) {TrueCRC1 = true; } //Serial.print("-TCRC:");
57.        Serial.println(String(TrueCRC1));
58.        if (HCRC == crcF2) {TrueCRC2 = true; } //Serial.print("-HCRC:");
59.        Serial.println(String(TrueCRC2));
60.        Formulas();
61.    }
62.    Serial.println("Sht40: No hay datos recibidos en
63. 0x");Serial.print(Nsensor, HEX);
64. }
65.
66. void SHT40::ResetBus()
67. {
68.     Serial2.write(0x66);
69.     Serial2.write(0x66);
70.     Serial2.write(0x66);
71. }
72.
73. void SHT40::CRC()
74. {
75.     // crc calculator
76.     // comparar los valores del crc calculado con el que envian si
77.     // no es correcto volver a mandar la solicitud de temperatura
78.     // hasta que sea correcto
79.     unsigned char crc = 0xFF;
80.     int i=0;
81.     bool t_control = false;
82.     for (int k = 0; k < 2; k++) // se debe repetir dos veces, uno
83.         para el bytes de temperatura y otros para humedad
84.     {
85.         if (t_control == false)
86.         {
87.             for ( i = 0; i < 2; i++)
88.             {
89.                 crc ^= buffer[i];
90.                 for (int j = 0; j < 8; j++)
91.                 {
92.                     if ((crc & 0x80) != 0)
93.                     {
94.                         crc = (unsigned char)((crc << 1) ^ 0x31);
95.                     }
96.                 else
97.                 {
98.                     crc <= 1;
99.                 }
100.            if( i == 1){ // i = 1
101.                crcF= crc;
102.                //Serial.print("1CRC= ");

```

```

103.          //Serial.println(crcF, HEX);
104.          t_control = true;
105.      }
106.  }
107.
108. }
109. else // true
110. {
111.     crc = 0xFF;
112.     for ( i = 3; i < 5; i++)
113.     {
114.         crc ^= buffer[i];
115.         for (int j = 0; j < 8; j++)
116.         {
117.             if ((crc & 0x80) != 0)
118.             {
119.                 crc = (unsigned char) ((crc << 1) ^ 0x31);
120.             }
121.             else
122.             {
123.                 crc <= 1;
124.             }
125.         }
126.         if( i == 4){ // i = 4
127.             crcF2= crc;
128.             //Serial.print("4CRC= ");
129.             //Serial.println(crcF2, HEX);
130.         }
131.     }
132. }
133. }
134. }
135.
136. void SHT40::Formulas()
137.
138.     Temp1 = buffer[0];
139.     Temp2 = buffer[1];
140.     Hum1 = buffer[3];
141.     Hum2 = buffer[4];
142.
143.     ST = (Temp1 * 256) + Temp2;
144.     SRH = (Hum1 * 256) + Hum2;
145.
146.     float ft2 = ST / 65535;
147.
148.     FTx = -45 + (175 * ft2);
149.     FRHx = (-6 + 125 * (SRH / 65535));
150.
151.     //Serial.print("Sht40 lib : Temperatura: ");
152.     //Serial.print( FTx);
153.     //Serial.println("°C ");
154.     //Serial.print("Sht40 lib: Humedad: ");
155.     //Serial.print(FRHx);
156.     //Serial.println(" %RH ");
157. }
158.
159. uint8_t SHT40::getDataSensores( uint8_t Nsensor)

```

```

160.      {
161.          if(Nsensor == adrSensor_x31){ dataSensor31.FT = FTx; dataSenso
r31.FRH = FRHx; return 1;}
162.          else if (Nsensor == adrSensor_x32){ dataSensor32.FT = FTx; data
Sensor32.FRH = FRHx; return 1;}
163.          else if (Nsensor == adrSensor_x33){ dataSensor33.FT = FTx; data
Sensor33.FRH = FRHx; return 1;}
164.          else if (Nsensor == adrSensor_x34){ dataSensor34.FT = FTx; data
Sensor34.FRH = FRHx; return 1;}
165.          else if (Nsensor == adrSensor_x35){ dataSensor35.FT = FTx; data
Sensor35.FRH = FRHx; return 1;}
166.          else if (Nsensor == adrSensor_x36){ dataSensor36.FT = FTx; data
Sensor36.FRH = FRHx; return 1;}
167.          else if (Nsensor == adrSensor_x37){ dataSensor37.FT = FTx; data
Sensor37.FRH = FRHx; return 1;}
168.          else { // se metio otro parametro o no sucedio return 0;
169.              return 0;
170.          }
171.      }
172.
173.
174.
175.
176.      void SHT40::impresion_trama()
177.      {
178.          Serial.print("Sht40: Trama Recibida: ");
179.          Serial.print(buffer[0], HEX);
180.          Serial.print(" ");
181.          Serial.print(buffer[1], HEX);
182.          Serial.print(" ");
183.          Serial.print(buffer[2], HEX);
184.          Serial.print(" ");
185.          Serial.print(buffer[3], HEX);
186.          Serial.print(" ");
187.          Serial.print(buffer[4], HEX);
188.          Serial.print(" ");
189.          Serial.println(buffer[5], HEX);
190.      }

```

7.1.3 CODIGO LIBRERÍA VARIADOR DE FRECUENCIA TECO_L510.H

```

1. #ifndef _TECO_L510_H
2. #define _TECO_L510_H
3. #include <Arduino.h>
4.
5. class TECO_L510
6. {
7.     public:
8.
9.
10.        TECO_L510(); // constructor
11.        ~TECO_L510(); // destructor
12.        void setup( uint8_t Pinn ); //Config Pines Input
13.        void runVariador();
14.        void stopVariador();
15.        uint8_t set_freq_variador( uint16_t freq );

```

```

16.         //uint16_t last_bytes_freq = 0;
17.
18.     private:
19.         uint16_t calculateCRC16(uint8_t *data, size_t length);
20.         void recepcion_impression_trama();
21.         void recepcion_frecuencia_trama();
22.         uint8_t comparacion_recepcion_trama_freq_crc16();
23.         uint8_t _EnTxPinn; //21
24.
25.     protected:
26.         uint16_t crc;
27.
28.         int var1, var2;// c1 = 0;
29.         byte buffer[8], bufferF[8];
30.         uint16_t vtiempo_delay = 80; // delay necesario para que
funciones el variador en el Esp32
31.     };
32. #endif // _TECO_L510_H

```

7.1.4 CODIGO LIBRERÍA VARIADOR DE FRECUENCIA TECO_L510.H

```

1. #include <TECO_L510.h>
2.
3.
4. TECO_L510::TECO_L510()
5. {
6. }
7.
8. TECO_L510::~TECO_L510()
9. {
10. }
11.
12. void TECO_L510::setup( uint8_t Pinn )
13. { // pin de habilitacion de transmision
14.     pinMode(Pinn, OUTPUT);
15.     _EnTxPinn = Pinn;
16. }
17.
18. void TECO_L510::runVariador()
19. {
20.     // Run Forward Command
21.     // 01 06 25 01 00 01 12 C6
22.     digitalWrite(_EnTxPinn, HIGH);
23.
24.     Serial2.write(0x01); // ID
25.     Serial2.write(0x06); // Fuction Code
26.     Serial2.write(0x25); // Starting Address #1 MSB
27.     Serial2.write(0x01); // Starting Address #2 LSB error en 2
28.     Serial2.write(0x00); // Send data #1 MSB
29.     Serial2.write(0x01); // Send data #2 LSB
30.     Serial2.write(0x12); // crc #1
31.     Serial2.write(0xC6); // crc #2
32.     Serial2.flush();
33.     vTaskDelay(pdMS_TO_TICKS(vtiempo_delay));
34.     digitalWrite(_EnTxPinn, LOW); // RS485 como receptor
35.     recepcion_impression_trama();

```

```

36.    }
37.
38.    void TECO_L510::stopVariador()
39.    {
40.        // Stop Command
41.        // 01 06 25 01 00 00 D3 06
42.        digitalWrite(_EnTxPinn, HIGH); // rs modo transmisor
43.        Serial2.write(0x01); // ID
44.        Serial2.write(0x06); // Fuction Code
45.        Serial2.write(0x25); // Starting Address #1 MSB
46.        Serial2.write(0x01); // Starting Address #2 LSB
47.        Serial2.write(0x00); // Send data #1 MSB
48.        Serial2.write(0x00); // Send data #2 LSB
49.        Serial2.write(0xD3); // crc #1
50.        Serial2.write(0x06); // crc #2
51.        Serial2.flush();
52.        vTaskDelay(pdMS_TO_TICKS(vtiempo_delay));
53.        digitalWrite(_EnTxPinn, LOW); // rs modo receptor
54.        recepcion_impresion_trama();
55.    }
56.
57.    uint8_t TECO_L510::set_freq_variador(uint16_t freq) // 
58.    { // funcion para establecer consigna de frecuencia de forma
      manual y codigo
59.        // Maxima frecuencia motor con variador es de 60.00hz donde el
      variador los representa como 6000
60.        uint16_t last_freq =0;
61.        if ( freq <= 7000)
62.        {
63.            uint8_t buffer_variable[] = {0x01, 0x06, 0x25, 0x02,
      highByte(freq), lowByte(freq)}; // ejemplo: toma 0xAB y 0xCD de 0xABCD
64.            size_t len = sizeof(buffer_variable) / sizeof(buffer_variable[0]
      );
65.            uint16_t crc = calculateCRC16(buffer_variable, len);
66.
67.            digitalWrite(_EnTxPinn, HIGH); // rs modo transmisor
68.            Serial2.write(0x01); // ID
69.            Serial2.write(0x06); // Fuction Code
70.            Serial2.write(0x25); // Starting Address #1 MSB
71.            Serial2.write(0x02); // Starting Address #2 LSB
72.            Serial2.write(highByte(freq)); // Send data #1 MSB
73.            Serial2.write(lowByte(freq)); // Send data #2 LSB
74.            Serial2.write(lowByte(crc)); // crc #2 // para el variador se
      debe invertir el orden (A little endian)
75.            Serial2.write(highByte(crc)); // crc #1
76.            Serial2.flush();
77.
78.            vTaskDelay(pdMS_TO_TICKS(vtiempo_delay));
79.            digitalWrite(_EnTxPinn, LOW); // rs modo receptor
80.            recepcion_frecuencia_trama();
81.
82.
83.            return 1; // si se pudo enviar la trama
84.        }else{
85.            return 0; //no se pudo enviar la trama
86.        }
87.    }

```

```

88.
89.
90.
91.     uint16_t TECO_L510::calculateCRC16(uint8_t *data, size_t length)
92.     {
93.         uint16_t crc = 0xFFFF;
94.         uint16_t i;
95.         //0xA001 Modbus Polinomio
96.         for (i = 0; i < length; i++) {
97.             crc ^= data[i];
98.             for (int j = 0; j < 8; j++) {
99.                 if (crc & 0x0001) {
100.                     crc = (crc >> 1) ^ 0xA001;
101.                 } else {
102.                     crc >>= 1;
103.                 }
104.             }
105.         }
106.
107.         return crc;
108.     }
109.
110.    void TECO_L510::reception_impression_trama() // Si hubo recepcion
      de datos entonces hubo un error en el envio o trama
111.    {
112.        if ( Serial2.available() > 0){
113.            Serial.println("Teco: Error en Trama: ");
114.            for (int i = 0; i < 8; i++)
115.            {
116.                buffer[i] = Serial2.read();
117.                delay(25);
118.            }
119.            Serial.println(" ");
120.            Serial.print(buffer[0], HEX); // ID
121.            Serial.print(" ");
122.            Serial.print(buffer[1], HEX); // Fuction Code
123.            Serial.print(" ");
124.            Serial.print(buffer[2], HEX); // Starting Address #1 MSB
125.            Serial.print(" ");
126.            Serial.print(buffer[3], HEX); // Starting Address #2 LSB
127.            Serial.print(" ");
128.            Serial.print(buffer[4], HEX); // Send data #1 MSB
129.            Serial.print(" ");
130.            Serial.print(buffer[5], HEX); // Send data #2 LSB
131.            Serial.print(" ");
132.            Serial.print(buffer[6], HEX); // crc #2 // para el variador
      se debe invertir el orden (A little endian)
133.            Serial.print(" ");
134.            Serial.println(buffer[7], HEX); // crc #1
135.
136.            if ( comparacion_recepcion_trama_freq_crc16() == 0 ){
137.                Serial.println("Teco: Error en CRC16, reenviando
      trama..falta agregar funcion");
138.                //conversion_frecuencia_variable();
139.            }
140.
141.        }else

```

```

142.      {
143.          // No hubo recepcion de datos entonces la trama fue enviada
144.          // correctamente
145.      }
146.
147.      void TECO_L510::recepccion_frecuencia_trama()
148.      {
149.          //buffer exclusivo para el valor de la frecuencia variable
150.          if ( Serial2.available() > 0){
151.              Serial.println("Teco: Error recepcion en Trama (): ");
152.              for (int i = 0; i < 9; i++)
153.              {
154.                  buffer[i] = Serial2.read();
155.                  delay(25);
156.              }
157.              Serial.print(bufferF[0], HEX); // ID
158.              Serial.print(" ");
159.              Serial.print(bufferF[1], HEX); // Fuction Code
160.              Serial.print(" ");
161.              Serial.print(bufferF[2], HEX); // Starting Address #1 MSB
162.              Serial.print(" ");
163.              Serial.print(bufferF[3], HEX); // Starting Address #2 LSB
164.              Serial.print(" ");
165.              Serial.print(bufferF[4], HEX); // Send data #1 MSB
166.              Serial.print(" ");
167.              Serial.print(bufferF[5], HEX); // Send data #2 LSB
168.              Serial.print(" ");
169.              Serial.print(bufferF[6], HEX); // crc #2    // para el variador
        se debe invertir el orden (A little endian)
170.              Serial.print(" ");
171.              Serial.println(bufferF[7], HEX); // crc #1
172.
173.              if ( comparacion_recepccion_trama_freq_crc16() == 0 ){
174.                  Serial.println("Teco: Error en CRC16, reenviando
        trama..falta agregar funcion x2");
175.                  //conversion_frecuencia_variable();
176.              }
177.
178.          }
179.          else{
180.              // No hubo recepcion de datos entonces la trama fue enviada
181.              // correctamente
182.          }
183.
184.          uint8_t TECO_L510::comparacion_recepccion_trama_freq_crc16()
185.          {
186.              uint8_t crc16_ok;
187.              uint8_t buffer_variable[] = {bufferF[0], bufferF[1],bufferF[2],
        bufferF[3], bufferF[4], bufferF[5]}; // ejemplo: toma 0xAB y 0xCD de
        0xABCD
188.              size_t len = sizeof(buffer_variable) / sizeof(buffer_variable[0
        ]);
189.              uint16_t crc = calculateCRC16(buffer_variable, len);
190.              if ( (lowByte(crc) << 8 | highByte(crc)) == (bufferF[6] << 8 |
        bufferF[7] ) ){// inversion a little endian

```

```

191.         //Serial.println("CRC16 OK");
192.         crc16_ok = 1;
193.     }else {
194.         //Serial.println("CRC16 ERROR");
195.         crc16_ok = 0;
196.     }
197.     return  crc16_ok;
198.
199. }
```

7.1.5 CÓDIGO LIBRERIA FH400.H

```

1. #ifndef _FH400_H
2. #define _FH400_H
3. #include <Arduino.h>
4.
5. class FH400
6. {
7.
8. public:
9.     struct sensor_fh400
10.    {
11.        float speed;
12.        float temp;
13.        float hum;
14.    };
15.    sensor_fh400 datafh400;
16.
17.    FH400();
// Constructor
18.    ~FH400();
// Destructor
19.
20.    void setup(uint8_t pinBlanco, uint8_t pinVerde, uint8_t pinCafe); // Config Pines Input
21.    float conversion_velocidad(); // devuelve en m/s
22.    float filtro_viento(); // adc con filtro
23.    float conversion_temp();
24.    float conversion_hum(); // devuelve la humedad
25.
26. private:
27.     uint8_t _pinBlanco; // pines genericos para adc su sensor
28.     uint8_t _pinVerde;
29.     uint8_t _pinCafe;
30.
31. protected:
32.     int adc_filtrado = 0; // Filtro Media Móvil como Pasa Bajos
33.     int adc_raw = 0; // Filtro Media Móvil como Pasa Bajos
34.     float alpha = 0.50; // aumentar el valor para disminuir el filtro y aumentar la velocidad en cambios de mediciones
35.     float blanco_velocidad = 0; // Valor de referencia para velocidad
```

