



MAESTRÍA EN SISTEMAS EMBEBIDOS

MEMORIA DEL TRABAJO FINAL

Emulador de la placa EDU-CIAA

Autor:

Esp. Ing. Jenny Chavez

Director:

Dr. Ing. Pablo Gomez (FIUBA)

Codirector:

Mg. Ing. Eric Pernía (UNQ, FIUBA)

Jurados:

Mg. Ing. Gonzalo Sánchez (FF.AA, FIUBA)

Mg. Ing. Iván Andrés León Vásquez (INVAP)

Ing. Juan Manuel Cruz (FIUBA/UTN)

*Este trabajo fue realizado en la Ciudad Autónoma de Buenos Aires,
entre agosto de 2019 y agosto de 2023.*

Resumen

La presente memoria describe el diseño e implementación de una plataforma web que emula la placa EDU-CIAA-NXP y permite desarrollar en lenguaje C sin la necesidad de tener la placa real. La interfaz de la plataforma permite escribir, probar y verificar aplicaciones rápidamente de una manera amigable y nace como una herramienta de aprendizaje para ser parte del Proyecto CIAA.

Para la realización de este trabajo fueron fundamentales los conocimientos relacionados al diseño de software, gestión de la tecnología e innovación, implementación de manejadores de dispositivos, sistema operativo de tiempo real, desarrollo de aplicaciones sobre sistemas operativos de propósito general y testing de sistemas embebidos.

Agradecimientos

Al Director de este trabajo Dr. Ing. Pablo Martín Gomez y al Codirector Mg. Ing. Eric Pernia.

Índice general

Resumen	I
1. Introducción general	1
1.1. Motivación	1
1.2. Descripción técnica-conceptual	1
1.3. Estado del arte	2
1.4. Propósito y Alcance	6
1.4.1. Propósito	6
1.4.2. Alcance	7
2. Introducción específica	9
2.1. Plataforma web	9
2.1.1. Frontend	11
2.1.2. Backend	12
2.2. Herramientas de trabajo	14
3. Diseño e implementación	17
3.1. Introducción	17
3.2. Arquitectura de la plataforma web	18
3.3. Backend	19
3.3.1. Biblioteca C	19
<i>sapi_tick</i>	22
<i>freertos</i>	23
<i>sapi_delay</i>	24
3.3.2. C HAL	25
<i>sapi_tick</i>	26
<i>sapi_delay</i>	27
3.3.3. JavaScript HAL	27
<i>sapi_tick</i>	28
3.4. Frontend	29
3.4.1. JavaScript UI	29
3.4.2. Aplicación de Usuario	30
3.5. Caso de estudio	31
3.6. Diseño de la Interfaz de Usuario	33
Área de ensamblado	36
Área de codificación	37
Área de consola integrada	38
4. Ensayos y resultados	41
4.1. Pruebas de Unidad	41
4.2. Pruebas de Integración	42
4.3. Pruebas de Interfaz	43
4.4. Integración Continua	44

4.5. Pruebas de Validación	44
4.6. Banco de pruebas	44
4.7. Casos de uso	45
4.7.1. Prueba de acceso	45
4.7.2. Pruebas de precisión	47
4.7.3. Prueba de funcionamiento	47
Ensayo en la plataforma EDU-CIAA-NXP	48
Ensayo en la plataforma de emulación para la placa EDU- CIAA-NXP	50
5. Conclusiones	55
5.1. Objetivos alcanzados	55
5.2. Próximos pasos	56
Bibliografía	57

Índice de figuras

1.1. Esquema Emulador EDU-CIAA-NXP.	2
1.2. Esquema de la plataforma ViHard. ¹	3
1.3. Plataforma UnoArduSim.	4
1.4. Plataforma Virtronics.	4
1.5. Plataforma Tinkercad.	5
1.6. Plataforma ARM Mbed OS.	5
2.1. Esquema modelo cliente/servidor.	9
2.2. Esquema de las tecnologías que se usaron en el trabajo.	10
3.1. Diagrama de bloques arquitectura de la plataforma.	18
3.2. Diagrama de clases del módulo <i>GPIO</i> de la biblioteca <i>sAPI</i> del proyecto CIAA.	20
3.3. Diagrama de clases del módulo <i>GPIO</i> de la plataforma de emulación para la placa EDU-CIAA-NXP.	21
3.4. Diagrama de funcionamiento de <i>emscripten</i>	25
3.5. Diagrama de bloques <i>EMSCRIPTEN_KEEPAALIVE</i>	26
3.6. Diagrama de bloques <i>emscripten_sleep</i>	27
3.7. Modelo de <i>publicación/suscripción</i>	28
3.8. Diagrama de bloques de <i>EventEmitter</i> implementado en la plataforma.	28
3.9. Diagrama de bloques de la función <i>ccall</i>	29
3.10. Diagrama de bloques de los oyentes de <i>EventEmitter</i> en la capa UI.	30
3.11. Interacción del usuario con los objetos del emulador.	31
3.12. Activación de evento con el nombre <i>gpio_write</i>	32
3.13. <i>GPIO</i> oyente del evento con el nombre <i>gpio_write</i>	32
3.14. Interacción entre todas las capas de programación.	33
3.15. Parte izquierda de la plataforma de emulación para la placa EDU-CIAA-NXP.	34
3.16. Parte derecha de la plataforma de emulación de la placa EDU-CIAA-NXP.	35
3.17. Agregar periférico.	36
3.18. Periférico agregado en el área de ensamblado.	37
3.19. Código que generó los errores de compilación.	38
3.20. Errores de compilación.	38
3.21. Programa de usuario.	39
3.22. Salida de la terminal serie.	39
4.1. Primera parte de la depuración de las pruebas unitarias con <i>CMocka</i> o <i>Check</i>	42
4.2. Segunda parte de la depuración de las pruebas unitarias con <i>CMocka</i> o <i>Check</i>	42
4.3. Depuración de las pruebas de integración.	43

4.4. Primera parte de la depuración de las pruebas unitarias.	44
4.5. Prueba plataforma emulador ejecutando el ejemplo <i>Blinky</i>	46
4.6. Petición de acceso a la plataforma.	47
4.7. Respuesta del servidor.	47
4.8. Ensayo del ejemplo <i>Dht11 temperature/humidity</i>	49
4.9. Código del ejemplo en eclipse.	49
4.10. Cambios en la placa EDU-CIAA-NXP durante el ensayo.	50
4.11. Salida de la terminal COM8 -Tera Term VT.	50
4.12. La plataforma muestra el resultado del CP02.	51
4.13. Ejemplo <i>Dht11 temperature/humidity</i> ejecutado en la plataforma de emulación.	51
4.14. Datos de temperatura/humedad mostrados por la plataforma.	52
4.15. Petición de datos de temperatura/humedad.	52
4.16. Repuesta de la petición de datos de temperatura/humedad.	53
4.17. Cambios en la placa EDU-CIAA-NXP durante el ensayo.	54

Índice de tablas

1.1. Comparación de características de plataformas de simulación . . .	6
3.1. Módulo <i>GPIO</i>	19
3.2. <i>sapi_peripheral_map.h</i>	22
3.3. Funciones <i>sapi_tick</i>	22
3.4. Comparación de la configuración de <i>sAPI</i> de la CIAA y la Plata- forma de Emulación.	23
3.5. Conceptos importantes de <i>freeRTOS</i> que se cumplen en el emulador.	24
3.6. Funciones <i>sapi_delay</i>	24
4.1. Recursos de hardware utilizados	45
4.2. Recursos de software utilizados	45

Dedicado a mi familia

Capítulo 1

Introducción general

En este capítulo se exponen los problemas encontrados con el hardware en el estudio de los sistemas embebidos y la necesidad de un emulador para la placa EDU-CIAA-NXP [1]. También se describe el estado actual de algunas soluciones implementadas y una breve descripción de la plataforma propuesta.

1.1. Motivación

Teniendo en cuenta la complejidad en el aprendizaje de programación de sistemas embebidos para los desarrolladores principiantes y, además, siendo el principal problema la interacción con el hardware, se propuso, como parte del proyecto CIAA [2], una plataforma que emule la placa EDU-CIAA-NXP.

Ahora bien, considerando que para el desarrollo de aplicaciones en sistemas embebidos es necesario tener de antemano la placa de desarrollo y los componentes de hardware para comenzar con el programa y, además, muchas veces un usuario sin experiencia no cuenta o se equivoca en la selección de los dispositivos para comenzar, una alternativa interesante es la utilización de una plataforma de emulación que permita al usuario probar determinados programas rápidamente.

Es decir, de esta manera, los usuarios principiantes podrían enfocarse en desarrollar sus aplicaciones y ejecutar sus pruebas en un sistema virtual dejando para más adelante la implementación en el hardware real.

El valor de tener un emulador podría apreciarse en estas cuatro aplicaciones prácticas:

- A nivel docente y de enseñanza en escuelas o academias de formación en programación de sistemas embebidos.
- Como herramienta para realizar primeras pruebas.
- Como herramienta para evaluar programas que no funcionen como se esperaba, ya que se aísla el hardware.
- Solución valiosa que permite a los estudiantes con bajos recursos económicos acceder virtualmente a hardware costoso.

1.2. Descripción técnica-conceptual

Los emuladores son desarrollos de software que modelan de forma precisa el funcionamiento del hardware real, de manera que permiten ejecutar programas

dentro de un ambiente que imita su comportamiento [3]. De esta forma el usuario puede simular su código antes de instalarlo en el dispositivo físico.

Por consiguiente, teniendo en cuenta dichas características, el presente trabajo se centró en construir una plataforma de desarrollo con interfaz gráfica que realiza la emulación de la placa EDU-CIAA-NXP dentro de un entorno web. La arquitectura responde a la emulación a nivel de API [4], para lo cual fue necesario integrar y adaptar las sAPI del proyecto CIAA.

En la figura 1.1 se observa una computadora que debe acceder a la plataforma de emulación de la placa EDU-CIAA-NXP mediante un navegador web. Entonces, la herramienta imitará las características de hardware e incluso las de software dentro del entorno de programación en la computadora del usuario, todo de manera virtual.

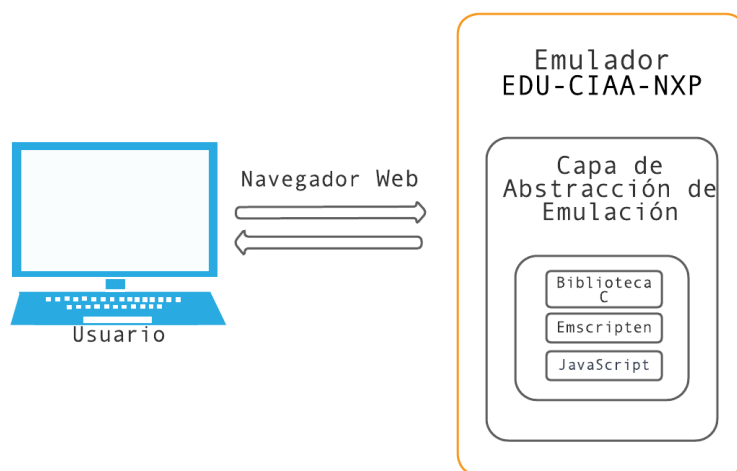


FIGURA 1.1. Esquema Emulador EDU-CIAA-NXP.

En otras palabras, con esta herramienta de emulación, una persona que recién comienza tendrá la experiencia de relacionarse con un sistema embebido solamente conectándose mediante internet a la aplicación web en su ordenador, liberando así el camino a los estudiantes primerizos de hacer la interacción con el hardware.

1.3. Estado del arte

Hoy en día no existe una herramienta de emulación para la placa EDU-CIAA-NXP. Sin embargo, existe la plataforma de código abierto ViHard [5] que emula dispositivos de hardware en la PC, pero conectada a la placa real a través del puerto USB.

La plataforma se compone de: un programa de PC con los periféricos virtuales de hardware y un programa de firmware con una biblioteca embebida, que controla y gestiona el funcionamiento del hardware virtual. Ambos programas se comunican con el sistema embebido mediante un puerto USB.

El programa de hardware virtual es una aplicación de escritorio multiplataforma desarrollada utilizando el framework Electron [6]. Por otro lado, la biblioteca embebida fue desarrollada en lenguaje C.

Por lo tanto, el usuario necesita ejecutar en su PC el programa de periféricos virtuales y contar con la biblioteca en el sistema embebido que controla el hardware virtual. A partir de ahí, procedería al desarrollo de su propio algoritmo y posteriormente realizar las pruebas correspondientes.

La figura 1.2 muestra el esquema de la plataforma ViHard.

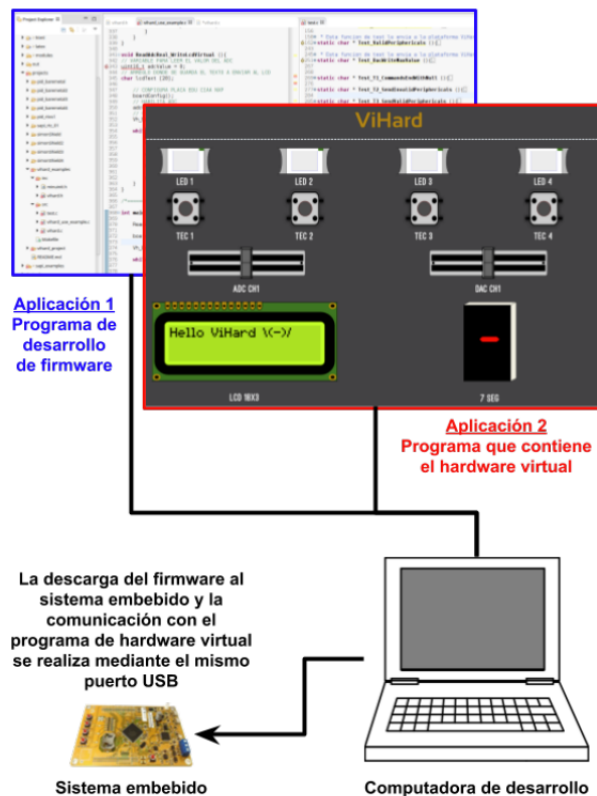


FIGURA 1.2. Esquema de la plataforma ViHard.¹

Además, se encontraron dentro de las plataformas educativas muchos simuladores para microcontroladores, sobre todo para la placa Arduino [7]. Para el análisis, se seleccionaron algunos de los simuladores más populares que además, implementan funcionalidades relevantes para el presente trabajo.

UnoArduSim [8] fue desarrollado en la Universidad de Queen [9] por el profesor Stan Simmons. La herramienta simula en la pantalla de la PC la placa Arduino Uno [10] y muchos de los dispositivos de entrada y salida más usados, asimismo, permite la depuración interactiva de funciones o programas completos. Por otro lado, está diseñada específicamente para ejecutarse en el sistema operativo Windows, además, el diseño de la interfaz gráfica no promueve la claridad visual, puesto que hay demasiados objetos en la pantalla y los que existen deberían estar mejor distribuidos.

En la figura 1.3 puede observarse la interfaz de la aplicación.

¹Imagen tomada de <http://laboratorios.fi.uba.ar/lse/tesis/LSE-FIUBA-Trabajo-Final-MSE-Agustin-Bassi-2018.pdf>

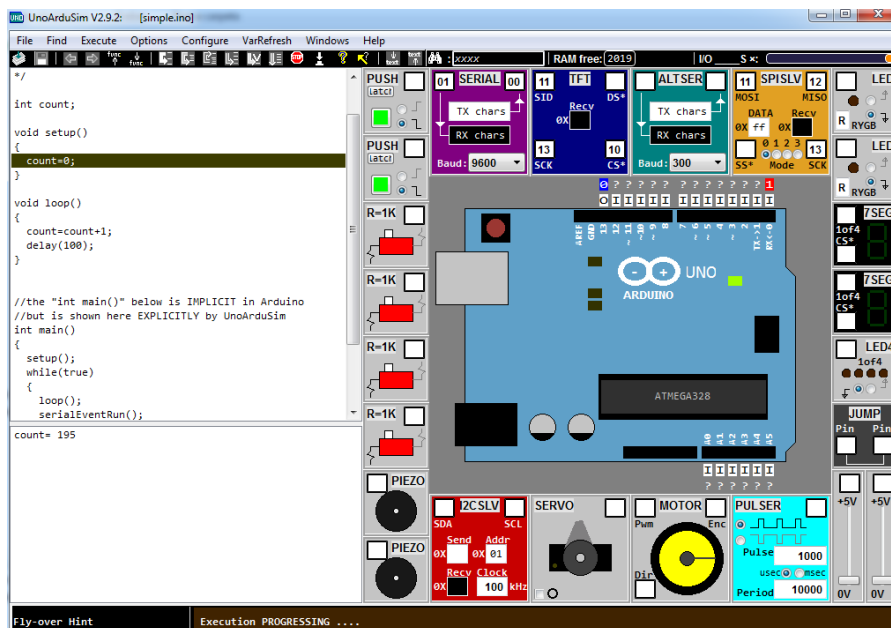


FIGURA 1.3. Plataforma UnoArduSim.

Virtronics [11] es uno de los simuladores más completos que hay hoy en día para Arduino [7], ya que permite simular varios modelos y, además, tiene dos versiones disponibles: una versión paga y otra gratuita, pero con funciones limitadas.

En la figura 1.4 se muestra la plataforma.

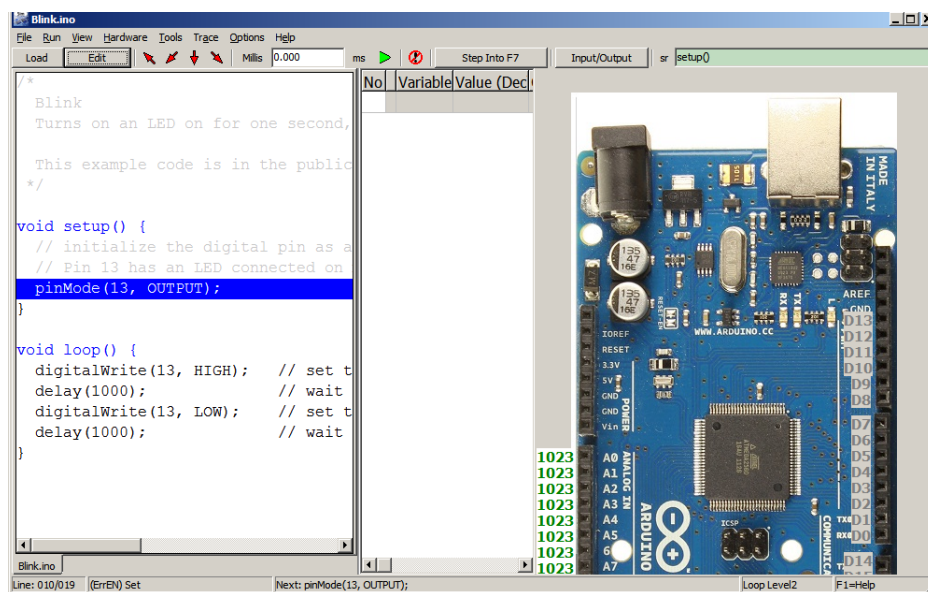


FIGURA 1.4. Plataforma Virtronics.

Tinkercad [12] es una plataforma online que permite el acceso desde cualquier navegador web. Fue desarrollado por ingenieros y diseñadores de software de Autodesk [13] y permite diseños 3D. Además, requiere una cuenta de correo para registrarse antes de empezar a utilizarlo, por consiguiente, todos los diseños se guardan en la cuenta creada.

En la figura 1.5 puede observarse la interfaz de usuario.

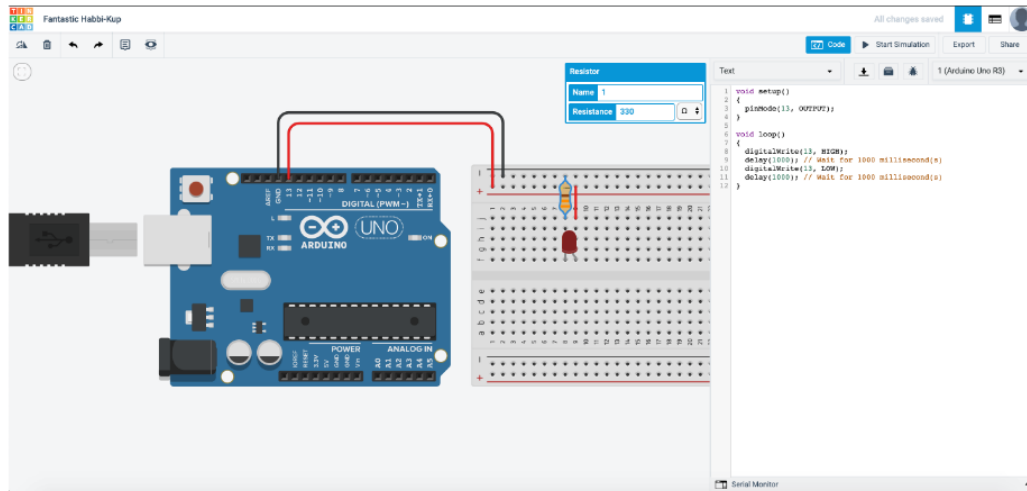


FIGURA 1.5. Plataforma Tinkercad.

