



MAESTRÍA EN SISTEMAS EMBEBIDOS

MEMORIA DEL TRABAJO FINAL

Emulador de la placa EDU-CIAA-NXP

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 octubre de 2023.*

Resumen

La presente memoria describe el diseño e implementación de un emulador para la placa de desarrollo EDU-CIAA-NXP del Proyecto CIAA, concebido para la enseñanza de la programación de Sistemas Embebidos. Además de la EDU-CIAA-NXP, se emulan otras placas electrónicas y periféricos externos a conectar, permitiendo desarrollar programas en lenguaje C/C++, que se pueden compilar y ejecutar sobre el hardware virtual.

Este emulador se desarrolló como una plataforma web, disponible *on line*, que posibilita escribir, probar y verificar aplicaciones rápidamente de forma amigable y gratuita, sin la necesidad de contar con el hardware real.

Para la realización de este trabajo fueron fundamentales los conocimientos relacionados al análisis y diseño de software, implementación de *drivers* de dispositivos, planificación y sincronización de tareas, sistemas operativos de tiempo real, desarrollo de aplicaciones sobre Linux, y desarrollo de módulos del kernel. Además de *tests* de calidad del software, planificación y gestión de proyectos.

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. Objetivos	2
1.3. Alcance	2
1.4. Requerimientos	3
1.5. Metodología de trabajo	3
2. Introducción específica	5
2.1. Estado del arte	5
2.1.1. UnoArduSim	6
2.1.2. Virtronics	7
2.1.3. Tinkercad	7
2.1.4. <i>Arm Mbed OS Simulator</i>	8
2.2. Análisis de los emuladores revisados	8
2.3. Análisis de <i>Arm Mbed OS Simulator</i>	10
2.3.1. Tecnologías utilizadas	10
2.3.2. Funcionalidad	12
2.3.3. <i>Frontend</i> y <i>Backend</i>	13
2.3.4. Capas de software	16
2.3.5. Análisis estructural de archivos y carpetas	17
3. Desarrollo e implementación	21
3.1. Herramientas de desarrollo	21
3.2. Introducción al desarrollo	22
3.3. Cambios realizados sobre la estructura del código existente	22
3.3.1. Nuevas tecnologías agregadas	22
3.4. Frontend	24
3.4.1. Diseño de la Interfaz de Usuario	24
Área de ensamblado	25
Área de codificación	26
Área de consola integrada	27
3.4.2. JavaScript UI	29
3.4.3. Aplicación de Usuario	29
3.5. Backend	30
3.5.1. Biblioteca C	30
3.5.2. C HAL	33
3.5.3. JavaScript HAL	36
3.5.4. <i>sapi_gpio</i>	37
3.5.5. <i>sapi_tick</i>	39
3.5.6. <i>sapi_delay</i>	40
3.5.7. <i>freeRTOS</i>	42

3.5.8. <i>sapi_dht11</i>	44
3.5.9. <i>sapi_adc</i>	45
3.6. Caso de estudio	48
3.7. Despliegue	50
4. Ensayos y resultados	53
4.1. Banco de pruebas	53
4.2. Pruebas de Unidad	54
4.3. Pruebas de Integración	55
4.4. Pruebas de Interfaz	56
4.5. Integracion Continua	57
4.5.1. Prueba de acceso	59
4.6. Pruebas de funcionamiento	61
4.6.1. Prueba del ejemplo <i>rtc printf</i>	61
Ensayo en la placa EDU-CIAA-NXP	63
4.6.2. Prueba del ejemplo <i>dht11</i>	64
Ensayo en la plataforma de emulación web	64
Ensayo en la placa EDU-CIAA-NXP	66
4.6.3. Prueba de un nuevo proyecto <i>tick hook</i>	69
Ensayo en la plataforma de emulación web	69
Ensayo en la placa EDU-CIAA-NXP	71
4.6.4. Periféricos implementados en <i>Mbed Simulator</i> y en el Emulador EDU-CIAA-NXP	71
5. Conclusiones	77
5.1. Objetivos alcanzados	77
5.2. Próximos pasos	78
Bibliografía	79

Índice de figuras

1.1. Esquema Emulador EDU-CIAA-NXP.	2
2.1. Esquema de la plataforma ViHard. ¹	5
2.2. Plataforma UnoArduSim.	6
2.3. Plataforma Virtronics.	7
2.4. Plataforma Tinkercad.	7
2.5. Plataforma Arm Mbed OS Simulator.	8
2.6. Esquema modelo cliente/servidor.	11
2.7. Funcionamiento de <i>Arm Mbed OS Simulator</i>	12
2.8. Esquema de las tecnologías utilizadas en <i>Arm Mbed OS Simulator</i>	13
2.9. Arquitectura de capas de la plataforma <i>Arm Mbed OS Simulator</i>	16
2.10. Ejemplo de flujo de información de pin de salida controlando un led en <i>Arm Mbed OS Simulator</i>	17
2.11. Estructura de carpetas y archivos de <i>Arm Mbed OS Simulator</i>	17
3.1. Estructura de carpetas y archivos de <i>EmuCIAA</i>	23
3.2. Plataforma de emulación para la placa EDU-CIAA-NXP.	25
3.3. El usuario puede elegir un componente en la aplicación.	26
3.4. Periférico agregado en el área de ensamblado.	27
3.5. Código que generó los errores de compilación.	28
3.6. Errores de compilación.	29
3.7. Estructura jerárquica de ejemplos.	30
3.8. Carga automática del periférico.	31
3.9. Salida de la terminal serie.	31
3.10. Programa de usuario que imprime por consola.	32
3.11. Diagrama de bloques de los oyentes de <i>EventEmitter</i> en la capa UI.	32
3.12. Diagrama de dependencias del módulo <i>GPIO</i> de la biblioteca <i>sAPI</i> del proyecto CIAA.	33
3.13. Diagrama de dependencias del módulo <i>GPIO</i> de la plataforma de emulación para la placa EDU-CIAA-NXP.	34
3.14. Diagrama de funcionamiento de <i>Emscripten</i>	35
3.15. Modelo de <i>publicación/suscripción</i>	36
3.16. Diagrama de bloques de <i>EventEmitter</i> implementado en la plataforma.	37
3.17. Diagrama de bloques de la macro <i>EM_ASM_</i>	38
3.18. Diagrama de bloques del envío de eventos de la capa <i>Javascript HAL</i>	39
3.19. Diagrama de bloques <i>EMSCRIPTEN_KEEPALIVE</i>	41
3.20. Diagrama de bloques de la función <i>ccall</i>	42
3.21. Diagrama de bloques <i>emsdk_sleep</i>	43
3.22. Diagrama de bloques de la macro <i>EM_ASM_INT</i>	45
3.23. Diagrama de bloques de la capa <i>JavaScript UI</i> con las dos opciones para el usuario.	46
3.24. Diagrama de bloques de <i>C HAL</i> y <i>JavaScript HAL</i> para el módulo <i>adc</i>	47

3.25. Diagrama de bloques con la interacción de las capas <i>JavaScript HAL</i> y <i>JavaScript UI</i>	48
3.26. Interacción del usuario con las dependencias del emulador.	49
3.27. Activación de evento con el nombre <code>gpio_write</code>	50
3.28. GPIO oyente del evento con el nombre <code>gpio_write</code>	50
3.29. Interacción entre todas las capas de programación.	51
4.1. Primera parte de la salida por consola de las pruebas unitarias con <i>CMocka</i> o <i>Check</i>	55
4.2. Segunda parte de la salida por consola de las pruebas unitarias con <i>CMocka</i> o <i>Check</i>	55
4.3. Depuración de las pruebas de integración.	56
4.4. Salida por consola durante la depuración de las pruebas de interfaz con <i>Mocha</i>	57
4.5. Información de las pruebas que se ejecutaron en Travis CI.	58
4.6. Información de las pruebas que se ejecutaron en GitLab CI.	59
4.7. Prueba plataforma emulador ejecutando el ejemplo <i>Blinky</i>	60
4.8. Respuesta del servidor.	61
4.9. Código del ejemplo <i>rtc printf</i>	62
4.10. Salida por consola del ensayo <i>rtc printf</i>	62
4.11. Ejemplo <i>rtc printf</i> importado en <i>eclipse</i>	63
4.12. Salida de la terminal COM7 -Tera Term VT.	63
4.13. Resultado del CP02 con la opción "Establecer Temperatura y Humedad manualmente(haga click y arrastre sobre los indicadores)." .	65
4.14. Resultado del CP02 con la opción "Obtener datos de servidor clímatico local."	66
4.15. Petición de datos de temperatura/humedad.	66
4.16. Ensayo en la plataforma EDU-CIAA-NXP del ejemplo <i>dht11</i>	67
4.17. Código del ejemplo en <i>eclipse</i>	67
4.18. Cambios en la placa EDU-CIAA-NXP durante el ensayo.	68
4.19. Salida de la terminal COM7 -Tera Term VT.	68
4.20. Código del ensayo del nuevo proyecto tick hook.	70
4.21. Salida por consola del ensayo tick hook.	70
4.22. Salida de la terminal COM7 -Tera Term VT.	71
4.23. Funcionamiento GPIO en <i>Mbed Simulator</i>	72
4.24. Funcionamiento PWM en <i>Mbed Simulator</i>	73
4.25. Periféricos externos de <i>Mbed Simulator</i>	74
4.26. Periféricos externos del emulador web EDU-CIAA.	75

Índice de tablas

2.1. Comparación de características de los emuladores revisados	9
3.1. Módulo <i>GPIO</i>	31
3.2. <i>sapi_peripheral_map.h</i>	35
3.3. Funciones <i>sapi_gpio</i>	37
3.4. Funciones <i>sapi_tick</i>	39
3.5. Funciones <i>sapi_delay</i>	42
3.6. Conceptos importantes de <i>freeRTOS</i> que se cumplen en el emulador.	44
3.7. Funciones <i>sapi_dht11</i>	44
3.8. Funciones <i>sapi_adc</i>	45
4.1. Recursos de hardware utilizados	53
4.2. Recursos de software utilizados	54
4.3. Comparación de características de periféricos internos implementados en <i>Mbed Simulator</i>	72
4.4. Comparación de características de Periféricos externos implementados en <i>Mbed Simulator</i>	73
4.5. Comparación de características de periféricos internos del Emulador EDU-CIAA	74
4.6. Comparación de características de periféricos externos del Emulador EDU-CIAA	75

Dedicado a mi familia

Capítulo 1

Introducción general

En este capítulo se exponen las problemáticas encontradas con el hardware en el estudio de los Sistemas Embebidos, que motiva la realización de un emulador para la placa EDU-CIAA-NXP [1]. Presentando los objetivos, el alcance, los requerimientos y la metodología de trabajo utilizada para llevar a cabo este proyecto.

1.1. Motivación

Los emuladores son desarrollos de software que modelan el funcionamiento del hardware real, de manera que permiten ejecutar programas dentro de un ambiente que imita su comportamiento [2]. De esta forma un usuario puede simular su código antes de instalarlo en el dispositivo físico.

Para colaborar con la enseñanza de la programación de Sistemas Embebidos, se propuso, como parte del Proyecto CIAA [3], realizar una plataforma de software que emule el funcionamiento de la placa EDU-CIAA-NXP, así como los periféricos y placas que se pueden conectar a ella.

Esto se debe a que para el desarrollo de aplicaciones en sistemas embebidos es necesario tener de antemano la placa de desarrollo y los componentes de hardware externos a conectar a la placa, para poder probar los programas realizados. Además, muchas veces un usuario sin experiencia, no cuenta con los dispositivos para comenzar, se equivoca en la selección de los mismos, o en las conexiones eléctricas, provocando daños irreparables en el hardware.

De esta manera, mediante el uso de un emulador se evita todos estos inconvenientes y permite al usuario probar sus programas rápidamente, enfocándose en el desarrollo de aplicaciones y ejecutando sus pruebas en un sistema virtual, dejando para más adelante la implementación en el hardware real.

Un aspecto importante a destacar es el alto costo del hardware y la variedad de dispositivos necesarios. Entonces, contar con un emulador habilita a personas con bajos recursos aprender a programar Sistemas Embebidos utilizando hardware virtual, que se comporta como el hardware real, permitiendo probar diferentes tecnologías.

Para disponibilizar esta herramienta lo más posible, se decidió construir una plataforma de desarrollo con interfaz gráfica dentro de un entorno web, accesible a través de un navegador. De esta manera, se obtiene una herramienta que esté disponible de forma *on line*, para que pueda usarse de forma gratuita, tanto en Computadoras, Tablets o Smartphones, como puede observarse en la figura 1.1.

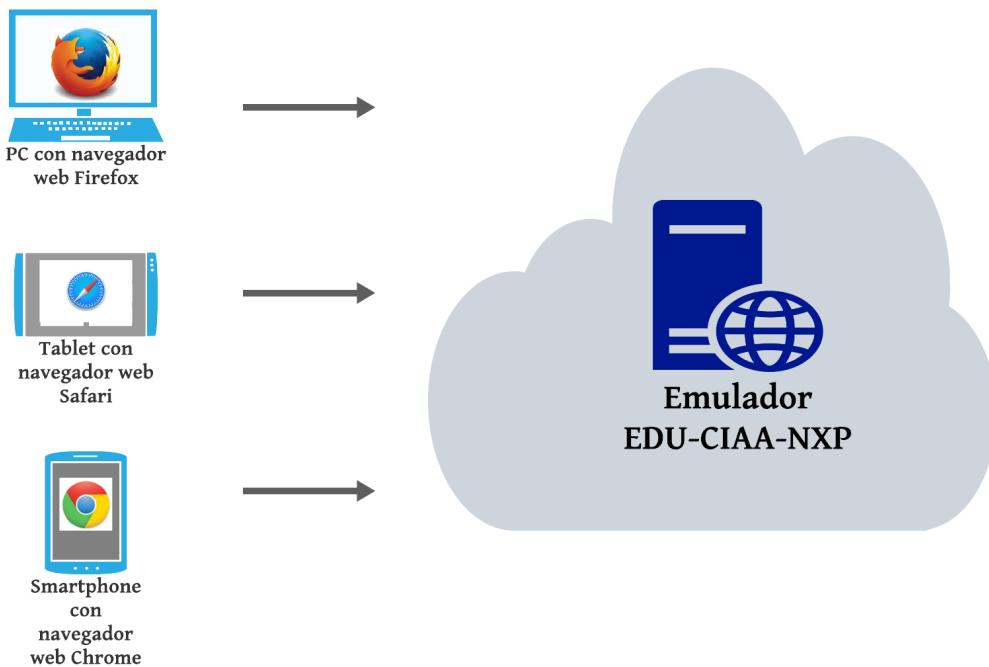


FIGURA 1.1. Esquema Emulador EDU-CIAA-NXP.

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 novatos de hacer la interacción con el hardware.

La autora del presente trabajo ha colaborado previamente en el Proyecto CIAA, mediante el desarrollo del software CIAA-BOT Debug como Trabajo Final de la Carrera en Especialización en Sistemas Embebidos de la FI-UBA [4], que permite depurar programas realizados mediante el lenguaje gráfico CIAA-BOT.

1.2. Objetivos

Dados los antecedentes explicados, el objetivo de este trabajo es desarrollar una herramienta educativa que brinda un entorno virtual para la placa EDU-CIAA-NXP, permitiendo al usuario seleccionar y conectar con la EDU-CIAA-NXP diferentes tipos de dispositivos virtuales e interactuar con los mismos. 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 comunes, todo de manera virtual, sin necesidad de disponer de ningún dispositivo de hardware.

1.3. Alcance

En el presente trabajo se realizó una primera versión de la herramienta para ser usada con la placa EDU-CIAA-NXP. En particular, se incluyen los siguientes aspectos:

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

El presente proyecto no incluye el desarrollo de la aplicación para emular otras placas que no sea EDU-CIAA-NXP.

1.4. Requerimientos

Se han identificado los siguientes requerimientos:

1. Investigación y definición de la arquitectura del software.
 - a) Investigar sobre las plataformas de emulación de hardware existentes.
 - b) Investigar la arquitectura y funcionamiento de las bibliotecas y ejemplos para la EDU-CIAA-NXP, disponibles en firmware v3 [5].
2. Desarrollo del Emulador.
 - a) Realizar la aplicación para emular el hardware.
 - b) Integrar las bibliotecas de lenguaje C de la placa EDU-CIAA-NXP portándolas a la paltaforma virtual.
 - c) Respetar el estilo de código de las bibliotecas.
 - d) Portar ejemplos de utilización de las bibliotecas.
 - e) Desarrollar nuevos ejemplos.
3. Documentación del proyecto.
 - a) Elaborar la documentación de la plataforma.
 - b) Confeccionar un manual de usuario.

1.5. Metodología de trabajo

Para el desarrollo de este trabajo se eligió utilizar las siguientes prácticas:

- Utilizar software libre para el desarrollo del proyecto. Reutilizar todo el software y ejemplos disponibles, tanto del Proyecto CIAA como de terceros.
- Utilizar un Sistema de Control de Versiones de código fuente distribuído [6].
- Desarrollar *tests* unitarios y de integración.
- Automatizar los *tests* y *deployments* mediante Integración Continua [7].

Capítulo 2

Introducción específica

En el presente capítulo se describe el estado actual de algunas soluciones implementadas, y se elige una plataforma existente, que cumple con los requerimientos del trabajo como base. También, se presenta el estudio del código fuente de *Arm Mbed OS Simulator*.

2.1. 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 [8] que emula dispositivos de hardware en la PC, pero con la placa ECU-CIAA-NXP real conectada a la PC a través del puerto USB. En la figura 2.1 se muestra el esquema de la plataforma ViHard.

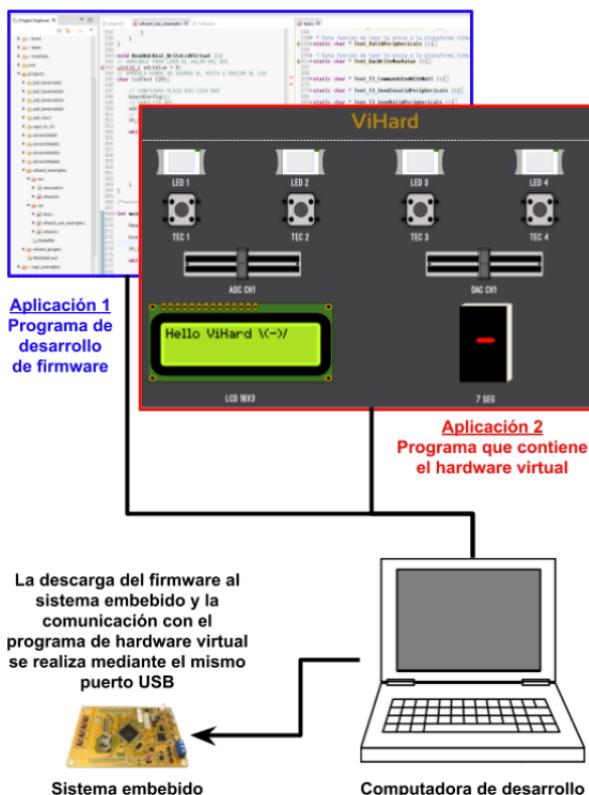


FIGURA 2.1. Esquema de la plataforma ViHard.¹

La plataforma se compone de un programa de PC con los periféricos virtuales de hardware y una biblioteca de firmware para la EDU-CIAA-NXP, que controla y gestiona el funcionamiento del hardware virtual. Ambos programas se comunican con el sistema embebido mediante UART, a través de un puerto USB.

El programa de hardware virtual es una aplicación de escritorio multiplataforma desarrollada utilizando el framework Electron [9]. 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 programa, que deberá compilar y descargar a la EDU-CIAA-NXP, y posteriormente realizar las pruebas correspondientes.

Por otro lado, se encontraron muchas plataformas educativas que simulan microcontroladores, sobre todo para la placa Arduino [10]. Para el análisis, se seleccionaron algunos de los simuladores más populares que implementan funcionalidades relevantes para el presente trabajo, los cuales se describen en las siguientes secciones.

2.1.1. UnoArduSim

UnoArduSim [11] fue desarrollado en la Universidad de Queen [12] por el profesor Stan Simmons. En la figura 2.2 puede observarse la interfaz del programa.

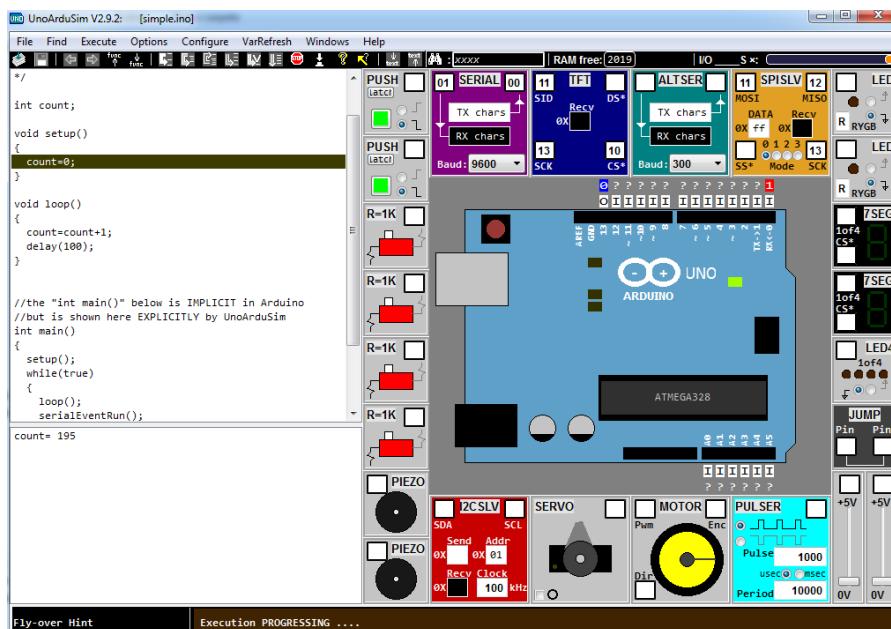


FIGURA 2.2. Plataforma UnoArduSim.

La herramienta simula en la pantalla de la PC la placa Arduino Uno [13] y muchos de los dispositivos de entrada y salida más usados, asimismo, permite la depuración interactiva de funciones o programas completos. 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.

2.1.2. Virtronics

Virtronics [14] es uno de los simuladores más completos que hay hoy en día para Arduino [10], 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 2.3 se muestra la plataforma.

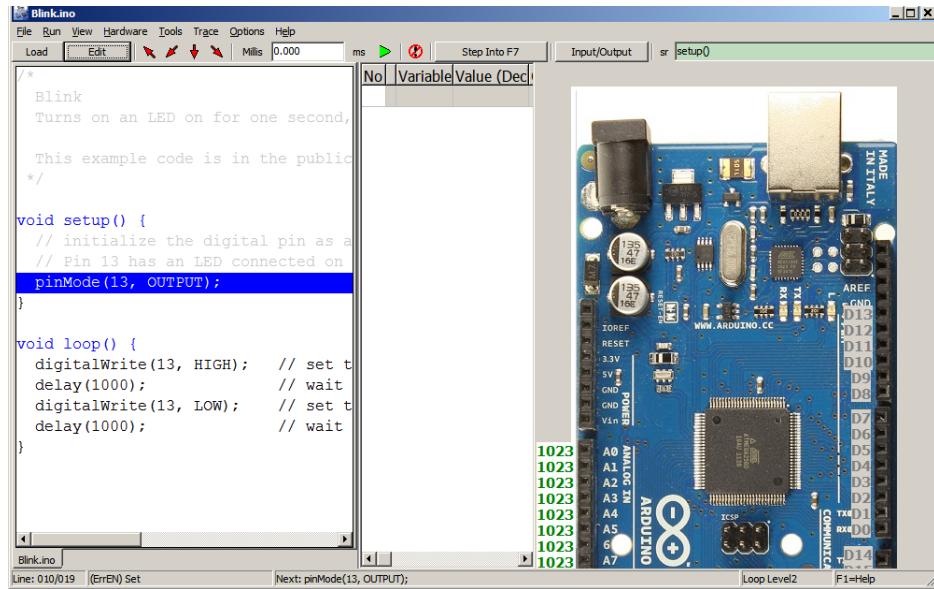


FIGURA 2.3. Plataforma Virtronics.

2.1.3. Tinkercad

Tinkercad [15] es una plataforma online que permite el acceso desde cualquier navegador web, cuya interfaz de usuario se muestra en la figura 2.4.

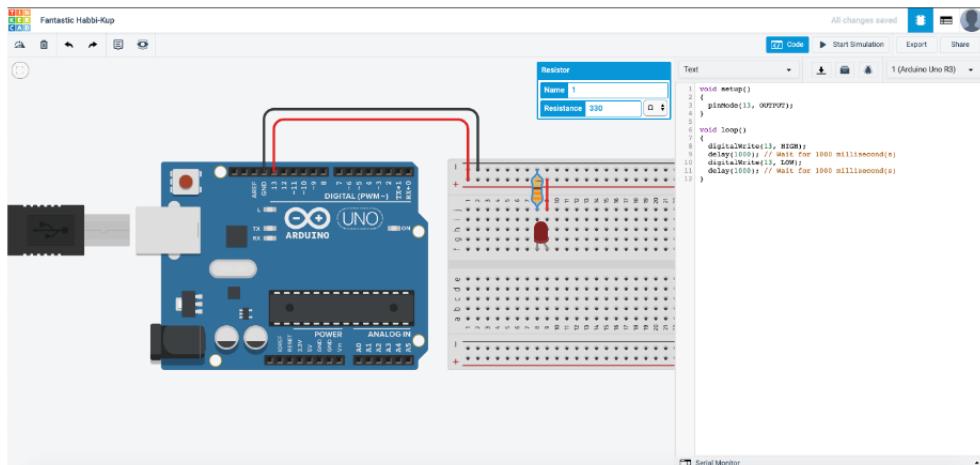


FIGURA 2.4. Plataforma Tinkercad.