```

35.         float verde_temperatura = 0; // Valor de referencia para
   temperatura
36.         float cafe_humedad = 0;
37.         const uint8_t vdcMax = 10; // Output Voltage at 100°C should
   be 10.0V.
38.         float velocidad_resultado; // variable global necesaria para
   la compensacion
39.     };
40.
41. #endif // _FH400_H

```

7.1.6 CÓDIGO LIBRERIA FH400.CPP

```

1. #include <FH400.h>
2.
3. FH400::FH400()
4. {
5. }
6.
7. FH400::~FH400()
8. {
9. }
10.
11.     void FH400::setup(uint8_t pinBlanco, uint8_t pinVerde, uint8_t pi
   nCafe)
12.     {
13.         pinMode(pinBlanco, INPUT);
14.         pinMode(pinVerde, INPUT);
15.         pinMode(pinCafe, INPUT);
16.         _pinBlanco = pinBlanco;
17.         _pinVerde = pinVerde;
18.         _pinCafe = pinCafe;
19.         // analogSetWidth(12);                                // 12 bits de
   resolucion (ya que el adc es de 12 bits)
20.         // analogSetPinAttenuation(_pinBlanco, ADC_11db); //2.5dB
   attenuation (ADC_ATTEN_DB_2_5) gives full-scale voltage 1.5V
21.
22.         // 6dB attenuation (ADC_ATTEN_DB_6) gives full-scale voltage
   2.2V
23.         // VDD de --volts
24.     }
25.
26.     float FH400::conversion_velocidad()
27.     {
28.         /* Application airflow range is 100 - 2000 fpm.
29.            Output voltage at 1000 fpm is 5.0V.
30.            At airflow velocities exceeding 1000 fpm, the output voltage
   should not exceed 5.0V.
31.            Checar data sheet
32.            */
33.         const uint16_t velocity_high_range = 10160; // mm/s
34.         const uint8_t muestreo = 20;                // cantidad de
   muestras a tomar para promediar
35.         uint16_t aux2;
36.         float aux = 0;
37.

```

```

38.         for (uint8_t i = 0; i < muestreo; i++)
39.         {
40.             aux2 = analogRead(_pinBlanco);
41.             aux = aux + ((aux2 * velocity_high_range) / (vdcMax * 1000));
42.         }
43.         blanco_velocidad = aux / muestreo; // division para promediar
44.
45.         velocidad_resultado = (float)blanco_velocidad / 1000; //
cambiamos la escala a m/s
46.         Serial.printf("fh400:sin Filtro Velocidad: %0.2f mm/s | %0.2f
m/s\n", blanco_velocidad, velocidad_resultado);
47.         return velocidad_resultado;
48.     }
49.
50.     float FH400::filtro_viento()
51.     {
52.         //Filtro Media Móvil como Pasa Bajos
53.         //An=a*M+(1-a)*An
54.         //alpha 1: Sin filtro
55.         //alpha 0: Filtrado totalmente
56.         //alpha clásico 0.05
57.         const uint16_t velocity_high_range = 10160; // mm/s
58.
59.
60.         adc_raw = analogRead(_pinBlanco);
61.         adc_filtrado = (alpha*adc_raw) + ((1-alpha)*adc_filtrado);
62.
63.         /* Serial.println();
64.         Serial.print(adc_raw);
65.         Serial.print(",");
66.         Serial.println(adc_filtrado); */
67.
68.         float velocidad_resultado = ((adc_filtrado * velocity_high_ra
nge) / (vdcMax * 1000));
69.         //Serial.printf("lib fh400:con filtro Velocidad: %0.2f mm/s | %
0.2f m/s \n", velocidad_resultado, velocidad_resultado/ 1000);
70.
71.
72.         return velocidad_resultado / 1000;
73.     }
74.
75.     float FH400::conversion_temp()
76.     { // Function to convert the voltage to temperature
77.
78.         const uint8_t CentrigradosMax = 100; // Application temperature
range is 0 - 100 °C
79.         const uint8_t muestreo = 5;           // cantidad de datos a tomar
para promediar
80.
81.         float aux, aux2;
82.         for (uint8_t i = 0; i < muestreo; i++)
83.         {
84.
85.             aux2 = analogRead(_pinVerde);
86.             aux = aux + ((aux2 * CentrigradosMax) / vdcMax);
87.
88.             // compensacion segun datasheet

```

```

89.         /* if (velocidad_resultado > 0.5)
90.         {
91.             aux = aux + (((aux2 * CentrigradosMax) / vdcMax) + 1);
92.         }
93.         else
94.         {
95.             aux = aux + (((aux2 * CentrigradosMax) / vdcMax) + 2);
96.         } */
97.     }
98.     verde_temperatura = aux / muestreo;
99.
100.    // compensacion necesario, segun datasheet
101.    float temperatura_resultado = verde_temperatura / 1000; // cambiamos us escala
102.    // Serial.printf("fh400: Temperatura: %f \n",
103.        temperatura_resultado);
104.    return temperatura_resultado;
105. }
106. float FH400::conversion_hum()
107. {
108.     const uint8_t humedad_range_high = 100; // porcentaje de humedad
109.     const uint8_t muestreo = 100;           // cantidad de datos a tomar para promediar */
110.
111.     float aux;
112.     for (uint8_t i = 0; i < muestreo; i++)
113.     {
114.         // voltage = analogRead(_pinCafe) * (3.3 / 4095.0);
115.
116.         aux = aux + (analogRead(_pinCafe) * humedad_range_high) / vdcMax;
117.         // Serial.printf("fh400: hum [%d]aux:%f \n", i,aux);
118.     }
119.     cafe_humedad = aux / muestreo;
120.
121.     float humedad_resultado = cafe_humedad / 1000;
122.     // Serial.printf("fh400: Humedad:%f \n", humedad_resultado);
123.     return humedad_resultado;
124. }
```

7.1.7 CÓDIGO INTERFAZ DE USUARIO LOCAL

```

1. #include <Arduino.h>
2. #include "LiquidCrystal_I2C.h"
3. LiquidCrystal_I2C lcd(0x27, 20, 4);
4. #include <freertos/task.h>
5. #include <HardwareSerial.h>
6. #include <OneWire.h>
7. #include <DallasTemperature.h>
8. const int oneWirePin = 5; // PIN PARA SENSOR DE TEMPERATURA DS18B20
9. OneWire oneWireBus(oneWirePin);
10. DallasTemperature sensor(&oneWireBus);
```

```

11.      DeviceAddress
12.      sensorTuberia = {0x28, 0xF8, 0x95, 0x4B, 0xB1, 0x21, 0x09, 0x52}; // 
13.      // direccion del sensor de temperatura
14.      #include <SHT40.h>      // libreria propia para el sensor SHT40
15.      SHT40 sht40;           // objeto para el sensor SHT40
16.      #include <TECO_L510.h> // libreria propia para el variador de
17.      // frecuencia TECO L510
18.      TECO_L510 teco;
19.      #include <FH400.h> // libreria propia para el sensor FH400
20.      FH400 fh400;
21.
22.      #define EnTxPinn 33 // pin de habilitacion de transmision para
23.      // rs485
24.      bool flag_clear = true;
25.      bool flag2_clear = true;
26.      bool flag_var = false;
27.
28.      // pin para adc fh400
29.      #define pinBlanco 32 // pin Velocidad fh400
30.      #define pinVerde 35 // pin Temperatura fh400
31.      #define pinCafe 34 // pin Humedad fh400
32.
33.      // BOTON PARA LCD
34.      #define btn_enter 27 // sin usar
35.      #define btn_back 26
36.      void interrupcion_enter();
37.      void interrupcion_back();
38.
39.      ulong prev_time, current_time, prev_time2;
40.      const uint16_t dt1 = 4000;
41.      const uint16_t dt2 = 9000;
42.      const uint16_t dt3 = 400;
43.      int conteo = 0;
44.
45.      float ponderadoTemp = 0; // variables para ponderar
46.      // temperatura y humedad
47.      float ponderadoHumedad = 0; // variables para ponderar
48.      // temperatura y humedad
49.      byte sendData[18]; // buffer para enviar y recibir
50.      // tramas
51.      byte sendData2[14]; // buffer para enviar y recibir
52.      // tramas
53.      byte sendData3[50];
54.      byte data[18]; // buffer para enviar y recibir tramas
55.      uint8_t on_off_boton = 0;
56.      uint8_t on_off_boton_next = 0; // 
57.      // variables de control tramas y webserver
58.      uint8_t ubi_pagina = 0; // 
59.      // variables de control tramas y webserver
60.      uint8_t ubi_pagina_next = 0; // 
61.      // variables de control tramas y webserver

```

```

52.     float temperatura_estadoMx_setpoint = 0; // variables para recepcion de trama y para sistema de control
53.     float humedad_estadoMx_setpoint = 0; // variables para recepcion de trama y para sistema de control
54.     float termisor1 = 0; // trama
55.     uint8_t estadoMx = 0; // trama
56.     const int timeOut = 100; // tiempo de espera para recepcion de trama
57.     char ACK = 'F'; // caracter de confirmacion para trama
58.     char NAK = 'E'; // caracter de confirmacion para trama
59.     void errorRecepcion(); // funciones para recepcion de trama
60.     void okRecepcion(); // funciones para recepcion de trama
61.     int ProccesSerialData(const int timeOut, byte *buffer, const uint8_t bufferLength, void (*okCallBack)(), void (*errorCallBack)()); // funcion de control para recibir o enviar trama
62.     void recibir_trama(); // enum SerialState // tipo de dato para recepcion de trama
63.     {
64.         OKEY,
65.         ERROR,
66.         NO_RESPONSE
67.     };
68.
69.
70.     float temp_emulada = 0; // variable para sistema de control
71.     float hum_emulada = 0;
72.     float temp_emulada_anterior = 0; // variable para sistema de control
73.     float hum_emulada_anterior = 0;
74.     void promedio_ponderado(float &ptemperatura_general, float &phumedad_general); // funcion para ponderar temperatura y humedad
75.     void sensores_actuadores(); // funcion para mostrar los sensores y actuadores en uart
76.     void tramaSensoresToServer(); // funcion que se encarga de armar la trama de los sensores a servidor
77.     void seleccion_estado(); // muestra el nombre del estado de mexico en el lcd segun la variable de control estadoMx creo
78.     void mostrarClimaEmular(String estadoMx); // funcion para mostrar el clima emulado con base a la trama recibida

```

```

79.      void mostrarSensores();
80.      void fromLongToBytes(byte *bytes, long ing);
81.      void fromFloatToBytes(byte *bytes, float f); // trama
    para enviar los datos de los sensores al servidor
82.      void fragmentar_trama_recibida(byte buffer[]);
    // funcion para fragmentar la trama de clima pronosticada por la base de
    datos del servidor
83.      int ProcesACK(const int timeOut, void (*okCallBack)()); // para
    validar la recepcion de trama
84.      int TryGetACK(int TimeOut);
85.      void i2c_requestFrom();
86.      void _i2cTramaControlToServer(); // para validar la recepcion de
    trama
87.      void setup()
88.      {
89.
90.          // i2c Maestro
91.          Serial.begin(9600);
92.          Serial2.begin(9600, SERIAL_8N1, 16, 17); // 16 es RX y 17 es TX
93.          pinMode(btn_back, INPUT_PULLUP); // no los uso
94.          pinMode(btn_enter, INPUT_PULLUP); // no los uso
95.          attachInterrupt(digitalPinToInterruption(btn_enter),
    interrupcion_enter, RISING);
96.          attachInterrupt(digitalPinToInterruption(btn_back),
    interrupcion_back, RISING);
97.          fh400.setup(pinBlanco, pinVerde, pinCafe); // Config Pines
    Input Adc
98.          teco.setup(EnTxPinn); // Config Pines
    Input
99.          sht40.setup(EnTxPinn); // Config Pines
    Input
100.         sensor.begin(); // inicializar
    sensor de temperatura DS18B20
101.         lcd.init();
102.         lcd.backlight();
103.         // pinMode(EnTx.Pin, OUTPUT); // pin de habilitacion de
    transmision
104.
105.         Serial.printf("\n Esp32 Control para Emulacion de Variables
    Climaticas \n\n");
106.     }
107.
108.     void loop()
109.     {
110.         // recibir_trama(); //USAR PARA UART
111.         fh400.datafh400.speed = fh400.filtro_viento();
112.
113.         seleccion_estado();
114.
115.         if (millis() - prev_time2 > 1000)
116.         {
117.             i2c_requestFrom();
118.             _i2cTramaControlToServer();
119.             prev_time2 = millis();
120.         }
121.     }

```

```

122.     void i2c_requestFrom()
123.     {
124.         Wire.requestFrom(20, 18);
125.         while (Wire.available())
126.         {
127.             {
128.                 for (int i = 0; i < 18; i++)
129.                 {
130.                     data[i] = Wire.read(); // Leer cada byte recibido y
131.                     almacenarlo en el arreglo
132.                 }
133.             Serial.printf("\n\t I2C TRAMA recibida \n ");
134.             fragmentar_trama_recibida(data);
135.         }
136.         void _i2cTramaControlToServer()
137.         {
138.             byte seleccion_pagina[4];
139.             byte boton[4];
140.             byte temp[4];
141.             byte hum[4];
142.
143.             // hacer la desfragmentacion de bytes necesaria para enviar la
144.             trama
145.             fromLongToBytes(seleccion_pagina, ubi_pagina);
146.             fromLongToBytes(boton, on_off_boton);
147.             fromFloatToBytes(temp, ponderadoTemp);
148.             fromFloatToBytes(hum, ponderadoHumedad);
149.             sendData[0] = 'I'; // ascii 73
150.             sendData[1] = seleccion_pagina[0]; // para saber en que pagina
151.             se encuentra el usuario
152.             sendData[2] = seleccion_pagina[1];
153.             sendData[3] = seleccion_pagina[2];
154.             sendData[4] = seleccion_pagina[3];
155.             sendData[5] = boton[0]; // para saber si el boton esta
156.             encendido o apagado
157.             sendData[6] = boton[1];
158.             sendData[7] = boton[2];
159.             sendData[8] = boton[3];
160.             sendData[9] = temp[0];
161.             sendData[10] = temp[1];
162.             sendData[11] = temp[2];
163.             sendData[12] = temp[3];
164.
165.             sendData[13] = hum[0];
166.             sendData[14] = hum[1];
167.             sendData[15] = hum[2];
168.             sendData[16] = hum[3];
169.             sendData[17] = ACK; // ascii 70
170.             sendData[18] = '\n'; // ascii 10
171.
172.             // impresion();

```

```

173.      Wire.beginTransmission(20); // Comenzar a comunicarse con
    esclavo #23
174.      Wire.write(sendData, 18);
175.      Wire.endTransmission(); // Finalizar comunicación
176.  }
177.  ///////////////////////////////////////////////////////////////////
178. void seleccion_estado()
179. {
180.     String nombreEstadoMx;
181.     if (ubi_pagina == 1)
182.     {
183.         nombreEstadoMx = "Sinaloa:";  

184.         mostrarClimaEmular(nombreEstadoMx); // muestra el clima en
    lcd y comienza a emular (sistema de control)
185.     }
186.     else if (ubi_pagina == 2)
187.     {
188.         nombreEstadoMx = "Yucatan:";  

189.         mostrarClimaEmular(nombreEstadoMx); // muestra el clima en
    lcd y comienza a emular (sistema de control)
190.     }
191.     else if (ubi_pagina == 3)
192.     {
193.         nombreEstadoMx = "Otro:";  

