

Librepass

Informe técnico

Un trabajo presentado para la materia de
Proyectos y Diseño Electrónico



Krapp Ramiro

Instituto tecnológico San Bonifacio
Departamento de electrónica

Hecho en L^AT_EX

Índice

1. Introducción	2
1.1. El proyecto y el Software Libre	2
1.2. Licencia	3
2. Diagrama en Bloques	4
3. Diagrama Esquematico	5
4. PCB	6
5. Partes del proyecto	7
5.1. DOIT ESP32 DevKit v1	7
5.1.1. Características Técnicas	7
5.1.2. Pinout	8
5.1.3. Usabilidad de pines	9
5.2. Sistema RFID	10
5.2.1. Pinout del dispositivo	11
5.2.2. Mapeo de Memoria del tag RFID	12
5.3. Pantalla OLED SSD1306	13
6. Gabinete diseñado en 3D	14
7. Frameworks	15
7.1. Arduino	15
7.1.1. Razones por las que usé Arduino	15
7.2. Flask	16
7.2.1. Razones por las que usé Flask	16
7.3. Bootstrap	16
7.3.1. Razones por las que usé Bootstrap	16
7.4. Amazon Web Services (AWS)	17
7.4.1. Razones por las que usé AWS	17
7.5. PostgreSQL	17
7.5.1. Razones por las que usé PostgreSQL	17
8. Protocolos de comunicacion	18
8.1. El protocolo de comunicación SPI	18
8.1.1. Ventajas	20
8.1.2. Desventajas	21
8.2. Protocolo I2C	22
9. Bibliografia	23

El índice tiene hipervínculos incorporados! Toca en cada sección y automáticamente tu lector de pdfs te llevará a esa página



1. Introducción

Librepass es un sistema FOSS(Free and Open Source) de seguridad para empresas. Al ser FOSS, su código esta publicamente accesible en [un repositorio público en GitHub](https://github.com/KrappRamiro/librepass) — <https://github.com/KrappRamiro/librepass>, y su uso y replicación es completamente libre y gratuito.

Este proyecto cuenta con 2 partes principales:

1. Un equipo de control de acceso, que se instala individualmente en cada puerta
2. Un sistema en la nube para gestionar los equipos y los empleados, ademas de poder ver los accesos.

1.1. El proyecto y el Software Libre

En el desarrollo de este proyecto, se planteó usar la filosofía del software libre. Segun GNU [22]:

“«Software libre» es el software que respeta la libertad de los usuarios y la comunidad. A grandes rasgos, significa que los usuarios tienen la libertad de ejecutar, copiar, distribuir, estudiar, modificar y mejorar el software. Es decir, el «software libre» es una cuestión de libertad, no de precio. Para entender el concepto, piense en «libre» como en «libre expresión», no como en «barra libre». En inglés, a veces en lugar de «free software» decimos «libre software», empleando ese adjetivo francés o español, derivado de «libertad», para mostrar que no queremos decir que el software es gratuito.

Puede haber pagado dinero para obtener copias de un programa libre, o puede haber obtenido copias sin costo. Pero con independencia de cómo obtuvo sus copias, siempre tiene la libertad de copiar y modificar el software, incluso de vender copias.

(...)

Un programa es software libre si los usuarios tienen las cuatro libertades esenciales:

- La libertad de ejecutar el programa como se deseé, con cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa, y cambiarlo para que haga lo que se deseé (libertad 1). El acceso al código fuente es una condición necesaria para ello.
- La libertad de redistribuir copias para ayudar a otros (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a terceros (libertad 3). Esto le permite ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones. El acceso al código fuente es una condición necesaria para ello.

Un programa es software libre si otorga a los usuarios todas estas libertades de manera adecuada. De lo contrario no es libre. Existen diversos esquemas de distribución que no son libres, y si bien podemos distinguirlos en base a cuánto les falta para llegar a ser libres, nosotros los consideramos contrarios a la ética a todos por igual.”



1.2. Licencia

Se escogió usar la licencia MIT [23], la cual, en inglés, es la siguiente:

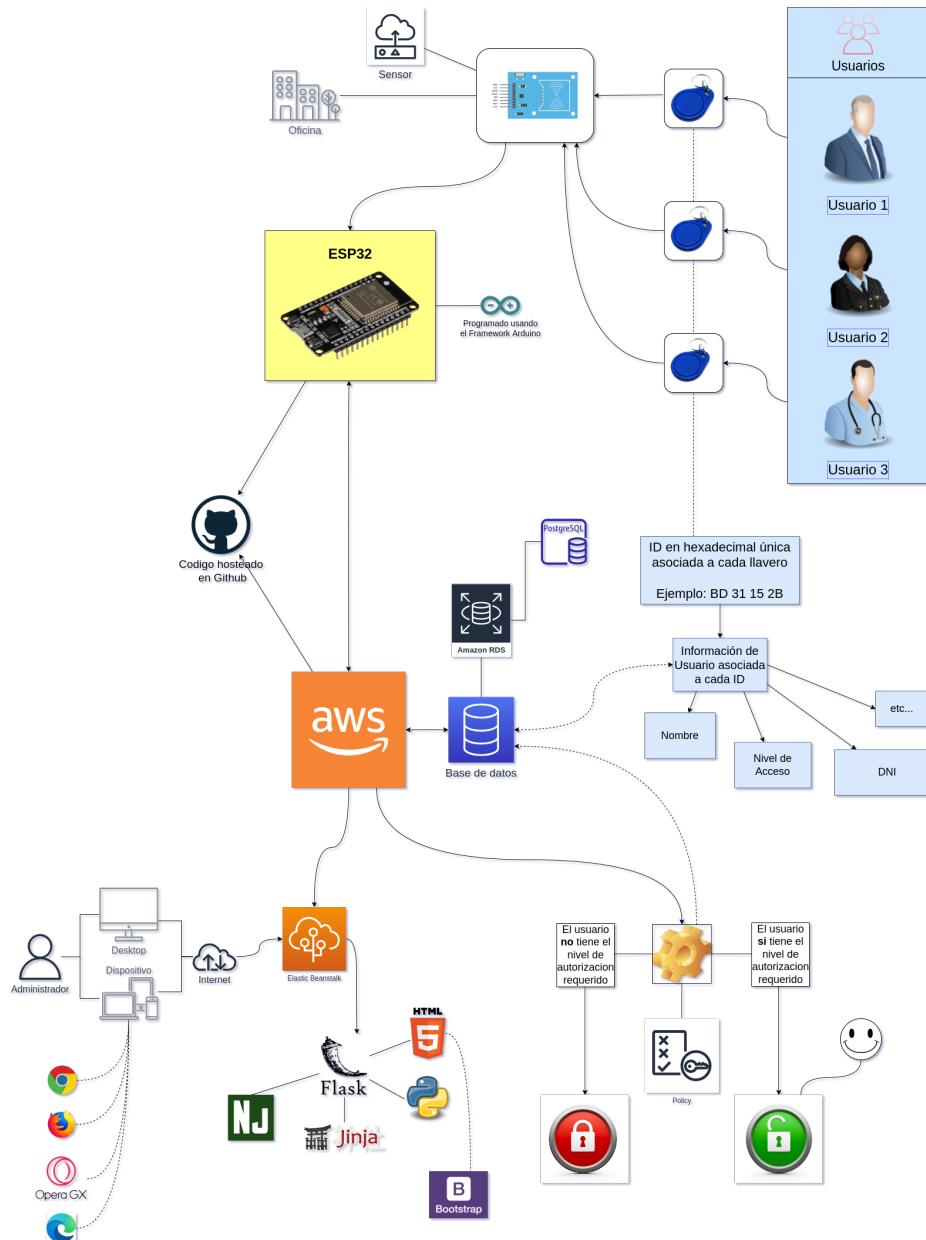
Copyright (c) 2022 Krapp Ramiro

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

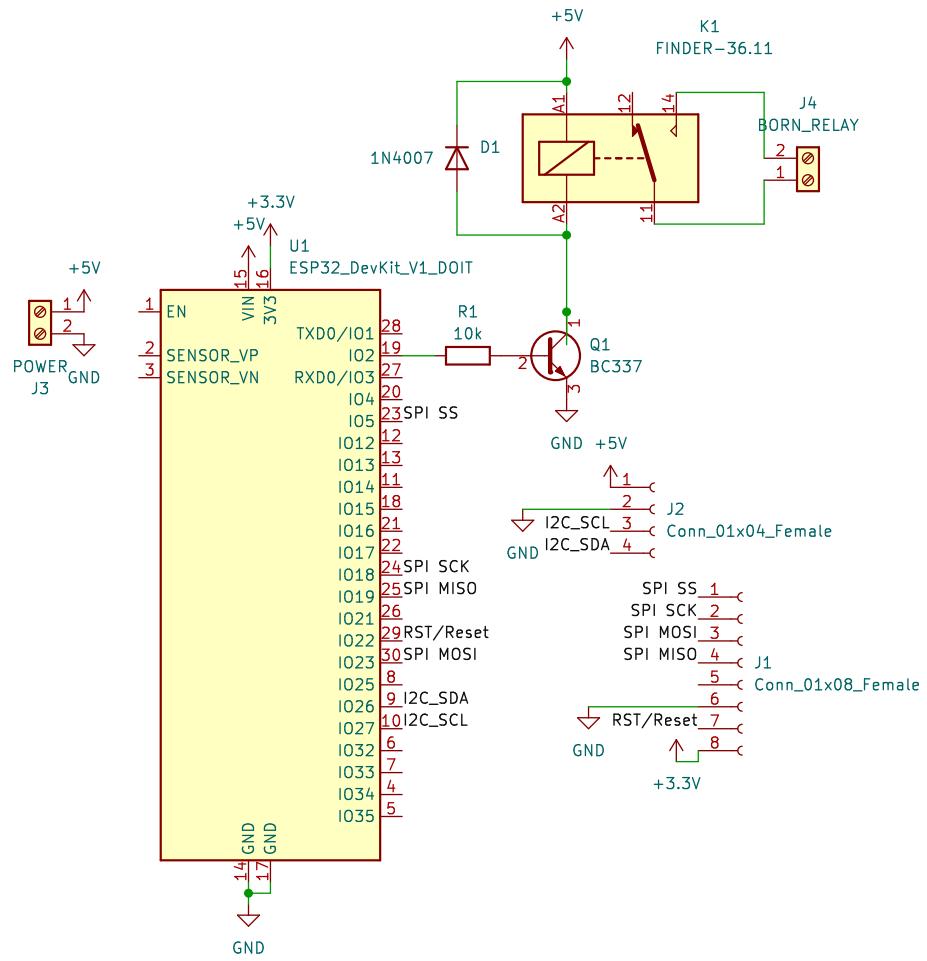
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2. Diagrama en Bloques



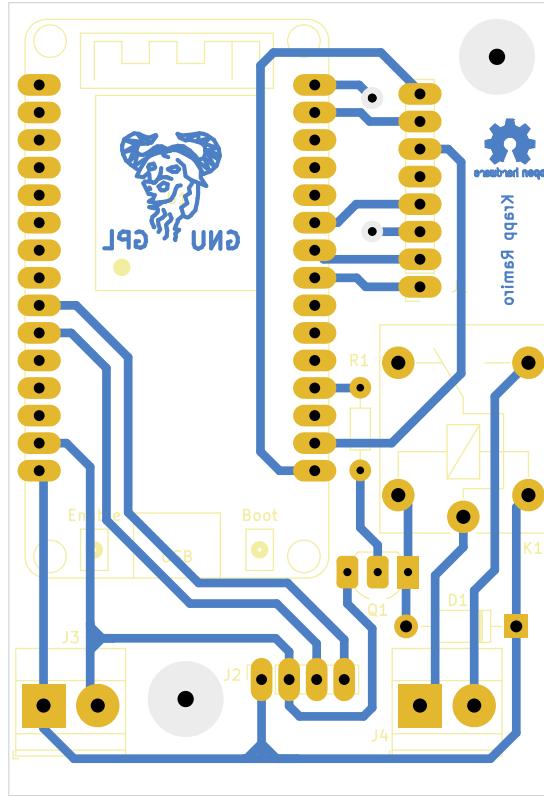
3. Diagrama Esquematico

Este es el diagrama esquematico del proyecto



4. PCB

Este es el PCB del proyecto



5. Partes del proyecto

5.1. DOIT ESP32 DevKit v1

El kit de desarrollo DOIT ESP32 DevKit v1 es una de las placas de desarrollo creadas por DOIT. Esta basada en el microcontrolador ESP32, que en un mismo chip tiene soporte para WiFi, Bluetooth, Ethernet y Low-Power

5.1.1. Características Técnicas

- Microcontrolador: Tensilica 32-bit Single/Dual-core CPU Xtensa LX6
- Tensión de operación: 3.3V
- Tensión de alimentación: 7-12V
- Pines I/O digitales (DIO): 25
- Pines analógicos de Entrada (ADC): 6
- Pines analógicos de Salida (DAC): 2
- UARTs: 3
- SPIs: 2
- I2Cs: 3
- Memoria Flash: 4 MB
- SRAM: 520 KB
- Velocidad de clock: 240 Mhz
- Wi-Fi: IEEE 802.11 b/g/n/e/i, con las siguientes características:
 - Switch TR, Balun, LNA, Amplificador de potencia y antena integrados
 - Autenticación WEP, WPA/WPA2, con la opcion de tambien acceder a redes abiertas.