Fue desarrollado por ingenieros y diseñadores de software de Autodesk [16] y permite diseños 3D. Adicionalmente, es necesario crear una cuenta antes de empezar a usar la plataforma, por consiguiente, todos los diseños se guardan en la cuenta creada.

2.1.4. Arm Mbed OS Simulator

Arm Mbed OS Simulator [17] fue desarrollado por ingenieros de Arm, encargados de mantener las bibliotecas Mbed OS, [18] y es parte de Mbed Labs. La plataforma era accesible *on line* al momento del comienzo de este proyecto, pero actualmente debe ser descargado desde su repositorio [19] y luego, puede ejecutarse utilizando cualquier navegador web en la red local, o bien, realizar el despliegue en un servidor para que esté disponible *on line*. En la figura 2.5 puede observarse la plataforma online de Mbed Simulator.

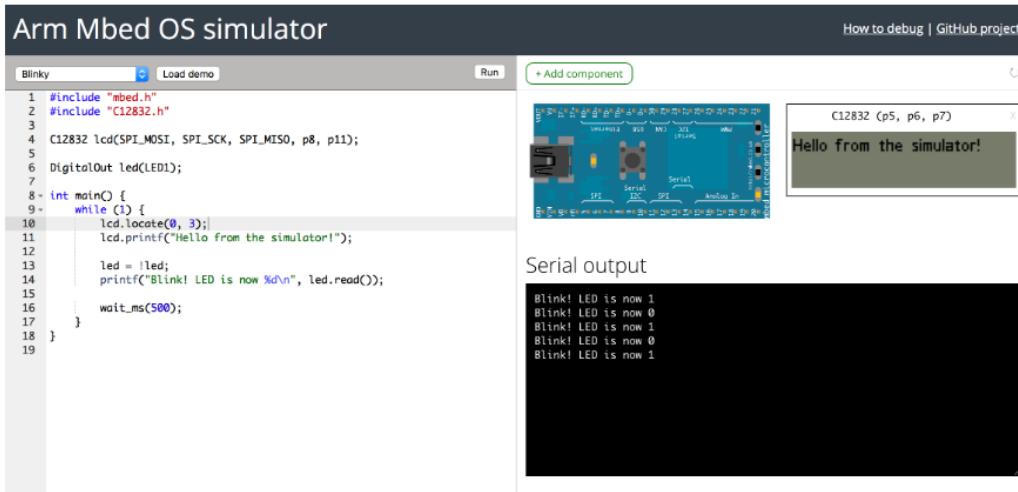


FIGURA 2.5. Plataforma Arm Mbed OS Simulator.

El funcionamiento de este emulador es muy simple. Se puede crear un programa desde cero, o bien, se puede elegir un ejemplo de la lista desplegable y cargar el proyecto desde el botón "Load demo", luego añadir los componentes necesarios para el programa, como por ejemplo, el display que se observa en la figura 2.5, donde se pide al usuario indicar a qué pines estará conectado. Es importante mencionar que el programa ya incluye la placa con el microcontrolador precargada. Una vez completo el programa y el ensamblado del hardware, se puede compilar y ejecutar en el emulador usando el botón "Run".

2.2. Análisis de los emuladores revisados

En la tabla 2.1 se comparan las características más importantes de estos emuladores.

TABLA 2.1. Comparación de características de los emuladores revisados

Característica	UnoArduSim	Virtronics	Tinkercad	Mbed OS
Placa/Plataforma emulada	Arduino Uno y Mega	Varios modelos Arduino	Arduino Uno	Arm Mbed OS
Gratuito	Sí	No	Sí	Sí
Aplicación	Escritorio	Escritorio	Web	Web
Plataforma	Windows	Windows/Linux	Todas	Todas
Código abierto	No	No	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
Ejemplos	Sí	Sí	Sí	Sí

Cabe destacar, que de las plataformas revisadas *Arm Mbed OS Simulator* es la única de código abierto. Sin embargo, este análisis sirve también para revisar, comprender, comparar las ventajas y desventajas de las otras plataformas, y tomar características útiles para el desarrollo del emulador.

Del análisis anterior, se desprende que *Arm Mbed OS Simulator* presenta las siguientes ventajas significativas:

- Código abierto: al tener acceso al código fuente permitió estudiar cómo funciona internamente el proyecto simulador, además, de la libertad de uso y distribución.
- Arquitectura de aplicación web.
- Su capacidad para simular dispositivos y componentes.
- 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.
- 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 Simulator recibe actualizaciones continuas con las últimas tecnologías que permite mantener el emulador para la placa EDU-CIAA-NXP actualizado.

Sim embargo, actualmente, *Arm Mbed OS Simulator* presenta las siguientes limitaciones:

1. 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.

2. En cada iteración, la ejecución del programa puede variar en términos de tiempo, lo que puede afectar a la precisión en la sincronización de eventos dentro de la aplicación.
3. En *Arm Mbed OS Simulator*, no hay restricciones significativas en cuanto a la cantidad de memoria que se puede asignar al stack o al heap de un programa, lo cual difiere del hardware físico, donde sí existen limitaciones de memoria.
4. Dentro del entorno web, las interrupciones no se manejan de la misma manera que en un sistema embutido real, debido a que no implementa el manejo de prioridades, por lo cual no afectan la ejecución del programa principal.
5. Sin RTOS. No tiene la capacidad de ejecutar múltiples hilos de manera concurrente como lo haría un RTOS. Todo el código se ejecuta en un solo hilo. Se puede utilizar la biblioteca mbed-events para manejar ciertos aspectos de concurrencia, usando eventos y temporizadores.
6. Emulación a nivel API de Mbed OS. No permite programar a bajo nivel, utilizando registros de core para la arquitectura ARM o periféricos.
7. Sin capacidad de debug paso a paso. Se realiza el programa, se compila y ejecuta en el hardware virtual pero no permite la depuración de código.

Dadas todas estas consideraciones y una vez confirmada su compatibilidad para los propósitos del presente trabajo, se decide basar el emulador de la EDU-CIAA-NXP en portar el proyecto *Arm Mbed OS Simulator*, para la EDU-CIAA-NXP y sus bibliotecas.

2.3. Análisis de *Arm Mbed OS Simulator*

Se procedió a analizar en detalle el código fuente de *Arm Mbed OS Simulator* para adquirir una compresión de su arquitectura, del funcionamiento del *Sistema Operativo Mbed*, de los periféricos simulados y sus interacciones, así como de las configuraciones necesarias para la plataforma web.

2.3.1. Tecnologías utilizadas

Arm Mbed OS Simulator es una aplicación web. Este tipo de aplicaciones 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 [20]. Las aplicaciones web se basan en una arquitectura cliente/servidor (figura 2.6), en donde un cliente o navegador web realiza peticiones al servidor y en consecuencia el servidor envía la respuesta de regreso.

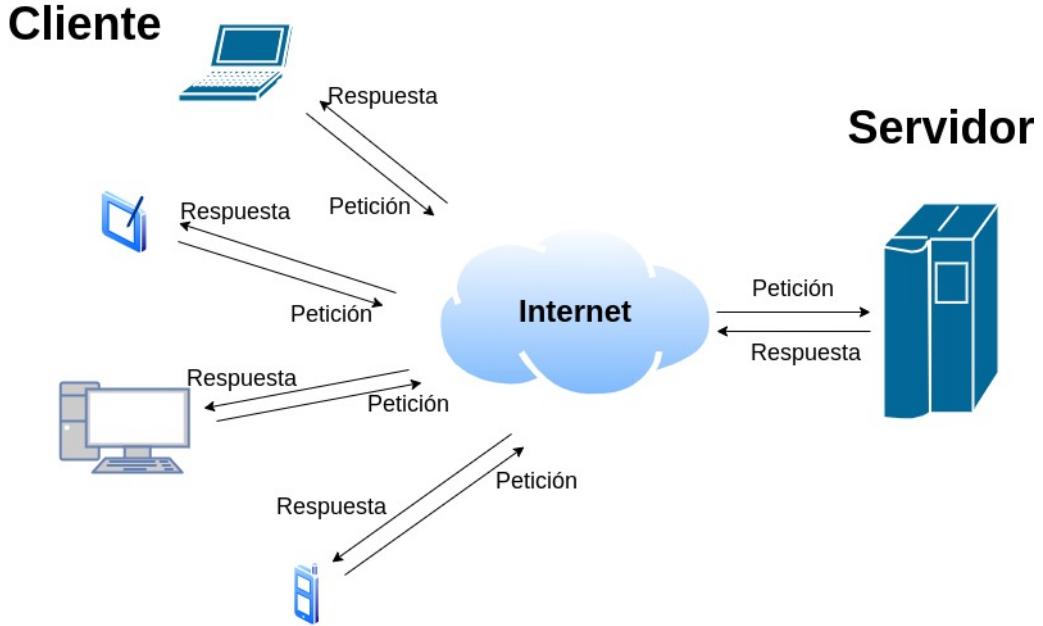


FIGURA 2.6. Esquema modelo cliente/servidor.

Las ventajas más importantes que presenta la tecnología web son:

- No es necesario instalar nada en la computadora del usuario.
- Muy bajo consumo de recursos del lado del cliente, la mayor carga se encuentra del lado del servidor.
- Es posible acceder al emulador desde cualquier ubicación con una conexión a internet.
- El usuario no requiere tener un sistema operativo específico, ya que se puede ejecutar en todos los dispositivos con acceso a un navegador web y una conexión a internet.

Arm Mbed OS Simulator está desarrollado principalmente sobre las siguientes tecnologías:

- *NodeJS* [21]: es un entorno de ejecución del lenguaje de programación *JavaScript* [22] cuyo propósito es el desarrollo de aplicaciones web y servicios del lado del servidor. También, proporciona una arquitectura orientada a eventos y no bloqueante.
- *Emscripten* [23]: es un *toolchain* de compilación completo para *WebAssembly*, que utiliza *LLVM* [24] y *Binaryen* [25], para compilar C y C++ en *WebAssembly* [26] y *JavaScript*. La salida de *Emscripten* puede ejecutarse en la web y en *NodeJS*. Esto permite ejecutar aplicaciones desarrolladas en C y C++ en la web, sin necesidad de plugins o complementos adicionales.
- *Arm Mbed OS* [27]: es un sistema operativo para sistemas embebidos, de código abierto, diseñado específicamente para los dispositivos *IoT*. Incluye todas las funciones que necesita para desarrollar un producto conectado, basado en un microcontrolador *Arm Cortex-M*, incluida seguridad, conectividad, *RTOS* y controladores para sensores y dispositivos de Entrada/Salida.

- *Arm Mbed CLI* [28]: es una herramienta de línea de comandos que facilita el desarrollo y la gestión de proyectos basados en la plataforma *Arm Mbed OS*, permite realizar tareas como la configuración del entorno de desarrollo, compilación de código, gestión de dependencias y la depuración, además, de identificar y resolver problemas en el código.

2.3.2. Funcionalidad

Estas tecnologías presentadas en la sección 2.3.1 se relacionan entre sí como se observa en el diagrama de secuencia de la figura 2.7.

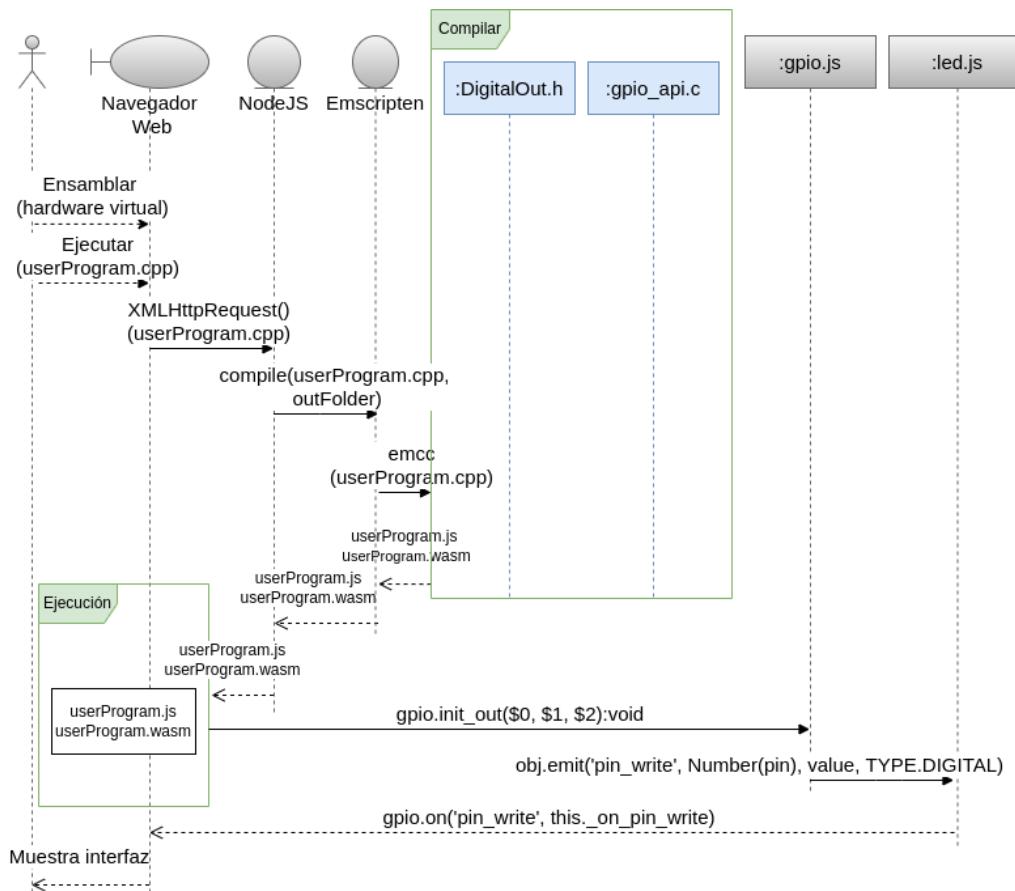


FIGURA 2.7. Funcionamiento de *Arm Mbed OS Simulator*.

Node.js ejecuta *Arm Mbed OS Simulator* dentro un servidor web. Un usuario puede acceder a la aplicación web mediante un navegador para utilizarla.

Cuando el usuario finaliza el programa en C/C++ y el ensamblado del hardware, puede proceder a presionar el botón *Run* para compilar y ejecutar el programa. En este proceso, se integra el programa de usuario con la biblioteca *Arm Mbed OS*. Posteriormente, se compila todo en conjunto mediante las herramientas *Emscripten* y *Arm Mbed CLI*.

El resultado de dicha compilación, el es mismo programa en *WebAssembly* y *javascript*, que luego se ejecuta sobre el hardware virtualizado, permitiéndole al usuario probar su funcionamiento.

2.3.3. Frontend y Backend

En el contexto del desarrollo de aplicaciones web, se denomina *Frontend* a la interfaz que los usuarios interactúan directamente, siendo la cara visible del sitio en el lado del cliente. Por otro lado, el *Backend* es la parte lógica que se encarga de la conexión con el servidor, de tomar los datos, procesarlos y devolverlos al *Frontend*.

En la figura 2.8 se muestra un esquema con las tecnologías utilizados tanto en el *Frontend* como en el *Backend* de *Arm Mbed OS Simulator*.

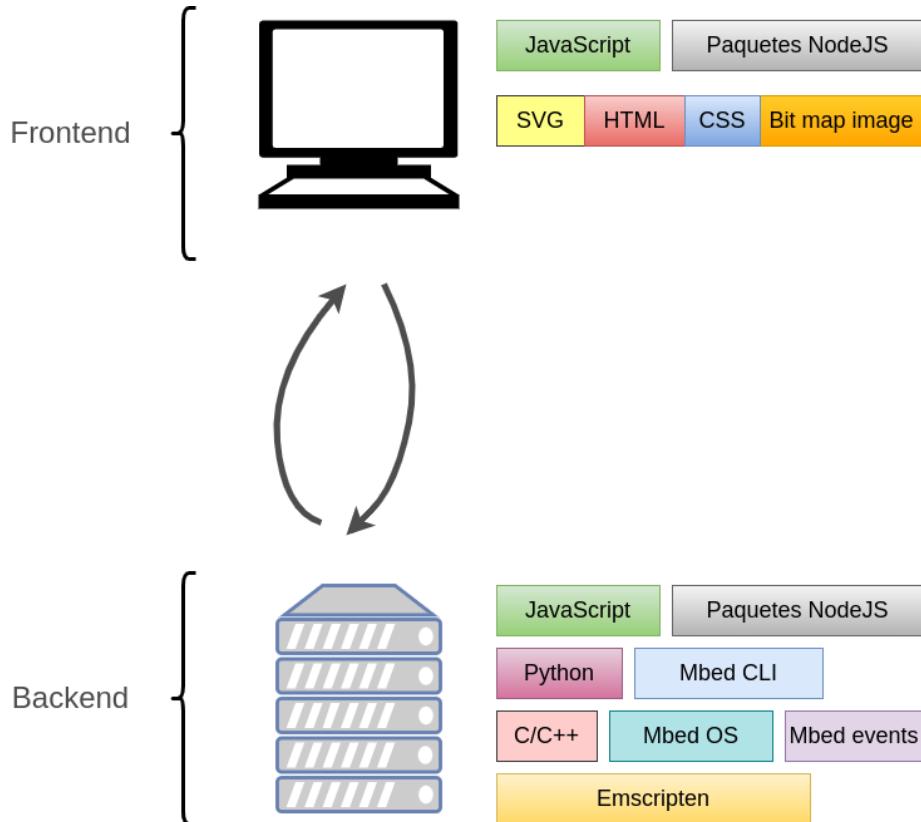


FIGURA 2.8. Esquema de las tecnologías utilizadas en *Arm Mbed OS Simulator*.

Como se puede observar, existen tecnologías que se utilizan en ambas partes de la aplicación y otras correspondientes únicamente al *Frontend*, o al *Backend*.

El lenguaje de programación *JavaScript* del lado del cliente, permite crear páginas web dinámicas y también responder a eventos causados por el propio usuario tales como modificaciones del DOM (siglas en inglés de *Document Object Model* [29]). Por consiguiente, el *frontend* permite cargar y ejecutar los archivos resultantes generados por el compilador en el navegador. Asimismo, es el responsable de configurar el entorno necesario para ejecutar la aplicación web y proporcionar la interfaz de usuario para la interacción.

En el *backend*, *JavaScript* se utiliza para gestionar la comunicación y la interacción entre los diferentes componentes a través de solicitudes HTTP, permitiendo la transferencia de datos y el flujo de información dentro de la plataforma web.

Los paquetes de *Node.JS* consisten en uno o más archivos *.js* o *.ts* (módulos) agrupados (o empaquetados) juntos. Los archivos en un paquete son código reutilizable que realiza una función específica en la aplicación *Node.js*. *Arm Mbed OS Simulator* utiliza los siguientes paquetes:

- En *Frontend* y *Backend*:

- *socket.io*
- *timesync*

- Solo en *Frontend*

- *xterm*

- Solo en *Backend*:

- *body-parser*
- *express*
- *command-exists*
- *commander*
- *compression*
- *es6-promisify*
- *getmac*
- *hbs*
- *opn*
- *puppeteer*

Se describen los paquetes más relevantes:

- *Socket.IO* [30]: es un paquete que utiliza *websockets* para comunicación bidireccional con baja latencia. La plataforma, utiliza *Socket.IO* para establecer la comunicación entre el *Frontend* y el *Backend*, y de esta manera, permitir la transferencia de datos de manera dinámica con el fin de actualizar los eventos entre ambas partes de la plataforma.
- *Xterm* [31]: es un paquete escrito en *TypeScript* [32] para el que permite que una aplicación pueda usar terminales emuladas con todas sus funciones en el navegador web (*Frontend*). La plataforma web utiliza esta tecnología en la interfaz de usuario para visualizar la salida de la UART de la placa simulada en una terminal serie.
- *Express* [33]: es un marco de aplicaciones web en el *Backend* para *Node.JS* diseñado para crear aplicaciones web y APIs (siglas en inglés de *application programming interface*) [34]. Se utiliza *Express* para configurar *middlewares* que permiten servir archivos estáticos desde carpetas específicas, como "*outUser*", que contiene los archivos generados por *Emscripten*.

Tecnologías que se utilizan solamente en el *frontend* de *Arm Mbed OS Simulator*:

- *HTML* [35]: son las siglas en inglés de *HyperText Markup Language*, o en español, Lenguaje de Marcas de Hipertexto. Es un lenguaje de marcado que permite la estructuración de información y contenido en un documento o

sitio web. El marcado se ejecuta a través de etiquetas que cumplen diferentes funciones en la estructuración del documento para visualizarlos en el navegador web. Este lenguaje es sencillo de aprender y es fácil de tanto por humanos como por máquinas. Se utiliza para definir el contenido de la interfaz de usuario, como los botones, lista desplegable, pantallas de visualización, y otros componentes necesarios para interactuar con la plataforma web.

- SVG [36]: de las siglas en inglés *Scalable Vector Graphics*, es un estándar web para definir imágenes vectoriales bidimensionales. Las imágenes creadas con este formato se pueden escalar y hacer zoom de forma arbitraria sin pérdida de resolución debido a que, en lugar de estar formados por píxeles, como otros formatos de imagen, están descritas en base a objetos tales como líneas, círculos y polígonos, entre otros. Este tipo de gráficos se describe mediante un archivo de texto cuya estructura está basada en el lenguaje de marcado extensible XML [37], siendo un formato muy útil para ser utilizado en entornos web. Este tipo de gráficos se utiliza en la plataforma para representar gráficamente la placa de desarrollo donde se ejecutan los programas, y permitir interactuar con su botón y LED.
- CSS [38]: de las siglas 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 pantalla en una página HTML o imagen SVG. Es la mejor forma de separar los contenidos de su diseño, y es necesario para crear páginas web complejas. De esta manera, CSS controla la presentación visual y el estilo del contenido HTML y SVG de esta plataforma web.
- Imágenes: Se utilizan imágenes .png de los LEDs virtuales a conectar con la placa virtual.

Finalmente, se describen las tecnologías se utilizan solamente en el *Backend* de *Arm Mbed OS Simulator*:

- *Python* [39]: es un poderoso y popular lenguaje de programación multiplataforma, de código abierto, y un entorno de ejecución. Se caracteriza por su sencillez y su gran potencia para el tratamiento de datos en el lado del servidor. En la plataforma web, *Python* 2.7 se utiliza para ejecutar scripts que realizan tareas específicas, como la configuración e inicialización y la ejecución de *Arm Mbed CLI*.
- Lenguajes C / C++ [40]: son lenguajes de propósito general y muy populares debido al eficiente código que produce al crear software de sistemas y de aplicaciones. Las bibliotecas *Arm MBed OS* y los programas que el usuario desarrolla para la placa virtual, están escritos en estos lenguajes.
- *Mbed events* [41]: es una biblioteca de código escrita en lenguaje C que se utiliza en el desarrollo de software embebido para facilitar la gestión de eventos y temporizadores. Para el desarrollo de la plataforma usa esta biblioteca para la creación y gestión de tareas en el *RTOS*. Aunque su uso no permite obtener las funcionalidades requeridas.

2.3.4. Capas de software

En el diagrama de bloques de la figura 2.9 se muestra la arquitectura básica de una aplicación de usuario que ejecuta en *Arm Mbed OS Simulator* y las capas de software involucradas.

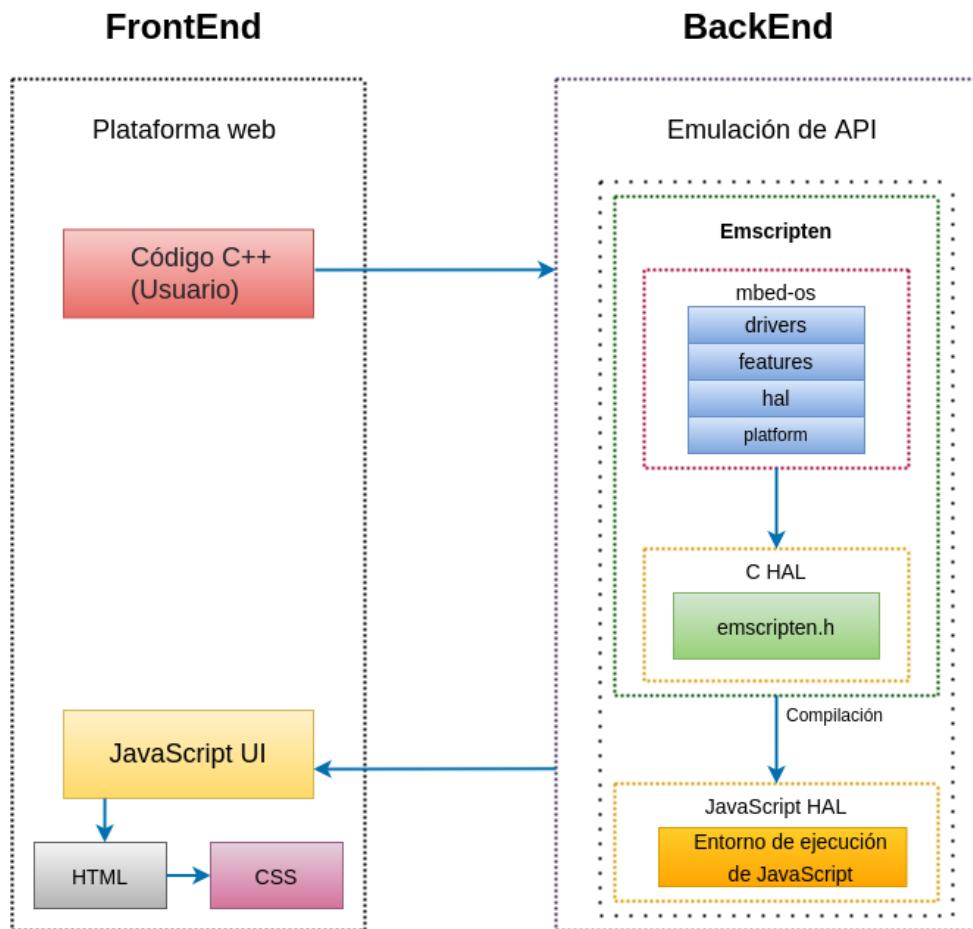


FIGURA 2.9. Arquitectura de capas de la plataforma *Arm Mbed OS Simulator*.