194.         mostrarClimaEmular(nombreEstadoMx); // muestra el clima en
    lcd y comienza a emular (sistema de control)
195.     }
196.     else
197.     {
198.         if (flag_clear == true)
199.             { // para solo limpiar una sola vez y que no se explote la
    lcd
200.                 lcd.clear();
201.                 flag_clear = false;
202.             }
203.             lcd.setCursor(7, 1);
204.             lcd.print("CIMAV");
205.         }
206.     }
207.
208. void recibir_trama()
209. {
210.     digitalWrite(EnTxPinn, LOW); // habilitar recepcion de trama
211.     if (Serial2.available() > 0)
212.     {
213.         byte buffer[18];
214.         size_t n = Serial2.readBytesUntil('\n', buffer, 18); //  

    devuelve el numero de bytes leidos
215.         for (int8_t i = 0; i < n; i++)
216.         {
217.             Serial.print(buffer[i]);
218.             Serial.print(" ");
219.         }
220.         if (n == 18)
221.         {
222.

```

```

223.          ProcesSerialData(timeOut, buffer, 18, okRepcion,
224.            errorRepcion);
225.        }
226.      }
227.
228.      ///////////////////////////////////////////////////
229.      int TryGetSerialData(int TimeOut, byte *buffer) // encontrar ACK
230.        en el buffer y saber como proceder
231.        {
232.          // intenta conseguir los delimitadores de la trama para
233.          // verificar que la trama se recibió correctamente dentro de un tiempo
234.          // de 100ms, si no se recibe la trama en ese tiempo, se
235.          // devuelve un error
236.          // si se recibe la trama, se devuelve un OK.
237.          // el Ok sirve para proceder a fragmentar la trama
238.          /* unsigned long StartTime = millis();
239.          while (!Serial2.available() && (millis() - StartTime) <
240.            TimeOut)
241.            {
242.              */
243.              if (buffer[0] == 'I' /* && buffer[17] == ACK */)
244.                return OKEY;
245.              else
246.                return ERROR;
247.            }
248.
249.            int rst = TryGetSerialData(timeOut, buffer);
250.            if (rst == OKEY)
251.            {
252.              Serial.print(ACK);
253.              if (okCallBack != NULL)
254.              {
255.                okCallBack(); // al comprobar ACK envia ACK por UART para
256.                // avisar correcta recepcion de datos y comienza a fragmentar la trama
257.                fragmentar_trama_recibida(buffer);
258.                Serial.printf(" ACK enviado \n");
259.              }
260.              else if (rst == ERROR)
261.              {
262.                if (okCallBack != NULL)
263.                {
264.                  errorCallBack(); // al comprobar que no esta
265.                  // ACK envia NAK por UART para avisar incorrecta recepcion de datos
266.                  digitalWrite(EnTxPinn, LOW); // deshabilitar recepcion de
267.                  // trama
268.                  Serial.printf("NAK enviado \n");

```

```

267.         }
268.     }
269.     return rst;
270. }
271.
272. void okRecepcion()
273. {
274.
275.     digitalWrite(EnTxPinn, HIGH); // deshabilitar recepcion de
      trama
276.     Serial2.write(ACK);
277.     Serial2.write('\n');
278.     Serial.printf("Recepcion correcta\n");
279. }
280.
281. void errorRecepcion()
282. {
283.     digitalWrite(EnTxPinn, HIGH); // deshabilitar recepcion de
      trama
284.     Serial2.write(NAK);
285.     Serial2.write('\n');
286.     Serial.printf("Recepcion incorrecta \n");
287. }
288.
289. void mostrarClimaEmular(String nombreEstadoMx)
290. { // funcion importante. Aqui comienza para tener la opcion de
      emular( sistema de control)
291.
292.     if (flag_clear == true)
293.     { // para solo limpiar una sola vez y que no se explote la lcd
294.         lcd.clear();
295.         flag_clear = false;
296.     }
297.
298.     lcd.setCursor(0, 0);
299.     lcd.print(nombreEstadoMx);
300.
301.     lcd.setCursor(3, 1);
302.     lcd.print("Tem.P:");
303.     lcd.print(temperatura_estadoMx_setpoint);
304.     lcd.print("C");
305.     lcd.setCursor(3, 2);
306.     lcd.print("Hum.P:");
307.     lcd.print(humedad_estadoMx_setpoint);
308.     lcd.print("%");
309.     lcd.setCursor(2, 3);
310.     lcd.print("Listo Para Emular");
311.
312.     if (on_off_boton == 0)
313.     {
314.         vTaskDelay(pdMS_TO_TICKS(50));
315.         teco.stopVariador();
316.     }
317.
318.     while (on_off_boton == 1) // comenzar sistema de control
319.     {

```

```

320.          // Mostrando datos de la temperatura y humedad pronosticada
321.          float tempX = temperatura_estadoMx_setpoint;
322.          float humX = humedad_estadoMx_setpoint;
323.
324.          if (flag_clear == false)
325.          { // para solo limpiar una sola vez y que no se explote la
326.              lcd
327.                  lcd.clear();
328.                  flag_clear = true;
329.
330.                  lcd.setCursor(0, 0);
331.                  lcd.print(nombreEstadoMx);
332.
333.                  lcd.setCursor(0, 1);
334.                  lcd.print("T.P:");
335.                  lcd.setCursor(4, 1);
336.                  lcd.print(tempX);
337.                  lcd.setCursor(9, 1);
338.                  lcd.print("C");
339.                  lcd.setCursor(11, 1);
340.                  lcd.print("H.P:");
341.                  lcd.print(humX);
342.                  lcd.setCursor(19, 1);
343.                  lcd.print("%");
344.          // Mostrando datos de la temperatura y humedad emulada
345.          lcd.setCursor(0, 3);
346.          lcd.print("T.E:");
347.          lcd.setCursor(4, 3);
348.          lcd.print(ponderadoTemp); // para mostrar la temperatura
            emulada
349.          lcd.setCursor(9, 3);
350.          lcd.print("C");
351.          lcd.setCursor(11, 3);
352.          lcd.print("H.E:");
353.          lcd.print(ponderadoHumedad); // para mostrar la humedad
            emulada
354.          lcd.setCursor(19, 3);
355.          lcd.print("%");
356.
357.          /////////////////////////////////
358.          // ACCION DE CONTROL AQUI
359.          sensores_actuadores(); // Ejemplo leer todos los sensores y
            actuadores
360.          /////////////////////////////////
361.
362.          recibir_trama(); // para usar con uart
363.          i2c_requestFrom();
364.          _i2cTramaControlToServer();
365.
366.          vTaskDelay(pdMS_TO_TICKS(1100));
367.      }
368.  }
369. void interrupcion_enter(
370. {

```

```

371.      // para entrar y salir a while para adquirir y procesar las
372.      // variables sensadas
373.      // auxiliar mientras no se haya corregido la comunicacion uart
374.      // entre esp32
375.      on_off_boton++;
376.      Serial.printf("+1 on_off_boton: %d \n", on_off_boton);
377.      if (on_off_boton == 3)
378.      {
379.          on_off_boton = 1;
380.      }
381.  void interrupcion_back()
382.  {
383.      ubi_pagina++;
384.      Serial.printf("+1 ubi_pagina: %d \n", ubi_pagina);
385.      if (ubi_pagina == 3)
386.      {
387.          ubi_pagina = 1;
388.      }
389.  }
390.  void fromFloatToBytes(byte *bytes, float f)
391.  {
392.      int length = sizeof(float);
393.      for (int i = 0; i < length; i++)
394.          bytes[i] = ((byte *)&f)[i];
395.  }
396.  void fromLongToBytes(byte *bytes, long ing)
397.  {
398.      bytes[0] = (byte)((ing & 0xff000000) >> 24);
399.      bytes[1] = (byte)((ing & 0x00ff0000) >> 16);
400.      bytes[2] = (byte)((ing & 0x0000ff00) >> 8);
401.      bytes[3] = (byte)((ing & 0x000000ff));
402.  }
403.
404.  unsigned long getUlong(byte packet[], byte i)
405.  {
406.      // big endian
407.      unsigned long value = 0;
408.      value = (value * 256) + packet[i];
409.      value = (value * 256) + packet[i + 1];
410.      value = (value * 256) + packet[i + 2];
411.      value = (value * 256) + packet[i + 3];
412.      return value;
413.  }
414.
415.  unsigned int getInt(byte packet[], byte i)
416.  {
417.      unsigned int value = 0;
418.      value = (value * 256) + packet[i];
419.      return value;
420.  }
421.
422.  float getFloat(byte packet[], byte i)
423.  {
424.      union tag

```

```

425.      {
426.          byte bin[4];
427.          float num;
428.      } u;
429.
430.      u.bin[0] = packet[i];
431.      u.bin[1] = packet[i + 1];
432.      u.bin[2] = packet[i + 2];
433.      u.bin[3] = packet[i + 3];
434.      return u.num;
435.  }
436.
437. void fragmentar_trama_recibida(byte buffer[])
438. {
439.     ubi_pagina = getUlong(buffer, 1);
440.     Serial.printf("\n ubi_pagina: %d \n", ubi_pagina);
441.
442.     on_off_boton = getUlong(buffer, 5);
443.     Serial.printf("boton en : %d \n", on_off_boton);
444.
445.     temperatura_estadoMx_setpoint = getFloat(buffer, 9);
446.     Serial.printf("temp recibida: %0.2f \n",
447.         temperatura_estadoMx_setpoint);
448.     humedad_estadoMx_setpoint = getFloat(buffer, 13);
449.     Serial.printf("hum recibida: %0.2f \n",
450.         humedad_estadoMx_setpoint);
451. }
452. void tramaSensoresToServer()
453. {
454.
455.     byte temp1[4];
456.     byte hum1[4];
457.     byte temp2[4];
458.     byte hum2[4];
459.     byte temp3[4];
460.     byte hum3[4];
461.     byte temp4[4];
462.     byte hum4[4];
463.     byte temp5[4];
464.     byte hum5[4];
465.     byte termi[4];
466.     byte viento[4];
467.     /* byte tempfh[4];
468.     byte humfh[4]; */
469.
470.     fromFloatToBytes(temp1, 30.20 /* sht40.dataSensor32.FT */);
471.     fromFloatToBytes(hum1, 30.10 /* sht40.dataSensor32.FRH */);
472.     fromFloatToBytes(temp2, 20.01 /* sht40.dataSensor33.FT */);
473.     fromFloatToBytes(hum2, 20.02 /* sht40.dataSensor33.FRH */);
474.     fromFloatToBytes(temp3, 40.11 /* sht40.dataSensor34.FT */);
475.     fromFloatToBytes(hum3, 40.12 /* sht40.dataSensor34.FRH */);
476.     fromFloatToBytes(temp4, 50.00 /* sht40.dataSensor35.FT */);
477.     fromFloatToBytes(hum4, 50.01 /* sht40.dataSensor35.FRH */);
478.     fromFloatToBytes(temp5, 09.01 /* sht40.dataSensor36.FT */);

```

```

479.     fromFloatToBytes (hum5, 09.02 /* sht40.dataSensor36.FRH */);
480.     fromFloatToBytes (termi, 88.7); // CORRECCION PARA DPESUES
481.     fromFloatToBytes (viento, 11.11 /* fh400.datafh400.speed */);
482.     /* fromFloatToBytes (tempfh, fh400.datafh400.temp ) */
483.     /* fromFloatToBytes (humfh, fh400.datafh400.hum ) */
484.
485.     sendData3[0] = 'I'; // ascii 73
486.     sendData3[1] = temp1[0];
487.     sendData3[2] = temp1[1];
488.     sendData3[3] = temp1[2];
489.     sendData3[4] = temp1[3];
490.
491.     sendData3[5] = hum1[0];
492.     sendData3[6] = hum1[1];
493.     sendData3[7] = hum1[2];
494.     sendData3[8] = hum1[3];
495.
496.     sendData3[9] = temp2[0];
497.     sendData3[10] = temp2[1];
498.     sendData3[11] = temp2[2];
499.     sendData3[12] = temp2[3];
500.
501.     sendData3[13] = hum2[0];
502.     sendData3[14] = hum2[1];
503.     sendData3[15] = hum2[2];
504.     sendData3[16] = hum2[3];
505.
506.     sendData3[17] = temp3[0];
507.     sendData3[18] = temp3[1];
508.     sendData3[19] = temp3[2];
509.     sendData3[20] = temp3[3];
510.
511.     sendData3[21] = hum3[0];
512.     sendData3[22] = hum3[1];
513.     sendData3[23] = hum3[2];
514.     sendData3[24] = hum3[3];
515.
516.     sendData3[25] = temp4[0];
517.     sendData3[26] = temp4[1];
518.     sendData3[27] = temp4[2];
519.     sendData3[28] = temp4[3];
520.
521.     sendData3[29] = hum4[0];
522.     sendData3[30] = hum4[1];
523.     sendData3[31] = hum4[2];
524.     sendData3[32] = hum4[3];
525.
526.     sendData3[33] = temp5[0];
527.     sendData3[34] = temp5[1];
528.     sendData3[35] = temp5[2];
529.     sendData3[36] = temp5[3];
530.
531.     sendData3[37] = hum5[0];
532.     sendData3[38] = hum5[1];
533.     sendData3[39] = hum5[2];
534.     sendData3[40] = hum5[3];

```

```

535.         sendData3[41] = termi[0];
536.         sendData3[42] = termi[1];
537.         sendData3[43] = termi[2];
538.         sendData3[44] = termi[3];
539.
540.         sendData3[45] = viento[0];
541.         sendData3[46] = viento[1];
542.         sendData3[47] = viento[2];
543.         sendData3[48] = viento[3];
544.         /*
545.             sendData3[49] = tempfh[0];
546.             sendData3[50] = tempfh[1];
547.             sendData3[51] = tempfh[2];
548.             sendData3[52] = tempfh[3];
549.
550.             sendData3[53] = humfh[0];
551.             sendData3[54] = humfh[1];
552.             sendData3[55] = humfh[2];
553.             sendData3[56] = humfh[3]; */
554.
555.         sendData3[49] = ACK; // ascii 70
556.         sendData3[50] = '\n'; // ascii 10
557.
558.
559.         Serial2.write(sendData3, 50);
560.         Serial.printf("Trama sensores enviada ");
561.     }
562.
563.     int ProccesACK(const int timeOut, void (*okCallBack)())
564.     {
565.
566.         int rst = TryGetACK(timeOut);
567.         if (rst == OKEY)
568.         {
569.             if (okCallBack != NULL)
570.                 okCallBack(); // se comprueba si la función de devolución
de llamada okCallBack no es un puntero nulo
571.             }
572.             else if (rst == ERROR)
573.             {
574.                 if (okCallBack != NULL)
575.                     // errorCallBack();
576.                     tramaSensoresToServer();
577.                     Serial.printf("Reenviado Trama por error en recepcion\n");
578.                 }
579.                 return rst;
580.             }
581.
582.             int TryGetACK(int TimeOut)
583.             {
584.                 unsigned long StartTime = millis();
585.                 digitalWrite(EnTxPinn, LOW); // habilita el envio de datos
586.                 while (!Serial.available() && (millis() - StartTime) < TimeOut)
587.                 {
588.                 }
589.             }

```

```

590.         if (Serial2.available())
591.         {
592.             if (Serial2.read() == ACK)
593.                 return OKEY;
594.             if (Serial2.read() == NAK)
595.                 return ERROR;
596.         }
597.         return NO_RESPONSE;
598.     }
599.
600.     void okAction()
601.     {
602.         Serial.printf("\n ACK recibido por el otro esp32 \n");
603.     }
604.     void mostrarSensores()
605.     {
606.
607.         if (flag_clear == true)
608.         { // para solo limpiar una sola vez y que no se explote la lcd
609.             lcd.clear();
610.             flag_clear = false;
611.             flag2_clear = false;
612.         }
613.
614.         lcd.setCursor(0, 0);
615.         lcd.print("Sensores Actuadores");
616.
617.         lcd.setCursor(2, 1);
618.         lcd.print("Enviando data");
619.
620.         if (millis() - prev_time > 1200)
621.         {
622.             tramaSensoresToServer();
623.             ProccesACK(timeOut, okAction);
624.             prev_time = millis();
625.         }
626.     }
627.
628.     void tramaEmulada_control()
629.     { // trama para enviar de los valores generados por el sistema de
       control de temperatura y humedad
630.
631.         byte temp_control[4];
632.         byte hum_control[4];
633.         byte var_extra[4];
634.         // byte var_extra2[4];
635.         fromFloatToBytes(temp_control, 30.20);
636.         fromFloatToBytes(hum_control, 30.10);
637.         fromFloatToBytes(var_extra, 20.01);
638.         // fromFloatToBytes(hum2, 20.02 );
639.         sendData2[0] = 'I'; // ascii 73
640.         sendData2[1] = temp_control[0];
641.         sendData2[2] = temp_control[1];
642.         sendData2[3] = temp_control[2];
643.         sendData2[4] = temp_control[3];
644.

```