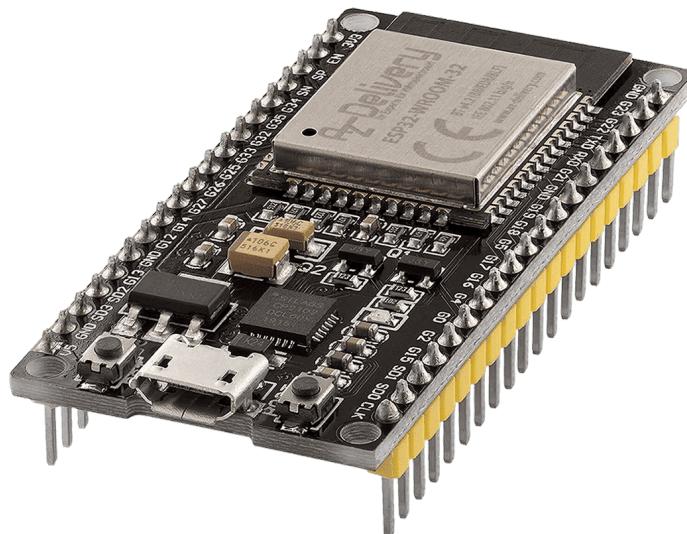


Figura 1: La placa DOIT DevKit ESP32

5.1.2. Pinout

ESP32 DEVKIT V1 – DOIT

version with 30 GPIOs

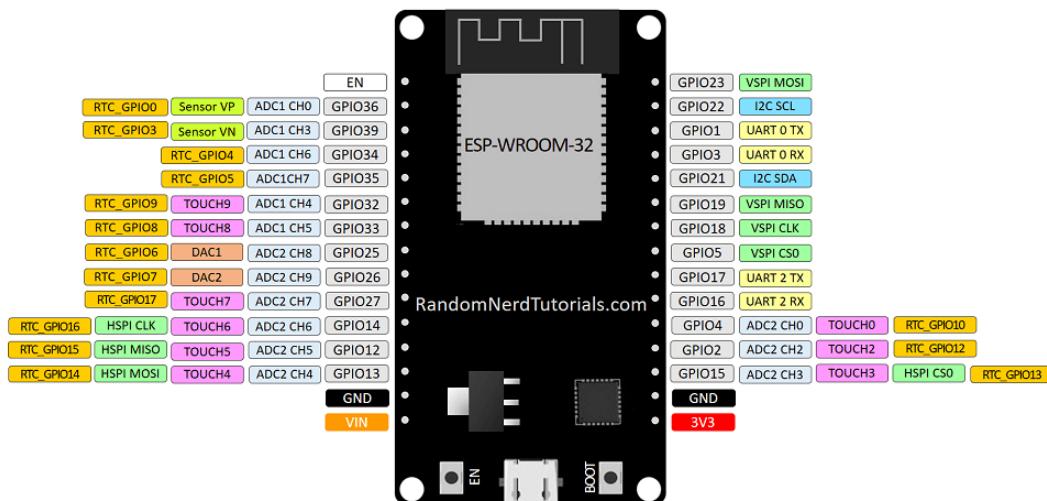
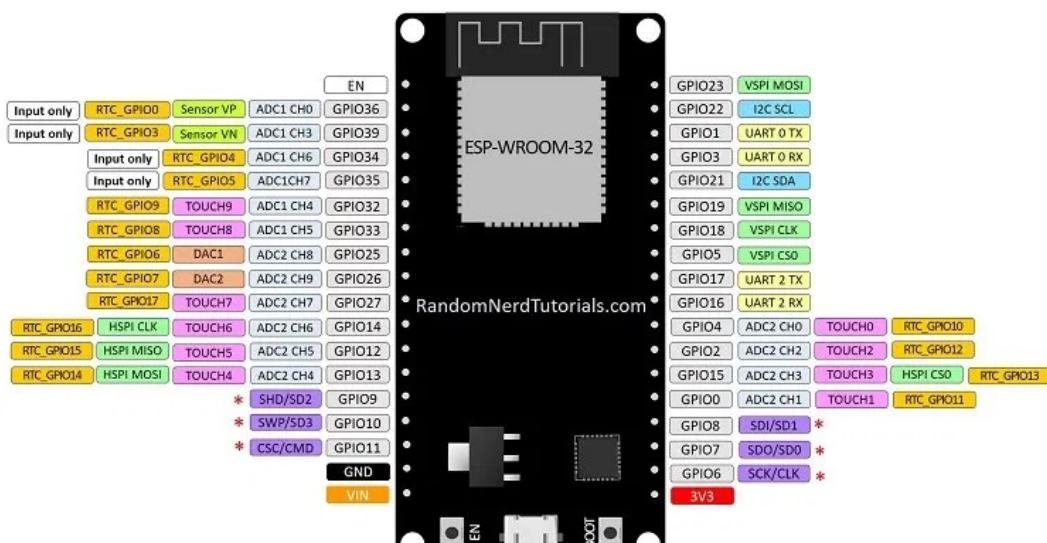


Figura 2: Pinout de la placa de desarrollo DOIT DevKit V1 ESP32 de 30 pines

ESP32 DEVKIT V1 – DOIT

version with 36 GPIOs



* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.

Figura 3: Pinout de la placa de desarrollo DOIT DevKit V1 ESP32 de 36 pines, cabe recordar que los pines GPIO 6-11 estan reservados al sistema SPI integrado, y su uso no es recomendado

5.1.3. Usabilidad de pines

El ESP32 cuenta con una multitud de pines, pero no todos pueden ser usados libremente, esto es explicado en la tabla 1 que muestra que pines pueden ser utilizados y cuales no, dependiendo de las circunstancias.

A la hora de ser usados en el código C++ del framework Arduino, simplemente se refieren por el número

GPIO	Input	Output	Notes
0	pulled up	OK	Hace output de señal PWM al arranque
1	TX pin	OK	debug output al arranque
2	OK	OK	Conectado al LED_ONBOARD
3	OK	RX pin	En estado HIGH al arranque
4	OK	OK	
5	OK	OK	Hace output de señal PWM al arranque
6	x	x	Conectado al flash SPI integrado
7	x	x	Conectado al flash SPI integrado
8	x	x	Conectado al flash SPI integrado
9	x	x	Conectado al flash SPI integrado
10	x	x	Conectado al flash SPI integrado
11	x	x	Conectado al flash SPI integrado
12	OK	OK	El arranque falla si está pulleado en HIGH
13	OK	OK	
14	OK	OK	Hace output de señal PWM al arranque
15	OK	OK	Hace output de señal PWM al arranque
16	OK	OK	
17	OK	OK	
18	OK	OK	
19	OK	OK	
21	OK	OK	
22	OK	OK	
23	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	
32	OK	OK	
33	OK	OK	
34	OK	x	Solamente de input
35	OK	x	Solamente de input
36	OK	x	Solamente de input
39	OK	x	Solamente de input

Cuadro 1: Una tabla con las funciones de cada pin de la placa de desarrollo DOIT DevKit v1 ESP32

5.2. Sistema RFID

Segun Wikipedia[13]:

“RFID o identificación por radiofrecuencia (del inglés Radio Frequency Identification) es un sistema de almacenamiento y recuperación de datos remotos que usa dispositivos denominados etiquetas, tarjetas o transpondedores RFID.