Arm Mbed OS Simulator presenta una emulación a nivel de API de la biblioteca *Mbed OS* debido a que gran parte de las API son genéricas entre diferentes plataformas de hardware objetivo soportadas (*targets*). Esto incluye las capas GPIO, stacks de redes IP4/IPV6 y bibliotecas de comunicación como *CoAP* o *LoRaWAN*.

El código específico del *target*, como, por ejemplo, en qué registros escribir para cambiar el valor de pin GPIO, se implementa utilizando la capa *Mbed C HAL*.

De esta manera, *Arm Mbed OS Simulator* utiliza el mismo enfoque, implementando un nuevo *target* (*TARGET_SIMULATOR*) que implementa la capa *Mbed C HAL*, que pasa eventos a una *HAL* de *JavaScript* (capa *JS HAL*). Luego, la interfaz de usuario se suscribe a estos eventos y actualiza el simulador en consecuencia (capa *JS UI*).

Se toma, por ejemplo, un elemento la API *DigitalOut* de *Mbed OS*, el flujo de la información para controlar un LED conectado a una salida es el que se describe en la figura 2.10.



FIGURA 2.10. Ejemplo de flujo de información de pin de salida controlando un led en *Arm Mbed OS Simulator*.

La capa (*JS HAL*), que actúa como intermediario, distribuyendo eventos entre los componentes de las capas *JS UI* y *C HAL*, implementa un bus de eventos para permitir que la interfaz de usuario se suscriba a eventos de C++, y se comunica con la capa *C HAL* a través de los archivos *WebAssembly* y *JavaScript* generados por el proceso de compilación de *Emscripten*. La capa *JS UI*, maneja los eventos de la interfaz de usuario y solo se comunica con *JS HAL*.

En resumen, el diagrama representa cómo un programa escrito en lenguaje C/C++ interactúa con la capa de abstracción de la emulación a nivel de API hasta llegar a la interfaz de usuario en *JavaScript* para interactuar con los componentes de hardware virtuales, ya sea de la placa de desarrollo, o de componentes externos.

2.3.5. Análisis estructural de archivos y carpetas

En la figura 2.11 se exhibe la estructura de arbol de las carpetas y archivos de *Arm Mbed OS Simulator* en el momento en que fue clonado desde *GitHub* para la realización de este trabajo.

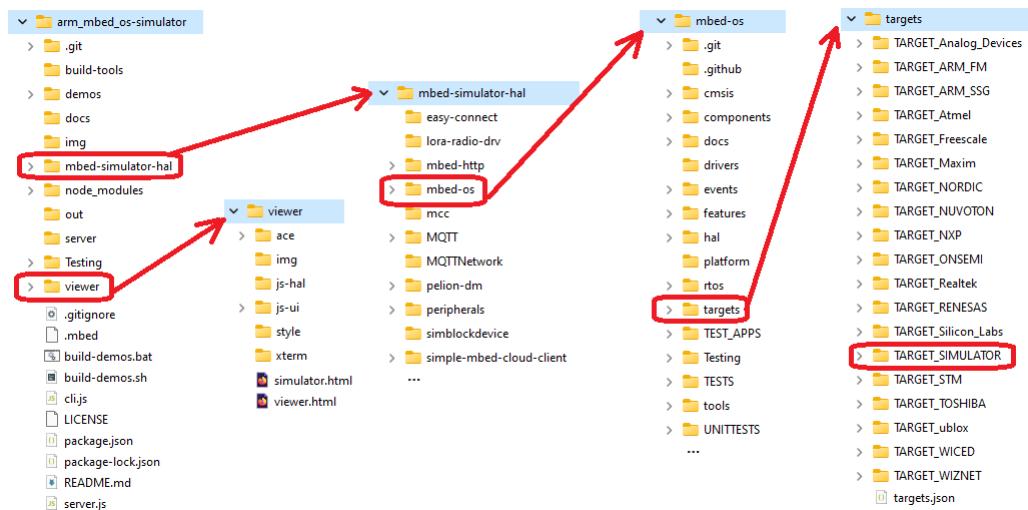


FIGURA 2.11. Estructura de carpetas y archivos de *Arm Mbed OS Simulator*.

La carpeta "build-tools" contiene tres archivos *JavaScript*:

1. *build-application.js*, que contiene funciones para construir aplicaciones en el contexto de *Arm Mbed OS Simulator*. Asimismo, contiene métodos para encontrar los periféricos y manejar la construcción de componentes.
2. *build-libmbed.js*, es un módulo de la aplaication web que realiza la compilación ((*build*)) y gestión de dependencias.

3. *helpers.js*, es un módulo de la aplicación web que proporciona funciones relacionadas con operaciones de sistema de archivos, compilación, y manipulación de directorios y archivos.

Dentro de la carpeta "demos" se encuentran todos los programas ejemplo que el usuario puede seleccionar en *Arm Mbed OS Simulator*. Estos están escritos en C/C++ y se utilizan como ejemplos didácticos que utilizan algunas de las funcionalidades de *Arm Mbed OS*.

La carpeta "server" contiene tres archivos *JavaScript*:

1. *compile.js*, este archivo realiza la generación de los archivos de salida compilados a partir del código fuente.
2. *get_ips.js*, proporciona una función para la configuración de las aplicaciones de red.
3. *launch-server.js*, define y configura un servidor web que permite ejecutar solicitudes de ejecución de aplicaciones, gestionar las conexiones de red y la comunicación LoRaWAN.

La carpeta "viewer" contiene los archivos necesarios para crear interacción con el navegador en el *Frontend*. Dentro de esta carpeta, se incluye:

- Carpeta *ace*: es un editor de código independiente escrito en *JavaScript*. El objetivo es crear un editor de código basado en web que coincida y amplíe las características, la facilidad de uso y el rendimiento de los editores nativos existentes, como TextMate, Vim o Eclipse.
- Carpeta *img*: contiene imágenes *.png* y *.svg* utilizadas para describir los LEDs y la placa de desarrollo respectivamente.
- Carpeta *js-hal*: incluye todos los archivos que implementan la capa de abstracción de hardware en *JavaScript*, o *JavaScript HAL*.
- Carpeta *js-ui*: incluye todos los archivos que implementan los componentes de interfaz de usuario en *JavaScript*, nombrado *JavaScript UI*.
- Carpeta *style*: contiene la hoja de estilos *simulator.css* que define los estilos de la página principal.
- Carpeta *xterm*: contiene la parte de *Frontend* de este paquete.
- Archivos *simulator.html* y *viewer.html*: definen el contenido de la página web de la interfaz gráfica de *Arm Mbed OS Simulator*.

La carpeta "mbed-simulator-hal" contiene las siguientes sub-carpetas:

- *easy-connect*, dentro de esta carpeta se encuentran archivos que facilitan la conexión a una red utilizando Ethernet para manipular conexiones de red, sockets y eventos.
- *lora-radio-drv*, proporciona un marco para interactuar y controlar el módulo de radio LoRa, lo que permite enviar y recibir datos, administrar el estado y el funcionamiento.
- *mcc*, establece una cola de eventos que se utilizará para manejar eventos en *Arm Mbed OS Simulator*.

- *MQTTNetwork*, establece la interfaz para la comunicación de red con el protocolo MQTT.
- *pelion-dm*, implementación de temporizadores y manejo de eventos para la plataforma mbed.
- *peripherals*, contiene las bibliotecas de *drivers* en C/C++ para los displays virtuales a conectar a la placa de desarrollo virtual.
- *simblockdevice*, establece una interfaz para interactuar con dispositivos de bloques que emula un dispositivo de almacenamiento físico en el navegador web.
- *mbed-os*, es la carpeta que contiene la biblioteca *Arm Mbed OS*.

La carpeta "mbed-os" se compone de:

- Carpeta *drivers*, dentro de esa carpeta se encuentran los diversos controladores que interactúan con los periféricos de hardware, y además, proporcionan una interfaz para acceder a ellos.
- Carpeta *events*, presenta archivos relacionados con la infraestructura de manejo de eventos para las tareas y operaciones de manera asíncrona.
- Carpeta *features*, contiene varias subcarpetas con varios módulos que gestionan la conexión celular en dispositivos integrados, proporcionan una interfaz para interactuar con LoRaWAN, manejar la manipulación de memoria para el uso de la pila LWIP, proporciona *mbed TLS*, implementación del protocolo de red 6LoWPAN, *sockets* de red, comunicación inalámbrica de corto alcance entre dispositivos y almacenamiento de datos en sistemas embebidos.
- Carpeta *hal*, contiene implementaciones específicas de hardware para diferentes plataformas y microcontroladores.
- Carpeta *platform*, proporciona una capa de abstracción adicional sobre la capa de abstracción de hardware (*HAL*).
- Carpeta *rtos*, contiene la implementación del sistema operativo en tiempo real (RTOS).
- Carpeta *targets*, cada sub-carpeta corresponde a una plataforma de hardware específica donde se puede ejecutar *Mbed OS*. Además, contiene información sobre cómo *Mbed OS* debe funcionar con cada plataforma en particular.
- Carpeta *TEST_APPS*, contiene ejemplos y aplicaciones de prueba de diferentes plataformas de hardware que se utilizan para probar y verificar diversas funcionalidades de *Mbed OS*.
- Carpeta *TESTS*, contiene pruebas unitarias y de integración que verifican y validan el correcto funcionamiento de los módulos y características de *Mbed OS*.
- Carpeta *tools*, contiene herramientas y utilidades para el desarrollo, compilación, depuración y prueba para diferentes plataformas de hardware y sistemas operativos.
- Carpeta *UNITTESTS*, contiene pruebas unitarias para diferentes componentes del sistema operativo Mbed.

- Archivo *mbed.h*, este archivo contiene declaraciones y definiciones iniciales para que estén disponibles para el desarrollo web del usuario.

Capítulo 3

Desarrollo e implementación

En este capítulo se detallan las herramientas utilizadas para el desarrollo del presente trabajo. Luego se exponen los cambios fundamentales realizados sobre la estructura del código existente, para portarlo a la EDU-CIAA-NXP y las mejoras introducidas. Se documenta además, el desarrollo de los nuevos componentes de hardware virtuales, los ports de las bibliotecas y ejemplos de programa, así como nuevos programas.

3.1. Herramientas de desarrollo

Se exponen las herramientas que se utilizan en 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 [42]: 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 [43], 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 [44]: 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 [45]: 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 [46]: 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 [47]: 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 [48]: 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.

3.2. Introducción al desarrollo

Luego de comprender el funcionamiento de *Arm Mbed OS Simulator*, se comenzó el desarrollo del emulador eliminando módulos y dependencias innecesarias, tales como: mbed-http, simple-mbed-cloud-client, features, rtos y todas las sub-carpetas dentro de la carpeta "target" excluyendo la sub-carpeta TARGET_SIMULATOR. Se modificaron las carpetas y archivos de configuración para adaptarlos al desarrollo del entorno del emulador para la placa EDU-CIAA-NXP. Luego, se reemplazó "mbed-os" por la biblioteca *sAPI*, pero se mantuvieron algunos componentes propios de Mbed, como "events" y "callback", para agilizar el desarrollo.

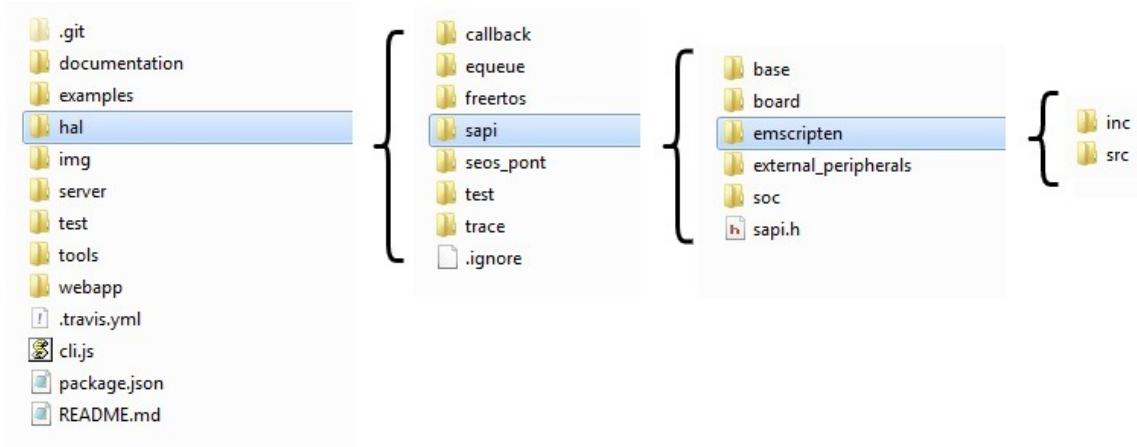
La figura 3.1 exhibe la nueva estructura de carpetas y archivos, realizada para el *Emulador de la placa EDU-CIAA-NXP*, el cual a partir de aquí se nombra como *EmuCIAA*.

3.3. Cambios realizados sobre la estructura del código existente

3.3.1. Nuevas tecnologías agregadas

Cambios en el Frontend

- SVG EDU-CIAA-NXP [49], SVG GLCD 128x64, SVG LCD DISPLAY 20x4, SVG DHT11, SVG Potenciómetro 10KΩ, SVG Termistor NTC, SVG Analog

FIGURA 3.1. Estructura de carpetas y archivos de *EmuCIAA*.

Stick, SVG LEDs: esta reutilización fue necesaria para brindar a los usuarios una experiencia visual interactiva. Asimismo, se incorporaron fuentes tipográficas en los dibujos SVG para las representaciones de texto. Estas fuentes están definidas en información vectorial, lo que permitió una visualización nítida y escalable en diferentes tamaños para las pantallas LCD.

- **Mocha [50]:** 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 [51]:** 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.

Cambios en el Backend

- **Check [52]:** 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 aislan 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 [53]:** es una biblioteca de pruebas unitarias especialmente diseñada para C, destacándose 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 [54]: 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[55] y Firmware v3[5]. 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 [56] 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-NXP 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-NXP, como ser la biblioteca del fabricante del microcontrolador (LPCopen[57]), o el acceso directo a registros físicos del microcontrolador, al igual que sucede con el emulador *Arm Mbed OS Simulator*.

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. 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.

Para su estudio, la figura 3.2 muestra la plataforma web del Emulador de la placa EDU-CIAA-NXP, dividida en tres partes.

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.

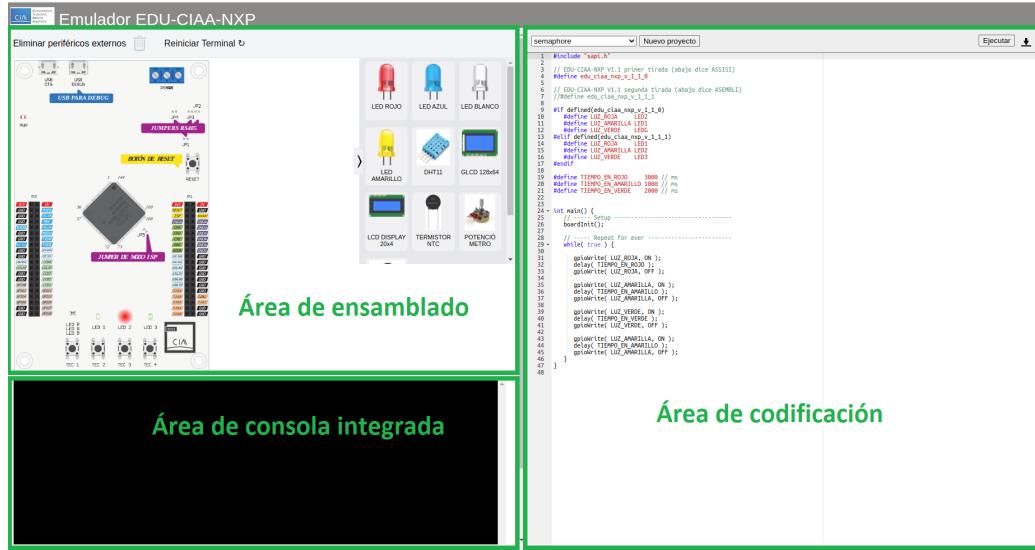


FIGURA 3.2. Plataforma de emulación para la placa EDU-CIAA-NXP.

- 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 y componentes externos se usaron dibujos en formato de gráficos vectoriales bidimensionales (SVG) por las siguientes características:

- Son más ligeras, entonces se cargan más rápido en el navegador.
- Por su capacidad de ser modificado por medio de *JavaScript*. Por lo tanto, se pudieron crear imágenes interactivas.
- Evitan que las imágenes se deformen y no pierden calidad.
- Permite programar animaciones.

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, que permitió mostrar dinámicamente en la placa el encendido y apagado.

Además, con el objetivo de optimizar la experiencia del usuario, esta área permite elegir uno o más dispositivos virtuales de entrada/salida desde la barra lateral *sidebars* derecho. En esta barra se exhiben todos los periféricos disponibles para la plataforma web.

Una vez que el usuario elige un periférico de la barra lateral y realiza la configuración de las conexiones, la barra automáticamente colapsa, ocultándose del área de ensamblado. Al colapsarse, se muestra el periférico integrado en la aplicación. Asimismo, el usuario tiene la opción de volver a expandir la barra lateral para agregar un nuevo periférico.

También, al añadir un periférico virtual, el usuario tiene la posibilidad de eliminarlo. Para realizarlo, simplemente debe seleccionar el periférico deseado, lo que habilitará el ícono de "Eliminar periféricos externos". Al hacer clic en el ícono, el periférico virtual es eliminado de forma automática y efectiva.

En la figura 3.3 se observa que el usuario puede elegir qué componente agregar a la aplicación y en la figura 3.4 se muestra que el componente se agregó a la aplicación.

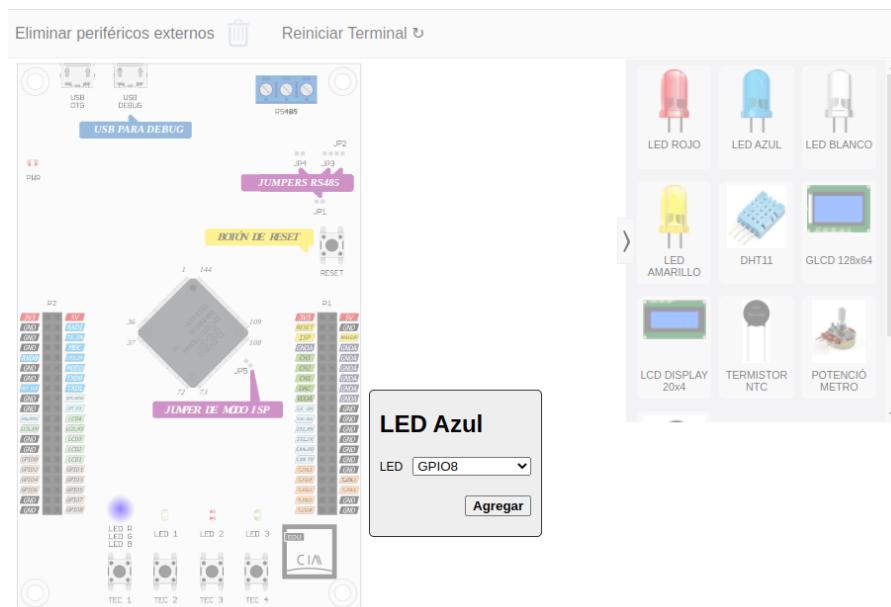


FIGURA 3.3. El usuario puede elegir un componente en la aplicación.

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).

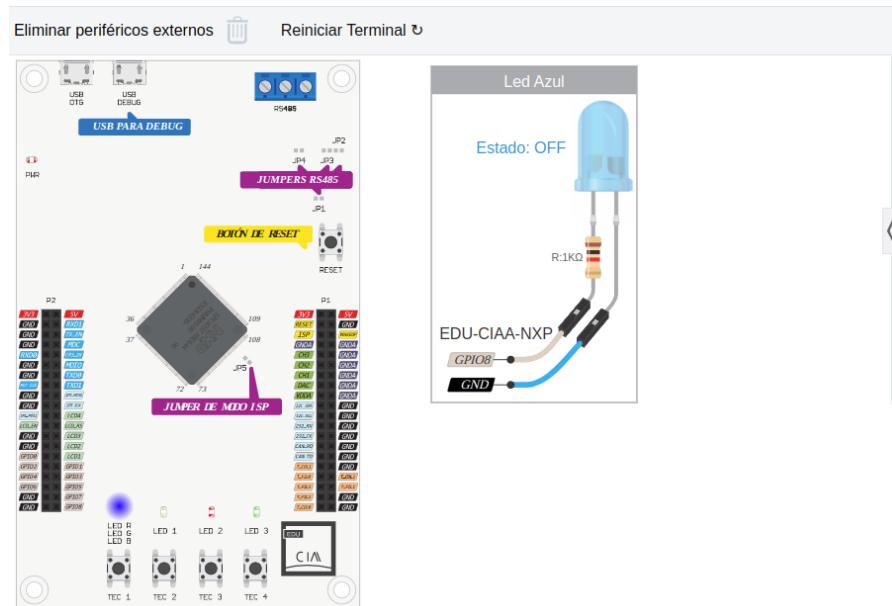


FIGURA 3.4. Periférico agregado en el área de ensamblado.

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 3.5 se muestra el código que generó los errores de compilación y en la figura 3.6 se observa los errores de compilación en el área de ensamblado.

También, se implementó una estructura jerárquica en la lista desplegable de ejemplos. El propósito es organizar y presentar los ejemplos agrupados por periféricos, de manera más ordenada y fácil de navegar para el usuario.

La figura 3.7 muestra la estructura jerárquica de la lista de ejemplos.

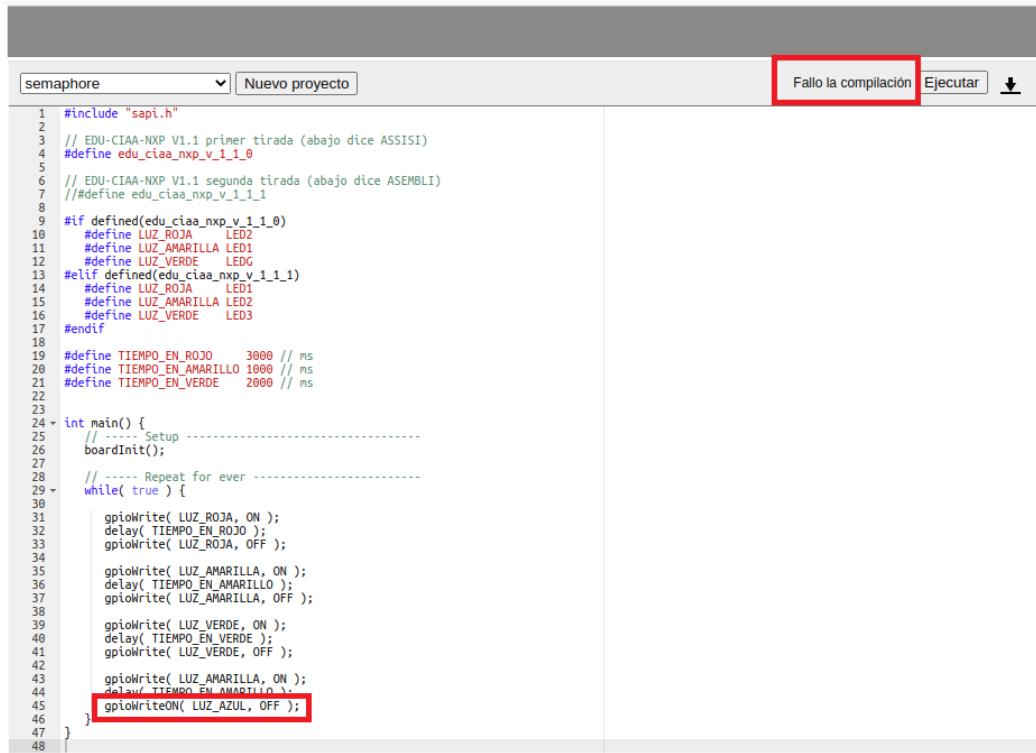
Ademas, al seleccionar algún ejemplo que contiene un periférico externo de la lista desplegable, la aplicación, automáticamente carga dentro del área de ensamblado el periférico con las conexiones a los pines configurados por defecto. Es decir, el usuario no tiene la necesidad de seleccionar y configurar el periférico desde la barra lateral del área de ensamblado.

La figura 3.8 muestra el periférico agregado automáticamente al seleccionar el ejemplo "potentiometer".

