Emuladores de sistemas embebidos dentro de contenedores

Esteban Carnuccio[1], Waldo Valiente[1], Mariano Volker[1], Matías Adagio[1], Micaela Antelo[1]

[1] Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas
Florencio Varela 1903 - San Justo, Argentina
{ecarnuccio, wvaliente, mvolker, maadagio, mantelo}@unlam.edu.ar
www.unlam.edu.ar

Resumen. En la educación de sistemas embebidos, es necesario que el estudiante interactúe con ellos, con el fin de poder completar su aprendizaje. Para esto, una manera es a través del uso de emuladores. Los cuales permiten el contacto con el embebido de forma similar a su empleo físico. En este sentido el presente artículo, expone los trabajos que se realizaron para plantear las bases que permitan emular mediante QEMU, las placas de desarrollo: Raspberry PI, ESP32 y STM32F103C8T6. Estas se ejecutan dentro de contenedores Docker. De manera tal, que los contenedores de las imágenes de los sistemas embebidos permitan realizar pruebas en un entorno estandarizado. De forma, que las aplicaciones del embebido puedan funcionar en un entorno virtual. Así se les podrá ofrecer a los estudiantes una herramienta con la cual puedan realizar sus trabajos sin tener la necesidad de incurrir en costos.

Palabras Clave. Emuladores, Docker, Sistemas Embebidos, QEMU

1 Introducción

En la actualidad existen varias iniciativas y proyectos educativos, que buscan enseñar las tecnologías que componen los Sistemas Ciber Físicos (CPS), término del inglés *Cyber-Physical Systems*. El objetivo de los CPS se basa principalmente en las interacciones dadas por sus componentes y el entorno, junto con las funciones de control y los mecanismos de comunicación entre ellos. Es por ello por lo que en esta investigación se orienta en la emulación de sistemas embebidos (SE), con conexión al exterior. Los destinatarios de los SE son los estudiantes universitarios y de postgrado. La fundamentación de lo antes descripto es recapitulada en el estudio sobre este tema [1]. En ella se diferencian los principales enfoques sobre los tópicos de gestión de proyecto, el diseño del sistema o las técnicas de redes. Además, resalta el uso de una plataforma flexible que ejecute un sobre SE con conectividad de red. Ya que, la conexión es una característica vital, debido a que el SE utiliza diferentes esquemas, que le permite comunicarse con otros dispositivos. Para esto se utilizan mecanismos tales como Wifi, Bluetooth,

Ethernet, entre otros. De esta manera, el SE puede intercambiar datos con servidores u otros dispositivos. Para así formar una topología de computación en la nube [2] [3].

Para gestar esta idea, los contenedores Docker son una de las herramientas que ayudan en la construcción, gestión y pruebas de las complejas topologías que componen estos sistemas. Gracias a que los contenedores, brindan un ambiente aislado que permiten trabajar con paquetes de software sin utilizar virtualización del hardware.

1.1 Docker

Para explicar el funcionamiento de Docker se utilizará la metáfora explicada en [4]: “Antes los trabajadores encargados de mover mercancías comerciales dentro y fuera de los barcos en el puerto (estibadores) requerían de habilidades especiales. Las mercancías eran de diferentes tamaños y formas. Los experimentados estibadores eran apreciados por su capacidad para acomodar los distintos tipos de mercancías dentro de los barcos. Contratar a estas personas no era económico, pero hacían un trabajo realmente eficiente. Como alternativa, surgieron los contenedores marítimos, que son cubos paralelepípedos de iguales proporciones, que permiten simplificar la carga y descarga de los barcos”. Dichos contenedores admiten que esas mercancías, que poseen formas irregulares, se guarden desde el origen antes de llegar al puerto. Como los contenedores poseen un tamaño estándar, los barcos cargan y descargan mucho más rápido, incluso en forma automatizada. Esta metáfora es conocida en el ámbito de proyectos de software, porque se invierte mucho tiempo y energía en conseguir software heterogéneo (mercancías). Estos se integran de formas complejas en el sistema (barco). Gracias a Docker, permite que los diferentes sistemas, involucrados en el proceso de desarrollo, hablen un mismo idioma. Haciendo de esta manera, que trabajar en la integración de diferentes sistemas sea más sencillo. Ya que cada imagen Docker, es generada y mantenida como una caja negra.