El propósito fundamental de la tecnología RFID es transmitir la identidad de un objeto (similar a un número de serie único) mediante ondas de radio. Las tecnologías RFID se agrupan dentro de las denominadas Auto ID (automatic identification, o identificación automática).

Las etiquetas RFID (RFID tag en inglés) son unos dispositivos pequeños, similares a una pegatina, que pueden ser adheridas o incorporadas a un producto, un animal o una persona. Contienen antenas para permitirles recibir y responder a peticiones por radiofrecuencia desde un emisor-receptor RFID. Las etiquetas pasivas no necesitan alimentación eléctrica interna, mientras que las activas sí lo requieren.

Una de las ventajas del uso de radiofrecuencia (en lugar, por ejemplo, de infrarrojos) es que no se requiere visión directa entre emisor y receptor”



(a) Un llavero RFID



(b) Una tarjeta RFID

Figura 4: Distintos tags RFID

5.2.1. Pinout del dispositivo

pin SDA — Este pin se utiliza de forma distinta dependiendo del protocolo de comunicación utilizado.

- En I2C, se usa como el pin SDA.
- En UART, se usa como pin RX.
- En SPI, se usa como el pin SS

pin SCK — El pin SCK se usa para mantener el sincronismo con una señal de reloj

pin MOSI — El pin MOSI sirve para hacer una transmisión Master Out - Slave In

pin MISO — El pin MISO sirve para hacer una transmisión Master In - Slave Out

pin IRQ — Se usa para las interrupciones

GND — Sirve para mantener la referencia con Masa

RST — Este pin sirve para resetear o desactivar el circuito integrado

VCC — Pin de alimentación **3.3v**

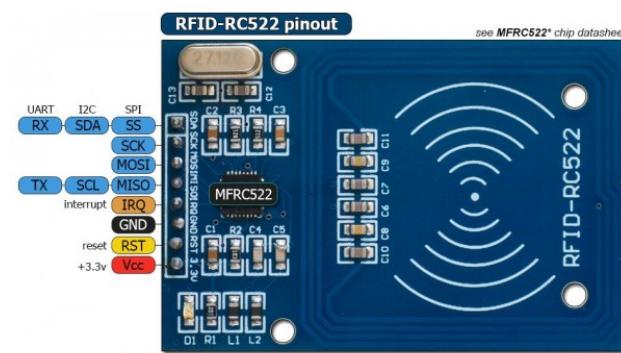
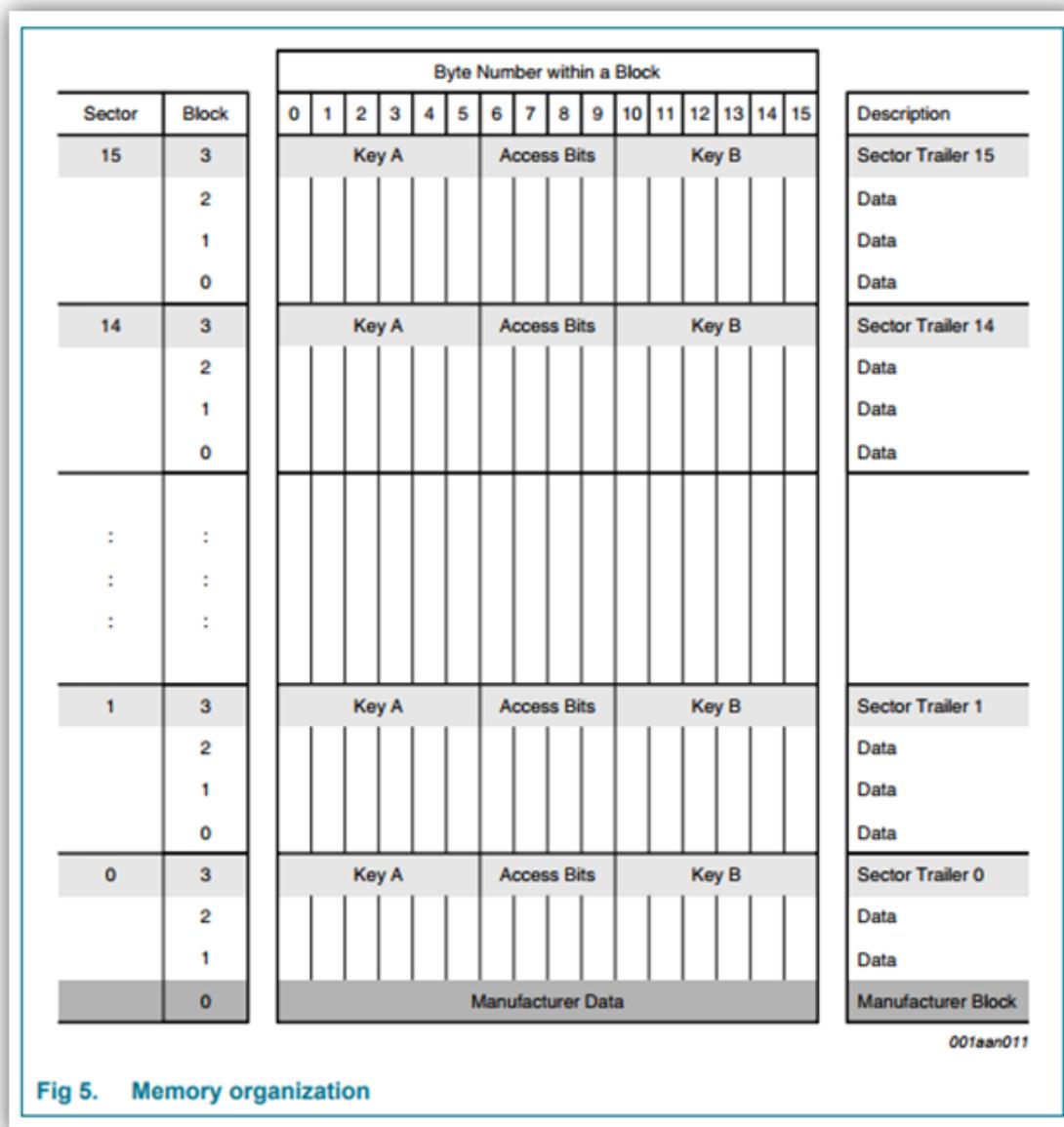


Figura 5: El pinout del lector RFID-RC522. Se puede notar como este dispositivo está adaptado para funcionar con 3 protocolos distintos, comunicación por UART, comunicación por I2C y comunicación por SPI

5.2.2. Mapeo de Memoria del tag RFID

La identificación se realiza con unos llaveros o unas tarjetas, que tienen este mapeo de memoria:

Tenemos 1k de memoria adentro de este chip, la cual esta organizada de la siguiente manera: Hay 16 sectores de 4 bloques, y cada bloque contiene 16 bytes.