Área de consola integrada

Dentro del código traducido en el proceso de compilación por *Emscripten*, se encuentran las siguientes funciones definidas y configuradas previamente, las cuales son:

- `print`, esta función envía el texto a la terminal de la interfaz gráfica del emulador web, al utilizar la función `terminal.write`.
- `printErr`, se comunica con la consola de error del navegador, al usar `console.error`.



```

1  #include "sapi.h"
2
3 // EDU-CIAA-NXP V1.1 primer tirada (abajo dice ASSISI)
4 #define edu_ciaa_nxp_v_1_1_0
5
6 // EDU-CIAA-NXP V1.1 segunda tirada (abajo dice ASEMBLI)
7 //#define edu_ciaa_nxp_v_1_1_1
8
9 #if defined(edu_ciaa_nxp_v_1_1_0)
10   #define LUZ_ROJA LED2
11   #define LUZ_AMARILLA LED1
12   #define LUZ_VERDE LEDG
13 #elif defined(edu_ciaa_nxp_v_1_1_1)
14   #define LUZ_ROJA 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_ROJA, ON );
32         delay( TIEMPO_EN_ROJO );
33         gpioWrite( LUZ_ROJA, 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 3.5. Código que generó los errores de compilación.

Ambas funciones interactúan con el código *JavaScript*. La función `printErr` se comunica con la consola de error del navegador y la función `print` se comunica con la terminal del emulador web a través de 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 de `xterm.js` 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. La dependencia principal para el funcionamiento básico es el propio navegador web.
- API bien documentada.

Además de las características destacadas, `xterm.js` es una biblioteca que fue adoptada por diversos proyectos populares, tales como `VS Code`, `Hyper` y `Theia`. La amplia adopción de `xterm.js` por parte de estos proyectos contribuyó a la expansión de su comunidad de desarrolladores, quienes brindan un sólido respaldo y soporte.

En la figura 3.9 se puede observar la salida por consola de un programa y en la figura 3.10 se muestra el programa de usuario que generó la salida por consola.

```

CIAA
Computadora Industrial Avanzada Argentina
Emulador EDU-CIAA-NXP

Falló la compilación

Application failed to build (1)
/outUser/user_1695614173866.c:45:7: error: implicit declaration of function 'gpioWriteON' is invalid in C99 [-Werror,-Wimplicit-function-declaration]
    gpioWriteON( LUZ_AZUL, OFF );
^
/outUser/user_1695614173866.c:45:8: 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_1695614173866.c:45:20: error: use of undeclared identifier 'LUZ_AZUL'
    gpioWriteON( LUZ_AZUL, OFF );
^
2 errors generated.
/home/jenny/Documents/UBA/Tesis/emsdk/emsdk/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/Documents/UBA/Tesis/emsdk/emsdk/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 3.6. Errores de compilación.

3.4.2. JavaScript UI

Esta capa se desarrolló con el propósito de que se comunique 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 3.11 describe esta situación.

3.4.3. Aplicación de Usuario

La plataforma de emulación para la placa EDU-CIAA-NXP es una aplicación en línea, que se ejecuta en el *browser* del usuario. 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, presenta una limitación en las unidades de tiempo especificadas por los valores en las constantes, por ejemplo cuando el usuario define `TASK1_PERIODICITY` con un valor de 1000 para ser usado dentro del código siguiente:

```

1 #define TASK1_PERIODICITY 1000
2
3 if( task1Counter++ == TASK1_PERIODICITY ){
4     task1();
5     task1Counter = 0;

```

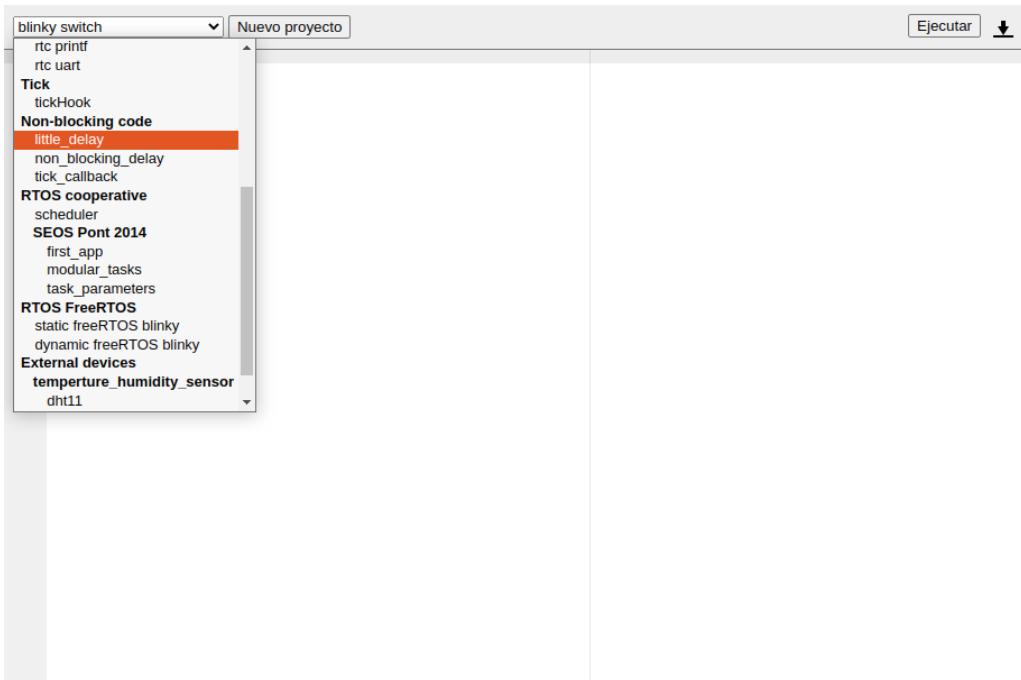


FIGURA 3.7. Estructura jerárquica de ejemplos.

6 }

CÓDIGO 3.1. Ejemplo TASK1_PERIODICITY

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. Estas diferencias en la ejecución se deben a las limitaciones inherentes de *JavaScript* y *Emscripten*, que pueden afectar la precisión del tiempo.

3.5. 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.

3.5.1. 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 incluyó 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

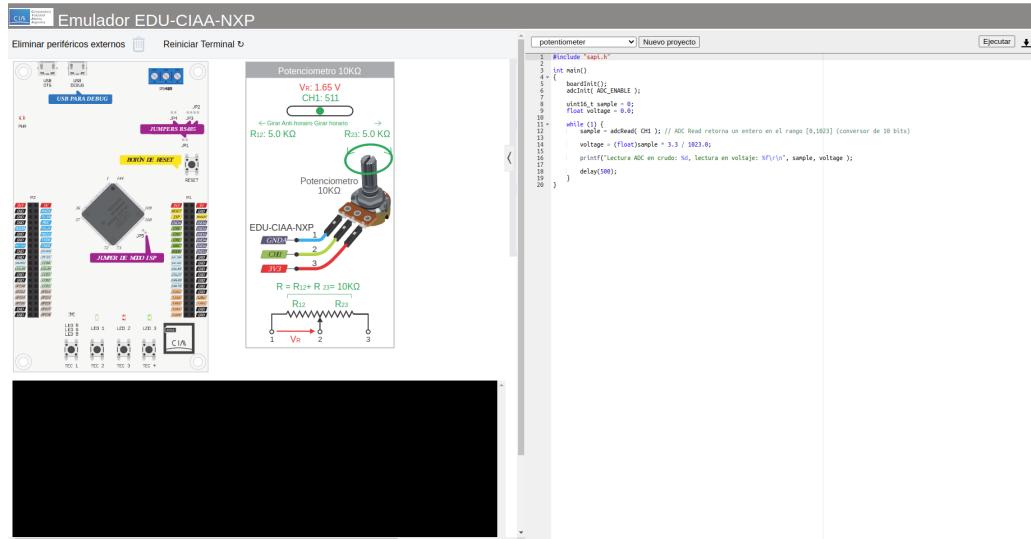


FIGURA 3.8. Carga automática del periférico.

```

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 3.9. Salida de la terminal serie.

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
gpioInit	gpioMap_t, gpioInit	bool_t
gpioRead	gpioMap_t	bool_t
gpioWrite	gpioMap_t, bool_t	bool_t
gpioToggle	gpioMap_t	bool_t

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 archivos 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

```

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.mday = 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
27        // Leer fecha y hora
28        rtcRead( &rtc ); // en la variable de estructura rtc te queda la fecha/hora actual
29
30        // Envio por UART de forma humanamente legible
31        // %%02d %%02d/%04d, %%02d:%%02d:%%02d\r\n",
32        //   rtc.mday, 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 ningun 5.0. */
40    return 0 ;
41
42 }
43

```

FIGURA 3.10. Programa de usuario que imprime por consola.

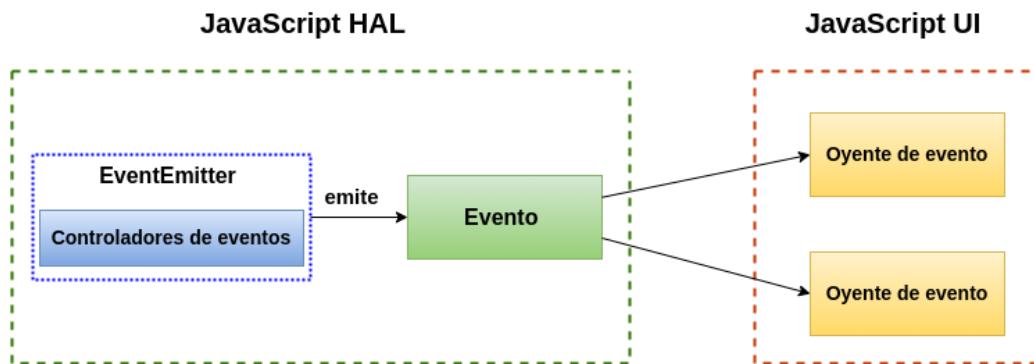


FIGURA 3.11. Diagrama de bloques de los oyentes de *EventEmitter* en la capa UI.

plataforma de emulación.

La figura 3.12 muestra las dependencias que se usan para la implementación de `sapi_gpio.c` del proyecto CIAA y en la figura 3.13 se muestran las dependencias utilizadas en el emulador para el mismo módulo, logrando el *port* al emulador de las funciones para el manejo del periférico GPIO.

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 reutilizó el archivo de encabezado `sapi.h` que cumple con la misma funcionalidad que en la biblioteca *sAPI*, la cual consiste en incluir todos los módulos que conforman la biblioteca para utilizarla en el programa de usuario.

Además, se reutilizaron los archivos de encabezado: `sapi_datatypes.h` y `sapi_peripheral_map.h` incluidos en todos los módulos de la biblioteca *sAPI*.

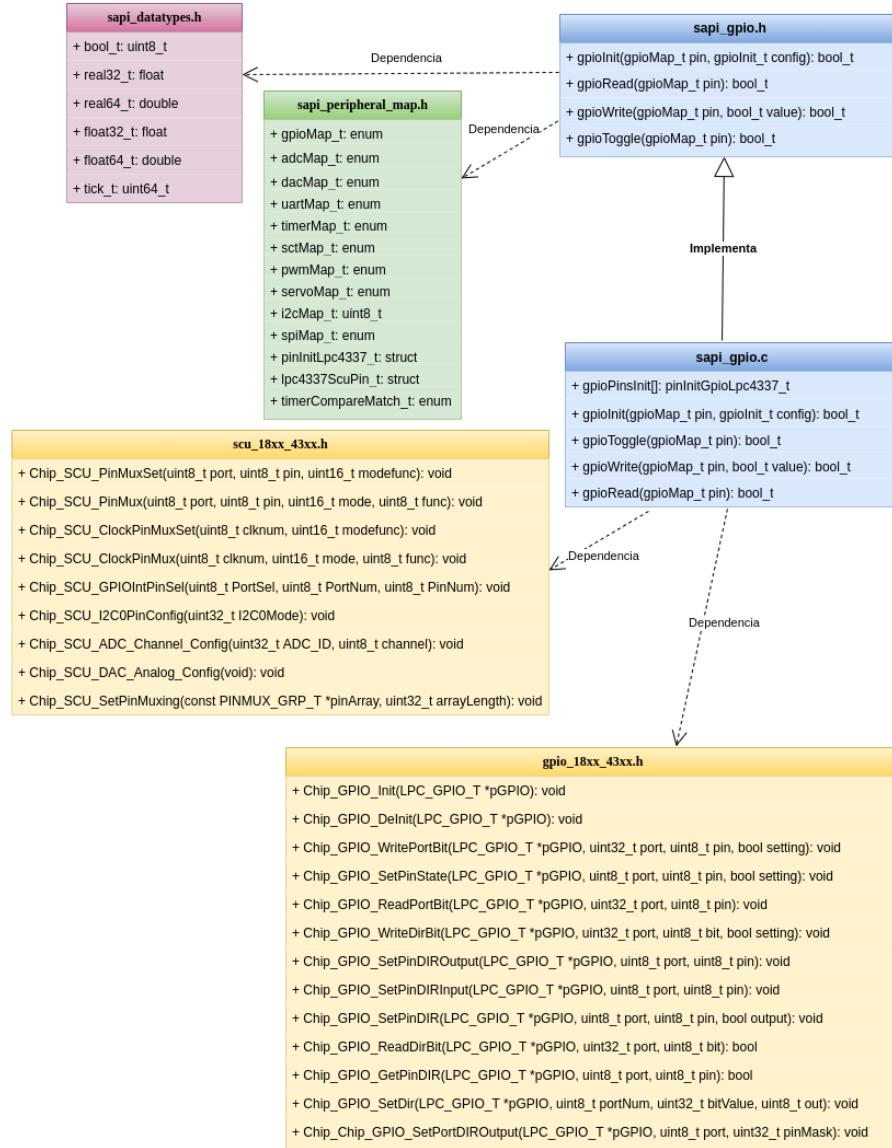


FIGURA 3.12. Diagrama de dependencias del módulo *GPIO* de la biblioteca *sAPI* del proyecto CIAA.

del proyecto CIAA. Esta reutilización busca 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. Se puede observar que se reutilizó 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.

3.5.2. 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.

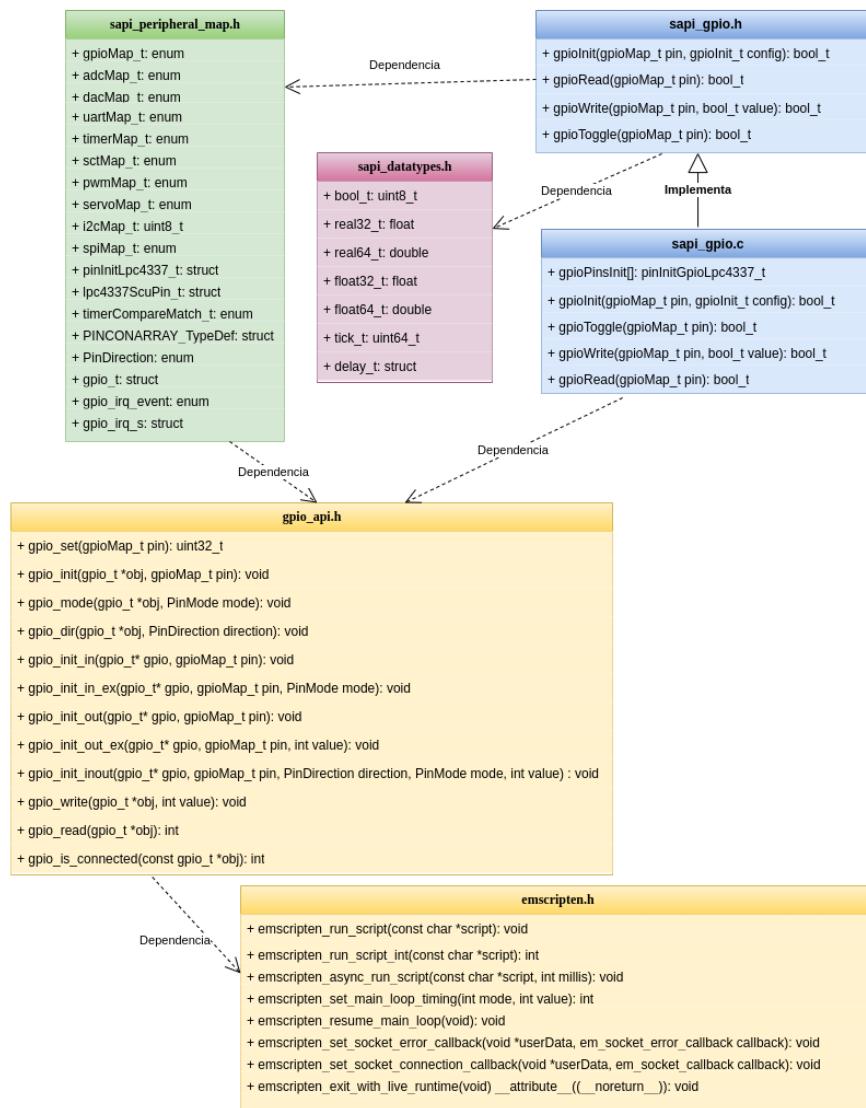


FIGURA 3.13. Diagrama de dependencias del módulo *GPIO* de la plataforma de emulación para la placa EDU-CIAA-NXP.

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 se ejecuta en el entorno de *Node.js* para compilar el código C a *JavaScript*. Además, incluye la biblioteca *emscripten.h* presentes en la capa C (C HAL). Esto permite que el código C use estas funciones nativas para interactuar con el entorno de *JavaScript*, llamando a funciones *JavaScript* desde código C o viceversa.

El proceso de compilación con *Emscripten* involucra los siguientes pasos:

- Preprocesamiento: donde prepara el código fuente antes de la compilación real del código C de manera similar a un compilador tradicional. Esto incluye el manejo de directivas como la inclusión de archivos de cabecera `#include`, directivas que permiten la inclusión condicional de código:

TABLA 3.2. Tipos de datos de `sapi_peripheral_map.h` que se reutilizan en la plataforma de emulación.

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_CRS_DV	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		

#ifdef, #ifndef, #else, #endif, la expansión de macros #define, etc.
Lo que facilita el trabajo del compilador.

- Compilación: *Emscripten* utiliza LLVM para compilar el código C en un formato intermedio binario llamado *bitcode*. Además, LLVM proporciona múltiples componentes que pueden intercambiarse entre sí, a diferencia de los compiladores *GCC* que presentan una estructura monolítica.
- Optimización: Después de obtener el *bitcode*, el compilador busca mejorar el rendimiento, la eficiencia y reducir el tamaño del código resultante, por tanto, aplica diversas técnicas de optimizaciones antes de traducirlo a *JavaScript* o *WebAssembly*. Estas optimizaciones pueden incluir eliminación de código muerto, reordenamiento de instrucciones, detección y eliminación de código redundante, *Inlining de funciones*, entre otras.
- Generación de código *JavaScript*: Finalmente, *Emscripten* toma el *bitcode* optimizado y lo traduce a código *JavaScript*.

En la figura 3.14 se muestra el diagrama con el principal funcionamiento de *Emscripten*.

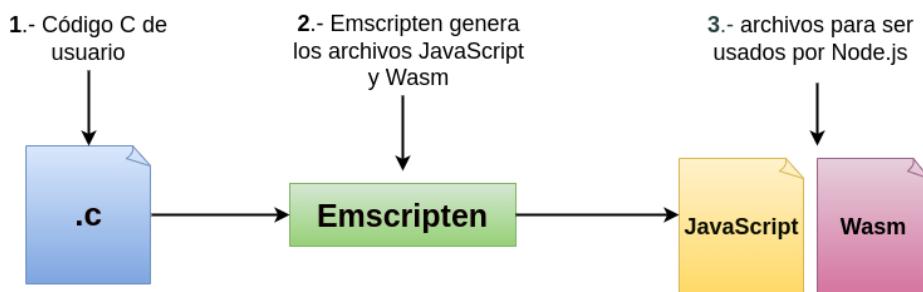


FIGURA 3.14. Diagrama de funcionamiento de *Emscripten*.

En ese sentido, para la aplicación de usuario dentro de la plataforma, cuando se ejecuta un programa de usuario, *Emscripten* utiliza una función del módulo de *Node.js* para la generación de un identificador único que será utilizado para los archivos generados. Este identificador está compuesto por el prefijo `user_` y un número entero que representa el tiempo en milisegundos. Es decir, *Emscripten* creará los siguientes archivos:

- Archivo `user_tiempoenmilisegundos.js`: resultado final de la compilación y contiene el código *JavaScript* que representa el programa *C*.
- Archivo `user_tiempoenmilisegundos.wasm`: contiene el código binario equivalente al código *C* compilado. Se utiliza si el navegador del usuario admite *WebAssembly* y proporciona un rendimiento mejorado en comparación con el código *JavaScript* puro.
- Archivo `user_tiempoenmilisegundos.js.components`: contiene datos utilizados por el programa.
- Archivo `user_tiempoenmilisegundos.wasm.map`: contiene rutas de mapas a los archivos de código *C* que fueron compilados en formato *Json*.
- Archivo `user_tiempoenmilisegundos.wast`: representa el módulo *WebAssembly* generado, pero en una representación de texto legible.

Estos archivos se ubicarán dentro del directorio de salida `outUser`, el cual fue predefinido en la configuración de la aplicación.

3.5.3. 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 3.15 se muestra el modelo de *publicación/suscripción*.

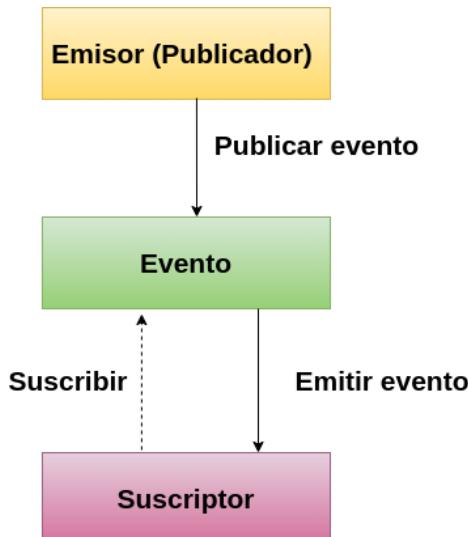


FIGURA 3.15. Modelo de *publicación/suscripción*.

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 3.16 muestra el diagrama de bloques de la instancia de *EventEmitter* en la plataforma de emulación.

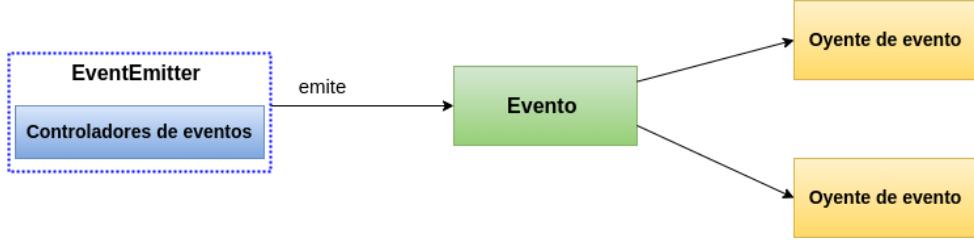


FIGURA 3.16. Diagrama de bloques de *EventEmitter* implementado en la plataforma.

En la sección de 3.6, se mostrará la implementación de este modelo en los archivos *JavaScript* del emulador web.

3.5.4. *sapi_gpio*

Para comenzar a emular la biblioteca *sapi_gpio* del proyecto CIAA, se identificaron las funciones principales que el entorno de la plataforma web debería ofrecer al usuario. La siguiente tabla 3.3 muestra las funciones del archivo de código fuente *sapi_gpio* en la biblioteca *sAPI* del proyecto CIAA que también están presentes en el emulador.

TABLA 3.3. Funciones *sapi_gpio*

Función	Parámetros	Tipo de retorno
gpioInit	gpioMap_t pin, gpioInit_t config	bool_t
gpioRead	gpioMap_t pin	bool_t
gpioWrite	gpioMap_t pin, bool_t value	bool_t
gpioToggle	gpioMap_t pin	bool_t

Además, para lograr replicar el comportamiento de cada función emulada de manera que, al invocarlas, produzcan resultados similares a los que se obtendrían al utilizar las funciones con el hardware físico, se utilizó la capa de C *HAL*. Esta capa interactúa con la capa *Javascript HAL*, que se encarga de comunicarse con la interfaz de usuario, permitiendo mostrar el comportamiento de los pines GPIO programados en la aplicación del usuario.

Entonces, se utilizó la capa de emulación correspondiente a C *HAL* para emular las siguientes funcionalidades:

- `Chip_GPIO_Init(LPC_GPIO_PORT)` se utiliza para inicializar y configurar el hardware de los pines GPIO de la placa. En la capa de emulación C *HAL*, utilizando *Emscripten*, también se realizaron configuraciones de variables para representar los tipos de pines e inicializarlos.
- `Chip_GPIO_SetDir(LPC_GPIO_PORT, gpioPort, (1 << gpioPin), GPIO_OUTPUT)` se utiliza para establecer si un pin GPIO se utilizará para recibir datos (entrada) o enviar datos (salida). De manera

similar, en la capa de abstracción de datos en C, se implementaron configuraciones similares para la capa *Javascript HAL* usando las macros de *Emscripten*.

- `Chip_GPIO_SetPinState(LPC_GPIO_PORT, gpioPort, gpioPin, 0)` se utiliza para gestionar el estado de los pines GPIO, permitiendo configurarlos como salidas y establecer su valor (alto o bajo). En la capa de abstracción de datos C, esta función actualiza la estructura de datos utilizada para almacenar información sobre la configuración de los pines GPIO. Además, registra el valor del pin especificado como parámetro.
- `Chip_GPIO_ReadPortBit(LPC_GPIO_PORT, gpioPort, gpioPin)` se utiliza para obtener el estado actual de un pin GPIO específico en la placa, lo que permite leer datos provenientes de dispositivos externos o de otros componentes conectados a los pines GPIO. En la capa de emulación C *HAL*, se emula la lectura del estado de un pin GPIO utilizando la información almacenada en la estructura de datos.

Para emular las funciones mencionadas anteriormente, se utilizó la macro `EM_ASM_`, mediante la cual se incrustó código *JavaScript* directamente en el código C. Este código *JavaScript* incrustado se compila junto con el código C y se ejecutará en el entorno de ejecución de *Emscripten* cuando la aplicación de usuario invoque a esas funciones. La figura 3.17 muestra el diagrama de bloques de la macro `EM_ASM_`.

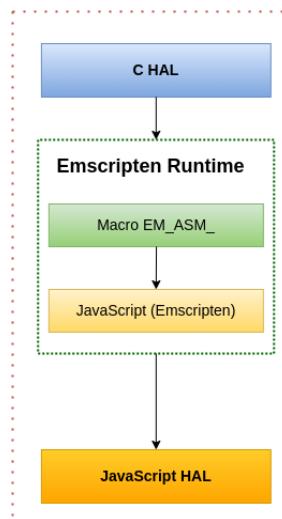


FIGURA 3.17. Diagrama de bloques de la macro `EM_ASM_`.

Asimismo, para emular las interacciones entre la interfaz de usuario y las GPIO TEC1, TEC2, TEC3 y TEC4, se utilizó en la capa C *HAL* la macro `EMSCRIPTEN_KEEPALIVE`. Su funcionamiento se explica en la siguiente sección 3.5.5.

A continuación, en la capa *Javascript HAL*, se distribuyen eventos a la capa *Javascript UI* para notificarle que el pin GPIO ha sido configurado desde la capa C *HAL*. Estos eventos incluyen información que será útil para gestionar los pines GPIO en la capa de interfaz de usuario. La figura 3.18 muestra el diagrama de bloques del envío de eventos de la capa *Javascript HAL* a la capa *Javascript UI*.

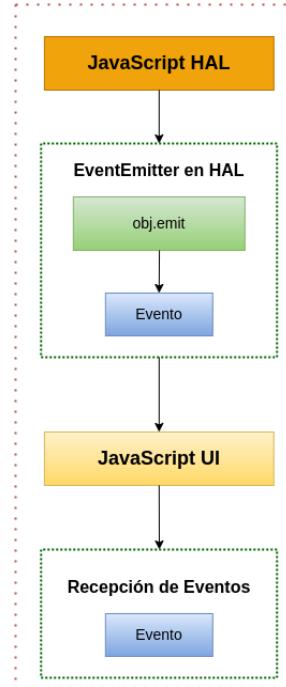


FIGURA 3.18. Diagrama de bloques del envío de eventos de la capa *Javascript HAL*.

Por consiguiente, en la capa *Javascript UI* se manipulan esos eventos, y en consecuencia, se actualiza la interfaz de la plataforma web para mostrar al usuario los cambios.

3.5.5. *sapi_tick*

En la tabla 3.4 se puede observar las funciones del archivo de código fuente *sapi_tick* en la biblioteca *sAPI* del proyecto CIAA y en el emulador.

TABLA 3.4. Funciones *sapi_tick*

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

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 *SysTick_Config* 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 emular la funcionalidad de *SysTick_Config*, se utilizó la capa de emulación correspondiente a *C HAL*. Esta capa de emulación permitió ejecutar código

JavaScript en el contexto de *Emscripten*, lo que posibilitó replicar el comportamiento del temporizador *SysTick*.

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, ejecuta la función establecida como *callback* mediante la función de *sAPI* `tickCallbackSet()`, 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 capítulo 4 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`. Primariamente, 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.

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 3.19 se muestra el funcionamiento de `EMSCRIPTEN_KEEPALIVE`.

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 3.20 el funcionamiento de `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.

3.5.6. `sapi_delay`

La tabla 3.5 expone las funciones del archivo de código fuente `sapi_delay` en la biblioteca *sAPI* del proyecto CIAA y en el emulador.

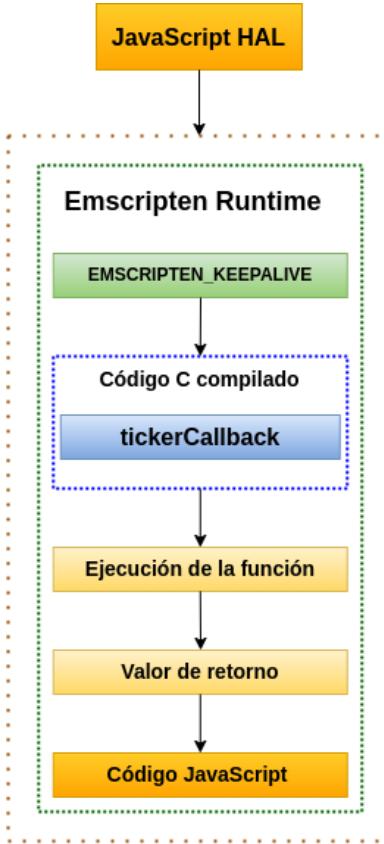


FIGURA 3.19. Diagrama de bloques `EMSCRIPTEN_KEEPALIVE`.

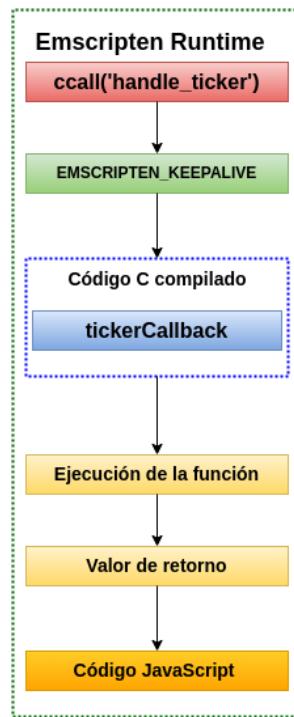
La función `delay` en la biblioteca `sAPI` del proyecto CIAA crea una pausa en la ejecución del programa durante el tiempo especificado en `duration_ms` implementando un bucle de espera. Este bucle se ejecutará mientras la diferencia de tiempo entre `tickRead()` y `startTime`(inicio actual de `tickRead()`) 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 las funciones nativas de *Emscripten* en la capa de emulación *C HAL*. En consecuencia, se aprovechó su eficiencia y precisión.

Para emular las funciones de espera de la biblioteca *C* se implementó la función `emscripten_sleep`, que utiliza funciones asíncronas 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.

FIGURA 3.20. Diagrama de bloques de la función `ccall`.TABLA 3.5. Funciones `sapi_delay`

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

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 3.21 representa el funcionamiento de `emscripten_sleep`.

3.5.7. freeRTOS

Para emular la funcionalidad de las tareas de *freeRTOS* en el contexto del emulador web, se utilizó 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 *Mbed events*:

- `int equeue_create`: crea una cola de eventos, configura e inicializa los recursos de plataforma necesarios, como semáforos y mutexes.

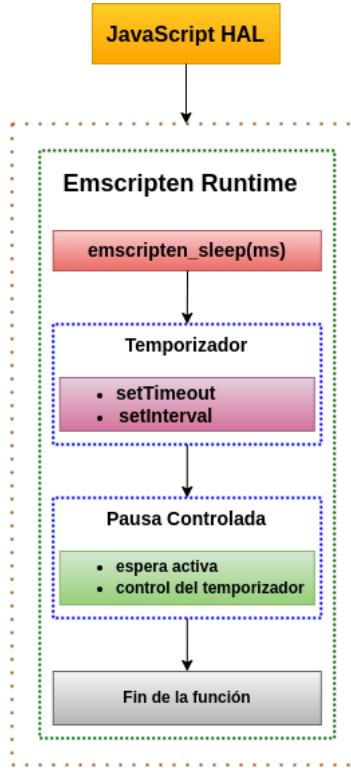


FIGURA 3.21. Diagrama de bloques `emscripten_sleep`.

- `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 a 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 emular 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 3.6 expone algunos de los conceptos importantes de *freeRTOS* que se cumplen en el emulador.

TABLA 3.6. 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*.

3.5.8. *sapi_dht11*

Al emular los periféricos externos de la biblioteca *sAPI* del proyecto CIAA, se continuó con la misma lógica de programación utilizada para interactuar con los periféricos de la placa EDU-CIAA-NXP. Asimismo, se mapearon las funcionalidades ofrecidas por la biblioteca *sAPI* a la plataforma web. En la siguiente tabla 3.7 se muestran las funciones presentes en ambas plataformas.

TABLA 3.7. Funciones *sapi_dht11*

Función	Parámetros	Tipo de retorno
dht11Init	int32_t gpio	void
dht11Read	float *phum, float *ptemp	bool_t

En esta primera versión de la plataforma web, no se ofrece la capacidad gráfica de las interacciones virtuales entre la placa y los periféricos externos. Sin embargo, el usuario debe realizar las configuraciones manualmente, eligiendo las conexiones correctas, que luego serán verificadas en la capa de *JavaScript UI*. Además, en *JavaScript UI*, principalmente se centró el trabajo de emular el envío de los datos de temperatura y humedad del sensor DHT11 al microcontrolador.

Entonces, en la capa de abstracción de datos C *HAL* se implementó la lectura de los datos provenientes de la capa *JavaScript HAL* al usar la macro `EM_ASM_INT` de *Emscripten*. A continuación, en la figura 3.22 se presenta el diagrama de bloques de la macro `EM_ASM_INT`.

La capa *JavaScript HAL* recibe los datos enviados desde *JavaScript UI* y los transmite a la capa C *HAL*. La generación de los datos emulados de temperatura y humedad, se realiza en *JavaScript UI* a través de dos opciones elegidas por el usuario:

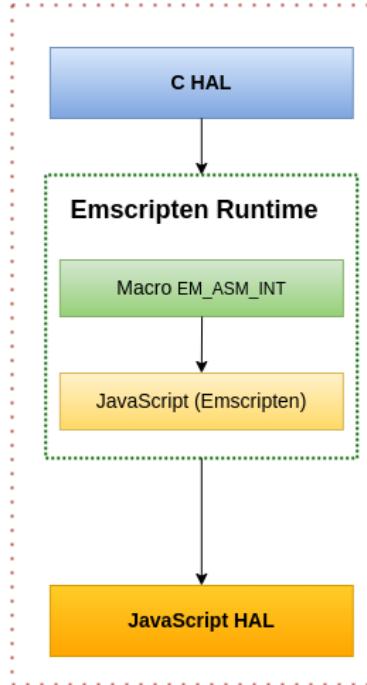


FIGURA 3.22. Diagrama de bloques de la macro `EM_ASM_INT`.

- Obtener los datos de temperatura y humedad local conectándose a una central meteorológica a través de la geolocalización del navegador del usuario. Sin embargo, si el servidor donde se encuentra desplegada la plataforma web no puede acceder al servicio de geolocalización del navegador por motivos de seguridad, o el usuario no permite el acceso, entonces se realizará la consulta a la central meteorológica utilizando la ubicación predeterminada de la ciudad de Buenos Aires. Acto seguido se actualizará la interfaz gráfica con los datos de temperatura y humedad.
- Generar los datos manualmente haciendo click en la interfaz gráfica que representa a la temperatura y humedad. De esta manera, el usuario puede generar los datos según su elección.

En la figura 3.23 se muestra el diagrama de bloques de la capa de interfaz de usuario *JavaScript UI* con las dos opciones de usuario.

3.5.9. *sapi_adc*

Para comenzar, se realizó el mapeo de las funciones del módulo de la biblioteca *sAPI* a la plataforma web. La tabla 3.8 expone las funciones presentes en ambas plataformas.

TABLA 3.8. Funciones *sapi_adc*

Función	Parámetros	Tipo de retorno
adcInit	adcInit_t config	void
adcRead	adcMap_t analogInput	uint16_t

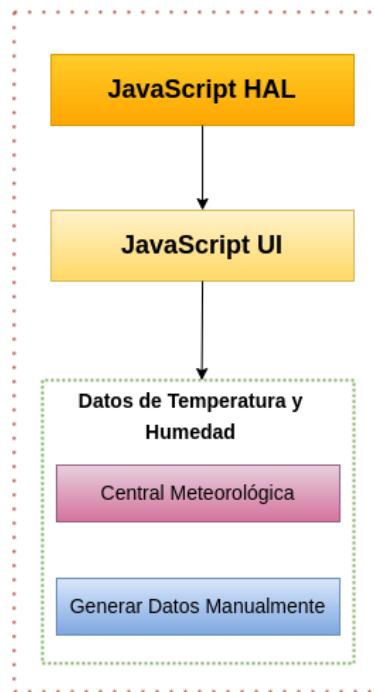


FIGURA 3.23. Diagrama de bloques de la capa *JavaScript UI* con las dos opciones para el usuario.

El módulo *adc* fue implementado en el emulador y es utilizado por varios periféricos externos, que incluyen: el potenciómetro, el termistor NTC y el joystick. De esta manera, permite al usuario realizar pruebas de funcionamiento de un *adc* real.

Luego, se implementaron las funciones de inicialización y de lectura del componente *adc* en la capa *C HAL*. Posteriormente, son utilizadas por la capa *JavaScript HAL* para la interacción con el hardware y permitir al usuario trabajar con los periféricos externos en un entorno web. La figura 3.24 presenta el diagrama de bloques de las capas: *C HAL* y *JavaScript HAL* para el *adc*.

En la capa *JavaScript UI*, se implementó la obtención de datos para los periféricos externos que interactúan con el *adc*. Por ejemplo, para el potenciómetro, los datos son establecidos por el usuario a través de la interfaz gráfica utilizando el componente HTML *input* de tipo *range*. De esta manera, el usuario al deslizar este componente dentro de un rango mínimo y máximo establecido, se va realizando el siguiente cálculo para obtener el valor del *adc* correspondiente:

```
1 window.JSHal gpio.write( self.dataPin.ADC, range.value / 3.3 * 1023);
```

CÓDIGO 3.2. Cálculo del ADC para el potenciómetro.

En consecuencia, los cálculos actualizados se muestran en la interfaz gráfica del emulador web.

En el caso del termistor NTC, se implementó en la interfaz gráfica un elemento gráfico (termómetro) para representar la temperatura en grados celsius. A medida que el usuario ajusta el termómetro, la temperatura en kelvin se va actualizando en función de la temperatura en grados celsius y, además, el valor del *adc* se actualiza mediante la siguiente función:

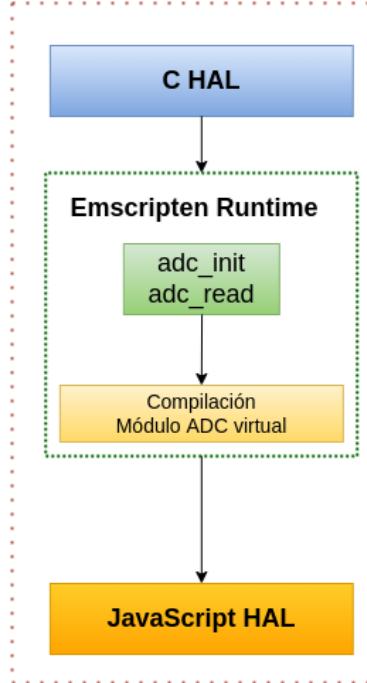


FIGURA 3.24. Diagrama de bloques de *C HAL* y *JavaScript HAL* para el módulo *adc*.

```

1 ThermistorNTC.prototype.updateSampleADC= function (R_NTC) {
2     let R_NTC_float = parseFloat(R_NTC);
3     let R_10k_float = parseFloat(R_10k);
4     let Vsupply_float = parseFloat(Vsupply);
5     let VoutT = (R_NTC_float * Vsupply_float) / (R_10k_float +
R_NTC_float);
6     Vout = parseFloat(VoutT.toFixed(2));
7
8     this.sample = parseFloat((Vout * 1023.0 / Vsupply).toPrecision
(4));
9     console.log('this.sample ', this.sample);
10    window.JSHal.gpio.write(this.dataPin.ADC, this.sample);
11  };
  
```

CÓDIGO 3.3. Cálculo ADC del termistor NTC.

Además, se implementó un componente web para el joystick, encargado de gestionar los movimientos y acciones en las interacciones con el usuario. Los datos de los movimientos de los ejes X e Y del joystick son obtenidos y se utilizan para realizar cálculos de voltajes en las respectivas resistencias de cada eje, así como también, para calcular el valor del *adc*. A modo de referencia se muestra las cálculos que se hicieron para el eje X del joystick:

```

1 var VRx = Joy.GetVRx();
2 var voltage = (Math.floor(VRx/ 3.3 * 1023)* 3.3 / 1023.0) .
toFixed(2);
3 joyVRx.textContent = voltage;
4 joyADCx.textContent = Math.trunc(VRx/ 3.3 * 1023);
5 window.JSHal.gpio.write(self.dataPin.ADCx, VRx/ 3.3 * 1023);
  
```

CÓDIGO 3.4. Cálculo de la resistencia del eje X y del ADC.

Luego, para cada periférico externo, el cálculo del *adc* es enviado a la capa *JavaScript HAL*. En la figura 3.25 se muestra el diagrama de bloques con la interacción de ambas capas de desarrollo.

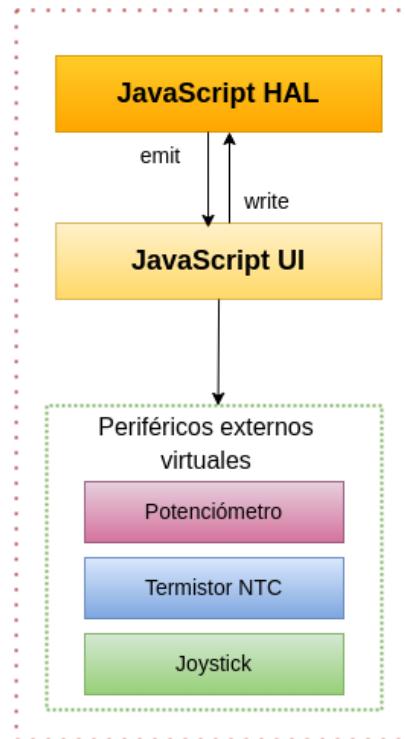


FIGURA 3.25. Diagrama de bloques con la interacción de las capas *JavaScript HAL* y *JavaScript UI*.

3.6. Caso de estudio

El usuario escribe o modifica el programa llamado `blinky` en lenguaje *C*, que utiliza la biblioteca *sAPI* para interactuar con los periféricos de hardware GPIO y controlar el LED. A continuación, ejecuta el programa dentro de la plataforma web. Para lograr esto, Node.js se encarga de ejecutar los comandos necesarios para que *Emscripten* realice la compilación del código *C*, que incluye:

- El código de la aplicación de usuario escrito en lenguaje *C*.
- El archivo `sapi_gpio.c` de la capa *Biblioteca C*.
- El archivo `gpio_api.c` de la capa *C HAL*.

El proceso de compilación comienza con el preprocessamiento del código *C*, que incluye el manejo de directivas del preprocesador como `#include` y `#define`. Luego, el compilador utiliza *LLVM* para compilar el código *C* en *bitcode*. Después, de obtener el *bitcode* realiza optimizaciones para mejorar el rendimiento y reducir el tamaño del código resultante. Finalmente, *Emscripten* toma el *bitcode* optimizado y lo traduce a código *JavaScript*, lo que permite que el programa escrito originalmente en *C* pueda ser ejecutado dentro del entorno web. Los archivos resultantes de este proceso incluyen:

- `user_tiempoenmilisegundos.js`.

- `user_tiemponmilisegundos.wasm`.
- `user_tiemponmilisegundos.wast`.
- `user_tiemponmilisegundos.js.components`.
- `user_tiemponmilisegundos.wasm.map`.

Una vez que los archivos `.js` y `.wasm` se han generado a partir del código C mediante *Emscripten*, pueden interactuar con el código *JavaScript* de las capas: *JavaScript HAL* y *JavaScript UI*. Ahora bien, desde estos archivos *JavaScript* de la *HAL* y *UI*, se pueden invocar directamente las funciones C compiladas como si fueran funciones *JavaScript* regulares, y podrán ser ejecutadas en el entorno del navegador. Por lo tanto, esto permite que el programa C interactúe con el resto del código *JavaScript* de la aplicación web y que las funciones C puedan ser utilizadas y llamadas de manera transparente en el navegador.

La figura 3.26 muestra la interacción del usuario con el sistema y el orden en que se producen. Además, se muestra los mensajes que se pasan entre las dependencias.

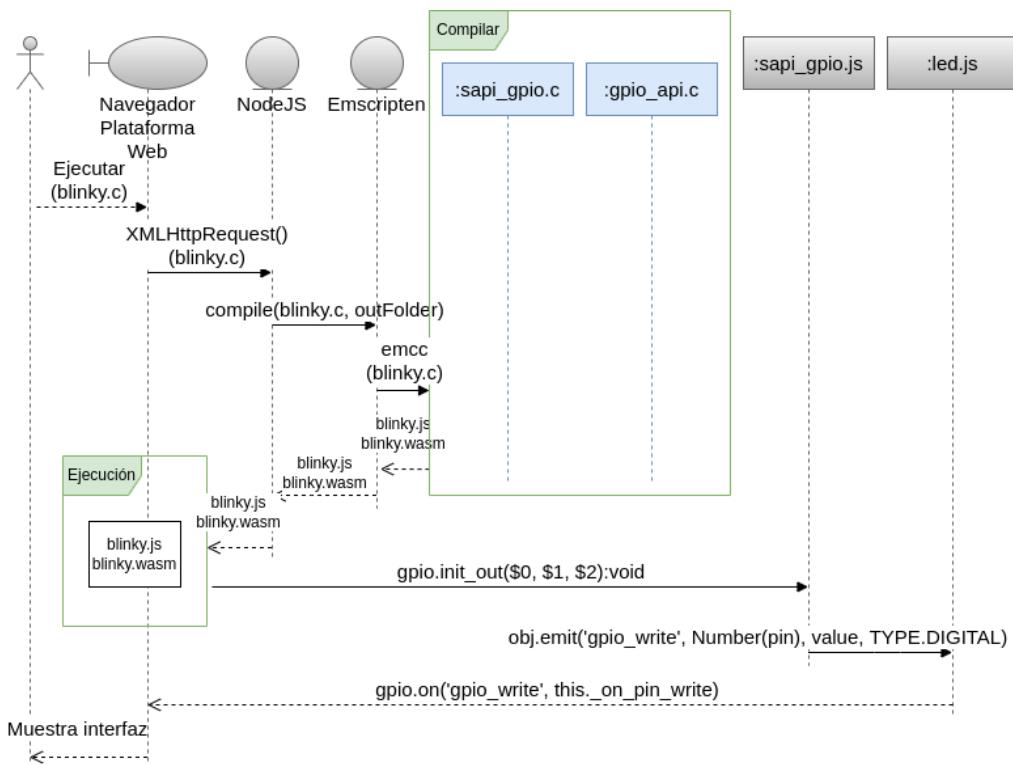


FIGURA 3.26. Interacción del usuario con las dependencias del emulador.

Además, la capa *JavaScript HAL*, que interactúa con el código *JavaScript* resultante de la compilación de la *Biblioteca C* y *C HAL*, se encarga de notificar los eventos ocurridos en esas capas para el programa de usuario *blinky*. Mediante la función `write`, se realiza la activación del evento que escribe en la GPIO. Como resultado, 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 3.27 muestra el diagrama en bloques del evento con el nombre `gpio_write`.

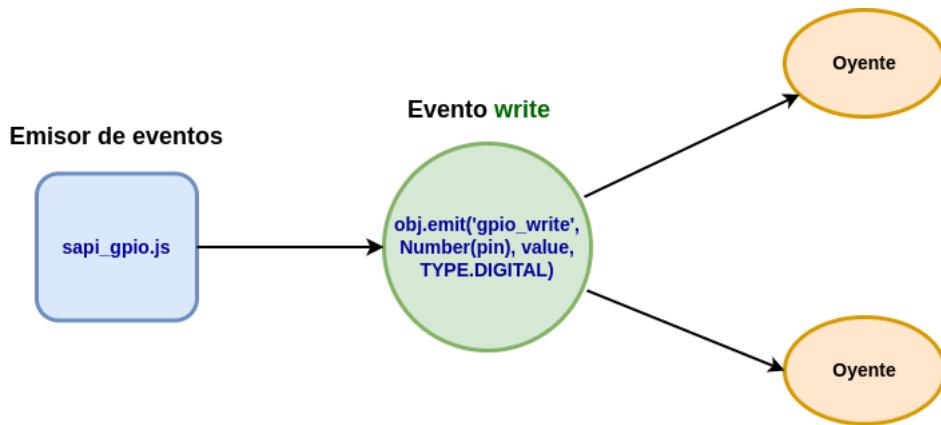


FIGURA 3.27. Activación de evento con el nombre `gpio_write`.

En ese sentido, en la capa *JavaScript UI* cuando se emite el evento con el nombre `gpio_write`, cualquier oyente que esté suscrito a ese evento podrá escucharlo y realizar las acciones correspondientes para la funcionalidad que se requiere. En este caso, la acción solicitada es encender el LED.

La figura 3.28 muestra el diagrama en bloques del oyente suscrito al evento `gpio_write`.

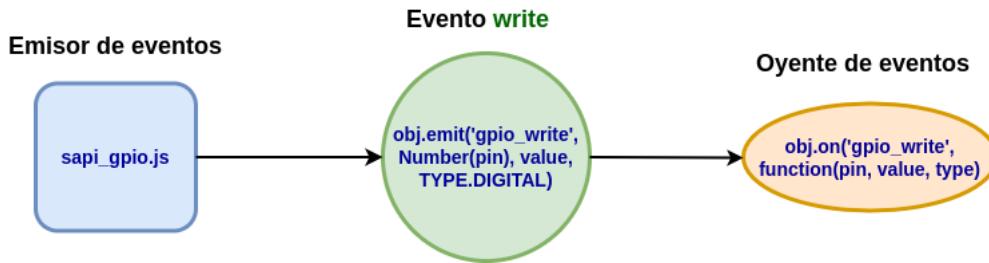


FIGURA 3.28. GPIO oyente del evento con el nombre `gpio_write`.

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

En la figura 3.29 se presenta para una función de la GPIO, la interacción entre todas las capas de programación.

3.7. Despliegue

Para hacer público el emulador de la plataforma en un servidor web, se realizó el proceso de despliegue en el servidor de *DigitalOcean* mediante los siguientes pasos:

- Crear una cuenta: implicó registrar una cuenta en la plataforma. Después de iniciar sesión, se creó un *droplet*, que es un servidor virtual. Luego, se instaló el sistema operativo *Ubuntu*, y se configuró la región geográfica donde se ubicaría el servidor y la cantidad de RAM.
- Acceso al servidor: después que el *droplet* fue creado, se pudo obtener la dirección IP pública y la clave SSH para acceder al servidor.

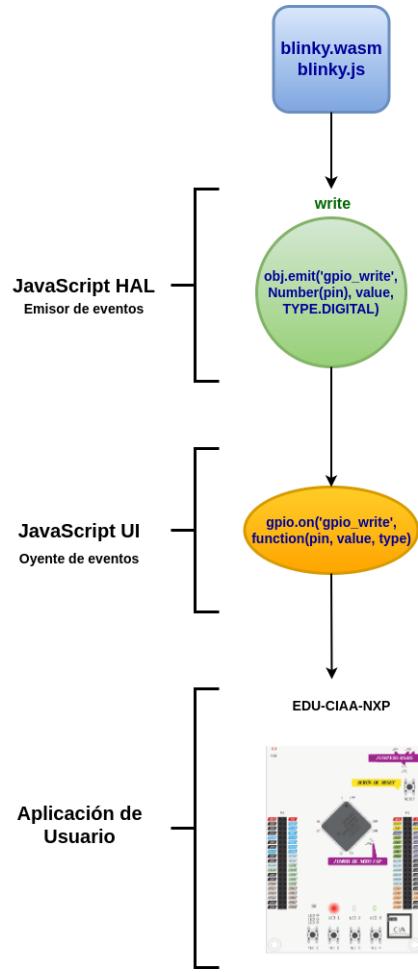


FIGURA 3.29. Interacción entre todas las capas de programación.

- Configuración: implicó instalar las herramientas, como *Mbed CLI* y *Emscripten*, y también, los servicios necesarios, como los entornos de ejecución de *Node.js* y *Python*. Además, se realizó las configuraciones de las reglas de *firewall*.
- Copiar la plataforma web al servidor: se utilizó *GitHub* para clonar el repositorio en el servidor. El código del presente trabajo, se encuentra en el repositorio de GitHub [58].
- Iniciar el emulador web: se utilizó la herramienta *TMUX* para mantener la sesión y la ventana de la terminal donde se ejecuta la aplicación de forma persistente, incluso cuando se cierra la conexión *SSH*. El presente trabajo, se encuentra actualmente ejecutandose en <http://134.209.168.175:7900>.

Capítulo 4

Ensayos y resultados

En este capítulo se presentan las pruebas realizadas para comprobar el 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. 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 real y la del emulador web, además, proporcionó 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 [59]	Pruebas de <i>request</i> de la plataforma y APIs.
CMocka	Pruebas automatizadas para el backend.
GCC	Para la compilación de las pruebas de backend.
Check	Para la ejecución de las pruebas de backend.
Mocha	Pruebas automatizadas para el frontend.
Chai	Pruebas automatizadas para el frontend.
Tera Term [60]	Emulador de la terminal serial.

4.2. 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 posibilitó 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, la consola indicará lo siguiente:

- El porcentaje de pruebas probadas.
- La cantidad de pruebas que aprobaron.
- La cantidad de pruebas que fallaron.
- Si alguna prueba falla, entonces, indicara 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 salida por consola durante la depuración de las pruebas unitarias y la figura 4.2 muestra la segunda parte.

