



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. Descripción funcional básica	12
2.3.3. <i>Frontend</i> y <i>Backend</i>	12
2.3.4. Descripción funcional en detalle	15
2.3.5. Caso de estudio: programa que activa un LED en la placa virtual	18
2.3.6. Análisis estructural de archivos y carpetas	20
3. Desarrollo e implementación	25
3.1. Herramientas de desarrollo	25
3.1.1. Plataforma de desarrollo EDU-CIAA-NXP	25
3.1.2. Editor <i>VSCode</i>	25
3.1.3. Editor de gráficos vectoriales <i>Inkscape</i>	25
3.1.4. Sistemas de control de versiones e integración continua	26
3.1.5. Herramientas de <i>Testing</i>	26
3.1.6. Servidor en la nube <i>DigitalOcean</i>	27
3.2. Restructuración de archivos y carpetas	27
3.3. <i>Frontend</i> : rediseño de la Interfaz de Usuario	28
3.3.1. Área de ensamblado	29
3.3.2. Área de codificación	31
3.3.3. Área de consola integrada	33
3.4. <i>Backend</i> : bibliotecas de C portadas	34
3.5. Biblioteca <i>sAPI</i>	34
3.5.1. <i>sapi_gpio</i>	36
3.5.2. <i>sapi_adc</i>	38
3.5.3. <i>sapi_tick</i>	40

3.5.4. <i>sapi_delay</i>	42
3.5.5. <i>sapi_dht11</i>	43
3.6. <i>freeRTOS</i>	44
3.7. Comparación de periféricos implementados	45
3.8. Ejemplos incluidos	48
3.9. Despliegue en un servidor web	49
4. Ensayos y resultados	51
4.1. Banco de pruebas	51
4.2. Pruebas de Unidad	52
4.3. Pruebas de Integración	53
4.4. Pruebas de Interfaz de usuario	54
4.5. Integración Continua	55
4.5.1. Prueba de acceso	57
4.6. Pruebas de funcionamiento	59
4.6.1. Prueba del ejemplo <i>rtc_printf</i>	59
Ensayo en <i>EmuCIAA</i>	59
Ensayo en la placa física EDU-CIAA-NXP	60
4.6.2. Prueba del ejemplo <i>dht11</i>	62
Ensayo en <i>EmuCIAA</i>	62
Ensayo en la placa física EDU-CIAA-NXP	64
4.6.3. Prueba de creación de un nuevo proyecto, utilizando el código del ejemplo <i>tick_hook</i>	67
Ensayo en <i>EmuCIAA</i>	67
Ensayo en la placa física EDU-CIAA-NXP	68
5. Conclusiones	71
5.1. Objetivos alcanzados	71
5.2. Próximos pasos	72
Bibliografía	73

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. Modelo de <i>publicación/suscripción</i>	17
2.12. Diagrama de bloques de los oyentes de <i>EventEmitter</i> en la capa UI.	18
2.13. Funcionamiento de <i>Arm Mbed OS Simulator</i>	19
2.14. Activación de evento con el nombre <code>pin_write</code>	20
2.15. GPIO oyente del evento con el nombre <code>pin_write</code>	20
2.16. Interacción entre todas las capas de programación.	20
2.17. Estructura de carpetas y archivos de <i>Arm Mbed OS Simulator</i>	21
3.1. Estructura de carpetas y archivos de <i>EmuCIAA</i>	27
3.2. Interfaz gráfica de <i>EmuCIAA</i>	28
3.3. El usuario puede elegir un componente en la aplicación.	30
3.4. Periférico agregado en el área de ensamblado.	30
3.5. Código que genera errores de compilación.	31
3.6. Errores de compilación.	32
3.7. Estructura jerárquica de ejemplos.	32
3.8. Carga automática del periférico.	32
3.9. Programa de usuario que imprime por consola.	33
3.10. Salida de la terminal serie.	33
3.11. Dependencias del módulo <i>GPIO</i> de la <i>sAPI</i>	36
3.12. Dependencias del módulo <i>GPIO</i> de <i>EmuCIAA</i>	37
3.13. Diagrama de bloques de <i>C HAL</i> y <i>JS HAL</i> para el módulo <i>adc</i>	38
3.14. Diagrama de bloques con la interacción entre las capas <i>JS HAL</i> y <i>JS UI</i>	40
3.15. Diagrama de bloques <code>EMSCRIPTEN_KEEPALIVE</code>	41
3.16. Diagrama de bloques de la función <code>ccall</code>	41
3.17. Diagrama de bloques <code>emscripten_sleep</code>	42
3.18. Diagrama de bloques de la macro <code>EM_ASM_INT</code>	43
3.19. Diagrama de bloques de la capa <i>JavaScript UI</i> con las dos opciones para el usuario.	44
3.20. Periféricos externos de <i>Arm Mbed OS Simulator</i>	46

3.21. Periféricos externos del emulador web EDU-CIAA.	47
4.1. Primera parte de la salida por consola de las pruebas unitarias con <i>CMocka</i> o <i>Check</i>	53
4.2. Segunda parte de la salida por consola de las pruebas unitarias con <i>CMocka</i> o <i>Check</i>	53
4.3. Depuración de las pruebas de integración.	54
4.4. Salida por consola durante la depuración de las pruebas de interfaz con <i>Mocha</i>	55
4.5. Información de las pruebas que se ejecutaron en <i>Travis CI</i>	56
4.6. Información de las pruebas que se ejecutaron en <i>GitLab CI</i>	57
4.7. Prueba plataforma emulador ejecutando el ejemplo <i>Blinky</i>	58
4.8. Respuesta del servidor.	59
4.9. Salida por consola del ensayo <i>rtc printf</i>	60
4.10. Ejemplo <i>rtc printf</i> importado en <i>eclipse</i>	61
4.11. Salida de la terminal COM7 -Tera Term VT.	61
4.12. Resultado del CP02 con la opción "Obtener datos de servidor climático local."	63
4.13. Resultado del CP02 con la opción "Establecer Temperatura y Humedad manualmente(haga click y arrastre sobre los indicadores)." .	64
4.14. Petición de datos de temperatura/humedad.	64
4.15. Ensayo en la plataforma EDU-CIAA-NXP del ejemplo <i>dht11</i>	65
4.16. Código del ejemplo en <i>eclipse</i>	65
4.17. Cambios en la placa EDU-CIAA-NXP durante el ensayo.	66
4.18. Salida de la terminal COM7, Tera Term VT.	66
4.19. Salida por consola del ensayo tick hook.	68
4.20. Salida de la terminal COM7 -Tera Term VT.	68

Índice de tablas

2.1. Comparación de características de los emuladores revisados	9
3.1. sapi_peripheral_map.h	35
3.2. Funciones sapi_gpio	36
3.3. Funciones sapi_adc	38
3.4. Funciones sapi_tick	40
3.5. Funciones sapi_delay	42
3.6. Funciones sapi_dht11	43
3.7. Comparación entre funcionalidades de <i>freeRTOS</i> que se cumplen en el emulador.	45
3.8. Periféricos internos implementados	45
3.9. Periféricos externos	46
4.1. Recursos de hardware utilizados	51
4.2. Recursos de software utilizados	52

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.

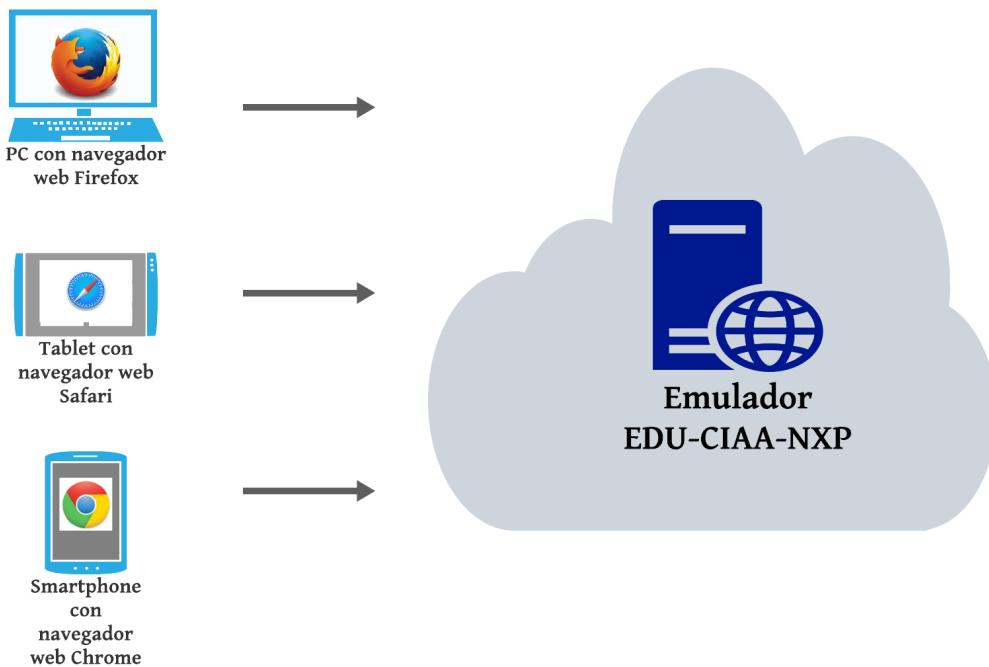


FIGURA 1.1. Esquema Emulador EDU-CIAA-NXP.

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.

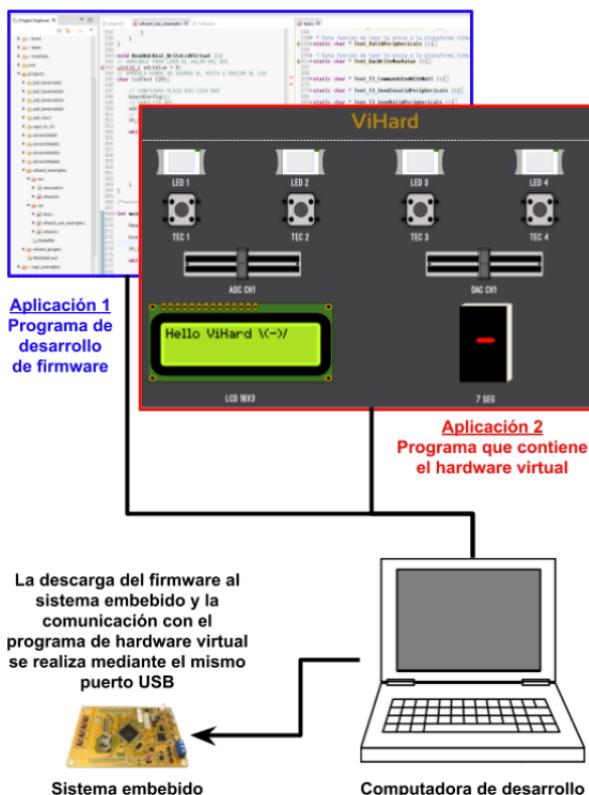


FIGURA 2.1. Esquema de la plataforma ViHard.¹

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.

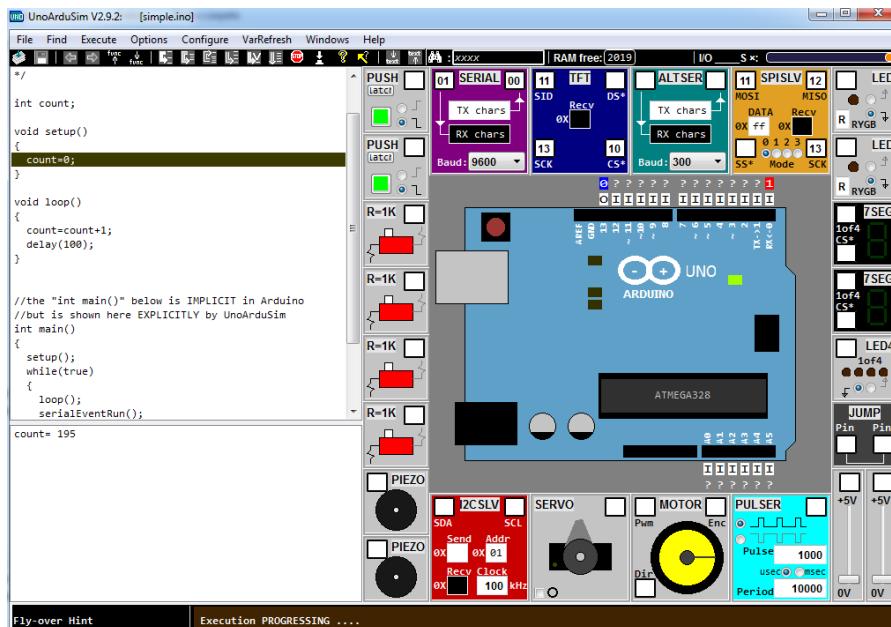


FIGURA 2.2. Plataforma UnoArduSim.

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.

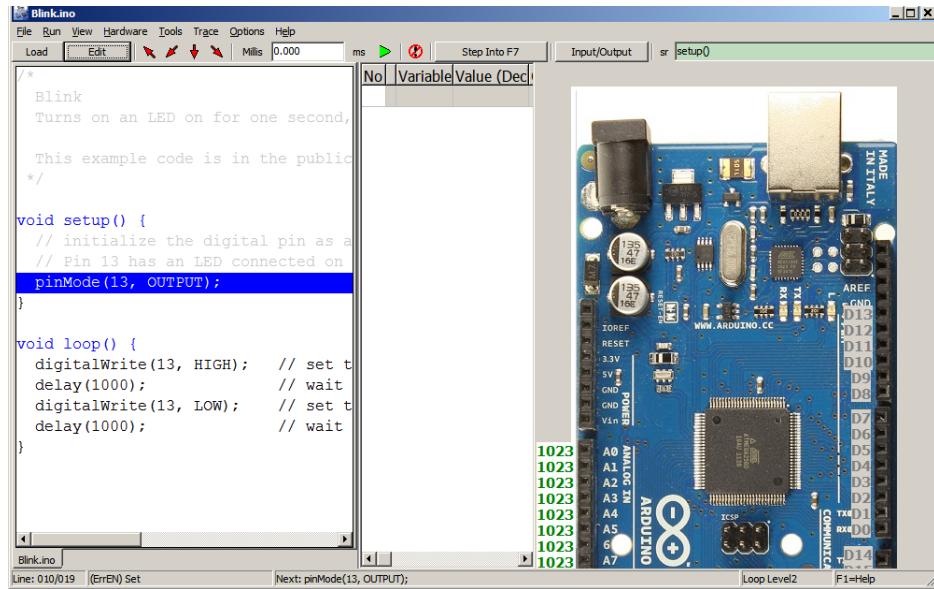


FIGURA 2.3. Plataforma Virtronics.

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.

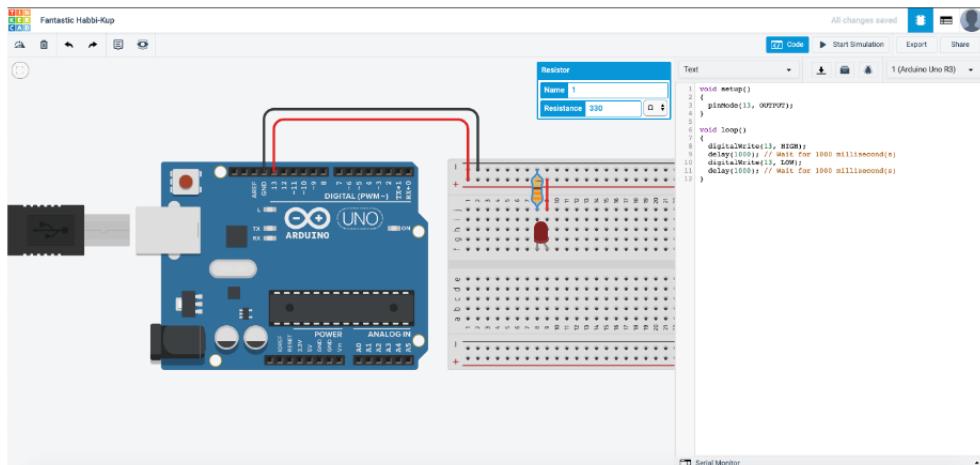


FIGURA 2.4. Plataforma Tinkercad.

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.