5.3. Pantalla OLED SSD1306

Las pantallas OLED se tratan de pantallas que utilizan diodos LED orgánicos capaces de consumir muy poca energía. Estas pantallas son muy delgadas, se comunican por I2C o SPI y producen una imagen más brillante y nítida que los típicos display LCD.

OLED son las siglas en inglés de Organic Light-Emitting Diode que traducido al español sería diodo orgánico de emisión de luz.



Las pantallas OLED están compuestas por láminas de materiales orgánicos como el carbón (por eso el nombre de diodo orgánico). Estas láminas emiten luz cuando se les aplica electricidad entre ellas.

Una de las ventajas de las pantallas OLED con respecto a pantalla LCD es que no requieren de una luz de fondo ni de filtros. Esto hace que las pantallas OLED sean más eficientes en términos de energía, más fáciles de fabricar y mucho más finas. A parte pueden ser flexibles y transparentes.

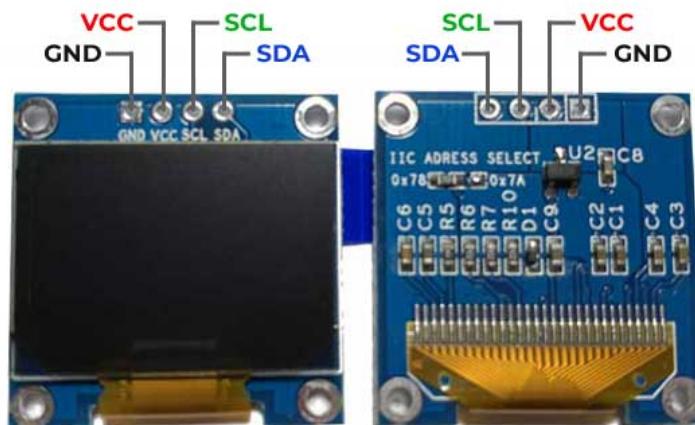


Figura 6: Pinout del dispositivo

6. Gabinete diseñado en 3D

El diseño 3D fue hecho en Tinkercad, el cual se puede encontrar aquí <https://www.tinkercad.com/things/agi002oQ6x5>

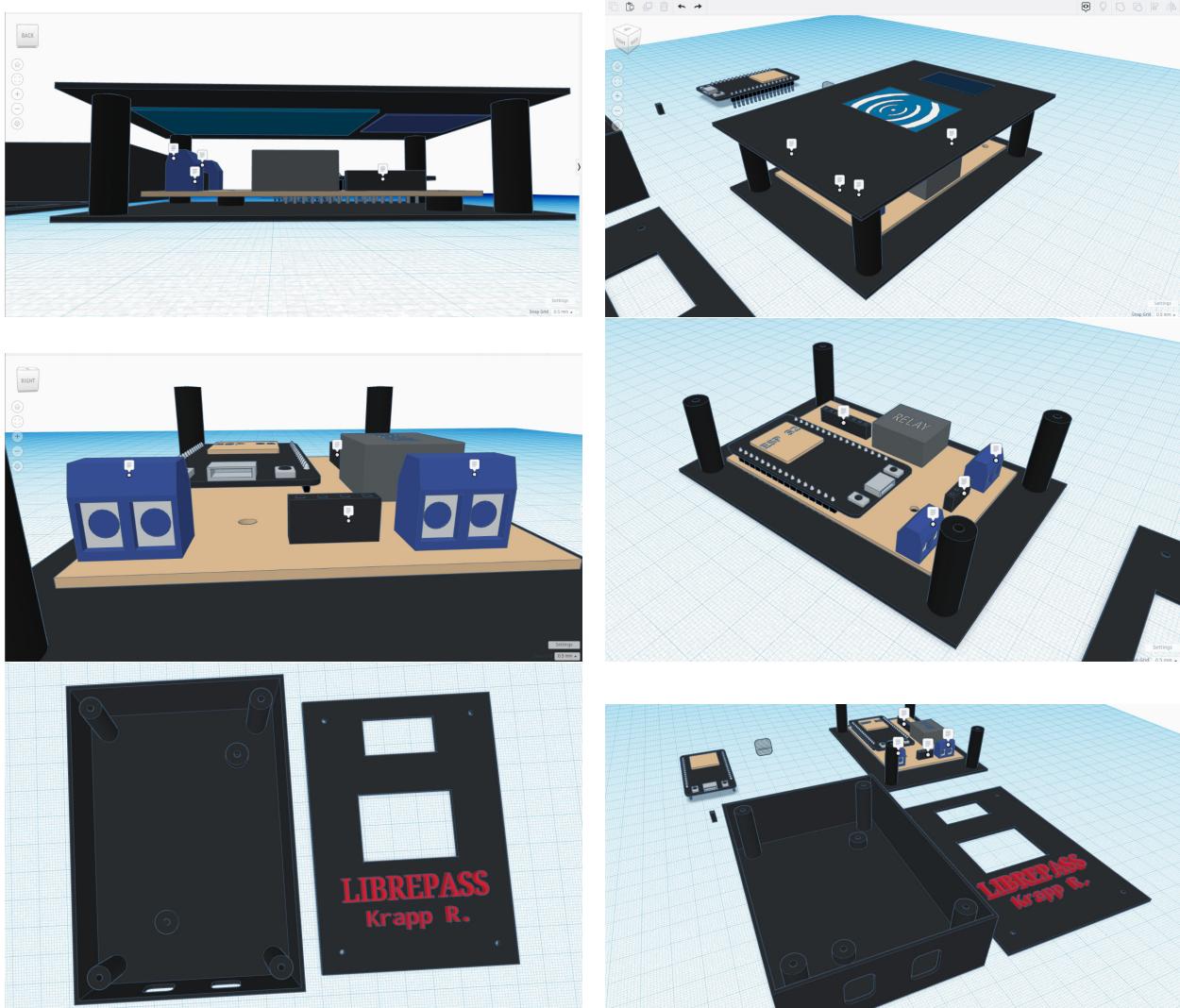


Figura 7: Capturas de pantalla del diseño en 3D

7. Frameworks

Un framework es un esquema o marco de trabajo que ofrece una estructura base para elaborar un proyecto con objetivos específicos, una especie de plantilla que sirve como punto de partida para la organización y desarrollo de software. Utilizar frameworks puede simplificar (y mucho) una tarea o proceso.