```
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ gcc -DTEST_BUILD -o hal/ciaa_tests_dht11 hal/test/check_sapi_dht11.c hal/sapi/external_peripherals/temperature_humidity/dht11/src/sapi_dht11.c hal/test/mock/dht11_api_mock.c -I . -I ./hal/sapi -I ./hal/test/mock -I hal/sapi/base -I hal/sapi/board -I hal/sapi/external_peripherals/temperature_humidity/dht11/include -Icheck -Icmocka -Ipthread -I m -Isubunit
Running suites(s): CMocka Suite sapi_dht11 dht11init
100% Checks: 1, Failures: 0, Errors: 0
Running suites(s): Check Suite sapi_dht11 dht11init
Mock: dht11_init invokedado con gpio = 10
100% Checks: 1, Failures: 0, Errors: 0
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ ./hal/ciaa_tests_dht11
Running suites(s): Check Suite sapi_gpio gpioInit
Mock: gpio_init invokedado con gpioMap_t = 50
100% Checks: 1, Failures: 0, Errors: 0
Running suites(s): CMocka Suite sapi_gpio gpioRead, gpioWrite, gpioToggle
Mock: gpio_get_0_invocado con gpioMap_t = 50
Mock: gpio_set_0_invocado con gpioMap_t = 50
100% Checks: 3, Failures: 0, Errors: 0
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ ./hal/ciaa_tests_gpio
Running suites(s): Check Suite sapi_gpio gpioInit
Mock: gpio_init invokedado con gpioMap_t = 50
100% Checks: 1, Failures: 0, Errors: 0
Running suites(s): CMocka Suite sapi_rtc rtcRead, rtcWrite
Mock: rtc_int invokedado
Mock: delay Invocado con duration_ms = 2100
100% Checks: 1, Failures: 0, Errors: 0
Running suites(s): Check Suite sapi_rtc rtcRead, rtcWrite
Mock: rtc_write invokedado
100% Checks: 2, Failures: 0, Errors: 0
```

FIGURA 4.1. Primera parte de la salida por consola de las pruebas unitarias con *CMocka* o *Check*.

```
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ gcc -DTEST_BUILD -o hal/ciaa_tests_delay hal/test/check_sapi_delay.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 -I hal/sapi/base -I hal/sapi/board -I hal/sapi/soc/peripherals/include -Icheck -Icmocka -Ipthread -I m -Isubunit
Running suites(s): Check Suite sapi_delay delayInaccurateMs, delayInaccurateUs, delay, delayInit, delayWrite
Mock: delay_ms invokedado con int = 500
Mock: delay_us invokedado con int = 1000
Mock: delay_ns invokedado con int = 100
Mock: delay_ms invokedado con int = 200
100% Checks: 6, Failures: 0, Errors: 0
Running suites(s): CMocka Suite sapi_delay delayRead
100% Checks: 1, Failures: 0, Errors: 0
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ ./hal/ciaa_tests_delay
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ gcc -DTEST_BUILD -include /dev/null -o hal/ciaa_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/eqeueue_mock.c -I . -I ./hal/sapi -I ./hal/test/mock -I hal/sapi/base -I hal/sapi/board -I hal/sapi/soc/peripherals/include -Icheck -Icmocka -Ipthread -I m -Isubunit
Running suites(s): Check Suite sapi_tick tickInit, tickWrite, tickPowerSet, tickRead
Mock: tick_detach invokedado con id = 1
Mock: tick_init invokedado con ms = 1
100% Checks: 4, Failures: 0, Errors: 0
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ ./hal/ciaa_tests_tick
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ gcc -DTEST_BUILD -o hal/ciaa_tests_tasks hal/test/check_tasks.c hal/test/check_tasks_api_mock.c -Ihal/sapi/base -Ihal/sapi/board -I ./hal/test/mock -Icheck -Icmocka -Ipthread -I m -Isubunit
Running suites(s): CMocka Suite task getTickCount
Mock: xstartTickCount invokedado
100% Checks: 1, Failures: 0, Errors: 0
Running suites(s): Check Suite task xstartTickCount setTaskDelayUntil
Mock: xstartTickCount invokedado
Mock: setTaskDelayUntil invokedado con xTimeIncrement = 2
100% Checks: 2, Failures: 0, Errors: 0
```

FIGURA 4.2. Segunda parte de la salida por consola de las pruebas unitarias con *CMocka* o *Check*.

4.3. 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*.

Al igual que en las pruebas de unidad se utilizó el compilador GCC para compilar. A continuación la figura 4.3 muestra los resultados por consola de las pruebas de integración.

```
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/nock -I hal/sapi/base -I hal/sapi/board -I hal/sapi/soc/peripherals/inc -lcheck -lcmocka -lrt -pthread -lm -lsbunit
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/nock -I hal/sapi/base -I hal/sapi/board -I hal/sapi/soc/peripherals/inc -lcheck -lcmocka -lrt -pthread -lm -lsbunit
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/nock/rtc_api_mock.c hal/test/mock/sapi_delay_mock.c -I . -I ./hal/sapi -I ./hal/test/nock -I hal/sapi/base -I hal/sapi/board -I hal/sapi/soc/peripherals/inc -lcheck -lcmocka -lrt -pthread -lm -lsbunit
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.

4.4. 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 estas pruebas automatizadas permitió que se ejecuten de forma rápida y confiable de manera recurrente.

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

Además, permitió asegurar que cada componente 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 salida por consola durante la depuración de las pruebas de interfaz con *Mocha*.

```
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
Contentido del ejemplo Blinky
  ✓ Compruebe que el ejemplo Blinky se ejecuta en el primer acceso
Blinky
  ✓ Ejecutar ejemplo blinky
Prueba del ejemplo static_mem_freeRTOS_blinky
  ✓ El ejemplo static_mem_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 (1019ms)
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. Salida por consola durante la depuración de las pruebas de interfaz con *Mocha*.

4.5. Integracion Continua

Para implementar la integración continua del código se evaluaron dos plataformas: GitLab CI y Travis CI. Ambas herramientas, proporcionaron excelentes resultados y demostraron ser igualmente eficaces. Sin embargo, se encontraron algunas diferencias importantes en su configuración y en cómo se presentan las pruebas en cada plataforma.

En el caso de GitLab CI, toda la configuración se realizó directamente en la plataforma de GitLab, lo que facilitó la integración con el repositorio de la plataforma web y permitió una gestión más centralizada del proceso de CI/CD.

Por otro lado, con Travis CI, se realizaron configuraciones separadas de GitHub. Esto implicó un enfoque más descentralizado, lo que puede ser útil si se trabaja en varios proyectos alojados en diferentes repositorios.

Después de evaluar ambas opciones, se decidió utilizar ambas tecnologías, dado que son gratuitas para código abierto. Además, esta elección brindó mayor flexibilidad y resguardo en caso de problemas con alguna de estas herramientas.

En general, la experiencia con ambas herramientas fue positiva y permite mejorar la calidad y eficiencia del proceso de desarrollo mediante la automatización de las pruebas unitarias y de integración. Además de los despliegues continuos.

En ambas plataformas, la información que se muestra en la consola proporciona la siguiente información útil:

- Resultado de las pruebas: la consola muestra si las pruebas se ejecutaron con éxito o si hubo fallas en alguna de ellas.
- Detalle de los fallos: en el caso de que alguna prueba falle, la consola proporcionará información detallada, como el nombre de la prueba, el nombre del archivo y la línea de código donde ocurrió el error.
- Información sobre el entorno de prueba: la consola muestra detalles sobre el entorno de prueba utilizado tanto en Travis CI como en GitLab CI, que incluye la versión del lenguaje de programación, la secuencia de dependencias instaladas y otros detalles de las pruebas.
- Duración de las pruebas: la consola muestra el tiempo que tardaron todas las pruebas en ejecutarse, de manera que, permite identificar las pruebas que deben ser modificadas para optimizar su rendimiento.
- Logs de ejecución: la consola mostrará registros detallados de la ejecución de las pruebas, que incluyen mensajes del progreso de las pruebas, información de depuración y los resultados de las pruebas.

La siguiente figura 4.5 presenta la consola de Travis CI.

```

1srunuit
526 The command "gcc -DTEST_BUILD -o hal/ciaa_tests.tasks hal/test/check_tasks_api.c hal/test/mock/tasks_api_mock.c -Ihal/sapi/base -Ihal/sapi/board -I./hal/test/mock -lcheck -lcmocka -lrt -pthead -lm -lsubunit" exited with 0.
527 $ ./hal/ciaa_tests.tasks
528 Running suite(s): CMocka Suite Task getTickCount
529 Mock: xStartTickCount Invocado
530 100%: Checks: 1, Failures: 0, Errors: 0
531 Mock: setTaskDelayUntil Invocado con XTimeIncrement = 2
532 100%: Checks: 2, Failures: 0, Errors: 0
533 Mock: setTaskDelayUntil Invocado con XTimeIncrement = 2
534 100%: Checks: 2, Failures: 0, Errors: 0
535 The command "./hal/ciaa_tests.tasks" exited with 0.
536 $ 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
537 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 -pthead -lm -lsubunit
538 The command "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
539 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 -pthead -lm -lsubunit
540 The command "./hal/ciaa_test_integration_delay_tick" exited with 0.
541 $ ./hal/ciaa_test_integration_delay_tick
542 100%: Checks: 1, Failures: 0, Errors: 0
543 The command "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
544 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 -pthead -lm -lsubunit
545 The command "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
546 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 -
547 pthead -lm -lsubunit" exited with 0.
548 $ ./hal/ciaa_test_integration_gpio_interrupt
549 100%: Checks: 1, Failures: 0, Errors: 0
550 The command "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
551 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 -pthead -lm -lsubunit
552 The command "./hal/ciaa_test_integration_gpio_interrupt" exited with 0.
553 $ ./hal/ciaa_test_integration_rtc_delay
554 100%: Checks: 1, Failures: 0, Errors: 0
555 The command "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
556 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 -pthead -lm -lsubunit
557 The command "./hal/ciaa_test_integration_rtc_delay" exited with 0.
558
559 $ echo "Pruebas unitarias completadas con éxito."
560 $ echo "Pruebas de integración completadas con éxito."
561 $ echo "Pruebas unitarias completadas."
562 $ echo "Pruebas de integración completadas."
563
564 Done. Your build exited with 0.

```

FIGURA 4.5. Información de las pruebas que se ejecutaron en Travis CI.

La figura 4.6 presenta la consola de GitLab CI.

```

922 $ gcc -DTEST_BUILD -o hal/ciaa_tests_tasks hal/test/check_tasks_api.c hal/test/mock/tasks_api_mock.c -Ihal/sapi/base -Ihal/sapi/board -I./hal/test/mock -Icheck -lcmocka -lrt -pthread -lm -lsubunit
923 $ ./hal/ciaa_tests_tasks
924 Running suite(s): CMocka Suite task getTickCount
925 Mock: xstartTickCount invocado
926 100%: Checks: 1, Failures: 0, Errors: 0
927 Running suite(s): Check Suite task xstartTickCount setTaskDelayUntil
928 Mock: xstartTickCount invocado
929 Mock: setTaskDelayUntil invocado con xTimeIncrement = 2
930 100%: Checks: 2, Failures: 0, Errors: 0
931 $ 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./hal/sapi -I./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/peripherals/inc -Icheck -lcmocka -lrt -pthread -lm -lsubunit
932 $ ./hal/ciaa_test_integration_delay_tick
933 Running suite(s): Test de Integración
934 100%: Checks: 1, Failures: 0, Errors: 0
935 $ 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./hal/sapi -I./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/peripherals/inc -Icheck -lcmocka -lrt -pthread -lm -lsubunit
936 $ ./hal/ciaa_test_integration_gpio_interrupt
937 Running suite(s): Test de Integración
938 Mock: gpio_init_out invocado con gpioMap_t = 50
939 Mock: gpio_init_out invocado con gpioMap_t = 50
940 100%: Checks: 3, Failures: 0, Errors: 0
941 $ 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./hal/sapi -I./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/peripherals/inc -Icheck -lcmocka -lrt -pthread -lm -lsubunit
942 $ ./hal/ciaa_test_integration_rtc_delay
943 Running suite(s): Test de Integración
944 Mock: rtc_init invocado
945 Mock: delay invocado con duration_ms = 2100
946 100%: Checks: 1, Failures: 0, Errors: 0
947 Running after_script
948 echo "Pruebas unitarias completadas."
949 Pruebas unitarias completadas.
950 echo "Pruebas de integración completadas."
951 Pruebas de integración completadas.
952 Cleaning up project directory and file based variables
953 Job succeeded

```

FIGURA 4.6. Información de las pruebas que se ejecutaron en GitLab CI.

4.5.1. Prueba de acceso

Para verificar y validar el acceso mediante solicitudes HTTP al servidor donde se encuentra publicado el emulador de la plataforma web, se utilizó la herramienta Postman.

Antes de comenzar con el ensayo, se creó un caso de prueba con el propósito de ser una guía estructurada y documentada para verificar si el acceso al servidor funciona como se espera.

ID Caso de prueba: CP01

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

Pre-condición:

- La computadora del usuario tiene conexión a internet y un navegador web instalado.

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 al encender y apagar el LEDB.
- FALLA: La plataforma no muestra ningún ejemplo predeterminado en ejecución.

Resumen del Test:

- Despues 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 del servidor donde se encuentra la plataforma, devolvió en formato **JSON** la respuesta del servidor.

La figura 4.7 muestra la plataforma de emulación ejecutando el ejemplo predeterminado **Blinky**.

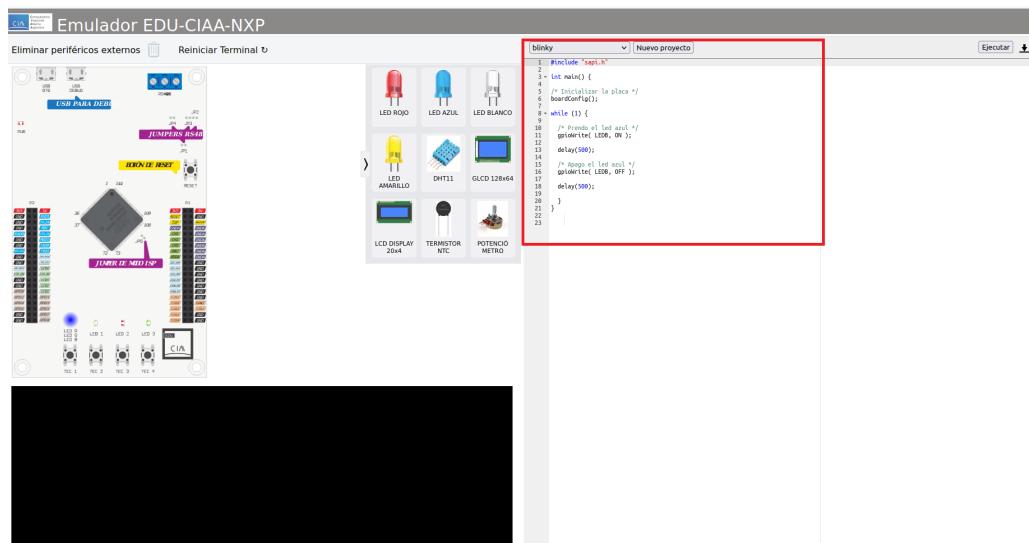


FIGURA 4.7. Prueba plataforma emulador ejecutando el ejemplo **Blinky**.

Postman es una aplicación que permite realizar pruebas API. La herramienta permitió realizar pruebas **HTTP requests** y acceder al servidor de la plataforma de emulación. Se obtuvo la respuesta en diferentes formatos como **JSON**, **XML**, **HTML** y **Text**.

En la figura 4.8 se muestra la petición y respuesta de acceso a la plataforma por medio de la herramienta **Postman**.

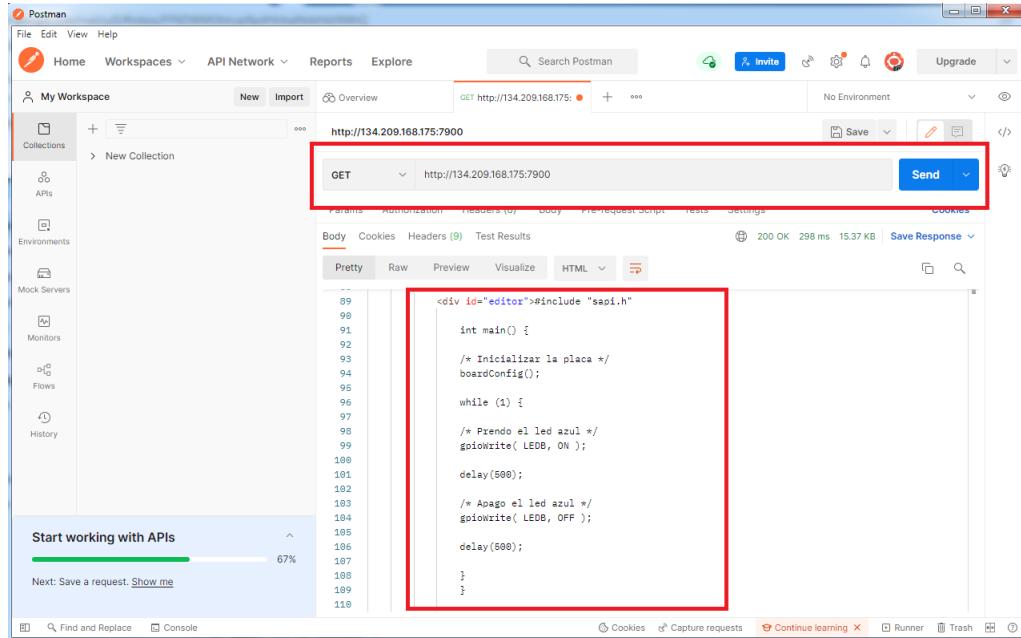


FIGURA 4.8. Respuesta del servidor.

4.6. Pruebas de funcionamiento

Para evaluar, verificar el comportamiento y desempeño del microcontrolador en el entorno web, se realizan a continuación pruebas funcionales con uno de los periféricos internos seleccionados, que es el RTC. Además, se elige para la prueba un periférico externo, el DHT11.

También, se realiza una prueba de un nuevo proyecto "tick hook", que muestra las diferencias en el funcionamiento y los resultados obtenidos entre la plataforma web y la placa física EDU-CIAA-NXP.

4.6.1. Prueba del ejemplo *rtc printf*

Para el ensayo en la plataforma de emulación web se siguió con los pasos del siguiente caso de uso:

ID Caso de prueba: CP02

Descripción: la plataforma de emulación permite al usuario ejecutar el ejemplo *rtc printf*.

Pre-condición:

- La computadora del usuario tiene conexión a internet y un navegador web instalado.

Flujo principal:

1. El usuario ingresa al entorno web de la plataforma de emulación para la placa EDU-CIAA-NXP.
2. El usuario selecciona desde la lista desplegable el ejemplo *rtc printf*.
3. La plataforma muestra en pantalla al usuario el código que corresponde al ejemplo *rtc printf*.

4. El usuario hace click en el botón "Ejecutar".
5. La plataforma muestra en la consola integrada la salida del rtc.
6. El usuario realiza la descarga del ejemplo *rtc printf* al hacer click en el botón de descarga, ubicado en el área de codificación.

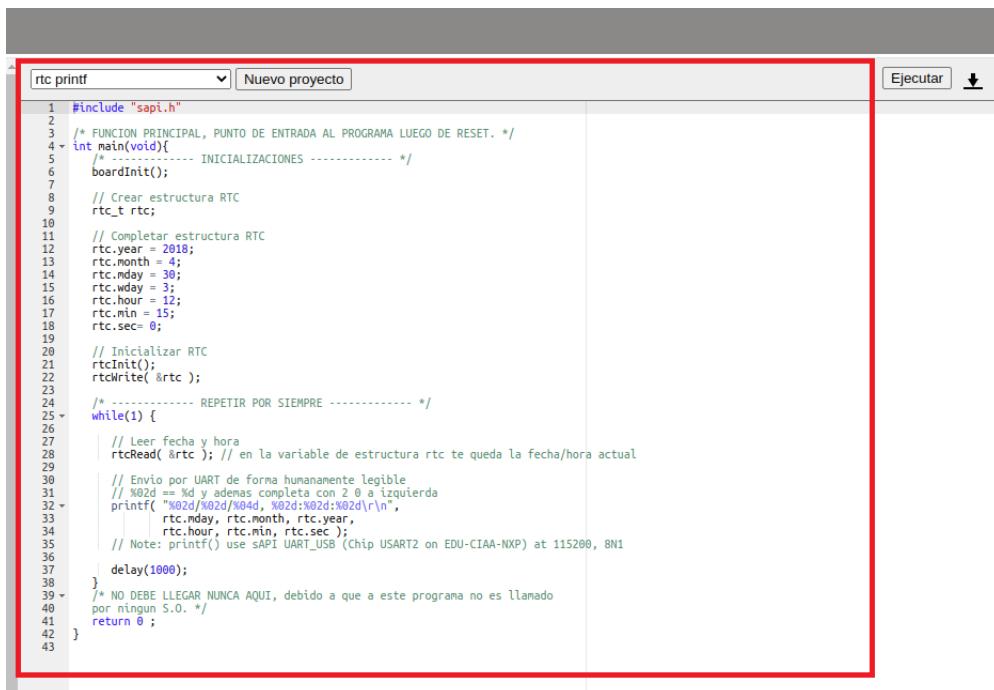
Post condiciones:

- ÉXITO: la plataforma muestra en ejecución el ejemplo *rtc printf* al actualizar la salida por consola del envío por UART.
- FALLA: La plataforma no muestra al usuario ningún cambio en la consola.

Luego de completar el caso de uso CP02, se obtuvo el siguiente resultado:

- ÉXITO: la plataforma muestra en ejecución el ejemplo *rtc printf*.

La figura 4.9 muestra el código que corresponde al ejemplo *rtc printf* y la figura 4.10 muestra en la consola integrada la salida del ensayo.

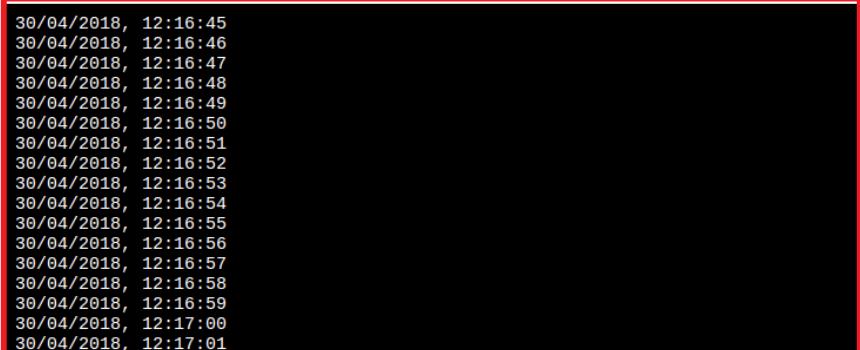


```

rtc printf          Nuevo proyecto
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.mday = 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
27        // Leer fecha y hora
28        rtcRead( &rtc ); // en la variable de estructura rtc te queda la fecha/hora actual
29
30        // Envio por UART de forma humanamente legible
31        // %02d == %d y ademas completa con 2 0 a izquierda
32        printf("%02d/%02d/%04d, %02d:%02d:%02d\r\n",
33               rtc.mday, rtc.month, rtc.year,
34               rtc.hour, rtc.min, rtc.sec);
35        // Note: printf() use SAPI USART_USB (Chip USART2 on EDU-CIAA-NXP) at 115200, 8N1
36
37        delay(1000);
38    }
39    /* NO DEBE LLEGAR NUNCA AQUI, debido a que a este programa no es llamado
40    por ningun S.O. */
41    return 0 ;
42 }
43

```

FIGURA 4.9. Código del ejemplo *rtc printf*.