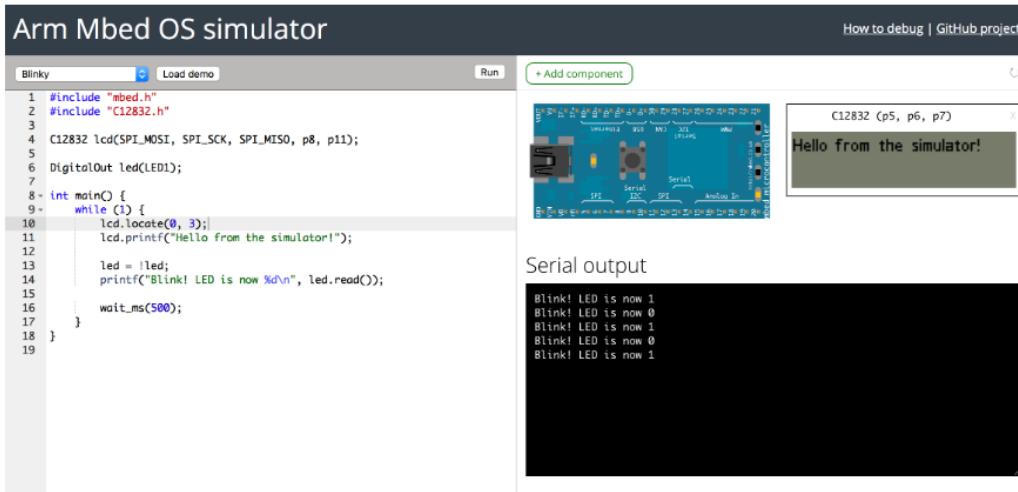


FIGURA 2.5. Plataforma Arm Mbed OS Simulator.

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 y 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.

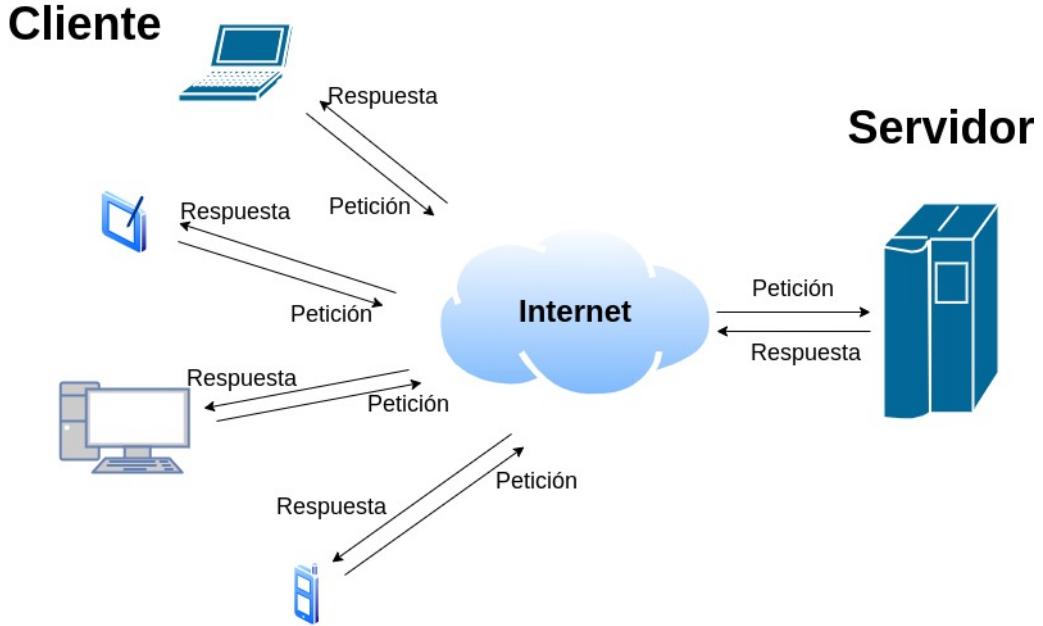


FIGURA 2.6. Esquema modelo cliente/servidor.

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:

- *Node.JS* [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 una *toolchain* de compilación completa 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 *Node.JS*. 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. Descripción funcional básica

Estas tecnologías presentadas en la sección 2.3.1 se relacionan entre sí como se muestra en la figura 2.7.

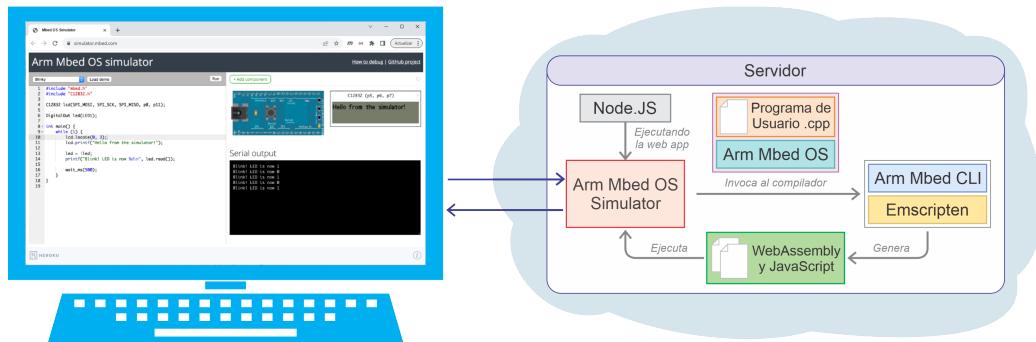


FIGURA 2.7. Funcionamiento de *Arm Mbed OS Simulator*.

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

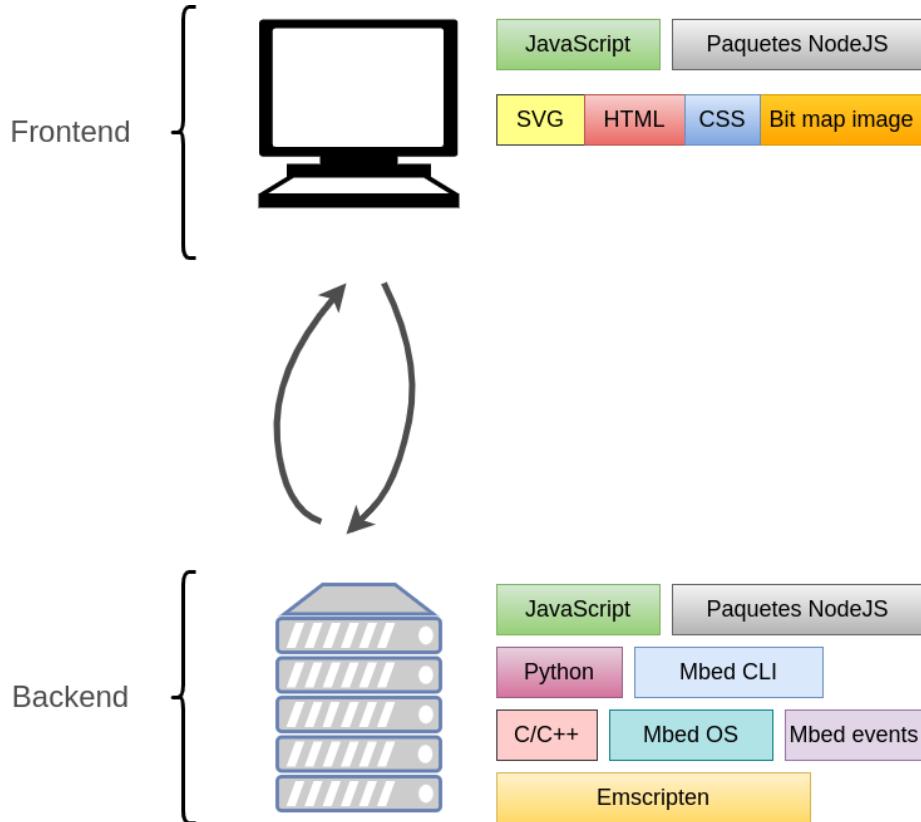


FIGURA 2.8. Esquema de las tecnologías utilizadas en *Arm Mbed OS Simulator*.

Como se puede observar, existen tecnologías que 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, en 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 "*out*", 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, son descriptos en base a objetos 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. Descripción funcional en detalle

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.

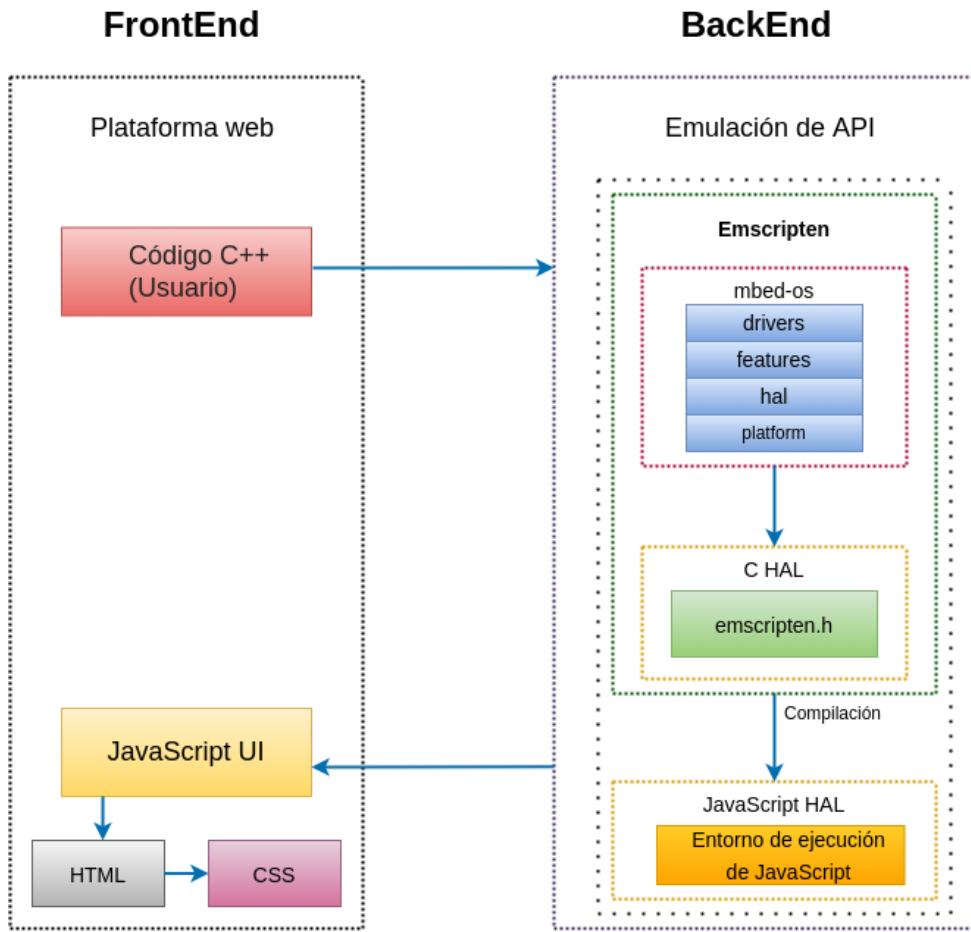


FIGURA 2.9. Arquitectura de capas de la plataforma *Arm Mbed OS Simulator*.

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 (en inglés, *targets*). Esto incluye las capas de periféricos, como 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*. *HAL* son las siglas en inglés de *Hardware Abstraction Layer*, en español, capa de abstracción de hardware. De esta manera, para implementar el *port* de la biblioteca *Arm Mbed OS* a una nueva plataforma de hardware, solamente se deben portar los archivos pertenecientes a la capa *Mbed C HAL*.

Arm Mbed OS Simulator utiliza el mismo enfoque, implementando un nuevo *target*, nombrado (`TARGET_SIMULATOR`) que implementa la capa *Mbed C HAL*. Sin embargo, la diferencia más importante, es que la capa *Mbed C HAL* que implementa en *Arm Mbed OS Simulator*, en lugar de escribir registros de un microcontrolador real, pasa eventos a una *HAL* implementada en JavaScript (nombrada capa *JS HAL*). Luego, la interfaz de usuario se suscribe a estos eventos y actualiza en consecuencia los componentes gráficos del simulador, esto se implementa en una capa nombrada *JS UI*.

Si se toma por ejemplo, un elemento la API *DigitalOut* de *Mbed OS*, el flujo de la información para controlar un LED conectado a un pin configurado como 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*.

Para que *Mbed C HAL* pueda pasar los eventos a la capa *JS HAL* se utiliza la biblioteca *emscripten.h*. Esta biblioteca provee las funciones y macros necesarias para interactuar con el compilador de *Emscripten*, permitiendo que el código C use funciones nativas *JavaScript*. De esta manera, cuando se compila el programa de usuario, junto a *Arm Mbed OS* con *Emscripten* se obtienen archivos *WebAssembly* y *JavaScript* que pueden comunicarse con la capa *JS HAL*.

La capa (*JS HAL*) actúa como intermediario, distribuyendo eventos entre los componentes de las capas *JS UI* y *C HAL*. Para relizarlo, implementa un bus de eventos para permitir que la interfaz de usuario se suscriba a eventos de C++. Para lograr este objetivo se utiliza la clase *EventEmitter* del paquete *Events* de *Node.JS*. *EventEmitter* monitoriza y activa los eventos, facilitando la interacción del navegador con el código *JavaScript* y permitiendo la actualización de la interfaz de usuario de manera flexible y eficiente.

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 2.11 se muestra el modelo de *publicación/suscripción*.



FIGURA 2.11. Modelo de *publicación/suscripción*.

Los objetos de la capa *JS UI* manejan los eventos de la interfaz de usuario y solo se comunica con *JS HAL*. Para enviar cambios desde la interfaz gráfica a la capa *JS HAL*, se utiliza directamente su API, mientras que para recibir cambios en la interfaz gráfica desde la capa *JS HAL*, los objetos de la capa *JS UI* se suscriben al detector de eventos.

Para describir este funcionamiento de la suscripción a los eventos, se debe introducir el concepto de *listeners* (en español, oyentes) de *JavaScript*. Los *listeners* se crean utilizando el método `on()` y pasando como argumento el nombre del evento al que se quiere suscribir.

La figura 2.12 muestra que cuando se emite algún evento en la *JS HAL*, entonces el oyente suscrito a ese evento en esta capa *JS UI* lo podrá escuchar y realizar las acciones que correspondan para la funcionalidad requerida.

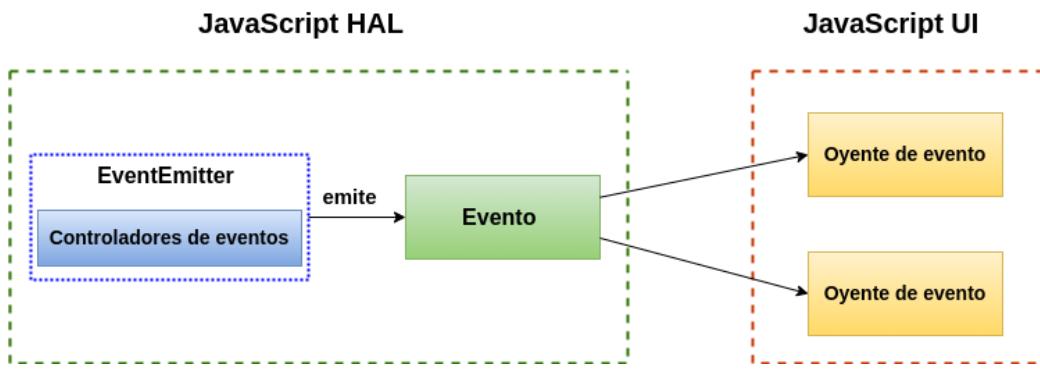


FIGURA 2.12. Diagrama de bloques de los oyentes de *EventEmitter* en la capa UI.

De esta manera, existen varios subscriptores para un mismo evento en diferentes archivos *JavaScript*. En consecuencia, se logra una mayor interactividad entre los componentes de la plataforma.

En resumen, se explica 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. Caso de estudio: programa que activa un LED en la placa virtual

Para exhibir el funcionamiento de *Arm Mned OS* en su totalidad, se presenta en esta sección, un caso de estudio, correspondiente a la ejecución de un programa que activa un LED en la placa virtual.

El usuario escribe un programa en lenguaje *C++*, que utiliza la biblioteca *Arm Mbed OS* para interactuar con un pin configurado como salida, conectado a un *LED*, ubicado en la propia placa virtual.

A continuación, ejecuta el programa dentro de la plataforma web. Para lograr esto, mediante *Node.js* se ejecutan 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++*.
- La biblioteca *Arm Mbed Os*, capa *Biblioteca C++*.
- El archivo `gpio_api.c` de la capa *C HAL*.
- El archivo `emsdk.h`.

