



FACULTAD DE INGENIERIA

Universidad de Buenos Aires

CARRERA DE ESPECIALIZACIÓN EN SISTEMAS EMBEBIDOS

MEMORIA DEL TRABAJO FINAL

Banco de pruebas automático para equipos eléctricos y electrónicos

Autor:
Ing. Gonzalo Gontad

Director:
Mg. Ing. Ramiro Alonso (FIUBA)

Jurados:
Esp. Ing. Jorge Fonseca (FIUBA)
Esp. Ing. Jerónimo La Bruna (FIUBA)
Ing. Juan Manuel Cruz (FIUBA, UTN-FRBA)

*Este trabajo fue realizado en la Ciudad Autónoma de Buenos Aires,
entre mayo de 2019 y diciembre de 2020.*

Resumen

La presente memoria detalla el proceso de diseño y desarrollo de un sistema de prueba de equipos electrónicos para la línea de producción y control de calidad de la empresa Posthac S.A. El sistema fue diseñado para hacer las pruebas de funcionamiento de temporizadores para iluminación y drivers para luminarias LED.

En este trabajo se aplicaron conocimientos adquiridos en relación al desarrollo de software sobre sistemas operativos de tiempo real, los protocolos de comunicación, el diseño de hardware y las metodologías de gestión de proyectos.

Agradecimientos

Esta sección es para agradecimientos personales y es totalmente **OPCIONAL**.

Índice general

Resumen	I
1. Introducción general	1
1.1. Contexto	1
1.2. Temporizadores para iluminación	1
1.3. Drivers para luminarias LED	2
1.4. Motivación	4
1.5. Objetivos y alcance	4
1.5.1. Objetivos	4
1.5.2. Alcance	5
2. Introducción específica	7
2.1. Requerimientos	7
2.2. Planificación	8
2.3. Módulo WIFI ESP-01	9
2.4. Protocolo HTTP	10
3. Diseño e implementación	13
3.1. Estructura general del sistema	13
3.1.1. Módulo principal	14
3.1.2. Módulos adicionales	14
3.1.3. Interfaz de usuario	14
3.2. Desarrollo de hardware del módulo principal	14
3.2.1. Placa de desarrollo EDU-CIAA	15
3.2.2. Puertos de conexión	15
3.2.3. Interfaz UART opto-aislada	16
3.2.4. Entradas y salidas digitales	18
3.2.5. Salida y entradas analógicas	20
3.3. Desarrollo de hardware del módulo prueba de drivers	21
3.4. Desarrollo de hardware del módulo prueba de temporizadores	23
3.5. Desarrollo de circuitos impresos	23
3.6. Arquitectura del software	25
3.6.1. Arquitectura del software ejecutado en los módulos Bluepill.	25
3.6.2. Implementación del software del módulo Bluepill.	26
3.6.3. Arquitectura del software ejecutado en la EDU-CIAA.	29
3.6.4. Tarea terminal	31
3.6.5. Tarea servidor web	32
3.6.6. Tarea intérprete	34
3.6.7. Tareas test	35
3.6.8. Interfaz de usuario	38
4. Ensayos y Resultados	41
4.1. Ensayos de hardware	41

4.1.1. Ensayo de UART optoaislada	41
4.1.2. Calibración de ADCs	43
4.2. Pruebas de integración	45
5. Conclusiones	49
5.1. Conclusiones generales	49
5.2. Próximos pasos	49

Índice de figuras

1.1. Temporizadores para iluminación de uso residencial	2
1.2. Probador de temporizadores.	3
1.3. Driver para LEDs.	3
2.1. Diagrama activity on node del trabajo.	9
2.2. Módulo WIFI ESP-01 de AI Thinker	10
2.3. Estructura de una petición HTTP.	11
2.4. Estructura de una respuesta HTTP.	12
3.1. Diagrama en bloques del sistema.	13
3.2. Diagrama en bloques del módulo principal.	15
3.3. Diagrama en bloques de un puerto de conexión.	16
3.4. Estructura de una trama de datos de maestro a esclavo.	17
3.5. Estructura de una trama de datos de esclavo a maestro.	17
3.6. Diagrama esquemático de la interfaz UART optoaislada.	18
3.7. Diagrama esquemático de una entrada digital.	19
3.8. Diagrama esquemático de una salida digital y la salida de alimentación de 220 VAC.	19
3.9. Diagrama esquemático de la salida analógica de un puerto.	20
3.10. Diagrama esquemático de una entrada analógica.	20
3.11. Diagrama en bloques del módulo prueba de drivers.	21
3.12. Esquemático del circuito de medición de corriente de drivers. . . .	22
3.13. Esquemático del circuito de medición de tensión de drivers. . . .	22
3.14. Diagrama en bloques del módulo prueba de temporizadores. . . .	23
3.15. Renderizado y fotografía de placa de un puerto del módulo principal	24
3.16. Fotografía de la placa del módulo prueba de drivers de LEDs y la placa del módulo prueba de temporizadores de tres y cuatro terminales	24
3.17. Protocolos de comunicación entre bloques de software	25
3.18. Diagrama en bloques general del software del módulo Bluepill. . .	26
3.19. Diagrama en bloques de la interrupción de recepción por UART. .	28
3.20. Diagrama en bloques del bucle principal e interrupción del timer. .	29
3.21. Modelo de capas del software en la EDU-CIAA.	30
3.22. Esquema de comunicación entre tareas.	31
3.23. Diagrama en bloques del servidor web.	33
3.24. Diagrama de transiciones entre el servidor web y el navegador. .	33
3.25. Diagrama de estados de la MEF del test de drivers.	37
3.26. Diagrama de estados de la MEF del test de temporizadores. . . .	38
3.27. Captura de la base de la interfaz de usuario.	39
3.28. Captura del panel de test de drivers.	39
3.29. Captura del panel de test de temporizadores.	40
3.30. Captura del panel de calibración de ADCs.	40

4.1.	Diagrama del banco de pruebas de la interfaz UART optoaislada.	41
4.2.	Captura de la terminal serie al concluir el envío de tramas.	42
4.3.	Captura de la comunicación maestro-esclavo.	43
4.4.	Banco de pruebas para la calibración de ADCs.	43
4.5.	Calibración de ADCs.	44
4.6.	Calibración de ADCs, zoom cerca del cero.	44
4.7.	Calibración de ADCs luego de corrección de offset por software . .	45

Índice de tablas

1.1.	Tabla de comparación de equipos de prueba de drivers	4
3.1.	Tabla de comparación de optoacopladores para la interfaz UART optoaislada	18
4.1.	Caso de uso de prueba de drivers	46
4.2.	Caso de uso de prueba de temporizadores	47

Dedicado a... [OPCIONAL]

Capítulo 1

Introducción general

Este capítulo introduce al lector a los procesos de prueba de temporizadores y drivers de LEDs y las diferentes opciones disponibles. También se presentan los objetivos y el alcance establecido.

1.1. Contexto

La industria manufacturera es aquella que se dedica a la transformación de la materia prima en bienes de consumo. Además de procesos de transformación de materia, se desarrollan otros en simultáneo, muchos de ellos asociados a la gestión de la calidad. La industria de la manufactura electrónica no es ajena a estos procesos y en particular dentro de aquellos que hacen a la calidad, hay uno muy importante que es la prueba final del producto. Esta consiste principalmente en la verificación de su correcto funcionamiento y del cumplimiento de sus especificaciones. Para estas pruebas es común el uso de instrumentos y dispositivos de prueba diseñados específicamente para un tipo o una familia de productos en particular.

La empresa Posthac S.A., la cual impulsó el desarrollo del sistema que trata esta memoria, cuenta con una amplia línea de productos en el área de iluminación automática e iluminación para vía pública, de los cuales hay dos grupos que se destacan: los temporizadores y los drivers para luminarias LED. La razón que destaca estos productos es que los primeros son los de mayor volumen de venta dentro de los que se producen en las instalaciones de la empresa, y los segundos son productos que están en etapa de desarrollo y con amplia perspectiva de venta en el área de iluminación de vía pública. Ante esta situación se realizó un relevamiento de la capacidad actual de la empresa, principalmente en lo que se refiere a prueba de productos, y se evaluaron distintas opciones de mejora que se detallarán a continuación.

1.2. Temporizadores para iluminación

Un temporizador para iluminación es un dispositivo semiautomático que tiene como función principal mantener encendida una luminaria durante un tiempo preestablecido luego de recibir un pulso de activación, normalmente generado con un pulsador. Estos dispositivos son de uso común en viviendas y edificios.

En la actualidad los temporizadores electrónicos son los de mayor difusión, pero existen otros tipos como los temporizadores mecánicos.

Dentro de los temporizadores electrónicos los hay de dos, tres y cuatro terminales de conexión, de tiempo fijo o ajustable y con salidas conmutadas por relé o triac. En la figura 1.1a se observa un temporizador con salida a rele de cuatro terminales, mientras que la figura 1.1b es de un temporizador con pulsador de tres terminales. La principal diferencia entre ambos es que los temporizadores de cuatro terminales poseen una entrada de activación independiente y los de tres terminales se activan por el mismo terminal que se sirve de salida.



(A) Temporizador de cuatro terminales. (B) Temporizador de tres terminales.

FIGURA 1.1. Temporizadores para iluminación de uso residencial

La prueba de este tipo de dispositivo consiste en verificar los siguientes puntos:

- Accionamiento correcto de la carga.
- Cumplimiento de tiempo de encendido máximo.
- Cumplimiento de tiempo de encendido mínimo.

En el momento de hacer la evaluación, las pruebas de temporizadores en la empresa las hacía un operario utilizando un cronómetro y el banco de pruebas de la figura 1.2, donde se podían conectar hasta seis temporizadores a la vez. Usando estos elementos el operario verificaba los puntos antes indicados en forma manual.

Dado que estos productos no tienen un nivel de producción y venta muy masivo, no se encuentran opciones comerciales de sistemas automáticos de prueba. Incluso grandes y medianas empresas en China, que el autor de esta memoria ha visitado, utilizan sistemas propios o métodos manuales para realizar estas pruebas.

1.3. Drivers para luminarias LED

Los diodos LED son dispositivos semiconductores capaces de convertir la energía eléctrica en luz. Para que el LED funcione de forma adecuada y alcanzar la vida útil establecida, la excitación se debe realizar manteniendo la temperatura dentro del rango de trabajo. Esto se consigue utilizando algún medio para extraer el calor, como disipadores o forzadores, y limitando la potencia sobre los LEDs. Para

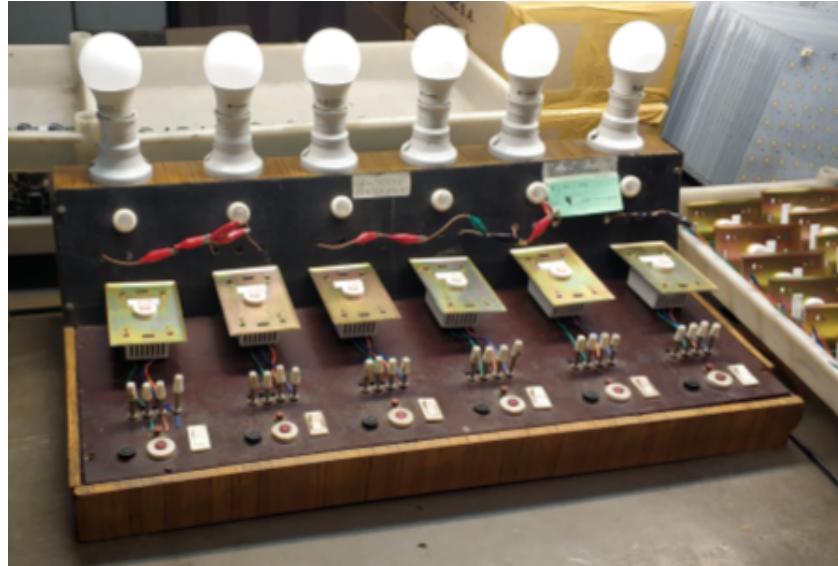


FIGURA 1.2. Probador de temporizadores.

lograr que la potencia sobre los LEDs sea estable y esté limitada al rango de trabajo se utilizan los drivers de LED. Estos drivers son equipos que convierten la energía eléctrica a una forma adecuada para excitar las cargas LED. La característica principal de este tipo de drivers es que entregan en su salida una corriente constante, de modo que la potencia sobre los LEDs resulte estable. Además, muchos de estos drivers tienen la posibilidad de dimerizar o controlar la corriente de salida mediante una entrada adicional para dicho fin. Esta entrada de dimerizado permite controlar el nivel de iluminación producido por los LEDs y además posibilita controlar la temperatura de trabajo de los LED. Para tal fin, se utilizan sensores que miden y retroalimentan al driver la temperatura sobre los LEDs. Si es necesario, el driver reducirá la potencia entregada y así se bajará la temperatura. En la figura 1.3 se puede ver un modelo de driver típico de luminarias LED para vía pública.



FIGURA 1.3. Driver para LEDs.

En lo que respecta a la prueba de este tipo de equipos, los principales parámetros son:

- Tensión de salida.
- Corriente de salida.
- Función de dimerizado si corresponde.
- Potencia de entrada.
- Factor de potencia.

Para hacer estas pruebas existen en el mercado diferentes opciones de equipamiento con distintas características. En la tabla 1.1 se comparan algunos modelos evaluados, donde se puede apreciar que en general los sistemas de prueba de drivers están diseñados para probar de un equipo a la vez.

TABLA 1.1. Comparación de equipos de prueba de drivers

Prueba	63115A	61501	LT101A	WT5080
Factor de potencia	-	SÍ	SÍ	SÍ
Tensión de entrada	-	0-500V	3-300V	1-500V
Corriente de entrada	-	0-30A	10mA-5A	5mA-5A
Tensión de salida	0-600V	-	3-300V	3-500V
Corriente de salida	0-5A	-	10mA-5A	5mA-5A
Dimerizado	NO	NO	NO	NO
Emulación de Carga	SÍ	NO	NO	NO
Pruebas simultáneas	2/4	1	1	1
Marca	Chroma	Chroma	Everfine	InvenFine

1.4. Motivación

Luego de analizar las opciones disponibles para la prueba de temporizadores y drivers de LEDs, es evidente que para el caso de la prueba de temporizadores se requiere un sistema diseñado a medida. Además, si bien hay distintas opciones para pruebas de drivers de LED, ninguna de ellas cumple con la necesidad de la empresa Posthac S.A. de tener un sistema de pruebas adaptable a distintos productos y expandible para poder probar múltiples equipos al mismo tiempo. De este modo surgió la idea de desarrollar este sistema de pruebas que, además de satisfacer las necesidades actuales, deje lugar a futuras adaptaciones para distintas pruebas.

1.5. Objetivos y alcance

1.5.1. Objetivos

Los objetivos inicialmente propuestos para el trabajo realizado son: reducir los tiempos de prueba para lograr un uso eficiente de las horas hombre, mejorar la

calidad de las pruebas y estar preparados para las futuras producciones de drivers de LED.

1.5.2. Alcance

El desarrollo se limitó a los siguientes puntos:

- Desarrollo y fabricación del prototipo del módulo principal.
- Desarrollo y fabricación de prototipos de dos módulos adicionales. Uno para ensayo de temporizadores y otro para ensayo de salida de drivers para luminarias LED.
- Desarrollo de la interfaz de usuario mediante conexión WIFI.
- Código fuente.
- Diagramas eléctricos y diseños de PCB.

No están incluidos:

- Otros módulos adicionales no detallados anteriormente.
- Interfaz gráfica para la conexión mediante cable con PC.

Capítulo 2

Introducción específica

En este capítulo se presentan al lector las tecnologías que fueron utilizadas y se establecen los requerimientos y la planificación del trabajo llevado a cabo.

2.1. Requerimientos

El diseño del sistema de pruebas se realizó siguiendo una serie de requerimientos que fueron establecidos al comienzo del proyecto. A continuación se detallan cada uno de ellos según la etapa del sistema a la que hacen referencia.

Módulo principal:

- Debe poder conectarse a una red WIFI que comparta con la pantalla de usuario.
- Debe ser capaz de transmitir información y representarla mediante una interfaz WEB.
- Debe ser capaz de recibir órdenes mediante interfaz WEB.
- Debe ser posible conectar seis módulos adicionales del mismo tipo a la vez.
- Debe ser capaz de realizar dos mediciones analógicas simultáneas con una tasa de muestreo de al menos mil muestras por segundo.
- Debe contar con al menos tres salidas digitales y tres entradas digitales por cada puerto.
- Debe disponer de una salida analógica por cada puerto 0-10v.
- Debe poder proveer a los módulos adicionales de alimentación en cada puerto.
- Debe poder conectarse con una PC mediante una conexión RS232 a modo de puerto de configuración y debugging.
- El centro de procesamiento deberá implementarse con un placa EDU-CIAA.