```

30/04/2018, 12:16:45
30/04/2018, 12:16:46
30/04/2018, 12:16:47
30/04/2018, 12:16:48
30/04/2018, 12:16:49
30/04/2018, 12:16:50
30/04/2018, 12:16:51
30/04/2018, 12:16:52
30/04/2018, 12:16:53
30/04/2018, 12:16:54
30/04/2018, 12:16:55
30/04/2018, 12:16:56
30/04/2018, 12:16:57
30/04/2018, 12:16:58
30/04/2018, 12:16:59
30/04/2018, 12:17:00
30/04/2018, 12:17:01

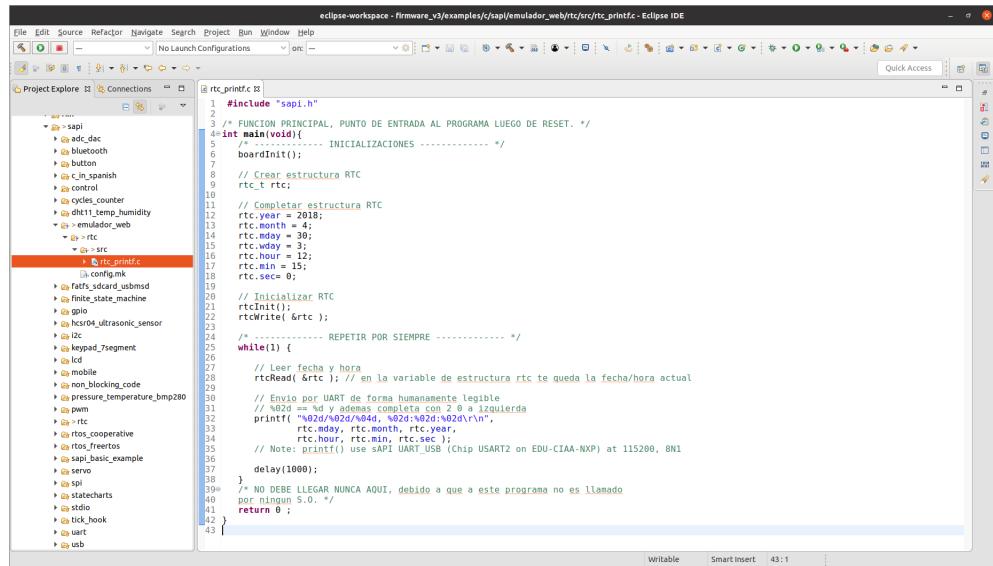
```

FIGURA 4.10. Salida por consola del ensayo *rtc printf*.

Ensayo en la placa EDU-CIAA-NXP

Primeramente, dentro de la herramienta de desarrollo *eclipse* se importa el archivo "rtc_printf.c" generado por la plataforma web, luego, se realiza las configuraciones necesarias para compilar el proyecto.

La figura 4.11 expone el ejemplo *rtc printf* importado en *eclipse*:



```

eclipse-workspace - firmware_v3/examples/c/api/emulador_web/rtc/src/rtc_printf.c - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help
File | Open | Save | New | Recent | No Launch Configurations | on: | 
Project Explorer Connections Quick Access
src
  +-- api
    +-- adc_dac
    +-- bluetooth
    +-- button
    +-- c_in_spanish
    +-- config
    +-- dht1_tamp_humidity
    +-- emulador_web
      +-- rtc
        +-- src
          +-- rtc_printf.c
          +-- config.mk
        +-- config_sdcard_usbm3d
        +-- config_state_machine
        +-- gpios
        +-- hcsr04_ultrasonic_sensor
        +-- i2c
        +-- keypad_7segment
        +-- lcd
        +-- mobile
        +-- non_blocking_code
        +-- pressure_temperature_bmp280
        +-- os
        +-- rtc
        +-- rtos_cooperative
        +-- rtos_freetios
        +-- sapi_basic_example
        +-- servo
        +-- stdio
        +-- tick_hook
        +-- uart
        +-- usb

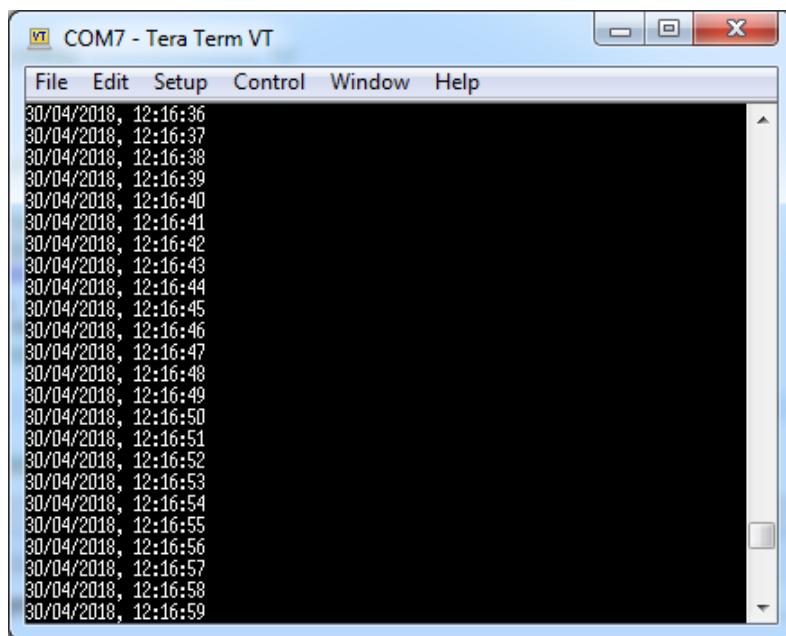
rtc_printf.c
#include "sapi.h"

/* FUNCION PRINCIPAL, PUNTO DE ENTRADA AL PROGRAMA LUEGO DE RESET. */
int main(void){
    /* ----- INICIALIZACIONES ----- */
    BoardInit();
    /* Crear estructura RTC */
    rtt_t rtc;
    /* Completar estructura RTC */
    rtc.year = 2018;
    rtc.month = 4;
    rtc.day = 30;
    rtc.wday = 3;
    rtc.hour = 12;
    rtc.min = 30;
    rtc.sec = 0;
    /* ----- REPETIR POR SIEMPRE ----- */
    while(1) {
        /* Leer fecha y hora */
        rtcRead(&rtc); // en la variable de estructura rtc se queda la fecha/hora actual
        /* Envio por UART de forma humanalemente legible
        // NO DEBE ir y ademas completa con 2 0's a la Izquierda
        printf("%02d/%02d/%04d, %02d:%02d:%02d\n",
               rtc.day, rtc.month, rtc.year,
               rtc.hour, rtc.min, rtc.sec);
        // Note: printf() use sapi USART2 on EDU-CIAA-NXP) at 115200, 8N1
        delay(1000);
    }
    /* NO DEBE LLEGAR NUNCA AQUI, debido a que a este programa no es llamado
    por ningun S.O. */
    return 0;
}

```

FIGURA 4.11. Ejemplo *rtc printf* importado en *eclipse*.

Finalmente, se ejecuta el mismo ejemplo en la placa física EDU-CIAA-NXP y se obtiene por consola la siguiente salida que se muestra en la figura 4.12:



```

COM7 - Tera Term VT
File Edit Setup Control Window Help
30/04/2018, 12:16:36
30/04/2018, 12:16:37
30/04/2018, 12:16:38
30/04/2018, 12:16:39
30/04/2018, 12:16:40
30/04/2018, 12:16:41
30/04/2018, 12:16:42
30/04/2018, 12:16:43
30/04/2018, 12:16:44
30/04/2018, 12:16:45
30/04/2018, 12:16:46
30/04/2018, 12:16:47
30/04/2018, 12:16:48
30/04/2018, 12:16:49
30/04/2018, 12:16:50
30/04/2018, 12:16:51
30/04/2018, 12:16:52
30/04/2018, 12:16:53
30/04/2018, 12:16:54
30/04/2018, 12:16:55
30/04/2018, 12:16:56
30/04/2018, 12:16:57
30/04/2018, 12:16:58
30/04/2018, 12:16:59

```

FIGURA 4.12. Salida de la terminal COM7 -Tera Term VT.

Resumen de la prueba Exitosa:

- Despues de seguir los pasos del flujo principal del caso de prueba, se comprobó que se muestra en ejecución el ejemplo *rtc printf*.

- El ensayo en la placa física EDU-CIAA-NXP con el mismo ejemplo *rtc printf* obtuvo la misma salida por consola de la plataforma web.

4.6.2. Prueba del ejemplo *dht11*

Ensayo en la plataforma de emulación web

Para el ensayo en la plataforma de emulación web se siguió con los pasos del siguiente caso de uso:

ID Caso de prueba: CP03

Descripción: la plataforma de emulación permite al usuario ejecutar el ejemplo *dht11*.

Pre-condición:

- La computadora del usuario tiene conexión a internet y un navegador web instalado.

Flujo principal:

1. El usuario ingresa al entorno web de la plataforma de emulación para la placa EDU-CIAA-NXP.
2. El usuario selecciona desde la lista desplegable jerárquica el ejemplo *dht11*.
3. La plataforma web realiza las siguientes acciones:
 - muestra al usuario el código del ejemplo *dht11* dentro del área de codificación.
 - carga el periférico virtual de manera automática dentro del área de ensamblado y muestra las conexiones a los pines configurados por defecto, los cuales son: "SIGNAL=GPIO1", "SDA/SDI=GND" y "VCC=3V3".
 - muestra seleccionado la opción por defecto "Obtener datos de servidor climático local."
 - actualiza los termómetros gráficos que corresponden a la temperatura y a la humedad con los datos obtenidos del servidor climático.
4. El usuario hace click en el botón "Ejecutar".
5. La plataforma muestra en la consola integrada la salida de la temperatura y humedad con los valores enviados desde la central meteorológica en línea.
6. El usuario realiza la descarga del ejemplo *dht11* al hacer click en el botón de descarga, ubicado en el área de codificación.

Flujo alternativo:

4. El usuario hace clic en la opción "Establecer Temperatura y Humedad manualmente(haga click y arrastre sobre los indicadores)."
5. La plataforma deselecciona la opción "Obtener datos de servidor climático local."
6. El usuario comienza a manipular los termómetros gráficos haciendo clic sobre los que corresponden a la temperatura y a la humedad.

7. La plataforma actualiza los termómetros gráficos que corresponden a la temperatura y a la humedad con los datos generados por el usuario.
8. El usuario hace click en el botón "Ejecutar".
9. La plataforma muestra en la consola integrada la salida de la temperatura y humedad según la selección de obtención de datos.
10. El usuario realiza la descarga del ejemplo **dht11** al hacer click en el botón de descarga, ubicado en el área de codificación.

Post condiciones:

- ÉXITO: la plataforma debe cumplir con todas las siguientes condiciones de éxito:
 - carga automáticamente el periférico virtual dentro del área de ensamblado y las conexiones configuradas por defecto.
 - actualiza los termómetros gráficos de temperatura y humedad, que corresponden a la elección de obtención de datos.
 - muestra en ejecución el ejemplo **dht11** y según la elección de obtención de datos, actualiza la salida de temperatura y humedad en la consola integrada.
- FALLA: la plataforma no cumple con todas las condiciones de éxito descritas.

Luego de completar el caso de uso CP02 con: flujo principal y flujo alternativo, se obtuvo el siguiente resultado:

- ÉXITO: la plataforma cumple con todas las condiciones de éxito para el ejemplo **dht11**.

La figura 4.14 muestra la consola con los datos de temperatura y humedad del ejemplo **dht11** con la opción "Establecer Temperatura y Humedad manualmente(haga click y arrastre sobre los indicadores)." "

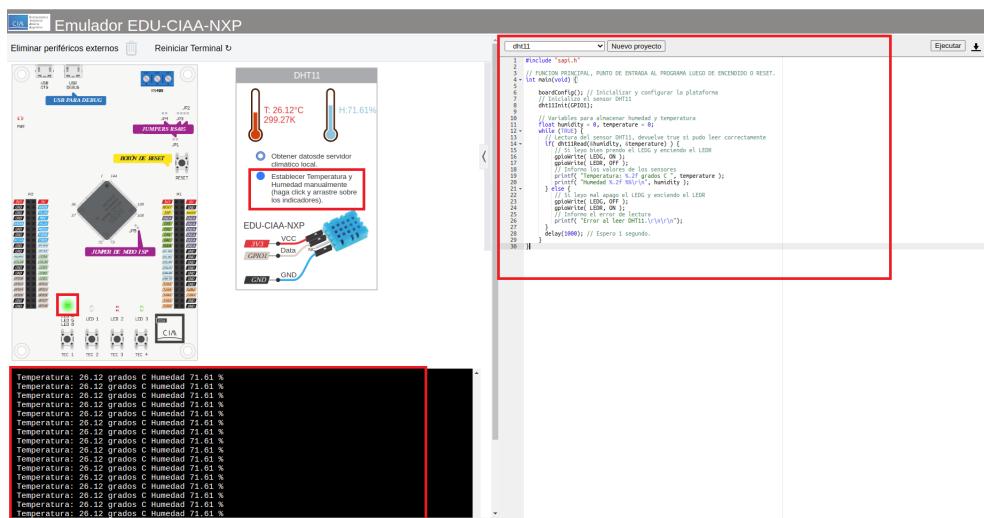


FIGURA 4.13. Resultado del CP02 con la opción "Establecer Temperatura y Humedad manualmente(haga click y arrastre sobre los indicadores)." "

La figura 4.14 muestra la consola con los datos de temperatura y humedad del ejemplo *dht11* con la opción "Obtener datos de servidor climático local."

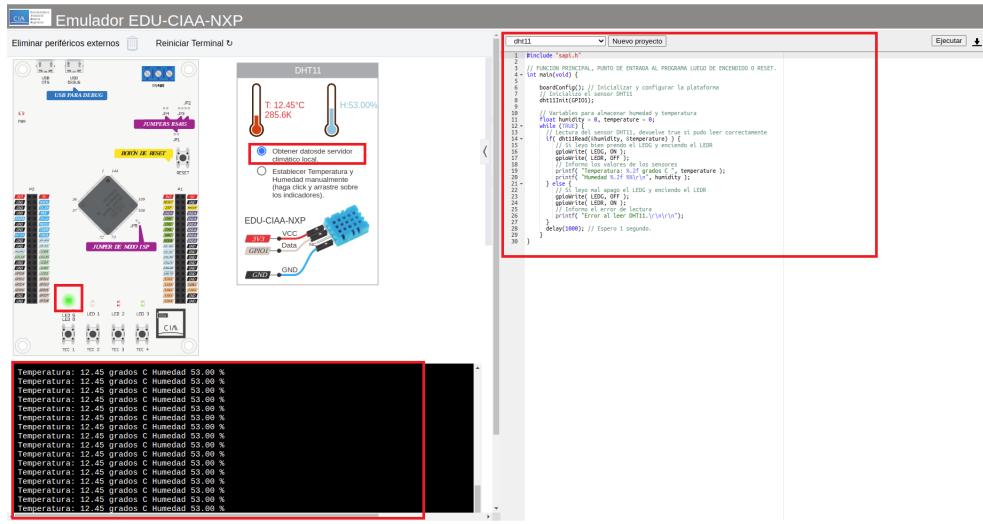


FIGURA 4.14. Resultado del CP02 con la opción "Obtener datos de servidor climático local."

Para verificar que la plataforma obtuvo los datos de temperatura/humedad de la 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.

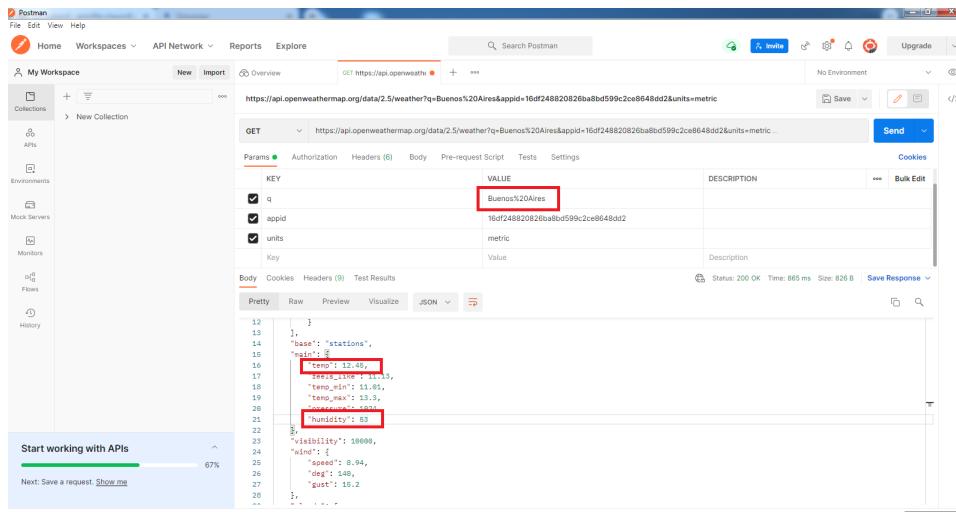


FIGURA 4.15. Petición de datos de temperatura/humedad.

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.

Ensayo en la placa EDU-CIAA-NXP

Este ensayo tuvo como objetivo identificar las diferencias entre los resultados reales producidos por la placa física EDU-CIAA-NXP 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 importar el archivo generado por la plataforma web "dht11_temp_humidity.c" y compilarlo dentro de la herramienta *eclipse*. Luego, se grabó en la placa física EDU-CIAA-NXP.

A continuación se muestra en la figura 4.16 la ejecución del ensayo en la plataforma EDU-CIAA-NXP.

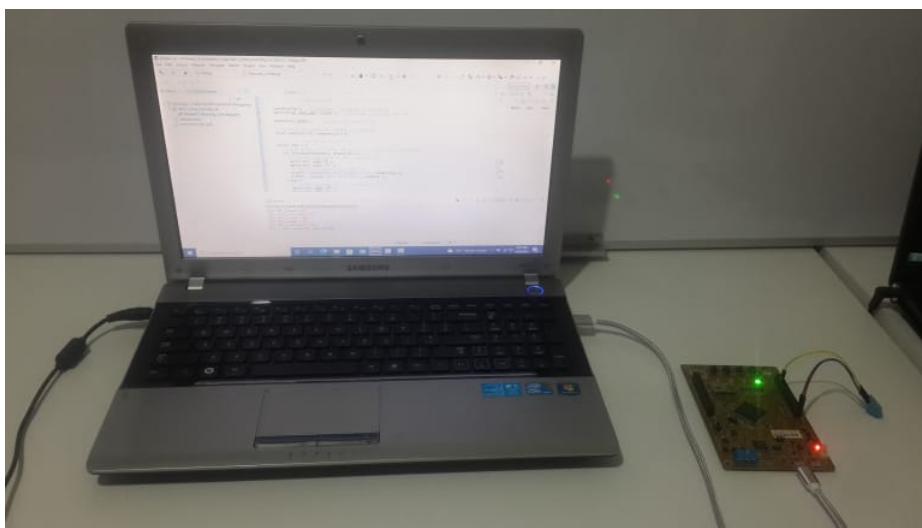


FIGURA 4.16. Ensayo en la plataforma EDU-CIAA-NXP del ejemplo *dht11*.

La figura 4.17 muestra el código del ejemplo *dht11* en la herramienta *eclipse* de la PC de prueba.

 A screenshot of the Eclipse IDE interface. The central workspace shows a code editor with the file "dht11_temp_humidity.c" open. The code is written in C and includes comments explaining the functionality, such as initializing the platform and reading DHT11 sensor data. The left side of the interface shows the "Project Explorer" view, which lists various projects and files, including "dht11_temp_humidity.c" under the "emulador_web" project.


```

eclipse-workspace - firmware_v3/examples/c/api/emulador_web/dht11/src/dht11_temp_humidity.c - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer Connections
dht11_temp_humidity.c
1 #include "sapi.h"
2
3 // FUNCION PRINCIPAL, PUNTO DE ENTRADA AL PROGRAMA LUEGO DE ENCENDIDO O RESET.
4 int main(void) {
5
6     boardConfig(); // Inicializar y configurar la plataforma
7     // Inicializo el sensor DHT11
8     dht11Init(GPIO1);
9
10    // Variables para almacenar humedad y temperatura
11    float humidity = 0, temperature = 0;
12    while (TRUE) {
13        // Lectura del sensor DHT11, devuelve true si pudo leer correctamente
14        if( dht11Read(&humidity, &temperature) ) {
15            // Si leyo bien prendo el LEDG y enciendo el LEDR
16            gpioWrite( LEDG, ON );
17            gpioWrite( LEDR, OFF );
18            // Informo los valores de los sensores
19            printf( "Temperatura: %.2f grados C ", temperature );
20            printf( "Humedad: %.2f %%\r\n", humidity );
21        } else {
22            // Si leyo mal apago el LEDG y enciendo el LEDR
23            gpioWrite( LEDG, OFF );
24            gpioWrite( LEDR, ON );
25            // Informo el error de lectura
26            printf( "Error al leer DHT11.\r\n\r\n" );
27        }
28        delay(1000); // Espero 1 segundo.
29    }
30}
31
  
```

FIGURA 4.17. Código del ejemplo en *eclipse*.

El programa *dht11* 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 EDU-CIAA-NXP y también, los mensajes de la terminal serie. De modo que, posteriormente, permitió compararlos con la plataforma de emulación.

En la figura 4.18 se observan los cambios en la placa EDU-CIAA-NXP que fueron registrados durante las pruebas.



FIGURA 4.18. 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 4.19 se muestra los datos de temperatura/humedad usando *Tera Term VT*.

The screenshot shows the Tera Term VT application window titled "COM7 - Tera Term VT". The menu bar includes File, Edit, Setup, Control, Window, and Help. The main window displays a series of temperature and humidity readings from the serial port:

```
Humedad: 41.00
Temperatura: 24.00 grados C.
Humedad: 41.00
Temperatura: 24.00 grados C.
Humedad: 41.00
Temperatura: 20.00 grados C.
Humedad: 39.00
Temperatura: 24.00 grados C.
Humedad: 41.00
Temperatura: 24.00 grados C.
Humedad: 41.00
Temperatura: 24.00 grados C.
Humedad: 41.00
Temperatura: 24.00 grados C.
```

FIGURA 4.19. Salida de la terminal COM7 -Tera Term VT.

Resumen de la prueba Exitosa:

- Después de acceder a la plataforma mediante el navegador y siguiendo los pasos del flujo principal y el flujo alternativo de la prueba, se comprobó que se muestra en ejecución el ejemplo *Dht11 temperature/humidity*.
- 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.
- Se ensayo en la placa física EDU-CIAA-NXP el mismo ejemplo *Dht11 temperature/humidity* y se registraron los resultados de la terminal serial.

4.6.3. Prueba de un nuevo proyecto *tick hook*

El objetivo de este ensayo es exponer las diferencias en los resultados encontrados entre el emulador web y la placa física EDU-CIAA-NXP.

Ensayo en la plataforma de emulación web

Para ensayar la plataforma web, se ejecuto el siguiente caso de prueba:

ID Caso de prueba: CP04

Descripción: la plataforma de emulación permite al usuario ejecutar un nuevo ejemplo de *tick hook*.

Pre-condición:

- La computadora del usuario tiene conexión a internet y un navegador web instalado.

Flujo principal:

1. El usuario ingresa al entorno web de la plataforma de emulación para la placa EDU-CIAA-NXP.
2. El usuario hace click en el botón "Nuevo Proyecto".
3. La plataforma muestra al usuario en el área de codificacion, la pantalla para que pueda ingresar su propio codigo.
4. El usuario ingresa el siguiente código de prueba:

```

1 #include "sapi.h"
2
3 void myTickHook( void *ptr )
4 {
5     gpioWrite( LED3, ON );
6     printf( "Blinky LED3.\r\n" );
7     while(TRUE) {
8         printf( " while(TRUE) Blinky LED1.\r\n" );
9         gpioToggle( LED1 );
10        delay(1);
11    }
12 }
13
14 int main()
15 {
16     boardConfig();

```

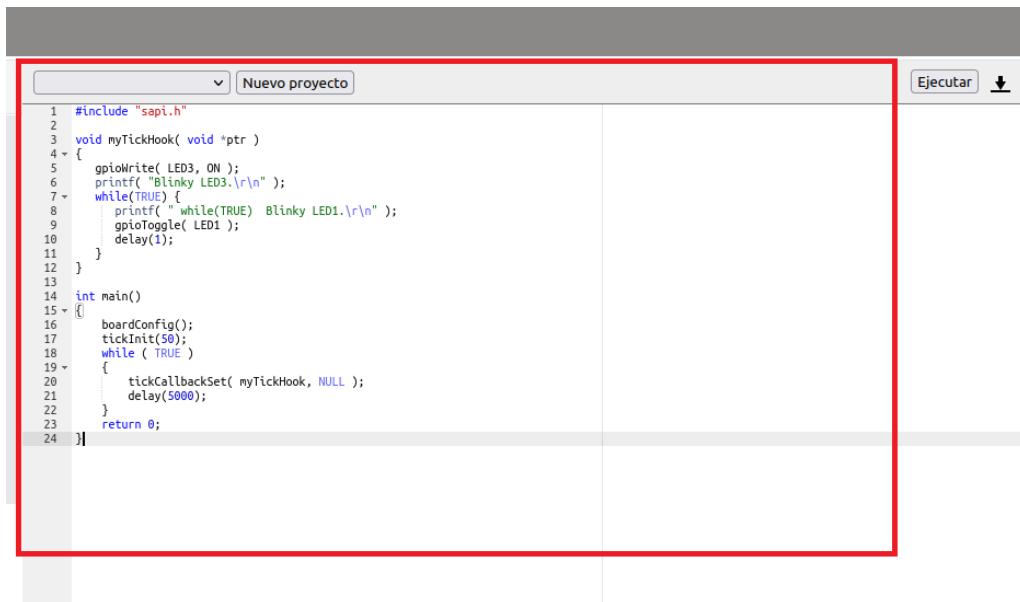
```

17     tickInit(50);
18     while ( TRUE )
19     {
20         tickCallbackSet( myTickHook, NULL );
21         delay(5000);
22     }
23     return 0;
24 }
```

CÓDIGO 4.1. nuevo proyecto

5. El usuario hace click en el botón "Ejecutar".
6. La plataforma muestra en la consola integrada la salida del programa en ejecución.

La figura 4.20 presenta el código de usuario en el área de codificación y la figura 4.21 muestra en la consola integrada la salida de "Blinky LED3"muchas veces.



The screenshot shows a software interface with a top menu bar and two main windows. The left window, which has a red border, contains the C code for the 'tick hook' project. The right window is currently empty and represents the integrated terminal or console where the program's output would be displayed.