Internamente Docker se compone de imágenes. Cuando se instancia esta, se asocia a un nuevo contenedor. Por lo que se pueden generar varios contenedores a partir de la misma imagen. En la figura (Fig. 1) se muestra la relación entre dos contenedores, que provienen de la misma imagen, con el sistema operativo (SO) en donde ejecutan.

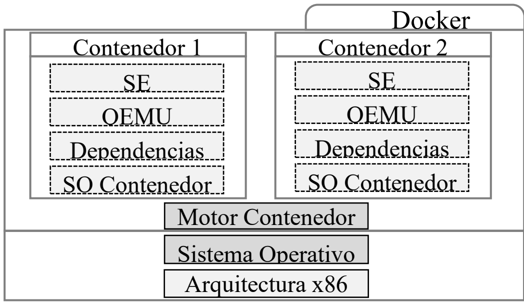

Fig. 1 *– Docker con el emulador del SE.*

Como se observa en la figura anterior, en la base de la pila se encuentra la arquitectura de hardware, que pertenece a la computadora. Sobre este funciona el Sistema Operativo Anfitrión y por encima del mismo el motor de los contenedores Docker. Dentro del contenedor se apilan las capas que forman la imagen. En la parte inferior se encuentra instalado el *kernel* mínimo del SO del contenedor (por ejemplo, Ubuntu). Luego se instalan las dependencias necesarias. Debido a que los SE suelen poseer una arquitectura hardware diferente a la que utiliza una computadora, que es del tipo x86. Para ejecutarlos se necesita de un programa que permita su funcionamiento. En esta investigación se eligió al emulador QEMU. Por eso la capa siguiente, de la imagen del contenedor, se instala QEMU, que se encuentra ya preparado y configurado para que emule los SE. Finalmente, en la capa superior se instala las herramientas propias para la construcción y uso del SE.

1.2 QEMU

QEMU (*Quick Emulator*) fue publicado inicialmente por Fabrice Bellard en 2003 [5]. En esta publicación se detalla su objetivo inicial: "*QEMU logra una emulación rápida del espacio de usuario en Linux en x86 y PowerPC, mediante la traducción dinámica. Su objetivo principal es lograr ejecutar el proyecto Wine en arquitecturas que no sean x86*". Lo anterior explica en forma simple y concisa su funcionamiento. Actualmente QEMU mantiene la idea inicial, con el agregado de diferentes tipos de arquitecturas. Es gratuito y de código abierto, pudiendo alcanzar un rendimiento nativo. Ya que el código se traduce a medida que se procesa. En la figura (Fig. 2), se ilustra que QEMU ejecuta como un programa desde la plataforma Anfitrión o *host*, por lo general, una máquina x86. Por el otro lado, el SE ejecuta dentro de la emulación que brinda QEMU, desde una arquitectura destino o *target*. Para unir estos dos mundos, QEMU usa el módulo TGC (*Tiny Code Generator*). El TCG permite la traducción dinámica del conjunto de instrucciones emuladas desde *target*, para que sean ejecutadas por las instrucciones que entiende el *host*. Esta traducción consiste en buscar las secuencias cortas de código de la arquitectura de origen, los traduce a la arquitectura de destino y captura las secuencias resultantes [6]. Lo antes dicho, es solo una de las muchas funciones que brinda el emulador. Ya que, la ejecución del SE es más complejo, intervienen lo referente al acceso a las regiones de memoria, el acceso al mapeo de los dispositivos de E/S, los temporizadores, los controladores, las interrupciones, depuración y estadísticas.

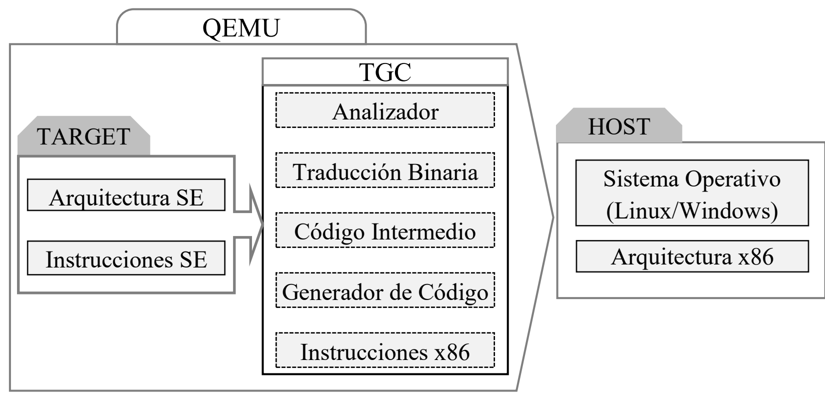

Fig. 2 - Emulación de SE con QEMU.