Módulo adicional para ensayo de temporizadores:

- Debe poder probar temporizadores de tres y cuatro terminales.
- Debe proveer de alimentación (220 VAC) al equipo bajo prueba.
- Debe generar señales que indiquen al módulo principal el estado de funcionamiento del equipo bajo prueba.

Módulo para ensayo de driver de luminaria LED:

- Debe poder generar señales que indiquen los valores de tensión y corriente de salida del driver bajo prueba.
- Debe poder conmutar cargas.
- Debe proveer de alimentación (220 VAC) al equipo bajo prueba.
- Debe proveer al equipo bajo prueba de una señal de control para dimerizado.

Control de versiones:

- Se debe montar un repositorio GIT.
- El repositorio deberá ser accesible mediante la plataforma github para poder trabajar en forma remota.

2.2. Planificación

El desarrollo del sistema de pruebas tuvo como guía la planificación realizada en el marco de la materia gestión de proyectos. Si bien esta planificación original contemplaba que la presentación del prototipo se realizaría en mayo de 2020, fue necesario posponerla debido a que en el momento de realizar las pruebas de las etapas analógicas del módulo principal, se encontró un error de diseño. No pudiendo corregirse este error, fue imperioso agregar una fase de rediseño de las etapas analógicas y la comunicación con los puertos del módulo principal. Esto se puede ver reflejado en el diagrama de activity on node de la figura 2.1 donde se debió incluir una etapa de rediseño del hardware. Finalmente, el tiempo invertido en el desarrollo del trabajo suma un total de 658 horas.

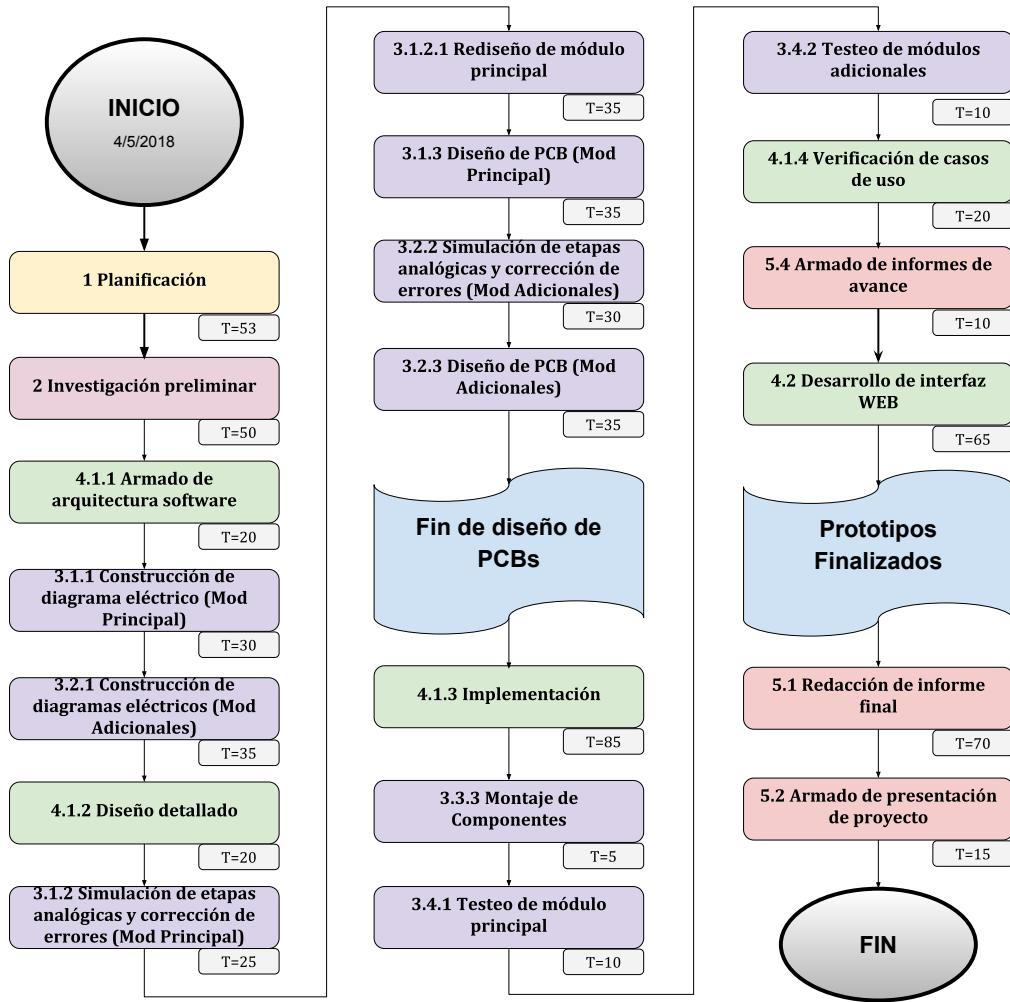


FIGURA 2.1. Diagrama activity on node del trabajo.

2.3. Módulo WIFI ESP-01

Uno de los requerimientos establecidos para el trabajo final fue dotar al equipo probador de una conexión WIFI para la implementación de la interfaz de usuario. Para dicho fin se utilizó un módulo WIFI modelo ESP-01, desarrollado por la empresa AI-Thinker. En la figura 2.2 se puede ver una fotografía del módulo.

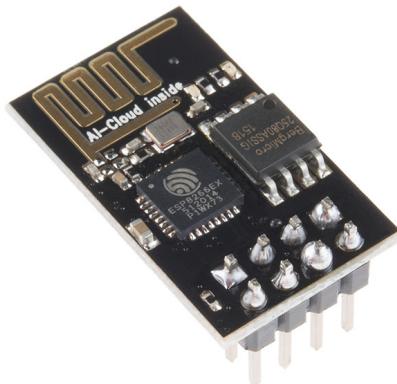


FIGURA 2.2. Módulo WIFI ESP-01 de AI Thinker

El módulo ESP-01 es un dispositivo basado en el SoC (*System on Chip*, sistema en chip) ESP8266EX fabricado por Espressif, cuyas principales características son:

- Núcleo Tensilica L106 de 32 bit funcionando a 80 Mhz.
- Stack TCP/IP y WIFI integrado.
- Compatible con estándar IEEE 802.11b/g/n .
- Seguridad WEP/WPA-PSK/WPA2-PSK.
- Posibilidad de funcionamiento en modo *station*, *access point* y *station+access point*.
- Configuración y comunicación mediante comandos AT por interfaz UART.
- Cuatro GPIOs disponibles, de los cuales dos son para la interfaz UART.

Si bien el módulo ESP-01 resuelve una parte de la comunicación estableciendo las conexiones y enviando y recibiendo paquetes de datos, es necesario montar un servidor que sea capaz de resolver las distintas peticiones de datos o recursos que lleguen a través del módulo. Para esto se utiliza el protocolo HTTP que se verá a continuación.

2.4. Protocolo HTTP

El protocolo de transferencia de hipertexto, conocido como HTTP, es un protocolo de comunicación que permite el intercambio o acceso a datos y recursos entre dispositivos servidores y clientes. Este protocolo es la base del intercambio de datos en la web.

Un uso muy habitual del protocolo HTTP es la carga de una página Web, donde el navegador Web actúa de cliente iniciando la comunicación y solicita a un servidor un recurso, por ejemplo una página Web. Esta solicitud se realiza mediante una petición HTTP con una estructura básica, como se ve en la figura 2.3, que incluye:

- Un método que indica al servidor qué es lo que se pretende hacer con la petición HTTP. Los métodos más usados son GET, para obtener un recurso del servidor, y POST para enviar información desde el cliente al servidor.

- La URL del recurso, que indica la ubicación del recurso al que se quiere acceder en el servidor.
- La versión del protocolo HTTP que se está utilizando.
- Una cabecera que opcionalmente se puede incluir con distintos parámetros, por ejemplo lenguaje aceptado, tipo de contenido, fecha y hora, datos del navegador, etc.
- El cuerpo del mensaje, normalmente utilizado en métodos POST para enviar información al servidor.

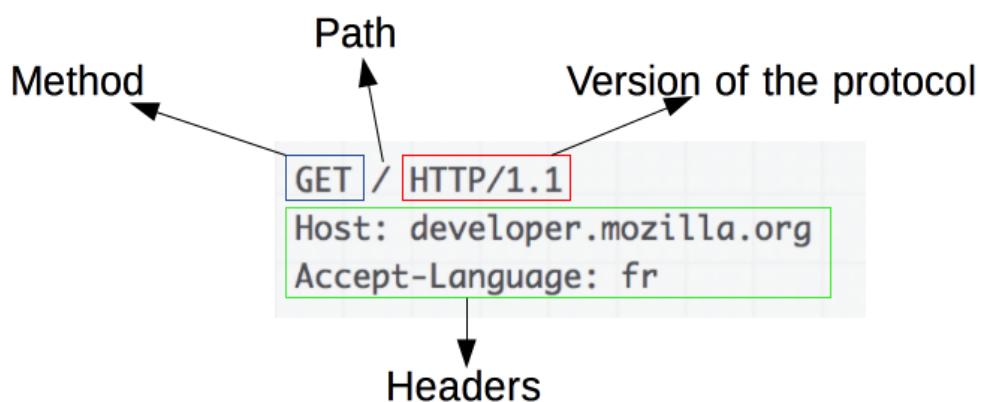


FIGURA 2.3. Estructura de una petición HTTP.

Luego, el servidor puede responder con los datos pedidos por el cliente, que al tratarse de la carga de una página Web, es habitual que ésta se realice por partes, transfiriendo documentos HTML, scripts, hojas de estilo y datos. Al igual que las peticiones HTTP, las respuestas HTTP también tienen una estructura, como se ve en la figura 2.4, compuesta por:

- La versión del protocolo.
- El código del estado, es decir un código que indica si se pudo ejecutar la solicitud y en caso contrario por qué no se pudo.
- Mensaje de estado.
- Una cabecera con parámetros al igual que en la petición.
- El cuerpo con el recurso pedido.

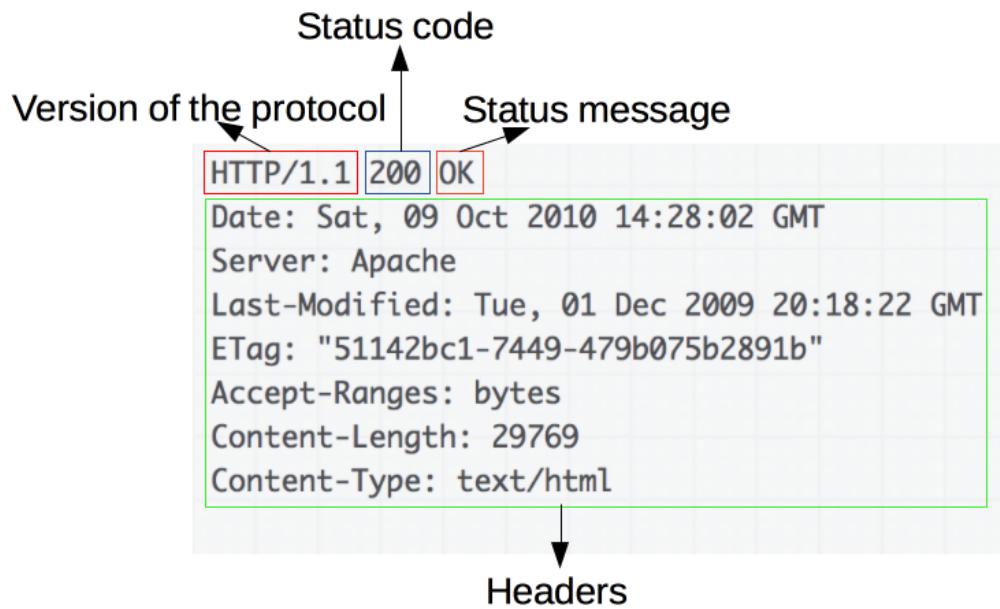


FIGURA 2.4. Estructura de una respuesta HTTP.

Finalmente, el cliente, que en este caso es un navegador Web, compone la página web y ejecuta los scripts.

Capítulo 3

Diseño e implementación

En este capítulo se explican con detalle todo los elementos que componen al sistema y los criterios utilizados para su desarrollo. Se parte de una breve descripción de la estructura del sistema, se hace un análisis detallado del desarrollo de los módulos de hardware y se concluye con la implementación del software.

3.1. Estructura general del sistema

Con el objetivo de lograr un sistema adaptable y escalable se optó por realizar un diseño modular. Este diseño consiste en tres tipos de módulos o etapas: el módulo principal, los módulos adicionales y la interfaz de usuario. La figura 3.1 indica en forma general la estructura del sistema y el modo en que los módulos están conectados entre sí.

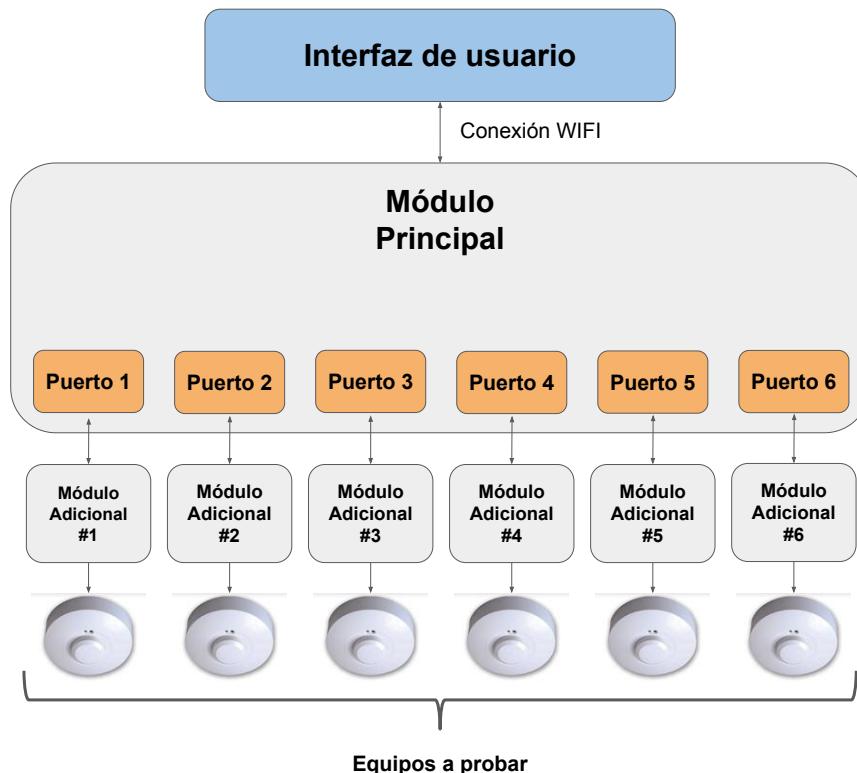


FIGURA 3.1. Diagrama en bloques del sistema.

3.1.1. Módulo principal

El módulo principal es la base del sistema, allí se encuentra el núcleo de procesamiento, la etapa de comunicación con la interfaz de usuario, el servidor web y los puertos de conexión con los módulos adicionales. Estos puertos de conexión son los encargados de capturar las señales provenientes de los módulos adicionales, tanto en forma analógica como digital. Además, son capaces de generar señales analógicas y digitales y proveer de alimentación a los módulos adicionales. Una característica importante de estos puertos de conexión es que se encuentran aislados eléctricamente uno del otro para poder trabajar con equipos no aislados y únicamente se comunican con el núcleo de procesamiento mediante una interfaz optoaislada.

3.1.2. Módulos adicionales

Los módulos adicionales cumplen las funciones de conectar y adaptar las señales entre los puertos del módulo principal y el equipo bajo prueba. Estos módulos se diseñan específicamente para cada tipo o familia de productos. Gracias a esta estructura modular se puede adaptar el sistema para realizar distintas pruebas con pocos cambios de hardware. Al momento se desarrollaron dos módulos adicionales, uno para prueba de temporizadores y otro para la prueba de salida de drivers.

3.1.3. Interfaz de usuario

La interfaz de usuario es el medio por el cual se controla al sistema. Se desarrolló de forma que pueda ejecutarse en cualquier plataforma que tenga un navegador web y esté conectada a la misma red que el módulo principal. Por ejemplo, se puede ejecutar desde una tablet, notebook o celular y con solo ingresar el IP del sistema de pruebas se tiene acceso al panel de control donde se puede seleccionar la prueba a realizar, configurar los parámetros, ejecutar las pruebas y visualizar los resultados.

3.2. Desarrollo de hardware del módulo principal

El módulo principal, como se mencionó anteriormente, posee el núcleo de procesamiento implementado con una placa de desarrollo EDU-CIAA, un módulo de comunicación WIFI ESP-01 para la interfaz de usuario y los puertos de conexión con los módulos adicionales. En la figura 3.2 se presenta un diagrama en bloques del módulo principal cuyos elementos serán detallados a continuación.