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 *WebAssembly* y *JavaScript*, permitiendo que el programa escrito originalmente en *C++* pueda ser ejecutado dentro del entorno web. Los archivos resultantes de este proceso incluyen:

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

Emscripten utiliza un identificador único para los archivos generados. Este identificador esta compuesto por el prefijo `user_` y un número entero que representa el tiempo en milisegundos.

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 los archivos en las capas *JS HAL* y *JS UI*.

La figura 2.13 muestra un diagrama de secuencia que expone la interacción del usuario con el sistema y sus capas interiores.

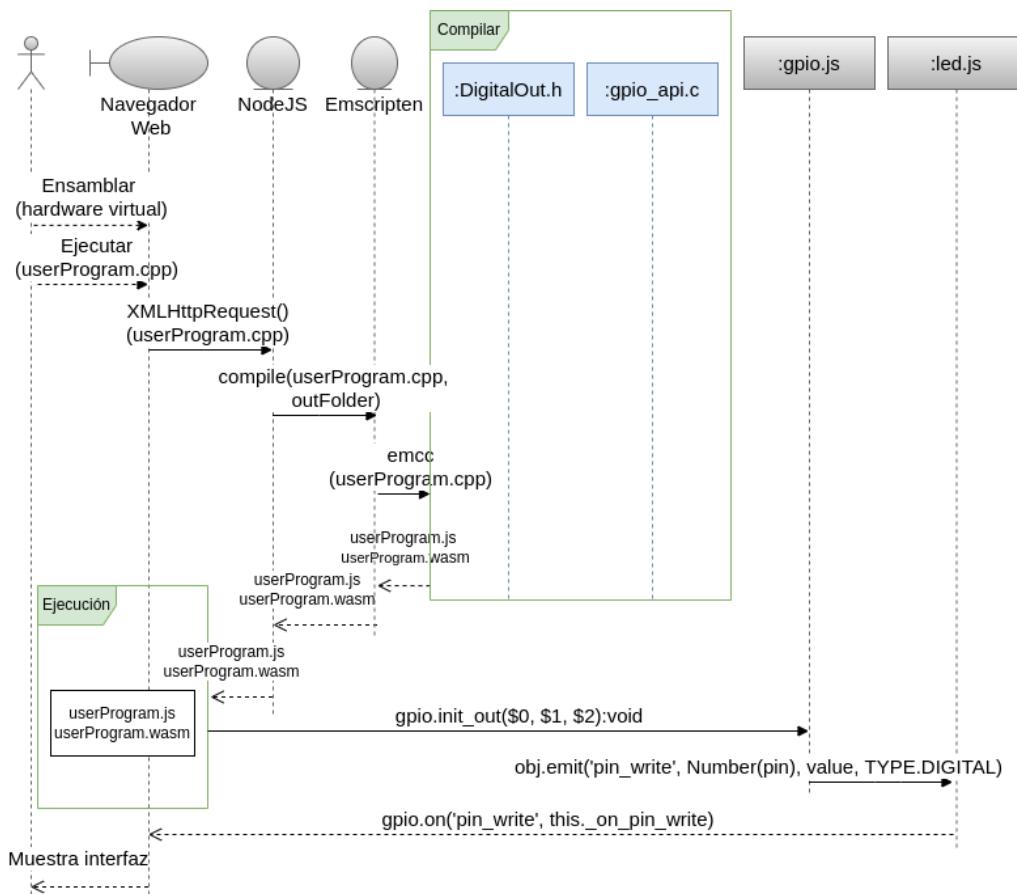


FIGURA 2.13. Funcionamiento de *Arm Mbed OS Simulator*.

La capa *JS HAL* se encarga de notificar los eventos ocurridos. Mediante la función `write`, se realiza la activación del evento que escribe en el GPIO. Como resultado, emitirá el evento con el nombre `pin_write`, pasando como argumentos el número de pin, el valor digital y el tipo de pin declarado, como se muestra en el diagrama en bloques de la figura 2.14.

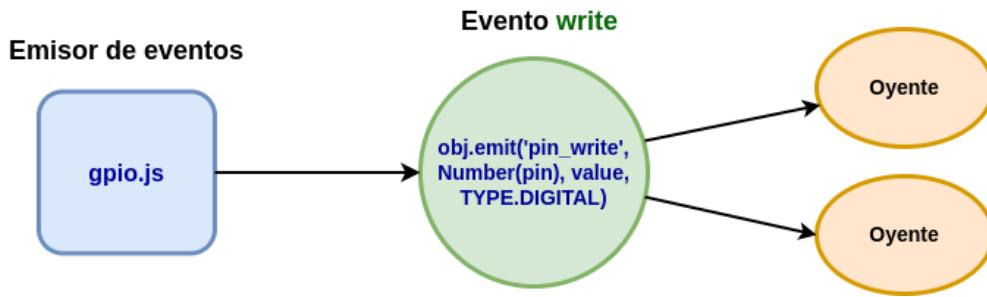


FIGURA 2.14. Activación de evento con el nombre `pin_write`.

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. Esto se ilustra en el diagrama en bloques de la figura 2.15.

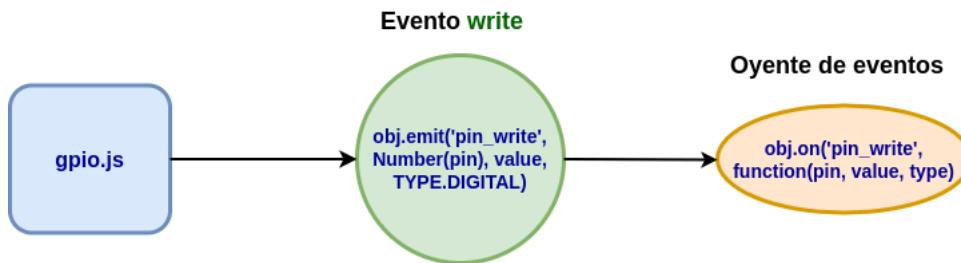


FIGURA 2.15. GPIO oyente del evento con el nombre `pin_write`.

Mediante estas interacciones entre todas las capas de programación, se muestran los cambios de `gpio_write` en la placa virtual. Esto se resume en la figura 2.16.

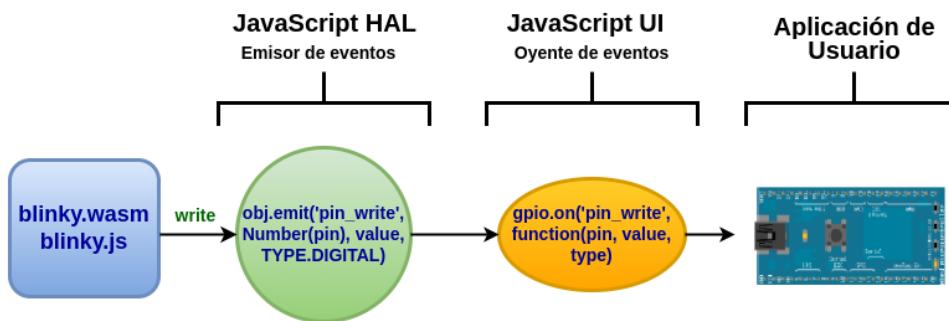


FIGURA 2.16. Interacción entre todas las capas de programación.

2.3.6. Análisis estructural de archivos y carpetas

En la figura 2.17 se exhibe la estructura de árbol de las carpetas y archivos de *Arm Mbed OS Simulator* al momento que fue clonado desde *Github* para la realización de este trabajo.

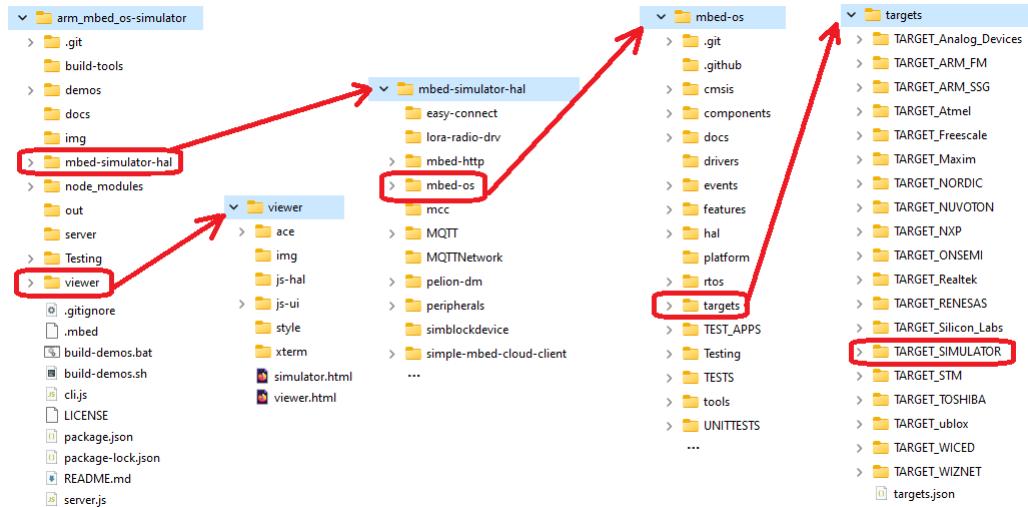


FIGURA 2.17. Estructura de carpetas y archivos de *Arm Mbed OS Simulator*.

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 aplaición web que realiza la compilación ((*build*)) y gestión de dependencias.
3. *helpers.js*, es un módulo de la aplaición 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 incuye:

- 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 dde 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 we 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 los cambios fundamentales realizados sobre la estructura del código existente, para portarlo a la EDU-CIAA-NXP y las mejoras introducidas. Además, se documenta, 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

En esta sección se exponen las herramientas del desarrollo que se utilizan en este trabajo.

3.1.1. Plataforma de desarrollo EDU-CIAA-NXP

La EDU-CIAA-NXP 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.

3.1.2. Editor VSCode

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.

3.1.3. Editor de gráficos vectoriales *Inkscape*

Inkscape [44]: es un editor de gráficos vectoriales que permite crear, editar y modificar gráficos. En el proceso de desarrollo de la interfaz gráfica, se utilizó para diseñar diversos componentes de hardware.

3.1.4. Sistemas de control de versiones e integración continua

- 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.

3.1.5. Herramientas de *Testing*

Arm Mbed OS Simulator no posee ninguna plataforma de *tests*. Los únicos tests encontrados pertenecen a la biblioteca *Arm Mbed OS*, es por esto que se agregan las siguientes herramientas para *testing* en el *Frontend*:

- Mocha [48]: es un marco de trabajo para pruebas de JavaScript que tiene funciones que se ejecutan en *Node.JS* y en el navegador web. En consecuencia, hace que las pruebas asíncronas 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 permite hacer pruebas sobre la interfaz de usuario de la plataforma.
- Chai [49]: es una biblioteca de aserciones (como por ejemplo, *assert*, *expect* y *should*) que puede usarse con cualquier marco de pruebas de Javascript. Chai se utiliza en las pruebas de *EmuCIAA* para verificar el comportamiento esperado de las funciones, componentes y datos generados.

Mientras que para *testing* en el *Backend*, se agregan las siguientes:

- Check [50]: 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 *EmuCIAA* para verificar el correcto funcionamiento de las funciones y componentes implementados en el backend escritos en lenguaje C.

- *CMocka* [51]: 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 permite simular funciones mediante *mocks* para controlar el comportamiento de las funciones dependientes y facilitar las pruebas de código que interactúa con dichas funciones.

En el capítulo 4 se muestran estas herramientas en uso.

3.1.6. Servidor en la nube *DigitalOcean*

DigitalOcean [52]: 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. Restructuración de archivos y carpetas

Luego de comprender el funcionamiento de *Arm Mbed OS Simulator*, se comienza el desarrollo de este trabajo. Se utiliza a partir de aquí el nombre *EmuCIAA*, para referirse al *Emulador de la placa EDU-CIAA-NXP*.

En principio, se eliminaron módulos y dependencias innecesarias para el alcance de este trabajo, 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 *EmuCIAA*.

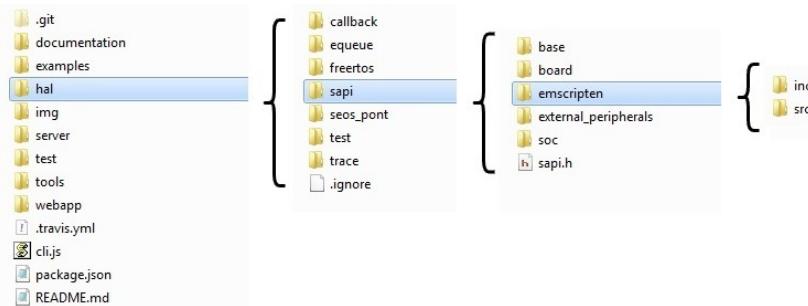


FIGURA 3.1. Estructura de carpetas y archivos de *EmuCIAA*.

En las siguientes secciones se detallan todos estos cambios y agregados realizados.

3.3. Frontend: rediseño de la Interfaz de Usuario

Para el rediseño 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, reordenando los elementos en pantalla.

Para su estudio, la figura 3.2 muestra la interfaz gráfica de *EmuCIAA* dividida en tres partes.

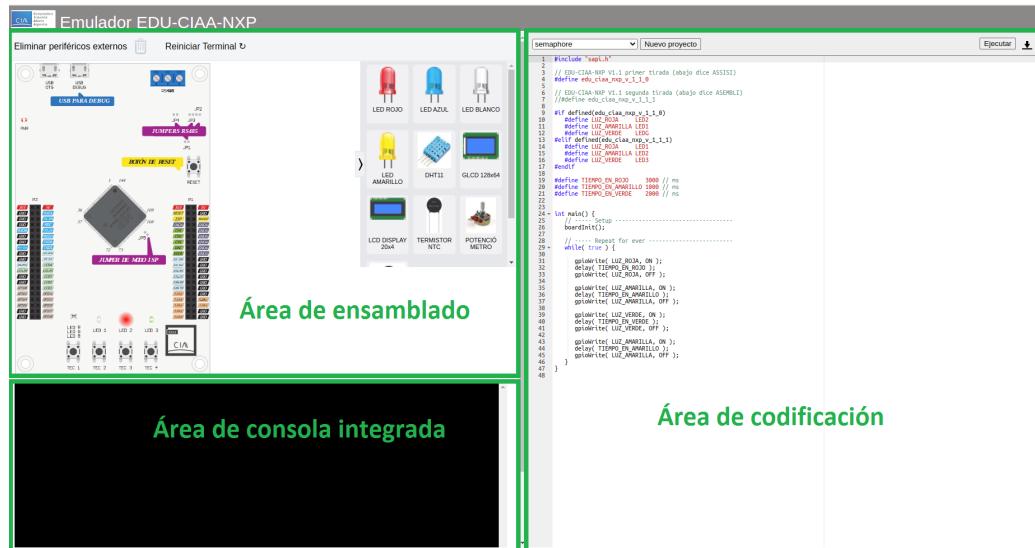


FIGURA 3.2. Interfaz gráfica de *EmuCIAA*.

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 externos.
- 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 (ejemplo *blinky.c*). Se mantuvo *Ace* como editor.
- ÁREA DE CONSOLA INTEGRADA: se muestra en una ventana la salida de las UARTs de la EDU-CIAA-NXP. Se reutilizó *xterm* para la consola.

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

- Probar los programas de ejemplo predeterminados.
- Crear un nuevo programa desde cero, o bien, modificar un ejemplo.
- Editar y ejecutar programas.
- Visualizar los cambios programados en la placa virtual.
- Agregar componentes de hardware virtual para conectar a la placa virtual.
- Ver los errores obtenidos en la programación.

- Ver la salida de las UARTs de la placa virtual.

3.3.1. Área de ensamblado

Para el desarrollo de la placa EDU-CIAA-NXP, y componentes externos para conectar a la misma, se agregaron los gráficos SVG (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.

Estos gáficos fueron proporcionados por el co-director del presente trabajo, Mg. Ing. Eric Pernia:

- SVG EDU-CIAA-NXP [53].
- SVG LED.
- SVG DHT11.
- SVG Potenciómetro $10K\Omega$.
- SVG Termistor NTC.
- SVG Analog Stick.
- SVG Display LCD 20x4 caracteres.
- SVG Display GLCD 128x64 píxeles y 16x4 caracteres.

Estos componentes se combinan con elementos de *JavaScript*, 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 y GLCD.

En la capa de programación *JS UI*, se implementa el comportamiento interactivo para los botones de la placa (TEC1, TEC2, TEC3 y TEC4), los cuales permiten ser pulsados; y los LEDs (LED_RGB, LED1, LED2 y LED3), que permiten mostrar los estados de encendido y apagado en la placa virtual.