A continuación, se describirán los principales trabajos de SE previos a esta investigación, como así también de las emulaciones que realizan.

2 Trabajos relacionados

En [7] se desarrolló un laboratorio virtual usando al emulador QEMU dentro de contendedores Docker. El cual permitía emular Raspberry pi y ESP32. Ese trabajo de investigación se centró en realizar la comunicación entre varias placas ESP32 y Raspberry pi emuladas. Para realizar su conexión virtual se utilizó el protocolo REST[1] y MQTT[2]. No obstante, la comunicación solo fue realizada en forma interna, dentro de la red virtual, por lo que no se generan comunicación con servidores externos. Al mismo tiempo, este trabajo se encuentra acotado, debido a que las placas ESP32 que genera QEMU no permite emular sensores y actuadores. Según el autor, esto se debe a que utilizó la versión de Qemu que generó el proyecto de Espressif.

Por otro lado, en [8] se diseñó e implementó un laboratorio virtual en la nube, el cual consiste en dos partes: Una parte cliente y otro servidor. En donde la primera, está compuesta por Raspberry Pi físicas. Mientras que la segunda se encuentra en servidores en la nube, los cuales ejecutan contenedores Docker con el emulador QEMU instalado. Allí el estudiante puede ejecutar determinados sistemas emulados remotamente. Pero este trabajo no contempla la utilización en la comunicación de los dispositivos emulados con elementos externos, tales como Smartphones.

Por ese motivo en este trabajo de investigación, se plantean las tareas iniciales que se llevaron a cabo para construir las bases de una plataforma de sistemas embebidos emulados dentro de contenedores Docker. Los cuales permitirán emplear a diferentes SE, tales como ESP32, STM32 y Raspberry, ejecutándose sobre QEMU. Para que posteriormente, en próximos trabajos, estas puedan comunicarse con el exterior, y de esta forma permitan intercambiar datos con aplicaciones móviles.

[1] Interfaz de comunicación entre cliente y servidor

2 Protocolo de comunicación enfocado a la conectividad Machine-to-Machine (M2M)

En este sentido el presente documento se divide en las siguientes secciones: Primero, se describen las diferentes características que presentan los SE de la investigación. Luego se explica el proceso de configuración, instalación y creación, que se ha seguido para construir las imágenes Docker. Finalmente se explica un ejemplo de comunicación externa utilizando la Raspberry Pi emulada.

3 Desarrollo

Cuando se trata de elegir una plataforma de sistemas embebido, existen una amplia variedad de SE compactos de bajo presupuesto. Con estos se pueden realizar proyectos con fines educativos, comercial, por entretenimiento a la electrónica o la programación. Todas ellas son diseñadas para diferentes mercados. No obstante, su elección depende de las especificaciones técnicas, que se dividen en características o cantidad de conexiones que posean.

3.1 Especificaciones técnicas de los sistemas embebidos

	ESP32	**STM32-*Blue Pill***	**Raspberry Pi 2**
Imagen			
Características			
SoC	Tensilica Xtensa	STM32F103C8T6	Broadcom BCM2836
Procesador	Tensilica Xtensa LX6	ARM Cortex-M3	ARM Cortex-A7
Núcleos	2	1	4
Dimensiones [mm]	48 x 25,5	53 x 23	86 x 56
Voltaje de trabajo [v]	3.3	3 - 5.5	5
Frecuencia del reloj [Hz]	240 M	72 M	900 M
Memoria interna [bytes]	512 K	128 K	1 G
Conexiones			
Conectores GPIO	34	37	40
I^2C	2	2	6
USB	✓	✓	✓
Bluetooth	✓	✗	✓
Ethernet	✓	✗	✓
Wifi	✓	✗	✓
Salida a monitor	✗	✗	✓

Tabla 1 - Comparación de Sistemas Embebidos utilizados.