```

645.         sendData2[5] = hum_control[0];
646.         sendData2[6] = hum_control[1];
647.         sendData2[7] = hum_control[2];
648.         sendData2[8] = hum_control[3];
649.
650.         sendData2[9] = var_extra[0];
651.         sendData2[10] = var_extra[1];
652.         sendData2[11] = var_extra[2];
653.         sendData2[12] = var_extra[3];
654.         sendData2[13] = ACK;
655.         sendData2[14] = '\n';
656.         Serial2.write(sendData2, 14);
657.     }
658.
659.     void sensores_actuadores() // esta funcion es para demostrar el
   funcionamiento. No es necesaria para el sistema de control
660.     {
661.         float viento = 0;
662.         float t33, h33;
663.         float t34, h34;
664.         float t35, h35;
665.         float t36, h36;
666.         float t37, h37;
667.         // mandamos peticion a los sensores para que nos envien los
   datos de temperatura y humedad
668.         /* sht40.SendGetSerial(sht40.adrSensor_x32);
669.            vTaskDelay(pdMS_TO_TICKS(50)); */
670.         sht40.SendGetSerial(sht40.adrSensor_x33);
671.         vTaskDelay(pdMS_TO_TICKS(50));
672.         t33 = sht40.dataSensor33.FT;
673.         h33 = sht40.dataSensor33.FRH;
674.         sht40.SendGetSerial(sht40.adrSensor_x34);
675.         vTaskDelay(pdMS_TO_TICKS(50));
676.         t34 = sht40.dataSensor34.FT;
677.         h34 = sht40.dataSensor34.FRH;
678.         sht40.SendGetSerial(sht40.adrSensor_x35);
679.         vTaskDelay(pdMS_TO_TICKS(50));
680.         t35 = sht40.dataSensor35.FT;
681.         h35 = sht40.dataSensor35.FRH;
682.         sht40.SendGetSerial(sht40.adrSensor_x36);
683.         vTaskDelay(pdMS_TO_TICKS(50));
684.         t36 = sht40.dataSensor36.FT;
685.         h36 = sht40.dataSensor36.FRH;
686.         sht40.SendGetSerial(sht40.adrSensor_x37);
687.         vTaskDelay(pdMS_TO_TICKS(50));
688.         t37 = sht40.dataSensor37.FT;
689.         h37 = sht40.dataSensor37.FRH;
690.
691.         // mostrar en puerto serial los datos de los sensores
692.         // Serial.printf("Temperatura addr x32: %0.2f , HR: %0.2f\n",
   sht40.dataSensor32.FT, sht40.dataSensor32.FRH);
693.         Serial.printf("Temperatura addr x33: %0.2f , HR: %0.2f\n", t33,
   h33);
694.         Serial.printf("Temperatura addr x34: %0.2f , HR: %0.2f\n", t34,
   h35);

```

```

695.      Serial.printf("Temperatura addr x35: %0.2f , HR: %0.2f\n", t35,
696.          h35);
697.      Serial.printf("Temperatura addr x36: %0.2f , HR: %0.2f\n", t36,
698.          h36);
699.      Serial.printf("Temperatura addr x37: %0.2f , HR: %0.2f\n", t37,
700.          h37);
701.      // promedio ponderado de temperatura y humedad
702.      promedio_ponderado(ponderadoTemp, ponderadoHumedad);
703.
704.      // Sensar velocidad del viento con sensor FH400
705.      fh400.datafh400.speed = fh400.filtro_viento();
706.      viento = fh400.conversion_velocidad();
707.      Serial.printf("Velocidad del viento: %0.2f\n",
708.          fh400.datafh400.speed);
709.      Serial.printf("Velocidad del viento2: %0.2f\n", viento);
710.
711.      // sensor temperatura DS18B20 de tuberia a intercambiador de
712.      calor
713.      sensor.requestTemperatures();
714.      float tuberia_temp = sensor.getTempC(sensorTuberia);
715.      Serial.printf("Temperatura tuberia: %0.2f\n", tuberia_temp);
716.
717.      // Variador de frecuencia aumentando cada segundo
718.      if (millis() - prev_time > 2000)
719.      {
720.          conteo = conteo + 100;
721.          teco.set_freq_variador(conteo);
722.          vTaskDelay(pdMS_TO_TICKS(50));
723.          teco.runVariador();
724.          Serial.printf("Variador a %d Hz\n", conteo / 100);
725.          prev_time = millis();
726.
727.          vTaskDelay(pdMS_TO_TICKS(1000));
728.      }
729.
730.      void promedio_ponderado(float &ptemperatura_general, float &phume
731.          dad_general)
732.      {
733.          // Promedio ponderado de temperatura y humedad
734.          // se usar referencias para retornar los valores de
735.          // temperatura y humedad
736.          // toma en cuenta cinco sensores
737.          float suma_pesos = 0;
738.          uint8_t num_sensores = 5;
739.
740.          struct SensorTemperatura
741.          {
742.              int sensor_id;
743.              float temperatura;
744.              float humedad;

```

```

743.         float peso; // es la importancia relativa que tiene cada
    sensor con respecto a donde se encuentran ubicados
744.     };
745.     SensorTemperatura sensores[num_sensores]; // arreglo de
    sensores
746.
747.     sensores[0].sensor_id = 1;
748.     sensores[0].temperatura = sht40.dataSensor33.FT;
749.     sensores[0].humedad = sht40.dataSensor33.FRH;
750.     sensores[0].peso = 0.5;
751.
752.     sensores[1].sensor_id = 2;
753.     sensores[1].temperatura = sht40.dataSensor34.FT;
754.     sensores[1].humedad = sht40.dataSensor34.FRH;
755.     sensores[1].peso = 0.5;
756.
757.     sensores[2].sensor_id = 3;
758.     sensores[2].temperatura = sht40.dataSensor37.FT;
759.     sensores[2].humedad = sht40.dataSensor37.FRH;
760.     sensores[2].peso = 0.5;
761.
762.     sensores[3].sensor_id = 4;
763.     sensores[3].temperatura = sht40.dataSensor36.FT;
764.     sensores[3].humedad = sht40.dataSensor36.FRH;
765.     sensores[3].peso = 0.5;
766.
767.     sensores[4].sensor_id = 5;
768.     sensores[4].temperatura = sht40.dataSensor35.FT;
769.     sensores[4].humedad = sht40.dataSensor35.FRH;
770.     sensores[4].peso = 0.5;
771.
772.     // calculo del promedio ponderado
773.     for (uint8_t i = 0; i < num_sensores; i++)
774.     {
775.
        ptemperatura_general = ptemperatura_general + (sensores[i].temperatura
        * sensores[i].peso);
776.
        phumedad_general = phumedad_general + (sensores[i].humedad * sensores[i]
        ].peso);
777.
        suma_pesos = suma_pesos + sensores[i].peso;
778.    }
779.    ptemperatura_general = ptemperatura_general / suma_pesos;
780.    phumedad_general = phumedad_general / suma_pesos;
781.    Serial.printf("Temperatura General: %f \n",
        ptemperatura_general);
782.    Serial.printf("Humedad General General: %f \n",
        phumedad_general);
783. }
784.
785. bool al_cambiarTrama_enviar()
786. { // al cambiar cualquier variable de control, temperatura o
    humedad enviar trama
787.
    if (on_off_boton != on_off_boton_next)
788.    {

```

```

790.         // _tramaServerToControl();
791.         on_off_boton_next = on_off_boton;
792.         ubi_pagina_next = ubi_pagina;
793.         return true;
794.     }
795.
796.     if ((temp_emulada != temp_emulada_anterior) || (hum_emulada != hum_emulada_anterior))
797.     {
798.         // _tramaServerToControl();
799.         temp_emulada_anterior = temp_emulada;
800.         hum_emulada_anterior = hum_emulada;
801.         return true;
802.     }
803.
804.     return false; // ninguno de los casos entonces no se ejecutara
la funcion de ProccesACK
805. }
```

7.1.8 CODIGO INTERFAZ DE USUARIO EN LA NUBE BACK-END

```

1. #include <Arduino.h>
2. #include <ESPAsyncWebServer.h>
3. AsyncWebServer server(80);
4. #include <ESPmDNS.h>
5. #include <SPIFFS.h>
6. #include <HardwareSerial.h>
7. #include <WiFi.h>
8. #include <SD.h>
9. File myFile;
10.    #include <SPI.h>
11.    #include <ESP32Time.h>
12.    ESP32Time rtc;
13.    #include <Wire.h>
14.    #include <freertos/task.h>
15.    #include <HTTPClient.h>
16.    #include <ArduinoJson.h>
17.    WiFiClient client;
18.    HTTPClient http;
19.
20.    String GetTemp_Pronos();
21.    String GetHum_Pronos();
22.    String GetTemp_Emular();
23.    String GetHum_Emular();
24.    void config_Wifi_mDns();
25.    void config_rtc();
26.    /// Configuracion wifi
27.    const char *ssid = "CIMAV-Visita";      // Enter SSID
28.    const char *password = "Investigacion"; // Enter Password
29.    const char *ssid2 = "x_xz";
30.    const char *password2 = "8831HGuadiana212@";
31.    IPAddress localIP(192, 168, 1, 68); // dirección IP
32.    IPAddress gateway(192, 168, 1, 254); // puerta de enlace
33.    IPAddress subnet(255, 255, 255, 0); // máscara de subred
34.
```

```

35.      // Configuracion Servidor NTP
36.      /* const char *ntpServer = "mx.pool.ntp.org";
37.      const char *ntpServer2 = "1.mx.pool.ntp.org";
38.      const long gmtOffset_sec = -21600; // offset en segundos GMT-6
   Durango Mexico
39.      const int daylightOffset_sec = 0; */
40.
41.      // Colas
42.      #define MAX_SIZE 24
43.      float queue_Temp[MAX_SIZE];
44.      int front1 = 0;
45.      int rear1 = -1;
46.      int itemCount1 = 0;
47.
48.      float queue_Hum[MAX_SIZE];
49.      int front2 = 0;
50.      int rear2 = -1;
51.      int itemCount2 = 0;
52.
53.      void enqueue_Temp(float value);
54.      float dequeue_Temp();
55.      void enqueue_Hum(float value);
56.      float dequeue_Hum();
57.
58.      /////////////////////////////////
59.      #define EnableTxRs485 33
60.      /////////////////////////////////
61.
62.      // funciones para UART Y TRAMA
63.      unsigned long getUlong(byte packet[], byte i);
64.      unsigned int getInt(byte packet[], byte i);
65.      float getFloat(byte packet[], byte i);
66.      void fromLongToBytes(byte *bytes, long ing);
67.      void fromFloatToBytes(byte *bytes, float f);
68.      void _tramaServerToControl();
69.      void fragmentar_trama_Sensores(byte buffer[]);
70.      void fragmentar_trama_recibida(byte buffer[]);
71.      void fragmentar_trama_recibidaI2C(byte buffer[]);
72.      int TryGetACK(int TimeOut);
73.      bool al_cambiarTrama_enviar();
74.      int ProccesACK(const int timeOut, void (*okCallBack)(), void (*er
rorCallBack)());
75.      void okAction();
76.      void recibir_trama_sensores();
77.      int ProccesSerialData(const int timeOut,
   byte *buffer, const uint8_t bufferLength, void (*okCallBack)(), void (*
errorCallBack)());
78.      void okRecepcion();
79.      void errorRecepcion();
80.      byte estadoMx1 = 0;
81.      int variador = 0;
82.      char ACK = 'F';
83.      char NAK = 'E';
84.      byte sendData[18];

```

```

85.     byte sendData2[30];
86.     byte sendData3[58];
87.     byte data[18];
88.     const int timeOut = 50;
89.
90.     enum SerialState
91.     {
92.         OKEY,
93.         ERROR,
94.         NO_RESPONSE
95.     };
96.
97. #define cs 5
98. String sinaloa = "/sinaloa1.csv";
99.
100.    void getClima_db(String estadoMX);
101.    float TempHoraCambio_Sonora = 0;
102.    float HumHoraCambio_Sonora = 0;
103.    int currentHour = 0;
104.    int currentMinute = 0;
105.    int previousHour;
106.    int8_t conteo_;
107.
108.    int ubi_pagina = 0;
109.    int ubi_pagina_siguiente = 0;
110.    int on_off_boton = 0;
111.    int on_off_boton_anterior = 0;
112.    float temperatura_estado_anterior = 0;
113.    float humedad_estado_anterior = 0;
114.    float temperatura_estadoMx_setpoint = 0;
115.    float humedad_estadoMx_setpoint = 0;
116.    String sensoresData;
117.    float control_tempFinal = 0;
118.    float control_humFinal = 0;
119.    void i2c_recibir(int num);
120.    void i2c_request();
121.    void _i2cTramaServerToControl();
122.
123. //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
124. void setup()
125. {
126.     Wire.begin(20); // esclavo I2C
127.     Wire.onReceive(i2c_recibir);
128.     Wire.onRequest(i2c_request);
129.     Serial.begin(9600);
130.     Serial2.begin(9600, SERIAL_8N1, 16, 17); // 16 es RX y 17 es TX
131.     Serial.setTimeout(500);
132.     Serial2.setTimeout(500);
133.     pinMode(EnableTxRs485, OUTPUT);
134.     // Imprimir mensaje de error al iniciar spiffs
135.     if (!SPIFFS.begin(true))
136.         Serial.println("An Error has occurred while mounting
137.                     SPIFFS");
138.     config_Wifi_mDns();

```

```

139.     config_rtc();
140.     /* configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
141.     struct tm timeinfo;
142.     if (getLocalTime(&timeinfo))
143.     {
144.         rtc.setTimeStruct(timeinfo); // establece la fecha y hora
145.         almacenadas en la estructura timeinfo en el Real-Time Clock (RTC) del
146.         ESP32
147.     }
148.     else
149.     {
150.         Serial.printf("RTC: Conectado correctamente a servidor
151.             NTP\n");
152.     }
153.     else
154.     {
155.         Serial.printf("RTC: Failed to obtain time \n"); */
156.     }
157.     Serial.print("Iniciando SD ...");
158.     if (SD.begin(cs))
159.     {
160.         Serial.printf("Sd: inicializacion exitosa \n\n");
161.     }
162.     else
163.     {
164.         Serial.printf("Sd: inicializacion fallida \n\n");
165.     }
166.     getClima_db(sinaloa); // extraer del SD los datos de
167.     temperatura y humedad de la base de datos
168.     // Seleccionando archivo index.html main page
169.     server.on("/", HTTP_GET, [] (AsyncWebServerRequest *request)
170.                 { request->send(SPIFFS, "/index.html"); });
171.     // Seleccionando archivo index.css main page
172.     server.on("/index.css",
173.               HTTP_GET, [] (AsyncWebServerRequest *request)
174.                           { request->send(SPIFFS, "/index.css"); });
175.     // Seleccionando archivo script.js main page
176.     server.on("/index.js",
177.               HTTP_GET, [] (AsyncWebServerRequest *request)
178.                           { request->send(SPIFFS, "/index.js"); });
179.     // Seleccionando archivo sensores.js main page
180.     server.on("/sensores.js",
181.               HTTP_GET, [] (AsyncWebServerRequest *request)
182.                           { request->send(SPIFFS, "/sensores.js"); });
183.     // Seleccionando archivo sensores.html y cargando temperatura
184.     server.on("/sensores.html",
185.               HTTP_GET, [] (AsyncWebServerRequest *request)
186.                           { request->send(SPIFFS, "/sensores.html"); });
187.     // Seleccionando archivo sensores.css
188.     server.on("/sensores.css",
189.               HTTP_GET, [] (AsyncWebServerRequest *request)
190.                           { request->send(SPIFFS, "/sensores.css"); });
191.     // Seleccionando archivo extra.html
192.     server.on("/extra.html",
193.               HTTP_GET, [] (AsyncWebServerRequest *request)
194.                           { request->send(SPIFFS, "/extra.html"); });
195.     // Seleccionando archivo extra.css
196.     server.on("/extra.css",
197.               HTTP_GET, [] (AsyncWebServerRequest *request)
198.                           { request->send(SPIFFS, "/extra.css"); });
199.     // yucantan
200.     server.on("/allyucatan.js",
201.               HTTP_GET, [] (AsyncWebServerRequest *request)

```