En *Arm Mbed OS Simulator*, para agregar componentes y conectarlos a la placa de desarrollo, existe un botón que muestra la lista de componentes disponibles en una ventana. Con el objetivo de optimizar la experiencia del usuario, se desarrolló para *EmuCIAA* un nuevo *sidebar* (barra lateral) colapsable, donde se exhiben todos los componentes. De esta manera, para agregar un componente, el usuario debe expandir la barra lateral y hacer *click* sobre el periférico elegido. Al realizar esta acción aparece una ventana modal para realizar la configuración de las conexiones entre el componente y la EDU-CIAA-NXPI (figura 3.3).

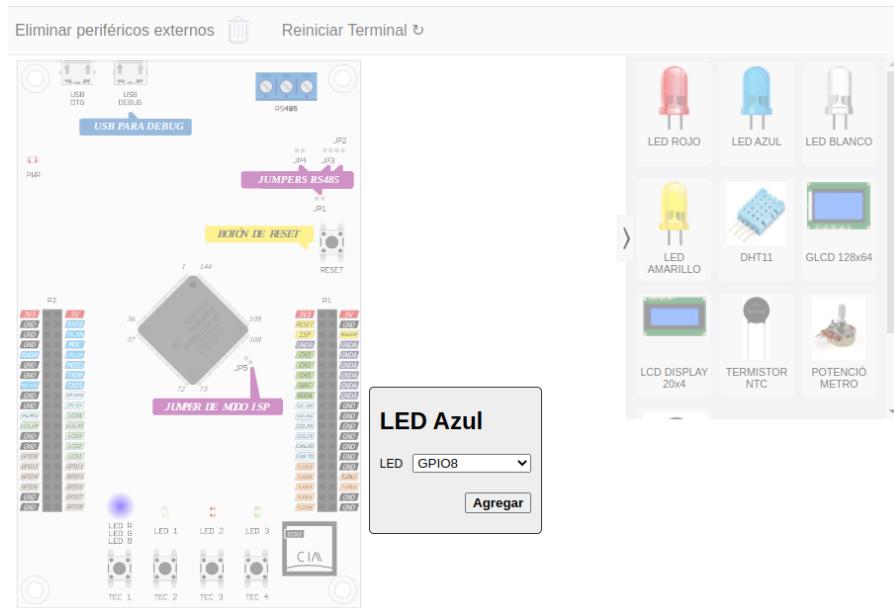


FIGURA 3.3. El usuario puede elegir un componente en la aplicación.

Una vez aceptada la configuración, la barra automáticamente colapsa, ocultándose del área de ensamblado. Al colapsarse, se muestra el periférico integrado en la aplicación (figura 3.4).

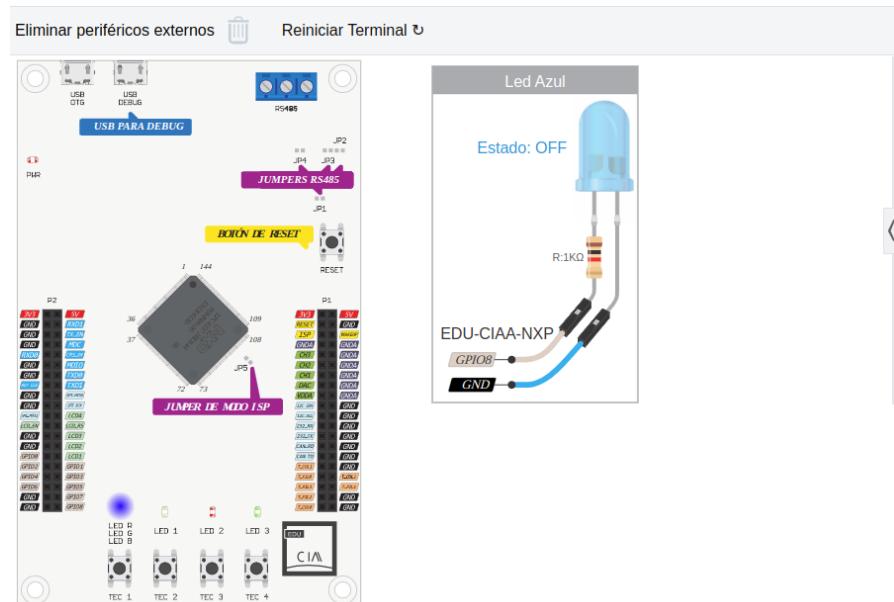


FIGURA 3.4. Periférico agregado en el área de ensamblado.

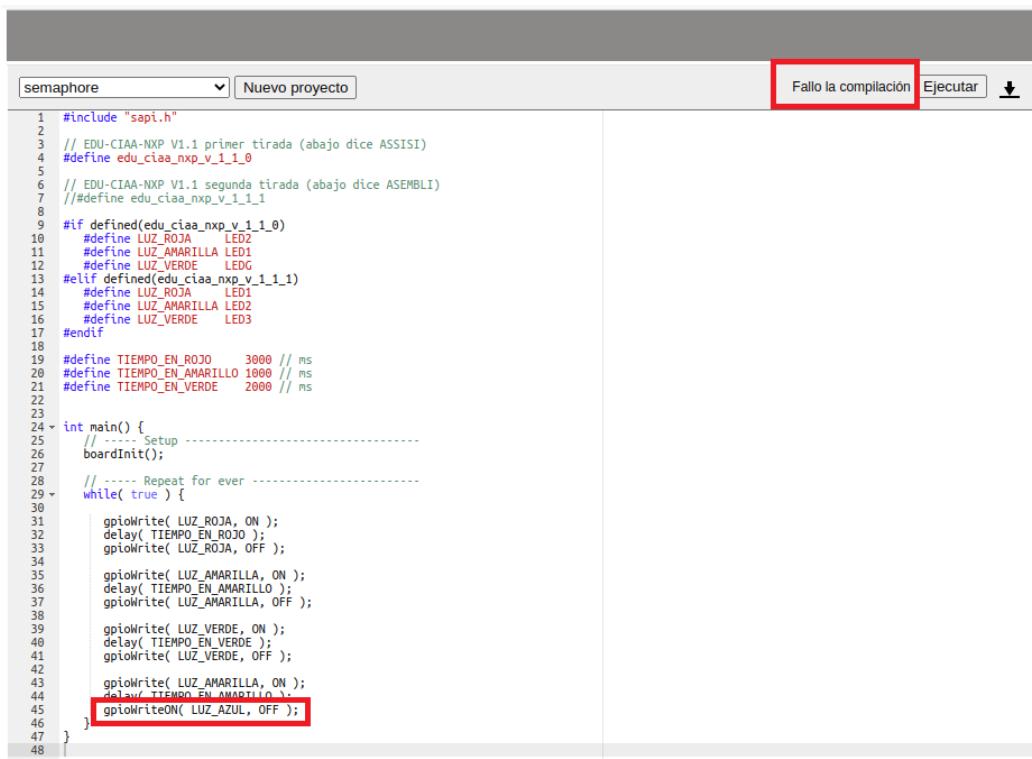
Para eliminar un componente, simplemente debe seleccionar el componente deseado, y se habilita el ícono de "Eliminar periféricos externos". Al hacer clic en el ícono, el periférico virtual es eliminado de la pantalla.

3.3.2. Área de codificación

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

- Manejar la sintaxis para los lenguajes C/C++. Esto significa que permite el uso de las palabras claves, comentarios, constantes, etc., realizando el colorido de sintaxis.
- Resaltado de líneas de código, sangría automática y número de línea.
- Función buscar (ctrl + f).
- Función buscar/reemplazar (ctrl + h).
- Función deshacer (ctrl + z).
- Función rehacer (ctrl + y).

Para compilar un programa, la plataforma provee al usuario el botón “Ejecutar”. Si existen errores en el código, que no permitan la compilación, se muestran los mismos en pantalla. Como ejemplo, en la figura 3.5 se muestra un código que genera los errores de compilación. Mientras que en la figura 3.6 se presentan los errores de compilación que se muestran en el área de ensamblado en dicho caso luego de intentar ejecutar el programa.



```

semaphore          ▾ Nuevo proyecto
Fallo la compilación Ejecutar ↴

1 #include "sapi.h"
2 // EDU-CIAA-NXP V1.1 primer tirada (abajo dice ASSISI)
3 #define edu_ciaaa_nxp_v_1_1_0
4
5 // EDU-CIAA-NXP V1.1 segunda tirada (abajo dice ASEMBLI)
6 //#define edu_ciaaa_nxp_v_1_1_1
7
8 #if defined(edu_ciaaa_nxp_v_1_1_0)
9     #define LUZ_ROJA    LED2
10    #define LUZ_AMARILLA LED1
11    #define LUZ_VERDE   LEDG
12 #elif defined(edu_ciaaa_nxp_v_1_1_1)
13     #define LUZ_ROJA    LED1
14     #define LUZ_AMARILLA LED2
15     #define LUZ_VERDE   LED3
16 #endif
17
18 #define TIEMPO_EN_ROJO    3000 // ms
19 #define TIEMPO_EN_AMARILLO 1000 // ms
20 #define TIEMPO_EN_VERDE   2000 // ms
21
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 genera errores de compilación.

```

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:45:20: 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.

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.

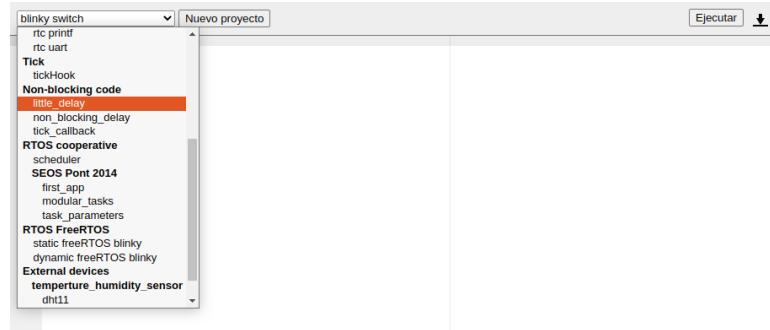


FIGURA 3.7. Estructura jerárquica de ejemplos.

Ademas, la aplicación automáticamente carga dentro del área de ensamblado un periférico con las conexiones a los pines configurados por defecto, cuando el usuario selecciona algún ejemplo que contenga un periférico externo. De esta manera, el usuario ya no tiene la necesidad de seleccionar y configurar el periférico para probar un ejemplo. La figura 3.8 muestra el periférico agregado automáticamente al seleccionar el ejemplo "potentiometer".

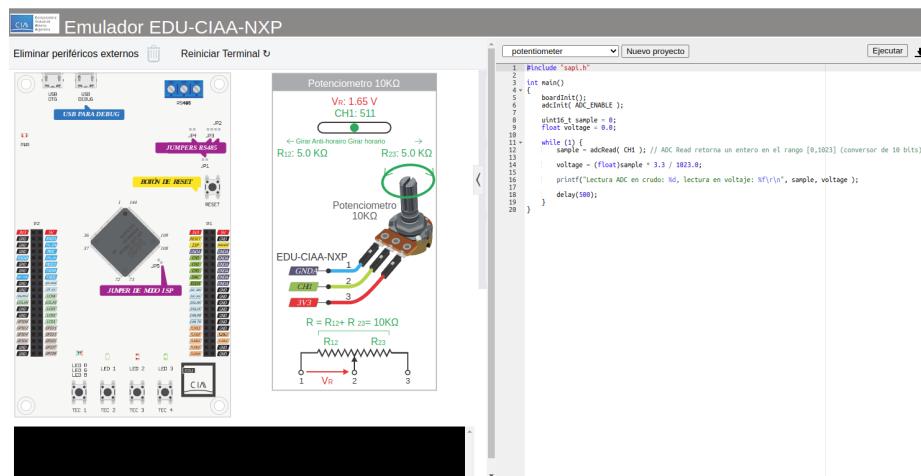


FIGURA 3.8. Carga automática del periférico.

3.3.3. Área de consola integrada

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

- `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`.

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 de *EmuCIAA* a través de la biblioteca *xterm.js*.

Entre las principales características de *xterm.js* se destaca:

- Funciona con la mayoría de las aplicaciones de terminal, como por ejemplo, `bash`, siendo 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 muestra un programa de ejemplo que genera imprime por consola y en la figura 3.10 se puede observar dicha salida reflejada.

```

rtc printf
Nuevo proyecto
Ejecutar

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
36      // Note: printf() use sAPI UART_USB (Chip USART2 on EDU-CIAA-NXP) at 115200, 8N1
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 }

```

FIGURA 3.9. Programa de usuario que imprime por consola.

```

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

```

FIGURA 3.10. Salida de la terminal serie.

3.4. Backend: bibliotecas de C portadas

A continuación se describen las principales bibliotecas de C portadas a *EmuCIAA*, las cuales fueron extraídas del repositorio *firmware_v3* del Proyecto CIAA, descargando la última release estable al momento de comenzar este proyecto. Esta release corresponde a la versión *r1.3.0* [54]:

- *sAPI* [55]: 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* [56] y *Firmware v3*.
- *seos_pont* [57]: es un sistema operativo de tiempo real para microcontroladores, de código abierto, muy sencillo que permite definir tareas que se ejecutan en forma cooperativa, periódicamente.
- *freeRTOS*: [58] FreeRTOS es un sistema operativo de tiempo real para microcontroladores, de código abierto, que facilita la programación, el despliegue, la protección, la conexión y la administración de dispositivos periféricos pequeños y de bajo consumo. Permite trabajar en modo de ejecución de tareas apropiativo o cooperativo.

En las siguientes secciones se describen los detalles más sobresalientes de la implementación realizada para llevar a cabo estos *ports*.

3.5. Biblioteca *sAPI*

Para la emulación a nivel de API, se implementa el *port* de la biblioteca *sAPI v0.6.2* para que funcione en la web, en lugar de funcionar en el hardware de un microcontrolador real. De esta manera, se proporciona 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[59]), o el acceso directo a registros físicos del microcontrolador, al igual que sucede con el emulador *Arm Mbed OS Simulator*.

Para llevar a cabo la tarea de portar la *sAPI* a *EmuCIAA*, se identificaron las funciones originales. Luego, en *EmuCIAA* se han creado interfaces que reflejan su composición, que incluye definiciones de funciones, estructuras de datos y constantes. De esta manera, 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 de *EmuCIAA*.

Asimismo, se utiliza un esquema de nomenclatura de los archivos de encabezado y de código fuente igual al de las bibliotecas originales. Esto permite mantener una estructura organizada y coherente en la emulación, que facilita el mantenimiento y comprensión.

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 utilizar en el programa de usuario.

También se reutilizaron los archivos de encabezado: `sapi_datatypes.h` y `sapi_peripheral_map.h` incluidos en todos los módulos de la biblioteca *sAPI* para permitir el uso de los tipos de datos básicos y nombres de periféricos de la placa. En la tabla 3.1 se muestran los tipos de datos de `sapi_peripheral_map.h` que se usan para los pines GPIO en la plataforma de emulación.

TABLA 3.1. 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莫斯I, ENET_RXD0	CAN_TD		

Se puede observar que se reutilizó los nombres: TEC1, TEC2, TEC3 y TEC4 para los botones y los nombres LEDR, LEDG, LEDEB, LED1, LED2 y LED3 para los LEDs de la placa EDU-CIAA-NXP.

Se realizó el *port* de los siguientes módulos de la biblioteca *sAPI*:

- *sapi_board*: Contiene funciones de inicialización para la plataforma de hardware.
- *sapi_gpio*: Es una *HAL* para pines E/S de propósito general (en inglés, *GPIO*).
- *sapi_uart*: Es una *HAL* para periféricos *UART*.
- *sapi_adc*: Es una *HAL* para conversores a Analógico-Digital *ADC*.
- *sapi_rtc*: Es una *HAL* para el periférico reloj de tiempo real, *RTC*.
- *sapi_tick*: Es una *HAL* que permite programar interrupciones periódicas.
- *sapi_delay*: Implementa funciones de retardos bloqueantes y no bloqueantes.
- *sapi_dht11*: Implementa un driver para el sensor de Humedad y temperatura DHT11.
- *sapi_display*: Implementa un driver para displays LCD y GLCD.

Existen otros módulos de *sAPI* independientes del hardware como, por ejemplo, *sapi_button*, los cuales son independientes del hardware y se utilizaron para los ejemplos de en este trabajo sin modificaciones. El resto de los módulos que implementan otros periféricos, como por ejemplo *I2C*, se dejan como trabajo a futuro.