3.1.1 Características de ESP32

Salió al mercado a fines del 2016 [9], está constituido en arquitectura System On Chip (SOC) diseñado por Espressif Systems. El microcontrolador más difundido entre los modelos, incluido en la placa de desarrollo, es el ESP-WROOM-32. También cuenta con el chip CP2102N, que permite la transferencia por USB. Además, posee dos pulsadores de *reset* y *boot* [10].

3.1.2 Características de STM32

También conocida como *BluePill* [11], es el apodo que se le da a la placa de desarrollo STM32F103. Esta posee un microcontrolador ARM Cortex-M3 de 32 bits. La placa posee 40 pines, un botón de *reset*, dos LED (uno de estado y otro de encendido), 2 conectores puentes (para configurar el modo de trabajo). Como conexión externa, solo posee puerto USB.

3.1.3 Características de Raspberry Pi 2

Tiene su origen en 2012, se diseñó como una minicomputadora de bajo costo. De forma tal, que le sea fácil de transportar a los estudiantes. En ella funciona un Sistema Operativo tipo Linux, pueden ejecutar aplicaciones avanzadas como suites ofimáticas, editores de fotos, así como aplicaciones de servidor web Apache. Es por ello, que se la puede conectar al monitor, teclado y mouse [12]. El nuevo modelo Raspberry Pi 4 modelo B, salió en junio de 2019. Posee dos puertos micro HDMI para conectar hasta dos pantallas. También incluye los puertos USB para conectar periféricos. Por el lado de la conectividad al exterior, incorpora un puerto Gigabit Ethernet, un módulo WiFi y otro de Bluetooth. No obstante, la versión que incluye la emulación es para la Raspberry Pi 2, las principales diferencias con el modelo actual es que soporta un solo puerto HDMI. Además, posee una menor frecuencia de reloj y memoria. Que son recursos más que suficientes para los proyectos incluidos.

3.2 Proceso de creación

La construcción del entorno para los SE, difiere ya que cada uno posee sus propias herramientas de compilación. Si bien todos usan QEMU, este también tiene diferentes versiones, ya que surgen como proyectos alternativos a la línea base del emulador. En la Fig. 3 se detalla las capas que componen las tres imágenes Docker generadas en esta investigación.

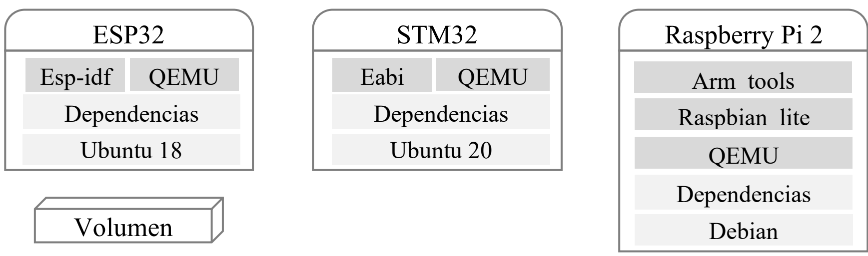

Fig. 3 - Composición de las imágenes ESP32 (a), STM32 (b) y Raspberry Pi 2(c)

3.2.1 Imagen de ESP32

Para armar el ambiente del ESP32, se implementó una imagen de Docker que posee las herramientas de compilación y el emulador QEMU. La selección de utilizar una versión anterior del Sistema Operativo base del contenedor, se debe a compatibilidad, requerida por el proyecto de QEMU que se construyó con dicha versión. Incluso se debió mantener como dependencias las dos versiones del lenguaje Pyhton (2.3 y la 3). Para el conjunto de herramientas de compilación y despliegue se utiliza el proyecto esp-idf [13]. Mientras que el emulador QEMU (versión 2.7.0), es un proyecto independiente del usuario Ebiroll [14]. Para funcionar, el emulador ejecuta directamente el grupo de ROMs que forman el bootloader, las funciones núcleo y la lógica del programa principal. Para trabajar en el proyecto se utiliza la técnica de volumen de Docker. En ella se permite trabajar con una estructura de archivos, que permanecerán consistente, aunque el contenedor se destruya, incluso permite usar el mismo volumen entre distintos contenedores. Para construir los proyectos con esp-idf para ESP32, se debe realizar la configuración inicial, desde la interfaz *menuconfig* y luego compilarlo. El resultado de la etapa de compilación es un binario con la lógica del programa. Este binario, se lo puede grabar directamente en el dispositivo físico o, como en nuestro caso, se lo ejecuta desde el emulador.