Generalmente, los frameworks son usados por programadores porque permiten acelerar el trabajo y favorecer que este sea colaborativo, reducir errores y obtener un resultado de más calidad

Un framework sirve para acometer un proyecto en menos tiempo, y en el sector de la programación, con un código más limpio y consistente, de manera rápida y eficaz. El framework ofrece una estructura base que los programadores pueden complementar o modificar según sus objetivos.

7.1. Arduino

El framework Arduino, que provee una amplia variedad de clases, métodos y funciones útiles para el desarrollo en sistemas embebidos. Este framework se implementó a través de platformIO, una extensión de Visual Studio Code. En este proyecto, todo el desarrollo del ESP32 fue hecho mediante este framework.

7.1.1. Razones por las que usé Arduino

Use Arduino porque es un framework con el que ya estoy familiarizado, además que, gracias a la existencia de bibliotecas como Wifi.h, HTTP-Client.h, MFRC522, Wire.h, etc..., gran cantidad del desarrollo ya está hecho, y lo único que tengo que hacer es construir alrededor de esas bibliotecas.

Además, el uso de Arduino se está volviendo un estandar en la industria, ya que permite que todos los programadores sigan una misma forma de desarrollo, cosa que resultaba complicada en C++, el cual es un lenguaje conocido por permitir múltiples formas de desarrollar una misma cosa, o dicho por el propio creador, "múltiples formas de pegarse un tiro en el pie"



7.2. Flask



Flask es un microwebframework usado para, como indica el nombre, crear aplicaciones webs. Este, al ser modular y escalable, tiene la posibilidad de permitir el agregado de ORM's (Object Relational Manager), routers, renderizador de templates, sistema de logins, forms, etc... En este proyecto, Flask se usó para el desarrollo del backend, y con la ayuda del renderizador de templates Jinja2 y Nunjucks, se templatizó el frontend.

7.2.1. Razones por las que usé Flask

Use Flask porque me lo aconsejó un programador conocido, ya que aprender a usarlo me iba a abrir las puertas al desarrollo backend con otros frameworks como Django, los cuales tienen un gran uso en el mercado laboral. Además, Flask me permite desarrollar mi programa de forma sencilla gracias a su alta modularidad

7.3. Bootstrap

Bootstrap es el framework CSS más popular para desarrollar aplicaciones responsivas y aptas para dispositivos móviles. En este proyecto, se usó la versión 5 de Bootstrap para el desarrollo del frontend.

Este framework cuenta con múltiples clases, las cuales se usan para crear las páginas webs al gusto del programador.

7.3.1. Razones por las que usé Bootstrap

La principal razón por la que use bootstrap es que ya tiene componentes creados previamente, lo cual libera carga al desarrollador, y permite enfocarse en partes más esenciales del proyecto.



7.4. Amazon Web Services (AWS)

Amazon Web Services (AWS) es la plataforma cloud más adoptada en el mundo, con más de 200 servicios distintos. En este proyecto, AWS se usó para la implementación de la infraestructura de la webapp. Se usarán los servicios de Relational Database Service (RDS) y Elastic Beanstalk.



7.4.1. Razones por las que usé AWS

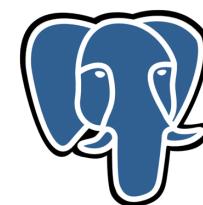
Hay dos razones principales por las que usé AWS: La primera razón es porque hostear un servidor físico requiere mucho tiempo y esfuerzo, además de que es **muy caro**, mientras que AWS es completamente escalable, y en tamaños chicos es realmente económico. La segunda es que en la industria ya no se está acostumbrando tanto a usar servidores físicos, sino que se están realizando muchas migraciones a servicios en la nube gracias a la comodidad que proveen.

Y en el ámbito de la comodidad, su servicio ElasticBeanstalk facilita mucho el desarrollo de aplicaciones web, ya que en ningún momento tuve que configurar más que el PYTHONPATH y el WSGIPATH, y es simplemente un archivo de texto, además de acoplarle la base de datos PostgreSQL

7.5. PostgreSQL

PostgreSQL (también conocido como Postgres por la comunidad) es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto.

Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre o apoyados por organizaciones comerciales. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).



PostgreSQL

7.5.1. Razones por las que usé PostgreSQL

La principal razón por la que uso PostgreSQL es que es un estandar en la industria moderna a la hora de hacer bases de datos SQL.

8. Protocolos de comunicación

8.1. El protocolo de comunicación SPI

El protocolo Serial Peripheral Interface es un protocolo de comunicación creado por Motorola, anunciado en el año 1979. El mismo se divide en 4 líneas de comunicación, cada una con una función específica (por favor, ver figura 8) con:

- Una señal de clock llamada SCLK, enviada desde el bus master a todos los slaves. Todas las señales del protocolo van asincrónicas a esta señal de clock
- Una señal de selección de slave llamada SS_n, usada para seleccionar con qué slave se está comunicando el master
- Una línea de datos desde master hacia slave, llamada MOSI (Master Out Slave In)
- Una línea de datos desde slave hacia master, llamada MISO (Master In Slave OUT)

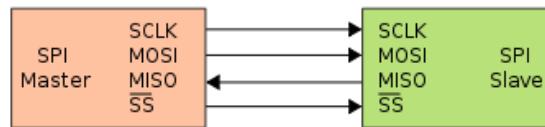
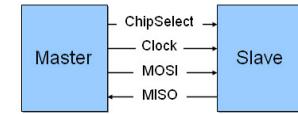


Figura 8: SPI master conectado a un único slave.

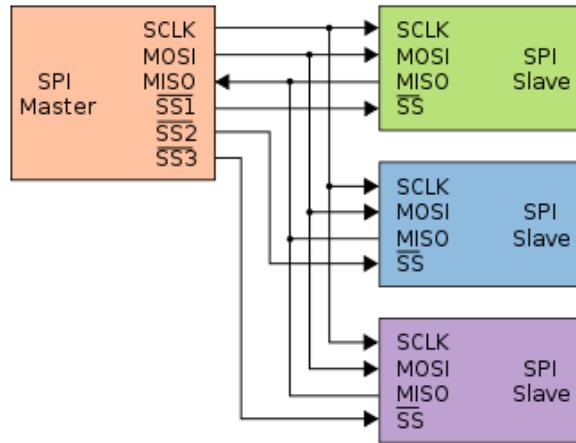


Figura 9: SPI master conectado a múltiples slaves.

SPI es un protocolo de comunicación single-master, esto significa que un dispositivo central (normalmente un microcontrolador) es el encargado de iniciar todas las comunicaciones con los slaves.