ARM Mbed OS [14] fue desarrollado por ingenieros de Mbed [15] y es parte del laboratorio sistema operativo Mbed. La plataforma es online y puede usarse desde cualquier navegador web.

En la figura 1.6 puede observarse la plataforma online de ARM Mbed OS.

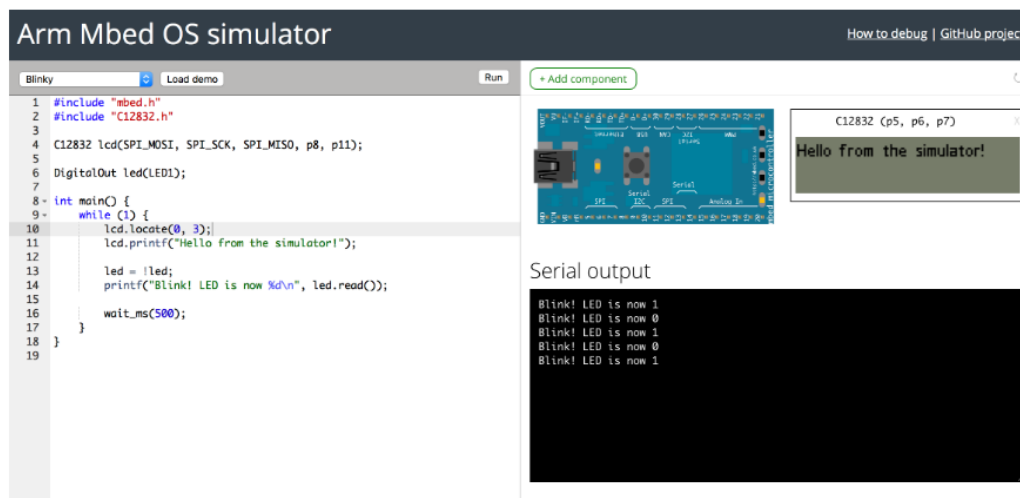


FIGURA 1.6. Plataforma ARM Mbed OS.

En la tabla 1.1 se presenta algunas de las características más importantes que tienen en común estas plataformas y también, algunas de las diferencias encontradas.

TABLA 1.1. Comparación de características de plataformas de simulación

Característica	UnoArduSim	Virtronics	Tinkercad	MbedOS
Microcontrolador	Arduino Uno	Varios modelos Arduino	Arduino Uno	Arm
Gratuito	Sí	No	Sí	Sí
Aplicación	Escritorio	Escritorio	Web	Web
Plataforma	Windows	Windows/Linux	Todas	Todas
Código abierto	No	Sí	No	Sí
Dispositivos E/S	Sí	Sí	Sí	Sí
Panel de desarrollo	Sí	Sí	Sí	Sí
Lenguaje	C	C	C	C/C++
Debugging	Sí	Sí	Sí	No
Biblioteca ejemplos	Sí	Sí	Sí	Sí

Para realizar el presente trabajo final, se decidió basarse en el proyecto de ARM Mbed OS debido a las siguientes ventajas significativas:

- **Comunidad y Soporte:** el proyecto tiene una comunidad activa de desarrolladores, que brinda acceso a una amplia base de conocimientos, documentación y soporte.
- **Reconocimiento de Marca ARM mbed OS:** al basarse en su proyecto simulador, se puede obtener cierto reconocimiento y confianza entre los usuarios.
- **Código abierto:** al tener acceso al código fuente permitió estudiar cómo funciona internamente el proyecto simulador, además, de la libertad de uso y distribución.
- **Reutilización:** ofrece un conjunto sólido de funcionalidades y características ya probadas que agilizó el desarrollo y redujo la probabilidad de introducir errores.
- **Actualizaciones y Mejoras Continuas:** el proyecto Mbed OS recibe actualizaciones continuas con las últimas tecnologías que permite mantener el emulador para la placa EDU-CIAA-NXP actualizado.

1.4. Propósito y Alcance

1.4.1. Propósito

El propósito de este trabajo fue desarrollar una herramienta educativa que brinda un entorno virtual para la placa EDU-CIAA-NXP, permitiendo al usuario seleccionar e interactuar con diferentes tipos de dispositivos virtuales de entrada/-salida. Este desarrollo permite al usuario cargar, ejecutar, editar y corregir sus programas escritos en lenguaje C, pudiendo monitorizar gráficamente en la pantalla del ordenador la placa EDU-CIAA-NXP y muchos de los dispositivos de entrada y salida más usuales, todo de manera virtual, sin necesidad de disponer de ningún dispositivo de hardware.

1.4.2. Alcance

Para la realización de este trabajo se realizó una primera versión de la herramienta para ser usada con la placa EDU-CIAA-NXP.

En el presente trabajo se incluyen los siguientes aspectos:

1. Desarrollo de aplicación para emular el hardware en PC.
2. Desarrollo de programas de ejemplo para ser usados dentro de la aplicación.
3. Documentación de referencia.

El presente proyecto NO incluye los siguientes aspectos:

1. Desarrollo de la aplicación para emular otras placas que no sea EDU-CIAA-NXP.

Capítulo 2

Introducción específica

En el presente capítulo se introducen las tecnologías más importantes involucradas en el desarrollo del emulador on-line y las características de funcionamiento y de diseño.

2.1. Plataforma web

Las aplicaciones web son provistas por un servidor web y pueden ser accedidas por los usuarios que se conecten a través de internet desde cualquier lugar mediante un navegador web [16]. Además, presentan la arquitectura cliente/servidor en donde un cliente o navegador web realiza peticiones al servidor y en consecuencia el servidor envía la respuesta de regreso. En la figura 2.1 se muestra el esquema del modelo cliente/servidor.

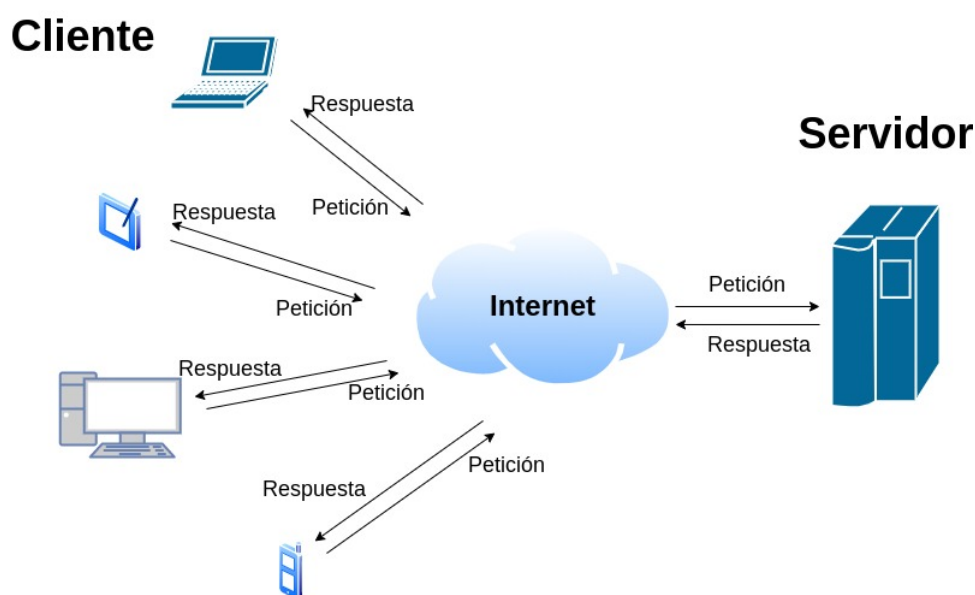


FIGURA 2.1. Esquema modelo cliente/servidor.

Las razones por las que se optó por la tecnología web se debe a las potenciales ventajas que presentan, de las cuales las más importantes son:

- No es necesario instalar nada en la computadora del usuario.
- No consumen los recursos del ordenador.
- No se encuentra limitado a un lugar físico específico para acceder y utilizar las capacidades de emulación.

- No obliga al usuario a usar un determinado sistema operativo, ya que se puede ejecutar en todos los dispositivos con acceso a un navegador web y una conexión a internet.

En el desarrollo web, el *frontend* es la parte del software que interactúa con el usuario y el *backend* es la parte lógica que se encarga de tomar los datos, procesarlos y devolverlos al *front end*.

En la figura 2.2 se muestra un esquema con las tecnologías web usadas en el presente trabajo.

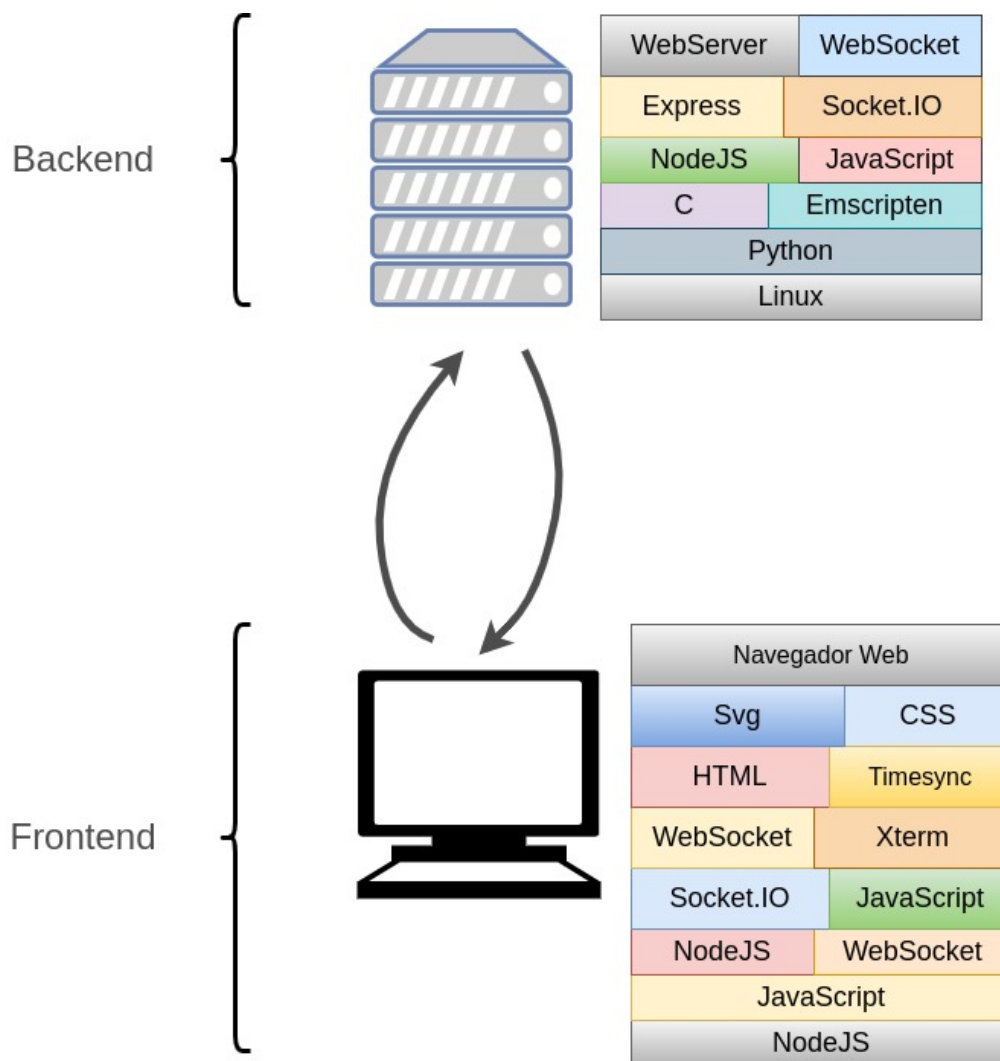


FIGURA 2.2. Esquema de las tecnologías que se usaron en el trabajo.

2.1.1. Frontend

Se describen brevemente las principales tecnologías que se usaron en esta parte del diseño de software.

- JavaScript [17]: es un lenguaje de programación que se ejecuta del lado del cliente en el navegador, o bien del lado del servidor mediante motores como NodeJS [18]. En el caso del lado del cliente, permite crear páginas web dinámicas y también responder a eventos causados por el propio usuario tales como modificaciones del DOM, de la sigla en inglés *Document Object Model* [19]. Por consiguiente, el desarrollo con JavaScript en el frontend permitió cargar y ejecutar los archivos resultantes generados por el compilador en el navegador. Asimismo, es el responsable de configurar el entorno necesario para ejecutar el emulador web y proporcionar la interfaz de usuario para la interacción. Además, en el backend fue útil para gestionar la comunicación y la interacción entre los diferentes componentes, permitiendo la transferencia de datos y el flujo de información dentro de la plataforma web.
- NodeJS [18]: es un entorno de ejecución de JavaScript cuyo propósito es el desarrollo de aplicaciones web y servicios del lado del servidor. También, proporciona una arquitectura orientada a eventos y no bloqueante. De manera que, en el contexto del frontend, se utilizó como parte del flujo de trabajo de desarrollo, incluyendo la gestión de dependencias, automatización de tareas de compilación, pruebas y despliegue. Mientras tanto, en el backend, fue aprovechado para desarrollar las rutas, controladores y manejar la lógica de la aplicación en respuesta a las solicitudes entrantes y el envío de respuestas.
- HTML [20]: de la sigla en inglés *Lenguaje de Marcas de Hipertexto*, del inglés *HyperText Markup Language*, es el lenguaje de marcas que sirve para etiquetar contenido y visualizarlos en el navegador web. Este lenguaje es sencillo de aprender y es fácil de interpretar tanto por humanos como por máquinas. En el desarrollo, se utilizó para crear los elementos visuales de la interfaz de usuario, como los botones, lista desplegable, pantallas de visualización, y otros componentes necesarios para interactuar con el emulador web.
- CSS [21]: de la sigla en inglés *Cascading Style Sheets*, es un lenguaje de diseño gráfico que permite definir estilos, colores, formato, tamaño, tipo de letra del texto, posición de cada elemento dentro de la página, etc. Es la mejor forma de separar los contenidos y es necesario para crear páginas web complejas. El desarrollo con CSS ayudó a controlar la presentación visual y el estilo de la plataforma.
- SVG [22]: de la sigla en inglés *Scalable Vector Graphics*, es un formato de gráficos vectoriales bidimensionales con una base matemática que pueden modificarse según se necesite. Las imágenes creadas con este formato se pueden escalar y hacer zoom de forma arbitraria sin pérdida de resolución debido a que no están formadas por píxeles. Además, esta basado en el lenguaje de marcado extensible XML [23] y es un formato muy útil para ser utilizado en entornos web. Para el desarrollo del diseño de la interfaz fue ideal el uso de este tipo de formato para evitar que las imágenes se deformen y también, ofrecer una experiencia visual interactiva.

- Xterm [24]: es un componente escrito en TypeScript [25] que permite que una aplicación pueda usar terminales emuladas con todas sus funciones en el navegador web. En el desarrollo del emulador web, se utilizó esta tecnología en la interfaz de usuario para visualizar la salida de la terminal.
- WebSocket [26]: esta tecnología permite la comunicación bidireccional entre el cliente y el servidor, trabaja sobre el protocolo TCP/IP y también, es una especificación de protocolo de HTML5, de la sigla en inglés *HyperText Markup Language, versión 5* [27]. Se utilizó WebSocket para establecer una conexión entre el frontend y el backend, y de esta manera, enviar los datos desde el backend al frontend para mostrarlos en la terminal serial. Es decir, permite que los datos de la terminal serial se reflejen de manera dinámica en la interfaz de usuario del frontend.
- Socket.IO [28]: es una biblioteca de Javascript que usa websocket para la comunicación bidireccional y para la baja latencia. También, está basada en el manejo de eventos entre un cliente y un servidor. El desarrollo con esta tecnología permitió facilitar la comunicación en tiempo real entre el cliente y el servidor para la terminal serial.
- Timesync [29]: es una biblioteca de JavaScript que se usa para sincronizar temporizadores en una aplicación y, además, las aplicaciones cliente sincronizan el tiempo con el servidor, ya sea a través de peticiones HTTP o WebSockets. El desarrollo de la plataforma web con esta tecnología aseguró que los eventos y acciones ocurran en el momento exacto tanto en el cliente como en el servidor.
- Mocha [30]: es un marco de trabajo para pruebas de JavaScript que tiene funciones que se ejecutan en NodeJS y en el navegador web. En consecuencia, hace que las pruebas asincrónicas sean simples. Asimismo, Las pruebas se ejecutan en serie y se realiza el envío de excepciones aún no detectadas a los casos de prueba correctos. El uso de este marco de trabajo permitió hacer pruebas sobre la interfaz de usuario de la plataforma.
- Chai [31]: es una biblioteca de aserciones que puede usarse con cualquier marco de pruebas de Javascript. Asimismo, tiene varias interfaces: *assert*, *expect* y *should*, que permiten al desarrollador elegir cuál usar. Chai se utiliza en las pruebas del emulador web para verificar el comportamiento esperado de las funciones, componentes y datos generados por el emulador.

2.1.2. Backend

Se describen brevemente las principales tecnologías que se usaron en el backend.

- Emscripten [32]: es un compilador que traduce la mayor parte del lenguaje LLVM, de la sigla en inglés *Low Level Virtual Machine* [33], a JavaScript. De esta forma, permite ejecutar el código de varios lenguajes de programación en los navegadores actuales. Emscripten compila C y C++ en WebAssembly [34] mediante LLVM y Binaryen [35]. La salida de Emscripten puede ejecutarse en la web y en NodeJS. El uso de esta herramienta tuvo por objetivo el compilar el código escrito en C, a WebAssembly (Wasm) y JavaScript. Esto permitió ejecutar aplicaciones nativas en la web sin necesidad de plugins o complementos adicionales.

- Python [36]: es un poderoso y popular lenguaje de programación multiplataforma de código abierto. Se caracteriza por su sencillez y su gran potencia para el tratamiento de datos en el lado del servidor. En el emulador web, Python se utilizó para escribir scripts que realicen tareas específicas, como la configuración, inicialización y depuración. De esta manera, facilitó el desarrollo.
- Express [37]: es un marco de aplicaciones web en el backend para NodeJS. También, está diseñado para crear aplicaciones web y APIs, de la sigla en inglés *application programming interface* [4]. El uso de este framework simplifico el manejo de solicitudes HTTP, la definición de rutas para interactuar con la plataforma web. De manera que, facilitó que el desarrollo del emulador web sea más simple y estructurado.
- Lenguaje C [38]: es de propósito general y es muy popular debido al eficiente código que produce al crear software de sistemas y de aplicaciones. Asimismo, es un lenguaje de tipos de datos estáticos, fuertemente tipado y tiene estructuras típicas de los lenguajes de alto nivel pero, a su vez, tiene construcciones que permiten un control de los lenguajes de bajo nivel. El desarrollo con C fue fundamental, ya que permitió la integración de las sAPI de la CIAA en el desarrollo del emulador web.
- Check [39]: es una biblioteca de pruebas unitarias para el lenguaje de programación C que proporciona un conjunto de macros y funciones que facilitan la escritura y la ejecución de las pruebas unitarias. Además, provee mecanismos que aíslan y ejecutan las pruebas en un entorno controlado y separado, usando suites de pruebas, funciones de inicialización y limpieza. Se utilizó en el emulador web para verificar el correcto funcionamiento de las funciones y componentes implementados en el backend del emulador escritos en lenguaje C.
- CMocka [40]: es una biblioteca de pruebas unitarias especialmente diseñada para C, destacandose por su capacidad de crear mocks (falsos) y stubs (simulaciones) de funciones. De esta manera se logra probar componentes de código que dependen de funciones externas. Y, además, facilita el aislamiento de las unidades de código y la creación de escenarios de prueba que pueden ser controlados. El uso de esta tecnología permitió simular funciones mediante mocks para controlar el comportamiento de las funciones dependientes y facilitar las pruebas de código que interactúa con dichas funciones.
- sAPI CIAA [41]: esta biblioteca escrita en lenguaje C y compatible con C++ implementa una API simple que funciona como una capa de abstracción de hardware para microcontroladores. Es la principal biblioteca del Proyecto CIAA para el desarrollo de aplicaciones en C/C++ en los frameworks Firmware v2[42] y Firmware v3[43]. Para la emulación a nivel de API, se utilizó como base la API de la sAPI v0.6.2 disponible en firmware v3 del Proyecto CIAA [44] y se realizaron las implementaciones necesarias para que funcione en la web, en lugar de funcionar en el hardware de un microcontrolador real. De esta manera, se proporcionó una interfaz idéntica, permitiendo a los usuarios del emulador programar en la plataforma web de la misma manera que lo harían con la placa EDU-CIAA real, logrando

que cualquier programa escrito utilizando la sAPI pueda correr en el emulador. Cabe destacar que al emular a nivel de sAPI, el usuario no podrá utilizar funciones de bajo nivel de la EDU-CIAA, como ser la biblioteca del fabricante del microcontrolador (LPCopen[45]), o el acceso directo a registros físicos del microcontrolador.