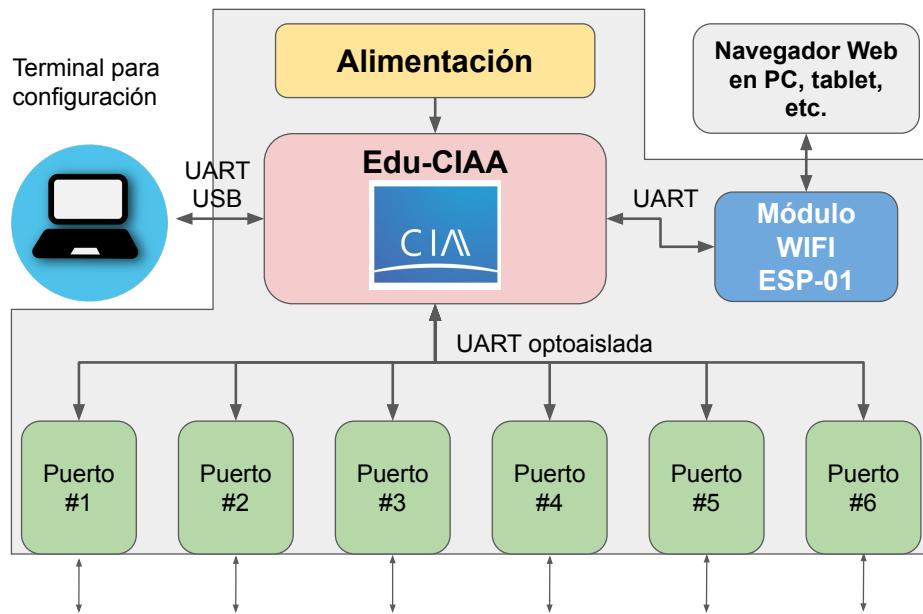


FIGURA 3.2. Diagrama en bloques del módulo principal.

3.2.1. Placa de desarrollo EDU-CIAA

El uso de la placa de desarrollo EDU-CIAA como núcleo de procesamiento, fue uno de los requerimientos establecidos al inicio del proyecto. Esta decisión se basó principalmente en que la EDU-CIAA es la plataforma sobre la que más se trabajó durante la especialización y se dispone de gran variedad de recursos de software y soporte por parte de la comunidad. Respecto al hardware, la placa dispone un procesador LPC4337 con dos núcleos asimétricos, un Cortex M4 y un Cortex M0, trabajando a 208 Mhz y un amplio abanico de interfaces de comunicación como ser bus I2C, SPI, USB, CAN y varias UARTs.

3.2.2. Puertos de conexión

Los puertos de conexión son las piezas de hardware que más trabajo requirieron en su diseño. Fue necesario desarrollar dos esquemas preliminares de hardware a causa de que las primeras pruebas que se realizaron no fueron exitosas. Esto se debió a que el diseño no contemplaba el hecho de que muchos drivers de LEDs no están aislados entre entrada y salida, lo que impide el uso de una masa común entre los distintos puertos de conexión. Siendo imposible solucionar esto sin alterar la topología del circuito, se optó por un esquema de puertos opto-aislados. La figura 3.3 representa en un esquema de bloques el circuito de un puerto de conexión donde se puede apreciar que cada puerto dispone de un microcontrolador de la familia STM32, en particular para el desarrollo se eligió un módulo Bluepill basado en el microcontrolador STM32F108C8T6 con núcleo Cortex M3. Este microcontrolador cumple las siguientes funciones:

- Comunicación mediante UART opto-aislada con la EDU-CIAA.
- Capturar el estado de las tres entradas digitales del puerto.

- Muestreo de las dos entradas analógicas del puerto.
- Comunicación con el conversor digital-analógico de la salida analógica 0-10v.
- Generar las señales de las tres salidas digitales del puerto y activar el relay de alimentación de 220 VAC.

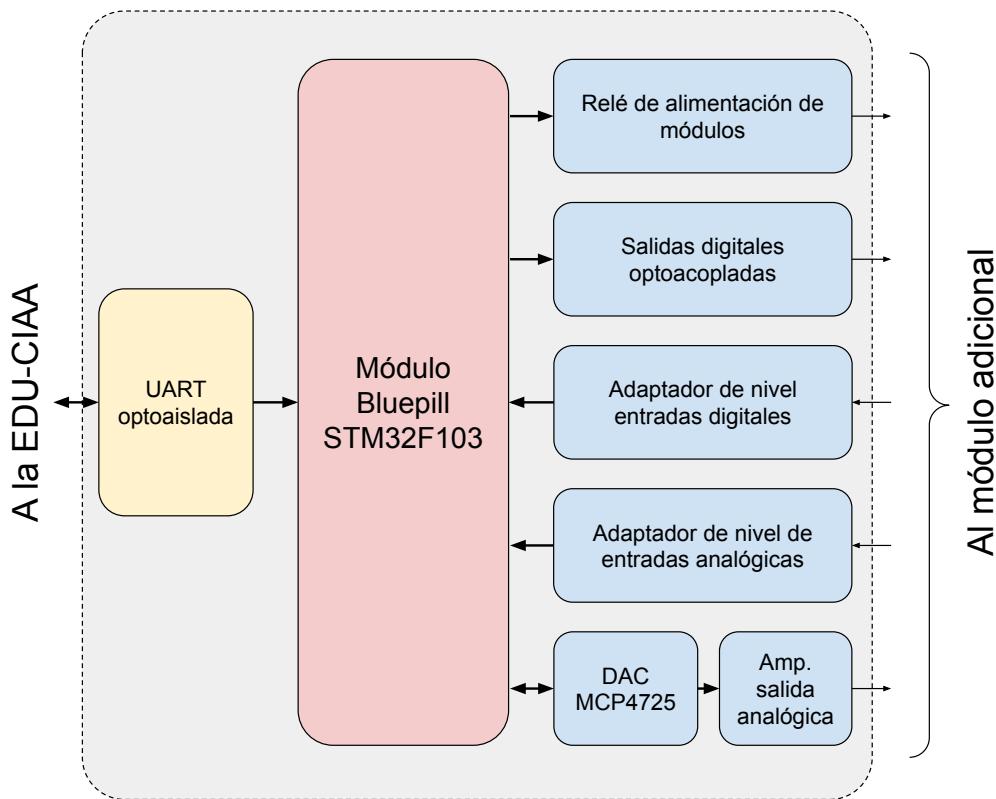


FIGURA 3.3. Diagrama en bloques de un puerto de conexión.

3.2.3. Interfaz UART opto-aislada

Para el diseño de la interfaz UART opto-aislada se tuvieron en cuenta los requerimientos de cantidad de canales analógico a digital y digital a analógico, la velocidad de muestreo y el número de entradas y salidas digitales, que en forma indirecta establecen la velocidad de comunicación mínima que debe poseer la interfaz UART opto-aislada entre los puertos y la EDU-CIAA. Asimismo, fue necesario establecer un protocolo para empaquetar los datos e identificar a qué puerto corresponde cada paquete de datos. El protocolo diseñado consiste en un protocolo maestro-esclavos donde un maestro, en este caso la EDU-CIAA, inicia la comunicación con un esclavo enviando una trama de 5 bytes, como lo indica la figura 3.4, que a través de la UART opto-aislada llega a todos los puertos.

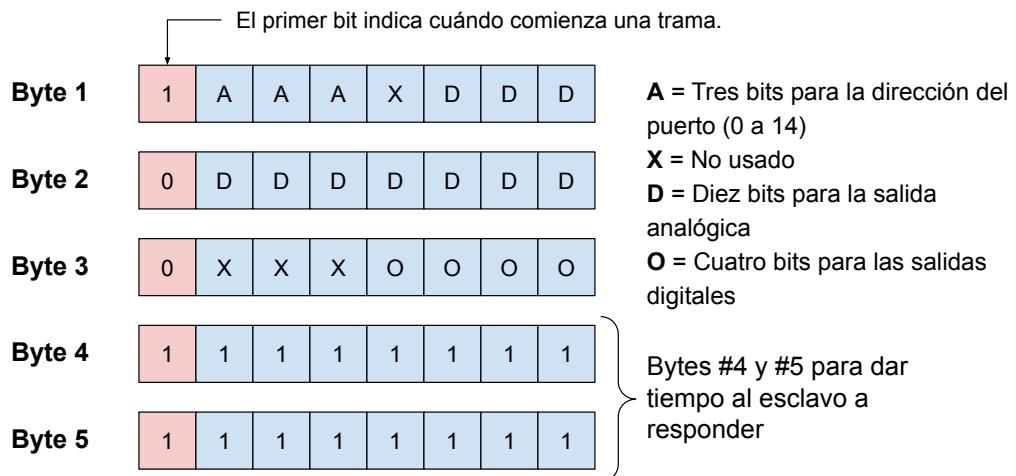


FIGURA 3.4. Estructura de una trama de datos de maestro a esclavo.

A continuación cada esclavo, es decir cada uno de los puertos, captura la trama y obtiene la dirección correspondiente a la trama recibida y la compara con la propia. Solo el esclavo con la misma dirección debe responder al maestro mediante una trama de 4 bytes con la estructura que se indica en la figura 3.5.

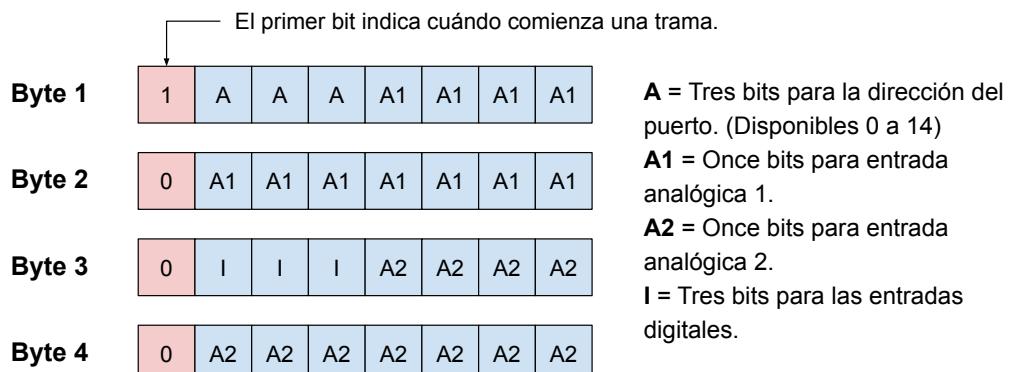


FIGURA 3.5. Estructura de una trama de datos de esclavo a maestro.

Una vez establecido el protocolo donde se deben transmitir 5 bytes de maestro a esclavo y 4 bytes de esclavo a maestro en configuración full-duplex, y teniendo en cuenta que en el listado de requerimientos se establecio una taza de refresco de mil veces por segundo por cada uno de los seis puertos, se calculó una velocidad de comunicación mínima requerida de 30 KB/s. Luego se escogió la velocidad estándar más cercana para un puerto UART, 460800 Kb/s que excede ampliamente lo mínimo requerido y deja espacio para futuras expansiones.

Para la selección de los opto acopladores, se evaluaron algunas alternativas disponibles en el mercado local. La tabla 3.1 resume las características principales de

los mismos donde se puede destacar al tiempo de respuesta como el factor limitante. Utilizando como criterio de diseño que el tiempo de respuesta del optoacoplador debe ser 10 veces menor que el tiempo de un bit para lograr una salida aceptable, significa que para lograr una velocidad de 460800 Kb/s se requiere un tiempo de respuesta menor a 217 ns. Por lo tanto en la tabla 3.1 se puede observar que la mejor opción para la aplicación es el optoacoplador 6N137 debido a que es el único capaz de trabajar a la velocidad de 460800 Kb/s requerida.

TABLA 3.1. Comparación de optoacopladores

Modelo	PC817(1)	4N35(2)	6N137(3)
Tensión de alimentación	<80 V	<70 V	5 V
Tipo de salida	Transistor	Transistor	<i>Open drain</i>
Tiempo de respuesta subida/bajada	18 us / 18 us	10 us / 10 us	75 ns / 75 ns
Tensión de aislación	5 KV	5 KV	5.3 KV
Cantidad de pines	4	6	8

Una vez seleccionado el optoacoplador se realizó el diagrama esquemático de la interfaz opto-aislada de la figura 3.6. El diseño contempla que la salida, es decir la línea de datos del puerto hacia la EDU-CIAA, sea por colector y resistencia pull-up para que se puedan conectar todos los puertos en paralelo. A su vez, se tuvo en cuenta que la entrada de todos los puertos están conectadas a la misma salida de la EDU-CIAA y esta no debe ser sobrecargada, por lo que se utilizó un transistor a la entrada que actúe de buffer.

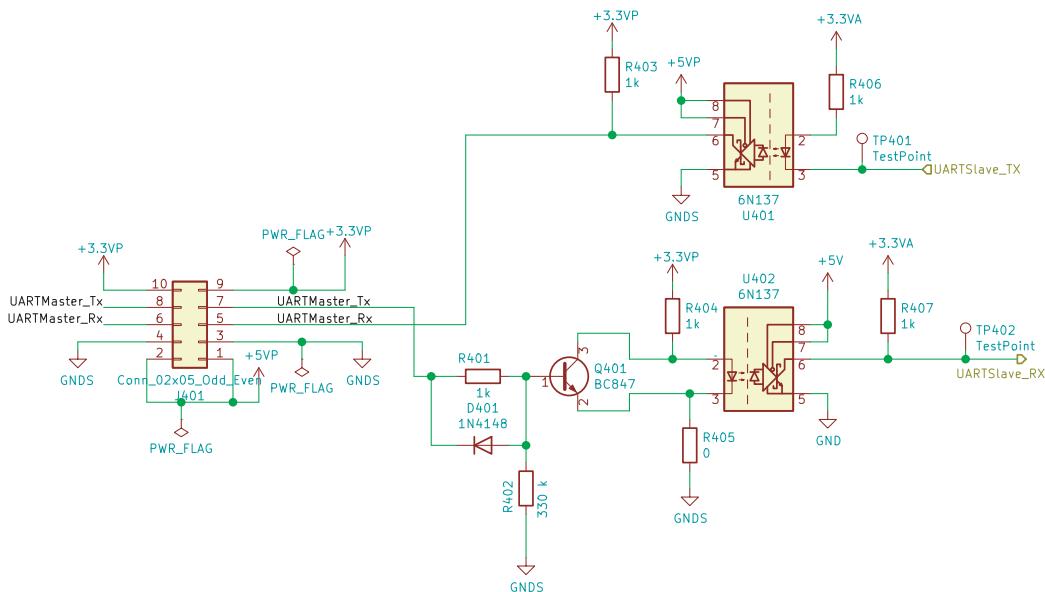


FIGURA 3.6. Diagrama esquemático de la interfaz UART optoaislada.

3.2.4. Entradas y salidas digitales

Del listado de requerimientos surge la necesidad de implementar tres entradas digitales, tres salidas digitales y una salida de alimentación para los módulos adicionales. La figura 3.7 muestra el diagrama esquemático del circuito adaptador

de nivel de una de las entradas digitales, la cual fue diseñada para tolerar niveles de tensión de hasta 30 V y proteger al microcontrolador STM32F108C8T6.

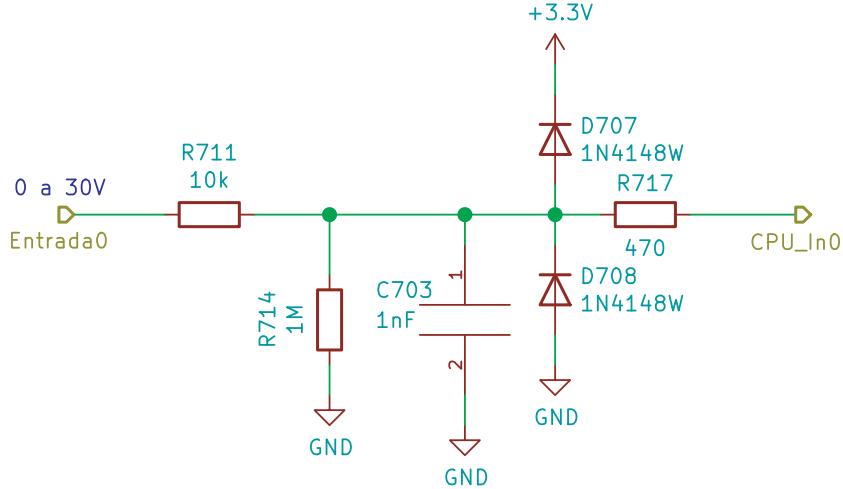


FIGURA 3.7. Diagrama esquemático de una entrada digital.

Para las salidas digitales se escogió una configuración de salida optoacoplada de colector abierto como se ve en la figura 3.8. En esta misma figura, además, se puede ver la salida de alimentación de 220 VAC de los módulos adicionales, esta se implementó con una salida a relé de dos contactos que permite la conexión y desconexión de fase y neutro al mismo tiempo.