En las siguientes secciones se muestran destalles de implementación de algunos de los módulos *portados*.

3.5.1. *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 3.2 muestra las funciones del archivo de código fuente *sapi_gpio* en la biblioteca *sAPI*, que incluye los nombres de la funciones, los tipos de parámetros y el tipo de valor de retorno. Cabe destacar que estas definiciones son idénticas tanto en las *sAPI* como en *EmuCIAA*, lo que permite una correspondencia directa entre ambas.

TABLA 3.2. 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

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 dependencias del módulo GPIO de la biblioteca *sAPI* del proyecto CIAA (figura 3.11) y el módulo GPIO implementado en *EmuCIAA* (figura 3.12).

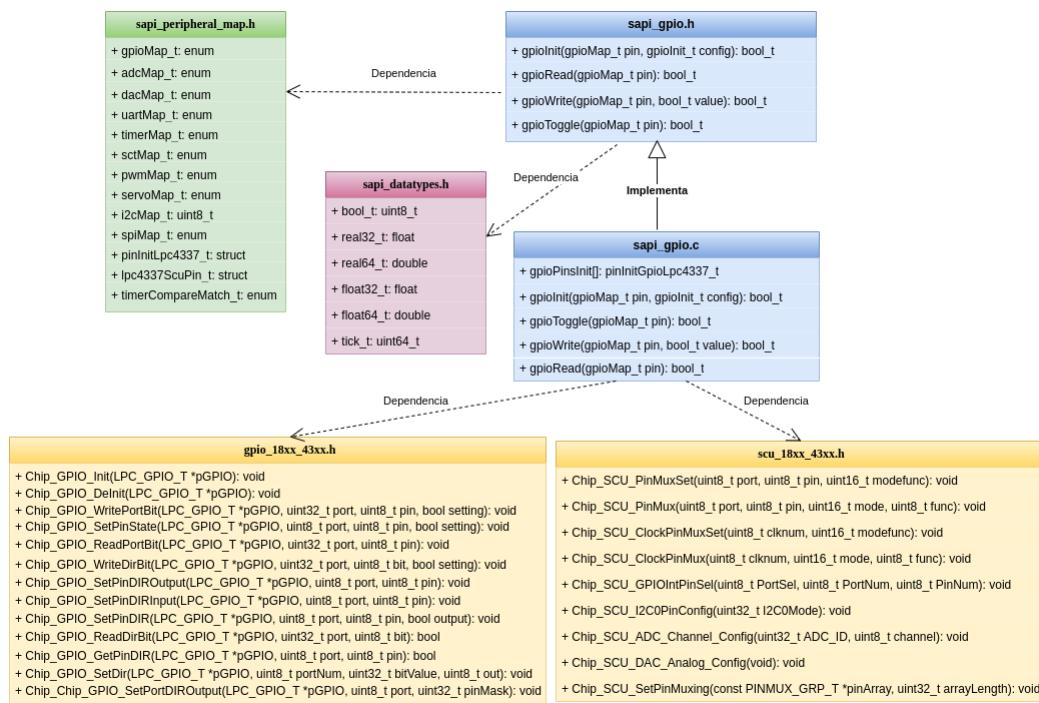
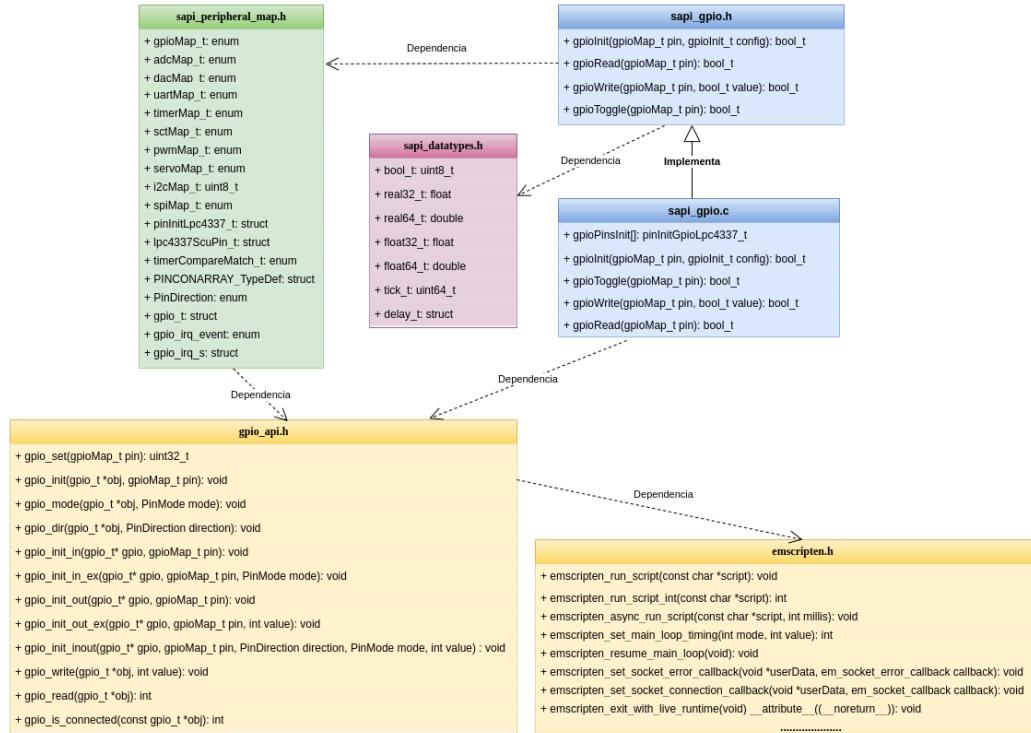


FIGURA 3.11. Dependencias del módulo *GPIO* de la *sAPI*

FIGURA 3.12. Dependencias del módulo GPIO de *EmuCIAA*.

Para lograr replicar el comportamiento de cada función, de manera que al invocarlas produzcan los mismos resultados que se obtendrían al utilizar las funciones con el hardware físico, se utiliza la *API* de *Emscripten* en la capa de *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* 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 la dirección de un pin, es decir, si se utilizará como entrada o salida. Se implementaron configuraciones similares para la capa *JS HAL* usando las macros de *Emscripten*.
- `Chip_GPIO_SetPinState(LPC_GPIO_PORT, gpioPort, gpioPin, 0)` se utiliza para establecer el estado de los pines configurados como salida y establecer su valor lógico (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 específico en la placa. 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 utiliza la macro de *Emscripten* nombrada `EM_ASM_`, mediante la cual se incrusta código *JavaScript* directamente en el código *C*. Este código *JavaScript* incrustado se compila junto con el código *C* y se ejecutará cuando la aplicación de usuario invoque a esas funciones. Esto es el equivalente a embeber un asesador de una dada arquitectura de procesamiento en un programa en *C*.

Asimismo, para emular las interacciones entre la interfaz de usuario y las GPIO TEC1, TEC2, TEC3 y TEC4, se utiliza en la capa *C HAL* la macro `EMSCRIPTEN_KEEPALIVE`. Su funcionamiento se detalla en la sección 3.5.3.

3.5.2. `sapi_adc`

Se realizó el mapeo de las funciones del módulo de la biblioteca *sAPI* a la plataforma web, que se expone en la tabla 3.3.

TABLA 3.3. Funciones `sapi_adc`

Función	Parámetros	Tipo de retorno
<code>adcInit</code>	<code>adcInit_t config</code>	<code>void</code>
<code>adcRead</code>	<code>adcMap_t analogInput</code>	<code>uint16_t</code>

El módulo *adc* es utilizado en los ejemplos desarrollados, junto con los periféricos externos: potenciómetro, termistor NTC y *AnalogStick*, los cuales no existían en *Arm Mbed Os Simulator* y fueron desarrollados para este trabajo.

Además, se implementaron las funciones de inicialización y de lectura del componente *adc* en la capa *C HAL*. Posteriormente, son utilizadas por la capa *JS HAL* para la interacción con el hardware y permitir al usuario trabajar con los periféricos externos en un entorno web. La figura 3.13 presenta el diagrama de bloques de las capas: *C HAL* y *JS HAL* para el *adc*.

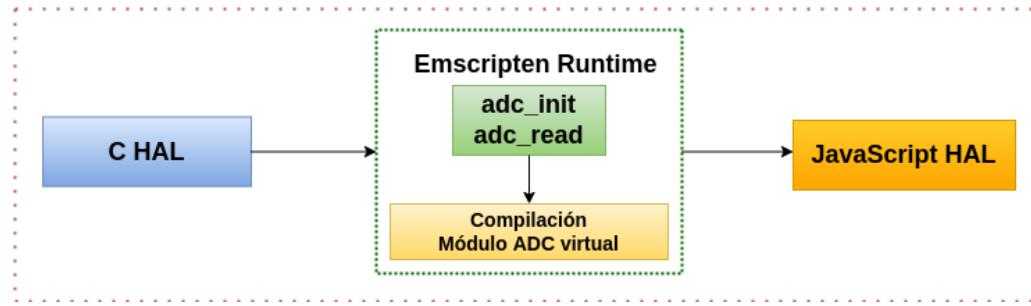


FIGURA 3.13. Diagrama de bloques de *C HAL* y *JS HAL* para el módulo *adc*.

En la capa *JS UI*, se implementó la obtención de datos para los periféricos externos que interactúan con el *adc*.

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, cuando el usuario desliza este componente dentro de un rango mínimo y máximo establecido (0 a 3.3V), se realiza el siguiente cálculo para obtener el valor del *adc* correspondiente en un valor entero:

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

CÓDIGO 3.1. Cálculo del ADC para el potenciómetro.

En consecuencia, los cálculos actualizados se muestran en la interfaz gráfica del emulador web emulando el comportamiento del conversor Analógico-Digital, que convierte de voltaje a un número entero dependiendo de su resolución y voltaje de referencia. En el caso del *ADC* del microcontrolador de la *EDU-CIAA-NXP*, posee 10 bits de resolución (dando valores entre 0 y 1023) y 3.3V de voltaje de referencia.

Para la implementación del termistor NTC, se reutiliza 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:

```
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).toFixed
(4));
9     console.log('this.sample ', this.sample);
10    window.JSHal.gpio.write(this.dataPin.ADC, this.sample);
11};
```

CÓDIGO 3.2. Cálculo ADC del termistor NTC.

Esto permite convertir entre temperatura a voltaje y de allí a muestra leída por el *ADC*.