3.2.2 Imagen de STM32

Esta investigación se basó en el trabajo realizado por [15], para poder implementar la emulación del SE *BluePill* sobre QEMU. Como ese proyecto se encuentra desarrollado para emular otra placa de la misma familia, pero de distinto tipo al STM32F10C8T6 utilizado, fue necesario realizar adaptaciones en su código fuente. Por lo tanto, debido a las modificaciones realizadas y para facilitar el uso de la emulación de esta placa de desarrollo, se creó un repositorio GitHub propio, con las adaptaciones que permiten formar a la imagen Docker para la *BluePill* [16]. De esta manera, dentro del contenedor asociado a la misma, se encuentra todo el entorno de trabajo configurado para poder emular esta placa fácilmente. A su vez, contiene código fuente de distintos ejemplos. Los cuales le permiten al usuario probar los diferentes componentes que ofrece QEMU, durante la emulación de la *BluePill*.

3.2.3 Imagen de Raspberry Pi 2

Para implementar la emulación de la Raspberry Pi, se hizo uso de la imagen de Docker creado por [18]. El cual posee instalado una versión QEMU, configurada para poder ejecutar una emulación de Raspberry Pi 2. Dentro del emulador se ejecuta una versión minimalista del Sistema Operativo oficial de este embebido, denominado *Raspbian Lite.* Esta versión no posee instalada su interfaz gráfica (GUI), sino que solamente se puede acceder a su línea de comandos. Desde allí se puede controlar la Raspberry emulada.

Por otro lado, dentro de un contenedor Docker no se pueden ejecutar aplicaciones gráficas. Esto se debe a que no está preparado por defecto para ello. Sin embargo, la imagen creada por [18], incorpora los cambios en la configuración para poder realizarlo. De esta forma, se puede ejecutar una versión adaptada de Debian en forma gráfica. Dentro de la cual, a su vez se ejecuta la versión de QEMU antes mencionado. Esto se puede visualizar en la (Fig. 3-c). Esta característica es de mucha utilidad, para poder ejecutar herramientas gráficas dentro del contenedor Docker.

Cuando se crea el contenedor, este automáticamente crea un servidor que permite acceder a través de un navegador web a la interfaz gráfica del SO Debian. De esta forma, por medio del localhost, se puede controlar el S.O gráfico y al emulador Qemu, en donde se ejecuta Raspbian.

Otra característica de este contenedor es que permite acceder al sistema de archivos del Raspbian Lite vía ssh. Lo que facilita el trabajo en el emulador desde el host anfitrión.

3.3. Ejemplo de GPIO y API REST en Raspberry Pi 2

Empleando la imagen de la Raspberry Pi, anteriormente mencionada, se realizaron pruebas iniciales para ejecutar ejemplos de script de Python. Estos programas tienen como funcionalidad probar desde la emulación la comunicación con servidores externos, empleando el protocolo REST. En dichos scripts se realizaron peticiones GET y POST a un servidor en la nube. Lo que genero muy buenos resultados.

Por otra parte, también se realizó la prueba de la emulación de los puertos GPIO de la Raspberry Pi. Para ello se hizo uso de las bibliotecas provistas por [19] , que permiten emular los puertos. No obstante, no se pudo hacer funcionar esta biblioteca dentro de QEMU. Por lo que se probó su funcionamiento dentro del contenedor, pero sobre el S.O Debian. Esto se realizó para poder aprovechar las pantallas GUI que genera dicho código. La cual permite visualizar en forma gráfica los pines GPIO. De esta manera se puede interactuar con los mismos. En la siguiente figura se muestra dicha pantalla.

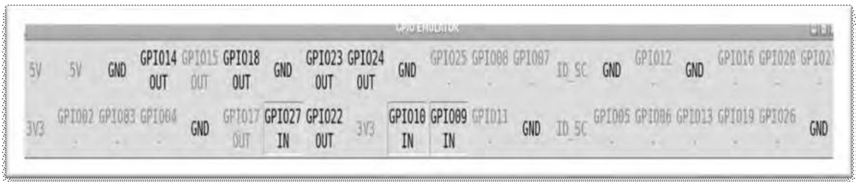

Fig. 4 - Interfaz gráfica GPIO.