```

183.           { request->send(SPIFFS, "/allyucatan.js"); });
184.       server.on("/yucatan.html",
185.           HTTP_GET, [] (AsyncWebServerRequest *request)
186.               { request->send(SPIFFS, "/yucatan.html"); });
187.       // Seleccionando archivo sinaloa.html
188.       server.on("/sinaloa.html",
189.           HTTP_GET, [] (AsyncWebServerRequest *request)
190.               { request->send(SPIFFS, "/sinaloa.html"); });
191.       // Seleccionando archivo sinaloa.css
192.       server.on("/all.css",
193.           HTTP_GET, [] (AsyncWebServerRequest *request)
194.               { request->send(SPIFFS, "/all.css"); });
195.       // Seleccionando archivo sinaloa.js
196.       server.on("/ubi", HTTP_GET, [] (AsyncWebServerRequest *request)
197.           {
198.               if (request->hasParam("ubi")) {
199.                   String ubiValue = request->getParam("ubi")->value();
200.                   // Realiza las acciones correspondientes con el valor
201.                   ubi_pagina = ubiValue.toInt();
202.                   request->send(200, "text/plain", "/ubi Valor recibido
correctamente");
203.               } else {
204.                   request->send(400, "text/plain", "/ubi Falta el parámetro
'ubi'");
205.               } });
206.       // Seleccionando archivo index.html para back page
207.       server.on("/index.html",
208.           HTTP_GET, [] (AsyncWebServerRequest *request)
209.               { request->send(SPIFFS, "/index.html"); });
210.       // Pagina no encontrada
211.       server.onNotFound([] (AsyncWebServerRequest *request)
212.           { request->send(400, "text/plain", "Not
found"); });
213.       // Seleccionando archivo CIMAV.png
214.       server.on("/CIMAV.png",
215.           HTTP_GET, [] (AsyncWebServerRequest *request)
216.               { request-
217.                   >send(SPIFFS, "/CIMAV.png", "image/png"); });
218.       // Subiendo temperatura a pagina sensores.html
219.       server.on("/GetTemp_Pronos",
220.           HTTP_GET, [] (AsyncWebServerRequest *request)
221.               { request->send(200, "text/plain",
222.                   GetTemp_Pronos()); });
223.       // Subiendo humedad relativa a pagina sensores.html
224.       server.on("/GetHum_Pronos",
225.           HTTP_GET, [] (AsyncWebServerRequest *request)
226.               { request->send(200, "text/plain",
227.                   GetHum_Pronos()); });
228.       // Parar de emular al ser presionado el boton
229.       server.on("/Emular",
230.           HTTP_GET, [] (AsyncWebServerRequest *request) { // 1 = on

```

```

223.         if (request->hasParam("estado"))
224.         {
225.             String myValue = request->getParam("estado")->value();
226.             // Utiliza el valor recibido como deseas
227.             Serial.println("on emular: " + myValue);
228.             on_off_boton = myValue.toInt();
229.             request->send(200, "text/plain", "Valor emular recibido
correctamente ");
230.         }
231.     else
232.     {
233.         request->send(400, "text/plain", "Falta el parámetro
emular");
234.     }
235.   );
236.   server.on("/StopEmular",
    HTTP_GET, [] (AsyncWebServerRequest *request) { // 0 = off
237.       if (request->hasParam("estado"))
238.       {
239.           String myValue = request->getParam("estado")->value();
240.           // Utiliza el valor recibido como deseas
241.           Serial.println("off stopemular " + myValue);
242.           on_off_boton = myValue.toInt();
243.           request->send(200, "text/plain", "Valor recibido
correctamente");
244.       }
245.     else
246.     {
247.         request->send(400, "text/plain", "Falta el parámetro
'estado'");
248.     }
249.   );
250.   /* server.on("/temperature", HTTP_GET, [] (AsyncWebServerRequest
*request)
251.           { request->send(200, "text/plain", sensoresData); });
*/
252.   // Subiendo temperatura a pagina sensores.html
253.   server.on("/GetTemp_Emular",
    HTTP_GET, [] (AsyncWebServerRequest *request)
254.           { request->send(200, "text/plain",
    GetTemp_Emular()); });
255.   // Subiendo humedad relativa a pagina sensores.html
256.   server.on("/GetHum_Emular",
    HTTP_GET, [] (AsyncWebServerRequest *request)
257.           { request->send(200, "text/plain",
    GetHum_Emular()); });
258.
259.   previousHour = rtc.getHour(true);
260.   TempHoraCambio_Sonora = dequeue_Temp(); // ingreso de dato de
temperatura y humedad de la cola
261.   HumHoraCambio_Sonora = dequeue_Hum();
262.   //_tramaServerToControl(); // envio de datos a control para que
tenga un valor/reference inicial. Normalmente Estada:0, Boton:0,
Temp:xx, Hum:xx
263.
264.   /* server.addHandler(&webSocket);

```

```

265.         webSocket.onEvent(onWebSocketEvent); */
266.
267.         // Iniciar el servidor
268.         server.begin();
269.         Serial.printf("Servidor HTTP iniciado\n\n");
270.     }
271.
272.     void loop()
273.     {
274.         currentHour = rtc.getHour(true);
275.         // currentMinute = rtc.getMinute();
276.         if (currentHour != previousHour)
277.             { // para al momento de cambiar la hora se actualicen los datos
278.                 conteo_++;
279.                 previousHour = currentHour;
280.                 TempHoraCambio_Sonora = dequeue_Temp();
281.                 HumHoraCambio_Sonora = dequeue_Hum();
282.                 Serial.printf("Hora %d , min %d \n", currentHour,
283.                     currentMinute);
284.                 Serial.println("Hora actualizada y datos de temperatura y
285.                     humedad actualizados");
286.                 if (conteo_ == 23)
287.                 {
288.                     getClima_db(sinaloa); // reabastecer la cola de datos de
289.                     temperatura y humedad al vaciarse la cola
290.                     conteo_ = 0;
291.                 }
292.             }
293.             al_cambiarTrama_enviar(); // funcion adaptada PARA I2C
294.             // USAR PARA UART
295.             /* if (al_cambiarTrama_enviar() == true)
296.             {
297.                 ProccesACK(timeOut, okAction, _tramaServerToControl); //
298.                 procesar el ACK de control . Tiempo de espera, accion a realizar si es
299.                 okey, funcion a realizar si es error
300.             } */
301.         }
302.         /**
303.             /// i2c
304.             void i2c_request()
305.             {
306.                 _i2cTramaServerToControl();
307.             }
308.
309.             while (Wire.available())
310.             {
311.                 for (int i = 0; i < 18; i++)
312.                 {
313.                     data[i] = Wire.read(); // Leer cada byte recibido y
314.                     almacenarlo en el arreglo

```

```

314.         }
315.     }
316.     Serial.printf("\n\t I2C TRAMA recibida \n ");
317.     fragmentar_trama_recibidaI2C(data);
318. }
319. void _i2cTramaServerToControl()
320. {
321.     byte seleccion_pagina[4];
322.     byte boton[4];
323.     byte temp[4];
324.     byte hum[4];
325.
326.     // hacer la desfragmentacion de bytes necesaria para enviar la
327.     trama
328.     fromLongToBytes(seleccion_pagina, ubi_pagina);
329.     fromLongToBytes(boton, on_off_boton);
330.     fromFloatToBytes(temp, TempHoraCambio_Sonora);
331.     fromFloatToBytes(hum, HumHoraCambio_Sonora);
332.
333.     sendData[0] = 'I';                      // ascii 73
334.     sendData[1] = seleccion_pagina[0]; // para saber en que pagina
335.     se encuentra el usuario
336.     sendData[2] = seleccion_pagina[1];
337.     sendData[3] = seleccion_pagina[2];
338.     sendData[4] = seleccion_pagina[3];
339.
340.     sendData[5] = boton[0]; // para saber si el boton esta
341.     encendido o apagado
342.     sendData[6] = boton[1];
343.     sendData[7] = boton[2];
344.     sendData[8] = boton[3];
345.
346.     sendData[9] = temp[0];
347.     sendData[10] = temp[1];
348.     sendData[11] = temp[2];
349.     sendData[12] = temp[3];
350.
351.     sendData[13] = hum[0];
352.     sendData[14] = hum[1];
353.     sendData[15] = hum[2];
354.     sendData[16] = hum[3];
355.     sendData[17] = ACK; // ascii 70
356.     sendData[18] = '\n'; // ascii 10
357.
358.     // impresion();
359.     Wire.write(sendData, 18);
360.     ////////////////////////////// Subir Datos al SERVIDOR WEB
361.     //////////////////////////////
362.     String GetTemp_Pronos() // Subir la temperatura pronosticada
363.     dependiendo de la pagina abierta
364.     {
365.         if (ubi_pagina == 1)
366.             return String(TempHoraCambio_Sonora);
367.     }

```

```

365.         return "_";
366.     }
367.     String GetHum_Pronos() // Subir la humedad pronosticada
368.     dependiendo de la pagina abierta
369.     {
370.         if (ubi_pagina == 1) // 1 es Sonora
371.             return String(HumHoraCambio_Sonora);
372.         return "_";
373.     }
374.     String GetTemp_Emular() // Subir la humedad emulada dependiendo
375.     de la pagina abierta
376.     {
377.         return String(control_tempFinal);
378.     }
379.     String GetHum_Emular() // Subir la humedad emulada dependiendo de
380.     la pagina abierta
381.     {
382.         return String(control_humFinal);
383.     }
384.     ////////////////////CONFIGURACION DE WIFI/////////////////////////////
385.     void config_Wifi_mDns()
386.     {
387.         // Conectarse a la red WiFi
388.         Serial.println("Connecting to Wifi ");
389.         WiFi.begin(ssid, password);
390.         while (WiFi.status() != WL_CONNECTED)
391.         {
392.             vTaskDelay(pdMS_TO_TICKS(500));
393.             Serial.print(".");
394.         }
395.         Serial.println("Connected to the WiFi network");
396.         // Imprimir IP local en el Monitor Serie
397.         // WiFi.config(localIP, gateway, subnet);
398.         Serial.println(WiFi.localIP());
399.
400.         // CONFIG MDNS
401.         /* if (!MDNS.begin("emuladorCIMAV"))
402.         {
403.             Serial.println("Error setting up MDNS responder!");
404.             while (1)
405.             {
406.                 delay(1000);
407.             }
408.         }
409.         Serial.printf("mDNS responder started. Nombre para acceder:
410.             emuladorCIMAV.local \n");
411.             MDNS.addService("http", "tcp", 80); */
412.     }
413.     //////////////////// EXTRAER DATOS DE LA SD /////////////////////
414.     void getClima_db(String estadoMX)
415.     {
416.         int d = rtc.getDayofYear(); //

```

```

417.     int horaBuscada = h + (d * 24); // hora actual
419.     int diaBuscado = d + 1;         // dia actual
420.
421.     Serial.printf("SD: diaBuscado: %d, horaBuscada: %d \n",
422.                     diaBuscado, horaBuscada);
423.     // Buscar los datos de temperatura y humedad a partir de la
424.     // hora y dia actual
425.     myFile = SD.open(sinaloa, FILE_READ); // abrimos el archivo
426.     if (myFile)
427.         Serial.printf("SD: Archivo abierto correctamente \n");
428.     else
429.         Serial.printf("SD: Error al abrir el archivo \n");
430.
431.     String line;
432.     while (myFile.available())
433.     {
434.         line = myFile.readStringUntil('\n');
435.         line.trim();
436.         // Dividir la linea en sus componentes: dia, hora,
437.         // temperatura, humedad
438.         int day = line.substring(0, line.indexOf(',')).toInt();
439.         line = line.substring(line.indexOf(',') + 1);
440.         int hour = line.substring(0, line.indexOf(',')).toInt();
441.
442.         if (day == diaBuscado && hour == horaBuscada)
443.         {
444.             // Encontrado el primer dato correspondiente al dia y hora
445.             // actual
446.             float temperature = line.substring(line.indexOf(',') + 1,
447.             line.lastIndexOf(',')).toFloat();
448.
449.             float humidity = line.substring(line.lastIndexOf(',') + 1).toFloat();
450.
451.             // Mostrar el primer dato
452.             Serial.print("Primer dato: Temperatura=");
453.             Serial.print(temperature);
454.             Serial.print(", Humedad=");
455.             Serial.println(humidity);
456.
457.             // Guardar el primer dato en la cola
458.             enqueue_Temp(temperature);
459.             enqueue_Hum(humidity);
460.
461.             // Extraer los siguientes 23 datos
462.             for (int i = 0; i < 23; i++)
463.             {
464.                 if (!myFile.available())
465.                 {

```

```

466.         line.trim();
467.
468.         // Extraer temperatura y humedad
469.         temperature = line.substring(line.indexOf(',',
470.             line.indexOf(',') + 1) + 1, line.lastIndexOf(',')).toFloat();
470.         /* Al agregar line.indexOf(',', line.indexOf(',') + 1)
471.            como el primer índice en line.substring(), se busca el siguiente
472.            carácter ',' después de la posición del primer carácter
473.            ','. */
472.         humidity = line.substring(line.lastIndexOf(',') + 1).toFloat();
473.
474.         // Guardar los siguientes datos en la cola
475.         enqueue_Temp(temperature);
476.         enqueue_Hum(humidity);
477.
478.         // Mostrar los datos extraídos
479.         Serial.print("Datos ");
480.         Serial.print(i + 1);
481.         Serial.print(": Temperatura=");
482.         Serial.print(temperature);
483.         Serial.print(", Humedad=");
484.         Serial.println(humidity);
485.     }
486.
487.     break;
488. }
489. }
490.
491. myFile.close();
492. }
493. //////////////// COLAS ///////////////////////////////
494. void enqueue_Temp(float value)
495. {
496.     if (itemCount1 < MAX_SIZE)
497.     {
498.         rear1 = (rear1 + 1) % MAX_SIZE;
499.         queue_Temp[rear1] = value;
500.         // Serial.printf("insertando queue[%d] = %d \n", rear1,
501.         value);
502.         itemCount1++;
503.     }
504.     else
505.     {
506.         // Cola llena, no se puede agregar más elementos
507.         Serial.println("La cola está llena.");
508.     }
509.     float dequeue_Temp()
510.     {
511.         if (itemCount1 > 0)
512.         {
513.             float value_temp_col = queue_Temp[front1];
514.             front1 = (front1 + 1) % MAX_SIZE;
515.             itemCount1--;
516.             return value_temp_col;

```