Cuando el master SPI desea enviar o recibir información de un slave, selecciona el slave seteando en LOW la linea SS correspondiente, y activa la señal de clock a una frecuencia usable por el master y el slave. A partir de ese momento, el master envía la información por el canal MOSI mientras lee la información que hay en el canal MISO

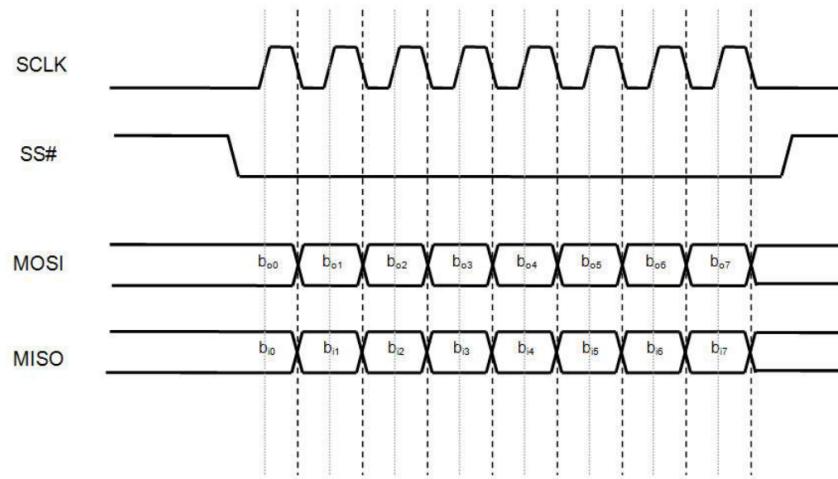


Figura 10: El timing de una comunicación SPI. En este ejemplo, La transmisión de datos por los canales MOSI y MISO es ejecutada por cada flanco descendente en la señal de clock en SCLK. En cambio, la lectura de datos es ejecutada por cada flanco ascendente. Esto se puede cambiar modificando el SPI mode

Como se menciona en la figura 10, hay 4 modos SPI, que van del 0 al 3. Los modos SPI definen en qué flanco se activa la linea MOSI, MISO, y el estado (LOW o HIGH) de inactividad (idle) del canal SCLK. Cada modo esta definido por un par de parámetros llamados clock polarity (polaridad de clock) (CPOL), y clock phase (fase de clock) (CPHA)

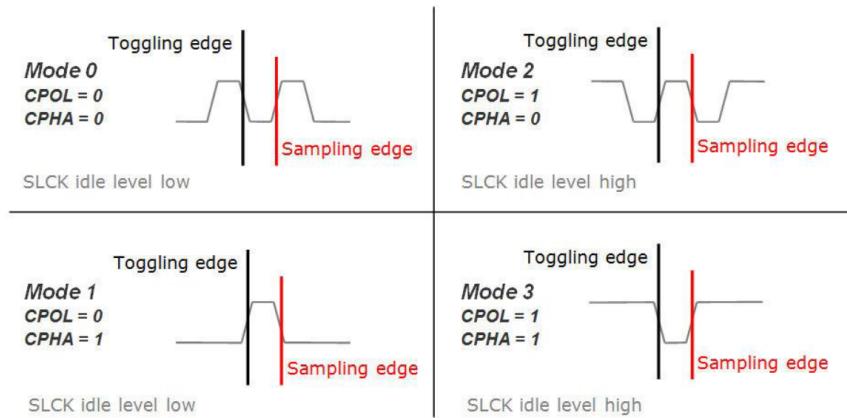


Figura 11: Los modos SPI son definidos con los parámetros CPOL (clock polarity) y CPHA (clock phase), que definen 3 parámetros: El flanco usado para envío de datos, el flanco usado para recepción de datos, y el estado de inactividad (idle) de SCLK

Una conexión SPI master/slave tiene que usar el mismo set de parámetros explicados en la figura 11 para poder efectuar una comunicación. Si de todas formas se desea que múltiples slaves tengan configuraciones distintas, el master deberá reconfigurarse cada vez que se desee comunicar con cada dispositivo.

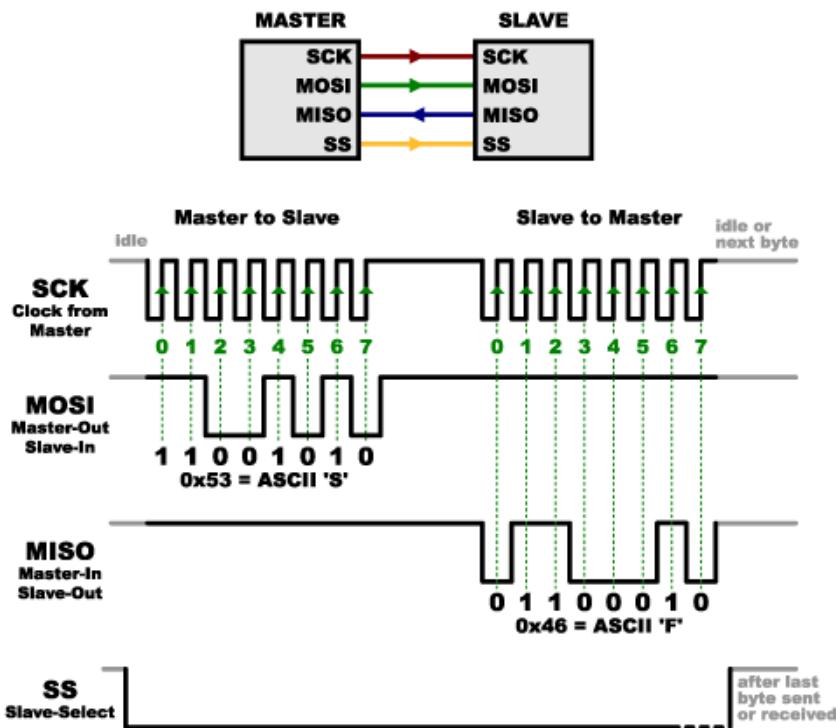


Figura 12: Grafico de comunicacion SPI

8.1.1. Ventajas

Segun Wikipedia[14]:

- Comunicación Full Duplex
- Mayor velocidad de transmisión que con I²C o SMBus
- Protocolo flexible en que se puede tener un control absoluto sobre los bits transmitidos
- No está limitado a la transferencia de bloques de 8 bits
- Elección del tamaño de la trama de bits, de su significado y propósito
- Su implementación en hardware es extremadamente simple
- Consume menos energía que I²C o que SMBus debido que posee menos circuitos (incluyendo las resistencias pull-up) y estos son más simples
- No es necesario arbitraje o mecanismo de respuesta ante fallos
- Los dispositivos clientes usan el reloj que envía el servidor, no necesitan por tanto su propio reloj
- No es obligatorio implementar un transceptor (emisor y receptor), un dispositivo conectado puede configurarse para que solo envíe, sólo reciba o ambas cosas a la vez
- Usa mucho menos terminales en cada chip/conector que una interfaz paralelo equivalente
- Como mucho una única señal específica para cada cliente (señal SS), las demás señales pueden ser compartidas

8.1.2. Desventajas

- Consumo más pines de cada chip que I²C, incluso en la variante de 3 hilos
- El direccionamiento se hace mediante líneas específicas (señalización fuera de banda) a diferencia de lo que ocurre en I²C que se selecciona cada chip mediante una dirección de 7 bits que se envía por las mismas líneas del bus
- No hay control de flujo por hardware
- No hay señal de asentimiento. El servidor podría estar enviando información sin que estuviese conectado ningún cliente y no se daría cuenta de nada
- No permite fácilmente tener varios servidores conectados al bus
- Sólo funciona en las distancias cortas a diferencia de, por ejemplo, RS-232, RS-485, o Bus CAN



8.2. Protocolo I2C

El protocolo I2C es un protocolo de comunicacion entre circuitos integrados, en el cual se definen dispositivos a funcionar como maestros o *master*, y dispositivos a funcionar como esclavos o *slave*.