- Mbed CLI [46]: es una herramienta de línea de comandos que facilita el desarrollo y la gestión de proyectos basados en la plataforma Mbed [15]. Incluso, permite realizar tareas como la configuración del entorno de desarrollo, compilación de código, gestión de dependencias y la depuración, permitiendo identificar y resolver problemas en el código. Se reutilizó esta herramienta, ya incluida en el emulador sobre el cual se basó este trabajo, para proporcionar una interfaz de línea de comandos que simplificó las tareas de configuración y construcción del emulador web.
- SVG EDU-CIAA-NXP [47]: se reutilizaron para el presente trabajo los dibujos de la placa EDU-CIAA-NXP. La reutilización de estos dibujos fueron necesarios para proporcionar al usuario una experiencia visual interactiva.
- Mbed events [15]: es una biblioteca de código que se utiliza en el desarrollo de software embebido para facilitar la gestión de eventos y temporizadores. Para el desarrollo de la plataforma del emulador web se reutilizó esta biblioteca para la creación y gestión de tareas en freeRTOS.

2.2. Herramientas de trabajo

Se exponen las principales plataformas que se usaron para el desarrollo de este trabajo.

- EDU-CIAA-NXP: esta es la plataforma de hardware objetivo a emular del presente trabajo. Es uno de los diseños de hardware del Proyecto CIAA. En particular, el enfoque es ayudar a las Universidades Argentinas a migrar de microcontroladores de 8 bits a modernos microcontroladores de 32 bits al usar una placa diseñada en Argentina con hardware y software abiertos. Estas placas se difundieron en todas las universidades Argentinas con carreras de electrónica y afines, y también, en algunos países limítrofes. Se utilizó la placa física para ensayos de comparación entre lo real y el emulador web desarrollado.
- Visual Studio Code [48]: es un editor de código fuente gratuito y de código abierto desarrollado por Microsoft. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos, refactorización de código y muchas otras herramientas más. Se eligió este IDE, de la sigla en inglés Integrated Development Environment [49], por la capacidad de sus herramientas y la simpleza de su editor de código. El uso de este editor facilitó la escritura y el mantenimiento del código del emulador, mejorando la productividad y la calidad del desarrollo.
- Inkscape [50]: es un editor de gráficos vectoriales que permite crear, editar y modificar gráficos. En el proceso de desarrollo de la interfaz del emulador web se utilizó para diseñar algunos elementos gráficos dentro del dibujo de la placa EDU-CIAA-NXP.

- GitHub [51]: es una plataforma de desarrollo colaborativo que permite alojar proyectos utilizando el sistema de control de versiones Git. Se utilizó los servicios de esta plataforma para almacenar y compartir el código fuente, de manera que se pueda hacer un seguimiento de las últimas modificaciones realizadas.
- Travis CI [52]: es un servicio de integración continua en la nube y es utilizado mayormente para configurar y ejecutar pruebas automatizadas en un entorno controlado y reproducible. Además, se integra con sistemas de control de versiones como GitHub y permite que con cada cambio realizado en el repositorio se ejecuten las pruebas definidas en el script de construcción. De esta manera, se asegura la calidad del software. Incluso, proporciona informes detallados de las pruebas realizadas y servicio de notificaciones por correo electrónico.
- GitLab [53]: es una plataforma web de gestión de repositorios y permite la colaboración en el desarrollo de software. Proporciona un conjunto completo de herramientas para el ciclo de vida del desarrollo de software, por lo tanto, permite configurar pipelines de integración y entrega continua, en consecuencia, automatiza la compilación, las pruebas y el despliegue de software de manera eficiente. Es una alternativa a otras plataformas como GitHub con Travis CI.
- DigitalOcean [54]: ofrece servicios de infraestructura de computación en la nube, tales como permitir a los usuarios crear y administrar servidores virtuales, conocidos como Droplets. Incluso, ofrece opciones de almacenamiento, configuración de redes privadas virtuales, servicios de bases de datos, entre otros. DigitalOcean se destaca por su enfoque en la simplicidad y la facilidad de uso de su plataforma. El emulador desarrollado en el presente trabajo se encuentra desplegado en el servidor de DigitalOcean.

Capítulo 3

Diseño e implementación

En este capítulo se exponen los criterios de selección para la arquitectura de la plataforma de emulación. Se presenta la arquitectura elegida y se describe la estructura y organización de las capas de programación.

3.1. Introducción

El principal criterio de selección de la arquitectura del emulador se fundamentó en el análisis de la arquitectura del sistema operativo *mbed* y cómo funciona dentro de su simulador. La fase inicial de investigación permitió adquirir una comprensión del comportamiento de los periféricos simulados, las interacciones y las configuraciones necesarias para la plataforma web.

Sin embargo, durante el análisis del repositorio, la presencia de múltiples módulos de código, bibliotecas y configuraciones que no estaban directamente relacionados con la simulación, inicialmente generó incertidumbre sobre su propósito y relevancia en la plataforma del simulador.

Después, al eliminar módulos y dependencias innecesarias, se procedió a evaluar la compatibilidad de su arquitectura. Durante el proceso, en primer lugar, se analizó si la arquitectura proporcionaba funcionalidades para la interacción con los usuarios. Incluso, de ser escalable y capaz de manejar nuevos desarrollos.

Además, la mantenibilidad y flexibilidad también fueron aspectos evaluados. Otro punto esencial, fueron las tecnologías y herramientas utilizadas, lo cual implicó una curva de aprendizaje adicional.

Asimismo, *emscripten* fue fundamental para el desarrollo de la plataforma, y debido a la experiencia previa con el uso de cross-compiling toolchains para el kernel de Linux, facilitó la transición y ayudó en la comprensión de los conceptos y las herramientas relacionadas con *emscripten*.

Finalmente, una vez confirmada su compatibilidad para los propósitos del presente trabajo, se tomó la decisión de seleccionarlo y empezar el desarrollo.

3.2. Arquitectura de la plataforma web

En el diagrama de bloques de la figura 3.1 se muestra la arquitectura básica de una aplicación de usuario que ejecuta el emulador de la placa EDU-CIAA-NXP.

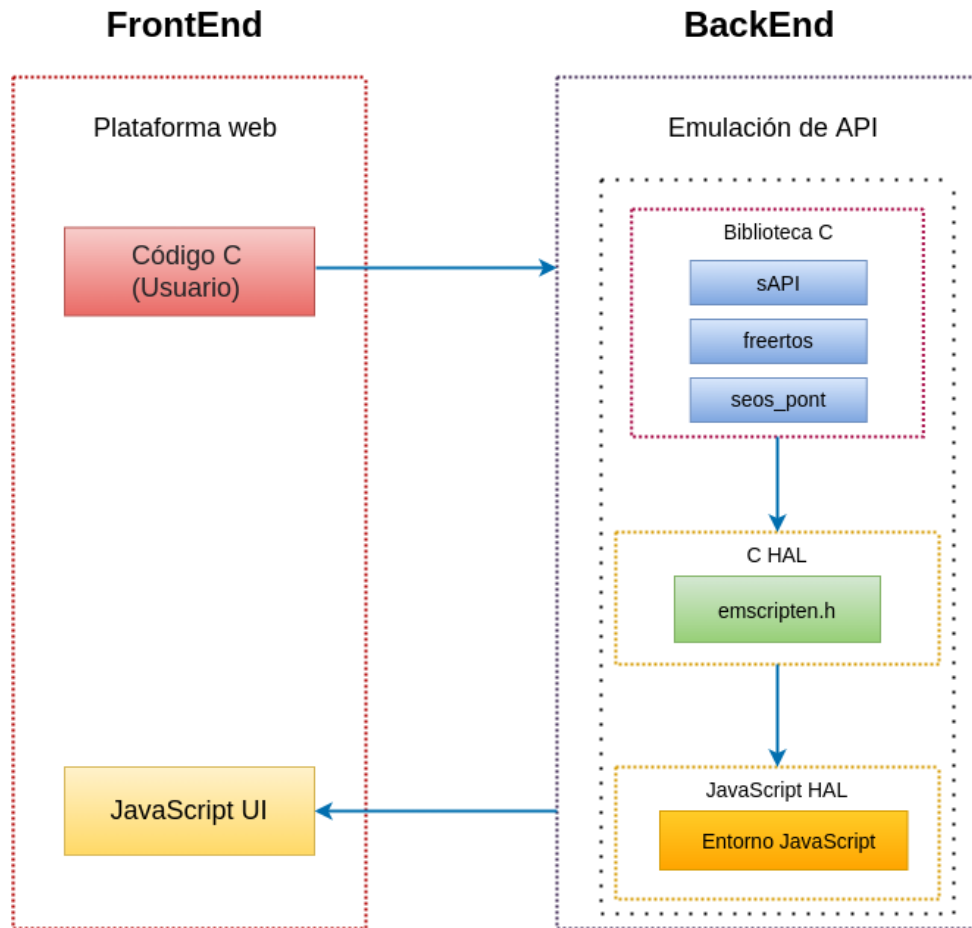


FIGURA 3.1. Diagrama de bloques arquitectura de la plataforma.

La emulación a nivel de API proporciona una capa de abstracción para el entorno de desarrollo del usuario, de modo que permite escribir aplicaciones en C y ejecutarlas en la plataforma de emulación.

Al utilizar esta capa de abstracción, se pudo replicar las funcionalidades y las características de configuración de la biblioteca *sAPI* del proyecto CIAA en un entorno diferente, en este caso, en un entorno web.

3.3. Backend

En esta capa de programación se desarrolló toda la lógica necesaria para emular las funcionalidades que proporcionan las bibliotecas: *sAPI*, *freertos* y *seos_pont* para la placa EDU-CIAA-NXP.

3.3.1. Biblioteca C

En primer lugar, se identificaron las funciones de las bibliotecas C originales para empezar a emular. Luego, en el emulador se crearon interfaces que reflejen las estructuras de las bibliotecas originales, que incluyen definiciones de funciones, estructuras de datos y constantes.

Es decir, en las funciones originales se examinaron los parámetros de entrada y los valores de retorno, para luego mapearlos correctamente en las definiciones de las funciones del emulador. A modo de referencia se muestra en la tabla 3.1 las definiciones de las funciones para `sapi_gpio.h`, que incluye los nombres de la funciones, los tipos de parámetros y el tipo de valor de retorno. Cabe destacar que estas definiciones son idénticas tanto en las *sAPI* como en el emulador, lo que permite una fácil correspondencia entre ambas.

TABLA 3.1. Módulo GPIO

Nombre de la función	Parámetros	Tipo de retorno
<code>gpioInit</code>	<code>gpioMap_t</code> , <code>gpioInit</code>	<code>bool_t</code>
<code>gpioRead</code>	<code>gpioMap_t</code>	<code>bool_t</code>
<code>gpioWrite</code>	<code>gpioMap_t</code> , <code>bool_t</code>	<code>bool_t</code>
<code>gpioToggle</code>	<code>gpioMap_t</code>	<code>bool_t</code>

Al igual que en la biblioteca *sAPI* del proyecto CIAA, los archivos de código fuente para la plataforma de emulación, conservan el mismo nombre, por ejemplo `sapi_gpio.c`. Sin embargo, la implementación de las funciones es totalmente distinta. En el caso de la biblioteca *sAPI* del proyecto CIAA, para `sapi_gpio.c` se incluyen los archivos de encabezado: `gpio_18xx_43xx.h` y `scu_18xx_43xx.h`. Y en el caso de la plataforma de emulación se usa otro archivo de encabezado como `gpio_api.h` para replicar el mismo comportamiento.

A continuación, se presenta una comparación entre las clases del módulo `gpio` de la biblioteca *sAPI* del proyecto CIAA y el módulo `gpio` implementado en la plataforma de emulación.

La figura 3.2 muestra las clases que se usan para la implementación de `sapi_gpio.c` del proyecto CIAA y en la figura 3.3 se muestran otras clases para el mismo módulo.

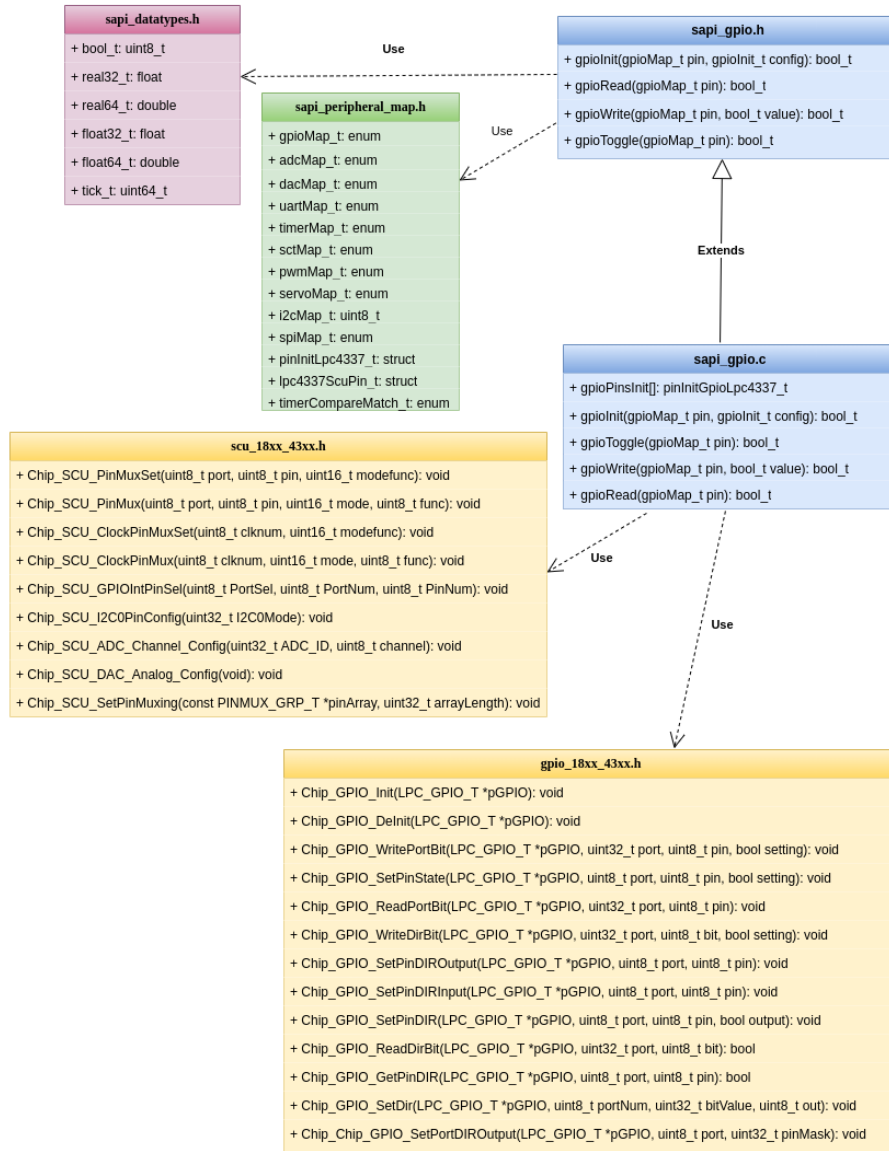


FIGURA 3.2. Diagrama de clases del módulo *GPIO* de la biblioteca *sAPI* del proyecto CIAA.

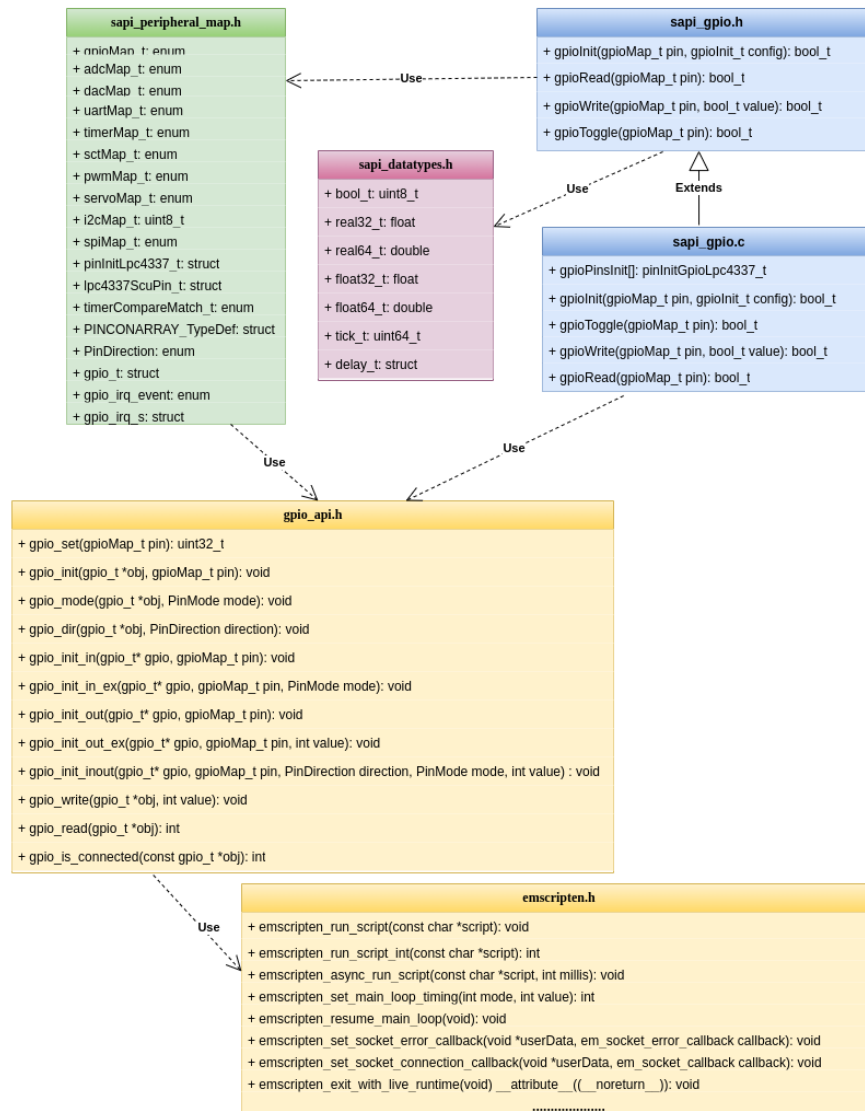


FIGURA 3.3. Diagrama de clases del módulo *GPIO* de la plataforma de emulación para la placa EDU-CIAA-NXP.

En ese sentido, para los demás archivos de la biblioteca C de la plataforma de emulación, se siguió con la misma lógica. Asimismo, se utilizó un esquema de nomenclatura de los archivos de encabezado y de código fuente similar al de las bibliotecas originales. Esto permitió mantener una estructura organizada y coherente en la emulación, que facilita el mantenimiento y comprensión.

También, se reservó el archivo de encabezado `sapi.h` que cumple con la misma funcionalidad que en la biblioteca *SAPI* del proyecto CIAA, de tal forma que permita mediante su inclusión en la aplicación de usuario, usar todos los archivos que se necesitan.

Además, se reutilizaron los archivos de encabezado: `sapi_datatypes.h` y `sapi_peripheral_map.h` incluidos en todos los módulos de la biblioteca *SAPI* del proyecto CIAA. La decisión de reutilización fue para emular las características de hardware y prevalecer el uso de todos los tipos de datos básicos y configuraciones de la placa.