```

517.     }
518.   else
519.   {
520.     // Cola vacía, no se puede extraer ningún elemento
521.     Serial.println("La cola está vacía.");
522.     return -1; // Otra forma de indicar un error
523.   }
524. }
525. void enqueue_Hum(float value)
526. {
527.   if (itemCount2 < MAX_SIZE)
528.   {
529.     rear2 = (rear2 + 1) % MAX_SIZE;
530.     queue_Hum[rear2] = value;
531.     // Serial.printf("insertando queue[%d] = %d \n", rear2,
      value);
532.     itemCount2++;
533.   }
534. else
535. {
536.   // Cola llena, no se puede agregar más elementos
537.   Serial.println("La cola está llena.");
538. }
539. }
540. float dequeue_Hum()
541. {
542.   if (itemCount2 > 0)
543.   {
544.     float value = queue_Hum[front2];
545.     front2 = (front2 + 1) % MAX_SIZE;
546.     itemCount2--;
547.     return value;
548.   }
549. else
550. {
551.   // Cola vacía, no se puede extraer ningún elemento
552.   Serial.println("La cola está vacía.");
553.   return -1; // Otra forma de indicar un error
554. }
555. }
556. //////////////// FUNCIONES PARA OBTENER TRAMA POR UART
557. ///////////////////
558. void recibir_trama_sensores()
559. {
560.   digitalWrite(EnableTxRs485, LOW); // habilita la recepcion de
      datos por el puerto serial
561.   if (Serial2.available() > 0)
562.   {
563.     byte buffer[50];
564.     size_t n = Serial2.readBytesUntil('\n', buffer, 50); //
      devuelve el numero de bytes leidos
565.     if (n <= 49)
566.     {
567.       Serial.printf("Procesando trama sensores\n");

```

```

569.          ProcesSerialData(timeOut, buffer, 50, okRepcion,
570.            errorRepcion);
571.          {
572.            Serial.print(buffer[i]);
573.            Serial.print(" ");
574.          }
575.        }
576.      }
577.    }
578.    //////////////// CONVERSIONES PARA TRAMA
579.    // FRAGMENTAR TRAMA
580.    unsigned long getUlong(byte packet[], byte i)
581.    {
582.      // big endian
583.      unsigned long value = 0;
584.      value = (value * 256) + packet[i];
585.      value = (value * 256) + packet[i + 1];
586.      value = (value * 256) + packet[i + 2];
587.      value = (value * 256) + packet[i + 3];
588.      return value;
589.    }
590.    unsigned int getInt(byte packet[], byte i)
591.    {
592.      unsigned int value = 0;
593.      value = (value * 256) + packet[i];
594.      return value;
595.    }
596.    float getFloat(byte packet[], byte i)
597.    {
598.      union tag
599.      {
600.        byte bin[4];
601.        float num;
602.      } u;
603.
604.      u.bin[0] = packet[i];
605.      u.bin[1] = packet[i + 1];
606.      u.bin[2] = packet[i + 2];
607.      u.bin[3] = packet[i + 3];
608.      return u.num;
609.    }
610.    // DESFRAGMETAR TRAMA
611.    void fromFloatToBytes(byte *bytes, float f)
612.    {
613.      int length = sizeof(float);
614.      for (int i = 0; i < length; i++)
615.        bytes[i] = ((byte *)&f)[i];
616.    }
617.    void fromLongToBytes(byte *bytes, long ing)
618.    {
619.      bytes[0] = (byte)((ing & 0xff000000) >> 24);
620.      bytes[1] = (byte)((ing & 0x00ff0000) >> 16);
621.      bytes[2] = (byte)((ing & 0x0000ff00) >> 8);
622.      bytes[3] = (byte)((ing & 0x000000ff));

```

```

623.    }
624.    //////////////// FUNCIONES envio y recepcion de trama DE
TRAMA ///////////////
625.    void fragmentar_trama_Sensores(byte buffer[])
626.    {
627.
628.        float t1 = getFloat(buffer, 1);
629.        Serial.println("");
630.        Serial.print("t1: ");
631.        Serial.println(t1);
632.
633.        float h1 = getFloat(buffer, 5);
634.        Serial.print("h1: ");
635.        Serial.println(h1);
636.
637.        float t2 = getFloat(buffer, 9);
638.        Serial.print("t2: ");
639.        Serial.println(t2);
640.
641.        float h2 = getFloat(buffer, 13);
642.        Serial.print("h2: ");
643.        Serial.println(h2);
644.
645.        float t3 = getFloat(buffer, 17);
646.        Serial.print("t3: ");
647.        Serial.println(t3);
648.
649.        float h3 = getFloat(buffer, 21);
650.        Serial.print("h3: ");
651.        Serial.println(h3);
652.
653.        float t4 = getFloat(buffer, 25);
654.        Serial.print("t4: ");
655.        Serial.println(t4);
656.
657.        float h4 = getFloat(buffer, 29);
658.        Serial.print("h4: ");
659.        Serial.println(h4);
660.
661.        float t5 = getFloat(buffer, 33);
662.        Serial.print("t5: ");
663.        Serial.println(t5);
664.
665.        float h5 = getFloat(buffer, 37);
666.        Serial.print("h5: ");
667.        Serial.println(h5);
668.
669.        float dstemp = getFloat(buffer, 41);
670.        Serial.print("termisor: ");
671.        Serial.println(dstemp);
672.
673.        float fhviento = getFloat(buffer, 45);
674.        Serial.print("fh400 viento: ");
675.        Serial.println(fhviento);
676.
677. /* float fhtemp = getFloat(buffer, 49);

```

```

678.      Serial.print("fh temp: ");
679.      Serial.println(fhtemp);
680.
681.      float fhhum = getFloat(buffer, 53);
682.      Serial.print("fh hum: ");
683.      Serial.println(fhhum);
684.      Serial.println(); */
685.      sensoresData = String(t1) + "," +
686.                      String(t2) + "," +
687.                      String(t3) + "," +
688.                      String(t4) + "," +
689.                      String(t5) + "," +
690.                      String(h1) + "," +
691.                      String(h2) + "," +
692.                      String(h3) + "," +
693.                      String(h4) + "," +
694.                      String(h5) + "," +
695.                      String(3) + "," +
696.                      String(fhviento);
697.
698.      /* webSocket.textAll(sensoresData); */
699.  }
700. void fragmentar_trama_recibida(byte buffer[])
701. {
702.     ubi_pagina = getUlong(buffer, 1);
703.     Serial.printf("\n ubi_pagina: %d \n", ubi_pagina);
704.
705.     on_off_boton = getUlong(buffer, 5);
706.     Serial.printf("boton en : %d \n", on_off_boton);
707.
708.     temperatura_estadoMx_setpoint = getFloat(buffer, 9);
709.     Serial.printf("temp recibida: %0.2f \n",
710.                   temperatura_estadoMx_setpoint);
711.     humedad_estadoMx_setpoint = getFloat(buffer, 13);
712.     Serial.printf("hum recibida: %0.2f \n",
713.                   humedad_estadoMx_setpoint);
714. void fragmentar_trama_recibidaI2C(byte buffer[])
715. {
716.     int ubi_paginai2c = getUlong(buffer, 1);
717.     Serial.printf("\n ubi_pagina: %d \n", ubi_paginai2c);
718.
719.     int on_off_botoni2c = getUlong(buffer, 5);
720.     Serial.printf("boton en : %d \n", on_off_botoni2c);
721.
722.     control_tempFinal = getFloat(buffer, 9);
723.     Serial.printf("temp recibida: %0.2f \n", control_tempFinal);
724.
725.     control_humFinal = getFloat(buffer, 13);
726.     Serial.printf("hum recibida: %0.2f \n", control_humFinal);
727. }
728. void _tramaServerToControl()
729. { // FUNCION PARA DESFRAGMETAR TRAMA DE 19 BYTES
730.     byte seleccion_pagina[4];
731.     byte boton[4];

```

```

732.     byte temp[4];
733.     byte hum[4];
734.
735.     // hacer la desfragmentacion de bytes necesaria para enviar la
    trama
736.     fromLongToBytes(seleccion_pagina, ubi_pagina);
737.     fromLongToBytes(boton, on_off_boton);
738.     fromFloatToBytes(temp, TempHoraCambio_Sonora);
739.     fromFloatToBytes(hum, HumHoraCambio_Sonora);
740.
741.     sendData[0] = 'I';                      // ascii 73
742.     sendData[1] = seleccion_pagina[0]; // para saber en que pagina
    se encuentra el usuario
743.     sendData[2] = seleccion_pagina[1];
744.     sendData[3] = seleccion_pagina[2];
745.     sendData[4] = seleccion_pagina[3];
746.
747.     sendData[5] = boton[0]; // para saber si el boton esta
    encendido o apagado
748.     sendData[6] = boton[1];
749.     sendData[7] = boton[2];
750.     sendData[8] = boton[3];
751.
752.     sendData[9] = temp[0];
753.     sendData[10] = temp[1];
754.     sendData[11] = temp[2];
755.     sendData[12] = temp[3];
756.
757.     sendData[13] = hum[0];
758.     sendData[14] = hum[1];
759.     sendData[15] = hum[2];
760.     sendData[16] = hum[3];
761.     sendData[17] = ACK; // ascii 70
762.     sendData[18] = '\n'; // ascii 10
763.
764.     // impresion();
765.     digitalWrite(EnableTxRs485, HIGH);
766.     Serial2.write(sendData, 18);
767.     vTaskDelay(pdMS_TO_TICKS(19)); // necesario para terminar de
    enviar toda la trama completa
768. }
769.
770. bool al_cambiarTrama_enviar()
771. {
    // al cambiar cualquier variable de control, temperatura o
    humedad enviar trama
772.
773.     if (on_off_boton != on_off_boton_anterior)
774.     {
775.         //_tramaServerToControl(); PARA UART
776.         on_off_boton_anterior = on_off_boton;
777.         return true;
778.     }
779.
780.     if (ubi_pagina != ubi_pagina_siguiente)
781.     {
782.         //_tramaServerToControl(); PARA UART

```

```

783.         ubi_pagina_siguiente = ubi_pagina;
784.         Serial.printf("pagina cambió: %d \n", ubi_pagina);
785.         return true;
786.     }
787.
788.     if ((TempHoraCambio_Sonora != temperatura_estado_anterior) || (
789.         HumHoraCambio_Sonora != humedad_estado_anterior))
790.     {
791.         // _tramaServerToControl(); PARA UART
792.         temperatura_estado_anterior = TempHoraCambio_Sonora;
793.         humedad_estado_anterior = HumHoraCambio_Sonora;
794.         Serial.printf("temp y hum cambio \n");
795.         return true;
796.     }
797.
798.     return false; // ninguno de los casos entonces no se ejecutara
    la funcion de ProccesACK
799. }
800.
801. int TryGetACK(int TimeOut)
802. {
803.     digitalWrite(EnableTxRs485, LOW); // habilita la recepcion de
    datos rs485
804.     unsigned long StartTime = millis();
805.     while ((millis() - StartTime) < TimeOut)
806.     {
807.     }
808.
809.     if (Serial2.available())
810.     {
811.         if (Serial2.read() == ACK)
812.             return OKEY;
813.         if (Serial2.read() == NAK)
814.             return ERROR;
815.     }
816.     return NO_RESPONSE;
817. }
818.
819. int ProccesACK(const int timeOut, void (*okCallBack)(), void (*er
rorCallBack)())
820. {
821.
822.     int rst = TryGetACK(timeOut);
823.     if (rst == OKEY)
824.     {
825.         if (okCallBack != NULL)
826.             okCallBack(); // se comprueba si la función de devolución
    de llamada okCallBack no es un puntero nulo
827.     }
828.     else if (rst == ERROR)
829.     {
830.         if (okCallBack != NULL)
831.             errorCallBack();
832.         Serial.printf("Reenviado Trama por error en recepcion\n");
833.     }

```

```

834.         else if (rst == NO_RESPONSE)
835.         {
836.             errorCallBack();
837.             Serial.printf("No se recibio respuesta, reenviar\n");
838.         }
839.         return rst;
840.     }
841.
842.     void okAction()
843.     {
844.         Serial.printf("\n ACK recibido por el otro esp32 \n");
845.     }
846.
847.     int TryGetSerialData(int TimeOut, byte *buffer) // encontrar ACK
en el buffer y saber como proceder
848.     {
849.         // intenta conseguir los delimitadores de la trama para
verificar que la trama se recibió correctamente dentro de un tiempo
850.         // de 100ms, si no se recibe la trama en ese tiempo, se
devuelve un error
851.         // si se recibe la trama, se devuelve un OK.
852.         // el Ok sirve para proceder a fragmentar la trama
853.         unsigned long StartTime = millis();
854.         digitalWrite(EnableTxRs485, LOW); // habilita la recepcion de
datos por el puerto serial
855.         while (!Serial2.available() && (millis() -
StartTime) < TimeOut)
856.         {
857.         }
858.
859.         if (buffer[0] == 'I' && buffer[49] == ACK)
860.             return OKEY;
861.         else
862.             return ERROR;
863.     }
864.
865.     int ProccesSerialData(const int timeOut,
byte *buffer, const uint8_t bufferLength, void (*okCallBack)(), void (*
errorCallBack)())
866.     { // Comprueba que en el buffer se encuentre el ACK , recepciono
de datos completa
867.
868.         int rst = TryGetSerialData(timeOut, buffer);
869.         if (rst == OKEY)
870.         {
871.             Serial.print(ACK);
872.             if (okCallBack != NULL)
873.             {
874.                 okCallBack(); // al comprobar ACK envia ACK por UART para
avisar correcta recepcion de datos y comienza a frangmentar la trama
875.                 fragmentar_trama_Sensores(buffer);
876.                 Serial.printf(" ACK enviado \n");
877.             }
878.         }
879.         else if (rst == ERROR)
880.         {

```

```

881.         if (okCallBack != NULL)
882.         {
883.             errorCallBack(); // al comprobar que no
esta ACK envia NAK por UART para avisar incorrecta recepcion de datos
884.             digitalWrite(EnableTxRs485, LOW); // deshabilitar recepcion
de trama
885.             Serial.printf("NAK enviado \n");
886.         }
887.     }
888.     return rst;
889. }
890.
891. void okRecepcion()
892. {
// envia ACK por UART para avisar correcta recepcion de datos
893.     Serial.println("Recepcion correcta");
894.     digitalWrite(EnableTxRs485, HIGH);
895.     Serial2.write(ACK);
896.     Serial2.write(ACK);
897. }
898. void errorRecepcion()
899. {
// envia NAK por UART para avisar incorrecta recepcion de datos
900.     digitalWrite(EnableTxRs485, HIGH);
901.     Serial.println("Recepcion incorrecta");
902.     Serial2.write(NAK);
903.     Serial2.write(NAK);
904. }
905.
906. void config_rtc()
907. {
908.     const String
serverName = "http://worldtimeapi.org/api/timezone/America/Monterrey";
909.     http.begin(client, serverName);
910.     http.GET();
911.
912.     StaticJsonDocument<768> doc;
913.     DeserializationError error = deserializeJson(doc,
http.getStream());
914.     // falta filtrar el json
915.     long unixtime = doc["unixtime"];
916.     if (error)
917.     {
918.         Serial.print("deserializeJson() failed: ");
919.         Serial.println(error.c_str());
920.         return;
921.     }
922.     http.end();
923.     Serial.printf("Unix time: %ld\n", unixtime);
924.
925.     rtc.setTime(unixtime);

```

7.1.9 CODIGO PARA INTERFAZ DE USARIO FRONT-END

HTML

Index.html

```
1. <!DOCTYPE html>
2. <html lang="es">
3.
4.   <head>
5.     <meta charset="UTF-8" />
6.     <link rel="icon" href="data:, ">
7.     <title>CIMAV Emulador</title>
8.     <meta name="viewport" content="width=device-width, initial-
   scale=1">
9.     <link rel="stylesheet" href="index.css">
10.    </head>
11.
12.    <body>
13.      <header>
14.        
15.        <nav class="nava">
16.          <a class="navlink" href="../extra.html">Extra</a>
17.        </nav>
18.      </header>
19.      </div>
20.      <div class="titulo">
21.        <h1>Sistema Solar de generación de aire sintético para
   aplicaciones de secado de productos agroindustriales</h1>
22.      </div>
23.      <p class="ope">Opciones a emular:</p>
24.      <form>
25.        <select id="opcionesestados">
26.          <option disabled selected="index.html">Selecciona una
   opción</option>
27.          <option value="sinaloa.html">Sinaloa</option>
28.          <option value="yucatan.html">Yucatán</option>
29.
30.        </select>
31.      </form>
32.
33.      <button type="button" class="btnop" onclick="redirect ()">Mostrar</but-
   ton>
34.      <script src="index.js"></script>
35.    </body>
36.    <footer>
37.      2023 CIMAV
38.    </footer>
39.    <div>
40.
41.  </html>
```

Sinaloa.html