En el caso del *AnalogStick* se implementó un componente web 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 *AnalogStick* 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 los cálculos para el eje X del *AnalogStick*:

```
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 3.3. Cálculo de la resistencia del eje X y del ADC.

Luego, para cada periférico externo, el cálculo del *adc* es enviado desde la capa *JS UI* a la capa *JS HAL*, como se esquematiza en la figura 3.14.

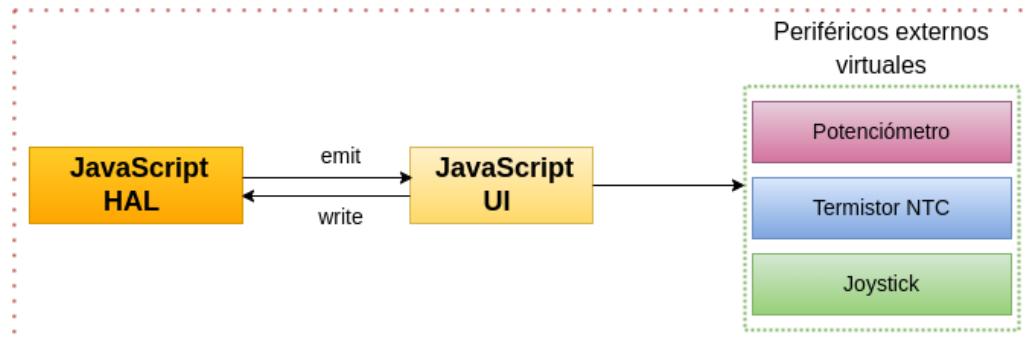


FIGURA 3.14. Diagrama de bloques con la interacción entre las capas *JS HAL* y *JS UI*.

3.5.3. *sapi_tick*

En la tabla 3.4 se detallan las funciones del archivo de código fuente *sapi_tick*.

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 el contador 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.

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 3.15 se muestra el funcionamiento de `EMSCRIPTEN_KEEPALIVE`.

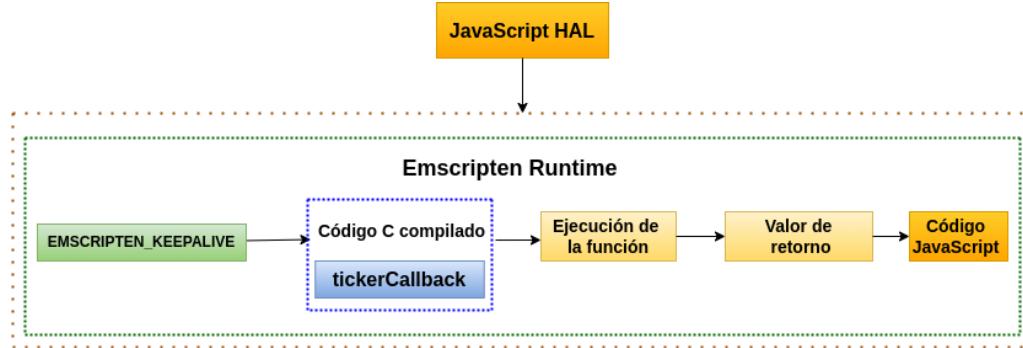


FIGURA 3.15. Diagrama de bloques `EMSCRIPTEN_KEEPALIVE`.

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 3.16 el funcionamiento de `ccall`.

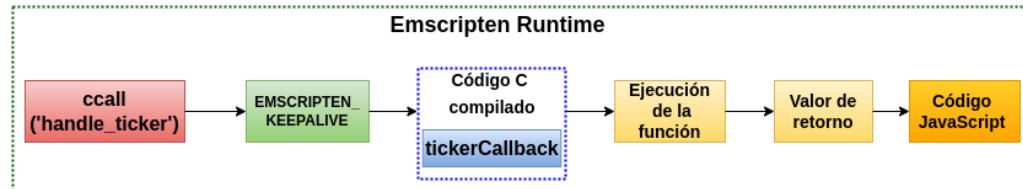


FIGURA 3.16. Diagrama de bloques de la función `ccall`.

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

En el capítulo 4 se detallarán las diferencias encontradas al realizar las pruebas entre la placa y el emulador utilizando estas funciones.

3.5.4. sapi_delay

La tabla 3.5 expone las funciones del archivo de código fuente `sapi_delay`.

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>

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 bloqueante se ejecutará mientras la diferencia de tiempo entre `tickRead()` y `startTime`(inicio actual de `tickRead()`) sea menor que `duration_ms / tickRateMS`.

Este comportamiento cae en las limitaciones de *Arm Mbed OS Simulator*, presentadas en la sección 2.2, causando que la ejecución de la plataforma web se bloquee o congele, debido a que es como si se ejecutase un `while(1)`. Por esta razón, se decide utilizar en su lugar las funciones nativas de *Emscripten* en la capa de emulación C *HAL*. En consecuencia, se aprovecha su eficiencia y precisión.

Para emular las funciones de espera de la biblioteca C se utiliza la función `emscripten_sleep`, que usa funciones asíncronas internas de *Emscripten* para realizar pausas. Por lo tanto, permite al navegador atender otros eventos mientras el programa se encuentra en espera. De esta manera, evita el bloqueo de la ejecución del resto del código y también, que la página no responda. Además, proporciona pausas precisas, debido a que, *Emscripten* utiliza las capacidades de temporización del navegador para garantizar que el tiempo indicado sea realizado. La figura 3.17 representa el funcionamiento de `emscripten_sleep`.

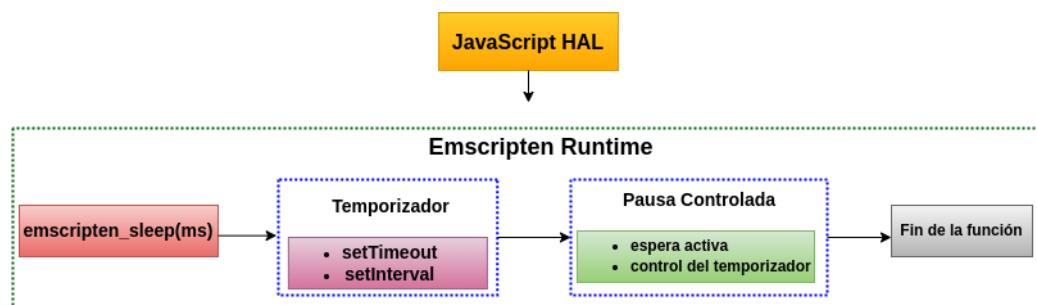


FIGURA 3.17. Diagrama de bloques `emscripten_sleep`.

3.5.5. sapi_dht11

Para emular drivers de periféricos externos de la biblioteca *sAPI*, se continuó con la misma lógica de programación utilizada para interactuar con los periféricos internos. Estos se mapean las funcionalidades ofrecidas por la biblioteca *sAPI* a la plataforma web. En la siguiente tabla 3.6 se describen las funciones presentes.

TABLA 3.6. 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 realizar las conexiones mediante cables virtuales entre la placa y los periféricos externos. Simplemente se muestran indicados los pines que el usuario debe elegir, para configurar a qué pin se conecta cada pin del sensor de humedad y temperatura *DHT11*. Estas conexiones luego serán verificadas en la capa de *JS UI*.

En la capa *JS UI* se centra principalmente el trabajo de emular el envío de los datos de temperatura y humedad del sensor *DHT11* a la placa de desarrollo. Entonces, en la capa de abstracción de datos *C HAL* se implementa la lectura de los datos provenientes de la capa *JS HAL* al usar la macro *EM_ASM_INT* de *Emscripten*, que se presenta mediante un diagrama en bloques en la figura 3.18.

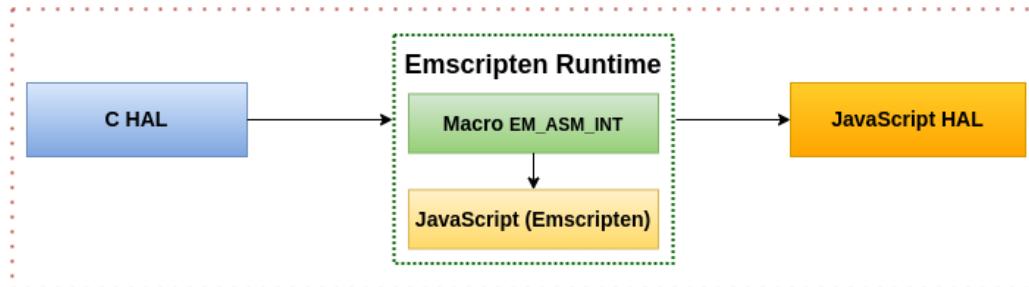


FIGURA 3.18. Diagrama de bloques de la macro EM_ASM_INT.

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

- 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 3.19 se muestra el diagrama de bloques de la capa de interfaz de usuario *JavaScript UI* con las dos opciones de usuario.

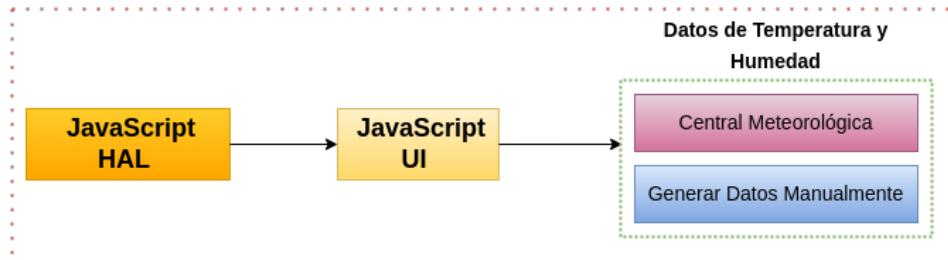


FIGURA 3.19. Diagrama de bloques de la capa *JavaScript UI* con las dos opciones para el usuario.

3.6. *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.
- `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 tiempos 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 proporciona 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 3.7 expone la comparación entre funcionalidades presentes en *freeRTOS* y las implementadas en *EmuCIAA*.

TABLA 3.7. Comparación entre funcionalidades de *freeRTOS* y las implementadas en *EmuCIAA*.

Funcionalidad	<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 esta primera versión del emulador, no se han incluido el manejo de multitareas y de prioridades presentes en *freeRTOS*.

3.7. Comparación de periféricos implementados

La tabla 3.8 expone los periféricos internos implementados actualmente en *Mbed OS Simulator* y los implementados en *EmuCIAA*.

TABLA 3.8. Comparación de periféricos internos implementados en *Mbed OS Simulator* y *EmuCIAA*

Periféricos	<i>Mbed OS Sim</i>	<i>EmuCIAA</i>
GPIO	Si	Si
UART	Si	Si
BUTTON	SI	Si
ADC	Si	Si
DAC	Si	No
PWM	Si	No
RTC	No	Si
Systick	No	Si

En la tabla 3.9 se comparan los periféricos externos implementados actualmente en *Mbed Simulator* con los implementados en *EmuCIAA*,

TABLA 3.9. Comparación de periféricos externos implementados en *Mbed OS Simulator* y *EmuCIAA*

Periféricos	MbedOS Sim	EmuCIAA
LED Rojo	Si	Si
LED Azul	Si	Si
LED Amarillo	Si	Si
LED Blanco	Si	Si
Potenciómetro	No	Si
AnalogStick	No	Si
Termistor NTC	Si	Si
DHT11	Si	Si
SHT31	Si	No
LCD C12832	Si	No
ST7789H2	Si	No
LCD HDD44780	No	Si
GLCD ST7920	No	Si

Finalmente, se compara el aspecto visual entre ambas paletas en las figuras 3.20, para *Mbed OS Simulator*, y 3.21 para *EmuCIAA*. De estas figuras se observa como para el desarrollo de *EmuCIAA* se tuvo especial cuidado para emular los componentes externos de la forma más fiel posible a la realidad y mostrar las conexiones de los pines de forma didáctica.

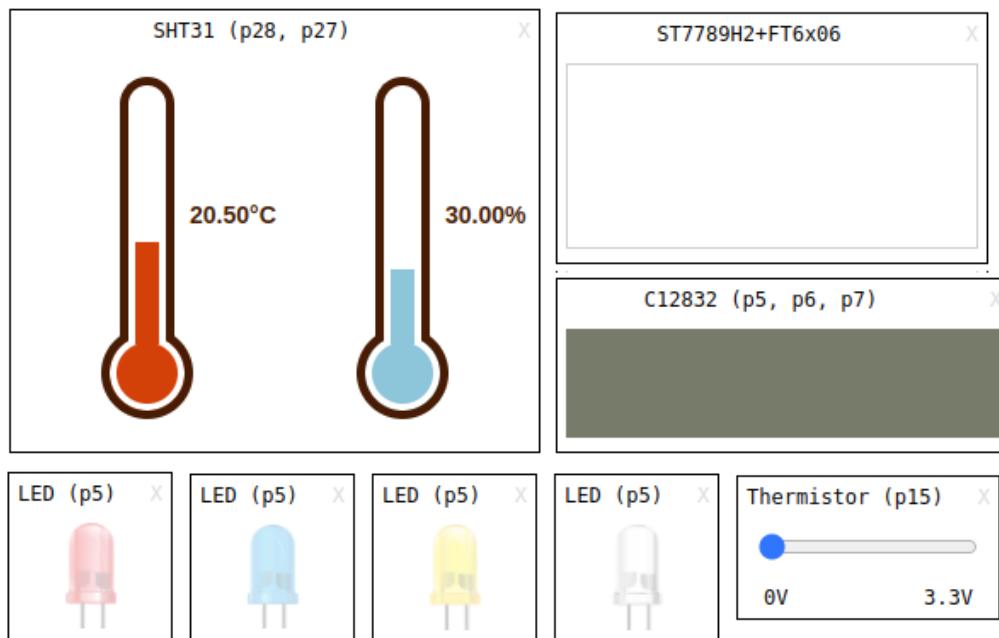


FIGURA 3.20. Periféricos externos de *Arm Mbed OS Simulator*.

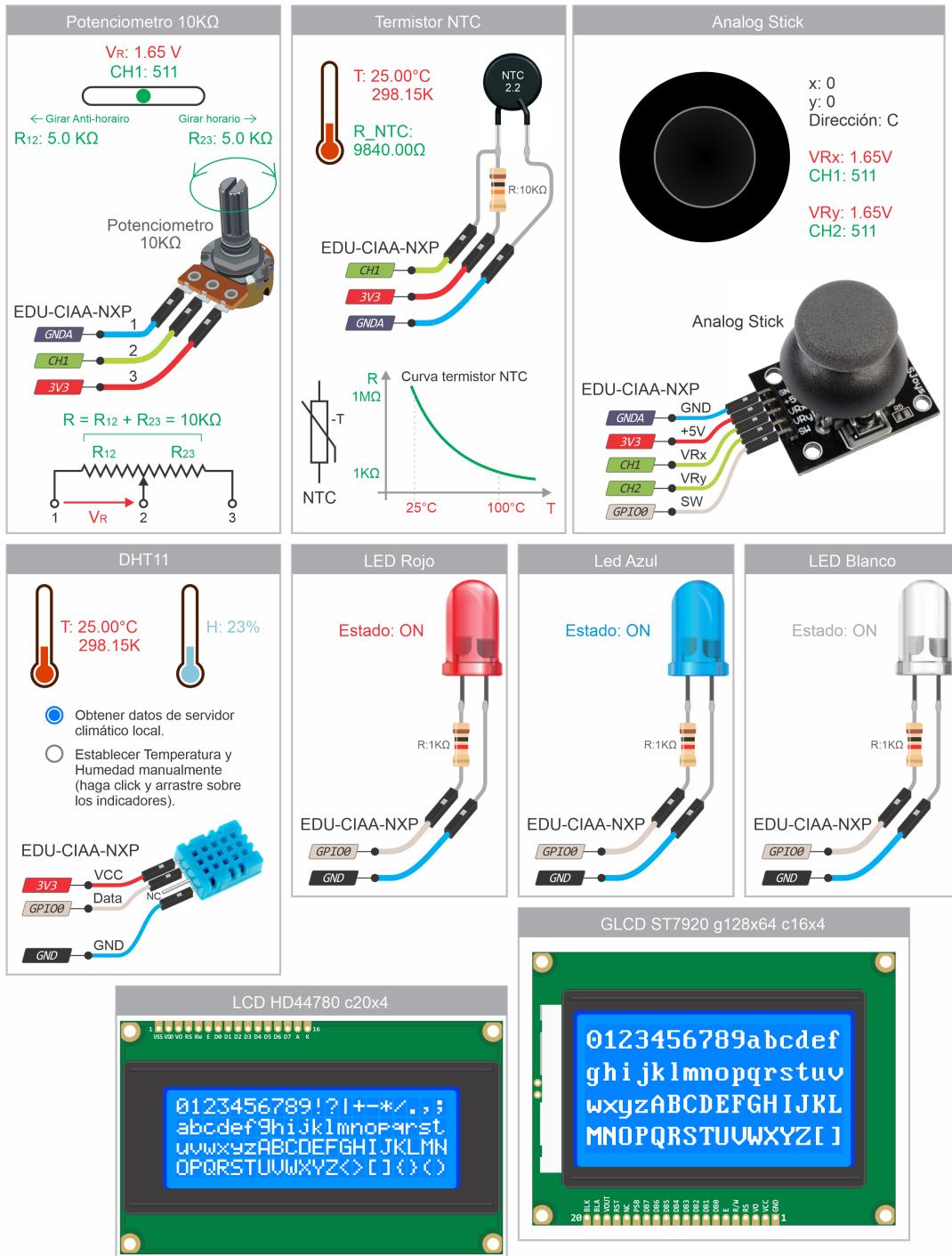


FIGURA 3.21. Periféricos externos del emulador web EDU-CIAA.

3.8. Ejemplos incluidos

Se incluyen en la plataforma los siguientes ejemplos de programas, los cuales se ordenaron cuidadosamente para permitir que se pueda ir aprendiendo de forma incremental el uso de varios periféricos y técnicas de programación de Sistemas Embebidos:

- **GPIO**

- blinky
- semaphore
- blinky_switch
- switches_leds
- led_sequences

- **UART**

- serial_output

- **RTC**

- rtc_uart
- rtc_prinf

- **ADC**

- potentiometer
- ntc_thermistor
- analog_stick

- **Tick**

- tick_callback

- **Sensors**

- dht11_temp_humidity

- **Displays**

- lcd_hd44780_c20x4_gpios
- lcd_hd44780_custom_chars
- glcd_st7920_c16x4
- glcd_st7920_g128x64

- **Non-blocking code**

- little_delay
- tick_counter
- non_blocking_delay

- **FSM**

- semaphore

- debounce
- button
- **RTOS cooperative**
 - scheduler
 - scheduler_bgfg
- **RTOS SEOS Pont**
 - pont_blinky
 - pont_tasks_parameters
 - pont_modular_tasks
 - pont_uart_clock
- **RTOS FreeRTOS**
 - freertos_blinky_dynamic_men
 - freertos_blinky_static_men

La mayoría de estos ejemplos fueron seleccionados de *firmware_v3*. Los nuevos programas realizados para *EmuCIAA* son:

- serial_output
- portentiometer
- ntc_thermistor
- analog_stick
- glcd_st7920_c16x4
- glcd_st7920_g128x64

3.9. Despliegue en un servidor web

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

- Crear una cuenta: que implica registrar una cuenta en la plataforma. Iniciar sesión y crear un *droplet*, que es un servidor virtual. Luego, se instala el sistema operativo, en este caso, *Ubuntu*, y se configura la región geográfica donde se ubicaría el servidor y la cantidad de RAM a utilizar.
- Acceso al servidor: se obtiene la dirección IP pública y la clave *SSH* para acceder al servidor.
- Configuración: implica 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 realizan las configuraciones de las reglas de *firewall*.
- Copiar la plataforma web al servidor: se utiliza *GitHub* para clonar el repositorio en el servidor. El código del presente trabajo, se encuentra en el repositorio de GitHub [60].

- Iniciar el emulador web: se utiliza 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 ejecutándose en la URL: <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 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 permiten una evaluación completa del funcionamiento de la plataforma, garantizando su confiabilidad y funcionalidad.

El proceso de verificación incluye también pruebas comparativas entre la plataforma emulación y la placa física, donde se ejecutaron casos de prueba idénticos en ambos entornos. Esto permite identificar y analizar las diferencias entre el comportamiento real y la del emulador web, además, proporciona 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.
LEDs	Pruebas de ejemplo.
Potenciómetro de 10K	Pruebas de ejemplo.
Analog Stick	Pruebas de ejemplo.
Termistor NTC	Pruebas de ejemplo.
Sensor de temp. y hum. DHT11	Pruebas de ejemplo.
Display LCD HD44780	Pruebas de ejemplo.
Display GLCD ST7920	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 [61]	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 [62]	Emulador de la terminal serial.

4.2. Pruebas de Unidad

Las pruebas de unidad se centran en evaluar de manera aislada cada función 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.

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

Además, para compilar las pruebas unitarias con *CMocka* o *Check*, se utiliza 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 muestran en la consola al ejecutar el archivo generado, donde se inica:

- 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/inc -lcheck -lcmocka -lrt -pthread -lm -lsubunit
Running suite(s): CMocka Suite sapi_dht11 dht11init
Mock: dht11_init invokedado con gpio = 10
100%: Checks: 1, Failures: 0, Errors: 0
Running suite(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_gpio
Running suite(s): Check Suite sapi_gpio gpioInit
Mock: gpio_init invokedado con gpioMap_t = 50
100%: Checks: 1, Failures: 0, Errors: 0
Running suite(s): CMocka Suite sapi_gpio gpioRead, gpioWrite, gpioToggle
Mock: gpio_get_0 invokedado con gpioMap_t = 50
Mock: gpio_set_0 invokedado con gpioMap_t = 50
100%: Checks: 1, Failures: 0, Errors: 0
Mock: gpio_get_1 invokedado con gpioMap_t = 50
100%: Checks: 1, Failures: 0, Errors: 0
jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ ./hal/ciaa_tests_rtc
Running suite(s): Check Suite sapi_rtc rtcInit
Mock: delay_invocado con duration_ms = 2100
100%: Checks: 1, Failures: 0, Errors: 0
Running suite(s): CMocka Suite sapi_rtc rtcRead rtcWrite
Mock: rtc_write invokedado
100%: Checks: 1, 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/inc -lcheck -lcmocka -lrt -pthread -lm -lsubunit
Running suite(s): Check Suite sapi_delay delayInaccurateNs, 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_inaccurate invokedado con int = 200
100%: Checks: 6, Failures: 0, Errors: 0
Running suite(s): CMocka Suite sapi_delay delayRead
100%: Checks: 1, Failures: 0, Errors: 0
jenny@jenny-pc:~/Documents/UBA/Tesis/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/inc -lcheck -lcmocka -lrt -pthread -lm -lsubunit
Running suite(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_tasks
Running suite(s): CMocka Suite task getTickCount
Mock: xstartTickCount invokedado
100%: Checks: 1, Failures: 0, Errors: 0
Running suite(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 identificaron 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. Desués, se crearon archivos de prueba de integración con escenarios específicos que combinan 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/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/peripherals/inc -lcheck -lcmocka -lrt -pthread -lm -lsubunit
Jenny@jenny-pc:~/Documents/UBA/Tesis/ciaa-emulador$ ./hal/ciaa_test_integration_delay_tick
Running suite(s): Test de Integración
100%: Checks: 1, Failures: 0, Errors: 0

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

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

FIGURA 4.3. Depuración de las pruebas de integración.

4.4. Pruebas de Interfaz de usuario

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 permite que se ejecuten de forma rápida y confiable de manera recurrente.

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

Esto asegura que cada componente de la interfaz funcione correctamente por separado. Incluso, se verifica si el código en el navegador web devuelve 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
Contento 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. Integración 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 implica 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, logrando mayor flexibilidad y respaldo en caso de problemas con alguna de estas herramientas.

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 expone registros detallados de la ejecución de las pruebas, que incluyen mensajes del progreso de las pruebas, información de depuración y los resultados de las mismas.

La siguiente figura 4.5 presenta la consola de *Travis CI*.