En la tabla 3.2 se muestran los tipos de datos de `sapi_peripheral_map.h` que se usan en la plataforma de emulación. De manera que, se puede observar que se reservan los nombres: TEC1, TEC2, TEC3 y TEC4 para los botones y los nombres LEDR, LEDG, LEDB, LED1, LED2 y LED3 para los LEDs de la placa EDU-CIAA-NXP.

TABLA 3.2. `sapi_peripheral_map.h`.

P2 header	P1 header	LEDs	Switches
GPIO8, GPIO7, GPIO5	T_FIL1	LEDR	TEC1
GPIO3, GPIO1, LCD1	T_COL2	LEDG	TEC2
LCD2, LCD3, LCDRS	T_COL0	LEDB	TEC3
LCD4, SPI_MISO, ENET_TXD1	T_FIL2	LED1	TEC4
ENET_TXD0, ENET_MDIO, ENET_CRSDV	T_FIL3	LED2	
ENET_MDC, ENET_TXEN, ENET_RXD1	T_FIL0	LED3	
GPIO6, GPIO4, GPIO2	T_COL1		
GPIO0, LCDEN, SPI_MOSI, ENET_RXD0	CAN_TD		

sapi_tick

En la tabla 3.3 se puede observar las funciones del archivo de código fuente `sapi_tick` en la biblioteca *sAPI* del proyecto CIAA y en el emulador.

TABLA 3.3. Funciones `sapi_tick`

Función	Parámetros	Tipo de retorno
<code>tickInit</code>	<code>tick_t tickRateMSvalue</code>	<code>bool_t</code>
<code>tickRead</code>	<code>void</code>	<code>tick_t</code>
<code>tickWrite</code>	<code>tick_t ticks</code>	<code>void</code>
<code>tickCallbackSet</code>	<code>callBackFuncPtr_t tickCallback, void*</code> <code>tickCallbackParams</code>	<code>bool_t</code>
<code>tickPowerSet</code>	<code>bool_t power</code>	<code>void</code>

Para emular a nivel de API, se tuvo como objetivo replicar el comportamiento de la función `tickInit`, la cual se encarga de la inicialización y configuración de la interrupción del temporizador en la placa física. Sin embargo, al realizar la emulación en la plataforma web, esta función no se encuentra disponible de forma nativa. Por lo tanto, fue necesario emular su comportamiento y proporcionar una alternativa compatible.

Para lograr emular la funcionalidad de `SysTick_Config`, se utilizó la capa de emulación correspondiente a *C HAL*, que se ira detallando en las siguientes secciones. Esta capa de emulación permitió ejecutar código *javaScript* en el contexto de *emscripten*, lo que posibilitó replicar el comportamiento del temporizador *SysTick*.

Por otra parte, en la implementación de la *sAPI* de la CIAA, el período se calcula multiplicando la frecuencia del reloj del sistema `SystemCoreClock` por `tickRateMSvalue` y dividiendo el resultado por 1000. Sin embargo, en el emulador, el período se corresponde directamente con el valor de `tickRateMSvalue`, lo que puede llevar a ciertas imprecisiones de tiempo.

En la siguiente tabla 3.4 se muestran las diferencias en el enfoque de la implementación de la *sAPI* de la CIAA y el emulador.

TABLA 3.4. Comparación de la configuración de *sAPI* de la CIAA y la Plataforma de Emulación.

Plataforma	Temporizador	Frecuencia
<i>sAPI</i> CIAA	SysTick	SystemCoreClock * tickRateMSvalue / 1000
Emulador	Emscripten	tickRateMSvalue en milisegundos

Una vez habilitada la interrupción del temporizador, se realiza una invocación periódica a la función `tickerCallback`, que tiene la misma implementación que en `sapi_tick` de la biblioteca *sAPI* del proyecto CIAA. La función `tickerCallback` realiza las siguientes acciones: incrementa los contadores de ticks y, si el puntero `tickHookFunction` no es nulo, lo ejecuta pasando los parámetros `callBackFuncParams`. En consecuencia, esto permite la ejecución de tareas específicas programadas por el usuario en cada interrupción del temporizador periódico.

En el siguiente capítulo se detallarán las diferencias encontradas al realizar las pruebas entre la placa y el emulador utilizando estas funciones.

En el contexto de la emulación a nivel de API, para implementar las demás bibliotecas de las *sAPI*, se siguió el mismo esquema utilizado en `sapi_tick`. primeramente, se identificaron las funciones que requerían interacción con el hardware de la placa. Luego, en la capa *C HAL* se implementaron funciones de emulación con `emscripten` para reflejar el comportamiento del hardware.

freertos

Para emular la funcionalidad de las tareas de *freeRTOS* en el contexto del emulador web, se implementó una solución utilizando la biblioteca de eventos de *mbed*. Entonces, para las funciones `xTaskCreate` y `xTaskCreateStatic`, se programaron funciones periódicas utilizando las siguientes funciones de la biblioteca de eventos:

- `int equeue_create`: crea una cola de eventos, configura e inicializa los recursos de plataforma necesarios, como semáforos y mutexes.
- `int equeue_call_every`: se utiliza para crear un evento periódico en la cola de eventos `equeue`, programando llamadas repetidas a una función en intervalos regulares.
- `int equeue_post`: permite publicar un evento en la cola de eventos `equeue`, estableciendo el tiempo y estableciendo el evento en la cola para su posterior procesamiento.
- `void equeue_dispatch`: se encarga de despachar los eventos en la cola de eventos `equeue` de manera continua, verificando los tiempo y realizando acciones específicas según la configuración.
- `void equeue_destroy`: permite liberar y limpiar todos los recursos asociados a una cola de eventos, libera los mutexes, semáforos y memoria asignada.

Estas funciones permiten ejecutar tareas periódicas en intervalos de tiempo regulares, lo que proporcionó una aproximación simplificada para emular la funcionalidad de tareas en el emulador web. Aunque esta solución no ofrece todas las características de un sistema operativo de tiempo real completo como *freeRTOS*, fue adecuada para simular el funcionamiento de programas de usuario simples.

Es importante destacar que la implementación de tareas en el emulador web tiene una limitación significativa. Debido a que solo puede ejecutar un subproceso (hilo de ejecución) a la vez, no es posible que se ejecuten tareas simultáneas. Esto significa que, a diferencia del sistema operativo de tiempo real *freeRTOS*, donde se pueden crear múltiples tareas que se ejecutan de manera concurrente, en el emulador web solo es posible ejecutar una sola tarea. Por lo tanto, esta solución es adecuada para programas de usuario simples que no requieran multitarea.

La tabla 3.5 expone algunos de los conceptos importantes de *freeRTOS* que se cumplen en el emulador.

TABLA 3.5. Conceptos importantes de *freeRTOS* que se cumplen en el emulador.

Capacidades	<i>freeRTOS</i>	Emulador
Multitareas	Si	No
Funciones de espera	Si	Si
Cambio de contexto	Si	Si
Tarea de procesamiento continuo	Si	Si
Manejo de prioridades	Si	No

En el emulador, se encuentran implementados varios conceptos importantes de *freeRTOS*, como funciones de espera y cambio de contexto. Sin embargo, en esta primera versión del emulador, no se han incluido el manejo de multitareas y de prioridades presentes en *freeRTOS*.

sapi_delay

La tabla 3.6 expone las funciones del archivo de código fuente *sapi_delay* en la biblioteca *sAPI* del proyecto CIAA y en el emulador.

TABLA 3.6. Funciones *sapi_delay*

Función	Parámetros	Tipo de retorno
<code>delayInaccurateMstick_t delay_ms</code>		void
<code>delayInaccurateUstick_t delay_us</code>		void
<code>delayInaccurateNstick_t delay_ns</code>		void
<code>delay</code>	<code>tick_t duration_ms</code>	void
<code>delayInit</code>	<code>delay_t * delay, tick_t duration</code>	void
<code>delayRead</code>	<code>delay_t * delay</code>	bool_t
<code>delayWrite</code>	<code>delay_t * delay, tick_t duration</code>	void

La función `delay` en la biblioteca *sAPI* del proyecto CIAA crea una pausa en la ejecución del programa durante el tiempo especificado en `duration_ms` implementando un bucle de espera. Este bucle se ejecutará mientras la diferencia de

tiempo entre `tickRead()` y `startTime`(inicio actual de `tickRead()`) sea menor que `duration_ms / tickRateMS`.

Sin embargo, cuando *emscripten* compila código *C* a *JavaScript*, la función `delay` tal como está escrita, causa que la ejecución de la plataforma web se bloquee o congele. Esto se debe a que la función `tickRead()` no se actualiza a la velocidad que se espera, lo que lleva a que se obtenga el mismo valor repetidamente. Es decir, debido a la naturaleza asíncrona de *JavaScript* y al no tener una pausa controlada en el bucle (como `delay(1)`), *JavaScript* no tiene tiempo suficiente para actualizar el valor de `tickRead()` entre iteraciones. De esta manera, el bucle `while` se queda esperando activamente e impide al navegador atender otros eventos.

Por esta razón, se decidió usar en las funciones de espera, las funciones nativas de *emscripten* usando la capa de emulación *C HAL*, que se ira detallando en la siguiente sección. En consecuencia, se aprovecho su eficiencia y precisión.

3.3.2. C HAL

La capa de Abstracción de Hardware en *C* (*C HAL*) permitió replicar el comportamiento del hardware de la placa, lo que a su vez posibilitó la compatibilidad de las bibliotecas de nivel superior escritas en *C* en el entorno de emulación de la plataforma web.

En esta capa se encuentra la biblioteca *emscripten.h*, la cual provee las funciones y macros necesarias para interactuar con el compilador de *emscripten*. El compilador transforma el código *C* a *JavaScript*, de esta manera, se facilita la interacción y comunicación entre el código *C* y el entorno web, donde se ejecuta el código ya convertido a *JavaScript*.

Emscripten, usa el compilador *LLVM* que actualmente tiene la mayor compatibilidad con *webAssembly*, y lo bueno de *LLVM* es que el rendimiento del código generado es muy eficiente.

Además, *LLVM* proporciona múltiples componentes que pueden intercambiarse entre sí, a diferencia de los compiladores *GCC* que presentan una estructura monolítica. Por tanto, *emscripten* compila el código *C* en un módulo *webAssembly* para que luego permita ser ejecutado en la web.

En la figura 3.4 se muestra el diagrama con el principal funcionamiento de *emscripten*.

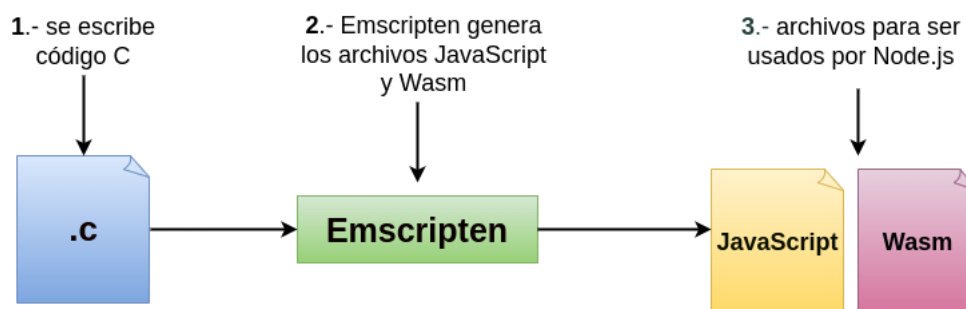


FIGURA 3.4. Diagrama de funcionamiento de *emscripten*.

Se puede observar que son dos los archivos generados:

- Archivo *JavaScript*.
- Archivo *Wasm*.

El segundo es el archivo *webAssembly* que contiene el código compilado, y el primero es un archivo *javascript* que contiene el soporte en tiempo de ejecución por *NodeJS* para cargarlo y ejecutarlo.

En ese sentido, para la aplicación de usuario dentro de la plataforma, cuando se ejecuta un programa, por ejemplo *blinky*, *emscripten* creará los archivos *blinky.js*, *blinky.wasm*, y adicionalmente creará otros archivos de configuración que son complementarios, los cuales son:

- Archivo *blinky.js.components*.
- Archivo *blinky.wasm.map*.
- Archivo *blinky.wast*.

sapi_tick

Para emular el comportamiento de la interrupción del temporizador *SysTick* y proporcionar la invocación periódica a la función `tickerCallback` de *sapi_tick*, se utilizó la macro `EMSCRIPTEN_KEEPALIVE` de *emscripten*, que le dice al compilador de *emscripten* que conserve la función marcada con esta macro en el código compilado, incluso si no es accedida desde el código *JavaScript* del lado del cliente.

Es decir, cuando la función marcada con la macro `EMSCRIPTEN_KEEPALIVE` sea invocada desde la capa *JavaScript HAL*, llamará a la función `tickerCallback` de la biblioteca C y la ejecutará. En la figura 3.5 se muestra el funcionamiento de `EMSCRIPTEN_KEEPALIVE`.

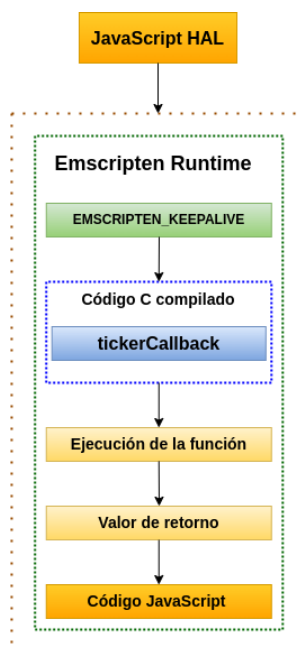


FIGURA 3.5. Diagrama de bloques `EMSCRIPTEN_KEEPALIVE`.

sapi_delay

Para emular las funciones de espera de la biblioteca C se implemento la función `emscripten_sleep`, que utiliza funciones asincrónicas internas de `emscripten` para realizar pausas.

Por lo tanto, permite al navegador atender otros eventos mientras el programa se encuentra en espera. Es decir, evita el bloqueo de la ejecución del resto del código y también, que la página no responda.

Además, proporciona pausas precisas, debido a que, *emscripten* utiliza las capacidades de temporización del navegador para garantizar que el tiempo indicado sea realizado.

La figura 3.6 representa el funcionamiento de `emscripten_sleep`.

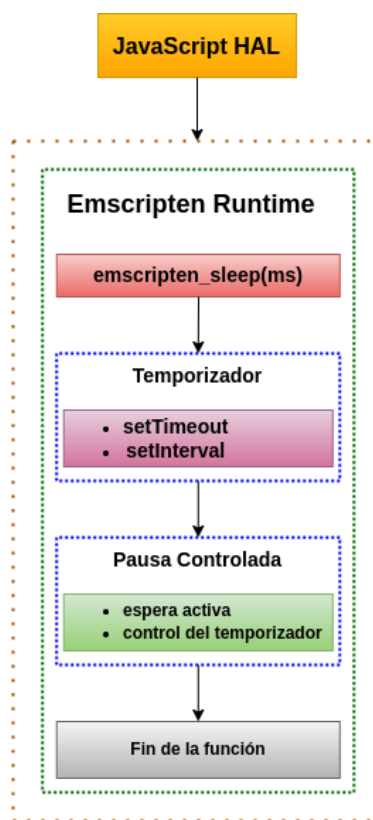


FIGURA 3.6. Diagrama de bloques `emscripten_sleep`.

3.3.3. JavaScript HAL

Esta capa de programación se diseñó para proporcionar la funcionalidad de distribuir los eventos entre los componentes de la interfaz de usuario de *JavaScript* y la capa C *HAL*. Para lograr este objetivo se usaron las clases *EventEmitter* del módulo *Events* que monitorizan y activan los eventos. Además, facilita la interacción del navegador con el código *JavaScript* y la actualización de la interfaz de usuario de manera flexible y eficiente.

También, la clase *EventEmitter* se basa en el modelo de publicación/suscripción que se trata de un paradigma de envío de mensajes asíncrono mediante el cual un usuario publica mensajes y uno o varios objetos se suscriben a esos eventos.

En la figura 3.7 se muestra el modelo de *publicación/suscripción*.

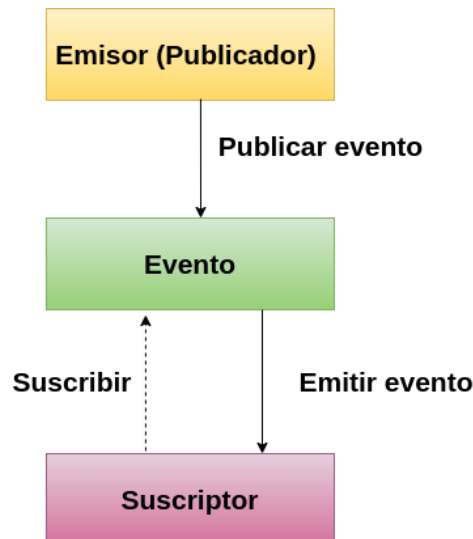


FIGURA 3.7. Modelo de *publicación/suscripción*.

Para la implementación de esta capa de emulación, se crearon archivos *javascript* con instancias de la clase *EventEmitter*, que al utilizar el método *emit* lanzan eventos con nombre. El nombre del evento es un string y permite que los oyentes registrados al evento sean notificados. La figura 3.8 muestra el diagrama de bloques de la instancia de *EventEmitter* en la plataforma de emulación.

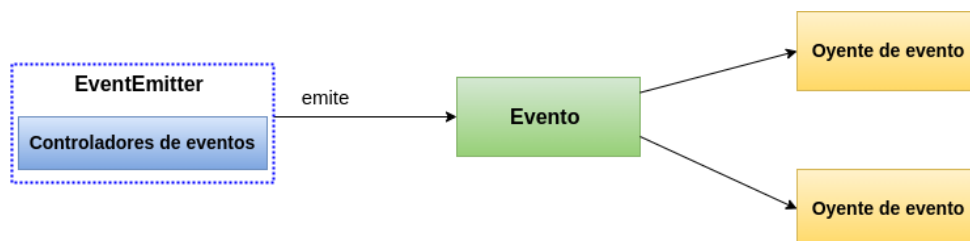


FIGURA 3.8. Diagrama de bloques de *EventEmitter* implementado en la plataforma.

En la sección de *Caso de Estudio*, se mostrará la implementación de este modelo en los archivos *javascript* del emulador web.

sapi_tick

Para lograr la interacción con la capa de emulación *C HAL*, y realizar la invocación periódica a la función que usa la macro `EMSCRIPTEN_KEEPALIVE` de *emscripten* se configuró en esta capa de desarrollo un temporizador de *javascript*.

Además, dentro del temporizador, se utilizó la función `ccall` de *emscripten*, que permite invocar a la función `tickerCallback` desde el código C compilado con *emscripten*.

A continuación, se muestra en la figura 3.9 el funcionamiento de `ccall`.

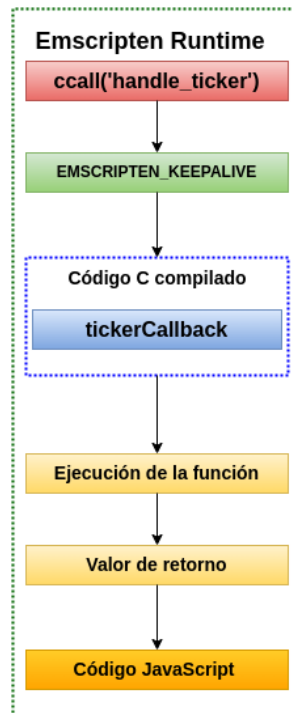


FIGURA 3.9. Diagrama de bloques de la función `ccall`.

Sin embargo, debido a la naturaleza asíncrona de *JavaScript* y al uso de la función `ccall`, la función no mantiene el contexto entre las ejecuciones del temporizador. En consecuencia, cada vez que se reinicia el temporizador y se ejecuta la función `tickerCallback`, la tarea específica programada por el usuario comienza desde el principio en lugar de continuar desde el punto donde quedó anteriormente.

3.4. Frontend

En esta parte de la plataforma web se implementó el desarrollo de la interacción entre el *backend* y el navegador del usuario.

3.4.1. JavaScript UI

Esta capa se desarrolló con el propósito de que se comuniquen con la capa *JavaScript HAL*, y también con el objetivo de proporcionar componentes de código, utilidades y manejo de eventos en la aplicación de usuario.

Por tanto, para lograr la comunicación con la capa *JavaScript HAL*, se desarrolló en esta capa los objetos que se suscriben al detector de eventos programados en la *HAL*.