4 Trabajos futuros

Se plantean tres líneas de investigaciones a desarrollar:

- La primera, en la cual se estima implementar la conectividad de los tres sistemas embebidos a un Gateway para lograr una topología compleja de internet de las cosas.
- De los tres sistemas embebidos listados en la presente investigación, el STM32 BluePill no presenta conectividad al exterior. Por lo tanto, se está analizando la factibilidad de implementar algún módulo que permita la conectividad mediante un puerto serial a través del SO anfitrión.
- Mientras que en la tercera se pretende realizar interfaces gráficas que faciliten su uso.

5 Conclusiones

En este trabajo se empezó a construir un entorno de emuladores de distintas placas de desarrollo. La cual se sustenta en el objetivo de facilitar las actividades de los estudiantes durante su formación académica. Para ello se está haciendo uso del emulador QEMU, dentro de contenedores Docker. De forma tal que los estudiantes puedan realizar prácticas con sistemas embebidos ESP32, STM32 (BluePill) y Raspberry Pi 2. A pesar de que su configuración es diferente, gracias al uso de los contenedores, se pretende que posea una rápida utilización para el usuario, de forma tal que lo opere fácilmente. A si mismo este trabajo sienta las bases, para en futuros trabajos realizar la interconexión con dispositivos externos, tales como Smartphone.

6 Referencias

[1] S. Martin, Teaching and Learning Advances on Sensors for IoT, Basel, Suiza: MDPI, 2021, pp. 10-27.

[2] P. Waher, Learning Internet of Things, Packt Publishing, 2015, pp. 2-5.

[3] Sachan, Internet de las cosas (IoT) y sus aplicaciones, Amazon Digital, 2020, p. 15.

[4] G. Sébastien, Docker Cookbook: Solutions and Examples for Building Distributed Applications, O'Reilly, 2015.

[5] F. Bellard, «QEMU x86 emulator version 0.1,» 2003. [En línea]. Available: https://www.winehq.org/pipermail/wine-devel/2003-March/015577.html.

[6] A. Höller, A. Krieg, T. Rauter, J. Iber y C. Kreiner, «QEMU-Based Fault Injection for a System-Level Analysis of Software Countermeasures Against Fault Attacks,» 2015 Euromicro Conference on Digital System Design (DSD), Madeira, Portugal, 2015, pp 4.

[7] J.-P. Ernits, «VIRTUAL IOT LAB FOR EMBEDDED SOFTWARE DEVELOPMENT FOR ESP32 AND RASPBERRY PI BASED DEVICES,» Tallin University Of Technology, 2020.

[8] G. Song, Y. Nie, G. Chen y Y. Tong, «Design and Implementation of virtual simulation experiment platform for computer specialized courses,» de *Journal of Physics: Conference Series*, 2020., pp. 1-7

[9] Á. B. Herranz, «Desarrollo de aplicaciones para IoT con el módulo ESP32,» Universidad de Alcalá, Alcalá de Henares, 2019, pp 15-20.

[10] Espressif Systems, «ESP32 Series Datasheet,» Espressif Systems, China, 2022, pp. 48

[11] STMicroelectronics, «STM32F103x8 DataSheet,» STMicroelectronics, 2015.

[12] Raspberry Pi, «Raspberry Pi 4 Model B DATASHEET,» Raspberry Pi, Pencoed, England, 2019.

[13] Espressif, «GitHub - espressif / esp-idf: Espressif IoT Development Framework,» 2022. [En línea]. https://github.com/espressif/esp-idf.

[14] Ebilroll, «GitHub - Ebiroll: Add tensilica esp32 cpu and a board to qemu and dump the rom to learn more about esp-idf,» 2021. [En línea]. Available: https://github.com/Ebiroll/qemu_esp32.

[15] Beckus, «beckus-qemu stm32» 2018. [En línea]. Available: http://beckus.github.io/qemu_stm32/.

[16] SOAUnlam, «soa-emulador-bluepill» 2022. [En línea]. Available: https://github.com/soaunlam2021/emulador-stm32-Bluepill.

[17] W. Gay, Beginning STM32: Developing with FreeRTOS, libopencm3 and GCC 1st ed. Edición, Berkley, Estados Unidos: Apress, 2018.

[18] M. Ambass, «deskctopcontainers raspberrypi,» 2017. [En línea]. Available: https://hub.docker.com/r/desktopcontainers/raspberrypi.

[19] nosix, «nosix-emulator» 2021. [En línea]. Available: https://github.com/nosix/raspberry-gpio-emulator.