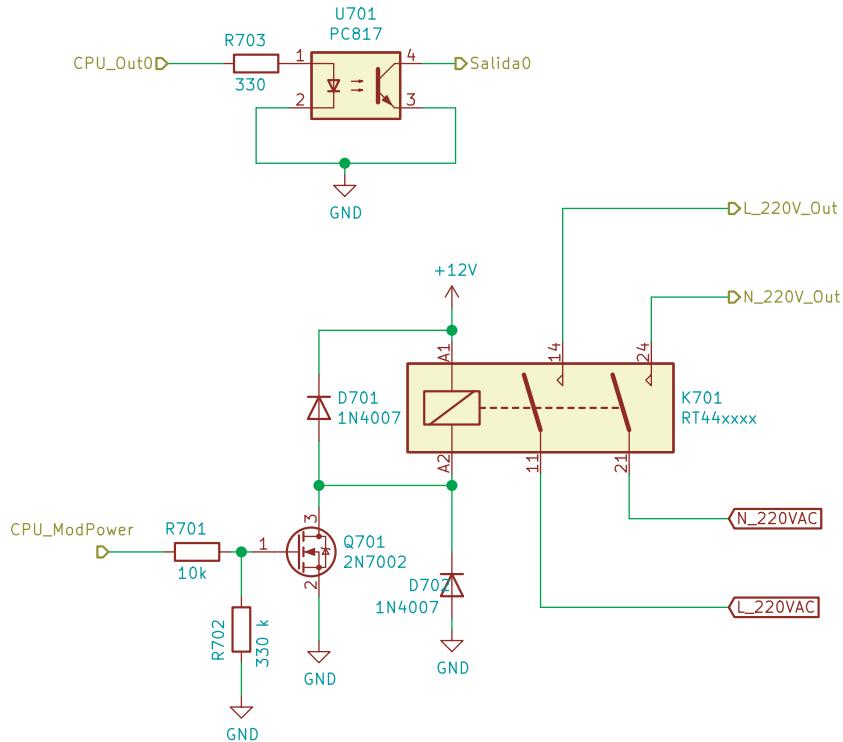


FIGURA 3.8. Diagrama esquemático de una salida digital y la salida de alimentación de 220 VAC.

3.2.5. Salida y entradas analógicas

Cada puerto del módulo principal dispone de una salida analógica con capacidad de generar señales entre 0-10 V. Como el microcontrolador STM32F108C8T6 no posee un conversor digital analógico fue necesario utilizar un conversor externo, el conversor elegido para el diseño fue el MCP4725 de microchip. Este conversor de 12 bits y un canal se comunica con el microcontrolador mediante un bus I2C. Como se puede ver en el diagrama esquemático de la figura 3.9, se diseñó una etapa de amplificación que eleva el nivel de tensión del rango 0 - 3.3 V, que puede entregar el MCP4725, hasta el rango de 0 - 10 V que fue especificado en los requerimientos para la salida analógica.

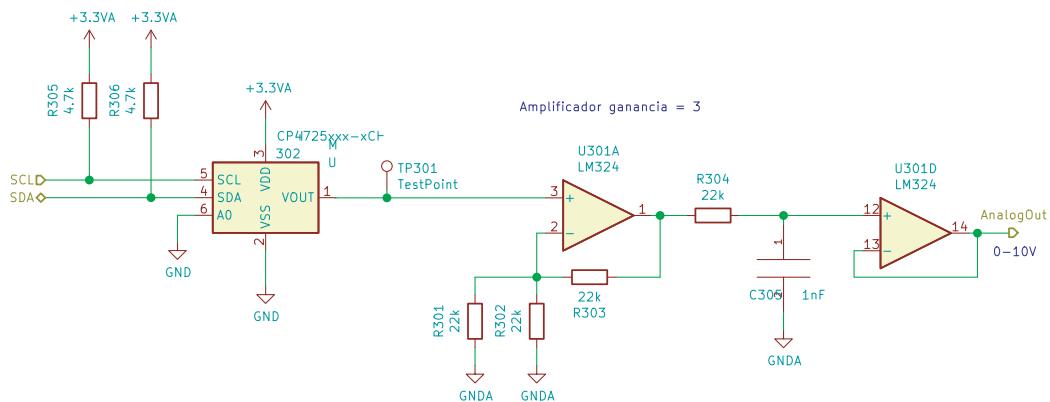


FIGURA 3.9. Diagrama esquemático de la salida analógica de un puerto.

Para las entradas analógicas se utilizaron los dos conversores analógico digitales incluidos en el microcontrolador STM32F103C8T6, que muestran ambas entradas en forma simultánea. Al igual que la salida analógica, fue necesario hacer una adaptación del nivel de tensión de entrada de 0 - 10 V al rango 0 - 3,3 V con un circuito atenuador y un buffer. En la figura 3.10 se puede ver el circuito atenuador implementado. Que además, para proteger al puerto, se diseñó una protección por sobre tensión en la entrada analógica que limita la tensión a un máximo de 12 V.

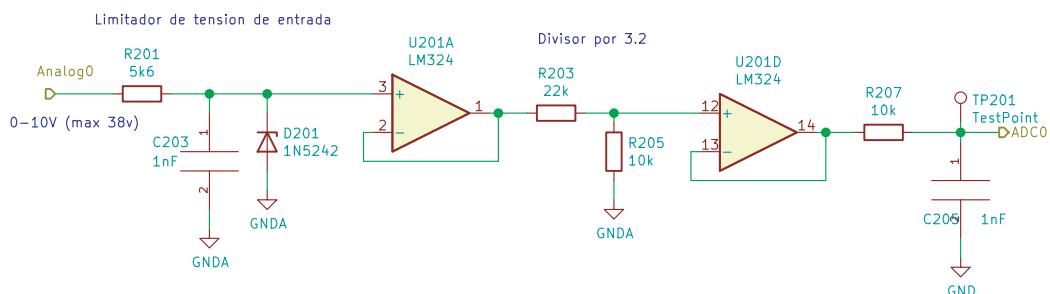


FIGURA 3.10. Diagrama esquemático de una entrada analógica.

3.3. Desarrollo de hardware del módulo prueba de drivers

El módulo test de drivers es un circuito diseñado para realizar la medición de los valores de tensión y corriente en la salida de un driver para iluminación LED. En el diagrama en bloques de la figura 3.11 se puede ver que el módulo está compuesto por cinco etapas que adaptan las señales a los niveles de tensión soportados por los puertos del módulo principal. Además, se puede observar la forma en que se conecta con el módulo principal y la carga.

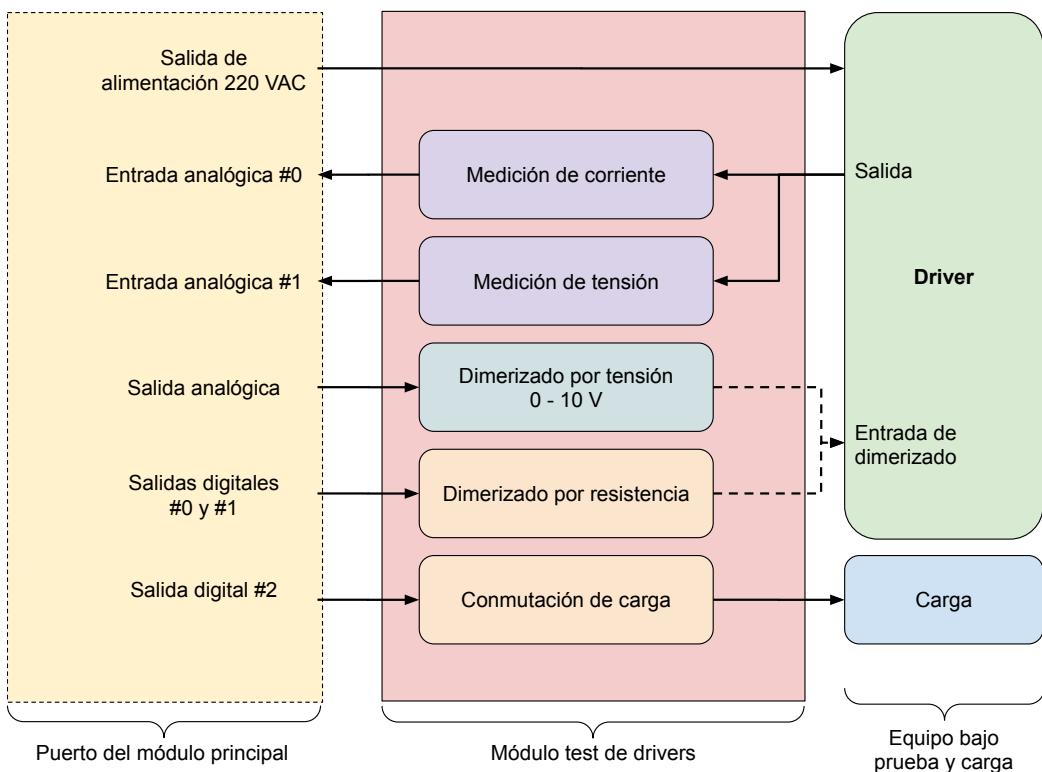


FIGURA 3.11. Diagrama en bloques del módulo prueba de drivers.

Dos de las cinco etapas de este módulo, sirven para mediciones analógicas. La primera etapa, cuyo esquemático se puede ver en la figura 3.12, sirve para la medición de corriente de salida de drivers de LEDs. Esta se diseñó para hacer mediciones entre 0 y 2,5 A de corriente continua entregando a la salida un nivel de tensión proporcional en el rango 0 - 10 V.

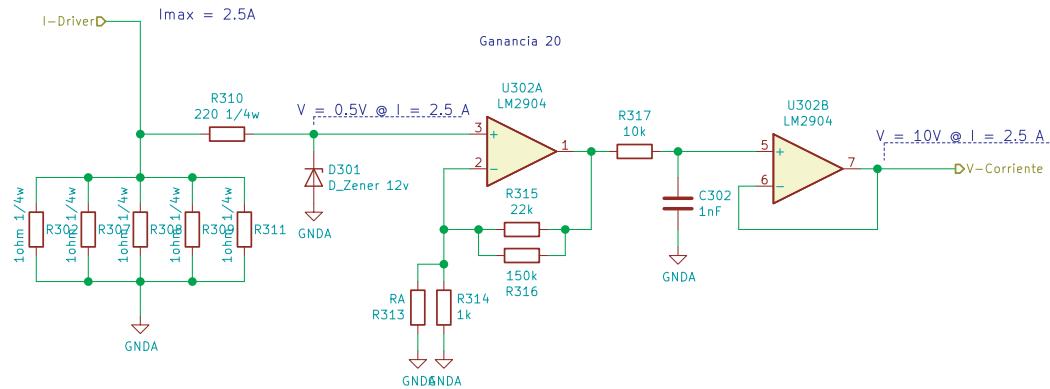


FIGURA 3.12. Esquemático del circuito de medición de corriente de drivers.

La segunda etapa de medición analógica es una etapa de medición de tensión continua de la salida del driver de LEDs. En el esquemático de la figura 3.13, se puede ver el circuito atenuador y separador diseñado para llevar el nivel de tensión del rango 0 - 500 VDC a 0 - 10 V compatible con los puertos del módulo principal.

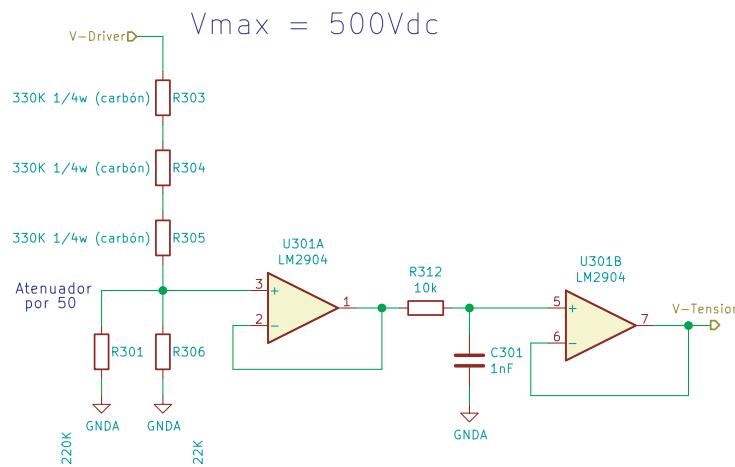


FIGURA 3.13. Esquemático del circuito de medición de tensión de drivers.

Este módulo además posee dos salidas para dimerizar drivers, una de ellas de dimerizado analógico, para la cual se construyó una etapa buffer de ganancia unitaria que separe al módulo principal del driver. La otra salida de dimerizado es una salida discreta, que mediante la combinación de dos relés, comunica resistencias con las que se puede configurar el dimerizado en algunos modelos de drivers. La última etapa de este módulo es una salida tipo on/off a relé para hacer la conexión y desconexión de la carga de los drivers de LEDs.

3.4. Desarrollo de hardware del módulo prueba de temporizadores

El módulo de prueba de temporizadores está formado por dos etapas, la etapa de disparo y la etapa de captura de salida. En la figura 3.14 se observa el diagrama en bloques, la conexión con el módulo principal y con los dos tipos de temporizadores que soporta.

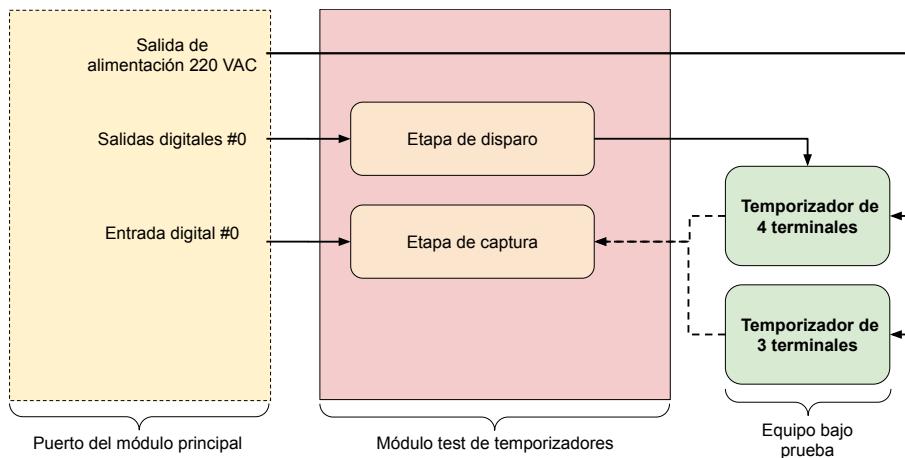
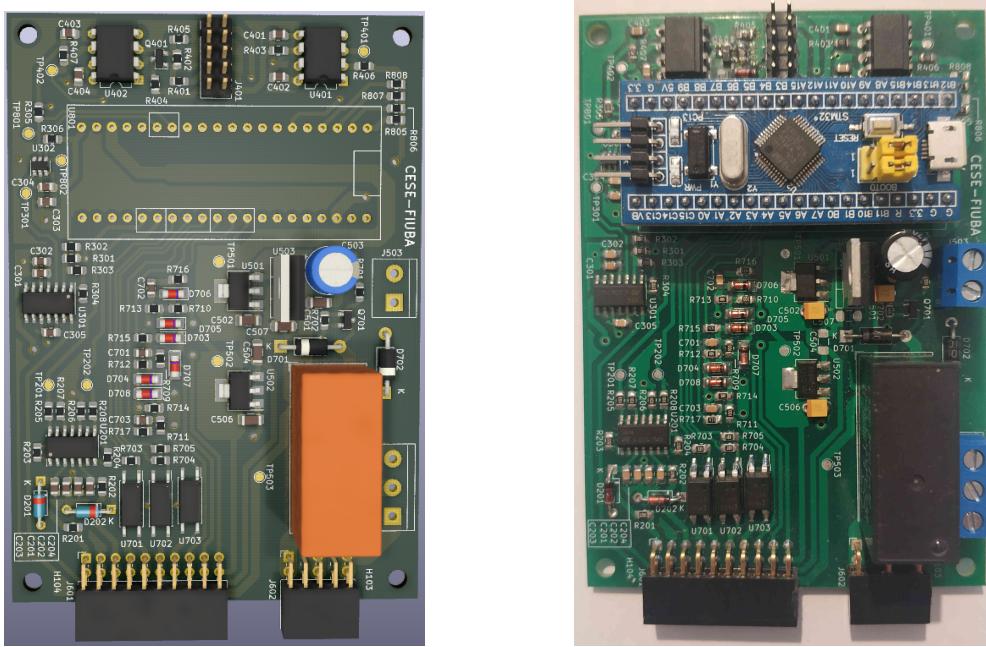


FIGURA 3.14. Diagrama en bloques del módulo prueba de temporizadores.