JavaScript permite crear oyentes, utilizando el método `on()` y pasando como argumento el nombre del evento al que se quiere suscribir. De esta manera, se programaron varios subscriptores para un mismo evento en diferentes archivos *JavaScript* dentro de esta capa. En consecuencia, se logró una mayor interactividad entre los componentes de la plataforma.

Es decir, cuando se emite algún evento en la *HAL*, entonces el oyente suscrito a ese evento en esta capa *UI* lo podrá escuchar y realizar las acciones que correspondan para la funcionalidad requerida. La figura 3.10 describe esta situación.

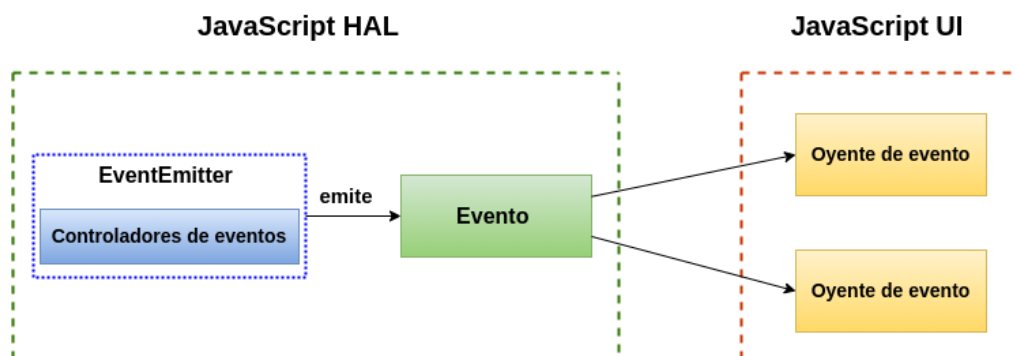


FIGURA 3.10. Diagrama de bloques de los oyentes de *EventEmitter* en la capa UI.

3.4.2. Aplicación de Usuario

La plataforma de emulación para la placa EDU-CIAA-NXP es una versión en línea, que se ejecuta en el *browser* del usuario. Además, la interfaz fue diseñada como una herramienta que permite al usuario realizar sus tareas de programación dentro de una plataforma simple e intuitiva.

Sin embargo, existen algunas limitaciones que se deben tener en cuenta.

- Las unidades de tiempo especificadas por los valores en las constantes, por ejemplo:

```
#define TASK1_PERIODICITY 1000
```

Cuando el usuario define `TASK1_PERIODICITY` con un valor de 1000 para ser usado dentro del siguiente código:

```
if( task1Counter++ == TASK1_PERIODICITY ){
    task1();
    task1Counter = 0;
}
```

Significa que la tarea planificada se ejecutará cada 1000 milisegundos en la placa física. Sin embargo, en el emulador web, la velocidad de ejecución puede variar y no necesariamente coincidir con el tiempo real. Por lo tanto, el tiempo de ejecución de cada iteración de la tarea `task1` podría ser más lento en el emulador web, por ejemplo, 1000 segundos. Estas diferencias en la ejecución se deben a las limitaciones inherentes de *JavaScript* y *emscripten*, que pueden afectar la precisión del tiempo.

- En el emulador dentro de un bucle infinito `while(1)`, es necesario agregar un retraso (`delay`), de lo contrario, el navegador no puede actualizar

la interfaz de la plataforma web ni responder a eventos del usuario. Esto significa que el navegador no tiene la oportunidad de realizar otras tareas o responder a eventos mientras el bucle está en ejecución. Como resultado, el navegador se bloquea o congela y puede dejar de responder.

3.5. Caso de estudio

Cuando un usuario ejecuta una aplicación simple como *Blinky* que enciende un led, entonces, la plataforma reflejará la acción en su interfaz, permitiendo al usuario ver el resultado dentro del entorno virtual.

La figura 3.11 muestra la interacción del usuario con los objetos del sistema y el orden en que se producen. Además, se muestra los mensajes que se pasan entre los objetos.

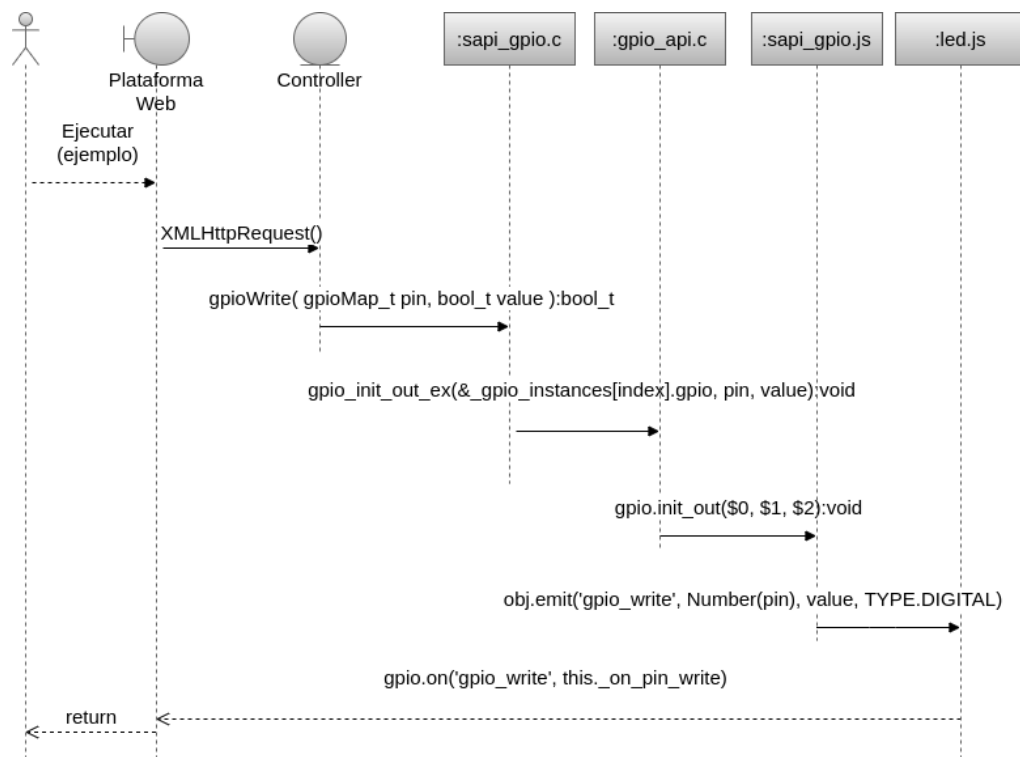


FIGURA 3.11. Interacción del usuario con los objetos del emulador.

En la plataforma se crea un objeto nativo del navegador web XMLHttpRequest que permite hacer solicitudes HTTP desde JavaScript. Luego, en el servicio de *backend* se invoca a la biblioteca C `sapi_gpio.c`, que llama a `gpio_api.c` y mediante el uso de funciones nativas de *emscripten* pasa el evento a la capa *JavaScript HAL*.

Además, mediante la función *write* se realiza la activación del evento, por lo tanto, el objeto emitirá el evento con el nombre `gpio_write` pasando como argumentos el número de pin, el valor digital y el tipo de pin declarado.

La figura 3.12 muestra el diagrama en bloques del evento con el nombre `gpio_write`.

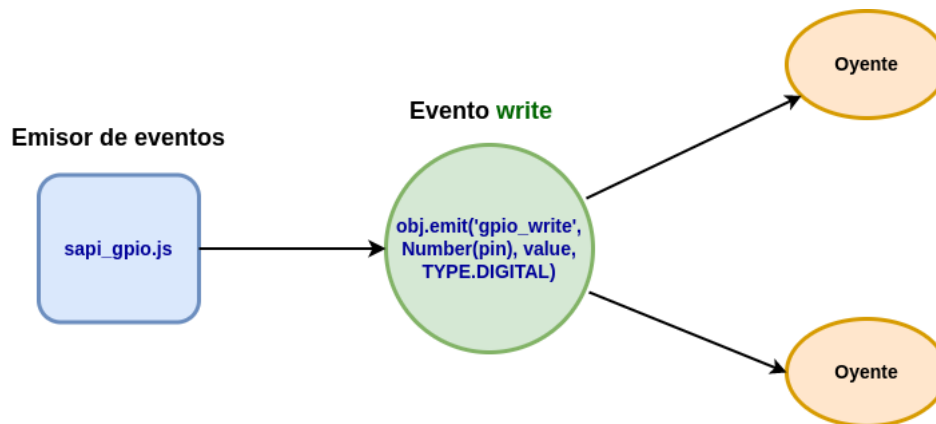


FIGURA 3.12. Activación de evento con el nombre `gpio_write`.

En ese sentido, en la capa *JavaScript UI* cuando se emite el evento con el nombre `gpio_write`, entonces el oyente suscrito a ese evento lo podrá escuchar y realizar las acciones que corresponden para la funcionalidad que se pide.

La figura 3.13 muestra el diagrama en bloques del oyente suscrito al evento `gpio_write`.

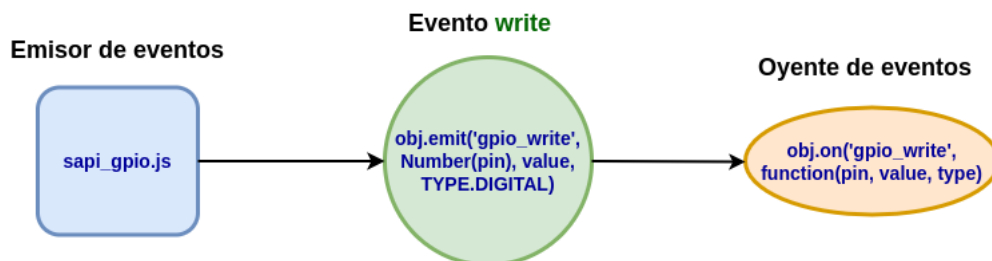


FIGURA 3.13. GPIO oyente del evento con el nombre `gpio_write`.

Entonces, en la plataforma web se mostrarán los cambios de `gpio_write` en la placa virtual.

En la figura 3.14 se presenta para una función de la GPIO, la interacción entre todas las capas de programación.

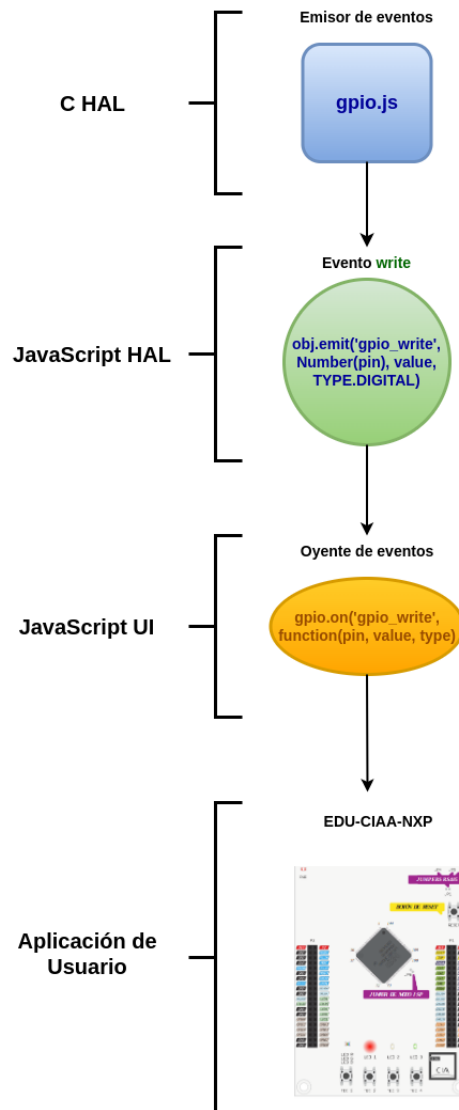


FIGURA 3.14. Interacción entre todas las capas de programación.

3.6. Diseño de la Interfaz de Usuario

Para el desarrollo de la interfaz, se optó por un diseño intuitivo, de manera que el usuario se sienta familiarizado con las herramientas de trabajo y que la disposición de los componentes sea cómoda y esté organizada al momento de usarlas.

La figura 3.15 muestra la parte izquierda de la interfaz gráfica para la aplicación de usuario y en la figura 3.16 muestra la parte derecha de la interfaz gráfica para la aplicación de usuario.



FIGURA 3.15. Parte izquierda de la plataforma de emulación para la placa EDU-CIAA-NXP.



FIGURA 3.16. Parte derecha de la plataforma de emulación de la placa EDU-CIAA-NXP.

Por consiguiente, el diseño de la interfaz de usuario de la plataforma proporciona las siguientes áreas:

- Área de ensamblado: el valor predeterminado muestra la placa EDU-CIAA-NXP, y también se permite agregar componentes.
- Área de codificación: se proporciona un editor de código en línea para programar con la placa EDU-CIAA-NXP. La primera vez que se accede a la plataforma se muestra en ejecución un ejemplo de código predeterminado.
- Área de consola integrada: se muestra en una ventana la salida que se vería a través del puerto serie.

Debido a las áreas de trabajo que presenta la plataforma, el usuario programador podrá realizar las siguientes tareas:

- Ver los programas de ejemplo predeterminados.
- Crear un nuevo proyecto.
- Ejecutar un programa de ejemplo o uno nuevo.
- Editar programas.
- Visualizar los cambios programados para la placa virtual.

- Agregar nuevos componentes.
- Ver los errores obtenidos en la programación.
- Ver lo programado en la salida de consola.

Área de ensamblado

Para el desarrollo de la placa EDU-CIAA-NXP se usó dibujos en formato de gráficos vectoriales bidimensionales (SVG). Primeramente, por las siguientes características:

- Evitan que las imágenes se deformen y no pierden calidad.
- Permite programar animaciones.
- Son más ligeras, entonces resulta que, al pesar menos se cargan más rápido.

Y en segundo lugar, por su capacidad de ser modificado por medio de *JavaScript*. Por lo tanto, se pudieron crear imágenes interactivas.

En ese sentido, para la capa de programación *JavaScript UI*, se pudo implementar el comportamiento interactivo para los botones de la placa (TEC1, TEC2, TEC3 y TEC4) usando el código SVG generado para la placa EDU-CIAA-NXP, incluso, se pudo modificar también el comportamiento de los LEDs y mostrarlo dinámicamente en la placa.

Además, en esta área se permite al usuario elegir uno o más dispositivos virtuales de entrada/salida y de esta manera integrarlo a la aplicación.

En la figura 3.17 se observa que el usuario puede elegir qué componente agregar a la aplicación y en la figura 3.18 se muestra que el componente se agregó a la aplicación.

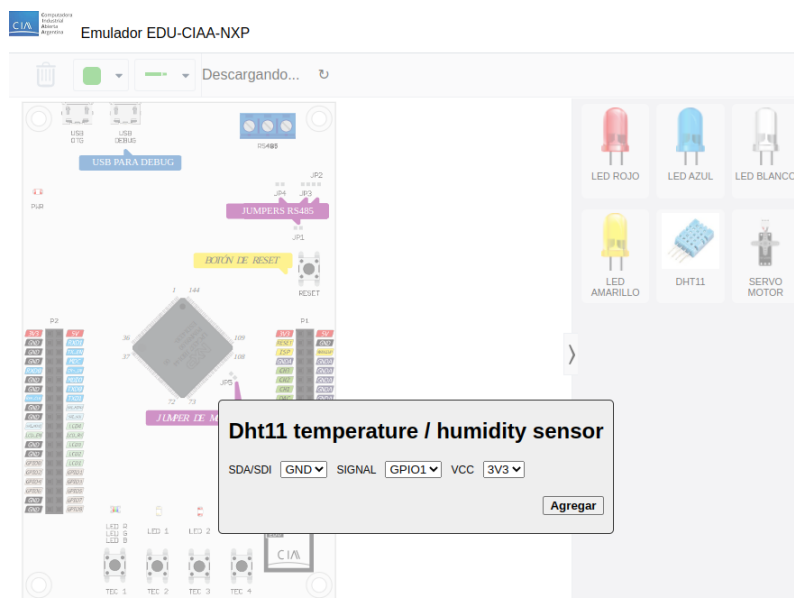


FIGURA 3.17. Agregar periférico.

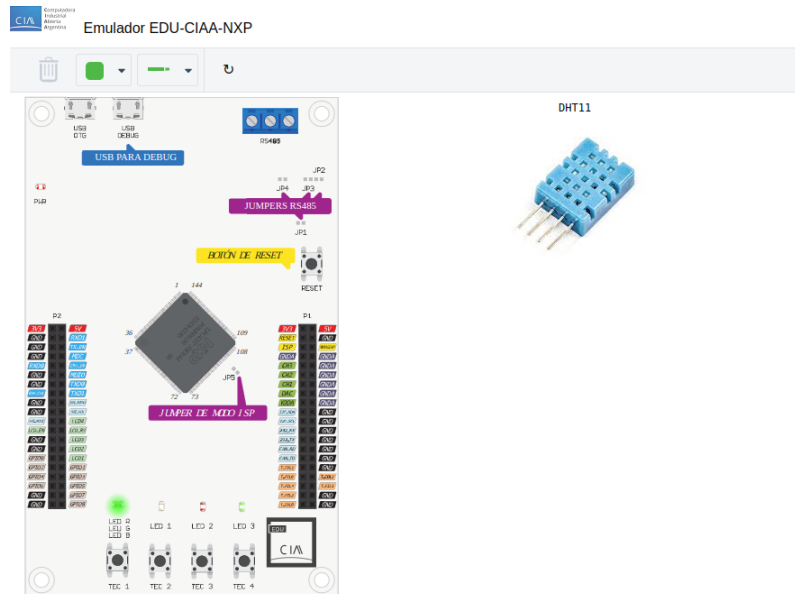


FIGURA 3.18. Periférico agregado en el área de ensamblado.

Área de codificación

Esta parte de la plataforma se reserva al usuario para que pueda programar sus propias aplicaciones. Esta ventana de edición presenta las siguientes capacidades:

- Manejar la sintaxis para el lenguaje C.
- Soportar el uso de las constantes, como por ejemplo: `'#define'`.
- Permitir el uso de las palabras claves, comentarios, etc.
- Permitir el resaltado de líneas de código, sangría automática y número de línea.
- Utilizar la función buscar (ctrl + f).
- Utilizar la función buscar/reemplazar (ctrl + h).
- Utilizar la función rehacer (ctrl + y).

Para compilar un programa, la plataforma provee al usuario el botón “ejecutar”. Sin embargo, si hubo problemas de sintaxis, errores lógicos, etc., se mostrarán esos errores al usuario.

En la figura 3.19 se muestra el código que generó los errores de compilación y en la figura 3.20 se observa los errores de compilación en el área de ensamblado.

```

1 #include "sapi.h"
2
3 // EDU-CIAA-NXP V1.1 primer tirada (abajo dice ASIS1)
4 #define edu_ciaa_nxp_v1_1_0
5
6 // EDU-CIAA-NXP V1.1 segunda tirada (abajo dice ASSEMBLI)
7 // #define edu_ciaa_nxp_v1_1_1
8
9 #if defined(edu_ciaa_nxp_v1_1_0)
10 #define LUZ_ROJO LED2
11 #define LUZ_AMARILLA LED1
12 #define LUZ_VERDE LED3
13 #elif defined(edu_ciaa_nxp_v1_1_1)
14 #define LUZ_ROJO LED1
15 #define LUZ_AMARILLA LED2
16 #define LUZ_VERDE LED3
17 #endif
18
19 #define TIEMPO_EN_ROJO 3000 // ms
20 #define TIEMPO_EN_AMARILLO 1000 // ms
21 #define TIEMPO_EN_VERDE 2000 // ms
22
23
24 int main() {
25     // ---- Setup ----
26     boardInit();
27
28     // ---- Repeat for ever ----
29     while( true ) {
30
31         gpioWrite( LUZ_ROJO, ON );
32         delay( TIEMPO_EN_ROJO );
33         gpioWrite( LUZ_ROJO, OFF );
34
35         gpioWrite( LUZ_AMARILLA, ON );
36         delay( TIEMPO_EN_AMARILLO );
37         gpioWrite( LUZ_AMARILLA, OFF );
38
39         gpioWrite( LUZ_VERDE, ON );
40         delay( TIEMPO_EN_VERDE );
41         gpioWrite( LUZ_VERDE, OFF );
42
43         gpioWrite( LUZ_AMARILLA, ON );
44         delay( TIEMPO_EN_AMARILLO );
45         gpioWriteON( LUZ_AZUL, OFF );
46     }
47 }
48

```

FIGURA 3.19. Código que generó los errores de compilación.

```

C:\A Emulador EDU-CIAA-NXP

Falló la compilación

La aplicación no se pudo compilar (1)
/outUser/user.1690213652001.c:45:7: error: implicit declaration of function 'gpioWriteON' is invalid in C99 [-Werror,-Wimplicit-function-declaration]
    gpioWriteON( LUZ_AZUL, OFF );
    ^
/outUser/user.1690213652001.c:45:7: note: did you mean 'gpioWrite'?
/hal/sapi/soc/peripherals/inc/sapi_gpio.h:83:8: note: 'gpioWrite' declared here
bool_t gpioWrite( gpioMap_t pin, bool_t value );
/outUser/user.1690213652001.c:45:20: error: use of undeclared identifier 'LUZ_AZUL'
    gpioWriteON( LUZ_AZUL, OFF );
                   ^
2 errors generated.
/home/jenny/plataforma/emsk/emscripten/1.38.21/emcc.py:810: SyntaxWarning: "is not" with a literal. Did you mean
"!="?
newargs = [arg for arg in newargs if arg is not '']
/home/jenny/plataforma/emsk/emscripten/1.38.21/emcc.py:921: SyntaxWarning: "is not" with a literal. Did you mean
"!="?
newargs = [a for a in newargs if a is not '']
shared:ERROR: compiler frontend failed to generate LLVM bitcode, halting

```

FIGURA 3.20. Errores de compilación.