```
1. <html>
2.
3. <head>
4.   <meta charset="UTF-8">
5.   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6.   <link rel="icon" href="data:,">
7.   <meta name="viewport" content="width=device-width, initial-
   scale=1.0">
8.   <title>Emular Sinaloa \ CIMAV</title>
9.   <link rel="stylesheet" href="all.css">
10.  </head>
11.
12.  <body>
13.    <header>
14.      
15.      <nav class="nava">
16.
17.      <a class="navlink" id="home" href="../index.html">Home</a>
18.      <a class="navlink" id="extra" href="../extra.html">Extra</a>
19.    </nav>
20.  </header>
21.  <div class="titulo">
22.    <h1>Sinaloa a Emular</h1>
23.    <h2>Escuinapa</h2>
24.  </div>
25.  <p class="ope">Datos Pronosticados:</p>
26.  <div class="contenedorRecuadro">
27.    <div class="recuadro1">
28.      <div class="contenedortexto">
29.        <p class="nombre">Clima:</p>
30.        <p class="t1">T:</p>
31.        <span id="valort1"></span>
32.        <p class="t1c">°C</p>
33.        <p class="hr">HR:</p>
34.        <span id="valorRH1"></span>
35.        <p class="hrc">%</p>
36.      </div>
37.    </div>
38.  </div>
39.  <div class="fuentex">Fuente: Meteonorm</div>
40.  <div> </div>
41.  <p class="ope">Datos Emulados:</p>
42.  <div class="contenedorRecuadro">
43.    <div class="recuadro1">
44.      <div class="contenedortexto">
45.        <p class="nombre">Clima</p>
46.        <p class="t1">T:</p>
47.        <span id="vfinalt1"></span>
48.        <p class="t1c">°C</p>
49.        <p class="hr">HR:</p>
50.        <span id="vfinalh1"></span>
51.        <p class="hrc">%</p>
```

```
52.          </div>
53.      </div>
54.  </div>          </div>
55.          </div>
56.      </div>
57.  </div>          </div>
58. <div class="divFuente">
59.     <div class="textol1">
60.         Presionar el boton para emular:
61.     </div>
62.     <button type="button" class="btnop" onclick="GetEmularDetener () ">Comenzar</button>
63.     </div>
64. <script src="all.js"></script>
65.
66.
67. </body>
68. <footer>
69.     2023 CIMAV
70. </footer>
71.
72. </html>
```

Yucatán.html

```
1. <html>
2.
3. <head>
4.     <meta charset="UTF-8">
5.     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6.     <link rel="icon" href="data:,>
7.     <meta name="viewport" content="width=device-width, initial-
scale=1.0">
8.     <title>Emular Sinaloa \ CIMAV</title>
9.     <link rel="stylesheet" href="all.css">
10.    </head>
11.
12.    <body>
13.        <header>
14.            
15.            <nav class="nava">
16.
17.            <a class="navlink" id="home" href="../index.html">Home</a>
18.
19.            <a class="navlink" id="extra" href="../extra.html">Extra</a>
20.        </nav>
21.        </header>
22.        <div class="titulo">
23.            <h1>Sinaloa a Emular</h1>
24.            <h2>Escuinapa</h2>
25.        </div>
26.        <p class="ope">Datos Pronosticados:</p>
27.        <div class="contenedorRecuadro">
28.            <div class="recuadro1">
29.                <div class="contenedortexto">
30.                    <p class="nombre">Clima:</p>
```

```

29.          <p class="t1">T:</p>
30.          <span id="valort1"></span>
31.          <p class="t1c">°C</p>
32.          <p class="hr">HR:</p>
33.          <span id="valorRH1"></span>
34.          <p class="hrc">%</p>
35.      </div>
36.  </div>
37.
38.  </div>
39.  <div class="fuentex">Fuente: Meteonorm</div>
40.  <div> </div>
41.  <p class="ope">Datos Emulados:</p>
42.  <div class="contenedorRecuadro">
43.    <div class="recuadro1">
44.      <div class="contenedortexto">
45.        <p class="nombre">Clima</p>
46.        <p class="t1">T:</p>
47.        <span id="vfinalt1"></span>
48.        <p class="t1c">°C</p>
49.        <p class="hr">HR:</p>
50.        <span id="vfinalh1"></span>
51.        <p class="hrc">%</p>
52.      </div>
53.    </div>
54.  </div>
55.  </div>
56.  </div>
57.  </div>
58.  <div class="divFuente">
59.    <div class="textol1">
60.      Presionar el boton para emular:
61.    </div>
62.    <button type="button" class="btnop" onclick="GetEmularDetener () ">Comenzar</button>
63.  </div>
64.  <script src="all.js"></script>
65.
66.
67.  </body>
68.  <footer>
69.    2023 CIMAV
70.  </footer>
71.
72.  </html>

```

CÓDIGO DE PROGRAMACIÓN CSS

All.css

```

1. * {
2.   margin: 0;
3.   padding: 0;
4.   background: #F7F7F7;
5.   font-family: sans-serif;
6. }

```

```

7.
8. header{
9.     display: flex;
10.    justify-content: space-between;
11.    align-items: center;
12.    padding: 30px;
13.    padding-right: 100px;
14.    padding-left: 70px;
15.    min-height: 70px;
16.
17. }
18. .img1{
19.     width: 110px;
20.     height: auto;
21. }
22.
23.
24.
25. .navlink{
26.     font-size: 20px;
27.     text-decoration: none;
28.     padding-left: 30px;
29. }
30.
31. .titulo {
32.     text-align: center;
33.     margin-bottom: 20px;
34.     font-size: 19px;
35.     padding-left: 15px;
36.     padding-right: 15px;
37. }
38.
39. .ope {
40.     margin-left: 50px;
41.     margin-right: 50px;
42.     display: flex;
43.     justify-content: center;
44.     align-items: center;
45.
46.
47.     font-size: 24px;
48.     font-weight: bold;
49.     color: #888888;
50. }
51.
52. .contenedorRecuadro {
53.     display: flex;
54.     justify-content: center;
55.     align-items: center;
56.
57. }
58.
59. .recuadro1 {
60.     padding: 10px;
61.     border: 1px solid #000;
62.     width: 390px;
63.

```

```

64.         border-radius: 10px;
65.         background: #F1F1F1;
66.         text-align: center;
67.         margin-bottom: 2px;
68.         margin-top: 3px;
69.     }
70.     .fuentex{
71.         display: flex;
72.         justify-content: center;
73.         align-items: center;
74.         margin-left: 15px;
75.         margin-right: 15px;
76.         padding-bottom: 10px;
77.     }
78.     .contenedortexto {
79.         display: flex;
80.         background: #F1F1F1;
81.         align-items: center;
82.     }
83. 
84.     .nombre {
85.         margin-right: 20px;
86.         font-weight: bold;
87.         color: #4C3EE9;
88.         background: #F1F1F1;
89.         font-size: 24px;
90.     }
91. 
92. 
93.     .t1,
94.     .hr,
95.     .v1,
96.     .t1c,
97.     .hrc,
98.     .vc {
99.         display: inline-block;
100.        margin-right: 5px;
101.        background: #F1F1F1;
102.        font-size: 17px;
103.        text-align: center;
104.        font-weight: bold;
105.    }
106. 
107.     .valort1,
108.     .valorRH1,
109.     .vfinalt1,.vfinalh1 {
110.         display: inline-block;
111.         font-size: 19px;
112.         text-align: center;
113.         color: #29a025;
114.     }
115. 
116.     .t1c,
117.     .hrc,
118.     .vc {
119.         margin-right: 30px;
120.     }

```

```

121.
122.     .divFuente {
123.
124.         display: flex;
125.         justify-content: center;
126.         align-items: center;
127.         margin-left: 15px;
128.         margin-right: 15px;
129.
130.     }
131.
132.     .a3 {
133.         text-decoration: none;
134.     }
135.
136.     .btnop {
137.
138.         background-color: #080808;
139.         /* Cambiar el color de fondo del botón */
140.         color: white;
141.         /* Cambiar el color del texto del botón */
142.         border: none;
143.         /* Eliminar el borde del botón */
144.         padding: 10px 20px;
145.         /* Cambiar el espaciado interno del botón */
146.         border-radius: 10px;
147.         font-size: 18px;
148.         /* Cambiar el tamaño de fuente del botón */
149.         cursor: pointer;
150.         /* Cambiar el cursor al pasar por encima del botón */
151.         margin-left: 60px;
152.         width: 123px;
153.         height: 40px;
154.         transition: all 0.6s ease;
155.     }
156.
157.
158.     footer {
159.
160.         background-color: #363636;
161.         /* Color de fondo del footer */
162.         position: fixed;
163.         left: 0;
164.         bottom: 0;
165.         width: 100%;
166.         padding: 4px;
167.         text-align: left;
168.     }
169.
170.
171.
172.     .cambiocolor2{
173.         background-color: #CC3819;
174.         transition: all 0.5s ease;
175.     }
176.
177.

```

```
178.      @media (max-width:700px) {  
179.  
180.  
181.      header{  
182.          flex-direction: column;  
183.      }  
184.  
185.  
186.  }
```

Index.css

```
1. * {  
2.     margin: 0;  
3.     padding: 0;  
4.     background: #F7F7F7;  
5.     font-family: sans-serif ;  
6. }  
7.  
8. header{  
9.     display: flex;  
10.        justify-content: space-between;  
11.        align-items: center;  
12.        padding: 30px;  
13.        padding-right: 100px;  
14.        padding-left: 70px;  
15.        min-height: 70PX;  
16.  
17.    }  
18.    .img1{  
19.        width:110px;  
20.        height:auto;  
21.    }  
22.  
23.  
24.  
25.    .navlink{  
26.        font-size: 20px;  
27.        text-decoration: none;  
28.        padding-left: 30px;  
29.    }  
30.  
31.    .titulo {  
32.        text-align: center;  
33.        padding-bottom: 15px;  
34.        padding-top: 5px;  
35.        padding-left: 50px;  
36.        padding-right: 50px;  
37.  
38.    }  
39.  
40.    .ope {  
41.        display: flex;  
42.        justify-content: center;  
43.        align-items: center;  
44.        font-size: 24px;
```

```

45.         font-weight: bold;
46.         color: #888888;
47.     }
48.
49.     form {
50.         height: 50px;
51.         display: flex;
52.         justify-content: center;
53.         padding: 10px;
54.
55.     }
56.
57.
58.     select {
59.         font-size: 1.4em;
60.         /* Tamaño de fuente */
61.         padding: 10px;
62.         /* Espacio alrededor del texto */
63.         border-radius: 10px;
64.         /* Borde redondeado */
65.         width: 500px;
66.         background-color: #F1F1F1;
67.         /* Color de fondo */
68.         color: #4C3EE9;
69.         align-items: center;
70.         margin-right: 15px;
71.
72.     }
73.
74.     input[type="submit"] {
75.
76.         background-color: #080808;
77.         /* Cambiar el color de fondo del botón */
78.         color: white;
79.         /* Cambiar el color del texto del botón */
80.         border: none;
81.         /* Eliminar el borde del botón */
82.         padding: 10px 20px;
83.         /* Cambiar el espaciado interno del botón */
84.         border-radius: 10px;
85.         font-size: 18px;
86.         /* Cambiar el tamaño de fuente del botón */
87.         cursor: pointer;
88.         /* Cambiar el cursor al pasar por encima del botón */
89.
90.     }
91.
92.     .btnop {
93.         display: flex;
94.         justify-content: center;
95.         margin: auto;
96.         background-color: #080808;
97.         /* Cambiar el color de fondo del botón */
98.         color: white;
99.         /* Cambiar el color del texto del botón */
100.        border: none;
101.        /* Eliminar el borde del botón */

```

```

102.         padding: 10px 20px;
103.         /* Cambiar el espacio interno del botón */
104.         border-radius: 10px;
105.         font-size: 18px;
106.         /* Cambiar el tamaño de fuente del botón */
107.         cursor: pointer;
108.         /* Cambiar el cursor al pasar por encima del botón */
109.     }
110.
111.     select option {
112.         color: #000;
113.     }
114.
115.     .contenedorRecuadro {
116.         display: flex;
117.         justify-content: center;
118.         align-items: center;
119.     }
120.
121.     .recuadro1 {
122.         margin: 10px;
123.         padding: 10px;
124.         border: 1px solid #000;
125.         width: 700px;
126.
127.         border-radius: 10px;
128.         background: #F1F1F1;
129.         text-align: center;
130.         margin-bottom: 20px;
131.         margin-top: 10px;
132.     }
133.
134.     .contenedortexto {
135.         display: flex;
136.         justify-content: space-between;
137.         background: #F1F1F1;
138.         align-items: center;
139.     }
140.
141.
142.     footer{
143.
144.         background-color: #363636;
145.         /* Color de fondo del footer */
146.         position: fixed;
147.         left: 0;
148.         bottom: 0;
149.         width: 100%;
150.         padding: 10px;
151.
152.     }
153.
154.
155.     @media (max-width:700px) {
156.
157.
158.         header{

```

```

159.           flex-direction: column;
160.       }
161.
162.
163.
164.   }

```

CÓDIGO JAVASCRIPT

Index.js

```

1. function redirect() { //funcion para redireccionar a la pagina de
  estados de mexico
2.   var seleccion = document.getElementById("opcionesestados").value;
3.   if (seleccion !== '') {
4.     window.location.href = seleccion;
5.     console.log(seleccion);
6.   }
7. }

```

AllYucatan.js

```

1. document.addEventListener("DOMContentLoaded", function () {
2.   var valueToSend = "2";
3.   var url = "/ubi?ubi=" + encodeURIComponent(valueToSend);
4.
5.   var xhttp = new XMLHttpRequest();
6.
7.   xhttp.onreadystatechange = function () {
8.     if (xhttp.readyState == 4 && xhttp.status == 200) {
9.       console.log("Valor enviado al servidor con éxito");
10.      }
11.    };
12.
13.    xhttp.open("GET", url, true);
14.    xhttp.send();
15.
16.
17.    setTimeout.ajaxCall, 500); //ejecuta un segund despues para
  alcanzar la varialbe de controlo dentro del esp32
18.  });
19.
20.  //ajax para subir los datos de temperatura y humedad emuladas por
  sistema de control
21.  var temp_control = new XMLHttpRequest();
22.  var hum_control = new XMLHttpRequest();
23.  function ajaxVariablesEmuladasControl() { // se ejecuta cada
  hora y sube la temperatura y humedad pronosticada
24.
25.
26.    temp_control.onreadystatechange = function () {
27.      if (temp_control.readyState == XMLHttpRequest.DONE) {
28.        if (temp_control.status == 200) {
29.          updateDataEmular(temp_control.responseText);
30.        }

```

```

31.         else {
32.             console.log('error', temp_control);
33.         }
34.     };
35. };
36.
37. hum_control.onreadystatechange = function () {
38.     if (hum_control.readyState == XMLHttpRequest.DONE) {
39.         if (hum_control.status == 200) {
40.             updateDataEmular2(hum_control.responseText);
41.         }
42.         else {
43.             console.log('error2', hum_control);
44.         }
45.     }
46. };
47.
48. temp_control.open("GET", "GetTemp_Emular", true);
49. temp_control.send();
50. hum_control.open("GET", "GetHum_Emular", true);
51. hum_control.send();
52.
53. var tempEmulada = document.getElementById("vfinalt1");
54. var humEmulada = document.getElementById("vfinalh1");
55.
56. function updateDataEmular(valor) {
57.     tempEmulada.innerHTML = valor;
58. }
59.
60. function updateDataEmular2(valor) {
61.     humEmulada.innerHTML = valor;
62. }
63.
64.
65. //AL presionarse el boton se envia una peticion al servidor para
   comenzar la emulacion, cambia a color rojo, cambia el texto
66. // y luego vuelve a cambiar de texto despues de unos segundo. Al
   presionarlo por segunda vez cambia color y texto y manda
67. // otra peticion al servidor para detener la emulacion
68. const boton = document.querySelector(".btноп");
69. const home = document.getElementById('home');
70. const extra = document.getElementById('extra');
71. home
72. var id = 1;
73. function GetEmularDetener() {
74.     // envia la peticion al servidor para comenzar y detener la
       emulacion
75.     // cambia el color y el texto al presionarse el boton
76.
77.     boton.classList.toggle("cambiocolor2");
78.
79.     if (id == 1) {
80.         console.log("btn estado: en ON");
81.         boton.innerHTML = "Emulando..";
82.         id = 0;
83.         GetEmular();
84.         setTimeout(printDetener, 3000);

```