```

lsbuild
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 -pthread -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: Check Suite task xStartTickCount setTaskDelayUntil
532 Mock: xStartTickCount Invocado
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/periipherals/src/sapi_delay.c hal/test/mock/delay_api_mock.c
537 hal/test/mock/sapi_delay_mock.c -I . -I ./hal/sapi -I ./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/periipherals/inc -lcheck -lcmocka -lrt -pthread -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/periipherals/src/sapi_delay.c hal/test/mock/delay_api_mock.c
539 hal/test/mock/sapi_delay_mock.c -I . -I ./hal/sapi -I ./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/periipherals/inc -lcheck -lcmocka -lrt -pthread -lm -lsubunit" exited with 0.
540 $ ./hal/ciaa_test_integration_delay_tick
541 Running suite(s): Test de Integración
542 100%: Checks: 1, Failures: 0, Errors: 0
543 The command "./hal/ciaa_test_integration_delay_tick" exited with 0.
544 $ gcc -DTEST_BUILD -o hal/ciaa_test_integration_gpio_interrupt hal/test/check_test_integration_gpio_interrupt.c hal/sapi/soc/periipherals/src/sapi_gpio.c hal/test/mock/gpio_api_mock.c
545 hal/test/mock/gpio_interrupt_mock.c -I . -I ./hal/sapi -I ./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/periipherals/inc -lcheck -lcmocka -lrt -pthread -lm -lsubunit
546 The command "gcc -DTEST_BUILD -o hal/ciaa_test_integration_gpio_interrupt hal/test/check_test_integration_gpio_interrupt.c hal/sapi/soc/periipherals/src/sapi_gpio.c
547 hal/test/mock/gpio_api_mock.c hal/test/mock/gpio_interrupt_mock.c -I . -I ./hal/sapi -I ./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/periipherals/inc -lcheck -lcmocka -lrt -
548 pthread -lm -lsubunit" exited with 0.
549 $ ./hal/ciaa_test_integration_gpio_interrupt
550 Running suite(s): Test de Integración
551 100%: Checks: 0, Failures: 0, Errors: 0
552 The command "./hal/ciaa_test_integration_gpio_interrupt" exited with 0.
553 $ gcc -DTEST_BUILD -o hal/ciaa_test_integration_rtc_delay hal/test/check_test_integration_rtc_delay.c hal/sapi/soc/periipherals/src/sapi_rtc.c hal/test/mock/rtc_api_mock.c
554 hal/test/mock/sapi_rtc_delay_mock.c -I . -I ./hal/sapi -I ./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/periipherals/inc -lcheck -lcmocka -lrt -pthread -lm -lsubunit
555 The command "gcc -DTEST_BUILD -o hal/ciaa_test_integration_rtc_delay hal/test/check_test_integration_rtc_delay.c hal/sapi/soc/periipherals/src/sapi_rtc.c hal/test/mock/rtc_api_mock.c
556 hal/test/mock/sapi_rtc_delay_mock.c -I . -I ./hal/sapi -I ./hal/test/mock -Ihal/sapi/base -Ihal/sapi/board -Ihal/sapi/soc/periipherals/inc -lcheck -lcmocka -lrt -pthread -lm -lsubunit" exited with 0.
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. -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. -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. -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 Running after script...
949 $ echo "Pruebas unitarias completadas."
950 Pruebas unitarias completadas.
951 $ echo "Pruebas de integración completadas."
952 Pruebas de integración completadas.
953 Pruebas de integración completadas.
954 Cleaning up project directory and file based variables
955 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 *EmuCIAA*, se utiliza la herramienta Postman.

Antes de comenzar con el ensayo, se crea 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 *EmuCIAA*.

Post condiciones:

- ÉXITO: la plataforma muestra el ejemplo *blink*y en ejecución, encendiendo y apagando el LEDB.
- FALLA: La plataforma no muestra ningún ejemplo predeterminado en ejecución.

Resumen del Test:

- Después de acceder a la plataforma mediante los navegadores web Chrome, Firefox y Explorer se comprobó que se muestra en ejecución el ejemplo *blinky*.
- Se realizó la prueba de HTTP *requests* por medio de la utilización de la herramienta *Postman* que luego de acceder a la dirección 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*.

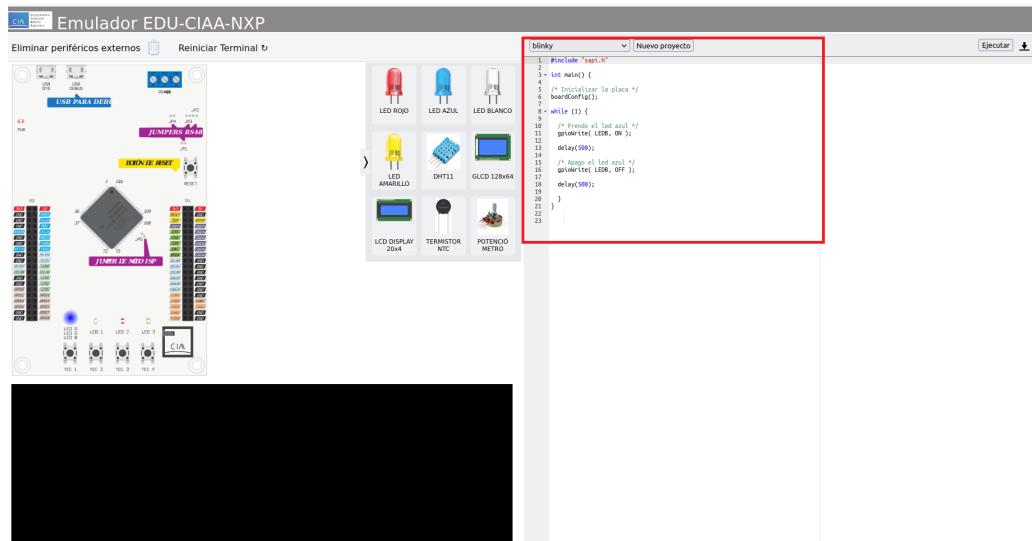


FIGURA 4.7. Prueba plataforma emulador ejecutando el ejemplo *Blinky*.

Postman es una aplicación que permite realizar pruebas de API. La herramienta permite realizar pruebas *HTTP requests* y acceder al servidor de la plataforma de emulación. Se obtiene 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*.

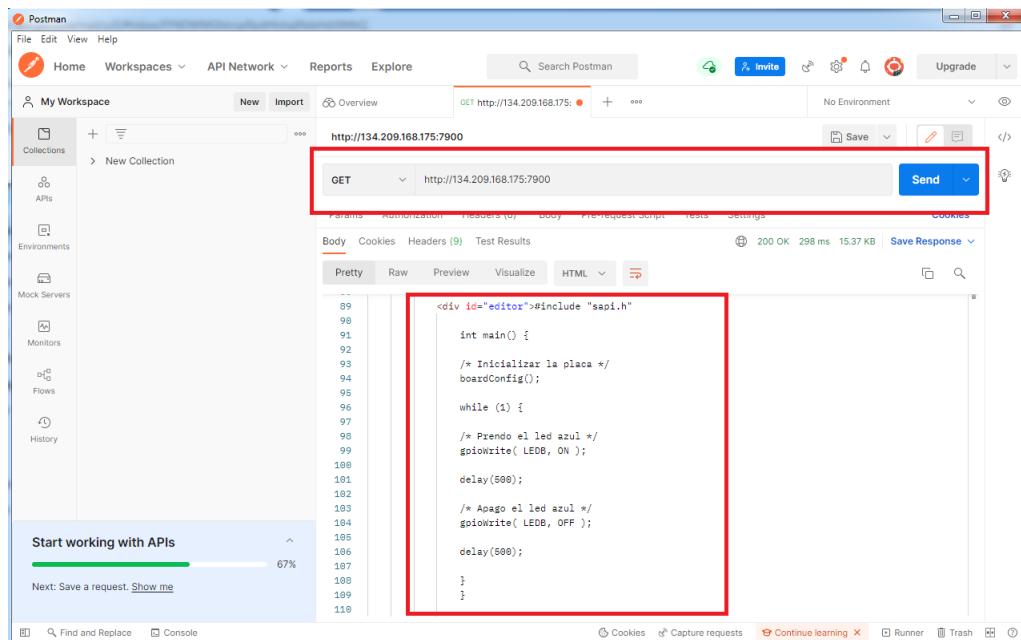


FIGURA 4.8. Respuesta del servidor.

4.6. Pruebas de funcionamiento

Para evaluar, verificar el comportamiento y desempeño de la EDU-CIAA-NXP en el entorno web, se realizan pruebas funcionales con cada uno de los ejemplos contenidos en la plataforma, que cubren la totalidad de periféricos desarrollados.

A modo de ejemplo, a continuación se exponen las pruebas funcionales con uno de los periféricos internos, en este caso, es el RTC, y las pruebas un periférico externo DHT11.

También, se realiza una prueba del ejemplo *tick_hook*, pero creando un programa utilizando el botón "Nuevo proyecto". Este ejemplo se utiliza para mostrar las diferencias de tiempo de ejecución entre la plataforma web y la placa física EDU-CIAA-NXP.

4.6.1. Prueba del ejemplo *rtc_printf*

Ensayo en EmuCIAA

Para el ensayo en la plataforma de emulación web se plantea el siguiente caso de prueba:

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 de ejemplos, 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 la UART, enviando los valores del RTC.
6. El usuario realiza la descarga del ejemplo *rtc_printf.c* 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 en la consola integrada la salida del ensayo.

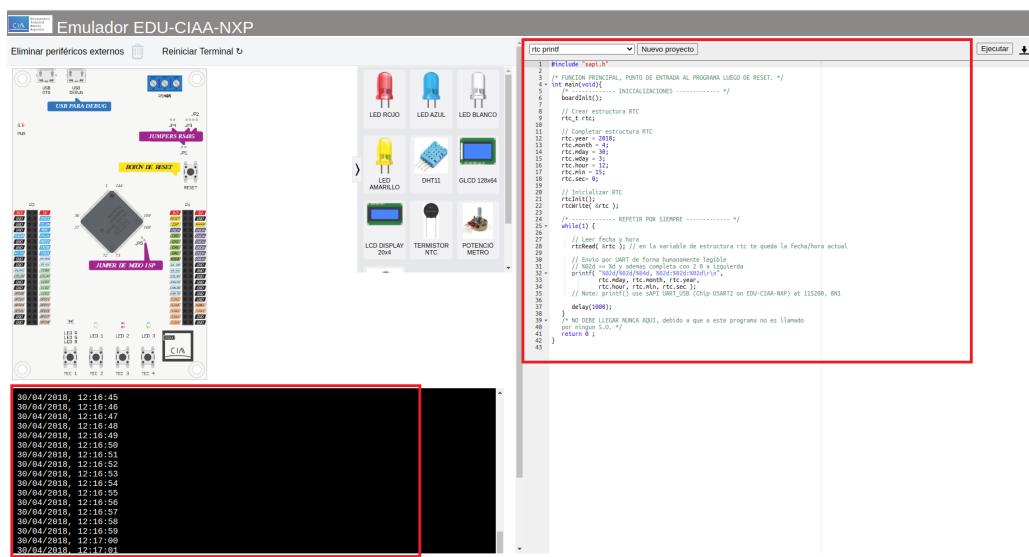


FIGURA 4.9. Salida por consola del ensayo *rtc printf*.

Ensayo en la placa física 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 4.10 expone el ejemplo *rtc_printf* importado en *eclipse*:

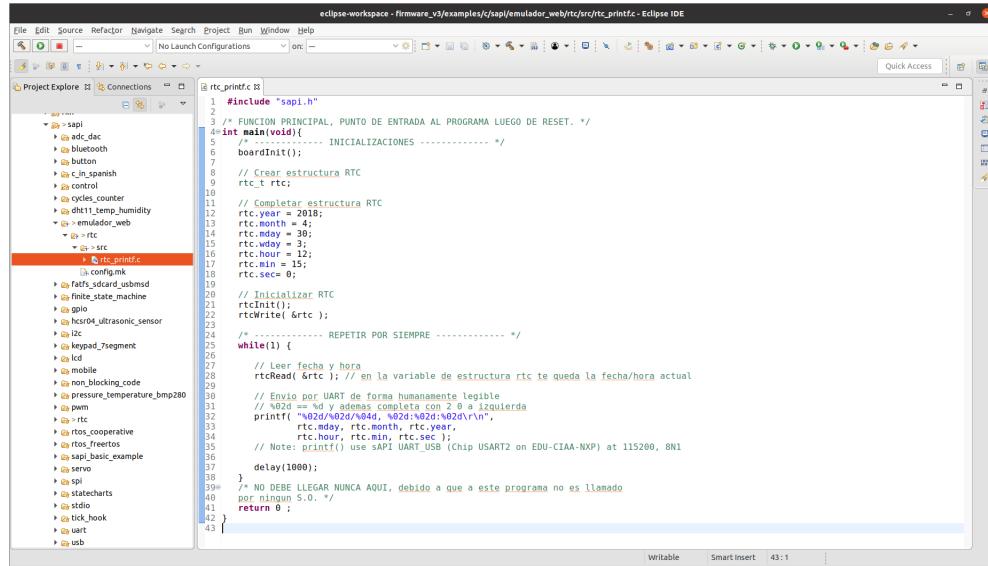


FIGURA 4.10. Ejemplo *rtc printf* importado en *eclipse*.

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 4.11:

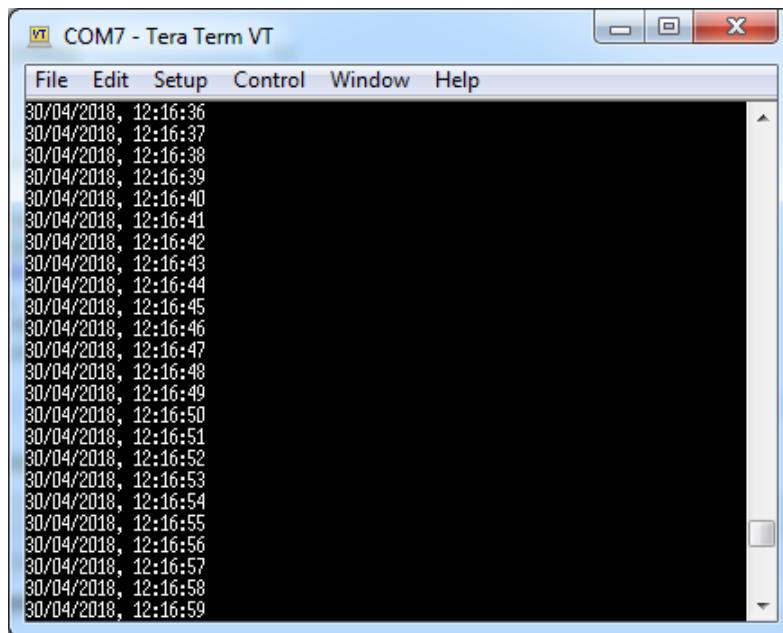


FIGURA 4.11. Salida de la terminal COM7 -Tera Term VT.