Área de consola integrada

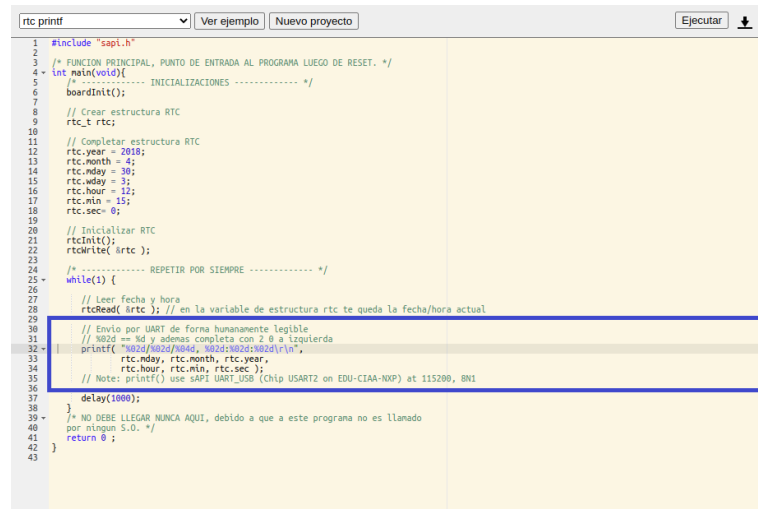
Para el desarrollo de esta ventana se usó la biblioteca Xterm.js, que es un componente de terminal de front-end escrito en JavaScript y que permite construir terminales en el navegador.

Entre las principales características se destaca:

- Funciona con la mayoría de las aplicaciones de terminal, como bash, es compatible con aplicaciones basadas y eventos de mouse.
- Es de alto rendimiento, por eso es realmente rápido.
- No requiere de dependencias externas para funcionar, Sin embargo, la dependencia principal para el funcionamiento básico es el propio navegador web.
- API bien documentada.

Además de las características de Xterm.js, esta biblioteca de JavaScript es usada por varios proyectos populares como VS Code, Hyper y Theia que lo usan, de ahí que, el soporte en la comunidad de desarrolladores es amplio.

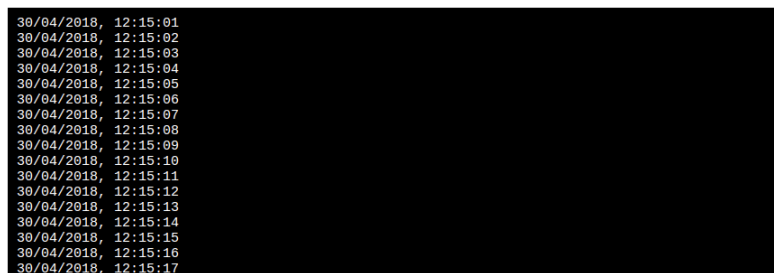
En la figura 3.21 se muestra un programa de usuario que generó la salida por consola.



```
1 #include "sapi.h"
2
3 /* FUNCION PRINCIPAL, PUNTO DE ENTRADA AL PROGRAMA LUEGO DE RESET. */
4 int main(void){
5     /* ----- INICIALIZACIONES ----- */
6     boardInit();
7
8     // Crear estructura RTC
9     rtc_t rtc;
10
11     // Completar estructura RTC
12     rtc.year = 2018;
13     rtc.month = 4;
14     rtc.nday = 30;
15     rtc.wday = 3;
16     rtc.hour = 12;
17     rtc.min = 15;
18     rtc.sec = 0;
19
20     // Inicializar RTC
21     rtcInit();
22     rtcWrite( rtc );
23
24     /* ----- REPETIR POR SIEMPRE ----- */
25     while(1) {
26         // Leer fecha y hora
27         rtcRead( rtc ); // en la variable de estructura rtc se queda la fecha/hora actual
28
29         // Envio por UART de forma humanamente legible
30         // %02d == %d y ademas completa con 2 0 a izquierda
31         printf( "%02d/%02d/%04, %02d:%02s:%02d\r\n",
32             rtc.nday, rtc.month, rtc.year,
33             rtc.hour, rtc.min, rtc.sec );
34         // Note: printf() use SAPI UART_USB (Chip USART2 on EDU-CIAA-NXP) at 115200, 8N1
35
36         delay(1000);
37     }
38     /* NO DEBE LLEGAR NUNCA AQUI, debido a que a este programa no es llamado
39     por ningún S.O. */
40     return 0;
41 }
42
43
```

FIGURA 3.21. Programa de usuario.

En la figura 3.22 se puede observar la salida por consola.



```
30/04/2018, 12:15:01
30/04/2018, 12:15:02
30/04/2018, 12:15:03
30/04/2018, 12:15:04
30/04/2018, 12:15:05
30/04/2018, 12:15:06
30/04/2018, 12:15:07
30/04/2018, 12:15:08
30/04/2018, 12:15:09
30/04/2018, 12:15:10
30/04/2018, 12:15:11
30/04/2018, 12:15:12
30/04/2018, 12:15:13
30/04/2018, 12:15:14
30/04/2018, 12:15:15
30/04/2018, 12:15:16
30/04/2018, 12:15:17
```

FIGURA 3.22. Salida de la terminal serie.

Para ver el código del presente trabajo, se puede visitar el repositorio de GitHub [55].

Capítulo 4

Ensayos y resultados

En este capítulo se presentan las pruebas realizadas para comprobar el correcto funcionamiento de la plataforma de emulación y las diferencias que presenta con respecto a la placa real. Además, se describen las diferentes herramientas que se utilizaron.

4.1. Pruebas de Unidad

Las pruebas de unidad se centraron en evaluar de manera aislada cada método de los archivos de código fuente de la biblioteca C, con el objetivo de que cada unidad de código funcione correctamente y produzca los resultados esperados.

Asimismo, para el desarrollo de las pruebas unitarias, se utilizaron *Check* y *CMocka*, que son bibliotecas de pruebas unitarias escritas en lenguaje C. *CMocka* proporcionó funcionalidades para simular o mockear las funciones y dependencias externas de *emscripten*, lo que permitió enfocarse en probar exclusivamente los módulos de la biblioteca C.

Además, para compilar las pruebas unitarias con *CMocka* o *Check*, se utilizó el compilador *GCC* (*GNU Compiler Collection*). El proceso consiste en compilar los archivos fuente de las pruebas y generar un archivo ejecutable que contiene el resultado del proceso de compilación y enlazado. Los resultados de las pruebas unitarias se mostrarán en la consola al ejecutar el archivo ejecutable generado.

Por ejemplo, si todas las pruebas pasaron con éxito, la consola mostrará el porcentaje de pruebas probadas, la cantidad de pruebas que aprobaron, la cantidad de pruebas que fallaron y si alguna prueba falla, entonces, indicará el error específico y además, proporcionará detalles adicionales para identificar el problema.

- El porcentaje de pruebas probadas.
- La cantidad de pruebas que aprobaron.
- La cantidad de pruebas que fallaron.
- Si alguna prueba falla, entonces, indicará el error específico.
- Si alguna prueba falla, proporcionará detalles adicionales para identificar el problema.

Se desarrollaron estas pruebas y se registraron los resultados en la consola. La figura 4.1 muestra la primera parte de la depuración de las pruebas unitarias y la figura 4.2 muestra la segunda parte.

```
jenny@jenny-pc: ~/Documents/UBA/Tesis/claa-emulador
jenny@jenny-pc:~/Documents/UBA/Tesis/claa-emulador$ gcc -DTEST_BUILD -o hal/claa_tests_dht11 hal/test/check
ck_sapi_dht11.c hal/sapi/external_peripherals/temperature_humidity/dht11/src/sapi_dht11.c hal/test/mock/d
ht11_api_mock.c -I. -I./hal/sapi -I./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/external_
peripherals/temperature_humidity/dht11/inc -lcheck -lcmocka -lrt -pthread -lm -lsunit
jenny@jenny-pc:~/Documents/UBA/Tesis/claa-emulador$ ./hal/claa_tests_dht11
Running suite(s): CMocka Suite sapi_dht11 dht11Read
100%: Checks: 1, Failures: 0, Errors: 0
Running suite(s): Check Suite sapi_dht11 dht11Init
Mock: dht11_init invocado con gpio = 10
100%: Checks: 1, Failures: 0, Errors: 0
jenny@jenny-pc:~/Documents/UBA/Tesis/claa-emulador$ gcc -DTEST_BUILD -o hal/claa_tests_gpio hal/test/check
ck_sapi_gpio.c hal/sapi/soc/peripherals/src/sapi_gpio.c hal/test/mock/gpio_api_mock.c hal/test/mock/sapi_
interrupt_mock.c -I. -I./hal/sapi -I./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/per
ipherals/inc -lcheck -lcmocka -lrt -pthread -lm -lsunit
jenny@jenny-pc:~/Documents/UBA/Tesis/claa-emulador$ ./hal/claa_tests_gpio
Running suite(s): Check Suite sapi_gpio gpioInit
Mock: gpio_init invocado con gpioMap_t = 50
100%: Checks: 1, Failures: 0, Errors: 0
Running suite(s): CMocka Suite sapi_gpio gpioRead, gpioWrite, gpioToggle
Mock: gpio_init_out invocado con gpioMap_t = 50
Mock: gpio_init_out invocado con gpioMap_t = 50
100%: Checks: 3, Failures: 0, Errors: 0
jenny@jenny-pc:~/Documents/UBA/Tesis/claa-emulador$ gcc -DTEST_BUILD -o hal/claa_tests_rtc hal/test/check
ck_sapi_rtc.c hal/sapi/soc/peripherals/src/sapi_rtc.c hal/test/mock/rtc_api_mock.c hal/test/mock/sapi_dela
y_mock.c -I. -I./hal/sapi -I./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/peripherals/
inc -lcheck -lcmocka -lrt -pthread -lm -lsunit
jenny@jenny-pc:~/Documents/UBA/Tesis/claa-emulador$ ./hal/claa_tests_rtc
Running suite(s): Check Suite sapi_rtc rtcInit
Mock: rtc_init invocado
Mock: delay invocado con duration_ms = 2100
100%: Checks: 1, Failures: 0, Errors: 0
Running suite(s): CMocka Suite sapi_rtc rtcRead rtcWrite
Mock: rtc_write invocado
100%: Checks: 2, Failures: 0, Errors: 0
```

FIGURA 4.1. Primera parte de la depuración de las pruebas unitarias con *CMocka* o *Check*.

```
jenny@jenny-pc: ~/Documents/UBA/Tesis/claa-emulador
jenny@jenny-pc:~/Documents/UBA/Tesis/claa-emulador$ gcc -DTEST_BUILD -o hal/claa_tests_delay hal/test/check
ck_sapi_delay.c hal/sapi/soc/peripherals/src/sapi_delay.c hal/test/mock/delay_api_mock.c hal/test/mock/sap
i_tick_mock.c -I. -I./hal/sapi -I./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/periph
erals/inc -lcheck -lcmocka -lrt -pthread -lm -lsunit
jenny@jenny-pc:~/Documents/UBA/Tesis/claa-emulador$ ./hal/claa_tests_delay
Running suite(s): Check Suite sapi_delay delayInaccurateMs, delayInaccurateUs, delayInaccurateNs, delay, d
elayInit, delayWrite
Mock: delay_ms invocado con int = 500
Mock: delay_us invocado con int = 1000
Mock: delay_ns invocado con int = 100
Mock: delay_ms invocado con int = 200
100%: Checks: 6, Failures: 0, Errors: 0
Running suite(s): CMocka Suite sapi_delay delayRead
100%: Checks: 1, Failures: 0, Errors: 0
jenny@jenny-pc:~/Documents/UBA/Tesis/claa-emulador$ gcc -DTEST_BUILD -include /dev/null -o hal/claa_tests
tick hal/test/check_sapi_tick.c hal/sapi/soc/peripherals/src/sapi_tick.c hal/test/mock/tick_api_mock.c hal
/test/mock/sapi_uart_mock.c hal/test/mock/queue_mock.c -I. -I./hal/sapi -I./hal/test/mock -Ihal/sapi/base
-Ihal/sapi/board -Ihal/sapi/soc/peripherals/inc -lcheck -lcmocka -lrt -pthread -lm -lsunit
jenny@jenny-pc:~/Documents/UBA/Tesis/claa-emulador$ ./hal/claa_tests_tick
Running suite(s): Check Suite sapi_tick tickInit, tickWrite, tickPowerSet, tickRead
Mock: tick_detach invocado con id = 1
Mock: tick_init invocado con ms = 1
100%: Checks: 4, Failures: 0, Errors: 0
jenny@jenny-pc:~/Documents/UBA/Tesis/claa-emulador$ gcc -DTEST_BUILD -o hal/claa_tests_tasks hal/test/check
ck_tasks.c hal/test/mock/tasks_api_mock.c -Ihal/sapi/base -Ihal/sapi/board -I./hal/test/mock -lcheck
-lcmocka -lrt -pthread -lm -lsunit
jenny@jenny-pc:~/Documents/UBA/Tesis/claa-emulador$ ./hal/claa_tests_tasks
Running suite(s): CMocka Suite task getTickCount
Mock: xStartTickCount invocado
100%: Checks: 1, Failures: 0, Errors: 0
Running suite(s): Check Suite task xStartTickCount setTaskDelayUntil
Mock: xStartTickCount invocado
Mock: setTaskDelayUntil invocado con xTimeIncrement = 2
100%: Checks: 2, Failures: 0, Errors: 0
```

FIGURA 4.2. Segunda parte de la depuración de las pruebas unitarias con *CMocka* o *Check*.

4.2. Pruebas de Integración

Las pruebas de integración se centran en evaluar la interacción y comunicación entre diferentes componentes. Además, asegura que trabajen en conjunto sin problemas.

A medida que se fueron desarrollando diferentes módulos y funcionalidades del emulador, las pruebas de integración fueron necesarias para identificar posibles conflictos o incompatibilidades entre los distintos componentes de código del emulador web.

Se fueron identificando las interacciones entre componentes que fueron relevantes a partir de las pruebas de unidad existentes, como `sapi_delay` y `sapi_tick`. Luego, se identificaron las dependencias de `emscripten` y las funciones que se invocan entre las diferentes pruebas de unidad.

Luego, se crearon archivos de prueba de integración con escenarios específicos que combinen las interacciones entre los componentes y se utilizaron mocks para simular comportamientos de funciones de `emscripten`.

la interacción con el usuario así como también con las peticiones hacia el backend.

Al igual que en las pruebas de unidad se utilizó el compilador GCC para compilar los archivos de código relevantes y cualquier dependencia necesaria.

A continuación se muestra los resultados por consola

delays con ticks gpio con interrupciones

En la figura 4.3 muestra la depuración de las pruebas de integración.

```

jenny@jenny-pc: ~/Documents/UBA/Tesis/ciaa-emulador
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ gcc -DTEST_BUILD -o hal/ciaa_test_integration_delay_tick hal/test/check_test_integration_delay_tick.c hal/sapi/soc/peripherals/src/sapi_delay.c hal/test/mock/delay_api_mock.c hal/test/mock/sapi_tick_mock.c -I. -I./hal/sapi -I./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/peripherals/inc -lcheck -lcmocka -lrt -pthread -lm -lsubunit
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ ./hal/ciaa_test_integration_delay_tick
Running suite(s): Test de Integración
100%: Checks: 1, Failures: 0, Errors: 0
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ gcc -DTEST_BUILD -o hal/ciaa_test_integration_gpio_interrupt hal/test/check_test_integration_gpio_interrupt.c hal/sapi/soc/peripherals/src/sapi_gpio.c hal/test/mock/gpio_api_mock.c hal/test/mock/sapi_interrupt_mock.c -I. -I./hal/sapi -I./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/peripherals/inc -lcheck -lcmocka -lrt -pthread -lm -lsubunit
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ ./hal/ciaa_test_integration_gpio_interrupt
Running suite(s): Test de Integración
Mock: gpio_init_out invocado con gpioMap_t = 50
Mock: gpio_init_out invocado con gpioMap_t = 50
100%: Checks: 3, Failures: 0, Errors: 0
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ gcc -DTEST_BUILD -o hal/ciaa_test_integration_rtc_delay hal/test/check_test_integration_rtc_delay.c hal/sapi/soc/peripherals/src/sapi_rtc.c hal/test/mock/rtc_api_mock.c hal/test/mock/sapi_delay_mock.c -I. -I./hal/sapi -I./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/peripherals/inc -lcheck -lcmocka -lrt -pthread -lm -lsubunit
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ ./hal/ciaa_test_integration_rtc_delay
Running suite(s): Test de Integración
Mock: rtc_init invocado
Mock: delay invocado con duration_ms = 2100
100%: Checks: 1, Failures: 0, Errors: 0

```

FIGURA 4.3. Depuración de las pruebas de integración.

Además, para probar código desde el frontend y el backend, se realizó prueba de ejecución del botón ejecutar de la interfaz, con ello se realiza el programa.

4.3. Pruebas de Interfaz

Para lograr que la interfaz de la plataforma de emulación cumpla con los requisitos funcionales y logre que los usuarios lo adopten con éxito fue necesario implementar las pruebas de la interfaz de usuario.

Por tanto, se implementaron pruebas automatizadas que verifiquen que el funcionamiento sea el correcto, tanto desde la interacción con el usuario así como también con las peticiones hacia el backend.

La implementación de pruebas automatizadas permitió ejecutarlas de forma rápida y confiable de manera recurrente.

Para automatizar las pruebas de la interfaz de usuario con *NodeJS*, el desarrollo se hizo mediante las bibliotecas *Mocha* y *Chai*, que permitieron crear pruebas unitarias muy completas para el desarrollo en JavaScript.

Además, permitió asegurar que cada módulo de la interfaz funcione correctamente por separado. Incluso, verificó si el código en el navegador web devolvió los nombres de los módulos correctos, los tipos de parámetros previstos y el tipo de retorno esperado.

La figura 4.4 muestra la primera parte de la depuración de las pruebas unitarias de *Mocha* en *Visual Studio Code*.



```
Jenny@jenny-pc: ~/Documents/UBA/Tesis/ciaa-emulador
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ npm run test
> ciaa-emulador@ test /home/jenny/Documents/UBA/Tesis/ciaa-emulador
> mocha

EDU-CIAA-NXP Emulador
  ✓ Contenido de la página principal

Contenido del ejemplo Blinky
  ✓ Compruebe que el ejemplo Blinky se ejecuta en el primer acceso

Blinky
  ✓ Ejecutar ejemplo blinky

Prueba del ejemplo static_non_freeRTOS_blinky
  ✓ El ejemplo static_non_freeRTOS_blinky debe cargarse en la Plataforma web

Prueba del ejemplo rtos_cooperative
  ✓ El ejemplo rtos_cooperative debe cargarse en la Plataforma web

Prueba del ejemplo non_blocking_delay
  ✓ El ejemplo non_blocking_delay debe cargarse en la Plataforma web

Prueba del ejemplo modular_tasks
  ✓ El ejemplo modular_tasks debe cargarse en la Plataforma web

Prueba del ejemplo rtc_printf
  ✓ El ejemplo rtc_printf debe cargarse en la Plataforma web

Prueba del ejemplo switches leds
  ✓ El ejemplo switches leds debe cargarse en la Plataforma web

Prueba del ejemplo button
  ✓ El ejemplo button debe cargarse en la Plataforma web

Prueba del ejemplo tick_callback
  ✓ El ejemplo tick_callback debe cargarse en la Plataforma web

Prueba del botón Ejecutar
  ✓ El botón Ejecutar debe funcionar correctamente al hacer clic (1027ms)

Prueba del botón Ver ejemplo
  ✓ El botón Ver ejemplo debe funcionar correctamente al hacer clic (1010ms)

Prueba del botón Nuevo proyecto
  ✓ El botón Nuevo proyecto debe funcionar correctamente al hacer clic (1045ms)

Prueba del botón Descargar
  ✓ El botón Descargar debe funcionar correctamente al hacer clic (1047ms)

15 passing (11s)
```

FIGURA 4.4. Primera parte de la depuración de las pruebas unitarias.