Lo que se hace es establecer comunicacion entre los dispositivos usando dos lineas:

- **SCL (Serial CLock):** Es la linea que transmite la señal de sincronía.
Eléctricamente se trata de una señal a colector o drenador abierto. En un dispositivo esclavo se trata de una entrada, mientras que en un dispositivo maestro es una salida.
El dispositivo maestro genera la señal de sincronía, necesaria para mantener la comunicación entre los dispositivos.
- **SDA (Serial DAta):** Es la linea que transmite los datos de forma semi-bidireccional.
Eléctricamente se trata de una señal a colector o drenador abierto. Es gobernada por el emisor, sea éste un maestro o un esclavo.
Sobre esta linea se montan los datos a transmitir entre los dispositivos.



Figura 13: Logo de I2C

9. Bibliografia

- [1] RFID 4u. *RFID Regulations*. URL: <https://rfid4u.com/rfid-regulations/>.
 - [2] Code Academy. *What is REST?* URL: <https://www.codecademy.com/article/what-is-rest>.
 - [3] Amazon. *AWS IoT Core*. URL: <https://aws.amazon.com/iot-core/>.
 - [4] Amazon. *Getting started with AWS IoT Core*. URL: <https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs.html>.
 - [5] Amazon. *What is cloud computing?* URL: <https://aws.amazon.com/what-is-cloud-computing/>.
 - [6] Corelis. *SPI Tutorial*. URL: <https://www.corelis.com/education/tutorials/spi-tutorial/>.
 - [7] Zerynth docs. *DOIT Esp32 DevKit v1 reference*. URL: https://testzdoc.zerynth.com/reference/boards/doit_esp32/docs/.
 - [8] Fireship. *Top 50+ AWS Services Explained in 10 Minutes*. URL: <https://www.youtube.com/watch?v=J1bIYCM48to>.
 - [9] Arduino Foundation. *A Brief Introduction to the Serial Peripheral Interface*. URL: <https://www.arduino.cc/en/reference/SPI>.
 - [10] Arduino Foundation. *Language Reference*. URL: <https://www.arduino.cc/reference/en/>.
 - [11] Free Software Foundation. *What is Free Software*. URL: <https://www.fsf.org/about/what-is-free-software>.
 - [12] Wikimedia Foundation. *Radio-frequency identification*. URL: https://en.wikipedia.org/wiki/Radio-frequency_identification.
 - [13] Wikimedia Foundation. *RFID*. URL: <https://es.wikipedia.org/wiki/RFID>.
 - [14] Wikimedia Foundation. *Serial Peripheral Interface*. URL: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface.
 - [15] Mike Grusin. *Serial Peripheral Interface (SPI)*. URL: <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi>.
 - [16] ESP32 IO. *ESP32 RFID/NFC*. URL: <https://esp32io.com/tutorials/esp32-rfid-nfc>.
 - [17] Fernando Koyanagi. *ESP32 With RFID: Access Control*. URL: <https://www.instructables.com/ESP32-With-RFID-Access-Control/>.
 - [18] Exostiv Labs. *Introduction to I2C and SPI Protocols*. URL: <https://www.exostivlabs.com/files/documents/Introduction-to-I2C-and-SPI-Protocols.pdf?/article/aa-00255/22/introduction-to-spi-and-ic-protocols.html>.
 - [19] Rus Shuler. *How Does the Internet Work?* URL: <https://web.stanford.edu/class/msande91si/www-spr04/readings/week1/InternetWhitepaper.htm>.
 - [20] Arduino Get Started. *Arduino RFID/NFC*. URL: <https://arduinogetstarted.com/tutorials/arduino-rfid-nfc>.
 - [21] Arduino Get Started. *How to connect multiple spi sensors/devices with Arduino?* URL: <https://arduinogetstarted.com/faq/how-to-connect-multiple-spi-sensors-devices-with-arduino>.
 - [22] GNU Operating System. *¿Qué es el Software Libre?* URL: <https://www.gnu.org/philosophy/free-sw.es.html>.
 - [23] Massachusetts Institute of Technology. *The MIT License*. URL: <https://mit-license.org/>.
 - [24] techtutorialsx. *ESP32: HTTP GET Requests*. URL: <https://techtutorialsx.com/2017/05/19/esp32-http-get-requests/>.
 - [25] Random Nerd Tutorials. *DHT Sensor Web Server*. URL: <https://randomnerdtutorials.com/esp32-dht11-dht22-temperature-humidity-web-server-arduino-ide/>.
-



- [26] Random Nerd Tutorials. *ESP32 Pinout Reference: Which GPIO pins should you use?* URL: <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>.
- [27] Random Nerd Tutorials. *ESP32 Web Server using Server-Sent Events (Update Sensor Readings Automatically)*. URL: <https://randomnerdtutorials.com/esp32-web-server-sent-events-sse/>.
- [28] Random Nerd Tutorials. *Getting Started with the ESP32 Development Board*. URL: <https://randomnerdtutorials.com/getting-started-with-esp32/>.
- [29] Random Nerd Tutorials. *How to use ESP32 Dual Core with Arduino IDE*. URL: <https://randomnerdtutorials.com/esp32-dual-core-arduino-ide/>.
- [30] Random Nerd Tutorials. *Security Access using MFRC522 RFID Reader with Arduino*. URL: <https://randomnerdtutorials.com/security-access-using-mfrc522-rfid-reader-with-arduino/>.
- [31] Tech TutorialsX. *ESP32 Arduino HTTP Server: Template processing with multiple placeholders*. URL: <https://techtutorialsx.com/2018/07/23/esp32-arduino-http-server-template-processing-with-multiple-placeholders/>.
- [32] Luis del Valle Hernández. *SSD1306 pantalla OLED con Arduino y ESP8266 I2C*. URL: <https://programarfacil.com/blog/arduino-blog/ssd1306-pantalla-oled-con-arduino/>.
- [33] W3Schools. *AJAX - Server Response*. URL: https://www.w3schools.com/XML/ajax_xmlhttprequest_response.asp.