```

85.         home.style.opacity = '0';
86.         home.disabled = true;
87.         extra.style.opacity = '0';
88.         extra.disabled = true;
89.         setTimeout(ajaxVariablesEmuladasControl, 500);
90.     } else {
91.         console.log("btn estado: en off");
92.         boton.innerHTML = "Comenzar ";
93.         id = 1;
94.         GetDetener();
95.         home.style.opacity = '1';
96.         home.disabled = false;
97.         extra.style.opacity = '1';
98.         extra.disabled = false;
99.     }
100.
101. }
102.
103. var tempPronosticada = document.getElementById("valort1");
104. var humPronosticada = document.getElementById("valorRH1");
105.
106. function updateData(valor) {
107.     tempPronosticada.innerHTML = valor;
108. }
109.
110. function updateData2(valor) {
111.     humPronosticada.innerHTML = valor;
112. }
113.
114.
115.
116.
117.
118. var xmlhttp_temp_pro = new XMLHttpRequest();
119. var xmlhttp2_hum_pro = new XMLHttpRequest();
120. function ajaxCall() { // se ejecuta cada hora y sube la
    temperatura y humedad pronosticada
121.
122.
123.     xmlhttp_temp_pro.onreadystatechange = function () {
124.         if (xmlhttp_temp_pro.readyState == XMLHttpRequest.DONE) {
125.             if (xmlhttp_temp_pro.status == 200) {
126.                 updateData(xmlhttp_temp_pro.responseText);
127.             }
128.             else {
129.                 console.log('error', xmlhttp_temp_pro);
130.             }
131.         }
132.     };
133.
134.     xmlhttp2_hum_pro.onreadystatechange = function () {
135.         if (xmlhttp2_hum_pro.readyState == XMLHttpRequest.DONE) {
136.             if (xmlhttp2_hum_pro.status == 200) {
137.                 updateData2(xmlhttp2_hum_pro.responseText);
138.             }
139.             else {
140.                 console.log('error2', xmlhttp2_hum_pro);

```

```

141.         }
142.     }
143. }
144.
145.     xmlhttp_temp_pro.open("GET", "GetTemp_Pronos", true);
146.     xmlhttp_temp_pro.send();
147.     xmlhttp2_hum_pro.open("GET", "GetHum_Pronos", true);
148.     xmlhttp2_hum_pro.send();
149. }
150.
151.
152.
153. // se ejecuta cada hora y sube la temperatura y humedad
154. // pronosticada
155. function scheduleAjax() {
156.     var currentTime = new Date();
157.     console.log('hora', currentTime.getHours());
158.
159.     // Calcular el tiempo hasta la próxima hora
160.     var nextHour = new Date(currentTime.getFullYear(), currentTime.
161.         getMonth(), currentTime.getDate(), currentTime.getHours() + 1, 0, 0);
162.     var timeToNextHour = nextHour.getTime() -
163.         currentTime.getTime();
164.
165.     setTimeout(function () {
166.         ajaxCall();
167.         scheduleAjax();
168.     }, timeToNextHour);
169. }();
170.
171.
172. function GetEmular() { // envia la peticion al servidor
173.     var valueToSend = "1";
174.     var url = "/Emular?estado=" + encodeURIComponent(valueToSend);
175.
176.     var xhttp = new XMLHttpRequest();
177.
178.     xhttp.onreadystatechange = function () {
179.         if (xhttp.readyState == 4 && xhttp.status == 200) {
180.             console.log("Valor enviado al servidor con éxito");
181.         }
182.     };
183.
184.     xhttp.open("GET", url, true);
185.     xhttp.send();
186. }
187.
188. //envia una peticion al servidor para detener la emulacion
189. //dicho de otra manera envia un valor a la esp32 para enviar ese
190. //dato al otro esp32 por uart y detener la ejecucion
191. function GetDetener() { // envia la peticion al servidor
192.     var valueToSend = "0";
193.     var url = "/StopEmular?estado=" + encodeURIComponent(valueToSend);
194. }

```

```

194.     var xhttp = new XMLHttpRequest();
195.
196.     xhttp.onreadystatechange = function () {
197.         if (xhttp.readyState == 4 && xhttp.status == 200) {
198.             console.log("Valor enviado al servidor con éxito");
199.         }
200.     };
201. };
202.
203. xhttp.open("GET", url, true);
204. xhttp.send();
205. }
206.
207. function printDetener() {
208.     boton.innerHTML = "Detener"
209. }

```

All.js

```

1. document.addEventListener("DOMContentLoaded", function () {
2.     var valueToSend = "1";
3.     var url = "/ubi?ubi=" + encodeURIComponent(valueToSend);
4.
5.     var xhttp = new XMLHttpRequest();
6.
7.     xhttp.onreadystatechange = function () {
8.         if (xhttp.readyState == 4 && xhttp.status == 200) {
9.             console.log("Valor enviado al servidor con éxito");
10.            }
11.        };
12.
13.        xhttp.open("GET", url, true);
14.        xhttp.send();
15.
16.
17.        setTimeout.ajaxCall, 500); //ejecuta un segund despues para
   alcanzar la varialbe de controlo dentro del esp32
18.    });
19.
20.    //ajax para subir los datos de temperatura y humedad emuladas por
   sistema de control
21.    var temp_control = new XMLHttpRequest();
22.    var hum_control = new XMLHttpRequest();
23.    function ajaxVariablesEmuladasControl() { // se ejecuta cada
   hora y sube la temperatura y humedad pronosticada
24.
25.
26.        temp_control.onreadystatechange = function () {
27.            if (temp_control.readyState == XMLHttpRequest.DONE) {
28.                if (temp_control.status == 200) {
29.                    updateDataEmular(temp_control.responseText);
30.                }
31.            else {
32.                console.log('error', temp_control);
33.            }
34.        }

```

```

35.      };
36.
37.      hum_control.onreadystatechange = function () {
38.          if (hum_control.readyState == XMLHttpRequest.DONE) {
39.              if (hum_control.status == 200) {
40.                  updateDataEmular2(hum_control.responseText);
41.              }
42.              else {
43.                  console.log('error2', hum_control);
44.              }
45.          }
46.      };
47.
48.      temp_control.open("GET", "GetTemp_Emular", true);
49.      temp_control.send();
50.      hum_control.open("GET", "GetHum_Emular", true);
51.      hum_control.send();
52.  }
53. var tempEmulada = document.getElementById("vfinalt1");
54. var humEmulada = document.getElementById("vfinalh1");
55.
56. function updateDataEmular(valor) {
57.     tempEmulada.innerHTML = valor;
58. }
59.
60. function updateDataEmular2(valor) {
61.     humEmulada.innerHTML = valor;
62. }
63.
64.
65. //AL presionarse el boton se envia una peticion al servidor para
   comenzar la emulacion, cambia a color rojo, cambia el texto
66. // y luego vuelve a cambiar de texto despues de unos segundo. Al
   presionarlo por segunda vez cambia color y texto y manda
67. // otra peticion al servidor para detener la emulacion
68. const boton = document.querySelector(".btnop");
69. const home = document.getElementById('home');
70. const extra = document.getElementById('extra');
71. home
72. var id = 1;
73. function GetEmularDetener() {
74.     // envia la peticion al servidor para comenzar y detener la
   emulacion
75.     // cambia el color y el texto al presionarse el boton
76.
77.     boton.classList.toggle("cambiocolor2");
78.
79.     if (id == 1) {
80.         console.log("btn estado: en ON");
81.         boton.innerHTML = "Emulando..";
82.         id = 0;
83.         GetEmular();
84.         setTimeout(printDetener, 3000);
85.         home.style.opacity = '0';
86.         home.disabled = true;
87.         extra.style.opacity = '0';
88.         extra.disabled = true;

```

```

89.         setTimeout (ajaxVariablesEmuladasControl, 500);
90.     } else {
91.         console.log("btn estado: en off");
92.         boton.innerHTML = "Comenzar ";
93.         id = 1;
94.         GetDetener();
95.         home.style.opacity = '1';
96.         home.disabled = false;
97.         extra.style.opacity = '1';
98.         extra.disabled = false;
99.     }
100.
101. }
102.
103. var tempPronosticada = document.getElementById("valort1");
104. var humPronosticada = document.getElementById("valorRH1");
105.
106. function updateData(valor) {
107.     tempPronosticada.innerHTML = valor;
108. }
109.
110. function updateData2(valor) {
111.     humPronosticada.innerHTML = valor;
112. }
113.
114.
115.
116.
117.
118. var xmlhttp_temp_pro = new XMLHttpRequest();
119. var xmlhttp2_hum_pro = new XMLHttpRequest();
120. function ajaxCall() { // se ejecuta cada hora y sube la
    temperatura y humedad pronosticada
121.
122.
123.     xmlhttp_temp_pro.onreadystatechange = function () {
124.         if (xmlhttp_temp_pro.readyState == XMLHttpRequest.DONE) {
125.             if (xmlhttp_temp_pro.status == 200) {
126.                 updateData(xmlhttp_temp_pro.responseText);
127.             }
128.             else {
129.                 console.log('error', xmlhttp_temp_pro);
130.             }
131.         }
132.     };
133.
134.     xmlhttp2_hum_pro.onreadystatechange = function () {
135.         if (xmlhttp2_hum_pro.readyState == XMLHttpRequest.DONE) {
136.             if (xmlhttp2_hum_pro.status == 200) {
137.                 updateData2(xmlhttp2_hum_pro.responseText);
138.             }
139.             else {
140.                 console.log('error2', xmlhttp2_hum_pro);
141.             }
142.         }
143.     };
144. }

```

```

145.         xmlhttp_temp_pro.open("GET", "GetTemp_Pronos", true);
146.         xmlhttp_temp_pro.send();
147.         xmlhttp2_hum_pro.open("GET", "GetHum_Pronos", true);
148.         xmlhttp2_hum_pro.send();
149.     }
150.
151.
152.
153.     // se ejecuta cada hora y sube la temperatura y humedad
154.     // pronosticada
155.     function scheduleAjax() {
156.         var currentTime = new Date();
157.         console.log('hora', currentTime.getHours());
158.
159.         // Calcular el tiempo hasta la próxima hora
160.         var nextHour = new Date(currentTime.getFullYear(), currentTime.
161.             getMonth(), currentTime.getDate(), currentTime.getHours() + 1, 0, 0);
162.         var timeToNextHour = nextHour.getTime() -
163.             currentTime.getTime();
164.
165.         setInterval(function () {
166.             ajaxCall();
167.             scheduleAjax();
168.         }, timeToNextHour);
169.     })();
170.
171.
172.     function GetEmular() { // envia la peticion al servidor
173.         var valueToSend = "1";
174.         var url = "/Emular?estado=" + encodeURIComponent(valueToSend);
175.
176.         var xhttp = new XMLHttpRequest();
177.
178.         xhttp.onreadystatechange = function () {
179.             if (xhttp.readyState == 4 && xhttp.status == 200) {
180.                 console.log("Valor enviado al servidor con éxito");
181.             }
182.         };
183.
184.         xhttp.open("GET", url, true);
185.         xhttp.send();
186.     }
187.
188.
189.     // envia una peticion al servidor para detener la emulacion
190.     // dicho de otra manera envia un valor a la esp32 para enviar ese
191.     // dato al otro esp32 por uart y detener la ejecucion
192.     function GetDetener() { // envia la peticion al servidor
193.         var valueToSend = "0";
194.         var url = "/StopEmular?estado=" + encodeURIComponent(valueToSend);
195.         var xhttp = new XMLHttpRequest();
196.
197.         xhttp.onreadystatechange = function () {

```

```

198.         if ( xhttp.readyState == 4 && xhttp.status == 200 ) {
199.             console.log("Valor enviado al servidor con éxito");
200.         }
201.     } ;
202.     xhttp.open("GET", url, true);
203.     xhttp.send();
204. }
205. }
206.
207. function printDetener() {
208.     boton.innerHTML = "Detener"
209. }

```

7.1.10 CÓDIGO GRAFICAS SENSORES EN MATLAB

```

classdef GraficaSensoresHumedad < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure           matlab.ui.Figure
        monitor            matlab.ui.control.EditField
        MonitorEditFieldLabel matlab.ui.control.Label
        SalirButton        matlab.ui.control.Button
        InicioButton       matlab.ui.control.Button
        grafica            matlab.ui.control.UIAxes
    end

    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: InicioButton
        function InicioButtonPushed(app, event)
            device = serialport("COM3",9600);
            configureTerminator(device,"LF");
            i=1;
            while(1)
                app.monitor.Value=readline(device);
                data(i)=str2double(readline(device));

                plot(app.grafica,data)
                i=i+1;
                pause(0.1);

            end
        end

        % Button pushed function: SalirButton
        function SalirButtonPushed(app, event)
            app.delete
        end

        % Value changed function: monitor
        function monitorValueChanged(app, event)

```

```

    value = app.monitor.Value;

end
end

% Component initialization
methods (Access = private)

% Create UIFigure and components
function createComponents(app)

% Create UIFigure and hide until all components are created
app.UIFigure = uifigure('Visible', 'off');
app.UIFigure.Position = [100 100 640 480];
app.UIFigure.Name = 'MATLAB App';

% Create grafica
app.grafica = uiaxes(app.UIFigure);
title(app.grafica, 'SENSOR ')
xlabel(app.grafica, 'Tiempo(seg)')
ylabel(app.grafica, 'Humedad Relativa %')
app.grafica.GridLineWidth = 0.7;
app.grafica.MinorGridLineWidth = 0.7;
app.grafica.LineWidth = 0.7;
app.grafica.Position = [44 159 553 261];

% Create InicioButton
app.InicioButton = uibutton(app.UIFigure, 'push');
app.InicioButton.ButtonPushedFcn = createCallbackFcn(app,
@InicioButtonPushed, true);
app.InicioButton.Position = [89 137 100 23];
app.InicioButton.Text = 'Inicio';

% Create SalirButton
app.SalirButton = uibutton(app.UIFigure, 'push');
app.SalirButton.ButtonPushedFcn = createCallbackFcn(app,
@SalirButtonPushed, true);
app.SalirButton.Position = [89 107 100 23];
app.SalirButton.Text = 'Salir';

% Create MonitorEditFieldLabel
app.MonitorEditFieldLabel = uilabel(app.UIFigure);
app.MonitorEditFieldLabel.HorizontalAlignment = 'right';
app.MonitorEditFieldLabel.FontSize = 18;
app.MonitorEditFieldLabel.Position = [188 435 65 23];
app.MonitorEditFieldLabel.Text = 'Monitor';

% Create monitor
app.monitor = uieditfield(app.UIFigure, 'text');
app.monitor.ValueChangedFcn = createCallbackFcn(app,
@monitorValueChanged, true);
app.monitor.Position = [268 428 185 30];

% Show the figure after all components are created
app.UIFigure.Visible = 'on';

```

```

        end
    end

% App creation and deletion
methods (Access = public)

    % Construct app
    function app = GraficaSensoresHumedad

        % Create UIFigure and components
        createComponents(app)

        % Register the app with App Designer
        registerApp(app, app.UIFigure)

        if nargout == 0
            clear app
        end
    end

    % Code that executes before app deletion
    function delete(app)

        % Delete UIFigure when app is deleted
        delete(app.UIFigure)
    end
end

```