```

1 #include "sapi.h"
2
3 void myTickHook( void *ptr )
4 {
5     gpioWrite( LED3, ON );
6     printf( "Blinky LED3.\r\n" );
7     while(TRUE) {
8         printf( " while(TRUE) Blinky LED1.\r\n" );
9         gpioToggle( LED1 );
10        delay(1);
11    }
12 }
13
14 int main()
15 {
16     boardConfig();
17     tickInit(50);
18     while ( TRUE )
19     {
20         tickCallbackSet( myTickHook, NULL );
21         delay(5000);
22     }
23     return 0;
24 }
```

FIGURA 4.20. Código del ensayo del nuevo proyecto tick hook.



The screenshot shows a terminal window with a red border, displaying the output of the 'Blinky LED3' program. The output consists of many lines of text, each starting with 'Blinky LED3.' followed by 'Blinky LED1.' repeated several times. This indicates that the program is running and printing its intended message to the console.

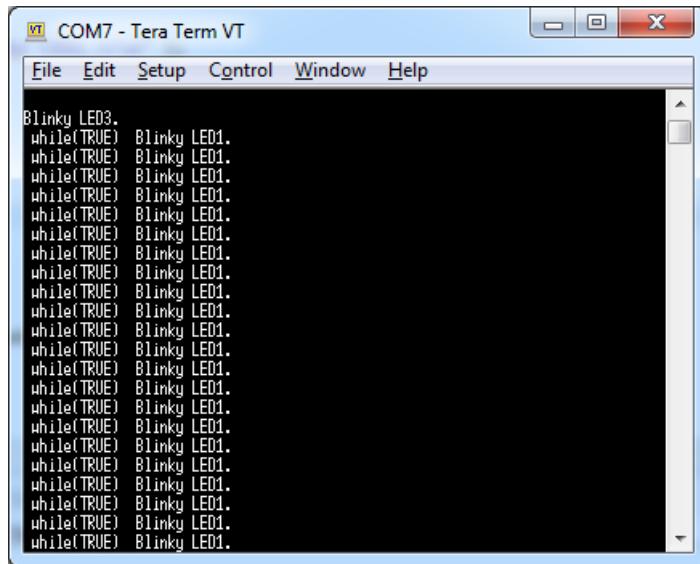
```

Blinky LED3.
while(TRUE) Blinky LED1.
Blinky LED3.
while(TRUE) Blinky LED1.
while(TRUE) Blinky LED1.
while(TRUE) Blinky LED1.
while(TRUE) Blinky LED1.
```

FIGURA 4.21. Salida por consola del ensayo tick hook.

Ensayo en la placa EDU-CIAA-NXP

Se ejecuta el mismo ejemplo en la placa física EDU-CIAA-NXP y se obtiene por consola la siguiente salida que se muestra en la siguiente figura 4.22:



The screenshot shows a Windows application window titled "COM7 - Tera Term VT". The menu bar includes File, Edit, Setup, Control, Window, and Help. The main text area displays the following repeating text:
Blinky LED3.
while(TRUE) Blinky LED1.
while(TRUE) Blinky LED1.

FIGURA 4.22. Salida de la terminal COM7 -Tera Term VT.

Se observa que la salida por consola de "Blinky LED3" se muestra solo una vez, lo cual difiere de la salida por consola de la plataforma web que muestra "Blinky LED3" muchas veces.

La diferencia de este comportamiento en el emulador web esta relacionada con la forma en cómo JavaScript reanuda la ejecución después de que ocurre un tick. Por lo tanto, la forma en que se gestionan los estados en la ejecución de funciones puede diferir del comportamiento en la placa EDU-CIAA-NXP.

En un entorno físico, el sistema operativo o el microcontrolador pueden mantener un seguimiento del estado de la ejecución y reanudarla adecuadamente después de una interrupción o tick del sistema. Sin embargo, para esta prueba en particular, dentro de la plataforma de emulación web, no se mantiene este estado de ejecución, lo que resultó la repetición de ciertas porciones de código, como se probó anteriormente.

Para solucionar esta diferencia en el entorno del emulador web, se podría explorar algunas opciones para mantener un seguimiento adecuado del estado de ejecución después de que ocurra un tick del sistema.

4.6.4. Periféricos implementados en *Mbed Simulator* y en el Emulador EDU-CIAA-NXP

A continuación, se presenta los periféricos internos y externos que están disponibles para la simulación en *Mbed Simulator*. La tabla 4.3 expone los periféricos internos implementados en *Mbed Simulator*, ademas de realizar una comparacion si responde a eventos, velocidad y tiempo real.

TABLA 4.3. Comparación de características de Periféricos internos implementados en *Mbed Simulator*

Periféricos	Velocidad y tiempo real	Responde a eventos
GPIO	Si	Si
UART	Si	Si
BUTTON	SI	Si
ADC	Si	Si
DAC	Si	No
PWM	Si	No

En la figura 4.23 se muestra el funcionamiento de GPIO y la figura 4.24 expone el funcionamiento de PWM del *Mbed Simulator*.

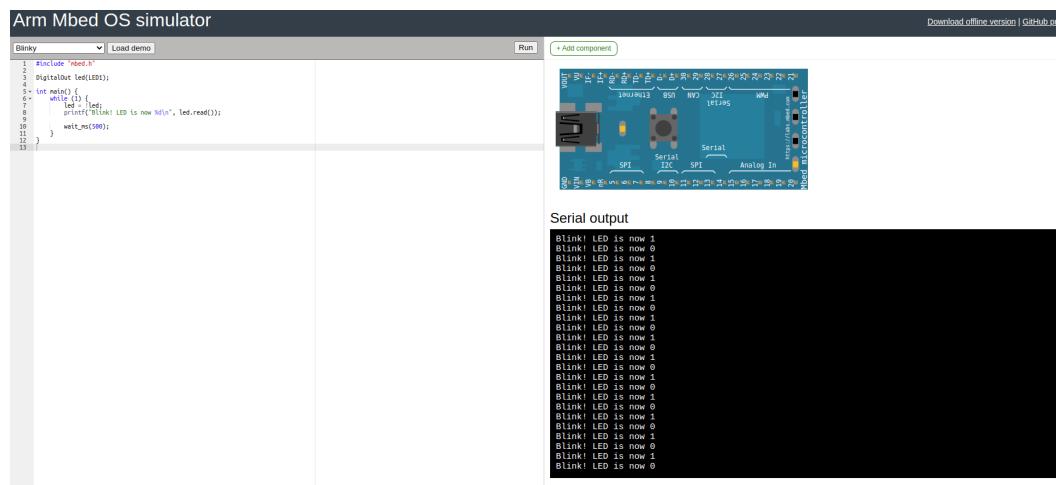
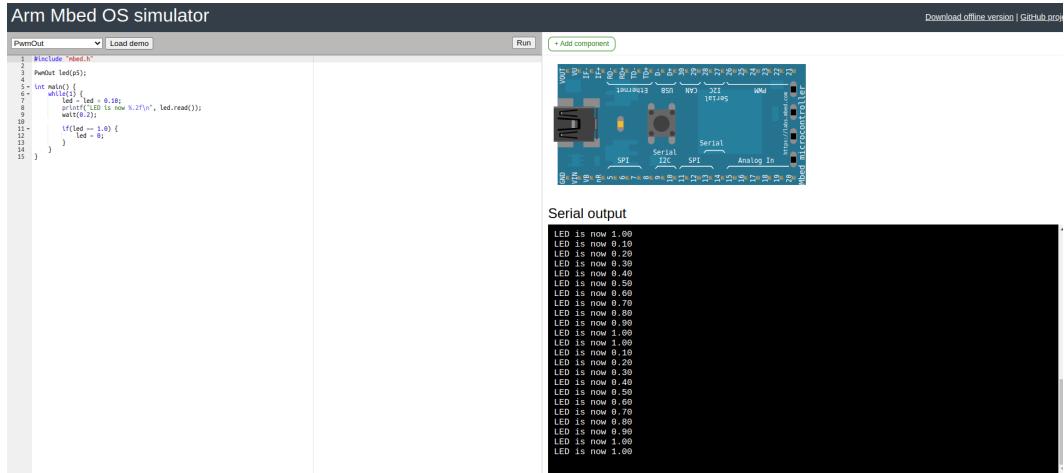


FIGURA 4.23. Funcionamiento GPIO en *Mbed Simulator*.

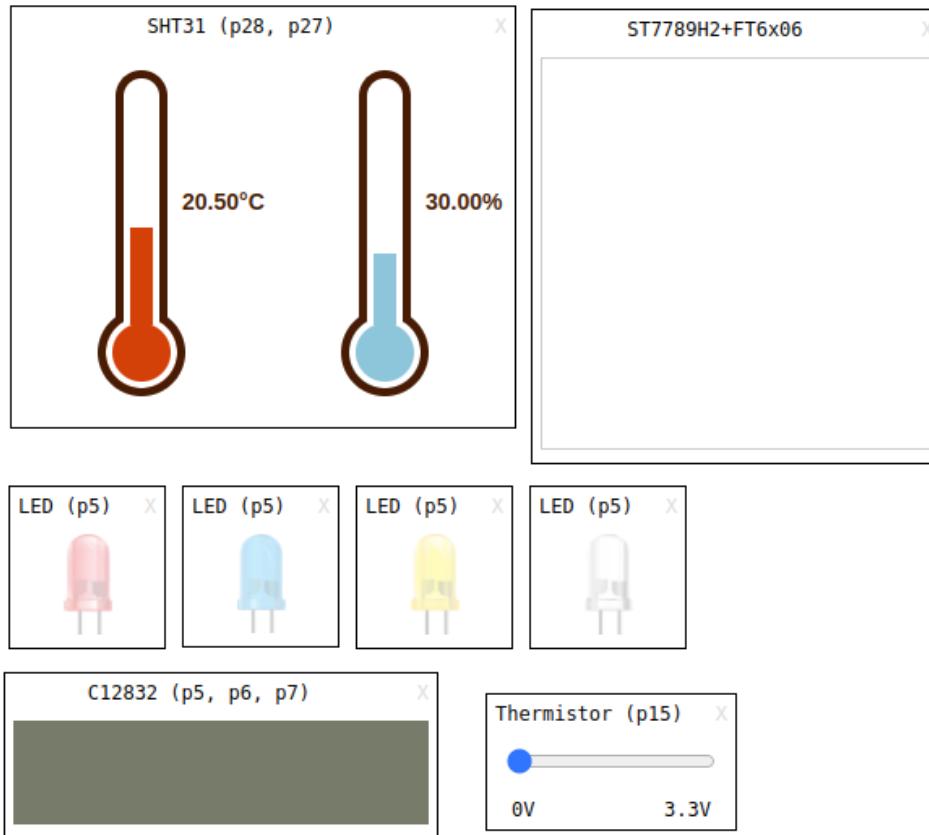
FIGURA 4.24. Funcionamiento PWM en *Mbed Simulator*.

Además, en la tabla 4.4 expone los periféricos externos implementados actualmente en *Mbed Simulator*.

TABLA 4.4. Comparación de características de Periféricos externos implementados en *Mbed Simulator*

Periféricos	Velocidad y tiempo	Responde a eventos real
DHT11	Si	Si
LCD Display C12832	Si	Si
LED	Si	Si
Thermistor	Si	Si
SHT31	Si	Si
Touch Screen ST7789H2	Si	Si

La figura 4.25 muestra los periféricos externos de *Mbed Simulator*.

FIGURA 4.25. Periféricos externos de *Mbed Simulator*.

También, se presenta los periféricos internos y externos implementados en la primera versión de la plataforma de emulación web de la placa EDU-CIAA. La tabla 4.5 expone los periféricos internos disponibles en el entorno web y expone una comparación si responde a eventos, velocidad y tiempo real.

TABLA 4.5. Comparación de características de Periféricos

Periféricos	Velocidad y tiempo	Responde a eventos real
GPIO	Si	Si
UART	Si	Si
BUTTON	SI	Si
RTC	Si	Si
SYSTICK	No	Si
ADC	Si	Si
DAC	Si	No

Además, la siguiente tabla 4.6 se exponen los periféricos externos implementados en la plataforma de emulación web.

TABLA 4.6. Comparación de características de los periféricos externos del Emulador EDU-CIAA

Periféricos	Velocidad y tiempo	Responde a eventos real
DHT11	Si	Si
LCD	Si	Si
LED	Si	Si
LCD DISPLAY 128x64	Si	Si
LCD DISPLAY 20x4	Si	Si
Thermistor NTC	Si	Si
Potentiometer	Si	Si
Joystick	Si	Si

La figura 4.26 muestra los dibujos en formato SVG de los periféricos externos y disponibles en el emulador web EDU-CIAA.

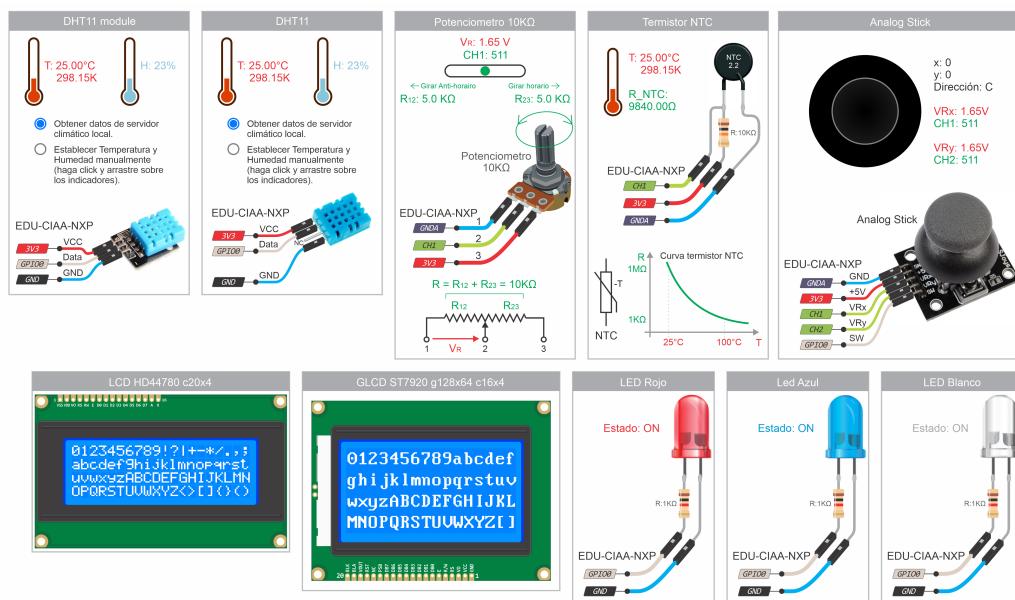


FIGURA 4.26. Periféricos externos del emulador web EDU-CIAA.

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.
- Implementación de periféricos externos virtuales de la placa EDU-CIAA-NXP en formato SVG.
- 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] 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.
- [3] Proyecto CIAA. *Computadora Industrial Abierta Argentina*. Visitado el 2023-06-15. 2014. URL: <http://proyecto-ciaa.com.ar/devwiki/doku.php?id=start>.
- [4] Jenny Chavez. *Trabajo Final CESE*. Visitado el 2023-09-27. 2023. URL: <https://lse-posgrados-files.fi.uba.ar/tesis/LSE-FIUBA-Trabajo-Final-CESE-Jenny-Chavez-2018.pdf>.
- [5] Comunidad Proyecto CIAA. *Firmware v3*. Visitado el 2023-07-29. 2018. URL: https://github.com/ciaa/firmware_v3.
- [6] GitHub. *Sistema de Control de Versiones de código fuente distribuído*. Visitado el 2023-09-24. 2023. URL: <https://docs.github.com/es/get-started/using-git/about-git#about-version-control-and-git>.
- [7] GitHub. *Integración Continua*. Visitado el 2023-09-24. 2023. URL: <https://docs.github.com/es/actions/automating-builds-and-tests/about-continuous-integration>.
- [8] Agustín 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>.
- [9] ACheng Zhao. *Electron*. Visitado el 2023-06-28. 2013. URL: <https://www.electronjs.org/>.
- [10] Arduino LLC. *Arduino*. Visitado el 2022-03-14. 2022. URL: <https://arduino.cl/>.
- [11] Stan Simmons. *UnoArduSim*. Visitado el 2022-03-15. 2022. URL: <https://sites.google.com/site/unoardusim/home>.
- [12] Queen's University. *Queen's University*. Visitado el 2022-03-15. 2022. URL: <https://www.queensu.ca/>.
- [13] Arduino. *Arduino Uno*. Visitado el 2022-03-14. 2022. URL: <https://arduino.cl/arduino-uno/>.
- [14] grupo de startups. *Virtronics*. Visitado el 2022-03-15. 2022. URL: <http://www.virtronics.com.au/simulator-for-arduino.html>.
- [15] Stan Simmons. *Tinkercad*. Visitado el 2022-03-15. 2022. URL: <https://www.tinkercad.com/dashboard>.

- [16] John Walker. *Autodesk*. Visitado el 2022-03-14. 2022. URL: <https://www.autodesk.com>.
- [17] Mbed Labs. *Mbed Simulator*. Visitado el 2022-03-15. 2022. URL: <https://os.mbed.com/blog/entry/introducing-mbed-simulator>.
- [18] ARM. *Arm Mbed Os*. Visitado el 2022-03-15. 2022. URL: <https://www.arm.com/products/development-tools/embedded-and-software/mbed-os>.
- [19] ARMmbed/mbed-simulator. *Github Repositorio*. Visitado el 2023-09-15. 2023. URL: <https://github.com/armmbed/mbed-simulator>.
- [20] Wikipedia. *Navegador web*. Visitado el 2022-03-13. 2022. URL: https://es.wikipedia.org/wiki/Navegador_web.
- [21] LLVM Developer Group. *NodeJS*. Visitado el 2022-03-13. 2022. URL: <https://xtermjs.org/>.
- [22] LLVM Developer Group. *JavaScript*. Visitado el 2022-03-13. 2022. URL: <https://es.wikipedia.org/wiki/JavaScript>.
- [23] Open Source License. *Emscripten*. Visitado el 2022-03-13. 2022. URL: <https://emscripten.org/>.
- [24] LLVM Developer Group. *Low Level Virtual Machine*. Visitado el 2022-03-13. 2022. URL: <https://es.wikipedia.org/wiki/LLVM>.
- [25] Apache. *Binaryen*. Visitado el 2022-03-14. 2022. URL: <https://www.npmjs.com/package/binaryen>.
- [26] WebAssembly Working Group. *WebAssembly*. Visitado el 2022-03-14. 2022. URL: <https://webassembly.org/>.
- [27] ARM. *ARM Mbed OS*. Visitado el 2022-03-14. 2014. URL: <https://github.com/ARMmbed/mbed-os>.
- [28] Arm. *Mbed CLI*. Visitado el 2020-03-14. 2014. URL: <https://github.com/ARMmbed/mbed-cli>.
- [29] W3C. *Document Object Model*. Visitado el 2022-03-14. 2022. URL: <https://www.w3.org/TR/WD-DOM/introduction.html>.
- [30] Guillermo Rauch. *Socket.IO*. Visitado el 2022-03-14. 2022. URL: <https://socket.io/>.
- [31] LLVM Developer Group. *Xterm*. Visitado el 2022-03-13. 2022. URL: <https://xtermjs.org/>.
- [32] Microsoft. *TypeScript*. Visitado el 2022-03-14. 2022. URL: <https://www.typescriptlang.org/>.
- [33] Express Working Group. *Express*. Visitado el 2022-03-14. 2022. URL: <https://expressjs.com/es/>.
- [34] informática. *Application Programming Interface*. Visitado el 2022-03-13. 2022. URL: <https://definicion.de/api/>.
- [35] Web Hypertext Application Technology Working Group. *Lenguaje de Marcas de Hipertexto*. Visitado el 2022-03-14. 2022. URL: <https://es.wikipedia.org/wiki/HTML>.
- [36] LLVM Developer Group. *SVG*. Visitado el 2022-03-13. 2022. URL: https://es.wikipedia.org/wiki/Gr%C3%A1ficos_vectoriales_escalables.
- [37] World Wide Web Consortium. *eXtensible Markup Language*. Visitado el 2022-03-14. 2022. URL: https://es.wikipedia.org/wiki/Extensible_Markup_Language.
- [38] CSS Working Group. *Cascading Style Sheets*. Visitado el 2022-03-14. 2022. URL: <https://es.wikipedia.org/wiki/CSS>.
- [39] Open Source License. *Python*. Visitado el 2022-03-13. 2022. URL: <https://emscripten.org/>.

- [40] 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)).
- [41] Arm Limited. *Mbed events*. Visitado el 2023-09-24. 2023. URL: <https://github.com/ARMmbed/mbed-os/blob/master/events/README.md>.
- [42] Microsoft. *Visual Studio Code*. Visitado el 2022-03-15. 2022. URL: <https://code.visualstudio.com/>.
- [43] Microsoft. *Integrated Development Environment*. Visitado el 2022-03-14. 2022. URL: https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado.
- [44] Comunidad Inkscape. *Inkscape*. Visitado el 2023-07-29. 2023. URL: <https://inkscape.org/>.
- [45] Microsoft. *GitHub*. Visitado el 2022-03-15. 2022. URL: <https://github.com/>.
- [46] Josh Kalderimis. *Travis CI*. Visitado el 2023-06-15. 2011. URL: <https://www.travis-ci.com/>.
- [47] Dmitriy Zaporozhets y Valery Sizov. *Gitlab*. Visitado el 2023-06-15. 2011. URL: <https://gitlab.com/>.
- [48] Ben Uretsky, Moisey Uretsky, Mitch Wainer, Jeff Carr y Alec Hartman. *DigitalOcean*. Visitado el 2023-06-15. 2011. URL: <https://www.digitalocean.com/>.
- [49] 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.
- [50] Mocha Working Group. *Mocha*. Visitado el 2022-03-14. 2022. URL: <https://mochajs.org/>.
- [51] Chai Working Group. *Chai*. Visitado el 2022-03-14. 2022. URL: <https://www.chaijs.com/>.
- [52] Roman Zimbelmann. *Check*. Visitado el 2023-06-14. 2000. URL: <https://libcheck.github.io/check/>.
- [53] Andreas Schneider y Jan Kratochvil. *CMocka*. Visitado el 2022-03-14. 2009. URL: <https://cmocka.org/>.
- [54] Comunidad Proyecto CIAA. *sAPI proyecto CIAA*. Visitado el 2022-03-14. 2022. URL: https://github.com/epernia/firmware_v3/blob/master/libs/sapi/documentation/api_reference_es.md.
- [55] Comunidad Proyecto CIAA. *Firmware v2*. Visitado el 2023-07-29. 2018. URL: https://github.com/ciaa/firmware_v2.
- [56] Comunidad Proyecto CIAA. *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.
- [57] 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>.
- [58] Jenny Chavez. *Emulador EDU-CIAA*. Visitado el 2023-06-24. 2023. URL: <https://github.com/jennifferch/ciaa-emulador>.
- [59] Postman Working Group. *Postman*. Visitado el 2022-03-24. 2022. URL: <https://www.postman.com>.
- [60] T. Teranishi. *Tera Term VT*. Visitado el 2022-03-24. 2022. URL: <svn.osdn.jp/svnroot/ttssh2/trunk/>.