3.5. Desarrollo de circuitos impresos

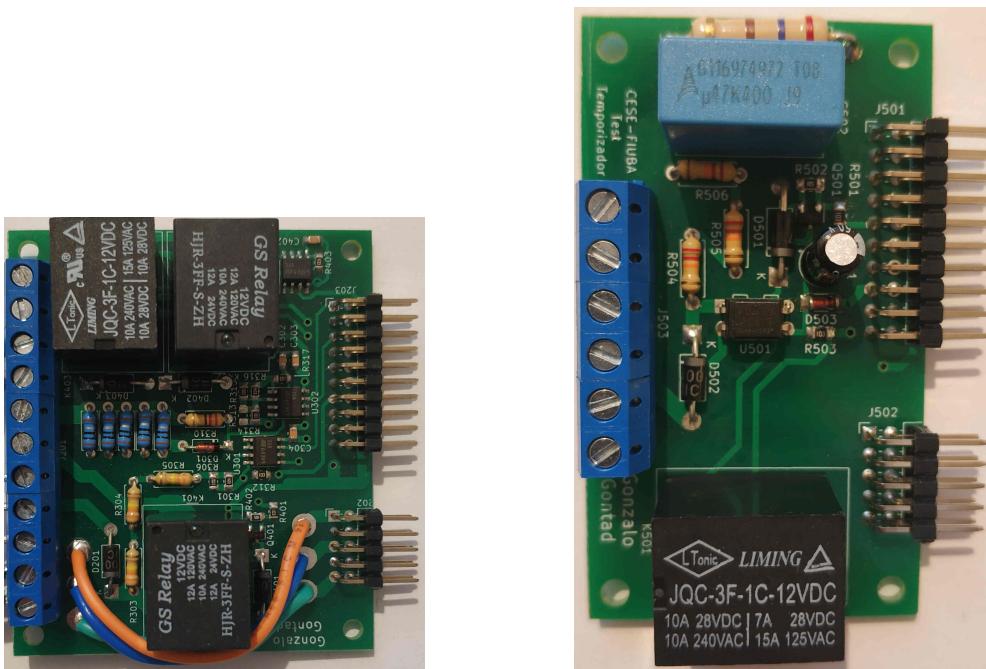
El diseño de los circuitos impresos y los diagramas esquemáticos se realizaron con el software libre KICAD. En total se diseñaron tres modelos, el circuito impreso de un puerto del módulo principal, en la figura 3.15 se puede ver un renderizado del mismo junto con una foto de la placa real, el módulo de prueba de drivers y el módulo de prueba de temporizadores, que se pueden ver en la figura 3.16. Se fabricaron seis unidades de cada uno.



(A) Renderizado en Kicad.

(B) Foto real.

FIGURA 3.15. Renderizado y fotografía de placa de un puerto del módulo principal



(A) Placa prueba de drivers.

(B) Placa prueba de temporizadores.

FIGURA 3.16. Fotografía de la placa del módulo prueba de drivers de LEDs y la placa del módulo prueba de temporizadores de tres y cuatro terminales

3.6. Arquitectura del software

Con la intención de facilitar al lector el entendimiento de la arquitectura del software implementado, se seccionó la documentación según la plataforma de hardware en la que se ejecuta cada pieza de software. Siguiendo con este criterio, podemos diferenciar tres secciones de software a analizar:

- Software ejecutado en los módulos Bluepill STM32 de los puertos.
- Software ejecutado en la placa de desarrollo EDU-CIAA.
- Software ejecutado en la terminal de la interfaz web.

Cada una de estas secciones está comunicada a través de un medio físico y un protocolo. En la figura 3.17 se pueden ver estos módulos de software y el protocolo con el que se comunican.

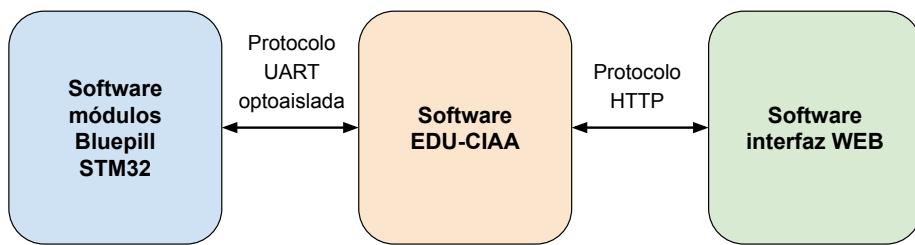


FIGURA 3.17. Protocolos de comunicación entre bloques de software

3.6.1. Arquitectura del software ejecutado en los módulos Bluepill.

El software que se ejecuta en el módulo Bluepill se desarrolló siguiendo una arquitectura del tipo interrupción. Esta arquitectura se utiliza en aplicaciones con eventos que deben ser atendidos en forma rápida y eficiente y esto debe hacerse independientemente de lo que esté haciendo el sistema en ese momento.(1) Teniendo en cuenta la arquitectura elegida en esta aplicación, podemos identificar un bloque de código principal que se ejecuta en un bucle infinito y dos interrupciones que requieren atención. Una de ellas es una interrupción temporal para el muestreo de señales analógicas y la otra está dada por el protocolo de la interfaz UART optoaislada, cuando el maestro genera una interrupción en el esclavo para comunicarse. El diagrama en bloques de la figura 3.18 indica los segmentos de software que se desarrollaron siguiendo esta arquitectura y la forma en que se comunican entre sí.

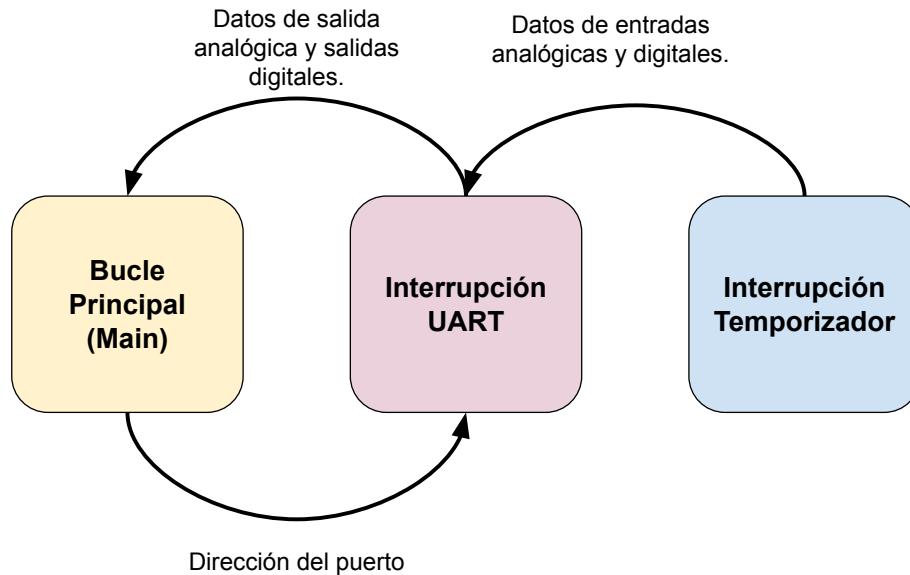


FIGURA 3.18. Diagrama en bloques general del software del módulo Bluepill.

3.6.2. Implementación del software del módulo Bluepill.

Para el desarrollo del software sobre el módulo Bluepill se escogió el entorno de desarrollo STM32CubeIDE de la empresa ST Microelectronics. Este entorno de desarrollo incorpora la herramienta gráfica STM32CubeMX utilizada para la configuración e inicialización de puertos, interrupciones y clock del procesador. Además, STM32CubeIDE ofrece distintas bibliotecas de software, entre ellas la STM32Cube HAL (Hardware abstraction layer, capa de abstracción de hardware), que se utilizó para el manejo de los distintos periféricos del microcontrolador.

A continuación se desglosan las funciones principales de los tres bloques de software del módulo Bluepill. Funciones bucle principal:

- Leer dirección del módulo
- Actualizar datos del DAC mediante el bus I2C.
- Actualizar salidas digitales.

Funciones interrupción por temporizador:

- Actualizar el estado de las entradas digitales.
- Actualizar el estado de las entradas analógicas, colocar en buffer y calcular promedio.
- Armar la trama con los datos de entradas analógicas y digitales que se transmitirán por UART.

Funciones interrupción UART:

- Identificar tramas que llegan por la UART y descartar las que no corresponden al módulo.
- Separar los datos recibidos para que el bucle principal los utilice.

- Transmitir por UART la trama formada en la interrupción del temporizador.

Funciones interrupción por temporizador:

- Actualizar el estado de las entradas digitales.
- Actualizar el estado de las entradas analógicas, colocar en buffer y calcular promedio.
- Armar la trama con los datos de entradas analógicas y digitales que se transmitirán por UART.

Funciones interrupción UART:

- Identificar tramas que llegan por la UART y descartar las que no corresponden al módulo.
- Separar los datos recibidos para que el bucle principal los utilice.
- Transmitir por UART la trama formada en la interrupción del temporizador.

En la figura 3.20 se muestra el diagrama en bloques del software implementado para el bucle principal y la interrupción del timer. Por otro lado, el diagrama en bloques de la figura 3.19 ayuda a comprender la implementación del código 3.1 de la interrupción UART. Allí se puede ver cómo se realiza el procesamiento de cada byte que llega y así separar las distintas partes de la trama.

```

1 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle)
2 {
3     uint8_t auxByte;
4     //Si el primer bit es 1 significa que llego el primer byte de una
5     //trama
6     if (bufferUARTin[receiveIndex]& 0x80){
7         //Chequeo address del dato, si es el propio envio los datos de ADCs
8         //y entradas
9         if (((auxByte=bufferUARTin[receiveIndex]>>4)&0x07)==moduleAddr){
10             receiveIndex = 1;//Primer byte de la trama recibido
11             for (auxByte=0; auxByte < OUTPUT_BUFFER_SIZE; auxByte++){
12                 bufferUARTout[auxByte] = packedDataOut [packedDataIndex][auxByte]
13             };
14         }
15         //Transmito la informacion por la UART
16         if(HAL_UART_Transmit_IT(&huart1 , bufferUARTout , OUTPUT_BUFFER_SIZE
17 )!= HAL_OK){
18             Error_Handler();
19         }
20     }
21     else receiveIndex = 0; //Si el dato no corresponde a este address
22     //reinicio el indice
23 }
24 //Si el primer bit es 0 significa que llego el 2 o 3 byte puede o no
25 //ser para el address actual
26 else{
27     if (receiveIndex == 1){
28         //Ordeno los datos para el DAC
29         DACData[1] = bufferUARTin[1]<<2;
30         DACData[0] = ((bufferUARTin[0]&0x07)<<1)|(bufferUARTin[1]>>6);
31         receiveIndex = 2; //Segundo byte recibido
32     }
33     else{
34         if (receiveIndex == 2)      {
35             //Ordeno los datos de las salidas digitales
36             outData=bufferUARTin[2]|((bufferUARTin[0] & 0x08)<<4);
37         }
38     }
39 }
```

```

31     dataFlag = 1; //Indico que hay datos para enviar al DAC y a las
32         salidas digitales
33         receiveIndex = 0; //Tercer byte recibido
34     }
35
36 //Reinicio la recepcion de otro dato por UART y lo guardo en el buffer
37 if(HAL_UART_Receive_IT(&huart1, (uint8_t *) &bufferUARTin[ receiveIndex
38     ], 1) != HAL_OK){
39     Error_Handler();
39 }
40 }
```

CÓDIGO 3.1. Código del callback de interrupción de recepción por UART.

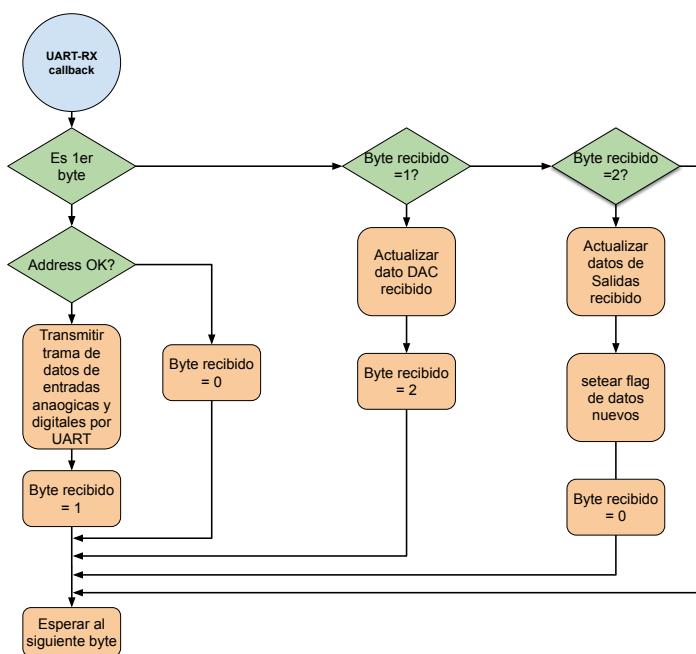


FIGURA 3.19. Diagrama en bloques de la interrupción de recepción por UART.

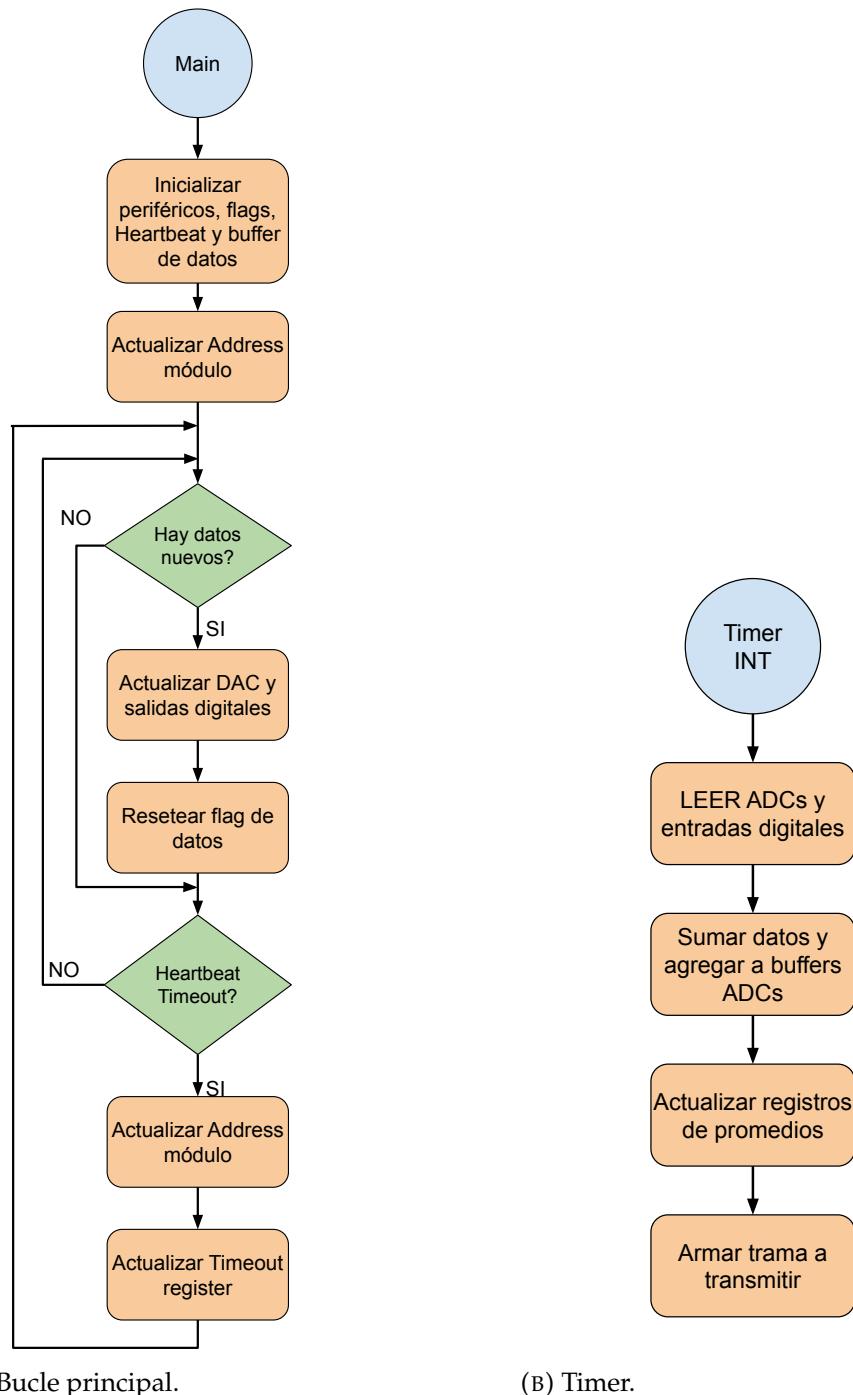


FIGURA 3.20. Diagrama en bloques del bucle principal e interrupción del timer.

3.6.3. Arquitectura del software ejecutado en la EDU-CIAA.

Al momento de elegir la arquitectura del software que se iba a utilizar para esta sección se realizó el siguiente listado de funciones que debían implementarse.