4.4. Integración Continua

Travis ci

Gitlab

4.5. Pruebas de Validación

4.6. Banco de pruebas

Para verificar el funcionamiento de la plataforma de emulación se emplearon diversos recursos de software y hardware, que permitió una evaluación completa

del funcionamiento de la plataforma de emulación, garantizando su confiabilidad y funcionalidad.

El proceso de verificación incluyó también pruebas comparativas entre la plataforma de emulación y la placa física, donde se ejecutaron casos de prueba idénticos en ambos entornos. Esto permitió identificar y analizar las diferencias entre el comportamiento del emulador web y la real real, proporcionando información valiosa para mejorar la precisión y fiabilidad de la plataforma web de emulación.

En la tabla 4.1 se presentan los recursos de hardware empleados en el banco de pruebas.

TABLA 4.1. Recursos de hardware utilizados.

Herramienta	Propósito
Computadora	Acceso a la plataforma de emulación.
Placa EDU-CIAA-NXP	Implementación de los ejemplos de la sAPI.
Dht11 temperature & humidity	Pruebas de ejemplo.

Asimismo, se utilizaron herramientas de software para realizar las pruebas en todos los módulos que componen el sistema. En la tabla 4.2 se describe el propósito de estas herramientas.

TABLA 4.2. Recursos de software utilizados.

Herramienta	Propósito
Mocha	Pruebas automatizadas para el frontend.
Chai	Pruebas automatizadas para el frontend.
Chrome	Pruebas de la plataforma web.
Firefox	Pruebas de la plataforma web.
Explorer	Pruebas de la plataforma web.
PostMan [56]	Pruebas de <i>request</i> de la plataforma y APIs.
Gtest	Pruebas automatizadas para el backend.
Make [57]	Para gestión de dependencias en las pruebas de backend.
CTest [58]	Para la ejecución de las pruebas de backend.
Tera Term [59]	Emulador de la terminal serial.

4.7. Casos de uso

En esta sección se realiza el análisis de algunos casos de prueba y se muestran los resultados obtenidos.

4.7.1. Prueba de acceso

ID Caso de prueba: CP01

Descripción: la primera vez que el usuario ingresa a la plataforma de emulación se muestra ejecutado el ejemplo predeterminado *Blinky*.

Pre-condición:

- La computadora del usuario tiene una conexión a internet.

Flujo principal:

- El usuario ingresa al entorno web de la plataforma de emulación para la placa EDU-CIAA-NXP.

Post condiciones:

- ÉXITO: la plataforma muestra el ejemplo *Blinky* en ejecución.
- FALLA: La plataforma no muestra ningún ejemplo predeterminado en ejecución.

Resumen del Test:

- Después de acceder a la plataforma mediante los navegadores web Chrome, Firefox y Explorer se comprobó que se muestra en ejecución el ejemplo *Blinky*.
- Se realizó la prueba de HTTP *requests* por medio de la utilización de la herramienta *Postman* que luego de acceder a la dirección donde se encuentra alojada la plataforma, devolvió en formato *JSON* la respuesta del servidor.

La figura 4.5 muestra la plataforma de emulación ejecutando el ejemplo predeterminado *Blinky*.

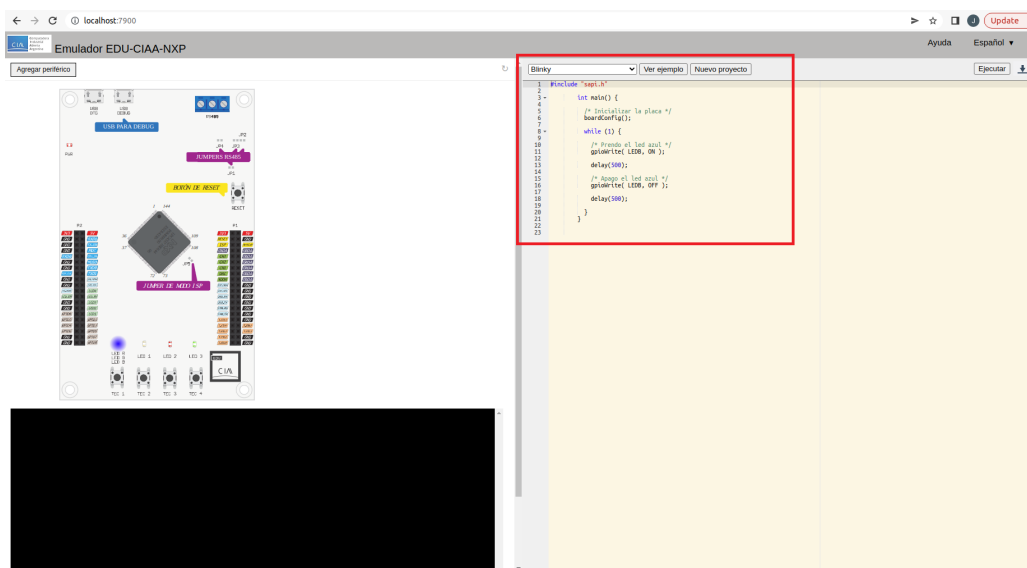


FIGURA 4.5. Prueba plataforma emulador ejecutando el ejemplo *Blinky*.

Postman es una aplicación que permite realizar pruebas API. Con este cliente *HTTP* se probó *HTTP requests* que permitió el acceso a la plataforma de emulación y por medio de la cual se obtuvo la respuesta en diferentes formatos como *JSON*, *XML*, *HTML* y *Text*.

En la figura 4.6 se muestra la petición de acceso a la plataforma por medio de la herramienta *Postman*.

En la figura 4.7 se muestra la respuesta a la petición de acceso de la plataforma por medio de la herramienta *Postman*.

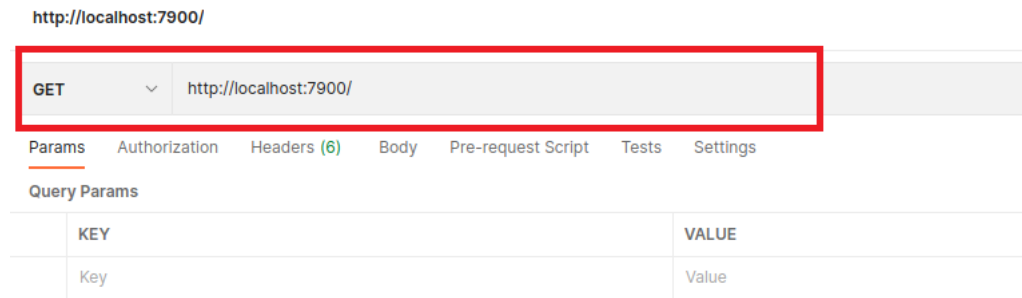


FIGURA 4.6. Petición de acceso a la plataforma.

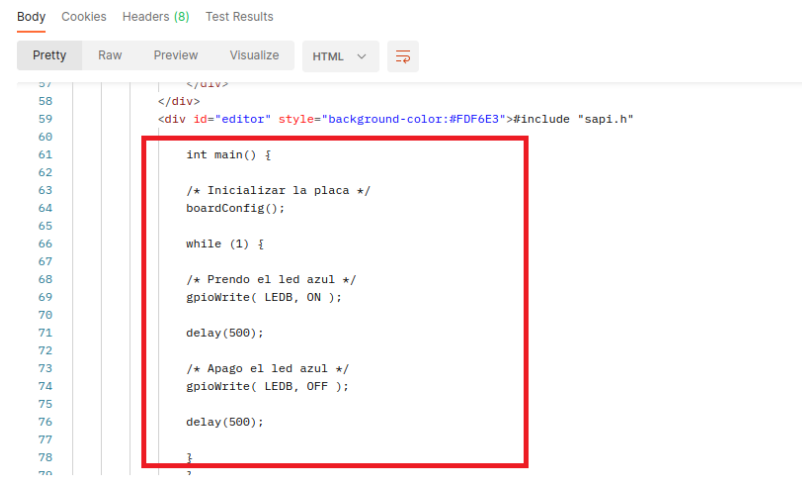


FIGURA 4.7. Respuesta del servidor.

4.7.2. Pruebas de precisión

Comparar la precisión de los cálculos realizados por el microcontrolador en la plataforma web con los resultados obtenidos en la placa física.

4.7.3. Prueba de funcionamiento

ID Caso de prueba: CP02

Descripción: la plataforma de emulación permite al usuario ejecutar el ejemplo *Dht11 temperature/humidity*.

Pre-condición:

- La computadora del usuario tiene una conexión a internet.

Flujo principal:

- El usuario ingresa al entorno web de la plataforma de emulación para la placa EDU-CIAA-NXP.

- El usuario selecciona desde la lista desplegable el ejemplo *Dht11 temperature/humidity* y hace click en el botón "Ver ejemplo".
- La plataforma muestra en pantalla al usuario el código del ejemplo *Dht11 temperature/humidity*.
- El usuario hace click en el botón "Agregar periférico".
- La plataforma muestra al usuario una nueva ventana con una lista desplegable de selección de periféricos.
- El usuario selecciona la opción "temperature/humidity".
- La plataforma muestra en el área de ensamblado el componente *Dht11 temperature/humidity* junto a la placa EDU-CIAA-NXP.
- El usuario hace click en botón "Ejecutar".
- La plataforma muestra los cambios en la placa EDU-CIAA-NXP y en la terminal de salida.

Post condiciones:

- ÉXITO: la plataforma muestra en ejecución el ejemplo *Dht11 temperature/humidity*.
- FALLA: La plataforma no muestra al usuario ningún cambio en pantalla.

Resumen del Test:

- Después de acceder a la plataforma mediante los navegadores web Chrome, Firefox y Explorer se comprobó que se muestra en ejecución el ejemplo *Dht11 temperature/humidity*.
- Se implementó en la placa real EDU-CIAA-NXP el mismo ejemplo *Dht11 temperature/humidity* y se registraron los resultados.
- Se realizó la prueba de HTTP *requests* usando la herramienta *Postman* para comprobar la respuesta del *API openweathermap* que consume la plataforma de emulación de manera que, los datos de temperatura y humedad sean los mismos.

Ensayo en la plataforma EDU-CIAA-NXP

Este ensayo tuvo como objetivo identificar las diferencias entre los resultados reales producidos por la placa real y los resultados esperados en la plataforma de emulación.

El primer paso fue conectar el componente Dht11 a la placa EDU-CIAA-NXP. En consecuencia, se procedió a compilar el ejemplo *Dht11 temperature/humidity* de la plataforma de emulación dentro de la herramienta *Eclipse* y luego, se grabó en la placa EDU-CIAA-NXP.

A continuación se muestra en la figura 4.8 la ejecución del ensayo en la plataforma EDU-CIAA-NXP.

La figura 4.9 muestra el código del ejemplo *Dht11 temperature/humidity* en la herramienta *eclipse* de la PC de prueba.

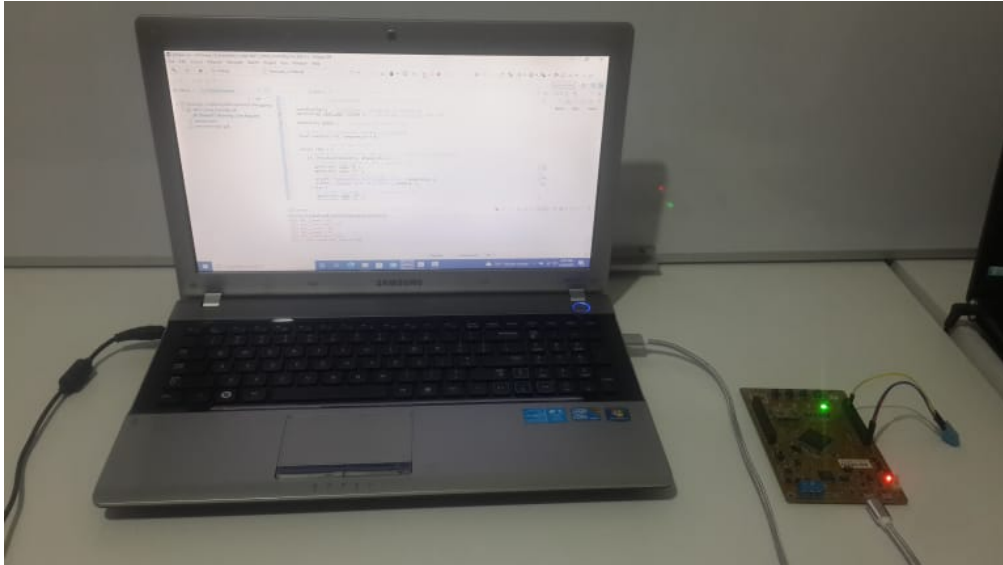
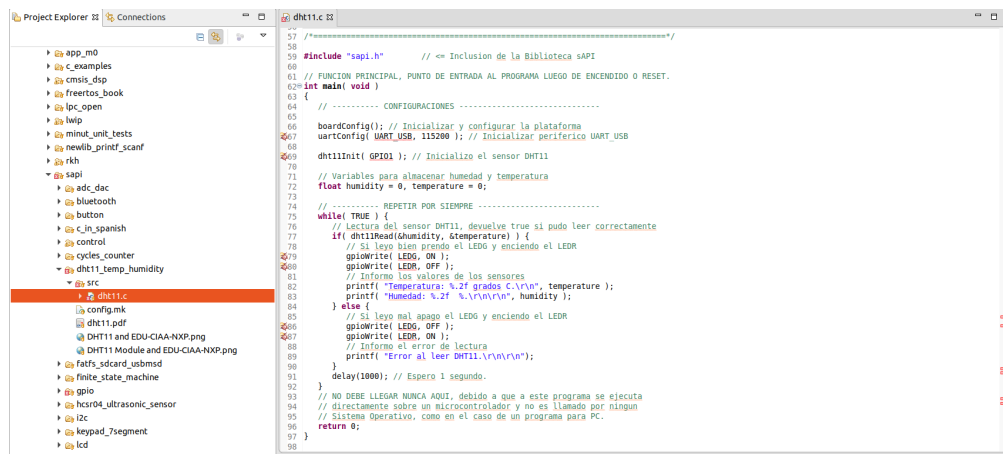
FIGURA 4.8. Ensayo del ejemplo *Dht11 temperature/humidity*.

FIGURA 4.9. Código del ejemplo en eclipse.

El programa *Dht11 temperature/humidity* de la plataforma de emulación es un ejemplo simple que solo enciende el LEDG en la placa y además, lee los datos generados del sensor Dht11 que consisten en temperatura/humedad. Luego, los datos leídos se imprimen por pantalla.

En este ensayo manual se registraron los cambios en la placa y también, los mensajes de la terminal serie. De modo que, posteriormente, permitió compararlos con la plataforma de emulación.

En la figura 4.10 se observan los cambios en la placa que fueron registrados durante las pruebas.



FIGURA 4.10. Cambios en la placa EDU-CIAA-NXP durante el ensayo.

Ahora bien, para leer los datos por pantalla se utilizó la herramienta *Tera Term VT* que permitió levantar los datos de temperatura/humedad.

En la figura 4.11 se muestra los datos de temperatura/humedad usando *Tera Term VT*.

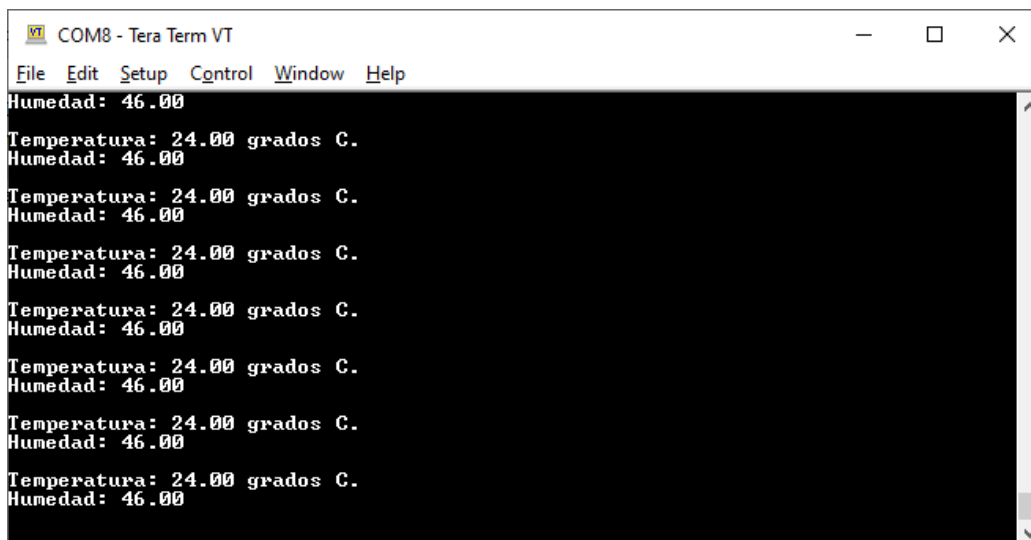


FIGURA 4.11. Salida de la terminal COM8 -Tera Term VT.

Ensayo en la plataforma de emulación para la placa EDU-CIAA-NXP

Después de registrar los resultados del ensayo del programa *Dht11 temperatur/humidity* en la plataforma real de la placa EDU-CIAA-NXP, se continuó con el ensayo en la plataforma de emulación siguiendo los pasos del caso de uso CP02.

En consecuencia, luego de completar el último paso y llegar a la post condición se obtuvo el siguiente resultado:

- ÉXITO: la plataforma muestra en ejecución el ejemplo *Dht11 temperature/humidity*.

La figura 4.12 muestra en ejecución el ejemplo *Dht11 temperature/humidity*.

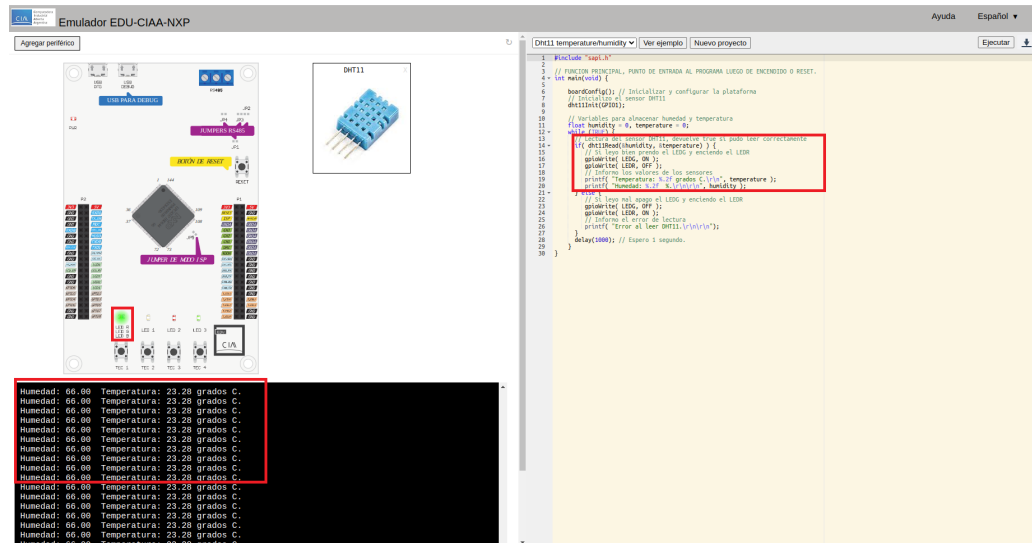


FIGURA 4.12. La plataforma muestra el resultado del CP02.

Primero, se verificó que el ejemplo *Dht11 temperature/humidity* es el que se utilizó para el ensayo en la plataforma real EDU-CIAA-NXP.