Resumen de la prueba Exitosa:

- Después de seguir los pasos del flujo principal del caso de prueba, se comprueba que se muestra en ejecución el ejemplo *rtc_printf*.
 - El ensayo en la placa física EDU-CIAA-NXP con el mismo ejemplo, obtuvo la misma salida por consola de la plataforma web.

4.6.2. Prueba del ejemplo *dht11*

Ensayo en *EmuCIAA*

Para el ensayo en la plataforma de emulación web se elabora :

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: "VCC=3V3", "DATA=GPIO1" y "GND=GND".
 - 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 cercano a la ubicación del usuario.
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 *click* 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 indicadores gráficos haciendo *click* sobre mismos, que corresponden a la temperatura y a la humedad.
7. La plataforma actualiza los indicadores gráficos 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 indicadores 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 alguna de las condiciones de éxito descriptas.

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

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

La figura 4.12 muestra la consola con los datos de temperatura y humedad del ejemplo *dht11* con la opción "Obtener datos de servidor climático local."

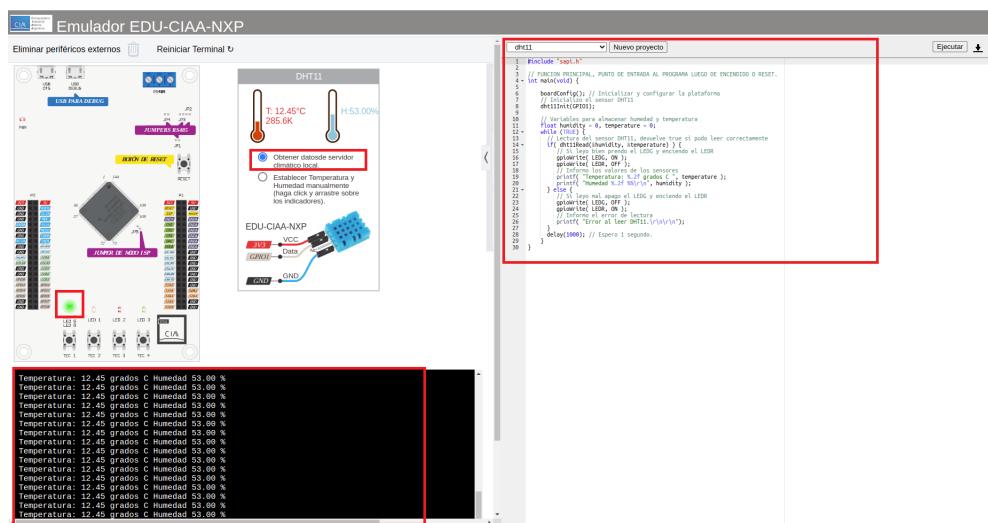


FIGURA 4.12. Resultado del CP02 con la opción "Obtener datos de servidor climático local."

La figura 4.13 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)."

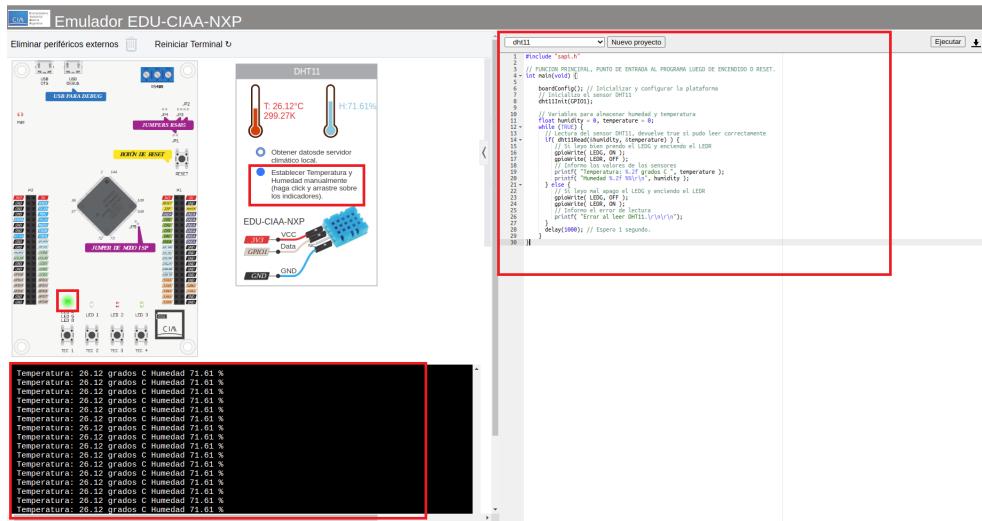


FIGURA 4.13. Resultado del CP02 con la opción "Establecer Temperatura y Humedad manualmente(haga click y arrastre sobre los indicadores)."

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

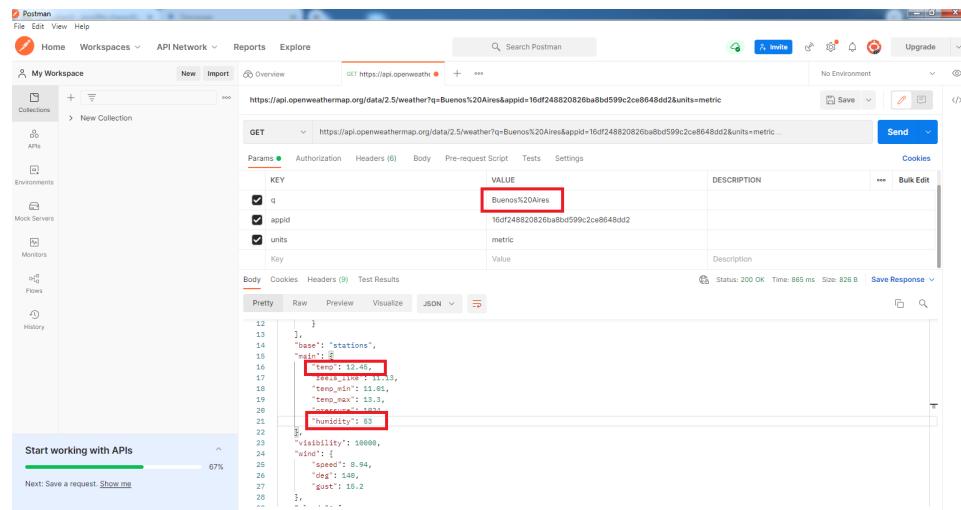


FIGURA 4.14. Petición de datos de temperatura/humedad.

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

Ensayo en la placa física EDU-CIAA-NXP

Este ensayo tiene 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. Luego, se procede a importar el archivo generado por la plataforma web "dht11_temp_humidity.c" y compilarlo dentro de la herramienta *eclipse*. Luego, se descarga a la placa física EDU-CIAA-NXP.

En la figura 4.15 se muestra la ejecución del ensayo en la plataforma EDU-CIAA-NXP.

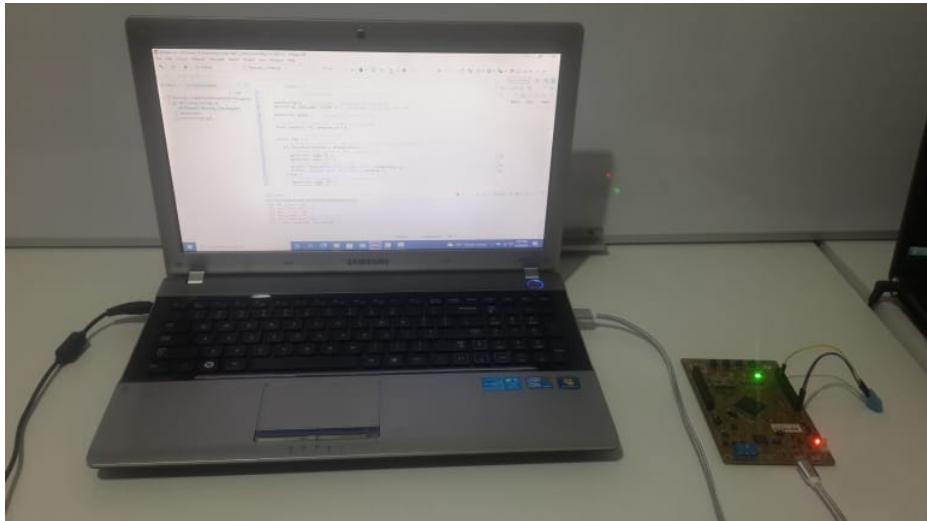


FIGURA 4.15. Ensayo en la plataforma EDU-CIAA-NXP del ejemplo *dht11*.

La figura 4.16 muestra el código del ejemplo *dht11* en la herramienta *eclipse* de la PC de prueba.

 A screenshot of the Eclipse IDE interface. The left pane shows a project structure with various source files and folders. The right pane displays the code for 'dht11_temp_humidity.c'. The code includes includes for 'sapi.h' and 'config.mk', defines the main function, initializes the DHT11 sensor, and reads its data. It then prints the temperature and humidity to the console. If the reading fails, it prints an error message and turns off the LEDG. The code is written in C and uses comments to explain its logic.


```

eclipse-workspace - firmware_v3\examples\c\api\emulador_web\dht11\src\dht11_temp_humidity.c - Eclipse IDE

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 4.16. 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 4.17 se observan los cambios en la placa EDU-CIAA-NXP que fueron registrados durante las pruebas.



FIGURA 4.17. 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 permite guardar los datos de temperatura/humedad.

En la figura 4.18 se muestra los datos de temperatura/humedad usando *Tera Term VT*.

```
VT COM7 - Tera Term VT
File Edit Setup Control Window Help
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 4.18. Salida de la terminal COM7, Tera Term VT.

Resumen de la prueba Exitosa:

- Despu  s de acceder a la plataforma mediante el navegador y siguiendo los pasos del flujo principal y el flujo alternativo de la prueba, se comprob   que se muestra en ejecuci  n el ejemplo *dht11*.
- Se realiz   la prueba de HTTP *requests* usando la herramienta *Postman* para comprobar la respuesta del *API openweathermap* que consume la plataforma de emulaci  n de manera que, los datos de temperatura y humedad sean los mismos.
- Se ensayo en la placa f  sica EDU-CIAA-NXP el mismo ejemplo *Dht11 temperature/humidity* y se registraron los resultados de la terminal serial.

4.6.3. Prueba de creaci  n de un nuevo proyecto, utilizando el c  digo del ejemplo *tick_hook*

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

Ensayo en *EmuCIAA*

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

ID Caso de prueba: CP04

Descripci  n: la plataforma de emulaci  n permite al usuario ejecutar el ejemplo *tick_hook* que se presenta en el c  digo 4.1.

```

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

C  DIGO 4.1. Nuevo proyecto, basado en el ejemplo *tick_hook*

Pre-condici  n:

- La computadora del usuario tiene conexi  n a internet y un navegador web instalado.

Flujo principal:

1. El usuario ingresa al entorno web *EmuCIAA*.
2. El usuario hace *click* en el botón "Nuevo Proyecto".
3. La plataforma muestra al usuario en el área de codificación, la pantalla para que pueda ingresar su propio código.
4. El usuario ingresa el código de prueba 4.1.
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 4.19 muestra en la consola integrada la salida de "Blinky LED3" muchas veces.

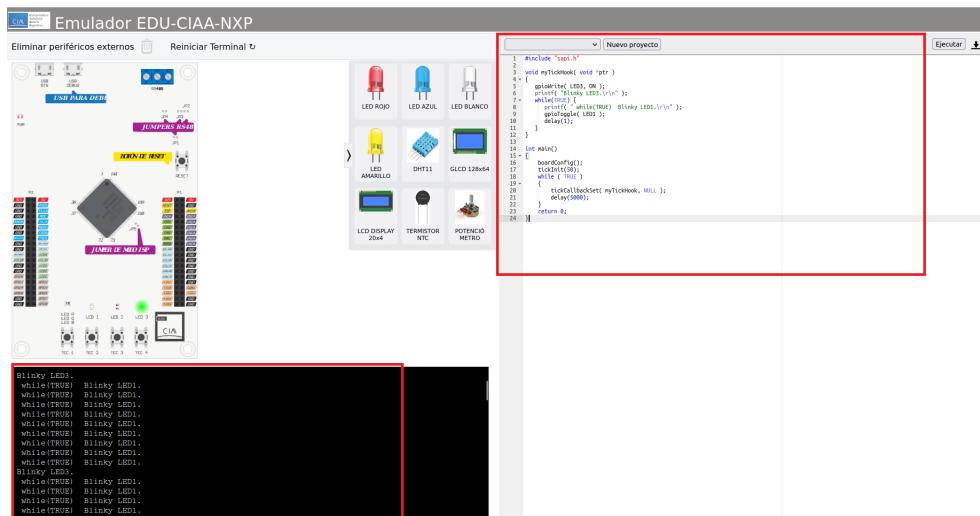


FIGURA 4.19. Salida por consola del ensayo tick hook.

Ensayo en la placa física 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 4.20:

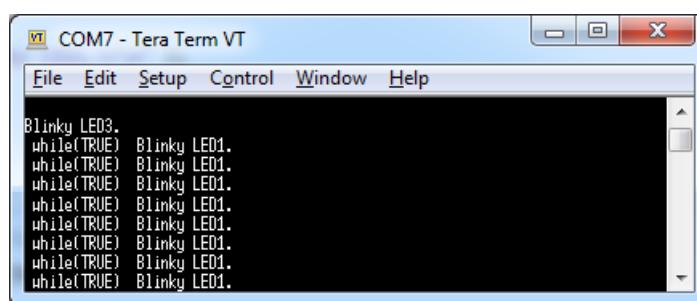


FIGURA 4.20. Salida de la terminal COM7 -Tera Term VT.

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 de *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 resulta en la repetición de ciertas porciones de código, como se ha demostrado.

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. Este tema de manejo fino de tiempos y performance queda para trabajo a futuro.

Se concluye entonces que para el uso en enseñanza de programación de Sistemas Embebidos la plataforma *EmuCIAA* cumple con los requerimientos establecidos.

Capítulo 5

Conclusiones

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

5.1. Objetivos alcanzados

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

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

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

5.2. Próximos pasos

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

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

Bibliografía

- [1] Proyecto CIAA. *Plataforma educativa del Proyecto CIAA*. Visitado el 2023-06-15. 2014. URL: <http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:edu-ciaa:edu-ciaa-nxp>.
- [2] 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] Mocha Working Group. *Mocha*. Visitado el 2022-03-14. 2022. URL: <https://mochajs.org/>.
- [49] Chai Working Group. *Chai*. Visitado el 2022-03-14. 2022. URL: <https://www.chaijs.com/>.
- [50] Roman Zimbermann. *Check*. Visitado el 2023-06-14. 2000. URL: <https://libcheck.github.io/check/>.
- [51] Andreas Schneider y Jan Kratochvil. *CMocka*. Visitado el 2022-03-14. 2009. URL: <https://cmocka.org/>.
- [52] Ben Uretsky, Moisey Uretsky, Mitch Wainer, Jeff Carr y Alec Hartman. *DigitalOcean*. Visitado el 2023-06-15. 2011. URL: <https://www.digitalocean.com/>.
- [53] 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.
- [54] Comunidad Proyecto CIAA. *Firmware V3 r130*. Visitado el 2023-03-14. 2023. URL: https://github.com/epernia/firmware_v3/releases.
- [55] 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.
- [56] Comunidad Proyecto CIAA. *Firmware v2*. Visitado el 2023-07-29. 2018. URL: https://github.com/ciaa/firmware_v2.
- [57] Comunidad Proyecto CIAA. *seos_pont*. Visitado el 2023-03-14. 2023. URL: https://github.com/epernia/firmware_v3/tree/master/libs/seos_pont.
- [58] Comunidad Proyecto RTOS. *FreeRTOS*. Visitado el 2023-03-14. 2023. URL: <https://www.freertos.org/index.html>.
- [59] 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>.
- [60] Jenny Chavez. *Emulador EDU-CIAA*. Visitado el 2023-06-24. 2023. URL: <https://github.com/jenniferch/ciaa-emulador>.
- [61] Postman Working Group. *Postman*. Visitado el 2022-03-24. 2022. URL: <https://www.postman.com>.

- [62] T. Teranishi. *Tera Term VT*. Visitado el 2022-03-24. 2022. URL: svn.osdn.jp/svnroot/ttssh2/trunk/.