- Montar un servidor web para la interfaz de usuario.
- Correr una terminal para configuración.

- Ejecutar los tests.
- Actualizar el estado de los puertos con una periodicidad de 1 ms.

En esta lista puede apreciarse que se trata de funciones o tareas de diversas características y complejidades que deben ejecutarse en paralelo y en tiempo real. Una forma de resolver este tipo de problemas es mediante el uso de sistemas operativos de tiempo real, es por ello que se escogió FreeRTOS como base para esta sección de software. Luego se creó la estructura en capas de la figura 3.21 con capas de abstracción de hardware, la capa del sistema operativo FreeRTOS, una capa de drivers propios y la capa de aplicación.

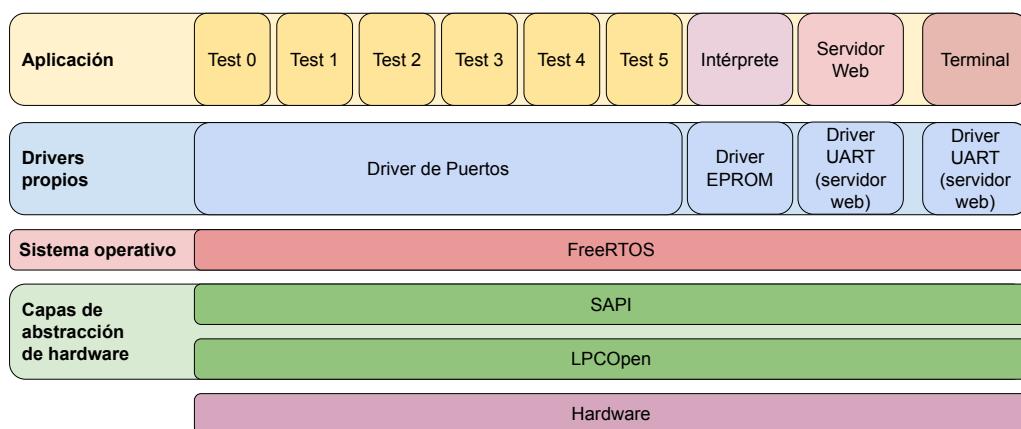


FIGURA 3.21. Modelo de capas del software en la EDU-CIAA.

Analizando en detalle las capas de aplicación y drivers, se puede ver que se incluyeron cuatro bloques de drivers y nueve tareas que a continuación se describen brevemente.

Drivers

- Driver de puertos: Este driver se desarrolló para comunicar la EDU-CIAA con los puertos de conexión vistos en la sección 3.2.2 mediante el protocolo que se estableció para la interfaz UART optoaislada. El driver utiliza un timer del sistema operativo para realizar un polling a los puertos cada 1ms cumpliendo con los requerimientos de taza de muestreo muestreo. Además este driver utiliza colas dos colas por cada puerto, mediante las cuales lee los datos a transferir y envia los datos recibidos a las tareas que los requieren.
- Drive EEPROM: Se utiliza para separa, grabar y leer la memoria EEPROM propia del microcontrolador de la EDU-CIAA. En ella se guardan los parámetros de cada uno de los test y los datos de conexión de la red WIFI.
- Driver UART servidor web y terminal: Son dos instanciaciones del mismo módulos de software. Este driver se desarrollo para permitir la comunicación con la UART del módulo ESP-01 y mediante el uso de colas evitar que el servidor web y la terminal produzcan un bloqueo de tareas.

Tareas

- Terminal: Consiste en una interfaz de usuario básica que utiliza la UART-USB que posee la EDU-CIAA únicamente destinada a la configuración de la

red WIFI y al debugging. Servidor web: Realiza la configuración y control del módulo ESP-01 para la conexión a la red WIFI, el reconocimiento de peticiones HTTP, envía comandos al intérprete y envía las respuestas HTTP.

- Intérprete: Esta tarea se ocupa de interpretar y ejecutar los comandos que llegan de la interfaz de usuario a través del servidor web y además recopila y empaqueta los datos de estado de las pruebas que el servidor web envía a la interfaz de usuario.
- Test 0 - 5: Las seis tareas test son propiamente el software de prueba. Allí residen los distintos tipos de pruebas que se pueden realizar. Cada una de estas tareas está asociada a un solo puerto del módulo principal al que puede controlar.

El esquema de la figura 3.22 detalla el modo en que se comunican las distintas tareas que se ejecutan sobre el sistema operativo.

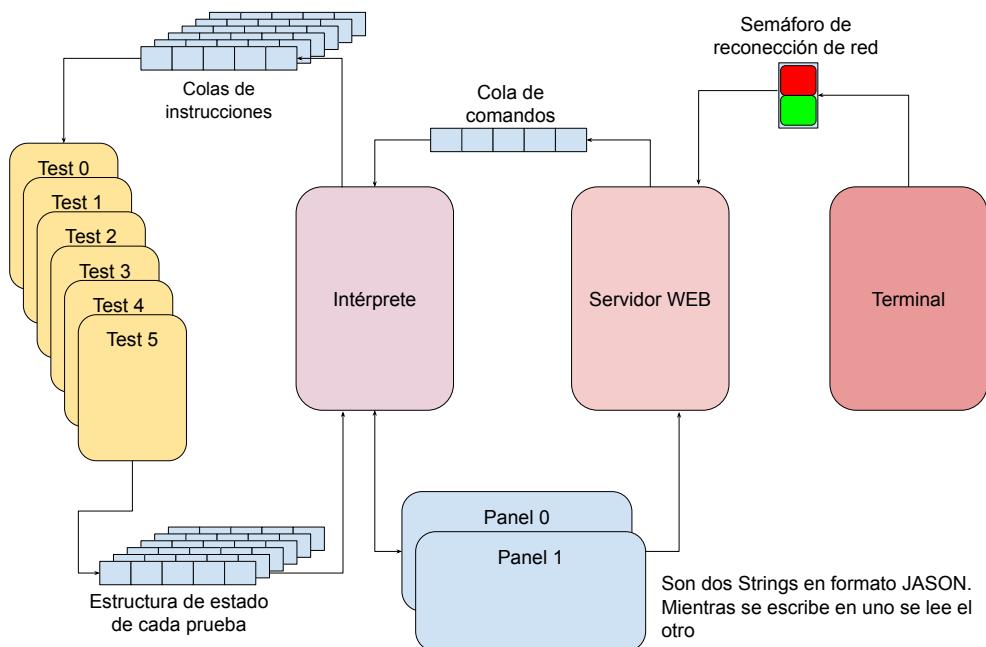


FIGURA 3.22. Esquema de comunicación entre tareas.

3.6.4. Tarea terminal

La tarea terminal fue diseñada con el fin de interpretar comandos recibidos mediante una UART y ejecutarlos. Por tratarse de una interfaz únicamente pensada para la configuración de la conexión a la red WIFI, el set de comandos aceptados se reduce a:

- SSID: Devuelve la identificación SSID actual de la red. También acepta un parámetro opcional para modificar la SSID. Ejemplo: SSID RedFiuba, actualiza el SSID de la red a la que se debe conectar a RedFiuba.
- PASS: Modifica la contraseña de la red. Requiere de un parámetro a continuación que es la contraseña que se desea configurar.

- WIIP: Devuelve la dirección IP actual del dispositivo en la red. También acepta un parámetro opcional para modificar la IP. Ejemplo: WIIP 192.168.0.1, fuerza la IP a la dirección 192.168.0.1.
- RECN: Indica al servidor web que debe iniciar un proceso de reconexión, es necesario ejecutar este comando luego de modificar alguno de los parámetros de conexión a la red.

El funcionamiento de esta tarea fue diagramado en cuatro etapa:

- Recibir datos y crear hash del comando
- Identificar comando
- Capturar parámetro
- Ejecutar el comando

La forma que se escogió para hacer el hash consiste en construir una variable de 32 bits con cuatro bytes donde la posición de cada byte en dicha variable corresponde al orden en que los bytes llegaron por la UART asignada a la terminal. Cada vez que llega un carácter por la UART este desplaza a los anteriores y elimina al más viejo formando un nuevo hash que debe ser identificado, luego capturado el parámetro si hubiese y ejecutado.

3.6.5. Tarea servidor web

En el desarrollo del servidor web se utilizó como base la biblioteca ESP8266 incluida en la sAPI, que permite mediante comandos AT la configuración del módulo ESP-01 y la comunicación utilizando el protocolo HTTP con la interfaz de usuario en un navegador web. Fue necesario adaptar la biblioteca para lo cual se sustituyeron las funciones originales para recepción de datos de la UART por funciones del driver UART propio que evita el bloqueo de tareas. Además se agregaron funciones y servicios que se detallan a continuación.

- Configuración de IP: Se incorporó la opción de configurar el del módulo ESP-01 mediante el comando AT+CIPSTA.
- Parseo request: Se amplió la capacidad de parseo de las request HTTP para identificar el método y el cuerpo.
- Request GET: Se amplió el soporte de la request con métodos GET permitiendo que la carga de la página web de la interfaz de usuario se haga por partes y actualice los datos en pantalla sin necesidad de cargar la página web completa.
- Request POST: Se dio soporte al método POST para enviar datos y órdenes desde la interfaz de usuario a la aplicación.

El diagrama en bloques de la figura 3.23 permite comprender el funcionamiento del servidor web implementado. En este diagrama se puede ver que el servidor inicia con un proceso de configuración y conexión a la red y luego se mantiene en un lazo en el que espera peticiones y las responde. Por otro lado la figura 3.24 muestra un diagrama de transacciones típico para la carga de la página web de interfaz de usuario, en el se puede observar que una vez transferidos todos los datos para construir la página web, solo es necesario hacer peticiones HTTP del

tipo GET /data a las que el servidor responde con los datos de estado de las pruebas en formato JSON y finalmente el navegador actualiza la información en pantalla.

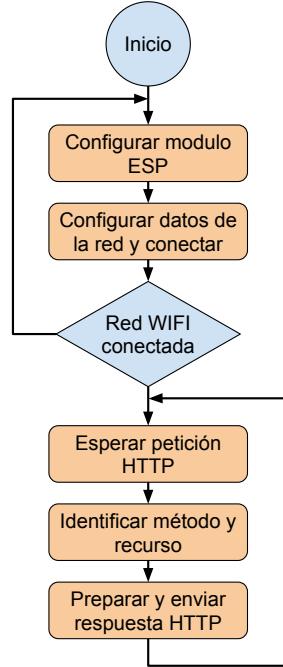


FIGURA 3.23. Diagrama en bloques del servidor web.

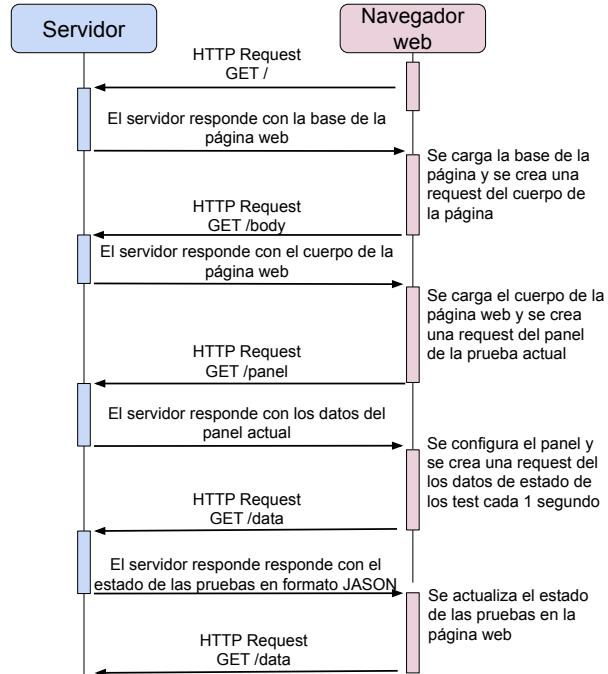


FIGURA 3.24. Diagrama de transiciones entre el servidor web y el navegador.

3.6.6. Tarea intérprete

Esta sección describe detalladamente el desarrollo realizado para implementar las funciones de la tarea intérprete que fueron descritas en la sección 3.6.3. Una de las funciones que el intérprete debe realizar es la de interpretar y ejecutar comandos provenientes del servidor web. En rigor estos comandos son el resultado de peticiones HTTP con método POST que envía el navegador web donde corre la interfaz de usuario. Estas peticiones pueden cambiar el estado de una o más pruebas, pueden cambiar el tipo de prueba en curso o enviar datos de configuración de las pruebas. Luego la secuencia puede resumirse en:

- Se genera la petición HTTP POST accediendo a un recurso.
- El servidor traduce la petición en un comando para el intérprete y lo coloca en una cola de comandos.
- El intérprete captura el comando de la cola, lo interpreta y lo ejecuta.

Todos los comandos que recibe el intérprete están asociados a eventos producidos al presionar un botón de la interfaz de usuario. Estos se pueden clasificar de la siguiente forma. Comandos de botones de base Son comandos que se producen al presionar alguno de los botones de la base de la interfaz de usuario comunes a todos los tipos de pruebas. Estos son:

- Panel anterior: Implica cambiar al tipo de prueba anterior.
- Panel siguiente: Implica cambiar al tipo de prueba posterior.
- Detener Pruebas: Detiene todas las pruebas en curso.

Comandos de botones de panel Son comandos originados al presionar alguno de los botones del panel en la interfaz de usuario en una prueba en particular. Los comandos disponibles son:

- Marcha/Parada: Al recibir este comando el intérprete debe alternar el estado de la prueba. Si estaba en curso la detiene y si estaba detenida la inicia.
- Guardar: Este comando detiene todas la pruebas en curso y guarda los parámetros recibidos para la configuración del test.

La otra función que realiza el intérprete es la de recopilar el estado de cada una de las pruebas, accediendo sus respectivas estructuras de estados, y formar una cadena en formato JSON con estos datos para que esté disponible cuando el servidor web la requiera. La forma de esta cadena se puede ver en el código 3.2

```

1  "panel": N de panel ,
2  "data": [
3      Estado (test 1),
4      Marcha/Parada (test 1),
5      Pasa/No pasa (test 1),
6      Parametro/Medicion 0 (test 1),
7      Parametro/Medicion 1 (test 1),
8      Parametro/Medicion 2 (test 1),
9      Parametro/Medicion 3 (test 1),
10     Guardar (test 1),
11     .
12     .
13     .
14     Estado (test N),
15     Marcha/Parada (test N),

```

```

16     Pasa/No pasa (test N),
17     Parametro/Medicion 0 (test N),
18     Parametro/Medicion 1 (test N),
19     Parametro/Medicion 2 (test N),
20     Parametro/Medicion 3 (test N),
21     Guardar (test N),
22 ]

```

CÓDIGO 3.2. Cadena de datos de panel en notación JSON.

3.6.7. Tareas test

Las tareas test son las responsables de controlar los puertos y ejecutar las distintas secuencias de prueba. Cada una de las seis tareas test es una instancia de un mismo algoritmo. En el código 3.3 se presenta una porción del algoritmo donde se crean las tareas test que ejecutan la misma función testTask y a cada una de estas tareas se le asigna una estructura FSMRegister del tipo testStatet cuya definición se puede ver en el código 3.4. Esta estructura contiene todas las variables asociadas al estado de la prueba, los parámetros de configuración, resultados, la cola de entrada de instrucciones del intérprete y el puerto del módulo principal que fue asignado a dicha tarea.

```

1 //Inicializar el vector de datos y comunicacion de los tests
2 for ( i=0;i<PORTS_NUMBER; i++)
3 {
4     FSMRegisters[ i ].port = ports.port[ i ]; //Asignar colas del puerto
5     FSMRegisters[ i ].test = 0;
6     FSMRegisters[ i ].state = INIT;
7     //Creo la cola por donde llegan las ordenes para la tarea de pruebas
8     FSMRegisters[ i ].testControlQueue = xQueueCreate(CONTROL_QUEUE_LEN,
9         sizeof( testOrder_t ) );
10
11    //Creo la tarea de la prueba y le paso el registro de datos
12    xTaskCreate(
13        testsTask , // Funcion de la tarea
14        ( const char * )"Test" ,
15        configMINIMAL_STACK_SIZE ,
16        ( void * )&FSMRegisters[ i ], // Registro de estado de la tarea .
17        tskIDLE_PRIORITY+1 ,
18        0
19    );
20 }

```

CÓDIGO 3.3. Algoritmo de creación de tareas test.

```

1 typedef struct
2 {
3     uint8_t test;      //Test en curso
4     uint8_t state;    //Estado
5     uint32_t result[RESULT_NUM]; //resultados de la prueba
6     uint8_t pasa;     //Pasa o no pasa
7     uint32_t *param;  //parametros de la prueba ( entrada )
8     QueueHandle_t testControlQueue; //Cola de instrucciones
9     portsData_t port; //Puerto
10
11
12 //Variables para calculos intermedios
13     uint8_t i;        // numero de iteracion

```

```

14     uint8_t adcSamples; //numero de samples de ADC
15     uint32_t tickRegister; //Captura del Tick counter .
16     uint32_t ADC_1; //Suma de muestras CH0
17     uint32_t ADC_2; //Suma de muestras CH1
18     //Entradas digitales
19     uint16_t in0Sum;
20     uint16_t in1Sum;
21     uint16_t in2Sum;
22
23 } testState_t;

```

CÓDIGO 3.4. Estructura de estado del test.