En la figura 4.13 se puede observar el ejemplo *Dht11 temperature/humidity* ejecutado en la plataforma de emulación.



FIGURA 4.13. Ejemplo *Dht11 temperature/humidity* ejecutado en la plataforma de emulación.

En segundo lugar, se verificó la salida de los datos de temperatura/humedad en la terminal serie.

Cabe destacar que para imitar los datos generados del sensor de temperatura/humedad, la plataforma de emulación se conecta al API de meteorología *openweathermap* que permite obtener datos meteorológicos de una región en particular. De manera que, la plataforma de emulación muestra la temperatura y humedad de la ciudad/país desde donde se accede a la plataforma web.

En la figura 4.14 se pueden observar los datos de temperatura/humedad devueltos por la plataforma de emulación.

```

Humedad: 66.00 Temperatura: 23.28 grados C.
Humedad: 66.00 Temperatura: 23.28 grados C.
Humedad: 66.00 Temperatura: 23.28 grados C.
Humedad: 66.00 Temperatura: 23.28 grados C.
Humedad: 66.00 Temperatura: 23.28 grados C.
Humedad: 66.00 Temperatura: 23.28 grados C.
Humedad: 66.00 Temperatura: 23.28 grados C.
Humedad: 66.00 Temperatura: 23.28 grados C.
Humedad: 66.00 Temperatura: 23.28 grados C.
Humedad: 66.00 Temperatura: 23.28 grados C.
Humedad: 66.00 Temperatura: 23.28 grados C.
Humedad: 66.00 Temperatura: 23.28 grados C.
Humedad: 66.00 Temperatura: 23.28 grados C.
Humedad: 66.00 Temperatura: 23.28 grados C.
Humedad: 66.00 Temperatura: 23.28 grados C.

```

FIGURA 4.14. Datos de temperatura/humedad mostrados por la plataforma.

Para verificar que la plataforma obtuvo los datos de temperatura/humedad del API de meteorología *openweathermap* se hicieron pruebas de request con la herramienta *Postman*.

En la figura 4.15 se observa que la petición de datos de temperatura/humedad recibe como parámetro la ciudad que se quiere consultar.

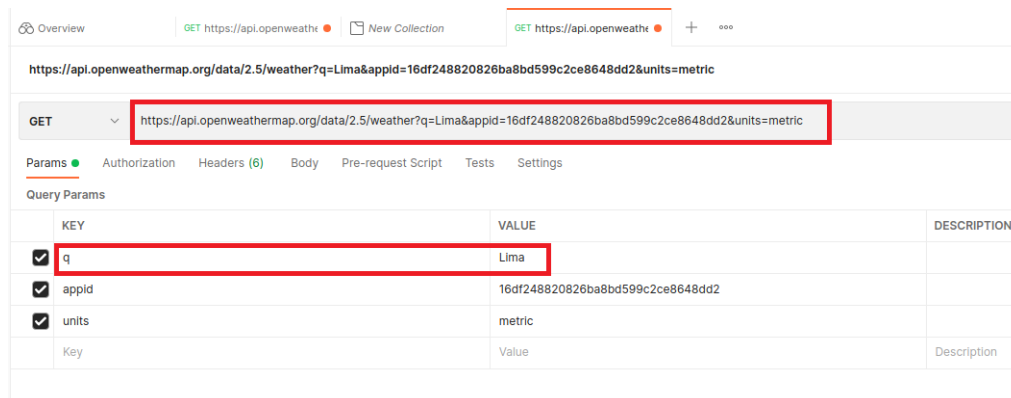


FIGURA 4.15. Petición de datos de temperatura/humedad.

En este paso se verifica que la respuesta de la petición de datos de temperatura/humedad al API *openweathermap* coincide con lo que se observó en la terminal de la plataforma de emulación.

En la figura 4.16 se observa la respuesta de la petición de datos de temperatura/humedad de la herramienta *Postman*.

Como último paso se verificó que los cambios en la placa coinciden con los que se registraron en la plataforma real de la placa EDU-CIAA-NXP. En consecuencia, se observó encendido el LEDG.

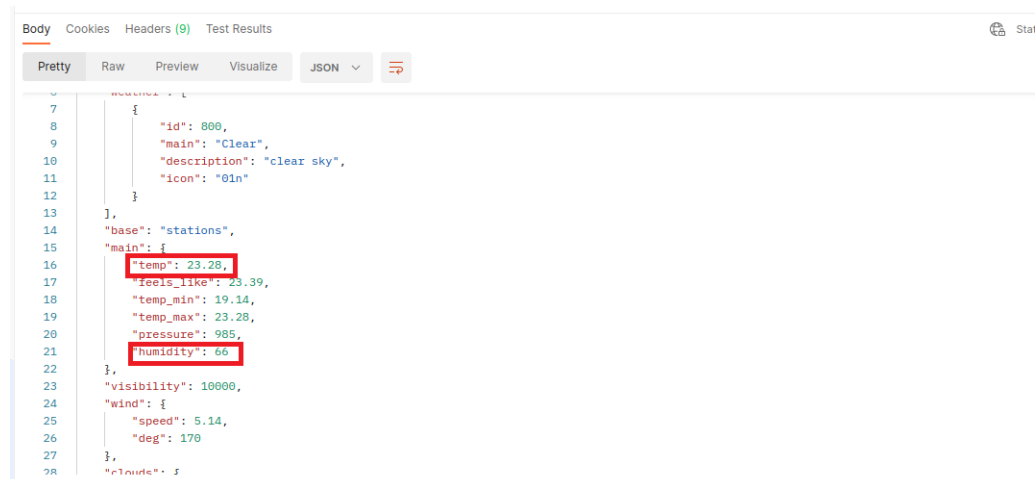


FIGURA 4.16. Respuesta de la petición de datos de temperatura/humedad.

En la figura 4.17 se observan los cambios en la placa EDU-CIAA-NXP de la plataforma de emulación.

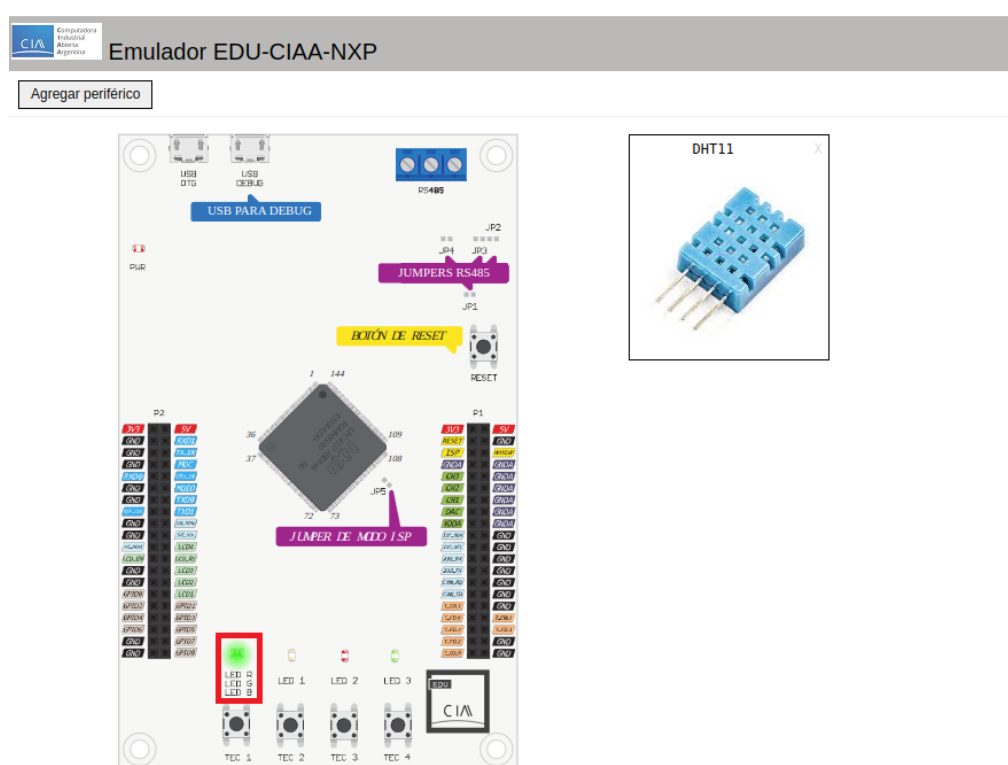


FIGURA 4.17. Cambios en la placa EDU-CIAA-NXP durante el ensayo.

Capítulo 5

Conclusiones

En este capítulo se presentan los aspectos más relevantes del trabajo realizado y se identifican los pasos a seguir.

5.1. Objetivos alcanzados

En el trabajo realizado se logró diseñar e implementar una plataforma de emulación para la placa EDU-CIAA-NXP mediante tecnología web. Se destacan a continuación los aportes del trabajo.

- El desarrollo de una plataforma de emulación para la placa EDU-CIAA-NXP que realiza un aporte al proyecto CIAA y a la comunidad de sistemas embebidos en general.
- El diseño de un sistema modular y flexible que permite agregar fácilmente nuevas funcionalidades.
- El desarrollo de una plataforma abierta que permite la colaboración de otros desarrolladores.
- La implementación de un sistema usable que facilita el aprendizaje y promueve la enseñanza de programación en sistemas embebidos.
- El desarrollo de una plataforma que es especialmente útil para realizar un prototipado rápido o pruebas de concepto sin depender de la placa.
- Emulación a nivel de API de la biblioteca sAPI del proyecto CIAA.
- Implementación de ejemplos funcionales predeterminados en la plataforma de emulación.
- La realización de pruebas de acceso y de funcionamiento para validar los resultados que se esperan de la plataforma on-line.
- Implementación de pruebas unitarias y de integración en la interfaz de usuario que verifican el cumplimiento de los requisitos funcionales.
- La creación de una herramienta que puede ser una nueva rama de desarrollo para el proyecto CIAA.

En este trabajo fue fundamental los conocimientos y habilidades adquiridos en las diferentes asignaturas de la carrera, destacando: implementación de manejadores de dispositivos, implementación de sistemas operativos, sistemas operativos de tiempo real y testing de software embebido.

5.2. Próximos pasos

A continuación, se indican las principales líneas de trabajo futuro para continuar con el desarrollo de la plataforma de emulación.

- Incorporar otras plataformas de hardware del proyecto CIAA.
- Emular nuevos periféricos, tales como servo motores, PWM, I2C, etc.
- Agregar características gráficas entre las conexiones de la placa y los periféricos.
- Implementar herramientas de depuración que permitan observar los valores de las variables, monitorear el flujo del programa y detectar posibles errores en el código..
- Implementar otros componentes o funcionalidades proporcionados por las bibliotecas de FreeRTOS, tales como queues, prioridades y hooks.

Bibliografía

- [1] Proyecto CIAA. *Plataforma educativa del Proyecto CIAA*. Visitado el 2023-06-15. 2014. URL: <http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:edu-ciaa:edu-ciaa-nxp>.
- [2] Proyecto CIAA. *Computadora Industrial Abierta Argentina*. Visitado el 2023-06-15. 2014. URL: <http://proyecto-ciaa.com.ar/devwiki/doku.php?id=start>.
- [3] Reinier Millo Sánchez, Alexis Fajardo Moya y Waldo Paz Rodríguez. «QEMU, una alternativa libre para la emulación de arquitecturas de hardware». En: (2022). Visitado el 2022-03-15. URL: https://www.researchgate.net/profile/Reinier-Millo-Sanchez/publication/283506874_QEMU_una_alternativa_libre_para_la_emulacion_de_arquitecturas_de_hardware/links/5840717208ae2d21755f3550/QEMU-una-alternativa-libre-para-la-emulacion-de-arquitecturas-de-hardware.pdf.
- [4] informática. *Application Programming Interface*. Visitado el 2022-03-13. 2022. URL: <https://definicion.de/api/>.
- [5] Agustin Bassi. *ViHard, Plataforma de emulación de hardware para sistemas embebidos*. Visitado el 2023-06-28. 2018. URL: <https://github.com/agustinBassi-others/electron-virtual-hardware-platform/tree/develop>.
- [6] ACheng Zhao. *Electron*. Visitado el 2023-06-28. 2013. URL: <https://www.electronjs.org/>.
- [7] Arduino LLC. *Arduino*. Visitado el 2022-03-14. 2022. URL: <https://arduino.cl/>.
- [8] Stan Simmons. *UnoArduSim*. Visitado el 2022-03-15. 2022. URL: <https://sites.google.com/site/unoardusim/home>.
- [9] Queen's University. *Queen's University*. Visitado el 2022-03-15. 2022. URL: <https://www.queensu.ca/>.
- [10] Arduino. *Arduino Uno*. Visitado el 2022-03-14. 2022. URL: <https://arduino.cl/arduino-uno/>.
- [11] grupo de startups. *Virtronics*. Visitado el 2022-03-15. 2022. URL: <http://www.virtronics.com.au/simulator-for-arduino.html>.
- [12] Stan Simmons. *Tinkercad*. Visitado el 2022-03-15. 2022. URL: <https://www.tinkercad.com/dashboard>.
- [13] John Walker. *Autodesk*. Visitado el 2022-03-14. 2022. URL: <https://www.autodesk.com>.
- [14] Mbed Labs. *Mbed Simulator*. Visitado el 2022-03-15. 2022. URL: <https://os.mbed.com/blog/entry/introducing-mbed-simulator>.
- [15] ARM. *Arm Mbed Os*. Visitado el 2022-03-15. 2022. URL: <https://www.arm.com/products/development-tools/embedded-and-software/mbed-os>.
- [16] Wikipedia. *Navegador web*. Visitado el 2022-03-13. 2022. URL: https://es.wikipedia.org/wiki/Navegador_web.

- [17] LLVM Developer Group. *JavaScript*. Visitado el 2022-03-13. 2022. URL: <https://es.wikipedia.org/wiki/JavaScript>.
- [18] LLVM Developer Group. *NodeJS*. Visitado el 2022-03-13. 2022. URL: <https://xtermjs.org/>.
- [19] W3C. *Document Object Model*. Visitado el 2022-03-14. 2022. URL: <https://www.w3.org/TR/WD-DOM/introduction.html>.
- [20] Web Hypertext Application Technology Working Group. *Lenguaje de Marcas de Hipertexto*. Visitado el 2022-03-14. 2022. URL: <https://es.wikipedia.org/wiki/HTML>.
- [21] CSS Working Group. *Cascading Style Sheets*. Visitado el 2022-03-14. 2022. URL: <https://es.wikipedia.org/wiki/CSS>.
- [22] LLVM Developer Group. *SVG*. Visitado el 2022-03-13. 2022. URL: https://es.wikipedia.org/wiki/Gr%C3%A1ficos_vectoriales_escalables.
- [23] World Wide Web Consortium. *eXtensible Markup Language*. Visitado el 2022-03-14. 2022. URL: https://es.wikipedia.org/wiki/Extensible_Markup_Language.
- [24] LLVM Developer Group. *Xterm*. Visitado el 2022-03-13. 2022. URL: <https://xtermjs.org/>.
- [25] Microsoft. *TypeScript*. Visitado el 2022-03-14. 2022. URL: <https://www.typescriptlang.org/>.
- [26] W3C. *WebSocket*. Visitado el 2022-03-14. 2022. URL: <https://es.wikipedia.org/wiki/WebSocket>.
- [27] W3C HTML WG, WHATWG. *TypeScript*. Visitado el 2022-03-14. 2022. URL: <https://es.wikipedia.org/wiki/HTML5>.
- [28] Guillermo Rauch. *Socket.IO*. Visitado el 2022-03-14. 2022. URL: <https://socket.io/>.
- [29] Timesync Working Group. *Timesync*. Visitado el 2022-03-14. 2022. URL: <https://www.npmjs.com/package/timesync>.
- [30] Mocha Working Group. *Mocha*. Visitado el 2022-03-14. 2022. URL: <https://mochajs.org/>.
- [31] Chai Working Group. *Chai*. Visitado el 2022-03-14. 2022. URL: <https://www.chaijs.com/>.
- [32] Open Source License. *Emscripten*. Visitado el 2022-03-13. 2022. URL: <https://emscripten.org/>.
- [33] LLVM Developer Group. *Low Level Virtual Machine*. Visitado el 2022-03-13. 2022. URL: <https://es.wikipedia.org/wiki/LLVM>.
- [34] WebAssembly Working Group. *WebAssembly*. Visitado el 2022-03-14. 2022. URL: <https://webassembly.org/>.
- [35] Apache. *Binaryen*. Visitado el 2022-03-14. 2022. URL: <https://www.npmjs.com/package/binaryen>.
- [36] Open Source License. *Python*. Visitado el 2022-03-13. 2022. URL: <https://emscripten.org/>.
- [37] Express Working Group. *Express*. Visitado el 2022-03-14. 2022. URL: <https://expressjs.com/es/>.
- [38] Dennis Ritchie y Laboratorios Bell. *Lenguaje C*. Visitado el 2022-03-14. 2022. URL: [https://es.wikipedia.org/wiki/C_\(lenguaje_de_programaci%C3%B3n\)](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)).
- [39] Roman Zimbelmann. *Check*. Visitado el 2023-06-14. 2000. URL: <https://libcheck.github.io/check/>.
- [40] Andreas Schneider y Jan Kratochvil. *CMocka*. Visitado el 2022-03-14. 2009. URL: <https://cmocka.org/>.

- [41] Comunidad Proyecto CIAA. *sAPI proyecto CIAA*. Visitado el 2022-03-14. 2022. URL: <http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:firmware:v3>.
- [42] Comunidad Proyecto CIAA. *Firmware v2*. Visitado el 2023-07-29. 2018. URL: https://github.com/ciaa/firmware_v2.
- [43] Comunidad Proyecto CIAA. *Firmware v3*. Visitado el 2023-07-29. 2018. URL: https://github.com/ciaa/firmware_v3.
- [44] Eric Pernia. *sAPI Proyecto CIAA version 0.6.2*. Visitado el 2023-07-28. 2023. URL: https://github.com/epernia/firmware_v3/tree/master/libs/sapi/sapi_v0.6.2.
- [45] NXP Semiconductors. *LPCOPEN v2.16 Drivers, Middleware and Examples*. Disponible: 2016-06-25. URL: <http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/lpc-cortex-m-mcus/software-tools/lpcopen-libraries-and-examples:LPC-OPEN-LIBRARIES>.
- [46] Arm. *Mbed CLI*. Visitado el 2020-03-14. 2014. URL: <https://github.com/ARMmbed/mbed-cli>.
- [47] Eric Pernia. *SVG FirmwareV3*. Visitado el 2022-03-14. 2022. URL: https://github.com/epernia/board-simulator/blob/gh-pages/img/edu_ciaa_board.svg.
- [48] Microsoft. *Visual Studio Code*. Visitado el 2022-03-15. 2022. URL: <https://code.visualstudio.com/>.
- [49] Microsoft. *Integrated Development Environment*. Visitado el 2022-03-14. 2022. URL: https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado.
- [50] Comunidad Inkscape. *Inkscape*. Visitado el 2023-07-29. 2023. URL: <https://inkscape.org/>.
- [51] Microsoft. *GitHub*. Visitado el 2022-03-15. 2022. URL: <https://github.com/>.
- [52] Josh Kalderimis. *Travis CI*. Visitado el 2023-06-15. 2011. URL: <https://www.travis-ci.com/>.
- [53] Dmitriy Zaporozhets y Valery Sizov. *Gitlab*. Visitado el 2023-06-15. 2011. URL: <https://gitlab.com/>.
- [54] Ben Uretsky, Moisey Uretsky, Mitch Wainer, Jeff Carr y Alec Hartman. *DigitalOcean*. Visitado el 2023-06-15. 2011. URL: <https://www.digitalocean.com/>.
- [55] Jenny Chavez. *Emulador EDU-CIAA*. Visitado el 2023-06-24. 2023. URL: <https://github.com/jennifferch/ciaa-emulador>.
- [56] Postman Working Group. *Postman*. Visitado el 2022-03-24. 2022. URL: <https://www.postman.com>.
- [57] Richard Stallman y Roland McGrath. *GNU Make, una herramienta de gestión de dependencias*. Visitado el 2022-03-24. 2022. URL: <https://www.gnu.org/software/make/>.
- [58] cmake Working Group. *CTest*. Visitado el 2022-03-24. 2022. URL: <https://cmake.org/cmake/help/v3.0/manual/ctest.1.html>.
- [59] T. Teranishi. *Tera Term VT*. Visitado el 2022-03-24. 2022. URL: svn.osdn.jp/svnroot/ttssh2/trunk/.