A través de la cola de instrucciones, el intérprete puede enviar órdenes a la tarea test. Esta ordenes están compuestas de dos parámetros: prueba a realizar y estado de la prueba que se desea forzar. De los posibles estados de las pruebas el intérprete sólo utiliza dos start y stop. En cuanto a los tipos de pruebas se implementaron dos, una para prueba de temporizadores y una para prueba de drivers. Además se desarrolló una tercera opción para la calibración de la prueba de drivers que únicamente presenta en pantalla las mediciones de los ADC. Cada una de las pruebas se desarrolló mediante máquinas de estados finitos (MEF). El diagrama de la figura 3.25 es una representación de la máquina de estados implementada para la prueba de drivers. A continuación se describen cada uno de sus estados

- Init: En este estado se inicializan la salida analógica y todas las salidas digitales del puerto asignado a un valor que garantice la manipulación segura del equipo a probar. Aquí se espera hasta que llegue la instrucción de iniciar. El mensaje en pantalla es INICIO.
- Start: Se configura el nivel de dimerizado. Por este estado se pasa tres veces para realizar mediciones con tres niveles de dimerización. El mensaje en pantalla es Probando.
- PoweOn: Activa la alimentación y espera 5 segundos a que el driver se estabilice.
- Measure: Luego de que el driver se estabilizó se hacen las mediciones de tensión y corriente. Aquí también se evalúan las mediciones. El mensaje en pantalla es Probando.
- Stop: Al igual que el estado Init se llevan todas las salidas del puerto a condición segura, pero en este estado el mensaje en pantalla es FIN.

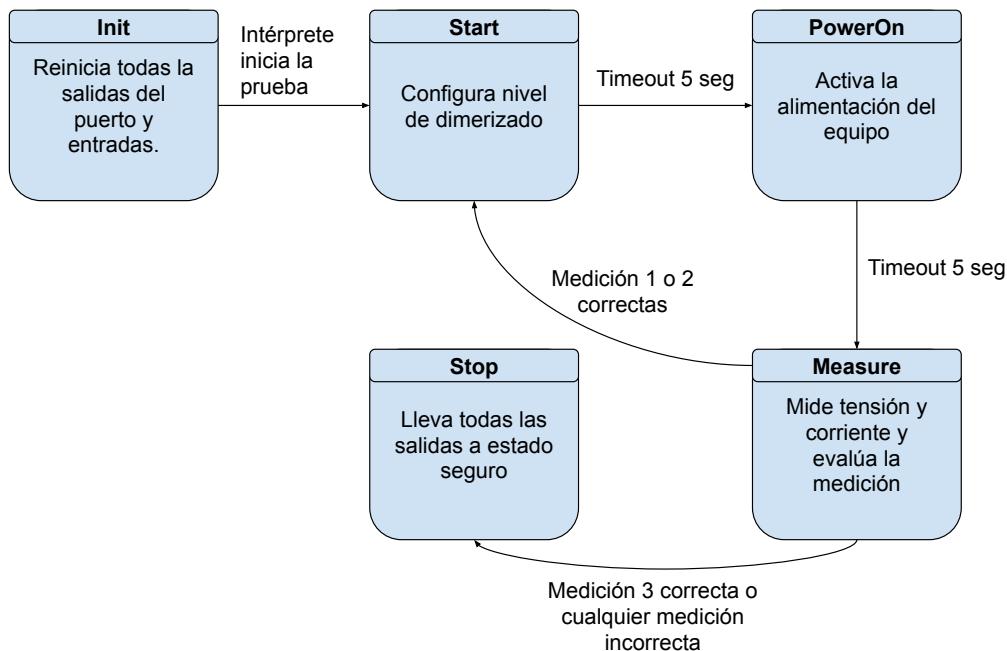


FIGURA 3.25. Diagrama de estados de la MEF del test de drivers.

En el caso de la prueba de temporizadores la máquina de estados implementada puede resumirse en los siguientes estados:

- Init: Inicializa todas las salidas en un valor seguro para la operación por parte del usuario. El mensaje en pantalla es INICIO.
- Start: Se alimenta al equipo bajo prueba. El mensaje en pantalla es Configurar temporizador.
- Trigger: genera un pulso de disparo del temporizador. El mensaje en pantalla es Probando.
- WaitON: Se espera a que el temporizador se encienda durante treinta segundos. El mensaje en pantalla es Probando.
- WaitOFF: Se espera a que el temporizador se apague durante el tiempo máximo configurado en pantalla.
- CheckTime: Se evalúa el tiempo medido. El mensaje en pantalla es Probando.
- Stop: Se llevan las salidas al valor seguro de operación. El mensaje en pantalla es FIN.

Todos estos estados y sus transiciones pueden verse en el diagrama de la figura 3.26.

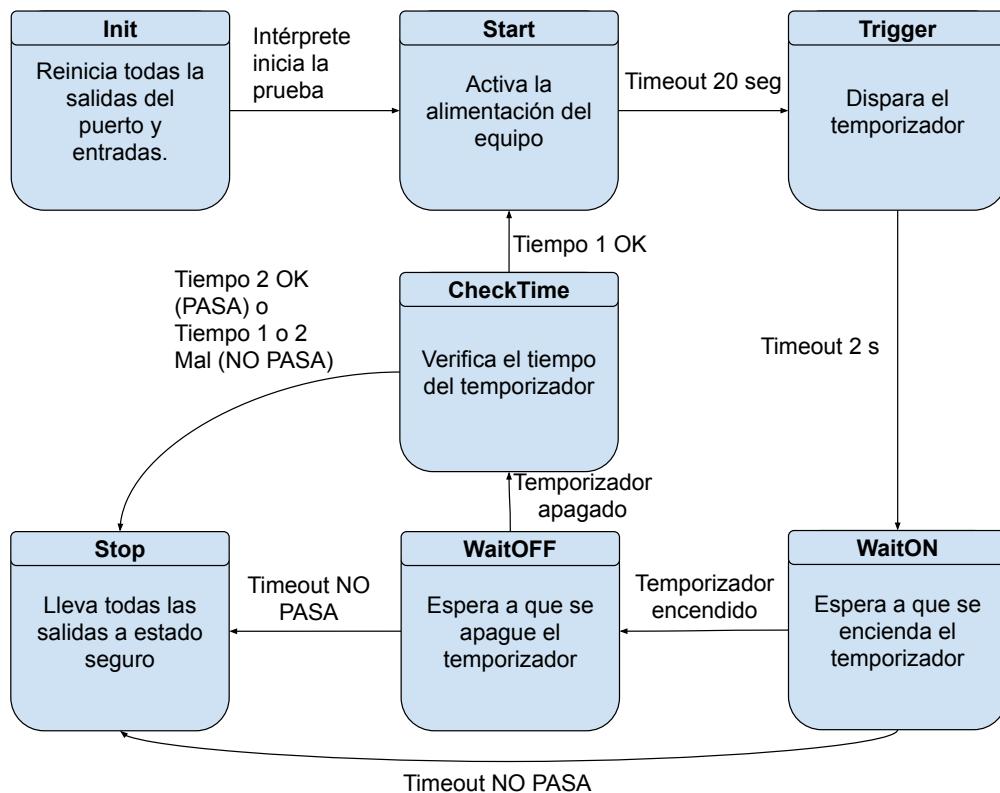


FIGURA 3.26. Diagrama de estados de la MEF del test de temporizadores.

3.6.8. Interfaz de usuario

La interfaz de usuario es el medio por el cual el operario puede controlar el equipo probador. Esta interfaz fue desarrollada de acuerdo con el requisito de que fuera accesible mediante un navegador web utilizando un dispositivo conectado a la misma red. Por tratarse de una interfaz que se presenta como una página web fue necesario el uso de las tecnologías de desarrollo propias de la plataforma. En particular las dos tecnologías que más se utilizaron fueron JavaScript y HTML, en las que no se tenía ninguna experiencia antes de realizar el trabajo. Desde el inicio del desarrollo, la página web de la interfaz, fue pensada para funcionar en forma de bloques, de modo que no fuera necesario cargar toda la página cada vez que algún valor indicado cambiase. Así se resolvió hacer un seccionamiento en tres partes:

- **Base:** En la base se encuentran los botones de control comunes a todas las pruebas. La figura 3.27 es una captura de la base de la interfaz en el navegador web y se pueden ver el botón Detener Pruebas, los botones Panel anterior y panel siguiente y la indicación del panel actual. Cada uno de estos botones al ser presionados le indican al servidor, mediante una petición HTTP, que debe producirse una acción. Cada una de estas acciones ya fue descrita en la sección 3.6.6.
- **Panel:** El panel es el espacio donde se visualizan todas las indicaciones de estado y los parámetros de configuración de la prueba en curso. Aquí también se ubican los botones de Marcha/Parada y Guardar. Existe un panel

distinto para cada tipo de prueba. Las figuras 3.28, 3.29 y 3.30 son capturas de los paneles de las pruebas de drivers, temporizadores y el panel de calibración de ADCs respectivamente.

- Datos: Toda la información sobre el estado de las pruebas y los parámetros de configuración son recibidos por separado del panel y la base. Estos datos, que el servidor envía en formato JSON, son solicitados periódicamente y lograr así un refresco cada un segundo.

En la sección 3.6.5 se vio el proceso de carga de la página web desde el lado del servidor. Si se analiza del lado del navegador web se puede reconocer la siguiente secuencia:

- El navegador solicita la página web mediante la petición GET / .
- El servidor responde con una página web en blanco, un conjunto de scripts para la carga de las distintas secciones y una instrucción inicial de ejecutar el script de carBody().
- Al ejecutar el script carBody() se crea un objeto XMLHttpRequest para enviar una petición HTTP GET /body, a la que el servidor responde con la sección Base de la interfaz.
- Una vez cargada la Base se ejecuta el script carPan() que envía la petición GET /panel. A esta petición el servidor responde con el panel correspondiente al test actual.
- Una vez cargado el panel, crea un timer de 1 segundo que periódicamente ejecuta el script carDat(). Este script envía la petición GET /data al servidor, que responde con los datos del panel actual. Luego se actualizan los datos en pantalla.



FIGURA 3.27. Captura de la base de la interfaz de usuario.

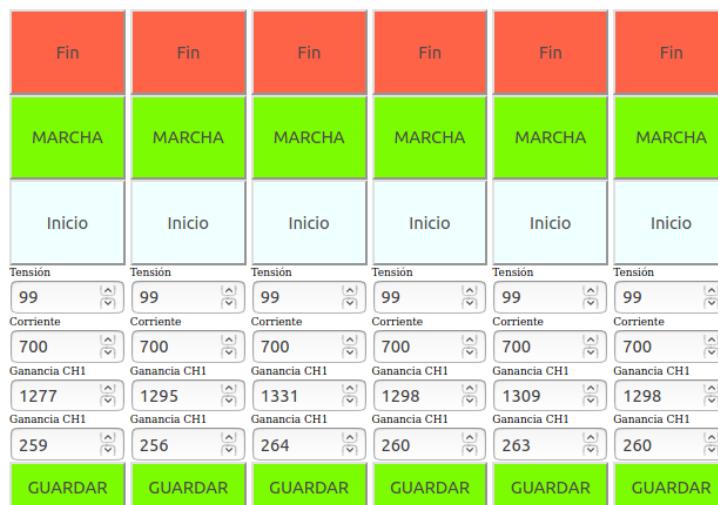


FIGURA 3.28. Captura del panel de test de drivers.



FIGURA 3.29. Captura del panel de test de temporizadores.



FIGURA 3.30. Captura del panel de calibración de ADCs.

Capítulo 4

Ensayos y Resultados

En este capítulo se presentan los ensayos realizados para validar el funcionamiento del equipo desarrollado.

4.1. Ensayos de hardware

A lo largo del desarrollo de este trabajo se implementaron diversas piezas de hardware. Algunas de ellas se trataban simplemente de etapas de adaptación de niveles de señales y otras más complejas como, los puertos del módulo principal, representaron un desafío y es por esto que merecen una mención detallada de los ensayos que fueron realizados sobre ellos. En particular el punto clave en este trabajo fue el hardware de la interfaz UART optoaislada de la que se ha hablado ampliamente es esta memoria y además fue sometida a prueba. Otro ensayo menos complejo, pero que aún así merece ser presentado en esta memoria es el ensayo de calibración de ADCs, ya que de ello depende el correcto funcionamiento de las pruebas de drivers.

4.1.1. Ensayo de UART optoaislada

Este ensayo buscó someter a un funcionamiento intensivo de la misma. Para ello se montó el banco de pruebas cuyo diagrama de conexión se presenta en la figura 4.1. En este banco de pruebas se utiliza un adaptador USB-RS232 conectado a una computadora, cumpliendo la función de maestro en el protocolo establecido en la sección 3.2.3, y los puertos del módulo principal cumplen la función de esclavos para la que fueron diseñados.

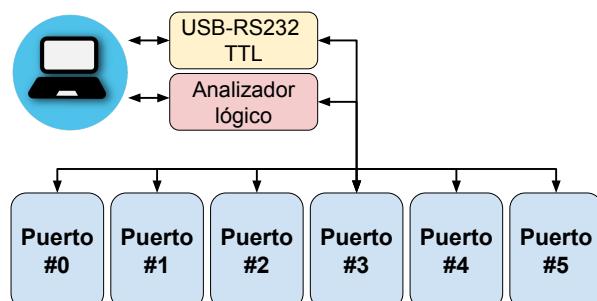
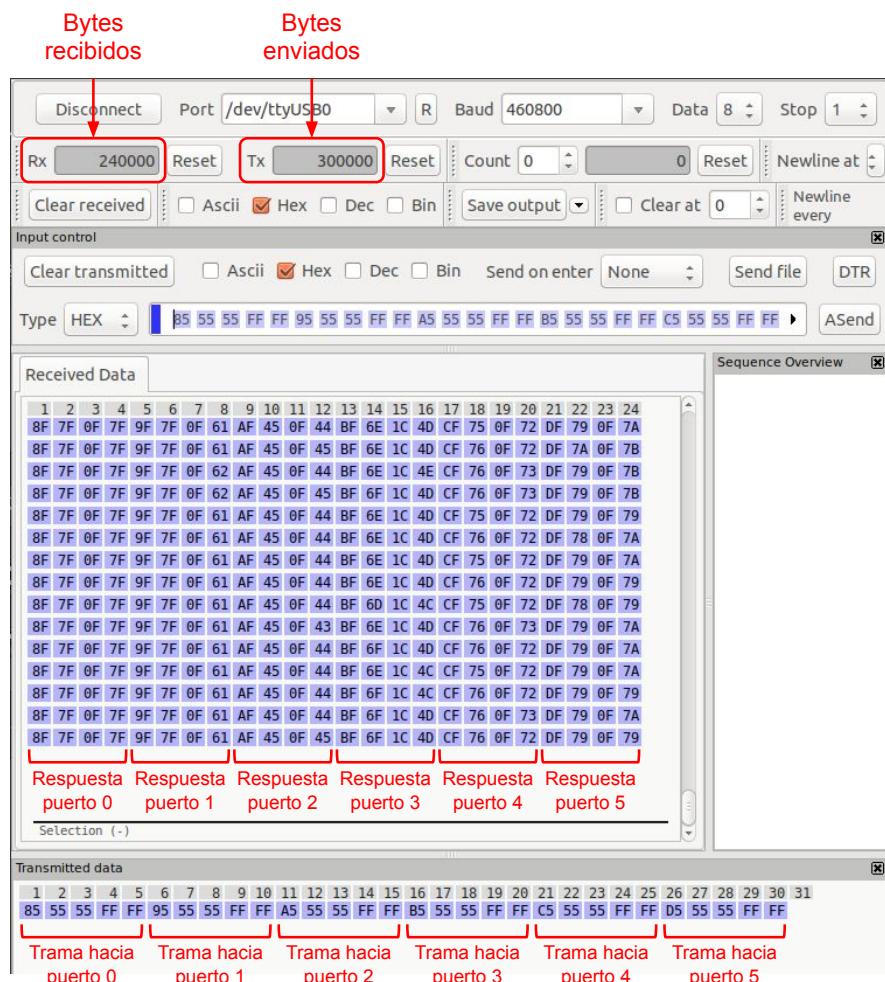


FIGURA 4.1. Diagrama del banco de pruebas de la interfaz UART optoaislada.

El ensayo de esta interfaz consiste en generar seis tramas de 5 bytes cada una acordes al protocolo, de modo que cada trama está dirigida a uno de los puertos y solo ese puerto responda a dicha trama. Este proceso se repitió 10.000 veces consecutivamente utilizando la terminal HTerm que permite hacer este tipo de automatización y se analizaron las respuestas de cada uno de los puertos. En la figura 4.2 se puede ver una captura de la aplicación HTerm con el resultado del ensayo. Allí se puede ver en primera instancia que fueron enviados 300.000 bytes que corresponden a las seis tramas de 5 bytes enviadas 10.000 veces. En la imagen, además, está resaltado el número de bytes recibidos que totalizan 240.000, estos corresponden a los 4 bytes de la respuesta de cada uno de los 6 puertos. Con este breve análisis se puede afirmar que durante la comunicación el volumen de datos fue el correcto, pero no se puede asegurar que las respuestas a cada trama enviada por el maestro la haya respondido el módulo adecuado. Es por eso que debieron analizarse las respuestas recibidas. Por tratarse de 10.000 respuestas por cada módulo no es viable analizar una por una, entonces se optó por verificar la dirección del puerto correspondiente a cada respuesta, que se puede ver en el primer byte de cada una. Así se puede ver que las direcciones de las respuestas se repiten cada 24 bytes y afirmar que los puertos responden únicamente a las tramas que le corresponden.



En la figura 4.1 también se puede ver la conexión de un analizador lógico. Esto se debe a que una parte del ensayo consiste en corroborar el correcto desempeño del protocolo. Un punto importante del protocolo es el agregado a la trama que envía el maestro de dos bytes que no transportan información. Estos bytes extra fueron incluidos con el único fin de evitar la superposición entre las respuestas de los puertos. La figura 4.3 es una captura, realizada con un analizador, de una trama enviada y su respuesta. Allí se puede apreciar que gracias al agregado de dos bytes vacíos se logra tener un espacio libre antes de que otro módulo pueda responder.

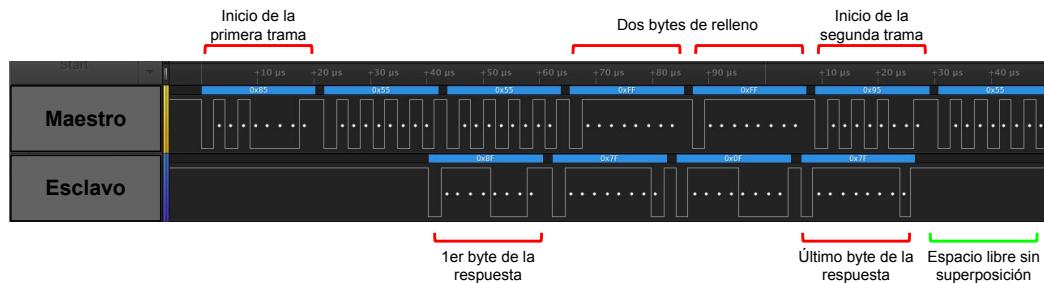


FIGURA 4.3. Captura de la comunicación maestro-esclavo.

4.1.2. Calibración de ADCs

Si bien este trabajo no consiste en el desarrollo de un instrumento de medición y no se establecieron requerimientos de precisión, es una buena práctica caracterizar las mediciones que realiza el equipo y si es posible hacer correcciones. Los ADCs del módulo principal son los principales encargados de las mediciones analógicas que realiza el equipo. Por ello se decidió realizar una contrastación de las mediciones de estos conversores con un instrumento patrón. Para hacer este ensayo se montó el banco de pruebas de la figura 4.4

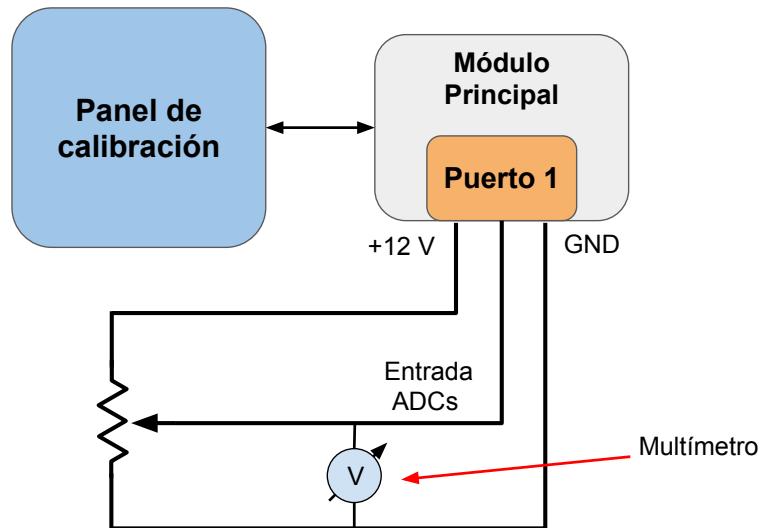


FIGURA 4.4. Banco de pruebas para la calibración de ADCs.

El ensayo consiste en establecer la misma tensión de entrada a los dos canales analógicos de cada puerto y comparar la medición indicada por el multímetro con la indicación en la interfaz de usuario. El multímetro que se utilizó para la contrastación es un multímetro Fluke 107 con su calibración de fábrica del año 2016. En la figura 4.5 corresponde a la calibración del canal analógico 0 de los 6 puertos. En primera instancia se parece mucho al comportamiento lineal esperado, pero si se presta atención al gráfico de la figura 4.6 podemos ver la zona de mediciones más cercanas al cero ampliada. En este gráfico se aprecia mejor la diferencia que hay entre las mediciones de los ADCs y la recta ideal de medición. Esta diferencia se mantiene casi constante en todo el rango. Otra cosa que se puede ver en el gráfico es que para tensiones por debajo de los 200 mV la curva se torna no lineal. Estas dos diferencias con la curva ideal tienen una misma causa y es un error de Offset en la medición posiblemente producido por una diferencia de potencial entre la masa del ADC y la del amplificador de entrada.

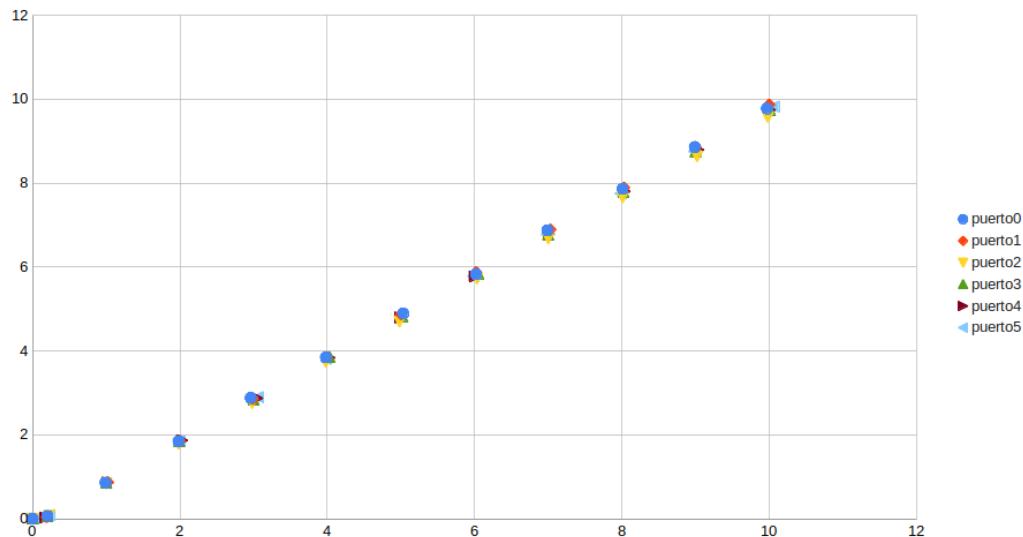


FIGURA 4.5. Calibración de ADCs.

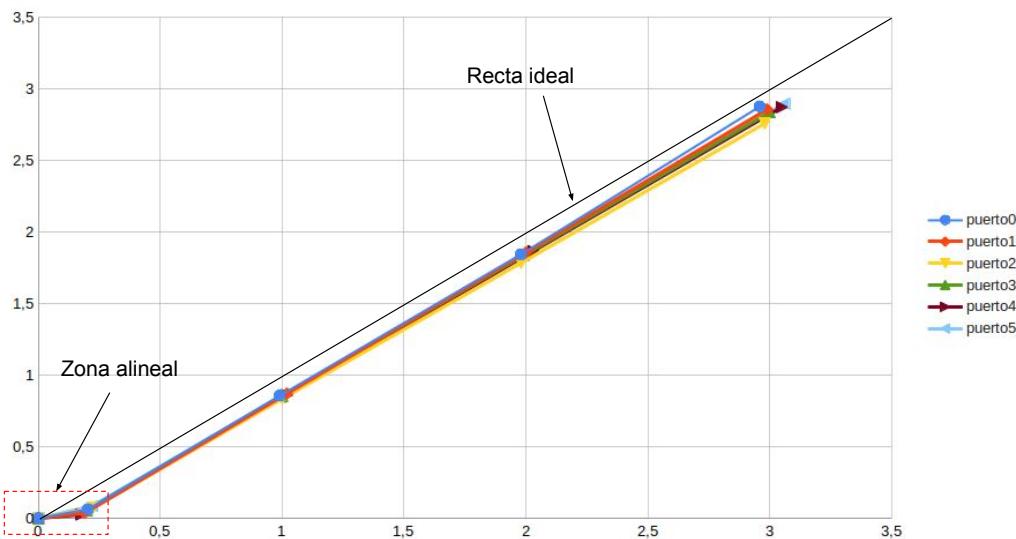


FIGURA 4.6. Calibración de ADCs, zoom cerca del cero.

Para mitigar los efectos del offset, se hizo una corrección por software. Luego se repitió el ensayo obteniendo el gráfico de la figura 4.7. En este nuevo ensayo se ve que la diferencia entre la medición de los ADCs y la recta ideal se redujo, pero la alinealidad en la parte baja persiste y solo podría corregirse por hardware. Sin embargo esto no representa un problema ya que todas las mediciones que se realizan en las pruebas están por encima de los 200 mV.

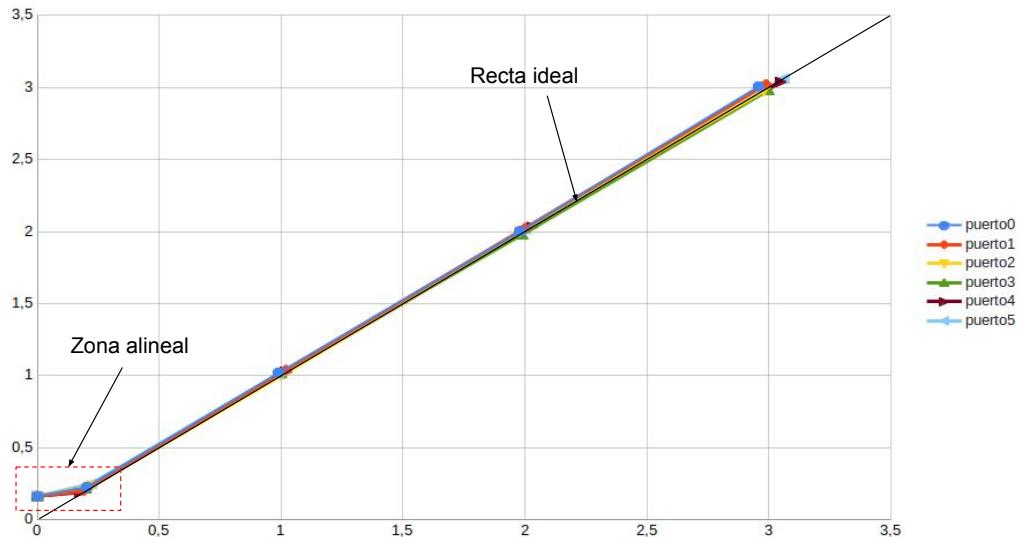


FIGURA 4.7. Calibración de ADCs luego de corrección de offset por software

4.2. Pruebas de integración

Las pruebas de integración buscar ensayar al equipo en forma integral. De este modo se prueba el correcto comportamiento e interacción entre las partes que componen al sistema en distintas condiciones de uso posibles. Una forma de hacer este tipo de pruebas es mediante la ejecución de casos de uso. Dos casos de uso muy importantes son el de la prueba de temporizadores y el de prueba de driver. Las tablas 4.1 y 4.2 indican paso a paso la secuencia de ambos casos de uso. En el caso de la prueba de temporizadores como precondición se requiere que el temporizador esté conectado al banco de pruebas. Para el ensayo se escogió un temporizador marca Secuen, modelo TGE de cuatro terminales. Por otro lado, en el caso de la prueba de drivers se utilizó un driver marca Meanwell, modelo ELG-150-C700B conectado a una carga LED de 70 W a 700 mA. Los dos casos de uso se pusieron a prueba y verificaron cada uno de los pasos satisfactoriamente.

TABLA 4.1. Caso de uso de prueba de drivers

Título	Descripción	Verifica
Nombre	Prueba de Drivers	
Breve descripción	Ejecución de la prueba de drivers para luminarias LED	
Actor principal	Operario	
Disparadores	Presionar en pantalla el botón de Marcha	
Flujo de eventos	1. Se presiona en pantalla el botón de Marcha. 2. El banco de pruebas alimenta el driver y se dimeriza al 10 % 3. Se miden los valores de tensión y corriente del driver 4. Si las mediciones están dentro de los valores de aceptación, se dimeriza al 50 % 5. Se miden los valores de tensión y corriente del driver 6. Si las mediciones están dentro de los valores de aceptación, se dimeriza al 100 % 7. Se miden los valores de tensión y corriente del driver 8. Si las mediciones están dentro de los valores de aceptación, se indica en pantalla PASA 9. Esperar nueva prueba.	SI SI SI SI SI SI SI SI SI
Flujo Básico	Del punto 3, 5 o 7 del flujo básico	
Flujo Alternativo	4A. Si las mediciones no están dentro de los valores de aceptación se aborta la prueba. 5A. Se indica en pantalla NO PASA Se regresa al punto 9 del flujo básico	SI SI SI
Requerimientos especiales		
Precondiciones	El driver debe estar conectado al banco de pruebas.	SI
Poscondiciones	Se debe desconectar la alimentación del driver (por software) para poder operar en forma segura.	SI

TABLA 4.2. Caso de uso de prueba de temporizadores

Título	Descripción	Verifica
Nombre	Prueba de Temporizadores	
Breve descripción	Ejecución de la prueba de temporizadores	
Actor principal	Operario	
Disparadores	Presionar en pantalla el botón de Marcha	
Flujo de eventos		
Flujo Básico	1. Se presiona el botón (en pantalla) de Marcha 2. Ajustar el temporizador al tiempo T1 (mínimo) cuando se indique en pantalla. 3. El banco de pruebas activa el temporizador o solicita presionar el pulsador. 4. El temporizador activa su salida. 5. Inicia la cuenta de tiempo de encendido. 6. Cuando finaliza el tiempo del temporizador, si el tiempo está dentro del rango definido se indica ajustar el temporizador al tiempo T2 (máximo) 7. El banco de pruebas activa el temporizador o solicita presionar el pulsador. 8. Inicia la cuenta de tiempo de encendido. 9. Cuando finaliza el tiempo del temporizador, si el tiempo está dentro del rango definido se detiene la prueba. 10. Se indica en pantalla PASA. 11. Esperar nueva prueba.	SI SI SI SI SI SI SI SI SI SI SI
Flujo Alternativo	Del punto 5 o del punto 8 del flujo básico 6A. Cuando finaliza el tiempo del temporizador, si el tiempo encendido no está dentro de los valores aceptables se aborta la prueba. 7A. Se indica en pantalla NO PASA Se regresa al punto 11 del flujo básico	SI SI
Requerimientos especiales	-	
Precondiciones	El temporizador debe estar conectado al banco de pruebas.	SI
Poscondiciones	Se debe desconectar la alimentación del temporizador (por software) para poder operar en forma segura.	SI

Capítulo 5

Conclusiones

5.1. Conclusiones generales

La idea de esta sección es resaltar cuáles son los principales aportes del trabajo realizado y cómo se podría continuar. Debe ser especialmente breve y concisa. Es buena idea usar un listado para enumerar los logros obtenidos.

Algunas preguntas que pueden servir para completar este capítulo:

- ¿Cuál es el grado de cumplimiento de los requerimientos?
- ¿Cuán fielmente se pudo seguir la planificación original (cronograma incluido)?
- ¿Se manifestó algunos de los riesgos identificados en la planificación? ¿Fue efectivo el plan de mitigación? ¿Se debió aplicar alguna otra acción no contemplada previamente?
- Si se debieron hacer modificaciones a lo planificado ¿Cuáles fueron las causas y los efectos?
- ¿Qué técnicas resultaron útiles para el desarrollo del proyecto y cuáles no tanto?

5.2. Próximos pasos

Acá se indica cómo se podría continuar el trabajo más adelante.